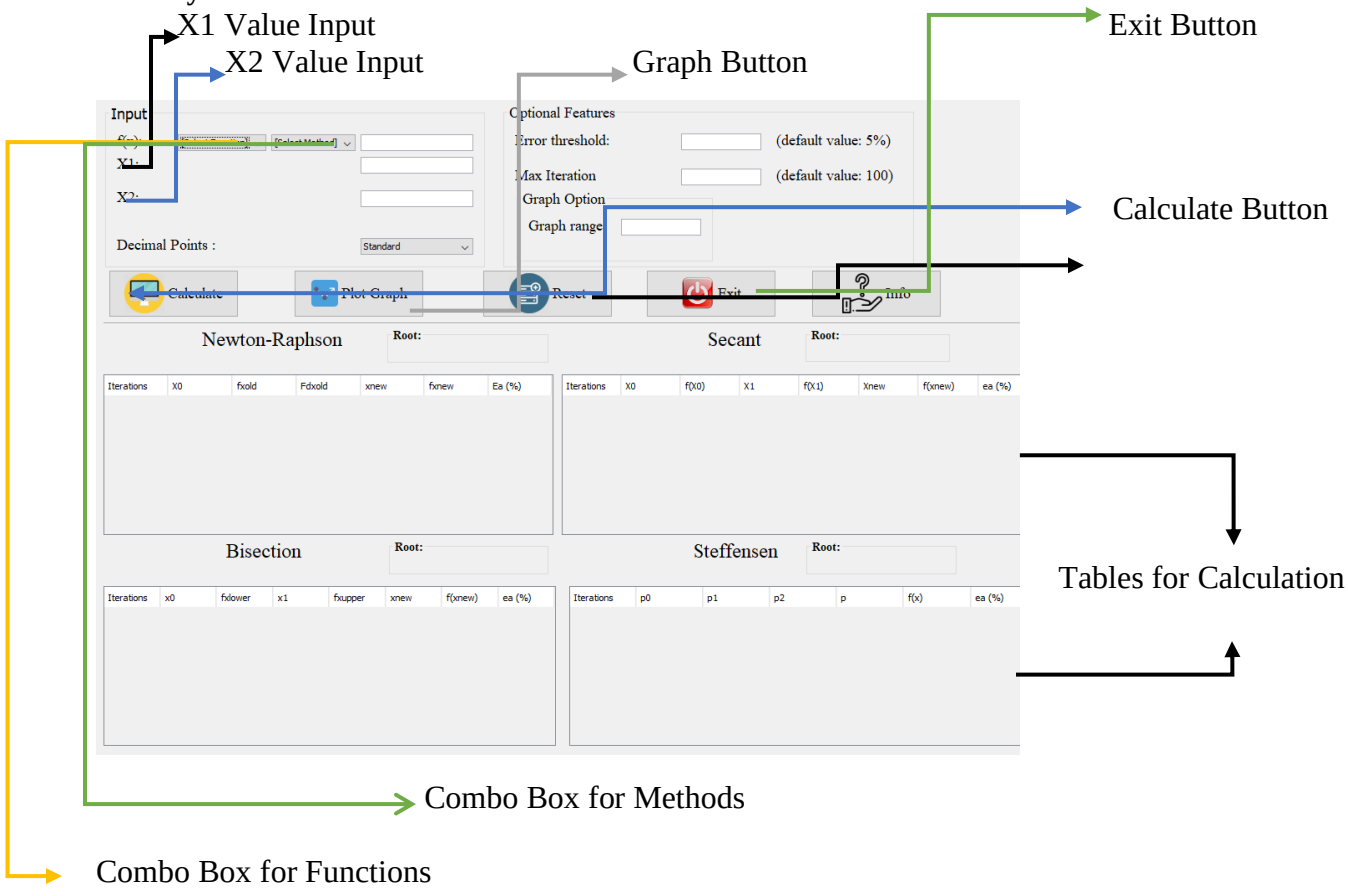
Table of Contents

A. User Documentation

The Layout:



Usability:

The program is simple to use, you are provided with the first two Combo Boxes which allow you to select a Function and a Method, the selected Function gets displayed in the Text Field to right.

