

Contents:

1. Algorithms and Flowcharts.
2. Important Functions.
3. User Instructions.
4. Testing and Verification.
5. Overall Notes and Features.

Project

Algorithm and Flowchart

As the program works based on the Qt signals and slots system, the program is broken up into many small algorithms which each play a part depending on the function selected by the user. The following section will describe how the algorithms work however will leave out any smaller helper functions for a later section.

Linear Regression algorithm:

Figure 1 shows the flowchart used to describe how the linear regression algorithm functions. The program starts once the input file button has been pressed. At this point the user is prompted by the program to provide a file (“.csv” only), following this assuming a valid file path has been provided the data is parsed to two vectors “x\_data” and “y\_data”.

The program then waits for the plot button to be pressed. When activated; depending on the setting provided by the user, the program will either calculate the regression with or without a constant term. It does this by using the function gsl\_fit\_linear for producing the data with both terms, and gsl\_fit\_mul for data that does not require the constant term. Subsequently using the coefficients provided the program performs y = c1x + c0 or y = c1x (if constant term is disabled). The result of these equations is stored in a “x\_plot” vector and a “y\_plot” vector accordingly.

Using the min and max values of this data the program then creates the axis. The plot data is then used to plot the line onto the screen using the QGraphics library of functions and classes. Following this the input data is used to plot the points to the screen. Both methods of plotting the data to the screen use an iterating for loop to make sure each term is plotted by iterating through the vectors.

As a result of the gsl functions the sumsq variable is filled. This variable is used in the calculation of R squared which is shown by the equation in figure 2 or figure 3. Finally, once the graph is plotted and R squared has been calculated, the data is printed to the screen and the LCD’s are updated with the appropriate coefficients.

Diagram

Description automatically generated

**[Figure 1 – Flowchart showing the linear regression algorithm.]**

**Text

Description automatically generated with medium confidence**

**[Figure 2 – Coefficient of determination, R squared.]**

**Text, letter

Description automatically generated**

**[Figure 3 – Coefficient of determination R squared for the two lines once segmented.]**

Diagram

Description automatically generated

**[Figure 4 – Flowchart describing the bilinear regression algorithm.]**

**Text

Description automatically generated**

**[Figure 5 – Bilinear regression output.]**

Bilinear Regression algorithm:

Figure 4 briefly details how the algorithm functions. This algorithm is the most complex one out of all the algorithms in the program. Essentially it functions in the same way as the linear algorithm. It inputs data, produces the linear regression of that data, and plots it. However, for the bilinear regression algorithm the line must be split into two lines which better fit the data than just one regression line on its own. To solve this linear regression documentation was reviewed and from that the algorithm was produced. As the breakpoint (point of intersection x0) occurs at the point where the two lines cross, and these two lines must fit the data they encompass as well as possible (high goodness of fit). The best way to find this breakpoint was to calculate the lines with different breakpoints and iterate through the different line lengths until the best net goodness of fit could be found.

Firstly, the data was inputted using the same method as the linear regression algorithm. Assuming suitable data the program would then begin by starting the breakpoint at the smallest x value. At this breakpoint, the input data would then be split into sub-vectors, line 1 and line 2. Take for example input data of length 10, instead of these 10 points belonging to one line like in linear regression, instead the data is split as follows. To start line 1 contains just one data point and line 2 contains 9 (assuming 10 total data points have been inputted). The linear regression is then performed on each line and the goodness of fit R squared is stored for each of the lines.

The breakpoint x0 is then iterated to the second element, and subsequently line 1 now contains 2 data points and line 2 contains 8. The goodness of fit is the found for both two lines, and consequently stored. This is repeated for all the possible combinations the data can be split up into. An example of the output data produced is shown in figure 5. Note that in figure 5 there are two R squared values (ignore the adjusted ones), the net difference of these values is then calculated and compared with the rest of the line pairs net differences.

The smallest differences means that the goodness of fit across the two lines is at its greatest and therefore the breakpoint has been found. This breakpoint is then returned so that the program knows which two lines to use. Using this breakpoint, the two lines of data are then separately used in the gsl\_fit\_linear function and the values are found. The buildGraph function is then called and subsequently the two lines are now drawn on the screen accompanied by the input data.

Important parts of this algorithm:

Figure 7 shows the for loop used to iterate through potential values for the breakpoint. The offset value is used to make sure the breakpoint is never set after 1 or 2 points of data when splitting the lines. This is because if a line is produced between 2 points of data, the goodness of fit for that data will be 1. This is bad as the program would assume this to be the breakpoint which would result in an incorrect outcome. It also ignores setting the breakpoint after 1 point of data as this would result in the goodness of fit being 0 as a line cannot be drawn from only one point of data.

Important Helper Functions

inputData:

This function spawns a QFileDialog which returns the path of the file they wish to use. The file is then read line by line, after each line is read it is processed using the process\_data function. When all the lines of the file have been read, it updates some Boolean flags and returns true.

processData:

This function is used to split the lines inputted from inputData. Firstly, it creates tokens using the delimiter of “,”. A for loop then iterates through the tokens adding them to the vectors x and y accordingly. When all the tokens have been stored it returns.

tableFillBilinear:

Used to fill the table views with the data which has been inputted from the document, the function is also used for the linear table view.

plotBilinear:

This function is explained in the algorithms above.

buildGraph:

This function, depending on certain Boolean flags functions differently. It is used to calculate the line for the graph and plot it. It also calls the createAxis function which produces an axis of suitable size to the user.

throwErrorBox:

Used for handling any errors, pass it a string and it will produce a message error box showing the details of the error.

plotLinear:

Described by the algorithms above.

createAxis:

This function creates the axis to be the same pixels width and heigh as the viewport, it also makes sure all the points that have been inputted are shown within the axis.

statistics:

This function is used to find the coefficient of determination of the line. It uses the equation in figure 3 to calculate R squared.

changeLineColour:

This function allows the user to change the line colour using the Qt built in dialog.

graphOptions:

This function spawns the dialog used to generate a custom graph, this function is deprecated and shouldn’t be used, it was left in for added functionality but is only used for testing purposes and debugging.

updateColours:

This function updates the colour pixmaps that are on the main Ui tool bar, it allows the tool bar to show the current colour selected.

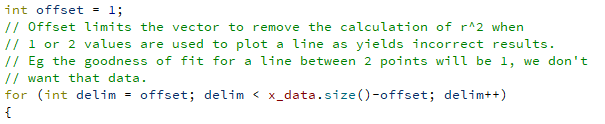
rsquComparator:

Used to compare all the values for R squared across all the potential lines. Described in bilinear algorithm above.

**Table

Description automatically generated with medium confidence**

**[Figure 6 – Suitable layout for data.]**



**[Figure 7 – Breakpoint iterating for loop.]**

User instructions:

The program takes input of a “.csv” document, the document should contain the values for x and y in vertical columns as shown in figure 6. Note that the program will throw an error if the format is not correct. If you are in the program depending which tab of the toolbar you are on you can see how to use the program by pressing the usage button. Figures 20-30 are used in the application for the usage feature. Refer to these figures for details on how to use each segment of the program. They can be found at the end of the report after the testing section or by clicking the usage buttons on the program.

Testing and verification:

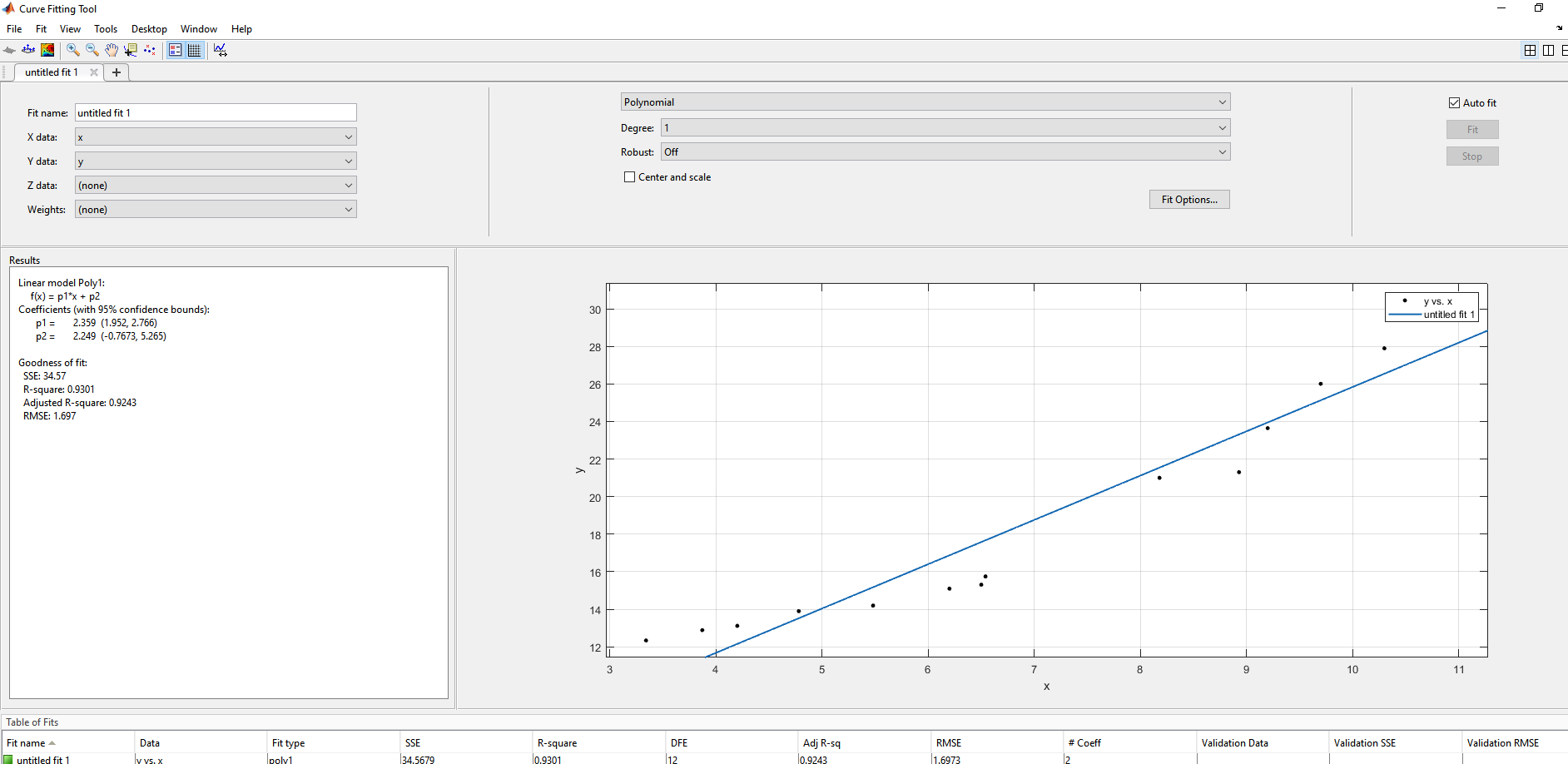
Test 1: Test that the program can produce a linear regression of the dataset shown in figure 6.

