DEVELOPING GRAPHICAL INTERFACES FOR INTERACTIVE APPLICATIONS
IN MATLAB USING GUIDE

A Thesis
Presented in Partial Fulfillment of the Requirements for
the Degree Master of Science in the
Graduate School of The Ohio State University

By
James Edward Quinlan, B.S., M.S.

The Ohio State University
2004

Master’s Examination Committee:
Dr. Bostwick Wyman, Adviser
Dr. Ed Overman

Approved by

[Signature]
Adviser
Department of Mathematics
ABSTRACT

Dynamic interactive applications are useful in mathematics and engineering. These applications can provide the means for simulation, experimentation, scientific calculation, and presentation. As a result, students, scientists, and educators may realize educational and operational value. They enable teachers to illustrate concepts both analytically and graphically. Moreover, they are invaluable for making problem solving for students real and exciting. At the heart of interactive programs is the graphical interface. Because of its cognitive benefits, the graphical interface eliminates learning difficult software packages, in particular, many of the rules of syntax in a specific programming language. Although there are several highly used science/mathematics software applications, one offers an easy to use tool for developing graphical user interfaces for dynamic interactive applications. MATLAB provides a high-level developing language for building interactive software applications utilizing a graphical user interface with the ability to access sophisticated mathematical and graphical capabilities. The following pages contain an exposition for creating interactive programs powered by MATLAB with a well-designed graphical user interface using GUIDE (Graphical User Interface Developing Environment).
DEDICATION

To my daughter, Hannah
ACKNOWLEDGMENTS

I would like to thank Randy Faust for arousing my initial interest in the field of mathematics. Many thanks are given to David Quinlan for his generosity in providing the means to finish this project and future ones, in addition to his word processing skills. I would like to recognize Butch Ramsey for inspiring interest in the fields of graphic design and human-computer interaction. Lastly, I would like to acknowledge Dr. Wyman and Dr. Overman for their agreeable attitudes and positive encouragement – but most of all – for the knowledge they imparted through their mentoring.
VITA

1969......................................................Born – East Liverpool, Ohio

1992......................................................Bachelor of Science, Mathematics,
                                       The Ohio State University

1992 – 1995..........................................Analyst,
                                       Cigna Health Insurance

1995 – 1997..........................................Graduate Teaching Assistant,
                                       Youngstown State University

1997......................................................Master of Science, Mathematics
                                       Youngstown State University

                                       Citizens Bank Company

1998 – Present........................................Graduate Teaching Assistant,
                                       The Ohio State University

FIELDS OF STUDY

Major Field: Mathematics
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Introduction</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>Front-End Design</td>
<td>3</td>
</tr>
<tr>
<td>2.1</td>
<td>Human Computer Interaction</td>
<td>3</td>
</tr>
<tr>
<td>2.2</td>
<td>Graphical User Interface</td>
<td>3</td>
</tr>
<tr>
<td>2.3</td>
<td>Design Principles</td>
<td>4</td>
</tr>
<tr>
<td>2.3.1</td>
<td>User-Centered Design Principles</td>
<td>5</td>
</tr>
<tr>
<td>2.3.2</td>
<td>Graphic Design Principles</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>Back End Design</td>
<td>7</td>
</tr>
<tr>
<td>3.1</td>
<td>MATLAB</td>
<td>7</td>
</tr>
<tr>
<td>3.1.1</td>
<td>MATLAB and M-Files: Programming</td>
<td>7</td>
</tr>
<tr>
<td>3.1.2</td>
<td>Object Orientation</td>
<td>9</td>
</tr>
<tr>
<td>3.2</td>
<td>Handle Graphics</td>
<td>9</td>
</tr>
<tr>
<td>3.2.1</td>
<td>Handle vs. handles</td>
<td>11</td>
</tr>
<tr>
<td>3.2.2</td>
<td>TAG property value</td>
<td>11</td>
</tr>
<tr>
<td>3.2.3</td>
<td>Finding an object’s Handle</td>
<td>12</td>
</tr>
<tr>
<td>3.2.4</td>
<td>Getting and Setting property values</td>
<td>12</td>
</tr>
<tr>
<td>3.3</td>
<td>Conversion commands</td>
<td>13</td>
</tr>
</tbody>
</table>
## LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>MATLAB's Graphic Object Parent-Child Hierarchy Structure</td>
<td>10</td>
</tr>
<tr>
<td>4.1</td>
<td>The three ways to open GUIDE from MATLAB</td>
<td>15</td>
</tr>
<tr>
<td>4.2</td>
<td>GUIDE Quick Start Dialog Box</td>
<td>16</td>
</tr>
<tr>
<td>4.3</td>
<td>GUIDE's Layout Editor</td>
<td>16</td>
</tr>
<tr>
<td>4.4</td>
<td>GUIDE Toolbar located in the Layout Editor</td>
<td>19</td>
</tr>
<tr>
<td>4.5</td>
<td>Property Inspector Window</td>
<td>20</td>
</tr>
<tr>
<td>4.6</td>
<td>Window illustrating all graphics objects plus Align Object Window</td>
<td>21</td>
</tr>
<tr>
<td>5.1</td>
<td>MATLET window from Example 1</td>
<td>26</td>
</tr>
<tr>
<td>5.2</td>
<td>Layout of Example 2</td>
<td>28</td>
</tr>
<tr>
<td>5.3</td>
<td>MATLET Window from Example 2</td>
<td>29</td>
</tr>
<tr>
<td>Table</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>-------</td>
<td>--------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>3.1</td>
<td>Commonly used MATLAB conversion commands</td>
<td>13</td>
</tr>
<tr>
<td>4.1</td>
<td>A list of Interface Objects and their function</td>
<td>17</td>
</tr>
<tr>
<td>4.2</td>
<td>Six kinds of Dialog Boxes</td>
<td>23</td>
</tr>
</tbody>
</table>
PREFACE

This exposition will provide an explanation of the process of designing and developing interactive MATLAB applications, or in other words, the front-end interface and the back-end engine or program. The goal is to enable the reader to acquire sufficient knowledge and skills to begin their own interactive MATLAB tasks in a reasonable amount of time with only a minimal set of prerequisites.

It is not enough to merely state the necessary command lines and syntax of the programming language, but to elaborate on the entire design process. Emphasis will be placed on the principles of layout and graphic design to ensure quality graphical user interfaces as well as the function prototypes, commands, syntax, and program specifics. Although the primary purpose is pedagogical, it is definitely not a tutorial in mathematics or computer science.

