A0B17MTB – Matlab

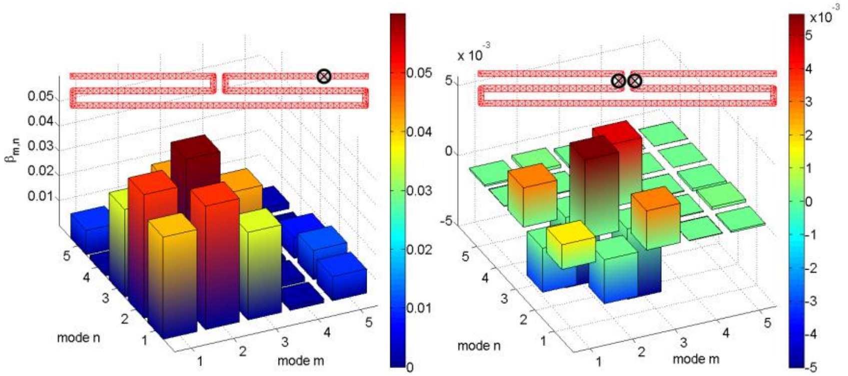
Part #6

Miloslav Čapek
miloslav.capek@fel.cvut.cz

Viktor Adler, Pavel Valtr, Filip Kozák

Department of Electromagnetic Field
B2-634, Prague

Learning how to …



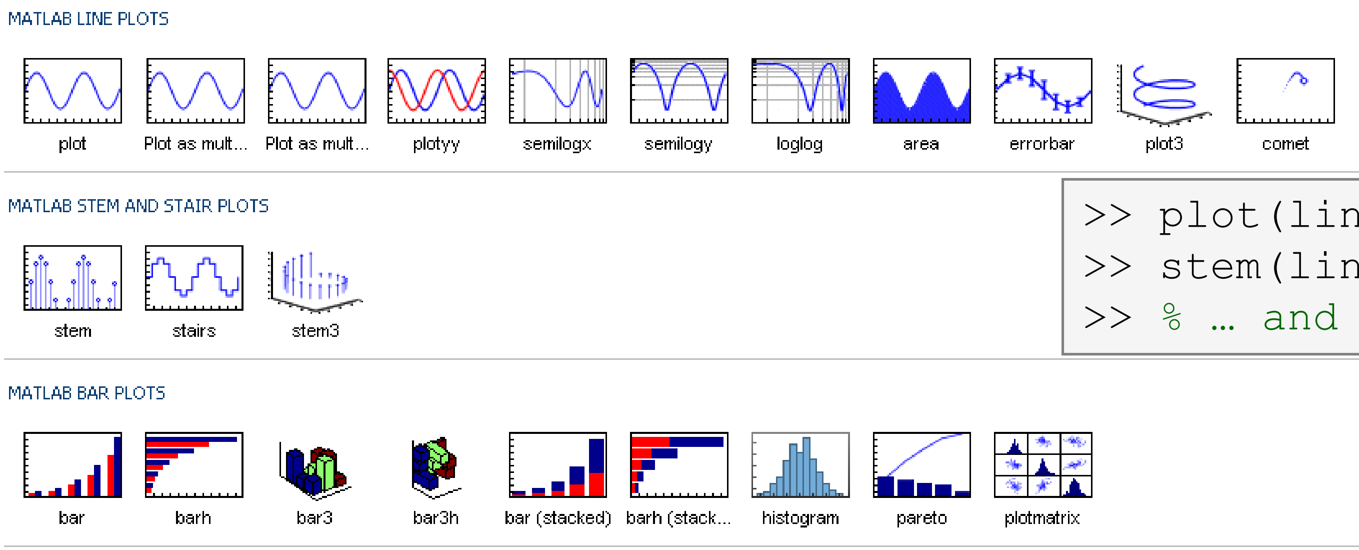
Visualizing in Matlab #1

Debugging

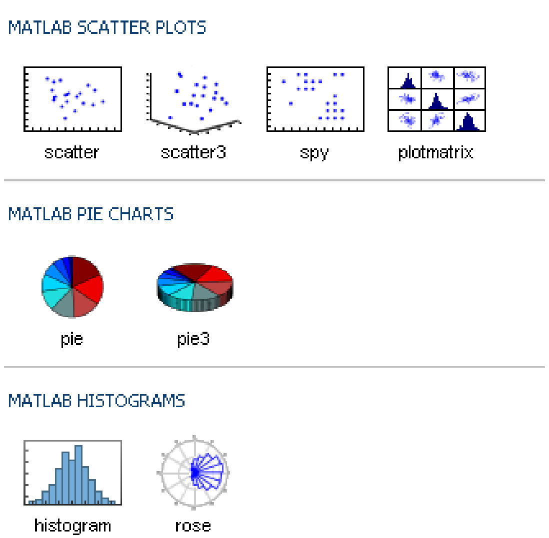
Introduction to visualizing

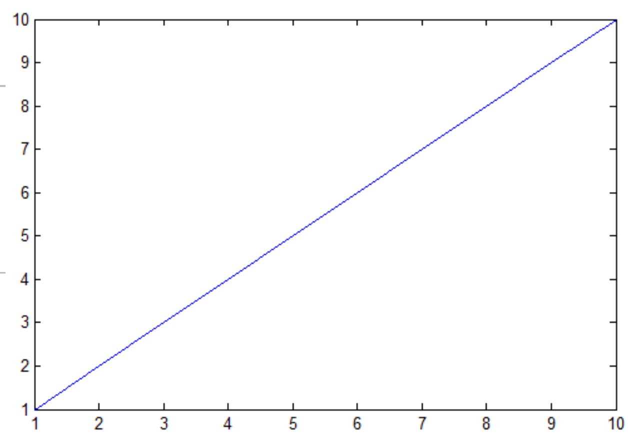
- we have already got acquainted (marginally) with some of Matlab graphs
  - `plot`, `stem`, `semilogx`, `surf`, `pcolor`
- in general, graphical functions in Matlab can be used as
  - higher level
    - access to individual functions, object properties are adjusted by input parameters of the function
    - 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 help:
  - Matlab → Graphics

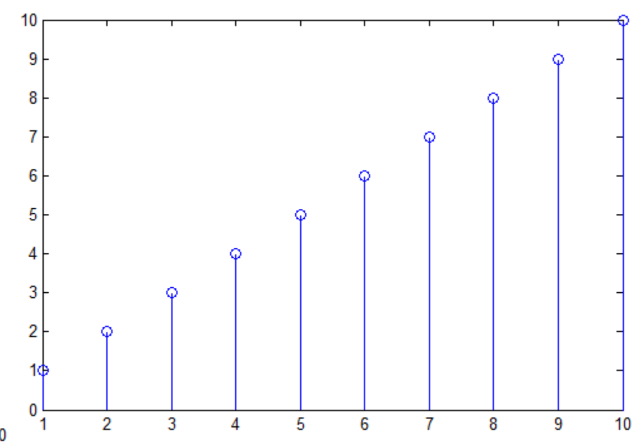
Selected graphs #1



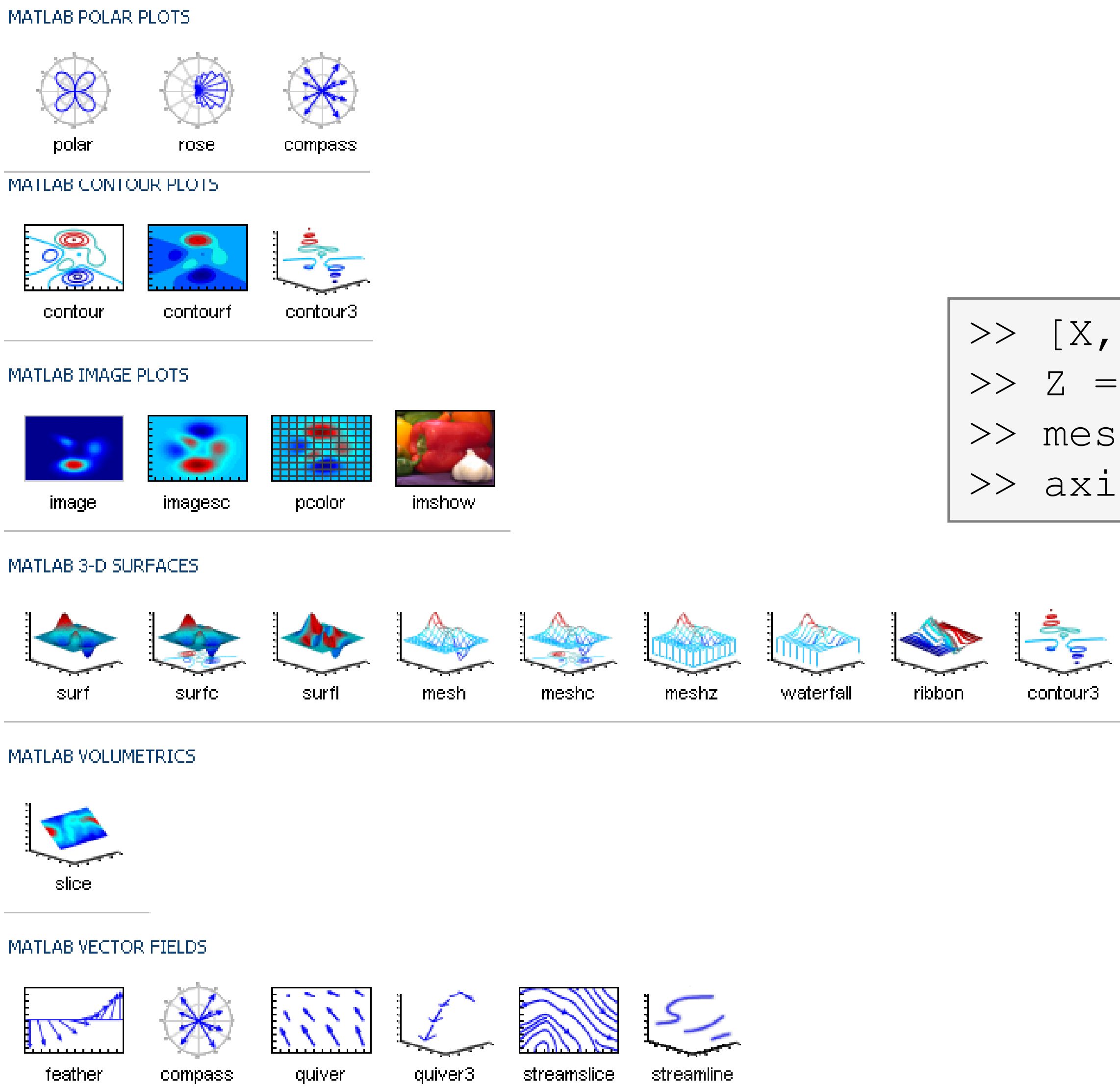

```
>> plot(linspace(1,10,10));
>> stem(linspace(1,10,10));
>> % … and others
```



