Matthew LeDuc
APPM 5600
Fall 2023
HW 3

1 Question 1

1. Let
$$T(x) = -16 + 6x + \frac{12}{x}.$$
Then we can calculate that
$$\frac{\mathrm{d}T}{\mathrm{d}x}(x_*) = 6 - \frac{12}{2^2} = 3$$
so by the continuity of $\frac{\mathrm{d}T}{\mathrm{d}x}$ there is no neighborhood of x_* where $\left|\frac{\mathrm{d}T}{\mathrm{d}x}\right| < 1$. This means that there is no neighborhood I of x_* such that we are guaranteed the fixed point iteration converges to x_* for any $x_0 \in I$.

2. We apply the contraction mapping theorem: Let $T(x) = \frac{2}{3}x + \frac{1}{x^2}$, then
$$\frac{\mathrm{d}T}{\mathrm{d}x} = \frac{2}{3} - \frac{2}{x^3}$$
Since $x_* = 3^{1/3}$, $\frac{\mathrm{d}T}{\mathrm{d}x}(x_*) = 0$ and so by the continuity of $\frac{\mathrm{d}T}{\mathrm{d}x}$, $\exists\ \delta > 0$ such that
$$|x - x_*| \leq \delta \implies \frac{\mathrm{d}T}{\mathrm{d}x}(x) \leq \frac{3}{4}.$$
Now let $I = [x_* - \delta, x_* + \delta]$, then the mean value theorem tells us that if $x \in I$ then
$$|T(x) - x_*| \leq \frac{3}{4}|x - x_*|.$$
This also implies that $T : I \to I$ and so by the Contraction Mapping Theorem we have that $x_0 \in I \implies T^n(x_0) = x_n \to x_*$, so there is a neighborhood of x_* such that the iterative scheme converges to x_* for any x_0 in the neighborhood as desired. Now we can use Taylor's Remainder Theorem to write that
$$T(x) = x_* + \frac{1}{2}T''(\eta)(x - x_*)^2$$
for η between x and x_*. This implies that, letting $x_n = T(x_{n-1})$,
$$\lim_{n\to\infty}\left|\frac{x_{n+1} - x_*}{(x_n - x_*)^2}\right| = \lim_{n\to\infty}\left|\frac{1}{2}T''(\eta_n)\right|$$
where η_n is between x_n and x_*. Since $x_n \to x_*$, by the Squeeze Theorem $\eta_n \to x_*$ and $T''(x) = 6x^{-4}$ is continuous at x_*, we have that
$$\lim_{n\to\infty}\left|\frac{x_{n+1} - x_*}{(x_n - x_*)^2}\right| = \frac{3}{3^{4/3}}$$
and therefore the convergence is second order.

3. Again we apply the contraction Mapping Theorem: Let $T(x) = \frac{12}{1+x}$, then

$$\frac{\mathrm{d}T}{\mathrm{d}x} = -\frac{12}{(1+x)^2}$$

Since $x_* = 3$, $\frac{\mathrm{d}T}{\mathrm{d}x}(x_*) = -\frac{3}{4}$ and by continuity we know that $\exists\ \delta > 0$ such that

$$|x - x_*| \leq \delta \implies \left|\frac{\mathrm{d}T}{\mathrm{d}x}(x) + \frac{3}{4}\right| < \frac{1}{8}$$

which implies that $\left|\frac{\mathrm{d}T}{\mathrm{d}x}(x)\right| \leq \frac{7}{8}\ \forall x \in I$ where $I = [x_* - \delta, x_* + \delta]$. This also implies by the mean value theorem that

$$|T(x) - x_*| \leq \frac{7}{8}|x - x_*|$$

and so we have that $T : I \to I$. So, we can apply the Contraction Mapping Theorem to tell us that if $x_0 \in I$, $T^n(x_0) = x_n \to x_*$.

By the Taylor Remainder Theorem we can write that

$$T(x) = x_* + T'(\eta)(x - x_*)$$

for η between x and x_*. Therefore, we can say that

$$\lim_{n\to\infty}\left|\frac{x_{n+1} - x_*}{x_n - x_*}\right| = \lim_{n\to\infty}|T'(\eta_n)|$$

where η_n is between x_n and x_*. By the Squeeze Theorem $\eta_n \to x_*$ and by continuity of $T'(x)$ we have that $|T'(\eta_n)| \to |T'(x_*)| = \frac{3}{4}$. Therefore, we have that

$$\lim_{n\to\infty}\left|\frac{x_{n+1} - x_*}{x_n - x_*}\right| = \frac{3}{4}$$

and therefore that the convergence is linear with rate $\frac{3}{4}$.

