AE0B17MTB – Matlab

Part #12

Miloslav Čapek
miloslav.capek@fel.cvut.cz
Viktor Adler, Pavel Valtr, Filip Kozák

Department of Electromagnetic Field
B2-634, Prague

Learning how to …

Data types `struct`

Import / export in Matlab

Time functions

`warning`, `error`, `try-catch`

`categorical`, `table`

Structured variable, `struct`

- data are stored in variables that are grouped in one structure
- concept is similar to OOP (without features of OOP)
- Ex. inventory:

```
>> stock(1).id = 1;
>> stock(1).thing = 'fridge';
>> stock(1).price = 750;
>> stock(1).units = 'USD';
>> stock(2).id = 2;
>> stock(2).thing = 'Bowmore_12yr';
>> stock(2).price = 1100;
>> stock(2).units = 'CZK';
>> stock
```

- or:

```
>> stock = struct('id', {1, 2}, 'thing', ...
        {'fridge', 'Bowmore_12yr'}, ...
        'price', {'750', '1100'}, 'units', {'USD', 'CZK'})
```

Functions for work with structures

- new field creation
  - direct command

```
>> stock(1).newField = 'test';
```

  - field name as a string

```
>> setfield(stock(1), 'newField', 'test')
```

```
>> stock(1).('newField2') = 'test2'
```

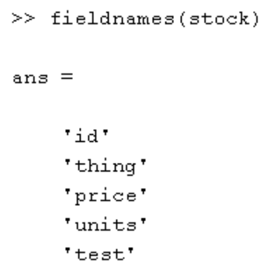
- setting field value
  - direct command

```
>> stock(1).id = 3;
```

  - field name and value

```
>> stock(1).('id') = 3;
```

Functions for work with structures



```
>> fieldnames(stock)

ans =

    'id'
    'thing'
    'price'
    'units'
    'test'
```

- list of all fields of structure – `fieldnames`

```
>> fieldnames(stock)
```

- value of given field

```
>> id2 = stock(2).id
>> id2 = stock(2).('id')
>> id2 = getfield(stock(2), 'id')
```

- does given field exist?

```
>> isfield(stock, 'id')   % = 1
>> isfield(stock, 'ID')   % = 0
```

- is given variable a structure?

```
>> isstruct(stock)        % = 1
```

Functions for work with structures

- delete field

```
>> rmfield(stock, 'id')
```

- more complex indexing of structures
  - structure may have more levels

```
>> stock(1).subsection(1).order = 1
>> stock(1).subsection(2).order = 2
```

  - it is possible to combine cells with structures

```
>> stock(1).subsection(3).check = [1; 2]
>> K{1} = stock;
```

  - certain fields can be indexed using name stored as a string

```
>> K{1}(1).subsection(3).('check')(2)
```

Functions for work with structures

- getting data from fields of structure array
  - comma-separated list (doc Comma-Separated Lists)

```
>> stock.id
```

  - concatenate values to vector

```
>> allIDs = [stock.id] % row vector
>> allIDs = horzcat(stock.id) % row vector
>> allIDs = vertcat(stock.id) % column vector
```

  - concatenate strings to cell array

```
>> allThings = [stock.thing] % useless
>> allThings = vertcat(stock.thing) % error
>> allThings = {stock.thing} % cell array
```

  - create multiple variables

```
>> allThings = {stock.thing} % cell array
>> [th1, th2] = allThings{:}
```

Functions for work with structures

- set data to fields of structure array
  - for cycle

```
IDs = [2 3];
for iStruct = 1:length(stock)
   stock(iStruct).id = IDs(iStruct);
end
```

  - utilizing comma-separated list

```
IDs = {2 3};
[stock.id] = IDs{:};
```

- creating multidimensional structure

```
>> stock(2, 2).thing = 'multi dim.'
```

```
>> allThings = reshape({stock.thing}, size(stock)).'
```

Typical application of structure

- export of data to Matlab
- all complex internal variables (exceptions, errors, ...)
- `callbackdata` (`event`) wit GUI (up to R2014a)

Data Import and export

- Matlab supports wide range of file formats
  - `mat`, `txt`, `xls`, `jpeg`, `bmp`, `png`, `wav`, `avi` and others, see
    - Matlab → Data Import and Analysis→ Data Import and Export → Supported File Formats for Import and Export
  - packages exist for work with, for instance, `dwg` and similar formats
  - it is possible to read a general file containing ASCII characters as well

