Computer Exercise 3.3.6

For the following functions, Newton's method and the secant method will be compared. For each function, an initial point x_0 is indicated. Since the secant method needs two initial points, the first iteration point of Newton's method will be used and will be designated as x_1 in the secant method.

a) $f(x) = x^3 - 3x + 1,\ x_0 = 2$

```
syms x; %declare symbol for Newton's method

f = x^3 - 3*x + 1; %symbolic form of function

x0 = 2;
err = 0.5 * 10^(-5);

N = 10; %max number of iterates

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
roota1 = newton(f, x0, N, err, m);
```

```
n = 0, xn =   2.000000000000, f(xn) = 3.000000000000e+00, error = 1.000000000000e+00
n = 1, xn =   1.666666666667, f(xn) = 6.296296296296e-01, error = 3.333333333333e-01
n = 2, xn =   1.548611111111, f(xn) = 6.804021722822e-02, error = 1.180555555556e-01
n = 3, xn =   1.532390161865, f(xn) = 1.218139881036e-03, error = 1.622094924573e-02
n = 4, xn =   1.532088989397, f(xn) = 4.169583671445e-07, error = 3.011724681557e-04
n = 5, xn =   1.532088886238, f(xn) = 4.891270035971e-14, error = 1.031592561078e-07
```

We see from Newton's method that $x_1 = \frac{5}{3}$

```
f = @(x) x^3 - 3*x + 1; %initialize non-symbolic function
                        %for secant method

x1 = 5/3; %second initial point acquired from Newton's method


roota2 = secant(f, x0, x1, N, err);
```

```
n = 1, xn =   1.666666666667, f(xn) = 6.296296296296e-01, error = 1.000000000000e+00
n = 2, xn =   1.578125000000, f(xn) = 1.959114074707e-01, error = 8.854166666667e-02
n = 3, xn =   1.538130548187, f(xn) = 2.458771935461e-02, error = 3.999445181280e-02
n = 4, xn =   1.532390697687, f(xn) = 1.220307098349e-03, error = 5.739850500114e-03
n = 5, xn =   1.532090947752, f(xn) = 8.332430295965e-06, error = 2.997499351634e-04
n = 6, xn =   1.532088886945, f(xn) = 2.858945080675e-09, error = 2.060806638005e-06
```

Here, we see that the Secant method converges slightly slower than Newton's method and takes one more iterate to satisfy the error condition.

b) $f(x) = x^3 - 2sin(x),\ x_0 = \frac{1}{2}$

```
%repeat same procedure as in part a
syms x;

f = x^3 - 2*sin(x);
x0 = 0.5;
err = 0.5 * 10^(-5);
N = 10;
m = 1;
rootb1 = newton(f, x0, N, err, m);
```

```
n = 0, xn =   0.500000000000, f(xn) = -8.338510772084e-01, error = 1.000000000000e+00
n = 1, xn =  -0.329566264767, f(xn) = 6.114698471984e-01, error = 8.295662647666e-01
n = 2, xn =   0.060769223454, f(xn) = -1.212392412858e-01, error = 3.903354882208e-01
n = 3, xn =  -0.000301417853, f(xn) = 6.028356685718e-04, error = 6.107064130676e-02
n = 4, xn =   0.000000000037, f(xn) = -7.302569926860e-11, error = 3.014178890552e-04
n = 5, xn =  -0.000000000000, f(xn) = 1.298093329544e-31, error = 3.651284963430e-11
```

We see that from Newton's method, $x_1 \approx -0.3296$

```
f = @(x) x^3 - 2*sin(x);
x1 = -0.329566264767;

rootb2 = secant(f, x0, x1, N, err);
```

```
n = 1, xn =  -0.329566264767, f(xn) = 6.114698471991e-01, error = 1.000000000000e+00
n = 2, xn =   0.021397143146, f(xn) = -4.278122447482e-02, error = 3.509634079134e-01
n = 3, xn =  -0.001552218324, f(xn) = 3.104431661941e-03, error = 2.294936147063e-02
n = 4, xn =   0.000000439529, f(xn) = -8.790588492353e-07, error = 1.552657853653e-03
n = 5, xn =  -0.000000000001, f(xn) = 1.411594619706e-12, error = 4.395301304150e-07
```

The secant method and Newton's method end up taking the same number of iterates to satisfy the error condition, but Newton's method tends to yield lower errors for the same number of iterates.

```
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
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function root = secant(f, x0, x1, N, err)
    xn = x1;
    xnm1 = x0; %initialize upper and lower values of root approximation
    error = 1; %initialize error to any value such that error < err
    n = 0; %initialize iterate
    while (error > err) && (n < N)
        yn = f(xn);
        ynm1 = f(xnm1);
        xnp1 = xn - yn*((xn-xnm1)/(yn-ynm1));%evaluate subsequent point
                                %using the secant method formula
        fprintf(['n = %d, xn = %16.12f, f(xn) = %16.12e, ' ...
            'error = %16.12e \n'], n+1, xn, yn, error)
        error = abs(xnp1 - xn);  %error value between successive points
        xnm1 = xn;
        xn = xnp1; %reassign upper and lower root values
        n = n+1; %increment iteration
    end
    fprintf(['n = %d, xn = %16.12f, f(xn) = %16.12e, ' ...
            'error = %16.12e \n'], n+1, xn, f(xn), error)
    root = xn;
end
```