In the two text fields, you are to input the choice of data for the X1 and X2, for convenience there are pre-set values for both fields to avoid typing in case of a test run. The options from both Combo Boxes are shown in Fig 1.2 and 1.1 below.

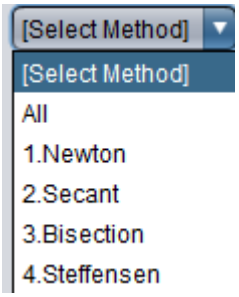


Fig 1.1

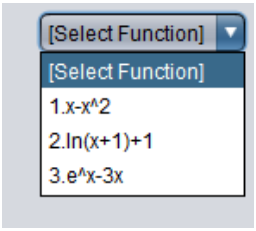


Fig 1.2

From there you can select a function and a method of your choice. There is a selection available "All" this allows you to calculate the roots using a function for all the methods Simultaneously. The final

root for convenience will be displayed on top each in Labels Fig 1.3. Furthermore, there is an additional Combo box for Decimal points Fig 1.4, allowing a set of up to 10 points to be selected.

Fig 1.3

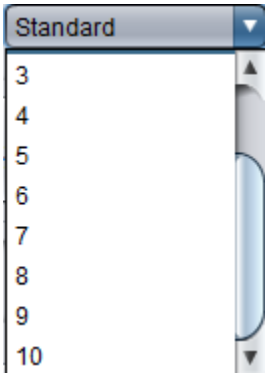


Fig 1.4

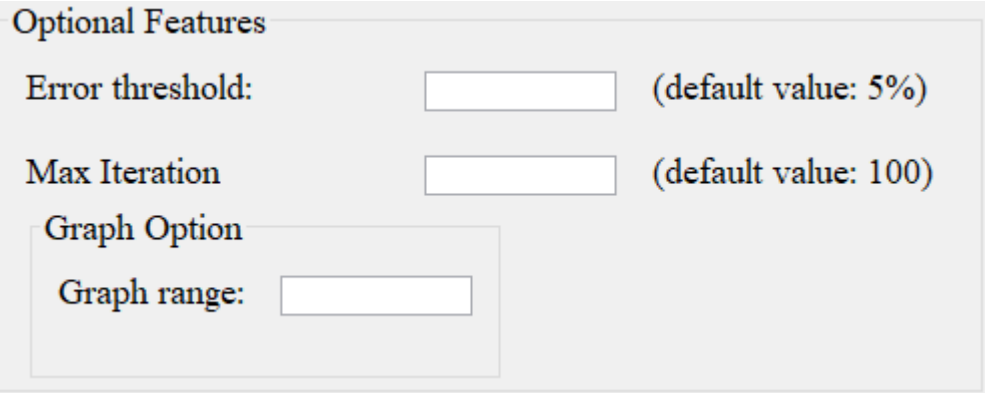


Fig 1.5

The figure above shows a panel with Additional features to alter a few settings in the code like extending the range of the graph, changing the max iteration etc.

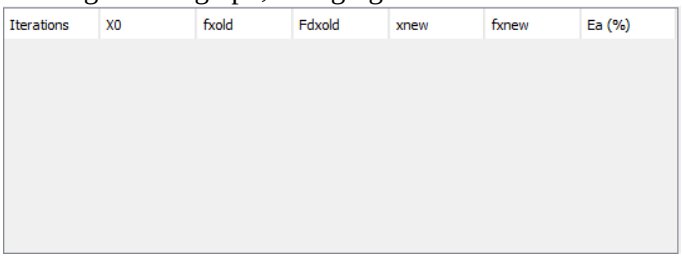


Fig 1.6

Figure above shows a table in which the data is set according to their variables like fxold, Fdxold etc., for each of the methods there are different tables set accordingly with their data rows and variables.

