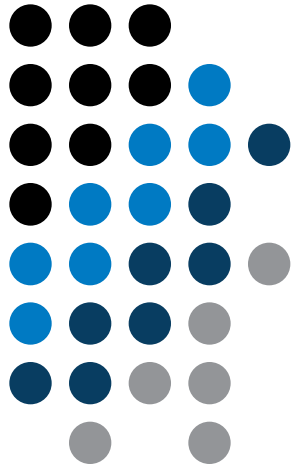
# AE0B17MTB – Matlab

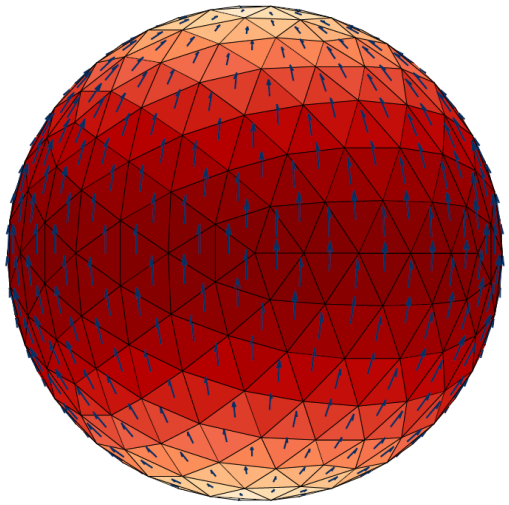
# Part #9


Miloslav Čapek
miloslav.capek@fel.cvut.cz
Viktor Adler, Pavel Valtr, Filip Kozák

Department of Electromagnetic Field
B2-634, 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 `LineSpec` (`doc LineSpec`)
  - 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
  - insering 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;
```

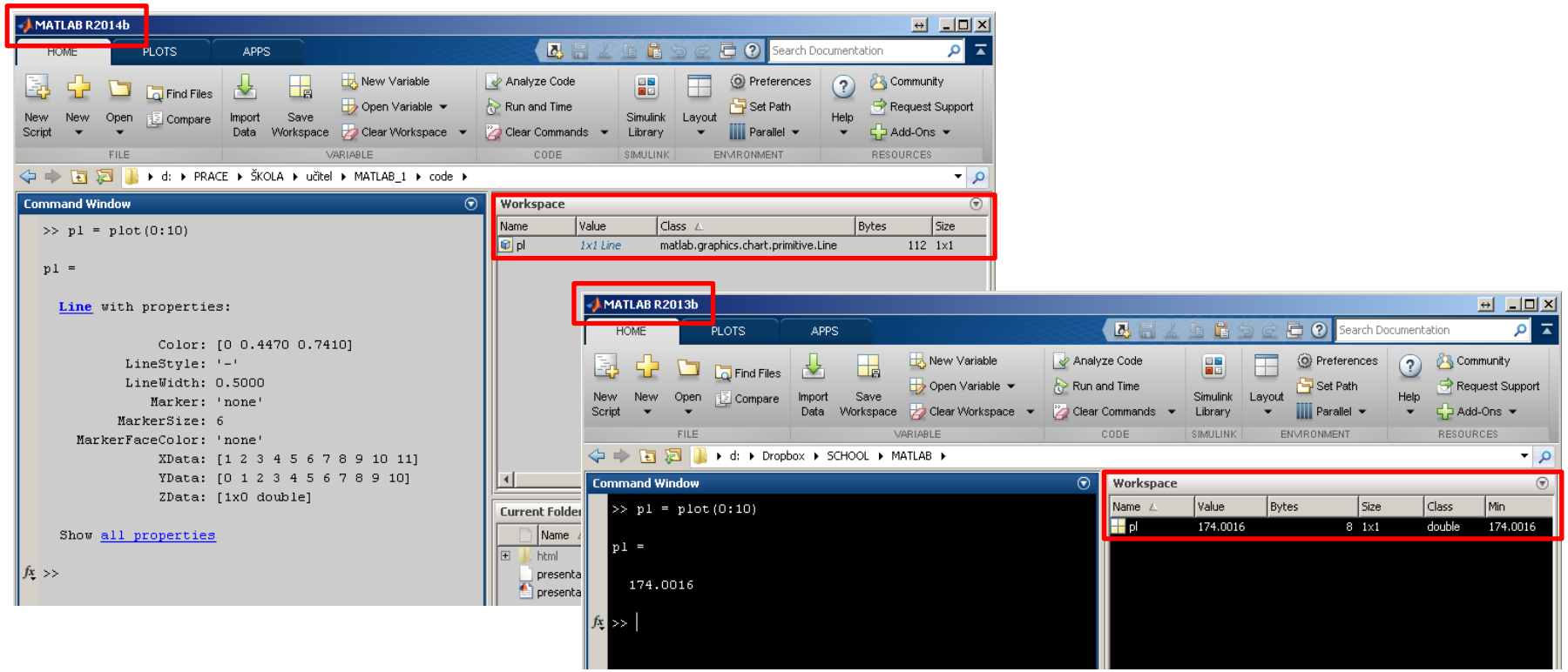
- number stored in `figHandle` variable exists even after closing the window, but it is not a handle any more

# 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)

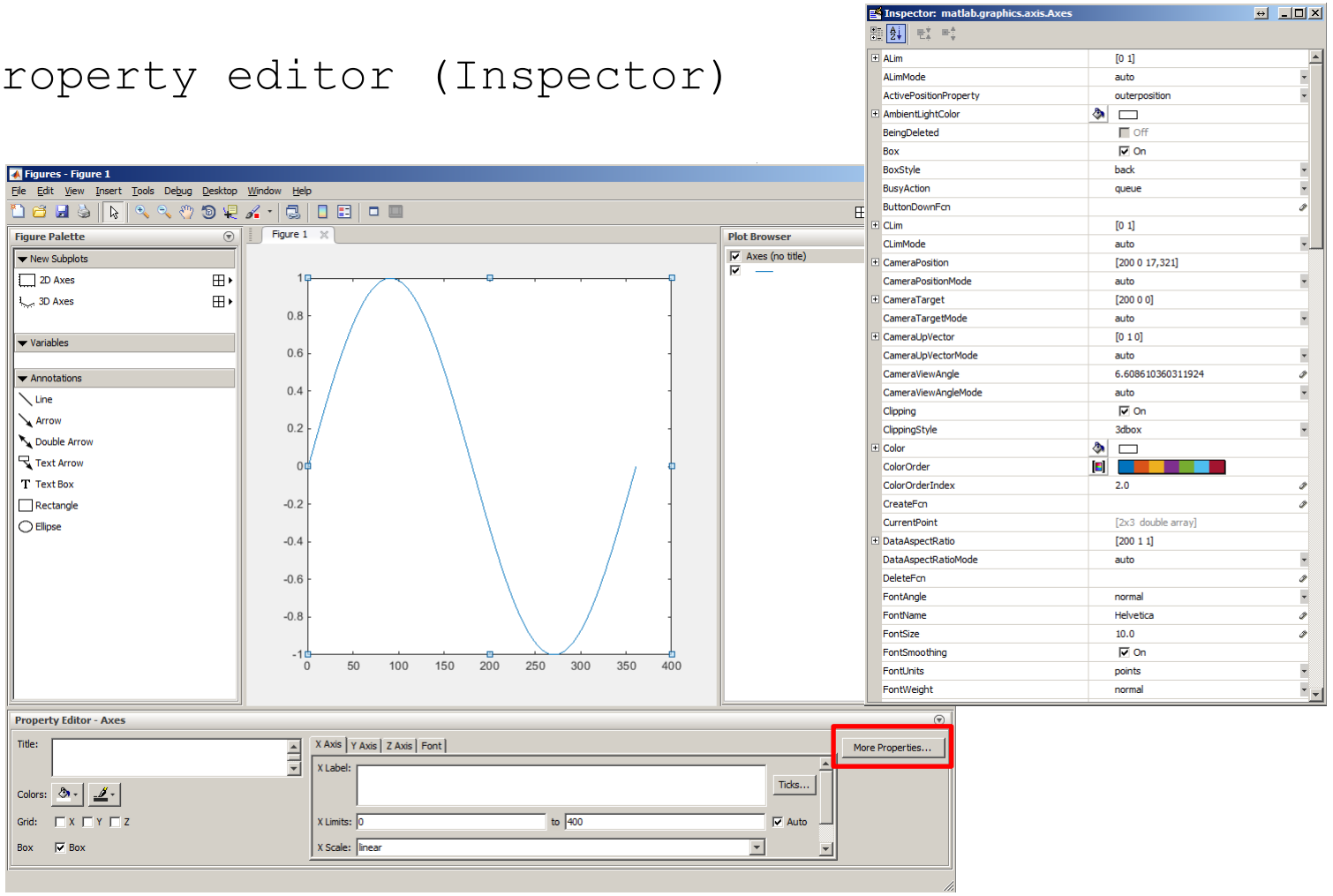
# Advanced visualization in Matlab

- 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



# Advanced visualization in Matlab

- Property editor (Inspector)



# Advanced visualization in Matlab

- the way of setting handle object properties
  - the possibility of using functions `set` a `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
```

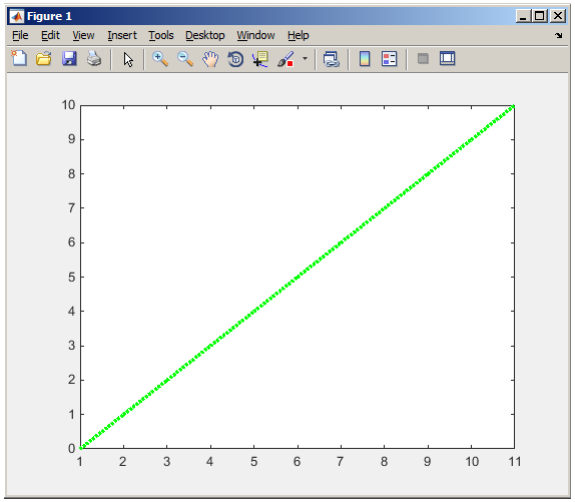
# `get` and `set` functions

60 s ↑

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

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

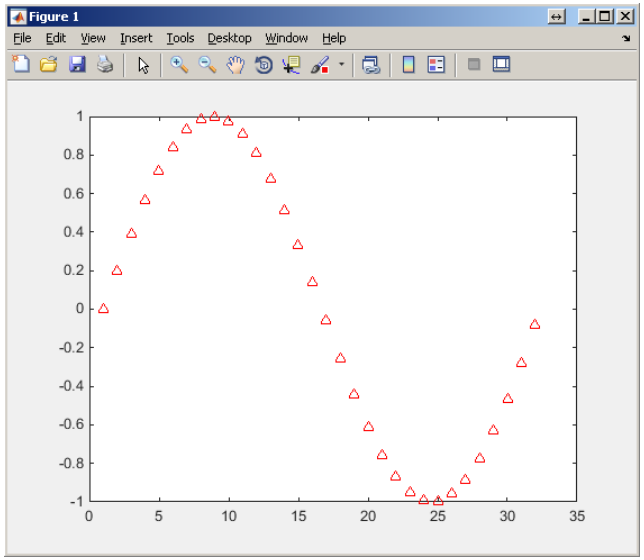
  - find out the thickness of the line and increase it by 1.5
  - set the line color to green
  - set the line style to dotted



# Dot notation application

60 s ↑

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

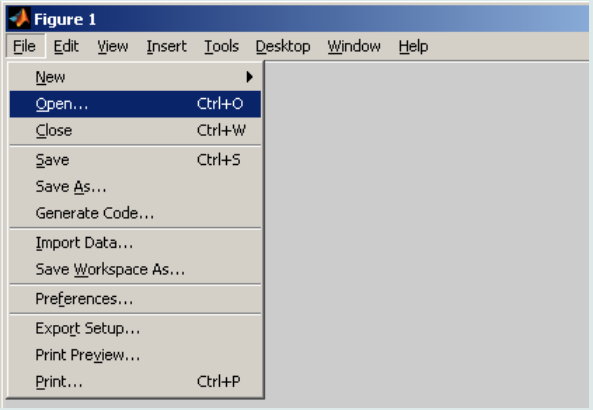


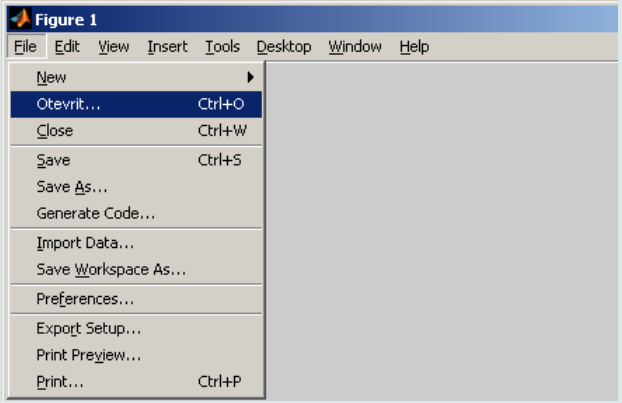

```
myPlotObj = plot(sin(0:0.2:2*pi));
```

# 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 ☺ )

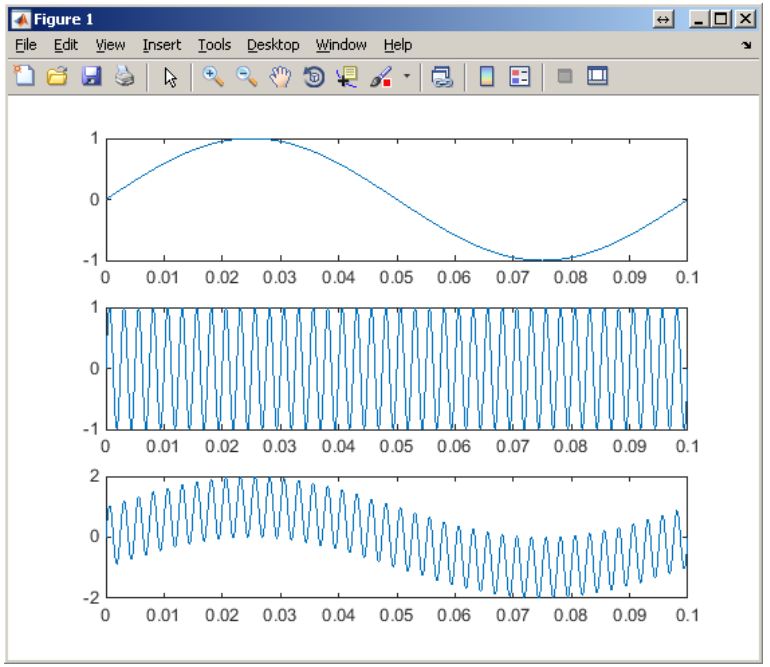
```
fhndl = figure('Toolbar','none');
allFigHndl = guihandles(fhndl);
set(allFigHndl.figMenuOpen,'Label','Otevrit...')
```



