

AE0B17MTB – Matlab

# Part #10



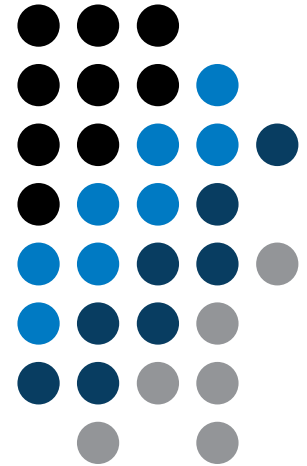
Miloslav Čapek

`miloslav.capek@fel.cvut.cz`

Filip Kozák, Viktor Adler, Pavel Valtr

Department of Electromagnetic Field

B2-626, Prague



# Learning how to ...

---

## Visualization in Matlab #2

### GUI #1

!!! **Attention:** SINCE MATLAB R2014b CHANGES IN GRAPHICS !!!

# Advanced visualizing in Matlab

- basic possibilities of visualizing mentioned in 6th part of the course
  - `figure` and basic plotting (`plot`, `stem`, ...)
  - setting curve options of a graph `LineStyleSpec` (doc [LineStyleSpec](#))
  - functions for graph description (`title`), `grid`, `legend`, etc.
- graph options
  - graph as a handle object (change since version R2014b)
  - way of setting property values of graphic "objects"
- selected advanced possibilities of visualizing
  - inserting more graphs in a single `figure`
  - tens of types of graphs (see `Help`)
  - projection of 3D graphs
  - `view`, `colormap`

# Object identifiers (up to R2014b)

- each individual object has its own identifier ('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
  - complex graphs (contours) may have more identifiers
- root has always handle = 0 (more on root later), figure usually an integer, other objects have handle equal to positive real number (of class double)

handles

```
>> figHandle = figure;  
>> axHandle = axes;
```

- number stored in `figHandle` variable exists even after closing the window, but it is not a handle any more

# Object identifiers (since R2014b)

- each graphic object is marked as an object in workspace
  - an object is defined by its class with its properties and methods
- `root` can still be accessed using function `get()` with parameter 0
  - `root` is newly `groot` object
  - (more in part GUI #1)
- after object destruction (closing figure)
  - the object still exists in workspace (it appears as a reference to deleted object)

# Advanced visualization in Matlab

- graph as a handle number (version < R2014b)
- graph as an object (since version R2014b)
  - note: in what follows we will reference graphs as handle objects

Command Window

```
>> p1 = plot(0:10)

p1 =

Line with properties:

    Color: [0 0.4470 0.7410]
  LineStyle: '-'
  LineWidth: 0.5000
    Marker: 'none'
  MarkerSize: 6
  MarkerFaceColor: 'none'
    XData: [1 2 3 4 5 6 7 8 9 10 11]
    YData: [0 1 2 3 4 5 6 7 8 9 10]
    ZData: [1x0 double]

Show all properties
```

Workspace

Name	Value	Class	Bytes	Size
p1	1x1 Line	matlab.graphics.chart.primitive.Line	112	1x1

Command Window

```
>> p1 = plot(0:10)

p1 =

174.0016
```

Workspace

Name	Value	Bytes	Size	Class	Min
p1	174.0016	8	1x1	double	174.0016

# Advanced visualization in Matlab

- Property editor (Inspector)

The screenshot displays the MATLAB environment with a plot of a sine wave. The plot is titled 'Figure 1' and shows a blue line oscillating between -1 and 1 on the y-axis, with the x-axis ranging from 0 to 400. The plot is surrounded by several toolboxes:

- Figure Palette:** Contains options for 'New Subplots' (2D Axes, 3D Axes), 'Variables', and 'Annotations' (Line, Arrow, Double Arrow, Text Arrow, Text Box, Rectangle, Ellipse).
- Plot Browser:** Shows the current plot as 'Axes (no title)'.
- Inspector (matlab.graphics.axes.Axes):** A detailed property editor for the axes, listing various settings such as 'ALim', 'Color', 'FontName', and 'FontSize'. A red box highlights the 'More Properties...' button at the bottom right of this window.
- Property Editor - Axes:** A secondary property editor for the axes, showing fields for 'X Label', 'X Limits' (0 to 400), and 'X Scale' (linear). A red box highlights the 'More Properties...' button at the bottom right of this window.

# Advanced visualization in Matlab

- the way of setting handle object properties
  - the possibility of using functions `set` and `get` exists for both versions
    - not case sensitive

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

```
>> set(myPlotObj, 'color', 'r')  
>> get(myPlotObj, 'color')
```

- dot notation (only for versions R2014b and higher)
  - is cAsE sEnSiTiVe

```
>> myPlotObj = plot(1:10);  
>> myPlotObj.Color
```

```
>> myPlotObj.Color = 'r';  
>> myPlotObj.Color
```



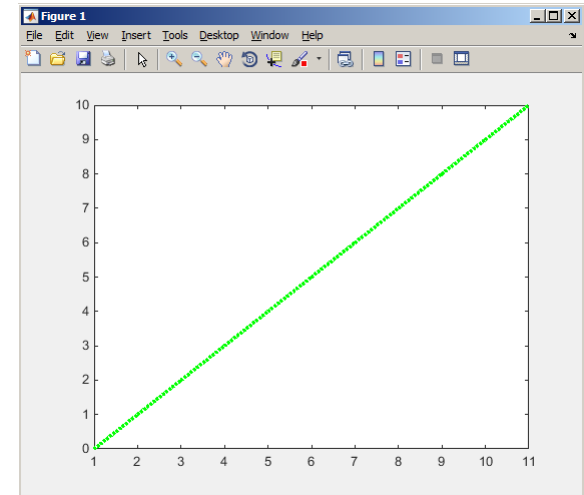
# get and set functions

60 s ↑

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

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

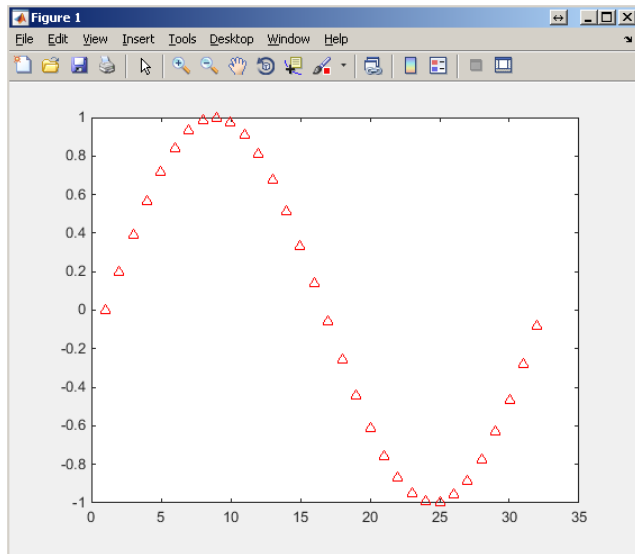
- find out the thickness of the line and increase it by 1.5
- set the line color to green
- set the line style to dotted



# Dot notation application

60 s ↑

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

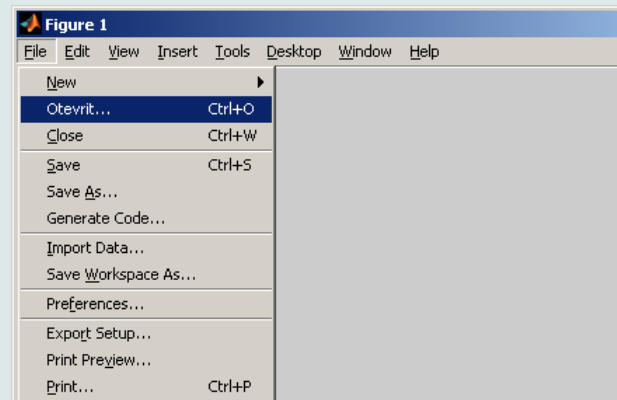
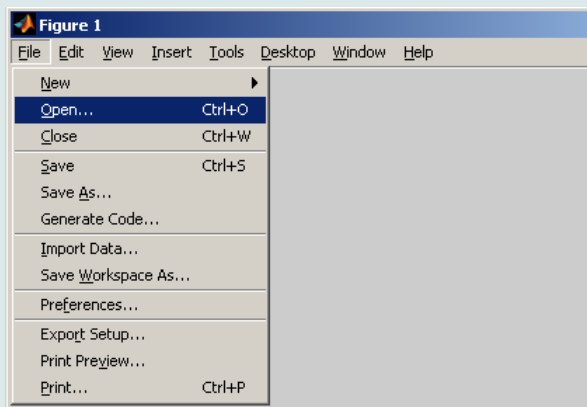


```
myPlotObj = plot(sin(0:0.2:2*pi));
```

# 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 😊 )

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



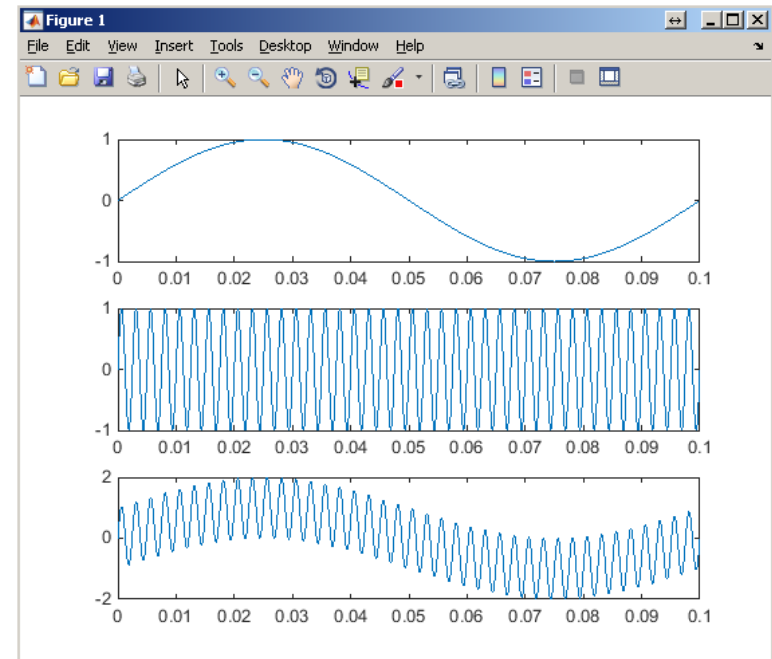
# More graphs in a window – subplot

- inserting several different graphs in a single window figure
  - function `subplot(m, n, p)`
  - `m` – number of lines
  - `n` – number of columns
  - `p` – position

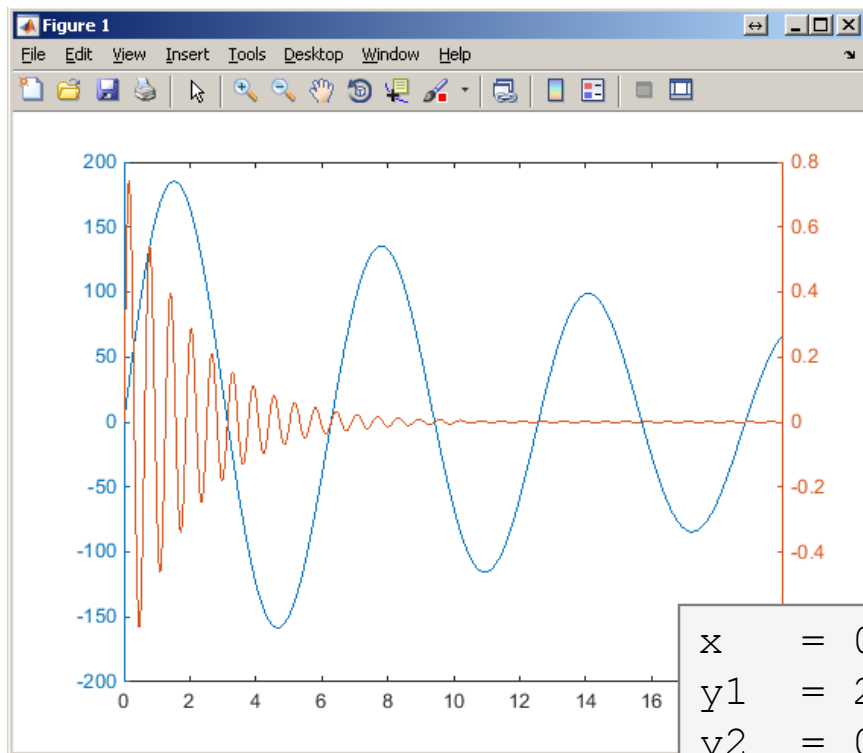
```
t = linspace(0, 0.1, 0.1*10e3);
f1 = 10;    f2 = 400;

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

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



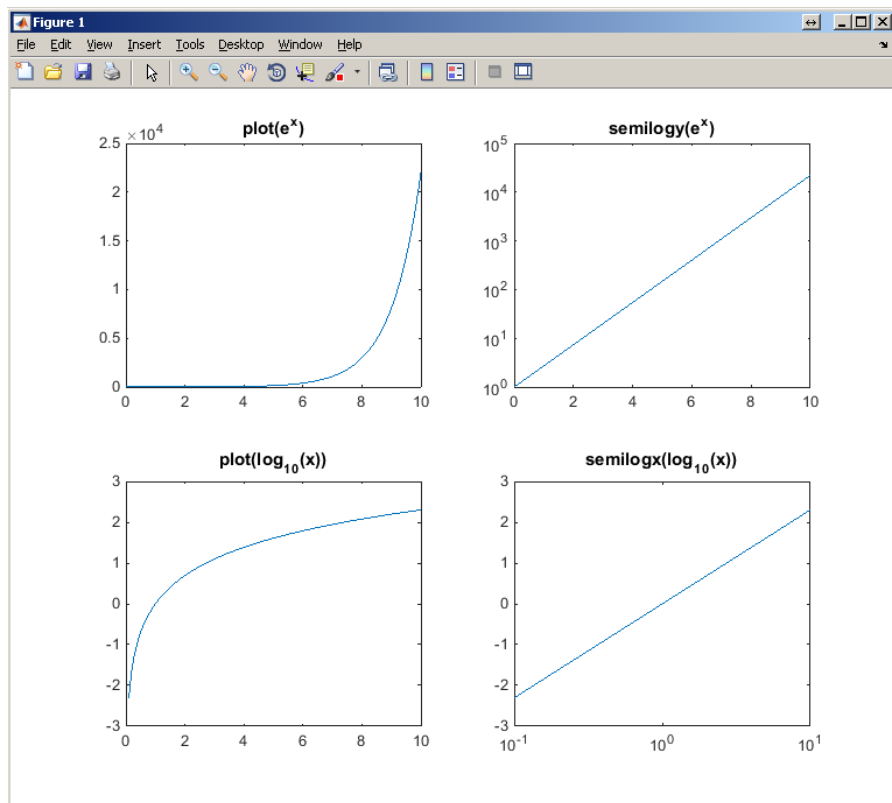
# Double y axis – yyaxis



