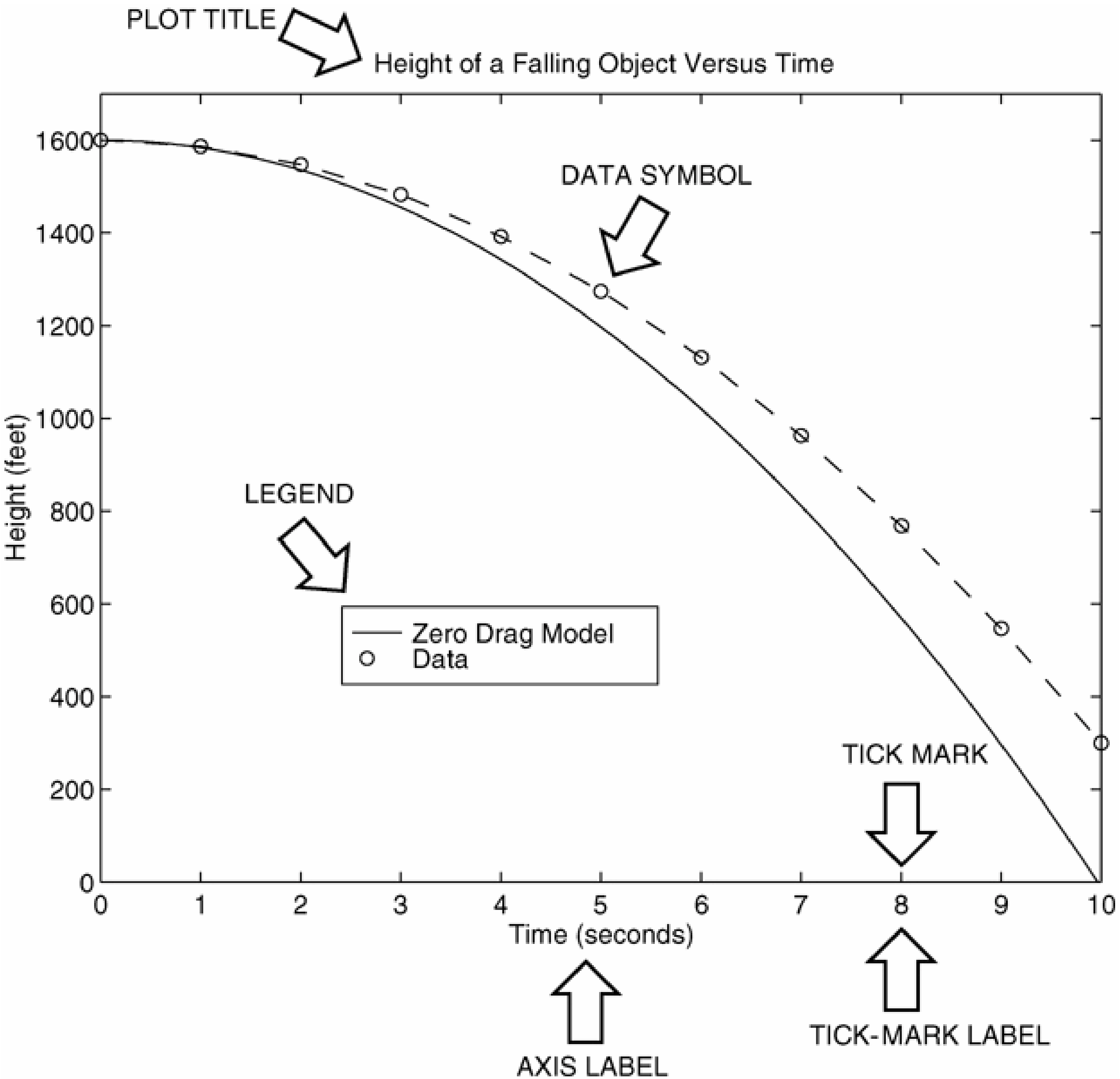
The McGraw-Hill Companies

PowerPoint to accompany

Introduction to MATLAB 7 for Engineers

Chapter 5
Advanced Plotting and Model Building

Nomenclature for a typical *xy* plot. Figure 5.1–1



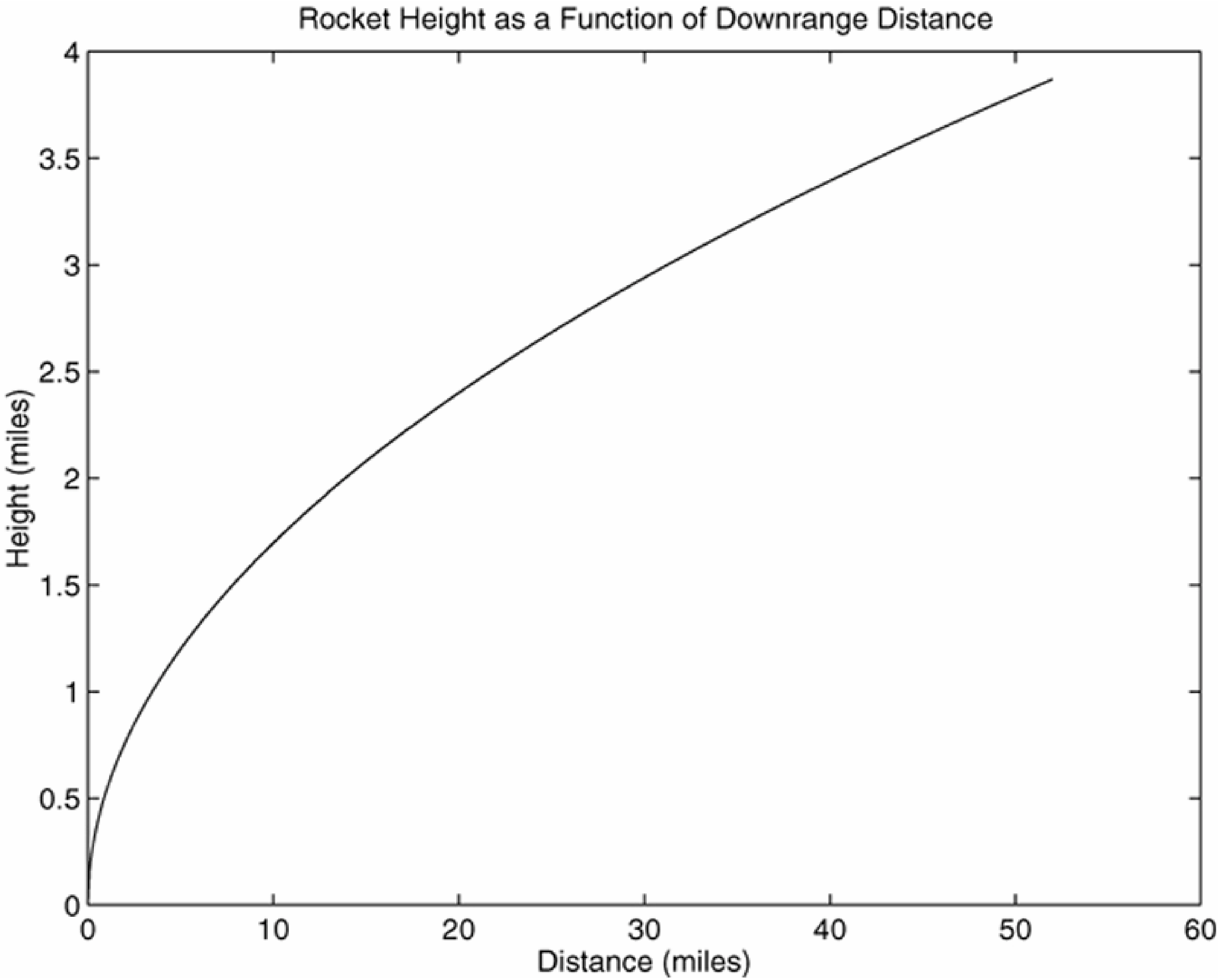
More? See pages 260-261.

The following MATLAB session plots $y = 0.4\sqrt{1.8x}$ for $0 \le x \le 52$, where y represents the height of a rocket after launch, in miles, and x is the horizontal (downrange) distance in miles.

```
>>x = [0:0.1:52];
>>y = 0.4*sqrt(1.8*x);
>>plot(x,y)
>>xlabel('Distance (miles)')
>>ylabel('Height (miles)')
>>title('Rocket Height as a Function of
  Downrange Distance')
```

The resulting plot is shown on the next slide.

The autoscaling feature in MATLAB selects tick-mark spacing. Figure 5.1–2



The plot will appear in the Figure window. You can obtain a hard copy of the plot in several ways:

1. Use the menu system. Select **Print** on the **File** menu in the Figure window. Answer **OK** when you are prompted to continue the printing process.

2. Type `print` at the command line. This command sends the current plot directly to the printer.

3. Save the plot to a file to be printed later or imported into another application such as a word processor. You need to know something about graphics file formats to use this file properly. See the subsection **Exporting Figures.**

When you have finished with the plot, close the figure window by selecting **Close** from the **File** menu in the figure window.

Note that using the **Alt-Tab** key combination in Windows-based systems will return you to the Command window without closing the figure window.

If you do not close the window, it will not reappear when a new `plot` command is executed. However, the figure will still be updated.

Requirements for a Correct Plot

The following list describes the essential features of any plot:

1. Each axis must be labeled with the name of the quantity being plotted *and its units!* If two or more quantities having different units are plotted (such as when plotting both speed and distance versus time), indicate the units in the axis label if there is room, or in the legend or labels for each curve.

2. Each axis should have regularly spaced tick marks at convenient intervals—not too sparse, but not too dense—with a spacing that is easy to interpret and interpolate. For example, use 0.1, 0.2, and so on, rather than 0.13, 0.26, and so on.

(continued …)

Requirements for a Correct Plot (continued)

3. If you are plotting more than one curve or data set, label each on its plot or use a legend to distinguish them.

4. If you are preparing multiple plots of a similar type or if the axes' labels cannot convey enough information, use a title.

5. If you are plotting measured data, plot each data point with a symbol such as a circle, square, or cross (use the same symbol for every point in the same data set). If there are many data points, plot them using the dot symbol.

(continued …)

Requirements for a Correct Plot (continued)

6. Sometimes data symbols are connected by lines to help the viewer visualize the data, especially if there are few data points. However, connecting the data points, especially with a solid line, might be interpreted to imply knowledge of what occurs between the data points. Thus you should be careful to prevent such misinterpretation.

7. If you are plotting points generated by evaluating a function (as opposed to measured data), do *not* use a symbol to plot the points. Instead, be sure to generate many points, and connect the points with solid lines.

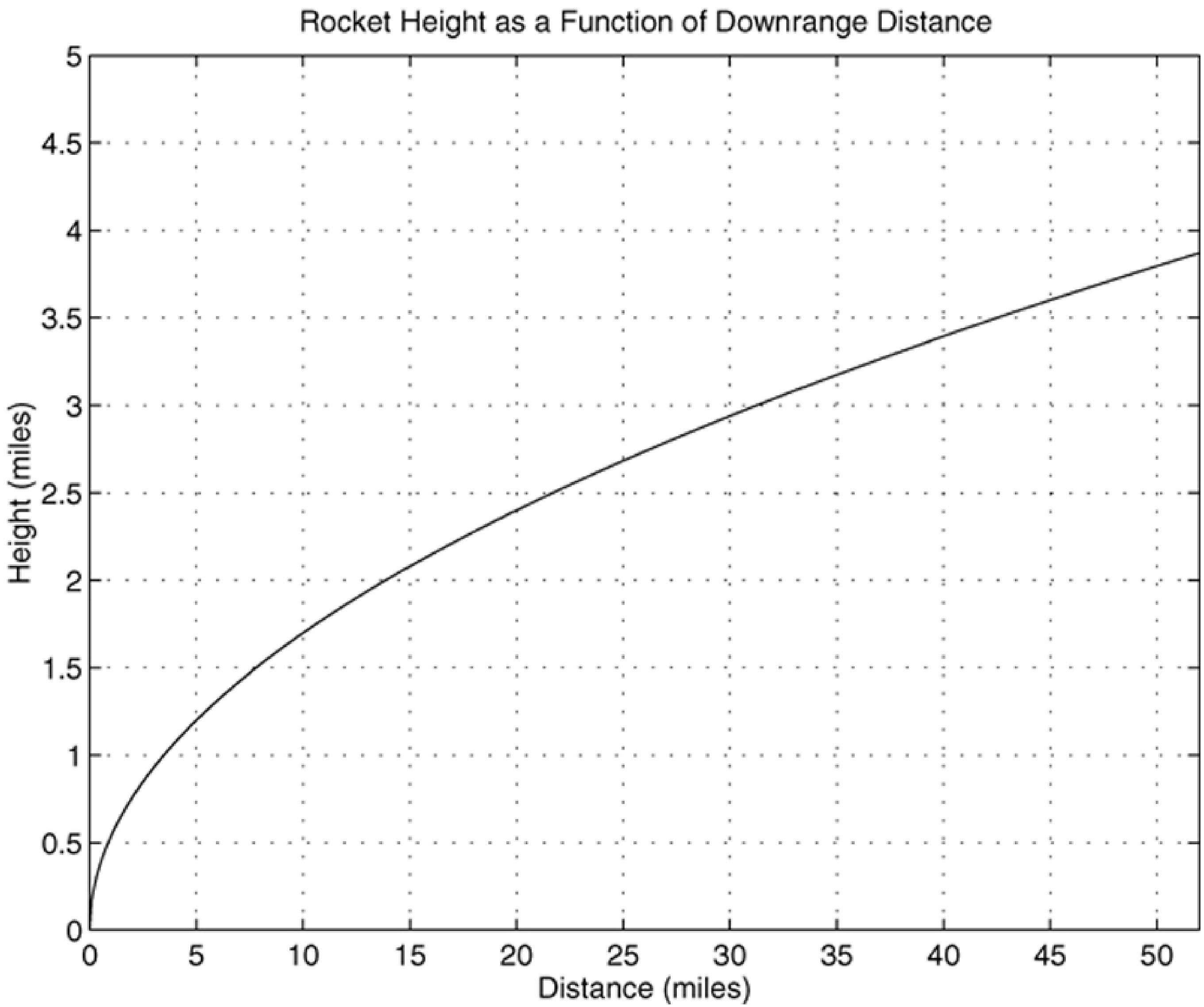
The `grid` and `axis` Commands

The `grid` command displays gridlines at the tick marks corresponding to the tick labels. Type `grid on` to add gridlines; type `grid off` to stop plotting gridlines. When used by itself, `grid` toggles this feature on or off, but you might want to use `grid on` and `grid off` to be sure.

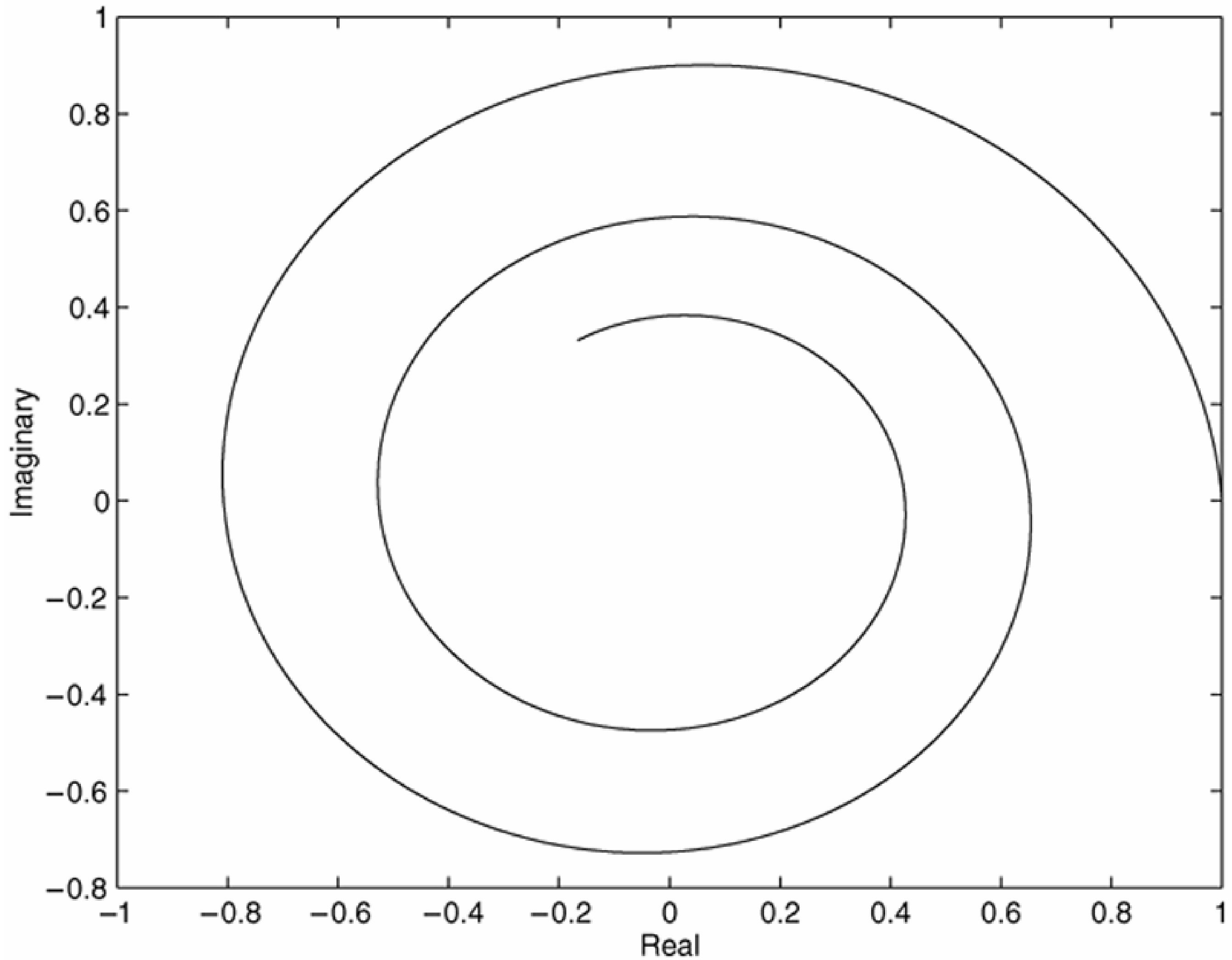
You can use the `axis` command to override the MATLAB selections for the axis limits. The basic syntax is `axis([xmin xmax ymin ymax])`. This command sets the scaling for the *x*- and *y*-axes to the minimum and maximum values indicated. Note that, unlike an array, this command does not use commas to separate the values.

More? See pages 264-265.

The effects of the `axis` and `grid` commands. Figure 5.1–3

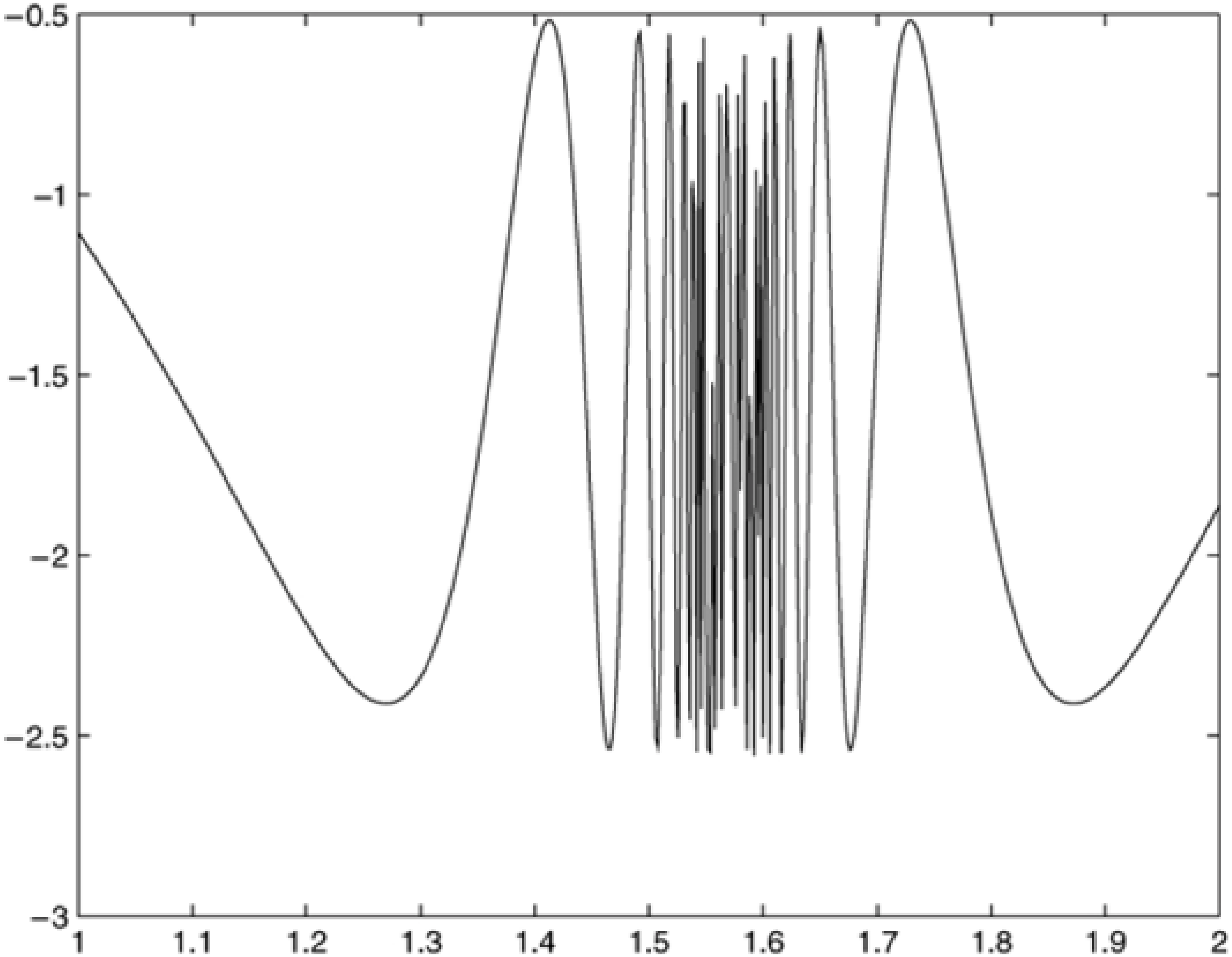


The `plot(y)` function plots the values in y versus the indices. Figure 5.1–4. See pages 265-266.

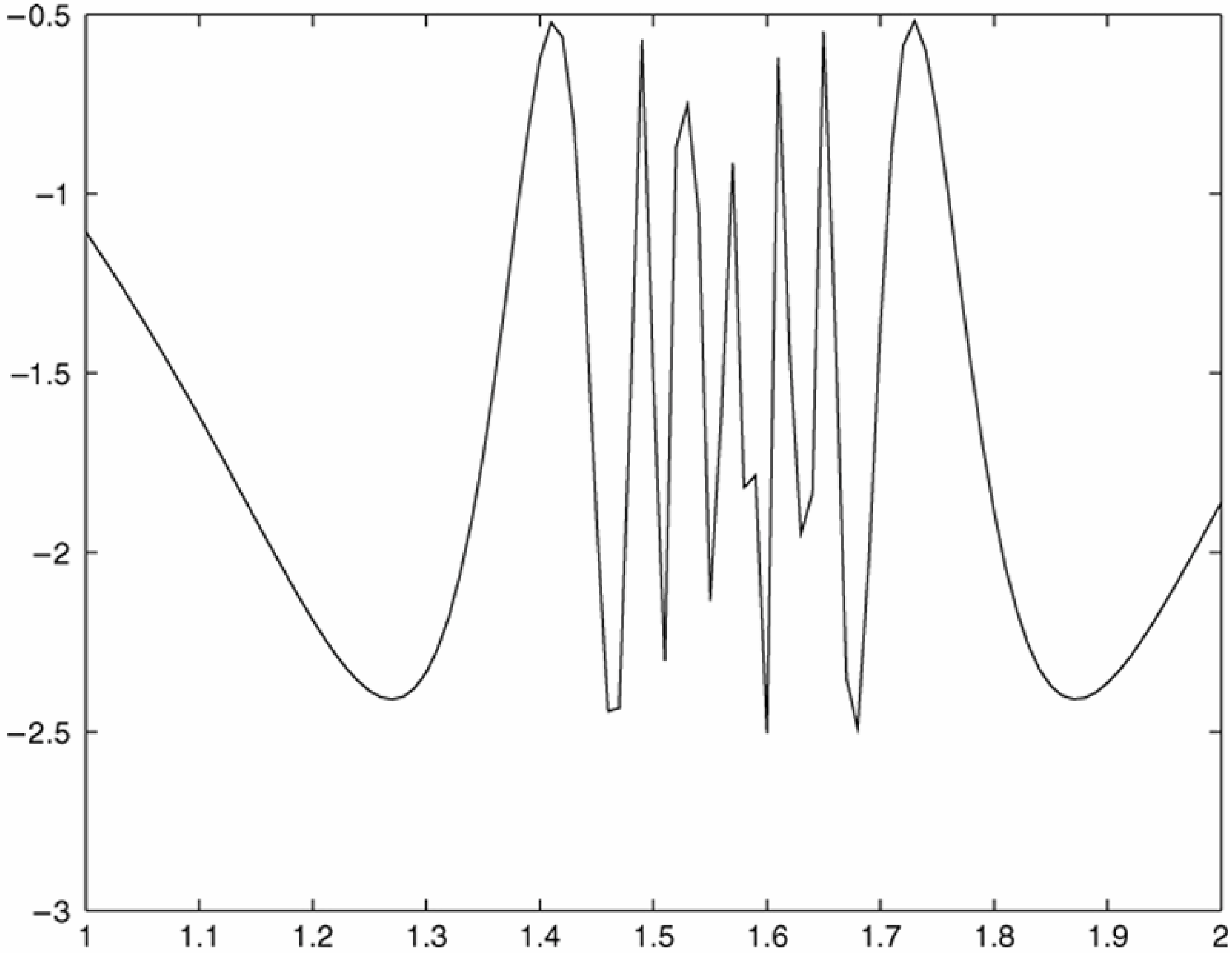


The `fplot` command plots a function specified as a string.

