CHAPTER 19

19.1 In the following equations, $\omega_0 = 2\pi/T$

$$\frac{\int_0^T \sin(\omega_0 t)dt}{T} = \frac{-\omega_0[\cos(\omega_0 t)]_0^T}{T} = \frac{-\omega_0}{T}(\cos 2\pi - \cos 0) = 0$$

$$\frac{\int_0^T \cos(\omega_0 t)dt}{T} = \frac{\omega_0[\sin(\omega_0 t)]_0^T}{T} = \frac{\omega_0}{T}(\sin 2\pi - \sin 0) = 0$$

$$\frac{\int_0^T \sin^2(\omega_0 t)dt}{T} = \frac{\left[\frac{t}{2} - \frac{\sin(2\omega_0 t)}{4\omega_0}\right]_0^T}{T} = \frac{\frac{T}{2} - \frac{\sin 4\pi}{4\omega_0} - 0 + 0}{T} = \frac{1}{2}$$

$$\frac{\int_0^T \cos^2(\omega_0 t)dt}{T} = \frac{\left[\frac{t}{2} + \frac{\sin(2\omega_0 t)}{4\omega_0}\right]_0^T}{T} = \frac{\frac{T}{2} + \frac{\sin 4\pi}{4\omega_0} - 0 - 0}{T} = \frac{1}{2}$$

$$\frac{\int_0^T \cos(\omega_0 t)\sin(\omega_0 t)dt}{T} = \left[\frac{\sin^2(\omega_0 t)}{2T\omega_0}\right]_0^T = \frac{\sin^2 2\pi}{2\omega_0 T} - 0 = 0$$

19.2 The angular frequency can be computed as $\omega_0 = 2\pi/360 = 0.017453$. Because the data are equispaced, the coefficients can be determined with Eqs. 19.14-19.16. The various summations required to set up the model can be determined as

t	Radiation	$\cos(\omega_0 t)$	$\sin(\omega_0 t)$	$y\cos(\omega_0 t)$	$y\sin(\omega_0 t)$
15	144	0.96593	0.25882	139.093	37.270
45	188	0.70711	0.70711	132.936	132.936
75	245	0.25882	0.96593	63.411	236.652
105	311	-0.25882	0.96593	-80.493	300.403
135	351	-0.70711	0.70711	-248.194	248.194
165	359	-0.96593	0.25882	-346.767	92.916
195	308	-0.96593	-0.25882	-297.505	-79.716
225	287	-0.70711	-0.70711	-202.940	-202.940
255	260	-0.25882	-0.96593	-67.293	-251.141
285	211	0.25882	-0.96593	54.611	-203.810
315	159	0.70711	-0.70711	112.430	-112.430
345	131	0.96593	-0.25882	126.536	-33.905
sum→	**2954**			**-614.175**	**164.429**

The coefficients can be determined as

$$A_0 = \frac{\Sigma y}{N} = \frac{2954}{12} = 246.1667$$

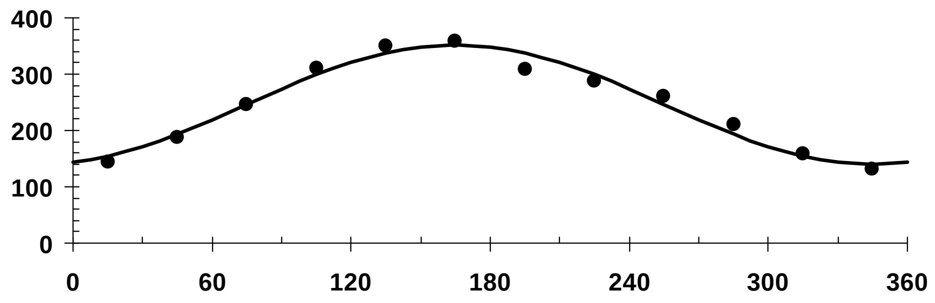
$$A_1 = \frac{2}{N}\Sigma y\cos(\omega_0 t) = \frac{2}{12}(-614.175) = -102.363$$

$$B_1 = \frac{2}{N}\Sigma y\sin(\omega_0 t) = \frac{2}{12}(164.429) = 27.4048$$

Therefore, the best-fit sinusoid is

$$R = 246.1667 - 102.363\cos(0.017453t) + 27.4048\sin(0.017453t)$$

The data and the model can be plotted as



The value for mid-August can be computed as

$$R = 246.1667 - 102.363\cos(0.017453(225)) + 27.4048\sin(0.017453(225)) = 299.1698$$

19.3 The angular frequency can be computed as $\omega_0 = 2\pi/24 = 0.261799$. The various summations required for the normal equations can be set up as

t	y	$\cos(\omega_0 t)$	$\sin(\omega_0 t)$	$\sin(\omega_0 t)\cos(\omega_0 t)$	$\cos^2(\omega_0 t)$	$\sin^2(\omega_0 t)$	$y\cos(\omega_0 t)$	$y\sin(\omega_0 t)$
0	7.3	1.00000	0.00000	0.00000	1.00000	0.00000	7.30000	0.00000
2	7	0.86603	0.50000	0.43301	0.75000	0.25000	6.06218	3.50000
4	7.1	0.50000	0.86603	0.43301	0.25000	0.75000	3.55000	6.14878
5	6.5	0.25882	0.96593	0.25000	0.06699	0.93301	1.68232	6.27852
7	7.4	-0.25882	0.96593	-0.25000	0.06699	0.93301	-1.91526	7.14785
8.5	7.2	-0.60876	0.79335	-0.48296	0.37059	0.62941	-4.38308	5.71214
12	8.9	-1.00000	0.00000	0.00000	1.00000	0.00000	-8.90000	0.00000
15	8.8	-0.70711	-0.70711	0.50000	0.50000	0.50000	-6.22254	-6.22254
20	8.9	0.50000	-0.86603	-0.43301	0.25000	0.75000	4.45000	-7.70763
22	7.9	0.86603	-0.50000	-0.43301	0.75000	0.25000	6.84160	-3.95000
24	7	1.00000	0.00000	0.00000	1.00000	0.00000	7.00000	0.00000
sum→	84	2.41618	2.01810	0.01704	6.00457	4.99543	15.46522	10.90713

The normal equations can be assembled as

$$\begin{bmatrix} 11 & 2.41618 & 2.01810 \\ 2.41618 & 6.00457 & 0.01704 \\ 2.01810 & 0.01704 & 4.99543 \end{bmatrix} \begin{Bmatrix} A_0 \\ A_1 \\ B_1 \end{Bmatrix} = \begin{Bmatrix} 84 \\ 15.46522 \\ 10.90713 \end{Bmatrix}$$

This system can be solved for $A_0 = 7.96304$, $A_1 = -0.62575$, and $B_1 = -1.03142$. Therefore, the best-fit sinusoid is

$$pH = 7.96304 - 0.62575\cos(\omega_0 t) - 1.03142\sin(\omega_0 t)$$

The result can also be expressed in the alternate form of Eq. 19.2 by computing the amplitude,

$$C_1 = \sqrt{0.62575^2 + 1.03142^2} = 1.2064$$

and the phase shift

$$\theta = \arctan\left(-\frac{-1.03142}{-0.62575}\right) = 2.11612$$

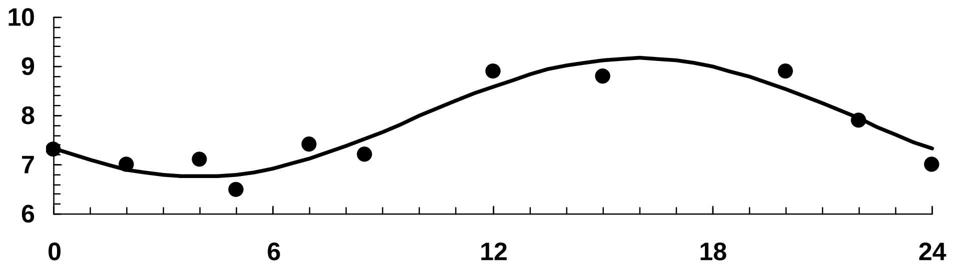
Therefore, the fit can also be expressed as

$$pH = 7.96304 + 1.2064\cos(\omega_0 t + 2.11612)$$

Consequently, the mean is 7.96304 and the amplitude is 1.2064. To determine the time of the maximum, inspection of Fig. 19.3 indicates that a positive phase shift represents the time prior to midnight that the peak occurs. Therefore the time of the maximum can be computed as

$$t_{\max} = 24 - 2.11612 \times \frac{24\text{ hr}}{2\pi} = 15.91702\text{ hrs}$$

This is equal to about 15:55:01 or 3:55:01 PM. The data and the model can be plotted as



