Homework 5 Numerical Methods

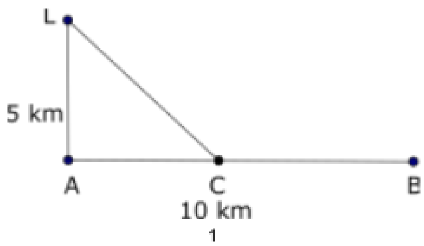
Carlos Giovanny Encinia González

September 1 2021

Exercise 1

A lighthouse L is located on a small island 5 km north of a point A on a straight east-west shoreline. A cable is to be laid from L to point B on the shoreline 10 km east of A. The cable will be laid through the water in a straight line from L to a point C on the shoreline between A and B, and from there to B along the shoreline. (see Figure below). The part of the cable lying in the water costs $\$5.000/km$, and the part along the shoreline costs $\$3,000/km$. Where should C be chosen to minimize the total cost of the cable?

Figure 1: Problem 1



Answer

First we will find the ecuation of sociated to the problem.

We have to calculate de expresion for the distance $\overline{LC}$ and the distance $\overline{CB}$, we know that the distance $\overline{LC}$ for each km wegoing to waste \$5000 and for $\overline{CB}$ we going to waste 3000 for esach km, so we can write the next equation.

First we will let the cordinates of the four points considering the cordinate axis in the point A.

$$A : (0, 0)$$
$$L : (0, 5)$$
$$C : (x, 0)$$
$$B : (10, 0)$$

Fot the distances we have that.

$$\overline{LC} = \sqrt{x^2 + 25}$$
$$\overline{CB} = 10 - x$$

For each distance we have to multiply by its respective cost, and then we can obtain the function of cost:

$$5000\sqrt{x^2 + 25} + 3000(x - 10)$$

for facility the computacional operations we will express the cost in unities that simbolizes thousands os dollars, so.

$$T(x) = 5\sqrt{x^2 + 25} + 3(x - 10) \quad (1)$$

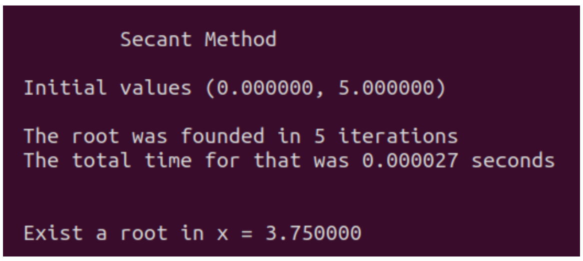
Now we will use the algoritm of Secant so we can to calculate the first derivative of T(x) and we have:

$$T'(x) = \frac{5x}{\sqrt{x^2 + 25}} - 3 \quad (2)$$

And the intervale can be $[0, 10]$ because is the interval where the point C can be.

In the next picture we show the result by the Secant Method.

Figure 2: Result



So round the value obtained in the algorithm we have tat the solution is.

$$C = 3.75$$

Exercise 2

Implement the following algorithms: Bisection, Newton, and Secant meth ods for optimization in 1D.

Answer

The algorithms use three criteria for to stop. The complete program request to the user select one of the tree equations to solve some of the problems, and then ask to the user about the interval of the critical point. Is important for the three methods have the derivative and double derivative(Newton method) for solve the optimization problem.

We show the complete program in the next code.

