

The MATLAB® Programming Language

Basic Programming using the MATLAB Environment

Kristopher Reese

ABSTRACT

This paper covers the Basics of programming in MATLAB and discusses how to use the included Programming environment. Topics included in the Paper include the history of MATLAB and Basic Features of MATLAB (Programming Paradigm, Data Types, Control Flow, Object-Orientation, and functions). Following these sections, a tutorial section is included which will discuss

Table of Contents

1. Introduction	3
2. History of MATLAB	3
3. Basic Features of MATLAB.....	5
3.1 MATLAB Programming Paradigm.....	5
3.2 Data Types.....	5
3.3 Control Flow	7
3.5 Subprograms (Functions).....	9
3.6 Object-Orientation	11
4. Tutorial.....	12
4.1 Hello World	12
4.2 Complex Mathematics	13
4.3 Plotting.....	15
5. Evaluation	20
Appendix: Further Learning and Resources.....	22
Works Cited.....	23

1. Introduction

MATLAB is a mathematical scripting language that looks very similar to C++. It has many features that are implemented into the language. These features include: efficient matrix and vector computations, easy creation of scientific and engineering graphics, application development (including GUI building), Object-Oriented programming, File I/O functions, and string processing. Because MATLAB is a mathematical scripting language the software is primarily used in scientific and engineering fields, and is mostly used for producing solutions to complex systems of equations; modeling, simulation, and prototyping; and Data Analysis, exploration, and visualization.

This survey will attempt to discuss all of the features that are found in the MATLAB programming language. The survey will discuss how MATLAB handles various Data Types, Control Flow that is found in MATLAB, Object-Oriented features that MATLAB handles, File I/O, as well as plotting.

MATLAB was chosen for this project because of its capabilities to simulate many aspects of computing including, but not limited to, Networking. My current research interests include Wireless Sensor Networks and because of MATLAB's ability to handle simulations and gather data, I will be using MATLAB. This survey will be used as a learning tool to help me learn the basic of simulation and data analysis in MATLAB.

This paper is broken down into four sections. The first section will discuss the history of the language starting from when it began to the current version of MATLAB as well as some possible future developments. This section will also discuss the lifetime of MATLAB. In section 2, the basic features of MATLAB will be discussed. Section 3 will act as a tutorial for the language. A Hello World program will be created in this section. Two or Three other features will be shown in other sample programs. Section 4 will evaluate the effectiveness of the MATLAB Environment and the Language.

2. History of MATLAB

MATLAB was created by Cleve Moler and has a diverse history. Three men greatly influenced the origins of what is now MATLAB. These men were John Todd, George Forsythe, and J.H. Wilkinson. During Moler's undergrad years at Caltech where he began to work with John Todd on projects involving Hilbert Matrices. Moler began to write programs used to solve these Hilbert matrices using an absolute numeric machine language.

After receiving a bachelor degree at Caltech, Moler went to Stanford to begin working with George Forsythe, a friend of his previous teacher John Todd. In 1962, after taking a numerical analysis class with Forsythe, Moler wrote a FORTRAN program to solve systems of simultaneous linear equations. These were distributed widely during the time.

Under the direction of Forsythe, Moler wrote his PhD dissertation entitled “Finite Difference Methods for the Eigenvalues of Laplace’s Operator” in 1965. The motivating example through this dissertation was the L-shaped membrane, which has now become the MathWork’s logo.

Following the publication of his dissertation, Forsythe and Moler published a textbook about matrix computation in 1967. This textbook has been listed by the ACM (Association for Computing Machinery) as an important early text in computer science because it contained working programs in Algol, Fortran, and PL/I for solving systems of simultaneous linear equations.

In the late 1960s, Wilkinson and his colleagues, including Moler, published various papers that included multiple algorithms in Algol for various aspects of matrix computation. These algorithms were later translated into Fortran by the Argonne National Laboratory to produce EISPACK. This was soon followed by LINPACK, which is a package of Fortran programs used for solving linear equations.

During the production of EISPACK and LINPACK, Moler was also teaching at the University of New Mexico, teaching various classes in numerical analysis and matrix theory. Moler soon wanted to allow his students to have the ability of the new packages without forcing a student to know how to write Fortran programs.

In the late 1970s, Moler used Fortran and portions of LINPACK and EISPACK to develop the first version of MATLAB. The first version of MATLAB only included 80 functions and there was no support for M-files or toolboxes. The graphics of the original MATLAB was also very primitive, and the first plots were made by printing asterisks on the teletypes or typewriters serving as terminals for the program.

In 1979, Moler moved to Stanford University where he taught the graduate numerical analysis course. Though many of the mathematics and computer science students taking the course were not impressed with the program, many of the engineering students taking the course found MATLAB invaluable to them due to its emphasis on matrices. These engineering students soon showed the program to Jack Little, a professor of engineering at Stanford.

In 1981, after IBM announced the first PC, Little and Steve Bangert translated to the C language and many of the features that are enjoyed today were included; adding M-Files toolboxes, and more powerful graphics. In 1984, Little, Bangert, and Moler founded The MathWorks in California.¹

Since 1984, MATLAB has gone through seven versions and is currently in version 7.5 (R2007b). Through its evolution, many features have been added or changed, including functions for mathematical computations, the programming language, and the graphics among many other features.

