Dendel Deguia

4.1.5

X	1	-2	0
Y	3	-3	-7

Table 5.1

$P(x) = 3 + 2(x-1) + 4(x-1)(x+2)$ (5.2)

$q(x) = 4x^2 + 6x - 7$ (5.3)

$P(1) = 3 + 2\cancel{(1-1)} + 4\cancel{(1-1)}(1+2) = 3 + 0 + 0 = 3$

$P(-2) = 3 + 2(-2-1) + 4(-2-1)\cancel{(-2+2)} = 3 - 6 + 0 = -3$

$P(0) = 3 + 2(0-1) + 4(0-1)(0+2) = 3 - 2 - 8 = -7$

$\Rightarrow$ $P(x)$ interpolates table 5.1

$q(1) = 4(1)^2 + 6(1) - 7 = 4 + 6 - 7 = 3$

$q(-2) = 4(-2)^2 + 6(-2) - 7 = 16 - 12 - 7 = -3$

$q(0) = 4(0)^2 + 6(0) - 7 = 0 + 0 - 7 = -7$

$\Rightarrow$ $q(x)$ interpolates table 5.1

By the existence and uniqueness theorem, this means that $p(x) = q(x)$; this becomes evident after multiplying out equation (5.2) and combining terms:

$P(x) = 3 + 2x - 2 + 4(x^2 + x - 2) = 1 + 2x + 4x^2 + 4x - 8$

$\Rightarrow$ $P(x) = 4x^2 + 6x - 7$

$\therefore$ $P(x) = q(x)$, so the existence and uniqueness theorem is not violated.

4.1.7 b) From the given table:

$$(x_0, x_1, x_2, x_3) = (-1, 1, 3, 4); \; (f[x_0], f[x_1], f[x_2], f[x_3]) = (2, -4, 46, 99.5)$$

and $f[x_2, x_3] = 53.5$

$$f[x_0, x_1] = \frac{f[x_1] - f[x_0]}{x_1 - x_0} = \frac{-4-2}{1-(-1)} = -3$$

$$f[x_1, x_2] = \frac{f[x_2] - f[x_1]}{x_2 - x_1} = \frac{46-(-4)}{3-1} = 25$$

$$f[x_0, x_1, x_2] = \frac{f[x_1, x_2] - f[x_0, x_1]}{x_2 - x_0} = \frac{25-(-3)}{3-(-1)} = 7$$

$$f[x_1, x_2, x_3] = \frac{f[x_2, x_3] - f[x_1, x_2]}{x_3 - x_1} = \frac{53.5-25}{4-1} = 9.5$$

$$f[x_0, x_1, x_2, x_3] = \frac{f[x_1, x_2, x_3] - f[x_0, x_1, x_2]}{x_3 - x_0} = \frac{9.5-7}{4-(-1)} = \frac{1}{2}$$

x	f[ ]	f[,]	f[, ,]	f[, , ,]
-1	$\boxed{2}$			
		$\boxed{-3}$		
1	-4		$\boxed{7}$	
		25		$\boxed{0.5}$
3	46		9.5	
		53.5		
4	99.5			

$$\Rightarrow \boxed{P_3(x) = 2 - 3(x+1) + 7(x+1)(x-1) + \tfrac{1}{2}(x+1)(x-1)(x-3)}$$

4.1.11

Year	1950	1960	1970	1980	1990
Population (mil)	150.7	179.3	203.3	226.5	249.6

Using divided difference yields:

$$
\begin{array}{llllll}
1950 & \boxed{150.7} & & & & \\
 & & \boxed{2.86} & & & \\
1960 & 179.3 & & \boxed{-0.023} & & \\
 & & 2.4 & & \boxed{0.0006\overline{333}} & \\
1970 & 203.3 & & -0.004 & & \boxed{-1.291\overline{66}\times10^{-5}} \\
 & & 2.32 & & 0.0001\overline{166} & \\
1980 & 226.5 & & -0.0005 & & \\
 & & 2.31 & & & \\
1990 & 249.6 & & & &
\end{array}
$$

$$
\begin{aligned}
\Rightarrow P_4(X) = {} & 150.7 + 2.86(X-1950) - \frac{23}{1000}(X-1950)(X-1960) \\
& + \frac{19}{30000}(X-1950)(X-1960)(X-1970) \\
& - \frac{31}{2400000}(X-1950)(X-1960)(X-1970)(X-1980)
\end{aligned}
$$

