A0B17MTB – Matlab

Part #4

Miloslav Čapek
miloslav.capek@fel.cvut.cz

Viktor Adler, Pavel Valtr, Filip Kozák

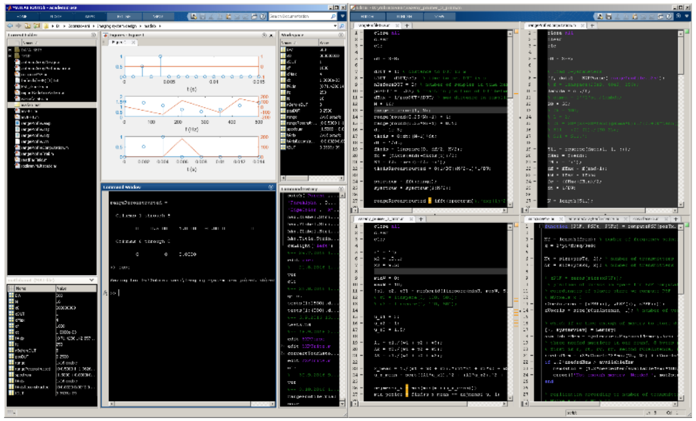
Department of Electromagnetic Field
B2-634, Prague

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 $\Rightarrow$ utilization of Matlab scripts (plain ACSII coding)
- the best option is to use `Matlab Editor`
  - to be opened using:

```
>> edit
```

- 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

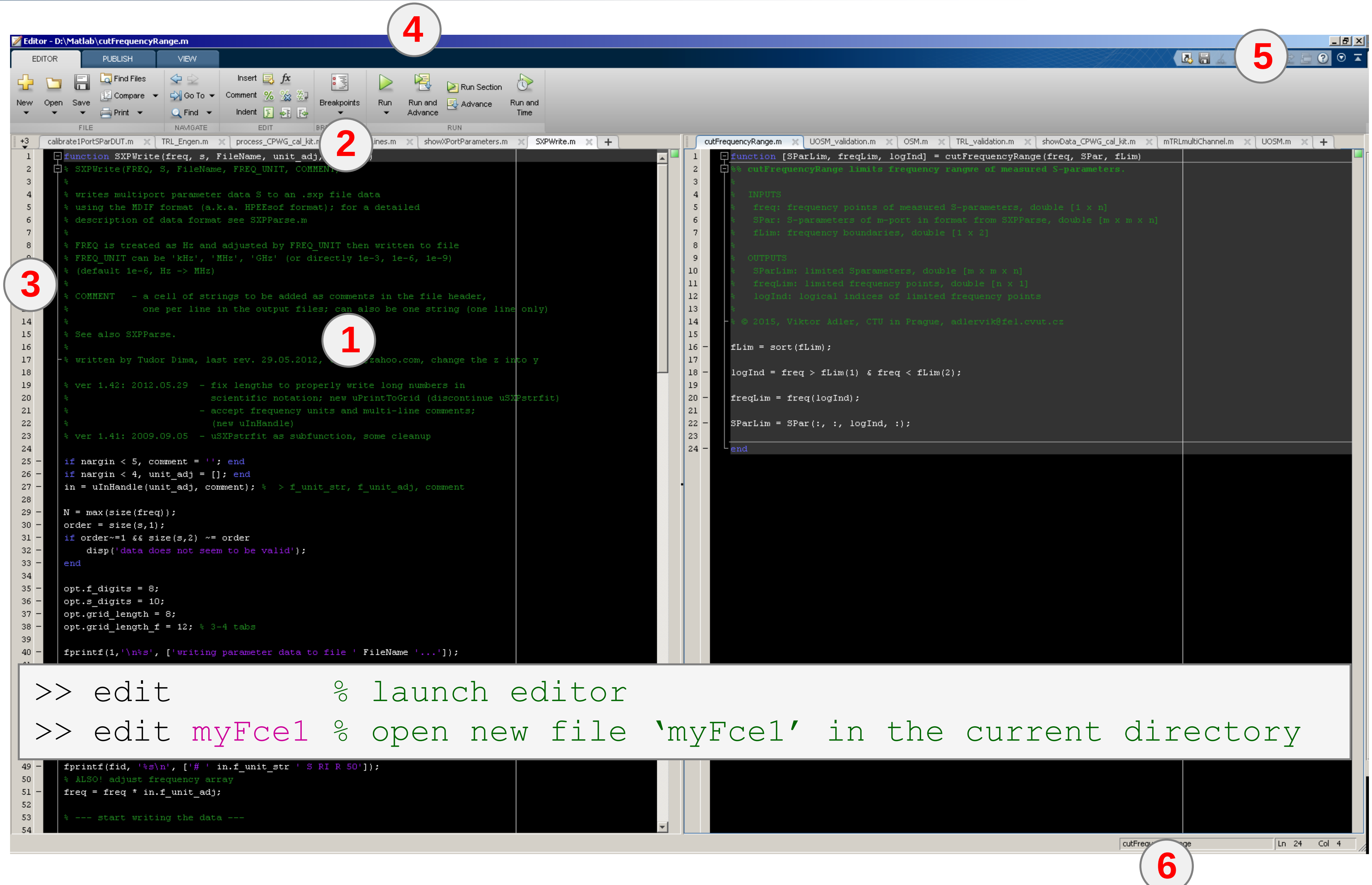
  - From the command line:

```
>> 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

Department of Electromagnetic Field, CTU FEE, miloslav.capek@fel.cvut.cz

Matlab Editor, R2016b




