Lecture 7: Visualization

B0B17MTB, BE0B17MTB – MATLAB

Miloslav Čapek, Viktor Adler, *et al.*

Department of Electromagnetic Field
Czech Technical University in Prague
Czech Republic
matlab@fel.cvut.cz

April 11, 2024
Summer semester 2023/24

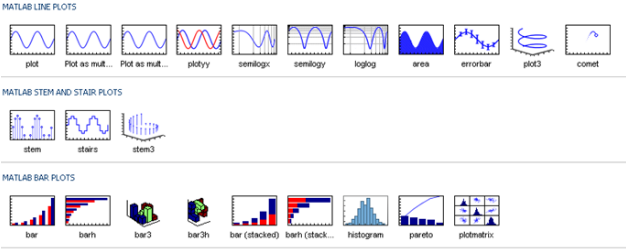
Outline

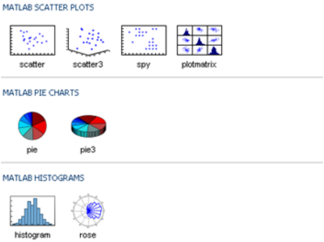
1. Visualizing in MATLAB
2. Object Handles
3. Excercises

Introduction to Visualizing

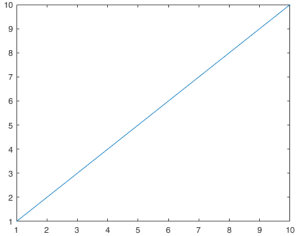
- We have already got acquainted (marginally) with some of MATLAB graphs.
  - `plot`, `stem`, `semilogx`, `pcolor`
- In general, graphical functions in MATLAB can be used as:
  - higher level
    - Access to individual functions, object properties are adjusted by input parameters of the function.
    - The first seven 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

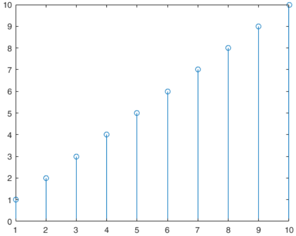
Selected Graphs I.





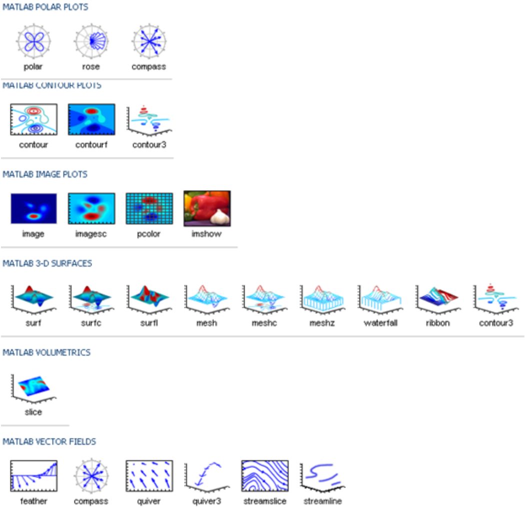

```
plot(linspace(1,10,10));
stem(linspace(1,10,10));
% ... and others
```



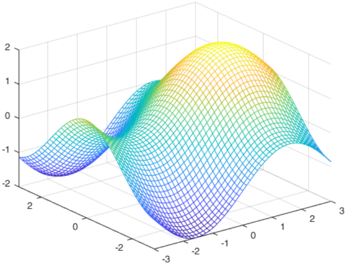


Visualization

Selected Graphs II.




```
x = -3:0.125:3;
y = x.';
z = sin(x) + cos(y);
mesh(x,y,z);
axis([-3 3 -3 3 -2 2]);
```

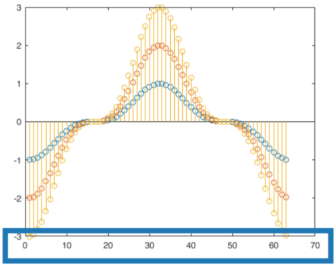


Function `figure`

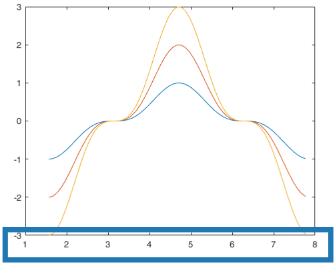
- `figure` opens empty figure to plot graphs.
  - The function returns object of class `matlab.ui.Figure`.
  - It is possible to plot matrix data (column-wise).
  - Don't forget about x-axis data!

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

```
figure;
stem(fx.');
```




```
figure;
plot(x, fx);
```



Visualization

LineSpec – Customizing Graph Curves I.

- What do plot function parameters mean?
  - See >> doc plot.
  - 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,
    - line style.

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 plot

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

line style	
'-'	solid
'--'	dashed
':'	dot
'-.'	dash-dot
''	none

Visualization

Selected Functions for Graph Modification

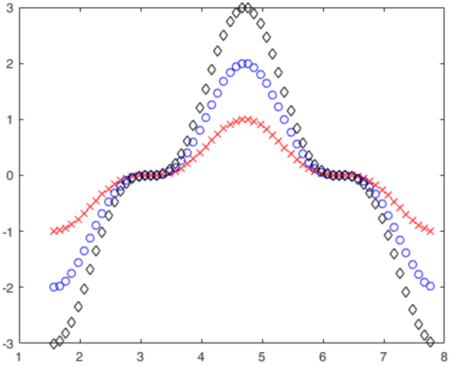
- Graphs can be customized in many ways, the basic ones are:

function	description
`title`	title of the graph
`xlabel, ylabel, zlabel`	label axes
`x-`, `y-`, `ztickformat`	specify axis tick label format
`grid on`, `grid off`	turns grid on / off
`hold on`	enables to add another graphical elements while keeping the existing ones
`xlim, ylim, zlim`	set axes' range
`legend`	display legend
`tiledlayout, nexttile`	create more axes in one figure
`yyaxis`	create chart with two y-axes
`box on`	display axes outline
`text`	adds text to graph
and others	

Function hold on

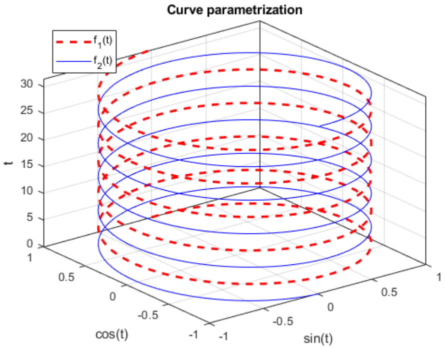
- Function hold on enables to plot multiple curves in one axis.
- It is possible to disable this feature by typing hold off.
- Advanced: function hold change property NextPlot of axes object to 'add' or 'replace'.

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



Visualizing – plot3

- The example below shows plotting a spiral and customizing plotting parameters.
- It is possible to use additional name-value pair arguments with majority of plotting functions.

