ELECTRONIC ENGINEERING DEPARTMENT

MATH 214 NUMERICAL METHODS

2020 – 2021 FALL

PROJECT 1

Student ID :1901022039

Name Surname :Alperen Arslan

Preperation date: 28.10.2020

function definition

In this part, we define the function that we want to find root and assign tolerance as 10^-10 like asked.

```
clear all
format long
f= @(r) (1/(4*pi*(1/(36*pi)*10^-9)))*(13*(r--7)/(abs(r--7)^3)+9*(r--4)/(abs(r--4)^3)+6*(r-
11)/(abs(r-11)^3)+3*(r-14)/(abs(r-14)^3));
tolerance=10^-10;
```

bisection method

For applying bisection method in matlab, first a range must be specified, in this case [-3,10] range has been defined as asked. Then, a middle point should be determined and algorithm should check if f(mid)*f(min)<0. If this statement is not true, then that means there aren't any root between [min-mid] interval so algorithm assigns min to mid and the new interval is going to be [mid-max]. But if this statement is true, then there has to be a root between [min-mid] interval so algorithm assigns max to mid. This goes on until (max - min)/2<tolerance which means algorithm is so close the real number that we can tolerate the difference.

And two arrays must be defined for storing the iteration and error data, later we going to use these data for plotting error graph.

```
min=-3;
max=10;
fprintf("bisection method\nresult             iteration    error\n");
for i=1:100
    mid= min+(max-min)/2;
    if f(min)*f(mid)<0
       max=mid;
    else
```

```
        min=mid;
    end
    a1(1,i)=i;
    bisection(1,i)=abs(0-f(mid));
    fprintf("%1.15f      %1.f    %1.15f\n",mid,i,abs(0-f(mid)));
if (max-min)/2<tolerance
    break
end
end
fprintf("the final result that found by bisection method is %1.15f at %1.f
iterations\n\n",mid,i);
```

```
bisection method
result             iteration    error
3.500000000000000      1    1296326530.612244367599487
6.750000000000000      2    2183531183.439598083496094
5.125000000000000      3    138673788.299402296543121
4.312500000000000      4    591369739.527054429054260
4.718750000000000      5    235408314.317859262228012
4.921875000000000      6    51465266.491284213960171
5.023437500000000      7    42716636.336356408894062
4.972656250000000      8    4581603.318632887676358
4.998046875000000      9    19013899.252542551606894
4.985351562500000      10    7202969.983223498798907
4.979003906250000      11    1307416.956199861131608
4.975830078125000      12    1637906.270845139399171
4.977416992187500      13    165448.367309299617773
4.978210449218750      14    570933.312210066011176
4.977813720703125      15    202729.733735550194979
4.977615356445313      16    18637.499390107557701
4.977516174316406      17    73406.229808295480325
4.977565765380859      18    27384.564184773640591
4.977590560913086      19    4373.582143009369247
4.977602958679199      20    7131.946186560156093
4.977596759796143      21    1379.178912512546503
4.977593660354614      22    1497.202392751473326
4.977595210075378      23    59.011934089303608
4.977595984935760      24    660.083440531811220
4.977595597505569      25    300.535741168395077
4.977595403790474      26    120.761900229693353
4.977595306932926      27    30.874982445694421
4.977595258504152      28    14.068475696904503
4.977595282718539      29    8.403252843569575
4.977595270611346      30    2.832611832592757
4.977595276664943      31    2.785320661613522
4.977595273638144      32    0.023645335689437
4.977595275151543      33    1.380837506836929
4.977595274394844      34    0.678596491499039
4.977595274016494      35    0.327475234429553
4.977595273827319      36    0.151914668344855
the final result that found by bisection method is 4.977595273827319 at 36 iterations
```

computing derivative

```
h=10^-10;
f_turev= @(x) (f(x+h)-f(x))/h; %derivative can be found by using limit definition
```

the limit definition of derivative formula is:

$$f'(x)=\lim_{h\to 0}\frac{f(x+h)-f(x)}{h}$$

so for calculating derivatives in matlab, a function must be defined and assigned to this formula.

