Halley's Method: A Cubically Converging Method of Root Approximation

Gabriel Kramer

Brittany Sypin

December 3, 2011

Abstract

This paper will discuss numerical ways of approximating the solutions to the equation $f(x) = 0$, where, for the purposes of this paper, $f(x) = ln(x)+x$. In particular, this paper will discuss the *bisection method*, *fixed-point iteration*, *Newtons method*, and *Halley's method*. Also, a convergence analysis will be done on each of these methods.

1 Introduction

In mathematics it is often desirable to solve the equation $f(x) = 0$ analytically, but often this cannot be done. Equations of this form occur often in algebra, calculus, physics, differential equations, chemistry, among others. When analytic solutions are not possible for these equations, numerical approximation methods are useful in obtaining approximate solutions.

2 Halley's Method

Halley's method is useful for finding a numerical approximation of the roots to the equation $f(x) = 0$ when $f(x)$, $f'(x)$, and $f''(x)$ are continuous. The *Halley's* method $n + 1$ recursive algorithm is given by:

$$x_{n+1} = x_n - \frac{2f(x_n)f'(x_n)}{2[f'(x_n)]^2 - f(x_n)f''(x_n)}$$

where x_0 is an initial guess.

2.1 Halley's Method Derivation

Halley's method considers the function

$$g(x) = \frac{f(x)}{\sqrt{|f'(x)|}}$$

and its derivative

$$g'(x) = \frac{2[f'(x)] - f(x)f''(x)}{2f'(x)\sqrt{|f(x)|}}$$

then using *Newton's method*[1] on $g(x)$ results in *Halley's method.*[2]. This can be done because, an assumption made in creating $g(x)$ is that the roots of $f(x)$ are not equal to the roots of $f'(x)$. Also, it is easy to see that $f(a) = 0$ if and only if $g(a) = 0$. The usefulness of considering $g(x)$ instead of $f(x)$, comes from the fact that *Newton's method* converges cubically when $g(x)$ is used; as opposed to using $f(x)$, for which the approximation converges quadratically.

[1] See Section 5.3 for a discussion of Newton's Method

[2] This method only works when the roots of the function are not the same as the roots of its derivative ($f'(r) \neq f(r) = 0$).

3 Analytic Cubic Convergence Analysis of Halley's Method

When using *Halley's Method* on a function where $f'''(x)$ is continuous, the iterates satisfy

$$|x_{n+1} - a| \leq K * |x_n - a|^3$$

where $f(a) = 0$ and k is a non-negative number. This will be shown by conducting a convergence analysis.

First consider r to be a root, such that $f(r) = 0$. Then by Taylor's theorem, one can say:

$$0 = f(r) = f(x_n) + f'(x_n)(r - x_n) + \frac{f''(x_n)}{2}(r - x_n)^2 + \frac{f'''(c_1)}{6}(r - x_n)^3$$

Taylor's theorem also gives:

$$0 = f(r) = f(x_n) + f'(x_n)(r - x_n) + \frac{f''(c_2)}{2}(r - x_n)^2$$

Where c_1 and c_2 are between r and x_n.

Next, multiply the first equation by $2f'(x_n)$:

$$0 = 2f'(x_n)f(r) = 2f(x_n)f'(x_n)+2[f'(x_n)]^2(r-x_n)+f'(x_n)f''(x_n)(r-x_n)^2+\frac{f'(x_n)f'''(c_1)}{3}(r-x_n)^3$$

and multiply the second equation by $f''(x_n)(r - x_n)$:

$$0 = f(x_n)f''(x_n)(r - x_n) + f'(x_n)f''(x_n)(r - x_n)^2 + \frac{[f''(c_2)]^2}{2}(r - x_n)^3$$

Now, subtracting the first equation from the second, results in:

$$0 = 2f(x_n)f'(x_n) + (2[f'(x_n)]^2 - f(x_n)f''(x_n))(r - x_n)$$

$$+(\frac{f'(x_n)f'''(c_1)}{3} - \frac{f''(x_n)f''(c_1)}{2})(r - x_n)^2$$

After rearranging the equation, it becomes:

$$r - x_n = \frac{-2f(x_n)f'(x_n)}{2[f'(x_n)]^2 - f(x_n)f''(x_n)} - \frac{2f'(x_n)f'''(c_1) - 3f''(x_n)f''(c_2)}{6(2[f'(x_n)]^2 - f(x_n)f''(x_n))}(r - x_n)^3$$

