

UNIVERSITÀ DEGLI STUDI DI SALERNO

Fondamenti di Informatica

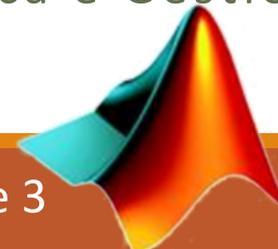
Introduzione alla programmazione in MATLAB: Parte 3 (La Ricorsione e Debugging)

Prof. Christian Esposito

Corso di Laurea in Ingegneria Meccanica e Gestionale (Classe I)

A.A. 2017/18

MATLAB



OUTLINE

- La ricorsione - Principi base
 - Caso Studio 1: Fattoriale
 - Caso Studio 2: Fibonacci

- Debugging dei Programmi MATLAB

Principi base (1)

Il concetto di ricorsione in informatica si riconduce a quello di induzione matematica.

Sia P un predicato sull'insieme N dei numeri naturali e sia vero il predicato $P(0)$; se per ogni K intero, dal predicato $P(k)$ discende la verità di $P(k+1)$ allora $P(n)$ è vero per qualsiasi n .

Principi base (2)

La dimostrazione induttiva avviene in due passi:

- Passo base : dimostrare $P(0)=\text{vero}$;
- Passo induttivo: dimostrare che per ogni $k>0$, si ha che

$$P(k) \longrightarrow P(k+1)$$

Così come nel principio di induzione la verità di $P(k+1)$ discende dalla verità dello stesso predicato $P(k)$, il calcolo di un funzione ricorsiva avviene mediante il calcolo della stessa funzione in un passo successivo.

Caso Studio: Fattoriale (1)

- Il **fattoriale** di un intero (positivo) n , indicato con $n!$, è il prodotto di tutti gli interi positivi minori o uguali di n (si noti che $0! = 1$)

Caso Studio: Fattoriale (1)

- Il **fattoriale** di un intero (positivo) n , indicato con $n!$, è il prodotto di tutti gli interi positivi minori o uguali di n (si noti che $0! = 1$)

$$n! = \prod_{i=0}^n i$$

Caso Studio: Fattoriale (1)

- Il **fattoriale** di un intero (positivo) n , indicato con $n!$, è il prodotto di tutti gli interi positivi minori o uguali di n (si noti che $0! = 1$)

$$n! = \prod_{i=1}^n i = 1 * 2 * 3 * \dots * (n - 1) * n$$

Caso Studio: Fattoriale (1)

- Il **fattoriale** di un intero (positivo) n , indicato con $n!$, è il prodotto di tutti gli interi positivi minori o uguali di n (si noti che $0! = 1$)

$$n! = \prod_{i=0}^n i = 1 * 2 * 3 * \dots * (n - 1) * n$$
$$\prod_{i=0}^{n-1} i$$

Caso Studio: Fattoriale (1)

- Il **fattoriale** di un intero (positivo) n , indicato con $n!$, è il prodotto di tutti gli interi positivi minori o uguali di n (si noti che $0! = 1$)

$$n! = \prod_{i=0}^n i = 1 * 2 * 3 * \dots * (n - 1) * n$$

The diagram illustrates the recursive definition of the factorial function. It shows the equation $n! = \prod_{i=0}^n i = 1 * 2 * 3 * \dots * (n - 1) * n$. An arrow points from the product term $1 * 2 * 3 * \dots * (n - 1)$ to a second equation: $\prod_{i=0}^{n-1} i$. A second arrow points from this second equation to a box containing $(n - 1)!$.

Caso Studio: Fattoriale (1)

- Il **fattoriale** di un intero (positivo) n , indicato con $n!$, è il prodotto di tutti gli interi positivi minori o uguali di n (si noti che $0! = 1$)
- **Definizione ricorsiva (parziale):**

$$n! = (n - 1)! * n$$

Caso Studio: Fattoriale (2)

- **Definizione ricorsiva:**

$$n! = \begin{cases} 1, & \text{se } n = 0 \\ (n - 1)! * n, & \text{se } n > 0 \end{cases}$$

Caso Studio: Fattoriale (2)

- **Definizione ricorsiva:**

$$n! = \begin{cases} 1, & \text{se } n = 0 \\ (n - 1)! * n, & \text{se } n > 0 \end{cases}$$

Caso base

Caso Studio: Fattoriale (2)

- Definizione ricorsiva:

$n!$ =

Il **caso base** è necessario affinché la ricorsione termini invece di avere una «ricorsione infinita»

Caso base

Caso Studio: Fattoriale (2)

- **Definizione**

Il **caso base** è necessario affinché la ricorsione termini invece di avere una «ricorsione infinita»

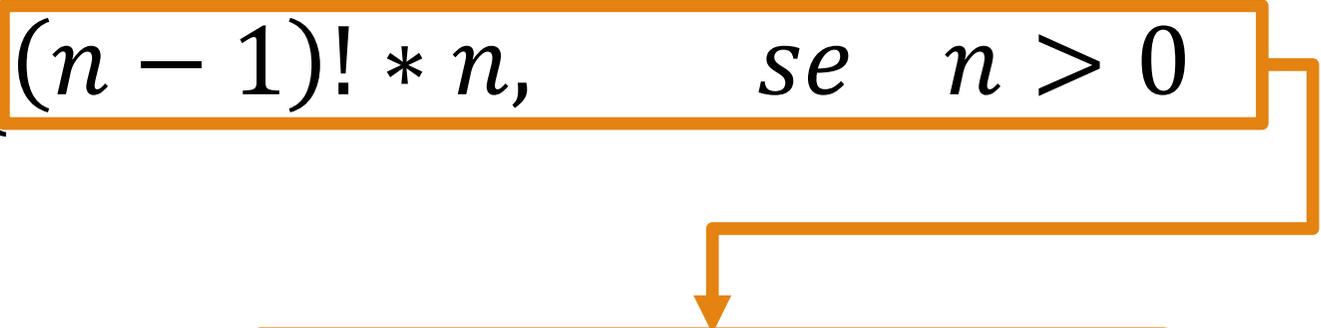
$n!$ =

Nel nostro esempio, se n è uguale a 0, allora la funzione restituirà 1 (dato che $0! = 1$)

Caso base

Caso Studio: Fattoriale (2)

- **Definizione ricorsiva:**

$$n! = \begin{cases} 1, & \text{se } n = 0 \\ (n - 1)! * n, & \text{se } n > 0 \end{cases}$$


**Versione più semplice
della definizione**

Caso Studio: Fattoriale (3)

- Codice MATLAB

$$n! = \begin{cases} 1, & \text{se } n = 0 \\ (n-1)! * n, & \text{se } n > 0 \end{cases}$$

```
function [n_fact] = fattoriale_ricorsivo(n)
    if n == 0
        n_fact = 1;
    else
        n_fact = fattoriale_ricorsivo(n - 1) * n;
    end
end
```

Caso Studio: Fattoriale (3)

- Codice MATLAB

$$n! = \begin{cases} 1, & \text{se } n = 0 \\ (n-1)! * n, & \text{se } n > 0 \end{cases}$$

```
function [n_fact] = fattoriale_ricorsivo(n)
    if n == 0
        n_fact = 1;
    else
        n_fact = fattoriale_ricorsivo(n - 1) * n;
    end
end
```

Invocazione ricorsiva

Caso Studio: Fattoriale (4)

- Dietro le quinte...

`fattoriale_ricorsivo(4)`

Caso Studio: Fattoriale (4)

- Dietro le quinte...

```
fattoriale_ricorsivo(4)
```

Caso Studio: Fattoriale (4)

- Dietro le quinte...

```
fattoriale_ricorsivo(4)
```

```
4 * fattoriale_ricorsivo(3)
```

Caso Studio: Fattoriale (4)

- Dietro le quinte...

