GEBZE TEKNİK ÜNİVERSİTESİ

DEPARTMENT OF ELECTRONIC ENGINEERING

MATH214 NUMERICAL METHODS

2020 – 2021 FALL TERM

Project 1

1801022001	Upload date
Ömer Sarımeşe	28.10.2020

Introduction

The purpose of this report is finding the closest point where electric field intensity will be zero, with given information of case. There is a formula that help us to solve the problem. $E_{(x)} = \frac{1}{4\pi\varepsilon_0}\sum_{k=1}^{N}\frac{q_k(r-r_k)}{|r-r_k|}$ [V/m] is the formula of electic field intensity. ε_o is equal to $\frac{1}{36\pi} \times 10^{-9}$. When the given values which from the question, be placed in formula there can be computable formula will outcome. Then we can find the approximate root of equation by using some sort of methods. After all of them, difference between methods can will be seen clearly by results although there will be a concluding section that includes evaluations of report. The results will be found in Matlab which is a programming platform specifially for engineers.

Used Functions

$$f(x) = \frac{1}{4\pi\varepsilon_0}\left[\frac{13(x+7)}{|x+7|} + \frac{9(x+4)}{|x+4|} + \frac{5(x-11)}{|x-11|} + \frac{3(x-11)}{|x-11|}\right]$$

$$f'(x) = \frac{1}{4\pi\varepsilon_0}\left[-\frac{9(x-15)^2}{|x-15|^5} - \frac{15(x-11)^2}{|x-11|^5} - \frac{27(x+4)^2}{|x+4|^5} - \frac{39(x+7)^2}{|x+7|^5} + \frac{3}{|x-15|^3} + \frac{5}{|x-11|^3} + \frac{9}{|x+4|^3} + \frac{13}{|x+7|^3}\right]$$

Section I: Bisection Method

There should be a continuous function at interval [a,b]. First step is the calculate midpoint of the interval. After that calculate the function value of midpoint. Then change boundaries to different sign one and that value. This steps repeating until stopping condition provided. If $\frac{|Upper\ boundry - Lower\ boundry|}{2^{current\ iteration\ number}}$ formula's result is smaller than our tolerance value, then its stopping time fort he algorithm. Explanation of matlab code on below, will illustrate better.

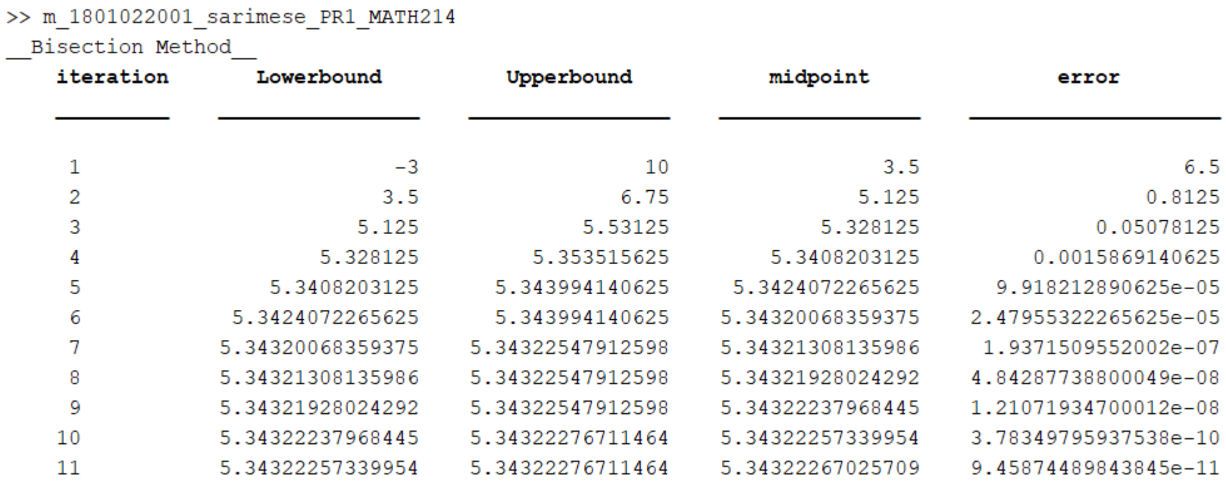
```
%-----------------------Bisection---------------------

UB=10;                        define upper bound value.
LB=-3;                        define lower bound value.
n=1;                          set iteration variable to 1.
temp=1;                       set temporary variable to 1.
a_1=zeros(20,5);              double array for saving values.

while(temp>tol && temp~=0)    if not enough close to result
                                    continue to compute.

    a_1(n,1)=n;               assign values to array to saving
    a_1(n,2)=LB;                them for creating table and
    a_1(n,3)=UB;                         graphics.


    mid=LB+(UB-LB)/2;         finding midpoint of interval.
    temp=abs(UB-LB)/(2^n);    finding how much close the
                                    boundaries.
    if(f(UB)*f(mid)>0)        finding have same sign bound.
        UB=mid;               set midpoint new upperbound.
    else
        LB=mid;               set midpoint new lowerbound.
    n=n+1;                    pass the next iteration value.
    a_1(n-1,4)=mid;           saving mid value to array.
    a_1(n-1,5)=temp;          saving temp value to array.
    end                       end of the if-else cond.
end                           end of the while loop.

%---------------TABLE PART--------------

disp("__Bisection Method__");
% creating & arranging & displaying table
T=array2table(a_1);
T.Properties.VariableNames={'iteration' 'Lowerbound'
'Upperbound' 'midpoint' 'error'};
disp(T);
```

