Matlab Laboratory 7. Graphics Objects for Graphical User Interfaces in Matlab

Aim of the laboratory

- Graphics objects
- Introduction to GUI

Necessary equipment

- Workstations with MATLAB

Theoretical Approach

GUI=Graphical User Interface, is the pictorial interface to a program. It is supposed to provide a consistent appearance and should behave in an understandable, user-friendly way. It contains elements like bushbuttons, listboxes, menus, etc which have to be intuitive.

The interface is supposed to respond to mouseclicks or keyboard input at any time, this is why the interface is called **event driven**. "Event handling" manages the input events from the mouse or keyboard but also the use of timers for real-time apps and the calls of periodic functions.

MATLAB has an object-oriented handle programming graphic structure; the software includes also the GUIDE application which permits an easy construction of GUI structures.

The most important MATLAB GUI components, functions and tools can be grouped as follows:

- Handle graphics objects
- Figures, axes and Uicontrols (=core GUI elements)
- GUIDE (GUI design environment)
- Event handling (callback functions, timers, mouse/keyboard input)
- Global variables
- Executable gui-s that are stand-alone

Handle graphics objects = HG objects, are object-oriented structures (like figures, axes, Uicontrols (edit boxes, sliders etc)) that create graphical content. Handles are the storage variables of the graphics objects.

Graphics objects have a set of properties, like the Figure object has a Color property that allows the change of the figure's background; or the Edit Box Uicontrol has a String property that displays text in it's text box area.

The elements that compose a GUI can be:

Components	**Graphical controls** – created with *uicontrol*	Pushbuttons, toggle buttons, radio buttons, edit boxes, check boxes, sliders
	Static elements – created with *uicontrol*	Frames, text strings(text boxes), listboxes
	Menus – created with *uimenu*, *uicontextmenu*	
	Toolbars – created with *uitoolbar*	
	Axes - created with *axes*	
Figures	Components are arranged within a figure, which is seen as a window on the screen	

Callbacks	An event created by a mouse click, buttonclick or keyboard typing; They implement the function of each graphical component of the GUI	

The GUI **components** have each a group of **properties**, like name, color, size, font, text, etc, common for all objects.

Properties can be classified in **Settings** and **Callback** functions. The first ones provide information about an object while the functions define actions taken when an object is "called" or acted through a mouse click.

Callback functions are used for handling events occuring to objects of the user interface, for example if the gui has a pushbutton, one needs to create a function (callback function) to be executed if the user clicks the mouse on that button, or operations like click on a graphics object or closing a figure window. So they are are functions assigned and executed as a response to predefined user actions.

Common Callback Functions (examples)	
`ButtonDownFcn`	Routine executing when the cursor is placed over an object or within a few pixels over the object and the left mouse button is clicked
`CreateFcn`	Routine executed when an object is instantiated (created)
`DeleteFcn`	Routine executed just before the object is deleted or closed
`Figure Window Callback Functions`	
`KeyPressFcn`	Routine executing when the cursor is inside the figure window
`ResizeFcn`	Routine executing when the figure is resized by the user
`WindowButtonDownFcn`	Routine executing when a mouse button is pressed while the cursor is over the figure background, a disabled control of the user interface or the background of the axes
`WindowButtonUpFcn`	Routine executing when a mouse button is de-pressed(released) inside the figure window, after the mouse button was pressed in the figure window

Most common architectures use local handle object data structures for communicating the gui parameters in callback functions. For expanding gui-s to include multiple figure architectures, the variables can be globally instantiated.

Another feature of the created gui-s, is, they can stand alone and be used even if the user has no MATLAB available, because of the MATLAB compiler add on tool.

The Figure Function

The first step in designing a graphical user interface (GUI) in Matlab is to **create a new window**. For this purpose one will use the ***figure*** function. The syntax of the ***figure*** function is

```
h = figure('PropertyName',propertyvalue,...);
```

where *h* is the handle to the figure, and the ***'PropertyName', propertyvalue*** pairs customize the figure. MATLAB uses the default values for any properties that you do not explicitly define as arguments . *The full list of figure properties can be consulted in the Matlab help*.
To be noted that a numeric **handle** can also be assigned to the figure using `figure(h);` where *h* is an integer. This command makes h the current figure, forces it to become visible and raises it above all other figures on the screen. If Figure h does not exist, and h is an integer, a new figure is created having the handle h. The figure handle can be used later in the Matlab code to switch back to the desired window.

For creating a new figure window, the function **figure** is used like for instance in the following example., which you can test directly in the command window.

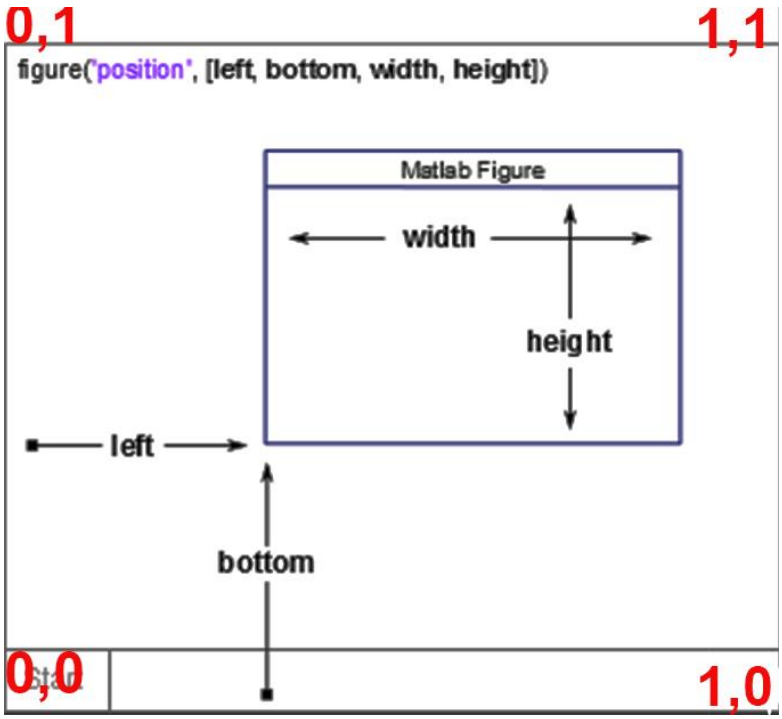
```
clear all;
close all;
Fig=figure('Name','New MatLab Figure',... %Name of the window
           'Units','normalized',...      % Measurement units between 0 and 1
           'Position',[.1 .1 .5 .5],...  % window position on the screen
           'NumberTitle','off', 'color',[0 0.5 0.5] ); % window Nr off=not assigned
```

The new window is created, at the specified position, with the specified title and a nice background color.
Close closes all windows that were previously opened.

