Using Visual Basic to Create a Graphical User Interface for use with Matlab

1 Why GUI?

One of the problems associated with using software in a course is that some time must be devoted to learning to use the software. This fact implies that there will be less time devoted to the material being covered. A graphical user interface (GUI) makes an application easier to use and therefore leaves more time for learning the topic at hand.

In this paper we will discuss how to create a GUI for Matlab using Visual Basic. In the next section we discuss how to create a GUI using native Matlab commands, following that is a section on creating GUI's with Visual Basic, and finally a section describing how to interface to Matlab from Visual Basic using the dynamic data exchange (DDE).

2 Why not GUI in Matlab?

As one would suspect from the title of the paper, this section will be relatively short. The main purpose of including this material is to convince the reader that Visual Basic is probably the better way to go.

The main drawbacks to creating a GUI in native Matlab are :

- The commands are complex.
- Placing a control requires the programmer to compute its location.
- Revising the GUI requires the recomputation of all the positions of all the objects in it.
- The only way to check your progress is to run the commands and see the results. This fact slows the development process.

These pitfalls are all aspects of the one major disadvantage of using Matlab to create a GUI: the creation of a *visual* object is being done *non-visually.*

Here's a partial GUI for using the MATLAB spline command.

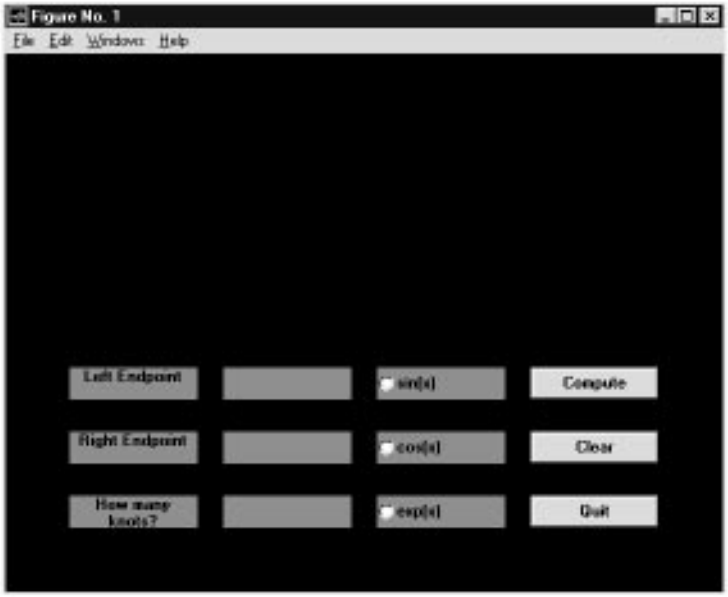

Figure 1: A Graphical User Interface

Here are the commands that are necessary to put the controls in their places. There are no commands associated with the controls.

```
compBtn = uicontrol('style','push',...
                    'position',[410 150 100 25],...
                    'string','Compute',...
                    'callback','a command goes here');

clearBtn = uicontrol( 'style','push',...
                    'position',[410 100 100 25],...
```

```
                    'string','Clear',...
                    'callback','a command goes here');

quitBtn = uicontrol( 'style','push',...
                    'position',[410 50 100 25],...
                    'string','Quit',...
                    'callback','a command goes here');

radio(1) = uicontrol('style','radio',...
                    'position',[290 150 100 25],...
                    'string','sin(x)',...
                    'callback','a command goes here');

radio(2) = uicontrol('style','radio',...
                    'position',[290 100 100 25],...
                    'string','cos(x)',...
                    'callback','a command goes here');

radio(3) = uicontrol('style','radio',...
                    'position',[290 50 100 25],...
                    'string','exp(x)',...
                    'callback','a command goes here');

leftEnd = uicontrol('style','edit',...
                    'position',[170 150 100 25],...
                    'string','',...
                    'callback','a command goes here');

rightEnd = uicontrol('style','edit',...
                    'position',[170 100 100 25],...
                    'string','',...
                    'callback','a command goes here');

numKnots = uicontrol('style','edit',...
                    'position',[170 50 100 25],...
                    'string','',...
                    'callback','a command goes here');
```

```
label1 = uicontrol('style','text',...
                   'position',[50 150 100 25],...
                   'string','Left Endpoint',...
                   'callback','a command goes here');

label2 = uicontrol('style','text',...
                   'position',[50 100 100 25],...
                   'string','Right Endpoint',...
                   'callback','a command goes here');

label3 = uicontrol('style','text',...
                   'position',[50 50 100 25],...
                   'string','How many knots?',...
                   'callback','a command goes here');
```

