1 Book Problems

3.1.) Apply the bisection routine to find the root of the function

$$f(x) = \sqrt{x} - 1.1$$

starting from interval $[0, 1]$ with tolerance equal to $1e - 8$.

(a) How many iterations are required?

Solution. At least $n = 26$ are required to converge within the tolerance. This is obtained from $n = \log(\frac{b-a}{2 \cdot tol})$.

(b) What is the resulting absolute error? Could this absolute error be predicted by our convergence analysis?

Solution. The absolute error is $6.556510889765832e - 09$. We could have clearly predicted that it would be below $1e - 8$, otherwise we would not have stopped.

3.3.) Consider the fixed point iteration $x_{k+1} = g(x_k)$ and let all the assumptions of the fixed point theorem hold. Use a Taylor's series expansion to show that the order of convergence depends on how many of the derivatives of g vanish at $x = x^*$. User your result to state how fast (at least) a fixed point iteration is expected to converge if $g'(x^*) = \cdots = g^{(r)}(x^*) = 0$ where the integer $r \geq 1$ is given.

Solution. Consider the absolute error of the fixed point iteration above $e_k = |x_k - x^*|$. It follows that

$$e_{k+1} = |x_{k+1} - x^*| = |g(x_k) - g(x^*)|$$

because x^* is our fixed point. Expanding $g(x_k)$ using a Taylor series gives us

$$e_{k+1} = |-g(x^*) + g(x^*) + g'(x^*)(x_k - x^*) + g''(x^*)(x_k - x^*)^2 + \ldots g^{(r+1)}(\eta)(x_k - x^*)^r|$$

Then, if $g'(x^*) = \cdots = g^{(r)}(x^*) = 0$, we obtain

$$e_{k+1} = g^{(r+1)}(\eta){e_k}^{r+1}$$

In general, if the first r derivatives of g at x^* are 0, then the rate of convergence is $r + 1$. This is precisely why Newton's Method has quadratic convergence.

3.4.) Consider the function $g(x) = x^2 + \frac{3}{16}$.

(a) This function has two fixed points. What are they?

Solution. Solving $x = x^2 + \frac{3}{16}$ yields $x = \{1/4, 3/4\}$.

(b) Consider the fixed point iteration $x_{k+1} = g(x_k)$ for this g. For which of the points you have found in (a) can you be sure that the iterations will converge to that fixed point? Briefly justify your answer. You may assume that the intial guess is sufficiently close to the fixed point.

Solution. Evaluating $g'(x) = 2x$ at x_1, x_2 gives us

$$g'(x_1) = 0.5$$
$$g'(x_2) = 1.5$$

Since $g'(x_1) < 1$, we know that if g is a contraction, then the fixed point iteration will converge to x_1 for sufficiently close x_0. However, since $g'(x_2) > 1$, fixed point iteration may not necessarily converge.

(c) For the point or points you found in (b), roughly how many iterations will be required to reduce the convergence error by a factor of 10?

Solution. Since $g'(x_1) = 0.5$, the convergence rate will be exactly the same as bisection. That is, it will take at least 4 iterations to reduce it by a factor of 10. This could be manually done by computing $\frac{-1}{\log_{10} 0.5}$

3.5.) Write a MATLAB script for computing the cube root of a number with only basic operations using Newton's method. Run your program for $a = 0, 2, 10$. For each of these cases, start with an initial guess reasonably close to the solution. Print out the values of x_k and $f(x_k)$ in each iteration. Comment on the convergence rates and explain how they match your expectations.