```
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);
```

# Logarithmic scale

- functions `semilogy`, `semilogx`, `loglog`



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

```
figure('color', 'w')
subplot(2, 2, 1); plot(x, y1);
title('plot(e^x)');
```

```
subplot(2, 2, 2); semilogy(x, y1);
title('semilogy(e^x)')
```

```
subplot(2, 2, 3); plot(x, y2);
title('plot(log_1_0(x))')
```

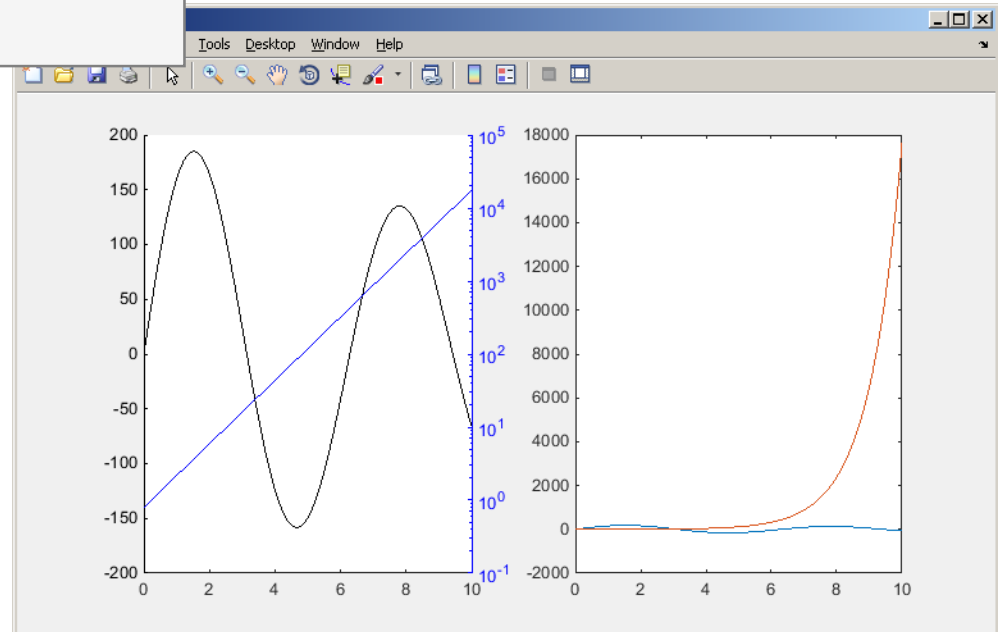
```
subplot(2, 2, 4); semilogx(x, y2);
title('semilogx(log_1_0(x))')
```

# Example

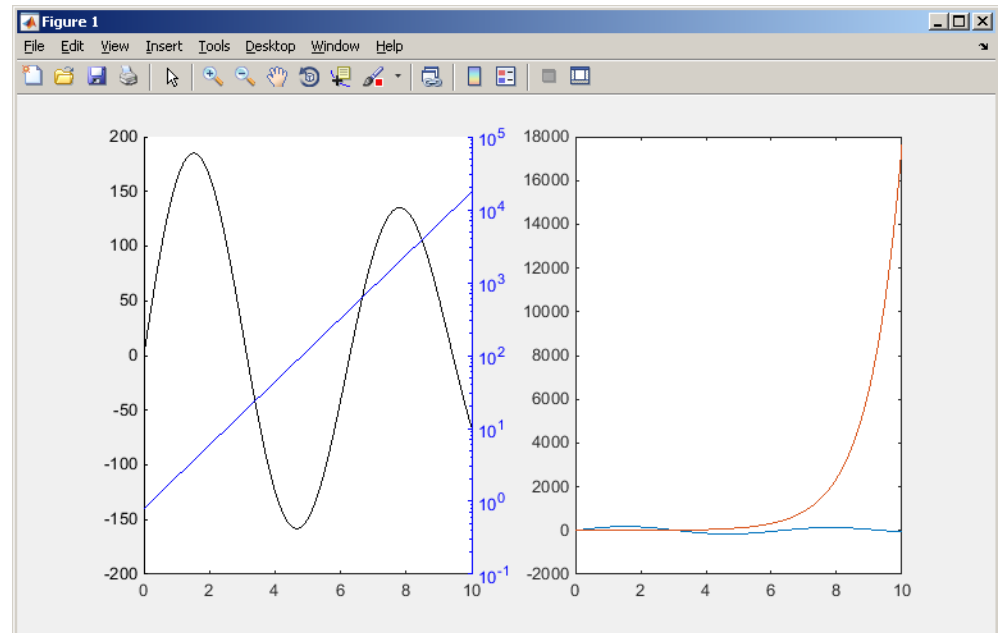
600 s ↑

- use functions `plot`, `yyaxis`, `semilogy` and `subplot` in one figure object for creation figure shown below
  - in the object created by `subplot` change default colors of individual axes to blue and black (property `YAxis`)

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



# Example - solution



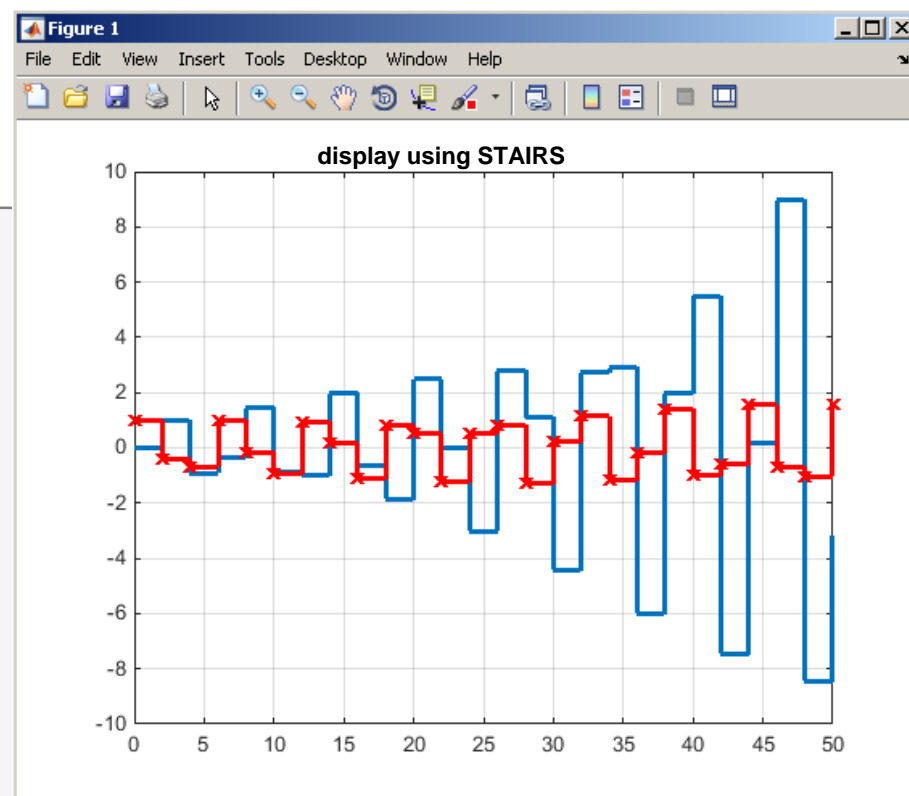


# stairs

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

figure('Color', 'w');
stairs(x, y1, 'LineWidth', 2);
hold on; grid on;
stairs(x, y2, ...
      'Color', 'r', ...
      'Marker', 'x', ...
      'LineWidth', 2);

title('display using STAIRS');
```



# Plotting 2-D functions

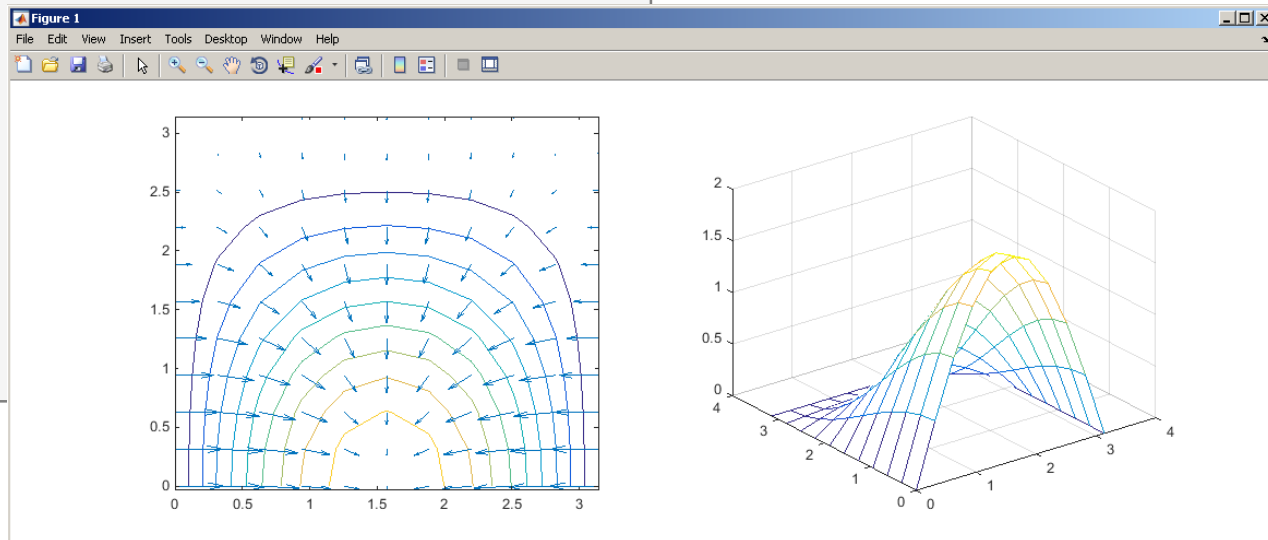
- `contour`, `quiver`, `mesh`

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

```
figure('Color','w');
```

```
subplot(1, 2, 1);
contour(x, y, z);
hold on;
quiver(t, t, gx, gy);
```

```
subplot(1, 2, 2);
mesh(x, y, z);
```