Since h approaches to zero, it can be thought of as 10^-10 which is very close to zero therefore derivative can be found from this method.

newton's method

For applying newton's method, first an initial point must be defined, in this case 6.5 is used since its middle point of [-3,10] interval. Then, inside of a loop, p variable is defined and assigned to newton's method's formula which is:

$$X_{n+1} = X_n - \frac{f(X_n)}{f'(X_n)}$$

And then, the algorithm checks whether the error is smaller than tolerance, if it's true, the result is found. If not, the algorithm assigns p to p0 and uses the formula until the error is less than tolerance.

And two arrays must be defined for storing the iteration and error data, later we going to use these data for plotting error graph.

```
p0=6.5;
fprintf("newton's method\nresult             iteration     error\n");
for i=1:100
p=p0-f(p0)/f_turev(p0);
a2(1,i)=p;
newton(1,i)=abs(0-f(p));
fprintf("%1.15f      %1.f    %1.15f\n",p,i,abs(0-f(p)));
if abs(p-p0)<tolerance
    break
end
p0=p;
end
fprintf("the result that found by newton method is %1.15f at %1.f iterations\n\n",p,i);
```

```
newton's method
result             iteration     error
5.356766396098938      1    365619630.205889284610748
4.993463240318037      2    14746502.113318217918277
4.977617626023528      3    20743.748697323040687
4.977595273918284      4    0.236333889958029
4.977595273663623      5    0.000000874300632
4.977595273663622      6    0.000000749400542
the result that found by newton method is 4.977595273663622 at 6 iterations
```

secant method

For applying secant method, at the first x1 and x2 points must be defined, in this question we started our initial points as -3 and 10 as asked. And then, we apply secant method formula:

$$x_n = x_{n-1} - f(x_{n-1})\frac{x_{n-1} - x_{n-2}}{f(x_{n-1}) - f(x_{n-2})}$$

If this process repeats until the error less than tolerance, then we can get the result.

And two arrays must be defined for storing the iteration and error data, later we going to use these data for plotting error graph.

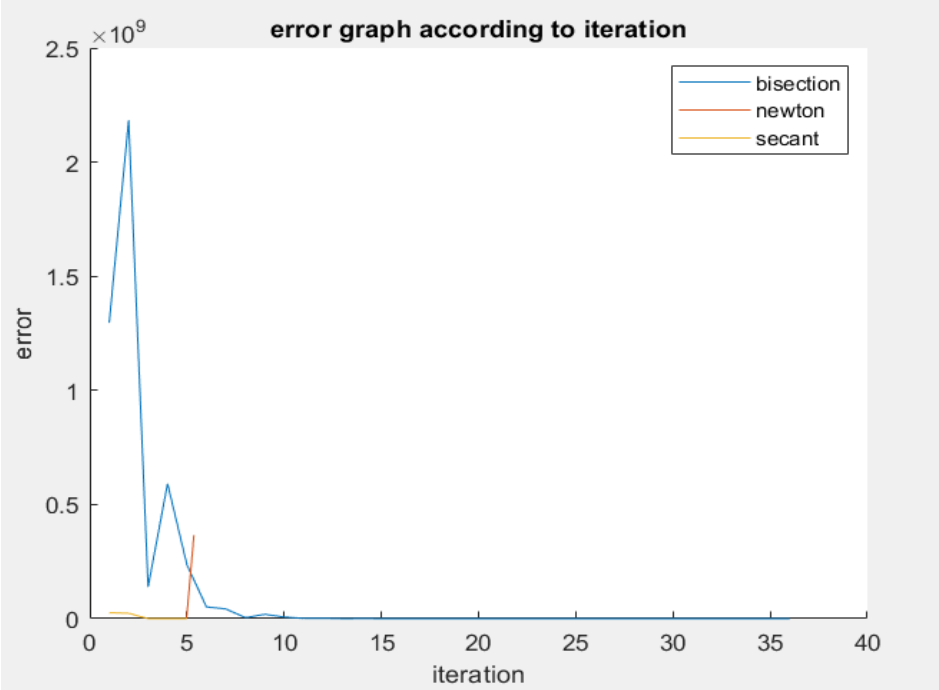
```
x1=-3;x2=10;
q0=f(x1);
q1=f(x2);
fprintf("secant method\nresult             iteration     error\n");
for i=1:100
    p=x2-q1*(x2-x1)/(q1-q0);
    fprintf("%1.15f      %1.f    %1.15f\n",p,i,abs(0-f(p)));
    if abs(p-x2)<tolerance
        break
    end
    a3(1,i)=i;
    secand(1,i)=abs(0-f(p));
    x1=x2;
    q0=q1;
    x2=p;
    q1=f(p);

end
fprintf("the result that found by secant method is %1.15f at %1.f iterations\n",p,i);
```

```
secant method
result             iteration     error
5.005340536006132      1    25811563.748005244880915
5.002989851133218      2    23619762.745349187403917
4.977657889463321      3    58109.791484059482173
4.977595413582982      4    129.849662834635723
4.977595273664387      5    0.000709245162600
4.977595273663623      6    0.000000874300632
the result that found by secant method is 4.977595273663623 at 6 iterations
```

