

# Catálogo Grupal de Algoritmos

## Integrantes:

- Josué Araya García - Carnet
- Jonathan Guzmán Araya - Carnet
- Mariano Muñoz Masís - Carnet
- Daniel Prieto - Carnet

## 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: tolerancia 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;
    iter = iter + 1;
    err = (b - a) / (2)^(iter-1);

    if(err < TOL)
        plot(iterl, errl, 'bx');
        title("Metodo de la Biseccion");
        xlabel("Iteraciones");
        ylabel("% Error");
        return;
    endif
endwhile

plot(iterl, errl, 'bx');
title("Metodo de la Biseccion");
xlabel("Iteraciones");
ylabel("% Error");
else
    error("Condiciones en los parametros de entrada no garantizan el cero de la funcion.")
endif
return;
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('xAprox = %f\n%Error = %d \n', xAprox, err);

```

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 para graficar
    errl = [] #Lista que almacena el % de error de cada iteracion para graficar
    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):
            plt.plot(iterl, errl, 'bx')
            plt.title("Metodo de Newton-Raphson")
            plt.xlabel("Iteraciones")
            plt.ylabel("% Error")
            plt.show()
            return xAprox, err
        else:
            itera = itera + 1

    plt.plot(iterl, errl, 'bx')
    plt.title("Metodo de Newton-Raphson")
    plt.xlabel("Iteraciones")
    plt.ylabel("% Error")
    plt.show()
    return xAprox, err

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('xAprox = {}\n%Error = {}'.format(xAprox, err))

```

Código 3: Lenguaje C++.

```

#include<iostream>
#include<cmath>

```

```

using namespace std;

double F(double x) {
    return exp(x) - x - 2;
}

double Biseccion(double a, double b, int MAXIT, double TOL) {
    int cont = 1;
    double x;
    double fx;

    while(cont < MAXIT) {
        x = (a + b)/ 2;
        fx = F(x);
        if(F(a) * fx < 0) {
            b = x;
        }
        if(F(b) * fx < 0) {
            a = x;
        }
        if(abs(fx) < TOL) {
            return x;
        }
        cont = cont + 1;
    }
    return x;
}

int main (int argc, char *argv[]) {
    cout<< Biseccion(0, 2, 100, 0.000001)<<endl;
    system("pause");
    return 0;
}

```

Código 4: 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)
        plot(iterl, errl);
        title("Metodo de Muller");
        xlabel("Iteraciones");
        ylabel("% Error");
        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

plot(iterl, errl);
title("Metodo de Muller");
xlabel("Iteraciones");
ylabel("% Error");
return;
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('r = %f\n%Error = %i \n', r, err);

```