Fig 1.7

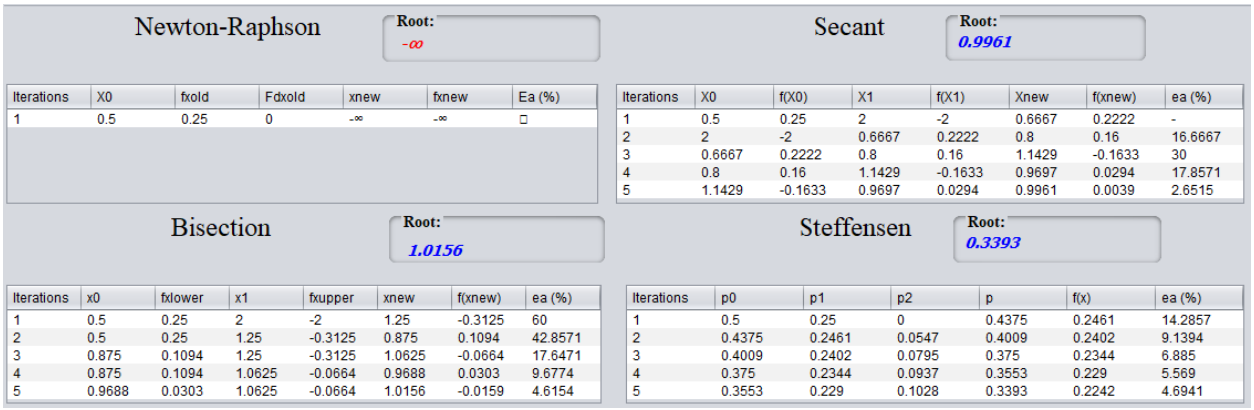
Above is the Panel row in which all the buttons are placed, their function is stated in their titles. For example, the Calculate button will display the roots in the table, Reset will set the Program back to its Default Settings etc.

B. Discussion for the choice of test data

To test the values in each of the numerical methods I will be choosing the "All" method to compare the values simultaneously. Each function was tested with a set of random values and their Tabular representation was presented.

For x-x^2

X VALUES	Method	Root Approximation
X1=0.5	Newton-Raphson	∞
X1=0.5 X2= 2	Secant	0.996
X1=0.5 X2= 2	Bisection	1.015
X1=0.5	Steffensen	0.3393



Newton-Raphson — Root: -∞

Iterations	X0	fxold	Fdxold	xnew	fxnew	Ea (%)
1	0.5	0.25	0	-∞	-∞	□

Secant — Root: 0.9961

Iterations	X0	f(X0)	X1	f(X1)	Xnew	f(xnew)	ea (%)
1	0.5	0.25	2	-2	0.6667	0.2222	-
2	2	-2	0.6667	0.2222	0.8	0.16	16.6667
3	0.6667	0.2222	0.8	0.16	1.1429	-0.1633	30
4	0.8	0.16	1.1429	-0.1633	0.9697	0.0294	17.8571
5	1.1429	-0.1633	0.9697	0.0294	0.9961	0.0039	2.6515

Bisection — Root: 1.0156

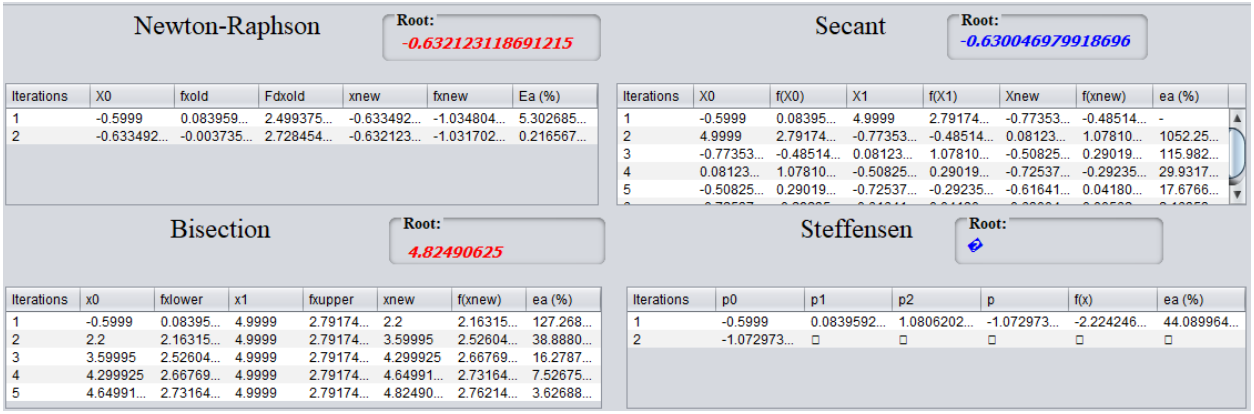
Iterations	x0	fxlower	x1	fxupper	xnew	f(xnew)	ea (%)
1	0.5	0.25	2	-2	1.25	-0.3125	60
2	0.5	0.25	1.25	-0.3125	0.875	0.1094	42.8571
3	0.875	0.1094	1.25	-0.3125	1.0625	-0.0664	17.6471
4	0.875	0.1094	1.0625	-0.0664	0.9688	0.0303	9.6774
5	0.9688	0.0303	1.0625	-0.0664	1.0156	-0.0159	4.6154

