Lendel Deguia

3.2.1 $f(x) = x^2 - R \Rightarrow f'(x) = 2x$

Newton's Method: $x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \Rightarrow x_{n+1} = x_n - \frac{(x_n^2 - R)}{2x_n}$

$\Rightarrow x_{n+1} = x_n - \frac{x_n}{2} + \frac{R}{2x_n} = \frac{1}{2}x_n + \frac{1}{2}\frac{R}{x_n}$

$$\Rightarrow \quad x_{n+1} = \frac{1}{2}\left(x_n + \frac{R}{x_n}\right)$$

3.2.6 $f(x) = (x-1)^m, \ m \in \{8, 12\}$

$\Rightarrow f'(x) = m(x-1)^{m-1} \ ; \ x_0 = 1.1$

using a calculator for Newton's method yields up to $n=4$:

m=8

$n=0, \ x_n = 1.1, \ f(x_n) = 1.0000 \times 10^{-8}$
$n=1, \ x_n = 1.0875, \ f(x_n) = 3.4361 \times 10^{-9}$
$n=2, \ x_n = 1.0766, \ f(x_n) = 1.1807 \times 10^{-9}$
$n=3, \ x_n = 1.0670, \ f(x_n) = 4.0569 \times 10^{-9}$
$n=4, \ x_n = 1.0586, \ f(x_n) = 1.3940 \times 10^{-10}$

m=12

$n=0, \ x_n = 1.1, \ f(x_n) = 1.0000 \times 10^{-12}$
$n=1, \ x_n = 1.0917, \ f(x_n) = 3.5200 \times 10^{-13}$
$n=2, \ x_n = 1.0840, \ f(x_n) = 1.2390 \times 10^{-13}$
$n=3, \ x_n = 1.0770, \ f(x_n) = 4.3612 \times 10^{-14}$
$n=4, \ x_n = 1.0706, \ f(x_n) = 1.5351 \times 10^{-14}$

3.2.6 continued

The reason convergence is painfully slow is because $f(x)$ has a root at $x=1$ of multiplicity 8 for $m=8$ and multiplicity 12 for $m=12$ — that is, $f(x)$ ~~has~~ shares a root with all derivatives until the 7^{th} and 11^{th} derivatives for $m=8$ and $m=12$ respectively.

~~We can use instead, the modified Newton's method~~

The multiplicity of the roots causes the convergence of Newton's method to become linear instead of quadratic. However, we can, instead, use the modified Newton's method $\left(x_{n+1} = x_n - m\frac{f(x_n)}{f'(x_n)}\right)$ to recover the quadratic convergence. Indeed, using a calculator:

$m=8$ $\quad n=0,\ x_n=1.1,\ f(x_n)=1.000\times10^{-8}$

$\quad n=1,\ x_n=1.0,\ f(x_n)=0$

$m=12$ $\quad n=0,\ x_n=1.1,\ f(x_n)=1.000\times10^{-12}$

$\quad n=1,\ x_n=1.0,\ f(x_n)=0$

3.2.12 a) $x_{n+1} = \frac{1}{3}\left(2x_n - \frac{r}{x_n^2}\right) = \frac{1}{3}x_n - \frac{1}{3}x_n + \frac{2}{3}x_n - \frac{1}{3}\frac{r}{x_n^2}$

$$\Rightarrow x_n - \left(\frac{x_n^3 + r}{3x_n^2}\right) = x_n - \frac{f(x_n)}{f'(x_n)}$$

$$\Rightarrow f(x) = x^3 + r, \quad f'(x) = 3x^2$$

$\therefore$ For an appropriate choice of a nonzero initial point, the sequence $\boxed{\text{will converge}}$ to $\boxed{x = -(r)^{1/3}}$

---

b) $x_{n+1} = \frac{1}{2}x_n + \frac{1}{x_n} = x_n - \frac{1}{2}x_n + \frac{1}{x_n} = x_n - \frac{x_n^2}{2x_n} + \frac{2}{2x_n}$

$$\Rightarrow x_n - \left(\frac{x_n^2 - 2}{2x_n}\right) = x_n - \frac{f(x_n)}{f'(x_n)}$$

$$\Rightarrow f(x) = x^2 - 2, \quad f'(x) = 2x$$

$\therefore$ For an appropriate choice of a nonzero initial point, the sequence $\boxed{\text{will converge}}$ to either $\boxed{x = \sqrt{2}}$ or $\boxed{x = -\sqrt{2}}$