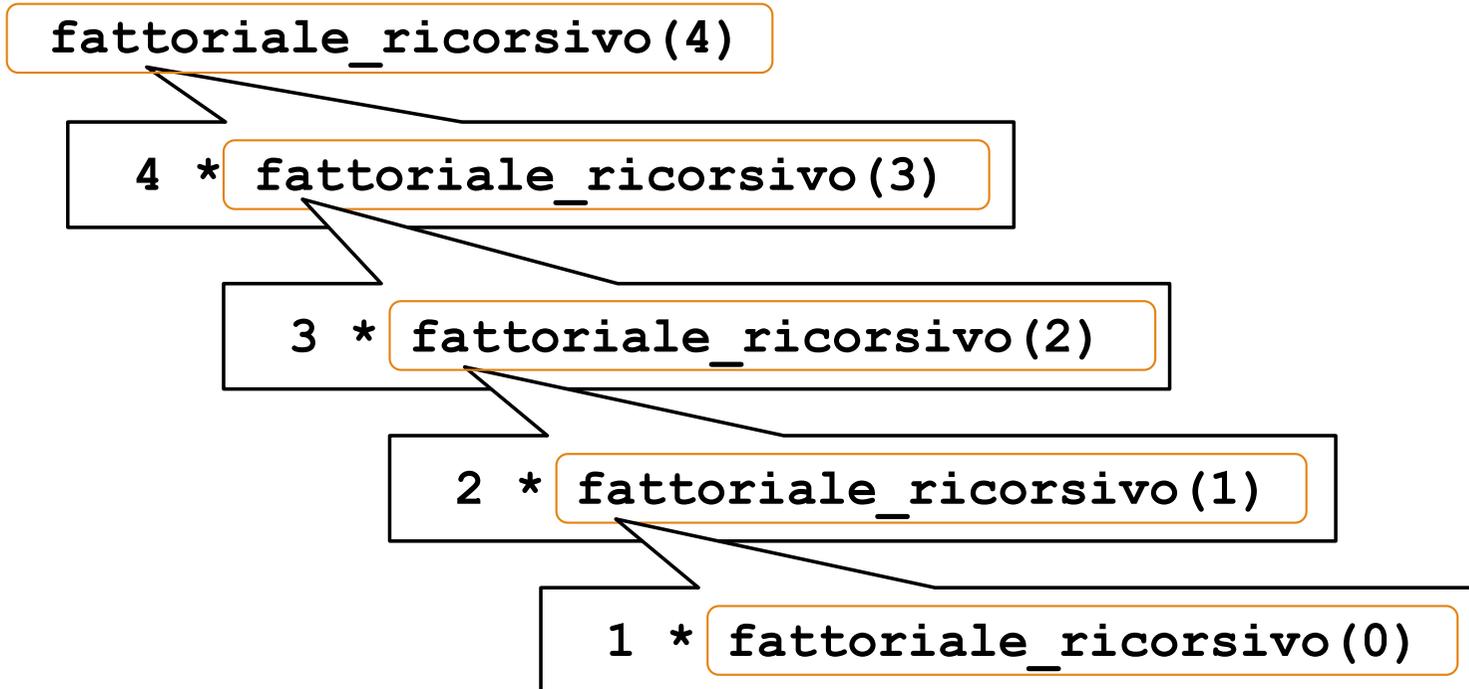
`fattoriale_ricorsivo(4)`

`4 * fattoriale_ricorsivo(3)`

`3 * fattoriale_ricorsivo(2)`

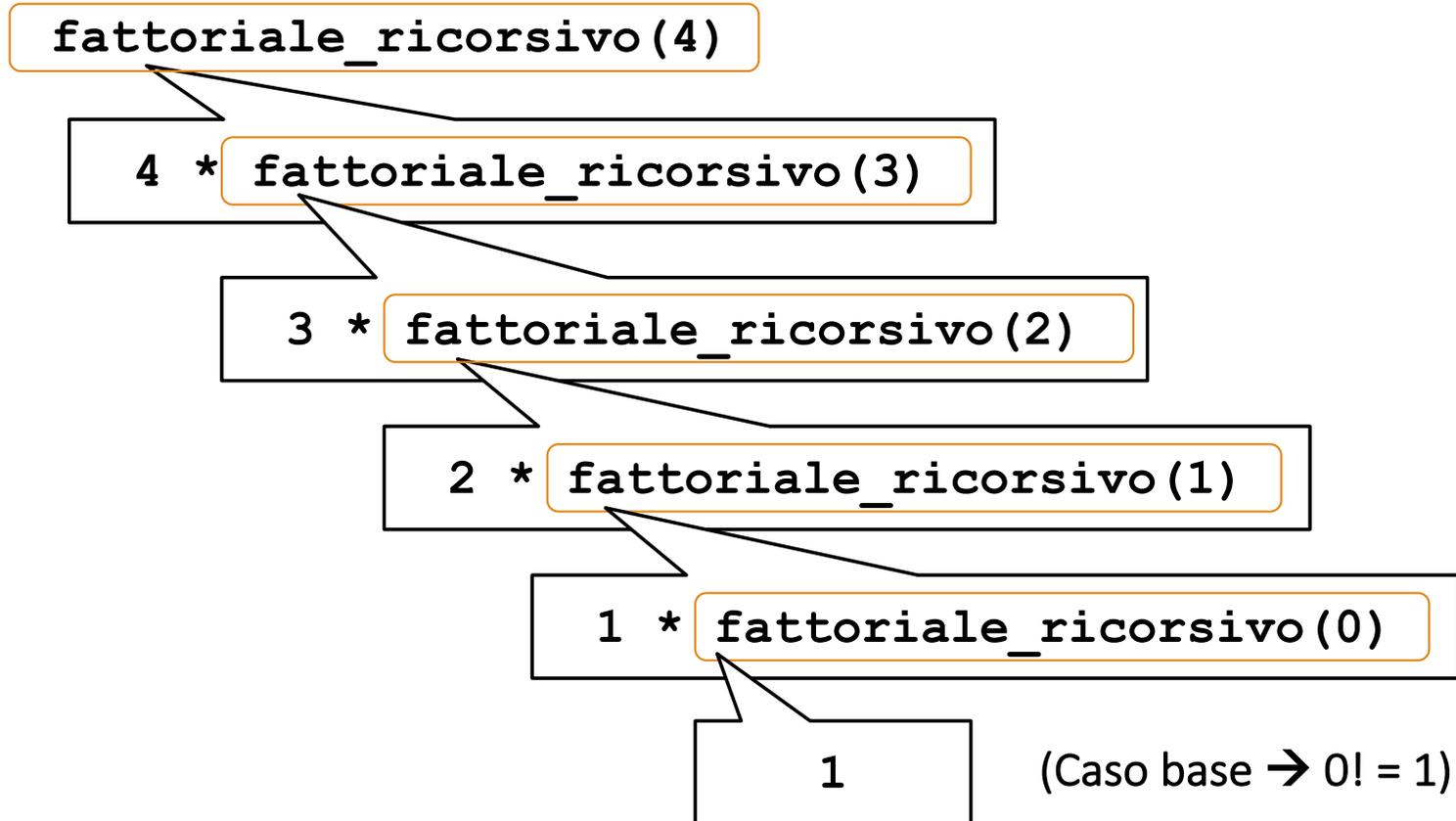
Caso Studio: Fattoriale (4)

- Dietro le quinte...



Caso Studio: Fattoriale (4)

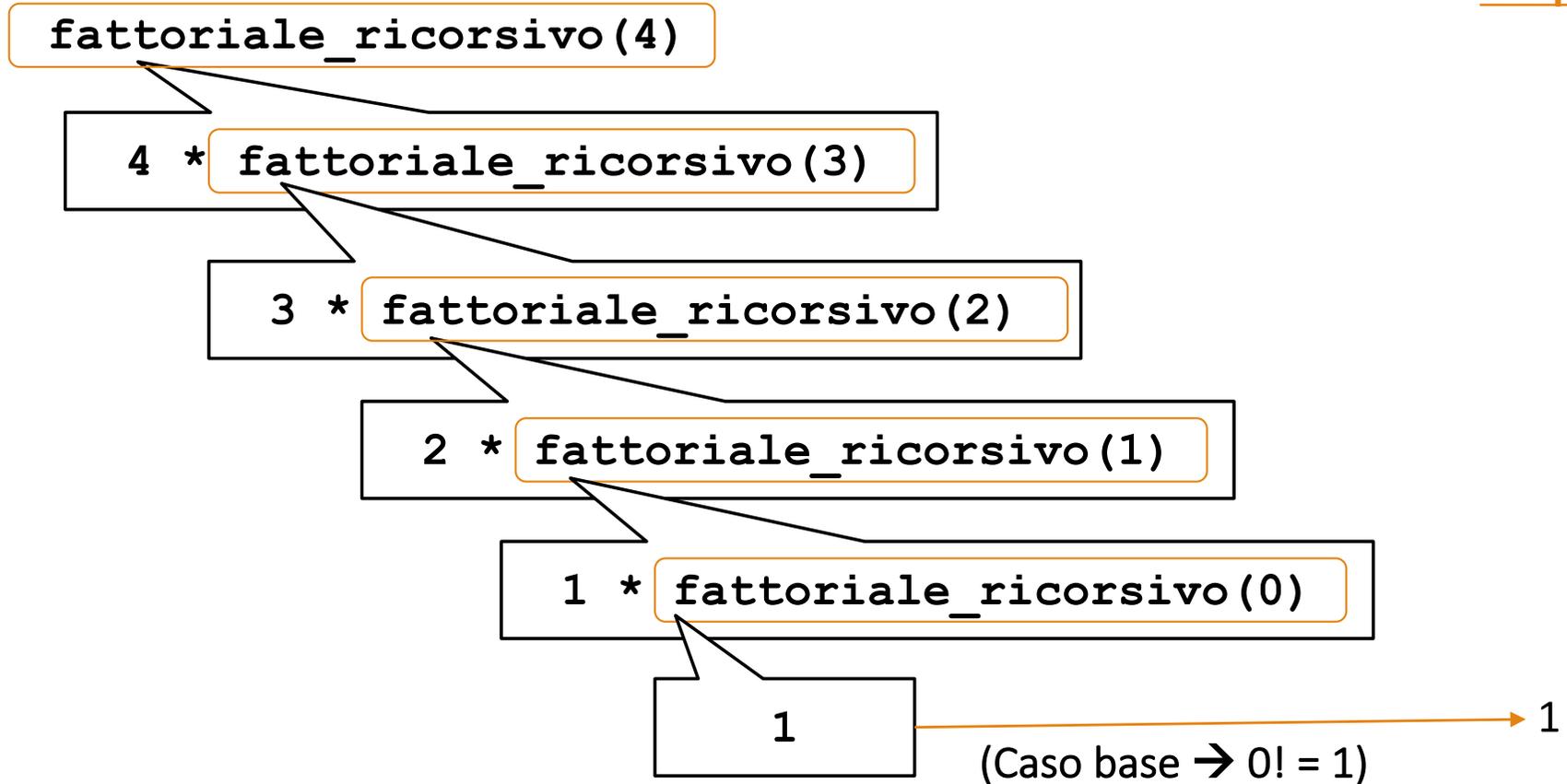
- Dietro le quinte...



Caso Studio: Fattoriale (4)

- Dietro le quinte...

