Ilkogretim Online - Elementary Education Online, 2021; Vol 20 (Issue 6): pp. 1551-1563
http://ilkogretim-online.org
doi: 10.17051/ilkonline.2021.06.154

Zeros Of Any Equation Of One Variable Through Numerical Methods Performed By Engineering Students

Orlando García Hurtado., Engineering Faculty, Universidad Distrital Francisco José de Caldas, Bogotá, Colombia

Oscar D. Flórez C., Engineering Faculty, Universidad Distrital Francisco José de Caldas, Bogotá, Colombia

Julián R. Camargo L., Engineering Faculty, Universidad Distrital Francisco José de Caldas, Bogotá, Colombia.

ABSTRACT-

In this work, the design and construction of an application made from an algorithm developed in Matlab to obtain the zeros of a polynomial function are presented, detailing the process of making the code, the structure of the application's graphical interface, the results and analysis, carried out by fifth-semester engineering students of a university in Bogotá Colombia. In addition, before the development of the algorithm, some basic concepts are explained to take into account when making the application, which allows understanding the objective of the application development and knowing what results were expected in addition to understanding the importance of obtaining the zeros of a Polynomial equation, from the bisection method, Newton's method, secant and false position.

Keywords: Bisection method, Newton's method, Numerical methods, Polynomial function, Zeros

I. INTRODUCTION

In the subject called Mathematical Methods of the fifth semester of the engineering faculty of a public university in Bogotá Colombia, numerical methods are developed to find approximations to solutions of equations in one variable among other topics, the students of this subject were explained the conceptual part of the topic and its mathematical development as its application through a spreadsheet, they were asked to develop an algorithm for each of the methods seen in class and its development in Matlab. The result of one of the groups is presented below.

The zeros of a polynomial equation are the roots that it has, as will be deepened later, the objective of this procedure and of calculating the zeros of the system is to find values for x that make f (x) = 0, which allows us to obtain the points of intersection of the graph with a given value of the dependent variable. In addition, it should be noted that to find the zeros of an equation, different studies have been carried out throughout history, because this is one of the oldest topics that have been studied in the area of mathematics, the importance of Finding this property is that from the roots of the equation, different values and properties of an equation can be obtained, such as maximums and minimums, proper values of matrices, solving systems of linear and differential equations, among others.

II. THEORETICAL FRAMEWORK

Zeros of an equation

The zeros of a polynomial equation refer to the roots of the graph or function, [1] which means that they are the values of "x" that make the value of the function 0. These are the points at which the graph cuts with the "x" axis, as seen in the following image, where the graph is the blue line and the roots are the black points.

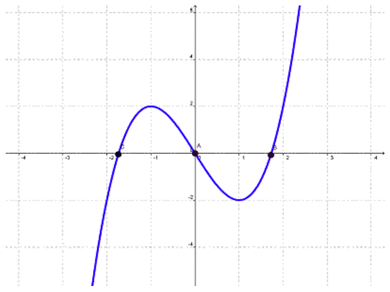


Figure 1: Example plot with roots

This can theoretically be expressed [2] as:

$$f: A \to \frac{B}{y} = f(x)$$

$$x_0 \to \text{raiz o cero} \Leftrightarrow x_0 \in A = D_f \quad y \quad f(x_0) = 0$$

Bisection method:

The bisection method [3] is one of the methods to be able to obtain the roots or zeros of an equation, which is a type of incremental search in which the interval is always divided in half.

This method is the oldest and is based on Bolzano's theorem, which says that "Be f a continuous function on a closed interval $[a, b]$ and that takes values of opposite sign at the ends, then there is at least one value $c \in (a, b)$ such that $f(c) = 0$". Therefore, the bisection method consists in that, knowing previously that the equation has at least one real root, a defined interval is reduced until it is as small as the precision required by the problem requires, if the function changes from the sign over the interval, the value of the function is evaluated at the midpoint, which is obtained with the following equation:

$$x_m = \frac{a+b}{2}$$

The position of the root is determined by placing it at the midpoint of the subinterval, within which a sign change occurs, this can be better understood thanks to the flow diagram observed in Figure 2.