ploting the graph

```
xlabel("iteration");
ylabel("error");
title("error graph according to iteration");
hold on
loglog(a1,bisection);
loglog(a2,newton);
loglog(a3,secand);
legend("bisection","newton","secant");
hold off
```



conclusion

In the end, if we compare these three methods we can say that newton's method is the best method because the newton's method gave us the most accurate result with less iterations.

The secant method and the newton's method both have six iteration but error is slightly smaller in newton's method therefore we can say that secant method is our second best method.

And the bisection method is our most inefficient and inaccurate method by far. It has 36 iterations and error is very massive comparing the others.

If we change the tolerance 10^-15, the results are:

```
bisection method
result             iteration    error
3.500000000000000      1    1296326530.612244367599487
6.750000000000000      2    2183531183.439598083496094
5.125000000000000      3    138673788.299402296543121
4.312500000000000      4    591369739.527054429054260
4.718750000000000      5    235408314.317859262228012
```

```
4.921875000000000      6    51465266.491284213960171
5.023437500000000      7    42716636.336356408894062
4.972656250000000      8    4581603.318632887676358
4.998046875000000      9    19013899.252542551606894
4.985351562500000      10    7202969.983223498798907
4.979003906250000      11    1307416.956199861131608
4.975830078125000      12    1637906.270845139399171
4.977416992187500      13    165448.367309299617773
4.978210449218750      14    570933.312210066011176
4.977813720703125      15    202729.733735550194979
4.977615356445313      16    18637.499390107557701
4.977516174316406      17    73406.229808295480325
4.977565765380859      18    27384.564184773640591
4.977590560913086      19    4373.582143009369247
4.977602958679199      20    7131.946186560156093
4.977596759796143      21    1379.178912512546503
4.977593660354614      22    1497.202392751473326
4.977595210075378      23    59.011934089303608
4.977595984935760      24    660.083440531811220
4.977595597505569      25    300.535741168395077
4.977595403790474      26    120.761900229693353
4.977595306932926      27    30.874982445694421
4.977595258504152      28    14.068475696904503
4.977595282718539      29    8.403252843569575
4.977595270611346      30    2.832611832592757
4.977595276664943      31    2.785320661613522
4.977595273638144      32    0.023645335689437
4.977595275151543      33    1.380837506836929
4.977595274394844      34    0.678596491499039
4.977595274016494      35    0.327475234429553
4.977595273827319      36    0.151914668344855
4.977595273732732      37    0.064134635102686
4.977595273685438      38    0.020244556031557
4.977595273661791      39    0.001700389828940
4.977595273673614      40    0.009272332901489
4.977595273667703      41    0.003785971536274
4.977595273664747      42    0.001042915753757
4.977595273663269      43    0.000328487237411
4.977595273664008      44    0.000356839557902
4.977595273663638      45    0.000013926360065
4.977595273663454      46    0.000157249213650
4.977595273663546      47    0.000071817551905
4.977595273663592      48    0.000028664570717
4.977595273663615      49    0.000007181755191
4.977595273663627      50    0.000003622102618
4.977595273663621      51    0.000002373101715
4.977595273663624      52    0.000001124100812
4.977595273663622      53    0.000000749400542
the final result that found by bisection method is 4.977595273663622 at 53
iterations

newton's method
result             iteration     error
5.356766396098938      1    365619630.205889284610748
4.993463240318037      2    14746502.113318217918277
4.977617626023528      3    20743.748697323040687
4.977595273918284      4    0.236333889958029
4.977595273663623      5    0.000000874300632
4.977595273663622      6    0.000000749400542
the result that found by newton method is 4.977595273663622 at 6 iterations

secant method
result             iteration     error
5.005340536006132      1    25811563.748005244880915
5.002989851133218      2    23619762.745349187403917
4.977657889463321      3    58109.791484059482173
4.977595413582982      4    129.849662834635723
4.977595273664387      5    0.000709245162600
4.977595273663623      6    0.000000874300632
4.977595273663622      7    0.000000749400542
the result that found by secant method is 4.977595273663622 at 7 iterations
```

as shown in here if we change tolerance 10^-15 all of the methods has the same error and the final result is also the same but bisection method's and secant method's iterations are increased. However newton's methods iteration remains the same.