This is a relatively uncomplicated GUI, although one would be hard pressed to discern this fact from the amount of code required to create it!

3 Why Visual Basic?

In this section, we will attempt to give a feel for how Visual Basic works. The presentation here suffers somewhat because it is static. To truly appreciate the difference between designing a GUI visually and non-visually one should get their hands on a copy of Visual Basic and try it out.

Visual Basic was designed for making GUI's. Figure 2 is a screen shot of a new Visual Basic project. The object in the center is called a form. On the left is the toolbox and on the right is a box which contains all the properties of the form. In the toolbox are items (referred to as controls) that can be placed on the form. These controls include a command button, a radio button, a text box, a label box, a picture box, and many other things. Placing one of these items on the form is done by double clicking on the item in the toolbox and then dragging it to the desired position.

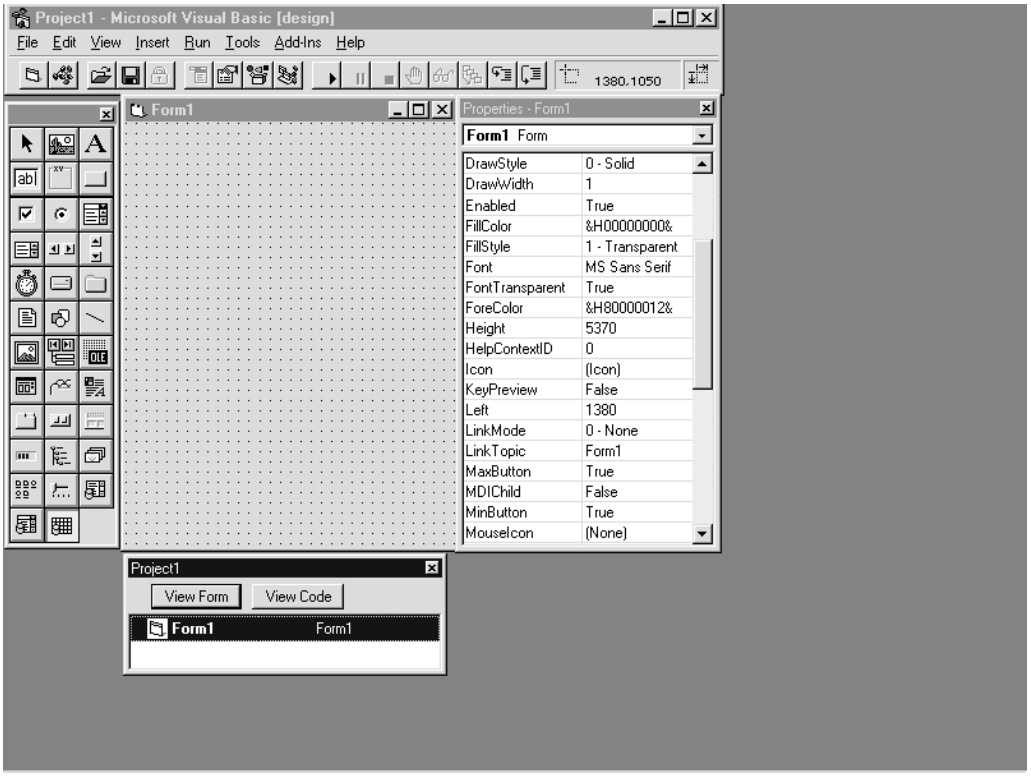

Figure 2: Visual Basic Start-up Screen

In Figure 3, we see a text box that has been placed on the form. Notice that the properties box showing on the right of the form is associated with the text box. The form and every object that is placed on it has a properties box. The appearance of an object can be manipulated by changing its properties. There are also other aspects of the object that can be controlled by changing the properties. For example, usually the name property is changed to something meaningful.