$\boxed{3.3.2}$

$$x_{n+1} = x_n - f(x_n)\left[\frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})}\right]$$

$\mathbb{N}_0 = \mathbb{N} \cup \{0\}$

$f(x) = x^3 - 2x + 2$; $x_0 = 0$, $x_1 = 1$

$\Rightarrow f(x_0) = 2 \quad f(x_1) = 1 - 2 + 2 = 1$

$\Rightarrow x_2 = 1 - 1 \cdot \left[\frac{1-0}{1-2}\right] = 1 + 1 \Rightarrow \boxed{x_2 = 2}$

$\boxed{3.3.11}$ (i) $\lim_{n\to\infty} x_n = r \ni \left[\left(x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}\right) \wedge \left(f'(r) \neq 0\right)\right], n \in \mathbb{N}_0$

$\Rightarrow f(r) = 0$ where $f(x) \in C^1$. (Newton method proof)

proof| Suppose $\lim_{n\to\infty} x_n = r$; then $\lim_{n\to\infty} x_{n+1} = r$, so :

$\lim_{n\to\infty} x_n = \lim_{n\to\infty} x_{n+1} \Rightarrow \lim_{n\to\infty} x_n = \lim_{n\to\infty}\left[x_n - \frac{f(x_n)}{f'(x_n)}\right]$

$\Rightarrow r = r - \frac{f(r)}{f'(r)} \Rightarrow -\frac{f(r)}{f'(r)} = 0 \Rightarrow f(r) = 0$ $\boxed{\text{QED}}$

(ii) $\lim_{n\to\infty} x_n = r \ni \left[\left(x_{n+1} = x_n - f(x_n)\left[\frac{f(x_n) - f(x_{n-1})}{x_n - x_{n-1}}\right]^{-1}\right) \wedge \left(f'(r) \neq 0\right)\right], n \in \mathbb{N}$

$\Rightarrow f(r) = 0$ where $f(x) \in C^1$; suppose WLOG, $x_{n-1} \leq x_n$. (secant method proof)

proof| Suppose $\lim_{n\to\infty} x_n = r$; then $\lim_{n\to\infty} x_{n+1} = r$. By the Mean Value Theorem Since $f(x) \in C^1$, $\exists c_n \in (x_{n-1}, x_n)$: $f'(c_n) = \frac{f(x_n) - f(x_{n-1})}{x_n - x_{n-1}}$.

Thus, the secant method sequence becomes $x_{n+1} = x_n - \frac{f(x_n)}{f'(c_n)}$ and since $\forall n \in \mathbb{N}$, $x_{n-1} \leq c_n \leq x_n$, $\lim_{n\to\infty} c_n = r$. Therefore:

$r = \lim_{n\to\infty}\left(x_n - \frac{f(x_n)}{f'(c_n)}\right) = r - \frac{f(r)}{f'(r)} \Rightarrow -\frac{f(r)}{f'(r)} = 0 \Rightarrow f(r) = 0$

$\boxed{\text{QED}}$

3.3.13 b) For $x_n = \frac{1}{2^n}$, $\lim_{n\to\infty} x_n = 0$

Define error as: $|\varepsilon| = |x_n - a|$ where $\lim_{n\to\infty} x_n = a$.

