A0B17MTB – Matlab

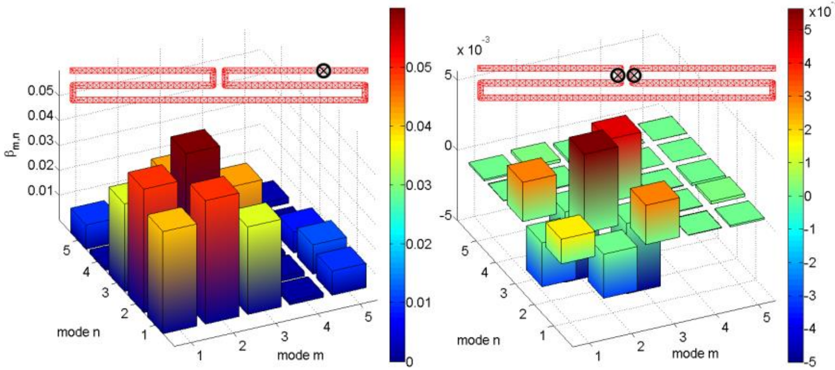
Part #5

Miloslav Čapek
miloslav.capek@fel.cvut.cz

Filip Kozák, Viktor Adler, Pavel Valtr

Department of Electromagnetic Field
B2-626, Prague

Learning how to …



Data type `cell`

Program branching #2

Visualizing in Matlab #1

Debugging #1

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

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

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
double
```

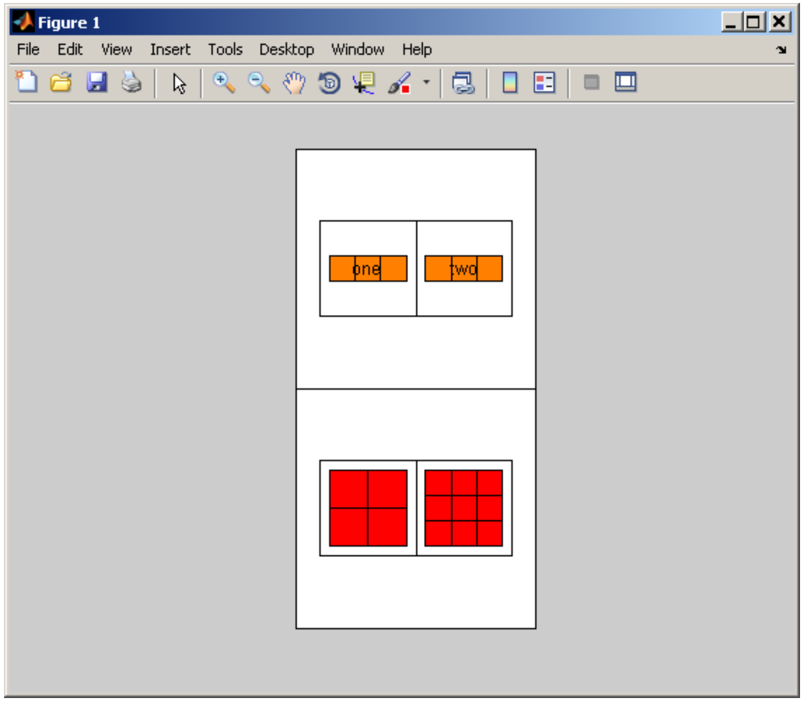

Department of Electromagnetic Field, CTU FEE, miloslav.capek@fel.cvut.cz

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
- …

Program branching – `switch` / `case`

- does a variable correspond to one of (usually many) values?
- the commands in the part `otherwise` are carried out when none of the cases above applies (compare to `else` in the `if`statement)

- suitable to evaluate conditions containing strings
  - if you want to learn more details on when to use `if` and when to use `switch`, visit pages `blogs.mathworks.com`

- it is appropriate to always terminate the statement by `otherwise` part

```
c = randi(1e2);
switch mod(c,2)
 case 1
    disp('c is odd');
 case 0 & c > 10
    disp('even, >10');
 otherwise
    disp('even, <=10');
end
```

Program branching – `switch` / `case`

450 s

- create a script that, given lengths of two sides of a right triangle, calculates the length of the third side (Pythagorean theorem)
  - two sides are known together with string marking the type of unknown side (`'leg'` for leg or `'hyp'` for hypotenuse)

```
%% HINT:
% input variables will be here
%(including type of unknown side)
switch aaa % aaa denotes the type of unknown side
  case yyy % calculation for the first type of side
%       calculation1
  case zzz % calculation for the second type of side
%       calculation2
  otherwise % unknown type
%       return empty (default) values
end
```


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

What does the script do?

300 s

- try to estimate what does the script below assign to `logResult` variable depending on input variable `vec` (a vector)
  - are you able to decide whether there is a Matlab function doing the same?

```
% vec is a given vector

logResult = false;
m = 1;
while (m <= length(vec)) && (logResult == false)
    if vec(m) ~= 0
        logResult = true;
    end
    m = m + 1;
end
```

What does the script do?

300 s

- try to estimate what does the script below assign to `logResult` variable depending on input variable `mat` (a matrix)
  - are you able to decide whether there is a Matlab function doing the same?

```
% mat is a given matrix
count  = 0;
[mRows, nColumns] = size(mat);
for m = 1:mRows
    for n = 1:nColumns
        if mat(m,n) ~= 0
            count = count + 1;
        end
    end
end
logResult = count == numel(mat);
```

Example of listing more options

- `switch` supports options listing
  - evaluation of options A1 a A2 in the same way:

```
switch my_expression
 case {'A1', 'A2'}
     % do something
otherwise
     % do something else
end
```

Inifinite loop – `for` cycle (a riddle)

- in the last lecture we learned how to construct the infinite loop with the `while` command (`>> while true, 'ok', end`)
  - Do you think, that the infinite loop can be constructed with the `for` cycle as well?
  - How?
  - Are there any restrictions? How many cycles will be performed and why?

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

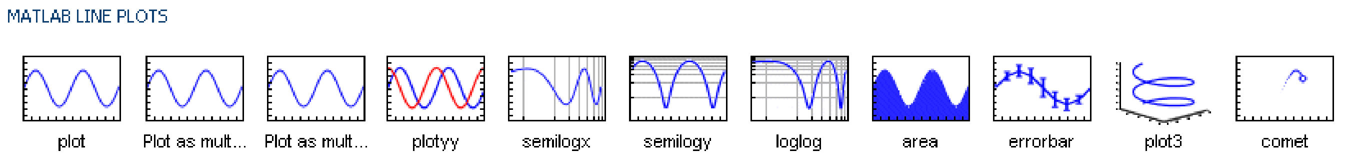
Introduction to visualizing

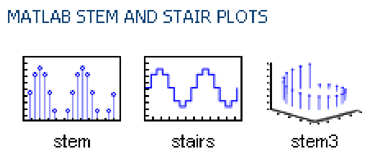
- we have already got acquainted (marginally) with some of Matlab graphs
  - `plot`, `stem`, `bar`, `hist`, `surf`

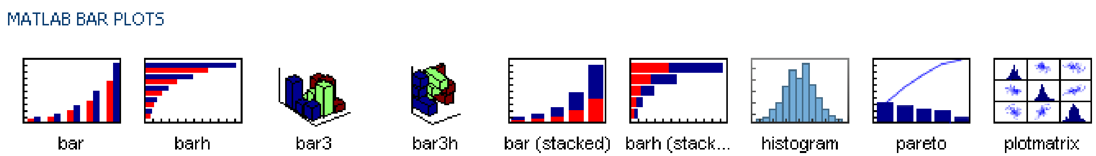
- in general, graphical functions in Matlab can be used as
  - higher level
    - access to individual functions, object properties are adjusted by input parameters of the function
    - first approx. 9-10 weeks of the semester
  - lower level
    - calling and working with objects directly
    - knowledge of Matlab handle graphics (OOP) is required
    - opens wide possibilities of visualization customization

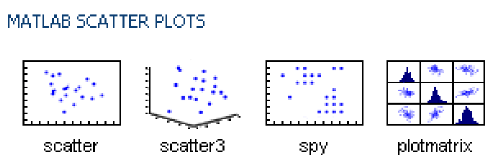
- details to be found in:
  - Matlab → Graphics → 2-D and 3-D Plots → Plotting Basics

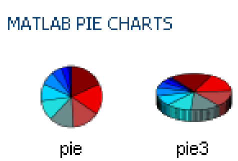
Selected graphs #1

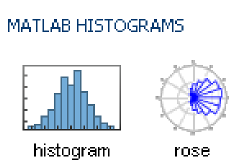




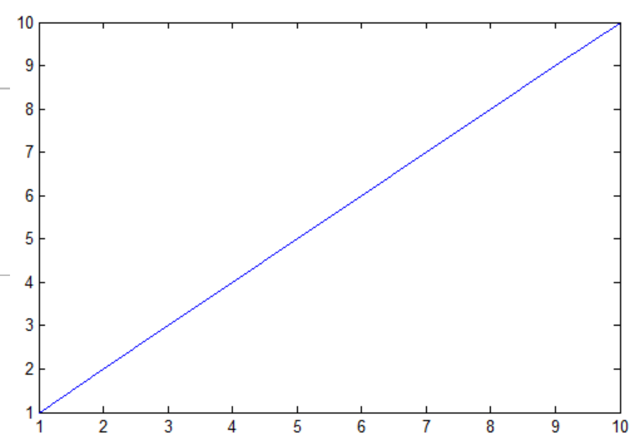


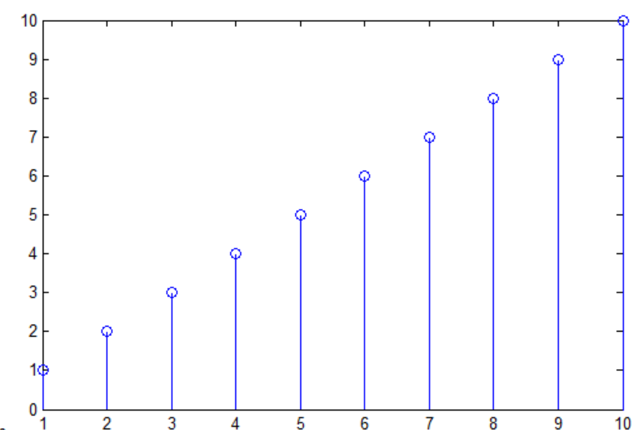




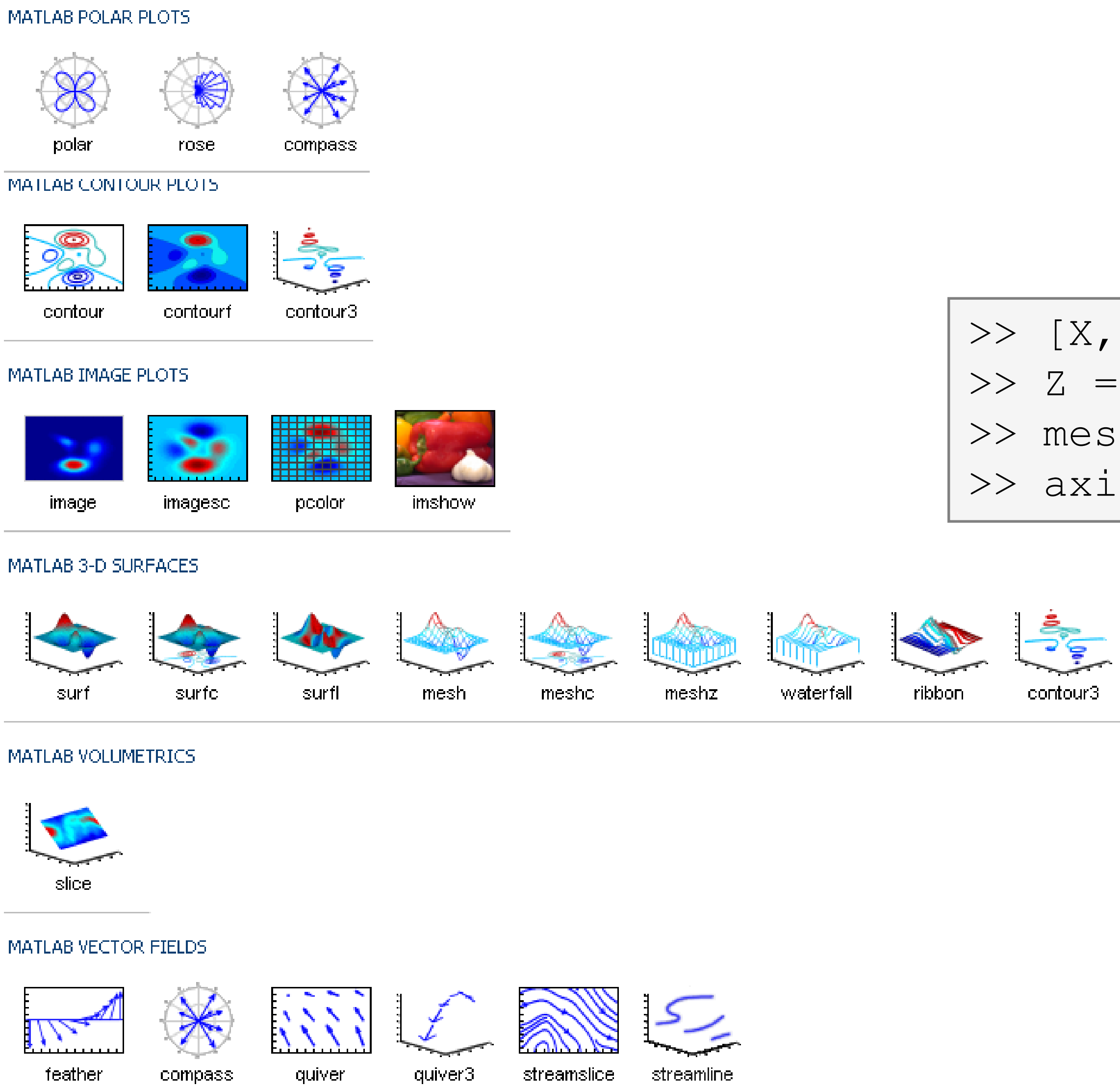



