CHAPTER 18

18.1 (a)

$$f_1(10) = 0.90309 + \frac{1.0791812 - 0.90309}{12-8}(10-8) = 0.991136$$

$$\varepsilon_t = \frac{1-0.991136}{1} \times 100\% = 0.886\%$$

(b)

$$f_1(10) = 0.9542425 + \frac{1.0413927 - 0.9542425}{11-9}(10-9) = 0.997818$$

$$\varepsilon_t = \frac{1-0.997818}{1} \times 100\% = 0.218\%$$

18.2 First, order the points

$x_0 = 9$	$f(x_0) = 0.9542425$
$x_1 = 11$	$f(x_1) = 1.0413927$
$x_2 = 8$	$f(x_2) = 0.9030900$

Applying Eq. (18.4)

$b_0 = 0.9542425$

Equation (18.5) yields

$$b_1 = \frac{1.0413927 - 0.9542425}{11-9} = 0.0435751$$

Equation (18.6) gives

$$b_2 = \frac{\dfrac{0.9030900 - 1.0413927}{8-11} - 0.0435751}{8-9} = \frac{0.0461009 - 0.0435751}{8-9} = -0.0025258$$

Substituting these values into Eq. (18.3) yields the quadratic formula

$$f_2(x) = 0.9542425 + 0.0435751(x-9) - 0.0025258(x-9)(x-11)$$

which can be evaluated at $x = 10$ for

$$f_2(10) = 0.9542425 + 0.0435751(10-9) - 0.0025258(10-9)(10-11) = 1.0003434$$

18.3 First, order the points

$x_0 = 9$	$f(x_0) = 0.9542425$
$x_1 = 11$	$f(x_1) = 1.0413927$
$x_2 = 8$	$f(x_2) = 0.9030900$
$x_3 = 12$	$f(x_3) = 1.0791812$

The first divided differences can be computed as

$$f[x_1,x_0]=\frac{1.0413927-0.9542425}{11-9}=0.0435751$$
$$f[x_2,x_1]=\frac{0.9030900-1.0413927}{8-11}=0.0461009$$
$$f[x_3,x_2]=\frac{1.0791812-0.9030900}{12-8}=0.0440228$$

The second divided differences are

$$f[x_2,x_1,x_0]=\frac{0.0461009-0.0435751}{8-9}=-0.0025258$$
$$f[x_3,x_2,x_1]=\frac{0.0440228-0.0461009}{12-11}=-0.0020781$$

The third divided difference is

$$f[x_3,x_2,x_1,x_0]=\frac{-0.0020781-(-0.0025258)}{12-9}=0.00014924$$

Substituting the appropriate values into Eq. (18.7) gives

$$f_3(x)=0.9542425+0.0435751(x-9)-0.0025258(x-9)(x-11)$$
$$+0.00014924(x-9)(x-11)(x-8)$$

which can be evaluated at x = 10 for

$$f_3(x)=0.9542425+0.0435751(10-9)-0.0025258(10-9)(10-11)$$
$$+0.00014924(10-9)(10-11)(10-8)=1.0000449$$

18.4

18.1 (a):

x_0 = 8	$f(x_0)$ = 0.9030900
x_1 = 12	$f(x_1)$ = 1.0791812

$$f_1(10)=\frac{10-12}{8-12}0.9030900+\frac{10-8}{12-8}1.0791812=0.991136$$

18.1 (b):

x_0 = 9	$f(x_0)$ = 0.9542425
x_1 = 11	$f(x_1)$ = 1.0413927

$$f_1(10)=\frac{10-11}{9-11}0.9542425+\frac{10-9}{11-9}1.0413927=0.997818$$

18.2:

x_0 = 8	$f(x_0)$ = 0.9030900
x_1 = 9	$f(x_1)$ = 0.9542425
x_2 = 11	$f(x_2)$ = 1.0413927

$$f_2(10) = \frac{(10-9)(10-11)}{(8-9)(8-11)}0.9030900 + \frac{(10-8)(10-11)}{(9-8)(9-11)}0.9542425$$
$$+\frac{(10-8)(10-9)}{(11-8)(11-9)}1.0413927 = 1.0003434$$

18.3:

$x_0 = 8$	$f(x_0) = 0.9030900$
$x_1 = 9$	$f(x_1) = 0.9542425$
$x_2 = 11$	$f(x_2) = 1.0413927$
$x_3 = 12$	$f(x_3) = 1.0791812$

$$f_3(10) = \frac{(10-9)(10-11)(10-12)}{(8-9)(8-11)(8-12)}0.9030900 + \frac{(10-8)(10-11)(10-12)}{(9-8)(9-11)(9-12)}0.9542425$$
$$+\frac{(10-8)(10-9)(10-12)}{(11-8)(11-9)(11-12)}1.0413927 + \frac{(10-8)(10-9)(10-11)}{(12-8)(12-9)(12-11)}1.0791812 = 1.0000449$$

18.5 First, order the points so that they are as close to and as centered about the unknown as possible

$x_0 = 2.5$	$f(x_0) = 14$
$x_1 = 3.2$	$f(x_1) = 15$
$x_2 = 2$	$f(x_2) = 8$
$x_3 = 4$	$f(x_3) = 8$
$x_4 = 1.6$	$f(x_4) = 2$

Next, the divided differences can be computed and displayed in the format of Fig. 18.5,

i	x_i	$f(x_i)$	$f[x_{i+1},x_i]$	$f[x_{i+2},x_{i+1},x_i]$	$f[x_{i+3},x_{i+2},x_{i+1},x_i]$	$f[x_{i+4},x_{i+3},x_{i+2},x_{i+1},x_i]$
0	2.5	14	1.428571	-8.809524	1.011905	1.847718
1	3.2	15	5.833333	-7.291667	-0.651042	
2	2	8	0	-6.25		
3	4	8	2.5			
4	1.6	2				

The first through third-order interpolations can then be implemented as

$$f_1(2.8) = 14 + 1.428571(2.8-2.5) = 14.428571$$
$$f_2(2.8) = 14 + 1.428571(2.8-2.5) - 8.809524(2.8-2.5)(2.8-3.2) = 15.485714$$
$$f_3(2.8) = 14 + 1.428571(2.8-2.5) - 8.809524(2.8-2.5)(2.8-3.2)$$
$$+1.011905(2.8-2.5)(2.8-3.2)(2.8-2.) = 15.388571$$

The error estimates for the first and second-order predictions can be computed with Eq. 18.19 as

$$R_1 = 15.485714 - 14.428571 = 1.057143$$
$$R_2 = 15.388571 - 15.485714 = -0.097143$$

The error for the third-order prediction can be computed with Eq. 18.18 as

$$R_3 = 1.847718(2.8-2.5)(2.8-3.2)(2.8-2)(2.8-4) = 0.212857$$

18.6 First, order the points so that they are as close to and as centered about the unknown as possible

$x_0 = 3$ $f(x_0) = 19$
$x_1 = 5$ $f(x_1) = 99$
$x_2 = 2$ $f(x_2) = 6$
$x_3 = 7$ $f(x_3) = 291$
$x_4 = 1$ $f(x_4) = 3$

Next, the divided differences can be computed and displayed in the format of Fig. 18.5,

i	x_i	$f(x_i)$	$f[x_{i+1},x_i]$	$f[x_{i+2},x_{i+1},x_i]$	$f[x_{i+3},x_{i+2},x_{i+1},x_i]$	$f[x_{i+4},x_{i+3},x_{i+2},x_{i+1},x_i]$
0	3	19	40	9	1	0
1	5	99	31	13	1	
2	2	6	57	9		
3	7	291	48			
4	1	3				

The first through fourth-order interpolations can then be implemented as

$$f_1(4) = 19 + 40(4-3) = 59$$
$$f_2(4) = 59 + 9(4-3)(4-5) = 50$$
$$f_3(4) = 50 \ + 1(4-3)(4-5)(4-2) = 48$$
$$f_4(4) = 48 + 0(4-3)(4-5)(4-2)(4-7) = 48$$

Clearly this data was generated with a cubic polynomial since the difference between the 4th and the 3rd-order versions is zero.

