VISUALZING MATHEMATICS CONCEPTS WITH USER INTERFACES IN MATLAB

David Szurley

Francis Marion University

Department of Mathematics

Florence, SC 29501

I. INTRODUCTION

Computers play a major role in today's business and academic settings. Exposure to commercial packages is a major advantage for graduates seeking employment or continuing their studies at graduate school. Matlab is a high-level language that provides an interactive graphical user interface. It is featured in hundreds of textbooks in engineering and the sciences and is integrated in the curriculum at many universities. The author has attempted to incorporate Matlab within courses covering calculus, linear algebra, and linear programming, among others. Assignments were created to graphically illustrate concepts taught within these courses. Code would be supplied to the students to solve an example problem. The students were expected to change the code to solve similar problems. The advantages were exposure to Matlab and a graphical understanding of the concept. Unfortunately, these assignments required students to modify or write code in Matlab. In order to accomplish this, time had to be spent in class explaining the code given to the students. This was very time-consuming or the author and frustrating to the students. Many students did not appreciate the intended effect of the assignments due to the time spent on modifying the code.

Graphical user interfaces (GUIs) provide a means of introducing students to scientific packages without entirely involving the students within the code. Instead of expecting students to modify code, they may simply edit text and click a button. The required calculations are then accomplished for the students without being visible. By creating GUIs, the author hopes to create assignments that require the students to change only a minimal amount of quantities. Moreover, the underlying calculations are hidden from the students. These GUIs will still graphically illustrate the concepts and expose students to Matlab, but also save class time owing to the simplified use.

In this paper, we will explain how to create simple GUIs with Matlab. The portions of a GUI will be explained and how information supplied within the GUI may be accessed. Finally, two examples will be discussed and advantages and disadvantages of GUIs with Matlab will be discussed.

II. HOW TO CREATE GUIS IN MATLAB – VISUAL ASPECT

One may view GUIs as consisting of two elements: the visual and actions aspects. The visual aspect is what the user sees while the action aspect is the calculations underneath. In Matlab, we may create the visual aspect in two ways, either by writing code or using GUIDE. author has attempted to write code to develop the visual aspect, but has encountered several difficulties. Sizing and placement of GUI components have been the most problematic for the author. Thus, we will concentrate our efforts on creating GUIs using guide.

GUIDE is the internal Matlab GUI development interface. It enables simplified arrangement and sizing of GUI components as well as auto-generation of code for these components. Examples of GUI components would be push buttons, axes, sliders, pop-up menus, among others. GUIDE may be started by typing *guide* at the Matlab command prompt. Figure 1 displays what the user will see when beginning GUIDE.

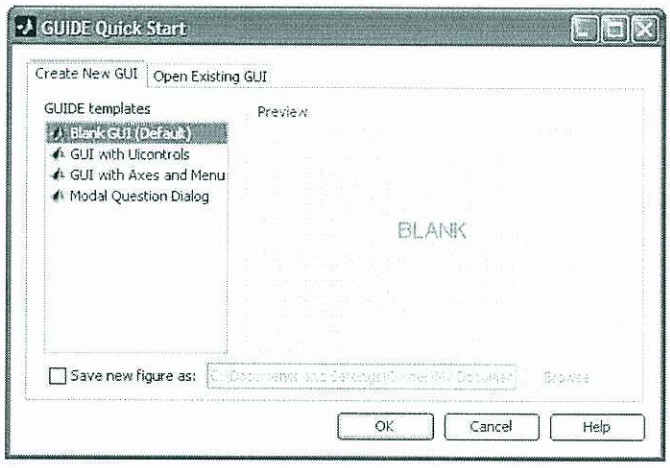

Figure 1: Beginning GUIDE.

In the quick start menu, one may load an existing GUI or start a new GUI. Although we will begin with a blank GUI, Matlab provides some basic layouts with specific components included. Figure 2 presents the blank GUI layout.

