**CSE 3212: Numerical Methods Lab**

**Name of the assignment:**

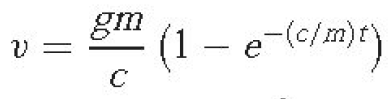
Finding roots of different equations using bisection, false position, Newton Raphson and secant method

**Credits:   
1. Tanjid Hasan Tonmoy**

**2. Sakib Hasan**

**Problem statement 1:**

The velocity v of a falling parachutist is given by



where g = 9.8 m/s2. For a parachutist with a drag coefficient c = 15 kg/s, compute the mass m

so that the velocity is v = 35 m/s at t = 9 s.

By using, (a) bisection and (b) false position.

For (a) and (b) use initial guesses from the ***user input,*** and iterate until the approximate error

falls below ***user specified tolerance.***

At first, print the value of m and f(m) from user lower input and user upper input, increasing by

0.1. Then, 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.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| iteration | Upper value | Lower value | Xm | f(Xm) | Relative approximate  error |

Lastly,

Draw six graphs from above solution.

In graph 1: the graph of x m and relative approximation error (bisection).

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

In graph 3: the graph of x r and relative approximation error (false position).

In graph 4: the graph of no of iteration and relative approximation error (false position).

In graph 5: Compare the relative approximate error with respect to number of iterations between the bisection method and false position method. For comparison, you need to draw the graph of number of iteration and relative approximation error.

In graph 6: Compare the relative approximate error with respect to *x* between the bisection

method and false position method. For comparison, you need to draw the graph of *x* and

relative approximation error.

**Solution 1:**

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#include <limits.h>

double error\_tolerance;

double func(double m) {

double v=35, g=9.8,c=15,t=9;

return (v - ( ((g\*m)/c) \* (1 - exp((-1\*c/m)\*t)) ));

}

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

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

}

void bisection(double xl, double xu) {

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

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

return;

}

printf("\titer\tupper\t\tlower\t\txr\t\tf(xr)\t\tREA\n\n");

double xr = 0.0;

int iteration = 0;

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

xr = (xl + xu) / 2;

if(func(xr) == 0.0) {

break;

} else {

printf("%8d\t%.8lf\t%0.8lf\t%0.8lf\t%0.8lf\t%0.8lf\n", ++iteration, xl, xu, xr, func(xr), get\_error\_bisection(xl, xu));

}

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

xu = xr;

} else {

xl = xr;

}

}

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

}

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;

}

printf("\titer\tupper\t\tlower\t\txr\t\tf(xr)\t\tREA\n\n");

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, xu, xl, 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("prob1\_0.csv", "w", stdout);

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);

printf("\n\n");

for(double i =xl; i<=xu; i+=0.1){

printf("%0.6lf\t%0.6lf\n",i,func(i));

}

printf("\n\n");

bisection(xl, xu); //prob 1(a)

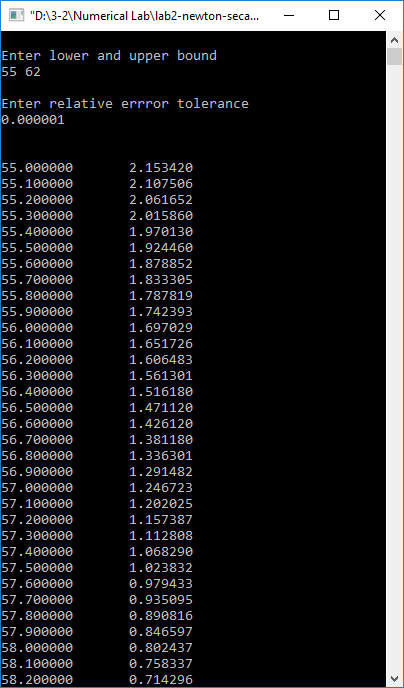
false\_position(xl,xu); //prob 1(b)

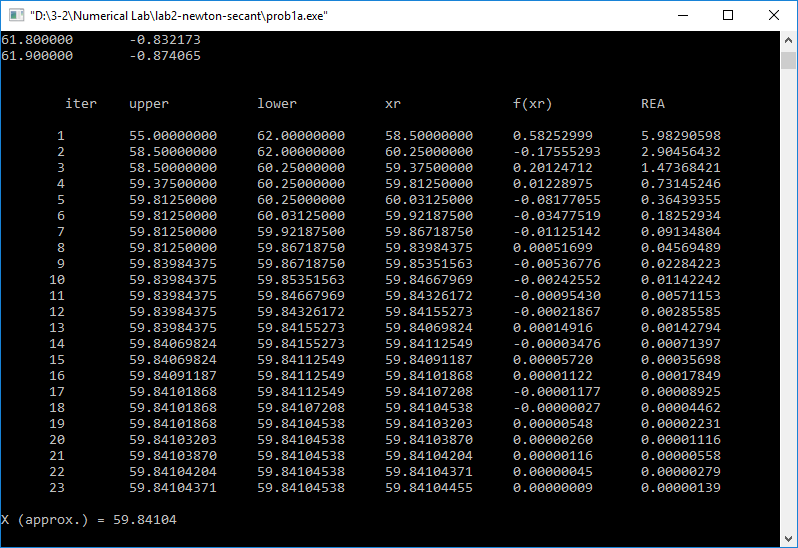
return 0;