Here some figure properties:

Color	RGB triplet – figure window background color **Long Name** / **Short Name** / **RGB Triplet** 'yellow' / 'y' / [1 1 0] 'magenta' / 'm' / [1 0 1] 'cyan' / 'c' / [0 1 1] 'red' / 'r' / [1 0 0] 'green' / 'g' / [0 1 0] 'blue' / 'b' / [0 0 1] 'white' / 'w' / [1 1 1] 'black' / 'k' / [0 0 0]
Name	Figure window title specified as string. By default, the figure title displays as `Figure 1`, `Figure 2`, and so on. When you set this property to a string, the figure title becomes `Figure:`*`n string`*. For displaying only the `Name` value, set **`IntegerHandle`** or **`NumberTitle`** to **`'off'`**
NumberTitle	Title number of the figure can be **`'on'` (default)** or **`'off'`.** The **`NumberTitle`** property determines whether MATLAB includes the string **`Figure`** ***n*** in the title bar, where *n* is the figure `Number`property value.
Position	Gives the location and size of the drawable area contained in the figure, given as: `[left bottom width height]` excluding the title bar, menu bar and tool bars.

	`left`	Distance from the left edge of the primary display to the inner left edge of the figure window. This value can be negative on systems that have more than one monitor.
	`bottom`	Distance from the bottom edge of the primary display to the inner bottom edge of the figure window. This value can be negative on systems that have more than one monitor.
	`width`	Distance between the right and left inner edges of the figure.
	`height`	Distance between the top and bottom inner edges of the figure



All measurements are in units specified by the Units property.

Units

Units of measurement, can be one of the following:

`'pixels'` (default)	Pixels. Starting in R2015b, distances in pixels are independent of y system resolution on Windows and Macintosh systems: On Windows systems, a pixel is 1/96th of an inch. On Macintosh systems, a pixel is 1/72nd of an inch. On Linux systems, the size of a pixel is determined by your system resolution.
`'normalized'`	These units are normalized with respect to the parent container. The lower-left corner of the container maps to `(0,0)` and the upper-right corner maps to `(1,1)`.
`'inches'`	Inches.
`'centimeters'`	Centimeters.
`'points'`	Points. One point equals 1/72nd of an inch.

<table>
<tr><td></td><td>
<table>
<tr><td><code>'characters'</code></td><td>These units are based on the default uicontrol font of the graphics root object:
Character width = width of the letter <code>x</code>.
Character height = distance between the baselines of two lines of text.
To access the default uicontrol font,
use <code>get(groot,'defaultuicontrolFontName')</code>
or <code>set(groot,'defaultuicontrolFontName')</code>.</td></tr>
</table>

MatLab measures all units from the lower corner of the parent object.
If you change the units, then it is good practice to return it to its default value after completing your computation to prevent affecting other functions that assume <code>Units</code> is the default value.

The order in which you specify the <code>Units</code> and <code>Position</code> properties has these effects:
If you specify the <code>Units</code> before the <code>Position</code> property, then MATLAB sets <code>Position</code> using the units you specify.
If you specify the <code>Units</code> property after the <code>Position</code> property, MATLAB sets the position using the default <code>Units</code>. Then, MATLAB converts the <code>Position</code> value to the equivalent value in units you specify.</td></tr>
<tr><td>Resize</td><td>Window resize mode can be <code>'on'</code> (default) or <code>'off'</code>;
When on, user can resize the figure window; if off, the window cannot be resized, it doesn't display any resizing window.</td></tr>
<tr><td>NextPlot</td><td>How to add next plot, in case of multiple plots:

<table>
<tr><td><code>'new'</code></td><td>Creates a new figure and uses it as the current figure.</td></tr>
<tr><td><code>'add'</code></td><td>Adds new graphics objects without clearing or resetting the current figure.</td></tr>
<tr><td><code>'replace'</code></td><td>Removes all axes objects and resets figure properties to their defaults before adding new graphics objects.

Equivalent to using the <code>clf</code> <code>reset</code> command.</td></tr>
</table>
</td></tr>
</table>

Creating user interface control elements

With the help of a gui, the user can modify the application's parameters without modifying each time the source code. Two functions are used for defining various graphics objects in gui: **uicontrol** and **uimenu**.

The buttons are graphics objects that, when being manipulated, the cause an action to take place, so an event to be driven, to happen.

The uicontrol Function

Control elements in Matlab are created using the ***uicontrol*** function. The syntax of the ***uicontrol*** function is

```
h = uicontrol('Name',Value,...)
```

where *h* is **the handle** to the control element, and the **'Name', Value pairs** specify the **properties** of the control element. *The complete list of control element properties can be consulted in the Matlab help*.

The properties that are probably most used are listed in Table 1.

Table 1

Name	Value
'Style'	string, type of control element
'Units'	string, interpretation of 'Position vector'
'Position'	four-element vector, coordinates of the control element
'String'	string, text to be displayed
'Callback'	string, action

The types of control elements which can be added on a Matlab GUI are:

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

The callback consists of Matlab code which states the actions to be taken when the control element is operated. Nevertheless, it is preferable to have the actions described in either a Matlab function or a Matlab script, and the callback should call the respective *.m* file.

The most important parameter of **uicontrol** is **style**, that selects the object type. The style parameter can be classified into **static** and **dynamic** (with callback) groups.

	Static:	**Dynamic:**
Style	Text Frame Listbox	Pushbutton Edit Radiobutton Checkbox Popupmenu Slider

Creating a **pushbutton**:

```
GO_p=uicontrol('Style','pushbutton',... %graphics object type
'Units','normalized',...      %units
'Position',[0.05 0.15 0.15 0.1],... %position in the figure window
'String','CLOSE',... %text on the button
'Callback','close'); %function called when the button is pressed
```

These buttons are small and usually text labeled. Regarding the **position** parameter, the buttons can be wherever placed inside the window, so the position is established in respect to the window, not the monitor.

Creating a **text** object. This object is static so it doesn't allow a callback function.

```
GO_t=uicontrol('Style','text',... %graphics object type
 'Units','normalized',...     %units
'Position',[0.3 0.85 0.16 0.05],... %position inside the figure window
'backgroundcolor','w',...  %of the button
'foregroundcolor','blue',... %of the text
'String','LOW PASS FILTER');   % label on the button
```

```
uicontrol('Style','text',...
    'Units','normalized',...
    'Position',[0.775 0.05 0.2 0.1],...
    'foregroundcolor','r',...
    'String','Dummy Pass Filter');
```

Creating an **edit** object:

```
GO_e=uicontrol('Style','edit',...
'Units','normalized',...
'Position',[0.6 0.85 0.16 0.05],...
'foregroundcolor','r',...
'String',20,...  %initial value-string, that the user can change
'Callback','');
```

These objects allow the user to load a string value (character array) that is used then by the application. This object makes possible an initial value to be accepted, edited, deleted or replaced.

Creating a **popupmenu** object:

```
GO_ppm=uicontrol('Style','PopupMenu',...
'Units','normalized',...
'Position',[0.8 0.85 0.06 0.05],...
'String','LPF|BPF|HPF',...
'Callback','');
```

In case of a popupmenu, options from a list can be chosen and taken various actions accordingly.