Now, using $P_4(X)$ to estimate populations in 1920 and 2000:

$$P_4(1920) = -47.2 \text{ million}$$

$$P_4(2000) = 270.2 \text{ million}$$

The estimate for $X=2000$ seems acceptable since there is an increasing trend in population w.r.t. year. However, the estimate for $X=1920$ is nonsensical which indicates that the predictive ~~capabilities~~ (and retrospective) capabilities of an interpolating polynomial is limited within the data it interpolates; beyond that data, ludicrous predictions are inevitable.

4.1.15 We can construct Newton's interpolation polynomial for the table. By the existence and uniqueness theorem, there is an interpolating polynomial of at most degree 5 that interpolates the table. Whatever polynomial is constructed is the only one possible for the provided data.

-2	1	3	2	-1	0	0
-1	4	7	-1	-1	0	
0	11	5	-4	-1		
1	16	-3	-7			
2	13	-17				
3	-4					

using divided differences, we see that the final two columns are zeros which indicates that the interpolating polynomial is degree 3

$$\Rightarrow \boxed{P(x) = 1 + 3(x+2) + 2(x+2)(x+1) - (x+2)(x+1)x}$$

Indeed, $P(x)$ is a 3^{rd} degree polynomial.

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
```

Computer Exercise 3.3.10

The following program will monitor the convergence of the ratio $\frac{e_{n+1}}{e_n e_{n-1}}$ when applying the secant method to $f(x) = arctan(x)$ where $e_n = r - x_n$ and r is a known root of $f(x)$. From properties of the indicated function we know that $r = 0$ is the only root of $arctan(x)$. From convergence properties of the secant method, we expect that the error ratio should converge to $-\frac{1}{2}\frac{f''(r)}{f'(r)}$ and from making the following evaluations:

$$f(x) = arctan(x) \Rightarrow f(x)|_{x=0} = 0$$

$$\frac{df}{dx} = \frac{1}{x^2+1} \Rightarrow \frac{df}{dx}|_{x=0} = 1$$

$$\frac{d^2f}{dx^2} = -\frac{2x}{(x^2+1)^2} \Rightarrow \frac{d^2f}{dx^2}|_{x=0} = 0$$

$$\Rightarrow \frac{e_{n+1}}{e_n e_{n-1}} \to -\frac{1}{2}\frac{f''(r)}{f'(r)} = -\frac{1}{2}\left(\frac{0}{1}\right) = 0 \text{ as } n \to \infty$$

Thus, we expect the error ratio to converge to zero.

```
r=0; %known root is used as input in the secant method function
f = @(x) atan(x);
x0 = -1.69; %initialize arbitrary points around x=0
x1 = 2.73;
N = 20; %max iterates
err = 0.5 * 10^(-20);

secant(f, x0, x1, N, err, r);
```

```
ratio =   0.073815824329
ratio =   0.580031172348
ratio =  -0.056781077976
ratio =  -0.171145861884
ratio =  -0.003155182463
ratio =   0.000320818071
ratio =  -0.000000015284
ratio =  -0.000000000000
```

Indeed, the error ratio does converge to zero with each iterate.

```
function secant(f, x0, x1, N, err, r) %No assigned output necessary
    %modified version of the earlier secant method I wrote that is
    %programmed to only output the error ratios
    xn = x1;
    xnm1 = x0;
    error = 1;
    n = 0;
    while (error > err) && (n < N)
        yn = f(xn);
        ynm1 = f(xnm1);
```

```
        xnp1 = xn - yn*((xn-xnm1)/(yn-ynm1));
        error = abs(xnp1 - xn);
        enm1 = r - xnm1; %error terms
        en = r - xn;
        enp1 = r - xnp1;
        q = enp1/(en*enm1); %error ratio
        fprintf('ratio = %16.12f\n', q)
        xnm1 = xn;
        xn = xnp1;
        n = n+1;
    end
end
```