7 Fitting Functions to Data

In this chapter we consider a variety of methods for fitting functions to data, describe some of the MATLAB functions that are available for this purpose, and develop some additional ones. The application of these functions is illustrated by appropriate examples.

7.1 Introduction

We fit functions to two general classes of data: data that is exact and data that we know contains errors. When we fit a function to exact data, we fit to the points exactly. When we fit a function to data that is known to contain errors, we try to obtain the best fit to the trend of the data, using some criterion. The user must exercise skill in making a sensible choice of the function to fit.

We begin by examining polynomial interpolation, which is an example of fitting to exact data.

7.2 Interpolation Using Polynomials

Suppose y is some unknown function of x. Given a table of values of x and y, we may wish to obtain a value of y corresponding to a value of x that is not tabulated. Interpolation implies that the untabulated value of x is within the range of the tabulated data. If the untabulated x is outside this range, the process is called *extrapolation* and is often less accurate.

The simplest form of interpolation is linear interpolation. In this method only the pair of data points enclosing the required value are used. Thus if (x_0, y_0) and (x_1, y_1) are two adjacent data points in a tabulation, to obtain the value of y corresponding to an x where $x_0 < x < x_1$, we fit the straight line $y = ax + b$ to these points and evaluate y as follows:

$$y = [y_0(x_1 - x) + y_1(x - x_0)]/(x_1 - x_0) \tag{7.1}$$

We may use the MATLAB function `interp1` for this purpose. For example, consider the function $y = x^{1.9}$, tabulated at $x = 1, 2, \ldots, 5$. If we require estimates of y for $x = 2.5$ and 3.8,

Numerical Methods Using MATLAB®. DOI: 10.1016/B978-0-12-386942-5.00007-2

we may use `interp1` setting the third parameter as `'linear'` to obtain linear interpolation as follows:

```
>> x = 1:5;
>> y = x.^1.9;
>> interp1(x,y,[2.5,3.8],'linear')

ans =
    5.8979   12.7558
```

The exact answers are $y = 5.7028$ and $y = 12.6354$ corresponding to $x = 2.5$ and 3.8, respectively. For some applications this may be sufficiently accurate.

Interpolation becomes more accurate when more of the tabulated data values are used because we can use a higher-degree polynomial. A polynomial of degree n can be adjusted to pass through $n+1$ data points. We do not need to know the coefficients of the polynomial explicitly, but they are used implicitly in the procedure to estimate y for a given value of x. For example, MATLAB allows cubic interpolation by calling `interp1` with the third parameter set as `'cubic'`. The following example implements cubic interpolation using the same data as the previous example.

```
>> interp1(x,y,[2.5 3.8],'cubic')

ans =
    5.6938   12.6430
```

The cubic interpolation gives a much more accurate result.

An algorithm that provides an efficient method for fitting any degree polynomial to data is Aitken's algorithm. In this procedure a sequence of polynomial functions are fitted to the data. As the degree of the polynomial is increased, more of the data points are used and the accuracy of the interpolation improves.

Aitken's algorithm proceeds as follows. Suppose we have five pairs of data values labeled 1, 2, ..., 5 and we wish to determine y^*, the value of y corresponding to a given x^*. Initially the algorithm determines straight lines (i.e., first-degree polynomials) that pass through data points 1 and 2, 1 and 3, 1 and 4, and 1 and 5, as shown in Figure 7.1(a). These four straight lines allow the procedure to determine four, probably poor, estimates for y^*.

Using x_2, x_3,...,x_5 from the tabulated data and the four estimates of y^* determined from the first-degree polynomial, the algorithm repeats the preceding procedure using these new points, but this now provides second-degree polynomials through the sets of data points {1,2,3}, {1,2,4}, and {1,2,5}, as shown in Figure 7.1(b). From these second-degree polynomials, the procedure determines three improved estimates for y^*.

Using x_3, x_4, x_5 from the tabulated data and the three new estimates for y^* obtained from the second-degree polynomials, the algorithm computes the third-degree polynomials passing through the sets of data points {1, 2, 3, 4} and {1, 2, 3, 5}, as shown in

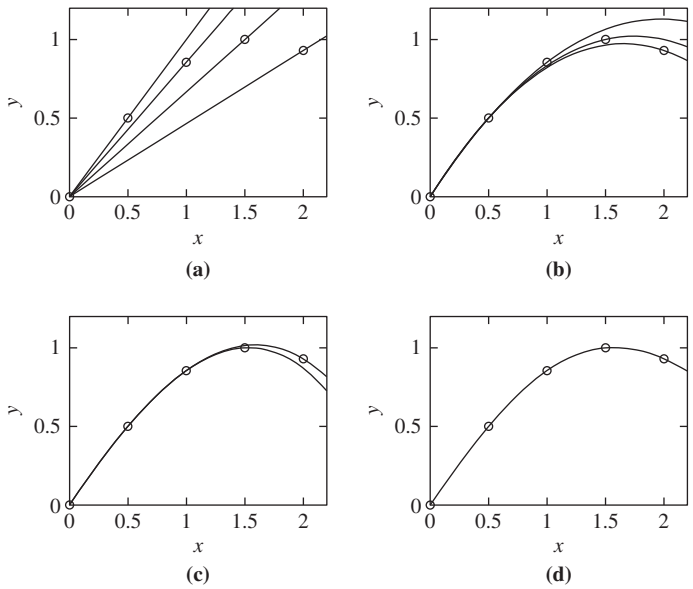

FIGURE 7.1 Increasing the degree of the polynomial fit: (a) 1st degree, (b) 2nd degree, (c) 3rd degree, and (d) 4th degree.

Figure 7.1(c) to allow the procedure to determine two further improved estimates for y^*. Finally a fourth-degree polynomial is computed that fits all the data. This fourth-degree polynomial provides the best estimate for y^*, as shown in Figure 7.1(d).

Aitken's algorithm has two advantages. First of all it is very efficient. Each new estimate for y^* requires only two multiplications and one division so that for $n+1$ data points, the estimate using all the data requires $n(n+1)$ multiplications and $n(n+1)/2$ divisions. It is interesting to note that if we attempted to determine the coefficients of a polynomial passing through $n+1$ data points by assembling a set of $n+1$ linear equations, then in addition to the computation required to assemble the equations, we would require $(n+1)^3/2$ multiplications and divisions to solve them. The second advantage of Aitken's algorithm is that the process of fitting higher- and higher-degree polynomials to more and more of the data can be terminated when we note that the estimate of y^* is no longer changing significantly.

The following MATLAB function `aitken` implements Aitken's algorithm. The user must provide a set of data for the vectors `x` and `y`. The function then determines a value of `y` corresponding to `xval`. The function provides the best value obtained and, if required, a table showing all the intermediate values obtained.

```
function [Q R] = aitken(x,y,xval)
% Aitken's method for interpolation.
% Example call: [Q R] = aitken(x,y,xval)
% x and y give the table of values. Parameter xval is
% the value of x at which interpolation is required.
```

```
% Q is interpolated value, R gives table of intermediate results.
n = length(x); P = zeros(n);
P(1,:) = y;
for j = 1:n-1
    for i = j+1:n
        P(j+1,i) = (P(j,i)*(xval-x(j))-P(j,j)*(xval-x(i)))/(x(i)-x(j));
    end
end
Q = P(n,n); R = [x.' P.'];
```

We now use this function to determine the reciprocal of 1.03 from a table of 10 equispaced values of x in the range 1 to 2 and $y = 1/x$. The following script calls the function `aitken` to solve this example:

```
% e3s701.m
x = 1:.2:2; y = 1./x;
[interpval table] = aitken(x,y,1.03);
fprintf('Interpolated value= %10.8f\n\n',interpval)
disp('Table = ')
disp(table)
```

Running this script gives the following output:

```
interpolated value= 0.97095439

Table =
  1.0000   1.0000        0        0        0        0        0
  1.2000   0.8333   0.9750        0        0        0        0
  1.4000   0.7143   0.9786   0.9720        0        0        0
  1.6000   0.6250   0.9813   0.9723   0.9713        0        0
  1.8000   0.5556   0.9833   0.9726   0.9713   0.9710        0
  2.0000   0.5000   0.9850   0.9729   0.9714   0.9711   0.9710
```

Notice that the first column in this table contains the tabulated x values, the second column contains the tabulated y values, and the remaining columns give successively higher-degree polynomial interpolants generated by Aitken's method. The zeros in this table are padding: The number of estimates in each column decreases as the estimates use more of the data. The exact value is $y = 0.970873786$; thus the Aitken interpolated value of $y = 0.97095439$ is correct to four decimal places. Linear interpolation gives 0.9750, a much poorer result with an error of approximately 0.2%.

Aitken's method provides an interpolated value of y for a given value of x by fitting a polynomial to the data, but the coefficients of the polynomial are not determined explicitly. Conversely, we can fit a polynomial explicitly to the data, determine its coefficients,

and then determine the required interpolated value by evaluating the polynomial. This approach may be less computationally efficient. The MATLAB function `polyfit(x,y,n)` fits a polynomial of degree n through the data given by x and y and returns the coefficients of descending powers of x. For an exact fit, n must equal $m-1$ where m is the number of data points. The polynomial represented by `p` can then be evaluated using the `polyval` function. For example, to determine the reciprocal of 1.03 from a table of six equispaced values of x in the range 1 to 2 and $y = 1/x$, we have

```
% e3s702.m
x = 1:.2:2; y = 1./x;
p = polyfit(x,y,5)
interpval = polyval(p,1.03);
fprintf('interpolated value = %10.8f\n',interpval)
```

Running this script gives

```
p =
   -0.1033    0.9301   -3.4516    6.7584   -7.3618    4.2282

interpolated value = 0.97095439
```

Thus

$$y = -0.1033x^5 + 0.9301x^4 - 3.4516x^3 + 6.7584x^2 - 7.3618x + 4.2282$$

The interpolated value is identical to that given by Aitken's method, as indeed it must be (except for possible rounding errors in the computation) because there is only one polynomial that passes through all six data points and both methods have used it. We use the MATLAB function `polyfit` again in Section 7.8.

7.3 Interpolation Using Splines

The spline is used to connect data points to each other using a curve that appears to the eye to be smooth, either for the purpose of visualization in design drawings or for interpolation. It has certain advantages over the use of a high-degree polynomial, which has a tendency to oscillate between data values.

We begin with an historical example of ship design. Ships' hulls have always curved in a complex manner in two dimensions. Figure 7.2 shows hull sections for a 74-gun British warship, circa 1813. The data points are taken from the original plans, and splines have

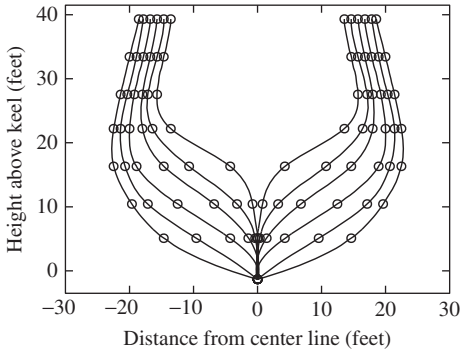

FIGURE 7.2 Use of splines to define cross-sections of a ship's hull.

been used to join the data points together smoothly. Each line shows a section of the ship; the innermost line is close to the stern, and the outermost is near amidships. The graph gives a clear impression of the way the ship builder chose to reduce the ship's cross-section toward the stern.

Polynomials of varying degrees are used for splines, but here we only consider the cubic spline. The cubic spline is a series of cubic polynomials joining data points or "knots." Suppose we have n data points joined by $n-1$ polynomials. Each cubic polynomial has four unknown coefficients so that there are $4(n-1)$ coefficients to be determined. Obviously each polynomial must pass through the two data points it joins. This provides $2(n-1)$ equations that must be satisfied. In order that the polynomials join together smoothly, we require both continuity of slope (y') and curvature (y'') between adjacent polynomials at the $n-2$ internal data points. This gives $2(n-2)$ extra equations, making a total of $4n-6$ equations. With these equations we can determine an identical number of coefficients uniquely, and so two further equations are required in order to determine *all* the unknown coefficients. The two remaining conditions can be chosen arbitrarily, but usually one of the following is used:

1. If the slope of the required curve is known at the outer ends, we can impose these two constraints. More often than not, these slopes are not known.
2. We can make the curvature at the outer ends zero, that is, $y_1'' = y_n'' = 0$. (These are called natural splines but have no particular advantage.)
3. We can make the curvature at x_1 and x_n equal to x_2 and x_{n-1}, respectively.
4. We can make the curvature at x_1 a linear extrapolation of the curvature at x_2 and x_3. Similarly, we make the curvature at x_n a linear extrapolation of the curvature at x_{n-1} and x_{n-2}.
5. We can make y''' continuous at x_2 and x_{n-1}. Since at any internal point, y, y', y'', and y''' are always made continuous, adding this condition is equivalent to using the same polynomial in the two outer panels. This is called the "not a knot" condition and is used in the MATLAB implementation of the function `spline`.

Table 7.1 Data for Spline Fit

x	0	1	2	3	4
y	3	1	0	2	4

We now illustrate the two uses of the MATLAB function `spline` applied to the small set of data given in Table 7.1. Running the script

```
% e3s703.m
x = 0:4; y = [3 1 0 2 4];
xval = 1.5; yval = spline(x,y,xval)
p = spline(x,y)
```

gives

```
yval =
    0.1719

p =
      form: 'pp'
    breaks: [0 1 2 3 4]
     coefs: [4x4 double]
    pieces: 4
     order: 4
       dim: 1
```

where `yval` is the interpolated value. There may be some occasions when the user wishes to know the values of the coefficient of the polynomials. In this case the p-p form is required, where the abbreviation p-p means piecewise polynomial. The variable `p` is a structure array that provides this information. In particular,

```
>> c = p.coefs

c =
    0.5417   -1.1250   -1.4167    3.0000
    0.5417    0.5000   -2.0417    1.0000
   -0.7083    2.1250    0.5833         0
   -0.7083   -0.0000    2.7083    2.0000
```

In this example, the coefficients of the polynomials are combined with the powers of x as follows:

$$
\begin{aligned}
y &= c_{11}x^3 + c_{12}x^2 + c_{13}x + c_{14}, && 0 \le x \le 1 \\
y &= c_{21}(x-1)^3 + c_{22}(x-1)^2 + c_{23}(x-1) + c_{24}, && 1 \le x \le 2 \\
y &= c_{31}(x-2)^3 + c_{32}(x-2)^2 + c_{33}(x-2) + c_{34}, && 2 \le x \le 3 \\
y &= c_{41}(x-3)^3 + c_{42}(x-3)^2 + c_{43}(x-3) + c_{44}, && 3 \le x \le 4
\end{aligned}
$$

It is not necessary for the MATLAB user to know the details of how the p-p values are interpreted. MATLAB provides a function `ppval` that evaluates a composite polynomial provided its p-p values are known. If `x` and `y` are vectors of data, then `y1 = spline(x,y,x1)` is equivalent to the statements `p = spline(x,y); y2 = ppval(p,x1)`.

The following script gives a plot of the spline fit to the data of Table 7.1.

```
% e3s704.m
x = 0:4; y = [3 1 0 2 4];
xx = 0:.1:4; yy = spline(x,y,xx);
plot(x,y,'o',xx,yy)
axis([0 4 -1 4])
xlabel('x'), ylabel('y')
```

Running this script generates Figure 7.3.

Section 7.2 described how polynomials are used in interpolation. However, their use is not always appropriate. When the data points are widely spaced, and when there are sudden changes in the y values, then polynomials can give very poor results. For example, the nine data points in Figure 7.4 are taken from the function

$$y = 2\{1 + \tanh(2x)\} - x/10$$

This function changes abruptly, and if an eighth-degree polynomial is fitted to the data it oscillates and the path between data points bears no relationship to the true path. In contrast the spline fit is reasonably smooth and close to the true function.

The reader should note that the MATLAB function `interp1` can also be used to fit splines to data. The call `interp1(x,y,xi,'spline')` is identical to `spline(x,y,xi)`.

A special type of spline is the Bézier curve. This is a cubic function defined by four points. The two end points are used, together with two "control" points. The slope of the curve at one end is a tangent to the line between that end point and one of the control points. Similarly, the slope at the other end point is a tangent to the line between that end

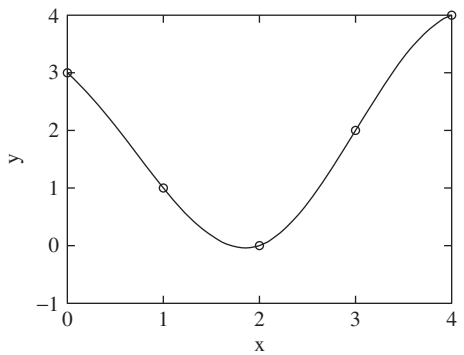

FIGURE 7.3 Spline fit to the data of Table 7.1 (denoted by ○).