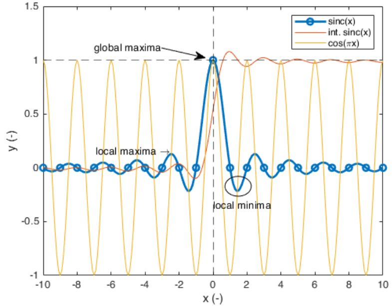



```
figure('color', 'w');
t = 0:0.05:10*pi;
plot3(sin(t), cos(t), t, 'r--', ...
   'LineWidth', 2);
hold on;
plot3(-sin(t), -cos(t), t, 'b')
box on;
grid on;
xlabel('sin(t)');
ylabel('cos(t)');
zlabel('t');
title('Curve parametrization');
legend('f_1(t)', 'f_2(t)', ...
   'Location', 'northwest');
```

Visualization

Visualizing – annotation, text

- annotation creates object into a graph with shape of line, arrows, rectangle and ellipse.
  - Shape position is defined in normalized coordinate system of the figure.
- text creates text labels into a graph possibly using 'latex' interpretter.
  - Text position is defined in coordinate system of a drawing area (axes).
- legend omits items with empty label ''.
- Property MarkerIndices of line defines positions of markers on it.



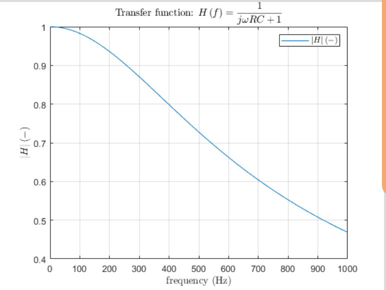

```
dx = 0.1;
x = -10:dx:10;
sFcn = sin(pi*x)./(pi*x); % normalized sinc function
sFcn(x == 0) = 1; % definition at x=0

figure
plot(x, sFcn, 'Marker', 'o', 'LineWidth', 2, ...
   'MarkerIndices', 1:1/dx:length(x)) % find(sFcn == 0)
xline(0, '--'); yline(1, '--'); % lines with constant x any y values
hold on
plot(x, cumsum(sFcn)*dx) % cummulative sum (integral)
plot(x, cos(pi*x)) % intersection with sinc indicates extrema
legend('sinc(x)', '', '', 'int. sinc(x)', 'cos(\pix)')
annotation('textarrow', [0.4, 0.5], [0.8, 0.77], 'String', 'global maxima')
annotation('ellipse', [0.545 0.35 0.06 0.06]) % [x y w h]
text(0, -0.35, 'local minima')
text(-2.5, 0.15, 'local maxima \rightarrow', 'HorizontalAlignment', 'right')
xlabel('x (-)')
ylabel('y (-)')
```

Visualization

LaTeX in Figures

- Labels and titles in figure have `Interpreter` property.
- Possible values are `'tex'`, `'latex'` and `'none'`.
- Font is default LaTeX font.

```
figure;
f = 1:1e3; R = 100; C = 3e-6;
Hf = abs(1./(1j*2*pi*f*R*C + 1));
plot(f, Hf);
grid on;
xlabel('frequency (Hz)', 'Interpreter', 'latex');
ylabel('$$\left| H \right|\left( - \right)$$', ...
  'Interpreter', 'latex');
title(['Transfer function: $$H\left( f \right)', ...
  ' = \frac{1}{{j\omega RC + 1}}$$'], ...
  'Interpreter', 'latex');
hL=legend('$$\left| H \right|\left( - \right)$$');
hL.Interpreter = 'latex';
```



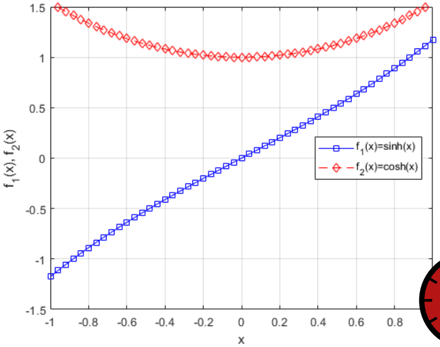
Visualization

LineSpec – Customizing Graph Curves II.a

- Evaluate following two functions in the interval $x \in [-1, 1]$ for 51 values:

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

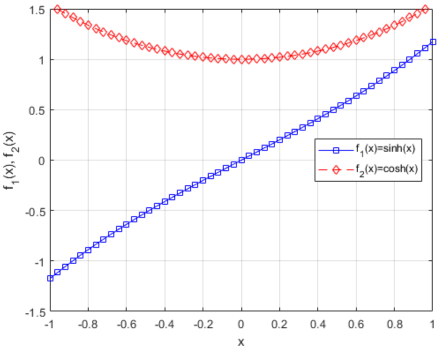
- Use the function `plot` to depict both f_1 and f_2 so that:
  - both functions are plotted in the same axis,
  - the first function is plotted in blue with □ marker as solid line,
  - the other function is plotted in red with ◇ marker and dashed line,
  - limit the interval of the y-axis to $[-1.5, 1.5]$,
  - add a legend associated to both functions,
  - label the axes (x-axis: x, y-axis: $\mathrm{f}_1(\mathrm{x})$, $\mathrm{f}_2(\mathrm{x})$),
  - apply grid to the graph.



450

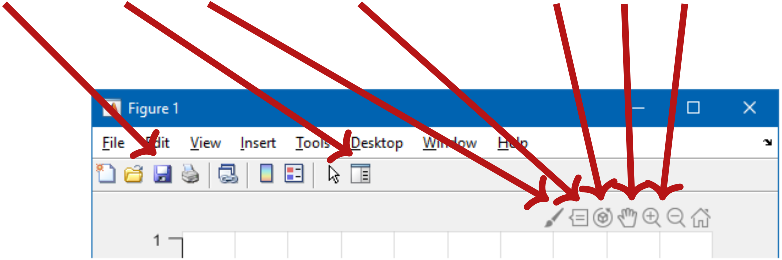
LineSpec – Customizing Graph Curves II.b

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



Visualizing – Plot Tools

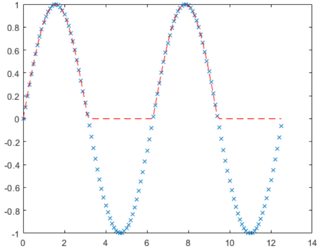
- It is possible to keep on editing the graph by other means.
- All operations can be carried out using MATLAB functions.
  - `saveas`, `inspect`, `brush`, `datacursormode`, `rotate3d`, `pan`, `zoom`



- Properties of all graphical objects can be set programmatically (see later).
  - Preferred for good-looking graphs with lot of graphical features.