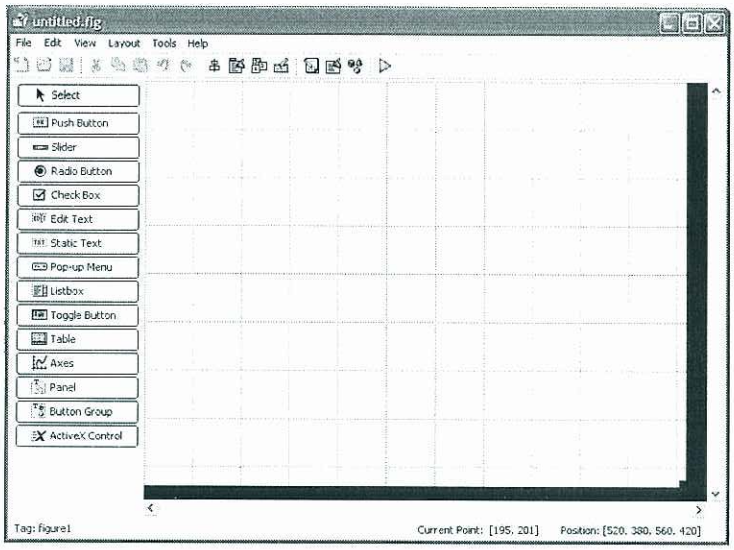

Figure 2: Blank GUI.

The GUI components may be viewed on the left in Figure 2. Components may be placed on the workspace by clicking on the desired, moving the cursor to the workspace, and clicking. The position of the cursor when clicked determines the upper-left corner of the component. Once the component is placed in the workspace, attributes such as size and position may be adjusted using the cursor. Other attributes may be adjusted a the Property Inspector, which may be viewed by doubling-clicking on the component or chosen from a drop-down menu when right-clicking on the component. Figure 3 shows an example of a push button with its property inspector open.

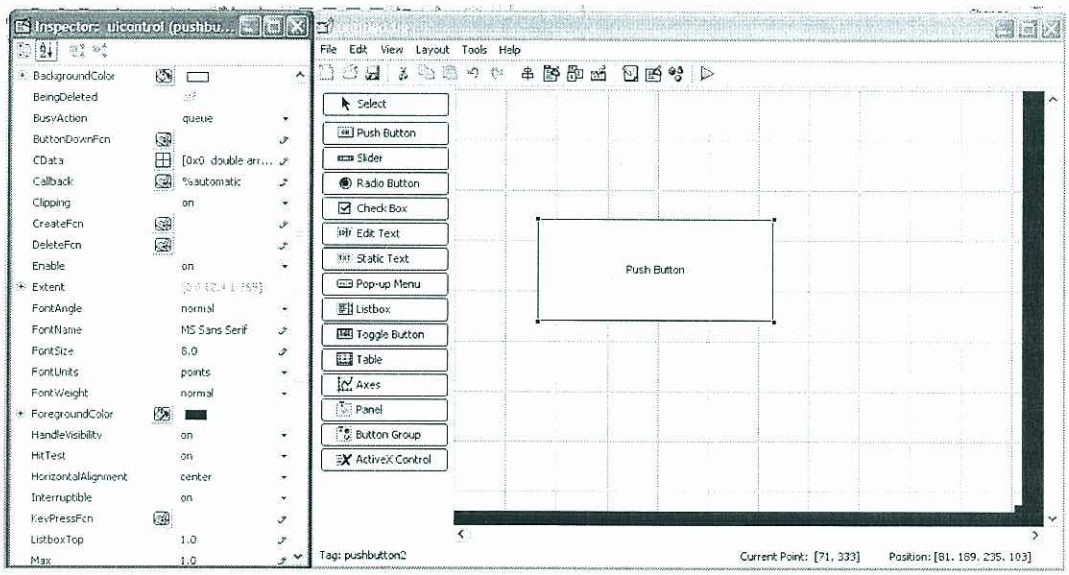

Figure 3: Push button with its property inspector.

Properties such as font, size, style, and the text on the push button may be edited via the property inspector. Note that *help <command>* or *doc <command>* will invoke the Matlab help menu or documentation for *<command>*. Generally, the author chooses to change only "String" (what rhe user initially sees) and "Tag" (reference title for the component).

One may choose "Save As..." at any time to save the GUI. However, the specified filename will be the title of the GUI. Also, the first time the GUI is saved, Matlab will auto-generate the code associated with the components of the GUI. This code will appear in the standaed Matlab m-file editor. An example is shown in Figure 4.

