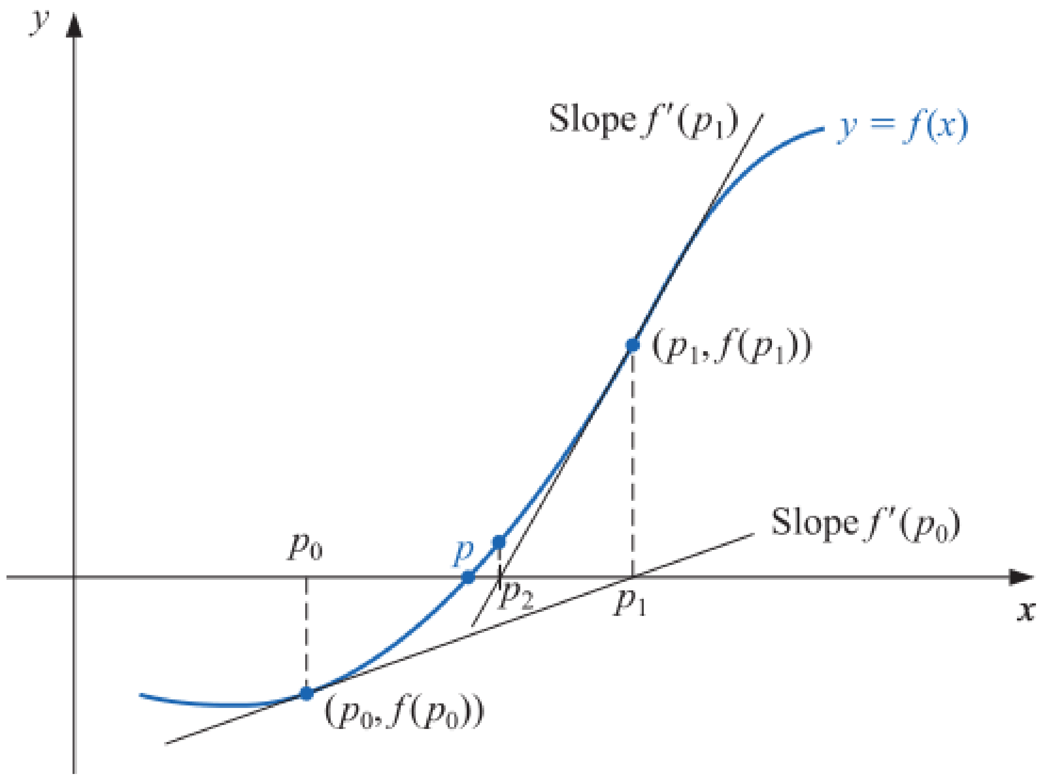
Computational Methods EP-208

APPLICATIONS & ROOTS USING NUMERICAL METHODS



INNOVATIVE MID-TERM PROJECT

BY **ADITYA SINGH** 2K19/EP/005

AND **ANSHUL SATIJA** 2K19/EP/018

Numerical Methods

In numerical analysis, a numerical method is a mathematical tool designed to solve numerical problems.
Numerical analysis is the area of mathematics and computer science that creates, analyzes, and implements algorithms for solving numerically the problems of continuous mathematics.

These problems occur throughout the natural sciences, social sciences, medicine, engineering, and business.
The roots of a function are the *x*-intercepts. By definition, the Y-coordinate of points lying on the *x*-axis is zero.

Various Numericals Methods for finding roots:

- Bisection Method
- Regula Falsi Method
- Secant Method
- Newton Raphson Method

Bisection Method

If a function f(x) is continuous on the interval [a..b] and sign of f(a) ≠ sign of f(b),

There is a value *c* ∈ [*a*..*b*] such that: *f(c) = 0.*

i.e there is a root c in the interval [a,b].

Steps :

Find two points, say a and b such that a < b and f(a)* f(b) < 0

Find the midpoint of a and b, say "c".

"c" is the root of the given function if f(c) = 0; else follow the next step. Divide the interval [a, b]

If f(c)*f(b) <0, let a = c.

