

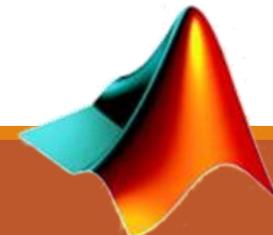
UNIVERSITÀ DEGLI STUDI DI SALERNO

Fondamenti di Informatica

Cenni di Debugging in MATLAB

Prof. Raffaele Pizzolante

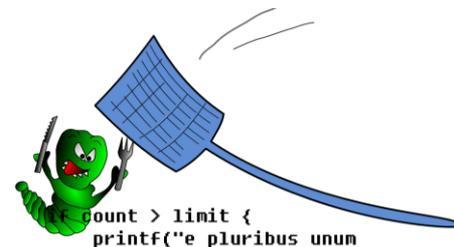
A.A. 2016/17



MATLAB®

Debugging – 1/7

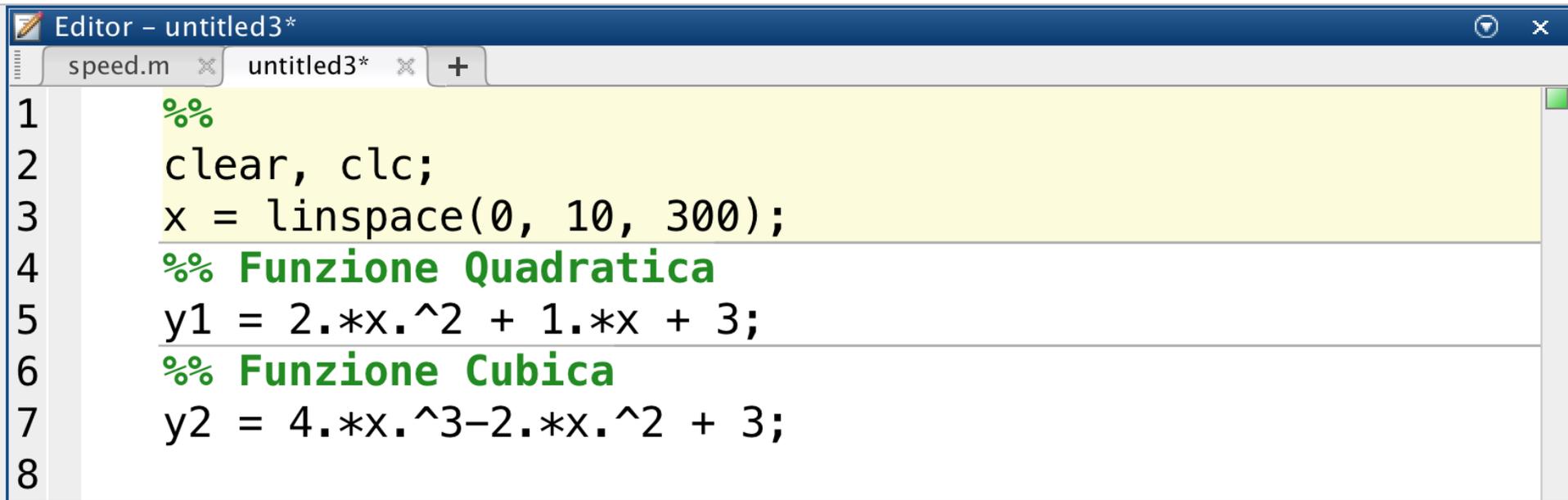
- Il **debugging** (o semplicemente debug) è l'attività che permette al programmatore di individuare la porzione di programma (codice sorgente) in cui è contenuto un errore o **bug** (bug letteralmente significa **insetto**), al fine di correggerlo
- MATLAB fornisce un *Debugger*
 - Strumento per supportare il programmatore durante la fase di debugging (ovvero per l'individuazione di errori/bug)



Debugging – 2/7

- MATLAB permette di isolare e valutare singolarmente *sezioni (celle)* di codice, al fine di restringere la ricerca di eventuali bug soltanto a tali sezioni
 - Questa tecnica viene detta modalità cella
- Una *sezione (o cella)* di codice valutata dal debugger inizia con %%
 - **N.B.**: Da non confondere con i **commenti**, che iniziano con %

Debugging – 3/7

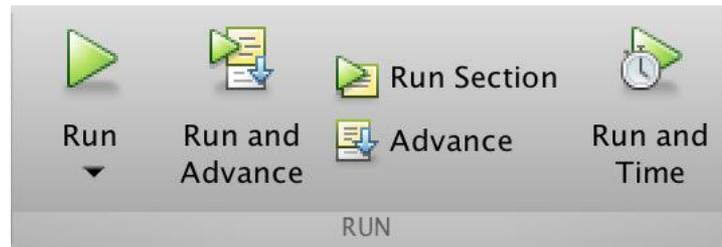


```
Editor - untitled3*
speed.m x untitled3* x +
1 %%
2 clear, clc;
3 x = linspace(0, 10, 300);
4 %% Funzione Quadratica
5 y1 = 2.*x.^2 + 1.*x + 3;
6 %% Funzione Cubica
7 y2 = 4.*x.^3-2.*x.^2 + 3;
8
```

- Questa porzione di codice è composta da **tre sezioni**
 - MATLAB separa visivamente ogni sezione mediante linee orizzontali

