JfxI-84

A JavaFX calculator similar to a TI 84

Dylan Fulop

CSE 205 MWF 9:40-10:30

Professor Navabi

CSE 205 Honors Project

**Introduction**

For my honors project, I decided to try to create an advanced calculator in Java using JavaFX. The calculator would work similarly to a Ti-84 calculator and have many similar functions that allow it to do powerful math. I decided to do this because it allows me to explore interesting details of how a product that combines two of my favorite subjects (programming and math) works. The goal was to create a calculator application that had many of the functions that a graphing calculator would have and could be used for high-level math such as graphing functions, trigonometry, estimating derivatives, finding the intersection of curves, and evaluating series.

**Requriements**

This project is meant to have as many functions of an advanced calculator as I could see myself having the skill, time, and knowledge to implement. Specifically, I want the calculator to be able to:

* Have a calculator tab that can do both basic and complex functions read from a string that can be typed manually or done with keyboard input such as
  + Addition, subtraction, multiplication, division, powers, roots (x+y,x - y, x\*y, x/y, x^n, n\|x, sqrt(x))
  + Trigonometric functions (sin/cos/tan,arcsin/arccos/arctan, csc/sec/cot)
  + Logarithms (log, ln, logbase (ran as logb(base, arg))
  + Floor and ceil functions
  + Max, min, sum, product, and average of a list of entries (max(1,2,3,5,4) = 5, etc.)
  + Modulus function (mod(n,m) ex. mod(3,10) = 3 mod(10,3) = 1)
  + Random functions (randint(min,max), rand(min,max))
* Have a mode tab with options for
  + Function type (allows parametric, polar, and function graphing)
  + Angle unit (degrees/radians)
  + Rounding (1-8 digits and min digits)
* Have an equations tab where you can input multiple functions to be used in other tabs
  + Same input menu, same ability, except can use x/t/theta as a variable and have multiple text fields
  + Can be called in calculator tab as Yi(x) where i is an integer 1-8ish and x is any number in function mode or r(theta) in polar mode or Yi(t) Xi(t) in parametric
* Have a graph tab where you can graph the equations in the equations tab and a window tab where you can change the window for the graph, also with the ability to
  + Graph in function, polar, and parametric mode with changeable step sizes to approximate graph
  + Approximate intersections, zeros, minimums, and maximums of curves in a region
  + Find value of curves at specific x (and be able to select different curves to do this with)

**Approach**

To begin this program, I thought about the different requirements that I would for sure want to include. I began by designing the GUI and then started to try to create the logic of the program in various steps.

From there, I began to consider how these requirements could be laid out in a more efficient GUI for a computer rather than basing it directly off the calculator. I decided to change a few things, such as whenever a button on the calculator would take you to a different page, that was translated into a different tab. I began by designing the GUI for each tab by first drawing out the basic elements of the GUI on paper and then writing the JavaFX code to make each tab. One thing that I realized, was that 2 of the tabs would need large grids of buttons (input grids) and that it made sense to make a class to represent that pane that would be able to build itself. In order to do this I would have a constant 2d array that included what all the buttons said. After designing the GUI, I implemented the basic classes for all of these panes so I could have a more concrete representation when I began thinking about the logic.

Next, I started to design what I thought would be the most difficult part of the program, but would also be the most important, the ExpressionParser class. This class would have several methods used to take a String and read it as an expression that it can solve. I began by listing different capabilities it should be able to have. I started with simple 4-function abilities. I realized that I could use the split method of the string class to separate terms by “+” “-” “\*” or “/”. I also immediately recognized that it would be useful to have a version of the expression parser method that takes a variable’s value and what that variable is for graphing purposes. That is, it should be able to take in x and a value of x (or t). This would be useful in the graphing tab when I need to parse expressions with variables. I realized the easiest way to do this is to just have a method that can be called with the equation and the variable to be replaced and the value to replace it with. I thought about how this should be done overall, and realized that I could use recursion to split up the expression until there are no more symbols and then solve in the reverse order. This means that these functions should appear in reverse order of operations, with addition first. I started by writing this amount of functionality to make sure I could use it, and used a Scanner to test it originally. I found an issue very quickly, because \* and + both have special meaning in the split method, so I tried using them as escape characters \+ \\* and it still didn’t work. After a lot of searching, I found out that I needed 2 backslashes, and that fixed that issue. I also began implementing powers “^” using the same method and math.pow to take a power. I then began implementing other functions similar ways such as sin cos tan, etc. I realized that I had to have a way to deal with their being more than 2 elements when split, and realized that would represent sinxsinx and decided on just multiplying these terms if that occured. I also realized that I now needed to create a class to handle information about the calculators mode, namely whether it was in degrees or radians. I made a class with several instance variables called MathMode to handle several options that would change. It took me a while to realize somehow that I had a problem with negative numbers, as they were split up as if they were subtraction. I decided that the simplest, if not ideal, solution for this was to just require a space on either side of the “-” sign. I then needed to work on some more complicated functionalities.

