**Graphical Calculator NEA**

A graph of a graph

Description automatically generated

Alessia Codres

Contents

[Analysis 2](#_Toc150853519)

[Background to and Identification of the Problem 2](#_Toc150853520)

[Current Systems 3](#_Toc150853521)

[Desmos 3](#_Toc150853522)

[Casio fx-CG50 4](#_Toc150853523)

[Prospective Users and Requirements 5](#_Toc150853524)

[Shunting Yard Parser vs. Recursive Descent Parser 7](#_Toc150853525)

[Abstract Syntax Trees (ASTs) 8](#_Toc150853526)

[Components of the Recursive Descent Parser 9](#_Toc150853527)

[Data Volumes 10](#_Toc150853528)

[Acceptable Limitations 10](#_Toc150853529)

[Numbered Objectives of the Project 11](#_Toc150853530)

[Extension Objectives 12](#_Toc150853531)

[Proposed Method of Solution 12](#_Toc150853532)

[Design 13](#_Toc150853533)

[System Diagram 13](#_Toc150853534)

[System Flowchart 13](#_Toc150853535)

[Object Diagrams 15](#_Toc150853536)

[Data Structures 17](#_Toc150853537)

[Description of Complex Algorithms 18](#_Toc150853538)

[Recursive Descent Parser 18](#_Toc150853539)

[Tree Traversal – using recursion 19](#_Toc150853540)

[Cartesian and Pixel Coordinates Conversion 20](#_Toc150853541)

[Root Finding – using the Newton-Raphson Method 21](#_Toc150853542)

[Finding Maximum and Minimum Points 22](#_Toc150853543)

[UI Design 22](#_Toc150853544)

[System Security and Integrity of Data 25](#_Toc150853545)

[Technical Solution 26](#_Toc150853546)

[Technical Skills 26](#_Toc150853547)

[Program Code 26](#_Toc150853548)

[Parser.cs 26](#_Toc150853549)

[Function.cs 35](#_Toc150853550)

[Graph.cs 38](#_Toc150853551)

[Settings.cs 53](#_Toc150853552)

[Help.cs 57](#_Toc150853553)

[Testing 57](#_Toc150853554)

[Test Plan 57](#_Toc150853555)

[Evaluation 58](#_Toc150853556)

# Analysis

## Background to and Identification of the Problem

I am aiming to produce an offline graphical calculator for use by students and teachers in maths lessons. The software should plot and solve functions, and display important information such as maximum and/or minimum points, roots, and points of intersection.

As an A-Level Mathematics and Further Mathematics student, I was drawn to this concept, as it would not only be useful when solving problems and checking solutions, but as a project is interesting, as I will be able to create, and improve a system that my classmates and myself use often.

Currently, graphical calculators are not accessed by all students since they are expensive. Other systems do exist, but they are often containing an impractical user interface and lack certain features that would add to the usefulness of the system. As well as this, they require an internet connection, which may not always be available.

Despite this, it is obvious that graphical calculators are in demand from students, with many choosing to purchase one. They are valued for the extra convenience that they provide through additional features that regular scientific calculators do not provide. Therefore, creating an offline system, that is able to provide a user-friendly interface that has an adequate number of additional features would be highly beneficial.

## Current Systems

The most popular systems currently used are the Desmos graphing calculator and the Casio fx-CG50 calculator.

### Desmos

Desmos is a graphical calculator accessed online through a website.

Good features:

* Ability to display many graphs at once, and select which ones to be displayed. They are shown in different colours, to easily distinguish between them.
* Calculates values of interest – points of intersection, maximum, minimum points etc.
* Can easily zoom in and out, and drag to see different parts of the graph.
* Graphs drawn on a square grid for easy visuals and for a scale.
* Can plot point, functions and other graphs (including circles).

Areas of improvement:

* To see important values, the user must hover over them, which can be awkward, and impractical as not all values can be seen at once.

Chart, line chart

Description automatically generated

The user hovers the mouse to see points, e.g. the intersection between two graphs.

Desmos also allows the use of constants and constant changers in order to alter graphs drawn:

Chart

Description automatically generated with low confidence

Although this feature may be useful, it is not often that students make use of this, and it does not provide enough additional functionality to justify its implementation into this project. Therefore, at this stage, it remains a possible extension objective.

### Casio fx-CG50

Good features:

* Ability to display many graphs at once, and select which ones to be displayed. They are shown in different colours, to easily distinguish between them.
* Can get important values about the graphs plotted e.g., points of intersection, maximum and minimum points etc.
* Can zoom in and out, a grid being displayed for scale as the background when zoomed in enough.

Areas of improvement:

* When zooming in and out, the center point must be chosen, which can be more of a slow and awkward process compared to the dragging of the image Desmos offers.
* There is not a grid displayed for scale when zoomed out to a certain point.
* Many students may struggle to identify the features of the calculator and use them with ease due to the particular use of menus.
* Not all important values can be seen at once.

Diagram

Description automatically generated

User navigates menus to see values of interest, zoom in/out and make use of other features.

## Prospective Users and Requirements

After conducting a survey on A-Level Mathematics and Further Mathematics students’ experiences of graphical calculators, I found the following data about user experience:

1. Of the respondents that use the Casio fx-CG50, 68% said it was easy to navigate, but 84% said that it was difficult to initially get used to. This could be due to the heavy use of menus. The respondents that used Desmos all said it was both easy to navigate and easy to get used to. Therefore, implementing a more application-style user interface should provide the user with more ease of use.
2. Only 41% of respondents said they made use of constant changers, with some unaware of what they were. This means that the implementation of these may be useful for some, but is not a widespread need for users as a whole.
3. 100% of respondents said that they would find a system that does not require an internet connection useful. Therefore, I will be creating a software that can be used offline that should be more easily accessible to users.
4. Respondents identified the usefulness and necessity of various features:

Chart, bar chart

Description automatically generated with medium confidence

Overall, all features listed have relatively high demand, but points of intersection and other useful information being displayed were identified as most important. Therefore, it will be important to ensure that these features are implemented in a very user-friendly way.

It may prove difficult to computationally calculate points of intersection and turning points, as this would require working with the derivative, but this can remain an extension objective for now.

1. Other general comments included:
   1. “It would be helpful if each page had a small help window that would explain how to use the features. As currently I have to google how to do something.” – A help window can be included in the main menu that gives the user the option of a small tutorial, although this should hopefully not be required to a great extent, as the user interface should be simple to use.
   2. “It would also be useful if it could restrict graphs through domains and range.” – this is an additional feature that can be implemented as part of the functionality of the system.

## Shunting Yard Parser vs. Recursive Descent Parser

An important and challenging aspect of this project is validating and interpreting the mathematical expressions which are input by the user. There are two main approaches to do this: Shunting Yard and Recursive Descent.

The Shunting Yard is a bottom-up technique that parses expressions input in infix form into postfix or Reverse Polish Notation (RPN). It uses two stacks, reading the input from left to right. Recursive descent is a top-down technique that recursively breaks down the input into its smaller parts. This typically produces an Abstract Syntax Tree (AST).

Both methods have benefits and drawbacks, and are suited for different situations:

|  |  |
| --- | --- |
| Shunting Yard | |
| Benefits | Drawbacks |
| Can handle expressions with multiple levels of precedence and parentheses, efficiently. | Evaluating the postfix expression can be quite costly in terms of processing time and complexity. |
| Can be extended to handle functions and other non-operator tokens in the input. | Requires the use of two data structures – a stack and a queue. |
| Can be implemented using a stack and queue, which are quite simple as data structures. | Adds unnecessary steps to the conversion of simple expressions with low levels of precedence. |
| On more complex operations, parsing is easier and more efficient. | Can be more difficult to understand. |
| Recursive Descent |  |
| Benefits | Drawbacks |
| Relatively easy to implement and understand, specifically for simple expressions with low levels of precedence. | Can have issues with operator precedence and associativity – each function is called in a fixed order. |
| Can be used to generate code or ASTs, by each parsing function representing each sub-expression. | Can be more difficult to implement for complex expressions- each parsing function must be designed to handle the levels of precedence. |
| Efficient for small or medium sized expressions. | Parsing functions are tightly coupled to grammar, so may not be flexible. |
| Handles left-recursive grammars. | Inefficient on larger and more complex expressions, as many function calls are required. |

Although both methods are likely to handle the requirements of my system well, I have decided to implement recursive descent as my system is unlikely to have expressions of complexity high enough to invalidate its use. The grammar can also be defined to easily suit the requirement for non-left-recursion.

## Abstract Syntax Trees (ASTs)

An Abstract Syntax Tree is a data structure that represents the hierarchical structure of a mathematical expression to be evaluated and is generally the output of the Recursive Descent Parser.

ASTs are advantageous since they give a clearer representation and can be modified quite easily. This is because the relationships between nodes determine the order of operations.

ASTs are made of terminal and non-terminal nodes. To evaluate an expression, the nodes are visited from the bottom towards the top, with the operation in the non-terminal being performed on its child nodes. In this way, the correct order of operation is maintained as the nodes towards the top contain the operations which have the lowest precedence.

An example of an expression to be handled by the system is the cubic y = 3x^3 + 5x^2 + 4x - 6. When running recursive descent on this, a tree is created where the expression is first split up by the + and - operators, which have the lowest precedence. Then the \* and ^ operators.

An AST representation of this would be:

## Components of the Recursive Descent Parser

The building of the Recursive Descent Parser can be broken down into the following components:

1. A grammar – this defines the syntax of the language used, for example breaking down the expressions into variables, powers, factors, terms etc.
2. A lexer – this is used to break the input down into tokens, which are components of the expression that have been defined in the grammar, and to remove whitespace.
3. Recursive procedures – a set of procedures that are each responsible for recognising one syntactic element. These call each other recursively until the input has been completely converted into a tree.
4. Error handling – this is necessary to ensure that if invalid data is present in the input string, this can be dealt with efficiently.