Figure 5.1–5 See pages 266-268.



The function in Figure 5.1–5 generated with the `plot` command, which gives more control than the `fplot` command. Figure 5.1–6 **See page 267.**



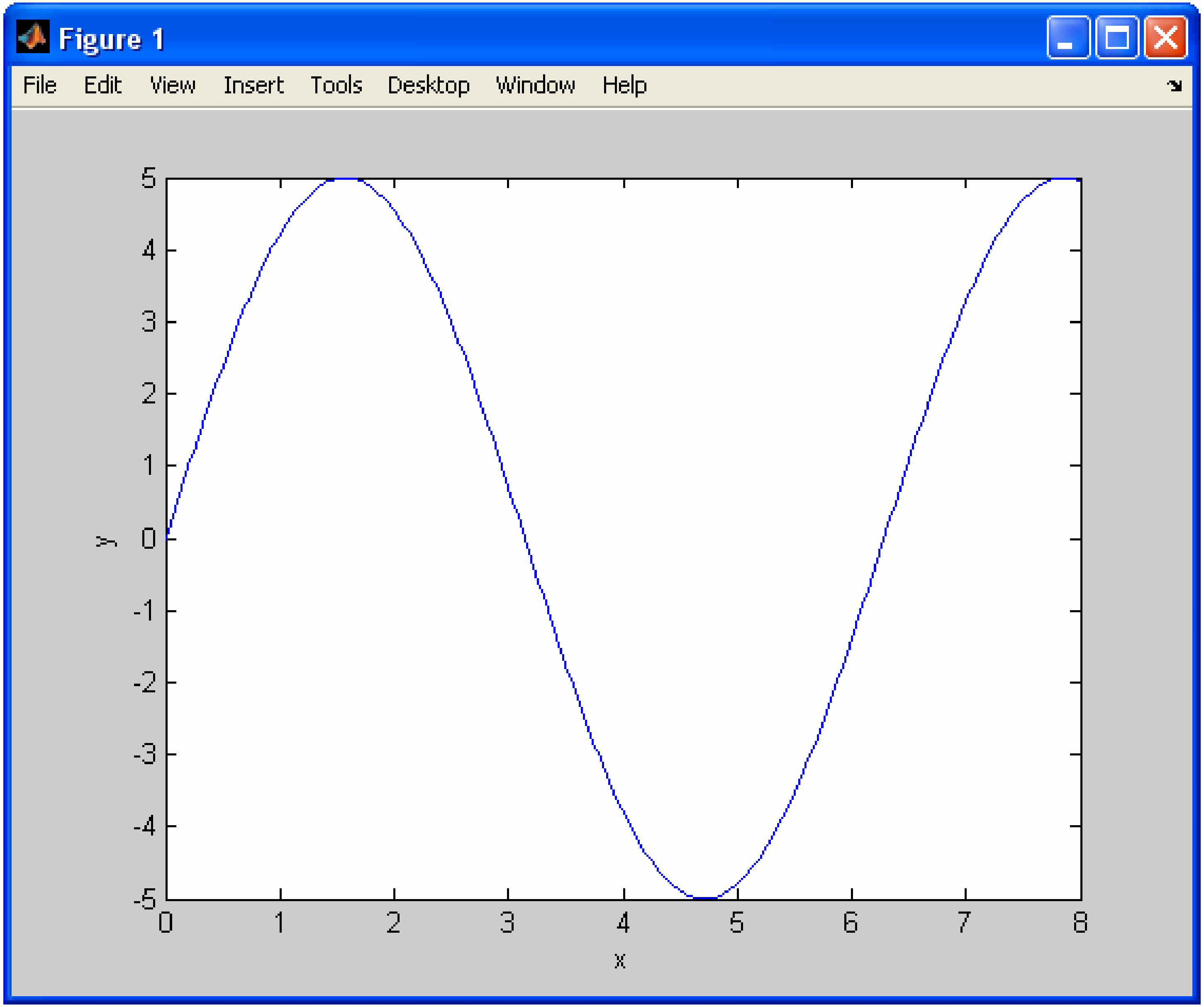
Plotting Polynomials with the `polyval` Function.

To plot the polynomial $3x^5 + 2x^4 – 100x^3 + 2x^2 – 7x + 90$ over the range $–6 \le x \le 6$ with a spacing of 0.01, you type

```
>>x = [-6:0.01:6];
>>p = [3,2,-100,2,-7,90];
>>plot(x,polyval(p,x)),xlabel('x'), ...
ylabel('p')
```

More? See page 268.

An example of a Figure window. Figure 5.1–7



Saving Figures

To save a figure that can be opened in subsequent MATLAB sessions, save it in a figure file with the .fig file name extension.

To do this, select **Save** from the Figure window **File** menu or click the **Save** button (the disk icon) on the toolbar.

If this is the first time you are saving the file, the **Save As** dialog box appears. Make sure that the type is MATLAB Figure (*.fig). Specify the name you want assigned to the figure file. Click OK.

Exporting Figures

To save the figure in a format that can be used by another application, such as the standard graphics file formats TIFF or EPS, perform these steps.

1. Select **Export Setup** from the **File** menu. This dialog lets you specify options for the output file, such as the figure size, fonts, line size and style, and output format.

2. Select **Export** from the **Export Setup** dialog. A standard **Save As** dialog appears.

3. Select the format from the list of formats in the **Save As** type menu. This selects the format of the exported file and adds the standard file name extension given to files of that type.

4. Enter the name you want to give the file, less the extension. Then click **Save.**

More? See pages 270-271.

On Windows systems, you can also copy a figure to the clipboard and then paste it into another application:

1. Select **Copy Options** from the **Edit** menu. The **Copying Options** page of the **Preferences** dialog box appears.
2. Complete the fields on the **Copying Options** page and click **OK.**
3. Select **Copy Figure** from the **Edit** menu.

Subplots

You can use the `subplot` command to obtain several smaller “subplots” in the same figure. The syntax is `subplot(m,n,p)`. This command divides the Figure window into an array of rectangular panes with *m* rows and *n* columns. The variable `p` tells MATLAB to place the output of the `plot` command following the `subplot` command into the *p*th pane.

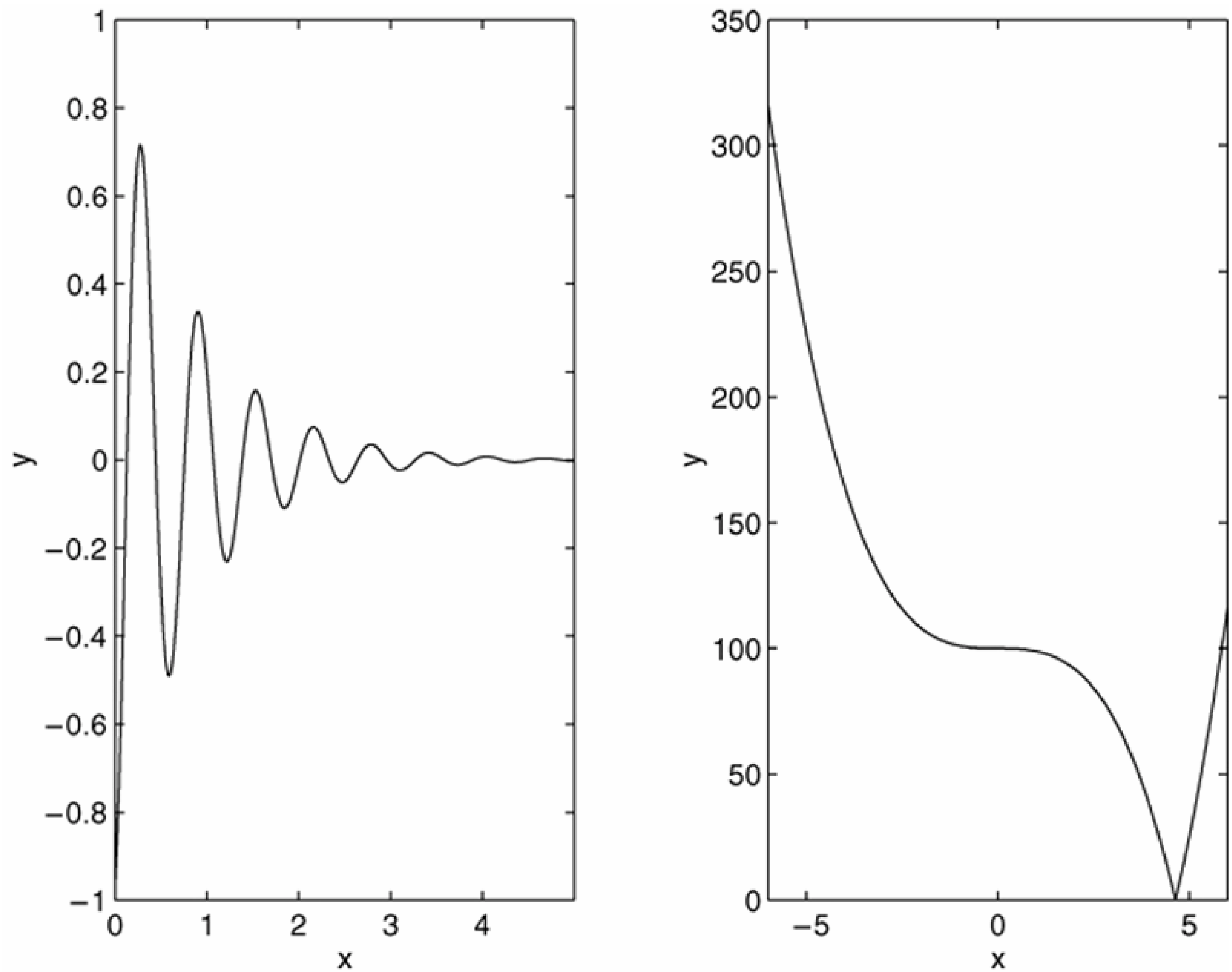
For example, `subplot(3,2,5)` creates an array of six panes, three panes deep and two panes across, and directs the next plot to appear in the fifth pane (in the bottom-left corner).

The following script file created Figure 5.2–1, which shows the plots of the functions $y = e^{-1.2x} \sin(10x + 5)$ for $0 \le x \le 5$ and $y = |x^3 - 100|$ for $-6 \le x \le 6$.

```
x = [0:0.01:5];
y = exp(-1.2*x).*sin(10*x+5);
subplot(1,2,1)
plot(x,y),axis([0 5 -1 1])
x = [-6:0.01:6];
y = abs(x.^3-100);
subplot(1,2,2)
plot(x,y),axis([-6 6 0 350])
```

The figure is shown on the next slide.

Application of the `subplot` command. Figure 5.2–1



More on subplots? See page 271.

Data Markers and Line Types

To plot `y` versus `x` with a solid line and `u` versus `v` with a dashed line, type `plot(x,y,u,v,'--')`, where the symbols `'--'` represent a dashed line.

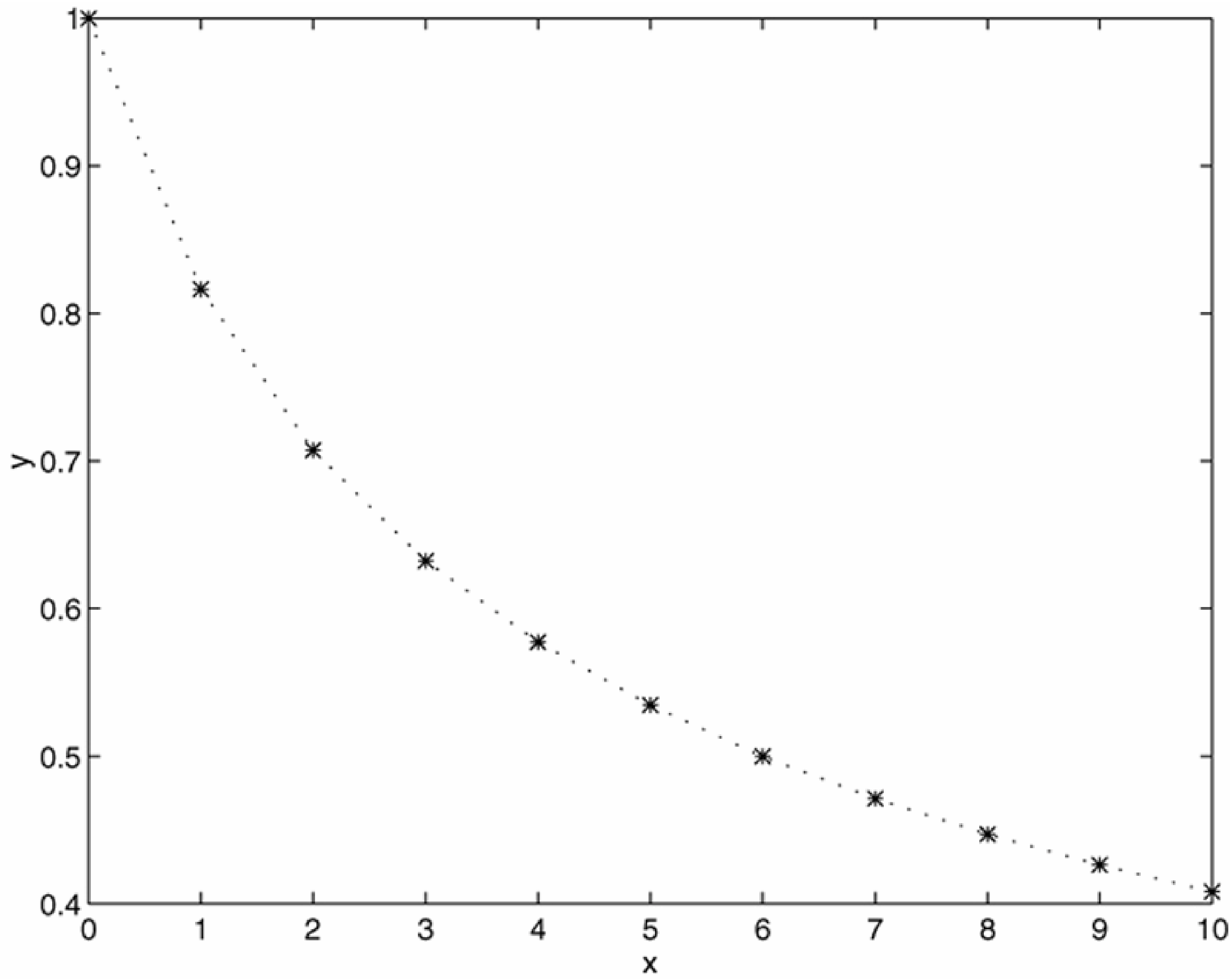
Table 5.2–1 gives the symbols for other line types.

To plot `y` versus `x` with asterisks (*) connected with a dotted line, you must plot the data twice by typing `plot(x,y,'*',x,y,':')`.

To plot `y` versus `x` with green asterisks (*) connected with a red dashed line, you must plot the data twice by typing `plot(x,y,'g*',x,y,'r--')`.

Data plotted using asterisks connected with a dotted line.

Figure 5.2–3



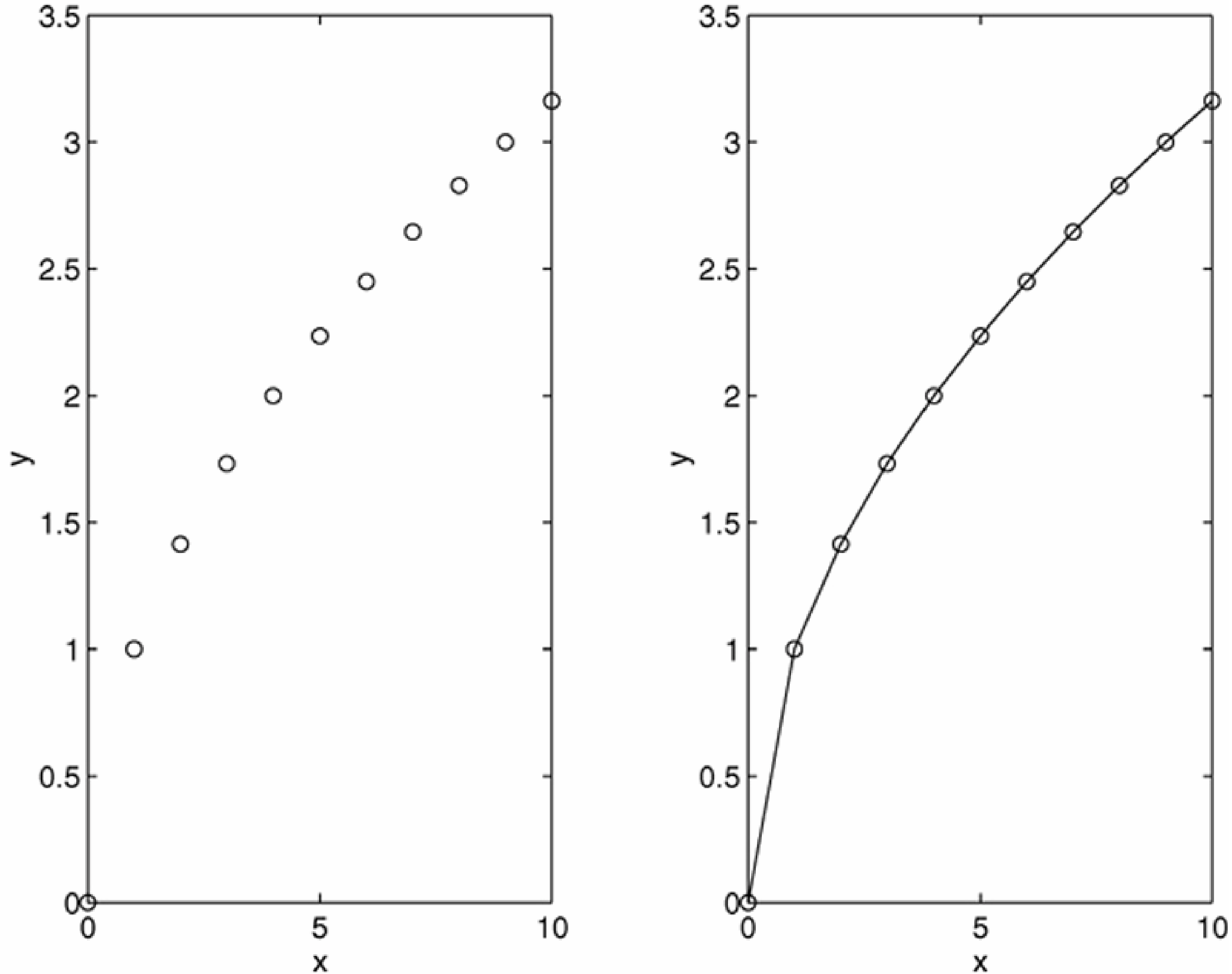
Specifiers for data markers, line types, and colors.

Table 5.2–1

Data markers†		Line types		Colors	
Dot (.)	.	Solid line	—	Black	k
Asterisk (*)	*	Dashed line	– –	Blue	b
Cross (×)	×	Dash-dotted line	– .	Cyan	c
Circle (○)	○	Dotted line	….	Green	g
Plus sign (+)	+			Magenta	m
Square (□)	s			Red	r
Diamond (◇)	d			White	w
Five-pointed star (w)	p			Yellow	y

†Other data markers are available. Search for “markers” in MATLAB help.

Use of data markers. Figure 5.2–2



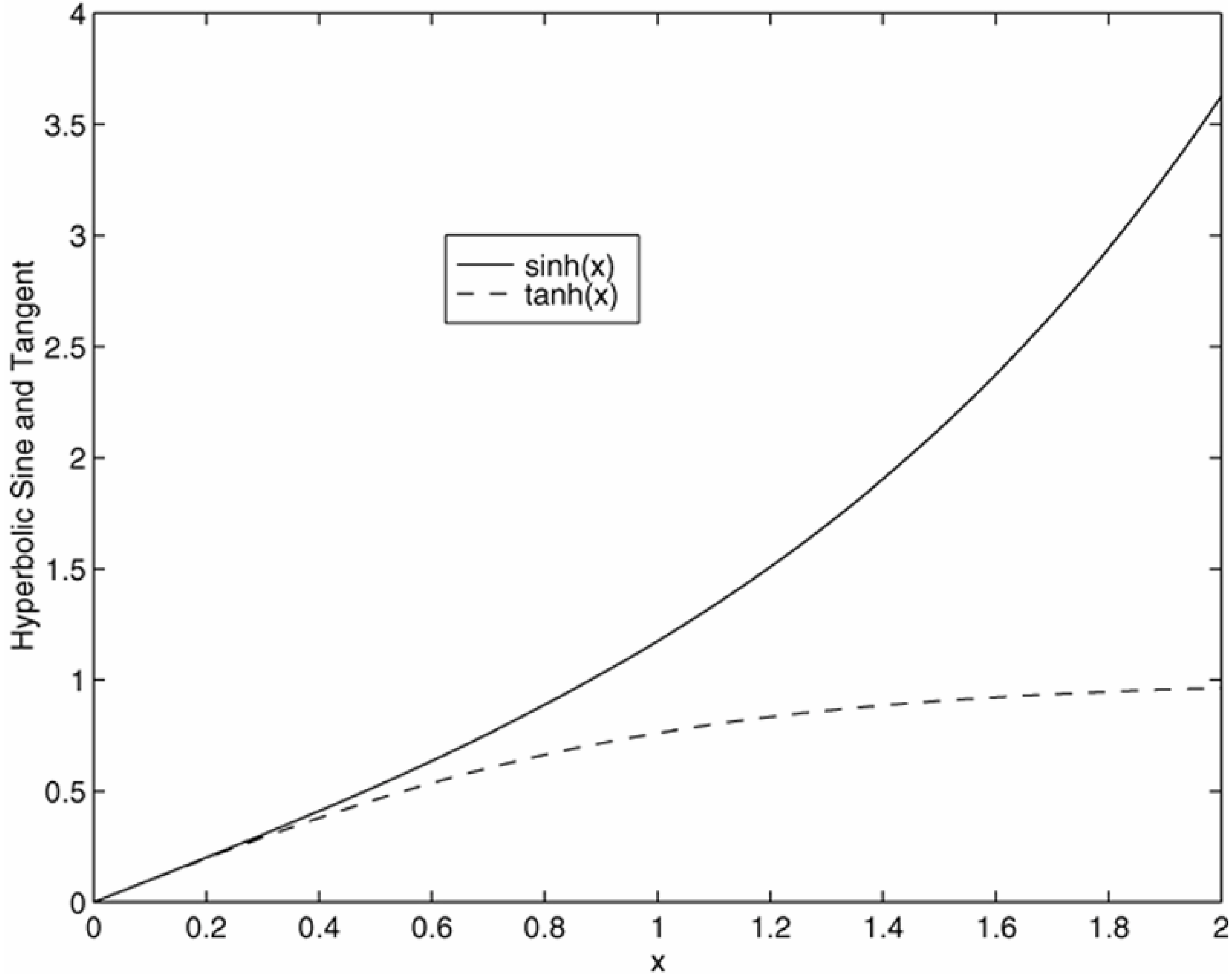
More?
See
pages
273-274.

