Creating a GUI in MATLAB

Table of Contents

Whenever you write a program, say for data analysis or process control, you (hopefully) know how to use it from the command line. However, as the functions get more complex, or you want other, less-Matlab-savvy people to use it, a graphical user interface (GUI) can make your program much easier to work with. Fortunately, Matlab is well-designed for this.

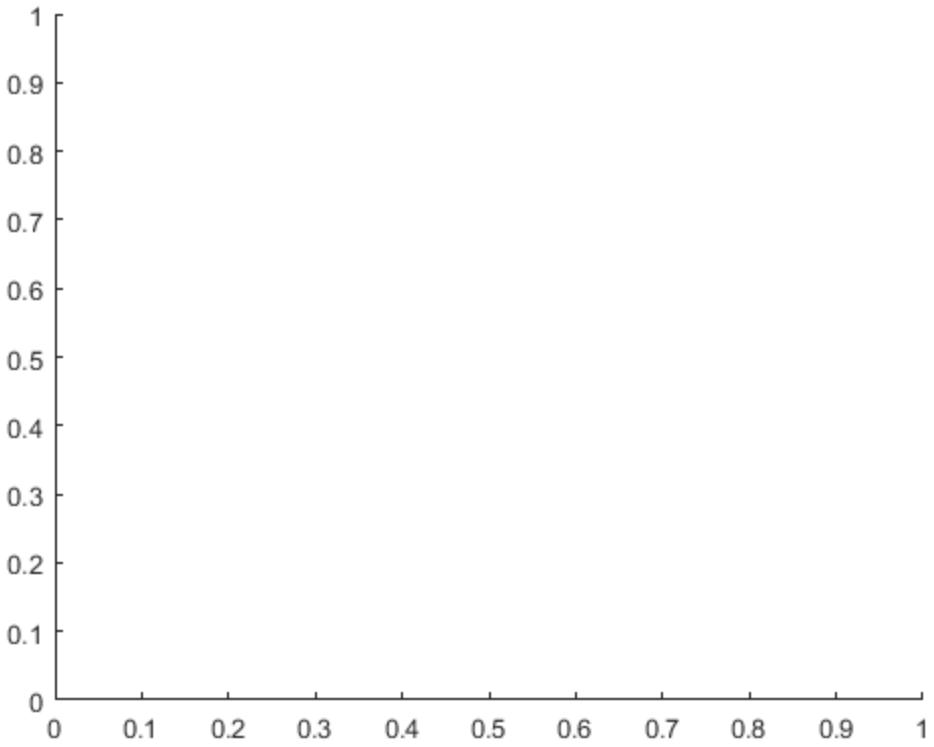
There are two main, somewhat interchangeable methods for building a GUI in Matlab. You can do it entirely from code, or you can use a graphical interface called GUIDE (GUI Development Environment) to build your graphical interface. You can manually edit the code that GUIDE produces, but it's kind of a mess. I think you can also write a bunch of code and then edit it further from GUIDE, but I'm not sure. I'm not very familiar with GUIDE, and programmatic construction is relatively straightforward, so that's what I'll be teaching you here. Sorry, no zombies in this one.

This will use structures extensively, so make sure you are familiar with those first.

Building the GUI

In the beginning, there was nothing. Then there was a window. Then there was an axis inside that window:

```
hwindow = figure;
h = struct('window',hwindow);
h.ax = axes('Parent',h.window);
```



It's very good practice when building a GUI to assign everything to a variable; hence `hwindow = figure` instead of simply `figure`. You'll see why as we move along. It'll simply your work to store everything in a structure, and if you have nothing better to call it it's common practice to call that structure "h", for "handle".

Note that I could have combined the first two lines into one step with `h = struct('window',figure)`.

Next we put an axis in this figure. We specified the parent as the figure we just created. Largely unnecessary at this point, since we only have one figure open, but good practice nonetheless.

