Coding Report 1

April 21, 2023

1 Problem Description

This report evaluates the convergence performance of four root finding methods: Bisection, Fixed-Point, Newton, and Secant methods, using two specific equations: $f_1(x) = \frac{1}{2}\sin((x-1)^3)$ and $f_2(x) = 6^{-x}-2$. In the code, the equations are defined as "`my_fun1`" and "`my_fun2`" respectively, with their derivatives labeled as "`my_fun_deriv1`" and "`my_fun_deriv2`". The report presents a visual comparison of their performance in terms of computing an approximation of the root with an accuracy of at least $1e-8$, by plotting the errors for each method on a single graph. The x-axis represents the number of error iterations, while the y-axis represents the absolute errors. The report includes four plots in total: two plots for the original functions f1(x) and f2(x), and two additional plots showing the log-log of f1(x) and f2(x). The report also identifies the smallest positive fixed points.

2 Results

Table 1: The appropriate starting points				
	Bisection	Fixed Point	Newton	Secant
	(a, b, tol, N0)	(p0, tol, N0)	(p0, tol, N0)	(p0, p1, tol, N0)
$f_1(x) = \frac{1}{2}\sin((x-1)^3)$	(-1, 2, $1e-8$, 100)	(1.8 , $1e-8$, 100)	(1.5 ,$1e-8$, 100)	(-1 , 2 , $1e-8$, 100)
$f_2(x) = 6^{-x} - 2$	(-1 , 0 , $1e-8$, 100)	(-0.3 , $1e-8$, 100)	(-0.25 ,$1e-8$, 100)	(-1 , 0 , $1e-8$, 100)

Table 1: A table of the appropriate starting points for $f_1(x) = \frac{1}{2}\sin((x-1)^3)$ and $f_2(x) = 6^{-x} - 2$ with Endpoints (a and b), Initial Guesses (p0 and p1), Tolerance (tol), and Maximum Iterations (N0). (Note: We set the tolerance tol=1e-8 and Maximum iteration N0=100 for all four methods.)

After thorough analysis of the results, it can be concluded that the Bisection method and Fixed-Point method exhibit faster convergence rates compared to the other methods for functions f1(x) and f2(x), as evidenced by the rate of change in the absolute error over 100 iterations.

For f1(x) = $\frac{1}{2}\sin((x-1)^3)$, the Bisection method was applied with -1 as the value for a and 2 as the value for b, chosen to have opposite signs of f(a) and f(b). The Fixed-Point method was implemented using the equation $g(x) = x - \frac{f(x)}{f'(x)}$, with an initial guess of 1.8 for p0. The Newton method was applied with 1.5 as the value of p0. The Secant method was applied with -1 and 2 as the values of p0 and p1. The Figure 1 depicts the convergence of f1(x), clearly showing the significantly faster convergence of the Bisection method (blue line in the plot) compared to the other methods. This observation is further supported by the log-log plot of f1(x) in Figure 2, where the blue line representing the Bisection method exhibits a steeper slope, indicating faster convergence. Then, we find that the root finding methods, ranked from fastest to slowest convergence rates, are as follows: Bisection, Newton, Fixed-Point, and Secant. Additionally, the smallest positive root of f1(x) is observed to be 1.

Next, for f2(x) = $6^{-x} - 2$, the Bisection method was applied with -1 as the value for a and 0 as the value for b, chosen to have opposite signs of f(a) and f(b). The Fixed-Point method was implemented using the equation $g(x) = x - \frac{f(x)}{f'(x)}$, with an initial guess of -0.3 for p0. The Newton method was applied with -0.25 as the value of p0. The Secant method was applied with -1 and 0 as the values of p0 and p1. Figure 3 showcases the convergence of f2(x), clearly demonstrating the much faster convergence of the Fixed-Point method (red line in the plot) compared to the other methods, as evident from the rate of change in the absolute error with respect to the number of iterations. This finding is further supported by the log-log plot of f2(x) in Figure 4, where the red line representing the Fixed-Point method exhibits a steeper slope, indicating faster convergence. Then, we find that the root finding methods, ranked from fastest to slowest convergence rates, are as follows: Fixed-Point, Newton, Secant, and Bisection. Furthermore, the smallest root of f2(x) is observed to be -0.3869.