Figure 8 shows the output of the application using the data from figure 6, this output is with the constant term enabled. The values for A = 2.359, and B = 2.249. Referring to figure 9 which shows the result in MATLAB, which also produces the same constants A = 2.359 and B = 2.249. Furthermore, MATLAB produces the R squared value to be 0.9301, whereas this application produces the R squared more accurately as 9.30145. The lines of best fit are also in the same position therefore to conclude test 1, the application can accurately plot the data and graph and calculate the suitable values.

Chart, scatter chart

Description automatically generated

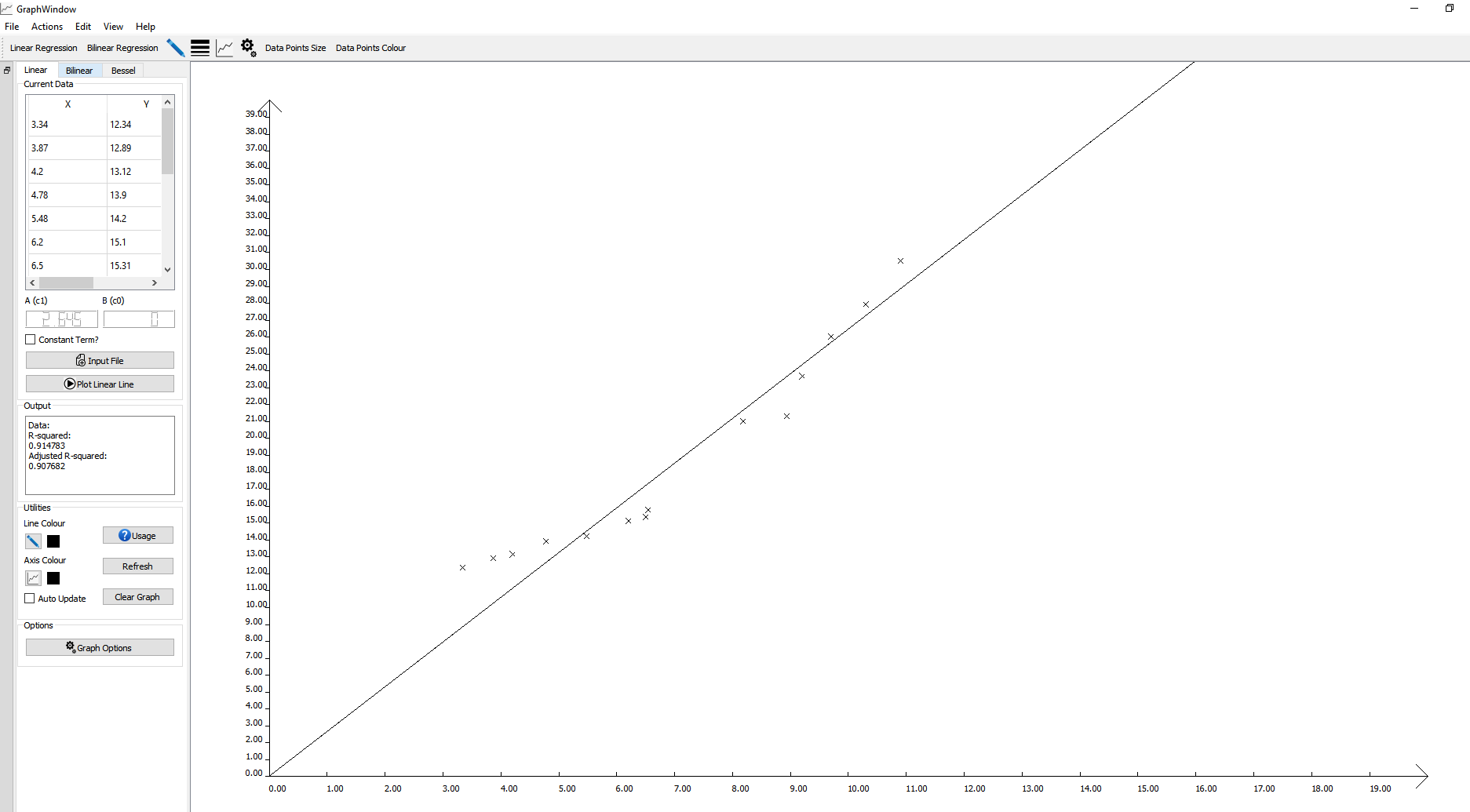
**[Figure 8 – Result of linear regression with data from figure 6.]**



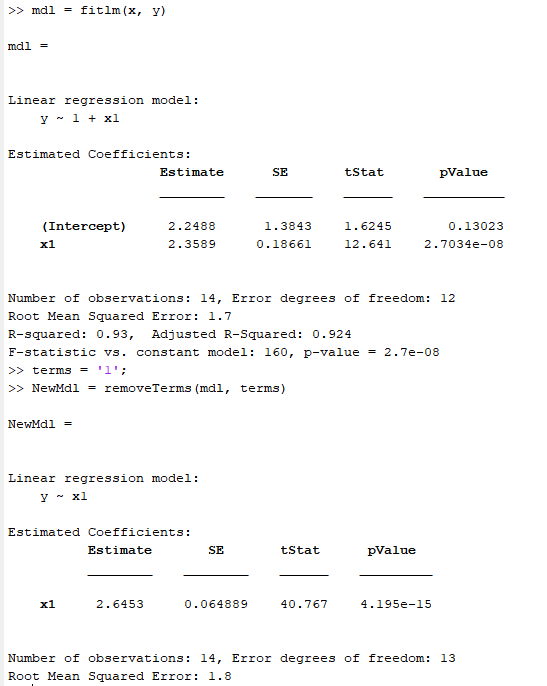
**[Figure 9 – MATLAB result with data from figure 6.]**

Test 2: Repeating test one without the constant term B.

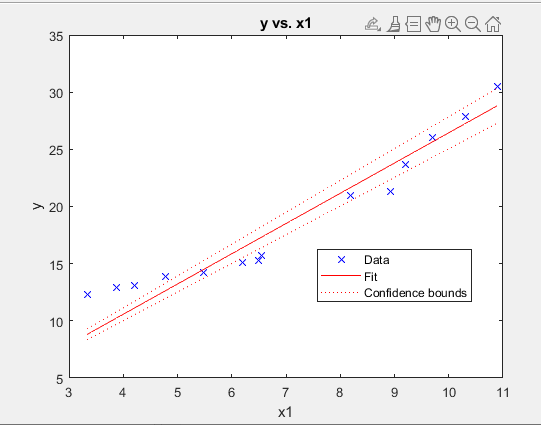
The application produces the graph as expected and recalculates the constant as A = 2.645. Referring to figure 11 which shows the MATLAB output for A = 2.6453, and figure 12 which shows the MATLAB graph output. Both graphs are the same and the coefficient A is the same for both applications, therefore this test was successful.



**[Figure 10 – Application result of linear regression plot without constant term for data in figure 6.]**



**[Figure 11 – MATLAB output of Linear regression terms with and without constant.]**



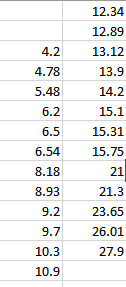
**[Figure 12 – Graph showing test 2 in MATLAB.]**

Test 3: Inputting invalid data into the application (wrong size/missing elements).

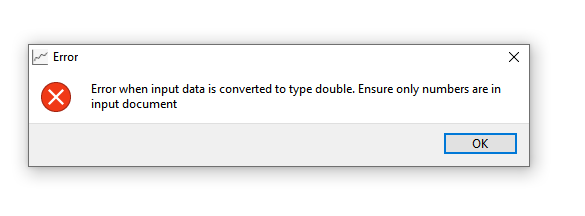
Using the data shown in figure 13, the program will be tested to see if the exception is handled or if it crashes. The result can be seen in figure 14. The program catches the error and throws a message box as expected. A key part of the program was preventing issues such as this, and the result was as expected. Upon closing the message box if the user attempts to plot data they cannot. Instead, they must first input the correct data.

Test 4: Inputting invalid data into the application (characters).

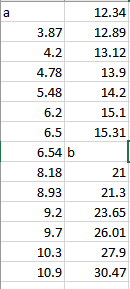
Replacing some of the data with characters, the document was then re-inputted into the application, as the error is triggered by the same function, the same error box (figure 14) is thrown as in test 3.



**[Figure 13 – Invalid data used in test 3.]**



**[Figure 14 – Result of test 3 and 4.]**



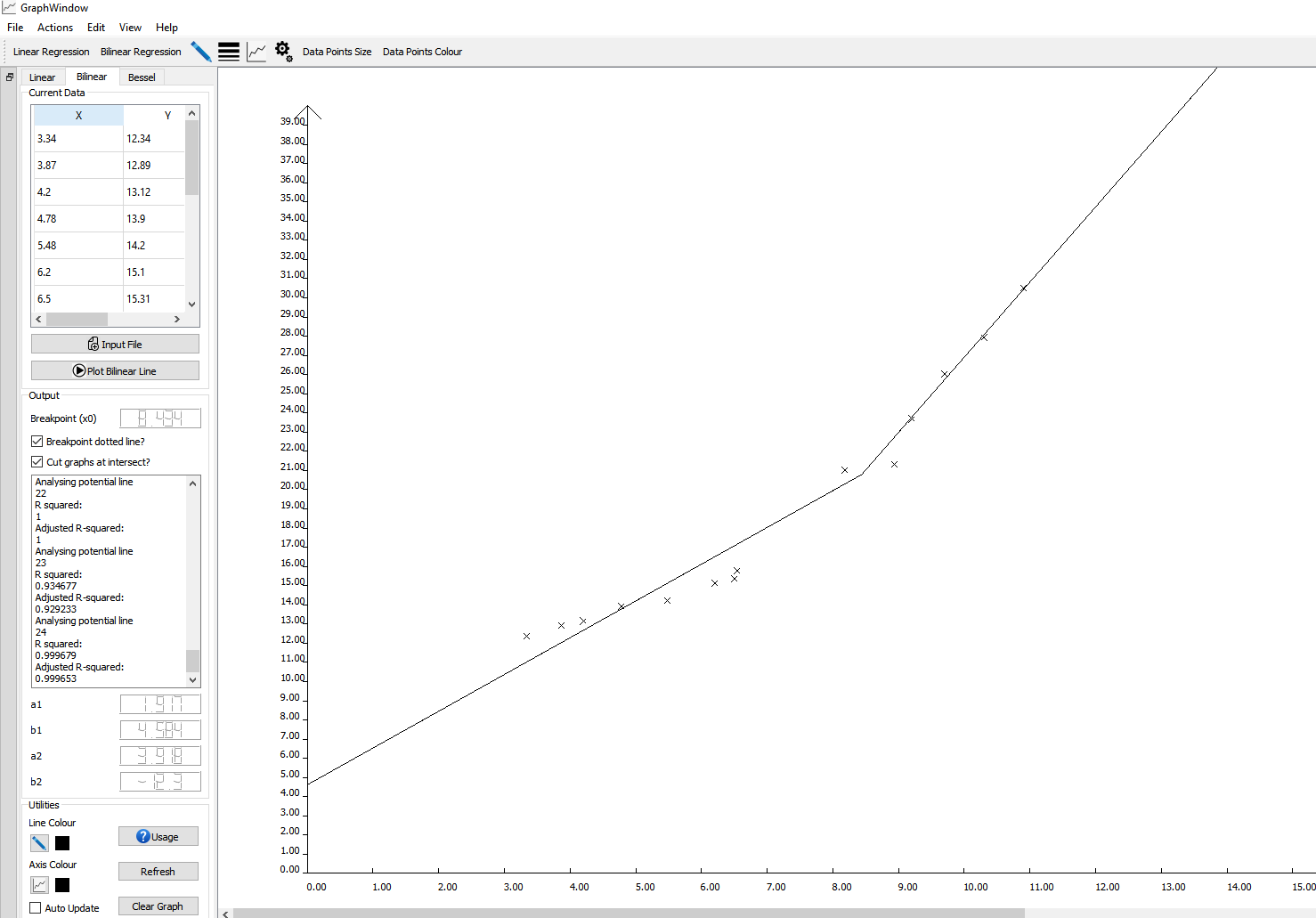
**[Figure 15 – Invalid excel document.]**

Test 5: Trying every button to look for exceptions or bugs.

