

AE0B17MTB – Matlab

# Part #11



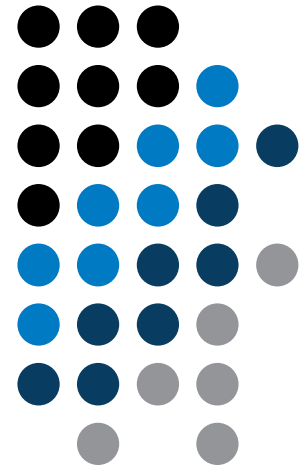
Miloslav Čapek

`miloslav.capek@fel.cvut.cz`

Filip Kozák, Viktor Adler, Pavel Valtr

Department of Electromagnetic Field

B2-626, Prague



# Learning how to ...

---

**Data types struct , categorical, table**

**Import / export in Matlab**

**Time functions**

**warning, error, try-catch**

**Basics of symbolic math**

$$I = \iint_S f(x, y) dS \quad f(x, y) = x + y$$
$$x \in (0, 2),$$
$$y \geq 0 \wedge y \leq 2 - x$$

# 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
```

# Functions for work with structures

- new field creation

- direct command

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

- field name as a string

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

- 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

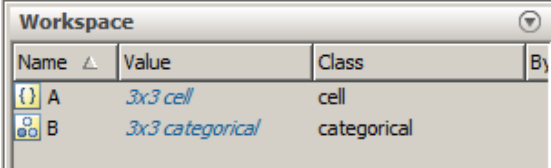
```
>> K{1}(1).subsection(3).('check')(2)
```

# Typical application of structure

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

# 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



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,...

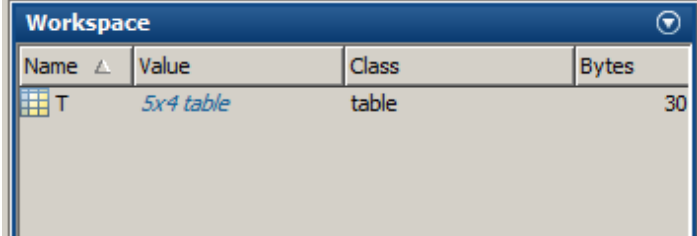
```
>> doc categorical arrays
```

# 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`)
- 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 tables

- 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



The screenshot shows the MATLAB Workspace window with a table variable 'T'. The table has 5 rows and 4 columns. The columns are labeled 'Name', 'Value', 'Class', and 'Bytes'. The row for 'T' shows a table icon, the name 'T', the value '5x4 table', the class 'table', and the size '30' bytes.

Name	Value	Class	Bytes
T	5x4 table	table	30

# Creation of tables

- 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 =
```

	semester	favoriteDrink
Miloslav	3	beer
Filip	3	milk
Viktor	2	water
Pavel	1	water

# Advantages of tables

- 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“

# Data Import and export

- Matlab supports wide range of file formats
  - mat, txt, xls, jpeg, bmp, png, wav, avi and others, see
    - Matlab → Data and File Management → Data Import and Export → Import and Export Basics
  - 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`, `VideoWriter`



# Import from Excel

- use function `xlsread` to import
  - alternatively, use aforementioned function `uiimport`

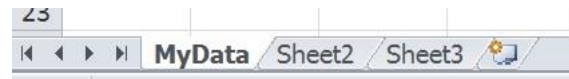
```
>> num = xlsread('MTB_Week11.xls', 'MyData', 'A1:B4');
```

File name  
(has to be visible to Matlab)



MTB\_Week11.xlsx - Microsoft Excel

name of the file's sheet



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  $f(x) = \cos(x) + \frac{\cosh(x)}{10}$  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.mph.txt');
```

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

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

```
mesh_ESA_MM1.mph.txt - 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.mph.txt');
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.mph.txt - 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

```