19.4 $a_0 = 0$

$$a_k = \frac{2}{T}\int_{-T/2}^{T/2} -2t\cos(k\omega_0 t)\,dt$$
$$= -\frac{4}{T}\left[\frac{1}{(k\omega_0)^2}\cos(k\omega_0 t) + \frac{t}{k\omega_0}\sin(k\omega_0 t)\right]_{-T/2}^{T/2}$$

$$b_k = \frac{2}{T}\int_{-T/2}^{T/2} -2t\sin(k\omega_0 t)\,dt$$
$$= -\frac{4}{T}\left[\frac{1}{(k\omega_0)^2}\sin(k\omega_0 t) - \frac{t}{k\omega_0}\cos(k\omega_0 t)\right]_{-T/2}^{T/2}$$

On the basis of these, all *a*'s = 0. For *k* = odd,

$$b_k = \frac{2}{k\pi}$$

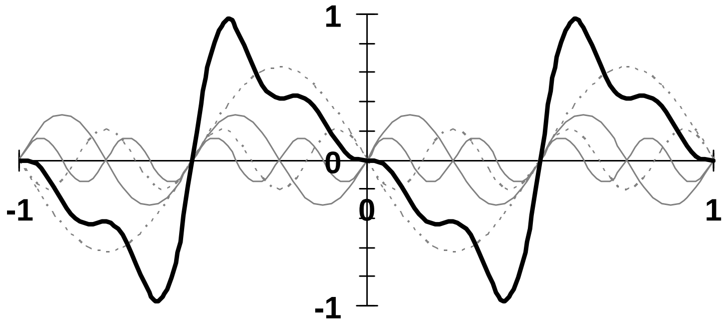
For k = even,

$$b_k = -\frac{2}{k\pi}$$

Therefore, the series is

$$f(t) = -\frac{2}{\pi}\sin(\omega_0 t) + \frac{1}{\pi}\sin(2\omega_0 t) - \frac{2}{3\pi}\sin(3\omega_0 t) + \frac{1}{2\pi}\sin(4\omega_0 t) + \cdots$$

The first 4 terms are plotted below along with the summation:



19.5 $a_0 = 0.5$

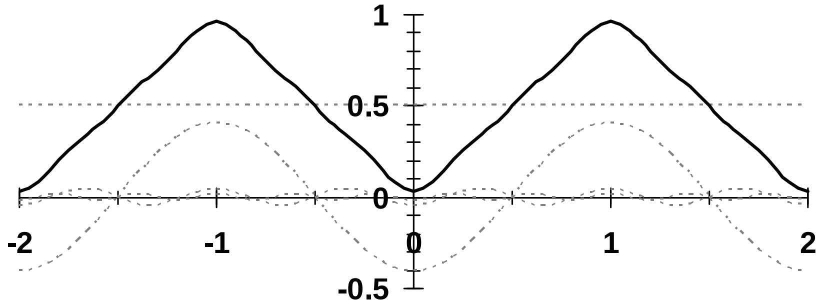
$$\begin{aligned} a_k &= \frac{2}{2}\left[\int_{-1}^{0} -t\cos(k\pi t)\,dt + \int_0^1 t\cos(k\pi t)\,dt\right] \\ &= 1\left\{\left[-\frac{\cos(k\pi t)}{(k\pi)^2} - \frac{t\sin(k\pi t)}{k\pi}\right]_{-1}^{0} + \left[\frac{\cos(k\pi t)}{(k\pi)^2} + \frac{t\sin(k\pi t)}{k\pi}\right]_0^1\right\} \\ &= \frac{2}{(k\pi)^2}(\cos k\pi - 1) \end{aligned}$$

$b_k = 0$

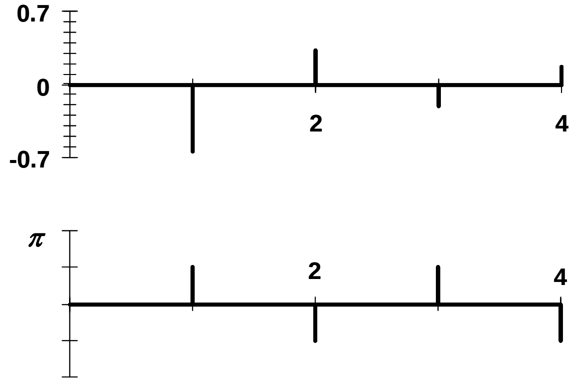
Substituting these coefficients into Eq. (19.17) gives

$$f(t) = \frac{1}{2} - \frac{4}{\pi^2}\cos(\pi t) - \frac{4}{9\pi^2}\cos(3\pi t) - \frac{4}{25\pi^2}\cos(5\pi t) + \cdots$$

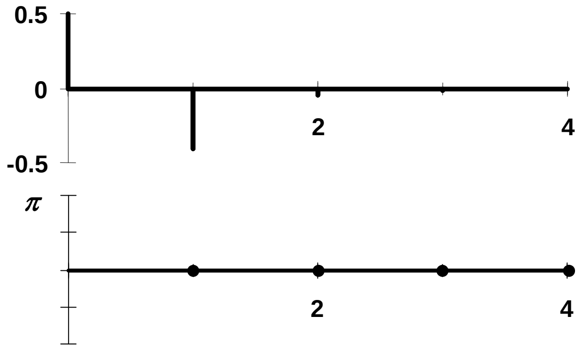
This function for the first 4 terms is displayed below:



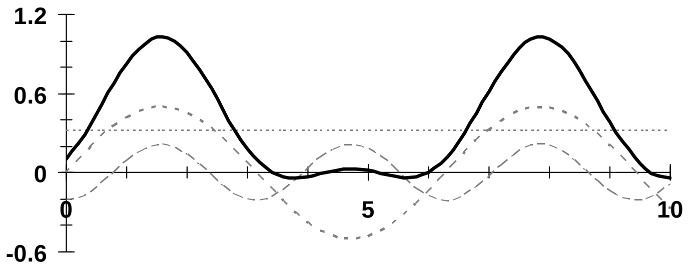
19.6



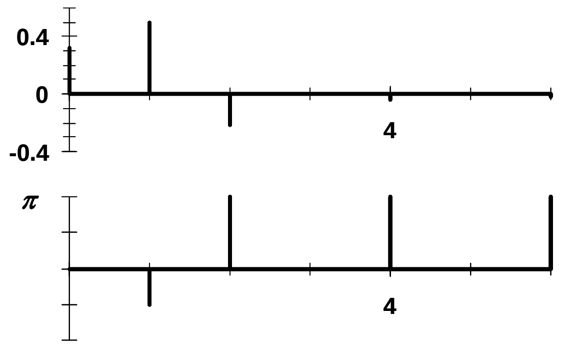
19.7



19.8



19.9



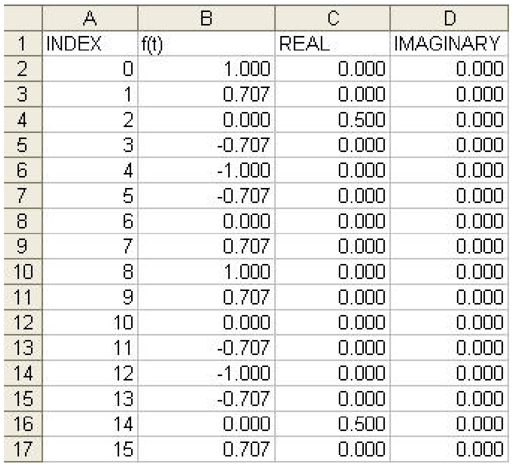
19.10 Here is a VBA code to implement the DFT. It is set up to duplicate Fig. 19.13.

```
Option Explicit

Sub Dfourier()
Dim i As Integer, N As Integer
Dim f(127) As Double, re(127) As Double, im(127) As Double
Dim t As Double, pi As Double, dt As Double, omega As Double
pi = 4# * Atn(1#)
N = 16
t = 0#
dt = 0.01
For i = 0 To N - 1
  f(i) = Cos(2 * 12.5 * pi * t)
  t = t + dt
Next i
omega = 2 * pi / N
Call DFT(f, N, re, im, omega)
Range("A1").Select
ActiveCell.Value = "INDEX"
ActiveCell.Offset(0, 1).Select
ActiveCell.Value = "f(t)"
ActiveCell.Offset(0, 1).Select
ActiveCell.Value = "REAL"
ActiveCell.Offset(0, 1).Select
ActiveCell.Value = "IMAGINARY"
Range("A2").Select
For i = 0 To N - 1
  ActiveCell.Value = i
  ActiveCell.Offset(0, 1).Select
  ActiveCell.Value = f(i)
  ActiveCell.Offset(0, 1).Select
  ActiveCell.Value = re(i)
  ActiveCell.Offset(0, 1).Select
  ActiveCell.Value = im(i)
  ActiveCell.Offset(1, -3).Select
Next i
Range("A1").Select
End Sub

Sub DFT(f, N, re, im, omega)
Dim k As Integer, nn As Integer
Dim angle As Double
For k = 0 To N – 1
  re(k) = 0: im(k) = 0
  For nn = 0 To N - 1
    angle = k * omega * nn
    re(k) = re(k) + f(nn) * Cos(angle) / N
    im(k) = im(k) - f(nn) * Sin(angle) / N
  Next nn
Next k
End Sub
```