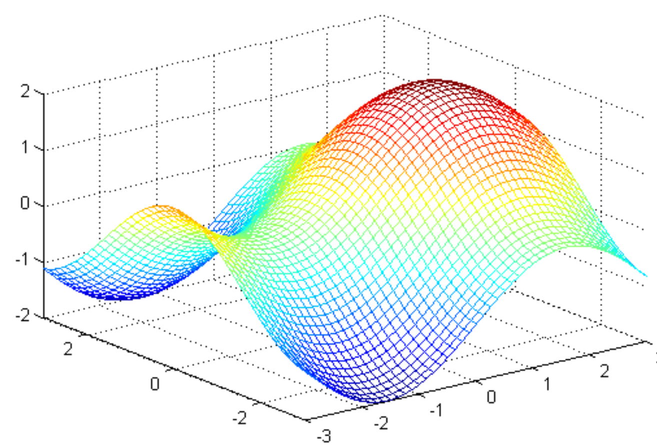




Selected graphs #2




```
>> [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:

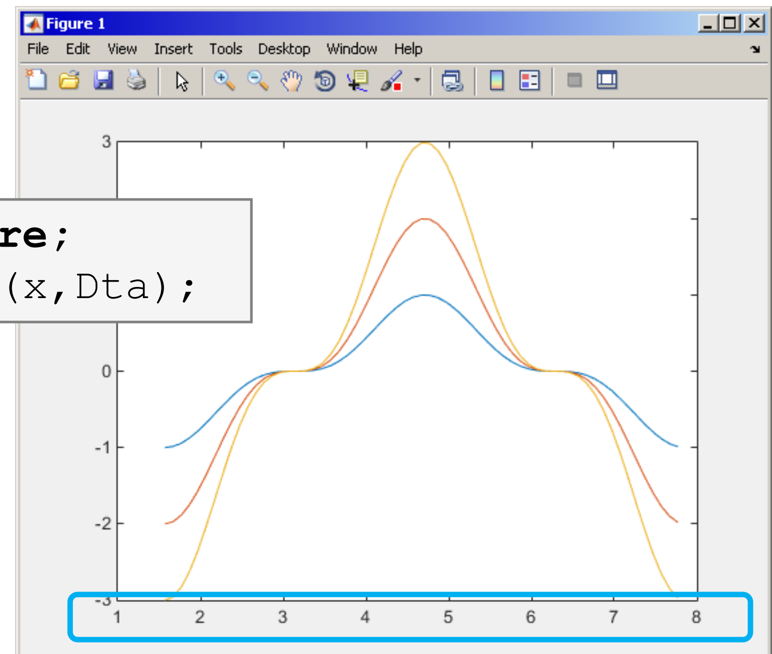
function	description
`title`	title of the graph
`grid on, grid off`	turns grid on / off
`xlim, ylim, zlim`	set axes' range
`xlabel, ylabel, ...`	label axes
`hold on`	enables to add another graphical elements while keeping the existing ones
`legend`	display legend
`subplot`	open more axes in one figure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

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

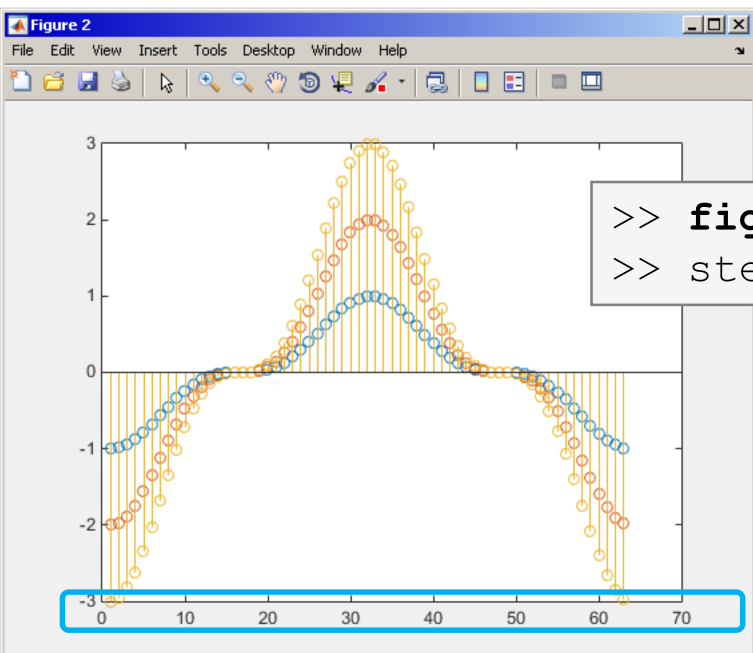
figure

- `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.');
```

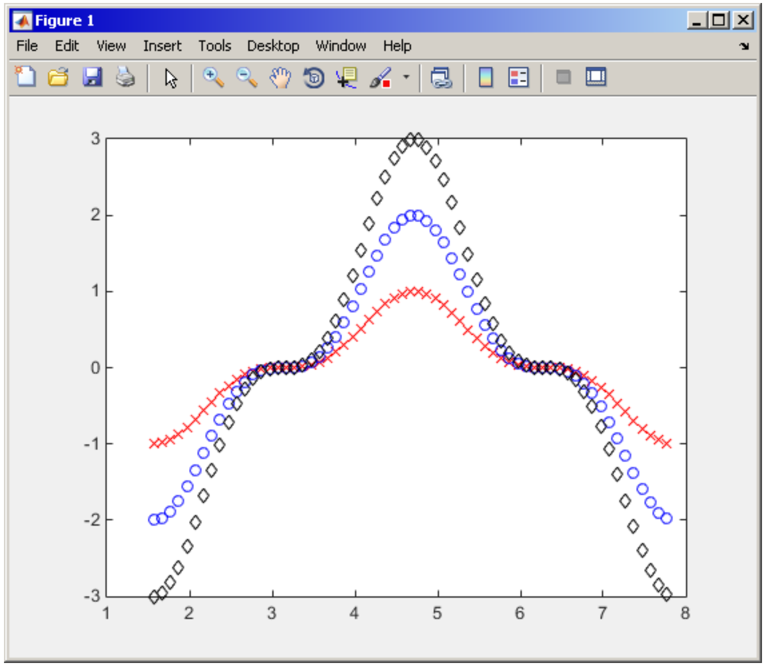
- it is possible to plot matrix data (column-wise)
- don't forget about x-axis data!

