Polynomial Interpolation

Berlin Chen
Department of Computer Science & Information Engineering
National Taiwan Normal University

Reference:

1. *Applied Numerical Methods with MATLAB for Engineers*, Chapter 17 & Teaching material

Chapter Objectives (1/2)

- Recognizing that evaluating polynomial coefficients with simultaneous equations is an ill-conditioned problem
- Knowing how to evaluate polynomial coefficients and interpolate with MATLAB's polyfit and polyval functions
- Knowing how to perform an interpolation with Newton's polynomial
- Knowing how to perform an interpolation with a Lagrange polynomial

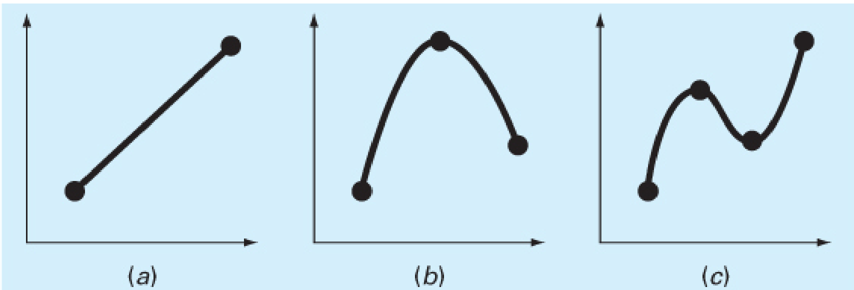


FIGURE 17.1
Examples of interpolating polynomials: (a) first-order (linear) connecting two points, (b) second-order (quadratic or parabolic) connecting three points, and (c) third-order (cubic) connecting four points.

Chapter Objectives (2/2)

- Knowing how to solve an inverse interpolation problem by recasting it as a roots problem
- Appreciating the dangers of extrapolation
- Recognizing that higher-order polynomials can manifest large oscillations

Polynomial Interpolation

- You will frequently have occasions to estimate intermediate values between precise data points
- The function you use to interpolate must pass through the actual data points - this makes interpolation more restrictive than fitting
- The most common method for this purpose is polynomial interpolation, where an $(n\text{-}1)^{\text{th}}$ order polynomial is solved that passes through n data points:

$$f(x) = a_1 + a_2 x + a_3 x^2 + \cdots + a_n x^{n-1}$$

MATLAB version:

$$f(x) = p_1 x^{n-1} + p_2 x^{n-2} + \cdots + p_{n-1} x + p_n$$

Determining Coefficients

- Since polynomial interpolation provides as many basis functions as there are data points (*n*), the polynomial coefficients can be found exactly using linear algebra
  - For *n* data points, there is one and only one polynomial of order (*n*-1) that passes through all the points

- MATLAB's built in polyfit and polyval commands can also be used - all that is required is making sure the order of the fit for *n* data points is *n*-1

Polynomial Interpolation Problems

- One problem that can occur with solving for the coefficients of a polynomial is that the system to be inverted is in the form:

$$\begin{bmatrix} x_1^{n-1} & x_1^{n-2} & \cdots & x_1 & 1 \\ x_2^{n-1} & x_2^{n-2} & \cdots & x_2 & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ x_{n-1}^{n-1} & x_{n-1}^{n-2} & \cdots & x_{n-1} & 1 \\ x_n^{n-1} & x_n^{n-2} & \cdots & x_n & 1 \end{bmatrix} \begin{Bmatrix} p_1 \\ p_2 \\ \vdots \\ p_{n-1} \\ p_n \end{Bmatrix} = \begin{Bmatrix} f(x_1) \\ f(x_2) \\ \vdots \\ f(x_{n-1}) \\ f(x_n) \end{Bmatrix}$$

- Matrices such as that on the left are known as ***Vandermonde matrices***, and they are very ill-conditioned - meaning their solutions are very sensitive to round-off errors
- The issue can be minimized by scaling and shifting the data

Newton Interpolating Polynomials

- Another way to express a polynomial interpolation is to use ***Newton's interpolating polynomial***
- The differences between a simple polynomial and Newton's interpolating polynomial for first and second order interpolations are:

Order	Simple	Newton
$1st$	$f_1(x) = a_1 + a_2 x$	$f_1(x) = b_1 + b_2(x - x_1)$
$2nd$	$f_2(x) = a_1 + a_2 x + a_3 x^2$	$f_2(x) = b_1 + b_2(x - x_1) + b3(x - x_1)(x - x_2)$

$$b_1 = f(x_1)$$

$$b_2 = \frac{f(x_2) - f(x_1)}{x_2 - x_1}$$

$$b_3 = \frac{\dfrac{f(x_3) - f(x_2)}{x_3 - x_2} - \dfrac{f(x_2) - f(x_1)}{x_2 - x_1}}{x_3 - x_1}$$

Newton Interpolating Polynomials (1/3)

- The **first-order Newton interpolating polynomial** may be obtained from linear interpolation and similar triangles, as shown
- The resulting formula based on known points x_1 and x_2 and the values of the dependent function at those points is:

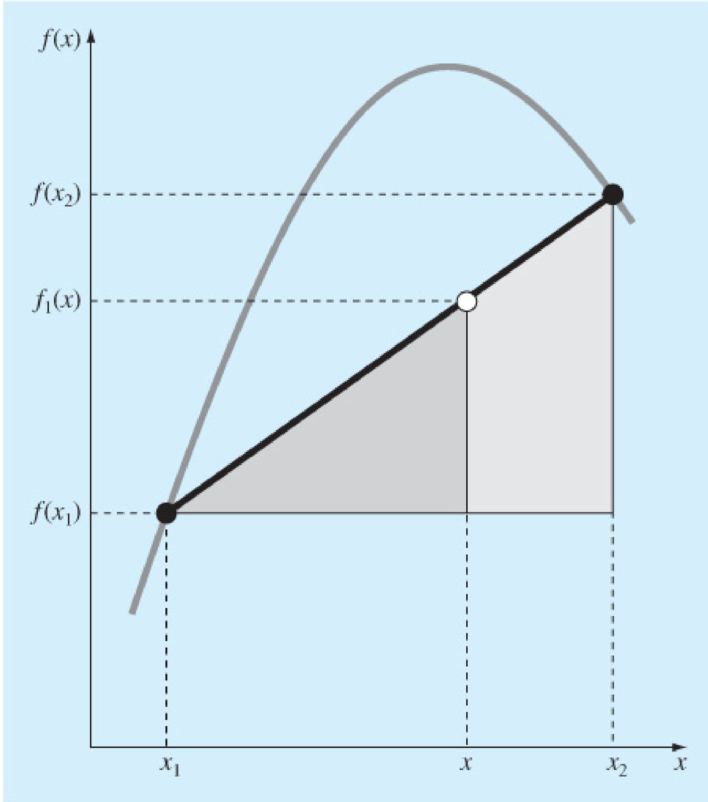


FIGURE 17.2
Graphical depiction of linear interpolation. The shaded areas indicate the similar triangles used to derive the Newton linear-interpolation formula [Eq. (17.5)].

$$f_1(x)=b_1+b_2(x-x_1)$$

$$\Rightarrow f_1(x)=f(x_1)+\frac{f(x_2)-f(x_1)}{x_2-x_1}(x-x_1) \Leftarrow \frac{f_1(x)-f(x_1)}{x-x_1}=\frac{f(x_2)-f(x_1)}{x_2-x_1}$$

designate a first-order polynomial

Linear Interpolation: An Example

Problem Statement. Estimate the natural logarithm of 2 using linear interpolation. First, perform the computation by interpolating between ln 1 = 0 and ln 6 = 1.791759. Then, repeat the procedure, but use a smaller interval from ln 1 to ln 4 (1.386294). Note that the true value of ln 2 is 0.6931472.

Example 17.2

Solution. We use Eq. (17.5) from $x_1 = 1$ to $x_2 = 6$ to give

$$f_1(2) = 0 + \frac{1.791759 - 0}{6 - 1}(2 - 1) = 0.3583519$$

which represents an error of $\varepsilon_t = 48.3\%$. Using the smaller interval from $x_1 = 1$ to $x_2 = 4$ yields

$$f_1(2) = 0 + \frac{1.386294 - 0}{4 - 1}(2 - 1) = 0.4620981$$

The smaller the interval between the data points, the better the approximation.