Creating a **checkbox**. They behave as an on-off switch.

```
GO_cbx1=uicontrol('Style','Checkbox',...
'Units','normalized',...
'String','CheckBox A',...
'BackgroundColor','y',...
'Position', [0.07 .7 .1 .1],...
'Callback','')

GO_cbx2=uicontrol('Style','Checkbox',...
'Units','normalized',...
'String','CheckBox B',...
'BackgroundColor','y',...
'Position', [0.07 .6 .1 .1],...
'Callback','')
```

Creating a **radiobutton** object. These buttons appear usually in a group and what's differentiating them from the checkbox is, the fact that just one of them can be selected, all the others becoming 'off'.

```
GO_rb1=uicontrol('Style','radiobutton',...
'Units','normalized',...
'Position',[0.85 0.7 .1 .1],...
'String',' RB1',...
'Callback','') ;
```

```
GO_rb2=uicontrol('Style','radiobutton',...
'Units','normalized',...
'Position',[0.85 0.6 .1 .1],...
'String',' RB2',...
'Callback','') ;
```

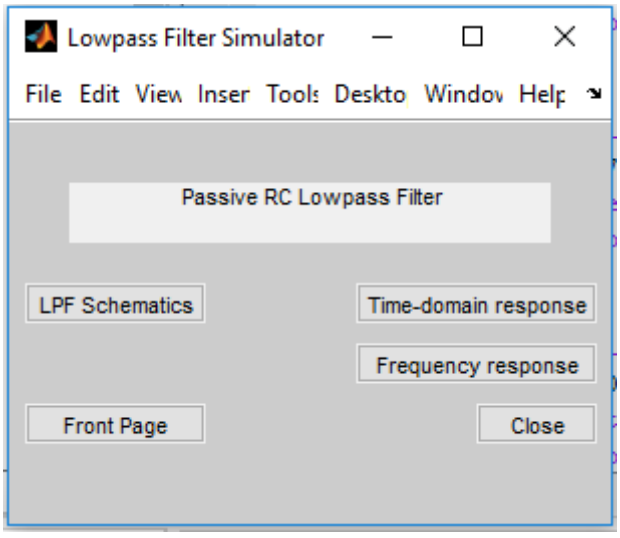
Creating a slider object. In itself, this object doesn't contain the values on it ends, so the limits in which it can vary. For the limit values, another 2 text objects have to be created, indicating the minimal and maximal value.

```
GO_s=uicontrol('Style','slide',...  %the slider
'Units','normalized',...
'Position',[0.4 0.6 .1 .05],...  %slider position
'Min',-150,'Max',150,'Value',1,... %min and max values indicated
'Callback','') ;

uicontrol('Style','text',...    %defining the minimal value
'Units','normalized',...
'Position',[0.37 0.6 0.03 0.05],... %!Min value position on the slider
'backgroundcolor','y',...
'foregroundcolor','black',...
'String',num2str(get(GO_s,'Min')));

uicontrol('Style','text',...    %defining the maximal value
'Units','normalized',...
'Position',[0.5 0.6 0.03 0.05],... %!Max value position on the slider
'backgroundcolor','y',...
'foregroundcolor','black',...
'String',num2str(get(GO_s,'Max')));
```

Note the **num2str** function, that transforms a numerical value into a string, in this case, the value of the minimal and maximal limit to be labeled on the text objects. It's inverse is **str2num**.

1. **Consider all the code parts written above and write them together into an .m file, save it and run it! Then you may personalize the created objects by modifying their properties.**

2. **Try to obtaing a similar window like the one below:**



3. **Test and try to understand the following code. Write it into a function file with the same name as the function (of course! ☺) and run it by calling the function from the prompter into the command window.**

```
function mygui   %this should be also the name of the function file
    % Creates a figure and axes
    f = figure('Visible','off');
    ax = axes('Units','pixels');
    surf(peaks)  % surf(X,Y,Z) plots a 3D colored surface and the color is proportional
%to surface height.

    % Creates a pop-up menu
    popup = uicontrol('Style', 'popup',...
           'String', {'spring','jet','hsv','hot','cool','gray'},...   %colormaps
           'Position', [20 340 100 50],...
           'Callback', @setmap);

   % Creates a push button
    btn = uicontrol('Style', 'pushbutton', 'String', 'Clear',...
        'Position', [20 20 50 20],...
        'Callback', 'cla');    %cla=clears current axis

   % Creates a slider, this time without a minimal and maximal value
    sld = uicontrol('Style', 'slider',...
        'Min',1,'Max',50,'Value',41,...
        'Position', [400 20 120 20],...
        'Callback', @surfzlim);

    % but... it adds a text object to label the slider!
    txt = uicontrol('Style','text',...
        'Position',[400 45 120 20],...
        'String','Vertical Exaggeration');

    % Make figure visble after adding all components
    f.Visible = 'on';
    % This code uses dot notation to set properties.
    % Dot notation runs in R2014b and later.
    % For R2014a and earlier: set(f,'Visible','on');

    function setmap(source,callbackdata)
        % To access individual properties of a graphics object, use object.PropertyName
        val = source.Value;
        maps = source.String;
        % For R2014a and earlier:
        % val = get(source,'Value');
        % maps = get(source,'String');

        newmap = maps{val};
        colormap(newmap);
    end
    function surfzlim(source,callbackdata)
        val = 51 - source.Value;
        % For R2014a and earlier:
        % val = 51 - get(source,'Value');

        zlim(ax,[-val val]);
    end
end
```

Matlab Laboratory 8. Graphical User Interfaces - Part2

Aim of the laboratory

- Graphical representations
- Customization of Matlab simple gui-s

Necessary equipment

- Workstations with LTSpice and MATLAB

Exercises

1. Display an image inside a figure, consider a simple circuit jpg.

Creating a new figure window:

```
clear all;
close all;
Fig=figure('Name','Figure in Laboratory_8 ',... %figure window name%
'Units','normalized',...   %Units 0->1 %
'Position',[.1 .1 .5 .5],... %figure position on screen - see L7 %
'NumberTitle','off');      %window Nr - does not assign%
```

In the new created figure, one wants to display an image, that can be for example the scheme of an electronic circuit like RLC, BPF, Amplifier etc. this is done with the command **image** or **imshow.**

imshow - Displays the image in Handle Graphics figure.

imshow(I) displays the grayscale image I

imshow(RGB) displays the truecolor image RGB

image(C) displays the data in array C as an image. Each element of C specifies the color for 1 pixel of the image. The resulting image is an m-by-n grid of pixels where m is the number of columns and n is the number of rows in C. The row and column indices of the elements determine the centers of the corresponding pixels.

