Root of the Equations

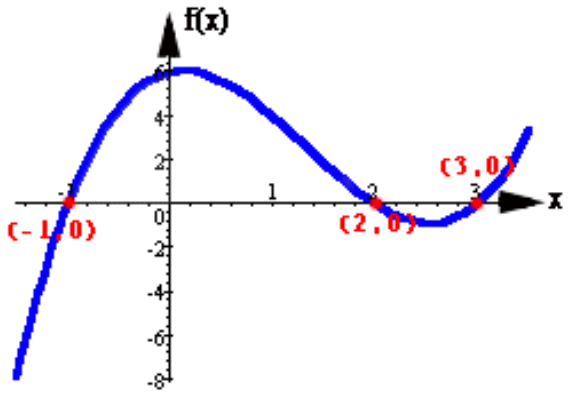
Introduction

Algebraic and Transcendental equations and their solution by numerical methods.

*Algebaric Equation*s: if *f(x)* is polynomial in *x* of degree *n >= 1* , then *f(x)=0* is called **Algebraic Equations**.
Transcendental Equations: if *f(x)* is a function of *x* in trignometric ,exponential and log expression then *f(x)=0* is called **Transcendental Equations**.
eg: $x^3 - 3x^2 + 2x + 4 = 0$
$x^3 - sin(x) + e^x = 0$

- *what is root?*

If *f(a)=0*, then a called root of the equation.
Geometrically, a root of equation *f(x)=0* is that value of x for which the graph of *y=f(x)* cross the *x-axis*.



- why numerical methods?

If *f(x)=0*, is quadratic, cubic, biquadratic then the algebraic method of solution are available but a higher degree of equations of transcendental for which no general method exists so such equations are solved by numerical method approximately.

Note: Almost all numerical methods of solution of an equation require an initial rough approximation to its root and then cyclically comes to a better approximation.

Intermediate value theorem of differential calculus which says that
if f(x) is continuous in [a,b] and f(a) and f(b) are of opposite sign then their exist at least 1 root lying between [a,b] of equation f(x)=0.

Newton Raphson Method

posted Dec 16, 2017, 2:07 PM by Atul Rana **[updated Jan 12, 2018, 12:15 AM]**

Description

Let x_0 be an initial approximation of the desired root of the equation *f(x)=0*, let *h* be the correction to be applied to x_0 to get the exact value of root i.e, ***x_0+h*** is the exact root.

$f(x_0 + h) = 0$ --------Equ 1

$f(x_0) + h*f`(x_0) / 1! + h^2 * f`(x_0) / 2! + h^3 * f`(x_0) / 3! + ... = 0$ ***by taylor's series***

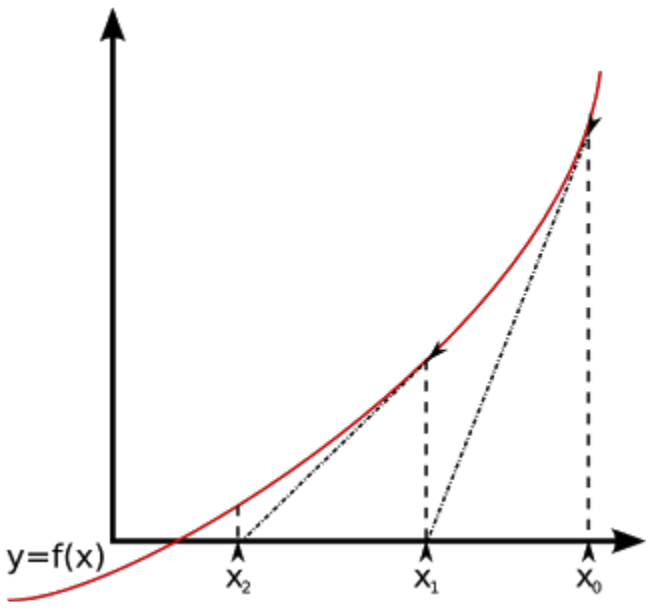
where **$f`(x_0)$** represents the defferentiation at perticular point x_0 and *n!* repersents the factorial of n.