# Advanced visualizing in Matlab

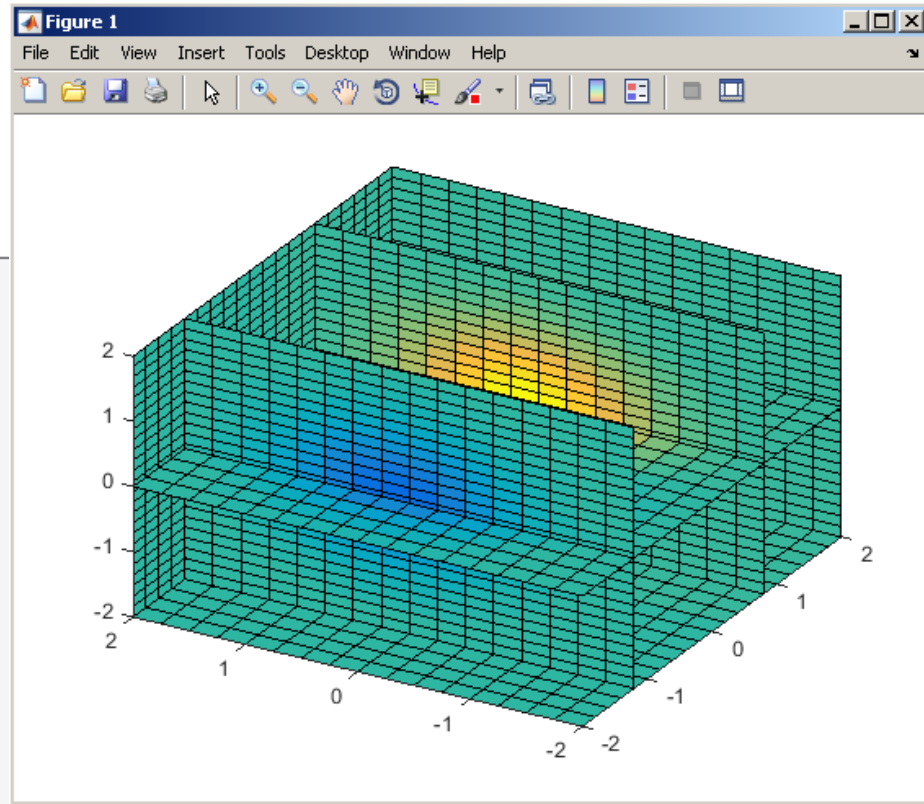
- function slice
- function view

```
[x, y, z] = meshgrid(-2:0.2:2, ...
                    -2:0.25:2, ...
                    -2:0.16:2);

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

xslice = [-1.2, 0.8, 2];
yslice = 2;
zslice = [-2, 0];

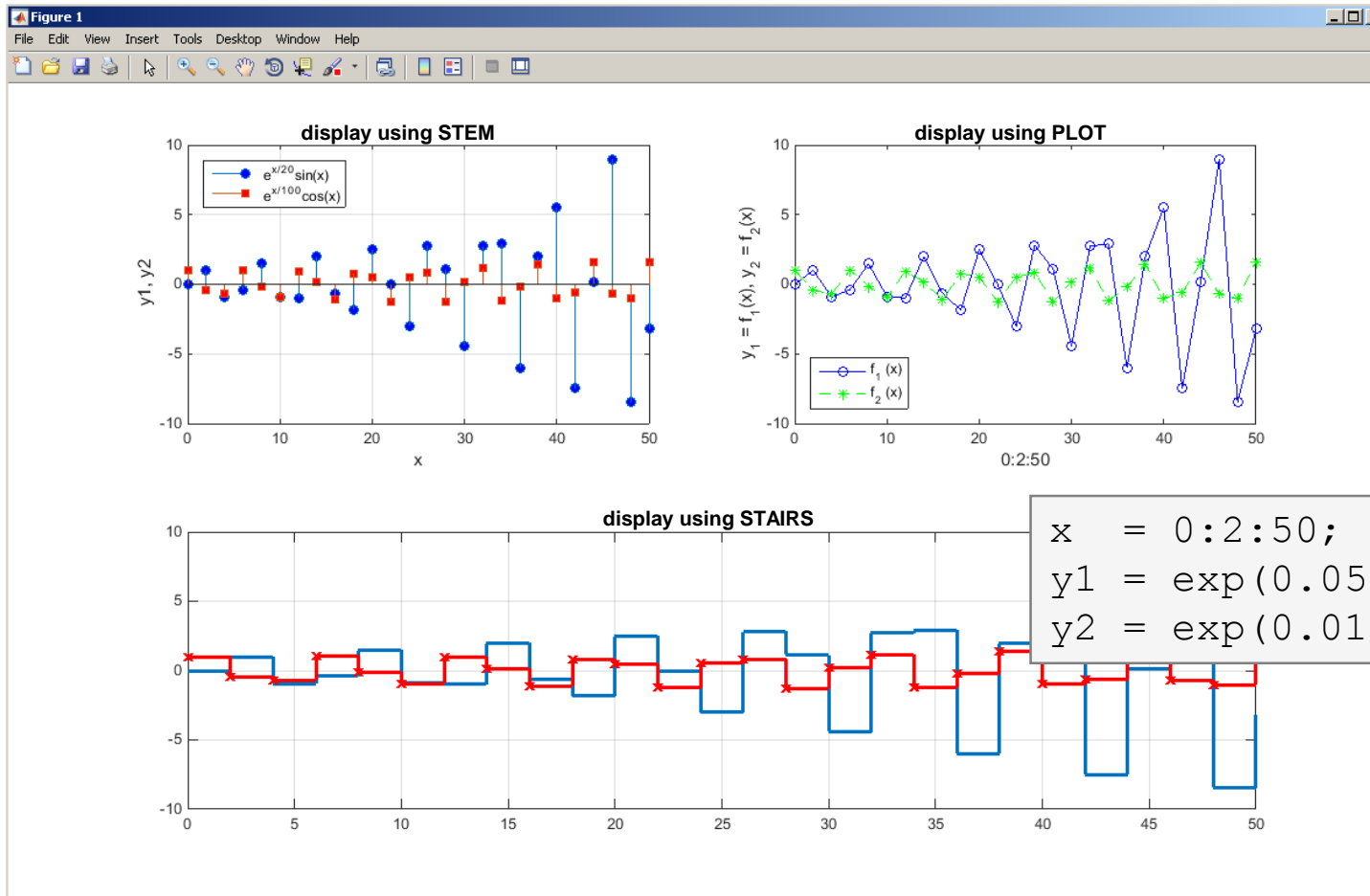
figure('Color', 'w');
slice(x, y, z, v, xslice, yslice, zslice);
% view(azimuth, elevation)
view(-60, 40);
```



# Exercise #1 assignment

600 s ↑

- try to imitate the figure below where functions  $y_1$  and  $y_2$  are defined as:



# Exercise #1 solution

---

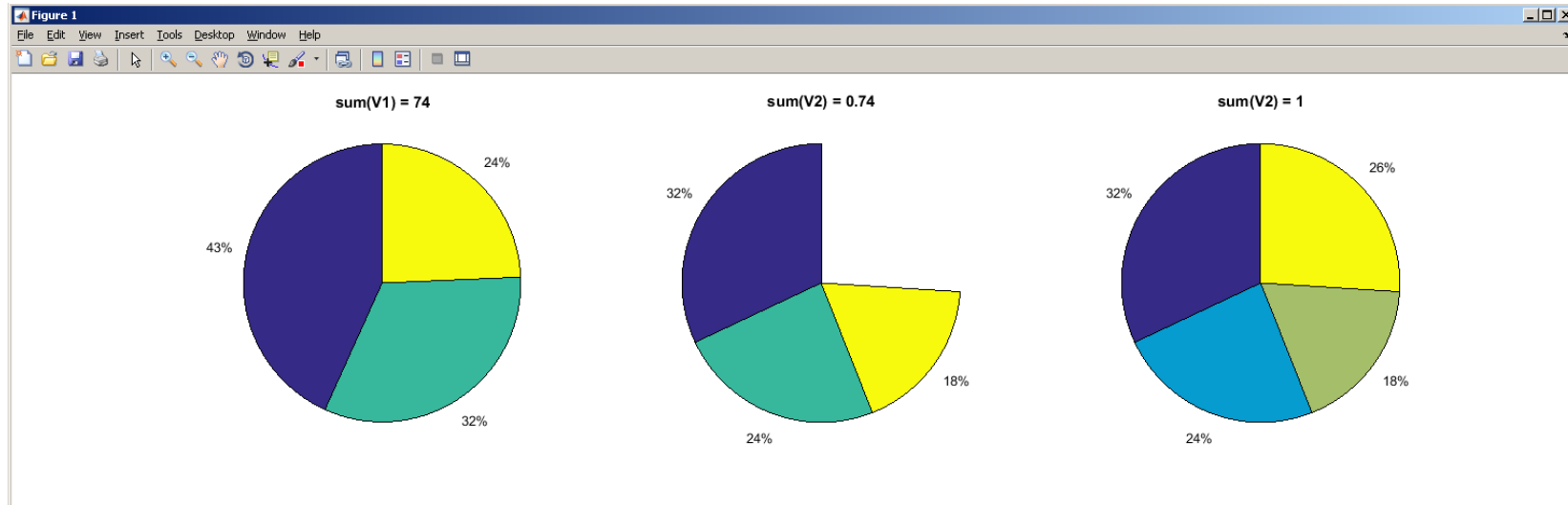
# Pie plot – pie, pie3

```

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');

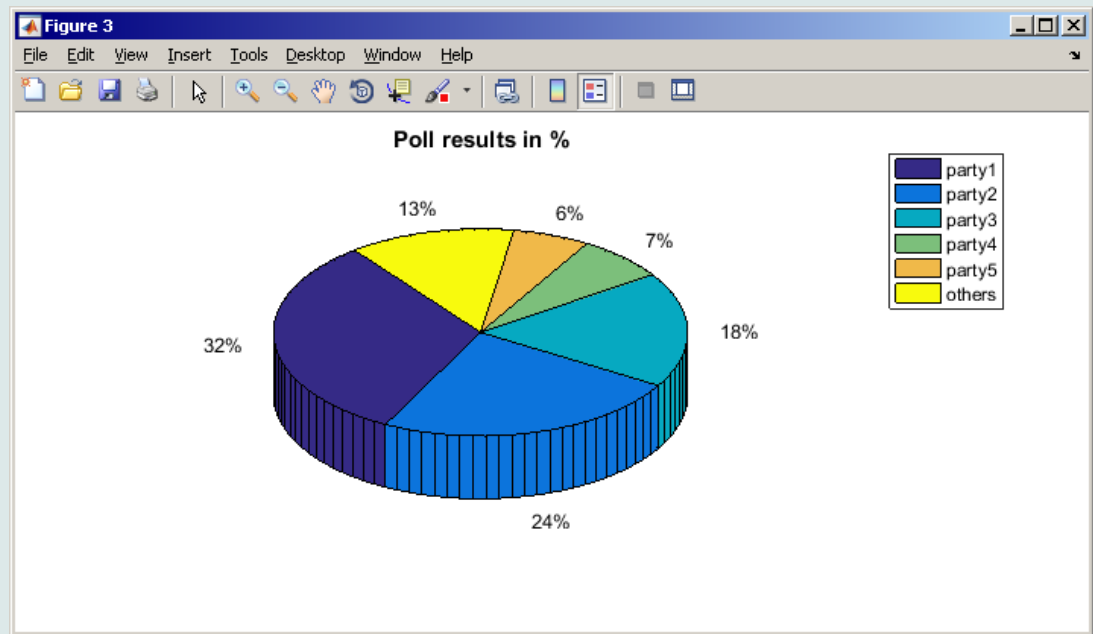
```



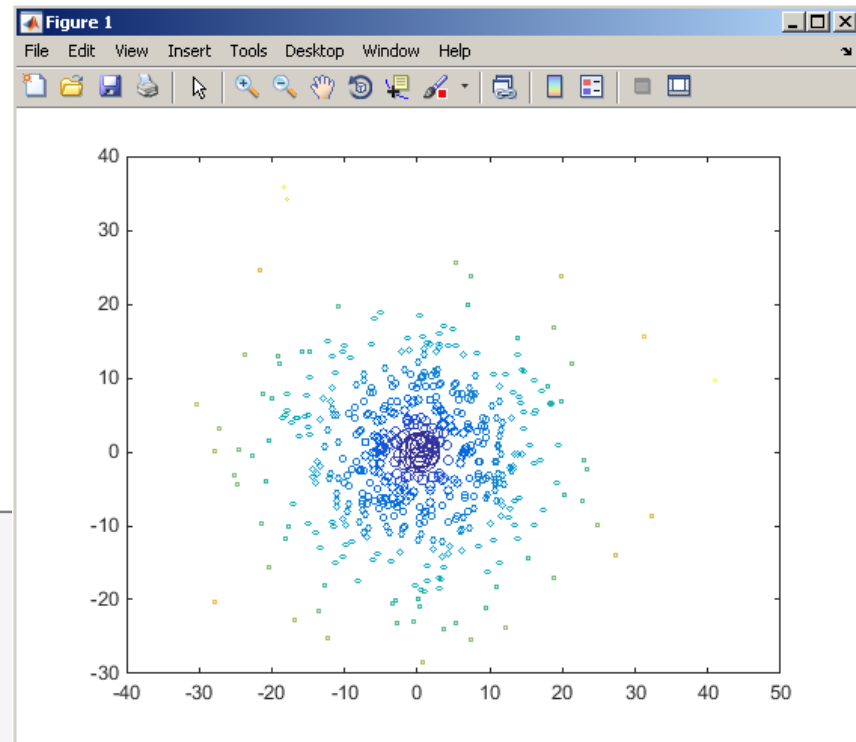
# Exercise

600 s ↑

- opinion polls show parties' preference projections as follows:
- plot the poll result using pie plot including the item 'others'
  - 1st party: 32%
  - 2nd party: 24%
  - 3rd party: 18%
  - 4th party: 7%
  - 5th party: 6%
  - 6th party: 6%



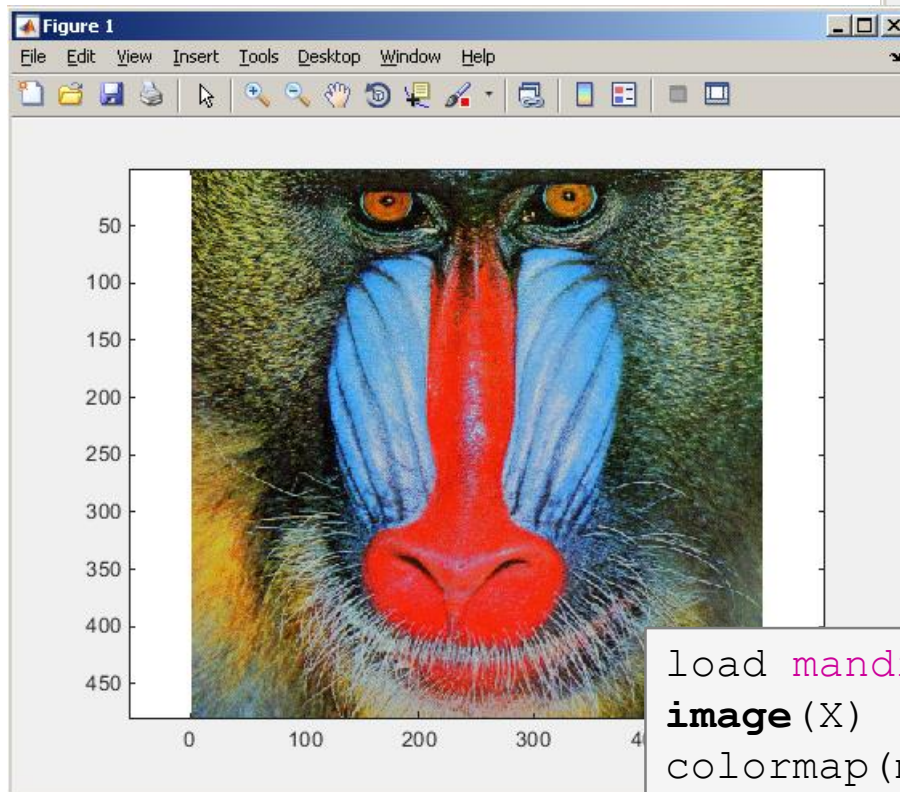
```
x = 10 * randn(500, 1);  
y = 10 * randn(500, 1);  
c = hypot(x, y);  
  
figure('color', 'w');  
scatter(x, y, 100./c, c);  
box on;
```