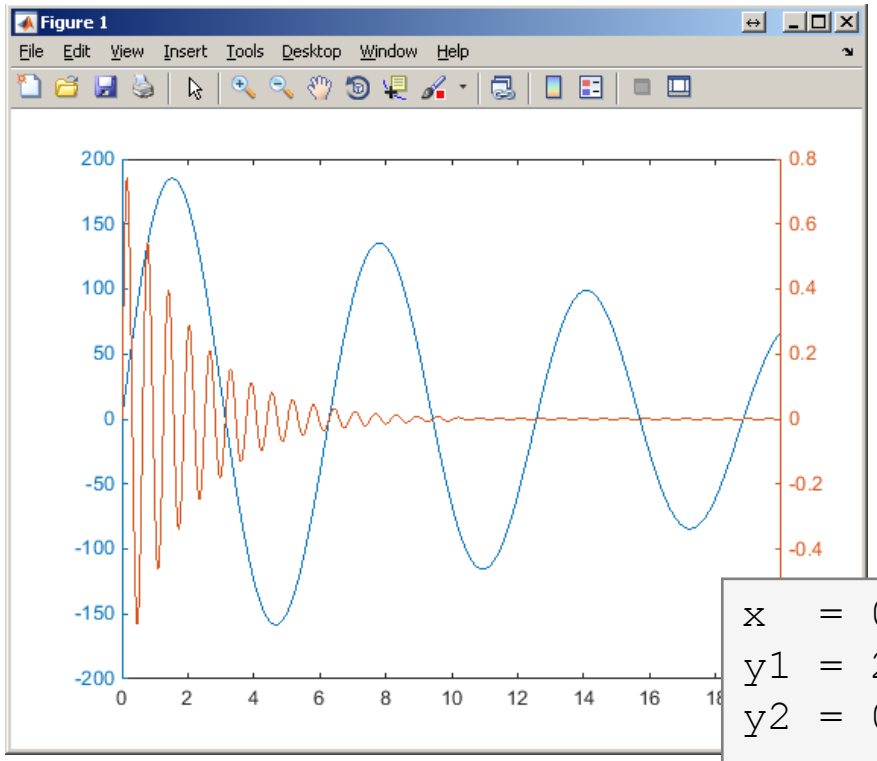
# More graphs in a window – `subplot`

- 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);
```

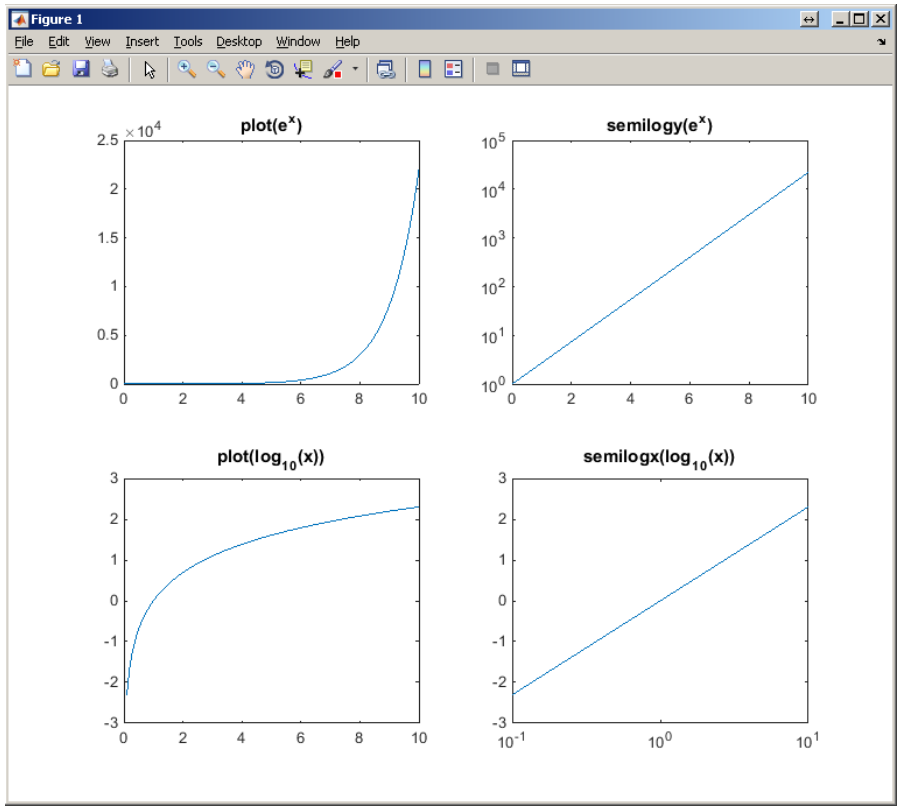
# Double y axis – `plotyy`




```
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');
plotyy(x, y1, x, y2); % old version
% new version:
yyaxis left; plot(x, y1);
yyaxis right; plot(x, y2);
```

# Logarithmic scale

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



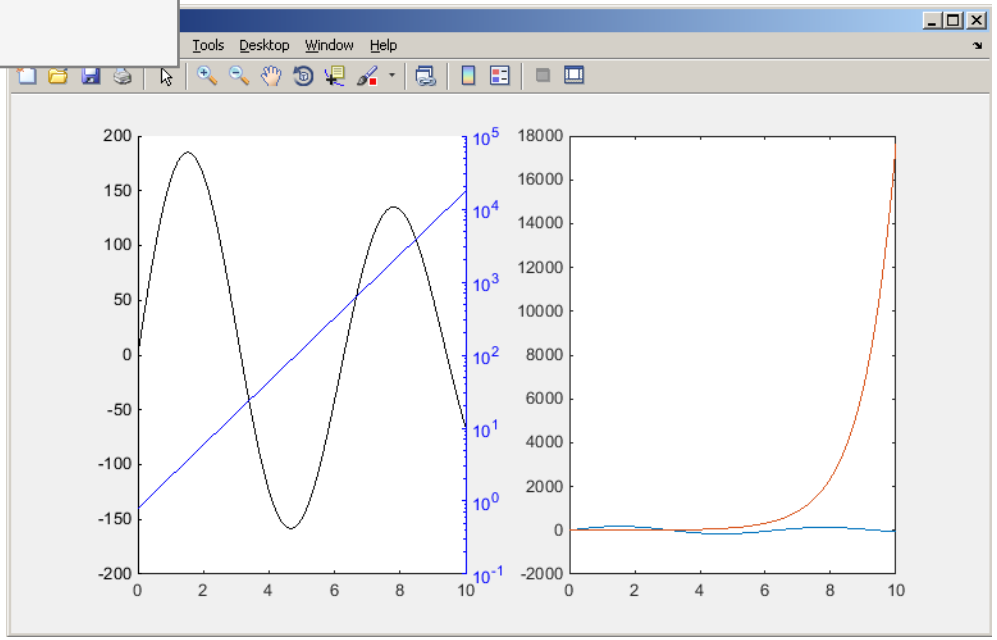

```
x    = 0:0.1:10;
y1   = exp(x);
y2   = log(x);

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

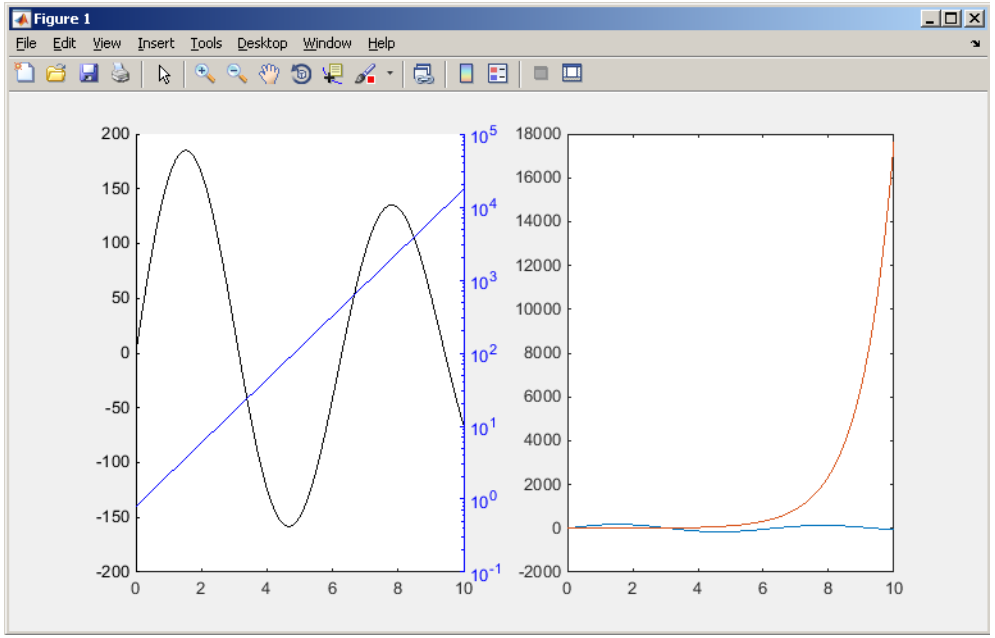
600 s ↑

- compare functions `plot` and `plotyy` in one `figure` object (using `subplot`) for functions shown below
  - in the object created by `plotyy` 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);
```

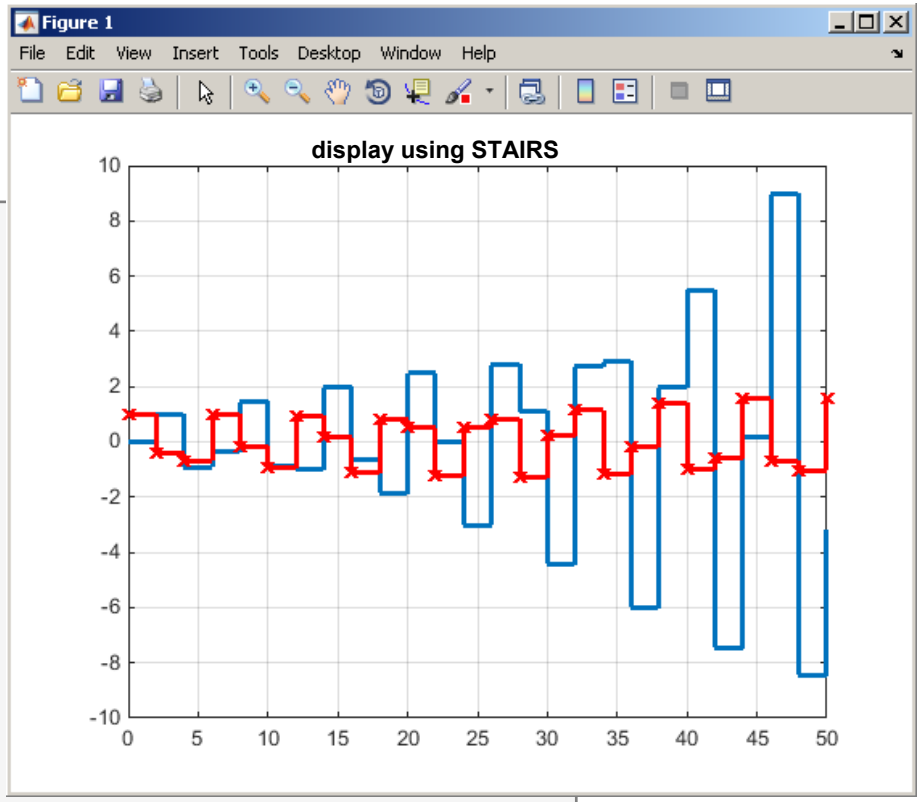


# Example - solution



# stairs

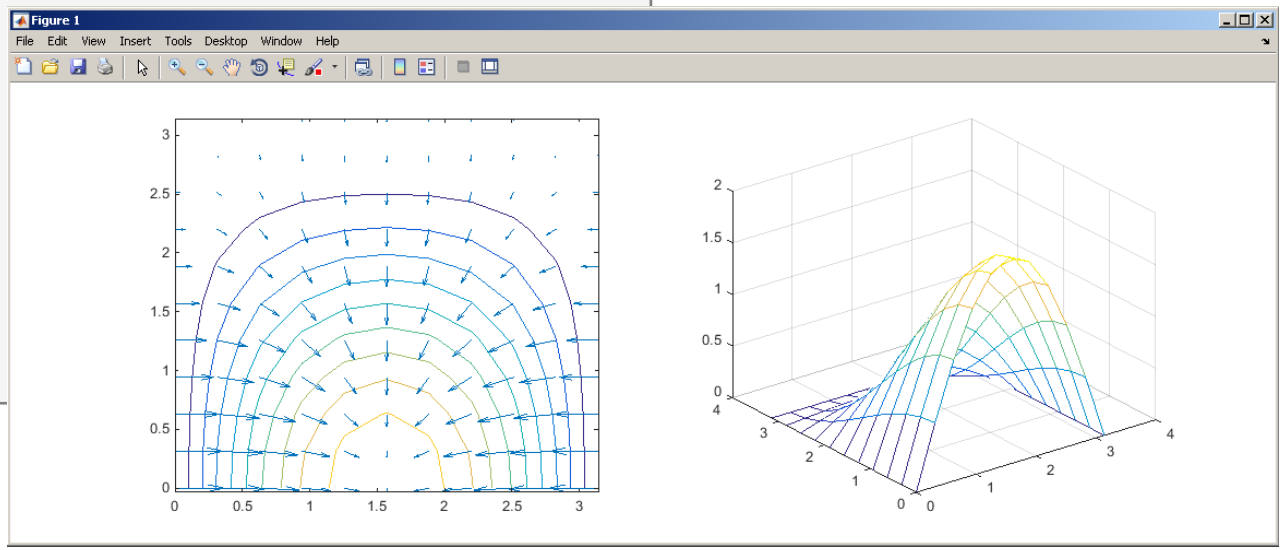
```
x  = 0:2:50;
y1 = exp(0.05*x) .* sin(x);
y2 = exp(0.01*x) .* cos(x);