Once we have the form looking the way we want it, we can put together the application. How do we make it do something? A GUI is event driven. The flow of execution is dependent on the action the user takes. For example, if the user presses a button on the GUI, the program handles that event.

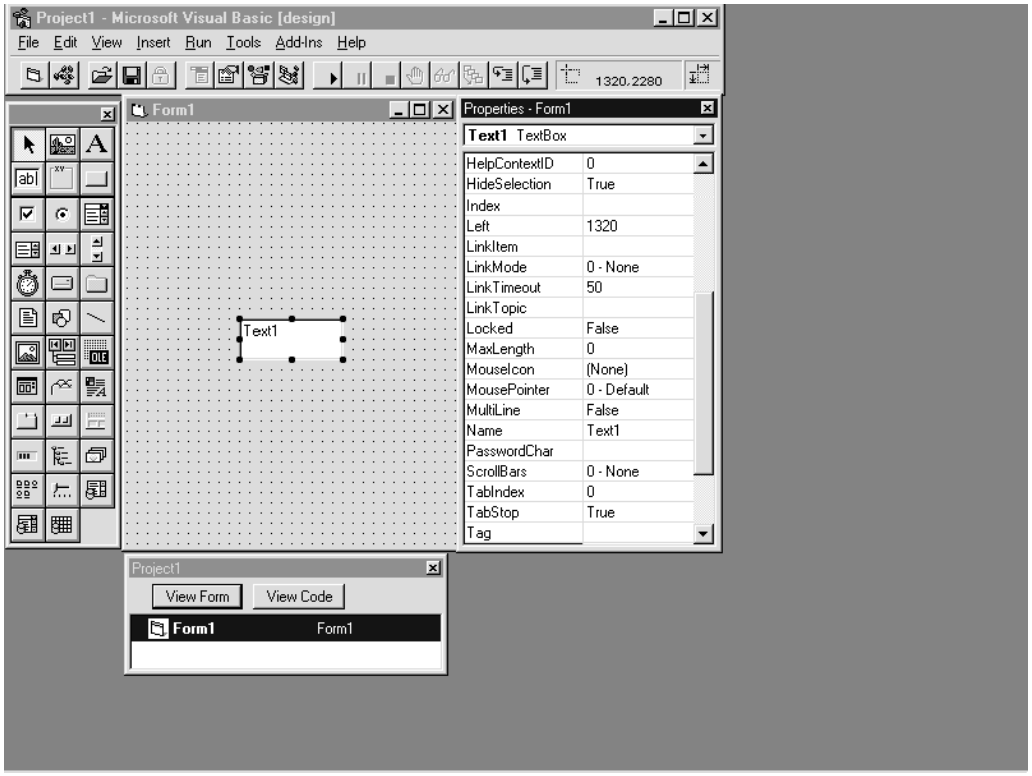

Figure 3: A Text Box

Each control has an event (possible more than one) that goes along with it. To write an event handler for a control, the programmer double clicks on that control to bring up a code window. The Basic code which handles the event goes in it. It is also possible to add to the form a separate code module which holds only subroutines and variable declarations. This feature is particularly useful for adding routines that need to be used by more than one control or that can be used by other applications.

In Figure 4, we see a simple GUI that acts as a front end to the Matlab spline command. In Appendix A is the code for all the event handlers in this application. This application also has a code module for handling communication to Matlab. This topic is discussed in the next section.