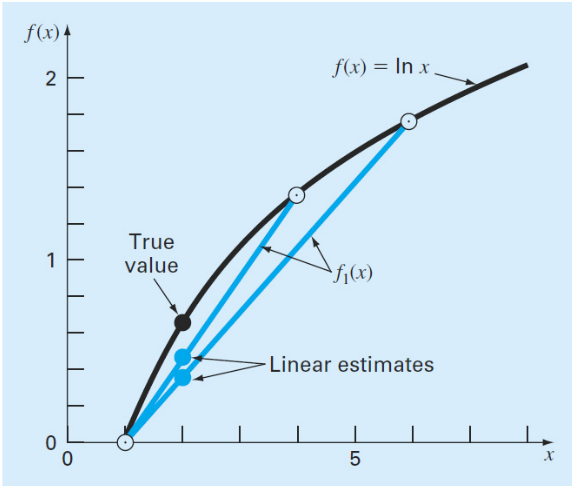


FIGURE 17.3
Two linear interpolations to estimate ln 2. Note how the smaller interval provides a better estimate.

Newton Interpolating Polynomials (2/3)

- The **second-order Newton interpolating polynomial** introduces some curvature to the line connecting the points, but still goes through the first two points
- The resulting formula based on known points x_1, x_2, and x_3 and the values of the dependent function at those points is:

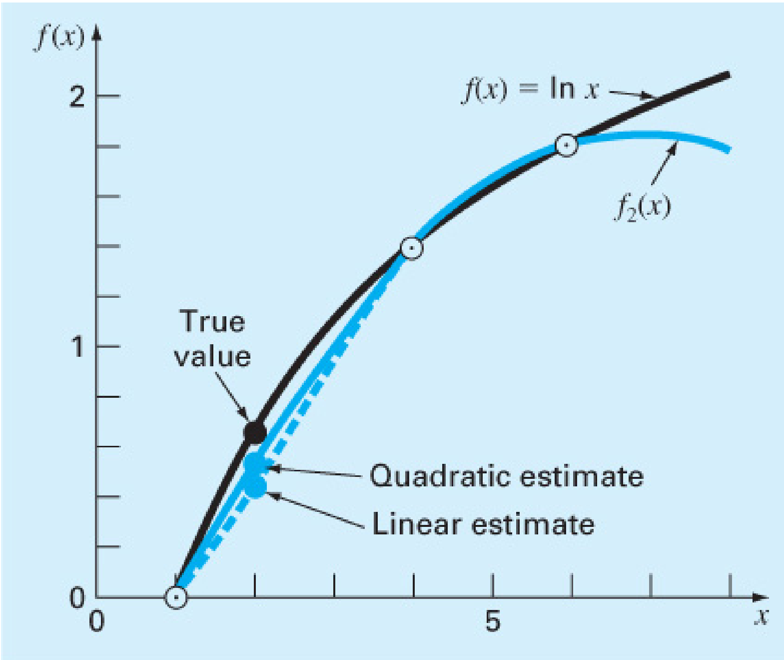


FIGURE 17.4
The use of quadratic interpolation to estimate ln 2. The linear interpolation from $x = 1$ to 4 is also included for comparison.

$$f_2(x) = b_1 + b_2(x - x_1) + b_3(x - x_1)(x - x_2)$$

$$\Rightarrow f_2(x) = f(x_1) + \frac{f(x_2) - f(x_1)}{x_2 - x_1}(x - x_1) + \frac{\frac{f(x_3) - f(x_2)}{x_3 - x_2} - \frac{f(x_2) - f(x_1)}{x_2 - x_1}}{x_3 - x_1}(x - x_1)(x - x_2)$$

Quadratic Interpolation: An Example

Example 17.3

Problem Statement. Employ a second-order Newton polynomial to estimate ln 2 with the same three points used in Example 17.2:

$$x_1 = 1 \qquad f(x_1) = 0$$
$$x_2 = 4 \qquad f(x_2) = 1.386294$$
$$x_3 = 6 \qquad f(x_3) = 1.791759$$