## Data Volumes

The need for persistent storage of data is limited, with the only possible requirement being the settings the user has set. Therefore, the settings of the system should be saved, and updated only when prompted by the user, or if the user enters invalid settings. This would improve the user’s experience as they would not have to readjust the view window each time they open the application, for example.

## Acceptable Limitations

There are many features that already existing systems implement that would provide useful functionality, but it would not be realistic for me to implement these in the time frame I have.

Therefore, the system should only be expected to:

1. Plot functions on the real plane (no use of complex numbers).
2. Plot Cartesian functions in the form y = .
3. Plot 2D functions.
4. Plot up to 5 functions at a time.
5. Accept the view window preferences to be within -100 and 100.
6. Plot exponential, logarithmic, linear, trigonometric functions, and polynomials, or a combination of these.
7. Use angles in the form of radians and/or degrees.
8. Calculate and display roots and y-intercepts.
9. Give the user the ability to view individual coordinates by tracing the graph.
10. Allow the user to zoom in/out and move up/down/left/right.

I have chosen to only use radians and degrees as angle types because these are the only two that are used by students in the A-Level Mathematics and Further Mathematics course, with gradians having very little use in general.

If extension objectives are met, then the system should also be able to calculate and display turning points and intersection points, as well as cope with domains, and calculate and display the range.

## Numbered Objectives of the Project

1. The Graph form is displayed.
   1. There will be the buttons “Help”, “Settings”, “Exit” displayed.
2. If the “Help” button is pressed, a help form opens providing a brief explanation of how to use the application.
3. If the “Settings” button is pressed, a Settings form should load that allows the user to enter preferences.
   1. Relevant text boxes/combo boxes should be present allowing the user to input information about their preferences.
      1. Text boxes for “Max Y”, “Max X”, “Min X”, “Min Y”.
      2. Combo box for “Angle Type”, users can select between radians and degrees.
   2. A “Close Settings” button is available to be pressed to save the information.
      1. If data is invalid, an error message is displayed.
         1. Invalid if view window settings empty.
         2. Invalid if view window settings equal.
         3. Invalid if view window settings outside of range (-100 and 100).
         4. Invalid if max values less than min.
      2. If all data is valid, the settings are saved to a text file.
      3. If the settings form is forced closed, the default settings are saved.
4. The Graph form will allow the user to input up to 5 functions to be graphed.
   1. Each function has a tick box next to it, allowing the function to be selected by the user to be graphed.
      1. Once selected, validation of the data input occurs.
         1. Invalid if letters other than x or e used (pi can also be used to denote π).
         2. Invalid if functions other than “sin”, “cos”, “tan”, “ln”, “e” used.
         3. Invalid if characters other than “+”, “-“, “/”, “\*”, “^”, “(“, “)” used.
         4. Invalid if double operators used, except in the case of “– “.
         5. Invalid if unpaired brackets used.
      2. If any invalid data entered, an appropriate message is displayed.
      3. If all data is valid, the function should be accepted. The Recursive Descent algorithm should be run on selected functions to create an AST, which is then evaluated in appropriate increments of x values.
5. The functions will be drawn in the grid box on the “Graph” form.
   1. A square grid with a correctly scaled set of axes appears, based on the settings selected by the user.
   2. Functions selected plotted based on coordinates obtained from the AST evaluation.
   3. Different graphs should be plotted in different colours.
   4. Linear functions should be correctly plotted.
   5. Polynomials should be correctly plotted.
   6. Exponential and logarithmic functions should be correctly plotted.
   7. Trigonometric functions should be correctly plotted.
6. Information about the selected functions is correctly obtained and stored.
   1. Roots correctly obtained and stored.
   2. Intercepts correctly obtained and stored.
   3. Roots and intercepts are displayed in list boxes below each function.
7. The form should allow the user to navigate the graph.
   1. The user should be able to zoom in.
   2. The user should be able to zoom out.
   3. The user should be able to move left, right, up, and down.
   4. The user should be able to view coordinates that they select on the graphs.
8. The “Exit” button can be pressed at any time to close the application.

## Extension Objectives

1. Allow for inputs of domains for each function and display the ranges.
2. Work out the derivative of the functions and use them to calculate and display turning points.
3. Find points of intersection of functions plotted together and display them.

## Proposed Method of Solution

I will be using Visual Studio for this project, programming in C#, as this is the language I am most comfortable with. Additionally, I will be using a Windows Forms Application, which I believe to be appropriate as users should find this easy to navigate.

# Design

## System Diagram

The application has 3 forms: Graph, Settings and Help:

* Upon loading the application, the Graph form is loaded – everything is centered around this form.
* The Help form can be accessed from the Graph form and the Settings form.
* The Settings form can be accessed from the Graph form.
* An Exit button on the Graph form allows the application to close.
* Close buttons are available on the Settings and Help forms to close the respective forms.

A black text on a white background

Description automatically generated

## System Flowchart

The main flowchart of the user’s navigation through the application is shown below:

A diagram of a process

Description automatically generated

## Object Diagrams

A screenshot of a graph

Description automatically generated

A screenshot of a computer program

Description automatically generated

## Data Structures

As discussed in the Analysis section, I will be using trees (Abstract Syntax Trees) to store the parsed expression that is entered by the user. These will be the result of running the Recursive Descent parser. The trees will be implemented by use of a node class:

class Node

{

public Node left = null;

public Node right = null;

public string value = "";

public Node(Node Left, Node Right, string Value)

{

left = Left;

right = Right;

value = Value;

}

}

The Node data type allows the tree to be built up from individual node units. The **left** and **right** Node types are pointers to the child nodes, and the **value** string allows for the respective token to be stored in the node.

A lexer will be used to break the expression down into tokens.

enum TokenType

{

LParen,

RParen,

Operator1,

Operator2,

Exp,

Trig,

Log,

Num,

Var,

Const,

End,

Error

}

The TokenType data type is used to assign tokens their type. The allowed types are any components that can make up the expressions the system can handle, as well as an **End** and **Error** type. These are used to allow the parser to understand when the end of an expression is reached or when an error has been encountered.

class Token

{

public TokenType type;

public string value;

public Token(TokenType Type, string Value)

{

type = Type;

value = Value;

}

}

The Token type is used to define a token. **type** is assigned to denote the type of token that it is, using one of the enumerated data types. **value** is used to store the actual value of the token. For example, **type** may store **Number** and **value** may store 35.

Once a function is input and selected to plot, an object is instantiated to store the function and its details.

class Function

{

private string expression = "";

private double yintercept = 0;

private PointF[] CartPoints = new PointF[5000];

private PointF[] PixPoints = new PointF[5000];

private List<string> roots = new List<string>();

private List<PointF> min = new List<PointF>();

private List<PointF> max = new List<PointF>();

private List<double> gradients = new List<double>();

}

The function itself is stored in **expression**. The calculated Cartesian coordinates are stored in **CartPoints** and their conversions to pixel coordinates are stored in **PixPoints**. Geometric information such as y-intercept, roots, turning points etc. are stored in their respective lists/variables.

Throughout the system, there is also use of arrays and lists. Arrays will be used when the number of items to be stored can be predetermined, for example for storing the calculated coordinates of the graph to be plotted (as seen above). Lists will be used when the number of items to be stored is unknown, for example when storing roots.

## Description of Complex Algorithms

### Recursive Descent Parser

The Recursive Descent Parser uses recursion in order to convert an expression, which is input by the user as a string, into a tree.

The parser contains a lexer, which will split the expression into tokens, that are fetched by the parser one by one, and a tree is built from the bottom up, preserving the correct order of operations. The tree will be made from nodes, which are implemented as a user-defined data type.

The three main recursive procedures are **Expression(), Term(), and Factor().** These call each other recursively depending on the current token. For example, the **Expression()** method will separate the expression by operators of precedence 2 (+ and -), and call **Term()** to separate the terms on either side of the operator. The **Term()** method will separate terms by operators of second precedence (\* and /), and call **Factor()** to separate the factors on either side of the operator. The **Factor()** method will separate factors into the individual tokens (numbers, variables, functions, brackets, powers).

If an error occurs (user inputs invalid expression), error is set to true and a node with the error as the value will be returned, so that the parser can stop parsing and return the error to be output to the user. This is seen in the if statements in each method.

The recursive procedures can be seen in Parser class in the Parser.cs file of the technical solution.

### Tree Traversal – using recursion

Tree Traversal is used to evaluate a tree. An x value is taken as a parameter (alongside the root node of the tree returned by the parser) and the tree is traversed, systematically carrying out each operation, eventually returning a y value. This is done by a method **Evaluate(Node node, string x).**

Initially, the root is checked to see whether it is the only node in the tree – this is the base case. Then the lowest non-leaf node on the left (grammar is left recursive) is found by recursively calling the function using nodes on the left as parameters. If there are more children nodes on the right, the function can be called recursively again, this time to the right. This allows this root to be evaluated by determining the operator in the root node, and applying this to the children nodes, using the x value passed in as a parameter if necessary. This node’s value is then replaced with the evaluated value.

This allows the nodes above this to evaluate upwards, as each non-leaf node is replaced with the result of the operation that was represented as each function call is evaluated.

The **Evaluate(Node node, string x)** method can be seen in Parser class in the Parser.cs file of the technical solution.

### Cartesian and Pixel Coordinates Conversion

In order to graph a function, the system must be able to convert between the cartesian coordinates that will appear on the screen to the equivalent pixel coordinates that can be used to draw the graph correctly.