In conclusion, based on the analysis of the convergence rates in terms of the absolute error and number of iterations, the Bisection method and Fixed-Point method demonstrate faster convergence compared to the other methods for the given functions: f1(x) = $\frac{1}{2}\sin((x-1)^3)$ and f2(x) = $6^{-x} - 2$.

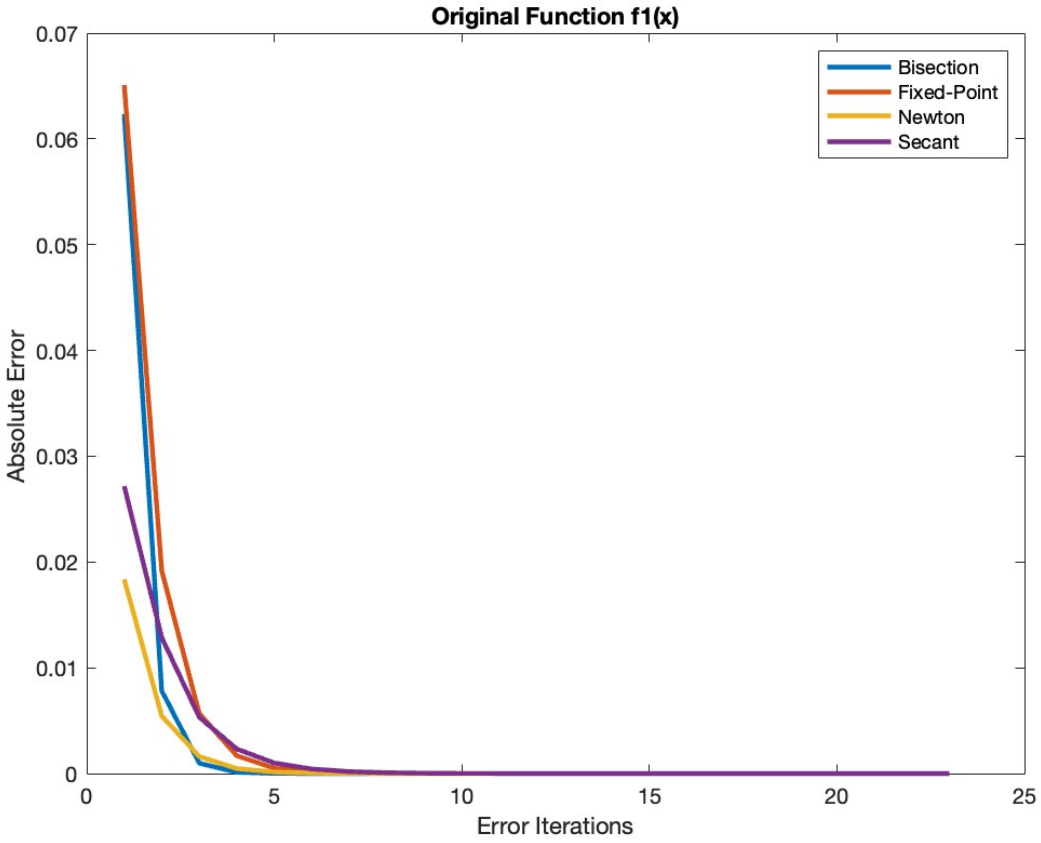

Figure 1: Plot number of error iteration (x-axis) vs. absolute error (y-axis) to evaluate convergence performance of Bisection, Fixed Point, Newton, and Secant methods for $f_1(x) = \frac{1}{2}\sin((x-1)^3)$.

(Note: The Bisection method, as observed in the Log-Log plot, exhibits the fastest convergence rate due to the steeper slope. The fastest to slowest convergence rates: Bisection -> Newton -> Fixed-point -> Secant)