Solution. Applying Eq. (17.7) yields

$$b_1 = 0$$

Equation (17.8) gives

$$b_2 = \frac{1.386294 - 0}{4 - 1} = 0.4620981$$

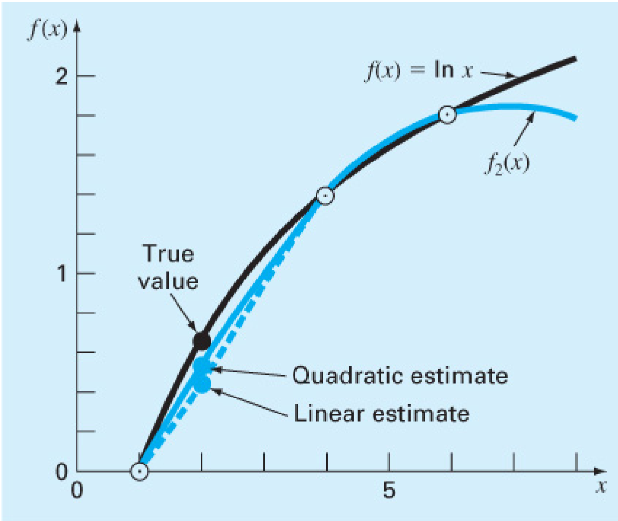
and Eq. (17.9) yields

$$b_3 = \frac{\dfrac{1.791759 - 1.386294}{6 - 4} - 0.4620981}{6 - 1} = -0.0518731$$

Substituting these values into Eq. (17.6) yields the quadratic formula

$$f_2(x) = 0 + 0.4620981(x - 1) - 0.0518731(x - 1)(x - 4)$$

which can be evaluated at $x = 2$ for $f_2(2) = 0.5658444$, which represents a relative error of $\varepsilon_t = 18.4\%$. Thus, the curvature introduced by the quadratic formula (Fig. 17.4) improves the interpolation compared with the result obtained using straight lines in Example 17.2 and Fig. 17.3.



Newton Interpolating Polynomials (3/3)

- In general, an (n-1)th Newton interpolating polynomial has all the terms of the (n-2)th polynomial plus one extra
- The general formula is:

$$f_{n-1}(x) = b_1 + b_2(x - x_1) + \cdots + b_n(x - x_1)(x - x_2)\cdots(x - x_{n-1})$$

where

$$b_1 = f(x_1)$$

$$b_2 = f[x_2, x_1]$$

$$b_3 = f[x_3, x_2, x_1]$$

$$\vdots$$

$$b_n = f[x_n, x_{n-1}, \cdots, x_2, x_1]$$

and the *f*[…] represent ***divided differences***

Divided Differences

- **Divided difference** are calculated as follows:

$$f[x_i, x_j] = \frac{f(x_i) - f(x_j)}{x_i - x_j}$$

$$f[x_i, x_j, x_k] = \frac{f[x_i, x_j] - f[x_j, x_k]}{x_i - x_k}$$

$$f[x_n, x_{n-1}, \cdots, x_2, x_1] = \frac{f[x_n, x_{n-1}, \cdots, x_2] - f[x_{n-1}, x_{n-2}, \cdots, x_1]}{x_n - x_1}$$

- **Divided differences** are calculated using divided difference of a smaller number of terms:

x_i	$f(x_i)$	First	Second	Third
x_1	$f(x_1)$	$f[x_2, x_1]$	$f[x_3, x_2, x_1]$	$f[x_4, x_3, x_2, x_1]$
x_2	$f(x_2)$	$f[x_3, x_2]$	$f[x_4, x_3, x_2]$	
x_3	$f(x_3)$	$f[x_4, x_3]$		
x_4	$f(x_4)$			

MATLAB Implementation

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

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

Lagrange Interpolating Polynomials (1/3)

- Another method that uses shifted values to express an interpolating polynomial is the ***Lagrange interpolating polynomial***

- The differences between a simply polynomial and Lagrange interpolating polynomials for first and second order polynomials is:

Order	Simple	Lagrange
$1st$	$f_1(x)=a_1+a_2x$	$f_1(x)=L_1^2 f(x_1)+L_2^2 f(x_2)$
$2nd$	$f_2(x)=a_1+a_2x+a_3x^2$	$f_2(x)=L_1^3 f(x_1)+L_2^3 f(x_2)+L_3^3 f(x_3)$

  - where the L_i^n are weighting coefficients of the *j*th polynomial, which are functions of *x*

$$L_1^2=\frac{x-x_2}{x_1-x_2}, L_2^2=\frac{x-x_1}{x_2-x_1}$$
$$L_1^3=\frac{(x-x_2)(x-x_3)}{(x_1-x_2)(x_1-x_3)}, L_2^3=\frac{(x-x_1)(x-x_3)}{(x_2-x_1)(x_2-x_3)}, L_3^3=\frac{(x-x_1)(x-x_2)}{(x_3-x_1)(x_3-x_2)}$$