Chapter 1 is a short introduction to the background, history, and motivation of using graphical interfaces. A rationale as to the choice of MATLAB is also provided.

Chapter 2 covers the front-end design principles that apply to the layout of the interface. Although some overlap exists, this section is divided into two more-manageable sub-sections: user-centered and graphic-layout design. Almost certain, this step of the process is the most overlooked and taken for granted. Ironically however, in terms of effective interfaces, it is the most important step. Serving as a literature review, it provides the theoretical framework for the importance of the graphical user interface to human-computer interactions. A brief history of the graphical interface is included.

Chapter 3 is concerned with specifics of interaction, control, and function of the application, the back-end. A look at object-oriented programming is practical and some MATLAB programming is covered. The main purpose of this chapter is an introduction to
Handle Graphics and how object-oriented programming applies. It is essential to the topic at hand to become familiar with Handle Graphics and the commands that manipulate them.

Chapter 4 is the highlight of this effort. Learning how to use MATLAB’s interface developing tool to create a small interactive application is the point of this paper. The focus of the chapter is the set of interface objects, their properties and their methods. Finally, obtaining and changing properties of an object to create the dynamic interactivity culminates the chapter.

Minimalist learners and those not interested in the theoretical framework and literature review should go directly to Chapter 5. There, the reader will gain immediate experience building interactive MATLAB applications. Two small projects are designed with “pointers” to illustrations demonstrating the process. As needed, the reader may consult Chapters 2, 3, 4 and Appendices.

Most steps refer to a Figure or Screen-shot. Illustrations are considered integral to the learning process. The value of illustrations will become self-evident to the reader. Moreover, while studying educational outcomes, Mayer found that illustrations enhance the construction on internal mental models and benefits revealed in higher order problem solving. [14]

Appendix A contains several of MATLAB’s symbolic toolkit commands. Students and teachers should find these commands worthwhile as some of the most commonly used commands for symbolic calculation and manipulation. Appendix B has the object callback methods as seen in the M-File from Example 2.

CONVENTIONS

If a term appears in the Glossary, it is italicized the first time it appears in the text. Bold face type indicates a built-in MATLAB function or command. M-File code and MATLAB examples are denoted by a different font. The MATLAB prompt as seen from the command window is represented by: >>.

---

1 Principles of Minimalist Learning theory, according to John Carroll, are based on brevity, simplicity, organization, and participation in meaningful activity and prevention of errors. [4]
CHAPTER 1

INTRODUCTION

There is little doubt that technology is a valuable part of our modern existence. Further, it plays an important role in mathematics, science, engineering, social science, economics, and consequently in both teaching and learning. In particular, computers and software have increased research opportunities through their ability to perform complicated calculations in less time. Research in differential equations, queuing theory, graph theory, and combinatorial design – to only name a few – utilizes computers. In fact, the proof\(^2\) of the Four-Color problem used a computer program.

Certain educational software enables teachers to illustrate concepts both analytically and graphically. University students often make use of sophisticated mathematical applications for problem solving and presentations. According to Nickerson and Landauer, tools for mathematical problem solving fall into one of two categories: those that facilitate the application of mathematical principles and methods; and those that are mechanical aides to computation. [15] Interactive mathematical software designed with a graphical user interface can provide a powerful analytic and computational tool as well as an exciting teaching and learning device.

Mathematica\(^\circledR\), Maple\(^\circledR\), and MATLAB\(^\circledR\) – widely used mathematical software packages – make difficult computations easy. For instance, this software can effortlessly find the determinant of a 10-by-10 matrix or calculate the integral of a hyperbolic trigonometric function. In addition, with powerful graphing capabilities, analysis of certain problems becomes more efficient. Maple provides the ability to create interactive applets with a graphical user interface; however the process is difficult because a high-level developing tool

\(^2\) Some mathematicians do not recognize this as an actual proof.
is not available. Conversely, MATLAB incorporated a GUI builder that is easy to learn and aides with the development of graphical user interfaces.

Although many applications might not need a graphical interface to successfully accomplish a task, a GUI can be used by a larger number of non-technical users, (i.e. non-power users). In addition, a GUI fosters experimentation by allowing more than one parameter or variable to be changed at any time during the execution. Furthermore, unlike linear execution programs in which one input mistake may force a user to complete the entire execution process, the GUI is the front-end of a dynamic program that accepts input in a non-linear fashion. Moreover, one mistake is not terminal; a well-designed GUI provides the user with a means of recovering from their mistakes.

The graphical user interface is a psychological aid to mathematical analysis. Perhaps Alfred Whitehead North said it best: “The whole of mathematics consists in the organization of a series of aids to the imagination in the process of reasoning” [20].

There are several reasons to develop interactive applications with a graphical user interface. For example, Hanselman and Littlefield suggest the following possible reasons [6]:

1. Developing an application for others to use;
2. Creating an interactive demonstration of a process, technique, or analysis method;
3. Writing a utility function such that text boxes make better sense for input and for output;
4. GUI’s are fun to develop and use.

There are two steps in developing interactive applications with an interface: Front-end and Back-end design. The front-end, or the interface, is the point of contact between the user and the computer. It is what the user sees and feels. The back-end design refers to the actual code creating functionality of the program including the interactions with the user.
CHAPTER 2
FRONT-END DESIGN

2.1 HUMAN COMPUTER INTERACTION

Human-Computer Interaction (HCI) is a multi-discipline field that combines cognitive psychology and computer science. In the most general terms, Human-computer interaction is a discipline concerned with the design, evaluation and implementation of interactive computing systems and phenomena surrounding them.[8]

Its goal is to enhance human usability, functionality, and interaction with computers. To achieve this goal the HCI analysis must incorporate into the human-computer interface familiar mental models, means of feedback, a sense of affordance\(^3\), and general consistency. Most of these concepts are addressed from a cognitive science perspective.

Cognitive science deals with human senses, perception, memory, reasoning, and learning. In HCI, the focus is on how the user knows, and not how the user responds. The interface design generally contributes about the same number of errors as instruction or training factors. Therefore, it is a major concern to system designers.[2]

2.2 GRAPHICAL USER INTERFACE

The history and timeline of the graphical user interface starts in 1973 at the Xerox research center in Palo Alto, California with the development of a computer that used a computer interactive device called a mouse. In 1974, Xerox wrote the first WYSIWYG (What You See Is What You Get, pronounced “wiz·e·wig”) application called Bravo. In

\(^3\) A technical term used to define the perceived properties of an object. A button is to be pushed. [17]
1977, Apple released the Apple II and established a benchmark for personal computers. Since then, Apple Computer has become the authority on user-centered development and interface design. Must-have references for the GUI designer are: [1], [7], [10], [17], and [18].

