

Catálogo Grupal de Algoritmos

Integrantes:

- Josué Araya García - 2017103205
- Jonathan Guzmán Araya - 2013041216
- Mariano Muñoz Masís - 2016121607
- Luis Daniel Prieto Sibaja - 2016072504

Índice

1. Tema 1: Ecuaciones no Lineales	1
1.1. Método 1: Bisección	1
1.2. Método 2: Newton-Raphson	3
1.3. Método 3: Secante	4
1.4. Método 4: Falsa Posición	6
1.5. Método 5: Punto Fijo	8
1.6. Método 6: Muller	9
2. Optimización	11
3. Sistemas de Ecuaciones	11
4. Polinomio de Interpolación	11
5. Integración Numérica	11
6. Diferenciación Numérica	11
7. Valores y Vectores Propios	11

1. Tema 1: Ecuaciones no Lineales

1.1. Método 1: Bisección

Código 1: Lenguaje M.

```
%[
```

```

Metodo de la Biseccion
Parametros de Entrada
    @param f: funcion a la cual se le aplicara el algoritmo
    @param a: limite inferior del intervalo
    @param b: limite superior del intervalo
    @param MAXIT: iteraciones maximas
    @param TOL: tolerencia del algoritmo

Parametros de Salida
    @return xAprox: valor aproximado de x
    @return error: porcentaje de error del resultado obtenido
%}

clc;
clear;

function [xAprox, err] = biseccion(f, a, b, MAXIT, TOL)

    if(f(a) * f(b) < 0)

        iter = 1;
        err = 1;
        iterl = []; % Lista que almacena el numero de iteraciones para despues graficar
        errl = []; % Lista que almacena el % de error de cada iteracion para despues graficar

        while(iter < MAXIT)
            xAprox = (a + b) / 2;
            fx = f(xAprox);

            if(f(a) * fx < 0)
                b = xAprox;
            elseif(f(b) * fx < 0)
                a = xAprox;
            endif

            iterl(iter) = iter;
            errl(iter) = err;
            err = (b - a) / (2)^(iter-1);

            if(err < TOL)
                grafica(iterl, errl);
                return;
            else
                iter = iter + 1;
            endif
        endwhile
        grafica(iterl, errl);
    else
        error("Condiciones en los parametros de entrada no garantizan el cero de la funcion.")
    endif
    return;
endfunction

%{

```

```

Parametros de Entrada
    @param listaValoresX: valores del eje 'x'
    @param listaValoresY: valores del eje 'y'

Parametros de Salida
    @return: Grafico de los datos ingresados
%}
function grafica(listaValoresX, listaValoresY)
    plot(listaValoresX, listaValoresY, 'bx');
    title("Metodo de la Biseccion");
    xlabel("Iteraciones");
    ylabel("% Error");
endfunction

%Valores iniciales
a = 0;
b = 2;
%Iteraciones maximas
MAXIT = 100;
%Tolerancia
TOL = 0.0001;
%Funcion
funct = @(x) e^x - x - 2;
%Llamado de la funcion
[xAprox, err] = biseccion(funct, a, b, MAXIT, TOL);
printf("##### \n");
printf("Metodo de la Biseccion \n");
printf('xAprox = %f\n%Error = %d \n', xAprox, err);

```

1.2. Método 2: Newton-Raphson

Código 2: Lenguaje Python.

```

# Metodo de Newton-Raphson
# Entradas:
    #func: es la funcion a analizar
    #x0: valor inicial
    #MAXIT: es la cantidad de iteraciones maximas a realizar
    #TOL: es la tolerancia del algoritmo
# Salidas:
    #xAprox: es la solucion, valor aproximado de x
    #error: pocentaje de error del resultado obtenido

#####
import math
import matplotlib.pyplot as plt
from scipy.misc import derivative
#####

def newtonRaphson(func, x0, MAXIT, TOL):
    itera = 1
    err = 1
    iterl = [] #Lista que almacena el numero de iteraciones

