

## **Coursework 1 PH20018 Submission**

This report is the 1st assignment for PH20018. It includes answers to the following problems:  
Diffraction Limit of a Telescope, Root-Finding and Multivariable Equations.

The report is divided in three sections, each of which solves one of the three problems.

It is important to note that all code pertaining to this report was written in Visual Studio 2019 for visualizing purposes, but was run through command line using Min GW compiler in Windows computer.

### **Table of Contents**

<b>Part 1: Diffraction Limit of a Telescope</b>	<b>2</b>
1.a Calculating the Bessel functions	2
Design	2
Results and Discussion	6
1.b Calculating the diffraction pattern	8
Calculating $J_1(x)$	9
Calculating $I(r)$	8
Calculating the $r$ values and troubleshooting issues within our $x$ data	11
Results and discussion	12
<b>Part 2: Root finding</b>	<b>16</b>
2.a Limits of numerical precision and possible sources of rounding errors in root finding	16
2.b Flowchart of a program to find the real roots of the equation	17
2.c Root finding program	20
Design	20
Results	21
Discussion on the result accuracy	22
<b>Part 3: Multivariable equations</b>	<b>23</b>
Mathematical reasoning	23
Equation 1	23
Equation 2	24
Mathematics behind the design	24
Code design	25
Results and discussion	27
<b>Bibliography</b>	<b>30</b>
<b>Code associate with file "Generalized_version3.c"</b>	<b>31</b>

## **Part 1: Diffraction Limit of a Telescope**

### **1.a Calculating the Bessel functions**

To answer the posed question, a C based program that contains a function  $J(m,x)$  that calculates the value of  $J_m(x)$  using the trapezium rule with an  $N = 10000$  and calculated the Bessel functions  $J_0(x)$ ,  $J_1(x)$  and  $J_2(x)$  as a function of  $x$  from  $x = 0$  to  $x = 20$ , was built.

In the following subsections the design steps, results and discussion of such, will be detailed.

### **Design**

For this code the following libraries were used: standard `stdio.h` (standard library), `stdlib.h` library (used for data file transfer, a property that will enable a later plot of the obtained results through Python's `matplotlib.pyplot` library), and the `math.h` library (used to obtain the value of  $\pi$ , elevate numbers to certain powers, and sin and cos functions).

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>

#define M_PI 3.14159265358979323846
```

Figure 1: Lines 1-5 C file 1a.c

The first function of the program (lines 7-18) defines the values inside the integral:

$$J_m(x) = \frac{1}{\pi} \int_0^{\pi} \cos(m\theta - x \sin(\theta)) d\theta$$

in a  $f(\theta, x, m)$  function. This function is useful for future handling of those values in our main code. A wrapper function follows it, to enable a return of the first function's values coherently with the rest of the code.

```
7 //define the function f(x)
8 double f(double theta, double x, double m)
9 {
10     double result;
11     result = cos(m * theta - x * sin(theta));
12     return result;
13 }
14 double f_wrapper(double *params)
15 {
16     // Returns value of f(m, x, theta), theta is in 0th pos ie variable the integration occurs over.
17     return f(params[0], params[1], params[2]);
18 }
19
```

Figure 2: Lines 7-19 C file 1a.c

As you can see most variable types in this program are doubles. This is the case as even though doubles take more memory space, they optimize the calculations for double arithmetic's and given the tasks nature the memory usage employed is not relevant.

Following the wrapper function a linear space function (lines 20-42) is defined with the help of `malloc` functions. It's role is to define the calculated range of  $x$  as well as the desired number of

iterations N through a pointer function in our main. This function prevents certain errors that could be caused by inputting maximum x smaller than the minimum x or vice versa, as that would cause false or no outputs.

Other errors this function is preventive of are runtime errors (line 34) or malloc errors caused by “inexistence” of the result (lines 24 to 28) .

```
20 //linear space construction
21 double *linspace(double start, double stop, int N)
22 {
23     double *result = (double *)malloc(N * sizeof(double));
24     if (!result)
25     {
26         fprintf(stderr, "Failed malloc");
27         exit(-1);
28     }
29     if (stop < start)
30     {
31         fprintf(stderr, "Endpoint & startpoint error");
32         exit(-1);
33     }
34     double current = start, dx = (double)(stop - start) / N;
35     int i;
36     for (i = 0; i < N; ++i)
37     {
38         result[i] = current;
39         current = current + dx;
40     }
41     return result;
42 }
```

Figure 3: Lines 20-42 C file 1a.c

An integral solving function defining the integrated elements follows (lines 44-75). It once again checks for any input range errors, and using pointers and a for loop calculates the integral of an inputted function through the trapezium rule method

```
44 //integration function
45 double integrate(double (*func)(double *), double *params, double a, double b, int N)
46 {
47     double *range = linspace(a, b, N);
48     if (b < a)
49     {
50         fprintf(stderr, "Endpoint & startpoint error");
51     }
52     //calculating h
53     double h = (b - a) / N;
54     double y0_params[] = {a, params[1], params[2]};
55     double yn_params[] = {b, params[1], params[2]};
56     double y0 = func(y0_params);
57     double yn = func(yn_params);
58     double sum = (y0 + yn) / 2;
59     int i;
60     for (i = 0; i < N; ++i)
61     {
62         params[0] = range[i];
63         sum = sum + func(params);
64     }
65     double result = h * sum;
66     return result;
67 }
```

Figure 4: Lines 44-75 C file 1a.c

In lines 77-85 a function that calculates the various Bessel functions  $J(m,x)$ , by inputting the original  $f(\theta,x,m)$  function into the previous integration function while taking into account the desired number of iterations  $N$  as well as including the division of our integrated values by  $\pi$ .

```
77 //calculate the integral
78 double J(double x, double m, int N)
79 {
80     double params[] = {0., x, m};
81     double area = integrate(f_wrapper, params, 0, M_PI, N);
82     //divide by pi and return the integration result
83     double result = area / M_PI;
84     return result;
85 }
```

Figure 5: Lines 77-85 C file 1a.c

Finally in the main function (lines 87-127), pointers are used to format the obtained x and y values into different columns of a newly created txt file. This is done in order to later plot the obtained results using Python's libraries. The x and y values are obtained with the use of for loops that iterate through the different values of m in our Bessel function and all the trapezium rule intervals (N=1000) in our integral function.

As a debugging precaution the code also prints the m iterations to ensure it is was working correctly.

```
87  int main()
88  {
89
90      int N = 1000;
91      double *x_range = linspace(0, 20, N);
92      double **y_vals = (double **)malloc(3 * sizeof(double *));
93      double *y_vals_0 = (double *)malloc(N * sizeof(double));
94      double *y_vals_1 = (double *)malloc(N * sizeof(double));
95      double *y_vals_2 = (double *)malloc(N * sizeof(double));
96
97      y_vals[0] = y_vals_0;
98      y_vals[1] = y_vals_1;
99      y_vals[2] = y_vals_2;
100
101      int i, m;
102      for (m = 0; m < 3; ++m)
103      {
104          printf("\nm = %d", m);
105          for (i = 0; i < N; ++i)
106          {
107              y_vals[m][i] = J(x_range[i], m, 10000);
108          }
109      }
```

Figure 6: Lines 87-109 C file 1a.c

At the end of the main function the program closes the created text file after it's usage.

```
110 //file fprintf for data analysis & modelling
111 putchar('\n');
112 FILE *fp = fopen("data1a.txt", "w");
113 if (!fp)
114 {
115     fprintf(stderr, "Failed to open file");
116     exit(-1);
117 }
118 for (i = 0; i < N; ++i)
119 {
120     fprintf(fp, "%lf, %lf, %lf, %lf\n", x_range[i], y_vals[0][i], y_vals[1][i], y_vals[2][i]);
121 }
122 //close file
123 fclose(fp);
124 return 1;
125 }
```

Figure 7: Lines 110-127 C file 1a.c

## Results and Discussion

This code was written in Visual Studio 2019 for visualizing purposes and was run through command line using Min GW compiler – Windows.

Once ran the command line outputs the  $m$  iterations as expected and creates (or in case it was already created, overwrites) a text file called “data1a.txt”. This text file is expected to contain and contains 4 columns with 1000 lines of data, which correspond to the 1000 intervals of  $x$  between 0 and 20 with respect to the Bessel functions  $J_0(x)$ ,  $J_1(x)$  and  $J_2(x)$ .

Once the data in the text file is plotted, see used Python code below, we obtain the following plot.

