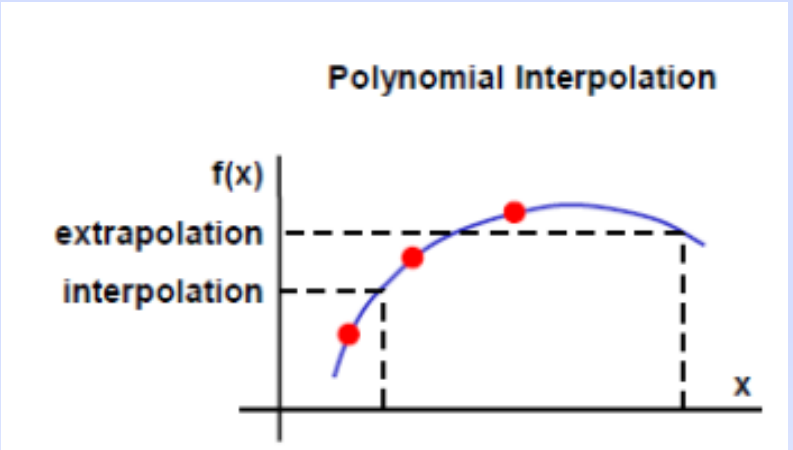
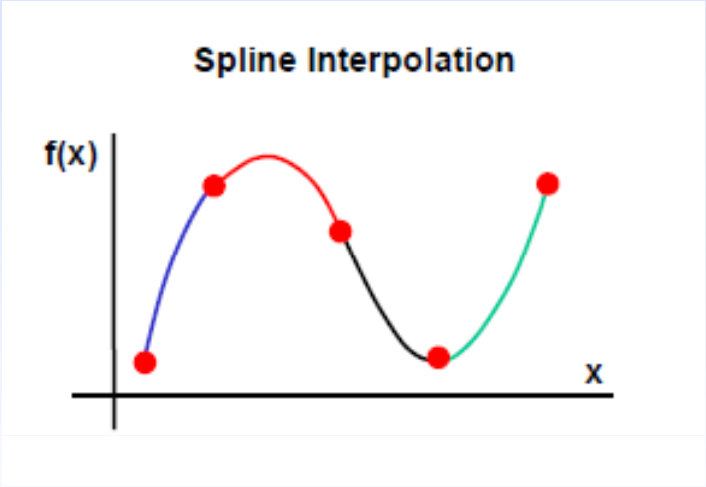
Interpolation

Chapter 18

- Estimating intermediate values between precise data points
- We first fit a function that exactly passes through the given data points and then evaluate intermediate values using this function
- Polynomial interpolation: A unique *n*th order polynomial passes through *n* points
- Spline interpolation: Pass different curves (mostly 3rd order) through different subsets of data points





Interpolation

Chapter 18

- The most common method is polynomial interpolation:

$$f(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n$$

- Although there is one and only one *n*th-order polynomial that fits *n+1* points, there are a variety of mathematical formats in which this polynomial can be expressed:
  - The Newton polynomial
  - The Lagrange polynomial

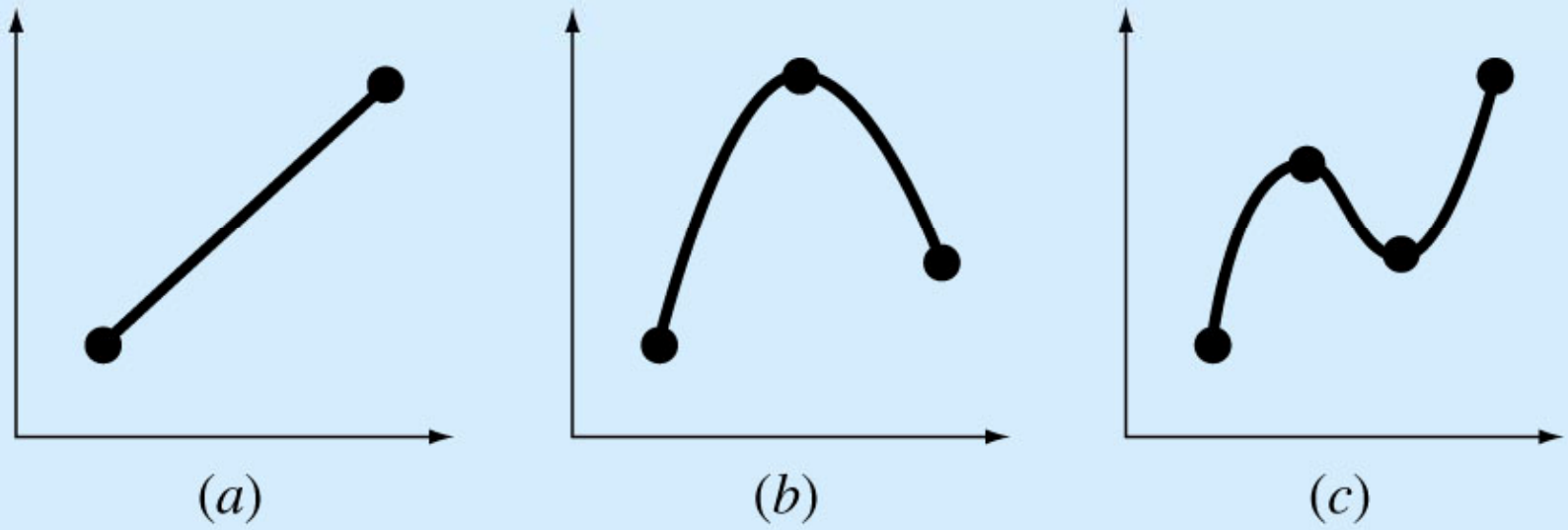


FIGURE 18.1
Examples of interpolating polynomials: (*a*) first-order (linear) connecting two points, (*b*) second-order (quadratic or parabolic) connecting three points, and (*c*) third-order (cubic) connecting four points.

Newton's Divided-Difference Interpolating Polynomials

Linear Interpolation/

- Is the simplest form of interpolation, connecting two data points with a straight line.

$$\frac{f_1(x) - f(x_0)}{x - x_0} = \frac{f(x_1) - f(x_0)}{x_1 - x_0}$$

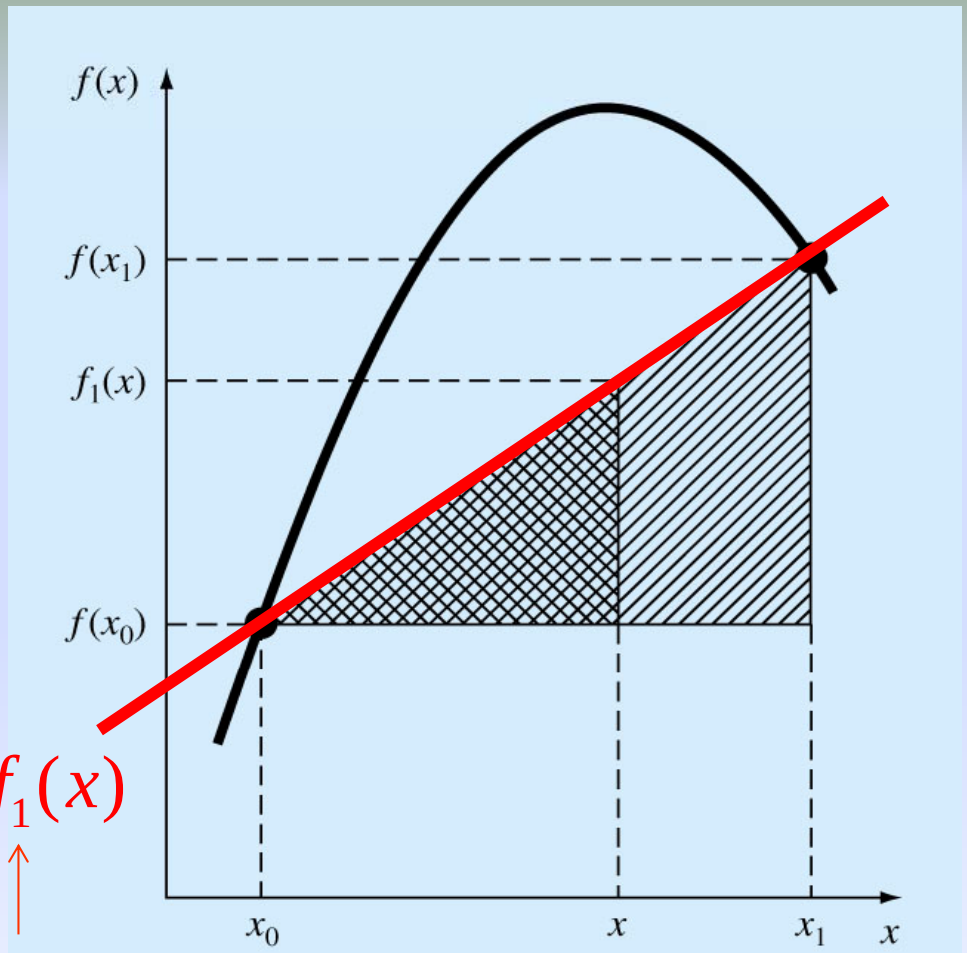
$$f_1(x) = f(x_0) + \frac{f(x_1) - f(x_0)}{x_1 - x_0}(x - x_0)$$

b_0 b_1

Slope and a finite divided difference approximation to 1st derivative

Linear-interpolation formula

- $f_1(x)$ designates that this is a first-order interpolating polynomial.



$f_1(x)$

First order interpolating polynomial

Similar triangles are shown in the figure. The ratio is:

$$\frac{f_1(x)-f(x_0)}{x-x_0}=\frac{f(x_1)-f(x_0)}{x_1-x_0}$$

$$f_1(x)=f(x_0)+\frac{f(x_1)-f(x_0)}{x_1-x_0}(x-x_0)$$

$$f_1(x)=a_0+a_1x \quad \text{or}$$

$$f_1(x)=b_0+b_1(x-x_0)$$

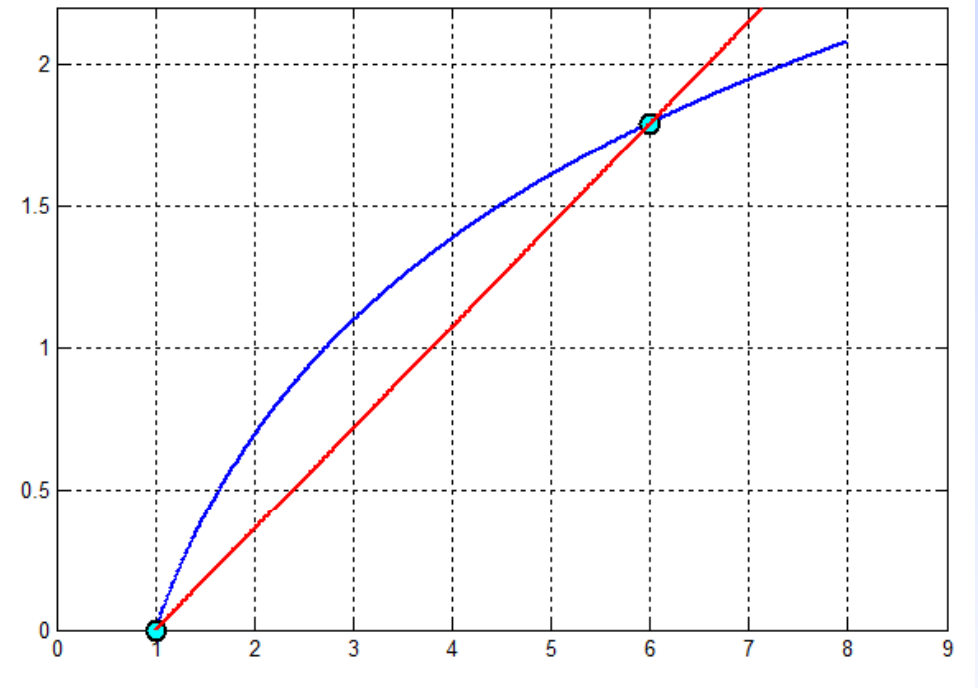
FIGURE 18.2
Graphical depiction of linear interpolation. The shaded areas indicate the similar triangles used to derive the linear-interpolation formula [Eq. (18.2)].

Example 18.1:

Linear Interpolation

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.