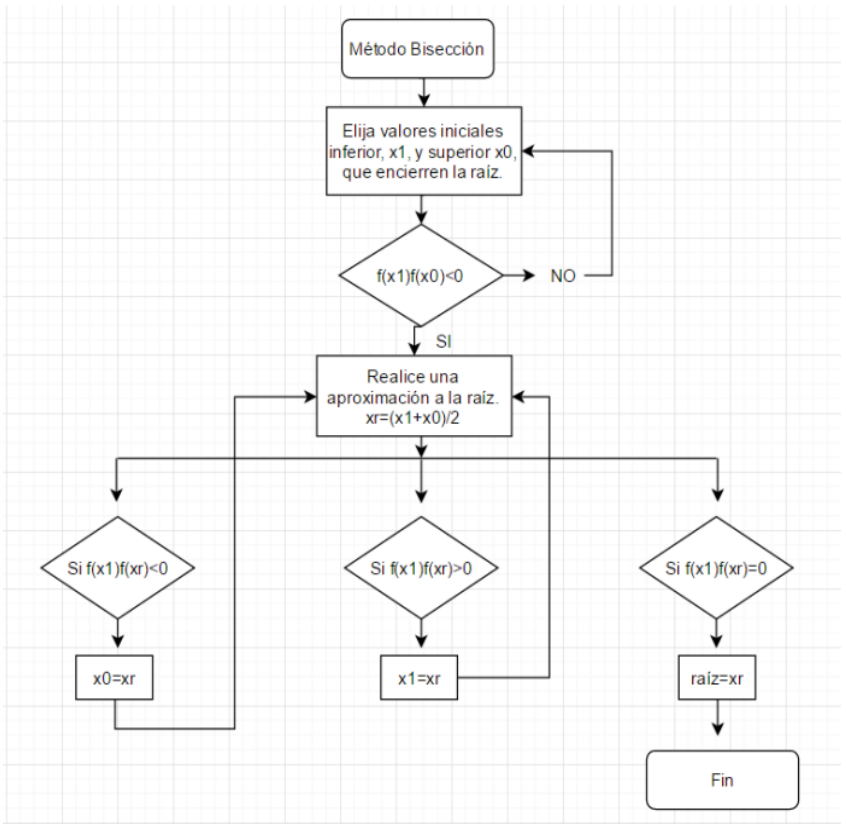

Método Bisección
Elija valores iniciales inferior, x1, y superior x0, que encierren la raíz.
f(x1)f(x0)<0
NO
SI
Realice una aproximación a la raíz. xr=(x1+x0)/2
Si f(x1)f(xr)<0
Si f(x1)f(xr)>0
Si f(x1)f(xr)=0
x0=xr
x1=xr
raíz=xr
Fin

Figure 2: Bisection method flow chart

Newton's method

Newton's numerical method [4] is one of the applications of differential calculus that is used to find the zeros of an nth-degree differentiable function with high accuracy. According to Abel's theorem, it is only possible to find approximations for the zeros of functions with a degree greater than four by applying numerical methods, it is also necessary that the functions be differentiable, to achieve continuity, everything starts from an initial approximation x0 to obtain a better approximation x1 which is defined by the following formula:$x_0 x_1$

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$

The expression can be derived from a Taylor series expansion, in which if r is a zero of f and x is an approximation to r such that r = x + h. then we have that f '' exists and is continuous, which from Taylor's theorem we would obtain:

$$0 = f(r) = f(x+h) = f(x) + hf'(x) + O(h^2)$$
$$\text{where: } h = r - x$$

If h is quite small, it would cause x to be close to r and therefore it can be stated that the term can be ignored by making 0 = f (x) + hf '(x) and we obtain the following expression for h:$O(h^2)$

$$h = -\frac{f(x)}{f'(x)}$$

In this way we can already calculate our approximation consistent with Newton's algorithm, this whole process can be better understood thanks to the flow diagram of the x_1Figure 3.

