Digital Image Processing
Chapter 14: GUI using Matlab

Prepared by: Eng. Mohamed Hassan
Supervised by: Dr. Ashraf Aboshosha

http://www.icgst.com/A_Aboshosha.html

editor@icgst.com
Tel.: 0020-122-1804952
Fax.: 0020-2-24115475

Ashraf Aboshosha, www.icgst.com, editor@icgst.com
Introduction

- Graphical User Interface (GUI)
- MatLab provides Graphical User Interface Development Environment (GUIDE)
- A MatLab tool used to create GUI’s
- Decide between using GUIDE or writing the code from scratch
- GUI’s give the user a simplified experience running a program
- Associates a “function(s)” with components of the GUI
Essential Characteristics

- **Components** –
  - **Graphical Components**
    - pushbuttons, edit boxes, sliders, labels, menus, etc…
  - **Static Components**
    - Frames, text strings,…
  - Both are created using the function `uicontrol`

- **Figures** – components are contained in figures

- **Callbacks** – The functions which perform the required action when a component is pushed
Guide – building user interfaces interactively.

- **Guide** is an interactive tool for designing and building Graphical User Interfaces (GUI) for your Matlab applications.

- GUI building process involves;
  - (a) Designing of the user interface and layout (The looks!)
  - (b) Programming of the GUI and its components (The Works!)
  - (c) Testing, debugging and finally running it.
Starting up Guide

- Either click on the Guide Icon (ver 7+) on the Matlab-Toolbar or type `guide` on the command window.

- This will open the Quickstart GUI template selection window. Matlab provides a few templates to help with the common GUI design task but these are limited in numbers and you will probably need to start with a blank template.

- The initial layout area is usually resizable by clicking and dragging the handles on the corners of the template area.
Guide Layout

Alignment tool  Menu Editor  Toolbar editor  M-file editor  Property Inspector  RUN
Components Palette

- The L.H.S. of the Guide window contains the components palette. Your layout-design task will involve dragging and dropping the GUI control components from the palette onto the layout area.

- File-> Preferences option of the GUIDE window will allow you to display the names of the items in the components palette via the Guide: Preferences > Show Names in the components palette.
Designing the interface layout

- The user interface will usually be made up of:
  - Toolbars & Menus
  - Input Control components such as:
    - Push Buttons, Radio Buttons, Check Boxes (SWITCHES)
    - Pop-up Menus, List-Boxes (SELECTIONS)
    - Sliders (CONTINIOUS CONTROL)
    - Edit Text (TEXT method of input)
  - Graphical Objects
    - Axes Objects
  - Text Objects
    - Static Text
Designing the interface layout

- Matlab help pages on GUIDE give a lot of guidance on how to create a good design by making the layout simple and uncluttered. Refer to those pages for design hints.

- In the following slides, we shall only cover the technical issues on how to create a GUI interface and present a simple plotting GUI example.
Designing the interface layout

- The initial stages of the GUI design will involve selecting the control components we wish to use from the LHS palette and dragging and dropping them to a location of our choice on the layout area.

- Once dropped into the layout area, these components can be moved around and resized at will until a pleasant interface layout is achieved.

- It is usually aesthetically pleasing to use correctly proportioned controls, for example all Push-buttons being the same size. This can easily be achieved by right-clicking onto an existing control on the layout panel and selecting <duplicate> to ensure that multiple components have the same size and appearance.

- While a control item is selected; pressing DEL or using the Edit-Cut menu will delete that item. Also the Undo button can be used to recover from mistakes.
Designing the interface layout

- As well as getting the proportions of the control elements right, it is also important to get the alignment of the control elements correct and the alignment tool in the toolbar can be used for this purpose.

- All items in the layout panel that needs to be aligned with each other should first be selected as a group. This can be achieved by using the `shift + left_click` button on each item in sequence or by choosing the Select tool and clicking and dragging to capture the items in a rectangular area.

- Now use the alignment tool in the toolbar to align the currently selected items in the lay-out panel.

- When the align objects button is clicked the Align and Distribute Objects panel will be opened on which you will need to select the type of alignment/’smooth distribution’ required (i.e. vertical, horizontal, equal-spacing) and press the Apply button to carry out the selected alignment action.
Grids and Guide-lines

- Normally, the use of the alignment and distribution tool will be sufficient for your layout designs.
- Finer controls can be achieved by using the Grids, Rulers and Guide-lines as well as controlling the way objects can be snapped onto grid-lines.
- Rulers and Guide-lines can be activated via the Tools- >Grids and Rulers menu item.
- Once the Rulers are activated, guide-lines can be formed by clicking and dragging the rulers. Guide-lines do not exist in the created GUI. They are only there to make alignments easier by visual means.
Panels and Button Groups

- The items labelled Panels and Button Groups in the components palette are not in themselves control elements but are usually used for grouping together of the control elements.

