Programmare

Condizioni Logiche
Costrutti Decisionali
Costrutti di Ciclo

Condizioni logiche

Le condizioni logiche sono istruzioni che fanno uso di:

- *operatori relazionali* (servono a confrontare il valore di due diverse variabili),
- *operatori logici* (servono a concatenare logicamente più istruzioni relazionali)

Operatori Relazionali	
Uguale	= =
Diverso	~ =
Minore	<
Maggiore	>
Minore Uguale	< =
Maggiore Uguale	> =

Operatori Logici	
AND	&
NOT	~
OR	|

Operatore Relazionali

Sono operatori binari che eseguono il confronto fra i valori di due variabili.

Gli operandi possono essere SCALARI o MATRICI

```
X == Y
```

SCALARI

Operandi Scalari possono essere:
- due variabili
- una variabile ed una costante
- Viene restituito VERO (1) se la relazione è verificata altrimenti FALSO (0)

MATRICI

Se operandi sono due Vettori o due Matrici di uguali dimensioni allora:
- il confronto viene eseguito elemento a elemento
- viene restituito un vettore o una matrice delle stesse dimensioni degli operandi con elementi uguali ad 1 nelle posizioni dei valori che verificano la relazione e 0 nelle altre posizioni

Confronto fra scalari

Confronto fra una variabile (x) ed una costante (3 o 5):

```
>> x = 5;
>> x == 3
ans
     0
```
FALSO

```
>> x = 5;
>> x ~= 3
ans
     1
```
VERO

```
>> x = 5;
>> x > 3
ans
     1
```
VERO

```
>> x = 5;
>> x < 3
ans
     0
```
FALSO

```
>> x = 5;
>> x >= 5
ans
     1
```
VERO

Confronto fra due variabile (x e y):

```
>> y = 3;
>> x = 5;
>> x == y
ans
     0
```
FALSO

```
>> y = 3;
>> x = 5;
>> x ~= y
ans
     1
```
VERO

```
>> y = 3;
>> x = 5;
>> x >= y
ans
     1
```
VERO

```
>> y = 3;
>> x = 5;
>> x < y
ans
     0
```
FALSO

```
>> y = 5;
>> x = 5;
>> x >= y
ans
     1
```
VERO

Confronto fra vettori

Confronto fra due vettori

```
>> r_x = [ 1 2 3];
>> r_y = [-1 7 3];
>> r_x == r_y

ans
       [0 0 1]
```

```
>> r_x = [ 1 2 3];
>> r_y = [-1 7 3];
>> r_x <= r_y

ans
       [0 1 1]
```

```
>> r_x = [ 1 2 3];
>> r_y = [-1 7 3];
>> r_x ~= r_y

ans
       [1 1 0]
```

Le dimensioni dei vettori da confrontare devono essere le stesse

```
>> r_x = [ 1 2 3];
>> r_y = [-1 7 3 0];
>> r_x == r_y

??? Error using ==> ==
Matrix dimensions must agree.
```


Operatori Logici: & e |

Operatori BINARI che mettono in relazione due o più condizioni logiche restituendo un valore:

VERO (1) o FALSO (0).

```
condizione_1 & condizione_2

condizione_1 | condizione_2
```

Operatore Logico: ~

Operatore UNARIO che inverte il valore di della condizione logica a cui viene applicato restituendo il valore:

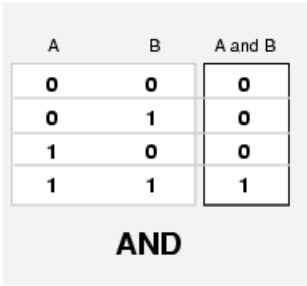
- VERO se la condizione è falsa
- FALSO se la condizione è vera.

```
~condizione_1
```

Operatore Logico &

& (AND)

restituisce VERO (1) se TUTTE le condizioni sono vere altrimenti restituisce il valore FALSO (0)



