Term paper

Solution of Algebraic and Transcendental Equations using MATLAB

Submitted by:

Athulkrishna S V

Reg. No. 20mscphy08

Department of Physics

Abstract

In this term paper we will discuss different methods to find Solution of Algebraic and Transcendental Equations. And find out which one is better

Table of contents

1. INTRODUCTION

In scientific and engineering studies, a frequently occurring problem is to find the roots of equations of the form.

$$f\,(x)\;=\;0$$

If $f\,(x)$ is a quadratic, cubic or a biquadratic expression, then algebraic formulae are available for expressing the roots in terms of the coefficients. On the other hand, when $f\,(x)$ is a polynomial of higher degree or an expression involving transcendental functions, algebraic methods are not available, and recourse must be taken to find the roots by approximate methods.

algebraic functions of the form

$$\mathrm{f_n(x)}\;=\;a_0x^{\mathrm{n}}+a_0\,x^{\mathrm{n}-1}+a_2\,x^{\mathrm{n}-2}+\cdots+a_{\,n-1}\,x+an,$$

are called polynomials.

non-algebraic function is called a transcendental function, e.g., $f\,(x)=ln\,x^3-0.7$ $\phi(x)=e^{-05\mathrm{x}}-5x$ $\psi(x)=sin^2x-x^2-2$, etc. The roots of Eq may be either real or complex.

If $f\,(x)$ is a polynomial , the following results, from the theory of equations would be useful in locating its roots.

(i) Every polynomial equation of the n^{th} degree has n and only n roots.

(ii) If n is odd, the polynomial equation has at least one real root whose sign is opposite to that of the last term.

(iii) If n is even and the constant term is negative, then the equation has at least one positive root and at least one negative root.

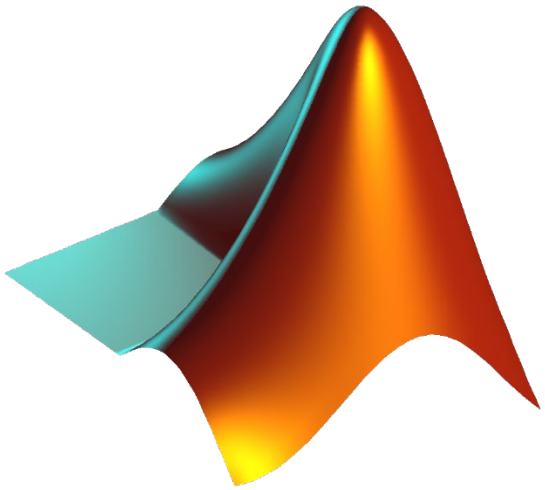
(iv) If the polynomial equation has (a) real coefficients, then imaginary roots occur in pairs and (b) rational coefficients, then irrational roots occur in pairs.

(v) Descartes' Rule of Signs

(a) A polynomial equation $f(x) = 0$ cannot have more number of positive real roots than the number of changes of sign in the coefficients of $f(x)$.

(b) In (a) above, $f(x) = 0$ cannot have more number of negative real roots than the number of changes of sign in the coefficients of $f(-x)$.

2. MATLAB



MATLAB (an abbreviation of "matrix laboratory") is a proprietary multi-paradigm programming language and numeric computing environment developed by MathWorks. MATLAB allows matrix manipulations, plotting of functions and data, implementation of algorithms, creation of user interfaces, and interfacing with programs written in other languages. Although MATLAB is intended primarily for numeric computing, an optional toolbox uses the MuPAD symbolic engine allowing access to symbolic computing abilities. An additional package, Simulink, adds graphical multi-domain simulation and model-based design for dynamic and embedded systems. As of 2020, MATLAB has more than 4

million users worldwide. MATLAB users come from various backgrounds of engineering, science, and economics.

3. BISECTION METHOD

This method is based on Theorem which states that if a function $f(x)$ is continuous between a and b, and $f(a)$ and $f(b)$ are of opposite signs, then there exists at least one root between a and b. For definiteness, let $f(a)$ be negative and $f(b)$ be positive. Then the root lies between a and b and let its approximate value be given by $x_O = (a+b)/2$. If $f(x_O) = 0$, we conclude that x_O is a root of the equation $f(x) = 0$. Otherwise, the root lies either between x_O and b, or between x_O and a depending on whether f (x_O) is negative or positive. We designate this new interval as [a_1, b_1] whose length is $|b - a|/2$. As before, this is bisected at x_1 and the new interval will be exactly half the length of the previous one. We repeat this process until the latest interval (which contains the root) is as small as desired, say e. It is clear that the interval width is reduced by a factor of one-half at each step and at the end of the nth step, the new interval will be [a_n, b_n] of length $|b - a|/2^n$. We then have