After already having implemented many functions, I realized that the way I had done it would not take into account parenthesis as a prioritization for order of operations. In order to solve this, I realized I needed to make a method to handle this by solving all things in parenthesis first, and then do the rest of the readExp method. I would simply replace (...) with whatever the inside solved to and then solve the expression from there. I also decided to add more functionality to the expression parser such as reading some functions that had multiple inputs. To do this, I needed to create another method that would go even before the solveParentheses method to solve these functions by first finding them, then figuring out where their opening and closing parentheses are, and finally by splitting the inside of the parentheses by commas. I designed this algorithm first on paper because it was somewhat confusing and I needed to write out some information and attempt to find the opening and closing parentheses in several scenarios. I realized I could simply increment a count of the number of opening parentheses each time one is reached and decrement it whenever a closed parenthesis is reached, and when that count is 0 the whole function is closed. (So max(1,2,(3+4),(5+max(7+8,9)),(3\*4),(3^(4+7))) would be able to recognize all 6 inputs). I decided after doing this that the prgoram would be much neater if I put sin, cos, tan, and all functions that take parameters in the form func(x) or func(x, y, …) as seperate methods so I created similar methods for most of the functions I had already implemented. After completing all of this, I decided it was time to move on to the next part of the project.

The next part of the project to work on was implementing functionality for the main pane. First I added a listener to the inputgrid class that would add the correct text to a specific text field. Then I needed to add a way to recall information about previous entries and print them on the main pane. I decided to do this by having 2 lists in the main class representing the string entries and the double outputs which would be displayed in the top half of the screen and removed when they started to go past the input grid. Next was the y=/eqs pane. This was fairly simple, as it simply required adding a listener to several pieces that allow you to a) select one of the equations to use the input grid for and b) press enter on either the input grid or enter on any text box to update a list of strings representing the equations in the main class. This was quite easily accomplished, so next I needed to add the function Y1-Y8 (the amount I decided fit on the pane best) to the expression parser, which I did by implementing it once and using a for loop.

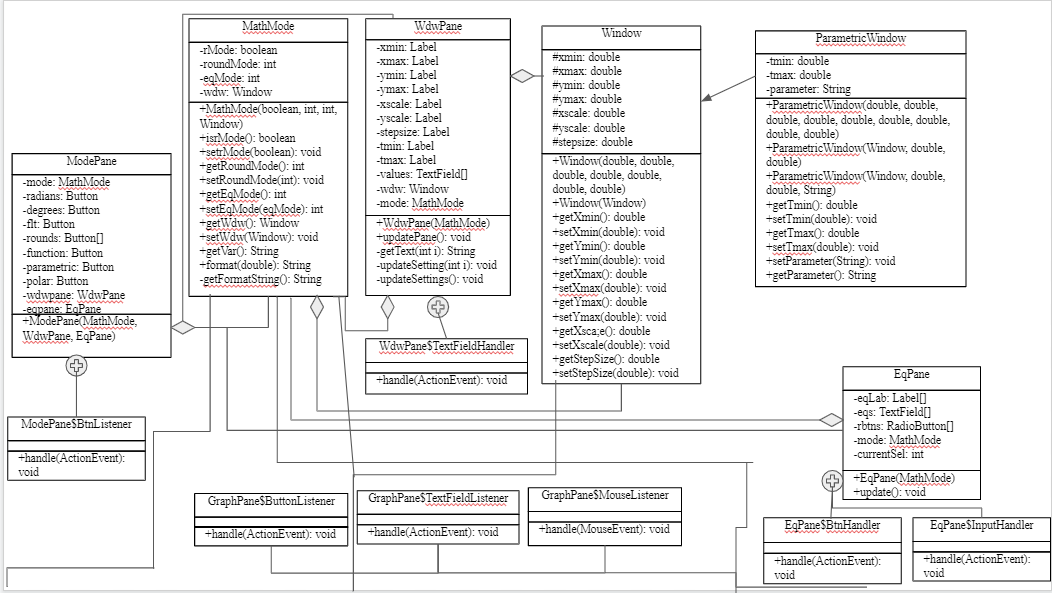
The next 2 things to work on were the mode and window tab. These were fairly simple and only required adding a few listeners and creating a Window class to handle all the information about the window and a ParametricWindow class that extended this to display when using parametric or polar mode that would allow you to change the min and max of the parameters. I also had to modify the window pane to be updated whenever the function/graphing mode was changed and to include if statements to add extra entries for these two options.

