



NAVI MUMBAI

# MATLAB

## Unit 7-Lecture 26

### Debugging M-files

---

BTech (CSBS) -Semester VII

14 October 2022, 09:35AM



# Debugging M-files

---

- 1) Preparing for debugging,
- 2) Examining values,
- 3) Debugging process
- 4) setting breakpoints
- 5) running with breakpoints
- 6) correcting and ending debugging,
- 7) correcting an M- file



# Debug MATLAB Code Files

---

You can diagnose problems in your MATLAB® code files by debugging your code interactively in the Editor and Live Editor or programmatically by using debugging functions in the Command Window. There are several ways to debug your code:

- 1) Display output by removing semicolons.
- 2) Run the code to a specific line and pause by clicking the Run to Here button .
- 3) Step into functions and scripts while paused by clicking the Step In button .
- 4) Add breakpoints to your file to enable pausing at specific lines when you run your code.



# Debug MATLAB Code Files

---

Before you begin debugging, to avoid unexpected results, save your code files and make sure that the code files and any files they call exist on the search path or in the current folder. MATLAB handles unsaved changes differently depending on where you are debugging from:

- 1) Editor — If a file contains unsaved changes, MATLAB saves the file before running it.
- 2) Live Editor — MATLAB runs all changes in a file, whether they are saved or not.
- 3) Command Window — If a file contains unsaved changes, MATLAB runs the saved version of the file. You do not see the results of your changes.



## Display Output: Example

---

For example, suppose that you have a script called **plotRand.m** that plots a vector of random data and draws a horizontal line on the plot at the mean.

```
n = 50;  
r = rand(n,1);  
plot(r)  
  
m = mean(r);  
hold on  
plot([0,n],[m,m])  
hold off  
title('Mean of Random Uniform Data')
```

### Command Window

```
>> plotRand
```

```
r =
```

```
0.9631
```

```
0.5468
```

```
0.5211
```

```
0.2316
```

```
0.4889
```

```
0.6241
```

```
fx 0.6791
```



# Display Output: Example



```
1 n = 50;  
2 r = rand(n,1)  
3 plot(r)  
4  
5 m = mean(r);  
6 hold on  
7 plot([0,n],[m,m])  
8 hold off  
9 title('Mean of Random Uniform Data')
```


```
r = 50x1  
0.0596  
0.6820  
0.0424  
0.0714  
0.5216  
0.0967  
0.8181  
0.8175  
0.7224  
0.1499  
:  
:
```





# Debug Using Run to Here

When debugging, the Run to Here button  becomes the Continue to Here button . In functions and classes, running to a specified line and then pausing is only available when debugging using the Continue to Here button.



```
1 n = 50;  
2  r = rand(n,1);  
3 ^ plot(r)  
4 Run to Here  
5 Run up to this line and pause  
6 hold on  
7 plot([0,n],[m,m])  
8 hold off  
9 title('Mean of Random Uniform Data')
```



# Debug Using Run to Here

---

When MATLAB pauses, multiple changes occur:

- 1) The  **Run** button in the **Editor** or Live Editor tab changes to a  **Continue** button.
- 2) The prompt in the Command Window changes to K>> indicating that MATLAB is in debug mode and that the keyboard is in control.
- 3) MATLAB indicates the line at which it is paused by using a green arrow and green highlighting.



```
1      n = 50;
2      → r = rand(n,1);
3      plot(r)
4
5      m = mean(r);
6      hold on
7      plot([0,n],[m,m])
8      hold off
9      title('Mean of Random Uniform Data')
```





# Debug Using Run to Here

---

The line at which MATLAB is paused does not run until after you continue running the code. To continue running the code, click the  **Continue** button. MATLAB continues running the file until it reaches the end of the file or a breakpoint. You also can click the Continue to Here button  to the left of the line of code that you want to continue running to.