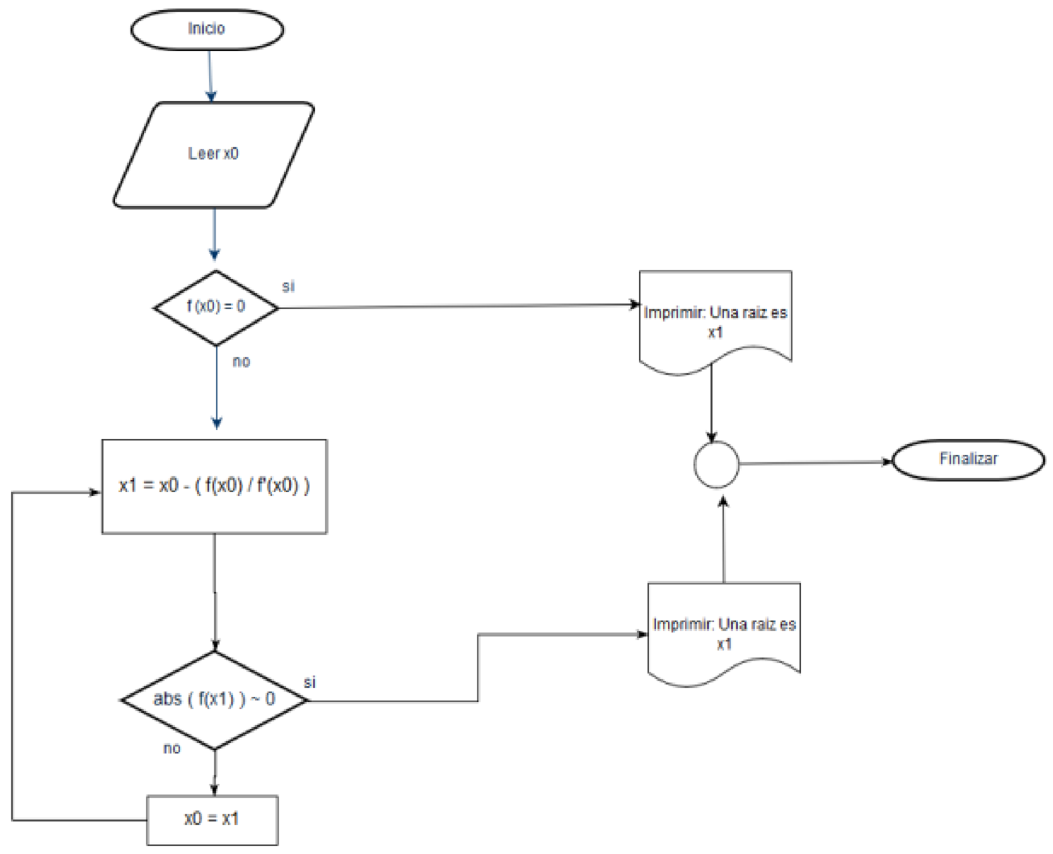


Figure 3: Newton's method flow diagram

Secant method

The secant method is another method [5] to find the zeros of a polynomial equation, which uses a series of roots of the secant lines to better approximate the root of the function studied, this method is widely used in large part because there is a drawback with Newton's method since it requires knowing the value of the first derivative of the function at the point and in certain cases, the form of the function makes it difficult to calculate its derivative, we could then affirm that it is an approximation infinite differences of the Newton-Raphson method. For these cases, it is more useful to use the secant method, which starts from two points and calculates an approximation of the slope of the line (tangent) through an approximation stipulated by the expression:

$$f'(x_1) = \frac{f(x_1) - f(x_0)}{x_1 - x_0}$$

By substituting this expression in the equation x1 of Newton's method, we obtain the expression of the secant method which allows us to know the following iteration point:

$$x_2 = x_0 - \frac{x_1 - x_0}{f(x_1) - f(x_0)} f(x_0)$$

As you can see, this method will need two initial approximations of the root to be able to induce an initial slope, this method can be better understood thanks to the flow diagram found in Figure 4.

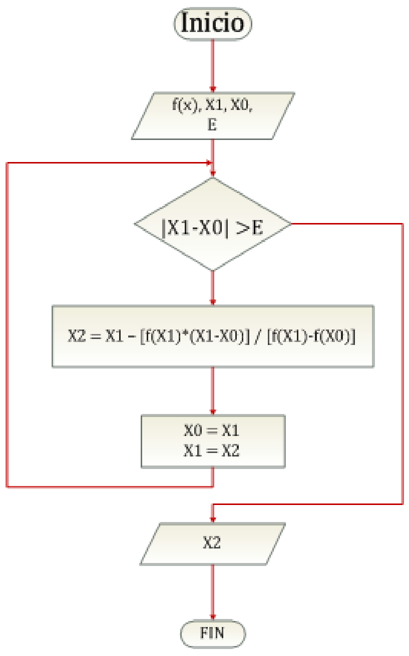
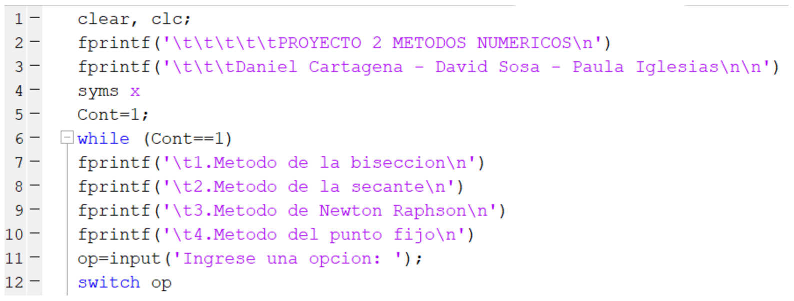


Figure 4: Secant method flow chart

III. DEVELOPMENT AND RESULTS

Algorithm - Code

To start the operation of the program to be carried out, a method selection menu was developed, through which the user by entering one of the options that appear on the screen by keyboard will be able to choose the method to use for the solution of the problem that arises. Consider, for this the developer implemented the switch method that compares the value entered by the user with the method corresponding to that value (Figure 5).