From how x_{n+1} is defined, the first term on the *right hand side* can be replaced with $[x_{n+1} - x_n]$:

$$r - x_n = [x_{n+1} - x_n] - \frac{2f'(x_n)f'''(c_1) - 3f''(x_n)f''(c_2)}{6(2[f'(x_n)]^2 - f(x_n)f''(x_n))}(r - x_n)^3$$

After rearranging again, the equation becomes:

$$r - x_{n+1} - = -\frac{2f'(x_n)f'''(c_1) - 3f''(x_n)f''(c_2)}{6(2[f'(x_n)]^2 - f(x_n)f''(x_n))}(r - x_n)^3$$

Next, consider

$$a = \lim_{x_n \to r} -\frac{2f'(x_n)f'''(c_1) - 3f''(x_n)f''(c_2)}{6(2[f'(x_n)]^2 - f(x_n)f''(x_n))}$$

$$a = -\frac{2f'(r)f'''(r) - 3f''(r)f''(r)}{6(2[f'(r)]^2)}$$

and

$$|a| = |\frac{2f'(r)f'''(r) - 3f''(r)f''(r)}{6(2[f'(r)]^2)}|$$

because $|a|$ is a fixed constant, *for all* $A \geq a$, it is possible to say:

$$|r - x_{n+1}| \leq |A(r - x_n)^3|$$

Thus proving that, *Halley's method* converges cubically.

4 Results

4.1 Coding Halley's Method

The approximation to the solution $ln(x) + x = 0$ using *Halley's method* can be coded in *MATLAB* where the inputs are an initial guess, x_0, and either the number of iterations or a stopping criteria (a tolerance).

4.1.1 Using a Given Number of Iterations

The *MATLAB* code to approximate the solution to the equation $ln(x) + x = 0$ using *Halley's Method* using an initial guess (p) and the number of iterations (k) is:

```
function halley (p,k)
fx=inline('y+log(y)');
dfx=inline('1+1/y');
ddfx=inline('0-1/y^2');
x=zeros(1,k);
x(1)=p;
for i=1:k
    x(i);
    x(i+1)=x(i)-2*fx(x(i))*dfx(x(i))/(2*[dfx(x(i))]^2-fx(x(i))*ddfx(x(i)));
end
```

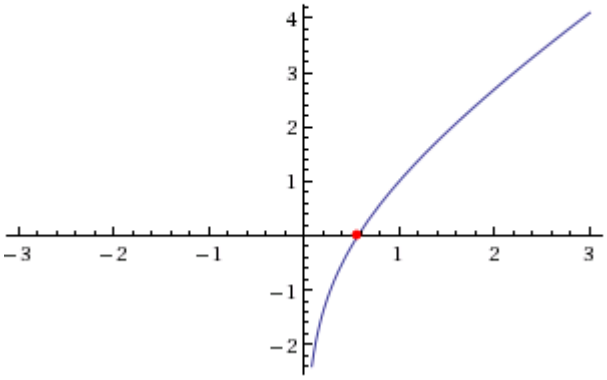
4.1.2 Using a Tolerance

The *MATLAB* code to approximate the solution to the equation $ln(x) + x = 0$ using *Halley's Method* using an initial guess (p) and the the number of digits correct (a tolerance, k) is:

```
function ans=halleytol(p,k)
fx=inline('y+log(y)');
dfx=inline('1+1/y');
ddfx=inline('0-1/y^2');
x=zeros(1,100);
x(1)=x(p)-2*fx(x(p))*dfx(x(p))/(2*[dfx(x(p))]^2-fx(x(p))*ddfx(x(p)));
x(2)=x(x(1))-2*fx(x(x(1)))*dfx(x(x(1)))/(2*[dfx(x(p))]^2-fx(x(p))*ddfx(x(p)));
i=1
while abs(x(p+1)-x(p))>5*10^(-k)
        if abs(x(p+1)-x(p))>5*10^(-k)
            break
        end
        x(p+1)=x(p)-2*fx(x(p))*dfx(x(p))/(2*[dfx(x(p))]^2-fx(x(p))*ddfx(x(p)));
        p=p+1;
end
ans=x(p+1)
```

4.2 Using the Code to Approximate the Roots to $ln(x) + x = 0$

When observing the graph of $f(x) = ln(x) + x$



$f(x) = ln(x) + x$

it is easy to see that the the function has a root around .5, so for the purpose of code testing, 1 will be used as the initial guess.

