EAS105 Fall 2010 Notes on how to use GUIDE, for project7 and project9

First, watch the video which is on youtube:
http://www.youtube.com/watch?v=D_hmws6dwgg

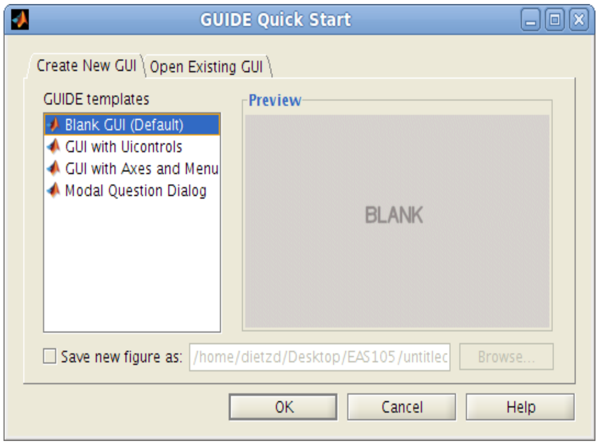
(You may also find other Matlab tutorials on Youtube by clicking on the username MATLAB found at the top left region of your screen when this is playing.)

In class, my plan is to basically follow these steps on my computer, while explaining various aspects of what I'm doing. I had originally followed the Hahn/Valentine book, but there are errors in that GUI discussion due to changes in versions of Matlab.

Example One: "Demo"

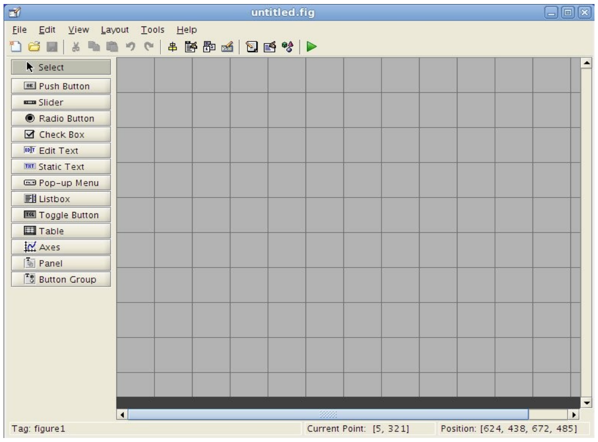
Open Matlab.

Now, in the Command Window, type "guide".



□ Save new figure as: /home/dietzd/Desktop/EAS105/untitlec

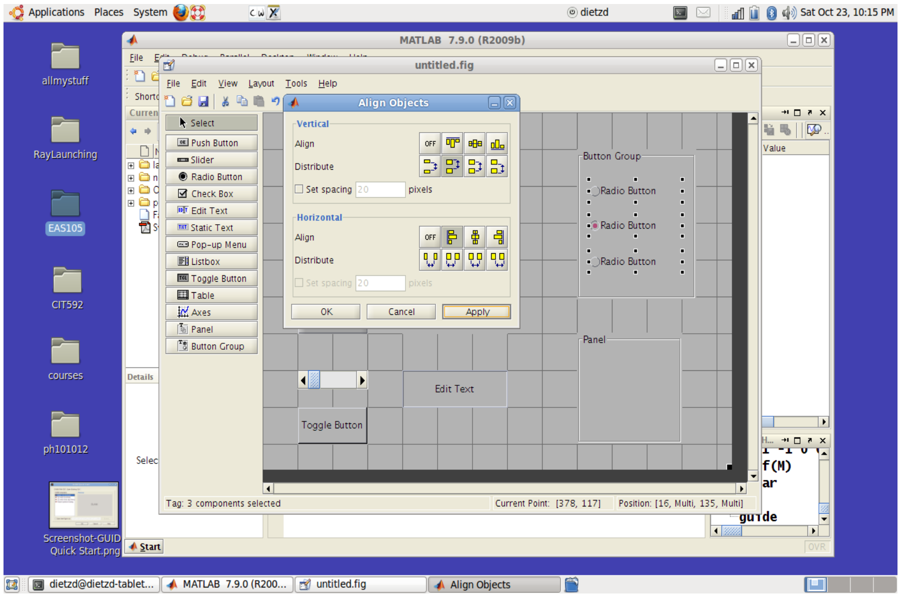
You should get this window. Hit "OK" if you want to create a new gui with the typical defaults.