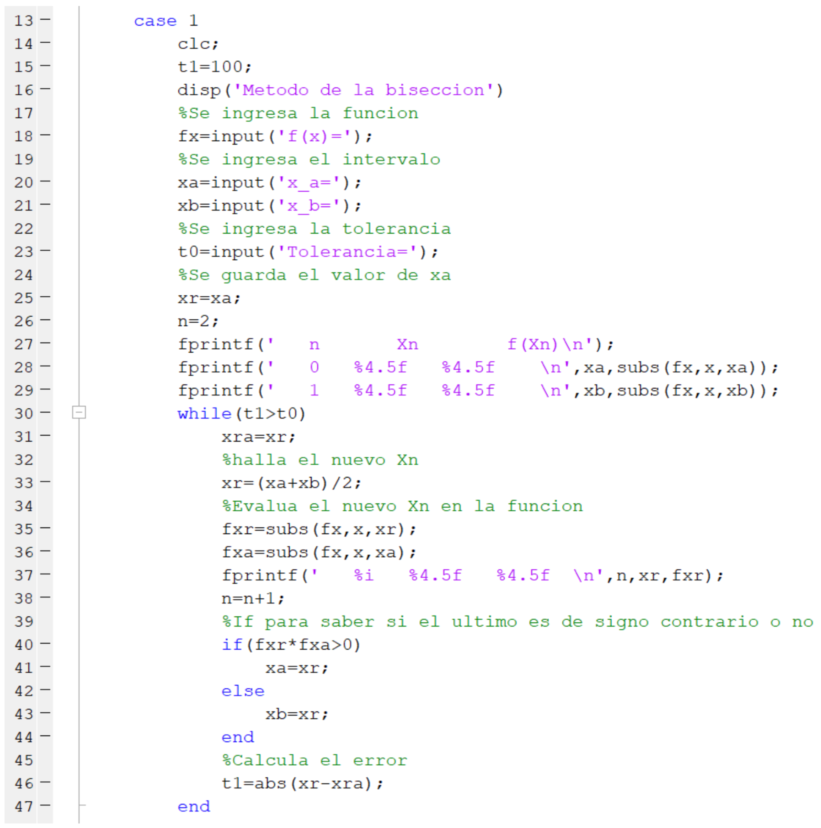


```
clear, clc;
fprintf('\t\t\t\t\tPROYECTO 2 METODOS NUMERICOS\n')
fprintf('\t\t\tDaniel Cartagena - David Sosa - Paula Iglesias\n\n')
syms x
Cont=1;
while (Cont==1)
fprintf('\t1.Metodo de la biseccion\n')
fprintf('\t2.Metodo de la secante\n')
fprintf('\t3.Metodo de Newton Raphson\n')
fprintf('\t4.Metodo del punto fijo\n')
op=input('Ingrese una opcion: ');
switch op
```

Figure 5: Method selector code.

Bisection method

The bisection method is selected in the previous part at the moment in which the user enters the numeric value of "1" by keyboard, proceeding after that to enter the function to which the roots will be found, once the function and knowing that it is being solved using the bisection method, we proceed to enter the closed interval in which said root is found, in addition to the above, the tolerance or error value is declared to later proceed with a while loop. which will be executed until the error is less than the tolerance previously defined.



```
        case 1
            clc;
            t1=100;
            disp('Metodo de la biseccion')
            %Se ingresa la funcion
            fx=input('f(x)=');
            %Se ingresa el intervalo
            xa=input('x_a=');
            xb=input('x_b=');
            %Se ingresa la tolerancia
            t0=input('Tolerancia=');
            %Se guarda el valor de xa
            xr=xa;
            n=2;
            fprintf('   n       Xn         f(Xn)\n');
            fprintf('   0    %4.5f    %4.5f     \n',xa,subs(fx,x,xa));
            fprintf('   1    %4.5f    %4.5f     \n',xb,subs(fx,x,xb));
            while(t1>t0)
                xra=xr;
                %halla el nuevo Xn
                xr=(xa+xb)/2;
                %Evalua el nuevo Xn en la funcion
                fxr=subs(fx,x,xr);
                fxa=subs(fx,x,xa);
                fprintf('    %i    %4.5f    %4.5f   \n',n,xr,fxr);
                n=n+1;
                %If para saber si el ultimo es de signo contrario o no
                if(fxr*fxa>0)
                    xa=xr;
                else
                    xb=xr;
                end
                %Calcula el error
                t1=abs(xr-xra);
            end
```

Figure 6: Code for the realization of the bisection method.