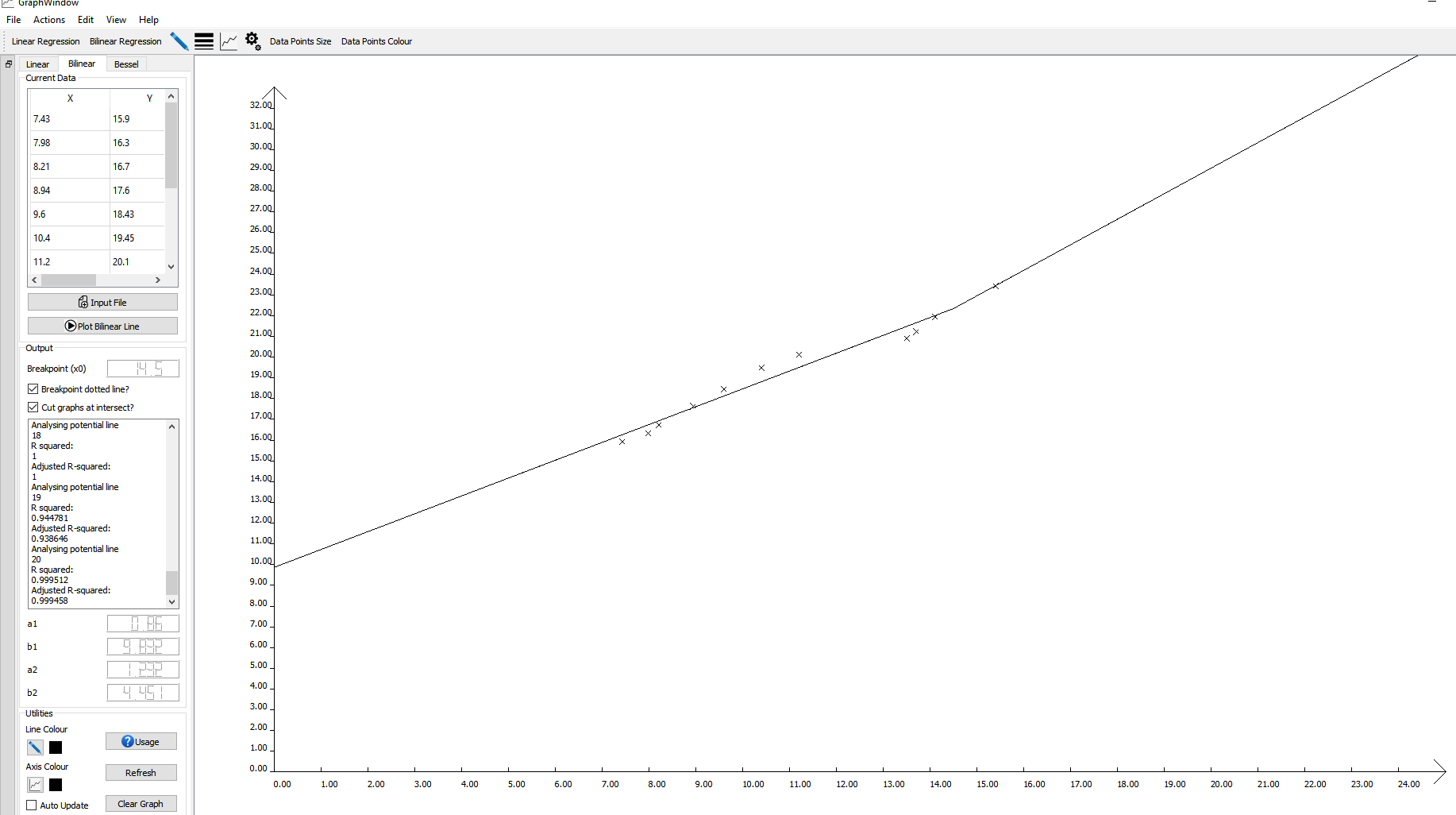
Throughout the designing and coding of this application, I coded so that certain buttons only become active once a flag is set. This allowed the program to stay bug free. However, one bug I discovered during test 3 was that if a document had been previously plotted (valid data), and then an invalid document was then inputted. The program would crash. This is the only bug I found during the testing stage however in the next steps of this application that bug should be fixed. In the early stages of the development, many of the dialogs which input data would crash when you hit the X. To resolve this, I coded closeEvents which set default values when X was pressed.

Test 6: Bilinear Regression.

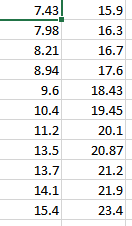
Using the data provided in figure 6, bilinear regression was applied to the data and the output can be found in figure 16. I was unable to get a graph to compare with however the application produces a result which looks as though it fits the data with a suitable breakpoint. Figure 17 shows the output with a different set of data which is shown in figure 18. After plotting the second set of data, I realised there was a bug. The breakpoint line from the first set of data remained on the axis even with the new graph (figure 19). This will be due to the object not being properly removed from the scene and will be resolved in the future.



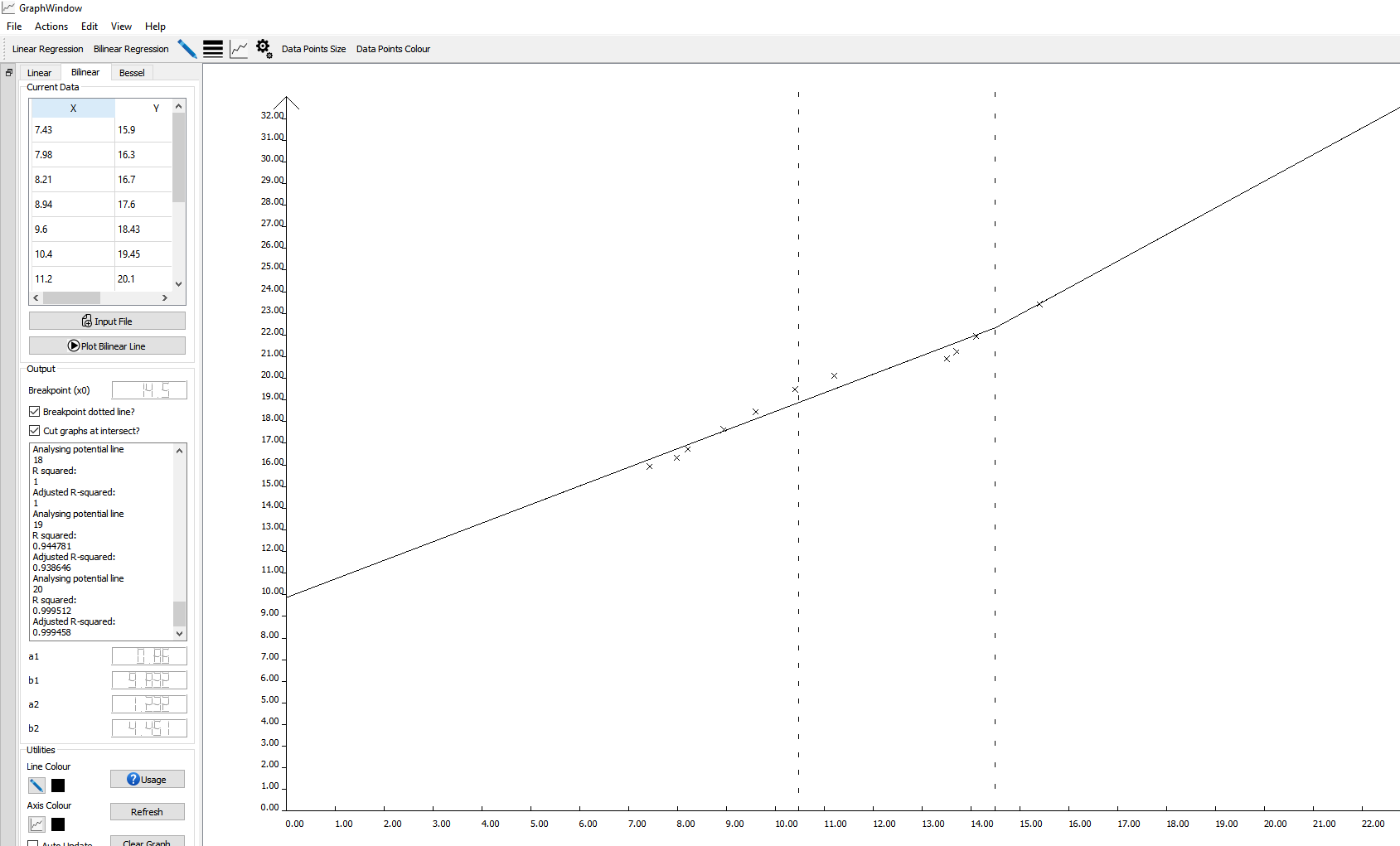
**[Figure 16 – Output of bilinear regression.]**



**[Figure 17 – Output of bilinear regression using data set in figure 18.]**



**[Figure 18 – Data set used in test 6.]**



**[Figure 19 – Bug produced when multiple different sets of data are plotted.]**

Overall Notes

This section will detail some extra features the program has.

The first main additional feature is the usage buttons, these buttons were briefly described in the user instructions section however they will be explained in more detail here. The usage buttons are specific to the regression tab the user is on, when pressed a dialog help box will show, in this dialog a slideshow of detailed images are presented to the user. This is meant to provide a detailed and clear user experience, furthermore if the project was to be developed further. These slideshows would be the basis of a more advanced user help system which would walk the user through the applications extensive features.

A key part of designing the software was to keep customisation paramount. The brief only requires the colour of the line to be changed in three different ways, however I instead opted into having customisation for the colour of the line, the axis, and the plotted points. Furthermore, the user can increase the size of the crosses and increase the thickness of the fitted curve if desired. There is also a feature which allows for a custom graph to be made, this graph is produced when the user enters different values for the minimum and maximum pixels etc. The graph options button however is a deprecated function and was only left in for future use.

