25/11/2013

Informatica per la Finanza
Informatica per le Scienze Economiche
Informatica e Programmazione per il Turismo

Script e Funzioni

- Tutte le funzioni, comprese molte di quelle già presenti in MatLab, vanno scritte su dei file denominati M-File
  - I file che contengono funzioni hanno estensione .m
- Le funzioni Matlab si dividono in due categorie:
  - **Script**: semplici sequenze di comandi MatLab che non hanno parametri in ingresso ed in uscita
    - Condividono lo stesso spazio delle variabili dell'ambiente di esecuzione (Workspace)
  - **Funzioni**: funzioni nel senso della programmazione C; hanno parametri in uscita ed in ingresso ed un proprio spazio di memoria
    - Un file può contenere più funzioni ma solo la prima è accessibile dall'ambiente comune, le altre possono essere richiamate solo all'interno del file
    - Il nome del file deve essere uguale al nome della prima funzione in esso contenuta

Fasi preliminari

- Gli M-file sono ricercati da MatLab in una serie di directory predefinite
  - Si visualizzano con il comando **`path`**
- Se si desidera creare una cartella nella quale salvare i propri file occorre aggiungere questo percorso a quelli conosciuti da MatLab:
  - Si usa il comando: **`addpath(`*`percorso`*`)`**
- Per verificare se lo M-file in cui è contenuto il codice è nei percorsi:
  - si usa il comando: **`which(`*`nomeM-file`*`)`**
    - Se risulta sconosciuto si deve aggiungere il percorso

Parti fondamentali di un M-File

Definizione funzione (solo per le funzioni!)

Riga H1: commento usato come help

```
function f = fact(n)
% Calcola il fattoriale di n.
% fact(n) restituisce il fattoriale
% di n, spesso indicato con n!

% in pratica, fact(n) e' prod(1:n).
f = prod(1:n);
```

Commento usato come ulteriore aiuto

Commento vero e proprio

Corpo della Funzione: inizia dopo la prima riga vuota dopo i commenti iniziali

Parti fondamentali di un M-file

- Il contenuto della riga H1 viene mostrato dai comandi **`help`** e **`lookfor`**.
  - **`lookfor`** ***`argomento`***: cerca nella riga H1 ***`argomento`***
- Il contenuto delle righe successive è mostrato dal comando **`help`**.
- I commenti sono di due tipi
  - Su di una riga: il commento è preceduto da %
  - Su più righe il commento è compreso fra

```
%{
...
}%
```

`%{` e **`}%`** devono essere scritti da soli sulla riga

Script

- Supponiamo di creare il file prScript.m

```
% riga di aiuto: esempio script
% calcola e disegna un grafico per la
% funzione y = r^2/(1-r) 0<= r <= 1

r = 0:0.05:1
y = r.^2./(1-r) % osservare il .
plot(r,y)
```

Strutture di controllo

- Le strutture di controllo presenti in Matlab sono:
  - **`if ... elseif ... else ... end`**
  - **`switch case ... otherwise ... end`**
  - **`for ... end`**
  - **`while ... end`**
    - Le istruzioni cicliche possono:
      - terminare anticipatamente con l'istruzione **`break`**
      - passare al ciclo successivo con l'istruzione **`continue`**
    - Le istruzioni cicliche possono essere spesso sostituite da istruzioni vettoriali

If

```
if espressione condizionale
  istruzioni
elseif espressione condizionale
  istruzioni
:
else
  istruzioni
end
```

La condizione in if e while

- L'espressione condizionale viene valutata nelle istruzioni **`if`** e **`while`** in maniera particolare se si usano matrici
  - Se la matrice contiene solo valori non nulli è valutata vera (equivale **`all(matr(:))`**); altrimenti falsa
  - Se la matrice è vuota è valutata falsa
  - Gli operatori logici elemento per elemento nelle istruzioni **`if`** e **`while`** valutano il secondo operando solo se il primo non ha portato ad una conclusione
    - Si comportano come operatori corto-circuito
    - si possono avere quindi dei risultati anche in espressioni che in generale portano ad un errore in esecuzione

La condizione in if e while

```
a = [8 9 -2];
b = [7 8 12 4];
a | b      ->  ERRORE
a & b      ->  ERRORE
if (a & b); disp 'vero'; end;
                          -> ERRORE
if (a | b); disp 'vero'; end;
```

In Octave dà errore; funziona così in Matlab -> **VERO**

Esempio

```
a=[1 2 90 0]
%a sara' valutato falso
b=[1 2 3]
% b sara' valutato vero

if (a & b)
    disp 'ciao'
else
    disp 'bye'
end
```


Esempio

- Scrivere il codice che verifica se la somma del contenuto dello scalare x vale 9 e nel caso visualizza ciao altrimenti bye

```
if sum(x)==9
    disp('ciao')
else
   disp('bye')
end
```

`disp()`: visualizza il contenuto dell' argomento.