When C is a 2-dimensional m-by-n matrix, the elements of C are used as indices into the current COLORMAP to determine the color. When C is a 3-dimensional m-by-n-by-3 matrix, the elements in C(:,:,1) are interpreted as red intensities, in C(:,:,2) as green intensities, and in C(:,:,3) as blue intensities.

image(C) places the center of element C(1,1) at (1,1) in the axes, and the center of element (M,N) at (M,N) in the axes, and draws each rectilinear patch as 1 unit in width and height.

image(x,y,C) specifies the image location. Use x and y to specify the locations of the corners corresponding to C(1,1) and C(m,n). To specify both corners, set x and y as two-element

vectors. To specify the first corner and let image determine the other, set x and y as scalar values. The image is stretched and oriented as applicable.

IM = image(...) returns the image object created. Use **IM** to set properties of the image after it is created. You can specify this output with any of the input argument combinations in the previous syntaxes.

IM1 = imread(FILENAME,FMT) reads a grayscale or color image from the file specified by the string FILENAME. FILENAME must be in the current directory, in a directory on the MATLAB path, or include a full or relative path to a file. The text string FMT specifies the format of the file by its standard file extension. For example, specify 'gif' for Graphics Interchange Format files. To see a list of supported formats, with their file extensions, use the IMFORMATS function. If imread cannot find a file named FILENAME, it looks for a file named FILENAME.FMT.

IM2 = imread(FILENAME) attempts to infer the format of the file from its content.

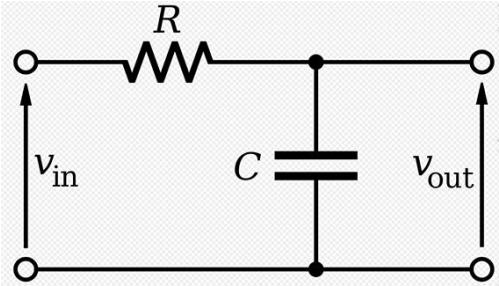
IM3= imread(URL,...) reads the image from an Internet URL.

For displaying the image, either **image** or **imshow** can be used, as shown below. The image has to be in the same working folder where also the running m-file is.

```
%reads the image the image has to be in the working folder, where also
% the m-file is

x = imread('LPf.jpg');
image(x);   %displays the image%
```

or:

```
w = imread('LPF.jpg'); %reads the picture%
imshow(w,'InitialMagnification',150) %displays the picture%
```



2. Consider a sine function that needs to be displayed into a MatLab figure, while allowing the modification of the amplitude of the signal. The initial values are amplitude A=1, frequency=50Hz, the time defined from 0 to 2T with a step of T/100. The function y is defined as y=A*sin(2π*f*t).

```
clear all;
close all;
A=1;   %amplitude
f=50;  %frequency
T=1/f; % period
```

```
%defines the time values from 0 to 2T with step T/100
%2*T means, in this case 2 periods of the sinus signal are viewed
t=(0:T/100:2*T);
y=A*sin(2*pi*f*t); % y function
plot(t,y);
```

For creating the interface, a script file will start the application and a function file will do the graphical representation and also the amplitude modification. The main file, the script contains the initializations and the call of the function.

```
%main_sinus.m
clear all;
close all;
A=1; %amplitude
f=50; %frequency
sinus(A,f); %function call sinus(A,f)
```

Remember that the function file has to have the exact name of the function, in this case it has to be „sinus.m”

```
%function file sinus.m
function sinus(A,f)
Fig=figure('Name','Sine Function',...
'Units','normalized',...
'Position',[0.3 0.3 0.5 0.5],...
'NumberTitle','off');
%for modifying the amplitude a text button is created that labels
%the edit button

uicontrol('Style','text', ...
'Units','normalized', ...
'Position',[0.91 0.95 0.1 0.04], ...
'String','A - Function amplitude');

%the edit button will contain the actual value of the signal
amplitude
% String, A = loads into string the value of amplitude transmitted
% by the script
uicontrol('Style','edit',...
'Units','normalized',...
'Position',[0.91 0.9 0.1 0.04],...
'String',A,...
'Callback','A=str2num(get(gco,''string'')),close;sinus(A,f);');
% the callback allows loading a new amplitude value by using the
% editable field-zone
%A=str2num converts a string array from the button area into a
number
% assigned to variable A
%get returns the object properties
%gco allows working with the objects properties
%close closes the application and sinus(a,f)does a re-call to the
% sinus function with the new value for A, so that the new signal
%is displayed in the figure window

T=1/f;
```

```
t=(0:T/100:2*T);
y=A*sin(2*pi*f*t);
plot(t,y);
grid on;
end
```

3. Having example 2, try to build the following interface which does the following:
   - Represents graphically in 3 different windows, functions sin, cos and sin+cos
   - Allows modifying the amplitudes for sin and cos, frequency and number of signal periods displayed; variables for sine amplitude is A, for cos amplitude is B, for frequency is f and number of signal periods is N
   - The initial values are A=10; B=5; f=50; N=4
   - The gui contains also a button for CLOSE and a button for RESET (reetting the characteristics to their initial values)

The trigonometric functions are defined as:

```
f1=A*sin(2*pi*f*t);     % first function,sine with amplitude A
f2=B*cos(5*2*pi*f*t);   % second function,cos with amplitude B
f3=f1+f2;               % their sum
```

The main file is:

```
clear all;
close all;
A=10;         % magnitude of signal 1  =sine
B=5;          % magnitude signal 2     =cos
f=50;         % frequency
N=4;          % nr of periods to be visualized
%the varbs are transmitted to the interface function
interface(A,B,f,N);
```