$$\frac{|b - a|}{2^n} \leq e$$

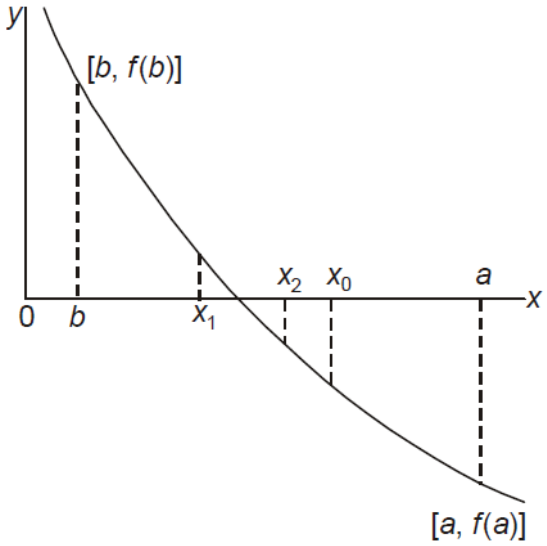
which gives on simplification

$$n \geq \frac{log_e\left(\frac{|b-a|}{e}\right)}{log_e 2}$$

Equation gives the number of iterations required to achieve an accuracy e. For example, if $|b-a| = 1$ and $e = 0.001$, then it can be seen that

$$n \geq 10$$

The method is shown graphically



It should be noted that this method always succeeds. If there are more roots than one in the interval, bisection method finds one of the roots. It can be easily programmed using the following computational steps:

(i) Choose two real numbers a and b such that $f\ (a)\ f\ (b)\ <\ 0$.

(ii) Set $x_r = (a + b)/2$.

(iii) If $f(a)f(x_r) < 0$, the root lies in the interval (a, x_r). Then, set $b\ =\ x\_r$ and go to step 2 above.

(b) If f (a) f (x_r) > 0, the root lies in the interval (x_r, b). Then, set a = x_r and go to step 2.

(c) If f (a) f (x_r) = 0, it means that x_r is a root of the equation f (x) = 0 and the computation may be terminated.

In practical problems, the roots may not be exact so that condition (c) above is never satisfied. In such a case, we need to adopt a criterion for deciding when to terminate the computations.

Bisection.m

```
%this is program to calculate root of Non linear
differential eqn
%c=.5 so choose  appropriate a and b


disp('you can change function in bsfun.m')
b=input("enter maximum value ");
a=input("enter minimum value ");
e=input("enter tolerance ");
if (bsfun(a)*bsfun(b)>0)
  disp("you have not assumed right values")
end
 i=0;
while (abs(b-a)>e)
  c=(b+a)/2;     %find middle point
  i=i+1;
  if (bsfun(c)==0)     %check middle point is the root
    break
  elseif (bsfun(a)*bsfun(c)<0)     % Decide the side to
%repeat the steps
```

```
    b=c;
  else
    a=c;
  end
  fprintf("on %d th iteration, value of c= %f\n",i,c);
end
```

bsfun.m

```
%bisection Function
function out = bsfun(x)
  out=x*exp(x)-1;
```

output

```
bisection
you can change function in bsfun.m
enter maximum value 1
enter minimum value 0
enter tolerance .001
on 1 th iteration, value of c= 0.500000
on 2 th iteration, value of c= 0.750000
```

```
on 3 th iteration, value of c= 0.625000

on 4 th iteration, value of c= 0.562500

on 5 th iteration, value of c= 0.593750

on 6 th iteration, value of c= 0.578125

on 7 th iteration, value of c= 0.570312

on 8 th iteration, value of c= 0.566406

on 9 th iteration, value of c= 0.568359

on 10 th iteration, value of c= 0.567383

>>
```

4. METHOD OF FALSE POSITION

This is the oldest method for finding the real root of a nonlinear equation $f\ (x)\ =\ 0$ and closely resembles the bisection method. In this method, also known as regula-falsi or the method of chords, we choose two points a and b such that f (a) and f (b) are of opposite signs. Hence, a root must lie in between these points. Now, the equation of the chord joining the two points [a, f (a)] and [b, f (b)] is given by