Graphical User Interfaces (GUI, pronounced “gooey”) are the point of contact between the user and computer; it “provides a means through which individuals can communicate with the computer without programming commands.” [13 – p. 385] The graphical user interface has two primary goals: communication and interactivity. In order to achieve these goals, the GUI must accurately and efficiently convey the task, relate to the culture of the user, and correspond to the user’s education and experience. User-centered design applies the Strong Thesis of Metaphor [9] to achieve these goals.

The design process involves computer programmers, cognitive scientists, graphic artists, and even the users themselves. To create quality applications, the GUI developer(s) must follow some basic design principles. These principles are the framework for functionality and usability (performance and preference). Fundamental principles are grounded in cognitive consideration. Further aspects of these design considerations are related to structure, organization, and aesthetics.

The first step in the design process is to determine the intended use of the program and to identify the target audience or user. That is, who will use it and how? Therefore, it is worthwhile to focus considerable effort in the function and form of the interface. Although there are hundreds of pages covering design principles of the computer interface, for the sake of brevity, only the most fundamental principles are extracted from those volumes. For a start on more in-depth coverage of design principles consult [1], [2], [3], [7], [10], and [18].

2.3 DESIGN PRINCIPLES

In particular, design principles refer to the front-end, the design of the user interface. There are two facets of front-end design: user-centered and graphic. There are five user-centered design principles that will be sufficient for the purpose of these pages and three graphic design principles. The user-centered design principles are referenced in example by DP-1, DP-2, DP-3, DP-4, and DP-5. These principles are paramount in human-computer interaction through human cognition.
There are several general ‘rules of thumb’ in the design of the user interface that are based on the principles below. A few ‘rules of thumb’ are: know the user; to err is human; error messages should be meaningful; provide help; user controls the system; every action has a reaction; things that look the same should act the same; and things that look different should act different.

2.3.1 User-Centered Design Principles

Design Principle 1 (DP-1): Simplicity – less is more  
   - Be direct and to the point – leave out “bells and whistles”

Design Principle 2 (DP-2): Consistency  
   - Use consistent text fonts, and sizes. Avoid extremely small fonts
   - Use Italic, Bold, and Underline sparingly.

Design Principle 3 (DP-3): Familiarity  
   - Create similar look and feel to widely popular programs and operating systems
   - See and Point – WYSIWYG
   - Use Metaphors for explanation of concepts
   - Use mental models

   - Encourages experimentation without negative repercussions
   - Does not jeopardize document security - changes are not lost or permanent
   - User-control – provide opportunity for recovery
   - Reduces user frustration and anxiety

Design Principle 5 (DP-5): Immediate feedback response  
   - Keep the user informed – Minimize frustration⁴
   - Maximize task efficiency – easier to learn
   - Provide help – TooltiptString property value
   - Use dialogue boxes for information or to alert user of possible errors

⁴ According to Bodker, response time of the computer should be no more than 2 seconds. Ideally it should seem continuous. [3]
The first principle is the most understated principle. Simple is elegant. Complicated and cluttered GUIs have a negative effect on usability. Each MATLET should try to accomplish one simple task or goal, such as simulating the “Monty-Hall” dilemma. Remember, the overall goal of the applet is to make it easy to use by others. Users should be familiar with the navigation of the software or it should be easily learned. It should be built similar in form to standard applications such as email programs and word processors in which the user points and clicks.

2.3.2 Graphic Design Principles

Aristotle said that the chief forms of beauty are order and symmetry. Many of the graphic-design principles are simple corollaries of the overall design principles. Visually unpleasing layouts are distracting. [10] Principles of graphic design and layout are directly related to order and symmetry, the aesthetics.

Graphic Principle 1 (GP-1): Use High Contrast
  - White background with black letters – e.g. books, newspapers
  - A frame can set-off similar objects and controls from others
  - Use colors to highlight important information or controls

Graphic Principle 2 (GP-2): Have White Space
  - White-space is visually pleasing
  - DO NOT USE ALL CAPITALS - Capitals reduce white space
  - Corollary of DP-1

Graphic Principle 3 (GP-3): Keep Organization
  - Properly align all objects – use the alignment tool within GUIDE (Figure 4.4)
  - Screen should look “clean” and readable. Easy to find controls
  - Corollary of DP-1, DP-2, DP-3
CHAPTER 3

BACK-END DESIGN

The interface is designed with functionality in mind. Without function, the interface is just an empty shell. After creating the layout of the interface, the “look”, computer code must be added behind the object elements of the interface to provide the “feel”. This chapter covers the back-end coding necessary to make the MATLET functional.

3.1 MATLAB

Developed in the 1970s for numerical analysis and matrix operations, the software application MATLAB stands for MATrix LABoratory. Primarily used for linear algebra operations, MATLAB can efficiently perform a wide range of mathematical processes, including: Fourier analysis, data analysis, cubic splines, optimization problems, and differential equations. The combination of the built-in math and graphic functions make MATLAB preferred for technical computing over programming languages like FORTRAN and C. Recently, MATLAB was upgraded with the ability to do symbolic manipulation.

3.1.1 MATLAB and M-Files: PROGRAMMING

All MATLAB elements (e.g. numbers, polynomials, and graphic handles) are stored in MATLAB’s memory as matrices. For instance, a number is stored as a one-by-one matrix. MATLAB can be programmed. Comparable to other programming languages, MATLAB uses basic flow control, branching, loops, and logic.