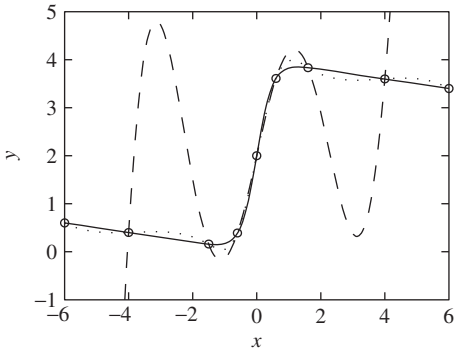

FIGURE 7.4 The *solid curve* shows the function $y = 2\{1 + \tanh(2x)\} - x/10$. The *dashed line* shows an eighth-degree polynomial fit; the *dotted line* shows a spline fit.

point and the other control point. In interactive computer graphics, the positions of the control points can be moved on the screen in order to adjust the slope of the curve at the end points.

7.4 Fourier Analysis of Discrete Data

Fourier analysis in its various forms is an important tool for the scientist or engineer engaged in the interpretation of data where a knowledge of the frequencies present in the data or function may give some insight into the mechanism that has generated it. For continuous periodic functions, the frequency content is determined from the coefficients of the terms in the well-known Fourier series; for nonperiodic functions, it is determined from the Fourier integral transform. In an analogous manner, the frequency content of a sequence of data can be determined by Fourier analysis, in this case from the *discrete* Fourier transform (DFT). The harmonic functions fit the data exactly, but here the purpose is most likely to determine the harmonic content of the data rather than to interpolate new data values.

The data can come from many sources. For example, the radial forces acting at discrete points around a cylinder constitute a sequence of data that must be periodic. The most frequently occurring form of data is a time series in which the value of some quantity is given at equal intervals of time—for example, data sampled from a signal from a transducer—and for this reason the analysis that follows is developed in terms of the independent variable t, which represents time. It must be stressed, however, that the DFT can be applied to any data regardless of the domain from which it originates. Determining the DFT for a sequence of data points is straightforward, although the computation is tedious.

We begin by defining a periodic function. A function $y(t)$ is periodic if it has the property that for any value of time t, $y(t) = y(t + T)$ where T is the time period, typically measured in seconds. The reciprocal of the period is equal to the frequency, denoted by f and measured in cycles/second. In the SI system of units, 1 Hertz (Hz) is defined as 1 cycle/second.

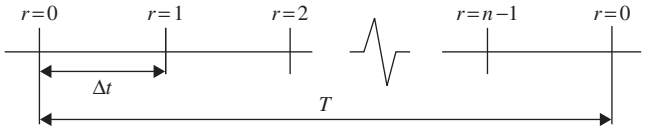

FIGURE 7.5 Numbering scheme for data points.

If we are concerned with a periodic function $z(x)$, where x is a spatial variable, then for any value of x, $z(x) = z(x+X)$ where X is the spatial period or wavelength, typically measured in meters. The frequency $f = 1/X$ is then measured in cycles/meter.

We now examine how to fit a finite set of trigonometric functions to n data points (t_r, y_r) where $r = 0, 1, 2, \ldots, n-1$. We assume the data points are equispaced and the number of data points, n, is even. Data values may be complex but in most practical situations they are real. The data points are numbered as shown in Figure 7.5. The point following the $(n-1)$th is assumed to equal the value of the zero point. Thus the DFT assumes the data is periodic with a period T equal to the range of the data.

Let the relationship between y_r and t_r be given by a finite set of sine and cosine functions as follows:

$$y_r = \frac{1}{n}\left[A_0 + \sum_{k=1}^{m-1}\left\{A_k \cos(2\pi k t_r/T) + B_k \sin(2\pi k t_r/T)\right\} + A_m \cos(2\pi m t_r/T)\right] \tag{7.2}$$

where $r = 0, 1, 2, \ldots, n-1$, $m = n/2$ and T is the range of the data as shown in Figure 7.5. The n coefficients A_0, A_m, A_k, and B_k (where $k = 1, 2, \ldots, m-1$) must be determined. Since we have n data values and n unknown coefficients, (7.2) can be made to fit the data exactly. The factor $1/n$ in (7.2) is omitted by some authors, and omitting it has the effect of reducing the size of the coefficients A_0, A_m, A_k, and B_k by the factor n. The reason for choosing $m+1$ coefficients multiplied by a cosine function (including $\cos 0$, which equals one and in fact multiplies A_0) and $m-1$ coefficients multiplied by a sine function in (7.2) will become apparent.

Each sine or cosine term of (7.2) represents k complete cycles in the range of the data T. Thus the period of each sine term is T/k, where $k = 1, 2, \ldots, (m-1)$, and the period of each cosine term is T/k, where $k = 1, 2, \ldots, m$. The corresponding frequencies are given by k/T. Thus the frequencies present in (7.2) are $1/T$, $2/T$, $\ldots$, m/T. Letting Δf be the frequency increment between components and f_{max} be the maximum frequency, then

$$\Delta f = 1/T \tag{7.3}$$

and

$$f_{max} = m\Delta f = (n/2)\Delta f = n/(2T) \tag{7.4}$$

The data values t_r are equally spaced in the range T and may be expressed as

$$t_r = rT/n, \quad r = 0, 1, 2, \ldots, n-1 \tag{7.5}$$

Letting Δt be the sampling interval (see Figure 7.5), then

$$\Delta t = T/n \tag{7.6}$$

Let T_0 be the period corresponding to f_{max}, the maximum frequency in (7.2). Then, from (7.4),

$$f_{max} = 1/T_0 = n/(2T)$$

Thus $T = T_0 n/2$. Substituting this relationship in (7.6), we have $\Delta t = T_0/2$. This tells us that the maximum frequency component in the DFT contains two samples of data per cycle. The maximum frequency, f_{max}, is called the Nyquist frequency, and the corresponding sampling rate is called the Nyquist sampling rate.

A harmonic with a frequency that is exactly equal to the Nyquist frequency cannot be properly detected because at this frequency the DFT has a cosine term but no corresponding sine term. This result has an important implication when data is sampled from a continuously varying function or signal. It implies that there must be *more than two* data samples per cycle at the highest frequency *present in the function or signal*. If there are frequencies in the signal higher than the Nyquist frequency, then, because of the periodic nature of the DFT itself, they appear as frequency components in the DFT at a lower frequency. This phenomenon is called "aliasing." For example, if data is sampled at 0.005 second intervals, that is, 200 samples per second, then the Nyquist frequency, f_{max}, is 100 Hz. A frequency of 125 Hz in this signal would appear as a frequency component at 75 Hz. A frequency of 225 Hz would appear as 25 Hz. The relationship between frequencies in a signal and the frequency components in the DFT is shown in Figure 7.6. Frequency aliasing should be avoided because it makes it difficult or impossible to relate the frequency components in the DFT to their physical causes.

We now return to the task of determining the n coefficients A_0, A_m, A_k, and B_k in (7.2). Replacing $t_r = rT/n$ in (7.2), we obtain

$$y_r = \frac{1}{n}\left[A_0 + \sum_{k=1}^{m-1}\left\{A_k \cos(2\pi kr/n) + B_k \sin(2\pi kr/n)\right\} + A_m \cos(\pi r)\right] \tag{7.7}$$

where $r = 0, 1, 2, \ldots, n-1$. It was previously noted that the coefficients B_0 and B_m are absent from (7.2). It is now clear that had we introduced these coefficients they would be multiplied by $\sin(0)$ and $\sin(\pi r)$, both of which are zero.

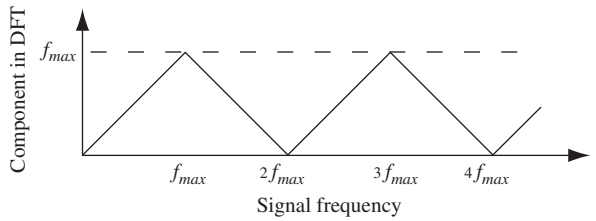

FIGURE 7.6 Relationship between a signal frequency and its component in the DFT derived by sampling using a Nyquist frequency f_{max}.

In (7.7) the n unknown coefficients are real. However, (7.7) can be expressed more concisely in terms of complex exponentials with complex coefficients. Using the fact that

$$\cos(2\pi kr/n) = \{\exp(\imath 2\pi kr/n) + \exp(-\imath 2\pi kr/n)\}/2$$
$$\sin(2\pi kr/n) = \{\exp(\imath 2\pi kr/n) - \exp(-\imath 2\pi kr/n)\}/2\imath$$

and

$$\exp\{\imath 2\pi(n-k)r/n\} = \exp(-\imath 2\pi kr/n)$$

where $k = 1, 2, \ldots, m-1$ and $\imath = \sqrt{-1}$, then it can be shown that (7.7) reduces to

$$y_r = \frac{1}{n}\sum_{k=0}^{n-1} Y_k \exp(\imath 2\pi kr/n), \quad r = 0,\ 1\ 2, \ldots,\ n-1 \tag{7.8}$$

In (7.8),

$$Y_0 = A_0 \quad \text{and} \quad Y_m = A_m, \quad \text{where} \quad m = n/2$$
$$Y_k = (A_k - \imath B_k)/2 \quad \text{and} \quad Y_{n-k} = (A_k + \imath B_k)/2, \quad \text{for} \quad k = 1, 2, \ldots, m-1$$

Note that if y_r is real, then A_k and B_k are also real, so that Y_{n-k} is the complex conjugate of Y_k, for $k = 1, 2, \ldots, (n/2-1)$. To find the values of the unknown complex coefficients of (7.8) we make use of the following orthogonal property of exponential functions sampled at n equispaced points:

$$\sum_{r=0}^{n-1} \exp(\imath 2\pi rj/n)\exp(\imath 2\pi rj/n) = \begin{cases} 0 & \text{if } |j-k| \neq 0,\ n,\ 2n \\ n & \text{if } |j-k| = 0,\ n,\ 2n \end{cases} \tag{7.9}$$

Multiplying (7.8) by $\exp(-\imath 2\pi rj/n)$, summing over the n values of r, and then using (7.9), an expression for the unknown coefficients can be found:

$$Y_k = \sum_{r=0}^{n-1} y_r \exp(\imath 2\pi kr/n) \quad k = 0,\ 1,\ 2, \ldots,\ n-1 \tag{7.10}$$

If we let $W_n = \exp(-\iota 2\pi/n)$, where W_n is a complex constant, then (7.10) becomes

$$Y_k = \sum_{r=0}^{n-1} y_r W_n^{kr} \quad k = 0,\ 1,\ 2, \ldots,\ n-1 \tag{7.11}$$

Note that in (7.11) W_n is raised to the power kr. Alternatively, we can write (7.11) in matrix notation, giving

$$\mathbf{Y} = \mathbf{W}\mathbf{y} \tag{7.12}$$

where W_n^{kr} is the element of the $(k+1)$th row, $(r+1)$th column of $\mathbf{W}$ since k and r both start at zero. Note that $\mathbf{W}$ is an $n \times n$ array of complex coefficients. $\mathbf{Y}$ is a *vector* of the complex Fourier coefficients and in this instance we are departing from our usual convention that emboldened uppercase letters represent arrays.

We can obtain the coefficients Y_k from the equispaced data (t_r, y_r) by using (7.10), (7.11), or (7.12). These equations are alternative statements of the DFT. Furthermore, by replacing k in (7.10) by $k + np$, where p is any integer, it can be shown that $Y_{k+np} = Y_k$. Thus the DFT is periodic over the range n. The inverse of the DFT is called the inverse discrete Fourier transform (IDFT) and is implemented by (7.8). By replacing r in (7.8) by $r + np$, where p is an integer, it can be shown that the IDFT is also periodic over the range n. Both y_r and Y_k may be complex, although, as previously stated, the samples y_r are usually real. These transforms constitute a pair: If the data values are transformed by the DFT to determine the coefficients Y_k, then they can be recovered in their entirety by means of the IDFT.

To evaluate the coefficients of the DFT it would appear convenient to use (7.12). Although using these equations is satisfactory for small sequences of data, calculating the DFT for n real data points requires $2n^2$ multiplications. Thus, to transform a sequence of 4096 data points would require approximately 33 million multiplications. In 1965 this situation was dramatically changed with the publication of the fast Fourier transform (FFT) algorithm (Cooley and Tukey, 1965). The FFT algorithm is extremely efficient, and approximately $2n\log_2 n$ multiplications are required to compute the FFT for real data. With this development, allied to the developments in computing hardware that have occurred in the past forty years, it is now possible to compute the FFT for a relatively large number of data points on a personal computer.

Many refinements have been made to the basic FFT algorithm since it was first formulated, and several variants have been developed. Here one of the simplest forms of the algorithm is outlined.

To develop the basic FFT algorithm one further restriction must be placed on the data. In addition to the data being equispaced, the number of data points must be an integer power of 2. This allows a sequence of data to be successively subdivided. For example, 16 data points can be divided into two sequences of 8, four sequences of 4, and finally eight sequences of only 2 data points. A crucial relationship on which the FFT algorithm is based is now developed from (7.10) as follows. Let y_r be the sequence of n data points for which

we require the DFT. We can subdivide y_r into two sequences of $n/2$ data points u_r and v_r as follows:

$$\left.\begin{aligned} u_r &= y_{2r} \\ v_r &= y_{2r+1} \end{aligned}\right\} \tag{7.13}$$

Note that alternate points in the original data sequence are placed in different subsets. We now determine the DFTs of the data sets u_r and v_r from (7.10), with n replaced by $n/2$:

$$\left.\begin{aligned} U_k &= \sum_{r=0}^{n/2-1} u_r \exp\{-\imath 2\pi kr/(n/2)\} \\ V_k &= \sum_{r=0}^{n/2-1} v_r \exp\{-\imath 2\pi kr/(n/2)\} \end{aligned}\right\} \quad k = 0,\ 1,\ 2, \ldots,\ n/2-1 \tag{7.14}$$

The DFT, Y_k, for the original data sequence y_r is given by using (7.10) as follows:

$$\begin{aligned} Y_k &= \sum_{r=0}^{n-1} y_r \exp(-\imath 2\pi kr/n) \\ &= \sum_{r=0}^{n/2-1} y_{2r} \exp\{-\imath 2\pi k2r/n\} + \sum_{r=0}^{n/2-1} y_{2r+1} \exp\{-\imath 2\pi k(2r+1)/n\} \end{aligned}$$

where $k = 0, 1, 2, \ldots, n$. Substituting for y_{2r} and y_{2r+1} from (7.13), we have

$$Y_k = \sum_{r=0}^{n/2-1} u_r \exp\{-\imath 2\pi kr/(n/2)\} + \exp(-\imath 2\pi k/n) \sum_{r=0}^{n/2-1} v_r \exp\{-\imath 2\pi kr/(n/2)\}$$

Comparing this equation with (7.14), we see that

$$Y_k = U_k + \exp(-\imath 2\pi k/n) V_k = U_k + (W_n^k) V_k \tag{7.15}$$

where $W_n^k = \exp(-\imath 2\pi k/n)$ and $k = 0, 1, 2, \ldots, n/2-1$.

Equation (7.15) provides only half of the required DFT. However, using the fact that U_k and V_k are periodic in k, it can be proved that

$$Y_{k+n/2} = U_k - \exp(-\imath 2\pi\, k/n) V_k = U_k - (W_n^k) V_k \tag{7.16}$$

We can use (7.15) and (7.16) to determine efficiently the DFT of the original data from the DFTs of subsets composed of alternate points of the original data. Of course, we can determine the DFTs of each subset of data by further subdividing these subsets until the final division leaves subsets consisting of a single data point. For a sequence of data comprising a single data point, we see from (7.10) with $n = 1$ that the DFT is equal to the value of the single data point. This is essentially how the FFT algorithm works.