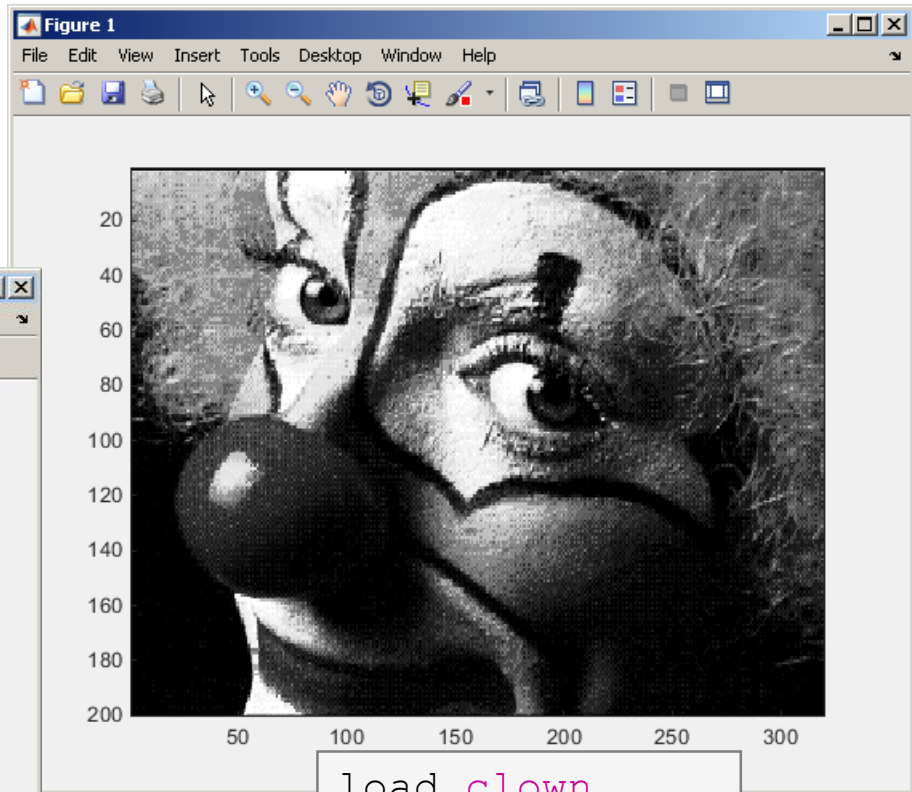


# Picture depiction

- functions `image`, `imagesc`
- function `colormap`



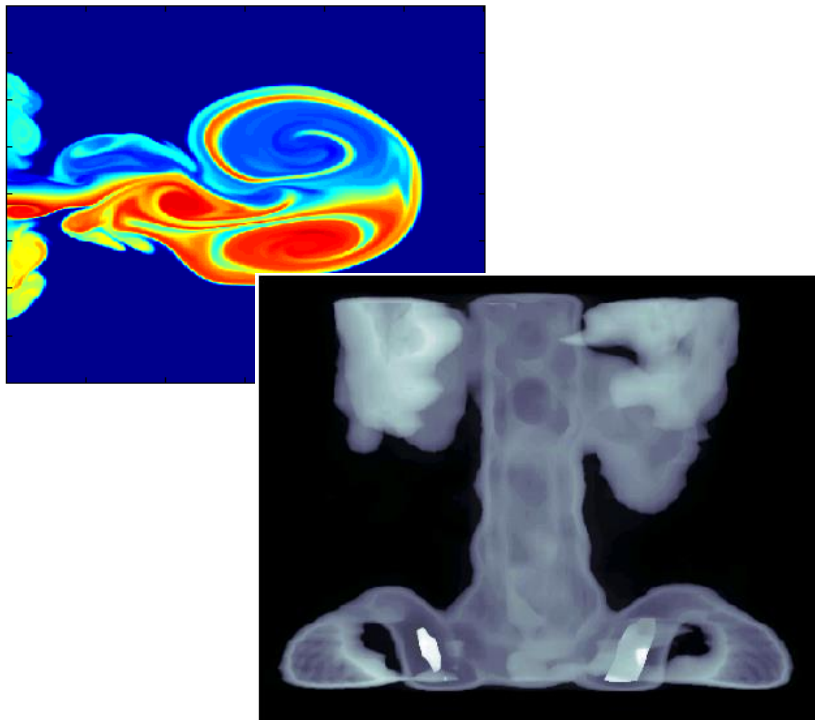
```
load mandrill
image(X)
colormap(map)
axis equal
```



```
load clown
imagesc(X)
colormap(gray)
```

# colormap

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

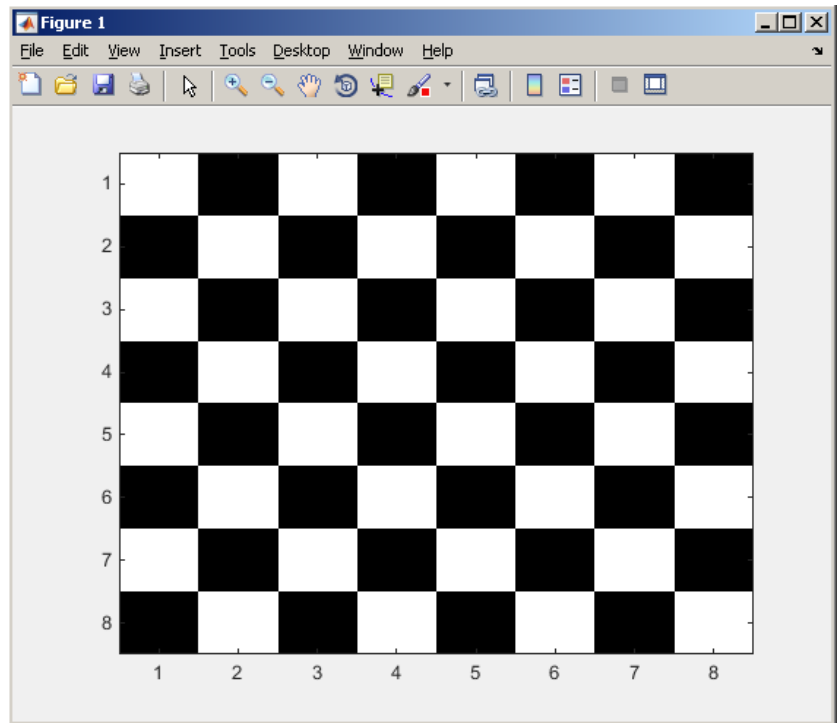


Colormap Name	Color Scale
parula	
jet	
hsv	
hot	
cool	
spring	
summer	
autumn	
winter	
gray	
bone	
copper	
pink	
lines	
colorcube	
prism	
flag	
white	

# Exercise

600 s ↑

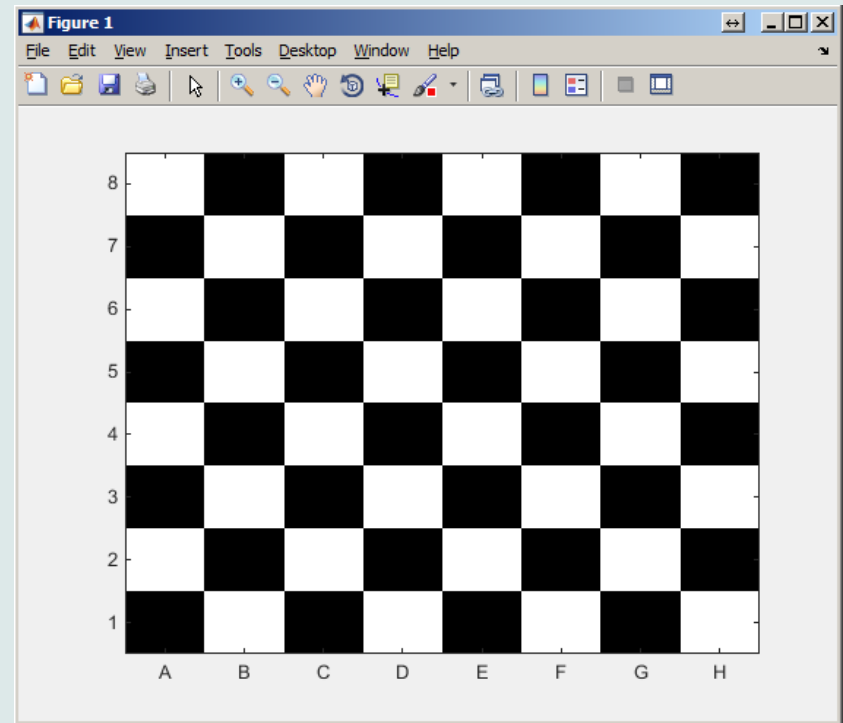
- create a chessboard as shown in the figure:
  - the picture can be drawn using the function `imagesc`
  - consider `colormap` setting



# Exercise

600 s ↑

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



# Structure of GUI #1

screen ~ groot

application window ~ figure

The screenshot shows the IFSMaker application window. The main drawing space contains a fractal composed of blue and light green polygons. The drawing space is overlaid with a dashed grid and axes. The application window has a menu bar (File, Tools, Points, Lines, Selection, Modification, Polygons, Transformations, IFS fractal, Parameter sweep) and a toolbar. A right-hand panel contains various controls and data tables.

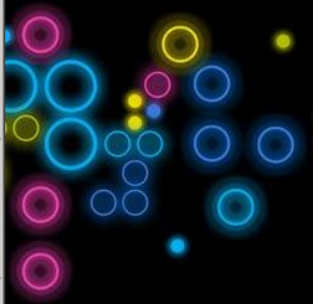
Name	Tag	X coord	Y coord	Show	pt:R	pt:G	pt:B	pSize
1	Point1	FRC1	-50	-30	<input checked="" type="checkbox"/>	0	0	1 medium
2	Point2	FRC2	50	-30	<input checked="" type="checkbox"/>	0	0	1 medium
3	Point3	FRC3	50	30	<input checked="" type="checkbox"/>	0	0	1 medium
4	Point4	FRC4	-50	30	<input checked="" type="checkbox"/>	0	0	1 medium

Name	Tag	IFS	Poly	Show	ID	Name	Tag
1	Polyg1	FRC	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	1	1 Point1	FRC1
2					2	2 Point2	FRC2
3					3	3 Point3	FRC3
4					4	4 Point4	FRC4

drawing space ~ axes

graphical objects ~ uiobjects



# Structure of GUI #2

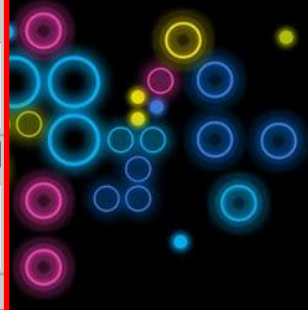
The screenshot shows the IFSMaker application window with various GUI components highlighted by red boxes and labels. The components are: uitoolbar (top toolbar), uimenu (top menu bar), uitable (table on the right side), uicontextmenu (context menu on the right side), uipanel (main workspace area), uibottongroup (bottom toolbar), and uicontrol (bottom status bar).

The main workspace displays a fractal structure with labels FRC4, FRC5, and FRC3. The right-hand panel contains a table of points and polygons, and a table of polygons.

Name	Tag	X coord	Y coord	Show	pt:R	pt:G	pt:B	pSize
1 Point1	FRC1	-50	-30	<input checked="" type="checkbox"/>	0	0	1	medium
2 Point2	FRC2	50	-30	<input checked="" type="checkbox"/>	0	0	1	medium
3 Point3	FRC3	50	30	<input checked="" type="checkbox"/>	0	0	1	medium
4 Point4	FRC4	-50	30	<input checked="" type="checkbox"/>	0	0	1	medium