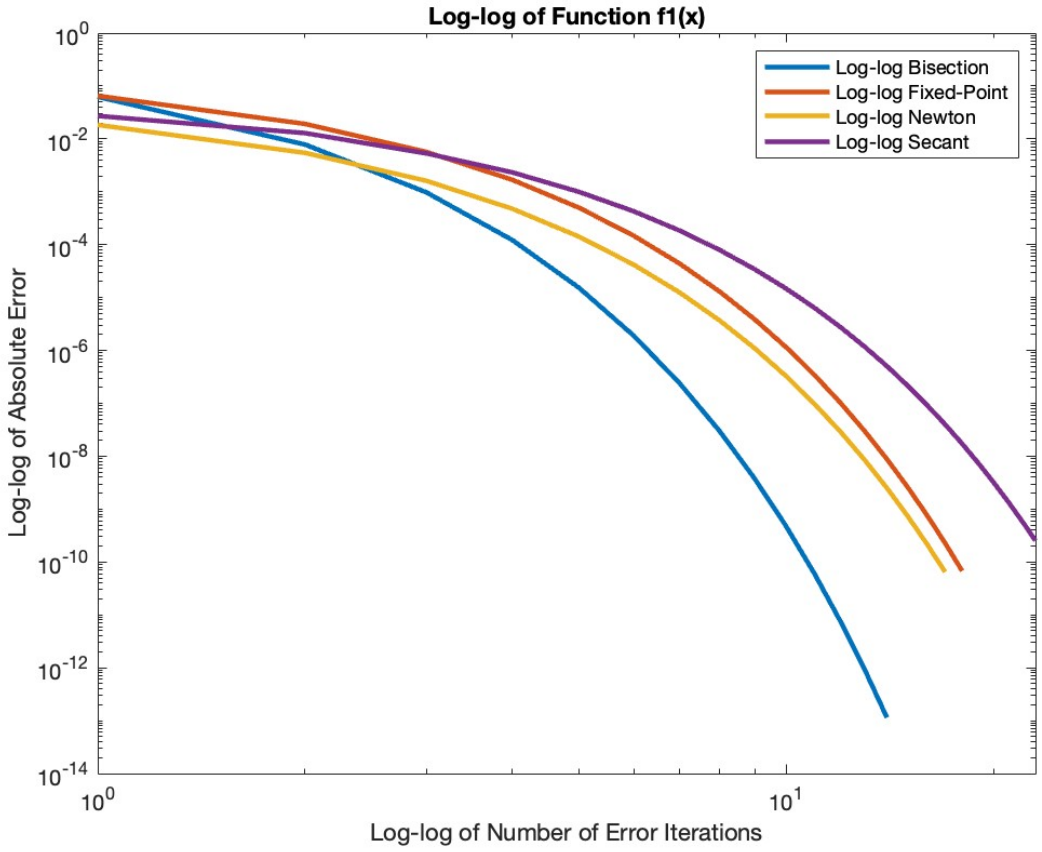

Figure 2: Plot Log-Log of number of error iterations (x-axis) vs. Log-Log of absolute error (y-axis) for Bisection, Fixed Point, Newton and Secant methods to determine the convergence order when taking Log-Log of $f_1(x) = \frac{1}{2}\sin((x-1)^3)$.

(Note: The Bisection method, as observed in the Log-Log plot, exhibits the fastest convergence rate due to the steeper slope. The fastest to slowest convergence rates: Log-Log Bisection -> Log-Log Newton -> Log-Log Fixed-point -> Log-Log Secant)