Solution. Below are the outputs of my code.

```
For a = 0
x_0 = 1.157613e+00 f(x_0) = 1.551280e+00
x_1 = 7.717421e-01 f(x_1) = 4.596386e-01
x_2 = 5.144947e-01 f(x_2) = 1.361892e-01
x_3 = 3.429965e-01 f(x_3) = 4.035236e-02
x_4 = 2.286643e-01 f(x_4) = 1.195626e-02
x_5 = 1.524429e-01 f(x_5) = 3.542594e-03
x_6 = 1.016286e-01 f(x_6) = 1.049658e-03
x_7 = 6.775239e-02 f(x_7) = 3.110096e-04
x_8 = 4.516826e-02 f(x_8) = 9.215100e-05
x_9 = 3.011217e-02 f(x_9) = 2.730400e-05
x_10 = 2.007478e-02 f(x_10) = 8.090074e-06
x_11 = 1.338319e-02 f(x_11) = 2.397059e-06
x_12 = 8.922125e-03 f(x_12) = 7.102397e-07
x_13 = 5.948084e-03 f(x_13) = 2.104414e-07
x_14 = 3.965389e-03 f(x_14) = 6.235301e-08
x_15 = 2.643593e-03 f(x_15) = 1.847496e-08
x_16 = 1.762395e-03 f(x_16) = 5.474064e-09


For a = 2
x_0 = 2.157613e+00 f(x_0) = 8.044324e+00
x_1 = 1.581615e+00 f(x_1) = 1.956418e+00
x_2 = 1.320916e+00 f(x_2) = 3.047598e-01
```

```
x_3 = 1.262694e+00 f(x_3) = 1.323550e-02
x_4 = 1.259927e+00 f(x_4) = 2.898328e-05
x_5 = 1.259921e+00 f(x_5) = 1.400027e-10


For a = 10
x_0 = 6.157613e+00 f(x_0) = 2.234733e+02
x_1 = 4.192989e+00 f(x_1) = 6.371757e+01
x_2 = 2.984923e+00 f(x_2) = 1.659495e+01
x_3 = 2.364070e+00 f(x_3) = 3.212376e+00
x_4 = 2.172475e+00 f(x_4) = 2.533126e-01
x_5 = 2.154584e+00 f(x_5) = 2.080340e-03
x_6 = 2.154435e+00 f(x_6) = 1.442271e-07
x_7 = 2.154435e+00 f(x_7) = 1.776357e-15
```

The output is exactly as expected. It is easily seen that for $a = 2, 10$, the convergence is roughly quadratic. The exponent on the error (or in this case, $f(x_k)$) is doubling with each iteration, up to machine precision. However, $a = 0$ does not exhibit this characteristic. It takes 16 iterations before the difference in iterates has fallen below 10^{-8}. This is because $f(x) = x^3 - 0 = x^3$ has a repeated root, which leads to linear convergence. The rate of convergence in this case is $\rho = \frac{3-1}{3}$ which is roughly the case seen in the experiment.

3.7.) Consider Steffensen's method

$$x_{k+1} = x_k - \frac{f(x_k)}{g(x_k)}$$

where

$$g(x) = \frac{f(x + f(x)) - f(x)}{f(x)}$$

(a) Show that in general, the method converges quadratically to a root of $f(x)$.

Proof. Let $F(x) = x - \frac{f(x)^2}{f(x+f(x))-f(x)}$. Then Steffensen's method is simply the fixed point iteration with $g = F$. From question 3.3, we know that if $F'(x^*) = 0$, then Steffensen's method will converge quadratically. Thus, it suffices to show that $F'(x^*) = 0$.
Via Taylor expansion, we can write $f(x + f(x)) = f(x) + f'(x)f(x) + \frac{f''(\eta)}{2}f(x)^2$. Now consider

$$F(x) = x - \frac{f(x)}{f'(x) + \frac{f''(\eta)}{2}f(x)}$$

Through elementary operations, we can reformulate this to be

$$\begin{aligned}\frac{F(x) - F(x^*)}{x - x^*} &= x - x^* - \frac{f(x) - f(x^*)}{f'(x) + \frac{f''(\eta)}{2}f(x)} \\ &= 1 - \frac{f(x) - f(^*)}{x - x^*} \cdot \frac{1}{f'(x) + \frac{f''(\eta)}{2}f(x)}\end{aligned}$$

Now, letting $x \to x^*$ gives

$$F'(x^*) = 1 - \frac{f'(x^*)}{f'(x^*)} = 0$$

which was our intent to show. □

(b) Compare the method's efficiency to the efficiency of the secant method.

Solution. First and foremost, the secant method has superlinear convergence whereas Steffensen's is quadratic. Furthermore, the secant method requires computation of $f(x_k)$ and $f(x_{k-1})$ (and thus storage of previous values). Steffensen's requires $f(x_k)$ and $f(x_k + f(x_k))$ but no storage. Steffensen's does require more operations per iteration, but to a negligible amount. The most significant point of caution for Steffensen's method is when we are close to the solution. When $f(x) \approx 0$, the computation $g(x) = \frac{f(x+f(x))-f(x)}{f(x)}$ will have a large relative error because the numerator is the difference of two small numbers very close in modulus. Because of this, it would probably be best to only use Steffensen's when high precision is not necessary.

3.17.) The derivative of the sinc function is given by

$$f(x) = \frac{x\cos(x) - \sin(x)}{x^2}$$

(a) Show that near $x = 0$, this function can be approximated by $f(x) \approx -x/3$.