Utilities box:

The utilities bar is used for help, to clear the graph/refresh it or to change the colours of the line or axis. The toolbox also has a tick box which will automatically update colours as and when you change them instead of waiting until the next graph is plotted. Furthermore, a live display of the current colour is shown in this utility’s toolbox.

Bilinear Graph Tab:

The bilinear graph tab has two features which show the true customisability of the program. Firstly, there is an option to show or hid the breakpoint dotted line, this is convenient for showing exactly where the breakpoint is. Secondly, there is an option to cut the lines at the point of intersection, this feature gives the output a much more professional look.

Tool / Menu bar:

All features of the program can be completed from the tool or menu bar. In the view tab the toolbox widget which sits on the left side of the application can be hidden giving a full screen. This is an extra feature designed to give the application a more professional finish. Furthermore, the user also has the option to turn the scrollbars on or off as needed. These features are the sorts of ones found on professional programs like MATLAB or EXCEL, therefore implementing them has made this application feel more like a professional application than a project.

Table view:

An important feature of the program was displaying the data to the user in a clear form. For the input data using a table view and model made this much easier as the data could be put into a clear table and shown to the user.

Graphical user interface, application, Word

Description automatically generated

**[Figure 20 – Bilinear help menu 1.]**

Graphical user interface, application, Word

Description automatically generated

**[Figure 21 – Bilinear help menu 2.]**

Graphical user interface, application

Description automatically generated

**[Figure 22 – Bilinear help menu 3.]**

Graphical user interface, application, Word

Description automatically generated

**[Figure 23 – Bilinear help menu 4.]**

Diagram

Description automatically generated

**[Figure 24 – Bilinear help menu 5.]**

Graphical user interface

Description automatically generated with medium confidence

**[Figure 25 – Bilinear help menu 6.]**

Graphical user interface, application

Description automatically generated

**[Figure 26 – Bilinear help menu 7.]**

Graphical user interface, application

Description automatically generated

**[Figure 27 – Linear help menu 1.]**

Graphical user interface, application

Description automatically generated

**[Figure 28 – Linear help menu 2.]**

Graphical user interface, diagram

Description automatically generated

**[Figure 29 – Linear help menu 3.]**

Graphical user interface, application

Description automatically generated

**[Figure 30 – Linear help menu 4.]**

#include "graphwindow.h"

#include "ui\_graphwindow.h"

#include "addnewdata.h"

GraphWindow::**GraphWindow**(QWidget \*parent)

: QMainWindow(parent)

, ui(new Ui::GraphWindow)

{

ui->setupUi(this);

x\_startval = 0; // X start value for the axis.

y\_startval = 0; // Y start value for the axis.

size\_crosses = 3; // Used for the size of the data points.

ui->dockWidget->setMinimumWidth(240); // Restricts dock width min.

ui->dockWidget->setMaximumWidth(240); // Restricts dock width max.

scrollbars(false); // Disable scrollbars by default.

setCentralWidget(ui->graphicsView); // Sets graphics view as main widget.

ui->graphicsView->setScene(scene); // Sets the scene.

line1 = new QGraphicsItemGroup(); // Line 1 item group. (linear and bilinear)

line2 = new QGraphicsItemGroup(); // Line 2 item group. (only bilinear)

x0\_line = new QGraphicsItemGroup(); // Dotted line that shows at breakpoint.

plot\_item = new QGraphicsLineItem(); // Used to plot the line in segments.

point\_item = new QGraphicsLineItem(); // Used to plot the points X's.

x0\_item = new QGraphicsLineItem(); // Used for drawing the breakpoint line.

plot\_pen.setColor(Qt::black); // Sets line colour.

axis\_pen->setColor(Qt::black); // Sets axis colour.

point\_pen.setColor(Qt::black); // Sets point colour.

pen\_thickness = 1;

plot\_pen.setWidth(pen\_thickness);

updateColours();

}

GraphWindow::~***GraphWindow***()

{

delete ui;

}

void GraphWindow::***contextMenuEvent***(QContextMenuEvent \*event)

{

// Called when right click occurs on app.

QMenu menu(this);

QMenu\* submenuAddData = menu.addMenu("Add Data");

QMenu\* submenuPlot = menu.addMenu("Plot Type");

submenuPlot->addAction(ui->actionPlot);

submenuAddData->addAction(ui->actionLinear\_Regression);

submenuPlot->addAction(ui->actionPlot\_Bilinear\_Line);

submenuAddData->addAction(ui->actionBilinear\_Regression);

menu.addAction(ui->actionThickness);

menu.addAction(ui->actionColour);

menu.exec(event->globalPos());

}

void GraphWindow::***closeEvent***(QCloseEvent \*event)

{

// Handles close event, if X is pressed this prompt comes up.

if (QMessageBox::Yes != QMessageBox::question(this, "Close Confirmation", "Are you sure you want to exit the program?", QMessageBox::Yes | QMessageBox::No))

{

event->ignore();

}

}

void GraphWindow::**scrollbars**(bool active)

{

// Used to toggle scrollbars on or off.

if (active)

{

ui->graphicsView->setHorizontalScrollBarPolicy(Qt::ScrollBarAsNeeded);

ui->graphicsView->setVerticalScrollBarPolicy(Qt::ScrollBarAsNeeded);

}

else

{

ui->graphicsView->setHorizontalScrollBarPolicy(Qt::ScrollBarAlwaysOff);

ui->graphicsView->setVerticalScrollBarPolicy(Qt::ScrollBarAlwaysOff);

}

}

void GraphWindow::**clearGraph**()

{

// Used to remove graph data depending on which data is currently present.

if (!bilinear\_data) scene->removeItem(line1);

else

{

scene->removeItem(line1);

scene->removeItem(line2);

if (dotted\_line) scene->removeItem(x0\_line);

}

ui->graphicsView->update();

}

void GraphWindow::**updateScene**()

{

// Updates scene.

ui->graphicsView->update();

ui->graphicsView->repaint();

}

void GraphWindow::**on\_actionEnable\_Scrollbars\_toggled**(bool arg1)

{

// Toggles scrollbars.

scrollbars(arg1);

}

void GraphWindow::**on\_pushButton\_inputfile\_clicked**()

{

// Inputs data and if valid data is present populates the table.

if(inputData(1))

tableFillLinear();

}

bool GraphWindow::**processData**(QString line\_read)

{

// Processes data from line\_read input.

QStringList tokens; // splits the line into tokens, x and y.

bool valid\_double = true;

// Splits the line using delimiter ",".

tokens = line\_read.split(QRegExp(","));

// Error handler.

if(tokens.size() == 1)

{

throwErrorBox("Invalid input data, x and y values must be provided.");

return false;

}

// Iterates through the tokens, filling x and y vectors.

for (int i = 0; i < tokens.size();i+=2)

{

try

{

x\_data.push\_back(tokens.at(i).toDouble(&valid\_double));

y\_data.push\_back(tokens.at(i+1).toDouble(&valid\_double));

} catch (std::exception&)

{

throwErrorBox("Error when input data is converted to type double. Ensure only numbers are in input document");

return false;

}

if (!valid\_double)

{

throwErrorBox("Error when input data is converted to type double. Ensure only numbers are in input document");

return false;

}

}

return true;

}

void GraphWindow::**tableFillBilinear**()

{

// Used to update the table views with the input data.

table\_bilinear = new TableModel(this);

table\_bilinear->populateData(x\_data, y\_data);

ui->tableView\_2->*setModel*(table\_bilinear);

ui->tableView\_2->setColumnWidth(30, 80);

ui->tableView\_2->horizontalHeader()->*setVisible*(true);

ui->tableView\_2->show();

}

void GraphWindow::**tableFillLinear**()

{

// Used to update the table views with the input data.

table\_linear = new TableModel(this);

table\_linear->populateData(x\_data, y\_data);

ui->tableView->*setModel*(table\_linear);

ui->tableView->setColumnWidth(30, 80);

ui->tableView->horizontalHeader()->*setVisible*(true);

ui->tableView->show();

}

void GraphWindow::**plotBilinear**()

{

// This function plots the bilinear data.

if (!valid\_bilinear\_data)

return;

// Line no describes how many lines have been attempted to fit the data.

line\_no = 1;

// Sets the output text edit to contain R squared data.

ui->textEdit\_bilinea\_data->setText("Calculating goodness of fit (R squared) across sample selections of points.");

// These vectors are used to split the data vectors x\_data and y\_data so that the goodness of fit for two different

// lines can be found. Eg if main vec contains 10, these sub vectors will contain say, 2 and 8. or 4 and 6. etc.

QVector <double> subvec1\_x, subvec1\_y;

QVector <double> subvec2\_x, subvec2\_y;

// Used to create the 2D vector.

r\_squared.push\_back(QVector <double> ());

r\_squared.push\_back(QVector <double> ());

// Offset value.

int offset = 1;

// Offset limits the vector to remove the calculation of r^2 when

// 1 or 2 values are used to plot a line as yields incorrect results.

// Eg the goodness of fit for a line between 2 points will be 1, we don't

// want that data.

for (int delim = offset; delim < x\_data.size()-offset; delim++)

{

// Clears the vectors before the iteration.

subvec1\_x.clear();

subvec1\_y.clear();

subvec2\_x.clear();

subvec2\_y.clear();

// The following two for loops are used to split the data into the sub vectors.

for (int j = 0;j < delim ;j++)

{

subvec1\_x.push\_back(x\_data[j]);

subvec1\_y.push\_back(y\_data[j]);

}

for (int j = delim;j < x\_data.size() ;j++ )

{

subvec2\_x.push\_back(x\_data[j]);

subvec2\_y.push\_back(y\_data[j]);

}

// The linear regression for the two sepparate lines is then calculated.

// statistics is then used to calculate the goodness of fit and stores it

// for later analysis.

try

{

gsl\_fit\_linear(subvec1\_x.data(), 1, subvec1\_y.data(), 1, subvec1\_x.size(), &c0[0], &c1[0], &cov00[0], &cov01[0], &cov11[0], &sumsq[0]);

statistics(2, 0);

gsl\_fit\_linear(subvec2\_x.data(), 1, subvec2\_y.data(), 1, subvec2\_x.size(), &c0[1], &c1[1], &cov00[1], &cov01[1], &cov11[1], &sumsq[1]);

statistics(2, 1);

} catch (std::exception&)

{

throwErrorBox("Error when performing gsl\_fit\_linear.");

return;

}

}

// This breakpoint is the array position of x0 once the data has been analysed in rsquComparator.

int breakpoint = rsquComparator();

// Using the break point, the correct data for each line is pushed into the vectors.

for (int index = 0;index < breakpoint ;index++)

{

subvec1\_x.push\_back(x\_data[index]);

subvec1\_y.push\_back(y\_data[index]);

}

for (int index = breakpoint;index < x\_data.size() ;index++ )

{

subvec2\_x.push\_back(x\_data[index]);

subvec2\_y.push\_back(y\_data[index]);

}

// The graph is cleared before it is plotted.

clearGraph();

// Since the correct breakpoint has been found, the new data is then used to calculate the two lines of best fit.

try

{

gsl\_fit\_linear(subvec1\_x.data(), 1, subvec1\_y.data(), 1, subvec1\_x.size(), &c0[0], &c1[0], &cov00[0], &cov01[0], &cov11[0], &sumsq[0]);

statistics(2, 0);

gsl\_fit\_linear(subvec2\_x.data(), 1, subvec2\_y.data(), 1, subvec2\_x.size(), &c0[1], &c1[1], &cov00[1], &cov01[1], &cov11[1], &sumsq[1]);

statistics(2, 1);

} catch (std::exception&)

{

x\_data.clear();

y\_data.clear();

throwErrorBox("Error when performing gsl\_fit\_linear.");

return;

}

// This function is used to create the graph and plot the line.

buildGraph(0);

buildGraph(1);

bilinear\_data = true;

valid\_linear\_data = false;

valid\_bilinear\_data = true;

// This clears the r\_squared vector.

r\_squared[0].clear();

r\_squared[1].clear();

// The following values are then displayed to the user.