Taking only $f(x_0) + h*f`(x_0) = 0$ and neglecting second and higher order terms, assuming that **h** is very small.
So, $h = -f(x_0) / f'(x_0)$ provided that $f'(x_0) != 0$

Now, this value of h is an approximate value of h considered in Equ 1. Hence first improvement value of the toot is given by
$x_1 = x_0 + h$ or $x_0 - f(x_0) / f'(x_0)$
SImilarly
$x_2 = x_1 - f(x_1) / f'(x_1)$
....
$x_{n+1} = x_n - f(x_n) / f'(x_n)$

provided that for all value of x_n n= 0,1,2,3... $f'(x_n)$!= 0 which is **Newton Raphson Method.**

- **Geometrical Representation of Newton Raphson Method**



Example: Find the root near x=2 for equ. x^4 - x - 10 =0 ?

choosing initial x_0 = 1.85 as f(1.8) >0 and f(1.9) < 0 hence (1.8 + 1.9)/2

$x_{n+1} = (3x_n^4 + 10) / (4x_n^3 -1)$

x_1 = 1.8956

x_2 = 1.8956

So, x = 1.8956 is the root of equation correct upto 4 decimal places.

Fixed Point Iteration Method

posted Dec 16, 2017, 1:16 AM by Atul Rana **[updated Jan 12, 2018, 12:30 AM]**

Description

Also called Functional Iteration method.

To find the root of equation *f(x)=0* by this method, first express the equation in the form ***x = phi(x)***

Let $x = x_0$ be the initial approximate of desired root **x = alpha**, which can be obtained by intermediate value theorem(find *[a,b]* where *f(a)* and *f(b)* are of opposite sign $x_0 = (a+b)/2$).

$x_1 = phi(x_0)$

$x_2 = phi(x_1)$

$x_3 = phi(x_2)$

proceeding in this way the nth approximation is given by

$x_n = phi(x_{n-1})$

thus we get a sequence of successive approximation of the root which may converge to the exact root **alpha**.

- Convergence of Iteration method

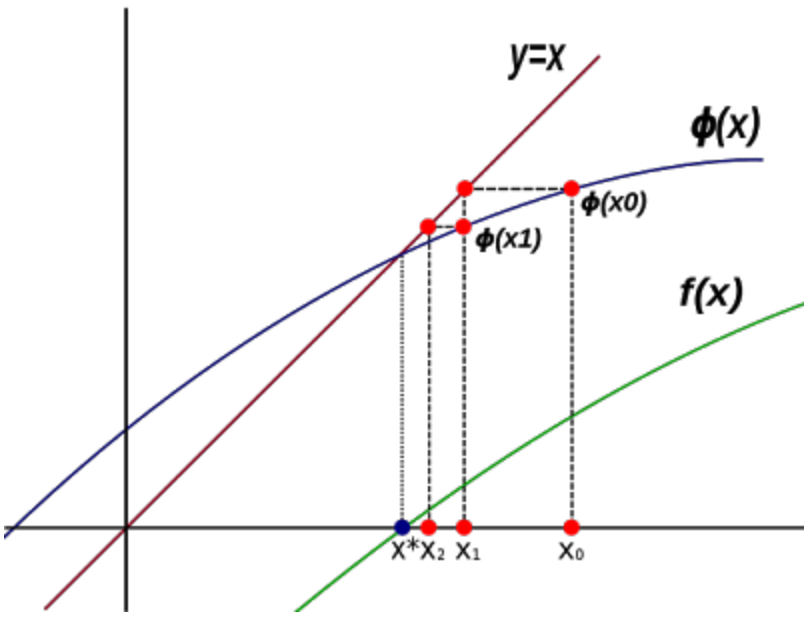
If **alpha** is a root of the equation *f(x)=0* which can be expressed as **x = phi(x)** and **I** be any interval containing alpha such that