When this program is run, the result is



	A	B	C	D
1	INDEX	f(t)	REAL	IMAGINARY
2	0	1.000	0.000	0.000
3	1	0.707	0.000	0.000
4	2	0.000	0.500	0.000
5	3	-0.707	0.000	0.000
6	4	-1.000	0.000	0.000
7	5	-0.707	0.000	0.000
8	6	0.000	0.000	0.000
9	7	0.707	0.000	0.000
10	8	1.000	0.000	0.000
11	9	0.707	0.000	0.000
12	10	0.000	0.000	0.000
13	11	-0.707	0.000	0.000
14	12	-1.000	0.000	0.000
15	13	-0.707	0.000	0.000
16	14	0.000	0.500	0.000
17	15	0.707	0.000	0.000

19.11 The program from Prob. 19.10 can be modified slightly to compute a DFT for the triangular wave from Prob. 19.8. Here is the part that is modified up to the point that the DFT routine is called. The remainder of the program is identical to the one from Prob. 19.10.

```
Option Explicit

Sub Dfourier()
Dim i As Integer, N As Integer
Dim f(127) As Double, re(127) As Double, im(127) As Double, t As Double
Dim pi As Double, Tp As Double, dt As Double, omega As Double
Dim t1 As Double, t2 As Double, Ni As Integer
pi = 4# * Atn(1#)
N = 32
omega = 2 * pi / N
t = 0#
Tp = 2 * pi
dt = 4 * Tp / N
For i = 0 To N - 1
  f(i) = Sin(t)
  If f(i) < 0 Then f(i) = 0
  t = t + dt
Next i
Call DFT(f, N, re, im, omega)
```

The results for the *n* = 32 case are displayed below:

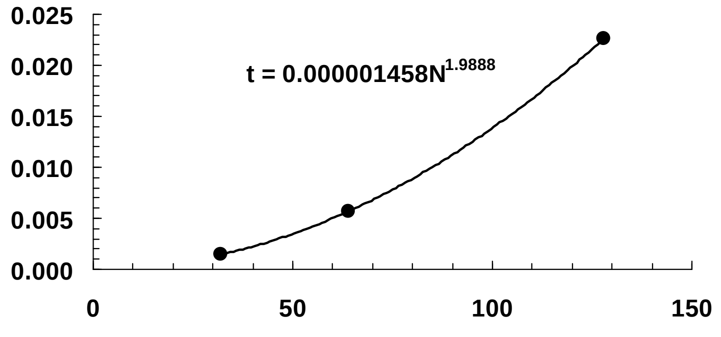
index	*f*(t)	real	imaginary
0	0	0.3018	0
1	0.7071	0	0
2	1	0	0
3	0.7071	0	0
4	0	0	-0.25
5	0	0	0
6	0	0	0

7	0	0	0
8	0	-0.125	0
9	0.7071	0	0
10	1	0	0
11	0.7071	0	0
12	0	0	0
13	0	0	0
14	0	0	0
15	0	0	0
16	0	-0.0518	0
17	0.7071	0	0
18	1	0	0
19	0.7071	0	0
20	0	0	0
21	0	0	0
22	0	0	0
23	0	0	0
24	0	-0.125	0
25	0.7071	0	0
26	1	0	0
27	0.7071	0	0
28	0	0	0.25
29	0	0	0
30	0	0	0
31	0	0	0

The runs for N = 32, 64 and 128 were performed with the following results obtained. (Note that we had to call the function numerous times to obtain measurable times. These times were then divided by the number of function calls to determine the time per call shown below)

N	**time (s)**
32	0.001437
64	0.0057
128	0.02264

A power (log-log) model was fit (see plot below) to this data. Thus, the result verifies that the execution time $\propto N^2$.



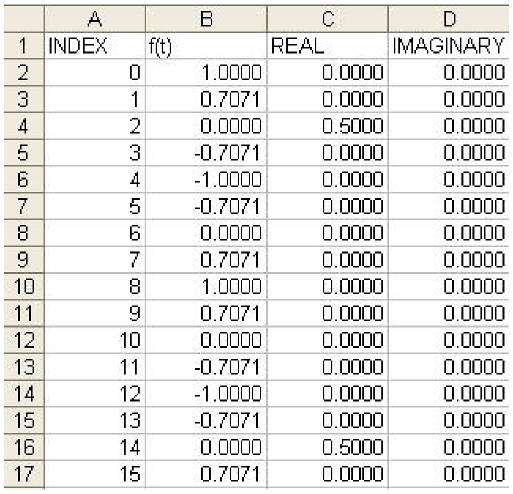
19.12 Here is a VBA code to implement the FFT. It is set up to duplicate Fig. 19.13.

```
Option Explicit

Sub Ffourier()
Dim i As Integer, N As Integer
Dim f(127) As Double, re(127) As Double, im(127) As Double, t As Double
Dim pi As Double, dt As Double, omega As Double
pi = 4# * Atn(1#)
N = 16
t = 0#
dt = 0.01
For i = 0 To N - 1
  re(i) = Cos(2 * 12.5 * pi * t)
  im(i) = 0#
  f(i) = re(i)
  t = t + dt
Next i
Call FFT(N, re, im)
Range("A1").Select
ActiveCell.Value = "INDEX"
ActiveCell.Offset(0, 1).Select
ActiveCell.Value = "f(t)"
ActiveCell.Offset(0, 1).Select
ActiveCell.Value = "REAL"
ActiveCell.Offset(0, 1).Select
ActiveCell.Value = "IMAGINARY"
Range("A2:D1026").ClearContents
Range("A2").Select
For i = 0 To N - 1
  ActiveCell.Value = i
  ActiveCell.Offset(0, 1).Select
  ActiveCell.Value = f(i)
  ActiveCell.Offset(0, 1).Select
  ActiveCell.Value = re(i)
  ActiveCell.Offset(0, 1).Select
  ActiveCell.Value = im(i)
  ActiveCell.Offset(1, -3).Select
Next i
Range("A1").Select
End Sub

Sub FFT(N, x, y)
Dim i As Integer, j As Integer, m As Integer
Dim N2 As Integer, N1 As Integer, k As Integer, l As Integer
Dim pi As Double, xN As Double, angle As Double
Dim arg As Double, c As Double, s As Double
Dim xt As Double, yt As Double
xN = N
m = CInt(Log(xN) / Log(2#))
pi = 4# * Atn(1#)
N2 = N
For k = 1 To m
  N1 = N2
  N2 = N2 / 2
  angle = 0#
  arg = 2 * pi / N1
  For j = 0 To N2 - 1
    c = Cos(angle)
    s = -Sin(angle)
    For i = j To N - 1 Step N1
      l = i + N2
      xt = x(i) - x(l)
      x(i) = x(i) + x(l)
      yt = y(i) - y(l)
      y(i) = y(i) + y(l)
      x(l) = xt * c - yt * s
      y(l) = yt * c + xt * s
    Next i
    angle = (j + 1) * arg
  Next j
Next k
j = 0
For i = 0 To N - 2
  If i < j Then
    xt = x(j)
    x(j) = x(i)
    x(i) = xt
    yt = y(j)
    y(j) = y(i)
    y(i) = yt
  End If
  k = N / 2
  Do
    If k >= j + 1 Then Exit Do
    j = j - k
    k = k / 2
  Loop
  j = j + k
Next i
For i = 0 To N - 1
  x(i) = x(i) / N
  y(i) = y(i) / N
Next i
End Sub
```

When this program is run, the result is



	A	B	C	D
1	INDEX	f(t)	REAL	IMAGINARY
2	0	1.0000	0.0000	0.0000
3	1	0.7071	0.0000	0.0000
4	2	0.0000	0.5000	0.0000
5	3	-0.7071	0.0000	0.0000
6	4	-1.0000	0.0000	0.0000
7	5	-0.7071	0.0000	0.0000
8	6	0.0000	0.0000	0.0000
9	7	0.7071	0.0000	0.0000
10	8	1.0000	0.0000	0.0000
11	9	0.7071	0.0000	0.0000
12	10	0.0000	0.0000	0.0000
13	11	-0.7071	0.0000	0.0000
14	12	-1.0000	0.0000	0.0000
15	13	-0.7071	0.0000	0.0000
16	14	0.0000	0.5000	0.0000
17	15	0.7071	0.0000	0.0000

19.13 The program from Prob. 19.12 can be modified slightly to compute a FFT for the triangular wave from Prob. 19.8. Here is the part that is modified up to the point that the FFT routine is called. The remainder of the program is identical to the one from Prob. 19.12.

```
Option Explicit
Sub Ffourier()
Dim i As Integer, N As Integer
Dim f(127) As Double, re(127) As Double, im(127) As Double, t As Double
Dim pi As Double, dt As Double, Tp As Double, omega As Double
pi = 4# * Atn(1#)
N = 32
```

```
t = 0#
Tp = 2 * pi
dt = 4 * Tp / N
For i = 0 To N - 1
  re(i) = Sin(t)
  If re(i) < 0 Then re(i) = 0
  im(i) = 0
  f(i) = re(i)
  t = t + dt
Next i
Call FFT(N, re, im)
```

