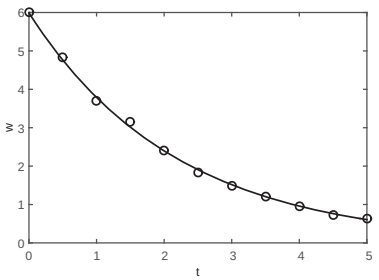
```
m=p(1)
b=exp(p(2))                    Determine the coefficient b.
tm=0:0.1:5;          Create a vector tm to be used for plotting the polynomial.
wm=b*exp(m*tm);      Calculate the function value at each element of tm.
plot(t,w,'o',tm,wm)            Plot the data points and the function.
```

When the program is executed, the values of the constants m and b are displayed in the Command Window.

```
m =
   -0.4580
b =
    5.9889
```

The plot generated by the program, which shows the data points and the function (with axis labels added with the Plot Editor) is



It should be pointed out here that in addition to the power, exponential, logarithmic, and reciprocal functions that are discussed in this section, many other functions can be written in a form suitable for curve fitting with the `polyfit` function. One example where a function of the form $y = e^{(a_2x^2 + a_1x + a_0)}$ is fitted to data points using the `polyfit` function with a third-order polynomial is described in Sample Problem 8-7.

8.3 *INTERPOLATION*

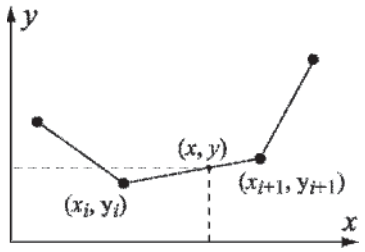
Interpolation is the estimation of values between data points. MATLAB has interpolation functions that are based on polynomials, which are described in this section, and on Fourier transformation, which is outside the scope of this book. In one-dimensional interpolation, each point has one independent variable (x) and one dependent variable (y). In two-dimensional interpolation, each point has two independent variables (x and y) and one dependent variable (z).

One-dimensional interpolation:

If only two data points exist, the points can be connected with a straight line and a linear equation (polynomial of first order) can be used to estimate values between the points. As was discussed in the previous section, if three (or four) data points exist, a second- (or a third-) order polynomial that passes through the points can be determined and then be used to estimate values between the points. As the number of points increases, a higher-order polynomial is required for the polynomial to pass through all the points. Such a polynomial, however, will not necessarily give a good approximation of the values between the points. This is illustrated in Figure 8-2 with $n = 6$.

A more accurate interpolation can be obtained if instead of considering all the points in the data set (by using one polynomial that passes through all the points), only a few data points in the neighborhood where the interpolation is needed are considered. In this method, called spline interpolation, many low-order polynomials are used, where each is valid only in a small domain of the data set.

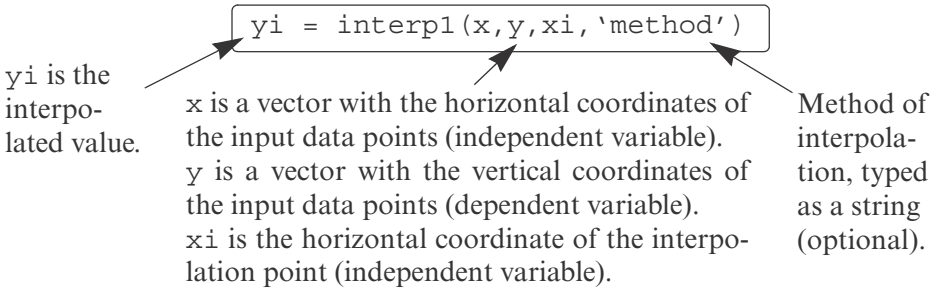
The simplest method of spline interpolation is called linear spline interpolation. In this method, shown on the right, every two adjacent points are connected with a straight line (a polynomial of first degree). The equation of a straight line that passes through two adjacent points (x_i, y_j) and (x_{i+1}, y_{j+1}) and that can be used to calculate the value of y for any x between the points is given by:



$$y = \frac{y_{i+1} - y_i}{x_{i+1} - x_i} x + \frac{y_i x_{i+1} - y_{i+1} x_i}{x_{i+1} - x_i}$$

In a linear interpolation, the line between two data points has a constant slope, and there is a change in the slope at every point. A smoother interpolation curve can be obtained by using quadratic or cubic polynomials. In these methods, called quadratic splines and cubic splines, a second-, or third-order polynomial is used to interpolate between every two points. The coefficients of the polynomial are determined by using data from points that are adjacent to the two data points. The theoretical background for the determination of the constants of the polynomials is beyond the scope of this book and can be found in books on numerical analysis.

One-dimensional interpolation in MATLAB is done with the `interp1` (the last character is the numeral one) function, which has the form:



- The vector `x` must be monotonic (with elements in ascending or descending order).
- `xi` can be a scalar (interpolation of one point) or a vector (interpolation of many points). `yi` is a scalar or a vector with the corresponding interpolated values.
- MATLAB can do the interpolation using one of several methods that can be specified. These methods include:

`'nearest'`	returns the value of the data point that is nearest to the interpolated point.
`'linear'`	uses linear spline interpolation.
`'spline'`	uses cubic spline interpolation.
`'pchip'`	uses piecewise cubic Hermite interpolation, also called `'cubic'`

- When the `'nearest'` and the `'linear'` methods are used, the value(s) of `xi` must be within the domain of `x`. If the `'spline'` or the `'pchip'` methods are used, `xi` can have values outside the domain of `x` and the function `interp1` performs extrapolation.
- The `'spline'` method can give large errors if the input data points are nonuniform such that some points are much closer together than others.
- Specification of the method is optional. If no method is specified, the default is `'linear'`.

Sample Problem 8-3: Interpolation

The following data points, which are points of the function $f(x) = 1.5^x \cos(2x)$, are given. Use linear, spline, and pchip interpolation methods to calculate the value of y between the points. Make a figure for each of the interpolation methods. In the figure show the points, a plot of the function, and a curve that corre-

sponds to the interpolation method.

x	0	1	2	3	4	5
y	1.0	–0.6242	–1.4707	3.2406	–0.7366	–6.3717

Solution

The following is a program written in a script file that solves the problem:

```
x=0:1.0:5;                    Create vectors x and y with coordinates of the data points.
y=[1.0 -0.6242 -1.4707 3.2406 -0.7366 -6.3717];
xi=0:0.1:5;                   Create vector xi with points for interpolation.
yilin=interp1(x,y,xi,'linear'); Calculate y points from linear interpolation.
yispl=interp1(x,y,xi,'spline'); Calculate y points from spline interpolation.
yipch=interp1(x,y,xi,'pchip'); Calculate y points from pchip interpolation.
yfun=1.5.^xi.*cos(2*xi);      Calculate y points from the function.
subplot(1,3,1)
plot(x,y,'o',xi,yfun,xi,yilin,'--');
subplot(1,3,2)
plot(x,y,'o',xi,yfun,xi,yispl,'--');
subplot(1,3,3)
plot(x,y,'o',xi,yfun,xi,yipch,'--');
```

The three figures generated by the program are shown below (axes labels were added with the Plot Editor). The data points are marked with circles, the interpolation curves are plotted with dashed lines, and the function is shown with a solid line. The left figure shows the linear interpolation, the middle is the spline, and the figure on the right shows the pchip interpolation.