A	B	A and B
0	0	0
0	1	0
1	0	0
1	1	1

AND

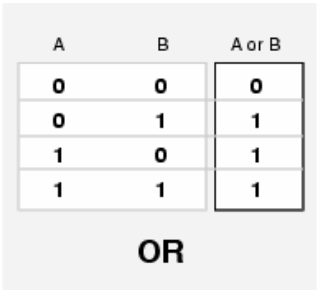
`>> y=3;`
`>> x=5;`
`>> x=5 & y=3`
`ans`
`1`
	`>> y=3;`
	`>> x=5;`
	`>> x=5 & y=x`
	`ans`
	`0`
		`>> x=5;`
		`>> x>=0 & x<=7`
		`ans`
		`1`
			`>> x=5;`
			`>> x>0 & x<=7`
			`ans`
			`1`

VERO	**FALSO**	**VERO**	**VERO**
		$x \in [\,0, 7\,]$	$x \in \,]\,0, 7\,]$

Operatore Logico |

| (OR)

restituisce VERO (1) se ALMENO una delle condizioni risulta vera altrimenti restituisce FALSO (0)



A	B	A or B
0	0	0
0	1	1
1	0	1
1	1	1

OR

`>> y=3;`
`>> x=5;`
`>> x=5 | y=x`
`ans`
`1`
	`>> y=3;`
	`>> x=5;`
	`>> x=3 | y=5`
	`ans`
	`0`
		`>> x=10;`
		`>> x<=0 | x>=7`
		`ans`
		`1`
			`>> x=10;`
			`>> x<0 | x>=7`
			`ans`
			`1`

VERO	**FALSO**	**VERO**	**VERO**
		$x \notin [\,0, 7\,]$	$x \notin \,]\,0, 7\,]$

Operatore Logico ~

~ (NOT)

restituisce VERO (1) se la condizione è vera altrimenti restituisce FALSO (0)

A	not A
0	1
1	0

NOT

Costrutti Decisionali

Permettono di eseguire un blocco di istruzioni se una certa condizione logica risulta vera o falsa

Ne esistono di 3 tipi:

1. if - end
2. if – else – end
3. if - elseif – else – end

```
if condizione
   blocco
end
```

```
if condizione
    blocco 1
else
   blocco 2
end
```

```
if condizione_1
    blocco 1
elseif condizione_2
    blocco 2
elseif condizione_3
    blocco 3
….
else
   blocco 2
end
```

if…end

Se la condizione logica risulta VERA allora viene eseguito il blocco di codice compreso fra `if` ed `end`, altrimenti l'esecuzione passa all'istruzione seguente: *istruzione_1*

```
if condizione
   blocco
end
istruzione_1
```

```
if x > 2
   x = x/2;
end
x
```

Se lo scalare x risulta maggiore di 2 allora il suo valore viene diviso per 2 ed il risultato assegnato alla variabile x che viene poi visualizzata.

```
if x >= -2 & x <= 2
   y = x*10;
end
y
```

Se lo scalare x è compreso fra i valori -2, 2 allora il suo valore moltiplicato per 10 viene assegnato alla y che viene poi visualizzata

if…else…end

Se la condizione logica risulta verificata allora vengono eseguite le istruzioni del *blocco_1* altrimenti quelle del *blocco_2*, quindi l'esecuzione passa all'istruzione seguente (*istruzione_1*)

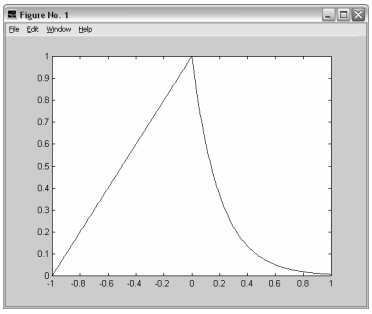
```
if condizione
   blocco_1
else
   blocco_2
end
istruzione_1
```

```
if x ~= 0
   y = 10/x;
else
   y = 1;
end
```

Se lo scalare x risulta diverso da 0 allora ad y viene assegnato il valore 10/x altrimenti ad y viene assegnato il valore 1.