- This is desirable when you design the GUI as made up of plots, graphs etc. that are controlled via various buttons etc located in a single ‘or a few’ control panel(s).

- A panel can be created in the usual manner by dragging and dropping a panel from the components pallet onto the layout panel.

- Following the creation of a panel, control items that are to be contained by that panel can simply be dragged and dropped onto the panel.

- A panel that contains control elements can now be moved, sized etc., thus treating all the control elements it contains as a single design item.

- Panels do not have to contain just the control elements, they can contain anything including axes elements and other panels. Used in this fashion, a GUI can be made to look like a multi component panel containing multiple graphs etc.
Button Groups

- Button Groups are like panels but their only real purpose is to group together the Radio Buttons and the Toggle Buttons.
- These are exclusive-selection items, where only one from a group of control-items will be allowed to be selected during the user interaction.
- **For example**: User will be asked to select one of the following three options for plot axis type (LINEAR AXIS), (LOG/LINEAR), (LOG/LOG).
- These options can be presented as three separate radio buttons (or toggle switches) contained within a Button-Group panel. If they were not contained in a button-group panel, these three controls would operate independently of each other making it possible to select any combinations of them which is not sensible.
- Whereas when they are contained in a button-group selection of one would automatically de-select the other two.
Adding Toolbars

- Toolbar Editor will allow adding toolbar items.

![Image of Toolbar Editor window]

- Toolbar Layout
- Tool Properties
- Tools Palette
Adding Menu items

Menu Editor

- File
  - Open
  - Close
- Edit
  - copy

Menu Properties

Label: copy
Tag: copy
Accelerator: Ctrl + None
Separator above this item
Check mark this item
Enable this item
Callback:

More Properties...

OK  Help
Context menus

Context menus can be used to define menus for objects which will pop-up when user right-clicks on it. These menus are defined using the menu-editor and then associated with object(s) by using the property editor and defining the Uicontextmenu Property.
Property Editor

- Having located the control-elements of your GUI in the layout panel, you can now customise the properties, which includes the appearance of these item.
- Each element in the layout panel is treated as an object that is contained in the figure which defines the GUI.
- Following from that, each object will possess a defined ‘and limited’ set of properties.
- Some of these properties determine the way that object looks, such as its colour, its title or the fonts used for text output associated with that object.
- There are also properties that determine the action that needs to be performed when a particular interaction is applied to an object, such as mouse-click on a button, slider-dragging with the mouse so on.
Saving your GUI

- Once you are happy with the layout of your GUI, you should save it for future use.
- Save the GUI using the `<file> <save>` menu or by clicking the save icon.
- Graphical User Interfaces generated by GUIDE are saved into two closely linked files—namely;
  - `your_gui_name.fig` and `your_gui_name.m`
- The figure (.fig) file contains all the information related to the layout and appearance.
- The script (.m) file contains all the programming logic aspects of the GUI.
- Most of the programming components of interest in the .m file will be contained in the set of function-stubs ready for use as call-back functions to various control objects.
- The task of programming the GUI will be one of finding the suitable code that need to be inserted into these call-back functions.
Some commonly used properties for programming the GUI Objects

- **String**
  - *Text or title to display* (e.g. button titles)

- **Tag**
  - *To identify the component during programming*

- **Callback**
  - *To specify the action to take when control activated* (e.g. what to do when mouse clicked on button)

- **Value**
  - *Indicates the state of the control object numerically* (e.g. position of the slider, button on/off)
String Properties

- String property of most of the GUI control items determine the titles to be printed on the screen to identify these controls.

- For buttons, check boxes, edit text, and static text, the String text is displayed on or next to the component. These titles will not normally be changed by the user during the execution of the gui.

- For pop-up menus and list-boxes all the items on the menu or the list-box should be entered as a column of $n$ number of strings where; $n$ is the number of options on the menu.

- For an edit text, the String property contains a list of strings that is displayed in the text box. When a user edits the text, the String property is updated.
Tags

- Tags are used for identifying the GUI components during programming the interface. Every object created on the GUI panel will have a tag property associated with it.

- By default, Guide gives simple descriptive values to these TAG properties, such as pushbutton1, slider3 so on and uses these values when it generates the call back functions in the auto-generated .m-file associated with that GUI.

