Introduction to MATLAB

Dr./ Ahmed Nagib Elmekawy

Mechanical Engineering department, Alexandria university, Egypt

Spring 2017

Lecture 6

- **Linear Algebric Equations**
- **Numerical Methods for Calculus and Differential Equations**
- **Symbolic Toolbox**
- **Simulink**

Linear Algebric Equations

Matrix notation enables us to represent multiple equations as a single matrix equation. For example, consider the following set:

$$2x_1 + 9x_2 = 5$$

$$3x_1 - 4x_2 = 7$$

This set can be expressed in vector-matrix form as

$$\begin{bmatrix} 2 & 9 \\ 3 & -4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 5 \\ 7 \end{bmatrix}$$

which can be represented in the following compact form

$$\mathbf{Ax} = \mathbf{b}$$

The MATLAB command `inv(A)` computes the inverse of the matrix **A**. The following MATLAB session solves the following equations using MATLAB.

$$2x + 9y = 5$$
$$3x - 4y = 7$$

```
>>A = [2,9;3,-4];b = [5;7]
>>x = inv(A)*b
x =
    2.3714
    0.0286
```

For the equation set $\mathbf{Ax} = \mathbf{b}$,

- if $|\mathbf{A}| = 0$, then there is no unique solution.

- Depending on the values in the vector **b**, there may be no solution at all, or an infinite number of solutions.

A *singular* problem refers to a set of equations having either no unique solution or no solution at all. For example, the set

$$3x - 4y = 5$$

$$6x - 8y = 10$$

is singular and has no unique solution because the second equation is identical to the first equation, multiplied by 2. The graphs of these two equations are identical. All we can say is that the solution must satisfy $y = (3x - 5)/4$, which describes an infinite number of solutions.

If you attempt to solve a singular problem using the `inv` command, MATLAB displays an error message.

The equations

$$6x - 10y = 2$$

$$3x - 4y = 5$$

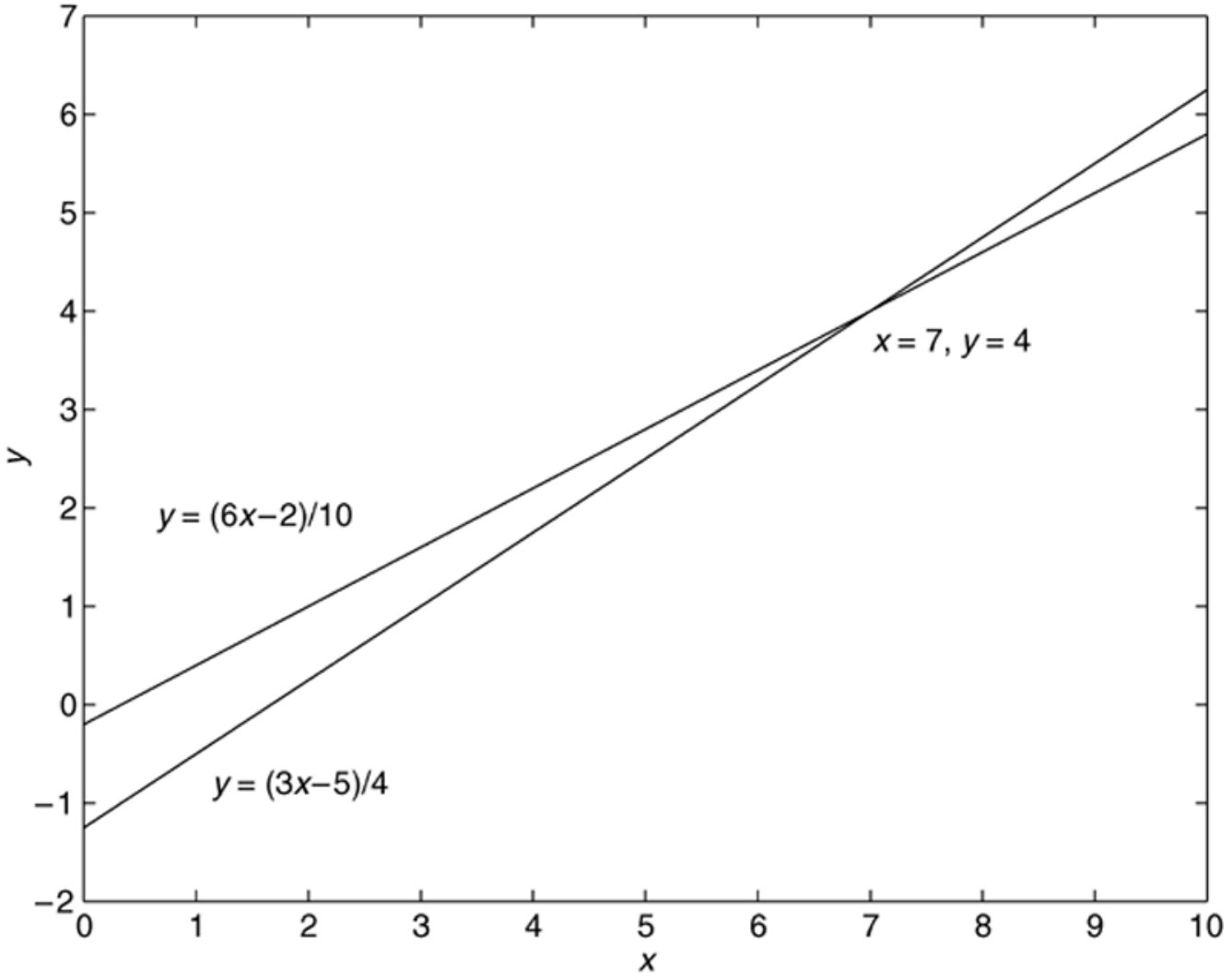
have graphs that intersect at the solution y = 4, x = 7. On the other hand, the set

$$3x - 4y = 5$$

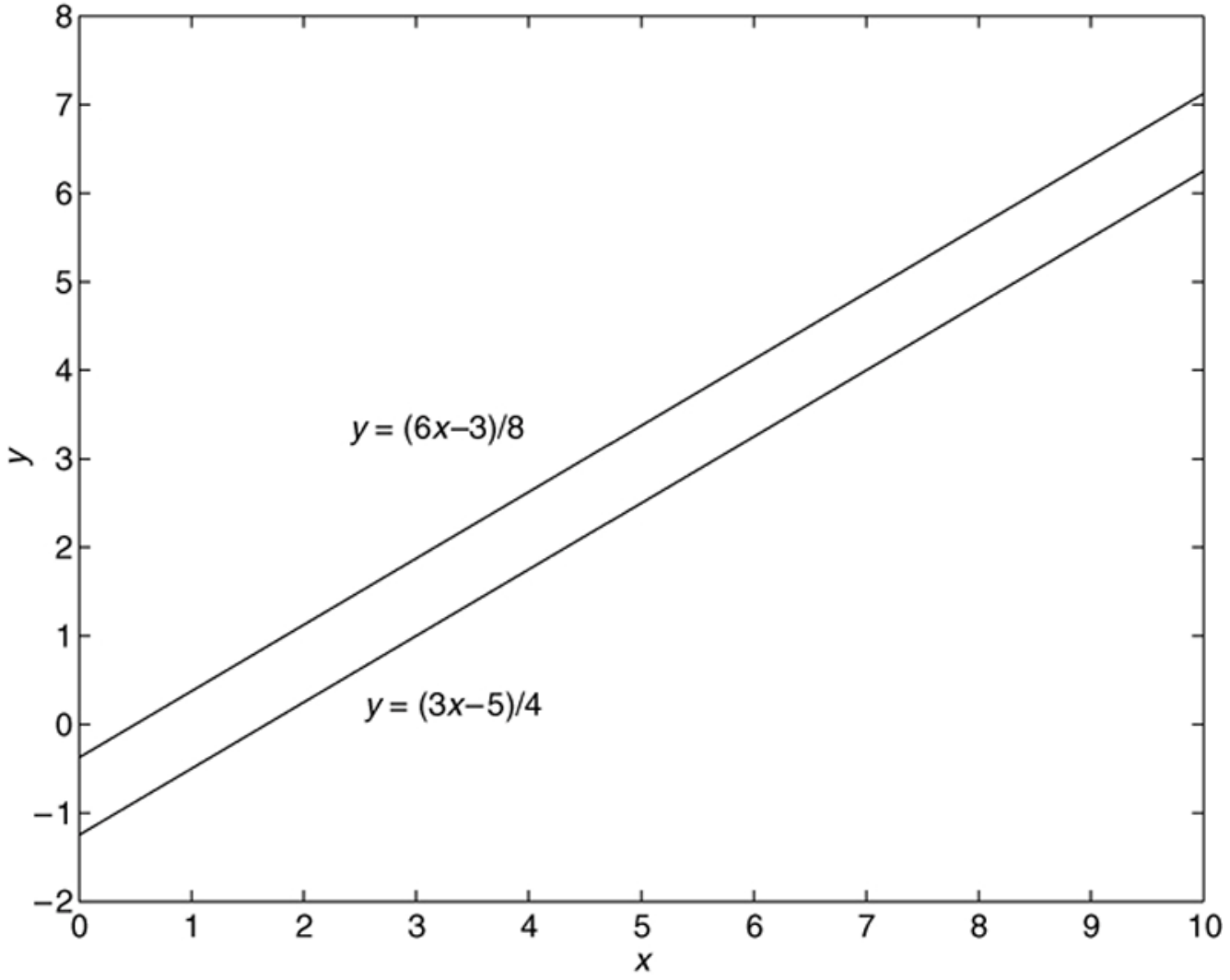
$$6x - 8y = 3$$

is singular but has no solution. The graphs of these two equations are distinct but *parallel* (see the next slide). Because they do not intersect, no solution exists.