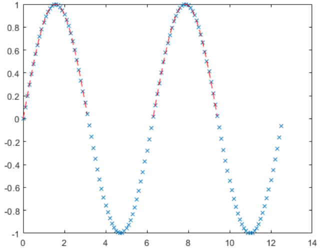
Visualizing — Use of NaN Values

- 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;
% ...
```

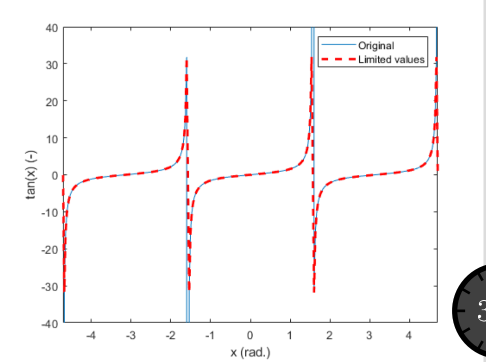


Visualization

Rounding

- Plot function $\tan(x)$ for $x \in [-3/2\pi, 3/2\pi]$ with step $pi/2$.
- 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) (-)');
```

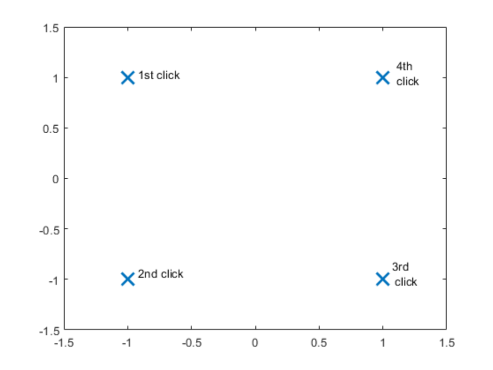


300

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'});
```



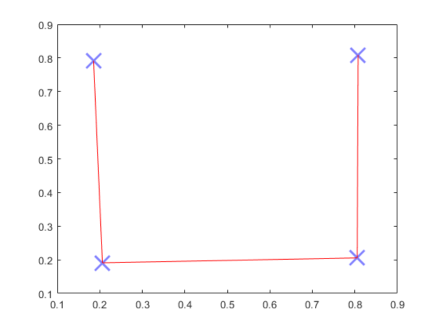
Visualization

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);
```

```
plot(P(:, 1), P(:, 2), ...
   'LineStyle', 'none', ...
   'LineWidth', 2, ...
   'Color', [0.5 0.5 1], ...
   'Marker', 'x', ...
   'MarkerSize', 20);
hold on;
plot(P(:, 1), P(:, 2), 'r');
```

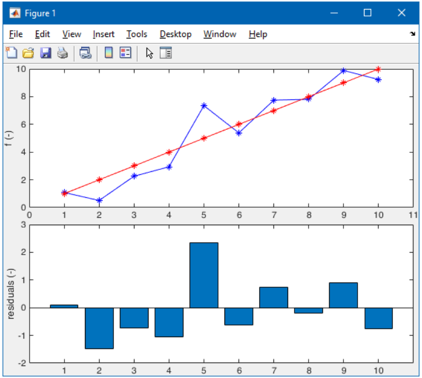


Visualization

More Graphs in a Figure I. – tiledlayout, nexttile

- tiledlayout creates invisible grid for advanced axes placement.
  - Properties TileSpacing and Padding set grid spacing and edges.
  - Property TileIndexing set indexing scheme as 'rowmajor' or 'columnmajor'.
  - tiledlayout('flow') - layout reflows as needed to accommodate the new axes.
  - nexttile(p, [r, c]) - place axes at position p spanning r × c tiles.

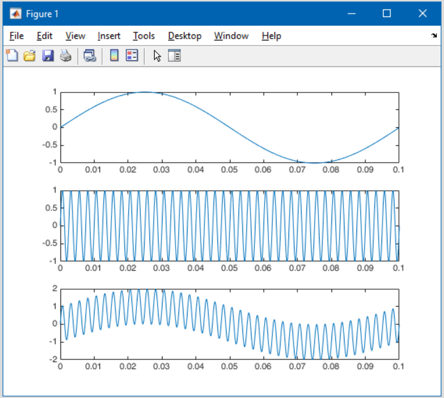
```
x = 1:10;
f = x + randn(size(x));

figure;
tiledlayout(2, 1, ... grid 2x1
  'TileSpacing', 'tight', ...
  'Padding', 'tight');
nexttile();
plot(x, f, '*-b', x, x, '*-r');
xlim([0 11]);
ylabel('f (-)');
nexttile();
bar(x, f - x); xlim([0 11]);
ylabel('residuals (-)');
```



More Graphs in a Figure II. – subplot

- Inserting several different graphs in a single window figure.
  - Function subplot(m, n, p):
  - m is number of rows,
  - n is number of columns,
  - p is position.

```
t = linspace(0, 0.1, 0.1*10e3);
f1 = 10; f2  = 400;

y1 = sin(2*pi*f1*t);
y2 = sin(2*pi*f2*t);
y3 = y1 + y2;

figure('color', 'w');
subplot(3, 1, 1); plot(t, y1);
subplot(3, 1, 2); plot(t, y2);
subplot(3, 1, 3); plot(t, y3);
```

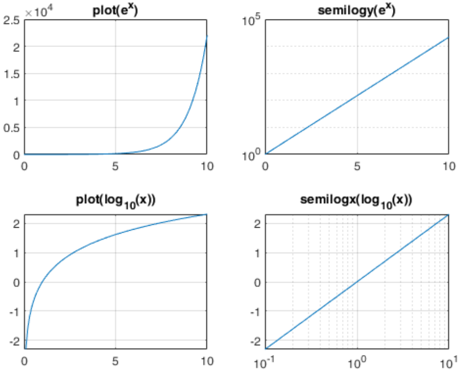


Visualization

Logarithmic Scale

- Functions semilogy, semilogx, loglog.

```
x  = 0:0.1:10;
y1 = exp(x);
y2 = log(x);

figure('color', 'w');
tiledlayout(2, 2);
nexttile(); plot(x, y1);
title('plot(e^x)'); grid on;

nexttile(); semilogy(x, y1);
title('semilogy(e^x)'); grid on;

nexttile(); plot(x, y2);
title('plot(log_{10}(x))'); grid on;

nexttile(); semilogx(x, y2);
title('semilogx(log_{10}(x))'); grid on;
```

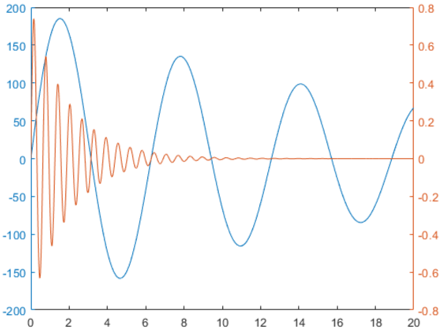


Visualization

Double y Axis — yyaxis I.

- Enable to draw more curves to a single graph with two y axis with different ranges.

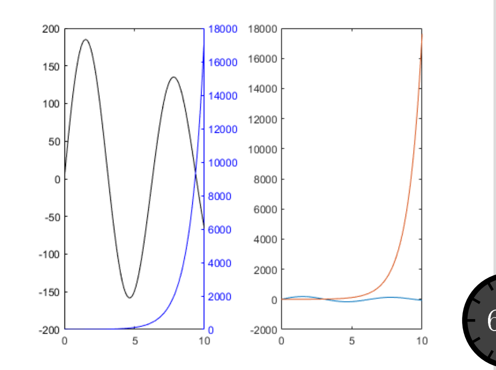
```
x  = 0:0.01:20;
y1 = 200 * exp(-0.05*x) .* sin(x);
y2 = 0.8 * exp(-0.5*x) .* sin(10*x);

figure('color', 'w');
yyaxis left; plot(x, y1);
yyaxis right; plot(x, y2);
```



Double y Axis — yyaxis II.