Esempio

- Dato il valore di uno scalare x scrivere se è positivo, negativo o nullo

```
if (x == 0)
   disp('nullo')
elseif (x>0)
  disp('positivo')
else
   disp('negativo')
end
```


Esempio: operatori &&, ||, &, |

```
if 5>0 || NaN()
  disp('vero')
else
  disp('falso')
end
```

Eseguito visualizza:

```
vero
```

```
if -5>0 || NaN()
    disp('vero')
else
    disp('falso')
End
```

Eseguito visualizza:

```
NaN's cannot be
  converted to logicals.
```

Verifica 5>0 vero e non verifica la seconda parte
Verifica -5>0 falso e va a verificare la seconda parte

Esempio: operatori &&, ||, &, |

```
if 5>0 | NaN()
  disp('vero')
else
  disp('falso')
end
```

Eseguito visualizza:

```
vero
```

```
if -5>0 | NaN()
    disp('vero')
else
    disp('falso')
End
```

Eseguito visualizza:

```
NaN's cannot be
converted to logicals.
```

Verifica **5>0 vero** e non verifica la seconda parte
Verifica **-5>0 falso** e va a verificare la seconda parte
Si comporta come operatore corto-circuito!

```
A = 5>0 | NaN()
NaN's cannot be converted to logicals.
```

Comportamento naturale.

Esempio

- Data una matrice b verificare se tutti i suoi elementi sono positivi o se contiene qualche elemento uguale a 0 o minore di 0

Esempio

b>0 valuta **[1 1 1; 1 1 1]**
un vettore/matrice di valori non nulli equivale a **true**
un vettore/matrice con qualche 0 equivale a **false**
Quindi basta solo **all(b > 0) o addirittura solo b>0**

```
if b > 0
  disp('positivi');
elseif any(any(b==0))
   disp('qc zero');
else
   disp('qc <0');
end;
```

b è una matrice

```
>> b=[1 2 3; 4 5 6]
   positivi

>> b=[1 2 3; 4 0 6]
   qc zero

>>b=[1 2 3; 4 -2 6]
   qc <0
```


Esempio

- Data una matrice M 3x2 scrivere il codice che conta quanti elementi sono maggiori di 2:

```
sum(sum(M>2))
```

Esempio

Verificare se tutti gli elementi di una matrice sono maggiori di 2

```
%la matrice si DEVE chiamare M
if sum(sum(M>2)) == prod(size(M))
 disp('tutti maggiori di 2')
else
 disp('non tutti sono maggiori di 2')
end
```


Esempio

- Dato il vettore **`coef`** che contiene i coefficienti di una equazione di secondo grado ax^2+bx+c indicare quante radici reali possiede e poi indicarle (`coef(1)` contiene a)

```
Delta = coef(2)^2 - 4*coef(1)*coef(3);
if Delta >0
  disp('due radici reali')
elseif Delta == 0
  disp('due radici reali coincidenti')
else
  disp('due radici complesse')
end
roots(coef)
```

while

```
while condizione
  istruzioni
end
```


Esercizio

```
%{
Dati i valori nelle variabili X ed Y
(numeri interi)
sommare tutti i valori fra X ed Y
%}
somma=0;
while X < Y
    somma = somma + X;
    X = X + 1;
end
```

Commento su più righe; l'apertura e la chiusura sono da soli sulla riga!

