ESCI-7205 Data Analysis in Geophysics
Lab 17 Oct. 24, 2013

Bob Smalley

MATLAB GUI

Today we are going to see how to build a Graphical User Interface (GUI) in Matlab. We will be doing the "easy" part – coding up a tool to do some interactive task. The harder part is to define

A) what you want this tool to do, and
B) how the user controls/interacts with the tool through the GUI.

This is the hard part.

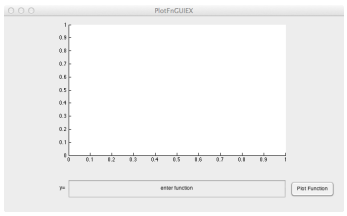
What do we want our tool to do?

Let's make a tool to display (plot) a function of a single variable. This tool should take any legal, single line, of Matlab code that can be evaluated for a real vector x and plot it.

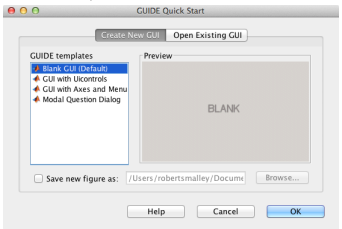
To start, we will fix the range and sampling of x. Later we can add features to the GUI to change these items. (We have to be careful with this "add later" idea, else our GUI will get confusing to use, write, and de-bug.) For now, we can plan on a "setup" button that would lead us to another GUI where we could set the parameters for defining the x vector.

What should the GUI "look" like? We need places to -
A) plot the function
B) enter the function
C) button to plot the function that was entered (so far will not do it on <CR>/Enter).
D) setup (later)

You might want to draw the GUI up on a paper, etc. Here is one possible design. (setup button missing).

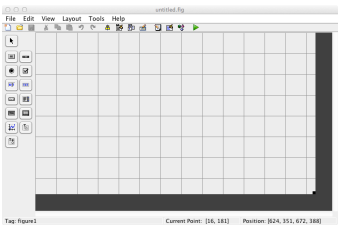


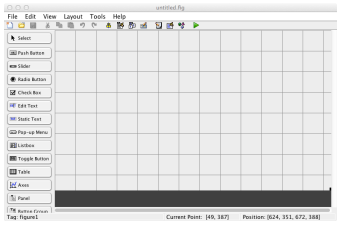
We will use a GUI tool in MATLAB to design our GUI (this cuts down on what you have to type, exactly correct, by about an order of magnitude). The MATLAB GUI tool is called the GUI Development Environment, GUIDE.



1) Start GUIDE. There are two ways to do this – type "guide" into the command window or start it from the icon in the tool bar (on the version in the Mac Lab, this changes with the changes-for-the-sake-of-changes between versions of MATLAB).

This will generate a pop-up window to initialize GUIDE. You can work on an existing GUI or start a new one by clicking on one of the two selections at the top of the window. We will be building a new GUI (the selection on the left is “highlighted”, and this selection has further selections of a number of pre-built templates on the left for the type of GUI – we will use the Blank GUI (Default) selection. When ready, click in the “OK” box.





This will bring up another pop-up-window, the GUIDE layout editor, in which we build the layout part of the GUI.

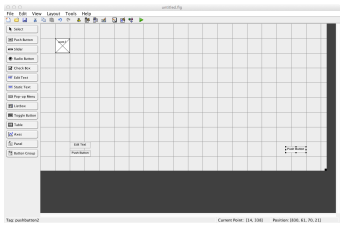
The palette on the left offers a number of components for building the GUI.

You can turn on a display of the component names using the preferences menu. (I am doing this from a different version of MATLAB, so things might not look exactly the same).

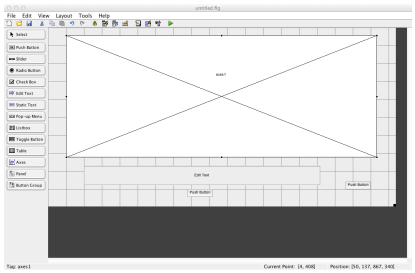
This helps a lot as it gives a descriptive name to each of the icons (here a word is worth a thousand pictures).