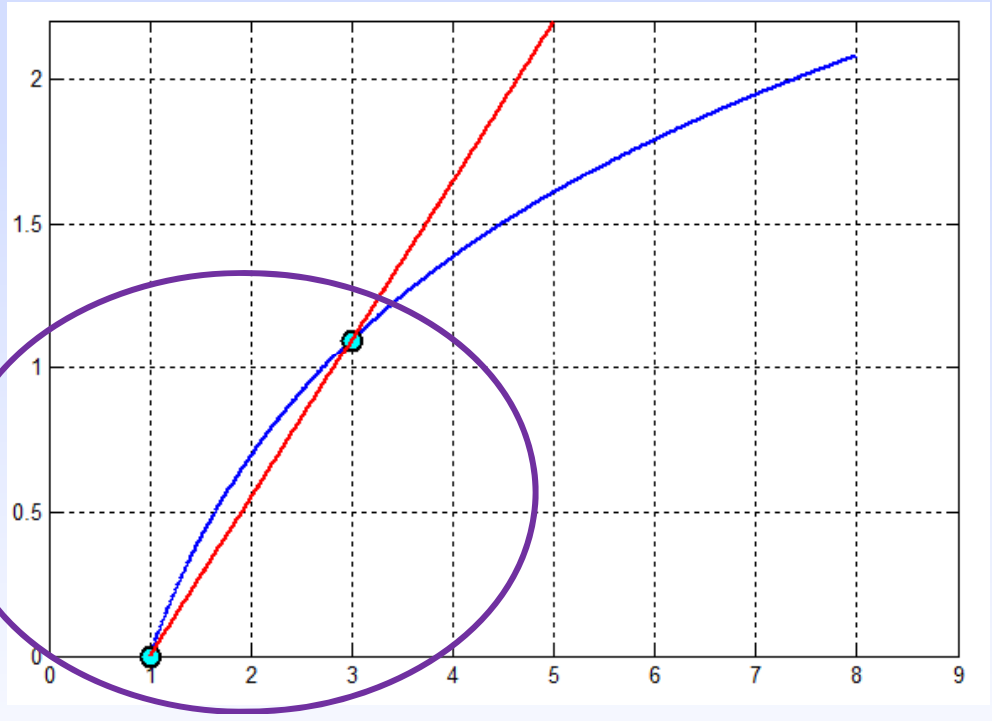
```
x0=1;
x1=6;
f=@(x) log(x);
x=x0:0.01:x1+0.1;
plot(x,f(x),'b') %function is blue
hold on
% interpolating line:
b0=f(x0);
b1=(f(x1)-f(x0))/(x1-x0);
y1=b0+b1*(x-x0);
plot(x,y1,'r') % line is red
hold off
```



Linear Interpolation

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.

```
x0=1;
x1=3;
f=@(x) log(x);
x=x0:0.01:x1+0.1;
plot(x,f(x),'b') %function is blue
hold on
% interpolating line:
b0=f(x0);
b1=(f(x1)-f(x0))/(x1-x0);
y1=b0+b1*(x-x0);
plot(x,y1,'r') % line is red
hold off
```



using the shorter interval reduces the percent relative error

Quadratic Interpolation/

- If three data points are available, the estimate is improved by introducing some curvature into the line connecting the points.

$$f_2(x) = b_0 + b_1(x - x_0) + b_2(x - x_0)(x - x_1)$$

second order interpolating polynomial.

- If we know three points (x_0,y_0), (x_1,y_1) and (x_2,y_2), the following simple procedure can be used to determine the values of the coefficients:

$$x = x_0 \qquad b_0 = f(x_0)$$

$$x = x_1 \qquad b_1 = \frac{f(x_1) - f(x_0)}{x_1 - x_0}$$

$$x = x_2 \qquad b_2 = \frac{\dfrac{f(x_2) - f(x_1)}{x_2 - x_1} - \dfrac{f(x_1) - f(x_0)}{x_1 - x_0}}{x_2 - x_0}$$

Homework:

Quadratic Interpolation

Problem Statement. Fit a second-order polynomial to the three points used in Example 18.1:

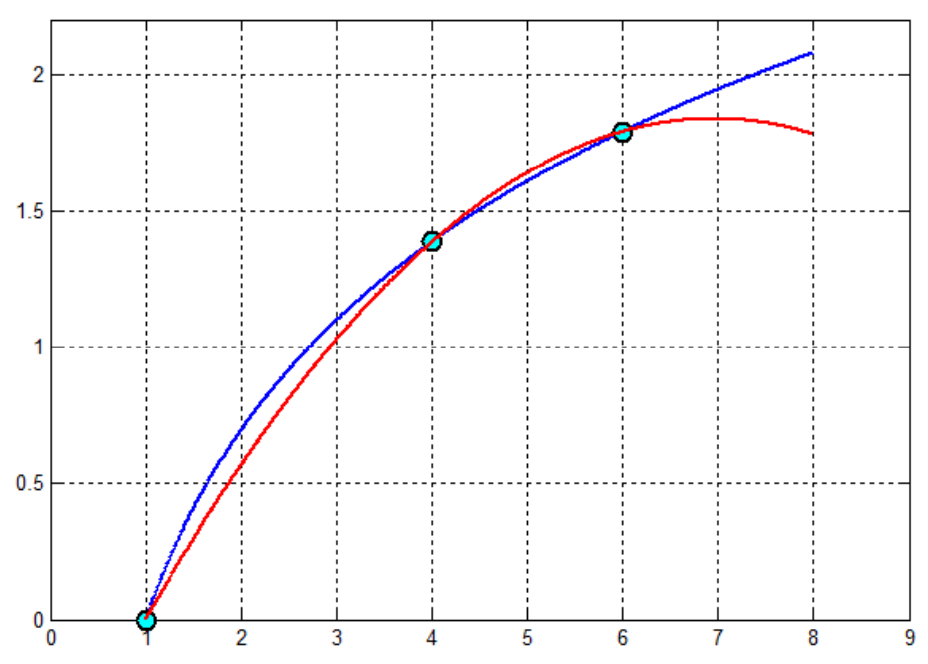
$x_0 = 1 \quad f(x_0) = 0$

$x_1 = 4 \quad f(x_1) = 1.386294$

$x_2 = 6 \quad f(x_2) = 1.791759$

Modify the MATLAB code given in the previous example for quadratic interpolation and try it with the given initial values and some extra initial values.

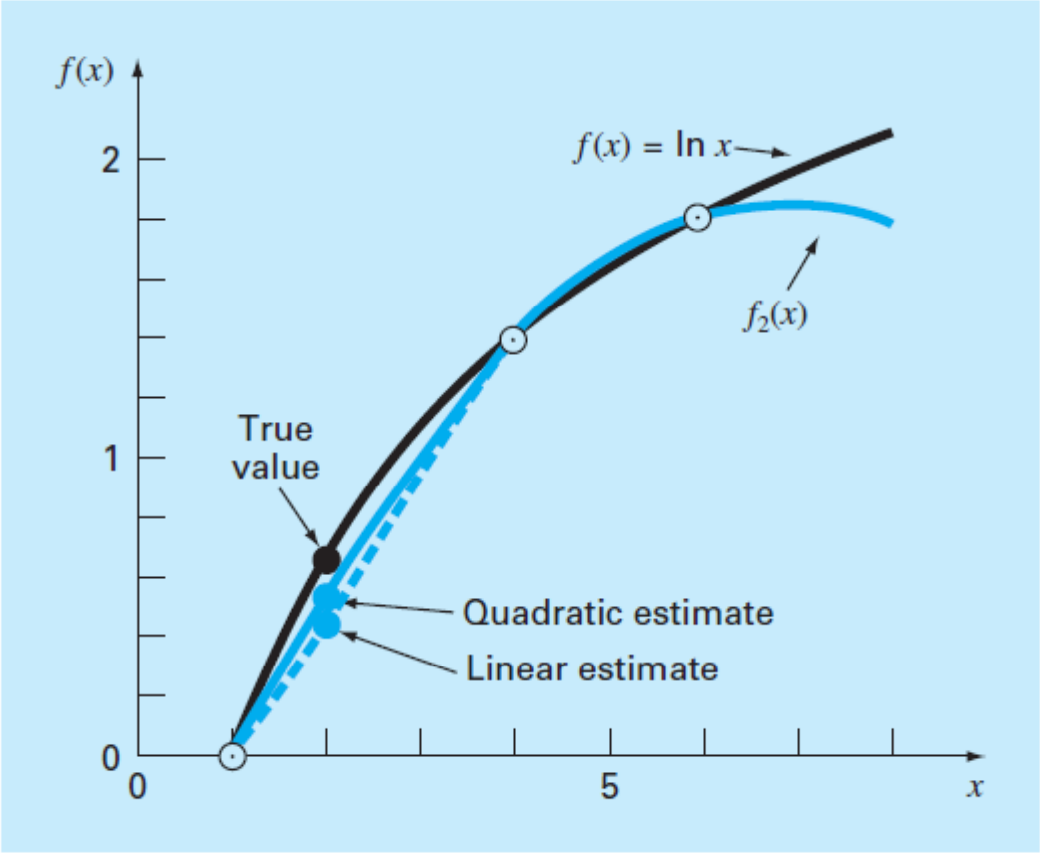
Answer:



Which one has less error – linear or quadratic interpolation?

FIGURE 18.4

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



General Form of Newton's Interpolating Polynomials/

$$f_n(x) = f(x_0) + (x - x_0) f[x_1, x_0] + (x - x_0)(x - x_1) f[x_2, x_1, x_0]$$
$$+ \cdots + (x - x_0)(x - x_1) \cdots (x - x_{n-1}) f[x_n, x_{n-1}, \cdots, x_0]$$

$$b_0 = f(x_0)$$

$$b_1 = f[x_1, x_0]$$

$$b_2 = f[x_2, x_1, x_0]$$

$$\vdots$$

$$b_n = f[x_n, x_{n-1}, \cdots, x_1, x_0]$$

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

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

$$\vdots$$

$$f[x_n, x_{n-1}, \ldots, x_1, x_0] = \frac{f[x_n, x_{n-1}, \ldots, x_1] - f[x_{n-1}, x_{n-2}, \ldots, x_0]}{x_n - x_0}$$

Bracketed function evaluations are finite divided differences

Finite divided differences used in the Newton's interpolating polynomials can be represented in a table form. This makes the calculations much simpler.

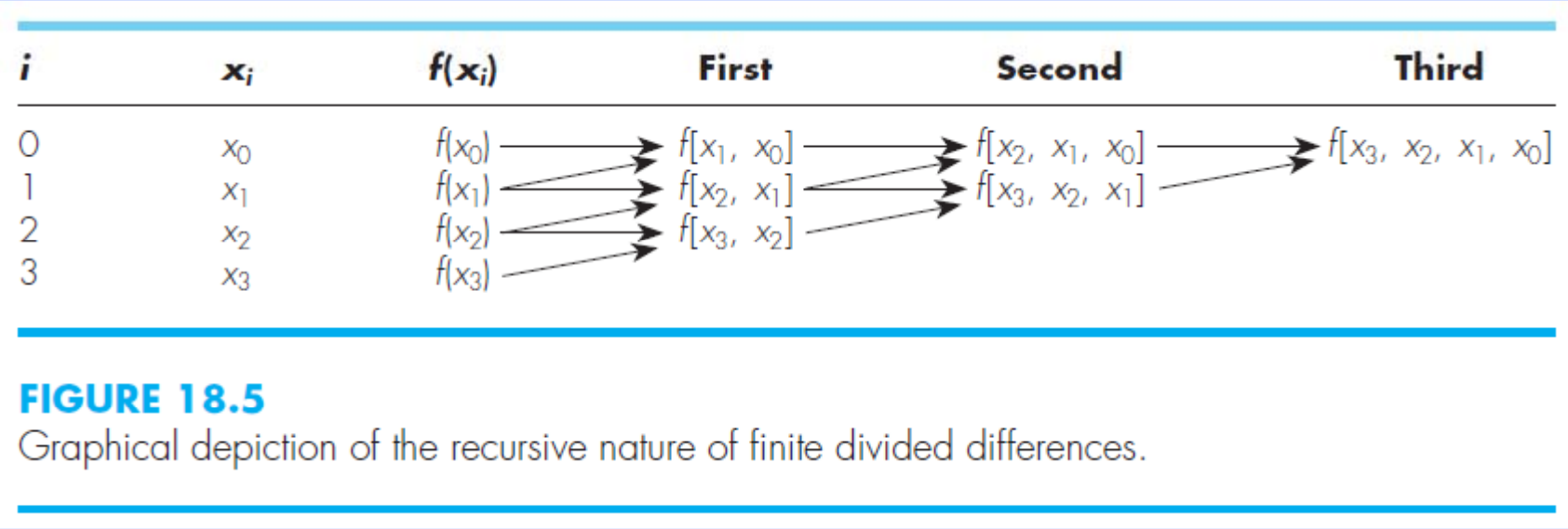


FIGURE 18.5
Graphical depiction of the recursive nature of finite divided differences.

In practice, the numbers generally decrease as we go right in the table. This means that the contribution of the higher order terms are less than the contribution of lower order terms.

A MATLAB algorithm to implement Newton's Interpolating Polynomials

Example 18.2 is tested with the following code. Note that, the code gives just *b* coefficients and the interpolated value of the function at some input *xx*.

```
x=[1 4 6]; % initial values
f=@(t) log(t);
y=f(x);
xx=2;
% 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 = initial points; y = f(x); xx = interpolation will be calculated at x=xx
% output: yint = interpolated value of xx  ( true value is ytrue=f(xx) )
n = length(x);
if length(y)~=n, error('x and y must be same length'); end
b = zeros(n,n);
% assign dependent variables to the first column of b.
b(:,1) = y(:); % the (:) ensures that y is a column vector.
for j = 2:n
    for i = 1:n-j+1
        b(i,j) = (b(i+1,j-1)-b(i,j-1))/(x(i+j-1)-x(i));
    end
end
% use the finite divided differences to interpolate
xt = 1;
yint = b(1,1);
for j = 1:n-1
    xt = xt*(xx-x(j));
    yint = yint+b(1,j+1)*xt;
end
out_b_coeffs=(b(1,:))
out_yint=yint
```

Homework:

Implement the previous code in MATLAB and try it with the Example 18.5 in your textbook. In each case, calculate the relative percent error. Compare your results with the textbook.