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

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

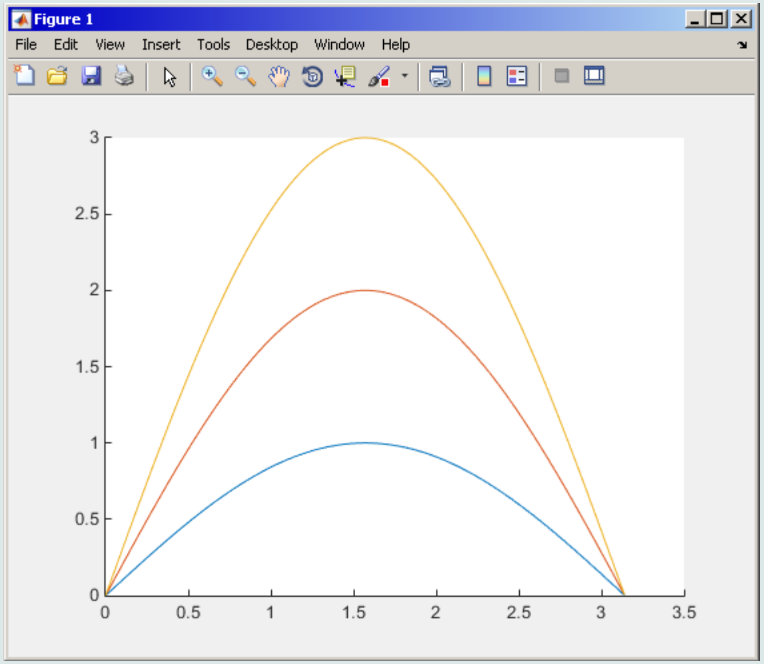
LineSpec – default setting in 2014b

- 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));
```

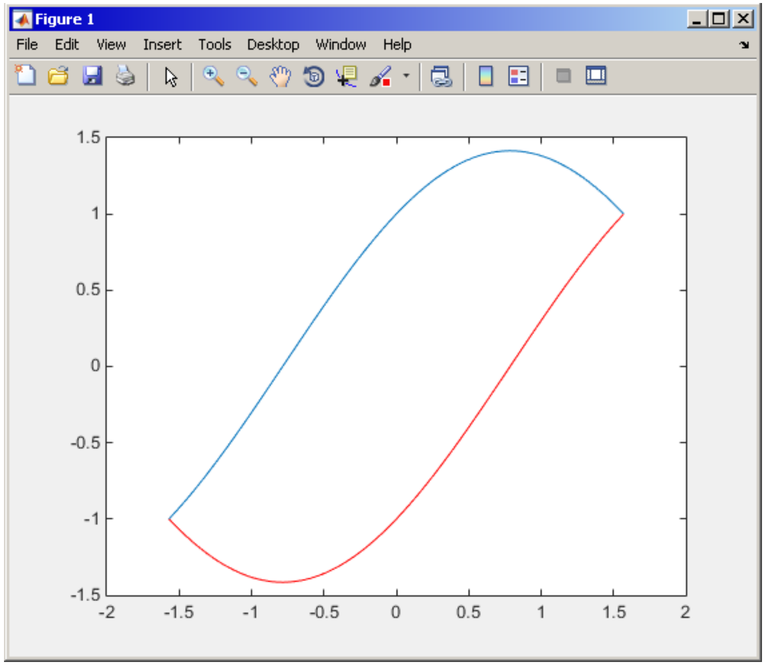


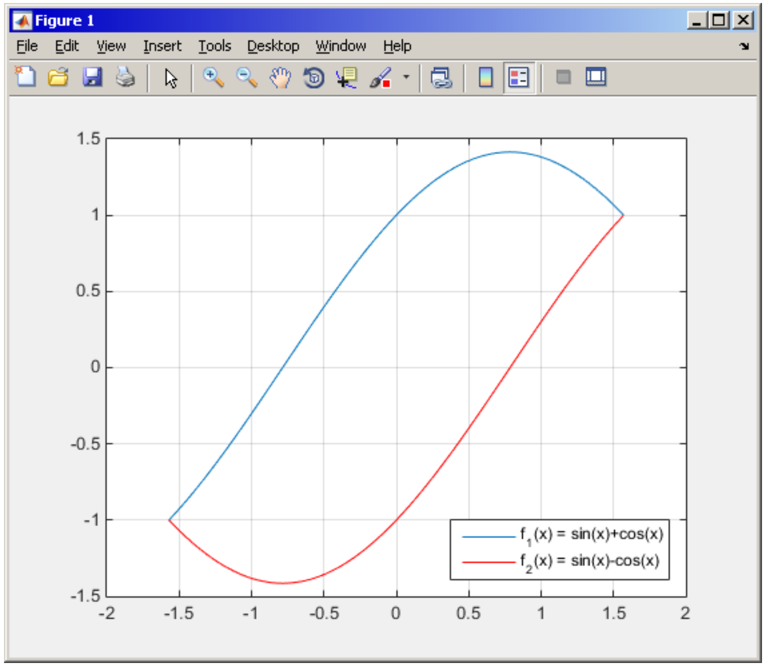
Visualizing – `legend`, `grid`

```
x = -pi/2:0.1:pi/2;
f1 = sin(x) + cos(x);
f2 = sin(x) - cos(x);
```

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

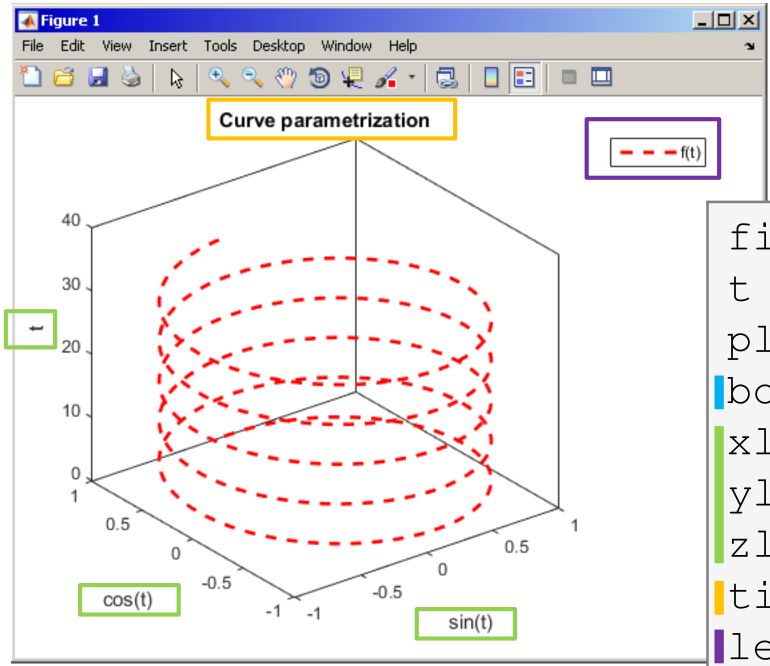
```
grid on;
legend('f_1(x) = sin(x)+cos(x)',...
    'f_2(x) = sin(x)-cos(x)',...
    'Location', 'southeast');
```





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

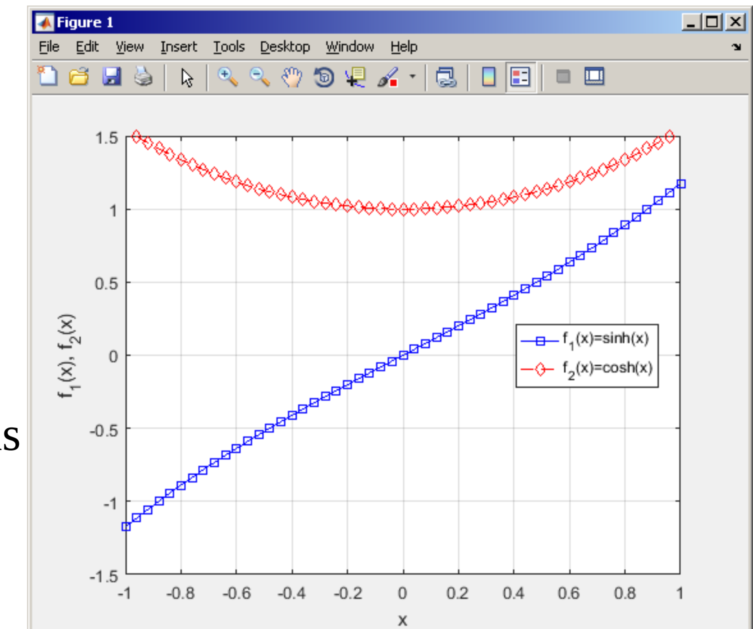
LineSpec – customizing graph curves

450 s ↑

- evaluate following two functions in the interval $x \in \langle -1, 1 \rangle$ for 51 values:

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

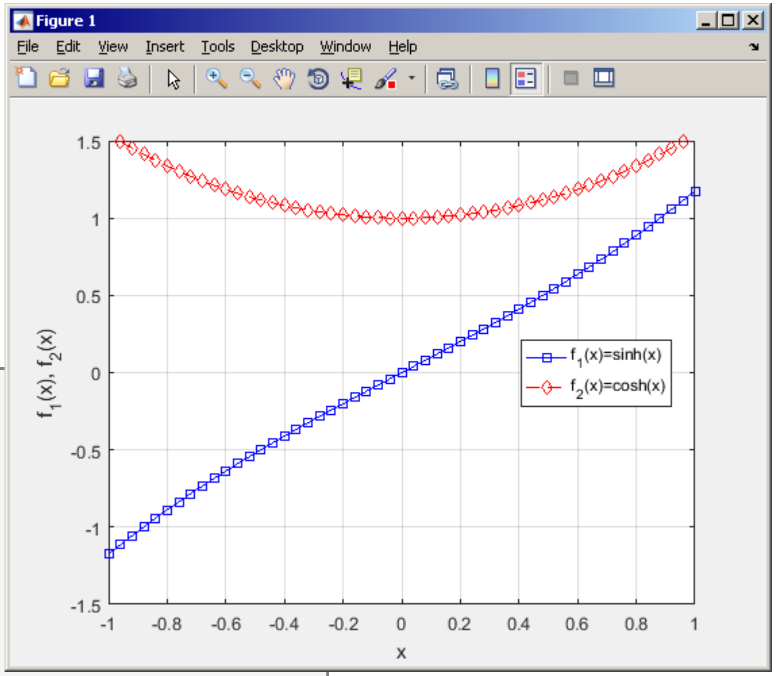
- use the function `plot` to depict both f_1 and f_2 so that
  - both functions are plotted in the same axis
  - the first function is plotted in blue with □ marker as solid line
  - the other function is plotted in red with ◊ marker and dashed line
  - limit the interval of the *y*-axis to [-1.5, 1.5]
  - 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

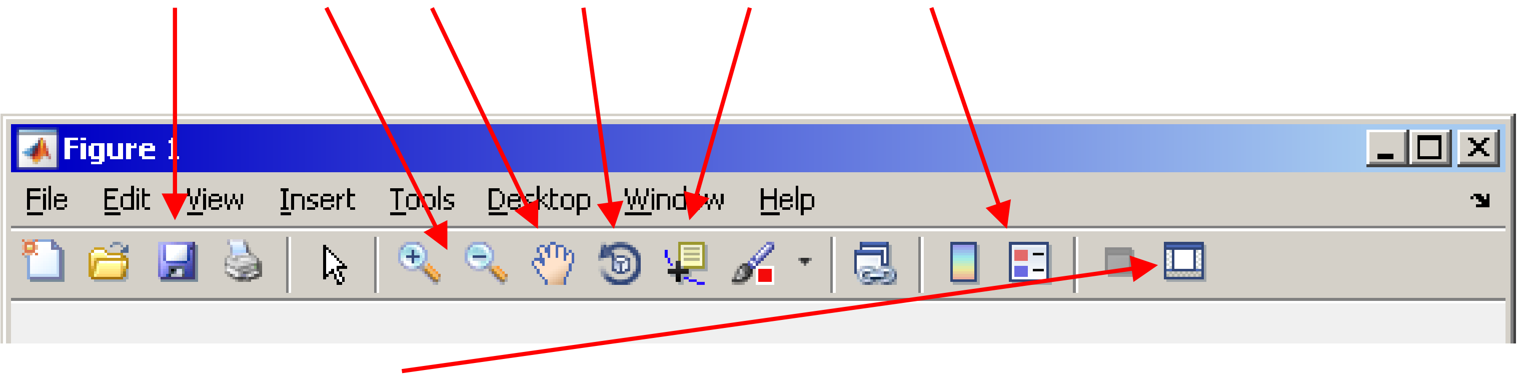
$$f_1(x) = \sinh(x), \qquad f_2(x) = \cosh(x)$$

