CAPE LAB

Group : 1

Name : Satyam Rahangdale
Roll no. : 22CH10062

Week – 1 (10/01/25)

Topic – Non-linear Equation

Summary

- Problem Statement
- Solutions:
  a) Fixed-point iteration
  b) Newton’s method
  c) Bisection method
  d) MATLAB built in function fzero
- Conclusion

Problem

Find the molar volume(v) of ammonia at temperature T = 250 ˚C and pressure P = 10 atm using Var der Waals Equation of state.

$$\left(P + \frac{a}{v^2}\right)(v - b) = RT$$

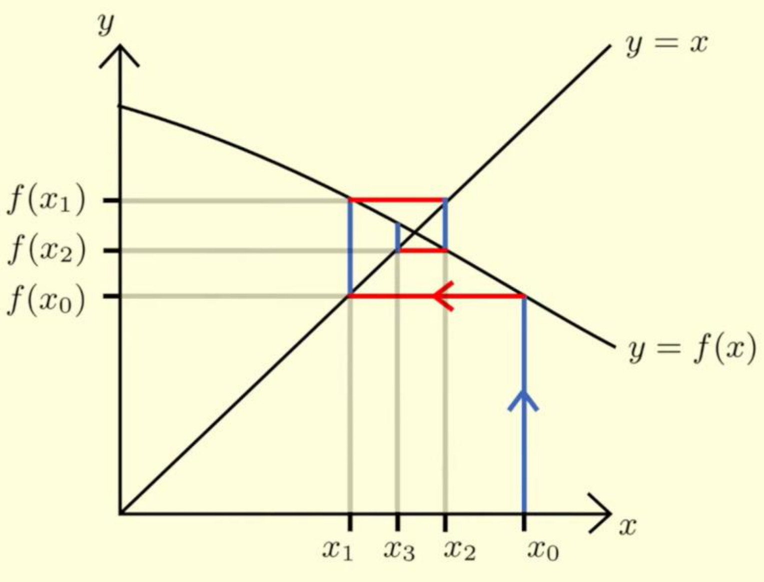
$$a = \frac{27R^2{T_c}^2}{64\text{Pc}} \qquad b = \frac{R\,T_c}{8\text{Pc}}$$

Given: T_c = 407.5 K , P_c = 111.3 atm , R = 0.08206 L atm $mol^{-1}\ K^{-1}$

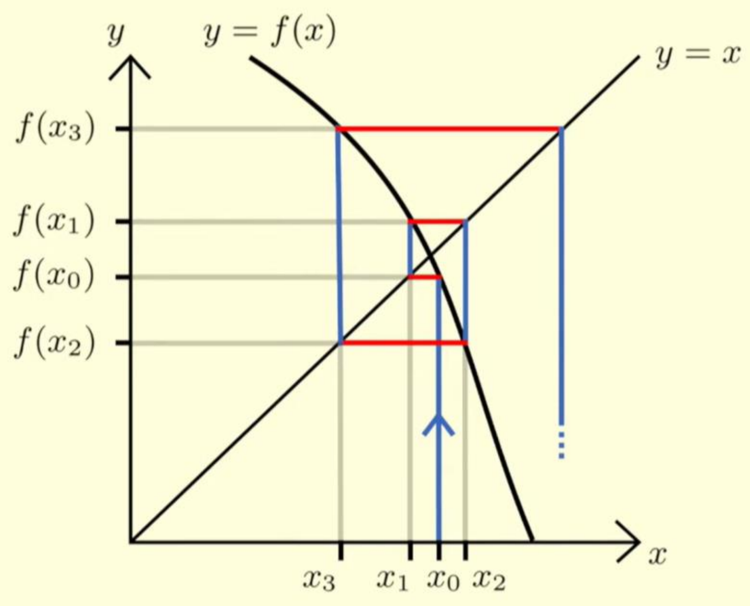
a) Fixed-point iteration

Algorithm :

1) To find the roots of the equation $f(x) = 0$, by successive approximations, we rewrite equation in the form $x = f(x)$.
2) Let $x = x_0$ be an initial approximation of the desired root. Then the first approximation x_1 is given by $x_1 = f(x_0)$.
3) Proceeding in this way, the nth approximation is given by $x_n = f(x_{n-1})$.



$-1 < f'(x) < 0$ [$convergence$]



$f'(x) < -1$ [$divergence$]

Results :-

- ✓ Let $x_0 = b + \frac{RT}{P} = 4$, assuming ideal gas behavior as a rough estimate.