Initially, an x-factor and y-factor is determined. This allows the magnitude of the cartesian coordinates to be scaled up or down approapriately into pixel coordiantes. Next, the x-origin and y-origin of the view window is determined. This is then converted into pixel coordinates by multiplying with the xfactor or the yfactor. In order to do this correctly, the 9 possible view window cases must be considered:

1. Origin is in view window (both x and y axis visible)
2. Only positive x axis visible
3. Only negative x axis visible
4. Only positive y axis visible
5. Only negative y axis visible
6. In 1st quadrant (no axes visible)
7. In 2nd quadrant (no axes visible)
8. In 3rd quadrant (no axes visible)
9. In 4th quadrant (no axes visible)

These possible cases and their outcomes can be seen in the **SetOffset()** function.

Then the conversion can take place using the **CartToPix(float x, float y)** function. The formula used can be seen below with an explanation of the steps taken:

A black and white text

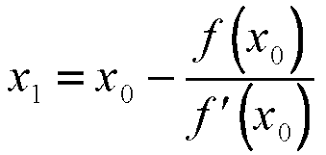
Description automatically generated

In order to convert from pixel to cartesian coordinates, the same formula can be used, but rearraged to have cartesian x and y as the subject.

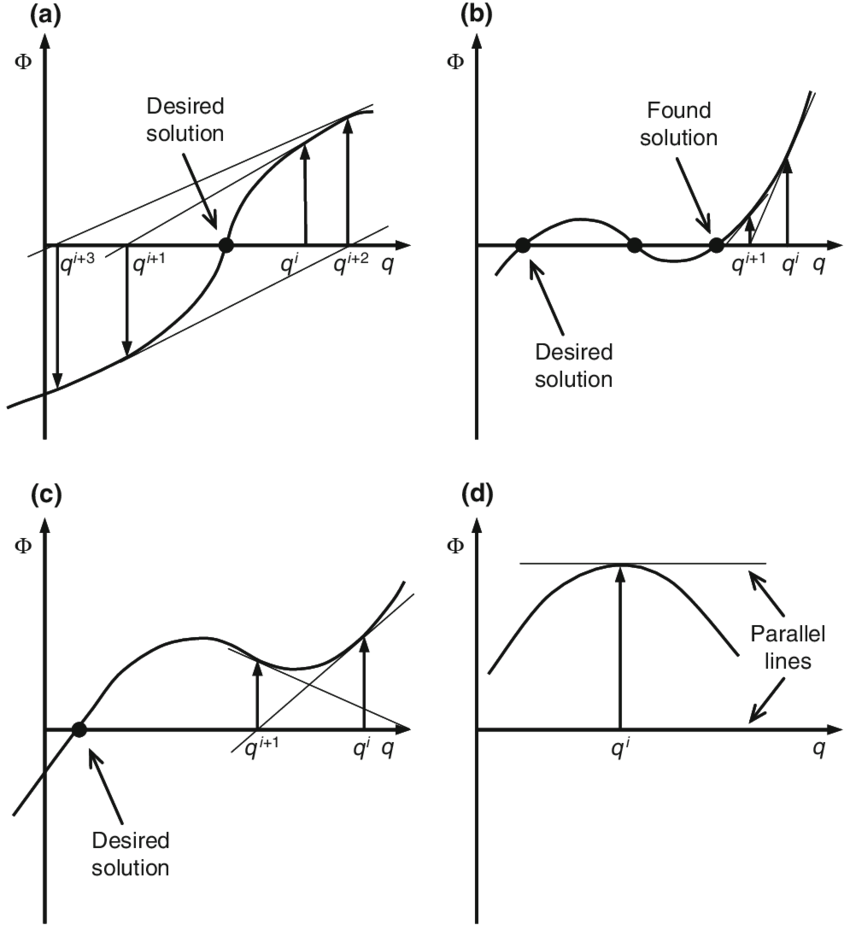
**CartToPix(float x, float y), PixToCart(float x, float y)** and **SetOffset()** can be found in the Graph class in the Graph.cs file.

### Root Finding – using the Newton-Raphson Method

In order to find the roots of functions computationally, a numerical method must be used as opposed to an algebraic method. After attempting various methods, I found the Newton-Raphson method to be the most accurate. The formula for the method can be seen below:



The implementation of this is seen in **NewtonRaphson(float x, int index)**. The x value passed into the function is important, as an inaproapriate value can lead to the method failing. This is because the method works by drawing a tangent on the graph at this point and taking its x intercept as the new estimation for the root. A few eamples of this can be seen below:



Generally, the closer the x value passed into the function is to the root, the more accurate the approximation is. For this reason, the x value to be passed into the function is determined by checking for a sign change in the y cooridnates, and that the y coordinates are close to 0 (this will exclude non-continuous functions, where there is a sign change, but the graph does not actually cross the x-axis i.e. there is an asymptote). As well as this, if a coordinate has a y value that is stored to be 0, this is also treated as a root, in order to take into consideration minimum points that touch the x-axis. In this case there is no sign change, so otherwise they would be lost. However, to ensure a root is present, a check for an asymptote must be done since if the x axis is an asymptote, the y value will be stored to be 0.

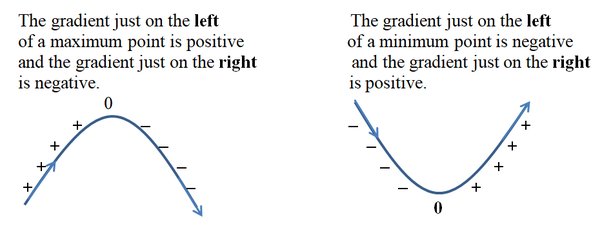
I have found that because the initial x value passed into the function is very close to the root, the method only requires one iteration to give a good approximation.

After finding roots, a check is done to ensure no roots are stored twice – this can occur if multiple y-values are very close to 0 for example.

**NewtonRaphson(double x, int index), CheckAsymptote(List <PointF> points, float MaxY, float MinY, float MaxX, float MinX)** and **FindRoots()** implement this and can be found in the Function.cs file.

### Finding Maximum and Minimum Points

Finding maximum and minimum points makes use of the **gradients** array. This is an array of the calculated gradients between any pairs of adjacent coordinates. Where a maximum or minimum point occurs, the gradient will go from positive to negative or vice versa.



To find these points, the gradients array is searched for an approapriate sign change, once one is found, there must be a maximum or minimum point. The only exception to this is where there is an asymptote, so **CheckAsymptote(List <PointF> points, float MaxY, float MinY, float MaxX, float MinX)** must be used to do this.

## FindMaxPoints(float MaxY, float MinY, float MaxX, float MinX), FindMaxPoints(float MaxY, float MinY, float MaxX, float MinX) and CheckAsymptote(List <PointF> points, float MaxY, float MinY, float MaxX, float MinX) can be found in the Function.cs file.

## UI Design

Graph Form

The user can click these arrows to move the view window along, up, or down.

The cursor options allow the user to zoom in and out of the graph as they wish by using the mouse pointer to select a centre.

A screenshot of a computer

Description automatically generated

Each function has:

* A textbox where the user can type in the function.
* A tick box to allow the user to select if the function is to be plotted.
* A colour dialog to allow the user to select what colour the function will be plotted in.
* A list box, where geometric information about the function will be displayed.

Settings and Help buttons allow the user to navigate to the appropriate forms. Exit button allows the user to close the application.

This picture box is where the graphs will be drawn.

Settings Form

Settings Form

A screenshot of a computer settings

Description automatically generated

The Help button allows the user to navigate to the Help window.

The angle type is selected by the user. Options are degrees and radians.

Settings form closes if settings valid, and settings are saved.

The user can type the max and min x and y values, which must be valid.

Help Form

A screenshot of a computer

Description automatically generated

The Help Form is not interactive, and simply informs the user on the basics of the system.

## System Security and Integrity of Data

Since my project does not involve the use of any sensitive data, I have not chosen to implement any security measures such as password. However, I have focused on preventing invalid data from being entered into the system, as well as ensuring no data becomes corrupt during execution.

Throughout the system, I have ensured to include error handling to prevent incorrect data being processed, causing the system to malfunction or crash. I have also attempted to carefully consider any methods used are functional for all cases. For example, when calculating roots of functions, I took care to eliminate any asymptotes that can computationally indicate a root may be present.

When the user inputs any data, validation always takes place to ensure data is valid. This occurs when the settings input by the user are to be saved. If invalid, the user is made to change them, otherwise the Settings form cannot close. However, if the form is force-closed, the default settings are saved instead. A second instance of this is when functions are input. These are taken through validation to prevent any errors from occurring when they are to be evaluated. For example, bracket pairs are checked.

Settings are saved in a text file. I have implemented exception handling to ensure that if the file cannot be found because it does not exist, the system does not crash, and instead a new file is created. Since the file is saved in the project’s binary files, this is unlikely to become corrupted, only being accessed by the program itself in a secure way.

# Technical Solution

## Technical Skills

The table below summarises the technical skills I have used and where they can be found:

|  |  |  |
| --- | --- | --- |
| Group | Technical Skill | Found |
| A | Trees – Graph/Tree Traversal | See Parser.cs file - an expression tree is created and traversed. |
| Lists – List Operations | See use of Min, Max, and Roots lists in the Function.cs file. |
| Complex scientific/mathematical model – recursive algorithms, complex user-defined algorithms | See Design – description of complex algorithms |
| Complex user-defined use of object-orientated programming (OOP) model | See the use of classes and objects to create user-defined data types (e.g. Node, Function, Token).  See interfaces used to ensure encapsulation and modular programming |
| B | Text Files – writing and reading from files | See Settings.txt text file used in the Settings.cs file. |
| Multidimensional Arrrays | See use of CartPoints and PixPoints arrays throughout. |
| C | Appropriate choice of simple data types | See use of variables throughout. |

## Program Code

The solution is divided into the following 5 files: Parser.cs, Function.cs, Graph.cs, Settings.cs and Help.cs.