- For easier understanding and programming of this m-file, we recommend that you define more meaningful tag values for the control components ‘buttons, sliders etc.’ of your GUI such as ‘rescale’, ‘convert’ etc.
Callback

- This is the most important property of the control objects.
- Callback property determines what action needs to be taken when the user activates that object (for example by clicking on the object, selecting an item from the list-box or choosing a menu item etc.) Most, if not all, the programming related to your GUI will be contained in the callback functions.
- Callback property will normally be set to the name of a function ‘as a string variable’ that needs to be invoked, but it can also contain a Matlab expression enclosed in a string (for simple tasks).
Programming your GUI

- After laying out the GUI and setting its component properties, the next step will be to program its behaviour.

- The code contained in the callback functions will control how the GUI responds to events such as button clicks, menu item selection, window resizing as well as creation and deletion of components.

- There will normally ‘but not always’ be one callback function per component on the GUI interface plus a couple of extra callback functions relating to the initialisation and output onto Matlab window. All these functions will be contained in a single .m file generated by GUIDE when you save your gui. (your_gui.m )
Callback function names

- Names of the callback functions for each gui component will also be automatically generated by GUIDE using the convention;
  
  `function <objects_tag>_<event_to_handle>`

- For example; the function that handles the listbox selection events for a listbox with tag `<listbox2>` will be named
  `function listbox2_Callback`

- A function that handles a key-press over a button with tag ‘redbox1’ will be named `function redbox1_KeyPressFcn`

- There will also be the following automatically generated callback functions namely;
  
  - `<guiname>_OpeningFcn`: For use in initialisation just before the gui is made visible.
  - `<guiname>_OutputFCn`: For use to output messages onto the Matlabs command output window.
Programming the callback functions

- Programming the behaviour of your gui will involve inserting your own Matlab code into the call back functions in the gui’s .m file.

- Initially GUIDE would have generated all the commonly needed callback functions but they will only be simple stubs, i.e. functions that contain very little or no executable code. Never-the-less this code is ready and functioning from the very start that can be run to confirm that all the mouse-clicks, slider-bars, pop-up menus etc. operate as expected.

- Without your own code not yet being in place, all your interactions will be generating calls to the corresponding callback functions that simply return control without doing any real work.
GUI .m file and the callback functions

- The .m file generated by GUIDE is made up of a list of callback functions.
- There is one main function that has the same name as your GUI and the rest of them are coded as the subfunctions of this main function.
- The main function is executed as your GUI is loaded. However the subfunctions are only executed if the user interactions invoke them via the callback properties of the objects.
- The main feature of subfunctions is the fact that they can only be invoked by the containing main function. This ensures that a call_back function designed for your GUI is not by accident called from somewhere else. This is why GUIDE uses subfunctions.
Transferring data between the callback functions

- Although all the callback functions are contained within the same .m file, they are semantically separate pieces of code that do not share any data except the data passed to them via the input parameters.

- To make the task of passing data easier, GUIDE inserts code for the creation of a handles structure that contains all the data for the GUI objects such as the control buttons, menus, axes.

- The handles structure (named: `handles`) is passed as an input parameter to all the callback functions that make up the GUI.

- Access to data is achieved by accessing the components of this `handles` structure using the rules relating to structures. See Matlab part II lecture notes.
Accessing the GUI data from the handles structure

- The **handles** structure contains fields that are named with the **tag** names of all the components in your gui. As these structure fields must be unique it is important that all the object ‘tags’ have unique names.

- For example if you have a radio_button which has the ‘tag’ property of ‘mediumwave’ then the handle of that radio_button will be accessible via the field `handles.mediumwave`.

- We can inspect, read and/or set all the properties of the above mentioned radio_button by using the general purpose Matlab `get()` and `set()` functions.

For example:

- **get** `(handles.mediumwave, ‘Value’)` will return whether the radio button is currently pressed (1.0) or not (0.0) Note that ‘Value’ is an important property of the radio button that indicates if it is currently in pressed position or not.

- To find out what properties an object has, use the command **get( handle_to_object )** which will list all the properties and the current values of that object. However the same can be done much more intuitively and easily by using the property inspector of GUIDE.
Altering the GUI data in the handles structure

- Most of the property related information stored in the handles structure can be altered by using the `get()` and `set()` combination of functions.