MATLAB is case-sensitive. A semi-colon is optional at the end of every line. The presence of a semi-colon suppresses the output; whereas the omission of the semi-colon
displays the result in the MATLAB command window. Although it is possible to write program code in the command window, it is better to create a function file, called an M-File, named for the .m file extension accessible from anywhere in MATLAB. Every function in the M-File has the following syntax:

```
function [output variable(s)] = function_Name(parameter list)
```

For example, suppose you create a function named `add_it`, which takes two numbers and returns their sum. The syntax in the M-File would be:

```
function z = add_it(x,y)
    z = x + y;
    return
```

To apply this function from the command window, type the following (result shown indented on the next line):

```
>> add_it(1,2)
3
```

Because interactive applications require user defined input, which MATLAB considers as a string vector, an extremely useful built-in function is the `eval(f)` function, where `f` is a string that represents a mathematical expression such as: \(2x + 1\). However, the `eval` function requires more MATLAB resources than the `feval` function. That is, `feval` does the same thing as `eval`, but more efficiently. [6] Below is an example as seen from the MATLAB command window.

```
>> % Evaluates the string '2x + 1' at x = 1
>> x = 1;
>> f = '2*x + 1';
>> eval(f)
3
```

In the M-File, the percent symbol (%) indicates a comment line and is ignored during execution of the M-File. It is good programming practice to maintain a well-documented program by adding appropriate comment lines. The lines immediately following the function declaration should contain comments that are displayed as the function’s help message (DP-5), `>>help function_Name`.

8
An M-File may include several functions. Any function following the main function is a sub-function or local-function. The first function’s name matches the name of the M-File. The GUI M-File will have many sub-functions – at least one for each graphic object. For comprehensive coverage of MATLAB, consult Hanselman and Littlefield’s *MATLAB 6: A Comprehensive Tutorial and Reference*. [6]

3.1.2 *Object Orientation*

Interface applications in MATLAB are based on object-oriented programming (OOP), similar to Java, C++, and Visual Basic. OOP forms a mapping between real world objects and software counterparts. [5] It exploits the parent-child paradigm by taking advantage of inheritance relationships, whereby new classes inherit characteristics of an existing class while also possessing unique characteristics of their own. [5 – p. 45]

Objects – similar to their metaphorical counterparts – have state and behavior. That is, associated with each interface object (e.g. Pushbuttons and Edit Text boxes) are characteristics or properties, sometimes referred to as instance variables (referred to here as *handles*). In MATLAB, all interface objects have several universal properties such as Parent, Child, TAG, and Type. Moreover, each object also has unique properties. For instance, lines have a property called LineWidth whereas Checkboxes do not. A list of an object’s properties can be viewed – and changed – within GUIDE by double-clicking on an object to open the object’s Property Inspector window, (Figure 4.5). A good habit to establish is to set an object’s default properties immediately upon creation. This will reduce later complications in the program and help to eliminate certain unexpected features. Although the default properties are set at creation, obtaining and modifying a property value during run-time is the essence of an interactive application. The object’s behavior or actions take place from sub-functions called Callback Functions, or *Callbacks* for short.

3.2 *HANDLE GRAPHICS*

The key to effective programming and interface design in MATLAB is mastering Handle Graphics. Every graphical object (e.g. Figure Windows, axes, interface controls,
menus, etc.) has a unique identifier called a Handle, which enables access and control of the graphics object. A Handle is essentially a discrete pseudo name that allows access to an object. It is either an integer or floating-point number. The root (i.e. the screen) and Figure Windows are assigned integer values. The Handle of the root is always 0. MATLAB assigns a whole number – starting at 1 – as the Handle for each Figure Window. All other graphics objects will be assigned a distinct floating-point number. A Handle cannot be changed until the object is deleted. Based on the standard Parent-Child relationship model, the graphics hierarchy structure is illustrated in Figure 3.1. [13 – p. 199]
3.2.1 Handle vs. handles

By and large, MATLAB literature refers to an object’s properties as just that – properties. However, students may find it useful, albeit unconventional, to refer to those properties as handles thereby making the important distinction between an object’s Handle and the object’s handles. A Graphics Handle (Handle for short) is the unique name of an object, the “self”, whereas handles are the properties of the object or attributes of the “self”. An appropriate language arts analogy would be: Handle is a noun and handles are adjectives describing the noun. Therefore, the author considers the terms properties, instance variables, characteristics, and handles interchangeable. Callback functions are methods of an object and correspond to a verb, continuing with the English analogy.

3.2.2 TAG Property Value

There are essentially two ways to work with a Handle, either by keeping track of it at the time the object is created and recalling the Handle using the guidata command, or on a “need-to-know” basis. Because MATLAB provides an easy way to obtain an object’s Handle, the latter will be the preferred method.

The premise of the user interface is the ability to change an object’s property at runtime. Interaction is complete when the program acts on the user-entered data and displays the results. Therefore, obtaining and changing an object’s properties during runtime is the crux of the matter; it adds the functionality and interactive feel. After obtaining an object’s Handle, access is available to any of its properties. However, before discussing manipulation of an object’s properties, it would be beneficial to point out the importance of the TAG property to this paper’s approach of working with Handle Graphics. Just as MATLAB uses the object’s Handle in any reference to the object, the developer will use the object’s TAG property. The value of the TAG property is a unique identifier to the developer and is universal to all objects. Upon creation of an object, it is highly recommended to assign a meaningful name (string vector) to the TAG property. When it
comes time to obtain or change an object's property value, that object will be selected by its TAg property. (See Section 3.2.4)

3.2.3 Finding an Object's Handle

Sometimes it is necessary to first find the object before obtaining its Handle in order to manipulate one of its properties. If this is the case, the built-in findobj command will return the Handle of an object that has a certain property. The syntax is:

```
h = findobj('Property_Name', 'Property_Value');
```

In particular, if two axes are on the same Figure window, a graph will be plotted on the last axis made current by the user, which in some cases is not according to the intended design (e.g. comparing two graphs).

3.2.4 Getting and Setting Property Values

When working with Handle Graphics, there are two basic tasks:

1. Get a property value;
2. Set a property value.

The two MATLAB commands for performing these functions are aptly named, get and set. The motives for using the get command are:

1. To assign a variable a certain Property Value of an object;
2. To use an object's Property Value in a calculation.

The syntax to obtain an object's property value is:

```
get(handles.TAG,'Property_Name');
```
The only purpose for using the `set` command is to change a particular Property Value of an object at run-time. The general syntax to set an object's property value is:

```matlab
set(handles.TAG, 'Property_Name', 'New_Value');
```

The code above roughly translates as: set a property of the object 'named' TAG, to the new value. MATLAB also includes short cuts to obtain the Handle of current graphics objects. These commands are `gcf`, `gca`, and `gco`; get current figure, get current axes, and get current object, respectively. A current object is an object most recently activated, by creation, manipulation, or a mouse click. [13]

