Homework 4. Numerical Methods

Carlos Giovanny Encinia González

30 agosto 2021

Problem 1

Implement the following algorithms: Bisection, Newton, and Secant methods.

Answer

Se ha creado un algoritmo en donde se implementan los 3 métodos mencionados anteriormente. Cada uno de los métodos utiliza tres criterios distintos para poder saber cuando encontró la raíz, el primer criterio es el error relativo que toma en cuenta el denominador como el $max\{1, x_0\}$, el segundo criterio es cierto tamaño del $|f(x)|$ y el tercer parámetro es el numero máximo de iteraciones que puede realizar el método. Es importante mencionar que las derivadas de las tres funciones se escribieron dentro del programa como una función mas de C. El algoritmo le pide al usuario que elija entre una de las tres funciones mostradas para poder encontrar la raíz figura (1), una vez hecho esto le pide que seleccione uno de los tres métodos para que pueda realizar su trabaja, mas tarde pide que de dos puntos que pertenecen al intervalo donde se encuentra la raíz figura (2).

Figure 1: Select equation

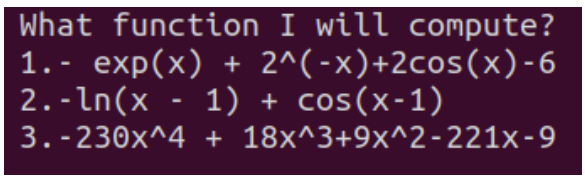
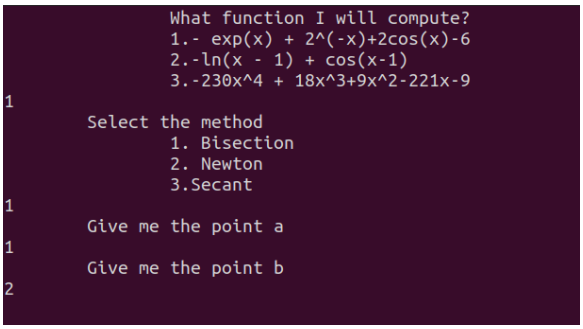


Figure 2: Select method



Una vez dado todos los inputs el algoritmo despliega el intervalo donde se encontró la raíz, el numero de iteraciones que realizo y el tiempo en que se tardo en encontrar la solución figura(3). Al final le pregunta al usuario si quiere seguir encontrando otra raíz, dando la posibilidad de encontrar otra raíz para el mismo método.

Si por alguna razón el usuario da datos que no son números, el programa terminara rápidamente figura(4). Al final del documento se muestra el código del algoritmo. Se separo el programa en una archivo de cabecera, un archivo donde están las funciones necesarias para que funciones el programa y un archivo main.

Figure 3: Get interval

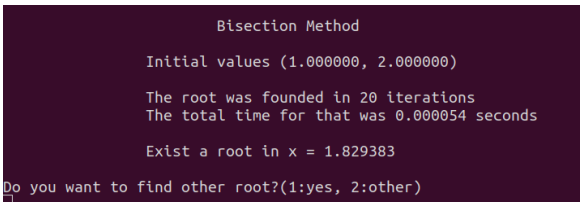
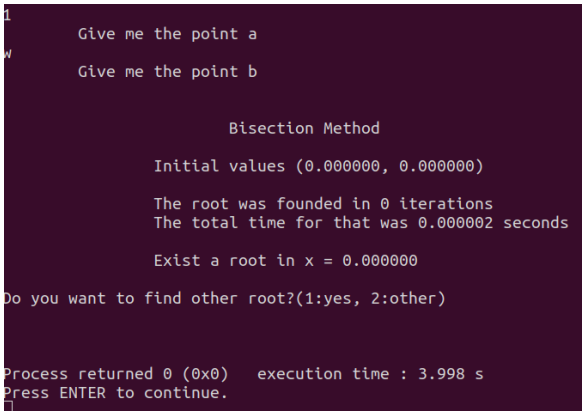


Figure 4: Error interval



Problem 2

Use the previous methods to compute a zero of the following functions:

$$e^x + 2^{-x} + 2cos(x) - 6 = 0, 1 \leq x \leq 2 \quad (1)$$

$$ln(x-1) + cos(x-1) = 0, 1.3 \leq x \leq 2 \quad (2)$$

Answer

En seguida se muestran los resultados obtenidos para cada una de las funciones para cada uno de los métodos que se piden en el ejercicio numero uno. Al reunir la información podemos observar que los algoritmos de bisección y de Newton-Raphson son los que necesitan menos iteraciones y por lo tanto son mucho mas rápidos, tal como lo aprendimos en la teoría. Es importante recordar que estos dos últimos métodos no siempre convergen. Para una misma ecuación vemos que dependiendo del método la respuesta varia en su sexto punto decimal.

