Learning how to …

Matlab Editor

Relational and logical operators

Data type cell
Matlab Editor

- it is often wanted to evaluate certain sequence of commands repeatedly ⇒ utilization of Matlab scripts (plain ASCII coding)
- the best option is to use Matlab Editor
  - to be opened using: `>> edit`

- or in Matlab < R2012a: Start → Desktop Tools → Editor

- a script is a sequence of statements that we have been up to now typing in the command line
  - all the statements are executed one by one on the launch of the script
  - the script operates with global data in Matlab Workspace
  - suitable for quick analysis and solving problems involving multiple statements

- there are specific naming conventions for scripts (and also for functions as we see later)
Script execution, m-files

- to execute script:
  - F5 function key in Matlab Editor
  - Current Folder → select script → context menu → Run
  - Current Folder → select script → F9

- From the command line: `>> script_name`

- Scripts are stored as so called m-files
  - `.m`
  - caution: if you have Mathematica installed, the `.m` files may be launched by Mathematica
Matlab Editor, 2016b

User defined scripts and functions

% launch editor
>> edit

% open new file ‘myFce1’ in the current directory
>> edit myFce1
Useful shortcuts for Matlab Editor

<table>
<thead>
<tr>
<th>key</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTRL + Pg. UP</td>
<td>switch among all open m-files - one direction</td>
</tr>
<tr>
<td>CTRL + Pg. DOWN</td>
<td>- other direction</td>
</tr>
<tr>
<td>CTRL + R</td>
<td>adds ‘%’ at the beginning of the selected lines, “comment lines“</td>
</tr>
<tr>
<td>CTRL + T</td>
<td>removes ‘%’ from selected lines</td>
</tr>
<tr>
<td>F5</td>
<td>execute current script / function</td>
</tr>
<tr>
<td>CTRL + S</td>
<td>save current file (done automatically after pressing F5)</td>
</tr>
<tr>
<td>CTRL + HOME</td>
<td>jump to the beginning of file</td>
</tr>
<tr>
<td>CTRL + END</td>
<td>jump to the end of file</td>
</tr>
<tr>
<td>CTRL + → / ←</td>
<td>jump word-by-word or expression-by-expression to the right / left</td>
</tr>
<tr>
<td>CTRL + W</td>
<td>close current file</td>
</tr>
<tr>
<td>CTRL + O</td>
<td>activates open file dialog box (drag and drop technique also available)</td>
</tr>
<tr>
<td>CTRL + F</td>
<td>find / replace dialog box</td>
</tr>
<tr>
<td>CTRL + G</td>
<td>„go to“, jumps to the indicated line number</td>
</tr>
<tr>
<td>CTRL + D</td>
<td>open m-file of the function at the cursor's position</td>
</tr>
<tr>
<td>CTRL + I</td>
<td>indention of block of lines corresponding to key words (for/while/if/switch-case)</td>
</tr>
<tr>
<td>F1</td>
<td>open context help related to the function at position of cursor</td>
</tr>
</tbody>
</table>
Matlab Editor

- open Matlab Editor and prepare to work with a new script, call it signal1.m, for instance
- use signal generation and limiting from the previous lecture as the body of the script
- save the script in the current (or your own) folder

- try to execute the script (F5)

```
>> edit signal1

%% script generates signal with noise
clear; clc;
t = linspace(0, 5, 5*40);
s_t = sqrt(2*pi)*sin(2*pi*t) + randn(1, 5*40);
plot(t, s_t);
```

- note: from now on, the code inside scripts will be shown without leading „>>“
Useful functions for script generation

- **function `disp`** displays value of a variable in Command Window
  - without displaying variable's name and the equation sign "="
  - can be combined with `s` text (more on that later)
  - more often it is advantageous to use more complicated but robust function `sprintf`

```matlab
>> a = 2^13-1;
b = [8*a 16*a];
b = 65536 131072
```

```matlab
a = 2^13-1;
b = [8*a 16*a];
disp(b);
```

- **function `input`** is used to enter variables
  - if the function is terminated with an error, the input request is repeated

```matlab
A = input('Enter parameter A: ');
```

```matlab
>> A = input('Enter parametr A: ');% Enter parametr A: 10.153
```

```matlab
>> A = input('Enter string str: ', 's');% Enter string str: this is a test
```

```matlab
>> who
Name        Size        Bytes Class Attributes
A          1x14        28 char
ans        1x1        8 double
```

- It is possible to enter strings as well:

```matlab
str = input('Enter String str: ', 's');
```
Matlab Editor – Exercise