```
In [1]: import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

```
In [2]: data = np.genfromtxt("data1a.txt", skip_header = 0 , delimiter = ",")

x = data[:,0]
J0 = data[:,1]
J1 = data[:,2]
J2 = data[:,3]
```

```
In [3]: plt.plot(x,J0)
plt.plot(x,J1)
plt.plot(x,J2)
plt.legend(['J0','J1','J2'])
plt.xlabel('J(m,x)', fontsize=12 )
plt.ylabel('X', fontsize=12)
plt.show()
```

Figure 8: Python 3 plotting code, ran in Jupyter Notebooks

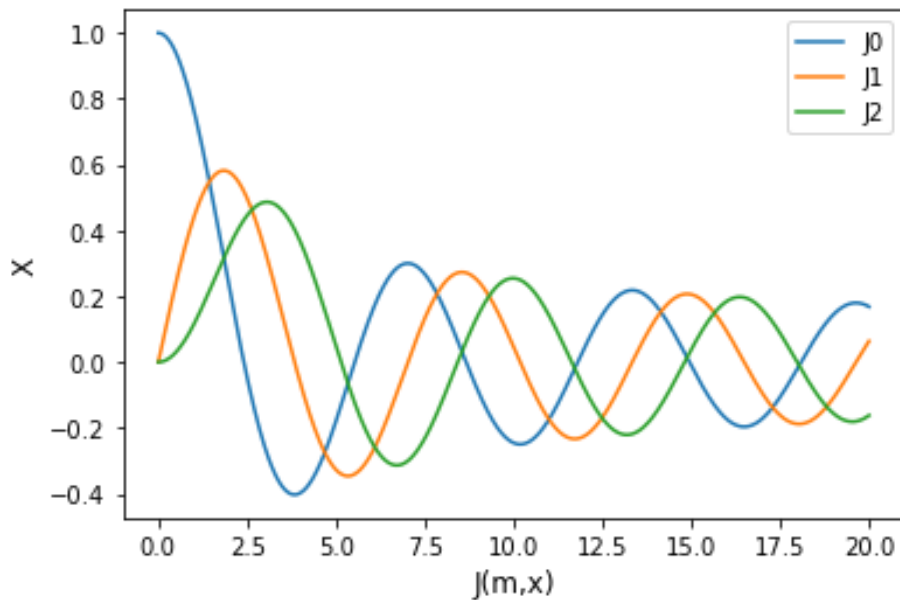


Figure 9: Obtained plot

The obtained results seem consistent with the definition of a first kind Bessel function (WolframMathWorld, n.d.):

“The Bessel function of the first kind  $J_n(x)$  are defined as the solution to the Bessel differential equation:  $x^2 \frac{d^2 y}{dx^2} + x \frac{dy}{dx} + (x^2 + n^2)y = 0$ ”

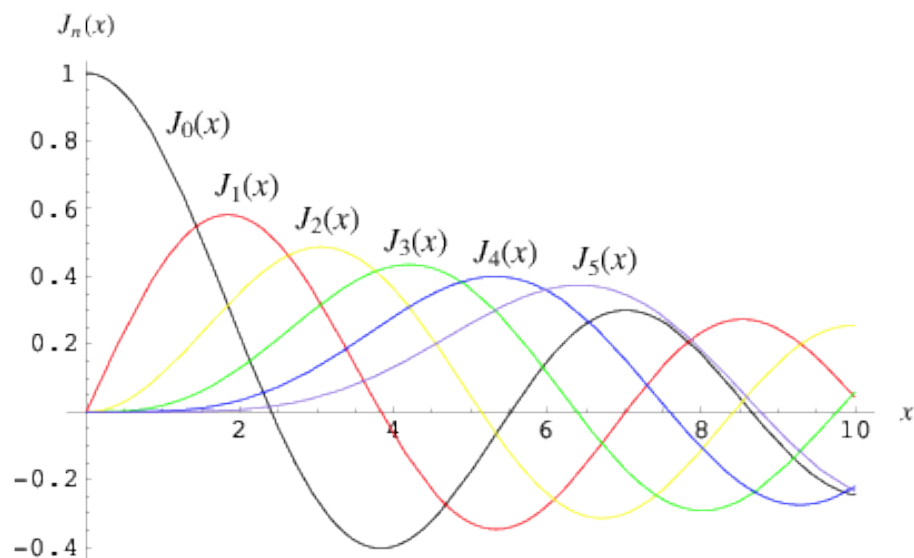


Figure 10: Bessel function plot - Wolfram Alpha reference

### **1.b Calculating the diffraction pattern**

In this section it is assumed that  $I_0 = 1$ , which enables us to interpret the obtained results a rate of  $I_0$ , were this one to be modified. It is also assumed that  $\lambda = 500\text{nm}$  as this is the upper threshold of High-Energy Visible Light.

With this information in the following sections we will discuss a program that calculates the diffraction pattern of a circular lens, for which the focal ratio is 10, using a value  $r$  covering the range  $\pm 25 \mu\text{m}$  and the following equation:

$$I(r) = I_0 \left( \frac{2J_1(x)}{x} \right)$$

#### **Calculating $J_1(x)$**

For this purpose,  $J_1(x)$  will be calculated with the new values of  $x$  obtained through the following equation:

$$x = k \sin(\sigma) = \frac{2\pi}{\lambda} a \frac{r}{R}.$$

After a primary mathematical analysis we can see that:

$$k \sin(\sigma) \frac{R}{a} = \frac{\pi r}{\lambda}$$

Given that  $\frac{R}{2a} = 10$  we can say that:  $k \sin(\sigma) = \frac{\pi r}{10\lambda}$

This implies that  $x = \frac{\pi r}{10\lambda}$ . Given our range of  $r$  is  $r = \pm 25 \times 10^6 \text{m}$ , we obtain a range of  $x = \pm 5\pi$ .

In order to obtain  $J_1(x)$  corresponding to the new  $x$  range, the following code is added to the main function of the program discussed in section 1a:

```
90      int N = 1000;  
91      //lambda 500nm  
92      double l = 10 * 500 / pow(10, 9);  
93      //x with lambda and r +- 25 micro meters  
94      double equation = (M_PI * 25 / (pow(10, 6)));  
95      double beginning = -equation / l;  
96      double ending = equation / l;
```

Figure 11: Lines 90-99 C file 1b\_section1.c

It would most likely have been more efficient and made the code more generalised, for the user to input  $x_{\min}$  and  $x_{\max}$  directly from the command line, nevertheless due to the nature of the this code calculates these itself given the parameters.



Using the new lines 90-96 the program outputs the newly calculated Bessel functions into a text file.

```
109     int i, m;
110     for (m = 0; m < 3; ++m)
111     {
112         printf("\nm = %d", m);
113         for (i = 0; i < N; ++i)
114         {
115             y_vals[m][i] = J(x_range[i], m, 10000);
116         }
117     }
118     //file fprintf for data analysis & modelling
119     putchar('\n');
120     FILE* fp = fopen("data1b_part1.txt", "w");
121     if (!fp)
122     {
123         fprintf(stderr, "Failed to open file");
124         exit(-1);
125     }
126     for (i = 0; i < N; ++i)
127     {
128
129         fprintf(fp, "%lf, %lf, %lf, %lf\n", x_range[i], y_vals[0][i], y_vals[1][i], y_vals[2][i]);
130     }
131
132     //close file
133     fclose(fp);
134     free(y_vals[0]);
135     free(y_vals[1]);
136     free(y_vals[2]);
137     return 1;
138 }
```

Figure 12: Lines 109-138 C file 1b\_section1.c

This text file contains the Bessel function  $J_0(x)$ ,  $J_1(x)$  and  $J_2(x)$  with respect to the chosen  $x$  interval.

### Calculating $I(r)$

The calculation of  $I(r)$  is done in a separate C file, with the following design.

With the use of `stdlib.h` library, the first function of the program reads the previously created data file and outputs the desired data :  $J_1(x)$  and it's corresponding  $x$ , while ignoring other columns.

This function (written between lines 9-35) also ensures the previously created data file has the necessary number of lines ( $N$ ) to avoid any errors.

