UNIT 4

Graphics User Interface

1. Introduction

MATLAB handle graphics allows:

- High level commands for data representation. This includes two-dimensional and three-dimensional representations, photograph/image processing, development of special graphics for presentations (such as bar diagrams or "cheese" diagrams), and special effects (movies, lighting, camera movements).

- Low level commands for creating and modifying objects. This constitutes the so-called Graphics User Interface (GUI), which allows the development of complex applications, consisting of figures, menu bars and control objects (such as buttons).

2. Graphics User Interface (GUI)

MATLAB GUI is the set of tools and low level commands that allow the development of friendly applications consisting of figures, menu bars and control objects.

MATLAB objects present the following hierarchy:

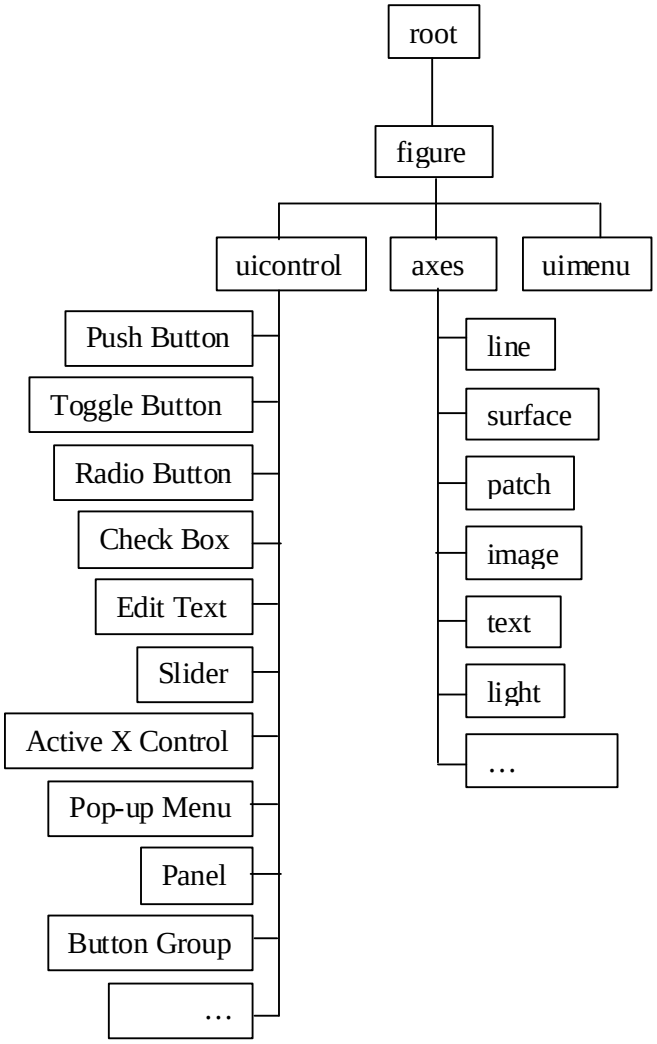


Fig. 1. Object hierarchy in MATLAB.

Objects: The **root** object is the command window. If **root** does not exist, none of the other objects shown in Fig.1 neither exists. In other words, if MATLAB is not active, it is not possible to generate any **figure**, any **axes**, etc.

When we type **>>plot(t,y)** in the **root** object, all objects needed for the representation are automatically generated, i.e., **figure** → **axes** → **line**, if they did not exist yet.

Handle and properties: Every object is associated to a "handle" (a number that identifies the object) and to a set of "properties" (**Color**, **Position**, etc.). Some properties can be modified by the user; others no. The handle for the **root** object is 0.

The object properties can be accessed from the low level text commands (**set**, **get**) and from the tools in the graphics editor **guide** (these tools are the *Property Inspector*, *Object Browser*, *Menu Editor*).

Versions beyond v6 include a good window-based editor for graphics properties and user-defined controls. However, the low level (text) commands are the same than in the previous versions. In the present course we present both the windows-based properties editor and the low level commands.

3. Low level commands: *set* and *get*

The two low-level text commands are **get** and **set**. The first one gives the value of one, some, or all the properties of an object and the second one modifies them (if it is possible).

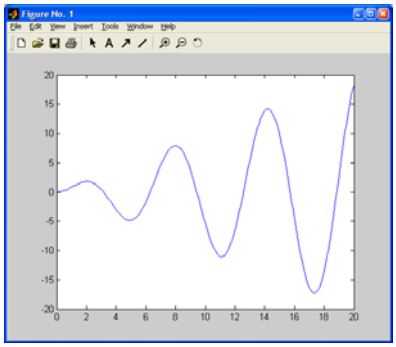
The command **set** can be also used to see which properties are modifiable and which are the valid values. For instance,

```
>> set(text)
```

Example 1. Objects, handles, and properties

1) Generate a time vector from 0 to 20*s* with samples uniformly spaced 0.1*s* and plot the curve $y = t\,\mathrm{sen}(t)$ using `plot`.

```
>> t=0:0.1:20;
>> y=t.*sin(t);plot(t,y)
```



2) Obtain the handle to the current figure object.

```
>> gcf
ans =
        1
```

Note: **gcf** stands for ***G**et handle to **C**urrent **F**igure*

Note: **gca** does the same that **gcf** but for an **axes** object and **gco** can be used for any type of object (the object has to be selected before).

```
» gca
ans =
  102.0009
» gco  %the line object has been clicked just before
ans =
    4.0002
```

Note: click now on the axes and see that effectively **gco** gives the handle to the axes object again

```
» gco
ans =
  102.0009
```

3) Obtain the 'Color' and 'Position' properties of the current figure:

```
» get(gcf,'color')
ans =
    0.8000    0.8000    0.8000
» get(gcf,'position')
ans =
   232   258   560   420
```

Note: The property names can be abbreviated (if there is no ambiguity possible). Hence, one can use, among others, the following expressions:

```
» get(gcf,'pos')
» get(gcf,'Posi')
```

4) Modify the 'Position' property. Notice the meaning of each one of the four vector elements

```
» set(gcf,'pos',[608 95 110 300])
```

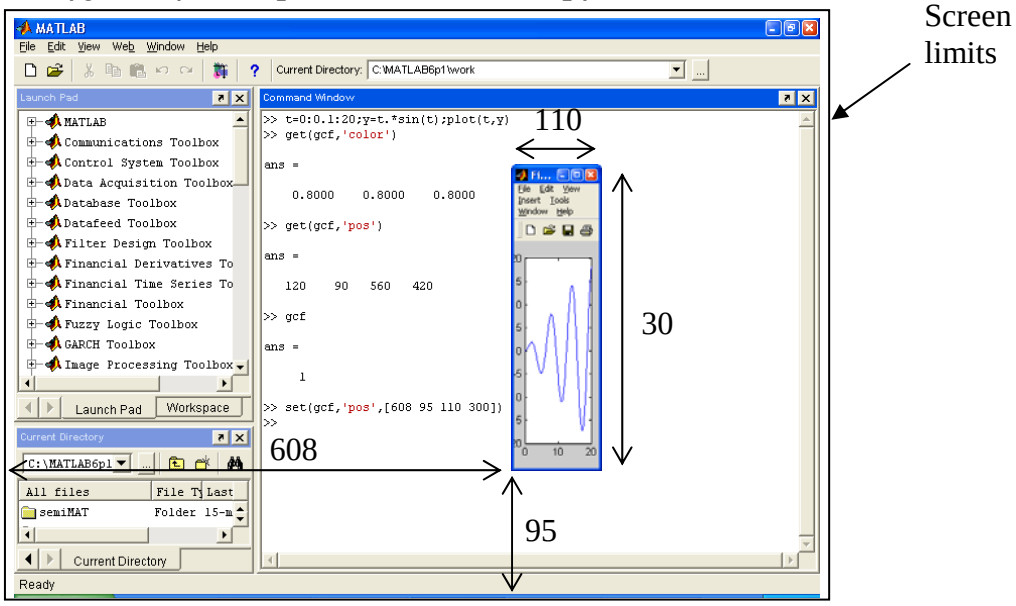


Fig. 2. Position property

5) See all the properties of current figure using **`get`**. Which are their 'Children'? Who is its 'Parent'?:

```
» get(gcf)
```

6) See all the modifiable properties using **`set`**. Which properties are not modifiable?

```
» set(gcf)
```

7) Close all figures:

```
» close all
```

and check that

```
» line
```

is equivalent to do:

```
» figure
» axes
» line
```

Suggestion: To see what each command does and how objects are numbered, repeat for the remaining graphic objects. For instance, you can try:

```
» uicontrol
» patch
» surface
```

etc.

