Catálogo Grupal de Algoritmos

Integrantes:

- Brayan Alfaro González - Carné 2019380074
- Sebastián Alba Vives - Carné 2017097108
- Kevin Zeledón Salazar - Carné 2018076244
- Daniel Camacho González - Carné 2017114058

1 Tema 1: Métodos para encontrar raíces en ecuaciones no lineales

1.1 Método de la Bisección

Código 1: Método de la Bisección en Lenguaje M

```
% Para ejecutarla en la ventana de comandos usar:
%1) biseccion
%2) f=@(x)x-2; Donde la funcion a estudiar se cambia
%3) [a,e] = biseccion(f,[4,7],10,1e-6) Donde estos parametros cambian
function [a,e] = biseccion(funcion,intervalo,IteracionesMaximas,tolerancia)
elemento_n=0;
error=1/tolerancia;
intervalo(1);
intervalo(2);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
while error(end)>tolerancia && elemento_n<IteracionesMaximas
elemento_n=elemento_n+1;
a=(intervalo(1)+intervalo(2))/2;
if funcion(intervalo(1))*funcion(a)<0
intervalo(2)=a;
else
intervalo(1)=a;
end
error(elemento_n)=abs(intervalo(1)-intervalo(2));
end
e=error(end);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Graficacion
plot(1:elemento_n,error,'--o')
xlabel('Cantidad Iteraciones')
ylabel('Error |f(Xk)|')
title('Gr fico Iteraciones vs Errores')
```

1.2 Método de Newton Raphson

Código 2: Método de Newton Raphson en Python

```
#Importamos las siguientes librerias para que se reconozca cualquier expresion que se
    introduzca.
from sympy import *
from sympy.parsing.sympy_parser import parse_expr #libreria para parsear
import matplotlib.pyplot #libreria para graficar, utilizamos pyplot de matplotlib
from math import * #Libreria matematica

#funcion:La funcion debe ir en formato de string para evitar errores.
#ValorInicial:es el valor inicial de la aproximacion
#tolerancia:tolerancia que se definira para la aproximacion

#Funcion principal
def newton_raphson(funcion,ValorInicial,tolerancia):
    newton_raphson_secundaria(funcion,ValorInicial,tolerancia,[]) #Se crea un vector
       para guardar los valores
#Funcion secundaria para el calculo
def newton_raphson_secundaria(funcion,ValorInicial,tolerancia,Conjunto_Valores):
    Conjunto_Valores.append(abs(ValorInicial)) #Se a ade  el valor absoluto del
       valor inicial al vector
    #El ultimo valor sera la resta del valor inicial menos el cosiente de la
       evaluacion de la funcion con el valor inicial entre la evaluacion de la
       derivada de la funcion con el valor inicial
    Ultimo_Valor =ValorInicial-(EvaluacionGlobal(funcion,ValorInicial)/
       Evaluacion_De_Derivada(funcion,ValorInicial))


    if abs(EvaluacionGlobal(funcion,Ultimo_Valor))<=tolerancia: #Verifica si la
       evaluacion de la funcion con el ultimo valor es menor o igual a la tolerancia
        #A ade el ultimo valor al vector de valores
        Conjunto_Valores.append(Ultimo_Valor)
        print(Ultimo_Valor) #Muestra el ultimo valor
        #Graficacion de los errores
        matplotlib.pyplot.plot(Conjunto_Valores)
        #Label de grafica para texto de "errores"
        matplotlib.pyplot.ylabel("Errores |f(Xk)|")
        #Label de grafica para texto de "Graficacion"
        matplotlib.pyplot.title("Graficaci n Errores vs Iteraciones")
        #Label de grafica para texto de "Iteracion"
        matplotlib.pyplot.xlabel("Iteraci n")
        #Se muestra la grafica
        matplotlib.pyplot.show()

        #Si no cumple la verificacion retorna a la funcion con el ultimo valor en
           lugar del valor inicial
    else:
        newton_raphson(funcion,Ultimo_Valor,tolerancia)

#Modulo de evaluacion
def EvaluacionGlobal(funcion,x):#Recibe la funcion y un valor
    return eval(funcion) #se analiza la expresion de la funcion
#Modulo de evaluacion de derivada
```

```
#En este modulo se evalua la derivada parseando la funcion
def Evaluacion_De_Derivada(funcion,x): #Recibe la funcion y un valor
    expresion = parse_expr(funcion) #Se parsea la funcion y se guarda en "expresion"
    derivada = expresion.diff(Symbol('x'))
    return eval(str(derivada))



#Para ejecturar en la ventana de comandos basta con utilizar: newton_raphson("funcion
   ",ValorInicial,tolerancia) y cambiar valores a estudiar
```