Explanation of table part of code:

T=array2table(a_1) : this function converts the a_1 array to a table class.

T.Properties.VariableNames : this line is naming the columns of table to present data more clear and understandable.

disp() : for displaying output which is table of results in that case.



```
>> m_1801022001_sarimese_PR1_MATH214
__Bisection Method__
    iteration        Lowerbound          Upperbound           midpoint                  error
    _________    ________________    ________________    ________________    ____________________

        1                      -3                  10                 3.5                     6.5
        2                     3.5                6.75               5.125                  0.8125
        3                   5.125             5.53125            5.328125              0.05078125
        4                5.328125         5.353515625        5.3408203125         0.0015869140625
        5            5.3408203125      5.343994140625     5.3424072265625      9.918212890625e-05
        6         5.3424072265625      5.343994140625    5.34320068359375    2.47955322265625e-05
        7        5.34320068359375    5.34322547912598    5.34321308135986     1.9371509552002e-07
        8        5.34321308135986    5.34322547912598    5.34321928024292    4.84287738800049e-08
        9        5.34321928024292    5.34322547912598    5.34322237968445    1.21071934700012e-08
       10        5.34322237968445    5.34322276711464    5.34322257339954    3.78349795937538e-10
       11        5.34322257339954    5.34322276711464    5.34322267025709    9.45874489843845e-11
```

Table-1 Bisection Method Matlab Output Table

Approximate root is midpoint of eleventh iteration on the Table-1 which is located above. Method converges a solution. Code is working perfectly.

Section II: Newton-Raphson Method

This method include derivative to do basic algebra operations that find x-intercepts tangent lines repetitively. That lines getting closer to actual root. Newton method convergence very fast. But there might be a problem when derivatives cannot found. $x_n = x_{n-1} - \frac{f(x_{n-1})}{f'(x_{n-1})}$ is the formula for finding more accurate root. Program continue to compute new x_n values until reached to enough tolerance. If gap between the roots is less than tolerance value, then last root is the most accurate one. Matlab code which located below will explain.