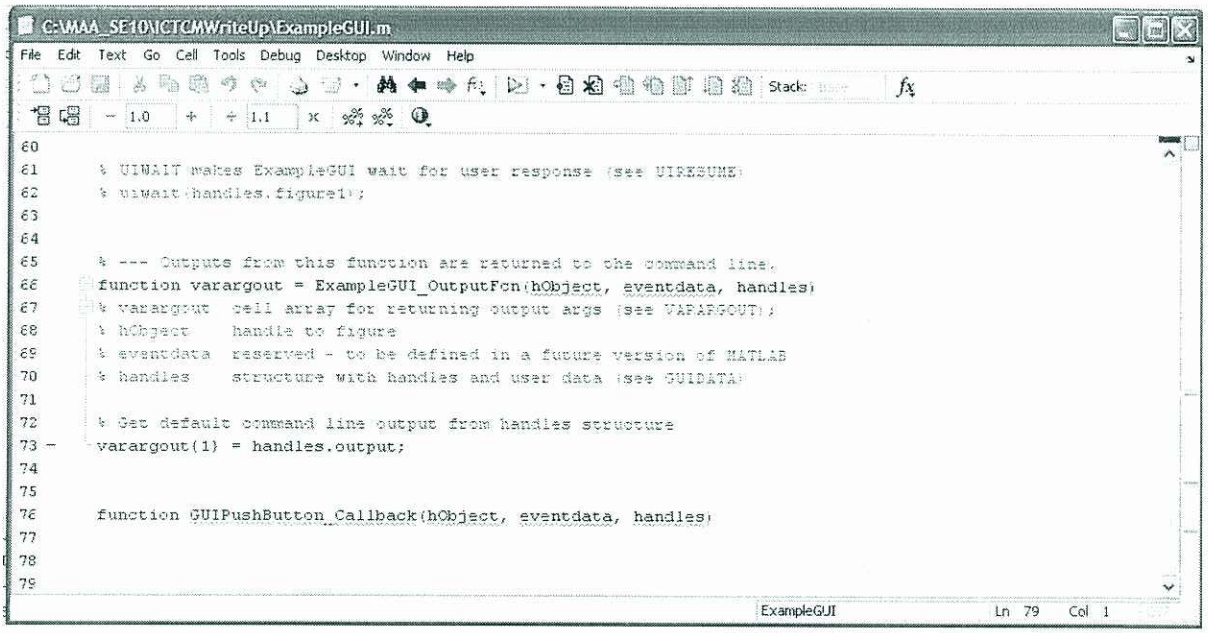

Figure 4: Auto-generated Matlab code for a GUI.

This code contains the code for the definition of each component in the GUI. Code for the callbacks (action upon execution of a component) for each component should be written within the appropriate function in this m-file. Two comments are in order here. First, the button that resembles $f_{()}$ lists the functions associated with the components of the GUI. Clicking on a name within this list will locate the cursor at the beginning of that function. Since the default Matlab names for GUI components tend to be generic (e.g., *pushbutton1*, etc.), the author chooses to modify the tag attribute as described previously. Secondly, note the function for the example single push button GUI is *GUIPushButton_Callback*. Here Matlab appended the user tag of *GUIPushButton* with *_Callback*. Code for the set of actions to be performed when the push button is pressed should be placed within this function.

III. HOW TO CREATE GUIS IN MATLAB – ACTION ASPECT

Regardless of how the visual aspect of the GUI is created, code must still be written for the callbacks. Since we have not been writing a GUI to solve a particular problem, it is not feasible to describe how to write these callbacks. However, we will concentrate on the communication between the user and Matlab. The commands *get* and *set* may be used to get or set component properties. Components within the GUI may be referenced via *handles.<Tag>*, where *<Tag>* is the tag of the component to be referenced. For example, *get(handles.EditText, 'String')* would obtain the string entered in the component with the tag *EditText*. The symbols that the user enters will be stored as characters and any quantity to be displayed to the GUI is required to be a character. We may use the commands *str2num* and *num2str* to convert from a string to a number and vice versa. One final comment is in order here. The command *guidata(hObject, handles)* may be used to update component data.

IV. HOW TO EXECUTE A GUI

Once a GUI is created, there are two ways to execute it: either via the command line or GUIDE. We may execute a GUI by typing the title of the GUI in the command line in Matlab or we may press the green play button on the toolbar when using GUIDE. In either case, errors are still appear in the Maltab command prompt.

V. EXAMPLE #1

Suppose we would like to create a GUI that will accept a user-determined number and evaluate a function as determined in a pop-up menu. The finished GUI is shown in Figure 5.

