Personal Project Report

Adeet Patel

000530 – 0200

Creating a Digital Math Toolbox

5,387 Words

Atlantic Community High School

2015 – 2016

Table of Contents

Criterion A: Investigating

Goal and Global Contexts

My goal for this Personal Project is to create a mathematical computer program (named MATHBOX 2015) in the MATLAB programming language that performs calculations with a variety of mathematics, such as basic operations, algebra I, algebra II, precalculus and calculus. More specifically, my goal is to create a mathematical computer program that reflects the various types of mathematical skills and processes that I have learned in the IB Math program and outside of school. This project most closely relates to the global context of scientific and technical innovation since it implies how humans create technologies, such as computer programs, in order to facilitate mundane tasks and responsibilities. In this situation, the project resembles how a mathematical computer program is created in order to facilitate mathematical computations applicable in the real world and in scientific processes.

My Introduction to MATLAB

I needed to research information about the code and structure I desired to incorporate into MATHBOX. This is mainly because I am fairly new to the MATLAB programming language, as my cousin had introduced it and its abilities to me over the summer. I was shown the various mathematical abilities of the programming language, such as 2D and 3D plots and graphs, solving equations, and differential and integral calculus and how it is the best programming

language to use if a program that needs to perform mathematical calculations wants to be written. I also learned that the term MATLAB actually stands for "**mat**rix **lab**oratory" and that it is optimized for performing calculations with matrices, which is a reason why I decided to write code in my program that would allow it to perform these types of calculations.

Research

I essentially taught myself the basics of the MATLAB programming language by researching information about the syntax of the language and how to do extremely basic tasks, such as defining variables, clearing the workspace (the place where MATLAB stores all variables defined in a program), and clearing the command window in which MATLAB outputs information about variables, mathematical computations and results, and any warnings or errors in the script's code. I received most of the information I learned from came from the MathWorks, Inc. website, which is the company that created the MATLAB programming language.

Then I taught myself how to write code for calculations with matrices and their inverses since I would need this knowledge in order to write the matrix computation code for MATHBOX. I also later taught myself more advanced functions and commands which would be vital for my program to perform the calculations I desire. Some of these functions and commands are alphabetically listed as follows*:

Command / Function	**Result / Purpose**
\n	Generates a carriage return (the character typed when the ENTER key is pressed on the keyboard)
clc	Clears the MATLAB command window
clear all	Clears the MATLAB workspace
diff(*y*, *n*, *var*)	Calculates the *n*th order derivative of a symbolic function *y* with respect to variable *var*
ezplot(*y*)	Generates a plot (graph) of the symbolic function *y*

fprintf(*text*)	Prints the string *text* to the MATLAB command window
int(*expr*, *var*, *a*, *b*)	Calculates the indefinite integral of a symbolic function *expr* with respect to the variable *var*. If *a* and *b* are provided, then it calculates the definite integral from *a* to *b*.
subs(*y*, *var*, *c*)	Substitutes a value *c* for the variable *var* in the symbolic function *y*
syms	Declares a symbolic variable for use in computations with calculus

* Note that the parameters for some functions have been changed in order to match their application in MATHBOX.

I used the functions listed above repeatedly and regularly throughout my program, and some were actually vital for the program to display information and communicate with the user. Fortunately, by the time school started, I had enough MATLAB knowledge to begin writing the basic structure of MATHBOX.

My research did not only involve learning the MATLAB programming language. It also involved teaching myself some mathematical concepts I had not learned before in school, because I wanted to write as much code as possible for the different mathematical concepts. For example, I researched inflection points, which are points where the concavity of a function changes (calculated using the second derivative test) and how to animate plots, which I use in the "PRTACCL" program in order to display a free-falling particle. See the Criterion B and C sections for details about the different capabilities of the program.