The interface function file is:

```
%interface.m

%creates a new figure
function interface(A,B,f,N)
Fig=figure('Name','Signals Plot',...
           'Units','normalized',...
           'NumberTitle','off',...
           'Position',[0.1 0.1 0.8 0.8],...
           'Color',[0.5 0.5 0.9]);

T=1/f;                  % signal period
t=0:T/100:N*T;          % time interval chosen
f1=A*sin(2*pi*f*t);     % first function
f2=B*cos(5*2*pi*f*t);   % second function
f3=f1+f2;               % sum of the 2 functions
```

```
% in defining the buttons observe the repetition of exercise 2
% pushbutton CLOSE
uicontrol('Style','pushbutton',...
'Units','normalized',...
'Position',[0.9 0.9 0.08 .05],...
'String','CLOSE',...
          'Callback','close');
% TextButton A
uicontrol('Style','text',...
    'Units','normalized',...
          'Position',[0.9 0.83 0.08 .05],...
          'backgroundcolor',[0.5 0.5 0.9],...
          'string','A');
% edit for A
uicontrol('Style','edit',...
          'Units','normalized',...
          'Position',[0.9 0.80 0.08 .05],...
          'String',A,...
'Callback',['A=str2num(get(gco,''string'')),close;interface(A,B,f,N);']);

% Text Button B
uicontrol('Style','text',...
          'Units','normalized',...
          'Position',[0.9 0.73 0.08 .05],...
          'backgroundcolor',[0.5 0.5 0.9],...
          'string','B');

% edit for B
uicontrol('Style','edit',...
          'Units','normalized',...
          'Position',[0.9 0.70 0.08 .05],...
          'String',B,...
'Callback',['B=str2num(get(gco,''string'')),close;interface(A,B,f,N);']);
% f  button
uicontrol('Style','text',...
          'Units','normalized',...
          'Position',[0.9 0.63 0.08 .05],...
          'backgroundcolor',[0.5 0.5 0.9],...
          'string','f');
% edit for f
uicontrol('Style','edit',...
          'Units','normalized',...
          'Position',[0.9 0.60 0.08 .05],...
          'String',f,...
'Callback',['f=str2num(get(gco,''string'')),close;interface(A,B,f,N);']);

% TextButton N
% in analogy to the definitions above, complete the code with the
% text button for N


% edit for N
% in analogy to the definitions above, complete the code with the
% edit button for N


% reset button
uicontrol('Style','pushbutton',...
          'Units','normalized',...
```

```
           'Position',[0.9 0.2 0.08 .05],...
           'string','RESET',...
           'Callback','close;interface(10,5,50,4);'); %function call

subplot('position',[0.1 0.72 0.4 0.25]);%plotting window set  plot(t,f1);
grid on;
plot(t,f1);
title('Sinus');
xlabel('time[s]');
ylabel('Magnitude[V]');


subplot('position',[0.1 0.38 0.4 0.25]);%plotting window
plot(t,f2);
grid on;
title('Cosinus');
xlabel('time[s]');
ylabel('Magnitude[V]');

subplot('position',[0.1 0.05 0.4 0.25]);%plotting window
plot(t,f3);
grid on;
title('Sin+Cos');
xlabel('time[s]');
ylabel('Magnitude[V]');

end
```

Matlab Laboratory 9. Differential Equations in Matlab

Theoretical Fundamentals

The previous laboratories have presented an overview of the Matlab computation engine, illustrating the employment of matrices, operations with matrices, program control statements and plots. Further on, this laboratory presents the employment of Matlab to solve differential equations.

Matlab provides a very powerful solver for differential equation. Accordingly, solving differential equations becomes both easy and straightforward. The syntax of the differential equation solver is

```
[T,Y] = solver(odefun,tspan,y0,options)
```

The Matlab differential equation solvers are ***ode23***, ***ode45***, ***ode113***, ***ode15s***, ***ode23s***, ***ode23t*** and ***ode23tb***. The characteristics of the different solvers can be consulted in the Matlab help.

The differential equation to be solved must be written in advance as a Matlab function, and is transmitted to the solver through the function handle *odefun*. The parameter *tspan* states the integration interval and *y0* state the initial conditions. Further options can be placed through *options*, as indicated in the Matlab help.

To illustrate the Matlab differential equation solver, consider plotting the RC circuit capacitor charge and discharge voltage and current respectively. The input voltage in this case is considered a square wave which switches between positive DC voltage V_{DC} and ground.

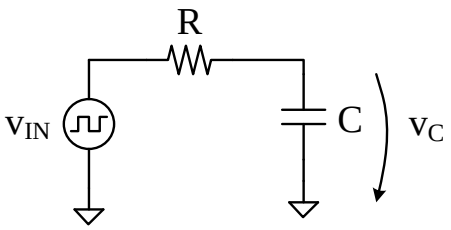
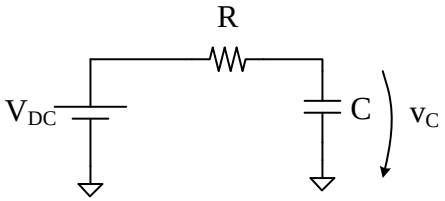
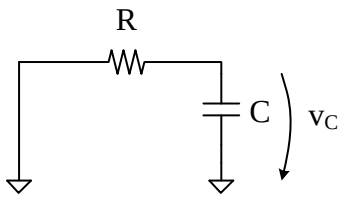


Figure 1.

Depending on the state of the input voltage, the circuit has two phases of operation. When the input voltage is high, the capacitance charges from DC voltage source V_{DC} through passive resistance R. The equivalent circuit is illustrated next.



When the input voltage is low, the capacitance discharges to ground through passive resistance R. The equivalent circuit is illustrated next.



Regardless of the phase of operation, the capacitance current is given by

$$i_C = C \cdot \frac{dv_C}{dt}$$

For the capacitance voltage, the two operation phases must be treated separately. During the charge phase, Kirchoff's law expresses that

$$V_{DC} = R \cdot i_C + v_C$$

The expression of the capacitance current in permanent regime is then expressed as

$$i_C = \frac{V_{DC} - v_C}{R}$$

which leads to the expression of the capacitance charging voltage

$$\frac{dv_C}{dt} = \frac{V_{DC} - v_C}{R \cdot C}$$

Similarly, during the discharge phase, Kirchoff's law expresses that

$$R \cdot i_C + v_C = 0$$

which leads to the expression of the discharge voltage

$$\frac{dv_C}{dt} = -\frac{v_C}{R \cdot C}$$

One will observe that the capacitance current and voltage are expressed by differential equations respectively. The *solver* which is required for this exercise is ***ode45***. The Matlab function which expresses the capacitance voltage during the charging phase is

```
function dy=F1(t,y,flag,VDC,R,C)
dy=(VDC-y)./(R.*C);
```

In the code section above, the key word *flag* separates the output and the input function arguments. Accordingly, the arguments before the *flag* will be returned by the function, whereas the arguments after the *flag* are required to compute the function result.

The Matlab function F2 which express the capacitance voltage during the discharge phase is

```
function dy=F2(t,y,flag,R,C)
dy=-y./(R.*C);
```

The function which solves the differential equations is listed as follows. To be noticed is that, besides the circuit parameters, the function also takes the number of periods to be displayed as an argument. This allows the illustration of the transient regime.

