1. Bisection Method

f(x) = $e^x - x - 1$ *on* $[-0.6, 1.4]$					
Iteration, n	Apprx. root, p_n	f(x)	Abs. Err., $\in_n = \lvert p_n - 0 \rvert$	red. factor, m_n	order, a_n
0	0.400000	0.091825	0.400000	-	-
1	0.900000	0.559603	0.55960311	1.3990078	0.588687009
2	1.150000	1.008193	1.00819291	1.8016	0.257047927
3	1.275000	1.303701	1.30370141	1.293107	0.121419876
4	1.337500	1.472008	1.47200782	1.129099	0.059137765
5	1.368750	1.561685	1.56168458	1.060921	0.029198038
6	1.384375	1.607955	1.60795492	1.029628	0.014508905
7	1.392187	1.631455	1.63145465	1.014615	0.007232239
8	1.396094	1.643296	1.64329649	1.007258455	0.003610606
9	1.398047	1.649241	1.64924051	1.003617132	0.00180393
10	1.399023	1.652218	1.65221831	1.001805558	0.000901617
11	1.399512	1.653709	1.65370865	1.000902024	0.000450728
12	1.399756	1.654454	1.65445419	1.000450829	0.000225342
13	1.399878	1.654827	1.65482705	1.000225367	0.000112664
14	1.399939	1.655013	1.65501350	1.00011267	5.63303E-05
15	1.399969	1.655107	1.65510673	1.000056332	2.8167E-05
16	1.399985	1.655153	1.65515335	1.000056332	1.40832E-05
17	1.399992	1.655177	1.65517666	1.000028167	7.0385E-06
18	1.399996	1.655188	1.65518831	1.000014083	3.52225E-06
19	1.399998	1.655194	1.65519414	1.000007039	1.7581E-06
20	1.399999	1.655197	1.65519705	1.000003522	8.8207E-07
21	1.400000	1.655199	1.65519851	1.000001758	4.41035E-07
22	1.400000	1.655199	1.65519924	1.000000882	2.17496E-07
23	1.400000	1.655200	1.65519960	1	2.17496E-07

g(x) = ln(x) - cos(x) on [0.6, 2.6]					
Iteration, n	Apprx. root, p_n	g(x)	Convergence, $\in_n = \lvert p_n - p_{n-1} \rvert$	red. factor, m_n	order, a_n
0	1.600000	0.499203	.499203	-	-
1	1.100000	-0.358286	0.35828594	0.6875	-0.374693449
2	1.350000	0.081098	0.08109791	1.227272727	-2.995732274
3	1.225000	-0.136005	0.13600513	0.907407407	-3.129263666
4	1.287500	-0.026820	0.02681974	1.051020408	-3.114593476
5	1.318750	0.027298	0.02729817	1.024271845	-3.124513063
6	1.303125	0.000279	0.00027882	0.988151659	-3.169221978

Compare performance of Bisection method in the cases of f(x) = 0 and g(x) = 0. Did bisection method performed similar or different? How do you explain this behavior using what we have learned in Chapter 2?

The two cases of f(x) = 0 and g(x) = 0 got very different performances. From above, we find that g(x) = ln(x) - cos(x) on [0.6, 2.6] has an approximated root at 1.303125 where the absolute error is 0.00027882. It means that it is very close with a very low chance of error that it is not its approximated root.

Unlike for f(x) = e^x - x - 1 on [-0.6, 1.4], its performance is not as accurate and has a very high absolute error. In the code, it keeps repeating its iterations where the approximated root stays at 1.40000... and the absolute error reaches to 1.65519997... because as its approximated root changes from 1.400001 to 1.40002, etc. Its absolute error keeps getting higher and higher from 1.65519960 to 1.65519997 at 200 iterations. What this tells us that as iterations go by, we are stuck with the error up to 1.65519960 if we find the approximate root as 1.40000.