Code 1: main.c

```c
//Giovanny Encinia
//26-06-2021
#include <stdio.h>
#include "function.h"
//constants
#define ZERO 0
#define ONE 1
#define TWO 2
#define EPSILON 1e-12
#define LIMIT 100000
#define test(x) (x+1)

int main(void)
{
    char option, next, method_sel;
    float a, b, x_r;
    _Bool true = ONE;
    float (*f)(float); // fucntion
    float (*f_p)(float);// derivate function

    printf("\t\tWhat function I will optimize?\n");
    printf("\t\t1.- 5sqrt(x^2 + 25) + 3(10-x)\n");
    printf("\t\t2.- -sin(x) + x^2 + 1\n");
    printf("\t\t3.- sin(x) - x^2 + 1\n");
    scanf(" %c", &option);

    switch(option)//select a function
    {
    case '1':
        f = &F_x;// exp(x) + 2^(-x)+2*cos(x)-6
        f_p = &F_p_x;// exp(x) + 2^(-x)*log(2)-2*sin(x)
        break;
    case '2':
        f = &G_x;// ln(x - 1) + cos(x-1)
        f_p = &G_p_x;// 1/(x - 1) - sin(x-1)
        break;
    case '3':
        f = &H_x;// 230*x^4 + 18*x^3+9*x^2-221*x-9
        f_p = &H_p_x; // 920*x^3+54*x^2+18*x-221
        break;
    default:
        printf("        Select a valid option\n");
        return ZERO;
    }

    while(true)//can select other method for compute the root
    {
        printf("        Select the method\n");
```

```
        printf("\t\t1. Bisection\n\t\t2. Newton\n\t\t3.Secant\n");
        scanf(" %c", &method_sel);




        switch(method_sel)
        {
        case '1'://Bisection
            imprime_resultado("Bisection Method", f, f_p, option, method_sel);
               break;
        case '2'://Newton Raphson
            imprime_resultado("Newton Raphson Method", f, f_p, option, method_sel);
               break;
        case '3'://Secant
            imprime_resultado("Secant Method", f, f_p, option, method_sel);
               break;
        default:
            printf("Select a correct option\n");
            return ZERO;

        }

        printf("\n");
        printf("Do you want to find other root?(1:yes, 2:other)\n");
        scanf(" %c", &next);
        printf("\n\n");

        if(next!='1')
        {
            break;
        }

    }

    return ZERO;
}
```

Code 2: function.h

```
#ifndef FUNCTION
#define FUNCTION
float bisection(float, float, float (*function)(float));
float secant(float, float, float (*function)(float));
float newton(float, float (*function)(float), float (*f_p)(float));
float F_x(float);
float F_p_x(float);
float G_x(float);
float G_p_x(float);
float H_x(float);
```

```
float H_p_x(float);
void print_iter(int, float);
void imprime_resultado(char *name_m,\
                       float (*f)(float), float (*f_p)(float), \
                       char option, char method_sel);
#endif // FUNCTION
```

Code 3: functions.c

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <ctype.h>
#include <time.h>
//constants
#define ZERO 0
#define ONE 1
#define TWO 2
#define EPSILON 1e-12
#define LIMIT 10000000
//Macros
#define MAX(a, b) ((a)<(b)?(b):(a))
#define CRITERIA(a, b) (fabs((b) - (a)) / MAX(1, (b)))

//Declare functions(), to solve the homework
float F_x(float x)//es la derivada
{
    return (5 * x /sqrt(pow(x, 2) + 25) - 3);
}

float F_p_x(float x)//segunda derivada
{
    return (125/pow(x*x + 25, 1.5));
}

float G_x(float x)//derivada
{
    return (-cos(x) + 2 * x);
}

float G_p_x(float x)//segunda derivada
{
    return (sin(x) + 2);
}

float H_x(float x)//derivada
{
    return (cos(x) - 2 * x);
}
```

```c
float H_p_x(float x)//segunda derivada
{
    return (-sin(x) - 2);
}


void print_iter(int i, float time)
{
    /*Print the execution time of the method and the number of iterations*/

    if(i < LIMIT)
    {
        printf("\t\tThe root was founded in %d iterations\n", i);
        printf("\t\tThe total time for that was %lf seconds\n\n", time);
    }
    else
    {
        printf("Exact root do not founded u.u\n");
        printf("\t\t %d iterations\n", i);
        printf("\t\tThe total time for that was %lf seconds\n\n", time);
    }

}

float newton(float x_0, float (*function)(float), float (*f_p)(float))
{
    float x = x_0, x_before = x + ONE;
    int i = ZERO;
    double elapsed;
    clock_t end, start = clock();

    //three criteria, error relative,  function value, and #iterations
    while(CRITERIA(x_before, x_0) > EPSILON
           && fabs(function(x_0))>EPSILON
          && i < LIMIT)
    {
        if(f_p(x) == ZERO)
            break;

        x_before = x_0;
        x_0  -= function(x_0)/f_p   (x_0); //definition Newton's method
        i++;
    }

    end = clock();
    elapsed = (double)(end - start)/CLOCKS_PER_SEC;
    print_iter(i, elapsed);

    return x_0;
}
```

```c
float secant(float a, float b, float (*function)(float))
{
    float x_before, f_a, f_b;
    int i = 0;
    double elapsed;
    clock_t end, start = clock();