2 Question 2

1. We are given that the temperature as a function of depth and time satisfies the equation

$$\frac{T(x,t)-T_s}{T_i-T_s} = \text{erf}\left(\frac{x}{2\sqrt{\alpha t}}\right) \tag{2.1}$$

where t is the time in seconds after the cold snap, α is the thermal conductivity, T_i is the initial temperature, T_s and T_s is the temperature during the cold snap. All temperatures are in degrees Celsius. We can cast this as a root finding problem by finding the value of x such that $T(x,t)=0$ at $t_0 = 60$ days $= 5.184 \times 10^6$ seconds, as we wish to find the value of x_* such that $T(x_*,t_0)=0$. From Equation 2.1, we can see that this happens when

$$f(x) = (T_i - T_s)\,\text{erf}\left(\frac{x}{2\sqrt{\alpha t_0}}\right) + T_s = 0 \tag{2.2}$$

By the Fundamental Theorem of Calculus, since

$$\text{erf}(t) = \frac{2}{\sqrt{\pi}}\int_0^t \exp(-s^2)\text{d}s$$

we can calculate that

$$\frac{\partial f}{\partial x} = \frac{T_i - T_s}{\sqrt{\pi\alpha t_0}}\exp\left(-\frac{x^2}{4\alpha t_0}\right) \tag{2.3}$$

Since $f(1) \approx 5.9 > 0$, in Figure 2.1 is a plot of $f(x)$ on the interval $[0,1]$.

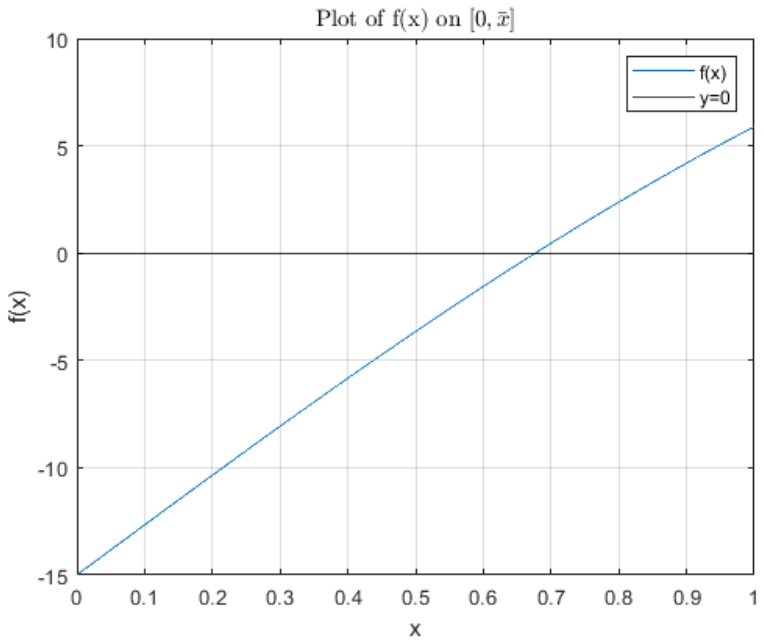


Figure 2.1: $f(x)$ on the interval $[0,1]$.

2. Using the bisection method, the root was found at approximately $x_{b*} = 0.67696$ meters.

3. Using Newton's Method with an initial guess $x_0 = 0.01$ meters, the root was found at approximately $x_{N*} = 0.67696$ meters. After this, I ran 100 trials of the bisection method and Newton's method using each starting point and timed them using the `MATLAB` functions `tic` and `toc`. Using these, the bisection code ran in an average of 6.27×10^{-5} seconds, and the Newton's method codes ran in about 2.57×10^{-5} seconds and 2.52×10^{-5} seconds for $x_0 = 0.01$ and 1 respectively. So, if we have a good idea of the location of the root it seems like the Newton's Method works best. But, if for example we set $\bar{x} = 10$, the Newton's Method code will run away to ∞ while the bisection code still converges, although more slowly than before. So, if we do not know the location of the root well, it may be preferable to choose the bisection method since we can find the sign change easily.