- ✓ Given, $\left(P + \frac{a}{v^2}\right)(v - b) - RT = 0 \Rightarrow \boldsymbol{v = \left((RT + Pb) * \frac{v^2}{a}\right) + b - \left(P * \frac{v^3}{a}\right)}$

Output :

[iteration :0] : v = 4.32929 (Initial Guess)
[iteration :1] : v = -619.947
[iteration :2] : v = 2.3866e+09
[iteration :3] : v = -1.35937e+29
[iteration :4] : v = 2.51198e+88
[iteration :5] : v = -1.58507e+266
[iteration :6] : v = inf
[iteration :7] : v = nan
[iteration :8] : v = nan
[iteration :9] : v = nan
[iteration :10] : v = nan
Solution did not converge even after 11 iterations.
It can be seen clearly that solution is diverging.

Iteration(i)	v
0	4.32929
1	-619.947
2	2.39E+09
3	-1.36E+29
4	2.51E+88
5	-1.59E+266
6	inf
7	nan
8	nan
9	nan

Results :-

- ✓ Let $x_0 = b + \frac{RT}{P} = 4$, assuming ideal gas behavior as a rough estimate.
- ✓ Given, $\left(P + \frac{a}{v^2}\right)(v - b) - RT = 0 \Rightarrow \boldsymbol{v = b + \frac{(R*T)}{P+\frac{a}{v^3}}}$

Output :

[iteration :0] : v = 4.32929 (Initial Guess)
[iteration :1] : v = 4.23439
[iteration :2] : v = 4.23018
[iteration :3] : v = 4.22999
[iteration :4] : v = 4.22998
[iteration :5] : v = 4.22998
Solution converged at 5th iteration.
[v = 4.22998]

Iteration(i)	v
0	4.32929
1	4.23439
2	4.23018
3	4.22999
4	4.22998
5	4.22998

Analysis of Results :-

We observe that solution diverges for one choice of f(v) while converges on other choice. Why ?

Theorem: Sufficient condition for convergence of iterations.

If (i) α be a root of $f(x) = 0$ which is equivalent to $x = \phi(x)$,

(ii) I, be any interval containing the point $x = \alpha$,

(iii) $|\phi'(x)| < 1$ for all x in I,

then the sequence of approximations $x_0, x_1, x_2, \ldots, x_n$ will converge to the root α provided the initial approximation x_0 is chosen in I.

Since the above condition is not satisfied in formula 1 of f(v) hence the solution don't converge while that same condition is satisfied in formula 2 of f(v) hence the solution converges.

Recommendation : First find an interval $I = [a, b]$ such that root of equation lies in between, then check whether the chosen f(v) satisfies $| f'(x) | < 1$ for all x in I .

Analysis of Results :-

Comments :-

1) **The smaller the value of $|f'(x)|$, the more rapid will be the convergence.**
2) **This method of iteration is particularly useful for finding the real roots of an equation given in the form of an infinite series.**

Algorithm (Modified):

1) Find an interval $\boldsymbol{I}=[\boldsymbol{a},\boldsymbol{b}]$ such that root of equation lies in between.
2) To find the roots of the equation $f(x) = 0$, by successive approximations, we rewrite equation in the form $x = f(x)$.
3) Check whether the chosen f(x) satisfies $|\,\boldsymbol{f'(x)}\,| < \mathbf{1}$ for all x in I, if not then chose another f(x).
4) Let $x = x_0$ be an initial approximation of the desired root. Then the first approximation x_1 is given by $x_1 = f(x_0)$.
5) Proceeding in this way, the nth approximation is given by $x_n = f(x_{n-1})$.