```

```

errl = [] #Lista que almacena el % de error de cada iteracion
xAprox = x0

while (itera < MAXIT):
    xk = xAprox
    fd = derivative(func, xk, dx=1e-6)
    xAprox = xk - (func(xk)) / (fd)
    err = (abs(xAprox - xk)) / (abs(xAprox))
    iterl.append(itera)
    errl.append(err)

    if(err < TOL):
        grafica(iterl, errl)
        return xAprox, err
    else:
        itera = itera + 1

grafica(iterl, errl)
return xAprox, err

#Grafica
#Entradas:
    #listaValoresX: valores que se graficaran en el eje 'x'
    #listaValoresY: valores que se graficaran en el eje 'y'
#Salidas:
    #Grafico con los valores ingresados
def grafica(listaValoresX, listaValoresY):
    plt.plot(listaValoresX, listaValoresY, 'bx')
    plt.title("Metodo de Newton-Raphson")
    plt.xlabel("Iteraciones")
    plt.ylabel("% Error")
    plt.show()

if __name__ == '__main__':
    #Valor inicial
    x0 = 1
    #Tolerancia
    TOL = 0.0001
    #Maximo iteraciones
    MAXIT = 100
    #Funcion
    func = lambda x: (math.e)**x - 1/x
    #Llamado de la funcion
    xAprox, err = newtonRaphson(func, x0, MAXIT, TOL)
    print("#####")
    print("Metodo de Newton-Raphson \n")
    print('xAprox = {}\n%Error = {}'.format(xAprox, err))

```

1.3. Método 3: Secante

Código 3: Lenguaje C++.

```

#include <iostream>

```

```

#include <ginac/ginac.h>
#include "mgl2/mgl.h"
#include <vector>

using namespace std;
using namespace GiNaC;

/*Funcion para crear una grafica:
 * Entradas: Pares ordenados en x y y, vectores de las graficas
 * Salidas: Grafica de iteraciones vs error*/
void createGraph(double x1, double x2, double y1, double y2, vector<double> x, vector<double> y) {
    mglGraph graph;
    //Estas funciones convierten los vectores de la entrada en arreglos de datos de la grafica
    mglData xGraph(x);
    mglData yGraph(y);
    //Se diseña la grafica con los parametros
    graph.Title("Error vs Iteracion");
    graph.SetOrigin(0, 0);
    graph.SetRanges(x1, x2, y1, y2); //Limites de la grafica
    graph.Plot(xGraph, yGraph, "o!rgb"); //Valores que va a contener la grafica
    graph.Axis();
    graph.Grid();
    //Se exporta la grafica a un archivo PNG
    graph.WritePNG("Graph.png");
}

/*Metodo de la secante:
 * Entradas: Funcion a la que se le va a aplicar el metodo (express), primer valor inicial,
    valor inicial, tolerancia y cantidad de iteraciones maximas
 * Salidas: Aproximacion de la solucion, error y cantidad de iteraciones realizadas*/
ex secante(string express, string firstValue, string secondValue, string tolerance, string iterations) {
    //Implementación del calculo simbolico
    symbol x("x");
    syntab table;
    table["x"] = x;
    parser reader(table);
    //Se traducen las entradas a variables de calculo simbolico
    ex function = reader(express);
    ex x0 = reader(firstValue);
    ex x1 = reader(secondValue);
    ex tol = reader(tolerance);
    ex iterMax = reader(iterations);
    //Se definen las variables de la iteracion, solucion y error necesarias
    int iter = 1;
    ex xk;
    ex error = tol + 1;
    //Vectores para la grafica
    vector<double> errors;
    vector<double> iters;
    //Funciones por evaluar
    ex f0 = evalf(subs(function, x == x0));
    ex f1 = evalf(subs(function, x == x1));
    while (iter < iterMax) {
        //Ecuacion del metodo de la secante

```

```

        xk = x1 - f1 * ((x1 - x0) / (f1 - f0));
        error = abs(xk - x1)/abs(xk); //Error de la solución
        ex aux = evalf(error);
        //Se actualizan los valores
        x0 = x1;
        x1 = xk;
        iter++;
        //Los vectores de iteración y error reciben valores
        double m = ex_to<numeric>(aux).to_double();
        errors.push_back(m);
        iters.push_back(iter);
        //Condición de parada
        if (error <= tol) {
            break;
        }
    }
    cout << "Aproximacion: " << xk << endl;
    cout << "Iteraciones : " << iter << endl;
    cout << "Error : " << error << endl;
    //Se crea la grafica respectiva
    createGraph(0, iter + 1, -ex_to<numeric>(evalf(error)).to_double(), ex_to<numeric>(e
    return xk;
}

int main() {
    //Se define la función por evaluar
    string express;
    cout << "Escriba la función: " << endl;
    cin >> express;
    //Se definen los valores iniciales
    string x0;
    cout << "Escriba el primer valor inicial: " << endl;
    cin >> x0;
    string x1;
    cout << "Escriba el segundo valor inicial: " << endl;
    cin >> x1;
    //Se define la tolerancia
    string tol;
    cout << "Escriba la tolerancia: " << endl;
    cin >> tol;
    //Se define el número máximo de iteraciones
    string iterMax;
    cout << "Escriba el número de iteraciones: " << endl;
    cin >> iterMax;
    //Metodo de la secante
    secante(express, x0, x1, tol, iterMax);
    return 0;
}