18.5 Error Estimates to Determine the Appropriate Order of Interpolation

Problem Statement. After incorporating the error [Eq. (18.18)], utilize the computer algorithm given in Fig. 18.7 and the following information to evaluate $f(x) = \ln x$ at $x = 2$:

x	$f(x) = \ln x$
1	0
4	1.3862944
6	1.7917595
5	1.6094379
3	1.0986123
1.5	0.4054641
2.5	0.9162907
3.5	1.2527630

Errors of Newton's Interpolating Polynomials/

- Structure of interpolating polynomials is similar to the Taylor series expansion in the sense that finite divided differences are added sequentially to capture the higher order derivatives.
- For an n^{th}-order interpolating polynomial, an analogous relationship for the error is:

$$R_n = \frac{f^{(n+1)}(\xi)}{(n+1)!}(x-x_0)(x-x_1)\cdots(x-x_n)$$

Ξξ ksi

ξ is somewhere between unknown and the data (*i.e.* between x_i and x_{i+1})

- For non differentiable functions, if an additional point $f(x_{n+1})$ is available, an alternative formula can be used that does not require prior knowledge of the function:

$$R_n \cong f[x_{n+1}, x_n, x_{n-1}, \ldots, x_0](x-x_0)(x-x_1)\cdots(x-x_n)$$

Lagrange Interpolating Polynomials

- The Lagrange interpolating polynomial is simply a reformulation of the Newton's polynomial that *avoids the computation of divided differences*. The *n*th order approximation can be represented as:

$$f_n(x) = \sum_{i=0}^{n} L_i(x) f(x_i)$$

$$L_i(x) = \prod_{\substack{j=0 \\ j \neq i}}^{n} \frac{x - x_j}{x_i - x_j}$$

$$f_1(x) = \frac{x - x_1}{x_0 - x_1} f(x_0) + \frac{x - x_0}{x_1 - x_0} f(x_1)$$

$$f_2(x) = \frac{(x - x_1)(x - x_2)}{(x_0 - x_1)(x_0 - x_2)} f(x_0) + \frac{(x - x_0)(x - x_2)}{(x_1 - x_0)(x_1 - x_2)} f(x_1) + \frac{(x - x_0)(x - x_1)}{(x_2 - x_0)(x_2 - x_1)} f(x_2)$$

• As with Newton's method, the Lagrange version has an estimated error of:

$$R_n = f[x, x_n, x_{n-1}, \ldots, x_0] \prod_{i=0}^{n} (x - x_i)$$

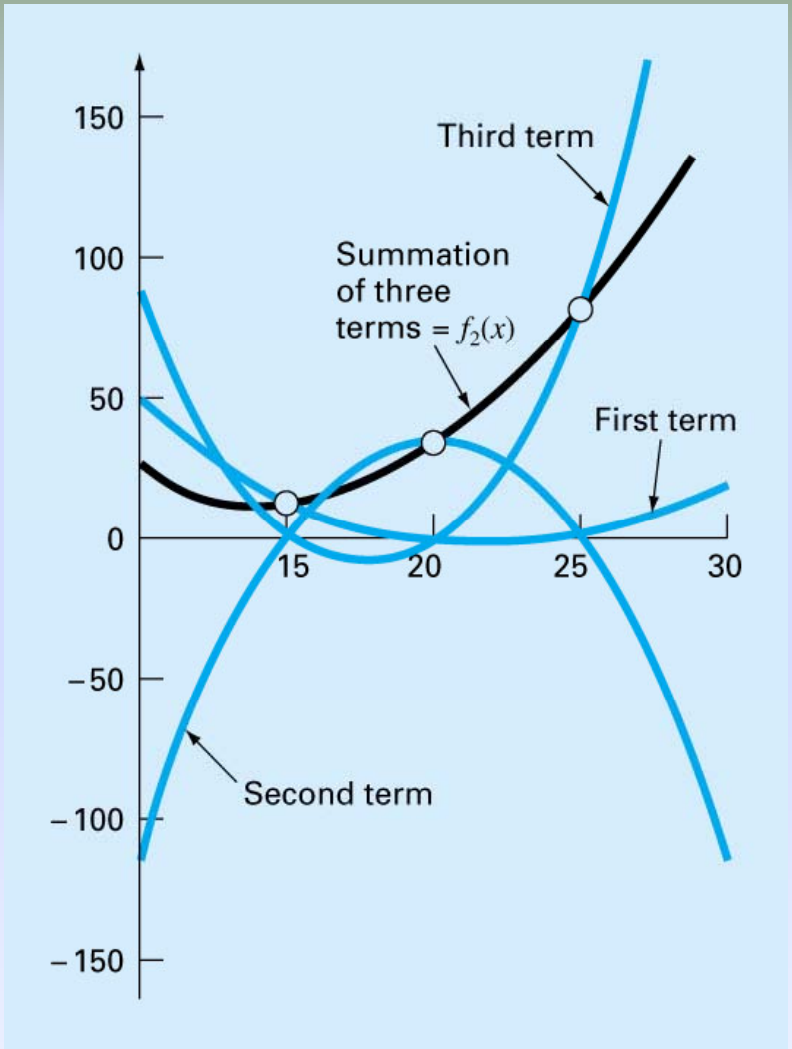


FIGURE 18.10
A visual depiction of the rationale behind the Lagrange polynomial. This figure shows a second-order case. Each of the three terms in Eq. (18.23) passes through one of the data points and is zero at the other two. The summation of the three terms must, therefore, be the unique second-order polynomial $f_2(x)$ that passes exactly through the three points.

A MATLAB algorithm to implement Lagrange Interpolating Polynomials

Example 18.5 is tested with the following code. Note that, the code gives just the interpolated value of the function at some input *xx*.

```
x=[1 4 6 5 3 1.5 2.5 3.5]; % initial values
f=@(t) log(t);
y=f(x);
xx=2;
% Lagrange interpolating polynomial
% Uses an (n - 1)-order Lagrange interpolating polynomial based on n data points
% to determine a value of the dependent variable (yint) at xx.
% inputs: x, y, xx
% output: yint = interpolated value of f at xx.
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
```

Example

x	f(x)
1	4.75
2	4.00
3	5.25
5	19.75
6	36.00

Calculate f(4) using Lagrange Interpolating Polynomials

(a) of order 1

(b) of order 2

(a) Linear interpolation. Select $x_0 = 3$, $x_1 = 5$

$f_1(x) = L_0(x)\, f(x_0) + L_1(x)\, f(x_1) = (x-5)/(3-5)\ 5.25 + (x-3)/(5-3)\ 19.75$

$f(4) \approx 12.5$

(b) Quadratic interpolation. Select $x_0 = 2$, $x_1 = 3$, $x_1 = 5$

$f_2(x) = L_0(x)\, f(x_0) + L_1(x)\, f(x_1) + L_2(x)\, f(x_2)$

$= (x-3)(x-5)/(2-3)(2-5)\ 4.00 + (x-2)(x-5)/(3-2)(3-5)\ 5.25 + (x-2)(x-3)/(5-2)(5-3)\ 19.75$

$f(4) \approx 10.5$

Coefficients of an Interpolating Polynomial

- Although both the Newton and Lagrange polynomials are well suited for determining intermediate values between points, they do not provide a polynomial in conventional form:

$$f(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_x x^n$$

- Since *n+1* data points are required to determine *n+1* coefficients, **simultaneous linear systems of equations** can be used to calculate "*a*"s.

$$f(x_0) = a_0 + a_1 x_0 + a_2 x_0^2 \cdots + a_n x_0^n$$

$$f(x_1) = a_0 + a_1 x_1 + a_2 x_1^2 \cdots + a_n x_1^n$$

$$\vdots$$

$$f(x_n) = a_0 + a_1 x_n + a_2 x_n^2 \cdots + a_n x_n^n$$

where "x_i"s are the knowns and "a_i"s are the unknowns.

Interpolation with equally spaced data

If data are equally spaced and in ascending order, then the independent variable assumes values of

$$x_1 = x_0 + h$$
$$x_2 = x_0 + 2h$$
$$\vdots$$
$$x_n = x_0 + nh$$

where h is the interval, or step size, between these data. On this basis, the finite divided differences can be expressed in concise form.

Interpolation with equally spaced data

$$f_n(x) = f(x_0) + \frac{\Delta f(x_0)}{h}(x - x_0)$$
$$+ \frac{\Delta^2 f(x_0)}{2!h^2}(x - x_0)(x - x_0 - h)$$
$$+ \cdots + \frac{\Delta^n f(x_0)}{n!h^n}(x - x_0)(x - x_0 - h)$$
$$\cdots[x - x_0 - (n - 1)h] + R_n$$

This equation is known as *Newton's formula,* or the *Newton-Gregory forward formula.*

It can be simplified further by defining a new quantity, α: $\alpha = \frac{x - x_0}{h}$

Interpolation with equally spaced data

$$f_n(x) = f(x_0) + \Delta f(x_0)\alpha + \frac{\Delta^2 f(x_0)}{2!}\alpha(\alpha - 1)$$

$$+ \cdots + \frac{\Delta^n f(x_0)}{n!}\alpha(\alpha - 1)\cdots(\alpha - n + 1) + R_n$$

where

$$R_n = \frac{f^{(n+1)}(\xi)}{(n+1)!}h^{n+1}\alpha(\alpha - 1)(\alpha - 2)\cdots(\alpha - n)$$

Prior to the advent of digital computers, these techniques had great utility for interpolation from tables with equally spaced arguments. In fact, a computational framework known as a divided-difference table was developed to facilitate the implementation of these techniques.

Extrapolation

- Extrapolation is the process of estimating a value of $f(x)$ that lies outside the range of the known base points, $x_0, x_1, \ldots, x_n$.
- The most accurate interpolation is usually obtained when the unknown lies near the center of the base points.
- Obviously, this is violated when the unknown lies outside the range, and consequently, the error in extrapolation can be very large.
- Extreme care should, therefore, be exercised whenever a case arises where one must extrapolate.

