Lecture 6: Debugging

B0B17MTB, BE0B17MTB – MATLAB

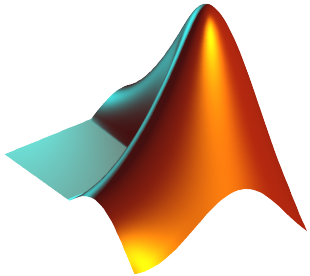
Miloslav Čapek, Viktor Adler, *et al.*

Department of Electromagnetic Field
Czech Technical University in Prague
Czech Republic
matlab@fel.cvut.cz

April 22
Summer semester 2025/26

Outline

1. Debugging
2. Measure the performance of a code
3. Error-treatment
4. Excercises

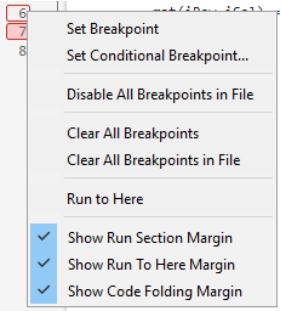


Debugging I.

- *Bug* → *debugging*.
- We distinguish:
  - Semantic errors (“logical” or “algorithmic” errors).
    - Usually difficult to identify.
  - Syntax errors (“grammatical” errors).
    - Pay attention to the content of error messages – it makes error elimination easier.
  - Unexpected events (see later).
    - For example, a problem with writing to open file, not enough space on disk etc.
  - Rounding errors (everything is calculated as it should be but the result is wrong anyway).
    - It is necessary to analyze the algorithm in advance, to determine the dynamics of calculation etc.
- Software debugging and testing is an integral part of software development.
  - Later we will discuss the possibilities of code acceleration using MATLAB profile.

Debugging II.

- We first focus on semantic and syntax errors in scripts.
  - We always test the program using test-case where the result is known.
- Possible techniques:
  - Using functions `who`, `whos`, `keyboard`, `disp`.
  - Using debugging tools in MATLAB editor (illustration).
    - Using MATLAB built-in debugging functions.



`dbclear`	Remove breakpoints
`dbcont`	Resume execution
`dbdown`	Reverse dbup workspace shift
`dbquit`	Quit debug mode
`dbstack`	Function call stack
`dbstatus`	List all breakpoints
`dbstep`	Execute next executable line from current breakpoint
`dbstop`	Set breakpoints for debugging
`dbtype`	Display file with line numbers
`dbup`	Shift current workspace to workspace of caller in debug mode
`keyboard`	Give control to keyboard
`echo`	Display statements during function or script execution

Useful Functions for Script Generation

- The function `keyboard` stops execution of the code and gives control to the keyboard.
  - The function is widely used for code debugging as it stops code execution at the point where any doubts about the code functionality exist

```
K>>
```

  - `keyboard` status is indicated by `K»` (K appears before the prompt).
  - The keyboard mode is terminated by `dbcont` or press F5 (Continue).
- Function `pause` halts code execution.
  - `pause(x)` halts code execution for x seconds.

```
% code; code; code
pause;
```

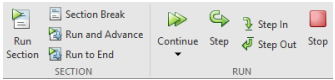
- See also: `echo`, `waitforbuttonpress`.
  - Special purpose functions.

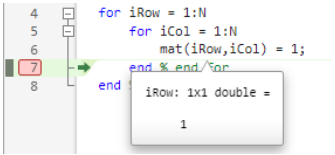
Debugging I.

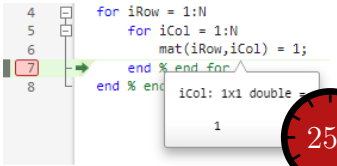
- For the following piece of code:

```
clear; clc;
N   = 5e2;
mat = nan(N,N);
for iRow = 1:N
   for iCol = 1:N
      mat(iRow,iCol) = 1;
   end % end for
end % end for
```

- Use MATLAB editor to:
  - set breakpoint (click on dash next to the line number),
  - run the script (F5),
  - check the status of variables (`keyboard` mode or hover over the variable's name),
  - keep on tracing the script.
    - Find difference between Continue and Step (F10)?