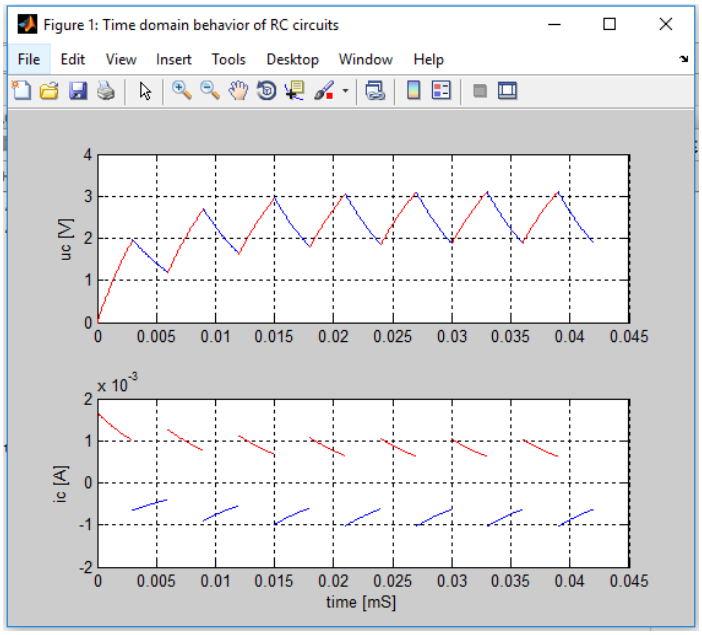
```
function RC_circuit(VDC,R,C,T,D,N)

figure('Name','Time domain behavior of RC circuits');

Tc=D.*T;
y=0; %initial conditions, capacitance is discharged

for j=1:N

    d(j)=min(y); %initial conditions for current frame, charge phase
    [t,y]=ode45('F1',[(j-1).*T (j-1).*T+Tc],d(j),[],VDC,R,C);
    i=(VDC-y)./R;  %capacitance current

    subplot(2,1,1);plot(t,y,'r');  %plot charge voltage and current
    hold on;grid on;
    subplot(2,1,2);plot(t,i,'r');
    hold on;grid on;

    a(j)=max(y); %initial conditions for current frame, discharge phase
    [t,y]=ode45('F2',[(j-1).*T+Tc j.*T],[a(j)],[],R,C);
    i=(-y)./R;   %capacitance current

    subplot(2,1,1);plot(t,y,'b');  %plot discharge voltage and current
    hold on;
    subplot(2,1,2);plot(t,i,'b');
    hold on;
end

subplot(2,1,1);     %customize axes
ylabel('vc [V]');
subplot(2,1,2);
ylabel('ic [A]');
xlabel('time [mS]');
```

To call this function, write the following Matlab script.

```
clear;
close all;

VDC = 5;
R = 3e3;
C = 2e-6;
T = 6e-3;     % period
```

```
D = 0.5;      % duty cycle, 0.5 for rectangular wave
N = 7;

RC_circuit(VDC,R,C,T,D,N);
```

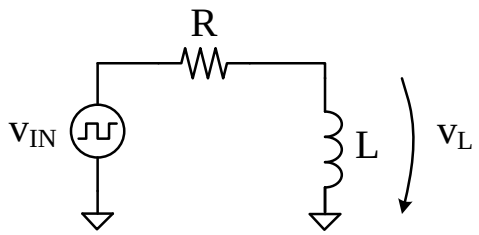
The resulting capacitance voltage and current are then plotted in the following figure.



Exercise 1. Design a GUI which allows you to change the values of the resistance, capacitance, period, duty cycle and number of periods to be displayed.

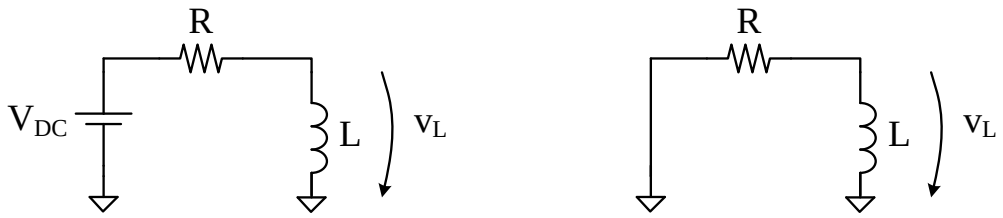
Hint: use textboxes and a button

Exercise 2. Plot the inductance voltage and current of the RL circuit



Depending on the state of the input voltage, the circuit has two phases of operation. When the input voltage is high, the inductance charges from DC voltage source V_{DC} through passive resistance R.

When the input voltage is low, the inductance discharges to ground through passive resistance R. The equivalent circuits are illustrated next.



Regardless of the operation phase, the inductance voltage is given by

$$v_L = L \cdot \frac{di_L}{dt}$$

For the inductance current, the two operation phases must be treated separately. During the charge phase, Kirchoff's law expresses that

$$V_{DC} = R \cdot i_L + v_L$$

The expression of the inductance voltage in permanent regime is then expressed as

$$v_L = V_{DC} - R \cdot i_L$$

which leads to the expression of the capacitance charging current

$$\frac{di_L}{dt} = \frac{V_{DC} - R \cdot i_L}{L}$$

Similarly, during the discharge phase, Kirchoff's law expresses that

$$R \cdot i_C + v_L = 0$$

which leads to the expression of the discharge voltage

$$\frac{di_L}{dt} = -\frac{R \cdot i_L}{L}$$

Exercise 3. Design a GUI for the RL circuit

Matlab Laboratory 10. Menus in GUI

Aim of the laboratory

- Creating menus in gui
- Customization of simple menus

Necessary equipment

- Workstations with MATLAB

The menu is useful in a gui for example to access a documentation that explains the project theme, that shows the circuits and schemes used in the project, that can have a theoretical part(like equations used) and a first page for the documentation indicating the authors of the project. The files accessed through the menu can be .pdf, .doc, .html, .bmp etc.

Import and display of text files to Matlab

Matlab allows the import of a series of text file formats, e.g.: *.txt*, *.doc*, *.docx*, etc. as well as web page formats like *.html*. One will choose the format of the textfile or web page depending on the visual requirements of the project.

A project documentation can contain sections like the following:

- Front page
- Table of contents
- Abstract
- Introduction
- Theoretical backgrounds

The documentation front page can contain the following elements:

- University name
- Faculty name
- Type of project (semester, annual, biannual, diploma thesys, etc.)
- Subject / Title/Theme
- Name and function of the coordinator
- Author’s name
- Place and date

An example of a project front page is illustrated in figure 1.

