**Name of the assignment**: Bisection Method and False Position Method

**Problem statement and Solution**

**Problem Statement 1:**

Consider the following function



In this case, first, you need to draw three graphs as following.

In graph 1: for n=0, and x = 0 to x = 10; increase x by 0.1 with value k starts from 1.

In graph 2: for n=1, and x = 0 to x = 10; increase x by 0.1 with value k starts from 1.

In graph 3: for n=2, and x = 0 to x = 10; increase x by 0.1 with value k starts from 1.

Secondly, Using the series expansion for Jn(x), find its root for J0(x) to an accuracy of four decimal places by using bisection method. Consider the initial guesses as 1 and 3. The desired level of accuracy is 0.00001. At first, print the value of x and J0(x) from 1 to 3, increasing by 0.1. Then, ask the user for upper bound and lower bound. If the root finding is possible, print the solution, otherwise print no root is possible. You also need to print the following table in your console view.

Lastly, Draw two graphs from above solution.

In graph 1: the graph of x and relative approximation error.

In graph 2: the graph of no of iteration and relative approximation error.

**Solution:**

Solution (program source code) for Problem statement 1:

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#include <limits.h>

double error\_tolerance = 0.00001;

long long int fact[15];

void calculate\_factorial(){

fact[0] = 1;

fact[1] = 1;

for(int i=2; i<14; i++){

fact[i] = fact[i-1] \* i;

}

}

double func(double x, int n){

double temp1 = pow((x/2), n);

double temp2 = 0.0;

for(int k= 0; k<=10; k++){

temp2 = temp2 + ( (pow(-1, k) \* pow((x\*x/4), k))/(fact[k]\* fact[k+n]));

}

    return temp2\*temp1;

}

double get\_error(double xl, double xu){

return (fabs((xu - xl) / (xu + xl)) \* 100.0);

}

void bisection(double xl, double xu){

    if(func(xl,0) \* func(xu,0) >= 0){

        printf("Wrong guess of xl and/or xu\n");

        return;

    }

    double xr = 0.0;

    int iteration = 0;

    while(get\_error(xl, xu) > error\_tolerance){

        xr = (xl + xu) / 2;

        if(func(xr,0) == 0.0){

            break;

        }else{

printf("%d\t%.4lf\t%0.4lf\t%0.4lf\t%0.4lf\t%0.4lf\n", ++iteration, xl, xu, xr, func(xr,0), get\_error(xl, xu));

        }

        if(func(xl,0) \* func(xr,0) < 0.0){

            xu = xr;

        }else{

            xl = xr;

        }

    }

    printf("\nX (approx.) = %0.5lf\n", xr);

}

int main(){

    //freopen("prob\_sln.csv", "w", stdout);

    calculate\_factorial();

    for(double i=0; i<3.1;i+= 0.1){

printf("%lf\t%lf\n",i, func(i, 0) );

}

    double xl,xu;

printf("\nEnter lower and upper bound\n");

scanf("%lf %lf", &xl, &xu);

printf("\nEnter relative errror tolerance\n");

scanf("%lf", &error\_tolerance);

    bisection(xl, xu);

    return 0;

}

**Sample Input/output:**

i J0(i)