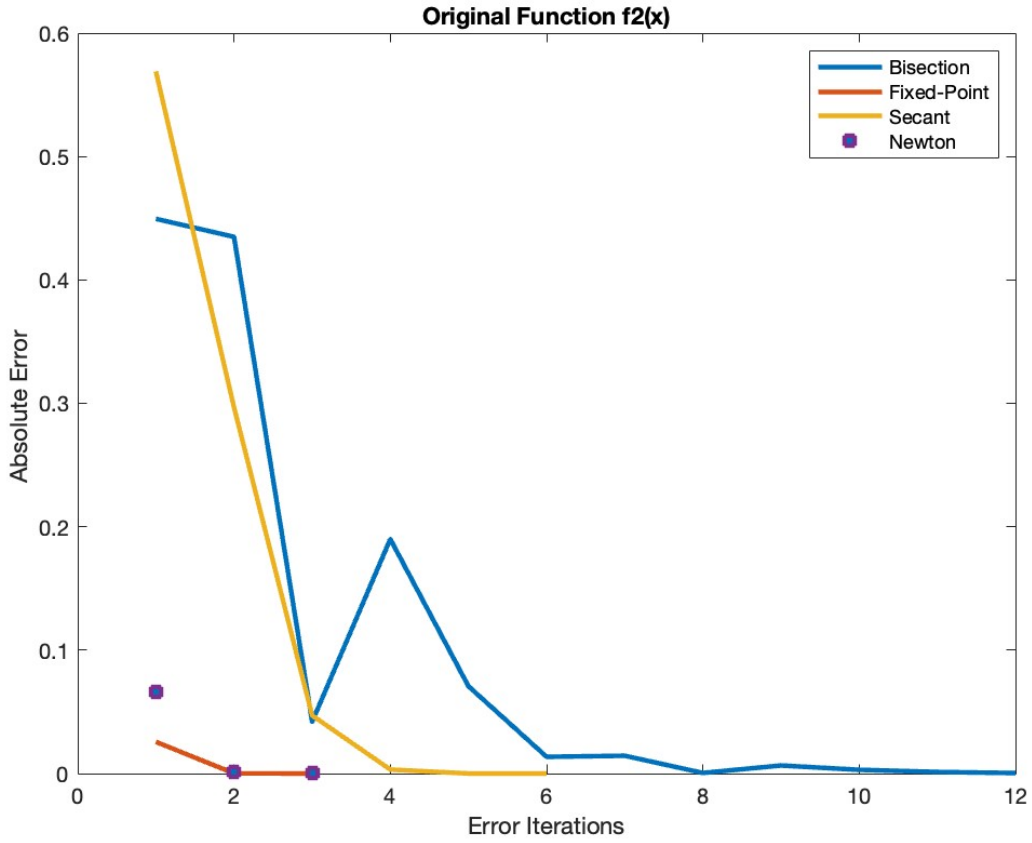

Figure 3: Plot number of error iteration (x-axis) vs. absolute error (y-axis) to evaluate convergence performance of Bisection, Fixed Point, Newton, and Secant methods for $f_2(x) = 6^{-x} - 2$

(Note: The Fixed-Point method, as observed in the Log-Log plot, exhibits the fastest convergence rate due to the steeper slope. The fastest to slowest convergence rates: Fixed-point -> Newton -> Secant -> Bisection)