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

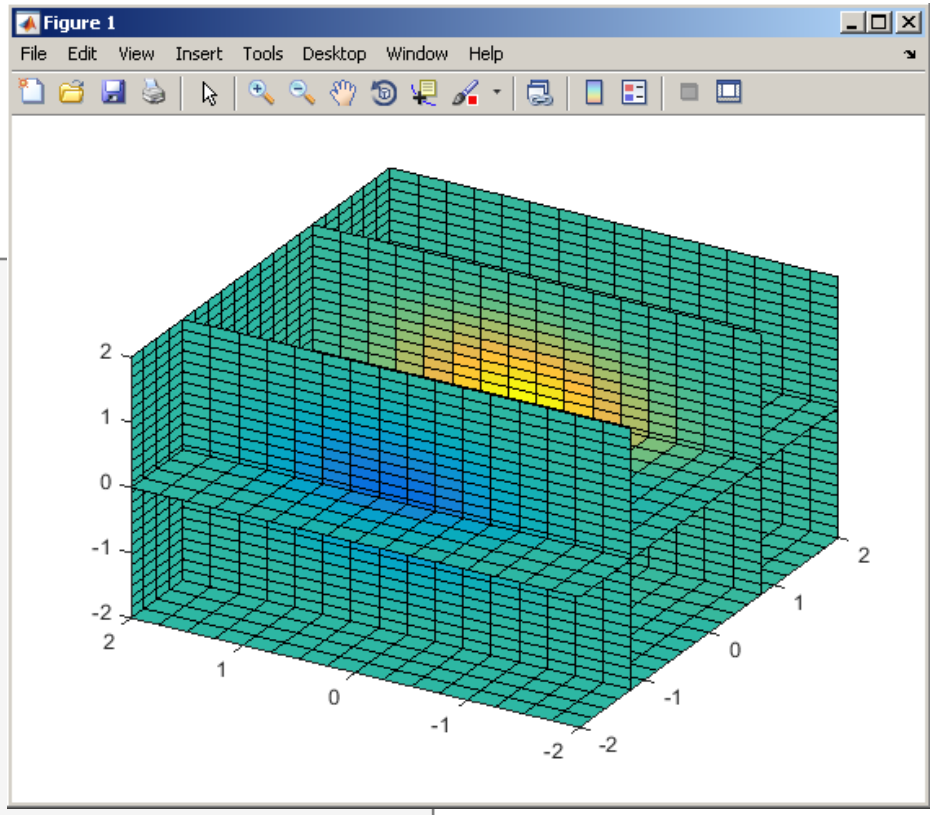
- `contour, quiver, mesh`

```
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`

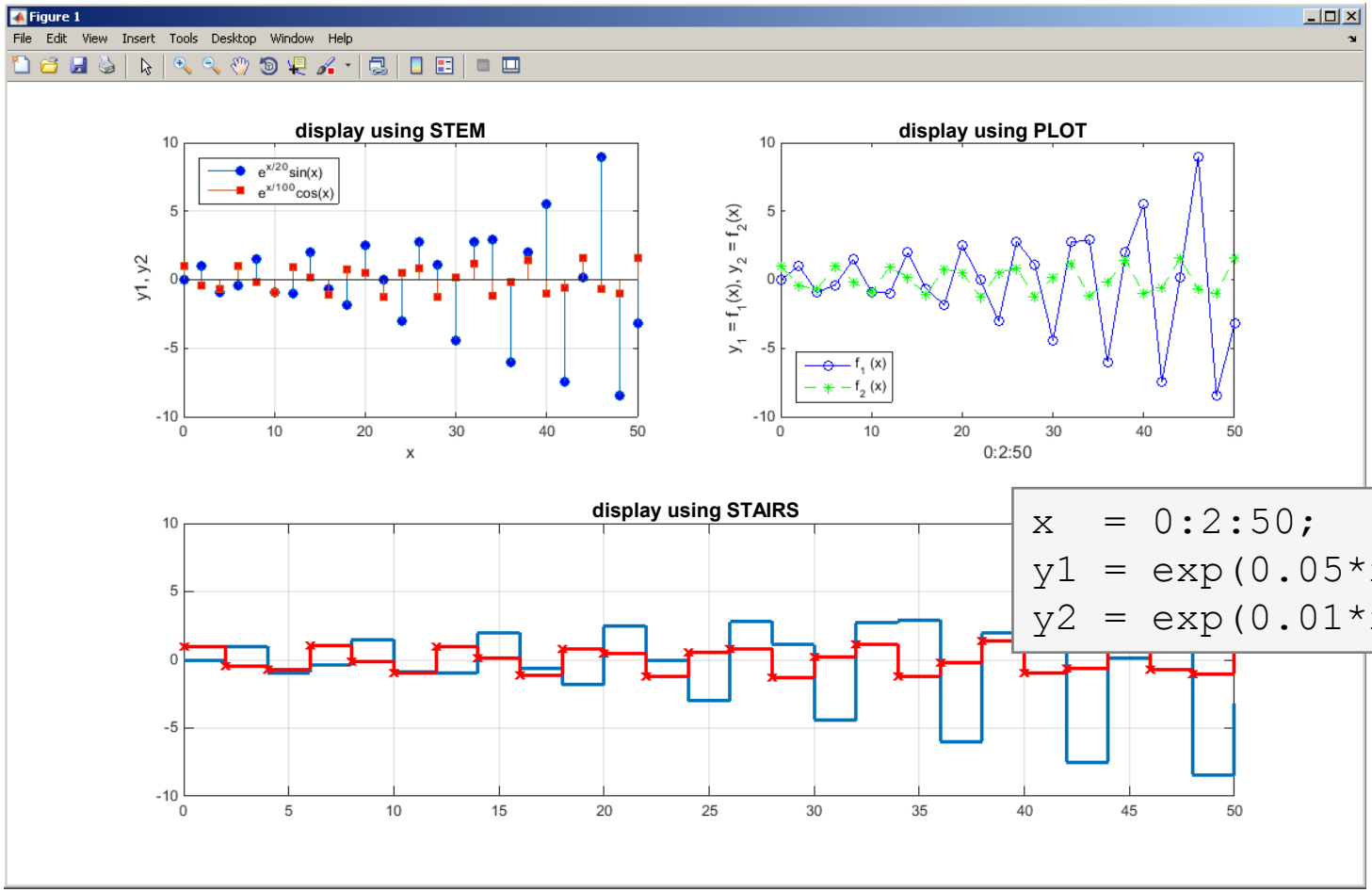



```
[x, y, z] = meshgrid(-2:0.2:2, ...
                     -2:0.25:2, ...
                     -2:0.16:2);

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 `y1` and `y2` are defined as:

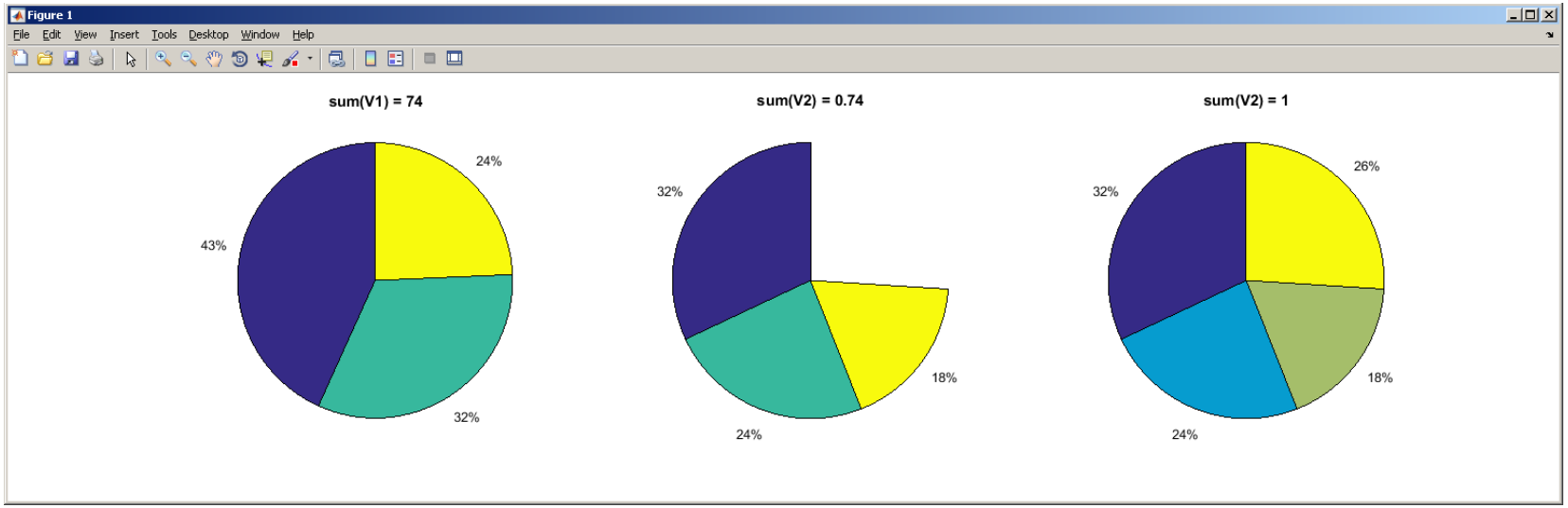



```
x  = 0:2:50;
y1 = exp(0.05*x) .* sin(x);
y2 = exp(0.01*x) .* cos(x);
```

# Exercise #1 solution

# Pie plot – `pie`, `pie3`

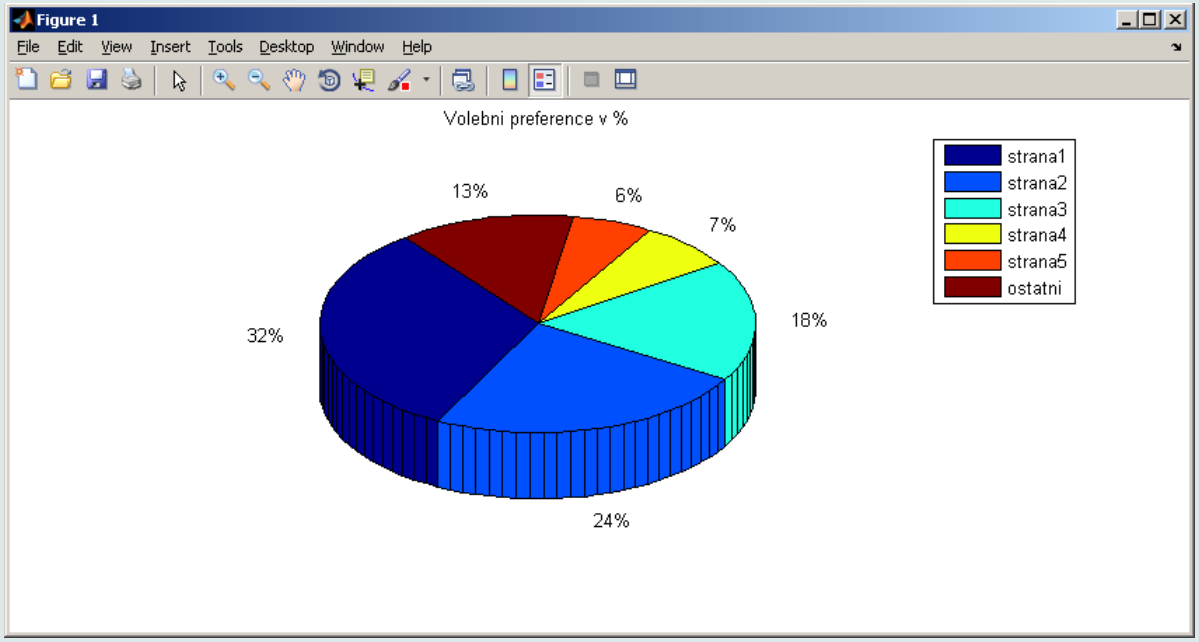
```
V1 = [32 24 18];          % sum(V1) = 74
V2 = V1/100;              % sum(V2) = 0.74
V3 = [V2 1-sum(V2)];      % sum(V3) = 1

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');
```