### 3.3 CONVERSION COMMANDS

User-input must typically be converted to a format that MATLAB recognizes. For example, when assigning a floating-point variable the contents of an Edit Text's String Property (a string), first convert the data using the `str2num` or `str2double` command\(^5\). The most common and useful built-in functions are listed in Appendix A. Presented below are the conversion commands used in Example 1 and Example 2 in Chapter 5. For more information, type "help" followed by the command in question. For example:

```matlab
>> help str2double
```

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>str2double</code></td>
<td>Converts a string to a double precision number</td>
</tr>
<tr>
<td><code>char</code></td>
<td>Builds a string from different data types.</td>
</tr>
<tr>
<td><code>sym</code></td>
<td>Creates a symbolic variable.</td>
</tr>
<tr>
<td><code>poly2sym</code></td>
<td>Changes a polynomial coefficient vector to a symbolic representation.</td>
</tr>
<tr>
<td><code>sym2poly</code></td>
<td>The opposite of poly2sym. Changes standard polynomial to coefficient vector.</td>
</tr>
<tr>
<td><code>vectorize</code></td>
<td>Inserts a &quot;dot&quot; before / and ^ to do vector operations.</td>
</tr>
</tbody>
</table>

Table 3.1 - Commonly used MATLAB conversion commands

---

\(^5\) Note the difference between `str2num` and `str2double`. Use appropriately. It is recommended to use `str2double` in most case as there are a few other problems that can occur by using `str2num`
CHAPTER 4

GUIDE

There are two ways to create graphical user interfaces in MATLAB: a high-level and a low-level. MATLAB has an Integrated Developing Environment (IDE), also known as high-level programming. In IDE, the developer creates a graphic object simply by ‘dragging’ an object from a design palette to the Layout Editor (Figure 4.3). Whereas low-level programming entails writing program language instructions – computer code – for the actual objects.

GUIDE (Graphical User Interface Developing Environment) is MATLAB’s Integrated Developing Environment for creating MATLETS\(^6\) and is similar to software development environments such as Visual Basic®. “User Interface Design Environments automate or assist the work of designing user interfaces by providing a kit of interface parts such as buttons, menus and forms, each with appropriate functionality, which can be fitted together into a complete user interface” [1] and reduce the need to manually enter computer code. It does, however, require computer code to be written for the desired action of the object (e.g. pushbutton, menu, or slider). That is, once a graphic object is created, the functionality behind the object is provided by computer code. In short, it is GUI software to build GUIs. The intent is to simplify development of software applications by experts and novices alike.

4.1 LAUNCHING GUIDE

To begin building a MATLET, start GUIDE using any of the following methods. (See Figure 4.1)

\(^6\) MATLET: Author-coined term for interactive MATLAB applet.
1. Type guide at the MATLAB prompt.

   ```matlab
   >> guide
   ```

2. From the menu, select File – New – GUI.

3. Click the Start button on the bottom left of the screen. Highlight MATLAB. Click GUIDE (GUI Builder).

![MATLAB GUI](image.png)

Figure 4.1 –The three ways to open GUIDE from MATLAB

The GUIDE Quick Start dialog box appears (Figure 4.2). At first, create a new GUI. Save the file as you would normally save any file on your computer. If a MATLET has previously been created, click on the “Open Existing GUI” tab at the top of the Quick Start dialog box and choose the application file name. It is important to note that GUIDE immediately creates two files: a `figure` file (.fig) and an `M-File` (.m). A blank figure window appears as the LAYOUT Editor (Figure 4.3).
Figure 4.2 – GUIDE Quick Start dialog box. Create a new GUI or open existing

Figure 4.3 – GUIDE’s Layout Editor
4.2  \textit{THE INTERFACE IN GUIDE}

Primitive days of the Command Line Interface - keyboard only input and
commands- are history. Interactive controls and objects have become the standard for
interface design. "The set of user interface components supplied with MATLAB allows you
to design GUIs that match those used in sophisticated software packages." [13– p. 385]

4.2.1  \textit{Interface Elements}

Interface objects are the elements the user will interact with, providing the user a
means for input and control. Graphical objects usually have a metaphorical meaning
associated with them (DP-3) for cognitive purposes. The basic elements of the interface are
the Figure Windows, Uicontrols, Uimenu, and Axes, which are objects that belong to
graphic classes. Uicontrols are the most typical user objects. Table 4.1 lists all of
MATLAB's Uicontrols in the order they appear in the Layout Editor. Refer to Figure 4.3
and Figure 3.1.

<table>
<thead>
<tr>
<th>OBJECT</th>
<th>FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pushbutton</td>
<td>Used to select an event, perform an operation or calculation, reset or cancel</td>
</tr>
<tr>
<td>Togglebutton</td>
<td>Switches between two choices</td>
</tr>
<tr>
<td>Radiobutton</td>
<td>A toggle switch (On / Off). Exclusive choice, one and only one option</td>
</tr>
<tr>
<td>Checkbox</td>
<td>An inclusive choice. Allows user to select many options</td>
</tr>
<tr>
<td>Edit Text</td>
<td>Basic user input field. (Can be also used as an output box)</td>
</tr>
<tr>
<td>Static Text</td>
<td>Labels or indicators for user input or computational output. Disabled to the user.</td>
</tr>
</tbody>
</table>
Slider  Scroll bar that allows dynamic change in a variable value
Frame   Organizes objects or information
Listbox Displays several lines of output. Similar in looks to a spreadsheet
PopupMenu Provides multiple choices for user. Used if various options are necessary
Axes    Used to display graphs of lines and equations

Table 4.1 - A list of Interface objects and their function

4.2.2 Adding Objects

Adding objects to the Layout editor to create the GUI is the easiest part of the process. These objects are easily assembled to create a GUI. The developer literally “drags” an object from the object palette to the blank Figure Window. (See left side of Figure 4.3.) The real benefit of a high-level developing tool (IDE) is realized at this step. Creating graphics object in a low-level programming language on the other hand usually requires importing a graphics library package and writing code for each object.

4.2.3 Setting Default Properties

Immediately after creating an object, double-click on the object to open the Property Inspector Window (Figure 4.5). Set all default properties as desired. Among the most important properties are: TAG, String, Enable, Horizontal Alignment, Position (See Section 4.2.4), and Tooltip String (DP-5).