Output



Caso Studio: Fattoriale (4)

- Dietro le quinte...

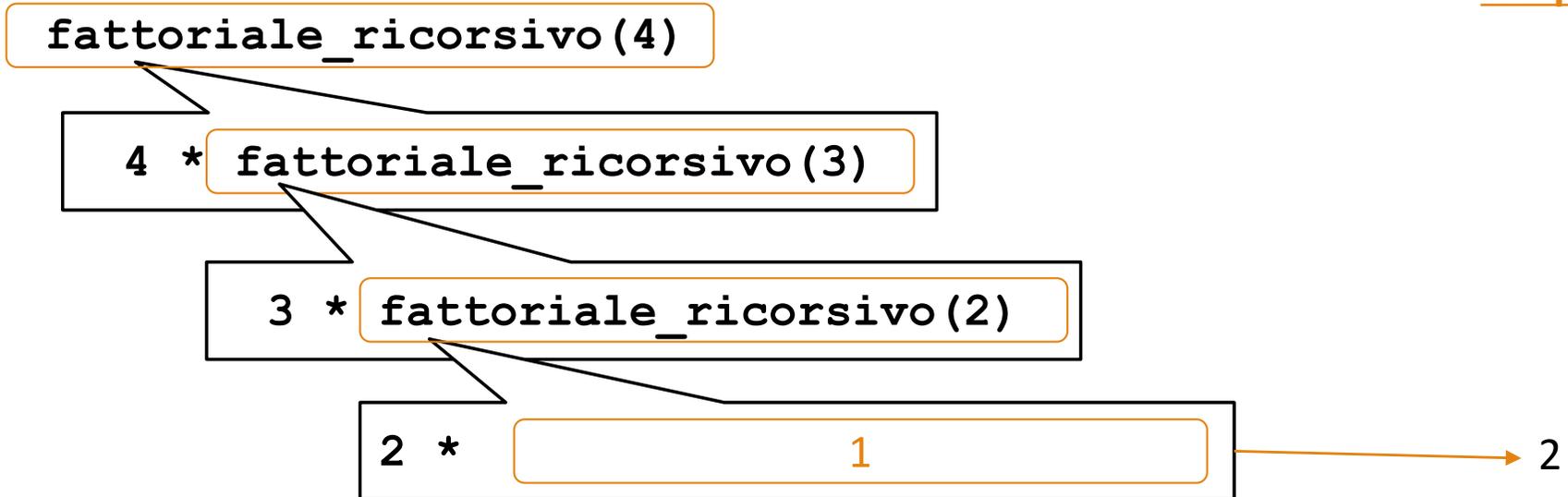
Output



Caso Studio: Fattoriale (4)

- Dietro le quinte...

Output



Caso Studio: Fattoriale (4)

- Dietro le quinte...

`fattoriale_ricorsivo(4)`



Output

Caso Studio: Fattoriale (4)

- Dietro le quinte...

```
>> f = fattoriale_ricorsivo(4)

f =
    24
```

Caso Studio: Fibonacci (1)

- Nella successione di **Fibonacci**, ogni valore $Fib(n)$ ($n \geq 0$) è espresso come la somma dei due precedenti valori
- Si noti che $Fib(0) = 0$ e $Fib(1) = 1$

0, 1, 1, 2, 3, 5, 8, 13, 21, ...

- **Definizione ricorsiva:**

$$Fib(n) = \begin{cases} 0 & \text{se } n = 0 \\ 1 & \text{se } n = 1 \\ Fib(n - 1) + Fib(n - 2) & \text{se } n \geq 2 \end{cases}$$

Caso Studio: Fibonacci (2)

- **Definizione ricorsiva:**

$$Fib(n) = \begin{cases} 0 & se\ n = 0 \\ 1 & se\ n = 1 \\ Fib(n - 1) + Fib(n - 2) & se\ n \geq 2 \end{cases}$$

Caso Studio: Fibonacci (2)

- **Definizione ricorsiva:**

$$Fib(n) = \begin{cases} 0 & \text{se } n = 0 \\ 1 & \text{se } n = 1 \\ Fib(n - 1) + Fib(n - 2) & \text{se } n \geq 2 \end{cases}$$

Casi base

Caso Studio: Fibonacci (2)

- **Definizione ricorsiva:**

$$Fib(n) = \begin{cases} 0 & \text{se } n = 0 \\ 1 & \text{se } n = 1 \\ Fib(n - 1) + Fib(n - 2) & \text{se } n \geq 2 \end{cases}$$

Versione più semplice
della definizione

Caso Studio: Fibonacci (3)

- Codice MATLAB

$$Fib(n) = \begin{cases} 0 & \text{se } n = 0 \\ 1 & \text{se } n = 1 \\ Fib(n-1) + Fib(n-2) & \text{se } n \geq 2 \end{cases}$$

```
function [valore] = fibonacci_ricorsivo(n)
    if n == 0
        valore = 0;
    elseif n == 1;
        valore = 1;
    else
        valore = fibonacci_ricorsivo(n - 1) + fibonacci_ricorsivo(n - 2);
    end
end
```

Caso Studio: Fibonacci (3)

- Codice MATLAB

$$Fib(n) = \begin{cases} 0 & \text{se } n = 0 \\ 1 & \text{se } n = 1 \\ Fib(n-1) + Fib(n-2) & \text{se } n \geq 2 \end{cases}$$

```
function [valore] = fibonacci_ricorsivo(n)
    if n == 0
        valore = 0;
    elseif n == 1;
        valore = 1;
    else
        valore = fibonacci_ricorsivo(n - 1) + fibonacci_ricorsivo(n - 2);
    end
end
```

Invocazioni ricorsive

Caso Studio: Fibonacci (3)

- Codice MATLAB

```
function [valore] = fibonacci_ricorsivo(n)
    if n == 0
        valore = 0;
    elseif n == 1;
        valore = 1;
    else
        valore = fibonacci_ricorsivo(n - 1) + fibonacci_ricorsivo(n - 2);
    end
end
```

- *Esempi d'uso*

```
>> fibonacci_ricorsivo(0)
ans =
    0
>> fibonacci_ricorsivo(4)
ans =
    3
>> fibonacci_ricorsivo(8)
ans =
   21
```

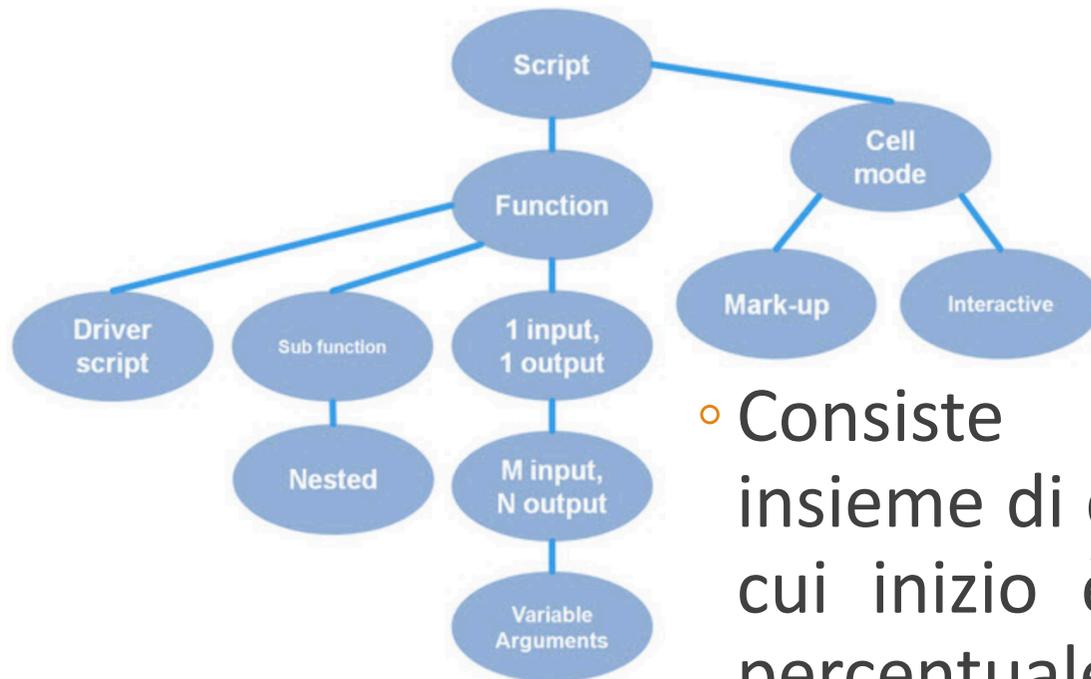
Quando usare la ricorsione

- La ricorsione deve essere evitata quando esiste una soluzione iterativa ovvia, e in situazioni in cui le prestazioni del sistema sono un elemento critico.
- Algoritmi che per loro natura sono ricorsivi piuttosto che iterativi dovrebbero essere formulati con procedure ricorsive. Ad esempio, considerare che alcune strutture dati sono inerentemente ricorsive:
 - Strutture ad albero
 - Sequenze
 -

Debugging (1)

- Il debugging (o semplicemente debug), in informatica, indica l'attività che consiste nell'individuazione da parte del programmatore della porzione di software affetta da errore (bug).
- L'editor che mette a disposizione MATLAB per la scrittura degli M-file contiene anche un Debugger, che ha lo scopo di supportare il programmatore durante il debugging.
- Va detto che i programmi MATLAB sono generalmente brevi, e non richiedono un debugger a meno non siano di grandi dimensioni.