- Using the example of the radio button of the previous slide, if we decide to make this button invisible we can do so by setting its property named ‘Visible’ to string ‘off’ by using the `set` function in the following manner:

  ```
  set (handles.mediumwave , ‘Visible’ , ‘off’ )
  ```

- When any item stored in the handles structure is changed, ‘which means almost everything’ it is important that the handles structure is updated and the effect of these updated values on the GUI itself are implemented.

- This is done by invoking a very useful function named `guidata` as follows;

  ```
  guidata(hObject , handles ) ;
  ```

- We recommend that you routinely add this line to wherever there is a change to any data stored in handles.

- It is also possible to delete objects including the entire GUI window by using the `delete` command with the object handle. For example if the GUI’s ‘tag’ is ‘figure1’ then

  ```
  delete( handles.figure1 )
  ```

  will close the GUI. A useful feature for programming the ‘quit’ action.
Adding and accessing your own application related data in & out of the handles structure

- Although there are other ways of accessing data that is to be used/created within your gui, the recommended method remains to be via the handles structure.
- This is done very easily by simply defining new field in the ‘handles’ structure.
- For example, the following code section within any one of the call-back functions will create a new field named `mydata` and update the data structures to reflect this change.

```matlab
A = rand(10,10);
handles.mydata = A ;
guidata(hObject , handles ) ;
```

- Having stored the matrix A into the handles structure it can now be retrieved from within another callback function by simply;

```matlab
A = handles.mydata ;
```

- Another method of sharing data between the call_back functions can be by the use of the `persistent` and `global` declarations which will make data declared in this way available without having to store and retrieve to/from the handles structure. This is not a recommended method by MathWorks.
Analog input

An analog input component receives user-defined choices that are analog.

The GUI components in this class are:
- Edit text
- Slider
Discrete input

A *discrete input* component receives user-defined choices that are discrete (clear cut choices)
Discrete input

Discrete input components
- Radio button
- Checkbox
- Popup menu
- Listbox
- Push button
- Toggle button
The 10 styles of Matlab Uicontrol objects

- Push Button.
- Toggle Button.
- Check Box.
- Radio Button.
- Editable Text.
- List Box.
- Pop-up Menu.
- Slider.
- Frame.
- Static Text.
Edit Text

A component that allows the user to enter text – string or number.
Editable Text

- Used in situations that require the user to enter strings or characters or numbers. The strings, in turn, are used by the application for which the interface has been built.

- Clicking anywhere within this object will change the mouse from a pointer to a text insertion indicator.

>>mcedit, mceditf
Edit Text (Callback)

>>K=get(hObject,'string');

A variable. You can call it anything.

Retrieves the ‘string’ entered by user. This is a standard command.

If you entered ‘abc’ into the Edit Text, then the command above will make K = ‘abc’
At this point, K is a string. If the user enters a number, it will still be a string. To change it to a number,

```
>>K_num=str2double(K);
```

Another variable that you can call anything.
To pass the variable K for processing (to another Callback), type

```matlab
>>handles.K=K;
>>guidata(hObject,handles);
```

Another variable. But must be in the format ‘handles.anyname’

Standard command line used to ‘save’ all handles information.
Radio button

- A radio button acts as an option to be chosen by the user. The user clicks on the button to make the choice.
Radio Button

- Similar to the check box in that there are two states associated with each other.
- Usually two or more radio buttons are linked together as a group. They are linked in the sense that only one of the buttons will be in its selected state.

>>mcradiox
Radio button (Callback)

>> K = get(hObject, 'value');

'value' will be equal to 1 if the user selects the radio button. Otherwise it will be 0.
Checkbox

- Identical to the Radiobutton, except in a different form.
Check Box

- Useful for representing two states of an option that you may want to provide (usually as on and off).
- In its ‘off’ state the check box will consist of an empty or unfilled square. In the ‘on’ state, the check box’s square will contain a ‘V’ sign.

>>mccheckbox,mccheckbox1
Checkbox (Callback)

>>&K=get(hObject,'value');

'value' will be equal to 1 if the user selects the checkbox. Otherwise it will be 0.
A pop-up menu, when clicked, displays a list of options for the user to choose from.
Pop-up Menus

- Used in situations where multiple choices need to be available to the user.
- When the user clicks and holds the mouse button anywhere within the object, a list of choices appear.