```
>> edit          % launch editor
>> edit myFce1   % open new file 'myFce1' in the current directory
```

Useful shortcuts for Matlab Editor

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

Matlab Editor

120 s

- 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;
N = 5; V = 40; T = 1;
t = linspace(0, N*T, N*V);
s_t = sqrt(2*pi)*sin(2*pi*t) + randn(1, N*V);
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`

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

b =

      65528      131056
```

```
a = 2^13-1;
b = [8*a 16*a];
b
```

vs.

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

```
>> a = 2^13-1;
b = [8*a 16*a];
disp(b);
       65528      131056
```

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

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

  - It is possible to enter strings as well:

```
str = input('Enter String str: ', 's');
```

```
>> A = input('Enter parametr A: ');
Enter parametr A: 10.153
>> A = input('Enter string str: ', 's');
Enter string str: this is a test
>> whos
  Name      Size            Bytes  Class     Attributes

  A         1x14               28  char
  ans       1x1                 8  double
```

Matlab Editor – Exercise

600 s

- create a script to calculate compound interest*
  - the problem can be described as :

$$P = \frac{rA\left(1+\frac{r}{n}\right)^{nk}}{n\left(\left(1+\frac{r}{n}\right)^{nk}-1\right)},$$

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

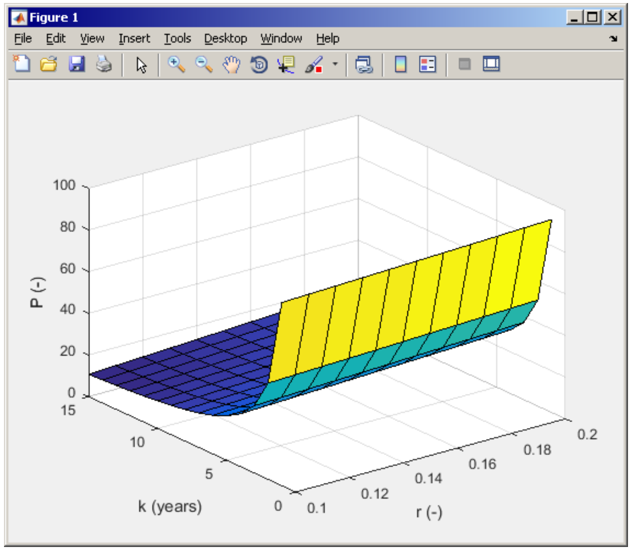
$$P = \frac{rA\left(1+\frac{r}{n}\right)^{nk}}{n\left(\left(1+\frac{r}{n}\right)^{nk}-1\right)}$$