Se lo scalare x è esterno all' intervallo di valori [-2, 2] allora alla variabile y viene assegnato il valore 0, altrimenti y assume il valore della x.

```
if x < -2 | x > 2
   y = 0;
else
   y = x;
end
```



Si vuole eseguire il grafico della funzione:

$$F(x) = \begin{cases} 1 - x & x \in [-1, 0[\\ e^{-5x} & x \in [0, 1] \end{cases}$$

1. E' stato creato prima il vettore r_x di 200 valori compresi fra -1 e 1.
2. Quindi il vettore r_y con dimensioni uguali a r_x ed elementi tutti nulli.
3. È stato realizzato un ciclo for con n che assume valori da 1 a 200 passo 1, ossia tutte le posizioni del vettore r_y.
4. Con un if sono stati controllati i singoli valori r_x(n) e opportunamente definiti i corrispondenti r_y(n)

```
r_x = -1:0.01:1;
r_y = zeros(size(r_x));
for n=1:length(r_x)
    if r_x(n) < 0
       r_y(n) = 1-r_x(n);
    else
       r_y(n) = exp(-5*r_x(n));
    end
end
plot(r_x, r_y);
```


if…elseif…else…end

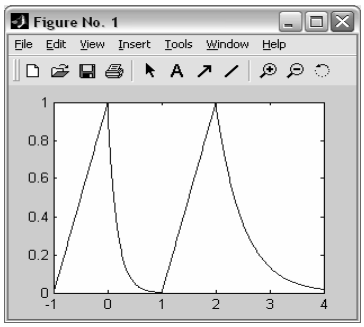
Verrà eseguito solo il blocco di codice relativo alla condizione che risulta verificata. Se nessuna delle condizioni logiche è verificata allora vengono eseguite le istruzioni del *blocco_else*, se presente. Quindi l'esecuzione passa all'istruzione seguente (*istruzione_1*)

Le condizioni devono essere mutuamente esclusive, ossia non devono essere verificate contemporaneamente

```
if condizione_1
   blocco_1
elseif
condizione_2
   blocco_2
elseif
condizione_3
   blocco_3
 elseif …
…
else
   blocco_else
end
istruzione_1
```

Si vuole eseguire il grafico della funzione: $F(x) = \begin{cases} 1 - x & x \in [-1,\ 0[\\ e^{-5x} & x \in [\ 0,\ 1[\\ 1 + x & x \in [\ 1,\ 2[\\ e^{-2x} & x \in [\ 2,\ 4] \end{cases}$

```
r_x = -1:0.01:4;
r_y = zeros(size(r_x));
for n=1:length(r_x)
    if r_x(n) < 0
        r_y(n) = 1+r_x(n);
    elseif r_x(n)>= 0 & r_x(n)< 1
        r_y(n) = exp(-5*r_x(n));
    elseif r_x(n)>= 1 & r_x(n)< 2
        r_y(n) = r_x(n)-1;
    else
        r_y(n) = exp( -2*(r_x(n)-2) );
    end
end
plot(r_x, r_y);
```



La funzione `trigo_plot` esegue il grafico di una funzione trigonometrica nell'intervallo [0, 2π] sulla base del parametro `n` passato alla funzione.

```
function trigo_plot(n)

r_x = 0:0.1:2*pi;

if n==1
   plot(r_x, sin(r_x))
elseif n==2
   plot(r_x, cos(r_x))
elseif n==3
   plot(r_x, tan(r_x))
else
   plot(r_x, r_x.^2)
end

return
```

Costrutti di ciclo

Permettono di ripetere un blocco di istruzioni il numero di volte desiderato

Ne esistono di due tipi:

1. while - end
2. for – end

```
while condizione
   blocco ciclo
end
```

```
for …
   blocco ciclo
end
```


Ciclo for

I cicli for utilizzano una variabile pivot che viene incrementata automaticamente ad ogni iterazione di un valore definito nStep, a partire dal valore iniziale nStart, fino al valore finale nStop.

```
for n=nStart:nStep:nStop
    blocco_ciclo
end
```

Il blocco di istruzioni compreso fra il comando for ed il comando end viene ripetuto finché la variabile pivot non raggiunge il valore finale.

Esempio ciclo for

```
r_x = zeros(1,6);
for n=1:2:length(r_x)
    r_x(n) = -n;
end

for n=2:2:length(r_x)
    r_x(n) = n;
end
r_x
```

Questo blocco di codice pone tutti gli elementi del vettore r_x in posizione dispari uguali all'indice n cambiato di segno e quelli in posizione pari uguali all'indice n.

```
>> r_x
   -1    2   -3    4   -5    6
```

Lo script Matlab riportato permette di definire in modo differenziato i valori del vettore r_x nelle posizioni pari e dispari rispettivamente.