Labeling Curves and Data

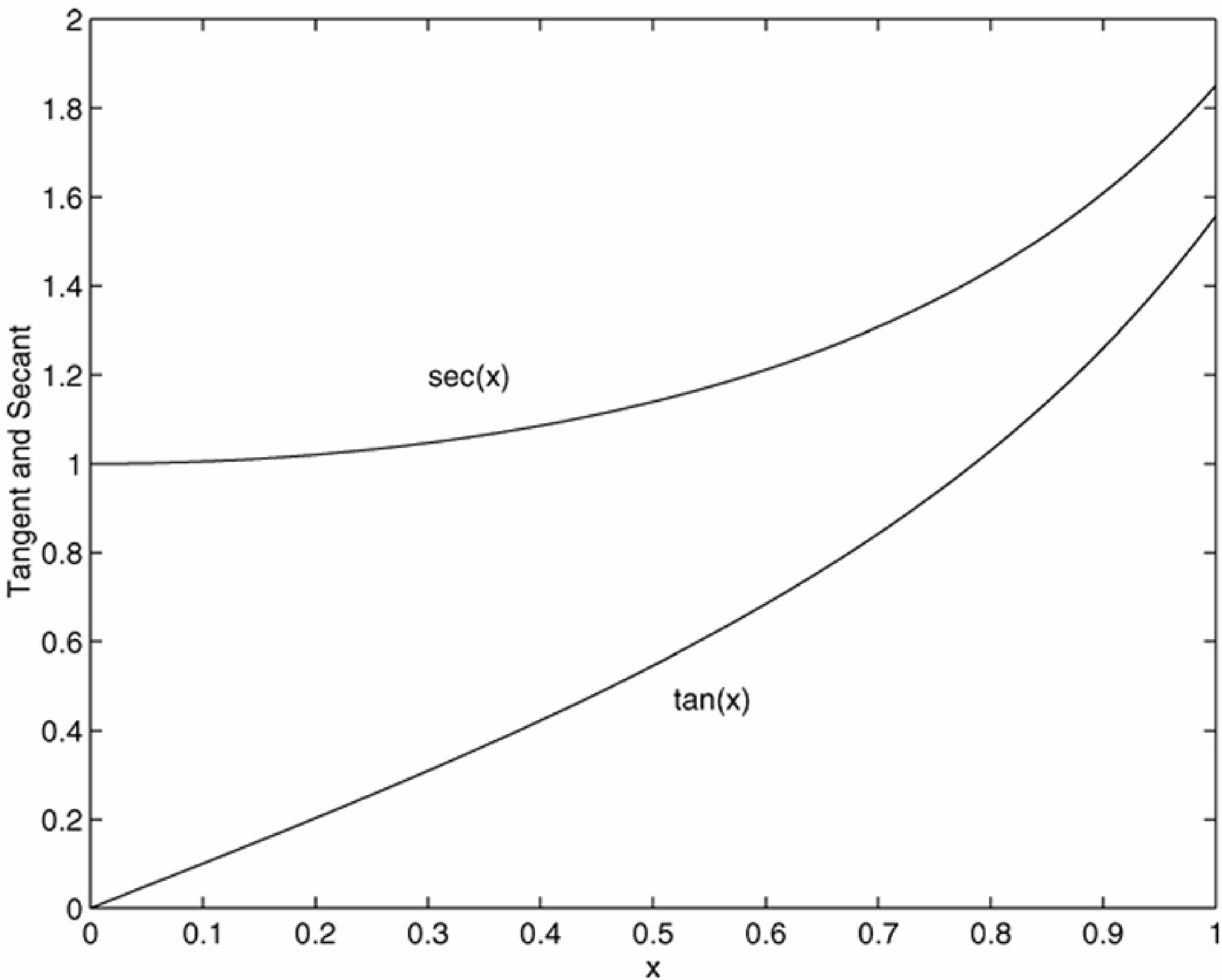
The `legend` command automatically obtains from the plot the line type used for each data set and displays a sample of this line type in the legend box next to the string you selected. The following script file produced the plot in Figure 5.2–4.

```
x = [0:0.01:2];
y = sinh(x);
z = tanh(x);
plot(x,y,x,z,'--'),xlabel('x'), ...
ylabel('Hyperbolic Sine and
Tangent'), ...
legend('sinh(x)','tanh(x)')
```

Application of the `legend` command. Figure 5.2–4

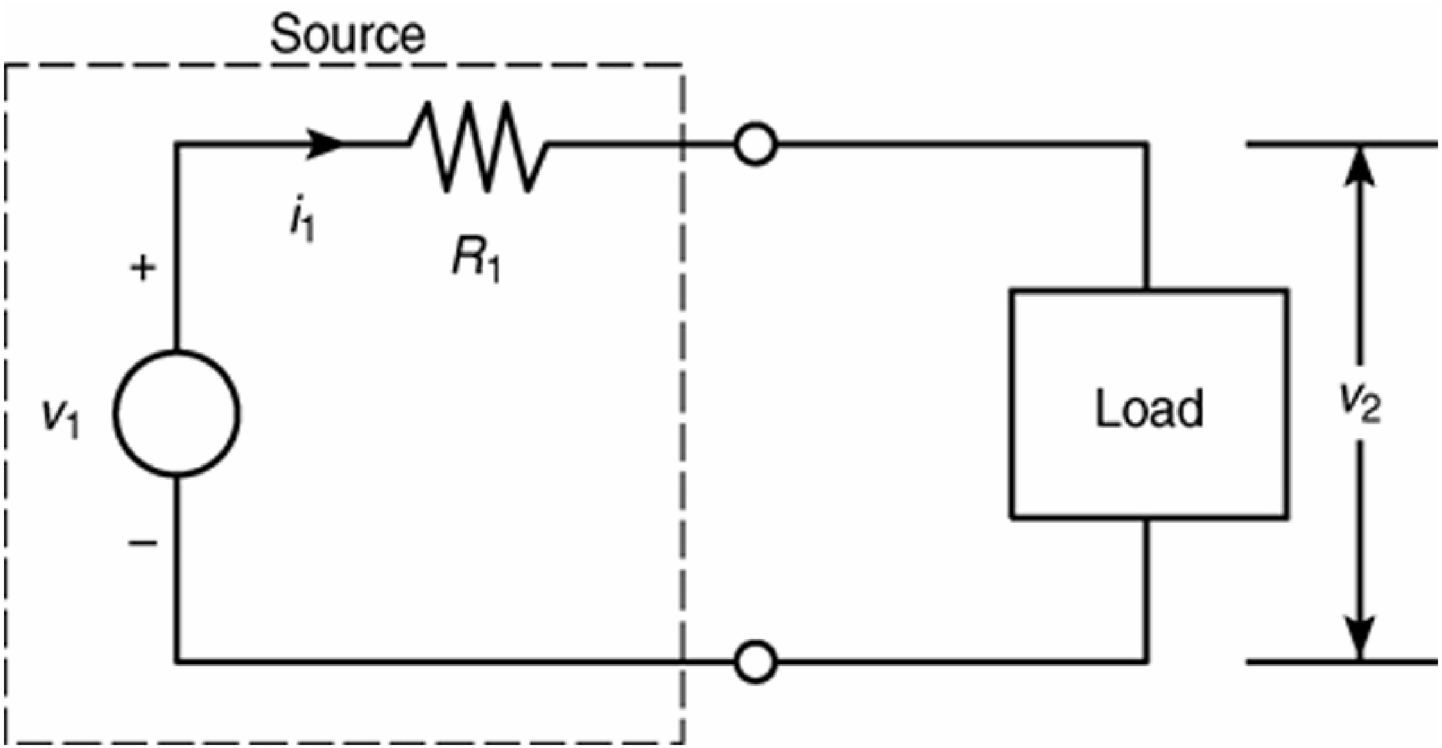


The `gtext` and `text` commands are also useful. Figure 5.2–5

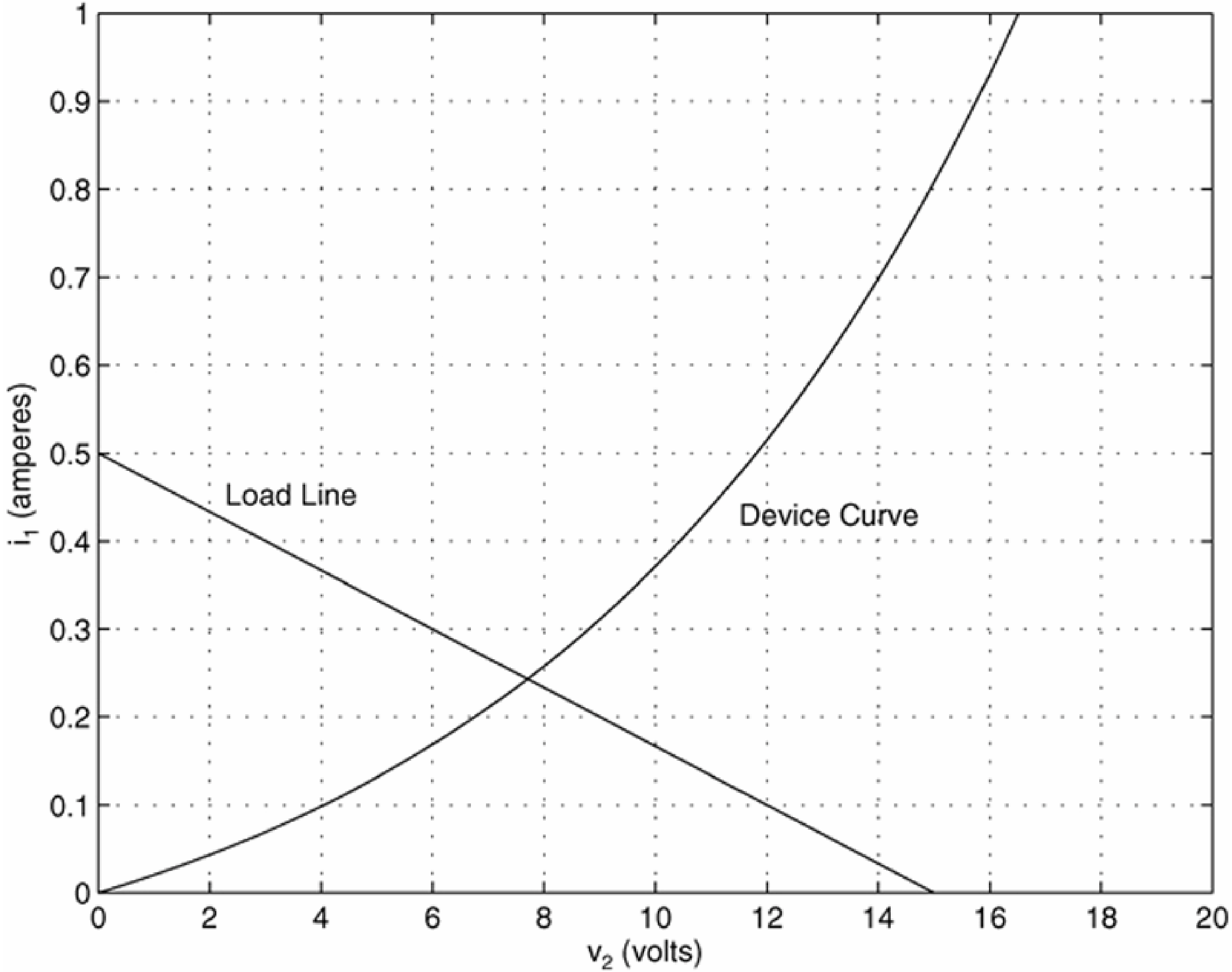


See page 276.

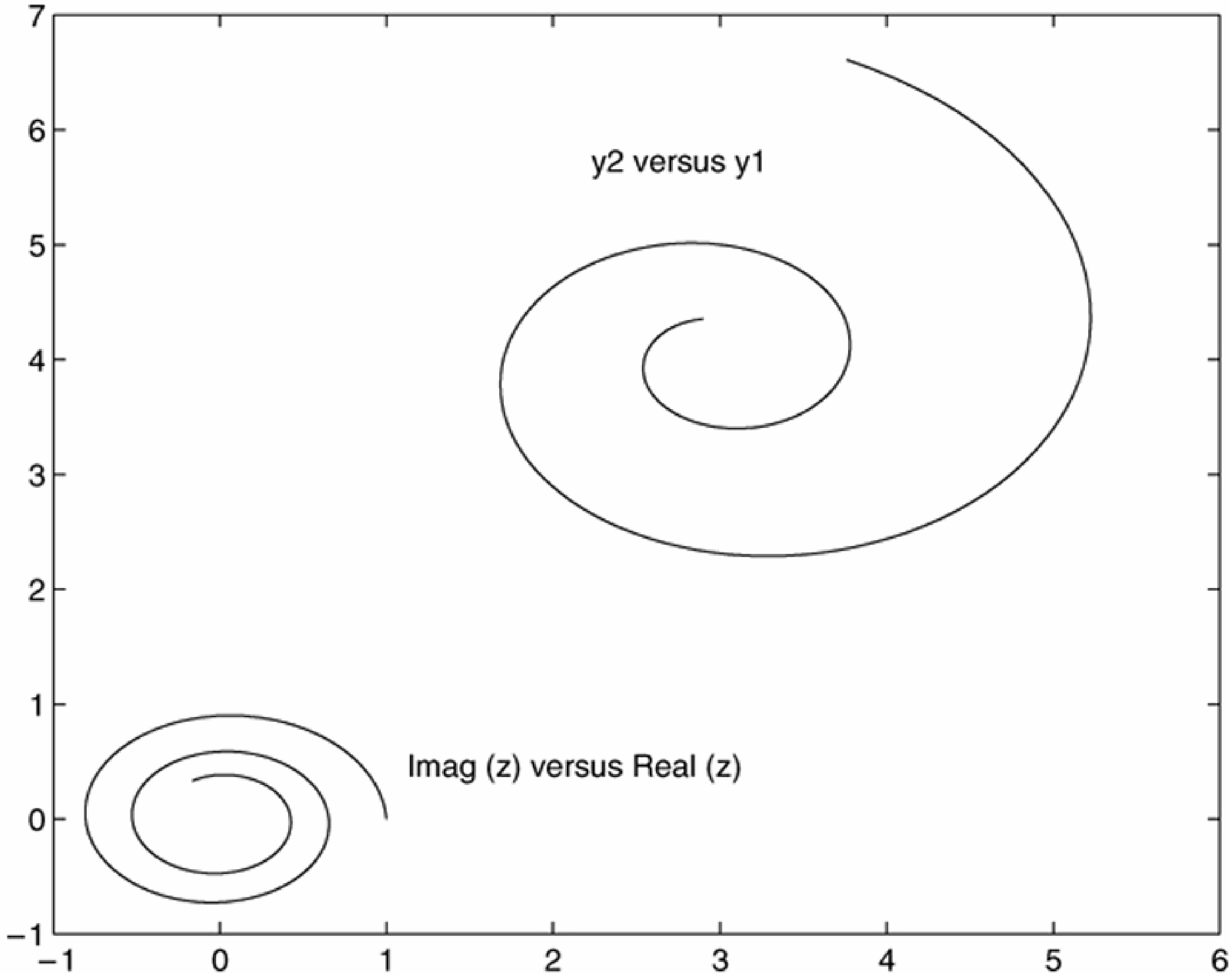
Graphical solution of equations: Circuit representation of a power supply and a load. Example 5.2-1. Figure 5.2–6



Plot of the load line and the device curve for Example 5.2–1.
Figure 5.2–7



Application of the `hold` command. Figure 5.2–8



See page 279.

Hints for Improving Plots

The following actions, while not required, can nevertheless improve the appearance of your plots:

1. Start scales from zero whenever possible. This technique prevents a false impression of the magnitudes of any variations shown on the plot.

2. Use sensible tick-mark spacing. If the quantities are months, choose a spacing of 12 because 1⁄10 of a year is not a convenient division. Space tick marks as close as is useful, but no closer. If the data is given monthly over a range of 24 months, 48 tick marks might be too dense, and also unnecessary.

(continued …)

Hints for Improving Plots (continued)

3. Minimize the number of zeros in the data being plotted. For example, use a scale in millions of dollars when appropriate, instead of a scale in dollars with six zeros after every number.

4. Determine the minimum and maximum data values for each axis before plotting the data. Then set the axis limits to cover the entire data range plus an additional amount to allow convenient tick-mark spacing to be selected.

For example, if the data on the *x*-axis ranges from 1.2 to 9.6, a good choice for axis limits is 0 to 10. This choice allows you to use a tick spacing of 1 or 2.

(continued …)

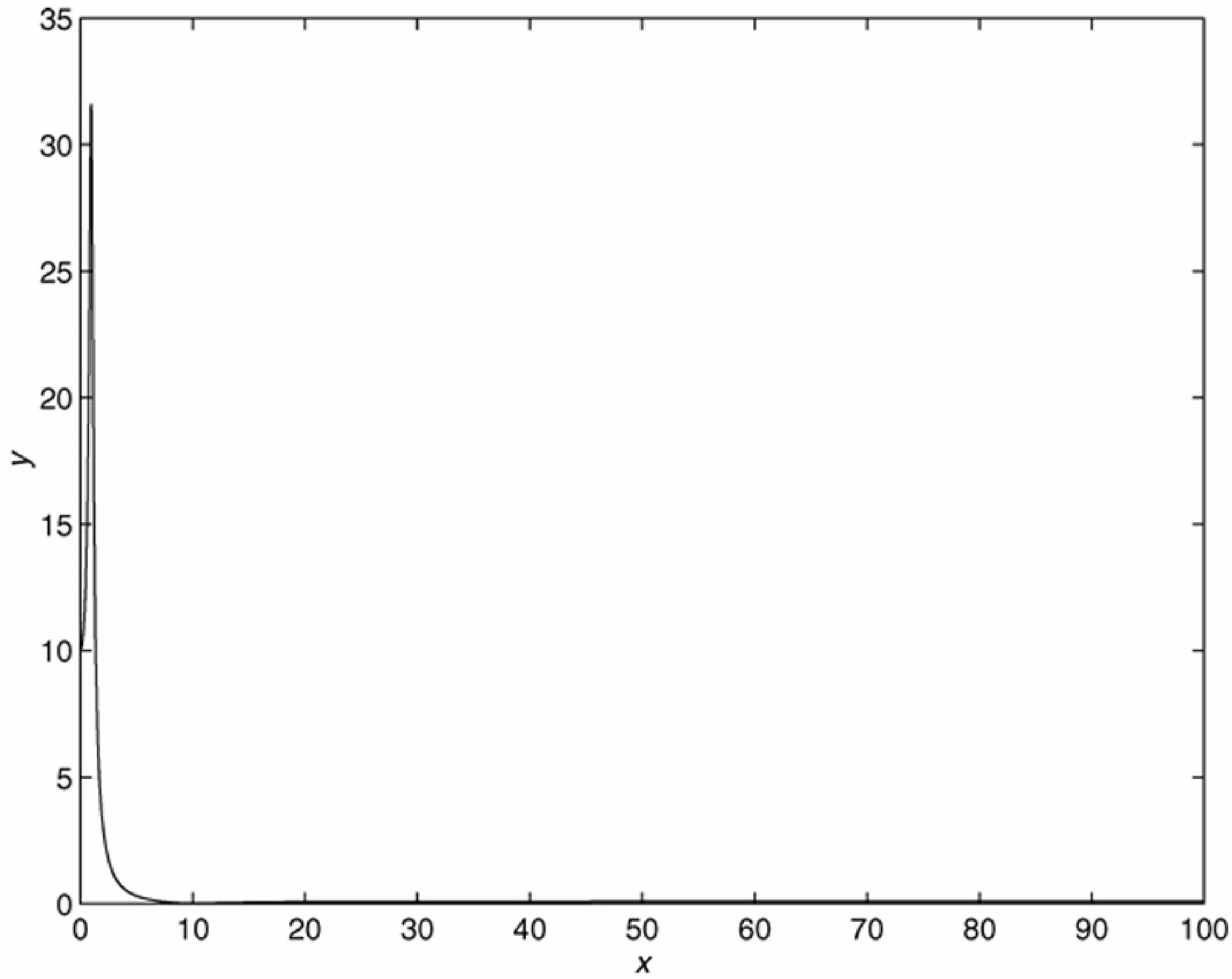
Hints for Improving Plots (continued)

5. Use a different line type for each curve when several are plotted on a single plot and they cross each other; for example, use a solid line, a dashed line, and combinations of lines and symbols. Beware of using colors to distinguish plots if you are going to make black-and-white printouts and photocopies.

6. Do not put many curves on one plot, particularly if they will be close to each other or cross one another at several points.

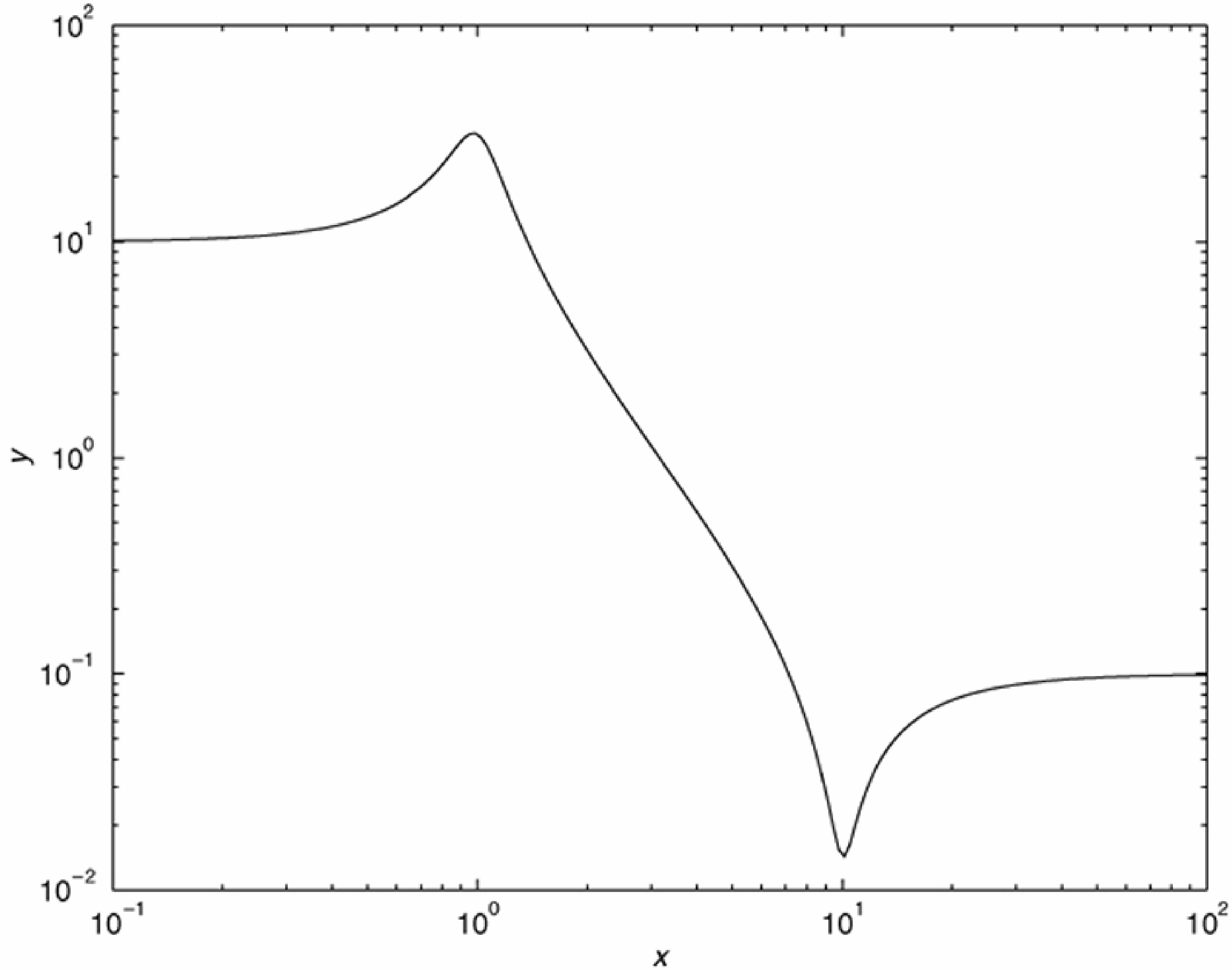
7. Use the same scale limits and tick spacing on each plot if you need to compare information on more than one plot.

Why use log scales? Rectilinear scales cannot properly display variations over wide ranges. **F**igure 5.3–1



A log-log plot can display wide variations in data values.

Figure 5.3–2



See page 282.

Logarithmic Plots

It is important to remember the following points when using log scales:

1. You cannot plot negative numbers on a log scale, because the logarithm of a negative number is not defined as a real number.

2. You cannot plot the number 0 on a log scale, because $\log_{10} 0 = \ln 0 = -\infty$. You must choose an appropriately small number as the lower limit on the plot.

(continued…)

Logarithmic Plots (continued)

3. The tick-mark labels on a log scale are the actual values being plotted; they are not the logarithms of the numbers. For example, the range of *x* values in the plot in Figure 5.3–2 is from $10^{-1} = 0.1$ to $10^2 = 100$.

4. Gridlines and tick marks within a decade are unevenly spaced. If 8 gridlines or tick marks occur within the decade, they correspond to values equal to 2, 3, 4, . . . , 8, 9 times the value represented by the first gridline or tick mark of the decade.

(continued…)

Logarithmic Plots (continued)

5. Equal distances on a log scale correspond to multiplication by the same constant (as opposed to addition of the same constant on a rectilinear scale).

For example, all numbers that differ by a factor of 10 are separated by the same distance on a log scale. That is, the distance between 0.3 and 3 is the same as the distance between 30 and 300. This separation is referred to as a *decade* or *cycle.*

The plot shown in Figure 5.3–2 covers three decades in x (from 0.1 to 100) and four decades in y and is thus called a *four-by-three-cycle plot.*

MATLAB has three commands for generating plots having log scales. The appropriate command depends on which axis must have a log scale.

1. Use the `loglog(x,y)` command to have both scales logarithmic.

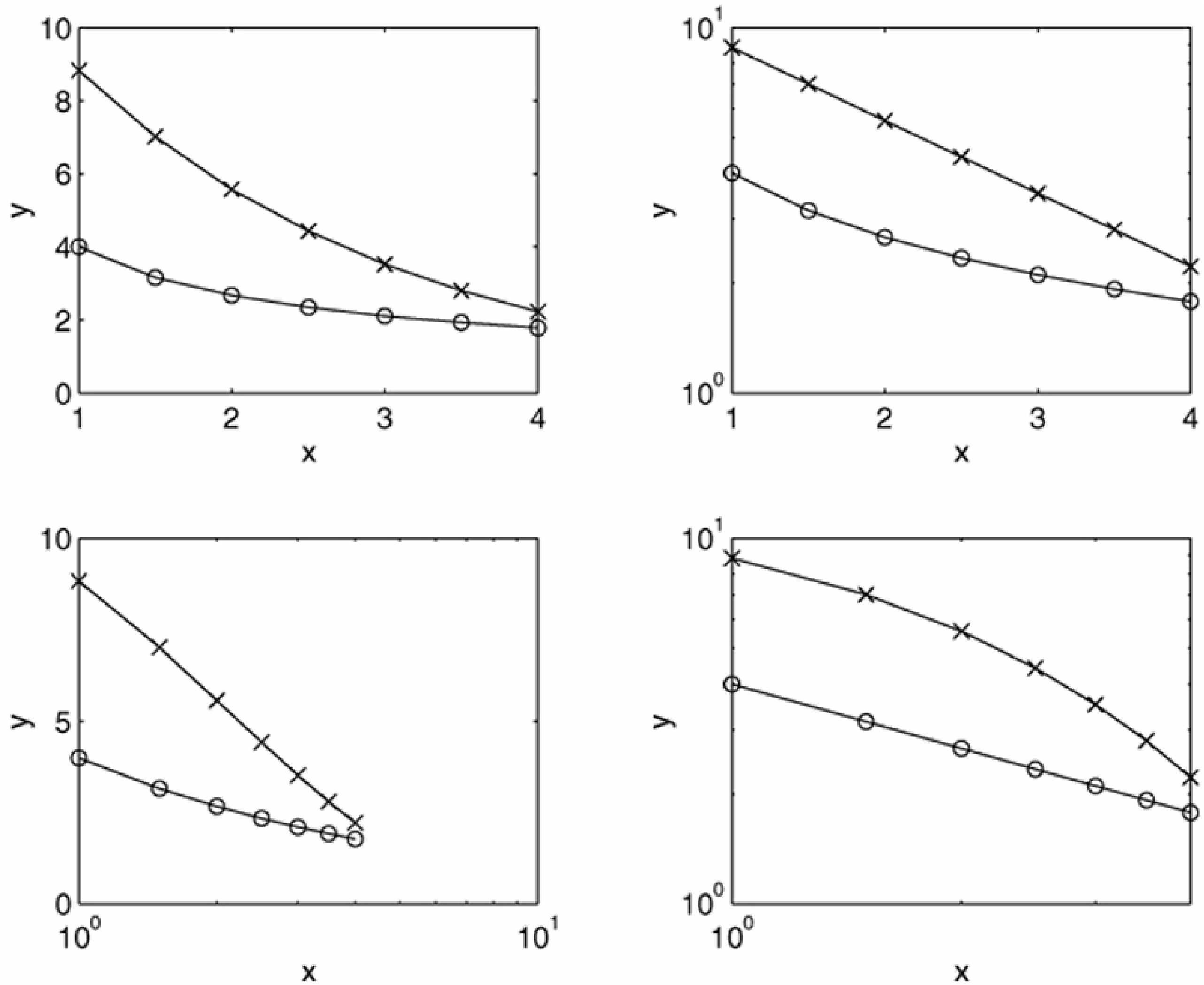
2. Use the `semilogx(x,y)` command to have the *x* scale logarithmic and the *y* scale rectilinear.