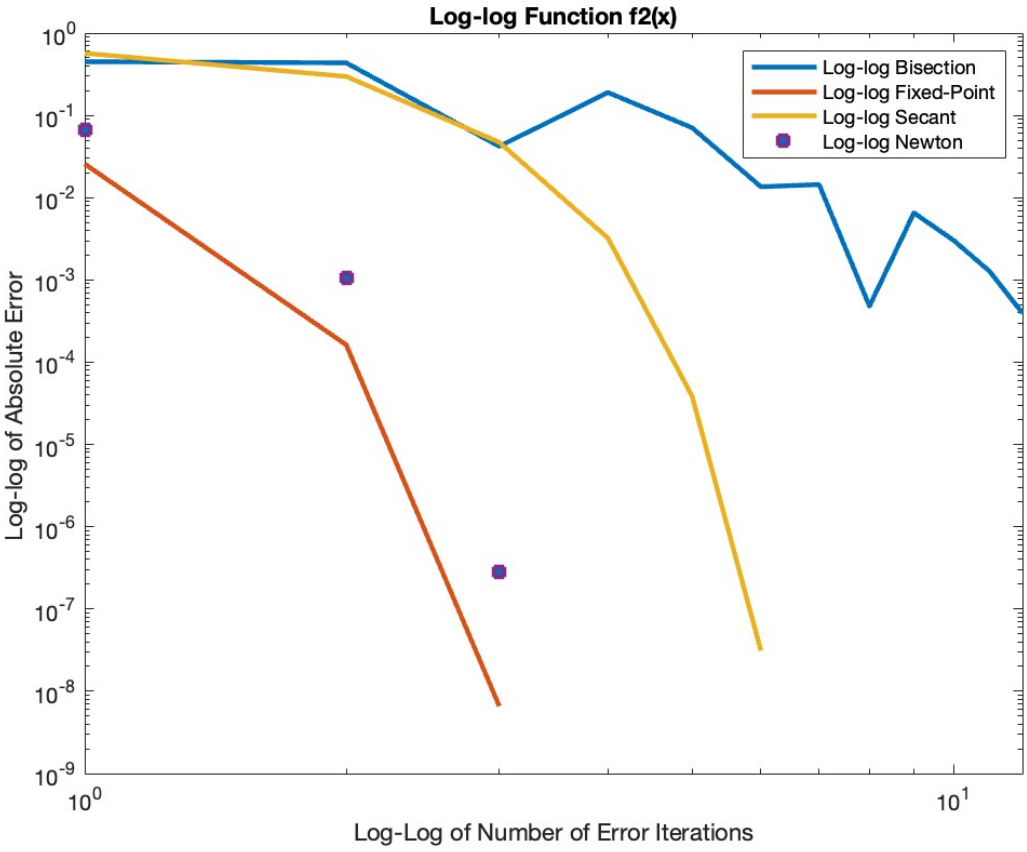

Figure 4: Plot Log-Log of number of error iterations (x-axis) vs. Log-Log of absolute error (y-axis) for Bisection, Fixed Point, Newton and Secant methods to determine the convergence order when taking Log-Log of $f_2(x) = 6^{-x} - 2$.

(Note: The Fixed-Point method, as observed in the Log-Log plot, exhibits the fastest convergence rate due to the steeper slope. The fastest to slowest convergence rates: Log-Log Fixed-point -> Log-Log Newton -> Log-Log Secant -> Log-Log Bisection)

3 Collaboration

Yirong Xu and Sebrina Zhu

4 Academic Integrity

On my personal integrity as a student and member of the UCD community, I have not given nor received any unauthorized assistance on this assignment.