```
>> size(Data)
```

```
ans =
```

```
582 2
```

# Writing to a file #1

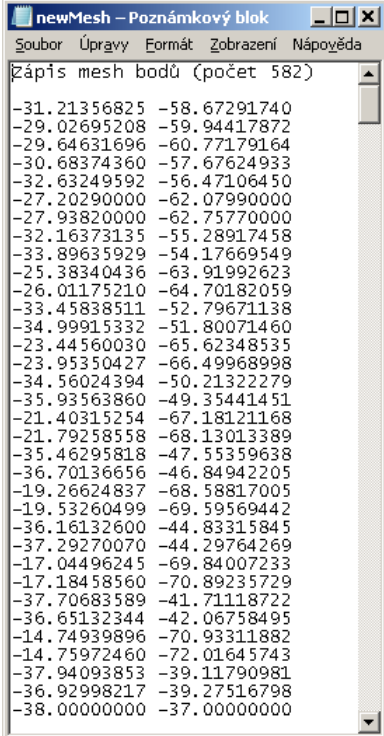
- we try to write variable `Data` from a file `data.mat` where the first line contains a header

```
>> 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



```
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 `warndiag` with GUI
  - but it is just a statement really, see last lecture

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





# 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 `warndiag` with GUI
  - but it is just a statement really, see last lecture

```
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

```
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;
    % rethrow;
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
<b>tic - toc</b>	measure length of time interval between expressions <code>tic</code> and <code>toc</code>
<code>clock</code>	return six element vector [year month day hour minute seconds]
<code>date</code>	return date in format dd-mmm-yyyy, variable is of type char (text)
<code>etime</code>	return time interval between <code>t1</code> and <code>t2</code> , <code>etime(t2,t1)</code>
<code>cputime</code>	
<code>now</code>	return current date and time as an integer
<b>timeit</b>	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  $\geq P4@$ 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

- `datevec`, `datenum`, `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

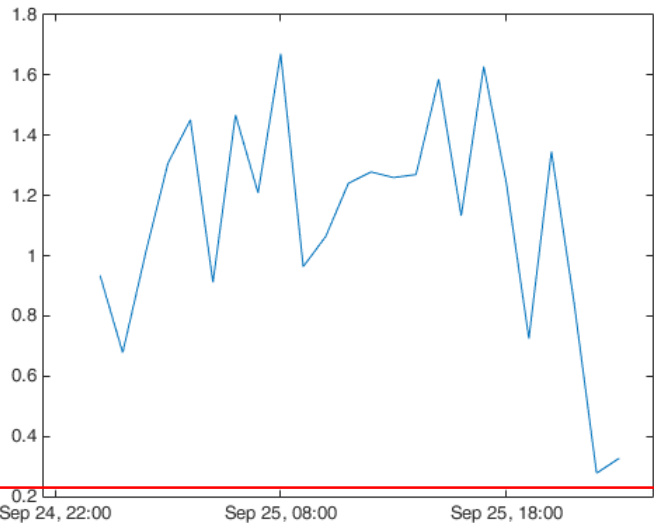
420 s ↑

- Try to implement selected time functions into your project

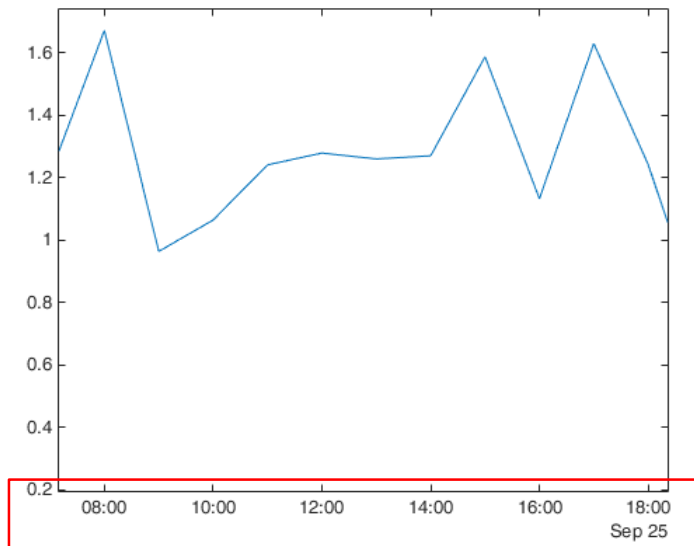
# Time series data

- 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
```