Let's look more closely at that axis we just created. Since Matlab 2014b, getting and setting properties of plots and figures and the like has been greatly simplified. We can get a quick look at the most important parameters:

```
h.ax
```

```
ans =

  Axes with properties:

            XLim: [0 1]
            YLim: [0 1]
          XScale: 'linear'
          YScale: 'linear'
    GridLineStyle: '-'
```

```
          Position: [0.1300 0.1100 0.7750 0.8150]
             Units: 'normalized'

  Use GET to show all properties
```

We can get an exhaustive list of all the parameters:

```
get(h.ax)
```

```
                      ALim: [0 1]
                 ALimMode: 'auto'
    ActivePositionProperty: 'outerposition'
         AmbientLightColor: [1 1 1]
              BeingDeleted: 'off'
                       Box: 'off'
                   BoxStyle: 'back'
                BusyAction: 'queue'
             ButtonDownFcn: ''
                      CLim: [0 1]
                 CLimMode: 'auto'
            CameraPosition: [0.5000 0.5000 9.1603]
        CameraPositionMode: 'auto'
              CameraTarget: [0.5000 0.5000 0.5000]
          CameraTargetMode: 'auto'
            CameraUpVector: [0 1 0]
        CameraUpVectorMode: 'auto'
           CameraViewAngle: 6.6086
       CameraViewAngleMode: 'auto'
                  Children: []
                  Clipping: 'on'
             ClippingStyle: '3dbox'
                     Color: [1 1 1]
                ColorOrder: [7x3 double]
           ColorOrderIndex: 1
                 CreateFcn: ''
              CurrentPoint: [2x3 double]
           DataAspectRatio: [1 1 1]
       DataAspectRatioMode: 'auto'
                 DeleteFcn: ''
                 FontAngle: 'normal'
                  FontName: 'Helvetica'
                  FontSize: 10
              FontSmoothing: 'on'
                  FontUnits: 'points'
                 FontWeight: 'normal'
                  GridAlpha: 0.1500
              GridAlphaMode: 'auto'
                  GridColor: [0.1500 0.1500 0.1500]
              GridColorMode: 'auto'
              GridLineStyle: '-'
            HandleVisibility: 'on'
                    HitTest: 'on'
               Interruptible: 'on'
```

```
LabelFontSizeMultiplier: 1.1000
                  Layer: 'bottom'
         LineStyleOrder: '-'
    LineStyleOrderIndex: 1
              LineWidth: 0.5000
         MinorGridAlpha: 0.2500
     MinorGridAlphaMode: 'auto'
         MinorGridColor: [0.1000 0.1000 0.1000]
     MinorGridColorMode: 'auto'
     MinorGridLineStyle: ':'
               NextPlot: 'replace'
          OuterPosition: [0 0 1 1]
                 Parent: [1x1 Figure]
          PickableParts: 'visible'
     PlotBoxAspectRatio: [1 0.7903 0.7903]
 PlotBoxAspectRatioMode: 'auto'
               Position: [0.1300 0.1100 0.7750 0.8150]
             Projection: 'orthographic'
               Selected: 'off'
    SelectionHighlight: 'on'
             SortMethod: 'childorder'
                    Tag: ''
                TickDir: 'in'
            TickDirMode: 'auto'
   TickLabelInterpreter: 'tex'
             TickLength: [0.0100 0.0250]
             TightInset: [0.0435 0.0532 0.0071 0.0202]
                  Title: [1x1 Text]
TitleFontSizeMultiplier: 1.1000
        TitleFontWeight: 'bold'
                   Type: 'axes'
          UIContextMenu: []
                  Units: 'normalized'
               UserData: []
                   View: [0 90]
                Visible: 'on'
          XAxisLocation: 'bottom'
                 XColor: [0.1500 0.1500 0.1500]
             XColorMode: 'auto'
                  XDir: 'normal'
                 XGrid: 'off'
                XLabel: [1x1 Text]
                  XLim: [0 1]
              XLimMode: 'auto'
             XMinorGrid: 'off'
             XMinorTick: 'off'
                 XScale: 'linear'
                  XTick: [1x11 double]
             XTickLabel: {11x1 cell}
         XTickLabelMode: 'auto'
     XTickLabelRotation: 0
              XTickMode: 'auto'
          YAxisLocation: 'left'
                 YColor: [0.1500 0.1500 0.1500]
```

```
                YColorMode: 'auto'
                      YDir: 'normal'
                     YGrid: 'off'
                    YLabel: [1x1 Text]
                      YLim: [0 1]
                  YLimMode: 'auto'
                YMinorGrid: 'off'
                YMinorTick: 'off'
                    YScale: 'linear'
                     YTick: [1x11 double]
                YTickLabel: {11x1 cell}
            YTickLabelMode: 'auto'
        YTickLabelRotation: 0
                 YTickMode: 'auto'
                    ZColor: [0.1500 0.1500 0.1500]
                ZColorMode: 'auto'
                      ZDir: 'normal'
                     ZGrid: 'off'
                    ZLabel: [1x1 Text]
                      ZLim: [0 1]
                  ZLimMode: 'auto'
                ZMinorGrid: 'off'
                ZMinorTick: 'off'
                    ZScale: 'linear'
                     ZTick: [0 0.5000 1]
                ZTickLabel: ''
            ZTickLabelMode: 'auto'
        ZTickLabelRotation: 0
                 ZTickMode: 'auto'
```