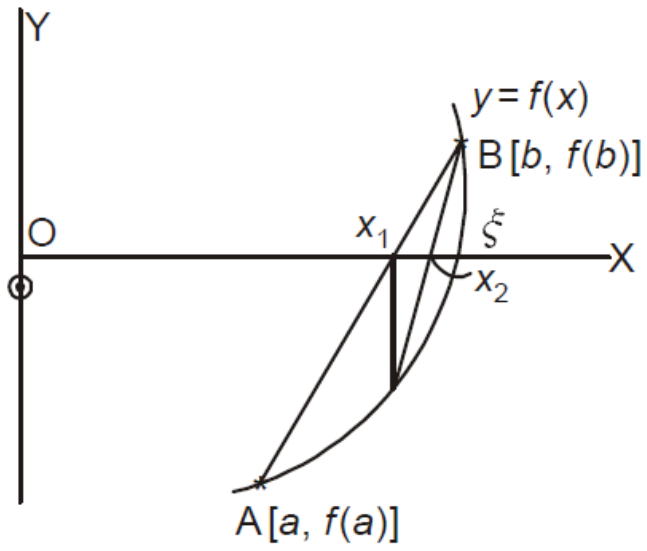
$$\frac{y-f(a)}{x-1}=\frac{f(b)-f(a)}{b-1}$$

The method consists in replacing the part of the curve between the points [a, f (a)] and [b, f (b)] by means of the chord joining these points, and taking the point of intersection of

the chord with the x-axis as an approximation to the root. The point of intersection in the present case is obtained by putting y = 0 in Eq. Thus, we obtain

$$x_1 = a - \frac{f(a)}{f(b) - f(a)}(b - a) = \frac{af(b) - bf(a)}{f(b) - f(a)}$$

which is the first approximation to the root of f (x) = 0. If now f (x_1) and f (a) are of opposite signs, then the root lies between a and x_1, and we replace b by x_1 in Eq. and obtain the next approximation. Otherwise, we replace a by x_1 and generate the next approximation. The procedure is repeated till the root is obtained to the desired accuracy. Figure gives a graphical representation of the method. The error criterion Eq. can be used in this case also.



fp.m

```
%this is program to calculate root of Non linear
differential eqn
%c=.56...
disp("you can change function in fpfun.m")
```

```
b=input("enter maximum value ");
a=input("enter minimum value ");
e=input("enter tolerance ");
if (fpfun(a)*fpfun(b)>0)
  disp("you have not assumed right values")
end
 i=0;
while (abs(a-b)>e)
  c = a-((b-a)*fpfun(a)/(fpfun(b)-fpfun(a)));    %find
%middle point
  i=i+1;
  if (bsfun(c)==0)    %check middle point is the root
    break
  elseif (fpfun(a)*fpfun(c)<0)    %Decide the side to
repeat the steps
    b=c;
  else
    a=c;
  end
  fprintf("on %d th iteration, value of c= %f\n",i,c);
  end
```

fpfun.m

```
function out = fpfun(x)

  out=(x*exp(x)-1);
```

output

```
>> fp

you can change function in fpfun.m

enter maximum value 1

enter minimum value 0

enter tolerance .001

on 1 th iteration, value of c= 0.367879

on 2 th iteration, value of c= 0.503314

on 3 th iteration, value of c= 0.547412

on 4 th iteration, value of c= 0.561115

on 5 th iteration, value of c= 0.565308

on 6 th iteration, value of c= 0.566585

on 7 th iteration, value of c= 0.566974

on 8 th iteration, value of c= 0.567092

on 9 th iteration, value of c= 0.567128

on 10 th iteration, value of c= 0.567139

on 11 th iteration, value of c= 0.567142
```

```
on 12 th iteration, value of c= 0.567143

on 13 th iteration, value of c= 0.567143

on 14 th iteration, value of c= 0.567143

on 15 th iteration, value of c= 0.567143

on 16 th iteration, value of c= 0.567143

on 17 th iteration, value of c= 0.567143

on 18 th iteration, value of c= 0.567143

on 19 th iteration, value of c= 0.567143

on 20 th iteration, value of c= 0.567143

on 21 th iteration, value of c= 0.567143

on 22 th iteration, value of c= 0.567143

on 23 th iteration, value of c= 0.567143

on 24 th iteration, value of c= 0.567143

on 25 th iteration, value of c= 0.567143

on 26 th iteration, value of c= 0.567143

on 27 th iteration, value of c= 0.567143

on 28 th iteration, value of c= 0.567143

on 29 th iteration, value of c= 0.567143

on 30 th iteration, value of c= 0.567143

>>
```