- 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
```

```
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
```

- 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.

# Higher math

- two different attitudes are distinguished
  - symbolic math
  - numeric math
    - numerical errors
  - possible classification: analytical result in principle enables to get result in infinite number of decimals
- there exist wide range of techniques in Matlab (symbolical as well as numerical)
  - only selected techniques will be covered

# Handle functions – revision

- enables indirect function invoking
- reference to the function is stored in handle

```
handle1 = @function_name  
handle2 = @(args) function_name
```

- it is quite powerful tool though a bit more complicated
  - enables to invoke a function from locations where it is not visible to Matlab
  - function handle is a data type in Matlab (see `whos`)

```
>> clear, clc;  
>> doc function_handle  
  
>> fxy = @(x, y) x^2 + y^2 - 5  
>> fxy(2, -2)  
  
>> fcos = @(alpha) cos(alpha)  
>> fcos(pi)
```

# Polynomials #1

- representation of polynomials in Matlab

$$P = C_n x^n + C_{n-1} x^{n-1} + \dots + C_1 x + C_0 = [C_n \quad C_{n-1} \quad \dots \quad C_1 \quad C_0]$$

```
>> x = roots([1 0 -1]);
>> x1 = x(1)
>> x2 = x(2)
```

- function `roots` finds roots of a polynomial
- polynomial evaluation: `polyval`

```
>> x = 2
>> p1 = 3*x^5 - 7*x^3 + 1/2*x^2 - 5
>> polyval([3 0 -7 1/2 0 -5], 2)
```

- polynomial multiplication: `conv`

$$A_1 = x - 1$$

$$A_2 = x + 1$$

$$A_1 \cdot A_2 = (x-1) \cdot (x+1) = x^2 - 1$$

```
>> A1 = [1 -1]
>> A2 = [1 1]
>> conv(A1, A2)
% = [1 0 -1]
```

# Polynomials #2

- polynomial division: `deconv`

```
>> deconv([1 0 -1], [1 1]) % = [1 -1]
```

$$\frac{x^2 - 1}{x + 1} = \frac{(x - 1) \cdot (x + 1)}{x + 1} = x - 1$$

- other polynomial related functions (selection of some):
  - `residue`: residue of ratio of two polynomials
  - `polyfit`: approximation of data with polynomial of order  $n$
  - `polyint`: polynomial integration
  - `polyder`: polynomial derivative

```
>> S = [1 1];
>> T = polyint(S) % = [0.5 1 0]
>> U = polyder(T) % = S = [1 1]
>> polyder(U) % = 1
```

$$\int (x + 1) dx = \frac{1}{2} x^2 + x$$

$$\frac{d\left(\frac{1}{2} x^2 + x\right)}{dx} = x + 1$$



# Polynomials #3

- polynomial multiplication

$$P1 = A + Bx$$

$$P2 = 4x^2 + 2x - 4$$

```
>> syms A B x
>> P1 = A + B*x;           % entering 1. polynomial
>> P2 = 4*x^2 + 2*x - 4;  % 2. polynomial
>> P0 = P1*P2;           % multiplication
>> P = expand(P0)         % expansion
```

- note: function `expand` requires Symbolic Math Toolbox

# $x = ? : f(x) == g(x)$

- two functions are given, we want to analytically find out points where these functions are equal to each other

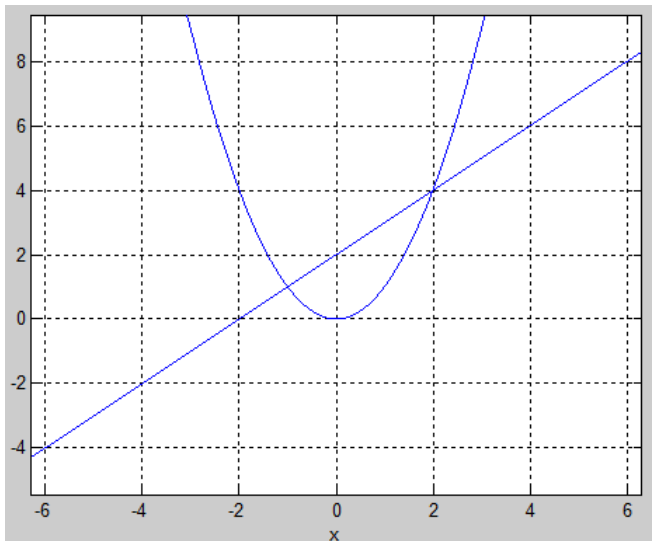
$$f(x) = x^2$$

$$g(x) = x + 2$$

$$x = ? : \{f(x) = g(x)\}$$

enter