Criterion B: Planning

MATHBOX Criteria

MATHBOX needs to have a separate script for each type of mathematical concept or mathematical calculation that it will perform (which will be referred to as ***mini-programs*** from now on), and it needs to have one base script or main script that incorporates each of the scripts

and consolidates them into one strong script for the program. This script's code should be simple yet essential to the program's execution. The script should be named "MATHBOX_Application.m", where .m is the extension of the file. The interface of this script will be referred to as the ***main interface*** from now on. This process will be more efficient rather than creating one huge script containing all of the program's code, as executing it will be cumbersome and will waste large amounts of time.

When MATHBOX is run, it should display rules on how to enter commands. These rules are: lowercase letters only because MATHBOX is case sensitive, all commands must be entered in single quotation marks, and type the command "help" to access a list of commands and their functions. Subsequently, MATHBOX must ask the user to enter a command. The conditional statements in the main script will determine whether or not the command entered is valid. If the command entered is valid, the program should clear the screen and launch the mini-program associated with the command. Conversely, if the command is invalid, MATHBOX should display an "invalid command" error and ask the user to enter a command again.

Once a mini-program is launched, user input is self-explanatory. The mini-program will give instructions by telling the user what kind of input to type, such as numbers, an operand (such as +, -, *, and /), a string, etc. After all user input has been done, the mini-program will perform the appropriate calculations with the user input and will print an answer or result on the screen, and then it will return to the main screen to ask the user to enter a command.

If the user input is inappropriate (e.g. if the user enters a string when he or she is supposed to enter a number, or vice versa), the mini-program should display an error message and, if the error is very minor, should return the user back to the main interface. However, if the user input error is so significant that it actually causes an error in the actual MATHBOX code

itself, there will be no other option but for MATHBOX to exit completely, and the user will then have to reopen MATHBOX. Since I am relatively new to MATLAB, I have not yet gained the experience to improve this problem due to a lack of time.

The script for each mini-program should have a reference to MATHBOX_Application.m at the end after the mathematical computations for that mini-program have been completed. A "Help.m" script also needs to be written, which must contain a list of all MATHBOX commands and their functions in order to help the user type the proper commands.

MATHBOX's Development Process

Visual C++, Encountering Problems, and Transition to MATLAB

I knew I had some prior experience in programming in the Visual C++ programming language, developed by Microsoft. As a result, my initial decision was to write MATHBOX in Visual C++. However, I did not want to start directly writing MATHBOX in this language, and instead I decided to first test previous applications and how Visual Studio behaves in regards to program errors. I noticed that I encountered many problems doing so.

One of my problems with Visual C++ was the fact that Visual Studio (Microsoft's program used to write code for various types of applications and subsequently build, or execute, these applications) had a tendency to generate fatal errors irrelevant to the code when I attempted to build the application. For example, Visual Studio once became unresponsive when I attempted to build an application. I had to forcefully terminate it using Task Manager, which successfully occurred. However, when I opened Visual Studio and attempted to build the same application, I received an error stating that Visual Studio was unable to open the .exe executable file for the program, which was required in order for the application to build successfully. As a result, from

that point on, I was unable to build that program regardless of the code's perfection. I attempted to fix the problem by installing Visual Studio inside of a virtual machine running the Windows XP operating system, and I re-created from scratch the program I wanted to build. Fortunately, the program built successfully and ran normally.

Later, I encountered a problem in which I was not able to delete a C++ project folder at all. The cause of this problem was unknown, and I even attempted to fix the problem by creating a script in another programming language, AutoIt, that would delete the folder, but to no avail. However, several days later, for unknown reasons, I managed to successfully delete the unnecessary files and folder.

Due to these reasons, I decided that it would be extremely difficult and cumbersome to write MATHBOX in the Visual C++ programming language. Once I discovered the MATLAB programming language and its mathematical capabilities, I even decided to write MATHBOX's code that was responsible for mathematical computations in MATLAB and the basic application interface in Visual C++ since I did not have experience with coding application interfaces in MATLAB.