The results obtained by using a specified number of iterations are:

Table 1: Number of Iterations

k	approx
1	0.555555555555556
3	0.567143844033509
5	0.567143290409784
10	0.567143290409784
20	0.567143290409784

The results obtained by using a tolerance (number of digits correct) are:

Table 2: Number of correct digits (k), and number of iterations (i) to achieve result

k	i	approx
1	1	0.555555555555556
3	3	0.567143844033509
5	3	0.567143844033509
8	4	0.567143290409784
10	4	0.567143290409784
12	4	0.567143290409784

5 Roots of Functions

In this section, we will use different root finding methods to numerically approximate the solution to the equation $ln(x) + x = 0$.

5.1 Bisection Method

The *Bisection method* is a method similar to *Halley's method*, in that it approximates the solution to $f(x) = 0$, where $f(x)$ is continuous on some interval $[a, b]$. It does so by using the *intermediate value theorem*, which states that if $f(x)$ is continuous on $[a, b]$, then for all y such that $min(f(a), f(b)) \leq y \leq max(f(a), f(b))$, there exists a $c \in [a, b]$ such that $f(c) = y$.

In order to use the *bisection method* a and b must be chosen so that $f(a)f(b) \leq 0$. In other words, one of our endpoints must give a negative y value, and the other a positive y value. Since our function is continuous, the *intermediate value theorem* can be applied, because now it is known that there exists a $c \in [a, b]$ such that $f(c) = 0$.

The algorithm for approximating the root to $f(x) = 0$ using the *bisection method* is as follows:

Step 1: Find $[a, b]$ such that $f(a)f(b) < 0$[3].

Step 2: Let $c = \frac{a+b}{2}$.

Step 3: Consider $f(c)$, if $f(c) = 0$ then the root is c. If not, continue.

Step 4: Consider $f(a)f(c)$ and $f(c)f(b)$, if $f(a)f(c) < 0$, let $b = c$ and restart from **Step 2**, otherwise, let $a = c$ and restart from **Step 2** until $|a - b| <$ *error* for some specified error.

Using this method to approximate the solution to $ln(x) + x = 0$ with $[a, b] = [.5, .6]$, the following table was obtained by preforming i iterations of this method in *MATLAB* (code attached in appendix).

Table 3: Using the Bisection Method with $tol = 0.0000000005$ to approximate $ln(x) + x = 0$

i	approx
1	0.550000000000000
2	0.575000000000000
3	0.562500000000000
⋮	⋮
15	0.567143249511719
⋮	⋮
25	0.567143288254738
26	0.567143289744854
27	0.567143290489912

From the previous analysis of *Halley's method*, it is clear that the *bisection method* converges much slower than *Halley's method* (27 iterations verses 4 iterations).

[3]If $f(a)f(b) = 0$ then either $f(a) = 0$ or $f(b) = 0$ and the root has been found.

5.2 Fixed Point Iteration

Fixed Point Iteration is a recursive method to approximate the solution to the function $f(x) = 0$. It is unlike any of the other methods discussed in this paper, because while the goal of the other methods is to find an a such that $f(a) = 0$, the goal of *fixed point iteration* is to find an $\hat{a}$ such that $\hat{a} = g(\hat{a})$. For this method, the $n + 1$ iterating step is:

$$x_{n+1} = g(x_n)$$

with an initial guess x_0.

The algorithm for *fixed point iteration* is as follows:

Step 1: Given $f(x) = 0$, rewrite the equation to obtain an equation in the form $x = g(x)$.

Note 1: There are usually multiple ways to solve for $g(x)$, where some may converge and some may not.

Step 2: Choose an initial guess relatively close to the root.

Note 2: If the method does not converge for this particular guess, changing it may help.

Step 3: Insert into the recursive algorithm.

$$x_{n+1} = g(x_n)$$

Step 4: Repeat **Step 3** until $|x_{n+1} - x_n| < error$ for some specified error.

Note 3: This method does not always converge.

Note 4: This method may not generate all the roots.

Using this method to approximate the solution to $ln(x) + x = 0$ with $g(x) = e^{-x}$, the following table was obtained by setting $tol < .000000005$ in *MATLAB* (code attached in appendix).