2.1 Code

The run script:

```
%% HW 3 Problem 2
addpath( '../HW 2')
t = 60*(3600*24);%Time
alpha = 0.138e-6;%Thermal conductivity
Ti = 20;%Initial temperature
Ts = -15;%Snap temperature
tol = 1e-13;%Tolerance, relative error
interval = [0,1];%The interval [0,\bar{x}]
fh = @(x)getTemperature(x, t, alpha, Ti, Ts);
fph = @(x)Tprime(x, t, alpha, Ti, Ts);
maxIters = 100;
%% Run the codes
[root_bisect,cnt, errMsg] = bisection(fh, [0,1], tol, maxIters);
fprintf('Bisection: Error Message %d, Approximate root %.5f\n Next: Newton with x0=0.0
[root_n1,errMsgN1,x1] = newtonsMethod(fh, fph, 0.01, tol, maxIters);
fprintf('Newtons Method, initial point 0.01: Error Message %d, Approximate root %.5f\n
[root_n2,errMsgN2,x2] = newtonsMethod(fh, fph, interval(2), tol, maxIters);
fprintf('Newtons Method, initial point %d: Error Message %d, Approximate root %.5f\n',
%% Speed test
nRuns = 100;
n001Times = zeros(1, nRuns);
n1Times = zeros(1, nRuns);
bisectTimes = zeros(1, nRuns);
for ii = 1:nRuns
    tic;
    [root_bisect,cnt, errMsg] = bisection(fh, [0,1], tol, maxIters);
    bisectTimes(ii)=toc;
    tic;
    [root_n1,errMsgN1,x1] = newtonsMethod(fh, fph, 0.01, tol, maxIters);
    n001Times(ii) = toc;
    tic;
```

```
    [root_n2,errMsgN2,x2] = newtonsMethod(fh, fph, interval(2), tol, maxIters);
    n1Times(ii) = toc;
end
```

Temperature and its derivative:

```
function temp = getTemperature( x, t, alpha, Ti, Ts )
%% Function for calculating the ground temperature
%Inputs:
%   x: depth, m, positive down
%   t, time after cold snap, seconds
%   alpha: thermal conductivity, m^2/s
%   Ti: Temperature before cold snap, deg C
%   Ts: Temperature during cold snap, deg C
%Outputs:
%   temp: Soil temperature at input parameters, deg C
deltaT = Ti-Ts;
temp = erf( x./(2*sqrt( alpha*t ) ))*deltaT+Ts;
end


function dTdx = Tprime(x, t, alpha, Ti, Ts)


delta = Ti-Ts;
dTdx = 1/sqrt(pi*alpha*t)*exp(-x^2/(4*alpha*t))*delta;
end
```

Newton's Method:

```
function [root, errMsg, x] = newtonsMethod(f, fp, x0, tol, maxIters, varargin)
%% Code for running Newton's method for rootfinding on a 1D function
%Inputs:
%   f: Function handle, the function we are finding the root of
%   fp: Function handle, the derivative of f
%   x0: Scalar, initial gruess
%   tol: scalar, relative error tolerance
%   maxIters: Scalar, maximum number of iterations allowed
%   varargin: Should be either the multiplicity of the root, or empty.
%Outputs:
%   root: scalar, approximate root location
%   errMsg: Flag that contains information on success/failure of the method
%       0: Success
%       1: Failure, took too many iterations
%       2: Failure, ran off to infinity (i.e. because the function was too
%       flat near the initial guess)
%   x: vector x_n, the sequence of approximations of the root
if ~isempty(varargin)
     p = varargin{1};
else
    p = 1;
end
```

```
x = zeros(1, maxIters);x(1)=x0;
cnt = 1;
del = tol+1;
while del>tol && cnt<maxIters && ~isnan(del) && f(x(cnt))~=0
    dx = p*f(x(cnt))/fp(x(cnt));
    x(cnt+1) = x(cnt)-dx;
    del = abs(dx/(x(cnt+1)));
    cnt=cnt+1;
end
x = x(1:cnt);
root = x(end);
if cnt == maxIters
    errMsg = 1;
elseif isnan(del)
    errMsg = 2;
else
    errMsg = 0;
end
end
```