4. Property Editor

Example 2. Property Editor

Plot again the curve $y = t\,\text{sen}(t)$ over a time vector from 0 to 20*s* spaced 0.1*s*.

```
>> t=0:0.1:20;y=t.*sin(t);plot(t,y)
```

Property Editor: Select in the figure the option **`Edit → Figure Properties...`** or alternatively **`View → Property editor`**. Note that if an object is selected in the figure, the corresponding property editor is opened below:

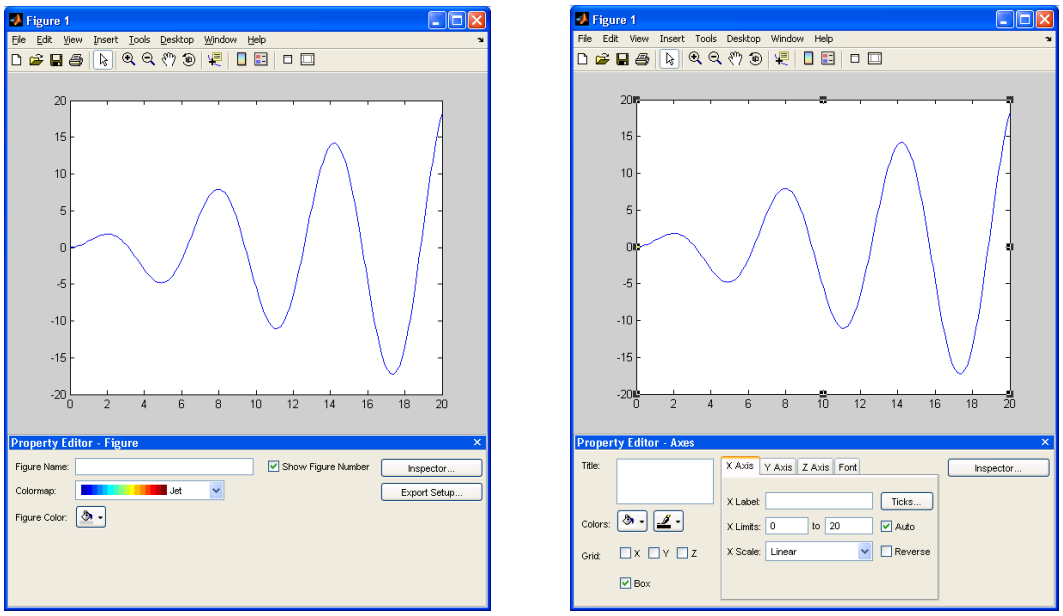
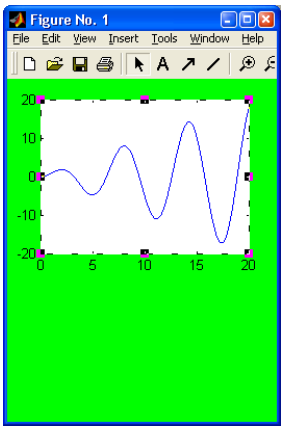


Fig. 3. Property editor

Example 3. Modification of the properties

Modify the background colour of the **`figure`** object to green (use the 

button).



Modify the size of the **`figure`** object by using the mouse.

To modify the **`axes`** object, select it with the mouse, too.

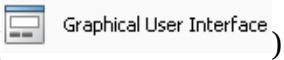
Save the figure (**`File→Save`**) as **`example.fig`** and close it.

To open it again, you can do: >> `openfig('example')`

---

5. Guide

In the version 7, click on the  icon in the main menu bar. In the version 8, go to the MATLAB toolstrip, then on the *Home* tab, in the *File* section, select *New* (

) and then *Graphical User Interface* (

). In the two versions it is also possible to type >>**guide** in the command window. All these actions open the *GUIDE Quick Start* window shown in the next figure.

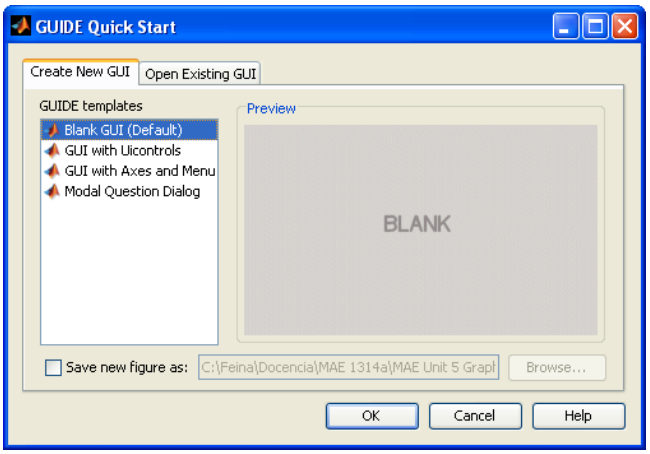


Fig. 4. Guide window

Let's illustrate the Guide usage by means of the figure saved in the previous section. To do so, from the *Guide Quick Start* window, go to the tab *Open existing GUI* and select the file **`example.fig`**. It is also possible to type >>`guide('example')`. (Note: If an axes icon appears with the legend "scribeOverlay" or similar, click on it and delete it).

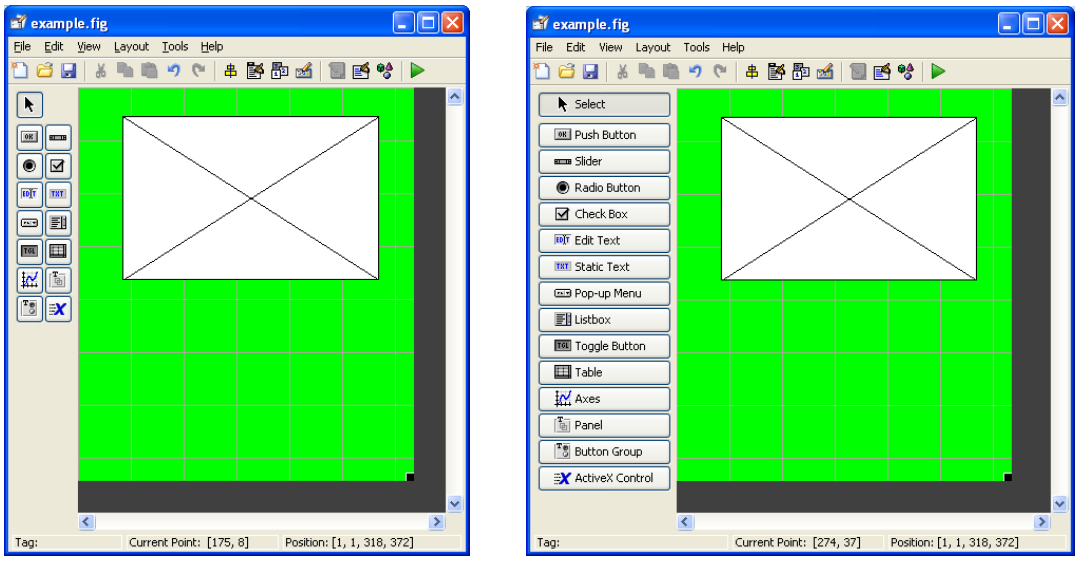


Fig. 5. Palette of objects

Palette of objects: By default only the icons appear in the object palette. If you want to see the names go to **`File→Preferences…`** and select **`Show names in component palette`**.