Table 4: Using the Fixed Point Iteration Method with $tol = .000000005$ to approximate $ln(x) + x = 0$

i	approx
1	1.000000000000000
2	0.367869441171442
3	0.692299617555346
⋮	⋮
50	0.567143290409964
⋮	⋮
98	0.567143290409784
99	0.567143290409784
100	0.567143290409784

It is clear that this converges slower than both *bisection method* and *Halley's method* (100 iterations verses 27 and 4 respectively).

5.3 Newton's Method

Newton's Method is another method used to approximate the solution to $f(x) = 0$. It is very similar to *Halley's method* since *Halley's method* is derived from *Newton's method*. *Newton's* iteration step is given by:

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

where x_n is an initial guess.

Using this method to approximate the solution to $ln(x) + x = 0$, the following table was obtained by setting $tol < .000000005$ in *MATLAB* (code attached in

appendix).

Table 5: Using Newton's method with a specified number of iterations to approximate $ln(x) + x = 0$

i	approx
1	0.500000000000000
2	0.564382393519982
3	0.567138987715060
4	0.567143290399369
5	0.567143290409784
⋮	⋮
10	0.567143290409784

The table shows that *Newton's method* converges much faster than both the *bisection* and the *fixed point iteration* method as it reaches 8 digits of accuracy after only 4 iterations.

6 Numerical Convergence Analysis of Approximations

The following tables are constructed in *(MATLAB)* using the functions for the methods, and their corresponding error functions which are listed in the appendix. Also, the root of $ln(x) + x$ was calculated to be approximately 0.567143290409783 by *WolframAlpha*. This value is what is used to calculate e_i.

6.1 Bisection Method

i	x_i	$f(x_i)$	e_i	$e_i + 1/e_i$
1	0.575	0.021614762	0.00785671	0.590996823
2	0.5625	-0.012864145	0.00464329	0.346028236
3	0.56875	0.004435691	0.00160671	0.944968785
4	0.565625	-0.004198965	0.00151829	0.029118007
5	0.5671875	0.000122158	4.42096E-05	16.67150512
6	0.56640625	-0.002037452	0.00073704	0.470008707
7	0.566796875	-0.00095741	0.000346415	0.43618992
8	0.566992188	-0.000417567	0.000151103	0.353710328
9	0.567089844	-0.000147689	5.34467E-05	0.086413909
10	0.567138672	-1.27621E-05	4.61853E-06	4.286105582
11	0.567163086	5.46988E-05	1.97955E-05	0.383343984
12	0.567150879	2.09686E-05	7.5885E-06	0.195688414
13	0.567144775	4.10333E-06	1.48498E-06	1.05508228
14	0.567141724	-4.32936E-06	1.56678E-06	0.02610331
15	0.56714325	-1.1301E-07	4.08981E-08	17.6546589
16	0.567144012	1.99516E-06	7.22041E-07	0.471678864
17	0.567143631	9.41075E-07	3.40572E-07	0.439956741
18	0.56714344	4.14032E-07	1.49837E-07	0.36352463
19	0.567143345	1.50511E-07	5.44694E-08	0.124577377
20	0.567143297	1.87503E-08	6.78565E-09	2.513569821
21	0.567143273	-4.71301E-08	1.70562E-08	0.301079722
22	0.567143285	-1.41899E-08	5.13528E-09	0.160689713
23	0.567143291	2.28017E-09	8.25187E-10	2.611586792
24	0.567143288	-5.95487E-09	2.15504E-09	0.308545542
25	0.56714329	-1.83735E-09	6.64929E-10	0.120506315
26	0.56714329	2.21412E-10	8.01291E-11	3.649160144
27	0.56714329	-8.07969E-10	2.924E-10	0.362982361
28	0.56714329	-2.93278E-10	1.06135E-10	0.122522074
29	0.56714329	-3.59331E-11	1.30032E-11	2.580901562
30	0.56714329	9.27396E-11	3.3563E-11	0.306267594
31	0.56714329	2.84031E-11	1.02799E-11	0.13255927
32	0.56714329	-3.76521E-12	1.36169E-12	3.271897662
33	0.56714329	1.23189E-11	4.4591E-12	0.347171033
34	0.56714329	4.27669E-12	1.54865E-12	0.059823542
35	0.56714329	2.55795E-13	9.34808E-14	6.857314149
36	0.56714329	-1.7546E-12	6.34048E-13	0.427172583
37	0.56714329	-7.49512E-13	2.70339E-13	0.329512894
38	0.56714329	-2.47025E-13	8.84848E-14	0.01863354
39	0.56714329	4.55191E-15	2.55351E-15	26.33333333
40	0.56714329	-1.21236E-13	4.29656E-14	0.481012658
41	0.56714329	-5.83977E-14	2.02061E-14	0.457894737
42	0.56714329	-2.67564E-14	8.77076E-15	0.413793103
43	0.56714329	-1.11022E-14	3.10862E-15	0.305555556
44	0.56714329	-3.44169E-15	3.33067E-16	0.181818182
45	0.567143290	1.11022E-15	2.5	0
46	0.56714329	-1.66533E-15	3.33067E-16	0.2
47	0.567143290	7.77156E-16	1	0
48	0.567143290	9.99201E-16	0^{14}	0
49	0.567143290	8.88178E-16	65535	0
50	0.567143290	9.99201E-16	0	0