Bisection:

```
function [root, cnt, errNum, errMsg] = bisection(fHandle, interval, tol, maxIters)
%BISECTION Implementation of bisection method
% Error messages:
%   0:Success
%   1:Root not found due to no detected sign change on interval
%   2:Root not found, max iterations reached
a = interval(1);b = interval(2);%
fa = fHandle(a);
fb = fHandle(b);
if fa*fb>0 %Handling the case where sign(a) and sign(b) are different
    newPts = linspace(a, b, 5);%checking if the initial choice of a and b
                               %were just bad
    for ii = 1:length(newPts)-1
        if fHandle(newPts(ii))*fHandle(newPts(ii+1))<0
            a = newPts(ii);%If we find a sign change in one of the new
                           %intervals we are cooking
            b = newPts(ii+1);
            fa = fHandle(a);
            fb = fHandle(b);
            rootExists = 1;
            break
        else
            rootExists = 0;
        end
    end
    if ~rootExists
        %If we don't find a sign change, output the midpoint
        root = (a+b)/2;
        cnt = 0;
        errNum = 1;
        errMsg = "Root not found: No sign change detected";
```

```
            return
        end
    end
    del = tol+1;
    cnt = 0;
    while del>tol && cnt<maxIters%Bisection method
        cnt = cnt+1;
        c = (a+b)/2;
        fc = fHandle(c);
        if fa*fc<0
            b = c;
            fb = fc;
        else
            a = c;
            fa = fc;
        end
        del = abs( (a-b)/b );
    end
    root = (a+b)/2;
    if cnt<maxIters
        errNum = 0;
        errMsg = "Successful";
    else
        errNum = 2;
        errMsg = "Root not found: Maximum iterations reached";
    end
    end
```

3 Question 3

1. Since we know that $f(x)$ is cubic with real roots α, β, γ we know that

$$f(x) = k(x-\alpha)(x-\beta)(x-\gamma)$$

where $k \in \mathbb{R}$. So, Newton's Method tells us that, if we start the iteration at $x_0 = \frac{1}{2}(\alpha+\beta)$,

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$

. We evaluate $f(x_0)$ and $f'(x_0)$ to find that

$$f(x_0) = k\left(\frac{1}{2}(\alpha+\beta)-\alpha\right)\left(\frac{1}{2}(\alpha+\beta)-\beta\right)\left(\frac{1}{2}(\alpha+\beta)-\gamma\right)$$

$$= -\frac{k}{4}(\alpha-\beta)^2\left(\frac{1}{2}(\alpha+\beta)-\gamma\right) = -\frac{k}{4}(\alpha-\beta)^2(x_0-\gamma)$$

and that

$$f'(x_0) = k\left[\frac{1}{2}(\alpha-\beta)\left(\frac{1}{2}(\alpha+\beta)-\gamma\right)+\frac{1}{2}(\beta-\alpha)\left(\frac{1}{2}(\alpha+\beta)-\gamma\right)-\frac{1}{4}(\beta-\alpha)^2\right]$$

$$= -\frac{k}{4}(\alpha-\beta)^2.$$

Therefore, plugging these into the expression for Newton's Method, we find that

$$x_1 = x_0 - \frac{-\frac{k}{4}(\alpha-\beta)^2(x_0-\gamma)}{-\frac{k}{4}(\alpha-\beta)^2} = x_0 - x_0 + \gamma = \gamma$$

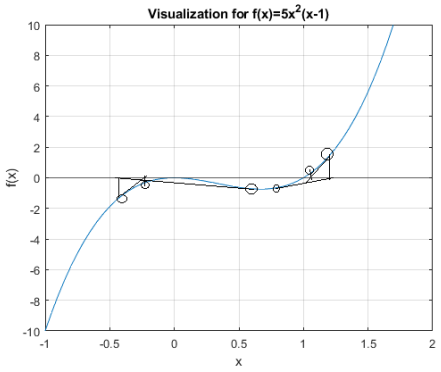
as desired, so the Newton's Method converges in one iteration.