A  $I(x, J_1)$  function follows (lines 37-43); it calculates the  $I(r)$  corresponding to each  $x$  and  $J_1(x)$  obtained in the data file, according to the equation  $I(r) = I_0\left(\frac{2J_1(x)}{x}\right)$ .

```
9  double** read_file(char* filename, int N)
10 {
11     FILE *fp = fopen(filename , "r");
12     //null stuff
13     double** result = (double **)malloc(2*sizeof(double*));
14     double * x = (double *)malloc(N*sizeof(double));
15     double * j1 = (double *)malloc(N*sizeof(double));
16     if (!result || !x || !j1)
17     {
18         fprintf(stderr, "\n Malloc  error\n");
19         exit(-1);
20     }
21     /*ignores
22     //EOF ends while loop
23     //extra security N size
24     int i =0;
25     while (fscanf(fp, "%lf, %lf, %lf, %lf\n", &x[i], &j1[i]) != EOF && i<N)
26     {
27         i++;
28     }
29     result[0]=x;
30     result[1]=j1;
31
32     fclose(fp);
33
34     return result;
35 }
36
37 double I(double x,double j1){
38     //Io=1
39     double eqt = 2 * j1 / x;
40     double result = pow(eqt, 2);
41     return result;
42 }
43 }
```

Figure 13: Lines 9-43 C file 1b\_section2.c

Finally, in this programs main, the data file obtained previously thanks to the stdlib.h library is read and  $I(r)$  is calculated and printed into a new "data1b\_part2.txt" text file.

The code is designed to exit if there were any issues reading the creation or opening of the previous or new file, and if this is not the case it then iterates through all  $N=1000$  lines of the original data text file to transform them into  $I(r)$  and then output them into a new text file.

```
46 int main()
47 {
48     double **test=read_file("data1b_part1.txt", 1000);
49     int i;
50     FILE* fp2 = fopen("data1b_part2.txt", "w");
51     if (!fp2)
52     {
53         fprintf(stderr, "Failed to open file");
54         exit(-1);
55     }
56     for (i = 0; i < 1000; ++i)
57     {
58         double data = I(test[0][i], test[1][i]);
59         fprintf(fp2,"%lf\t%lf\n", data , test[0][i]);
60     }
61
62     //closing file & free arrays
63     fclose(fp2);
64     free(test[0]);
65     free(test[1]);
66     free(test);
67     return 1;
68 }
69 }
```

Figure 14: Lines 46-69 C file 1b\_section2.c

### Calculating the r values and troubleshooting issues within our x data

This program, nonetheless, has one major issue: it does not take into consideration that x cannot be equal to 0 as we cannot divide by zero. In order to deal with this problem, the calculating values of r are calculated through a for loop that "avoids" values of x=0. In order to make this idea a reality the following changes are made within the code structure:

Firstly, the pi value is defined at the top of our previous code for later usage.

```
4 #include <stdio.h>
5 #include <stdlib.h>
6 #include <math.h>
7 //added to calc r
8 #define M_PI 3.14159265358979323846
~
```

Figure 15: Lines 4-8 C file 1b\_section3.c

Then a function r intaking the x values and transforming them to their corresponding r values, is created. It's results are multiplied  $10^6$  to facilitate the later construction of the r plot axis.

```
45 //reversing x->r
46 double r(double x) {
47     //Io=1
48     double l = 500 / pow(10, 9);
49     double result2 = 10 * x * l / M_PI;
50     //to make it easier to visualize w will show it with nm as it's unit
51     double modified_results2 = result2 * pow(10, 6);
52     return modified_results2;
53 }
```

Figure 16: Lines 45-53 C file 1b\_section3.c

Additionally, this function is added into the main function and creates a new data column within the outputted text file.

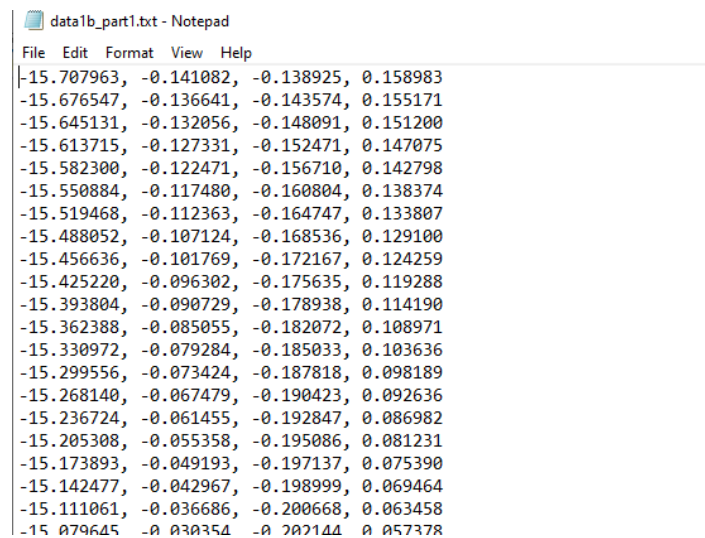
```
56 int main()
57 {
58     double** test = read_file("data1b_part1.txt", 1000);
59     int i;
60     FILE* fp2 = fopen("data1b_part2_version2.txt", "w");
61     if (!fp2)
62     {
63         fprintf(stderr, "Failed to open file");
64         exit(-1);
65     }
66     for (i = 0; i < 1000; ++i)
67     {
68         double data = I(test[0][i], test[1][i]);
69         double x_data = r(test[0][i]);
70         if (x_data != 0)
71         {
72             fprintf(fp2, "%lf\t%lf\n", data, x_data);
73         }
74     }
75 }
76
77 //closing file & free arrays
78 return 1;
79 fclose(fp2);
80 free(test[0]);
81 free(test[1]);
82 free(test);
83 }
```

Figure 17: Lines 56-83 C file 1b\_section3.c

### Results and discussion

The set of data obtained through my first code, contained in a text file named "data1b\_part1.txt" contains not only the x values corresponding to the desired range ( $\pm 5\pi$  which corresponds to approximately  $\pm 15.7$ ) but also the corresponding Bessel functions for  $J_0(x)$ ,  $J_1(x)$  and  $J_2(x)$ .

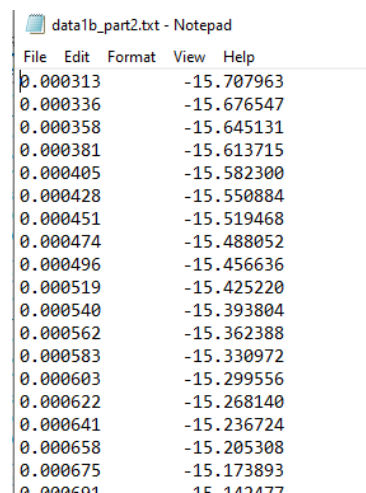
PH20018 Programming skills  
Semester 2, 2021  
Coursework 1  
Candidate 23858



```
data1b_part1.txt - Notepad
File Edit Format View Help
-15.707963, -0.141082, -0.138925, 0.158983
-15.676547, -0.136641, -0.143574, 0.155171
-15.645131, -0.132056, -0.148091, 0.151200
-15.613715, -0.127331, -0.152471, 0.147075
-15.582300, -0.122471, -0.156710, 0.142798
-15.550884, -0.117480, -0.160804, 0.138374
-15.519468, -0.112363, -0.164747, 0.133807
-15.488052, -0.107124, -0.168536, 0.129100
-15.456636, -0.101769, -0.172167, 0.124259
-15.425220, -0.096302, -0.175635, 0.119288
-15.393804, -0.090729, -0.178938, 0.114190
-15.362388, -0.085055, -0.182072, 0.108971
-15.330972, -0.079284, -0.185033, 0.103636
-15.299556, -0.073424, -0.187818, 0.098189
-15.268140, -0.067479, -0.190423, 0.092636
-15.236724, -0.061455, -0.192847, 0.086982
-15.205308, -0.055358, -0.195086, 0.081231
-15.173893, -0.049193, -0.197137, 0.075390
-15.142477, -0.042967, -0.198999, 0.069464
-15.111061, -0.036686, -0.200668, 0.063458
-15.079645, -0.030354, -0.202144, 0.057378
```