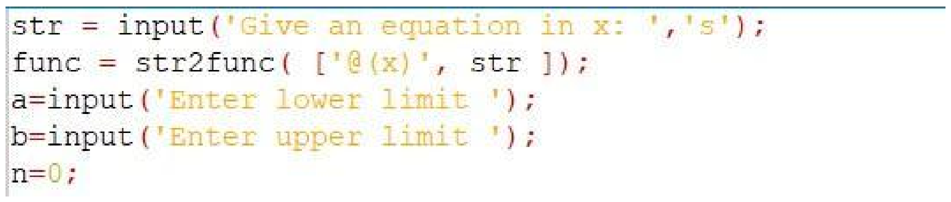
Else if f(c) *f(a) > 0 , let b = c. Repeat above three steps until f(c) = 0.

Facts about Bisection Method

- The method always converges.
- Error while calculating roots can be controlled.
- Error bound decreases by ½ with each iteration.
- The calculations are very simple.
- The convergence rate is relatively slower and thus making the bisection method a slow method to calculate the roots.
- The bisection method fails when roots are complex or in the condition of double roots.

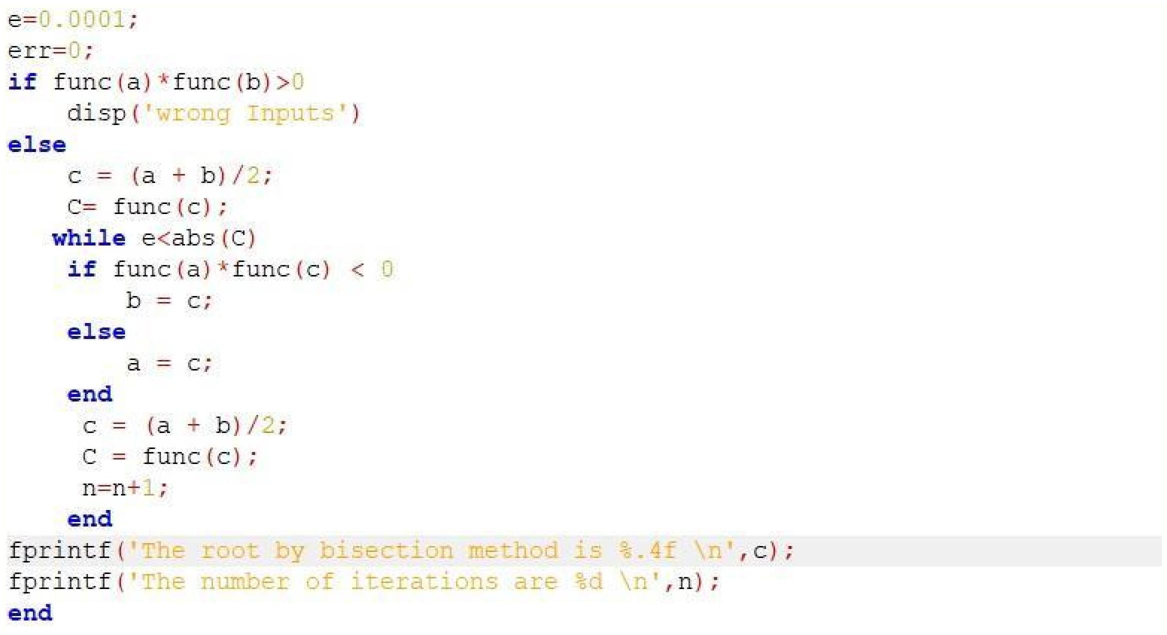
Simulation for Bisection Method

The first part of the code deals with taking input of function and all the necessary values for calculating the root.