- create a script to calculate compound interest*
  - the problem can be described as:
    \[ P = \frac{rA \left( 1 + \frac{r}{n} \right)^{nk}}{n \left( 1 + \frac{r}{n} \right)^{nk} - 1}, \]

  where \( P \) is regular repayment of debt \( A \), paid \( n \)-times per year in the course of \( k \) years with interest rate \( r \) (decimal number)

- create a new script and save it
- at the beginning delete variables and clear Command Window
- implement the formula first, then proceed with inputs (input) and outputs (disp)
- try to vectorize the code, e.g. for various values of \( n \), \( r \) or \( k \)
- check your results (for \( A = 1000 \), \( n = 12 \), \( k = 15 \), \( r = 0.1 \) is \( P = 10.7461 \))

*interest from the prior period is added to principal
Matlab Editor – Exercise

%% script loanRepayment.m

```
clear; clc;
...
...
...
...
...
...
```

- try to vectorize the code, both for $r$ and $k$

- use scripts for future work with Matlab
  - bear in mind, however, that parts of the code can be debugged using command line

$$P = \frac{rA \left( 1 + \frac{r}{n} \right)^{nk}}{n \left( 1 + \frac{r}{n} \right)^{nk} - 1}$$
Matlab Editor – Exercise

- vectorized code for both \( r \) and \( k \)
  - `meshgrid` replicates grid vectors \( r \) and \( k \) to produce a full grid
  - `surf` creates 3D surface plot

```matlab
%% script loanRepaymentVectorized.m
clear; clc; close all
... ...
```

User defined scripts and functions
Useful functions for script generation

- **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 doubts about the code functionality exist
    
    \[ \text{K}>> \]

- **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
    
    \[
    \text{\% code; code; code; pause;}
    \]

- **see also**: echo, waitforbuttonpress
  - special purpose functions
Matlab Editor – Exercise

- modify the script for compound interest calculation in the way that
  - values $A$ and $n$ are entered from the command line (function `input`)
  - test the function `keyboard` (insert it right after parameter input)
    - is it possible to use `keyboard` mode to change the parameters inserted by `input`?
    - arrange for exiting the `keyboard (K>>)` mode, use `dbcont`
  - interrupt the script before displaying results (function `pause`)
    - note the warning „`Paused`“ in the bottom left part of main Matlab window

```matlab
%% script loanRepayment.m calculates regular repayment
clear; clc;
...
...
...
...
...
```
Script commenting

- MAKE COMMENTS!!
  - important / complicated parts of code
  - description of functionality, ideas, change of implementation

```
% A = magic(3);
matX = dataIn(:,1);
SumX = sum(matX); % all members are summed
%% CELL mode (must be enabled in Editor)
disp(num2str(SumX));
Z = inv(ZZ);
%
% This is a multi-line comment.
% Mostly, it is more appropriate to use more single-line comments.
%}
```

User defined scripts and functions

Shortcut:
CTRL+R
CTRL+T
When not making comments...

... no one will understand!

```matlab
edgTotal = MeshStruct.edgTotal;
RHO_P = zeros(3,edgTotal);
RHO_M = zeros(3,edgTotal);
for m = 1:edgTotal
    RHO_P(:,m) = repmat(MeshStruct.Rho_Plus1(:,m),[1 3]);
    RHO_M(:,m) = repmat(MeshStruct.Rho_Minus1(:,m),[1 3]);
end
Z = zeros(edgTotal,edgTotal) + l*j*zeros(edgTotal,edgTotal);
for p = 1:MeshStruct.trTotal
    Plus = find(MeshStruct.TrianglePlus - p == 0);
    Minus = find(MeshStruct.TriangleMinus - p == 0);
    D = MeshStruct.trCenter9 - ...
        repmat(MeshStruct.trCenter(:,p),[1 9 MeshStruct.trTotal]);
    R = sqrt(sum(D.*D));
    g = exp(-R/R);
    gP = g(:,MeshStruct.TrianglePlus);
    gM = g(:,MeshStruct.TriangleMinus);
    F1 = sum(gP) - sum(gM);
    ZF = FactorF1.*reshape(F1,edgTotal,1);
    for k = 1:length(Plus)
        n = Plus(k);
        RF = repmat(MeshStruct.Rho_Plus9(:,n),[1 1 edgTotal]);
        REi = repmat(MeshStruct.Rho_Minus9(:,n),[1 1 edgTotal]);
        A = sum(gP.*sum(RE.*RHO_P)) + sum(gM.*sum(RE.*RHO_M));
        Zl = FactorA.*reshape(A,edgTotal,1);
        Z(:,n) = Z(:,n) + MeshStruct.edgLength(n)*(Zl+ZF);
    end
    for k = 1:length(Minus)
        n = Minus(k);
        RF = repmat(MeshStruct.Rho_Minus9(:,n),[1 1 edgTotal]);
        REi = repmat(MeshStruct.Rho_Plus9(:,n),[1 1 edgTotal]);
        A = sum(gP.*sum(RE.*RHO_P)) + sum(gM.*sum(RE.*RHO_M));
        Zl = FactorA.*reshape(A,edgTotal,1);
        Z(:,n) = Z(:,n) + MeshStruct.edgLength(n)*(Zl-ZF);
    end
end
```

