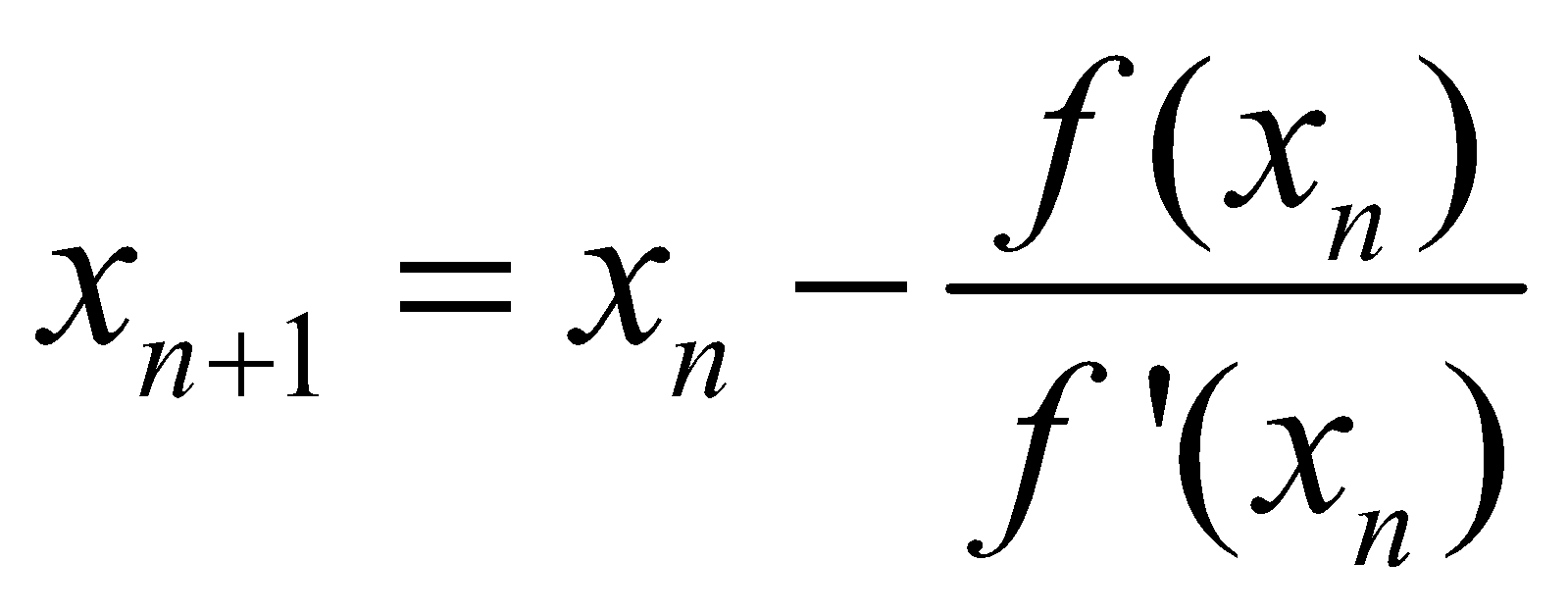
|  |  |
| --- | --- |
| NEWTONRAPHSON MANUAL DE DESARROYADOR | Javier Emmanuel González Orozco  22110330 |

# **Introducción**

# El método Newton-Raphson es un algoritmo utilizado para hallar las raíces de una función. Es un método iterativo que utiliza la derivada de la función para mejorar la precisión de la estimación de la raíz en cada iteración.



## Dentro de la estructura del código se puede encontrar el funcionamiento de cada cosa en forma de notas

#include <iostream>

#include <iomanip> // setprecision

#include <cmath>

#define PRECISION 10

#define MAX\_ITERACIONES 100

#define INTERVALOS 6

using namespace std;

void tabula(double a, double b, int intervalos);    // Muestra un # tabulado de intervalos

double f(double x); // Retorna el valor de la función evaluada en x

double f\_derivada(double x); // Retorna la derivada de la función evaluada en x

int main()

{

    double a;

    double b;

    double tolerancia;  // Tolerancia

    double x0; // Primera aproximación

    double x1; // Siguiente aproximación

    double error;   // Diferencia entre dos aproximaciones sucesivas: x1 - x0

    int iteracion; // # de iteraciones

    bool converge = true;

    cout << setprecision(PRECISION);    // Se establece la precisión

    cout << "\nCalculo de las raices de una funcion aplicando el metodo de Newton - Raphson\n";

    cout << "\nIngrese el intervalo inicial [a,b]:" << endl;

    // Se ingresa el intervalo

    cout << "\na = ";

    cin >> a;

    cout << "b = ";

    cin >> b;

    // Se tabulan los valores de f para INTERVALOS intervalos

    tabula(a, b, INTERVALOS);

    // Se pide elegir una aproximación inicial

    cout << "\nEscoja el punto inicial adecuado:   x0 = ";

    cin >> x0;

    // Se pide ingresar la tolerancia

    cout << "Tolerancia = ";

    cin >> tolerancia;

    // Iteraciones

    // Se imprimen los valores de la primera aproximación

    cout << "\nAproximacion inicial:\n";

    cout << "x0 = " << x0 << "\n"

        << "f(x0) = " << f(x0) << "\n"

        << "f'(x0) = " << f\_derivada(x0) << endl;

    iteracion = 1;

    do {

        if (iteracion > MAX\_ITERACIONES) {

            converge = false;   // Se sobrepasó la máxima cantidad de iteraciones permitidas

            break;

        } else {

            x1 = x0 - f(x0) / f\_derivada(x0); // Cálculo de la siguiente aproximación

            error = fabs(x1 - x0);  // El error es la diferencia entre dos aproximaciones sucesivas

            // Se imprimen los valores de la siguiente aproximación x1, f(x1), f\_derivada(x1), error

            cout << "\nIteracion #" << iteracion << endl;

            cout << "x" << iteracion << " = " << x1 << "\n"

              << "f(x" << iteracion << ") = " << f(x1) << "\n"

              << "f'(x" << iteracion << ") = " << f\_derivada(x1) << "\n"

              << "error = " << error << endl;

            // La diferencia entre dos aproximaciones sucesivas es también conocida como error.

            // La condición de terminación consiste en que que el error debe ser <= que la tolerancia dada

            // Si se cumple la condición de terminación, se ha encontrado la raiz aproximada buscada.

            if (error <= tolerancia) { // Condición de terminación

                converge = true;

                break;

                // Si no se cumple el criterio de terminación, se pasa a la siguiente iteración

            } else {

                x0 = x1;

                iteracion++;

            }

        }

    } while (1);

    // Respuesta final

    if (converge) {

        cout << "\n\nPara una tolerancia de " << tolerancia << " la raiz de f es: " << x1 << endl;

    } else {

        cout << "\n\nSe sobrepasó la máxima cantidad de iteraciones permitidas" << endl;

    }

    cin.get();

    cin.get();

    return 0;

}

void tabula(double a, double b, int intervalos)

{

    int puntos = intervalos + 1;

    double ancho = (b - a) / intervalos;

    cout << "\n\tx\t\tf(x) " << endl;

    for (int i = 0; i < puntos; i++) {

        cout << "\t" << a << "\t\t" << f(a) << endl;

        a = a + ancho;

    }

}

double f(double x)

{

    return x \* exp(cos(x)) / 1.5 - 1;

    //return exp(-x) + 3 \* x - 3;

}

double f\_derivada(double x)

{

    return exp(cos(x)) \* (1 - x \* sin(x)) / 1.5;

    //return -1 \* exp(-x) + 3;

}