$| phi`(x) | <= k < 1$ **for all x belogs to I**

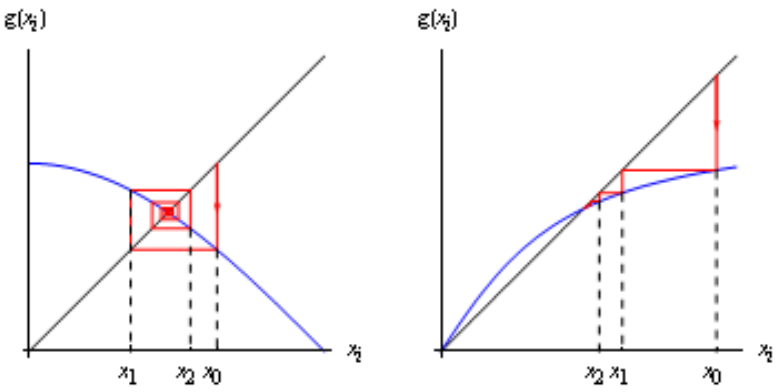
where phi`(x) represents the differentiation of phi(x).

then the sequence of approximation $x_1, x_2, x_3, ..., x_n$ was given by $\mathbf{x_n = phi(x_{n-1})}$ converges to root **alpha**, provided that the initial approximate root is chosen to form the interval **I**.

- **Graphical Representation of above Method can be shown below**.



if **x = g(x)** not using x=phi(x) ,there are more cases but finally cross point of g(x) or phi(x) and x=y will be the root of the equation.



Example: Find the root of equation sin(x) = 5x -2 or f(x) = 5x - 2 - sin(x).
x = (sin(x) + 2)/5 or phi(x) =1/5(sin(x)+2) also | phi'(x) | < 1. So we are good to go with our iterations.

x_0 = 0.45 as f(0.4) < 0 and f(0.5) > 0 , So (0.4 + 0.5)/2
x_1 = phi(x_0) = 0.4870
x_2 = phi(x_1) = 0.4936
x_3 = phi(x_2) = 0.4947
x_4 = phi(x_3) = 0.4950
x_5 = phi(x_4) = 0.4950
Finally , x = 0.4950 correct upto 4 decimal places.

Regula Falsi Method

posted Dec 15, 2017, 1:01 AM by Atul Rana **[updated 52 minutes ago]**

Description

Also called **Method of false position.**

Let the given equation is *f(x)=0*, in this method, we find sufficient sort interval in which a root lies.So we have **[x_0,x_1]** in which *f(x)=0* is assumed to be continuous and *f(x_0)* and *f(x_1)* are of opposite sign.

This method is based on the principle that a small portion of the smooth continuous function is approximately strict(straight line but actually it may be curved).

The curve y=f(x) is approximately straight line in between points *(x_a, y_a)* and *(x_b, y_b)* and the equation for this line can be given as

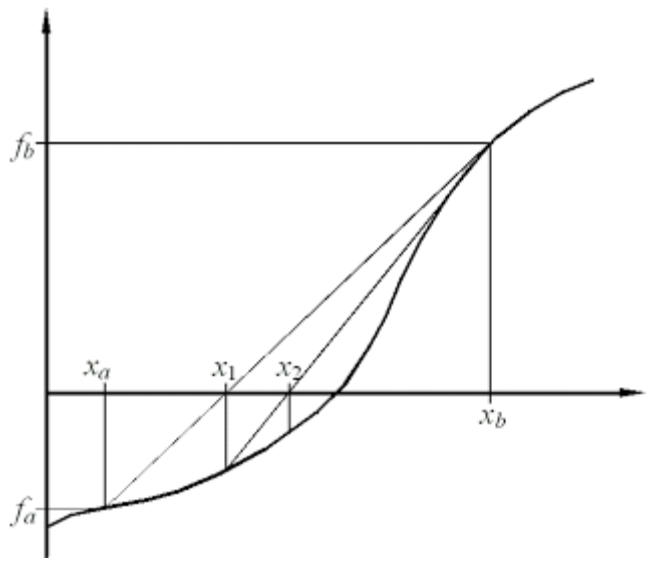
$y-y_a = (y_b - y_a) / (x_b - x_a)) * (x - x_a)$ ----------> *tangent property*

where $y_a = f(x_a)$ and $y_b = f(x_b)$

so the first approximate root value can obtained (x_1, y_1) by putting y_1=0 as this line cross the x-axis where the value of y_1 is 0,