>>mcpopup
Pop-up menu

To generate the list of options for the user

1. Press this button in the ‘String’ field

2. And this window will appear
Pop-up menu

- To generate the list of options

3. Type in the options, each separated by ‘Enter’
Pop-up menu (Callback)

\[ \text{>>K}=\text{get(hObject,'value')}; \]

‘value’ will indicate the index of the choice made.
Example: If the 2\textsuperscript{nd} choice is made, then ‘value’=2.
Listbox

- Identical to the Pop-up menu, except that the options are already visible without being clicked on. (To see options in Pop-up menu, need to click on it first).
List Boxes

- New style provided by MATLAB 5.x
- Very similar to pop-up menus.
- The main difference with a list box is that you can make the set of options visible to the user at all times.

```plaintext
>> mccheckbox
```
Pushbutton

- A push-button is one that will execute a series of commands when pushed.

- No specific Callback commands. Just type in the commands you want to execute.
Push/Toggle Buttons

- The push button is widely prevalent uicontrol style that is used primarily to indicate that a desired action should immediately take place.
- The toggle button look just like push button, except there is no intermediate state. Rather, the button will remain in its selected or not selected state after the user clicks on it.

>>mcpush
Toggle button

Gives a state when pressed.
Toggle button (Callback)

>>K=get(hObject,'value');

'value' will be equal to 1 if the toggle button is pressed. Otherwise it will be 0.
Sliders

- A component that allows the user to select a value from the slider, between the pre-specified minimum and maximum.
Sliders

- Useful in representing a fixed range of values from which to choose.
- The slider has no way of explicitly indicating the numeric value that the slider represents. Therefore, it is recommended that an editable text or static text style uicontrol accompany the slider.

```matlab
>> mcslider, mcslider2
```
Slider

The maximum and minimum values of the slider can be set in the Property Inspector.
Slider (Callback)

>> K = get(hObject, 'value');

>> handles.K = K;

>> guidata(hObject, handles);

This is a standard command. A slider will always return a numerical value, that's why the label 'value'.

Reminder: These commands are used if you want the value of 'K' to be passed to other Callbacks for further processing.
Frames

- Provide a solid background that helps blend a set of uicontrols into one complete and cohesive interface.
- Used as an effective method of organizing the GUI in a logical and intuitive fashion.
Static text

- Allows you to type text on your GUI.

- Also allows you to output text depending on the user’s actions.
Static Text

- Available for creating labels, status messages or other information pertinent to the user.
- Static text does not perform any action if the user clicks on any part of the object. In addition, the user can not edit the information that is displayed.
Static text

- No Callback generated for Static Text.
- To output ‘user-dependent’ text, type

```matlab
>>set(handles.StaticText,'string','Your text');
```

The Tag of the Static Text (you assigned). Format is `handles.Tag`

Text to output
Properties that Track User Actions

- **ButtonDownFcn** - When clicking the mouse button while the pointer is located over or near the object.
- **WindowButtonDownFcn** - When clicking the mouse button down within the figure boundaries.
- **WindowButtonUpFcn** - When clicking the mouse button up within the figure boundaries.
- **WindowButtonMotionFcn** - When the mouse pointer moves within the figure boundaries.
- **KeyboardFcn** - When the figure is active.
- **CreatFcn** - When creating an object.
- **DeleteFcn** - When deleting an object.
- **ResizeFcn** - When resizing the figure.
Display output

A *display output* component is used to show the result of the GUI calculation/processing.

The GUI components in this class are:
- Axes
- Static text

They have no Callbacks.
Some useful instructions.

- AXES(axes_handle) - make the axes, current.
- Object_H=GCBO - to get the object that make the callback.
- RBUTTON(Radio_H) - use to select and deselect radio buttons with the same Tag Name.
Axes

- Allows the user to plot graphs
Axes

No Callback generated for Axes.

To plot a graph, type

```
>> axes(handles.axes1);
```

The Tag of the axes (you assigned). Format is handles.Tag

You have now referred to a specific graph. Now type whatever plotting commands you wish to.
Extra: Retrieving data

- You can also retrieve data from another component without that Callback sending out the data (see example on Edit Text and Slider)

- e.g. – you want to retrieve the data (choice) from a Listbox
Extra: Retrieving data

In the Listbox Callback

```matlab
>> K = get(hObject, 'value');
>> K = handles.K;
>> guidata(hObject, handles);
```

Then in the Pushbutton Callback

```matlab
>> K = handles.K;
>> guidata(hObject, handles);
```
Extra: Retrieving data

You don’t have to do that!
There is an alternative!