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

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

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



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

```
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

Department of Electromagnetic Field, CTU FEE, miloslav.capek@fel.cvut.cz

Matlab Editor – Exercise

360 s

- 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

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

Script commenting

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

typical comment (one-/multiple- line)

enables to separate function into more blocs (%% …)

Multiple-line comment

```
% 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.
%}
```

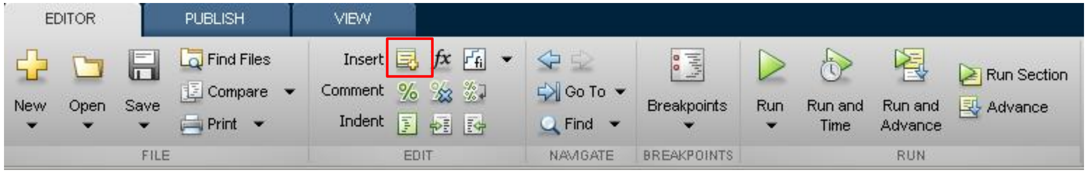
Shortcuts:
CTRL+R
CTRL+T

When not making comments…

- … no one will understand!

```
edgTotal  = MeshStruct.edgTotal;
RHO_P     = zeros(3,9,edgTotal);
RHO_M     = zeros(3,9,edgTotal);
for m = 1:edgTotal
    RHO_P(:,:,m) = repmat(MeshStruct.Rho_Plus1(:,m),[1 9]);
    RHO_M(:,:,m) = repmat(MeshStruct.Rho_Minus1(:,m),[1 9]);
end
Z         = zeros(edgTotal,edgTotal) + 1j*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(-K*R)./R;
    gP    = g(:,:,MeshStruct.TrianglePlus);
    gM    = g(:,:,MeshStruct.TriangleMinus);
    Fi    = sum(gP) - sum(gM);
    ZF    = FactorFi.*reshape(Fi,edgTotal,1);
    for k = 1:length(Plus)
        n      = Plus(k);
        RP     = repmat(MeshStruct.Rho_Plus9(:,:,n),[1 1 edgTotal]);
        RPi    = repmat(MeshStruct.Rho_Minus9(:,:,n),[1 1 edgTotal]);
        A      = sum(gP.*sum(RP.*RHO_P)) + sum(gM.*sum(RP.*RHO_M));
        Z1     = FactorA.*reshape(A,edgTotal,1);
        Z(:,n) = Z(:,n) + MeshStruct.edgLength(n)*(Z1+ZF);
    end
    for k = 1:length(Minus)
        n      = Minus(k);
        RP     = repmat(MeshStruct.Rho_Minus9(:,:,n),[1 1 edgTotal]);
        RPi    = repmat(MeshStruct.Rho_Plus9(:,:,n),[1 1 edgTotal]);