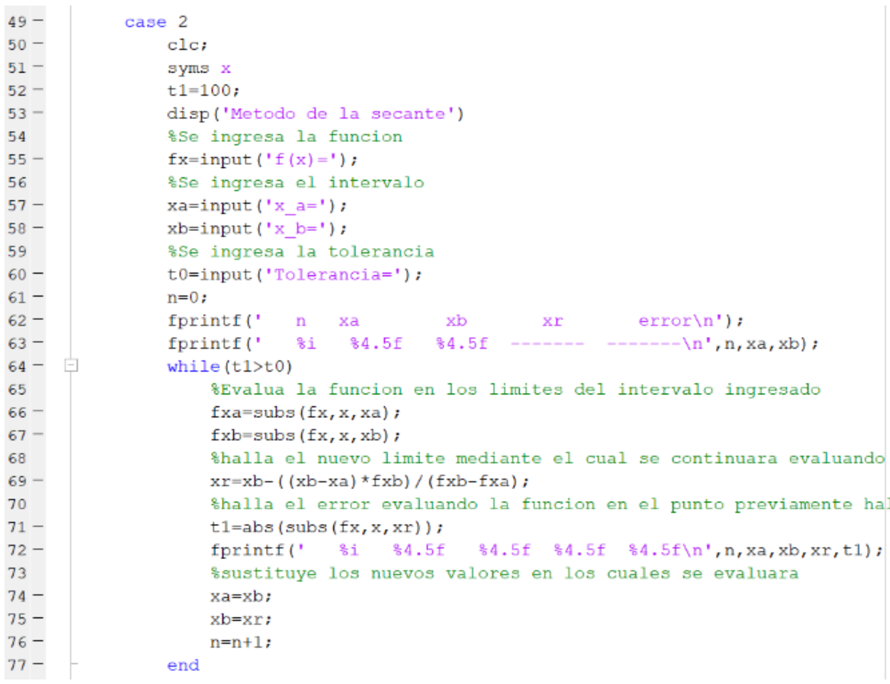
Within the while loop, the program starts by finding the midpoint of the interval that was previously defined and subsequently evaluating the function at the said midpoint, in addition to evaluating the function at the lower limit of the closed interval, once this step is carried out, it proceeds to evaluate by mean of an if the sign of the value of the function at the midpoint is the same as the function evaluated in the previous point if the above is true, the mean value found will replace the value of the variable that initially stores the lower limit of the interval, and if the condition is not fulfilled and the signs are different, the value of the midpoint will replace the value of the variable that initially stored the upper limit of the interval.

Secant method

The second code that can be selected by the user when entering option "2" by keyboard is the secant method, where initially the user is asked to enter the function from which they want to extract the roots, and Later, the interval in which said root is found is entered, in addition to entering the tolerance that will serve to define how accurately the value of the root is found (Figure 7).

Once the parameters have been defined, the program proceeds to enter a while loop to proceed with the solution of the problem posed by the user, for this first the function at the ends of the interval previously entered by the user is evaluated, to later find the next point of iteration through the equation defined above during the explanation of the said method.

After finding the next iteration point, we proceed to save this value in the variable that saved the value of the upper limit of the interval and the value of this will go to the variable that saved the value of the lower limit, to proceed with the subsequent iterations until the error is reduced to a value less than the tolerance defined above.



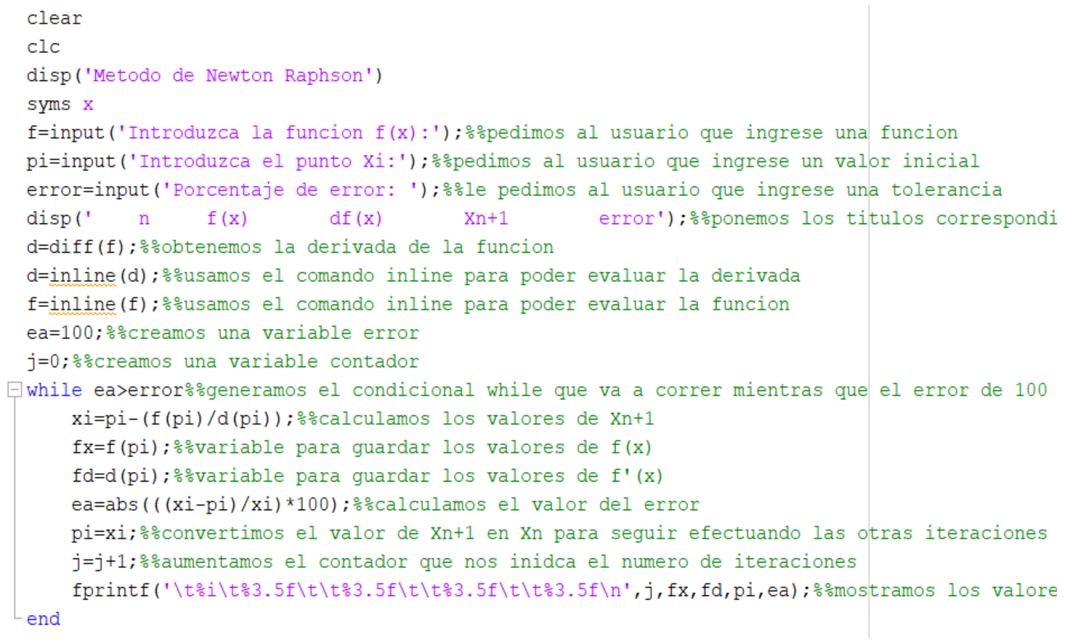
```
    case 2
        clc;
        syms x
        t1=100;
        disp('Metodo de la secante')
        %Se ingresa la funcion
        fx=input('f(x)=');
        %Se ingresa el intervalo
        xa=input('x_a=');
        xb=input('x_b=');
        %Se ingresa la tolerancia
        t0=input('Tolerancia=');
        n=0;
        fprintf('   n   xa        xb        xr        error\n');
        fprintf('   %i   %4.5f   %4.5f  -------  -------\n',n,xa,xb);
        while(t1>t0)
            %Evalua la funcion en los limites del intervalo ingresado
            fxa=subs(fx,x,xa);
            fxb=subs(fx,x,xb);
            %halla el nuevo limite mediante el cual se continuara evaluando
            xr=xb-((xb-xa)*fxb)/(fxb-fxa);
            %halla el error evaluando la funcion en el punto previamente hallado
            t1=abs(subs(fx,x,xr));
            fprintf('   %i   %4.5f   %4.5f  %4.5f  %4.5f\n',n,xa,xb,xr,t1);
            %sustituye los nuevos valores en los cuales se evaluara
            xa=xb;
            xb=xr;
            n=n+1;
        end
```