- Compare plot and `yyaxis` in one figure object (using `subplot`) for functions shown below.
  - In the object created by `yyaxis` change default colors of individual lines to blue and black (don't forget about the axes).

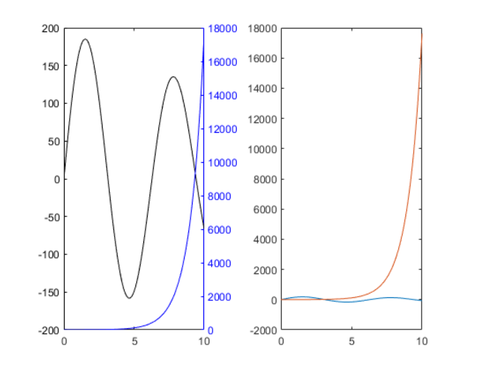
```
x  = 0:0.1:10;
y1 = 200 * exp(-0.05*x) .* sin(x);
y2 = 0.8 * exp(x);
```



600

Double y Axis — yyaxis II.

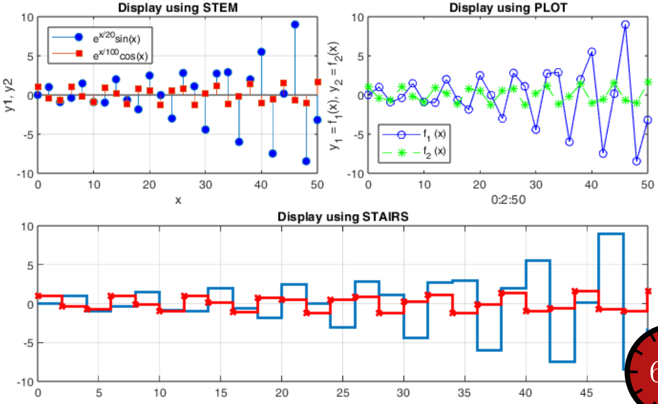
```
hAx = subplot(1, 2, 1);
yyaxis left;
lin1Obj = plot(x, y1);
yyaxis right;
lin2Obj = plot(x, y2);

lin1Obj.Color = 'k';
lin2Obj.Color = 'b';

hAx.YAxis(1).Color  = 'k';
hAx.YAxis(2).Color  = 'b';

subplot(1, 2, 2);
plot(x, y1, x, y2);
```



Functions `stem`, `stairs`

- Try to imitate the figure where functions `y1` and `y2` are defined below.
  - See documentation of `stem` and `stairs` function.
  - Hints: property `MarkerFaceColor` of `line`, upper index: `'e^{x/100}'`.

```
x  = 0:2:50;
y1 = exp(0.05*x).*sin(x);
y2 = exp(0.01*x).*cos(x);
```

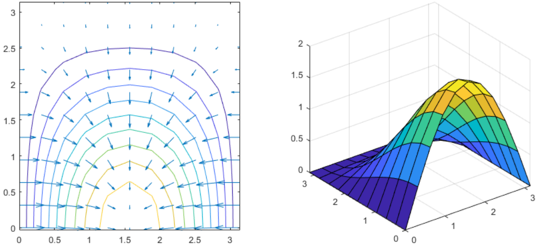


stem, stairs

Plotting 2-D Functions

- contour, quiver, surf

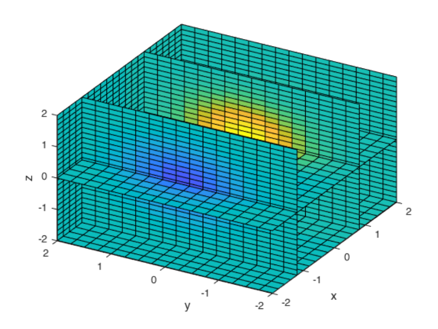
```
x = 0:pi/10:pi;
y = x.';
z = sin(x) + cos(y).*sin(x);
[gx, gy] = gradient(z);

figure('Color', 'w');
tiledlayout(1, 2);
nexttile();
hold on;
contour(x, y, z);
quiver(x, y, gx, gy);

nexttile();
surf(x, y, z);
```



Volumetric Visualizing

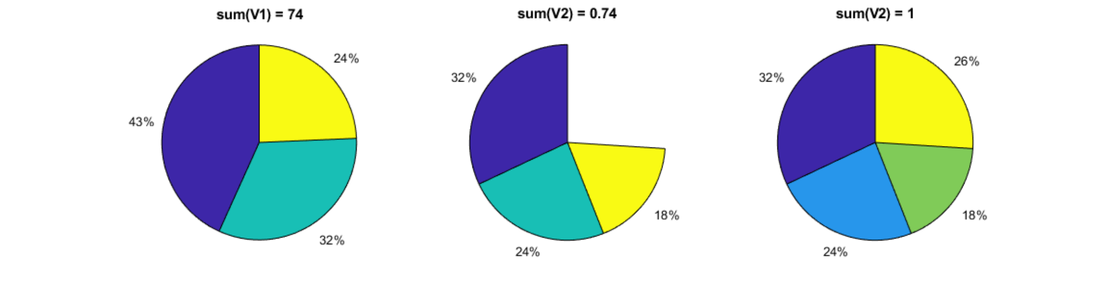
- Function `slice`.
  - Draw slices for the volumetric data.

```
x = -2:0.2:2;
y = (-2:0.25:2).';
z = shiftdim(-2:0.16:2, -1);

v = x.*exp(-x.^2 - y.^2 - z.^2);

xSlice = [-1.2, 0.8, 2];
ySlice = 2;
zSlice = [-2, 0];

figure('Color', 'w');
slice(x, y, z, v, xSlice, ySlice, zSlice);
xlabel('x'); ylabel('y'); zlabel('z');
% view(azimuth, elevation)
view(-60, 40);
```



Functions pie, pie3

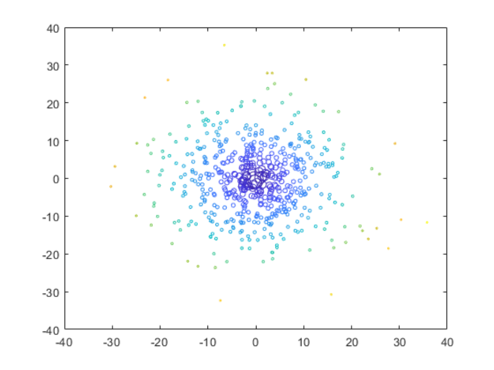
```
V1 = [32 24 18]; % sum(V1) = 74
V2 = V1/100; % sum(V2) = 0.74
V3 = [V2 1-sum(V2)]; % sum(V3) = 1

figure('Color', 'w');
subplot(1, 3, 1); pie(V1); title('sum(V1) = 74');
subplot(1, 3, 2); pie(V2); title('sum(V2) = 0.74');
subplot(1, 3, 3); pie(V3); title('sum(V2) = 1');
```



Function scatter

- Scatter function enables effective (fast) ploting of huge number of points.
  - Color and size can be set to all individual points.

