Palestine Polytechnic University

College of IT & CE

Numerical Systems Analysis

Dr.Ahmad Khamayseh

StudentName:Maria abu Smoor

StudentID:201172

First Method

Bisection:

the binary search method . This method is used to find root of an equation in a given interval that is value of X for which f(x) = 0 there are two real numbers a and b such that f(a)*f(b) < 0), then it is guaranteed that it has at least one root between them.

The Algorithm:

For any continuous function f(x),

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

2.Find the midpoint of a and b, say “p”.

3.p is the root of the given function if f(p) = 0; else follow the next step.

4.Divide the interval [a, b]

– If f(p)*f(a) <0, there exist a root between t p and a so let b=p.
– else if f(p) *f (b) < 0, there exist a root between p and b so let a=p.

5.Repeat above three steps until the absolute value of f(p) became <= tolerance.

6.After end of the loop return p and here p is the solution of function f(x) on [a,b].

Bisection Code:

```
package numericalanalysisproject;

import java.util.Scanner;

public class NumericalAnalysisProject {

    public static void main(String[] args) {
         final double tolerance=Math.pow(10, -3);
        Scanner input = new Scanner(System.in);
        double a,b,result;
        System.out.println("Enter First limit of the interval : ");
        a=input.nextDouble();
        System.out.println("Enter Second limit of the interval : ");
        b=input.nextDouble();
        result=bisection(a, b, tolerance);
        if(result==0)
             System.out.println("can not apply bisection method tp find a solution on this inteval");
        else
        System.out.println("The root of the equation is :"+result);
    }
    public static double bisection(double a, double b,double tolerance ){
        double p=0.0;
        if((functionValue(a)*functionValue(b)>=0)||(a<0||b<0)) {
            return 0;
                    }
        else{
            p=(a+b)/2;
            while(Math.abs(functionValue(p))>tolerance){
                if(functionValue(p)==0)
                    break;
```



```
    else
    System.out.println("The root of the equation is :"+result);
}
public static double bisection(double a, double b,double tolerance ){
    double p=0.0;
    if((functionValue(a)*functionValue(b)>=0)||(a<0||b<0)) {
        return 0;
                }
    else{
        p=(a+b)/2;
        while(Math.abs(functionValue(p))>tolerance){
            if(functionValue(p)==0)
                break;
            else if(functionValue(p)*functionValue(a)<0)
                b=p;
            else
                a=p;
            p=(a+b)/2;
        }
    }
    return p;
}
public static double functionValue(double x){
    if(x!=-1)
      return (Math.pow(x, 2)+Math.sqrt(x)-1)/(x+1);
    return 0;

}
```

Output:

EX1:

```
Output - NumericalAnalysisProject (run) #3
run:
Enter First limit of the interval :
0
Enter Second limit of the interval :
1
The root of the equation is :0.525390625
BUILD SUCCESSFUL (total time: 8 seconds)
```

EX2:

```
Output - NumericalAnalysisProject (run) #3
run:
Enter First limit of the interval :
-2
Enter Second limit of the interval :
0
can not apply bisection method tp find a solution on this inteval
BUILD SUCCESSFUL (total time: 13 seconds)
```

Second Method

Fixed Point:

Fixed point iteration method is open and simple method for finding real root of non-linear equation by successive approximation. It requires only one initial guess to start. Since it is open method its convergence is not guaranteed. This method is also known as Iterative Method

To find the root of nonlinear equation f(x)=0 by fixed point iteration method, we write given equation f(x)=0 in the form of x = g(x). x0 is initial guess.

The Algorithm:

1. Start

2. Define function f(x)

3. Define function g(x) which is obtained

   from f(x)=0 such that x = g(x) and |g'(x) < 1| //function method

4. Choose initial guess p0, Tolerable Error tolerance.

5. Calculate pn = g(p0)

6. Initialize iteration counter: ctn = 1

7. if absolute value of pn-p0> tolerance go to next step

Else go to step 10

8. Set p0 = pn for next iteration

9. Increment iteration counter: ctn= ctn + 1

10. Display pn as root.

Fixed Point Code:

```java
package fixedpoint;

import java.util.Scanner;

public class FixedPoint {
    public static void main(String[] args) {
        final double tolerance=Math.pow(10, -3);
        Scanner input = new Scanner(System.in);
        double a,b,p0,result;
        System.out.println("Enter First limit of the interval : ");
        a=input.nextDouble();
        System.out.println("Enter Second limit of the interval : ");
        b=input.nextDouble();
       if(a<=-1||b<=-1)
            System.out.println("cannot solve for this interval");
       else{ System.out.println("Enter your intial guess \"it should be included on the interval\" : ");
        p0=input.nextDouble();
        if(p0>=b|| p0<=a)
            System.out.println("cannot solve for this initial guess");
        else
            fixedPoint(p0, tolerance);
    }
    }
    public static void fixedPoint(double p0,double tolerance){
          double pn=function(p0);
          int ctn=0;
          while(Math.abs(pn-p0)>tolerance){
              System.out.println(++ctn+"\t "+pn);
              p0=pn;
              pn=function(p0);
          }
```



```java
        System.out.println("Enter First limit of the interval : ");
        a=input.nextDouble();
        System.out.println("Enter Second limit of the interval : ");
        b=input.nextDouble();
       if(a<=-1||b<=-1)
            System.out.println("cannot solve for this interval");
       else{ System.out.println("Enter your intial guess \"it should be included on the interval\" : ");
        p0=input.nextDouble();
        if(p0>=b|| p0<=a)
            System.out.println("cannot solve for this initial guess");
        else
            fixedPoint(p0, tolerance);
    }
    }
    public static void fixedPoint(double p0,double tolerance){
          double pn=function(p0);
          int ctn=0;
          while(Math.abs(pn-p0)>tolerance){
              System.out.println(++ctn+"\t "+pn);
              p0=pn;
              pn=function(p0);
          }
    }
    public static double function(double p){
     return (p-Math.sqrt(p)+1)/(p+1);

    }
}
```

Output:

EX1:

Output - NumericalAnalysisProject (run) #5

```
run:
Enter First limit of the interval :
0
Enter Second limit of the interval :
10
Enter your intial guess "it should be included on the interval" :
9
1        0.7
2        0.5078470432152496
3        0.5273831781515597
4        0.5245387439285399
BUILD SUCCESSFUL (total time: 12 seconds)
```

EX2:

tput - NumericalAnalysisProject (run) #5

```
run:
Enter First limit of the interval :
0
Enter Second limit of the interval :
1
Enter your intial guess "it should be included on the interval" :
0.0005
1        0.9776504949775133
2        0.5000319293891068
3        0.5285904623365892
4        0.5243707926399686
BUILD SUCCESSFUL (total time: 12 seconds)
```

EX3:

Output - NumericalAnalysisProject (run) #5

```
run:
Enter First limit of the interval :
-1
Enter Second limit of the interval :
0
cannot solve for this interval
BUILD SUCCESSFUL (total time: 3 seconds)
```

Third Method

Newton’s Method:

is an open method and starts with one initial guess for finding real root of nonlinear equations.

In Newton Raphson method if p0is initial guess then next approximated root pn is obtained by following formula:

P1 = P0- f(P0) / g(P0)//g(P0) is the derivative of f(x) with respect to p0

And an algorithm for Newton Raphson method involves repetition of above process i.e. we use p1 to find p2 and so on until we find the root within desired accuracy.

The Algorithm:

1. Start

2. Define function as f(x)

3. Define first derivative of f(x) as g(x)

4. Input initial guess (P0) and tolerance

5. Calculate pn = p0 - f(p0) / g(p0)

6.Initialize iteration counter i = 1

7. if absolute value of pn-p0> tolerance go to next step

Else go to step 10

8. Increment iteration counter i = i + 1

9. let p0=pn then pn = p0 - f(p0) / g(p0).

10. Print root as pn

12. Stop

Newton’s Method Code:

```java
package newton;
import java.util.Scanner;

public class NewtonMethod {
    public static void main(String[] args) {
         final double tolerance=Math.pow(10, -3);
        Scanner input = new Scanner(System.in);
        double a,b,p0,result;
        System.out.println("Enter First limit of the interval : ");
        a=input.nextDouble();
        System.out.println("Enter Second limit of the interval : ");
        b=input.nextDouble();
       if(a<=-1||b<=-1)
            System.out.println("cannot solve for this interval");
       else{
           System.out.println("Enter your intial guess \"it should be included on the interval\" : ");
        p0=input.nextDouble();
        if(p0>=b|| p0<=a)
            System.out.println("cannot solve for this initial guess");
        else
            newton(p0, tolerance);
    }
    }
    public static void newton(double p0,double tolerance){
        double pn=p0-(function(p0)/derivative(p0));
        int ctn=0;
        while(Math.abs(pn-p0)>tolerance){
              System.out.println(++ctn+"\t "+pn);
              p0=pn;
              pn=p0-(function(p0)/derivative(p0));
         }
```



```java
    public static void newton(double p0,double tolerance){
        double pn=p0-(function(p0)/derivative(p0));
        int ctn=0;
        while(Math.abs(pn-p0)>tolerance){
              System.out.println(++ctn+"\t "+pn);
              p0=pn;
              pn=p0-(function(p0)/derivative(p0));
         }
```