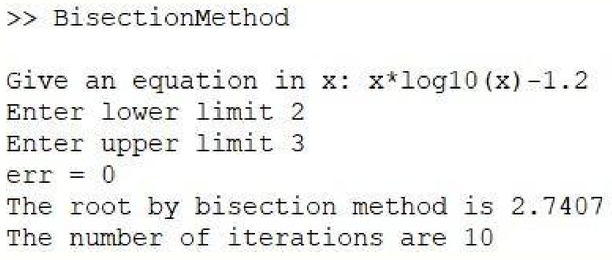
```
str = input('Give an equation in x: ','s');
func = str2func( ['@(x)', str ]);
a=input('Enter lower limit ');
b=input('Enter upper limit ');
n=0;
```

The next step is to define an initial value of tolerance and check all the conditions and run the loop until a particular value of tolerance is reached



```
e=0.0001;
err=0;
if func(a)*func(b)>0
    disp('wrong Inputs')
else
    c = (a + b)/2;
    C= func(c);
   while e<abs(C)
    if func(a)*func(c) < 0
        b = c;
    else
        a = c;
    end
     c = (a + b)/2;
     C = func(c);
     n=n+1;
    end
fprintf('The root by bisection method is %.4f \n',c);
fprintf('The number of iterations are %d \n',n);
end
```

Output



```
>> BisectionMethod

Give an equation in x: x*log10(x)-1.2
Enter lower limit 2
Enter upper limit 3
err = 0
The root by bisection method is 2.7407
The number of iterations are 10
```

Regula-Falsi Method

The Regula–Falsi Method is a numerical method for estimating the roots of a polynomial f(x). A value x replaces the midpoint in the Bisection Method and serves as the new approximation of a root of f(x). It was developed because the bisection method converges at a fairly slow speed. It depends only on the choice of endpoints of the interval [a,b]. The function f(x) does not have any role in finding the point c (which is just the midpoint of a and b).

Steps :

1. Find points a and b such that $a < b$ and $f(a) * f(b) < 0$.
2. Take the interval [a, b] and determine the next value of x, by

$$x = b - \frac{f(b) * (b-a)}{f(b) - f(a)}$$

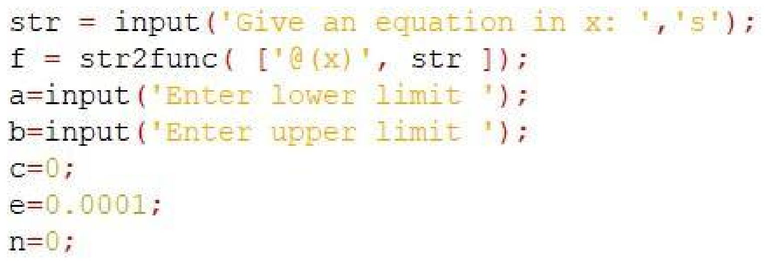
3. If $f(x) = 0$ then x is an exact root, else if $f(x) * f(b) < 0$ then let $b = x$, else if $f(a) * f(x) < 0$ then let $a = x$.
4. Repeat steps 2 & 3 until $f(x) = 0$.

Facts about Regula-Falsi Method

- This method also always converges.
- As the method works on the principle of hit and trial, the method slows down in relatively unfavorable conditions.
- The method does not involve any complex calculations.