1)

Figure 5: Bisection

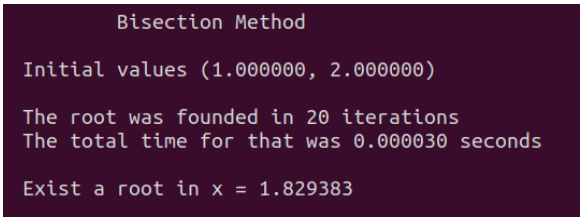


Figure 6: Newton Raphson

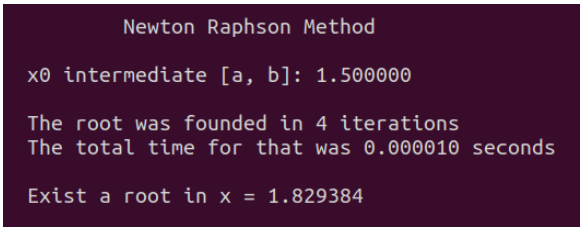
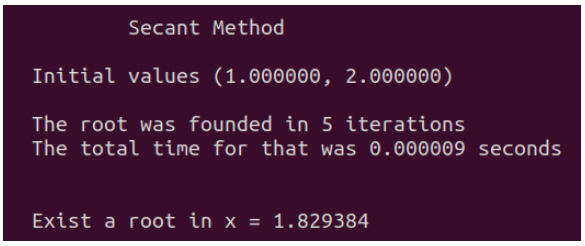


Figure 7: Secant



2)

Figure 8: Bisection

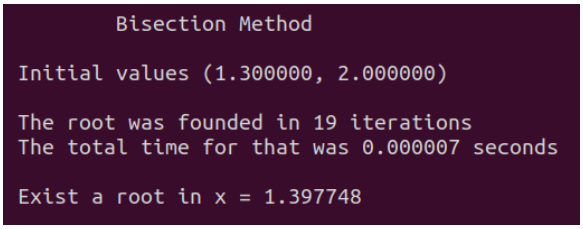


Figure 9: Newton Raphson

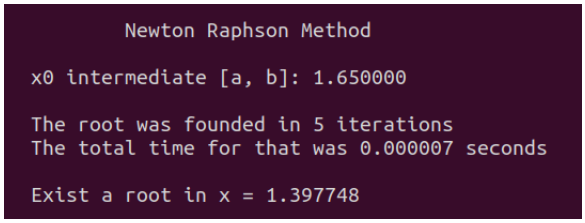
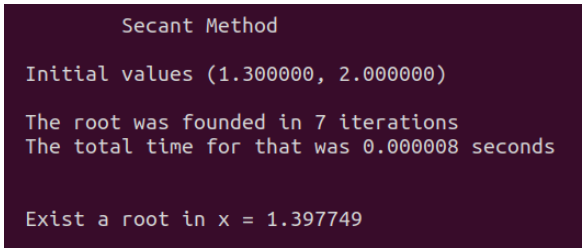


Figure 10: Secant



Un resumen de los resultados de las raices se muestran en la tabla (1).

Función	Rango	Método		
		Bisección	Newton	Secante
(1)	(1, 2)	1.829383	1.829384	1.829384
(2)	(1.3, 2)	1.397748	1.397748	1.397749

Problem 3

The fourth-degree polynomial

$$f(x) = 230x^4 + 18x^3 + 9x^2 - 221x - 9 \quad (3)$$

has two real zeros, one in $[-1, 0]$ and the other in $[0, 1]$. Attempt to approximate these zeros to within 10^{-6} using the algorithms implemented in exercise (1). Use the endpoints of each interval as the initial approximations in the Bisection and Secant algorithms and the midpoints as the initial approximation for the Newton Method.