MATLAB has gained a solid footing in the Academic and industrial settings, especially when complex mathematics is typically involved in the research

¹ Moler, Clove. (2004). “The Origins of Matlab.” *The MathWorks, Inc.* Retrieved November 12, 2007, from <http://thewritesource.com/apa/apa.htm>.

beingdone.² MATLAB is well suited for these areas of research because of its ability to quickly do most forms of complex mathematics.

3. Basic Features of MATLAB

This section will discuss many of the basic features for programming in MATLAB. The first subsection will discuss the programming paradigm that is used by the MATLAB programming language. This is followed by subsections on the Data Types found in MATLAB, the control flow, as well as a brief description of the Object Oriented features of the language and various other features found in the language.

3.1 MATLAB Programming Paradigm

The Programming Paradigm for the MATLAB programming language is actually a combination of paradigms. MATLAB supports Object-Oriented paradigms and Functional paradigms. Though MATLAB does support Object Oriented paradigms, it is rare to find MATLAB programs written in the Object Oriented paradigm. The functional paradigm is the most frequently used paradigm of the two when full programs are written.

3.2 Data Types³

MATLAB supports many of the data types that are found in many other programs including integers, floating point data types, logical types, and strings. However, it also supports some numerical data types that are not, or rarely, found in other programming languages included complex numbers, Infinity and NaN.

Unlike other programming languages, MATLAB handles all of its data types differently. MATLAB is not a typed language and therefore focuses primarily on matrices. It is because of this that MATLAB is much faster at mathematical computations than other programming languages. When inputting variables, there are various ways to input data. When inputting one number in the matrix, you would write code as such:

```
x = 3 %Storing a variable in a 1x1 matrix
```

and, though it does not appear that this is stored in a matrix, it is stored as a one by one matrix in MATLAB. To store multiple variables in MATLAB your code would look like the below code.

² Goering, Richard. (2004 October). "Matlab edges closer to electronic design automation world." *EE Times*. Retrieved November 12, 2007, from <http://www.eetimes.com/news/design/showArticle.jhtml?articleID=49400392>

³ (1984-2007) "MATLAB Documentation." *The Mathworks*. Retrieved on Nov. 25, 2007 from http://www.mathworks.com/access/helpdesk/help/techdoc/index.html?/access/helpdesk/help/techdoc/matlab_prog/f2-43934.html&http://www.google.com/search?q=matlab+data+types&ie=utf-8&oe=utf-8&aq=t&rls=org.mozilla:en-US:official&client=firefox-a

```
x = [3, 4, 5; 6, 7, 8] %storing variables in a 2x3 matrix.
    Commas are used to separate row values. Semi-colons begin a
    new row in the matrix.
```

MATLAB's integer types are different than some other languages. MATLAB supports signed and unsigned integers. Unlike many other languages, you must declare the bit value of the integer you will be using. Because of this, MATLAB supports 4 signed and 4 unsigned types. These types are:

Data Type	Range of Values	Conversion Function
Signed 8-bit integer	-2^7 to 2^7-1	int8
Signed 16-bit integer	-2^{15} to $2^{15}-1$	int16
Signed 32-bit integer	-2^{31} to $2^{31}-1$	int32
Signed 64-bit integer	-2^{63} to $2^{63}-1$	int64
Unsigned 8-bit integer	0 to 2^8-1	uint8
Unsigned 16-bit integer	0 to $2^{16}-1$	uint16
Unsigned 32-bit integer	0 to $2^{32}-1$	uint32
Unsigned 64-bit integer	0 to $2^{64}-1$	uint64

To declare a value as an integer, you would type something as such:

```
x = int8(127)
```

By declaring an integer as shown above, MATLAB will determine the best integer type. However, the conversion functions will override the integer type that was selected and store the integer in the new data type. The conversion functions will also convert other data types into the integer data type selected.

Floating-point data types are treated as many other languages treat floating-point numbers. Double precision floating-point numbers are the default numerical data types that are used by MATLAB. This makes typing any double precision number easy to type, however in order to use single precision floating-point numbers, the single conversion function must be used. The first example below shows how to declare a double precision and the second shows a single precision floating point variable.

```
x = 24.578; %Double Precision Floating Point
x = single(24.578) %Single Precision Floating Point
```

Unlike other languages, MATLAB will do computations with complex numbers and infinity. Because of this, several new data types are needed in order to work with these new variables. Though NaN is typically found in other languages, it is a combination of numbers that are either imaginary or infinite numbers in these languages. MATLAB treats NaN differently; NaN is only given when a result is not a real, imaginary, or infinite number. Infinite numbers are also given their own variable. This number is more often used as a result. This is often displayed as "Inf" in the answer portion of the equation.

Complex numbers, the result of imaginary numbers, are formed with one of two letters, *i* or *j*. Both of these letters represent the square root of -1. To declare

a variable to be a complex number, you can declare a complex number by writing the number with the real number added to the imaginary number. An example is given below.

```
x = 2 + 3i %Complex number
```

MATLAB also holds Logical variables used for Boolean Logic. Logical types are treated in very much the same way as Boolean types in other languages. These are achieved through typing true and false in a matrix. Because of this, MATLAB will quickly do truth tables by using several of its built-in functions. Strings are also treated in much the same way as strings and character arrays in other languages. However instead of storing characters in arrays, they are stored in matrices. A variable holding the string "Hello World" would be a one by eleven matrix because of the eleven characters found in the string.