```
>> clear, clc;
>> syms x;
>> f = x^2;
>> g = x + 2;
```



solve

```
>> x0 = solve(f - g) % = 2; -1
```

check

```
>> ezplot(f);
>> hold on;
>> grid on;
>> ezplot(g);
```

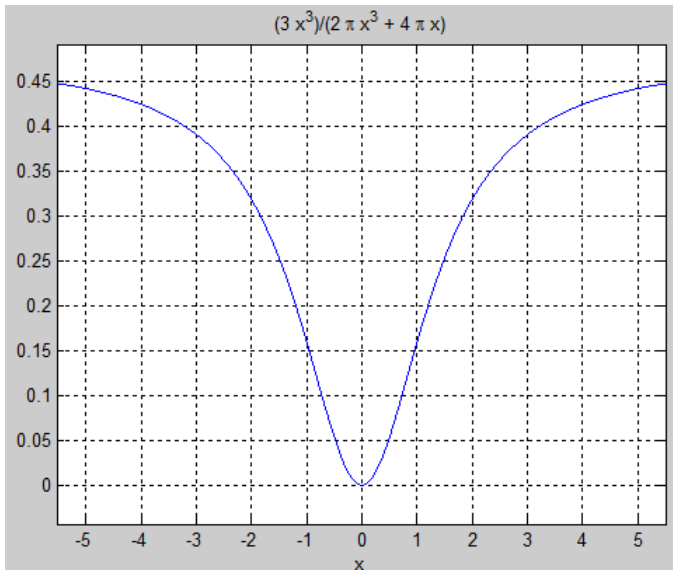
# Function limit

- find out function limit

$$f(x) = \frac{3x^3}{2\pi x^3 + 4\pi x} \quad f(x) = \frac{3}{2\pi} \left( \frac{x^2}{x^2 + 2} \right) \quad \lim_{x \rightarrow -\infty} f(x) = \lim_{x \rightarrow \infty} f(x) \stackrel{L'H.P.}{=} \frac{3}{2\pi} = 0.4775$$

enter

```
>> clear, clc, close all;
>> syms x real;
>> f = 3*x^3/(2*pi*x^3 + 4*pi*x)
```



solve

```
>> lim1 = limit(f, x, -inf)
>> lim2 = limit(f, x, inf)