Debugging (2)



- Una prima tecnica di debugging è la modalità cella, da usare con gli script.
- Consiste nel raggruppare un insieme di comandi in una cella, il cui inizio è stabilito dal doppio percentuale.
- Una volta definite le celle possono essere valutate singolarmente, passare da una cella ad un'altra e valutare l'intero programma.

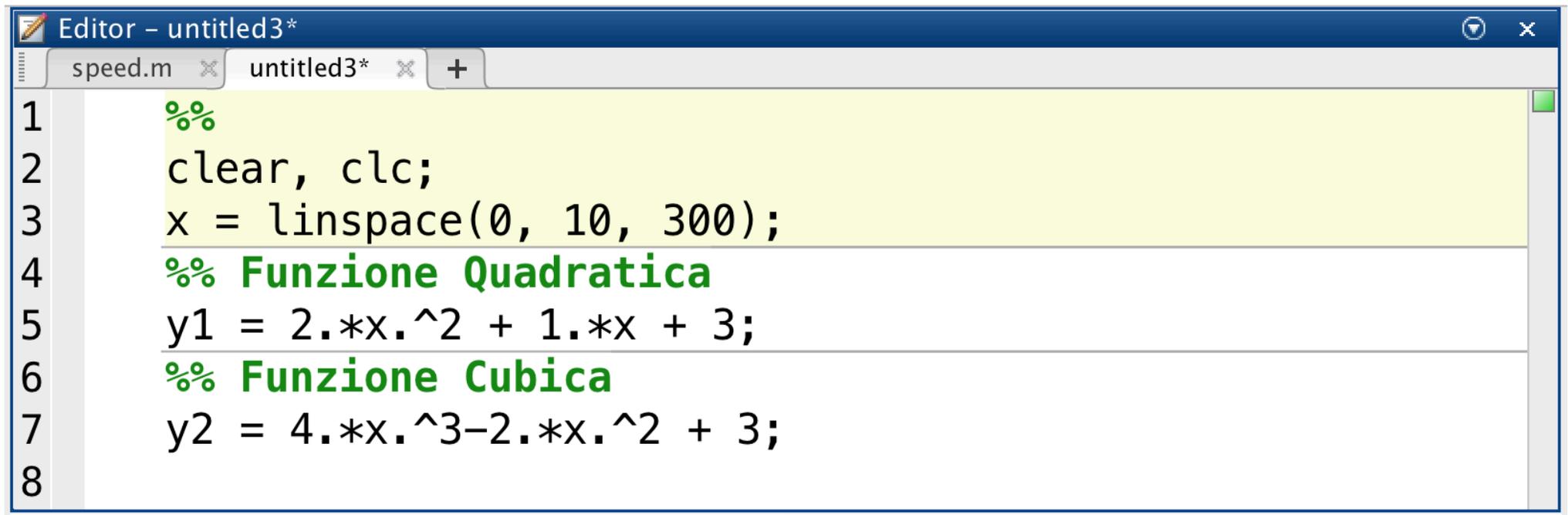
Debugging (3)

- Consideriamo il seguente esempio:

```
%%  
clear, clc;  
x = linspace(0, 10, 300);  
%% Funzione Quadratica  
y1 = 2.*x.^2 + 1.*x + 3;  
%% Funzione Cubica  
y2 = 4.*x.^3 - 2.*x.^2 + 3;
```

- Se digitiamo e salviamo questo file, possiamo notare che esso è stato suddiviso in tre porzioni.

Debugging (3)

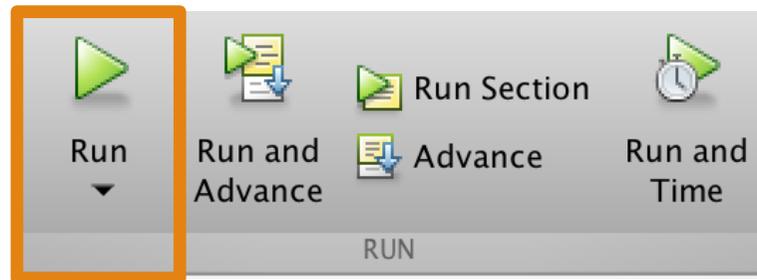


```
Editor - untitled3*
speed.m x untitled3* x +
1 %%
2 clear, clc;
3 x = linspace(0, 10, 300);
4 %% Funzione Quadratica
5 y1 = 2.*x.^2 + 1.*x + 3;
6 %% Funzione Cubica
7 y2 = 4.*x.^3-2.*x.^2 + 3;
8
```

- Se digitiamo e salviamo questo file, possiamo notare che esso è stato suddiviso in tre porzioni.

Debugging (4)

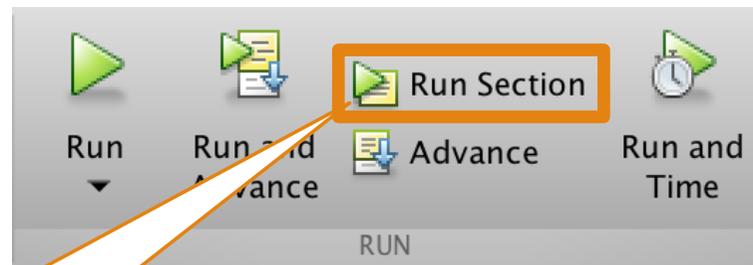
- È possibile eseguire lo script nella sua interezza, oppure sezione per sezione, o solo una determinata sezione:



Esegue tutto lo script.

Debugging (4)

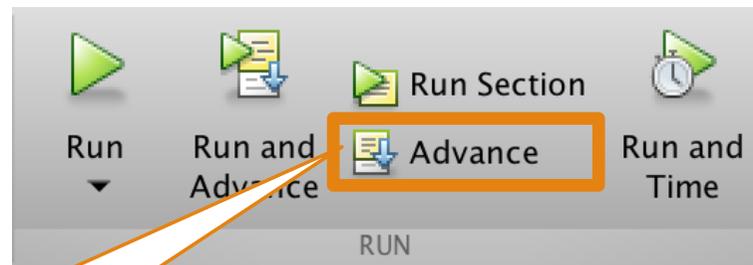
- È possibile eseguire lo script nella sua interezza, oppure sezione per sezione, o solo una determinata sezione:



Esegue solo la cella selezionata, o corrente.

Debugging (4)

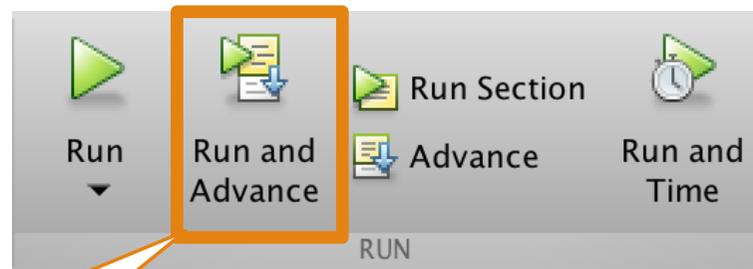
- È possibile eseguire lo script nella sua interezza, oppure sezione per sezione, o solo una determinata sezione:



Sposta la selezione alla prossima cella.

Debugging (4)

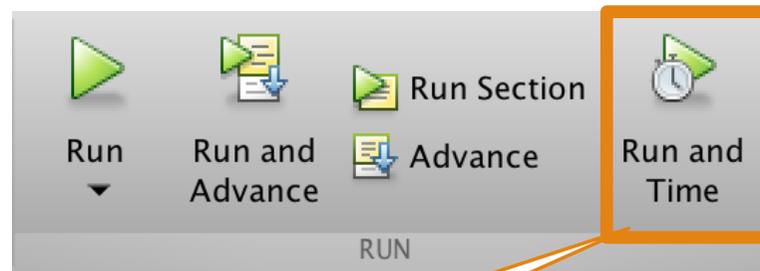
- È possibile eseguire lo script nella sua interezza, oppure sezione per sezione, o solo una determinata sezione:



Esegue la cella attualmente sezionata, e si sposta alla successiva.

Debugging (4)

- È possibile eseguire lo script nella sua interezza, oppure sezione per sezione, o solo una determinata sezione:

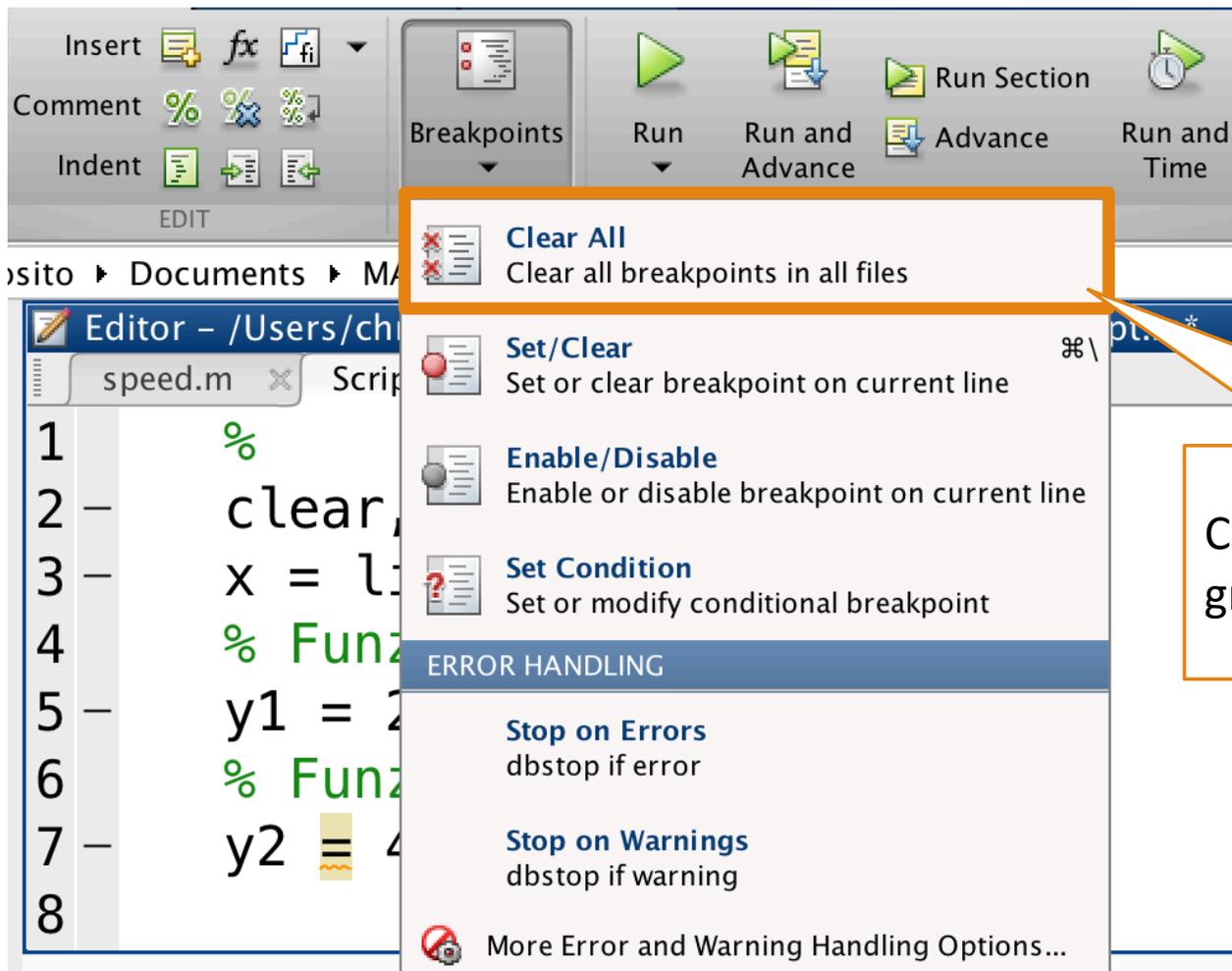


Esegue tutto lo script e misura il suo tempo di esecuzione.

Debugging (5)

- La modalità Debug è un altro meccanismo per supportare il debugging e consiste nel collocare all'interno dell'M-file i cosiddetti breakpoints:
- Un punto all'interno del programma in corrispondenza del quale l'esecuzione si interrompe temporaneamente in modo che il programmatore possa esaminare i valori correnti delle variabili.
- L'editor di MATLAB mette a disposizione un apposito menù per la gestione dei breakpoints all'interno di un programma (loro creazione e/o cancellazione).

Debugging (6)



The image shows the MATLAB Editor interface. The top toolbar contains several icons for debugging, including 'Breakpoints', 'Run', 'Run and Advance', 'Run Section', 'Advance', and 'Run and Time'. A context menu is open over the 'Breakpoints' icon, listing several options: 'Clear All', 'Set/Clear', 'Enable/Disable', 'Set Condition', and 'ERROR HANDLING'. The 'Clear All' option is highlighted with an orange box. Below the menu, the editor window shows a script named 'speed.m' with the following code:

```
1 %  
2 clear  
3 x = 1  
4 % Funz  
5 y1 = 2  
6 % Funz  
7 y2 = 4  
8
```

Cancella tutti i breakpoints nel programma.

Debugging (6)

The screenshot shows the MATLAB IDE interface. The top toolbar contains icons for 'Breakpoints', 'Run', 'Run and Advance', 'Run Section', 'Advance', and 'Run and Time'. Below the toolbar, the 'Breakpoints' menu is open, displaying the following options:

- Clear All**: Clear all breakpoints in all files
- Set/Clear**: Set or clear breakpoint on current line (highlighted with an orange box)
- Enable/Disable**: Enable or disable breakpoint on current line
- Set Condition**: Set or modify conditional breakpoint
- ERROR HANDLING**: A section header for error handling options.
- Stop on Errors**: dbstop if error
- Stop on Warnings**: dbstop if warning
- More Error and Warning Handling Options...**

The background shows a code editor with the following code:

```
1 %  
2 clear  
3 x = 1  
4 % Funz  
5 y1 = 2  
6 % Funz  
7 y2 = 4  
8
```

Inserisce o rimuove un breakpoint alla riga corrente (ovvero dove è collocato il cursore).

Debugging (6)

Insert

Comment

Indent

EDIT

Breakpoints

Run

Run and Advance

Run Section

Advance

Run and Time

osito ▶ Documents ▶ MA

Editor - /Users/ch

speed.m x Scrip

1 %

2 - clear

3 - x = 1

4 % Funz

5 - y1 = 2

6 % Funz

7 - y2 = 4

8

Clear All
Clear all breakpoints in all files

Set/Clear
Set or clear breakpoint on current line

Enable/Disable
Enable or disable breakpoint on current line

Set Condition
Set or modify conditional breakpoint

ERROR HANDLING

Stop on Errors
dbstop if error

Stop on Warnings
dbstop if warning

More Error and Warning Handling Options...

Abilita, o disabilita, il breakpoint collocato alla riga corrente. Quando un breakpoint è disabilitato, non è considerato quando il programma è eseguito.

Debugging (6)

The screenshot shows the MATLAB IDE interface. The top toolbar includes the 'Breakpoints' button, which is currently selected. A context menu is open over the 'Breakpoints' button, listing several options: 'Clear All', 'Set/Clear', 'Enable/Disable', 'Set Condition', and an 'ERROR HANDLING' section with 'Stop on Errors', 'Stop on Warnings', and 'More Error and Warning Handling Options...'. The 'Set Condition' option is highlighted with an orange border. In the background, the MATLAB Editor window shows a script named 'speed.m' with the following code:

```
1 %  
2 clear  
3 x = 1  
4 % Funz  
5 y1 = 2  
6 % Funz  
7 y2 = 4  
8
```

La considerazione di un breakpoint può essere condizionata alla verifica di una determinata condizione, dati i valori correnti delle variabili. La condizione è valutata prima dell'esecuzione della riga del breakpoint.

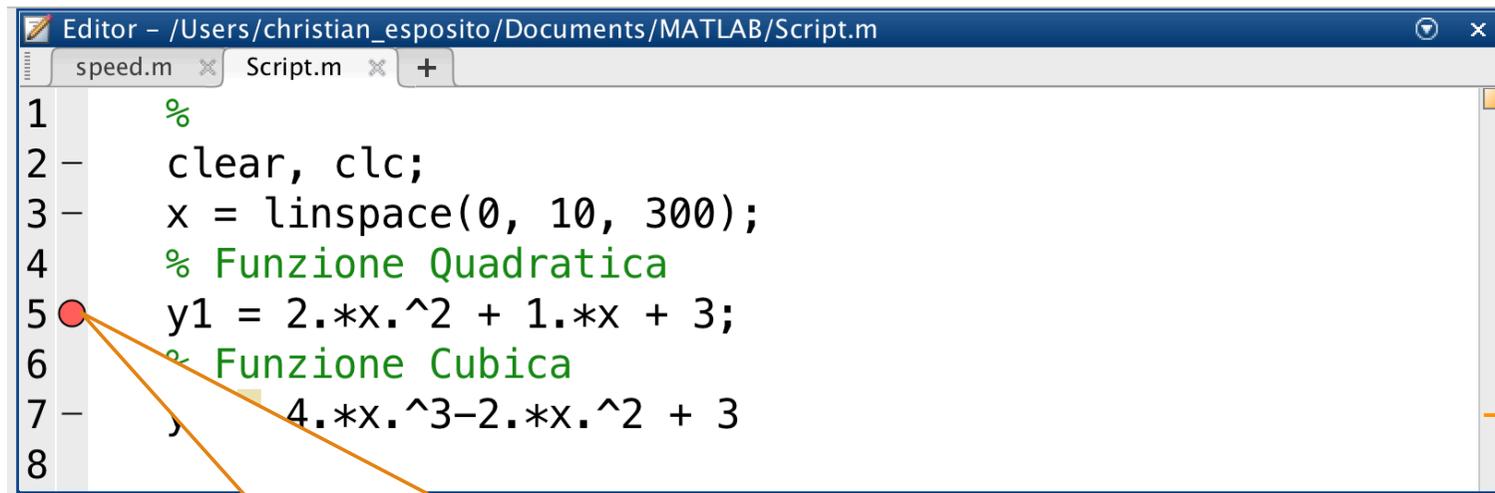
Debugging (6)

The screenshot shows the MATLAB IDE interface. The top toolbar contains icons for 'Breakpoints', 'Run', 'Run and Advance', 'Run Section', 'Advance', and 'Run and Time'. Below the toolbar, the 'Run' menu is open, displaying several options: 'Clear All', 'Set/Clear', 'Enable/Disable', and 'Set Condition'. The 'ERROR HANDLING' sub-menu is also open, showing 'Stop on Errors', 'Stop on Warnings', and 'More Error and Warning Handling Options...'. The background shows a code editor with MATLAB code for a function named 'speed.m'.

```
1 %  
2 clear  
3 x = l  
4 % Funz  
5 y1 = 2  
6 % Funz  
7 y2 = 4  
8
```

Strategie di gestione di eventuali errori di programmazione quando l'M-file viene eseguito.