18.7
First order:
$x_0 = 3$ $f(x_0) = 19$
$x_1 = 5$ $f(x_1) = 99$

$$f_1(10) = \frac{4-5}{3-5}19 + \frac{4-3}{5-3}99 = 59$$

Second order:
$x_0 = 3$ $f(x_0) = 19$
$x_1 = 5$ $f(x_1) = 99$
$x_2 = 2$ $f(x_2) = 6$

$$f_2(10) = \frac{(4-5)(4-2)}{(3-5)(3-2)}19 + \frac{(4-3)(4-2)}{(5-3)(5-2)}99 \ + \frac{(4-3)(4-5)}{(2-3)(2-5)}6 = 50$$

Third order:
$x_0 = 3$ $f(x_0) = 19$
$x_1 = 5$ $f(x_1) = 99$
$x_2 = 2$ $f(x_2) = 6$
$x_3 = 7$ $f(x_3) = 291$

$$f_3(10) = \frac{(4-5)(4-2)(4-7)}{(3-5)(3-2)(3-7)}19 + \frac{(4-3)(4-2)(4-7)}{(5-3)(5-2)(5-7)}99$$
$$+ \frac{(4-3)(4-5)(4-7)}{(2-3)(2-5)(2-7)}6 + \frac{(4-3)(4-5)(4-2)}{(7-3)(7-5)(7-2)}291 = 48$$

18.8 First set up a difference table with the points properly ordered (proximity to and balance around the unknown):

x	*y*			
5	5.375	-3.45	0.375	0
1.8	16.415	-3.075	0.375	
6	3.5	-3.75		
0	26			

Notice that the zero third divided difference tips us off that the 2nd order polynomial will be exact.

Zero order:
$f_0(3.5) = 5.375$

First order:
$f_1(3.5) = 5.375 + (-3.45)(3.5 - 5) = 5.375 + 5.175 = 10.55$

Second-order
$f_2(3.5) = 10.55 + 0.375(3.5 - 5)\,(3.5 - 1.8) = 10.55 - 0.95625 = 9.59375$

Because the third divided difference is zero, we know we can stop.

18.9 First set up a difference table with the points properly ordered (proximity to and balance around the unknown):

x	*y*	first	second	third	fourth	fifth	sixth
3	7.5625	0.588533	0.083367	0.145826667	-7.61905E-06	1.26984E-06	7.90123E-07
4.5	8.4453	0.54685	0.37502	0.145841905	-3.80952E-06	-1.10053E-06	
2.5	7.3516	0.73436	-0.13543	0.14583619	1.14286E-06		
5	9.1875	0.9375	0.375	0.145833333			
1	5.4375	1.3125	-0.35417				
6	12	1.666667					
0	2						

Notice that the near-zero fourth divided difference tips us off that a third-order polynomial will be optimal.

Zero order:
$f_0(3.5) = 7.5625$

First order:
$f_1(3.5) = 7.5625 + 0.588533(3.5 - 3) = 7.5625 + 0.294266667 = 7.8567667$

Second-order
$f_2(3.5) = 7.8567667 + 0.083366667(3.5 - 3)(3.5 - 4.5) = 7.8567667 - 0.041683333 = 7.8150833$

Third-order
$f_3(3.5) = 7.8150833 + 0.14582667(3.5 - 3)(3.5 - 4.5)(3.5 - 2.5) = 7.8150833 - 0.072913333 = 7.74217$

Because the fourth divided difference is so small, we know we can stop because additional terms have only a marginal impact on the result.

18.10 First set up a difference table with the points properly ordered (proximity to and balance around the unknown):

x	*y*	first	second	third	fourth	fifth
5.5	9.9	0.054545	-0.06394	0.006911977	0.000165176	-4.13834E-05

11	10.2	-0.425	-0.08813	0.006168687	-0.000269351
13	9.35	0.368182	-0.14982	0.004821934	
2	5.3	2.166	-0.13535		
1	3.134	0.271067			
16	7.2				
0	0.5				

Zero order:
$f_0(8) = 9.9$

First order:
$f_1(8) = 9.9 + 0.054545(8 – 5.5) = 9.9 + 0.1363636 = 10.03636$

Second-order
$f_2(8) = 10.03636 – 0.06394(8 – 5.5)(8 – 11) = 10.03636 + 0.4795454 = 10.51591$

Third-order
$f_3(8) = 10.51591 + 0.006912(8 – 5.5)(8 – 11)(8 – 13) = 10.51591 + 0.259199 = 10.77511$

Fourth-order
$f_4(8) = 10.77511 + 0.000165176(8 – 5.5)(8 – 11)(8 – 13)(8 – 2) = 10.77511 + 0.03716 = 10.81227$

The fact that the fifth divided difference is getting small suggests that we are reaching the point of diminishing returns. In fact, beyond the fourth order, the results start to oscillate as shown in the following table:

Order	Increment	*f*(8)
0	9.9	9.90000
1	0.136363636	10.03636
2	0.479545455	10.51591
3	0.259199134	10.77511
4	0.037164502	10.81227
5	-0.065178932	10.74709
6	0.060233564	10.80733
7	-0.072824364	10.73450

18.11 The following points are used to generate a cubic interpolating polynomial

$x_0 = 3$	$f(x_0) = 0.3333$
$x_1 = 4$	$f(x_1) = 0.25$
$x_2 = 5$	$f(x_2) = 0.2$
$x_3 = 6$	$f(x_3) = 0.1667$

The polynomial can be generated in a number of ways including simultaneous equations (Eq. 18.26) or a software tool. The result is

$$f_3(x) = 0.949 - 0.329883x + 0.04985x^2 - 0.002767x^3$$

The roots problem can then be developed by setting this polynomial equal to the desired value of 0.23

$$0 = 0.719 - 0.329883x + 0.04985x^2 - 0.002767x^3$$

Bisection can then be used to determine the root. Using initial guesses of $x_l = 4$ and $x_u = 5$, the first five iterations are

i	x_l	x_u	x_r	$f(x_l)$	$f(x_r)$	$f(x_l)\times f(x_r)$	ε_a
1	4.00000	5.00000	4.50000	0.02000	-0.00813	-0.00016	11.11%
2	4.00000	4.50000	4.25000	0.02000	0.00503	0.00010	5.88%
3	4.25000	4.50000	4.37500	0.00503	-0.00176	-0.00001	2.86%
4	4.25000	4.37500	4.31250	0.00503	0.00158	0.00001	1.45%
5	4.31250	4.37500	4.34375	0.00158	-0.00010	0.00000	0.72%

If the iterations are continued, the final result is x = 4.34179.

18.12 (a) Analytically

$$0.85 = \frac{x^2}{1+x^2}$$

$$0.85 + 0.85x^2 = x^2$$

$$x = \sqrt{0.85/0.15} = 2.380476$$

(b) Cubic interpolation of x versus y

$y_0 = 0.5$	$x_0 = 1$
$y_1 = 0.8$	$x_1 = 2$
$y_2 = 0.9$	$x_2 = 3$
$y_3 = 0.941176$	$x_3 = 4$

The polynomial can be generated as

$$x = -62.971 + 282.46y - 404.83y^2 + 191.59y^3$$

This function can then be used to compute

$$x = -62.971 + 282.46(0.85) - 404.83(0.85)^2 + 191.59(0.85)^3 = 2.290694$$

$$\varepsilon_t = \left|\frac{2.380476 - 2.290694}{2.380476}\right| \times 100\% = 3.77\%$$

(c) Quadratic interpolation of y versus x yields

$x_0 = 2$	$f(x_0) = 0.8$
$x_1 = 3$	$f(x_1) = 0.9$
$x_2 = 4$	$f(x_2) = 0.941176$

The polynomial can be generated as

$$f_2(x) = 0.423529 + 0.247059x - 0.029412x^2$$

The roots problem can then be developed by setting this polynomial equal to the desired value of 0.85 to give

$$f_2(x) = -0.42647 + 0.247059x - 0.029412x^2$$

The quadratic formula can then be used to determine the root as

$$x = \frac{-0.247059 + \sqrt{(-0.247059)^2 - 4(-0.42647)(-0.029412)}}{2(-0.029412)} = 2.428009$$

$$\varepsilon_t = \left|\frac{2.380476 - 2.428009}{2.380476}\right| \times 100\% = 2.00\%$$