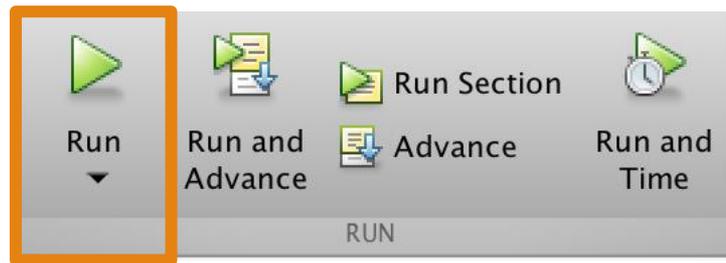
Debugging – 4/7

- È possibile eseguire la porzione di codice
 - Interamente
 - Sezione per sezione
 - Solo una determinata sezione



Debugging – 4/7

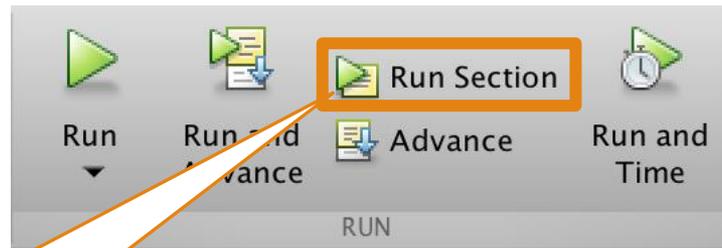
- È possibile eseguire la porzione di codice
 - Interamente
 - Sezione per sezione
 - Solo una determinata sezione



Esegue l'intera porzione di codice

Debugging – 4/7

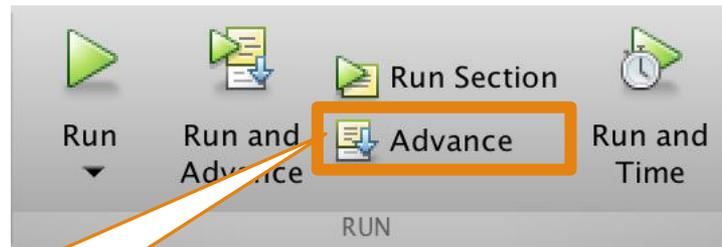
- È possibile eseguire la porzione di codice
 - Interamente
 - Sezione per sezione
 - Solo una determinata sezione



Esegue solo la sezione selezionata, o corrente

Debugging – 4/7

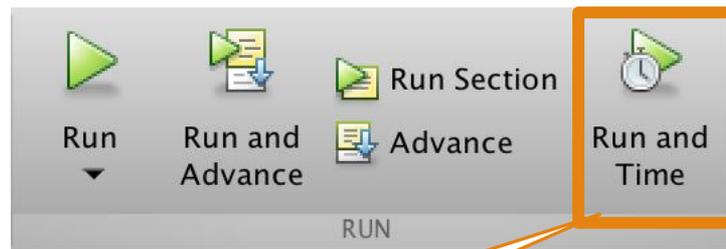
- È possibile eseguire la porzione di codice
 - Interamente
 - Sezione per sezione
 - Solo una determinata sezione



Sposta la selezione alla prossima sezione

Debugging – 4/7

- È possibile eseguire la porzione di codice
 - Interamente
 - Sezione per sezione
 - Solo una determinata sezione



Esegue tutto il codice e misura il relativo tempo di esecuzione

Debugging – 5/7

- La **modalità debug** è un altro meccanismo per supportare il debugging di codice MATLAB
 - Consiste nel collocare dei *breakpoints* (letteralmente *punti di rottura*) all'interno di tale codice
- Un *breakpoint* è un punto (riga di codice) in cui l'esecuzione del programma si interrompe temporaneamente (una sorta di pausa)
 - In tal modo è possibile esaminare i valori correnti (ed intermedi) delle variabili, semplificando l'individuazione di eventuali bug

Debugging – 6/7

L'editor di MATLAB fornisce un apposito menù per la gestione dei *breakpoints* all'interno di un dato programma

The screenshot shows the MATLAB editor interface. The top toolbar contains the 'Breakpoints' menu, which is highlighted with an orange box. The 'Breakpoints' menu is also highlighted with an orange box and contains the following options:

- 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

Below these options is a section titled 'ERROR HANDLING' with the following options:

- Stop on Errors**: dbstop if error
- Stop on Warnings**: dbstop if warning
- More Error and Warning Handling Options...**

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
```

Debugging – 6/7

The screenshot shows the MATLAB Editor 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/Clear' option is highlighted with an orange border. In the background, 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
```

- Inserisce o rimuove un *breakpoint* alla riga corrente
 - Ciascuna riga è selezionata mediante il cursore del mouse

Debugging – 6/7

The screenshot displays the MATLAB IDE interface. At the top, the 'Breakpoints' menu is open, showing options: 'Clear All', 'Set/Clear', 'Enable/Disable', 'Set Condition', and 'ERROR HANDLING'. The 'Enable/Disable' option is highlighted with an orange box. Below the menu, the code editor shows the following code:

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

- Abilita (o disabilita) il *breakpoint* collocato alla riga corrente
 - Quando un *breakpoint* è disabilitato, esso è ignorato durante l'esecuzione del programma

Debugging – 6/7

The screenshot displays the MATLAB IDE interface. At the top, the 'EDIT' tab is active, showing various editing tools like 'Insert', 'Comment', and 'Indent'. Below this, the 'Breakpoints' menu is open, listing several options: 'Clear All', 'Set/Clear', 'Enable/Disable', 'Set Condition', and 'ERROR HANDLING'. The 'Set Condition' option is highlighted with an orange box. The background shows a code editor with the following MATLAB code:

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

- Un *breakpoint* può essere considerato (valutato) solo al verificarsi di una determinata condizione (preimpostata), in base ai valori correnti delle variabili
 - La condizione è valutata prima dell'esecuzione dell'istruzione definita alla riga dove è stato inserito il *breakpoint*

Debugging – 6/7

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', and 'Set Condition'. Below these options is a section titled 'ERROR HANDLING' which includes 'Stop on Errors', 'Stop on Warnings', and 'More Error and Warning Handling Options...'. The background shows a code editor with a script named 'speed.m' containing MATLAB code. The code includes comments, a 'clear' command, variable assignments, and function calls.