If we change tolerance to 10^-16 the result is:

```
bisection method
result              iteration    error
3.500000000000000       1    1296326530.612244367599487
6.750000000000000       2    2183531183.439598083496094
5.125000000000000       3    138673788.299402296543121
4.312500000000000       4    591369739.527054429054260
4.718750000000000       5    235408314.317859262228012
4.921875000000000       6    51465266.491284213960171
5.023437500000000       7    42716636.336356408894062
4.972656250000000       8    4581603.318632887676358
4.998046875000000       9    19013899.252542551606894
4.985351562500000       10    7202969.983223498798907
4.979003906250000       11    1307416.956199861131608
4.975830078125000       12    1637906.270845139399171
4.977416992187500       13    165448.367309299617773
4.978210449218750       14    570933.312210066011176
4.977813720703125       15    202729.733735550194979
4.977615356445313       16    18637.499390107557701
4.977516174316406       17    73406.229808295480325
4.977565765380859       18    27384.564184773640591
4.977590560913086       19    4373.582143009369247
4.977602958679199       20    7131.946186560156093
4.977596759796143       21    1379.178912512546503
4.977593660354614       22    1497.202392751473326
4.977595210075378       23    59.011934089303608
4.977595984935760       24    660.083440531811220
4.977595597505569       25    300.535741168395077
4.977595403790474       26    120.761900229693353
4.977595306932926       27    30.874982445694421
4.977595258504152       28    14.068475696904503
4.977595282718539       29    8.403252843569575
4.977595270611346       30    2.832611832592757
4.977595276664943       31    2.785320661613522
4.977595273638144       32    0.023645335689437
4.977595275151543       33    1.380837506836929
4.977595274394844       34    0.678596491499039
4.977595274016494       35    0.327475234429553
4.977595273827319       36    0.151914668344855
4.977595273732732       37    0.064134635102686
4.977595273685438       38    0.020244556031557
4.977595273661791       39    0.001700389828940
4.977595273673614       40    0.009272332901489
4.977595273667703       41    0.003785971536274
4.977595273664747       42    0.001042915753757
4.977595273663269       43    0.000328487237411
4.977595273664008       44    0.000356839557902
4.977595273663638       45    0.000013926360065
4.977595273663454       46    0.000157249213650
4.977595273663546       47    0.000071817551905
4.977595273663592       48    0.000028664570717
4.977595273663615       49    0.000007181755191
4.977595273663627       50    0.000003622102618
4.977595273663621       51    0.000002373101715
4.977595273663624       52    0.000001124100812
4.977595273663622       53    0.000000749400542
4.977595273663623       54    0.000000874300632
4.977595273663622       55    0.000000749400542
4.977595273663622       56    0.000000749400542
4.977595273663622       57    0.000000749400542
4.977595273663622       58    0.000000749400542
4.977595273663622       59    0.000000749400542
4.977595273663622       60    0.000000749400542
4.977595273663622       61    0.000000749400542
4.977595273663622       62    0.000000749400542
4.977595273663622       63    0.000000749400542
4.977595273663622       64    0.000000749400542
4.977595273663622       65    0.000000749400542
4.977595273663622       66    0.000000749400542
4.977595273663622       67    0.000000749400542
4.977595273663622       68    0.000000749400542
4.977595273663622       69    0.000000749400542
4.977595273663622       70    0.000000749400542
4.977595273663622       71    0.000000749400542
4.977595273663622       72    0.000000749400542
4.977595273663622       73    0.000000749400542
4.977595273663622       74    0.000000749400542
```

```
4.977595273663622       75    0.000000749400542
4.977595273663622       76    0.000000749400542
4.977595273663622       77    0.000000749400542
4.977595273663622       78    0.000000749400542
4.977595273663622       79    0.000000749400542
4.977595273663622       80    0.000000749400542
4.977595273663622       81    0.000000749400542
4.977595273663622       82    0.000000749400542
4.977595273663622       83    0.000000749400542
4.977595273663622       84    0.000000749400542
4.977595273663622       85    0.000000749400542
4.977595273663622       86    0.000000749400542
4.977595273663622       87    0.000000749400542
4.977595273663622       88    0.000000749400542
4.977595273663622       89    0.000000749400542
4.977595273663622       90    0.000000749400542
4.977595273663622       91    0.000000749400542
4.977595273663622       92    0.000000749400542
4.977595273663622       93    0.000000749400542
4.977595273663622       94    0.000000749400542
4.977595273663622       95    0.000000749400542
4.977595273663622       96    0.000000749400542
4.977595273663622       97    0.000000749400542
4.977595273663622       98    0.000000749400542
4.977595273663622       99    0.000000749400542
4.977595273663622       100    0.000000749400542
the final result that found by bisection method is 4.977595273663622 at 100
iterations

newton's method
result              iteration     error
5.356766396098938       1    365619630.205889284610748
4.993463240318037       2    14746502.113318217918277
