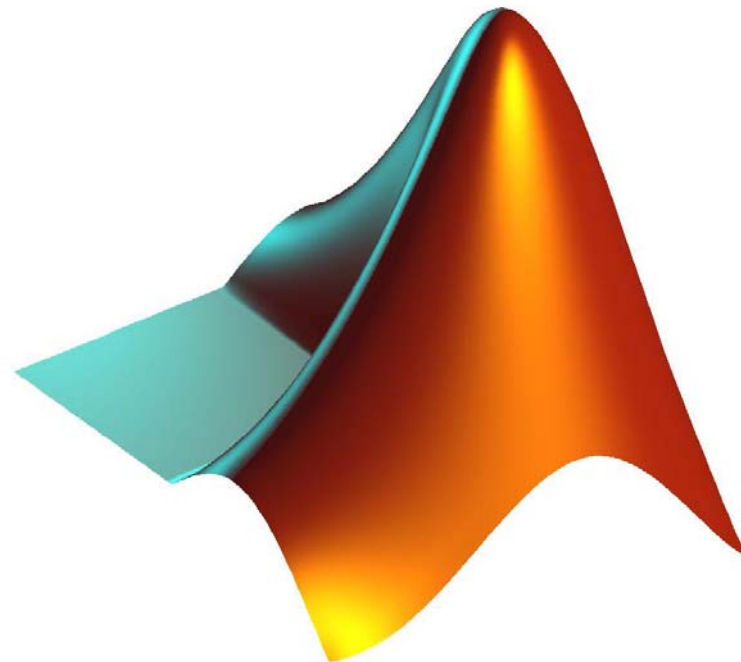
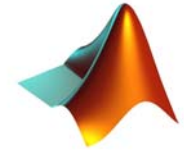


Introduzione all'ambiente MATLAB

Parte II



Programmazione MATLAB



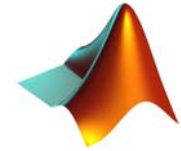
MATLAB non è un vero e proprio linguaggio di programmazione, ma permette comunque di realizzare programmi utilizzando le classiche strutture di programmazione come i cicli, i flussi di controllo e la gestione input/output.

Due tipi di programmi, noti come “m-file”,
(es. sort.m)

- script**

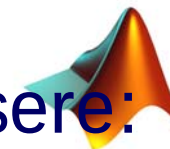
- function**

Programmazione MATLAB



- ❑ Per **creare un m-file** si seleziona **New/m-file** dal **menu File**.
- ❑ **Script e funzioni** vengono quindi inseriti mediante l'editor di MATLAB. Si **salva poi il file** con il **suffisso .m**
- N.B.** Assicurarsi di aver selezionato dalla *current directory* il percorso (path) relativo alla directory (cartella) di lavoro contenente l'm-file.

Gli errori in cui ci si può imbattere possono essere:



Errori di sintassi

nell'uso del linguaggio di programmazione. **LEGGERE il messaggio visualizzato in rosso nella Command Window** aiuta a capire che tipo di errore è stato fatto.

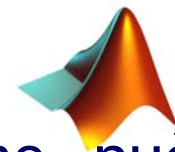
Interruzione dell'esecuzione

Sono causati da overflow, divisioni per zero,... Istruzioni illegali

Errori nel risultato

Sono causati da errori o nell'algoritmo o nella sua implementazione in MATLAB. Si correggono facendo il **“debug”** del programma, cioè seguendo il programma istruzione per istruzione, prevedendo il risultato dell'istruzione e controllando, tramite scrittura su video, i valori interessati nell'istruzione (basta eliminare il “;” alla fine dell'istruzione).

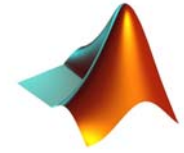
Script



Uno **SCRIPT** è una lista di comandi MATLAB che può eventualmente richiamare funzioni MATLAB built-in o create utilizzando altri m-file.

- Non richiede input
- Non fornisce output espliciti
- Tutte le variabili usate sono disponibili nel workspace
- Simile ad un programma principale
- .m file deve essere disponibile nel proprio path corrente

N.B. Per **eseguire** i comandi contenuti nello script (cioè richiedere a MATLAB che il file venga **interpretato**) si richiama semplicemente il suo nome dalla Command Window (senza estensione .m).



Esempio:

scrivere uno script che calcola la media fra 4 variabili

- Scrivere nell'editor lo script

```
a=10;
```

```
b=20;
```

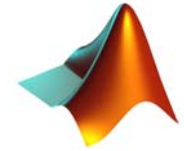
```
c=30;
```

```
d=40;
```

```
media=(a+b+c+d)/4;
```

- Salvare l'm-file come ***prova.m***
- Eseguire lo script digitando nella Command Window ***prova***

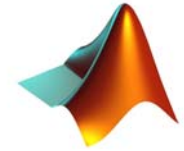
Function



Un m-file che contiene una funzione MATLAB si **identifica nella prima riga con la parola function seguita da eventuali parametri di input e di output.**

Le variabili al suo interno sono viste solo localmente dalla funzione stessa e non dall'eventuale m-file chiamante o dall'ambiente MATLAB che la richiama.