Tag: figure1   Current Point: [5, 321]   Position: [624, 438, 672, 485]

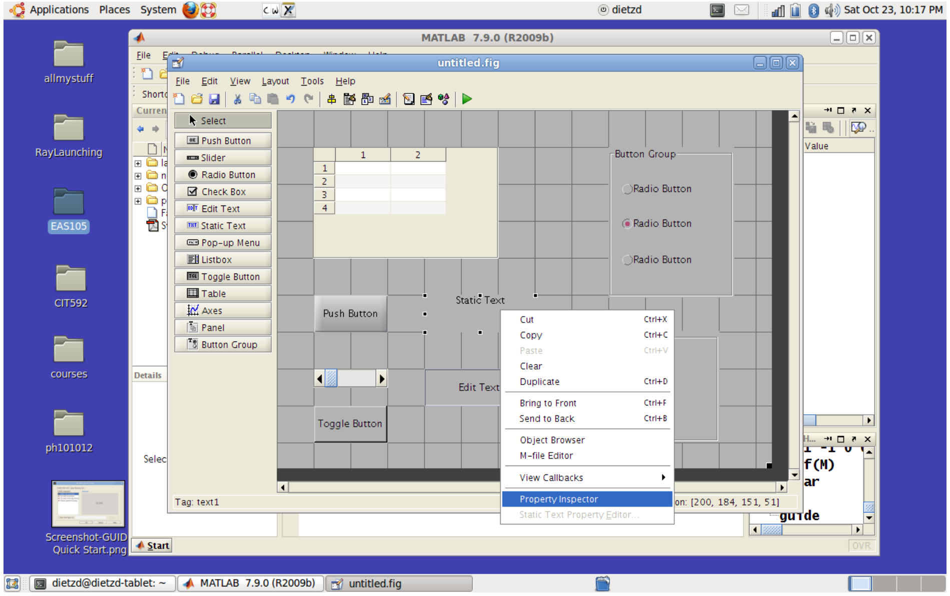
At this point, what you need to do is fairly obvious. You mess around clicking on the various options, on the left, then graphically designing your gui, in the gridded area.

If you want to create a set of "radio buttons", you place them inside a "button group". This will look messy if you don't align it. So, here's how you align things:



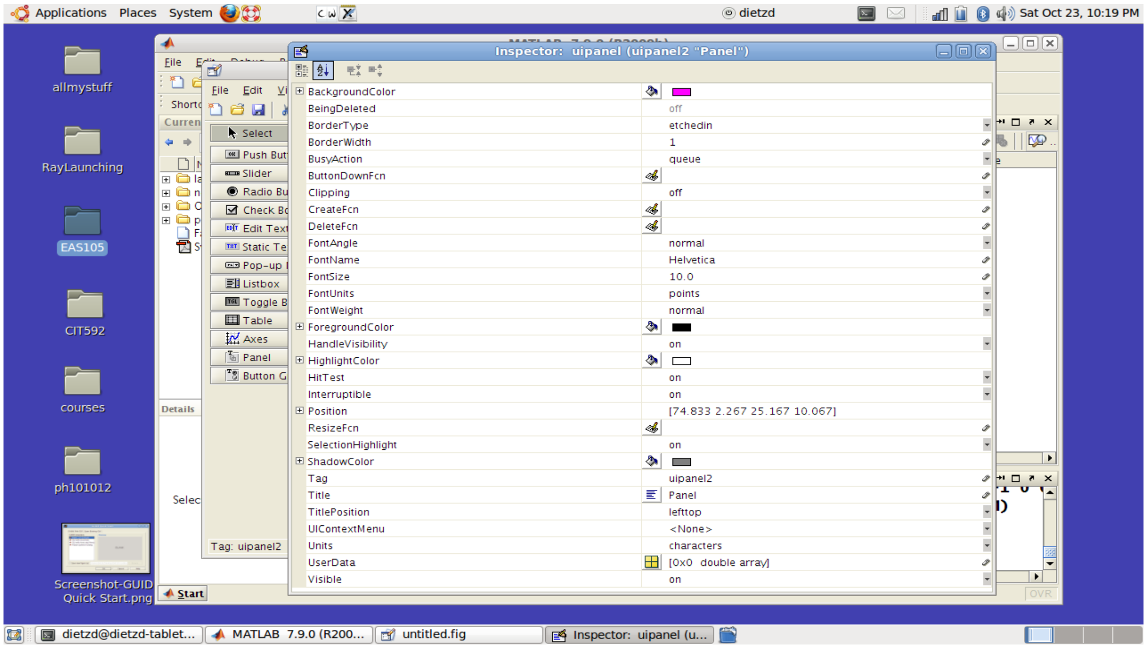
Tag: 3 components selected   Current Point: [378, 117]   Position: [16, Multi, 135, Multi]

Click on "Tools" and "align objects". You then click on the first item, then shift-click on the subsequent items you wish to align. In this example, I'm aligning three radio buttons which are together in a group. Note that the guiding dots are all visible for the three buttons. You should have both a Vertical and Horizontal option activated. When you are ready, hit "Apply" and everything should line up for you!