Lagrange Interpolating Polynomials (2/3)

- The first-order Lagrange interpolating polynomial may be obtained from a weighted combination of two linear interpolations, as shown
- The resulting formula based on known points x_1 and x_2 and the values of the dependent function at those points is:

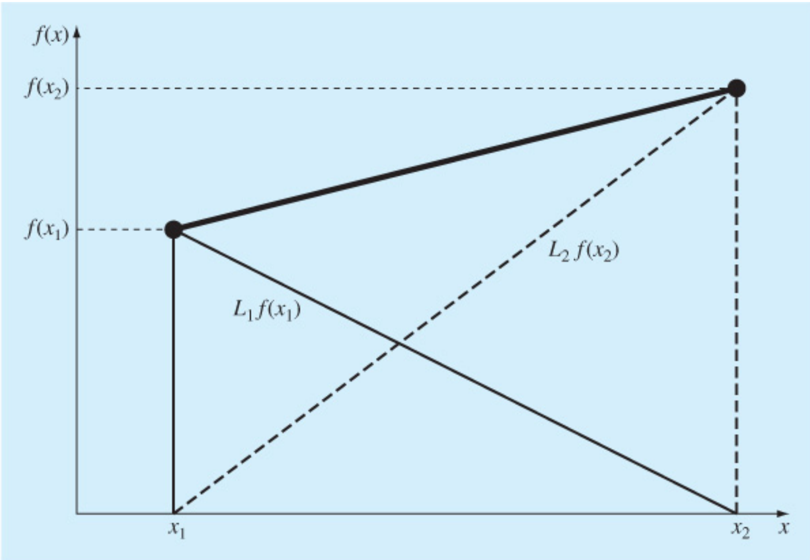


FIGURE 17.8
A visual depiction of the rationale behind Lagrange interpolating polynomials. The figure shows the first-order case. Each of the two terms of Eq. (17.20) passes through one of the points and is zero at the other. The summation of the two terms must, therefore, be the unique straight line that connects the two points.

$$f_1(x) = L_1^2 f(x_1) + L_2^2 f(x_2)$$

$$L_1^2 = \frac{x - x_2}{x_1 - x_2}, L_2^2 = \frac{x - x_1}{x_2 - x_1}$$

$$f_1(x) = \frac{x - x_2}{x_1 - x_2} f(x_1) + \frac{x - x_1}{x_2 - x_1} f(x_2)$$

Lagrange Interpolating Polynomials (3/3)

- In general, the Lagrange polynomial interpolation for *n* points is:

$$f_{n-1}(x_i) = \sum_{i=1}^{n} L_i^n(x) f(x_i)$$

  – where L_i^n is given by:

$$L_i^n(x) = \prod_{\substack{j=1 \\ j \neq i}}^{n} \frac{x - x_j}{x_i - x_j}$$

Lagrange Interpolating Polynomial: An Example

Problem Statement. Use a Lagrange interpolating polynomial of the first and second order to evaluate the density of unused motor oil at $T = 15$ °C based on the following data:

$$x_1 = 0 \qquad f(x_1) = 3.85$$
$$x_2 = 20 \qquad f(x_2) = 0.800$$
$$x_3 = 40 \qquad f(x_3) = 0.212$$

Solution. The first-order polynomial [Eq. (17.20)] can be used to obtain the estimate at $x = 15$:

$$f_1(x) = \frac{15-20}{0-20}3.85 + \frac{15-0}{20-0}0.800 = 1.5625$$

In a similar fashion, the second-order polynomial is developed as [Eq. (17.21)]

$$f_2(x) = \frac{(15-20)(15-40)}{(0-20)(0-40)}3.85 + \frac{(15-0)(15-40)}{(20-0)(20-40)}0.800$$
$$+ \frac{(15-0)(15-20)}{(40-0)(40-20)}0.212 = 1.3316875$$

MATLAB Implementation

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

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

Inverse Interpolation (1/2)

- Interpolation general means finding some value *f*(*x*) for some *x* that is between given independent data points

- Sometimes, it will be useful to find the *x* for which *f*(*x*) is a certain value - this is *inverse interpolation*

$f(x) = 1/x$:

x	1	2	3	4	5	6	7
$f(x)$	1	0.5	0.3333	0.25	0.2	0.1667	0.1429

$f(x)$	0.1429	0.1667	0.2	0.25	0.3333	0.5	1
x	7	6	5	4	3	2	1

Some adjacent points of *f*(*x*) are bunched together and others spread out widely.