Figure 18: Text file obtained from 1b\_section1.c

The following text file created during the construction of this program: “data1b\_part2” contains the data pertaining to  $l(r)$  and it’s corresponding  $x$ , therefore forming only two columns.



```
data1b_part2.txt - Notepad
File Edit Format View Help
0.000313, -15.707963
0.000336, -15.676547
0.000358, -15.645131
0.000381, -15.613715
0.000405, -15.582300
0.000428, -15.550884
0.000451, -15.519468
0.000474, -15.488052
0.000496, -15.456636
0.000519, -15.425220
0.000540, -15.393804
0.000562, -15.362388
0.000583, -15.330972
0.000603, -15.299556
0.000622, -15.268140
0.000641, -15.236724
0.000658, -15.205308
0.000675, -15.173893
0.000694, -15.142477
```

Figure 19: Text file obtained from 1b\_section2.c

When plotted in Jupyter Notebooks using Python’s library `matplotlib`, this data produces the following plot:

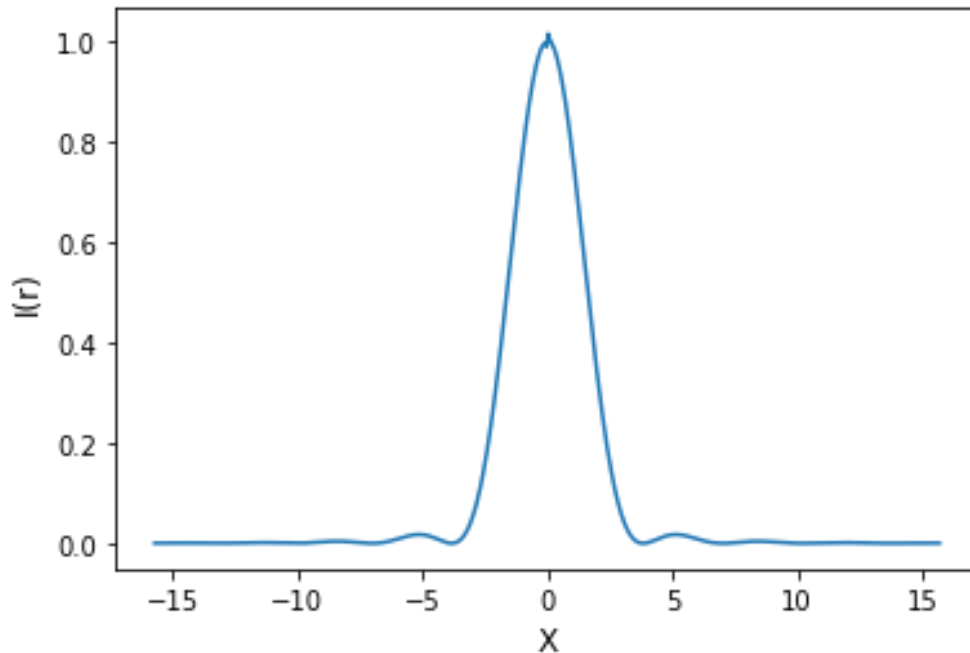


Figure 20: Plot obtained from data1b\_part2.txt

Which remind us of a diffraction pattern with a maxima at  $x=0$ .

Following these results, the data obtained when running the final program (including the  $r$  function) follows the structure see in the previous data file (two columns, one pertaining to the  $I(r)$  values and another pertaining to their corresponding  $r$  values).

data1b\_part2\_version2.txt - Notepad

File	Edit	Format	View	Help
0.000313			-25.000000	
0.000336			-24.949999	
0.000358			-24.899999	
0.000381			-24.849999	
0.000405			-24.800001	
0.000428			-24.750001	
0.000451			-24.700000	
0.000474			-24.650000	
0.000496			-24.600000	
0.000519			-24.550000	
0.000540			-24.500000	
0.000562			-24.450000	
0.000583			-24.400000	
0.000603			-24.350000	
0.000622			-24.300000	
0.000641			-24.249999	
0.000658			-24.199999	
0.000675			-24.150001	
0.000691			-24.100001	
0.000705			-24.050001	
0.000719			-24.000000	
0.000731			-23.950000	
0.000742			-23.900000	
0.000751			-23.850000	
0.000760			-23.800000	
0.000766			-23.750000	
0.000772			-23.700000	

Figure 21: Text file obtained from 1b\_section3.c

When this data is plotted we obtain the following plot:

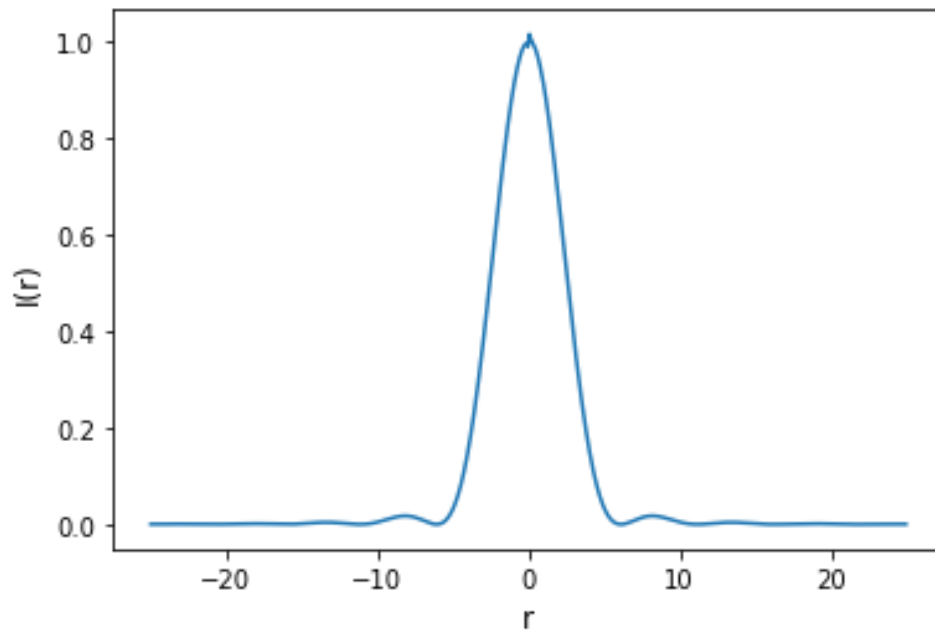


Figure 22: Plot of results obtained for question 1b

This plot is coherent with our previous results.

## **Part 2: Root finding**

### **2.a Limits of numerical precision and possible sources of rounding errors in root finding**

There exist two types of errors than can limit the numerical precision and be possible sources of rounding errors in root finding: round off errors and truncation errors.

Round off errors occur as a result of the representation of numbers using a finite number of digits. These errors can accumulate and cause significant mathematical discrepancies between the expected and obtained results when running the code.

Truncation errors on the other hand arise when exact mathematical formulations are represented by approximations. An example of a truncation error could be ignoring smaller terms in the calculation of an infinite series. Similarly, to round off errors, truncation errors can cause significant errors within our results.

Generically speaking four main types of issues may arise when root finding with C.

Firstly, rounding issues may be directly related to memory usage and thus employed data types within the code. If roots are found beyond  $x=\pm 10^{-67}$ , they will exceed the available computational precision.

Secondly, issues can arise if the programmer chooses to work with floats. The use of these can cause issues when comparing for equality with other floats as they cannot precisely store certain values such as 0.5 in binary.

Thirdly, complex roots due to their nature may cause issues regarding numerical precision or rounding errors.

Fourthly, functions might change sign without there being a root in the location of the change, or even behave contrarily and not change sign when a root is present. Similarly, to the previous point due to the nature of this challenge, it might arise issues regarding numerical precision or rounding errors.

Depending on the method used to find roots, we might also see other types of errors. I will shortly introduce the Bisection method and the Newton-Raphson Method; and elaborate on issues that arise when using these methods.

The Bisection Method is according to (Mathcs.emory.edu, n.d.) “a successive approximation method that narrows down an interval that contains a root of the function  $f(x)$ ”.

The use of this method can cause errors if the root lies in the bounds of our approximation. As well as if we don't rerun the code after finding a root, as we might miss others. For obvious reasons we design the code to ignore the previously found root. Similarly, if the used approximation “stepping” is too big we can miss double roots.



The Newton-Raphson method is according to (Brilliant.org, n.d.) “a way to quickly find a good approximation for the root of a real-valued function  $f(x) = 0$  or  $f(x) = 0$ . It uses the idea that a continuous and differentiable function can be approximated by a straight line tangent to it”.