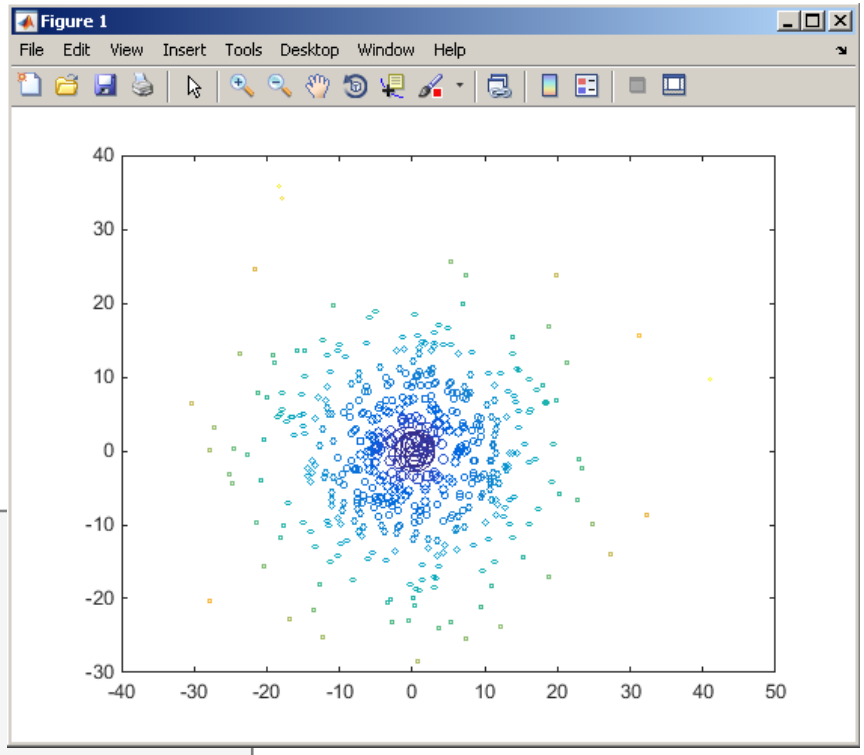
# Exercise

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%



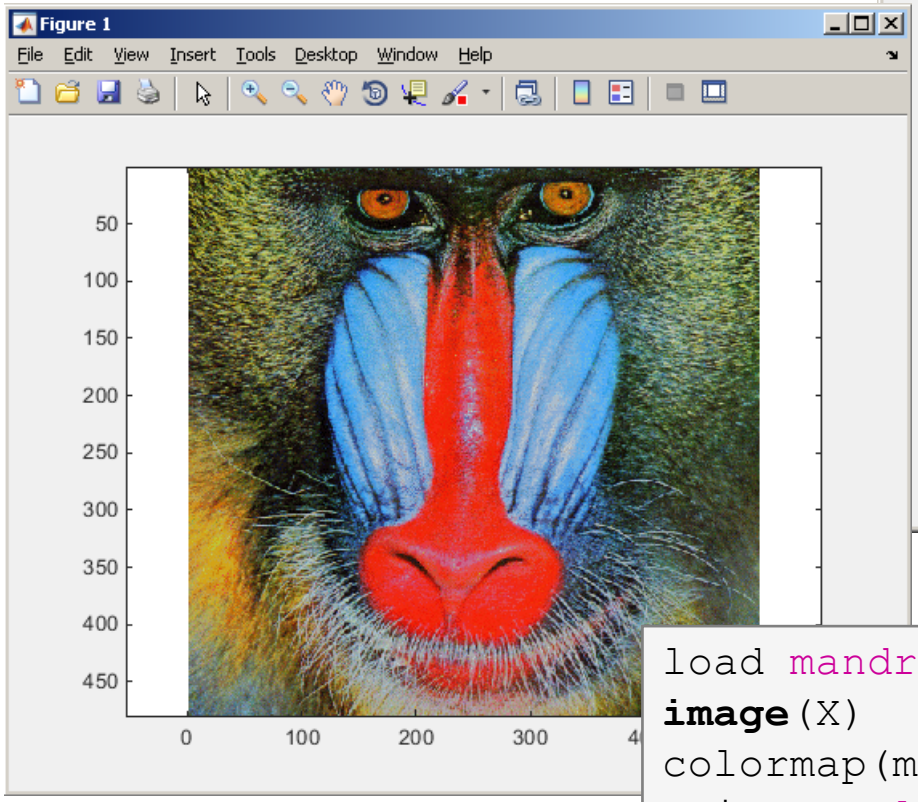
# scatter



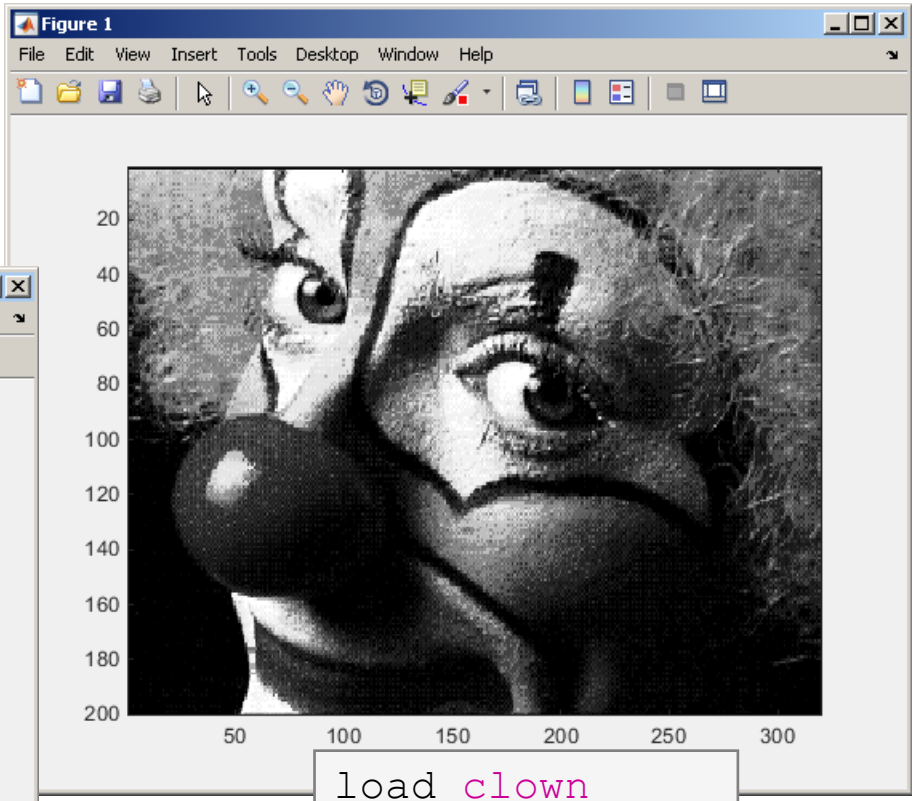

```
x = 10 * randn(500, 1);
y = 10 * randn(500, 1);
c = hypot(x, y);

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

# Picture depiction

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



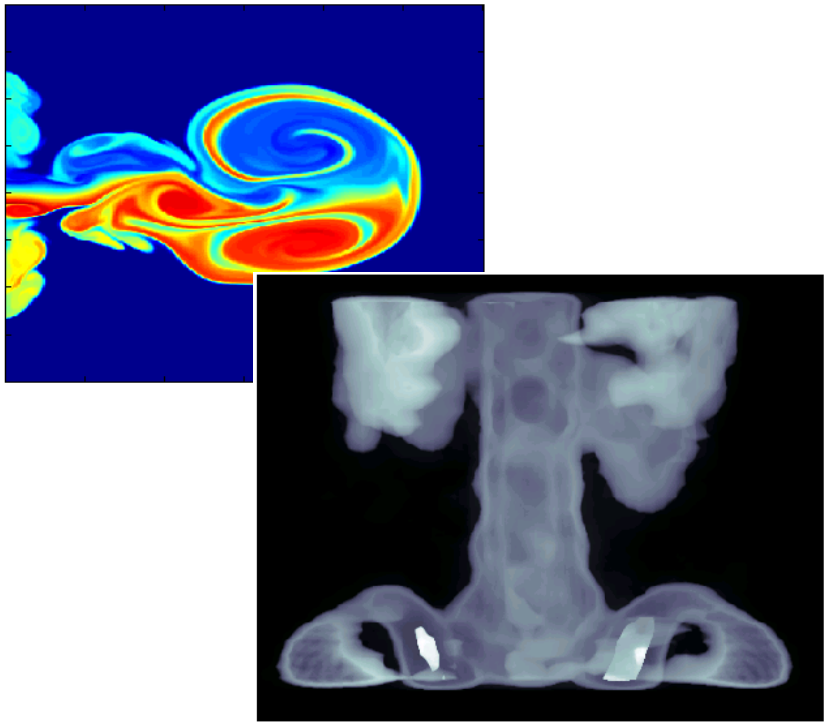

```
load mandrill
image(X)
colormap(map)
axis equal
```




```
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`

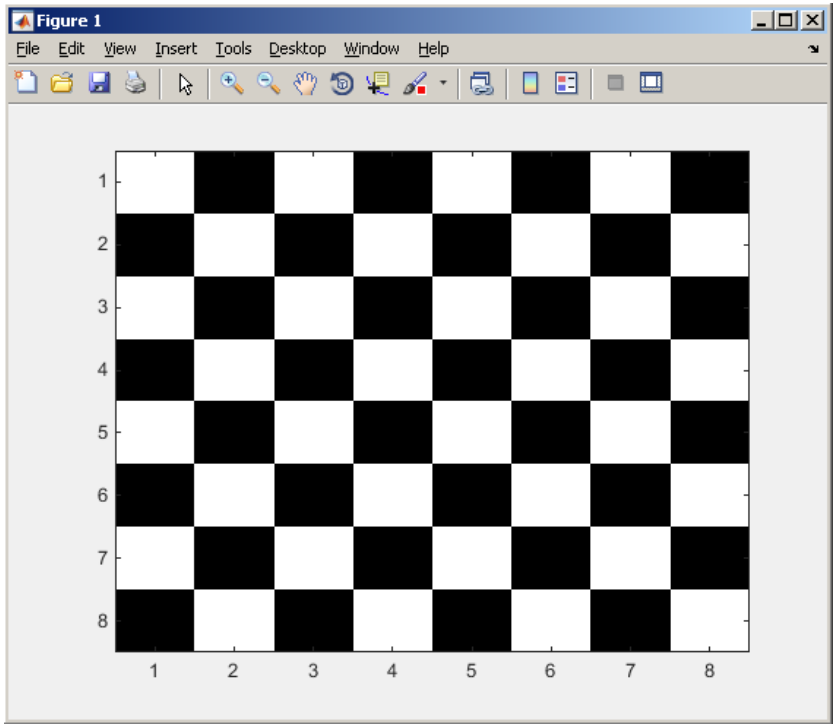


| Colormap Name | Color Scale |
|---|---|
| parula | |
| jet | |
| hsv | |
| hot | |
| cool | |
| spring | |
| summer | |
| autumn | |
| winter | |
| gray | |
| bone | |
| copper | |
| pink | |
| lines | |
| colorcube | |
| prism | |
| flag | |
| white | |

# Exercise

600 s ↑

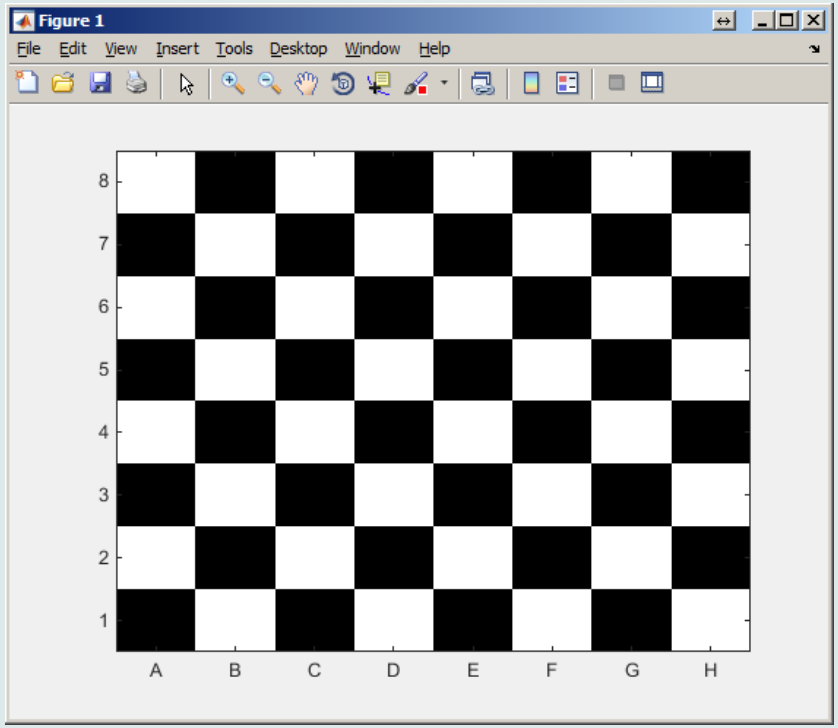
- create a chessboard as shown in the figure:
  - the picture can be drawn using the function `imagesc`
  - consider `colormap` setting



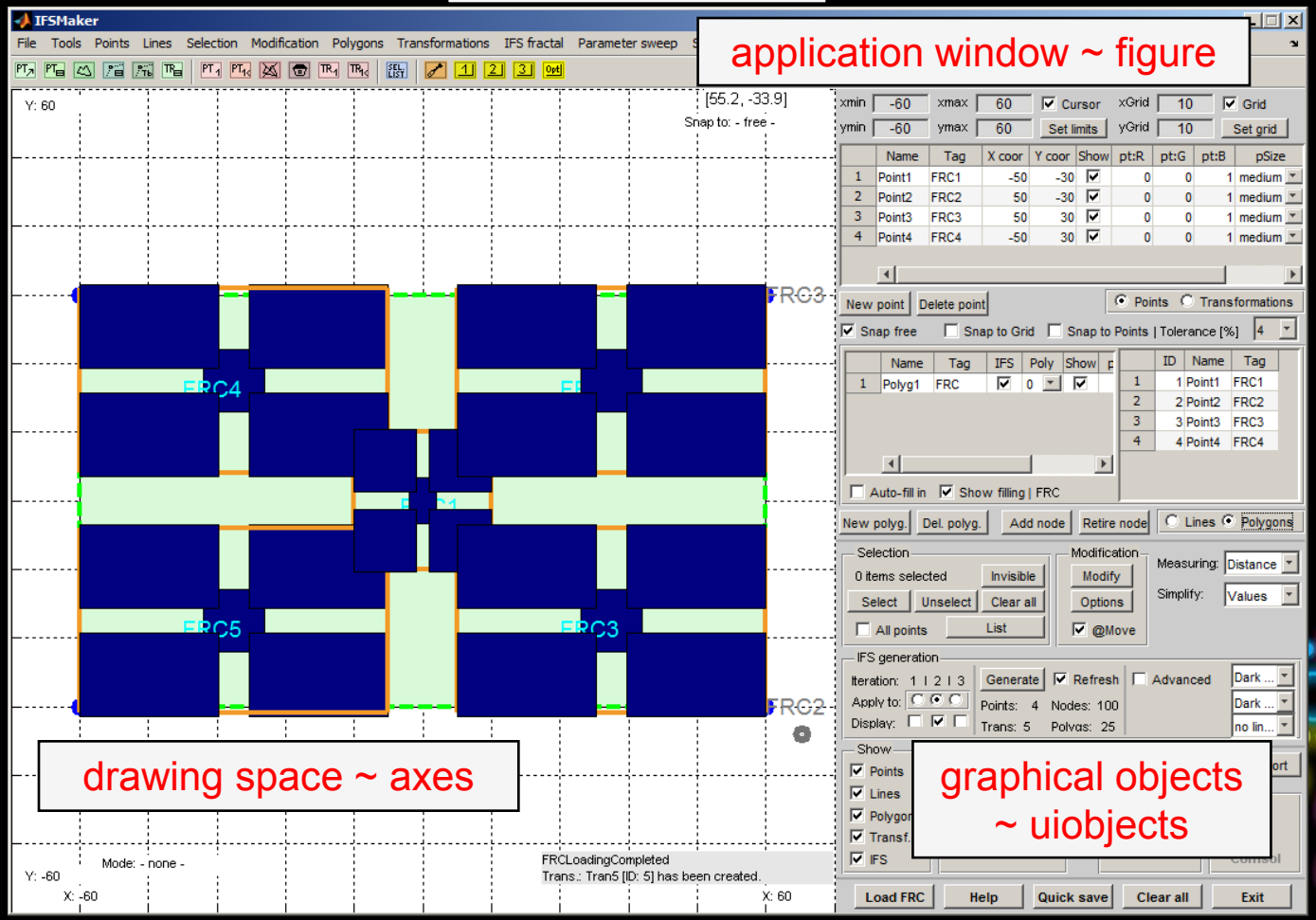
# Exercise

600 s ↑

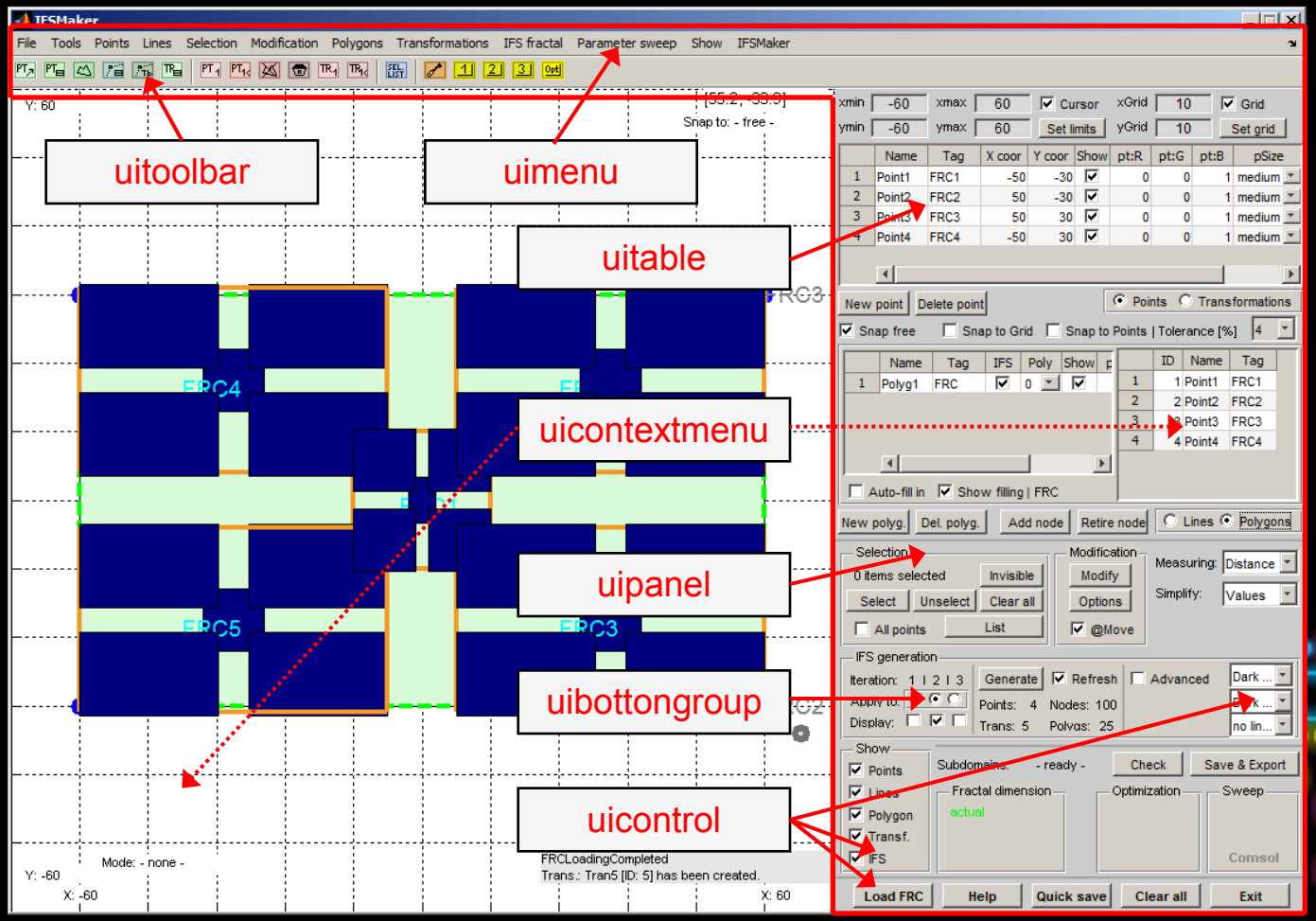
- Modify the axes of the chessboard so that it corresponded to reality :