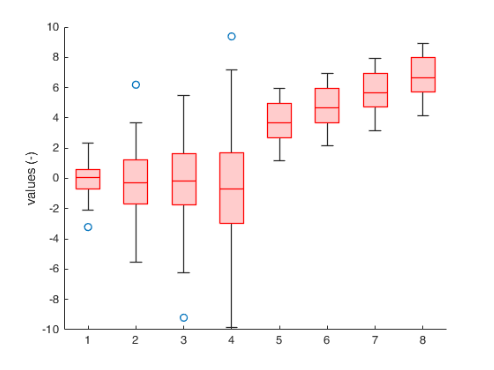
```
x = 10*randn(500, 1);
y = 10*randn(500, 1);
c = hypot(x, y);

figure('color', 'w');
scatter(x, y, 100./c, c);
box on;
```



Box Plot – boxchart

- Box plot shows basic statistical properties of random data.
  - Median, lower and upper quartiles, outliers and minimal/maximal values (outside outliers).

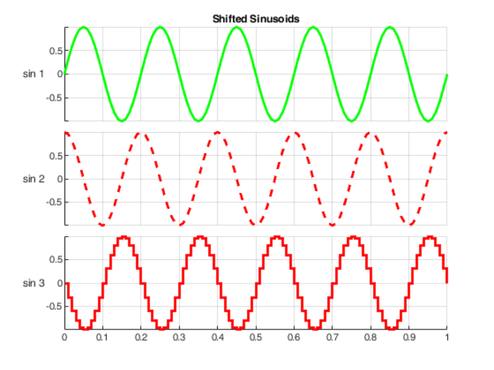
```
nSamples = 1e2;
data = [randn(nSamples, 4).*(1:4), ...
  5*rand(nSamples, 1) + (1:4)];

figure
boxchart(data, 'BoxFaceColor', 'r')
ylabel('values (-)')
```



Stacked Plot Sharing x-axis – stackedplot

- Stacked plot enables to plot columns of a numeric matrix in separate graphs sharing a single x-axis.
  - Reference of the stacked plot enables to set style of individual lines.

```
t = linspace(0, 1, 101).';
phaseShift = 0:pi/2:pi;
signals = sin(2*pi*5*t + phaseShift);

figure
hSP = stackedplot(t, signals, 'r', ...
   'LineWidth', 2, 'DisplayLabels', ...
   {'sin 1', 'sin 2', 'sin 3'});
grid on;
title('Shifted Sinusoids');
xlabel('time (s)');

%% set individual lines
hSP.LineProperties(1).Color = 'g';
hSP.LineProperties(2).LineStyle = '--';
hSP.LineProperties(3).PlotType = 'stairs';
```



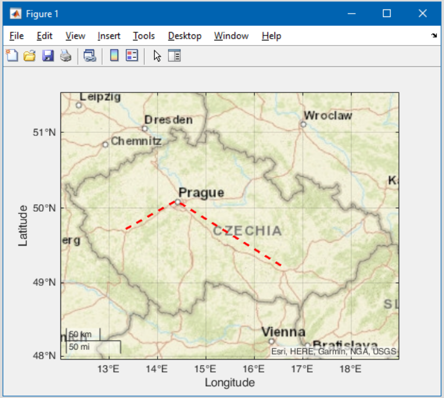
Visualization

Geographic Plots

- `geoplot` visualize latitude and longitude data over interactive map.
- Resulting axes (`geoaxes`) enables panning and zooming.
- Type of the map can be switched by `geobasemap`.

```
ZCU = [49.7237817, 13.3496361];
CVUT = [50.1026947, 14.3929308];
VUT = [49.2274472, 16.5742747];

figure;
geoplot([ZCU(1), CVUT(1), VUT(1)], ...
   [ZCU(2), CVUT(2), VUT(2)], 'r--', ...
   'LineWidth', 2);
geolimits([48.5 51], [12 19]);
geobasemap colorterrain; % streets, ...
```

```
% get coordinates from the map
n = 2;
[lat, lon] = ginput(n)
```



Visualization

Picture Depiction

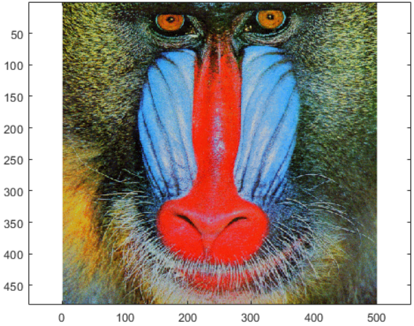
- Function `image`, `imagesc`, `colormap`.

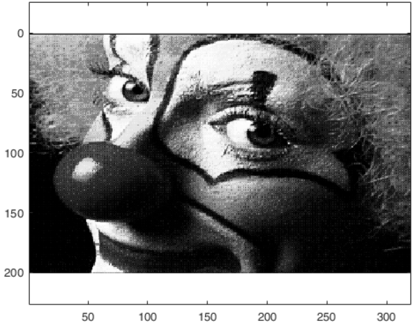
```
load mandrill
image(X)
axis equal
```

```
colormap(map)
```

```
load clown
imagesc(X)
axis equal
```

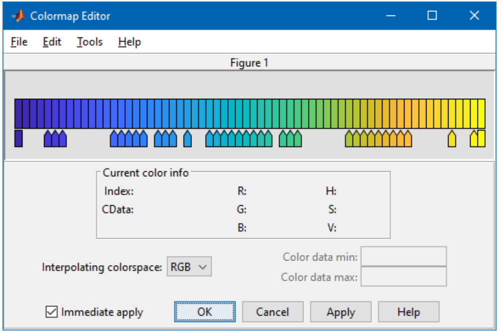
```
colormap(gray)
```





Function colormap I.

- Determines the scale used in picture color mapping.
- It is possible to create/apply an own one: `colormapeditor`.

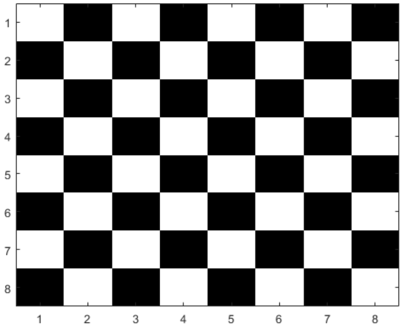


Colormap Name	Color Scale
parula	
jet	
hsv	
hot	
cool	
spring	
summer	
autumn	
winter	
gray	
bone	
copper	
pink	
lines	
colorcube	
prism	
flag	
white	

Visualization

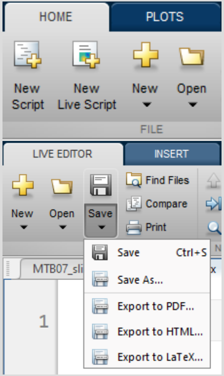
Function `colormap` II.