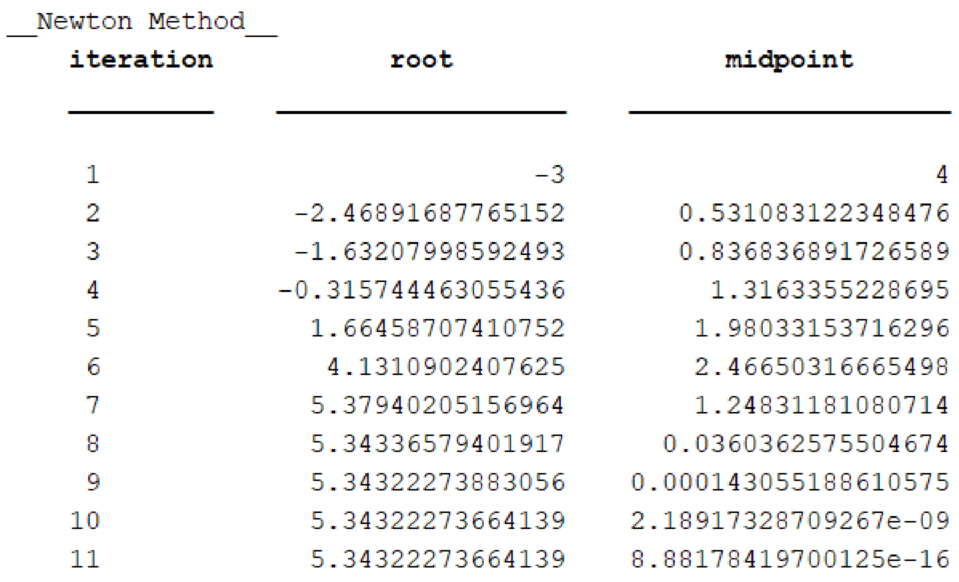
```
%---------------------------newton--------------------------

prev=-3;                     prev is $x_{n-1}$ ,and set -3.
next=1;                      next is $x_n$ , and set something(1).
i=1;                         iteration number variable.
temp=1;                      temp variable is error value
                                 for method and set 1.

a_2=zeros(20,3);             double array for saving values.
a_2(1,2)=prev;               assigning the first x value.
a_2(1,3)=abs(next-prev);     assigning the first error value.

while(tol<temp)              if not enough to close, continue
                                     to compute.

   a_2(i,1)=i;
   next=(prev)-(f(prev)/fd(prev));  finding $x_n$ value.
   temp=abs((next)-(prev));         finding error value.
   prev=next;                       set $x_n$ to new $x_{n-1}$.
   i=i+1;                           pass the next iteration.
   a_2(i,2)=next;                   save $x_n$ value.
   a_2(i,3)=temp;                   save error value.
End                                 ending the while loop.
a_2(i,1)=i;                         save iteration value.

%---------------TABLE PART--------------

disp("__Newton Method__");
% creating & arranging & displaying table
T=array2table(a_2);
T.Properties.VariableNames={'iteration' 'root' 'error'};
disp(T);
```

Explanation of table part of code:

T=array2table(a_2) : this function converts the a_2 array to a table class.

T.Properties.VariableNames : this line is naming the columns of table to present data more clear and understandable.

disp() : for displaying output which is table of results in that case.



```
__Newton Method__
    iteration            root                       midpoint

    _________    __________________     _____________________


     1                           -3                          4
     2             -2.4689168776515 2       0.531083122348476
     3             -1.63207998592493       0.836836891726589
     4            -0.315744463055436         1.3163355228695
     5              1.66458707410752        1.98033153716296
     6               4.1310902407625        2.46650316665498
     7              5.37940205156964        1.24831181080714
     8              5.34336579401917      0.0360362575504674
     9              5.34322273883056   0.000143055188610575
    10              5.34322273664139    2.18917328709267e-09
    11              5.34322273664139    8.88178419700125e-16
```

Table-2   Newton Method Matlab Output Table

Approximate root is midpoint of eleventh iteration on the Table-2 above. This method is converges more faster than bisection method although the start point was not a close one.

Section III: Secant Method

This method requires two initial values unlikely Newton method. Secant method can might not always converge because it does not require close interval. It is just using the function's result not derivative of function, it will be an advantage in basic problems.

The formula that give us the next potential root of the method is $x_{n+1} = x_n - \frac{f(x_n).[x_{n-1} - x_n]}{f(x_{n-1}) - f(x_n)}$. Stopping condition is nearly same as previous methods. To be mention again stopping condition is when gap between the terms of x is less than our tolerance value, we will stop computing. Last x value is approximate root. Looking into the code will give us much more visulation.

