KU LEUVEN

MATLAB

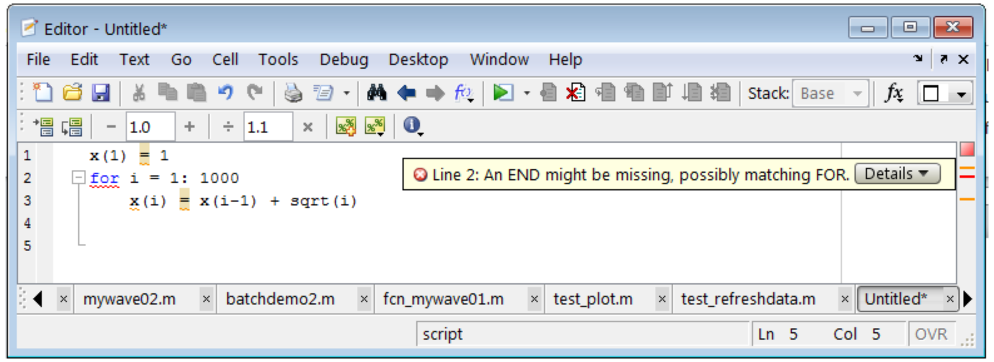
Editor + debugging

Edit MATLAB files

- use ANY ascii text editor
  - Windows: Notepad, Notepad++, …
    - Not Word!
  - Linux: vi, emacs, gedit, …
- MATLAB has a built in editor, to start it:
  - **`edit`**
    or to edit an existing code
    `edit filename.m`
  - Select from Home menu, click icon to start editing new file (`ctrl+n`)
  - Select file from directory (double click)

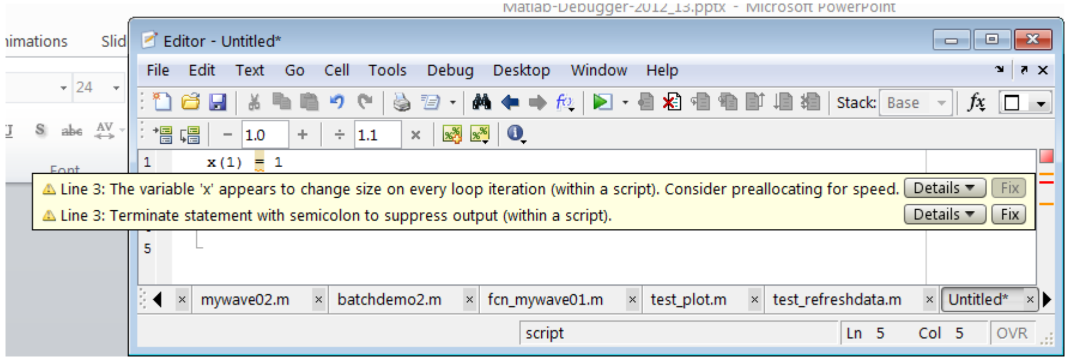
MATLAB editor

- Syntax highlighting
- Code checking



```
x(1) = 1
for i = 1: 1000
    x(i) = x(i-1) + sqrt(i)
```

Line 2: An END might be missing, possibly matching FOR.



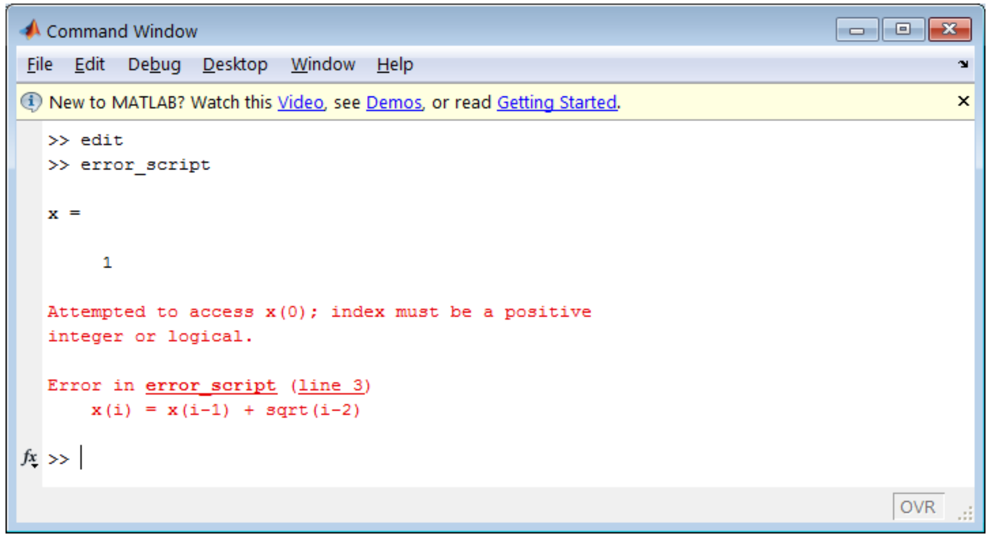
Line 3: The variable 'x' appears to change size on every loop iteration (within a script). Consider preallocating for speed.