The use of this method can cause issues when the local gradient is 0 or when the form of the function  $f(x)$  causes  $(x, x+\alpha)$  -with  $\alpha$  being the small interval used- cycle in a loop without converging to the required loop. The following image, taken from the (PH20018), illustrates this event.

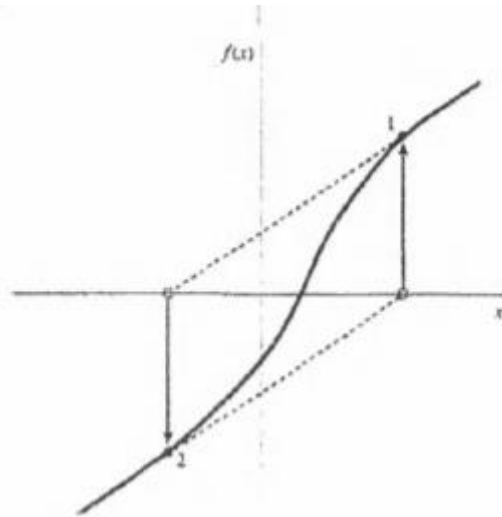


Figure 23: Image taken from the notes

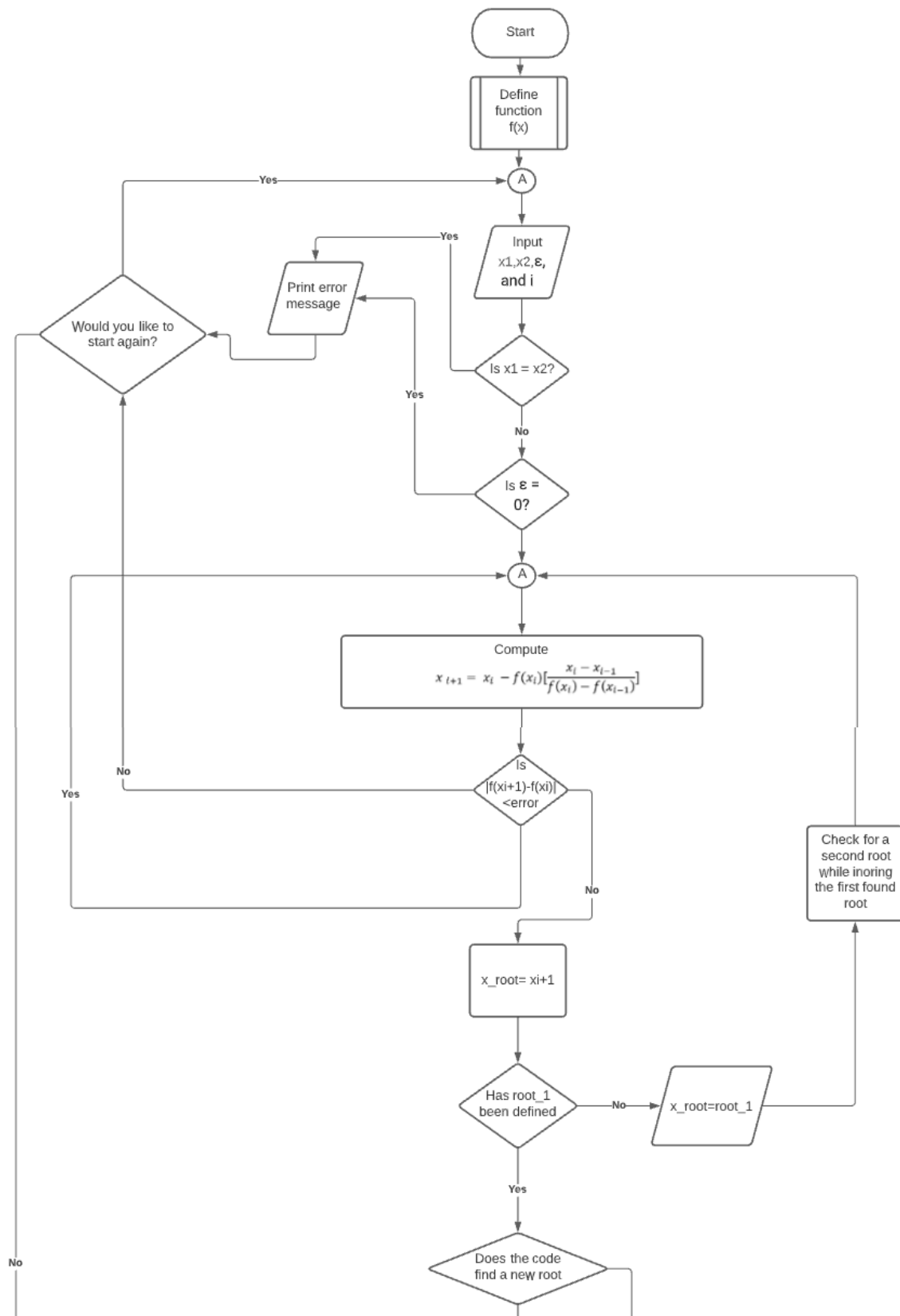
## **2.b Flowchart of a program to find the real roots of the equation**

The following flowchart describes a program that will find the real roots to an inputted equation  $f(x)$ , and in which you can specify the required precision of the result.

It considers the issues you discussed in part 2.a, as it minimizes the issues caused by either discussed method by employing a combination of both, the Secant method.

According to it's (Wikipedia, n.d.) page, the “secant method is a root-finding algorithm that uses a succession of roots of secant lines to better approximate a root of a function  $f$ ”. The method is defined by the following recurrence relation (“equation that recursively defines a sequence or multidimensional array of values” - (Wikipedia, n.d.)) :

$$x_{i+1} = x_i - f(x_i) \left[ \frac{x_i - x_{i-1}}{f(x_i) - f(x_{i-1})} \right]$$



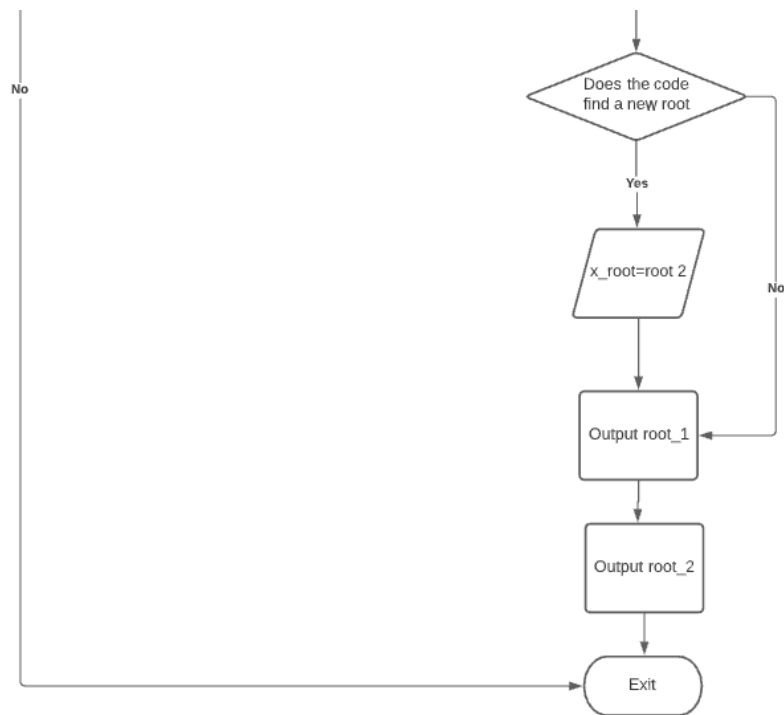


Figure 24: Designed Flowchart

## 2.c Root finding program

By taking into consideration the items discussed in the two previous subsections, a C program that calculates the roots of the function  $f(x) = \cos(x)^2 + x^3 - x^2 - 0.291064$  with the use of the Secant method, was designed.

In the following subsections I will elaborate on the design steps, results and discussion of the accuracy of such.