        A      = sum(gP.*sum(RP.*RHO_P)) + sum(gM.*sum(RP.*RHO_M));
        Z1     = FactorA.*reshape(A,edgTotal,1);
        Z(:,n) = Z(:,n) + MeshStruct.edgLength(n)*(Z1-ZF);
    end
end
```

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

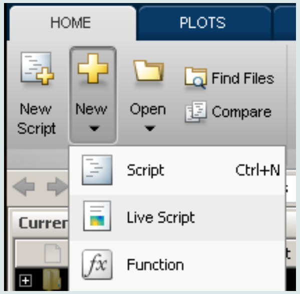
240 s

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

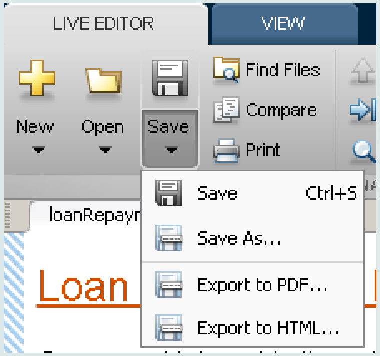
```
% script loanRepayment.m
clear; clc;
...
...
...
...
...
...
...
...
...
...
...
...
```

Live Script

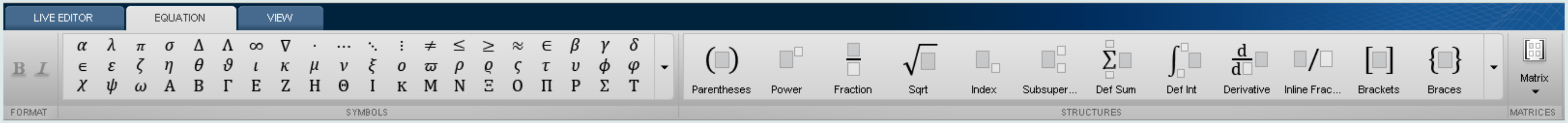
- In Matlab from R2016a
- Live script can contain code, generated output, formatted text, images, hyperlinks, equations, ...
  - it is necessary to use Live Editor
  - HOME → New → Live Script
  - editor creates *.mlx files



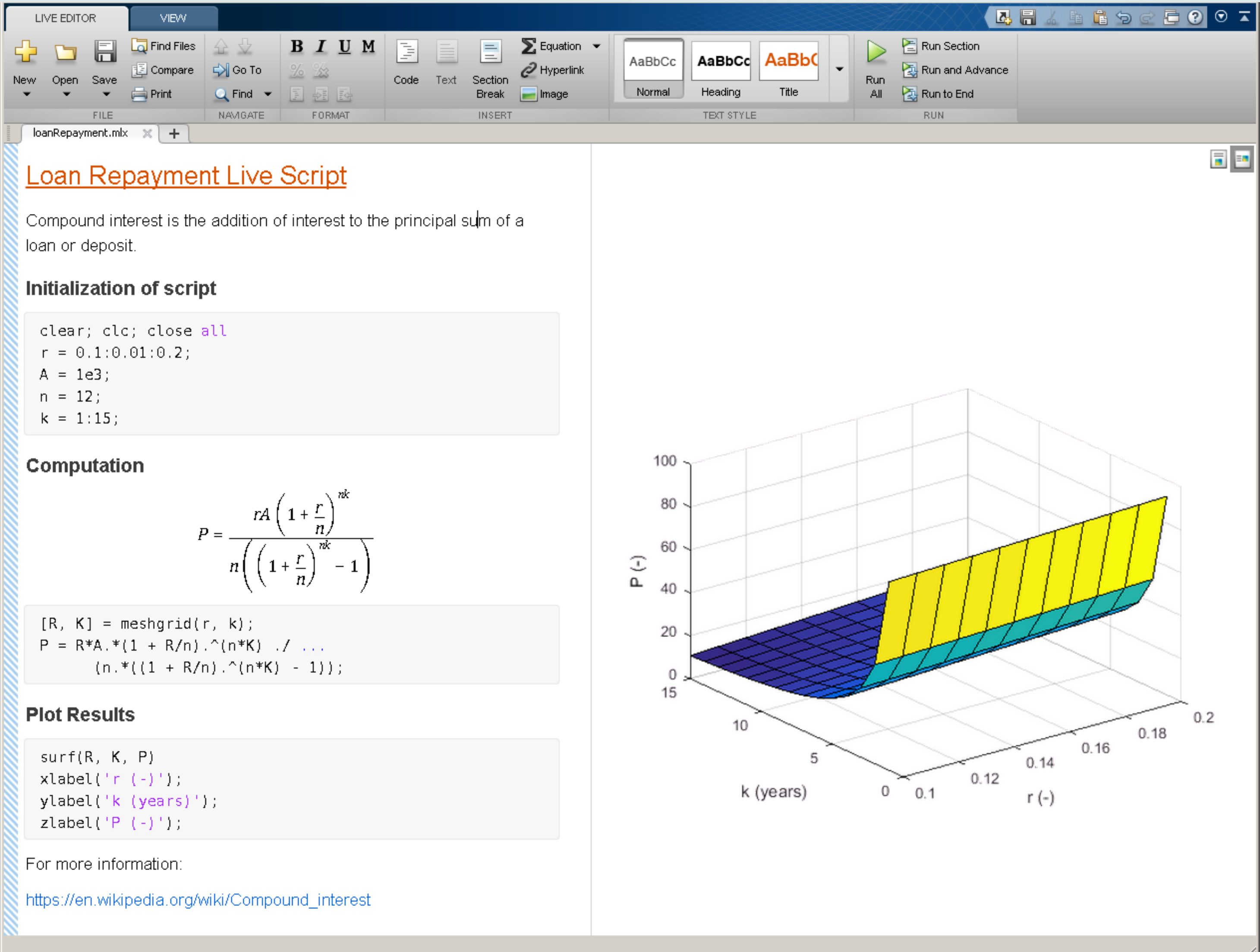
- Export options: PDF, HTML



- Internal extensive equation editor



Live Script


LIVE EDITOR
VIEW
New
Open
Save
Find Files
Compare
Print
Go To
Find
Code
Text
Section Break
Equation
Hyperlink
Image
Normal
Heading
Title
Run All
Run Section
Run and Advance
Run to End
FILE
NAVIGATE
FORMAT
INSERT
TEXT STYLE
RUN
loanRepayment.mlx
Loan Repayment Live Script
Compound interest is the addition of interest to the principal sum of a loan or deposit.
Initialization of script
clear; clc; close all
r = 0.1:0.01:0.2;
A = 1e3;
n = 12;
k = 1:15;
Computation
[R, K] = meshgrid(r, k);
P = R*A.*(1 + R/n).^(n*K) ./ ...
(n.*((1 + R/n).^(n*K) - 1));
Plot Results
surf(R, K, P)
xlabel('r (-)');
ylabel('k (years)');
zlabel('P (-)');
For more information:
https://en.wikipedia.org/wiki/Compound_interest
P (-)
k (years)
r (-)

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

Department of Electromagnetic Field, CTU FEE, miloslav.capek@fel.cvut.cz

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:
  - `http://www.mathworks.com/matlabcentral/fileexchange/2529-matlab-programming-style-guidelines`

- 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 :