4.977617626023528       3    20743.748697323040687
4.977595273918284       4    0.236333889958029
4.977595273663623       5    0.000000874300632
4.977595273663622       6    0.000000749400542
4.977595273663623       7    0.000000874300632
4.977595273663622       8    0.000000749400542
4.977595273663623       9    0.000000874300632
4.977595273663622       10    0.000000749400542
4.977595273663623       11    0.000000874300632
4.977595273663622       12    0.000000749400542
4.977595273663623       13    0.000000874300632
4.977595273663622       14    0.000000749400542
4.977595273663623       15    0.000000874300632
4.977595273663622       16    0.000000749400542
4.977595273663623       17    0.000000874300632
4.977595273663622       18    0.000000749400542
4.977595273663623       19    0.000000874300632
4.977595273663622       20    0.000000749400542
4.977595273663623       21    0.000000874300632
4.977595273663622       22    0.000000749400542
4.977595273663623       23    0.000000874300632
4.977595273663622       24    0.000000749400542
4.977595273663623       25    0.000000874300632
4.977595273663622       26    0.000000749400542
4.977595273663623       27    0.000000874300632
4.977595273663622       28    0.000000749400542
4.977595273663623       29    0.000000874300632
4.977595273663622       30    0.000000749400542
4.977595273663623       31    0.000000874300632
4.977595273663622       32    0.000000749400542
4.977595273663623       33    0.000000874300632
4.977595273663622       34    0.000000749400542
4.977595273663623       35    0.000000874300632
4.977595273663622       36    0.000000749400542
4.977595273663623       37    0.000000874300632
4.977595273663622       38    0.000000749400542
4.977595273663623       39    0.000000874300632
4.977595273663622       40    0.000000749400542
4.977595273663623       41    0.000000874300632
4.977595273663622       42    0.000000749400542
4.977595273663623       43    0.000000874300632
4.977595273663622       44    0.000000749400542
4.977595273663623       45    0.000000874300632
4.977595273663622       46    0.000000749400542
```

```
4.977595273663623      47    0.000000874300632
4.977595273663622      48    0.000000749400542
4.977595273663623      49    0.000000874300632
4.977595273663622      50    0.000000749400542
4.977595273663623      51    0.000000874300632
4.977595273663622      52    0.000000749400542
4.977595273663623      53    0.000000874300632
4.977595273663622      54    0.000000749400542
4.977595273663623      55    0.000000874300632
4.977595273663622      56    0.000000749400542
4.977595273663623      57    0.000000874300632
4.977595273663622      58    0.000000749400542
4.977595273663623      59    0.000000874300632
4.977595273663622      60    0.000000749400542
4.977595273663623      61    0.000000874300632
4.977595273663622      62    0.000000749400542
4.977595273663623      63    0.000000874300632
4.977595273663622      64    0.000000749400542
4.977595273663623      65    0.000000874300632
4.977595273663622      66    0.000000749400542
4.977595273663623      67    0.000000874300632
4.977595273663622      68    0.000000749400542
4.977595273663623      69    0.000000874300632
4.977595273663622      70    0.000000749400542
4.977595273663623      71    0.000000874300632
4.977595273663622      72    0.000000749400542
4.977595273663623      73    0.000000874300632
4.977595273663622      74    0.000000749400542
4.977595273663623      75    0.000000874300632
4.977595273663622      76    0.000000749400542
4.977595273663623      77    0.000000874300632
4.977595273663622      78    0.000000749400542
4.977595273663623      79    0.000000874300632
4.977595273663622      80    0.000000749400542
4.977595273663623      81    0.000000874300632
4.977595273663622      82    0.000000749400542
4.977595273663623      83    0.000000874300632
4.977595273663622      84    0.000000749400542
4.977595273663623      85    0.000000874300632
4.977595273663622      86    0.000000749400542
4.977595273663623      87    0.000000874300632
4.977595273663622      88    0.000000749400542
4.977595273663623      89    0.000000874300632
4.977595273663622      90    0.000000749400542
4.977595273663623      91    0.000000874300632
4.977595273663622      92    0.000000749400542
4.977595273663623      93    0.000000874300632
4.977595273663622      94    0.000000749400542
4.977595273663623      95    0.000000874300632
4.977595273663622      96    0.000000749400542
4.977595273663623      97    0.000000874300632
4.977595273663622      98    0.000000749400542
4.977595273663623      99    0.000000874300632
4.977595273663622      100    0.000000749400542
the result that found by newton method is 4.977595273663622 at 100 iterations

secant method
result             iteration     error
5.005340536006132      1    25811563.748005244880915
5.002989851133218      2    23619762.745349187403917
4.977657889463321      3    58109.791484059482173
4.977595413582982      4    129.849662834635723
4.977595273664387      5    0.000709245162600
4.977595273663623      6    0.000000874300632
4.977595273663622      7    0.000000749400542
4.977595273663622      8    0.000000749400542
the result that found by secant method is 4.977595273663622 at 8 iterations
```