This would lead to oscillations in the resulting polynomial.

Inverse Interpolation (2/2)

- Rather than finding an interpolation of x as a function of $f(x)$, it may be useful to find an equation for $f(x)$ as a function of x using interpolation and then solve the corresponding ***roots problem***:
  $f(x)-f_{\text{desired}}=0$ for x

For example, for the problem just outlined, a simple approach would be to fit a quadratic polynomial to the three points: (2, 0.5), (3, 0.3333), and (4, 0.25). The result would be

$$f_2(x) = 0.041667x^2 - 0.375x + 1.08333$$

The answer to the inverse interpolation problem of finding the x corresponding to $f(x) = 0.3$ would therefore involve determining the root of

$$0.3 = 0.041667x^2 - 0.375x + 1.08333$$

For this simple case, the quadratic formula can be used to calculate

$$x = \frac{0.375 \pm \sqrt{(-0.375)^2 - 4(0.041667)0.78333}}{2(0.041667)} = \begin{matrix} 5.704158 \\ 3.295842 \end{matrix}$$

Thus, the second root, 3.296, is a good approximation of the true value of 3.333.

Extrapolation

- *Extrapolation* is the process of estimating a value of $f(x)$ that lies outside the range of the known base points x_1, x_2, …, x_n
- Extrapolation represents a step into the unknown, and extreme care should be exercised when extrapolating!

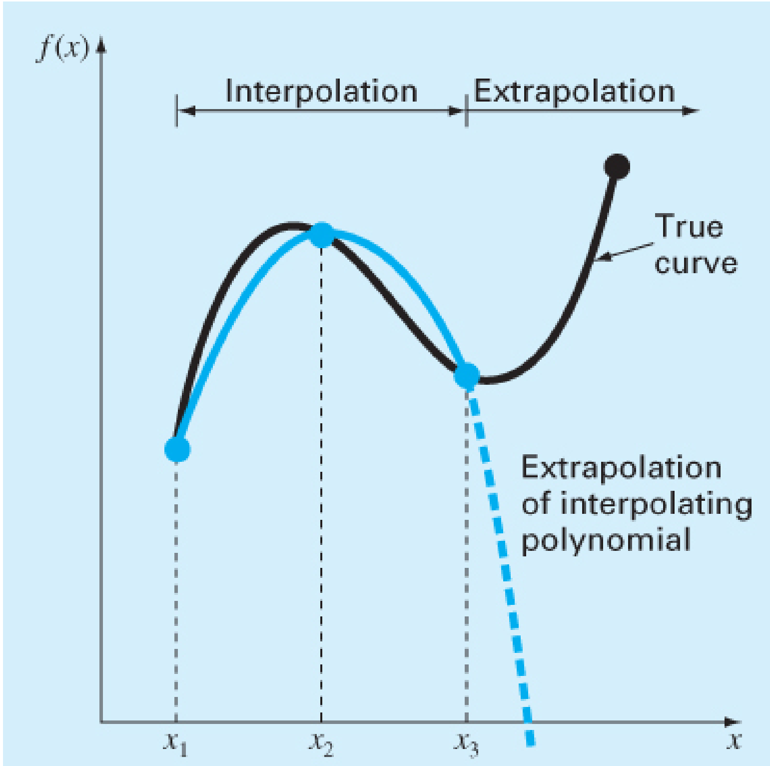


FIGURE 17.10
Illustration of the possible divergence of an extrapolated prediction. The extrapolation is based on fitting a parabola through the first three known points.

Extrapolation Hazards

- The following shows the results of extrapolating a seventh-order polynomial which is derived from the first 8 points (1920 to 1990) of the USA population data set:

Date	1920	1930	1940	1950	1960	1970	1980	1990	2000
Population	106.46	123.08	132.12	152.27	180.67	205.05	227.23	249.46	281.42