Debugging (7)



The screenshot shows the MATLAB Editor window with the following code:

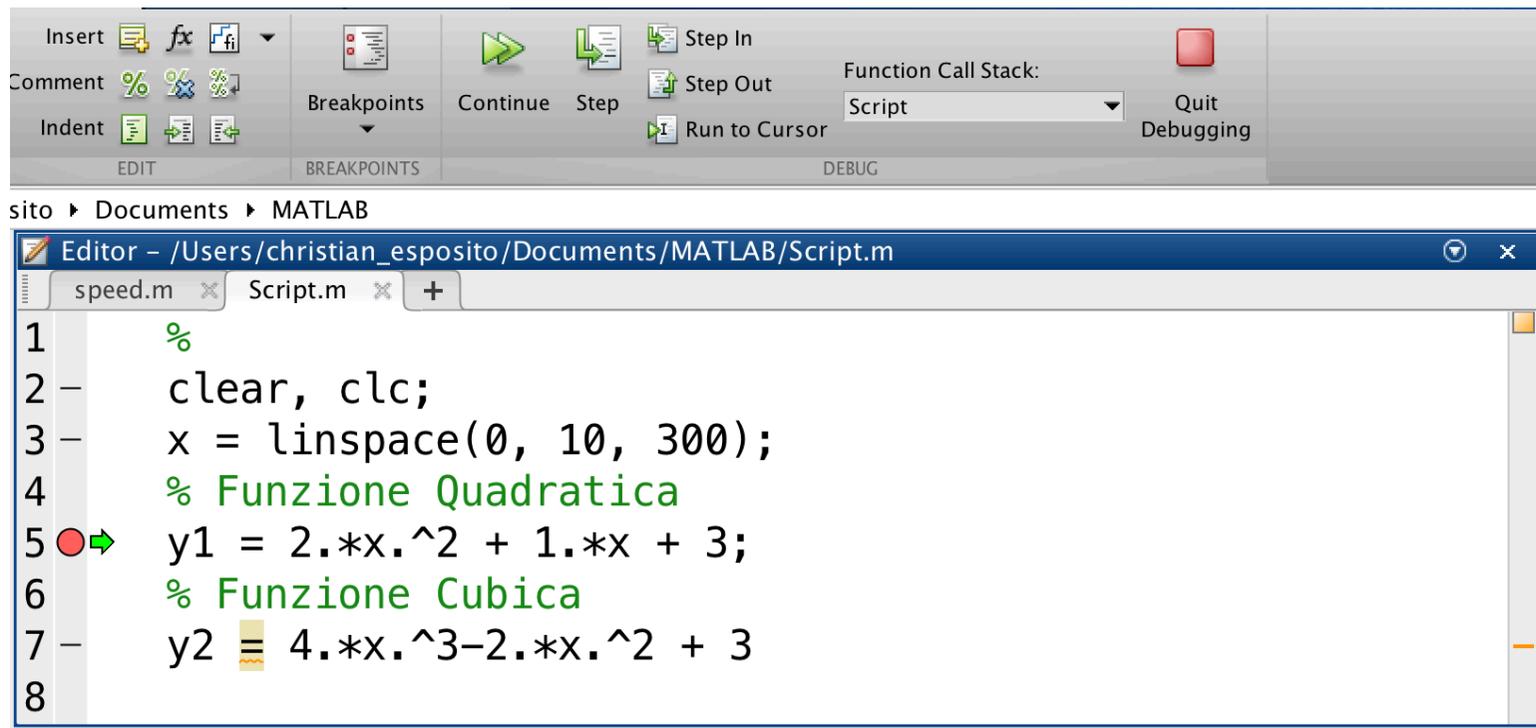
```
1 %  
2 clear, clc;  
3 x = linspace(0, 10, 300);  
4 % Funzione Quadratica  
5 y1 = 2.*x.^2 + 1.*x + 3;  
6 % Funzione Cubica  
7 y2 = 4.*x.^3 - 2.*x.^2 + 3;  
8
```

A red dot is placed on the left margin of line 5, indicating a breakpoint. An orange box highlights this red dot, with a line pointing to the explanatory text below.

Quando un breakpoint è collocato in una riga, l'editor lo segnala con un pallino rosso.

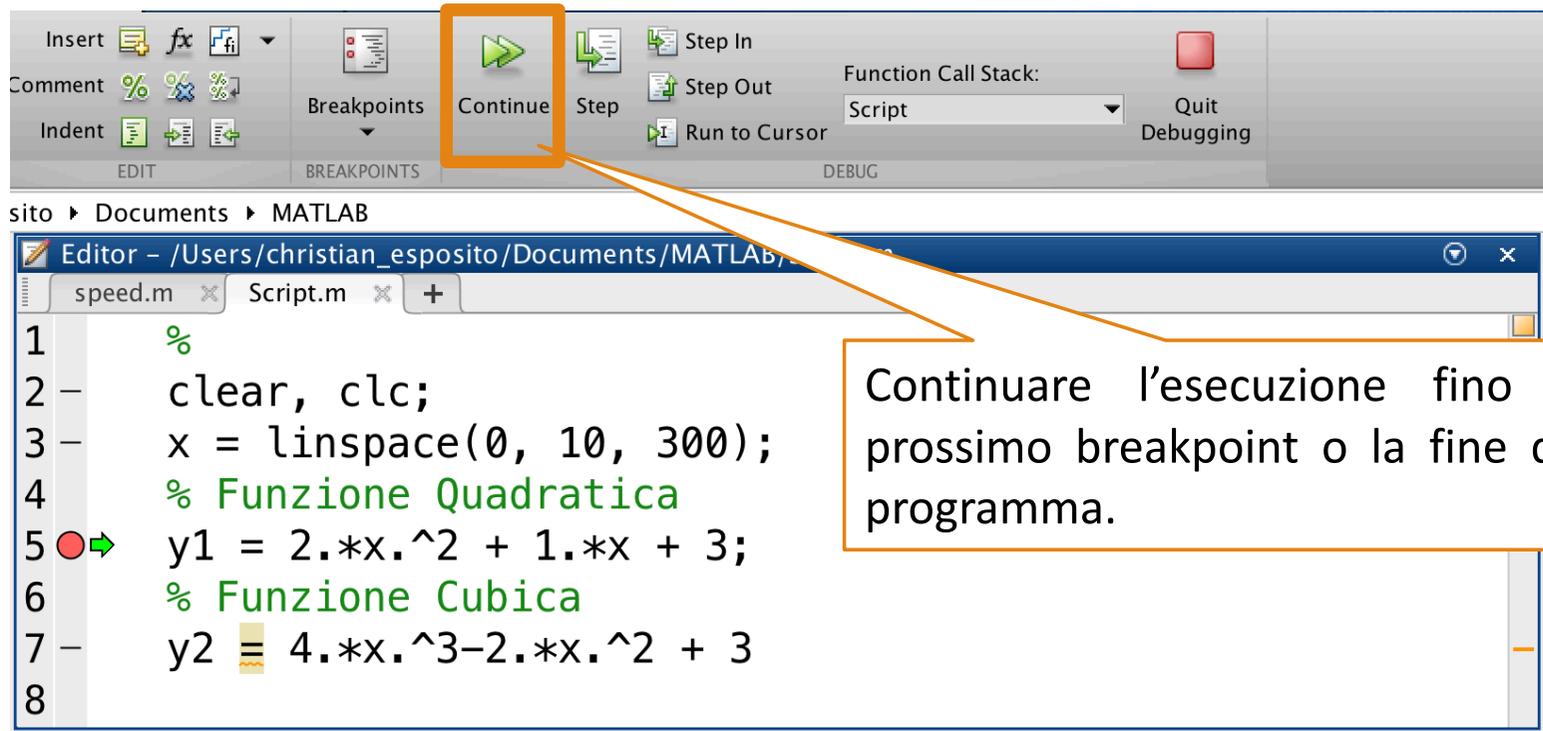
- Se scegliamo di eseguire lo script vediamo che al punto del breakpoint si ha un'interruzione.

Debugging (7)



- Appare un menù così che il programmatore può scegliere cosa fare.

Debugging (7)