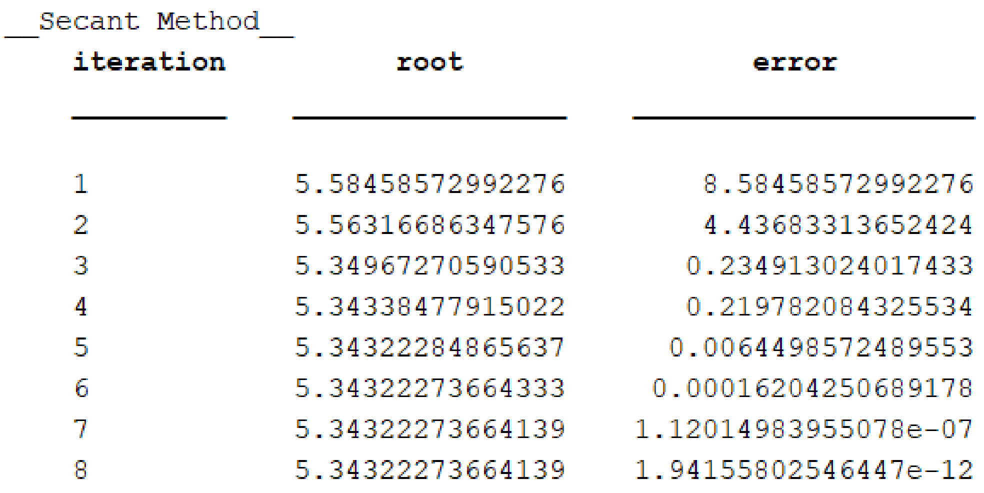
```
%---------------------------secant---------------------------

prev=-3;                  prev is $x_{n-1}$ value.
mid=10;                   mid is $x_n$ value.
next=0;                   next is $x_{n+1}$ value.
temp=1;                   temp is error value.
i=1;                      set iteration value to 1.
a_3=zeros(20,3);          array for saving values.

while(temp>tol)           if not enough close to tolerance
                           value, continue to compute.

    next=mid-((f(mid)*(prev-mid))/(f(prev)-f(mid)));  formula.

    temp=abs(next-prev);        finding error value.

    prev=mid;                   set $x_n$ to new $x_{n-1}$ value.
    mid=next;                   set $x_{n+1}$ to new $x_n$ value.

    a_3(i,1)=i;                 saving iteration number.
    a_3(i,2)=next;              saving $x_{n+1}$ value.
    a_3(i,3)=temp;              saving error value.
    i=i+1;                      increasing iteration value.
End                             ending the while loop.


%---------------TABLE PART--------------

disp("__Secant Method__");
% creating & arranging & displaying table
T=array2table(a_3);
T.Properties.VariableNames={'iteration' 'root' 'error'};
disp(T);
```

Explanation of table part of code:

They are here just for the show table of result. These Functions same as previous method's codes.

Root values actually means x_{n+1} .

Error value explains how much the x 's closing.