```
%% 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!

Department of Electromagnetic Field, CTU FEE, miloslav.capek@fel.cvut.cz

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"

>	greater than
>=	greater than or equal to
<	lesser than
<=	lesser than or equal to
==	equal to
~=	not equal to

- 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

300 s

- having the vector $\mathbf{G} = \begin{pmatrix} \frac{\pi}{2} & \pi & \frac{3}{2}\pi & 2\pi \end{pmatrix}$, find elements of **G** that are
  - greater than π
  - lesser or equal to π
  - not equal to π
- try similar operations for $\mathbf{H} = \mathbf{G}^{\mathrm{T}}$ as well
- try to use relational operators in the case of a matrix and scalar as well
- find out whether **V** ≥ **U**:

$$\mathbf{V} = \begin{pmatrix} -\pi & \pi & 1 & 0 \end{pmatrix}$$

$$\mathbf{U} = \begin{pmatrix} 1 & 1 & 1 & 1 \end{pmatrix}$$

Relational operators

200 s

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

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

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

```
>> (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

`&`	`and`
`|`	`or`
`~`	`not`
	`xor`
	`all`
	`any`

Department of Electromagnetic Field, CTU FEE, miloslav.capek@fel.cvut.cz

Logical operators – application

- assume a vector of 10 random numbers ranging from -10 to 10

```
>> a = 20*rand(10, 1) - 10
```

- following command returns `true` for elements fulfilling the condition:

```
>> a < -5 % relation operator
```

- following command returns values of those elements fulfilling the condition (logical indexing):