The graphs of two equations that intersect at a solution.



Parallel graphs indicate that no solution exists.



Consider the following set of *homogeneous equations* (which means that their right sides are all zero)

$$6x + ay = 0$$

$$2x + 4y = 0$$

where a is a parameter. Multiply the second equation by 3 and subtract the result from the first equation to obtain

$$(a - 12)y = 0$$

The solution is $y = 0$ *only if* $a \neq 12$; if $a = 12$, there is an infinite number of solutions for x and y, where $x = -2y$.

Existence and uniqueness of solutions

The set $\mathbf{Ax} = \mathbf{b}$ with m equations and n unknowns has solutions if and only if

$$\text{rank}[\mathbf{A}] = \text{rank}[\mathbf{A}\ \mathbf{b}] \qquad (1)$$

Let $r = \text{rank}[\mathbf{A}]$.

● If condition (1) is satisfied and if $r = n$, then the solution is unique.

● If condition (1) is satisfied but $r < n$, an infinite number of solutions exists and r unknown variables can be expressed as linear combinations of the other $n - r$ unknown variables, whose values are arbitrary.

Homogeneous case.

The homogeneous set $\mathbf{Ax} = \mathbf{0}$ is a special case in which $\mathbf{b} = \mathbf{0}$.

For this case rank[$\mathbf{A}$] $=$ rank[$\mathbf{A}$ $\mathbf{b}$] always, and thus the set always has the trivial solution $\mathbf{x} = \mathbf{0}$.

A nonzero solution, in which at least one unknown is nonzero, exists if and only if rank[$\mathbf{A}$] $< n$.

If $m < n$, the homogeneous set always has a nonzero solution.

If the number of equations equals the number of unknowns and if $|\mathbf{A}| \neq 0$, then the equation set has a solution and it is unique.

If $|\mathbf{A}| = 0$ or if the number of equations does not equal the number of unknowns, then you must use the methods presented in Sections 8.3 or 8.4.

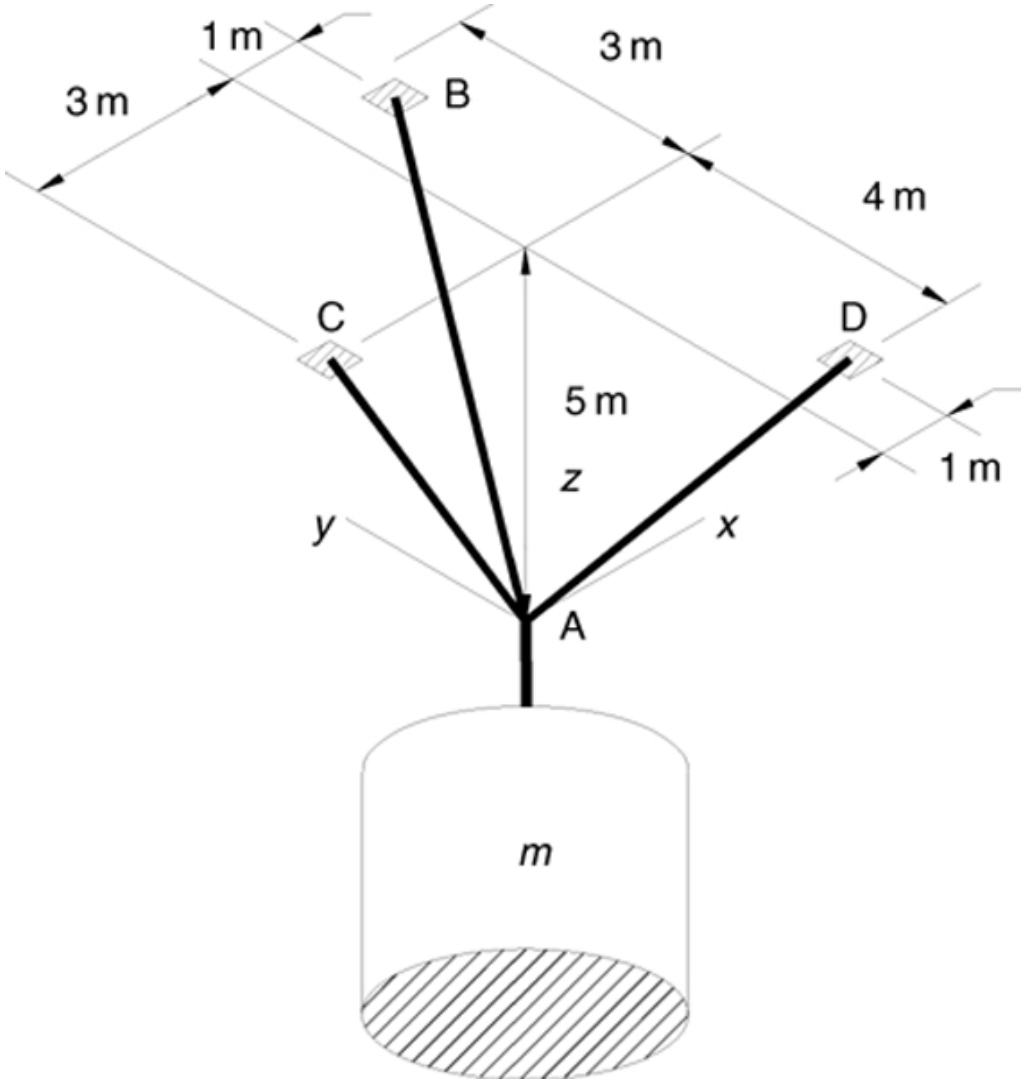
MATLAB provides the *left-division* method for solving the equation set **Ax** = **b**. The left-division method is based on Gauss elimination. (Section 8.2 on page 335) To use the left-division method to solve for **x**, type `x = A\b`. For example,

```
>> A = [6, -10; 3, -4]; b = [2; 5];
>> x = A\b
x =
      7     4
```

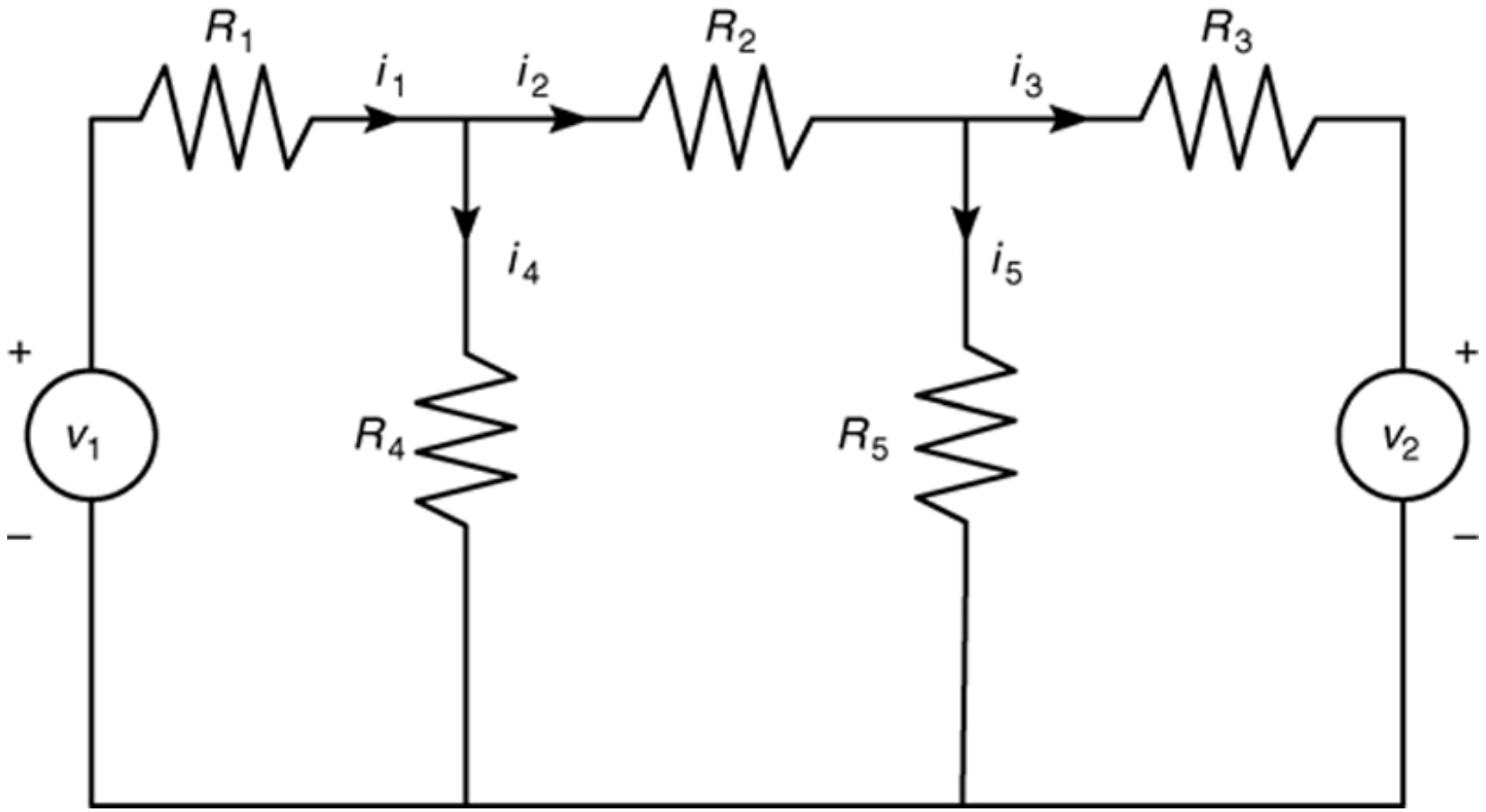
This method also works in some cases where the number of unknowns does not equal the number of equations.