5.1 Using GUIDE

Object creating and aligning: Insert two push buttons and align them using the option (**`Tools → Align Objects...`**) or clicking to the corresponding icon  .

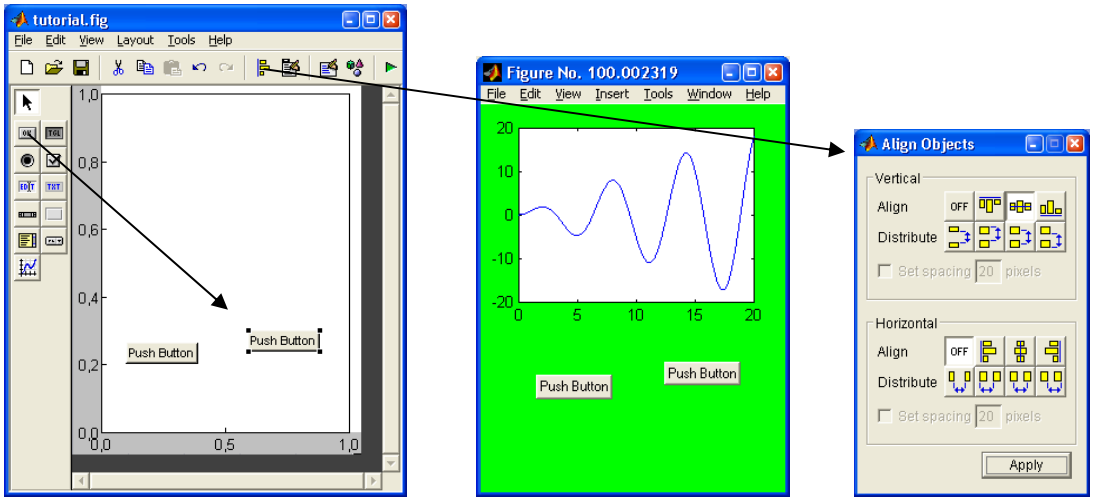


Fig. 6. Align objects (v6.x)

5.2 Object browser. Tag and String properties

Open the *Object Browser* clicking on  or selecting **`View→Object Browser`**. The object browser lists all the objects in the GUI (see Fig. 7).

Tag and String properties: Note that these two properties appear in the Object Browser.

The **`Tag`** property is an alias (name) that allows to refer to the objects by means of a name instead of using their handle number (more difficult to recall). In Fig. 7 the tags of the two pushbuttons are **`pushbutton1`** and **`pushbutton2`** respectively. These are the default names. It is common practice to change the tags for names much more related to the application, for example, **`close_button`**.

The **`String`** property refers to the text appearing in each object (it is shown inside " "). In the Fig. 7 the *String* in each pushbutton is **`Push Button`**. This is also the default string for the push button objects.

To view all the properties double-click on each one of the objects or select it in the object browser. These actions open the *Property Inspector*  .

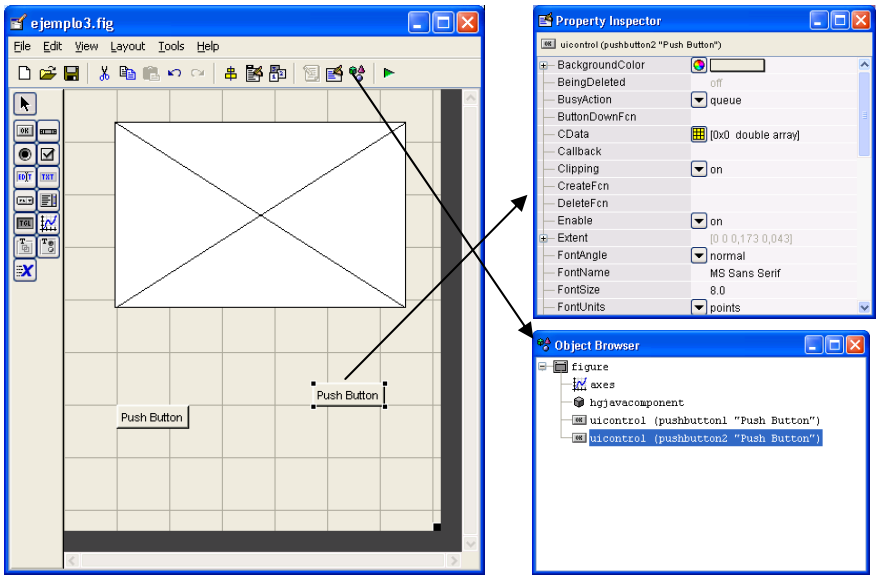


Fig. 7. Object Browser and Property Inspector (v7.x)

5.3 Property inspector. Changing properties

Open the *Property Inspector* by selecting **`View→Property Inspector`** or clicking on  or clicking on the object or clicking on the object name in the object browser. The property inspector shows all the properties associated to an object (see Fig. 7).

Modifying properties of an object (String and Callback): Select one of the two pushbuttons with the mouse or from the **`Object Browser`**.

Open the **`Property Inspector`**. Modify the **`String`** and **`CallBack`** properties by writing **`grid on`** in them both. Idem with the other pushbutton, but now write **`grid off`**.

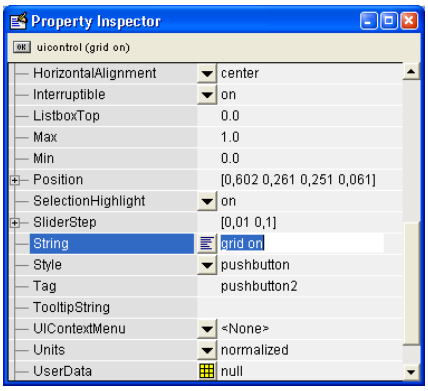


Fig. 8. Properties modification (v7.x)

The **`String`** property establishes the text in the button. The **`Callback`** property establishes which command or commands will be executed when the button is activated.

Run the figure with  or from the menu bar (**`Tools → Activate Figure`** or **`Tools → Run`**, depending on the version). Once it is activated it is already possible to execute the controls:

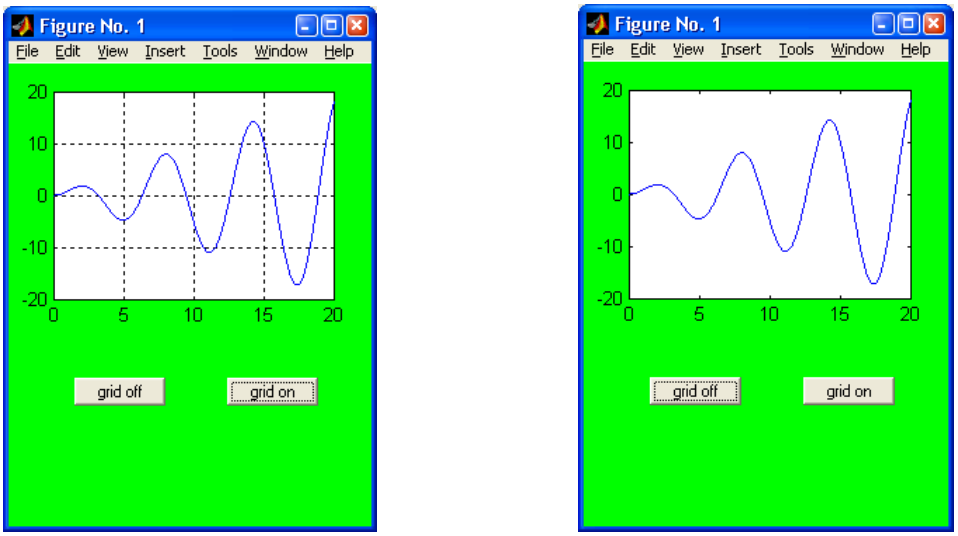


Fig. 9. Checking the GUI behavior

NextPlot Property: Sometimes, when you create a graphic object (for example, some axes) is desirable to change the default properties (change the color, the limits of the axes, to put a grid, ...). These axes object properties may be lost in situations where, in response to the activation of other controls, these axes will change the children objects represented (e.g., clicking a button shows a sine and clicking again a cosine). To avoid this we must change the property *NextPlot*. The possible values are:

- **`New`**: creates an new object (figure or axes)
- **`Add`**: you use the object as it is
- **`Replace`**: deletes all children objects (line, ...) and all the properties are reset except the position.
- **`Replacechildren`**: children objects are deleted but the properties are not reset

Other commands related to the updating of graphic objects are **`newplot`** and **`drawnow`**.

5.4 Menu editor. Findobj function

To open the menu editor click on the corresponding button  or select **`Tools → Menu Editor...`** from menu bar.

Following with the on-going example let us create a new menu bar with label **`grids`** and two submenus depending on it with the labels (and *callbacks*) **`grid on`** and **`grid off`**. This menu options will do the same as the pushbuttons.

Click on  to open the *Menu Editor*. Assign the value "grids" to the **`Label`** property and to the **`Tag`** property. **`Label`** is the name that will appear in the menu bar. Create two submenus (click on ) from menu "grid" with equal properties Label and Callback: **`grid on`** and **`grid off`**. Notice the use of symbols (→, ←, ↓, ↑) to establish the hierarchy.

Once the menu options are edited, close the editor, activate the figure and check its performance.

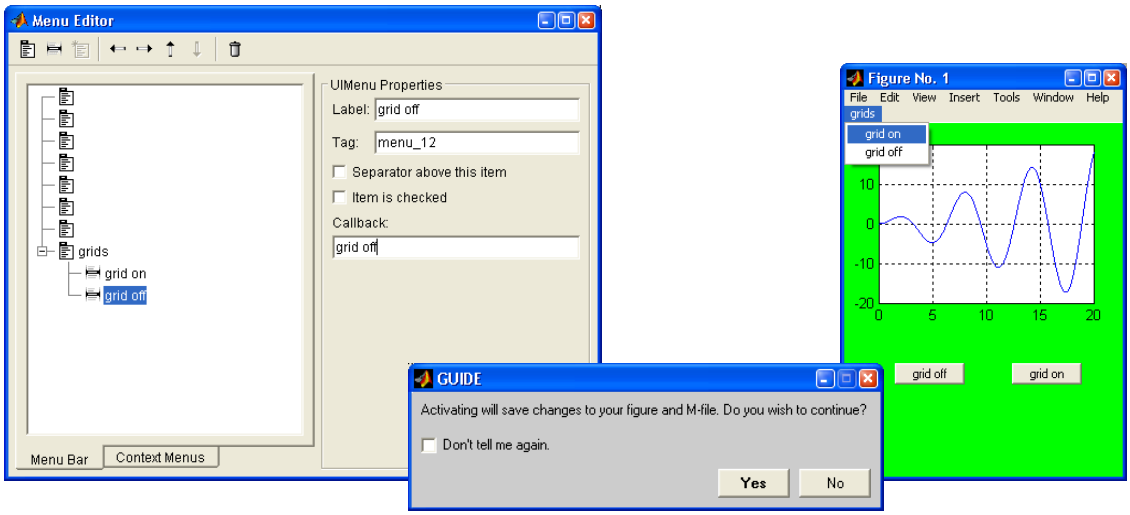


Fig. 10. Menu editor (v6.x)

Function findobj: A useful function is **`findobj`**, which gives the handle of all objects whom a specific property has a particular value. For instance, if we have several menu options and want to know which one is selected we can do:

```
h=findobj('Checked','on');
```

Property editor: Note that the new objects are already modifiable from the figure itself (**`Edit → Figure Properties`**) and it is not necessary to open the **`guide`** tool each time we need to change a property:

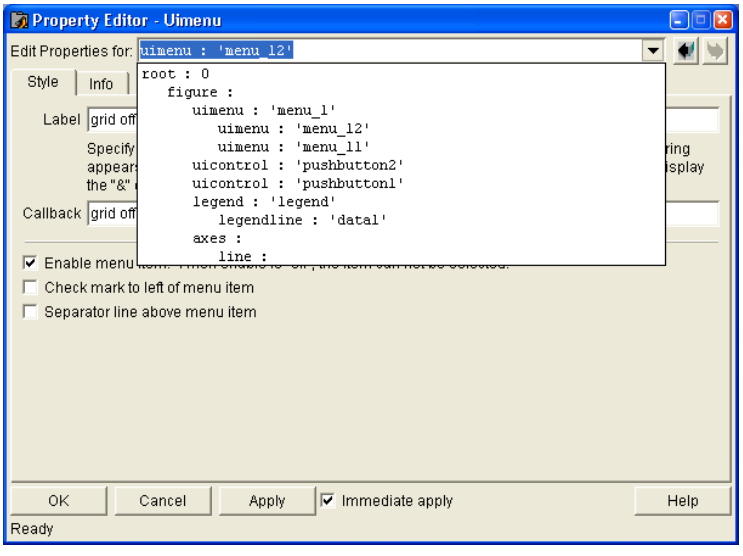


Fig. 11. Property editor in v6.x

Finally, if one wants to use the `guide` to edit a current figure or to insert new objects, it is possible to do: `>>guide(gcf)`.

6. Developing applications

The steps for the design of applications are:

1. Make a sketch of the appearance that should have the GUI and define its functionality.

2. Create the main window and drag all the objects to it. Align, distribute and sort the different objects.

3. Set the properties of the objects. Some of the typical properties to be set out in this stage of the design are: **`String`**, **`Tag`**, **`Color`**,… In the text boxes: **`BackgroundColor`**, **`foregroundColor`**, **`FontSize`**... In the panels: **`Title`**,... In the figures: **`Name`**,… In the axes: **`Xlim`**, **`ylim`**, ...

   The *Callback* property is used to attain the desired behavior. If the task to be executed when you activate a control is simple, you can put this task in the field Callback of the Property Inspector. For example, Callback = close or Callback = grid on.

   But when the task is complex (generate, manipulate and plot data, for example) is better to use the *. m file that is created by default when saving the file *. fig. Next section shows an example of how to edit the default *. m file.

   You can also use another custom made file *.m instead of the default one This latter option is illustrated in the Section 8.

4. Create menus and submenus and assign them the properties Label, Tag and Callback (if the callback is simple enough).

5. Edit the *.m file that controls the behavior of the application. We recommend checking the behavior of each object once its properties are edited.

6. Verify the behavior of the GUI as a whole.

7. Example (using the default M file)

7.1 Main figure and graphic objects

Guide: Open the GUIDE utility by typing **>>guide** in the command window o by clicking on the  icon in the toolbar.