```
>> plot(linspace(1,10,10));
>> stem(linspace(1,10,10));
>> % … and others
```

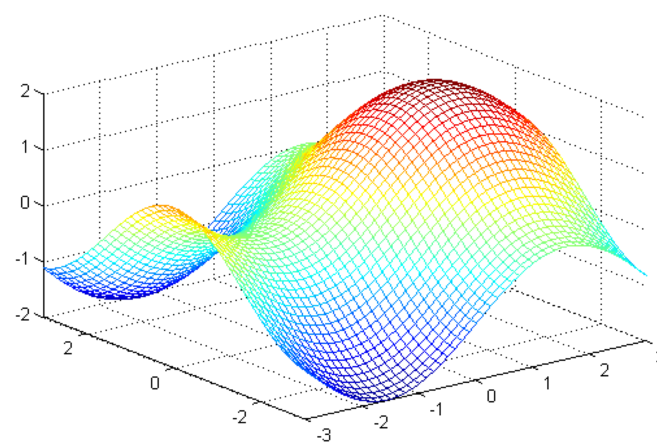




Selected graphs #2




```
>> [X,Y] = meshgrid(-3:.125:3);
>> Z = sin(X) + cos(Y);
>> mesh(X,Y,Z);
>> axis([-3 3 -3 3 -2 2]);
```



Selected functions for graph modification

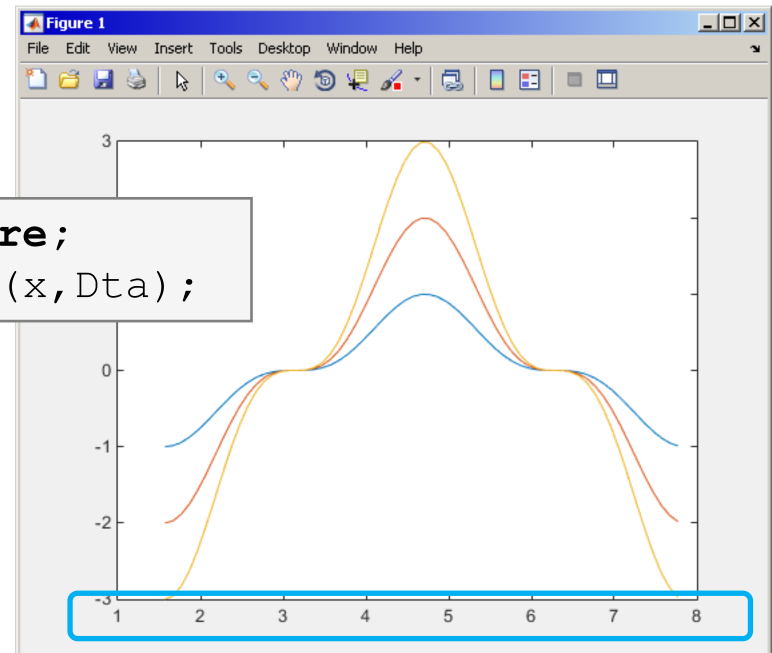
- Graphs can be customized in many ways, the basic ones are:

function	description
`title`	title of the graph
`grid on, grid off`	turns grid on / off
`xlim, ylim, zlim`	set axes' range
`xlabel, ylabel, ...`	label axes
`hold on`	enables to add another graphical elements while keeping the existing ones
`legend`	display legend
`subplot`	open more axes in one figure
`text`	adds text to graph
`gtext, ginput`	insert text using mouse, add graph point using mouse
and others	

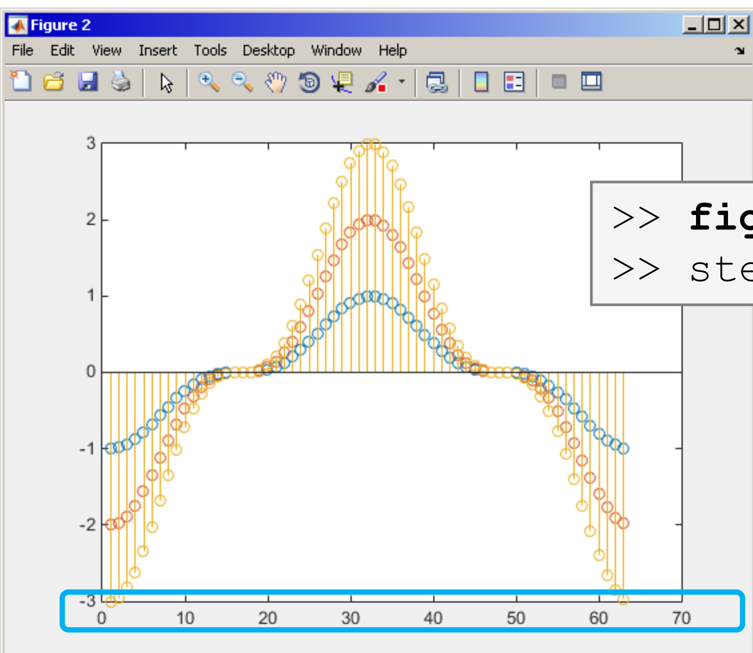
figure

- `figure` opens empty figure to plot graphs
  - the function returns object of class `Figure`

```
>> x = (0:0.1:2*pi) + pi/2;
>> Dta = -[1 2 3]'*sin(x).^3;
```




```
>> figure;
>> plot(x,Dta);
```



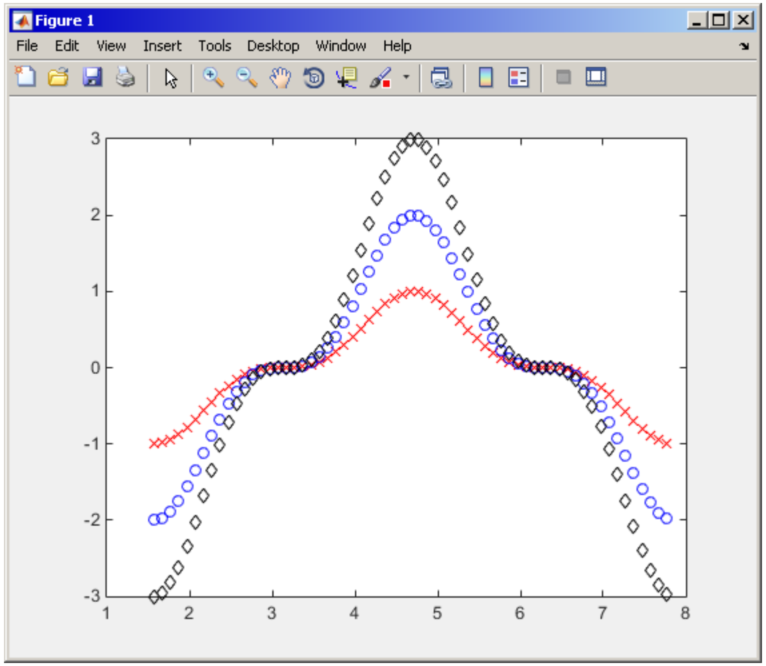

```
>> figure;
>> stem(Dta');
```

- it is possible to plot matrix data (column-wise)
- don't forget about x-axis data!

hold on

- function `hold on` enables to plot multiple curves in one axis, it is possible to disable this feature by typing `hold off`
- functions `plot`, `plot3`, `stem` and others enable to add optional input parameters (as strings)

```
x = (0:0.1:2*pi) + pi/2;
Dta = -[1 2 3]'*sin(x).^3;
figure;
plot(x, Dta(1,:), 'xr');
hold on;
plot(x, Dta(2,:), 'ob');
plot(x, Dta(3,:), 'dk');
```



LineSpec – customizing graph curves

- what do `plot` function parameters mean?
  - see `>> doc LineSpec`
  - the most frequently customized parameters of graph's lines
    - color (can be entered also using matrix [R G B], where R, G, B vary between 0 a 1)
    - marker shape (*Markers*)
    - line style
- big changes since 2014b version!