In the Pushbutton Callback

```>> get(handles.tag, 'value');```

The tag of the Listbox

The value returned by the Listbox
Setting figure.

- \( h0 = \text{figure('Color',[0.8 0.8 0.8],...'
  'Units','Normal',...
  'Position',[0.3 0.3 0.3 0.3],...
  'Tag','Fig1');?>

- \%Setting figure name
- \text{set}(h0,'Name','Check and radio');?>
Setting Push Button.

- \( h1 = \text{uicontrol ('Parent',h0, 'Units','Normal','Position',[0.1 0.3 0.3 0.4] , 'Tag','Pushbutton1'); } \)
- \%Setting the callBack Function
  - set(h1,'Callback','Button_1_callBack_Function');
- \%Setting the string on the button
  - set(h1,'String','Push.B_1');
- \%Setting Tool Tip String
  - set(h1,'TooltipString','Push Me To Call The CallBack Function!');
- \%Setting The Text Color on the button (the foreground color)
  - set(h1,'ForegroundColor',[0.0,0.0,0.0]);
- \%setting the background color
  - set(h1,'BackgroundColor',[0.0,1.0,0.0]);
Edit - Box call back function.

- `EditBox_H=GCBO;`
- `EditBoxString=get (EditBox_H,'String')`
Example 1

Create a GUI that says ‘Hello’ when you press a Push button.
Example 1 (solution)

- Start GUIDE
- Drag and drop a Push Button to the front panel
Example 1 (solution)
Example 1 (solution)

- Double click on the Push Button to change its properties (String and Tag)
Example 1 (solution)

The ‘String’ field is what will be seen by the GUI user. You can change it to e.g. ‘Say Hello’

The ‘Tag’ is the name that Matlab recognizes for this component. It is recommended that you change it to e.g. ‘PB_hello’
Example 1 (solution)

- You’ve finished designing the front panel.
- Now program the Callback (in an mfile).
Example 1 (solution)

Press this to open the ‘mfile editor’ and program the Callback.

PS – be prepared to give it a filename to save
Example 1 (solution)

- Now look for a function ‘PB_hello_Callback’.

  The mfile automatically creates a function using the name that you gave the component (PB_hello) and adds ‘Callback’ behind it.

- Under the function, write the commands that you want it to execute.
Example 1 (solution)

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

disp('Hello')
```

This is the command that you have typed.
Example 1 (solution)

- Now you’re ready to go!
- Execute the GUI

Press to execute
Example 1 (solution)

- It will appear as a Figure.
Example 1 (solution)

- The result

Displays ‘Hello’ as commanded
Example 1 (solution)

- The result

Displays ‘Hello’ as commanded
Example 2

Create a GUI that plots the function entered by the user when the user pushes a Push Button

- The user can enter the upper and lower bounds of the x-axis, and the resolution.
- A pop-up menu enables the user to choose the graph colour – blue (default), red, green or yellow.
Example 2 (solution)
Example 2 (solution)

- Now let’s look at how to program the Callbacks

- It can all be done within the Push Button Callback
Example 2 (solution)

```matlab
>> lowbound = get(handles.Edtxt_lowbound,'string');

Retrieve data

Tag of the Edit Text box which enters the lower bound of the x-axis

>> lowbound_no = str2double(lowbound);

Converts the string to a numeric value
```
Example 2 (solution)

Do the same for the
- Upper bound
- Interval
- Function entered by the user (but no need to convert to number)
Example 2 (solution)

>> for i=1:((upbound_no-lowbound_no)/interval_no)+1;
>> u(i)=lowbound_no+interval_no*(i-1);
>> x=u(i);
>> y(i)=eval(graph_func);
>> end

>> hline=plot(u,y);

Data retrieved by the graph function

Giving the plotted line a name, to set properties later
Example 2 (solution)

```matlab
>> 1+1
This will give a number 2

>> '1+1'
This will still give a string '1+1'

>> eval('1+1')
This will give a number 2

>> x=3; eval('x+4')
This will give a number 7
```
Example 2 (solution)

```matlab
>> linecolour = get(handles.Popup_colour, 'value');
>> linestyle = get(handles.Listbox_style, 'value');
```

Retrieve data on what user specified with regard to line colour and style
Example 2 (solution)

```matlab
>> switch linecolour
   >> case 1
   >>    set(hline,'color','b');
   >> case 2
   >>    set(hline,'color','r');
   >> case 3
   >>    set(hline,'color','y');
>> end
```

This is why we gave the line a name earlier.