```
%% script evaluates two hyperbolic
%  functions and plot them
...
...
...
...
...
...
...
...
...
...
...
...
```



Visualizing – `Plot tools`

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



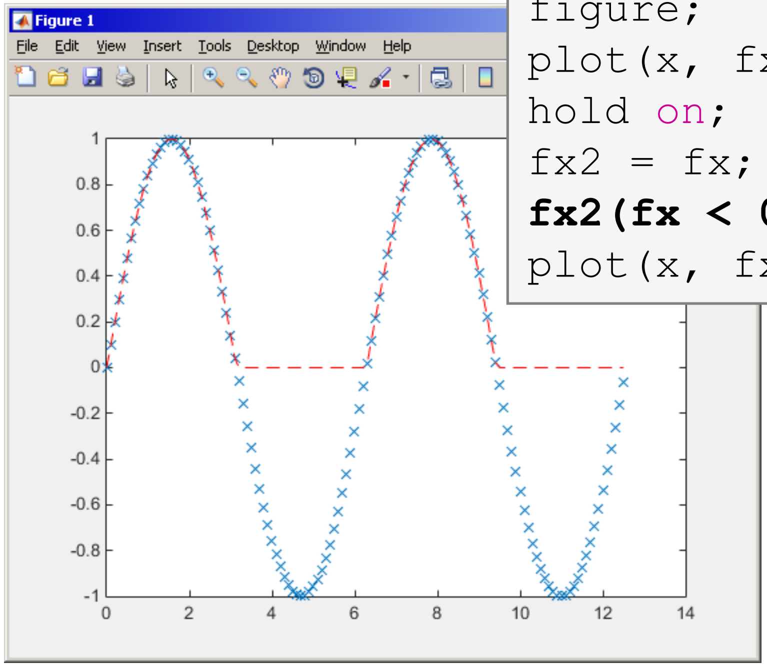
  - show plot tools (`showplottool`)

- 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`)

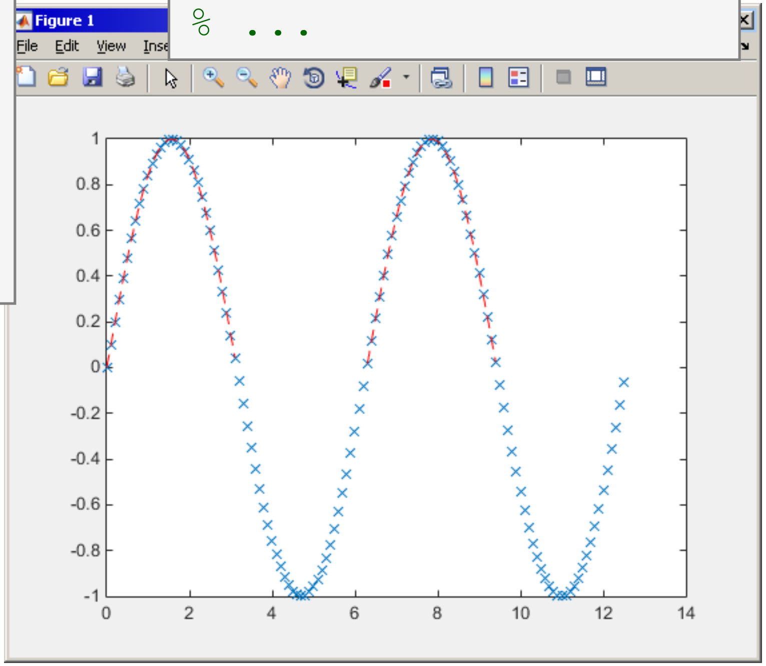
Visualizing – use of `NaN` values

- `NaN` values are not depicted in graphs
  - it is quite often needed to distinguish zero values from undefined 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;
% ...
```

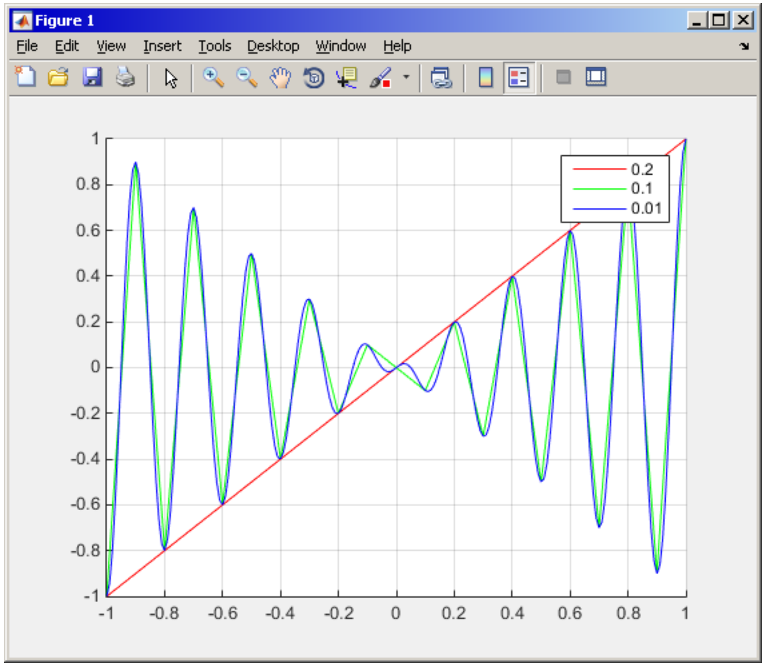


Exercise - sampling

300 s ↑

- plot function $f(x) = x \sin\left(\frac{\pi}{2}(1+20x)\right)$ in the interval $x \in \langle -1;1 \rangle$ with step 0.2, 0.1 a 0.01
- compare the results!

```
close all; clear; clc;
...
...
...
...
...
...
...
...
...
...
...
...
```