(d) Cubic interpolation of *y* versus *x* yields

$x_0 = 1$	$f(x_0) = 0.5$
$x_1 = 2$	$f(x_1) = 0.8$
$x_2 = 3$	$f(x_2) = 0.9$
$x_3 = 4$	$f(x_3) = 0.941176$

The polynomial can be generated as

$$f_3(x) = -0.14118 + 0.858824x - 0.24118x^2 + 0.023529x^3$$

The roots problem can then be developed by setting this polynomial equal to the desired value of 0.85

$$f_3(x) = -0.99118 + 0.858824x - 0.24118x^2 + 0.023529x^3$$

Bisection can then be used to determine the root. Using initial guesses of $x_l = 2$ and $x_u = 3$, the first five iterations are

i	x_l	x_u	x_r	$f(x_l)$	$f(x_r)$	$f(x_l)\times f(x_r)$	ε_a
1	2.00000	3.00000	2.50000	-0.05000	0.01617	-0.00081	20.00%
2	2.00000	2.50000	2.25000	-0.05000	-0.01177	0.00059	11.11%
3	2.25000	2.50000	2.37500	-0.01177	0.00335	-0.00004	5.26%
4	2.25000	2.37500	2.31250	-0.01177	-0.00390	0.00005	2.70%
5	2.31250	2.37500	2.34375	-0.00390	-0.00020	0.00000	1.33%

If the iterations are continued, the final result is *x* = 2.345481.

$$\varepsilon_t = \left|\frac{2.380476 - 2.345481}{2.380476}\right| \times 100\% = 1.47\%$$

18.13 For the present problem, we have five data points and *n* = 4 intervals. Therefore, 3(4) = 12 unknowns must be determined. Equations 18.29 and 18.30 yield 2(4) – 2 = 6 conditions

$$4a_1 + 2b_1 + c_1 = 8$$
$$4a_2 + 2b_2 + c_2 = 8$$
$$6.25a_2 + 2.5b_2 + c_2 = 14$$
$$6.25a_3 + 2.5b_3 + c_3 = 14$$
$$10.24a_3 + 3.2b_3 + c_3 = 15$$
$$10.24a_4 + 3.2b_4 + c_4 = 15$$

Passing the first and last functions through the initial and final values adds 2 more

$$2.56a_1 + 1.6b_1 + c_1 = 2$$
$$16a_4 + 4b_4 + c_4 = 8$$

Continuity of derivatives creates an additional 4 – 1 = 3.

$$4a_1 + b_1 = 4a_2 + b_2$$
$$5a_2 + b_2 = 5a_3 + b_3$$
$$6.4a_3 + b_3 = 6.4a_4 + b_4$$

Finally, Eq. 18.34 specifies that $a_1 = 0$. Thus, the problem reduces to solving 11 simultaneous equations for 11 unknown coefficients,

$$\begin{bmatrix} 2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 4 & 2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 6.25 & 2.5 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 6.25 & 2.5 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 10.24 & 3.2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 10.24 & 3.2 & 1 \\ 1.6 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 16 & 4 & 1 \\ 1 & 0 & -4 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 5 & 1 & 0 & -5 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 6.4 & 1 & 0 & -6.4 & -1 & 0 \end{bmatrix} \begin{Bmatrix} b_1 \\ c_1 \\ a_2 \\ b_2 \\ c_2 \\ a_3 \\ b_3 \\ c_3 \\ a_4 \\ b_4 \\ c_4 \end{Bmatrix} = \begin{Bmatrix} 8 \\ 8 \\ 14 \\ 14 \\ 15 \\ 15 \\ 2 \\ 8 \\ 0 \\ 0 \\ 0 \end{Bmatrix}$$

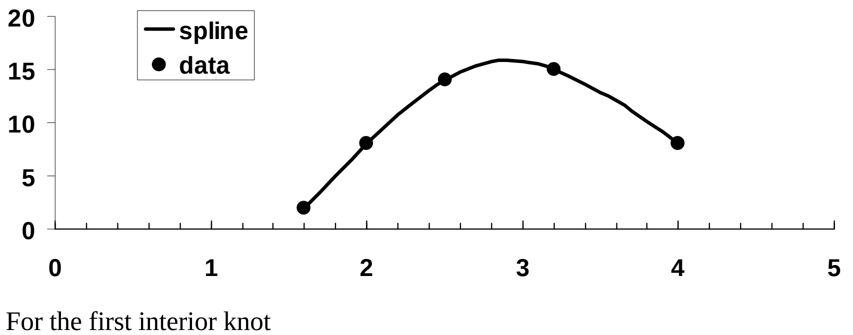
which can be solved for

$b_1 = 15$	$c_1 = -22$	
$a_2 = -6$	$b_2 = 39$	$c_2 = -46$
$a_3 = -10.816327$	$b_3 = 63.081633$	$c_3 = -76.102041$
$a_4 = -3.258929$	$b_4 = 14.714286$	$c_4 = 1.285714$

The predictions can be made as

$$f(3.4) = -3.258929(3.4)^2 + 14.714286(3.4) + 1.285714 = 13.64107$$
$$f(2.2) = -6(2.2)^2 + 39(2.2) - 46 = 10.76$$

Finally, here is a plot of the data along with the quadratic spline,



18.14 For the first interior knot

$x_0 = 1$	$f(x_0) = 3$
$x_1 = 2$	$f(x_1) = 6$

$x_2 = 3$ $f(x_2) = 19$

$$(2-1)f''(1) + 2(3-1)f''(2) + (3-2)f''(3) = \frac{6}{3-2}(19-6) + \frac{6}{2-1}(3-6)$$

Because of the natural spline condition, *f*'(1) = 0, and the equation reduces to

$$4f''(2) + f''(3) = 60$$

Equations can be written for the remaining interior knots and the results assembled in matrix form as

$$\begin{bmatrix} 4 & 1 & 0 & 0 \\ 1 & 6 & 2 & 0 \\ 0 & 2 & 8 & 2 \\ 0 & 0 & 2 & 6 \end{bmatrix} \begin{Bmatrix} f''(2) \\ f''(3) \\ f''(5) \\ f''(7) \end{Bmatrix} = \begin{Bmatrix} 60 \\ 162 \\ 336 \\ 342 \end{Bmatrix}$$

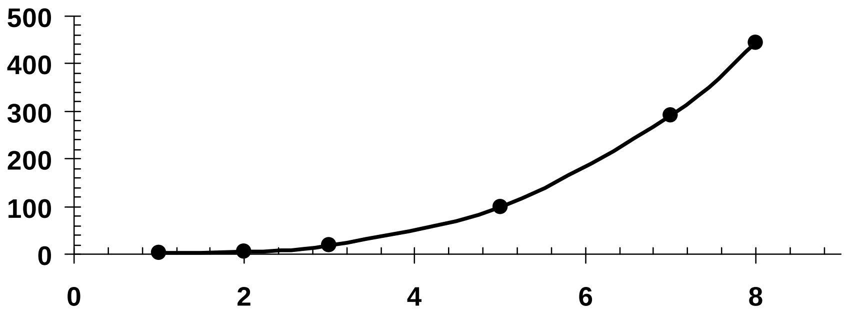
which can be solved for

$$f''(2) = 10.84716$$
$$f''(3) = 16.61135$$
$$f''(5) = 25.74236$$
$$f''(7) = 48.41921$$

These values can be used in conjunction with Eq. 18.36 to yield the following interpolating splines for each interval,

$$f_1(x) = 1.80786(x-1)^3 + 3(2-x) + 4.19214(x-1)$$
$$f_2(x) = 1.80786(3-x)^3 + 2.768559(x-2)^3 + 4.19214(3-x) + 16.23144(x-2)$$
$$f_3(x) = 1.384279(5-x)^3 + 2.145197(x-3)^3 + 3.962882(5-x) + 40.91921(x-3)$$
$$f_4(x) = 2.145197(7-x)^3 + 4.034934(x-5)^3 + 40.91921(7-x) + 129.3603(x-5)$$
$$f_5(x) = 8.069869(8-x)^3 + 282.9301(8-x) + 444(x-7)$$

The interpolating splines can be used to make predictions along the interval. The results are shown in the following plot.