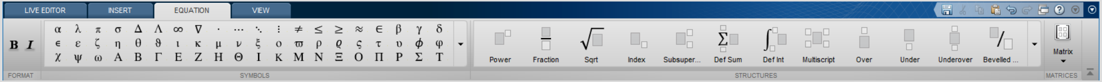
- Create a chessboard as shown in the figure.
  - The picture can be drawn using the function `imagesc`.
  - Consider `colormap` setting.



Live Script I.

- Live script can contain code, generated output, formatted text, images, hyperlinks, equations, ...
  - It is necessary to use Live Editor.
  - From MATLAB window: HOME → New Live Script.
  - From editor: EDITOR → New → Live Script
  - Editor creates `*.mlx` files.
- Export options: PDF, HTML, LaTeX.
- Internal extensive equation editor.





Live Script II.

loanRepayment.mlx *

Loan Repayment Live Script

Compound interest is the addition of interest to the principal sum of a loan or deposit.

Initialization of script

```
clear; clc; close all
r = 0.1:0.01:0.2;
A = 1e3;
n = 12;
k = (1:15).';
```

Computation

$$P = \frac{rA\left(1+\frac{r}{n}\right)^{nk}}{n\left(\left(1+\frac{r}{n}\right)^{nk}-1\right)}$$

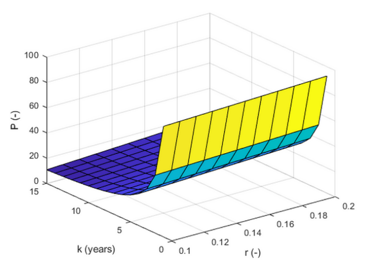
```
P = r*A.*(1 + r/n).^(n*k) ./ ...
      (n.*((1 + r/n).^(n*k) - 1));
```

Plot Results

```
surf(r, k, P)
xlabel('r (-)');
ylabel('k (years)');
zlabel('P (-)');
```

For more information:

https://en.wikipedia.org/wiki/Compound_interest



Visualization

Object Handles I.

- Each individual graphical object has its own pointer ('handle' in Matlab terms).
- These handles are practically a reference to an existing object.
- Handle is always created by MATLAB, it is up to the user to store it.
- One handle can be saved to several variables but they refer to a single object.
- All graphical objects inherit superclass `handle`.
  - Inherits several useful methods (`set`, `get`, `delete`, `isvalid`, ...).

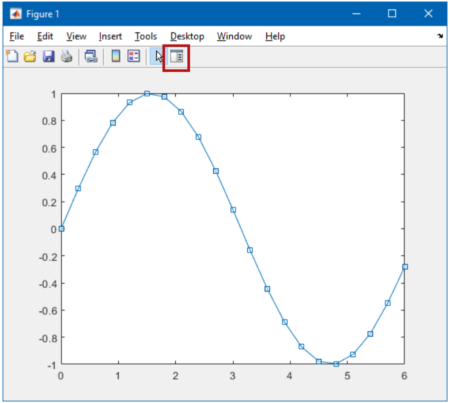
```
hFig = figure;
hAx = axes('Parent', hFig);
hLine1 = line('Parent', hAx);
```

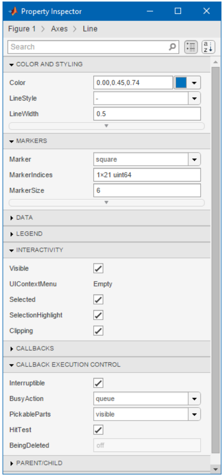
- Graphical objects respect specific hierarchy.
- See help for list of properties (`>> doc Figure Properties`, `>> doc Axes Properties`, `>> doc Line Properties`, ...).

Visualization

Object Handles II.

- Property inspector (`inspect`).





Visualization

Object Handles III.

- The way of setting handle object properties.
  - Using functions `set` and `get`.
    - It is not case sensitive.

```
myLineObj = plot(1:10);
get(myLineObj, 'color')
```

```
set(myLineObj, 'color', 'r')
```

  - Dot notation.
    - It is cAsE sEnSiTiVe.

```
myLineObj = plot(1:10);
myLineObj.Color
```

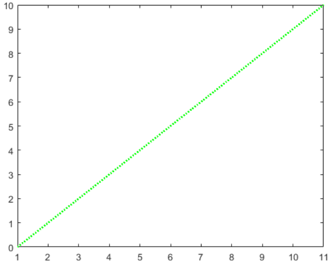
```
myLineObj.Color = 'r';
myLineObj.Color
```

Functions get and set

- Create a graphic object in the way shown. Then using functions `get` and `set` perform following tasks.

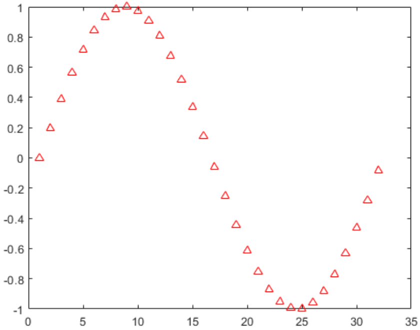
```
myLineObj = plot(0:10);
```

  - Find out the thickness of the line and increase it by 1.5.
  - Set the line color to green.
  - Set the line style to dotted.



Dot Notation Application

- Using dot notation change the initial setting of the function shown to get plot as in the figure.

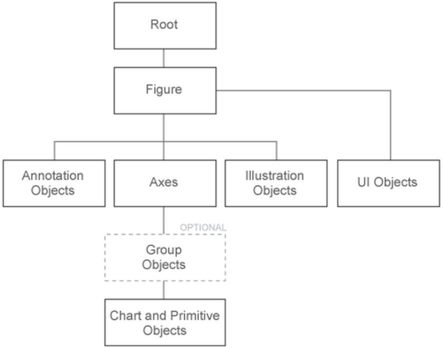



```
myLineObj = plot(sin(0:0.2:2*pi));
```

60

Graphics Object Hierarchy

- All graphical objects are connected in the hierarchy via `Children` and `Parent` properties.
  - If the `Children` property is a vector, do not index this vector for obtaining a reference to a single object! Order of objects changes between MATLAB versions.




```
hRoot = groot;
hFig = figure('Parent', hRoot);
hAx = axes('Parent', hFig);
hLine = line('Parent', hAx, ...
  'XData', -10:10, ...
  'YData', (-10:10).^3);
hTitle = title(hAx, 'Cubic fcn.');
```

```
hRoot.Children % ans = hFig
hFig.Children % ans = hAx
hAx.Children % ans = hLine
hLine.Children
% ans = 0x0 GraphicsPlaceholder
hTitle.Parent % ans = hAx
```

```
hRoot.Children.Children.Color = 'y';
```