Insert

Comment

Indent

EDIT

Breakpoints

Run

Run and Advance

Run Section

Advance

Run and Time

osito ▶ Documents ▶ MA

Editor -

speed.m x Scrip

1 %

2 – clear

3 – x = l

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...

pt.m*

MATLAB fornisce diverse strategie per la gestione di eventuali errori di programmazione, che possono presentarsi durante l'esecuzione di un programma

Debugging – 6/7

The screenshot shows the MATLAB IDE interface. At the top, there is a toolbar with various icons for editing and debugging. Below the toolbar, the 'Breakpoints' menu is open, displaying several options. The 'Clear All' option is highlighted with an orange box. The menu items are:

- 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

Below these options, there is a section for 'ERROR HANDLING' with the following options:

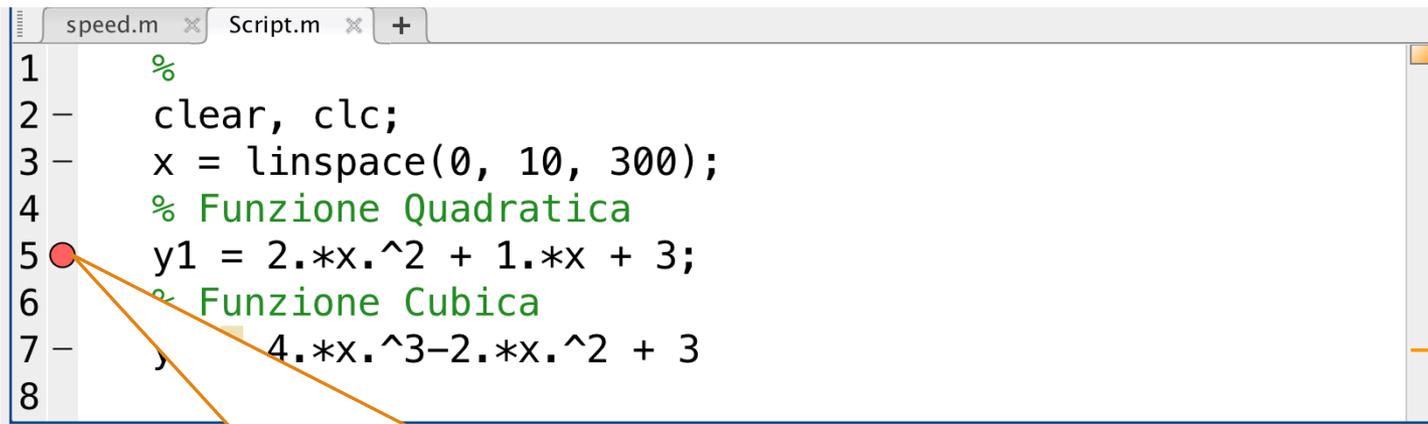
- Stop on Errors**: dbstop if error
- Stop on Warnings**: dbstop if warning
- More Error and Warning Handling Options...**

In the background, a code editor window is visible 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 – 7/7



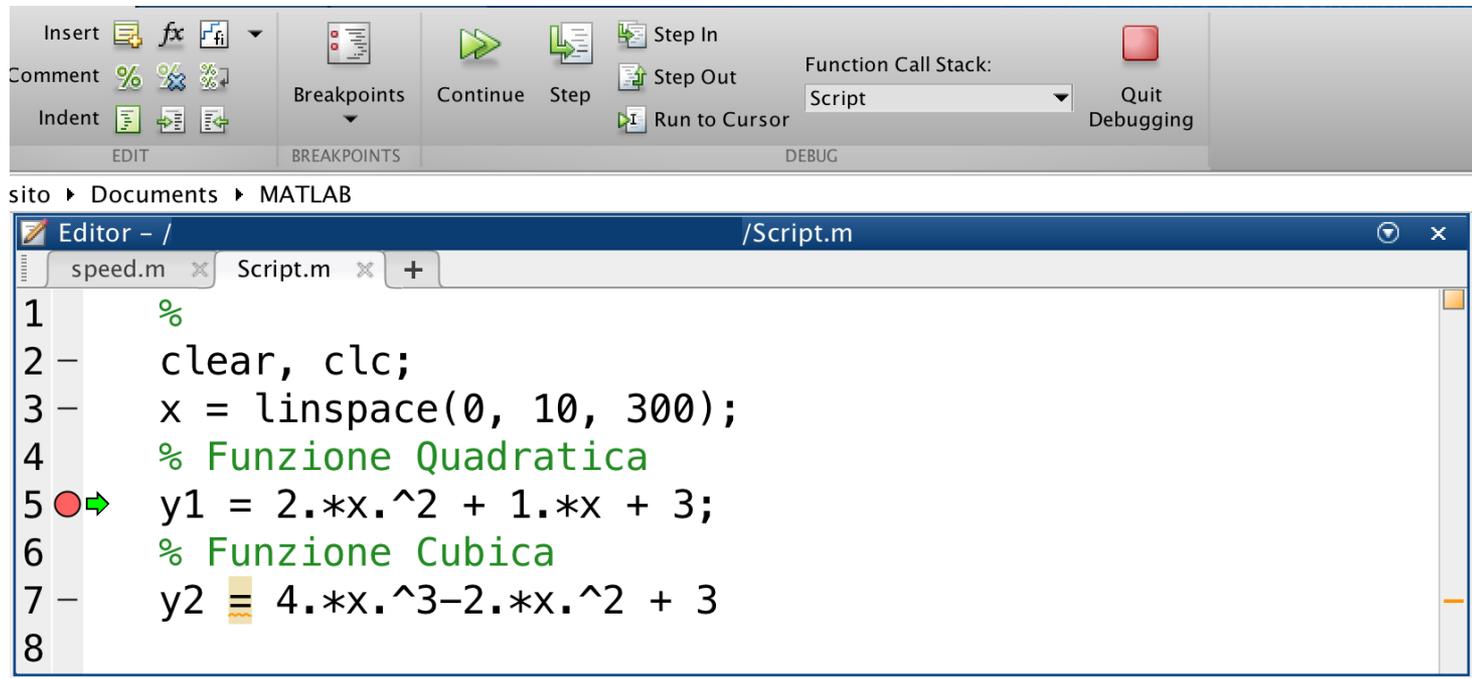
```
1      %  
2      clear, clc;  
3      x = linspace(0, 10, 300);  
4      % Funzione Quadratica  
5      y1 = 2.*x.^2 + 1.*x + 3;  
6      % Funzione Cubica  
7      y2 = 4.*x.^3 - 2.*x.^2 + 3;  
8
```