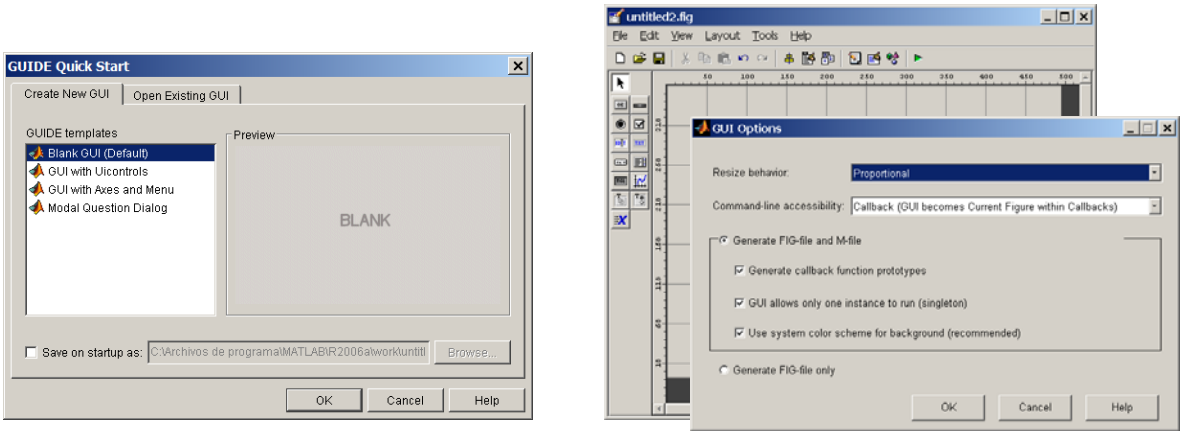


Fig. 12. GUIDE

Main figure: Select **Blank GUI (Default)** to create a new **figure**. In the figure menu bar you can select "*Tools → GUI Options*" to change the resizing property (the default value is *Non-resizable* but you can set it to *Proportional*). Note that it is also possible to generate the *.fig file only.

In the menu bar you can select "*File → Preferences… → GUIDE → Show names in component palette*" if you want to see the name of each object.

Objects: Insert different objects (for instance, two **axes**  , one **pop-up menu**  , one **panel**  , two **pushbuttons** ). Put the **pushbuttons** inside the **panel** and see how they are now a group that can be moved and resized altogether. If you drag and click on an **ActiveX**  control, a window with the **ActiveX** controls available in your computer will appear.

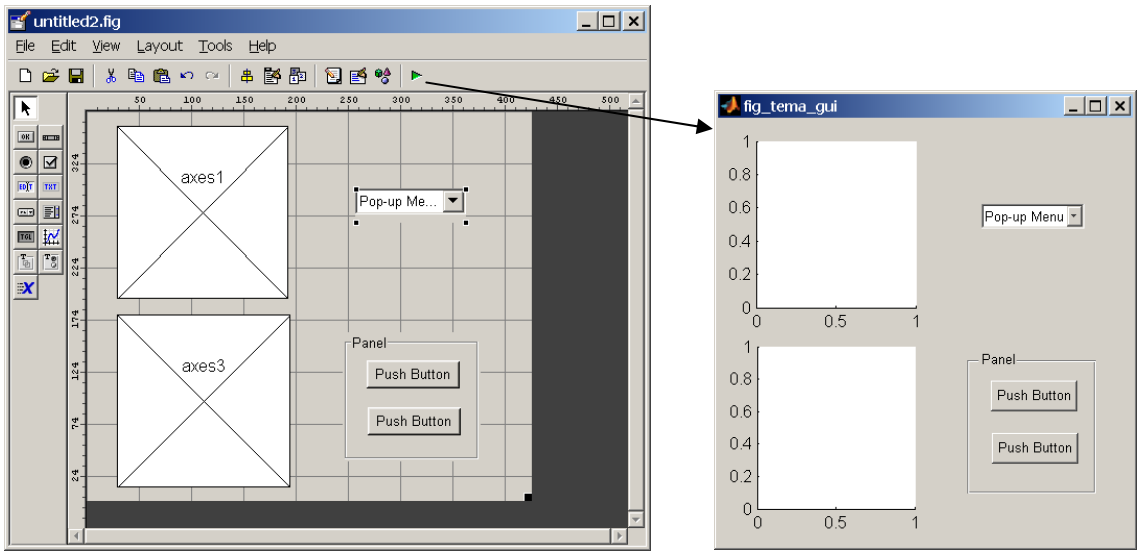


Fig. 13. Graphic objects

Click on and to access to the alignments, distribution and tab order tools.

Object properties: Open the **`Property Inspector`** of the following objects and set the properties listed in the next table:

Object	**`String`**	**`Title`**	**`Tag`**	**`Callback`**
Push button 1	grid		but_grid	
Push button 2	exit		but_exit	close
Uipanel 1		buttons		
Pop up menu 1	initial|balls|hat			

For instance, the *Property Inspector* of the second **`pushbutton`** is:

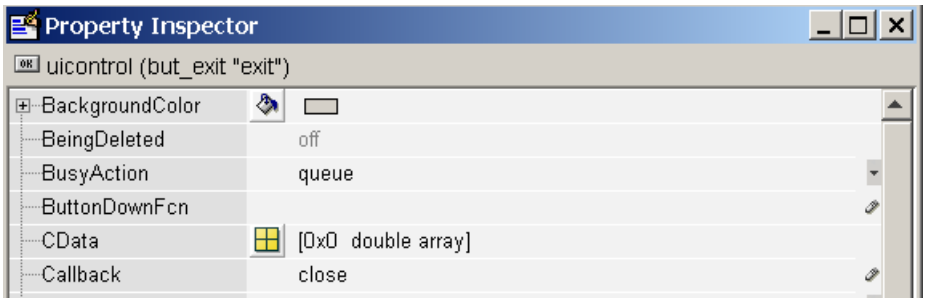


Fig. 14. Setting properties with the *Property Inspector*

To set the **`String`** property of the **`popupmenu`** object, click on the icon in its *Property Inspector* and list the desired labels.

Window activation and M file creation: Click on or select "*Tools→Run*". Choose a name for the window, for example, "**`demo_gui`**". By default two files are generated:

`demo_gui.fig` and **`demo_gui.m`**. The M file associated to the GUI can be edited from the GUIDE window. To do so, click on  or select *View→M-file Editor*.

The M file consists of different functions. To see them click on  in the M file editor toolbar.

For the ongoing example the M file generated by guide is the following (version 7.2):

```
function varargout = demo_gui(varargin)
% DEMO_GUI M-file for demo_gui.fig
%      help comments

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @demo_gui_OpeningFcn, ...
                   'gui_OutputFcn',  @demo_gui_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before demo_gui is made visible.
function demo_gui_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to demo_gui (see VARARGIN)

% Choose default command line output for demo_gui
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes demo_gui wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = demo_gui_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)
```

```
% hObject    handle to pushbutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes on selection change in popupmenu1.
function popupmenu1_Callback(hObject, eventdata, handles)
% hObject    handle to popupmenu1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = get(hObject,'String') returns popupmenu1 contents as
cell array
%        contents{get(hObject,'Value')} returns selected item from
popupmenu1


% --- Executes during object creation, after setting all properties.
function popupmenu1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to popupmenu1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

There are one initialization function and one termination function (functions **`demo_gui_OpeningFcn`** y **`demo_gui_OutputFcn`**). Then there are other functions corresponding to the **`Callbacks`** of the different objects in the GUI.

Note: There are two main types of Callbacks: (a) the user callbacks and (b) other callbacks depending on the particular object (for instance, CreateFcn, DeleteFcn, ButtonDownFcn, etc…). To see which callbacks are in the GUI, select *"View→View Callbacks"* in the guide window. It is not necessary to fill in all the callbacks of an object (sometimes they are already filled in with default initialization commands). The usual procedure is to work only with the user callbacks.

In a typical application the user can create additional functions inside the application M file. It is also possible to call the function **`demo_gui`** with additional input parameters (note the use of **`varargin`**).

Input arguments hObject, evendata and handles: All the inner functions inside the application M file pass information by means these input arguments. In particular, "handles" is a **`struct`** which fields are the tags of the graphic objects and the value of such fields is the associate handle number, for example:

```
handles =
       figure1: 189.0020
      uipanel1: 15.0029
    popupmenu1: 200.0020
         axes2: 195.0020
         axes1: 190.0020
      but_exit: 17.0026
      but_grid: 16.0026
```

The variable **`hObject`** contains the handle number of each object. For example

```
hObject =
  189.0021
```

The user can add more fields to the handles struct containing application data. Another possibility is to create a new struct to contain the application data and let the handles struct refer only to the graphic objects handles.

7.2 Application M file

Header: It contains some help comments and other initialization commands that need no modification.