- in this course we shall see how to
  - read data from file, read image, read files line by line
  - store in file, write in file
  - import from Excel
  - export to Excel

Data Import and export

- following can be applied to whole group of formats
  - old Matlab: use `File` → `Import Data`
  - new Matlab: `Home` → `Import Data`

  - command `uiimport` + following interface
  - file drag and drop to Workspace window

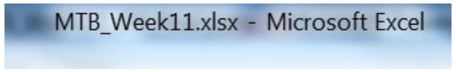
- for storing in various formats see following functions
  - `save`, `dlmwrite`, `xlswrite`, `imwrite`, `audiowrite`, `...`

Import from Excel

- use funcrion `xlsread` to import
  - alternativelly, use aforementioned function `uiimport`

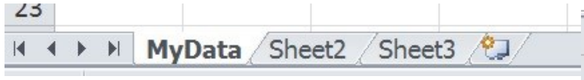
```
>> num = xlsread('MTB_Week11.xls', 'MyData', 'A1:B4');
```

File name
(has to be visible to Matlab)



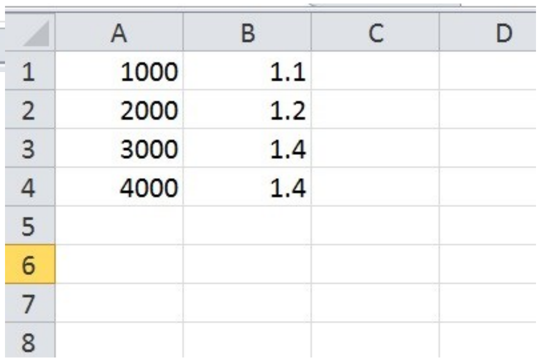
MTB_Week11.xlsx - Microsoft Excel

name of the file's sheet



MyData Sheet2 Sheet3

range of cells



	A	B	C	D
1	1000	1.1		
2	2000	1.2		
3	3000	1.4		
4	4000	1.4		
5				
6				
7				
8				

Import from Excel

420 s

- read all numerical data from Excel file on course's webpage
  - thereafter plot dependence of values in column `values` on values in column `experiment`
  - verify the size of data read

Export to Excel

- function `xlswrite` is used to export data from Matlab to Excel
  - example: write data `fx` in file `file.xlsx` in sheet `Sheet1` in line 1 starting with column `A`

```
>> fx = 1:10;
>> xlswrite('file.xlsx', fx, 1, 'A1');
```

  - example: write data `fx` in file `file2.xlsx` in sheet `NewSheet` in column `B` starting with line 1

```
>> fx = 1:10;
>> xlswrite('file2.xlsx', fx', 'NewSheet', 'B1');
```

Export to Excel

420 s ↑

- evaluate function $f(x) = \cos(x) + \dfrac{\cosh(x)}{10}$ on the interval $x \in \langle -\pi, \pi \rangle$ with step 0.01
  - resulting variables *x* and *f*(*x*) write to file `Excel_file.xlsx` in 1st sheet, variable *x* is in column `A`, variable *f*(*x*) is in column `B`
  - verify whether data written in the sheet are correct