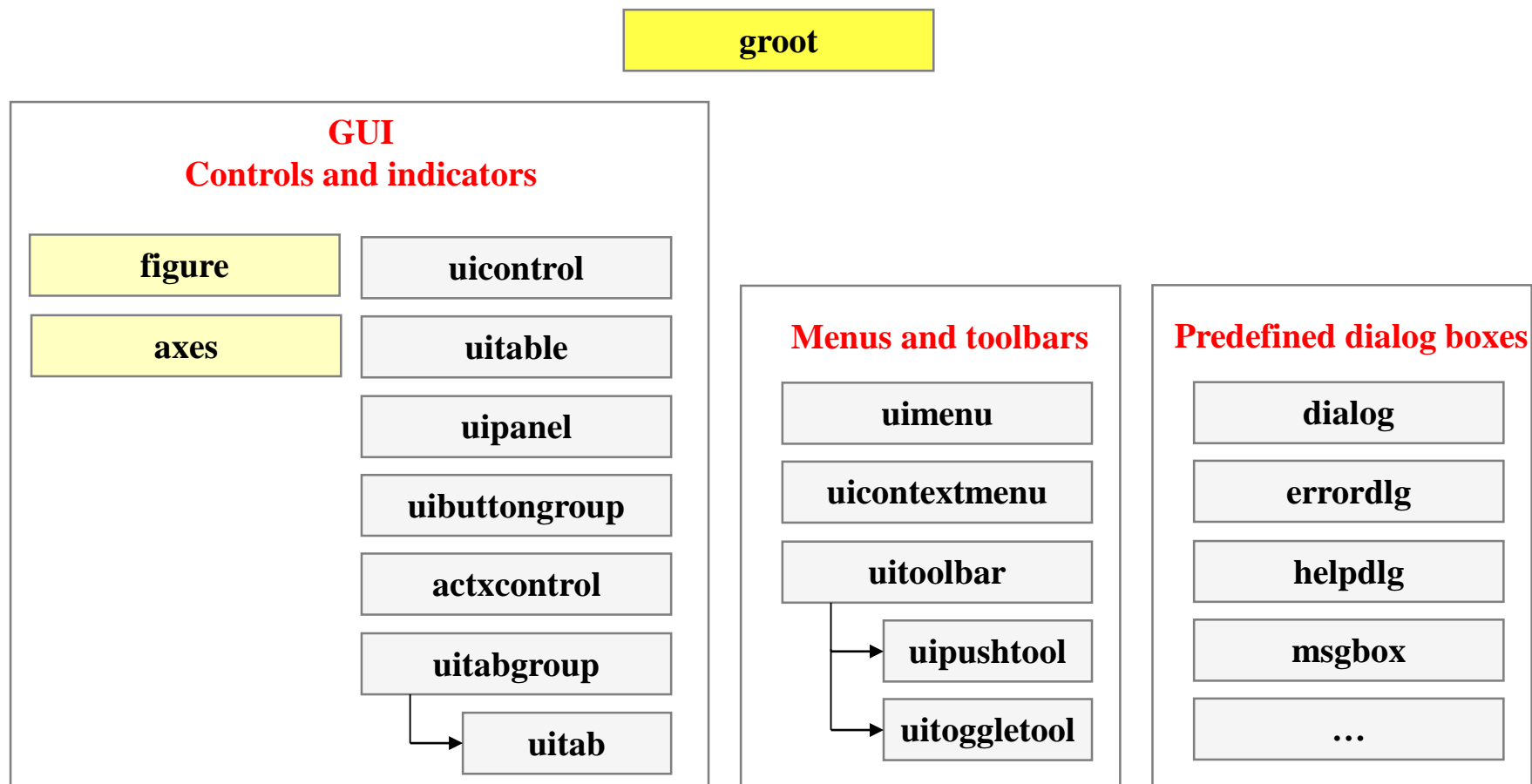
  

Name	Tag	IFS	Poly	Show	p	ID	Name	Tag
1 Polyg1	FRC	<input checked="" type="checkbox"/>	0	<input checked="" type="checkbox"/>		1	1 Point1	FRC1
						2	2 Point2	FRC2
						3	3 Point3	FRC3
						4	4 Point4	FRC4



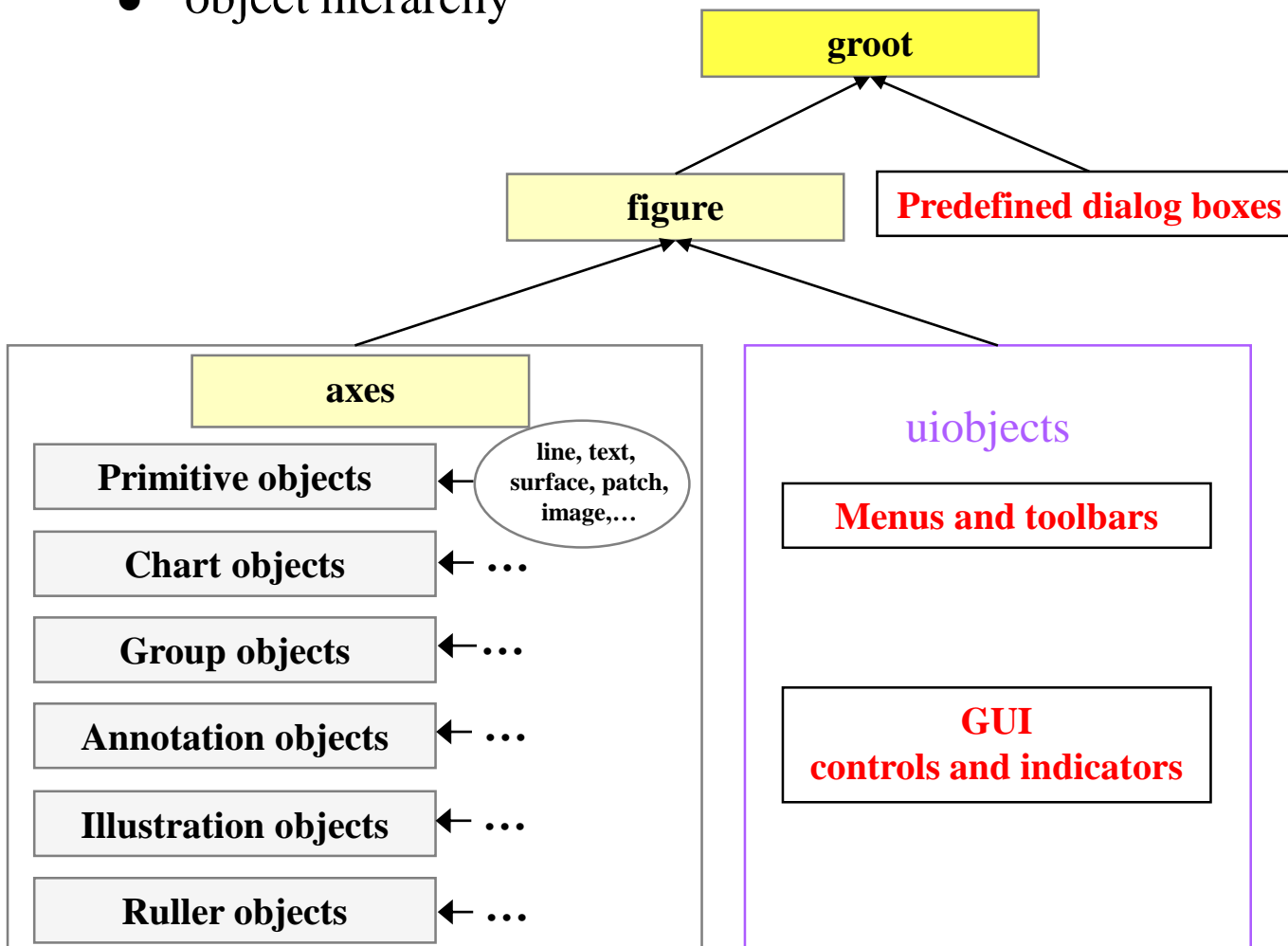
# Structure of GUI

- objects are sorted in a logical way



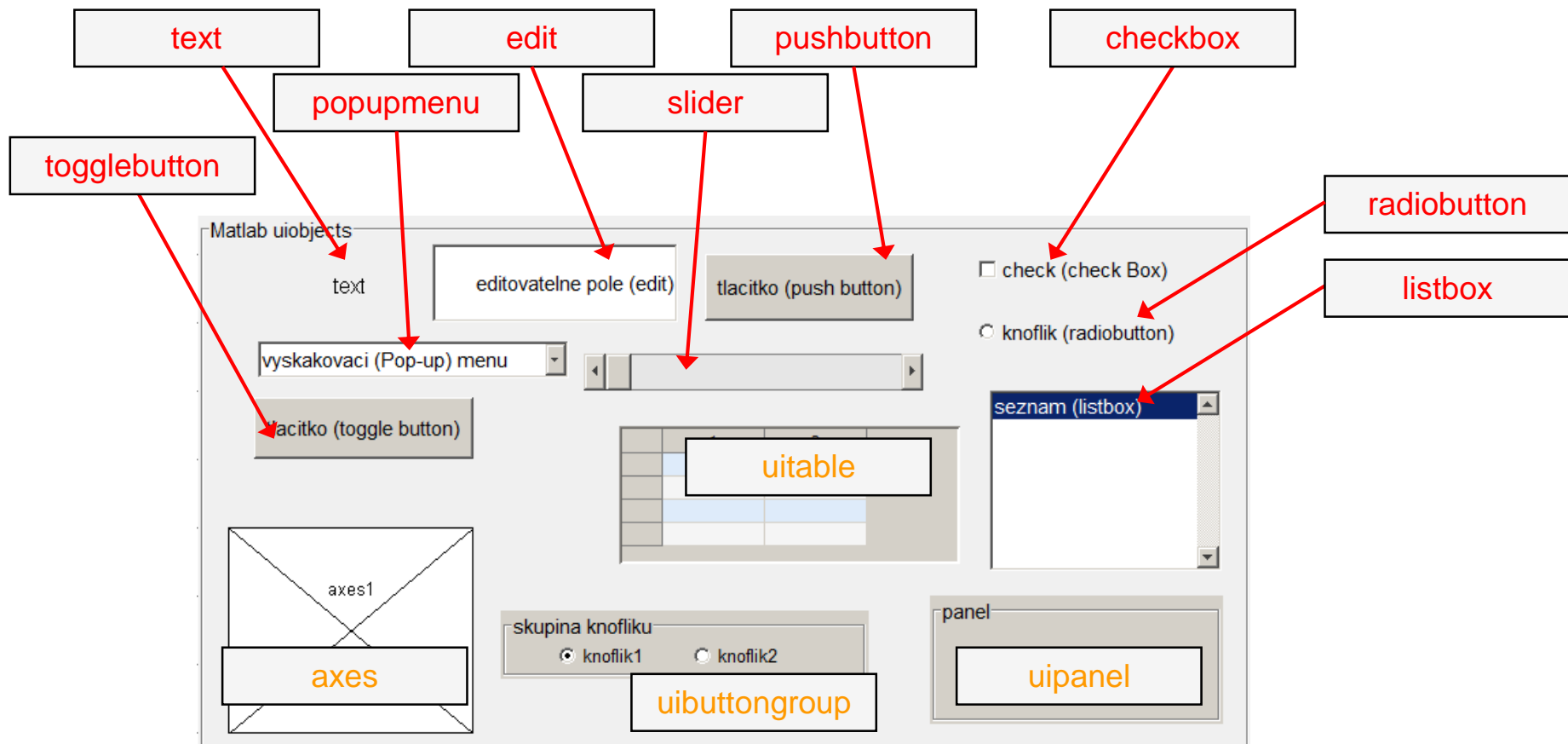
# Structure of GUI

- object hierarchy





# Structure of GUI #3



# Screen properties, `groot`

- corresponds to computer screen in Matlab
- is unique and callable using function
  - `get(0)`
    - in workspace – data structure
  - `groot`
    - in workspace – handle object
- all other objects are children (descendants)

```
>> groot
```

```
ans =
```

```
Graphics Root with properties:
```

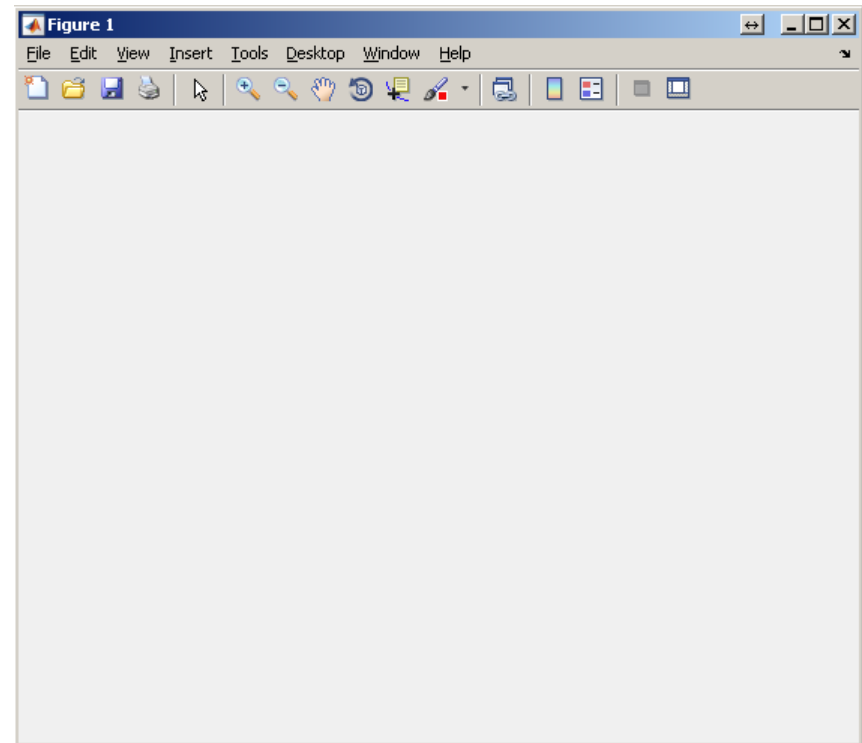
```
    CurrentFigure: [0x0 GraphicsPlaceholder]
  ScreenPixelsPerInch: 96
           ScreenSize: [1 1 1920 1200]
  MonitorPositions: [2x4 double]
                Units: 'pixels'
```

```
Show all properties
```

```
    CallbackObject: [0x0 GraphicsPlaceholder]
           Children: [0x0 GraphicsPlaceholder]
    CurrentFigure: [0x0 GraphicsPlaceholder]
  FixedWidthFontName: 'Courier New'
    HandleVisibility: 'on'
  MonitorPositions: [2x4 double]
           Parent: [0x0 GraphicsPlaceholder]
  PointerLocation: [2401 787]
           ScreenDepth: 32
  ScreenPixelsPerInch: 96
           ScreenSize: [1 1 1920 1200]
  ShowHiddenHandles: 'off'
           Tag: ''
           Type: 'root'
           Units: 'pixels'
           UserData: []
```

# Graphical window, figure

- `object figure` creates standalone graphical window
  - a new window is created on calling the function when the window doesn't exist
  - all windows are descendants of the object `groot`
  - all secondary graphic objects are descendants of the object `figure` and are drawn in the window
  - `figure` has many properties
    - see `get(figure)`
    - `hFig = figure`