Coding MATHBOX in the MATLAB Programming Language

After discovering MATLAB's mathematical capabilities, I found that it would be much easier to code MATHBOX in MATLAB. This is when I began to create mini-programs for MATHBOX. I wanted each of my mini-programs to ask the user for input necessary to perform the mathematical computations respective to the appropriate mini-program. I also wanted each program to display the results in the command window and then create a graph of the results (e.g. if the result was a function or equation, I wanted to display the graph of this function or equation).

I had plans to create mini-programs for differentiation and its applications, integration, conversion of equations from one coordinate system to another, compound interest, completing the square, graphing functions and equations, determining a line's equation from two given points, matrix operations, finding the area under a curve, solving quadratic equations, and finding the sum of an arithmetic sequence or a finite or infinite geometric sequence.

Criterion C: Taking Action

Changes in Mini-Program Plans

My plans were working most of the time for the simple mini-programs, but for complex mini-programs, I had to change my plans based on my experience with MATLAB programming. Some of these plan changes are discussed below.

Systems of Equations ("SYSEQ2", "SYSEQ3", "SYSEQ4")

My initial plan for mini-programs solving systems of equations was to use the "solve" function that would set a particular expression equal to another expression and solve for the specified variable. However, since I wanted to do systems of equations with up to 4 variables, I decided that this task would be too complicated. Instead, I decided that I would code the mini-program to use matrices (Cramer's rule) in order to solve a system of equations with two, three, and four variables. I knew that this is much easier to do because MATLAB is optimized for matrix computations.

Abandoned Mini-Programs

Significant Figures ("SIGFIGS")

For example, I had plans to create a mini-program that would calculate the number of significant figures in a number. However, I had to abandon the idea due to the potentially

complex nature of the mini-program's code. The program involved splitting the number into digits and then determining which digits were significant, and then it would return a result stating the amount of significant figures in the number. The idea worked for decimals such as 2.76 but did not work for integers with zeros such as 6033.

Implicit Differentiation ("IMPDIFF")

Since I had already created a mini-program that calculates explicit derivatives, I had the idea to create a mini-program that calculates implicit derivatives. However, when I attempted to do this, I found that due to my lack of skill and knowledge of this topic, it was very difficult for me to have MATLAB perform multivariable differentiation instead of simply differentiating with respect to one variable, and as a result, I abandoned the idea.

Functions of Mini-Programs

MATHBOX's developmental process mainly involved my often spontaneous creation of ideas. I decided to begin MATHBOX by creating simple mini-programs for simple concepts and then making the mini-programs more complex in order to account for more complex mathematical situations.

Each mini-program will be discussed in the order they were initially created, from oldest to newest. Mini-programs that were abandoned will be discussed last. The headings of each section display the command the user needs to enter into MATHBOX in order to access that specific mini-program.

"POLYTANLINE"

This mini-program calculates the equation of a tangent line of a polynomial function (with a maximum degree of 5) at a specific *x* value. It gives the user the standard equation of a polynomial, which is the following:

$$f(x) = ax^5 + bx^4 + cx^3 + dx^2 + ex + f$$

It subsequently prompts the user to enter values for *a*, *b*, *c*, *d*, *e*, and *f*. It then asks the user for an *x* value at which to calculate the tangent line's equation. Next, it asks the user for the horizontal window size (range of *x* values) for the graph and the interval value at which to plot the graph (smaller interval values will result in a smoother graph). The mini-program subsequently performs necessary computations, such as first differentiating the specified polynomial function with respect to the variable *x* and then calculating the slope and y-intercept. Finally, the mini-program displays the equation of the tangent line and the graph of the polynomial function along with the tangent line.