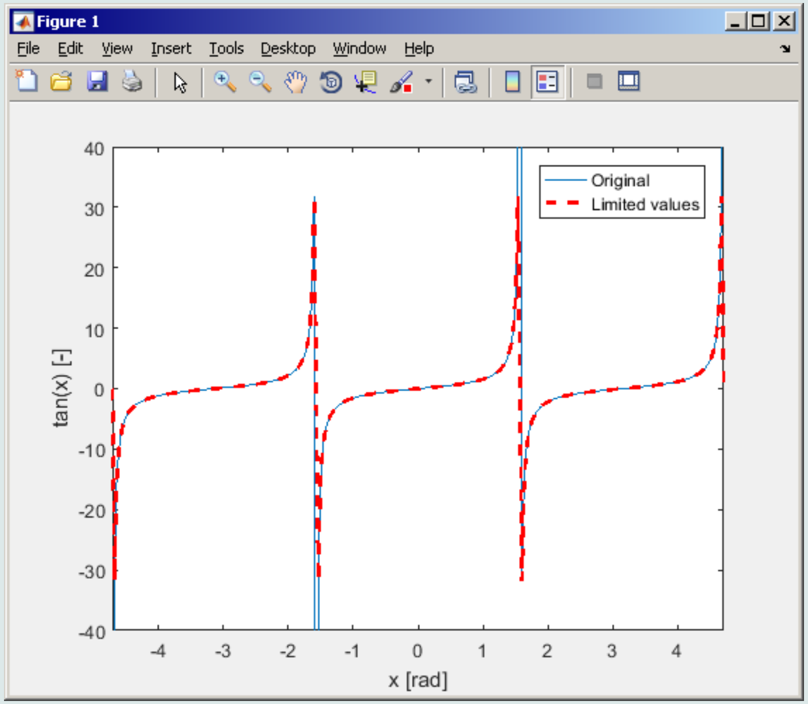


Exercise - rounding

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

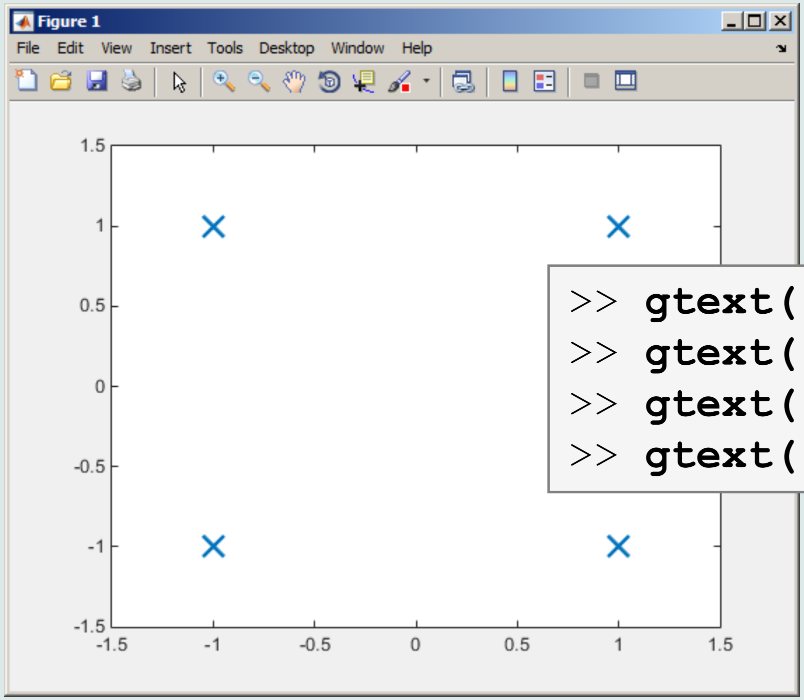
```
close all; clear; clc
x = -(3/2)*pi:pi/100:(3/2)*pi;
y = tan(x);
z = y.*(abs(y) < 1e10);
figure;
plot(x, y);
hold on;
plot(x, z, '--r', 'LineWidth', 2);
axis([-(3/2)*pi, (3/2)*pi, -40, 40]);
legend('Original', 'Limited values');
xlabel('x [rad]');
ylabel('tan(x) [-]');
```



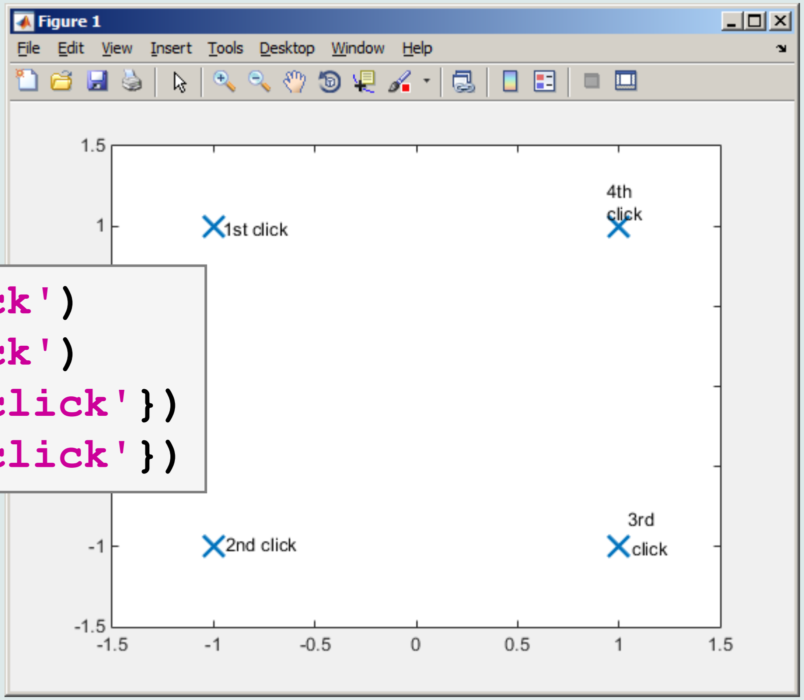
Function `gtext`

- 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]);
```

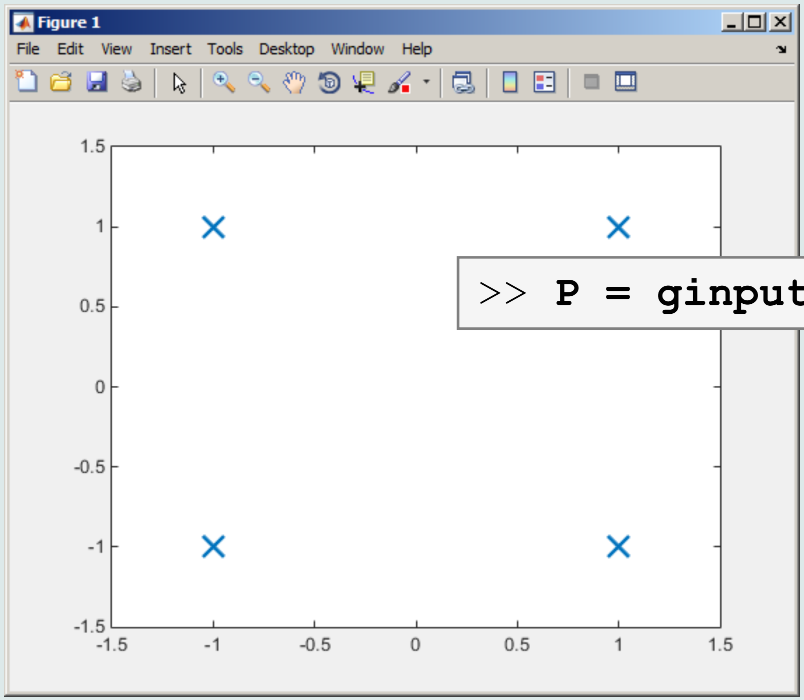



```
>> gtext('1st click')
>> gtext('2nd click')
>> gtext({'3rd';'click'})
>> gtext({'4th','click'})
```



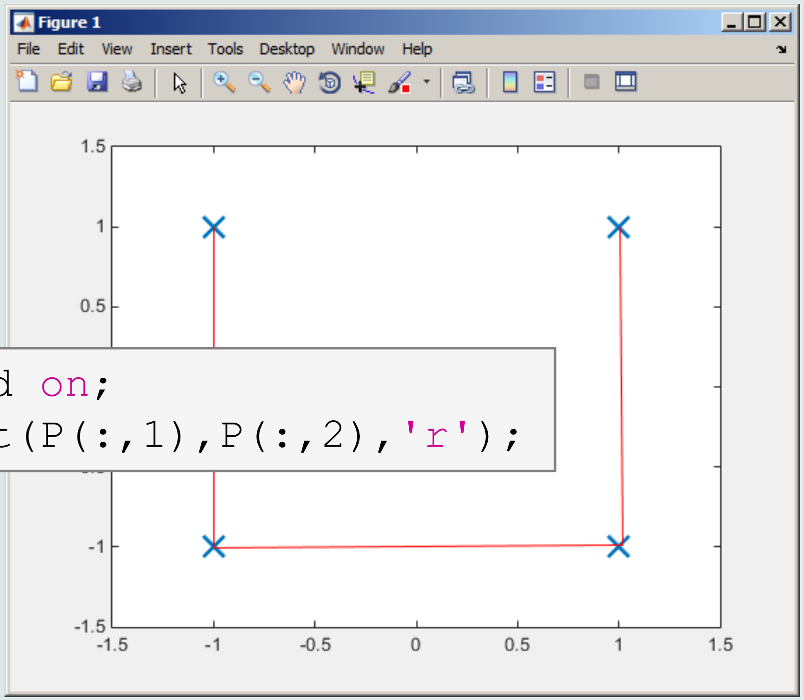
Function `ginput`

- 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');
```