```

1.4. Método 4: Falsa Posición

Código 4: Lenguaje C++.

```

#include <iostream>
#include <ginac/ginac.h>
#include "mgl2/mgl.h"
#include <vector>

using namespace std;
using namespace GiNaC;

/*Funcion para crear una grafica:
 * Entradas: Pares ordenados en x y y, vectores de las graficas
 * Salidas: Grafica de iteraciones vs error*/
void createGraph(double x1, double x2, double y1, double y2, vector<double> x, vector<double> y) {
    mglGraph graph;
    //Estas funciones convierten los vectores de la entrada en arreglos de datos de la grafica
    mglData xGraph(x);
    mglData yGraph(y);
    //Se diseña la grafica con los parametros
    graph.Title("Error vs Iteracion");
    graph.SetOrigin(0, 0);
    graph.SetRanges(x1, x2, y1, y2); //Limites de la grafica
    graph.Plot(xGraph, yGraph, "o!rgb"); //Valores que va a contener la grafica
    graph.Axis();
    graph.Grid();
    //Se exporta la grafica a un archivo PNG
    graph.WritePNG("Graph.png");
}

/*Metodo de la secante:
 * Entradas: Funcion a la que se le va a aplicar el metodo (express), primer valor inicial,
    valor inicial, tolerancia y cantidad de iteraciones maximas
 * Salidas: Aproximacion de la solucion, error y cantidad de iteraciones realizadas*/
ex secante(string express, string firstValue, string secondValue, string tolerance, string iterations) {
    //Implementación del calculo simbolico
    symbol x("x");
    syntab table;
    table["x"] = x;
    parser reader(table);
    //Se traducen las entradas a variables de calculo simbolico
    ex function = reader(express);
    ex x0 = reader(firstValue);
    ex x1 = reader(secondValue);
    ex tol = reader(tolerance);
    ex iterMax = reader(iterations);
    //Se definen las variables de la iteracion, solucion y error necesarias
    int iter = 1;
    ex xk;
    ex error = tol + 1;
    //Vectores para la grafica
    vector<double> errors;
    vector<double> iters;
    //Funciones por evaluar
    ex f0 = evalf(subs(function, x == x0));
    ex f1 = evalf(subs(function, x == x1));
    while (iter < iterMax) {

```

```

        //Ecuacion del metodo de la secante
        xk = x1 - f1 * ((x1 - x0) / (f1 - f0));
        error = abs(xk - x1)/abs(xk); //Error de la solucion
        ex_aux = evalf(error);
        //Se actualizan los valores
        x0 = x1;
        x1 = xk;
        iter++;
        //Los vectores de iteracion y error reciben valores
        double m = ex_to<numeric>(aux).to_double();
        errors.push_back(m);
        iters.push_back(iter);
        //Condicion de parada
        if (error <= tol) {
            break;
        }
    }
    cout << "Aproximacion: " << xk << endl;
    cout << "Iteraciones : " << iter << endl;
    cout << "Error : " << error << endl;
    //Se crea la grafica respectiva
    createGraph(0, iter + 1, -ex_to<numeric>(evalf(error)).to_double(), ex_to<numeric>(e
    return xk;
}

int main() {
    //Se define la funcion por evaluar
    string express;
    cout << "Escriba la funcion: " << endl;
    cin >> express;
    //Se definen los valores iniciales
    string x0;
    cout << "Escriba el primer valor inicial: " << endl;
    cin >> x0;
    string x1;
    cout << "Escriba el segundo valor inicial: " << endl;
    cin >> x1;
    //Se define la tolerancia
    string tol;
    cout << "Escriba la tolerancia: " << endl;
    cin >> tol;
    //Se define el numero maximo de iteraciones
    string iterMax;
    cout << "Escriba el numero de iteraciones: " << endl;
    cin >> iterMax;
    //Metodo de la secante
    secante(express, x0, x1, tol, iterMax);
    return 0;
}