```
>> a(a < -5)
```

- following command puts value of -5 to the position of elements fulfilling the condition :

```
>> a(a < -5) = -5
```

- following command sets value of the elements in the range from -5 to 5 equal to zero (opposite to tresholding):

```
>> a(a > -5 & a < 5) = 0
```

- tresholding function (values below -5 sets equal to -5, values above 5 sets equal to 5):

```
>> a(a < -5 | a > 5) = sign(a(a < -5 | a > 5))*5
```

Logical operators

420 s

- determine which of the elements of the vector $\mathbf{A} = \begin{pmatrix} \frac{\pi}{2} & \pi & \frac{3}{2}\pi & 2\pi \end{pmatrix}$

  - are equal to π <u>or</u> are equal to 2π
    - pay attention to the type of the result (= logical values `true` / `false`)
  - are greater than π/2 <u>and at the same time</u> are not equal 2π
  - concatenate elements from the previous condition to vector A

Logical operators

150 s

- 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

Department of Electromagnetic Field, CTU FEE, miloslav.capek@fel.cvut.cz

Logical operators

240 s

- create matrix `M = 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 M contains positive numbers only

$$\mathbf{M} = \begin{pmatrix} 8 & 1 & 6 \\ 3 & 5 & 7 \\ 4 & 9 & 2 \end{pmatrix}$$

```
>> M = magic(3);
>> all(M > 2)
>> any(M >= 8, 2)
>> all(all(M > 0))
>> all(M(:) > 0)
```

$$\text{any}\begin{pmatrix} 0 & 1 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 \end{pmatrix}, \quad \text{all}\begin{pmatrix} 0 & 1 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 \end{pmatrix}, \quad \text{any}\left(\text{all}\begin{pmatrix} 0 & 1 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix}\right) = \text{any}\begin{pmatrix} 0 & 1 & 0 \end{pmatrix} = 1$$

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

```
>> clear;
>> 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 …

```
>> clear;
>> a = true;
>> b = true;
>> a && b && c && d
```


Department of Electromagnetic Field, CTU FEE, miloslav.capek@fel.cvut.cz

Logical operators

240 s

- find out the result of following operation and interpret it

```
>> ~(~[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!

Department of Electromagnetic Field, CTU FEE, miloslav.capek@fel.cvut.cz

Matrix indexation using own values

300 s

- create matrix `A`

```
>> N = 4;
>> A = magic(N)
```

```
A =

    16     2     3    13
     5    11    10     8
     9     7     6    12
     4    14    15     1
```

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

```
>> 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`

Department of Electromagnetic Field, CTU FEE, miloslav.capek@fel.cvut.cz

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:

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

  - variable of type cell can be easily allocated:

```
>> 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.:

```
>> 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
```

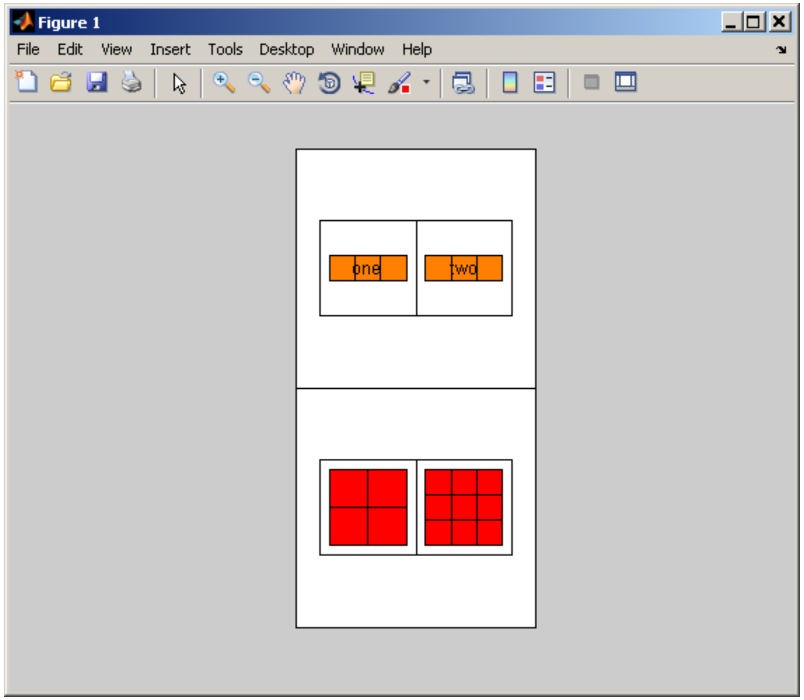

Department of Electromagnetic Field, CTU FEE, miloslav.capek@fel.cvut.cz

Cell indexing #2

- Example.:

```
>> 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
  - cellplot




```
>> celldisp(CL)

CL{1}{1} =

one


CL{1}{2} =

two


CL{2}{1} =

     1     2
     3     4



CL{2}{2} =

     8     1     6
     3     5     7
     4     9     2
```

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
- …

Department of Electromagnetic Field, CTU FEE, miloslav.capek@fel.cvut.cz

Discussed functions

`edit`	open Matlab Editor	●
`keyboard`	stops execution ot the file and gives control to keyboard	●
`return, input`	return control to invoking function, value input request	●
`disp, pause`	display result in command line, pauses code execution	●
`num2str`	conversion from datatype `numeric` to `char`	●
`and, or, not, xor`	functions overloading logical operators	
`all, any`	evaluation of logical arrays („all of“, „at least one of“)	●
`sign`	signum function	

Exercise #1

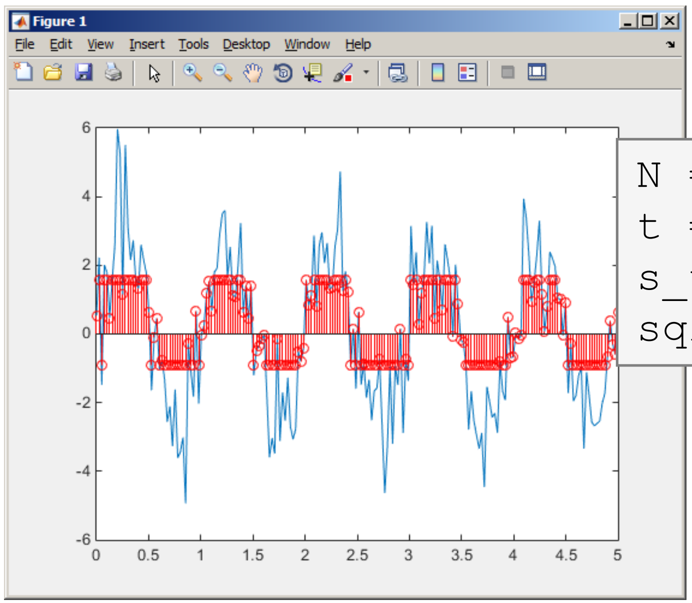
360 s

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

$$s_{\mathrm{p}}(t)=\begin{cases} s_{\mathrm{min}} \Leftrightarrow s(t)<s_{\mathrm{min}} \\ s_{\mathrm{max}} \Leftrightarrow s(t)>s_{\mathrm{max}} \\ s(t) \ldots \text{otherwise} \end{cases}$$

$$s_{\mathrm{min}}=-\frac{9}{10}$$

$$s_{\mathrm{max}}=\frac{\pi}{2}$$




```
N = 5; V = 40;
t = linspace(0, N, N*V);
s_t = randn(1, N*V) + ...
sqrt(2*pi)*sin(2*pi*t);
```

Exercise #2

300 s ↑

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

- 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`

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

- compare with

  - what is the difference?

Thank you!

ver. 9.1 (12/03/2018)
Miloslav Čapek, Pavel Valtr
miloslav.capek@fel.cvut.cz
Pavel.Valtr@fel.cvut.cz

Document created as part of A0B17MTB course.