Debugging #1

- *bug ⇒ debugging*

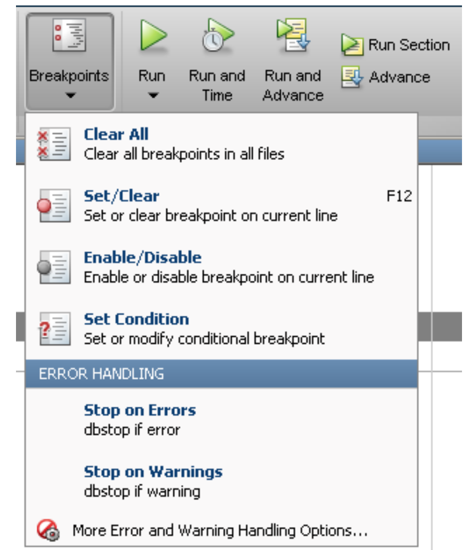
- we distinguish:
  - semantic errors (“logical“ or “algorithmic” errors)
    - usually difficult to identify
  - syntax errors (“grammatical” errors)
    - pay attention to the contents of error messages - it makes error elimination easier
  - unexpected events (see later)
    - e.g. problem with writing 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

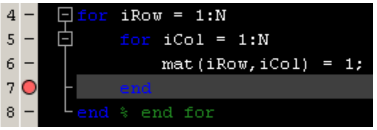
Debugging

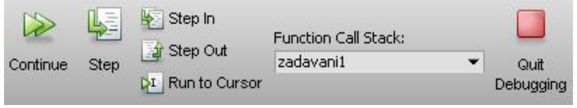
250 s ↑

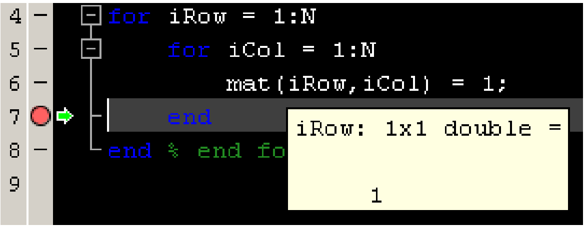
- for the following piece of code:

```
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)?





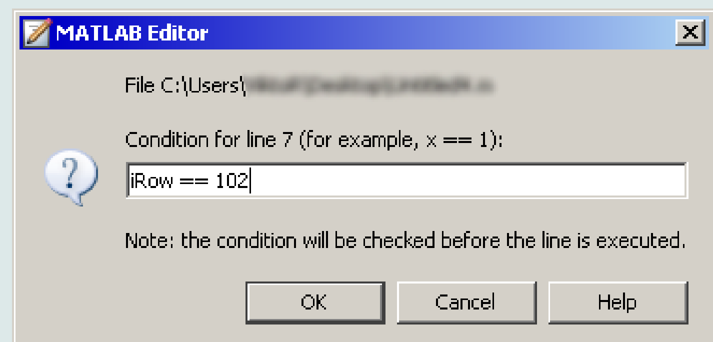


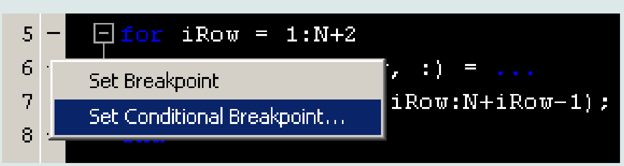
Department of Electromagnetic Field, CTU FEE, miloslav.capek@fel.cvut.cz

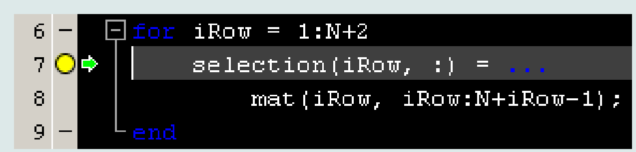
Advanced debugging

- *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

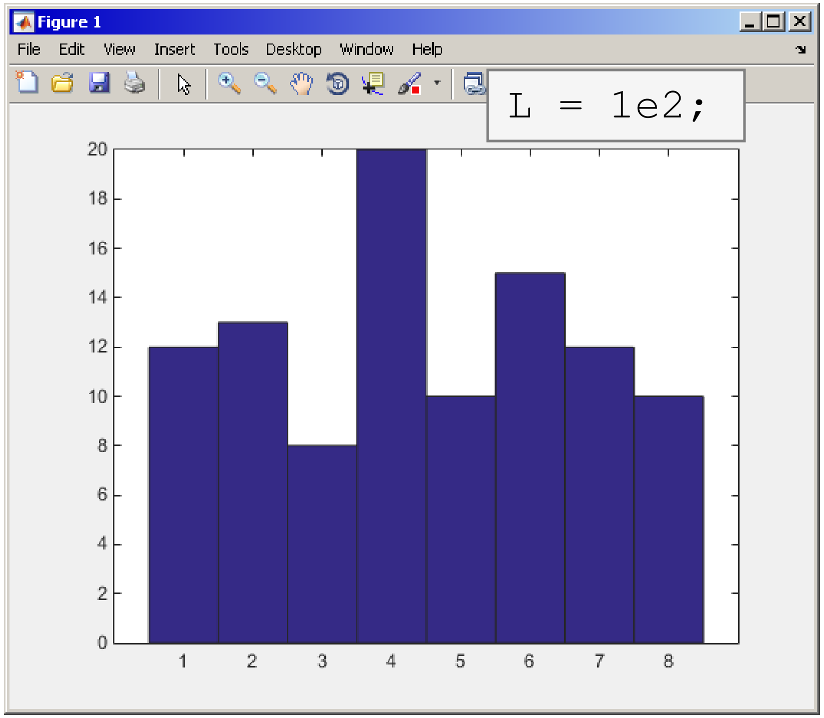
Discussed functions

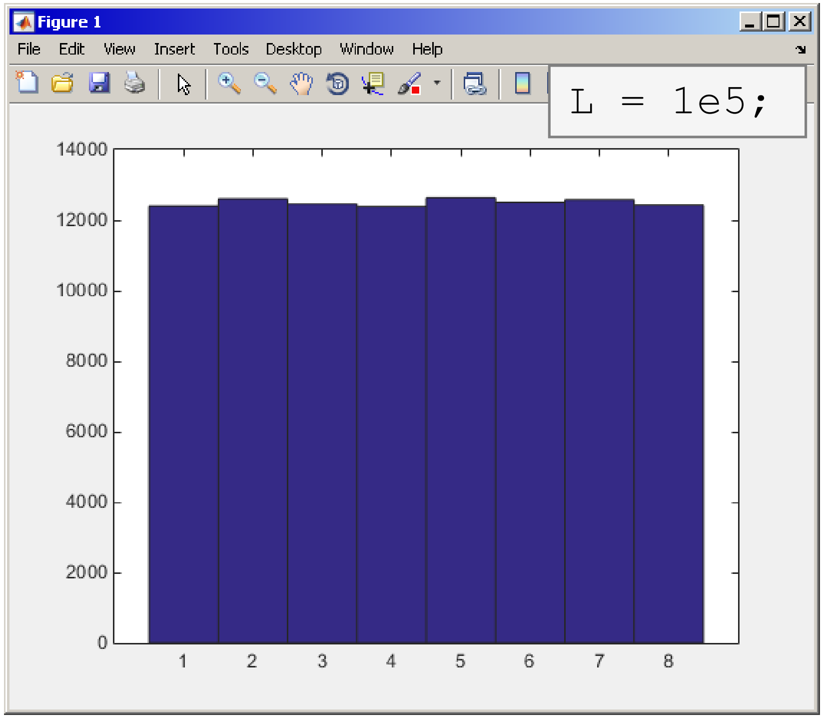
`figure, hold`	open new figure, enable multiple curves in one axis	●
`title, xlim, ..., xlabel, ...`	heading, axes limits, axes labels	●
`legend, grid`	legend, grid	●
`gtext, ginput`	interactive text insertion, interactive input from mouse or cursor	

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()`

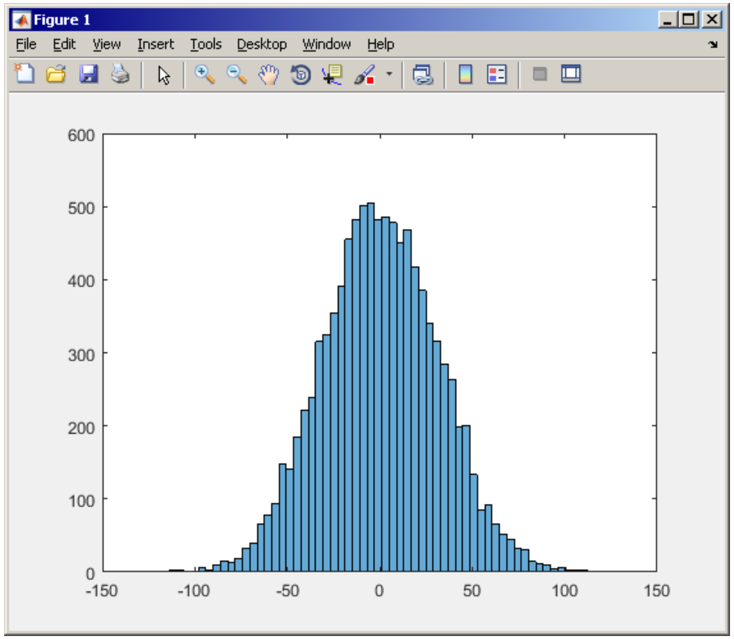
Exercise #3