The TAG property is the object’s name assigned by the programmer (See Section 3.2.2) while the String property is the initial value of the object the user will see. Unless there are specific reasons to leave an object’s String property blank, initial values should be assigned that will allow the applet to run without user defined values. It also provides the user with information as to what type of data is expected in the field.
4.2.4 Aligning Objects

Focus on consistency in alignment (GP-4). Use similar alignments for all objects, i.e. left, center, or right-justify. Proper alignment supplies quality aesthetics. Beyond setting the default Horizontal Alignment Property, either use the Alignment Tool provided by GUIDE (Figure 4.4), or set the Position Properties of the object manually.

Select more than one object to be aligned by holding down the shift key while clicking on the desired objects. Align objects both horizontally and vertically. However, do so one direction at a time. If aligning five checkboxes horizontally, click the Vertical Align switch to OFF before applying the change. (See Figure 4.6)

Figure 4.4 – GUIDE Toolbar located at the top of Layout Editor. From left to right: Alignment Tool, Menu Editor, M-File Editor, Property Inspector, Object Browser, and the green arrow runs the application
Figure 4.5 – Property Inspector Window. The TAG property is highlighted
4.3  **ADDING CODE TO THE CALLBACKS IN THE M-FILE**

In order to provide object functionality, add the code to the callback function by clicking on the M-File Editor icon located at the top of the Layout Editor window (Figure 4.4). A new window opens containing the code of the entire project. For each object that requires action, locate its callback sub-function. The executable code will be added after the last comment line. For example, suppose the user clicks a button (dubbed ACTION). The callback needs to get the String value of an Edit Text field (tagged TEXT_FIELD) and assign the value to the variable f. Add the following line of code to the end of the code segment as noted in the example below:

```matlab
f = get(handles.TEXT_FIELD, 'String');
```

% --- Executes on button press in ACTION.
function ACTION_Callback(hObject, eventdata, handles)
    % hObject    handle to ACTION (see GCBO)
    % eventdata  reserved - to be defined in a future version of MATLAB
    % handles    structure with handles
    f = get(handles.TEXT_FIELD, 'String');
```
4.3.1 Error Handling

Errors come in many varieties, but two broad cases are: design errors and user input errors. User input errors can cause run-time errors. Often times, the user will enter data into an Edit Text field that MATLAB might not understand. In order for the MATLET to run smoothly, the developer needs to expect errors (e.g. entering zero as a divisor) and must account for this possibility by handling them promptly and properly. One possibility is to confirm input 'up-front' with a Message Dialogue box. Another way is to catch the error with a built-in function. Similar to other programming languages, MATLAB uses the **try-catch** command codes. Using these commands in conjunction with Dialog boxes can prevent errors and construct stable applications. The comma after the statements and before the catch and end is depicted in bold type to indicate significant syntax. The syntax for the **try-catch** command is:

```matlab
try statement, ..., statement, catch statement, ...statement, end
```

For example:

```matlab
x=eye(2);
y=eye(3);
try z=x*y, catch msgbox( 'Incompatible matrix sizes' ), end ;
```

4.3.2 Dialog Boxes

Dialog boxes are useful in providing feedback and in preventing errors from occurring. They also can confirm a choice made by the user. There are six varieties listed below including syntax and usage. The general command syntax for a dialog box is in bold below and the six different kinds of dialog boxes are listed in Table 4.1.

```matlab
dialog_Box_Type ( 'Message/ Warning/ Question/ etc ', 'Window Name' );
```
<table>
<thead>
<tr>
<th>TYPE</th>
<th>SYNTAX</th>
<th>USAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error</td>
<td>errordlg</td>
<td>An actual error message with possible ways to recover</td>
</tr>
<tr>
<td>Help</td>
<td>helpdlg</td>
<td>User invoked dialog box for displaying hints or help</td>
</tr>
<tr>
<td>Input</td>
<td>inputdlg</td>
<td>Prompts user for input data</td>
</tr>
<tr>
<td>Message</td>
<td>msgbox</td>
<td>An information box, simple feedback</td>
</tr>
<tr>
<td>Question</td>
<td>questdlg</td>
<td>Prompts with choice. e.g. “Do you want to quit?”</td>
</tr>
<tr>
<td>Warning</td>
<td>warndlg</td>
<td>An information box, warning user of potential error</td>
</tr>
</tbody>
</table>

Table 4.1 – Six kinds of Dialog Boxes

4.4 **RUNNING THE MATLET**

Running the MATLET is as easy as pressing a button. In fact, to run the MATLET press the green arrow button located at the top of the Layout Editor window. (Figure 4.4) Stand-alone applications can be built and run without MATLAB being loaded on the host machine by using MATLAB’s professional version along with MATLAB runtime server.

4.4.1 **Debugging**

A *bug* is an unexpected action or result. There are three different types of errors: syntax errors, logical errors, and runtime errors. A logic error is a flaw in the logic of the program – for example, addition instead of multiplication. A syntax error is usually a typographical error in a command line or function. Attempting to multiply non-compatible matrices or dividing by zero are examples of runtime errors. MATLAB provides tools for debugging. From within the M-File editor window, use the commands under the Debug and Breakpoint menus.

Run the program; test various inputs and options available to the user. Let several potential users test-run the MATLET. Nielsen & Landauer (1993) hypothesized that one-half of the flaws are discovered by two users, three-quarters with four, and so. [16]
CHAPTER 5

EXAMPLE PROJECTS

Through two examples, this chapter covers the MATLET development process from start to finish. It includes hardware specifications, developing environment start-up, design and layout of the interface, and adding code to the M-File to create the functionality. The process is illustrated with figures included at each step. Well-commented code for each callback is contained. In the first example, we will build a MATLET that will approximate the integral of a user-defined function on an interval by calculating the left Riemann sums. Actually, MATLAB provides a built-in function in the symbolic toolkit called rsums, which makes this task trivial.

5.1 INSTRUMENTS

Hardware: Macintosh PowerBook 1GHz G4, 768MB RAM
System Software: Mac OS 10.3 (Panther)  
Application Software: MATLAB 6.5 Student-Version Release 13 for Mac OS X  
Development Software: MATLAB GUIDE

5.2 EXAMPLE 1 - APPROXIMATING A DEFINITE INTEGRAL

Objective To compute an estimate of a definite integral of a user defined function on a given interval by left hand Riemann sums.

7 To run MATLAB on Panther, download a patch from Mathworks.com

8 Documentation CD must be in the CD-ROM drive at start-up. After starting, the CD may be removed
5.2.1 Layout and Design

1. Open a blank Figure window. Double click on the Figure to open the Property Inspector (Figure 4.5). Change the Background color. By GDP-1, white is best. Also, scroll down and change the TAG property.
   
   Note: The best practice is to change the objects properties first upon creation. This avoids later conflicts and is a good programming design habit.

2. Place a frame on the blank figure that will contain all other graphic objects, GDP-3.

3. Place three more rectangular frames on top of the first frame from step 2, one for each set of related objects (input and output). In this case, setup the frames as follows: (1) endpoints of the interval and the number of sub-divisions; (2) user-defined function; (3) output results. Label these three frames similar to Figure 5.1.

4. Inside the Interval and Subdivisions frame, place two textboxes, one slider, and four labels. (EditText boxes, a scrollbar, and static textboxes). (See Figure 5.1)

5. Set the default property values of the text boxes, slider value, and labels. (DP-5)

6. Repeat Step 4 and 5 for the other two frames. Label and set properties of the user input fields. At this point, the layout and design of the interface might look something like Figure 5.1.

7. Align all objects horizontally and vertically (DP-2, GP-3) (See Figure 4.4 and 4.6).

8. Add a pushbutton at the bottom of the window. The reader may customize the MATLET as desired.
5.2.2 Adding Code to the Callbacks

Open the associated M-File and locate the object Callbacks. In this example, the only code that needs added is to the ‘Calculate’ pushbutton. When the pushbutton object was created, MATLAB automatically generated the following five lines of code, four of which are comment lines. The code segment is a sub-function of the main .m function file.