User defined scripts and functions
Cell mode in Matlab Editor

- cells enable to separate the code into smaller logically compact parts
  - separator: `%%`
- the separation is visual only, but it is possible to execute a single cell - shortcut CTRL+ENTER
Cell mode in Matlab Editor

- split previous script (*loanRepayment.m*) into separate parts
  - use the (cell) separator `%%`

```matlab
% script loanRepayment.m
clear; clc;
...
...
...
...
...
...
...
```
Data in scripts

- scripts can use data that has appeared in Workspace

- variables remain in the Workspace even after the calculation is finished

- operations on data in scripts are performed in the base Workspace
Naming conventions of scripts and functions

- names of scripts and functions
  - max. number of characters is 63 (additional characters are ignored)
  - naming restrictions similar to variable names apply
  - choose names describing what the particular function calculates
  - avoid existing names as the new script is called instead of an existing built-in function (overloading can occur)

- more information:

- in the case you want to apply vector functions row-wise
  - check whether the function enables calculation in the other dimension (max)
  - transpose your matrix
  - some of the functions work both column-wise and row-wise (sort × sortrows)
**startup.m script**

- **script** startup.m
  - always executed at Matlab start-up
  - it is possible to put your predefined constants and other operations to be executed (loaded) at Matlab start-up

- **location (use >> which startup):**
  - ...\Matlab\R201Xx\toolbox\local\startup.m

- **change of base folder after Matlab start-up:**

```matlab
%% script startup.m in ..\Matlab\Rxxx\toolbox\local\
clc;
disp('Workspace is changing to:');
cd('d:\Data\Matlab\');
cd
disp(datestr(now, 'mmmm dd, yyyy HH:MM:SS.FFF AM'));
```

Workspace is changing to:
d:\Data\Matlab

February 25, 2014  3:36:03.347 PM
Keep on working...

>>
**matlabrc.m script**

- executed at Matlab start-up (or manually executed: `>> matlabrc`)
- contains some basic definitions, e.g.
  - figure size, set-up of some graphic elements
  - sets Matlab path (see later)
  - and others
- in the case of a multi-license it is possible to insert a message in the script that will be displayed to all users at the start-up
- location (use `>> which matlabrc`):
  - `...\Matlab\R201Xx\toolbox\local\matlabrc.m`

- last of all, `startup.m` is called (if existing)

- `matlabrc.m` is to be modified only in the case of absolute urgency!
Relational operators

- to inquire, to compare, whether ‘something’ is greater than, lesser than, equal to etc.
- the result of the comparison is always either
  - positive (true), logical one „1“
  - negative (false), logical zero „0“

- all relational operators are vector-wise
  - it is possible to compare as well vectors vs. vectors, matrices vs. matrices, …

- often in combination with logical operators (see later)
  - more relational operators applied to a combination of expressions
Relational operators

- Having the vector \( \mathbf{G} = \left( \frac{\pi}{2}, \pi, \frac{3}{2}, 2\pi \right) \), find elements of \( \mathbf{G} \) that are:
  - greater than \( \pi \)
  - lesser or equal to \( \pi \)
  - not equal to \( \pi \)

- Try similar operations for \( \mathbf{H} = \mathbf{G}^T \) as well

- Try to use relational operators in the case of a matrix and scalar as well

- Find out whether \( \mathbf{V} \geq \mathbf{U} \):
  \[
  \mathbf{V} = \begin{pmatrix} -\pi & \pi & 1 & 0 \end{pmatrix}, \quad \mathbf{U} = \begin{pmatrix} 1 & 1 & 1 & 1 \end{pmatrix}
  \]
Relational operators

- find out results of following relations
- try to interpret the results

```matlab
>> 2 > 1 & 0 % ???
```

```matlab
>> r = 1/2;
>> 0 < r < 1 % ???
```

```matlab
>> (1 > A) <= true
```
Logical operators