Il primo ciclo `for`, infatti, ha una variabile pivot n che assumerà valori da 1 a 6 passo 2, ossia n = 1, 3, 5. Dopo di esso `r_x` risulta:

```
>> r_x
  -1    0    -3    0    -5     0
```

Mentre nel secondo ciclo `for` la variabile pivot n assume valori da 2 a 6 passo 2, ossia n = 2, 4, 6. Dopo il secondo ciclo r_x risulta:

```
>> r_x
   -1    2    -3    4    -5    6
```

Script Matlab

```
r_x = zeros(1,6);
for n=1:2:length(r_x)
    r_x(n) = -n;
end
r_x
for n=2:2:length(r_x)
    r_x(n) = n;
end
r_x
```


Sommatorie con ciclo `for`

Questo blocco di codice utilizza un ciclo `for` per effettuare la sommatoria di tutti gli elementi del vettore `r_x` e confronta il risultato con la funzione `sum`.

Si noti come prima del ciclo sia stata creata una variabile scalare `somma`, inizializzata a zero, a cui ad ogni iterazione del ciclo viene sommato un elemento del vettore in posizione via via crescente.

```
r_x = rand(1,6)
somma = 0;

for n=1:length(r_x)
   somma = somma + r_x(n);
end

disp(sprintf('ciclo = %f', somma))
disp(sprintf('sum   = %f', sum(r_x)))
```

```
>> r_x =
   0.9218    0.7382    0.1763    0.4057    0.9355    0.9169

ciclo = 4.094367
sum   = 4.094367
```

Produttorie con ciclo for

Questo blocco di codice utilizza un ciclo `for` per effettuare la produttoria di tutti gli elementi del vettore `r_x` e confronta il risultato con la funzione `sum`.

Si noti come prima del ciclo sia stata creata una variabile scalare `prodotto`, inizializzata ad uno, che ad ogni iterazione del ciclo viene moltiplicata per elemento del vettore in posizione via via crescente.

```
r_x = rand(1,6)
prodotto = 1;
for n=1:length(r_x)
   prodotto = prodotto * r_x(n);
end

disp(sprintf('ciclo = %f', prodotto))
disp(sprintf('sum   = %f', prod(r_x)))
```

```
>> r_x =
   0.9218    0.7382    0.1763    0.4057    0.9355    0.9169
ciclo = 0.041740
prod  = 0.041740
```


Ciclo while-end

Esegue il blocco di istruzioni compreso fra il comando while ed il comando end finché la condizione risulta verificata

```
while condizione
    blocco_ciclo
end
```

Il ciclo while riportata al lato permette di trovare la potenza di due immediatamente superiore al valore di x: in questo caso p=9 e 2^9=512.

Ad ogni iterazione del ciclo viene incrementato il valore di p finché 2^p risulta inferiore ad x, appena p raggiunge il valore 9 poiché la condizione 2^p<x non è più verificata il ciclo termina e il valore di p viene visualizzato

```
x = 505;
p = 1;
while 2^p < x
    p=p+1;
end
p
```

Ciclo while-end

Esegue il blocco di istruzioni compreso fra il comando while ed il comando end finché la condizione risulta verificata

```
while condizione
    blocco_ciclo
end
```

Il ciclo while riportata al lato permette di trovare la potenza di due immediatamente superiore al valore di x: in questo caso p=9 e 2^9=512.

Ad ogni iterazione del ciclo viene incrementato il valore di p finché 2^p risulta inferiore ad x, appena p raggiunge il valore 9 poiché la condizione 2^p<x non è più verificata il ciclo termina e il valore di p viene visualizzato

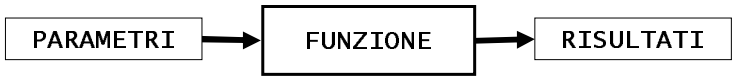
```
x = 505;
p = 1;
while 2^p < x
    p=p+1;
end
p
```


Funzioni

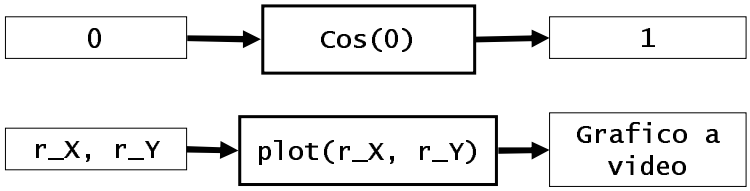
Funzioni di Libreria
Funzioni definite dall'utente