In the preceding discussion we started from a sequence of data and continuously subdivided it (with alternate points in different subsets) until the subdivisions produced single

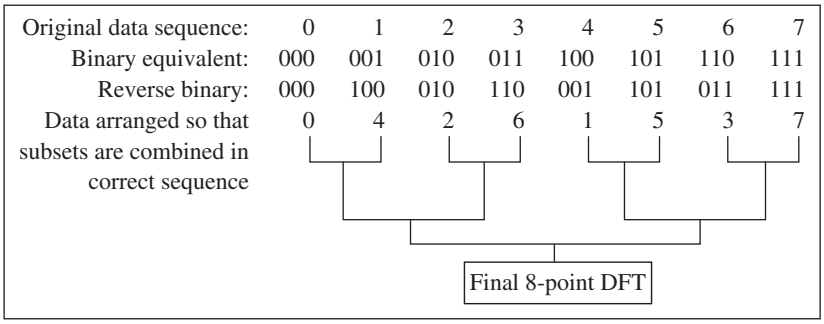

FIGURE 7.7 Stages in the FFT algorithm.

data points. What we require is a method of starting with single data points and ordering them in such a way that successively combining the DFTs of the subsets ultimately forms the required DFT of the original data. This can be achieved by the "bit reversed algorithm," and we illustrate it and the subsequent stages of the FFT by assuming a sequence of eight data points, y_0 to y_7. To determine the correct order for combining the data we express the subscript denoting the original position of each data point as a binary number and reverse the order of the digits (or bits). This reversed-order binary number determines the position of each data point in the reordered sequence and is shown for eight data points in Figure 7.7. The diagram also shows the stages of the FFT algorithm, which repeatedly uses (7.15) and (7.16) as follows:

Stage 1 Determine $\mathbf{Y}_{04}$ from Y_0 and Y_4, determine $\mathbf{Y}_{26}$ from Y_2 and Y_6, determine $\mathbf{Y}_{15}$ from Y_1 and Y_5, determine $\mathbf{Y}_{37}$ from Y_3 and Y_7.

Stage 2 Determine $\mathbf{Y}_{0246}$ from $\mathbf{Y}_{04}$ and $\mathbf{Y}_{26}$, determine $\mathbf{Y}_{1357}$ from $\mathbf{Y}_{15}$ and $\mathbf{Y}_{37}$.

Stage 3 Determine $\mathbf{Y}_{01234567}$ from $\mathbf{Y}_{0246}$ and $\mathbf{Y}_{1357}$.

Note that there are three stages in this process. For an n point DFT, the number of stages equals $\log_2 n$. In this small example, $n = 8$ and hence $\log_2 8 = \log_2 2^3 = 3$. Thus the process requires three stages.

MATLAB provides both the function `fft` to determine the DFT of a sequence of data values using the FFT algorithm, and the function `ifft` to determine the IDFT using a slight modification of the FFT algorithm. Thus to determine the DFT of the data in `y` we use the `fft` function as the following script illustrates:

```
% e3s705.m
v = 0:15;
y = [2.8 -0.77 -2.2 -3.1 -4.9 -3.2 4.83 -2.5 3.2 ...
     -3.6 -1.1 1.2 -3.2 3.3 -3.4 4.9];
s = sum(y), Y = fft(y);
[v' Y.']
```

Running this script gives the following results:

```
s =
   -7.7400

ans =
         0            -7.7400
    1.0000             3.2959 + 8.3851i
    2.0000            13.9798 +10.9313i
    3.0000             8.0796 - 6.6525i
    4.0000            -0.2300 + 4.7700i
    5.0000             4.3150 + 6.8308i
    6.0000            14.2202 + 1.4713i
    7.0000           -17.2905 +15.0684i
    8.0000            -0.2000
    9.0000           -17.2905 -15.0684i
   10.0000            14.2202 - 1.4713i
   11.0000             4.3150 - 6.8308i
   12.0000            -0.2300 - 4.7700i
   13.0000             8.0796 + 6.6525i
   14.0000            13.9798 -10.9313i
   15.0000             3.2959 - 8.3851i
```

We have already noted that for real data Y_{n-k} is the complex conjugate of Y_k, for $k = 1, 2, \ldots, (n/2 - 1)$. The preceding results illustrate this relationship and in this case $Y_{15}, Y_{14}, \ldots, Y_9$ are the complex conjugates of $Y_1, Y_2, \ldots, Y_7$, respectively, and provide no extra information. Note also that Y_0 is equal to the sum of the original data values y_r.

We now give examples of the use of the `fft` function to examine the frequency content of data sequences sampled from continuous functions.

Example 7.1

Determine the DFT of a sequence of 64 equispaced data points, sampled at intervals of 0.05 s from the function $y = 0.5 + 2\sin(2\pi f_1 t) + \cos(2\pi f_2 t)$, where $f_1 = 3.125$ Hz and $f_2 = 6.25$ Hz. The following script calls the `fft` function and displays the resulting DFT in various ways:

```
% e3s706.m
clf
nt = 64; dt = 0.05; T = dt*nt
df = 1/T, fmax = (nt/2)*df
t = 0:dt:(nt-1)*dt;
y = 0.5+2*sin(2*pi*3.125*t)+cos(2*pi*6.25*t);
f = 0:df:(nt-1)*df;  Y = fft(y);
figure(1)
```

```
subplot(121), bar(real(Y),'r')
axis([0 63 -100 100])
xlabel('Index k'), ylabel('real(DFT)')
subplot(122), bar(imag(Y),'r')
axis([0 63 -100 100])
xlabel('Index k'), ylabel('imag(DFT)')
fss = 0:df:(nt/2-1)*df;
Yss = zeros(1,nt/2); Yss(1:nt/2) = (2/nt)*Y(1:nt/2);
figure(2)
subplot(221), bar(fss,real(Yss),'r')
axis([0 10 -3 3])
xlabel('Frequency (Hz)'), ylabel('real(DFT)')
subplot(222), bar(fss,imag(Yss),'r')
axis([0 10 -3 3])
xlabel('Frequency (Hz)'), ylabel('imag(DFT)')
subplot(223), bar(fss,abs(Yss),'r')
axis([0 10 -3 3])
xlabel('Frequency (Hz)'), ylabel('abs(DFT)')
```

Running the preceding script gives

```
T =
    3.2000

df =
    0.3125

fmax =
    10
```

together with Figures 7.8 and 7.9. Note that in the script we have used the `bar` rather than the `plot` statement to emphasize the discrete nature of the DFT. Figure 7.8 shows the amplitudes of the 64 real and imaginary components of the DFT plotted against the index number k. Note that components 63 to 33 are the complex conjugates of components 1 to 31. While these plots display the DFT, the amplitude and frequency of the harmonic components in the original signal cannot easily be recognized. To achieve this the DFT must be scaled and displayed as shown in Figure 7.8. In the real part of the DFT there are components at $k = 0$, 20, and 44, each with an amplitude of 32, and in the imaginary part of the DFT there are components at $k = 10$ and 54, with amplitudes of 64 and -64, respectively. Since they contain no extra information, we ignore the components above $k = 32$ (i.e., $k = 44$ and 54) and consider only the components in the range $k = 0, 1, \ldots, 31$; in this case specifically $k = 0$, 10, and 20. We can convert the DFT index number to frequency by multiplying by Δf ($= 0.3125$ Hz) to give components at 0 Hz, 3.125 Hz, and 6.25 Hz, respectively. We now scale the DFT in the range $k = 1, 2, \ldots, 31$ by dividing it by $(n/2)$, in this case by 32.

The plots of the 31 scaled DFT components (most of which are zero) corresponding to frequencies in the range 0 to 9.6875 Hz are shown in Figure 7.9. We now see that the real component at 6.25 Hz has an amplitude of 1 and the imaginary component at 3.125 Hz has

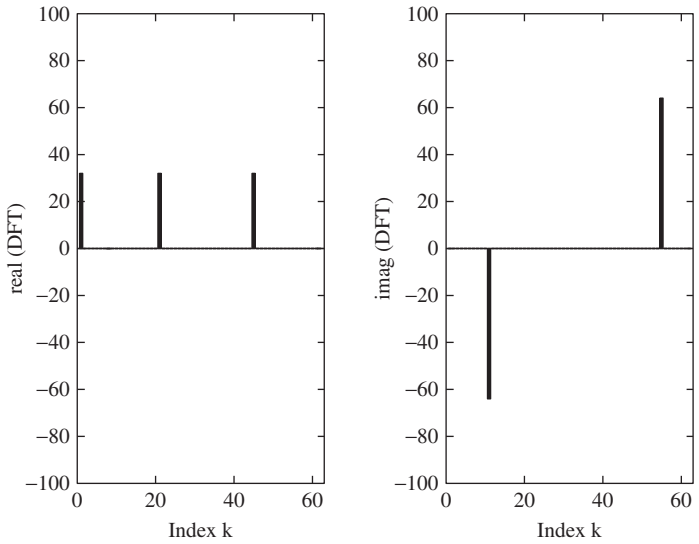

FIGURE 7.8 Plots of the real and imaginary part of the DFT.

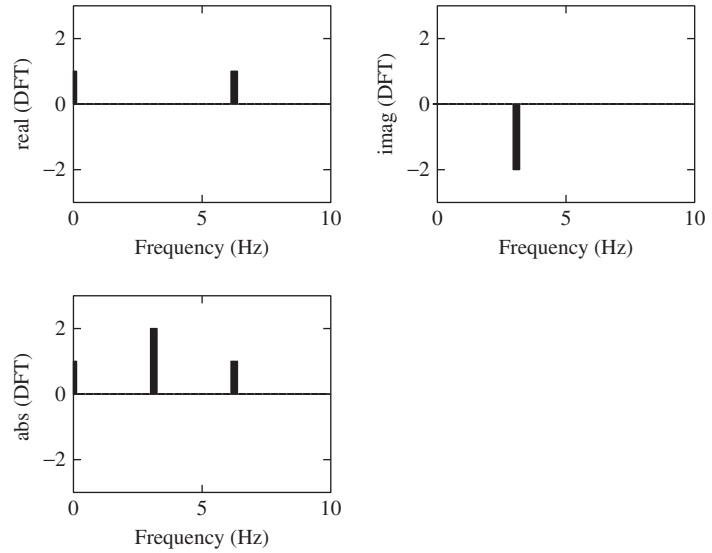

FIGURE 7.9 Frequency spectra.

an amplitude of −2. These components correspond, respectively, to the cosine component and the negative of the sine component in the original signal from which the data was sampled. If we only wish to know the amplitude of the frequency components, then we can display the absolute values of the scaled DFT. The component at $f = 0$ Hz is equal to *twice* the mean value of the data; in this case we have $2 \times 0.5 = 1$. These plots are called frequency spectra or periodograms. If sampling is over an integer number of cycles of the harmonics present in the signal,

the amplitude of the components in the scaled DFT equal the amplitude of the corresponding harmonics, as shown in this example. If the sampling is not over an integer number of cycles of any harmonic present in the signal, then the component in the DFT closest to the frequency of the harmonic is reduced in amplitude and spread into other frequencies. This phenomenon is called "smearing" or "leakage" and is further discussed in Problem 7.15.

Example 7.2

We now determine the spectrum of a sequence of 512 data points sampled over a period of 2 seconds from the function

$$y = 0.2\cos(2\pi f_1 t) + 0.35\sin(2\pi f_2 t) + 0.3\sin(2\pi f_3 t) + \text{random noise}$$

where $f_1 = 20$ Hz, $f_2 = 50$ Hz, and $f_3 = 70$ Hz. The random noise is normally distributed with a standard deviation of 0.5 and a mean of zero. The following script plots the time series and the DFT scaled by the factor $n/2$.

```
% e3s707.m
clf
f1 = 20; f2 = 50; f3 = 70;
nt = 512; T = 2; dt = T/nt
t_final = (nt-1)*dt; df = 1/T
fmax = (nt/2)*df;
t = 0:dt:t_final;
dt_plt = dt/25;
t_plt = 0:dt_plt:t_final;
y_plt = 0.2*cos(2*pi*f1*t_plt)+0.35*sin(2*pi*f2*t_plt) ...
                                    +0.3*sin(2*pi*f3*t_plt);
y_plt = y_plt+0.5*randn(size(y_plt));
y = y_plt(1:25:(nt-1)*25+1); f = 0:df:(nt/2-1)*df;
figure(1);
subplot(211), plot(t_plt,y_plt)
axis([0 0.04 -3 3])
xlabel('Time (sec)'), ylabel('y')
yf = fft(y);
yp(1:nt/2) = (2/nt)*yf(1:nt/2);
subplot(212), plot(f,abs(yp))
axis([0 fmax 0 0.5])
xlabel('Frequency (Hz)'), ylabel('abs(DFT)');
```

Running the preceding script gives

```
dt =
    0.0039

df =
    0.5000
```

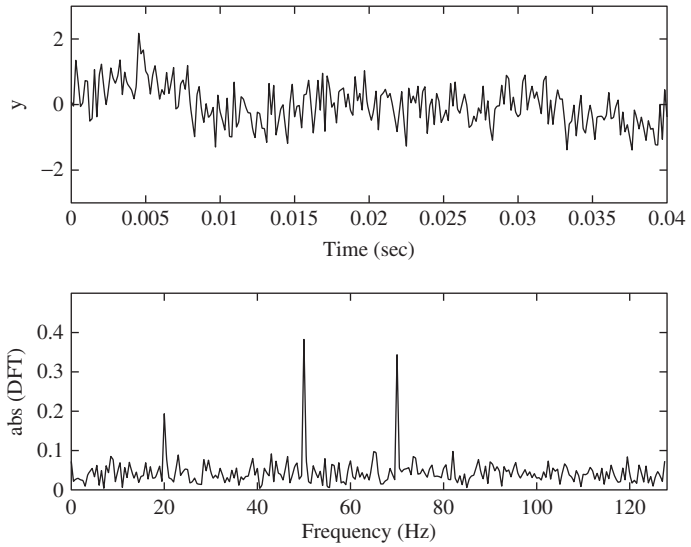

FIGURE 7.10 Signal and frequency spectrum showing frequency components at 20, 50, and 70 Hz.

together with the graphical output shown in Figure 7.10. The lower plot of Figure 7.10 shows that random noise in the signal does not prevent the frequency components 20, 50, and 70 Hz from revealing themselves in the spectrum. These components are not obviously visible in the original time series data shown in the upper plot of Figure 7.10.

Example 7.3

Determine the spectrum of a triangular wave of amplitude ±1 and period 1 second, sampled at 1/32 second intervals over one cycle. The following script outputs the DFT scaled by the factor $n/2$.

```
% e3s708.m
nt = 32; T = 1, dt = T/nt
t = 0:dt:(nt-1)*dt;
df = 1/T, fmax = nt/(2*T)
f = 0:df:df*(nt/2-1);
y = 0.125*[8 7 6 5 4 3 2 1 0 -1 -2 -3 -4 -5 -6 -7 -8 ...
    -7 -6 -5 -4 -3 -2 -1 0 1 2 3 4 5 6 7];
Yss = zeros(1,nt/2); Y = fft(y);
Yss(1:nt/2) = (2/nt)*Y(1:nt/2);
[f' abs(Yss)']
```

Running this script gives

```
T =
     1

dt =
    0.0313

df =
     1

fmax =
    16

ans =
         0         0
    1.0000    0.8132
    2.0000         0
    3.0000    0.0927
    4.0000         0
    5.0000    0.0352
    6.0000         0
    7.0000    0.0194
    8.0000         0
    9.0000    0.0131
   10.0000         0
   11.0000    0.0100
   12.0000         0
   13.0000    0.0085
   14.0000         0
   15.0000    0.0079
```

The Fourier series for the triangular wave of this example is

$$f(t) = \frac{8}{\pi^2}\left(\cos(2\pi t) + \frac{1}{3^2}\cos(6\pi t) + \frac{1}{5^2}\cos(10\pi t) + \frac{1}{7^2}\cos(14\pi t) + \cdots\right)$$

The first eight frequency components in the scaled DFT at frequencies 1, 3, 5 Hz, and so on, are not equal to $8/\pi^2$, $8/(3\pi)^2$, $8/(5\pi)^2$ (i.e., 0.8106, 0.0901, 0.0324), and so on, because of the effect of aliasing. A triangular wave contains an infinite number of harmonics and because of aliasing these appear as components in the DFT as shown in Table 7.2. Thus the size of the 3 Hz component in the DFT is $(8/\pi^2)(1/3^2 + 1/29^2 + 1/35^2 + 1/61^2 + \cdots)$. By summing a large number of terms down the columns of Table 7.2, the terms in the DFT are obtained. The DFT as shown in the preceding is correct, and the inverse DFT recovers the original data. However,

Table 7.2 Coefficients of Aliased Harmonics

f	**3*f***	**5*f***	**7*f***	**9*f***	**11*f***	**13*f***	**15*f***
$8/\pi^2$	$8/(3\pi)^2$	$8/(5\pi)^2$	$8/(7\pi)^2$	$8/(9\pi)^2$	$8/(11\pi)^2$	$8/(13\pi)^2$	$8/(15\pi)^2$
$8/(31\pi)^2$	$8/(29\pi)^2$	$8/(27\pi)^2$	$8/(25\pi)^2$	$8/(23\pi)^2$	$8/(21\pi)^2$	$8/(19\pi)^2$	$8/(17\pi)^2$
$8/(33\pi)^2$	$8/(35\pi)^2$	$8/(37\pi)^2$	$8/(39\pi)^2$	$8/(41\pi)^2$	$8/(43\pi)^2$	$8/(45\pi)^2$	$8/(47\pi)^2$
$8/(63\pi)^2$	$8/(61\pi)^2$	$8/(59\pi)^2$	$8/(57\pi)^2$	$8/(55\pi)^2$	$8/(53\pi)^2$	$8/(51\pi)^2$	$8/(49\pi)^2$
$8/(65\pi)^2$	$8/(67\pi)^2$	etc.					

when using it to provide information about the contribution of frequency components in the original data, the DFT must be interpreted with care.