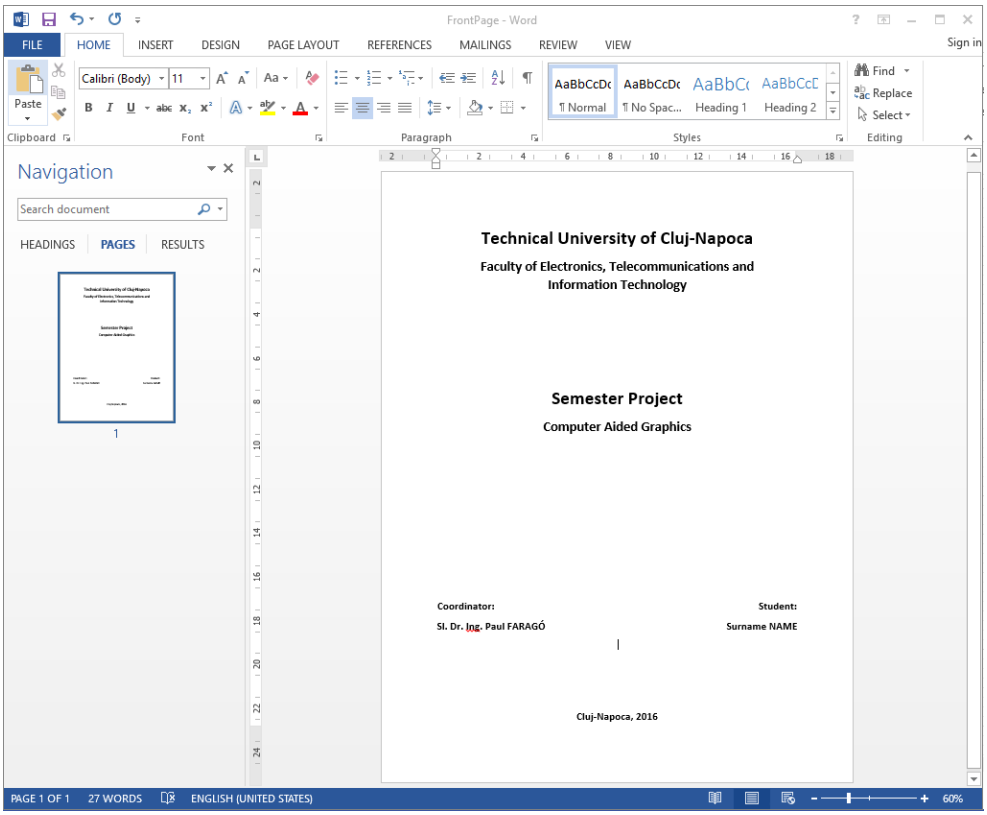


Figure 1

Importation of a text file to Matlab is performed with the ***open*** function:

T = **open**(name)

Table 1

Where name is a string depending on the type of the object named by that string:	
variable	Opens named array in variable editor
.m file	Opens program file in MatLab editor
.fig file	Opens the indicated figure in handle graphics
.prj file	Opens the project in compiler development tool
.html file	Opens a html document in MatLab browser; the html file can be created in Word and saved as “html” file
.url file	Opens an internet location in the default web browser
.doc file	Opens a document in MSWord
.pdf file	Opens a pdf file in Adobe Acrobat
.ppt file	Opens a document in MSPowerPoint
.exe file	Runs a MSWindows executable file

Open works similary to the load command, in the way that it searches for files. If the “name” file exists on the MatLab path, it opens the file returned by “which”. If the file “name” exists in the file system, it opens the file with the “name”.

Example1:

```
function front_page
open('FrontPage.docx');
```

This function will open the *FrontPage.docx* file in MS Word, as illustrated in Figure 1, if the file exists and is placed on the corresponding path! However, since the *.docx* file is opened in a different environment than Matlab, the ***close*** function will not close the text document!

Similary, the front page can be saved in MS Word as a webpage with *.htm* extension. The Matlab function to open the web page is

```
function front_page
open('FrontPage.htm');
```

This function will open the *FrontPage.htm* webpage in a new window in Matlab, as illustrated in Figure 2. Since the webpage was opened in Matlab, closing the file in this case is performed with the ***close*** function. Calling the ***close()*** function without any arguments will close the currently active window.

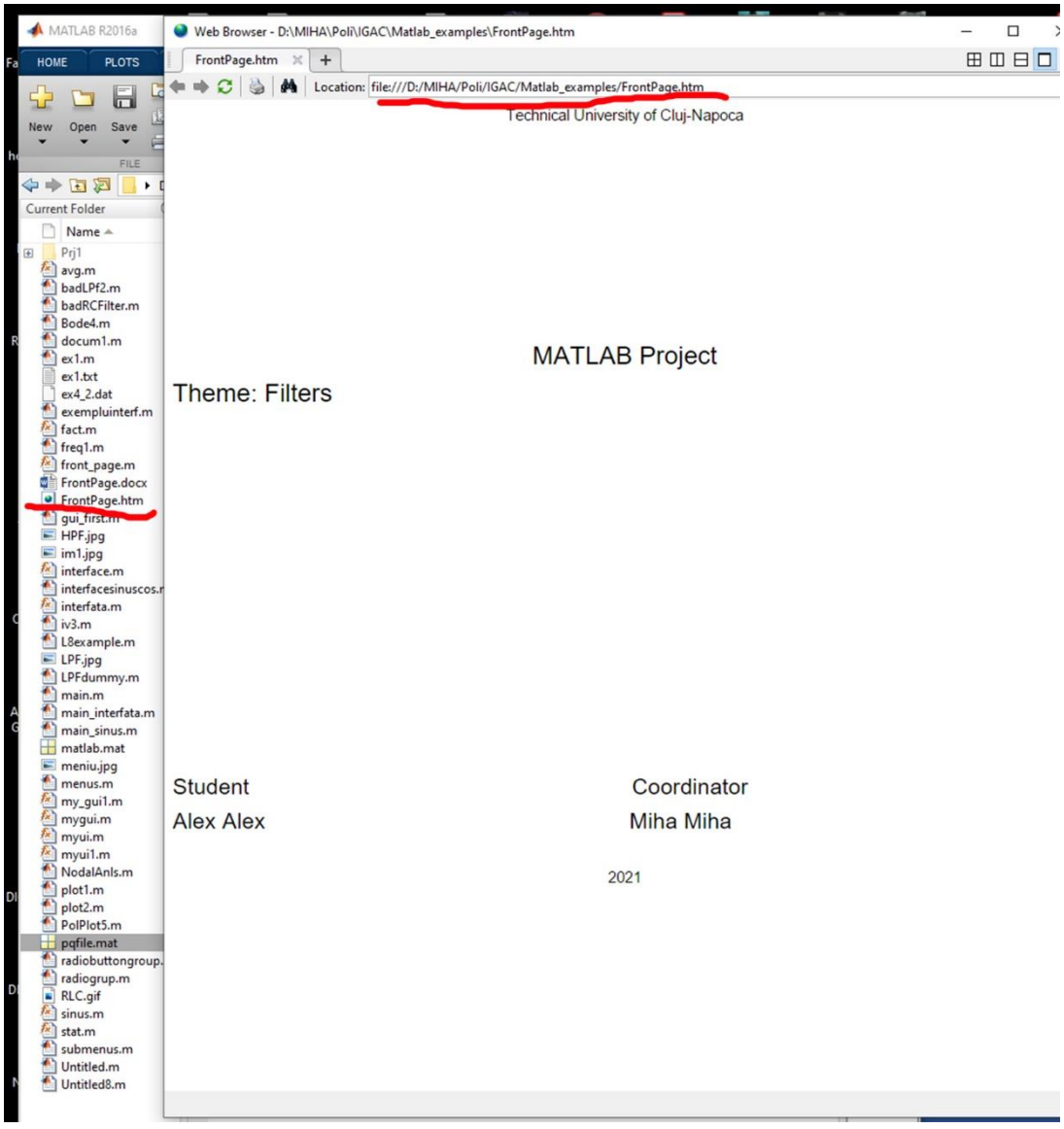


Figure 2

The uimenu command(function)

The menus in MatLab gui can be created by using the command(function) **uimenu**.