These are the data types that are found in MATLAB. Though many of these data types exist, it is rare that data types will need to be converted. MATLAB chooses the best form of data type for storing the information in the matrix. Between character arrays, logical types, and floating point numbers.

3.3 Control Flow⁴

The Control Flow found in MATLAB is similar to other programming languages. Four types of Control Flow are found in MATLAB. These include IF-ELSE statements, Switch statements, FOR loops, and WHILE loops.

IF-ELSE statements are treated in a similar way as IF-ELSE statements in other programming languages. However, at the end of an IF statement, the keyword END must be used in order to end the statement. An IF statement must also have a condition in each of the IFs. The Example below calculates the letter grade by converting the percentage grade.

```
avg = 76;
if avg >= 90
    Grade = 'A';
end
if (avg >= 80) & (avg < 90)
    Grade = 'B';
end
if (avg >= 70) & (avg < 80)
    Grade = 'C';
end
if (avg >= 60) & (avg < 70)
    Grade = 'D';
end
if (avg < 60)
    Grade = 'F';
end
```

⁴ Hertiner, Marc E. (2001) Programming in MATLAB®. Australia: Brookes/Cole.

This form of IF statement is very long and we would want to cut down the amount of code used in the program. Because of this, we can use ELSEIF statements rather than creating new IF statements for every condition. The below code gives an example of how the ELSEIF statements are used:

```
avg = 76;
if s >= 90
    Grade = 'A';
elseif s >= 80
    Grade = 'B';
elseif s >= 70
    Grade = 'C';
elseif s >= 60
    Grade = 'D';
else
    Grade = 'F';
end
```

These are how the IF-ELSE statements work in MATLAB.

In most languages, excessive use of the IF-ELSE statements becomes tedious and bogs the program in many computations. In this case, SWITCH statements are typically used. MATLAB allows the use of SWITCH statements like these other languages. The below code shows how the grade program can be taken from IF-ELSE statements and made into SWITCH statements:

```
avg = 76;
switch avg
    case {100, 99, 98, 97, 96, 94, 93, 92, 91, 90}
        fprintf('A\n'); %fprintf prints to the command line, \n starts
                        new line
    case {89, 88, 87, 86, 85, 84, 83, 82, 81, 80}
        fprintf('B\n');
    case {79, 78, 77, 76, 75, 74, 73, 72, 71, 70}
        fprintf('C\n');
    case {69, 68, 67, 66, 65, 64, 63, 62, 61, 60}
        fprintf('D\n');
    otherwise
        fprintf('F\n');
end
```

As you can see, the SWITCH statement requires the use of a number array in order to find the value for `avg`. This is because the switch statement in MATLAB will only search for equivalent numbers. There is no way to set up the cases to compute anything other than equivalents. When typing in the array, you must separate each index with a comma and if the index is text, you must put the text inside of single-quotation marks. The array is unnecessary when only check for equivalency to one index, but is necessary when using SWITCH statements that check for one of several conditions.

MATLAB also supports two forms of loops, the FOR and the WHILE loops. Though FOR loops are treated the same as in most programming languages, the

syntax for the loop is different than these same languages. When declaring a variable, you must include the starting value, the ending value, and an incremental value may be optionally added. If no incremental value is used, MATLAB will assume that the incremental value is one. These values are separate by colons and the order that they must be placed is: starting value : incremental value : ending value. The FOR loop below will print out the value of each value that is reached in the loop:

```
for i = 1:0.5:5
    disp(i); %disp is another way to output to command line
end

% Loop will Print 1, 1.5000, 2, 2.5000, 3, 3.5000, 4, 4.5000, 5
```

We can see that the loop will print every value starting from 1 adding 0.5 every time the loop is called again, until the value reaches 5. If the loop would have read “i = 1 : 5”, the loop would only print the values: 1, 2, 3, 4, and 5 because the loop will increment by 1.

The WHILE loop is treated as it is in most languages and the syntax is similar to these other languages. The WHILE loop below will do a similar task to the FOR loop above, printing numbers between and including five:

```
i = 1;
while i <= 5
    disp(i);
    i = i + 1; % increment value by 1
end

% Will print the values: 1, 2, 3, 4, and 5
```

As can be seen from this code, the while loop is started with the WHILE keyword and is followed by a condition statement to keep the WHILE loop from ending. These are the various basic control flow statements that are included in the MATLAB environment.

3.5 Subprograms (Functions)⁵

MATLAB has many functions built into the language to handle mathematics, File Input/Output, graphical displays of data, and much more. One of the most frequently used of the functions is the `plot` or `ezplot` function. The `plot` function itself is relatively easy to you. You simply type in the mathematical equation for the plot and the y value range. Then use the `plot(x, y)` or `ezplot(x, y)` to plot the equation. The below code gives an example of how the `plot` function can be achieved:

⁵ Ibid

```
syms x; % x is now a symbol
x = sin(x);
y = [-3, 3];
ezplot(x, y)
```

In this example we use the `ezplot` function because we are declaring `x` to be a symbol. If we were to use the `plot` function instead we would get an error because `plot` requires that `x` be a value as well as the `y` values. The `plot` function, and other functions relating to plotting, is discussed in more detail in section 4 of this paper.