5 Appendix

```
clc;clear; close all;
%---------------Bisection for f1(x)-----------
[p1_bisec, abs_err1_bisec] = Bisection1(-1, 2,1e-8, 100);
%----------------Fixed_point for f1(x---------
[p1_fix, abs_err1_fix] = Fixed_Point1(1.8,1e-8, 100);
%----------------Netwon for f1(x)-----------
[p1_newt, abs_err1_newt] = Newton1(1.5,1e-8,100);
%----------------Secant for f1(x)-------------
[p1_sec, abs_err1_sec] = Secant1(-1, 2,1e-8, 100);


%---------------Bisection for f2(x)------------------
[p2_bisec, abs_err2_bisec] = Bisection2(-1, 0,1e-8, 100);
%----------------Fixed_point for f2(x)---------------
[p2_fix, abs_err2_fix] = Fixed_Point2(-0.3,1e-8, 100);
%----------------Netwon for f2(x)--------------------
[p2_newt, abs_err2_newt] = Newton2(-0.25,1e-8,100);
%----------------Secant for f2(x)--------------------
[p2_sec, abs_err2_sec] = Secant2(-1, 0,1e-8, 100);


%--------Figure for Function f1(x)---------
figure;
x1_bisec = 1:1:numel(abs_err1_bisec);
x1_fix = 1:1:numel(abs_err1_fix);
x1_newt = 1:1:numel(abs_err1_newt);
x1_sec = 1:1:numel(abs_err1_sec);
plot(x1_bisec,abs_err1_bisec,x1_fix,abs_err1_fix,x1_newt,
   abs_err1_newt,x1_sec,abs_err1_sec,'LineWidth',2.2)

title('Original Function f1(x)')
legend('Bisection','Fixed-Point','Newton','Secant')
xlabel('Error Iterations')
ylabel('Absolute Error ')

%--------Figure for Log-Log of Function f1(x)------------
figure;
loglog(x1_bisec,abs_err1_bisec,x1_fix,abs_err1_fix,x1_newt,
   abs_err1_newt,x1_sec,abs_err1_sec,'LineWidth',2.2)

title('Log-log of Function f1(x)')
legend('Log-log Bisection','Log-log Fixed-Point','Log-log
   Newton','Log-log Secant')
xlabel('Log-log of Number of Error Iterations')
ylabel('Log-log of Absolute Error')
```

```
%-------Figure for Function f2(x)----------------------
figure;
x2_bisec = 1:1:numel(abs_err2_bisec);
x2_fix = 1:1:numel(abs_err2_fix);
x2_newt = 1:1:numel(abs_err2_newt);
x2_sec = 1:1:numel(abs_err2_sec);
plot(x2_bisec,abs_err2_bisec,x2_fix,abs_err2_fix,x2_sec,
   abs_err2_sec,x2_newt, abs_err2_newt, 's','MarkerFaceColor'
   ,[0 0.447 0.741],'LineWidth',2.2)

title('Original Function f2(x)')
legend('Bisection','Fixed-Point','Secant','Newton')
xlabel('Error Iterations')
ylabel('Absolute Error ')

%---------Figure for Log-Log of Function f2(x)------------
figure;
loglog(x2_bisec,abs_err2_bisec,x2_fix,abs_err2_fix,x2_sec,
   abs_err2_sec,x2_newt,abs_err2_newt, 's','MarkerFaceColor'
   ,[0 0.447 0.741],'LineWidth',2.2)

title('Log-log Function f2(x)')
legend('Log-log Bisection','Log-log Fixed-Point','Log-log
   Secant','Log-log Newton')
xlabel('Log-Log of Number of Error Iterations')
ylabel('Log-log of Absolute Error ')

%-------------------f1(x)--------------------------
% define the function of f1(x)
function f1 = my_fun1(x)
    f1 = (1/2) * sin((x-1).^3);
end
%-------------------f1(x) derivative----------------
% define the derivative function of f1(x)
function f1_deriv = my_fun_deriv1(x)
    f1_deriv = (3/2) * (x - 1).^2 .* cos((x - 1).^3);
end
%-------------------f2(x)--------------------------
% define the function of f2(x)
function f2 = my_fun2(x)
    f2 = (6.^(-x))-2;
end
%-------------------f2(x) derivative----------------
% define the derivative function of f2(x)
function f2_deriv = my_fun_deriv2(x)
    f2_deriv = -log(6) .* (6.^(-x));
end
```

```
%--------bisection for f1(x)----------------
% define the bisection method for f1(x)
function [p1_bisec, abs_err1_bisec] = Bisection1( a, b, tol,
   N0)
    i = 1;
    FA = my_fun1(a);
    %trueValue = 1;
    abs_err1_bisec=[];
    while i <= N0
        p1_bisec = a + (b-a)/2;
        q1 = my_fun1(p1_bisec);
        %abs_err1_bisec(end+1)=abs(p1_bisec-trueValue);
        abs_err1_bisec(end+1)=abs(q1-0);
        FP = my_fun1(p1_bisec);
        if FP==0 || (b-a)/2 < tol
            return
        end
        % Another case for i = i+1
        i = i+1;
        if FA*FP > 0
            a=p1_bisec;
            FA=FP;
        else
            b=p1_bisec;
        end
    end
    if i > N0
        fprintf('Method failed after N0 iterations, N0=',N0);
    end
end

%-------bisection for f2(x)----------------------------
% define the bisection method for f2(x) (p2_bisec,
   abs_err2_bisec)
function [p2_bisec, abs_err2_bisec] = Bisection2( a, b, tol,
   N0)
    i = 1;
    FA = my_fun2(a);
    %trueValue = (-log(2))/log(6);
    abs_err2_bisec=[];
    while i <= N0
        p2_bisec = a + (b-a)/2;