For more examples, see pages 335-341.

Application of linear equations: Calculating cable tension for a **suspended mass. Example 8.2-2. Figure 8.2-1, page 337.**



Application of linear equations: An electrical-resistance network. Example 8.2-3. Figure 8.2-2 on page 338



Solving Linear Equations: Summary

If the number of equations in the set *equals* the number of unknown variables, the matrix **A** is square and MATLAB provides two ways of solving the equation set **Ax** = **b**:

1. The matrix inverse method; solve for **x** by typing `x = inv(A)*b`.

2. The matrix left-division method; solve for **x** by typing `x = A\b`.

(continued …)

Solving Linear Equations: Summary (continued)

If **A** is square and if MATLAB does not generate an error message when you use one of these methods, then the set has a unique solution, which is given by the left-division method.

You can always check the solution for `x` by typing `A*x` to see if the result is the same as `b`.

(continued …)

Solving Linear Equations: Summary (continued)

If you receive an error message, the set is underdetermined, and either it does not have a solution or it has more than one solution.

In such a case, if you need more information, you must use the following procedures.

(continued …)

Solving Linear Equations: Summary (continued)

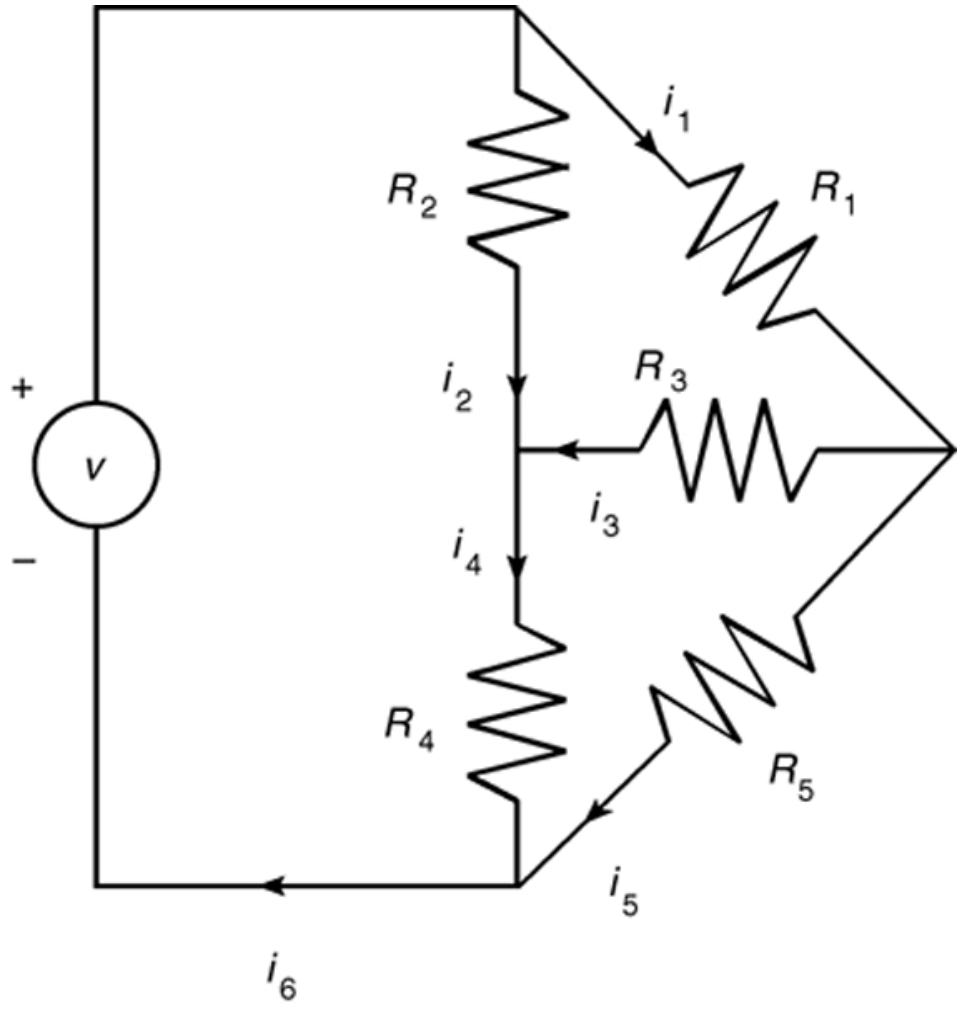
For underdetermined and overdetermined sets, MATLAB provides three ways of dealing with the equation set **Ax = b.** (Note that the matrix inverse method will never work with such sets.)

The matrix left-division method; solve for **x** by typing `x = A\b`.

(continued …)

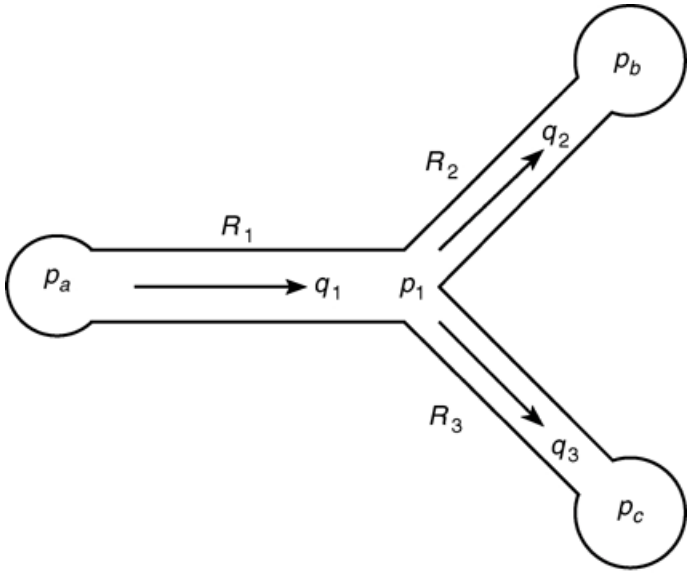
The following slides contain figures from the chapter's homework problems.

Figure P4, page 358

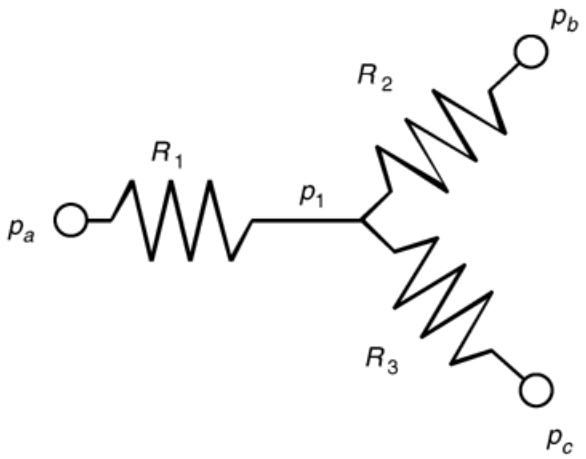


8-43

Figure P6, page 360



(a)



(b)