>> double(lim1) % = 0.4775
>> double(lim2) % = 0.4775
```

check

```
>> figure;
>> ezplot(f);
>> grid on;
```

# Function derivative #1

- apply L'Hospital's rule to previous function
  - function  $f(x)$  contains 3<sup>rd</sup> power of  $x$ ; carry out 3<sup>rd</sup> derivative (of numerator and denominator separately) in  $x$

$$f(x) = \frac{3x^3}{2\pi x^3 + 4\pi x}$$

$$f_1(x) = 3x^3$$

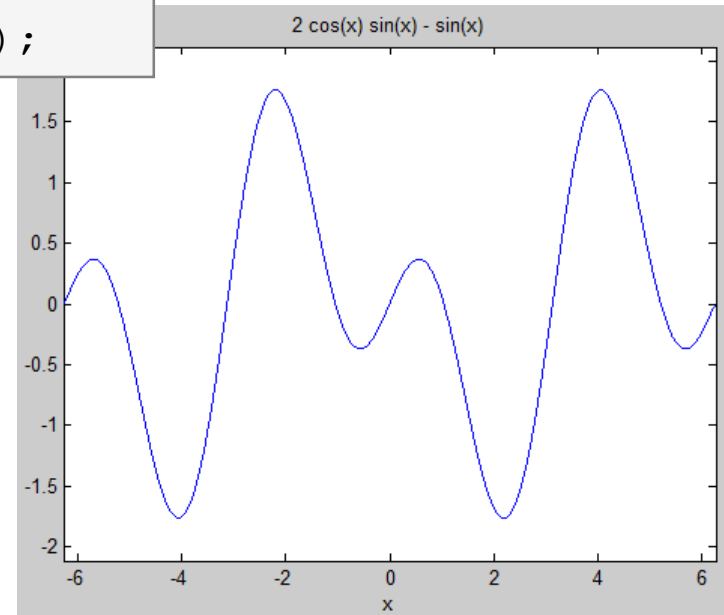
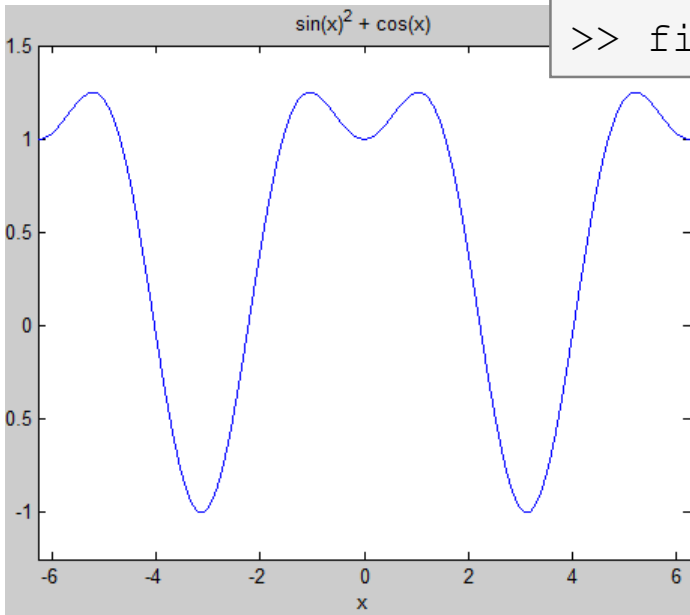
$$f_2(x) = 2\pi x^3 + 4\pi x$$

```
>> f1 = 3*x^3;  
>> f2 = 2*pi*x^3 + 4*pi*x;  
>> A1 = diff(f1,3)  
>> A2 = diff(f2,3)  
>> double(A1/A2) % = 0.4775
```

# Function derivative #2

- carry out derivative of the following function in  $x$   $f(x) = \sin^2(x) + \cos(x)$ 
  - compare results and plot them

```
>> clear, clc;  
>> syms x;  
>> f = sin(x)^2 + cos(x);  
>> figure; ezplot(f);  
>> fd = diff(f);  
>> figure; ezplot(fd);
```



# Integration #1

- let's first symbolically carry out derivative of function  $f(x) = \sin(x) + 2$
- save the second derivative of  $f$  and call it  $g$ , compare results
- now integrate function  $g$  ( $1\times, 2\times$ ), do we get the original function  $f$ ?
  - ignore integration constants

```
>> clear, clc;
>> x = sym('x');

>> f = sin(x) + 2
>> figure; ezplot(f);

>> fd = diff(f)
>> figure; ezplot(fd);

>> fdd = diff(f, 2)
>> figure; ezplot(fdd);
```

```
>> g = fdd;
>> gi = int(g)
>> figure; ezplot(gi);