### Design

Similarly to the previous programs this one employs the standard library `stdio.h`, the `math.h` library for the use of the `cos` and `pow` functions and the `stdlib.h` library for the use of the function `fabs(x)` (that gives us the absolute value of  $x$ ).

```
1  #include<stdio.h>
2  #include<math.h>
3  //used for absolute value function - fabs
4  #include<stdlib.h>
```

: Lines 1-4 C file *Source\_code\_question2.c*

The first function in this code (written in lines 7-15) defines our function  $f(x) = \cos(x)^2 + x^3 - x^2 - 0.291064$ , for later usage within the main function. Given the design of the C computing language this function employs with radial degrees for it's calculations. It has been designed with double variable types to avoid round off errors.

```
7  //defining function
8  double f(double x)
9  {
10     double result;
11     //calculation is done in radius
12     result = pow(cos(x), 2) + pow(x, 3) - pow(x, 2) - 0.291064;
13     //printf("\t\tf(%lf) = %lf\n", x, result);
14     return result;
15 }
```

Figure 25: Lines 7-15 C file *Source\_code\_question2.c*

A root finding function based on the recurrence relation that defines the secant Method (see section 2.b), follows it (in lines 17-40). This function takes into account the desired accuracy for the calculation, and calculates the root of the function  $f(x)$  based on an inputed range. In addition to the desired accuracy of our results it also takes into consideration a desired number of iterations of the secant loop.

```
17 double root_finding(double (*f)(double x), double x0, double x1, double e, int N)
18 {
19     int i = 0;
20     double x2;
21     if (e == 0.)
22     {
23         printf("\nError cannot have an accuracy equal zero!\n");
24         exit(-1);
25     }
26     if (x1 < x2 || x0==x1)
27     {
28         printf("\nError range!\n");
29         exit(-1);
30     }
31     //secant loop
32     while (N > i++ && fabs(x1 - x0) > fabs(e))
33     {
34         x2 = x1 - ((f(x1) * (x1 - x0)) / (f(x1) - f(x0)));
35         x0 = x1;
36         x1 = x2;
37         //printf("\t%d: %.20lf\n", i, x2);
38     }
39     return x2;
40 }
```

Figure 26: Lines 17-40 C file Source\_code\_question2.c

Finally within the main function, there is a defined accuracy  $10^{-6}$ , this choice was based of the numbers given in the original function and ran the root finding function twice to find both roots of the function.

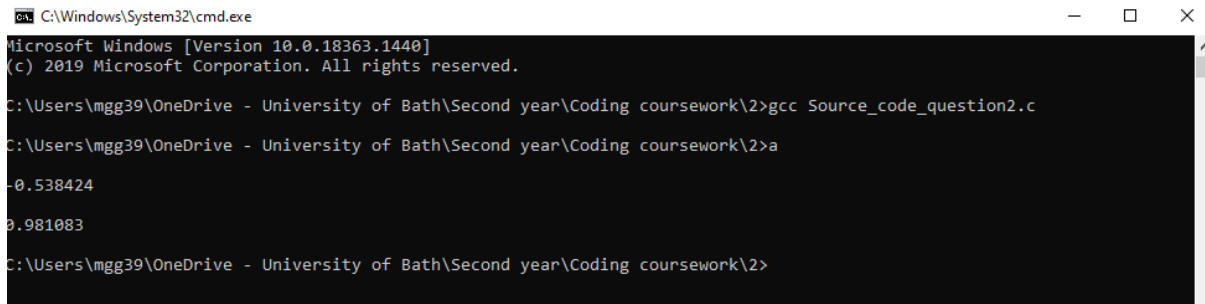
The chosen ranges are based on mathematical approximations of the expected results. The code can run with other input ranges.

```
44 int main()
45 {
46     //let's put a precision of 10^-6
47     double e = 1 / pow(10, 6);
48
49     double root = root_finding(f, -1, 0, e, 10000);
50     printf("\n%lf\n", root);
51     root = root_finding(f, 2, 4, e, 10000);
52     printf("\n%lf\n", root);
53
54     return 1;
55 }
```

Figure 27: Lines 44-55 C file Source\_code\_question2.c

## Results

The following results are obtained when running this code:



```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.18363.1440]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\magg39\OneDrive - University of Bath\Second year\Coding coursework\2>gcc Source_code_question2.c
C:\Users\magg39\OneDrive - University of Bath\Second year\Coding coursework\2>a
-0.538424
0.981083
C:\Users\magg39\OneDrive - University of Bath\Second year\Coding coursework\2>
```

Figure 28: Obtained results in Command line when running file `Source_code_question2.c`

These results seem correct at first glance. In the following subsection we can see an elaboration of this impression.

### **Discussion on the result accuracy**

As discussed previously, this code was designed to output roots with a level of accuracy of  $10^{-6}$ , given that that was the accuracy of the numbers given in our original function. We can see that both results have this level of accuracy.

Nonetheless this is not directly tied to their correctness. Thus why we shall run the function with these proposed roots to verify our results.

1.  $f(-0.538424) = \cos(-0.538424)^2 - 0.538424^3 - 0.538424^2 + 0.538424 = 0.00000323583$
2.  $f(0.981083) = \cos(0.981083)^2 + 0.981083^3 - 0.981083^2 - 0.29106 = 0.00000449564$

As we can see both resulting roots, when calculated within the original function output results nearby 0 with an accuracy of  $10^{-6}$ , which confirms the correctness of our results.

In order to make this code more generalized this program could be designed so the user inputs the desired accuracy and search ranges. See file “Generalized\_version2.c” for a code that does this.

### **Part 3: Multivariable equations**

To answer the posed question, a C program that solves the following set of simultaneous equations:

$$(x + 1)^2 + (y + 1)^2 = 25 \text{ and } xy + y^2 = 5$$

was designed and encoded.

In the following subsections we will elaborate on the mathematical reasoning, design and results of the produced code.

#### **Mathematical reasoning**

Before solving this problem through code, we shall look at the mathematics behind both individual equations and solve them for  $y$  to obtain more palatable versions of these.

The following two subsection will elaborate on this idea.

$$\textbf{Equation 1 : } (x + 1)^2 + (y + 1)^2 = 25$$

This equation corresponds to the equation of a circle with radius 5.

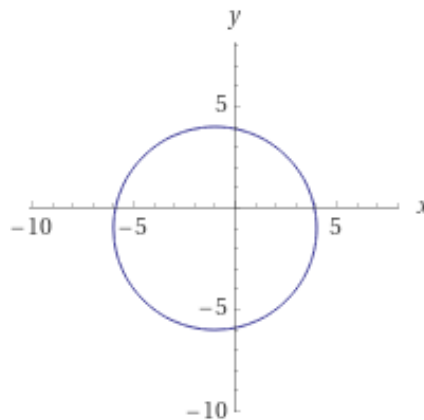


Figure 29: Plot of Equation 1

Image obtained from (1 Alpha, n.d.)

In order to solve for  $y$  the following steps are followed:

1. *Subtract  $(x + 1)^2$  from both sides*  $\rightarrow (y + 1)^2 = 25 - (x + 1)^2$
2. *Take the square root from both sides*  $\rightarrow y = \sqrt{25 - (x + 1)^2} - 1$  or  $y + 1 = -\sqrt{25 - (x + 1)^2}$
3. *Finally, subtract 1 from both sides*  $\rightarrow y = \sqrt{25 - (x + 1)^2} - 1$  or  $y = -\sqrt{25 - (x + 1)^2} - 1$

Thus we obtain the following equivalent equation:

$$y = \sqrt{25 - (x + 1)^2} - 1 \text{ or } y = -\sqrt{25 - (x + 1)^2} - 1$$

**Equation 2:  $xy + y^2 = 5$**

The second equation resembles a curve flipped in the y and x axis when plotted.

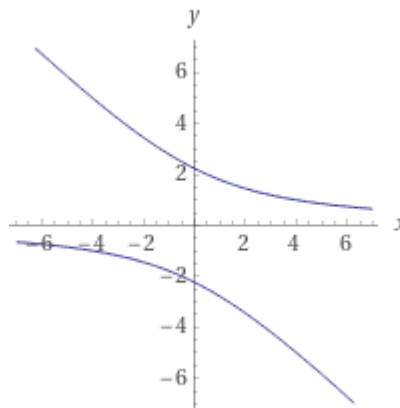


Figure 30: Plot of equation 2

Image obtained from (2 Alpha, n.d.)