Figure P7, page 361

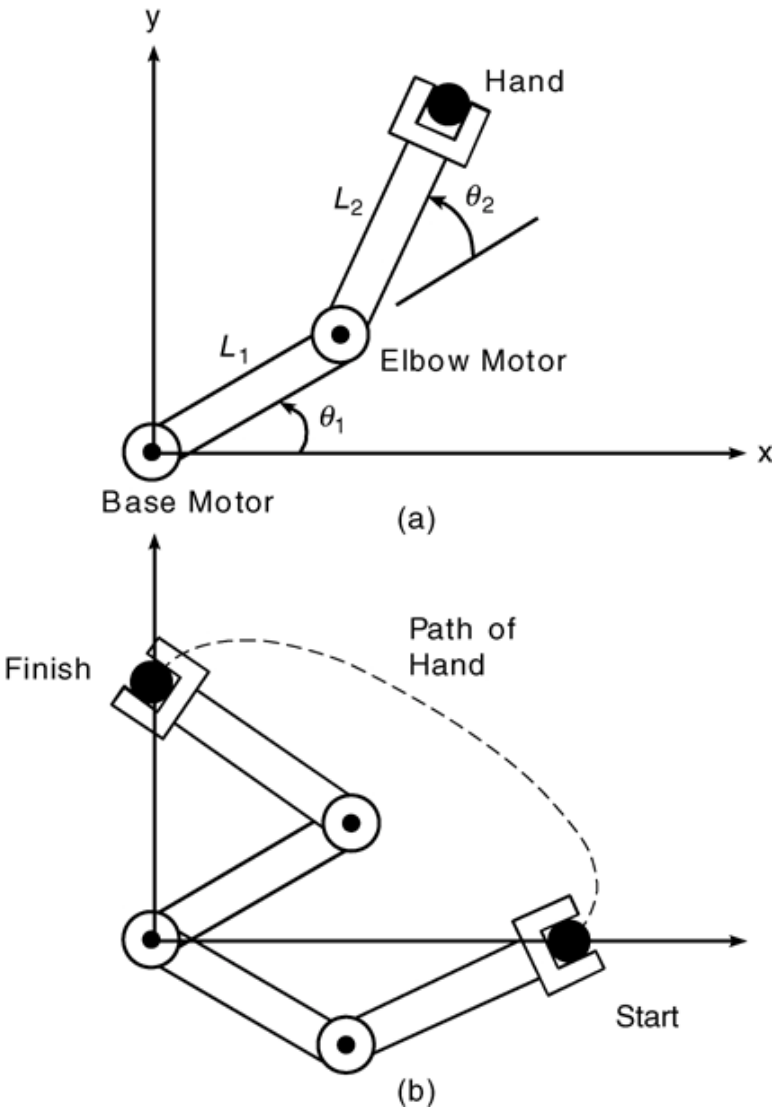
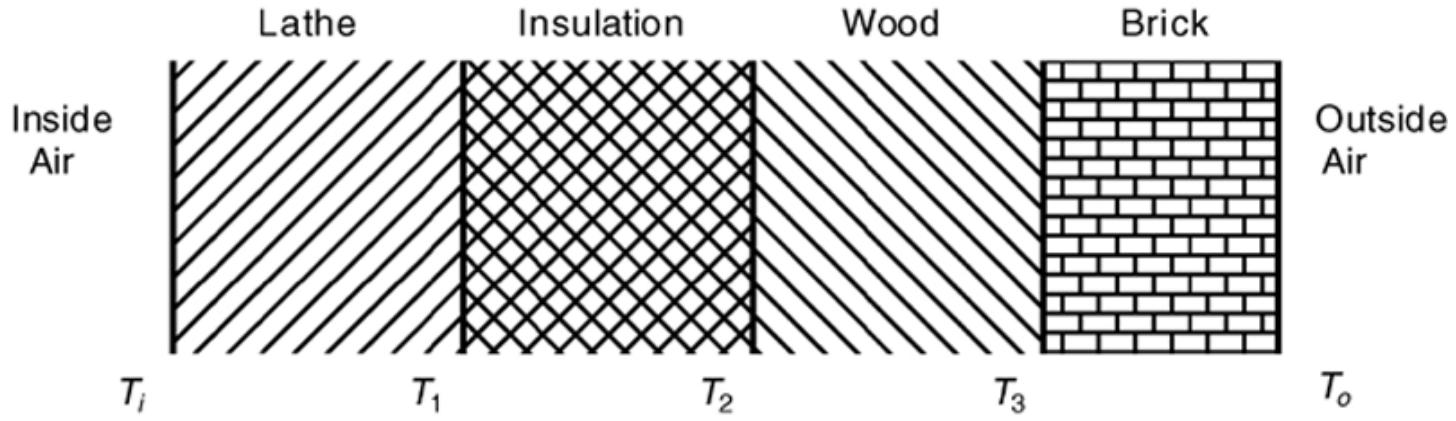
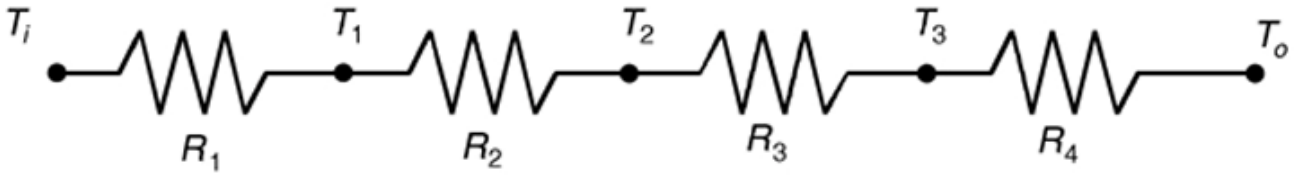


Figure P8, page 362



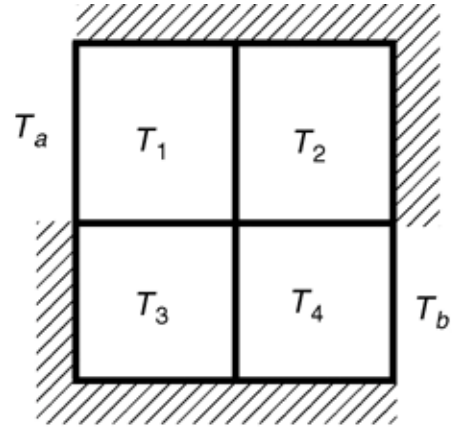
(a)



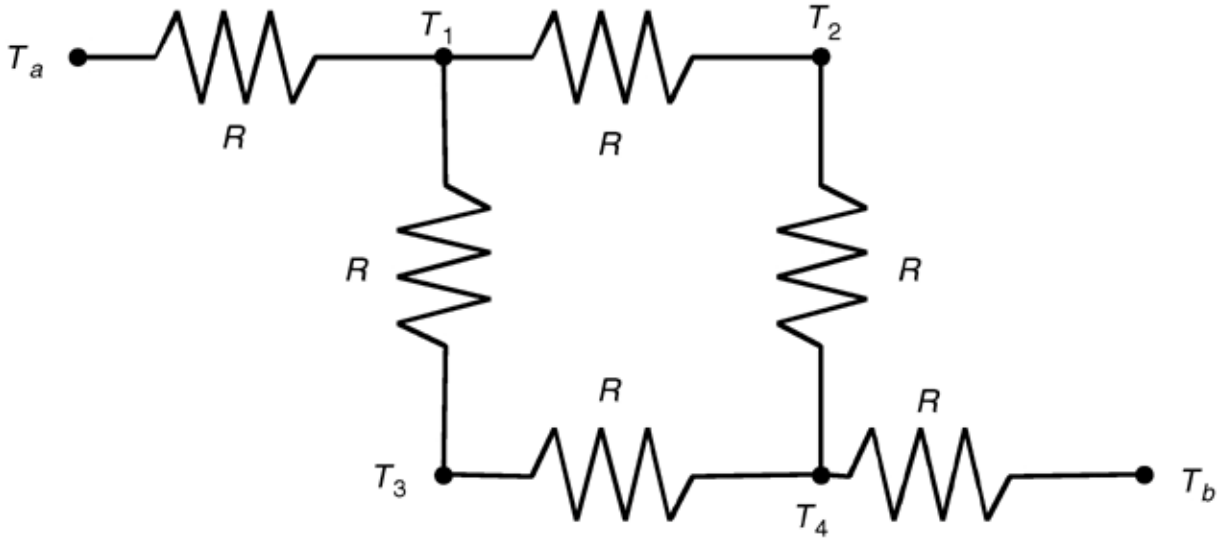
(b)