//c0 is b1 & b2 depending on the array.

//c1 is a1 & a2 accordingly.

x0 = (c0[1]-c0[0])/(c1[0] - c1[1]);

ui->lcdNumber\_breakpoint->display(x0);

ui->lcdNumber\_a1->display(c1[0]);

ui->lcdNumber\_b1->display(c0[0]);

ui->lcdNumber\_a2->display(c1[1]);

ui->lcdNumber\_b2->display(c0[1]);

}

void GraphWindow::**buildGraph**(int array\_index)

{

x\_plot.clear();

y\_plot.clear();

// The start and end vectors determin how much of the line is drawn.

// for example the bilinear graph cuts the lines at the point of intersection.

// this is achieved by setting the variables suitably.

double start = 0;

double end = 0;

if (limit\_line == true && array\_index == 0)

{

// Used for cutting line at intersection.

start = x\_startval;

end = x0;

}

else if (limit\_line == true && array\_index == 1)

{

// Used for cutting line at intersection.

start = x0;

end = x\_data.back()+offset;

}

else

{

// Used for default line drawing.

start = x\_startval;

end = x\_data.back()+offset;

}

// This for loop iterates across the graph and calculates the values so that the line

// can be drawn.

for(double i = start; i < end; i+= 0.01)

{

if (const\_term) y\_plot.push\_back((i\*c1[array\_index]) + c0[array\_index]);

else y\_plot.push\_back((i\*c1[array\_index]));

x\_plot.push\_back(i);

}

if (!array\_index) line1 = new QGraphicsItemGroup();

else line2 = new QGraphicsItemGroup();

// Creates a suitable axis for the line data.

createAxis(x\_data.back()+offset, y\_data.back()+offset, x\_startval, y\_startval);

for (int i = 1; i < x\_plot.size(); i++)

{

// Plots the fitted line.

plot\_item = new QGraphicsLineItem(x\_pix\_to\_data\*x\_plot[i-1], -1\*(y\_pix\_to\_data\*y\_plot[i-1]), x\_pix\_to\_data\*x\_plot[i], -1\*(y\_pix\_to\_data\*y\_plot[i]));

// Sets the line colour.

plot\_item->setPen(plot\_pen);

if (!array\_index) line1->addToGroup(plot\_item);

else line2->addToGroup(plot\_item);

}

for (int i = 0; i < x\_data.size(); i++)

{

// Plots the points and sets the colour they will take.

// note the size\_crosses variable which is used to adjust the crosses.

point\_item = new QGraphicsLineItem(x\_pix\_to\_data\*x\_data[i]-size\_crosses, -1\*(y\_pix\_to\_data\*y\_data[i])-size\_crosses, x\_pix\_to\_data\*x\_data[i]+size\_crosses, -1\*(y\_pix\_to\_data\*y\_data[i])+size\_crosses);

point\_item->setPen(point\_pen);

if (!array\_index) line1->addToGroup(point\_item);

else line2->addToGroup(point\_item);

point\_item = new QGraphicsLineItem(x\_pix\_to\_data\*x\_data[i]-size\_crosses, -1\*(y\_pix\_to\_data\*y\_data[i])+size\_crosses, x\_pix\_to\_data\*x\_data[i]+size\_crosses, -1\*(y\_pix\_to\_data\*y\_data[i])-size\_crosses);

point\_item->setPen(point\_pen);

if (!array\_index) line1->addToGroup(point\_item);

else line2->addToGroup(point\_item);

}

// Adds line to scene.

if (!array\_index) scene->addItem(line1);

else scene->addItem(line2);

// Translates view to center on graph.

ui->graphicsView->translate(100, 0);

}

void GraphWindow::**x0DottedLine**()

{

// This function draws or removes the breakpoint line.

if (!bilinear\_data)

return;

else if (dotted\_line)

{

// Print it.

for (int i = 0; i < y\_data.back()+offset; i++)

{

// Plots the fitted line.

x0\_item = new QGraphicsLineItem(x\_pix\_to\_data\*x0, -1\*(y\_pix\_to\_data\*i)-5, x\_pix\_to\_data\*x0, -1\*(y\_pix\_to\_data\*i));

x0\_item->setPen(plot\_pen);

x0\_line->addToGroup(x0\_item);

}

scene->addItem(x0\_line);

}

else

{

// Remove it.

scene->removeItem(x0\_line);

updateScene();

}

}

void GraphWindow::**throwErrorBox**(QString error\_msg)

{

// Used for handling error messages.

// This function has been used to prevent repeating

// code each time an error box is needed.

error\_box->setWindowTitle("Error");

error\_box->setIcon(QMessageBox::Critical);

error\_box->setText(error\_msg);

error\_box->setButtonText(0, "OK");

error\_box->*exec*();

}

void GraphWindow::**plotLinear**()

{

// Plots the linear data.

if (!valid\_linear\_data)

return;

clearGraph();

try

{

if (const\_term) gsl\_fit\_linear(x\_data.data(), 1, y\_data.data(), 1, x\_data.size(), &c0[0], &c1[0], &cov00[0], &cov01[0], &cov11[0], &sumsq[0]);

else gsl\_fit\_mul(x\_data.data(), 1, y\_data.data(), 1, x\_data.size(), &c1[0], &cov11[0], &sumsq[0]);

} catch (std::exception&)

{

x\_data.clear();

y\_data.clear();

throwErrorBox("Error when performing gsl\_fit\_linear.");

return;

}

// Creates the graph.

buildGraph(0);

// Updates displays.

ui->lcdNumber\_a->display(c1[0]);

if (const\_term) ui->lcdNumber\_b->display(c0[0]);

else ui->lcdNumber\_b->display(0);

// Translates view to center on graph.

ui->graphicsView->translate(100, 0);

// Calculates the r squared.

statistics(1, 0);

valid\_bilinear\_data = false;

bilinear\_data = false;

}

void GraphWindow::**createAxis**(int num\_x\_ticks, int num\_y\_ticks, int start\_x, int start\_y)

{

// Deltes the current axis if one is present.

if (!first\_pass)

{

scene->removeItem(x\_axis);

delete x\_axis;

scene->removeItem(y\_axis);

delete y\_axis;

scene->update();

}

// Creates new axis with a margin of 100 pix.

int margin = 100;

x\_axis = new QGraphicsAxisItem('x', 0, ui->graphicsView->width() - margin\*2, num\_x\_ticks, 1, start\_x, axis\_colour);

y\_axis = new QGraphicsAxisItem('y', 0, ui->graphicsView->height() - margin, num\_y\_ticks, 1, start\_y, axis\_colour);

scene->addItem(x\_axis);

scene->addItem(y\_axis);

x\_pix\_to\_data = x\_axis->pixel\_to\_data();

y\_pix\_to\_data = y\_axis->pixel\_to\_data();

// Used to detect if this is the first pass of the program.

first\_pass = false;

}

void GraphWindow::**statistics**(int mode, int index)

{

// This function is used to calculate the goodness of fit.

int n = x\_data.size();

// This totals up y vector.

double running\_total = 0;

for (int i = 0; i < y\_data.size(); i++)

running\_total = running\_total + y\_data[i];

// Finds average y using total.

double y\_ave = running\_total/n;

// Finds the degree of liberty which is used for calculating the adjusted r squared value.

double deg\_lib = n-2;

// This totals up the SCT. See equation @https://en.wikipedia.org/wiki/Segmented\_regression

double sq\_total = 0;

for (int i = 0;i <y\_data.size(); i++)

sq\_total = sq\_total + pow(y\_data[i] - y\_ave, 2);

// Finds r squared based on the equation found @https://en.wikipedia.org/wiki/Segmented\_regression

// scroll and look for the Cd equation (coefficient of determination for all data).

double Rsqu = 1 - sumsq[index]/sq\_total;

// Finds the adjusted r squared value.

double adjus\_Rsqu = 1 - (double)(n-1) / deg\_lib\*(1-Rsqu);

// Sends output to screen.

if (mode == 1)

{

ui->textEdit->setText("Data:");

ui->textEdit->append("R-squared:");

ui->textEdit->append(QString::number(Rsqu));

ui->textEdit->append("Adjusted R-squared:");

ui->textEdit->append(QString::number(adjus\_Rsqu));

}

else

{

r\_squared[index].push\_back(Rsqu);

ui->textEdit\_bilinea\_data->append("Analysing potential line ");

ui->textEdit\_bilinea\_data->append(QString::number(line\_no));

ui->textEdit\_bilinea\_data->append("R squared: ");

ui->textEdit\_bilinea\_data->append(QString::number(Rsqu));

ui->textEdit\_bilinea\_data->append("Adjusted R-squared:");

ui->textEdit\_bilinea\_data->append(QString::number(adjus\_Rsqu));

}

line\_no++;

}

void GraphWindow::**changeLineColour**()

{

// Changes line colour.

line\_colour = QColorDialog::getColor(Qt::black, this, tr("Pick Colour Of Line"));

plot\_pen.setColor(line\_colour);

updateColours();

// Used to auto update the colours if bool is true.

if (first\_pass)

return;

else if (auto\_change == true && bilinear\_data == false)

plotLinear();

else if (auto\_change == true && bilinear\_data == true)

plotBilinear();

}

void GraphWindow::**changeAxisColour**()

{

// Changes axis colour using Qt dialog.

axis\_colour = QColorDialog::getColor(Qt::black, this, tr("Pick Colour Of Axis"));

axis\_pen->setColor(axis\_colour);

updateColours();

// Used to auto update the colours if bool is true.

if (first\_pass)

return;

else if (auto\_change == true && bilinear\_data == false)

plotLinear();

else if (auto\_change == true && bilinear\_data == true)

plotBilinear();

}

void GraphWindow::**graphOptions**()

{

// Function used to produce a custom axis. Not recommended for regular use.

// Calls the graph options ui.