5. NEWTON–RAPHSON METHOD

This method is generally used to improve the result obtained by one of the previous methods. Let x_0 be an approximate root of f (x) = 0 and let $x_1 = x_0 + h$ be the correct root so that f (x_1) = 0. Expanding f (x_0 + h) by Taylor's series, we obtain

$$f(x_0) + hf'(x_0) + \frac{1}{2}h^2 f''(x_0) + \cdots = 0$$

Neglecting he second and higher-order derivatives, we have

$$f\,(x_0\,) \;+\; hf\,'(x_0\,) \;=\; 0$$

which gives

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

A better approximation than x_0 is, therefore, given by x_1, where

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

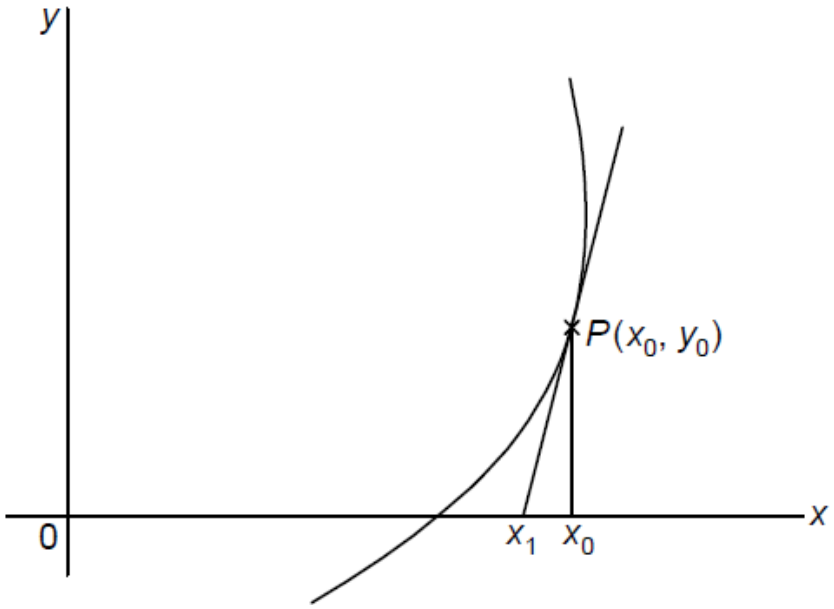
Successive approximations are given by x_2 , x_3 , … x_{n+1}, where

$$x_{n+1} = x_n - \frac{f(x_0)}{f'(x_0)}$$

which is the *Newton-Raphson formula.*

Geometrically, the method consists in replacing the part of the curve between the point [x_0, f (x_0)] and the x-axis by means of the tangent to the curve at the point, and is

described graphically in Fig. 2.3. It can be used for solving both algebraic and transcendental equations and it can also be used when the roots are complex.



nr.m

```
%Newton Rapson method to find root
x0=input('enter initial guess ');  %choose 0.5+
e=input('enter tolerance ');
x1=x0;
k=0;
while(1)
    k=k+1;
    x0=x1;
    x1=x0-nrfun(x0)/nrfun_diff(x0);
```

```
    fprintf('One root is %f obtained at %dth...
iteration\n',x1,k)
    if (abs(x1-x0)<e)
        break
    end
end
```

nrfun.m

```
function out = nrfun(x)
out=x*exp(x)-1;
end
```

nrfun_diff.m

```
function out = nrfun_diff(x)
out=x*exp(x)+x*exp(x);
end
```

output

```
>> nr
enter initial guess 5
```

```
enter tolerance .001
One root is 4.500674 obtained at 1th iteration
One root is 4.001907 obtained at 2th iteration
One root is 3.504191 obtained at 3th iteration
One root is 3.008482 obtained at 4th iteration
One root is 2.516686 obtained at 5th iteration
One root is 2.032725 obtained at 6th iteration
One root is 1.564942 obtained at 7th iteration
One root is 1.131750 obtained at 8th iteration
One root is 0.774214 obtained at 9th iteration
One root is 0.571978 obtained at 10th iteration
One root is 0.565361 obtained at 11th iteration
One root is 0.567832 obtained at 12th iteration
One root is 0.566882 obtained at 13th iteration
```

6. Conclusion

In open ended method we only need one guess to find the root. In case of Bracket method, we need 2 guess to find the root. Also in both bracket method we have linear error convergence, but in Newton Rapson method error convergence in Quadratic so we reach root speedily .

7. Reference

S.S. Sastry - Introductory methods of numerical analysis

https://www.mathworks.com/