8-46

Figure P9, page 363



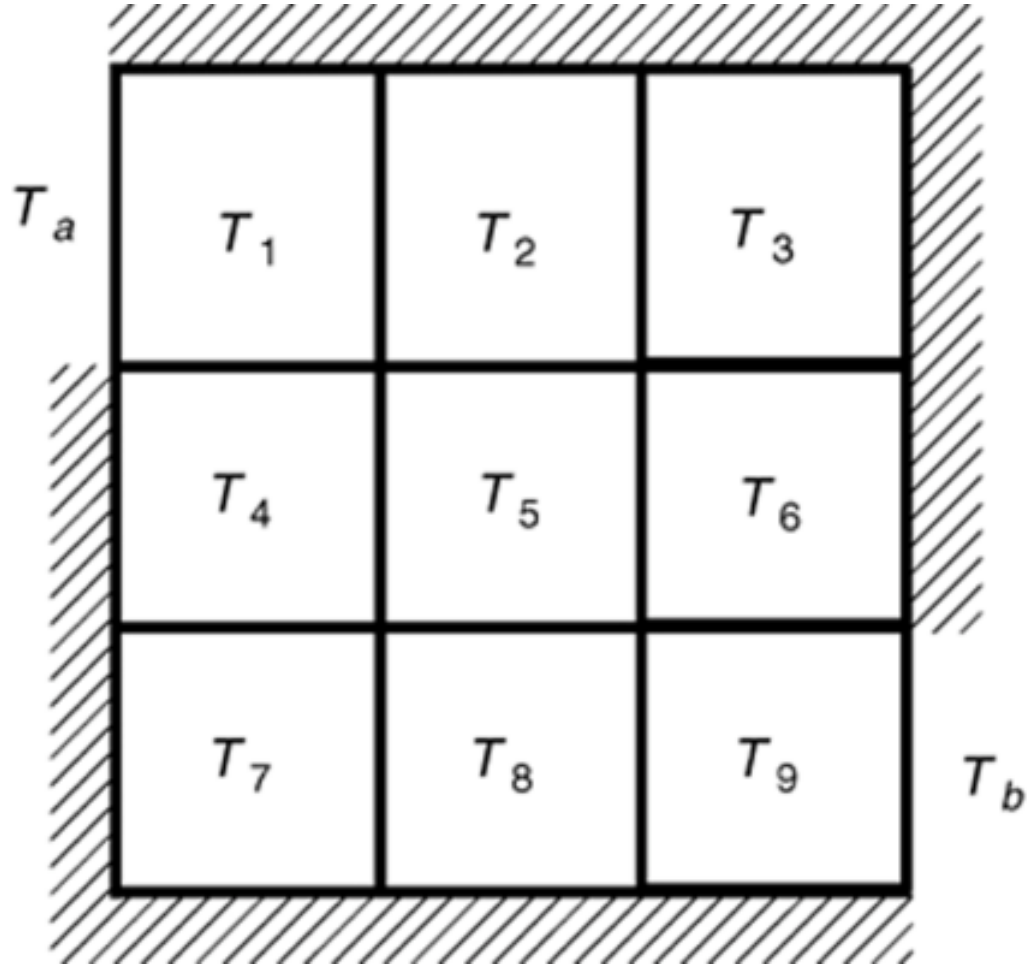
(a)



(b)

8-47

Figure P10, page 365



8-48

Numerical Methods for Calculus and Differential Equations

The integral of *f(x)* interpreted as the area *A* under the curve of *f* (*x*) from *x* = *a* to *x* = *b*.

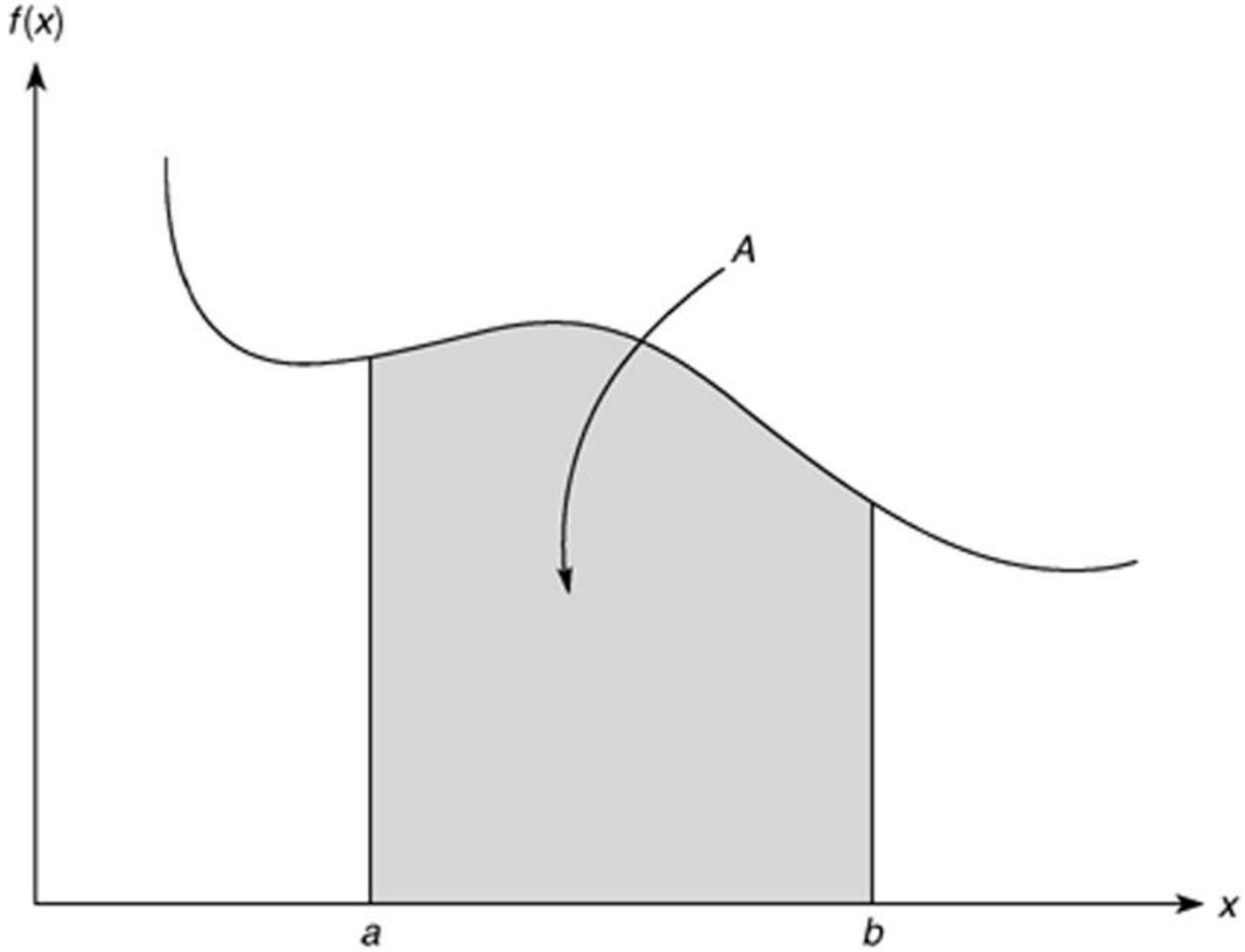
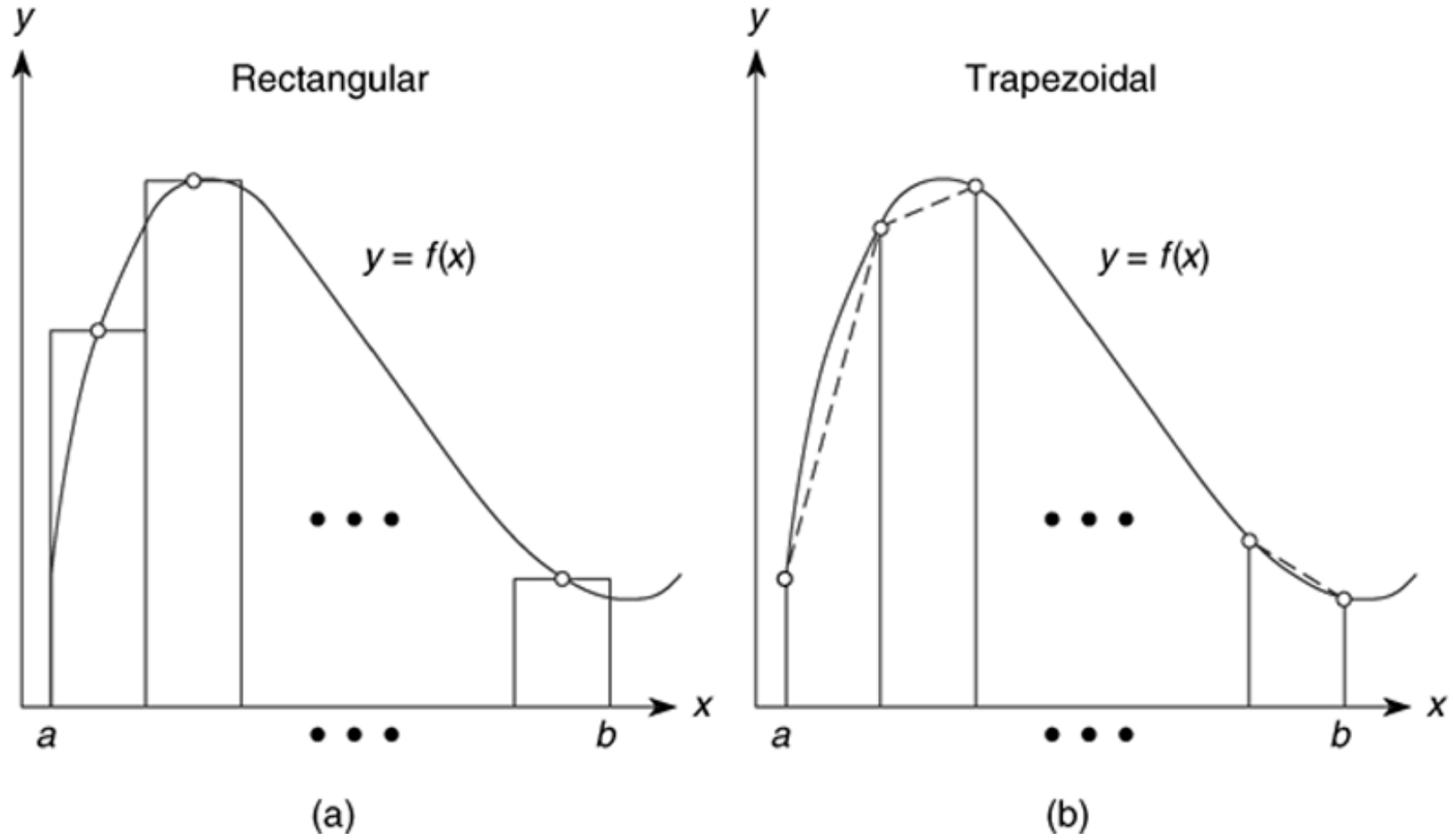


Illustration of (a) rectangular and (b) trapezoidal numerical integration. Figure 9.1–1, page 370.