Since f(x) has a much higher error than g(x), we see that g(x) approximate root is more accurate. It also tells us that bisection method works better with g(x) rather than with f(x). One of the reasons is because of the interval. By using the bisection method, it finds its approximated root by using the given intervals, [a,b]. From the given interval, it plugs a into f(x) and b into f(x). We will call the outcome $f(a_o) and\ f(b_o)$ where o stands for original interval. After seeing the outcome, we need to find the red. factor. To find its red. factor, plug the intervals given into the formula of $\frac{a-a-b}{2}$. The outcome

of this will be called r_n. We use the r_n to plug in the function to tells us how to reduce its interval. So if the outcome happens to be less than 0, then we would replace a with the r_n. If the outcome was greater than 0, then we would replace b with the r_n. This process is to slowly making the intervals smaller and smaller. As the intervals get smaller, we can find the approximated root easier since it reduced the amount of numbers. We keep repeating this step until we find its approximated root. While we are doing this step, we have to make sure to include the absolute error. This absolute error number will determine if the number we obtain for the approximation root is accurate or not.

Back to talking about the function f(x)... We see that it started off with a bigger interval of [-0.6, 1.4], it had a much bigger chance of finding its root. The absolute error was not high either. As we slowly reduced the scale of the interval by using bisection method, we actually ended up reducing its chance of finding its root. The absolute error was getting higher and the red. factor was slowly reaching to 1. Once it reached to 1, it told us that our approximation root cannot be found further. Same thing with our order, a_n where it ended up at 2.1746E-07 and kept repeating as each iteration goes by.

For g(x), it was the quite the opposite. As we kept finding its approximated root, the absolute error and order was getting smaller. The red. factor had an equilibrium to 1.

From these differences, we see that g(x) function is more successful than the f(x) function in the bisection method. If we had to make f(x) as successful as g(x) in the bisection method, we would have to use a different method. Some methods work better for some functions.

2. Newtons Method

f(x) = $e^x - x - 1$ *on* $[-0.6, 1.4]$					
Iteration, n	Apprx. root, p_n	f(x)	Abs. Err., $\in_n = \lvert p_n - 0 \rvert$	red. factor, m_n	order, a_n
1	0.400000	0.091825	0.091825	-	-
2	0.213298	0.024455	0.18670209	0.533245	-0.628774298
3	0.110437	0.006329	0.10286049	0.517759191	-0.658245026
4	0.056235	0.001611	0.05420255	0.509204343	-0.674905884
5	0.028381	0.000407	0.02785392	0.504685694	-0.683819432
6	0.014258	0.000102	0.01412335	0.502378352	-0.688401754
7	0.007146	0.000026	0.00711186	0.501192313	-0.690765393
8	0.003577	0.000006	0.00356861	0.500559754	-0.692028299
9	0.001790	0.000002	0.00178750	0.500419346	-0.69230884
10	0.000895	0.000000	0.00089455	0.5	-0.693147181

g(x) = ln(x) - cos(x) on [0.6, 2.6]					
Iteration, n	Apprx. root, p_n	g(x)	Convergence $\in_n = \lvert p_n - p_{n-1} \rvert$	red. factor, m_n	order, a_n
1	1.600000	0.499203	1.6	-	-
2	1.292717	-0.017762	0.30728257	0.807948125	-0.213257424
3	1.302954	-0.000017	0.01023677	1.00791898	0.007887789
4	1.302964	-0.000000	0.00000980	1.000007675	7.67484E-06

Compare performance of Newton's method in the cases of f(x) = 0 and g(x) = 0. Did Newton's method performed similar or different? How do you explain this behavior using results of Chapter 2?

From the tables, we see that f(x) = $e^x - x - 1$ finds its approximated root at 0.000895 with an absolute error of 0.00089455 in 10 iterations. We also see that g(x) = ln(x) - cos(x) found its approximated root at 1.302964 with a convergence at 0.00000980 in 4 iterations.

These both have very different behaviors. In f(x), the red. factor, m_n values are getting closer and close to 5. The order, a_n is in negatives, has small jumps, and it is less than 0. The function when plugging in the approximated roots, it reaches to 0

which means we are very close.

In g(x), the red. factor, m_n is going back and forth around 1. The order, a_n is going in big jumps, and starts negative then positive. The approximated roots are repeating around 1.302960 and the outcome when plugged into g(x) is a very small negative number close to 0. As it gets negative number gets smaller to zero, the convergence number gets smaller also.

By looking at these two tables, it seems that f(x) is more accurate since it provides more iteration and data. Also, the numbers in f(x) intended to be more smaller jump meaning it is very close to its approximated root. However in g(x), it does work, but the data seems not successful because of the order, a_n is off the charts. If we ignored the order, a_n I see that g(x) is a successful method because it did find its root at a very low number of convergence.