```
...
...
...
...
...
...
...
```

- mean and standard deviation of `nOnesOverAverage`:

$$\mu = \frac{1}{N}\sum_i x_i \approx 0$$

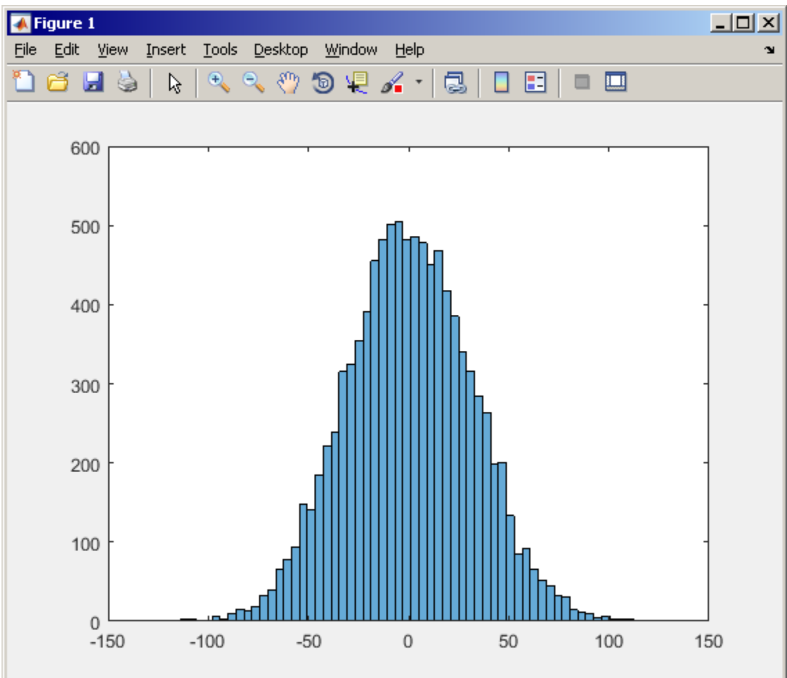
$$\sigma = \sqrt{\frac{\sum_i (\mu - x_i)^2}{N}} = \sqrt{1000} \approx 31.62$$



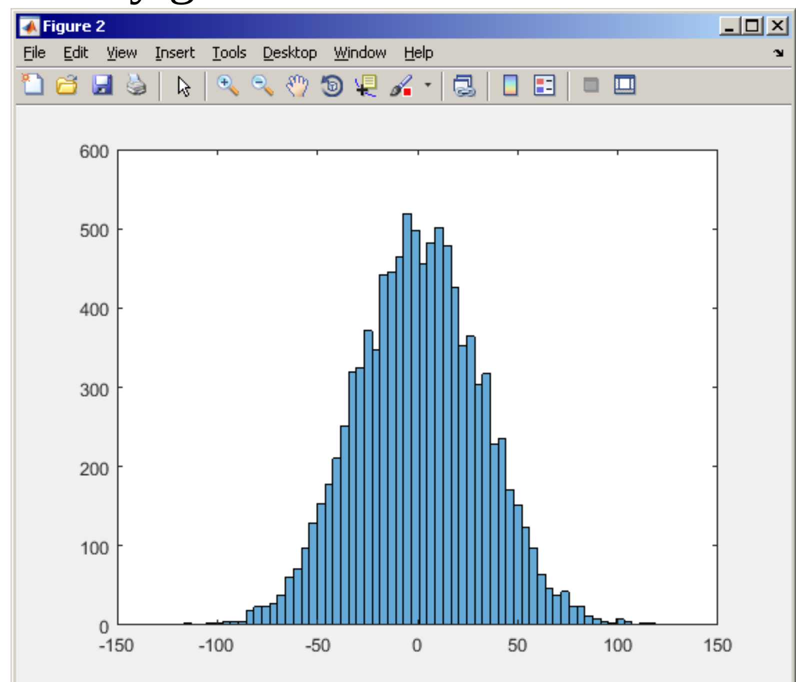
Exercise #4

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

coin toss:



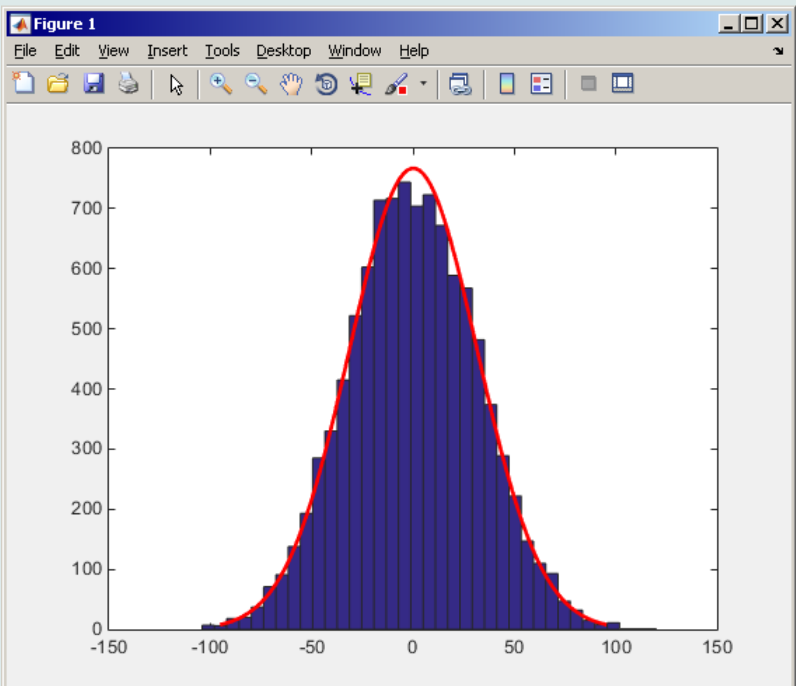
directly generated data:



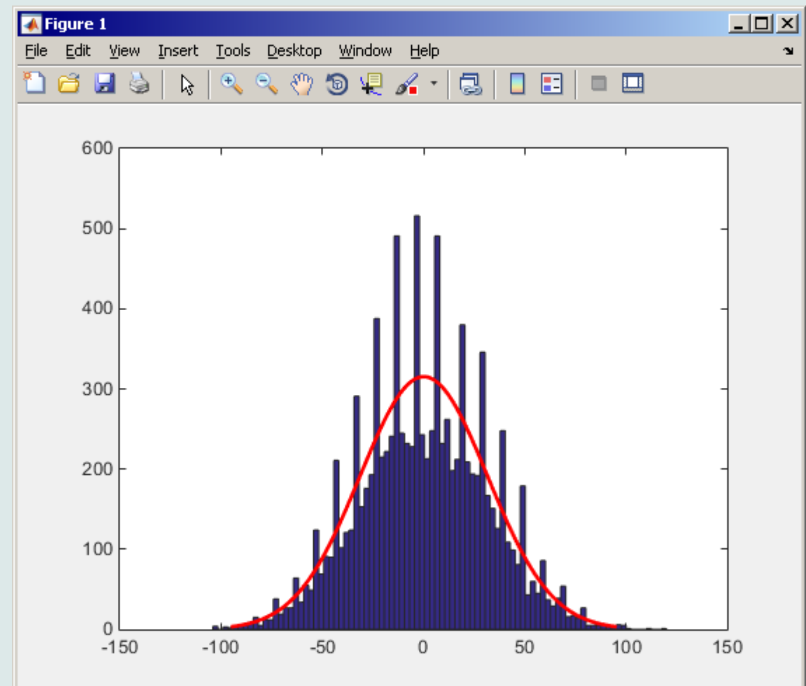
Exercise #5

- 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

```
histfit(nOnesOverAverage, 37);
```



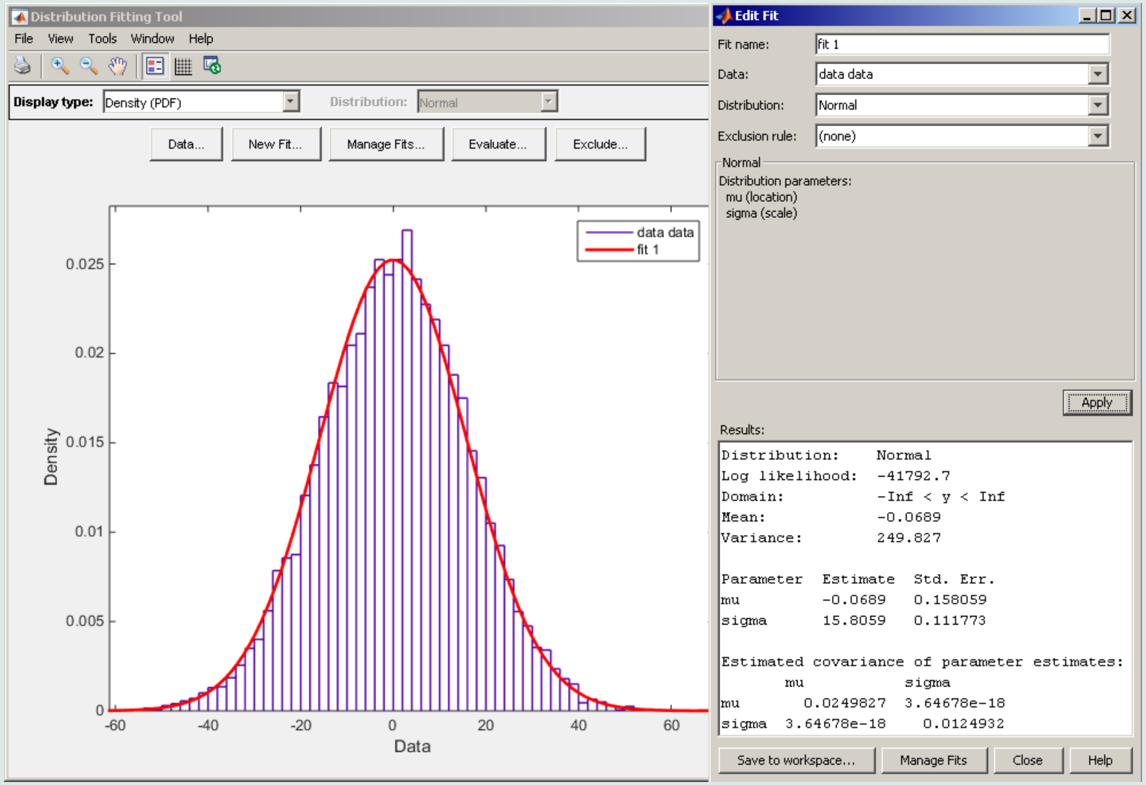