(a) The interpolating equations can be used to determine

$f_3(4) = 48.41157$
$f_2(2.5) = 10.78384$

(b)

$$f_2(3) = 1.80786(3-3)^3 + 2.768559(3-2)^3 + 4.19214(3-3) + 16.23144(3-2) = 19$$

$$f_3(3) = 1.384279(5-3)^3 + 2.145197(3-3)^3 + 3.962882(5-3) + 40.91921(3-3) = 19$$

18.15 The points to be fit are

$x_0 = 3.2$	$f(x_0) = 15$
$x_1 = 4$	$f(x_1) = 8$
$x_2 = 4.5$	$f(x_2) = 2$

Using Eq. 18.26 the following simultaneous equations can be generated

$$a_0 + 3.2a_1 + 10.24a_2 = 15$$
$$a_0 + 4a_1 + 16a_2 = 8$$
$$a_0 + 4.5a_1 + 20.25a_2 = 2$$

These can be solved for $a_0 = 11$, $a_1 = 9.25$, and $a_2 = -2.5$. Therefore, the interpolating polynomial is

$f(x) = 11 + 9.25x - 2.5x^2$

18.16 The points to be fit are

$x_0 = 1$	$f(x_0) = 3$
$x_1 = 2$	$f(x_1) = 6$
$x_2 = 3$	$f(x_2) = 19$
$x_3 = 5$	$f(x_3) = 99$

Using Eq. 18.26 the following simultaneous equations can be generated

$$a_0 + a_1 + a_2 + a_3 = 3$$
$$a_0 + 2a_1 + 4a_2 + 8a_3 = 6$$
$$a_0 + 3a_1 + 9a_2 + 27a_3 = 19$$
$$a_0 + 5a_1 + 25a_2 + 125a_3 = 99$$

These can be solved for $a_0 = 4$, $a_1 = -1$, $a_2 = -1$, and $a_3 = 1$. Therefore, the interpolating polynomial is

$f(x) = 4 - x - x^2 + x^3$

18.17 Here is a VBA/Excel program to implement Newton interpolation.

```
Option Explicit
Sub Newt()
Dim n As Integer, i As Integer
Dim yint(10) As Double, x(10) As Double, y(10) As Double
Dim ea(10) As Double, xi As Double
Sheets("Sheet1").Select
Range("a5").Select
n = ActiveCell.Row
Selection.End(xlDown).Select
n = ActiveCell.Row - n
Range("a5").Select
For i = 0 To n
```

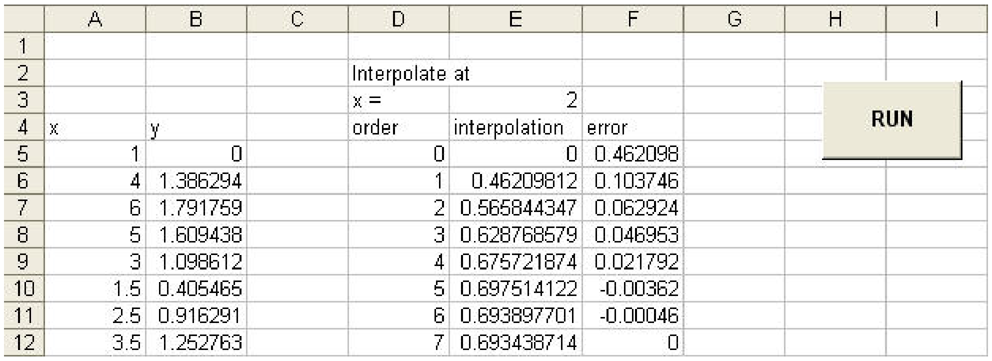
```
  x(i) = ActiveCell.Value
  ActiveCell.Offset(0, 1).Select
  y(i) = ActiveCell.Value
  ActiveCell.Offset(1, -1).Select
Next i
Range("e3").Select
xi = ActiveCell.Value
Call Newtint(x, y, n, xi, yint, ea)
Range("d5:f25").ClearContents
Range("d5").Select
For i = 0 To n
  ActiveCell.Value = i
  ActiveCell.Offset(0, 1).Select
  ActiveCell.Value = yint(i)
  ActiveCell.Offset(0, 1).Select
  ActiveCell.Value = ea(i)
  ActiveCell.Offset(1, -2).Select
Next i
Range("a5").Select
End Sub

Sub Newtint(x, y, n, xi, yint, ea)
Dim i As Integer, j As Integer, order As Integer
Dim fdd(10, 10) As Double, xterm As Double
Dim yint2 As Double
For i = 0 To n
  fdd(i, 0) = y(i)
Next i
For j = 1 To n
  For i = 0 To n - j
    fdd(i, j) = (fdd(i + 1, j - 1) - fdd(i, j - 1)) / (x(i + j) - x(i))
  Next i
Next j
xterm = 1#
yint(0) = fdd(0, 0)
For order = 1 To n
  xterm = xterm * (xi - x(order - 1))
  yint2 = yint(order - 1) + fdd(0, order) * xterm
  ea(order - 1) = yint2 - yint(order - 1)
  yint(order) = yint2
Next order
End Sub
```

For those who are using MATLAB, here is an M-file that implements Newton interpolation:

```
function yint = Newtint(x,y,xx)
% yint = Newtint(x,y,xx):
%   Newton interpolation. Uses an (n - 1)-order Newton
%   interpolating polynomial based on n data points (x, y)
%   to determine a value of the dependent variable (yint)
%   at a given value of the independent variable, xx.
% input:
%   x = independent variable
%   y = dependent variable
%   xx = value of independent variable at which
%        interpolation is calculated
% output:
%   yint = interpolated value of dependent variable

% compute the finite divided differences in the form of a
% difference table
```

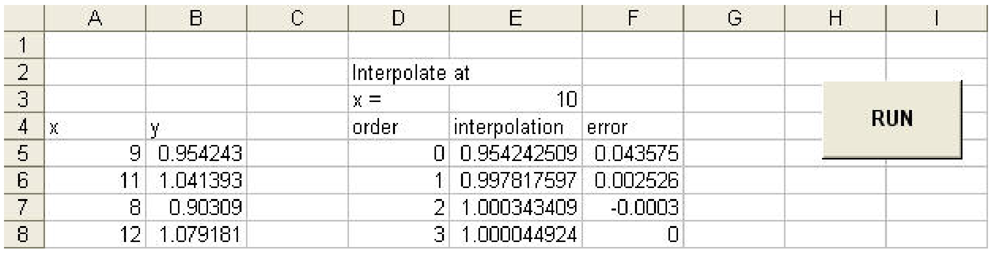
```
n = length(x);
if length(y)~=n, error('x and y must be same length'); end
b = zeros(n,n);
% assign dependent variables to the first column of b.
b(:,1) = y(:);  % the (:) ensures that y is a column vector.
for j = 2:n
  for i = 1:n-j+1
    b(i,j) = (b(i+1,j-1)-b(i,j-1))/(x(i+j-1)-x(i));
  end
end
% use the finite divided differences to interpolate
xt = 1;
yint = b(1,1);
for j = 1:n-1
  xt = xt*(xx-x(j));
  yint = yint+b(1,j+1)*xt;
end
```