### Parser.cs

class Node

{

public Node left = null;

public Node right = null;

public string value = "";

public Node(Node Left, Node Right, string Value)

{

left = Left;

right = Right;

value = Value;

}

}

enum TokenType

{

LParen,

RParen,

Operator1,

Operator2,

Exp,

Trig,

Log,

Num,

Var,

Const,

End,

Error

}

class Token

{

public TokenType type;

public string value;

public Token(TokenType Type, string Value)

{

type = Type;

value = Value;

}

}

class Lexer

{

private char currchar;

private Token currtok = new Token(TokenType.LParen, null);

private int pos;

private string input;

public Lexer(string Input)

{

input = Input;

pos = -1;

input = RemoveWhitespace(input);

input = Negatives(input);

CheckBracketPairs();

}

private string RemoveWhitespace(string input)

{

string temp = input;

input = "";

for (int i = 0; i < temp.Length; i++)

{

if (temp[i] != ' ')

{

input = input + temp[i].ToString();

}

}

return input;

}

private string Negatives(string input)

{

if (input[0] == '-')

{

input = "0" + input;

}

for (int i = 1; i < input.Length; i++)

{

if (input[i - 1] == '-' && input[i] == '-')

{

input = input.Substring(0, i - 1) + "+" + input.Substring(i + 1);

}

}

return input;

}

public void CheckBracketPairs()

{

int LCount = 0;

int RCount = 0;

for (int i = 0; i < input.Length; i++)

{

if (input[i] == '(')

{

LCount++;

}

if (input[i] == ')')

{

RCount++;

}

}

if (LCount != RCount)

{

currtok.value = "ERROR: Bracket pair missing.";

currtok.type = TokenType.Error;

currchar = '!';

}

}

private void Next()

{

pos++;

if (pos == input.Length)

{

currchar = ';';

}

else

{

currchar = input[pos];

}

}

private void CheckEnd()

{

if (pos + 1 == input.Length)

{

currtok.type = TokenType.Error;

currtok.value = "ERROR: Expression unrecognised.";

}

}

public Token GetNextToken()

{

while (currchar != '!')

{

TokenType type\_ = currtok.type;

Next();

if (char.IsDigit(currchar) || currchar == '.')

{

currtok = GetNumber();

}

else if (char.IsLetter(currchar))

{

currtok = GetWord();

}

else if (currchar == '(')

{

currtok = new Token(TokenType.LParen, "(");

CheckEnd();

}

else if (currchar == ')')

{

currtok = new Token(TokenType.RParen, ")");

}

else if (currchar == '^')

{

currtok = new Token(TokenType.Exp, "^");

CheckEnd();

}

else if (currchar == '\*' || currchar == '/')

{

currtok = new Token(TokenType.Operator1, currchar.ToString());

CheckEnd();

}

else if (currchar == '+' || currchar == '-')

{

currtok = new Token(TokenType.Operator2, currchar.ToString());

CheckEnd();

}

else if (currchar == ';')

{

currtok = new Token(TokenType.End, ";");

}

else

{

currtok = new Token(TokenType.Error, currchar.ToString());

}

if ((type\_ == TokenType.Num || type\_ == TokenType.Var || type\_ == TokenType.Const || type\_ == TokenType.RParen) &&

(currtok.type == TokenType.Var || currtok.value == "e" || currtok.type == TokenType.LParen))

{

currtok = new Token(TokenType.Operator1, "\*");

pos--;

}

if ((type\_ == TokenType.Num || type\_ == TokenType.Var || type\_ == TokenType.Const || type\_ == TokenType.RParen) &&

(currtok.type == TokenType.Trig))

{

currtok = new Token(TokenType.Operator1, "\*");

pos -= 3;

}

if ((type\_ == TokenType.Num || type\_ == TokenType.Var || type\_ == TokenType.Const || type\_ == TokenType.RParen) &&

(currtok.value == "pi" || currtok.type == TokenType.Log))

{

currtok = new Token(TokenType.Operator1, "\*");

pos -= 2;

}

if (currtok.type == TokenType.Error)

{

if (currtok.value != "ERROR: Expression unrecognised.")

{

currtok.value = "ERROR: " + currtok.value + " not recognised.";

}

}

return currtok;

}

return currtok;

}

private Token GetNumber()

{

string num = Convert.ToString(currchar);

if (pos == input.Length - 1)

{

return new Token(TokenType.Num, num);

}

else

{

while (pos + 1 < input.Length)

{

if (input[pos + 1] == '.' || char.IsDigit(input[pos + 1]))

{

Next();

if (currchar == '!')

{

break;

}

num = num + Convert.ToString(currchar);

}

else

{

return new Token(TokenType.Num, num);

}

}

}

return new Token(TokenType.Num, num);

}

private Token GetWord()

{

string word = Convert.ToString(currchar);

while (pos + 1 < input.Length)

{

if (char.IsLetter(input[pos + 1]))

{

Next();

if (currchar == '!')

{

break;

}

word = word + Convert.ToString(currchar);

}

else

{

break;

}

}

if (word == "sin" || word == "cos" || word == "tan")

{

return new Token(TokenType.Trig, word);

}

if (word == "x")

{

return new Token(TokenType.Var, word);

}

if (word == "e")

{

return new Token(TokenType.Const, Convert.ToString(Math.Exp(1)));

}

if (word == "pi")

{

return new Token(TokenType.Const, Convert.ToString(Math.PI));

}

if (word == "ln")

{

return new Token(TokenType.Log, word);

}

return new Token(TokenType.Error, word);

}

}

internal class Parser

{

public Node root = new Node(null, null, null);

private Node node = new Node(null, null, null);

private Token currtok;

private Lexer lexer;

private bool error = false;

public Parser(string input)

{

lexer = new Lexer(input);

currtok = lexer.GetNextToken();

root = Expression();

}

private void CheckBracket()

{

if (currtok.type == TokenType.LParen)

{

return;

}

else

{

currtok.value = "ERROR: Bracket expected.";

currtok.type = TokenType.Error;

}

}

private void Consume()

{

currtok = lexer.GetNextToken();

}

private Node Expression()

{

Node node = Term();

while (currtok.type == TokenType.Operator2)

{

Node temp = new Node(node, null, currtok.value);

Consume();

temp.right = Term();

if (error == true)

{

node = temp.right;

}

else

{

node = temp;

}

}

return node;

}

private Node Term()

{

Node node = Factor();

while (currtok.type == TokenType.Operator1)

{

Node temp = new Node(node, null, currtok.value);

Consume();

temp.right = Factor();

if (error == true)

{

node = temp.right;

}

else

{

node = temp;

}

}

return node;

}

private Node Factor()

{

while (currtok.type == TokenType.Num || currtok.type == TokenType.Var || currtok.type == TokenType.Trig || currtok.type == TokenType.Log

|| currtok.type == TokenType.Exp || currtok.type == TokenType.Const || currtok.type == TokenType.LParen || currtok.type == TokenType.Error)

{

if (currtok.type == TokenType.Num || currtok.type == TokenType.Var || currtok.type == TokenType.Const)

{

node = new Node(null, null, currtok.value);

Consume();

}

if (currtok.type == TokenType.Trig)

{

Node temp = new Node(null, null, currtok.value);

Consume();

CheckBracket();

temp.left = Factor();

temp.right = temp.left;

if (error == true)

{

node = temp.left;

}

else

{

node = temp;

}

node = temp;

}

if (currtok.type == TokenType.Log)

{

Node temp = new Node(null, null, currtok.value);

Consume();

CheckBracket();

temp.left = Factor();

temp.right = temp.left;

if (error == true)

{

node = temp.left;

}

else

{

node = temp;

}

node = temp;

}

if (currtok.type == TokenType.Exp)

{

Node temp = new Node(node, null, currtok.value);

Consume();

CheckBracket();

temp.right = Factor();

if (error == true)

{

node = temp.right;

}

else

{

node = temp;

}

}

if (currtok.type == TokenType.LParen)

{

Consume();

node = Expression();

Consume();

}

if (currtok.type == TokenType.Error)

{

node = new Node(null, null, currtok.value);

error = true;

currtok.type = TokenType.End;

}

}

return node;

}

public Node Evaluate(Node node, string x)

{

if (error == true)

{

return node;

}

if (node.left == null && node.right == null)

{

if (node.value == "x")

{

node.value = x;

}

return node;

}

string left = "";

if (node.left != null)

{

while (node.left.left != null)

{

node.left = Evaluate(node.left, x);

}

left = node.left.value;

}

if (left == "x")

{

left = x;

}

string right = "";

if (node.right != null)

{

while (node.right.right != null)

{

node.right = Evaluate(node.right, x);

}

right = node.right.value;

}

if (right == "x")

{

right = x;

}

node.right = null;

node.left = null;

switch (node.value)

{

case "+":

node.value = Convert.ToString(Convert.ToDouble(left) + Convert.ToDouble(right));

return node;

case "-":

node.value = Convert.ToString(Convert.ToDouble(left) - Convert.ToDouble(right));

return node;

case "\*":

node.value = Convert.ToString(Convert.ToDouble(left) \* Convert.ToDouble(right));

return node;

case "/":

node.value = Convert.ToString(Convert.ToDouble(left) / Convert.ToDouble(right));

return node;

case "^":

node.value = Convert.ToString(Math.Pow(Convert.ToDouble(left), Convert.ToDouble(right)));

return node;

case "cos":

node.value = Convert.ToString(Math.Cos(Convert.ToDouble(left)));

return node;

case "sin":

node.value = Convert.ToString(Math.Sin(Convert.ToDouble(left)));

return node;

case "tan":

node.value = Convert.ToString(Math.Tan(Convert.ToDouble(left)));

return node;

case "ln":

node.value = Convert.ToString(Math.Log(Convert.ToDouble(left)));

return node;