Though MATLAB has many built-in functions, it will also allow a user to create their own functions. This allows for more dynamic programming. In the grade program above, we had simply written the test score in the program. With the use of functions, we can allow a user to type a test score when calling the function.

To start a function, you would simply use the keyword `FUNCTION` followed by the function name. An equal sign is used to tell the function how it is called. You can also reference a value in the function when calling it, as in most other languages. The below code is an extension to our Grade program and allows a user to pass a value into the function:

```
function Grade = score1(s) % variable s is a value that is passed
                           into the function
if s >= 90
    Grade = 'A';
elseif s >= 80
    Grade = 'B';
elseif s >= 70
    Grade = 'C';
elseif s >= 60
    Grade = 'D';
else
    Grade = 'F';
end

% On the command line, type score1(76) to receive the same answer
% as the first few grade programs.
```

As can be seen, the function is named `Grade` and is called by typing `score1(#)` to solve the function. Functions are the primary mode of creating dynamic programs in MATLAB due to the primary paradigm of the language.

3.6 Object-Orientation⁶

Though MATLAB is primarily a functional programming language, Object Oriented features have been added to language and the language does allow the Object Oriented paradigm to be used. Some of the Object Oriented features of MATLAB are creation of classes and Operator Overloading. Many of the other features found in other Object-Oriented programming languages are not found in MATLAB. Creating objects in MATLAB are much more complex than in a language such as C++. This section will compare various features in MATLAB Objects to C++ objects.

Declaring an object, or a class, is not done through actual code in MATLAB as it is in C++. Rather than typing “`class name`”, declaring a class is done by creating a subdirectory beginning with the @ symbol followed by the name of the object. For example, if I wanted to create a score class, I would find the MATLAB directory and create a new sub-directory called “@score”. Creating a constructor in MATLAB is done by simply creating a file with the object name in the directory created for the object. The chart in figure 1 compares how to create an object and constructor in MATLAB with its counterpart in C++.

Member Data and Member methods are only treated privately in MATLAB. In C++, one has the option to declare a method or allow its data to be public. Because of this member data is only accessible by member function, functions found within the object. If a method is not located within the object, member data will not be accessible.

Overloading Operators is also possible in MATLAB. In MATLAB, the symbolic operators are represented by their respective names. This makes Operator overloading easy because you can simply create a M-file with the name of the operator in the class. For example to overload “+”, you would create a M-file called plus.m and include the “`function out = plus(eq1, eq2)`”. When you use “+” in the object, it will access the new plus.m file and treat the symbol the way you have overloaded it.

Component	MATLAB	C++
Declaration	Windows: C:\MATLAB\@score Mac/Unix: \MATLAB\@score	<code>class score</code> { ... }
Constructor	Windows: C:\MATLAB\@score\score.m Mac/Unix: \MATLAB\@score\score.m	<code>public:</code> score();
Operator overloading	File: \Matlab\@simp_eq\plus.m Contains: <code>function out=plus(eq1, eq2)</code>	<code>public:</code> simp_eqoperator+(simp_eq&);

Figure 1: A comparison of Object creation and operator overloading in MATLAB and its C++ counterparts.

⁶ (August 1997) “MATLAB Programming: Object Oriented Programming Example.” *University of Michigan, College of Eng, and Computer Science*. Retrieved on Nov. 25, 2007 from <http://www.engin.umd.umich.edu/CIS/course.des/cis400/matlab/oop.html>

4. Tutorial

This section will discuss some of the more advanced topics in MATLAB. The first subsection will discuss several ways to create a “Hello World” program using MATLAB’s programming environment. Following this, complex integrals will be solved using MATLAB. Then more plotting functions and techniques will be discussed in the next subsection. The final subsection...

4.1 Hello World

There are several different ways to create a Hello World program in the MATLAB environment. The first way to write a Hello World is to use the command line. The other way is to create a M-file and run the file inside of the command line.

There are many ways to output HELLO WORLD in the command line. To use the command line, you would simply type the MATLAB code directly into the given Command Window inside of the environment that comes with MATLAB. There are three ways to create the HELLO WORLD program. The first two will print the program inside of the command window while the third creates a GUI.

The first two use the `disp()` function and the `fprintf()` function. Both of these functions act in exactly the same way and output in the command line. To use either of the functions, you simply add the text, in quotes, inside of the function or a variable, not inside quotes. The below code is an example of how to use these functions:

```
>>disp('HELLO WORLD')
HELLO WORLD
>>fprintf('HELLO WORLD\n')
HELLO WORLD
```

The “>>” represents a line in the command window. These functions were typed directly into the command window and the output is displayed below the function. The “\n” in the `fprintf()` function is needed to create a newline after outputting the text.

The third way creates a GUI and can also be done through the command line. In this case, we will use the `msgbox()` function. Simply typing the text in the function, as above, will output the message inside of a message box. We can also add a title to the box by adding new text after the message. The below code is an example of how to use the `msgbox()` function (Figure 2 is an image of the message box that is created).

```
>>msgbox('HELLO WORLD', 'Hello')
```

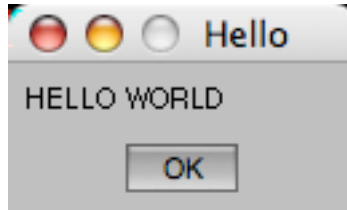


Figure 2 The Output Display of the `msgbox()` function in MATLAB

The first portion of the text that is being passed into the function is the message that is displayed in the message box; the text following the message is the title of the message box.