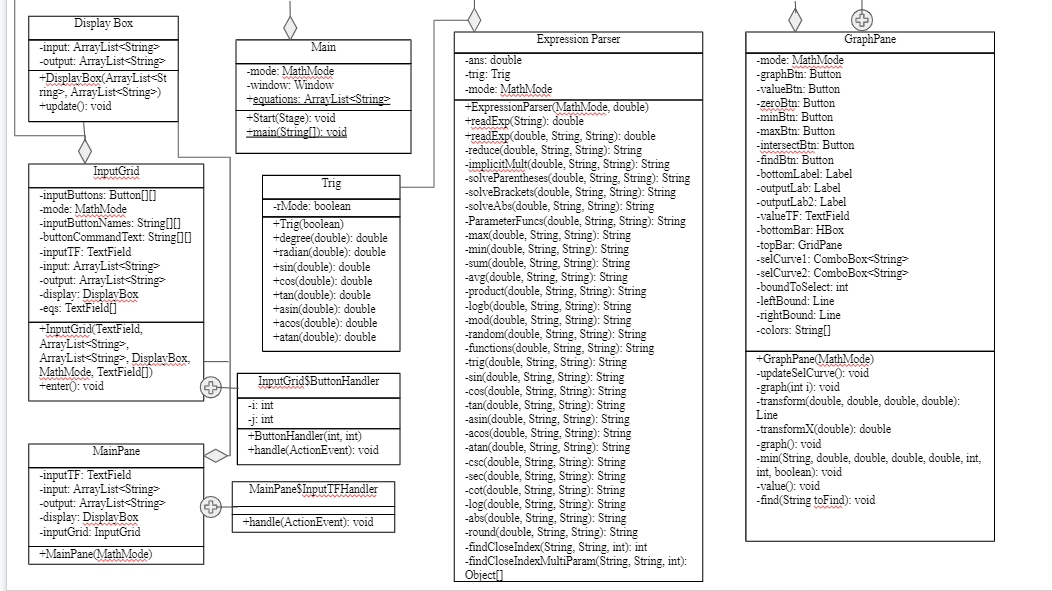
The final tab to work on before beginning more complex functionality was the graphing tab. At first I was very worried about the difficulty of this, but it turned out to be very simple because I had thought so much about it ahead of time. All I had to do was add a graph(int i) method that would graph y(i) and loop through that. The way it would graph it would be by looping with a variable x from the windows xmin to xmax and increasing by stepsize. It would solve yi at x and x + stepsize and draw a tiny line between those two points. This would require doing some transformations to change the scale to fit the pane based on the windows settings, so I created a transform method that would return a line based on the x and y coordinates and return the x and y coordinates in the pane. Polar mode and parametric mode would simply increase t or theta by the step size and solve the x and y coordinates (in polar rcostheta and rsintheta).

The final step was to add some functionality to the graph tab, such as finding values in all 3 modes and in function mode, finding maxes mins intersections and zeros. First I added panes to the top and bottom that would be used for output and input accordingly, then I added buttons for all of these features to the bottom pane. The first thing to work on was entering a value. I created a textfield that would appear when the value button was pressed, and a combobox to select a graph, in fact I created 2 combo boxes at this point with anticipation of needing 2 for finding intersections. When enter was pressed in the text field or the value button was pressed an output label would appear at the top evaluating the selected function(s) (depending on mode) at the entered x/t/theta value. I divided the other 4 tasks into 2 parts: finding the minimum/maximum (same thing but switching < and > signs and some variable names) and finding intersections and zeros (I would do zeros first then find the zero of the subtraction of the 2 selected curves for finding an intersection).