```
plot(x,f,'bo-');
plot(x,f,'g*--');
```

```
figure('color', ...
    [.5 .1 .4]);
```

line color	
'r'	red
'g'	green
'b'	blue
'c'	cyan
'm'	magenta
'y'	yellow
'k'	black
'w'	white

marker	
'+'	plus
'o'	circle
'*'	asterisk
'.'	dot
'x'	x-cross
's'	square
'd'	diamond
'^'	triangle
and others	`>> doc LineSpec`

line style	
'-'	solid
'--'	dashed
':'	dot
'-.'	dash-dot
'none'	no line

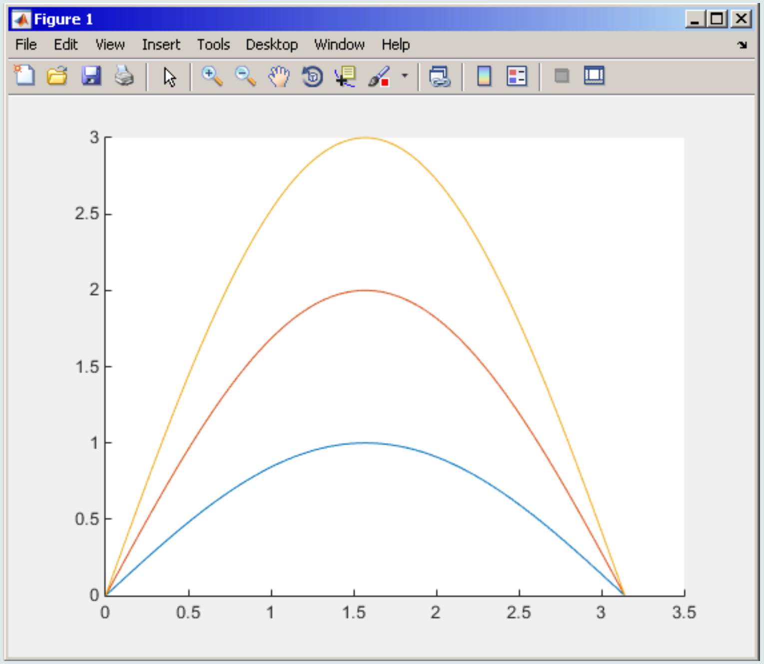
LineSpec – default setting in 2014b

- colors in given order are used when plotting more lines in one axis
  - this color scheme was changed in 2014b and later versions:
- it is not necessary to set color of each curve separately when using `hold on`
  - following default color order is used:

```
>> get(groot, 'DefaultAxesColorOrder')

% ans =
%
%          0    0.4470    0.7410
%     0.8500    0.3250    0.0980
%     0.9290    0.6940    0.1250
%     0.4940    0.1840    0.5560
%     0.4660    0.6740    0.1880
%     0.3010    0.7450    0.9330
%     0.6350    0.0780    0.1840
```

```
close all; clear; clc;
x = 0:0.01:pi;
figure;
hold on;
plot(x, 1*sin(x));
plot(x, 2*sin(x));
plot(x, 3*sin(x));
```

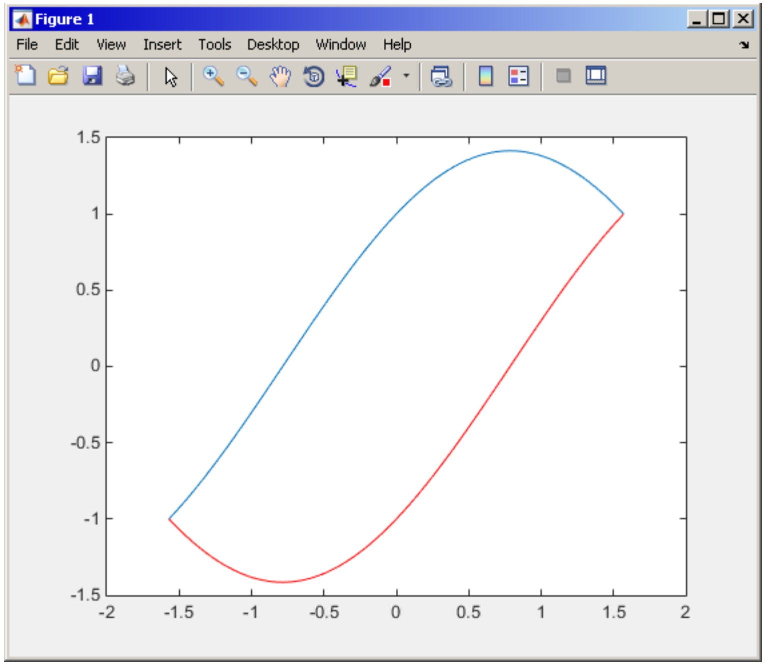


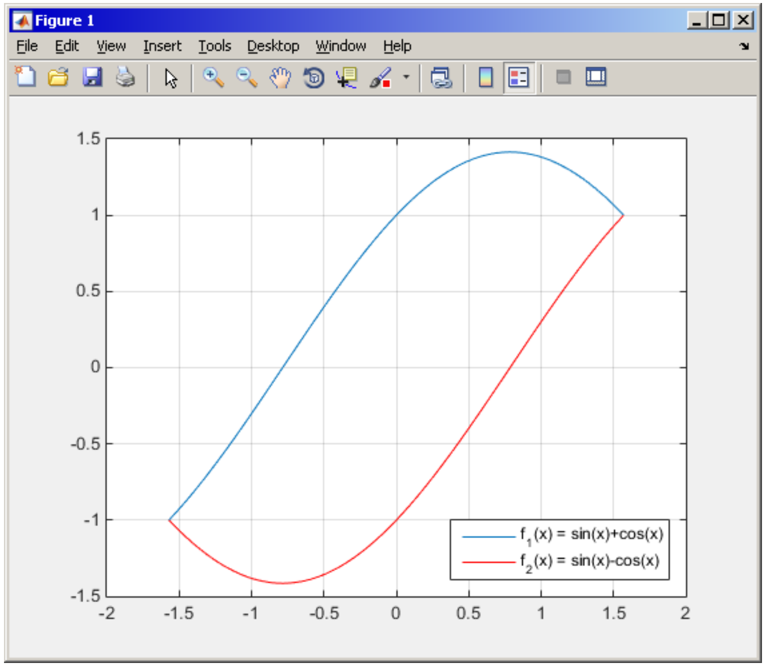
Visualizing – `legend`, `grid`

```
x = -pi/2:0.01:pi/2;
f1 = sin(x) + cos(x);
f2 = sin(x) - cos(x);
```

```
plot(x, f1);
hold on;
plot(x, f2, 'r');
```

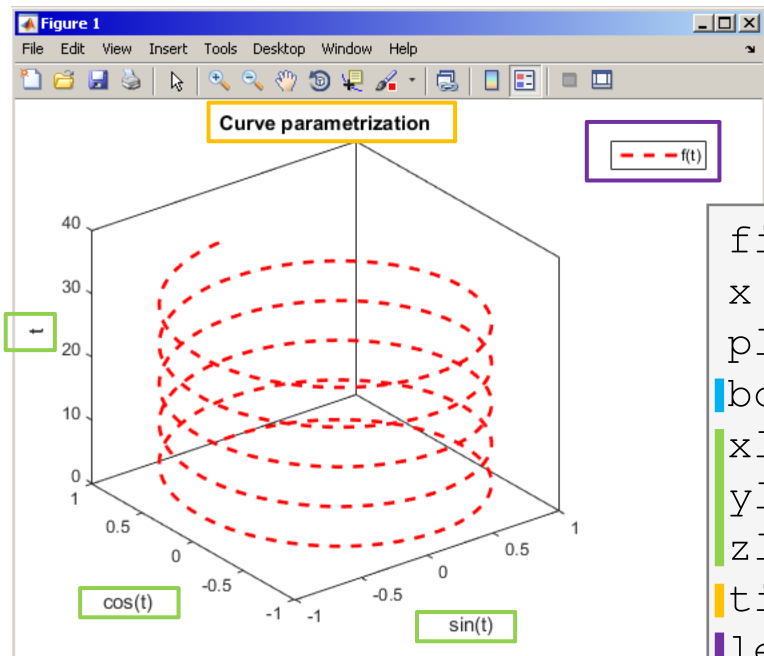
```
grid on;
legend('f_1(x) = sin(x)+cos(x)',...
    'f_2(x) = sin(x)-cos(x)',...
    'Location', 'southeast');
```





plot3

- the example below shows plotting a spiral and customizing plotting parameters
  - functions `xlabel`, `ylabel` and `zlabel` are used to label the axes
  - function `title` is used to display the heading
  - function `legend` pro characterize the curve



- function `box` sets boundary to the graph