Now we have a design, so let’s implement it.

The default GUI template produces a fixed size GUI. It does not rescale with its window. We can fix this with additional programming (luckily - using the tool bar). For now we will leave it fixed size, but make it bigger. First make the GUIDE window bigger using the regular mac window sizer in the lower right corner. Then resize the GUI window using the black square in the lower right corner of the GUI template.



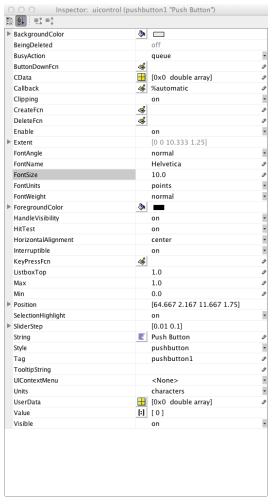
Next we will place the four items we identified above – an “axis” (a place to display MATLAB graphics), an “edit text” box (to enter the function), and two push buttons (one to plot the function and one to enter the setup GUI).



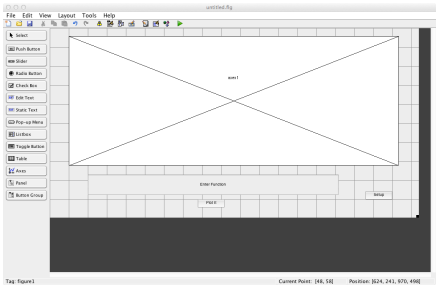
This is done by selecting items from the palette on the left and dragging them into out GUI template. You can resize them and move them around as you need/want. Can also align, distribute, group in “panels”, etc.

Put in an “axis”, “edit text” box, and one (no setup button) or two (plot and setup buttons) “push buttons”. They come up small so resize them.

Now we can put the prompts, etc. into the buttons. This is done using the "Property Inspector" (right click on an item or select and item and use the view drop down menu). Selecting the center bottom push button, the property inspector opens a pop-up window with the properties of the button. Some are fixed based on the type of item, and some are editable.



The field names here are supposed to be "obvious" (if you are the person who wrote it!). The text displayed on the button is stored in the "string" field, and the default is "Push Button" (which is what is seen in the GUI template). Change this to "plot function" (or something "trazarlo"). You can see the changes with a <CR>/Enter). Close the pop-up window to accept the changes (the red dot with the x at the top left). Do this for the other button (setup). Do the same for the Edit box (the string field again) ("Enter Function") and the template window (called the "figure"). The "tag" property identifies/names the control and helps you identify the various controls and is used to refer to that control in the program. We will need to remember the tag "pushbutton1" for "Plot Function" (and "pushbutton2 for "Setup").



Now we can "run" (actually dry-run, preview) the GUI to see "what it looks like". Use the "run" button on the tool bar. We will have to save it first. Saving will make two files – an m-file and a fig-file. They will both have the root filename from the dialog box. Two windows will open – an editor window with a lot of "gibberish" and one with the GUI. You can edit in the Enter Function window, the two push buttons react when you click on them (the flash inverse video).

The m-file window contains automatically generated code with spaces for you to enter your custom code to do whatever it is the GUI is supposed to do – here - read an edit text box, evaluate and plot the function, do the setup.

Look in the m-file. The first section sets up the GUI. Notice the comment at the end of the section to NOT EDIT that part of the file.

The two things we are interested in here are the "callback" routines (when you push a button, etc., the "callback" routine contains the code that gets executed). The second thing of interest is the "handle" – this is a unique identifier that is associated with A) a file, B) a graphics object (ranges from a plot to features of the plot such as which symbol), or C) a function. Handles are used to pass information about the object between objects, to callback routines, to matlab functions such a plot, etc. (See http://blogs.mathworks.com/loren/2006/08/16/handle-this/ for more info. Also ask other people who have written GUIs, look on the web, etc. Handles are not intuitively obvious and they have a lot of poorly defined components, it is hard to

find all their properties, and the values the properties can take. You will also get lots of typing exercise and most of the identifiers were designed to have a low probability of you making a variable by that name and causing a conflict. This means most of the automatically generated names are pretty long winded. When you set things like plot(…,LineColor = RBG::Red…) in a call to plot you are modifying fields in the handle of that plot.)