}

return node;

}

}

### Function.cs

internal class Function

{

public string expression = "";

public double yintercept = 0;

public PointF[] CartPoints = new PointF[5000];

public PointF[] PixPoints = new PointF[5000];

public List<string> roots = new List<string>();

public List<PointF> min = new List<PointF>();

public List<PointF> max = new List<PointF>();

public List<double> gradients = new List<double>();

public void GetYIntercept(Function function)

{

Parser parser = new Parser(function.expression);

function.yintercept = Math.Round(Convert.ToDouble(parser.Evaluate(parser.root, Convert.ToString(0)).value), 2);

}

public string NewtonRaphson(Function function, double x, int index)

{

Parser parser = new Parser(function.expression);

double derivative = (function.CartPoints[index + 1].Y - function.CartPoints[index].Y) / (function.CartPoints[index + 1].X - function.CartPoints[index].X);

double y = Convert.ToDouble(parser.Evaluate(parser.root, Convert.ToString(x)).value);

x = Math.Round(x - (y / derivative), 2);

if (x == -0)

{

return "x = " + Convert.ToString(0);

}

return "x = " + Convert.ToString(x);

}

public void GetRoots(Function function)

{

function.roots.Clear();

string root;

int count = 0;

for (int i = 0; i < function.CartPoints.Length - 1; i++)

{

if (((function.CartPoints[i].Y \* function.CartPoints[i + 1].Y < 0) && Math.Abs(function.CartPoints[i].Y) < 1) || function.CartPoints[i].Y == 0)

{

for (int j = i - 40; j < i + 40; j++)

{

if (j >= 0 && j < CartPoints.Length)

{

if (Math.Abs(function.CartPoints[j].Y) < 0.005)

{

count++;

}

}

}

if (count < 20 && function.CartPoints[i].Y != 0)

{

root = NewtonRaphson(function, function.CartPoints[i].X, i);

if (!function.roots.Contains(root))

{

function.roots.Add(root);

}

}

if (count < 20 && function.CartPoints[i].Y == 0)

{

root = "x = " + Math.Round(function.CartPoints[i].X, 2);

if (root == "x = -0")

{

root = "x = 0";

}

if (!function.roots.Contains(root))

{

function.roots.Add(root);

}

}

}

}

}

private bool CheckAsymptote(List<PointF> points, float MaxY, float MinY)

{

bool asymptote = false;

for (int i = 0; i < points.Count; i++)

{

if (points[i].Y > MaxY || points[i].Y < MinY)

{

asymptote = true;

return asymptote;

}

}

return asymptote;

}

public void GetMaxPoints(Function function, float MaxY, float MinY)

{

function.max.Clear();

double temp = 0;

int index = 0;

for (int i = 0; i < function.gradients.Count; i++)

{

if (function.gradients[i] > 0)

{

temp = function.gradients[i];

index = i + 1;

}

if (function.gradients[i] < 0 && temp > 0)

{

double x = Math.Round((function.CartPoints[index].X + function.CartPoints[i + 1].X) / 2, 1);

if (x == -0)

{

x = 0;

}

double y = Math.Round(function.CartPoints[i - 1].Y, 1);

PointF point = new PointF(Convert.ToSingle(x), Convert.ToSingle(y));

if (point.Y == 0)

{

function.roots.Add("x = " + point.X);

}

function.max.Add(point);

temp = 0;

index = 0;

}

}

bool asymptote = CheckAsymptote(function.max, MaxY, MinY);

if (asymptote)

{

function.max.Clear();

}

}

public void GetMinPoints(Function function, float MaxY, float MinY)

{

function.min.Clear();

double temp = 0;

int index = 0;

for (int i = 0; i < function.gradients.Count; i++)

{

if (function.gradients[i] < 0)

{

temp = function.gradients[i];

index = i + 1;

}

if (function.gradients[i] > 0 && temp < 0)

{

double x = Math.Round((function.CartPoints[index].X + function.CartPoints[i + 1].X) / 2, 1);

if (x == -0)

{

x = 0;

}

double y = Math.Round(function.CartPoints[i - 1].Y, 1);

PointF point = new PointF(Convert.ToSingle(x), Convert.ToSingle(y));

if (point.Y == 0)

{

function.roots.Add("x = " + point.X);

}

function.min.Add(point);

temp = 0;

index = 0;

}

}

bool asymptote = CheckAsymptote(function.min, MaxY, MinY);

if (asymptote)

{

function.min.Clear();

}

}

public void GetGradients(Function function)

{

function.gradients.Clear();

double m;

for (int i = 0; i < function.CartPoints.Length - 1; i++)

{

m = (function.CartPoints[i + 1].Y - function.CartPoints[i].Y) / (function.CartPoints[i + 1].X - function.CartPoints[i].X);

if (!double.IsNaN(m) && !double.IsInfinity(m))

{

function.gradients.Add(m);

}

}

}

}

### Graph.cs

public partial class Graph : Form

{

Function function1 = new Function();

Function function2 = new Function();

Function function3 = new Function();

Function function4 = new Function();

Function function5 = new Function();

float xoffset;

float yoffset;

float xorigin;

float yorigin;

float MinX;

float MaxX;

float MinY;

float MaxY;

float xfactor;

float yfactor;

float xaxisPos = 0;

float yaxisPos = 0;

Graphics graphObj;

Parser parser;

string cursor = "default";

bool error = false;

public Graph()

{

InitializeComponent();

MinimumSize = new Size(800, 700);

if (!File.Exists("Settings.txt"))

{

Settings settings = new Settings();

settings.InitialiseSettings();

}

}

private void SetOffset()

{

if ((MinX >= 0 && MinY >= 0) || (MinX >= 0 && MaxY <= 0) || (MaxX <= 0 && MaxY <= 0) || (MaxX <= 0 && MinY >= 0))

{

xoffset = 0;

yoffset = 0;

xorigin = MinX;

yorigin = MaxY;

}

if (MinX <= 0 && MaxX >= 0 && MinY <= 0 && MaxY >= 0)

{

xoffset = -MinX \* xfactor;

yoffset = MaxY \* yfactor;

xorigin = 0;

yorigin = 0;

}

if ((MinX >= 0 && MinY <= 0 && MaxY >= 0) || (MaxX <= 0 && MinY <= 0 && MaxY >= 0))

{

xoffset = 0;

yoffset = MaxY \* yfactor;

xorigin = MinX;

yorigin = 0;

}

if ((MinX <= 0 && MaxX >= 0 && MinY >= 0) || (MinX <= 0 && MaxX >= 0 && MaxY <= 0))

{

xoffset = -MinX \* xfactor;

yoffset = 0;

xorigin = 0;

yorigin = MaxY;

}

}

private PointF CartToPix(float x, float y)

{

PointF pix = new PointF(0, 0);

pix.X = xoffset + ((x - xorigin) \* xfactor);

pix.Y = yoffset + ((yorigin - y) \* yfactor);

return pix;

}

private PointF PixToCart(float x, float y)

{

PointF cart = new PointF(0, 0);

cart.X = ((x - xoffset) / xfactor) + xorigin;

cart.Y = yorigin - ((y - yoffset) / yfactor);

return cart;

}

private void GetPointArrays(Function function)

{

ReadSettings();

SetOffset();

float step = (MaxX - MinX) / 5000;

for (int i = 0; i < function.CartPoints.Length; i++)

{

function.CartPoints[i].X = MinX + step \* i;

}

for (int i = 0; i < function.CartPoints.Length; i++)

{

try

{

Parser parser = new Parser(function.expression);

string y = Convert.ToString(Math.Round(Convert.ToDouble(parser.Evaluate(parser.root, Convert.ToString(function.CartPoints[i].X)).value), 3));

function.CartPoints[i].Y = (float)Convert.ToDouble(y);

}

catch

{

Parser parser = new Parser(function.expression);

MessageBox.Show(parser.Evaluate(parser.root, Convert.ToString(function.CartPoints[i].X)).value);

slctFunc1.Checked = false;

slctFunc2.Checked = false;

slctFunc3.Checked = false;

slctFunc4.Checked = false;

slctFunc5.Checked = false;

error = true;

break;

}

function.PixPoints[i].Y = CartToPix(function.CartPoints[i].X, function.CartPoints[i].Y).Y;

function.PixPoints[i].X = CartToPix(function.CartPoints[i].X, function.CartPoints[i].Y).X;

}

}

private void DrawAxes(float xgap, float ygap)

{

Pen blackpen = new Pen(Color.Black, 2);

if (MinX <= 0 && MaxX >= 0)

{

graphObj.DrawLine(blackpen, -MinX \* xfactor, 0, -MinX \* xfactor, pbGraph.Height);

yaxisPos = -MinX \* xfactor;

}

else if (MaxX <= 0)

{

yaxisPos = pbGraph.Width;

}

else if (MinX >= 0)

{

yaxisPos = 0;

}

if (MinY <= 0 && MaxY >= 0)

{

graphObj.DrawLine(blackpen, 0, MaxY \* yfactor, pbGraph.Width, MaxY \* yfactor);

xaxisPos = MaxY \* yfactor;

}

else if (MaxY <= 0)

{

xaxisPos = 0;

}

else if (MinY >= 0)

{

xaxisPos = pbGraph.Height;