"POLYAREAFUNC"

This mini-program calculates the area under a polynomial, with a maximum degree of 5, within the user-specified bound. The mini-program then calculates the definite integral of the polynomial with respect to the variable *x*, and then displays the result and the graph of the polynomial and bound region. Displayed below is the Fundamental Theorem of Calculus, which the mini-program uses to calculate the area of the bound region of a sample polynomial function f(x) from the left bound *a* to the right bound *b*:

$$\int_a^b f(x) = F(b) - F(a), \quad where \ \frac{d}{dx}F(x) = f(x)$$

"QUADEQ"

This mini-program provides the standard quadratic equation, and asks the user to enter values of *a*, *b*, and *c*. This quadratic equation is displayed below:

$$f(x) = ax^2 + bx + c$$

Subsequently, the mini-program uses the quadratic formula to determine the solutions:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Finally, the mini-program will print the two values of *x* in the MATLAB command window.

"MATRIX"

This mini-program adds, subtracts, multiplies, and calculates the inverses of two matrices, A and B, provided by the user. The user is first asked to specify the number of rows and columns in each matrix. Next, the mini-program asks the user to enter one component (number in the matrix) at a time while displaying the matrix values they have entered so far (the components that are not yet entered by the user are displayed as zeros). After the user has entered all components of both matrices, the mini-program asks the user for an operator. The available operators and their meanings are listed in the table below:

+	Adds the two matrices.
-	Subtracts the two matrices.
*	Multiplies the two matrices.
inv	Calculates the inverse of each matrix.
inv+	Adds the inverses of each matrix.
inv-	Subtracts the inverses of each matrix.
inv*	Multiplies the inverse of matrix A by the inverse of matrix B.
scal	Multiplies each matrix by a scalar.

If the user does not specify an operator above, MATHBOX will return "ERROR: Invalid operator" and the program may exit since it causes a syntax error in MATHBOX's code.

"GRAPH"

This mini-program asks the user to enter a function. First, it declares the symbolic variable *x* so that the user can use it when he or she enters a function. After the user enters a function, the mini-program will make the appropriate ordered pair calculations and will display the graph in a separate window.

"VECTOR"

This mini-program asks the user for the initial points and components of vectors *u*, *v*, and their cross product $u \times v$ (which is calculated later). The mini-program first calculates the dot product and magnitude for vectors *u* and *v*. Using this new information, it then calculates the angle between the two vectors, which is:

$$\theta = \cos^{-1} \frac{u \cdot v}{|u|\,|v|}$$

However, since MATLAB calculates angles in radians, the angle has to be multiplied by 180 / π to be in degrees. Next, it calculates the cross product vector by using matrix determinants and expansion by minors, and finally displays a 3D graph of the three vectors using the "quiver3" function in MATLAB.

"MATHBOX_Application.m"

This file is the main script of MATHBOX. It is where all of the mini-programs are incorporated into one large and complex program. The code of the main script is simple yet very powerful, and it is vital for MATHBOX to run and pack together all of the mini-programs. This main script works entirely on the basis of conditional statements. When the script is run, it

displays the main MATHBOX interface, where the user is asked to enter a command. When the user enters the command, the script will analyze which command was entered. It will execute the appropriate conditional block and will subsequently launch the mini-program that is associated with the command. If the user enters an invalid command, MATHBOX will display the error "ERROR: Invalid command entered" and it will ask the user to enter a command again.

"HELP"

This file is the help section of MATHBOX. It contains a full list of all MATHBOX commands and their functions. New users should type "help" in order to display the list.

"SYSEQ2", "SYSEQ3", "SYSEQ4"

These three mini-programs give the user a standard system of equations. "SYSEQ2", "SYSEQ3", and "SYSEQ4" solve systems of equations with two, three, and four variables respectively. Each program asks the user to enter coefficients for each variable. Below is a standard system of equations in two variables and its corresponding matrix form that MATHBOX uses to solve the system.

$$\begin{cases} a_1x + a_2y = a_3 \\ b_1x + b_2y = b_3 \end{cases}$$

$$\begin{vmatrix} a_1 & a_2 \\ b_1 & b_2 \end{vmatrix} \begin{vmatrix} x \\ y \end{vmatrix} = \begin{vmatrix} a_3 \\ b_3 \end{vmatrix}$$

The mini-program solves for the *x* and *y* matrix by multiplying by the inverse of the matrix containing the coefficients on both sides. It then displays the *x* and *y* values separately (not in matrix form) in the MATLAB command window.