Steffensen — Root: 0.3393

Iterations	p0	p1	p2	p	f(x)	ea (%)
1	0.5	0.25	0	0.4375	0.2461	14.2857
2	0.4375	0.2461	0.0547	0.4009	0.2402	9.1394
3	0.4009	0.2402	0.0795	0.375	0.2344	6.885
4	0.375	0.2344	0.0937	0.3553	0.229	5.569
5	0.3553	0.229	0.1028	0.3393	0.2242	4.6941

For ln(x+1) +1

X VALUES	Method	Root Approximation
X1= -0.5999	Newton-Raphson	-0.632
X1= -0.5999 X2= 4.9999	Secant	-0.630
X1= -0.5999 X2= 4.9999	Bisection	4.824
X1= -0.5999	Steffensen	NaN



Newton-Raphson — Root: -0.632123118691215

Iterations	X0	fxold	Fdxold	xnew	fxnew	Ea (%)
1	-0.5999	0.083959...	2.499375...	-0.633492...	-1.034804...	5.302685...
2	-0.633492...	-0.003735...	2.728454...	-0.632123...	-1.031702...	0.216567...

Secant — Root: -0.630046979918696

Iterations	X0	f(X0)	X1	f(X1)	Xnew	f(xnew)	ea (%)
1	-0.5999	0.08395...	4.9999	2.79174...	-0.77353...	-0.48514...	-
2	4.9999	2.79174...	-0.77353...	-0.48514...	0.08123...	1.07810...	1052.25...
3	-0.77353...	-0.48514...	0.08123...	1.07810...	-0.50825...	0.29019...	115.982...
4	0.08123...	1.07810...	-0.50825...	0.29019...	-0.72537...	-0.29235...	29.9317...
5	-0.50825...	0.29019...	-0.72537...	-0.29235...	-0.61641...	0.04180...	17.6766...

Bisection — Root: 4.82490625

Iterations	x0	fxlower	x1	fxupper	xnew	f(xnew)	ea (%)
1	-0.5999	0.08395...	4.9999	2.79174...	2.2	2.16315...	127.268...
2	2.2	2.16315...	4.9999	2.79174...	3.59995	2.52604...	38.8880...
3	3.59995	2.52604...	4.9999	2.79174...	4.299925	2.66769...	16.2787...
4	4.299925	2.66769...	4.9999	2.79174...	4.64991...	2.73164...	7.52675...
5	4.64991...	2.73164...	4.9999	2.79174...	4.82490...	2.76214...	3.62688...

Steffensen — Root: ?

Iterations	p0	p1	p2	p	f(x)	ea (%)
1	-0.5999	0.0839592...	1.0806202...	-1.072973...	-2.224246...	44.089964...
2	-1.072973...	□	□	□	□	□

X VALUES	Method	Root Approximation
X1= 1.009	Newton-Raphson	0.6189
X1= 1.009 X2= -2.009	Secant	0
X1= 1.009 X2= -2.009	Bisection	Failed to converge in 100 Iterations
X1= 1.009	Steffensen	-0.027

Newton-Raphson **Root:** 0.618939325909863

Iterations	X0	fxold	Fdxold	xnew	fxnew	Ea (%)
1	1.009	-0.284143...	-0.257143...	-0.095999...	-0.105215...	1151.043...
2	-0.095999...	1.196463...	-2.091535...	0.476050...	0.249426...	120.1658...
3	0.476050...	0.181552...	-1.390295...	0.606636...	0.238628...	21.52621...
4	0.606636...	0.014342...	-1.165748...	0.618939...	0.235853...	1.987769...