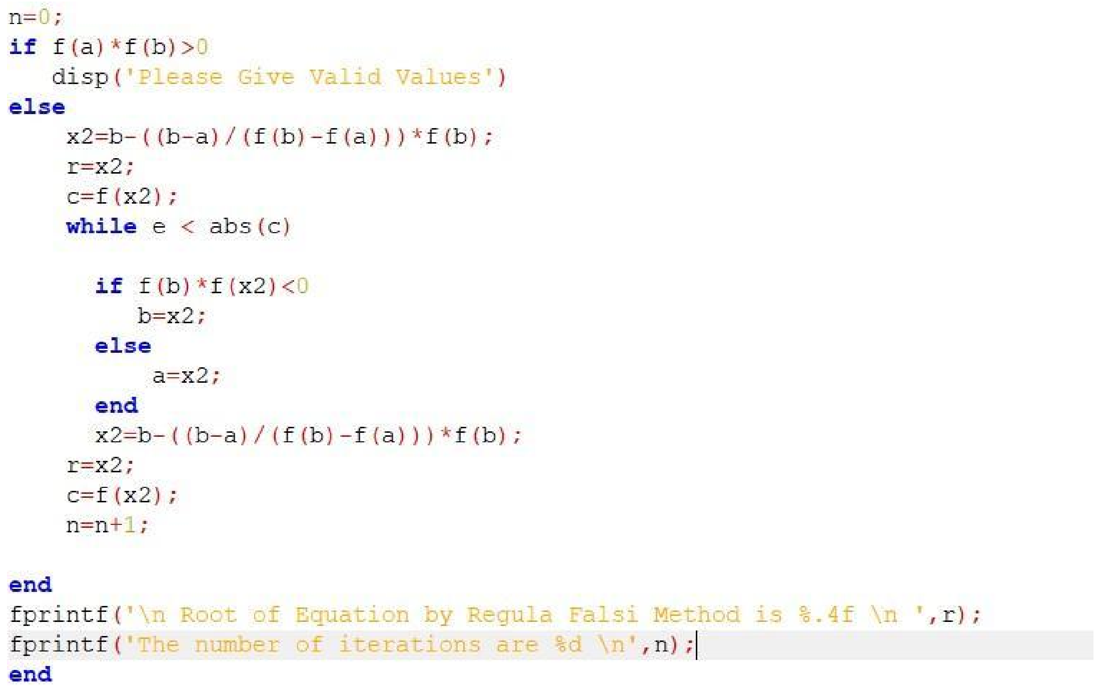
Simulation for Regula-Falsi Method

The first part of the code deals with taking input of function and all the necessary values for calculating the root.



```
str = input('Give an equation in x: ','s');
f = str2func( ['@(x)', str ]);
a=input('Enter lower limit ');
b=input('Enter upper limit ');
c=0;
e=0.0001;
n=0;
```

The next step is to define an initial value of tolerance and check all the conditions and run the loop of the calculations until a particular value of tolerance is reached

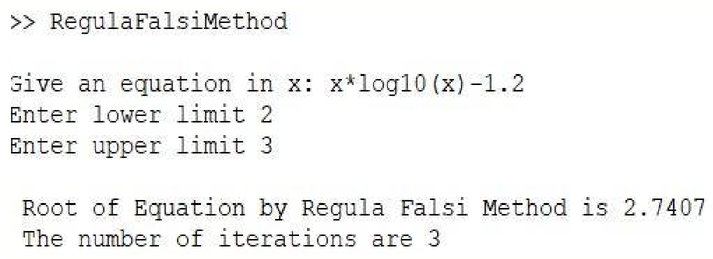


```
n=0;
if f(a)*f(b)>0
   disp('Please Give Valid Values')
else
    x2=b-((b-a)/(f(b)-f(a)))*f(b);
    r=x2;
    c=f(x2);
    while e < abs(c)

      if f(b)*f(x2)<0
         b=x2;
      else
          a=x2;
      end
      x2=b-((b-a)/(f(b)-f(a)))*f(b);
    r=x2;
    c=f(x2);
    n=n+1;

end
fprintf('\n Root of Equation by Regula Falsi Method is %.4f \n ',r);
fprintf('The number of iterations are %d \n',n);
end
```

Output



```
>> RegulaFalsiMethod

Give an equation in x: x*log10(x)-1.2
Enter lower limit 2
Enter upper limit 3

 Root of Equation by Regula Falsi Method is 2.7407
 The number of iterations are 3
```

Secant Method

The secant method is very similar to the bisection method except instead of dividing each interval by choosing the midpoint the secant method divides each interval by the secant line connecting the endpoints. The secant method can be thought of as a finite-difference approximation of Newton's method.

Steps :

1. Find points a and b such that a < b.
2. Take the interval [a, b] and determine the next value of x, by

$$x = b - \frac{f(b) * (b-a)}{f(b) - f(a)}$$

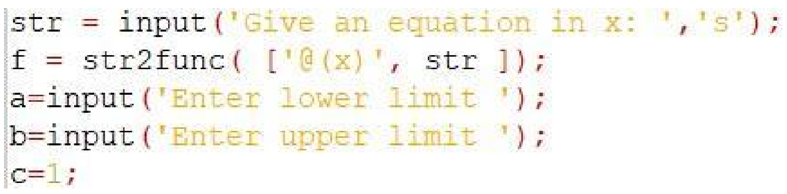
3. If f(x) = 0 then x is an exact root, if f(b)*f(x) <0 let a=b and b = x else b= a and a= x.
4. Repeat steps 2 & 3 until f(x) = 0.