Figure 7: Code for the realization of the secant method.

Newton Raphson method

For Newton's Method, the first thing we do is clean the console, then we place the title corresponding to the method, then we indicate that the variable "x" is of a symbolic type, this to execute the function correctly, then we create the variable " f ”that will be in charge of storing the function entered by the user, then a variable“ pi ”corresponding to the initial value given by the user and finally an“ error ”variable that will store the value of the tolerance, given value by the user in the same way.

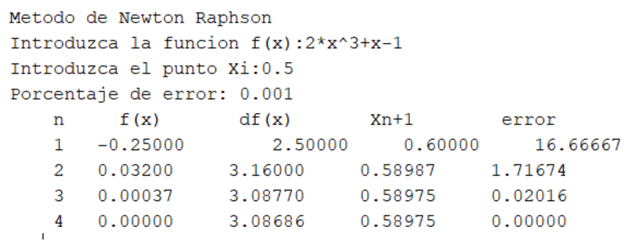
Then we generate the respective names of our columns of the final results table, then we assign to the variable "d" the value of the derivative of the function "f", then we use the inline command in the variables "d" and "f" To evaluate each equation with the corresponding values, then we create the variable "ea" that allows us to have a reference value of the error of 100, and we create a variable "j" that will be a counter (Figure 8).



```
clear
clc
disp('Metodo de Newton Raphson')
syms x
f=input('Introduzca la funcion f(x):');%%pedimos al usuario que ingrese una funcion
pi=input('Introduzca el punto Xi:');%%pedimos al usuario que ingrese un valor inicial
error=input('Porcentaje de error: ');%%le pedimos al usuario que ingrese una tolerancia
disp('    n     f(x)       df(x)       Xn+1        error');%%ponemos los titulos correspondi
d=diff(f);%%obtenemos la derivada de la funcion
d=inline(d);%%usamos el comando inline para poder evaluar la derivada
f=inline(f);%%usamos el comando inline para poder evaluar la funcion
ea=100;%%creamos una variable error
j=0;%%creamos una variable contador
while ea>error%%generamos el condicional while que va a correr mientras que el error de 100
    xi=pi-(f(pi)/d(pi));%%calculamos los valores de Xn+1
    fx=f(pi);%%variable para guardar los valores de f(x)
    fd=d(pi);%%variable para guardar los valores de f'(x)
    ea=abs(((xi-pi)/xi)*100);%%calculamos el valor del error
    pi=xi;%%convertimos el valor de Xn+1 en Xn para seguir efectuando las otras iteraciones
    j=j+1;%%aumentamos el contador que nos inidca el numero de iteraciones
    fprintf('\t%i\t%3.5f\t\t%3.5f\t\t%3.5f\t\t%3.5f\n',j,fx,fd,pi,ea);%%mostramos los valore
end
```

Figure 8: Matlab code newton's method

Now we create a while loop which will be executed while the variable "ea" is greater than the tolerance given by the user, then we calculate the values of Xn + 1, which are stored in the variable "xi", then we go to have the variables "fx" and "fd", which will store the value of the function and the derivative of the function respectively with the corresponding Xn values in each iteration, then we proceed to calculate the error of the obtained values in each iteration and is stored in the variable "ea", then we assign the value of "xi" to the variable "pi", this to convert the value of Xn + 1 into the value of Xn of the next iteration for To be able to continue with the data calculation, we increase the counter variable "j", and we show the values of the iteration number, function, derivative, Xn + 1 and the error of each cycle.



```
Metodo de Newton Raphson
Introduzca la funcion f(x):2*x^3+x-1
Introduzca el punto Xi:0.5
Porcentaje de error: 0.001
    n     f(x)       df(x)       Xn+1        error
    1   -0.25000       2.50000      0.60000      16.66667
    2   0.03200      3.16000      0.58987      1.71674
    3   0.00037      3.08770      0.58975      0.02016
    4   0.00000      3.08686      0.58975      0.00000
```