```
function varargout = demo_gui(varargin)
% DEMO_GUI M-file for demo_gui.fig
%
%bla,bla,bla
%

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @demo_gui_OpeningFcn, ...
                   'gui_OutputFcn',  @demo_gui_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT
```

After the header we have some functions:

OpeningFcn: This is the first function to be executed. Note that it is possible to use additional input parameters in the GUI by means the **`varargin`** input argument.

In this function the user can generate (or load) some initial data. And, for instance, it is possible to plot the initial data.

In this function two commands exist by default:

```
handles.output = hObject;
guidata(hObject, handles);
```

The first one, `handles.output = hObject`, adds the "output" field to the handles struct containing the handle of the main figure, hObject. The resulting handles struct is:

```
handles =
       figure1: 189.0021
      uipanel1: 15.0033
    popupmenu1: 200.0021
         axes2: 195.0021
         axes1: 190.0021
      but_exit: 17.0028
      but_grid: 16.0028
        output: 189.0021
```

Note that **`handles.output`** (i.e., the application main figure window) is the minimal (by default) output of the application M file (see function **`OutputFcn`** next).

The second command, **`guidata(hObject, handles)`**, updates the handles struct. This is useful if we have generated new data and we have stored as new fields in the handles struct.

In the ongoing example we will use the **`OpeningFcn`** function to generate an initial plot. The commands in **`OpeningFcn`** are:

```
function demo_gui_OpeningFcn(hObject, eventdata, handles, varargin)
initial(handles);
handles.output = hObject;
guidata(hObject, handles);
```

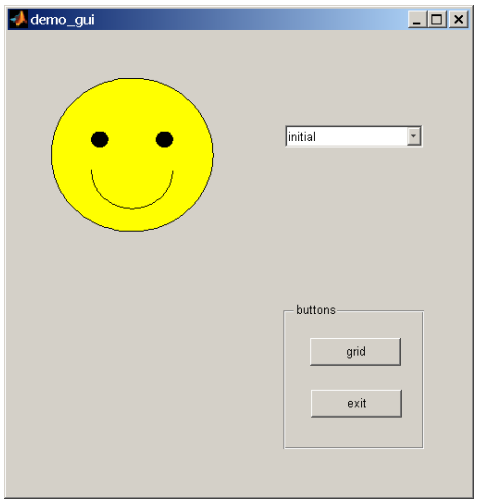
And the subroutine initial is defined (at the end of the application file) as:

```
function initial(handles)
newplot;
x=linspace(0,2*pi);y=exp(j*x);
fill(real(y),imag(y),'y','Parent',handles.axes1);
hold(handles.axes1,'on')
o1=y*0.1-0.4+j*0.2;o2=y*0.1+0.4+j*0.2;
patch(real(o1),imag(o1),'k','Parent',handles.axes1);
patch(real(o2),imag(o2),'k','Parent',handles.axes1);
x=linspace(-pi,0);y=exp(j*x)*0.5-0.2*j;
plot(real(y),imag(y),'k','Parent',handles.axes1)
axis(handles.axes1,'off'),axis(handles.axes2,'off'),
hold(handles.axes1,'off')
```

Note the use of the **`Parent`** property to set which **`axes`** is going to be used. Note also the use of **`hold`** and **`grid`** when we need to specify which **`axes`** we refer.

The **`newplot`** command is included for precaution, to clean the figure and avoid possible interaction between the different plots.

With this code, the initial execution of the GUI is the following:



Function OutputFcn: This is the output function. Note that the GUI can generate output data (additional to the handle to the main figure) by means the use of **`varargout`**. This function will not be edited in our example.

```
function varargout = demo_gui_OutputFcn(hObject, eventdata, handles)
varargout{1} = handles.output;
```

Now let us see the callback functions for the different **`uicontrols`**:

Function associated to the button but_grid: Here we write the commands to be executed when the **`but_grid`** is pushed (these commands could be entered by means the property inspector as well). Since the callback is the **`grid`** command, this means that clicking on this button will alternatively set and release the grid.

```
function but_grid_Callback(hObject, eventdata, handles)
grid(handles.axes1)
grid(handles.axes2)
```

Function associated to the pushbutton but_exit: Here we may write the commands to be executed when the **`but_exit`** pushbutton is activated. In this example we do not write anything since the callback for this pushbutton (the **`close`** command) is already entered in the property inspector of the object.

```
function but_exit_Callback(hObject, eventdata, handles)
```

Functions associated to the popup menu: They are two: **`callback`** and **`createFcn`**. We edit only the first one. This way, different selections on the popup menu will generate different data and plots.

```
function popupmenu1_Callback(hObject, eventdata, handles)
val=get(hObject,'Value');
str=get(hObject,'String');
switch str{val}
```

```
    case 'balls'
        [B1,B2]=balls;
        rep_balls(B1,B2,handles);
    case 'hat'
        S=hat;
        rep_hat(S,handles);
    case 'initial'
        initial(handles);
end

function popupmenu1_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

The last functions in the application M file are the custom functions that compute and plot the balls and the Mexican hat:

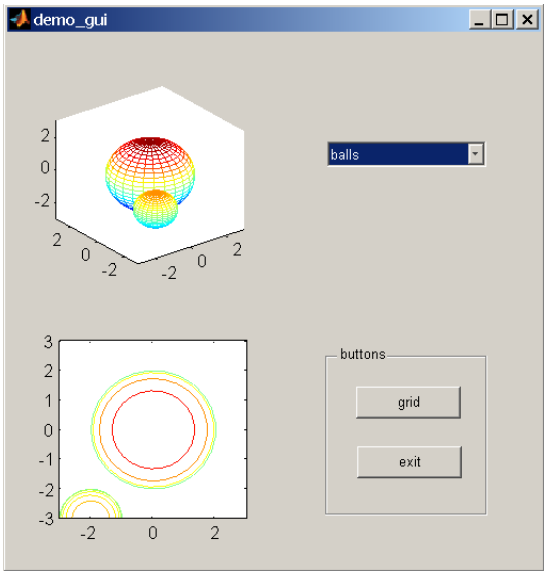
```
function [B1,B2]=balls
[x,y,z]=sphere(25);
x1=x-2;y1=y-3;z1=z;
x2=x*2;y2=y*2;z2=z*2;
B1={x1,y1,z1};B2={x2,y2,z2};

function S=hat
warning('off','MATLAB:divideByZero');
[x,y]=meshgrid(-10:0.5:10,-10:0.5:10);
z=sin(sqrt(x.^2+y.^2))./sqrt(x.^2+y.^2);
S={x,y,z};

function rep_balls(B1,B2,handles)
mesh(B1{1},B1{2},B1{3},'Parent',handles.axes1),
hold(handles.axes1,'on')
mesh(B2{1},B2{2},B2{3},'Parent',handles.axes1),
set(handles.axes1,'XLim',[-3 3],'YLim',[-3 3],'ZLim',[-3 3]);
hold(handles.axes1,'off');
%
contour(B1{1},B1{2},B1{3},'Parent',handles.axes2),
hold(handles.axes2,'on')
contour(B2{1},B2{2},B2{3},'Parent',handles.axes2),
set(handles.axes2,'XLim',[-3 3],'YLim',[-3 3]);
hold(handles.axes2,'off');
%
grid(handles.axes1,'off');grid(handles.axes2,'off');

function rep_hat(S,handles)
mesh(S{1},S{2},S{3},'Parent',handles.axes1);
contour(S{1},S{2},S{3},'Parent',handles.axes2),
grid(handles.axes1,'off');grid(handles.axes2,'off');
```