Code

```cpp
#include <iostream> #include <cmath>
using namespace std;

int main() {
    // Process Constants
    double T = 250 + 273;// Temperature
    double T_c = 407.5 ;// Critical temperature
    double P = 10 ; // Pressure
    double P_c = 111.3; // Critical pressure
    double R = 0.08206; // Gas constant
    double a = 27*R*R*T_c*T_c/(64*P_c);
    double b = R*T_c/(8*P_c);

    // Initial guess
    double v_old = b + (R * T) / P;
    double v_new = 0.0;
    cout<<"[iteration :"<<0<<"] : v = "<<v_old<<" (Initial Guess) "<<endl;

    // Iterative process constants
    int maxIteration = 25;
    double tol = 1e-6;

    for(int i=1 ;i<=maxIteration ; i++){
        // Fixed-point iteration formula
        // Formula 1 :-
        // v_new = (((R*T + P*b)/a)*v_old*v_old) + b - ((P*v_old*v_old*v_old)) ;
        // Formula 2 :
        v_new = b + (R * T) / (P + (a / (v_old * v_old)));

        // Printing succesive values of v;
        cout<<"[iteration :"<<i<<"] : v = "<<v_new<<endl;
```

```
        // Check convergence
        if (abs(v_new - v_old) < tol) {
          cout << "Solution converged at " << i << "th iteration.\n";
          cout << "[ v = " << v_new <<" ]"<< endl;
          return 0;
        }
        v_old = v_new ; // Update v_old for next iteration

    }
    // If it still not converged after maxIteration allowed then..
    cout << "Solution did not converge even after " << maxIteration << " iterations.\n";
}
```

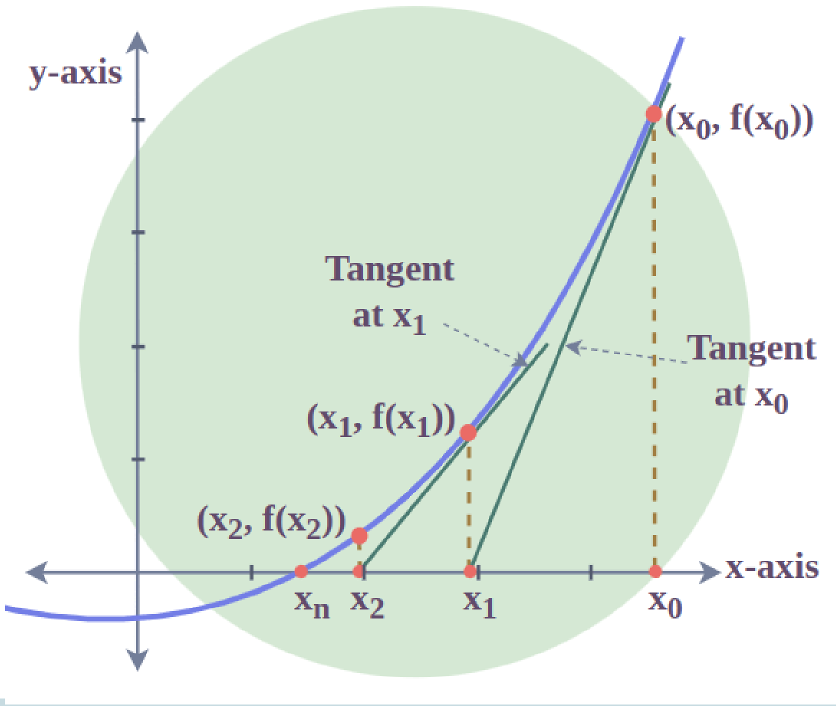
Code[Matlab]

b) Newton's Rapson Method

Algorithm :

1) Guess x_k
2) Find $f(x_k)$ and $f'(xk)$.
3) Find $x_{(k+1)} = x_k - \frac{f_k}{f'_k}$

✓ Order of convergence = 2.



Results :-

[iteration :0] : v = 10.00000000 (Initial Guess)
[iteration :1] : v = 4.26293462
[iteration :2] : v = 4.22998900
[iteration :3] : v = 4.22998303
[iteration :4] : v = 4.22998303

Iteration	v
0	10
1	4.262935
2	4.229989
3	4.229983
4	4.229983

Analysis of Results :-

Comments :-

1) The larger the value of $|f'(x)|$, the more rapid will be the convergence.
2) $f'(x)$ is small in the vicinity of the root, then computation of the root is slow or may not be possible. Thus, this method is not suitable in those cases where the graph of f(x) is nearly horizontal while crossing the x-axis.
3) Newton's method is generally used to improve the result obtained by other methods. It is applicable to the solution of both algebraic and transcendental equations.
4) Newton's formula will converge if $|f(x)\ f''(x)| < |f'(x)|^2$ in the interval considered. Assuming f(x), f '(x) and f ''(x) to be continuous, we can select a small interval in the vicinity of the root, in which the above condition is satisfied. Hence the result.