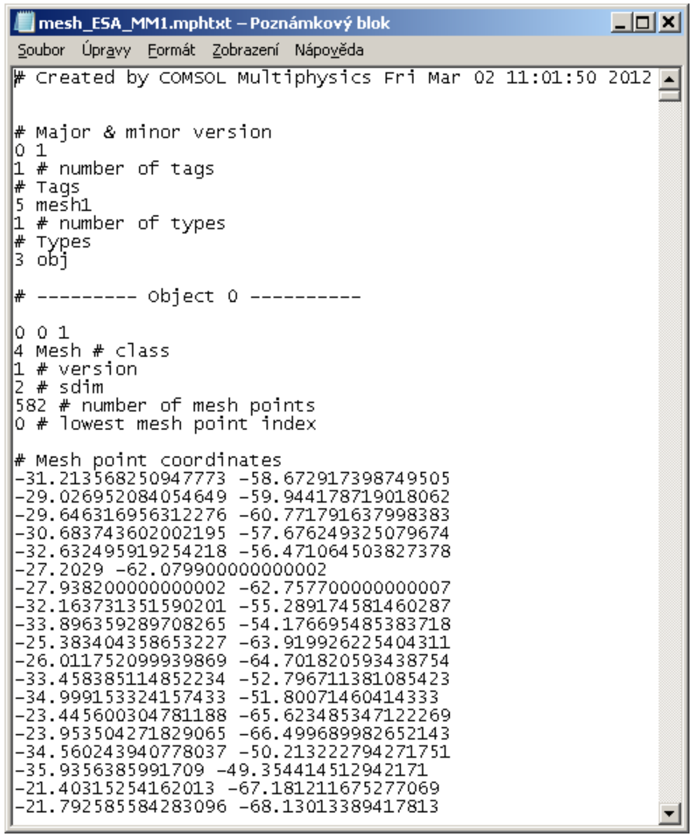
Reading binary data from file #1

- we will be using what we learned earlier (`while`, `str2double`, …)
  - on top of that the file has to be opened (`fopen`) and closed afterwards (`fclose`)

```
>> fid = fopen('mesh_ESA_MM1.mphtxt');
```

```
% allocation
while ~feof(fid)
    % reading
end
```

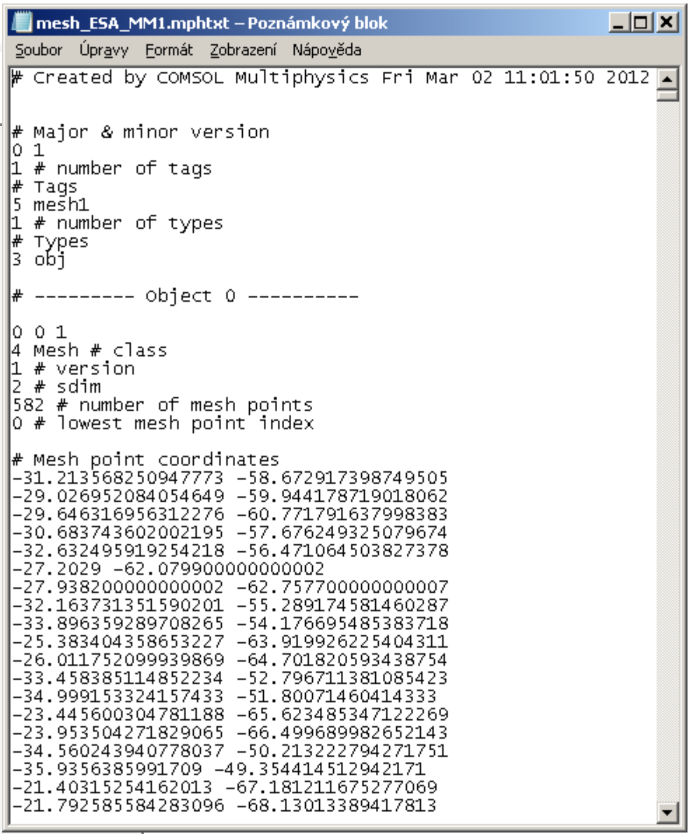
```
>> fclose(fid);
```


mesh_ESA_MM1.mphtxt – Poznámkový blok
Soubor Úpravy Formát Zobrazení Nápověda

```
# Created by COMSOL Multiphysics Fri Mar 02 11:01:50 2012

# Major & minor version
0 1
1 # number of tags
# Tags
5 mesh1
1 # number of types
# Types
3 obj

# --------- Object 0 ----------

0 0 1
4 Mesh # class
1 # version
2 # sdim
582 # number of mesh points
0 # lowest mesh point index

# Mesh point coordinates
-31.213568250947773 -58.672917398749505
-29.026952084054649 -59.944178719018062
-29.646316956312276 -60.771791637998383
-30.683743602002195 -57.676249325079674
-32.632495919254218 -56.471064503827378
-27.2029 -62.079900000000002
-27.938200000000002 -62.757700000000007
-32.163731351590201 -55.289174581460287
-33.896359289708265 -54.176695485383718
-25.383404358653227 -63.919926225404311
-26.011752099939869 -64.701820593438754
-33.458385114852234 -52.796711381085423
-34.999153324157433 -51.80071460414333
-23.445600304781188 -65.623485347122269
-23.953504271829065 -66.499689982652143
-34.560243940778037 -50.213222794271751
-35.9356385991709 -49.354414512942171
-21.40315254162013 -67.181211675277069
-21.792585584283096 -68.13013389417813
```