Visualization

Fast Graphics Update

- Graphics are updated with a lower priority than other calculations.
- `drawnow` force to immediate update the graphics.
- High-level functions (`plot`, `surf`, `image`, ...) are slow.
- Set object's properties (`XData`, `YData`, `CData`, ...) is the fastest option.

```
f0 = 1e9;
x = linspace(0, 1, 201);
y = x.';
c0 = 3e8;
lambda = c0/f0;
t = linspace(0, 1/f0);
k = 2*pi/lambda;
R = sqrt(x.^2 + y.^2);
```

```
figure; drawnow();
tic;
for thisTime = t
  E = exp(1j*(-k*R + 2*pi*f0*thisTime));
  surf(x, y, real(E), 'LineStyle', 'none');
%   drawnow limitrate % skip some frames
%   drawnow() % immediately update graphics
%   pause(0.001) % like drawnow
end
tEnd = toc;
fprintf('Reached FPS: %.2f Hz.\n', length(t)/tEnd);
```

```
hFig = figure;
hAx = axes('Parent', hFig);
hSurf = surf('Parent', hAx, 'LineStyle', 'none', ...
  'XData', x, 'YData', y, 'ZData', nan(size(R)));
drawnow();
tic;
for thisTime = t
  E = exp(1j*(-k*R + 2*pi*f0*thisTime));
  hSurf.ZData = real(E);
  drawnow();
end
tEnd = toc;
fprintf('Reached FPS: %.2f Hz.\n', length(t)/tEnd);
```

LineSpec — Default Setting

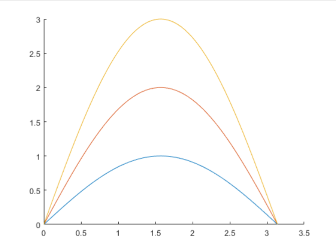
- Colors in given order are used when plotting more lines in one axis.
- It is not necessary to set color of each curve separately when using `hold on`, nor plotting matrix columns.

```
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));
```

```
figure, plot(x, 1:3*sin(x));
```

```
set(groot, 'defaultAxesColorOrder', ...
  myColors)
```

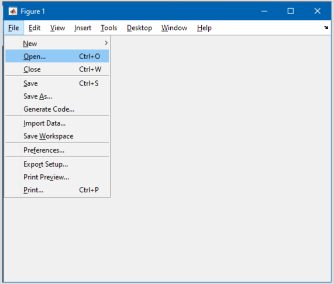
```
>> get(groot, 'DefaultAxesColorOrder')
% ans =
%         0    0.4470    0.7410
%    0.8500    0.3250    0.0980
%    0.9290    0.6940    0.1250
%    0.4940    0.1840    0.5560
%    0.4660    0.6740    0.1880
%    0.3010    0.7450    0.9330
%    0.6350    0.0780    0.1840
```

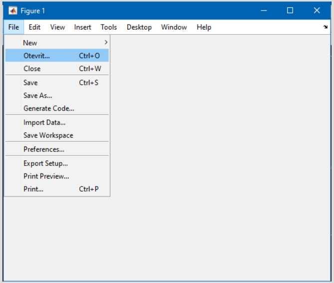


What Is Handle Good For?

- When having a handle, one can entirely control given object.
- The example below returns all identifiers existing in window `figure`.
- In this way we can, for instance, change item 'Open. . . ' to 'Otevrit. . . '.
    - Or anything else (*e.g.* callback of file opening to callback of window closing :)).

```
hFig = figure('Toolbar', 'none');
allFigHndl = guihandles(hFig);
set(allFigHndl.figMenuOpen, 'Label', 'Otevrit...');
```





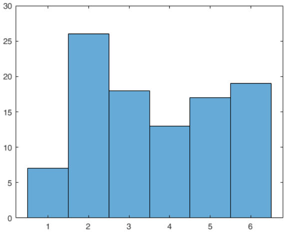
Exercises

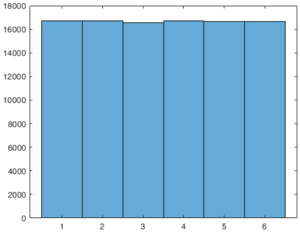
Exercise I.

- 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).

```
L = 1e2;
```

```
L = 1e5;
```

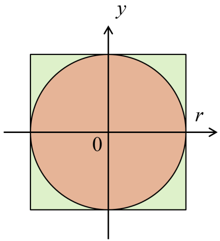




Exercise II.a

- Use Monte Carlo method to estimate the value of π.
  - Monte Carlo is a stochastic method using pseudo-random 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.

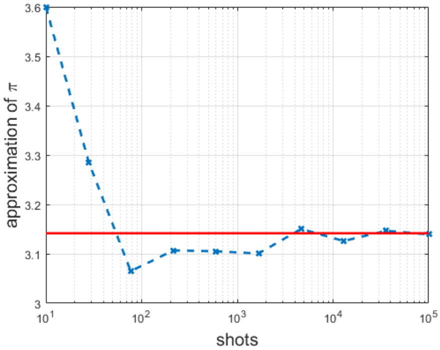
600

Exercise II.b

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

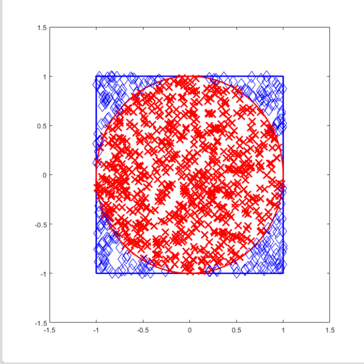
Exercise II.c

- Approximation of Ludolph's number - visualization:



Exercise II.d

- Visualization of the task:




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

figure('color', 'w', 'pos',[50 50 700 700], ...
    'Menubar', 'none');
line([-1 1 1 -1 -1], [-1 -1 1 1 -1], ...
   'LineWidth', 2, 'Color', 'b');
hold on;
xlim([-1.5 1.5]); ylim([-1.5 1.5]); box on;
plot(cos(0:0.001:2*pi), sin(0:0.001:2*pi), ...
    'LineWidth', 2, 'Color', 'r');

plot(shotsdisplay(Rdisplay < 1, 1), ...
   shotsdisplay(Rdisplay < 1, 2), 'x', ...
   'MarkerSize', 14, 'LineWidth', 2, 'Color', 'r');
plot(shotsdisplay(Rdisplay >= 1, 1),...
   shotsdisplay(Rdisplay >= 1, 2), 'bd',...
   'MarkerSize', 12);
```

Exercise III.a

- 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` $\times$ `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).

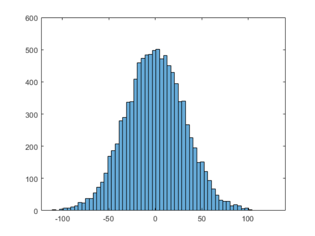
  - What probability distribution do you expect?
  - Plot resulting deviations of number of heads.
    - Use function `histogram`.

600

Exercise III.b

```
N = 1e4; % number of series
M = 1e3; % number of throws in one set
throws = randi([0 1], M, N)*2 - 1; % generate numbers -1 and 1
nOnes = sum(throws == 1);
nOnesOverAverage = sum(throws); % is vector
figure(1);
histogram(nOnesOverAverage, 60); % 60 bins
```