Code

```
#include <bits/stdc++.h>
using namespace std;
   // Process Variables:
    double T = (250 + 273);  // Temperature in Kelvin
    double T_c = (407.5);    // Critical temperature
    double P = (10);         // Pressure
    double P_c = (111.3);    // Critical pressure
    double R = 0.08206;      // Gas constant

    double a = 27 * R * R * T_c * T_c / (64 * P_c);  // Van der Waals 'a' constant
    double b = R * T_c / (8 * P_c);  // Van der Waals 'b' constant


double fxn(double v){
  return (P + (a / (v * v))) * (v - b) - R * T ;
}

double fxn_prime(double v){
    return (P + (a / (v * v))) - (2 * (a * (v - b)) / (v * v * v));
}

int main() {

    // Equation : (P + (a / v^2)) * (v - b) = R * T
    int T_itr = 20; // Maximum number of iterations
    // Tolerance limit
    double tol = 1e-6;
    // Unknown Variable (initial guess for v)
    double v = 10;
    cout<<fixed<<setprecision(8)<<"[iteration :"<<0<<"] : v = "<<v<<" (Initial Guess) "<<endl;
```

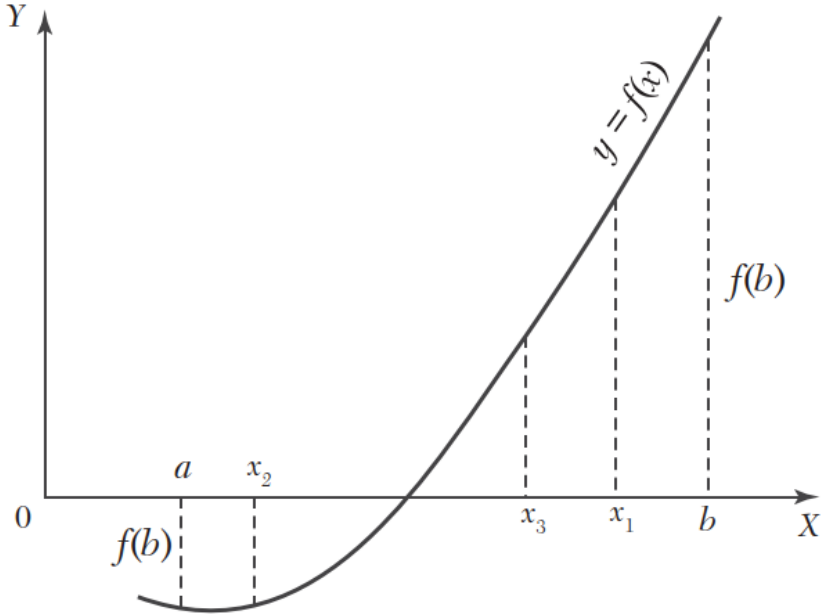
```
    // Iterative
    for(int i=1 ;i<=T_itr ; i++){
        // Update v using Newton's method
        double v_new = v - (fxn(v) / fxn_prime(v));

        // Output the current iteration and the new value of v
        cout<<"[iteration :"<<i<<"] : v = "<<v_new<<endl;

        // Check if the change in v is small enough to stop
        if (abs(v_new - v) < tol) {
            cout << "Solution converged at " << i << "th iteration.\n";
            cout << "[ v = " << v_new <<" ]"<< endl;
            return 0;
        }
        v = v_new;  // Update v for the next iteration
    }

    // If it still not converged after maxIteration allowed then...
    cout << "Solution did not converge even after " << T_itr << " iterations.\n";
}
```

Code[Matlab]

c) Bisection Method

Algorithm :

1) Find an interval $\boldsymbol{I} = [v1, v2]$ such that root of equation lies in between i.e. $f1 * f2 < 0$.
2) Initial guess $v = \frac{v1+v2}{2}$, $f\ = (value\ of\ fxn\ at\ v)$.
3) $If\ f >= 0\ , v2 = v\ ,\ else\ v1 = v$.
4) Continue till the $error = (v2 - v1) < 10^{-6}$.



✓ Order of convergence = 1.

