Experiment-4
Root Finding - Newton's Method

Purpose: The aim of this lab is to find a root of a non-polynomial function by performing the Newton's method.

Introduction

The Newton's method is a numerical technique for root finding. It begins the search for a root with some initial guess. Then, it fits a tangent to the function through the guess and finds the root of the tangent line which becomes the next guess. This process continues until the difference between the guess and the root of the tangent line is small. Newton's method uses the derivative of the function to find the slope of the tangent line.

The idea belongs to Newton, and Raphson writes the equation in the final form so the method is sometimes called by the two. In other root-finding methods such as secant and bisection, at least two points and their outputs are required, while in Newton method one point and its output, and the derivative value with the tangent line passing through that point are sufficient. It is one of the most used methods for optimization in science and engineering because it is simple and easily applicable.

The Newton method is based on solving the scalar equation f(x) = 0, where f is a continuous function. A tangent line is drawn using the known point $(x_n, f(x_n))$ on the graph of f(x). After the derivative of $f(x_n)$, $f'(x_n)$ from the tangent line is achieved, the equation is written as Taylor polynomial as in Equation 1.

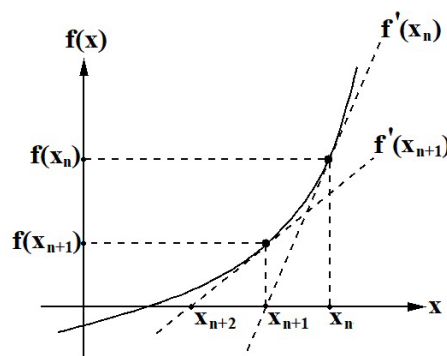$$f(x_n) + (x_{n+1} - x_n)f'(x_n) = 0 \tag{1}$$

Figure 1: Geometrical representation of the Newton's method

In Figure 1, $f'(x_n)$ can be written as $f'(x_n) = f(x_n)/(x_n - x_{n+1})$ since it means the slope of the tangent line. Thus, x_{n+1} is assigned at the intersection point between the tangent line and the x-axis and can be written as in Equation 2.

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \tag{2}$$

The sequence of x_n values is obtained consequently, and this sequence becomes the linear interpolation of $f(x)$. x_n sequence produces an iterative solution and the point where the iteration converges, gives the root of $f(x)$.

Problem Statement

Newton's method is used in many fields of engineering and science, as well as in electromagnetism. In this lab, reconstruction of dielectric constant measurement data in an RF system will be addressed by Newton's method. We will first complete the iteration for the following function with Newton's method.

$$f(x_n) = tan(x_n) - Kx_n = 0 \tag{3}$$

where

$$K = \frac{tan\left(\frac{2\pi(d+L)}{\lambda_g}\right)}{2\pi\left(\frac{d}{\lambda_g}\right)} \tag{4}$$

Remember you should iterate the function as follows.

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \tag{5}$$

Derivative of $f(x_n)$:

$$f'(x_n) = sec^2(x_n) - K \tag{6}$$

Our main aim is to find the value of dielectric constant, e_r. The value of e_r is found using the root of the function $f(x_n)$.

$$e_r = \frac{\left(\frac{a}{d}\right)^2\left(\frac{x_n}{\pi}\right)^2 + 1}{\left(\frac{2a}{\lambda_g}\right)^2 + 1} \tag{7}$$

After the iteration is completed, the value of x_n can be replaced in Equation 7 and the e_r value can then be calculated. In the above equations, some unknown parameters are determined by measurements. In this lab, we will assume that these values have been measured as follows.

$$d = 0.5,\ L = 1.4,\ \lambda_g = 3.6,\ a = 2.28 \tag{8}$$

Lab Procedure

- Define d, L, λ_g and a as global variables. You may also need to define π. (4 pts.)

- Declare f, f', K and e_r as separate functions. You must follow the prototypes below. (12 pts.)

```
double f(double x); // f(x): it should take x as an input argument
double f_prime(double x); // f'(x): it should take x as an input argument
double K_(); // since the variables are globally defined, it has no need an argument
double e_r(double x); // it needs the final form of x, not the iterative one!
```

- Your program should ask the user for the initial guess of x in main. Force the user to enter the initial guess in degrees, and the initial guess must be greater than 90 and no greater than 270. (4 pts.)