Find the function(s) at the bottom of the file with the routines "PlotIt_Callback" (I changed the Tag from pushbutton1) (and "pushbutton2_Callback) near the bottom of the m file. We will now have to put the code to read the function from the "edit text box", calculate the function, and finally plot it.

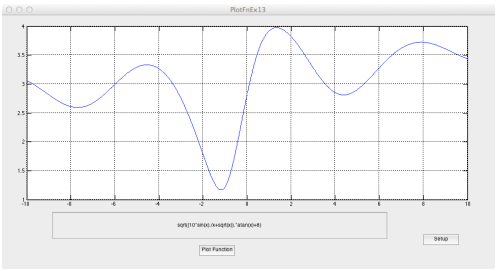
Once you have found the PlotIt_Callback, paste the following code after the end of the automatically generated code if this was the last function in the file, or before the beginning of the next function, identified by (or some other comment saying you are moving on).

% --- Executes on button press in …

Code to paste:

```
%set up x range and step size (future - allow user to change this using setup button
x=-10:.01:10;
%now get the string from the "edit text" box.
s = get(handles.EnterFunction, 'String');
y = eval(s); %eval just evaluates the given string
handles.axes1; %Subsequent commands draw on axes1.
plot(x, y); %now plot the function
grid;
```

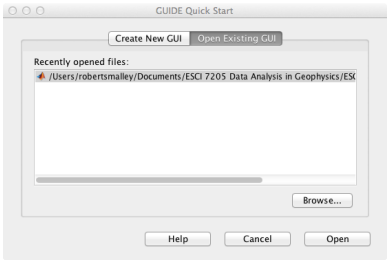
Save the code. Note that when you save changes you made in the graphical template, this will update what you see in the editor window, it fixes the automatically generated code (you have to save the graphics template to update the m-file). As you change your callback routines you also have to save the m-file (chages to the m-file brought about by saving the graphical template are not saved when they are moved to the m-file, you have to save the m-file. It will prompt you about saving the file if there were changes made since it was opened.) To run the code enter the m-file name into the command window.



To build this GUI we played around a bit with a graphical interface design tool, labeled and named a few things, and supplied all of 6 lines of code!

Writing GUIs was a horror show before the days of GUIs to write GUIs. The m-file is ~120 lines long, mostly comments explaining the rather opaque setup stuff. We don't have to deal with that anymore.

This code is just to demonstrate how to program the GUI. It does not do input error checking (you could enter something that is not an evaluable function and you would get a MATLAB error complaint in the command window), etc.



Returning to continue work on GUI project. When you start guide, select the "open existing gui" selection and then select the file from the list. This will open the GUI template fig file for modification, but will not open the associated m file in the MATLAB editor (until you save it, then it will open an editor window with the m file).

Some places to find out more about building GUI's

http://www.mathworks.com/discovery/matlab-gui.html

http://www.mathworks.com/help/matlab/gui-building-basics.html

(the video link on this page is for the newest version of MATLAB, which has a very different look than the version on the machines in the Mac Lab. The link below seems to be the earlier version of the same video

http://www.youtube.com/watch?v=D_hmws6dwgg

for the example from Mathworks (the callback code is missing, this is about how to lay out the GUI)

http://www.mathworks.com/help/matlab/matlab_prog/calling-a-function-using-its-handle.html

for handles and callbacks

http://blogs.mathworks.com/pick/2008/04/17/advanced-matlab-handles-and-other-inputs-to-guide-callbacks/

In a GUI, we oftentimes have to call a matlab function without knowing beforehand (i.e. when we write the code) what function will be called. This can be done using the handle of the function we want to execute as an input parameter to a function call that will execute other functions. This can get pretty complicated.

http://www.mathworks.com/help/matlab/matlab_prog/calling-a-function-using-its-handle.html