```
N = 1e4; M = 1e3; % economy code
histogram(sum(randi([0 1], M, N)*2 - 1), 60);
```



- Mean and standard deviation of `nOnesOverAverage`:

```
mean(nOnesOverAverage)
```

```
std(nOnesOverAverage)
```

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

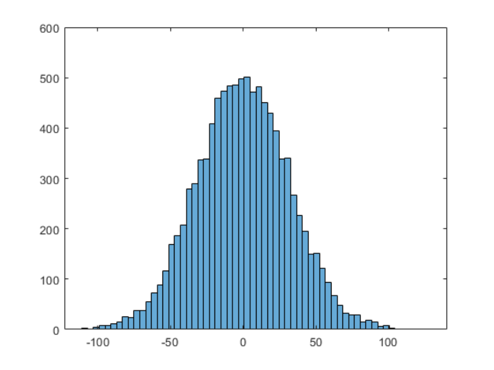
$$\sigma = \sqrt{\frac{\sum_i (\mu - x_i)^2}{N}} = \sqrt{1000} \approx 31.62$$

Exercise III.c

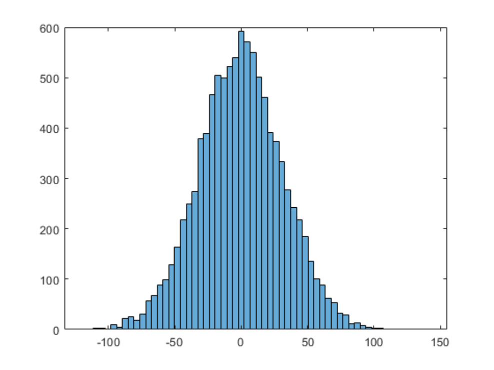
- To test whether we get similar distribution for directly generated data:

```
figure(2);
histogram(0 + 31.62*randn(N,1), 60);
```

Coin toss:



Directly generated data:

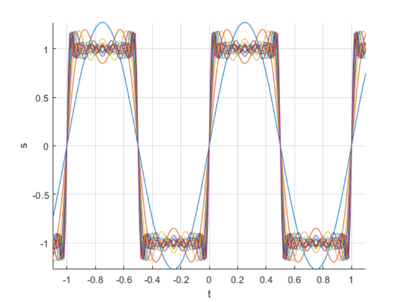


Exercise IV.

- 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 $t \in [-1.1;\ 1.1]$ s; use $A = 1$ V and $T = 1$ s.

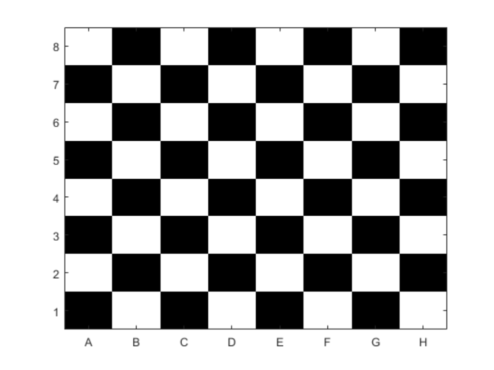



```
close all; clear; clc;
t = -1.1:0.01:1.1;
s = zeros(1, length(t));
T = 1; A = 1;
figure;
hold on; grid on; axis tight;
xlabel('t'); ylabel('s');

for k = 0:10
  s = s + A*4/pi* ...
    sin(2*pi*t*(2*k+1)/T)/(2*k+1);
  plot(t, s);
end
```

600

Exercise V.

- Modify the axes of the chessboard so that it corresponded to reality:

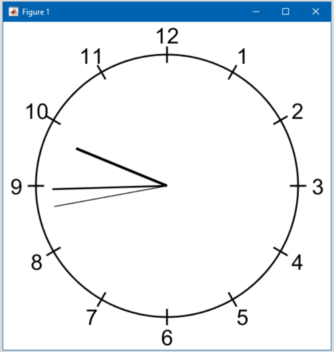



```
CH = repmat(eye(2), 4, 4);
hAx = axes;
imagesc(hAx, CH);
colormap gray;

str = char(65:72);
hAx.XTickLabel = str(:);
hAx.YTickLabel = ...
  hAx.YTickLabel(end:-1:1);
```

600

Exercise VI.a

- Create a script which shows a figure with a clock face showing actual time.
- To determine actual time use function `clock`.



Exercise VI.b

```
close all; clear; clc;
actualTime = clock;
actualTime = actualTime(4:6); % get just hours, minutes and seconds
relativeTime = actualTime./[1 60 60^2]; % in hours
figSize = [500, 500]; % figure size
screenSize = get(groot, 'ScreenSize');
dialRadius = 0.8;
hourCoord = [(dialRadius + [1; -1]*0.05)*exp(1j*(pi/6:pi/6:2*pi)); nan(1, 12) + 1j*nan(1, 12)];

hFig = figure('MenuBar', 'none', 'Color', 'w', 'Position', [(screenSize(3:4) - figSize)/2, figSize]);
hAx = axes('Parent', hFig, 'XLim', [-1, 1], 'YLim', [-1, 1], 'Position', [0 0 1 1], 'XColor', 'none', 'YColor', 'none');
dialArg = 0:0.01:2*pi;
% dial:
line('Parent', hAx, 'XData', dialRadius*cos(dialArg), 'YData', dialRadius*sin(dialArg), 'Color', 'k', 'LineWidth', 2);
% hour marks:
line('Parent', hAx, 'Color', 'k', 'XData', real(hourCoord(:)), 'YData', imag(hourCoord(:)), 'LineWidth', 2);
% hour labels:
hTexts = gobjects(12, 1);
for iObj = 1:12
   iAngle = -iObj*pi/6 + pi/2;
   hTexts(iObj) = text('Parent', hAx, 'Color', 'k', 'FontSize', 25, 'HorizontalAlignment', 'center', ...
      'String', sprintf('%i', iObj), 'Position', (dialRadius + 0.12)*[cos(iAngle), sin(iAngle)]);
end
% hands:
hHour = line('Parent', hAx, 'LineWidth', 3, ...
   'XData', [0, 0.6*cos(-sum(relativeTime)*pi/6 + pi/2)], ...
   'YData', [0, 0.6*sin(-sum(relativeTime)*pi/6 + pi/2)]);
hMinute = line('Parent', hAx, 'LineWidth', 2, ...
   'XData', [0 0.7*cos(-sum(relativeTime(2:3))*2*pi + pi/2)], ...
   'YData', [0 0.7*sin(-sum(relativeTime(2:3))*2*pi + pi/2)]);
hSecond = line('Parent', hAx, 'LineWidth', 1, ...
   'XData', [0 0.7*cos(-actualTime(3)*pi/30 + pi/2)], ...
   'YData', [0 0.7*sin(-actualTime(3)*pi/30 + pi/2)]);
```

Questions?

B0B17MTB, BE0B17MTB – MATLAB
matlab@fel.cvut.cz

April 11, 2024
Summer semester 2023/24

Acknowledgement: Filip Kozák, Pavel Valtr, Michal Mašek, and Vít Losenický.