First I focused on finding minimums. The algorithm for this would be to have right and left bound, and set a variable (x) to the left bound.The x value would increase by a certain step size, and reset the value of another variable if the corresponding y was the new minimum. Then the method would repeat with a smaller step size, but would have smaller bounds that surrounded a certain radius of the last minimum it found (25 step sizes to be exact). I wrote this using recursion. For finding the maximum, it would simply minimize -Yi(x) and for finding zeros I would minimize abs(Yi(x)). Finally, for finding intersections I would need to change the min method a little bit to have a boolean that changed how it displayed its final results if set to true, and would then simply minimize abs(Yi(x) - Yn(x)) if finding the intersection of Yi and Yn.

**Design**

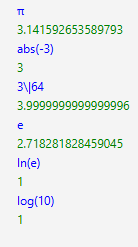
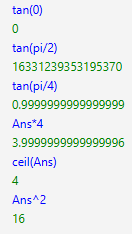
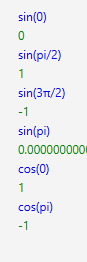
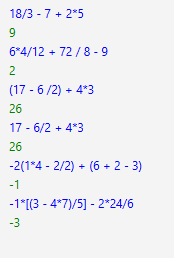
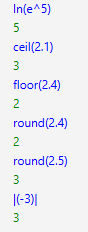
****

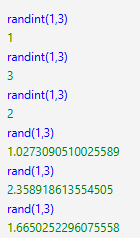
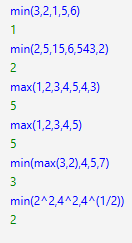
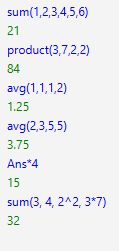
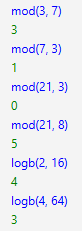
****

**Simulation**

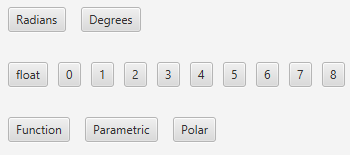
Main Tab Basics



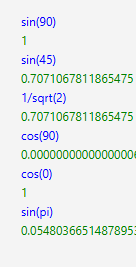
 



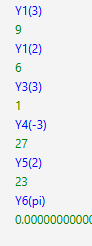
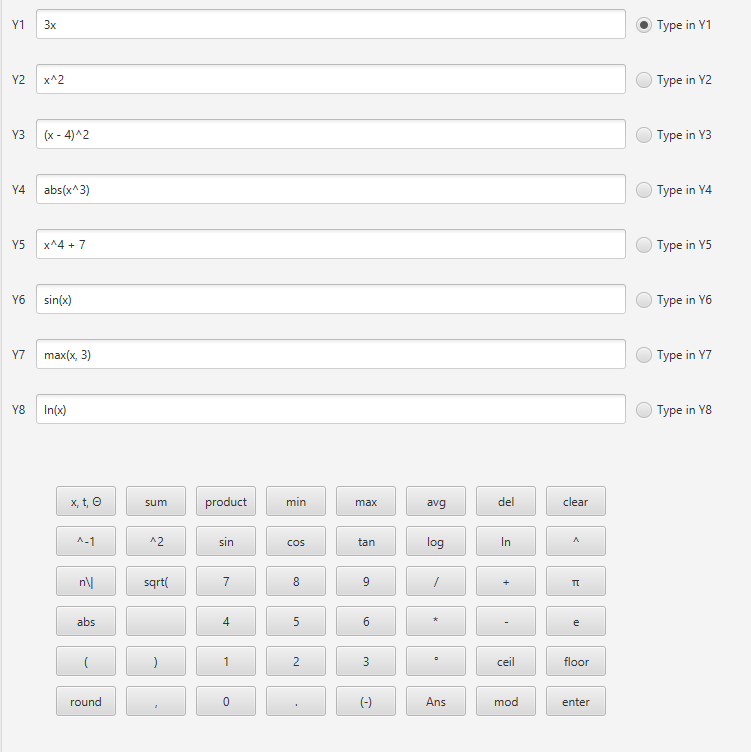
Mode Tab/Main Tab Basics