>> gii = int(gi);
>> err = f - gii

figure;
subplot(1, 2, 1);
ezplot(f);
subplot(1, 2, 2);
ezplot(gii);
```

# Integration #2

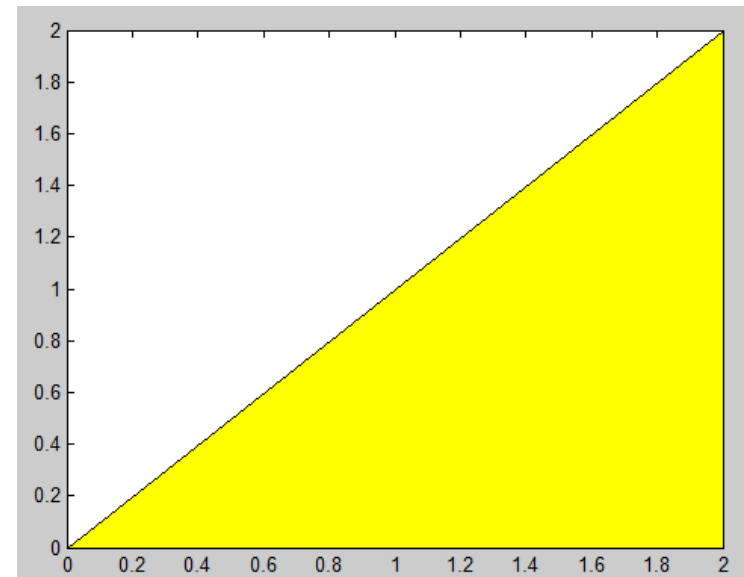
- integral of a function  $f(x) = x$ 
  - calculate following integral
  - do the calculation manually, plot the function
  - calculate indefinite integral in Matlab
  - calculate definite integral on interval (0, 2), use e.g. function `int`

$$I = \int_0^2 f(x) dx$$

$$I = \int_0^2 f(x) dx = \int_0^2 x dx = \left[ \frac{x^2}{2} \right]_0^2 = \frac{4}{2} - 0 = 2$$

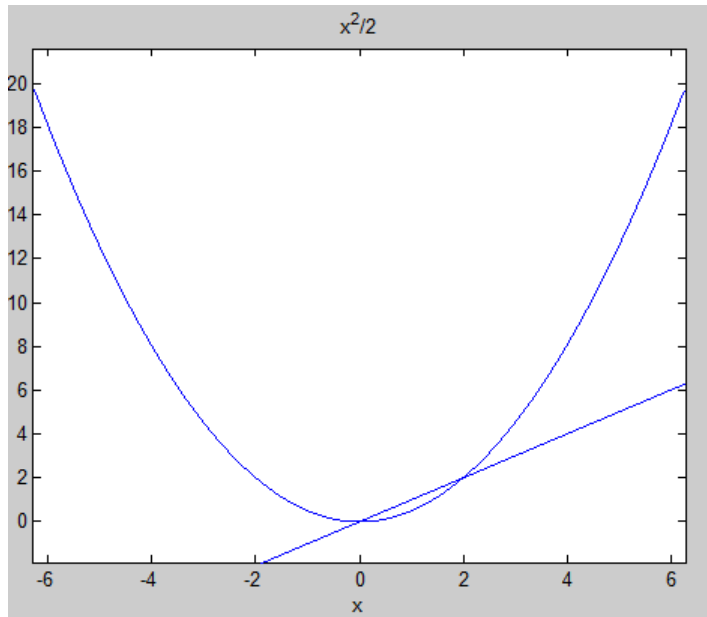
$$I = \frac{2 \cdot 2}{2} = 2$$

```
>> fill([0 2 2], [0 0 2], 'y')
```



# Integration #3

- integral of a function



```

>> clear, clc;
>> syms x;
>> f = x;
>> g = int(x);

>> figure;
>> ezplot(f);
>> hold on;
>> ezplot(g);