Funzioni in Matlab

Una funzione in programmazione è un blocco di codice (sequenza di istruzioni) che operano su di un insieme definito di variabili passate alla funzione stessa come parametri (variabili di input) mediante un'opportuna sintassi:



La funzione può restituire, oppure no, delle variabili risultato:



Funzioni

L'ambiente Matlab permette all'utente:

- di utilizzare una larga serie di funzioni già presenti nell'ambiente di sviluppo,
- di creare nuove funzioni in base alle esigenze specifiche del programma da realizzare.

Funzioni di Libreria

Funzioni Utente

Sia le funzioni di libreria che le funzioni utente possono essere utilizzate direttamente in finestra di comando (*modalità console*) o essere richiamate all'interno di uno script (*modalità interprete*) utilizzando l'opportuna sintassi

Sintassi

La sintassi di chiamata di una funzione consiste nell'elenco ordinato delle variabili:

1. di input: i parametri, ossia i valori da passare alla funzione,
2. di output: i risultati: ossia i valori ottenuti dalla funzione.

```
[p, q, …] = function (x, y, z…)
```

Elenco risultati | Nome Funzione | Elenco parametri

Sintassi

```
[p, q, …] = function (x, y, z…)
```

Nome Funzione

Il nome di una funzione è una qualsiasi sequenza di lettere e numeri, con in prima posizione una lettera, senza altri simboli eccetto '_'

Elenco parametri

L'elenco delle variabili da passare ad una funzione come parametri di input va scritto: dopo il nome della funzione fra parentesi tonde, separati da virgole.

Elenco risultati

I valori dei risultati di output di una funzione possono essere assegnati ad variabili i cui nomi siano posti, fra parentesi quadre, sulla sinistra del segno uguale, separati da virgole.

Funzioni di Libreria

Le Funzioni di libreria sono funzioni disponibili nell'ambiente di calcolo Matlab e possono essere usate sia in modalità console che interprete.

Ne esistono di due tipi:

- Funzioni in formato binario: (*built-in functions*) sono di solito le funzioni matematiche usate più spesso e che quindi devono essere più efficienti, e vengono distribuite con la versione base di Matlab.
- Funzioni presenti come *scripts*: (*M-file functions*) sono di solito funzioni che risolvono metodi numerici più complessi (calcolo di integrali, soluzione di sistemi di equazioni differenziali, ecc.) e possono essere vendute a parte rispetto al programma principale sottoforma di estensioni dette *Toolbox.*

NOTA BENE: La sintassi di chiamata resta comunque la stessa !

Esempi funzioni di Libreria

```
>> [r, c] = size (rand(2, 3))
r =
       2
c =
       3
```

funzione:	`size`
1 parametro:	1 matrice
2 risultati:	2 scalari

```
>> m_x = zeros (2, 3)
m_x =
       0      0      0
       0      0      0
```

funzione:	`zeros`
2 parametri:	2 scalari
1 risultato:	1 matrice

```
>> x = cos(pi)
x =
       0
>> r_x = cos([0, pi])
r_x =
       1      0
```

funzione:	`cos`
1 parametri:	1 scalare o matrice
1 risultato:	1 scalare o matrice

Funzioni Utente

Si possono creare funzioni utilizzando il comando riservato `function`:

La funzione `converti` accetta la variabile `angolo` in gradi e ne restituisce il valore in radianti: variabile `rad`.

```
function [rad] = converti(angolo)
        rad = angolo/180*pi;
return
```

1. Il primo rigo della funzione deve contenere la sintassi della funzione che si vuole realizzare preceduta dal comando `function`,
2. i parametri, se presenti, devono essere elencati fra parentesi tonde e separati da virgole sulla destra del nome della funzione,
3. i risultati, variabili di output, se presenti, devono essere elencati fra parentesi quadre, separati da virgole sulla sinistra del segno uguale,
4. `return` fa terminare l'esecuzione della funzione (può essere omesso).