18.18 Here is the solution when the Excel VBA program from Prob. 18.17 is run for Example 18.5.



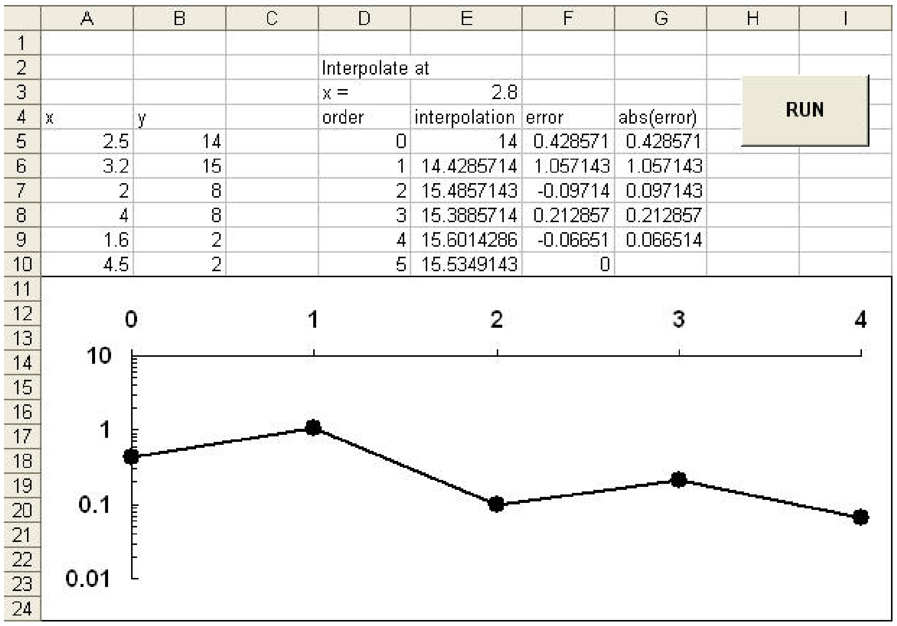
	A	B	C	D	E	F	G	H	I
1									
2				Interpolate at					
3				x =	2				
4	x	y		order	interpolation	error			
5	1	0		0	0	0.462098			
6	4	1.386294		1	0.46209812	0.103746			
7	6	1.791759		2	0.565844347	0.062924			
8	5	1.609438		3	0.628768579	0.046953			
9	3	1.098612		4	0.675721874	0.021792			
10	1.5	0.405465		5	0.697514122	-0.00362			
11	2.5	0.916291		6	0.693897701	-0.00046			
12	3.5	1.252763		7	0.693438714	0			

18.19 Here are the solutions when the program from Prob. 18.17 is run for Probs. 18.1 through 18.3.



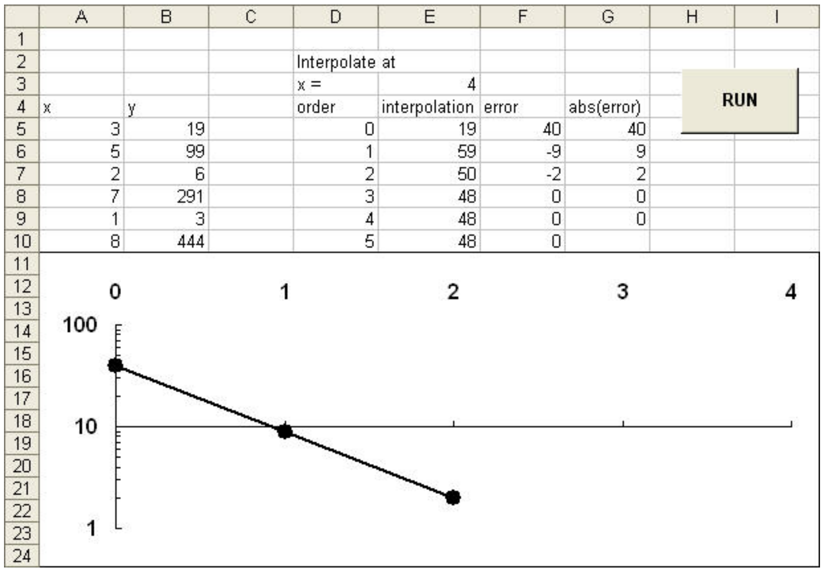
	A	B	C	D	E	F	G	H	I
1									
2				Interpolate at					
3				x =	10				
4	x	y		order	interpolation	error			
5	9	0.954243		0	0.954242509	0.043575			
6	11	1.041393		1	0.997817597	0.002526			
7	8	0.90309		2	1.000343409	-0.0003			
8	12	1.079181		3	1.000044924	0			

18.20 A plot of the error can easily be added to the Excel application. Note that we plot the absolute value of the error on a logarithmic scale. The following shows the solution for Prob. 18.5:



	A	B	C	D	E	F	G	H	I
1									
2				Interpolate at					
3				x =	2.8				
4	x	y		order	interpolation	error	abs(error)		
5	2.5	14		0	14	0.428571	0.428571		
6	3.2	15		1	14.4285714	1.057143	1.057143		
7	2	8		2	15.4857143	-0.09714	0.097143		
8	4	8		3	15.3885714	0.212857	0.212857		
9	1.6	2		4	15.6014286	-0.06651	0.066514		
10	4.5	2		5	15.5349143	0			

The following shows the solution for Prob. 18.6:

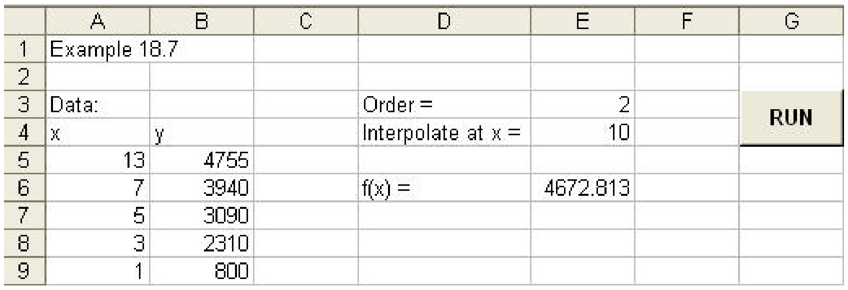


	A	B	C	D	E	F	G	H	I
1									
2				Interpolate at					
3				x =	4				
4	x	y		order	interpolation	error	abs(error)		
5	3	19		0	19	40	40		
6	5	99		1	59	-9	9		
7	2	6		2	50	-2	2		
8	7	291		3	48	0	0		
9	1	3		4	48	0	0		
10	8	444		5	48	0			

18.21

```
Option Explicit
Sub LagrInt()
Dim n As Integer, i As Integer, order As Integer
Dim x(10) As Double, y(10) As Double, xi As Double
Range("a5").Select
n = ActiveCell.Row
Selection.End(xlDown).Select
n = ActiveCell.Row - n
Range("a5").Select
For i = 0 To n
  x(i) = ActiveCell.Value
  ActiveCell.Offset(0, 1).Select
  y(i) = ActiveCell.Value
  ActiveCell.Offset(1, -1).Select
Next i
Range("e3").Select
order = ActiveCell.Value
ActiveCell.Offset(1, 0).Select
xi = ActiveCell.Value
ActiveCell.Offset(2, 0).Select
ActiveCell.Value = Lagrange(x, y, order, xi)
End Sub
Function Lagrange(x, y, order, xi)
Dim i As Integer, j As Integer
Dim sum As Double, prod As Double
sum = 0#
For i = 0 To order
  prod = y(i)
  For j = 0 To order
    If i <> j Then
      prod = prod * (xi - x(j)) / (x(i) - x(j))
    End If
  Next j
  sum = sum + prod
Next i
Lagrange = sum
End Function
```

Application to Example 18.7:



	A	B	C	D	E	F	G
1	Example 18.7						
2							
3	Data:			Order =	2		RUN
4	x	y		Interpolate at x =	10		
5	13	4755					
6	7	3940		f(x) =	4672.813		
7	5	3090					
8	3	2310					
9	1	800					

For those who are using MATLAB, here is an M-file that implements Lagrange interpolation:

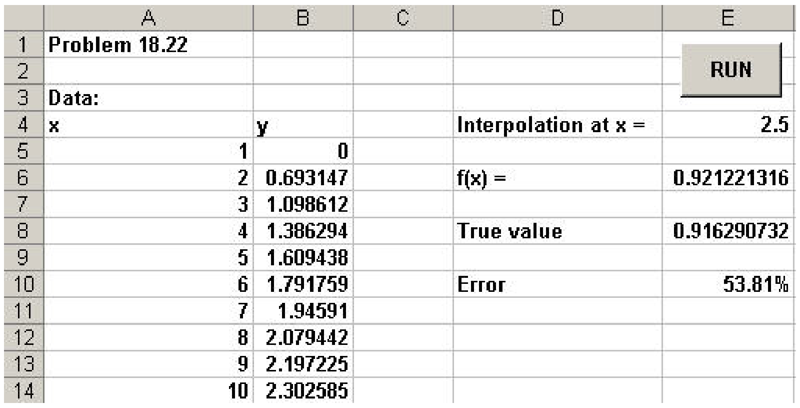
```
function yint = Lagrange(x,y,xx)
% yint = Lagrange(x,y,xx):
%   Lagrange interpolation. Uses an (n - 1)-order Lagrange
%   interpolating polynomial based on n data points (x, y)
%   to determine a value of the dependent variable (yint)
%   at a given value of the independent variable, xx.
% input:
%   x = independent variable
%   y = dependent variable
%   xx = value of independent variable at which the
%        interpolation is calculated
% output:
%   yint = interpolated value of dependent variable

n = length(x);
if length(y)~=n, error('x and y must be same length'); end
s = 0;
for i = 1:n
  product = y(i);
  for j = 1:n
```

```
      if i ~= j
        product = product*(xx-x(j))/(x(i)-x(j));
      end
    end
    s = s+product;
  end
  yint = s;
```

18.22 The following VBA program uses cubic interpolation for all intervals:

```
Option Explicit
Sub TableLook()
Dim n As Integer, i As Integer
Dim x(10) As Double, y(10) As Double
Dim xi As Double
Range("a5").Select
n = ActiveCell.Row
Selection.End(xlDown).Select
n = ActiveCell.Row - n
Range("a5").Select
For i = 0 To n
  x(i) = ActiveCell.Value
  ActiveCell.Offset(0, 1).Select
  y(i) = ActiveCell.Value
  ActiveCell.Offset(1, -1).Select
Next i
Range("e4").Select
xi = ActiveCell.Value
ActiveCell.Offset(2, 0).Select
ActiveCell.Value = Interp(x, y, n, xi)
Range("a5").Select
End Sub
Function Interp(x, y, n, xx)
Dim ii As Integer
If xx < x(0) Or xx > x(n) Then
  Interp = "out of range"
Else
  If xx <= x(ii + 1) Then
    Interp = Lagrange(x, y, 0, 3, xx)
  ElseIf xx <= x(n - 1) Then
  For ii = 0 To n - 2
    If xx >= x(ii) And xx <= x(ii + 1) Then
      Interp = Lagrange(x, y, ii - 1, 3, xx)
      Exit For
    End If
  Next ii
  Else
    Interp = Lagrange(x, y, n - 3, 3, xx)
  End If
End If
End Function
Function Lagrange(x, y, i0, order, xi)
Dim i As Integer, j As Integer
Dim sum As Double, prod As Double
sum = 0#
For i = i0 To i0 + order
  prod = y(i)
  For j = i0 To i0 + order
    If i <> j Then
      prod = prod * (xi - x(j)) / (x(i) - x(j))
    End If
  Next j
  sum = sum + prod
Next i
Lagrange = sum
End Function
```

Application to evaluate ln(2.5):



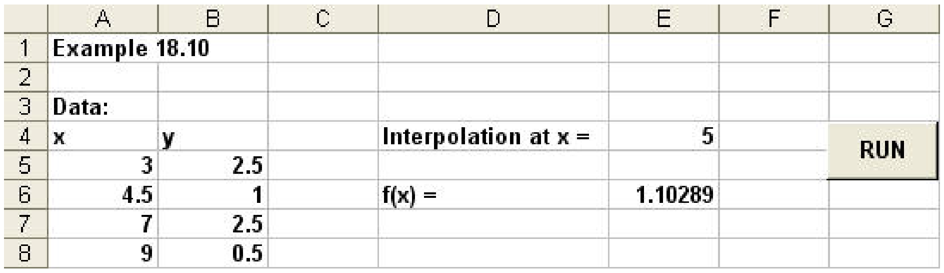
	A	B	C	D	E
1	Problem 18.22				
2					RUN
3	Data:				
4	x	y		Interpolation at x =	2.5
5	1	0			
6	2	0.693147		f(x) =	0.921221316
7	3	1.098612			
8	4	1.386294		True value	0.916290732
9	5	1.609438			
10	6	1.791759		Error	53.81%
11	7	1.94591			
12	8	2.079442			
13	9	2.197225			
14	10	2.302585			

18.23

```
Option Explicit
Sub Splines()
Dim i As Integer, n As Integer
Dim x(7) As Double, y(7) As Double, xu As Double, yu As Double
Dim dy As Double, d2y As Double
Range("a5").Select
n = ActiveCell.Row
Selection.End(xlDown).Select
n = ActiveCell.Row - n
Range("a5").Select
For i = 0 To n
  x(i) = ActiveCell.Value
  ActiveCell.Offset(0, 1).Select
  y(i) = ActiveCell.Value
  ActiveCell.Offset(1, -1).Select
Next i
Range("e4").Select
xu = ActiveCell.Value
Call Spline(x, y, n, xu, yu, dy, d2y)
ActiveCell.Offset(2, 0).Select
ActiveCell.Value = yu
End Sub
Sub Spline(x, y, n, xu, yu, dy, d2y)
Dim e(10) As Double, f(10) As Double, g(10) As Double, r(10) As Double,
d2x(10) As Double
Call Tridiag(x, y, n, e, f, g, r)
Call Decomp(e, f, g, n - 1)
Call Substit(e, f, g, r, n - 1, d2x)
Call Interpol(x, y, n, d2x, xu, yu, dy, d2y)
End Sub

Sub Tridiag(x, y, n, e, f, g, r)
Dim i As Integer
f(1) = 2 * (x(2) - x(0))
g(1) = x(2) - x(1)
r(1) = 6 / (x(2) - x(1)) * (y(2) - y(1))
```