}

if (MinX < 0 && MaxX > 0 && MinY < 0 && MaxY > 0)

{

PointF point = new PointF(yaxisPos - 10, xaxisPos);

Brush brush = new SolidBrush(Color.Black);

graphObj.DrawString("0", new Font("Segoe UI", 8, FontStyle.Regular), brush, point);

}

for (float i = 5 \* xgap; i < MaxX; i = i + 5 \* xgap)

{

if (i > MinX)

{

RectangleF rectangle;

float x = CartToPix(i, 0).X;

string str = Convert.ToString(i);

Font font = new Font("Segoe UI", 8, FontStyle.Regular);

SizeF stringSize = new SizeF(graphObj.MeasureString(str, font));

if (xaxisPos != pbGraph.Height)

{

rectangle = new RectangleF(x - (stringSize.Width / 2), xaxisPos, stringSize.Width, stringSize.Height);

}

else

{

rectangle = new RectangleF(x - (stringSize.Width / 2), xaxisPos - stringSize.Height - 2, stringSize.Width, stringSize.Height);

}

Brush brush = new SolidBrush(Color.Black);

graphObj.DrawString(str, font, brush, rectangle);

}

}

for (float i = -5 \* xgap; i > MinX; i = i - 5 \* xgap)

{

if (i < MaxX)

{

RectangleF rectangle;

float x = CartToPix(i, 0).X;

string str = Convert.ToString(i);

Font font = new Font("Segoe UI", 8, FontStyle.Regular);

SizeF stringSize = new SizeF(graphObj.MeasureString(str, font));

if (xaxisPos != pbGraph.Height)

{

rectangle = new RectangleF(x - (stringSize.Width / 2), xaxisPos, stringSize.Width, stringSize.Height);

}

else

{

rectangle = new RectangleF(x - (stringSize.Width / 2), xaxisPos - stringSize.Height - 2, stringSize.Width, stringSize.Height);

}

Brush brush = new SolidBrush(Color.Black);

graphObj.DrawString(str, font, brush, rectangle);

}

}

for (float i = 5 \* ygap; i < MaxY; i = i + 5 \* ygap)

{

if (i > MinY)

{

RectangleF rectangle;

float y = CartToPix(0, i).Y;

string str = Convert.ToString(i);

Font font = new Font("Segoe UI", 8, FontStyle.Regular);

SizeF stringSize = new SizeF(graphObj.MeasureString(str, font));

if (yaxisPos != 0)

{

rectangle = new RectangleF(yaxisPos - stringSize.Width - 2, y - 5, stringSize.Width, stringSize.Height);

}

else

{

rectangle = new RectangleF(yaxisPos, y - 5, stringSize.Width, stringSize.Height);

}

Brush brush = new SolidBrush(Color.Black);

graphObj.DrawString(str, font, brush, rectangle);

}

}

for (float i = -5 \* ygap; i > MinY; i = i - 5 \* ygap)

{

if (i < MaxY)

{

RectangleF rectangle;

float y = CartToPix(0, i).Y;

string str = Convert.ToString(i);

Font font = new Font("Segoe UI", 8, FontStyle.Regular);

SizeF stringSize = new SizeF(graphObj.MeasureString(str, font));

if (yaxisPos != 0)

{

rectangle = new RectangleF(yaxisPos - stringSize.Width - 2, y - 5, stringSize.Width, stringSize.Height);

}

else

{

rectangle = new RectangleF(yaxisPos, y - 5, stringSize.Width, stringSize.Height);

}

Brush brush = new SolidBrush(Color.Black);

graphObj.DrawString(str, font, brush, rectangle);

}

}

}

private void DrawGrid()

{

Pen greypen = new Pen(Color.Lavender, 1);

Pen bluepen = new Pen(Color.LightBlue, 1);

float[] gaps = new float[] { 0.01f, 0.05f, 0.1f, 0.2f, 0.5f, 1, 2, 5 };

float xgap = gaps[0];

int track = 0;

while ((MaxX - MinX) / xgap > 100)

{

if (track < 7)

{

track++;

xgap = gaps[track];

}

else

{

xgap += 5;

}

}

track = 0;

float ygap = gaps[0];

while ((MaxY - MinY) / ygap > 100)

{

if (track < 7)

{

track++;

ygap = gaps[track];

}

else

{

ygap += 5;

}

}

for (float i = 0; i < MaxX; i = i + xgap)

{

if (i > MinX)

{

float x = CartToPix(i, 0).X;

graphObj.DrawLine(greypen, x, 0, x, pbGraph.Height);

}

}

for (float i = 0; i > MinX; i = i - xgap)

{

if (i < MaxX)

{

float x = CartToPix(i, 0).X;

graphObj.DrawLine(greypen, x, 0, x, pbGraph.Height);

}

}

for (float i = 0; i < MaxX; i = i + 5 \* xgap)

{

if (i > MinX)

{

float x = CartToPix(i, 0).X;

graphObj.DrawLine(bluepen, x, 0, x, pbGraph.Height);

}

}

for (float i = -5 \* xgap; i > MinX; i = i - 5 \* xgap)

{

if (i < MaxX)

{

float x = CartToPix(i, 0).X;

graphObj.DrawLine(bluepen, x, 0, x, pbGraph.Height);

}

}

for (float i = 0; i < MaxY; i = i + ygap)

{

if (i > MinY)

{

float y = CartToPix(0, i).Y;

graphObj.DrawLine(greypen, 0, y, pbGraph.Width, y);

}

}

for (float i = 0; i > MinY; i = i - ygap)

{

if (i < MaxY)

{

float y = CartToPix(0, i).Y;

graphObj.DrawLine(greypen, 0, y, pbGraph.Width, y);

}

}

for (float i = 5 \* ygap; i < MaxY; i = i + 5 \* ygap)

{

if (i > MinY)

{

float y = CartToPix(0, i).Y;

graphObj.DrawLine(bluepen, 0, y, pbGraph.Width, y);

}

}

for (float i = -5 \* ygap; i > MinY; i = i - 5 \* ygap)

{

if (i < MaxY)

{

float y = CartToPix(0, i).Y;

graphObj.DrawLine(bluepen, 0, y, pbGraph.Width, y);

}

}

DrawAxes(xgap, ygap);

}

private void ReadSettings()

{

StreamReader reader = new StreamReader("Settings.txt");

MinX = float.Parse(reader.ReadLine());

MaxX = float.Parse(reader.ReadLine());

MinY = float.Parse(reader.ReadLine());

MaxY = float.Parse(reader.ReadLine());

reader.Close();

xfactor = pbGraph.Width / (MaxX - MinX);

yfactor = pbGraph.Height / (MaxY - MinY);

}

private void WriteSettings()

{

StreamWriter writer = new StreamWriter("Settings.txt");

writer.WriteLine(MinX);

writer.WriteLine(MaxX);

writer.WriteLine(MinY);

writer.WriteLine(MaxY);

writer.Close();

xfactor = pbGraph.Width / (MaxX - MinX);

yfactor = pbGraph.Height / (MaxY - MinY);

}

private void DrawGraph(Pen pen, PointF[] pixPoints)

{

if (error != true)

{

DrawGrid();

try

{

for (int i = 0; i < pixPoints.Length - 1; i++)

{

if (!double.IsNaN(pixPoints[i].Y) && !double.IsNaN(pixPoints[i + 1].Y) && !double.IsInfinity(pixPoints[i].Y) && !double.IsInfinity(pixPoints[i + 1].Y)

&& pixPoints[i].Y > 0 && pixPoints[i].Y > 0)

{

graphObj.DrawLine(pen, pixPoints[i], pixPoints[i + 1]);

}

}

}

catch (OverflowException)

{

}

}

}

private void DisplayInfo(ListBox lb, Function function)

{

lb.Items.Clear();

if (error != true)

{

if (!double.IsInfinity(function.yintercept))

{

lb.Items.Add("Y-Intercept: " + function.yintercept);

}

if (function.min.Count > 0 && function.max.Count > 0)

{

for (int i = function.max.Count - 1; i < function.max.Count; i++)

{

for (int j = function.min.Count - 1; j < function.min.Count; j++)

{

if (function.max[i].X == function.min[j].X)

{

function.min.RemoveAt(i);

function.max.RemoveAt(i);

}

}

}

}

if (function.max.Count != 0)

{

if (function.max.Count > 20)

{

lb.Items.Add("There are too many maximum points to display.");

}

else

{

lb.Items.Add("Maximum Points:");

for (int i = 0; i < function.max.Count; i++)

{

lb.Items.Add("(" + function.max[i].X + ", " + function.max[i].Y + ")");

}

}

}

if (function.min.Count != 0)

{

if (function.min.Count > 20)

{

lb.Items.Add("There are too many minimum points to display.");

}

else

{

lb.Items.Add("Minimum Points:");

for (int i = 0; i < function.min.Count; i++)

{

lb.Items.Add("(" + function.min[i].X + ", " + function.min[i].Y + ")");

}

}

}

if (function.roots.Count > 20)

{

lb.Items.Add("There are too many roots to display.");

}

if (function.roots.Count != 0)

{

lb.Items.Add("Roots:");

for (int i = 0; i < function.roots.Count; i++)

{

lb.Items.Add(function.roots[i]);

}

}

}

}

private void CheckCoord(PointF[] pixPoints, PointF[] cartPoints, Point pixPoint)

{

for (int i = 0; i < pixPoints.Length; i++)

{

if (Math.Abs(Math.Abs(pixPoint.X) - Math.Abs(pixPoints[i].X)) < 5 &&

Math.Abs(Math.Abs(pixPoint.Y) - Math.Abs(pixPoints[i].Y)) < 5)