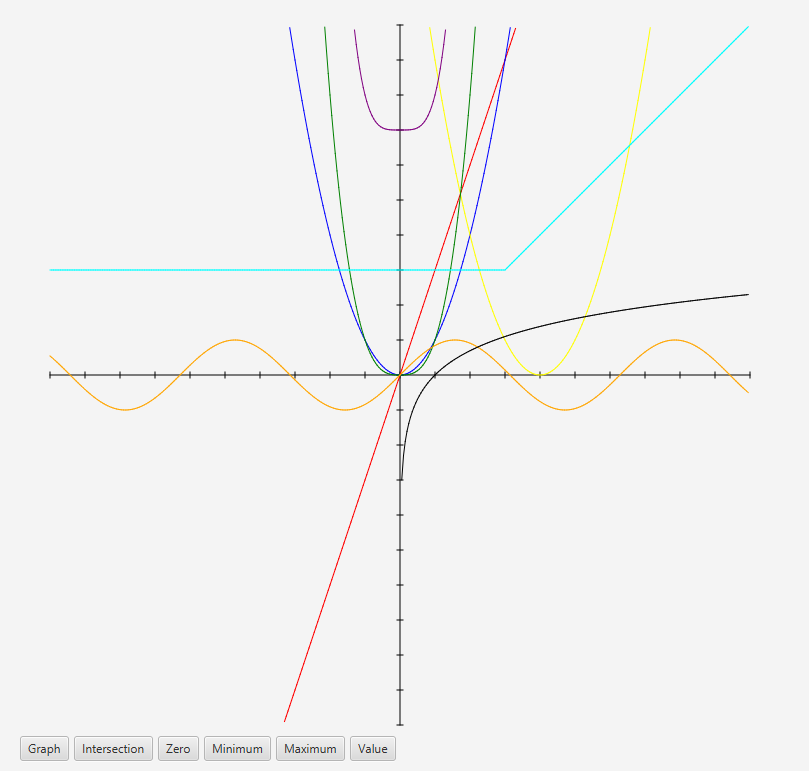
Degrees mode (left) Different Rounding modes (right)



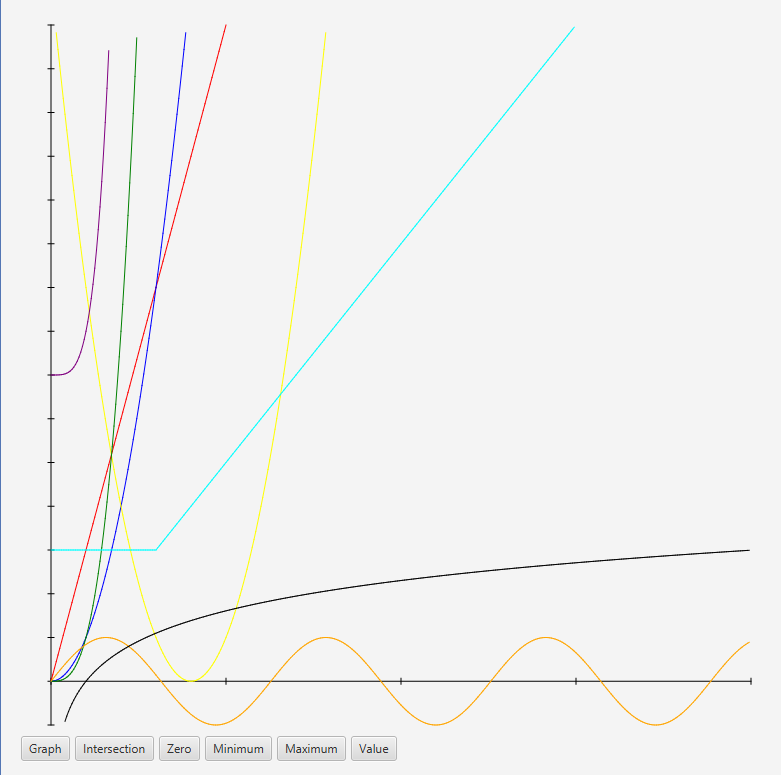
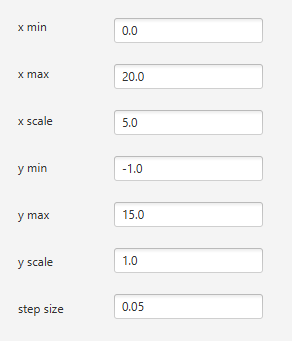
Y= Tab/Main Tab