# Structure of GUI #1

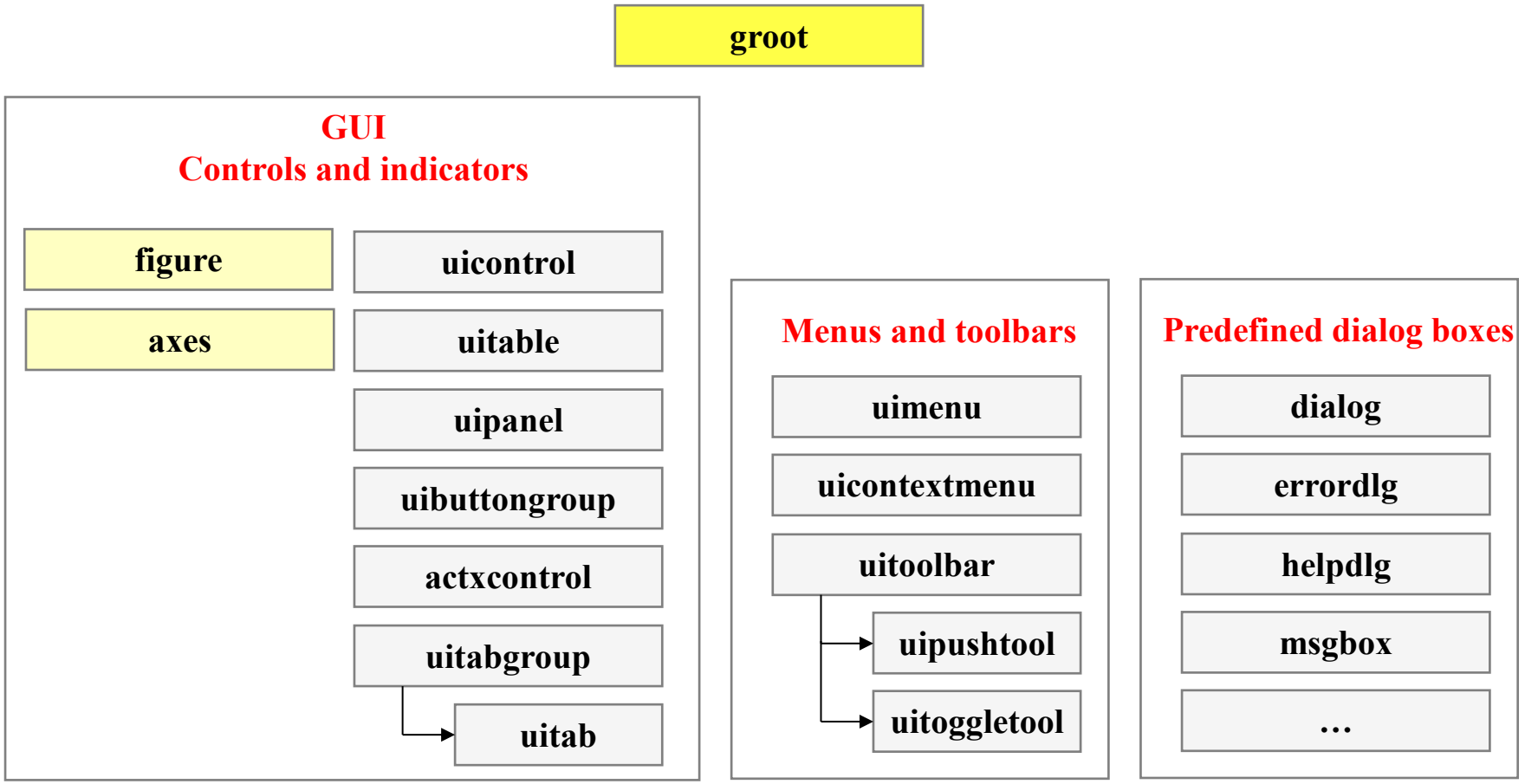


# Structure of GUI #2



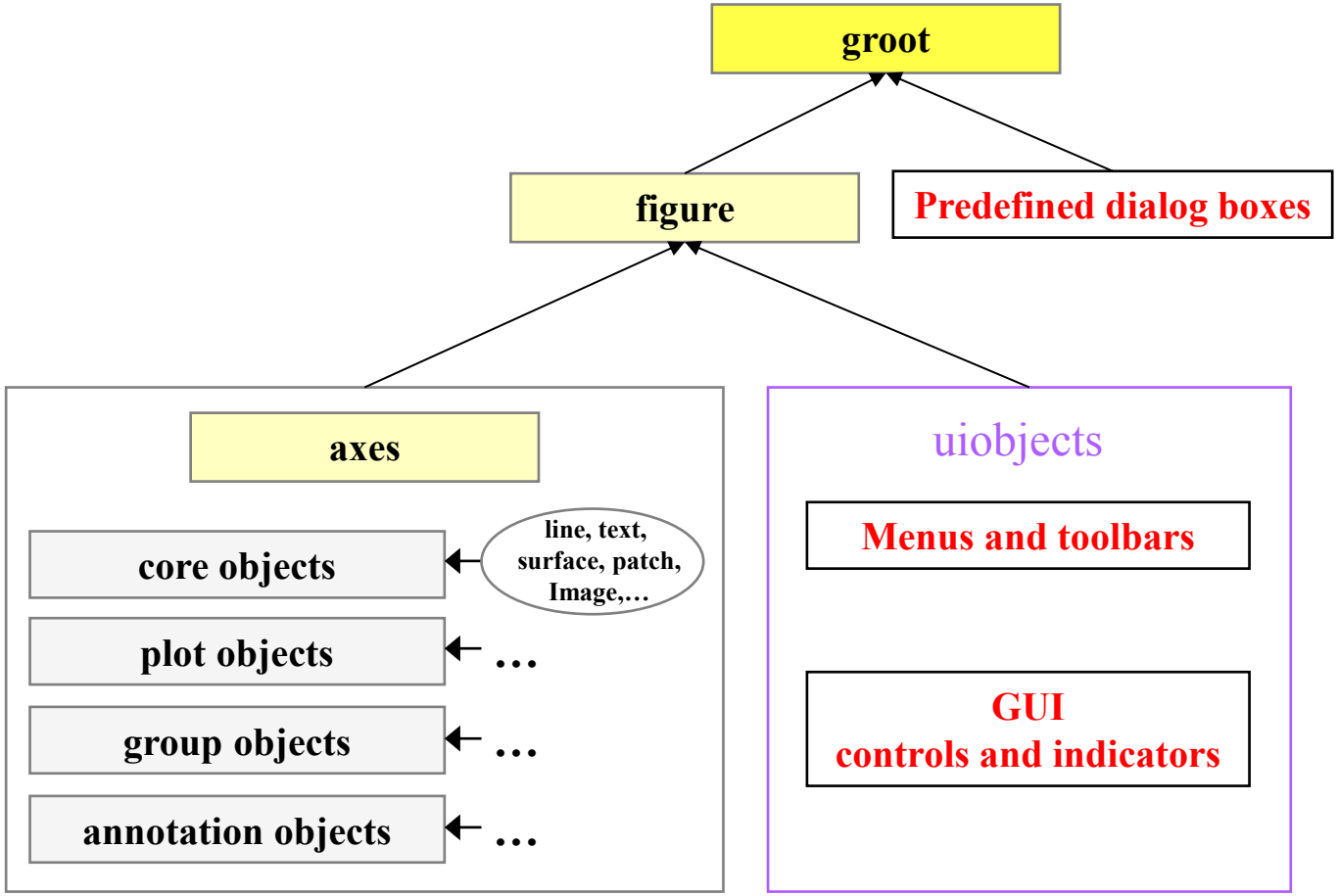
# Structure of GUI

- objects are sorted in a logical way

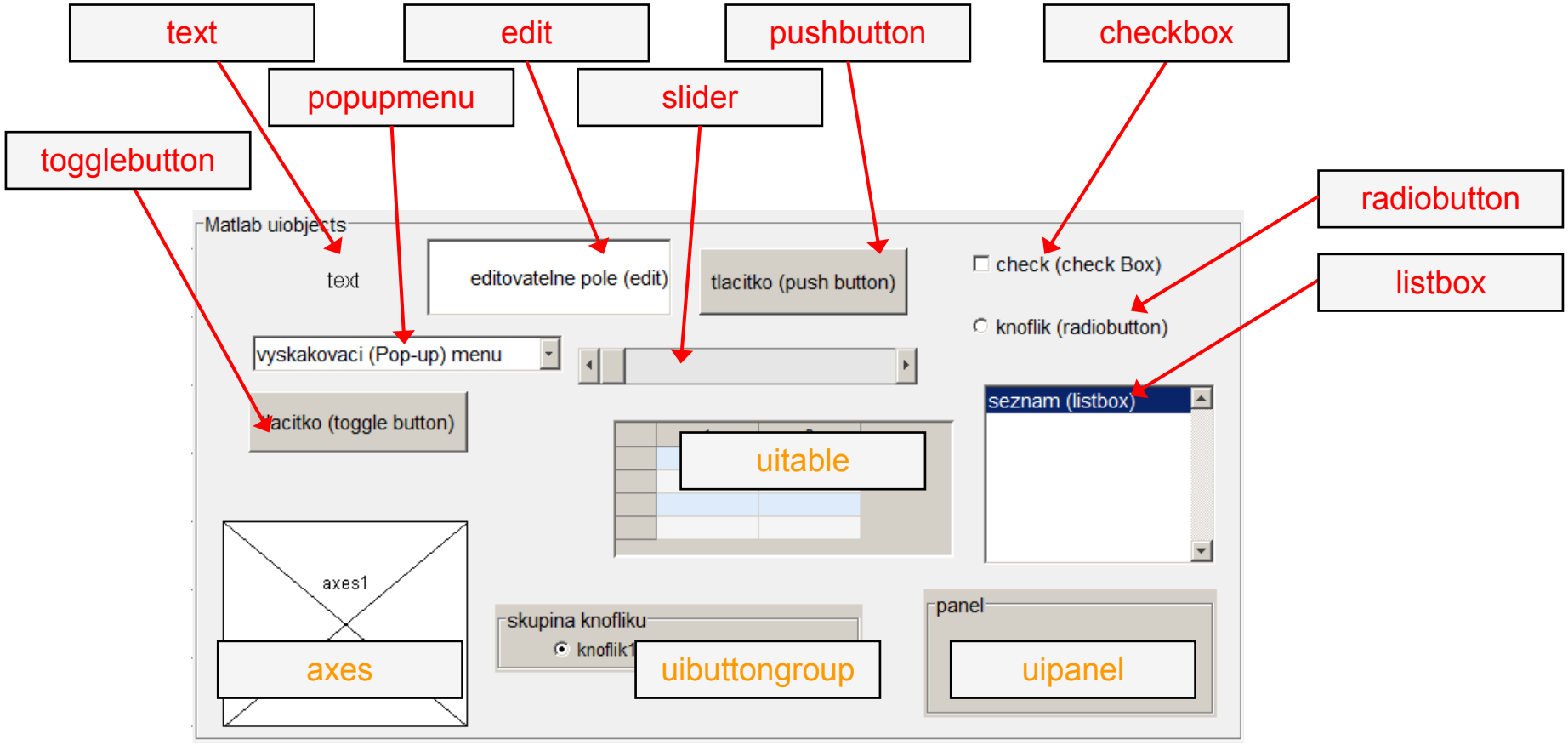


# Structure of GUI

- object hierarchy



# Structure of GUI #3



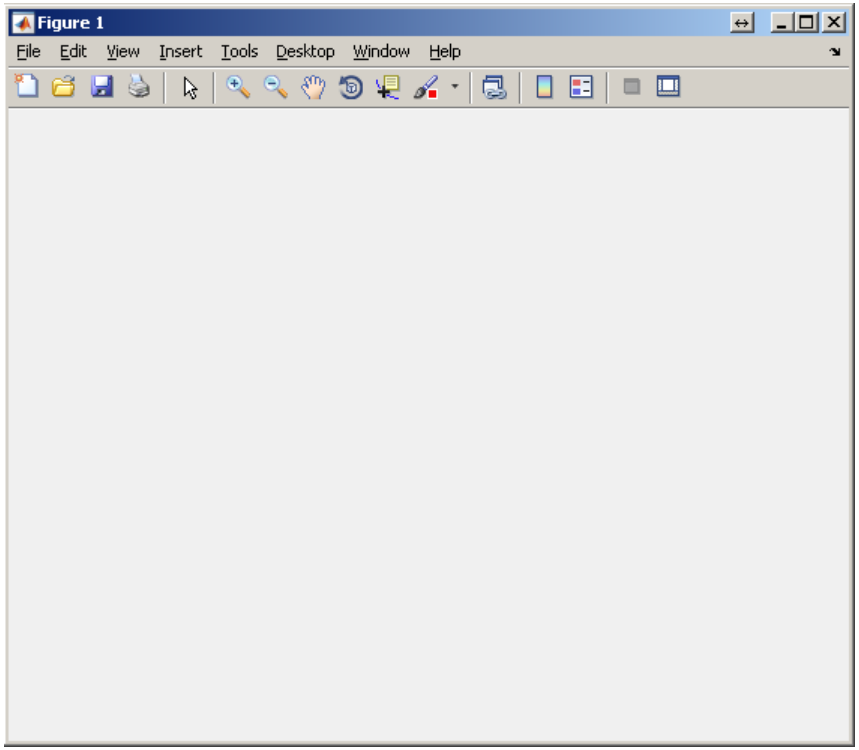
# Screen properties, `groot`

- 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`



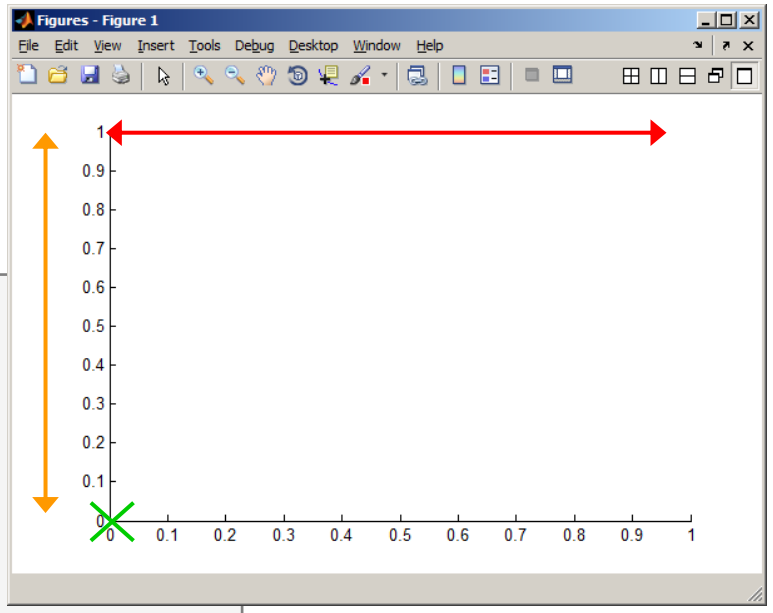
# Position property

- 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]
```

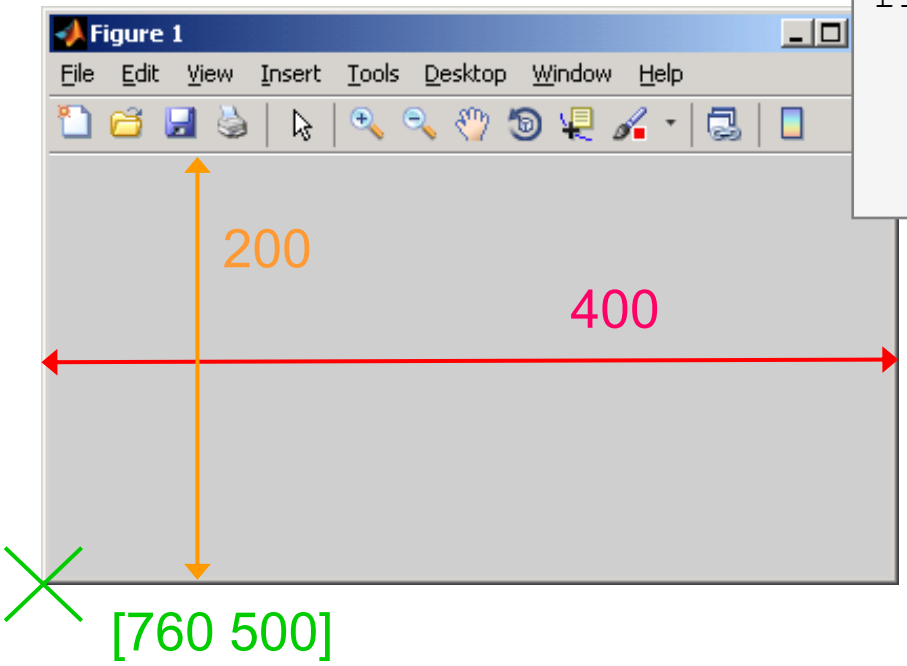



```
%% A)
uicontrol('Units','pixels',...
          'Style','pushbutton',...
          'Position',[50 150 75 25]);
%% B)
uicontrol('Units','normalized',...
          'Style','pushbutton ',...
          'Position',[0.05 0.12 0.1 0.05]);
```

# Figure creation

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

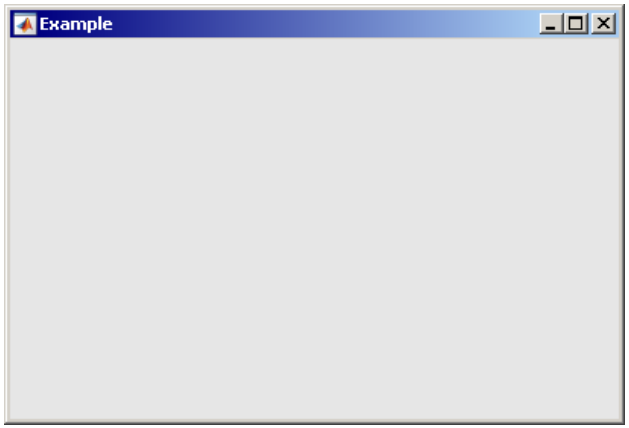
```
dispSize = get(0, 'ScreenSize');
figSize = [400 200];
figHndl = figure('pos', ...
     [(dispSize(3)-figSize(1))/2 ...
      (dispSize(4)-figSize(2))/2 ...
       figSize(1)  figSize(2)]);
```



# Exercise – GUI window creation

400 s ↑

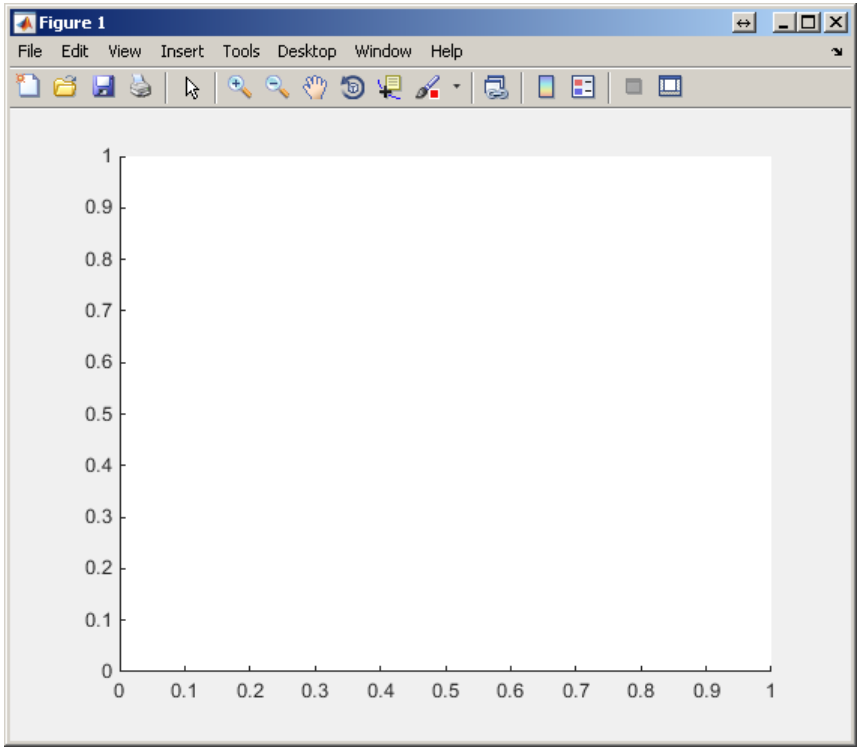
- in a new script that we will be extending throughout todays lecture create a `figure` window that opens in the center of the screen having width of 400 pixels and height of 250 pixels
  - make sure the figure's name is „Example“ and the title `figure 1` doesn’t display
  - use `Tag` property for naming (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)`

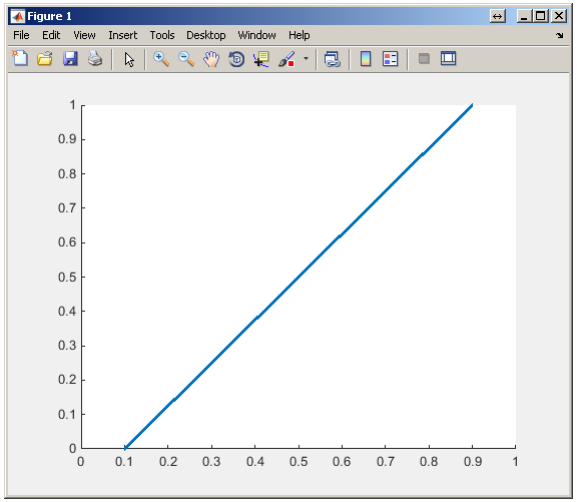
  or
  - `properties(axes)`

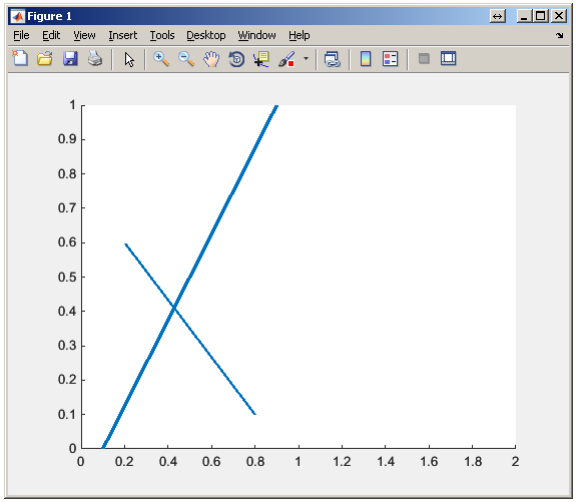


# Function `axis`

- `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])
```



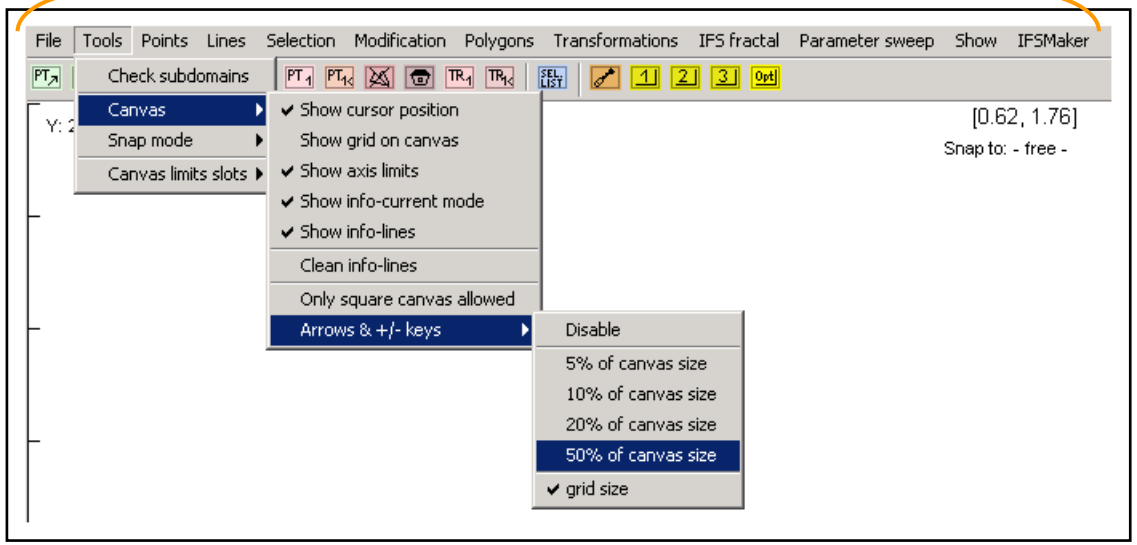




```
line([0.8 0.2], [0.1 0.6], 'LineWidth', 2)
axis([0 2 0 1])
```

# 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



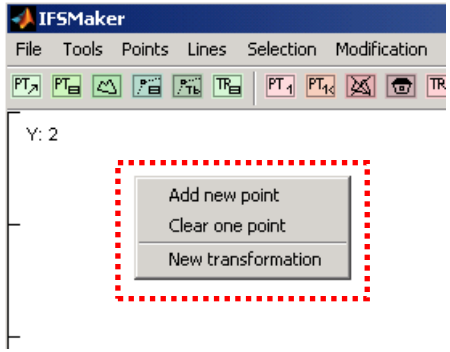