- to enquire, to find out, whether particular condition is fulfilled
- the result is always either
  - positive (true), logical one „1“
  - negative (false), logical zero „0“

- all, any is used to convert logical array into a scalar

- Matlab interprets any numerical value except 0 as true
- all logical operators are vector-wise
  - it is possible to compare as well vectors vs. vectors, matrices vs. matrices, …

- functions is* extend possibilities of logical enquiring
  - we see later
Logical operators – application

- assume a vector of 10 random numbers ranging from -10 to 10
  \[
  \gg a = 20*\text{rand}(10, 1) - 10
  \]

- following command returns true for elements fulfilling the condition:
  \[
  \gg a < -5 \quad \% \text{ relation operator}
  \]

- following command returns values of those elements fulfilling the condition (logical indexing):
  \[
  \gg a(a < -5)
  \]

- following command puts value of -5 to the position of elements fulfilling the condition:
  \[
  \gg a(a < -5) = -5
  \]

- following command sets value of the elements in the range from -5 to 5 equal to zero (opposite to tresholding):
  \[
  \gg a(a > -5 \& a < 5) = 0
  \]

- tresholding function (values below -5 sets equal to -5, values above 5 sets equal to 5):
  \[
  \gg a(a < -5 \mid a > 5) = \text{sign}(a(a < -5 \mid a > 5))*5
  \]
Logical operators

- determine which of the elements of the vector \( A = \left( \frac{\pi}{2}, \pi, \frac{3\pi}{2}, 2\pi \right) \)

- are equal to \( \pi \) or are equal to \( 2\pi \)
  - pay attention to the type of the result (= logical values true / false)

- are greater than \( \pi/2 \) and at the same time are not equal \( 2\pi \)

- elements from the previous condition add to vector \( A \)
Logical operators: &&, ||

• in the case we need to compare scalar values only then "short-circuited" evaluation can be used

• evaluation keeps on going till a point where it makes no sense to continue
  • i.e. when evaluating

    ```matlab
    >> clear; clc;
    >> a = true;
    >> b = false;
    >> a && b && c && d
    ...
    ```

    … no problems with undefined variables c, d, because the evaluation is terminated earlier

• however:
  • terminated with error …

    ```matlab
    >> clear; clc;
    >> a = true;
    >> b = true;
    >> a && b && c && d
    ```
Logical operators

- create a row vector in the interval from 1 to 20 with step of 3
- create the vector filled with elements from the previous vector that are greater than 10 and at the same time smaller than 16; use logical operators
**Logical operators**

- create matrix \( M = \text{magic}(3) \) and find out using functions `all` and `any`
  - in which columns all elements are greater than 2
  - in which rows at least one element is greater than or equal to 8
- whether the matrix \( A \) contains positive numbers only

\[
A = \begin{pmatrix}
8 & 1 & 6 \\
3 & 5 & 7 \\
4 & 9 & 2
\end{pmatrix}
\]

\[
\begin{align*}
\text{any} \begin{pmatrix}
0 & 1 & 1 \\
1 & 1 & 0 \\
0 & 1 & 1
\end{pmatrix} &= \begin{pmatrix}
1 & 1 & 1
\end{pmatrix}, \\
\text{all} \begin{pmatrix}
0 & 1 & 1 \\
1 & 1 & 0 \\
0 & 1 & 1
\end{pmatrix} &= \begin{pmatrix}
0 & 1 & 0
\end{pmatrix}, \\
\text{any} \begin{pmatrix}
0 & 1 & 1 \\
1 & 1 & 0 \\
0 & 1 & 1
\end{pmatrix} &= \text{any} \begin{pmatrix}
0 & 1 & 0
\end{pmatrix} = 1
\end{align*}
\]
Logical operators

- find out the result of following operation and interpret it

```matlab
>> ~(~[1 2 0 -2 0])
```

- test whether variable $b$ is not equal to zero and then test whether at the same time $a / b > 3$
  - following operation tests whether both conditions are fulfilled while avoiding division by zero!
Matrix indexation using own values

- create matrix A

\[
\begin{bmatrix}
16 & 2 & 3 & 13 \\
5 & 11 & 10 & 8 \\
9 & 7 & 6 & 12 \\
4 & 14 & 15 & 1
\end{bmatrix}
\]

- first think about what will be the result of the following operation and only then carry it out

\[
>> \ N = 4; \\
>> \ A = \text{magic}(N)
\]

\[
>> \ B = A(A)
\]