Using the Command line is nice when you are simply computing mathematics and treating MATLAB as a calculator. When large programs are being created, creating an M-file is going to be the easiest way to create dynamic programs. Creation of the M-files can be done in any text editor. MATLAB is packaged with its own text editor as well. To access MATLAB's text editor, you would type "`edit filename.m`" in the command window.

From here you can create the function or type any code that you would normally type in the command window. The functions that were stated above will work the same way in a M-file as they would when typing the code in the command window, so no other code needs to be learned for the M-files.

4.2 Complex Mathematics⁷

In many languages, solving complex mathematics would require the programmer to develop their own functions to solve these equations. This can be done; however, MATLAB has these functions built into the language that allows the developers to do mathematical equations without having to create these functions. In this portion of the tutorial we will discuss how to use some of these functions to solve Initial Value Problems (IVP) and Integrals.

MATLAB supports two functions to solve IVPs, the `ode23` and the `ode45` functions. In both cases, you will get approximately the same answer using either of the equations. Using the `ode45` function will approximate more points giving a more accurate answer.

Both functions use the RungeKutta method for solving IVPs. The numbers in the functions represent the order method used in the function. For example, the `ode23` function uses the second- and third- order RungeKutta method. The `ode45` function uses the fourth- and fifth- order RungeKutta method for the solution.

The code below creates an inline function for the differential equation $y'(t) = ty^2$ where the initial condition is $y(-2) = 5$. Figures 3a and 3b are the outputs for the `ode45` and `ode23` functions respectively.

⁷ **Concepts and function information for all functions in this section taken from:** (1984-2007) "MATLAB Documentation." *The Mathworks*. Retrieved on Nov. 25, 2007 from <http://www.mathworks.com/access/helpdesk/help/techdoc/index.html>

```
g = inline('t*y^2', 't', 'y');
ode45(g, [-2, 2], 5); % switch ode45 to ode23 to use the other
                      function
```

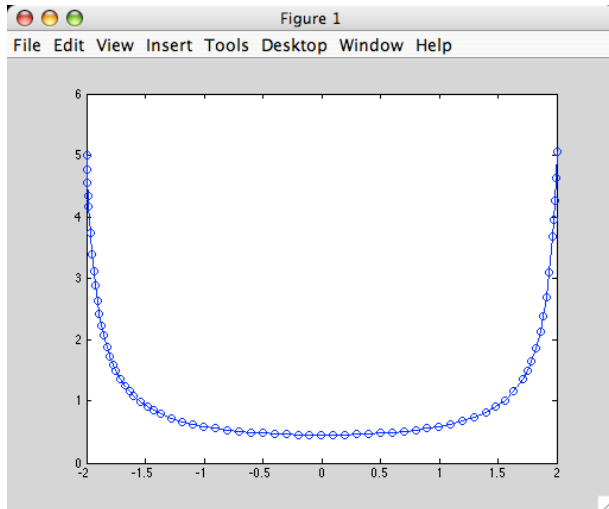


Figure 3a The output for the ode45 function in MATLAB.

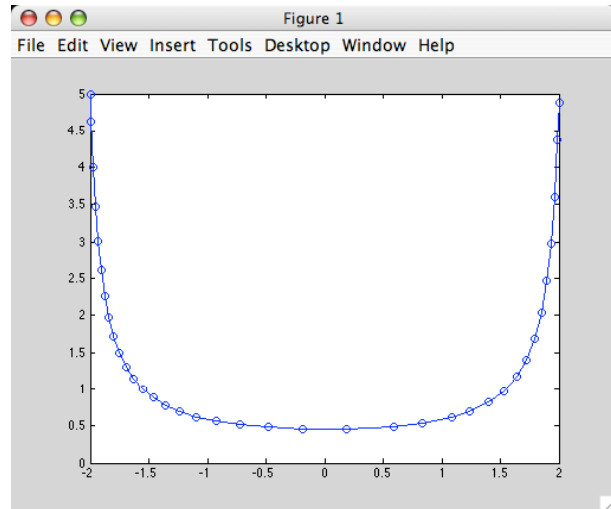


Figure 3b The output for the ode23 function in MATLAB

As can be seen in Figure 3, the `ode45` function outputs more points on the graph (Figure 3a) than the `ode23` function graph (Figure 3b). For this reason the `ode45` function is much more accurate than the `ode23` graph, though the shape is approximately the same in both. The `ode45` function takes more time to compute than the `ode23` function and this may make the `ode23` function more useful when the equation is much larger than the one presented in this tutorial.