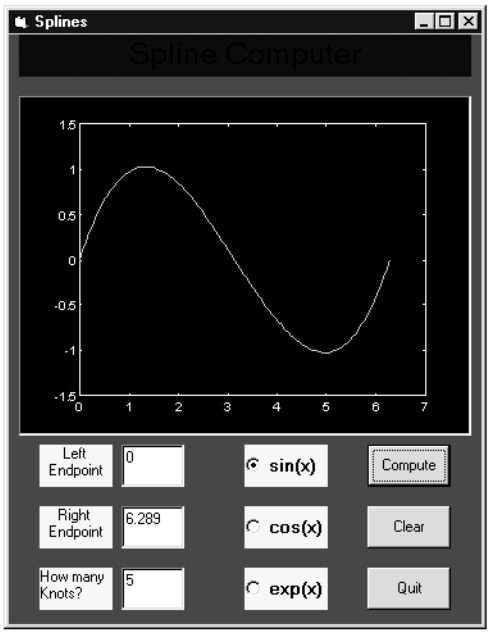

Figure 4: A Visual Basic GUI

4 DDE Basics

A DDE takes place between source and destination applications. In our situation, the Visual Basic program is the destination and Matlab is the source. The destination initiates a conversation with the source by supplying a topic that the source recognizes. The conversation has three possible modes : manual, automatic, and notify. We restrict ourselves to the manual mode. In this mode the programmer explicitly controls when a conversation can happen. Once the topic is known an item is exchanged. The details of this transaction depend on what has been exchanged.

Matlab recognizes the topics System and Engine. Since the Engine is the part of Matlab that does the work, this is the topic that the Visual Basic program uses. The items that can be exchanged are EngEvalString, EngStringResult, EngFigureResult, and a matrix name. The EngEvalString item is used to send commands to Matlab. The EngStringResult is for bringing the result of a comand back from Matlab and the EngFigureResult is for getting a figure back from Matlab. A matrix can also be brought back from Matlab. Data is received from Matlab by executing a LinkRequest and data is sent to Matlab by executing a LinkPoke.

In Visual Basic, every control that can communicate via DDE has link propeties. They are LinkTopic, LinkMode, LinkItem, and LinkTimeout (see Figure 2). The LinkTopic is set to "Matlab | Engine" and the LinkMode is set to 2 (manual). The LinkTimeout is left at its default value. The LinkItem property is used when sending or receiving data. In Appendix B are three routines for communicating with Matlab. The first routine is for moving a number from a text box to Matlab, the second is for sending a command to Matlab, and the last is for getting a figure back from Matlab. These routines can be included in any application that needs to communicate to Matlab.

5 Conclusions

Visual Basic makes creating a graphical user interface painless and it is relatively easy to communicate to Matlab via the dynamic data exchange. If the reader is is interested in developing applications for Matlab, she should consider the advantages of a GUI. If a GUI is deemed to be important then she should consider creating the GUI with a product designed for the task : Visual Basic.

6 References

[1] Norton, Peter *Peter Norton's Guide to Visual Basic 4*, SAMS Publishing, Indianapolis, IN