Secant **Root:** 0

Iterations	X0	f(X0)	X1	f(X1)	Xnew	f(xnew)	ea (%)
1	1.009	-2.018	-2.009	8.036	0.40323...	-0.80647...	-
2	-2.009	8.036	0.403238...	-0.80647...	0.18323...	-0.36646...	120.071...
3	0.403238...	-0.80647...	0.183230...	-0.36646...	0	0	∞
4	0.183230...	-0.36646...	0	0	0	0	□

Bisection **Root:** Failed to converge in 100 iterations

Iterations	x0	fxlower	x1	fxupper	xnew	f(xnew)	ea (%)
1	1.009	-2.018	-2.009	8.036	-0.5	2	301.799...
2	1.009	-2.018	-0.5	2	0.2545	-0.509	296.463...
3	0.2545	-0.509	-0.5	2	-0.12275	0.491	307.331...
4	0.2545	-0.509	-0.12275	0.491	0.065875	-0.13175	286.337...
5	0.065875	-0.13175	-0.12275	0.491	-0.0284...	0.11375	331.648...

Steffensen **Root:** -0.027489493554114

Iterations	p0	p1	p2	p	f(x)	ea (%)
1	1.009	-0.009081	-1.027162	-0.027488...	-0.028244...	3770.5694...
2	-0.027488...	-0.028244...	-0.029000...	-0.027489...	-0.028245...	0.0020771...

X1= -1, 0 X2 = 2	Bisection	Failed to converge in 100 Iterations
X1= -1 ,0	Steffensen	-∞

Newton-Raphson **Root:** 0

Iterations	X0	fxold	Fdxold	xnew	fxnew	Ea (%)
1	-1	-2	3	-0.333333...	-0.444444...	199.9999...
2	-0.333333...	-0.444444...	1.666666...	-0.066666...	-0.071111...	400.0000...
3	-0.066666...	-0.071111...	1.133333...	-0.003921...	-0.003936...	1599.999...
4	-0.003921...	-0.003936...	1.007843...	-0.000015...	-0.000015...	25600.00...
5	-0.000015...	-0.000015...	1.000030...	-0.000000...	-0.000000...	6553599....
6	-0.000000...	-0.000000...	1.000000...	-0	-0	42949672...
7	-0	-0	1	0	0	∞
8	0	0	1	0	0	□

Secant **Root:**

Iterations	X0	f(X0)	X1	f(X1)	Xnew	f(xnew)	ea (%)
1	-1	-2	2	-2	-∞	-∞	-
2	2	-2	-∞	-∞	□	□	□

Bisection **Root:** Failed to converge in 100 iterations

Iterations	x0	fxlower	x1	fxupper	xnew	f(xnew)	ea (%)
1	-1	-2	2	-2	0.5	0.25	300
2	-1	-2	2	-2	0.5	0.25	300

Steffensen **Root:** -∞

Iterations	p0	p1	p2	p	f(x)	ea (%)
1	-1	-2	-3	-2	-6	50
2	-2	-6	-10	-18	-342	88.888888...

Bisection **Root:** 1.9375

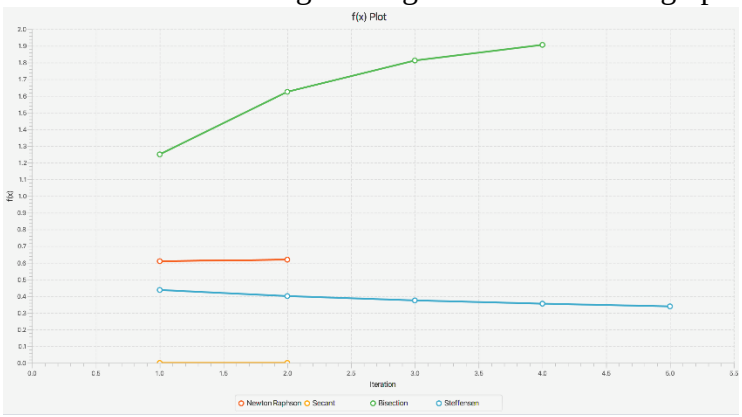
Iterations	x0	fxlower	x1	fxupper	xnew	f(xnew)	ea (%)
1	0	1	2	2.09861...	1	1.69314...	100
2	1	1.69314...	2	2.09861...	1.5	1.91629...	33.3333...
3	1.5	1.91629...	2	2.09861...	1.75	2.01160...	14.2857...
4	1.75	2.01160...	2	2.09861...	1.875	2.05605...	6.66666...
5	1.875	2.05605...	2	2.09861...	1.9375	2.07755...	3.22580...