Line 3: Terminate statement with semicolon to suppress output (within a script).

Code Analyzer

- **Red** indicates syntax errors were detected. Another way to detect some of these errors is using syntax highlighting to identify unterminated strings, and delimiter matching to identify unmatched keywords, parentheses, braces, and brackets.
- **Orange** indicates warnings or opportunities for improvement, but no errors, were detected.
- **Green** indicates no errors, warnings, or opportunities for improvement were detected.

Error messages


Command Window
File Edit Debug Desktop Window Help
New to MATLAB? Watch this Video, see Demos, or read Getting Started.
OVR

```
>> edit
>> error_script

x =

     1

Attempted to access x(0); index must be a positive
integer or logical.

Error in error_script (line 3)
    x(i) = x(i-1) + sqrt(i-2)

>> 
```

- Click the link to jump into editor to the line where the error occured.
- *File: caller_script.m*
- *File: error_script_1.m*

Bugs & debugging

- Matlab finds syntax errors during compilation.
  These errors are usually easy to fix. (syntax highlighting)
- Matlab run-time errors; these errors tend to be more difficult to track down
  - the workspace local to the function is lost when the error forces a return to the Matlab prompt and the base workspace.
  - If you use the semicolon to suppress the display of intermediate results, you won't know where the error occurred.
- Debugging code: combination of art and science.
  Every programmer has his/her own approach, but there are common techniques for finding problems with the code.

Debugging

- Organized programs are. . .
  - easier to maintain
  - easier to debug
  - not much harder to write
- Debugging. . .
  - is inevitable
  - can be anticipated with good program design
  - can be done interactively within Matlab

Dealing with errors

Advice:

- Avoid frustration, never expect a program to work immediately
- The error mentioned in the error message is an indication
- The line number mentioned in the error message is an indication
- Treat error messages with suspicion
- Never spend more than 15 minutes of staring for errors.
- After 15 minutes the time needed to find the error is inversely related to the distance between you and the computer

Bugs

- Advice_1: Information in a program is manipulated using variables
  - most software bugs arise from the incorrect use of variables
  - most common debugging method is to *inspect variables* during operation of the program.
    - `disp` command, or
    - omit the semicolon (for "silent output") at the end of the line,
    - add keyboard statements to allow to you examine the workspace state at the point where keyboard is issued.
    - comment out the leading function declaration so the M-file can be run as a script, making the intermediate results accessible in the base workspace.
    - use the Matlab debugger.

Bugs

- Advice_2: a Matlab function or program can call other functions
  - which in turn call *other* functions.
  - When an error occurs, Matlab provides and error message displaying what is known as the "stack," i.e. function(s) that *called* that function are listed.
  - cause of confusion is when the error message refers to a built-in Matlab function you have never even *heard* of.
    - happens when you use a Matlab function that in turn calls *other* Matlab functions that you are not aware of.
    - almost always the result of passing an improper argument to the Matlab function.