To continue running the code line-by-line, on the **Editor** or **Live Editor** tab, click  **Step**. MATLAB executes the current line at which it is paused and the pauses at the next line.

```
1  n = 50;  
2  r = rand(n,1);  
3  → plot(r)  
4  
5  m = mean(r);  
6  hold on  
7  plot([0,n],[m,m])  
8  hold off  
9  title('Mean of Random Uniform Data')
```



# View Variable Value While Debugging

---

To view the value of a variable while MATLAB is paused, place your cursor over the variable. The current value of the variable appears in a data tip. The data tip stays in view until you move the cursor. To disable data tips, go to the **View** tab and click the  **Datatips** button off.





```
1  n = 50;
2  → r = rand(n,1);
3  plot(r) △
4
5  n: 1x1 double =
6      50
7
8  hold off
9  title('Mean of Random Uniform Data')
```

You also can view the value of a variable by typing the variable name in the Command Window. For example, to see the value of the variable `n`, type `n` and press **Enter**. The Command Window displays the variable name and its value. To view all the variables in the current workspace, use the Workspace browser.



# Pause a Running File

---

You can pause long running code while it is running to check on the progress and ensure that it is running as expected. To pause running code, go to the **Editor** or **Live Editor** tab and click the  **Pause** button. MATLAB pauses at the next executable line, and the  **Pause** button changes to a  **Continue** button. To continue running the code, press the  **Continue** button.

## Note

Clicking the  **Pause** button can cause MATLAB to pause in a file outside your own code.



# Examine Values While Debugging

---

When debugging a code file, you can view the value of any variable currently in the workspace while MATLAB® is paused. If you want to determine whether a line of code produces the expected result or not, examining values is useful. If the result is as expected, you can continue running the code or step to the next line. If the result is not as you expect, then that line, or a previous line, might contain an error.





# View Variable Value

---

There are several ways to view the value of a variable while debugging:

- Workspace browser — The Workspace browser displays all variables in the current workspace. The Value column of the Workspace browser shows the current value of the variable.

Workspace		
Name ▲	Value	Class
 n	6	double
 x	[1,2,4,8,16,32,1,1,1,1]	double

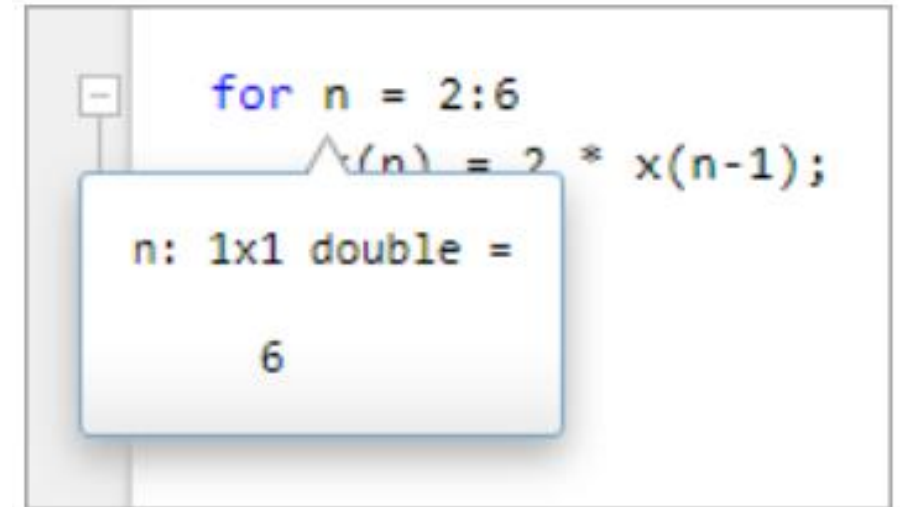


## View Variable Value

---

To view more details, double-click the variable. The Variables Editor opens, displaying the content for that variable. You also can use the `openvar` function to open a variable in the Variables Editor.

- Editor and Live Editor — To view the value of a variable in the Editor and Live Editor, place your cursor over the variable. The current value of the variable appears in a data tip. The data tip stays in view until you move the cursor. If you have trouble getting the data tip to appear, click the line containing the variable, and then move the pointer next to the variable.