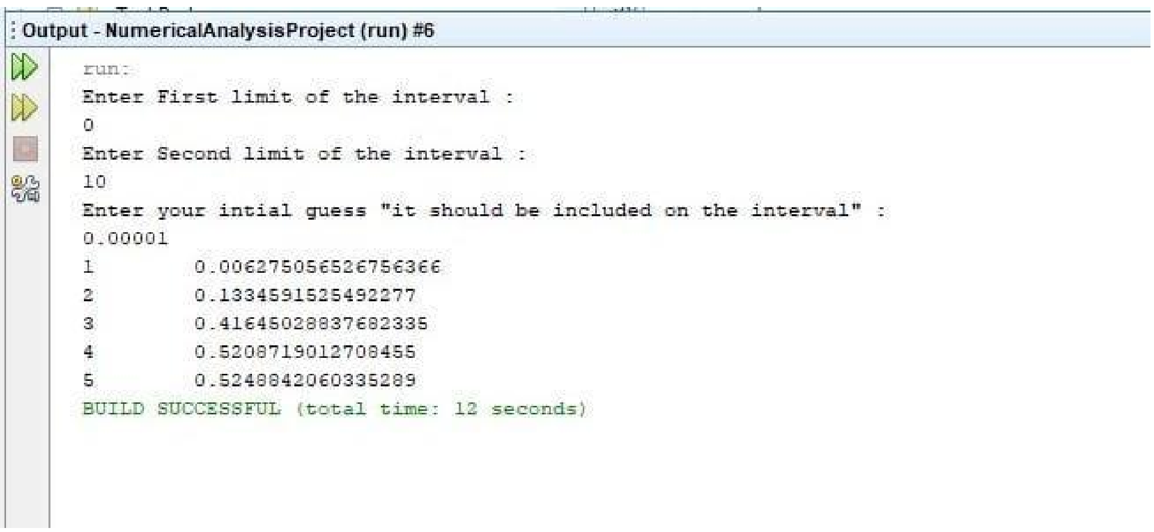

```java




    }
    public static double derivative(double p){
    return (Math.pow(p, 2)+2*p+((1-p)/(2*Math.sqrt(p)))+1)/Math.pow(p+1, 2);
    }
    public static double function(double p){
      return (Math.pow(p, 2)+Math.sqrt(p)-1)/(p+1);
    }
}
```

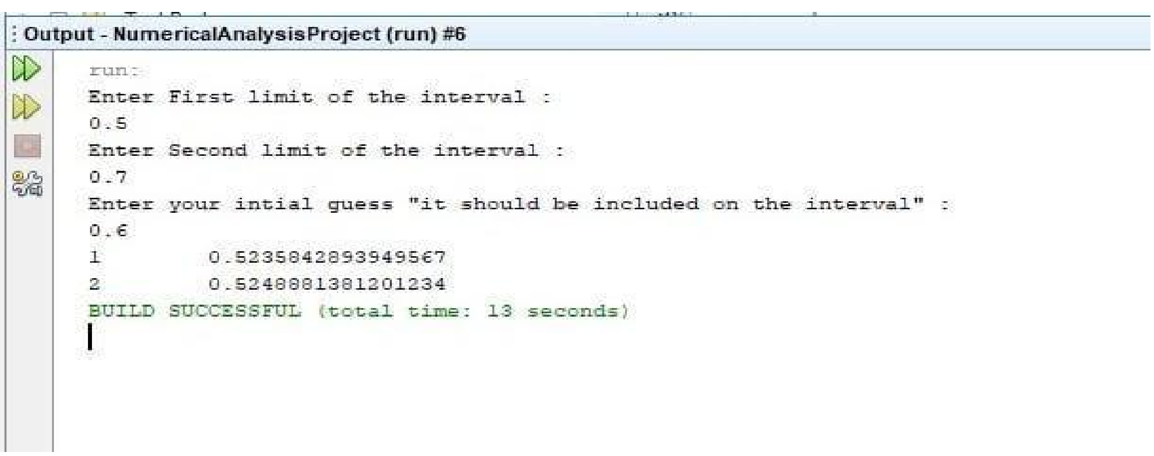
Output:

EX1:



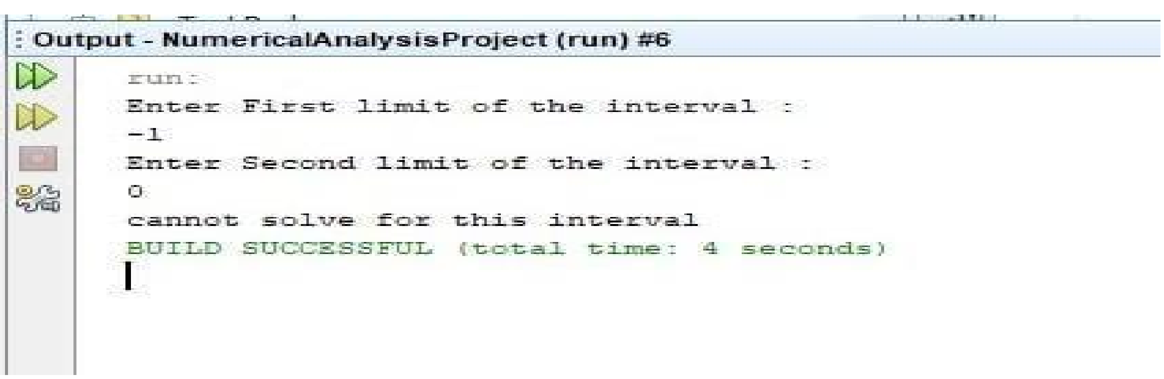
```
Output - NumericalAnalysisProject (run) #6
run:
Enter First limit of the interval :
0
Enter Second limit of the interval :
10
Enter your intial guess "it should be included on the interval" :
0.00001
1        0.006275056526756366
2        0.1334591525492277
3        0.41645028837682335
4        0.5208719012708455
5        0.5248842060335289
BUILD SUCCESSFUL (total time: 12 seconds)
```

EX2:



```
Output - NumericalAnalysisProject (run) #6
run:
Enter First limit of the interval :
0.5
Enter Second limit of the interval :
0.7
Enter your intial guess "it should be included on the interval" :
0.6
1        0.5235842893949567
2        0.5248881381201234
BUILD SUCCESSFUL (total time: 13 seconds)
```

EX3:



```
Output - NumericalAnalysisProject (run) #6
run:
Enter First limit of the interval :
-1
Enter Second limit of the interval :
0
cannot solve for this interval
BUILD SUCCESSFUL (total time: 4 seconds)
```

Fourth Method

Secant Method:

In Secant method if p0 and p1 are initial guesses then next approximated root p2 is obtained by following formula:

p2 = p1 - (p1-p0) * f(p1) / (f(p1) - f(p0))

And an algorithm for Secant method involves repetition of above process i.e. we use p1 and p2 to find p3 and so on until we find the root within desired accuracy.

Secant Method Algorithm:

1. Start

2. Define function as f(x)

3. Input initial guesses (p0 and p1) and tolerable error (tolerance)

4. Initialize iteration counter ctn = 1

5. Calculate pn = p1 - (p1-p0) * f(p1) / (f(p1) - f(p0))

6. Increment iteration counter ctn = ctn + 1

7.let p0 = p1 and p1 = p2 Then calculate pn again.

8. if absolute value of p1-p0> tolerance go to next step(loop)