- for more see `help uimenu`

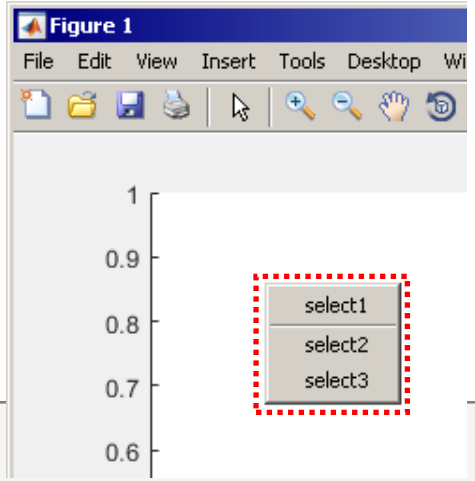
uiobjects

- 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



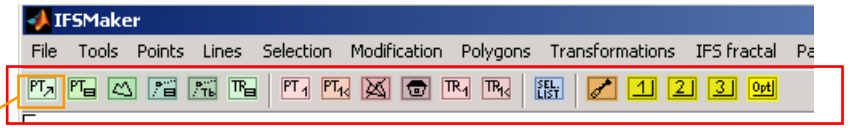




uiobjects
- uimenu
- **uicontextmenu**
- uitoolbar
- uipanel
- uitabgroup
- uitable
- uibuttongroup
- actxcontrol
- uicontrol

```
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);
```

# Group uiobjects: `uitoolbar`

- it is possible to create own menu icons in Matlab
  - not complicated but out of scope of this course
  - for those interested see >> `doc uitoolbar`


icon ‘drawn‘ in m-file

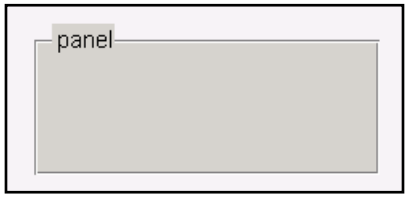
```
pics = [1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1;...
        1 .6 .6 .6 .6 .6 .6 .6 .6 .6 .6 .6 .6 .6 .6 .6 .6 .6 .6 1;...
        1 .6  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1 .6 1;...
        1 .6  1  0  0  0  1  0  0  0  0  0  1  1  1  1  1  1 .6 1;...
        1 .6  1  0  1  1  0  1  1  0  1  1  1  1  1  1  1  1 .6 1;...
        1 .6  1  0  1  1  0  1  1  0  1  1  1  1  1  1  1  1 .6 1;...
        1 .6  1  0  0  0  1  1  1  0  1  1  1  1  1  1  1  1 .6 1;...
        1 .6  1  0  1  1  1  1  1  0  1  1  1  0  0  0  0  1 .6 1;...
        1 .6  1  0  1  1  1  1  1  0  1  1  1  1  1  0  0  1 .6 1;...
        1 .6  1  0  1  1  1  1  1  0  1  1  1  1  1  0  1  0  1 .6 1;...
        1 .6  1  1  1  1  1  1  1  1  1  1  1  1  0  1  1  0  1 .6 1;...
        1 .6  1  1  1  1  1  1  1  1  1  1  1  0  1  1  1  1  1 .6 1;...
        1 .6  1  1  1  1  1  1  1  1  1  1  0  1  1  1  1  1  1 .6 1;...
        1 .6  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1 .6 1;...
        1 .6 .6 .6 .6 .6 .6 .6 .6 .6 .6 .6 .6 .6 .6 .6 .6 .6 .6 1;...
        1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1 1];
icon(:,:,1) = .85. * pics;
icon(:,:,2) = .98. * pics;
icon(:,:,3) = .85. * pics;
```

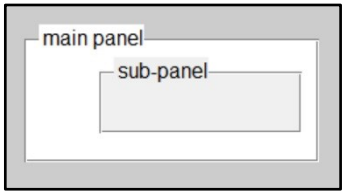
- way of icon placement
  - >> `doc uipushtool`
  - >> `doc uitoggletool`