Note that $\forall n \in \mathbb{N}$, $\frac{x_{n+1}}{x_n} = \frac{2^n}{2^{n+1}} = \frac{1}{2}$

so $|x_{n+1} - 0| = \frac{1}{2}|x_n - 0| < \frac{3}{4}|x_n - 0|$

$\therefore \exists C \in [0,1) : |x_{n+1} - 0| \leq C|x_n - 0|$

so $\{x_n\}$ is $\boxed{\text{linearly convergent}}$

---

d) $a_0 = a_1 = 1$; $a_2 = 2, a_3 = 3, a_4 = 5, a_5 = 8, \ldots$

$\Rightarrow a_n \in \{1, 1, 2, 3, 5, 8, \ldots\} \Rightarrow$ a_n is a Fibonacci sequence

$a_{n+1} = a_n + a_{n-1} \Rightarrow \forall n \in \mathbb{N},\ a_{n+1} > a_n$

$x_n = 2^{-a_n} \Rightarrow \lim_{n\to\infty} x_n = 0$ since a_n is an increasing sequence.

$$\lim_{n\to\infty} \frac{|x_{n+1}|}{|x_n|} = \lim_{n\to\infty} \frac{2^{-a_n - a_{n-1}}}{2^{-a_n}} = \lim_{n\to\infty} 2^{-a_{n-1}} = 0$$

also note that $\lim_{n\to\infty} \frac{|x_{n+1}|}{|x_n|^\alpha} = \lim_{n\to\infty} 2^{(\alpha-1)a_n - a_{n-1}}$

so if $\alpha \leq 1$, $\lim_{n\to\infty} \frac{|x_{n+1}|}{|x_n|^\alpha} = 0$ and if $\alpha \geq 2$, the limit diverges. There must be some α such that $\frac{|x_{n+1}|}{|x_n|^\alpha}$ converges to a nonzero value, so we can conclude that $1 < \alpha < 2 \Rightarrow$ $\boxed{\{x_n\} \text{ is superlinearly convergent}}$

Computer Exercise 3.2.2

This program will solve the equation $x^3 + 2x^2 + 10x = 20$ by finding the root of $f(x) = x^3 + 2x^2 + 10x - 20$ closest to an initial point $x_0 = 2$. Moreover, $f(x)$ will be casted via nested multiplication as $f(x) = x(10 + x(2 + x(1))) - 20$ for efficient computation of the polynomial.

The program will terminate when either the number of iterates exceeds 10 or when the following error condition is met: $|x_n - x_{n-1}| < \frac{1}{2} \times 10^{-5}$

Iterate number, corresponding x value, and the the function at x will be displayed.

```
syms x; %program is modified to only take a function
        %as input by having the derivative evaluated
        %within the Newton algorithm; to accomplish this
        %symbolic functions are utilized

%%inputs for Newton's method

f = x*(10 + x*(2 + x*(1))) - 20; %symbolic function

x0=2; %initial point

N=10; %max number of iterates

err = 0.5 * 10^(-5); %error tolerance for successive points

m=1; %if root multiplicity is known ahead of time,
     %this value can be changed to account for
     %root multiplicity and use the modified
     %Newton's method

root1 = newton(f, x0, N, err, m);
```

```
n = 0, xn =   2.000000000000, f(xn) = 1.600000000000e+01, error = 1.000000000000e+00
n = 1, xn =   1.466666666667, f(xn) = 2.123851851852e+00, error = 5.333333333333e-01
n = 2, xn =   1.371512013806, f(xn) = 5.708664190432e-02, error = 9.515465286075e-02
n = 3, xn =   1.368810222634, f(xn) = 4.461440696415e-05, error = 2.701791172025e-03
n = 4, xn =   1.368808107823, f(xn) = 2.731055306559e-11, error = 2.114811228029e-06
```

The program terminates after four iterates have been evaluated. The error has also been displayed; the quadratic convergence of Newton's method becomes evident from the third to fourth iterations. We can see that the root x_r converging to four decimal places so we can conclude that $x_r \approx 1.3688$.

```
function root = newton(f, x0, N, err, m)
    n=0; %initialize iterate
    syms x; %symbols needs to be redeclared otherwise
            %algorithm yields an error

    fd = diff(f); %symbolically evaluate derivative of
                  %input function
```

```
    y = subs(f, x, x0); %evaluate input function at x0

    dy = subs(fd, x, x0); %evaluate derivative at x0

    xn = x0; %initialize iterate x value for loop

    error = 1; %initialize error to any value such that error < err

    fprintf(['n = %d, xn = %16.12f, f(xn) = %16.12e, ' ...
        'error = %16.12e \n'], n, xn, y, error)
    %display the zeroth iterate
    while (error > err) && (n < N)
        root = xn - ((m*y)/dy); %evaluate subsequent point
                                %using Newton's method formula
        y = subs(f, x, root);
        dy = subs(fd, x, root);
        error = abs(xn -root); %error value between successive points
        xn = root;
        n = n + 1; %increment iteration number
        fprintf(['n = %d, xn = %16.12f, f(xn) = %16.12e, ' ...
            'error = %16.12e \n'], n, xn, y, error)
        %display nth iterate
    end
end
```