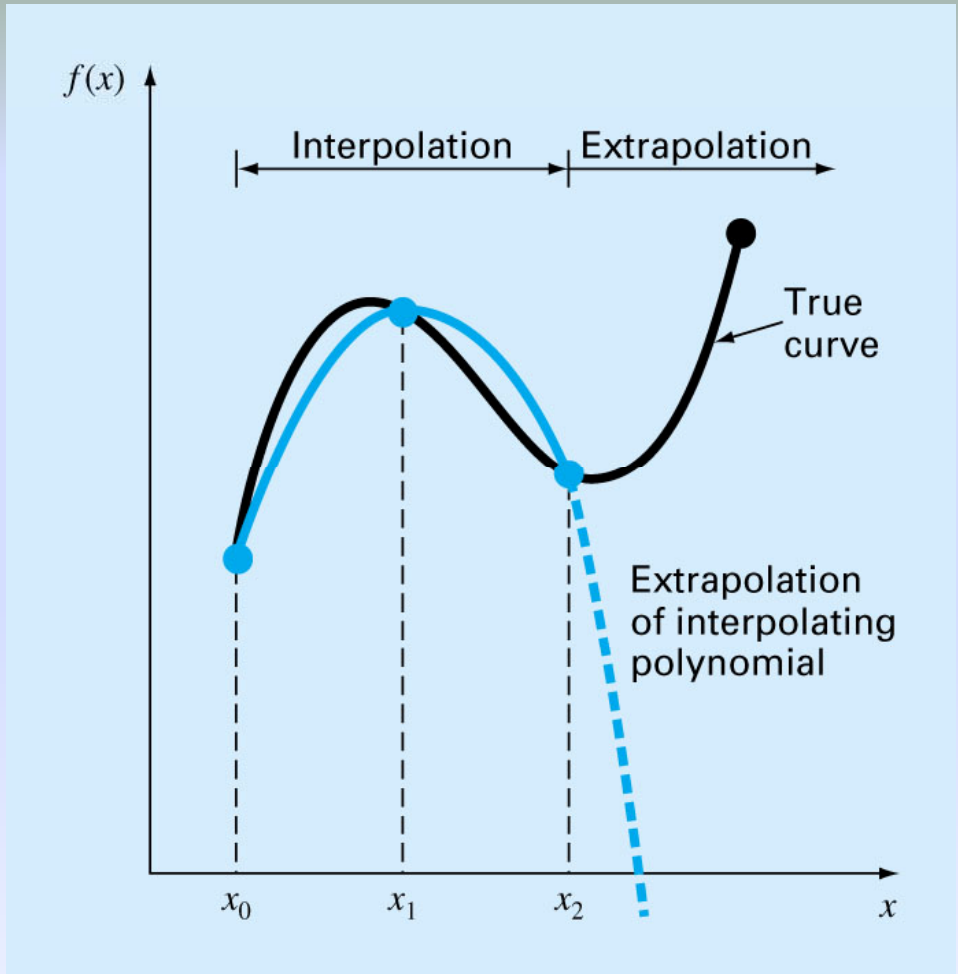
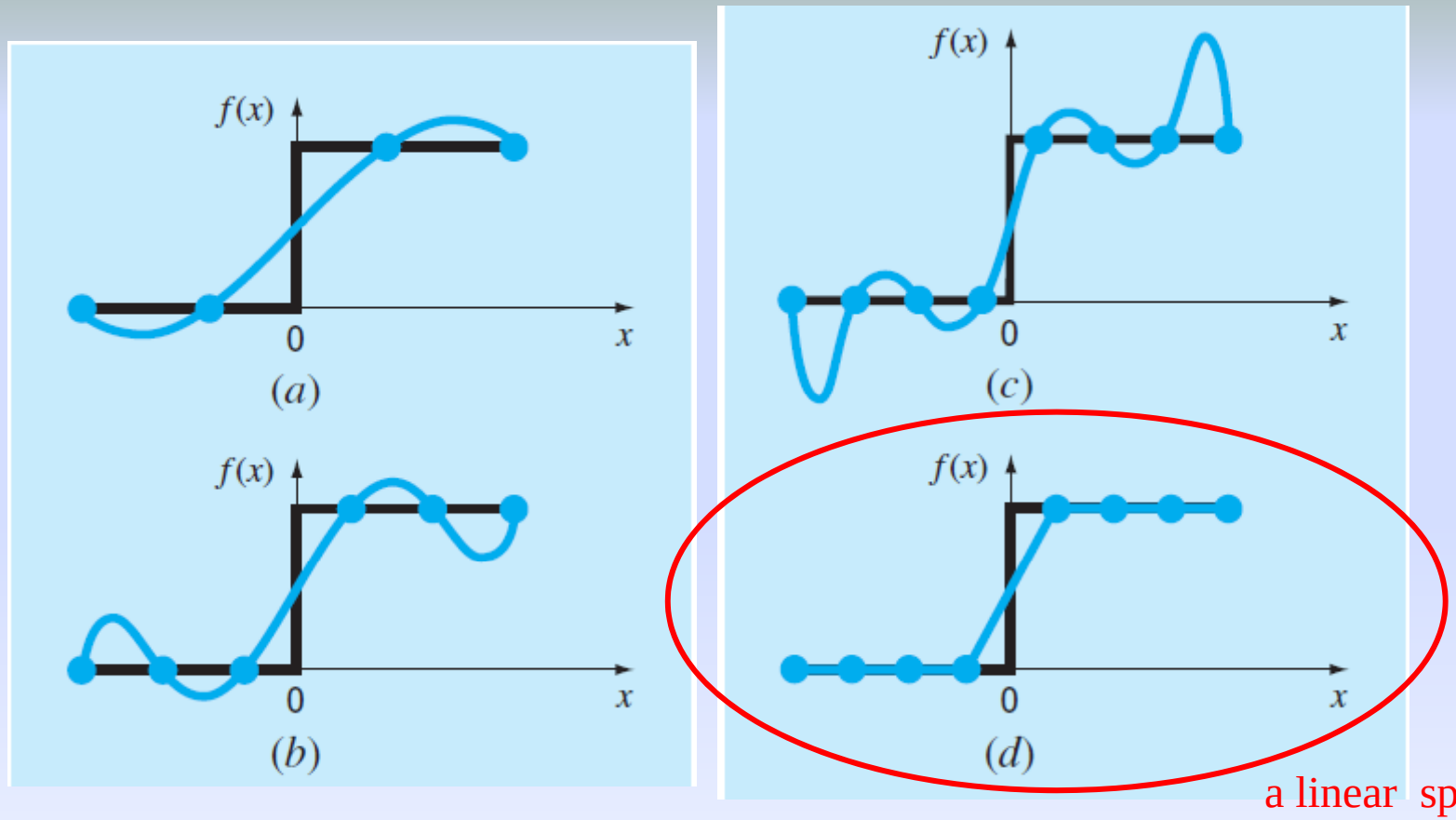


FIGURE 18.13
Illustration of the possible divergence of an extrapolated prediction. The extrapolation is based on fitting a parabola through the first three known points.

Spline Interpolation

- There are cases where polynomials can lead to erroneous results because of round off error and overshoot.
- Alternative approach is to apply lower-order polynomials to subsets of data points. Such connecting polynomials are called ***spline functions***.
- Instead of using a single high-order polynomial that passes through the data points, we can use different low-order polynomials between each data pair.



a linear spline

FIGURE 18.14

A visual representation of a situation where the splines are superior to higher-order interpolating polynomials. The function to be fit undergoes an abrupt increase at $x = 0$. Parts (*a*) through (*c*) indicate that the abrupt change induces oscillations in interpolating polynomials. In contrast, because it is limited to third-order curves with smooth transitions, a linear spline (*d*) provides a much more acceptable approximation.

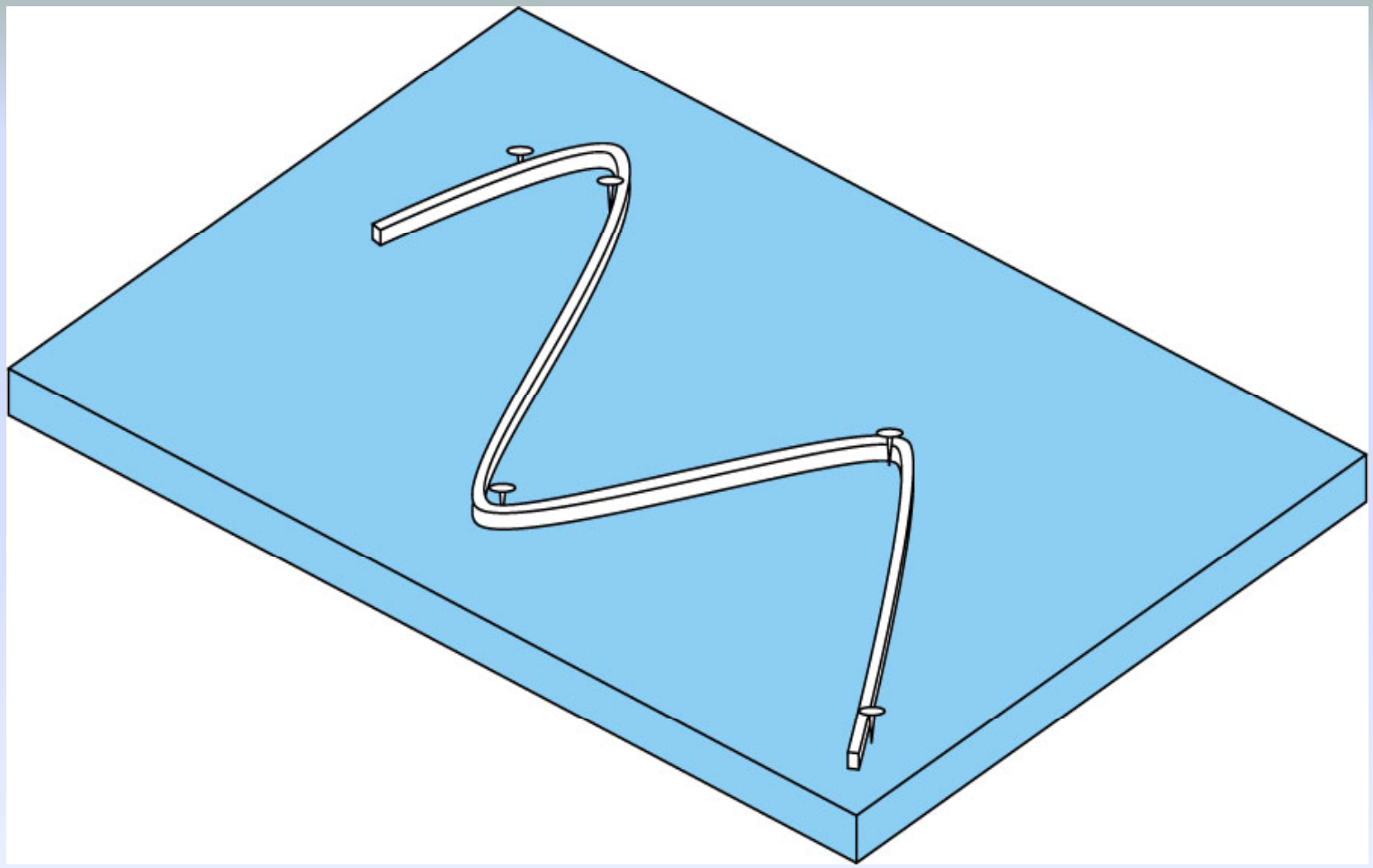
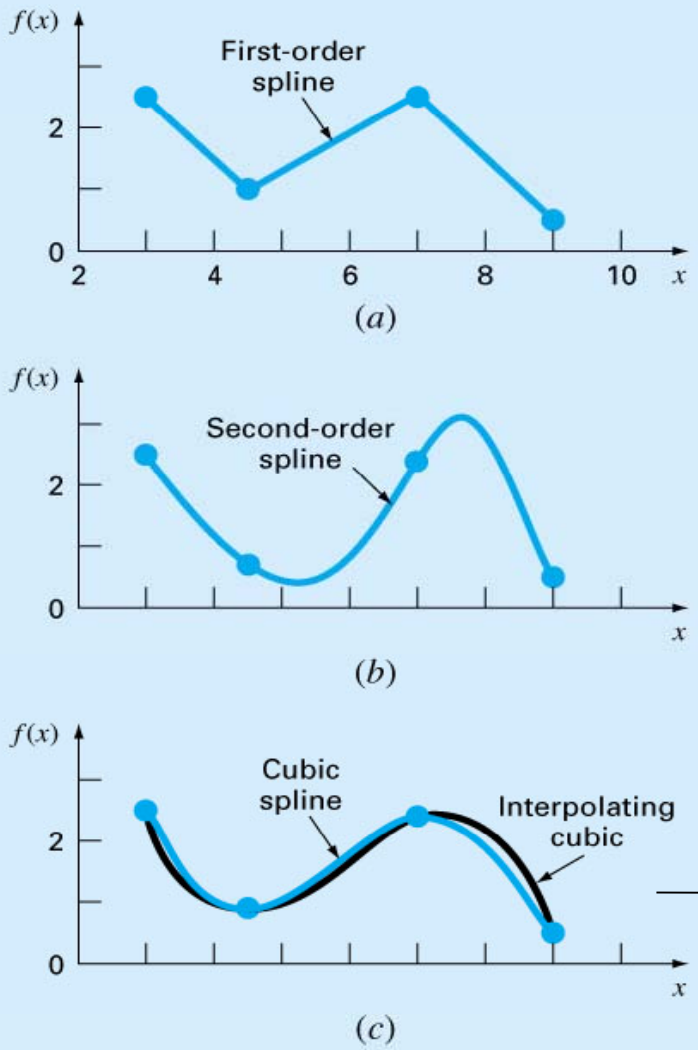


FIGURE 18.15
The drafting technique of using a spline to draw smooth curves through a series of points. Notice how, at the end points, the spline straightens out. This is called a "natural" spline.



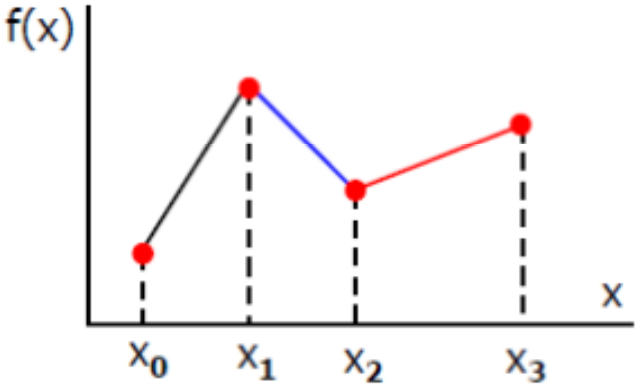
→ third-order curves employed to connect each pair of data points are called ***cubic splines***. Most widely used splines are these types.

FIGURE 18.16
Spline fits of a set of four points. (*a*) Linear spline, (*b*) quadratic spline, and (*c*) cubic spline, with a cubic interpolating polynomial also plotted.

Linear Splines/

- The simplest connection between two points is a straight line. The first-order splines for a group of ordered data points can be defined as a set of linear functions:

$$
\begin{aligned}
f(x) &= f(x_0) + m_0(x - x_0) && x_0 \le x \le x_1 \\
f(x) &= f(x_1) + m_1(x - x_1) && x_1 \le x \le x_2 \\
&\vdots \\
f(x) &= f(x_{n-1}) + m_{n-1}(x - x_{n-1}) && x_{n-1} \le x \le x_n
\end{aligned}
$$



where m_i is the slope of the straight line connecting the points:

$$m_i = \frac{f(x_{i+1}) - f(x_i)}{x_{i+1} - x_i}$$

Example:

First-Order Splines

Problem Statement. Fit the data in Table 18.1 with first-order splines. Evaluate the function at $x = 5$.