Example 7.4

Determine the DFT of a sequence of 128 data points sampled from a signal at intervals of 0.0625 seconds. The signal has a constant amplitude of 1 unit which, after 10 samples, is switched to zero.

The following script determines the DFT for the data.

```
% e3s709.m
clf
nt = 128; nb = 10;
y = [ones(1,nb) zeros(1,nt-nb)];
dt = 0.0625; T = dt*nt
df = 1/T, fmax = (nt/2)*df;
f = 0:df:(nt/2-1)*df;
yf = fft(y);
yp = (2/nt)*yf(1:nt/2);
figure(1), bar(f,abs(yp),'w')
axis([0 fmax 0 0.2])
xlabel('Frequency (Hz)'), ylabel('abs(DFT)')
```

Running this script gives

```
T =
    8

df =
    0.1250
```

together with the graphical output shown in Figure 7.11. The plot shows that the frequency spectrum is continuous and the largest components are clustered near the zero frequency. This is in contrast to the spectra of Examples 7.1 and 7.2, which show sharp peaks due to the presence

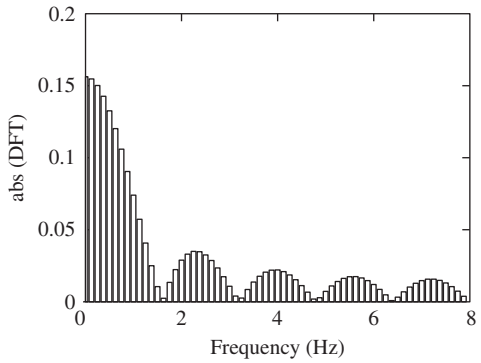

FIGURE 7.11 Spectrum of a sequence of data.

of periodic components in the original data. Note that because the original signal is a step and not periodic, the amplitude of its DFT is dependent on the sampling period.

In this section we have provided examples of how the DFT (computed using the FFT) can be used to study the distribution of frequency components in data. There are other applications of the DFT. It is sometimes used for interpolation, as is any procedure that fits mathematical functions to a sequence of data. MATLAB provides the function `interpft` to allow interpolation using the DFT.

The advances in computer hardware have extended the range of problems to which the DFT can usefully be applied, and this in turn has encouraged the development of new and powerful variants of the FFT algorithm. A detailed description of the FFT algorithm is given by Brigham (1974), and a straightforward introduction that emphasizes the practical problems in using this type of analysis is given by Ramirez (1985).

7.5 Multiple Regression: Least Squares Criterion

We now consider the problem of fitting a function to a relatively large amount of data that contains errors. It would not be sensible, nor computationally possible, to fit a very high-degree polynomial to a large amount of experimental data as may be done for interpolation. What is required is a function that smooths out fluctuations in the data due to errors and reveals any underlying trend. We therefore adjust the coefficients of a chosen function to provide a "best fit" according to some criterion. For example, the criterion may be to minimize the maximum error, the sum of the modulus of the errors, or the sum of the squares of the errors between the chosen function and the actual data points. The least squares method is the most widely used of these, criteria and we now examine how this process is carried out.

Suppose we have an n component vector of observations $\mathbf{y}$ and p separate vectors of explanatory variables, $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_p$. The variables $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_p$ are generally independent variables that are measured with negligible error or even controlled in an experiment and $\mathbf{y}$ is a single dependent variable, uncontrolled and containing random measurement errors. In some cases $\mathbf{x}_1$, $\mathbf{x}_2$, and so on, may be different functions of a single explanatory variable—in which case we call them *predictors* (see Sections 7.8 and 7.9). Our basic model consists of a *regression equation* and is said to be a regression of $\mathbf{y}$ upon the explanatory variables $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_p$. We assume a simple linear regression model, and so for the ith observation we have

$$y_i = \beta_0 + \beta_1 x_{1i} + \beta_2 x_{2i} + \cdots + \beta_p x_{pi} + \varepsilon_i, \quad i = 1, 2, \ldots, n \tag{7.17}$$

where β_j $(j = 0, 1, 2, \ldots, p)$ are the unknown coefficients and ε_i are random errors. Initially we simply assume that these random errors are identically distributed with a zero mean and a common unknown variance, σ^2, and that they are independent of each other. This implies that

$$\beta_0 + \beta_1 x_{1i} + \beta_2 x_{2i} + \cdots + \beta_p x_{pi}$$

represents the mean value of y_i.

Our main task is to find an estimate b_j for each unknown β_j so that we can estimate the mean of y_i by fitting a function of the form

$$\hat{y}_j = b_0 + b_1 x_{1i} + b_2 x_{2i} + \cdots + b_p x_{pi}, \quad i = 1, 2, \ldots, n \tag{7.18}$$

The difference between the observed and the fitted value for the ith observation

$$e_i = y_i - \hat{y}_i \tag{7.19}$$

is called the *residual* and is an estimate of the corresponding ε_i. Our criterion for choosing the b_j estimates is that they should minimize the sum of the squares of the residuals, which is often called the *sum of squares of the errors* and is denoted by *SSE*. Thus

$$SSE = \sum_{i=1}^{n} e_i^2 = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \tag{7.20}$$

To perform the necessary calculations efficiently, we rewrite the model in matrix form as follows. We define $\mathbf{b}$ as a $(p+1) \times 1$ vector such that

$$\mathbf{b} = [b_0 \quad b_1 \ \ldots \ b_p]^\top$$

Similarly we define $\mathbf{e}$, $\mathbf{y}$, $\mathbf{u}$, and $\mathbf{x}_j$ to be $n \times 1$ vectors as follows:

$$\begin{aligned}
\mathbf{e} &= [e_1 \quad e_2 \ \ldots \ e_n]^\top \\
\mathbf{y} &= [y_1 \quad y_2 \ \ldots \ y_n]^\top \\
\mathbf{u} &= [1 \quad 1 \ \ldots \ 1]^\top \\
\mathbf{x}_j &= [x_{j1} \quad x_{j2} \ \ldots \ x_{jn}]^\top, \quad j = 1, 2, \ldots, p
\end{aligned}$$

From these vectors the $n \times (p+1)$ matrix, $\mathbf{X}$, can be defined as follows:

$$\mathbf{X} = [\mathbf{u} \quad \mathbf{x}_1 \quad \mathbf{x}_2 \ \ldots \ \mathbf{x}_p]$$

This matrix corresponds to the coefficients of the equation system (7.18). From (7.19) the residuals may then be written as

$$\mathbf{e} = \mathbf{y} - \mathbf{X}\mathbf{b}$$

and the *SSE*, from (7.20), is then

$$SSE = \mathbf{e}^\top \mathbf{e} = (\mathbf{y} - \mathbf{X}\mathbf{b})^\top (\mathbf{y} - \mathbf{X}\mathbf{b}) \tag{7.21}$$

Differentiating the *SSE* with respect to $\mathbf{b}$, we get a vector of partial derivatives, as follows:

$$\frac{\partial}{\partial \mathbf{b}}(SSE) = -2\mathbf{X}^\top (\mathbf{y} - \mathbf{X}\mathbf{b})$$

Matrix differentiation is described in Appendix A. Equating this derivative to zero we have $\mathbf{X}\mathbf{b} = \mathbf{y}$ and this overdetermined system of equations could be solved directly to obtain the coefficients $\mathbf{b}$ using the MATLAB operator \. However, in this instance it is more convenient to proceed as follows. Premultiplying $\mathbf{X}\mathbf{b} = \mathbf{y}$ by $\mathbf{X}^\top$, we obtain what are called the *normal equations*:

$$\mathbf{X}^\top \mathbf{X}\mathbf{b} = \mathbf{X}^\top \mathbf{y}$$

The formal solution of these equations is then

$$\mathbf{b} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y} = \mathbf{C}\mathbf{X}^\top \mathbf{y} \tag{7.22}$$

where

$$\mathbf{C} = (\mathbf{X}^\top \mathbf{X})^{-1} \tag{7.23}$$

Note that $\mathbf{C}$ is a $(p+1) \times (p+1)$ square matrix.

Using the expression for $\mathbf{b}$ in (7.22), the vector of fitted values corresponding to $\mathbf{y}$ is

$$\hat{\mathbf{y}} = \mathbf{X}\mathbf{b} = \mathbf{X}\mathbf{C}\mathbf{X}^\top \mathbf{y}$$

So, defining $\mathbf{H} = \mathbf{X}\mathbf{C}\mathbf{X}^\top$, we can write

$$\hat{\mathbf{y}} = \mathbf{H}\mathbf{y} \tag{7.24}$$

The matrix $\mathbf{H}$, which converts $\mathbf{y}$ to $\hat{\mathbf{y}}$ ("y-hat") is called the *hat matrix* and plays an important role in the interpretation of the regression model. Among its important properties is that it is idempotent (see Appendix A).

From (7.22) it can be shown that the minimum value of *SSE* is given by

$$SSE = \mathbf{y}^\top(\mathbf{I} - \mathbf{H})\mathbf{y} \tag{7.25}$$

where $\mathbf{I}$ is an $n \times n$ identity matrix. Our original data consisted of n sets of observations and we have introduced $p+1$ constraints into the system to estimate the parameters $\beta_0, \beta_1, \ldots, \beta_p$, so there are now $(n-p-1)$ degrees of freedom. Statistical theory shows that by dividing the minimum *SSE* from (7.25) by the number of degrees of freedom, we obtain an unbiased estimate of the unknown error variance, σ^2, which is

$$s^2 = \frac{SSE}{n-p-1} = \frac{\mathbf{y}^\top(\mathbf{I} - \mathbf{H})\mathbf{y}}{n-p-1} \tag{7.26}$$

On its own, the value of s obtained by fitting a single model is not very informative. However, we can also use $\mathbf{H}$ to evaluate the overall goodness of fit on an absolute scale and to examine how good each b_j is as an estimate of the corresponding β_j.

The most widely used measure of the overall goodness of fit is the *coefficient of determination*, R^2, which is defined by

$$R^2 = \sum_{i=1}^{n} (\hat{y}_i - \bar{y})^2 \Bigg/ \sum_{i=1}^{n} (y_i - \bar{y})^2$$

where

$$\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$$

Thus $\bar{y}$ is the mean of the observed y values. Using matrix notation, we can evaluate R^2 from the equivalent definition

$$R^2 = \frac{\mathbf{y}^\top(\mathbf{H} - \mathbf{u}\mathbf{u}^\top/n)\mathbf{y}}{\mathbf{y}^\top(\mathbf{I} - \mathbf{u}\mathbf{u}^\top/n)\mathbf{y}} \tag{7.27}$$

The value of R^2 will lie between 0 and 1 and represents the proportion of the total observed variance of $\mathbf{y}$ that is accounted for by the explanatory variables. Thus a value close to 1 indicates that nearly all the observed variance is accounted for and we have a good fit. However, on its own this does not necessarily indicate that the model is satisfactory

because the value of R^2 can always be increased by introducing more explanatory variables, even though their introduction can have other effects that are very undesirable. In the next section we briefly consider some methods for deciding which explanatory variables should be included in the model.

7.6 Diagnostics for Model Improvement

To see which variables might be removed to improve our original model, we now examine the b_j estimates in more detail to check whether they indicate that the corresponding β_j coefficients are nonzero, which would confirm that the corresponding x_j really do contribute to explaining y, or in the case of b_0, whether it is appropriate to include the constant term β_0 in the model.

If we make the assumption that the random errors, ε_i, are normally distributed, it can be shown that each of the b_j estimates behave as if they were observed values of normal random variables whose means are the β_j and whose covariance matrix is $s^2\mathbf{C}$ where $\mathbf{C}$ is defined by (7.23). Thus the covariance matrix is

$$s^2\mathbf{C} = s^2 \begin{bmatrix} c_{00} & c_{01} & c_{02} & \dots & c_{0p} \\ & c_{11} & c_{12} & \dots & c_{1p} \\ & & c_{22} & \dots & c_{2p} \\ & & & & \vdots \\ & & & & c_{pp} \end{bmatrix}$$

Note that we numbered the rows and columns of $\mathbf{C}$ from zero to p. The matrix $\mathbf{C}$ is symmetric and so the subdiagonal elements are not shown.

The variance of the distribution for b_j is s^2 times the corresponding diagonal element of $\mathbf{C}$, and the *standard error* (SE) of b_j is the square root of this variance:

$$SE(b_j) = s\sqrt{c_{jj}}$$

If β_j is really zero, the statistic

$$t = b_j/SE(b_j)$$

has a Student's t-distribution with $n - p - 1$ degrees of freedom. Thus a formal hypothesis test may be carried out to check whether it is reasonable to assume that β_j is zero and hence the predictor x_j does not make a significant contribution to explaining y. However, as an initial guide to whether x_j should be included in the regression model, it is usually sufficient to check whether the magnitude of the corresponding t-statistic is numerically greater than about 2. If $|t| > 2$, then x_j should be left in the model; otherwise, consideration should be given to removing it.

When there is more than one explanatory variable or predictor in the original regression, there is a possibility that two or more of these may be highly correlated with each

other. This situation is called *multicollinearity*; when it occurs the columns in $\mathbf{X}$ corresponding to the correlated variables are almost linearly related, and this causes $\mathbf{X}^\top\mathbf{X}$ and its inverse $\mathbf{C}$ to be ill-conditioned. Although we shall be able to solve the normal equations for $\mathbf{b}$ as long as $\mathbf{X}^\top\mathbf{X}$ does not actually become singular, the solution is very sensitive to small changes in the data and there will be large off-diagonal elements in $\mathbf{C}$, indicating highly correlated b_j estimates. It is therefore worth calculating the *correlation matrix*, which shows the correlation between y and each of the explanatory variables and the correlations between all pairs of explanatory variables. As in the case of the covariance matrix, we number the rows and columns of the correlation matrix from 0 to p. Thus the correlation matrix is

$$\begin{array}{c} \\ y \\ x_1 \\ x_2 \\ \\ \end{array} \begin{array}{c} \begin{array}{cccc} y & x_1 & x_2 & \end{array} \\ \begin{pmatrix} r_{00} & r_{01} & r_{02} & \dots \\ r_{10} & r_{11} & r_{12} & \dots \\ r_{20} & r_{21} & r_{22} & \dots \\ \dots & \dots & \dots & \dots \end{pmatrix} \end{array}$$

where we define the typical element of the correlation matrix r_{ij} to be the correlation between x_i and x_j and r_{0j} to be the correlation between y and x_j.

In situations such as *polynomial regression*, which is considered in Section 7.8, there will always be high correlations between the predictors, but examination of the t-statistics may indicate that some of the predictors do not contribute significantly to explaining y. Those with the smallest $|t|$ and the smallest $|r_{0j}|$ are the most obvious candidates for discarding.

Another set of statistics that are useful in this context are the *variance inflation factors* (*VIFs*). To find the *VIF* for x_j, we regress x_j upon the other $p-1$ explanatory variables and calculate the coefficient of determination, R_j^2. The corresponding *VIF* is

$$VIF_j = \frac{1}{1-R_j^2}$$

If x_j is almost entirely explained by the other variables, R_j^2 will be close to 1 and VIF_j will be large. A good working rule is to regard any x_j with $VIF_j > 10$ as a candidate for removal.

Note that when there are only two explanatory variables, they will always have equal variance inflation factors; if these are greater than 10, it is best to discard the variable that has the smallest correlation with y. When the model contains predictors that are different functions of some common explanatory variable, the corresponding *VIF* values may be very large, as in the case of polynomial regression described in Section 7.8. Such models would normally be considered for physical data when predictors of this type could be shown to have a causal relationship with y.