GraphOptions edit(this);

int ret\_code = edit.*exec*();

if (!ret\_code)

return;

if (!first\_pass)

{

scene->removeItem(x\_axis);

delete x\_axis;

scene->removeItem(y\_axis);

delete y\_axis;

scene->update();

}

ui->graphicsView->update();

// Makes new axis.

x\_axis = new QGraphicsAxisItem('x', edit.x\_pix\_min, edit.x\_pix\_max, edit.x\_ticks, edit.tick\_increment, edit.x\_startval, axis\_colour);

y\_axis = new QGraphicsAxisItem('y', edit.y\_pix\_min, edit.y\_pix\_max, edit.y\_ticks, edit.tick\_increment, edit.y\_startval, axis\_colour);

scene->addItem(x\_axis);

scene->addItem(y\_axis);

ui->graphicsView->update();

first\_pass = false;

}

void GraphWindow::**updateColours**()

{

// This sets the labels in the toolbox to change to the current colour of the line or axis.

line\_pixmap.fill(line\_colour);

ui->label\_linecolour->setPixmap(line\_pixmap);

ui->label\_linecolour\_2->setPixmap(line\_pixmap);

axis\_pixmap.fill(axis\_colour);

ui->label\_axiscolour->setPixmap(axis\_pixmap);

ui->label\_axiscolour\_2->setPixmap(axis\_pixmap);

}

bool GraphWindow::**inputData**(int mode)

{

// mode 1 is linear, mode 2 is bilinear.

bool val\_data = false;

x\_data.clear();

y\_data.clear();

// Gets file path from file Ui.

file\_path = QFileDialog::getOpenFileName(this, tr("Open File"), QDir::currentPath(), tr("File(\*.csv)"));

// Opens file accordingly.

QFile file(file\_path);

if (!file.*open*(QIODevice::ReadOnly | QIODevice::Text))

{

// Handles invalid file paths such as clicking X on ui.

valid\_linear\_data = false;

valid\_bilinear\_data = false;

return false;

}

// Following try catch reads first line from file.

QString line;

try

{

line = file.readLine();

} catch (std::exception&)

{

throwErrorBox("Read line function failed.");

return false;

}

// While loop iterates until EOF is read.

while(!line.isNull())

{

// This function is used to split the input line.

val\_data = processData(line); // Returns true if valid.

// Error checking for valid data.

if(!val\_data) break;

// Reads next line.

line = file.readLine();

}

// Error handling.

if (!val\_data)

{

x\_data.clear();

y\_data.clear();

return false;

}

// Sets variables according to mode.

if(mode == 1) valid\_linear\_data = true;

else valid\_bilinear\_data = true;

return true;

}

int GraphWindow::**rsquComparator**()

{

// This function is used to compare all the R squared values which have been

// calculated, it determines which two R squared values overall have the best

// goodness of fit value.

double smallest\_val = 1.0;

int smallest\_index = 0;

double total[r\_squared[0].size()];

for (int i = 0; i < r\_squared[0].size(); i ++)

{

total[i] = r\_squared[0][i] - r\_squared[1][i];

try {total[i] = fabs(total[i]);}

catch (std::exception&)

{

throwErrorBox("Error Occurred when finding absolute value of double.");

return 0;

}

if (total[i] < smallest\_val)

{

// If the current R squared value is smaller than

// the smallest value, update the variable.

total[i] = smallest\_val;

// This is used to determine where the breakpoint exists.

smallest\_index = i;

}

}

// Returns the breakpoint.

return smallest\_index;

}

void GraphWindow::**on\_actionColour\_triggered**()

{

changeLineColour();

}

void GraphWindow::**on\_pushButton\_plot\_clicked**()

{

plotLinear();

}

void GraphWindow::**on\_pushButton\_clear\_2\_clicked**()

{

clearGraph();

}

void GraphWindow::**on\_pushButton\_graphoptions\_clicked**()

{

graphOptions();

}

void GraphWindow::**on\_actionCurrent\_Plot\_Toolbar\_toggled**(bool arg1)

{

ui->dockWidget->*setVisible*(arg1);

}

void GraphWindow::**on\_actionLinear\_Regression\_triggered**()

{

on\_pushButton\_inputfile\_clicked();

}

void GraphWindow::**on\_pushButton\_data\_clicked**()

{

// Used for bessel feature which has been removed.

}

void GraphWindow::**on\_pushButton\_options\_clicked**()

{

// Used for bessel feature which has been removed.

}

void GraphWindow::**on\_pushButton\_clear\_clicked**()

{

// Used for bessel feature which has been removed.

}

void GraphWindow::**on\_actionData\_triggered**()

{

// Used for bessel feature which has been removed.

}

void GraphWindow::**on\_checkBox\_clicked**(bool checked)

{

const\_term = checked;

}

void GraphWindow::**on\_actionColour\_axis\_triggered**()

{

changeAxisColour();

}

void GraphWindow::**on\_actionGraph\_Options\_triggered**()

{

graphOptions();

}

void GraphWindow::**on\_actionData\_Points\_Colour\_triggered**()

{

// Used to change the colour of the data points.

points\_colour = QColorDialog::getColor(Qt::black, this, tr("Pick Colour Of Points"));

point\_pen.setColor(points\_colour);

if (first\_pass)

return;

else if (auto\_change == true && bilinear\_data == false)

plotLinear();

else if (auto\_change == true && bilinear\_data == true)

plotBilinear();

}

void GraphWindow::**on\_actionData\_Points\_Size\_triggered**()

{

// Used to change the size of the data points.

ChangeThickness edit(size\_crosses, 1, this);

edit.*exec*();

size\_crosses = edit.thickness;

if (first\_pass)

return;

else if (auto\_change == true && bilinear\_data == false)

plotLinear();

else if (auto\_change == true && bilinear\_data == true)

plotBilinear();

}

void GraphWindow::**on\_toolButton\_axiscolour\_clicked**()

{

changeAxisColour();

}

void GraphWindow::**on\_toolButton\_linecolour\_clicked**()

{

changeLineColour();

}

void GraphWindow::**on\_actionThickness\_triggered**()

{

// Changes thickness of line.

ChangeThickness edit(pen\_thickness, 0, this);

edit.*exec*();

pen\_thickness = edit.thickness;

plot\_pen.setWidth(pen\_thickness);

if (first\_pass)

return;

else if (auto\_change == true && bilinear\_data == false)

plotLinear();

else if (auto\_change == true && bilinear\_data == true)

plotBilinear();

}

void GraphWindow::**on\_actionConstant\_Term\_toggled**(bool arg1)

{

const\_term = arg1;

if (const\_term) ui->checkBox->setCheckState(Qt::Checked);

else ui->checkBox->setCheckState(Qt::Unchecked);

}

void GraphWindow::**on\_actionClear\_Graph\_triggered**()

{

clearGraph();

}

void GraphWindow::**on\_actionPlot\_triggered**()

{

plotLinear();

}

void GraphWindow::**on\_pushButton\_refresh\_clicked**()

{

updateScene();

}

void GraphWindow::**on\_pushButton\_usage\_clicked**()

{

// Shows the help dialog pictures for linear data.

HelpDialog \*dialog = new HelpDialog(4 , this);

dialog->setImage(":/help/help/linear\_help1.PNG");

dialog->setImage(":/help/help/linear\_help2.PNG");

dialog->setImage(":/help/help/linear\_help3.PNG");

dialog->setImage(":/help/help/linear\_help4.PNG");

dialog->onLaunch();

dialog->show();

}

void GraphWindow::**on\_pushButton\_inputbilineardata\_clicked**()

{

if(inputData(2))

tableFillBilinear();

}

void GraphWindow::**on\_pushButton\_plotbilineardata\_clicked**()

{

// Plots the bilinear data.

plotBilinear();

// Prints the x0 line.

x0DottedLine();

}

void GraphWindow::**on\_checkBox\_autoupdate\_toggled**(bool checked)

{

auto\_change = checked;

}

void GraphWindow::**on\_pushButton\_usage\_2\_clicked**()

{

// Shows the help dialog pictures for bilinear graph usage.

HelpDialog \*dialog = new HelpDialog(7, this);

dialog->setImage(":/help/help/bilinear\_help1.PNG");

dialog->setImage(":/help/help/bilinear\_help2.PNG");

dialog->setImage(":/help/help/bilinear\_help3.PNG");

dialog->setImage(":/help/help/bilinear\_help4.PNG");

dialog->setImage(":/help/help/bilinear\_output1.PNG");

dialog->setImage(":/help/help/bilinear\_output2.PNG");

dialog->setImage(":/help/help/bilinear\_output3.PNG");

dialog->onLaunch();

dialog->show();

}

void GraphWindow::**on\_pushButton\_refresh\_2\_clicked**()

{

on\_pushButton\_refresh\_clicked();

}

void GraphWindow::**on\_pushButton\_clear\_3\_clicked**()

{

clearGraph();

}

void GraphWindow::**on\_checkBox\_autoupdate\_2\_toggled**(bool checked)

{

auto\_change = checked;

}

void GraphWindow::**on\_toolButton\_linecolour\_2\_clicked**()

{

changeLineColour();

}

void GraphWindow::**on\_toolButton\_axiscolour\_2\_clicked**()

{

changeAxisColour();

}

void GraphWindow::**on\_checkBox\_2\_toggled**(bool checked)

{

dotted\_line = checked;

x0DottedLine();

}

void GraphWindow::**on\_actionBilinear\_Regression\_triggered**()

{

if(inputData(2))

tableFillBilinear();

}

void GraphWindow::**on\_actionPlot\_Bilinear\_Line\_triggered**()

{

plotBilinear();

x0DottedLine();

}

void GraphWindow::**on\_actionCut\_Line\_at\_x0\_triggered**(bool checked)

{

limit\_line = checked;

}

void GraphWindow::**on\_actionLinear\_triggered**()

{

on\_pushButton\_usage\_clicked();

}

void GraphWindow::**on\_actionBilinear\_triggered**()

{

on\_pushButton\_usage\_2\_clicked();

}

void GraphWindow::**on\_checkBox\_3\_clicked**(bool checked)

{

// Used to either show or not show the x0 breakpoint line.

limit\_line = checked;

if (first\_pass)

return;

else if (bilinear\_data == false)

plotLinear();

else if (bilinear\_data == true)

plotBilinear();

}

#include "addnewdata.h"

#include "ui\_addnewdata.h"

AddNewData::**AddNewData**(double start\_value, double end\_value, int number\_of\_values, int function\_order, QWidget \*parent) :

QDialog(parent),

ui(new Ui::AddNewData)

{

ui->setupUi(this);

setWindowFlags(Qt::Dialog | Qt::MSWindowsFixedSizeDialogHint);

ui->doubleSpinBox\_startval->setValue(start\_value);

ui->doubleSpinBox\_endval->setValue(end\_value);

ui->spinBox\_numofval->setValue(number\_of\_values);

ui->spinBox\_functorder->setValue(function\_order);

}

AddNewData::~***AddNewData***()

{

delete ui;

}

void AddNewData::***closeEvent***(QCloseEvent \*)

{

start\_value = 0;

end\_value = 0;

number\_of\_values = 0;

function\_order = 0;

}

void AddNewData::**on\_buttonBox\_accepted**()

{

this->start\_value = ui->doubleSpinBox\_startval->value();

this->end\_value = ui->doubleSpinBox\_endval->value();

this->number\_of\_values = ui->spinBox\_numofval->value();

this->function\_order = ui->spinBox\_functorder->value();

index = ui->comboBox\_functype->currentIndex();