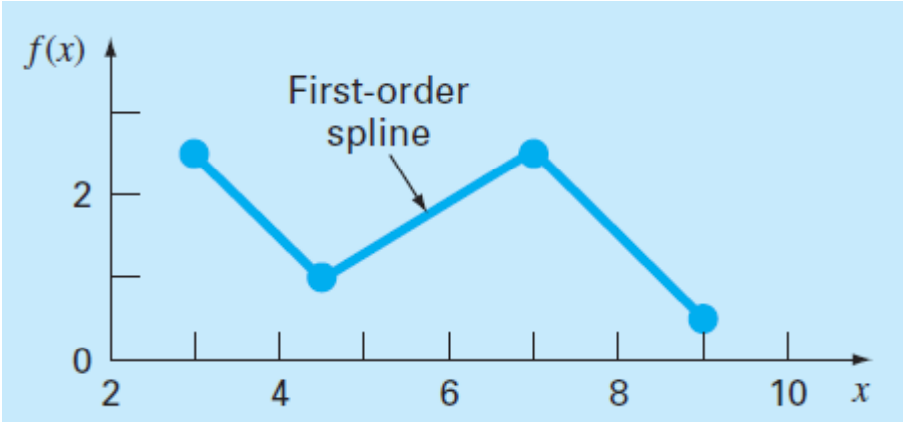
Solution. These data can be used to determine the slopes between points. For example, for the interval $x = 4.5$ to $x = 7$ the slope can be computed using Eq. (18.27):

$$m = \frac{2.5 - 7}{7 - 4.5} = 0.60$$

The slopes for the other intervals can be computed, and the resulting first-order splines are plotted in Fig. 18.16*a*. The value at $x = 5$ is 1.3.

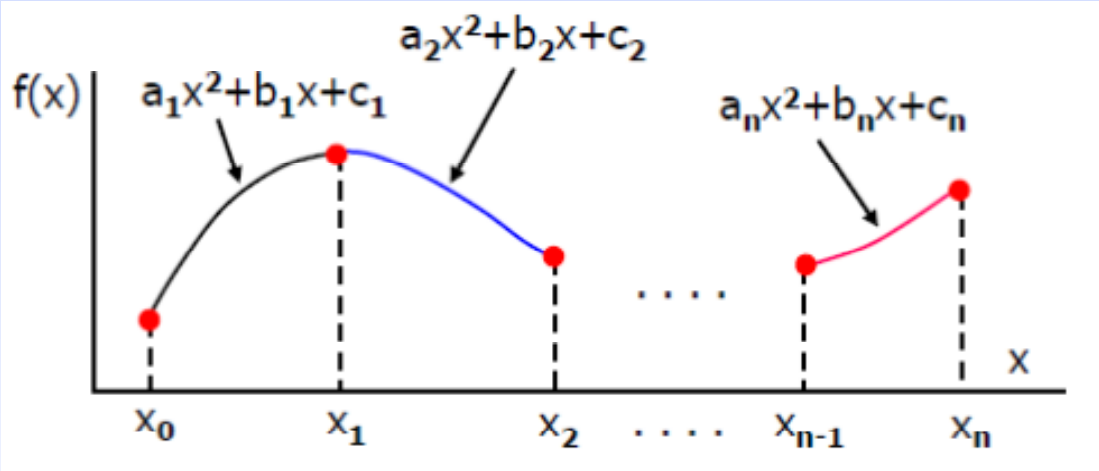
TABLE 18.1
Data to be fit with spline functions.

x	f(x)
3.0	2.5
4.5	1.0
7.0	2.5
9.0	0.5



Quadratic Splines/

- Every pair of points are connected by quadratic functions.
- For *n+1* data points, there are *n* splines and 3*n* unknowns.
- We need 3*n* equations to solve them.



Quadratic Splines/

These 3*n* equations can be found as given in the following:

1. The function values of adjacent polynomials must be equal at the interior knots.
2. The first and last functions must pass through the end points.
3. The first derivatives at the interior knots must be equal.

The first 3 conditions supply 3*n*–1 equations. The last equation can be chosen through an arbitrary assumption:

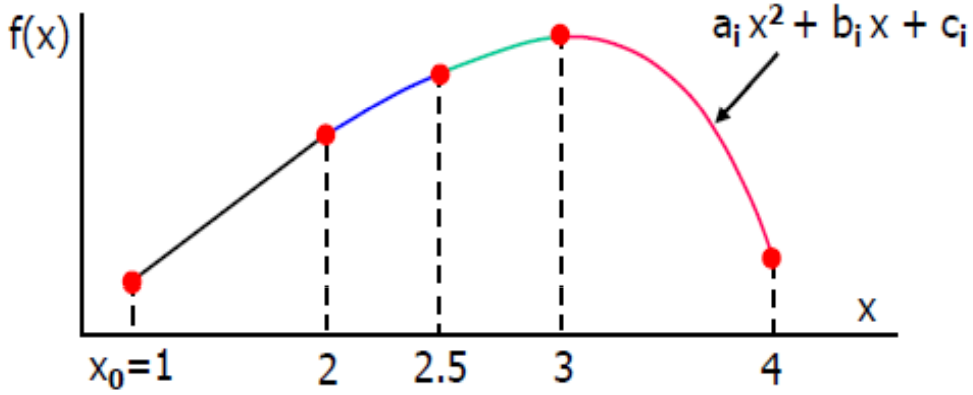
4. Assume that the second derivative is zero at the first point.

The visual interpretation of this condition is that the first two points will be connected by a straight line.

Example:

x	f(x)
1	1
2	5
2.5	7
3	8
4	2

Develop quadratic splines for these data points and predict f(3.4) and f(2.2)



- There are 5 points and n=4 splines. Totally there are 3n=12 unknowns. Equations are
- End points: $a_1 1^2 + b_1 1 + c_1 = 1$, $a_4 4^2 + b_4 4 + c_4 = 2$
- Interior points: $a_1 2^2 + b_1 2 + c_1 = 5$, $a_2 2^2 + b_2 2 + c_2 = 5$

  $a_2 2.5^2 + b_2 2.5 + c_2 = 7$, $a_3 2.5^2 + b_3 2.5 + c_3 = 7$

  $a_3 3^2 + b_3 3 + c_3 = 8$, $a_4 3^2 + b_4 3 + c_4 = 8$
- Derivatives at the interior points: $2a_1 2 + b_1 = 2a_2 2 + b_2$

  $2a_2 2.5 + b_2 = 2a_3 2.5 + b_3$

  $2a_3 3 + b_3 = 2a_4 3 + b_4$
- Arbitrary choice for the missing equation: $a_1 = 0$

Example - continued:

- a_1=0 is already known. Solve for the remaining 11 unknowns.

$$\begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 16 & 4 & 1 \\ 2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 4 & 2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 6.25 & 2.5 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 6.25 & 2.5 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 9 & 3 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 9 & 3 & 1 \\ 1 & 0 & -4 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 5 & 1 & 0 & -5 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 6 & 1 & 0 & -6 & -1 & 0 \end{bmatrix} \begin{Bmatrix} b_1 \\ c_1 \\ a_2 \\ b_2 \\ c_2 \\ a_3 \\ b_3 \\ c_3 \\ a_4 \\ b_4 \\ c_4 \end{Bmatrix} = \begin{Bmatrix} 1 \\ 2 \\ 5 \\ 5 \\ 7 \\ 7 \\ 8 \\ 8 \\ 0 \\ 0 \\ 0 \end{Bmatrix} \rightarrow \begin{Bmatrix} b_1 \\ c_1 \\ a_2 \\ b_2 \\ c_2 \\ a_3 \\ b_3 \\ c_3 \\ a_4 \\ b_4 \\ c_4 \end{Bmatrix} = \begin{Bmatrix} 4 \\ -3 \\ 0 \\ 4 \\ -3 \\ -4 \\ 24 \\ -28 \\ -6 \\ 36 \\ 46 \end{Bmatrix}$$

- Equations for the splines are

  1st spline: $f(x) = 4x - 3$ (Straight line.)

  2nd spline: $f(x) = 4x - 3$ (Same as the 1st. Coincidence)

  3rd spline: $f(x) = -4x^2 + 24x - 28$

  4th spline: $f(x) = -6x^2 + 36x - 46$

- To predict f(3.4) use the 4th spline. $f(3.4) = -6(3.4)^2 + 36(3.4) - 46 = 7.04$

  To predict f(2.2) use the 2nd spline. $f(2.2) = 4(2.2) - 3 = 5.8$

Cubic Splines/

- The objective in cubic splines is to derive a third-order polynomial for each interval between knots.

$$f_i(x) = a_i x^3 + b_i x^2 + c_i x + d_i$$

- Thus, for n+1 data points (i = 0, 1, 2, . . . , n), there are n intervals and, consequently, 4n unknown constants to evaluate:

1. The function values must be equal at the interior knots (2n–2 conditions).

2. The first and last functions must pass through the end points (2 conditions).

3. The first derivatives at the interior knots must be equal (n–1 conditions).

4. The second derivatives at the interior knots must be equal (n–1 conditions).

5. The second derivatives at the end knots are zero (2 conditions).

Cubic Splines/

- The above five types of conditions provide the total of $4n$ equations required to solve for the $4n$ coefficients. This is a costly procedure.
- Whereas there is an alternative technique that requires the solution of only $n-1$ equations.
- The first step in the derivation is based on the observation that because each pair of knots is connected by a cubic, the second derivative within each interval is a straight line.
- On this basis, the second derivatives can be represented by a first-order Lagrange interpolating polynomial:

$$f_i''(x) = f_i''(x_{i-1})\frac{x - x_i}{x_{i-1} - x_i} + f_i''(x_i)\frac{x - x_{i-1}}{x_i - x_{i-1}}$$

Let's integrate this twice to yield an expression for $f_i(x)$.

Cubic Splines/

$$f_i(x) = \frac{f_i''(x_i)}{6(x_i - x_{i-1})}(x_i - x)^3 + \frac{f_i''(x_i)}{6(x_i - x_{i-1})}(x - x_{i-1})^3 + \left[\frac{f(x_{i-1})}{x_i - x_{i-1}} - \frac{f''(x_{i-1})(x_i - x_{i-1})}{6}\right](x_i - x) + \left[\frac{f(x_i)}{x_i - x_{i-1}} - \frac{f''(x_i)(x_i - x_{i-1})}{6}\right](x - x_{i-1})$$

This equation contains only two unknowns, the second derivatives at the end of each interval. These unknowns can be evaluated using the following equation:

$$(x_i - x_{i-1})f''(x_{i-1}) + 2(x_{i+1} - x_{i-1})f''(x_i) + (x_{i+1} - x_i)f''(x_{i+1}) = \frac{6}{x_{i+1} - x_i}[f(x_{i+1}) - f(x_i)] + \frac{6}{x_i - x_{i-1}}[f(x_{i-1}) - f(x_i)]$$

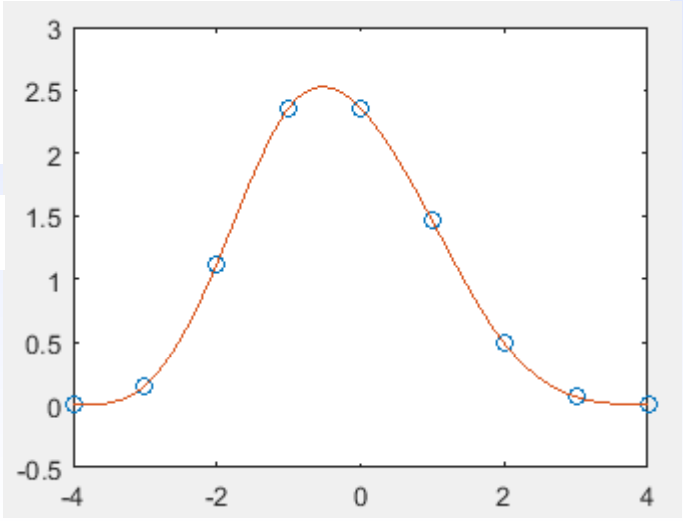
If this equation is written for all interior points, n–1 simultaneous equations result with n–1 unknowns. The second derivatives at the end points are zero.

MATLAB Function: spline

```
>> help spline
 spline Cubic spline data interpolation.
    PP = spline(X,Y) provides the piecewise polynomial form of the
    cubic spline interpolant to the data values Y at the data sites X,
    for use with the evaluator PPVAL and the spline utility UNMKPP.
    X must be a vector.
```

```
Example:
This illustrates the use of clamped or complete spline interpolation where
end slopes are prescribed. In this example, zero slopes at the ends of an
interpolant to the values of a certain distribution are enforced:
   x = -4:4; y = [0 .15 1.12 2.36 2.36 1.46 .49 .06 0];
   cs = spline(x,[0 y 0]);
   xx = linspace(-4,4,101);
   plot(x,y,'o',xx,ppval(cs,xx),'-');
```

See also interp1, mkpp, pchip, ppval, unmkpp



Homework:

Cubic Splines

Problem Statement. Fit cubic splines to the same data used in Examples 18.8 and 18.9 (Table 18.1). Utilize the results to estimate the value at $x = 5$.