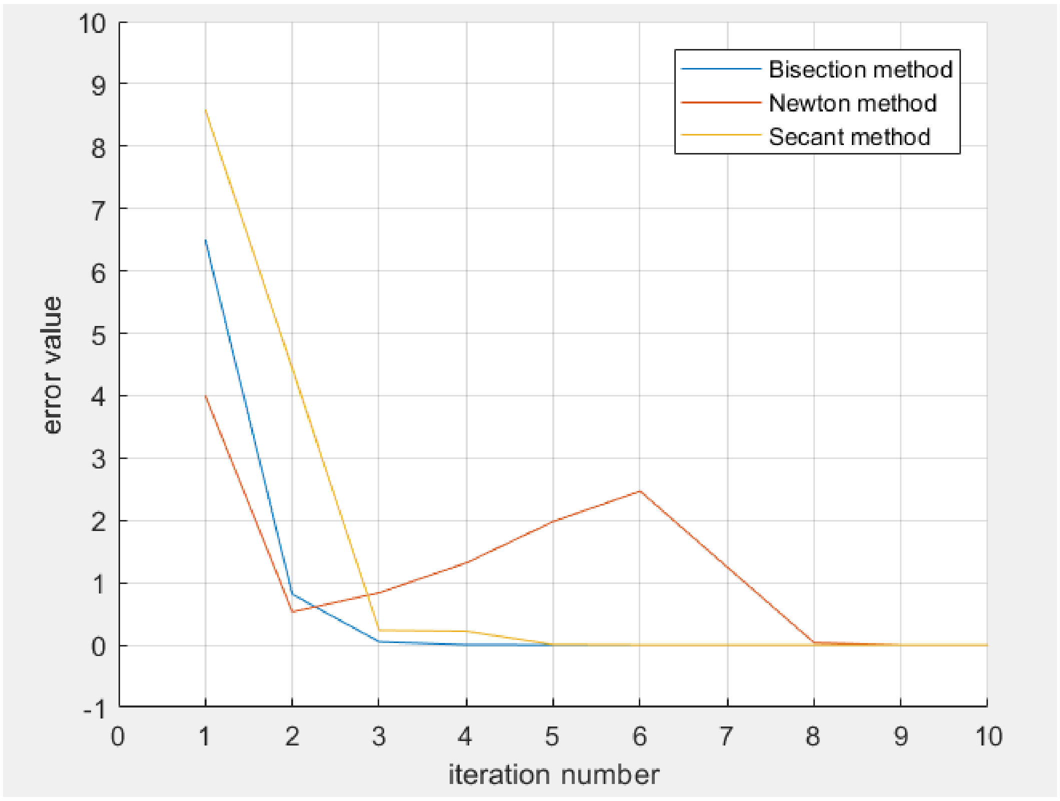
```
__Secant Method__
   iteration          root                    error

   _________     ________________    ____________________

   1             5.58458572992276        8.58458572992276
   2             5.56316686347576        4.43683313652424
   3             5.34967270590533       0.234913024017433
   4             5.34338477915022       0.219782084325534
   5             5.34322284865637      0.0064498572489553
   6             5.34322273664333     0.00016204250689178
   7             5.34322273664139    1.12014983955078e-07
   8             5.34322273664139    1.94155802546447e-12
```

Table-3 Secant Method Matlab Output Table

Interestingly this method found root values more accurately in early iterations. Last root values are same in fourteen digits as can be seen by Table-3. I think the difference between Secant and Newton methods is Newton method uses one point's tangent line, Secant method uses two points' secant line. The remaining procedures are looking same. We basically use two points for the compute slope of the line However in Newton method we use derivative to find that.

Section IV: Concluding

Results show us that all methods are converges at different rates but all of them are successfully converges. The root is approximate 5.343227 in all calculations. Every iteration closes the gap between the roots and this gap is our error value. We use 10^{-10} for tolerance value. We compare error and tolerance to have most accurate answer to problem. The methods' error amount by iteration is shown Graph-1 which is below.



Graph-1   Matlab comparison graph

Quick Matlab Code Explanation

```
%--------------------------graphs------------------------

hold on;             retain current plot when adding new plots.
grid on;             show grid on graph.

p=plot(a_1(:,5));   plot double array’s last column.(bisection)
p=plot(a_2(:,3));   plot double array’s last column(newton).
p=plot(a_3(:,3));   plot error of secant method.(same as above)

legend('Bisection method','Newton method','Secant method');
xlabel('iteration number');
ylabel('error value');
axis([0 10 -1 10]);
```

Remaining functions for naming lines, axis and scaling the graph.

In my observation all methods converges very fast at early iterations. Then getting closer to the actual root. But in Newton method after the second iteration, the root is started getting away from correct side. After some iterations it found true way to reach the root. Bisection method close the gap looking to different sign of borders; Newton method uses tangent line to a single point and takes x-intercept point as a root everytime; Secant method connects two point with a secant line. Newton and Secant methods are pretty same like Newton and Secant methods are an open interval method unlikely Bisection method. Newton method may not converges sometimes. Bisection method guaranteed to converges but slowly. So I conclude that Newton method is much sensiable than other to use on difficulty cases.