<u>Facts about Secant Method</u>

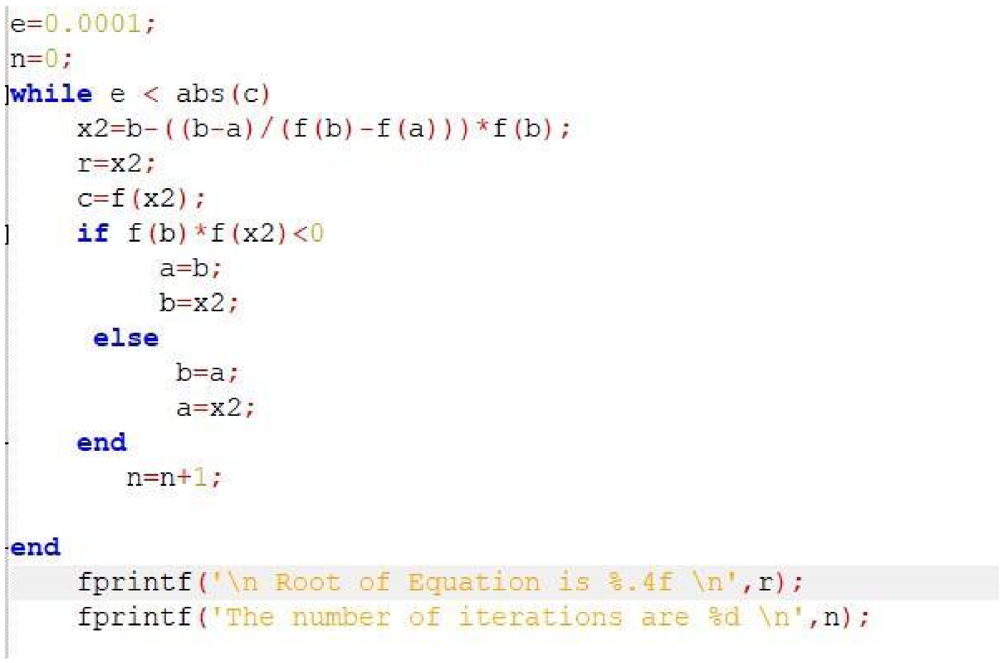
- The convergence rate is 1.618.
- The method may or may not converge.

Simulation for Secant Method

The first part of the code deals with taking input of function and all the necessary values for calculating the root.

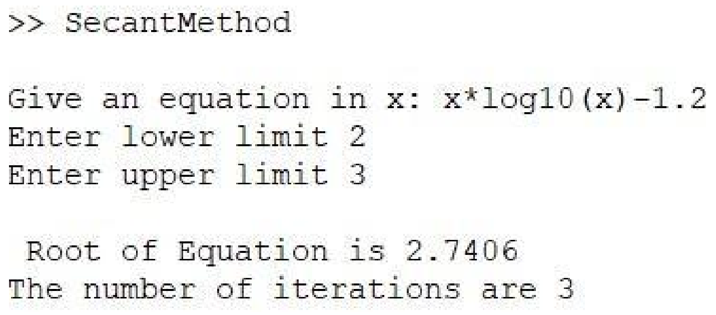

```
str = input('Give an equation in x: ','s');
f = str2func( ['@(x)', str ]);
a=input('Enter lower limit ');
b=input('Enter upper limit ');
c=1;
```

The next step is to define an initial value of tolerance and check all the conditions and run the loop of the calculations until a particular value of tolerance is reached.


```
e=0.0001;
n=0;
while e < abs(c)
    x2=b-((b-a)/(f(b)-f(a)))*f(b);
    r=x2;
    c=f(x2);
    if f(b)*f(x2)<0
         a=b;
         b=x2;
     else
          b=a;
          a=x2;
    end
       n=n+1;

end
    fprintf('\n Root of Equation is %.4f \n',r);
    fprintf('The number of iterations are %d \n',n);
```

Output


```
>> SecantMethod

Give an equation in x: x*log10(x)-1.2
Enter lower limit 2
Enter upper limit 3

 Root of Equation is 2.7406
The number of iterations are 3
```

Newton Raphson Method

The Newton-Raphson method is one of the most widely used methods for root finding. The technique is quadratically convergent as we approach the root. The technique requires only one initial value x_0, which we will refer to as the initial guess for the root.