2. If two roots coincide, then there will be exactly one point x_0 (other than the repeated root) such that $f'(x_0) = 0$, and so starting at x_0 will cause the method to fail. However, everywhere else the slope is not zero, and the roots of the tangent line will get successively closer to the root of $f(x)$ on the same side of x_0 as the initial guess, and in fact it will end up to the left of the smallest root if the initial guess is to the left of x_0, and to the right of the largest root if the initial guess is to the right of x_0. So, the point x_0 separates the basins of attraction of the two roots. In addition, there is no other point x_n such that

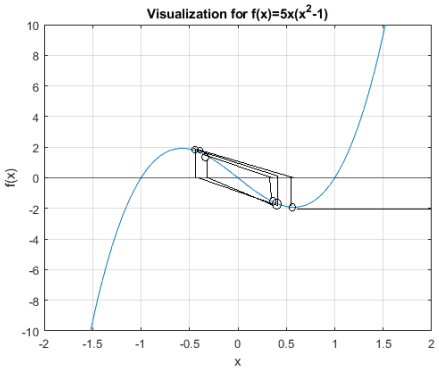
$$x_0 = x_n - \frac{f(x_n)}{f'(x_n)}$$

and so there is only one initial guess other than the repeated root that will cause the iteration to fail, and that point is x_0. This is illustrated in Figure 3.1a, where 3 iterations are plotted for two starting points, one to the left of x_0 and one to the right.

3. There are two points $x_{0,1}$ and $x_{0,2}$ at which $f'(x) = 0$, and landing there will cause the Newton iteration to fail. However, for each of these points there are infinitely many points such that the iteration will reach them if started there, so there are infinitely many points that will cause the iteration to fail. We can see this by selecting one of the roots and backtracking the iteration: if we backtrack, we find that these points are all contained between $x_{0,1}$ and $x_{0,2}$ and form a spiral like shape that eventually ends at one of the $x_{0,k}$ and then goes to infinity. This is illustrated in Figure 3.1b.



(a) 3 iterations of Newton's Method for an example cubic.



(b) Iterations of Newton's Method converging to a point of failure.

4 Question 4

1. Since $f(x) = (x - x_*)^p q(x)$,

$$\frac{\mathrm{d}f}{\mathrm{d}x} = p(x - x_*)^{p-1} q(x) + (x - x_*)^p \frac{\mathrm{d}q}{\mathrm{d}x}$$

and so we can implement Newton's Method as

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

$$= x_n - \frac{(x_n - x_*) q(x_n)}{p q(x_n) + (x_n - x_*) q'(x_n)}$$

. The convergence is of order m if

$$0 < \lim_{n\to\infty} \frac{|x_{n+1} - x_*|}{|x_n - x_*|^m} = R < \infty$$

. Plugging in our expression for x_{n+1} we can find that

$$\frac{|x_{n+1} - x_*|}{|x_n - x_*|} = \frac{\left| x_n - \alpha - \frac{(x_n - \alpha) q(x_n)}{p q(x_n) + (x_n - \alpha) q'(x_n)} \right|}{|x_n - \alpha|}$$

$$= \left| 1 - \frac{q(x_n)}{p q(x_n) + (x_n - \alpha) q'(x_n)} \right|$$

Therefore,

$$\lim_{n\to\infty} \frac{|x_{n+1} - x_*|}{|x_n - x_*|} = \lim_{n\to\infty} \left| 1 - \frac{q(x_n)}{p q(x_n) + (x_n - \alpha) q'(x_n)} \right| = 1 - \frac{1}{p}.$$

If $p > 1$, this tells us that the convergence is exactly linear, and if $p = 1$ the root has multiplicity 1 and we already know that the convergence is quadratic. So, we have that $\forall p$, the convergence is no worse than linear.

2. We wish to show that $|e_{n+1}| \leq C|e_n|^2$ where $e_n = x_n - x_*$ when $f(x)$ is of the form in part 1 and the iteration is

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

Using the expression derived in part 1 and noting that $|e_n| = |x_n - x_*|$,

$$|e_{n+1}| = \left| x_n - \alpha - \frac{f(x_n)}{f'(x_n)} \right| = \left| e_n - \frac{p e_n^p q(x_n)}{p e_n^{p-1} q(x_n) + e_k^p q'(x_n)} \right|$$

$$= \left| \frac{p e_n^p q(x_n) - p e_n^p q(x_n) + e_n^{p+1} q'(x_n)}{p e_n^{p-1} q(x_n) + e_n^p q'(x_n)} \right| = \left| \frac{e_n^{p+1} q'(x_n)}{p e_n^{p-1} q(x_n) + e_n^p q'(x_n)} \right|$$

$$= \left| \frac{e_n^2 q'(x_n)}{pq(x_n) + e_n q'(x_n)} \right|$$

Then, as $x_n \to x_*$, we get that $e_n \to 0$ and so we have that

$$|e_{n+1}| \leq \left| \frac{e_n^2 q'(x_*)}{pq(x_*)} \right|$$

, and so we have the desired result for

$$C = \left| \frac{q'(x_*)}{pq(x_*)} \right|$$

3. The desired tables for the Newton's Method and Modified Newton's Method are shown in Figures 4.1a and 4.1b respectively. The plots of error vs iteration number are shown in Figures 4.2a and 4.2b respectively. In those plots, since I used the `semilogy` command to plot the error since the 6th iteration of the modified Newton's method gets to $x_k = 1$ exactly, the error is zero and so that point gets cut off of the log plot. Note that since $x_* = 1$ the relative error and absolute error are the same.