0.000000 1.000000

0.100000 0.997502

0.200000 0.990025

0.300000 0.977626

0.400000 0.960398

0.500000 0.938470

0.600000 0.912005

0.700000 0.881201

0.800000 0.846287

0.900000 0.807524

1.000000 0.765198

1.100000 0.719622

1.200000 0.671133

1.300000 0.620086

1.400000 0.566855

1.500000 0.511828

1.600000 0.455402

1.700000 0.397985

1.800000 0.339986

1.900000 0.281819

2.000000 0.223891

2.100000 0.166607

2.200000 0.110362

2.300000 0.055540

2.400000 0.002508

2.500000 -0.048384

2.600000 -0.096805

2.700000 -0.142449

2.800000 -0.185036

2.900000 -0.224312

3.000000 -0.260052

Enter lower and upper bounds

1 3

Enter relative error tolerance

0.00001

1 1.0000 3.0000 2.0000 0.2239 50.0000

2 2.0000 3.0000 2.5000 -0.0484 20.0000

3 2.0000 2.5000 2.2500 0.0827 11.1111

4 2.2500 2.5000 2.3750 0.0156 5.2632

5 2.3750 2.5000 2.4375 -0.0168 2.5641

6 2.3750 2.4375 2.4063 -0.0007 1.2987

7 2.3750 2.4063 2.3906 0.0074 0.6536

8 2.3906 2.4063 2.3984 0.0033 0.3257

9 2.3984 2.4063 2.4023 0.0013 0.1626

10 2.4023 2.4063 2.4043 0.0003 0.0812

11 2.4043 2.4063 2.4053 -0.0002 0.0406

12 2.4043 2.4053 2.4048 0.0000 0.0203

13 2.4048 2.4053 2.4050 -0.0001 0.0102

14 2.4048 2.4050 2.4049 -0.0000 0.0051

15 2.4048 2.4049 2.4048 -0.0000 0.0025

16 2.4048 2.4048 2.4048 0.0000 0.0013

17 2.4048 2.4048 2.4048 -0.0000 0.0006

18 2.4048 2.4048 2.4048 0.0000 0.0003

19 2.4048 2.4048 2.4048 -0.0000 0.0002

20 2.4048 2.4048 2.4048 0.0000 0.0001

21 2.4048 2.4048 2.4048 -0.0000 0.0000

22 2.4048 2.4048 2.4048 -0.0000 0.0000

X (approx.) = 2.40483

**Graphs:**

(a)

1. Graph of the function for the value of for n=0, and x = 0 to x = 10; increase x by 0.1 with value k starts from 1 to 10

Fig 1.1

1. Graph of the function for the value of for n=1, and x = 0 to x = 10; increase x by 0.1 with value k starts from 1 to 10

Fig 1.2

1. Graph of the function for the value of for n=2, and x = 0 to x = 10; increase x by 0.1 with value k starts from 1

Fig 1.3

(b)

Fig 1.4: Graph of Xm and relative approximate error (error value is not multiplied by 100 here)

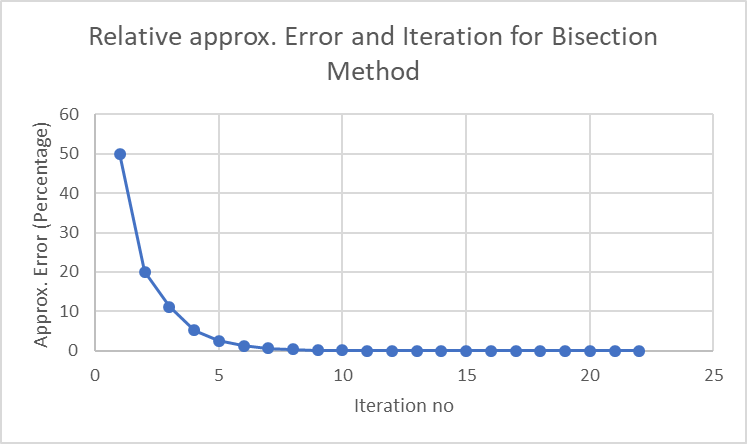
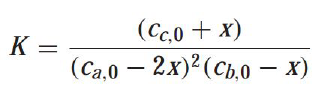


Fig: 1.5: Relative approx. Error and Iteration for Bisection Method

**Problem Statement 2:**

Conservation of mass can be used to re-formulate the equilibrium relationship as



where the subscript 0 designates the initial concentration of each constituent. If K = 0.016, ca,0 = 42, cb,0 = 28, and cc,0 = 4, determine the value of x.

1. Obtain the solution graphically by plotting the value 0 from 20, with increment of 1.
2. On the basis of (a), solve for the root with initial guesses of xl = 0 and xu= 20 with desired accuracy level of 0.00001 Choose false position to obtain your solution. Ask the user for upper bound and lower bound. Justify your solution if it is not possible.
3. Compare the relative approximate error between the bisection method and false position method. You need to use the previous problem (Problem 1) solution partially. For comparison, you need to draw the graph of number of iteration and relative approximation error.

**Solution:**

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#include <limits.h>

double error\_tolerance = 0.00001;

double func(double x) {

return ((x +4)/( (42 - 2\*x)\*(42 - 2\*x)\*(28 -x) ) ) - 0.016;

}

double false\_position(double xl, double xu) {

int iter = 1;

double xr;

double xr\_prev = INT\_MAX, approxError = INT\_MAX;

if(func(xl) \* func(xu) > 0) {

printf("Wrong guess of xl xu\n");

return 0;

}

do {

if(iter != 1)

xr\_prev = xr;

xr = xu-func(xu)\*(xl-xu)/( func(xl)-func(xu) );

approxError = fabs((xr - xr\_prev) / xr) \* 100;

printf("%8d\t%0.8lf\t%0.8lf\t%0.8lf\t%0.8lf\t%0.8lf\n", iter, xl, xu, xr, func(xr), (iter==1) ? 0 : approxError);

