**A level Computer Science Project**

2022

**My details**  
Name .…………………………………………………………………………………………………………………………… Peter Metcalfe  
Candidate number ……………………………………………………………………………...………………………...................... 1651  
School ………………………………………………………………………………………………….…………….. The Cottesloe School  
Centre number ……………………………………………………………………………………………………………………….. ?52141?

**Contents**Introduction …………………………………………………………………………………………….……………………………………………………… 2  
Stakeholders …………………………………………………………………………………………………………………………………………………... 2

**Analysis**  
Knowledge …………..……………………………………………..………………………………………………………………………………………..… 3  
Existing programs ..………………………………………………………………………………….…………………………………………………….. 7  
Stakeholder(s) / User(s) needs ……………………….………………………………………………………………………..………………..…... 7  
Computational methods …………………………………………………………………………………………..…………………………………… 7  
Hardware and software requirements ………………………….………………………………………………………………………..…….... 7  
Success Criteria ……………….………………………………………………………………………..…………………………………………………… 8

**Design**  
Structure diagram …………………………………………………………………………………………………………………………………………. 9  
Modules and processes ………………………………………………………………………………………………………………………….……. 10 Interface design…………………………………………………………………………………………………………………………………………….. 11  
Validation of inputs ………………………………………………………………………………………………………………………………..……. 12  
Pseudocode …………………………………………………………………………………………………………………………………………….…… 13  
Class diagrams …………………………………………………………………………………………………………………………………………….. 17  
Identifying test data to be used during iterative development …………………………………………………………………... 18

**Developing The Coded Solution**Milestone 1 – User interface ………………………………………………………………………………………………………………………... 21  
Milestone 2 – Polynomials …………………………………………………………………………………………………………………………... 22  
Milestone 3 – Circles ……………………………………………………………………………………………………………………………………. 23  
Milestone 4 – Points …………………………………………………………………………………………………………………………………….   
Milestone 5 – Regression Line ……………………………………………………………………………………………………………………..   
Milestone 6 – Graph details …………………………………………………………………………………………………………………………   
Milestone 7 – Mandelbrot Set ……………………………………………………………………………………………………………………..

**Evaluation**

**Introduction**  
Students and teachers today are often expected to learn and draw graphs, for exams, textbooks, practice and for real life examples too. Graphs are an important and a significant part of many subjects, especially at A level, in maths, physics, chemistry, biology, and many more…   
To assist students and teachers in their use of graphs and maths in general, I intend to create an easy-to-use graphing calculator application.  
Existing graphing software is limiting and complicated and I have experienced this first-hand. It is much easier for students to understand a question or problem by simply being able to visualise it and see what is happening.  
My application will enable the user to plot graphs and shapes with ease. I hope to make this a tool for students to further their understanding, as well as a tool for teachers to display solutions and problems. My program can include more tools and functions than existing software making it more useful for students (like myself) taking subjects that go into more depth (such as further maths), where you may be looking at Argand diagrams as well as ordinary cartesian grids.  
I now need to research what features students and teachers may want to see/use on a graphing calculator and what is important to include to make sure that the application is useful and valued.

**1**

**Stakeholder(s)**As mentioned above, my application will aim to help students and teachers. Therefore, I am going to make sure I am in constant contact with my maths teacher (Mrs. Evans) and my peers who specifically take subjects suited to using my application. This way I will be able to get live feedback on what I am creating, ensuring a good result for the end user.

After a short interview with my math teacher, Mrs Evans, I had answers to some of my initial questions regarding what she thought a ‘graphical calculator’ should have, and what one could have.

**2**

**Analysis**

**Prior Knowledge needed**

For this project I am going to need to know and be able to use confidently:  
1. Linear, quadratic, cubic, quartic, etc, functions.  
2. Find roots of said equations.  
3. Find turning points of said equations.  
4. Calculate sin, cos, and tan graphs.  
5. Calculate the equation of a regression line.  
6. Be able to differentiate.  
7. Calculate the Mandelbrot set

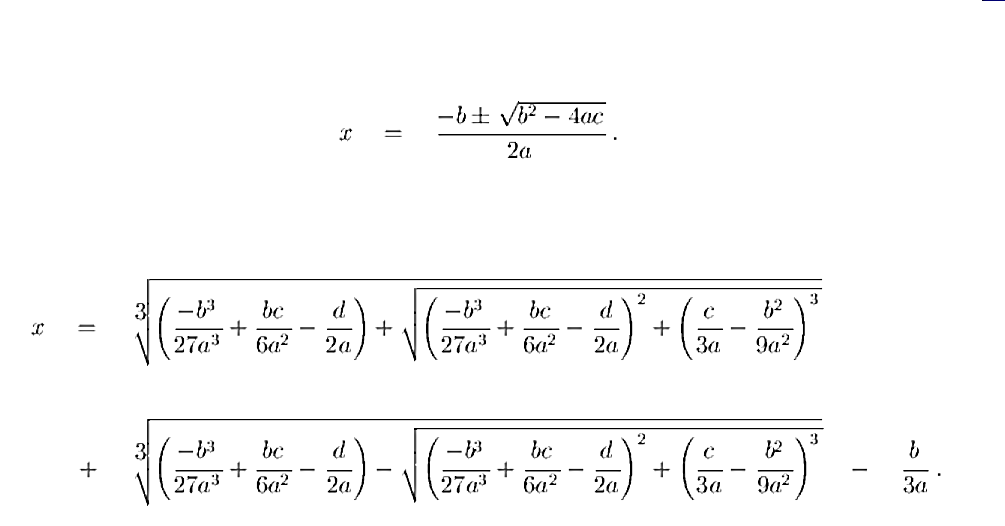
**(1-3) Linear equation**: y = ax + b  
Where ‘b’ is the y intercept  
root = -b / a