Reading binary data from file #2

```
clc; clear;
fid = fopen('mesh_ESA_MM1.mphtxt');
start = false;
Data  = [];
k     = 1;
while ~feof(fid)
    line  = fgetl(fid);
    if start && isempty(line)
        break
    end
    if start
        data        = str2num(line);
        Data(k, :)  = data;
        k           = k + 1;
    end
    if strcmp(line,'# Mesh point coordinates')
       start = true;
    end
end
fclose(fid);
```


mesh_ESA_MM1.mphtxt – Poznámkový blok
Soubor Úpravy Formát Zobrazení Nápověda

```
# Created by COMSOL Multiphysics Fri Mar 02 11:01:50 2012


# Major & minor version
0 1
1 # number of tags
# Tags
5 mesh1
1 # number of types
# Types
3 obj

# --------- Object 0 ----------

0 0 1
4 Mesh # class
1 # version
2 # sdim
582 # number of mesh points
0 # lowest mesh point index

# Mesh point coordinates
-31.213568250947773 -58.672917398749505
-29.026952084054649 -59.944178719018062
-29.646316956312276 -60.771791637998383
-30.683743602002195 -57.676249325079674
-32.632495919254218 -56.471064503827378
-27.2029 -62.079900000000002
-27.938200000000002 -62.757700000000007
-32.163731351590201 -55.289174581460287
-33.896359289708265 -54.176695485383718
-25.383404358653227 -63.919926225404311
-26.011752099939869 -64.701820593438754
-33.458385114852234 -52.796711381085423
-34.999153324157433 -51.800714604143333
-23.445600304781188 -65.623485347122269
-23.953504271829065 -66.499689982652143
-34.560243940778037 -50.213222794271751
-35.9356385991709 -49.354414512942171
-21.40315254162013 -67.181211675277069
-21.792585584283096 -68.13013389417813
```

```
>> size(Data)

ans =

   582     2
```

Writing to a file #1

- we try to write variable `Data` from a file `data.mat` where the first line contains a header

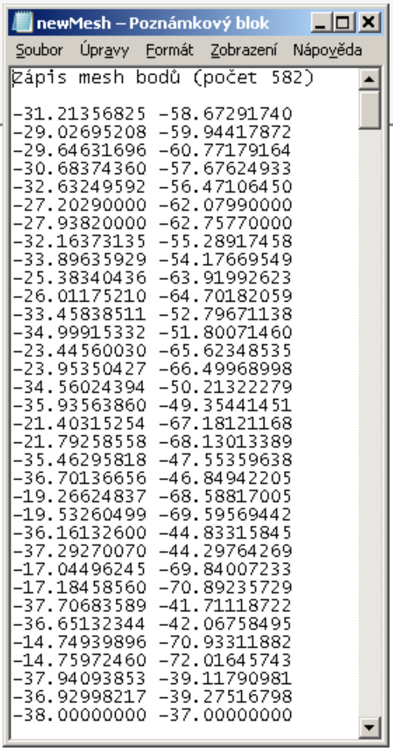
```
>> fid = fopen('newMesh.txt');
```

```
for k = 1:size(Data,1)
    fprintf(fid, '%3.8f %3.8f\r\n', Data(k, :));
end
```

```
>> fclose(fid);
```

Writing to a file #2

```
clc;
fid = fopen('newMesh.txt', 'w');
fprintf(fid, 'Mesh points (number %3.0f)\r\n\r\n', size(Data,1));
for k = 1:size(Data, 1)
    fprintf(fid, '%3.8f %3.8f\r\n', Data(k, :));
end
fclose(fid);
```