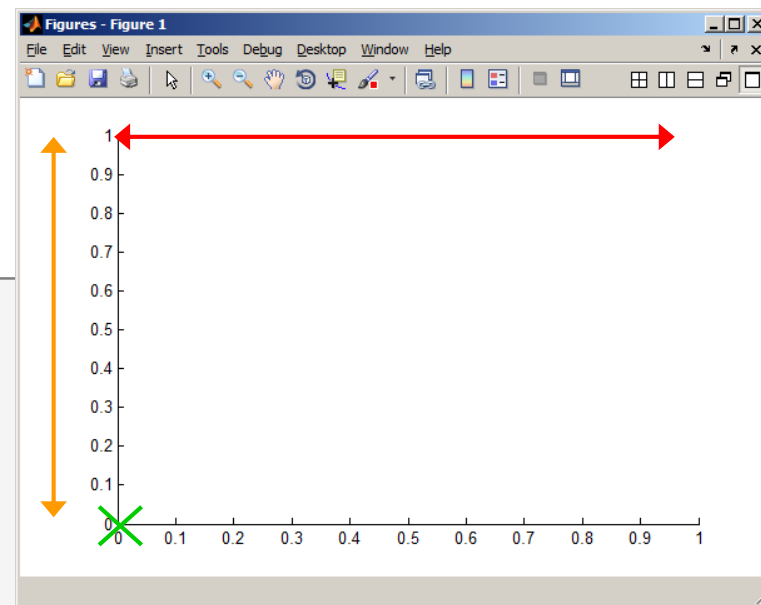
# Position **property**

- Matlab combines size of an object and its position in one matrix
- two ways of entering exist
  - (A) absolute position in pixels
  - (B) normalized position related to the size of parent object

`[left bottom width height]`

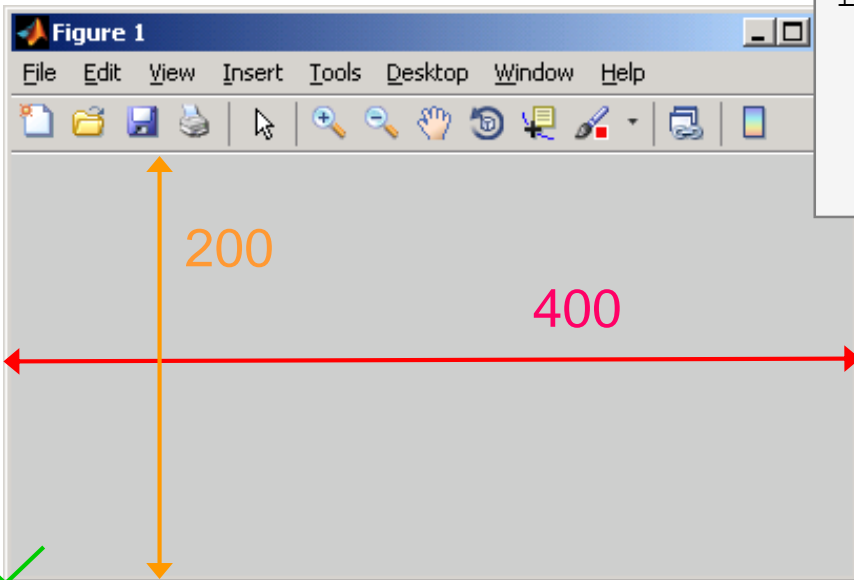
```
%% A)
uicontrol('Units','pixels',...
         'Style','pushbutton',...
         'Position',[50 150 75 25]);

%% B)
uicontrol('Units','normalized',...
         'Style','pushbutton',...
         'Position',[0.05 0.12 0.1 0.05]);
```



# Figure creation

- used when we want, for instance, to put figure in the center of the screen
  - window width: 400px, window height: 200px



[760 500]

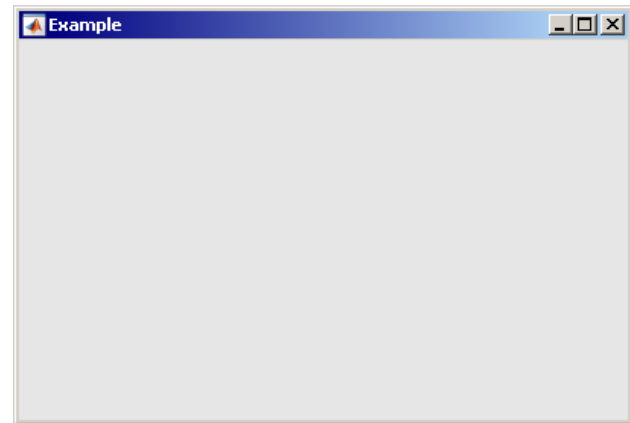
```
dispSize = get(0, 'ScreenSize');
figSize = [400 200];
figHndl = figure('pos', ...
    [(dispSize(3)-figSize(1))/2 ...
    (dispSize(4)-figSize(2))/2 ...
    figSize(1) figSize(2)]);
```

# Exercise – GUI window creation

400 s

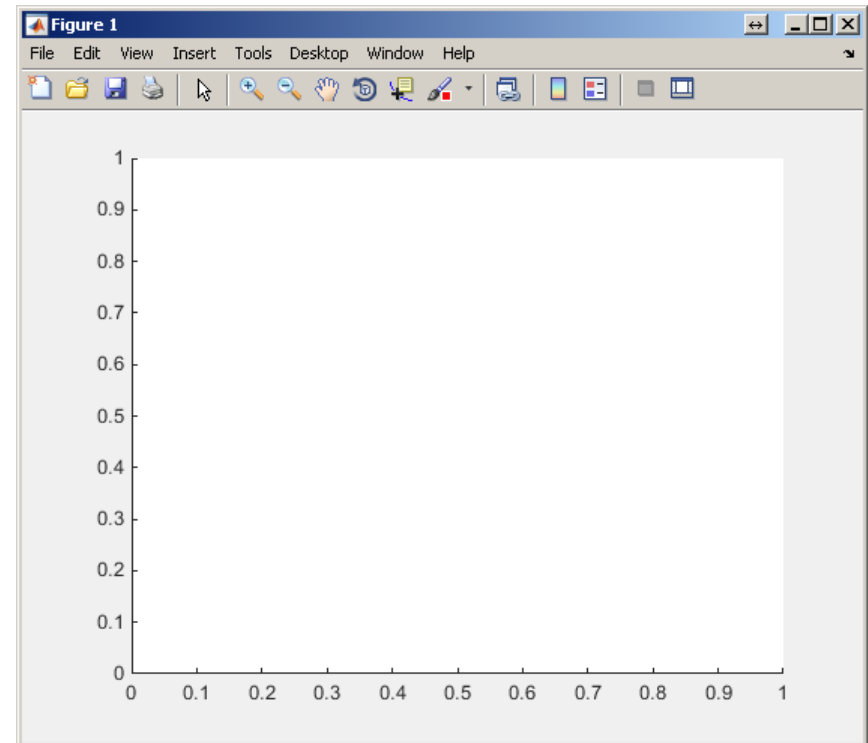


- in a new script that we will be extending throughout today's lecture create a figure window that opens in the center of the screen having width of 400 pixels and height of 250 pixels
  - make sure the figure's name is „Example“ and the title Figure 1 doesn't display
  - use Tag property for naming (e.g. `'figExample'`)
  - change window's color (up to you)



# Graph area, axes

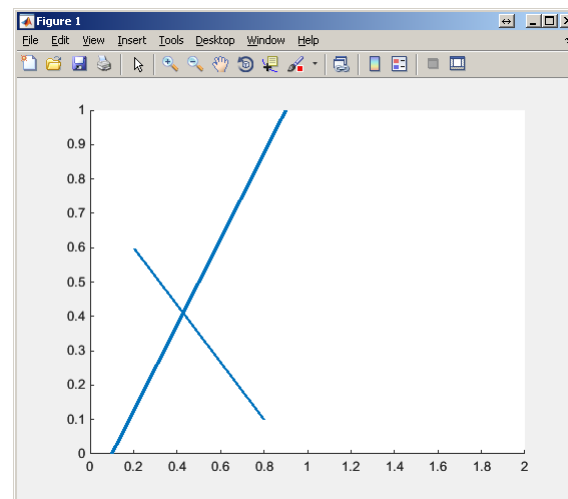
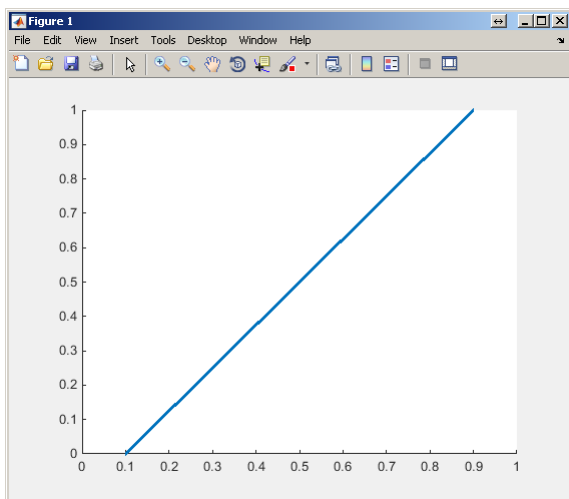
- defines area where descendants of object `axes` are placed
  - all objects related to `axes` object generate axes even when not yet exist (similarly to `figure`)
  - `axes` has many properties
    - see `get (axes)`
- or
- `properties (axes)`



# Function axis

- axis scales axes
  - format (2D): [x\_min x\_max y\_min y\_max]
  - format (3D): [x\_min x\_max y\_min y\_max z\_min z\_max]

```
line([0.1 0.9], [0 1], 'LineWidth', 3)
axis([0 1 0 1])
```



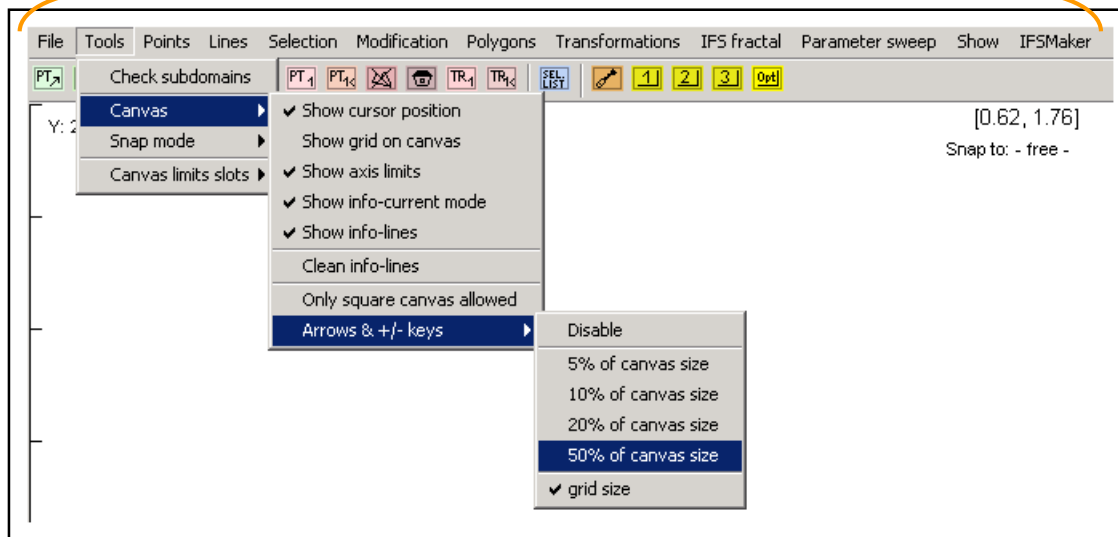
```
line([0.8 0.2], [0.1 0.6], 'LineWidth', 2)
axis([0 2 0 1])
```



# Group uiobjects: uimenu

- it is possible to define keyboard shortcuts (e.g. CTRL+L)
- it is possible to move in the menu using ALT+character
- callback function can be assigned

490 lines of code



- for more see doc [uimenu](#)

## uiobjects

uimenu

uicontextmenu

uitoolbar

uipanel

uitabgroup

uitable

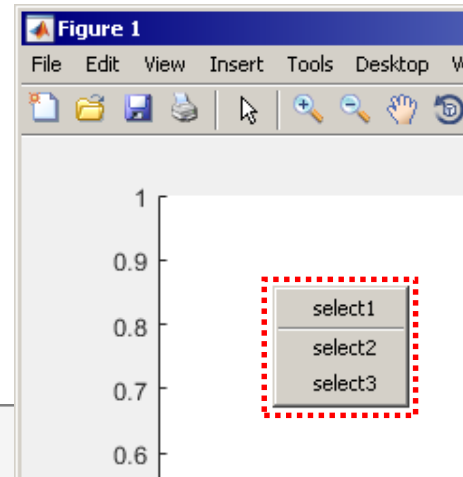
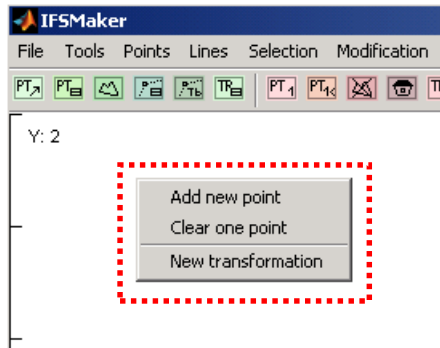
uibuttongroup

actxcontrol

uicontrol

# Group uiobjects: uicontextmenu

- creates context menu
  - appears upon mouse right-click
  - menu item selection activates related callback



```

figHndl = figure;
cMenu   = uicontextmenu;
axsHndl = axes('Parent', figHndl, 'UIContextMenu', cMenu);
uimenu(cMenu, 'Label', 'select1', 'Callback', @callbackFcn1);
uimenu(cMenu, 'Label', 'select2', 'Callback', @callbackFcn2, ...
        'Separator', 'on');
uimenu(cMenu, 'Label', 'select3', 'Callback', @callbackFcn3);
  