A property field (standard command) of the line.
Example 2 (solution)

```matlab
>> switch linestyle
>>     case 1
>>         set(hline,'linestyle','-');
>>     case 2
>>         set(hline,'linestyle',':');
>>     case 3
>>         set(hline,'linestyle','--');
>> end
```
Example 2 (solution)

```matlab
>>grid_on=get(handles.Radio_grid,'value');
>>if grid_on==1
    grid on;
>>else
    grid off;
>>end
```

If the Radio button is clicked, it will return a value of 1.
Example 2 (solution)

Now, test the GUI
- Enter the function in terms of ‘x’
- Enter the lower bound, upper bound, and interval
- Press the Plot It button
- Make other modifications in terms of line colour, style and grid
Example 3

Create a GUI with the following features

- Calculates the volume of a cuboid when the user enters the height, width and depth
- Outputs the result in a Static Text box
Example 3 (solution)

Tag – Edtxt_height
Tag – Edtxt_width
Tag – Edtxt_depth
Tag – Pb_calculate
Tag – Sttxt_result
Example 3 (solution)

In the Edtxt_height Callback

```matlab
>>height=get(hObject,'String');
>>height_num=str2double(height);
>>handles.height_num=height_num;
>>guidata(hObject, handles);
```
Example 3 (solution)

In the Edtxt_width Callback

```matlab
>> width = get(hObject, 'String');
>> width_num = str2double(width);
>> handles.width_num = width_num;
>> guidata(hObject, handles);
```
Example 3 (solution)

In the Edtxt_depth Callback

```matlab
>>depth=get(hObject,'String');
>>depth_num=str2double(depth);
>>handles.depth_num=depth_num;
>>guidata(hObject, handles);
```
Example 3 (solution)

In the PB_calculate Callback

```matlab
>> depth_num = handles.depth_num;
>> width_num = handles.width_num;
>> height_num = handles.height_num;

>> result = depth_num * width_num * height_num
>> result_str = num2str(result)         
>> set(handles.Sttxt_result,'string',result_str)
```

Convert the number to a string first before sending out to the Static Text.
Message boxes

There are 3 classes of message boxes

- Error messages
- Warning messages
- Help messages

All have same functionality except for icon
Error message

>>errordlg('message','title')

![Error message dialog box](image)
Error message

An example

>>errordlg(‘You have entered a wrong sign!’,’Wrong sign’)
Warning message

>>warndlg('message','title')
Help message

>>helpdlg('message','title')
Summary of message boxes

All message boxes have the same function
Only the icon is different
Dialog boxes

Dialog boxes will ask the user for input
1. Input dialog – user enters analog input
2. Question dialog – user enters discrete input of ‘Yes’, ‘No’ or ‘Cancel’
Input dialog

An input dialog box enables the user to enter (analog) inputs for processing.
Input dialog

To generate the dialog box previously

```python
>>prompt={'Height','Width','Depth'}
```
Input dialog

>>name={'Calculate the volume of a cuboid'}
Input dialog

>>numlines=1
Input dialog

>>defaultanswer={'10','10','10'}
Input dialog

>>options.Resize=on

Not Matlab specific. Can Call any name

Specific Matlab name. Cannot call anything.

Enables user to resize the dialog box
Input dialog

Finally, type

```matlab
>>answer=inputdlg(prompt,name,numlines,defaultanswer,options)
```

‘answer’ will return the user inputs
‘answer’ will be a cell array with 3 elements
Input dialog

If the user enters
Height=20; Width=30; Depth=40

Matlab will return

answer =
'20'
'30'
'40'
Input dialog

To access the Height
>>answer{1}
To access the Width
>>answer{2}
To access the Depth
>>answer{3}

All will be in strings
Input dialog

1. ‘Prompt’ and ‘Defaultanswer’ fields must have the same number of elements.
2. ‘Answer’ will have same number of elements as ‘Prompt’.
A question dialog enables the user to enter an input of ‘Yes’, ‘No’ or ‘Cancel’
Question dialog

To generate the question dialog

```matlab
>> button = questdlg('Do you wish to go back?', 'Wrong number')
```

Any name you wish
Question dialog

If user clicks ‘Yes’, then button=‘Yes’
If user clicks ‘No’, then button=‘No’
If user clicks ‘Cancel’, then button=‘Cancel’

To user result for processing, use the command ‘strcmp’
Question dialog

>>strcmp(button, ‘Yes’)

Returns value of ‘1’ if button=‘Yes’. Returns value of ‘0’ if not.
Question dialog

But …. you are not constrained to just ‘Yes’, ‘No’ or ‘Cancel’
You can enter up to 3 custom buttons

>>button=questdlg('Message', 'Title', 'Button1', 'Button2', 'Button3', 'Default')