1.3 Método de la Secante

Código 3: Método de la Secante en C++

```
#include <iostream>
#include <cmath>
#include <ginac/ginac.h>
#include <cln/dfloat_io.h>
#include "matplotlibcpp.h"
using namespace std;
using namespace GiNaC;
using namespace cln;
namespace plt = matplotlibcpp;

/** Metodo que realiza la evaluacion de una funcion en un valor especifico dado
    utilizando la libreria GiNaC
* @param funct La funcion a evaluar en formato de string
* @param value El valor en el cual se va a evaluar la funcion
*/
ex f(string funct, ex value){
  symbol x;
  symtab table;
  table["x"]=x;
  parser reader(table);
  ex f = reader(funct);
  return evalf(f.subs(x==value));
}

/** Funcion que calcula la aproximacion de un cero de una funcion dada mediante el
    metodo de secante
* @param x0 First initial value
* @param x1 Second initial value
* @param maxIt Max Iterations for the function
* @param tol Tolerancia de la funcion
* @param funct Funcion a evaluar
*/
ex * secante(double x0, double x1, int maxIt, string funct, ex tol){
  vector<double> graphX(maxIt), graphY(maxIt);
  ex xk = x1;
  ex xkm1 = x0;
  ex xkp1;
  int i=0;
  ex err=tol+1;
  static ex r[2]; // Array de resultado
  while(i<maxIt && err>tol){
    // Nuevo resultado
    xkp1 = xk-f(funct,xk)*(xk-xkm1)/(f(funct,xk)-f(funct,xkm1));
    // Reasignando valores para la siguiente iteracion
    xkm1 = xk;
    xk = xkp1;
    err = abs(f(funct, xk));
    // Para la grafica
    graphX.push_back(ex_to<numeric>(i).to_double());
    graphY.push_back(ex_to<numeric>(abs(f(funct,xk))).to_double());
    // Contador de iteracion
```

```
    i+=1;
  }
  plt::plot(graphX,graphY);
  plt::show();
  r[0] = xk; // Aproximacion
  r[1] = abs(f(funct,xk)); // Error
  return r;
}

int main(void)
{
  ex *r;
  r = secante(0.75,0.437193, 5, "(cos(2*x)^(2))-((x)^(2))",1e-10);
  cout << "Aproximacion: "<<*r<<endl;
  cout << "Error: " << *(r+1) << endl;
  return 0;
}
```

1.4 Método de la Falsa Posición

Código 4: Método de la Falsa Posición en C++

```
#include <cmath>
#include <iostream>
#include <ginac/ginac.h>
#include "matplotlibcpp.h"
using namespace std;
using namespace GiNaC;
namespace plt = matplotlibcpp;

/** Metodo encargado de evaluar una funcion representada en string en un valor
    especifico utilizando la libreria GiNaC
* @param funct La funcion que se quiere evaluar en formato de string
* @param value El valor en el cual se va a evaluar la funcion
*/
ex f(string funct, ex value){
  symbol x;
  symtab table;
  table["x"] = x;
  parser reader(table);
  ex f = reader(funct);
  return evalf(f.subs(x==value));
}

/** Metodo encargado de calcular una solucion aproximada a una funcion utilizando el
    metodo de falsa posicion
* @param funct String de la funcion a la cual se le va a calcular el cero aproximado
* @param startInt El inicio del intervalo
* @param endInt El final del intervalo
* @param maxIt Las iteraciones maximas que va a realizar la funcion
* @param tol La tolerancia maxima permitida
*/
ex * falsa_posicion(string funct,ex startInt, ex endInt, int maxIt, ex tol){
  ex a = startInt;
  ex b = endInt;
  ex xk = a-((a-b)/(f(funct, a)-f(funct, b)))*f(funct, a);
  ex functXk;
  vector<double> graphX(maxIt), graphY(maxIt);
  int i=0;
  static ex r[2];
  ex err = tol+1;
  while(i<maxIt && err>tol){
    functXk = f(funct, xk);
    if(f(funct, a)*functXk<0){
      xk = xk-functXk*(xk-a)/(functXk-f(funct, a));
      // Para la siguiente iteracion
      b = xk;
    }else if(functXk*f(funct, b)<0){
      xk = xk-functXk*(xk-b)/(functXk-f(funct, b));
      // Para la siguiente iteracion
      a = xk;
    }else{
      cout << "No se puede resolver por falsa posicion" << endl;
```

```
      return 0;
    }
    // Grafica
    graphX.push_back(i);
    graphY.push_back(ex_to<numeric>(abs(f(funct,xk))).to_double());
    i+=1;
  }
  plt::plot(graphX, graphY);
  plt::show();
  r[0] = evalf(xk);
  r[1] = abs(f(funct, xk));
  return r;
}

int main(void){
  ex *r;
  r = falsa_posicion("cos(x)-x", 1/2, Pi/4, 4, 1e-10);
  cout << "Aproximacion: " << *r << endl;
  cout << "Error: " << *(r+1) << endl;
}
```