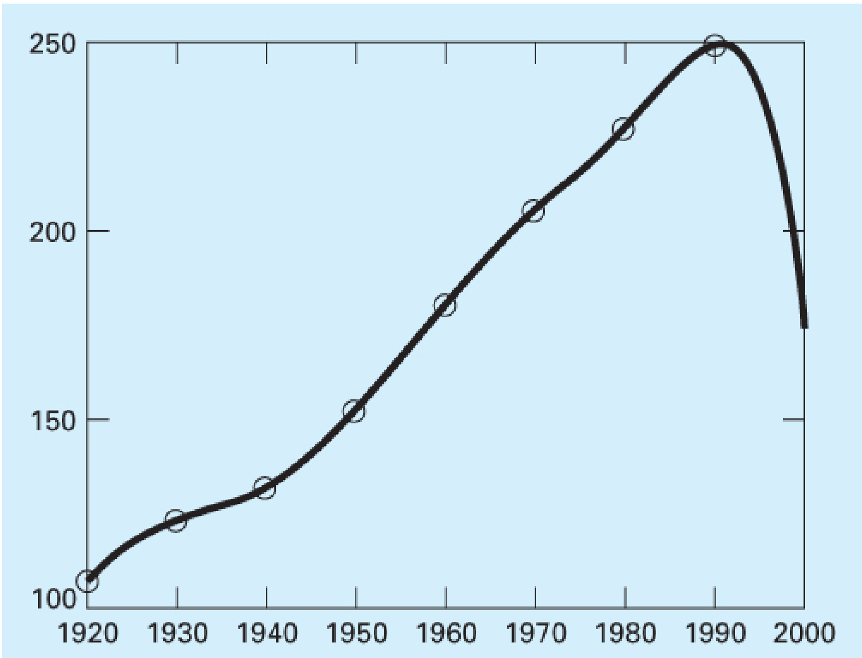
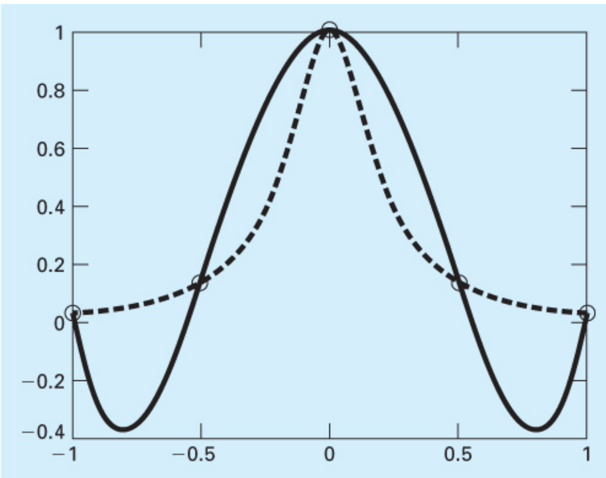


FIGURE 17.11
Use of a seventh-order polynomial to make a prediction of U.S. population in 2000 based on data from 1920 through 1990.

Oscillations

- Higher-order polynomials can not only lead to round-off errors due to ***ill-conditioning***, but can also introduce oscillations to an interpolation or fit where they should not be
- In the figures below, **the dashed line represents an function**, the circles represent samples of the function, and **the solid line represents the results of a polynomial interpolation**:



$$f(x) = \frac{1}{1+25x^2}$$

(Runge's function)

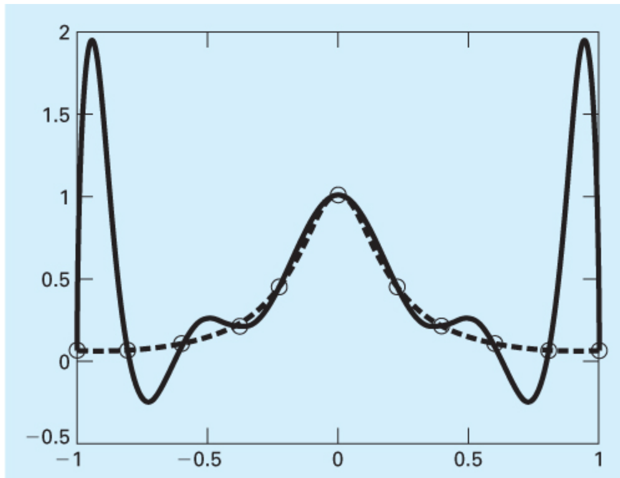


FIGURE 17.12
Comparison of Runge's function (dashed line) with a fourth-order polynomial fit to 5 points sampled from the function.

FIGURE 17.13
Comparison of Runge's function (dashed line) with a tenth-order polynomial fit to 11 points sampled from the function.