The last argument is always the default answer
Example 4

See Example 3. Modify it to generate an error message if any of the user inputs (height, width or depth) are negative.
Example 4 (solution)

Open solution to example 3
Rename it first
Front panel does not need to change
Only change code
Example 4 (solution)

>>if depth_num<0 || width_num<0 || height_num<0;

>>errordlg('You have entered a negative number! The result is invalid. Please re-enter!','Error!');
>>set(handles.Sttxt_result,'string','Invalid');

>>else

>>result=depth_num*width_num*height_num;
>>result_str=num2str(result);
>>set(handles.Sttxt_result,'string',result_str);

>>end

Outputs the result as ‘invalid’

These 3 lines the same as before
Example 4 (solution)

Now test your GUI!

Enter some negative numbers and see the result!
Example 5

Now modify example 4. When the user enters a negative number

- Generate the error message. Ask user if wish to re-enter or close the GUI.
- If wish to re-enter, let user enter using an Input Dialog
- Keep repeating if user keeps entering negative numbers
Example 5 (outline solution)

>>while (statement of violation)
>>  (ask if user wants to re-enter, Yes or No)
>>
>>  if (user says ‘No’)
>>     (break ‘while’ loop and close GUI)
>>  else (user says ‘Yes’)
>>     (user Input dialog for user to re-enter)
>>     (obtain answers of Input dialog)
>>  end (end ‘if’ loop)
>>
>>  >>end (exit ‘while’ loop)
Example 5

Obtain initial user input

```matlab
>>depth_num=handles.depth_num;
>>width_num=handles.width_num;
>>height_num=handles.height_num;
```
Example 5

Start ‘while’ loop

```matlab
>>while depth_num<0 || width_num<0 || height_num<0

>>answer=questdlg('Do you want to re-enter?',
'Error! Negative number entered!','Yes','No','Yes');
```

3 arguments entered after first 2 arguments. Means there are 2 options, where ‘Yes’ is the default.
Example 5

If user says ‘No’

```matlab
>> if strcmp(answer,'No');
>>     close
>>     breaker=1;
>>     break
```

- Close the GUI
- Logic variable to close GUI without error messages
- Exit the ‘while’ loop even violation still occurs
Example 5

If user says ‘Yes’

```matlab
>> else
>>     prompt={"height","width","depth"};
>>     name='Please re-enter the height, width and depth';
>>     numlines=1;
>>     defaultanswer={'0','0','0'};
>>     options.Resize='on';

>>     answer_reenter=inputdlg(prompt,name,numlines,
                 defaultanswer,options);
>>     breaker=0;
```
Example 5

Obtain answers from user re-enter

```matlab
>> height=answer_reenter{1};
>> width=answer_reenter{2};
>> depth=answer_reenter{3};
>> height_num=str2double(height);
>> width_num=str2double(width);
>> depth_num=str2double(depth);
>>end
>>end
```

End ‘if’ loop

End ‘while’ loop
Example 5

Do final calculation if user inputs are correct (positive)

>> if breaker==1
    nothing=0;clear nothing;
>> else
    result=depth_num*width_num*height_num;
    result_str=num2str(result);
    set(handles.Sttxt_result,'string',result_str);
>> end

If violation occurs and user wants to exit. Then the final calculation will not be performed.
UI menus and Context menus

Additional menus that help enhance the GUI
UI menus

A UI menu creates a hierarchy of menus and submenus that are displayed in the figure window’s menu bar.
UI menus

To create a UI menu

Click on the ‘Menu Editor’
UI menus

The Menu Editor appears

Click on ‘Menu Bar’
UI menus

Then create your menu

Label: Seen by user

Tag: Name of Callback
UI menus

- Program the Callbacks as normal. The Callbacks will be executed when the user clicks on the option.
- Each ‘child’ menu item must have unique Tags, even though they have different ‘parents’.
UI Context menu

A UI context menu appears when the user right-clicks a graphics object
UI Context menus

To create a UI Context menu

Click on the ‘Menu Editor’
UI menus

The Menu Editor appears

Click on ‘Context menus’
UI Context menus

Create your context menu

Name of the Context menu

Label: Seen by user

Tag: Name of Callback
UI Context menus

- Give a name to the graphics object which the Context menu is attached to
- Link the graphics object to the Context menu

```matlab
>>hline=plot(u,y);
>>handles.hline=hline;
>>set(hline,'UIContextmenu',handles.LineCmenu);
```

- Plot the graph and give it a name
- Name of the Context menu
- Matlab name
- Line name