**(1-3) Quadratic equations**: y = ax2 + bx + c  
Where ‘c’ is the y intercept

Turning point = ( (-c/2\*b) , (b\*(-c/2\*b)2 + c\*(\*(-c/2\*b)+d) ) 🡪 I derived this from ‘completing the square’

**(1-3) Cubic equations**: ax3 + bx2 +cx + d  
Where ‘d’ is the y incept

When b = 0 and there is only one root, it can be found with this formula:  
root = cube\_root((-((b3)/(27\*(a3)))+((b\*c)/(6\*(a2)))-(d/(2\*a)))-(((-((b3)/(27\*(a3)))+((b\*c)/(6\*(a2)))-(d/(2\*a)))2+((c/(3\*a))-((b2)/(9\*(a2))))3)0.5))+cube\_root((-((b3)/(27\*(a3)))+((b\*c)/(6\*(a2)))-(d/(2\*a)))+  
(((-((b3)/(27\*(a3)))+((b\*c)/(6\*(a2)))-(d/(2\*a)))2+((c/(3\*a))-((b2)/(9\*(a2))))3)0.5))-b/(3\*a))



Source = https://math.vanderbilt.edu/schectex/courses/cubic/

**3**

When b = 0 and there **are** 3 roots, you can solve them using parts of the equation. This will give you 3 complex numbers which you have to transform to real numbers, using the fact that the three complex numbers and their complex conjugate form equilateral triangles.

My pseudo code:

part1 = (b3) / (27\*(a3)) #Parts of cubic formula above

part2 = (b\*c) / (6\*(a2))

part3 = d / (2\*a)

part4 = c / (3\*a)

part5 = (b2) / (9\*(a2))

mainP1 = -part1 + part2 - part3

mainP2 = (part4 - part5)3

a = mainP1 #real part of complex number

b = (abs(mainP12 + mainP2))1/2 #imaginary part of complex number

r = ( a2 + b2 )1/2 # magnitude of vector

IF a < 0: #deciding what quadrant vector is in

theta1 = pi - (arctan((b)/(a)) / 3) #to do that, need to find angle

ELSE: #angle calculation depends on quadrant

theta1 = arctan((b)/(a)) / 3

IF theta1 > (5\*pi)/6: #bottom right quadrant

theta2 = 2\*pi - (theta1 + ((2/3)\*pi)) #find angle 2

ELSE IF: theta1 < pi/3: #top left quadrant

theta2 = pi - (theta1 + ((2/3)\*pi))

ELSE IF: theta1 < (5\*pi)/6 and theta1 > pi/3: #bottom left quadrant

theta2 = -( (2\*pi) - (theta1 + ((2/3)\*pi)) )

theta3 = (theta1 - ((2/3)\*pi)) #find angle 3

z1 = ((r1/3)\*(cos(theta1))) + ((r1/3)\*(cos(theta1))) #conveting complex number into real

x1 = z1

IF a\*-a == a: #negative

z2 = ((r1/3)\*(cos(theta2))) + ((r1/3)\*(cos(theta2))) #conveting complex number into real

ELSE: #positive

z2 = -(((r1/3)\*(cos(theta2))) + ((r1/3)\*(cos(theta2)))) #conveting complex number into real

x2 = z2

z3 = ((r1/3)\*(cos(theta3))) + ((r1/3)\*(cos(theta3))) #conveting complex number into real

x3 = z3

RETURN [x1, x2, x3] #returning 3 roots

Source of information = Video: https://www.youtube.com/channel/UC1\_uAIS3r8Vu6JjXWvastJg  
(My working code, derived from video source)

**4**

Another way to try and find the roots, of any polynomial, is to scan the x-axis, subbing in y. Every time y changes sign, I know the graph has just crossed the x-axis, so one of the roots must be **approximately** this x value. Here is my pseudo code for my algorithm to find the solutions to a cubic:

Nroots = 3 #number of roots

i = 0 #our step

y1 = 0

y1s = " " #will store the sign +/-

y2s = " "

limit = 80

roots = [ 0 , 0 , 0 ] #this is where we’ll store the roots we find

WHILE i < limit \* 2: #checks 80 either side of y axis (160)

x= i - limit

y1 = (a\*(x3)) +(b\*(x2))+(c\*x)+d #finding y on each iteration

y1s = str(y1)[0] # looking at the first character. + or - ?