## View Variable Value Outside Current Workspace

You also can use the **dbstack** function to view the current workspace in the Command Window:

To examine the values of variables outside of the current workspace, select a different workspace. In the Editor or Live Editor, select a workspace from the drop-down list to the right of the function call stack at the top of the file.

```
dbstack
```

```
> In mean (line 48)  
In plotRand (line 5)
```





## View Variable Value Outside Current Workspace

---

You also can use the **dbup** and **dbdown** functions in the Command Window to select the previous or next workspace in the function call stack. To list the variables in the current workspace, use **who** or **whos**. If you attempt to view the value of a variable in a different workspace while MATLAB is in the process of overwriting it, MATLAB displays an error in the Command Window.






```
K>> x
Variable "x" is inaccessible. When a variable appears on both sides of an assignment
statement, the variable may become temporarily unavailable during processing.
```

The error occurs whether you select the workspace by using the drop-down list to the right of the function call stack or the **dbup** command.





# Debug by Using Keyboard Shortcuts or Functions

Action	Description	Keyboard Shortcut	Function
Continue 	Continue running file until the end of the file is reached or until another breakpoint is encountered.	F5	<a href="#">dbcont</a>
Step 	Run the current line of code.	F10  (Shift+Command+O on macOS systems)	<a href="#">dbstep</a>
Step In 	Run the current line of code, and, if the line contains a call to another function, step into that function.	F11  (Shift+Command+I on macOS systems)	<a href="#">dbstep</a> in
Step Out 	After stepping in, run the rest of the called function, leave the called function, and then pause.	Shift+F11  (Shift+Command+U on macOS systems)	<a href="#">dbstep</a> out
Stop 	End debugging session.	Shift+F5	<a href="#">dbquit</a>
Set Breakpoint	Set a breakpoint at the current line, if no breakpoint exists.	F12	<a href="#">dbstop</a>
Clear Breakpoint	Clear the breakpoint at the current line.	F12	<a href="#">dbclean</a>



# Function Call Stack

---

When you step into a called function or file, MATLAB displays the list of the functions it executed before pausing at the current line. The list, also called the **function call stack**, is shown at the top of the file and displays the functions in order, starting on the left with the first called script or function, and ending on the right with the current script or function in which MATLAB is paused.



For each function in the function call stack, there is a corresponding workspace. Workspaces contain variables that you create within MATLAB or import from data files or other programs. Variables that you assign through the Command Window or create by using scripts belong to the base workspace. Variables that you create in a function belong to their own function workspace.



# HW

---

dbcont	
dbclear	
dbstack	
dbstatus	
dbstop	
dbstep	



NAVI MUMBAI

# MATLAB

## Unit 7-Lecture 27

Debugging process , setting breakpoints

---

BTech (CSBS) -Semester VII

14 October 2022, 09:35AM



# Debugging M-files

---

- 1) preparing for debugging,
- 2) examining values,
- 3) Debugging process
- 4) setting breakpoints
- 5) running with breakpoints
- 6) correcting and ending debugging
- 7) correcting an M- file



# Build Process

- Before you can build an executable program or shared library for a model, choose and set up a compiler or IDE and configure the target environment.
- Several methods are available for initiating the build process.
- Tooling is available for reloading, rebuilding, and relocating generated code.
- If your system includes referenced models, reduce build time and control whether the code generator regenerates code for the top model.
- To improve the speed of code execution, consider using available profiling capabilities.

## Functions

▼ Initiate Build Process	
<code>packNGo</code>	Package generated code in ZIP file for relocation
<code>rtw_precompile_libs</code>	Build model libraries without building model
<code>codebuild</code>	Compile and link generated code
<code>rtwrebuild</code>	Rebuild generated code from model
<code>slbuild</code>	Build standalone executable file or model reference target for model



# Build Process

## ▼ Get or Modify Build Process Controls

<code>coder.buildstatus.close</code>	Close Build Status window
<code>coder.buildstatus.open</code>	Open Build Status window
<code>RTW.getBuildDir</code>	Get build folder information from model build information
<code>Simulink.fileGenControl</code>	Specify root folders for files generated by diagram updates and model builds
<code>switchTarget</code>	Select target for model configuration set