✓ Let $v1 = 1, v2 = 8 \rightarrow v = 4.5$

Results :-

Output :

f1(at initial guess = 1) : -29.21366439 ; f2(at initial guess = 8) : 37.23438351

[iteration :0] : v1 = 1 : v2 = 8 : v = 4.5
[iteration :1] : v1 = 1 : v2 = 4.5 : v = 2.75
[iteration :2] : v1 = 2.75 : v2 = 4.5 : v = 3.625
[iteration :3] : v1 = 3.625 : v2 = 4.5 : v = 4.0625
[iteration :4] : v1 = 4.0625 : v2 = 4.5 : v = 4.28125
[iteration :5] : v1 = 4.0625 : v2 = 4.28125 : v = 4.171875
[iteration :6] : v1 = 4.171875 : v2 = 4.28125 : v = 4.2265625
[iteration :7] : v1 = 4.2265625 : v2 = 4.28125 : v = 4.25390625
[iteration :8] : v1 = 4.2265625 : v2 = 4.25390625 : v = 4.240234375
[iteration :9] : v1 = 4.2265625 : v2 = 4.240234375 : v = 4.233398438
[iteration :10] : v1 = 4.2265625 : v2 = 4.233398438 : v = 4.229980469
[iteration :11] : v1 = 4.229980469 : v2 = 4.233398438 : v = 4.231689453
[iteration :12] : v1 = 4.229980469 : v2 = 4.231689453 : v = 4.230834961
[iteration :13] : v1 = 4.229980469 : v2 = 4.230834961 : v = 4.230407715
[iteration :14] : v1 = 4.229980469 : v2 = 4.230407715 : v = 4.230194092
[iteration :15] : v1 = 4.229980469 : v2 = 4.230194092 : v = 4.23008728
[iteration :16] : v1 = 4.229980469 : v2 = 4.23008728 : v = 4.230033875
[iteration :17] : v1 = 4.229980469 : v2 = 4.230033875 : v = 4.230007172
[iteration :18] : v1 = 4.229980469 : v2 = 4.230007172 : v = 4.22999382
[iteration :19] : v1 = 4.229980469 : v2 = 4.22999382 : v = 4.229987144
[iteration :20] : v1 = 4.229980469 : v2 = 4.229987144 : v = 4.229983807
[iteration :21] : v1 = 4.229980469 : v2 = 4.229983807 : v = 4.229982138
[iteration :22] : v1 = 4.229982138 : v2 = 4.229983807 : v = 4.229982972

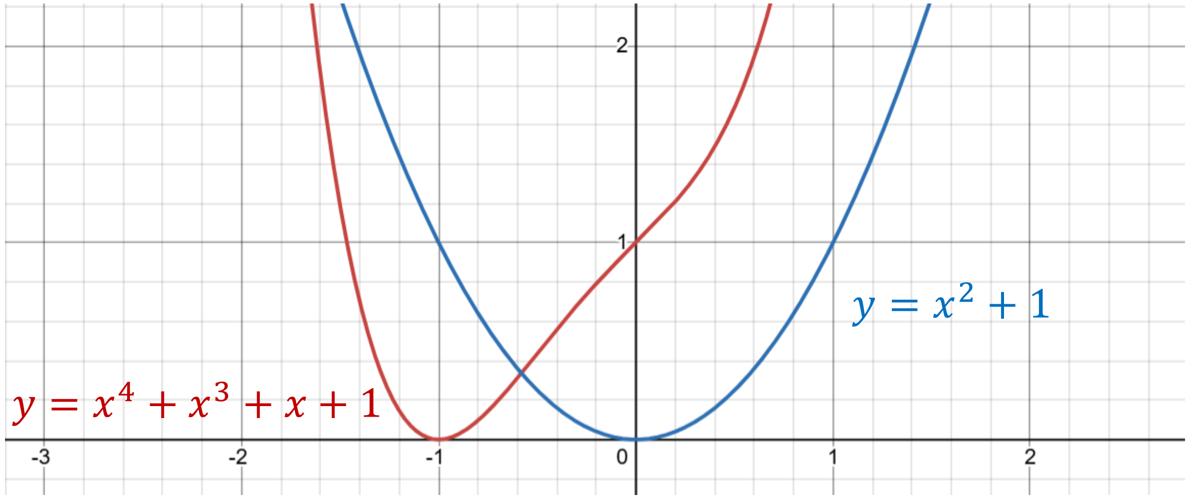
Final value of f = -5.899387077e-07 at v = 4.229982972