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]);
```

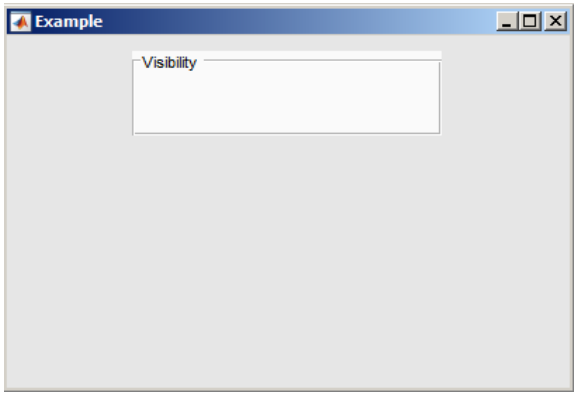
uiobjects

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

# Exercise – `panel`

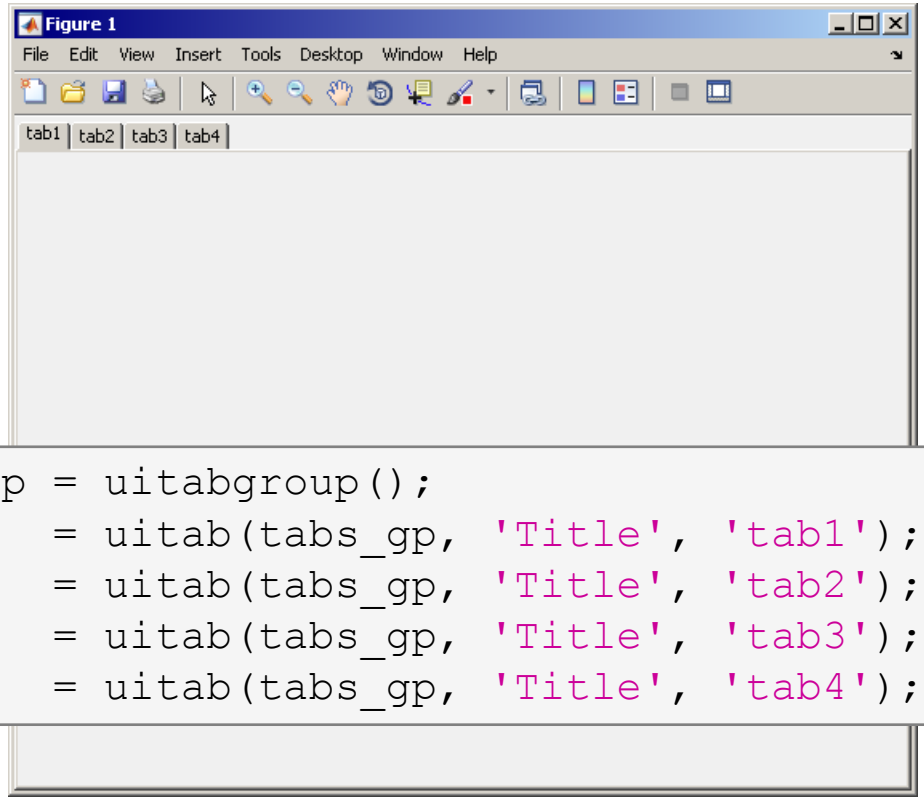
400 s ↑

- 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`



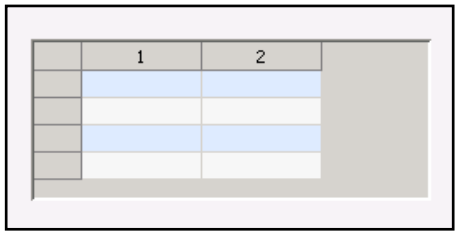

```
tabs_gp = uitabgroup();
tabs_1  = uitab(tabs_gp, 'Title', 'tab1');
tabs_2  = uitab(tabs_gp, 'Title', 'tab2');
tabs_3  = uitab(tabs_gp, 'Title', 'tab3');
tabs_4  = uitab(tabs_gp, 'Title', 'tab4');
```

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 | |

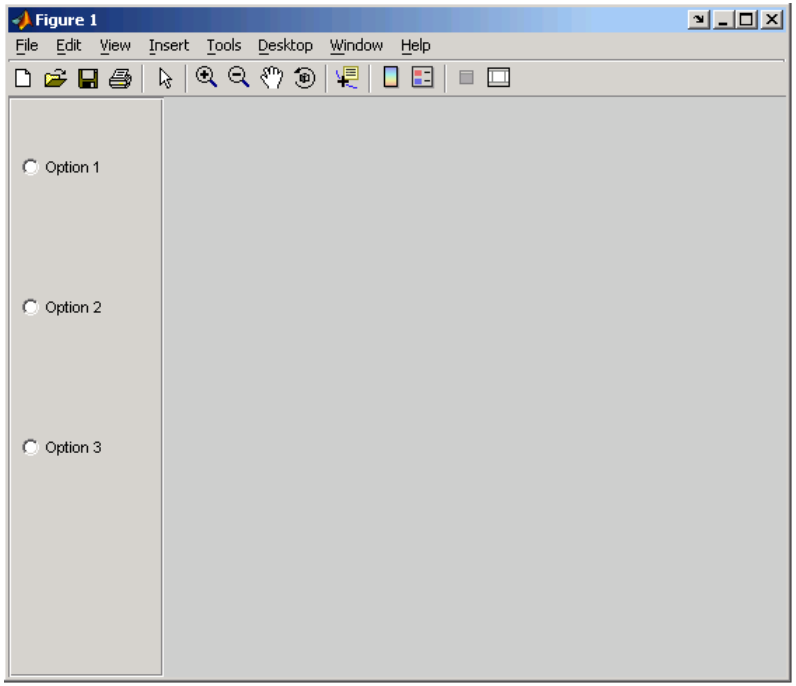
```
>> figure
>> t = uitable;
>> set(t, 'Data', magic(10));
>> set(t, 'ColumnWidth', {35})
```

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
  - seznam podporovaných Microsoft ActiveX control

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

  - examples
    - web browser

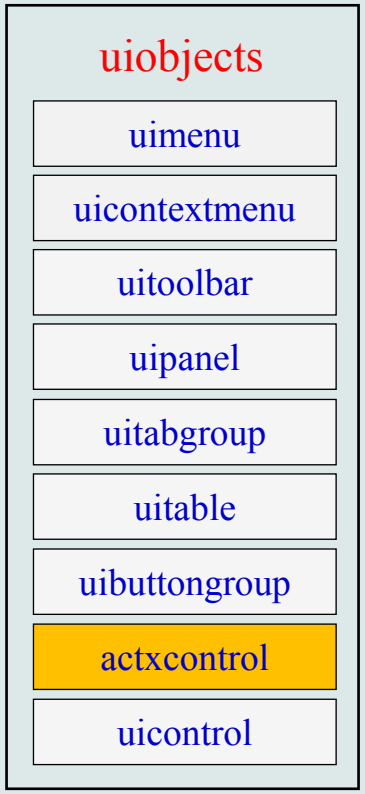
```
>> h = actxcontrol('AcroPDF.PDF.1',...
```

    - PDF reader

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

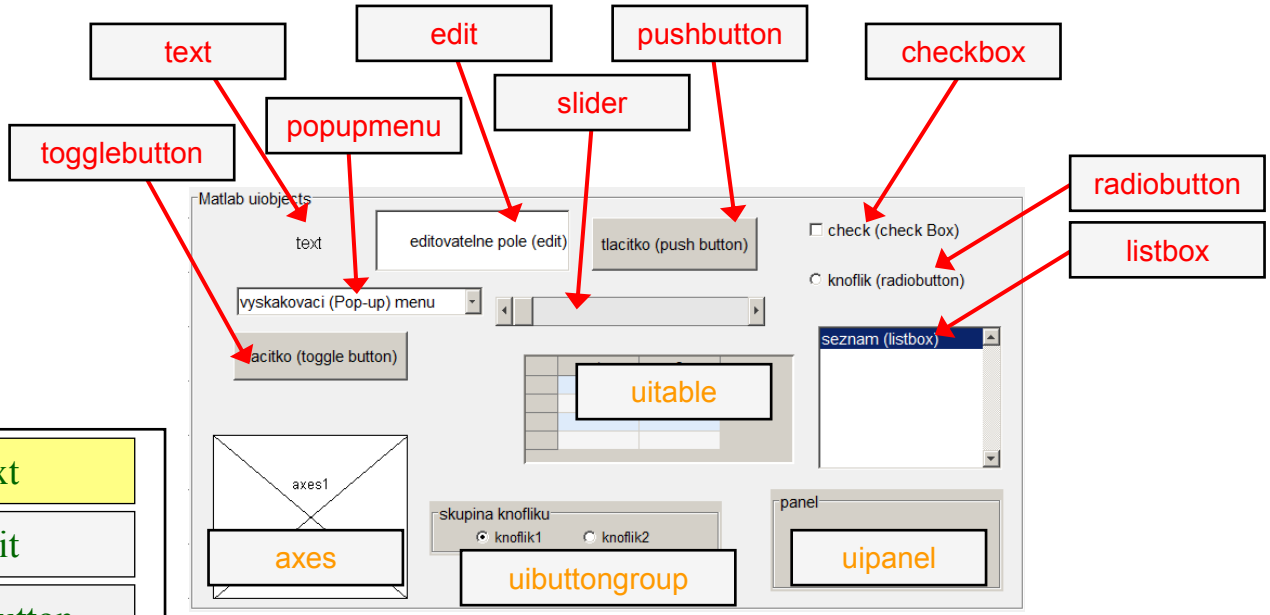
  - for more information see

```
>> docsearch getting started with COM
```



# 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

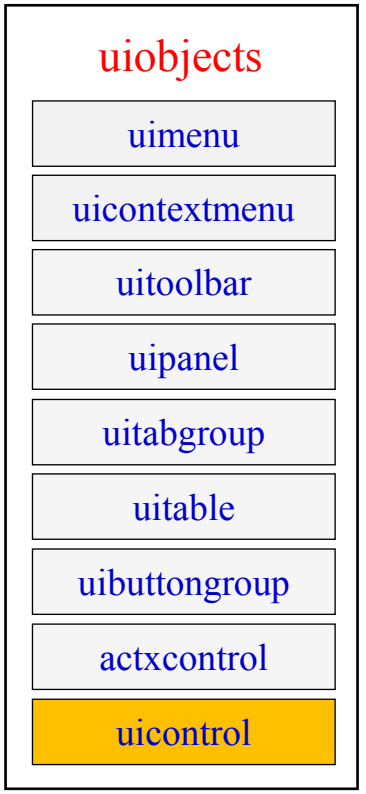
- uicontrol creates basic functional elements of GUI
- to change style of uicontrol use property style

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

- to get properties of uicontrol use

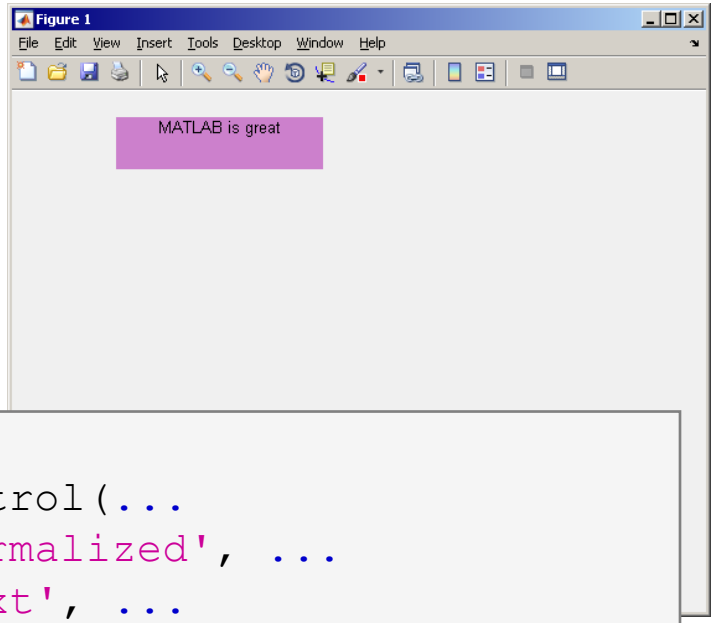
```
>> get(t);
```

- for more see >> doc uicontrol

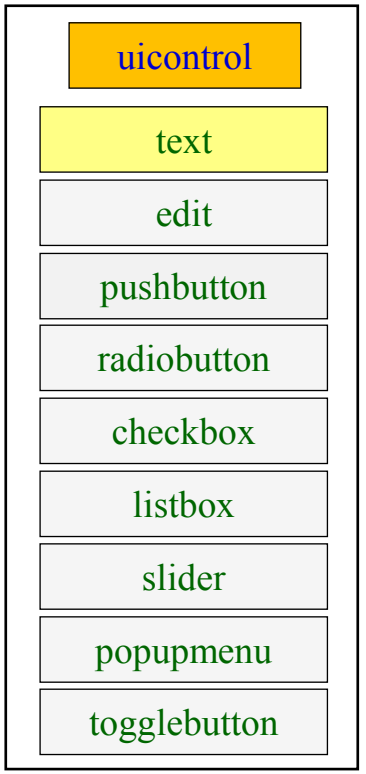


# Group uicontrol: `text`

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



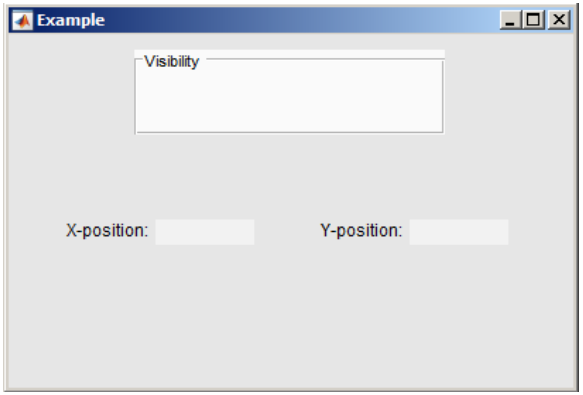