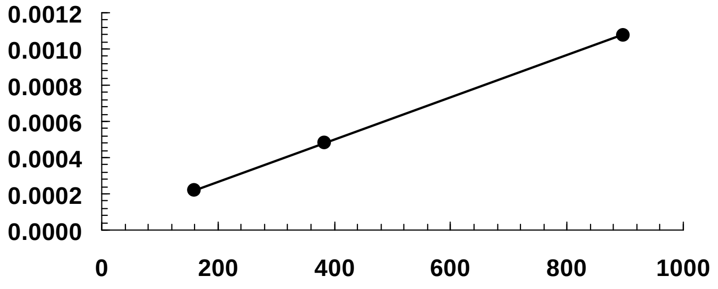
The results for the *n* = 32 case are displayed below:

index	f(t)	real	imaginary
0	0	0.3018	0
1	0.7071	0	0
2	1	0	0
3	0.7071	0	0
4	0	0	-0.25
5	0	0	0
6	0	0	0
7	0	0	0
8	0	-0.125	0
9	0.7071	0	0
10	1	0	0
11	0.7071	0	0
12	0	0	0
13	0	0	0
14	0	0	0
15	0	0	0
16	0	-0.0518	0
17	0.7071	0	0
18	1	0	0
19	0.7071	0	0
20	0	0	0
21	0	0	0
22	0	0	0
23	0	0	0
24	0	-0.125	0
25	0.7071	0	0
26	1	0	0
27	0.7071	0	0
28	0	0	0.25
29	0	0	0
30	0	0	0
31	0	0	0

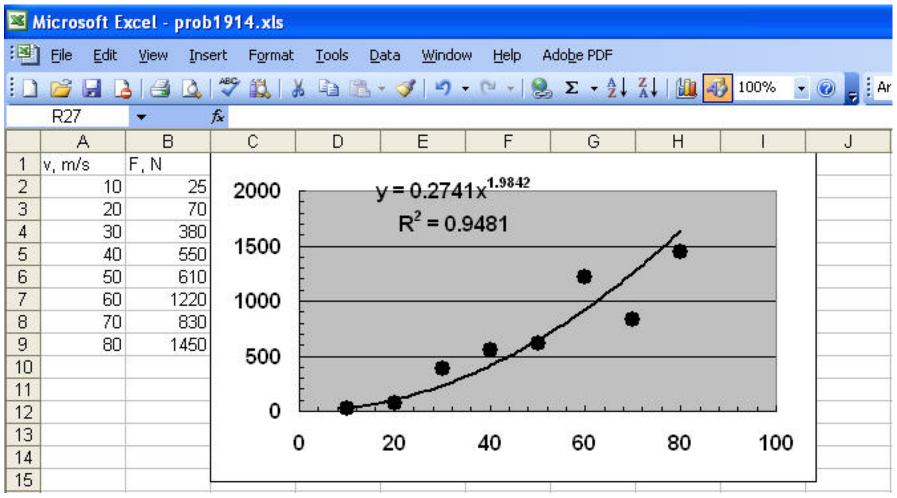
The runs for *N* = 32, 64 and 128 were performed with the following results obtained. (Note that we had to call the function numerous times to obtain measurable times. These times were then divided by the number of function calls to determine the time per call shown below)

N	time (s)
32	0.000219
64	0.000484
128	0.001078

A plot of time versus $N \log_2 N$ yielded a straight line (see plot below). Thus, the result verifies that the execution time $\propto N \log_2 N$.

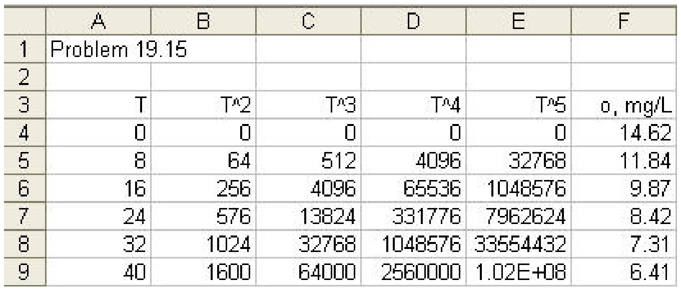


19.14


Microsoft Excel - prob1914.xls
File Edit View Insert Format Tools Data Window Help Adobe PDF
100%
R27
y = 0.2741x^1.9842
R² = 0.9481
2000
1500
1000
500
0
0
20
40
60
80
100

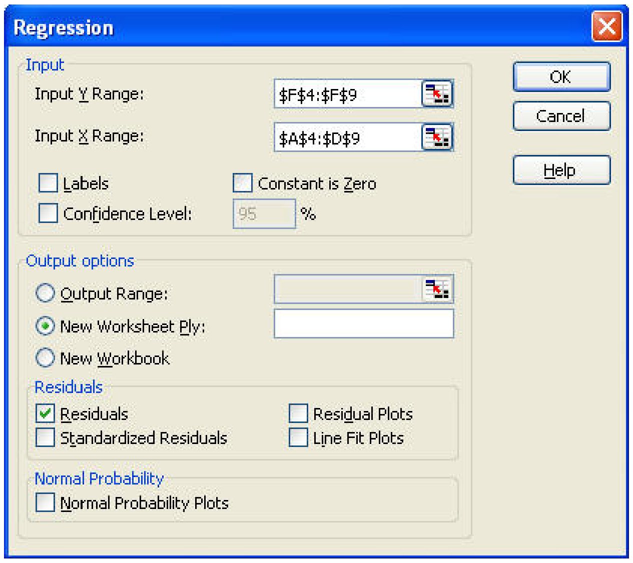
	A	B
1	v, m/s	F, N
2	10	25
3	20	70
4	30	380
5	40	550
6	50	610
7	60	1220
8	70	830
9	80	1450

19.15 An Excel worksheet can be developed with columns holding the dependent variable (o) along with the independent variable (T). In addition, columns can be set up holding progressively higher powers of the independent variable.



	A	B	C	D	E	F
1	Problem 19.15					
2						
3	T	T^2	T^3	T^4	T^5	o, mg/L
4	0	0	0	0	0	14.62
5	8	64	512	4096	32768	11.84
6	16	256	4096	65536	1048576	9.87
7	24	576	13824	331776	7962624	8.42
8	32	1024	32768	1048576	33554432	7.31
9	40	1600	64000	2560000	1.02E+08	6.41

The Data Analysis Toolpack can then be used to develop a regression polynomial as described in Example 19.4. For example, a fourth-order polynomial can be developed as



Input

Input Y Range: $F$4:$F$9

Input X Range: $A$4:$D$9

☐ Labels ☐ Constant is Zero

☐ Confidence Level: 95 %

Output options

○ Output Range:

● New Worksheet Ply:

○ New Workbook

Residuals

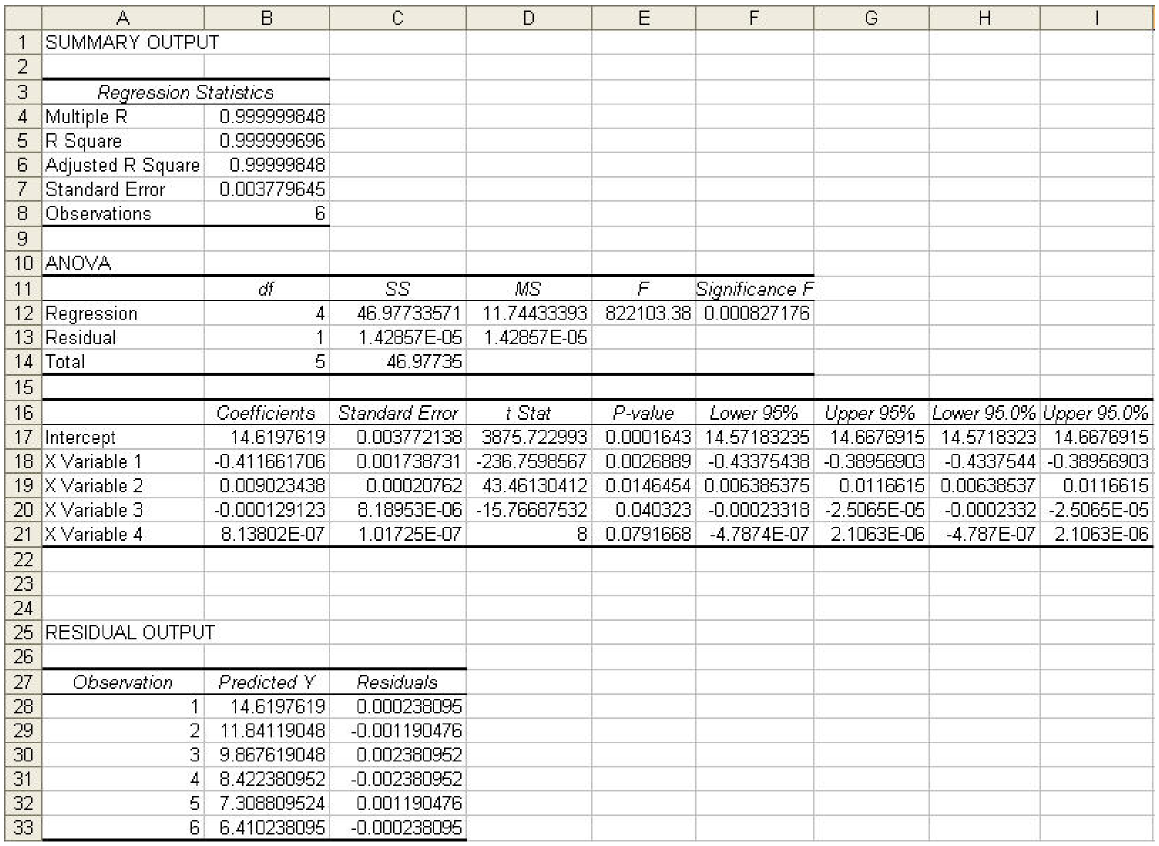
☑ Residuals ☐ Residual Plots

☐ Standardized Residuals ☐ Line Fit Plots

Normal Probability

☐ Normal Probability Plots

Notice that we have checked off the Residuals box. Therefore, when the regression is implemented, the model predictions (the column labeled Predicted Y) are listed along with the fit statistics as shown below:

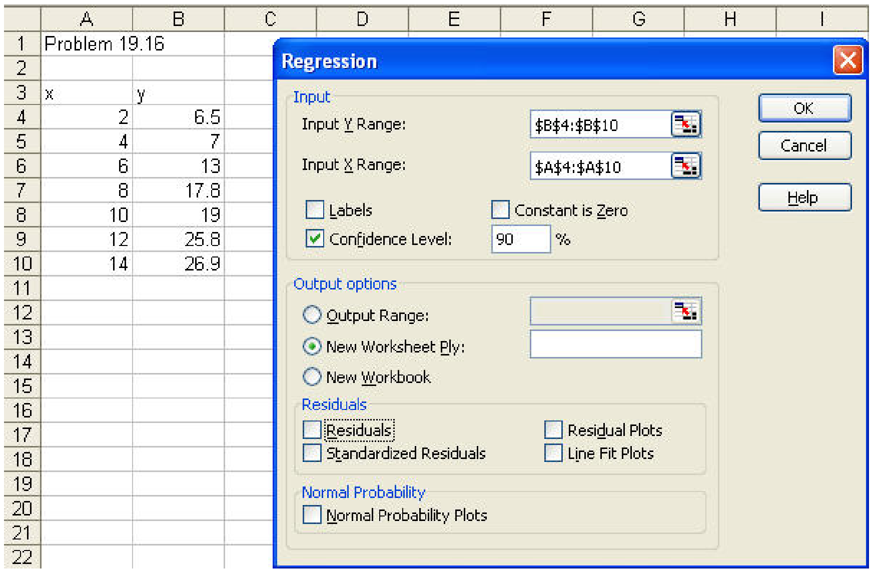


	A	B	C	D	E	F	G	H	I
1	SUMMARY OUTPUT								
2									
3	*Regression Statistics*								
4	Multiple R	0.999999848							
5	R Square	0.999999696							
6	Adjusted R Square	0.99999848							
7	Standard Error	0.003779645							
8	Observations	6							
9									
10	ANOVA								
11		*df*	*SS*	*MS*	*F*	*Significance F*			
12	Regression	4	46.97733571	11.74433393	822103.38	0.000827176			
13	Residual	1	1.42857E-05	1.42857E-05					
14	Total	5	46.97735						
15									
16		*Coefficients*	*Standard Error*	*t Stat*	*P-value*	*Lower 95%*	*Upper 95%*	*Lower 95.0%*	*Upper 95.0%*
17	Intercept	14.6197619	0.003772138	3875.722993	0.0001643	14.57183235	14.6676915	14.5718323	14.6676915
18	X Variable 1	-0.411661706	0.001738731	-236.7598567	0.0026889	-0.43375438	-0.38956903	-0.4337544	-0.38956903
19	X Variable 2	0.009023438	0.00020762	43.46130412	0.0146454	0.006385375	0.0116615	0.00638537	0.0116615
20	X Variable 3	-0.000129123	8.18953E-06	-15.76687532	0.040323	-0.00023318	-2.5065E-05	-0.0002332	-2.5065E-05
21	X Variable 4	8.13802E-07	1.01725E-07	8	0.0791668	-4.7874E-07	2.1063E-06	-4.787E-07	2.1063E-06
22									
23									
24									
25	RESIDUAL OUTPUT								
26									
27	*Observation*	*Predicted Y*	*Residuals*						
28	1	14.6197619	0.000238095						
29	2	11.84119048	-0.001190476						
30	3	9.867619048	0.002380952						
31	4	8.422380952	-0.002380952						
32	5	7.308809524	0.001190476						
33	6	6.410238095	-0.000238095						

As can be seen, the results match to the level of precision (second decimal place) of the original data. If a third-order polynomial was used, this would not be the case. Therefore, we conclude that the best-fit equation is

$$o = 14.61976 - 0.4116617T + 9.023438\times10^{-3}T^2 - 1.29123\times10^{-4}T^3 + 8.13802\times10^{-7}T^4$$

19.16 An Excel worksheet can be developed with columns holding the dependent variable (o) along with the independent variable (T). The Data Analysis Toolpack can then be used to develop a linear regression.



	A	B
1	Problem 19.16	
2		
3	x	y
4	2	6.5
5	4	7
6	6	13
7	8	17.8
8	10	19
9	12	25.8
10	14	26.9

Regression

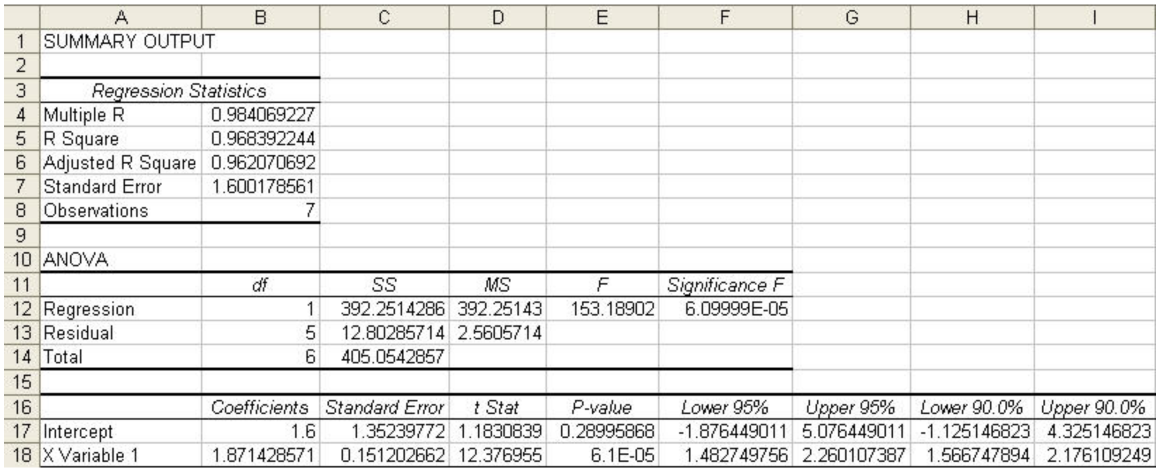
Input
Input Y Range: $B$4:$B$10
Input X Range: $A$4:$A$10
☐ Labels ☐ Constant is Zero
☑ Confidence Level: 90 %

Output options
○ Output Range:
● New Worksheet Ply:
○ New Workbook

Residuals
☐ Residuals ☐ Residual Plots
☐ Standardized Residuals ☐ Line Fit Plots

Normal Probability
☐ Normal Probability Plots

Notice that we have checked off the Confidence Level box and entered 90%. Therefore, when the regression is implemented, the 90% confidence intervals for the coefficients will be computed and displayed as shown below:



SUMMARY OUTPUT

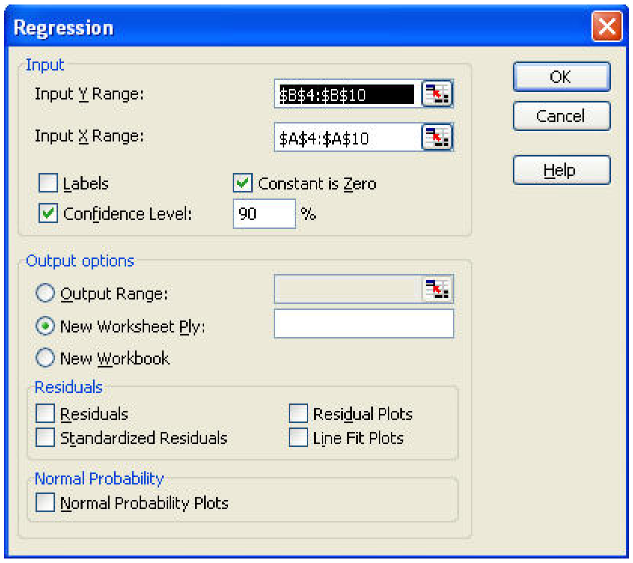
Regression Statistics	
Multiple R	0.984069227
R Square	0.968392244
Adjusted R Square	0.962070692
Standard Error	1.600178561
Observations	7

ANOVA

	df	*SS*	*MS*	*F*	*Significance F*
Regression	1	392.2514286	392.25143	153.18902	6.09999E-05
Residual	5	12.80285714	2.5605714		
Total	6	405.0542857			

	Coefficients	*Standard Error*	*t Stat*	*P-value*	*Lower 95%*	*Upper 95%*	*Lower 90.0%*	*Upper 90.0%*
Intercept	1.6	1.35239772	1.1830839	0.28995868	-1.876449011	5.076449011	-1.125146823	4.325146823
X Variable 1	1.871428571	0.151202662	12.376955	6.1E-05	1.482749756	2.260107387	1.566747894	2.176109249

As can be seen, the 90% confidence interval for the intercept (–1.125, 4.325) encompasses zero. Therefore, we can redo the regression, but forcing a zero intercept. As shown below, this is accomplished by checking the Constant is Zero box.