Iteration	v1	v2	v
0	1	8	4.5
1	1	4.5	2.75
2	2.75	4.5	3.625
3	3.625	4.5	4.0625
4	4.0625	4.5	4.28125
5	4.0625	4.28125	4.171875
6	4.171875	4.28125	4.2265625
7	4.2265625	4.28125	4.25390625
8	4.2265625	4.25390625	4.240234375
9	4.2265625	4.240234375	4.233398438
10	4.2265625	4.233398438	4.229980469
11	4.229980469	4.233398438	4.231689453
12	4.229980469	4.231689453	4.230834961
13	4.229980469	4.230834961	4.230407715
14	4.229980469	4.230407715	4.230194092
15	4.229980469	4.230194092	4.23008728
16	4.229980469	4.23008728	4.230033875
17	4.229980469	4.230033875	4.230007172
18	4.229980469	4.230007172	4.22999382
19	4.229980469	4.22999382	4.229987144
20	4.229980469	4.229987144	4.229983807
21	4.229980469	4.229983807	4.229982138
22	4.229982138	4.229983807	4.229982972

Analysis of Results :-

Comments :-

1) As the error decreases with each step by a factor of ½ the convergence in the bisection method is linear.
2) $n \geq [\log(b - a) - \log e]/\log 2$. This gives the number of iterations required for achieving an accuracy e.
3) If the function is continuous on [a,b] and $f(a) \cdot f(b) < \mathbf{0}$, the method is guaranteed to converge to a root.
4) The Bisection Method is a highly reliable but slow technique.
5) Fails only on cases where real and equal roots appear like ...



Code

```cpp
#include <bits/stdc++.h>
using namespace std;
// #define double long double
// Tolerance limit
double tol = 1e-6 ;
// Process Variables:
double T = (250 + 273) ;
double T_c = (407.5) ;
double P = (10) ;
double P_c = (111.3) ;
double R = 0.08206 ;
double a = 27*R*R*T_c*T_c/(64*P_c);
double b = R*T_c/(8*P_c);
// Unknown Variable
double v;
// Given Function
double fxn(double v){
  return ( ( P + (a/(v*v)) ) *  (v-b) ) - ( R * T );
}

int main()
{
  double v1 = 1; // inital guess 1
  double v2 = 8; // inital guess 1
  double f1,f2,f ;
  f1 = fxn(v1);
  f2 = fxn(v2);
  double v = (v1+v2)/2 ;
  f = fxn(v) ;
  cout<<setprecision(10)<<"f1( at initial guess = "<<v1<<" ) : "<<f1<<" ;  "<<"f2( at initial
guess = "<<v2<<" ) : "<<f2<<endl;
```

```
  int i=0;
  double e=1;
  // cout<<(long double)(2.0/3)<<endl;
  // cout<<"[iteration :"<<i++<<"] : v = "<<v<<endl;
  while(abs(e)>=tol){
    f1 = fxn(v1);
    f2 = fxn(v2);
    v = (v1+v2)/2 ;
    f = fxn(v) ;

    cout<<"[iteration :"<<i++<<"] : v1 = "<<v1<<" : v2 = "<<v2<<" : v = "<<v<<endl;
    if(f >= 0){
      v2 = v;
      e=v-v1;
    }
    else{
      v1 = v;
      e=v-v2;
    }
  }
  cout<<"Final value of f = "<<f<<" "<<"at [ v = "<<v<<" ]"<<endl;
}
```

Code

Code[Matlab]

d) Built in fzero

Algorithm :

1) Make initial guess of v as close as possible to solution.
2) Call built in fzero function

Code[Matlab]

```
% Matlab Inbuilt fxn fzero
clc,clearvars,tic
% Tolerance limit
tol = 1e-6 ;
% Process Variables:
T = (250 + 273) ;
T_c = (407.5) ;
P = (10) ;
P_c = (111.3) ;
R = 0.08206 ;
a = 27*R*R*T_c*T_c/(64*P_c);
b = R*T_c/(8*P_c);
% functions Required
fxn = @(v) (P + a/v^2) * (v - b) - R * T ;
% initial guess
v = 101 ;
v_new = fzero(fxn,v) ;
fprintf("Solution of given equation, v = %5f \n",v_new);
toc
```

Conclusion

Feature	Fixed Point Iteration	Newton-Raphson	Bisection Method	Inbuilt fxn : fzero
Order of Convergence	m <= 2 (depending on f(v))	m = 2	m = 1	-
Speed of Convergence	slow to moderate (depending on f(v))	Fast	slow	-
Accuracy	Moderate, depends on the function and fixed-point formulation.	High, provided the derivative is well-behaved and the initial guess is close.	Moderate, depends on the size of the interval but provides error bounds.	-
Computation Time(ms)	0.09	0.1	0.4	0.2
Number of Iterations	6	4	22	-
Requirements	$v = f(v)$	$f'(x)$	$-$	$f(x)$