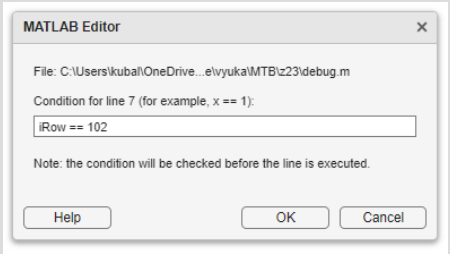


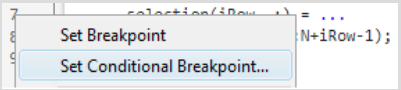


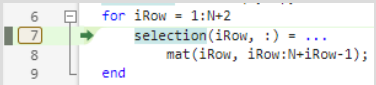
Advanced Debugging

- Conditional breakpoints:
  - Serve to suspend the execution of code when a condition is fulfilled.
    - Sometimes, the set up of the correct condition is not an easy task ...
  - Easier to find errors in loops.
    - Code execution can be suspended in a particular loop.
  - The condition may be arbitrary evaluable logical expression.

```
% code with an error
clear; clc;
N  = 100;
mat = magic(2*N);
selection = zeros(N, N);
for iRow = 1:N+2
   selection(iRow, :) = ...
      mat(iRow, iRow:N+iRow-1);
end
```







Selected Hints for Code Readability I.

- Use indention of loop's body, indention of code inside conditions(TAB).
  - Size of indention can be adjusted in `preferences` (usually 3 or 4 spaces).
- Use "positive" conditions.
  - *i.e.*use `isBigger` or `isSmaller`, not `isNotBigger` (can be confusing).
- Complex expressions with logical and relation operators should be evaluated separately → higher readability of code.
  - Compare:

```
if (val>lowLim)&(val<upLim)&~ismember(val,valArray)
   % do something
end
```

```
isValid = (val > lowLim) & (val < upLim);
isNew   = ~ismember(val, valArray);
if isValid & isNew
   % do something
end
```

Selected Hints for Code Readability II.

- Code can be separated with a blank line to improve clarity.
- Use two lines for separation of blocks of code.
  - Alternatively use cells or commented lines, for example:

```
% ---------
```

- Consider to use of spaces to separate operators (=, &, |).
  - To improve code readability:

```
(val>lowLim)&(val<upLim)&~ismember(val,valArray)
```

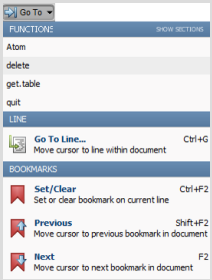
  - vs.

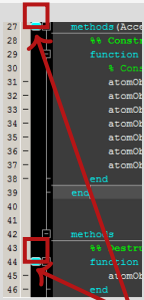
```
(val > lowLim) & (val < upLim) & ~ismember(val, valArray)
```

- In the case of nesting use comments placed after `end`.

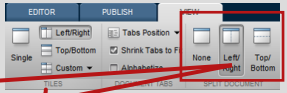
Useful Tools for Long Functions

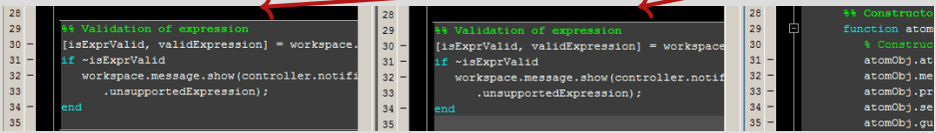
- Bookmarks:
  - CTRL+F2 (add/remove bookmark),
  - F2 (next bookmark),
  - SHIFT+F2 (previous bookmark).
- Go to ...
  - CTRL+G (go to line).
- Long files can be split.
  - Same file can be opened *e.g.*twice.