We can even get a list of the possible values for those parameters:

```
set(h.ax)
```

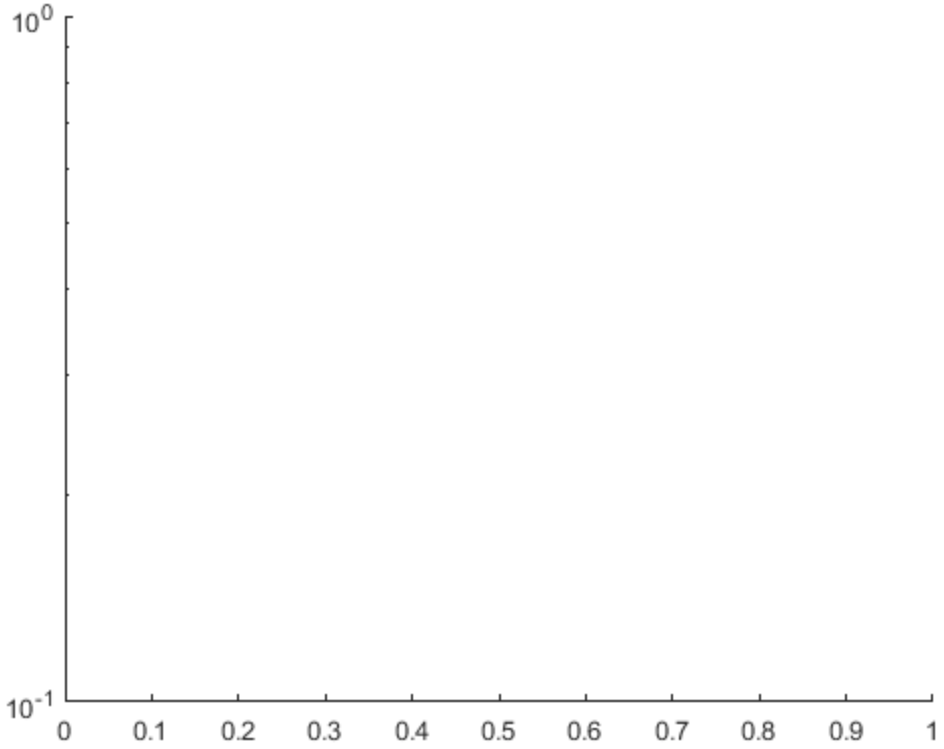
```
                      ALim: {}
                 ALimMode: {'auto'  'manual'}
     ActivePositionProperty: {'position'  'outerposition'}
          AmbientLightColor: {1x0 cell}
                        Box: {'on'  'off'}
                  BoxStyle: {'full'  'back'}
                 BusyAction: {'queue'  'cancel'}
              ButtonDownFcn: {}
                       CLim: {}
                  CLimMode: {'auto'  'manual'}
             CameraPosition: {}
         CameraPositionMode: {'auto'  'manual'}
               CameraTarget: {}
           CameraTargetMode: {'auto'  'manual'}
             CameraUpVector: {}
         CameraUpVectorMode: {'auto'  'manual'}
            CameraViewAngle: {}
        CameraViewAngleMode: {'auto'  'manual'}
                  Children: {}
                  Clipping: {'on'  'off'}
```

```
            ClippingStyle: {'rectangle'  '3dbox'}
                    Color: {1x0 cell}
               ColorOrder: {}
          ColorOrderIndex: {}
                CreateFcn: {}
          DataAspectRatio: {}
      DataAspectRatioMode: {'auto'  'manual'}
                DeleteFcn: {}
                FontAngle: {'normal'  'italic'}
                 FontName: {}
                 FontSize: {}
            FontSmoothing: {'on'  'off'}
                FontUnits: {1x5 cell}
               FontWeight: {'normal'  'bold'}
                GridAlpha: {}
            GridAlphaMode: {'auto'  'manual'}
                GridColor: {1x0 cell}
            GridColorMode: {'auto'  'manual'}
            GridLineStyle: {'-'  '--'  ':'  '-.'  'none'}
         HandleVisibility: {'on'  'callback'  'off'}
                  HitTest: {'on'  'off'}
             Interruptible: {'on'  'off'}
  LabelFontSizeMultiplier: {}
                    Layer: {'bottom'  'top'}
           LineStyleOrder: {}
      LineStyleOrderIndex: {}
                LineWidth: {}
           MinorGridAlpha: {}
       MinorGridAlphaMode: {'auto'  'manual'}
           MinorGridColor: {1x0 cell}
       MinorGridColorMode: {'auto'  'manual'}
       MinorGridLineStyle: {'-'  '--'  ':'  '-.'  'none'}
                 NextPlot: {'add'  'replace'  'replacechildren'}
            OuterPosition: {}
                   Parent: {}
            PickableParts: {'visible'  'none'  'all'}