"EXPDIFF"

This mini-program calculates the derivative of an explicit function (a function in terms of x only) to the order specified (second order, third order, etc.). It first declares the symbolic variable x so it can be used in the user's function. The mini-program then makes the calculation and displays the derivative in a clear and neat mathematical form (e.g. with the fraction bar instead of a forward slash).

"DEGTORAD", "RADTODEG"

These mini-programs convert an angle from degrees to radians and radians to degrees, respectively. The former mini-program asks the user for an angle in degrees, and then simply multiplies it by π / 180 to express the angle in radians.

The latter mini-program asks the user for an angle in radians, and then simply multiplies it by 180 / π to express the angle in degrees.

"PRTACCL"

This physics mini-program animates a free falling particle. It asks the user to enter an initial height (h_0) in meters and gravitational acceleration (g) entered as a negative value in meters per second squared. After these values have been entered, a graphing window will be displayed and the animation of the free-falling object will occur. Every second, the downward velocity of the object will increase by $|g|$ (absolute value of g) meters per second. This animation does not take terminal velocity (the maximum velocity achieved by the object) into account. It also assumes that air resistance is negligible.

"LTRANSFORM"

This mini-program performs one of three calculations: (1) it calculates the new coordinates of a given initial point after a linear transformation has been performed, (2) it calculates the linear transformation matrix, given two initial points and two terminal points, and (3) it calculates the initial coordinates of a given transformed point after a linear transformation has been performed. A linear transformation occurs when a point (expressed as a vertical matrix with the *x*-coordinate at the top and the *y*-coordinate at the bottom) is multiplied by a matrix.

"SERIESSUM"

This mini-program asks the user if he or she wants to calculate the sum of an arithmetic or geometric series. If the user chooses the arithmetic series, the user must simultaneously choose to calculate the sum by entering a first term, last term, and number of terms, or by entering a first term, common difference, and number of terms. After the user enters the required input, the mini-program will print the result (the sum) in the MATLAB command window.

If the user chooses the geometric series, the user must simultaneously choose to calculate the sum of a finite or an infinite series. If the user chooses a finite series, the first term, common ratio, and number of terms must be entered. If the user chooses an infinite series, the first term and common ratio must be entered.

If the user chooses an infinite series, the mini-program will first check the value of the common ratio. If the magnitude of the common ratio is less than 1, the mini-program will proceed to calculate the sum. On the other hand, if the magnitude of the common ratio is greater than or equal to 1, the mini-program will not calculate the sum and will return a message stating "Sum does not exist. The series diverges."

"VOLSOLIDREV"

This mini-program will ask the user to enter a function *f(x)*, and the left bound and right bound for which to calculate the volume of the solid of revolution about the x-axis. Like the "POLYAREAFUNC" mini-program, it also uses the Fundamental Theorem of Calculus, but in this case, it multiplies the entire integral by pi.

Unfortunately, due to a lack of enough experience with programming in MATLAB, I did not have the skill and knowledge to create a 3D plot of the solid of revolution, and thus, this mini-program only calculates the **volume** of the solid of revolution. However, the mini-program does graph the function *f(x)* entered by the user.

"APPLDIFF"

I regard this mini-program as the most complex I have created out of all mini-programs. Given a polynomial function of any degree (> 0), this mini-program uses applications of differentiation in order to calculate critical points (where the derivative is zero) and inflection points by using the first and second derivative tests, respectively. I was not able to calculate critical points where the derivative is undefined, due to a lack of enough skill and knowledge on the MATLAB functions dealing with this situation.

First, the user is asked to enter the degree of the polynomial, and then the user is asked to input coefficient for each *x* term. The mini-program also shows the part of the polynomial that the user has defined so far, which is stored in a variable called "CurrentPolynomial." However, CurrentPolynomial will not display *x* terms whose coefficients are zero (since they are obviously negligible).