Numerical integration functions. Table 9.1–1, page 371.

Command	Description
`quad(fun,a,b)`	Uses an adaptive Simpson's rule to compute the integral of the function whose handle is `fun`, with `a` as the lower integration limit and `b` as the upper limit. The function `fun` must accept a vector argument.
`quadl(fun,a,b)`	Uses Lobatto quadrature to compute the integral of the function `fun`. The rest of the syntax is identical to `quad`.

(continued)

Table 9.1-1 (continued)

`trapz(x,y)`	Uses trapezoidal integration to compute the integral of `y` with respect to `x`, where the array `y` contains the function values at the points contained in the array `x`.

Although the `quad` and `quadl` functions are more accurate than `trapz`, they are restricted to computing the integrals of functions and cannot be used when the integrand is specified by a set of points. For such cases, use the `trapz` function.

Using the `trapz` function. Compute the integral

$$\int_0^{\pi} \sin x \, dx$$

First use 10 panels with equal widths of $\pi/10$. The script file is

```
x = linspace(0,pi,10);
y = sin(x);
trapz(x,y)
```

The answer is 1.9797, which gives a relative error of 100(2 - 1.9797)/2) = 1%. For more about the `trapz` function, see pages 371-373.

MATLAB function `quad` implements an adaptive version of Simpson's rule, while the `quadl` function is based on an adaptive Lobatto integration algorithm.

To compute the integral of sin(*x*) from 0 to π, type

```
>>A = quad(@sin,0,pi)
```

The answer given by MATLAB is 2.0000, which is correct. We use `quadl` the same way; namely,

```
>> A = quadl(@sin,0,pi).
```

To integrate $\cos(x^2)$ from 0 to $\sqrt{(2\pi)}$, create the function:

```
function c2 = cossq(x)
% cosine squared function.
c2 = cos(x.^2);
```

Note that we must use array exponentiation.

The `quad` function is called as follows:

```
>>quad(@cossq,0,sqrt(2*pi))
```

The result is 0.6119.

More? See pages 374-375.

Polynomial Integration

`q = polyint(p,C)` returns a polynomial `q` representing the integral of polynomial `p` with a user-specified scalar constant of integration `C`. See page 375.

Double Integrals

`A = dblquad(fun, a, b, c, d)` computes the integral of *f(x,y)* from *x* = *a* to *b*, and *y* = *c* to *d*. Here is an example using an anonymous function to represent $f(x,y) = xy^2$.

```
>>fun = @(x,y)x.*y^2;
>>A = dblquad(fun,1,3,0,1)
```

The answer is *A* = 1.3333. For more, see pages 376 to 377.

Triple Integrals

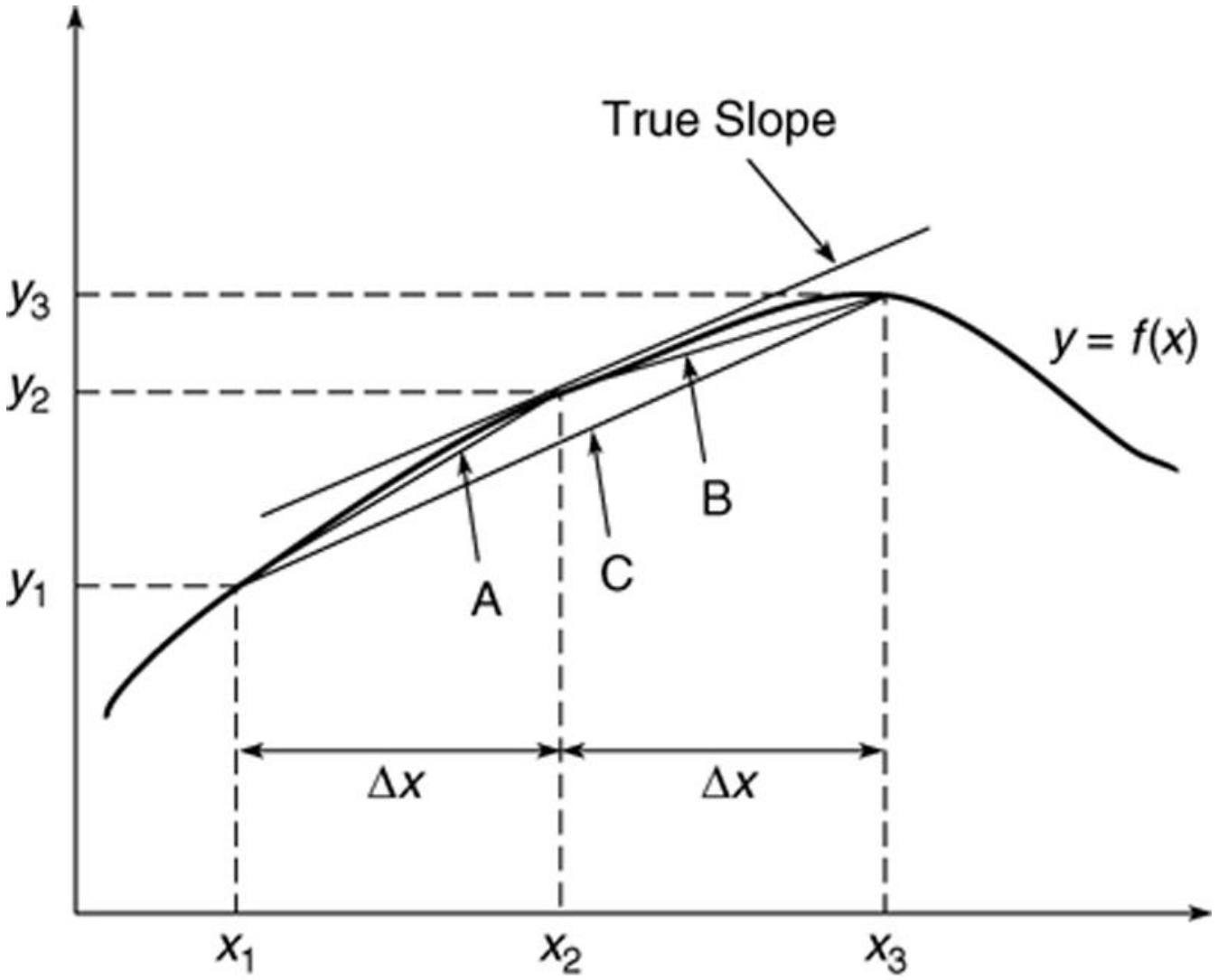
`A = triplequad(fun, a, b, c, d, e, f)` computes the triple integral of $f(x,y,z)$ from $x = a$ to b, $y = c$ to d, and $z = e$ to f. Here is an example using an anonymous function to represent $f(x,y,z) = (xy - y^2)/z$.

```
>>fun = @(x,y,z)(x*y -y^2)/z;
>>A = triplequad(fun,1,3,0,2,1,2)
```

The answer is $A = 1.8484$. Note that the function must accept a vector x, but scalar y and z. See page 377.

9-11

Numerical differentiation: Illustration of three methods for estimating the derivative *dy*/*dx*. Figure 9.2–1, page 378.



9-12

MATLAB provides the `diff` function to use for computing derivative estimates.

Its syntax is `d = diff(x)`, where `x` is a vector of values, and the result is a vector `d` containing the differences between adjacent elements in `x`.

That is, if `x` has n elements, `d` will have $n - 1$ elements, where

$$d = [x(2) - x(1), x(3) - x(2), \ldots, x(n) - x(n-1)].$$

For example, if `x = [5, 7, 12, -20]`, then `diff(x)` returns the vector `[2, 5, -32]`. For more, see pages 377-379 and Table 9.2-1.

Polynomial differentiation functions from Table 9.2–1, page 382. For examples, see page 380.

Command	Description
`b = polyder(p)`	Returns a vector `b` containing the coefficients of the derivative of the polynomial represented by the vector `p`.
`b = polyder(p1,p2)`	Returns a vector `b` containing the coefficients of the polynomial that is the derivative of the product of the polynomials represented by `p1` and `p2`.
`[num, den] = polyder(p2,p1)`	Returns the vectors `num` and `den` containing the coefficients of the numerator and denominator polynomials of the derivative of the quotient p_2/p_1, where `p1` and `p2` are polynomials.

Computing gradients

Typing

```
[df_dx, df_dy] = gradient(f,dx,dy)
```

computes the gradient of the function *f(x,y)*, where df_dx and df_dy represent the partial derivatives, and dx, dy represent the spacing.

For an example, see Figure 9.2-2 and the program on pages 381-382.