//printf("%8d,%0.8lf,%0.8lf,%0.8lf,%0.8lf,%0.8lf\n", iter, xl, xu, xr, func(xr), (iter==1) ? 0 : approxError);

if(func(xr) == 0.0) {

break;

}

if(func(xl) \* func(xr) < 0) {

xu = xr;

} else {

xl = xr;

}

iter++;

} while(approxError >= error\_tolerance);

printf("x = %lf\n", xr);

}

int main() {

//freopen("prob2.csv", "w", stdout);

for(double i=0; i<=20;i++){

printf("%lf\t%lf\r\n",i, func(i) );

}

double xl,xu;

printf("Enter lower and upper bound\n");

scanf("%lf %lf", &xl, &xu);

printf("\nEnter relative errror tolerance\n");

scanf("%lf", &error\_tolerance);

false\_position(xl,xu);

return 0;

}

1. **Sample Input/output:**

0.000000 -0.015919

1.000000 -0.015884

2.000000 -0.015840

3.000000 -0.015784

4.000000 -0.015712

5.000000 -0.015618

6.000000 -0.015495

7.000000 -0.015332

8.000000 -0.015112

9.000000 -0.014812

10.000000 -0.014393

11.000000 -0.013794

12.000000 -0.012914

13.000000 -0.011573

14.000000 -0.009440

15.000000 -0.005850

16.000000 0.000667

17.000000 0.013830

18.000000 0.045111

19.000000 0.143722

20.000000 0.734000

Enter lower and upper bound

10 18

Enter relative error tolerance

0.0001

1 10.00000000 18.00000000 11.93506173 -0.01298224 0.00000000

2 11.93506173 18.00000000 13.29040572 -0.01105597 10.19791285

3 13.29040572 18.00000000 14.21744549 -0.00881691 6.52043835

4 14.21744549 18.00000000 14.83587073 -0.00658566 4.16844585

5 14.83587073 18.00000000 15.23894941 -0.00464384 2.64505562

6 15.23894941 18.00000000 15.49664987 -0.00312879 1.66294304

7 15.49664987 18.00000000 15.65901465 -0.00203924 1.03687734

8 15.65901465 18.00000000 15.76026151 -0.00129919 0.64241865

9 15.76026151 18.00000000 15.82295968 -0.00081539 0.39624804

10 15.82295968 18.00000000 15.86161133 -0.00050686 0.24368048

11 15.86161133 18.00000000 15.88537074 -0.00031317 0.14956788

12 15.88537074 18.00000000 15.89994958 -0.00019277 0.09169114

13 15.89994958 18.00000000 15.90888523 -0.00011838 0.05616766

14 15.90888523 18.00000000 15.91435829 -0.00007259 0.03439069

15 15.91435829 18.00000000 15.91770910 -0.00004448 0.02105084

16 15.91770910 18.00000000 15.91976006 -0.00002723 0.01288311

17 15.91976006 18.00000000 15.92101521 -0.00001667 0.00788359

18 15.92101521 18.00000000 15.92178326 -0.00001020 0.00482390

19 15.92178326 18.00000000 15.92225322 -0.00000624 0.00295158

20 15.92225322 18.00000000 15.92254077 -0.00000382 0.00180593

21 15.92254077 18.00000000 15.92271671 -0.00000234 0.00110494

22 15.92271671 18.00000000 15.92282435 -0.00000143 0.00067604

23 15.92282435 18.00000000 15.92289021 -0.00000088 0.00041363

24 15.92289021 18.00000000 15.92293051 -0.00000054 0.00025307

25 15.92293051 18.00000000 15.92295516 -0.00000033 0.00015483

26 15.92295516 18.00000000 15.92297025 -0.00000020 0.00009473

x = 15.922970

1. **Graphs:**

a.

Fig 2.1: Solution using graphical method

c.

Fig 2.2: False position Error vs Iteration

**Comparison of error with bisection:**

Although false position typically needs fewer iteration than bisection to reach low relative approximation error, in this case, false position requires more iteration than the former. This is because of the nature of the function specified in the second problem. In this case we can see that by comparing figure 1(e ) and 2 (b).

Since xr replaces one of upper or lower bound as opposed to halving in bisection, one of the bounds remain the same and it takes longer time to arrive at a solution with low error.