```

1.5. Método 5: Punto Fijo

Código 5: Lenguaje Python.

```

import matplotlib.pyplot as plt
import numpy as np

#Punto Fijo
#Entradas: funcion - Funcion por aproximar - funcion lambda
#valor - inicial - Valor por el cual se empezara a aproximar - int, float, double
#iteraciones - maximas - Numero maximo de itreaciones - int
#
#

def punto_fijo(funcion,valor_inicial,iteraciones_maximas):
    lista_error = [] #lista para graficar
    iteracion = 1
    b = funcion(valor_inicial) #valor para obtener error
    error = abs(b-valor_inicial)
    while(iteracion<=iteraciones_maximas ): #condicion de parada
        valor_inicial = b #reajuste de valores de error
        b = funcion(valor_inicial)
        error = abs(b-valor_inicial)
        lista_error.append(error)
        iteracion += 1

    aproximacion = b
    plt.plot(lista_error, label='errores por iteracion') #Construccion de tabla
    plt.ylabel('Error')
    plt.xlabel('Iteracion')

    plt.axis([0, iteraciones_maximas,0,lista_error[0]]) #Los ejes estan limitados p
    plt.title('Punto Fijo')
    plt.legend()
    plt.show()
    print ('Aproximacion: '+ str(aproximacion)+ ', error: '+ str(error))
    return aproximacion, error

funcion =lambda x: np.exp(-x)
punto_fijo(funcion, 0, 15)

```

1.6. Método 6: Muller

Código 6: Lenguaje M.

```

%{
Metodo de Muller
Parametros de Entrada
    @param func: funcion a la cual se le aplicara el algoritmo
    @param x0: primer valor inicial
    @param x1: segundo valor inicial
    @param x2: segundo valor inicial
    @param MAXIT: iteraciones maximas
    @param TOL: tolerancia del algoritmo

Parametros de Salida

```

```

        @return r: valor aproximado de x
        @return error: porcentaje de error del resultado obtenido
    %}

clc;
clear;

function [r, err] = muller(func, x0, x1, x2, MAXIT, TOL)
    iter = 1;
    err = 1;
    iterl = []; %Lista que almacena el numero de iteraciones para despues graficar
    errl = []; %Lista que almacena el % de error de cada iteracion para despues graficar

    while(iter < MAXIT)

        a = ((x1 - x2)*[func(x0) - func(x2)] - (x0 - x2)*[func(x1) - func(x2)]) / ((x0 - x1)*(x0 - x2)*(
            x1 - x2));
        b = (((x0 - x2)^2)*[func(x1) - func(x2)] - ((x1 - x2)^2)*[func(x0) - func(x2)]) / ((x0 - x1)*(x0
            - x2)*(x1 - x2));
        c = func(x2);

        discriminante = b^2 - 4*a*c;

        if(discriminante < 0)
            error("Error, la solucion no es real.")
            return;
        endif

        r = x2 - (2*c) / (b + (sign(b))*(sqrt(discriminante)));
        err = (abs(r - x2)) / (abs(r));
        errl(iter) = err;
        iterl(iter) = iter;
        iter = iter + 1;

        if(err < TOL)
            grafica(iterl, errl);
            return;
        endif

        x0Dist = abs(r - x0);
        x1Dist = abs(r - x1);
        x2Dist = abs(r - x2);

        if (x0Dist > x2Dist && x0Dist > x1Dist)
            x0 = x2;
        elseif (x1Dist > x2Dist && x1Dist > x0Dist)
            x1 = x2;
        endif
        x2 = r;
    endwhile

    grafica(iterl, errl);
    return;
endfunction

```

```

%{
    Parametros de Entrada
    @param listaValoresX: valores del eje 'x'
    @param listaValoresY: valores del eje 'y'

    Parametros de Salida
    @return: Grafico de los datos ingresados
}%}
function grafica(listaValoresX, listaValoresY)
    plot(listaValoresX, listaValoresY, 'bx');
    title("Metodo de Muller");
    xlabel("Iteraciones");
    ylabel("% Error");
endfunction

%Valores iniciales
x0 = 2;
x1 = 2.2;
x2 = 1.8;
%Iteraciones maximas
MAXIT = 100;
%Tolerancia
TOL = 0.0000001;
%Funcion
func = @(x) sin(x) - x/2;
%llamado de la funcion
[r, err] = muller(func, x0, x1, x2, MAXIT, TOL);
printf("##### \n");
printf("Metodo de Muller \n");
printf('r = %f\n%Error = %f \n', r, err);

```

2. Optimización

3. Sistemas de Ecuaciones

4. Polinomio de Interpolación

5. Integración Numérica

6. Diferenciación Numérica

7. Valores y Vectores Propios