6.2 Newton's Method

i	x_i	$f(x_i)$	e_i	$e_i + 1/e_i$
1	0.5	-0.193147181	0.06714329	0.041119476
2	0.564382394	-0.007640861	0.002760897	0.001558441
3	0.567138988	-1.18893E-05	4.30269E-06	2.42053E-06
4	0.56714329	-2.87784E-11	1.04139E-11	0
5	0.56714329	0	8.88178E-16	0
6	0.56714329	0	8.88178E-16	0
7	0.56714329	0	8.88178E-16	0
8	0.56714329	0	8.88178E-16	0
9	0.56714329	0	8.88178E-16	0
10	0.56714329	0	8.88178E-16	0
11	0.56714329	0	8.88178E-16	0
12	0.56714329	0	8.88178E-16	0
13	0.56714329	0	8.88178E-16	0
14	0.56714329	0	8.88178E-16	0
15	0.56714329	0	8.88178E-16	0
16	0.56714329	0	8.88178E-16	0
17	0.56714329	0	8.88178E-16	0
18	0.56714329	0	8.88178E-16	0
19	0.56714329	0	8.88178E-16	0
20	0.56714329	0	8.88178E-16	0
21	0.56714329	0	8.88178E-16	0
22	0.56714329	0	8.88178E-16	0
23	0.56714329	0	8.88178E-16	0
24	0.56714329	0	8.88178E-16	0
25	0.56714329	0	8.88178E-16	0
26	0.56714329	0	8.88178E-16	0
27	0.56714329	0	8.88178E-16	0
28	0.56714329	0	8.88178E-16	0
29	0.56714329	0	8.88178E-16	0
30	0.56714329	0	8.88178E-16	0
31	0.56714329	0	8.88178E-16	0
32	0.56714329	0	8.88178E-16	0
33	0.56714329	0	8.88178E-16	0
34	0.56714329	0	8.88178E-16	0
35	0.56714329	0	8.88178E-16	0
36	0.56714329	0	8.88178E-16	0
37	0.56714329	0	8.88178E-16	0
38	0.56714329	0	8.88178E-16	0
39	0.56714329	0	8.88178E-16	0
40	0.56714329	0	8.88178E-16	0
41	0.56714329	0	8.88178E-16	0
42	0.56714329	0	8.88178E-16	0
43	0.56714329	0	8.88178E-16	0
44	0.56714329	0	8.88178E-16	0
45	0.56714329	0	8.88178E-16	0
46	0.56714329	0	8.88178E-16	0
47	0.56714329	0	8.88178E-16	0
48	0.56714329	0	8.88178E-16	0
49	0.56714329	0	8.88178E-16	0
50	0.56714329	0	8.88178E-16	0