    while(CRITERIA(b, a)>EPSILON
          && fabs(function(b))>EPSILON
          && i < LIMIT)
    {
        f_a = function(a);
        f_b = function(b);
        x_before = b; //short the interval
        b  = (a*f_b - b*f_a)/(f_b - f_a);// secant method definition
        a = x_before;// new a
        i++;
    }

    end = clock();
    elapsed = (double)(end - start)/CLOCKS_PER_SEC;
    print_iter(i, elapsed);

    return b;
}

float bisection(float a, float b, float (*function)(float))
{
    float x_1;
    int i = ZERO;
    double elapsed;
    clock_t end, start = clock();

    x_1 = a + (b-a) / TWO; //intermediate point

    while(CRITERIA(a, b) > EPSILON && i < LIMIT)
    {


        if( function(a)*function(x_1) < ZERO)//Bolzano theorem
        {
            b = x_1;
        }
        else
        {
            a = x_1;
        }

        x_1 = a + (b - a) /TWO;
```

```
        i++;
    }

    end = clock();
    elapsed = (double)(end - start)/CLOCKS_PER_SEC;
    print_iter(i, elapsed);

    return x_1;
}

void imprime_resultado\
(char *name_m,float (*f)(float), \
 float (*f_p)(float), char option, char method_sel)
{
    float a, b, x_r;

    if(method_sel == '2')
    {
        printf("        Give me the point x_0\n");
            scanf(" %f", &a);
        printf("\t\t\t%s\n\n", name_m);
        printf("\t\tInitial value %f\n\n", a);
    }
    else
    {
        printf("        Give me the point a\n");
            scanf(" %f", &a);
            printf("        Give me the point b\n");
            scanf(" %f", &b);
            printf("\n\n");
            printf("\t\t\t%s\n\n", name_m);
            printf("\t\tInitial values (%f, %f)\n\n", a, b);
    }

            switch(method_sel)
            {
            case '1'://Bisection
                x_r = bisection(a, b, f);
                break;
            case '2'://Newton Raphson
                x_r = newton(a, f, f_p);
                break;
            case '3'://Secant
                x_r = secant(a,b, f);
                break;
            default:
                printf("Select a correct option\n");
            }

```