# Debugging Approaches

---

There are probably a lot of ways to debug programs. These include:

- 1) editing the code and removing semicolons or adding a **keyboard** statement at judicious locations
- 2) **setting a breakpoint** at a particular line and stepping through code from there
- 3) using a variant of setting a breakpoint by using **dbstop if error**
- 4) seeing if the **mlint code analyser** can help (also reachable from the Tools menu)
- 5) comparing variants of the code using the **File and Folder Comparisons** tool or **visdiff** for command-line access





# Set Breakpoint

---

Setting breakpoints pauses the execution of your MATLAB® program so that you can examine values where you think an issue might have occurred. You can set breakpoints interactively in the Editor or Live Editor, or by using functions in the Command Window.

There are three types of breakpoints:

- 1) Standard
- 2) Conditional
- 3) Error




# Set Breakpoint

---

By default, when MATLAB reaches a breakpoint, it opens the file containing the breakpoint.

To disable this option:

1. From the **Home** tab, in the **Environment** section, click  **Preferences**.
2. In the Preferences window, select **MATLAB > Editor/Debugger**.
3. Clear the **Automatically open file when MATLAB reaches a breakpoint** option and click **OK**.



# Standard Breakpoints

---

A standard breakpoint pauses at a specific line in a file. To set a standard breakpoint, click the gray area to the left of the executable line where you want to set the breakpoint. Alternatively, you can press the F12 key to set a breakpoint at the current line. If you attempt to set a breakpoint at a line that is not executable, such as a comment or a blank line, MATLAB sets it at the next executable line.

```
1  n = 50;  
2  r = rand(n,1);  
3  plot(r)  
4  
5  m = mean(r);  
6  hold on  
7  plot([0,n],[m,m])  
8  hold off  
9  title('Mean of Random Uniform Data')
```



# Standard Breakpoints

---

To set a standard breakpoint programmatically, use the **dbstop** function.

```
dbstop in plotRand at 3
```

When debugging a file that contains a loop, set the breakpoint inside the loop to examine the values at each increment of the loop. Otherwise, if you set the breakpoint at the start of the loop, MATLAB pauses at the loop statement only once. For example, this code creates an array of ten ones and uses a for loop to perform a calculation on items two through six of the array:

```
x = ones(1:10);  
  
for n = 2:6  
    x(n) = 2 * x(n-1);  
end
```



# Standard Breakpoints

---

For MATLAB to pause at each increment of the for loop (a total of five times), set a breakpoint at line four.

```
3  for n = 2:6  
4      x(n) = 2 * x(n-1);  
5  end
```



# Conditional Breakpoints

---

- To set a conditional breakpoint, right-click the gray area to the left of the executable line where you want to set the breakpoint and select Set Conditional Breakpoint.
- If a breakpoint already exists on that line, select Set/Modify Condition. In the dialog box that opens, enter a condition and click OK.
- A condition is any valid MATLAB expression that returns a logical scalar value.
- **Example:** Set a conditional breakpoint at line four with the condition  $n \geq 4$ . When you run the code, MATLAB runs through the for loop twice and pauses on the third iteration at line four when  $n$  is 4. If you continue running the code, MATLAB pauses again at line four on the fourth iteration when  $n$  is 5, and then once more, when  $n$  is 6.



# Conditional Breakpoints: Example

Code:

```
x = ones(1:10)

for n = 2:6
    x(n) = 2 * x(n-1);
end
```

Result:

1	x = ones(1:10);
2	
3	for n = 2:6
4	x(n) = 2 * x(n-1);
5	end



Stop the file:

```
dbstop in myprogram at 6 if n>=4
```



# Error Breakpoints

---

To set an error breakpoint, on the **Editor** tab, click  **Run**  and select from these options:

- **Pause on Errors** to pause on all errors.
- **Pause on Warnings** to pause on all warnings.
- **Pause on NaN or Inf** to pause on NaN (not-a-number) or Inf (infinite) values.

```
dbstop if error
```

```
dbstop if caught error MATLAB:ls:InputsMustBeStrings
```