IF y1s != "-": #looking for a change in y sign, that’s when y ~ 0

y1s = "+"

IF y1s == "-" and y2s == "+":

Nroots -= 1

roots[Nroots] = round(x,8) #add the root to list and round result as approximation

IF y1s == "+" and y2s == "-":

Nroots -= 1

roots[Nroots] = round(x,8) #add the root to list and round result as approximation

y2s = y1s #storing previous value of y to compare

i = i + 0.001

IF Nroots == 0: #saves us checking more y values if we've already found 3 roots

i = 400\*2 #size of axis

break

END WHILE

IF Nroots > 0: #return null if couldn’t find any roots

print("I can't find all the roots :( ")

roots = [None, None, None]

RETURN roots

Turning points of a cubic:   
First I need to check if it has any, by taking the determinant of the first derivative checking weather is greater than 0.   
Then I can calculate the turning points.   
= ( ((-2\*b)+sqrt((4\*(b2))-(12\*(a\*c))))/(6\*a) , a\*((TPx1)3) + b\*((TPx1)2) + (c\*(TPx1)) + d )  
= ( ((-2\*b)-sqrt((4\*(b2))-(12\*(a\*c))))/(6\*a) , a\*((TPx2)3) + b\*((TPx2)2) + (c\*(TPx2)) + d )  
I derived these from solution of the first derivative.

**5**

**(4) Calculating Sine, Cosine and Tangent:**For these I can just use the math.sin() function or .cos() or .tan() (I will have to import math). However this will give me a result is radians so I can convert it to degrees by multiplying by pi/180.

**(5) Least squares regression line:**This is basically the mathematical line of best fit for a set of given points. So to calculate the equation of a line you need the gradient and a point on that line.   
So to calculate a point on the regression line, you are you just going to take an average of all the points on the line = ( average(x), average(y) )  
Sxy = ∑( x - average (x) )\*( y – average(y) )  
Sxx = ∑( x – average (x) )2

Gradient = Sxy / Sxx

Now you have the gradient and a point on that line, you can plot it.

Source of information = Video: <https://youtu.be/MVJv01xr9lM>

**(6) Differentiate:**

if you have the function f(x) = ax2 + bx + c

f’(x) = 2ax + b  
You times the coefficient by the power and then reduce the power by 1.

**(7) Mandelbrot Set:**Your just need to sub the complex number c in the function f(c) = z2 + c where z increments by some step (e.g. 0.1) on each pass. If f(c) is ever > 2 you know it is unstable, but if after a set limit of iterations it still isn’t greater than 2 you say it is stable and you plot that as black on an Angrand diagram.

**6**

**Existing programs**  
There are two existing programs that I have researched: ‘Desmos’ and ‘GeoGebra’.  
Desoms = <https://www.desmos.com/calculator>  
GeoGebra = https://www.geogebra.org

Features of these programs:

* Online / web based
* Draw / plot multiple functions on same axis
* Plot tables
* Label graphs
* Colour graphs
* Find roots
* Find turning points
* Find intersections
* Zoom

Issues / limitations:

* Don’t do complex numbers
* Don’t calculate regression line
* Don’t draw Mandelbrot set
* Can’t draw / sketch
* Don’t work offline

I hope to include all the features above as well as all the limitations, to try and make my program more useful and helpful. These current programs don’t consider complex numbers, and this is where I will try to make my program exceed. However, one limitation of my program will be that it is in 2D where as GeoGebra does have the ability to go into 3 dimensions. However this could be something I could consider adding if I have enough time.

**Stakeholder(s) / User(s) needs**

After having a discussion with my client (Mrs Evans – Maths teacher), we have fallen upon a short list of problems that are the things that I need to solve.  
She said that a graphical calculator should be able to calculate, at least, some basic sums and be able to draw, at least, some basic graphs. She also added that some of the things that a graphical calculator should output are the turning points, roots, intercepts, and asymptotes.   
Improvements for the traditional calculator that she wanted to see, were a clearer user interface. She said that current calculators are too complicated, especially for beginners and that it could be a lot simpler and quicker to use. She also added that these calculators are hard to navigate and find all of its different functions, so this is also something I can consider when designing mine.

**Computational methods**  
Computers a great for reducing the need for human interaction. That is why a computer will be great for performing this task / program. Less human interaction will mean less human error and less wasted time as computers are extremely well handled to deal with data, quickly. Computers can also store this data reliably allowing you to keep you graph plot forever.

**Hardware and software requirements**I am going to use Visual Studio code to program python on a windows operating system. The reason for python is the extremely useful libraries that can be used, for example: I will mainly be using the ‘matplotlib’ library to draw and ‘math’ library to perform certain mathematical calculations, as well as the ‘Tkinter’ library for the GUI that will take inputs and produce outputs.   
I will program my application using object orientated code to make it easier to reuse code and to debug.  
The hardware needed are laptop, monitor, mouse, and a keyboard.