>> int(f, x, 0, 2)           % = 2
>> polyarea([0 2 2], [0 0 2]) % = 2

% BUT!:
>> f = @(x) x % function_handle!
>> I = quad(f, 0, 2)         % = 2

```



# Numerical integration #1

- numerical approach is used whenever the closed-form (analytical) solution is not known which happens quite often in technical sciences (almost always)
- it is possible to use various numerical integration methods, see literature
- alternatively, Matlab functions can be utilized
  - `quad`, `dblquad`, `triplequad` and others
    - `integral`, `integral2`, `integral3` functions in new versions of Matlab
  - define function to be integrated (write your own function or use *function handle*)

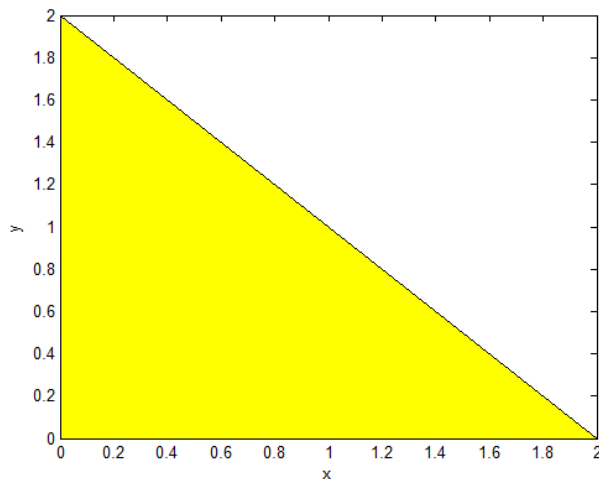
# Numerical integration #2

- solve the following integral on the interval

$$x \in (0,2),$$

$$y \geq 0 \wedge y \leq 2 - x$$

$$I = \iint_S f(x, y) dS \quad f(x, y) = x + y$$



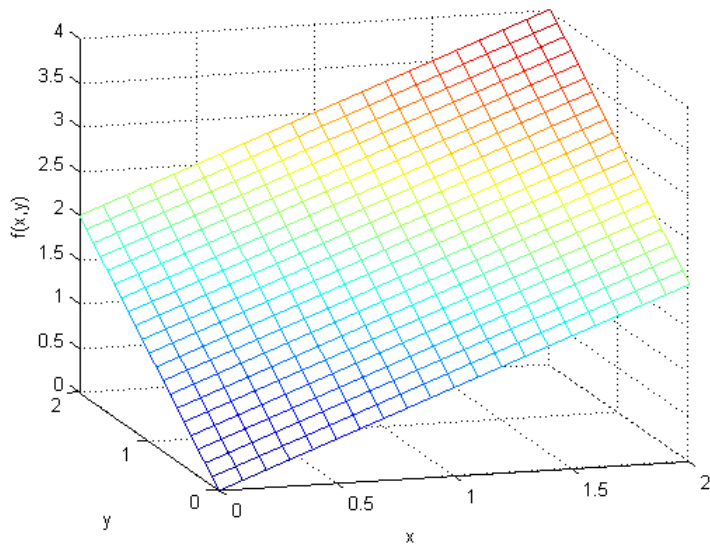
$$\begin{aligned} I &= \int_0^2 \int_0^{y_{\max}} f(x, y) dx dy = \int_0^2 \int_0^{2-x} (x + y) dx dy = \int_0^2 \left( x[y]_0^{2-x} + \left[ \frac{y^2}{2} \right]_0^{2-x} \right) dx \\ &= \int_0^2 \left( x(2-x) + \frac{(2-x)^2}{2} \right) dx = \int_0^2 \left( 2x - x^2 + 2 - 2x + \frac{x^2}{2} \right) dx \\ &= \int_0^2 \left( 2 - \frac{x^2}{2} \right) dx = 2[x]_0^2 - \frac{1}{2} \left[ \frac{x^3}{3} \right]_0^2 = 4 - 8 \cdot \frac{1}{6} = \frac{12-4}{3} = \frac{8}{3} = \underline{\underline{2.666}} \end{aligned}$$

# Numerical integration #3

- solve the following integral on the interval

$$x \in (0,2),$$
$$y \geq 0 \wedge y \leq 2 - x$$

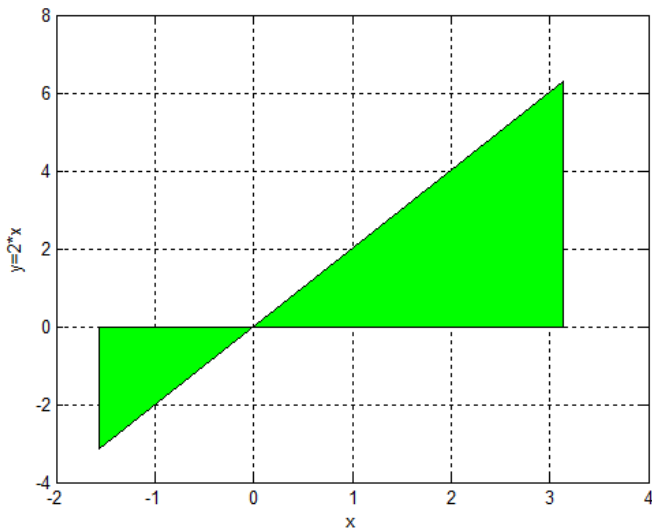
$$I = \iint_S f(x, y) dS \quad f(x, y) = x + y$$



# Numerical integration #4

- it is possible to work with external scripts as well; i.e. having „complex“ expression that we don't want to process as handle:

$$I = \int_x f(x) dx = \int_{-\frac{\pi}{2}}^{\pi} 2x dx = 2 \int_{-\frac{\pi}{2}}^{\pi} x dx = 2 \left[ \frac{x^2}{2} \right]_{-\frac{\pi}{2}}^{\pi} = \pi^2 - \frac{\pi^2}{4} = \underline{\underline{\frac{3}{4} \pi^2}}$$



# Numerical integration #1

- general problem of derivative (it is not possible to approach zero)

$$\lim_{\Delta x \rightarrow 0} \frac{f(x + \Delta x) - f(x)}{\Delta x}$$

- various sophisticated numerical methods of various complexity are used
- web pages to solve this problem in a complex way :
  - <http://www.matrixlab-examples.com/derivative.html>

# Closing notes

- in the case there is a lot of symbolic calculations or when approaching Matlab limits, try another mathematical tool (for analytical solution especially Maple, Mathematica)
- nevertheless Matlab is a perfect choice for numerical computing (although both Mathematica's symbolic and numerical kernels are excellent)

# Higher math

- polynomials
  - <http://www.matrixlab-examples.com/polynomials.html>
- single and double integration (symbolic)
  - <http://www.matrixlab-examples.com/definite-integrals.html>
- derivative (numerical)
  - analytic input:
    - <http://www.matrixlab-examples.com/derivative.html>
  - numeric input
    - manual derivative

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



# 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)
```