Next, the user is asked for an interval which will be used to calculate values of the constant *c* using the Mean Value Theorem.

Then the mini-program will calculate critical points and possible inflection points, and will determine whether or not the possible inflection points are actually inflection points by analyzing the concavity of the function. If it calculates that the concavity changes at that specific point, then the mini-program counts it as an inflection point. If there are no inflection points, the mini-program will return a message stating so.

After the mini-program has completed analytical calculus computations, it will generate a graph of the function, and will also plot points marking the location of critical points, inflection points, and *x*-intercepts (calculated by using MATLAB's "solve" function).

"COMPSQ"

This mini-program completes the square of a quadratic equation in expanded form. It uses the formulas below to calculate the vertex and the coefficient of the equation, and subsequently put the equation into a completed square form:

$$h = -\frac{b}{2a}$$

$$k = c - \frac{b^2}{4a}$$

The mini-program will change the signs of the values inside the completed square form based on the signs of *h* and *k*. Finally, the completed square form of the equation is printed in the MATLAB command window.

"COMPINT"

This mini-program calculates compound interest (continuous or for a specific number of compoundings per year). The user is asked to enter the principal amount, interest rate per year, number of compoundings, and time in years. The mini-program then performs the calculations. It displays the new investment amount and interest earned over time.

The mini-program also projects the growth of the investment for a specific number of years, which is specified by the user. It then displays the graph of the growth of the investment and compares the growth to what it would be if interest compounded continuously.

"INVNORMCDF"

Given the mean and standard deviation for the normal distribution, this statistics mini-program either calculates the area between a particular range of X values or calculates the X value at and below a given area. This mini-program resembles the invNorm and normalcdf functions on the Texas Instruments calculator TI-84 Plus C and other calculators in that particular TI family. The two function names were combined to create the name for this mini-program, "INVNORMCDF."

"STRDSCORES"

Given the means, standard deviations, and particular X values of two normal distributions, this mini-program standardizes (scales the X values on the standard normal distribution, which has a mean of 0 and standard deviation of 1) the two X values and returns the two standardized scores (also known as Z-scores).

"LINEEQPT"

This mini-program calculates the equation of a line given two points. The user is asked to enter the *x*- and *y*- coordinates of two points. The mini-program calculates the slope and y-intercept of the line that passes through the two points by using the algebraic formula for slope and then substituting a point and the slope into the standard linear equation ($y = mx + b$)and solving for the variable b (the y-intercept).

"CARTPOLEQ"

This mini-program converts an equation in the Cartesian (also known as rectangular) coordinate system and converting it to an equation in the polar coordinate system. The algorithm that the mini-program uses is correct, although the mini-program has weaknesses. For example, it cannot convert the equation of a straight line and other equations. MATLAB does not generate any errors while performing the computations, but the mini-program may sometimes return an incorrect answer.

"SYNTAX"

This mini-program displays the syntax rules that must be followed in MATHBOX. It discusses rules such as operators and functions. It also discusses how function notation used in manual and written mathematics calculations may not always be the same as the function notation used in MATHBOX. For example, to express the natural logarithm of *x*, one would normally write ln *x*. However, in MATHBOX, "log(*x*)" has to be entered.

Technically, these are not MATHBOX rules, but are in fact MATLAB rules. This is because it is very complex and cumbersome to create MATHBOX rules, so I instead decided to use MATLAB rules for simplicity.

"MATH1", "MATH2"

These mini-programs perform the most basic mathematical operations. "MATH2" performs calculations with two numbers such as basic number addition, subtraction, multiplication, and division. "MATH1" performs calculations with one number including, but not limited to, the square root and nth root, logarithms, and raising e to the specified number.

Commonalities Between All Mini-Programs

Each mini-program has the commonality of displaying a message that says "Press OK to continue" after all mathematical computations and displays have been performed and completed. The user can press OK in order to return to MATHBOX's main text interface.