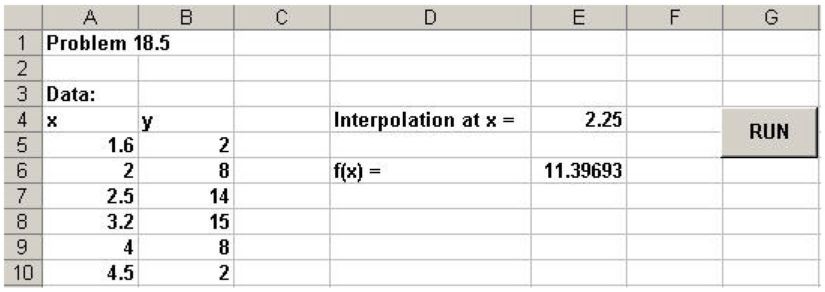
```
r(1) = r(1) + 6 / (x(1) - x(0)) * (y(0) - y(1))
For i = 2 To n - 2
  e(i) = x(i) - x(i - 1)
  f(i) = 2 * (x(i + 1) - x(i - 1))
  g(i) = x(i + 1) - x(i)
  r(i) = 6 / (x(i + 1) - x(i)) * (y(i + 1) - y(i))
  r(i) = r(i) + 6 / (x(i) - x(i - 1)) * (y(i - 1) - y(i))
Next i
e(n - 1) = x(n - 1) - x(n - 2)
f(n - 1) = 2 * (x(n) - x(n - 2))
r(n - 1) = 6 / (x(n) - x(n - 1)) * (y(n) - y(n - 1))
r(n - 1) = r(n - 1) + 6 / (x(n - 1) - x(n - 2)) * (y(n - 2) - y(n - 1))
End Sub

Sub Interpol(x, y, n, d2x, xu, yu, dy, d2y)
Dim i As Integer, flag As Integer
Dim c1 As Double, c2 As Double, c3 As Double, c4 As Double
Dim t1 As Double, t2 As Double, t3 As Double, t4 As Double
flag = 0
i = 1
Do
  If xu >= x(i - 1) And xu <= x(i) Then
    c1 = d2x(i - 1) / 6 / (x(i) - x(i - 1))
    c2 = d2x(i) / 6 / (x(i) - x(i - 1))
    c3 = y(i - 1) / (x(i) - x(i - 1)) - d2x(i - 1) * (x(i) - x(i - 1)) / 6
    c4 = y(i) / (x(i) - x(i - 1)) - d2x(i) * (x(i) - x(i - 1)) / 6
    t1 = c1 * (x(i) - xu) ^ 3
    t2 = c2 * (xu - x(i - 1)) ^ 3
    t3 = c3 * (x(i) - xu)
    t4 = c4 * (xu - x(i - 1))
    yu = t1 + t2 + t3 + t4
    t1 = -3 * c1 * (x(i) - xu) ^ 2
    t2 = 3 * c2 * (xu - x(i - 1)) ^ 2
    t3 = -c3
    t4 = c4
    dy = t1 + t2 + t3 + t4
    t1 = 6 * c1 * (x(i) - xu)
    t2 = 6 * c2 * (xu - x(i - 1))
    d2y = t1 + t2
    flag = 1
  Else
    i = i + 1
  End If
  If i = n + 1 Or flag = 1 Then Exit Do
Loop
If flag = 0 Then
  MsgBox "outside range"
  End
End If
End Sub
Sub Decomp(e, f, g, n)
Dim k As Integer
For k = 2 To n
  e(k) = e(k) / f(k - 1)
  f(k) = f(k) - e(k) * g(k - 1)
Next k
End Sub
Sub Substit(e, f, g, r, n, x)
Dim k As Integer
For k = 2 To n
  r(k) = r(k) - e(k) * r(k - 1)
Next k
```

```
x(n) = r(n) / f(n)
For k = n - 1 To 1 Step -1
  x(k) = (r(k) - g(k) * x(k + 1)) / f(k)
Next k
End Sub
```



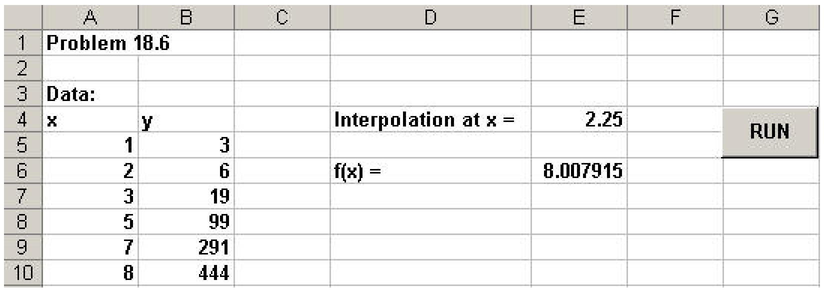
	A	B	C	D	E	F	G
1	Example 18.10						
2							
3	Data:						
4	x	y		Interpolation at x =	5		
5	3	2.5					
6	4.5	1		f(x) =	1.10289		
7	7	2.5					
8	9	0.5					

18.24 The following shows the solution for the data from Prob. 18.5 evaluated at 2.25:



	A	B	C	D	E	F	G
1	Problem 18.5						
2							
3	Data:						
4	x	y		Interpolation at x =	2.25		
5	1.6	2					
6	2	8		f(x) =	11.39693		
7	2.5	14					
8	3.2	15					
9	4	8					
10	4.5	2					

The following shows the solution for the data from Prob. 18.6 evaluated at 2.25:



	A	B	C	D	E	F	G
1	Problem 18.6						
2							
3	Data:						
4	x	y		Interpolation at x =	2.25		
5	1	3					
6	2	6		f(x) =	8.007915		
7	3	19					
8	5	99					
9	7	291					
10	8	444					

18.25 (a) Linear interpolation:

$$s = 6.4147 + \frac{6.5453 - 6.4147}{0.11144 - 0.10377}(0.108 - 0.10377) = 6.486726$$

(b) Quadratic interpolation:

$$s = 6.486726 + \frac{15.83811 - 17.02738}{0.1254 - 0.10377}(0.108 - 0.10377)(0.108 - 0.11144) = 6.487526$$

(c) Inverse interpolation. First, we fit a quadratic to the data

$$f_2(v) = 4.011945 + 28.860154v - 54.982456v^2$$

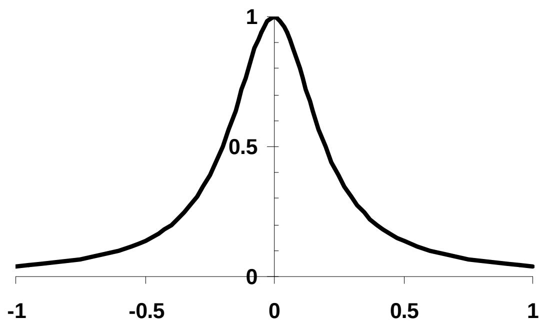
The roots problem can then be developed by setting this polynomial equal to the desired value of 6.6 to give

$$f_2(v) = -2.588055 + 28.860154v - 54.982456v^2$$

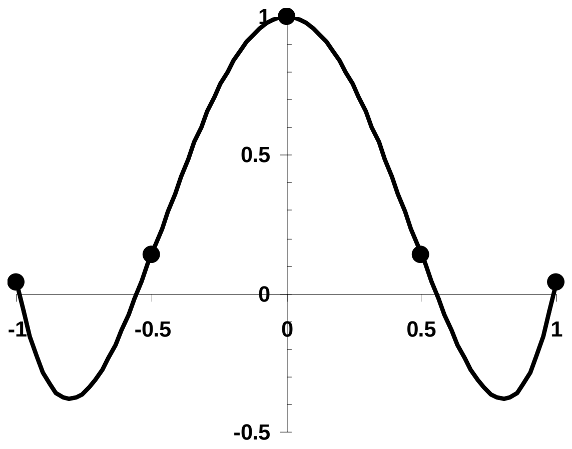
The quadratic formula can then be used to determine the root as

$$v = \frac{-28.860154 + \sqrt{(28.860154)^2 - 4(-54.982456)(-2.588055)}}{2(-54.982456)} = 0.11477$$

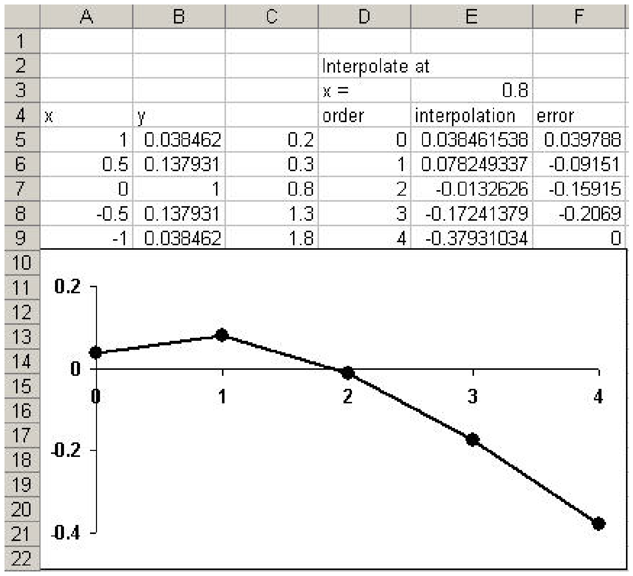
18.26 (a)



(b)

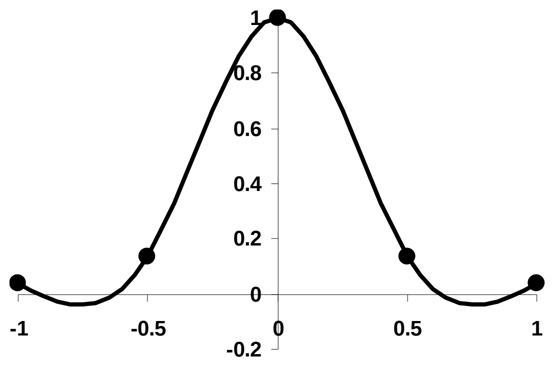


(c) After ordering the points so that they are as close to and as centered about the unknown as possible, the program developed in Prob. 18.17 can be used to implement Newton's method. As indicated below the results indicate that the method is not converging on a stable value.