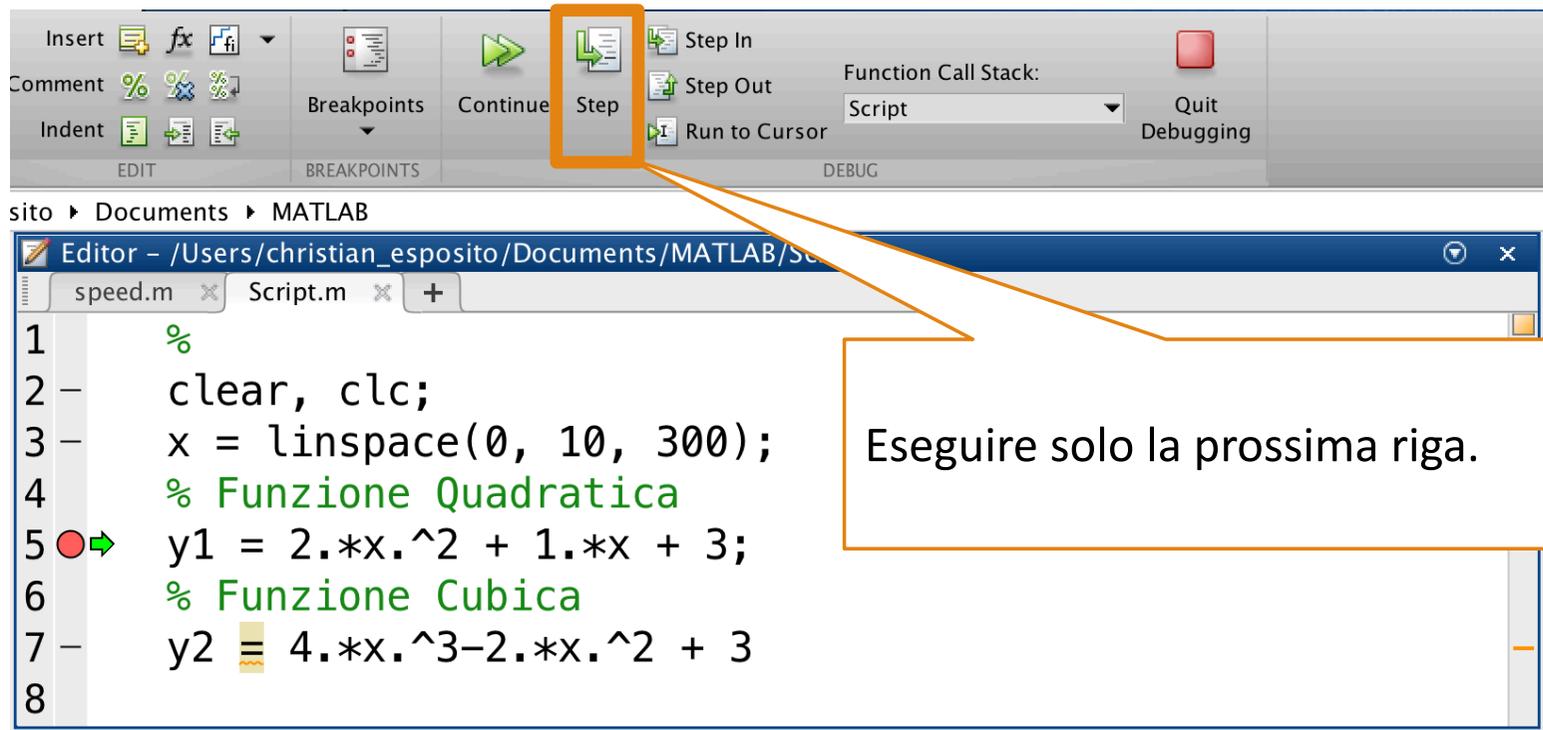
The screenshot shows the MATLAB debugging toolbar with the 'Continue' button (a green right-pointing arrow) highlighted with an orange box. Below the toolbar, the MATLAB Editor window is open, showing a script named 'Script.m' with the following code:

```
1 %  
2 clear, clc;  
3 x = linspace(0, 10, 300);  
4 % Funzione Quadratica  
5 y1 = 2.*x.^2 + 1.*x + 3;  
6 % Funzione Cubica  
7 y2 = 4.*x.^3 - 2.*x.^2 + 3;  
8
```

A callout box with an orange border points to the 'Continue' button and contains the text: "Continuare l'esecuzione fino al prossimo breakpoint o la fine del programma."

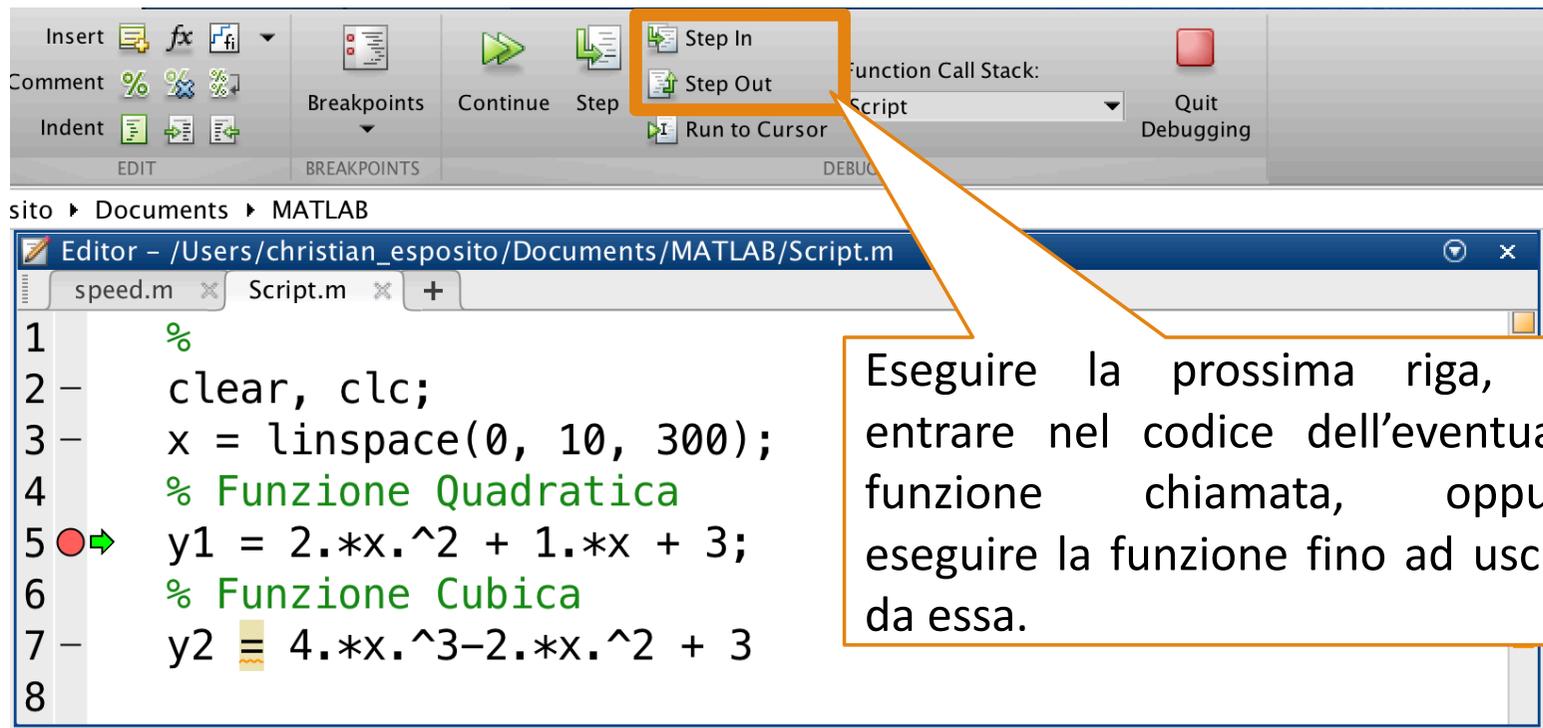
- Appare un menù così che il programmatore può scegliere cosa fare.

Debugging (7)



- Appare un menù così che il programmatore può scegliere cosa fare.

Debugging (7)



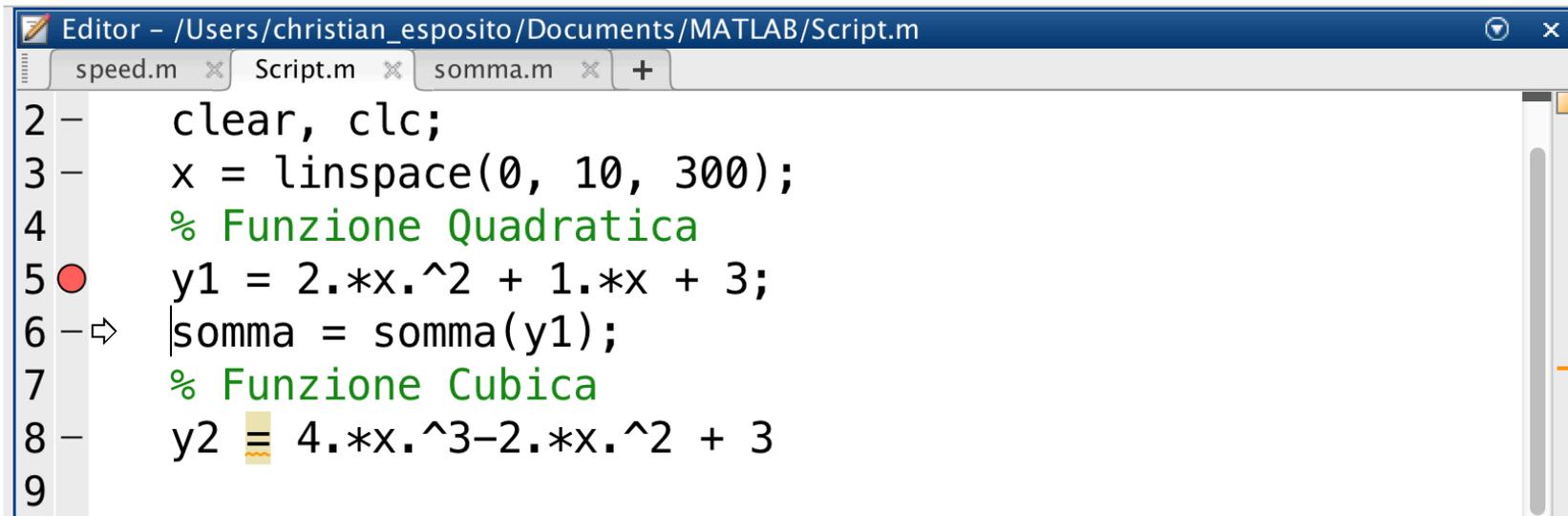
The screenshot shows the MATLAB IDE interface. The top toolbar contains several debugging icons: 'Continue' (green play button), 'Step' (green play button with a vertical line), 'Step In' (green play button with a vertical line and a downward arrow), 'Step Out' (green play button with a vertical line and an upward arrow), and 'Run to Cursor' (green play button with a vertical line and a cursor). The 'Step In' and 'Step Out' icons are highlighted with an orange box. Below the toolbar, the 'Function Call Stack' is visible, showing 'Script'. A callout box points to the 'Step In' and 'Step Out' icons with the text: 'Eseguire la prossima riga, ed entrare nel codice dell'eventuale funzione chiamata, oppure eseguire la funzione fino ad uscire da essa.'