Criterion D: Reflecting

Evaluation Against Criteria

I discovered that I was able to write MATHBOX without many problems after I switched to MATLAB, although I was experiencing major issues attempting to write the program in the Visual C++ programming language.

I was generally able to write the program as planned, despite that I had to abandon a few ideas as discussed earlier, such as implicit differentiation and significant figures. However, I feel as if it may have been possible for me to have created more mini-programs that do more types of mathematical calculations. For example, I did not have many mini-programs on statistics, and if I had more time, it might have been possible for me to create programs that calculate correlation between two variables, for instance, or to display statistical graphs such as boxplots and histograms.

Obviously, I knew it was not possible that I would write MATHBOX perfectly according to my plan. As I wrote the mini-programs, I had experienced problems with graphs and other mathematical calculations. For example, at first, I was not able to code my "GRAPH" mini-program in such a way that I was able to graph horizontal lines. For example, if I attempted to enter $f(x) = 3$ as a function, MATLAB would generate an error and would terminate the MATHBOX application. Then I had figured out that I could still keep my code as is and enter $f(x) = 0*x + 3$ instead of $f(x) = 3$. Also, for some of my mini-programs that graph functions along with separate points, I received warnings that stated that MATLAB was only displaying the real part of imaginary data. I was not able to solve the issue causing the warning, and as a result, I had to code the mini-program to state that "x-intercepts may not appear on the function."

Also, my initial plan was to insert a "clear all" line at the beginning of each mini-program. However, I later discovered that this actually slowed down the program (especially when defining symbolic variables), so I removed the line.

In short, there were many small changes I had to make to the program in order for it to suit the criteria I created for the product, but it mostly ended up the way I expected.

Extension of Topic Knowledge and the Global Context

Creating the MATHBOX 2015 program has greatly increased by knowledge of MATLAB and how the programming language works. I have already been writing code in various programming languages for about four to five years, so I do have experience with programming, but MATLAB is the first more powerful programming languages I discovered. Creating mini-programs such as "APPLDIFF" has greatly increased my knowledge of how MATLAB performs calculations with functions, derivatives, arrays, etc. Other mini-programs have increased my knowledge of how MATLAB creates both two-dimensional and three-

dimensional graphs because I now have some experience with how MATLAB performs mathematical and graphical calculations. In short, I have learned about the tremendously powerful abilities of the MATLAB programming language and how mathematicians, scientists, and engineers can use it to perform powerful and amazingly complex tasks.

Development as an IB Learner

This personal project has facilitated my development as an IB learner. For example, this project helped me build the IB learner profile traits of thinker, open-minded, risk-taker, inquirer, reflective, and knowledgeable. This is because this personal project required me to create innovative and creative ideas with each mini-program (including any graphing capabilities), be open to all ideas for how I can write the code for each mini-program, attempt new strategies for coding and designing the mini-program, learning how to do different things never done before (e.g. creating the animation in the "PRTACCL" program, even though I had no prior experience in MATLAB animations), reflect upon my own work and assessing what the program is good at doing and what features I need to improve, and to develop an understanding of how the MATLAB code works and what it can be used for.

Works Cited

Campus Learning Assistance Services. *Inflection Points*. n.d. Web. 1 November 2015.

EG1002JCU. *Week 4 - 6 Animating plots*. 6 March 2012. Web.

Kumon North America. *Math Level XM Solution Book*. Teaneck: Kumon Institute of Education, 2006. Print.

MathWorks, Inc. *Create line object - MATLAB line*. n.d. Web. 8 October 2015.

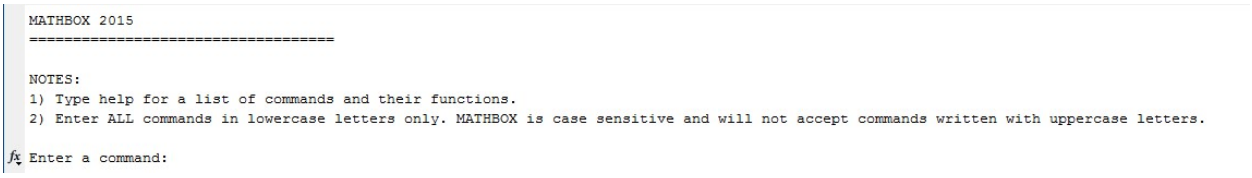
—. *Substitute into an object - MATLAB*. n.d. Web. 8 October 2015.