       PlotBoxAspectRatio: {}
   PlotBoxAspectRatioMode: {'auto'  'manual'}
                 Position: {}
               Projection: {'orthographic'  'perspective'}
                 Selected: {'on'  'off'}
      SelectionHighlight: {'on'  'off'}
                 SortMethod: {'depth'  'childorder'}
                        Tag: {}
                    TickDir: {'in'  'out'  'both'}
                TickDirMode: {'auto'  'manual'}
     TickLabelInterpreter: {'none'  'tex'  'latex'}
                 TickLength: {}
                      Title: {}
  TitleFontSizeMultiplier: {}
          TitleFontWeight: {'normal'  'bold'}
            UIContextMenu: {}
                    Units: {1x6 cell}
                 UserData: {}
```

```
                View: {}
             Visible: {'on'  'off'}
       XAxisLocation: {'bottom'  'top'}
              XColor: {1x0 cell}
          XColorMode: {'auto'  'manual'}
                XDir: {'normal'  'reverse'}
               XGrid: {'on'  'off'}
              XLabel: {}
                XLim: {}
            XLimMode: {'auto'  'manual'}
          XMinorGrid: {'on'  'off'}
          XMinorTick: {'on'  'off'}
              XScale: {'linear'  'log'}
               XTick: {}
          XTickLabel: {}
      XTickLabelMode: {'auto'  'manual'}
  XTickLabelRotation: {}
           XTickMode: {'auto'  'manual'}
       YAxisLocation: {'left'  'right'}
              YColor: {1x0 cell}
          YColorMode: {'auto'  'manual'}
                YDir: {'normal'  'reverse'}
               YGrid: {'on'  'off'}
              YLabel: {}
                YLim: {}
            YLimMode: {'auto'  'manual'}
          YMinorGrid: {'on'  'off'}
          YMinorTick: {'on'  'off'}
              YScale: {'linear'  'log'}
               YTick: {}
          YTickLabel: {}
      YTickLabelMode: {'auto'  'manual'}
  YTickLabelRotation: {}
           YTickMode: {'auto'  'manual'}
              ZColor: {1x0 cell}
          ZColorMode: {'auto'  'manual'}
                ZDir: {'normal'  'reverse'}
               ZGrid: {'on'  'off'}
              ZLabel: {}
                ZLim: {}
            ZLimMode: {'auto'  'manual'}
          ZMinorGrid: {'on'  'off'}
          ZMinorTick: {'on'  'off'}
              ZScale: {'linear'  'log'}
               ZTick: {}
          ZTickLabel: {}
      ZTickLabelMode: {'auto'  'manual'}
  ZTickLabelRotation: {}
           ZTickMode: {'auto'  'manual'}
```