}

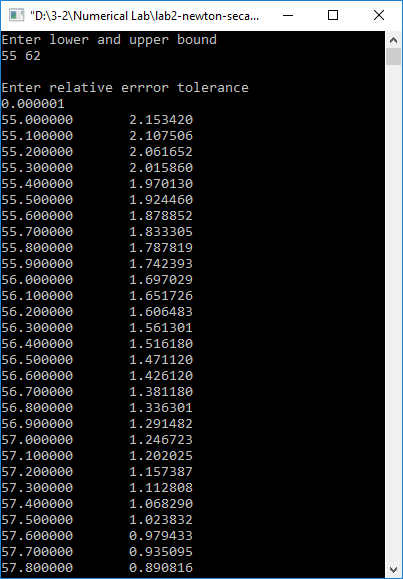
**Sample Input output:**

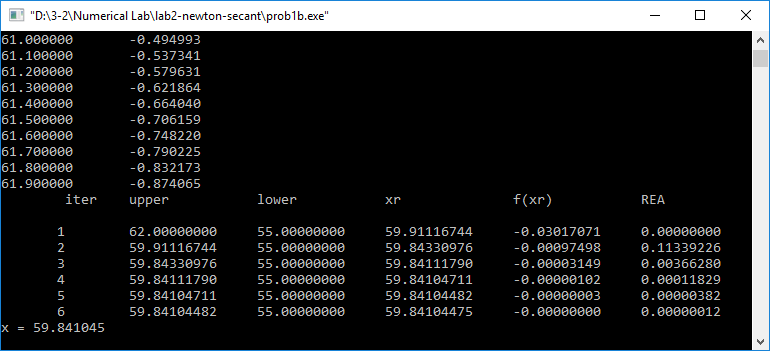
**(a):**





**(b):**





**Graphs:**

Fig: Relative approx. Error vs Iteration for bisection method

Fig: Relative approx. Error vs Xm for bisection method

Fig: Relative approx. Error vs Iteration for false position method

Fig: Relative approx. Error vs Xr for false position method

Fig: Comparison between bisection and false position method for Error vs iteration

Fig: Comparison between bisection and false position method for Error vs X

**Problem Statement 2**

Write a single program (source **file name must be** problem2. extension) to solve the following

(a) Use the Newton-Raphson method to determine a root of f (x) = −x 2 + 1.8x + 2.5 using x 0 = 5.

Perform the computation until εa is less than user specified tolerance.

Also perform an error check of your final answer as the following table.

(b) Use the Newton-Raphson method to find the root of

f (x) = e−0.5x (4 − x) − 2

Employ initial guesses of (i) 2, (ii) 6, and (iii) 8.

Explain your results.

You also need to print the following table in your console view.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| iteration | xi | f(xi ) | f’(xi ) | Relative approximate  error |

**Solution:**

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#include <limits.h>

double error\_tolerance;

double f\_a(double x){

return -x\*x+1.8\*x+2.5;

}

double f\_d\_a(double x){

    return -2\*x + 1.8;

}

double f\_b(double x){

return (4-x)\*exp(-0.5\*x)- 2;

}

double f\_d\_b(double x){

    return ((x-6)\*exp(-0.5\*x))/2;;

}

double get\_error(double current, double prev){

return fabs((current - prev) / current) \* 100;

}

void newton\_raphson(double x, double error\_tolerance)

{

double prev\_x, error = 0;

int i = 1;

while (i ==1 || error >= error\_tolerance)

{

printf("%5d %5.5lf %5.5lf %5.5lf %5.5lf\n", i++, x, f(x), f\_d(x), error);

if(f(x) == 0) break;

prev\_x = x;

x = x - f(x)/f\_d(x);

error = get\_error(x, prev\_x);

}

printf("\n\nThe approx value of the root is : %lf\n\n", x);

}

int main(){

    double x;

printf("\nEnter guess\n");

scanf("%lf", &x);

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

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

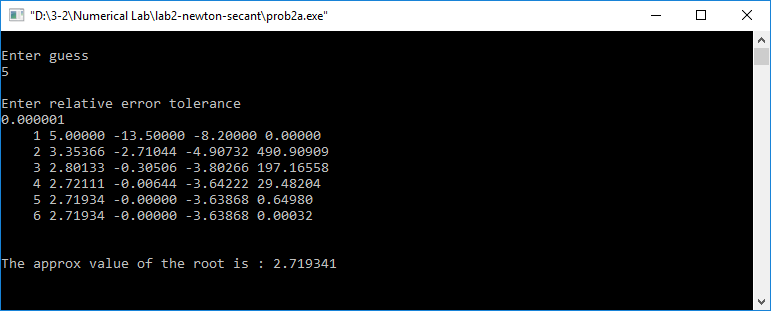
    newton\_raphson(x, error\_tolerance);

    return 0;