Answer

Interval $[-1, 0]$

En seguida se muestran los resultados obtenidos para el primer intervalo que se pide, es importante mencionar que para el método de newton el programa toma como punto medio el punto intermedio entre los dos puntos que al inicio del programa se dan.

Como era de esperarse los algoritmos que mas rápido encontraron la solución fueron el que usa el método de la bisección y el que usa el método de Newton-Raphson.

Figure 11: Bisection

```
        Bisection Method

Initial values (-1.000000, 0.000000)

The root was founded in 20 iterations
The total time for that was 0.000023 seconds

Exist a root in x = -0.040659
```

Figure 12: Newton Raphson

```
        Newton Raphson Method

x0 intermediate [a, b]: -0.500000

The root was founded in 4 iterations
The total time for that was 0.000007 seconds

Exist a root in x = -0.040659
```

Figure 13: Secant

```
        Secant Method

Initial values (-1.000000, 0.000000)

The root was founded in 4 iterations
The total time for that was 0.000007 seconds


Exist a root in x = -0.040659
```

Interval $[0, 1]$

En seguida se muestran los resultados obtenidos para este intervalo. Es muy interesante remarcar que para los métodos de bisección y Newton, para el intervalo $[0, 1]$ arrojaban la raíz que se encontró con el intervalo $[-1, 0]$ figura (15) y figura (17). Así que se tuvo que dar un numero a un poco más a la derecha que el original para que nos pudiera dar una raíz distinta.

Figure 14: Bisection

```
        Bisection Method

Initial values (0.000000, 1.000000)

The root was founded in 20 iterations
The total time for that was 0.000012 seconds

Exist a root in x = 0.962398
```

Figure 15: Newton Raphson Wrong

```
        Newton Raphson Method

x0 intermediate [a, b]: 0.500000

The root was founded in 5 iterations
The total time for that was 0.000008 seconds

Exist a root in x = -0.040659
```

Figure 16: Newton Raphson [0.9, 1]

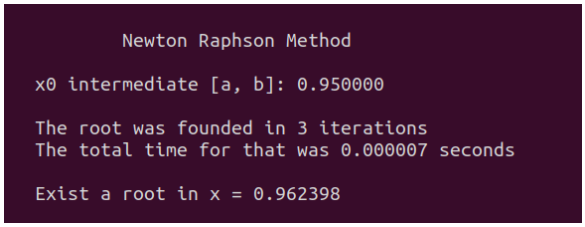


Figure 17: Secant Wrong

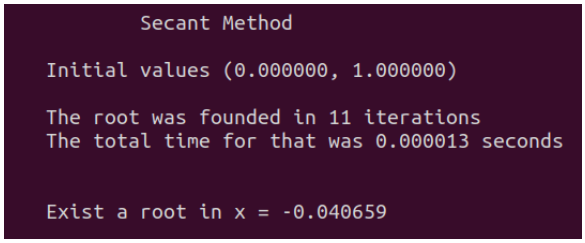
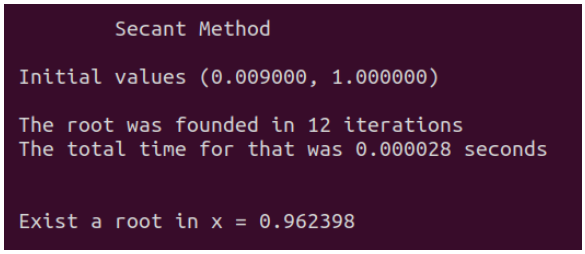


Figure 18: Secant



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 0.000001
#define LIMIT 100000
#define test(x) (x+1)

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

    printf("\t\tWhat function I will compute?\n");
    printf("\t\t1.- exp(x) + 2^(-x)+2cos(x)-6\n");
    printf("\t\t2.-ln(x - 1) + cos(x-1)\n");
    printf("\t\t3.-230x^4 + 18x^3+9x^2-221x-9\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");
```

```
49          printf("\t\t1. Bisection\n\t\t2. Newton\n\t\t3.Secant\n");
            scanf(" %c", &method_sel);
51
            printf("         Give me the point a\n");
53          scanf(" %f", &a);
            printf("         Give me the point b\n");
55          scanf(" %f", &b);
            printf("\n\n");
57