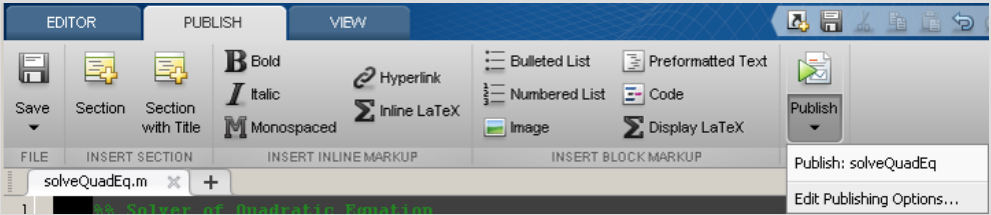
bookmarks





Function `publish` I.

- Serves to create script, function or class documentation.
- Provides several output formats (html, doc, ppt, LaTeX, ...).
- Help creation (» `doc my_fun`) directly in the code comments!
  - Provides wide scale of formatting properties(titles, numbered lists, equations, graphics insertion, references, ...).
- Enables to insert print screens into documentation.
  - Documented code is implicitly launched on publishing.
- Supports documentation creation directly from editor menu:



Function `publish` II.

```
%% Solver of Quadratic Equation
% Function *solveQuadEq* solves quadratic equation.
%% Theory
% A quadratic equation is any equation having the form
% $ax^2+bx+c=0$
% where |x| represents an unknown, and |a|, |b|, and |c|
% represent known numbers such that |a| is not equal to 0.
%% Head of function
% All input arguments are mandatory!
function x = solveQuadEq(a, b, c)
%%
% Input arguments are:
%%
% * |a| - _qudratic coefficient_
% * |b| - _linear coefficient_
% * |c| - _free term_
%% Discriminant computation
% Gives us information about the nature of roots.
D = b^2 - 4*a*c;
%% Roots computation
% The quadratic formula for the roots of the general
% quadratic equation:
%
% $$x_{1,2} = \frac{ - b \pm \sqrt D }{2a}.$$
%
% Matlab code:
%%
x(1) = (-b + sqrt(D))/(2*a);
x(2) = (-b - sqrt(D))/(2*a);
%%
% For more information visit <http://elmag.org>.
```

publish

Solver of Quadratic Equation

Function **solveQuadEq** solves quadratic equation.

Contents

- Theory
- Head of function
- Discriminant computation
- Roots computation

Theory

A quadratic equation is any equation having the form $ax^2 + bx + c = 0$ where x represents an unknown, and a, b, and c represent known numbers such that a is not equal to 0.

Head of function

All input arguments are mandatory!

```
function x = solveQuadEq(a, b, c)
```

Input arguments are:

- a - *qudratic coefficient*
- b - *linear coefficient*
- c - *free term*

Discriminant computation

Gives us information about the nature of roots.

```
D = b^2 - 4*a*c;
```

Roots computation

The quadratic formula for the roots of the general quadratic equation:

$$x_{1,2} = \frac{-b \pm \sqrt{D}}{2a}.$$

Matlab code:

```
x(1) = (-b + sqrt(D))/(2*a);
x(2) = (-b - sqrt(D))/(2*a);
```

For more information visit http://elmag.org/matlab.

Time Functions in MATLAB – Profiling

- What is the way to measure how long it takes for a program to be executed?
  - Distinguish between reasonably fast and very fast codes:

```
t0 = tic;
% code
t1 = toc(t0)
```

```
tic
   for k = 1:100
      % code
   end
toc
```

  - Other options? Which one is the best?
  - MathWorks recommends functions `tic` – `toc` (esp. for new CPUs with hyper-threading).
  - Function `timeit` uses also `tic` – `toc` but it provides additional statistic processing.

```
t0a = tic;
fft(x);
toc(t0a)
```

```
t0c = cputime;
fft(x);
e = cputime - t0c;
```

```
f = @() fft(x);
timeit(f, 1)
```

▶ See MATLAB tips.