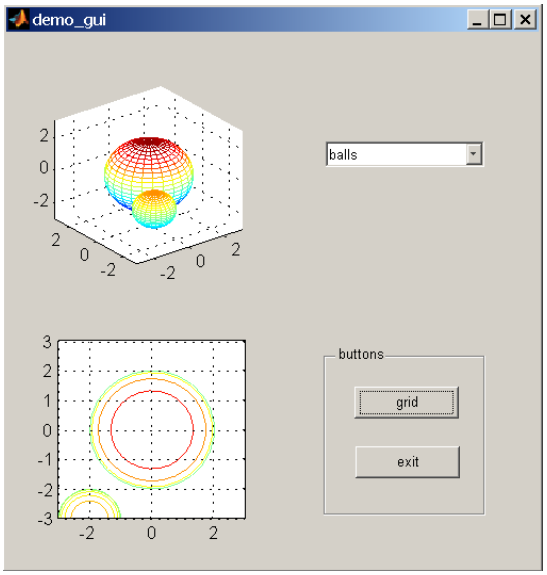
7.3 Running the application

To run the GUI simply type:

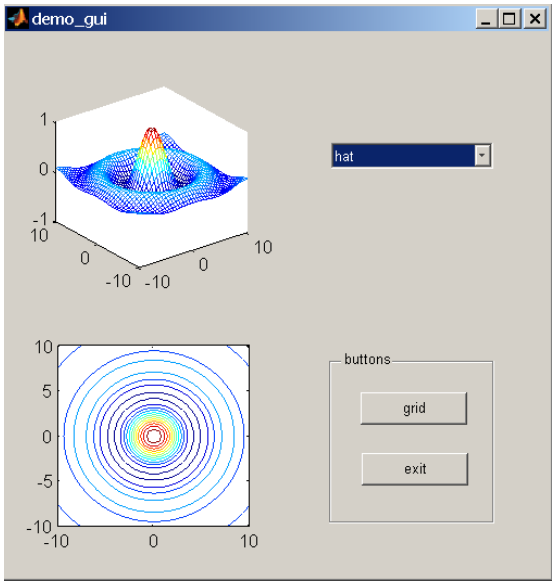
```
>> demo_gui
```

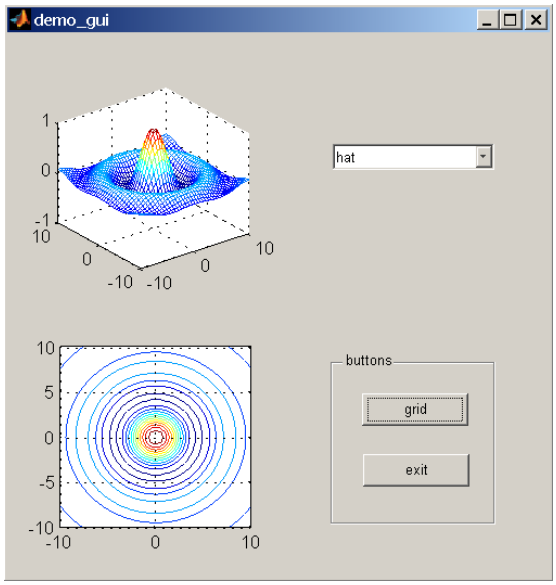
Next we show the result of selecting “balls” and then clicking on the “grid” button:





When we select “hat” we have:





And when you push the button “exit” the GUI is closed.

8. Example (using a custom made M file)

It is possible work with the *.fig file and a custom *.m file instead of using the application m.* file generated by default and explained in the previous section. There are many alternatives; here we illustrate one of them.

8.1 Main figure and graphic objects

Guide: Open the GUIDE utility by typing **>>guide** in the command window or by clicking on the icon  in the toolbar.

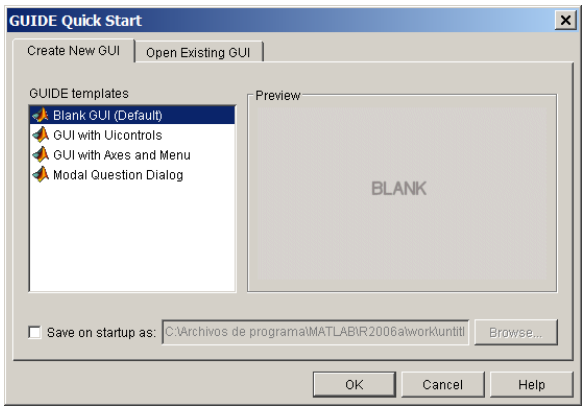


Fig. 15. GUIDE

Main figure: Select the option **Blank GUI (Default)** to create a new **figure** object. In the figure menu bar you can select "*Tools→GUI Options*" to activate the re-sizing property (the default value is *Non-resizable* but you can set it to *Proportional*).

In "*Tools → GUI Options*" you can select also the option "*Generate Fig-file only*".

Objects: Drag to the main figure the graphics objects shown in Fig. 16 (two **axes**  , three **push buttons**  , one **edit text**  and one **static text** ).

Object properties: Open the **Property Inspector**  of the following objects and set the properties:

Object	String	Tag	Callback
Edit text		freq	unit4('refresh')
Static text	Frequency (rad/s)		
Push button 1	grid		grid
Push button 2	EXIT	EXIT	unit4('exit')
Push button 3	load		[name,dir]=uigetfile('*.*')

See that in the *Object Browser* appear the Strings (inside " ") and Tags of the table above. Regarding the Callback property, this allows that a Matlab function will be executed when the object is activated. In this example, when we click on the **pushbutton2**, the subroutine **'exit'** of function **'unit4.m'** is executed (see the code of this function in the next section). And when we click on the **load** button a standard dialog box for selecting a file is opened.