$0-y_a = (y_b - y_a) / (x_b - x_a)) * (x_1 - x_a)$
$x_1 = (x_a*y_b - x_b*y_a) / (y_b - y_a)$
similarlly,
$x_2 = (x_a*y_1 - x_1*y_a) / (y_1 - y_a)$ if sign of x_1 maches with the x_b
$x_2 = (x_b*y_1 - x_1*y_b) / (y_1 - y_b)$ if sign of x_1 maches with the x_a
x_3 .....
x_4 .....
.....
where $y_1 = f(x_1)$, $y_2 = f(x_2)$ and $y_3 = f(x_3)$ so on.

- **Graphical Representation of Regula Falsi Method**



- **Programming implementation of Regula Falsi Method in C++.**

```
// including headers
#include<bits/stdc++.h>
using namespace std;
// function add your own for which you are working with.
double f0(double x){
    return x*x*x - x -1;
}
double f1(double x){
    return x*log10(x)-1.2;
}
double f2(double x){
    return x*exp(x)-2;
}
double f3(double x){
    return x*x*x*x*x*x - x*x*x*x - x*x*x -1;
}
// you can use pow function may be faster then is multiplication.
double f4(double x){
    return -1*x*x*x*x*x + 100*x*x*x - 20*x*x + 4*x -6;
}
int main(){
    double x0,x1,x2;
    double y0,y1,y2;

    // warning: enter the value seeing the value that double type and function that you
       //are using can store actually i am trying 10^6 with f3 ..HAHa :)

    do{
        cout<<"Give value of x where f(x) is -ve:"<<endl;
        cin>>x0;
    }while( f4(x0) > 0 );

    if( f4(x0) == 0){
        cout<<"You already got the ans. :)"<<endl;
        return 0;
    }

    do{
        cout<<"Give value of x where f(x) is +ve:"<<endl;
        cin>>x1;
    }while( f4(x1) < 0 );

    if( f4(x1) == 0){
        cout<<"You already got the ans. :)"<<endl;
        return 0;
    }
```

```
    //  here 100 is the number of iterations.
```

```
    for(int i=0;i < 100;i++){
        y0 = f4(x0);
        y1 = f4(x1);
        x2 = (x1*y0 - x0*y1)/(y0 - y1);
        y2 = f4(x2);

        if(y2 < 0)  x0 = x2;
        else if(y2 > 0) x1 = x2;
        else break;
    }

    printf("Value of x:%0.10f\n",x2);
    printf("Value of f(x):%0.10f\n",fx2);
    return 0;
}
```

- Execution of code: for f4(x) function

```
Give value of x where f(x) is -ve:
100
Give value of x where f(x) is +ve:
50
Value of x:99.7999947566
Value of f(x):-0.0000000072
```

For same function f4(x), same interval size and number of iterations, which method gives you the better approximation?

Try out the code, do experiment with it increase the displaying decimal digit, measure accuracy and have fun.

Bisection Method

posted Dec 14, 2017, 6:55 PM by Atul Rana **[updated 54 minutes ago]**

Description

Also called **Bolzano method or Interval halving method.**
This method is based on the repetitive application of the intermediate value theorem of differential calculus.

So *f(x)=0* be a given equation where *f(x)* is a countinuous in [a,b] where *f(a)* and *f(b)* are of opposite sign, then the root of equation must lies in [a,b].

- initial approximate root $x_0 = (a+b)/2$.