Debugger

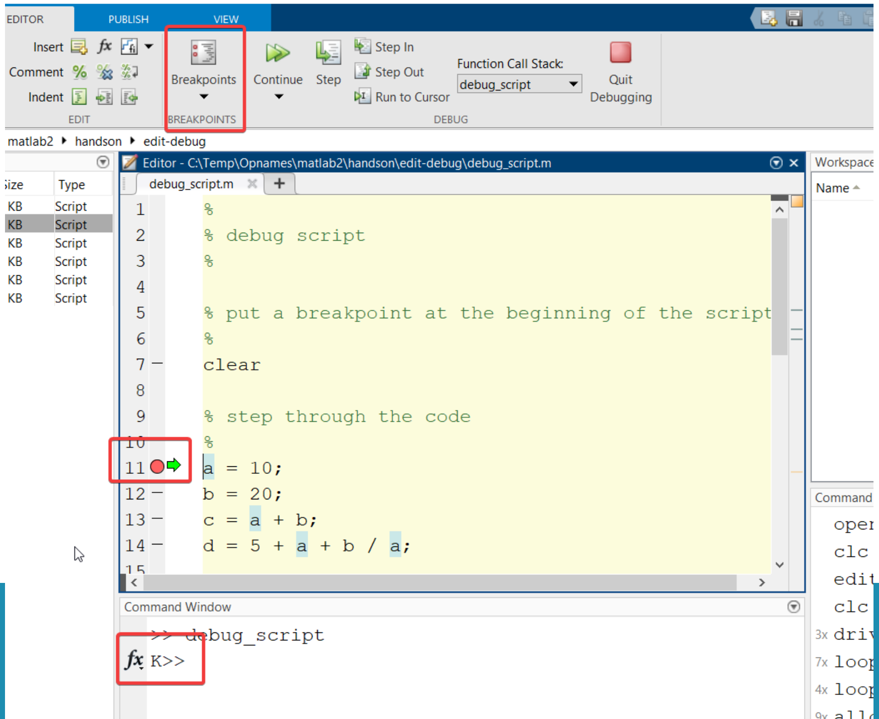
- Debugger: useful in tracking errors
  Once your code is written (and saved), you can test it by making use of the debugger within the Matlab editor.
- Useful in determining where a programming error has occurred or in examining the flow of execution.
- Suspend execution at specified points for checking the values of variables in prior lines.

Debugger

- Debugging recipe
  - Open the file.
  - Place one/more breakpoints at the points of interest
    - go to the line of code to investigate and click on ● or F12
  - Run the file
- The computer will pause at these breakpoints during execution of the code to allow you to inspect the values of variables.
- Advice:
  - DO NOT rely on the debugger to program!
  - Think before programming!!!