}

void AddNewData::**on\_buttonBox\_rejected**()

{

start\_value = 0;

end\_value = 0;

number\_of\_values = 0;

function\_order = 0;

}

#include "changethickness.h"

#include "ui\_changethickness.h"

ChangeThickness::**ChangeThickness**(int current\_val, int mode, QWidget \*parent) :

QDialog(parent),

ui(new Ui::ChangeThickness)

{

type = mode;

ui->setupUi(this);

setWindowFlags(Qt::Dialog | Qt::MSWindowsFixedSizeDialogHint);

thickness = current\_val;

ui->spinBox\_thickness->setValue(thickness);

}

ChangeThickness::~***ChangeThickness***()

{

delete ui;

}

void ChangeThickness::**on\_spinBox\_thickness\_valueChanged**(int arg1)

{

thickness = arg1;

}

void ChangeThickness::**on\_pushButton\_default\_clicked**()

{

if (type) thickness = 3;

else thickness = 1;

ui->spinBox\_thickness->setValue(thickness);

}

#include "graphoptions.h"

#include "ui\_graphoptions.h"

GraphOptions::**GraphOptions**(QWidget \*parent) :

QDialog(parent),

ui(new Ui::GraphOptions)

{

ui->setupUi(this);

setWindowFlags(Qt::Dialog | Qt::MSWindowsFixedSizeDialogHint);

// Axis variables

// Axis variables

this->x\_pix\_min = -400;

this->x\_pix\_max = 400;

this->y\_pix\_min = -400;

this->y\_pix\_max = 400;

this->x\_startval = -5;

this->y\_startval = -5;

this->tick\_increment = 1;

this->x\_ticks = 10;

this->y\_ticks = 10;

defaultValues();

}

GraphOptions::~***GraphOptions***()

{

delete ui;

}

void GraphOptions::**defaultValues**()

{

ui->spinBox\_xmin->setValue(x\_pix\_min);

ui->spinBox\_xmax->setValue(x\_pix\_max);

ui->spinBox\_ymin->setValue(y\_pix\_min);

ui->spinBox\_ymax->setValue(y\_pix\_max);

ui->spinBox\_xstart->setValue(x\_startval);

ui->spinBox\_ystart->setValue(y\_startval);

ui->spinBox\_increments->setValue(tick\_increment);

ui->spinBox\_xnumticks->setValue(x\_ticks);

ui->spinBox\_ynumticks->setValue(y\_ticks);

}

void GraphOptions::***closeEvent***(QCloseEvent \*)

{

x\_pix\_min = ui->spinBox\_xmin->value();

x\_pix\_max = ui->spinBox\_xmax->value();

y\_pix\_max = ui->spinBox\_ymax->value();

y\_pix\_min = ui->spinBox\_ymin->value();

x\_ticks = ui->spinBox\_xnumticks->value();

y\_ticks = ui->spinBox\_ynumticks->value();

tick\_increment = ui->spinBox\_increments->value();

x\_startval = ui->spinBox\_xstart->value();

y\_startval = ui->spinBox\_ystart->value();

}

void GraphOptions::**on\_buttonBox\_accepted**()

{

x\_pix\_min = ui->spinBox\_xmin->value();

x\_pix\_max = ui->spinBox\_xmax->value();

y\_pix\_max = ui->spinBox\_ymax->value();

y\_pix\_min = ui->spinBox\_ymin->value();

x\_ticks = ui->spinBox\_xnumticks->value();

y\_ticks = ui->spinBox\_ynumticks->value();

tick\_increment = ui->spinBox\_increments->value();

x\_startval = ui->spinBox\_xstart->value();

y\_startval = ui->spinBox\_ystart->value();

}

#include "helpdialog.h"

#include "ui\_helpdialog.h"

HelpDialog::**HelpDialog**(int image\_count, QWidget \*parent) :

QWidget(parent),

ui(new Ui::HelpDialog)

{

ui->setupUi(this);

setWindowFlags(Qt::Dialog | Qt::MSWindowsFixedSizeDialogHint);

num\_of\_images = image\_count;

help\_images.resize(num\_of\_images);

pre\_index = 0;

index = 0;

}

HelpDialog::~***HelpDialog***()

{

delete ui;

}

void HelpDialog::**setImage**(QString path)

{

help\_images[pre\_index] = QPixmap(path);

pre\_index++;

}

void HelpDialog::**onLaunch**()

{

ui->label->setPixmap(help\_images[0]);

if (index == num\_of\_images-1)

ui->pushButton->*setVisible*(false);

}

void HelpDialog::**on\_pushButton\_clicked**()

{

index++;

ui->label->setPixmap(help\_images[index]);

if (index == num\_of\_images-1)

ui->pushButton->*setVisible*(false);

}

#include "graphwindow.h"

#include <QApplication>

int main(int argc, char \*argv[])

{

QApplication a(argc, argv);

GraphWindow w;

a.setWindowIcon(QIcon(":/icons/img/scattericon.png"));

w.setWindowState(Qt::WindowMaximized);

w.show();

return a.exec();

}

#include "qgraphicsaxisitem.h"

QGraphicsAxisItem::**QGraphicsAxisItem**(char mode, qreal pix\_min, qreal pix\_max, int num\_of\_ticks, qreal tick\_increment, qreal tick\_start, QColor colour)

{

// Assigns values to private variables.

this->pix\_min = pix\_min;

this->pix\_max = pix\_max;

this->mode = mode;

this->num\_of\_ticks = num\_of\_ticks;

this->tick\_increment = tick\_increment;

this->tick\_start = tick\_start;

axis\_colour = colour;

n\_pix = (pix\_max-pix\_min)/(double)num\_of\_ticks;

margin\_x = 30;

margin\_y = 50;

height\_rect = 100;

length\_offset = 0;

arrow\_pix\_offset = 15;

tick\_length = 5;

significant\_digit = 2;

}

QRectF QGraphicsAxisItem::***boundingRect***() const

{

QRectF boundingRect;

if (mode == 'x')

{

qreal length\_box = pix\_max - pix\_min;

boundingRect.setRect(pix\_max - margin\_x, - margin\_y, length\_box + length\_offset, height\_rect);

}

else

{

qreal height\_box = pix\_max - pix\_min;

boundingRect.setRect(- height\_rect, - (pix\_max+length\_offset), height\_rect, height\_box + length\_offset);

}

return boundingRect;

}

void QGraphicsAxisItem::***paint***(QPainter \*painter, const QStyleOptionGraphicsItem \*, QWidget \*)

{

// Used to draw the axis onto the screen.

qreal current\_value = pix\_min;

qreal current\_tick = tick\_start;

QString text;

// Number of pixels between tick and value.

int label\_offset = 20;

painter->setPen(axis\_colour);

if (mode == 'x')

{

// 0 specifies the coordinates that the line is drawn from in this case (pix\_min, 0), (pix\_max, 0).

painter->drawLine(pix\_min , 0, pix\_max, 0);

// Arrow at end of axis.

painter->drawLine(pix\_max - arrow\_pix\_offset, arrow\_pix\_offset, pix\_max, 0);

painter->drawLine(pix\_max - arrow\_pix\_offset, -arrow\_pix\_offset, pix\_max, 0);

// For ticks.

for (int i = 0; i < num\_of\_ticks; i++)

{

painter->drawLine(current\_value, 0, current\_value, -tick\_length);

// Text item of type float with decimal places of significant\_digit.

text=QString::number(current\_tick, 'f', significant\_digit);

painter->drawText(current\_value, label\_offset, text);

current\_value += n\_pix;

current\_tick += tick\_increment;

}

}

else if(mode == 'y')

{

// Y axis rotate for ease.

painter->rotate(180);

// 0 specifies the coordinates that the line is drawn from in this case (0, pix\_min), (0, pix\_max).

painter->drawLine(0, pix\_min, 0, pix\_max);

// Arrow at end of axis.

painter->drawLine(arrow\_pix\_offset, pix\_max - arrow\_pix\_offset, 0, pix\_max);

painter->drawLine(-arrow\_pix\_offset, pix\_max - arrow\_pix\_offset, 0, pix\_max);

// For ticks.

for (int i = 0; i < num\_of\_ticks; i++)

{

painter->drawLine(0, current\_value, tick\_length, current\_value);

// Text item of type float with decimal places of significant\_digit.

text=QString::number(current\_tick, 'f', significant\_digit);

painter->rotate(-180);

painter->drawText(-label\_offset-10, -current\_value, text);

painter->rotate(180);

current\_value += n\_pix;

current\_tick += tick\_increment;

}

}

}

qreal QGraphicsAxisItem::**pixel\_to\_data**()

{

return n\_pix/tick\_increment;

}

#include "tablemodel.h"

TableModel::**TableModel**(QObject \*parent) : QAbstractTableModel(parent)

{

}

void TableModel::**populateData**(const QVector<double> x\_data, const QVector<double> y\_data)

{

x.clear();

y.clear();

x = x\_data;

y = y\_data;

}

int TableModel::***rowCount***(const QModelIndex &) const

{

return x.size();

}

int TableModel::***columnCount***(const QModelIndex &) const

{

return 2;

}

QVariant TableModel::***data***(const QModelIndex &index, int role) const

{

if (!index.isValid() || role != Qt::DisplayRole) {

return QVariant();

}

if (index.column() == 0) {

return x.at(index.row());

} else if (index.column() == 1) {

return y.at(index.row());

}

return QVariant();

}

QVariant TableModel::***headerData***(int section, Qt::Orientation orientation, int role) const

{

if (role == Qt::DisplayRole && orientation == Qt::Horizontal) {

if (section == 0) {

return QString("X");

} else if (section == 1) {

return QString("Y");

}

}

return QVariant();

}

void TableModel::**clear**()

{

this->beginResetModel();

this->endResetModel();

}

#ifndef ADDNEWDATA\_H

#define ADDNEWDATA\_H

#include <QDialog>

namespace **Ui** {

class **AddNewData**;

}

class **AddNewData** : public QDialog

{

Q\_OBJECT

public:

explicit **AddNewData**(double start\_value, double end\_value, int number\_of\_values, int function\_order, QWidget \*parent = nullptr);

~***AddNewData***();

void ***closeEvent***(QCloseEvent \*event);

// Variables.

double start\_value;

double end\_value;

int number\_of\_values;

int function\_order;

int index;

private slots:

void **on\_buttonBox\_accepted**();

void **on\_buttonBox\_rejected**();

private:

Ui::AddNewData \*ui;

};

#endif // ADDNEWDATA\_H

#ifndef CHANGETHICKNESS\_H

#define CHANGETHICKNESS\_H

#include <QDialog>

namespace **Ui** {

class **ChangeThickness**;

}

class **ChangeThickness** : public QDialog

{

Q\_OBJECT

public:

explicit **ChangeThickness**(int current\_val, int mode, QWidget \*parent = nullptr);

~***ChangeThickness***();

int thickness;

int type;

private slots:

void **on\_spinBox\_thickness\_valueChanged**(int arg1);

void **on\_pushButton\_default\_clicked**();

private:

Ui::ChangeThickness \*ui;