```
>> figure
>> text1 = uicontrol(...
    'Units', 'Normalized', ...
    'Style', 'Text', ...
    'Position', [0.15 0.85 0.3 0.1], ...
    'Tag', 'MTB', ...
    'FontSize', 10, ...
    'BackgroundColor', [0.8 0.5 0.8], ...
    'HorizontalAlignment', 'center', ...
    'String', 'MATLAB is great');
```



# Exercise – `text`

400 s ↑

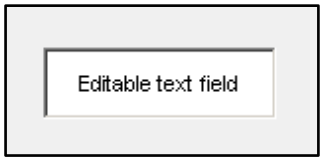
- create four text arrays having following properties that will be placed to following positions (normalized values)
  - `[0.1  0.4 0.15  0.075]  font  9  figureColor`
  - `[0.26 0.4 0.175 0.075]  font 10  textColor`
  - `[0.55 0.4 0.15  0.075]  font  9  figureColor`
  - `[0.71 0.4 0.175 0.075]  font 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

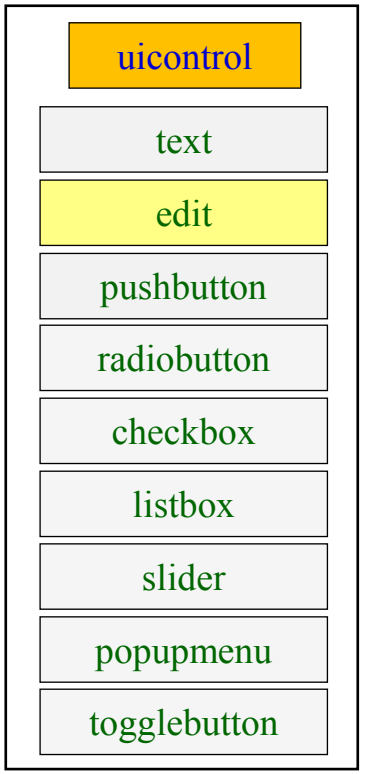


# Exercise – `text`, solution

# Group uicontrol: `edit`

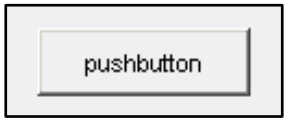
- enables to read an array of characters
  - the array of characters is of type `char`
  - the string has to be processed (`str2num`, `str2double`,...)
- CTRL+C,+V,+X,+A,+H shortcuts are available to user
- a console can be created using `edit` in Matlab

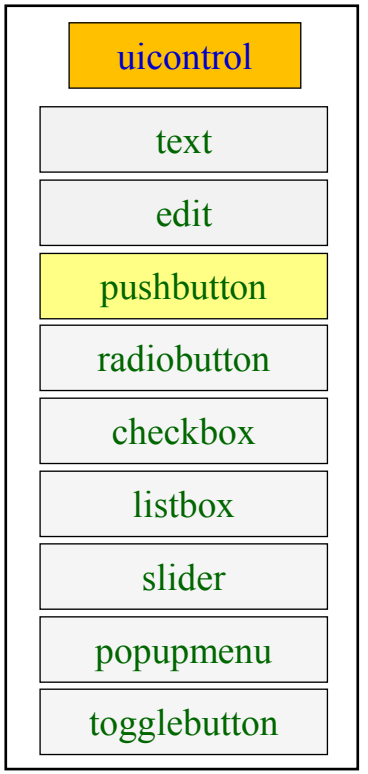


# Group uicontrol: `pushbutton`

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

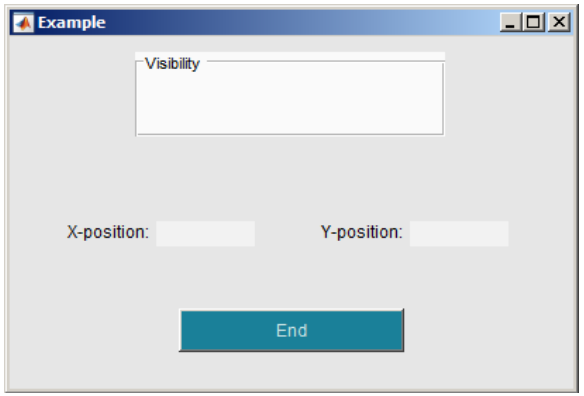


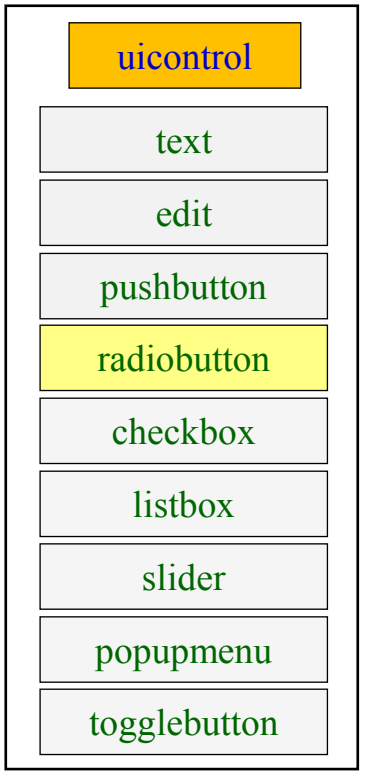
# Exercise – pushbutton

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]`



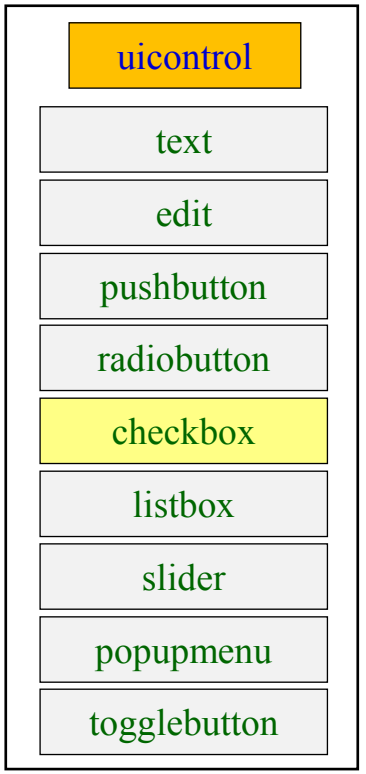
# Group uicontrol: `radiobutton`




- 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



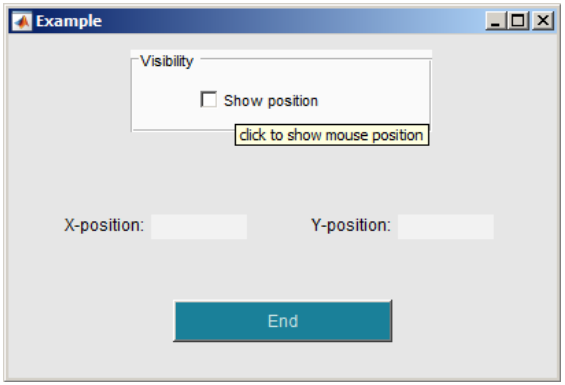

- similar to `radiobutton`
- tick box (with a text attached)
- `callback` called on state change


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

# Exercise – checkbox

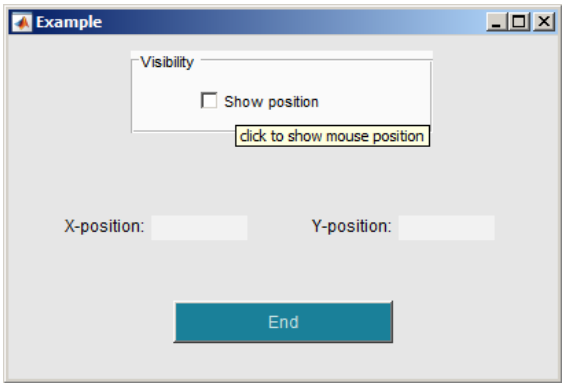
400 s ↑

- 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



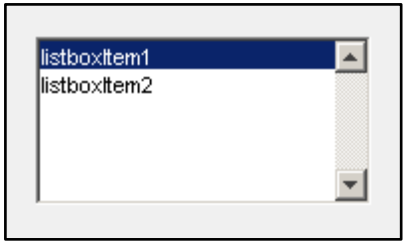
# Exercise

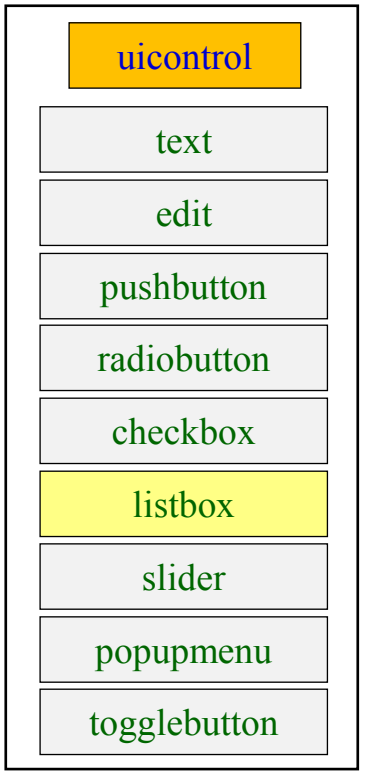
- Save your GUI file for later use (during next lecture)



# Group uicontrol: `listbox`

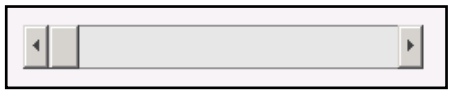
- list of items, it is possible to choose one or more items
- property `string` contains list of strings (items)
- property `value` contains matrix of selected items
- values `max` and `min` have impact on selection

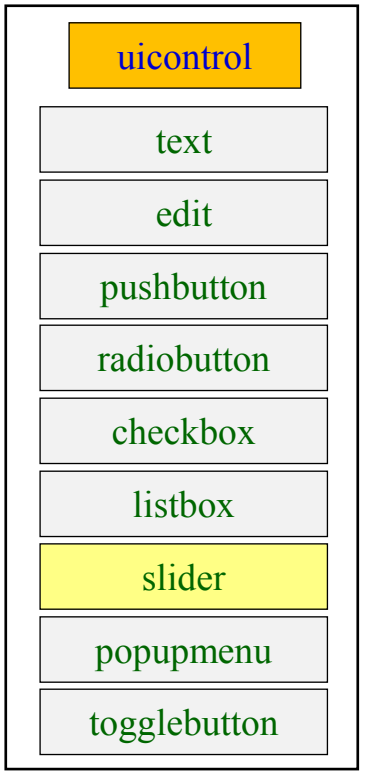


# Group uicontrol: `slider`

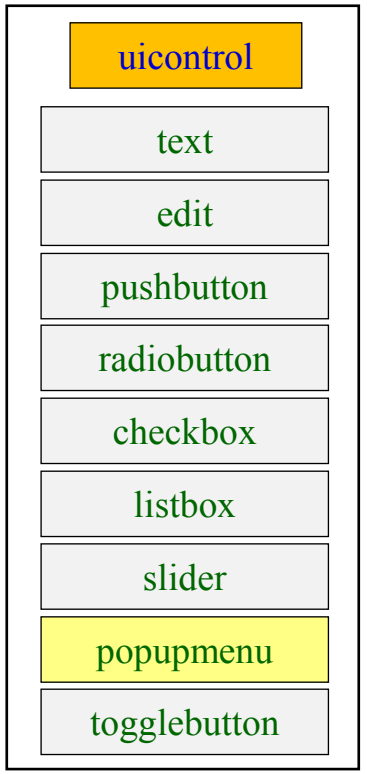
- input value is a numerical range (`min` and `max`)
- user moves slider by steps (`sliderstep`)
- requires
  - range
  - slider step
  - initial value



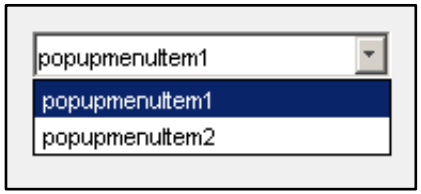



```
maxVal = 10;
minVal = 2;
slider_step(1) = 0.4/(maxVal-minVal);
slider_step(2) = 1/(maxVal-minVal);
set(sliderHndl, 'SliderStep', ...
    slider_step, 'Max', maxVal, ...
    'Min', minVal, 'Value', 6.5);
```

# Group uicontrol: popupmenu



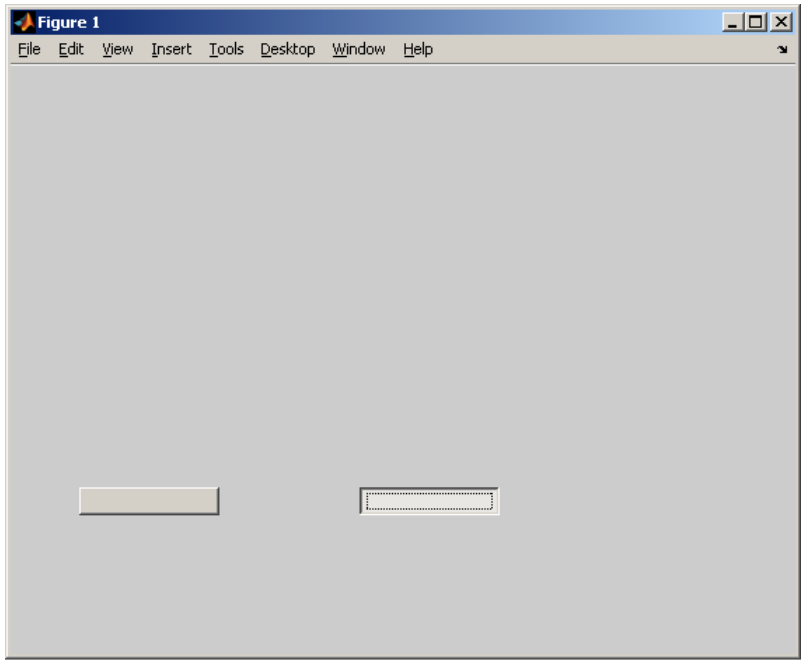

- clicking on arrow displays item list and enables to choose one item
  - `string` contains list of strings
  - `value` contains index of the selected item
- more info >> `doc uicontrol`

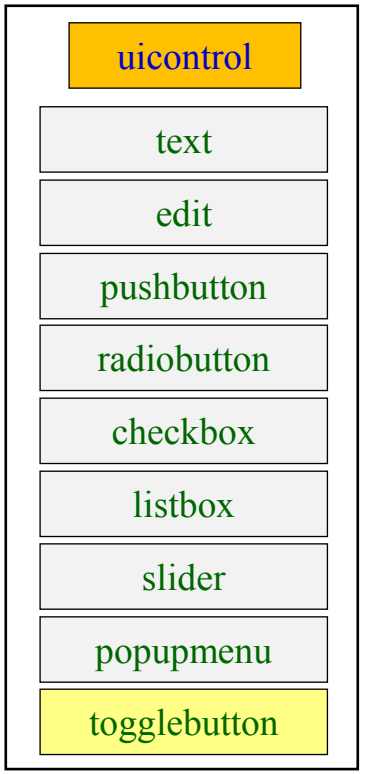



```
function popupFcn(hObj) % treated
val = get(hObj,'Value');

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

# Group uicontrol: `togglebutton`

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



# Discussed functions

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

# Exercise #1

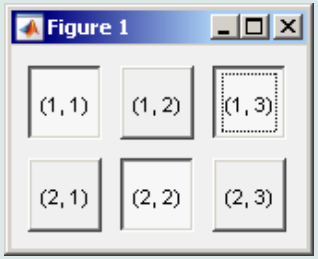
600 s ↑

- create function with two inputs and one output

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

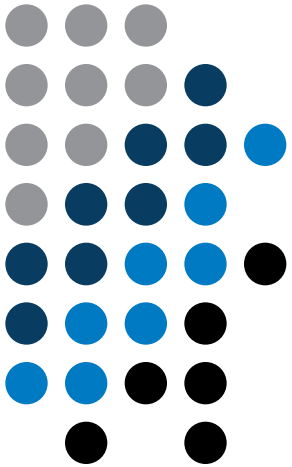
- function creates figure with toggle buttons arranged in matrix `nRows` x `nColumns`
- after clicking on toggle buttons and close window function returns matrix of logical values representing state of toggle buttons

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

logicState =

     1     0     1
     0     1     0
```



# Exercise #1 - solution

# Thank you!

ver. 8.1 (12/11/2017)
Miloslav Čapek, Pavel Valtr
miloslav.capek@fel.cvut.cz