Proof. The Taylor expansion of $f(x)$ at $x = 0$ is simply

$$f(x) = -x/3 + x^3/30 + \mathcal{O}(x^5)$$

Thus, for sufficiently small x, $f(x) \approx -x/3$. □

(b) Find all the roots of f in the interval $[-10, 10]$ for $tol = 10^{-8}$.

Solution. My code produced the following results

```
Root 1 found at -7.7252518369
Root 2 found at -4.4934094579
Root 3 found at 0.0000000003
Root 4 found at 4.4934094579
Root 5 found at 7.7252518369
```

The aforementioned program can be found in the appendix.

9.7.) Use Newton's method to solve a discretized version of the differential equation

$$y'' = -(y')^2 - y + \ln x,\ 1 \le x \le 2,\ y(1) = 0,\ y(2) = \ln 2$$

The discretization on a uniform mesh, with the notation of Example 9.3, can be

$$\frac{y_{i+1} - 2y_i + y_{i-1}}{h^2} + \left(\frac{y_{i+1} - y_{i-1}}{2h}\right)^2 + y_i = \ln(ih), \quad i = 1, 2, \ldots, n$$

The actual solution of this problem is $y(x) = \ln x$. Compare your numerical results to the solution $y(x)$ for $n = 8, 16, 32$, and 64. Make observations regarding the convergence behavior of Newton's method in terms of the iterations and the mesh size, as well as the solution error.

Solution. The results from my code in the appendix are

```
For n=8
At step 1, the norm of F(x_k) is 1.409541e+02
At step 2, the norm of F(x_k) is 1.364191e+01
At step 3, the norm of F(x_k) is 1.320465e+00
At step 4, the norm of F(x_k) is 2.784597e-02
At step 5, the norm of F(x_k) is 9.131461e-04
At step 6, the norm of F(x_k) is 4.543229e-05
At step 7, the norm of F(x_k) is 2.297099e-06
At step 8, the norm of F(x_k) is 1.162189e-07
At step 9, the norm of F(x_k) is 5.880229e-09
At step 10, the norm of F(x_k) is 2.975187e-10
But the error between our computed solution and ln(x) is 1.510161e-04

For n=16
At step 1, the norm of F(x_k) is 7.180007e+02
At step 2, the norm of F(x_k) is 5.352052e+01
At step 3, the norm of F(x_k) is 7.773236e+00
At step 4, the norm of F(x_k) is 4.445745e-01
At step 5, the norm of F(x_k) is 9.258122e-03
At step 6, the norm of F(x_k) is 4.578369e-04
At step 7, the norm of F(x_k) is 2.376486e-05
At step 8, the norm of F(x_k) is 1.229115e-06
At step 9, the norm of F(x_k) is 6.350217e-08
At step 10, the norm of F(x_k) is 3.280002e-09
At step 11, the norm of F(x_k) is 1.694077e-10
But the error between our computed solution and ln(x) is 5.812408e-05

For n=32
At step 1, the norm of F(x_k) is 3.693786e+03
At step 2, the norm of F(x_k) is 3.665842e+02
At step 3, the norm of F(x_k) is 3.462514e+01
At step 4, the norm of F(x_k) is 2.005433e+00
At step 5, the norm of F(x_k) is 1.837410e-02
At step 6, the norm of F(x_k) is 5.725169e-04
At step 7, the norm of F(x_k) is 2.819104e-05
At step 8, the norm of F(x_k) is 1.437552e-06
At step 9, the norm of F(x_k) is 7.381994e-08
At step 10, the norm of F(x_k) is 3.796790e-09
At step 11, the norm of F(x_k) is 1.953549e-10
But the error between our computed solution and ln(x) is 2.148431e-05
```

```
For n=64
At step 1, the norm of F(x_k) is 2.457873e+04
At step 2, the norm of F(x_k) is 2.132237e+03
At step 3, the norm of F(x_k) is 2.161847e+02
At step 4, the norm of F(x_k) is 2.186607e+01
At step 5, the norm of F(x_k) is 4.860178e-01
At step 6, the norm of F(x_k) is 3.685293e-03
At step 7, the norm of F(x_k) is 1.730403e-04
At step 8, the norm of F(x_k) is 8.724365e-06
At step 9, the norm of F(x_k) is 4.447616e-07
At step 10, the norm of F(x_k) is 2.273275e-08
At step 11, the norm of F(x_k) is 1.162650e-09
At step 12, the norm of F(x_k) is 5.949407e-11
But the error between our computed solution and ln(x) is 7.771106e-06
```

The convergence with respect to iteration count appear to be either linear or quadratic with a large constant factor. As for mesh size, the convergence is most definitely linear. It appears to improve roughly by a factor of 2 at each doubling of the mesh size.