In order to solve for y the following steps are followed:

1. Add  $(\frac{x}{2})^2$  to both sides  $\rightarrow y^2 + xy + \frac{x^2}{4} = \frac{x^2}{4} + 5$
2. Writte the left hand side as a square and take the square of both sides  $\rightarrow \sqrt{(y + \frac{x}{4})^2} = \sqrt{\frac{x^2}{4} + 5}$
3. Subtract  $\frac{x}{2}$  from both sides  $\rightarrow y = \sqrt{\frac{x^2}{4} + 5} - \frac{x}{2}$  or  $y = -\sqrt{\frac{x^2}{4} + 5} - \frac{x}{2}$

Thus the following equivalent equation is obtained:

$$y = \sqrt{\frac{x^2}{4} + 5} - \frac{x}{2} \text{ or } y = -\sqrt{\frac{x^2}{4} + 5} - \frac{x}{2}$$

**Mathematics behind the design**

For comprehension purposes from this point on, equation 1 and equation 2 will be reference to the following:

1. Equation 1:  $y = \sqrt{25 - (x + 1)^2} - 1$  or  $y = -\sqrt{25 - (x + 1)^2} - 1$
2. Equation 2:  $y = \sqrt{\frac{x^2}{4} + 5} - \frac{x}{2}$  or  $y = -\sqrt{\frac{x^2}{4} + 5} - \frac{x}{2}$



The mathematics behind this code's design are very simple and thus efficient, "Everything should be made as simple as possible, but no simpler." - Albert Einstein. It consists of a division of both equations into their positive and negative sectors, followed by output of these values into a text file - which permits a human verification of the obtained - and an analysis comparing all the obtained values to find equivalents and thus solutions to the sets of simultaneous equations. To avoid errors regarding this comparison double variable types as well as a high level of accuracy are used.

### Code design

The libraries used for this program are the following: the C standard stdio.h library, the stdlib.h library used for data file transfer, the math.h library for the use of the functions pow and cos, as well as the library complex.h used as the standard C library for complex numbers.

```
1  #include<stdio.h>
2  #include<math.h>
3  #include<stdlib.h>
4  //Standard Library of Complex Numbers
5  #include <complex.h>
```

Figure 31: Lines 1-5 C file Source\_code\_question3.c

The first two functions (lines 10 to 21 and 24 to 30) in the program define the non changing sections between the positive and negative counterparts of both equations (1 and 2).

For example, for equation 1 this would correspond to  $y = \sqrt{25 - (x + 1)^2} - 1$  or  $y = -\sqrt{25 - (x + 1)^2} - 1$

These will later be used to calculate their positive and negative sections.

```
8  //see work for why the use of these versions of the function
9  //defining function 1
10 double f1(double x)
11 {
12     double equation;
13     if (x < 4 && x > -6)
14     {
15         equation = sqrt(-pow(x, 2) - 2 * x + 24);
16     }
17     else {
18         equation = sqrt(pow(x, 2) + 2 * x - 24) * I;
19     }
20     return equation;
21 }
22
23 //defining function 2
24 double f2(double x)
25 {
26     double equation = sqrt(pow(x,2)+20);
27     //double positive = 0.5*(-equation - x);
28     //double negative = 0.5*(equation - x);
29     return equation;
30 }
```

Figure 32: Lines 8-30 C file Source\_code\_question3.c

A comparison function follows these equation. This function is designed to find equivalent y outputs for two compared equations with a inputted accuracy. This accuracy is 10 times inferior to the accuracy of our inputted y values in order to avoid as many repetitions of the same root with slightly different decimal points, as possible. These will still be present I our outputs, but we minimize them in this manner. The double variable types are used in this function for similar purposes and to avoid roundoff errors.

If the function finds such numbers it prints they x and why values into the command line, these would correspond to an answer of our initial problem.

```
--  
32     //function comparing obtained x  
33     double comparison(double x, double function_a, double function_b, double accuracy)  
34     {  
35  
36         double comparison_accuracy_factor = 10 / (accuracy);  
37         if (fabs(function_a - function_b) < comparison_accuracy_factor)  
38         {  
39             printf("%lf, \t %lf \n", x, function_a);  
40         }  
41         return 1;  
42     }  
--
```

Figure 33: Lines 32-42 C file Source\_code\_question3.c

Finally within the main function the desired accuracy our calculations is defined ( $10^6$  was chosen in this case to maintain a consistency with previous questions) of as well as the range in which we will iterate our equations. The ranges present in the code were based on intuition and the given equations. The values of such range are multiplied by the desired accuracy in lines 36-37. This is done given that for loops run with integer values. They will late be re-modified once inside the loop to their corresponding x equivalents:  $4 \rightarrow 4 \times 10^6 \rightarrow 4$ .

Following this a text file named "data3.txt" is created with the intention of inputting our obtained data of the equations in order to verify it's mathematical consistency with our expected results and check for any issues or bugs. The purpose of this file is solely for debugging.

A for loop is then present in lines 48-59. It's purpose is to iterate through the values of x in both equations and compare these with the help of the comparison function previously described to find equivalencies between they're y results.

As explained before for this purpose the equations are divided into their positive and negative sections (lines 52-55).

```
44 int main()
45 {
46     int i;
47     int accuracy = pow(10, 6);
48     int value_end = 4 * accuracy;
49     int value_start = -7 * accuracy;
50
51
52     //file fprintf for data analysis & modelling
53     putchar('\n');
54     FILE* fp = fopen("data_question3.txt", "w");
55     if (!fp)
56     {
57         fprintf(stderr, "Failed to open file");
58         exit(-1);
59     }
60     for (i = value_start; i < value_end; ++i)
61     {
62         double recalc_value = (double)i / (double)accuracy;
63         // x corresponds to recalc value
64         double positive_f2 = 0.5 * (f2(recalc_value) - recalc_value);
65         double negative_f2 = 0.5 * (-f2(recalc_value) - recalc_value);
66         double f1_value_positive = f1(recalc_value) - 1;
67         double f1_value_negative = -f1(recalc_value) - 1;
68
69         //printf("\t %d \t %f \n", i, recalc_value);
70         //printf(" \t %f \n", f1(0.2));
71         fprintf(fp, "%lf, %lf, %lf, %lf, %lf\n", recalc_value, positive_f2, negative_f2, f1_value_positive, f1_value_negative);
72
73         //finding the answers through comparison
74         comparison(recalc_value, f1_value_positive, positive_f2, accuracy);
75         comparison(recalc_value, f1_value_positive, negative_f2, accuracy);
76         comparison(recalc_value, f1_value_negative, positive_f2, accuracy);
77         comparison(recalc_value, f1_value_negative, negative_f2, accuracy);
78     }
79
80
81
82     fclose(fp);
83     return 1;
84 }
```

Figure 34: Lines 44-84 C file Source\_code\_question3.c

## **Results and discussion**

As explained previously this program outputs the data corresponding to the equations in a text file. To ensure the correctness of this data it was briefly analysed in Jupyter notebooks using Python 3 and the matplotlib library. The used code was the following:

```
In [1]: import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

In [4]: data = np.genfromtxt("data3.txt", skip_header = 0, delimiter=",")

x = data[:,0]
positive_f2 = data[:,1]
negative_f2 = data[:,2]
positive_f1 = data[:,3]
negative_f1 = data[:,4]

In [7]: plt.plot(x,positive_f2)
plt.plot(x,negative_f2)
plt.plot(x,positive_f1)
plt.plot(x,negative_f1)

plt.legend(['+F2','-F2','+F1','-F1'])

plt.ylabel('Equations', fontsize=12)
plt.xlabel('x', fontsize=12)
plt.show()
```

Figure 35: Python 3 code used to plot obtained data.txt file

The following plot was obtained:

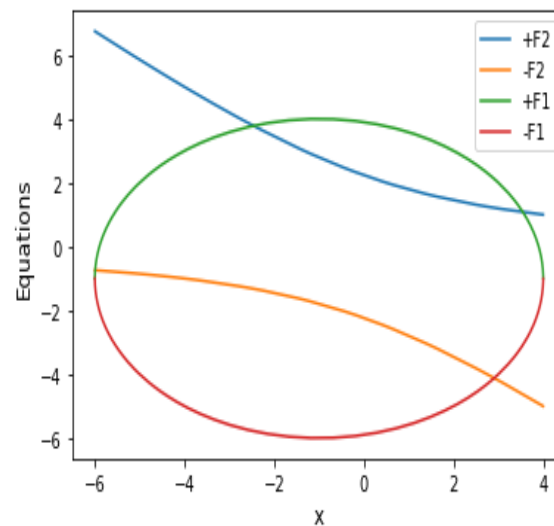


Figure 36: Obtained plot

These results seemed to correspond to the expected ones.