The uimenu command allows creating a user interface menu on the menu bar at the top of the current figure window, on the right of the implicite figure menu, and returns a handle to it.

H=uimenu([parent,] 'PropertyName1',value1,'PropertyName2',value2,...) creates a menu on the menu bar at the top of the current figure window and returns a handle **H** to it.

Parent – if it exists in the command, is the handle to a previously defined menu item, and the **[*Name*, *Value*] pairs** specify the properties of the menu item. The properties of interest in this laboratory are listed in Table 2.

Table 2

Name	**Value**
'Label'	string, name of the menu item
'Separator'	string: 'on'/'off', separation line within the menu
'Accelerator'	character, keyboard shortcut for the menu item
'Callback'	string, action

Example2: Therefore, regarding our previous example with the frontpage, for obtaining a simple menu that accesses the front page and a picture, one has to do the following:

1. Create the function and the function file that opens the front page, meaning:

Function file called "`front_page.m`"

Containing:

```
function front_page
open('FrontPage.htm');
end
```

2. Create the function and function file that reads the image, in our case a circuit.

The function file is called "`LowPassFilter.m`" containing (see L8) :

```
%reads the image the image has to be in the working folder, where
also the m-file is

function LowPassFilter
i=imread('LPF.jpg');
image(i) %displays the image%
title('LOW PASS FILTER');
```

```
uicontrol('Style','pushbutton',...
'Units','normalized',...
'Position',[0.78 0.03 0.2 0.07],...
'String','CLOSE',...
'Callback','close;');

end
```

3. The main menu file – called the way you want, and that is the file that you will run, containing:

```
figure();
h = uimenu('Label','NewMenu');
uimenu(h,'Label','Front Page','Callback','front_page');
uimenu(h,'Label','LPF','Callback','LowPassFilter' ,...
                  'Accelerator','L');
uimenu(h,'Label','Close','Callback','close',...
         'Separator','on','Accelerator','Q');
```

Try it!

The Save Command

The **save** command/function saves workspace variables into files.

save(*FILENAME*) stores all variables from the current workspace in a MATLAB formatted binary file (***mat***-file) called *FILENAME*.

save(*FILENAME*,VARIABLES) stores only the specified variables into the file.

If "FileName" is not specified, the save function saves the variables to a file created implicitly as "***matlab.mat***". If the filename does not include an extension, the implicite extension "mat" is appended. If the filename doesn't include a full path, MatLab saves the file in the current folder if permitted. Otherwise, one has to change the path to a current folder path were saving of files is allowed, meaning you have permission to write the file and to the file.

Example4:

1. If the command is used

save example.mat

It saves into the "example.mat" file all the variables from the workspace.

2. Saves two variables into a file, where "filename" is a variable:

```
savefile = 'pqfile.mat';
p = rand(1, 10);
q = ones(10);
```

```
save(savefile, 'p', 'q');
```

Menu properties can be set at object creation time using PropertyName/PropertyValue pair arguments to uimenu, or changed later using the SET command. Execute GET(H) to see a list of uimenu object properties and their current values. Execute SET(H) to see a list of uimenu object properties and legal property values.

If the ***uimenu*** function is called without the *parent* argument, it creates a new menu hierarchy in the menu bar of the window. The handle to the new menu is assigned to variable *h*, which is later used for creating menus and submenus in the menu hierarchy. **Example5:** Consider for instance:

```
figure();
h = uimenu('Label','Workspace');
uimenu(h,'Label','New Figure','Callback','figure');
uimenu(h,'Label','Save','Callback','save');
uimenu(h,'Label','Close','Callback','close',...
    'Separator','on','Accelerator','Q');
```

This code sequence adds a new menu item, namely “Workspace”, to the Matlab figure as illustrated in figure 3.

Figure 3

The menu handle is assigned to variable *h*, which is later used to create the “New Figure”, “Save” and “Close” menu elements. A horizontal line is drawn above the “Close” menu item by activating the *Separator* property. Additionally, the “Q” character is added as keyboard shortcut to the “Close” menu item within the *Accelerator* property. Finally, action of the menu elements is added in the *Callback* property as follows:

- ***figure*** – creates and opens a new Matlab widow,
- ***save*** – saves workspace variables to a file,
- ***close*** – closes the currently opened window.

The functionality of the code sequences can be customized according to the GUI designer’s needs.

Example6: In order to create submenus, the menus require dedicated handles. Consider for exemplification changing the definition of the “Save” menu item as follows:

```
figure();
h = uimenu('Label','Workspace');
uimenu(h,'Label','New Figure','Callback','figure');
h1=uimenu(h,'Label','Save','Callback','save');
uimenu(h,'Label','Close','Callback','close',...
        'Separator','on','Accelerator','Q');
uimenu(h1,'Label','Save
R','Callback','save(''R.mat'',''R'')');
uimenu(h1,'Label','Save
L','Callback','save(''R.mat'',''R'')');
uimenu(h1,'Label','Save All','Callback','save');
```

Accordingly, the Menu bar changes as illustrated in figure 4

Figure 4

Another example of creating a menu with submenus is presented below. To be noted that only the menu structure is done, other eventual functions and actions have to be afterwards added, using for example the open function. Save the code into an .m file and inspect the obtained menu.

```
clear all;
close all;
Fig=figure('Name','MATLAB Graphics Project',...
     'Units','normalized',...
     'Position',[0.2 0.2 0.5 0.5],...
     'NumberTitle','off','color',[0.3,0.5,0.5]);
f=uimenu('Label','Documentation');
uimenu(f,'Label','First Page','Callback','firstpage');

f1=uimenu(f,'Label','Circuits','Separator','on');
f1_1=uimenu(f1,'Label','Buck');
uimenu(f1_1,'Label','Circuit','Callback','');
uimenu(f1_1,'Label','Equations','Callback','');
f1_2=uimenu(f1,'Label','Boost');
uimenu(f1_2,'Label','Circuit','Callback','');
uimenu(f1_2,'Label','Equations','Callback','');
f1_3=uimenu(f1,'Label','Buck-Boost');
uimenu(f1_3,'Label','Circuit','Callback','');
uimenu(f1_3,'Label','Equations','Callback','');
f1_4=uimenu(f1,'Label','Flyback');
uimenu(f1_4,'Label','Circuit','Callback','');
uimenu(f1_4,'Label','Equations','Callback','');
```

```
f1_5=uimenu(f1,'Label','Forward');
uimenu(f1_5,'Label','Circuit','Callback','');
uimenu(f1_5,'Label','Equations','Callback','');
f1_6=uimenu(f1,'Label','Push-Pull');
uimenu(f1_6,'Label','Circuit','Callback','');
uimenu(f1_6,'Label','Equations','Callback','');
uimenu(f,'Label','Examples','Callback','',...
'Separator','on');
uimenu(f,'Label','References','Callback','references');
uimenu(f,'Label','Close','Callback','close',...
'Separator','on','Accelerator','Q');
```