TABLE 18.1
Data to be fit with spline functions.

x	$f(x)$
3.0	2.5
4.5	1.0
7.0	2.5
9.0	0.5

MATLAB Function: interp1

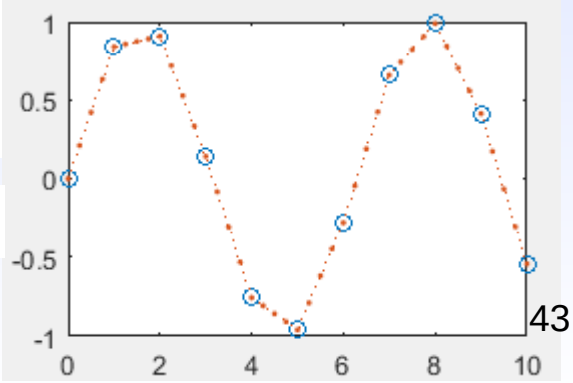
```
>> help interp1
 interp1 1-D interpolation (table lookup)

    Vq = interp1(X,V,Xq) interpolates to find Vq, the values of the
    underlying function V=F(X) at the query points Xq.

    X must be a vector. The length of X is equal to N.
    If V is a vector, V must have length N, and Vq is the same size as Xq.
    If V is an array of size [N,D1,D2,...,Dk], then the interpolation is
    performed for each D1-by-D2-by-...-Dk value in V(i,:,:,...,:). If Xq
    is a vector of length M, then Vq has size [M,D1,D2,...,Dk]. If Xq is
    an array of size [M1,M2,...,Mj], then Vq is of size
    [M1,M2,...,Mj,D1,D2,...,Dk].
```

For example, generate a coarse sine curve and interpolate over a finer abscissa:

```
    X = 0:10; V = sin(X); Xq = 0:.25:10;
    Vq = interp1(X,V,Xq); plot(X,V,'o',Xq,Vq,':.')
```



See also griddedInterpolant, interp2, interp3, interpn

Multidimensional Interpolation/

- The interpolation methods for one-dimensional problems can be extended to multidimensional interpolation.
- The simplest case is two-dimensional interpolation in Cartesian coordinates.

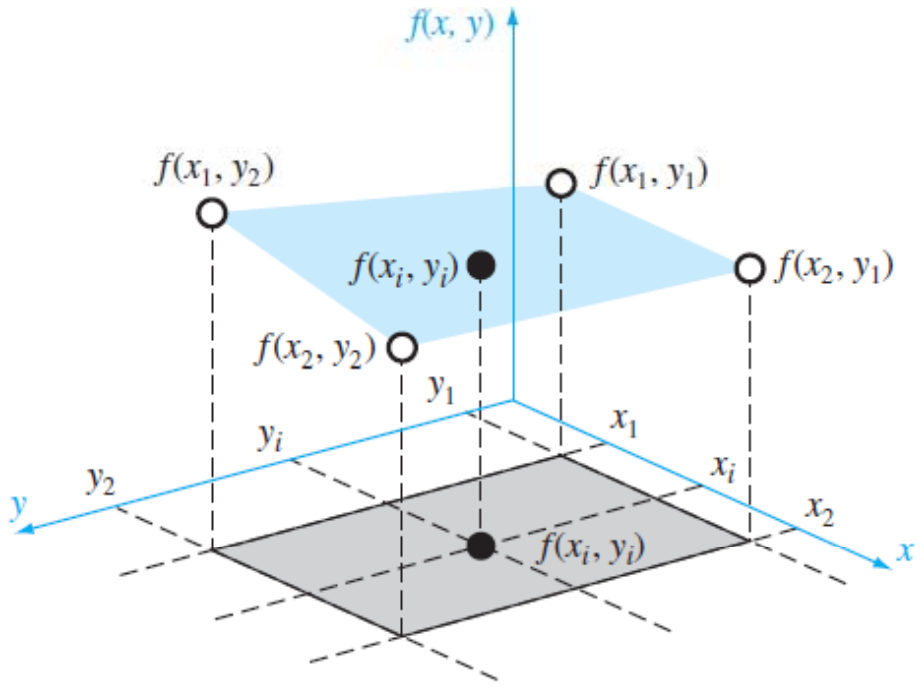


FIGURE 18.19
Graphical depiction of two-dimensional bilinear interpolation where an intermediate value (filled circle) is estimated based on four given values (open circles).

Fourier Approximation

Chapter 19

- Engineers often deal with systems that *oscillate* or *vibrate*.
- Therefore trigonometric functions play a fundamental role in modeling such problems.
- Fourier approximation represents a systemic framework for using trigonometric series for this purpose.

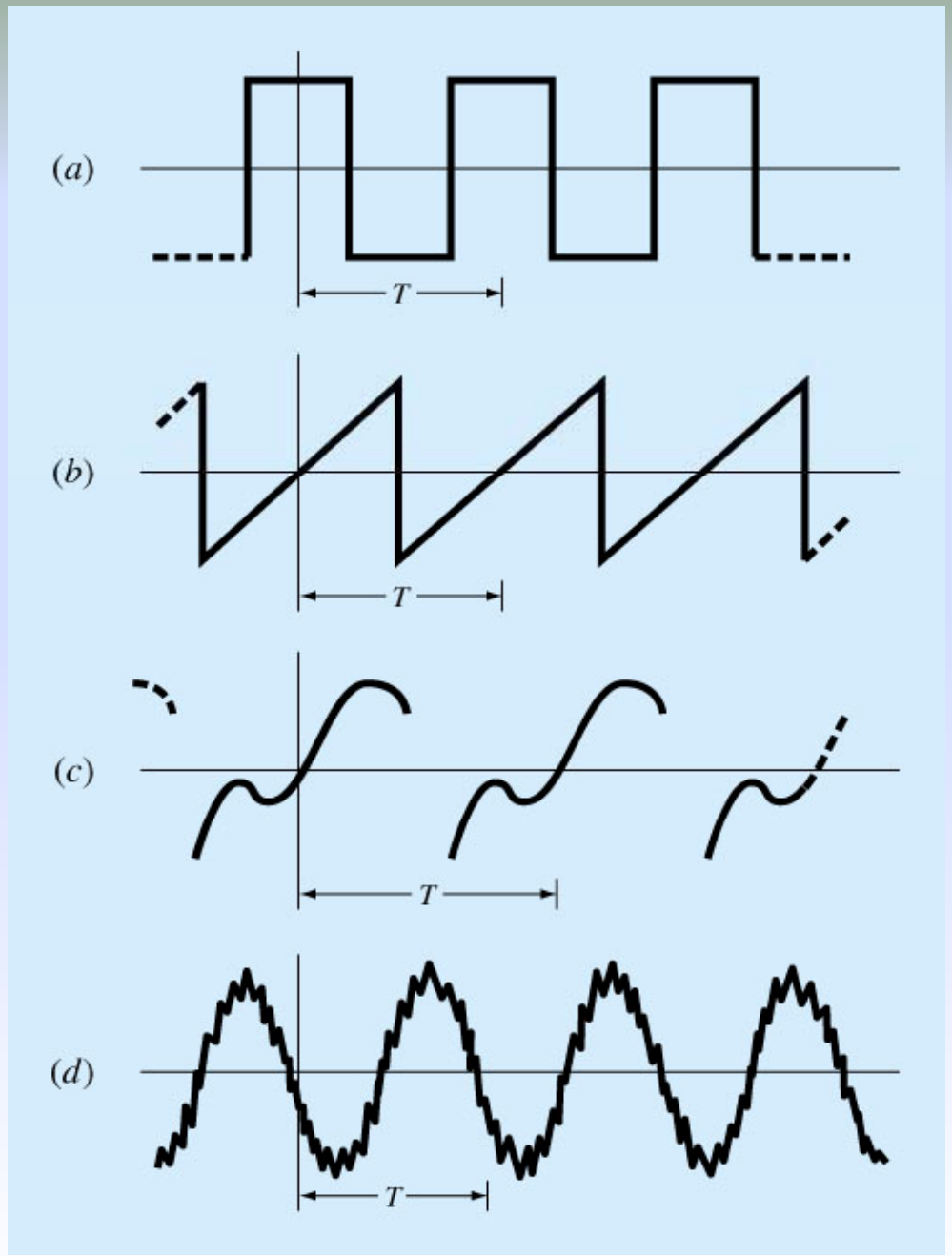


FIGURE 19.2
Aside from trigonometric functions such as sines and cosines, periodic functions include waveforms such as (*a*) the square wave and (*b*) the sawtooth wave. Beyond these idealized forms, periodic signals in nature can be (*c*) non-ideal and (*d*) contaminated by noise. The trigonometric functions can be used to represent and to analyze all these cases.

Curve Fitting with Sinusoidal Functions

- A periodic function *f(t)* is one for which

$$f(t) = f(t+T)$$

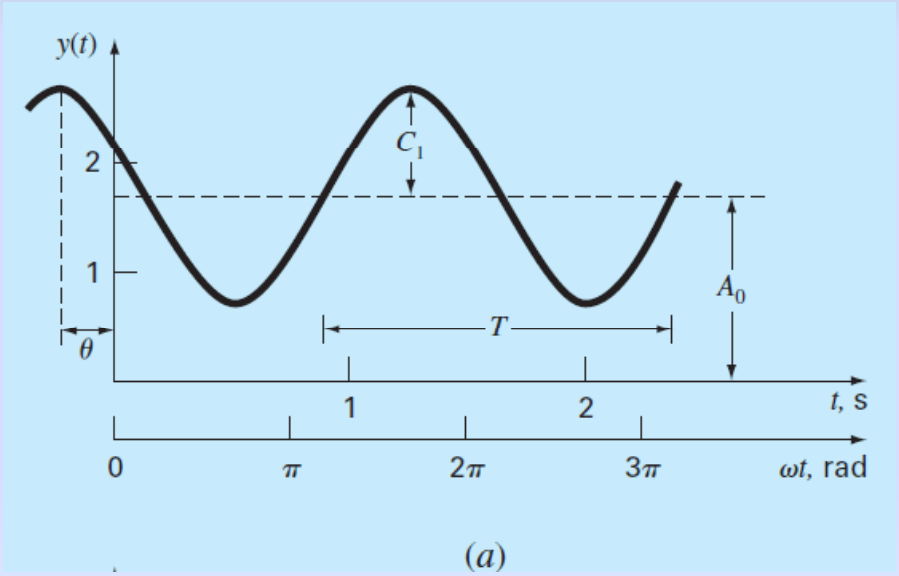
where *T* is a constant called the *period* that is the smallest value for which this equation holds.

- Any waveform that can be described as a sine or cosine is called sinusoid:

$$f(t) = A_0 + C_1 \cos(\omega_0 t + \theta)$$

Four parameters serve to characterize the sinusoid. The *mean value* A_0 sets the average height above the abscissa. The *amplitude* C_1 specifies the height of the oscillation. The *angular frequency* ω_0 characterizes how often the cycles occur. The *phase angle*, or *phase shift*, θ parameterizes the extent which the sinusoid is shifted horizontally.

Addition of these 3 function gives the function shown in (a)



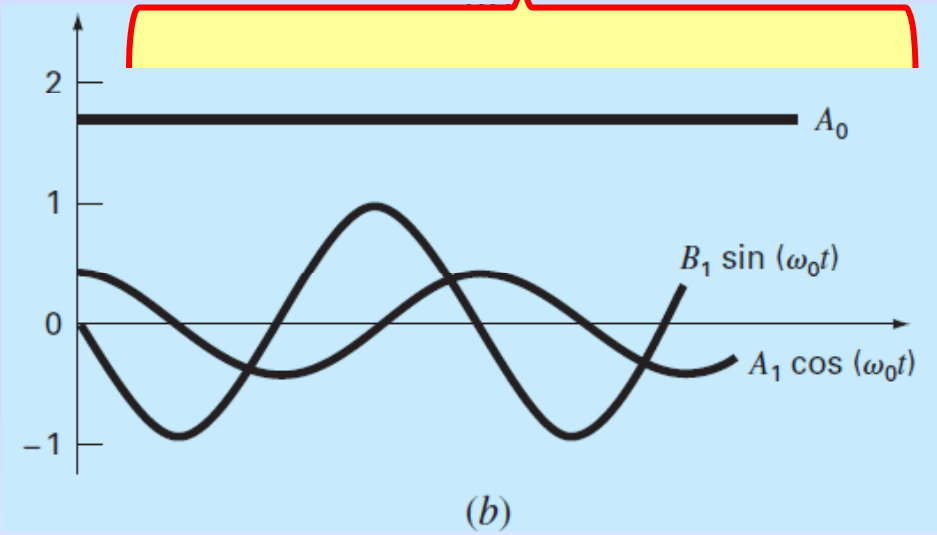
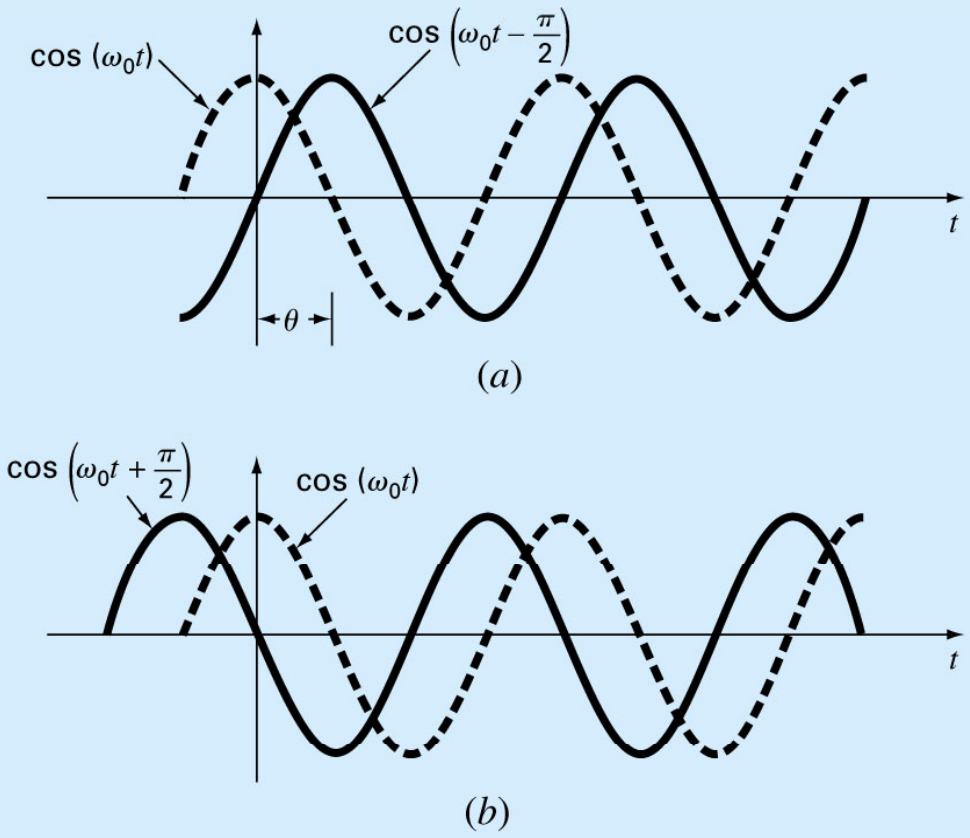