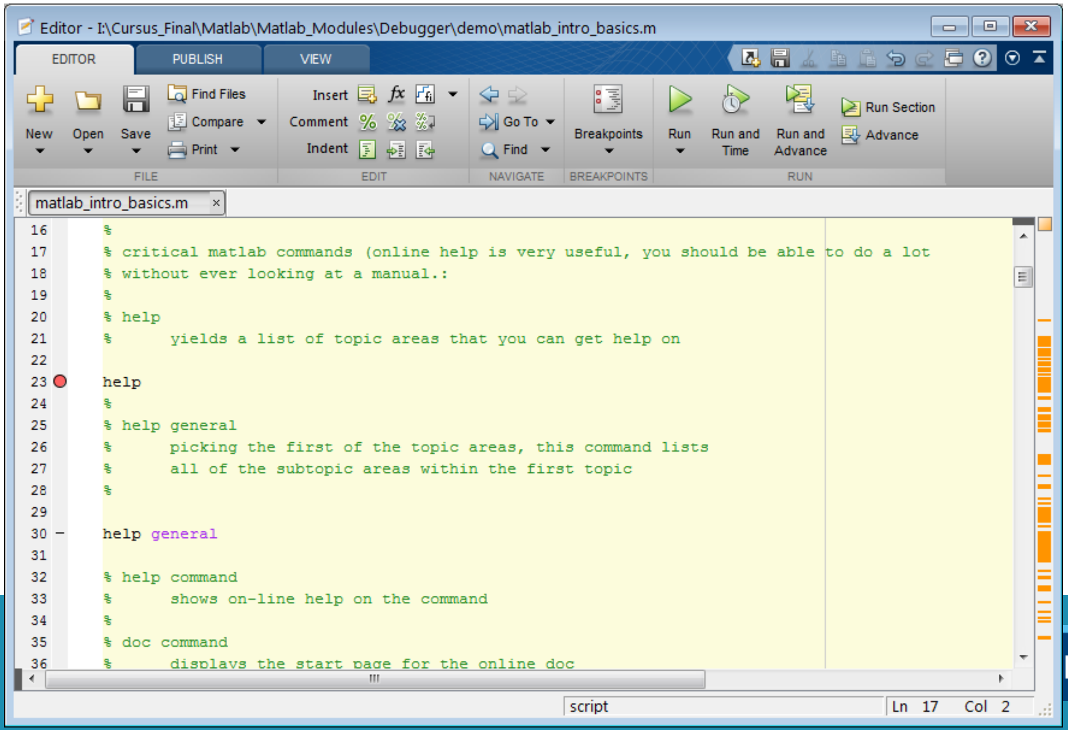
Debugger

- Debugging mode: the command line prompt will change from '>>' to 'K>>'.
- When the program has paused at a breakpoint, a green arrow appears in the editor window indicating the breakpoint location
- Exit from debugger mode: `dbquit`



```
%
% debug script
%

% put a breakpoint at the beginning of the script
%
clear

% step through the code
%
a = 10;
b = 20;
c = a + b;
d = 5 + a + b / a;
```

```
>> debug_script
K>>
```



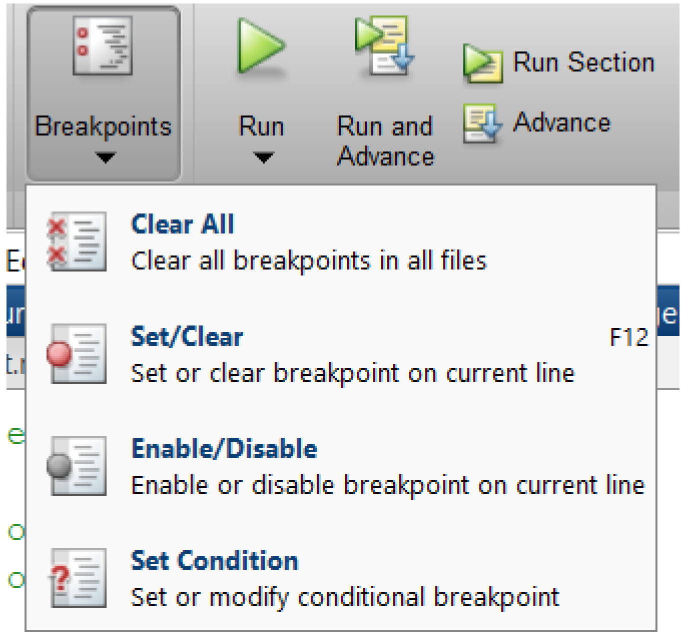
```
%
% critical matlab commands (online help is very useful, you should be able to do a lot
% without ever looking at a manual.:
%
% help
%       yields a list of topic areas that you can get help on

help
%
% help general
%       picking the first of the topic areas, this command lists
%       all of the subtopic areas within the first topic
%

help general

% help command
%       shows on-line help on the command
%
% doc command
%       displays the start page for the online doc
```

Debugger: breakpoints

- There are three types of breakpoints:
  - Standard breakpoints
    - A standard breakpoint pauses at a specified line in a file.
  - Conditional breakpoints
    - Causes MATLAB to pause at a specified line in a file only when the specified condition is met.
    - Use conditional breakpoints when you want to examine results after some iterations in a loop.
  - Error breakpoints (from Run menu)

Debugger: breakpoints

- Clear All: remove all the breakpoints
- Set/Clear: this option is used to set and remove a breakpoint at a specific spot
- Enable/Disable: disable selected breakpoints so that your program temporarily ignores them
- Set Condition: set the condition for a conditional breakpoint