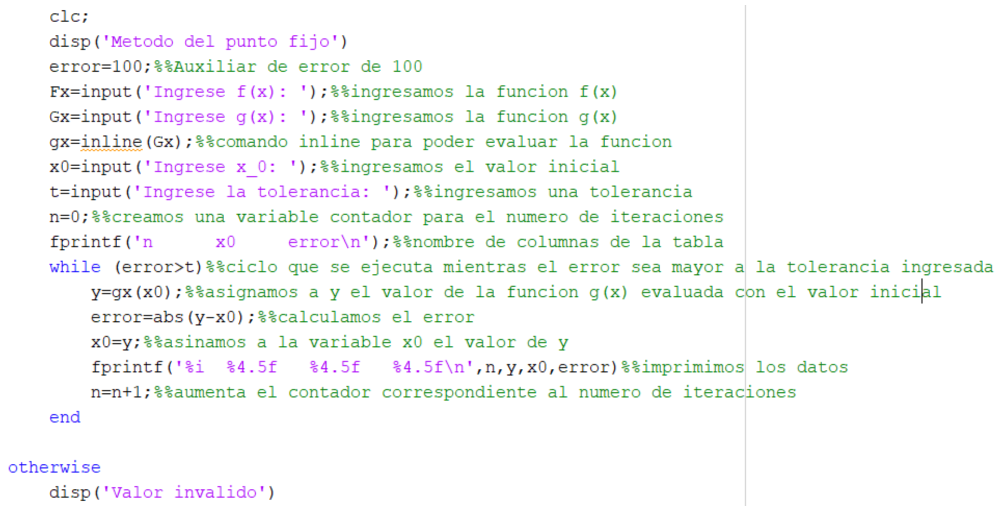
Figure 9: Newton method code simulation and operation

Now we proceed to simulate said program, for which we digitize the function with an initial value of 0.5 and a tolerance of 0.001, and we run the code and, as we can see, four iterations were carried out and our function takes a value of 0 when the value of x is 0.58975.$2x^3 + x - 1$

Fixed Point Method

For the fixed point method, we proceed to create an "error" variable with a value of 100 that will serve as an auxiliary, then we assign to "Fx" the value of the function f (x) entered by the user, then we assign to "Gx" the value of the function g (x) given by the user, we use the inline command to be able to evaluate the function correctly, we create a variable "x0", which in the first instance will correspond to the initial value given by the user, then we have a variable "t" that will indicate the value of the tolerance, then a variable "n" which is a counter and finally we print the names of the column of our table.

We continue with a while loop which will be executed as long as the value of "error" is greater than the tolerance entered by the user, followed by that we assign to the variable and the value of the function g (x) evaluated with the value of Initial x0, value given by the user for the first iteration, then we calculate the error of the data obtained, followed by this we assign a new value to “x0”, which will be that of “y”, previously calculated, and thus have the replacement data for the next iteration, until we find the value of x that complies with the rule of f (x) = 0, we print the values obtained and we increase the counter variable "n" corresponding to the iterations (Figure 10).

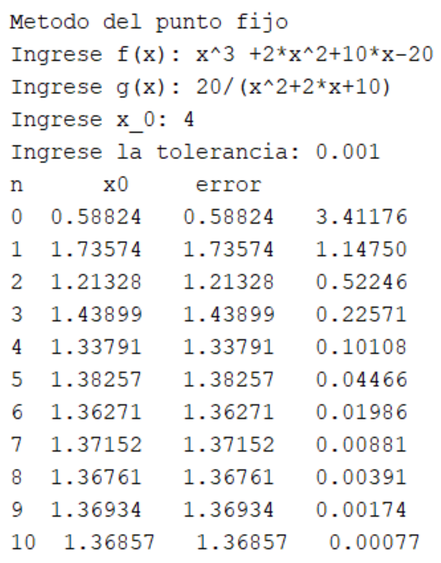


```
    clc;
    disp('Metodo del punto fijo')
    error=100;%%Auxiliar de error de 100
    Fx=input('Ingrese f(x): ');%%ingresamos la funcion f(x)
    Gx=input('Ingrese g(x): ');%%ingresamos la funcion g(x)
    gx=inline(Gx);%%comando inline para poder evaluar la funcion
    x0=input('Ingrese x_0: ');%%ingresamos el valor inicial
    t=input('Ingrese la tolerancia: ');%%ingresamos una tolerancia
    n=0;%%creamos una variable contador para el numero de iteraciones
    fprintf('n      x0     error\n');%%nombre de columnas de la tabla
    while (error>t)%%ciclo que se ejecuta mientras el error sea mayor a la tolerancia ingresada
        y=gx(x0);%%asignamos a y el valor de la funcion g(x) evaluada con el valor inicial
        error=abs(y-x0);%%calculamos el error
        x0=y;%%asinamos a la variable x0 el valor de y
        fprintf('%i  %4.5f   %4.5f   %4.5f\n',n,y,x0,error)%%imprimimos los datos
        n=n+1;%%aumenta el contador correspondiente al numero de iteraciones
    end

otherwise
    disp('Valor invalido')
```