if we change tolerance to 10^-16 the result don't change comparing to 10^-15 but the iterations are increased, especially at the newton's method and bisection method. And since newton's methods iteration increased so much we can say that secant method is our most efficient method. newton's method and bisection method are has the same iteration and same final error so we can say that these two methods equal about efficiency and they start to share the second place.

Appendix

Full part of the code

```
%% function definition
clear all
format long
f= @(r) (1/(4*pi*(1/(36*pi)*10^-9)))*(13*(r--7)/(abs(r--7)^3)+9*(r--
4)/(abs(r--4)^3)+6*(r-11)/(abs(r-11)^3)+3*(r-14)/(abs(r-14)^3));
tolerance=10^-10;

%% bisection method
min=-3;
max=10;
fprintf("bisection method\nresult              iteration    error\n");
for i=1:100
    mid= min+(max-min)/2;
    if f(min)*f(mid)<0
       max=mid;
    else
        min=mid;
    end
    a1(1,i)=i;
    bisection(1,i)=abs(0-f(mid));
    fprintf("%1.15f      %1.f    %1.15f\n",mid,i,abs(0-f(mid)));
if (max-min)/2<tolerance
    break
end

end
fprintf("the final result that found by bisection method is %1.15f at %1.f
iterations\n\n",mid,i);

%% newton's method
h=10^-10;
f_turev= @(x) (f(x+h)-f(x))/h; %derivative can be found by using limit
definition
p0=6.5;
fprintf("newton's method\nresult              iteration     error\n");
for i=1:100
p=p0-f(p0)/f_turev(p0);
a2(1,i)=p;
newton(1,i)=abs(0-f(p));
fprintf("%1.15f      %1.f    %1.15f\n",p,i,abs(0-f(p)));
if abs(p-p0)<tolerance
    break
end
p0=p;
end
fprintf("the result that found by newton method is %1.15f at %1.f
iterations\n\n",p,i);

%% secant method
x1=-3;x2=10;
q0=f(x1);
q1=f(x2);
fprintf("secant method\nresult              iteration     error\n");
for i=1:100
    p=x2-q1*(x2-x1)/(q1-q0);
    fprintf("%1.15f      %1.f    %1.15f\n",p,i,abs(0-f(p)));
    if abs(p-x2)<tolerance
        break
```

```
    end
    a3(1,i)=i;
    secand(1,i)=abs(0-f(p));
    x1=x2;
    q0=q1;
    x2=p;
    q1=f(p);

end
fprintf("the result that found by secant method is %1.15f at %1.f
iterations\n",p,i);
%% ploting the graph
xlabel("iteration");
ylabel("error");
title("error graph according to iteration");
hold on
loglog(a1,bisection);
loglog(a2,newton);
loglog(a3,secand);
legend("bisection","newton","secant");
hold off
```