9-15

Solving First-Order Differential Equations, Section 9.3.

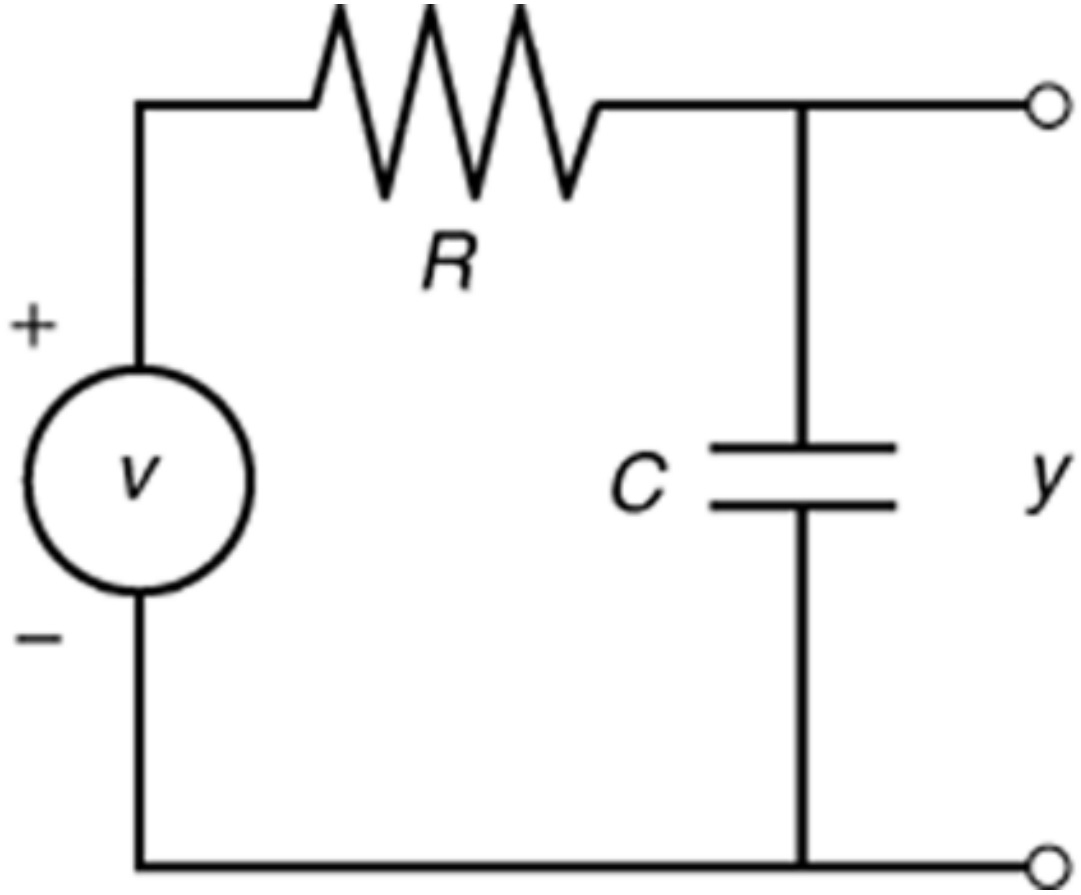
The MATLAB ode solver, `ode45`. To solve the equation *dy/dt = f(t,y)* the syntax is

```
[t,y] = ode45(@ydot,tspan,y0)
```

where `@ydot` is the handle of the function file whose inputs must be *t* and *y*, and whose output must be a column vector representing *dy/dt*; that is, *f(t,y)*. The number of rows in this column vector must equal the order of the equation. The array `tspan` contains the starting and ending values of the independent variable *t*, and optionally any intermediate values. The array `y0` contains the initial values of *y*. If the equation is first order, then `y0` is a scalar.

9-16

Application of an `ode` solver to find the response of an RC circuit . Figure 9.3-1, page 386.



9-17

The circuit model for zero input voltage *v* is

$$dy/dt + 10y = 0$$

First solve this for *dy*/*dt*:

$$dy/dt = -10y$$

Next define the following function file. Note that the order of the input arguments must be *t* and *y*.

```
function ydot = RC_circuit(t,y)
ydot = -10*y;
```

9-18

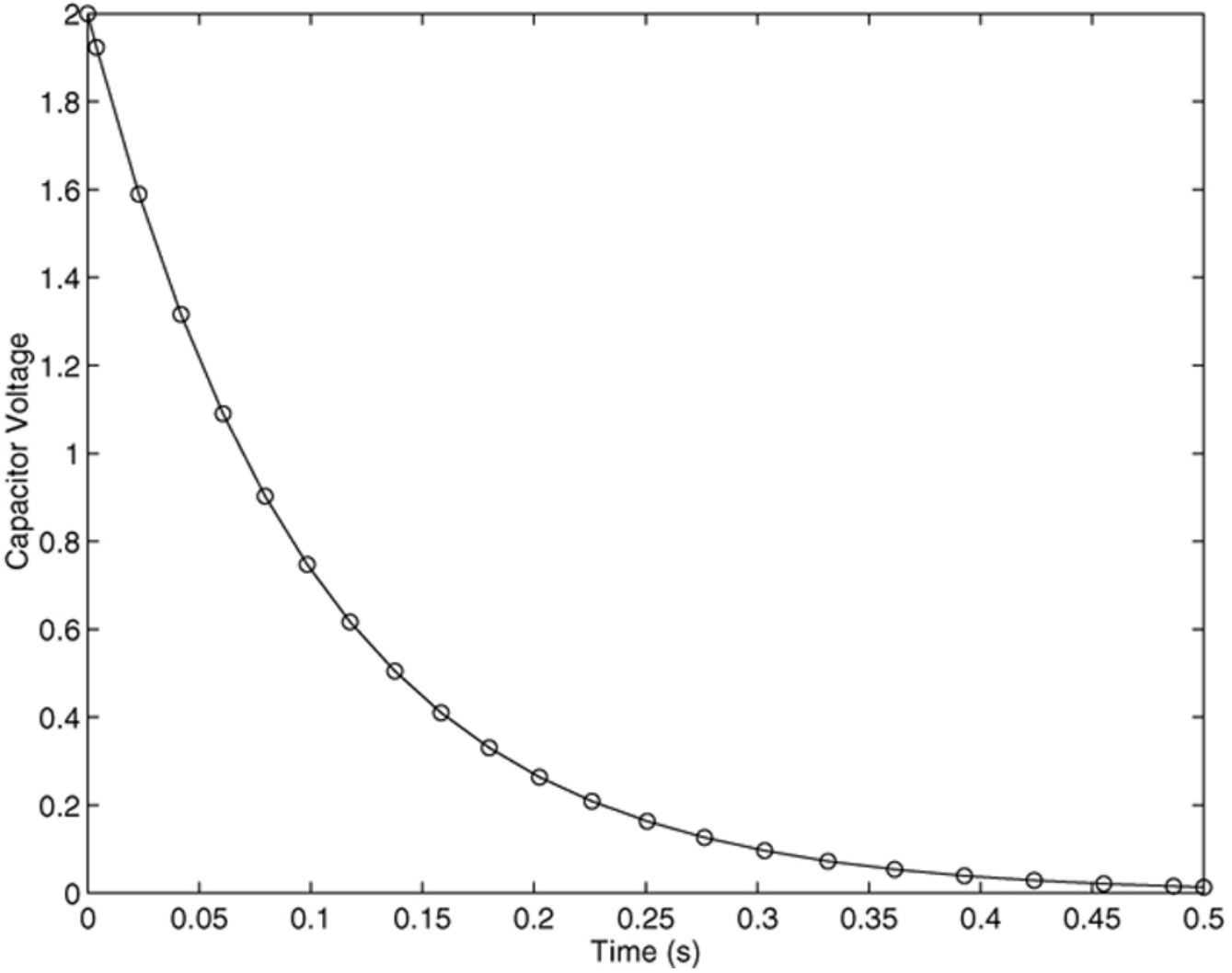
The function is called as follows, and the solution plotted along with the analytical solution `y_true`. The intial condition is $y(0)=2$.

```
[t, y] = ode45(@RC_circuit, [0, 0.5], 2);
y_true = 2*exp(-10*t);
plot(t,y,'o',t,y_true),xlabel('Time(s)'),...
    ylabel('Capacitor Voltage')
```

Note that we need not generate the array `t` to evaluate `y_true`, because `t` is generated by the `ode45` function.

The plot is shown on the next slide and in Figure 9.3-2 on page 387..