newMesh – Poznámkový blok
Soubor Úpravy Formát Zobrazení Nápověda

```
Zápis mesh bodů (počet 582)

-31.21356825 -58.67291740
-29.02695208 -59.94417872
-29.64631696 -60.77179164
-30.68374360 -57.67624933
-32.63249592 -56.47106450
-27.20290000 -62.07990000
-27.93820000 -62.75770000
-32.16373135 -55.28917458
-33.89635929 -54.17669549
-25.38340436 -63.91992623
-26.01175210 -64.70182059
-33.45838511 -52.79671138
-34.99915332 -51.80071460
-23.44560030 -65.62348535
-23.95350427 -66.49968998
-34.56024394 -50.21322279
-35.93563860 -49.35441451
-21.40315254 -67.18121168
-21.79258558 -68.13013389
-35.46295818 -47.55359638
-36.70136656 -46.84942205
-19.26624837 -68.58817005
-19.53260499 -69.59569442
-36.16132600 -44.83315845
-37.29270070 -44.29764269
-17.04496245 -69.84007233
-17.18458560 -70.89235729
-37.70683589 -41.71118722
-36.65132344 -42.06758495
-14.74939896 -70.93311882
-14.75972460 -72.01645743
-37.94093853 -39.11790981
-36.92998217 -39.27516798
-38.00000000 -37.00000000
```

Warning message in Matlab – `warning`

- warning message in Matlab is displayed using function `warning`

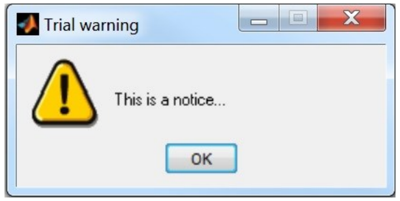
```
a = 1e3;
if a > 1e2
    warning('Input coefficient has to be smaller than 10!');
end
```

  - 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
  - but it is just a statement really, see last lecture

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


Trial warning

This is a notice...

Error message in Matlab – `error`

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

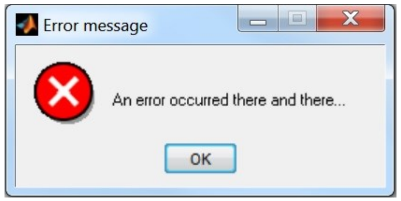
```
a = 100;
if a > 10
    error('Input has to be equal of smaller than 10!');
end
```

  - terminates program execution
  - identifier can be attached

```
error('Input has to be equal of smaller than 10!');
```

- it is advantageous to use function `errordlg` with GUI
  - but it is just a statement really, see last lecture

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



An error occurred there and there...

Catching errors #1

- 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 #2

- 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:dimagree')
        disp('Mind the vector size!');
    end
    % throw(exc); % local stack shown
    % rethrow(exc); % complete stack shown
end
```

Time functions in Matlab

- there is a whole variety of time functions but just one of them is enough to measure time duration of a function

Function	Description
`tic - toc`	measure length of time interval between expressions `tic` and `toc`
`clock`	return six element vector `[year month day hour minute seconds]`
`date`	return date in format `dd-mmm-yyyy`, variable is of type `char` (text)
`etime`	return time interval between `t1` and `t2`, `etime(t2,t1)`
`cputime`	return total CPU time in seconds used by Matlab application
`now`	return current date and time as an integer
`timeit`	measure time required to run function (new from R2013b, originally from fileexchange)

```
>> tic
>>    %% code
>> toc
```

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

Time functions in Matlab – an example

- what is the way to measure how long it takes for a program to be executed?
  - more time consuming code × very fast code

```
tic
    % code
toc
```

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

- other options – which one is the best?
- Mathworks recommends functions `tic-toc` mainly for ≥P4@hyperthreading

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

```
t0b = clock;
fft(x);
etime(clock, t0b)
```

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

Time functions in Matlab – specialties

- conversions between individual ways of displaying date in Matlab
  - `datavec`, `datanum`, `datastr`
  - this is how to transform date into standard form

```
>> datevec(now)
```

- day of week:

```
>> weekday(date)
```

  - caution, US way of counting days (Saturday ~ last day of the week)

- last day of month:

```
>> eomday(2014, 1:12)
```

- calendar

```
>> calendar
```

  - caution, last day of month is Saturday again!

Time functions in Matlab

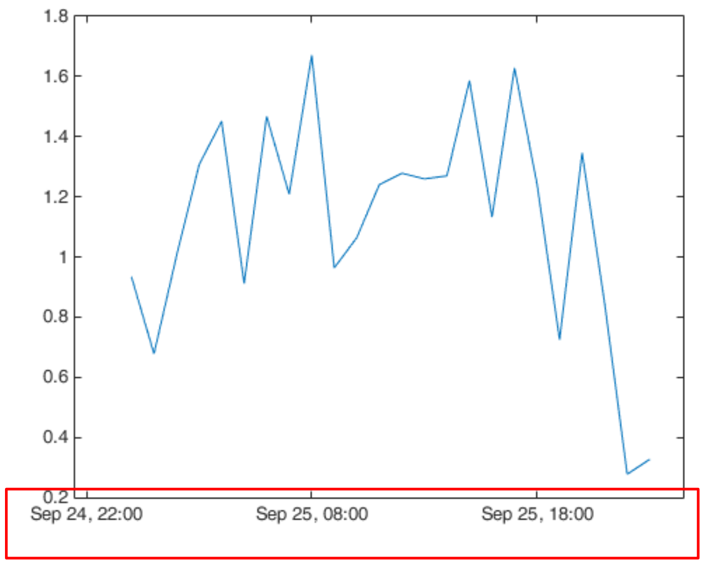
420 s ↑

- Try to implement selected time functions into your project

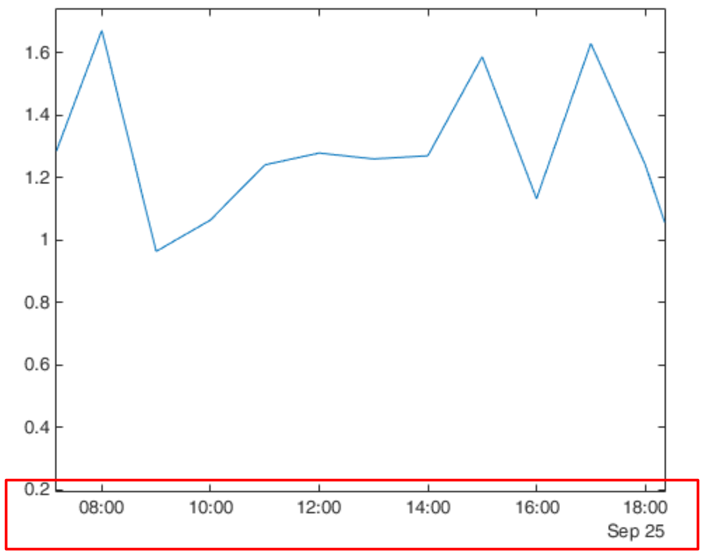
Time series data

- having data as a function of time, it is possible to display the data as a time series

```
>> d  = datetime(2015, 9, 25, 0:23, 0, 0);
>> fx = sin(linspace(0, pi, 24)) + rand(1,24);
>> plot(d, fx)
```



zoom



- for more details see:
  Matlab → Language Fundamentals → Data Types → Dates and Time

Time data

- time entries created using `datetime` are not of class `double`, but of class `datetime`
  - it is possible to specify time zones/difference
  - all functions support vectorizing

```
>> t = datetime
>> t.Format
>> % nonsense but possible:
>> t.Format = 'd-h'
```

```
>> t1 = datetime('22/09/15 17:00:00');
>> t2 = datetime('24/12/15 19:00:00');
>> t = t1:days(7):t2
```

- it is possible to create and work with time intervals as well (class `duration`)

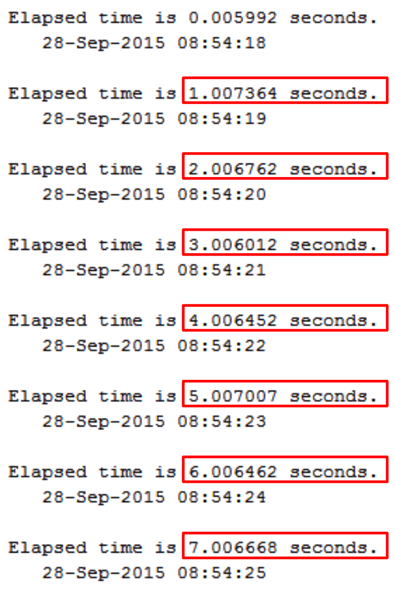
```
>> tInt = days(5) + hours(10)
>> 2*tInt - days(4) + 4*minutes(3)
```

`'yyyy-MM-dd'`	`2014-04-19`
`'dd/MM/yyyy'`	`19/04/2014`
`'dd.MM.yyyy'`	`19.04.2014`
`'MMMM d, yyyy'`	`April 19, 2014`
and other...	

Class timer

- if it is desired to cyclically repeat an action, it is possible to use class `timer`
  - better possibilities compared to infinite loop
- great advantage is the fact that `timer` creates its own thread
  - it is possible to keep on working with Matlab on launching, or alternatively launch another `timer`
- example: time display + data in 1 sec interval:

```
>> tm = timer; tic; % create an instance of timer
>> tm.ExecutionMode = 'fixedRate';
>> tm.TimerFcn = 'disp(datetime); toc;';
>> start(tm); % start the timer
```



```
Elapsed time is 0.005992 seconds.
  28-Sep-2015 08:54:18

Elapsed time is 1.007364 seconds.
  28-Sep-2015 08:54:19

Elapsed time is 2.006762 seconds.
  28-Sep-2015 08:54:20

Elapsed time is 3.006012 seconds.
  28-Sep-2015 08:54:21

Elapsed time is 4.006452 seconds.
  28-Sep-2015 08:54:22

Elapsed time is 5.007007 seconds.
  28-Sep-2015 08:54:23

Elapsed time is 6.006462 seconds.
  28-Sep-2015 08:54:24

Elapsed time is 7.006668 seconds.
  28-Sep-2015 08:54:25
```

  - it is possible to keep on Working with Matlab even as `timer` is still running
  - it is not possible to terminate the thread using CTRL+C, use:

```
>> stop(tm); % stop the timer
```

- for more information see `>> doc timer`

Class timer – Example

```
myLine = line([0 0], [0 0]); view(45, 45); box on;
xlim([-1 1]); ylim([-1 1]); zlim([-1 1]);

thisTimer = timer;                      % create timer
thisTimer.StartDelay      = 1;     % wait 1 second
thisTimer.Period          = 0.1; % repeat action after 0.1s
thisTimer.ExecutionMode   = 'fixedSpacing'; % spacing
thisTimer.UserData        = 0;     % data which we need...
thisTimer.TimerFcn        = {@timer_update, myLine, pi/16};
start(thisTimer);                       % start the timer...

fghndl = gcf;                  % stop timer if the figure is closed
fghndl.CloseRequestFcn = 'stop(thisTimer); closereq;';
```

```
function timer_update(myTimer, ~, myLine, dPhi)

myLine.XData = [1 -1]*sin(myTimer.UserData);
myLine.YData = [1 -1]*cos(myTimer.UserData);
drawnow('update');          % update graphics

myTimer.UserData = myTimer.UserData + dPhi;
```

Layout of your own instance of `timer`

420 s ↑

- Create a timer that displays, with 0.5 sec interval, "*XX* | *Hello world.*", where *XX* is the order of the message being displayed. Timer will be terminated after reaching 15 displays.

Summary of `is*` functions

- asterisk stands for whole range of functions
  - return value is logical (`true` / `false`)
- selection of the interesting ones (some even have multiple parameters)

Function	Description
`ischar`	determine whether item is character array
`isempty`	determine whether array is empty
`isfinite`	determine whether elements are of finite size
`isnan`	determine whether elements are `NaN`
`isletter`	determine whether elements are alphabetical letters (a-z, A-Z)
`islogical`	determine whether input is logical array
`isnumeric`	determine whether elements are numeric values (real, complex scalars, matrices, vectors,integers)
`isreal`	determine whether input is real array
`isstudent`	determine whether Matlabu version is Student Version?
and others	see >> `doc is*`

Function `is*`

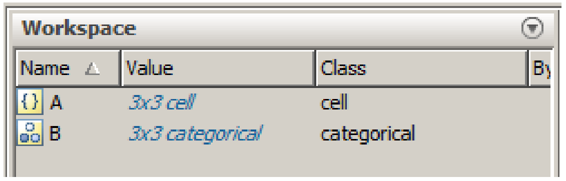
420 s ↑

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

Data type, `categorical` arrays

- array of qualitative data with values from finite set of discrete non-numerical data
  - array of non-numerical values corresponding to a category (e.g. to the category 'mean of transport' correspond following values: scooter, wheelbarrow ...)
  - values can be specified by name (e.g. values `'r'`, `'g'`, `'b'`, they can be an attribute for name `'red'`, `'green'`, `'blue'`)
  - `categorical` arrays has its own icon in Workspace


Workspace

Name	Value	Class	By
A	*3x3 cell*	cell	
B	*3x3 categorical*	categorical	

Creation of `categorical` arrays

- creation of `categorical` array from an arbitrary array of values (e.g. `cell` array of strings)