```
histfit(nOnesOverAverage, 90);
```



Exercise #6

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

```
dfittool(nOnesOverAverage);
```



Exercise #7

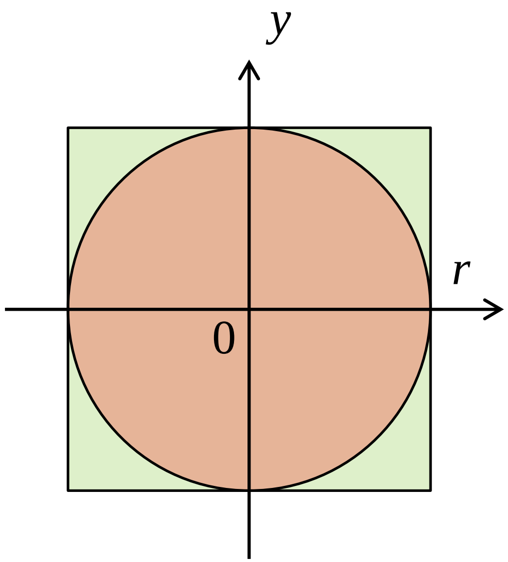
600 s

- 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

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



- 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

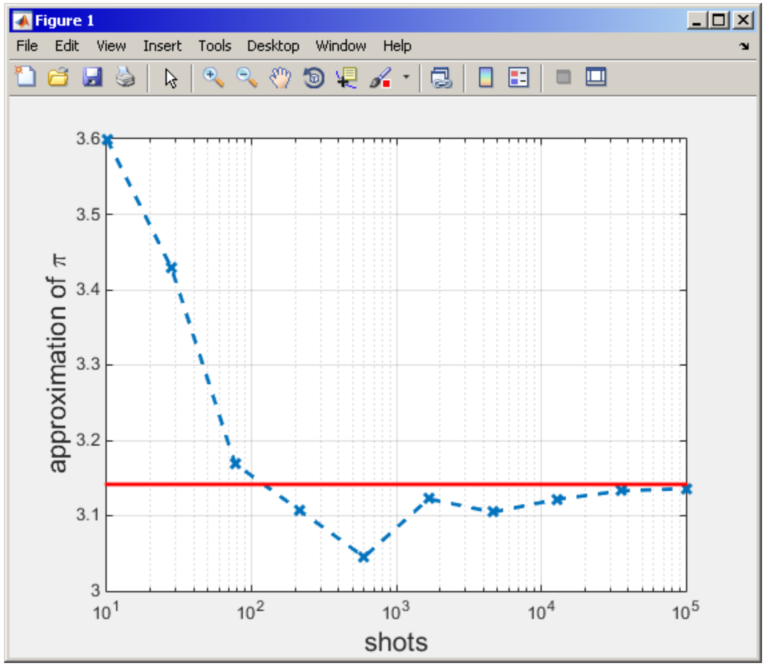
Exercise #7- solution

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

```
clear; close all; clc;
...
...
...
...
...
...
...
...
...
...
```

Exercise #8

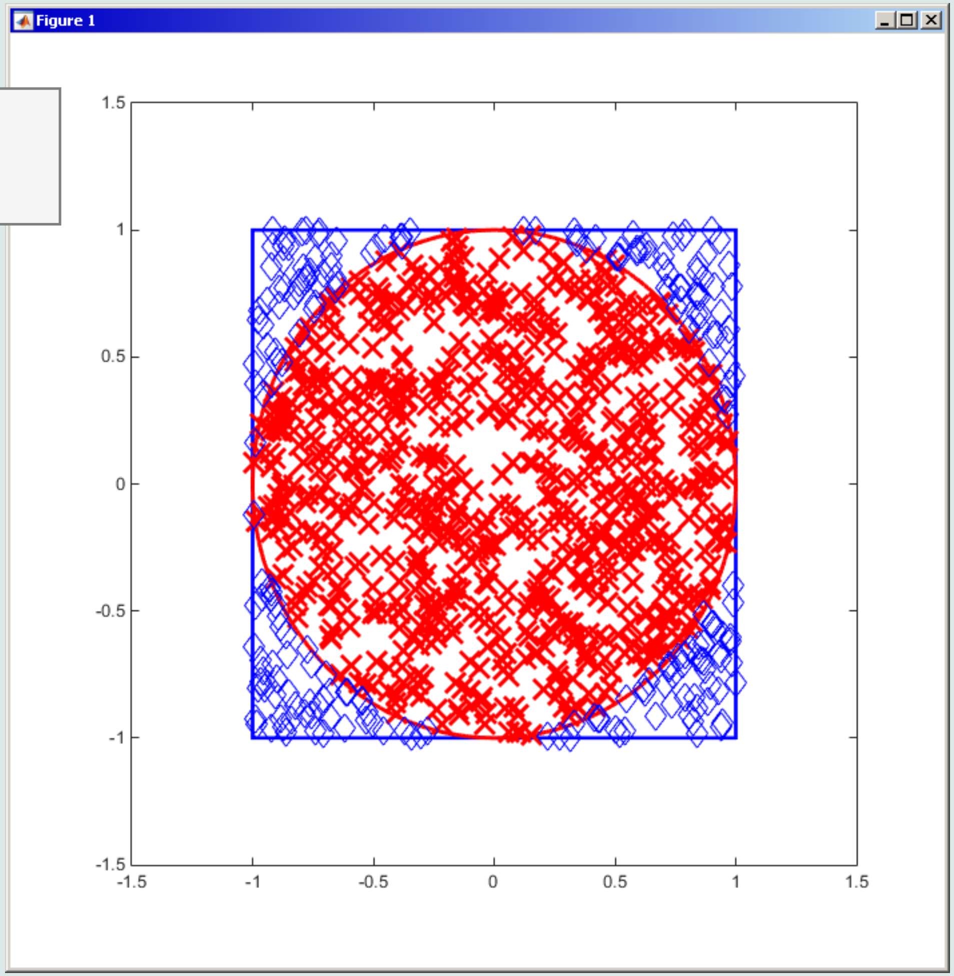
- approximation of Ludolph's number - visualization:



Exercise #9

- visualization of the task:

```
display      = 1000;
Rdisplay     = R(1:display,1);
shotsdisplay = shots(1:display,1:2);
```



Exercise #10

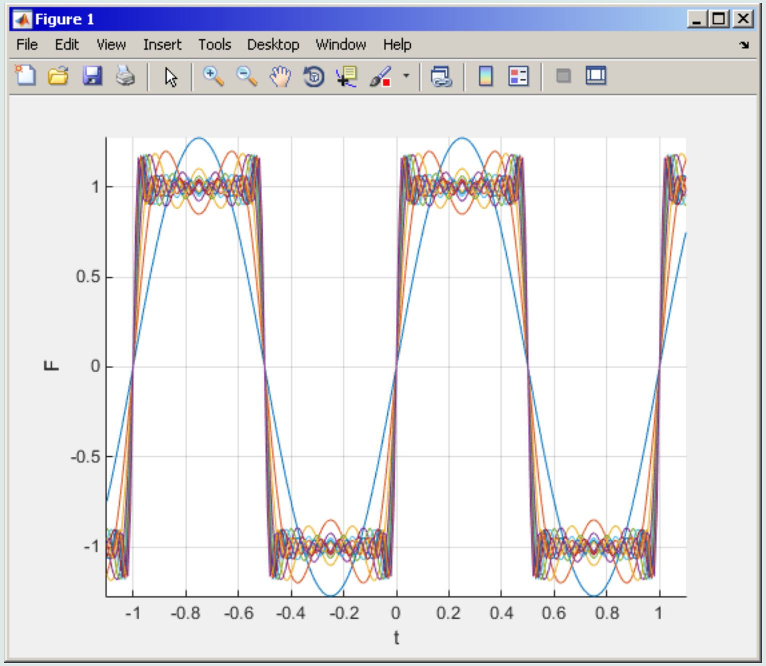
600 s

- 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

```
close all; clear; clc;
...
...
...
...
...
...
...
...
...
...
...
...
```



Thank you!

ver. 9.1 (26/03/2017)
Miloslav Čapek, Pavel Valtr
miloslav.capek@fel.cvut.cz

Document created as part of A0B17MTB course.