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, \textit{struct}

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

\begin{verbatim}
>> 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
\end{verbatim}

- or:

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

- list of all fields of structure – fieldnames

```matlab
>> fieldnames(stock)
ans =
    'id'
    'thing'
    'price'
    'units'
    'test'
```

- value of given field

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

- does given field exist?

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

- is given variable a structure?

```matlab
>> isstruct(stock)  % = 1
```
Functions for work with structures

- delete field
  ```matlab
  >> rmfield(stock, 'id')
  ```

- more complex indexing of structures
  - structure may have more levels
    ```matlab
    >> stock(1).subsection(1).order = 1
    >> stock(1).subsection(2).order = 2
    ```
  - it is possible to combine cells with structures
    ```matlab
    >> stock(1).subsection(3).check = [1; 2]
    >> K{1} = stock;
    ```
  - certain fields can be indexed using name stored as a string
    ```matlab
    >> 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

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

- utilizing comma-separated list

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

- creating multidimensional structure

```matlab
>> 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)` with 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 function `xlsread` to import
- alternatively, use aforementioned function `uiimport`

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

- File name (has to be visible to Matlab)
- name of the file’s sheet
- range of cells
Import from Excel

- 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
    ```matlab
    >> 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
    ```matlab
    >> fx = 1:10;
    >> xlswrite('file2.xlsx', fx, 'NewSheet', 'B1');
    ```
Export to Excel

- evaluate function \( f(x) = \cos(x) + \frac{\cosh(x)}{10} \) on the interval \( x \in (-\pi, \pi) \) 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)

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

% allocation
while ~feof(fid)
    % reading
end

>> fclose(fid);
```

Program flow

```
 mesh_ESA_MM1.mphtxt - Poznámkový blok

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

% Major & minor version
0 1
1 # number of tags
 Tags
  mesh
  1 # number of types
   Types
   3 obj
   # ----------- object 0 -----------
0 0 1
1 Mesh # class
0 # id
82 # number of mesh points
0 # lowest mesh point index

% Mesh point coordinates
-31.213568250947778 -58.672917398749505
-29.02695208054629 -59.9417571901869
-29.648361958127276 -60.771791827991563
-30.687436020021966 -57.676649325079674
-32.622495192552108 -56.770649038273786
-37.2029 -62.07699999999999
-2.938200000000002 -92.717700000000007
-32.163713151591401 -55.289174591460287
-35.896359897082158 -54.176695485383718
-35.391404348413227 -51.919992214041711
-26.011720999939698 -64.703820953438754
-35.45385114852234 -52.796711381085712
-34.990153324174633 -11.8007140014333
-32.445600304718138 -63.632885437212269
-33.953504271920666 -66.4956849682652143
-34.560243960778907 -50.2132272794271711
-35.9256858591709 -49.3544145110924171
-21.4013124186013 -67.181211675277059
-21.792508542839066 -68.1301389417813
```
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);

>> 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

```matlab
>> 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\n', size(Data,1));
for k = 1:size(Data, 1)
    fprintf(fid, '%3.8f %3.8f\r\n', Data(k, :));
end
fclose(fid);
Warning message in Matlab – warning

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

```matlab
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

```matlab
f = warndlg('This is a notice...', ...
            'Trial warning', 'modal');
```
Error message in Matlab – error

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

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

- terminates program execution
- identifier can be attached

```matlab
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

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

```matlab
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

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

```matlab
>> 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;
filt(x);
toc(t0a)
```

```
t0b = clock;
filt(x);
etime(clock, t0b)
e = cputime - t0c
```
Time functions in Matlab – specialties

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

- 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
  - caution, last day of month is Saturday again!
Time functions in Matlab

- 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

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

- 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'
```

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

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

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

- various representations of the same date or time:

<table>
<thead>
<tr>
<th>Format</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>'yyyy-MM-dd'</td>
<td>2014-04-19</td>
</tr>
<tr>
<td>'dd/MM/yyyy'</td>
<td>19/04/2014</td>
</tr>
<tr>
<td>'dd.MM.yyyy'</td>
<td>19.04.2014</td>
</tr>
<tr>
<td>'MMMM d, yyyy'</td>
<td>April 19, 2014</td>
</tr>
<tr>
<td>and other...</td>
<td></td>
</tr>
</tbody>
</table>
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:

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

- 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:

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

- for more information see `>> doc timer`
Class timer – Example

```matlab
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

- 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)

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

- try following examples
  - consider in what situation they could prove useful...

```matlab
>> 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](image)
Creation of categorical arrays

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

```matlab
>> 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,…

```matlab
>> doc categorical arrays
```

Data types
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`)

```matlab
>> 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 (similar to `cell` array)
  - each column has to have the same number of lines (same as matrix)
- tables have its own icon in Workspace
Creation of table

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

```matlab
>> 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, calendat) |
| 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? |
| uiiimport                                 | 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!

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

ver. 8.1 (12/11/2017)

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.