2 Xue Problems

2.) For a nonlinear system $F(x) = 0 (x \in \mathbb{R}^n)$, let x^* be a solution, assume that $J_f(x_k)$ is nonsingular, and that all Hessian $H_t = [\frac{\partial f_t}{\partial x_i \partial x_j}]$, $(1 \leq l \leq n)$ are finite near x^*. Show that Newton's method

$$x^{(k+1)} = x^{(k)} - J_f^{-1}(x^{(k)})F(x^{(k)}$$

converges quadratically if $x^{(k)}$ is sufficiently close to x^*.

Proof. We will prove this in the same fashion as the proof of Newton's one dimensional case (indeed, it is just a generalization). Begin by considering the update of Newton's methods under the assumptions above.

$$x^{(k+1)} = x^{(k)} - [J_F(x^{(k)})]^{-1}F(x^{(k)})$$

Because $F(x^*) = 0$, this is equivalent to

$$x^{(k+1)} = x^{(k)} - [J_F(x^{(k)})]^{-1}[F(x^{(k)}) - F(x^*)]$$

Now, upon Taylor expansion of $F(x^*)$, we see

$$\begin{aligned} x^{(k+1)} &= x^{(k)} - [J_F(x^{(k)})]^{-1}[F(x^{(k)}) - (F(x^{(k)}) + J_F(x^{(k)}(x^* - x^{(k)}) + \mathcal{O}\|x^* - x^{(k)}\|^2)] \\ &= x^* + [J_F(x^{(k)})]^{-1}\mathcal{O}\|x^{(k)} - x^*\|^2 \end{aligned}$$

after some cancellation. This implies that

$$x^{(k+1)} - x^* = [J_F(x^{(k)}]^{-1}\mathcal{O}\|x^{(k)} - x^*\|^2$$

That is, for sufficiently close $x^{(k)}$ and finite Hessian, the convergence of Newton's method is quadratic. $\square$

3.) Solve the nonlinear system of equations by Newtons method

$$\begin{aligned} y^2\cos(x) + x\ln z &= pi - 4 \\ \cot(x/4) + yz^2 &= 2e^2 + 1 \\ z\sin(x) + y\ln z &= 2 \end{aligned}$$

Let $x^{(0)} = [3, 3, 3]^T$ and show the error $e_k = \|x^{(k)} - x^*\|$ where $x^*[\pi, 2, e]^T$

Solution. My program requires the Jacobian to be known before-hand. Thus, we pre-compute it to be

$J_F(x, y, z) =$

$$\begin{bmatrix} -y^2\sin(x) + \ln(z) & 2y\cos(x) & \frac{x}{z} \\ -\csc^2(\frac{x}{4}) & z^2 & 2yz \\ z & \ln z & \sin(x) + \frac{y}{z} \end{bmatrix}$$

The output I received was

```
x_0 = [3.000000e+00, 3.000000e+00, 3.000000e+00]  e_0 = 1.048529e+00
x_1 = [3.167523e+00, 2.151783e+00, 2.746043e+00]  e_1 = 1.564647e-01
x_2 = [3.142876e+00, 2.004980e+00, 2.717253e+00]  e_2 = 5.244954e-03
x_3 = [3.141593e+00, 2.000004e+00, 2.718277e+00]  e_3 = 6.615003e-06
```

The convergence should be quadratic. Indeed, the exponents of the error roughly double every iteration.

3 Appendix

3.1 Script files

3.1.1 Chapter 3 Question 1

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Fall 2018 Math 8600 w/ Xue
%   Homework 2
%
% Question
%   3.1
%
% Function Dependencies
%   bisection.m
%
% Notes
%   None
%
% Author
%   Trevor Squires
%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear
clc
close all;

f = @(x) sqrt(x) - 1.1;
tol = 1e-8;
a = 0;
b = 2;