Steps :

1) Calculate the derivative of the function.
2) Simply put values in the formula

$$X_{n+1} = X_n + f(x_n) / f`(x_n)$$

Facts:

- This is the fastest converging method.
- The rate of convergence is 2.
- The method may or may not always converge.
- The method involves the calculating values of two functions for every iteration, the process might be long and time-consuming.

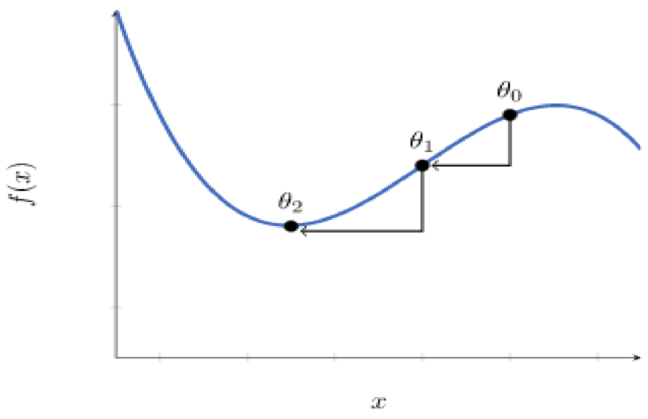
GRADIENT DESCENT

Gradient Descent is an optimization algorithm used to find a local minimum of a given function. It's widely used within machine learning algorithms to minimize loss functions.

We start with a continuous function that we want to minimize, and that's differentiable in the interval of our interest.

Then, we identify a starting point that is sufficient proximity to the function's local minimum, in this case, θ_0

Then, iteratively, we move towards the closest local minimum by exploiting the gradient of the function around θ_2.



Newton's Method works in a different manner. This is because it's a method for finding the root of a function, rather than its maxima or minima.

We, therefore, apply Newton's method on the derivative of the cost function, not on the cost function itself. This means that the cost function we use must be differentiable twice, not just once, as was the case for gradient descent.

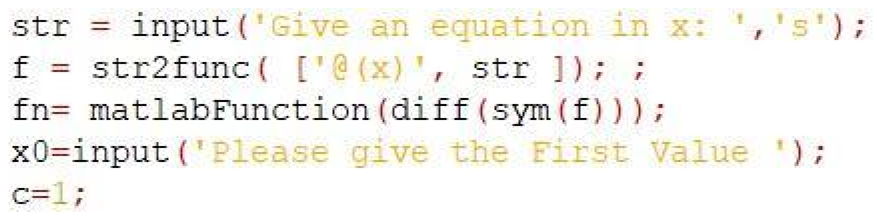
If we start from a point x_n that's sufficiently close to the minimum of f, we can then get a better approximation by computing this formula:

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

$\frac{f'(x)}{f''(x)}$ here, indicates that we're approximating the function f'(x) with a linear model, in the proximity of x_n.

