**PHY1003 – Assignment 1 – Optimization and Root Finding**

Abstract

A function *f(x) = sin(xa – x1/a + ax)* was investigated by using a program to calculate and plot the graph of the function. A second program was written to find the roots of this function. A third program was written to find the turning points of this graph.

Part 1 – Plotting f(x):

A program was written to calculate f(x) for a range of x values above zero (values below zero do not work as a square root is required, and we require real solutions).

The program uses a *for* loop to calculate f(x) for a range of values and writes the values into an excel document, allowing a graph to be created for the function. A small increment was added to x each time so that a smooth and filled out curve could be plotted.

Graph 1: Graph of f(x) for 0 < x ≤ 6

Part 2 – Root Finding:

A program was written to find the root (where f(x)=0) of the function graphed in Graph 1. This utilised the Newton-Raphson method to calculate the next best guess.

All the results are presented in the web-form.

Part 3 – Maximum Finding:

A series of programs were written to find values of varying accuracy for the first maximum of f(x).

The first program calculated f(x) values for incrementing x values, stopping once it reached f(x)=0. This incorporated a do…while loop to make sure that the code would stop when f(x)>0. This produced a result of 0.72.

The second program utilised the Newton-Raphson method in a similar sense to part 2. For part 3 the program aimed to find the point where the graph was level ie. f’(x)=0. This method produced a result of 0.716822 which is much more precise.

The third program is the Bisection method. Although it is more complex a program and has to do more to find a similar answer it is beneficial that f’(x) and f’’(x) do not need to be calculated as the input for these calculations was an arduous task which resulted in a long sequence of commands which resulted in many mistakes. The result given by this method is 0.717285. Note that the same tolerance was used for this program but a different answer was given.

Overall the first method was the least effective. It had a very large granularity which was limited by the incremental value chosen of 0.01. This could be improved by making the incremental step value smaller. However, this would mean that the program has to do many calculations, which will decrease performance and take much longer. The second method (Newton-Raphson) may be the most effective. It achieves a high degree of accuracy with relatively few calculations as the equation quickly zeroes in on a value. The third program is comparable to the first, it gave a number to a similar degree of accuracy but more calculations were required, meaning that it has a slightly worse performance and more rounding errors can occur, this is a possible reason to why the values for these two methods differ.

Web-form

Assignment 1

\* Required

**Please enter your anonymous code (with no spaces)** \*

83694040

**Part 1**

**f(x) when x=0.0**

0

**f(x) when x=0.01**

0.0638763

**f(x) when x=0.02**

0.0998638

**f(x) when x=0.03**

0.129626

**f(x) when x=0.04**

0.15582

**f(x) when x=0.05**

0.17954

**Part 2**

**'Table of results - you must start with x0=2 - if you got an error then LEAVE THE ANSWER BLANK FOR ANY NUMBER YOU DID NOT FIND**

**f(x0) (x0=2)**

-0.302166

**f'(x0) (x0=2)**

-5.25219

**x1**

1.94247

**f(x1)**

-0.225823

**f'(x1)**

-5.13798

**x2**

1.89852

**f(x2)**

-0.168445

**f'(x2)**

-5.02413

**x3**

1.86499

**f(x3)**

-0.125419

**f'(x3)**

-4.92412

**Comment that best describes the result of your Newton-Raphson search starting with x0=2**

First root above zero found

**first root above zero - value of x**

1.76332

**second root above zero - value of x**

3.44118

**Part 3**

**First maximum; value of x**

0.716822

**First maximum; value of y**

0.6238

Annotated Code:

**Part 1 – Creating the Graph:**

//including libraries which contain different c operations

**#include** <stdio.h>

**#include** <stdlib.h>

**#include** <math.h>

//main loop, this is the set of instructions the program will process

**int** **main**(**void**) {

//these are added to correct a bug in eclipse

**setvbuf**(stdout,NULL,\_IONBF,0);

**setvbuf**(stderr,NULL,\_IONBF,0);

//Assigning a file pointer variable the name file\_pointer

FILE \* file\_pointer;

//Having the program open the file and calling it "A1 Data" and indicating it is to be written in.

file\_pointer = **fopen**("A1 Data.xls", "w");

//Defining variables, in this case all doubles (like decimal values)

**double** a=0.4+(5427.0/25000.0), fx, x;

/\* Using a for loop to cycle over values of x beginning at 0 and ending just before 6.