else go to step 10

9. Print root as x2

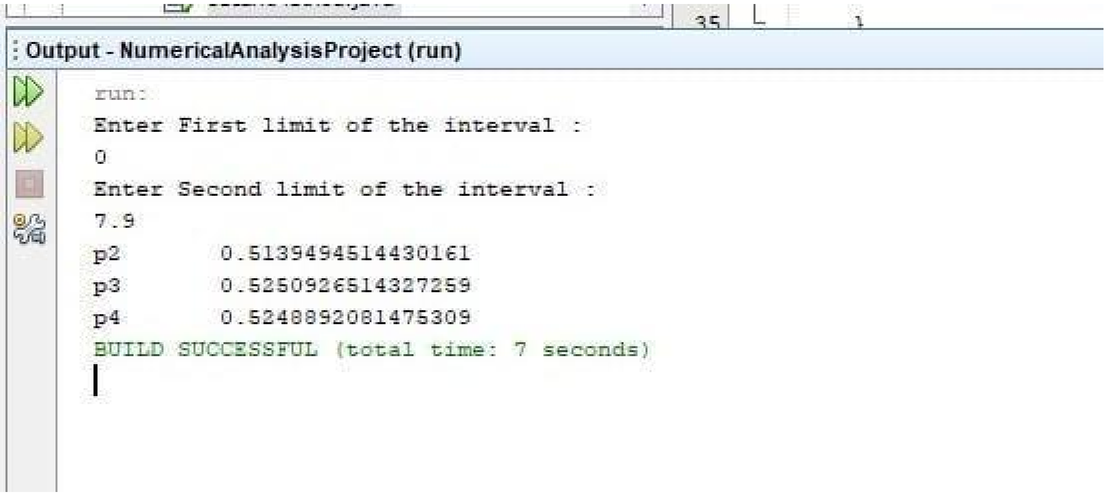
10. Stop

Secant Method Code:

```java
public class secantMethod {
    public static void main(String[] args) {
         final double tolerance=Math.pow(10, -3);
        Scanner input = new Scanner(System.in);
        double a,b,p0,p1;
        System.out.println("Enter First limit of the interval : ");
        a=input.nextDouble();
        System.out.println("Enter Second limit of the interval : ");
        b=input.nextDouble();
       if(a<=-1||b<=-1)
            System.out.println("cannot solve for this interval");
       else{
           p0=(a+b)/2;
           p1=(p0-Math.sqrt(p0)+1)/(p0+1);
           secant(p0, p1, tolerance);
       }
}
    public static double function(double p){
      return (Math.pow(p, 2)+Math.sqrt(p)-1)/(p+1);
    }
    public static void secant(double p0,double p1,double tolerance){
       double pn;
       int ctn=1;
       pn=(p0*function(p1)-p1*function(p0))/(function(p1)-function(p0));
       while (Math.abs(p1-p0)>tolerance){
           System.out.println("p"+(++ctn)+"\t "+pn);
           p0=p1;
           p1=pn;
           pn=(p0*function(p1)-p1*function(p0))/(function(p1)-function(p0));
       }
    }
}
```

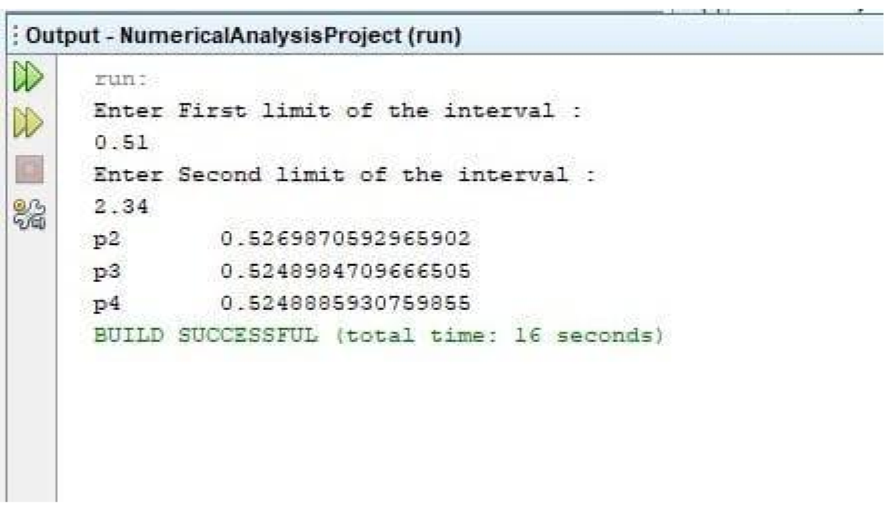
Output:

EX1:



```
Output - NumericalAnalysisProject (run)
run:
Enter First limit of the interval :
0
Enter Second limit of the interval :
7.9
p2	0.5139494514430161
p3	0.5250926514327259
p4	0.5248892081475309
BUILD SUCCESSFUL (total time: 7 seconds)
```

EX2:



```
Output - NumericalAnalysisProject (run)
run:
Enter First limit of the interval :
0.51
Enter Second limit of the interval :
2.34
p2	0.5269870592965902
p3	0.5248984709666505
p4	0.5248885930759855
BUILD SUCCESSFUL (total time: 16 seconds)
```

Drive Link For Codes:

https://drive.google.com/drive/folders/1V-4nbZo0cVGxg-8r99UyVBAZ7pIyVScv?usp=sharing