Regression

Input

Input Y Range: $B$4:$B$10

Input X Range: $A$4:$A$10

[ ] Labels [✓] Constant is Zero

[✓] Confidence Level: 90 %

Output options

( ) Output Range:

(●) New Worksheet Ply:

( ) New Workbook

Residuals

[ ] Residuals [ ] Residual Plots

[ ] Standardized Residuals [ ] Line Fit Plots

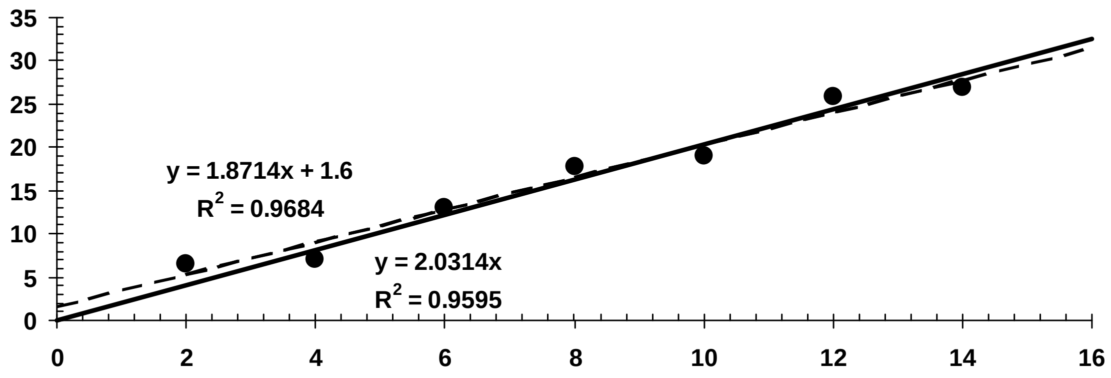
Normal Probability

[ ] Normal Probability Plots

When the regression is implemented, the best-fit equation is

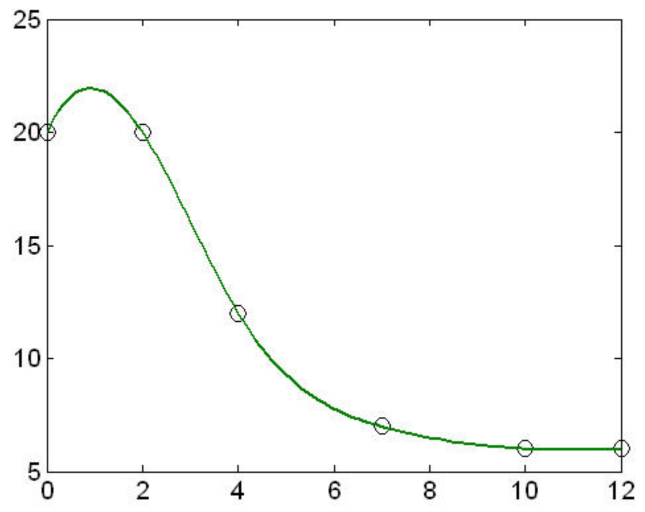
$$y = 2.031429x$$

This equation, along with the original linear regression, is displayed together with the data on the following plot:



19.17 (a) Here is a MATLAB session to develop the spline and plot it along with the data:

```
>> x=[0 2 4 7 10 12];
>> y=[20 20 12 7 6 6];
>> xx=linspace(0,12);
>> yy=spline(x,y,xx);
>> plot(x,y,'o',xx,yy)
```

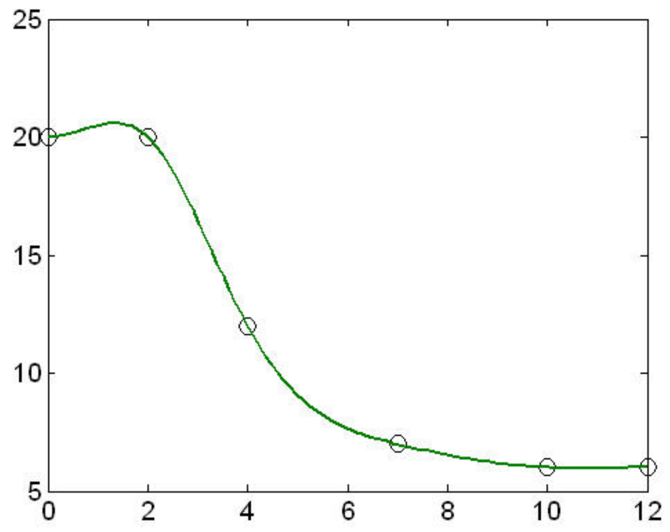


Here is the command to make the prediction:

```
>> yp=spline(x,y,1.5)
yp =
   21.3344
```

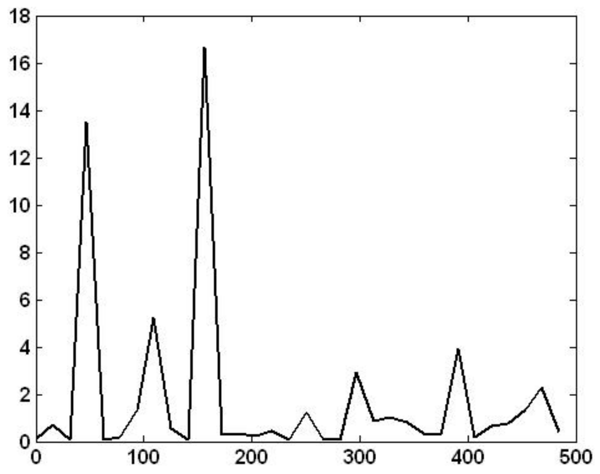
(b) To prescribe zero first derivatives at the end knots, the *y* vector is modified so that the first and last elements are set to the desired values of zero. The plot and the prediction both indicate that there is less overshoot between the first two points because of the prescribed zero slopes.

```
>> yd=[0 y 0];
>> yy=spline(x,yd,xx);
>> plot(x,y,'o',xx,yy)
>> yy=spline(x,yd,1.5)
yy =
   20.5701
```



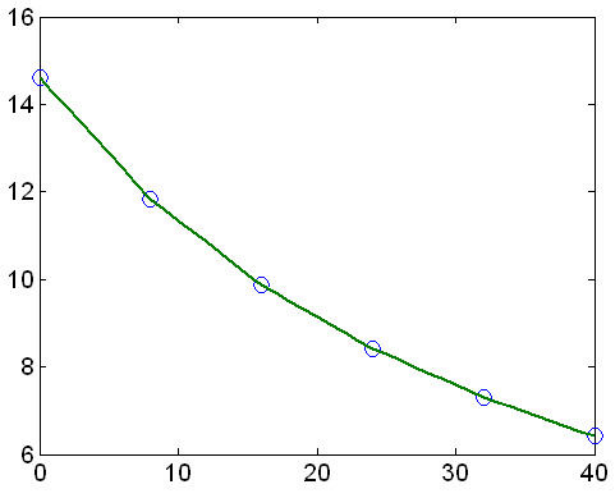
19.18 The following MATLAB session develops the fft along with a plot of the power spectral density versus frequency.

```
>> t=0:63;
>> y=cos(10*2*pi*t/63)+sin(3*2*pi*t/63)+randn(size(t));
>> Y=fft(y,64);
>> Pyy=Y.*conj(Y)/64;
>> f=1000*(0:31)/64;
>> plot(f,Pyy(1:32))
```