\* The increment being increased by is 0.01 in this case. This was selected to have good graph line definition. \*/

**for**(x=0; x<6; x=x+0.01)

{

//The function

fx=sin(pow(x,a)-pow(x,(1/a))+a\*x);

//'Printing' the values for x and fx into my A1 data file.

**fprintf**(file\_pointer, "%g %g\n", x, fx);

//Also printing to screen so you dont have to go into file to see results

**printf**("%g %g\n", x, fx);

}

//we do not need a value to be returned here so exit\_success is used, closing the loop.

**return** EXIT\_SUCCESS;

}

**Part 2 – Finding the Roots:**

//including libraries for certain c operations

**#include** <stdio.h>

**#include** <stdlib.h>

**#include** <math.h>

//main loop

**int** **main**(**void**) {

//to correct a bug in eclipse

**setvbuf**(stdout,NULL,\_IONBF,0);

**setvbuf**(stderr,NULL,\_IONBF,0);

//setting up file pointer and opening a file called "A1 Data pt2" to be written to

FILE \* file\_pointer;

file\_pointer = **fopen**("A1 Data pt2.xls", "w");

//declaring variables, in this case all doubles

**double** a=0.4+(5427.0/25000.0), ffx, fx, x, tolerance=1e-5, gx;

//having the user input a starting value for x

**printf**("Please enter start value of x: ");

**scanf**("%lf", &x);

/\*using a do...while loop meaning that the commands will execute once before

\* the while statement kicks in. Calculations are done each time as the values

\* will be continuously changing. Each time the values are printed to file and

\* shown on screen. \*/

**do**

{

gx=a\*(pow(x,(a-1)))-(1/a)\*(pow(x,(1/a))+a);

fx=sin(pow(x,a)-pow(x,(1/a))+a\*x);

ffx=gx\*cos(pow(x,a)-pow(x,(1/a))+a\*x);

**fprintf**(file\_pointer, "%g %g %g\n", x, fx, ffx);

**printf**("%g %g %g\n", x, fx, ffx);

/\*This is the newton-raphson method equation which will produce our next

\* "guess" by calculating the ratio of the y value to it's derivative

\* and subtracting this from the current x value, creating a new x value

\*/

x=x-(fx/ffx);

//the loop terminates when the absolute value of fx is less than my tolerance

}**while**(fabs(fx)>tolerance);

//program terminates

**return** EXIT\_SUCCESS;

}

**Part 3A – Finding the First Maximum (Data-Search method):**

//libraries

**#include** <stdio.h>

**#include** <stdlib.h>

**#include** <math.h>

//main loop

**int** **main**(**void**) {

//bug correction in eclipse

**setvbuf**(stdout,NULL,\_IONBF,0);

**setvbuf**(stderr,NULL,\_IONBF,0);

//assigning variables

**double** a=0.4+(5427.0/25000.0), gx, ffx, x=0.01, fxmax=0;

/\* Again using a do...while loop. The choice to use this loop came

\* from the fact that you'd like the calculations to happen once

\* no matter what the tolerance is. If your first guess is exactly

\* right (past tolerance or out of scope of 'while' statement you

\* still want the operation to run through once to give an answer. \*/

**do**

{

gx=a\*pow(x,a-1)-(1/a)\*pow(x,(1/a)-1)+a;

ffx=gx\*cos(pow(x,a)-pow(x,1/a)+a\*x);

**printf**("%g %g %g\n", x, ffx, fxmax);

x+=0.01;

/\* Simply saying calculate x and fx, increasing x by 0.01 each time

\* and stopping when ffx (derivative) is greater than 0, as this means

\* the highest value will have been found \*/

}**while**(ffx>fxmax);