Funzioni Globali

FUNZIONE GLOBALE → è una funzione che può essere utilizzata sia:

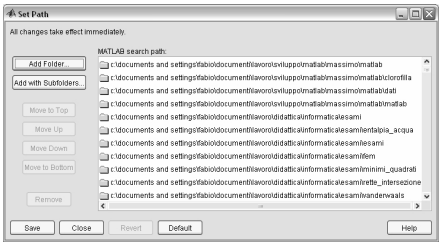
- all'interno di un qualsiasi script,
- in finestra di comando.

1. La funzione deve essere creata in uno script a se stante, salvato in un M-File che ha lo stesso nome della funzione:

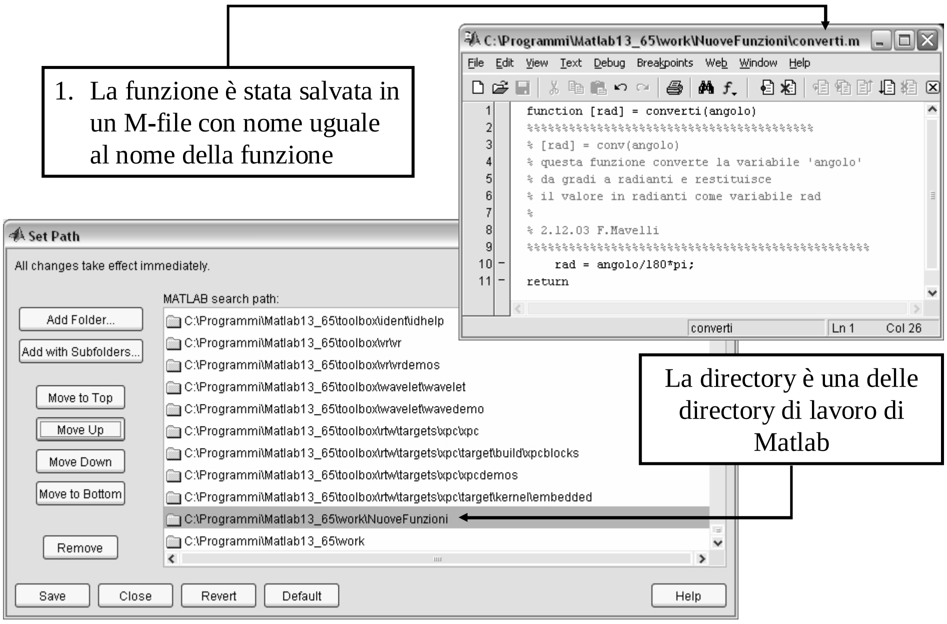
   nome file → converti.m

2. Il file con la funzione deve essere salvato in una delle directory di lavoro di Matlab.

```
function [rad]=converti(angolo)
        rad = angolo/180*pi;
return
```



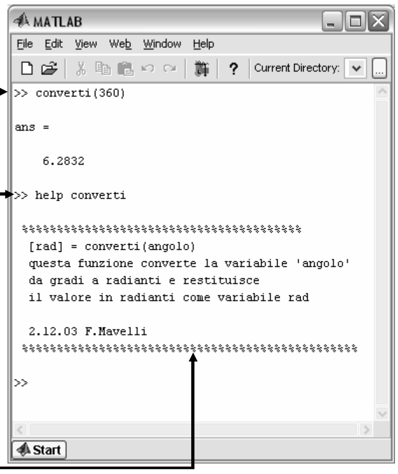
1. La funzione è stata salvata in un M-file con nome uguale al nome della funzione

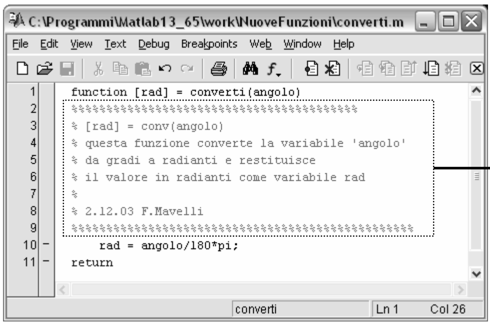


La directory è una delle directory di lavoro di Matlab

La funzione può, quindi, essere utilizzata direttamente in finestra di comando.