# Discussed functions

---

<code>tic, toc, clock, date, etime, cputime, now</code>	time functions, measurement of code speed	
<code>datevec, weekday, eomday, calendar</code>	time functions (days in week, month, calendar)	
<code>warning, error, try-catch</code>	warning, error message, error catching	•
<code>throw, rethrow</code>	exception issue	•
<code>cell, celldisp, cellplot</code>	variable <code>cell</code> (allocation, display)	
<code>setfield, fieldnames, getfield, rmfield</code>	structure-related functions	
<code>isfield, isstruct</code>	input is array field?, input is struct?	
<code>uiimport</code>	Matlab import Wizard	•
<code>xlsread, xlswrite</code>	read/write Excel spreadsheet	•
<code>fopen, feof, fclose, fgetl</code>	file open, test for end-of-file, file close, read line from file	•
<code>sym, syms</code>	create symbolic variable(s)	
<code>roots, polyval, conv, deconv</code>	polynomial-related functions 1	
<code>residue, polyfit, polyder, polyint, expand</code>	polynomial-related functions 2	
<code>solve</code>	equations and systems solver	•
<code>limit, diff, int</code>	function limit, derivative, function integration	
<code>ezplot</code>	symbolic function plotter	
<code>quad (integral), quad2d (integral2)</code>	numeric integration	•

---

# Thank you!



ver. 4.2 (05/01/2016)

Miloslav Čapek, Pavel Valtr  
miloslav.capek@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.