Integration is also a challenge in most programming languages and would require a large amount of work to create a function to solve integrals. MATLAB has functions for solving integrals numerically, the `quad` function, and symbolically, the `int` function. The `quad` function passes in three values. The first value will be the equation, placed in quotes. The next two values being passed into the equation are the interval at which the function is solving. The below code solves the integral

$\int_0^5 3\cos(x+2)$ numerically.

```
quad('3*cos(x+2)', 0, 5)

%output:
%ans =
%
%   -0.7569
```

The `int` function passes in four values to the functions. The first value is again the equation in quotes. Unlike the `quad` function, the `int` function may have

numbers or other letters, representing constants. The next value being passed is the variable for which the symbolic solution is being solved. The final two values represent the interval at which the equation is being solved. The below code represents an integral in which A and B represent constants and x is the variable

being solved for between [x1, x2], $\int_{x1}^{x2} A \cos(x + B)$.

```
int('a*cos(x+b)', 'x', 'x1', 'x2')

%output:
%ans =
%
%   -a*sin(x1+b)+a*sin(x2+b)
```

These functions are only four of the many mathematical functions found in MATLAB. MATLAB handles many types of mathematical computations from basic arithmetic to the most advanced mathematics. MATLAB can also be used as a simple calculator by using the command line to type mathematical equations. A list of resources to more books on the mathematic functionality of can be found in the Appendix of this Paper.

4.3 Plotting⁸

Plotting is one of the most powerful features that MATLAB has over other languages. MATLAB allows for various 2-dimensional types of plots to be used, including Log-Log plots, Semi-Log plots, and Polar plots. We briefly discussed in a previous section how to use the `ezplot` function. In this portion of the Tutorial, we will discuss how to create other forms of graphs and manipulating these graphs by adding labels and other things.

We will use three equations in this tutorial, $y_1 = e^x$, $y_2 = e^x + \sin(x)$, and $y_3 = 2x^2 + 5x + 1$; x will be computed from the range of [-5, 5]. We will first discuss the `ezplot` function again. `Ezplot` inputs two variables into the function, the first is the equation being plotted, the second is the range for the plot. In order to plot various graphs at the same time, the keywords “hold on” must be used to prevent the plot from disappearing when plotting the new graph. The below code is an example of how to use the `ezplot` function, Figure 4 shows the output of the code.

```
% plot1.m
hold on;
ezplot('exp(x)', [-5, 5]);
ezplot('sin(x)', [-5, 5]);
ezplot('2*x^2 + 5*x + 1', [-5, 5]);
hold off;
```

⁸ Plot function concepts and information obtained from:

Hertiner, Marc E. (2001) Programming in MATLAB®. Australia: Brookes/Cole.

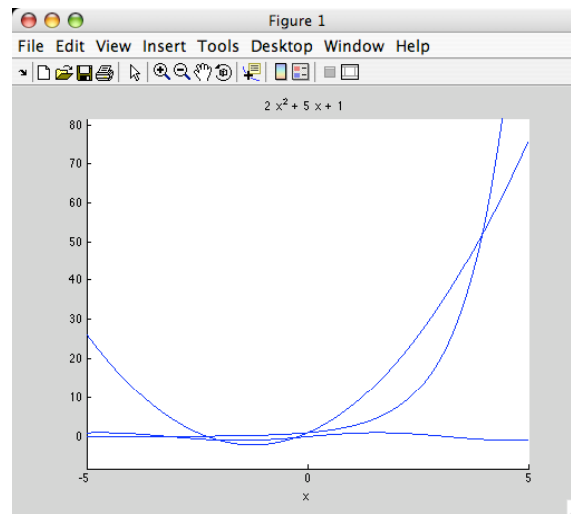


Figure 4 Output using the ezplot function

In some cases, `ezplot` is not the best suited for plotting. In this case, we may want to use the `plot` function. We can also get rid of excessive and recurring code that is found in the `ezplot` program. To use the `plot` function, we will have to declare that `x` will be a range from -5 to 5. Unlike before, we cannot just tell the function to plot between those points. If we do this, we will get linear lines for all of the functions. In order to use the `plot` function, we have to declare the range inside of the `linspace` function. This function creates linearly spaced vectors between the ranges. The equations can all be declared the same way inside of a new variable. The below code should output the same graphs as the `ezplot` program above.

```
% plot2.m
x = linspace(-5, 5);
y1 = exp(x);
y2 = sin(x);
y3 = 2.*x.^2 + 5.*x + 1;
plot(x, y1, x, y2, x, y3);
```

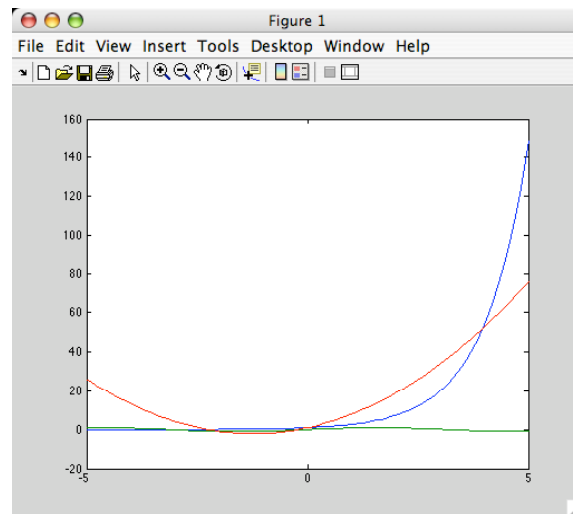


Figure 5 Output using the plot function.

You can notice that the plot function, when used to plot all of the functions at the same time, plots all of the functions with different colors. There are three other 2-dimensional plot types that MATLAB handles. To create any of these graphs, you would use their respective functions. For a semi-log graph along the x-axis, you will use the `semilogx` function, while along the y axis you would use the `semilogy` function; for a log-log plot, you would use the `loglog` function; a polar graph is created using the `polar` function. The code below will create three different graphs, the semi-log of x plot, the log-log plot, and the polar plot on the same workspace.