**7**

**Success Criteria**

|  |  |
| --- | --- |
| **Criteria** | **Justification** |
| Have several functions allowing the user to freely select the graphing function they would like to perform. This should include as a minimum: polynomials, circles, regression lines and complex numbers. | This gives the user control and allows them to achieve the result they were looking for. The more graphing functions, the more freedom the user has and the more useful my program will be. |
| The user can input graph functions to be processed. | This is vital for the program to be able to know what to output, including the graph and extra details like roots and intersections. |
| The user can clearly see the graph they inputted on the screen and be able to make necessary observations form it. This can include the ability to manipulate the graph by zooming and moving it around the screen. | The reason for this is to really allow the user to make the most of the application. Being able to visualise the inputted graph functions should aid the user in their understanding. |
| The user should be able to clear the set of axis. | This is important if the user plots several graphs on the same axis as it could get messy. The user should be able to clear this without having to restart the application. |
| Allow the user to plot a variety of polynomials, at least up to cubic functions. | The purpose of this is that the user may have several types of graph they would like to input, and I would like my program to be able to handle this. |
| The program should output data about the inputted graph, including roots, turning points and intersections. | Extra information about the graph will be useful for the user as my stakeholders has mentioned. It not only makes my program more useful but also helps the user with mathematical problems too. |
| The program must recognise and alert the user when the user has inputted something that is invalid. | This is important so that the user knows they have done something incorrect. My program must be interactive and malleable in this way to prevent a lack of communication between the user and my program, resulting in a lack of interest and usage. |
| The user must be able to distinguish between different graphs drawn on the same axis, by colour coordination and/or labelling. | This allows the user to use the application easier and with less effort. It stops the set of axis getting out of control and messy when lots of graphs are drawn. |
|  |  |

**8**

**Design**

**Structure diagram**

Graphing application

These green boxes represent the pages my application will have.  
  
 Blue boxes represent the buttons/functions on that page.

Draw wave function

Draw wave

Zoom

Move graph

Validate inputs

Output error message function

Complex plane

Draw Mandelbrot set

Draw complex number

Graph page

Help page

Regression line function

Point function

Polynomial function

Circle function

Draw regression line

Draw point

Draw circle

Draw polynomial

The Orange boxes are an indication of the objects of code that I will have to write.

**9**

**Structure diagram explanation**  
The diagram shows that my application will have 3 main pages, leading form the home page. The graph page will have all the functions I am offering, on the traditional cartesian coordinates system, whereas the complex page will be plotted on the complex plane. Finally, the help page will guide the user of different functionalities of my program if necessary.

Off of each sub-page (i.e. draw polynomial) will be where the user inputs their values to be drawn. Once they have finished using this tool, they will be directed back to the home page, to use any of the other tools.

The ‘zoom’ and ‘move’ functions will be of the ‘graph’ and ‘complex’ page and will be used by the user to manipulate the graphs they draw. This allows the user to make clearer observations about their graphs and provides more useability.

**Modules and processes**

The graph page. This will consist of all the functions that I am offering to the user, which can be plotted on the usual cartesian coordinates grid.

First we have, drawing polynomials. Here the user will have the ability to input functions up to at least cubic equations, maybe more depending on the time I have left. This should then draw the graph of the inputted function and output the useful data about the graph.   
This will allow the user to make suitable observations about their graph and should allow them to understand the numbers in their function. Added with the extra outputted data about their graph should allow them to know almost everything about it, including its shape, roots, intersections, turning points, etc… All this information will allow students to help answer problems and allow teachers to clearly present their problems and / or solutions, providing an overall better understanding.

Next there is drawing circles, this should take input values for the radius and centre coordinate, then plot the circle on the axis. This allows for a wider variety of questions / problems to be drawn using my program, making it more accessible for users.

The draw point module will simply draw a point (small circle) at the inputted coordinate. This gives the user the ability to plot specific points that may be useful for their problem. This ability also lets the user plot many points, maybe data collected in a science experiment, and see what shape the points make. This power to observe the shape of graphs at an instant will be very useful for students and teachers to come to conclusions about their experiment, for example.

The regression line function will calculate the mathematical bast line of best fit for the user’s inputted data points or coordinates. This function is very helpful for subjects involving graphs and collecting data. Most modern calculators can provide the function of this line, but most cannot show the shape of this graph, but my application will perform both tasks. Using the maths described above in the ‘Prior knowledge’ section, I will use the user’s points to calculate the function of the line (like calculators can), and then I shall draw the graph.

Moving on to the complex plane page, the first function will be to plot complex numbers. This a useful tool for students taking further maths as these are the students that will have to learn to use complex numbers. On top of this, my program will output the modulus and argument of these complex numbers, which I know from experience could be very useful as these terms are used a lot.