3. Use the `semilogy(x,y)` command to have the *y* scale logarithmic and the *x* scale rectilinear.

Specialized plot commands. Table 5.3–1

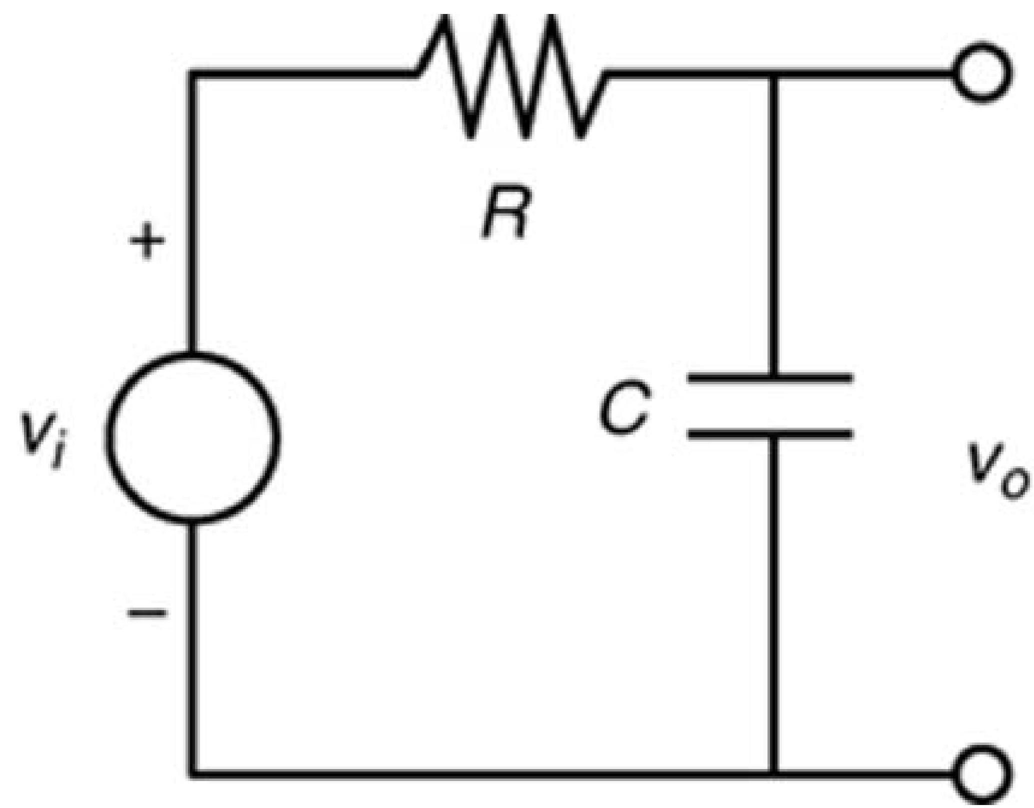
Command	Description
`bar(x,y)`	Creates a bar chart of `y` versus `x`.
`plotyy(x1,y1,x2,y2)`	Produces a plot with two *y*-axes, `y1` on the left and `y2` on the right.
`polar(theta,r,'type')`	Produces a polar plot from the polar coordinates `theta` and `r`, using the line type, data marker, and colors specified in the string `type`.
`stairs(x,y)`	Produces a stairs plot of `y` versus `x`.
`stem(x,y)`	Produces a stem plot of `y` versus `x`.

Two data sets plotted on four types of plots. Figure 5.3–3



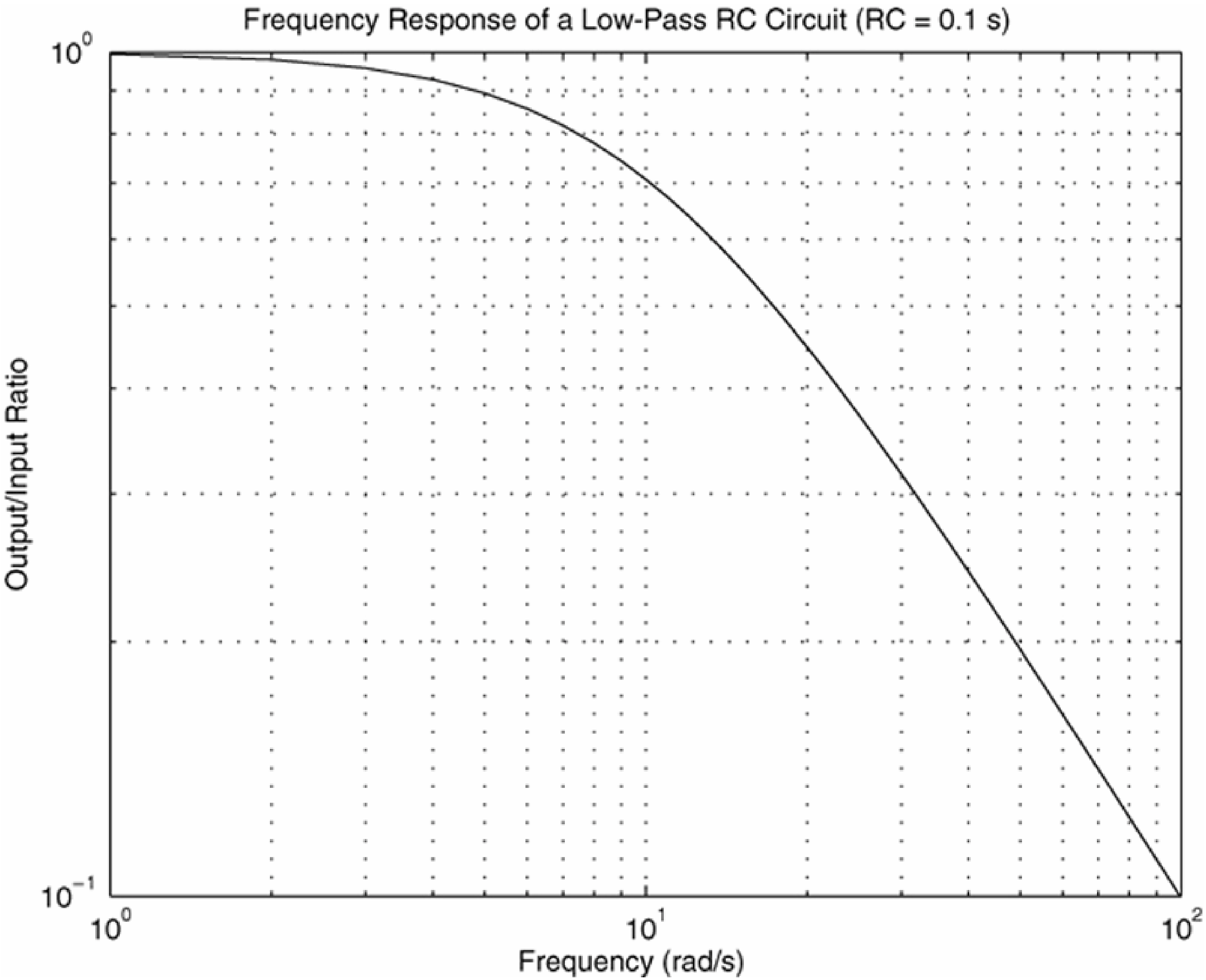
See page 285.

Application of logarithmic plots: An RC circuit. Figure 5.3–4



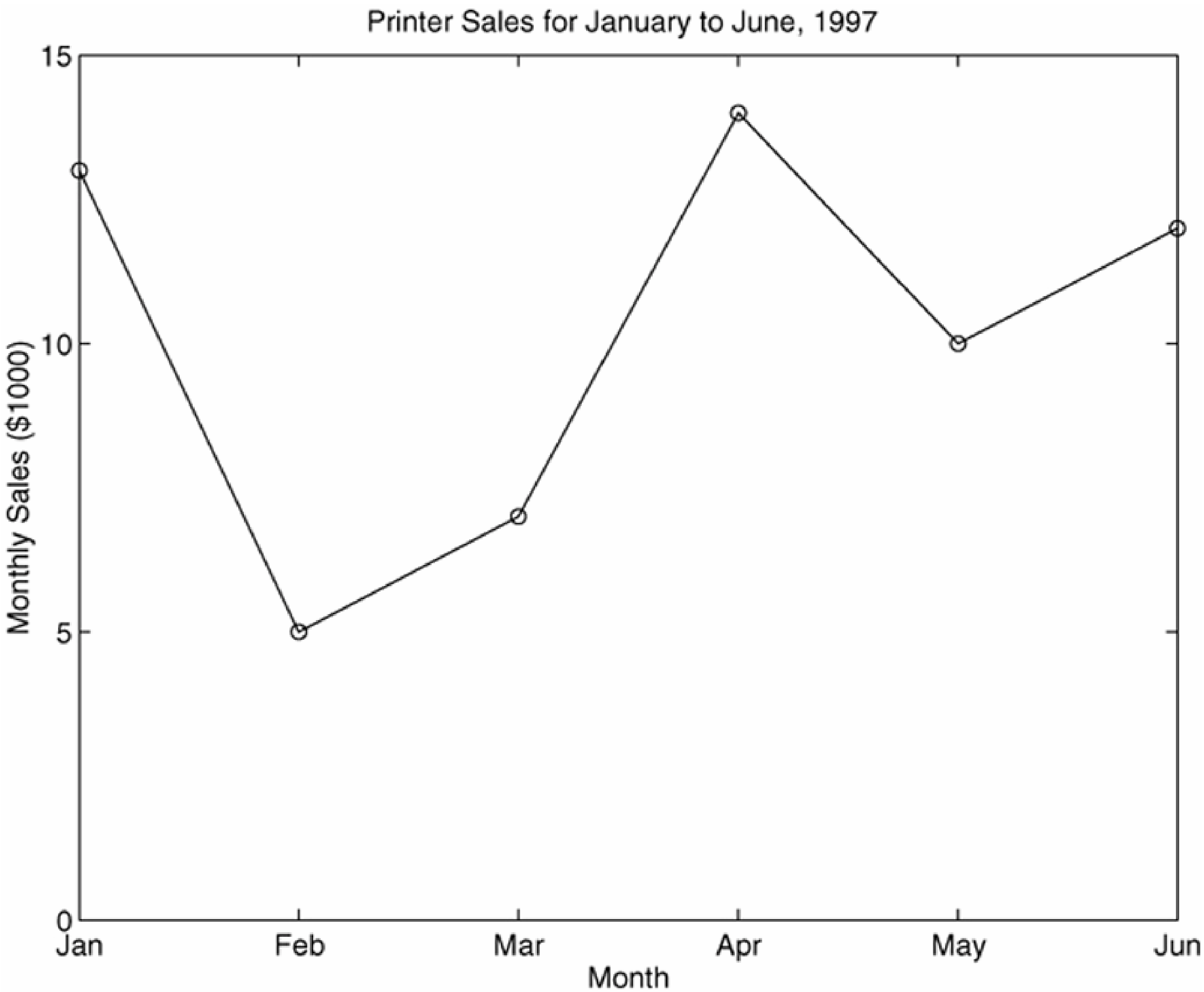
Frequency-response plot of a low-pass RC circuit.

Figure 5.3–5



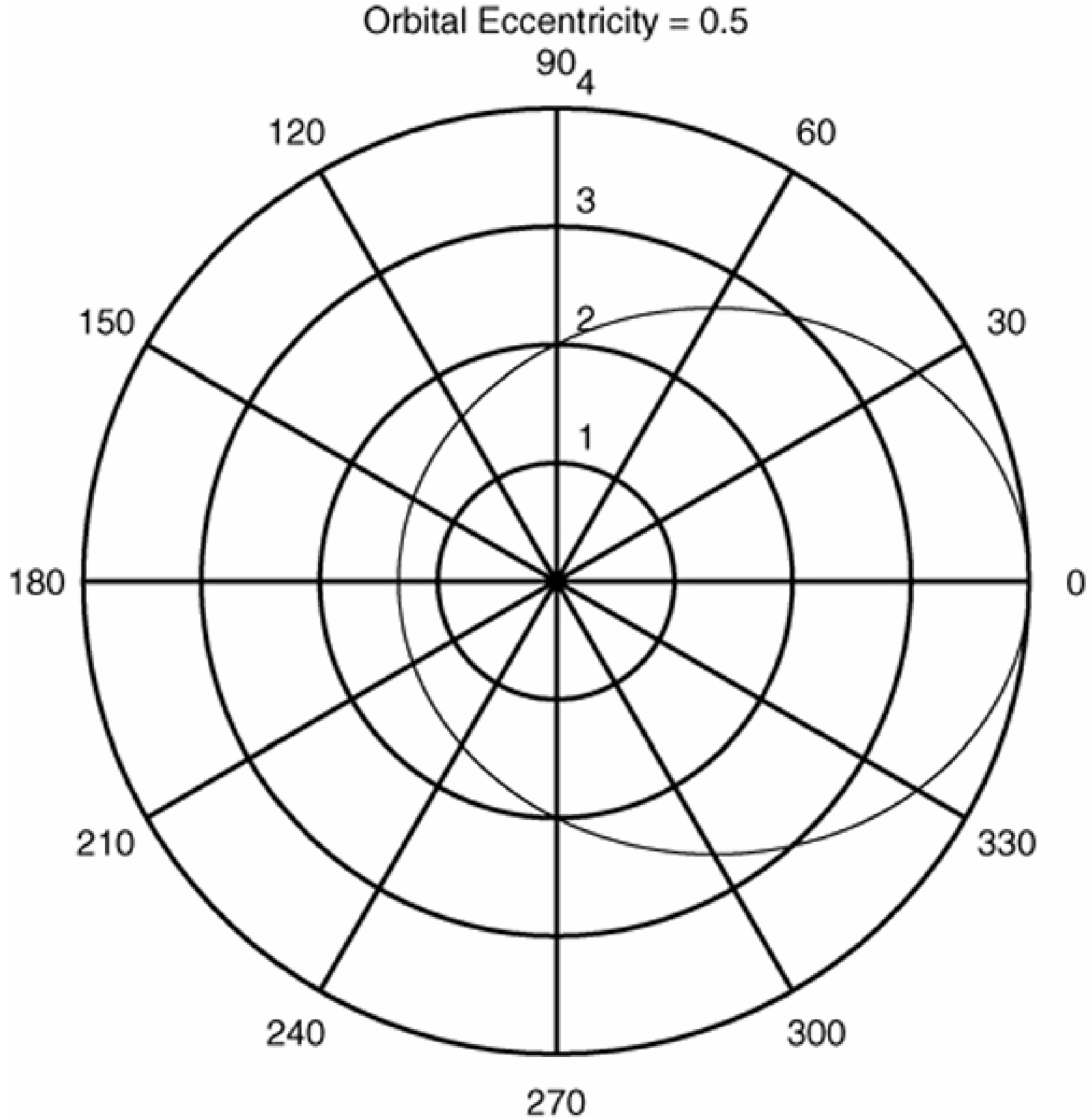
See pages 286-287.

An example of controlling the tick-mark labels with the `set` command. Figure 5.3–6



See pages 287-289.

A polar plot showing an orbit having an eccentricity of 0.5.
Figure 5.3–7



See pages 290-291.

Interactive Plotting in MATLAB

This interface can be advantageous in situations where:

- You need to create a large number of different types of plots,
- You must construct plots involving many data sets,
- You want to add annotations such as rectangles and ellipses, or
- You want to change plot characteristics such as tick spacing, fonts, bolding, italics, and colors.

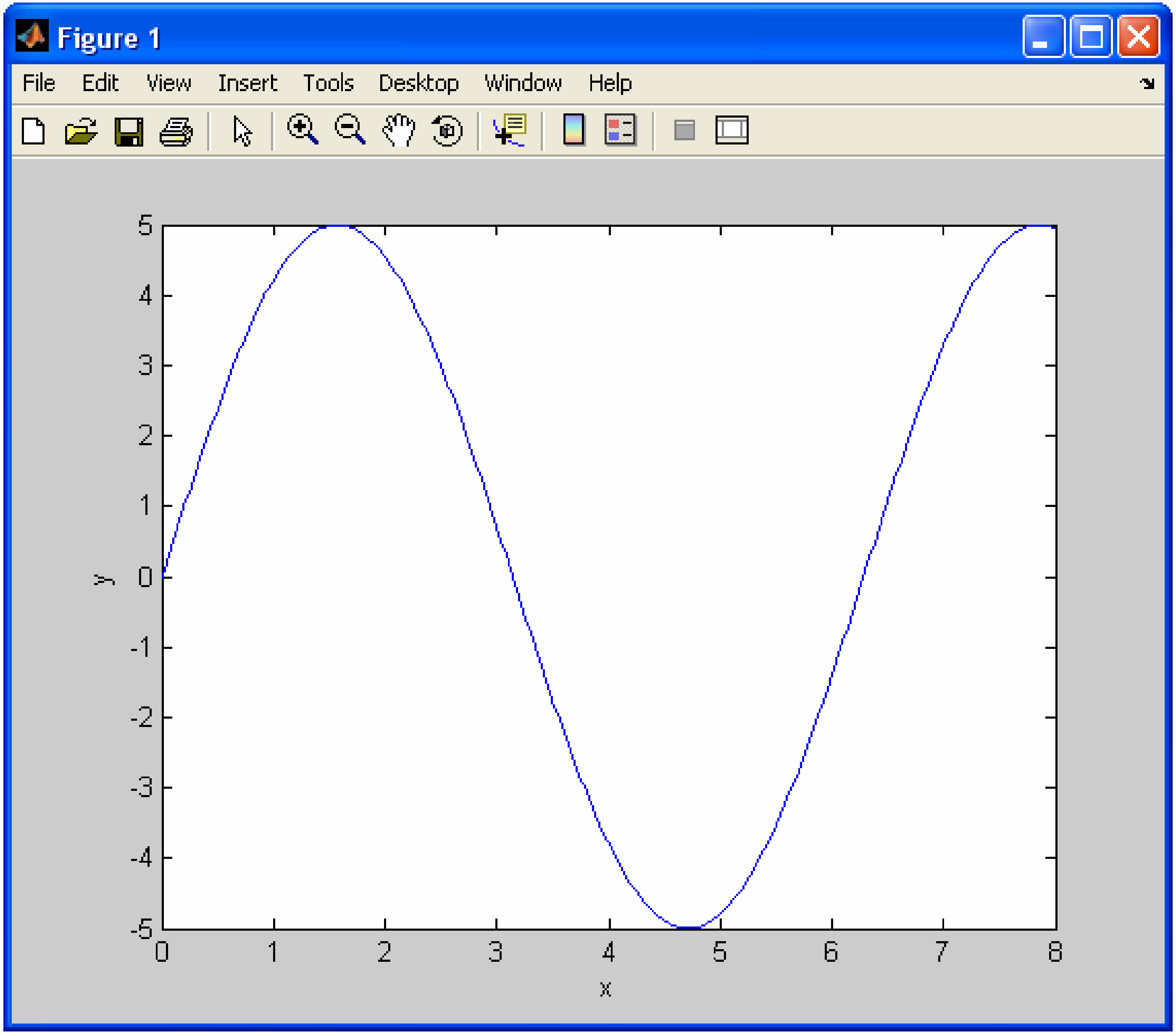
More? See pages 292-298.

The interactive plotting environment in MATLAB is a set of tools for:

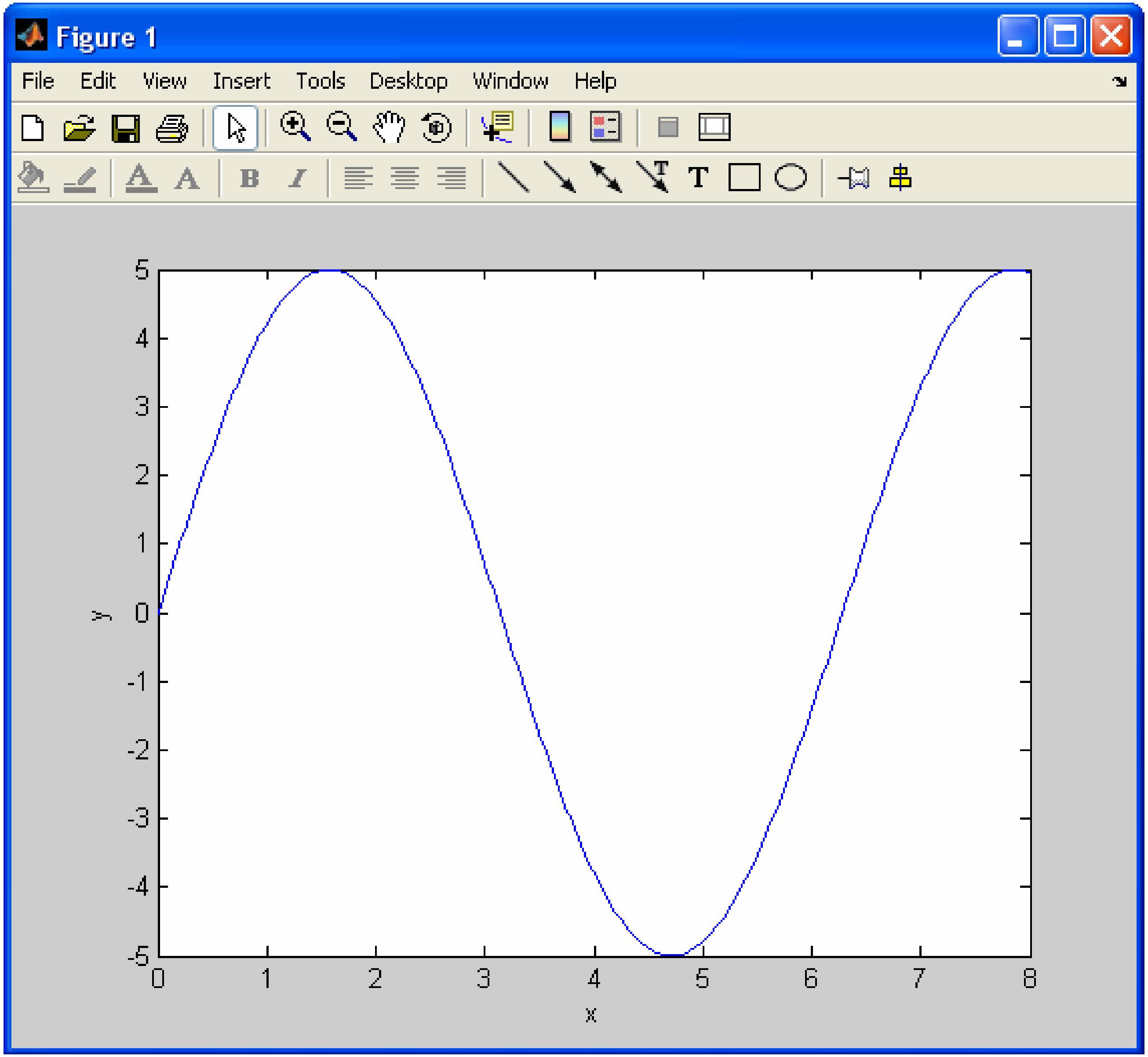
- Creating different types of graphs,
- Selecting variables to plot directly from the Workspace Browser,
- Creating and editing subplots,
- Adding annotations such as lines, arrows, text, rectangles, and ellipses, and
- Editing properties of graphics objects, such as their color, line weight, and font.

The Figure window with the Figure toolbar displayed.