The screenshot shows a MATLAB editor window with two tabs: 'speed.m' and 'Script.m'. The script content is as follows: Line 1: %; Line 2: clear, clc; Line 3: x = linspace(0, 10, 300); Line 4: % Funzione Quadratica; Line 5: y1 = 2.*x.^2 + 1.*x + 3; Line 6: % Funzione Cubica; Line 7: y2 = 4.*x.^3 - 2.*x.^2 + 3; Line 8: (empty). A red circle is placed on the left margin of line 5, indicating a breakpoint. An orange arrow points from this red circle to a text box below the editor.

Quando è inserito un *breakpoint* in una determinata riga, l'editor lo segnala con un **pallino rosso**

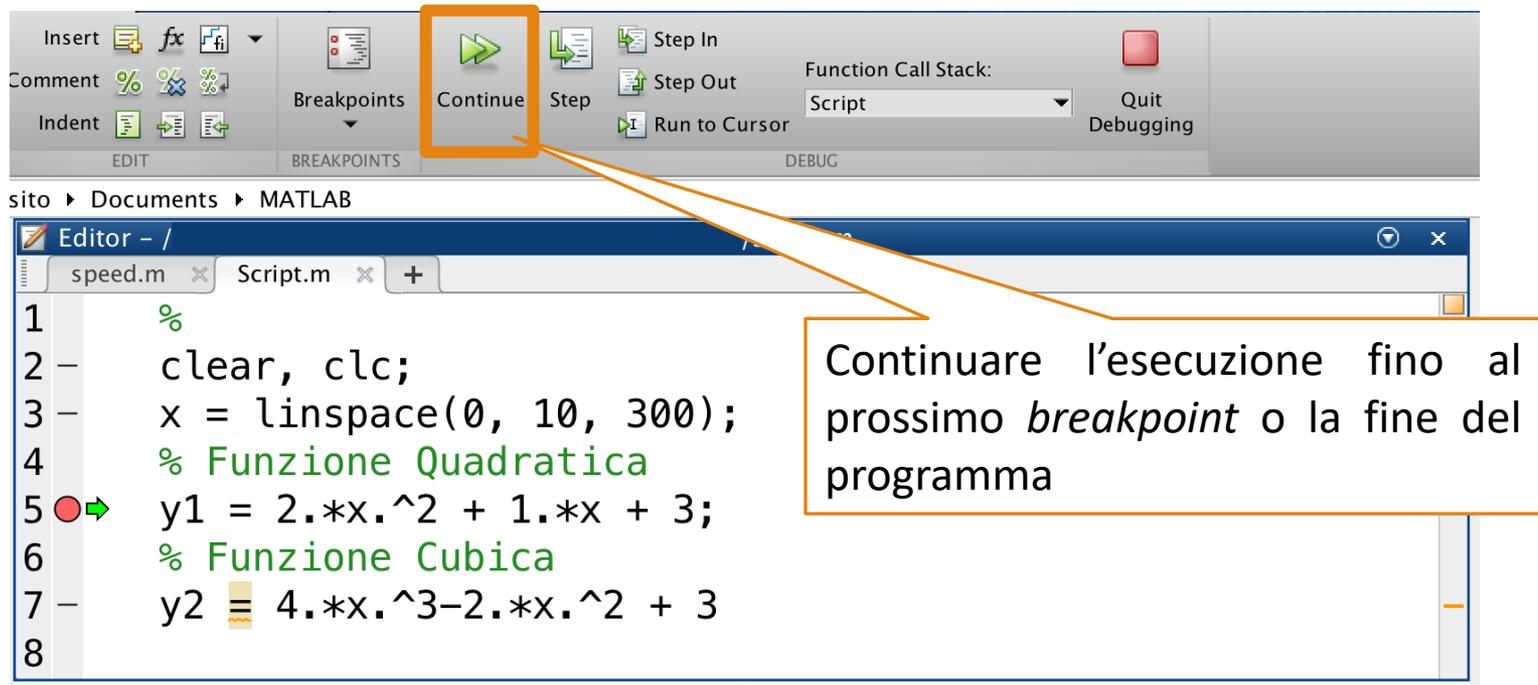
- Se scegliamo di eseguire la suddetta porzione di codice, noteremo che al punto del *breakpoint* si avrà un'interruzione temporanea del programma