- Write a function that converts the degree to radian to properly operate with tan() function. (10 pts.)

```
void degree_to_radian(double *x); //takes degree and converts it to radian
```

It is imperative to follow the prototype above. If you fail to write this function, you can get the initial guess from the user directly in radians, roughly between 1.6 and 4.7. In this case, you will lose the score of this section.

Department of Electrical and Electronics Engineering

- The tolerance value to end the iteration must be generated in a separate function. This function should generate 6 different logarithmically increasing tolerance values from 10^{-6} to 10^{-1}, and return these values appropriately. You have been asked to write a function that returns a pointer double, considering that it should return more than one value. The arguments of the function must be left blank as in the prototype given below. (20 pts.)
  If you cannot complete this function, you can generate tolerance values with a loop in main() just to continue, provided that you forfeit the score of this section.
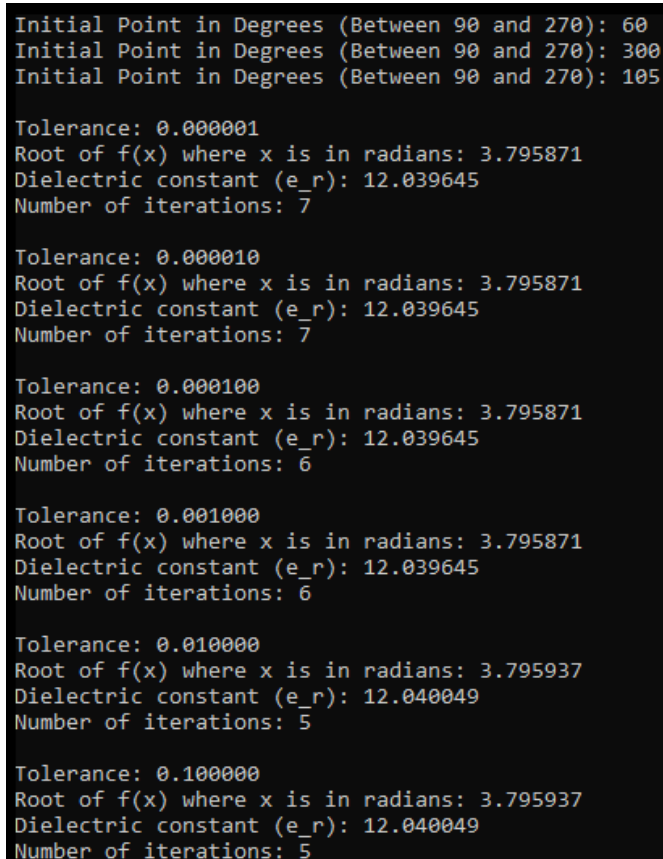
  ```
  double *tolerance_value_generator(); //returns 6 different tolerance value
  // with a single pointer: 10^-6, 10^-5, ..., 10^-1
  ```

  Hint: You may need "static double" definition for the parameter that will be returned.

- Write a function that calculates the root of f by using the Newton's method. The input arguments of the function must be the initial guess in radians and the tolerance value that is generated in the function above. It must return the root of f and the number of iterations by using a single pointer. (30 pts.)

  ```
  double *newtonsAlgorithm(double initial_guess, double tolerance); //calculate the root
  ```

- Print the root in radians, e_r and number of iterations in main() in a single loop for six different tolerance values generated by *tolerance_value_generator(). Using printf() anywhere other than main() is strictly forbidden. (20 pts.)

```
Initial Point in Degrees (Between 90 and 270): 60
Initial Point in Degrees (Between 90 and 270): 300
Initial Point in Degrees (Between 90 and 270): 105

Tolerance: 0.000001
Root of f(x) where x is in radians: 3.795871
Dielectric constant (e_r): 12.039645
Number of iterations: 7

Tolerance: 0.000010
Root of f(x) where x is in radians: 3.795871
Dielectric constant (e_r): 12.039645
Number of iterations: 7

Tolerance: 0.000100
Root of f(x) where x is in radians: 3.795871
Dielectric constant (e_r): 12.039645
Number of iterations: 6

Tolerance: 0.001000
Root of f(x) where x is in radians: 3.795871
Dielectric constant (e_r): 12.039645
Number of iterations: 6

Tolerance: 0.010000
Root of f(x) where x is in radians: 3.795937
Dielectric constant (e_r): 12.040049
Number of iterations: 5

Tolerance: 0.100000
Root of f(x) where x is in radians: 3.795937
Dielectric constant (e_r): 12.040049
Number of iterations: 5
```

Figure 2: Screenshot of the output