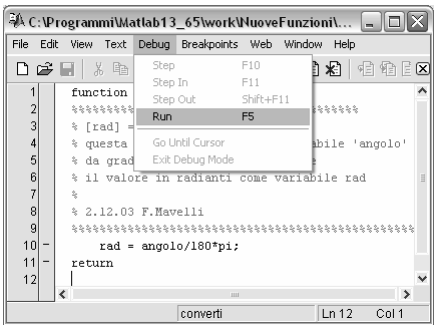
Il comando `help` può essere utilizzato per visualizzare i commenti inseriti subito dopo il comando `function` nello script

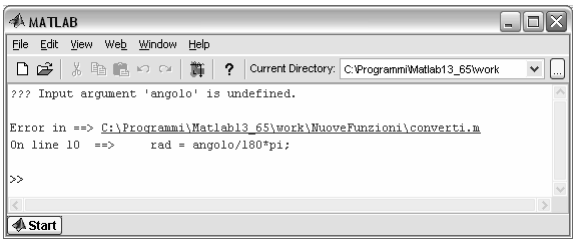




Funzioni Globali

Una funzione non può essere mandata in esecuzione direttamente dall'edito degli script (tasto F5 o voce di menù Debug→Run) perché Matlab restituisce un messaggio di errore





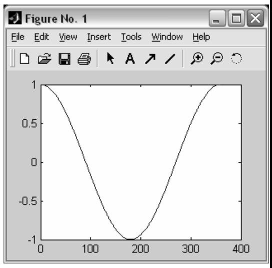
Infatti risulta non definita la variabile `angolo` da passare alla funzione

Funzioni Locali

FUNZIONE LOCALE → è una funzione che può essere utilizzata solo all'interno dello script in cui è definita.

All'interno di uno script possono essere definite delle funzioni locali solo se lo script stesso è una funzione.

Uno script può essere sempre definito come una funzione senza parametri di input e variabili di output.

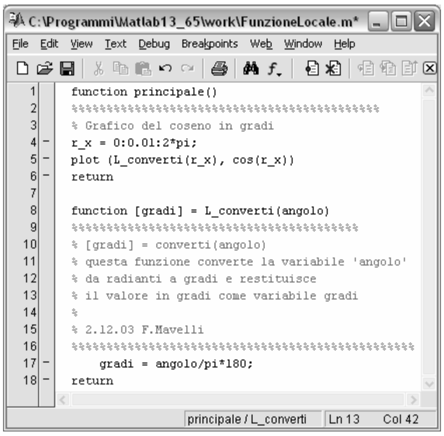


Lo script `FunzioneLocale.m` permette di eseguire il grafico del coseno nell'intervallo [0, 2π] riportando i valori degli angolo in gradi anziché in radianti.

E' stata creata una prima funzione senza variabili di input e di output che viene eseguita se si lancia lo script

Questa funzione chiama la funzione locale `L_converti`

La funzione locale `L_converti` viene definita alla fine della funzione `principale` (dopo il comando `return`)



Funzione `feval`

```
y = feval('NomeFunzione', variabile)
```

La funzione `feval` permette di valutare una funzione calcolata per un certo valore delle variabili indipendenti passandogli il nome della funzione come variabile stringa

```
>> sin (2)
ans =

    0.9093

>> feval('sin', 2)
ans =

    0.9093
```

```
>> r_y = feval('sin', 0:0.5:pi)
r_y =
     0    0.4794    0.8415    0.9975    0.9093    0.5985    0.1411    0.9093
```

feval

La funzione locale L_myplot automatizza la creazione di un grafico per 4 differenti funzioni i cui nomi vengono passati come variabili stringa. Il grafico è riportato nella diapositiva seguente

Si noti come la IV funzione sia anch'essa una funzione locale

```
function main()
r_x = 0:0.1:pi;
figure(1);
subplot(2,2,1), L_myplot('sin', r_x)
subplot(2,2,2), L_myplot('cos', r_x)
subplot(2,2,3), L_myplot('exp', r_x)
subplot(2,2,4), L_myplot('func', r_x)
function r_y = func(r_x)
    r_y = sin(r_x).*cos(r_x);
return
function L_myplot(s_Name, r_x)
    r_y = feval(s_Name, r_x);
    plot(r_x, r_y),
    xlim([r_x(1), r_x(end)])
    xlabel('x'), ylabel(s_Name)
return
```