}

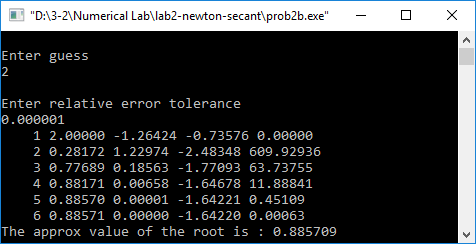
**Sample input output:**

**(a)**

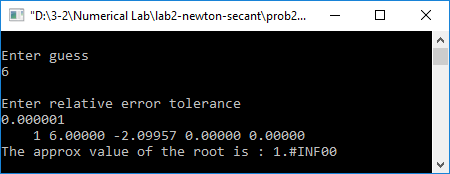


**(b).**

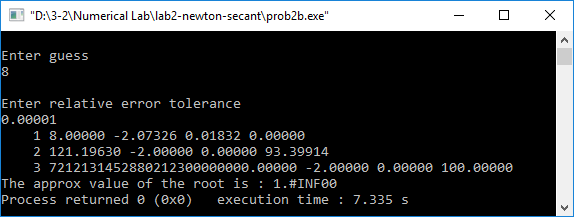
Using the initial guess of 2:



Using the initial guess of 6:



Using the initial guess of 8:



**Explanation:**

Using this method, we get valid roots for all the given guess point2 but do not get expected result for 6 and 8. For initial guess 6 we do not get proper root since f’(6) = 0, and Newton-Raphson method uses f(x)/f’(x) to get the next approximation. As we can see from the plot of the function, the slope of f(x) at 8 is very close to zero. So we do not get approximation using this method.

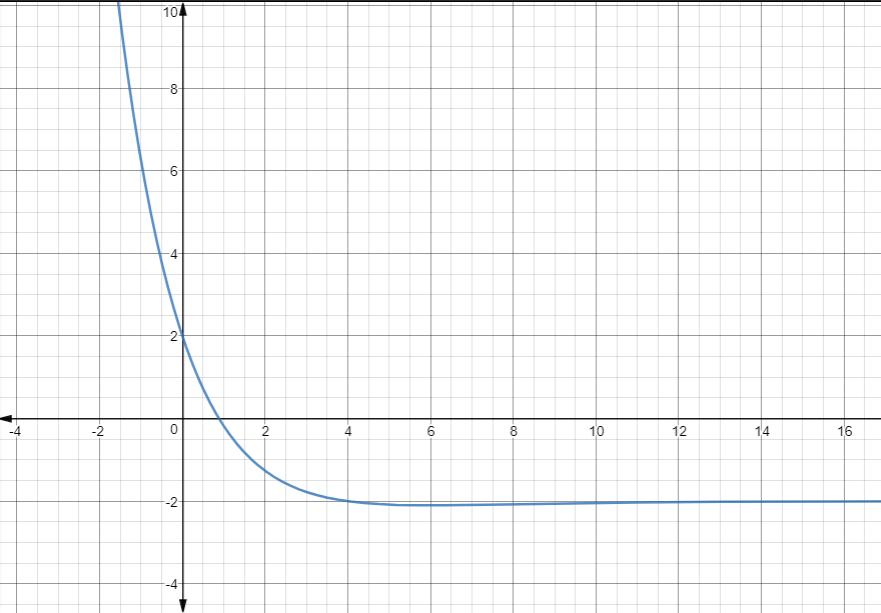


Fig: Graph of the function

**Problem Statement 3**

Write a single program (source **file name must be** problem3. extension) to solve the following

(a) Consider following easily differentiable function,

f (x) = 8 sin(x)e–x − 1

Use the secant method, when initial guesses of xi–1 = 0.5 and xi = 0.4 with user specified

tolerance. You also need to print the following table in your console view.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| iteration | Upper value | Lower value | Xm | f(Xm) | Relative approximate  error |

**Solution:**

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#include <limits.h>

double error\_tolerance = 0.00001;

double f(double x){

return (8\*sin(x)\*exp(-x))-1;

}

double get\_error(double current, double prev){

return fabs((current - prev) / current) \* 100;

}

void secant(double x\_prev, double x\_cur, double error\_tolerance)

{

double x\_next,error;

int i =0;

while(get\_error(x\_cur,x\_prev) >= error\_tolerance){

x\_next = x\_cur - (f(x\_cur)\*(x\_cur-x\_prev))/(f(x\_cur) - f(x\_prev));

printf("%5d %5.5lf %5.5lf %5.5lf %5.5lf %5.5lf\n", i++, x\_prev, x\_cur, x\_next, f(x\_cur), get\_error(x\_next, x\_cur));

x\_prev = x\_cur;

x\_cur = x\_next;

if(f(x\_cur) == 0) break;

}

printf("apprx root = %lf", x\_cur);

}

int main(){

    double x1,x2;

printf("\nEnter guesses\n");

scanf("%lf %lf", &x1, &x2);

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

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

    secant(x1,x2, error\_tolerance);

    return 0;

}

**Sample input output:**