```

## uiobjects

uimenu

uicontextmenu

uitoolbar

uipanel

uitabgroup

uitable

uibuttongroup

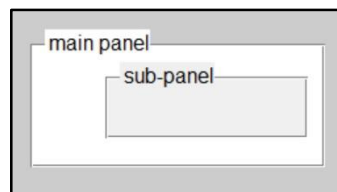
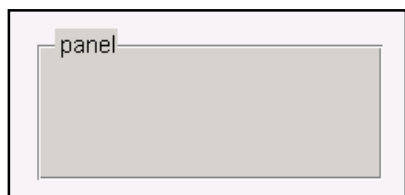
actxcontrol

uicontrol



# Group uiobjects: uipanel

- create panel as a parent to other objects
- objects inside are oriented related to the panel
- many features available (see >> doc `uipanel`)



```
fgHnd = figure;
h1p   = uipanel('Title', 'main panel', ...
               'FontSize', 12, 'BackgroundColor', ...
               'white', 'Position', [0.25 0.25 0.4 0.25]);
h2p   = uipanel('Parent', h1p, ...
               'Title', 'sub-panel', 'FontSize', 12, ...
               'Position', [0.25 0.25 0.7 0.7]);
```

## uiobjects

uimenu

uicontextmenu

uitoolbar

uipanel

uitabgroup

uitable

uibuttongroup

actxcontrol

uicontrol

# Exercise – panel

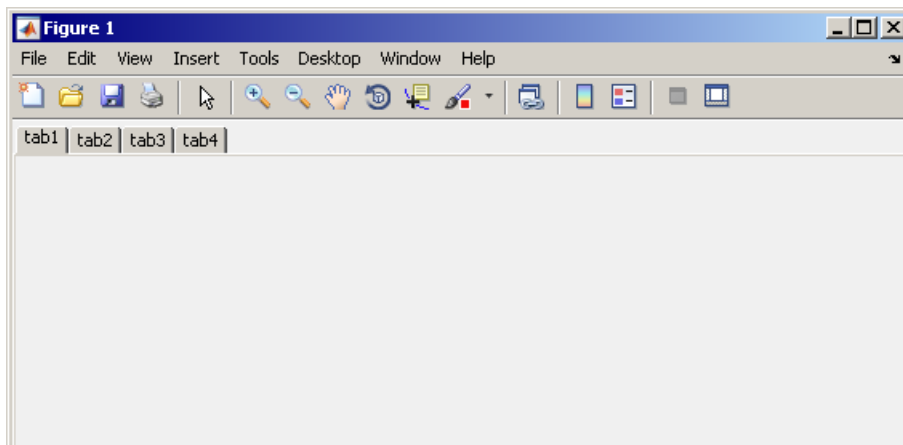
400 s ↑

- create panel and place it to position [90 180 220 60] px
- call the panel „Visibility“, set Tag to „panelVisibility“
- find out its color and store it in a variable which we will be later using to unify colors of other objects within the panel



# Group uiobjects: uitab

- creates a tab that will be parent for other object (same as with panel)
- for more see `>> doc uitabgroup`



```

tabs_gp = uitabgroup();
tabs_1  = uitab(tabs_gp, 'Title', 'tab1');
tabs_2  = uitab(tabs_gp, 'Title', 'tab2');
tabs_3  = uitab(tabs_gp, 'Title', 'tab3');
tabs_4  = uitab(tabs_gp, 'Title', 'tab4');

```

## uiobjects

uimenu

uicontextmenu

uitoolbar

uipanel

**uitabgroup**

uitable

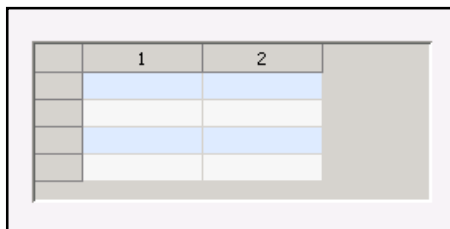
uibuttongroup

actxcontrol

uicontrol

# Group uiobjects: uitable

- creates a 2D table
  - can be placed anywhere in the figure window
  - has a wide range of properties and items (check, popup)
- see >> doc `uitable`



	1	2	3	4	5	6	7	8
1	92	99	1	8	15	67	74	
2	98	80	7	14	16	73	55	
3	4	81	88	20	22	54	56	
4	85	87	19	21	3	60	62	
5	86	93	25	2	9	61	68	
6	17	24	76	83	90	42	49	
7	23	5	82	89	91	48	30	
8	79	6	13	95	97	29	31	
9	10	12	94	96	78	35	37	
10	11	18	100	77	84	36	43	

```
>> figure
>> t = uitable;
>> set(t, 'Data', magic(10));
>> set(t, 'ColumnWidth', {35})
```

## uiobjects

uimenu

uicontextmenu

uitoolbar

uipanel

uitabgroup

uitable

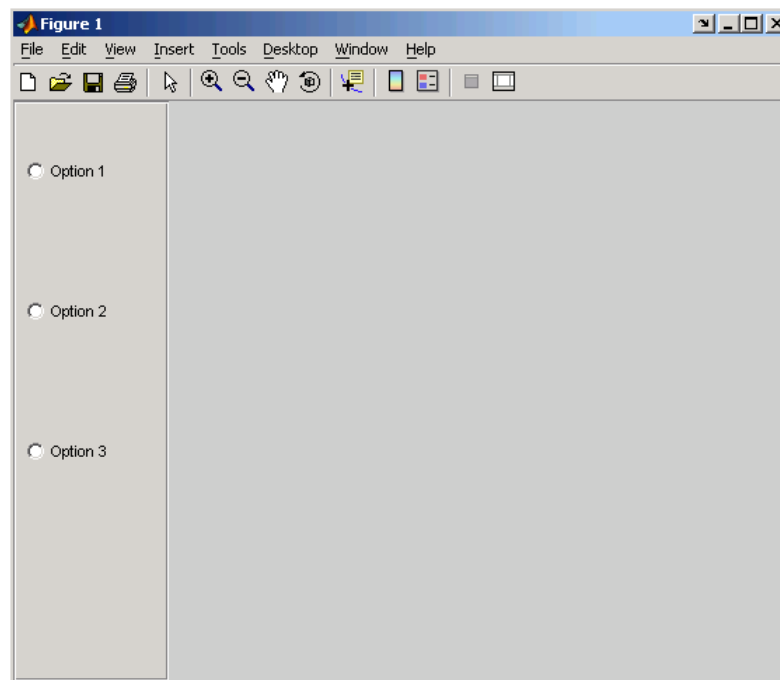
uibuttongroup

actxcontrol

uicontrol

# Group uiobjects: `uibuttongroup`

- block with a group of buttons
- for more see >> doc `uibuttongroup`



## uiobjects

`uimenu``uicontextmenu``uitoolbar``uipanel``uitabgroup``uitable``uibuttongroup``actxcontrol``uicontrol`



# Group uiobjects: actxcontrol

- enables to create Microsoft ActiveX control in the figure window

- list of supported Microsoft ActiveX control

```
>> list = actxcontrollist
>> h     = actxcontrolselect
```

- examples

- web browser

```
>> h = actxcontrol('AcroPDF.PDF.1', ...
```

- PDF reader

```
>> h = actxcontrol('Shell.Explorer.2', ...
```

- for more information see

```
>> doc getting started with COM
```

## uiobjects

uimenu

uicontextmenu

uitoolbar

uipanel

uitabgroup

uitable

uibuttongroup

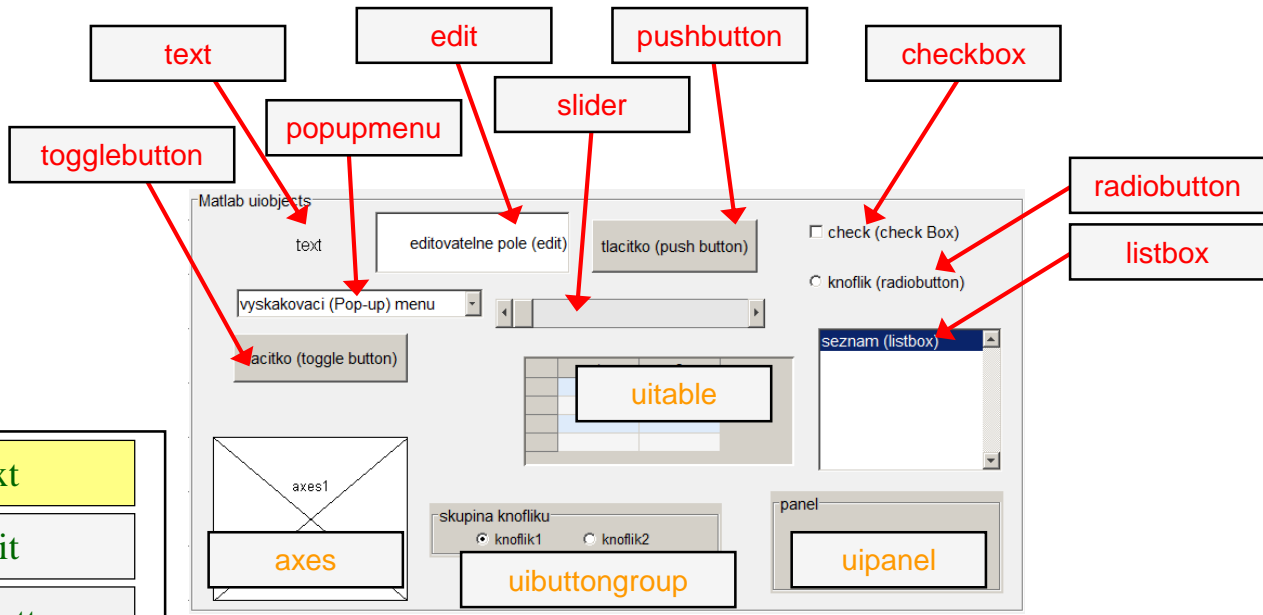
actxcontrol

uicontrol

# Group uiobjects: uicontrol

- uiobjects
  - uimenu
  - uicontextmenu
  - uitoolbar
  - uipanel
  - uitabgroup
  - uitable
  - uibuttongroup
  - actxcontrol
  - uicontrol**

- text**
- edit**
- pushbutton**
- radiobutton**
- checkbox**
- listbox**
- slider**
- popupmenu**
- togglebutton**



# Group uiobjects: `uicontrol`

- `uicontrol` creates basic functional elements of GUI
- to change style of `uicontrol` use property `Style`

```
>> t = uicontrol;  
>> set(t, 'Style', 'text');
```

- to get properties of `uicontrol` use

```
>> get(t);
```

- for more see `>> doc uicontrol`

## uiobjects

`uimenu`

`uicontextmenu`

`uitoolbar`

`uipanel`

`uitabgroup`

`uitable`

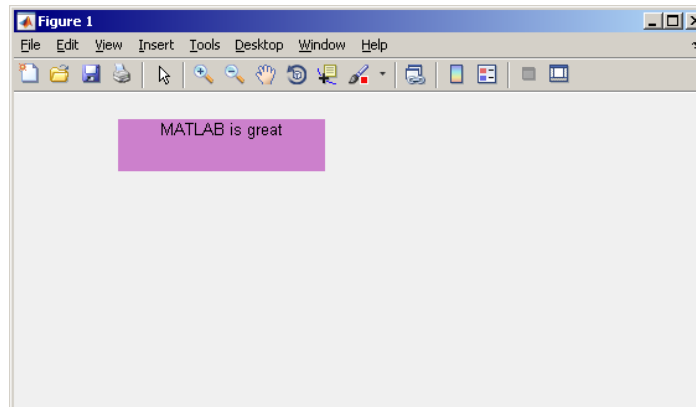
`uibuttongroup`

`actxcontrol`

`uicontrol`

# Group uicontrol: text

- place text at a given spot
- usually used to
  - as a label for other items
  - information text for user



```
>> figure
>> text1 = uicontrol(...
    'Units', 'Normalized', ...
    'Style', 'Text', ...
    'Position', [0.15 0.85 0.3 0.1], ...
    'Tag', 'MTB', ...
    'FontSize', 10, ...
    'BackgroundColor', [0.8 0.5 0.8], ...
    'HorizontalAlignment', 'center', ...
    'String', 'MATLAB is great');
```

uicontrol

text

edit

pushbutton

radiobutton

checkbox

listbox

slider

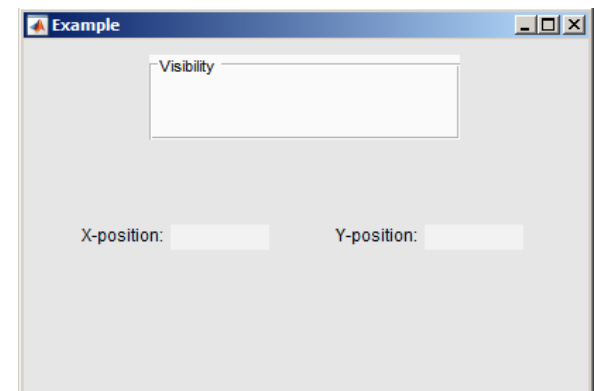
popupmenu

togglebutton

# Exercise – text

400 s ↑

- create four text arrays having following properties that will be placed to following positions (normalized values)
  - [0.1 0.4 0.15 0.075] font 9 figureColor
  - [0.26 0.4 0.175 0.075] font 10 textColor
  - [0.55 0.4 0.15 0.075] font 9 figureColor
  - [0.71 0.4 0.175 0.075] font 10 textColor
- assign labels X-position/Y-position to the arrays with figureColor, others leave without labels
- assign its own handle to each text array