Below the toolbar, the MATLAB editor window is open, showing the following code:

```
1 %  
2 clear, clc;  
3 x = linspace(0, 10, 300);  
4 % Funzione Quadratica  
5 y1 = 2.*x.^2 + 1.*x + 3;  
6 % Funzione Cubica  
7 y2 = 4.*x.^3 - 2.*x.^2 + 3;  
8
```

- Appare un menù così che il programmatore può scegliere cosa fare.

Debugging (7)



```
Editor - /Users/christian_esposito/Documents/MATLAB/Script.m
speed.m x Script.m x somma.m x +
2 - clear, clc;
3 - x = linspace(0, 10, 300);
4   % Funzione Quadratica
5 ● y1 = 2.*x.^2 + 1.*x + 3;
6 -> somma = somma(y1);
7   % Funzione Cubica
8 - y2 = 4.*x.^3-2.*x.^2 + 3
9
```

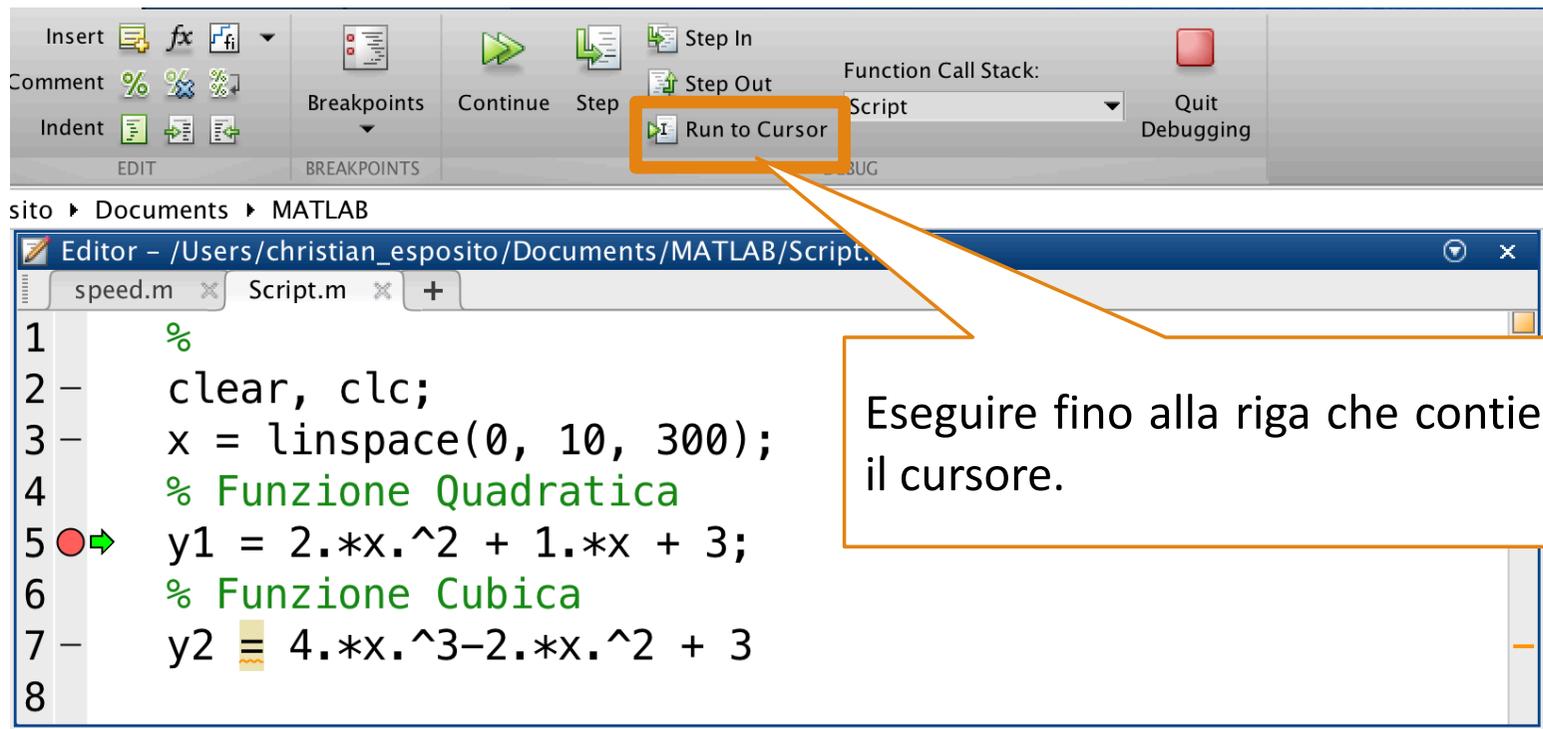
Debugging (7)

```
Editor - /Users/christian_esposito/Documents/MATLAB/Script.m
speed.m x Script.m x somma.m x +
2 - clear, clc;
3 - x = linspace(0, 10, 300);
4 - % Funzione Quadratica
5 ● y1 = 2.*x.^2 + 1.*x + 3;
6 -> somma = somma(y1);
7 - % Funzione Cubica

Editor - /Users/christian_esposito/Documents/MATLAB/somma.m
speed.m x Script.m x somma.m x +
1 - function somma = somma( x )
2 -     l = length(x);
3 ->     somma = 0;
4 -     for i = 1:l
5 -         somma = somma+x(i);
6 -     end
7 - end
8 -
```

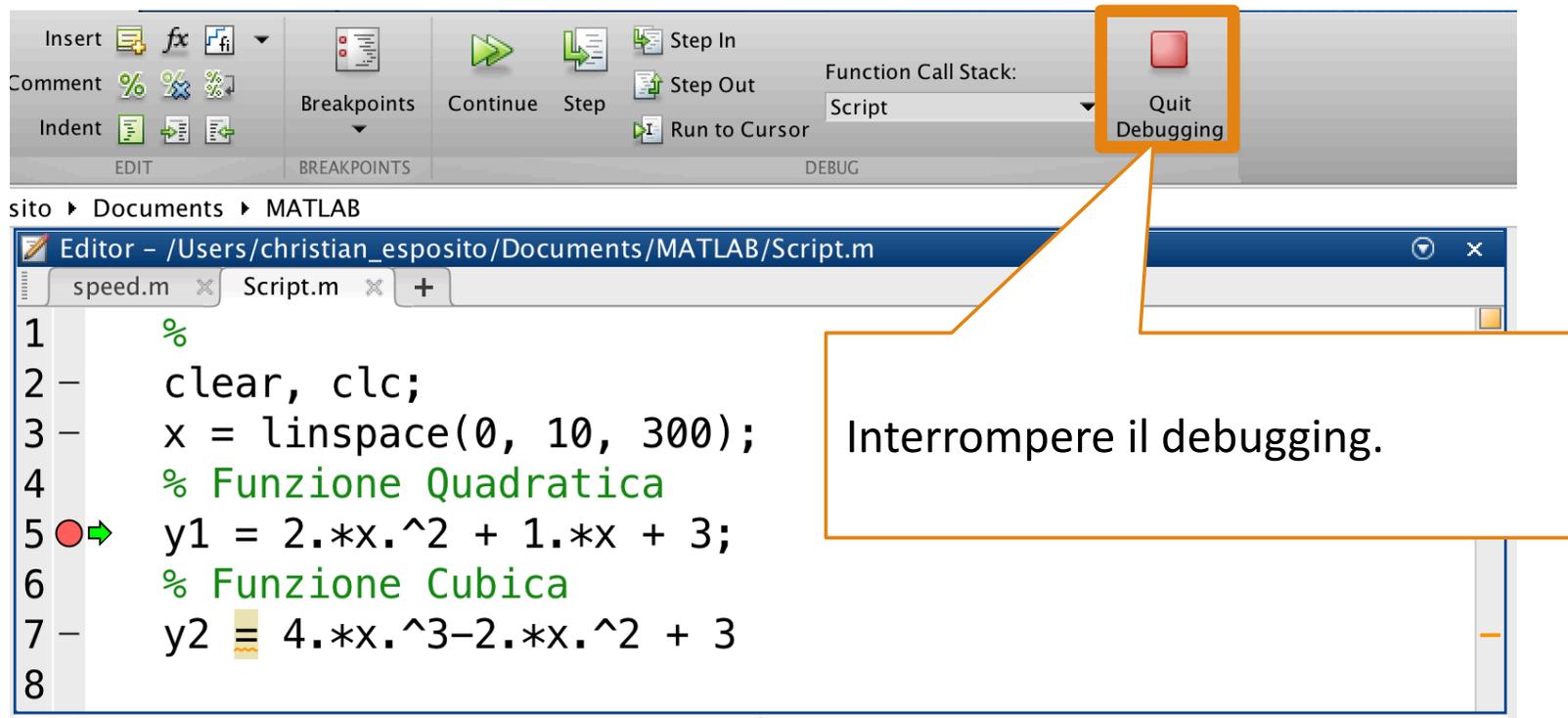
Step

Debugging (8)



- Appare un menù così che il programmatore può scegliere cosa fare.

Debugging (9)



- Appare un menù così che il programmatore può scegliere cosa fare.

Riferimenti

- Capitolo 4
 - Paragrafi 8 [**Debugging dei programmi di Matlab**].