6.3 Halley's Method

i	x_i	$f(x_i)$	e_i	$e_i + 1/e_i$
1	1	1	0.43285671	0.026770371
2	0.555555556	-0.032231109	0.011587735	4.77767E-05
3	0.567143844	1.52979E-06	5.53624E-08	0
4	0.56714329	0	8.88178E-16	0
5	0.56714329	0	8.88178E-16	0
6	0.56714329	0	8.88178E-16	0
7	0.56714329	0	8.88178E-16	0
8	0.56714329	0	8.88178E-16	0
9	0.56714329	0	8.88178E-16	0
10	0.56714329	0	8.88178E-16	0
11	0.56714329	0	8.88178E-16	0
12	0.56714329	0	8.88178E-16	0
13	0.56714329	0	8.88178E-16	0
14	0.56714329	0	8.88178E-16	0
15	0.56714329	0	8.88178E-16	0
16	0.56714329	0	8.88178E-16	0
17	0.56714329	0	8.88178E-16	0
18	0.56714329	0	8.88178E-16	0
19	0.56714329	0	8.88178E-16	0
20	0.56714329	0	8.88178E-16	0
21	0.56714329	0	8.88178E-16	0
22	0.56714329	0	8.88178E-16	0
23	0.56714329	0	8.88178E-16	0
24	0.56714329	0	8.88178E-16	0
25	0.56714329	0	8.88178E-16	0
26	0.56714329	0	8.88178E-16	0
27	0.56714329	0	8.88178E-16	0
28	0.56714329	0	8.88178E-16	0
29	0.56714329	0	8.88178E-16	0
30	0.56714329	0	8.88178E-16	0
31	0.56714329	0	8.88178E-16	0
32	0.56714329	0	8.88178E-16	0
33	0.56714329	0	8.88178E-16	0
34	0.56714329	0	8.88178E-16	0
35	0.56714329	0	8.88178E-16	0
36	0.56714329	0	8.88178E-16	0
37	0.56714329	0	8.88178E-16	0
38	0.56714329	0	8.88178E-16	0
39	0.56714329	0	8.88178E-16	0
40	0.56714329	0	8.88178E-16	0
41	0.56714329	0	8.88178E-16	0
42	0.56714329	0	8.88178E-16	0
43	0.56714329	0	8.88178E-16	0
44	0.56714329	0	8.88178E-16	0
45	0.56714329	0	8.88178E-16	0
46	0.56714329	0	8.88178E-16	0
47	0.56714329	0	8.88178E-16	0
48	0.56714329	0	8.88178E-16	0
49	0.56714329	0	8.88178E-16	0
50	0.56714329	0	8.88178E-16	0

6.4 Fixed Point Iteration

i	x_i	$f(x_i)$	e_i	$e_i + 1/e_i$
1	0.692200628	0.324321186	0.125057337	0.627596715
2	0.500473501	-0.191727127	0.06666979	0.533113781
3	0.606243535	0.105770035	0.039100245	0.586476195
4	0.545395786	-0.060847749	0.021747504	0.556198679
5	0.579612336	0.03421655	0.012469045	0.573355216
6	0.560115461	-0.019496874	0.007027829	0.563622073
7	0.571143115	0.011027654	0.003999825	0.56914086
8	0.564879347	-0.006263768	0.002263943	0.566010564
9	0.568428725	0.003549378	0.001285435	0.567785765
10	0.566414733	-0.002013992	0.000728557	0.566778934
11	0.567556637	0.001141904	0.000413347	0.567349939
12	0.566908912	-0.000647725	0.000234378	0.567026093
13	0.567276232	0.00036732	0.000132942	0.567209759
14	0.567067898	-0.000208334	7.5392E-05	0.567105594
15	0.56718605	0.000118152	4.27597E-05	0.56716467
16	0.56711904	-6.701E-05	2.42504E-05	0.567131165
17	0.567157044	3.80039E-05	1.37536E-05	0.567150167
18	0.56713549	-2.15538E-05	7.8002E-06	0.56713939
19	0.567147714	1.22241E-05	4.42385E-06	0.567145502
20	0.567140781	-6.9328E-06	2.50895E-06	0.567142036
21	0.567144713	3.93189E-06	1.42294E-06	0.567144002
22	0.567142483	-2.22995E-06	8.07008E-07	0.567142887
23	0.567143748	1.2647E-06	4.5769E-07	0.567143519
24	0.567143031	-7.17265E-07	2.59576E-07	0.567143161
25	0.567143438	4.06792E-07	1.47217E-07	0.567143364
26	0.567143207	-2.30709E-07	8.34929E-08	0.567143248
27	0.567143338	1.30845E-07	4.73524E-08	0.567143315
28	0.567143264	-7.4208E-08	2.68556E-08	0.567143276
29	0.567143306	4.20866E-08	1.5231E-08	0.5671433
30	0.567143282	-2.38691E-08	8.63815E-09	0.567143284
31	0.567143295	1.35372E-08	4.89907E-09	0.567143302
32	0.567143288	-7.67754E-09	2.77847E-09	0.567143289
33	0.567143292	4.35427E-09	1.57579E-09	0.56714333
34	0.56714329	-2.46949E-09	8.93699E-10	0.567143273
35	0.567143291	1.40056E-09	5.06857E-10	0.56714339
36	0.56714329	-7.94316E-10	2.87459E-10	0.567143096
37	0.567143291	4.50491E-10	1.63032E-10	0.567143479
38	0.56714329	-2.55493E-10	9.2461E-11	0.567143109
39	0.56714329	1.44901E-10	5.24402E-11	0.567144519
40	0.56714329	-8.21798E-11	2.97397E-11	0.567142534
41	0.56714329	4.66077E-11	1.68681E-11	0.567144121
42	0.56714329	-2.64331E-11	9.56513E-12	0.567137949
43	0.56714329	1.49913E-11	5.42622E-12	0.567145991
44	0.56714329	-8.5022E-12	0.566992187	0.567131193
45	0.56714329	4.82192E-12	1.74594E-12	0.567150177
46	0.56714329	-2.7347E-12	9.88765E-13	0.567120499
47	0.56714329	1.55098E-12	5.62217E-13	0.567197667
48	0.56714329	-8.7963E-13	3.17413E-13	0.567049051
49	0.56714329	4.99045E-13	1.81521E-13	0.567492152
50	0.56714329	-2.82996E-13	1.01474E-13	0