To solve f(x), use the Newtons method. It will give you more accurate data than bisections method.

To solve g(x), use the bisection method or the newtons method. They both end up with the same approximation root. However, they did get different convergence numbers. The convergence number is smaller in the newtons method than the bisections method. It means that the newton method was more accurate.

THE CODES FOR MATLAB

$Thecodeoffxmaincodeprojectfinal.m$:

```
clearall
closeall
clc
symsx
```

$f(x) = exp(x) - x - 1;$
$jj = diff(f, x);$
$a = -0.6; b = 1.4;$
$error = 1e - 3;$
$N = 100;$
$xp = (a + b)/2;$
$fplot(f, [a, b])$
$holdon$
$fplot(0, [ab]);$
$fprintf('BisectionMethod :');$
rt1 iteration1 error1
=
$fxbisectionMethod(f, a, b, error);$
$fprintf('FixedPointMethod :');$
rt2 iteration2 error2
$= fxfixed\_point(f, xp, error, N);$

fprintf('Newton Method:'); [rt3 iteration3 error3]= fxNewtons_method(f,jj,xp,error,N);

fprintf('The root is obtianed at
fprintf('The root is obtianed at
fprintf('The root is obtianed at
h1=plot(rt1,0,'rs');
h2=plot(rt2,0,'mo');
h3=plot(rt3,0,'kp');
legend([h1,h2,h3],'Root found by Bisection method','Root found by Fixed-point method','Root found by Newton method')

$Thecodeofgxmaincodeprojectfinal.m$:

$clearall$
$closeall$
clc

$symsx$
$f(x) = log(x) - cos(x);$
$jj = diff(f, x);$
$a = 0.6; b = 2.6;$
$error = 1e - 3;$
$N = 100;$
$xp = (a + b)/2;$
$fplot(f, [a, b])$
$holdon$
$fplot(0, [ab]);$
$fprintf('BisectionMethod :');$
rt1 iteration1 error1
=
$fxbisectionMethod(f, a, b, error);$
$fprintf('FixedPointMethod :');$
rt2 iteration2 error2
$= fxfixed\_point(f, xp, error, N);$
$fprintf('NewtonMethod :'); [rt3iteration3error3] = fxNewtons\_method(f, jj, xp, error, N);$

fprintf('The root is obtianed at
fprintf('The root is obtianed at
fprintf('The root is obtianed at
h1=plot(rt1,0,'rs');
h2=plot(rt2,0,'mo');
h3=plot(rt3,0,'kp');
legend([h1,h2,h3],'Root found by Bisection method','Root found by Fixed-point method','Root found by Newton method')

The code of fxgxbisectionMethod.m:

$function[yiterationerr] = fxgxbisectionMethod(f, a, b, error)$
$xmin = a; xmax = b;$
$y = (a + b)/2;$

$err = abs(double(f(y)));$
$iteration = 0;$
$disp('----------')$
$fprintf('iteration\quad f(x)\quad |an+11-an|'); disp('----------$

$The\ code\ of\ fxgx fixed\_point.m:$

$function[p, iteration, err] = fxgx fixed\_point(f, xp, error, n)$
$iteration = 0;$
$c = double(feval(f, xp));$
$err = Inf;$
$disp('----------')$
$disp('iter\quad xf(x)\quad |xn+11-xn|')$
$disp('----------')$
$fprintf('$
$while(err > error)(iteration <= n)$
$x1 = c;$
$err = abs(x1 - xp);$
$xp = x1;$
$c = double(feval(f, xp));$
$iteration = iteration + 1;$
$fprintf('$
end
$if(iteration > n)$
$display('Method\ failed\ to\ converge')$
$p = xp;$
end
$p = xp;$
end

$The\ code\ of\ fxgx Newtons\_method.m:$

$function[p, n, err] = fxgxNewtons\_method(f, jj, xp, error, N)symsx$
$x\_iterations = xp;$
$err = Inf;$
$n = 1;$
$disp('--')$
$fprintf('iter æ\Gamma\Gamma f(x) æ\Gamma\Gamma |xn + 11 - xn|'); disp('---------------------------$

y=xp-(double(f(xp))/double(jj(xp)));

x$_iterations = [x_iterations; y];$
$err = abs(y - xp);$

xp=y;n=n+1;

fprintf('end
p=xp;

end