```
figure('color','w');
x = 0:0.05:10*pi;
plot3(sin(x),cos(x),x,'r--','LineWidth',2);
box on;
xlabel('sin(t)');
ylabel('cos(t)');
zlabel('t');
title('Curve parametrization')
legend('f(t)');
```

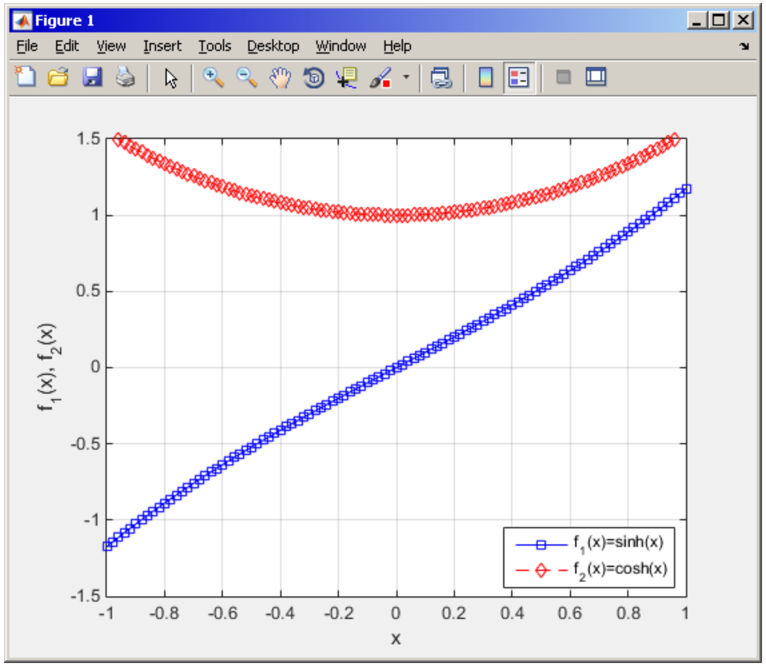
LineSpec – customizing graph curves

450 s

- evaluate following two functions in the interval [-1,1] for 101 values:

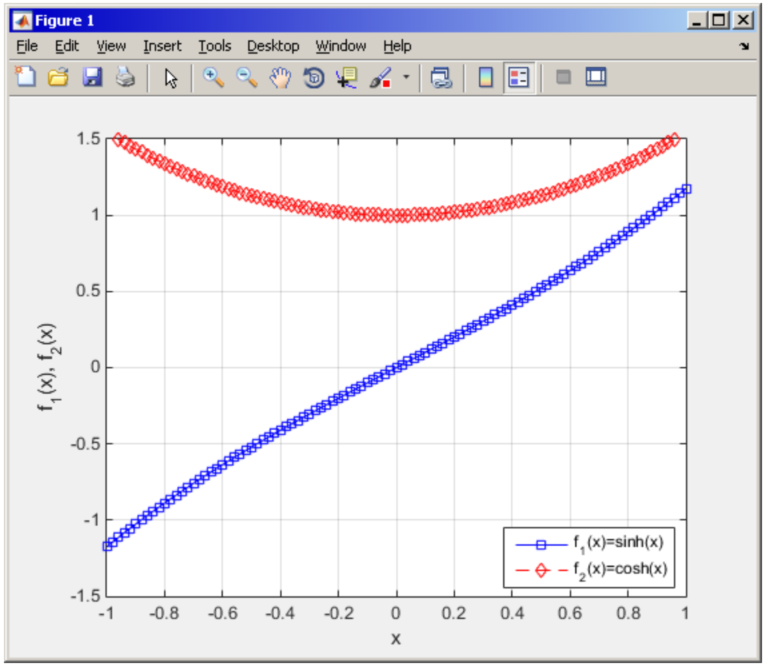
$$f_1(x) = \sinh(x), \qquad f_2(x) = \cosh(x)$$

- use the function `plot` to depict both f_1 and f_2 so that
  - both functions are plotted in the same axis
  - the first function is plotted in blue with □ marker as solid line
  - the other function is plotted in red with ◊ marker and dashed line
  - limit the interval of the *y*-axis to [-1.5, 1.5]
  - add a legend associated to both functions
  - label the axes (*x*-axis: *x*, *y*-axis: f_1, f_2)
  - apply grid to the graph



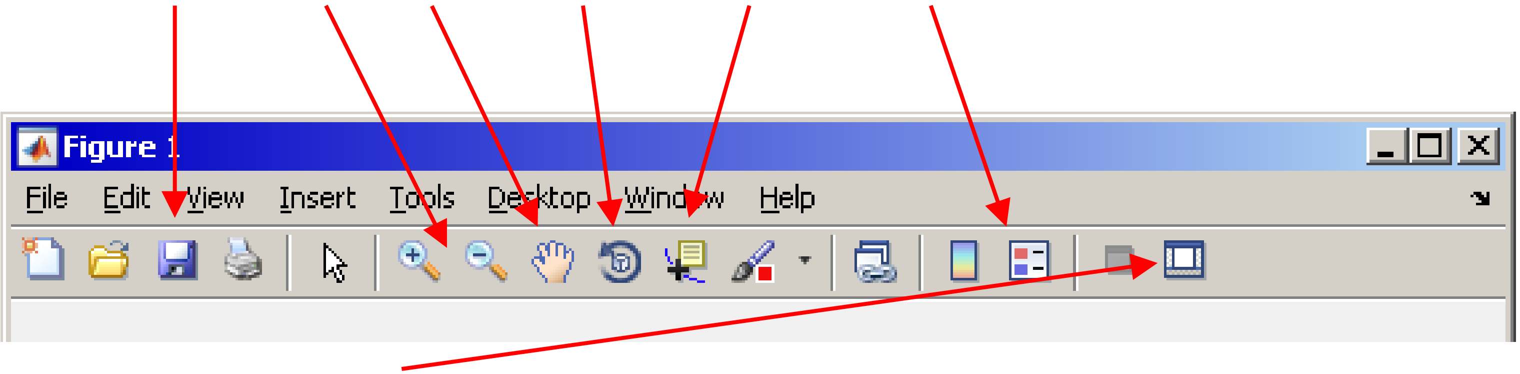
LineSpec – customizing graph curves

$$f_1(x) = \sinh(x), \qquad f_2(x) = \cosh(x)$$



Visualizing – Plot tools

- it is possible to keep on editing the graph by other means
  - save, zoom, pan, rotate, marker, legend



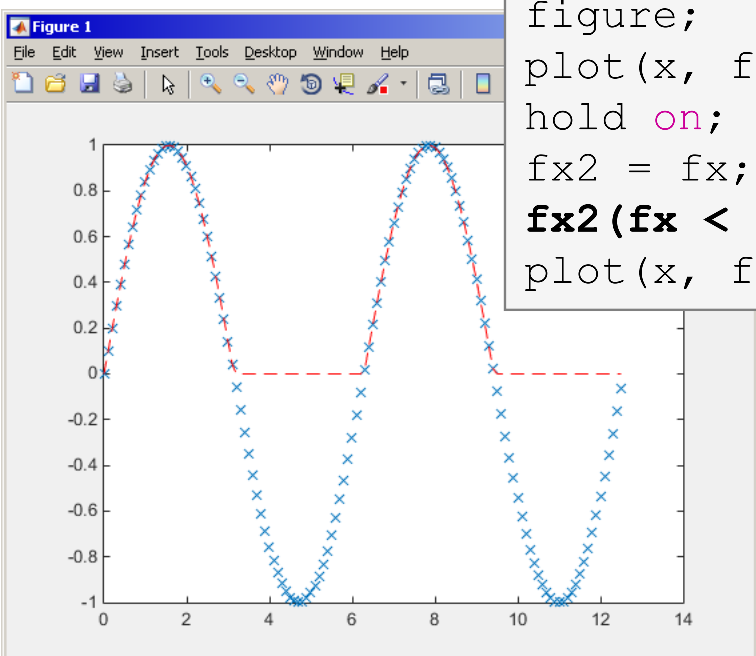
  - open Matlab Property Editor (we discuss later)

- all these operations can be carried out using Matlab functions
  - we discuss later (e.g. rotate3d activates figure's rotation tool, view(az,el) adjusts 3D perspective of the graph for given azimuth az and elevation el)

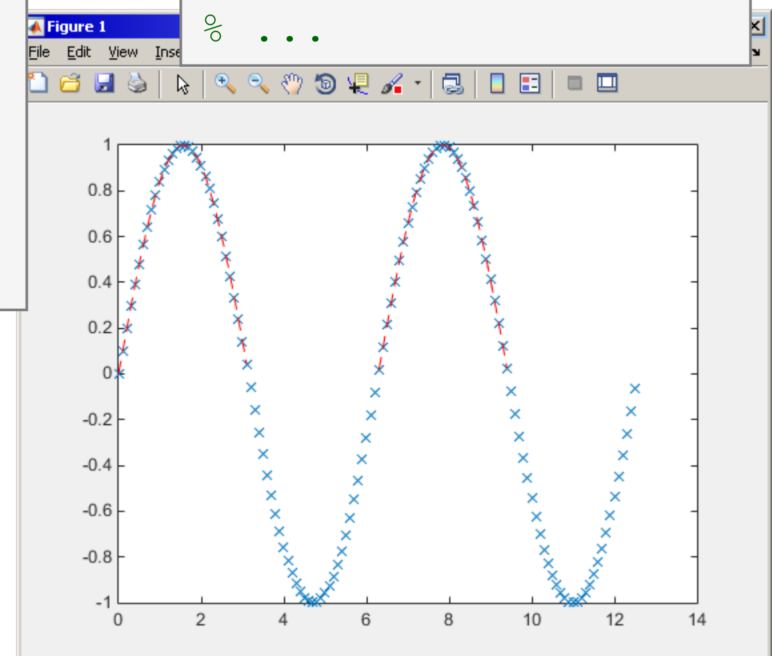
Visualizing – use of `NaN` values

- `NaN` values are not depicted in graphs
  - it is quite often needed to distinguish zero values from undefinied values
  - plotting using `NaN` can be utilized in all functions for visualizing

```
x = 0:0.1:4*pi;
fx = sin(x);
figure;
plot(x, fx, 'x');
hold on;
fx2 = fx;
fx2(fx < 0) = 0;
plot(x, fx2, 'r--');
```