And, finally, we can very easily set individual parameters.

```
h.ax.YScale = 'log';
```



Now let's change it so the axis only takes up half of the figure. Two things to pay attention to here. First is the Units parameter, which is currently set to 'normalized'. This means that the positioning data will be relative to the dimensions of the parent object, in this case `h.window` . There are several other options available, but normalized in my mind is the easiest.

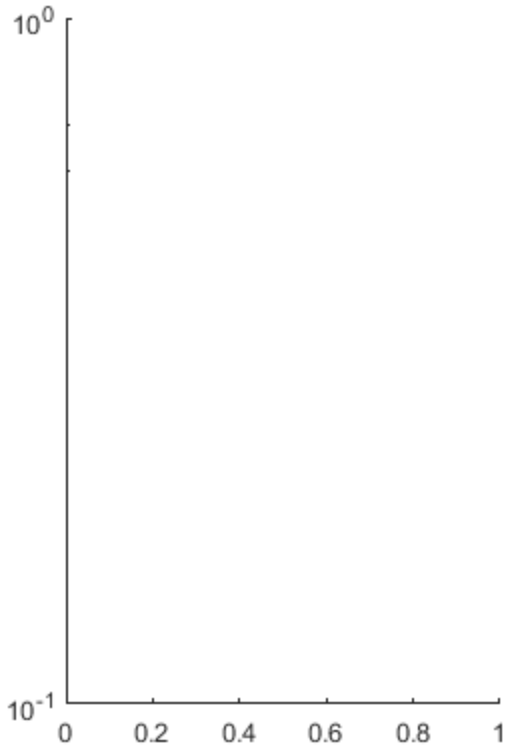
```
s = set(h.ax);
s.Units
```

```
ans =

    'inches'
    'centimeters'
    'characters'
    'normalized'
    'points'
    'pixels'
```

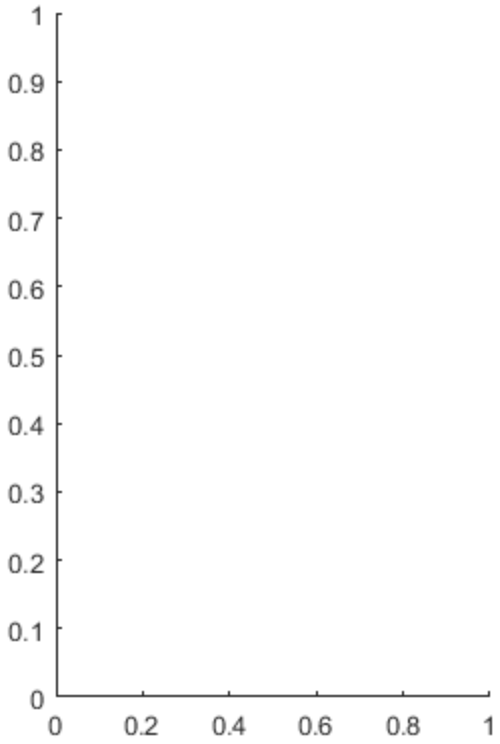
Second, the position data can be somewhat confusing. It's stored as `[Xorigin Yorigin Xwidth Yheight]` , measured from the bottom-left corner, so here's how we half the width and move it to the right:

```
old = h.ax.Position;
h.ax.Position = [old(1)+0.4 old(2) old(3)/2 old(4)];
```



Of course, we could have set that when we first created the axis. Let's delete that axis and re-build it.

```
delete(h.ax)
h.ax = axes('Parent',h.window,'Position',[0.53 0.11 0.3875 0.8150]);
```