1.5 Método del Punto Fijo

Código 5: Método del Punto Fijo en Python

```
import sympy as S
import matplotlib.pyplot as plt

# Utiliza las librer as sympy y matplotlib


# Funcion para calcular ceros de funcion con el Metodo del Punto Fijo
# Entradas:
#       funcion : la funcion en texto
#       valorInicial : estimacion del cero
#       iterMax : iteraciones maximas
# Salidas:
#       [cero aproximado, error]

def punto_fijo(funcion, valorInicial, iterMax):
    # Se establece el simbolo x
    x = S.symbols("x")
    # Se lee la funcion
    funcion = S.sympify(funcion)
    # Se convierte a funcion de python
    funcion = S.lambdify(x, funcion)

    # Se inicializan los valores de la iteraci n y el array de errores
    x_k = valorInicial
    error_array= [float('inf')]

    try:
        #Se agrega el primer error
        error_array=[funcion(x_k)]

        # Se itera
        for k in range(0, iterMax):

            # Se calcula el valor de la siguiente aproximaci n
            x_k = funcion(x_k)
            error_array.append(funcion(x_k))
    except:
        pass

    #Se inicializan el plot
    fig, ax = plt.subplots()

    fig.suptitle("Error en el M todo de Punto Fijo")
    plt.xlabel("Iteraciones")
    plt.ylabel("Error")

    #Se actualiza el plot
    ax.scatter(list(range(1,k+3)), error_array, s=6, c="red")
    plt.show()

    # Se retorna el cero y el error
```

```
    return [x_k, error_array[-1]]


# Ejemplo Numerico
def prueba():
    [x_k, error] = punto_fijo("sin(x)", 2, 1000)
    print("Error: " + str(error) + " |  Cero: " + str(x_k))

prueba()
```

1.6 Método de Muller

Código 6: Método de Muller en Lenguaje M

```
function muller
   f='sin(x) - x / 2';
   x0=2;
   x1=2.2;
   x2=1.8;
   iterMax=100;
   tol=10^-68;
   [rx,error]=mullerAux(f,x0,x1,x2,tol,iterMax)
end

function[rx,error]=mullerAux(f,x0,x1,x2,tol,iterMax)
  pkg load symbolic
  syms x;
  f1=sym(f);
  f=matlabFunction(f1);
  rx=0;
  e=[];
  for i=1:iterMax

    a = ((x1 - x2) * (f(x0) - f(x2)) - (x0 - x2) * (f(x1) - f(x2))) / ((x0 - x1) * (x0 - x2) * (x1 - x2))
        ;
    b = (((x0 - x2) ** 2) * (f(x1) - f(x2)) - ((x1 - x2) ** 2) * (f(x0) - f(x2))) / ((x0 - x1) * (x0 - x2
        ) * (x1 - x2));
    c = f(x2);

    rx=x2 - ((2 * c) / (b + (b / abs(b)) * sqrt(b ** 2 - 4 * a * c)));
    error=abs(f(rx));
    e=[e error];
    if error<tol
      break
    endif
    rx0=abs(rx-x0);
    rx2=abs(rx-x2);

    if rx0<rx2
      x2=x1;
      x1=rx;
    else
      x0=x1;
      x1=rx;
    endif
  endfor
  plot(e)
  display(["Aproximacion: ",num2str(rx)]);
  display(["Iteraciones:",num2str(i)]);
  display(["Error relativo Normalizado:",num2str(error)]);

end
```