Figure 5.4–1



The Figure window with the Figure and Plot Edit toolbars displayed. Figure 5.4–2

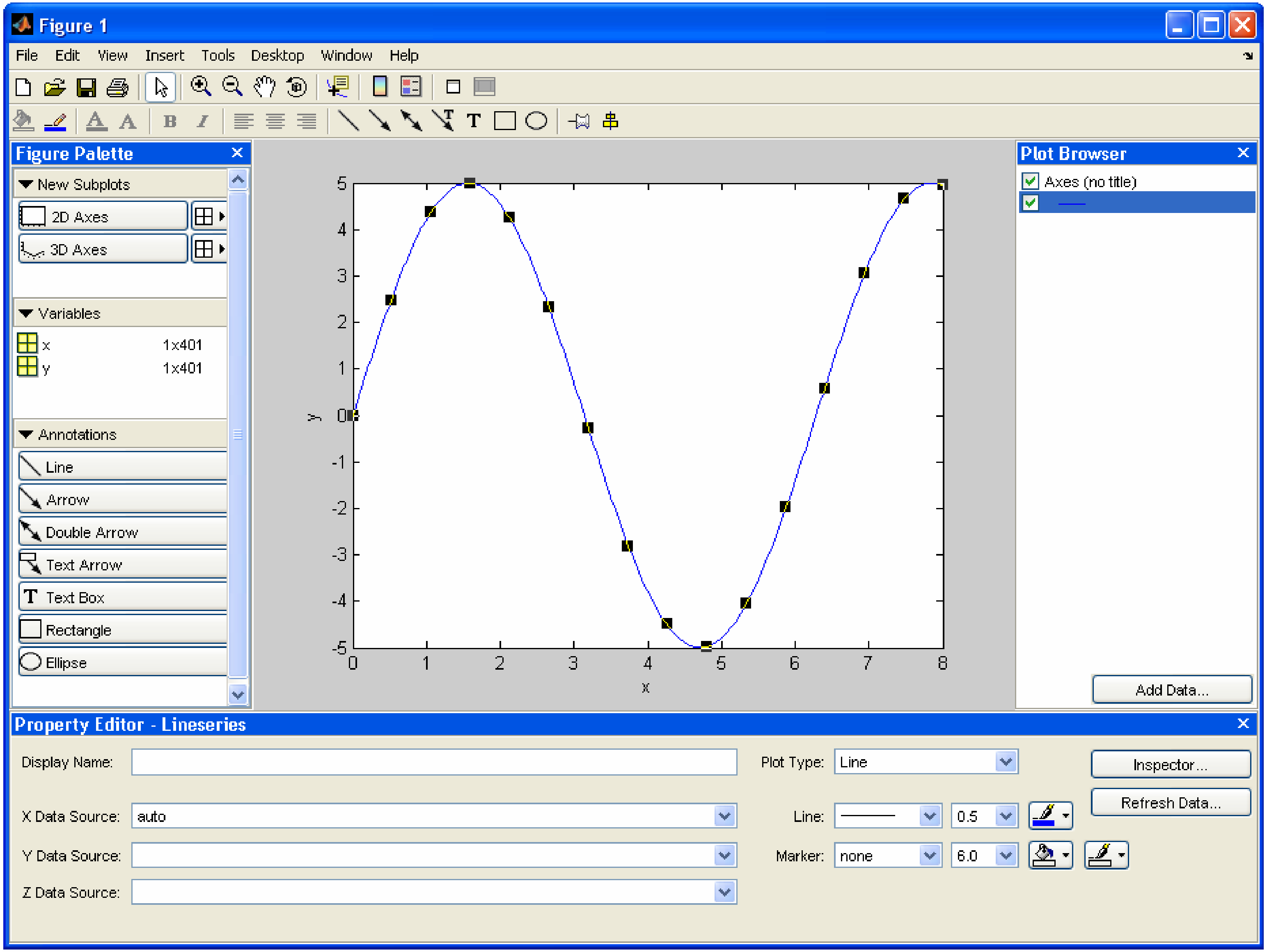


The Plot Tools interface includes the following three panels associated with a given figure.

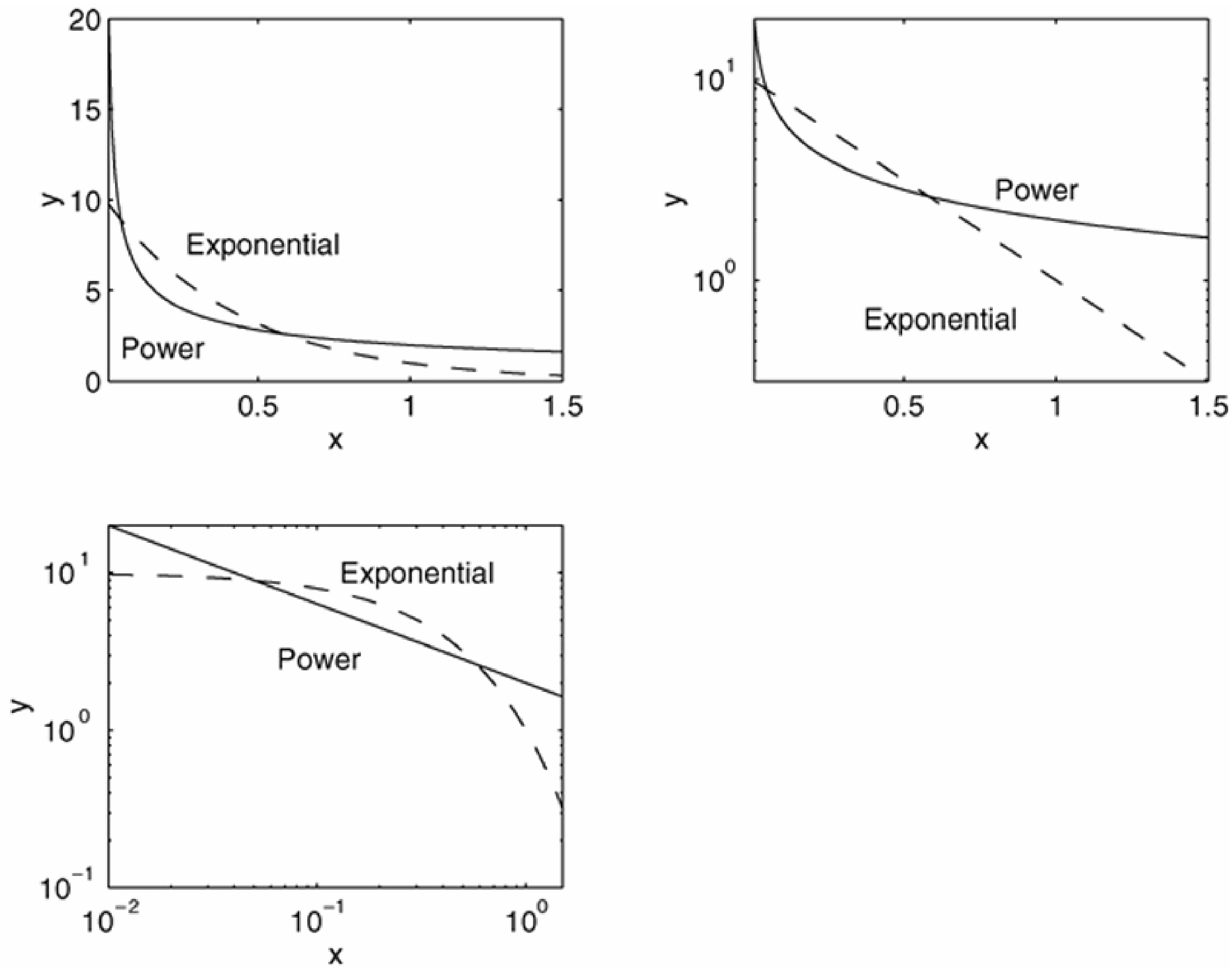
- **The Figure Palette:** Use this to create and arrange subplots, to view and plot workspace variables, and to add annotations.
- **The Plot Browser:** Use this to select and control the visibility of the axes or graphics objects plotted in the figure, and to add data for plotting.
- **The Property Editor:** Use this to set basic properties of the selected object and to obtain access to all properties through the Property Inspector.

The Figure window with the Plot Tools activated.

Figure 5.4–3



Function Discovery. The power function $y = 2x^{-0.5}$ and the exponential function $y = 10^{1-x}$. Figure 5.3–8



Using the Linear, Power, and Exponential Functions to Describe data.

Each function gives a straight line when plotted using a specific set of axes:

1. The linear function $y = mx + b$ gives a straight line when plotted on rectilinear axes. Its slope is m and its intercept is b.

2. The power function $y = bx^m$ gives a straight line when plotted on log-log axes.

3. The exponential function $y = b(10)^{mx}$ and its equivalent form $y = be^{mx}$ give a straight line when plotted on a semilog plot whose y-axis is logarithmic.

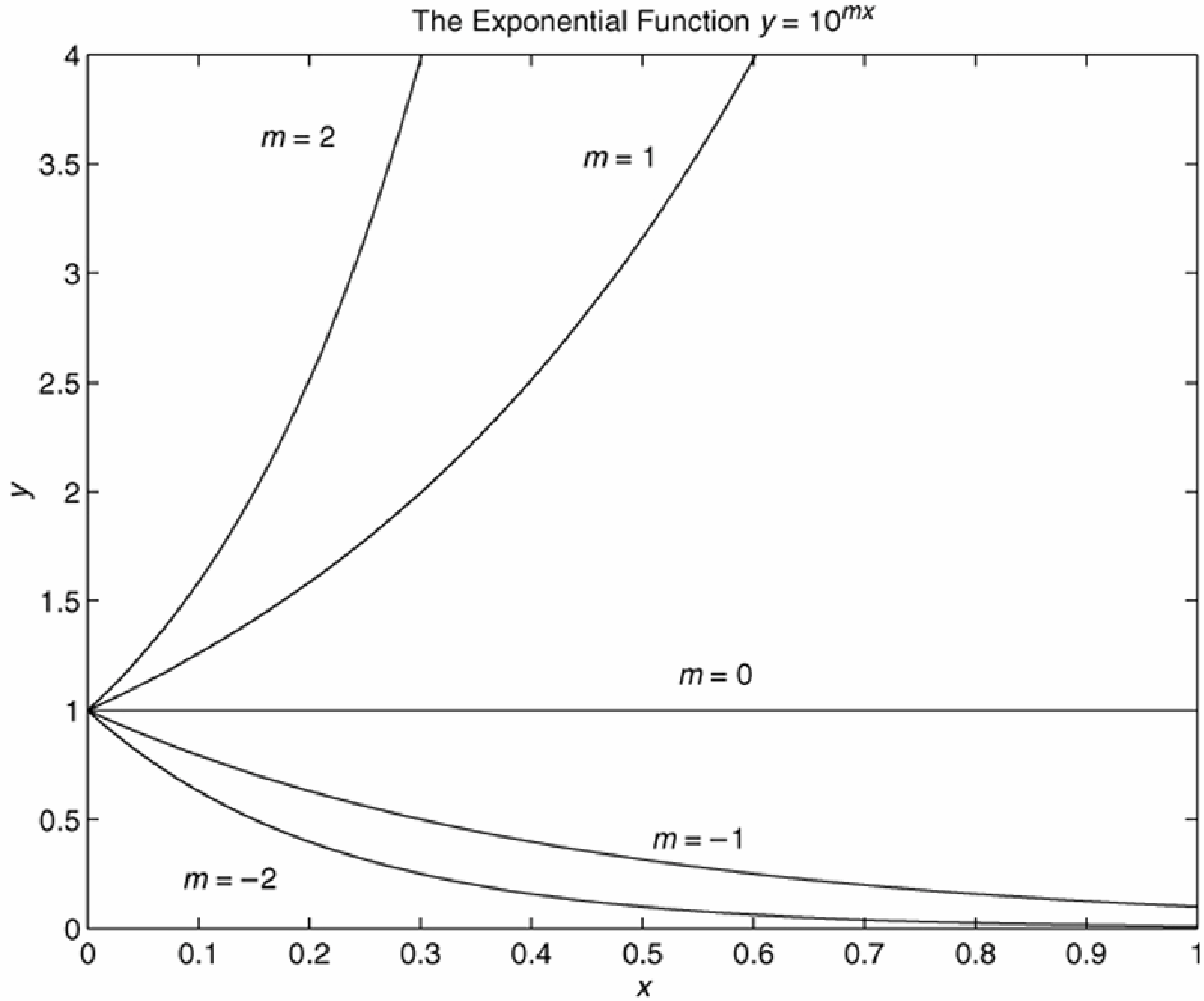
More? See pages 299-300.

Steps for Function Discovery

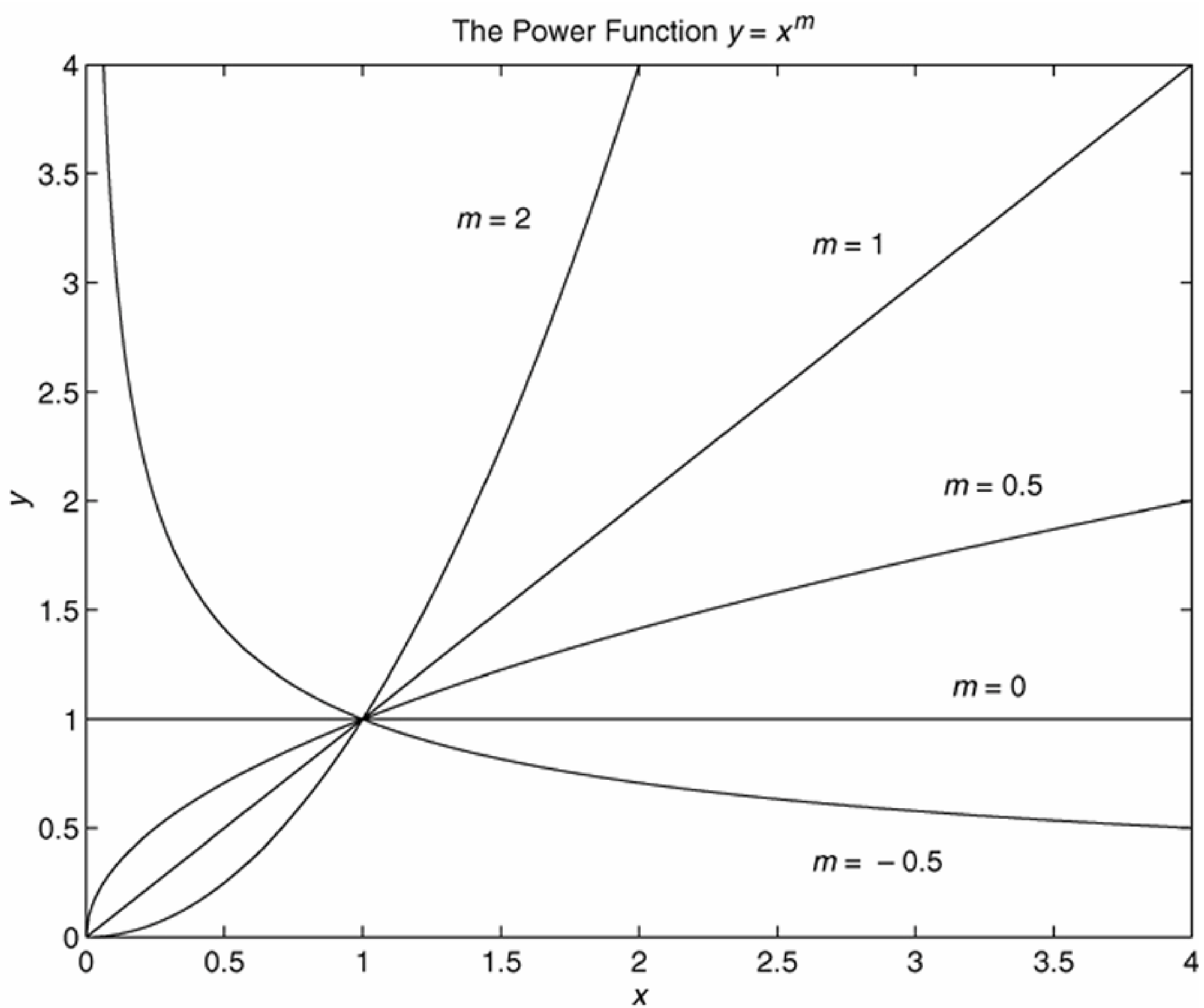
1. Examine the data near the origin. The exponential function can never pass through the origin (unless of course $b = 0$, which is a trivial case). (See Figure 5.5–1 for examples with $b = 1$.)

The linear function can pass through the origin only if $b = 0$. The power function can pass through the origin but only if $m > 0$. (See Figure 5.5–2 for examples with $b = 1$.)

Examples of exponential functions. Figure 5.5–1



Examples of power functions. Figure 5.5–2



Steps for Function Discovery (continued)

2. Plot the data using rectilinear scales. If it forms a straight line, then it can be represented by the linear function and you are finished. Otherwise, if you have data at $x = 0$, then

a. If $y(0) = 0$, try the power function.

b. If $y(0) \neq 0$, try the exponential function.

If data is not given for $x = 0$, proceed to step 3.

(continued…)

Steps for Function Discovery (continued)

3. If you suspect a power function, plot the data using log-log scales. Only a power function will form a straight line on a log-log plot. If you suspect an exponential function, plot the data using the semilog scales. Only an exponential function will form a straight line on a semilog plot.

(continued…)

Steps for Function Discovery (continued)

4. In function discovery applications, we use the log-log and semilog plots *only* to identify the function type, but not to find the coefficients b and m. The reason is that it is difficult to interpolate on log scales.

The `polyfit` function. Table 5.5–1

Command	Description
`p = polyfit(x,y,n)`	Fits a polynomial of degree *n* to data described by the vectors *x* and *y*, where *x* is the independent variable. Returns a row vector `p` of length *n* + 1 that contains the polynomial coefficients in order of descending powers.

Using the `polyfit` Function to Fit Equations to Data.

Syntax: `p = polyfit(x,y,n)`

where x and y contain the data, `n` is the order of the polynomial to be fitted, and `p` is the vector of polynomial coefficients.

The linear function: $y = mx + b$. In this case the variables w and z in the polynomial $w = p_1 z + p_2$ are the original data variables x and y, and we can find the linear function that fits the data by typing `p = polyfit(x,y,1)`. The first element p_1 of the vector `p` will be m, and the second element p_2 will be b.

The power function: $y = bx^m$. In this case

$$\log_{10} y = m \log_{10} x + \log_{10} b \qquad (5.5\text{–}5)$$

which has the form

$$w = p_1 z + p_2$$

where the polynomial variables w and z are related to the original data variables x and y by $w = \log_{10} y$ and $z = \log_{10} x$. Thus we can find the power function that fits the data by typing

```
p = polyfit(log10(x),log10(y),1)
```

The first element p_1 of the vector `p` will be m, and the second element p_2 will be $\log_{10} b$. We can find b from $b = 10^{p_2}$.

The exponential function: $y = b(10)^{mx}$. In this case

$$\log_{10} y = mx + \log_{10} b \qquad (5.5\text{–}6)$$

which has the form

$$w = p_1 z + p_2$$

where the polynomial variables w and z are related to the original data variables x and y by $w = \log_{10} y$ and $z = x$. We can find the exponential function that fits the data by typing

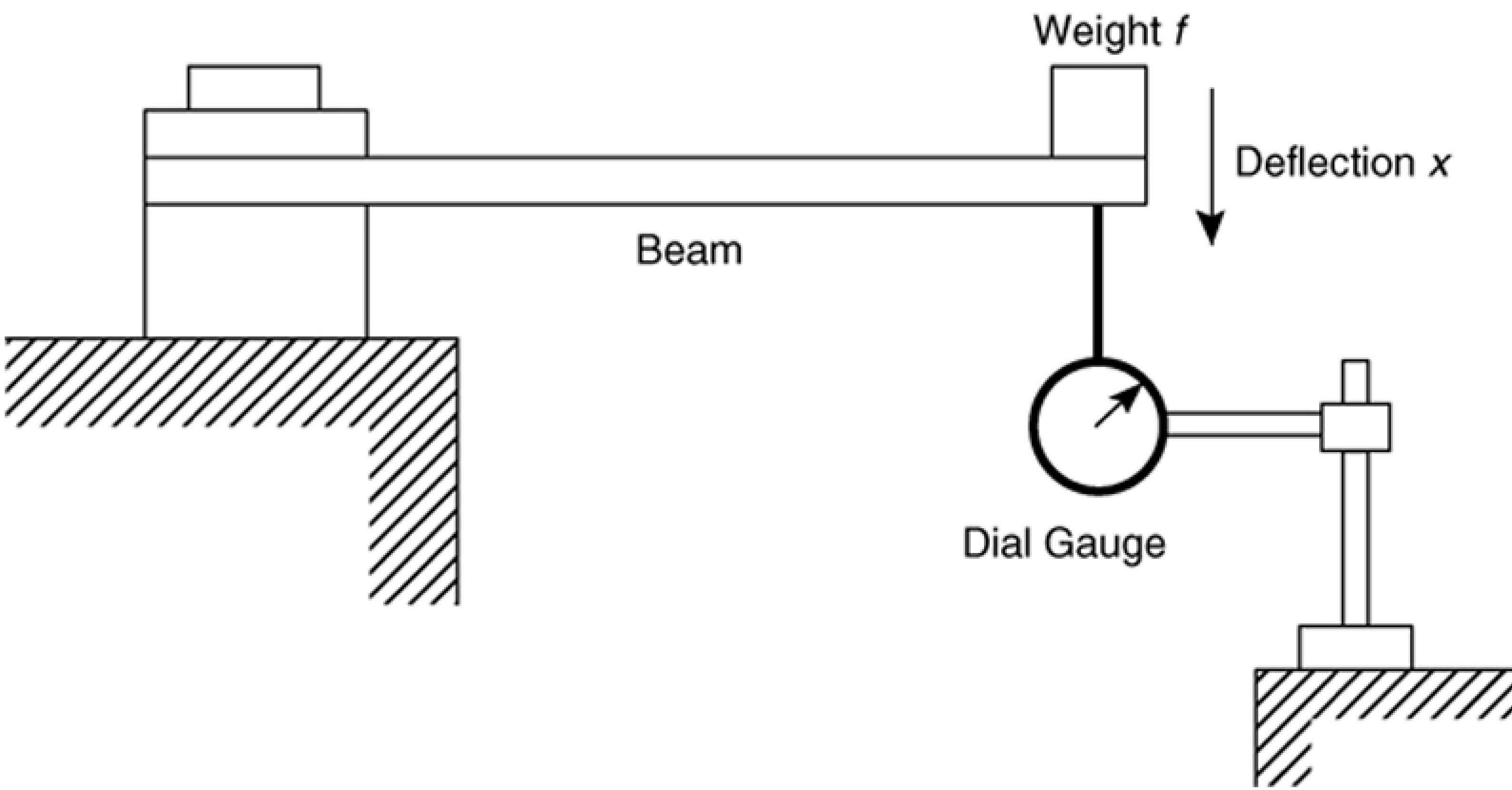
```
p = polyfit(x, log10(y),1)
```

The first element p_1 of the vector `p` will be m, and the second element p_2 will be $\log_{10} b$. We can find b from $b = 10^{p_2}$.

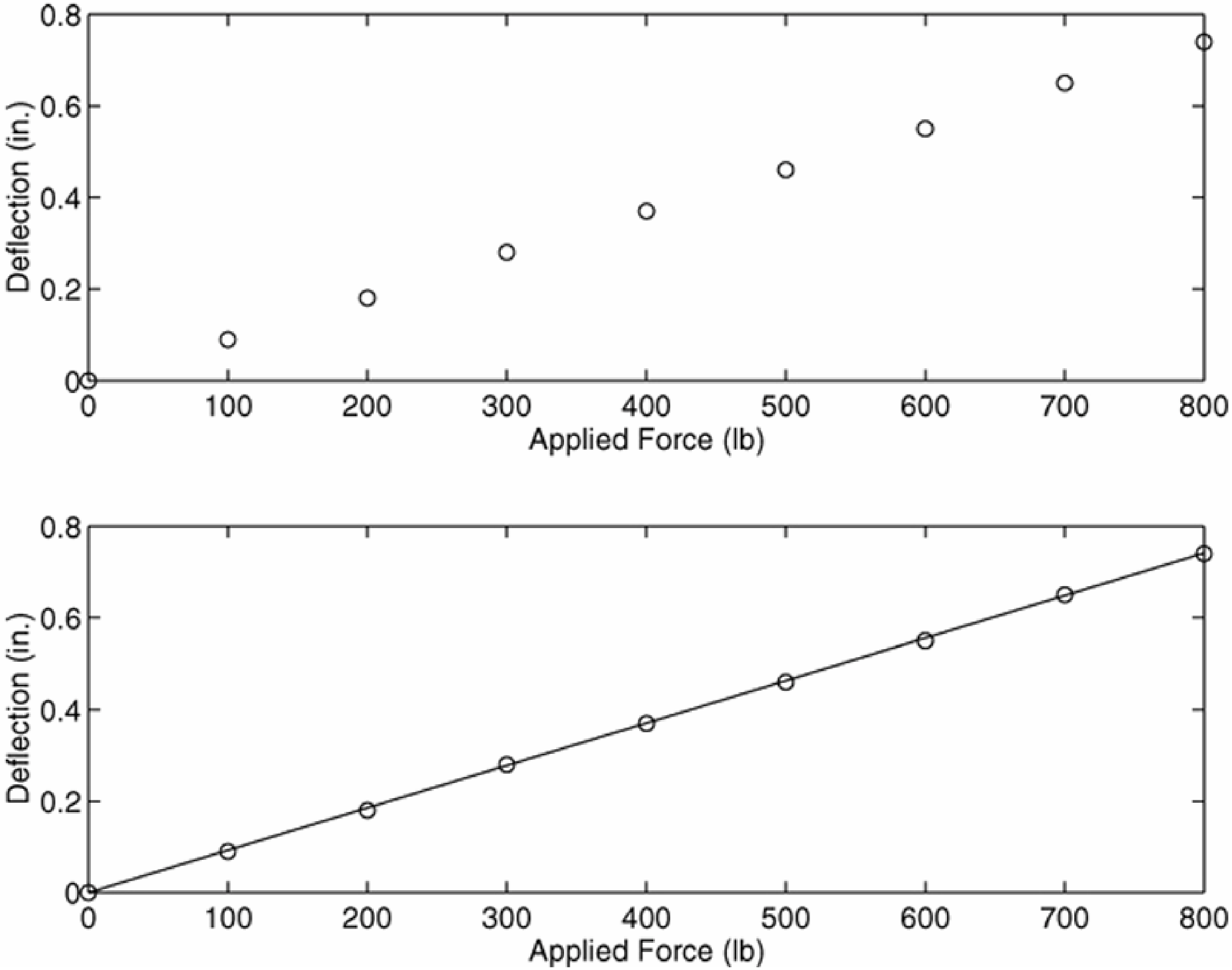
More? See pages 302-303.

Fitting a linear equation: An experiment to measure force and deflection in a cantilever beam. Example 5.5-1.