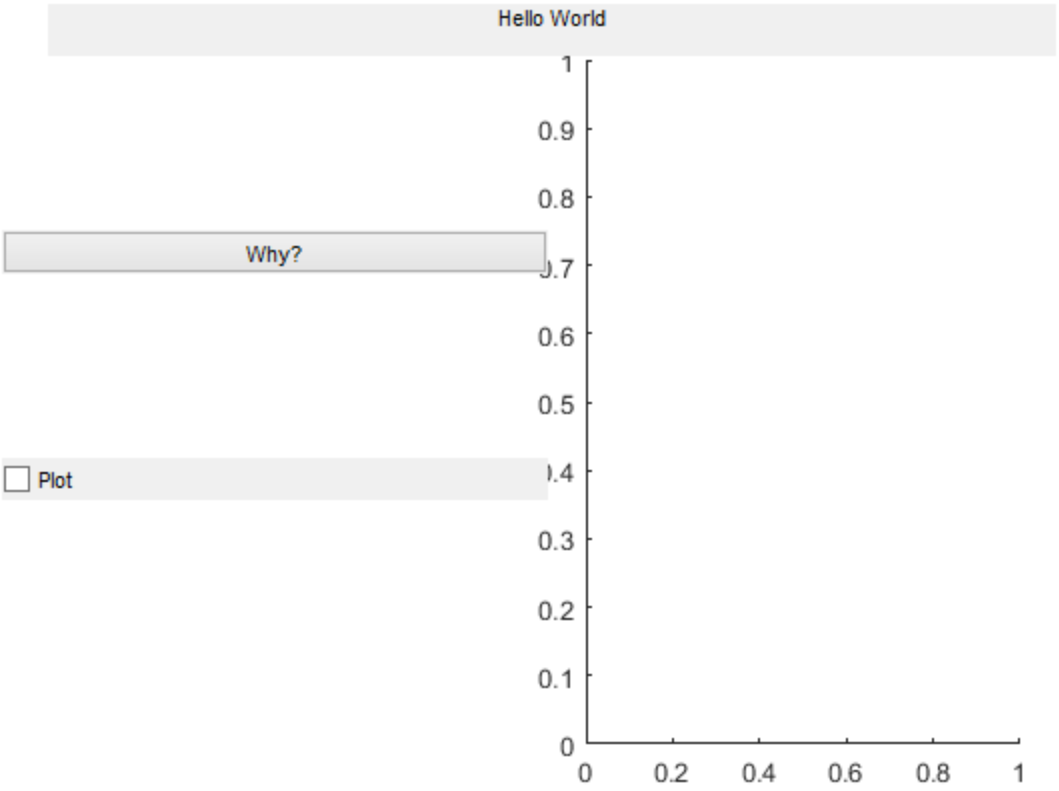


You can define the size of your window this way, too. Use `scrsz = get(0,'ScreenSize')` to, well, get your screen size.

Interacting with the GUI: uicontrol

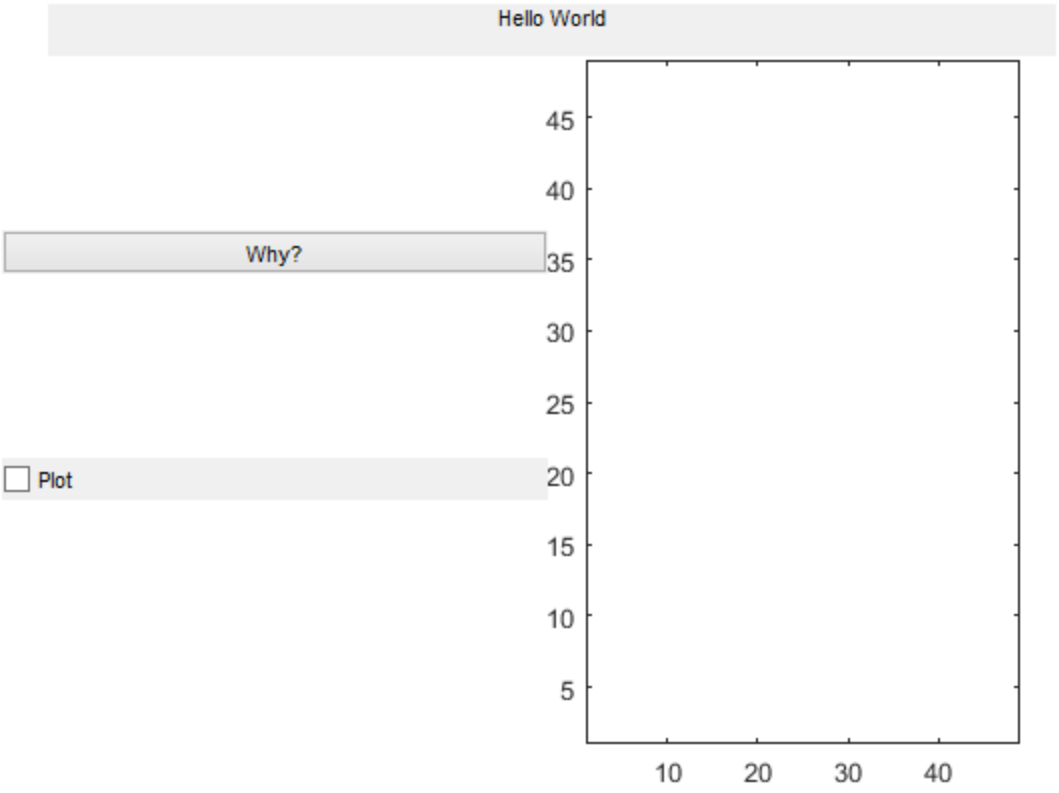
Now let's add some elements that let us interact with the GUI. This is done with the `uicontrol` function.

```
h.button1 = uicontrol('Parent',h.window,'Units','Normalized',...
    'Style','PushButton','Position',[0.01 0.67 0.485 0.05],...
    'String','Why?'); % A button
h.button2 = uicontrol('Parent',h.window,'Units','Normalized',...
    'Style','Checkbox','Position',[0.01 0.4 0.485 0.05],...
    'String','Plot'); % A checkbox
h.talk = uicontrol('Parent',h.window,'Units','Normalized',...
    'Style','Text','Position',[0.05 0.93 0.9 0.06],...
    'String','Hello World'); % A text box
```