```
% ...
fx2(fx < 0) = NaN;
% ...
```

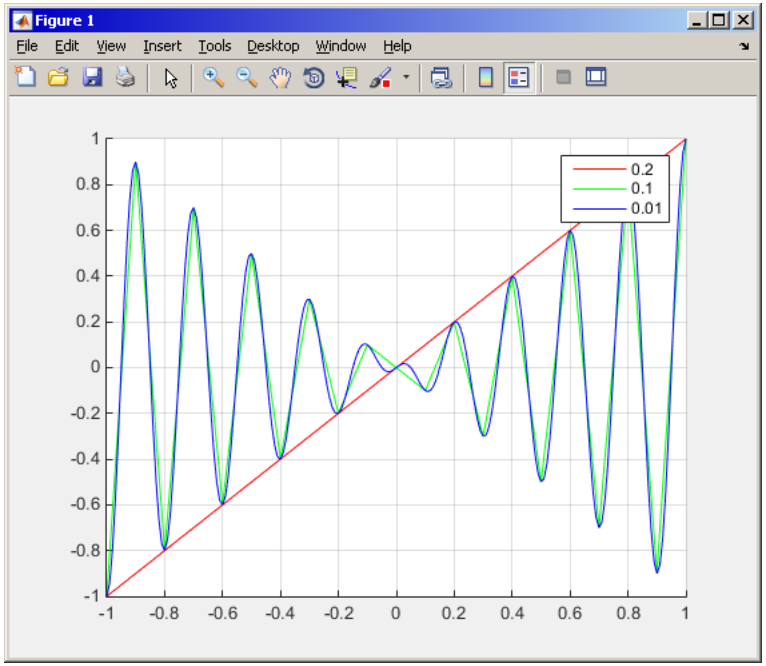


Exercise - sampling

300 s

- plot function $f(x) = x\sin\left(\frac{\pi}{2}(1+20x)\right)$ in the interval $\langle -1;\ 1\rangle$

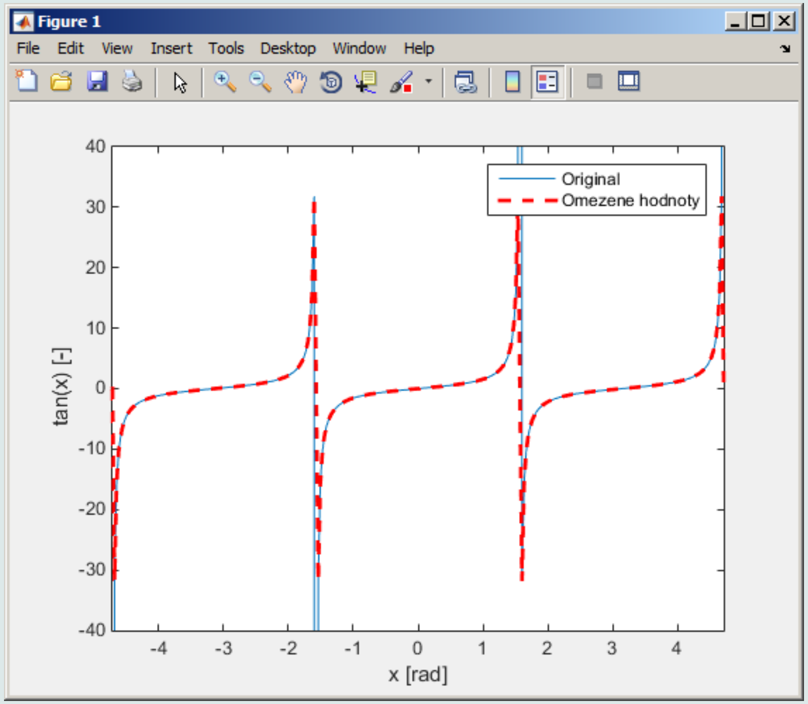
  with step 0.2, 0.1 a 0.01
- compare the results!



Exercise - rounding

300 s

- plot function `tan(x)` for $x \in \langle -3/2\pi; 3/2\pi \rangle$ with step $\pi/100$
- limit depicted values by ±40
- values of the function with absolute value greater than $1 \cdot 10^{10}$ replace by 0
  - use logical indexing
- plot both results and compare them

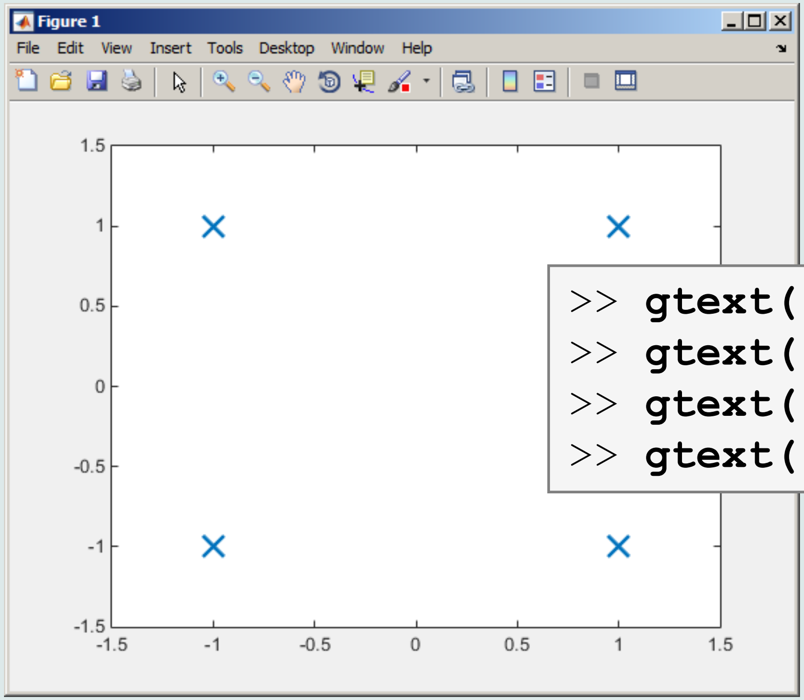


Department of Electromagnetic Field, CTU FEE, miloslav.capek@fel.cvut.cz

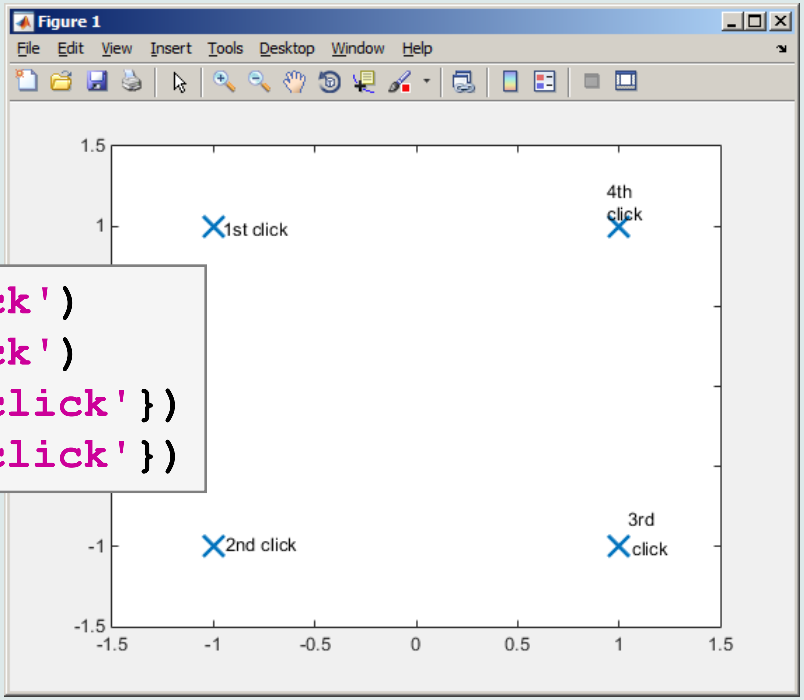
Function `gtext`

- function `gtext` enables placing text in graph
  - the placing is done by selecting a location with the mouse

```
>> plot([-1 1 1 -1], [-1 -1 1 1], ...
     'x','MarkerSize',15,'LineWidth',2);
>> xlim(3/2*[-1 1]); ylim(3/2*[-1 1]);
```

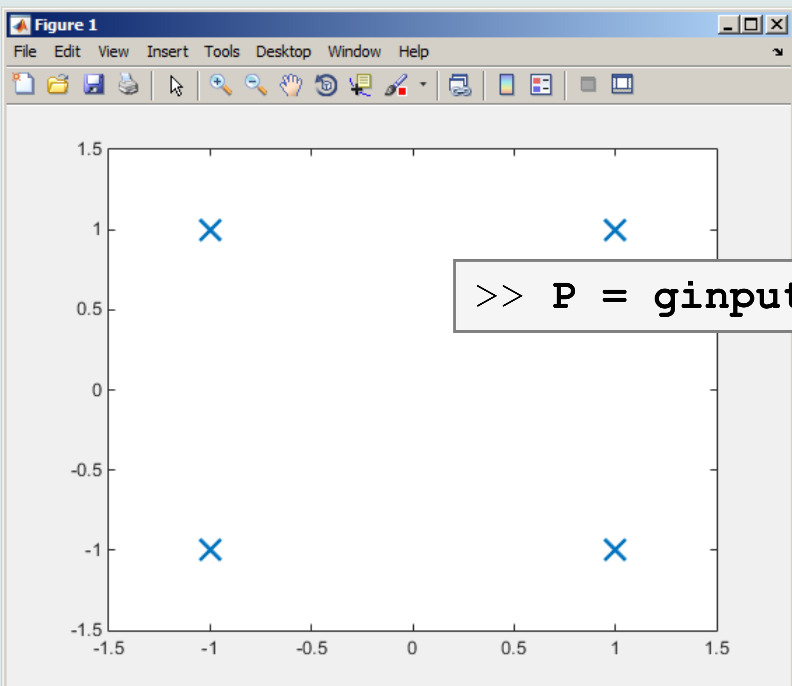



```
>> gtext('1st click')
>> gtext('2nd click')
>> gtext({'3rd';'click'})
>> gtext({'4th','click'})
```

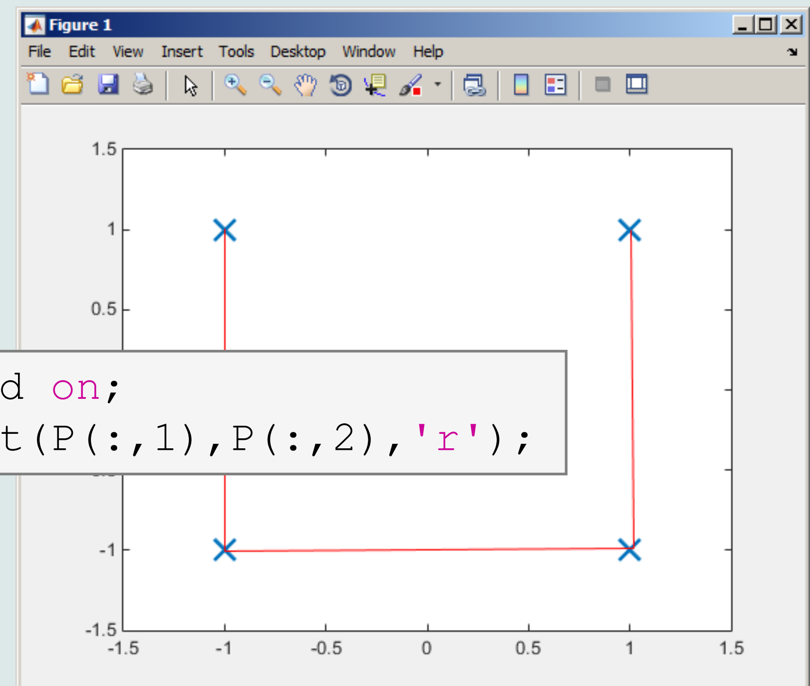


Function `ginput`

- function `ginput` enables selecting points in graph using the mouse
  - we either insert requested number of points (`P = ginput(x)`) or terminate by pressing Enter