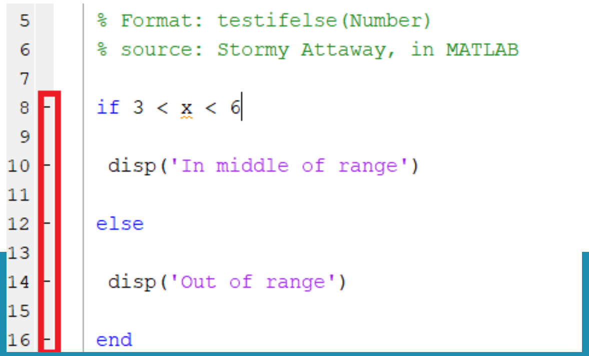
Clear All
Clear all breakpoints in all files

Set/Clear F12
Set or clear breakpoint on current line

Enable/Disable
Enable or disable breakpoint on current line

Set Condition
Set or modify conditional breakpoint

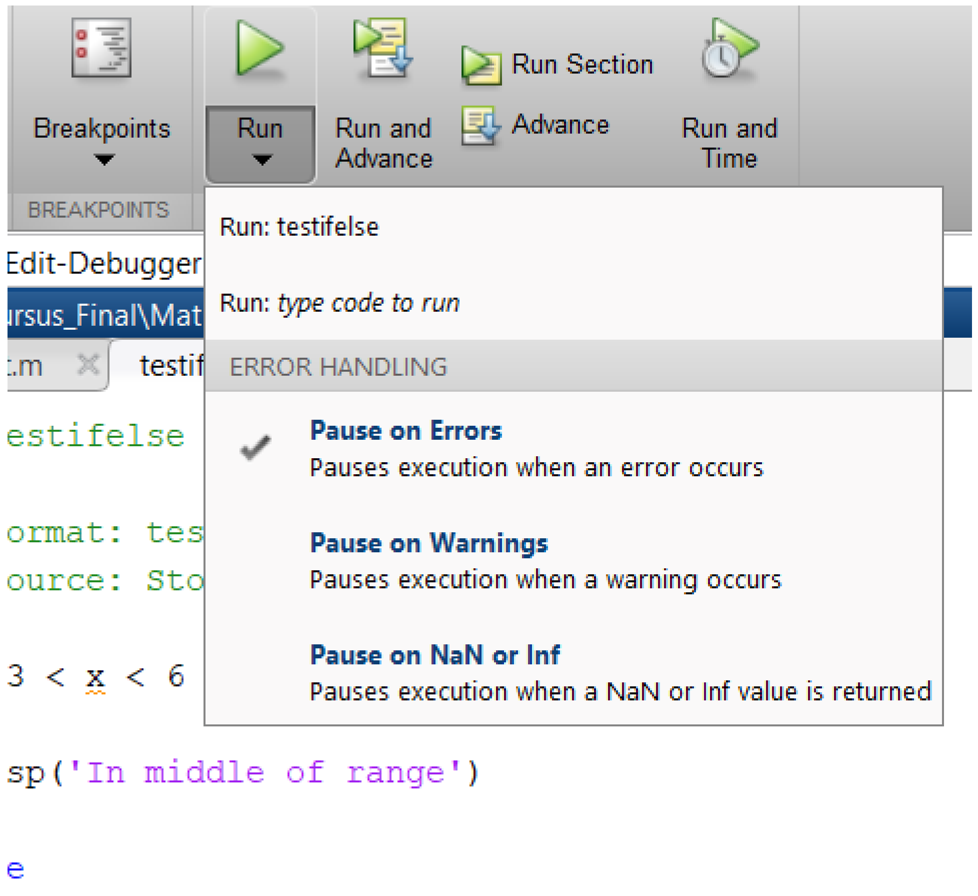
- Executable lines are indicated by a dash (—) in the breakpoint alley. Click the breakpoint alley next to the line to add a breakpoint at that line.



```
% Format: testifelse(Number)
% source: Stormy Attaway, in MATLAB

if 3 < x < 6

 disp('In middle of range')

else

 disp('Out of range')

end
```

Debugger: breakpoints

- An error breakpoint causes MATLAB to pause program execution and enter debug mode if MATLAB encounters a problem.
- Unlike standard and conditional breakpoints, you do not set these breakpoints at a specific line in a specific file. When you set an error breakpoint, MATLAB pauses at any line in any file if the error condition specified occurs. MATLAB then enters debug mode and opens the file containing the error, with the execution arrow at the line containing the error.