Computer Exercise 3.2.15

This program will solve the equation $\frac{1}{2}x^2 + x + 1 = e^x$ by finding the root of $f(x) = \frac{1}{2}x^2 + x + 1 - e^x$ closest to an initial point $x_0 = 1$.

The program will terminate when either the number of iterates exceeds a set amount or when the following error condition is met: $|x_n - x_{n-1}| < \frac{1}{2} \times 10^{-5}$

Iterate number, corresponding x value, and the the function at x will be displayed.

```
syms x; %program is modified to only take a function
        %as input by having the derivative evaluated
        %within the Newton algorithm; to accomplish this
        %symbolic functions are utilized

%%inputs for Newton's method

f = (1/2)*(x^2) + x + 1 - exp(x); %symbolic function

x0=1; %initial point

N=6; %max number of iterates

err = 0.5 * 10^(-5); %error tolerance for successive points

m=1; %if root multiplicity is known ahead of time,
     %this value can be changed to account for
     %root multiplicity and use the modified
     %Newton's method

root1 = newton(f, x0, N, err, m);
```

```
n = 0, xn =   1.000000000000, f(xn) = -2.182818284590e-01, error = 1.000000000000e+00
n = 1, xn =   0.696105595589, f(xn) = -6.753849522062e-02, error = 3.038944044113e-01
n = 2, xn =   0.478112902284, f(xn) = -2.061871165303e-02, error = 2.179926933045e-01
n = 3, xn =   0.325285123719, f(xn) = -6.234992633393e-03, error = 1.528277785653e-01
n = 4, xn =   0.219857803679, f(xn) = -1.873025059444e-03, error = 1.054273200397e-01
n = 5, xn =   0.147933877018, f(xn) = -5.601354445951e-04, error = 7.192392666102e-02
n = 6, xn =   0.099236404329, f(xn) = -1.670001644371e-04, error = 4.869747268924e-02
```