Function



Sintassi della function:

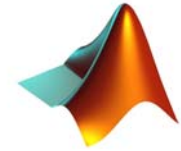
function y = < *functionname* > (argomenti input)
y variabile in output

Quando invece le variabili output sono più di una, la forma precedente si modifica nel modo seguente:

function [y,z] = < *functionname* > (argomenti input)

Il nome dell'm-file deve essere il nome dato alla funzione, eccetto chiaramente l'estensione .m

Function - esempio



Variabili

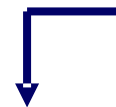
output



Nome function



Parametri input



```
function [mean,stdev] = stat(x)
```

```
% esempio di function.
```

```
n = length(x);
```

```
mean = sum(x) / n;
```

```
stdev = sqrt(sum((x - mean).^2)/n);
```

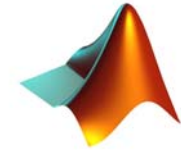
Definisce una nuova funzione STAT che calcola la media e la deviazione standard di un vettore x. Le variabili nel corpo della funzione sono tutte variabili locali.

```
[m,s]=stat (1:5)
```



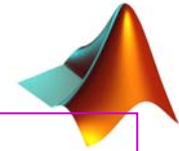
Richiamo della function
da command window o da
uno script o da altra function

Funzioni secondarie



- Uno stesso file .m può contenere più function:
 - la function principale sta all'inizio e dà il nome al file;
 - Seguono le function secondarie.
 - Soltanto la function principale può essere richiamata da altre function esterne al file o dal prompt.
- **Esempio:**
La funzione **polyGeom** mostra l'uso di una funzione principale che richiama due funzioni secondarie per calcolare area e perimetro di un poligono regolare con numero di lati e lunghezza assegnata.

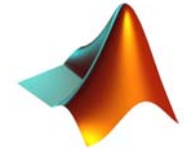
Esempio (in file polyGeom.m)



```
function [a,p] = polyGeom(s,n)
% polyGeom Compute area and perimeter of a regular
% polygon
% Input:  s = length of one side of the polygon
%         n = number of sides of the polygon
%
% Output: a = total area of the polygon
%         p = total perimeter of the polygon
r = s/(2*tan(pi/n)); % "radius" of the polygon
a = area(r,n);
p = perimeter(r,n);
% ===== subfunction "area"
function a = area(r,n)
% area Compute area of an n-sided polygon of radius r
a = n*r^2*sin(pi/n);
% ===== subfunction "perimeter"
function p = perimeter(r,n)
% perimeter Compute perimeter of an n-sided polygon of
% radius r
p = n*2*r*tan(pi/n);
```

Le funzioni **area** e **perimeter** non possono essere richiamate dalla command window

Loading e Saving



Load e Save in binario e ASCII

ASCII più facile da condividere con altre applicazioni

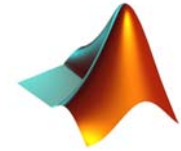
save 'c:\temp\my_matrix.txt' A -ascii -double

load 'c:\temp\my_matrix.txt'

- my_matrix diventa una variabile
- Carica righe e colonne di numeri

Esempio

```
» x = [0:10]`;  
» y = [x x.^2 x.^3 x.^4]
```

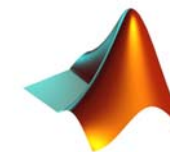


y =

0	0	0	0
1	1	1	1
2	4	8	16
3	9	27	81
4	16	64	256
5	25	125	625
6	36	216	1296
7	49	343	2401
8	64	512	4096
9	81	729	6561
10	100	1000	10000

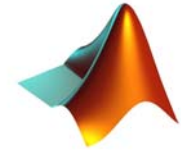
```
» save `c:\temp\y1.txt` y -ascii
```

Contenuto di y1.txt



0.0000000e+000	0.0000000e+000	0.0000000e+000	0.0000000e+000
1.0000000e+000	1.0000000e+000	1.0000000e+000	1.0000000e+000
2.0000000e+000	4.0000000e+000	8.0000000e+000	1.6000000e+001
3.0000000e+000	9.0000000e+000	2.7000000e+001	8.1000000e+001
4.0000000e+000	1.6000000e+001	6.4000000e+001	2.5600000e+002
5.0000000e+000	2.5000000e+001	1.2500000e+002	6.2500000e+002
6.0000000e+000	3.6000000e+001	2.1600000e+002	1.2960000e+003
7.0000000e+000	4.9000000e+001	3.4300000e+002	2.4010000e+003
8.0000000e+000	6.4000000e+001	5.1200000e+002	4.0960000e+003
9.0000000e+000	8.1000000e+001	7.2900000e+002	6.5610000e+003
1.0000000e+001	1.0000000e+002	1.0000000e+003	1.0000000e+004

Input/output



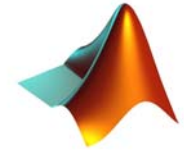
Il comando **input** interrompe l'esecuzione e richiede un valore da tastiera come input. Una volta inserito il valore, l'm-file continua la sua esecuzione.