In addition, i do not think increasing tolerance value doesnt change the root that much but maybe if problem needs more accurate solution in scale of larger digits, then tolerance should be large enough to not make a critical mistake.

MATLAB CODE

```
clear all;
close all;

format long; % reason why is be able to display more
decimal digits
tol=10^(-10); % tolerance value

%-----------------------Bisection---------------------

UB=10; % Upper Bound Value
LB=-3; % Lower Bound Value
n=1; % iteration variable
temp=1; % temporary value
a_1=zeros(20,5); % space for saving values

while(temp>tol && temp~=0)
    % saving LB,UB values before creating table...
    a_1(n,1)=n;
    a_1(n,2)=LB;
    a_1(n,3)=UB;

    mid=LB+(UB-LB)/2;
    temp=abs(UB-LB)/(2^n);
    if(f(UB)*f(mid)>0)
        UB=mid;
    else
        LB=mid;
    n=n+1;
    % saving res,temp values before creating table...
    a_1(n-1,4)=mid;
    a_1(n-1,5)=temp;
    end
end

disp("__Bisection Method__");
% creating & arranging & displaying table
T=array2table(a_1);
T.Properties.VariableNames={'iteration' 'Lowerbound'
'Upperbound' 'midpoint' 'error'};
disp(T);

%---------------------------newton------------------------
--

prev=-3;
next=1;
```

```
i=1; % iteration variable
temp=1; % temporary value
a_2=zeros(20,3); % space for saving values
a_2(1,2)=prev;
a_2(1,3)=abs(next-prev);

while(tol<temp)
    a_2(i,1)=i;
    next=(prev)-(f(prev)/fd(prev));
    temp=abs((next)-(prev));
    prev=next;
    i=i+1;
    a_2(i,2)=next;
    a_2(i,3)=temp;
end
a_2(i,1)=i;

disp("__Newton Method__");
% creating & arranging & displaying table
T=array2table(a_2);
T.Properties.VariableNames={'iteration' 'root' 'error'};
disp(T);
%---------------------------secant------------------------
--

prev=-3;
mid=10;
next=0;
temp=1; % temporary value
i=1; % iteration variable
a_3=zeros(20,3); % space for saving values

while(temp>tol)
    next=mid-((f(mid)*(prev-mid))/(f(prev)-f(mid)));
    temp=abs(next-prev);
    prev=mid;
    mid=next;
    a_3(i,1)=i;
    a_3(i,2)=next;
    a_3(i,3)=temp;
    i=i+1;
end

disp("__Secant Method__");
% creating & arranging & displaying table
T=array2table(a_3);
T.Properties.VariableNames={'iteration' 'root' 'error'};
```

```
disp(T);

%--------------------------graphs-------------------------
--
% i didnt find any other way to plot double array values.
% i could not use loglog() & semilog().
hold on;
grid on;
p=plot(a_1(:,5));
p=plot(a_2(:,3));
p=plot(a_3(:,3));
legend('Bisection method','Newton method','Secant
method');
xlabel('iteration number');
ylabel('error value');
axis([0 10 -1 10]);

%-------------------------functions-----------------------
--
function val=f(x)
val=(1/(4*pi*(1/36*pi)*10^(-
9)))*((13*(x+7)/(abs((x+7)^3)))+(9*(x+4)/(abs((x+4)^3)))+(
5*(x-11)/(abs((x-11)^3)))+(3*(x-15)/(abs((x-15)^3))));
end

function deg=fd(x)
deg=(1/(4*pi*(1/36*pi)*10^(-9)))*((3/((abs(x-
15))^3))+(5/((abs(x-
11))^3))+(9/((abs(x+4))^3))+(13/((abs(x+7))^3))-((9*(x-
15)^2)/((abs(x-15))^5))-((15*(x-11)^2)/((abs(x-11))^5))-
((27*(x+4)^2)/((abs(x+4))^5))-
((39*(x+7)^2)/((abs(x+7))^5)));
end
```