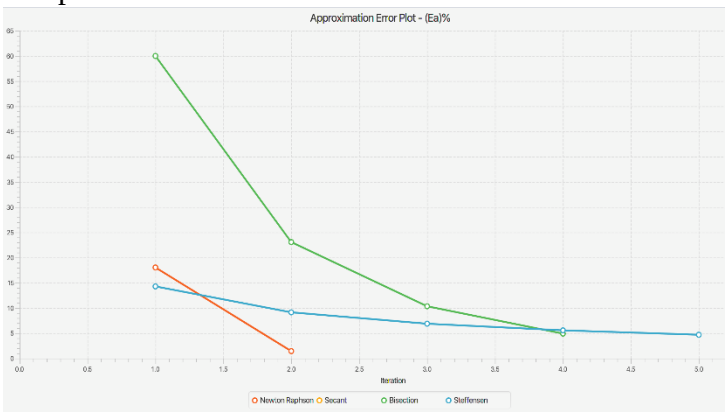
Steffensen **Root:** 2.303706876707812

Iterations	p0	p1	p2	p	f(x)	ea (%)
1	0	1	1.6931471...	1	0	100
2	1	1.6931471...	1.9907104...	1.5813808...	-0.919384...	36.764125...
3	1.5813808...	1.9483244...	2.0812370...	2.3195829...	-3.060882...	31.824776...
4	2.3195829...	2.1998391...	2.1631005...	2.3037068...	-3.003358...	0.6891522...

Graphs

The following two Figures below are the graphical representation of one of the tested tables.





Testing Exceptions

There has been an outmost attempt to reduce errors and add checks in this project, below are a few examples of they were tested.

1. No selection of both Combo Boxes Fig 1.8
2. No Selection of a Function Fig 1.9
3. No Selection of a Method Fig 2.0

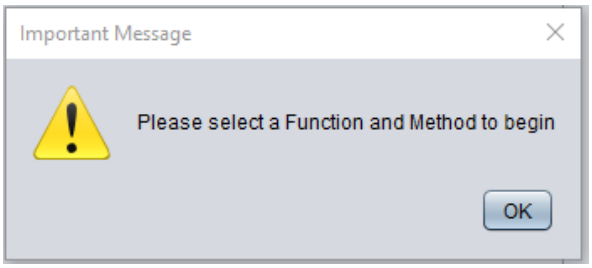


Fig 1.8

Fig 2.0

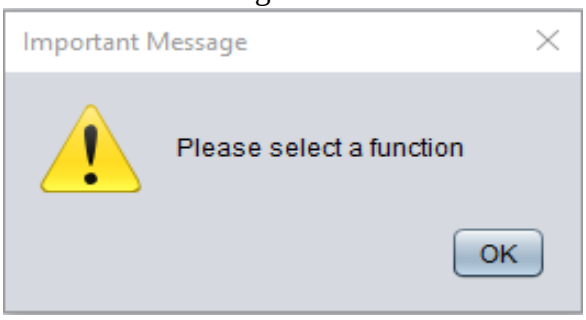


Fig 1.9

Non-Numerical Values

If either of the X values are not valid Fig 2.1

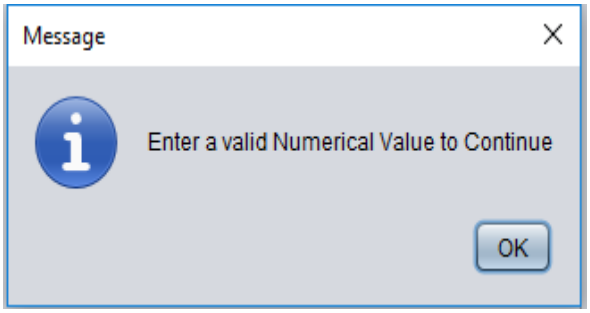


Fig 2.1

If any of the methods don't converge to the roots within the maximum iterations set.