- does the result correspond to what you expected?
- can you explain why the result looks the way it looks?
- notice the interesting mathematical properties of the matrix A and B
- are you able to estimate the evolution?, \( C = B(B) \)

- try similar process for \( N = 3 \) or \( N = 5 \)
Cell

• variable of type cell enables to store all types of variables (i.e. for instance variable of type cell inside another variable of type cell)

• Examples of cell:

```matlab
>> CL1 = {zeros(2), ones(3), rand(4), 'test', {NaN(1), inf(2)}}
```

• variable of type cell can be easily allocated:

```matlab
>> CL0 = cell(1,3)
```

• memory requirements is a trade-off for complexity of cell type
Cell indexing #1

- there are two possible ways of cell structure indexing
  - round brackets ( ) are used to access cells as such

- curly brackets { } are used to access data in individual cells

**Example:**

```matlab
>> CL = {[1 2; 3 4]; eye(3); 'test'}
>> CL(2:3)             % returns cells 2, 3 of CL
>> CL{1}               % returns matrix [1 2; 3 4]
>> CL{1}(2,1)          % = 3

>> CL1 = CL{1}         % CL1 is still a cell!
>> M = CL1{1}          % M is a matrix of numbers of type double
                      % double
```
Cell indexing #2

Example:

```matlab
>> CL1 = {'one','two'};
>> CL2 = {[1, 2; 3, 4], magic(3)};
>> CL = {CL1; CL2};
>> CL{2}{1}(2,1)
```

- functions to get oriented in a cell

```
>> celldisp(CL)

CL{1}{1} =
  one

CL{1}{2} =
  two

CL{2}{1} =
  1  2
  3  4

CL{2}{2} =
  8 12  6
  3  5  7
  9  2
```

- `celldisp`
- `cellplot`
Typical application of cells

- in **switch-case** branching for enlisting more possibilities
- work with variously long strings
- GUI
- all iteration algorithms with variable size of variables
- …
## Discussed functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>edit</code></td>
<td>open Matlab Editor</td>
</tr>
<tr>
<td><code>keyboard</code></td>
<td>stops execution of the file and gives control to keyboard</td>
</tr>
<tr>
<td><code>return, input</code></td>
<td>return control to invoking function, value input request</td>
</tr>
<tr>
<td><code>disp, pause</code></td>
<td>display result in command line, pauses code execution</td>
</tr>
<tr>
<td><code>num2str</code></td>
<td>conversion from datatype numeric to char</td>
</tr>
<tr>
<td><code>and, or, not, xor</code></td>
<td>functions overloading logical operators</td>
</tr>
<tr>
<td><code>all, any</code></td>
<td>evaluation of logical arrays (&quot;all of&quot;, &quot;at least one of&quot;)</td>
</tr>
<tr>
<td><code>sign</code></td>
<td>signum function</td>
</tr>
</tbody>
</table>
Exercise #1

- recall the signal from lecture 3
  - try again to limit the signal by values $s_{\text{min}}$ and $s_{\text{max}}$
  - use relational operators (\(>\) / \(<\)) and logical indexing \((s(a>b) = c)\) instead of functions \(\text{max}\), \(\text{min}\)
- solve the task item-by-item

\[
s_p(t) = \begin{cases} 
  s_{\text{min}} \iff s(t) < s_{\text{min}} \\
  s_{\text{max}} \iff s(t) > s_{\text{max}} \\
  s(t) \ldots \text{otherwise}
\end{cases}
\]

\[
\begin{align*}
N &= 5; \ V = 40; \\
t &= \text{linspace}(0, N, N*V); \\
s_t &= \text{randn}(1, N*V) + \ldots \\
sqrt{2\pi}*\sin(2\pi t);
\end{align*}
\]
Exercise #2

- consider following matrix: \( A = \begin{pmatrix} 1 & 1 & 2 \\ 2 & 3 & 5 \end{pmatrix} \)

- write a condition testing whether all elements of \( A \) are positive and at the same time all elements of the first row are integers
  - if the condition is fulfilled display the result using `disp`

```matlab
A = [1 1 2; 2 3 5];
if logicalExpr
    % display result
end
```

- compare with

  - what is the difference?
Thank you!

ver. 7.1 (13/3/2017)

Miloslav Čapek, Pavel Valtr
miloslav.capek@fel.cvut.cz
Pavel.Valtr@fel.cvut.cz

Apart from educational purposes at CTU, this document may be reproduced, stored or transmitted only with the prior permission of the authors. Document created as part of A0B17MTB course.