59          switch(method_sel)
            {
61          case '1'://Bisection
                printf("\t\t\tBisection Method\n\n");
63              printf("\t\tInitial values (%f, %f)\n\n", a, b);
                printf("\t\tExist a root in x = %f\n", \
65                 bisection(a, b, f));
                   break;
67          case '2'://Newton Raphson
                printf("\t\t\tNewton Raphson Method\n\n");
69              printf("\t\tx0 intermediate [a, b]: %f\n\n", (a+b)/2);
                printf("\t\tExist a root in x = %f\n", \
71                 newton((a + b)/2, f, f_p));
                   break;
73          case '3'://Secant
                printf("\t\t\tSecant Method\n\n");
75              printf("\t\tInitial values (%f, %f)\n\n", a, b);
                printf("\n\t\tExist a root in x = %f\n", \
77                 secant(a, b, f));
                   break;
79          default:
                printf("Select a correct option\n");
81              return ZERO;

83          }

85          printf("\n");
            printf("Do you want to find other root?(1:yes, 2:other)\n");
87          scanf(" %c", &next);
            printf("\n\n");
89

            if(next!='1')
91          {
                printf("\n\n");
93              break;
            }

95      }

97      return ZERO;
    }
```

Code 2: functions.c

```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 0.000001
#define LIMIT 100000
//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)
{
    return (exp(x) + pow(TWO, (-(x))) + 2*cos(x) - 6);
}

float F_p_x(float x)
{
    return (exp(x) - log(2) / pow(TWO, x) - 2*sin(x));
}

float G_x(float x)
{
    return (log((x) - 1) + cos((x)-1));
}

float G_p_x(float x)
{
    return (1 / (x - 1) - sin(x - 1));
}

float H_x(float x)
{
    return (230*pow(x, 4) + 18*pow(x, 3)+9*pow(x, 2)-221*x-9);
}

float H_p_x(float x)
{
    return (920*pow(x, 3)+54*pow(x, 2)+18*x-221);
}

```

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

52     if(i < LIMIT)
       {
54         printf("\t\tThe root was founded in %d iterations\n", i);
           printf("\t\tThe total time for that was %lf seconds\n\n", time);
56     }
       else
58     {
           printf("Exact root do not founded u.u\n");
60     }

62 }

64 float newton(float x_0, float (*function)(float), float (*f_p)(float))
   {
66     float x = x_0, x_before = x + ONE;
       int i = ZERO;
68     double elapsed;
       clock_t end, start = clock();
70
       //three criteria, error relative, function value, and #iterations
72     while(CRITERIA(x_before, x_0) > EPSILON
              && fabs(function(x_0))>EPSILON
74            && i < LIMIT)
       {
76         if(f_p(x) == ZERO)
               break;
78
           x_before = x_0;
80         x_0  -= function(x_0)/f_p(x_0); //definition Newton's method
           i++;
82     }

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

88     return x_0;
   }
90
   float secant(float a, float b, float (*function)(float))
92 {
       float x_before, f_a, f_b;
94     int i = 0;
       double elapsed;
96     clock_t end, start = clock();
```

```
98      while(CRITERIA(b, a)>EPSILON
              && fabs(function(b))>EPSILON
100           && i < LIMIT)
        {
102         f_a = function(a);
            f_b = function(b);
104         x_before = b; //short the interval
            b  = (a*f_b - b*f_a)/(f_b - f_a);// secant method definition
106         a = x_before;// new a
            i++;
108     }

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

114     return b;
    }
116
    float bisection(float a, float b, float (*function)(float))
118 {
        float x_1;
120     int i = ZERO;
        double elapsed;
122     clock_t end, start = clock();

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

126     while(CRITERIA(a, b) > EPSILON && i < LIMIT)
        {
128

130         if( function(a)*function(x_1) < ZERO)//Bolzano theorem
            {
132             b = x_1;
            }
134         else
            {
136             a = x_1;
            }
138
            x_1 = a + (b - a) /TWO;
140         i++;
        }
142
        end = clock();
144     elapsed = (double)(end - start)/CLOCKS_PER_SEC;
        print_iter(i, elapsed);
146
```

```
    return x_1;
}
```

Code 3: 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);

#endif // FUNCTION
```