Il comando **echo** permette di visualizzare a video i comandi durante la loro esecuzione.

Il comando **pause** interrompe l'esecuzione fino a quando non si digita un tasto, mentre il comando **pause(n)** mette in pausa l'esecuzione per n secondi.

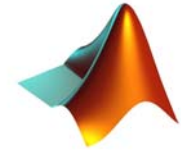
Il comando **keyboard** consente di inserire altri comandi da tastiera durante l'esecuzione di un m-file. Si presenta con **K>>** e attende l'inserimento del comando. Per riprendere la normale esecuzione dell'm-file basterà scrivere **return**.

Flussi di controllo



- **if elseif else end**
- **switch case otherwise end**
- **for end**
- **while end**
- **break**

For *counter* = start:step:final **end**



Il ciclo for permette di eseguire uno o più comandi un dato numero di volte.

Es. si vuole riempire una matrice di dimensione $m \times n$.

```
for i=1:m
    for j=1:n
        A(i,j)=1/(i+j);
    end
end
```

Ricordiamo di mettere un **;** alla fine di ogni istruzione per evitare la visualizzazione dell'output su schermo. Ogni ciclo for ovviamente deve terminare con il proprio end.

While (condizione di entrata) ... end

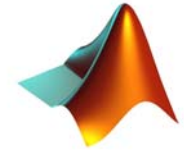


Il ciclo while ripete l'esecuzione finchè la condizione logica risulta vera.

Es. Calcolare il più grande intero n per il quale $n!$ (fattoriale) è un numero ≤ 100 .

Possiamo utilizzare un ciclo while che calcola $n!$ e terminare l'esecuzione quando il valore calcolato è maggiore di 100.

```
n=1;  
prod=1;  
while (prod<=100)  
    n=n+1; prod=prod*n;  
end  
n-1
```

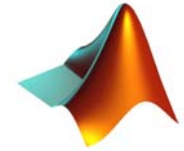


Esempio:

Calcolare la somma di alcuni numeri introdotti da tastiera fino ad ottenere un valore maggiore di 100. Il ciclo si interrompe se si inserisce come numero 0.

```
somma = 0;  
n=1;  
while (n~= 0 ) & (somma<=100)  
    n = input('Inserire un numero');  
    somma = somma + n;  
end
```

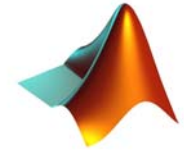
if elseif else end



L'istruzione condizionale **if** può essere accompagnata da **else** o **elseif**. L'istruzione **break** può essere inserita dentro al ciclo per terminare il ciclo.

```
if expression1
    statements1
elseif expression2
    statements2
else
    statements3
end
```

N.B. **else** e **elseif** possono essere omessi



Esempio:

si vuole verificare che il numero inserito sia pari o dispari.

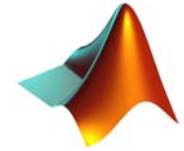
```
n=input ('Inserisci n ')\nif n <= 0, break,end\nif rem(n,2)==0 % funzione resto\n    disp(' n pari')\nelse\n    disp('n dispari')\nend
```

switch case otherwise end



Il comando switch può essere usato nel caso di if multipli.

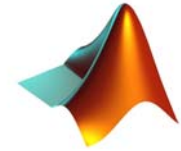
```
switch(relation expression)
  case 1(value)
    block of statements
  case 2(value)
    block of statements
  case 3 (value)
    block of statements
  otherwise
  :
end
```



```
if rem(n,2) < 0
    disp ( 'il numero è negativo e dispari' )
elseif rem(n,2) > 0
    disp( 'il numero è positivo e dispari' )
else
    disp( 'il numero è pari' )
end
```

```
switch rem(n,2)
    case -1
        disp ( 'il numero è negativo e dispari' )
    case 1
        disp( 'il numero è positivo e dispari' )
    otherwise
        disp( 'il numero è pari' )
end
```

Function handle



`funhandle = @function_name`

Ritorna un puntatore alla funzione di nome `function_name`.