{

string x = Convert.ToString(Math.Round(cartPoints[i].X, 3));

string y = Convert.ToString(Math.Round(cartPoints[i].Y, 3));

lbCoords.Text = "(" + x + ", " + y + ")";

break;

}

}

}

public void FuncCheck()

{

error = false;

graphObj = pbGraph.CreateGraphics();

graphObj.Clear(Color.White);

ReadSettings();

SetOffset();

if (slctFunc1.Checked)

{

function1.expression = txtFunc1.Text;

UpdateFunction(function1);

Pen pen = new Pen(Func1Colour.BackColor);

DrawGraph(pen, function1.PixPoints);

DisplayInfo(lbFunc1Info, function1);

}

if (slctFunc2.Checked)

{

function2.expression = txtFunc2.Text;

UpdateFunction(function2);

Pen pen = new Pen(Func2Colour.BackColor);

DrawGraph(pen, function2.PixPoints);

DisplayInfo(lbFunc2Info, function2);

}

if (slctFunc3.Checked)

{

function3.expression = txtFunc3.Text;

UpdateFunction(function3);

Pen pen = new Pen(Func3Colour.BackColor);

DrawGraph(pen, function3.PixPoints);

DisplayInfo(lbFunc3Info, function3);

}

if (slctFunc4.Checked)

{

function4.expression = txtFunc4.Text;

UpdateFunction(function4);

Pen pen = new Pen(Func4Colour.BackColor);

DrawGraph(pen, function4.PixPoints);

DisplayInfo(lbFunc4Info, function4);

}

if (slctFunc5.Checked)

{

function5.expression = txtFunc5.Text;

UpdateFunction(function5);

Pen pen = new Pen(Func5Colour.BackColor);

DrawGraph(pen, function5.PixPoints);

DisplayInfo(lbFunc5Info, function5);

}

}

private void UpdateFunction(Function function)

{

GetPointArrays(function);

if (!error)

{

function.GetYIntercept(function);

function.GetRoots(function);

function.GetGradients(function);

function.GetMaxPoints(function, MaxY, MinY);

function.GetMinPoints(function, MaxY, MinY);

}

}

private void Zoom(float x, float y, float multiplier)

{

float[] settings = new float[4];

float xdiff = MaxX - MinX;

float ydiff = MaxY - MinY;

xdiff = xdiff \* multiplier;

ydiff = ydiff \* multiplier;

x = PixToCart(x, y).X;

y = PixToCart(x, y).Y;

settings[0] = Convert.ToInt32(x - (xdiff / 2));

settings[1] = Convert.ToInt32(x + (xdiff / 2));

settings[2] = Convert.ToInt32(y - (ydiff / 2));

settings[3] = Convert.ToInt32(y + (ydiff / 2));

for (int i = 0; i < 4; i++)

{

if (settings[i] > 100)

{

settings[i] = 100;

}

if (settings[i] < -100)

{

settings[i] = -100;

}

}

MinX = settings[0];

MaxX = settings[1];

MinY = settings[2];

MaxY = settings[3];

WriteSettings();

FuncCheck();

}

private void Graph\_Load(object sender, EventArgs e)

{

btHelpG.Anchor = (AnchorStyles.Top | AnchorStyles.Right);

btSettingsG.Anchor = (AnchorStyles.Top | AnchorStyles.Right);

btExitG.Anchor = (AnchorStyles.Top | AnchorStyles.Right);

pbZoomIn.Anchor = (AnchorStyles.Top | AnchorStyles.Right);

pbZoomOut.Anchor = (AnchorStyles.Top | AnchorStyles.Right);

pbCursor.Anchor = (AnchorStyles.Top | AnchorStyles.Right);

pbUp.Anchor = (AnchorStyles.Top | AnchorStyles.Right);

pbDown.Anchor = (AnchorStyles.Top | AnchorStyles.Right);

pbLeft.Anchor = (AnchorStyles.Top | AnchorStyles.Right);

pbRight.Anchor = (AnchorStyles.Top | AnchorStyles.Right);

lbCoords.Anchor = (AnchorStyles.Bottom | AnchorStyles.Left);

graphObj = pbGraph.CreateGraphics();

}

private void Graph\_Resize(object sender, EventArgs e)

{

int gap = ((ClientRectangle.Height - 65) / 5) - 5;

txtFunc1.SetBounds(36, 80, 170, 23);

txtFunc2.SetBounds(36, 80 + gap, 170, 23);

txtFunc3.SetBounds(36, 80 + gap \* 2, 170, 23);

txtFunc4.SetBounds(36, 80 + gap \* 3, 170, 23);

txtFunc5.SetBounds(36, 80 + gap \* 4, 170, 23);

lby1.SetBounds(16, txtFunc1.Location.Y + 3, 24, 15);

lby2.SetBounds(16, txtFunc2.Location.Y + 3, 24, 15);

lby3.SetBounds(16, txtFunc3.Location.Y + 3, 24, 15);

lby4.SetBounds(16, txtFunc4.Location.Y + 3, 24, 15);

lby5.SetBounds(16, txtFunc5.Location.Y + 3, 24, 15);

slctFunc1.SetBounds(211, txtFunc1.Location.Y + 5, 15, 14);

slctFunc2.SetBounds(211, txtFunc2.Location.Y + 5, 15, 14);

slctFunc3.SetBounds(211, txtFunc3.Location.Y + 5, 15, 14);

slctFunc4.SetBounds(211, txtFunc4.Location.Y + 5, 15, 14);

slctFunc5.SetBounds(211, txtFunc5.Location.Y + 5, 15, 14);

label2.SetBounds(232, txtFunc1.Location.Y + 3, 89, 15);

label3.SetBounds(232, txtFunc2.Location.Y + 3, 89, 15);

label4.SetBounds(232, txtFunc3.Location.Y + 3, 89, 15);

label5.SetBounds(232, txtFunc4.Location.Y + 3, 89, 15);

label6.SetBounds(232, txtFunc5.Location.Y + 3, 89, 15);

Func1Colour.SetBounds(323, txtFunc1.Location.Y, 23, 23);

Func2Colour.SetBounds(323, txtFunc2.Location.Y, 23, 23);

Func3Colour.SetBounds(323, txtFunc3.Location.Y, 23, 23);

Func4Colour.SetBounds(323, txtFunc4.Location.Y, 23, 23);

Func5Colour.SetBounds(323, txtFunc5.Location.Y, 23, 23);

lbFunc1Info.SetBounds(16, txtFunc1.Location.Y + 30, 330, gap - 40);

lbFunc2Info.SetBounds(16, txtFunc2.Location.Y + 30, 330, gap - 40);

lbFunc3Info.SetBounds(16, txtFunc3.Location.Y + 30, 330, gap - 40);

lbFunc4Info.SetBounds(16, txtFunc4.Location.Y + 30, 330, gap - 40);

lbFunc5Info.SetBounds(16, txtFunc5.Location.Y + 30, 330, gap - 40);

int x = pbGraph.Location.X;

int y = pbGraph.Location.Y;

pbGraph.SetBounds(x, y, ClientRectangle.Width - x - 10, ClientRectangle.Height - y - 10);

if (WindowState == FormWindowState.Maximized)

{

FuncCheck();

}

}

private void btExitG\_Click(object sender, EventArgs e)

{

for (int i = 0; i >= 0; i--)

{

Application.OpenForms[i].Close();

}

}

private void btSettingsG\_Click(object sender, EventArgs e)

{

Settings settings = new Settings();

if (settings.Visible == false)

{

settings.ShowDialog();

}

else

{

settings.Hide();

}

graphObj = pbGraph.CreateGraphics();

graphObj.Clear(Color.White);

slctFunc1.Checked = false;

slctFunc2.Checked = false;

slctFunc3.Checked = false;

slctFunc4.Checked = false;

slctFunc5.Checked = false;

}

private void btHelpG\_Click(object sender, EventArgs e)

{

Help help = new Help();

if (help.Visible == false)

{

help.ShowDialog();

}

else

{

help.Hide();

}

}

private void slctFunc1\_CheckedChanged(object sender, EventArgs e)

{

if (slctFunc1.Checked)

{

function1.expression = txtFunc1.Text;

UpdateFunction(function1);

}

FuncCheck();

}

private void slctFunc2\_CheckedChanged(object sender, EventArgs e)

{

if (slctFunc2.Checked)

{

function2.expression = txtFunc2.Text;

UpdateFunction(function2);

}

FuncCheck();

}

private void slctFunc3\_CheckedChanged(object sender, EventArgs e)

{

if (slctFunc3.Checked)

{

function3.expression = txtFunc3.Text;

UpdateFunction(function3);

}

FuncCheck();

}

private void slctFunc4\_CheckedChanged(object sender, EventArgs e)

{

if (slctFunc4.Checked)

{

function4.expression = txtFunc4.Text;

UpdateFunction(function4);

}

FuncCheck();

}

private void slctFunc5\_CheckedChanged(object sender, EventArgs e)

{

if (slctFunc5.Checked)

{

function5.expression = txtFunc5.Text;

UpdateFunction(function5);

}

FuncCheck();

}

private void Graph\_ResizeEnd(object sender, EventArgs e)

{

FuncCheck();

}

private void Func1Colour\_Click(object sender, EventArgs e)

{

colorDialog1.ShowDialog();

Func1Colour.BackColor = colorDialog1.Color;

FuncCheck();

}

private void Func2Colour\_Click(object sender, EventArgs e)

{

colorDialog1.ShowDialog();

Func2Colour.BackColor = colorDialog1.Color;

FuncCheck();

}

private void Func3Colour\_Click(object sender, EventArgs e)

{

colorDialog1.ShowDialog();

Func3Colour.BackColor = colorDialog1.Color;

FuncCheck();