The following MATLAB function `mregg2` implements multiple regression; the diagnostics necessary for model improvement are also computed.

```
function [s_sqd R_sqd b SE t VIF Corr_mtrx residual] = mregg2(Xd,con)
% Multiple linear regression, using least squares.
% Example call:
%   [s_sqd R_sqd bt SEt tt VIFt Corr_mtrx residual] = mregg2(Xd,con)
% Fits data to y = b0 + b1*x1 + b2*x2 + ... bp*xp
% Xd is a data array. Each row of X is a set of data.
% Xd(1,:) = x1(:), Xd(2,:) = x2(:), ... Xd(p+1,:) = y(:).
% Xd has n columns corresponding to n data points and p+1 rows.
% If con = 0, no constant is used, if con ~= 0, constant term is used.
% Output arguments:
% s_sqd = Error variance, R_Sqd = R^2.
% b is the row of coefficients b0 (if con~=0), b1, b2, ... bp.
% SE is the row of standard error for the coeff b0 (if con~=0),
%  b1, b2, ... bp.
% t is the row of the t statistic for the coeff b0 (if con~=0),
%  b1, b2, ... bp.
% VIF is the row of the VIF for the coeff b0 (if con~=0), b1, b2, ... bp.
% Corr_mtrx is the correlation matrix.
% residual is an arrray of 4 columns and n rows.
% For each row i, the residual
% array contains the value of y(i), the residual(i), the standardized
% residual(i) and the Cook distance(i) where i is the ith data value.
if con==0
    cst = 0;
else
    cst = 1;
end
[p1,n] = size(Xd);
p = p1-1;  pc = p+cst;
y = Xd(p1,:)';
if cst==1
    w = ones(n,1);
    X = [w Xd(1:p,:)'];
else
    X = Xd(1:p,:)';
end
C = inv(X'*X);  b = C*X'*y; b = b.';
H = X*C*X';  SSE = y'*(eye(n)-H)*y;
s_sqd = SSE/(n-pc); Cov = s_sqd*C;
Z = (1/n)*ones(n);
num = y'*(H-Z)*y; denom = y'*(eye(n)-Z)*y;
R_sqd = num/denom;
```

```
SE = sqrt(diag(Cov)); SE = SE.';
t = b./SE;
% Compute correlation matrix
V(:,1) = (eye(n)-Z)*y;
for j = 1:p
    V(:,j+1) = (eye(n)-Z)*X(:,j+cst);
end
SS = V'*V; D = zeros(p+1,p+1);
for j=1:p+1
    D(j,j) = 1/sqrt(SS(j,j));
end
Corr_mtrx = D*SS*D;
% Compute VIF
for j = 1+cst:pc
    ym = X(:,j);
    if cst==1
        Xm = X(:,[1 2:j-1,j+1:p+1]);
    else
        Xm = X(:,[1:j-1,j+1:p]);
    end
    Cm = inv(Xm'*Xm); Hm = Xm*Cm*Xm';
    num = ym'*(Hm-Z)*ym; denom = ym'*(eye(n)-Z)*ym;
    R_sqr(j-cst) = num/denom;
end
VIF = 1./(1-R_sqr); VIF = [0 VIF];
% Analysis of residuals
ee = zeros(length(y),1);  sr = zeros(length(y),1);
cd = zeros(length(y),1);
if nargout>7
    ee = (eye(n)-H)*y;
    s = sqrt(s_sqd);
    sr = ee./(s*sqrt(1-diag(H)));
    cd = (1/pc)*(1/s^2)*ee.^2.*(diag(H)./((1-diag(H)).^2));
    residual = [y ee sr cd];
end
```

To use `mregg2` it is necessary to provide the $(p+1) \times n$ data array, `Xd`, where p is the number of explanatory variables and n is the number of data sets. Rows 1 to p contain the values of the explanatory variables, x_1 to x_p, and row $p+1$ contains the corresponding value of y. If the parameter `con` is set to zero, then the constant term is removed from the regression model; otherwise, it is included. Examples of the use of the function `mregg2` are given in Sections 7.7 and 7.8.

The multiple regression model has wider applications. For example:

1. *Polynomial regression.* Here y is a polynomial function of a single variable x, so that x_j in the general model is replaced by x^j. Thus we have

$$y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \cdots + \beta_p x_i^p + \varepsilon_i, \quad i = 1, 2, \ldots, n \tag{7.28}$$

   Although this is no longer linear in the explanatory variable x, it is still linear in the β_j coefficients and so the theory for the linear regression model still applies. We can use `mregg2` to carry out polynomial regression where the rows of data are x, x^2, x^3, ... and the last row of data is y. (See Examples 7.7, 7.8, and 7.9).
2. *Multiple polynomial regression.* Suppose that we wished to fit data to the following regression model:

$$y_i = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_1^2 + \beta_4 x_2^2 + \beta_5 x_1 x_2 + \varepsilon_i, \quad i = 1, 2, \ldots, n$$

   In this case the five predictors are x_1, x_2, x_1^2, x_2^2, and $x_1 x_2$. The predictors are still linear in the β_j coefficients. To use the function `mregg2` the six rows of the data array must contain the values of x_1, x_2, x_1^2, x_2^2, $x_1 x_2$, and y, respectively.

7.7 Analysis of Residuals

Besides considering the contributions made by each of the explanatory variables or predictors, it is important to consider how well the model fits at each data point and whether our assumptions about the error distribution are valid.

We recall from (7.24) that

$$\hat{\mathbf{y}} = \mathbf{H}\mathbf{y}$$

so we may write the residual vector as

$$\mathbf{e} = \mathbf{y} - \mathbf{H}\mathbf{y} = (\mathbf{I} - \mathbf{H})\,\mathbf{y}$$

It can be shown that the diagonal elements of $s^2(\mathbf{I} - \mathbf{H})$ represent the variances of the individual residuals, so the standard deviation of e_i is $s\sqrt{1 - h_{ii}}$. Since the standard deviation varies from one data point to another, it is difficult to make a direct comparison between residuals at different points. However, if we *standardize* the residuals by dividing each by its standard deviation, we obtain statistics that are similar to the t-ratios that we used for analyzing $\mathbf{b}$. Thus the standardized residual r_i is

$$r_i = \frac{e_i}{s\sqrt{1 - h_{ii}}} \quad i = 1, 2, \ldots, n$$

To distinguish this from other kinds of standardized residuals, it is sometimes called the *Studentized residual*. If the assumptions behind our model are correct, the average value

of the standardized residual should be close to zero but if any $|r_i|$ is larger than about 2 this may indicate one or more of the following:

1. The point where this occurs is an *outlier*.
2. The assumption about equal error variance at all points is incorrect.
3. There has been a fault in specifying the model.

In this context, an outlier is an observation that was not obtained under the same conditions as the others. It is not always easy to distinguish points that result from mistakes in observation from those that correspond to values of the explanatory variables that lie far away from those used for the other observations. Such points tend to have a large influence on the fitting process.

A useful statistic for measuring the influence of a particular point is the *Cook's distance*, which combines the size of the squared residual with the distance of a particular point from the mean value, called the *leverage*. This is the corresponding diagonal element of $\mathbf{H}$ determined by the values of the explanatory variables.

$$\text{Cook's distance, } d_i = \left(\frac{1}{p+1}\right)\frac{e_i^2}{s^2}\left[\frac{h_{ii}}{(1-h_{ii})^2}\right] \quad i = 1, 2, \ldots, n$$

Any point for which $d_i > 1$ will have a considerable effect on the regression and should be checked in detail. The point may have been correctly observed and provide information that is very useful in model building, but the modeler should be aware of its influence.

Example 7.5

Fit a regression model to the following data. To save space the data is given in the form required by the function `mregg2`. The first, second, and third rows of the matrices are the values of the explanatory variables x_1, x_2, and x_3, respectively, and the fourth row contains the corresponding values of y. The following script implements this.

```
% e3s710.m
X0 = [1.00 1.00 1.00 1.00 1.00 2.00 2.00 2.00;
      2.00 2.00 4.00 4.00 6.00 2.00 2.00 4.00;
         0 1.00 0 1.00 2.00 0 1.00 0;
     -2.52 -2.71 -8.34 -8.40 -14.60 -0.62 -0.47 -6.49];
X1 = [2.00 3.00 3.00 3.00 3.00 3.00 3.00 3.00;
      6.00 2.00 2.00 2.00 4.00 6.00 6.00 6.00;
         0 0 1.00 2.00 1.00 0 1.00 2.00;
    -12.46 1.36 1.40 1.60 -4.64 -10.34 -10.43 -10.30];
Xd = [X0 X1];
[s_sqd R_sqd b SE t VIF Corr_mtrx res] = mregg2(Xd,1);
```

```
fprintf('Error variance = %7.4f     R_squared = %7.4f \n\n',s_sqd,R_sqd)
fprintf('                  Coeff     SE     t_ratio     VIF \n')
fprintf('Constant   :     %7.4f %7.4f %8.2f \n',b(1),SE(1),t(1))
fprintf('Coeff x1   :     %7.4f %7.4f %8.2f %8.2f\n',b(2),SE(2),t(2),VIF(2))
fprintf('Coeff x2   :     %7.4f %7.4f %8.2f %8.2f\n',b(3),SE(3),t(3),VIF(3))
fprintf('Coeff x3   :     %7.4f %7.4f %8.2f %8.2f\n\n',b(4),SE(4),t(4),VIF(4))
fprintf('Correlation matrix \n')
disp(Corr_mtrx)
fprintf('\n        y         Residual   St Residual   Cook dist\n')
for i = 1:length(Xd)
    fprintf('%12.4f %12.4f %12.4f %12.4f\n',res(i,1), ...
                                    res(i,2), res(i,3), res(i,4))
end
```

Running this script gives

```
Error variance =  0.0147     R_squared =  0.9996

                  Coeff     SE     t_ratio     VIF
Constant   :      1.3484  0.1006    13.40
Coeff x1   :      2.0109  0.0358    56.10     1.03
Coeff x2   :     -2.9650  0.0179  -165.43     1.03
Coeff x3   :     -0.0001  0.0412    -0.00     1.04

Correlation matrix
    1.0000    0.2278   -0.9437   -0.0944
    0.2278    1.0000    0.1064    0.1459
   -0.9437    0.1064    1.0000    0.1459
   -0.0944    0.1459    0.1459    1.0000

         y         Residual   St Residual   Cook dist
   -2.5200       0.0508       0.4847       0.0196
   -2.7100      -0.1390      -1.3223       0.1426
   -8.3400       0.1609       1.5062       0.1617
   -8.4000       0.1010       0.9274       0.0507
  -14.6000      -0.1688      -1.9402       0.8832
   -0.6200      -0.0601      -0.5458       0.0157
   -0.4700       0.0901       0.8035       0.0270
   -6.4900       0.0000       0.0002       0.0000
  -12.4600      -0.0399      -0.3835       0.0130
    1.3600      -0.0909      -0.8808       0.0730
    1.4000      -0.0508      -0.4736       0.0154
    1.6000       0.1493       1.5774       0.3958
   -4.6400      -0.1607      -1.4227       0.0755
  -10.3400       0.0692       0.6985       0.0602
  -10.4300      -0.0206      -0.1930       0.0026
  -10.3000       0.1095       1.1123       0.1589
```

The coefficient of x_3 is small, and, more important, the corresponding absolute value of the t-ratio is very small (it is in fact not zero but -0.0032). This suggests that x_3 does not make a significant contribution to the model and can be removed.

If we change the last value of y to -8.3 (and showing the analysis of the residuals only), we have

```
      y         Residual   St Residual   Cook dist
  -2.5200       0.3499       0.7758       0.0503
  -2.7100      -0.0756      -0.1670       0.0023
  -8.3400       0.3095       0.6735       0.0323
  -8.4000       0.0140       0.0300       0.0001
 -14.6000      -0.6418      -1.7153       0.6903
  -0.6200       0.1361       0.2876       0.0044
  -0.4700       0.0507       0.1052       0.0005
  -6.4900       0.0457       0.0941       0.0003
 -12.4600      -0.1447      -0.3232       0.0093
   1.3600       0.0024       0.0054       0.0000
   1.4000      -0.1930      -0.4184       0.0120
   1.6000      -0.2284      -0.5610       0.0501
  -4.6400      -0.4534      -0.9331       0.0325
 -10.3400      -0.1384      -0.3247       0.0130
 -10.4300      -0.4638      -1.0077       0.0713
  -8.3000       1.4308       3.3791       1.4664
```

For the observation $y = -8.3$ we see that the residual, the standard residual, and Cook's distance are all large, compared with the values for the rest of the data. Either we have a recording error in this particular observation or the data is correct and thus the model we are using fits this particular data point poorly.

Example 7.6

Using the same data as Example 7.5, fit a regression model using the explanatory variables x_1 and x_2 only. The data array `Xd` is not shown in the following script.

```
% e3s711.m
X0 ....
X1 ....
Xd ....
Xd = [X0 X1];
[s_sqd R_sqd b SE t VIF Corr_mtrx] = mregg2(Xd([1 2 4],:),1);
fprintf('Error variance = %7.4f     R_squared = %7.4f \n\n',s_sqd,R_sqd)
fprintf('                  Coeff     SE     t_ratio     VIF \n')
fprintf('Constant   :     %7.4f %7.4f  %8.2f \n',b(1),SE(1),t(1))
fprintf('Coeff x1   :     %7.4f %7.4f  %8.2f %8.2f\n',b(2),SE(2),t(2),VIF(2))
```

```
fprintf('Coeff x2   :     %7.4f %7.4f  %8.2f %8.2f\n\n',b(3),SE(3),t(3),VIF(3))
fprintf('Correlation matrix \n')
disp(Corr_mtrx)
```

Running this script gives

```
Error variance =  0.0135     R_squared =  0.9996

                  Coeff     SE     t_ratio     VIF
Constant    :      1.3483  0.0960     14.04
Coeff x1    :      2.0109  0.0341     58.91     1.01
Coeff x2    :     -2.9650  0.0171   -173.71     1.01

Correlation matrix
    1.0000    0.2278   -0.9437
    0.2278    1.0000    0.1064
   -0.9437    0.1064    1.0000
```

This is a better model than that derived in Example 7.5 because the absolute values of the t-ratios are now all greater than 2. In fact, the original data was generated from the model

$$y = 1.5 + 2x_1 - 3x_2 + \text{random errors}$$

Thus x_3 was not linked to the model used to generate the data and variations in x_3 only appeared to influence y because of the random errors in the measurement of y.

Note that the standard errors (*SEs*) can be used to construct confidence intervals for b_j. In this case the true values of each β_j lie within the 95% confidence interval: that is, the interval within which we expect that β_j would lie with a probability of 0.95 if we did not know its true value. The precise width of the 95% confidence interval depends on the number of degrees of freedom (see Section 7.5), but to a reasonable approximation it lies between $b_j - 2SE(b_j)$ and $b_j + 2SE(b_j)$.

More comprehensive presentations of aspects of multiple regression, model improvement, and regression analysis can be found in Draper and Smith (1998), Walpole and Myers (1993), and Anderson, Sweeney and Williams (1993).

7.8 Polynomial Regression

The polynomial regression model is given by (7.28) and is repeated here:

$$y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \cdots + \beta_p x_i^p + \varepsilon_i, \quad i = 1, 2, \ldots, n$$

Although this is no longer linear in the explanatory variable x, it is still linear in the β_j coefficients and so the theory for the linear regression model still applies.

The processes of fitting, checking the b_j estimates, and deciding whether some predictors can be discarded may all be done in the same manner as in the general case described in Section 7.5, and diagnostics for model improvement and residual analysis follow the general case described in Sections 7.6 and 7.7. It will inevitably be found that there are quite high correlations between the predictors, which are now all powers of the same explanatory variable. As discussed in Section 7.6, these high correlations between predictors can result in an ill-conditioned coefficient matrix $\mathbf{X}^\top\mathbf{X}$. For a large number of data points this matrix tends to the Hilbert matrix.

In Chapter 2 it was shown that the Hilbert matrix is very ill-conditioned. To illustrate the influence of this on the accuracy of computations, we note that the number of decimal places lost when working with an ill-conditioned matrix $\mathbf{A}$ is given approximately by the MATLAB expression `log10(cond(A))`. Thus if we were fitting a fifth-degree polynomial, the number of decimal places that could be lost may be estimated by `log10(cond(hilb(5)))`. This equals 5.6782; that is, five or six of the 16 significant digits that MATLAB uses are lost. One way to avoid this difficulty is to formulate the problem so that no system of linear equations has to be solved. An ingenious way of doing this is to use orthogonal polynomials. We will not describe this method here but refer the reader to Lindfield and Penny (1989). However, the worst effects of ill-conditioning can be avoided provided that p (which now represents the degree of the fitted polynomial) is kept reasonably small.