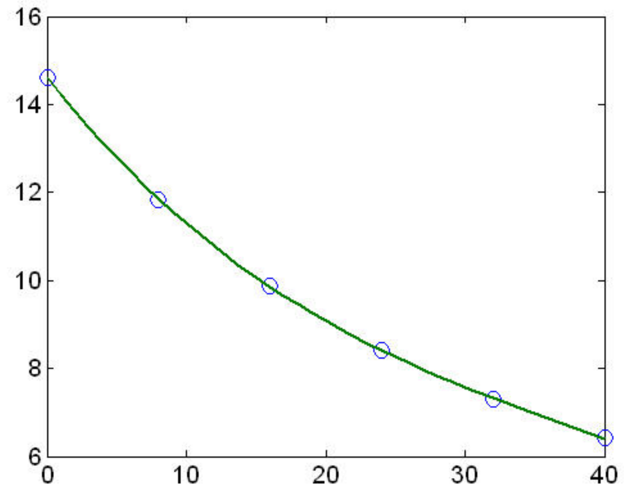
19.19 (a) Linear interpolation

```
>> T=[0 8 16 24 32 40];
>> o=[14.62 11.84 9.87 8.42 7.31 6.41];
>> Ti=0:1:40;
>> oi=interp1(T,o,Ti);
>> plot(T,o,'o',Ti,oi)
>> ol=interp1(T,o,10)
ol =
   11.3475
```



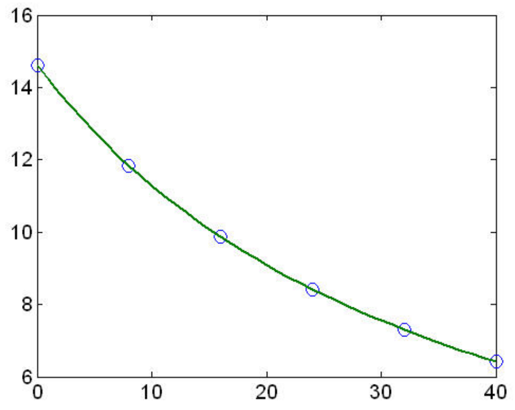
(b) Third-order regression polynomial

```
>> p=polyfit(T,o,3)
p =
   -0.0001    0.0074   -0.3998   14.6140
>> oi=polyval(p,Ti);
>> plot(T,o,'o',Ti,oi)
>> op=polyval(p,10)
op =
   11.2948
```



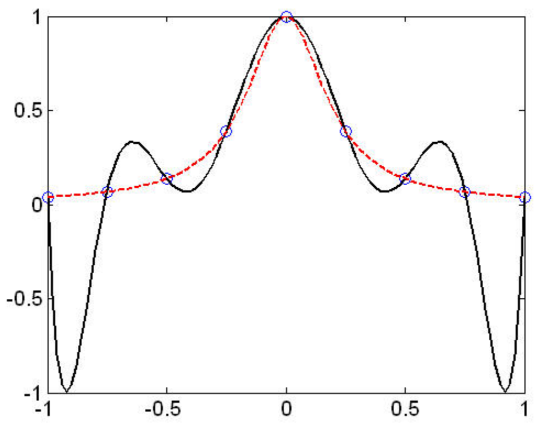
(c) Cubic spline

```
>> oi=spline(T,o,Ti);
>> plot(T,o,'o',Ti,oi)
>> os=spline(T,o,10)
os =
   11.2839
```



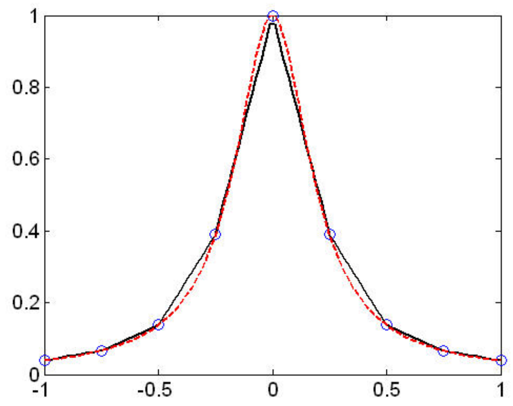
19.20 (a) Eighth-order polynomial:

```
>> x=linspace(-1,1,9);
>> y=1./(1+25*x.^2);
>> p=polyfit(x,y,8);
>> xx=linspace(-1,1);
>> yy=polyval(p,xx);
>> yr=1./(1+25*xx.^2);
>> plot(x,y,'o',xx,yy,xx,yr,'--')
```



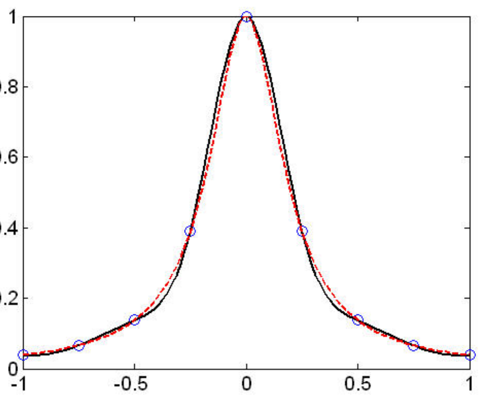
(b) linear spline:

```
>> x=linspace(-1,1,9);
>> y=1./(1+25*x.^2);
>> xx=linspace(-1,1);
>> yy=interp1(x,y,xx);
>> yr=1./(1+25*xx.^2);
>> plot(x,y,'o',xx,yy,xx,yr,'--')
```

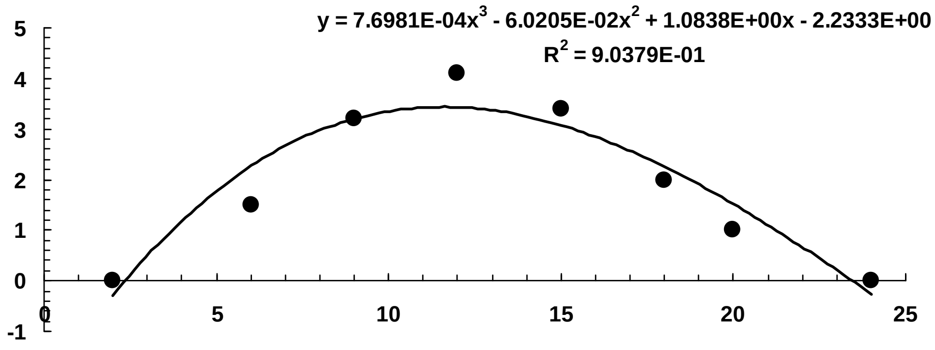


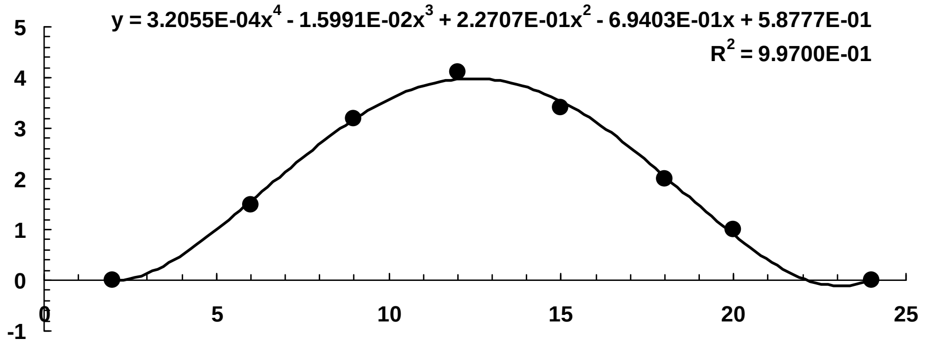
(b) cubic spline:

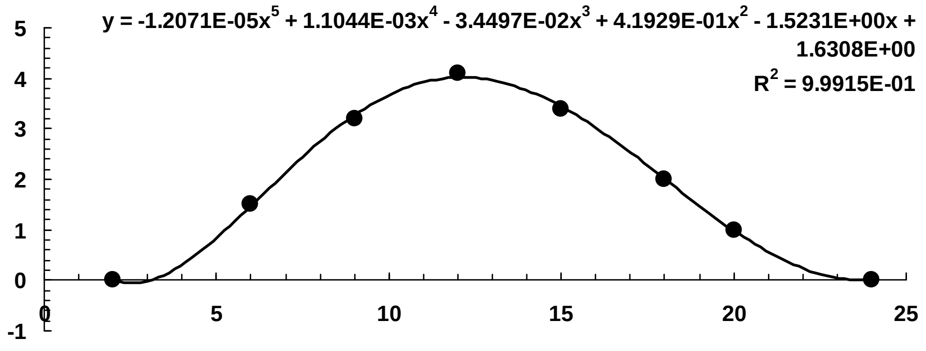
```
>> x=linspace(-1,1,9);
>> y=1./(1+25*x.^2);
>> xx=linspace(-1,1);
>> yy=spline(x,y,xx);
>> yr=1./(1+25*xx.^2);
>> plot(x,y,'o',xx,yy,xx,yr,'--')
```



19.21 Using Excel, plot the data and use the trend line function to fit a polynomial of specific order. Obtain the r^2 value to determine the goodness of fit.