  If *f(x0)=0* , yeah we got solution else

- first approximate root

$x_1 = (a + x_0)/2$, if sign of x_0 matches with b.
$x_1 = (x0 + b)/2$, if sign of x_0 matches with a.
If $f(x_1)=0$, yeah we got solution else

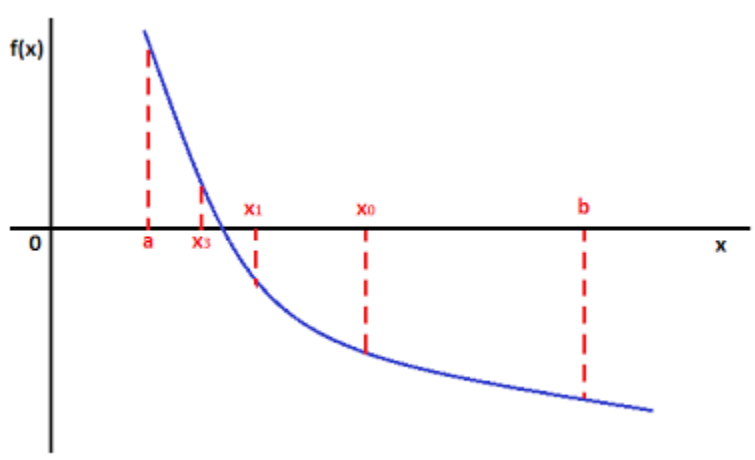
- second approximate root

similarly, every time we have two points in the function where the value of f(x) is of opposite sign.

- third approximate root
- ....
- n^{th} approximate root

As we proceed for more approximation error reduces for our approximate solution, and this is how you can achieve correct solution to nth decimal place of x for which *f(x)* approaches to 0;

- **Graphical Representation of Bisection Method.**



- **Programming implementation of Bisection Method.**

```
// including header

#include<bits/stdc++.h>
using namespace std;

// functions you can define your own
double f0(double x){
 return x*x*x - x -1;
}
double f1(double x){
    return x*log10(x)-1.2;
}
double f2(double x){
    return x*exp(x)-2;
}
double f3(double x){
 return x*x*x*x*x*x - x*x*x*x - x*x*x -1;
}
double f4(double x){
 return -1*x*x*x*x*x + 100*x*x*x - 20*x*x + 4*x -6;
}
// like above define your problem function change the name of function in the main()
// execution check and varify your solution.

int main(){
    double x0,x1,x2;
    double fx0 , fx1 , fx2;
    // taking values with checking the value of function at that point
    // weather satisfying the statement that is asking for.
```

```
   do{
    cout<<"Give value of x where f(x) is -ve:"<<endl;
    cin>>x0;
   }while( f4(x0) > 0 );

   if( f4(x0) == 0){
cout<<"You already got the ans. :)"<<endl;
return 0;
   }

   do{
    cout<<"Give value of x where f(x) is +ve:"<<endl;
    cin>>x1;
   }while( f4(x1) < 0 );

   if( f4(x1) == 0){
cout<<"You already got the ans. :)"<<endl;
return 0;
   }
   //where 100 is the number of iteration of for getting the better approximation

   for(int i=0;i < 100;i++){
fx0 = f4(x0);
fx1 = f4(x1);
   // here goes the bisection method the center of program
   // it just like binary search( or it is a binary search, :)
               x2 = (x0+x1)/2;
               fx2 = f4(x2);

               if(fx2 < 0)  x0 = x2;
               else if(fx2 > 0) x1 = x2;
               else break;
   }
    printf("Value of x:%0.10f\n",x2);
    printf("Value of f(x):%0.10f\n",fx2);
    return 0;
}
```

- Execution of code is like: as you can see we are using f4(x)

```
Give value of x where f(x) is -ve:
10
Give value of x where f(x) is -ve:
5
Give value of x where f(x) is -ve:
0
Give value of x where f(x) is +ve:
1
Value of x:0.4314601856
Value of f(x):0.0000000000
```

- Roots are interval dependent of course f4(x) is a 4th-degree polynomial equation.

```
Give value of x where f(x) is -ve:
100
Give value of x where f(x) is +ve:
50
Value of x:99.7999947566
Value of f(x):0.0000000376
```

try out code with different equations, interval size, the number of iterations, modify the code as required.