From these tables, it is easily observable that the *fixed point iteration* and *bisection methods* converge rather slowly when compared to *Halley* and *Newton's methods*.

7 Appendix

7.1 Bisection Method Codes

7.1.1 Code to Approximate the solution of f(x)=0, where r$\in [a,b]$

```
%Computes approximate solution of f(x)=0
%Input: inline function f; a,b such that f(a)*f(b)<0,
% and tolerance tol
%Output: Approximate solution xc
function xc = bisectnotol(f,a,b,k)
if sign(f(a))*sign(f(b)) >= 0
error('f(a)f(b)<0 not satisfied!') %ceases execution
end
fa=f(a);
fb=f(b);
for i=1:k
c=(a+b)/2;
fc=f(c);
if fc == 0 %c is a solution, done
break
end
if sign(fc)*sign(fa)<0 %a and c make the new interval
b=c;fb=fc;
else %c and b make the new interval
a=c;fa=fc;
end
end
xc=(a+b)/2; %new midpoint is best estimate
```

7.1.2 Bisection Method Error Code

```
%This function displays a table of errors as shown in section 1.2
%a is the beginning point of the interval
%b is the ending point of the interval
%k is the size of the desired number iterations
function ans=errb(a,b,k)
fx=inline('x+log(x)');
for i=1:k
err(i)=abs(0.5671432904-bisectnotol(fx,a,b,i));
err(i+1)=abs(0.5671432904-bisectnotol(fx,a,b,i+1));
ans(i)=err(i+1)/err(i);
end
```

7.2 Fixed Point Iteration Codes

7.2.1 Code to Approximate the solution of g(x)=x

```
%Computes approximate solution of g(x)=x
%Input: inline function g, starting guess x0,
% number of steps k
%Output: Approximate solution xc
function xc=fpi(g,x0,k)
x(1)=x0;
for i=1:k
x(i+1)=g(x(i));
end
e=x-1.81003792923*ones(1, length(x));
for j=2:k
s(j)=e(j)/e(j-1);
end
x'; %transpose output to a column
xc=x(k+1);
```

7.2.2 Fixed Point Iteration Error Code

```
%This function displays a table of errors as shown in section 1.2
%k is the size of the desired number iterations
function ans=errf(k)
gx=inline('exp(-x)');
for i=1:k
e(i)=abs(0.56714329040978387300-fpi(gx,1,i));
e(i+1)=abs(0.56714329040978387300-fpi(gx,1,i+1));
ans(i)=e(i+1)/e(i);
end
```