}

private void Func4Colour\_Click(object sender, EventArgs e)

{

colorDialog1.ShowDialog();

Func4Colour.BackColor = colorDialog1.Color;

FuncCheck();

}

private void Func5Colour\_Click(object sender, EventArgs e)

{

colorDialog1.ShowDialog();

Func5Colour.BackColor = colorDialog1.Color;

FuncCheck();

}

private void pbZoomIn\_Click(object sender, EventArgs e)

{

Cursor zoomIn = new Cursor("CursorZoomIn.cur");

Cursor = zoomIn;

cursor = "zoomin";

}

private void pbZoomOut\_Click(object sender, EventArgs e)

{

Cursor zoomOut = new Cursor("CursorZoomOut.cur");

Cursor = zoomOut;

cursor = "zoomout";

}

private void pbCursor\_Click(object sender, EventArgs e)

{

Cursor = Cursors.Default;

cursor = "default";

}

private void pbUp\_Click(object sender, EventArgs e)

{

MinY += Math.Abs((MaxY - MinY) / 10);

MaxY += Math.Abs((MaxY - MinY) / 10);

WriteSettings();

FuncCheck();

}

private void pbDown\_Click(object sender, EventArgs e)

{

MinY -= Math.Abs((MaxY - MinY) / 10);

MaxY -= Math.Abs((MaxY - MinY) / 10);

WriteSettings();

FuncCheck();

}

private void pbRight\_Click(object sender, EventArgs e)

{

MinX += Math.Abs((MaxX - MinX) / 10);

MaxX += Math.Abs((MaxX - MinX) / 10);

WriteSettings();

FuncCheck();

}

private void pbLeft\_Click(object sender, EventArgs e)

{

MinX -= Math.Abs((MaxX - MinX) / 10);

MaxX -= Math.Abs((MaxX - MinX) / 10);

WriteSettings();

FuncCheck();

}

private void txtFunc1\_TextChanged(object sender, EventArgs e)

{

slctFunc1.Checked = false;

}

private void txtFunc2\_TextChanged(object sender, EventArgs e)

{

slctFunc2.Checked = false;

}

private void txtFunc3\_TextChanged(object sender, EventArgs e)

{

slctFunc3.Checked = false;

}

private void txtFunc4\_TextChanged(object sender, EventArgs e)

{

slctFunc4.Checked = false;

}

private void txtFunc5\_TextChanged(object sender, EventArgs e)

{

slctFunc5.Checked = false;

}

private void pbGraph\_MouseClick(object sender, MouseEventArgs e)

{

if (cursor == "zoomin")

{

Zoom(pbGraph.PointToClient(MousePosition).X, pbGraph.PointToClient(MousePosition).Y, float.Parse("0.5"));

}

if (cursor == "zoomout")

{

Zoom(pbGraph.PointToClient(MousePosition).X, pbGraph.PointToClient(MousePosition).Y, 2);

}

lbCoords.Text = "";

if (Cursor == Cursors.Default)

{

Point pixPoint = new Point(e.Location.X, e.Location.Y);

if (slctFunc1.Checked)

{

CheckCoord(function1.PixPoints, function1.CartPoints, pixPoint);

}

if (slctFunc2.Checked)

{

CheckCoord(function2.PixPoints, function2.CartPoints, pixPoint);

}

if (slctFunc3.Checked)

{

CheckCoord(function3.PixPoints, function3.CartPoints, pixPoint);

}

if (slctFunc4.Checked)

{

CheckCoord(function4.PixPoints, function4.CartPoints, pixPoint);

}

if (slctFunc5.Checked)

{

CheckCoord(function5.PixPoints, function5.CartPoints, pixPoint);

}

}

}

### Settings.cs

public partial class Settings : Form

{

private string[] lines = new string[5];

private bool valid = false;

public Settings()

{

InitializeComponent();

}

private string[] ReadFile()

{

StreamReader reader = new StreamReader("Settings.txt");

for (int i = 0; i <= 4; i++)

{

lines[i] = reader.ReadLine();

}

reader.Close();

return lines;

}

private string[] WriteFile()

{

StreamWriter writer = new StreamWriter("Settings.txt");

for (int i = 0; i <= 4; i++)

{

writer.WriteLine(lines[i]);

}

writer.Close();

Graph graph = new();

graph.FuncCheck();

return lines;

}

public void InitialiseSettings()

{

StreamWriter writer = new StreamWriter("Settings.txt");

writer.WriteLine("-100");

writer.WriteLine("100");

writer.WriteLine("-100");

writer.WriteLine("100");

writer.WriteLine("Radians");

writer.Close();

}

private void Validation()

{

try

{

if (Convert.ToDouble(tbxMinX.Text) >= Convert.ToDouble(tbxMaxX.Text) || Convert.ToDouble(tbxMinY.Text) >= Convert.ToDouble(tbxMaxY.Text))

{

lbInvalid.Text = "Invalid: Minimum cannot be >= maximum.";

valid = false;

}

if (Convert.ToDouble(tbxMinX.Text) < -100 || Convert.ToDouble(tbxMinY.Text) < -100 ||

Convert.ToDouble(tbxMaxX.Text) > 100 || Convert.ToDouble(tbxMaxY.Text) > 100)

{

lbInvalid.Text = "Invalid: Must be in the range -100 <= Settings <= 100.";

valid = false;

}

else

{

lbInvalid.Text = null;

lbRejectClose.Text = null;

valid = true;

}

}

catch

{

lbInvalid.Text = "Invalid.";

valid = false;

}

}

private void lbSettings\_Load(object sender, EventArgs e)

{

btCloseS.Anchor = AnchorStyles.Bottom | AnchorStyles.Left;

btHelpS.Anchor = AnchorStyles.Bottom | AnchorStyles.Right;

lbAngle.Anchor = AnchorStyles.None;

lbMinX.Anchor = AnchorStyles.None;

lbMaxX.Anchor = AnchorStyles.None;

lbMinY.Anchor = AnchorStyles.None;

lbMaxY.Anchor = AnchorStyles.None;

tbxMaxX.Anchor = AnchorStyles.None;

tbxMaxY.Anchor = AnchorStyles.None;

tbxMinX.Anchor = AnchorStyles.None;

tbxMinY.Anchor = AnchorStyles.None;

rbtDegrees.Anchor = AnchorStyles.None;

rbtRadians.Anchor = AnchorStyles.None;

lbViewWin.Anchor = AnchorStyles.None;

lbRejectClose.Anchor = AnchorStyles.Top | AnchorStyles.Right;

lbInvalid.Anchor = AnchorStyles.Top | AnchorStyles.Right;

try

{

ReadFile();

}

catch

{

WriteFile();

}

tbxMinX.Text = lines[0];

tbxMaxX.Text = lines[1];

tbxMinY.Text = lines[2];

tbxMaxY.Text = lines[3];

if (lines[4] == "Degrees")

{

rbtRadians.Checked = false;

rbtDegrees.Checked = true;

}

else

{

rbtDegrees.Checked = false;

rbtRadians.Checked = true;

}

Validation();

}

private void btCloseS\_Click(object sender, EventArgs e)

{

if (valid == true)

{

WriteFile();

Close();

}

else

{

lbRejectClose.Text = "Cannot close settings if settings invalid.";

}

}

private void btHelpS\_Click(object sender, EventArgs e)

{

Help help = new Help();

if (help.Visible == false)

{

help.Show();

}

else

{

help.Hide();

}

}

private void tbxMinX\_TextChanged(object sender, EventArgs e)

{

Validation();

lines[0] = tbxMinX.Text;

}

private void tbxMaxX\_TextChanged(object sender, EventArgs e)

{

Validation();

lines[1] = tbxMaxX.Text;

}

private void tbxMinY\_TextChanged(object sender, EventArgs e)

{

Validation();

lines[2] = tbxMinY.Text;

}

private void tbxMaxY\_TextChanged(object sender, EventArgs e)

{

Validation();

lines[3] = tbxMaxY.Text;

}

private void rbtRadians\_CheckedChanged(object sender, EventArgs e)

{

if (rbtRadians.Checked)

{

lines[4] = "Radians";

}

else

{

lines[4] = "Degrees";

}

}

private void Settings\_FormClosing(object sender, FormClosingEventArgs e)

{

if (valid == false)

{

InitialiseSettings();

}

}

}

### Help.cs

public partial class Help : Form

{

public Help()

{

InitializeComponent();

MinimumSize = new Size(1216, 758);

MaximumSize = new Size(1216, 758);

}

}

# Testing

## Test Plan

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test No. | Test Description (objective/s) | Test Data | Expected Outcome | Actual Outcome |
| 1 | The Graph form loads appropriately. (1, 1.1) | N/A | Graph form loads, and all components appear as expected. |  |
| 2 | The connection between Graph, Settings and Help is correctly established. (2, 3) | N/A | When Settings and Help buttons are pressed, the respective forms are opened. |  |
| 3 | The application is correctly closed when prompted by the user. (8) | N/A | The exit buttons on the Graph form can be pressed at any time to close all forms. |  |
| 4 |  |  |  |  |
| 5 |  |  |  |  |
| 6 |  |  |  |  |
| 7 |  |  |  |  |
| 8 |  |  |  |  |
| 9 |  |  |  |  |
| 10 |  |  |  |  |
| 11 |  |  |  |  |
| 12 |  |  |  |  |
| 13 |  |  |  |  |
| 14 |  |  |  |  |
| 15 |  |  |  |  |
| 16 |  |  |  |  |
| 17 |  |  |  |  |
| 18 |  |  |  |  |
| 19 |  |  |  |  |
| 20 |  |  |  |  |
| 21 |  |  |  |  |
| 22 |  |  |  |  |

# Evaluation