Iteration	x_k	relative error
1	0	1
11	0.914946281251782	0.0850537187482183
21	0.991039742964181	0.00896025703581937
31	0.999039822288404	0.000960177711596288
41	0.999896923792645	0.00010307620735539
51	0.99998893253109	1.10674689104417e-05
61	0.999998811642509	1.18835749052248e-06
71	0.99999987240112	1.27598880128588e-07
81	0.999999986299175	1.37008250300497e-08
91	0.999999998528885	1.47111489656737e-09
101	0.99999999984204	1.57959756386106e-10
111	0.999999999983039	1.69608771471985e-11
121	0.999999999998179	1.82109882729264e-12
131	0.999999999999804	1.9551027463649e-13
141	0.999999999999979	2.1094237467878e-14
151	0.999999999999998	2.33146835171283e-15
160	1	3.33066907387547e-16

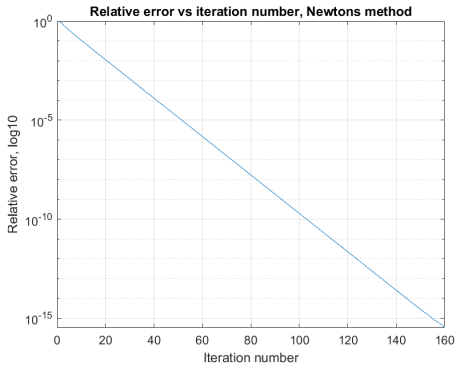
(a) Table of error vs iteration number and x_k, Newton's Method

Iteration	x_k	relative error
1	0	1
2	1.25	0.25
3	1.01190476190476	0.0119047619047619
4	1.00002827734419	2.82773441917517e-05
5	1.00000000015992	1.59920743314501e-10
6	1	0

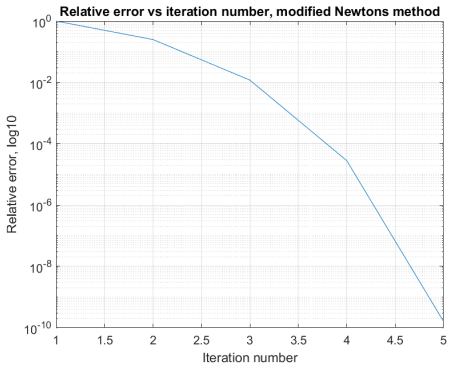
(b) Table of error vs iteration number and x_k, Modified Newton's Method

We can see that the figures and tables both agree with the theory: We get much slower convergence using the regular Newton's Method near the repeated root than we do with the Modified Newton's Method, and error does seem to be decreasing linearly. However with Modified Newton's Method we converge in much fewer iterations. In addition, by examining the plot we see that at iteration 2 we have error that is approximately 10^{-1}, then at iterations 3, 4, and 5 the error is approximately $10^{-2}, 10^{-5}$, and 10^{-10} (within a factor of 10) so the error is approximately decreasing quadratically, i.e. $e_{n+1} \sim e_n^2$, which lines up with the theory.