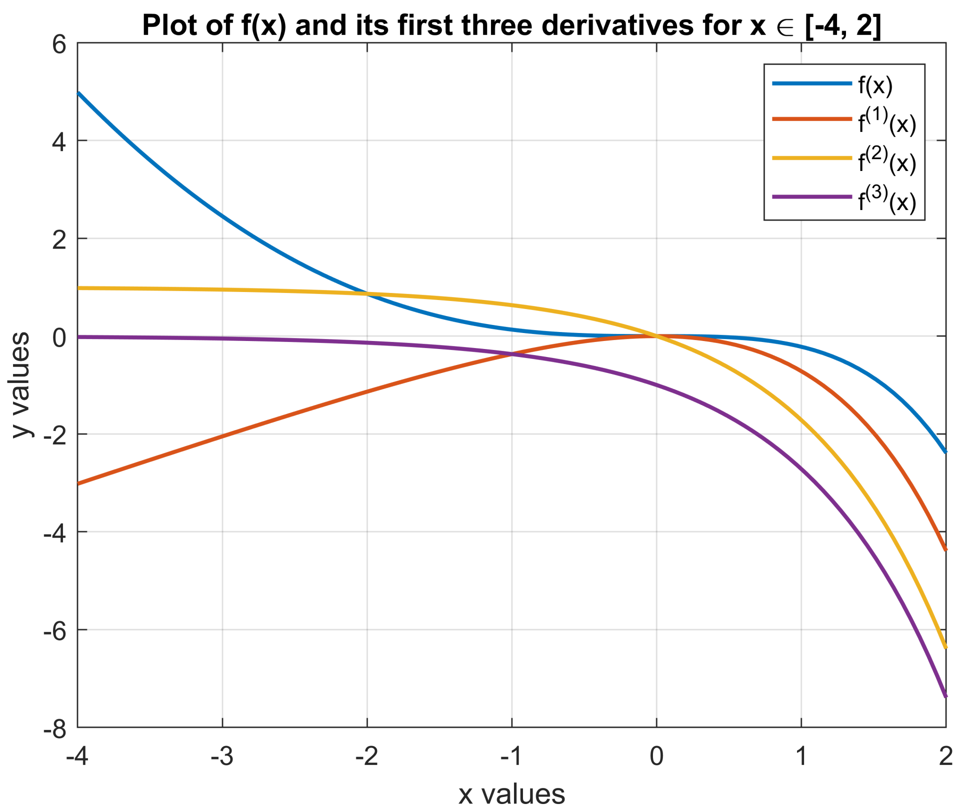
After six iterates, there is no sign of the beloved quadratic convergence of Newton's method; the program is painfully slow past six iterates so I stopped it there. There is a reason for this slow convergence which we can investigate with the following plot:

```
a=4;
fd = diff(f);
fd2 = diff(fd);
fd3 = diff(fd2);

clf
xvals = linspace(-a,a-2);
```

```
yvals1 = vpa(subs(f, x, xvals), 16);
yvals2 = vpa(subs(fd, x, xvals), 16);
yvals3 = vpa(subs(fd2, x, xvals), 16);
yvals4 = vpa(subs(fd3, x, xvals), 16);

h = plot(xvals, yvals1, xvals, yvals2, xvals, yvals3, xvals, yvals4);
thickness = 1.5;
set(h(1),'linewidth', thickness);
set(h(2),'linewidth', thickness);
set(h(3),'linewidth', thickness);
set(h(4),'linewidth', thickness);
legend('f(x)', 'f^{(1)}(x)', 'f^{(2)}(x)', 'f^{(3)}(x)')
title('Plot of f(x) and its first three derivatives for x \in [-4, 2]')
grid on
xlabel('x values')
ylabel('y values')
```



It appears that there is a root at (x,y) = (0,0), but $f(x)$ shares this exact same root with both $f'(x)$ and $f''(x)$! Therefore, the root has multiplicity of three! We can recover the quadratic convergence by using $m = 3$ as an input in the Newton algorithm:

```
root_mod = newton(f, x0, 10, err, 3); %repeat the algorithm with N=10 and m=3
```

```
n = 0, xn =   1.000000000000, f(xn) = -2.182818284590e-01, error = 1.000000000000e+00
n = 1, xn =   0.088316786766, f(xn) = -1.173900339448e-04, error = 9.116832132340e-01
n = 2, xn =   0.000653786133, f(xn) = -4.658293544846e-11, error = 8.766300063257e-02
```

```
n = 3, xn =   0.000000035621, f(xn) = -7.533139829068e-24, error = 6.537505121852e-04
n = 4, xn =   0.000000000000, f(xn) = -1.970423440204e-49, error = 3.562124467583e-08
```

Amazing! We have a quadratic convergence again after only four iterates with an insane amount of accuracy for the function near the root (on the order of 10^{-49}).

```
function root = newton(f, x0, N, err, m)
    n=0; %initialize iterate
    syms x; %symbols needs to be redeclared otherwise
            %algorithm yields an error

    fd = diff(f); %symbolically evaluate derivative of
                  %input function

    y = subs(f, x, x0); %evaluate input function at x0

    dy = subs(fd, x, x0); %evaluate derivative at x0

    xn = x0; %initialize iterate x value for loop

    error = 1; %initialize error to any value such that error < err

    fprintf(['n = %d, xn = %16.12f, f(xn) = %16.12e, ' ...
        'error = %16.12e \n'], n, xn, y, error)
    %display the zeroth iterate
    while (error > err) && (n < N)
        root = xn - ((m*y)/dy); %evaluate subsequent point
                                %using Newton's method formula
        y = subs(f, x, root);
        dy = subs(fd, x, root);
        error = abs(xn -root); %error value between successive points
        xn = root;
        n = n + 1; %increment iteration number
        fprintf(['n = %d, xn = %16.12f, f(xn) = %16.12e, ' ...
            'error = %16.12e \n'], n, xn, y, error)
        %display nth iterate
    end
end
```

Computer Exercise 3.2.18

This program will solve the equation $2x(1+x^2)^{-1} = arctan(x)$ by approximating the positive root of $f(x) = 2x(1+x^2)^{-1} - arctan(x)$ with the bisection method. This root approximation will then be used as the initial point in using Newton's method to find the root of $f(x) = arctan(x)$.