Use the 5th order polynomial:

$$C = -1.2071\times10^{-5}t^5 + 1.1044\times10^{-3}t^4 - 3.4497\times10^{-2}t^3 + 0.41929t^2 - 1.5231t + 1.6308$$

Integrate to find the area under the curve:

$$\int_2^{24} -1.2071\times10^{-5}t^5 + 1.1044\times10^{-3}t^4 - 3.4497\times10^{-2}t^3 + 0.41929t^2 - 1.5231t + 1.6308\,dt = 44.37$$

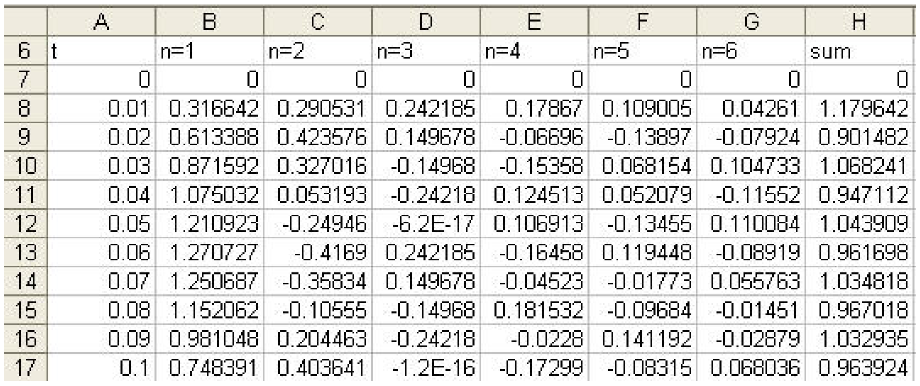
Area under curve: 44.37 mg sec/L

$$\text{Cardiac output} = \frac{5\text{ mg}}{44.37\text{ mg sec/L}} = 0.11269\text{ L/sec} = 6.76\text{ L/min}$$

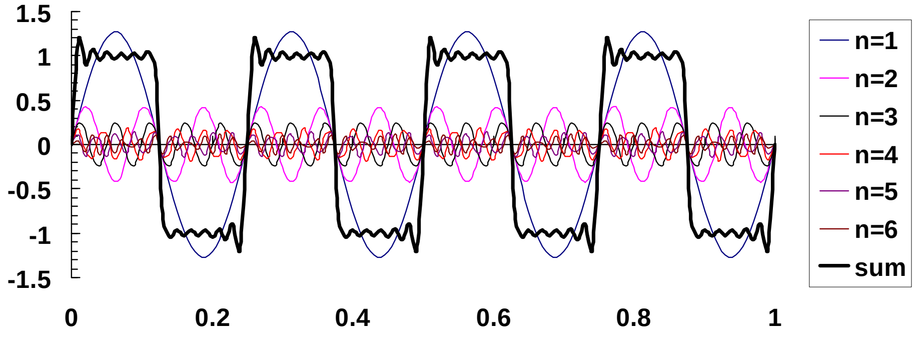
19.22 Plug in A_0 = 1 and T = 0.25,

$$f(t) = \sum_{n=1}^{\infty}\left(\frac{4}{(2n-1)\pi}\right)\sin\bigl(8\pi(2n-1)t\bigr)$$

Make a table and plot in Excel. The following shows the first several entries from the table along with the plot.



	A	B	C	D	E	F	G	H
6	t	n=1	n=2	n=3	n=4	n=5	n=6	sum
7	0	0	0	0	0	0	0	0
8	0.01	0.316642	0.290531	0.242185	0.17867	0.109005	0.04261	1.179642
9	0.02	0.613388	0.423576	0.149678	-0.06696	-0.13897	-0.07924	0.901482
10	0.03	0.871592	0.327016	-0.14968	-0.15358	0.068154	0.104733	1.068241
11	0.04	1.075032	0.053193	-0.24218	0.124513	0.052079	-0.11552	0.947112
12	0.05	1.210923	-0.24946	-6.2E-17	0.106913	-0.13455	0.110084	1.043909
13	0.06	1.270727	-0.4169	0.242185	-0.16458	0.119448	-0.08919	0.961698
14	0.07	1.250687	-0.35834	0.149678	-0.04523	-0.01773	0.055763	1.034818
15	0.08	1.152062	-0.10555	-0.14968	0.181532	-0.09684	-0.01451	0.967018
16	0.09	0.981048	0.204463	-0.24218	-0.0228	0.141192	-0.02879	1.032935
17	0.1	0.748391	0.403641	-1.2E-16	-0.17299	-0.08315	0.068036	0.963924



19.23 This problem is convenient to solve with MATLAB

(a) When we first try to fit a sixth-order interpolating polynomial, MATLAB displays the following error message

```
>> x=[0 100 200 400 600 800 1000];
>> y=[0 0.82436 1 .73576 .40601 .19915 .09158];
>> p=polyfit(x,y,6);
Warning: Polynomial is badly conditioned. Remove repeated data points
         or try centering and scaling as described in HELP POLYFIT.
> In polyfit at 79
```

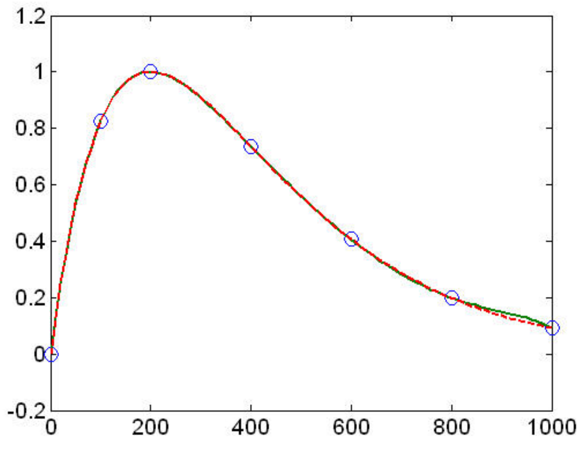
Therefore, we redo the calculation but centering and scaling the *x* values as shown,

```
>> xs=(x-500)/100;
>> p=polyfit(xs,y,6);
```

Now, there is no error message so we can proceed.

```
>> xx=linspace(0,1000);
>> yy=polyval(p,(xx-500)/100);
>> yc=xx/200.*exp(-xx/200+1);
>> plot(x,y,'o',xx,yy,xx,yc,'--')
```

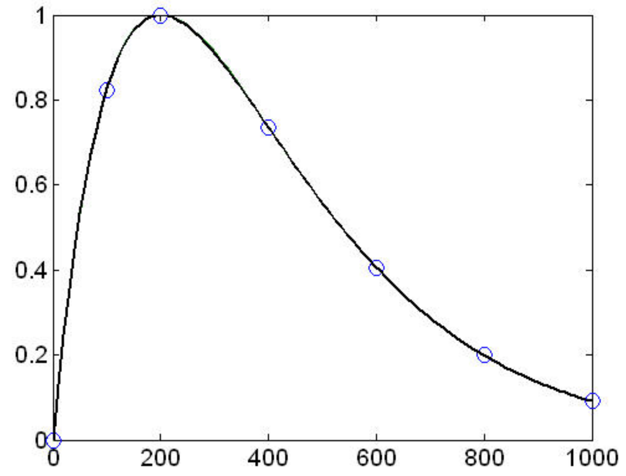
This results in a decent plot. Note that the interpolating polynomial (solid) and the original function (dashed) almost plot on top of each other:



(b) Cubic spline:

```
>> x=[0 100 200 400 600 800 1000];
>> y=[0 0.82436 1 .73576 .40601 .19915 .09158];
>> xx=linspace(0,1000);
>> yc=xx/200.*exp(-xx/200+1);
>> yy=spline(x,y,xx);
>> plot(x,y,'o',xx,yy,xx,yc,'--')
```

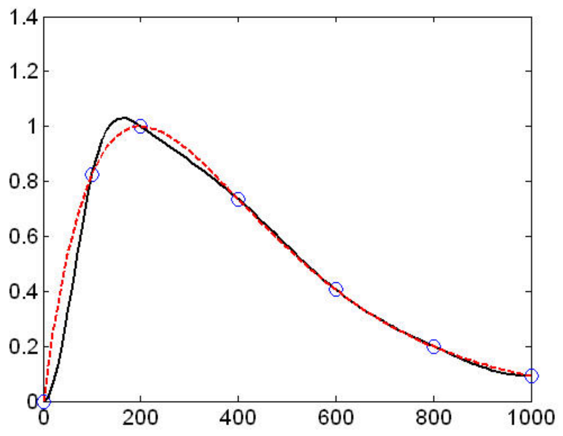
For this case, the fit is so good that the spline and the original function are indistinguishable.



(c) Cubic spline with clamped end conditions (zero slope):

```
>> x=[0 100 200 400 600 800 1000];
>> y=[0 0.82436 1 .73576 .40601 .19915 .09158];
>> ys=[0 y 0];
>> xx=linspace(0,1000);
>> yc=xx/200.*exp(-xx/200+1);
>> yy=spline(x,ys,xx);
>> plot(x,y,'o',xx,yy,xx,yc,'--')
```

For this case, the spline differs from the original function because the latter does not have zero end derivatives.



1.4
1.2
1
0.8
0.6
0.4
0.2
0
0
200
400
600
800
1000