4.1 Code



(a) Plot of error vs iteration number, Newton's Method



(b) Plot of error vs iteration number, Modified Newton's Method

```
%% HW3 Problem 4
clear variables;close all
format long
fh = @(x)(x-1)^5*exp(x); %f(x)
fph = @(x)(x-1)^5*exp(x)+5*(x-1)^4*exp(x);%f'(x)
tol = 10^(-16);%Tolerance
x0=0;%STarting point
maxIters = 1000;%Maximum number of iterations
[root_newton, err_newton, x_newton] = newtonsMethod(fh, fph, x0, tol, maxIters);
figure; semilogy(1:length(x_newton), abs(x_newton-1) )
xlabel('Iteration number');ylabel('Relative error, log10')
grid on
title('Relative error vs iteration number, Newtons method')
saveas(gcf, 'fig_newtons.png');
savefig('fig_newtons.fig')
resN = [[1:10:length(x_newton),length(x_newton)];[x_newton(1:10:end),x_newton(end)];...
    abs([x_newton(1:10:end),x_newton(end)]-1/1) ].';
tableN = array2table(resN, 'variablenames',{'Iteration','x_k', 'relative error'});

[root_mn, err_mn, x_mn] = newtonsMethod(fh, fph, x0, tol, maxIters, 5);
figure; semilogy(1:length(x_mn), abs(x_mn-1) )
xlabel('Iteration number');ylabel('Relative error, log10')
title('Relative error vs iteration number, modified Newtons method')
grid on
saveas(gcf, 'fig_mod.png');
savefig('fig_mod.fig')
resM = [1:length(x_mn);x_mn;abs( x_mn-1 )/1 ].';
tableM = array2table(resM, 'variablenames',{'Iteration','x_k', 'relative error'});
```

Newton's Method (Note: I used a `varargin` to allow me to run both regular and modified Newton's Method with the same code)

```
function [root, errMsg, x] = newtonsMethod(f, fp, x0, tol, maxIters, varargin)
%% Code for running Newton's method for rootfinding on a 1D function
%Inputs:
%   f: Function handle, the function we are finding the root of
%   fp: Function handle, the derivative of f
```

```
%   x0: Scalar, initial gruess
%   tol: scalar, relative error tolerance
%   maxIters: Scalar, maximum number of iterations allowed
%   varargin: Should be either the multiplicity of the root, or empty.
%Outputs:
%   root: scalar, approximate root location
%   errMsg: Flag that contains information on success/failure of the method
%       0: Success
%       1: Failure, took too many iterations
%       2: Failure, ran off to infinity (i.e. because the function was too
%       flat near the initial guess)
%   x: vector x_n, the sequence of approximations of the root
if ~isempty(varargin)
     p = varargin{1};
else
    p = 1;
end
x = zeros(1, maxIters);x(1)=x0;
cnt = 1;
del = tol+1;
while del>tol && cnt<maxIters && ~isnan(del) && f(x(cnt))~=0
    dx = p*f(x(cnt))/fp(x(cnt));
    x(cnt+1) = x(cnt)-dx;
    del = abs(dx/(x(cnt+1)));
    cnt=cnt+1;
end
x = x(1:cnt);
root = x(end);
if cnt == maxIters
    errMsg = 1;
elseif isnan(del)
    errMsg = 2;
else
    errMsg = 0;
end
end
```

5 Question 5

Let x_0 and x_1 be two successive points from applying secant method to the problem $f(x) = 0$, with $f_n = f(x_n)$. Then the next iterate is given by

$$x_2 = x_1 - f_1 \frac{x_1 - x_0}{f_1 - f_0}. \tag{5.1}$$

We wish to show that the value of x_2 will be the same regardless of whether x_0 or x_1 is considered to be the previous iterate. To this end, we can write that

$$x_1 - f_1 \frac{x_1 - x_0}{f_1 - f_0} = \frac{(f_1 - f_0)x_1 - f_1(x_1 - x_0)}{f_1 - f_0} = \frac{-f_0 x_1 + f_1 x_0}{f_1 - f_0}.$$

Adding and subtracting a factor of $f_0 x_0$ to the numerator, we can rewrite this as

$$\begin{aligned}\frac{-f_0 x_1 + f_1 x_0}{f_1 - f_0} &= \frac{-f_0 x_1 + f_1 x_0 + f_0 x_0 - f_0 x_0}{f_1 - f_0} = \frac{x_0(f_1 - f_0)}{f_1 - f_0} + \frac{f_0(x_1 - x_0)}{f_1 - f_0} \\ &= x_0 - f_0 \frac{x_1 - x_0}{f_1 - f_0} = x_0 - f_0 \frac{x_0 - x_1}{f_0 - f_1}\end{aligned}$$

By Equation 5.1, we have that

$$x_2 = x_1 - f_1 \frac{x_1 - x_0}{f_1 - f_0} = x_0 - f_0 \frac{x_0 - x_1}{f_0 - f_1} \tag{5.2}$$

so, regardless of whether we choose x_0 or x_1 as our initial iterate, we will get the same value of x_2.