If the diagnostics that we have developed in Sections 7.5, 7.6, and 7.7 are required, then we can use the function `mregg2` given in Section 7.6 for polynomial regression; in this case the data must be prepared as follows. The first row of the array `Xd` contains the values of x, the second row contains the values of x^2, and so on. The last row contains the corresponding values of y. In interpreting the output from `mregg2`, note that all the *VIF* values are inevitably high because the powers of x are correlated with each other. The usual rule about discarding predictors with $VIF > 10$ should be ignored since we have good reason to suppose that a polynomial model is appropriate.

If the diagnostics are not required, the calculation of the b_j estimates can be done using the MATLAB function `polyfit`. This uses the method of least squares to fit a polynomial of specified degree to given data. The following examples illustrate some of these issues.

Example 7.7

Fit a cubic polynomial to the following data, which has been generated from $y = 2 + 6x^2 - x^3$ with added random errors. The random errors have a normal distribution with a zero mean value and a standard deviation of 1. The following script calls the MATLAB function `polyfit` to determine the coefficients of the cubic polynomial followed by `polyval` to evaluate it for plotting. It then calls the function `mregg2` to compute a regression model using x, x^2, and x^3 as the explanatory variables.

```
% e3s712.m
x = 0:.25:6;
```

```
y = [1.7660 2.4778 3.6898 6.3966 6.6490 10.0451 12.9240 15.9565 ...
     17.0079 21.1964 24.1129 25.5704 28.2580 32.1292 32.4935 34.0305 ...
     34.0880 32.9739 31.8154 30.6468 26.0501 23.4531 17.6940 9.4439 ...
     1.7344];
xx = 0:.02:6;
p = polyfit(x,y,3), yy = polyval(p,xx);
plot(x,y,'o',xx,yy)
axis([0 6 0 40]), xlabel('x'), ylabel('y')
Xd = [x; x.^2; x.^3; y];
[s_sqd R_sqd b SE t VIF Corr_mtrx] = mregg2(Xd,1);
fprintf('Error variance = %7.4f     R_squared = %7.4f \n\n',s_sqd,R_sqd)
fprintf('                  Coeff      SE     t_ratio     VIF \n')
fprintf('Constant    :     %7.4f %7.4f %8.2f \n',b(1),SE(1),t(1))
fprintf('Coeff x     :     %7.4f %7.4f %8.2f %8.2f\n',b(2),SE(2),t(2),VIF(2))
fprintf('Coeff x^2   :     %7.4f %7.4f %8.2f %8.2f\n',b(3),SE(3),t(3),VIF(3))
fprintf('Coeff x^3   :     %7.4f %7.4f %8.2f %8.2f\n\n',b(4),SE(4),t(4),VIF(4))
fprintf('Correlation matrix \n')
disp(Corr_mtrx)
```

Running this script gives the following output, together with the graph shown in Figure 7.12. The polynomial coefficients from `polyfit` are in the order of descending powers of x. The diagnostic output from `mregg2` is self-explanatory.

```
p =
   -0.9855    5.8747    0.1828    2.2241

Error variance =  0.5191     R_squared =  0.9966

                  Coeff      SE     t_ratio     VIF
Constant    :      2.2241  0.4997     4.45
Coeff x     :      0.1828  0.7363     0.25    84.85
Coeff x^2   :      5.8747  0.2886    20.36   502.98
Coeff x^3   :     -0.9855  0.0316   -31.20   202.10
```

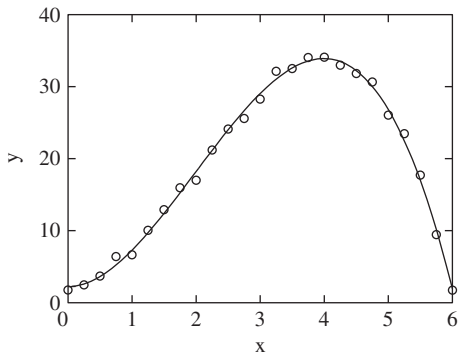

FIGURE 7.12 Fitting a cubic polynomial to data. Data points are denoted by ○.

```
Correlation matrix
    1.0000    0.4917    0.2752    0.1103
    0.4917    1.0000    0.9659    0.9128
    0.2752    0.9659    1.0000    0.9858
    0.1103    0.9128    0.9858    1.0000
```

Note that the absolute value of the t-ratio for the explanatory variable x is less than 2, indicating that x should be removed (see Example 7.8).

The cubic polynomial that fits the data is

$$\hat{y} = 2.2241 + 0.1828x + 5.8747x^2 - 0.9855x^3$$

Example 7.8

Fit a cubic polynomial to the data of Example 7.7 but use the explanatory variables x^2 and x^3 only. The following script solves this problem:

```
% e3s713.m
x = 0:.25:6; y = 2+6*x.^2-x.^3;
y = y+randn(size(x)); Xd = [x.^2; x.^3; y];
[s_sqd R_sqd b SE t VIF Corr_mtrx] = mregg2(Xd,1);
fprintf('Error variance = %7.4f     R_squared = %7.4f \n\n',s_sqd,R_sqd)
fprintf('                  Coeff     SE     t_ratio     VIF \n')
fprintf('Constant   :     %7.4f %7.4f %8.2f \n',b(1),SE(1),t(1))
fprintf('Coeff x^2  :     %7.4f %7.4f %8.2f %8.2f\n',b(2),SE(2),t(2),VIF(2))
fprintf('Coeff x^3  :     %7.4f %7.4f %8.2f %8.2f\n\n',b(3),SE(3),t(3),VIF(3))
fprintf('Correlation matrix \n')
disp(Corr_mtrx)
```

Running this script gives

```
Error variance =  0.4970     R_squared =  0.9965

                  Coeff     SE     t_ratio     VIF
Constant   :      2.3269  0.2741     8.49
Coeff x^2  :      5.9438  0.0750    79.21    35.52
Coeff x^3  :     -0.9926  0.0130   -76.61    35.52

Correlation matrix
    1.0000    0.2752    0.1103
    0.2752    1.0000    0.9858
    0.1103    0.9858    1.0000
```

Thus our improved model (compared with Example 7.7) is

$$\hat{y} = 2.1793 + 6.0210x^2 - 1.0084x^3$$

The true β_j are well within the 95% confidence limits given by this improved model. The error variance of 0.4970 is somewhat less than the unit variance of the random errors initially added to the data.

Example 7.9

Fit a third- and a fifth-degree polynomial to data generated from the function

$$y = \sin\{1/(x+0.2)\} + 0.2x$$

contaminated with random noise, normally distributed with a standard deviation of 0.06 to simulate measurement errors as follows:

```
>> xs = [0:0.05:0.25 0.25:0.2:4.85];
>> us = sin(1./(xs+1))+0.2*xs+0.06*randn(size(xs));
>> save testdata1 xs us
```

The 30 data values are stored in the file `testdata1` so that it can be used in an example in Section 7.9. The following script loads the data, and fits and plots the least squares polynomial.

```
% e3s714.m
load testdata
xx = 0:.05:5;
t1 = 'Error variance = %7.4f     R_squared = %7.4f \n\n';
t2 = '                  Coeff     SE     t_ratio       VIF \n';
t3 = 'Constant   :     %7.4f %7.4f %8.2f \n';
t4 = 'Coeff x    :     %7.4f %7.4f %8.2f %12.2f\n';
t4a = 'Coeff x    :     %7.4f %7.4f %8.2f \n';
t5 = 'Coeff x^2  :     %7.4f %7.4f %8.2f %12.2f\n';
t6 = 'Coeff x^3  :     %7.4f %7.4f %8.2f %12.2f\n';
t7 = 'Coeff x^4  :     %7.4f %7.4f %8.2f %12.2f\n';
t8 = 'Coeff x^5  :     %7.4f %7.4f %8.2f %12.2f\n';
t9 = 'Correlation matrix \n';
p = polyfit(xs,us,3), yy = polyval(p,xx);
Xd = [xs; xs.^2; xs.^3; us];
[s_sqd R_sqd b SE t VIF Corr_mtrx] = mregg2(Xd,1);
fprintf(t1,s_sqd,R_sqd), fprintf(t2)
fprintf(t3,b(1),SE(1),t(1))
fprintf(t4,b(2),SE(2),t(2),VIF(2))
fprintf(t5,b(3),SE(3),t(3),VIF(3))
fprintf([t6 '\n'],b(4),SE(4),t(4),VIF(4))
fprintf(t9), disp(Corr_mtrx)
```

```
[s_sqd R_sqd b SE t VIF Corr_mtrx] = mregg2(Xd,0);
fprintf(t1,s_sqd,R_sqd), fprintf(t2)
fprintf(t4a,b(1),SE(1),t(1))
fprintf(t5,b(2),SE(2),t(2),VIF(2))
fprintf([t6 '\n'],b(3),SE(3),t(3),VIF(3))
fprintf(t9), disp(Corr_mtrx)
plot(xs,us,'ko',xx,yy,'k'), hold on
axis([0 5 -2 2])
p = polyfit(xs,us,5), yy = polyval(p,xx);
Xd = [xs; xs.^2; xs.^3; xs.^4; xs.^5; us];
[s_sqd R_sqd b SE t VIF Corr_mtrx] = mregg2(Xd,1);
fprintf(t1,s_sqd,R_sqd), fprintf(t2)
fprintf(t3,b(1),SE(1),t(1))
fprintf(t4,b(2),SE(2),t(2),VIF(2))
fprintf(t5,b(3),SE(3),t(3),VIF(3))
fprintf(t6,b(4),SE(4),t(4),VIF(4))
fprintf(t7,b(5),SE(5),t(5),VIF(5))
fprintf([t8 '\n'],b(6),SE(6),t(6),VIF(6))
fprintf(t9)
disp(Corr_mtrx)
plot(xx,yy,'k--'), xlabel('x'), ylabel('y'), hold off
```

Figure 7.13 shows the result of fitting a third- and a fifth-degree polynomial to the data and clearly displays the inadequacies of these polynomial approximations. The polynomials oscillate about the points and do not fit the data satisfactorily. In Section 7.9 we see that we can improve the fit by using different functions. The output from the script is as follows:

```
p =
    0.0842   -0.6619    1.5324   -0.0448

Error variance =  0.0980     R_squared =  0.6215
```

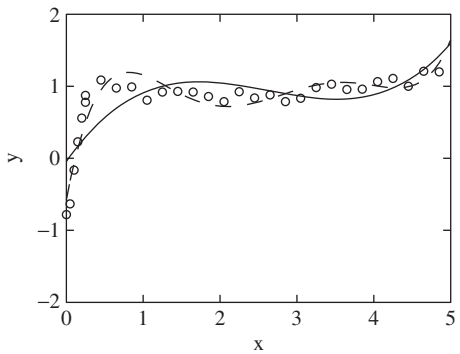

FIGURE 7.13 Fitting third- and fifth-degree polynomials (shown by a full line and a dashed line, respectively) to a sequence of data. Data points are denoted by ○.

```
                  Coeff     SE     t_ratio       VIF
Constant    :     -0.0448  0.1402    -0.32
Coeff x     :      1.5324  0.3248     4.72        79.98
Coeff x^2   :     -0.6619  0.1708    -3.87       478.23
Coeff x^3   :      0.0842  0.0239     3.52       193.93

Correlation matrix
    1.0000    0.5966    0.4950    0.4476
    0.5966    1.0000    0.9626    0.9049
    0.4950    0.9626    1.0000    0.9847
    0.4476    0.9049    0.9847    1.0000
```

The absolute value of the t-ratio for the constant terms is low, implying that it should not be included in the cubic model. The script also fits a cubic equation without a constant term to the data and gives the following output:

```
Error variance =  0.0947     R_squared =  0.6200

                  Coeff     SE     t_ratio       VIF
Coeff x     :      1.4546  0.2116     6.87
Coeff x^2   :     -0.6285  0.1329    -4.73        35.13
Coeff x^3   :      0.0801  0.0199     4.02       299.50

Correlation matrix
    1.0000    0.5966    0.4950    0.4476
    0.5966    1.0000    0.9626    0.9049
    0.4950    0.9626    1.0000    0.9847
    0.4476    0.9049    0.9847    1.0000
```

This is a more robust model. Finally, the script fits a fifth-degree polynomial to the data and gives the following, final, output:

```
p =
    0.0434   -0.5856    2.8998   -6.3340    5.7099   -0.5789

Error variance =  0.0341     R_squared =  0.8783

                  Coeff     SE     t_ratio       VIF
Constant    :     -0.5789  0.1122    -5.16
Coeff x     :      5.7099  0.6443     8.86       904.01
Coeff x^2   :     -6.3340  0.9052    -7.00     38560.71
Coeff x^3   :      2.8998  0.4918     5.90    234903.50
Coeff x^4   :     -0.5856  0.1137    -5.15    262672.06
Coeff x^5   :      0.0434  0.0094     4.62     38084.24
```

```
Correlation matrix
    1.0000    0.5966    0.4950    0.4476    0.4172    0.3942
    0.5966    1.0000    0.9626    0.9049    0.8511    0.8041
    0.4950    0.9626    1.0000    0.9847    0.9555    0.9232
    0.4476    0.9049    0.9847    1.0000    0.9918    0.9742
    0.4172    0.8511    0.9555    0.9918    1.0000    0.9949
    0.3942    0.8041    0.9232    0.9742    0.9949    1.0000
```

Note the large values of *VIF*, caused by the regression being based on different functions of a single explanatory variable.

We now illustrate a difficulty that can sometime arise when trying to fit a polynomial function to data. To illustrate the problem we will begin by simulating experimental data based on the following relationship:

$$y = \frac{1}{\sqrt{0.02 + \left(4 - x^2\right)^2}} \tag{7.29}$$

Data values are generated by sampling this function from $x = 1$ to $x = 3$ in increments of 0.05, and small random errors are added to simulate measurement errors. The results of attempting to fit polynomials to these data values are shown in Figure 7.14. The plot shows that as the degree of the polynomial is increased from 4 to 8, and finally to 12, the polynomial fits the data better in the sense that the total least squares error decreases, but the higher-degree polynomials tend to oscillate between the data points. Thus even a twelfth-degree polynomial does not accurately represent the data, nor does it give us any insight into the underlying mathematical relationship between x and y. We return to this problem in Section 7.11.

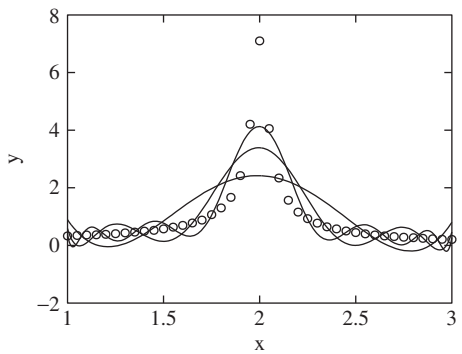

FIGURE 7.14 Polynomials of degree 4, 8, and 12 attempting to fit a sequence of data indicated by ○ in the graph.

7.9 Fitting General Functions to Data

We now consider a regression based on (7.17) with the separate explanatory variables x_j replaced by predictors that are various functions ϕ_j of a single explanatory variable x:

$$y_i = \beta_0 + \beta_1\varphi_1(x_i) + \beta_2\varphi_2(x_i) + \cdots + \beta_p\varphi_p(x_i) + \varepsilon_i$$

The analysis presented in Section 7.5 extends directly to this regression model. Hence we can use the MATLAB function `mregg2` to fit a set of any prescribed functions to data.