```
>> P = ginput(4);
```




```
>> hold on;
>> plot(P(:,1),P(:,2),'r');
```


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

Debugging #1

- *bug ⇒ debugging*

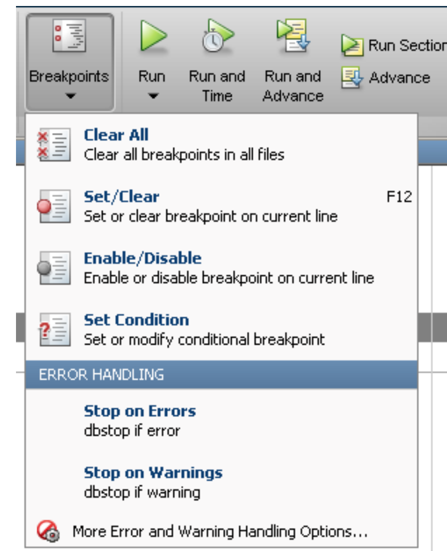
- we distinguish:
  - semantic errors (“logical“ or “algorithmic” errors)
    - usually difficult to identify
  - syntax errors (“grammatical” errors)
    - pay attention to the contents of error messages - it makes error elimination easier
  - unexpected events (see later)
    - e.g. problem with writing to open file, not enough space on disk etc.
  - rounding errors (everything is calculated as it should but the result is wrong anyway)
    - it is necessary to analyze the algorithm in advance, to determine the dynamics of calculation etc.

- software debugging and testing is an integral part of software development
  - later we will discuss the possibilities of code acceleration using `Matlab profile`

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

Debugging #2

- we first focus on semantic and syntax errors in scripts
  - we always test the program using test-case where the result is known
- possible techniques:
  - using functions `who`, `whos`, `keyboard`, `disp`
  - using debugging tools in `Matlab Editor` (illustration)



MATLAB Functions

`dbclear`	Clear breakpoints
`dbcont`	Resume execution
`dbdown`	Reverse workspace shift performed by `dbup`, while in debug mode
`dbquit`	Quit debug mode
`dbstack`	Function call stack
`dbstatus`	List all breakpoints
`dbstep`	Execute one or more lines from current breakpoint
`dbstop`	Set breakpoints for debugging
`dbtype`	List text file with line numbers
`dbup`	Shift current workspace to workspace of caller, while in debug mode
`checkcode`	Check MATLAB code files for possible problems
`keyboard`	Input from keyboard
`mlintrpt`	Run `checkcode` for file or folder, reporting results in browser

- using Matlab built-in debugging functions

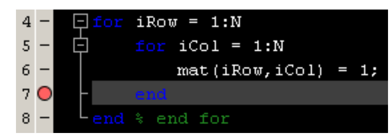
Debugging

250 s

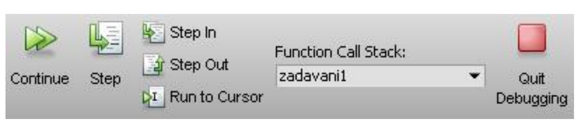
- for the following piece of code:

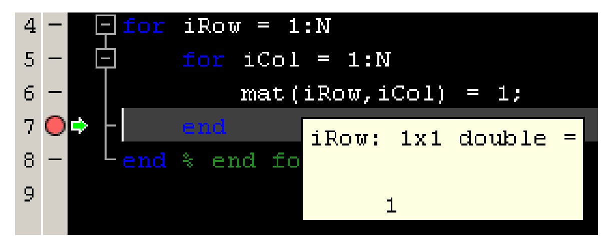
```
clear; clc;
N   = 5e2;
mat = nan(N,N);
for iRow = 1:N
    for iCol = 1:N
        mat(iRow,iCol) = 1;
    end % end for
end % end for
```

- use `Matlab Editor` to:
    - set *Breakpoint* (click on dash next to line number)



    - run the script (F5)
    - check the status of variables (`keyboard` mode or hover over variable's name with the mouse in Editor)
    - keep on tracing the script
        - what is the difference between *Continue* a *Step* (F10)?

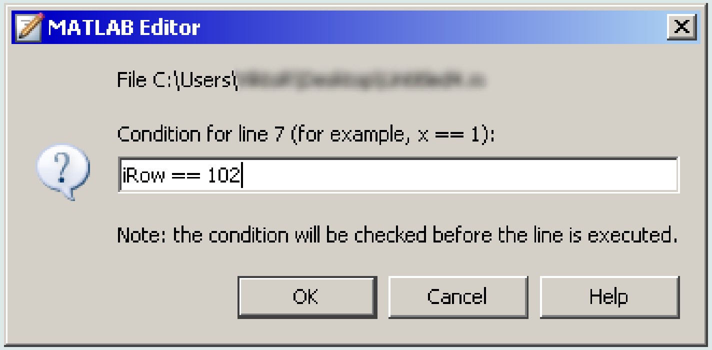


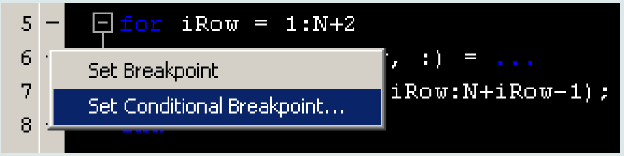


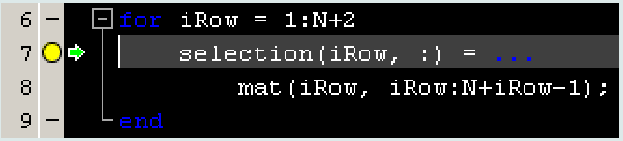
Advanced debugging

- *Conditional Breakpoints*
  - serve to suspend the execution of code when a condition is fulfilled
    - sometimes, the set up of the correct condition is not an easy task…
  - easier to find errors in loops
    - code execution can be suspended in a particular loop
  - the condition may be arbitrary evaluable logical expression

```
% code with an error
clear; clc;
N  = 100;
mat = magic(2*N);
selection = zeros(N, N);
for iRow = 1:N+2
    selection(iRow, :) = ...
        mat(iRow, iRow:N+iRow-1);
end
```







Selected hints for code readability #1

```
for iRow = 1:N
    mat(iRow,:) = 1;
end % end of ...
```

- use indention of loop's body, indention of code inside conditions (TAB)
  - size of indention can be adjusted in `Preferences` (usually 3 or 4 spaces)
- use "positive" conditions
  - i.e. use `isBigger` or `isSmaller`, not `isNotBigger` (can be confusing)
- complex expressions with logical and relational operators should be evaluated separately → higher readability of code
  - compare:

```
if (val>lowLim)&(val<upLim)&~ismember(val,valArray)
    % do something
end
```

and

```
isValid = (val > lowLim) & (val < upLim);
isNew   = ~ismember(val, valArray);
if isValid & isNew
    % do something
end
```

Selected hints for code readability #2

- code can be separated with a line to improve clarity
- use two lines for separation of blocks of code
  - alternatively use cells or commented lines `%-----------------`, etc.
- consider the use of spaces to separate operators (`= & |`)
  - to improve code readability:

```
(val>lowLim)&(val<upLim)&~ismember(val,valArray)
```

vs.