- creiamo una funzione `funprova` in `funprova.m`

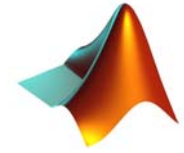
```
function y=funprova(x)
y=cos(x).*sin(x);
```

- creiamo un puntatore alla funzione

```
f = @funprova;
```

- chiamiamo la funzione con i suoi argomenti

```
funprova(-pi,pi) oppure f(-pi,pi)
```



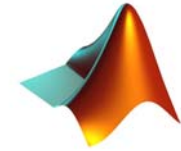
Funzioni di Funzioni

- Una funzione prende un'altra funzione come parametro in input.

passiamo il puntatore a funzione in una chiamata ad un'altra funzione

```
>>f = @funzprova;  
>>x = calcolo(f,-pi,pi);
```

Esempio di una funzione MATLAB



```
function [m,q]=calcolo(fp,vi,vf);
```

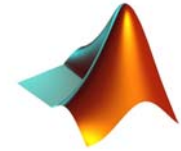
Dice a MATLAB che
questa è una funzione

Lista di argomenti di
output, separati da
virgola

Nome della
funzione

Lista degli argomenti
di input, separati da
virgola

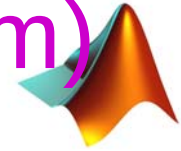
Esempio (file calcolo.m)



```
function [m,q]=calcolo(fp,vi,vf)
% PLOT della funzione fp in [vi,vf], calcolo del minimo
% e dell'area della funzione in [vi,0].

% FPLOTT grafico della funzione puntata da fp nel dominio di x [vi,vf].
fplot(fp,[vi,vf]);
% FMINBND trova il minimo di fp nel dominio (0,vf).
m = fminbnd(fp,0,vf,optimset('Display','off'));
hold on;
plot(m,fp(m),'ro');           % valutazione di funzione dentro al plot
% QUAD trova l'integrale definito di fp nel dominio [vi,vi/2].
q = quad(fp,vi,vi/2);
x=vi:1/100:vi/2;
area(x,fp(x));
title(['Area = ',num2str(q)]);
```

Esempio di uno script (in randscript.m)



```
% script randscript
% Un semplice script per generare un vettore di n numeri random,
% calcolarne la somma e stampare sullo schermo il tempo impiegato
```

```
n = 100000;      % il numero dei random che si vogliono generare
y = zeros(n,1);  % allocazione della memoria per y
fprintf('Simulazione di %d numeri random.....\n\n',n);
```

```
% utilizziamo il ciclo for per creare i numeri random
fprintf('Ciclo for ..... \n');
t=cputime;       % inizializzo il timer
```

```
for i=1:n
    y(i) = rand(1);
end
```

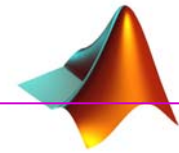
→
oppure

```
z = rand(n,1);
```

```
total = sum(y);
fprintf('Somma di %d numeri random = %f\n',n,total);
% tempo utilizzato da MATLAB per svolgere il calcolo (in secondi)
e=cputime-t;
```

```
fprintf('Tempo impiegato usando il ciclo for = %6.5f
        microsecondi\n\n', (e)*1000);
```

Esempio di una funzione MATLAB

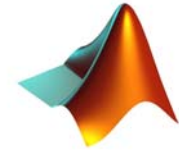


```
function [meanr, stdr, z] = simulate(n);
% Una semplice funzione che restituisce la media e la deviazione
% standard di n numeri random uniformemente distribuiti.
% INPUTS:
%     n: numero dei random generati (intero positivo)
% OUTPUTS:
%     meanr: media degli n numeri random
%     stdr:  deviazione standard degli n numeri random
%     z:     vettore n x 1 dei numeri random generati

% test per verificare se n è un intero positivo
if (rem(n,1)~=0) | n<=0
    error('Input n must be a positive integer');
end
fprintf('Simulazione di %d numeri random.....\n\n',n);
z = rand(n,1);           % genero n numeri random
meanr= mean(z);          % calcolo la media
fprintf('Media dei %d numeri random = %f\n',n,meanr);
stdr= std(z);            % calcolo la deviazione standard
fprintf('Deviazione standard dei %d numeri random = %f\n',n,stdr);
```

Non serve un'istruzione esplicita di return. Ogni valore non restituito è conosciuto solo localmente all'interno della funzione

Richiamare una funzione MATLAB



```
» [m, s] = simulate(100);  
Simulazione di 100 numeri random.....
```

```
Media dei 100 numeri random = 0.499702  
Deviazione standard dei 100 numeri random = 0.499702
```

```
» [m, s] = simulate(1000000);  
Simulazione di 1000000 numeri random.....
```

```
Media dei 1000000 numeri random = 0.499684  
Deviazione standard dei 1000000 numeri random = 0.288456
```

```
» m  
m =  
0.4997
```

```
» s  
s =  
0.2885
```