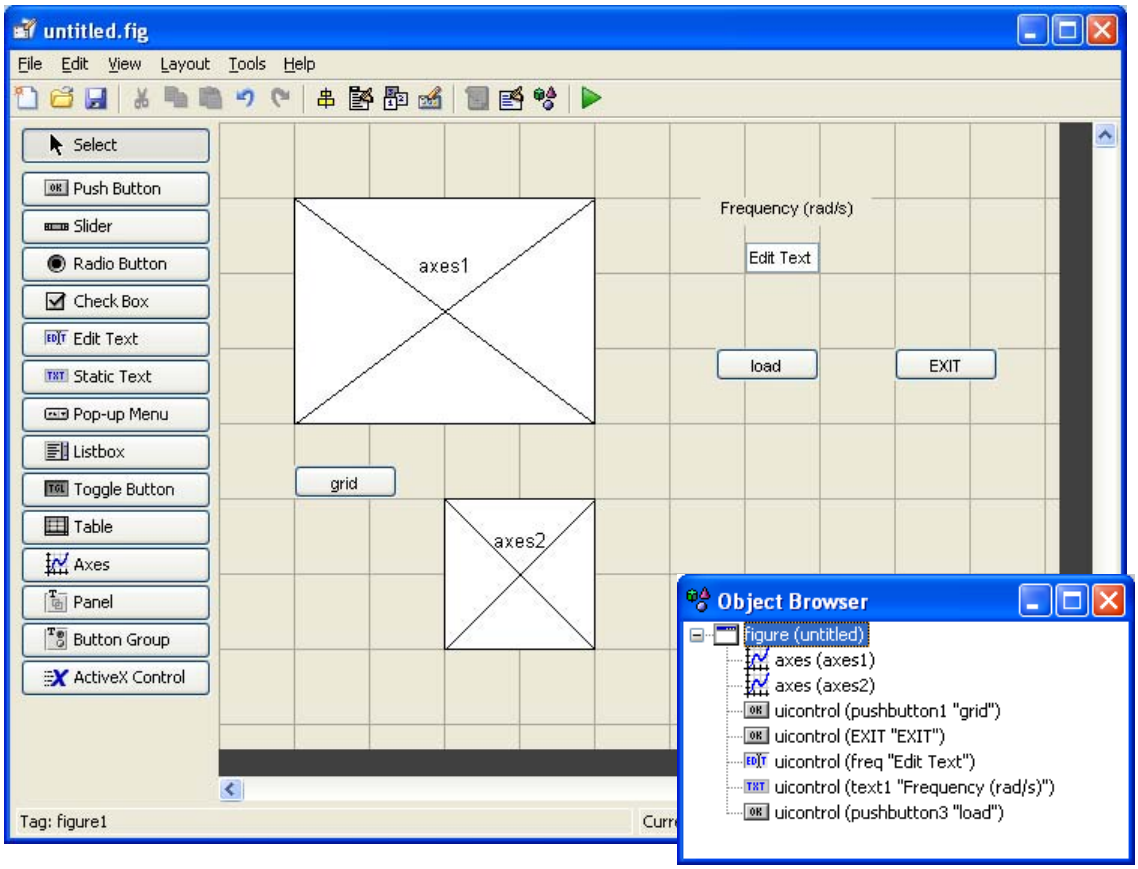


Fig. 16. Objects in the GUI

Save the GUI with the name **`fig_unit4`**.

8.2 M-file for the application. Callbacks

Let us ignore the **`fig_unit4.m`** file if it has been created by default. In this example we will use a custom M file to control the application. To do so, open the M file editor and edit the following function (unit4.m)

```
function unit4(action)

if nargin<1
    action='ini';
end

global ctrl

if strcmp(action,'ini')

    f=openfig('fig_unit4');
    ctrl=guihandles(f)

elseif strcmp(action,'refresh')

    freq=str2double(get(ctrl.freq,'string'))
    t=0:0.1:20;
```

```
    y=t.*sin(freq*t);
    plot(t,y,'Parent',ctrl.axes1)

elseif strcmp(action,'exit')

    close
    clear global ctrl

end
```

Note the conversion **`str2double`** for obtaining the numerical value **`freq`** and how we have said to object **`line`** in which **`axes`** must appear.

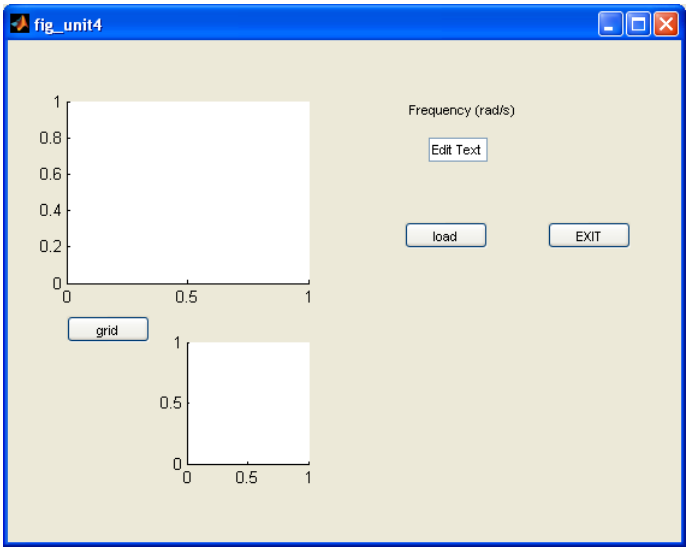
In this example, **`unit4.m`** is a function file but it is also possible to edit an application file with no input arguments, that is, a script file.

8.3 Running the application

To run the application simply type **`>>unit4`** in the command window or click on the run button of the m-file editor.

The function **`guihandles`**, **`ctrl=guihandles(f)`**, gives a struct data containing the handles to the graphic objects in the GUI. The fields of this struct are the object Tags.

```
>> unit4

ctrl =

        figure1: 198.0129
    pushbutton3: 18.0132
          text1: 17.0132
           freq: 16.0132
           EXIT: 15.0133
    pushbutton1: 221.0128
          axes2: 216.0128
          axes1: 199.0129
```



Write different frequency values in the *Edit text* box and see what it happens. Check the performance of the pushbuttons “grid”, “load” and “EXIT”.

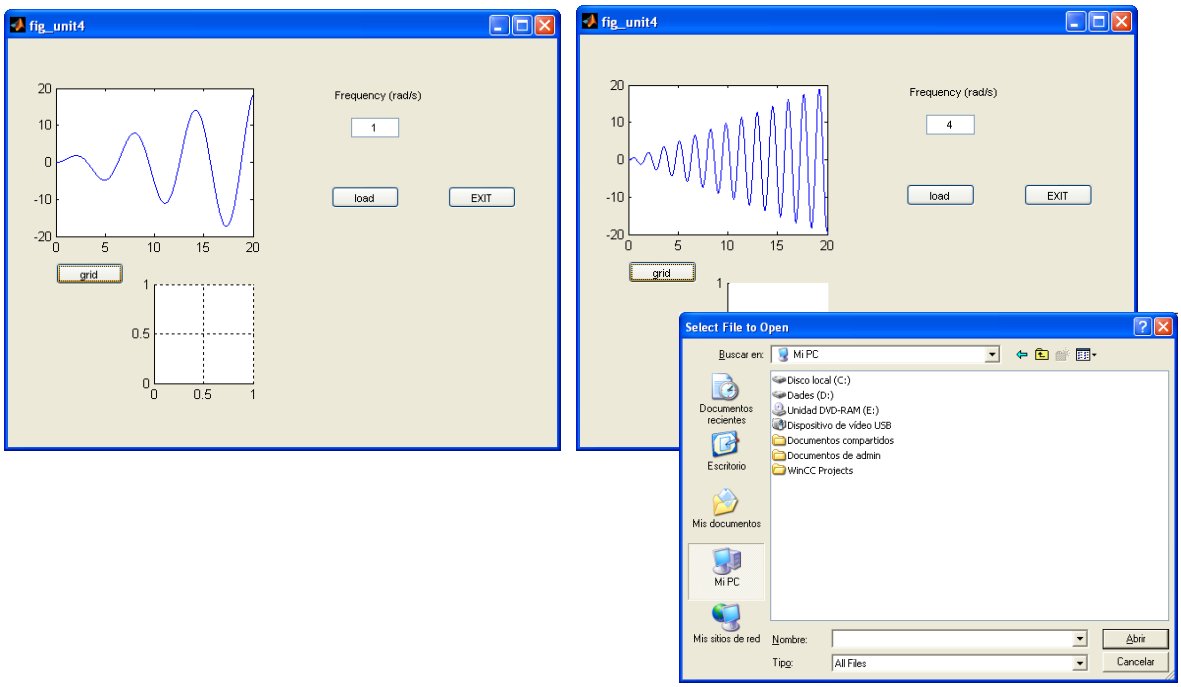


Fig. 17. Checking the GUI behavior