Tag: text1   on: [200, 184, 151, 51]

Next, I want you do become familiar with the Property Inspector. To get to this tool, right click on the gui component you wish to find information about. The values in here are the initialization values, and you are able to change them during runtime as you desire. Be aware of the names of these objects. If you don't like the default names, now is the time to adjust them. (Once you have launched the gui for the first time, it will become harder to adjust any names. Most of the names are inconsequential anyhow.)



Inspector: uipanel (uipanel2 "Panel")

BackgroundColor	🟪
BeingDeleted	off
BorderType	etchedin
BorderWidth	1
BusyAction	queue
ButtonDownFcn	
Clipping	off
CreateFcn	
DeleteFcn	
FontAngle	normal
FontName	Helvetica
FontSize	10.0
FontUnits	points
FontWeight	normal
ForegroundColor	■
HandleVisibility	on
HighlightColor	□
HitTest	on
Interruptible	on
Position	[74.833 2.267 25.167 10.067]
ResizeFcn	
SelectionHighlight	on
ShadowColor	▨
Tag	uipanel2
Title	Panel
TitlePosition	lefttop
UIContextMenu	<None>
Units	characters
UserData	[0x0 double array]
Visible	on

In this example, I'm turing one of the gui components hot-pink! Your choices are automatically saved, there is no "save" button.

To "launch" this gui, you can simply click on the little green "run" arrow at the top-middle of the guide screen. You should name your gui something other than "Untitled" or "gui", because that's pathetic.

Two files are produced, the fig file, and the m file. In this case, I named this "Demo", so I got "Demo.fig" and "Demo.m". Both files are important and must work together. The Demo.m is auto-generated code which runs your gui, but it doesn't do anything else. To see what this means, play with the Demo. The buttons click, the Toggle button toggles, the radio buttons are together in a group, which means only one may be clicked at a time, the slider slides, but nothing "happens". That's because you have to edit the Demo.m file if you want want anything to actually "happen".

Example Two: "SquareMe"

Follow the same steps as given above, and create a gui with a static text window, an editable text window, and a pushbutton. The editable text window will be where the user can type in a number, the static text window is where the square of the input number will show up, and the pushbutton will be what activates the calculation.

Launch this application by clicking the green arrow.

Next, we will edit the SquareMe.m file. Go to about line 80, and you will see the pushbutton1 Callback.

```
% --- 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)
d=str2double(get(handles.edit1, 'String'));
set(handles.text1,'String',num2str(d*d));
```

Only the last two lines of this function need to be written for the entire gui to do its job!

Each gui component is controlled by "getters" and "setters". All we have to do is use a getter to grab the user's input from the editable textbox, then square that value and place it in the static text box with a setter.

One more thing we have to understand first, though, is this handles structure. handles is a structure of whatever dimensions or types are needed, and you are welcome to add to those dimensions or types as you wish. They contain all the information you will be passing around between the various gui components.

This dot.m file is not exactly like the dot.m files you are used to, because code is not executed in the order you might expect. The actual control is held by a function called gui_mainfcn which you will undoubtedly encounter in your "travels". You may view but not edit this file, however, I don't see too much point in viewing it yet. The important thing to understand is that this is where the actual control is, and your code is just helping gui_mainfcn along.

For each action taken by your user on your gui, you have the option to tell gui_mainfcn what is to be done when that action is taken. All the data is stored in handles, which is essentially a global variable (which I realize we haven't discussed yet).

Right now, you need to know that all the Properties you saw in the Property Insepctor are controlled by get and set. It will take awhile to become comfortable with how to format these commands, but you have a few examples to follow (such as in above or in Newton.m).

Another thing we have not discussed is this handles.variable syntax. Matlab supports this "object oriented" style of naming things, so handles.edit1 is a different thing than handles.text1, but they are bundled together in the same handles structure (similar to a cellarray).