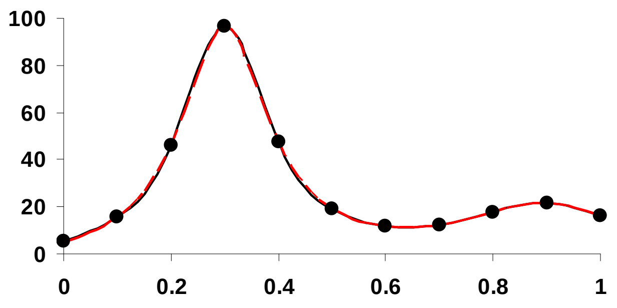
	A	B	C	D	E	F
1						
2				Interpolate at		
3				x =	0.8	
4	x	y		order	interpolation	error
5	1	0.038462	0.2	0	0.038461538	0.039788
6	0.5	0.137931	0.3	1	0.078249337	-0.09151
7	0	1	0.8	2	-0.0132626	-0.15915
8	-0.5	0.137931	1.3	3	-0.17241379	-0.2069
9	-1	0.038462	1.8	4	-0.37931034	0

(d) A cubic spline can be generated using the 5 points from **(b)**:



(e) Although Runge's function is an extreme case, the results indicate that care should be taken when using higher-order polynomials for interpolation. Of all the results, the spline is the best because by limiting itself to cubics, it is more constrained and hence less liable to oscillations than the Newton and Lagrange versions.

18.27 Using a program based on Fig. 18.18, a cubic spline can be generated for the humps function. The following plot displays the points, the spline (black solid line) and the actual humps function (red dashed line). As can be seen the actual function and the spline fit are in close agreement.



18.28 (a) linear interpolation yields

$$o_1(27) = 8.418 + \frac{7.305 - 8.418}{32 - 24}(27 - 24) = 8.000625$$

$$\varepsilon_t = \left|\frac{7.986 - 8.000625}{7.986}\right| \times 100\% = 0.183\%$$

(b) After ordering the points so that they are as close to and as centered about the unknown as possible, a difference table can be developed:

T	*o*	first	second	third	fourth	fifth
24	8.418	-0.13913	0.002648	-3.84115E-05	6.51042E-07	-9.6639E-09
32	7.305	-0.16031	0.002034	-4.88281E-05	8.82975E-07	
16	9.87	-0.14404	0.003206	-7.70833E-05		
40	6.413	-0.16969	0.004439			
8	11.8430	-0.34725				
0	14.621					

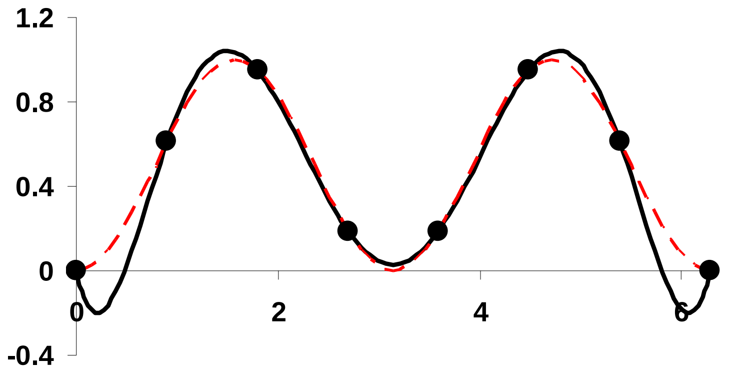
These can then be used to generate the various order polynomials as tabulated below:

Order	**Increment**	**o(27)**	ε_t
0	8.418	8.418	5.409%
1	-0.417375	8.000625	0.183%
2	-0.039726563	7.960898	0.314%
3	0.006337891	7.967236	0.235%
4	0.001396484	7.968633	0.217%
5	-0.000393852	7.968239	0.222%

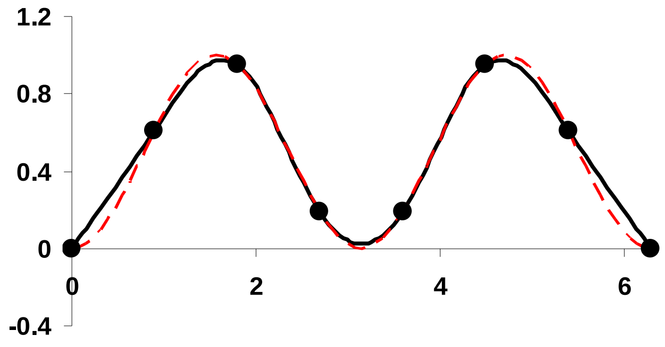
Therefore, the estimate with Newton's method would be 7.968 mg/L with an error of about 0.22%.

(c) Applying a cubic spline using the algorithm from Fig. 18.18 yields a prediction of *o*(27) = 7.9657 with an error of 0.254%.

18.29 (a) A 7th-order interpolating polynomial (solid line) is shown in the following graph along with the 8 points as well as the exact function (dashed line).



(b) The cubic spline fit (solid line) is shown in the following graph along with the 8 points as well as the exact function (dashed line). The spline clearly does a better job than the higher-order polynomial.



18.30 This part of the problem is well-suited for Newton interpolation. First, order the points so that they are as close to and as centered about the unknown as possible, for $x = 4$.

$y_0 = 4$	$f(y_0) = 38.43$
$y_1 = 2$	$f(y_1) = 53.5$
$y_2 = 6$	$f(y_2) = 30.39$
$y_3 = 0$	$f(y_3) = 80$
$y_4 = 8$	$f(y_4) = 30$

The results of applying Newton's polynomial at $y = 3.2$ are

Order	*f*(*y*)	Error
0	38.43	6.028
1	44.458	-0.8436
2	43.6144	-0.2464
3	43.368	0.112448
4	43.48045	

The minimum error occurs for the third-order version so we conclude that the interpolation is 43.368.

(b) This is an example of two-dimensional interpolation. One way to approach it is to use cubic interpolation along the *y* dimension for values at specific values of *x* that bracket the unknown. For example, we can utilize the following points at $x = 2$.

$y_0 = 0$	$f(y_0) = 90$
$y_1 = 2$	$f(y_1) = 64.49$
$y_2 = 4$	$f(y_2) = 48.9$
$y_3 = 6$	$f(y_3) = 38.78$

$T(x = 2, y = 2.7) = 58.13288438$

All the values can be tabulated as

$T(x = 2, y = 2.7) = 58.13288438$
$T(x = 4, y = 2.7) = 47.1505625$
$T(x = 6, y = 2.7) = 42.74770188$
$T(x = 8, y = 2.7) = 46.5$

These values can then be used to interpolate at $x = 4.3$ to yield

$T(x = 4.3, y = 2.7) = 46.03218664$

Note that some software packages allow you to perform such multi-dimensional interpolations very efficiently. For example, MATLAB has a function interp2 that provides numerous options for how the interpolation is implemented. Here is an example of how it can be implemented using linear interpolation,

```
>> Z=[100 90 80 70 60;
85 64.49 53.5 48.15 50;
70 48.9 38.43 35.03 40;
55 38.78 30.39 27.07 30;
40 35 30 25 20];
>> X=[0 2 4 6 8];
>> Y=[0 2 4 6 8];
>> T=interp2(X,Y,Z,4.3,2.7)
T =
   47.5254
```

It can also perform the same interpolation but using bicubic interpolation,

```
>> T=interp2(X,Y,Z,4.3,2.7,'cubic')
T =
   46.0062
```

Finally, the interpolation can be implemented using splines,

```
>> T=interp2(X,Y,Z,4.3,2.7,'spline')
T =
   46.1507
```