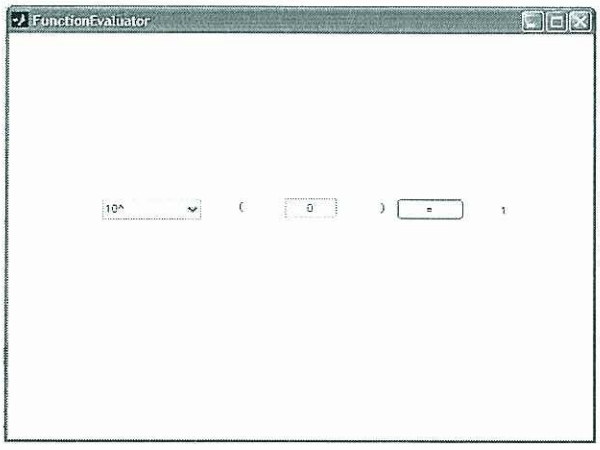

Figure 5: Format of the GUI for example #1.

This GUI will consist of six components:

- Pop-up menu to determine the function to be evaluated
- Two static text boxes for parentheses (for looks)
- Edit text for the user-specified value
- Push button for the evaluation
- Static text to contain the answer

Using the procedure described previously, we may add each component and modify their string and tag properties. However, we will look at the pop-up menu to determine the function in more detail. The pop-up menu will contain seven strings: Select, cos, sin, ln, log, e^, and 10^. The first is for looks; the others will represent the functions sine, cosine, natural logarithm, common logarithm, natural exponential, and exponential function with base 10. Since there is more than one string to enter, the "String" attribute has to be modified in a different manner than previously. To the left of the default string is a button that may be pushed to obtain what resembles a text editor. Here we may enter the strings to be displayed by the pop-up menu on separate lines. Figure 6 contains an image of the property editor along with the editor containing the strings.

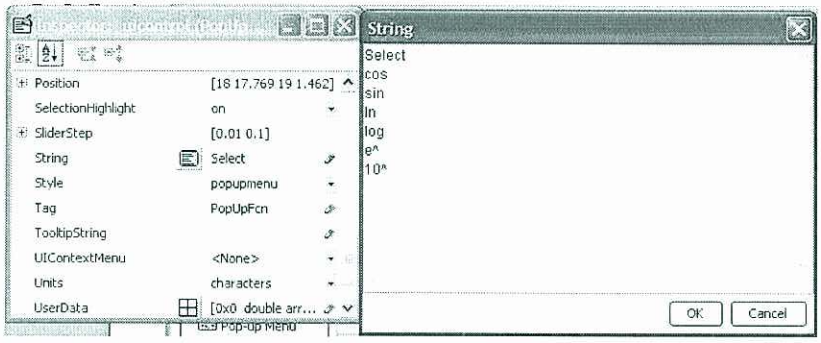

Figure 6: Pop-up menu strings.

One way Matlab refers to the strings in the pop-up menu is by assigning each string a number. Matlab will assign the first string the number one, second string the number two, and so on. Hence, *get(handles.PopUpMenu, 'Value')* will obtain the number associated with the string selected by the user. A switch construct (or equivalent) may then be used to evaluate the correct function.

Once the visual aspect of the GUI is completed, code must be written to obtain the user-defined information from the GUI and display the result. There are four steps that must be accomplished.

- Obtain the function

- Obtain the value
- Evaluate the function
- Display the result

Since no actions are required to be accomplished by either the pop-up menu or the edit text box (the simply obtain information from the user), all of these may be accomplished within the function for the push button. Figure 7 contains the code that will accomplish these four tasks.

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

%Obtain value
Value = str2num(get(handles.EditNum, 'String')) ;

%Operation switch
switch ( get(handles.PopUpFcn, 'Value') )
    case 1
        %Word 'Select' - of no use
    case 2
        Output = cos(Value) ;
    case 3
        Output = sin(Value) ;
    case 4
        Output = log(Value) ;
    case 5
        Output = log10(Value) ;
    case 6
        Output = exp(Value) ;
    case 7
        Output = 10^(Value) ;
    otherwise
        %Default
end

set(handles.StaticEquals, 'String', num2str(Output)) ;