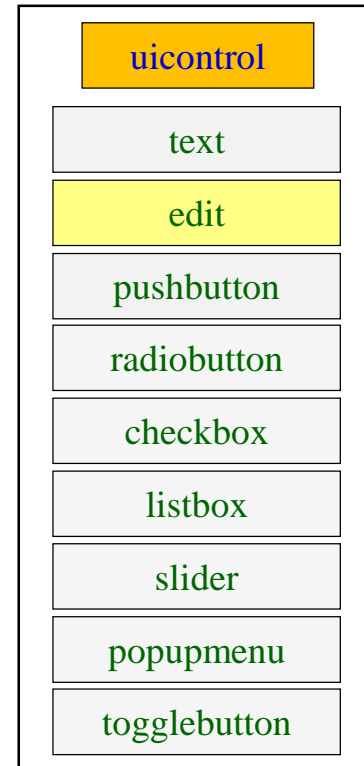
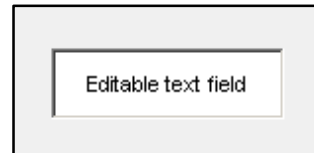


# Exercise – text, solution

---

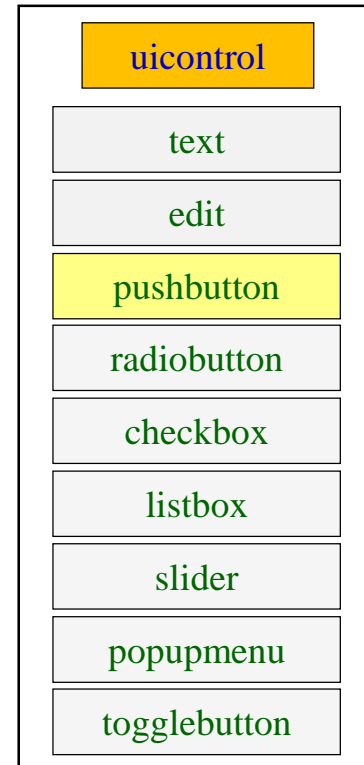
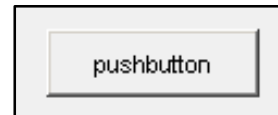
# Group uicontrol: `edit`

- enables to read an array of characters
  - the array of characters is of type `char`
  - the string has to be processed (`str2num`, `str2double`,...)
- CTRL+C,+V,+X,+A,+H shortcuts are available to user
- a console can be created using `edit` in Matlab



# Group uicontrol: pushbutton

- one-state button
- callback function is called on push
- appearance setting is similar to object text

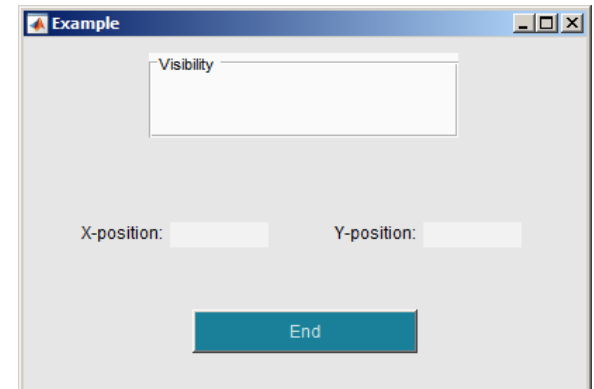




# Exercise – pushbutton

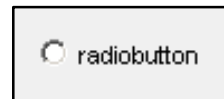
400 s ↑

- create a button with label „End“
  - place it at (normalized) position [0.3 0.1 0.4 0.125]
  - font size set to 9
  - background color: [0.1 0.5 0.6]
  - text color: [0.8 0.9 0.9]

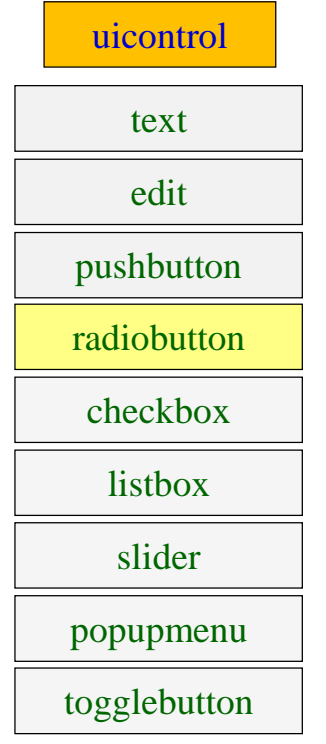
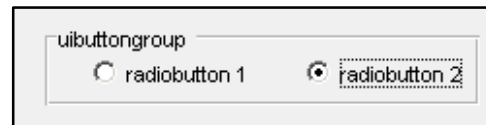


# Group uicontrol: radiobutton

- two-state (on/off)

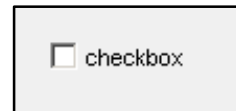


- these elements can be grouped
  - button group (object `uibuttongroup`)
- callback function can detect switching from one radiobutton to other

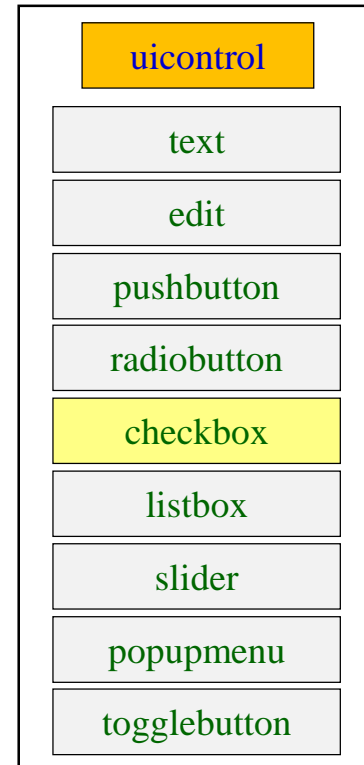


# Group uicontrol: checkbox

- similar to radiobutton
- tick box (with a text attached)
- callback called on state change



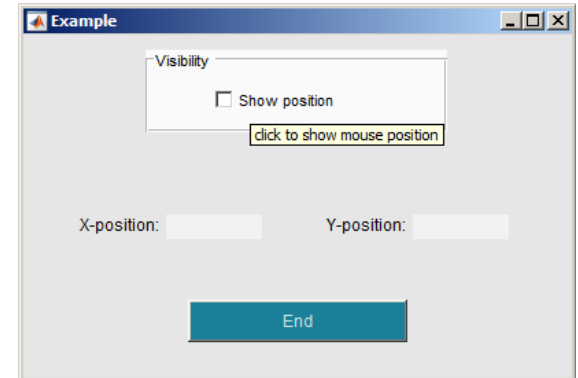
```
function checkboxFcn(hObject) % treated
%% to find out, whether the box is ticked
if hObject.Value % ticked
    % ...
else % not ticked
    % ...
end
```



# Exercise – checkbox

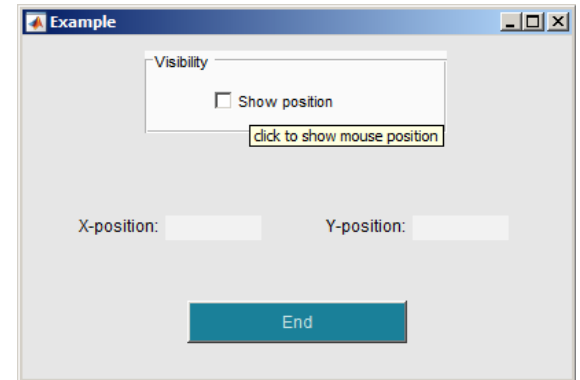
400 s ↑

- create a checkbox placed inside panel `panel1`
- the label is „Show position“
  - make sure to show hint help on mouse cursor close to the checkbox
- assign its own tag to the checkbox
- set the same background color as that of panel



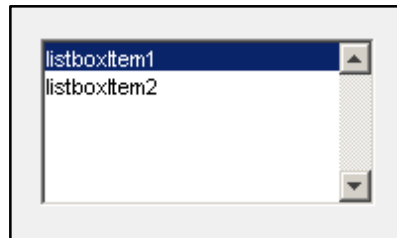
# Exercise

- Save your GUI file for later use (during next lecture)



# Group uicontrol: listbox

- list of items, it is possible to choose one or more items
- property `string` contains list of strings (items)
- property `value` contains matrix of selected items
- values `max` and `min` have impact on multiple selection

**uicontrol**

text

edit

pushbutton

radiobutton

checkbox

**listbox**

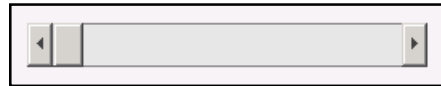
slider

popupmenu

togglebutton

# Group uicontrol: slider

- input value is a numerical range (min and max)
- user moves slider by steps (sliderstep)
- requires
  - range
  - slider step
  - initial value



```
sliderHndl = uicontrol('Style', 'slider')
maxVal = 10;
minVal = 2;
slider_step(1) = 0.4/(maxVal-minVal);
slider_step(2) = 1/(maxVal-minVal);
set(sliderHndl, 'SliderStep', ...
    slider_step, 'Max', maxVal, ...
    'Min', minVal, 'Value', 6.5);
```

uicontrol

text

edit

pushbutton

radiobutton

checkbox

listbox

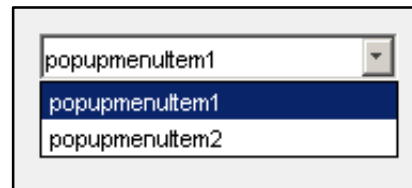
slider

popupmenu

togglebutton

# Group uicontrol: popupmenu

- clicking on arrow displays item list and enables to choose one item
  - string contains list of strings
  - value contains index of the selected item
- more info >> doc `uicontrol`



```
function popupFcn(hObject) % treated
val = get(hObject, 'Value');

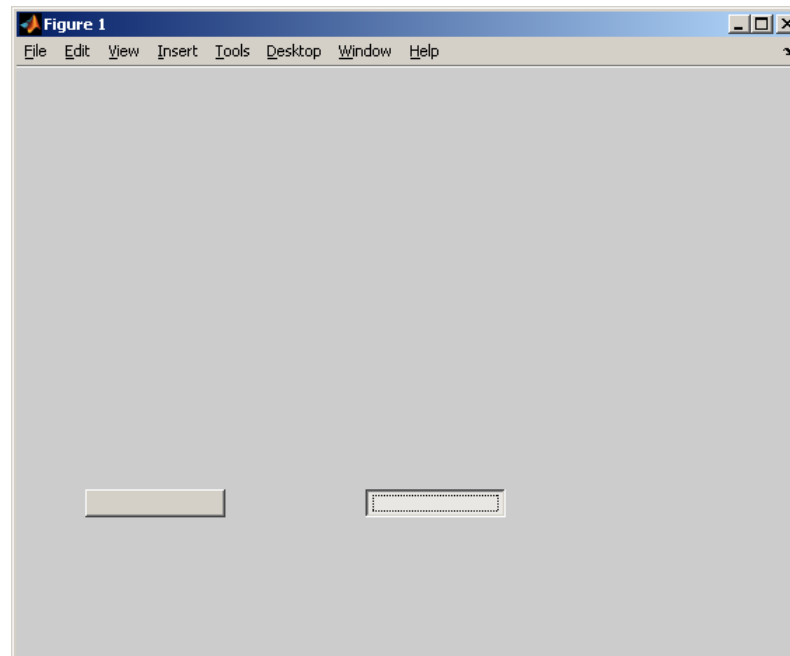
string_list      = get(hObject, 'String');
selected_string  = string_list{val};
% ...
```





# Group uicontrol: togglebutton

- toggle button
  - stays turned on after clicking
- more info >> doc `uicontrol`

`uicontrol``text``edit``pushbutton``radiobutton``checkbox``listbox``slider``popupmenu``togglebutton`

# Discussed functions

---

---

<code>get, set</code>	get or set object's property	●
<code>subplot</code>	placing more graphs in one figure	●
<code>yyaxis, semilogy, semilogx, loglog,</code>	2D graphs with modified axis/axes	●
<code>pie, stairs, contour, quiver</code>	2D graphs	●
<code>image, imagesc</code>	draw matrix as a picture	●
<code>pie3, mesh, slice, scatter</code>	3D graphs	●
<code>colormap</code>	change colormap of a plot	●
<code>view</code>	defines view of 3D graph	●
<code>axis</code>	sets axis range	●

---

# Exercise #1

600 s ↑

- create function with two inputs and one output

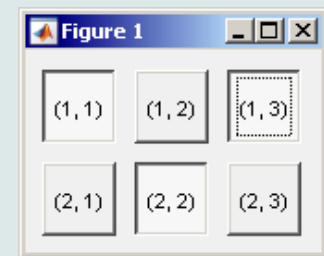
```
function logicState = createToggles(nRows, nColumns)
% function generating GUI with toggle buttons
```

- function creates figure with toggle buttons arranged in matrix nRows x nColumns
- after clicking on toggle buttons and close window function returns matrix of logical values representing state of toggle buttons
- hint: doc `gobjects`

```
>> logicState = createToggles(2, 3)

logicState =

     1     0     1
     0     1     0
```



# Exercise #1 - solution

---

# Thank you!



ver. 6.2 (13/12/2016)  
Miloslav Čapek, Pavel Valtr  
miloslav.capek@fel.cvut.cz

Apart from educational purposes at CTU, this document may be reproduced,  
stored or transmitted only with the prior permission of the authors.  
Document created as part of A0B17MTB course.