**10**

Finally, the Mandelbrot set. This will output the same image, as there is only one Mandelbrot set, however it will allow the user to control the colour scheme used and the pixel quality. The reason for this is there are a lot of points to calculate and the smaller the pixel count the quicker it will take to render. This ability to visualise the Mandelbrot set may come in useful for visualisation and possibly artistic reasons.Other modules that maybe aren’t as important may involve the colour changing function that changes the colour of the graphs being plotted, as well as an input validation object that will check the input values for the different functions of my program that all need separate requirements, or the function that that clears the set of axis when the user commands this, etc…

Home Page

Graph

Complex Plane

Help

Polynomial Page

Plot

Back

a =  
  
b =   
  
c =   
  
d =

y = ax3 + bx2 +cx + d

This is the initial design for the application, with 3 simple clear buttons on the home page.  
  
  
This is the polynomial page, where the user will input their values of a, b, c, d.  
All the pages where the user has to input values are probably going to look like this.

0 1 2 3 4 5 6 7 8

7  
  
6  
  
5  
  
4  
  
3  
  
2  
  
1  
  
0

\_ y = (x-4)2 \_ y = x - 1

This an example of what the graph might look like.   
  
Notice each line plotted has a colour coordinated label, so the user can keep track.

**Interface design**

**11**

**Validation of inputs**As my GUI contains text-based user inputs, I need to validate / check these inputs before I can use them to do any kind of calculations or drawing. This is what I will check…

|  |  |  |
| --- | --- | --- |
| **Validation check** | **How it works** | **Example** |
| Left blank | Checks weather the text bar was left empty. If it is, it makes the input equal to zero. (sanitisation) | Input: ‘ ‘ Output: 0 |
| String check | If the input requires an integer / float, this will reject any strings or characters and alert the user. | Input: ‘hello’ Output: [Pop-up message] [‘This input is not valid’] |
| Range check | This checks to see whether the inputted values are within the range of the axis to be drawn on. | Input: ‘100000’ Output: [Pop-up message] [‘Input too large’] |
| Sign check | Similar to range check, but some inputs cannot be negative, like the circle radius. | Input: ‘-54’ Output: [Pop-up message] [‘Input must be greater than 0’] |

**Pseudocode**

**12**

**Drawing polynomials**This algorithm needs to start at the left-hand side of the axis (e.g. x = -200) and on each loop increment x by some step. On each loop, the algorithm needs to calculate the y coordinate relating to that x value. To do this, it will sub the value of x into the inputted graph function. It will then store all the points calculated in a list to be plotted once the loop has ended at the right-hand side of the axis (x = 200).

The variable ‘step’ controls the size of the increment of x, each time round the loop. I have made this a variable as I may want to change this depending on the type of graph. This variable is going to be responsible for how quickly but also how well it draws the graph, as bigger ‘steps’ means more jagged curves.

I have also made the axis size a variable as I may also want to change this, either at the user’s discretion or I may set it depending on the graph. If the graph inputted is only small, I only need a small set of axis to draw it on, so this will save time.

I have also made sure to include the ‘check\_vars’ routine that will validate the input variable and return them as floats, to be used in calculations.

This algorithm will be one function of a ‘draw’ object. This object may also contain the other algorithms that follow, such as the ‘draw circle’ function and the ‘draw point’ function.

INPUT a  
INPUT b

INPUT c

INPUT d

a , b , c , d = SUBROUTINE check\_vars( “polynomial”, a , b , c , d )  
  
step = 0.1  
axis = 200  
x = -axis  
Xvalues = [ ]  
Yvalues = [ ]  
  
WHILE x <= axis:

Yvalues.append( (a \* x3 )+ (b \* x2 ) + (c \* x) + d )  
Xvalues.append( x )  
x += step

END WHILE  
SUBROUTINE change\_colour()  
PLOT( Xvalues ,Yvalues )

**Draw circle**   
For the algorithm to draw the circle, it is very similar to the one thet draws polynomials, but I effectively have to do it twice as the circle will give two outputs each time I sub x, into to find y. This means, I start at the left-hand side of the x axis, increment x until I reach the right-hand side, then go back again, decreasing x until I am back at the left-hand side. If I didn’t do this, I would just make a semi-circle.  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
 **Draw point**This is a very simple algorithm that will take an input, of the point, and plot it, after checking it.

**13**

INPUT radius  
INPUT centreX  
INPUT centreY  
  
radius , centreX , centreY = SUBROUTINE check\_vars( “circle”, radius , centreX , centreY )  
  
step = 0.1  
axis = 200  
Xvalues = [ ]  
Yvalues = [ ]  
  
x = -axis  
WHILE x <= axis:  
 Yvalues.append( square\_root( radius2 - x2 – ( 2 \* centreX \* x ) – centreX2 ) - centreY   
 Xvalues.append( x )  
 x += step  
END WHILE  
  
x = axis  
WHILE x <= -axis:  
 Yvalues.append( - square\_root( radius2 - x2 – ( 2 \* centreX \* x ) – centreX2 ) - centreY   
 Xvalues.append( x )  
 x -= step  
END WHILE  
SUBROUTINE change\_colour()  
PLOT( Xvalues, Yvalues )

INPUT x  
INPUT y  
x , y = SUBROUTINE check\_vars( “point”, x , y)  
SUBROUTINE change\_colour()  
PLOT( x , y )  
  
  
  
1

**Regression line**This algorithm will calculate the mathmatical perfect line of best fit. To do this, the user will input at least 2 coorinates and the this algorithm will ouput the equation of the line of best fit, using the maths descirbed in the ‘Prior knowledge’ section. It can then plot this graph using a similar alogrithm to the polynomial psueodcpde.