Creating two workspaces can be done using the `subplot` function. You pass in the number of rows you will be using first; this is followed by the number of graphs you would like to place for every row in the workspace. The number after this is the number of the graph that is being placed. The keywords “grid on” will toggle the grid on in the workspace.

```
% varPlot.m
x = logspace(0, 5, 100);
y1 = exp(x);
subplot(3, 1, 1); % 1 row, 3 graphs per row, graph 1
semilogx(x, y1);
grid on;
subplot(3, 1, 2); % graph 2
loglog(x, y1);
grid on;

theta = linspace(0, 8*pi, 200); % creates radians which is used
                                by polar graphs
r = 2*sin(theta);
subplot(3, 1, 3); % graph 3
polar(theta, r);
```

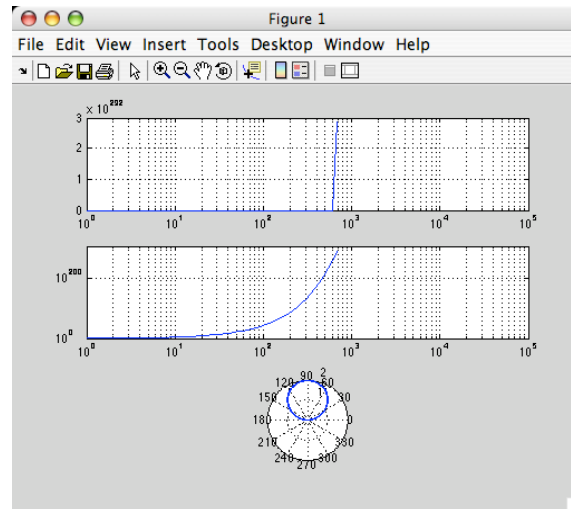


Figure 6 Examples: Semi-Log graph at the top, log-log in the middle, and polar graph on the bottom.

Now that we know how to create each of these graphs, we can change the line's color, line style, and add markers to the points on the graph. We can also learn to add a legend to determine which equation is represented by which line. We can add a title and labels to the graphs. To discuss this, we will return to and extend the `plot2.m` program. To add a title and labels to the graph, we will use these functions (which are really almost self-explanatory):

```
title('Various Equations'); % adds a title to the graph
xlabel('x axis'); % adds a x-label to the graph
ylabel('y axis'); % adds a y-label to the graph
```

After this, we can add a legend by using the `legend` function. When typing the name of each line, you must remember the order in which they were plotted. Since we have three lines, we will have three inputs in the function. We have to order the inputs in the order that they were plotted or they will not represent the correct equation. The below code is an example of how this would be written:

```
legend('e^x', 'sin(x)', '2x^2 + 5x + 1'); % creates a legend
```

Instead of using the colors that are automatically generated by MATLAB or using solid lines, let's change the line color, the line style, and add an X as a marker to $y_1 = e^x$, and a circle to $y_3 = 2x^2 + 5x + 1$. In order to do this, we have to update our plot line. After plotting the x and the y , we add a new input to declare the color, line style and marker style. Figure 7 is a table with all the display property that a plot can handle.

Color Code	Color	Marker Code	Marker	Line Style Code	Line Style
y	Yellow	.	Point	-	solid
m	magenta	o	Circle	:	dotted
c	Cyan	x	x-mark	-.	dash-dot
r	Red	+	Plus	--	dashed
g	Green	*	Star		
b	Blue	s	Square		
w	White	d	Diamond		
k	black	^	Triangle (up)		
		<	Triangle (left)		
		>	Triangle (right)		
		p	pentagram		
		h	hexagram		

Figure 7 Line styles, markers, and colors for the PLOT command.⁹

So if we wanted to change `y3` from a red line to a black dotted line with star markers, we would write this in the `plot` function:

```
plot(x, y1, x, y2, x, y3, 'k*');
```

If we compile everything into one m-file, our program would look like this:

```
x = linspace(-5, 5);
y1 = exp(x);
y2 = sin(x);
y3 = 2.*x.^2 + 5.*x + 1;
plot(x, y1, x, y2, x, y3, 'k*');
title('Various Equations'); % adds a title to the graph
xlabel('x axis'); % adds a x-label to the graph
ylabel('y axis'); % adds a y-label to the graph
legend('e^x', 'sin(x)', '2x^2 + 5x + 1'); % creates a legend
```

The output of this program would be (Figure 8):

⁹ Hertiner, Marc E. (2001) Programming in MATLAB®. Australia: Brookes/Cole.

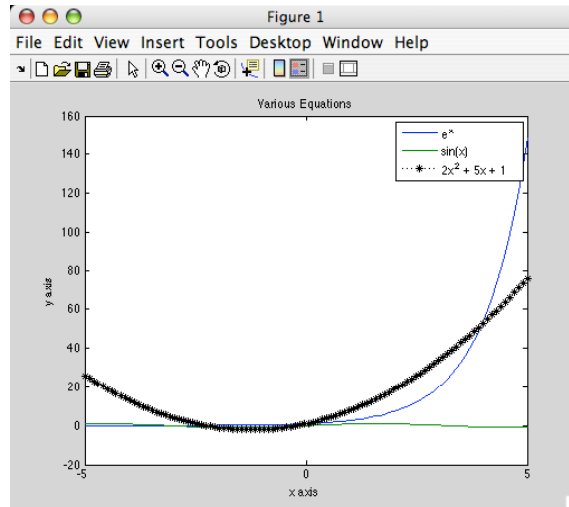


Figure 8 Output for the entire plot2.m program (It may be hard to see the stars but the stars give the black line its thickness.)