```
(val > lowLim) & (val < upLim) & ~ismember(val, valArray)
```

- in the case of nesting use comments placed after `end`

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

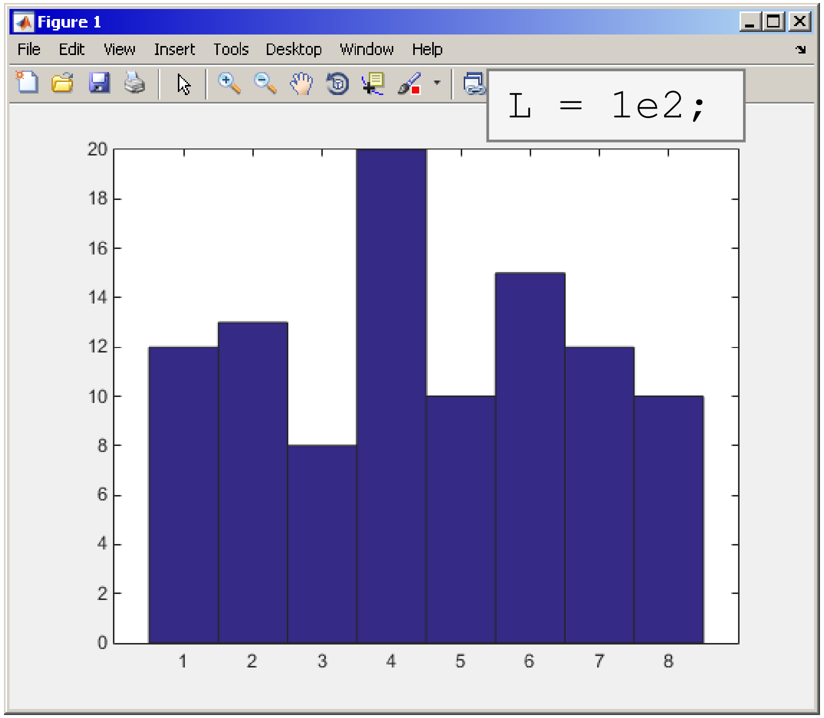
Discussed functions

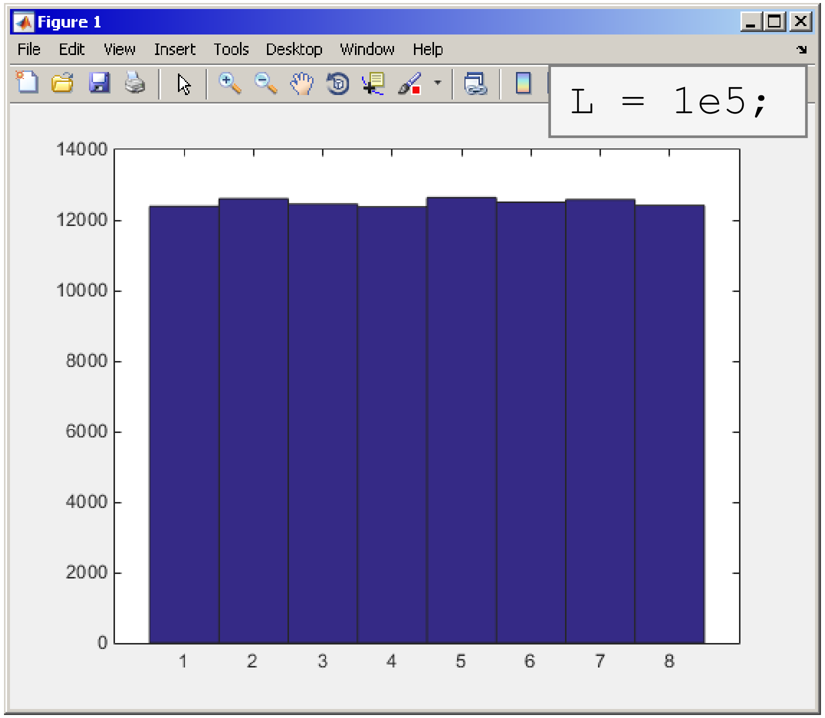
`switch-case-otherwise-end`	condition statement	●
`figure, hold`	open new figure, enable multiple curves in one axis	●
`title, xlim, ..., xlabel, ...`	heading, axes limits, axes labels	●
`legend, grid`	legend, grid	●
`gtext, ginput`	interactive text insertion, interactive input from mouse or cursor	●

Exercise #1

600 s

- create a script to simulate `L` roll of the dice
  - what probability distribution do you expect?
  - use `histogram` to plot the result
  - consider various number of tosses `L` (from tens to millions)





Exercise #2

600 s

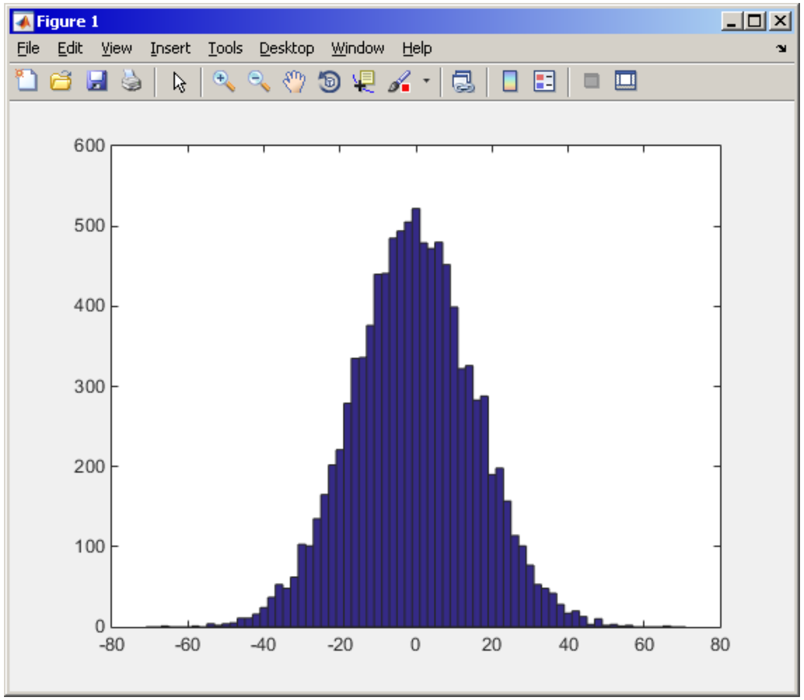
- create a script to simulate N series of trials, where in each series a coin is tossed `M` times (the result is either head or tail)
  - generate a matrix of tosses (of size M×N)
  - calculate how many times head was tossed in each of the series (a number between 0 and M)
  - calculate how many times more (or less) the head was tossed than the expected average (given by uniform probability distribution)
  - what probability distribution do you expect?
  - plot resulting deviations of number of heads
    - use function `histogram()`

Exercise #3

- mean and standard deviation :

$$N = 1 \cdot 10^4 :$$

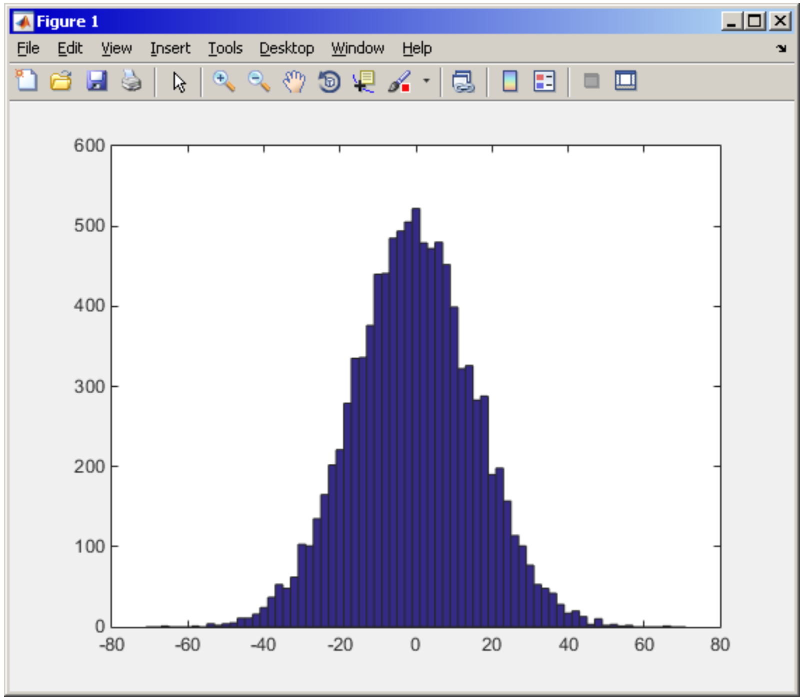
$$\mu = \frac{1}{N}\sum_i x_i \approx 0 \qquad \sigma = \sqrt{\frac{\sum_i (\mu - x_i)^2}{N}} = 15.7742$$



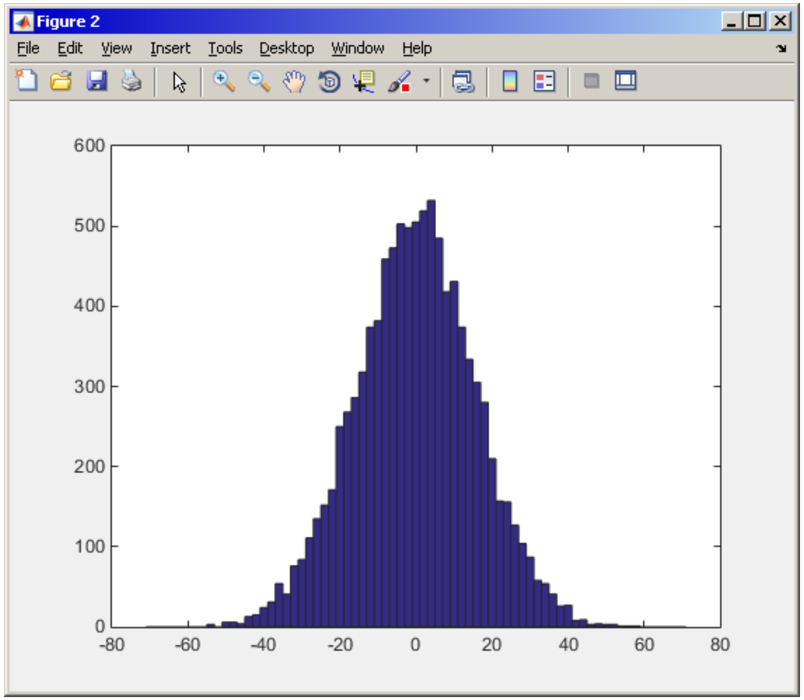
Exercise #4

- to test whether we get similar distribution for directly generated data:

coin toss:

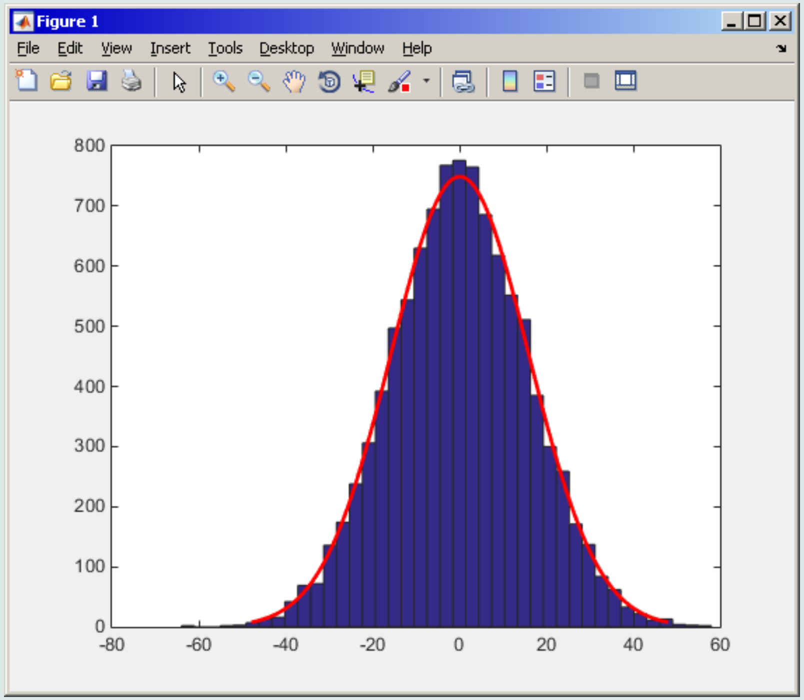


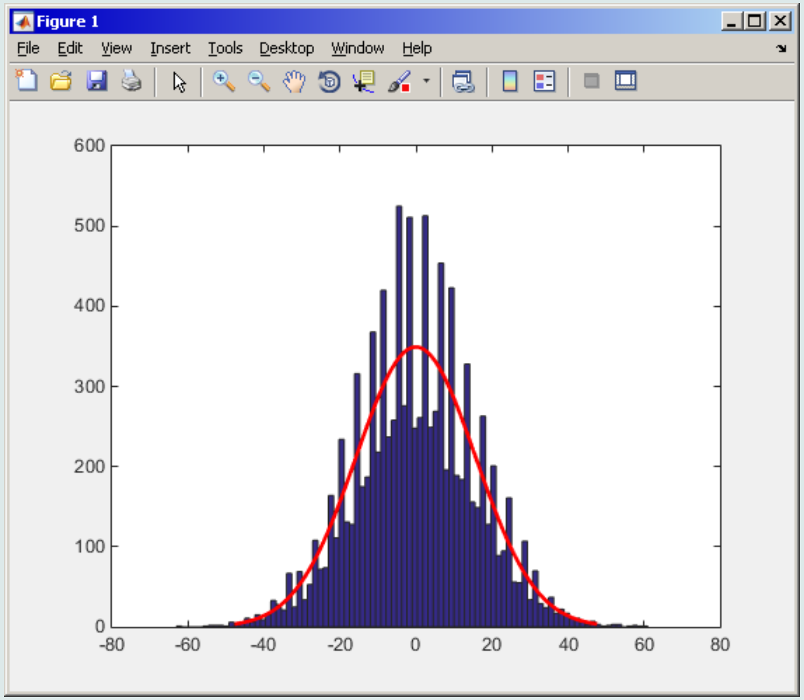
directly generated data:



Exercise #5

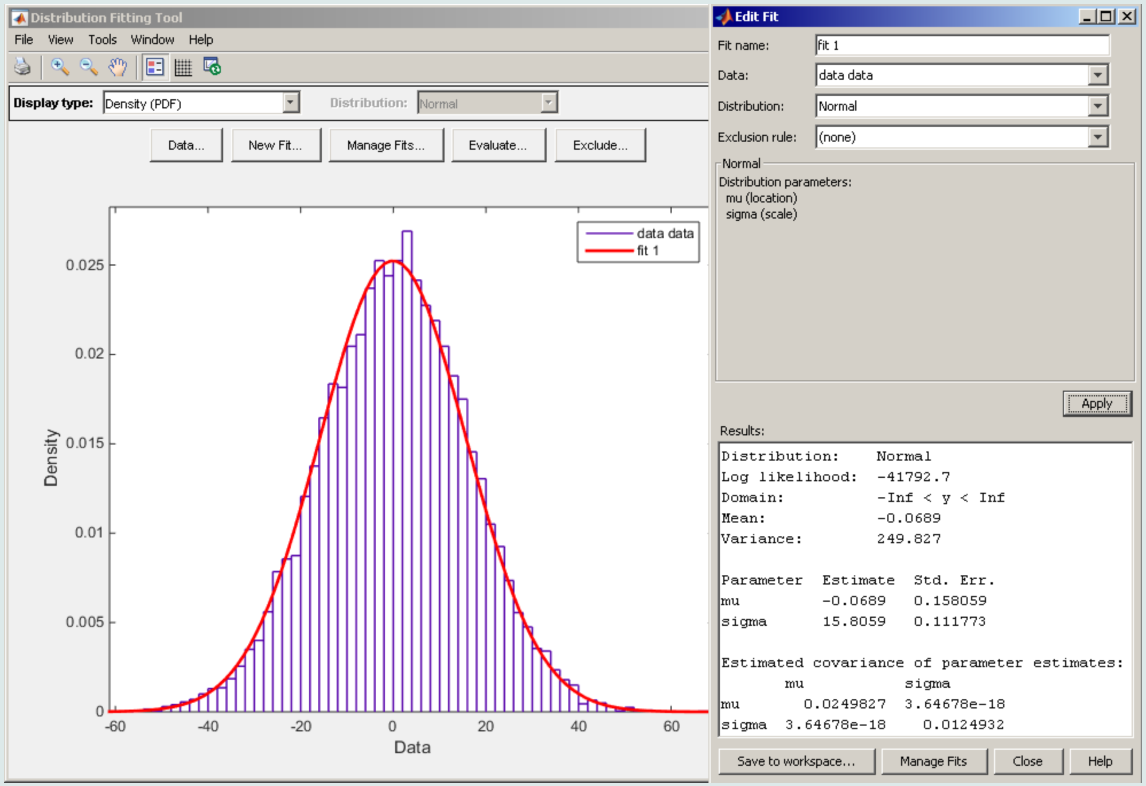
- use function `histfit` (Statistics Toolbox) to plot probability density function related to a histogram
  - set the parameter `nbins` accordingly to properly display histogram of discrete random variable





Exercise #6

- use Distribution Fitting Tool (`dfittool`) to approximate probability distributions of random trials



Exercise #7

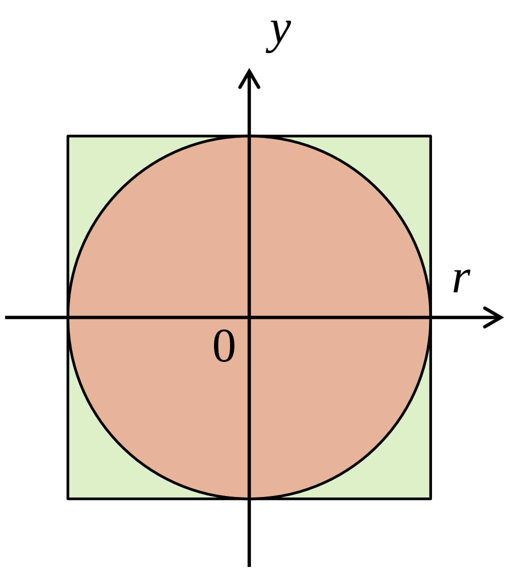
600 s ↑

- use Monte Carlo method to estimate the value of π
  - Monte Carlo is a stochastic method using pseudorandom numbers
- The procedure is as follows:

  (1) generate points (uniformly distributed) in a given rectangle

  (2) compare how many points there are in the whole rectangle and how many there are inside the circle

$$\frac{S_{\mathrm{o}}}{S_{\square}} = \frac{\pi r^2}{(2r)^2} = \frac{\pi}{4} \approx \frac{\text{hits}}{\text{shots}}$$



- write the script in the way that the number of points can vary
  - notice the influence of the number of points on accuracy of the solution

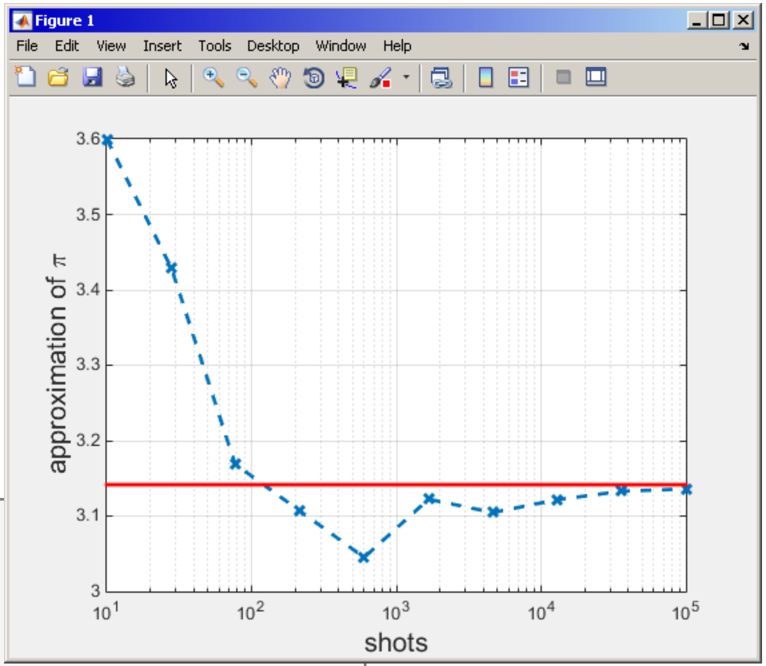
Exercise #7- solution

- resulting code (circle radius $r = 1$):

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

Exercise #8

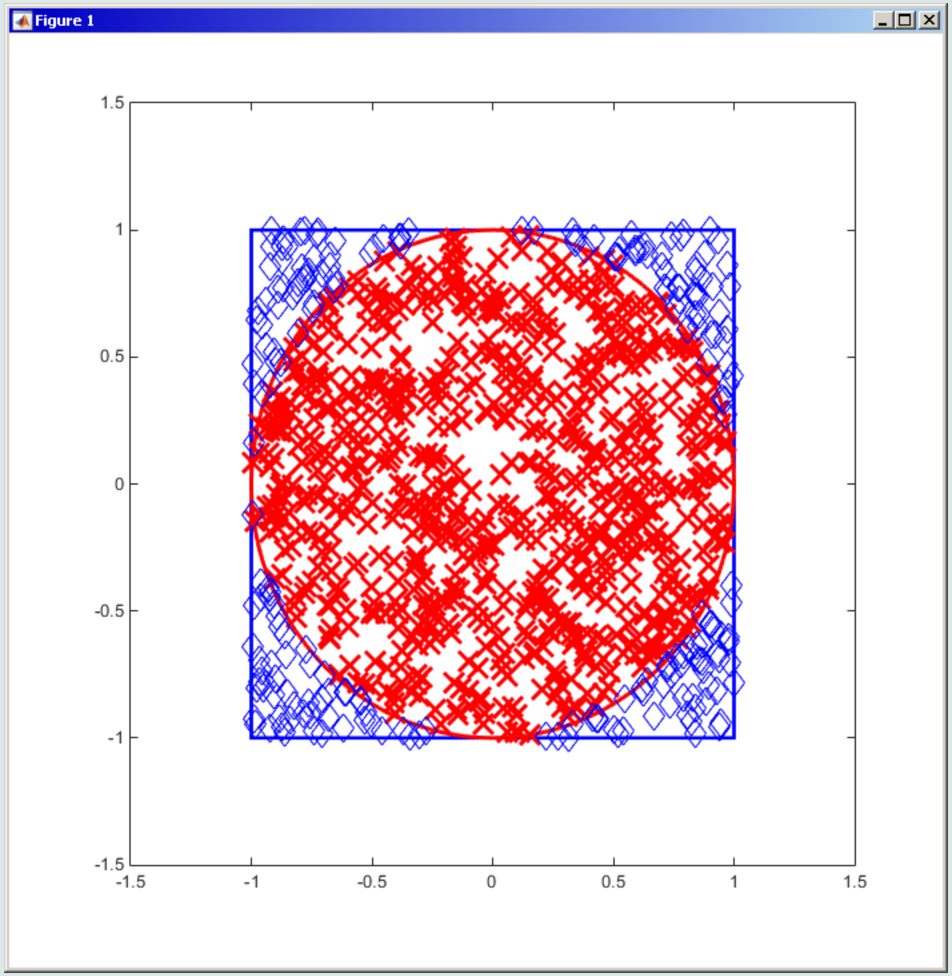
- approximation of Ludolph's number - visualization:




```
figure;
semilogx(N, my_pi, 'x--', 'linewidth',2);
xlim([N(1) N(end)]);
hold on; grid on;
xlabel('shots','FontSize', 15);
ylabel('approximation of \pi','FontSize', 15);
line([N(1) N(end)],[pi pi],'color','r','linewidth',2);
```

Exercise #9

- visualization of the task:



Exercise #10

600 s ↑

- following expansion holds true:

$$\arctan(x) = \sum_{n=0}^{\infty}(-1)^n \frac{(x)^{2n+1}}{2n+1} = x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \frac{x^9}{9} - \ldots$$

- based on the expansion for $x = 1$ estimate value of π:

$$\arctan(1) = \frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \ldots$$

- determine the number of elements of the sum and computational time required to achieve estimation accuracy better than $1 \cdot 10^{-6}$

Exercise #11

600 s ↑

- estimate value of π using following expansion

$$\frac{\pi}{8} = \sum_{n=0}^{\infty} \frac{1}{(4n+1)(4n+3)} = \frac{1}{1\cdot 3} + \frac{1}{5\cdot 7} + \frac{1}{9\cdot 11} + \ldots$$

- determine the number of elements of the sum and computational time required to achieve estimation accuracy better than $1\cdot 10^{-6}$

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

Exercise #12

600 s

- use following expression to approximate π :

$$\frac{\pi}{4} = 6\arctan\left(\frac{1}{8}\right) + 2\arctan\left(\frac{1}{57}\right) + \arctan\left(\frac{1}{239}\right)$$

- use following expression to implement the arctan function :

$$\arctan(x) = \sum_{n=0}^{\infty}(-1)^n \frac{(x)^{2n+1}}{2n+1} = x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \frac{x^9}{9} - \dots$$

- determine the number of elements of the sum and computational time required to achieve estimation accuracy better than $1\cdot10^{-6}$ and compare the solution with previous solutions

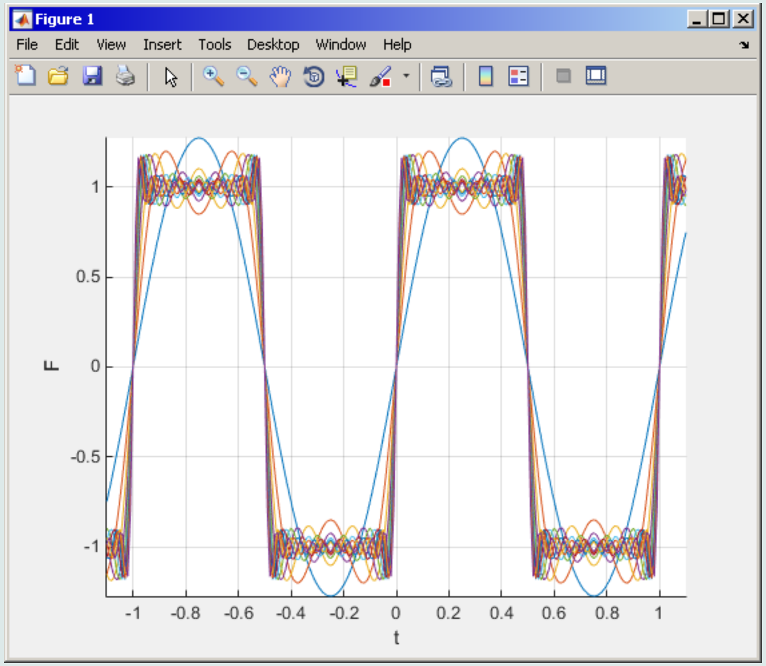
Exercise #13

600 s

- Fourier series approximation of a periodic rectangular signal with zero direct component, amplitude A and period T is

$$s(t) = \frac{4A}{\pi} \sum_{k=0}^{\infty} \frac{1}{2k+1} \sin\left(\frac{2\pi t(2k+1)}{T} \right)$$

- plot resulting signal $s(t)$ approximated by one to ten harmonic components in the interval $t \in \langle -1.1; 1.1 \rangle$ s; use A=1 V a T=1 s



Thank you!

ver. 5.2 (04/04/2016)
Miloslav Čapek, Miloslav Čapek
miloslav.capek@fel.cvut.cz

Document created as part of A0B17MTB course.