The program will terminate when either the number of iterates exceeds 50 for the Bisection method and 7 for Newton's method or when the following error condition is met: $|x_n - x_{n-1}| < \frac{1}{2} \times 10^{-5}$

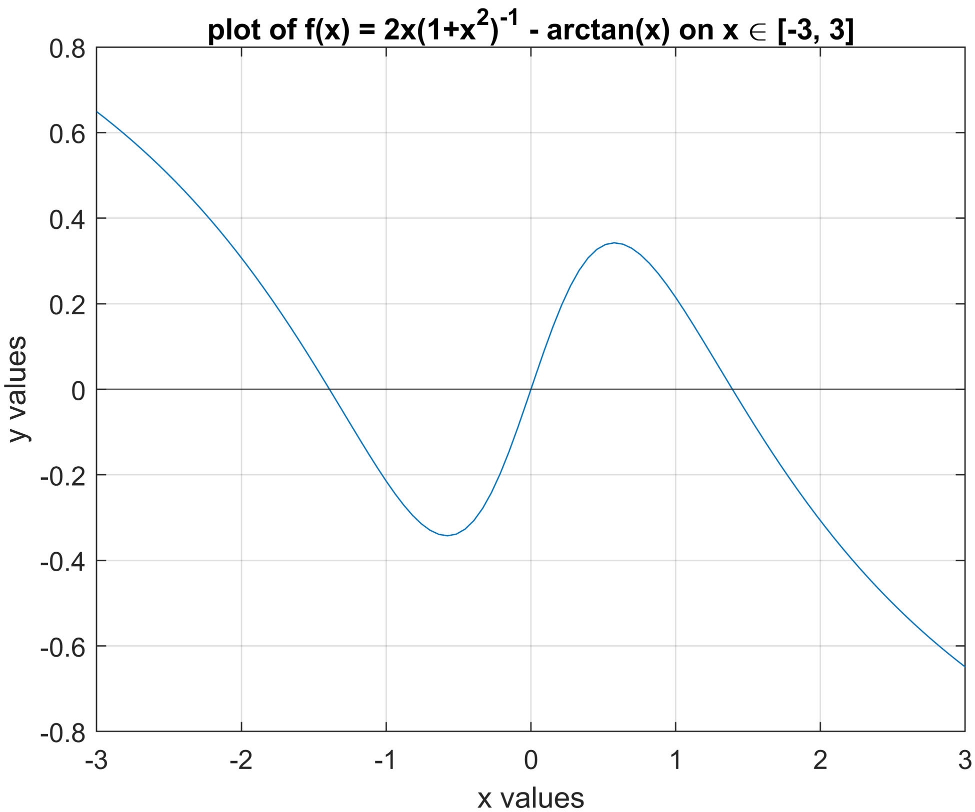
Iterate number, corresponding x value, and the the function at x will be displayed.

```
f = @(x) ((2.*x).*((1 + x.^2).^(-1))) - atan(x);
```

We need a good idea of where to pick our starting points for the bisection method. The function is plotted out to assist in doing so.

```
xvals = linspace(-3,3);
yvals = f(xvals);

plot(xvals, yvals)
yline(0)
title('plot of f(x) = 2x(1+x^2)^{-1} - arctan(x) on x \in [-3, 3]')
xlabel('x values')
ylabel('y values')
grid on
```



Based on the plot, the positive root is somewhere between x=1 and x=2 so those look like good starting points for the bisection method.

```
a=1;
b=2;
n=50;
error = 0.5 * 10^(-5);


root0 = bisect(f, a, b, n, error);

fprintf('root = %8.8f', root0)
```

```
root = 1.39175415
```