iterates = bisection(f,a,b,tol);
n = length(iterates);
solu = iterates(n);
```

3.1.2 Chapter 3 Question 5

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Fall 2018 Math 8600 w/ Xue
%   Homework 2
%
% Question
%   3.5
%
% Function Dependencies
%   fixedPoint.m
%
% Notes
%   Uses the fixedPoint function to do Newton's Method.
%
% Author
%   Trevor Squires
%%%%%%%%%%%%%%%%%%%%%%%%%%%

a = [0,2,10];
n = length(a);
solu = zeros(1,n);
epsilon = rand()+1;

for i = 1:length(a)
    f = @(x) x.^3-a(i);
    g = @(x) x - (x.^3-a(i))/(3*x.^2);
    iterates = fixedPoint(f,g,epsilon+a(i)/2,1e-8);
    fprintf('\n\nFor a = %d\n',a(i))
    for j=1:length(iterates)
        fprintf('x_%d = %e\tf(x_%d) = %e\n',j-1,iterates(j),j-1,f(iterates(j)))
    end
    solu(i) = iterates(end);
end
```

3.1.3 Chapter 3 Question 17

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Fall 2018 Math 8600 w/ Xue
%   Homework 2
%
% Question
%   3.17b
%
% Function Dependencies
%   rootFinder.m
%
% Notes
%   Uses secant to find roots after bracketing by probes
%
% Author
%   Trevor Squires
%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear
clc
close all;

f = @(x) (x.*cos(x)-sin(x))./x.^2;
tol = 1e-8;
a = -10;
b= 10;
pts = 20;

roots = rootFinder(f,a,b,pts,tol);

for i = 1:length(roots)
    fprintf('Root %d found at %0.10f\n',i,roots(i))
end
```

3.1.4 Chapter 9 Question 7

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Fall 2018 Math 8600 w/ Xue
%   Homework 2
%
% Question
%   9.7
%
% Function Dependencies
%   None
%
% Notes
%   Uses precomputed Jacobian to do Newton's method.
%
% Author
%   Trevor Squires
%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear
clc
close all;

%% Initialize necessary variables
tol = 10e-10;
n = [8,16,32,64];

for j = 1:length(n)
    h = 1/(n(j)+1);
    t = 1:h:2;
    solu = log(t(2:n(j)+1));
    y0 = rand(1,n(j));
    error = norm(y0,2);
    iterate = y0;
    i = 1;
    fprintf('\nFor n=%d\n',n(j))
    while error > tol
        y = iterate(i,:);
        [jfxk, fxk] = functionEval(y,h);
        update = (jfxk\fxk'); %solve y = inv(jfxk)*fxk
        iterate(i+1,:) = iterate(i,:)-update'; %update xk+1 = xk - update
        i = i+1;
        error = norm(fxk); %refresh error
        fprintf('At step %d, the norm of F(x_k) is %e\n',i-1,error)
    end
    fprintf('But the error between our computed solution and ln(x) is %e\n',norm(iterate(i,:)-
end

%% Evaluation function
function [myJac,fy] = functionEval(y,h)

n = length(y);
y = [0 y log(2)];

%% Evaluate our F at x
fy = zeros(1,n);
for i = 2:n+1
    fy(i-1) = (y(i+1) - 2*y(i) + y(i-1))/h^2 + ((y(i+1) - y(i-1))/(2*h))^2 + y(i) - log(1+(i-1
end

%% Evaluate Jf at x
myJac = zeros(n);
diags = zeros(1,n-1);

for i = 1:n-1
    diags(i) = (y(i+2)-y(i))/2/h^2;
end

myJac = myJac + diag(ones(1,n)*(1-2/(h^2)));
myJac = myJac + diag((1/h^2)+diags,1) + diag(diags*(-1)+(1/h^2),-1);

end
```

3.1.5 Xue Question 3

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Fall 2018 Math 8600 w/ Xue
%   Homework 2
%
% Question
%   Problem 3
%
% Function Dependencies
%   None
%
% Notes
%   Uses precomputed Jacobian to do Newton's method.
%
% Author
%   Trevor Squires
%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear
clc
close all;

%% Initialize necessary variables
tol = 10e-8;
xstar = [pi 2 exp(1)];
x0 = [3 3 3];

%pre-compute jacobian using f
f = @(x,y,z) [cos(x)*y^2 + x*log(z) - pi + 4; cot(x/4) + y*z^2 - 1-2*exp(2); z*sin(x) + y*log(
myJacobian = @(x,y,z) [-sin(x)*y^2 + log(z), 2*y*cos(x), x/z;(-1/4)*(csc(x/4))^2, z^2, 2*y*z; 

%set error to start loop
error = norm(x0-xstar);
iterate = x0;
i = 1;