FIGURE 19.3

(*a*) A plot of the sinusoidal function $y(t) = A_0 + C_1 \cos(\omega_0 t + \theta)$. For this case, $A_0 = 1.7$, $C_1 = 1$, $\omega_0 = 2\pi/T = 2\pi/(1.5\text{ s})$, and $\theta = \pi/3$ radians $= 1.0472$ $(= 0.25\text{ s})$. Other parameters used to describe the curve are the frequency $f = \omega_0/(2\pi)$, which for this case is 1 cycle/(1.5 s) and the period $T = 1.5$ s. (*b*) An alternative expression of the same curve is $y(t) = A_0 + A_1 \cos(\omega_0 t) + B_1 \sin(\omega_0 t)$. The three components of this function are depicted in (*b*), where $A_1 = 0.5$ and $B_1 = -0.866$. The summation of the three curves in (*b*) yields the single curve in (*a*).

As depicted in Fig. 19.4*a*, a negative value is referred to as a *lagging phase angle* because the curve $\cos(\omega_0 t - \theta)$ begins a new cycle θ radians after $\cos(\omega_0 t)$. Thus, $\cos(\omega_0 t - \theta)$ is said to lag $\cos(\omega_0 t)$. Conversely, as in Fig. 19.4*b*, a positive value is referred to as a leading phase angle.



$$\omega_0 = 2\pi f$$

$$f = \frac{1}{T}$$

FIGURE 19.4
Graphical depictions of (*a*) a lagging phase angle and (*b*) a leading phase angle. Note that the lagging curve in (*a*) can be alternatively described as $\cos(\omega_0 t + 3\pi/2)$. In other words, if a curve lags by an angle of α, it can also be represented as leading by $2\pi - \alpha$.

- An alternative model that still requires four parameters but that is cast in the format of a general linear model can be obtained by invoking the trigonometric identity:

$$C_1 \cos(\omega_0 t + \theta) = C_1[\cos(\omega_0 t)\cos(\theta) - \sin(\omega_0 t)\sin(\theta)]$$

$$f(t) = A_0 + A_1 \cos(\omega_0 t) + B_1 \sin(\omega_0 t)$$

where

$$A_1 = C_1 \cos(\theta) \qquad B_1 = -C_1 \sin(\theta)$$

$$\theta = \arctan\left(-\frac{B_1}{A_1}\right)$$

$$C_1 = \sqrt{A_1^2 + B_1^2}$$

Least-squares Fit of a Sinusoid/

- Sinusoid equation can be thought of as a linear least-squares model

$$y = A_0 + A_1 \cos(\omega_0 t) + B_1 \sin(\omega_0 t) + e$$

$$y = a_0 z_0 + a_1 z_1 + a_2 z_2 + \cdots + a_m z_m + e$$

$$z_0 = 1, \quad z_1 = \cos(\omega_0 t), \ldots$$

- Thus our goal is to determine coefficient values that minimize

$$S_r = \sum_{i=1}^{N} \{y_i - [A_0 + A_1 \cos(\omega_0 t_i) + B_1 \sin(\omega_0 t_i)]\}^2$$

Least-squares Fit of a Sinusoid/

Thus, for equispaced points the normal equations become

$$\begin{bmatrix} N & 0 & 0 \\ 0 & N/2 & 0 \\ 0 & 0 & N/2 \end{bmatrix} \begin{Bmatrix} A_0 \\ A_1 \\ B_1 \end{Bmatrix} = \begin{Bmatrix} \Sigma y \\ \Sigma y \cos(\omega_0 t) \\ \Sigma y \sin(\omega_0 t) \end{Bmatrix}$$

The inverse of a diagonal matrix is merely another diagonal matrix whose elements are the reciprocals of the original. Thus, the coefficients can be determined as

$$\begin{Bmatrix} A_0 \\ A_1 \\ B_1 \end{Bmatrix} = \begin{bmatrix} 1/N & 0 & 0 \\ 0 & 2/N & 0 \\ 0 & 0 & 2/N \end{bmatrix} \begin{Bmatrix} \Sigma y \\ \Sigma y \cos(\omega_0 t) \\ \Sigma y \sin(\omega_0 t) \end{Bmatrix}$$

or

$$A_0 = \frac{\Sigma y}{N} \qquad A_1 = \frac{2}{N}\Sigma y \cos(\omega_0 t) \qquad B_1 = \frac{2}{N}\Sigma y \sin(\omega_0 t)$$

Least-squares Fit : $y = A_0 + A_1 \cos(\omega_0 t) + B_1 \sin(\omega_0 t)$

Example:

Least-Squares Fit of a Sinusoid

Problem Statement. The curve in Fig. 19.3 is described by $y = 1.7 + \cos(4.189t + 1.0472)$. Generate 10 discrete values for this curve at intervals of $\Delta t = 0.15$ for the range $t = 0$ to 1.35. Use this information to evaluate the coefficients of Eq. (16.11) by a least-squares fit.

Solution. The data required to evaluate the coefficients with $\omega = 4.189$ are

t	y	$y\cos(\omega_0 t)$	$y\sin(\omega_0 t)$
0	2.200	2.200	0.000
0.15	1.595	1.291	0.938
0.30	1.031	0.319	0.980
0.45	0.722	−0.223	0.687
0.60	0.786	−0.636	0.462
0.75	1.200	−1.200	0.000
0.90	1.805	−1.460	−1.061
1.05	2.369	−0.732	−2.253
1.20	2.678	0.829	−2.547
1.35	2.614	2.114	−1.536
$\Sigma =$	17.000	2.502	−4.330

$$A_0 = \frac{17.000}{10} = 1.7 \qquad A_1 = \frac{2}{10}2.502 = 0.500 \qquad B_1 = \frac{2}{10}(-4.330) = -0.866$$

$$\Rightarrow \quad y = 1.7 + 0.500\cos(\omega_0 t) - 0.866\sin(\omega_0 t)$$

Continuous Fourier Series

- In course of studying heat-flow problems, Fourier showed that an arbitrary periodic function can be represented by an infinite series of sinusoids of harmonically related frequencies.
- For a function with period T, a continuous Fourier series can be written:

$$f(t) = a_0 + a_1 \cos(\omega_0 t) + b_1 \sin(\omega_0 t) + a_2 \cos(2\omega_0 t) + b_2 \sin(2\omega_0 t) + \cdots$$

more concisely

$$f(t) = a_0 + \sum_{k=1}^{\infty} \left[a_k \cos(k\omega_0 t) + b_k \sin(k\omega_0 t) \right]$$

- Where $\omega_0 = 2\pi/T$ is called *fundamental frequency* and its constant multiples $2\omega_0$, $3\omega_0$, etc., are called *harmonics*.
- The coefficients of the equation can be calculated as follows:

$$a_k = \frac{2}{T}\int_0^T f(t)\cos(k\omega_0 t)dt$$

$$b_k = \frac{2}{T}\int_0^T f(t)\sin(k\omega_0 t)dt$$

$$\text{for } k = 1,\ 2,\ \ldots$$

$$a_0 = \frac{1}{T}\int_0^T f(t)dt$$

Frequency and Time Domains

- Although it is not as familiar, the *frequency domain* provides an alternative perspective for characterizing the behavior of oscillating functions.
- Just as an amplitude can be plotted versus time, it can also be plotted against frequency. In such a plot, the magnitude or amplitude of the curve, *f(t)*, is the dependent variable and time *t* and frequency $f = \omega_0/2\pi$ are independent variables. Thus, the amplitude and time axis form a *time plane*, and the amplitude and the frequency axes form a *frequency plane*.

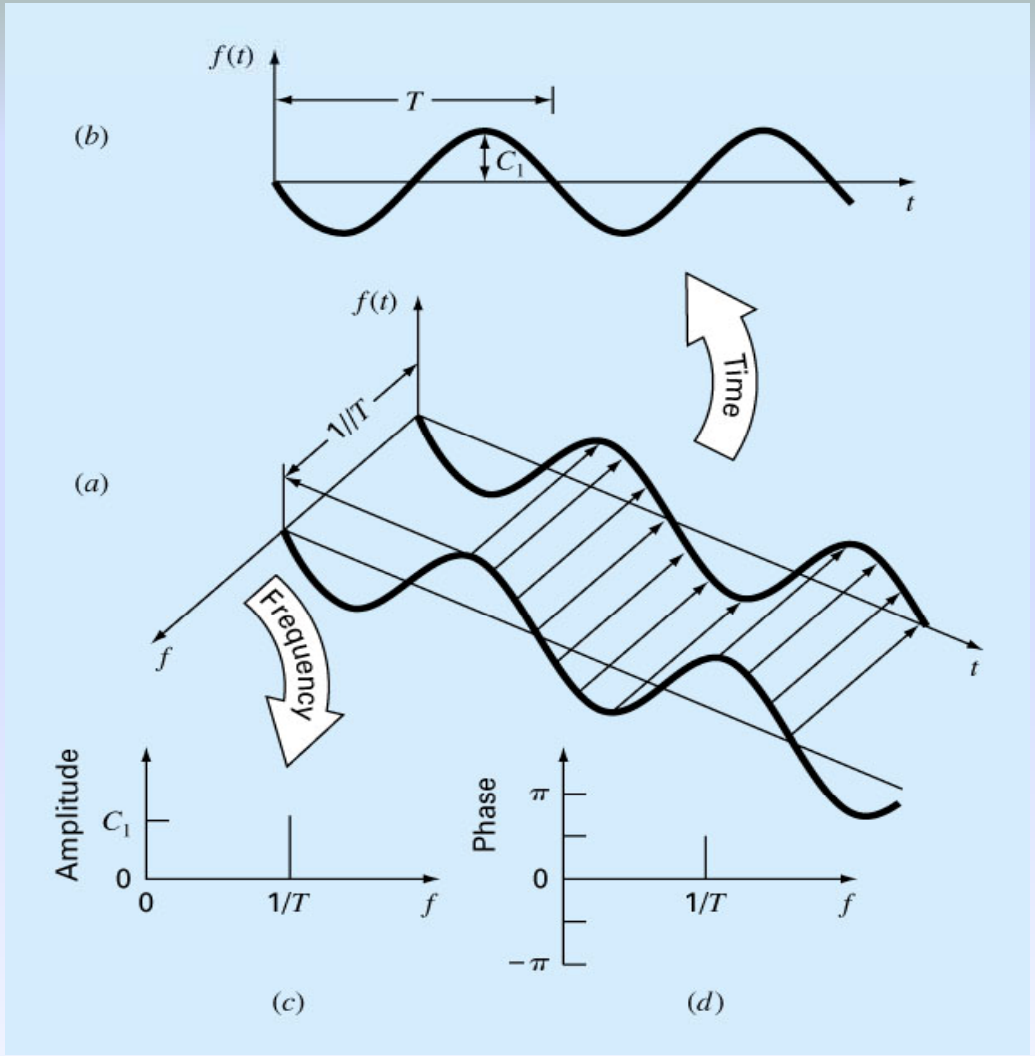


FIGURE 19.7

(*a*) A depiction of how a sinusoid can be portrayed in the time and the frequency domains. The time projection is reproduced in (*b*), whereas the amplitude-frequency projection is reproduced in (*c*). The phase-frequency projection is shown in (*d*).