This is the value that will act as the initial starting point for Newton's method.

```
syms x; %program is modified to only take a function
        %as input by having the derivative evaluated
        %within the Newton algorithm; to accomplish this
        %symbolic functions are utilized

%%inputs for Newton's method

f = atan(x); %symbolic function
```

```
x0=root0; %initial point set as the positive root from the
          %bisection method

N=7; %max number of iterates

err = 0.5 * 10^(-5); %error tolerance for successive points

m=1; %if root multiplicity is known ahead of time,
     %this value can be changed to account for
     %root multiplicity and use the modified
     %Newton's method

root1 = newton(f, x0, N, err, m);
```

```
n = 0, xn =   1.391754150391, f(xn) = 9.477501809191e-01, error = 1.000000000000e+00
n = 1, xn =  -1.391768811204, f(xn) = -9.477551726839e-01, error = 2.783522961595e+00
n = 2, xn =   1.391807487769, f(xn) = 9.477683410827e-01, error = 2.783576298973e+00
n = 3, xn =  -1.391909523013, f(xn) = -9.478030792036e-01, error = 2.783717010782e+00
n = 4, xn =   1.392178729453, f(xn) = 9.478947150013e-01, error = 2.784088252466e+00
n = 5, xn =  -1.392889136981, f(xn) = -9.481364200390e-01, error = 2.785067866434e+00
n = 6, xn =   1.394764817280, f(xn) = 9.487738096523e-01, error = 2.787653954261e+00
n = 7, xn =  -1.399724050556, f(xn) = -9.504536024798e-01, error = 2.794488867836e+00
```

I actually tried 10 iterates of Newton's method at first (which was really slow) but from what can be seen here, this convergence is quite.. well...garbage. I tried looking at the derivative of $f(x) = arctan(x)$, but there is no multiplicity in the root (in fact, there is no root for the derivative). I just copied and pasted this algorithm for Newton's method from the previous problem and it worked fine there. To make sure that it was not my algorithm that was the issue, I attempted Newton's method (somewhat) manually using a calculator (keying in each iterate) and I still end up getting similar results. The root should end up being zero since $tan(0) = 0$, and after seven iterates of Newton's method, the error has not even manage to dip even below one. It turns out that the only explanation (that I can think of) for this terrible convergence is that we have a bad starting point.

```
function c=bisect(f, a, b, n, error) %Bisection algorithm
c=(a+b)/2;
i=1; %set iteration counter
%error = |f(c) - 0|
    while (abs(f(c))>error) && (i<=n) %exit while loop once error tolerance
       %  or max iterations has been reached
        if f(a)*f(c)<0
            b=c;
        else
            a=c;
        end
        c=(a+b)/2;
        i = i+1; %update iteration counter
    end
    if (abs(f(c))>error) %if this triggers, this means that max iterations
       %  has been reached before error tolerance
        fprintf(['bisection algorithm was unsucessful after %d iterations; ' ...
            'error = %f'], i-1, abs(f(c)))
    end
end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function root = newton(f, x0, N, err, m) %Newton Algorithm
    n=0; %initialize iterate
    syms x; %symbols needs to be redeclared otherwise
            %algorithm yields an error

    fd = diff(f); %symbolically evaluate derivative of
                  %input function

    y = subs(f, x, x0); %evaluate input function at x0

    dy = subs(fd, x, x0); %evaluate derivative at x0

    xn = x0; %initialize iterate x value for loop

    error = 1; %initialize error to any value such that error < err

    fprintf(['n = %d, xn = %16.12f, f(xn) = %16.12e, ' ...
        'error = %16.12e \n'], n, xn, y, error)
    %display the zeroth iterate
    while (error > err) && (n < N)
        root = xn - ((m*y)/dy); %evaluate subsequent point
                                %using Newton's method formula
        y = subs(f, x, root);
        dy = subs(fd, x, root);
        error = abs(xn -root); %error value between successive points
        xn = root;
        n = n + 1; %increment iteration number
        fprintf(['n = %d, xn = %16.12f, f(xn) = %16.12e, ' ...
            'error = %16.12e \n'], n, xn, y, error)
        %display nth iterate
    end
end
```