Ricordarsi che lo script modifica l'ambiente Globale (Workspace)

switch

```
switch espressione
   case valore1
       istruzioni
   case valore2
       istruzioni
   :
   otherwise
      istruzioni
end
```

- ***espressione***
  - Coinvolge scalari o stringhe
- ***valore***
  - È il valore da confrontare
    - Se ci sono più valori vanno racchiusi fra parentesi graffe
    - Valore deve stare sulla stessa riga di **case**
- Al primo case che verifica la condizione l'istruzione termina
  - NON funziona come in C
- La clausola **otherwise** è applicata quando non vi è corrispondenza con nessun valore in **case**

Esempio

- Voglio confrontare il valore dello scalare X
  - Se X è uguale a
    - 5 7 9 scrivo qc
    - -1 -2 -3 scrivo qc2
  - Altrimenti scrivo qc3

```
switch X
 case {5, 7, 9}
      disp ('qc')
 case {-1, -2, -3}
      disp ('qc2')
 otherwise
      disp ('qc3')
end
```

switch

- Provare il seguente codice con diversi valori di **`variabile`**:

```
switch variabile
  case 1
    disp('1')
  case {2,3,4}
    disp('2 o 3 o 4')
  case 5
    disp('5')
  otherwise
    disp('altro')
end
```


Esempio

```
switch var / 3 + 1
  case 1
    disp('1')
  case {2,3,4}
    disp('2 o 3 o 4')
  case 5
    disp('5')
  otherwise
    disp('altro')
end
```

for

```
for indice = part:incr:fine
  istruzioni
end
```

- ***incr*** è facoltativo, se omesso vale 1
- ***part:incr:fine*** può essere una matrice, in questo caso la matrice è manipolata colonna per colonna

Esempio

- Sommare tutti i numeri compresi fra X ed Y

```
s=0;
for i=X:Y
    s = s + i;
end
```

Esempio

```
X=[-3 4 12; 3 1 2];
somma= 0;
for i=X
    disp(i);
    somma = somma + i;
end
disp(somma)
```


for

```
m=[1 2 3;
   4 5 6;
   7 8, 9;
   10 11, 12];
for k = m, k', end

ans =
     1     4     7    10
ans =
     2     5     8    11
ans =
     3     6     9    12
```

- Si notino i diversi usi dei caratteri `,` e `;`
  - Il carattere `,` è usato come
    - separatore per gli elementi di una matrice
    - separatore per le diverse parti di una istruzione
  - Il carattere ; è usato come
    - terminatore di riga nella definizione di una matrice
    - per non visualizzazione del risultato prodotto da una espressione

Vettorizzazione

- Spesso è possibile sostituire il ciclo **`for`** utilizzando la capacita di Matlab di compiere operazioni sui vettori.
  - Le operazioni vettorizzate sono più semplici di quelle che usano i cicli
  - Per valutare l'efficienza
    - si possono racchiudere le istruzioni da verificare fra l'istruzione

      `tic; … varT=toc;`

      Al termine dell'esecuzione delle istruzioni il tempo trascorso sarà scritto nella variabile **`varT.`**

Vettorizzazione

Scrivere il valore della somma di tutti i numeri pari compresi fra 2 e 5000

```
tic
I=2;
sm=0;
while I<= 5000
  sm = sm + I;
  I = I + 2;
end
t=toc

tic
sm=0;
for I=2:2:5000
  sm = sm + I;
end
q=toc
```

```
tic
v=2:2:5000;
sum(v);
z=toc

tic
sum(2:2:5000);
k=toc
```

Media dei valori su 1000 esecuzioni:

```
t = 2.7895e-05
q = 2.4135e-05
z = 1.7165e-05
k = 2.2913e-05
```

Vettorizzazione

```
M=[];
for lim=50:100:100000
    tic;
    for z=1:lim ;
        x(z) = z * 5;
    end;
    tTra=toc;

    tic;
    y = 1:lim;
    q = y * 5;
    tT=toc;

    M=[M;lim,tTra, tT];

 %esce quando modo iterativo peggiore
 %modo vettoriale
    if (tTra > tT)
        break;
    end
end
```

```
M =  50    1.0755e-05     1.8088e-05
     150   1.5644e-05     7.3333e-06
```


Esempio

- dato un vettore di numeri decimali di 20 elementi caricare una matrice inserendo nella prima riga il vettore dato, nella seconda il doppio dei valori nella prima riga; quindi visualizzare un grafico nel quale i valori della prima riga fungono da ascissa, quella della seconda da ordinata

```
tic;
vet = rand(1,20);
vet= sort(vet);
matr=[vet;vet*2];
plot((matr(1,:))',(matr(2,:).^2)');
tT=toc;
```