%% Newton's Update
while error > tol
    x = iterate(i,:);
    fprintf('x_%d = [%e, %e, %e]\t e_%d = %e\n',i-1,x(1),x(2),x(3),i-1,error)

    fxk = f(x(1),x(2),x(3)); %compute fxk(x,y,z)
    jfxk = myJacobian(x(1),x(2),x(3)); % and jfxk(x,y,z)
    update = (jfxk\fxk)'; %solve y = inv(jfxk)*fxk
    iterate(i+1,:) = iterate(i,:)-update; %update xk+1 = xk - update
    i = i+1;
    error = norm(xstar-iterate(i,:)); %refresh error

end
```

3.2 Accompanying Functions

3.2.1 Fixed Point Iteration

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%
% FIXEDPOINT.m
%
% DESCRIPTION
%   Finds fixed point of a given function using the fixed point method
%
% AUTHOR
%   Trevor Squires
%
% ARGUMENTS
%   f - function handle
%   x0 - initial starting point
%   tol - tolerance for quiting
%
% OUTPUT
%   history - vector of previous iterates
%
% NOTES
%   The function f must satisfy the fixed point theorem requirements
%   Can choose to change stopping criterion by choosing how to compute the
%   error
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%


function [history] = fixedPoint(f,g,x0,tol)
error = tol+1;
history = x0;
count = 1;

while error>tol
    history(count+1) = g(history(count)); %add to history
    %error = abs(history(count+1)-history(count));
    error = abs(f(history(count+1)));  %update error
    count = count + 1;
end
```

3.2.2 General Root Finder

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%
% ROOTFINDER.m
%
% DESCRIPTION
%   Probes a given function for possible roots between specified bounds.
%   Proceeds to find all roots in the bounds.
%
% AUTHOR
%   Trevor Squires
%
% ARGUMENTS
%   f - function handle
%   a - lower bound for searching
%   b - upper bound for searching
%   pts - the number of points to initially probe by
%   tol - tolerance for quiting
%
% OUTPUT
%   roots - the roots of f between a and b
%
% NOTES
%   The function f must satisfy the fixed point theorem requirements
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%


function [roots] = rootFinder(f,a,b,pts,tol)

t = a:(b-a)/(pts-1):b;

currentSign = sign((f(t(1))));
bounds = 1;

for i = 2:pts
    if currentSign*sign(f(t(i))) < 0
        currentSign = currentSign*-1;
        bounds = [bounds i];
    end
end

n = length(bounds)-1;
roots = zeros(1,n);

for i = 1:n
```

```
    %iterates = bisection(f,t(bounds(i)),t(bounds(i+1)),tol);
    %or
    iterates = secant(f,t(bounds(i)),t(bounds(i+1)),tol);
    roots(i) = iterates(end);
end
```

3.2.3 Bisection Method

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%
% BISECTION.m
%
% DESCRIPTION
%   Finds a root of a given function using the bisection method
%
% AUTHOR
%   Trevor Squires
%
% ARGUMENTS
%   f - function handle
%   a - left end point
%   b - right endpoint
%   tol - absolute tolerance for quiting
%
% OUTPUT
%   history - vector of previous iterates
%
% NOTES
%   Requires [a,b] to bracket the root.
%   Precomputes number of iterations to better store iterates.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%


function [history] = bisection(f,a,b,tol)

fa = f(a);
fb = f(b);

if fa*fb > 0
    error('Please bracket the root appropriately');
end

c = (a+b)/2;
fc = f(c);

n = ceil(log2((b-a)/(2*tol)))-1; %precompute number of iterations
history = zeros(1,n);


for i = 1:n
    if fc ==0
        break;
    elseif fc*fa > 0
        a = c;
        fa = fc;
    else
        b = c;
    end
    c = (a+b)/2;
    fc = f(c);
    history(i) = c;

end
```

3.2.4 Secant Method

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%
% SECANT.m
%
% DESCRIPTION
%   Finds root of a given function using the secant method
%
% AUTHOR
%   Trevor Squires
%
% ARGUMENTS
%   f - function handle
%   x0 - initial starting point
%   x1 - second starting point
%   tol - tolerance for quiting
%
% OUTPUT
%   history - vector of previous iterates
%
% NOTES
%   Uses change in iterates as stopping criterion
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [history] = secant(f,x0,x1,tol)

error = tol+1;
count = 2;
history = [x0 x1];

while error > tol
    history(count+1) = history(count) - f(history(count))*(history(count)- history(count-1))/(
    count = count +1;
    error = abs(history(count)-history(count-1));
end
```