Awesome. Now we have fun things to play with. Problem is, they don't do anything. But we'll get to that in a bit. Let's look at those elements more closely. Each `uicontrol` element had a "style" defining how it can be interacted with. There are many styles for uicontrol (drop-down menus, radiobuttons, text-entry boxes), just see the documentation. Each element also has a parent. It may seem unnecessary in this case but it becomes useful as your UI grows more complex, particularly when you want to group elements such as radiobuttons.

```
% Let's put something on the axis, but keep it from being visible for
 now.
h.p = pcolor(h.ax,peaks);
h.p.Visible = 'off';
```



Note that the first argument to our plot command is actually it's Parent, the axis we created earlier, rather than data as you may be used to. If we wanted to, we can directly manipulate the X, Y, or Z data by setting, for example, `h.p.XData` equal to some set of values. Let me say that again. If you have a plot on an axis, you can directly change the X and the Y data independently with `h.p.XData = values` . I wonder if that will be useful? I also wonder why I re-typed that when you could have just read it again? I also wonder why I'm putting question marks at the ends of statements? In any event, notice the `...` syntax. It will really help clean up your code and make it easier to look at if you have no more than two parameter-value pairs per line.

Interacting with the GUI: Callbacks

Okay, now let's get really crazy and make things happen when we press fun buttons. We're going to do this with *Callbacks* . A callback is a function that gets executed when you interact with the `uicontrol` element. Here we'll set the Callbacks for our buttons to the same function called GUI_Callbacks, which we'll define in a minute, and pass our structure containing all the UI components, h

```
h.button1.Callback = {@GUI_Callbacks,h};
h.button2.Callback = {@GUI_Callbacks,h};
```

The @ symbol indicates that the function we're passing is in the same file as this one (we'll put it at the bottom of the file). We can pass as many variables as we want, separated by commas after the h. By the way, there's also a cool function called `deal` that simplifies this a bit:

```
[h.button2.Callback,h.button1.Callback] = deal({@GUI_Callbacks,h});
```

Now, after all that is typed in our function, we're going to define our GUI_Callbacks function at the bottom of the same file (note the lack of @ in our function name)

```
function GUI_Callbacks(hObject,eventdata,h)

switch hObject
    case h.button1
        why
        h.talk.String = 'Check the console';
    case h.button2
        switch h.button2.Value
            case 1
                h.p.Visible = 'on';
            case 0
                h.p.Visible = 'off';
        end
end
```

Play around with this and see what it does. But wait, why does GUI_Callbacks have three inputs, when we only passed it one? `hObject` and `eventdata` are parameters which are always passed to the callback, in that order, so we always have to have them there, in that order. `hObject` tells the function which `uicontrol` object was interacted with, and `eventdata` contains what happened. Here, we compare the `hObject` value to our list of `uicontrol` objects and manipulate the UI accordingly.

See how convenient structures made this? otherwise we would have had to pass every element we wanted to interact with or control individually, which would get messy quickly. Note that the *structure* of this file is important. We have a main function, which we call from the command line to build our UI. Within that function, we *first* define all of our UI elements, *then* set their callbacks to a subfunction, which we define after the end of the main function. Why didn't we set the callbacks when we built the `uicontrol` elements in the first place? We could have, but at that point we hadn't finished populating our h structure with our display objects. If we defined the callback at that point, we would be passing `h` as it existed *at that point in the function* . For example, if we set the button callbacks at creation, they wouldn't be able to interact with our textbox or our plot because they didn't exist yet!

How do we actually run the function? Just like any other function, we can type its name into the command line, or press "run" in the editor since it has no inputs. If you're curious how long it takes your computer to build this interface, try typing `tic;GUI_builder;t = toc` and see what happens.

Conclusion

Well, hope you now have a good idea on how to build a GUI in Matlab from scratch. I've provided the skeleton GUI we just built as a reference. Feel free to take that and make it your own.

Now go. Go build GUIs to visualize the zombie apocolypse data you've stored in your magnificent structures. And save the world. Either order.

Published with MATLAB® R2015a