[2] *External Interface Guide Supplement*, The MathWorks, Inc. 1994

[3] *Matlab Reference Guide*, The MathWorks, Inc. 1992

7 Appendix A

```
Private Sub cmdCompute_Click()
'Sends data to MATLAB then issues commands
'to compute a spline interpolant to the data.

  Dim left As String      'holds left end point
  Dim right As String     'holds right end point
  Dim numKnots As String  'holds number of knots
  Dim command As String   'For sending to MATLAB

  left = "a"      'poke the left point into a
  right = "b"     'poke the right point into b
  numKnots = "n"  'poke the number of knots into n
  Call sendNumber(txtLeftEnd, left)
  Call sendNumber(txtRightEnd, right)
  Call sendNumber(txtNumKnots, numKnots)

  'Set up the knots for the spline
  command = "x = a:(b-a)/(n-1):b"
  'Need a textbox to link?
  Call sendCommand(txtLeftEnd, command)

  'Which function are we using?
  Call whichFunction(txtLeftEnd)

  'points to evaluate the spline at
  command = "xi = a: (b-a)/100 : b;"
  Call sendCommand(txtLeftEnd, command)

  'Call MATLAB m-file
  command = "splineit"
  Call sendCommand(txtLeftEnd, command)
  Call getPicture(picPlotBox)

End Sub
```

```
Private Sub cmdClear_Click()
'clear out the stuff

  Dim command As String 'to clear the plot
  command = "clf"
  Call sendCommand(txtLeftEnd, command)
  Call getPicture(picPlotBox)
  txtLeftEnd.Text = ""
  txtRightEnd.Text = ""
  txtNumKnots.Text = ""
  isPlotReady = False
  optFunc(0).Value = True

End Sub


Private Sub optFunc_Click(Index As Integer)
  'Which function do we pass to MATLAB?

  If (Index = 0) Then
    isSin = True
    isCos = False
    isExp = False
  ElseIf (Index = 1) Then
    isSin = False
    isCos = True
    isExp = False
  ElseIf (Index = 2) Then
    isSin = False
    isCos = False
    isExp = True
  End If

End Sub
```

```
Private Sub txtLeftEnd_Change()

'Check to see if the data is defined before
'we enable the compute button

  If (txtNumKnots.Text = "") Then
    cmdCompute.Enabled = False
  ElseIf (txtLeftEnd.Text = "") Then
    cmdCompute.Enabled = False
  ElseIf (txtRightEnd.Text = "") Then
    cmdCompute.Enabled = False
  Else
    cmdCompute.Enabled = True
  End If

End Sub


Private Sub txtRightEnd_Change()

'Check to see if the data is defined before
'we enable the compute button

  If (txtNumKnots.Text = "") Then
    cmdCompute.Enabled = False
  ElseIf (txtLeftEnd.Text = "") Then
    cmdCompute.Enabled = False
  ElseIf (txtRightEnd.Text = "") Then
    cmdCompute.Enabled = False
  Else
    cmdCompute.Enabled = True
  End If

End Sub
```

```
Private Sub txtNumKnots_Change()
'Check to see if the data is defined before
'we enable the compute button

  If (txtNumKnots.Text = "") Then
    cmdCompute.Enabled = False
  ElseIf (txtLeftEnd.Text = "") Then
    cmdCompute.Enabled = False
  ElseIf (txtRightEnd.Text = "") Then
    cmdCompute.Enabled = False
  Else
    cmdCompute.Enabled = True
  End If

End Sub


Private Sub cmdQuit_Click()
  End
End Sub
```

8 Appendix B

```
Public Sub sendNumber(textControl As Control, itemName As String)
'Send the contents of textControl to MATLAB. Store it in itemName.

  Dim temp As String
  'Save that string so we can put it back later
  temp = textControl.Text
  'MATLAB needs a <LF> on the datum
  textControl.Text = textControl.Text & Chr(13)
  'poke contents of text box into itemName
  textControl.LinkMode = 0
  textControl.LinkTopic = "MATLAB|Engine"
  textControl.LinkItem = itemName
  textControl.LinkMode = 2
  textControl.LinkPoke
  textControl.LinkMode = 0
  'put the number back the way we found it
  textControl.Text = temp

End Sub


Public Sub getPicture(pictureBox As Control)
'Let's go get the plot we made in MATLAB

    pictureBox.LinkMode = 0
    pictureBox.LinkTopic = "MATLAB|Engine"
    'This is the way to get a picture back.
    pictureBox.LinkItem = "EngFigureResult"
    pictureBox.LinkMode = 2
    'LinkRequest asks for data.
    pictureBox.LinkRequest
    pictureBox.LinkMode = 0

End Sub
```

```
Public Sub sendCommand(textControl As Control, MatCom As String)
'This subroutine sends a command to MATLAB for execution.

  On Error GoTo startMatlab 'Make sure Matlab is running
  textControl.LinkMode = 0 'don't talk to MATLAB yet
  textControl.LinkTopic = "MATLAB|Engine" 'the topic
  textControl.LinkMode = 2 'Open a channel
  textControl.LinkExecute MatCom 'tell MATLAB what to do
  textControl.LinkMode = 0 'Close the channel
  Exit Sub

startMatlab:
'This code will start up Matlab.

  If (Err.Number = 282) Then
    I = Shell("c:\matlab\bin\matlab.exe", vbMinimizedNoFocus)
    I = DoEvents()
  End If

End Sub
```