Debugging – 7/7



- Quando l'esecuzione del programma si interrompe su un *breakpoint*
- MATLAB fornisce al programmatore diverse azioni da poter svolgere

Debugging – 7/7

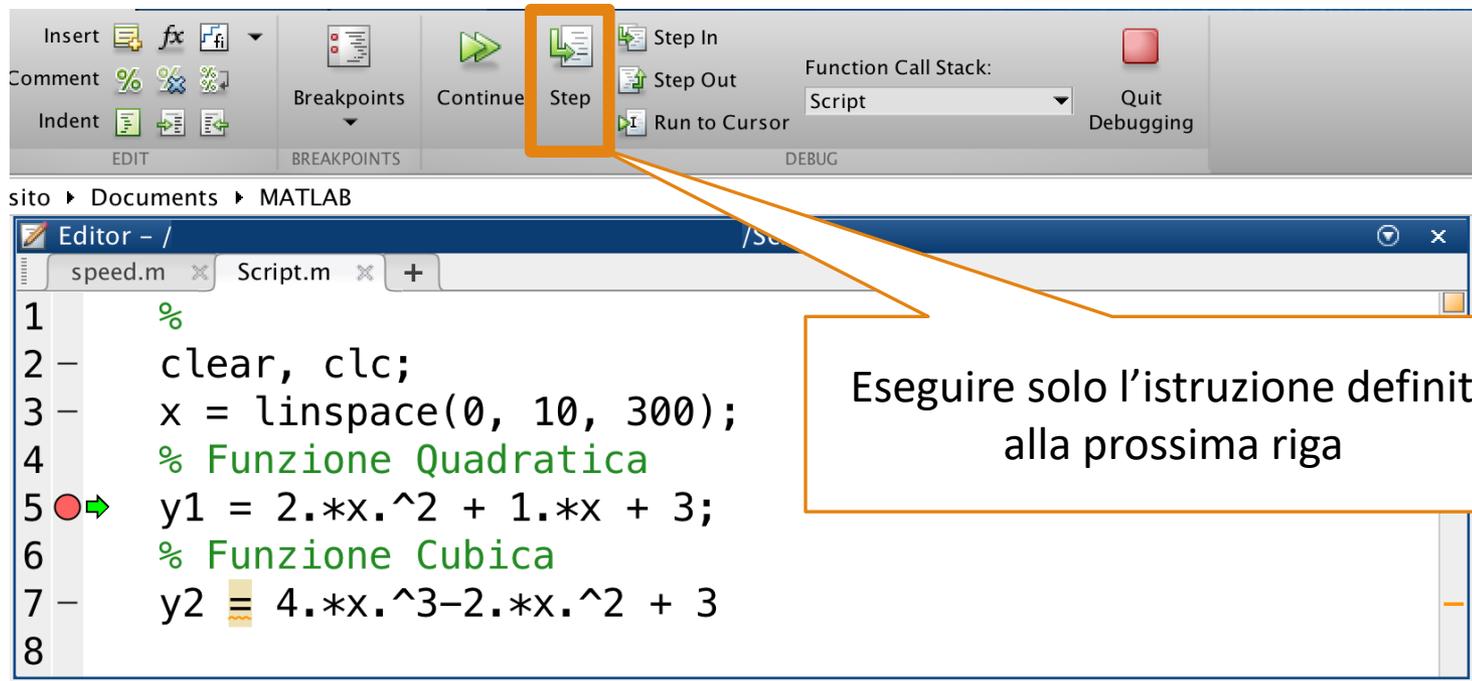


The screenshot shows the MATLAB debugging toolbar with the 'Continue' button (a green play icon) highlighted with an orange box. A callout box points to this button with the text: "Continuare l'esecuzione fino al prossimo *breakpoint* o la fine del programma". Below the toolbar, the MATLAB Editor window shows 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
```

- Quando l'esecuzione del programma si interrompe su un breakpoint
- MATLAB fornisce al programmatore diverse azioni da poter svolgere

Debugging – 7/7



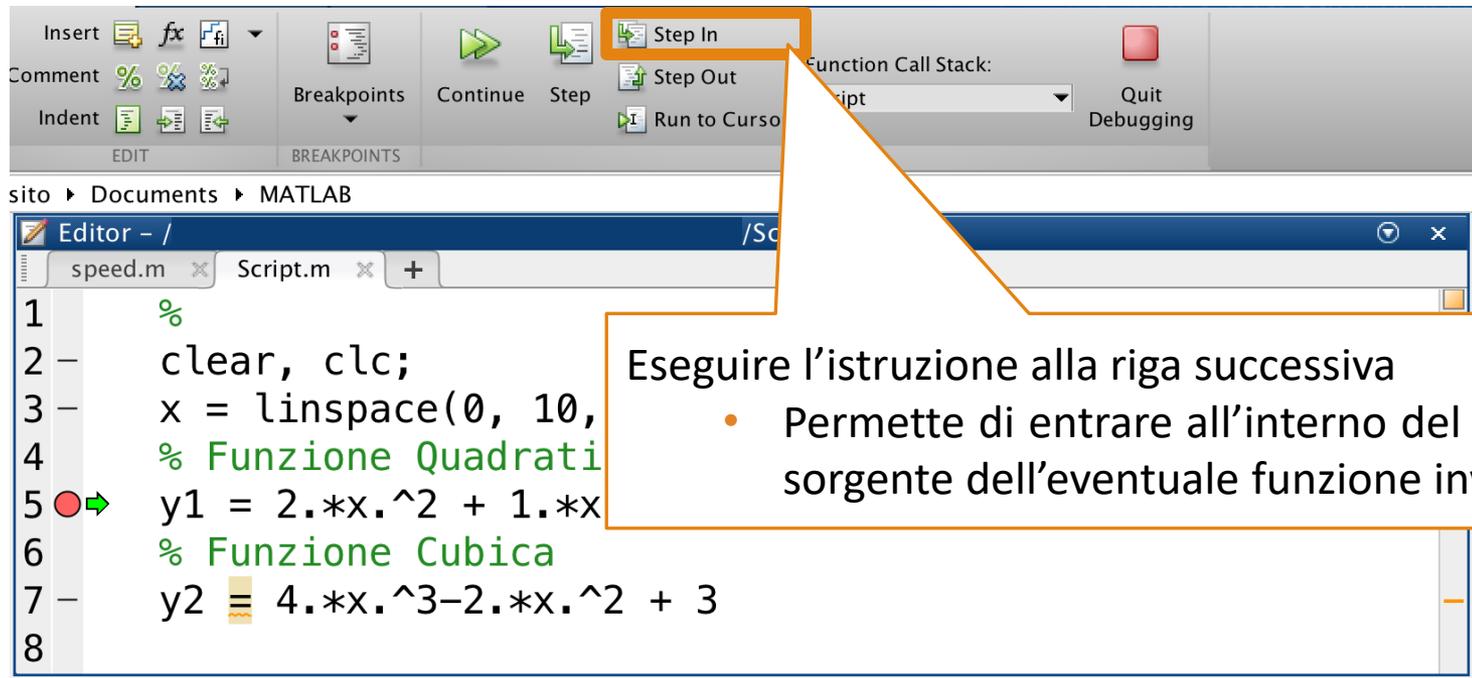
The screenshot shows the MATLAB debugging toolbar. The 'Step' button, represented by a green arrow pointing down, is highlighted with an orange box. A callout box with an orange border points to this button, containing the text: "Eeguire solo l'istruzione definita alla prossima riga". Below the toolbar, the MATLAB Editor window is open, showing a script named 'Script.m'. The script contains 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
```