Figure 5.5–3

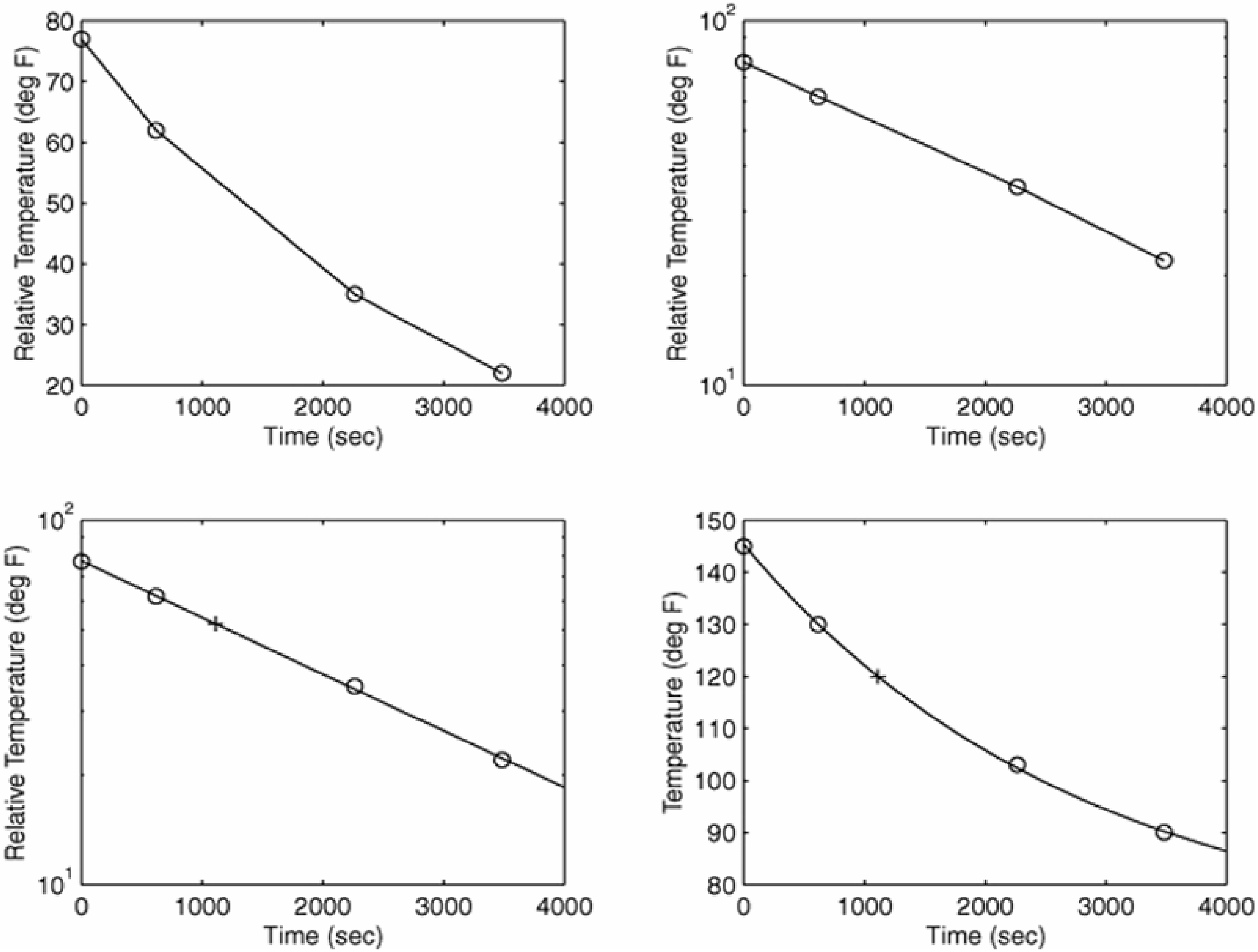


Plots for the cantilever beam example. Figure 5.5–4

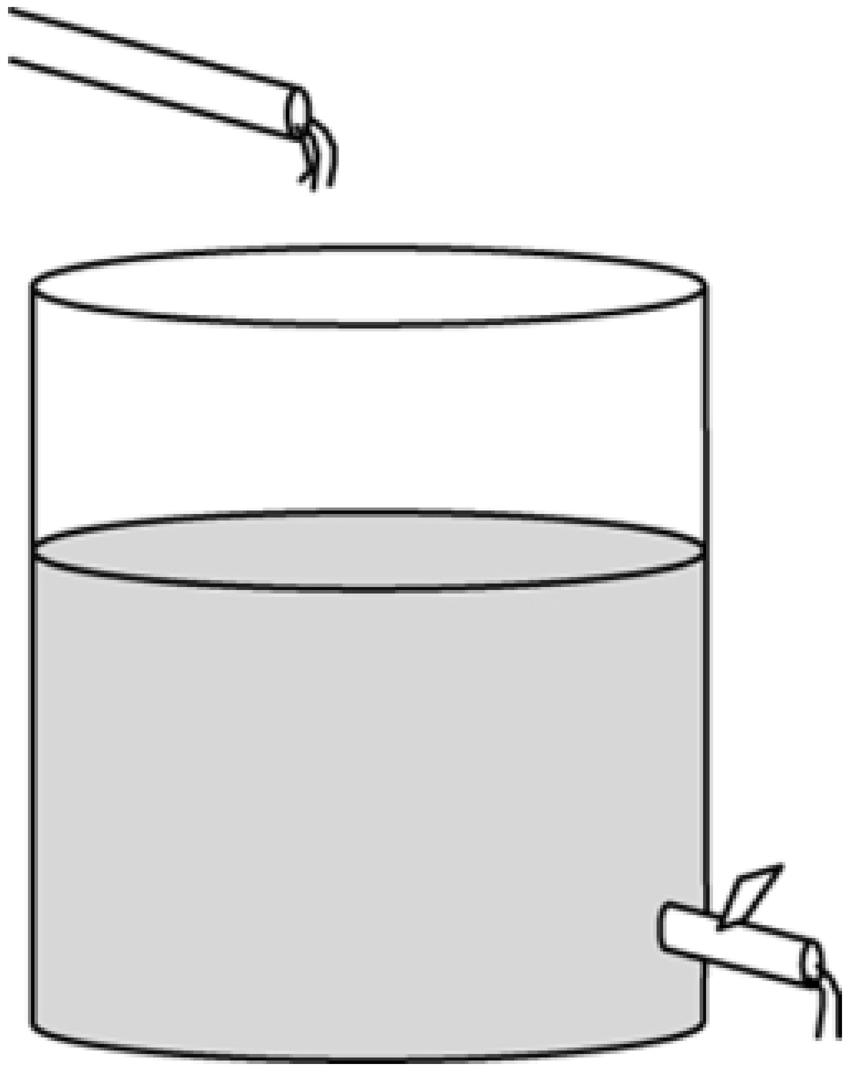


Fitting an exponential function. Temperature of a cooling cup of coffee, plotted on various coordinates. Example 5.5-2.

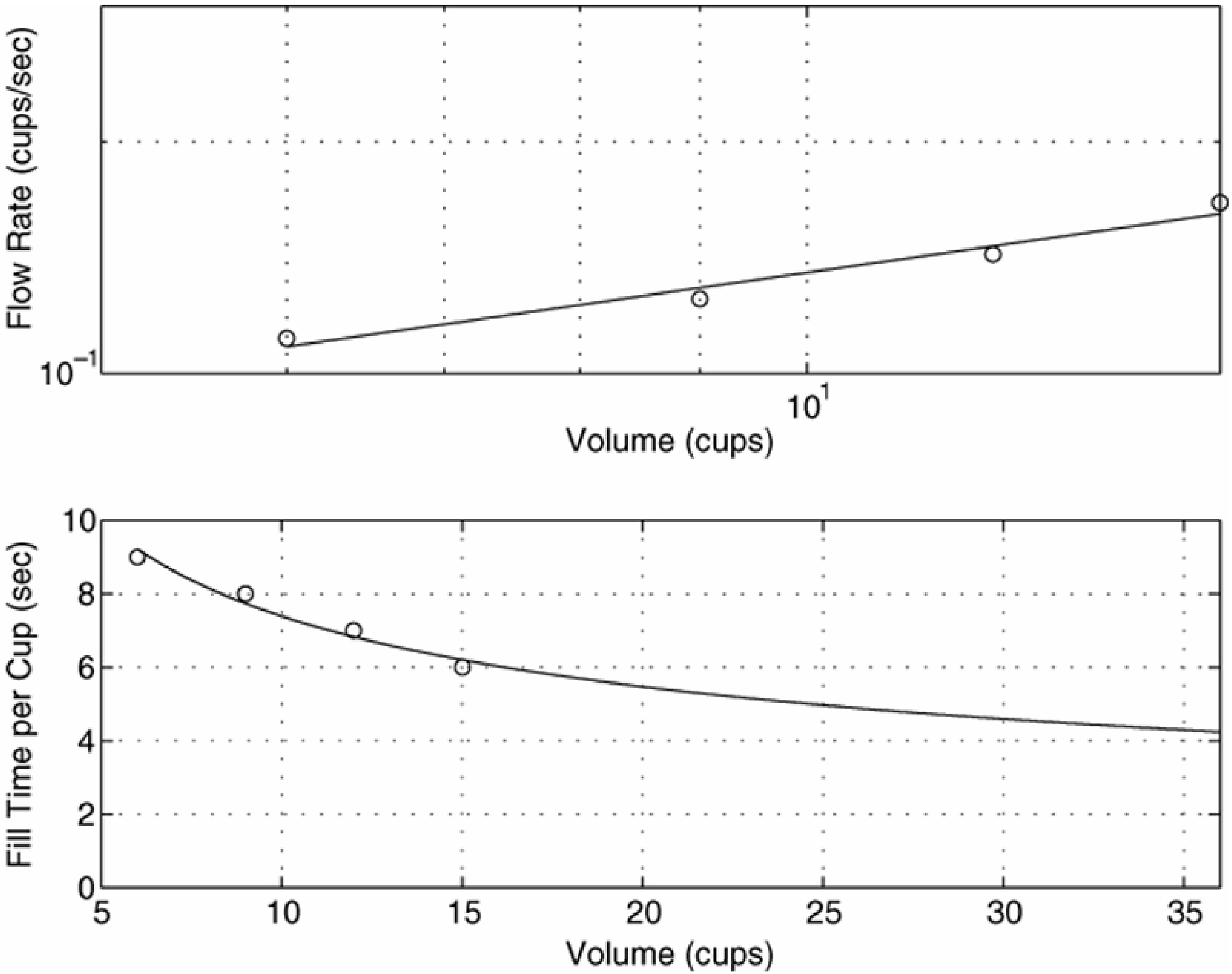
Figure 5.5–5



Fitting a power function. An experiment to verify Torricelli's principle. Example 5.5-3. Figure 5.5–6



Flow rate and fill time for a coffee pot. Figure 5.5–7

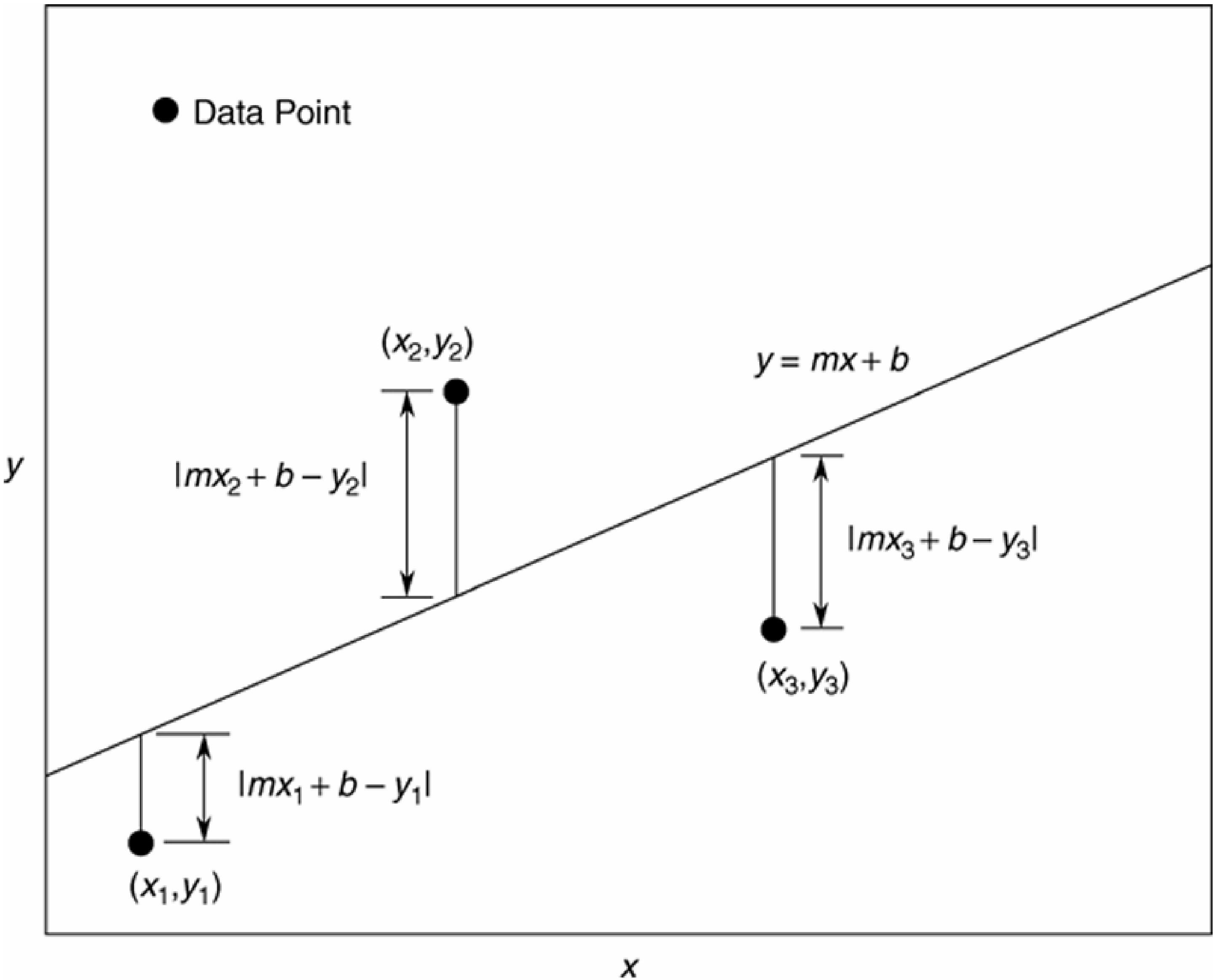


The Least Squares Criterion: used to fit a function $f(x)$. It minimizes the sum of the squares of the residuals, J. J is defined as

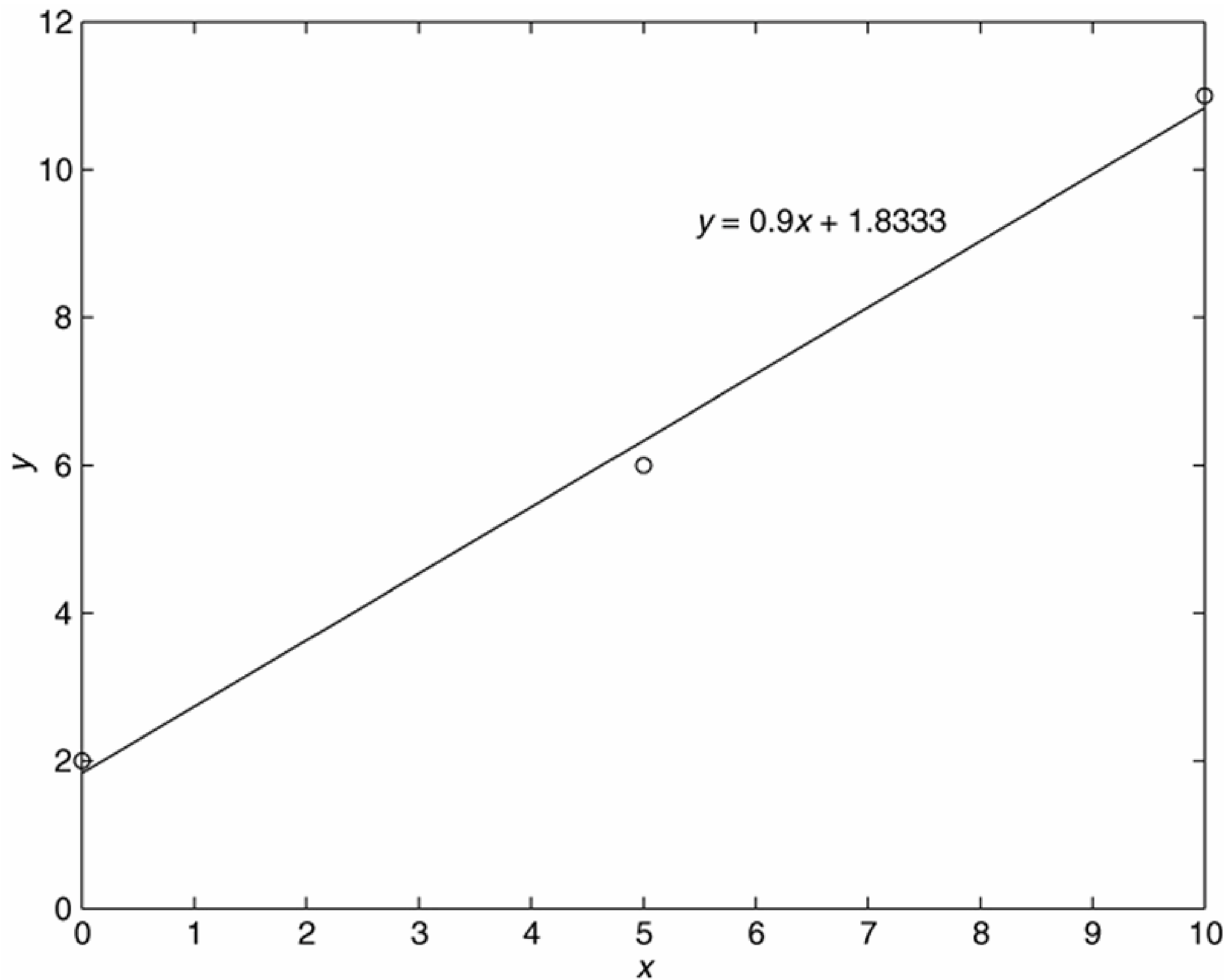
$$J = \sum_{i=1}^{m} [f(x_i) - y_i]^2 \qquad (5.6\text{–}1)$$

We can use this criterion to compare the quality of the curve fit for two or more functions used to describe the same data. The function that gives the smallest J value gives the best fit.

Illustration of the least squares criterion. Figure 5.6–1



The least squares fit for the example data. Figure 5.6–2



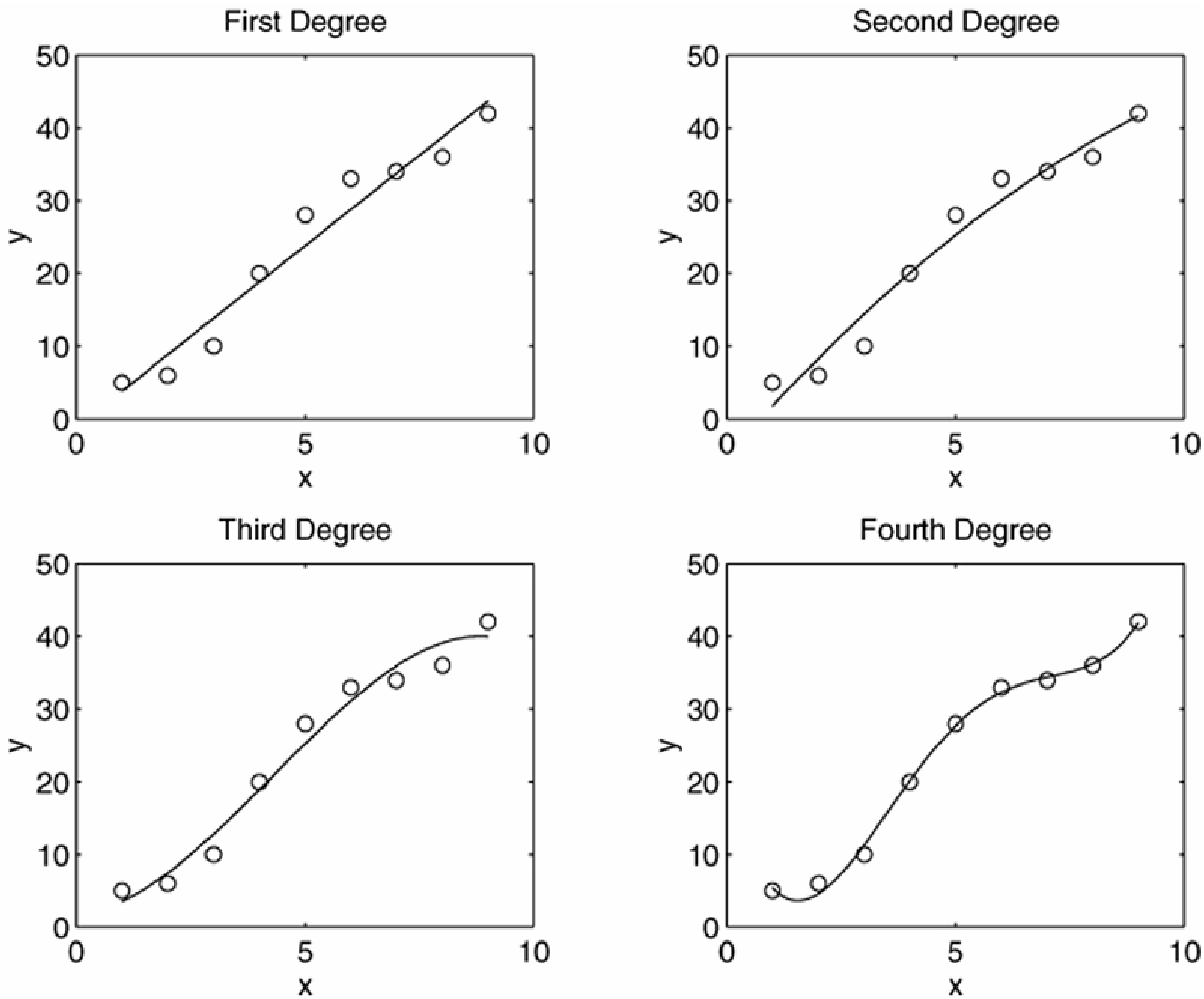
See pages 312-315.

The `polyfit` function is based on the least-squares method. Its syntax is

`p = polyfit(x,y,n)`	Fits a polynomial of degree `n` to data described by the vectors `x` and `y`, where `x` is the independent variable. Returns a row vector `p` of length `n+1` that contains the polynomial coefficients in order of descending powers.

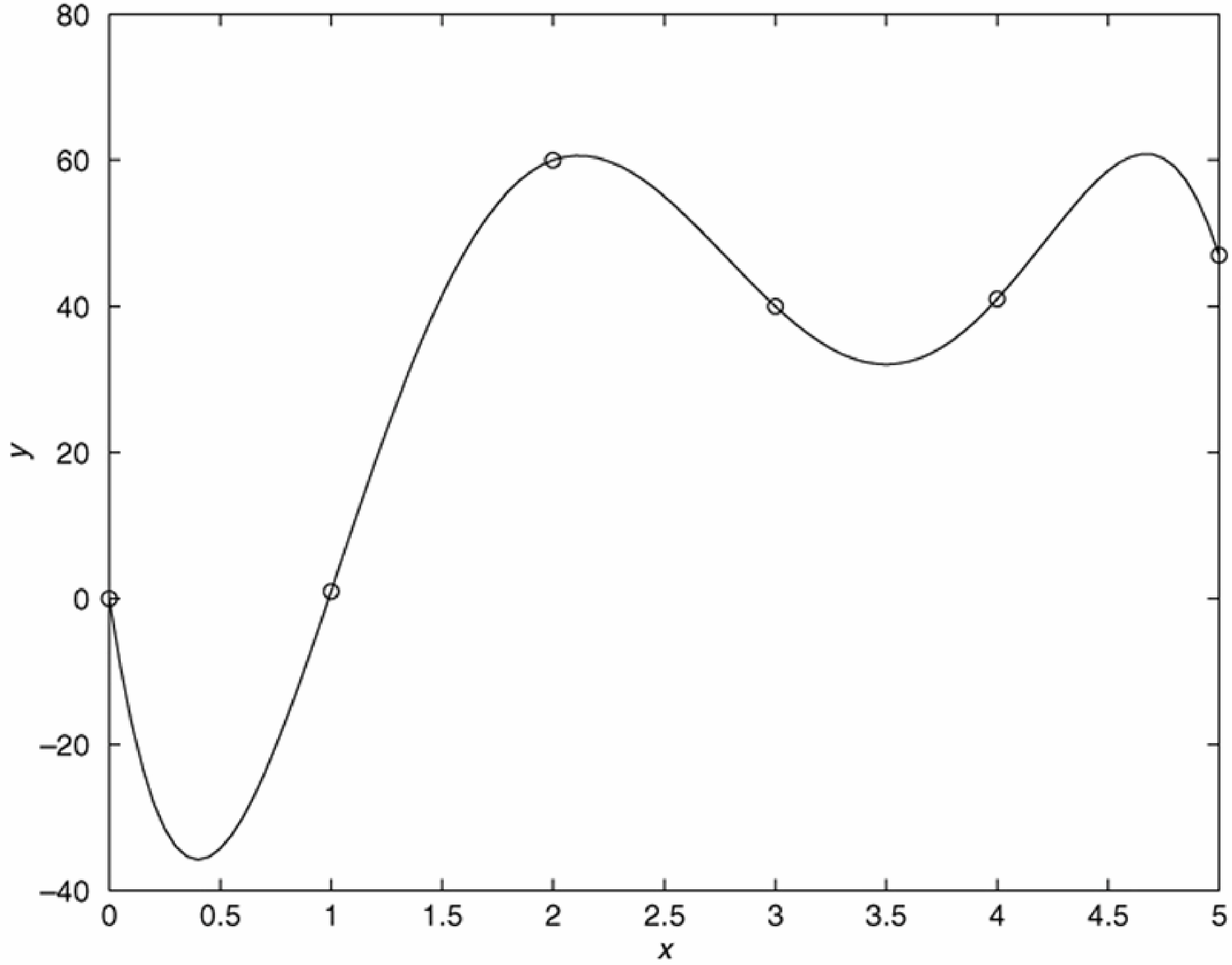
See page 315, Table 5.6-1.