Simon, Jan. *Scientific notation to decimal? - MATLAB Answers - MATLAB Central*. 4 August 2013. Web. 31 October 2015.

Appendices

This section contains screenshots of MATHBOX running and some details about issues in Visual C++.

MATHBOX's main interface



MATHBOX printing the list of commands (note that this is not a complete list)

```
ALGEBRA II AND PRECALCULUS (6)
==================================================
COMPINT: Calculates compound interest, given the principal, interest rate, number of compoundings per year (or continuous compounding),
and the time in years.DEGTORAD: Converts degrees to radians.
LTRANSFORM: Calculates the linear transformation of a point by using matrices.
MATRIX: Performs matrix operations.
RADTODEG: Converts radians to degrees.
VECTOR: Performs the vector calculations of dot product, cross product, and the angle between two vectors and graphs the input vectors.

CALCULUS (6)
==================================================
APPLDIFF: Performs calculations related to applications of differentiation, given a polynomial function.
CARTPOLEQ: Converts an equation from Cartesian form to polar form.
EXPDIFF: Calculates the derivative of an explicit function (a function in terms of x).
POLYAREAFUNC: Calculates the area bounded by a polynomial function and two vertical lines.
POLYTANLINE: Calculates the equation of a tangent line at a specific point on a polynomial function of at most degree 5.
VOLSOLIDREV: Calculates the volume of a solid of revolution, given the function and interval.

PHYSICS (1)
==================================================
PRTACCL: Displays an animation of a free-falling particle using an initial height and gravitational acceleration specified by the user.

STATISTICS (2)
==================================================
INVNORMCDF: Either calculates the probability of a normal distribution between two bounds or the X value where a specific probability
occurs.
STRDSCORES: Calculates and compares the standardized scores (Z-scores) of two normal distributions.
```

A snippet of the main interface's code

```
12 -    if strcmp(cmd, 'exit')
13 -        break
14 -        clc
15 -    elseif strcmp(cmd, 'help')
16 -        HELP
17 -    elseif strcmp(cmd, 'syntax')
18 -        SYNTAX
19 -        PressOK
20
21      % GENERAL UTILITIES AND BASIC MATHEMATICS
22 -    elseif strcmp(cmd, 'graph')
23 -        GRAPH
24 -        PressOK
25 -    elseif strcmp(cmd, 'math1')
26 -        MATH1
27 -        PressOK
28 -    elseif strcmp(cmd, 'math2')
29 -        MATH2
30 -        PressOK
31
32      % ALGEBRA I ==============================
33 -    elseif strcmp(cmd, 'compsq')
34 -        COMPSQ
35 -        PressOK
36 -    elseif strcmp(cmd, 'lineeqpt')
37 -        LINEEQPT
38 -        PressOK
39 -    elseif strcmp(cmd, 'quadeq')
40 -        QUADEQ
41 -        PressOK
42 -    elseif strcmp(cmd, 'seriessum')
43 -        SERIESSUM
44 -        PressOK
45 -    elseif strcmp(cmd, 'syseq2')
46 -        SYSEQ2
```

MATHBOX in mathematical action (this is the "POLYAREAFUNC" mini-program)

```
THE AREA PROBLEM
================================

y = ax^5 + bx^4 + cx^3 + dx^2 + ex + f

Enter value of a: 1
Enter value of b: 3
Enter value of c: -2
Enter value of d: 1
Enter value of e: 4
Enter value of f: 5
Enter left bound: x = 3
Enter right bound: x = 5
Calculating area...
```

A graph generated by MATHBOX for the “POLYAREAFUNC” mini-program