```matlab
% --- Executes on button press in Calculate.
function Calculate_Callback(hObject, eventdata, handles)
% hObject    handle to Calculate (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles
```

To add functionality, the following eleven lines were added to the Callback function of the ‘Calculate’ button. The first line assigns the variable a to the value of the Edit Text field TAG-ed ‘a’. Notice that its ‘value’ must be converted to a number. Similarly, b and n are assigned. The string value f is the user input function. To evaluate f at any value, use the `eval ( string )` command. The end-result of the MATLET is the numeric approximation of
Left Sums of any continuous function on the interval \([a, b]\) to arbitrary accuracy\(^9\), by displaying the sum in the String property of the text field with the TAG ‘Left_Sum’.

```matlab
a= str2double(get(handles.a, 'String'));
b= str2double(get(handles.b, 'String'));
n= str2double(get(handles.n, 'String'));
f= get(handles.f, 'String');
sum= 0;
step= (b-a)/n;
for i= 1:n
    x= a+ i*step;
    sum= sum + (eval(f))*step;
end
set(handles.Left_Sum, 'String', sum);
```

### 5.3 Example 2: Symbolic Differentiation and Integration

**Objective**
To use MATLAB’s symbolic toolkit to compute the derivative or indefinite integral of a user defined function of one-variable and to graph both the input and the output functions.

**Input**
A one-variable function of \(x\).

**Output**
The derivative or integral and graphs of (both functions)

**Reader’s Objective**
To gain familiarity and experience using symbolic toolkit, plotting to axes, coding events for pushbuttons, and converting input using MATLAB build-in commands.

### 5.3.1 Layout and Design

1. Open a blank figure window.
2. Add Frames, text fields, and pushbuttons as seen in Figure 5.2.
3. Design from big to small. Create the outer most objects, such as frames, first.
4. Use the principles of Chapter 2; make use of the Alignment tool.

---

\(^9\) Limited to the MATLAB program, machine epsilon = 2.2204e-16
5.3.2 **Adding Code**

1. Open the associated M-File.
2. Add code to the Callback functions: Differentiate, Integrate, and Reset. See Appendix B for complete code of callbacks.

5.3.3 **Executing the M-File / Running the MATLET**

1. Press the green error at the top of the GUIDE window. (Figure 4.4)
2. Experiment with the many possible inputs and options. (Figure 5.3).

![Diagram of Function, Select, and Output boxes with buttons for Differentiate, Integrate, and Reset, and two graphs labeled graph1 and graph2.]

Figure 5.2 – Layout of Example 2
Figure 5.3 – MATLET Window from Example 2
LIST OF REFERENCES


## GLOSSARY OF TERMS

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Back-End</td>
<td>The program instruction code that gives the application functionality</td>
</tr>
<tr>
<td>Callback Function</td>
<td>Callback, for short, is a method (or function) of an object. Each graphical object has a callback sub-function in the M-File.</td>
</tr>
<tr>
<td>Debugging</td>
<td>Removing the bugs from the program</td>
</tr>
<tr>
<td>Front-End</td>
<td>The part of the software that the user interacts with, the GUI</td>
</tr>
<tr>
<td>Graphical User Interface</td>
<td>Point of contact between the human and computer. Means of communication. Usually comprised of metaphorical objects such as Pushbuttons, slider bars, and Checkboxes. Pronounced: “gooey”</td>
</tr>
<tr>
<td>GUI</td>
<td>High-level programming tool for creating the graphical objects that compose a GUI</td>
</tr>
<tr>
<td>Handle</td>
<td>A unique identifier of a graphics object. Uniqueness is ensured by assigning numeric values to these elements.</td>
</tr>
<tr>
<td>Handles</td>
<td>For characteristics of an object, see “Properties” below. For unique object identifier, see “Handle” above.</td>
</tr>
<tr>
<td>Term</td>
<td>Definition</td>
</tr>
<tr>
<td>-----------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>High-Level Programming</td>
<td>Programming with tools that makes development easier. Masks the abstract syntax from the developer. Done though an Integrated Developing Environment</td>
</tr>
<tr>
<td>Interactive Applet</td>
<td>Small application file designed with a GUI that a user can dynamically change parameters an expect seemingly instantaneous results</td>
</tr>
<tr>
<td>Low-Level Programming</td>
<td>Hard coding, close to machine language</td>
</tr>
<tr>
<td>MATLET</td>
<td>Author-coined term for interactive applet written with MATLAB</td>
</tr>
<tr>
<td>M-File</td>
<td>MATLAB program function file. Called M-File for the dot m (.m) file extension.</td>
</tr>
<tr>
<td>Minimalist Learning</td>
<td>John Carroll's Minimalist Learning theory applies to users of technology and software applications. The basic premise is for a new user to learn the application with minimal learning difficulties. This is achieved by error prevention, providing immediate feedback, and investment of time on meaningful tasks.</td>
</tr>
<tr>
<td>Term</td>
<td>Definition</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Object-Oriented Programming (OOP)</td>
<td>Forms a mapping between real world objects and software counterparts. Exploits the parent-child paradigm taking advantage of inheritance relationships where new classes inherit characteristics of an existing class; yet also possess unique characteristics of their own. (p.45), [5]</td>
</tr>
<tr>
<td>Power User</td>
<td>Technical professional. (Computer “Geek”)</td>
</tr>
<tr>
<td>Program Bug</td>
<td>An error in the program producing unexpected results or inability to compile and run. There are three types of errors: compiling errors, logical errors, and run-time errors.</td>
</tr>
<tr>