```
>> A = {'r' 'b' 'g'; 'g' 'r' 'b'; 'b' 'r' 'g'} % cell array of strings
>> B = categorical(A)    % categorical arrays
>> categories(B)         % listing of individual categories
```

- wide range of tools for combining, adding, removing, renaming, arranging,…

```
>> doc categorical arrays
```

Advantages of `categorical` arrays

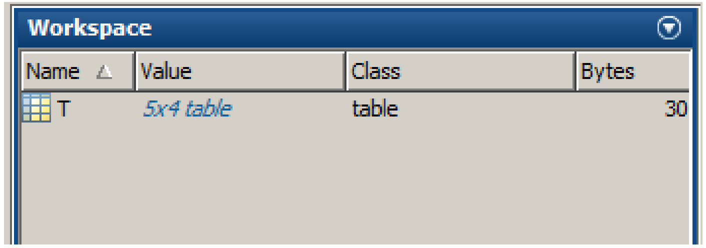
- more natural arranging of data by names
  - note: as in numerical arrays, logical operator `eq` (==) is used to compare strings in `categorical` arrays instead of function `strcmp()` used with strings
- mathematical arranging of strings
  - setting „size“ of string in other than alphabetical manner (e.g. `small < medium < large`)

```
>> allSizes = {'medium','large','small','small','medium',...
        'large','medium','small'};
>> valueset = {'small','medium','large'};
>> sizeOrd = categorical(allSizes, valueset, 'Ordinal', true);
>> comparison = sizeOrd > fliplr(sizeOrd)
```

- memory is used efficiently to store data
  - data in memory is not stored as string
  - only categories are stored as string in memory

Data type `table`

- array in form of a table that enables to have columns of various data types and sizes (silimar to `cell` array)
  - each column has to have the same number of lines (same as matrix)
  - tables have its own icon in Workspace



Name	Value	Class	Bytes
T	5x4 table	table	30

Creation of `table`

- created by inserting individual vectors as columns of the table (same length of all vectors has to be observed)

```
>> name = {'Miloslav'; 'Filip'; 'Viktor'; 'Pavel'};
>> matlabSemester = [3; 3; 2; 1];
>> favoriteDrink = categorical({'b'; 'm'; 'w'; 'w'}, ...
        {'w'; 'm'; 'b'}, ...
        {'water'; 'milk'; 'beer'});

>> T = table(matlabSemester, favoriteDrink, 'RowNames', name)
```

- more >> `doc tables array`

```
T =

                   matlabSemester    favoriteDrink
                   ______________    _____________

    Miloslav       3                 beer
    Filip          3                 milk
    Viktor         2                 water
    Pavel          1                 water
```

Advantages of `table`

- advantageous way of storing data of various data types
- access to data via numerical and name indexing
  - e.g. listing all „Smiths“ in the table and display their „age“
- possibility to store metadata in table’s properties
  - e.g. for column „age“ it is possible to set unit to „year“

Discussed functions

`tic, toc, clock, date, etime, cputime, now`	time functions, measurement of code speed	
`datevec, weekday, eomday, calendar`	time functions (days in week, month, callendat)	
`warning, error, try-catch`	warning, error message, error catching	●
`throw, rethrow`	exception issue	●
`cell, celldisp, cellplot`	variable `cell` (allocation, display)	
`setfield, fieldnames, getfield, rmfield`	structure-related functions	
`isfield, isstruct`	input is array field?, input is `struct`?	
`uiimport`	Matlab import Wizard	●
`xlsread, xlswrite`	read/write Excel spreadsheet	●
`fopen, feof, fclose, fgetl`	file open, test for end-of-file, file close, read line from file	●

Thank you!

ver. 8.1 (12/11/2017)
Miloslav Čapek, Pavel Valtr
miloslav.capek@fel.cvut.cz