Consider again Example 7.9 in Section 7.8. We will fit the following function (or model) to the data:

$$\hat{y} = b_1 \sin\{1/(x+0.2)\} + b_2 x$$

This function has been chosen because the data was originally generated from it with $b_1 = 1$ and $b_2 = 0.2$ with normally distributed random noise added. Note that there is no constant term in our model. The following script calls the function `mregg2`. Note how the first row of the data matrix for `mregg2` contains the values of $\sin(1/(x+0.2))$ and the second row contains the values of x.

```
% e3s715.m
load testdata
Xd = [sin(1./(xs+0.2)); xs; us];
[s_sqd R_sqd b SE t VIF Corr_mtrx] = mregg2(Xd,0);
fprintf('Error variance = %7.4f\n\n',s_sqd)
fprintf('                      Coeff     SE     t_ratio\n')
fprintf('sin(1/(x+0.2)):     %7.4f %7.4f %8.2f \n',b(1),SE(1),t(1))
fprintf('Coeff x       :     %7.4f %7.4f %8.2f \n\n',b(2),SE(2),t(2))
fprintf('Correlation matrix \n')
disp(Corr_mtrx)
xx = 0:.05:5; yy = b(1)*sin(1./(xx+0.2))+b(2)*xx;
plot(xs,us,'o',xx,yy,'k')
axis([0 5 -1.5 1.5]), xlabel('x'), ylabel('y')
```

Running this script gives

```
Error variance =  0.0044

                     Coeff     SE     t_ratio
sin(1/(x+0.2)):      0.9354  0.0257    36.46
Coeff x       :      0.2060  0.0053    38.55

Correlation matrix
    1.0000    0.7461    0.5966
    0.7461    1.0000   -0.0734
    0.5966   -0.0734    1.0000
```

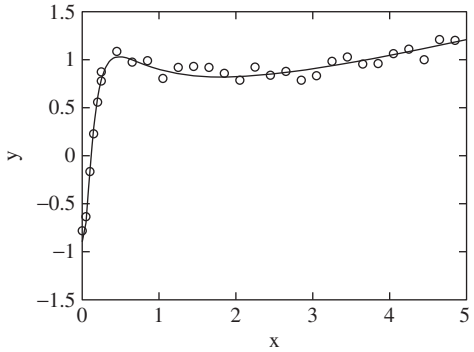

FIGURE 7.15 Data sampled from the function $y = \sin[1/(x+0.2)] + 0.2x$. Data points denoted by "$\circ$".

and the graph of Figure 7.15. The function that fits the data in a least squares sense is given by $\hat{y} = 0.9354 \sin\{1/(x+0.2)\} + 0.2060x$. This is very close to the original function. Note also that the error variance of 0.0044 compares quite well with 0.0036, the variance of the noise that was added to simulate measurement errors. If a constant term is included in the model, it has a very low absolute t-ratio, indicating that it should be removed.

7.10 Nonlinear Least Squares Regression

We now consider the problem of fitting data to a function where the relationship between the unknown coefficients is nonlinear. We still use the least squares criterion, and there are several methods for fitting data to this type of model. Here we present a very simple iterative method based on a Taylor series.

Let $y = f(x, \mathbf{a})$ where f is a nonlinear function of the unknown coefficients $\mathbf{a}$. To determine these coefficients, let a trial set of coefficents be $\mathbf{a}^{(0)}$. Thus

$$y = f\left(x, \mathbf{a}^{(0)}\right)$$

This trial solution will not satisfy the requirement that the sum of the squares of the errors is a minimum. However, we can adjust the coefficients $\mathbf{a}^{(0)}$ in order to minimize the sum of the squares of the errors. Thus, letting the improved coefficients be $\mathbf{a}^{(1)}$, where

$$\mathbf{a}^{(1)} = \mathbf{a}^{(0)} + \Delta\mathbf{a}$$

we have

$$y = f\left(x, \mathbf{a}^{(1)}\right) = f\left(x, \mathbf{a}^{(0)} + \Delta\mathbf{a}\right)$$

Expanding the function as a Taylor series, and retaining only the first derivative terms in the Taylor series we have

$$y \approx f\left(x, \mathbf{a}^{(0)}\right) + \sum_{k=0}^{m} \Delta a_k \left[\partial f / \partial a_k\right]^{(0)}$$

Let $f_i^{(0)} = f\left(x_i, \mathbf{a}^{(0)}\right)$. The error between the function and y_i is given by

$$\varepsilon_i = y_i - f_i^{(0)} - \sum_{k=0}^{m} \Delta a_k \left[\partial f_i / \partial a_k\right]^{(0)}, \quad i = 1, 2, \ldots, n$$

Thus, the sum of the squares of the errors is

$$S = \sum_{i=0}^{n} \left\{ y_i - f_i^{(0)} - \sum_{k=0}^{m} \Delta a_k \left[\frac{\partial f_i}{\partial a_k}\right]^{(0)} \right\}^2$$

To determine the minimum of the sum of the squares of the errors, we have

$$\frac{\partial S}{\partial(\Delta a_p)} = -2 \sum_{i=0}^{n} \left\{ y - f_i^{(0)} - \sum_{k=0}^{m} \Delta a_k \left[\frac{\partial f_i}{\partial a_k}\right]^{(0)} \right\} \left[\frac{\partial f_i}{\partial a_p}\right]^{(0)} = 0, \quad p = 0, 1, \ldots, m$$

Rearranging we have

$$\sum_{i=0}^{n} \left(y - f_i^{(0)}\right) \left[\frac{\partial f_i}{\partial a_p}\right]^{(0)} = \sum_{k=0}^{m} \Delta a_k \left\{ \sum_{i=0}^{n} \left[\frac{\partial f_i}{\partial a_k}\right]^{(0)} \left[\frac{\partial f_i}{\partial a_p}\right]^{(0)} \right\}, \quad p = 0, 1, \ldots, m$$

These equations can be expressed in matrix notation as follows:

$$\mathbf{K}(\Delta \mathbf{a}) = \mathbf{b}$$

where $\Delta \mathbf{a}$ has elements $\Delta a_p, p = 0, 1, \ldots, m$ and

$$K_{pk} = \sum_{i=0}^{n} \left[\frac{\partial f_i}{\partial a_k}\right]^{(0)} \left[\frac{\partial f_i}{\partial a_p}\right]^{(0)}$$

$$b_p = \sum_{i=0}^{n} \left(y - f_i^{(0)}\right) \left[\frac{\partial f_i}{\partial a_p}\right]^{(0)}, \quad p, k = 0, 1, \ldots, m$$

Solving for $\Delta \mathbf{a}$ we can then determine new values for the coefficients from

$$\mathbf{a}^{(1)} = \mathbf{a}^{(0)} + \Delta \mathbf{a}$$

Because we have discarded higher-order terms in the Taylor series, $\mathbf{a}^{(1)}$ is not the exact solution, but it is a better solution than $\mathbf{a}^{(0)}$. We therefore iterate until the norm of $\Delta\mathbf{a}$ is less than a prescribed tolerance.

The following function `nlls` implements the preceding method for fitting a given nonlinear function to a set of data.

```
function [a iter] = nlls(f,df,x,y,a0,err)
% Data given by vectors x and y are to be fitted to the function f(a)
% with an error of err. Function f(a) has n variables, a(1) ... a(n).
% a0 is a vector of n trial values for the unknown paramenters a.
% Function df is a column vector [df/da(1); df/da(2); .... df/da(n)].
iter = 0;  n = length(a0); a = a0;
v = 10*err*ones(1,n);
while norm(v,2) > err
    p = feval(df,x,a);  q = y-feval(f,x,a);
    A = p*p'; b = q*p'; v = A\b';
    a = a + v';  iter = iter+1;
end
```

The next script fits the function $y = a_1 e^{a_2 x} + a_3 e^{a_4 x}$ to 16 data points using the function `nlls`.

```
% e3s718
p = @(x,a) a(1)*exp(a(2)*x)+a(3)*exp(a(4)*x);
dp = @(x,a) [exp(a(2)*x); a(1)*x.*exp(a(2)*x);
             exp(a(4)*x); a(3)*x.*exp(a(4)*x)];
x = [-10:2:0  1:1:10]; xn = length(x);
xp = -10:0.05:10;
y = [26.56 21.60 18.14 17.00 14.46 17.38 15.07 16.76 ...
    16.90 17.32 18.61 20.79 21.65 25.22 26.16 27.84];
a = [7 -0.3 7 0.3];
[a iter] = nlls(p,dp,x,y,a,1e-5)
plot(x,y,'o',xp,p(xp,a))
xlabel('x'), ylabel('y')
```

Running this script gives the following results:

```
a =
    5.4824   -0.1424   10.0343    0.0991

iter =
     7
```

together with Figure 7.16.

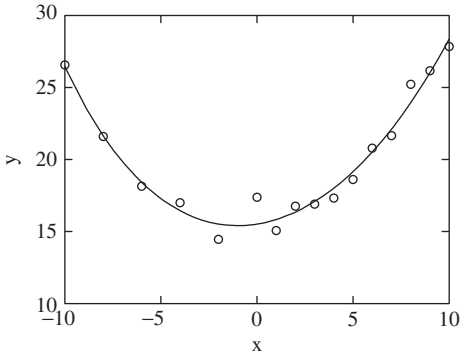

FIGURE 7.16 Fitting $y = a_1 e^{a_2 x} + a_3 e^{a_4 x}$ to data values indicated by "$\circ$".

7.11 Transforming Data

We now consider an alternative approach to the problem of fitting data to functions where the relationship between the unknown coefficients is nonlinear. This is to transform both the data and the function so that an equivalent function provides a linear relationship between y and the unknown coefficients. The only difficulty with this method is that no general rule can be given to provide a suitable transform; indeed, such a transform may not even be possible. Consider the problem of fitting data to the following function:

$$\hat{y} = \frac{1}{\sqrt{a_0 + (a_1 - a_2 x^2)^2}} \tag{7.30}$$

Letting $\hat{Y} = 1/\hat{y}^2$ and $X = x^2$, we have

$$\begin{aligned} \hat{Y} &= 1/\hat{y}^2 = a_0 + (a_1 - a_2 x^2)^2 = (a_0 + a_1^2) - 2a_1 a_2 x^2 + a_2^2 x^4 \\ \hat{Y} &= a_0 + a_1^2 - 2a_1 a_2 X + a_2^2 X^2 \\ \hat{Y} &= b_0 + b_1 X + b_2 X^2 \end{aligned} \tag{7.31}$$

Thus $\hat{Y}$ is a quadratic in X. If the data values are transformed by letting $Y_i = 1/y_i^2$ and $X_i = x_i^2$, then the process of fitting $Y = f(X)$ to these transformed data values will be a standard least squares polynomial fit giving b_0, b_1, and b_2. Hence estimates of a_0, a_1, and a_2 can be easily determined. Note, however, that the residual of the errors, $e_i = Y_i - \hat{Y}_i$, will not provide a good estimate of the measurement errors, $y_i - \hat{y}_i$, because of the transformations.

We illustrate the preceding process by considering a sequence of data values related by (7.29) to which we have added normally distributed random errors having a zero mean value and a standard deviation of 1%. We can transform these data points using (7.31), and the following script generates the required data, transforms the data points, and fits a polynomial to them.

```
% e3s716.m
x = 1:.05:3; xx = 1:.005:3;
```

```
y = [0.3319   0.3454   0.3614   0.3710   0.3857   0.4030   0.4372 ...
     0.4605   0.4971   0.5232   0.5753   0.6363   0.6953   0.7782 ...
     0.8793   1.0678   1.3024   1.6688   2.4233   4.2046   7.0961 ...
     4.0581   2.3354   1.5663   1.1583   0.9278   0.7764   0.6480 ...
     0.5741   0.4994   0.4441   0.4005   0.3616   0.3286   0.3051 ...
     0.2841   0.2645   0.2407   0.2285   0.2104   0.2025];
Y = 1./y.^2; X = x.^2; XX = xx.^2;
p = polyfit(X,Y,2)
YY = polyval(p,XX);
for i = 1:length(xx)
    if YY(i)<0
        disp('Transformation fails with this data set');
        return
    end
end
figure(1), plot(X,Y,'o',XX,YY)
axis([1 9 0 25]), xlabel('X'), ylabel('Y')
yy = 1./sqrt(YY);
figure(2), plot(x,y,'o',xx,yy)
axis([1 3 -2 8]), xlabel('x'), ylabel('y')
```

Running this script gives the following results:

```
p =
    0.9944   -7.9638   15.9688
```

together with the plots shown in Figures 7.17 and 7.18. From the output of the script we see that the relationship between X and $\hat{Y}$ is

$$\hat{Y} = 0.9944X^2 - 7.9638X + 15.9688$$

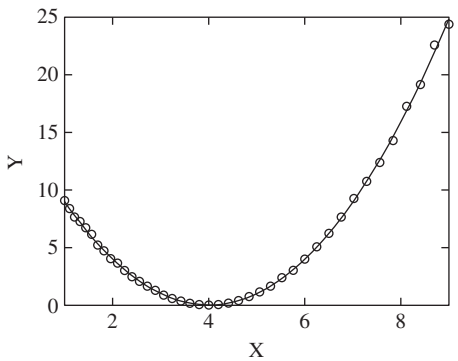

FIGURE 7.17 Fitting transformed data denoted by "○" to a quadratic function.

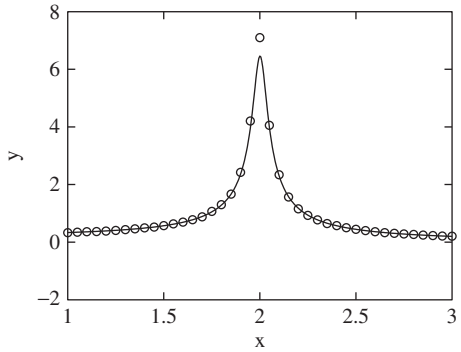

FIGURE 7.18 Fitting (7.30) to the given data denoted by ○.

We can deduce the values of the unknown coefficients by comparing the preceding equation with (7.31) to give

$$a_2^2 = 0.9944, \quad \text{hence} \quad a_2 = \pm 0.9972 \quad \text{(take positive value as in (7.29))}$$
$$-2a_1a_2 = 7.9748, \quad \text{hence } a_1 = 3.9931$$
$$a_1^2 + a_0 = 15.9688, \quad \text{hence } a_0 = 0.0149$$

We use these values in the original function (7.30) and fit it to the given data. This is shown in Figure 7.18. This function provides a much better fit than the polynomials shown earlier in Figure 7.14. However, even this fit does not pass through the peak value. This is caused by the sensitivity of the process to small random errors in the data. If the random errors are removed, the fit is exact. If the script is rerun with the random errors, the fit may be worse, and if the size of the random errors is increased, the process may fail. This is because in the region of $x = 2$, the value of y is essentially only dependent on a_0. This has a small value that may vary in sign.

Table 7.3 lists some functions with a nonlinear relationship between $\hat{y}$ and the coefficients of the function and the corresponding transformations that linearize these relationships so they have the form $\hat{Y} = BX + C$.

The following script implements the first two relationships shown in Table 7.3. It also determines the sum of the squares of the errors for the two original relationships and plots a graph of the data and the two fitted functions.

```
% e3s717.m
x = 0.2:0.2:4;
y = 2*exp(0.5*x).*(1+0.2*rand(size(x)));
X = log(x);  Y = log(y);
```

Table 7.3 Functions with Nonlinear Relationships

Original equation	Substitution	Transformed equation
$y = ax^b$	$Y = \log_e(y), X = \log_e(x)$	$Y = A + bX$ so that $a = e^A$
$y = ae^{bx}$	$Y = \log_e(y)$	$Y = A + bx$ so that $a = e^A$
$y = axe^{bx}$	$Y = \log_e(y/x)$	$Y = A + bx$ so that $a = e^A$
$y = a + \log_e(bx)$	$Y = e^y$	$Y = A + bx$ so that $a = \log_e(A)$
$y = 1/(a + bx)$	$Y = 1/y$	$Y = a + bx$
$y = 1/(a + bx)^2$	$Y = 1/\sqrt{(y)}, X = 1/x$	$Y = a + bx$
$y = x/(b + ax)$	$Y = 1/y, X = 1/x$	$Y = a + bX$
$y = ax/(b + x)$	$Y = 1/y, X = 1/x$	$Y = A + BX$ so that $a = 1/A$, $b = B/A$

```
% Case 1: Fit y = a*x^b
v = polyfit(X,Y,1);
A1 = v(2); b1 = v(1); a1 = exp(A1);
e1 = y-a1*x.^b1; s1 = e1*e1';
fprintf('\n y = %8.4f*x^(%8.4f): SSE = %8.4f',a1,b1,s1)
% case 2: Fit y = a*exp(b*x)
v = polyfit(x,Y,1);
A2 = v(2); b2 = v(1); a2 = exp(A2);
e2 = y-a2*exp(b2*x); s2 = e2*e2';
fprintf('\n y = %8.4f*exp(%8.4f*x): SSE = %8.4f \n',a2,b2,s2)
% Plotting
n = length(x);
r = x(n)-x(1); inc = r/100;
xp = [x(1):inc:x(n)];
yp1 = a1*xp.^b1;  yp2 = a2*exp(b2*xp);
plot(x,y,'ko',xp,yp1,'k:',xp,yp2,'k')
xlabel('x'), ylabel('f(x)')
```

Running this script gives

```
y =   4.5129*x^(  0.6736): SSE =  78.3290
y =   2.2129*exp(  0.5021*x): SSE =   2.0649
```

Figure 7.19 and the preceding output confirms that the best fit is, as one would expect, the exponential function. Clearly this is a warning to the user to use discrimination in selecting the function to fit. This MATLAB function could be adapted to fit data to a wide range of mathematical functions.