**14**

Xvalues = [ ]  
Yvalues = [ ]  
Xpoints = [ ]  
Ypoints = [ ]   
X = 0  
Y = 0  
Sxx = 0  
Sxy = 0  
axis = 200  
step = 1  
  
FOR user\_inputs:  
 INPUT FLOAT = x  
 INPUT FLOAT = y  
 Xpoints.append( x )  
 Ypoints.append( y )  
END FOR  
   
FOR x in RANGE ( 0 , Xpoints\_length):  
 X += Xpoints[ x ]  
 Y += Ypoints[ x ]  
END FOR  
  
X = X / Xpoints\_length  
Y = Y / Ypoints\_length  
  
FOR p in RANGE ( 0 , Xpoints\_length ):  
 Sxy += ( Xpoints[ p ] - X ) \* ( Ypoints[ p ] - Y )  
 Sxx += ( Xpoints[ p ] -X )2  
c = Sxy / Sxx  
d = Y – ( c \* X )  
END FOR  
  
WHILE x <= axis:

Yvalues.append( (c \* x) + d )  
Xvalues.append( x )  
x += step

END WHILE  
  
SUBROUTINE change\_colour()  
PLOT( Xvalues ,Yvalues )

**15**

**Complex number plotting**This is exactly the same as the ‘draw point’ algorithm except I am plotting it on the complex plane.  
  
  
  
  
  
  
  
  
**Mandelbrot set #**Here, ‘MS’ refers to ‘Mandelbrot Set’  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
In this algorithm, I am going through loop, checking every coordinate to see whether it is ‘stable’ or not. I have described the maths in the ‘Prior knowledge’ section, but the comments descirbe each line. Once it has plotted the lines betweens the points calculated, it should produce the Mandelbrot set image, with a nice colour gradient.  
  
  
**Class diagrams**  
Arrows show inheritance:  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
**Identifying test data to be used during interative devlopment**  
  
I have a lot of functions and objects to write so here I have made a clear list of the milestones I would like to reach when it comes to creating my coded solution:

step = 0.01  
axis = 200  
stable = FALSE  
colour\_before = “ “  
colour = “ “   
WHILE x <= ( axis - step ):  
 x += step #x and y increments, to get every point  
 y = -axis  
 WHILE y <= axis:  
 y += step  
 Xscale = x / ( 0.5 \* axis ) #makes it bigger as the MS is only between -2 🡪 2  
 Yscale = y / ( 0.5 \* axis )  
 answer = complex( 0 , 0 )  
 FOR i IN RANGE ( 0 , 100 ):  
 answer = ( answer2 + complex( Xscale , Yscale ) ) #MS sum  
 IF ( answer.real2 ) + (answer.imag2) >= 4: #circle with radius of 2, stable?  
 stable = FALSE  
 IF INPUT(“colour”) == "red": #user picks form colour scheme

FOR c IN range ( 0 , len( MBredScale ) ):

IF i > MSredScale[ c ][ 0 ]: #picks colour gradient

colour = MBredScale[ c ][ 1 ]

BREAK

IF INPUT(“colour”) == "blue":

FOR c IN RANGE ( 0 , len( MBblueScale ) ):

IF i > MSblueScale[ c ][ 0 ]:

colour = MBblueScale[ c ][ 1 ]

BREAK  
 BREAK  
 ELSE:  
 stable = TRUE  
 IF stable == TRUE: #if stable, colour black  
 colour = ‘black’  
 IF colour\_before != colour or y >= axis:  
 PLOT( x , y , colour\_before )  
 colour\_before = colour

IF y >= 400:   
 PLOT( ( x + step ) , -axis )  
 colour\_before = " "

**16**

INPUT a  
INPUT b  
a , b = SUBROUTINE check\_vars( “point”, a , b)  
SUBROUTINE change\_colour()  
PLOT( a , b )

Draw

a: float  
b: float  
c: float  
d: float  
colour\_index: float  
limit = integer

Draw\_polynomial  
Draw\_circle  
Draw\_point  
Draw regression line

Check\_vars

Limit: integer

check\_polynomial  
check\_circle  
check\_point  
check regression line

Main

Container: string  
filemenu: string  
menubar: string

\_\_innit\_\_  
Show\_frame

Home\_page

Label:  
button1:  
button2:  
button3:

\_\_innit\_\_

Graph\_page

Label: string  
button1: string  
button2: string  
button3: string

\_\_innit\_\_

Complex\_page

Label: string  
button1: string  
button2: string  
button3: string

\_\_innit\_\_

Draw\_complex

a: float  
b: float   
colour\_index: float  
limit: integer

Draw\_complex\_number   
Draw\_mandelbrot\_set

Graph\_deails

Graph: string

Search\_roots()  
Search\_turningPoints()  
linear()  
quadratic()  
cubic()

Help\_page

Label: string  
button1: string

\_\_innit\_\_

**17**

1. Creating the user interface
2. Drawing polynomials: Taking and validating an input and drawing the assosiate graph
3. Drawing circles
4. Drawing points
5. Drawing regression lines
6. Displaying detailed information along with the ouputted plot
7. Drawing mandelbrot set

|  |  |  |
| --- | --- | --- |
|  | **Milestone 1: User Interface** |  |
| **Test Number** | **What is being tested and inputs** | **Expected output(s)** |
| 1 | That buttons do as they say.  Test – Press them. | The button does as it syas.  e.g. takes you to that page. |
| 2 | Text inputs work, and reject unusable inputs. Test – Enter text, try numbers, letters, strings, 0, negative numbers. | When correct data type entered, the assosiated reaction should occur. If unwanted data inputted, an error message occurs. |
| 3 | Checkboxes work.  Test – tick them, and untick them. | The checkboxe should have the intended result. |
| 4 | Scroll bars work as intended.  Test – Scroll them. | The scroll bars should have the intended result. |

|  |  |  |
| --- | --- | --- |
|  | **Milestone 2: Drawing Polynomials** |  |
| **Test Number** | **What is being tested and inputs** | **Expected output(s)** |
| 1 | Test linear functions. e.g. 2x - 3 | The correct line should be plotted on the correct posistion on the axis. e.g. gradient of 2 and y-interept of -3. |
| 2 | Test quaratic functions.  e.g. 2x^2 +3x -5 | The correct line should be plotted on the correct posistion on the axis. |
| 3 | Test cubic functions e.g. x^3 +2x^2 +3x +5 | The correct line should be plotted on the correct posistion on the axis. |
| 4 | Test the 3 previus functions but with negative coefficents. | The correct line should be plotted on the correct posistion on the axis. |
| 5 | Test the colour of the graph changes when a new function is plotted. | The colours should cycle as different functions are plotted, so the user can distinguish between them. |
| 6 | Test limits of axis. e.g. make the y-intercept greater than or less than the axis. | If the graph doesn’t fit on the axis, either display an error message or resize the axis and plot. |

**18**

|  |  |  |
| --- | --- | --- |
|  | **Milestone 3: Circles** |  |
| **Test Number** | **What is being tested and inputs** | **Expected output(s)** |
| 1 | Draw cicle at origin.  e.g. radius = 10, centreX = 0, centreY = 0. | This should draw a circle at the origin with a radius of 10. |
| 2 | Draw circles with a differrent centre. e.g. radius = 10, centreX = 2, centreY = -5. | This should draw a circle at this position with that radius. |
| 3 | Test inputing negative radii. e.g. Radius = -23 | This should ouput a suitable error massage, as this is not paossible to draw. |
| 4 | Test axis limits.  E.g. Input a centre off the axis. | If the graph doesn’t fit on the axis, either display an error message or resize the axis and then plot. |
| 5 | Test the colour of the graph changes when a new circle is plotted. | The colours should cycle as different things are plotted, so the user can distinguish between them. |

|  |  |  |
| --- | --- | --- |
|  | **Milestone 4: Drawing Points** |  |
| **Test Number** | **What is being tested and input(s)** | **Expected output(s)** |
| 1 | Draw a point.  e.g x= 10, y =2 | This should draw a point at (10,2). |
| 2 | Test negatives.  e.g. x=-10, y-2 | This should draw a point at (-10,-2). |
| 3 | Test mixture of both negatives and positives.  e.g. x=10, y=-2 | This should draw a point at (10, -2). |
| 4 | Test point off the axes scale.  e.g. x=200, y=1000 | This should show a suitable error message. |
| 5 | Test inputting a string.  e.g. “Hello” | This should output a suitable error message. |

|  |  |  |
| --- | --- | --- |
|  | **Milestone 5: Regression Line** |  |
| **Test Number** | **What is being tested and input(s)** | **Expected output(s)** |
| 1 | Test what values can be inputted.  e.g. x=10, y=5 | Programm should accept this input and ask the user for the next point. |
| 2 | Test inputtig a string instead of an integer.  e.g. “woow” | This should output a suitable error message. |
| 3 | Test the correct regression line is outputed after at least 2 points have been inputted. | The correct regression line should be outputted. |
| 4 | Test point off the axes scale.  e.g. x=200, y=1000 | This should show a suitable error message. |