The execution cursor is positioned at the beginning of line 5, indicated by a red circle and a green arrow.

- Quando l'esecuzione del programma si interrompe su un *breakpoint*
- MATLAB fornisce al programmatore diverse azioni da poter svolgere

Debugging – 7/7



The screenshot shows the MATLAB IDE interface. The top toolbar contains several debugging icons, with the 'Step In' icon (a green arrow pointing into a document) highlighted by an orange box. Below the toolbar, the 'Function Call Stack' panel is visible. The main editor window shows a script named 'Script.m' with the following code:

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

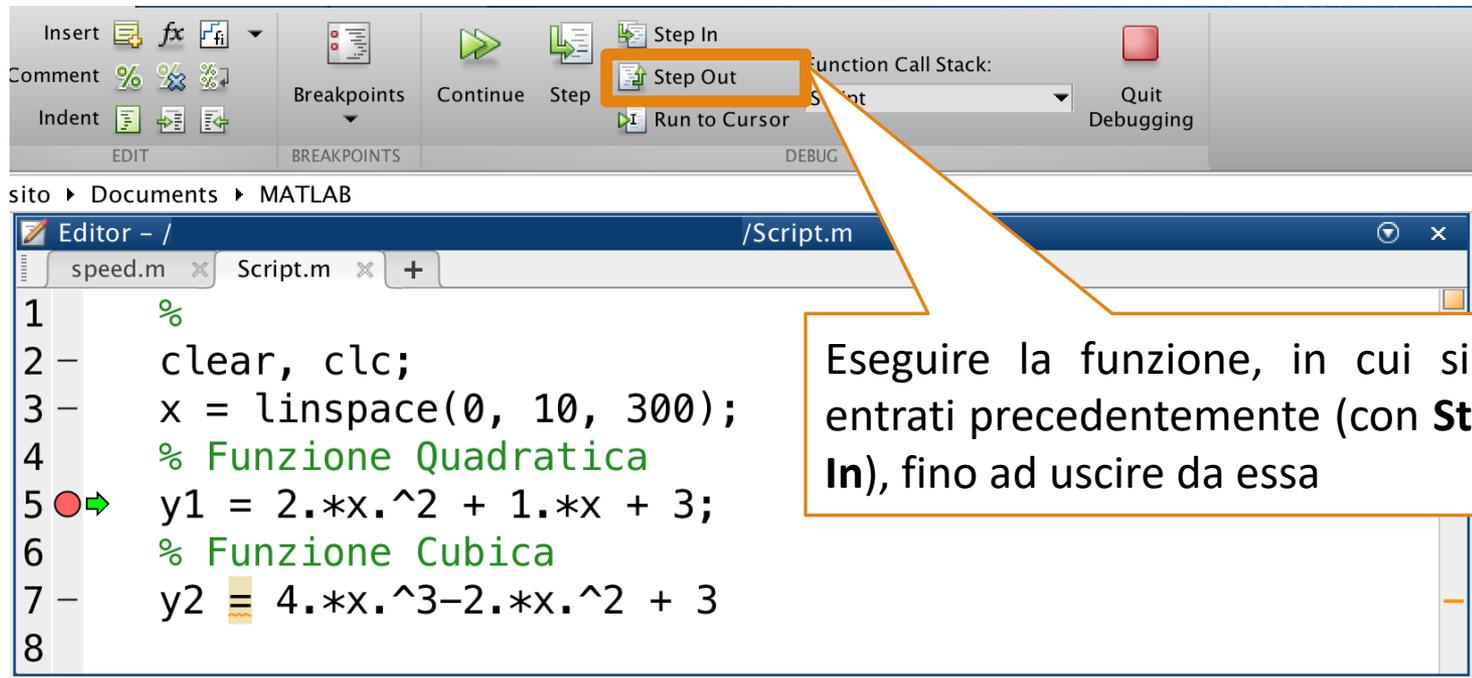
A red circle and a green arrow point to line 5. An orange callout box points to the 'Step In' button and contains the following text:

Eseguire l'istruzione alla riga successiva

- Permette di entrare all'interno del codice sorgente dell'eventuale funzione invocata

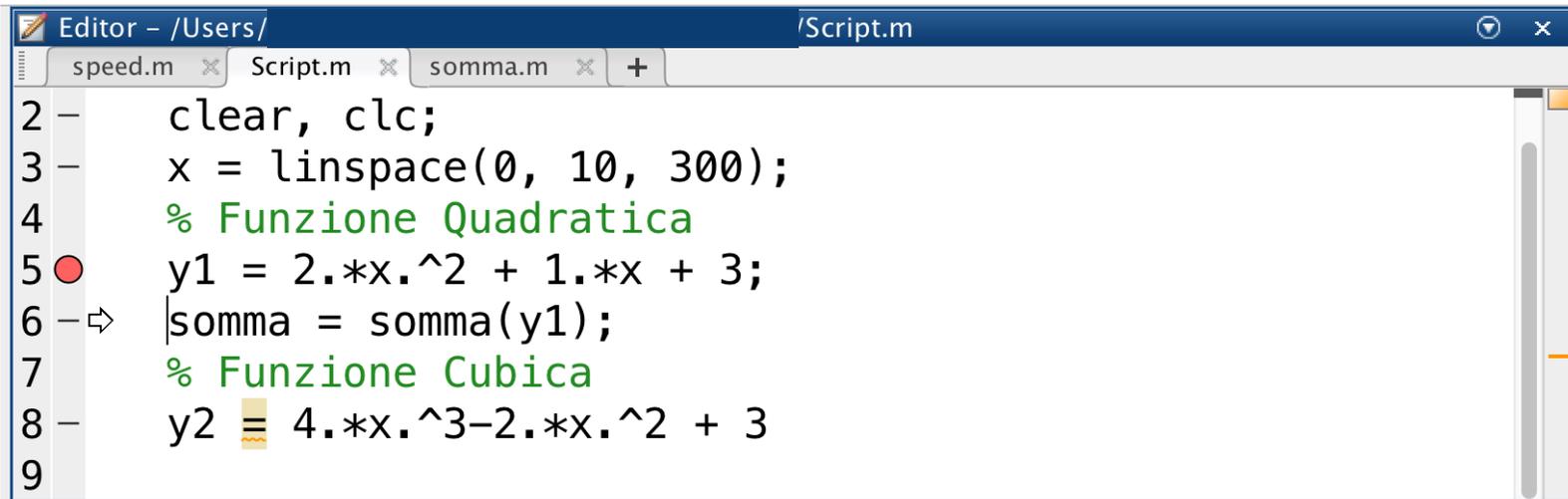
- Quando l'esecuzione del programma si interrompe su un *breakpoint*
- MATLAB fornisce al programmatore diverse azioni da poter svolgere

Debugging – 7/7



- Quando l'esecuzione del programma si interrompe su un *breakpoint*
- MATLAB fornisce al programmatore diverse azioni da poter svolgere

Debugging – 7/7



The image shows a MATLAB Editor window titled "Editor - /Users/ Script.m". The window contains a script with the following code:

```
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
```

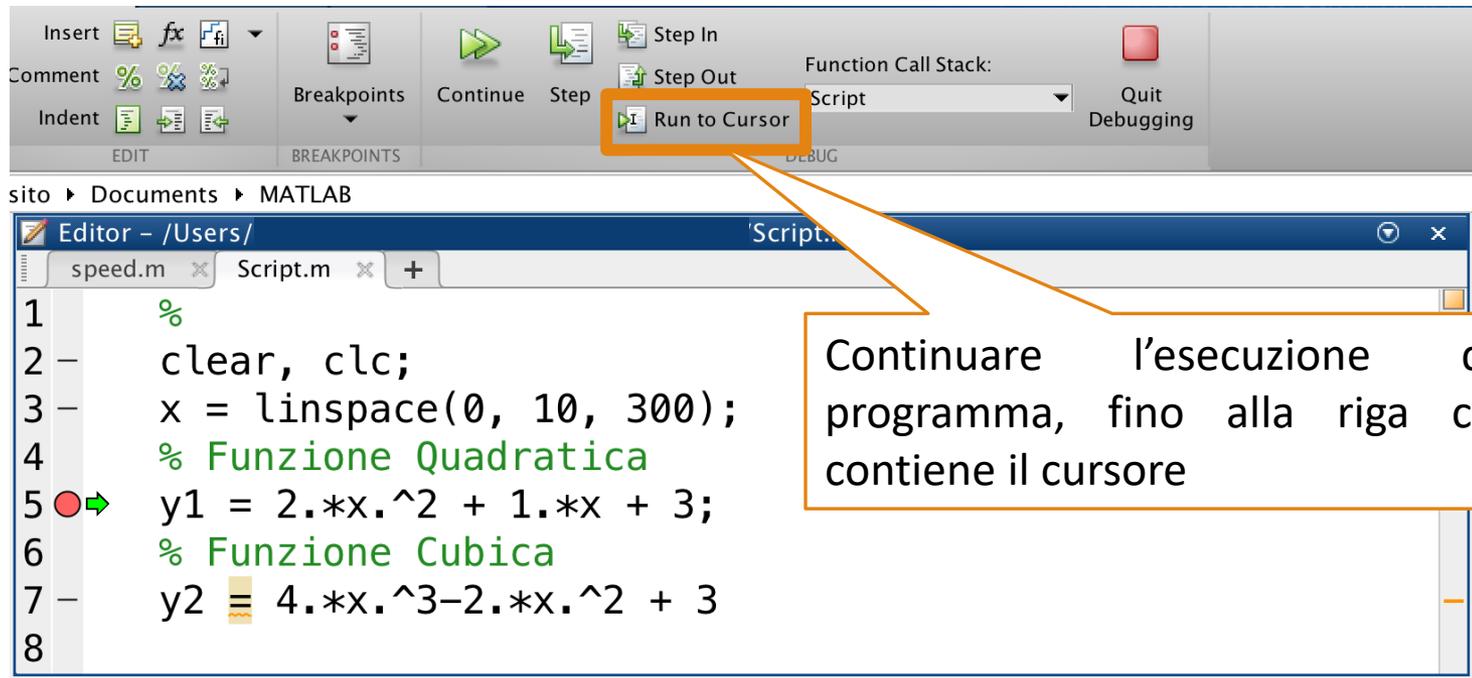
A red circle breakpoint is set on line 5. A cursor is positioned at the start of line 6. The script includes comments in Italian: "% Funzione Quadratica" and "% Funzione Cubica".