****

Graph Tab Graphing

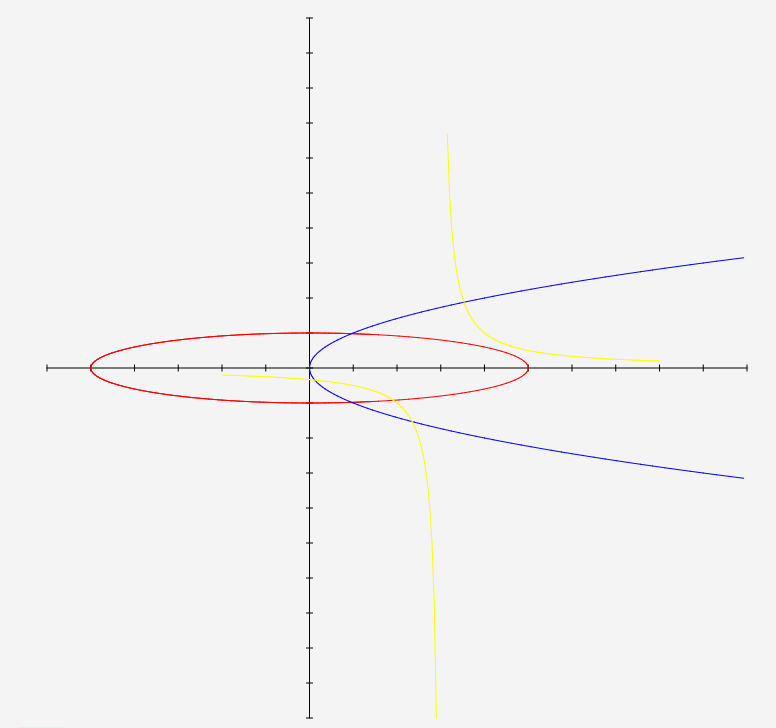
 (Graph of the equations from the last y= tab picture)

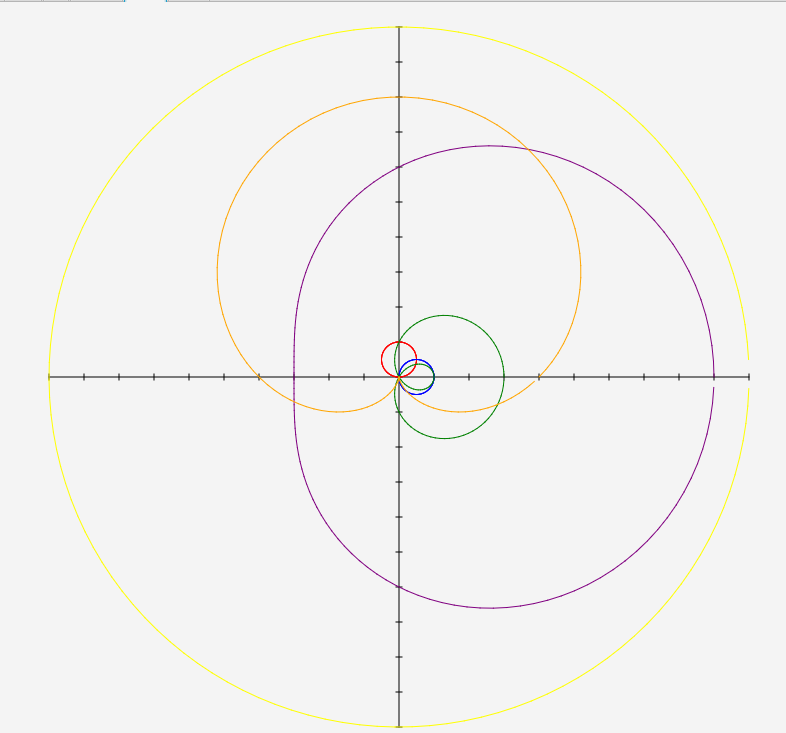
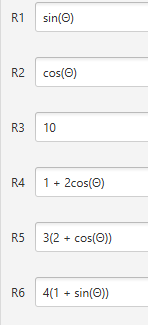
Graph Tab/Window Tab Changing window