Place a red dot (stop point) within the pushbutton1 Callback function, so you can see what the handles structure looks like in the debugger. Type "SquareMe" to run it. Then, when you click the pushbutton1, you can go to the Command Window and type something like

```
K>> handles
```

You will see the various parts of this data structure.

```
K>> get(handles.text1)
```

Will show you all the properties you may "set" on handles.text1

We could be more specific, and just ask:

```
K>>get(handles.text1,'String')
```

This will return the actual string, which happens to be preset to 'Static Text'.

Example Three: "Newton"

The name of this example should be a clue as to what's going on here, but nevermind that....
This gui computes the square root of a given value, similar to the way the above example computed the square.

To write this function, I just followed the instructions in the suggested course textbook, Hahn/Valentine. I will explain the method to you, because you should be curious!

This is similar to the second example, except that a little more code is written. Lines 104 to 111 of Newton.m contain the non-autogenerated code, which does the work of calculating the square root.

Yes, it is "just" Newton's method, but this is my chance to get some more mathy engineering concepts into the course!

Many numerical methods make use of what's called a "contraction mapping" to locate a "fixed point". If you have a function which maps from a domain back into itself, but always a smaller portion of itself, it stands to reason that at least one point must be fixed. This is easy enough to see in the cartesian plane. If a function f(x) is monotonically strictly increasing (or monotonically strictly decreasing) and its slope has an absolute value always less than or equal to one (but never zero except at a finite number of points), and the domain of interest maps to a range which is a subset of of itself, you have a contraction mapping, and thus you have a fixed point.

Take a function you know, such as square-root. Consider the square root function from zero to 100. It maps to values from 0 to 10. Now, since f(0)=0, we have already found one fixed point, but f(1)=1 is another fixed point. It is often convenient to ensure that the fixed point is unique, also! (Note: A weakness in this example is that the square root function is not actually a contraction mapping from 0 to 1, but that will be forgiven, I hope!)

So, if you see loops in programs where the output value of one iteration of the loop becomes the input on the next iteration, this is often a contraction mapping under the hood.

However, without using Calculus to prove that the derivative of the function always has a magnitude less than one, we can use algebra to just show that the error goes down.

Concerning the method in Newton.m

```
        ← will change
105  x = a ← stays the same
```

Goal: $x \to \sqrt{a}$

```
106      for i=1:8
107          x = 1/2(x + a/x)
108          str{i} = sprint('%g', x);
109      end
```

$\underbrace{\text{str\{i\}}}_{\text{will be used to set text2}} = \underbrace{\text{sprint ('\%g', x);}}_{\text{similar to num2str}}$

Verify: if $x = \sqrt{a}$, this process is stable

$$x = \frac{1}{2}\left(x + \frac{a}{x}\right) = \frac{1}{2}\left(\sqrt{a} + \frac{a}{\sqrt{a}}\right) = \sqrt{a}$$

Next: Show this process attracts a solution:

if $x = \sqrt{a} + \varepsilon$, error decreases $\varepsilon_0 = a - \sqrt{a}$

$$x = \frac{1}{2}\left(\sqrt{a} + \varepsilon + \frac{a}{\sqrt{a}+\varepsilon}\right)$$

$$x = \frac{1}{2}\left(\frac{(\sqrt{a}+\varepsilon)^2 + a}{\sqrt{a}+\varepsilon}\right)$$

show:

$$\sqrt{a} \le \frac{1}{2}\left(\frac{(\sqrt{a}+\varepsilon)^2 + a}{\sqrt{a}+\varepsilon}\right) \le \sqrt{a} + \varepsilon$$

$$2(\sqrt{a}+\varepsilon)\sqrt{a} \le (\sqrt{a}+\varepsilon)^2 + a \le 2(\sqrt{a}+\varepsilon)^2$$

$$\cancel{2a} + 2\varepsilon\sqrt{a} \le \cancel{2a} + 2\varepsilon\sqrt{a} + \varepsilon^2 \le \cancel{2a} + 4\varepsilon\sqrt{a} + 2\varepsilon^2$$

$$0 \le \varepsilon^2 \le 2\varepsilon\sqrt{a} + 2\varepsilon^2$$

Note that as soon as the error (epsilon) is less than one, epsilon squared will be less than epsilon. That is often a typical assumption about epsilon. However, in this example, the original error can easily be well larger than 1, but as you can see, even then, errors must be progressively smaller.

One may also question whether this is enough to show convergence. It's not. Sometimes iterations can decrease to another limit which you don't care about! Can you see why the above iteration cannot approach another limit?

If it were to approach another limit, say with an error of epsilon, it would again decrease, causing a contradiction, because we just assumed we were converging to a different limit!