Debugging – 7/7

```
Editor - /Users/ /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/ /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 -
```

Debugging – 7/7



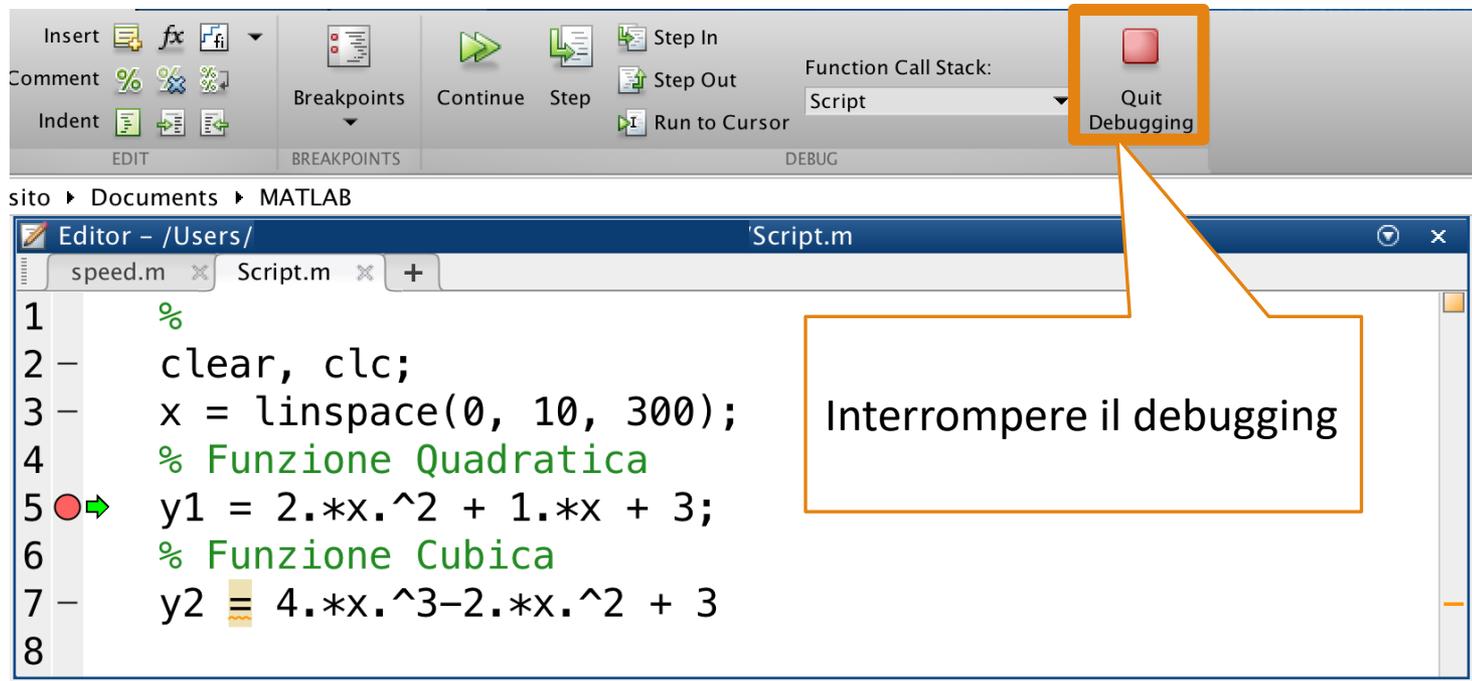
The screenshot shows the MATLAB IDE interface. The top toolbar contains various debugging tools. The 'Run to Cursor' button, represented by a green play icon and a cursor, is highlighted with an orange box. A callout box with an orange border points to this button, containing the text: 'Continuare l'esecuzione del programma, fino alla riga che contiene il cursore'. Below the toolbar, the MATLAB Editor window is open, showing a script named 'Script.m'. The script contains 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
```

The cursor is positioned at the beginning of line 5, and a red circle with a green arrow points to the 'Run to Cursor' button in the toolbar.

- Quando l'esecuzione del programma si interrompe su un *breakpoint*
- MATLAB fornisce al programmatore diverse azioni da poter svolgere

Debugging – 7/7



- Quando l'esecuzione del programma si interrompe su un *breakpoint*
- MATLAB fornisce al programmatore diverse azioni da poter svolgere

Riferimenti

- Capitolo 4
 - Paragrafo 8 [**Debugging dei programmi di MATLAB**]