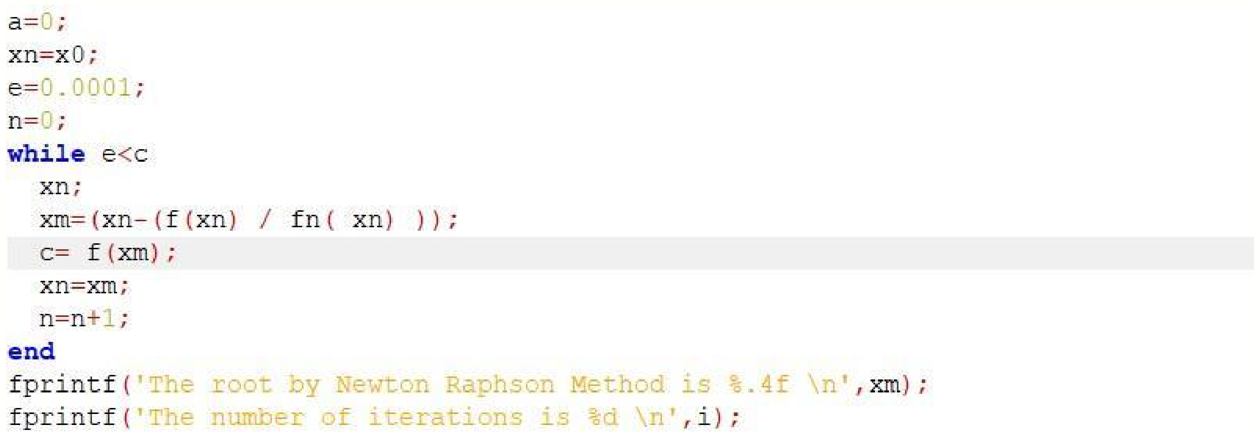
Simulation of Newton-Raphson Method

The first part of the code deals with taking input of function and all the necessary values for calculating the root.



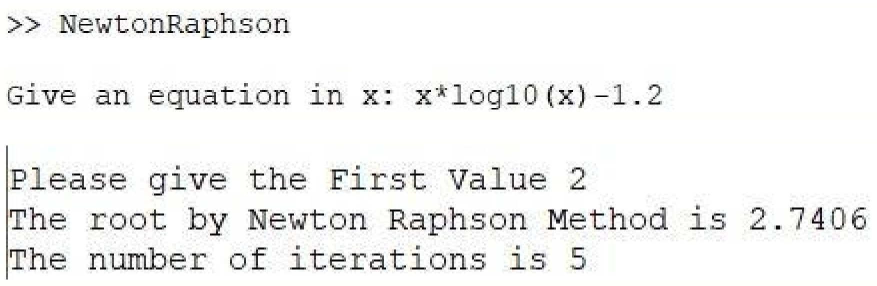
```
str = input('Give an equation in x: ','s');
f = str2func( ['@(x)', str ]); ;
fn= matlabFunction(diff(sym(f)));
x0=input('Please give the First Value ');
c=1;
```

The next step is to define an initial value of tolerance and check all the conditions and run the loop of the calculations until a particular value of tolerance is reached.



```
a=0;
xn=x0;
e=0.0001;
n=0;
while e<c
  xn;
  xm=(xn-(f(xn) / fn( xn) ));
  c= f(xm);
  xn=xm;
  n=n+1;
end
fprintf('The root by Newton Raphson Method is %.4f \n',xm);
fprintf('The number of iterations is %d \n',i);
```

Output



```
>> NewtonRaphson

Give an equation in x: x*log10(x)-1.2

Please give the First Value 2
The root by Newton Raphson Method is 2.7406
The number of iterations is 5
```

A Brief Comparison of all the methods

	Bisection Method	Regula-Falsi Method	Secant Method	Newton-Raphson Method
Always Converging	Always converges	Always converges	Not always converges	May or May not Converges
Order of Convergence	1	1	1.618	2
Method Speed	Slowest	Faster than Bisection Method	Faster than Secant Method	Fastest Method

The bisection method is slow, but has no limitations and will always get you to the same answer eventually.

Bisection Method guarantees convergence Newton Raphson technique has a quadratic rate of convergence whereas the Bisection method has a linear rate of Convergence.

NR is more Complicated than the Bisection method. And also the Bisection method is definite to give a result where Newton Raphson fails for many cases.

Even despite these shortcomings, it is the speed of convergence for which we prefer the Newton Raphson algorithm. Another problem with Newton's method is instability.

Some more applications and Conclusions

The bisection method is used for determining the adequate population size.
The Regula-Falsi method is used in the prediction of quantities of air pollutants produced by combustion reaction.
Mathematics is an aspect of life needed in every field in some or another way and is used in subjects dealing with all the practical applications. Calculating everything with pen and paper is difficult in practical applications.
Even the slightest of error in calculations can lead to big problems. The application of simulations can ease this burden a lot. Right now in this program, it may not seem so but when the values are very large and calculations become hefty then simulations can provide results faster and chances of any error are very less.

Even the slightest of error in calculations can lead to a big disaster which can be life-threatening. The application of simulations can ease this burden a lot. Right now in the program, it may not seem so but when the values are very large and calculations become hefty then simulations can provide results faster and chances of any error are very less.

END