<td>Properties (handles)</td>
<td>Usually called instance variables in OOP. Characteristics of an object that can change during run-time. Properties correspond to the state of an object.</td>
</tr>
<tr>
<td>Screen-Shot</td>
<td>A picture of the computer screen used to illustrate program specific information.</td>
</tr>
<tr>
<td>User-Center Design</td>
<td>Designing systems with the user in mind. Make software easy to learn and provide comfortable hardware. Use mental models and metaphors to make familiar to the user.</td>
</tr>
<tr>
<td>WYSIWYG</td>
<td>What You See Is What You Get – pronounced “wiz-ewig.” No hidden features in the applications, not using abstract commands, no significant difference between expectations and results. Using metaphors, being able to point-click, and providing immediate feedback. (Apple)</td>
</tr>
</tbody>
</table>
### APPENDIX A

**MATLAB SYMBOLIC TOOLBOX COMMANDS**

<table>
<thead>
<tr>
<th>Calculus</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Diff</td>
<td>Differentiate</td>
</tr>
<tr>
<td>Int</td>
<td>Integrate</td>
</tr>
<tr>
<td>Limit</td>
<td>Limit</td>
</tr>
<tr>
<td>Taylor</td>
<td>Taylor series</td>
</tr>
<tr>
<td>Jacobian</td>
<td>Jacobian matrix</td>
</tr>
<tr>
<td>Symsum</td>
<td>Summation of series</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Simplification</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simplify</td>
<td>Simplify</td>
</tr>
<tr>
<td>Expand</td>
<td>Expand</td>
</tr>
<tr>
<td>Factor</td>
<td>Factor</td>
</tr>
<tr>
<td>Subs</td>
<td>Symbolic substitution</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Solution of Equations</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Solve</td>
<td>Symbolic solution of algebraic equations</td>
</tr>
<tr>
<td>Dsolve</td>
<td>Symbolic solution of differential equations</td>
</tr>
<tr>
<td>Finverse</td>
<td>Functional inverse</td>
</tr>
<tr>
<td>Compose</td>
<td>Functional composition</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Integral Transforms</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fourier</td>
<td>Fourier transform</td>
</tr>
<tr>
<td>Laplace</td>
<td>Laplace transform</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Conversions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Double</td>
<td>Convert symbolic matrix to double</td>
</tr>
<tr>
<td>poly2sym</td>
<td>Coefficient vector to symbolic polynomial</td>
</tr>
<tr>
<td>sym2poly</td>
<td>Symbolic polynomial to coefficient vector</td>
</tr>
<tr>
<td>char</td>
<td>Convert sym object to string</td>
</tr>
<tr>
<td>sym</td>
<td>Convert string to a symbolic object</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>String handling utilities</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vectorize</td>
<td>Converts an expression for calculation, adds a '.' before a '/'</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Pedagogical and Graphical</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rsnums</td>
<td>Riemann sums</td>
</tr>
<tr>
<td>Explot</td>
<td>Easy to use function, implicit, and parametric curve plotter</td>
</tr>
<tr>
<td>Expolar</td>
<td>Easy to use polar coordinates plotter</td>
</tr>
<tr>
<td>Ezsurf</td>
<td>Easy to use surface plotter</td>
</tr>
<tr>
<td>Funtool</td>
<td>Function calculator</td>
</tr>
<tr>
<td>Tylortool</td>
<td>Taylor series calculator</td>
</tr>
</tbody>
</table>

35
APPENDIX B

EXAMPLE 2 – M-FILE CALLBACKS

Code Segment # - Callbacks for Example 2

% --- Executes on button press of Differentiate button.
function diff_Callback(hObject, eventdata, handles)

% 1. Take the derivative of user defined function (String)
x=sym('x'); % Declares x as a symbolic object
f=get(handles.f_in, 'String'); % Read user input function
F=diff(f); % Symbolic differentiation of f
outString=char(F); % converts the derivative to a string
outPlot=vectorize(outString); % convert string to use eval
set(handles.f_out, 'String', outString); % Displays output
clear x; % clears x to use later as a number

% 2. Plot user input function
h=findobj('Tag','graph1'); % gets handle of graph 1
axes(h); % makes graph 1 current
x=-4:.1:4; % vector of x values
f=vectorize(f); % adds . to '/' and '\'
y=eval(f); % evaluates the user function (string) f at all x values
plot(x,y); % graphs x versus y

% 3. Plot calculated function
h=findobj('Tag','graph2'); % gets handle of graph 2
axes(h); % makes graph 2 current
y=eval(outPlot); % evaluates derivative for all x values
plot(x,y); % graphs derivative function
END FUNCTION

% --- Executes on button press in Int.
function Int_Callback(hObject, eventdata, handles)

% 1. Symbolic Integration
x=sym('x'); % Declares x as a symbolic object
f=get(handles.f_in, 'String'); % Read user input function
F=int(f); % Indefinite integral of f
outString=char(F); % converts the symbolic derivative to a string
outPlot=vectorize(outString); % convert string to be able to use eval
set(handles.f_out, 'String', outString);
clear x; % clears x to use as number

% 2. Plot user input function
h=findobj('Tag','graph1'); % gets handle of graph 1
axes(h); % makes graph 1 current
x=-4:.1:4;
f=vectorize(f);
y=eval(f);
plot(x,y);

% 3. Plot calculated function
h=findobj('Tag','graph2'); % gets handle of graph 2
axes(h); % makes graph 2 current
y=eval(outPlot);
plot(x,y); % graphs output function
% END FUNCTION

% --- Executes on button press in reset.
function reset_Callback(hObject, eventdata, handles)

% 1. Clear input and output
set(handles.f_in, 'String',''); % clears edit-field, input
set(handles.f_out, 'String',''); % clear static text output

% 2. Reset graph 1
h=findobj('Tag','graph1');
cla(h);

% 3. Reset graph 2
h=findobj('Tag','graph2');
axes(h);
cla;