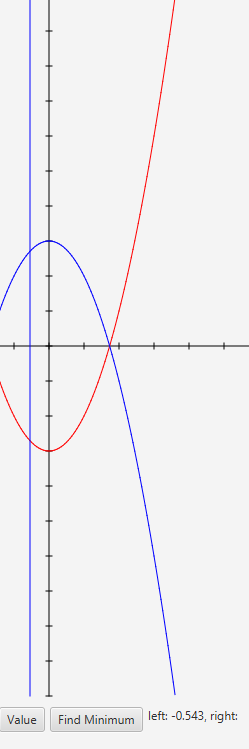
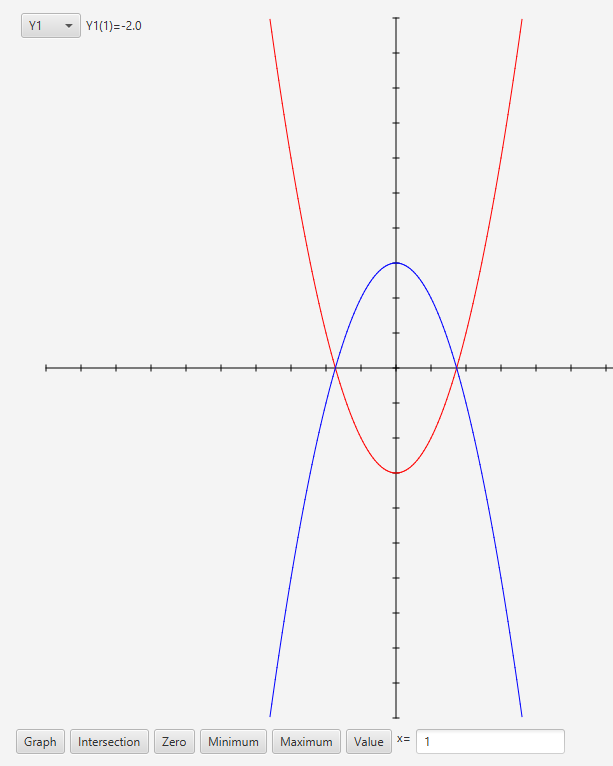
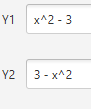
Graph Tab/y=Tab Changing Modes

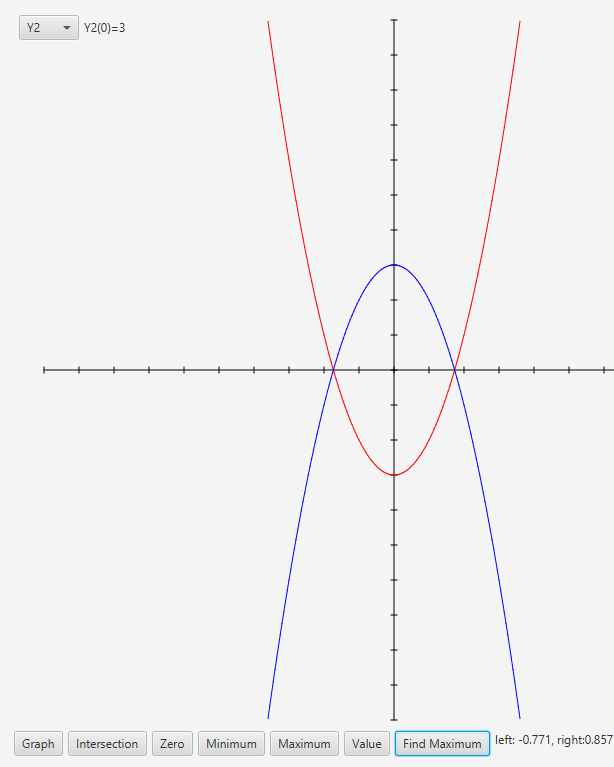
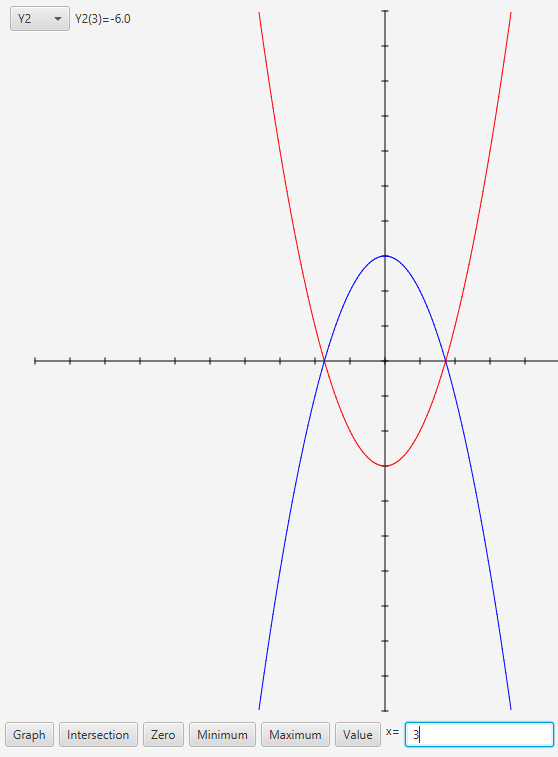