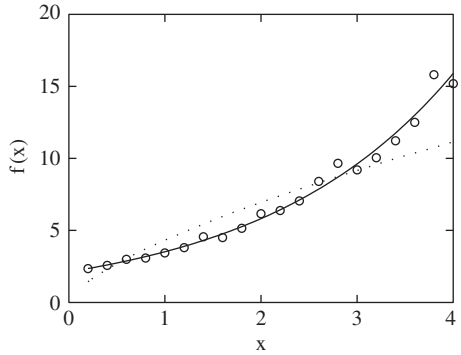

FIGURE 7.19 This graph shows the original data and the fits obtained from $y = be^{(ax)}$ (*full line*) and $y = ax^b$ (*dotted line*).

7.12 Summary

Methods have been described for fitting functions to data for the purposes of interpolation. These have included Aitken's method and spline fits. For periodic data, we have examined the fast Fourier transform. Finally, we have discussed least squares approximations to experimental data using polynomial and more general functions.

The reader who wishes to study the application of splines may find the Mathworks Spline toolbox useful.

Problems

7.1. The following tabulation gives values of the complete elliptic integral

$$E(\alpha) = \int_0^{\pi/2} \sqrt{(1 - \sin^2\alpha \sin^2\theta)}\, d\theta$$

α	0°	5°	10°	15°	20°	25°	30°
$E(\alpha)$	1.57079	1.56780	1.55888	1.54415	1.52379	1.49811	1.46746

Determine $E(\alpha)$ for $\alpha = 2°$, $13°$, and $27°$ using the MATLAB function `aitken`.

7.2. Generate a table of values of $f(x) = x^{1.4} - \sqrt{x} + 1/x - 100$ for $x = 20 : 2 : 30$. Find the value of x corresponding to $f(x) = 0$ using the MATLAB function `aitken`. This is an example of inverse interpolation since we are finding the value of x corresponding to a given value of $f(x)$. In particular, this gives an approximation to the root of the equation $f(x) = 0$. Compare your solution with that of Problem 3.2 in Chapter 3.

7.3. Given $x = -1 : 0.2 : 1$, calculate values of y from $y = \sin^2(\pi x/2)$. Using the data you have, calculate:

(a) Generate quadratic and quartic polynomials to fit this data using the least squares MATLAB function `polyfit`. Display the data and the curve fitted. *Hint*: Example 7.6 in Section 7.7 gives some guidance.

(b) Fit a cubic spline to the data using the MATLAB function `spline`. Display the data and the fitted `spline`. Compare the quality of this spline fit with the two graphs from (a).

7.4. For the data of Problem 7.3, determine the values of y for $x = 0.85$ using the MATLAB function `interp1` for a linear, spline, and cubic interpolating function. Also use the MATLAB function `aitken`.

7.5. Fit a cubic spline and a fifth-degree polynomial to the following data.

x	-2	0	2	3	4	5
y	4	0	-4	-30	-40	-50

Plot the data points, the spline, and the polynomial on the same graph. Which curve appears to give the more realistic representation of any underlying function from which the data might have been taken?

7.6. For the data given by the vectors $x = 0 : 0.25 : 3$ and

$$y = [6.3806\ \ 7.1338\ \ 9.1662\ \ 11.5545\ \ 15.6414\ \ 22.7371\ \ 32.0696 \ldots \\ 47.0756\ \ 73.1596\ \ 111.4684\ \ 175.9895\ \ 278.5550\ \ 446.4441]$$

fit the following functions:

(a) $f(x) = a + be^x + ce^{2x}$ using the MATLAB function `mregg2`

(b) $f(x) = a + b/(1+x) + c/(1+x)^2$ using the MATLAB function `mregg2`

(c) $f(x) = a + bx + cx^2 + dx^3$ using the MATLAB functions `polyfit` or `mregg2`

You should plot the three trial functions and the data. How well do these functions fit the data? The data values were in fact generated from $f(x) = 3 + 2e^x + e^{2x}$ with a small amount of random noise added.

7.7. The following values of x and corresponding values of y_u and y_l define an airfoil section:

$$x = [0\ \ 0.005\ \ 0.0075\ \ 0.0125\ \ 0.025\ \ 0.05\ \ 0.1\ \ 0.2\ \ 0.3\ \ 0.4 \ldots 0.5\ \ 0.6\ \ 0.7\ \ 0.8\ \ 0.9\ \ 1]$$

$$y_u = [0\ \ \ 0.0102\ \ \ 0.0134\ \ \ 0.0170\ \ \ 0.0250\ \ \ 0.0376\ \ \ 0.0563\ \ \ 0.0812 \ldots \\ 0.0962\ \ 0.1035\ \ 0.1033\ \ 0.0950\ \ 0.0802\ \ 0.0597\ \ 0.0340\ \ 0]$$

$$y_l = [0\ \ \ -0.0052\ \ \ -0.0064\ \ \ -0.0063\ \ \ -0.0064\ \ \ -0.0060\ \ \ -0.0045 \ldots \\ -0.0016\ \ 0.0010\ \ 0.0036\ \ 0.0070\ \ 0.0121\ \ 0.0170\ \ 0.0199\ \ 0.0178\ \ 0]$$

The (x, y_u) coordinates define the upper surface and the (x, y_l) coordinates define the lower surface. Use the MATLAB function `spline` to fit separate splines to the upper and lower surfaces and plot the results as a single figure.

7.8. Consider the approximation

$$\prod_{p<P}\left(1+\frac{1}{p}\right) \approx C_1 + C_2 \log_e P$$

where the product is taken of all the prime numbers p less than a prime number P. Write a script to generate these products from the list of prime numbers provided and fit the function $C_1 + C_2 \log_e P$ to the points given by the primes P and the corresponding values of the products using the MATLAB function `polyfit`. Generate a list of prime numbers using the MATLAB function `primes(103)`.

7.9. The gamma function may be approximated by a fifth-degree polynomial as

$$\Gamma(x+1) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4 + a_5 x^5$$

Use the MATLAB function `gamma` to generate values of $\Gamma(x+1)$ for $x = 0 : 0.1 : 1$. Then, using the MATLAB function `polyfit`, fit a fifth-degree polynomial to this data. Compare your answers with the approximation for the gamma function given by Abramowitz and Stegun (1965), which gives $a_0 = 1$, $a_1 = -0.5748666$, $a_2 = 0.9512363$, $a_3 = -0.6998588$, $a_4 = 0.4245549$, and $a_5 = -0.1010678$. These coefficients give an accuracy for the gamma function in the range $0 \le x \le 1$ of less than or equal to 5×10^{-5}.

7.10. Generate a table of values of z from the function

$$z(x,y) = 0.5\left(x^4 - 16x^2 + 5x\right) + 0.5\left(y^4 - 16y^2 + 5y\right)$$

in the range $x = -4 : 0.2 : 4$ and $y = -4 : 0.2 : 4$. Use this data and the MATLAB function `interp2` to interpolate a value for z at $x = y = -2.9035$. Use both linear and cubic interpolation and check your answer by direct substitution in the function. This point gives the global minimum of this function.

7.11. The difference between the mean Sun and the real Sun is called the equation of time. Thus the value of the equation of time

$$E = (\text{mean Sun time} - \text{real Sun time})$$

The following values represent E in minutes at 20 equispaced intervals during the year, beginning January 1.

$$\begin{aligned} E = [&-3.5 \quad -10.5 \quad -14.0 \quad -14.25 \quad -9.0 \quad -4.0 \quad 1.0 \quad 3.5 \quad 3.0 \ldots \\ &-0.25 \quad -3.5 \quad -6.25 \quad -5.5 \quad -1.75 \quad 4.0 \quad 10.5 \quad 15.0 \quad 16.25 \quad 12.75 \quad 6.5] \end{aligned}$$

Plot a graph of the data values E against time of year. Then use the function `interpft` to interpolate 300 points and plot E over a period of one year. (Use the MATLAB `help` function to obtain information on `interpft`). Finally, use the command `[x,y]=ginput(4)` to read from the graph the values of the two minimum and two maximum values of E. At what times do these maxima and minima occur?

7.12. Determine the real and imaginary parts of the DFT, using the MATLAB function `fft`, for the following periodic data where the 32 data points are sampled at intervals of 0.1 second. Examine the amplitude and frequency of its components. What conclusions can you draw from these results?

$$\begin{aligned} y = [&2 \ -0.404 \ 0.2346 \ 2.6687 \ -1.4142 \ -1.0973 \ 0.8478 \ -2.37 \ 0 \ \ldots \\ &2.37 \ -0.8478 \ 1.0973 \ 1.4142 \ -2.6687 \ -0.2346 \ 0.404 \ -2 \ \ldots \\ &1.8182 \ 1.7654 \ -1.2545 \ 1.4142 \ -0.3169 \ -2.8478 \ 0.9558 \ \ldots \\ &0 \ -0.9558 \ 2.8478 \ 0.3169 \ -1.4142 \ 1.2545 \ -1.7654 \ -1.8182] \end{aligned}$$

7.13. Determine the DFT of $y = 32\sin^5(2\pi ft)$ where $f = 30$ Hz. Use 512 points sampled over 1 second. From the imaginary part of the DFT, estimate the coefficients a_0, a_1, a_2 in the relationship

$$32\sin^5(2\pi ft) = a_0\sin[2\pi ft] + a_1\sin[2\pi(3f)t] + a_2\sin[2\pi(5f)t]$$

Repeat the process for $y = 32\sin^6(2\pi ft)$ where $f = 30$ Hz. Use 512 points sampled over 1 second. From the real part of the DFT, estimate the coefficients b_0, b_1, b_2, b_3 in the relationship

$$32\sin^6(2\pi ft) = b_0 + b_1\cos[2\pi(2f)t] + b_2\cos[2\pi(4f)t] + b_3\cos[2\pi(6f)t]$$

7.14. Determine the DFT of a set of 512 data points sampled over a 1 second period from

$$y = \sin(2\pi f_1 t) + 2\sin(2\pi f_2 t)$$

where $f_1 = 30$ Hz and $f_2 = 400$ Hz. Explain why there is a large component in the spectrum at 112 Hz.

7.15. Determine the DFT of a set of 256 data points sampled over 1 second from $y(t) = \sin(2\pi ft)$ for $f = 25$, 30.27, and 35.49 Hz. Plot the absolute value of the DFT against frequency for all three values of f in the same figure. It will be noted that even though the amplitude of the sine function from which the samples are taken is the same in each case, the frequency components corresponding to f have different amplitudes. This is because, in the case of the 30.27 Hz and 35.49 Hz waves the sampling is not over an integer number of periods of y. This phenomenon is known as "leakage" or "smearing," and part of the pure sine wave seems to have smeared into adjacent frequencies. Its effect may be reduced by applying a "window" to the data. The Hanning window is $w(t) = 0.5\{1 - \cos(2\pi t/T)\}$ where T is the sampling

period. Multiply $y(t)$ by $w(t)$ and determine the DFT of the resulting data. Plot the absolute value of this DFT against frequency for all three values of f in the same figure. Note that the amplitude variation of the frequency components corresponding to f and the smearing into other frequencies has been reduced significantly.

7.16. The following 32 data points are sampled over a period of 0.0625 second.

$$
\begin{aligned}
y = [&0\ \ 0.9094\ \ 0.4251\ \ -0.6030\ \ -0.6567\ \ 0.2247\ \ 0.6840\ \ 0.1217\ \ldots\\
&-0.5462\ \ -0.3626\ \ 0.3120\ \ 0.4655\ \ -0.0575\ \ -0.4373\ \ -0.1537\ \ldots\\
&0.3137\ \ 0.2822\ \ -0.1446\ \ -0.3164\ \ -0.0204\ \ 0.2694\ \ 0.1439\ \ -0.1702\ \ldots\\
&-0.2065\ \ 0.0536\ \ 0.2071\ \ 0.0496\ \ -0.1594\ \ -0.1182\ \ 0.0853\ \ldots\\
&0.1441\ -0.0078]
\end{aligned}
$$

(a) Determine the DFT and estimate the frequency of the most significant component present in the data. What is the frequency increment in the DFT?

(b) To the end of the existing data, add an additional 480 zero values, thus increasing the number of data points to 512. This process is called "zero padding" and is used to improve the frequency resolution in the DFT. Determine the DFT of the new data set and estimate the frequency of the most significant component. What is the frequency increment in the DFT?

7.17. The cost of producing an electronic component varied over a four-year period as follows:

Year	0	1	2	3
Cost	\$30.2	\$25.8	\$22.2	\$20.2

Assuming the equation relating production cost and time is (a) a cubic and (b) a quadratic polynomial, estimate the cost of production in year 6. A small error was discovered in the data. The cost of production in year 2 should have been \$22.5 and in year 3, \$20.5. Recompute the estimated production cost in year 6 using a cubic and a quadratic equation as before. What conclusions can you draw from the results?

7.18. From the following table of values of the gamma function, use inverse interpolation to find the value of x in the range $x = 2$ to $x = 3$ that makes $\Gamma(x) = 1.3$. Use the MATLAB function `interp1` with the cubic option selected; also use the function `aiken`.

x	2	2.2	2.4	2.6	2.8	3
$\Gamma(x)$	1.0000	1.1018	1.2422	1.4296	1.6765	2.0000

7.19. The following table gives the value of the integral

$$I = \int_0^{\pi/2} \frac{d\varphi}{\sqrt{1 - \sin^2\alpha \sin^2\varphi}}$$

for various values of α. (This integral is the complete elliptical integral of the first kind.)

α	0°	5°	10°	15°	20°	25°
I	1.57080	1.57379	1.58284	1.59814	1.62003	1.64900

Using polynomial interpolation, find I when $\alpha = 2°$. Then use inverse interpolation to find the value of α such that $I = 1.58$. In both cases use the MATLAB function `interp1` with the cubic option selected; also use the function `aiken`.

7.20. It is required to find a formula to calculate the number of nodes in one corner of a cube. If n is the number of equally spaced nodes on an edge of the cube and f_n is the number of nodes on three half-faces, including nodes on the face diagonals, then the following table shows values of f_n for given values of n.

n	1	2	3	4
f_n	1	4	10	20

By fitting a cubic function to this data (using `polyfit`), find a general formula for the relationship between f_n and n and verify that when $n = 5$, $f_n = 35$.

7.21. It is required to fit a regression model of the form $z = f(x, y)$ to the following data:

x	0.5	1.0	1.0	2.0	2.5	2.0	3.0	3.5	4.0
y	2.0	4.0	5.0	2.0	4.0	5.0	2.0	4.0	5.0
z	−0.19	−0.32	−1.00	3.71	4.49	2.48	6.31	7.71	8.51

(a) Use the function `mregg2` to generate a model of the form $z = a + bx + cy$ and also $z = a + bx + cy + dxy$. Which model do you consider the best fit to the data? It is important to consider the differences in the error variance.

(b) By analysis of the residuals (particularly the Cook distance), decide whether any data point could be considered to be an *outlier*.

7.22. One of the data values given in Problem 7.21 has been found to be in error. The value of z corresponding to $x = 4$, $y = 5$ should have been recorded as 9.51, not 8.51, a common human error. Use the function `mregg2` to generate models of the form $z = a + bx + cy$ and also $z = a + bx + cy + dxy$. Again, assess the quality of the models.

7.23. Using the following table, obtain the pressure across a shock wave when the upstream Mach number is 4.4 by

(a) Linear interpolation
(b) Aitken's method
(c) Spline interpolation

Mach no.	1.00	2.00	3.00	4.00	5.00
p_2/p_1	1.00	4.50	10.33	18.50	29.00

7.24. MATLAB has available test data sets, including data collected for sunspot activity. This may be obtained using the MATLAB statement

```
load sunspot.dat
```

The set of data `sunspot(:,1)` gives the year of the observed sunspot activity and `sunspot(:,2)` the Wolfer number, which indicates the level of sunspot activity for that year. Let `wolfer = sunspot(:,2)`. Generate a simple plot of the the Wolfer number against the year. To further analyze this data take the fast Fourier transform of the variable `wolfer`. Scaling this data will help in its interpretation. Do this by using the transformations `Power = abs(Y(1:N/2)).^2` and `freq = (1:N/2)/N`; where `N` is the length of the vector `Y`. Plot `freq` against `power`.