guidata(hObject, handles) ;
```

Figure 7: Code for the GUI of example #1.

The first portion obtains the user-defined value and converts it from a string to a number. Recall that *get* will obtain the value while *str2num* will convert the value from a string to a number. The next portion contains the switch construct that evaluates the appropriate function. First the value of the pop-up menu is obtained (recall this defines which string the user selected). Keeping in mind that one represents the word "Select", values from two to seven will evaluate the appropriate function and store the result in the variable *Output*. The next to last line converts

the result to a string and displays that string in the output static text box. Finally, the last line updates the components of the GUI.

VI. EXAMPLE #2

Suppose now we would like to create a GUI that will accept the terms of a user-defined sequence and plot them. This GUI will consist of the following components.

- Four static text boxes (for looks)
- Axes to plot the sequence
- Three edit text boxes to obtain the terms of the sequence and the beginning and ending indices
- Push button to plot the sequence

Figure 8 contains a sample GUI to accomplish this.

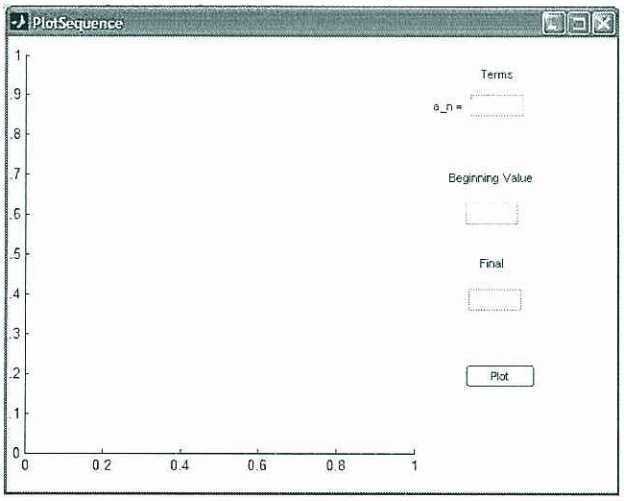

Figure 8: of the GUI for example #2.

Here it is intended that the user will enter the terms of a sequence, the beginning and ending values, and click on the push button to plot the sequence. It should be noted that the user needs to write the sequence using array arithmetic notation. The code to update the axes, obtain the sequence and set it as a function, and obtain the initial and final bounds is shown in Figure 9.

```
axes(handles.SeqAxes) ;

Seq = get(handles.SeqDef, 'String') ;

a_n = inline(Seq, 'n') ;

n_0 = str2num(get(handles.n_0Val, 'String')) ;

N = str2num(get(handles.NVal, 'String')) ;
```

Figure 9: Portion of the code for example #2.

Note that the internal Matlab function *inline* is used to create the form of the sequence. This function will automatically designate whichever variable students use as the index for the sequence. The command *axes(handles.SeqAxes)* ensures that the axes on the GUI is the current working axes.

Evaluating the sequence and plotting the points may be handled in the usual manner. Code to accomplish these tasks and the final GUI appear in Figures 10 and 11, respectively.

```
for i = n_0 : N
    n(i, 1) = i ;
end

Seq = a_n(n) ;

plot(n, Seq, 'o', n, Seq, ':') ;
title(char(a_n)) ;
xlabel('n') ;
ylabel('a_n') ;
grid on ;

guidata(hObject, handles) ;
```

Figure 10: Code to define and plot the user-defined sequence.

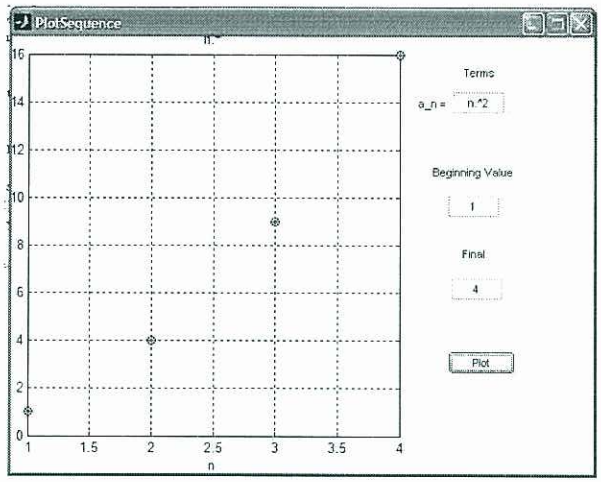

Figure 11: GUI for example #2.

VII. ADVANTAGES/DISADVANTAGES/CONCLUSIONS

Creating graphical user interfaces with Matlab have the advantage of removing the code from the sight of the user. This accomplishes two tasks: the user only has to modify a minimal amount of elements and the instructor does not have to spend as much class time explaining the code. A disadvantage is seen in the second example. In order to input the terms of the sequence, the students still need to understand array arithmetic. Graphical user interfaces provide an excellent means of exposing students to a scientific package with a minimal amount of knowledge of the language.