Here is a Plot of the set of equations obtained from (Alpha, n.d.), for comparison:

Plot of solution set:

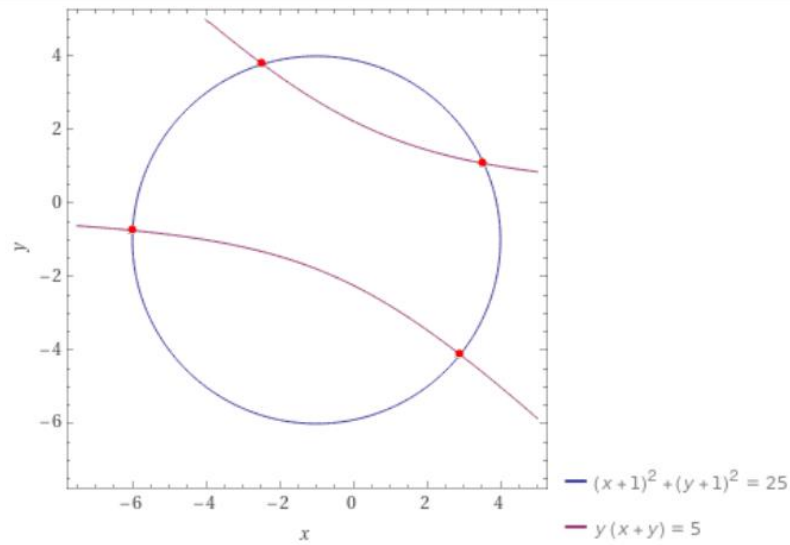


Figure 37: Plot of Equation 1 and 2

The results of the code are printed in the command line. Here is a screenshot of it's outputs:

```

C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.18363.1440]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\magg39\OneDrive - University of Bath\Second year\Coding coursework\3>gcc Source_code_question3.c
C:\Users\magg39\OneDrive - University of Bath\Second year\Coding coursework\3>a

-5.993356,      -0.742326
-5.993355,      -0.742307
-2.460060,      3.782073
-2.460059,      3.782074
-2.460058,      3.782074
-2.460057,      3.782074
-2.460056,      3.782074
-2.460055,      3.782075
-2.460054,      3.782075
-2.460053,      3.782075
-2.460052,      3.782076
-2.460051,      3.782076
-2.460050,      3.782076
-2.460049,      3.782077
-2.460048,      3.782077
-2.460047,      3.782077
-2.460046,      3.782078
-2.460045,      3.782078
-2.460044,      3.782078
-2.460043,      3.782078
-2.460042,      3.782079
2.906845,      -4.120346
2.906846,      -4.120345
2.906847,      -4.120344
2.906848,      -4.120343
2.906849,      -4.120341
2.906850,      -4.120340
2.906851,      -4.120339
2.906852,      -4.120338
2.906853,      -4.120336
2.906854,      -4.120335
3.546553,      1.080590
3.546554,      1.080588
3.546555,      1.080586
3.546556,      1.080584
3.546557,      1.080582
3.546558,      1.080579
3.546559,      1.080577
3.546560,      1.080575
3.546561,      1.080573
3.546562,      1.080571

```

Figure 38: Obtained results on command line when running file Source\_code\_question3.c

As we can see, many of these results are almost identical yet appear multiple times due to the accuracy chosen in their calculations.

We can nonetheless distinguish four results with an accuracy of  $10^{-2}$ . These are:

1.  $x=-5.99 \rightarrow y=-0.74$
2.  $x=-2.46 \rightarrow y=3.78$
3.  $x=2.90 \rightarrow y=-4.12$
4.  $x=-3.54 \rightarrow y=1.08$

These results correspond to the intersection points observable in the plot obtained with the text file data and thus we can assume they are correct. Their input into the original equations confirms this.

In order to make this code more generalized this program could be designed for the user to input their desired range and accuracy, and the creation of the text file section could be eliminated, for a faster running code. See file "Generalized\_version3.c" for a code that does this – unfortunately I was not able to upload this document thus I have copied the involved program, you can see it in pages 31-32. Sorry for any inconvenience.

### **Bibliography**

- 1 Alpha, W. (n.d.). Retrieved from <https://www.wolframalpha.com/input/?i=%28x%2B1%29%5E2+%2B+%28y%2B1%29%5E2+%3D25>
- 2 Alpha, W. (n.d.). Retrieved from [https://www.wolframalpha.com/input/?i=x\\*y%2B%5E2%3D5](https://www.wolframalpha.com/input/?i=x*y%2B%5E2%3D5)
- Alpha, W. (n.d.). Retrieved from <https://www.wolframalpha.com/input/?i=%E3%80%96%28x%2B1%29%E3%80%97%5E2%2B%E3%80%96%28y%2B1%29%E3%80%97%5E2%3D25+and+xy%2B+y%5E2%3D5>
- Brilliant.org. (n.d.). Retrieved from [https://brilliant.org/wiki/newton-raphson-method/#:~:text=The%20Newton%2DRaphson%20method%20\(also,straight%20line%20tangent%20to%20it.](https://brilliant.org/wiki/newton-raphson-method/#:~:text=The%20Newton%2DRaphson%20method%20(also,straight%20line%20tangent%20to%20it.)
- Mathcs.emory.edu. (n.d.). Retrieved from <http://www.mathcs.emory.edu/~cheung/Courses/170/Syllabus/07/bisection.html>
- Wikipedia. (n.d.). Retrieved from [https://en.wikipedia.org/wiki/Secant\\_method](https://en.wikipedia.org/wiki/Secant_method)
- WolframMathWorld. (n.d.). *Math World Wolfram*. Retrieved 03 18, 2021, from <https://mathworld.wolfram.com/BesselFunctionoftheFirstKind.html>

```
//Code for the file "Generalized_version3.c"

#include<stdio.h>
#include<math.h>
#include<stdlib.h>
//Standard Library of Complex Numbers
#include <complex.h>

//see work for why the use of these versions of the function
//defining function 1
double f1(double x)
{
    double equation;
    if (x < 4 && x > -6)
    {
        equation = sqrt(-pow(x, 2) - 2 * x + 24);
    }
    else {
        equation = sqrt(pow(x, 2) + 2 * x - 24) * I;
    }
    return equation;
}

//defining function 2
double f2(double x)
{
    double equation = sqrt(pow(x,2)+20);
    //double positive = 0.5*(-equation - x);
    //double negative = 0.5*(equation - x);
    return equation;
}

//function comparing obtained x
double comparison(double x, double function_a, double function_b, double accuracy)
{
    double comparison_accuracy_factor = 10 / (accuracy);
    if (fabs(function_a - function_b) < comparison_accuracy_factor)
    {
        printf("%lf, \t %lf \n", x, function_a);
    }
    return 1;
}

int main()
{
    int i;
    int accuracy;
    int x_min;
    int x_max;
    printf("Desired accuracy: ");
    scanf("%d", &accuracy);
    printf("Max x: ");
    scanf("%d", &x_max);
    printf("Min x: ");
    scanf("%d", &x_min);
    int value_start = x_min * accuracy;
    int value_end = x_max * accuracy;
```

```
//file fprintf for data analysis & modelling
if (value_end < value_start)
{
    printf("Error range");
}
else
{
    for (i = value_start; i < value_end; ++i)
    {
        double recalc_value = (double)i / accuracy;
        // x corresponds to recalc value
        double positive_f2 = 0.5 * (f2(recalc_value) - recalc_value);
        double negative_f2 = 0.5 * (-f2(recalc_value) - recalc_value);
        double f1_value_positive = f1(recalc_value) - 1;
        double f1_value_negative = -f1(recalc_value) - 1;

        //finding the answers through comparison
        comparison(recalc_value, f1_value_positive, positive_f2, accuracy);
        comparison(recalc_value, f1_value_positive, negative_f2, accuracy);
        comparison(recalc_value, f1_value_negative, positive_f2, accuracy);
        comparison(recalc_value, f1_value_negative, negative_f2, accuracy);

    }

    return 1;
}
```