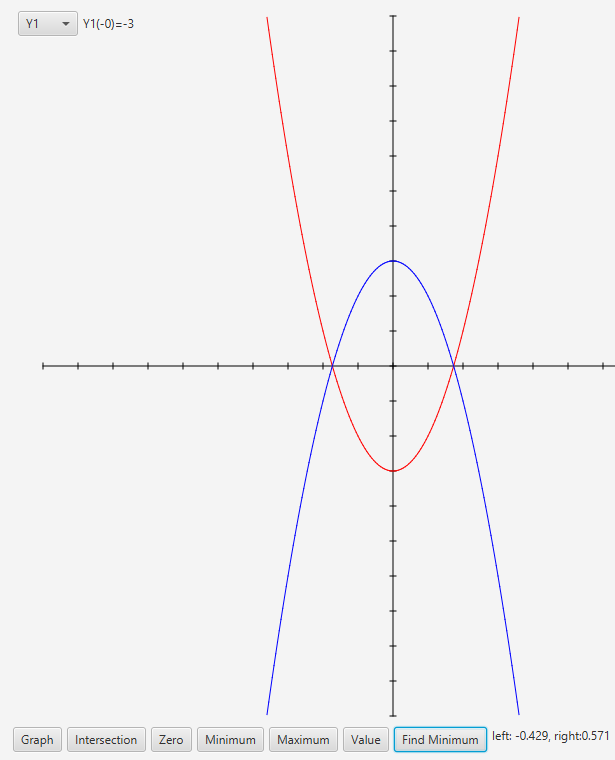




Graph Tab Features

**** (value and drawing bounds)

****

******(**Zeros)

<Y1 = x^2 (bouncing zero) (intersection)

**Conclusion**

There were many problems I had while completing this assignment. The first one I came across was that the String class’s split method did not work with “\*” or “+” as an argument, which I found out could be solved by instead doing split(“\\\*”). The next problem was that there was little differentiation between negatives and subtraction, and I solved this issue in a way that should probably be changed in the future, but as a temporary solution I required subtraction to have spaces around the minus sign. Another problem was that the split method did not work with the degree symbol, solving this simply required replacing the degree symbol with something that wouldn’t come up otherwise, which was not too difficult. Another issue I came across much later in the process was that when finding a zero by looking for a sign change, it cannot find zeros that simply touch the axis then bounce off. This is a big problem, but knowing that the algorithm I was using was very similar to the one a TI uses, I checked and TI’s actually have the same problem but no solution for it. I simply decided to implement a check for when it didn’t find a sign change after several tries that would then switch algorithms to one that simply looked for a low absolute value rather than a sign change. All of these issues and many more came up during the process but were not too difficult to fix. The most significant and difficult to fix issue however, is that some of the calculations done by java are imprecise, such as sin(pi) which is not 0.

In the future there are many features that could be added and things that could be fixed. One thing that I’ve already mentioned is that subtraction should work without spaces. Another is that the graphing tab should have functionality other than finding values in all 3 modes. Some more significant additions could be adding a statistics tab for storing data and calculating means, standard deviations, etc. and an ability to solve summations and products with a specific formula (sigma and pi notation). Another feature could be the ability to find derivatives and integrals on the graph tab and maybe even outside of it. All of these features and fixes could be added to the program in the future.