```
            printf("\t\tExist a root in x = %.12f\n", \
              x_r);
            if(option == '2')
                printf("\t\tF(x) = %.12f\n", -sin(x_r) + pow(x_r, 2) + 1);
            if(option == '3')
                printf("\t\tF(x) = %.12f\n", sin(x_r) - pow(x_r, 2) + 1);
            printf("\t\tF''(x) = %.12f\n", f_p(x_r));
}
```

Exercise 3

Find the minimum value and minimum point of the following function

$$f(x) = -sin(x) + x^2 + 1$$

on the interval $[-1, 1]$ using the previous implemented algorithms.

• Compare the results in terms of number of iterations.

• Compare and comment the results obtained for each algorithm

if

$$f(x) = sin(x) - x^2 + 1$$

on the interval $[-1, 1]$

Note:

• In Bisection method, use $[a, b] = [-1, 1]$

• In Newton method, use $x_0 = 0$.

• In Secant method, use $x_0 = -1.0$ and $x_1 = 1.0$

• For tolerances or uncertainty-interval use: $1e-12$

0.4501833915710

Answer

Primero tenemos que calcular la primera y segunda derivada para cada una de las funciones.

Sea

$$f(x) = -sin(x) + x^2 + 1 \tag{3}$$

y

$$g(x) = sin(x) - x^2 + 1 \tag{4}$$

$$f'(x) = -cos(x) + 2x$$

$$f''(x) = sin(x) + 2$$

$$g'(x) = cos(x) - 2x$$

$$g''(x) = -sin(x) - 2$$

Now we will show the reults for the equation (1)

Figure 3: Bisection

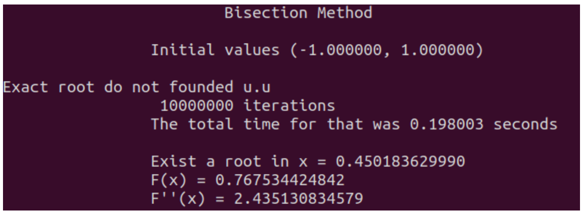


Figure 4: Newton Raphson

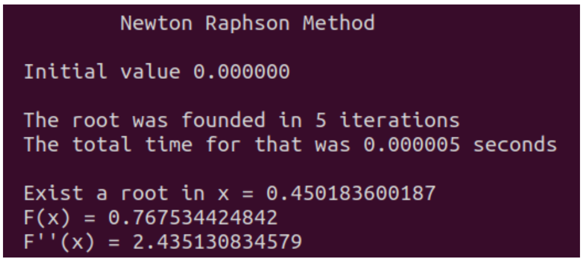
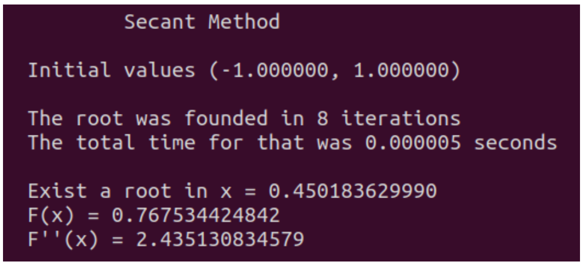


Figure 5: Secant



Ahora resumiremos los datos obtenidos en una tabla.

Method	Iterations	Time	x	F(x)	F"(x)
Bisection	10E6	0.2	0.450183	0.767534	2.435131
Newton	3	5E-6	0.450184	0.767534	2.435131
Secant	5	4E-6	0.450184	0.767534	2.435131

Como podemos observar el método que fue mas rápido fue el de la Secante, aunque si hablamos de iteraciones el algoritmo que menos hizo fue el de Newton Raphson, es importante mencionar que en tiempo solo hay una millonésima de segundo de diferencia. El peor método fue el de Bisección realizo 10 millones de iterasiones y no pudo llegar a la solucion que hace f(x) cercana a 0 con un error de 10E-12. Si bien el método de de Newton y de Bisección son mas rápidos y realizan menos iteraciones, estos no siempre convergen.

En cuanto a la precisión, el método que tiene una millonésima de valor distinta a los demás es el de Bisección.

Para finalizar diremos que el valor minimo y el punto minimo de la funcion (3) son:

$$x = 0.4501836001873$$

$$f(x) = 0.767534424842$$

And then we will show the results for the equation (2).

Figure 6: Bisection

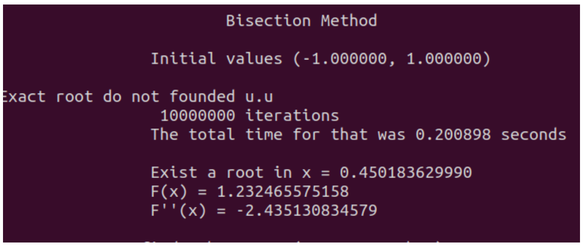


Figure 7: Newton Raphson

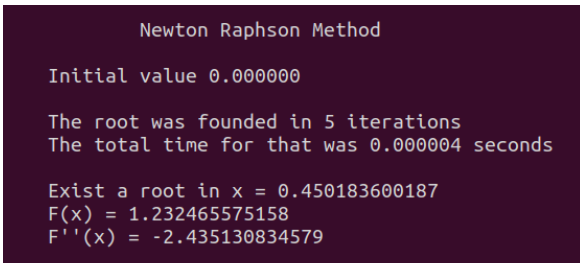
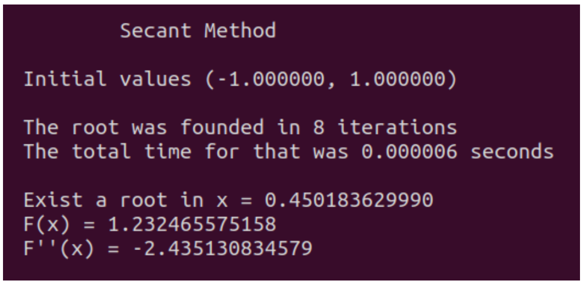


Figure 8: Secant



Resumiremos los resultados en la siguiente tabla

Method	Iterations	Time	x	F(x)	F”(x)
Bisection	10E6	0.2	0.450183	1.232466	-2.435131
Newton	5	5E-6	0.450184	1.232466	-2.435131
Secant	8	4E-6	0.450184	1.232466	-2.435131

Observamos que sucede lo mismo que en la ecuación numero (3), aunque para esta función se realizan un poco mas de iteraciones pero el tiempo de ejecución es muy similar.
Para este caso es importante mencionar que se encontró un punto critico máximo, y esto lo podemos corroborar observando el valor de la segunda derivada, si aplicamos el criterio de la segunda derivada no encontramos con que la x encontrada corresponde a un punto máximo.