7.3 Halley's Method Codes

7.3.1 Code to Approximate the Solution of f(x)=0 With Specified Number of Iterations

```
%This program approximates the roots to the equation f(x)=0 using Halley's method
%Where p is an initial guess
%and k is the number of desired iterations
function halley (p,k)
fx=inline('y+log(y)');
dfx=inline('1+1/y');
ddfx=inline('0-1/y^2');
x=zeros(1,k);
x(1)=p;
for i=1:k
x(i);
x(i+1)=x(i)-2*fx(x(i))*dfx(x(i))/(2*[dfx(x(i))]^2-fx(x(i))*ddfx(x(i)));
end
x=zeros(1,k);
p=1
while abs(x(p+1)-x(p))>5*10^(-k)
if abs(x(p+1)-x(p))>5*10^(-k)
break
end
x(p+1)=x(p)-2*fx(x(p))*dfx(x(p))/(2*[dfx(x(p))]^2-fx(x(p))*ddfx(x(p)));
p=p+1;
end
```

7.3.2 Code to Approximate the Solution of f(x)=0 With a Given Tolerance

```
%This program approximates the roots to the equation f(x)=0 using Halley's method
%Where p is an initial guess
%and k is the number of desired correct digits
function ans=halleytol(p,k)
fx=inline('y+log(y)');
dfx=inline('1+1/y');
ddfx=inline('0-1/y^2');
x=zeros(1,100);
x(1)=(p)-2*fx(p)*dfx((p))/(2*[dfx((p))]^2-fx((p))*ddfx((p)));
x(2)=x(1)-2*fx(x(1))*dfx(x(1))/(2*[dfx(x(1))]^2-fx(x(1))*ddfx(x(1)));
i=2;
while
abs(x(i)-x(i-1))>.5*10^(-k)
x(i+1)=x(i)-2*fx(x(i))*dfx(x(i))/(2*[dfx(x(i))]^2-fx(x(i))*ddfx(x(i)));
i=i+1;
end
i
ans=x(i-1);
```

7.3.3 Halley's Method Error Code

```
%This function displays a table of errors as shown in section 1.2
%k is the size of the desired number iterations
%This program is run with an initial guess of 1
function ans= errh(k)
for i=1:k
err(i)=abs(0.56714329040978387300-(halley(1,i)));
err(i+1)=abs(0.56714329040978387300-halley(1,i+1));
ans(i)=err(i+1)/err(i);
end
```

7.4 Newton's Method Codes

7.4.1 Code to Approximate the Solution of f(x)=0 With a Specified Number of Iterations

```
%f is the objective function, which will be input inline
%df is the derivative of f, which will also be input inline
%p is an initial guess of r, where f(r)=0
%k is the number of desired iterations
function ans=newtonnotol(fx,dfx,p,k)
ans(1)=p-fx(p)/dfx(p);
x(1)=p;
for i=1:k
x(i+1)=x(i)-fx(x(i))/dfx(x(i)); %Newton's iteration step
ans=x(i+1);
end
```

7.4.2 Code to Approximate the Solution of f(x)=0 With a Given Tolerance

```
%f is the objective function, which will be input inline
%df is the derivative of f, which will also be input inline
%x0 is an initial guess of r, where f(r)=0
%tol is the backward tolarance
%max is the maximum number of iterations
function [n, err, x0,fx]=newton(f,df,x0, tol, max)
for n=1:max
fx=f(x0);
dfx=df(x0);
x1=x0-fx/dfx; %Newton's iteration step
err=abs(x1-x0);
x0=x1;
if abs(fx)<tol || err<tol
break
end
end
```

7.4.3 Newton's Method Error Codes

```
%This function displays a table of errors as shown in section 1.2
%p is an initial guess
%k is the size of the desired number iterations
function ans=errn(p,k)
for i=1:k
e(i)=abs(0.56714329040978387300-newtonnotol(inline('x+log(x)'),inline('1+1/x'),p,i));
e(i+1)=abs(0.56714329040978387300-newtonnotol(inline('x+log(x)'),inline('1+1/x'),p,i+1));
ans(i)=e(i+1)/e(i);
end
```

8 References

"Halleys Method." Halleys Method n.pag. Wikipedia. Web. 3 Dec 2011. <http://en.wikipedia.org/wiki/Halley>.

Sauer, Timothy. Numerical Analysis. Pearson Education, Inc., 2006. Print.

Weisstein, Eric W. "Halley's Method." From MathWorld–A Wolfram Web Resource. http://mathworld.wolfram.com/HalleysMethod.html