Fig 2.2

C. Critical evaluation of the performance of the numerical algorithms when solving the three functions provided

This Java program, in many aspects is functioning properly with appropriate checks and catch exceptions, However, there are patches in code that bring limitations. In this section I will be specifically detailing those limitations and how the program could have been Improved.

1. Arrays and Linked Lists:

I was able to use arrays to store the values of each method and later on use them array list for the graphs, However, due to time and coordination management I was unable to utilise the LinkedList's, I realized there will be two linked lists required one to store the values and one to extract the stores values and convert them into arrays for Graph plotting. I have attempted one linkedList, but I have not used it the Newton Raphson method.

2. Graphs

Using the "All" was beneficial but also time consuming, since all the arrays are stored in the "list" array of the graphs, the graph will only be plotted if all the methods are called. The graphs will not work for individual methods as there was more programming involved in creating separate private void for each of the methods.

In general, the program functions as it is supposed to almost all the time, there are some functions which give some exceptional values like ∞ and NaN. I have done my utmost to add as many catch Exceptions to direct the user to what type of error is occurring. NaN is shown when the previous iteration had a negative value, for example in the function for ln(x+1) +1 almost all the methods were giving a NaN because the Log (0) will not go into a second iteration.

For the e^x-3x function many of the methods could not converge because even the root would be something like 0.00001 and so forth but still would not go infinite. There is an exception in which if the method continues to iterate more than the max number of iterations set, the message "Failed to converge" appears. Furthermore, if the calculated root is out of upper and lower limit set by the user, the calculated root is displayed in red colour.

All the methods nearly gave close roots to one another even with different equations, the odd one out was the Steffensen which didn't have the roots close to the other method, using online sources I was able to compute the mathematical equations into java compatible code. Unlike other methods I couldn't find an online calculator to verify the accuracy of the data, but I manually calculated the roots for Steffensen and all the functions except x-x^2. The roots don't alter much with different data values which is a bit sceptical.

For the first function, using the bisection when the x1 is set as a negative value the result gets no convergence until the 100^{th} iteration. Another limitation of the program is if the x1 value is bigger than x2 it will display an error for the user and reset the values. It might not be exactly a limitation, but it could've been improved by switching the x1 with x2.

In using the decimal rounding to give the root according the user selection, I was only able to have the point selection up to 10, this due to the decimal format function. I was unable to exceed it to more than 10, due to lack of programming knowledge in this field.

Conclusions including a reflection of the skills learned and bibliography

To sum it all, I can confidently say that working and building on this made me better at programming in java. Firstly, I used Java swing to build this application, I had no prior knowledge on using swing before, I had to invest days' worth of time into it to become proficient enough to use and implement in this Java CW. Furthermore, I was able to utilise the lecture notes to create properly functioning methods, with which I might have had difficulty otherwise.

Also, before this I didn't know how to use External APIs to use graphs and display data from the array list. This was quite challenging for me as there were limited sources online for using Javafx Charts in NetBeans and the current JDK. I also learned a lot about catch Exceptions and how the vitality of their placement is crucial because the same line of code placed somewhere different will not function as desired. Earlier to working on this project I didn't know how well combo boxes and text fields can be used to create a perfect block of code to avoid any errors or out of bound exceptions. All in All, this project was truly an eye opener for me has it had me invested for a few weeks and truly made me see out such a program consisting of 1000s of lines of code can be so simple, yet complexity lurks in its roots.

References

https://www.youtube.com/watch?v=GPuQsJdQ-WU – For Understanding Javafx Charts

https://www.youtube.com/watch?v=cAUHOKsFsn8- A tutorial for Javafx charts

http://tutorials.jenkov.com/java-internationalization/decimalformat.html- Java DecimalFormat

https://www.math.colostate.edu/~gerhard/MATH331/lab/newton.html- Basic Newton Method

https://stackoverflow.com/questions/20526917/load-arraylist-data-into-jtable- Array list to Jtable

https://stackoverflow.com/questions/8499698/trim-a-string-based-on-the-string-length- Decimal Format

https://www.youtube.com/watch?v=_ZW4ktG1DEE- Understanding how swing works

External API'S

Jxfrt.jar

Exp4j- This was for the expression building, later on not utilised