Regression using polynomials of first through fourth degree.
Figure 5.6–3



The program is on pages 315 to 316.

Beware of using polynomials of high degree. An example of a fifth-degree polynomial that passes through all six data points but exhibits large excursions between points. Figure 5.6–4



Assessing the Quality of a Curve Fit:

Denote the sum of the squares of the deviation of the y values from their mean $\bar{y}$ by S, which can be computed from

$$S = \sum_{i=1}^{m} (y_i - \bar{y})^2 \qquad (5.6\text{–}2)$$

This formula can be used to compute another measure of the quality of the curve fit, the *coefficient of determination,* also known as the *r-squared value.* It is defined as

$$r^2 = 1 - \frac{J}{S} \qquad (5.6\text{–}3)$$

The value of S indicates how much the data is spread around the mean, and the value of J indicates how much of the data spread is unaccounted for by the model.

Thus the ratio J / S indicates the fractional variation unaccounted for by the model.

For a perfect fit, $J = 0$ and thus $r^2 = 1$. Thus the closer r^2 is to 1, the better the fit. The largest r^2 can be is 1.

It is possible for J to be larger than S, and thus it is possible for r^2 to be negative. Such cases, however, are indicative of a very poor model that should not be used.

As a rule of thumb, a good fit accounts for at least 99 percent of the data variation. This value corresponds to $r^2 \geq 0.99$.

More? See pages 319-320.

Scaling the Data

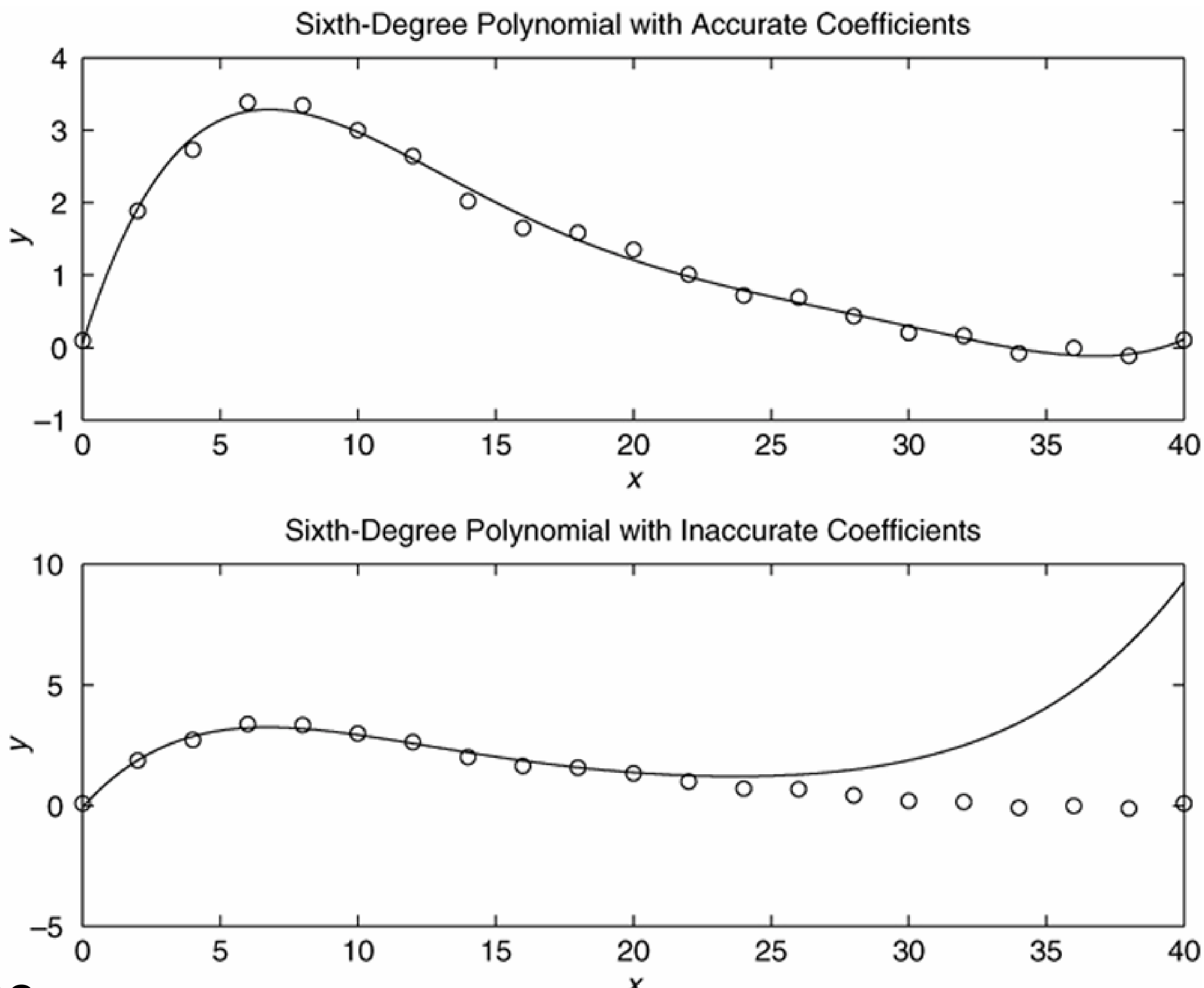
The effect of computational errors in computing the coefficients can be lessened by properly scaling the *x* values. You can scale the data yourself before using `polyfit`. Some common scaling methods are

1. Subtract the minimum *x* value or the mean *x* value from the *x* data, if the range of *x* is small, or

2. Divide the *x* values by the maximum value or the mean value, if the range is large.

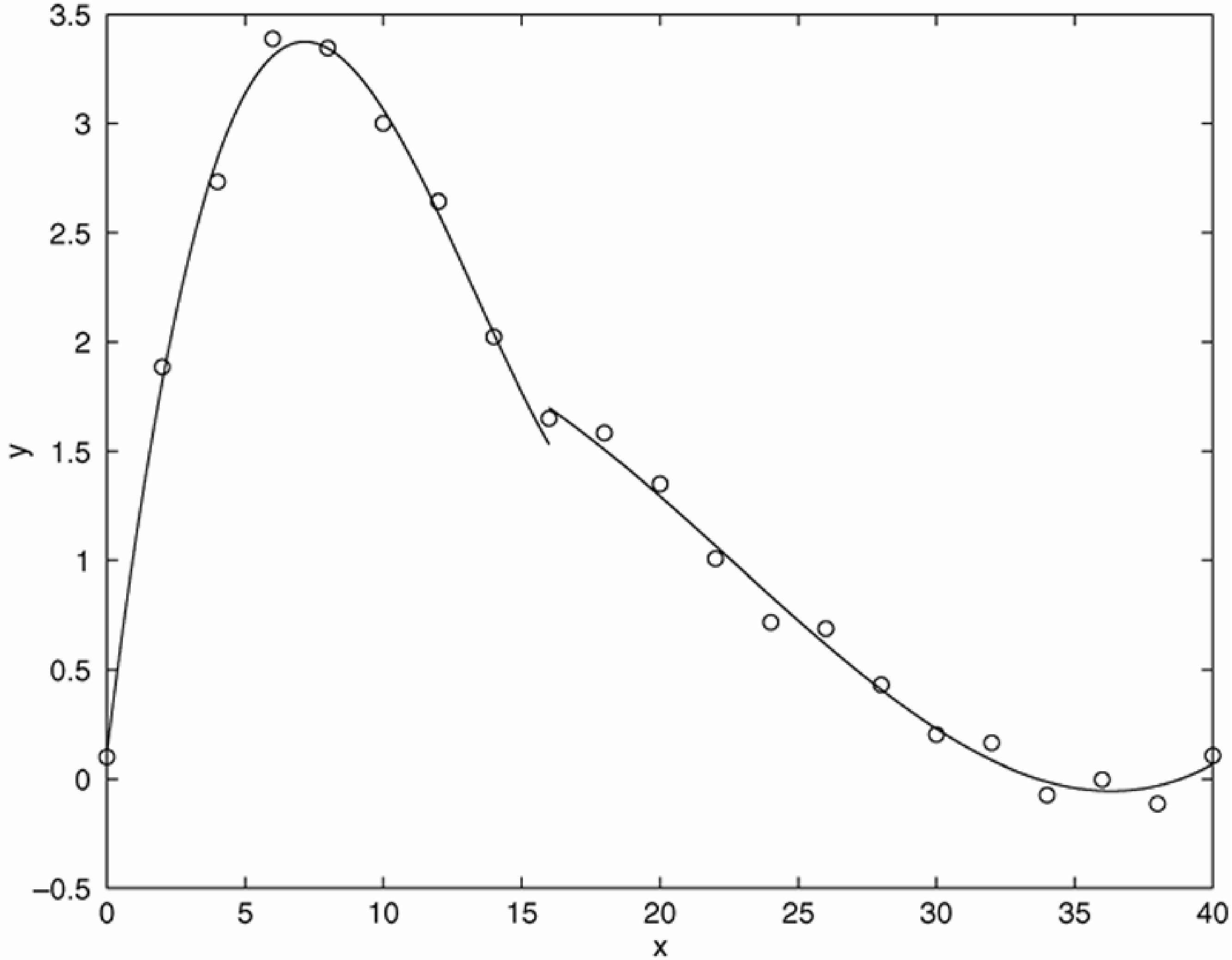
More? See pages 323-324.

Effect of coefficient accuracy on a sixth-degree polynomial. Top graph: 14 decimal-place accuracy. Bottom graph: 8 decimal-place accuracy. Figure 5.6–5



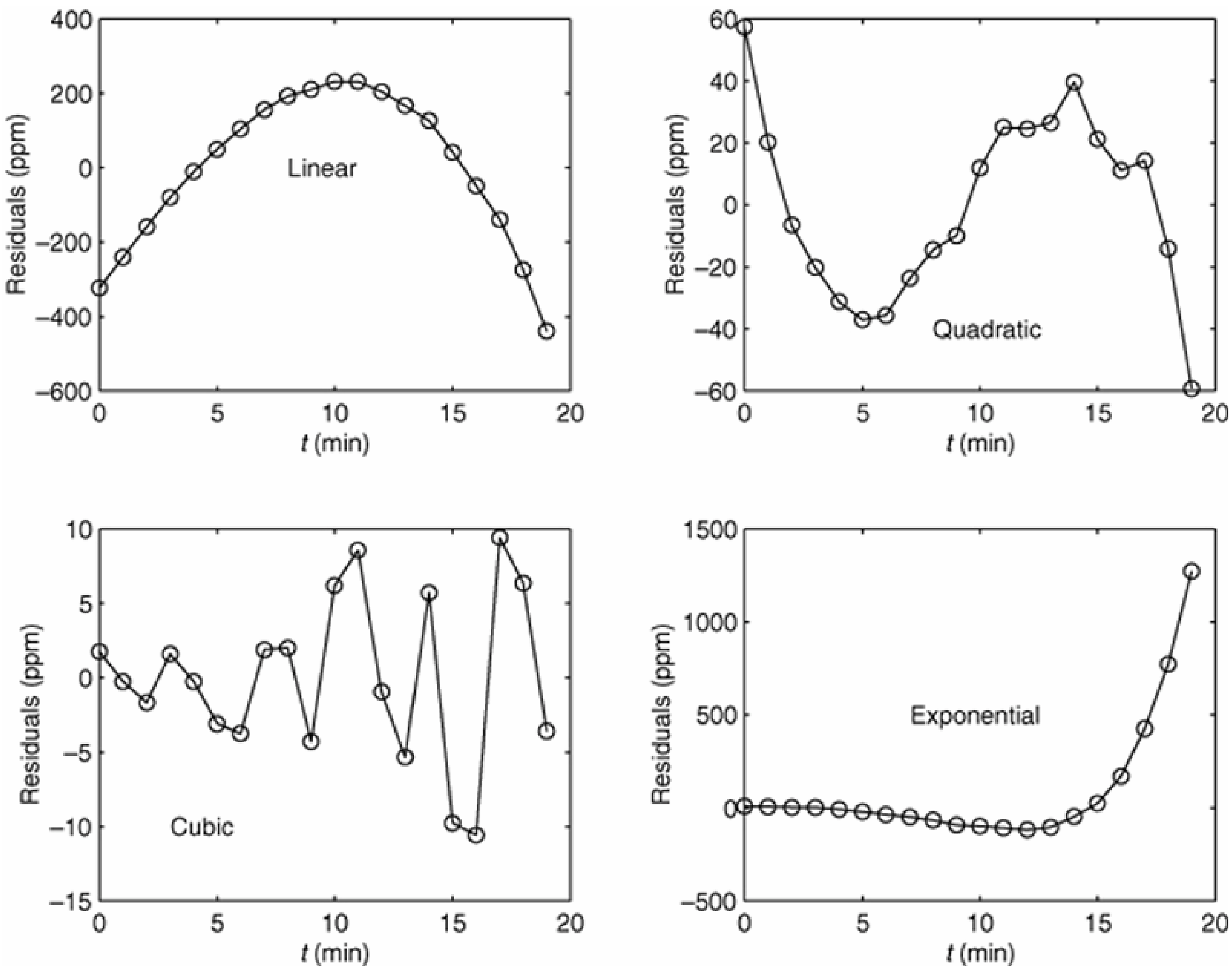
More? See pages 320-321.

Avoiding high degree polynomials: Use of two cubics to fit data. Figure 5.6–6



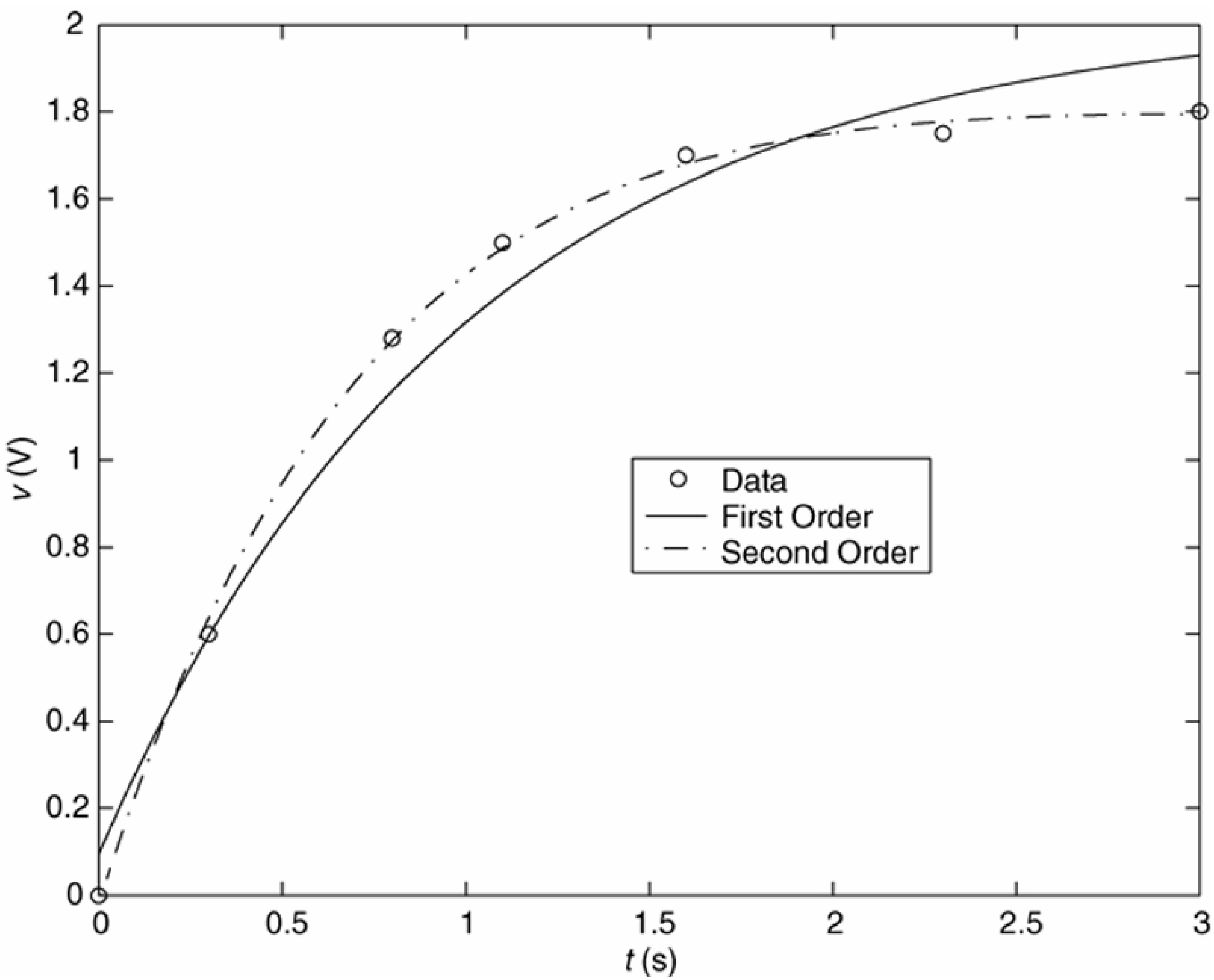
See pages 321-322.

Using Residuals: Residual plots of four models. Figure 5.6–7



See pages 325-326.

Linear-in-Parameters Regression: Comparison of first- and second-order model fits. Figure 5.6–8



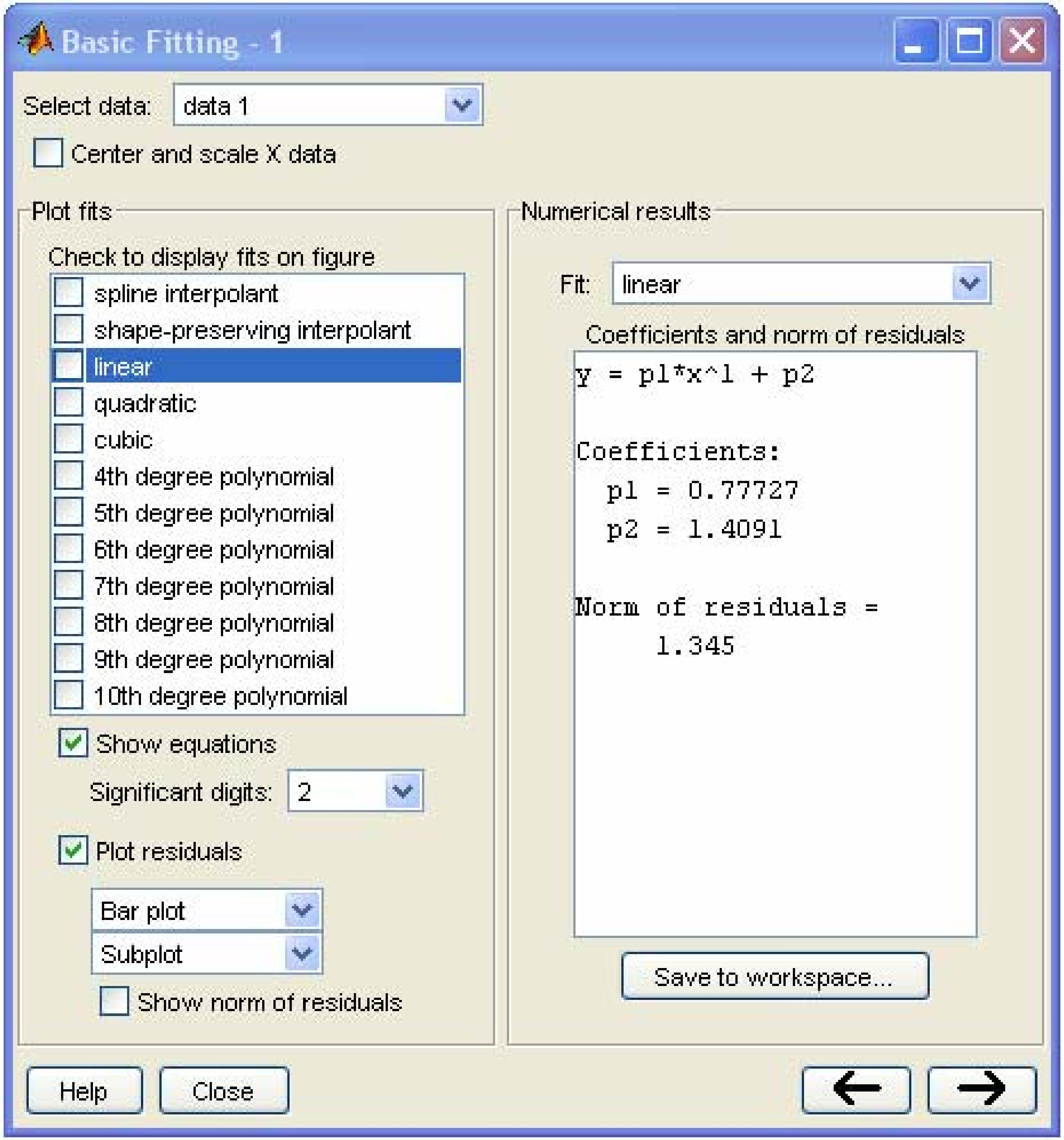
See pages 329-331.

Basic Fitting Interface

MATLAB supports curve fitting through the Basic Fitting interface. Using this interface, you can quickly perform basic curve fitting tasks within the same easy-to-use environment. The interface is designed so that you can:

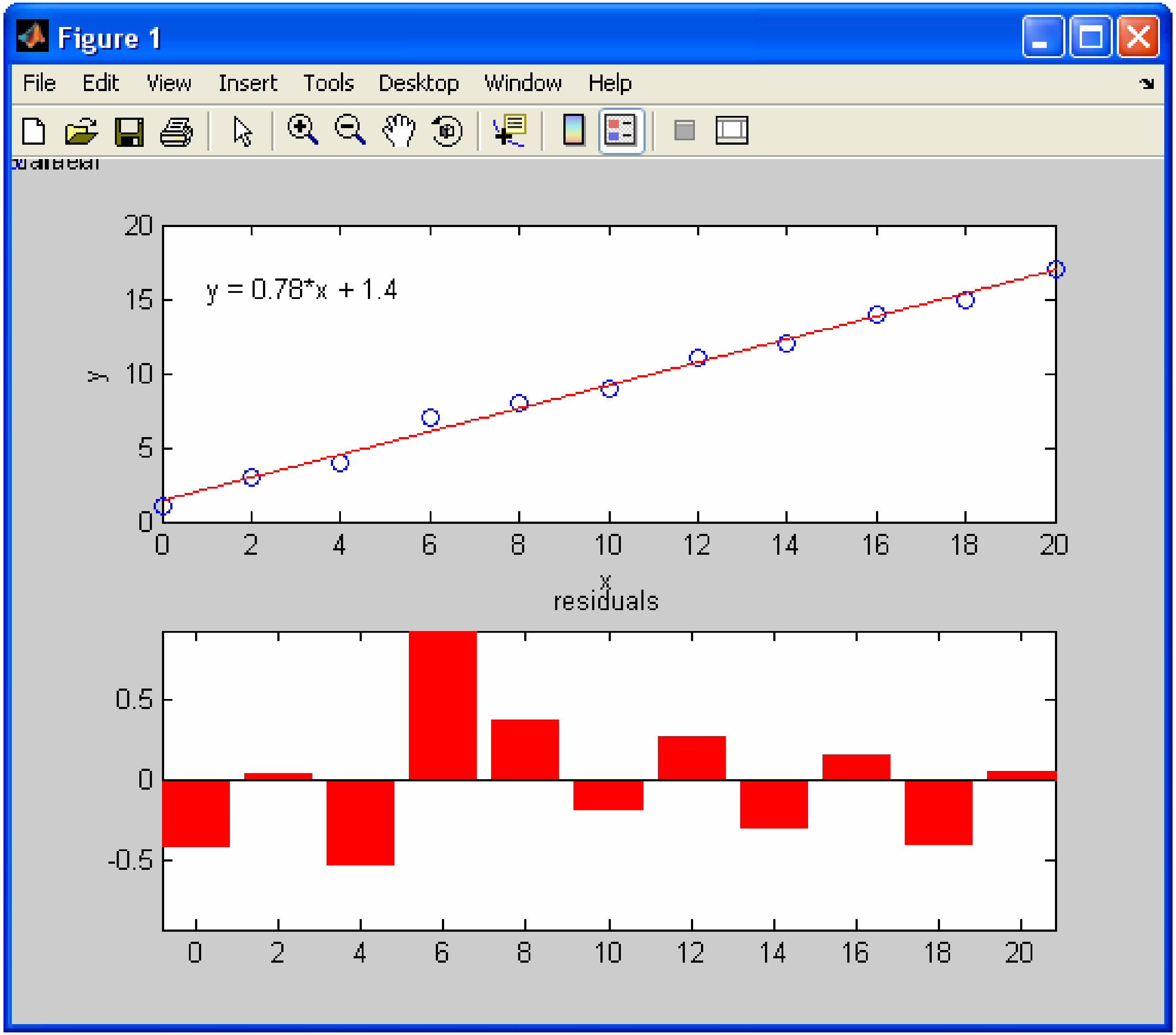
- Fit data using a cubic spline or a polynomial up to degree 10.
- Plot multiple fits simultaneously for a given data set.
- Plot the residuals.
- Examine the numerical results of a fit.
- Interpolate or extrapolate a fit.
- Annotate the plot with the numerical fit results and the norm of residuals.
- Save the fit and evaluated results to the MATLAB workspace.