Figure 10: Matlab code fixed point method

At the end of the code, we have that if a value is entered outside the logic of the program, an error message with the phrase "invalid value" will be indicated.



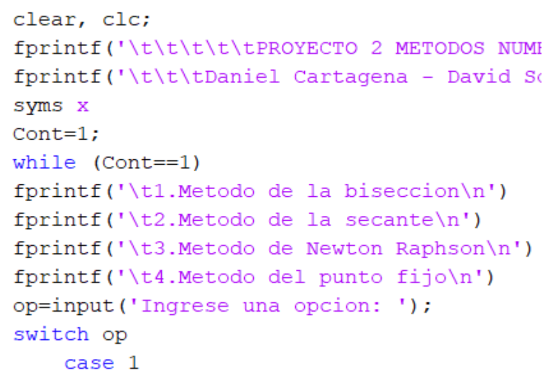
```
Metodo del punto fijo
Ingrese f(x): x^3 +2*x^2+10*x-20
Ingrese g(x): 20/(x^2+2*x+10)
Ingrese x_0: 4
Ingrese la tolerancia: 0.001
n      x0      error
0  0.58824   0.58824   3.41176
1  1.73574   1.73574   1.14750
2  1.21328   1.21328   0.52246
3  1.43899   1.43899   0.22571
4  1.33791   1.33791   0.10108
5  1.38257   1.38257   0.04466
6  1.36271   1.36271   0.01986
7  1.37152   1.37152   0.00881
8  1.36761   1.36761   0.00391
9  1.36934   1.36934   0.00174
10  1.36857   1.36857   0.00077
```

Figure11: Fixed point method code simulation and operation

Now we proceed to simulate said program with a function, a function, an initial value x0 of 4 and a tolerance of 0.001, as we can see in the previous image, we see that our result as a value of x is 1.36857.$f(x) = x^3 + 2x^2 + 10x - 20g(x) = \frac{20}{x^2+2x+10}$

Functioning

For the operation of the program, we proceed to use the Matlab console and through a menu, we are going to ask the user to choose the method to use to solve the exercise, depending on the option that he chooses, the different methods previously explained will be executed, this menu is develop using the switch command and the use of the respective case.



```
clear, clc;
fprintf('\t\t\t\t\tPROYECTO 2 METODOS NUM
fprintf('\t\t\tDaniel Cartagena - David S
syms x
Cont=1;
while (Cont==1)
fprintf('\t1.Metodo de la biseccion\n')
fprintf('\t2.Metodo de la secante\n')
fprintf('\t3.Metodo de Newton Raphson\n')
fprintf('\t4.Metodo del punto fijo\n')
op=input('Ingrese una opcion: ');
switch op
    case 1
```

Figure12: Simulation and operation

IV. CONCLUSIONS

To prevent Matlab from collapsing when calculating the zero of a polynomial, the number of iterations was limited to 70, if this number is exceeded, a message is launched warning that the zero is not in the specified range

The methods used are very useful when finding the roots of equations in this polynomial program, more equally applicable with any other type of function.

Open methods are more useful if the possible interval in which the root of the equation is found is not known, since when starting from a specific point, the iterations lead to the initial point at the zero of the equation, as long as the equation exists. root.

The projects that are proposed in class generate in the studentsgreater interest in the subject.

REFERENCES

1. Villanueva, W. D. (s.f.). 4. Cálculo de raíces de una ecuación. [Online]: Retrieved on 15-June-2021, at URL:https://www.uv.es/~diaz/mn/node17.html
2. Teorema de Bolzano y raíces de una función. [Online]: Retrieved on 22-June-2021, at URL: https://www.superprof.es/apuntes/escolar/matematicas/calculo/derivadas/teorema-de-bolzano.html.
3. Método de la bisección. [Online]: Retrieved on 12-July-2021, at URL: https://cs.famaf.unc.edu.ar/~hoffmann/mn/practico03.html
4. NewtonRaphson. [Online]:Retrieved on 25-Aug-2021, at URL:http://test.cua.uam.mx/MN/Methods/Raices/NewtonRaphson/NewtonRaphson.php
5. Método de la Secante. [Online]: Retrieved on 11-Sept-2021, at URL: https://sites.google.com/site/cm4312014/sis/metodo-de-la-secante