Run: testifelse

Run: *type code to run*

ERROR HANDLING

✓ **Pause on Errors**
Pauses execution when an error occurs

Pause on Warnings
Pauses execution when a warning occurs

Pause on NaN or Inf
Pauses execution when a NaN or Inf value is returned

```
estifelse

ormat: tes
ource: Sto

3 < x < 6

sp('In middle of range')

e
```

Debugger

Actions:

- *Run (F5):* execute the program
- *Continue:* continues program execution where execution was suspended.
- *Step (F10):* executes the current program line.
- *Step In (F11):* goes into the code of a function call at the current program line.
- *Clear All Breakpoints:* removes all breakpoints from the current file.

File:debug_script.m

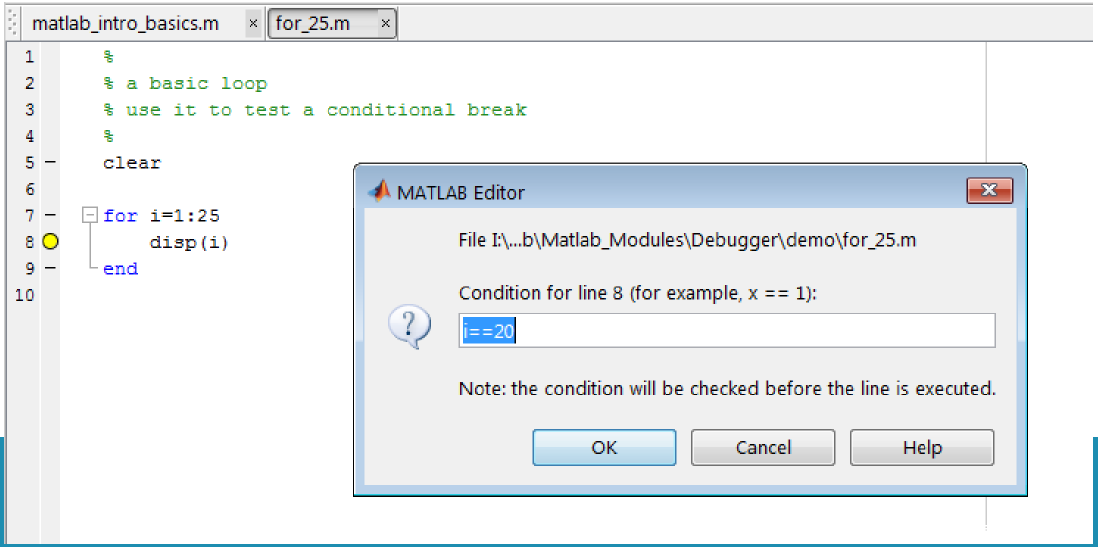
More info

- Conditional breakpoint

http://blogs.mathworks.com/desktop/2010/08/09/debugging-points/

- On error/warning

http://blogs.mathworks.com/desktop/2008/05/19/stop-in-the-name-of-error/



```
%
% a basic loop
% use it to test a conditional break
%
clear

for i=1:25
    disp(i)
end
```

MATLAB Editor

File I:\...b\Matlab_Modules\Debugger\demo\for_25.m

Condition for line 8 (for example, x == 1):

i==20

Note: the condition will be checked before the line is executed.

Cell mode

- Cell mode feature in the Matlab Editor/Debugger (can) make the experimental phase of your work with M-file scripts easier.
- Ideas:
  - overall structure of many M-file scripts consist of multiple sections.
  - in larger files, you often focus efforts on a single section at a time, refining the code in just that section.
- M-file *cells:* defined section of code.
- Using cells for rapid code iteration:
  - In the Matlab Editor/Debugger, enable cell mode. Select **Cell -> Enable Cell Mode**. Items in the **Cell** menu become selectable and the cell toolbar appears.
  - Define the boundaries of the cells in an M-file script using cell features. Cells are denoted by a specialized comment syntax **%%**.
- ex. *Matlab_debug_cell_2*