The Basic Fitting interface. Figure 5.7–1



A figure produced by the Basic Fitting interface.

Figure 5.7–2



More? See pages 331-334.

Three-Dimensional Plotting in Matlab 7

In Chapter 1 (and more recently, in Chapter 5), Palm introduced you to the 2-D plot. You learned how to plot on rectilinear, semi-log, and log-log scales using the `plot` and `fplot` commands. Now Palm introduces you to the 3-D plot which is useful for visualizing certain functions of two variables (these functions are often difficult to visualize with a 2-D plot). The Matlab help feature provides a good overview of 3-D plotting too. `graph 3d`

With 2-D plots you learned about `xlabel` and `ylabel`. With 3-D plots you will add a third label, `zlabel`. All of the other features of plotting (e.g. grid, title, legend, etc.) work with 3-D plots. In Section 5-8, you will learn about 3-D line, surface, and contour plots.

$x = e^{-0.05t} \sin t$

$y = e^{-0.05t} \cos t$

$z = t$

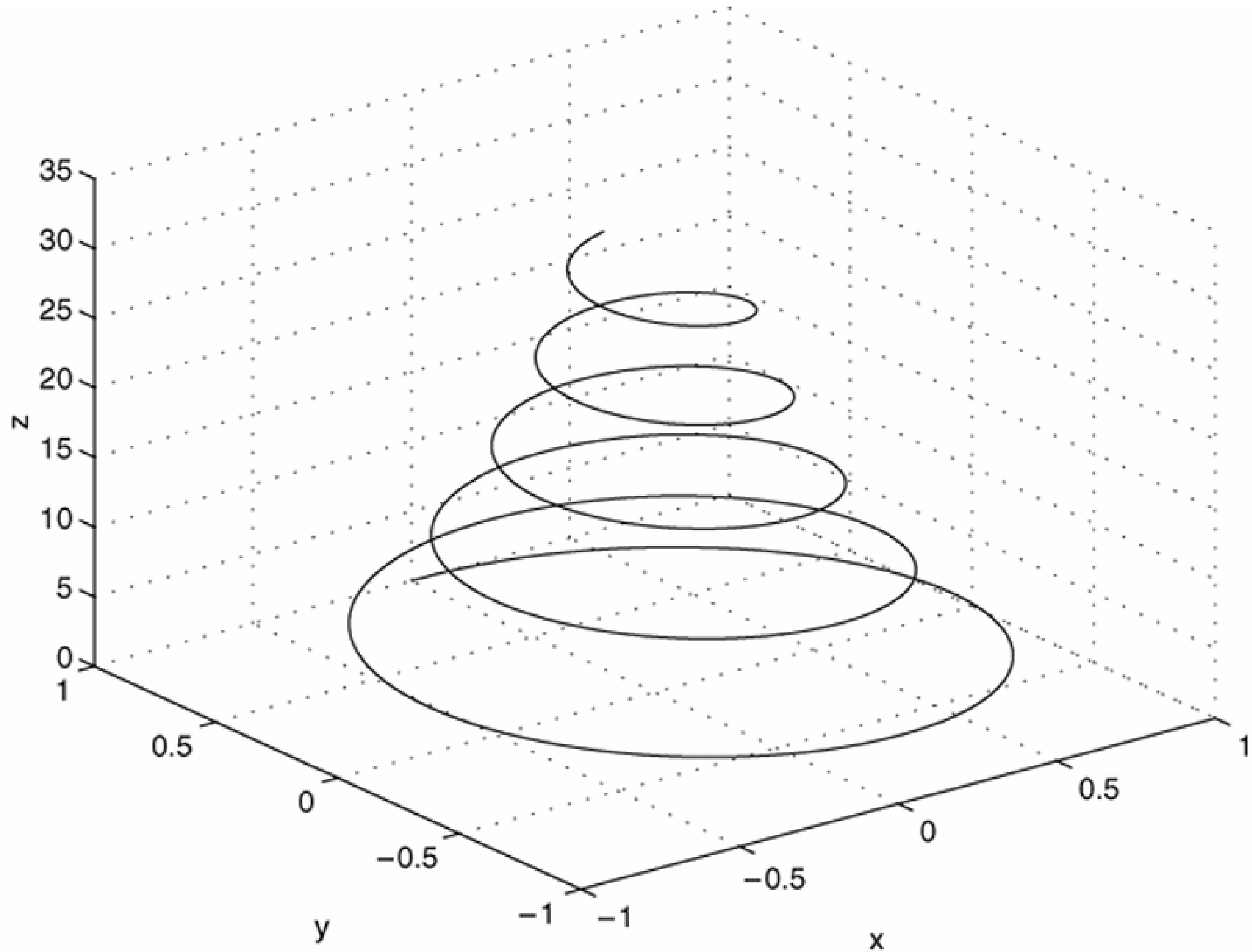
Three-Dimensional Line Plots:

The following program uses the `plot3` function to generate the spiral curve shown in Figure 5.8–1 (p.338).

```
>>t = [0:pi/50:10*pi];
>>plot3(exp(-0.05*t).*sin(t),...
exp(-0.05*t).*cos(t),t),...
xlabel('x'),ylabel('y'),zlabel('z'),grid
```

Remember: When you plot in 2-D and 3-D, don't forget to set your range and make sure you choose an increment that allows you to visualize your data.

The curve $x = e^{-0.05t} \sin t$, $y = e^{-0.05t} \cos t$, $z = t$ plotted with the `plot3` function. Figure 5.8–1



More? See pages 334-335.

Surface Plots: An Overview

The function $z = f(x,y)$ represents a surface when plotted on *xyz* axes, and the `mesh` function provides the means to generate a surface plot. Before you can use this function, you must generate a grid of points on the *xy* plane, and then evaluate the function $f(x,y)$ at these points. The `mesgrid` function generates the grid.

Continues on next slide……

Its syntax is `[X,Y] = meshgrid(x,y)`. If `x = [xmin:xspacing:xmax]` and

`Y = [ymin:yspacing:ymax]`, then this function will generate the coordinates of a rectangular grid with one corner at *(xmin, ymin)* and the opposite corner at *(xmax, ymax)*. Each rectangular panel in the grid will have a width equal to *xspacing* and a depth equal to *yspacing*. The resulting matrices `X` and `Y` contain the coordinate pairs of every point in the grid. These pairs are then used to evaluate the function.

The function `[X,Y] = meshgrid(x)` is the equivalent to `[X,Y] = meshgrid(x,x)` and can be used if *x* and *y* have the same minimum values, the same maximum values, and the same spacing. Using this form, you can type `[X,Y] = meshgrid (min:spacing:max)`, where `min` and `max` specify the minimum and maximum values of both *x* and *y* and `spacing` is the desired spacing of the *x* and *y* values.

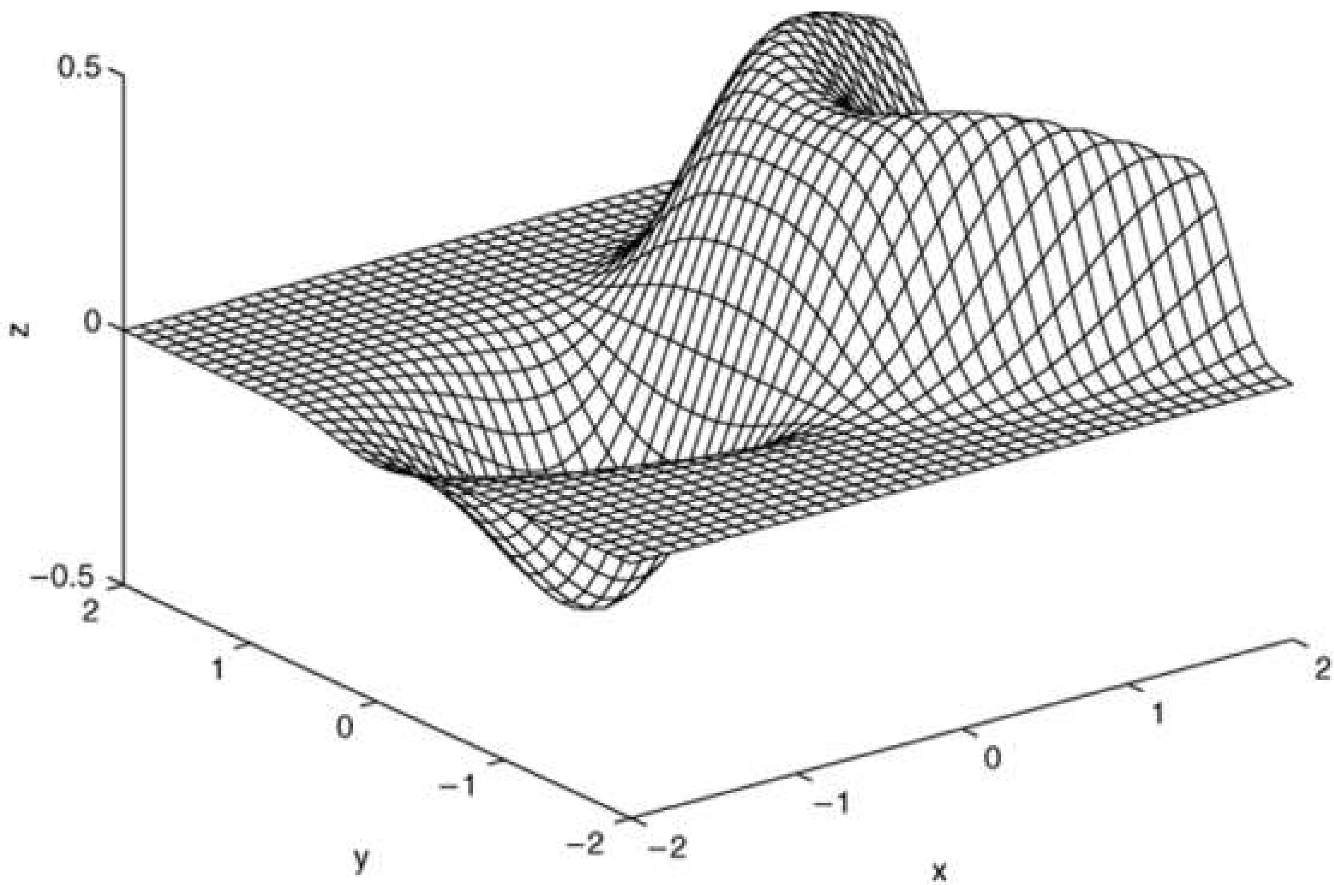
After the grid is computed, you create the surface plot with the `mesh` function. Its syntax is `mesh(x,y,z)`. As always, the grid, label, and text functions can be used with the `mesh` function.

The following session shows how to generate the surface plot of the function $z = xe^{-[(x-y^2)^2+y^2]}$, for $-2 \leq x \leq 2$ and $-2 \leq y \leq 2$, with a spacing of 0.1. This plot appears in Figure 5.8–2.

```
>>[X,Y] = meshgrid(-2:0.1:2);
>>Z = X.*exp(-((X-Y.^2).^2+Y.^2));
>>mesh(X,Y,Z),xlabel('x'),ylabel('y'),zlabel('z')
```

The next slide shows the resulting surface plot.

A plot of the surface $z = xe^{-[(x-y^2)^2+y^2]}$ created with the `mesh` function. Figure 5.8–2



More? See pages 335-336.

Contour Plots: An Overview

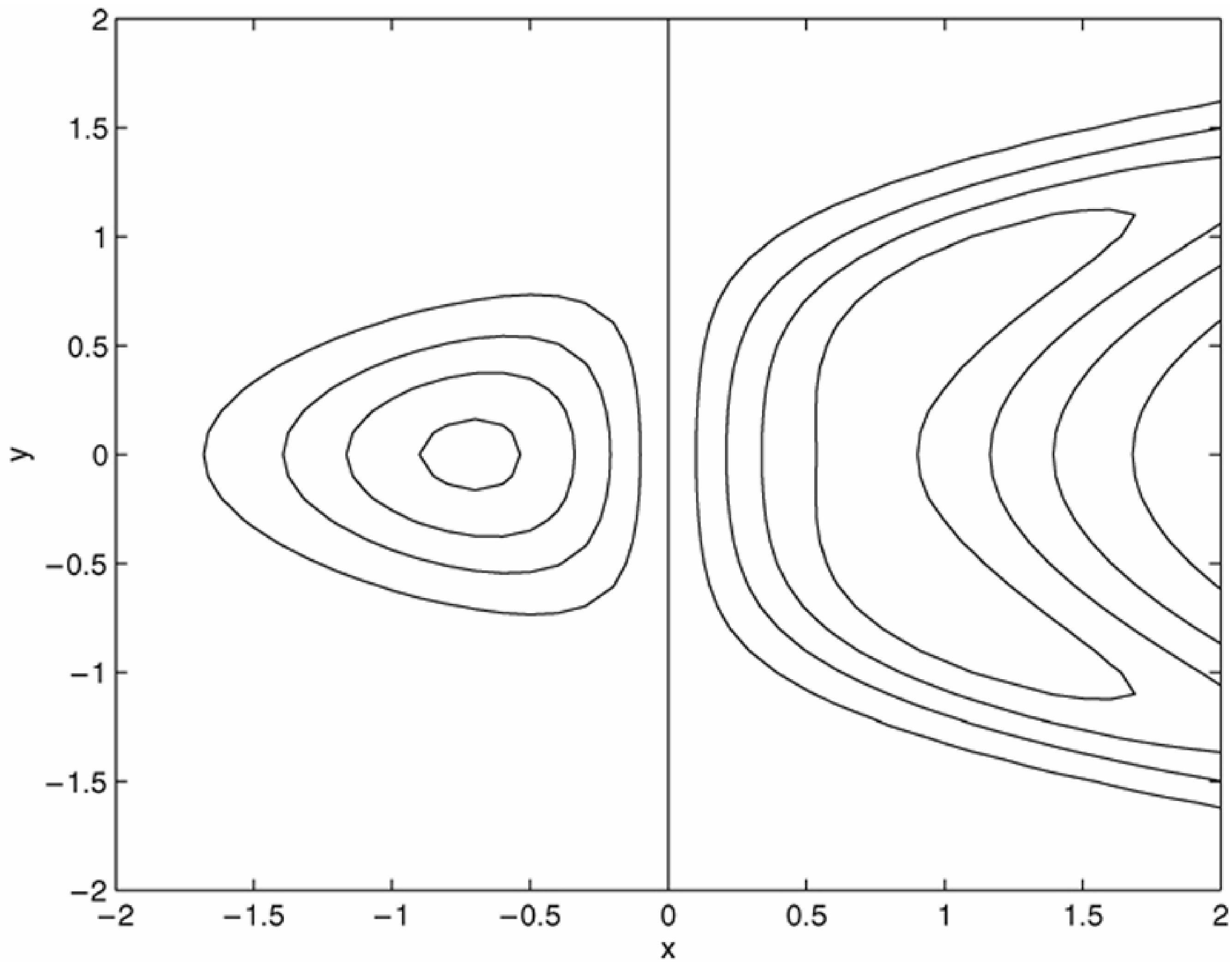
Topographic plots show the contours of the land by means of constant elevation lines also called *contour lines*. Such a plot is called a *contour plot*. These plots help engineers to visual the shape of a function. You use the `contour` function whose syntax is `contour(X,Y,Z)` and you use this function the same way you use the `mesh` function—first use the `meshgrid` function to generate the grid and then generate the function lines. The following slide provides a good example.

The following session generates the contour plot of the function whose surface plot is shown in Figure 5.8–2; namely, $z = xe^{-[(x-y^2)^2+y^2]}$, for $-2 \le x \le 2$ and $-2 \le y \le 2$, with a spacing of 0.1. This plot appears in Figure 5.8–3.

```
>>[X,Y] = meshgrid(-2:0.1:2);
>>Z = X.*exp(-((X- Y.^2).^2+Y.^2));
>>contour(X,Y,Z),xlabel('x'),ylabel('y')
```

See the next slide.

A contour plot of the surface $z = xe^{-[(x-y^2)^2+y^2]}$ created with the `contour` function. Figure 5.8–3



More? See page 337.

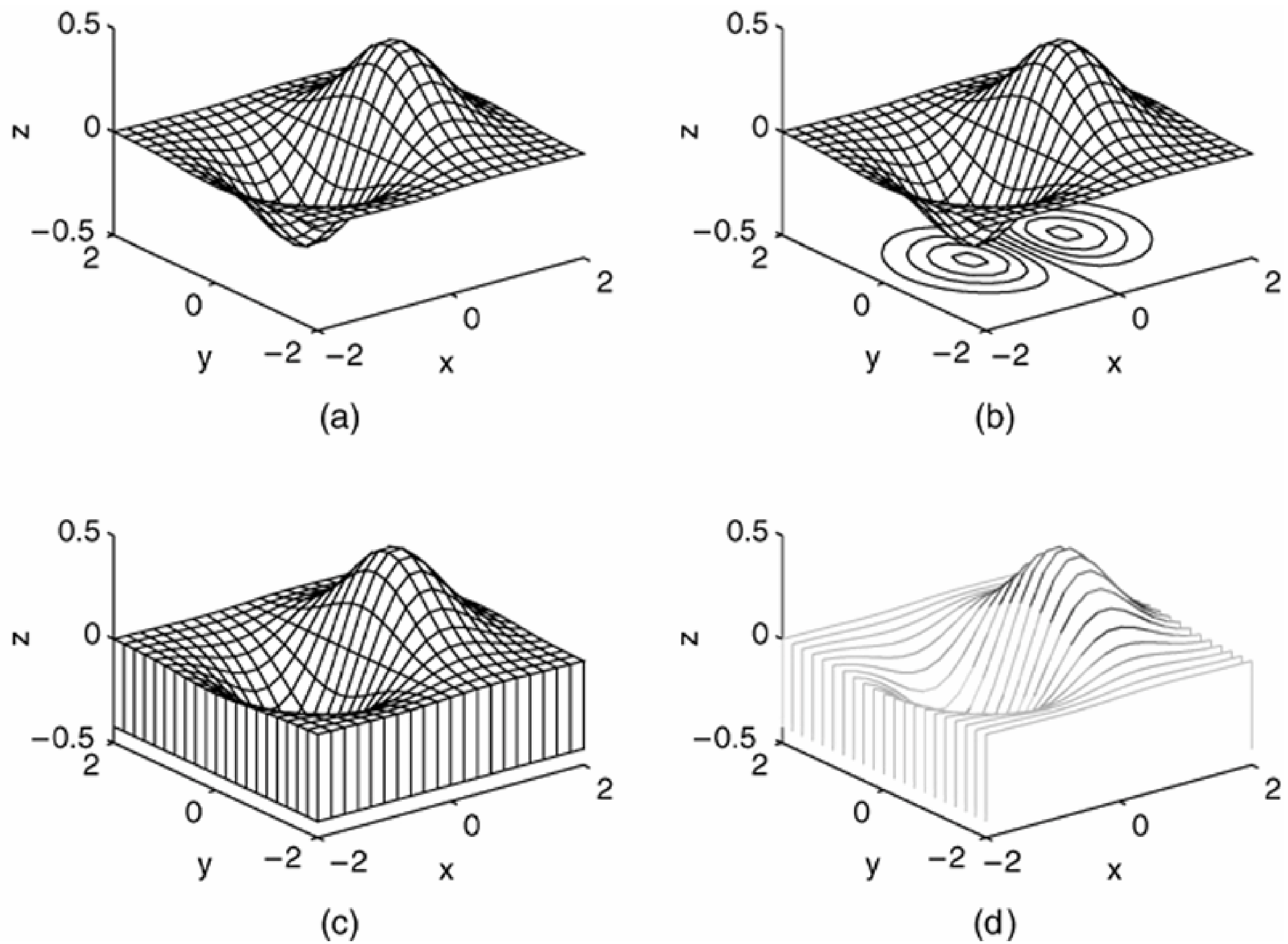
Contour plots and surface plots can be used together to clarify a function. For example, unless the elevations are labeled on contour lines, you cannot tell whether there is a minimum or a maximum point. However, a surface plot will provide that information. On the other hand, accurate measurements are not possible on a surface plot; these can be done on the contour plot because no distortion is involved.

The `meshc` function can be useful—it shows the contour lines beneath a surface plot; the `meshz` function draws a series of vertical lines under a surface plot and the `waterfall` function draws mesh lines in one direction only. The following slides will illustrate this using the following function:

$z=xe^{-(x2+y2)}$ *[note: x squared and y squared]*

Plots of the surface $z = xe^{-(x^2+y^2)}$ created with the `mesh` function and its variant forms: `meshc`, `meshz`, and `waterfall`. a) `mesh`, b) `meshc`, c) `meshz`, d) `waterfall`.

Figure 5.8–4



Three-dimensional plotting functions. Table 5.8–1

Function	Description
`contour(x,y,z)`	Creates a contour plot.
`mesh(x,y,z)`	Creates a 3D mesh surface plot.
`meshc(x,y,z)`	Same as `mesh` but draws contours under the surface.
`meshz(x,y,z)`	Same as `mesh` but draws vertical reference lines under the surface.
`surf(x,y,z)`	Creates a shaded 3D mesh surface plot.
`surfc(x,y,z)`	Same as `surf` but draws contours under the surface.
`[X,Y] = meshgrid(x,y)`	Creates the matrices `X` and `Y` from the vectors `x` and `y` to define a rectangular grid.
`[X,Y] = meshgrid(x)`	Same as `[X,Y]=meshgrid(x,x)`.
`waterfall(x,y,z)`	Same as `mesh` but draws mesh lines in one direction only.

The following slides contain figures from the chapter's homework problems.

Figure P27

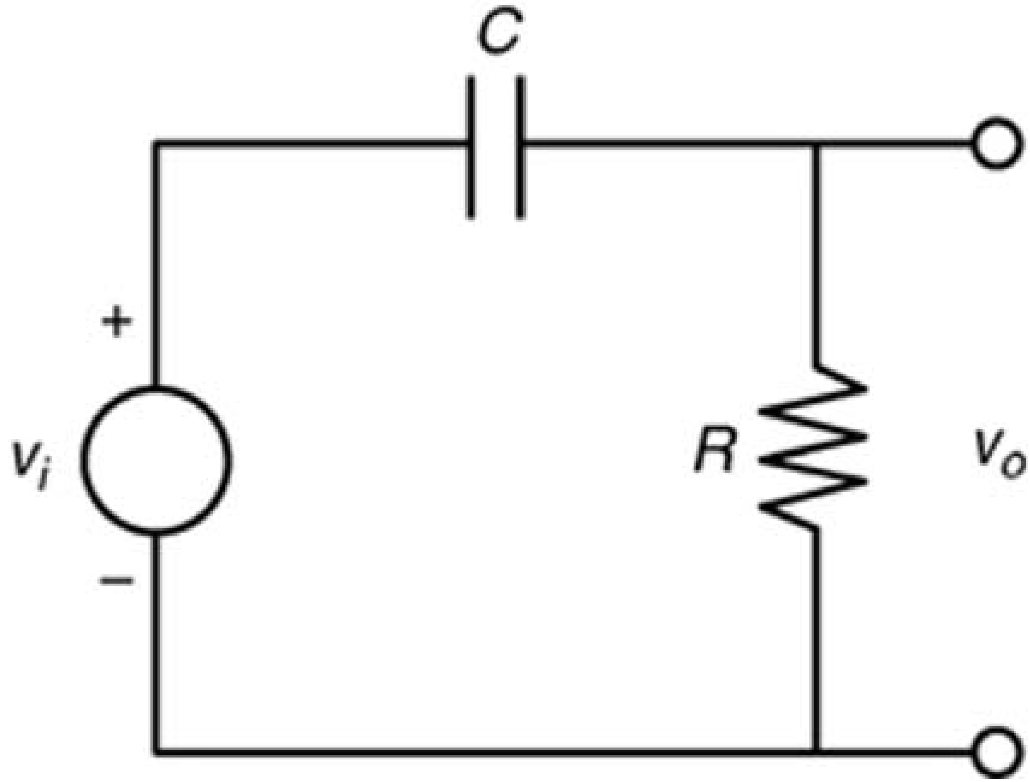


Figure P28

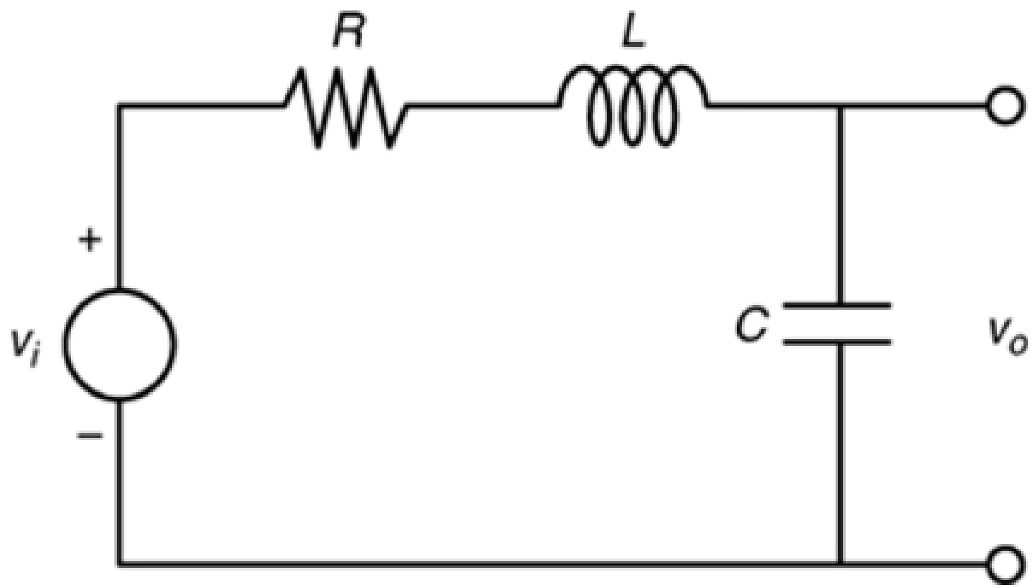


Figure P56