Measure the performance

- Use `tic-toc` to measure the comp. time.
- Does pre-allocation improve the performance?
- Can you vectorize the code?
- Measure the comp. time of the vectorized code.
- Use `timeit()` to perform robust measurement.

```
clear;
N = 500;
A = rand(N);
B = magic(N);

for iRow=1:N
   for iCol=1:N
      C(iRow,iCol) = A(iRow,:)*B(:,iCol);
   end
end
```

Measure the performance

- Use `tic-toc` to measure the comp. time.
- Does pre-allocation improve the performance?
- Can you vectorize the code?
- Measure the comp. time of the vectorized code.
- Use `timeit()` to perform robust measurement.

```
clear;
N = 500;
A = rand(N);
B = magic(N);

for iRow=1:N
   for iCol=1:N
      C(iRow,iCol) = A(iRow,:)*B(:,iCol);
   end
end
```

```
t0 = tic;
  A*B;
t = toc(t0)

myFun = @() A*B;
t = timeit(myFun)
```

MATLAB Profile

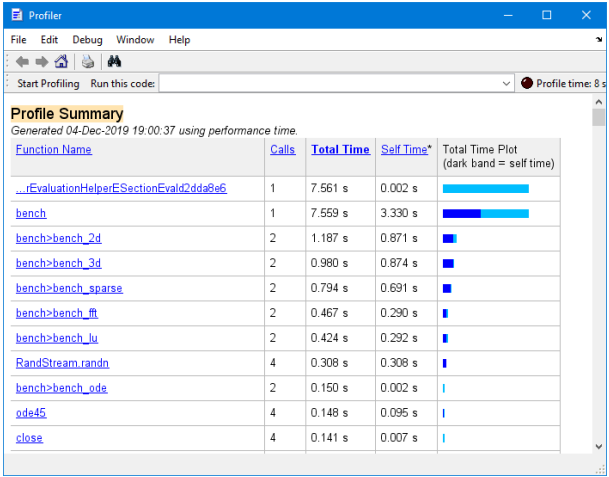
- Profile execution time for functions.
- Extremely powerful tool!

```
profile clear;
profile on;
% code to profile
profile off;
profile viewer;
```

▶ See MATLAB tips.

- Example: Profiling the `bench.m` function:

```
profile clear;
profile on;
bench();
profile off;
profile viewer;
```



Function `exist`

- The function `exist` finds out whether the given word corresponds to existing
  - `=1` variable in MATLAB workspace,
  - `=5` built-in function,
  - `=7` directory,
  - `=3` mex/dll function/library,
  - `=6` p-file,
  - `=2` m-file known to MATLAB (including user defined functions, if visible to MATLAB),
  - `=4` mdl-file,
  - `=8` class.
  - Sorted in the order of priority, returned value in bracket.

```
type = exist('sin')   % type = 5
exist('task1', 'var')  % is the task1 a variable ...
exist('task1', 'dir')  % a directory ...
exist('task1', 'file') % or a file?
```

Function `arguments` I.

- Declare specific restrictions on function arguments (inputs and outputs): constrain the class, size, and other aspects of arguments.
- Eliminate cumbersome argument-checking code and improve the readability, robustness, and maintainability of your code. ▶ See MATLAB doc.
- Simple definition:

```
function myFunction(in1)
   arguments
      in1 (Size) ClassName {Functions} = defaultValue
   end
   % Function code
end
```

- Size – the length of each dimension.
- ClassName – the name of a single MATLAB class, *e.g.*, `double`.
- Functions – A comma-separated list of validation functions. ▶ See MATLAB doc.

Function `arguments` II.

- Simple example:

```
function out1 = myFunction(in1, in2)
   arguments
      in1 (:,1) double {mustBePositive, mustBeFinite}
      in2 (:,1) double {mustBePositive}  = 1
   end

   out1 = in1 * in2.';
end
```

- Function `arguments` greatly improves code readability and robustness by handling validation at the input stage, simplifying the function logic.
- Suitable only for functions with fixed number of inputs!
- Use function `varargin` for functions with variable numbers of inputs.

Catching Errors I.