};

#endif // CHANGETHICKNESS\_H

#ifndef GRAPHOPTIONS\_H

#define GRAPHOPTIONS\_H

#include <QDialog>

#include <QMessageBox>

namespace **Ui** {

class **GraphOptions**;

}

class **GraphOptions** : public QDialog

{

Q\_OBJECT

public:

explicit **GraphOptions**(QWidget \*parent = nullptr);

~***GraphOptions***();

void **defaultValues**();

void ***closeEvent***(QCloseEvent \*);

qreal x\_pix\_min, x\_pix\_max;

qreal y\_pix\_min, y\_pix\_max;

int x\_ticks, y\_ticks;

int tick\_increment;

int x\_startval, y\_startval;

private slots:

void **on\_buttonBox\_accepted**();

private:

Ui::GraphOptions \*ui;

};

#endif // GRAPHOPTIONS\_H

#ifndef GRAPHWINDOW\_H

#define GRAPHWINDOW\_H

#include <QMainWindow>

#include <QContextMenuEvent>

#include <QMenu>

#include <QGraphicsScene>

#include <QGraphicsView>

#include <QDebug>

#include <QVector>

#include <QList>

#include <QGraphicsLineItem>

#include <QGraphicsItemGroup>

#include <QMessageBox>

#include <QFileDialog>

#include <QTextStream>

#include <QColorDialog>

#include <QPen>

#include <QTableView>

#include <gsl/gsl\_sf\_bessel.h>

#include <gsl/gsl\_fit.h>

#include <gsl/gsl\_vector.h>

#include "qgraphicsaxisitem.h"

#include "adddata.h"

#include "graphoptions.h"

#include "tablemodel.h"

#include "changethickness.h"

#include "helpdialog.h"

QT\_BEGIN\_NAMESPACE

namespace **Ui** { class **GraphWindow**; }

QT\_END\_NAMESPACE

class **GraphWindow** : public QMainWindow

{

Q\_OBJECT

public:

**GraphWindow**(QWidget \*parent = nullptr);

~***GraphWindow***();

// Handles right click events on graph.

void ***contextMenuEvent***(QContextMenuEvent \*event);

// Close event.

void ***closeEvent***(QCloseEvent \*event);

// Removes scroll bars.

void **scrollbars**(bool active);

// Clears the current graph.

void **clearGraph**();

// Update graph/scene.

void **updateScene**();

// Checks input data is valid and adds to vectors.

bool **processData**(QString line\_read);

// Adds data to tables depending on which data has been entered.

void **tableFillLinear**();

void **tableFillBilinear**();

// Calculates and plots the lines accordingly.

void **plotLinear**();

void **plotBilinear**();

// Creates the axis, depending on the data.

void **createAxis**(int num\_x\_ticks, int num\_y\_ticks, int start\_x, int start\_y);

// Calculates the statistics associated with the lines, R values etc.

void **statistics**(int mode, int index);

// Changes colour of the item accordingly.

void **changeLineColour**();

void **changeAxisColour**();

// Used for generating a custom graph, not recommended, instead let graph generate automatically.

void **graphOptions**();

// Updates the colour display boxes in the tool bar.

void **updateColours**();

// Used to spawn the File Ui for entering a file.

bool **inputData**(int mode);

// Used to compare r squared values for the bilinear lines.

int **rsquComparator**();

// Creates the graph and plots it.

void **buildGraph**(int array\_index);

// Used for the breakpoint line.

void **x0DottedLine**();

// Error handling

void **throwErrorBox**(QString error\_msg);

private slots:

void **on\_actionData\_triggered**();

void **on\_actionEnable\_Scrollbars\_toggled**(bool arg1);

void **on\_pushButton\_clear\_clicked**();

void **on\_pushButton\_inputfile\_clicked**();

void **on\_actionColour\_triggered**();

void **on\_pushButton\_plot\_clicked**();

void **on\_pushButton\_clear\_2\_clicked**();

void **on\_pushButton\_graphoptions\_clicked**();

void **on\_actionCurrent\_Plot\_Toolbar\_toggled**(bool arg1);

void **on\_actionLinear\_Regression\_triggered**();

void **on\_pushButton\_options\_clicked**();

void **on\_pushButton\_data\_clicked**();

void **on\_checkBox\_clicked**(bool checked);

void **on\_actionColour\_axis\_triggered**();

void **on\_actionGraph\_Options\_triggered**();

void **on\_actionData\_Points\_Colour\_triggered**();

void **on\_actionData\_Points\_Size\_triggered**();

void **on\_toolButton\_axiscolour\_clicked**();

void **on\_toolButton\_linecolour\_clicked**();

void **on\_actionThickness\_triggered**();

void **on\_actionConstant\_Term\_toggled**(bool arg1);

void **on\_actionClear\_Graph\_triggered**();

void **on\_actionPlot\_triggered**();

void **on\_pushButton\_refresh\_clicked**();

void **on\_pushButton\_usage\_clicked**();

void **on\_pushButton\_inputbilineardata\_clicked**();

void **on\_pushButton\_plotbilineardata\_clicked**();

void **on\_checkBox\_autoupdate\_toggled**(bool checked);

void **on\_pushButton\_usage\_2\_clicked**();

void **on\_pushButton\_refresh\_2\_clicked**();

void **on\_pushButton\_clear\_3\_clicked**();

void **on\_checkBox\_autoupdate\_2\_toggled**(bool checked);

void **on\_toolButton\_linecolour\_2\_clicked**();

void **on\_toolButton\_axiscolour\_2\_clicked**();

void **on\_checkBox\_2\_toggled**(bool checked);

void **on\_actionBilinear\_Regression\_triggered**();

void **on\_actionPlot\_Bilinear\_Line\_triggered**();

void **on\_actionCut\_Line\_at\_x0\_triggered**(bool checked);

void **on\_actionLinear\_triggered**();

void **on\_actionBilinear\_triggered**();

void **on\_checkBox\_3\_clicked**(bool checked);

private:

// Vaiables for axis.

int offset = 10;

qreal x\_pix\_min, x\_pix\_max;

qreal y\_pix\_min, y\_pix\_max;

int x\_ticks, y\_ticks;

int tick\_increment;

int x\_startval, y\_startval;

int size\_crosses;

// Variables for linear regression plot.

double c0[2], c1[2], cov00[2], cov01[2], cov11[2], sumsq[2];

double x0;

QVector <QVector <double> > r\_squared;

bool bilinear\_data = false;

bool limit\_line = false;

//

bool auto\_change = false;

QPixmap line\_pixmap = \*new QPixmap(16, 16);

QPixmap axis\_pixmap = \*new QPixmap(16, 16);

// Variables for file.

QString file\_path;

bool valid\_linear\_data = false;

bool valid\_bilinear\_data = false;

// Pen colour.

QColor line\_colour;

QColor axis\_colour;

QColor points\_colour;

// Main user interface / main window.

Ui::GraphWindow \*ui;

int line\_no = 1;

// Creates graphics scene for axis.

QGraphicsScene \*scene = new QGraphicsScene(this);

TableModel \*table\_linear = new TableModel(this);

TableModel \*table\_bilinear = new TableModel(this);

// Creates axis.

bool dotted\_line = true;

bool first\_pass = true;

QGraphicsAxisItem \*x\_axis;

QGraphicsAxisItem \*y\_axis;

QGraphicsItemGroup \*line1;

QGraphicsItemGroup \*line2;

QGraphicsItemGroup \*x0\_line;

QGraphicsLineItem \*plot\_item;

QGraphicsLineItem \*point\_item;

QGraphicsLineItem \*x0\_item;

bool const\_term = true;

QPen plot\_pen = \*new QPen();

QPen point\_pen = \*new QPen();

QPen \*axis\_pen = new QPen();

int pen\_thickness;

// Pixels to data.

qreal x\_pix\_to\_data, y\_pix\_to\_data;

// Data vectors.

QVector<double> x\_data, y\_data;

QVector <double> x\_plot, y\_plot;

// Error handling.

QMessageBox \*error\_box = new QMessageBox();

};

#endif // GRAPHWINDOW\_H

#ifndef HELPDIALOG\_H

#define HELPDIALOG\_H

#include <QWidget>

namespace **Ui** {

class **HelpDialog**;

}

class **HelpDialog** : public QWidget

{

Q\_OBJECT

public:

explicit **HelpDialog**(int image\_count, QWidget \*parent = nullptr);

~***HelpDialog***();

void **setImage**(QString path);

void **onLaunch**();

private slots:

void **on\_pushButton\_clicked**();

private:

Ui::HelpDialog \*ui;

QVector <QPixmap> help\_images;

int num\_of\_images;

int index, pre\_index;

};

#endif // HELPDIALOG\_H

#ifndef QGRAPHICSAXISITEM\_H

#define QGRAPHICSAXISITEM\_H

#include <QGraphicsItem>

#include <QRectF>

#include <QPainter>

#include <QString>

class **QGraphicsAxisItem** : public QGraphicsItem

{

public:

// Constructor.

**QGraphicsAxisItem**(char mode, qreal pix\_min, qreal pix\_max, int num\_of\_ticks, qreal tick\_increment, qreal tick\_start, QColor colour);

// Required becuase of the inherited QGraphicsItem.

QRectF ***boundingRect***() const override;

// Paints the item, in this case the axis.

void ***paint***(QPainter \*painter, const QStyleOptionGraphicsItem \*, QWidget \*) override;

// Used for comverting the pixel values into data.

qreal **pixel\_to\_data**();

private:

QRectF boundingRectx, boundingRecty;

QColor axis\_colour;

// Maximum and minimum pixel values.

qreal pix\_min, pix\_max;

// Number of pixels in a given interval.

qreal n\_pix;

// Defines either x or y for axis.

char mode;

// Number of ticks along the axis.

int num\_of\_ticks;

// Tick length.

int tick\_length;

// The increments along each axis.

qreal tick\_increment;

// The starting tick value.

qreal tick\_start;

// The margin that surrounds the axis.

int margin\_x;

int margin\_y;

int height\_rect;

int length\_offset;

// Sets the arrow pixel offset.

int arrow\_pix\_offset;

// Number of decimal places value label has.

int significant\_digit;

};

#endif // QGRAPHICSAXISITEM\_H

#ifndef TABLEMODEL\_H

#define TABLEMODEL\_H

#include <QAbstractTableModel>

class **TableModel** : public QAbstractTableModel

{

public:

**TableModel**(QObject \*parent);

void **populateData**(const QVector <double> x\_data, const QVector <double> y\_data);

int ***rowCount***(const QModelIndex &parent = QModelIndex()) const Q\_DECL\_OVERRIDE;

int ***columnCount***(const QModelIndex &parent = QModelIndex()) const Q\_DECL\_OVERRIDE;

QVariant ***data***(const QModelIndex &index, int role = Qt::DisplayRole) const Q\_DECL\_OVERRIDE;

QVariant ***headerData***(int section, Qt::Orientation orientation, int role = Qt::DisplayRole) const Q\_DECL\_OVERRIDE;

void **clear**();

private:

QVector <double> x, y;

};

#endif // TABLEMODEL\_H