        q2 = my_fun2(p2_bisec);
        %abs_err2_bisec(end+1)=abs(p2_bisec - trueValue);
        abs_err2_bisec(end+1)=abs(q2-0);
        FP = my_fun2(p2_bisec);
        if FP==0 || (b-a)/2 < tol
```

```
            return
        end
        % Another case for i = i+1
        i = i+1;
        if FA*FP > 0
            a=p2_bisec;
            FA=FP;
        else
            b=p2_bisec;
        end
    end
    if i > N0
        fprintf('Method failed after N0 iterations, N0=',N0);
    end
end

%----------fixed-point for f1(x)-------------------------
% define the fixed-point method implementation for f1(x)
function [p1_fix, abs_err1_fix] = Fixed_Point1(p0, TOL, N0)
    i = 1;
    %trueValue = 1;
    abs_err1_fix=[];
    while i <= N0
        p1_fix=p0-my_fun1(p0)/my_fun_deriv1(p0);
        q1_fix = my_fun1(p1_fix);
        %abs_err1_fix(end+1)=abs(p1_fix-trueValue);
        abs_err1_fix(end+1)=abs(q1_fix - 0);
        if abs(p1_fix-p0) < TOL
            return
        end
      i = i + 1;
      p0 = p1_fix;
    end
    if i > N0
        fprintf('Method failed after N0 iterations, N0=',N0);
    end
end

%---------fixed-point for f2(x)-------------------------
% define the fixed-point method implementation for f2(x)
function [p2_fix, abs_err2_fix] = Fixed_Point2(p0, TOL, N0)
    i = 1;
    abs_err2_fix=[];
    %trueValue = (-log(2))/log(6);
    while i <= N0
        p2_fix=p0-my_fun2(p0)/my_fun_deriv2(p0);
        q2_fix = my_fun2(p2_fix);
        %abs_err2_fix(end+1)=abs(p2_fix-trueValue);
```

```
        abs_err2_fix(end+1)=abs(q2_fix -0);
        if abs(p2_fix-p0) < TOL
            return
        end
      i = i + 1;
      p0 = p2_fix;
    end
    if i > N0
        fprintf('Method failed after N0 iterations, N0=',N0);
    end
end

%--------Newton for f1(x)--------------------------
% define the Newton function for f1(x)
function [p1_newt, abs_err1_newt] = Newton1(p0, TOL, N0)
    i = 1;
    abs_err1_newt=[];
    while i <= N0
        p1_newt= p0 - my_fun1(p0)/my_fun_deriv1(p0);
        q1_newt = my_fun1(p1_newt);
        %abs_err1_newt(end+1)=abs(p1_newt-1);
        abs_err1_newt(end+1)=abs(q1_newt - 0);
        if abs(p1_newt-p0) < TOL
            return
        end
      i = i + 1;
      p0 = p1_newt;
    end
    if i > N0
        fprintf('Method failed after N0 iterations, N0=',N0);
    end
end

%-------Newton for f2(x)--------------------------
% define the Newton function for f2(x)
function [p2_newt, abs_err2_newt] = Newton2(p0, TOL, N0)
    i = 1;
    abs_err2_newt=[];
    trueValue = (-log(2))/log(6);
    while i <= N0
        p2_newt= p0 - my_fun2(p0)/my_fun_deriv2(p0);
        q2_newt = my_fun2(p2_newt);
        % abs_err2_newt(end+1)=abs(p2_newt-trueValue);
        abs_err2_newt(end+1)=abs(q2_newt - 0);
        if abs(p2_newt-p0) < TOL
            return
        end
      i = i + 1;
```

```
      p0 = p2_newt;
    end
    if i > N0
        fprintf('Method failed after N0 iterations, N0=',N0);
    end
end

%----------Secant for f1(x)---------------------------
% define the Secant function for f1(x)
function [p1_sec, abs_err1_sec] = Secant1(p0, p1, TOL, N0)
    i = 2;
    % trueValue = 1;
    abs_err1_sec=[];
    q0=my_fun1(p0);
    q1=my_fun1(p1);
    while i <= N0
        p1_sec= p1 - q1*(p1-p0)/(q1-q0);
        q1_sec = my_fun1(p1_sec);
        %abs_err1_sec(end+1)=abs(p1_sec-trueValue);
        abs_err1_sec(end+1)=abs(q1_sec-0);
        if abs(p1_sec-p1) < TOL
            return
        end
      i = i + 1;
      p0 = p1;
      q0=q1;
      p1=p1_sec;
      q1=my_fun1(p1_sec);
    end
    if i > N0
        fprintf('Method failed after N0 iterations, N0=',N0);
    end
end

%--------------Secant for f2(x)---------------------------
% define the Secant function for f2(x)
function [p2_sec, abs_err2_sec] = Secant2(p0, p1, TOL, N0)
    i = 2;
    abs_err2_sec=[];
    q0=my_fun2(p0);
    q1=my_fun2(p1);
    % trueValue = (-log(2))/log(6);
    while i <= N0
        p2_sec= p1 - q1*(p1-p0)/(q1-q0);
        q2_sec = my_fun2(p2_sec);
        %abs_err2_sec(end+1)=abs(p2_sec-trueValue);
        abs_err2_sec(end+1)=abs(q2_sec-0);
```

```
        if abs(p2_sec-p1) < TOL
            return
        end
      i = i + 1;
      p0 = p1;
      q0=q1;
      p1=p2_sec;
      q1=my_fun2(p2_sec);
    end
    if i > N0
        fprintf('Method failed after N0 iterations, N0=',N0);
    end
end
```