Fourier Integral and Transform

- Although the Fourier series is useful tool for investigating the spectrum of a periodic function, there are many waveforms that do not repeat themselves regularly, such as a signal produced by a lightning bolt.
- Such a nonrecurring signal exhibits a continuous frequency spectrum and the *Fourier integral* is the primary tool available for this purpose.

$$f(t) = \sum_{k=-\infty}^{\infty} \tilde{c}_k e^{ik\omega_0 t}$$

$$\tilde{c}_k = \frac{1}{T} \int_{-T/2}^{T/2} f(t) e^{ik\omega_0 t} dt$$

$$\omega_0 = 2\pi / T \quad k = 0,1,2,\ldots$$

• The transition from a periodic to a nonperiodic function can be affected by allowing the period to approach infinity. In other words, as T becomes infinite, the function never repeats itself and thus becomes aperiodic. Then the Fourier series reduces to:

Inverse Fourier transform of $F(i\omega_0)$

$$f(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} F(i\omega_0) e^{i\omega_0 t} d\omega_0$$

$$F(i\omega_0) = \int_{-\infty}^{\infty} f(t) e^{i\omega_0 t} dt$$

Fourier transform pair

Fourier integral of $f(t)$, or Fourier transform of $f(t)$

Distinction between Fourier Series and Fourier Transform

- each applies to a different class of functions
  - the series to periodic and
  - the transform to nonperiodic waveforms
- The two approaches differ in how they move between the time and the frequency domains.
  - The Fourier series converts a continuous, periodic time-domain function to frequencydomain magnitudes at discrete frequencies.
  - The Fourier transform converts a continuous time-domain function to a continuous frequency-domain function.
- The discrete frequency spectrum generated by the Fourier series; a continuous frequency spectrum generated by the Fourier transform.

Discrete Fourier Transform (DFT)

- In engineering, functions are often represented by finite sets of discrete values and data is often collected in or converted to such a discrete format.
- An interval from 0 to t can be divided into N equispaced subintervals with widths of $\Delta t = T/N$.

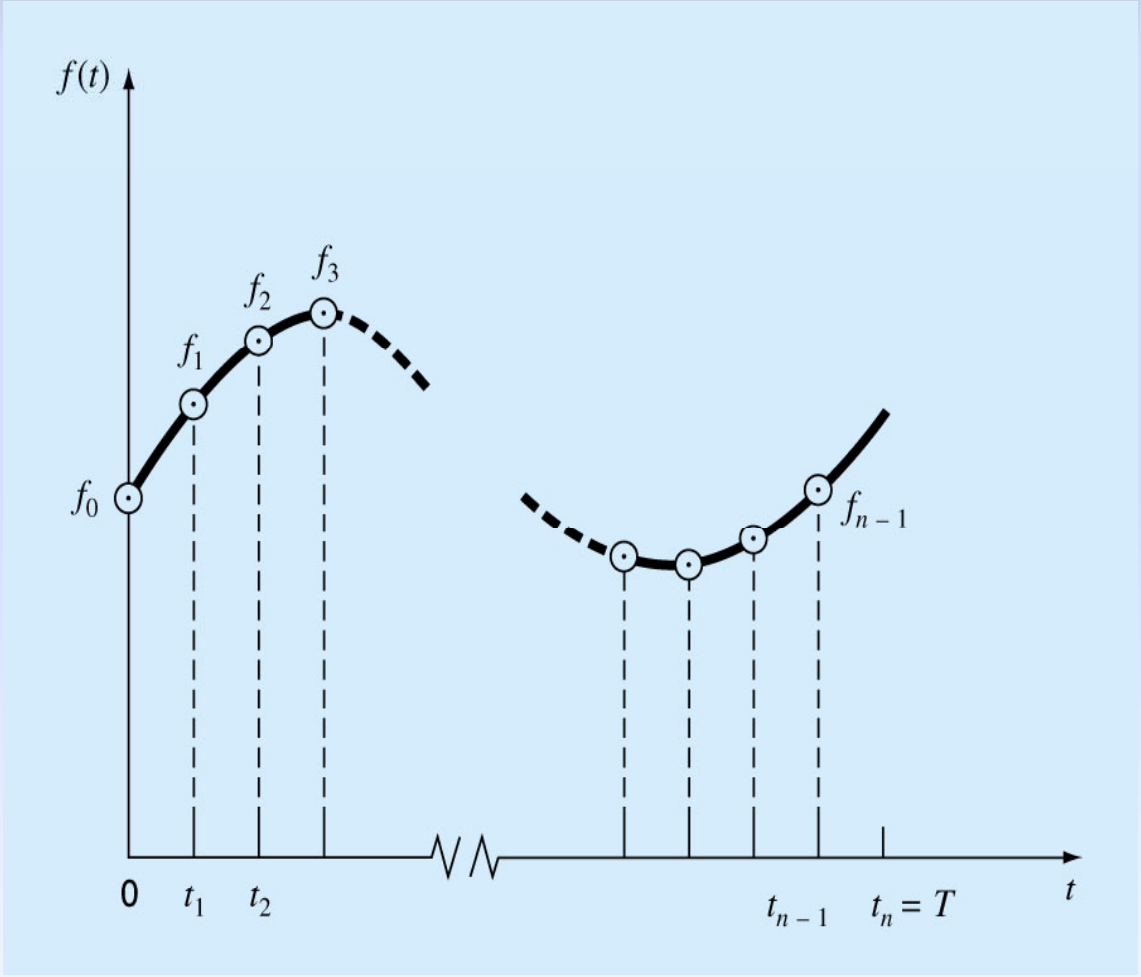


FIGURE 19.11
The sampling points of the discrete Fourier series.

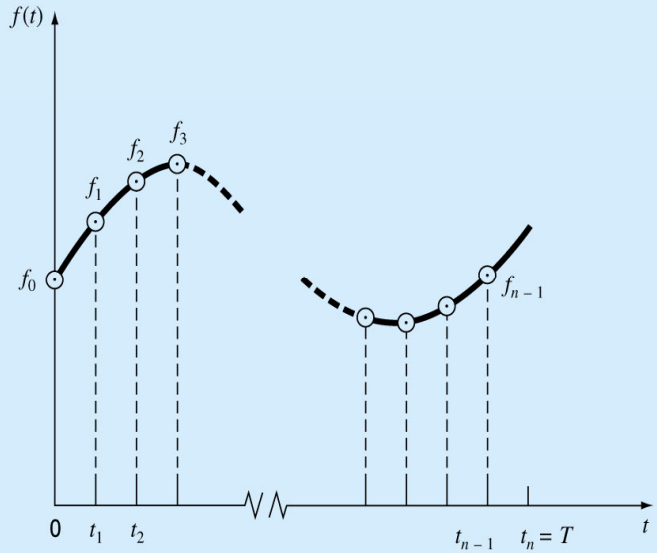
For such a sampled functions f_n, a discrete Fourier transform can be written as

$$F_k = \sum_{n=0}^{N-1} f_n e^{-ik\omega_0 n} \qquad \text{for } k = 0 \text{ to } N-1$$

and the inverse Fourier transform as

$$f_n = \frac{1}{N} \sum_{k=0}^{N-1} F_k e^{ik\omega_0 n} \qquad \text{for } n = 0 \text{ to } N-1$$

where $\omega_0 = 2\pi/N$.




```
DOFOR k = 0, N − 1
  DOFOR n = 0, N − 1
    angle = kω0n
    real_k = real_k + f_n cos(angle)/N
    imaginary_k = imaginary_k − f_n sin(angle)/N
  END DO
END DO
```

FIGURE 19.12
Pseudocode for computing the DFT.

Fast Fourier Transform (FFT)

- FFT is an algorithm that has been developed to compute the DFT in an extremely economical (fast) fashion by using the results of previous computations to reduce the number of operations.

- FFT exploits the periodicity and symmetry of trigonometric functions to compute the transform with approximately $N \log_2 N$ operations. Thus for N=50 samples, the FFT is 10 times faster than the standard DFT. For N=1000, it is about 100 times faster.
  - Sande-Tukey Algorithm
  - Cooley-Tukey Algorithm

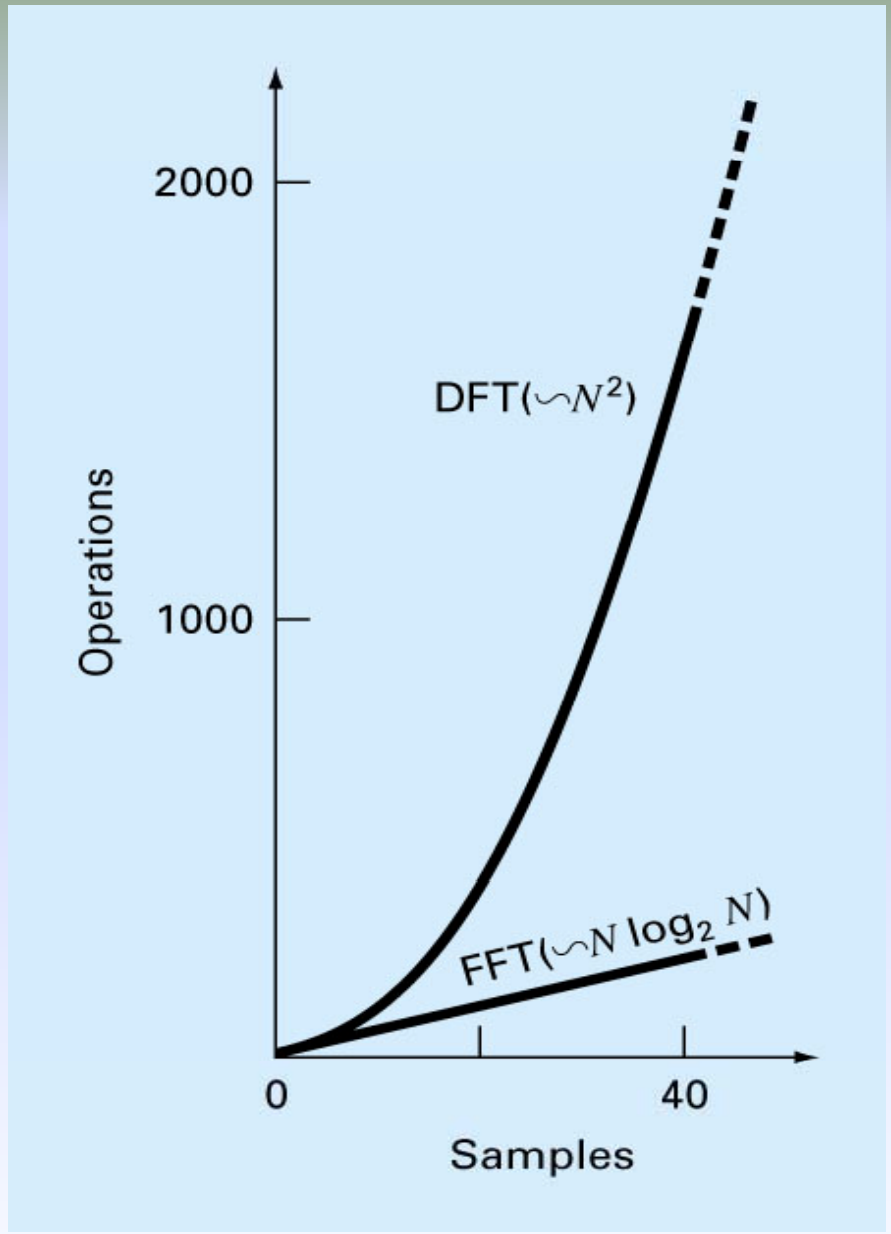


FIGURE 19.14
Plot of number of operations vs. sample size for the standard DFT and the FFT.

MATLAB Function: `fft`

fft - Fast Fourier transform

This MATLAB function returns the discrete Fourier transform (DFT) of vector x, computed with a fast Fourier transform (FFT) algorithm.

```
Y = fft(x)
Y = fft(X,n)
Y = fft(X,[],dim)
Y = fft(X,n,dim)
```

See also fft2, fftn, fftshift, fftw, filter, ifft

A common use of Fourier transforms is to find the frequency components of a signal buried in a noisy time domain signal. Consider data sampled at 1000 Hz. Form a signal containing a 50 Hz sinusoid of amplitude 0.7 and 120 Hz sinusoid of amplitude 1 and corrupt it with some zero-mean random noise:

```
Fs = 1000;                    % Sampling frequency
T = 1/Fs;                     % Sample time
L = 1000;                     % Length of signal
t = (0:L-1)*T;                % Time vector
% Sum of a 50 Hz sinusoid and a 120 Hz sinusoid
x = 0.7*sin(2*pi*50*t) + sin(2*pi*120*t);
y = x + 2*randn(size(t));     % Sinusoids plus noise
plot(Fs*t(1:50),y(1:50))
title('Signal Corrupted with Zero-Mean Random Noise')
xlabel('time (milliseconds)')
```

Homework: Try the following MATLAB code at home and try to understand what's going on at each step.

```
clear all
close all

% fft Example
n=8; dt=0.02; fs=1/dt; T = 0.16;
tspan=(0:n-1)/fs; % omits the last point
y=5+cos(2*pi*12.5*tspan)+sin(2*pi*31.25*tspan);
subplot(3,1,1);
plot(tspan,y,'-og','linewidth',2);
title('(a) f(t) versus time (s)'); grid;
Y=fft(y)/n;
nyquist=fs/2;fmin=1/T;
f = linspace(fmin,nyquist,n/2);
Y(1)=[];YP=Y(1:n/2);
subplot(3,1,2)
stem(f,real(YP),'linewidth',2,'MarkerFaceColor','blue')
grid;title('(b) Real component versus frequency')
subplot(3,1,3)
stem(f,imag(YP),'linewidth',2,'MarkerFaceColor','blue')
grid; title('(b) Imaginary component versus frequency')
xlabel('frequency (Hz)')
```