//terminate program

**return** EXIT\_SUCCESS;

}

**Part 3B – Finding the First Maximum (Newton-Raphson Method):**

//libraries

**#include** <stdio.h>

**#include** <stdlib.h>

**#include** <math.h>

//main loop

**int** **main**(**void**) {

//eclipse bug

**setvbuf**(stdout,NULL,\_IONBF,0);

**setvbuf**(stderr,NULL,\_IONBF,0);

//assigning variables. Here I separated "fixed" values like a and

//my tolerance

**double** gx, ffx, fffx, x, fx;

**double** a=0.4+(5427.0/25000.0), tolerance=1e-10;

//user input value for x

**printf**("Please enter start value of x: ");

//Using & to write this input to a variable

**scanf**("%lf", &x);

/\* do...while used for same reason as previous part

\* This is a very very similar program to pt2, except

\* the newton-raphson eqn is different \*/

**do**

{

fx=sin(pow(x,a)-pow(x,(1/a))+a\*x);

gx=a\*pow(x,a-1)-(1/a)\*pow(x,(1/a)-1)+a;

ffx=gx\*cos(pow(x,a)-pow(x,1/a)+a\*x);

//this had to be calculated and input correctly

fffx=a\*(a-1)\*pow(x,a-2)+(1/a)\*(1/a-1)\*pow(x,1/a-1)\*-sin(pow(x,a)-pow(x,1/a)+a\*x);

**printf**("%g, %g, %g, %g\n", x, fx, ffx, fffx);

/\* This eqn is the main difference between this

\* and the last part. The ratio has changed to

\* the derivative over the second derivative \*/

x=x-(ffx/fffx);

}**while**(fabs(ffx)>tolerance);

//terminate program

**return** EXIT\_SUCCESS;

}

**Part 3C – Finding the First Maximum (Bisection Search Method):**

//libraries

**#include** <stdio.h>

**#include** <stdlib.h>

**#include** <math.h>

//prototyping a new function

**double** **findfx**(**double**);

//main function

**int** **main**(**void**) {

//eclipse bug

**setvbuf**(stdout,NULL,\_IONBF,0);

**setvbuf**(stderr,NULL,\_IONBF,0);

/\* Declaring variables, divided into two lines

\* and in order of how they appear on the curve,

\* Easier to keep track of for my calculations. \*/

**double** x0, x3, x1, x4, x2;

**double** fx0, fx3, fx1, fx4, fx2;

//initial guesses for x0 and x2

**printf**("Please insert a value for x0: \n");

**scanf**("%lf", &x0);

**printf**("Please insert a value for x2: \n");

**scanf**("%lf", &x2);

//do...while for same reasons

**do**{

/\* Needed to have these equations within the do..while

\* as they have to be recalculated every time \*/

x1=(x0+x2)/2;

x3=(x1+x0)/2;

x4=(x1+x2)/2;

/\* Function called upon to define the fx values for

\* each x variable. The function allowed me to not have to

\* repeat the calculation for each one, saved on a bit of time

\* and easier to keep track of. \*/

fx3 = findfx(x3);

fx1 = findfx(x1);

fx4 = findfx(x4);

/\* each time the do..while cycle happens a series of checks need

\* to be done to see what action should take place next. Depending

\* on which variable is now highest a different series of reassignments

\* should take place. As there were multiple options an else if

\* statement made the most sense. \*/

**if**(fx3>fx1){

/\* The order of these reassignments had to occur in a way that

\* no variable was redefined as to "overwrite" it's value before

\* THAT value could be reassigned. \*/

x1=x3;

x2=x1;

}**else** **if**(fx4>fx1){

x1=x4;

x0=x1;

}**else**{

x0=x3;

x4=x2;

}

/\* Here the tolerance is being tested against the values on the outer

\* limits of the curve. As we could keep getting smaller and smaller

\* a stop is needed to tell the program that we have a high enough

\* degree of accuracy to stop the calculations \*/

}**while**(x2-x0>1e-10);

//printing the final values for x1 (middle and highest value) and fx1

**printf**("%g, %g\n", x1, fx1);

//terminate the main loop

**return** EXIT\_SUCCESS;

}

/\* This function simply takes an input of a double, it then calculates

\* the fx value corresponding to the x value and returns the fx value as

\* a double. \*/

**double** **findfx**(**double** x){

**double** a=0.4+(5427.0/25000.0);

**double** fx=sin(pow(x,a)-pow(x,(1/a))+a\*x);

**return** fx;

}