- Used particularly in the cases where unexpected event can occur:
  - in general operations with files (reading, saving),
  - evaluation of encapsulated code (function `eval`, `assignin`),
  - working with variables, properties of which (*e.g.*, size) is not yet known,
  - evaluation of code related to an object that may not exist anymore (GUI).

```
try
   % regular piece of code
catch
   % code that is evaluated if the regular code failed
end
```

- It is possible (and is recommended) to use an identifier of the error.

Catching Errors II.

- Error identifier can be used to decide what to do with the error.
  - Example: In the case of multiplication error caused by different size of vectors, it is possible to display a warning.
  - Also, the error can be later raised again either by evoking the last error occurred or as a new error with its own identifier.

```
try
   A = [1 1 1];
   B = [1 1];
   c = A.*B;
catch exc
   if strcmp(exc.identifier, 'MATLAB:sizeDimensionsMustMatch')
      disp('Mind the vector size!');
   end
   % throw(exc); % local stack shown
   % rethrow(exc); % complete stack shown
end
```

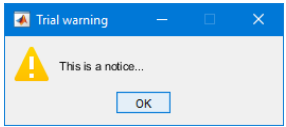
Warning Message in MATLAB

- Warning message in MATLAB is displayed using function `warning`.

```
a = 1e3;
if a > 1e2
   warning('Input coefficient has to be smaller than 10!');
end
```

  - The function is used by MATLAB, therefore, it is possible to temporarily deactivate selected internal warnings.
- Function `lastwarn` returns last warning activated.
- It is advantageous to use function `warndlg` with GUI (it just show a window, not throws the warning).

```
f = warndlg('This is a notice..', ...
   'Trial warning', 'modal');
```



Error Message in MATLAB

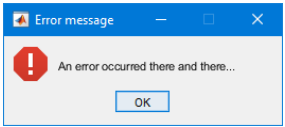
- Error message (in red color) is displayed using function `error`.

```
a = 100;
if a > 10
   error('Input has to be equal of smaller than 10!');
end
```

  - Terminates program execution.
  - Identifier can be attached.

- It is advantageous to use function `errordlg` with GUI (it just show a window, not throws the error).

```
f = errordlg('An error occurred there and there..',
...
   'Error message', 'modal');
```



is* Functions Related to Sets

- Function `issorted` returns true if array is sorted.

- Function `ismember(A,B)` tests whether elements of array `A` are also elements of array `B`.

```
ismember([1 2 3; 4 5 6; 7 8 9], [0 0 1; 2 1 4])
```

```
>> ismember([1 2 3; 4 5 6; 7 8 9], [0 0 1; 2 1 4])

ans =

  3×3 logical array

   1   1   0
   1   0   0
   0   0   0

>>
```

Summary of `is*` Functions

- Asterisk (`is*`) stands for whole range of functions...
  - They return logical (`true`/`false`) value(s).
- A selection of the interesting ones (some even have multiple parameters):

Function	Description
`ischar`	Determines whether item is character array.
`isempty`	Determines whether array is empty.
`isfinite`	Determines whether elements are of finite size.
`isnan`	Determines whether elements are `NaN`.
`isletter`	Determines whether elements are alphabetical letters (a-z, A-Z).
`islogical`	Determines whether input is logical array.
`isnumeric`	Determines whether elements are numeric values.
`isreal`	Determines whether input is real array.
`isstudent`	Determines whether MATLAB version is a student Version.
`isa`	Determine if input has specified data type.
and many others... (see `>> doc is*`)	

Function `is*`

- Try following examples...
  - Consider in what situation they could prove useful.

```
A = 'pi5_7';
B = pi;
C = [Inf NaN 5.31 true false pi];
D = [[] []];

ischar(A), isa(A,'char'), ischar(B),
isstudent, isunix, computer,
isnan(C)
isempty(C), isempty(D),
isfinite(A), isfinite(C),
isletter(A),
islogical(C), islogical([true false]),
isnumeric(A), isnumeric(C)
```

Exercises

Questions?

B0B17MTB, BE0B17MTB – MATLAB

matlab@fel.cvut.cz

April 22

Summer semester 2025/26

Acknowledgement: Filip Kozák, Pavel Valtr, Michal Mašek, Vít Losenický, and Jakub Liška.