**19**

|  |  |  |
| --- | --- | --- |
|  | **Milestone 6: Outputting data about graph** |  |
| **Test Number** | **What is being tested and input(s)** | **Expected output(s)** |
| 1 | Test if the outputted roots are correct for linear graphs.  E.g. Input = 2x | E.g. Output = 0 |
| 2 | Test if the outputted roots are correct for quadratic graphs.  E.g. Input = (x+4)2 | E.g. Output = -4 |
| 3 | Test if the outputted roots are correct for cubic graphs.  E.g. Input = 5x3 +5x2 +5x + 5 | E.g. Output = -1 |
| 4 | Test if the outputted roots are approximately correct for other graphs.  E.g. Input = x7 - 2x3 + 2x - 2 | E.g. Output = ~1.158 |
| 5 | Test If the outputted turning points are corect.  E.g. Input = 5x3 +5x2 +5x + 5 | E.g. Output = (0,5) |
| 6 | Test if the outputted graph type is correct.  E.g. Input = x3 + 3x2- 2 | E.g. Output = ‘Cubic’ |
| 7 | Test that the colour of the graph changes. | Every gragh should have a different colour to the one plotted before it. |

**20**

|  |  |  |
| --- | --- | --- |
|  | **Milestone 7: Mandelbrot Set** |  |
| **Test Number** | **What is being tested and input(s)** | **Expected output(s)** |
| 1 | Test the plotted Mandelbrot set is acurate/to scale. | Should be plotted on imaginary axis between 2 and -2. |
| 2 | Test that the inputted colour scheme is correclty used. | If the user selects the Red scheme. The MS should be Red. |

**Developing The Coded Solution**

**Prototyping**

**21**

The first thing I am going to do is create a prototype of my application. This will have simpler graphics and slower speeds, but I am going to do this to make sure I can get all the maths working first. I am going to write some ‘simple’ python code using the ‘Turtle’ Library that will be able to do all the functions I have described above. I will then use parts of this code to create my final product which will be of a better quality and faster to run.

Overall, I am going to be doing each milestone twice. Once for the prototype, and once for the final product. I will do all the testing and verification on the version 2, so I will skip this for this prototype.

**Milestone 1 – User Interface:**

A picture containing table

Description automatically generatedFor this prototype I am going to keep the interface simple as this is not what I need to test. I will create a simple ‘Turtle Canvas’ to draw graphs on, and a simple input bar to perform tasks.

#

Now, a ‘turtle’ draws the axis, and this menu comes up. The functions don’t do anything yet, but you can see what I plan to get working.

**Milestone 2 – Drawing polynomials:**

Table

Description automatically generated with low confidenceI now plan to get the ‘turtles’ to draw curves and graphs on the set of axis shown on the previous page. For the purpose of this prototype, I will only go up to cubic equations. So the user can input function with exponents up to and including cubic. This will be in the format ax3 + bx2 + cx + d, where a, b, c and d are the constants that the user will input.

Here is the simple interface that I implemented to get these inputs.

Now that I have the function to draw, I need to draw it.  
I thought of a couple of different ways of doing this:  
- I could scan the x axis, substituting in every x value into the function,   
to find the y value, then plot this.  
- I could do the same thing as the point above, but to increase   
resolution, differentiate the function, find the gradient at this x value,   
and draw a small line, at this gradient, between the previous x value and  
the next.  
- I could do the same thing, but scan the y axis instead of the x axis. However, this will often output two results, for example if there is a ‘U’ shaped curve, so this will involve two plots.

Graphical user interface, text, application

Description automatically generatedI ended up going for the first point as it was easiest to implement and the resolution could be increased by decreasing the step at which I scan the x axis (However, this increases the time taken to draw it). I tried implementing all the methods, but you couldn’t tell the apart, so I ended up choosing on complexity.

In this code:  
- ‘tim’ and ‘tom’ are the ‘turtles’ that draw the graph. To make it quicker I used two ‘turtles’ that draw half the graph each.   
- ‘step’ is the rate at which they scan the x axis. E.g. (step =1) 1 🡪 2 🡪 3 (step = 0.2) 1 🡪 1.2 🡪 1.4  
- ‘Axis’ is the size of the axis, show in this case it is = 200.  
- ‘.xcor()’ / ‘.ycor()’ is another built-in function that returns the ‘turtles’ position.

A picture containing shape

Description automatically generatedChart, radar chart

Description automatically generatedThe results:

**22**

**Milestone 3 – Circles:**

Graphical user interface, text, application, email

Description automatically generatedThis works in a similar way to the polynomial function, to draw them. The only difference is that I do it twice, once getting the positive y values and once getting the negative y values. When I plot both together I get a circle.

**23**

You can see here that I ended up using the ‘try’ and ‘except’ instructions. This is because I wanted to avoid errors being thrown up when trying to square root negatives.  
‘cy’ and ‘cx’ refer to the centre coordinates of the circle (centre X and centre Y) and ‘r’ refers to the radius; these are all inputs of the user.

Diagram

Description automatically generatedResults:

**Milestone 4 – Drawing points:**