5. Evaluation

This section evaluates the properties of MATLAB, which will cover topics that will critique the language. Topics will include simplicity, critiquing the control structures and data types, and Type checking among other things.

MATLAB is a complex language and at first the language can become very overwhelming to a new programmer. After spending a few solid hours of messing with MATLAB, you get used to the language and the language becomes very simple to use; much of the syntax is similar to C++. The code is easy to understandably read and write.

MATLAB has many of the data types that are needed in a programming language however since MATLAB focuses primarily on numerical computations, MATLAB's strings are inferior to the strings in other languages. MATLAB is also made more simplistic by the addition of weakly typed variables.

MATLAB's control structure is more than enough to control data flow in programs. Loops and IF-ELSE statements are treated in exactly the same way as most other languages. Syntax for the FOR loops are different than other languages. However, the concepts are the same as other languages. Syntax for IF-ELSE statements and WHILE loops are similar to most other languages and make these control flow statements easy to learn and use in the programmers code.

The expressivity of the MATLAB programming language is very easy to read and write. The expressivity allows the programmer to accomplish large tasks and computations with a small amount of code. This makes the readability and the writeability of the code very easy. However if one did not know what some of the functions did, it may get hard to read the code, because of this, there is a small tradeoff for readability.

MATLAB, unfortunately, does not have any support for Object Oriented Programming Abstraction as a language such as C++. MATLAB also lacks any form of type checking in the language. Because MATLAB is primarily focused on numerical computations, the need for abstraction and type checking is unnecessary and really has no effect on the language.

The chart below, figure 9, follows Sebesta's evaluation criteria for programming languages; It breaks each characteristic of the programming language into three criteria, readability, writeability, and reliability.

Characteristic	Criteria		
	Readability	Writeability	Reliability
Simplicity	X	X	X
Orthogonality	X	X	X
Control Structures	X	X	X
Data Types and Structures	X	X	X
Syntax Design	X	X	X
Support for abstraction	-	-	-
Expressivity	X	X	X
Type Checking	-	-	-
Exception Handling	-	-	-

Figure 9 Table evaluating the MATLAB programming language using the criteria laid down by Sebesta in the book "Programming Languages".

From this evaluation, we can determine that MATLAB is a great tool for the development of engineering, scientific computing, and simulation programs. Though MATLAB may be able to handle the creation of other forms of programs, if we take into consideration cost, MATLAB would probably not be the best choice due to having to create unique "work-arounds" for not having certain features that other languages would have. I would highly recommend any mathematics, engineering, or computer science major to learn MATLAB for the development of computational programs, especially for students looking to continue their education into graduate school.

Appendix: Further Learning and Resources

This paper only touches on the necessary basics for grasping MATLAB; for a more in depth study on the MATLAB programming language, these books are a good starting point:

Programming in MATLAB by Marc Herniter (Brooks/Cole)

A Guide to MATLAB: for beginners and Experienced users by Brian Hunt, Ronald Lipsman, and Nathan Rosenberg. (Cambridge Press)

A Guide to MATLAB Object-Oriented Programming by Andy Register (Chapman & Hall)

Basics of MATLAB and Beyond by Andrew Knight (Chapman & Hall)

These book touch on the basics of MATLAB. One of the best books, and one of the greatest helps in writing this paper was Herniter's *Programming in MATLAB*. If you are looking for one book to get you started in programming in MATLAB, I feel that this is a must own book.

There are also two other sources that provide a great deal of information for free. The first option is to use the help menu within the program itself. The second option is the online documentation for the language. Both of these sources are a bit more technical and if you are looking for simplicity, these may not be right for you, they do provide a great deal of information and the online documentation was a great source throughout the writing of this paper.

There are many other great books written that discuss how to do other things in MATLAB such as simulations, more complex mathematics, and much more. A quick look on an online bookstore will help you find the book that is right for your uses.

Works Cited

- (1984-2007) "MATLAB Documentation." *The Mathworks*. Retrieved on Nov. 25, 2007 from <http://www.mathworks.com/access/helpdesk/help/techdoc/index.html>
- (August 1997) "MATLAB Programming: Object Oriented Programming Example." *University of Michigan, College of Eng, and Computer Science*. Retrieved on Nov. 25, 2007 from <http://www.engin.umd.umich.edu/CIS/course.des/cis400/matlab/oop.html>
- Goering, Richard. (2004 October). "Matlab edges closer to electronic design automation world." *EE Times*. Retrieved November 12, 2007, from <http://www.eetimes.com/news/design/showArticle.jhtml?articleID=49400392>
- Hertiner, Marc E. (2001) "Programming in MATLAB®". Australia: Brookes/Cole.
- Moler, Clove. (2004). "The Origins of Matlab." *The MathWorks, Inc*. Retrieved November 12, 2007, from http://www.mathworks.com/company/newsletters/news_notes/clevescorner/dec04.html