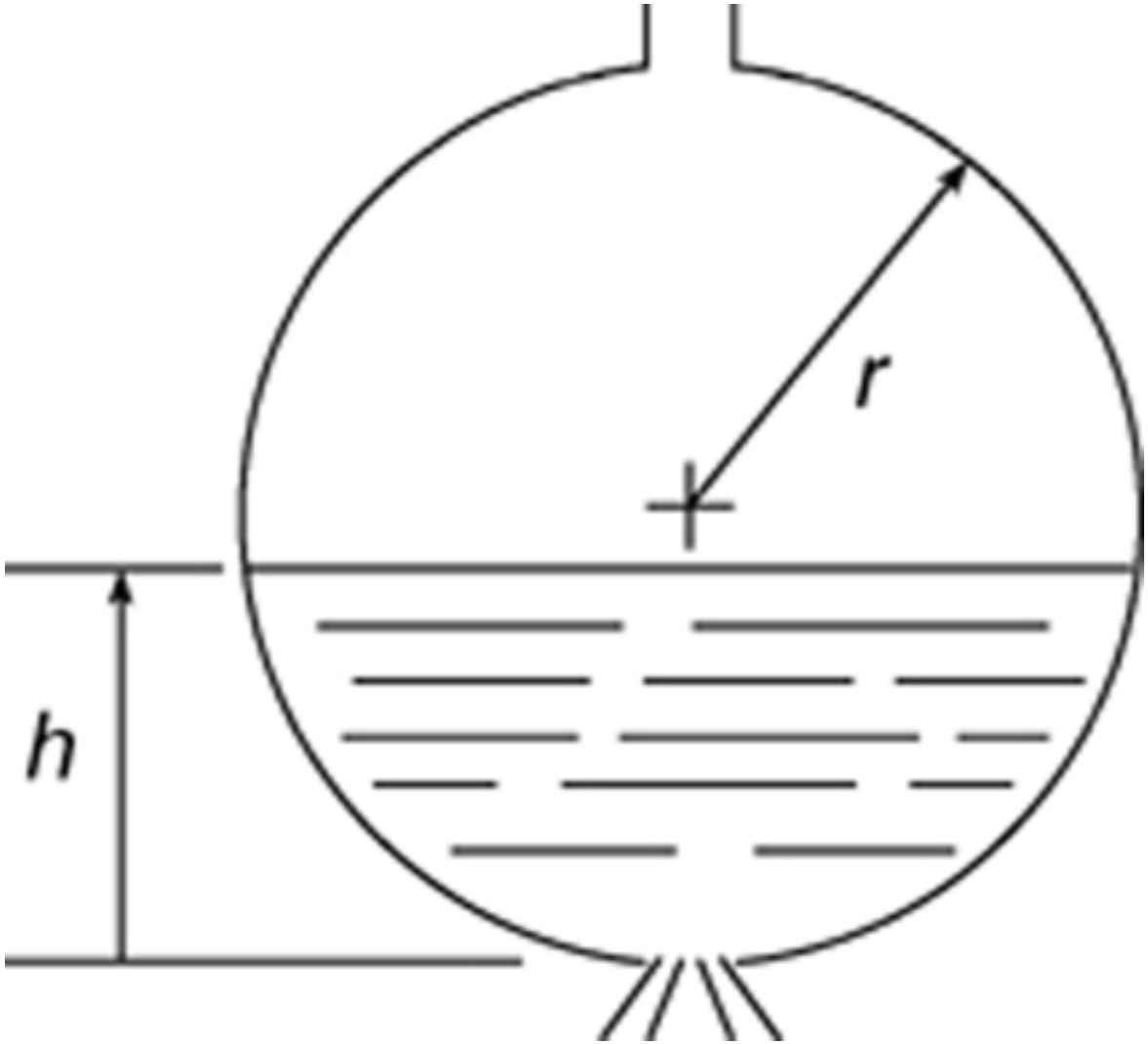
Free response of an RC circuit. Figure 9.3-2



9-20

More? See pages 498-500.

Application example: Draining of a spherical tank. Example 9.3-2 and Figure 9.3-3



9-21

The equation for the height is

$$\frac{dh}{dt} = -\frac{0.0344\sqrt{h}}{10h - h^2}$$

First create the following function file.

```
function hdot = height(t,h)
Hdot = -(0.0344*sqrt(h))/(10*h-h^2);
```

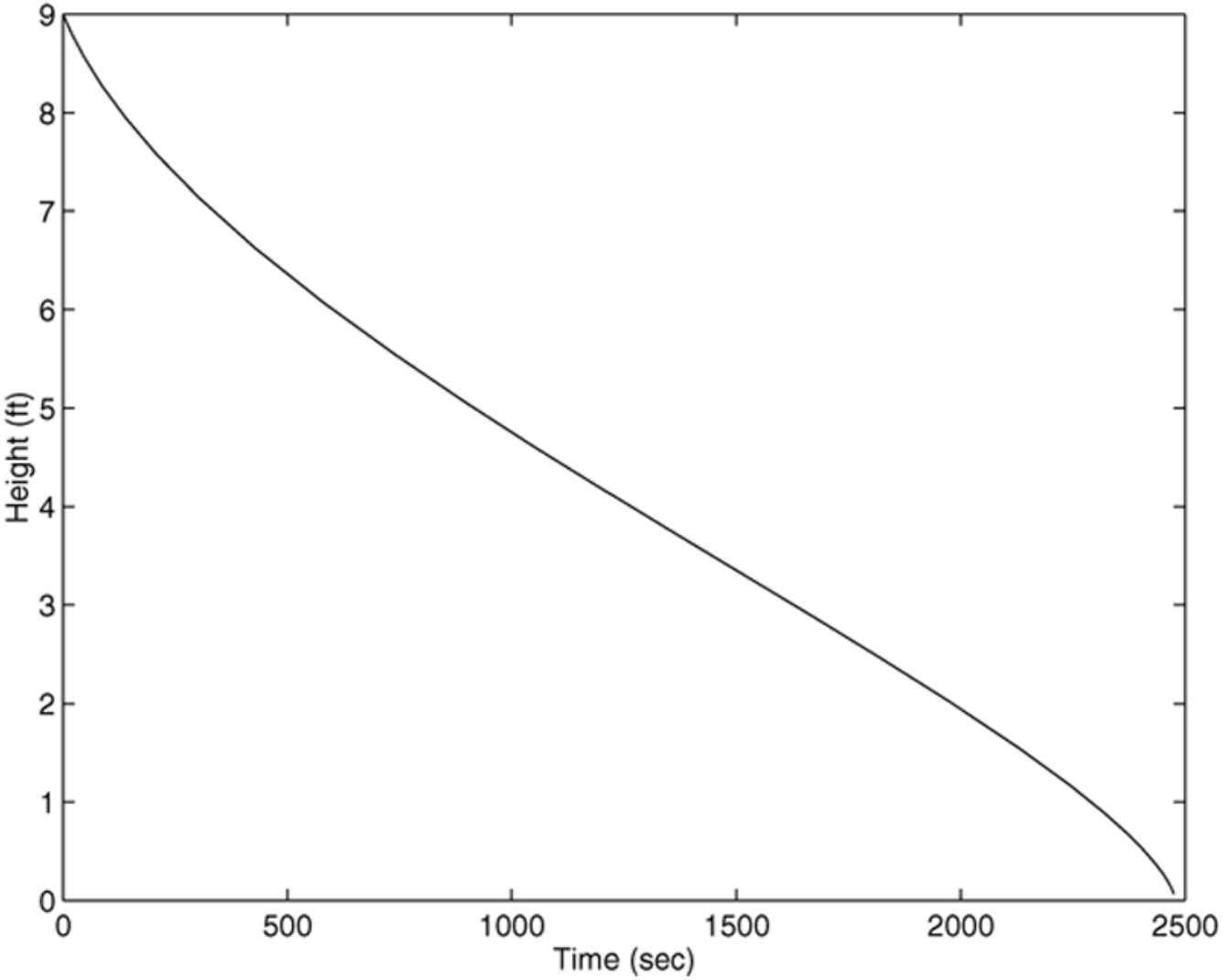
This file is called as follows. The initial height is 9 ft.

```
[t,h] = ode45(@height, [0, 2475], 9);
plot(t,h),xlabel('Time(sec)',ylabel('Height'(ft)')
```

The plot is shown on the next slide.

9-22

Plot of water height in a spherical tank. Figure 9.3-4, page 390.



9-23

Extension to Higher-Order Equations
Section 9.4, page 391

To use the ODE solvers to solve an equation higher than order 2, you must first write the equation as a set of first-order equations.

For example, consider the equation

$$5\ddot{y} + 7\dot{y} + 4y = f(t) \qquad (9.4\text{-}1)$$

Define x_1 = y and x_2 = dy/dt. Then the above equation can be expressed as two equations:

$$\dot{x}_1 = x_2$$

$$\dot{x}_2 = \frac{1}{5} f(t) - \frac{4}{5} x_1 - \frac{7}{5} x_2$$

This form is sometimes called the *Cauchy form* or the *state-variable form.*

9-25

Suppose that $f(t) = \sin t$. Then the required file is

```
function xdot = example_1(t,x)
xdot(1) = x(2);
xdot(2) = (1/5)*(sin(t)-4*x(1)-7*x(2));
xdot = [xdot(1); xdot(2)];
```

Note that:

`xdot(1)` represents dx_1/dt

`xdot(2)` represents dx_2/dt

`x(1)` represents x_1, and `x(2)` represents x_2.

9-26

Suppose we want to solve (9.4-1) for $0 \le t \le 6$ with the initial conditions $x_1(0) = 3$, $x_2(0) = 9$ and $f(t) = \sin t$. Then the initial condition for the *vector* **x** is `[3, 9]`. To use `ode45`, you type

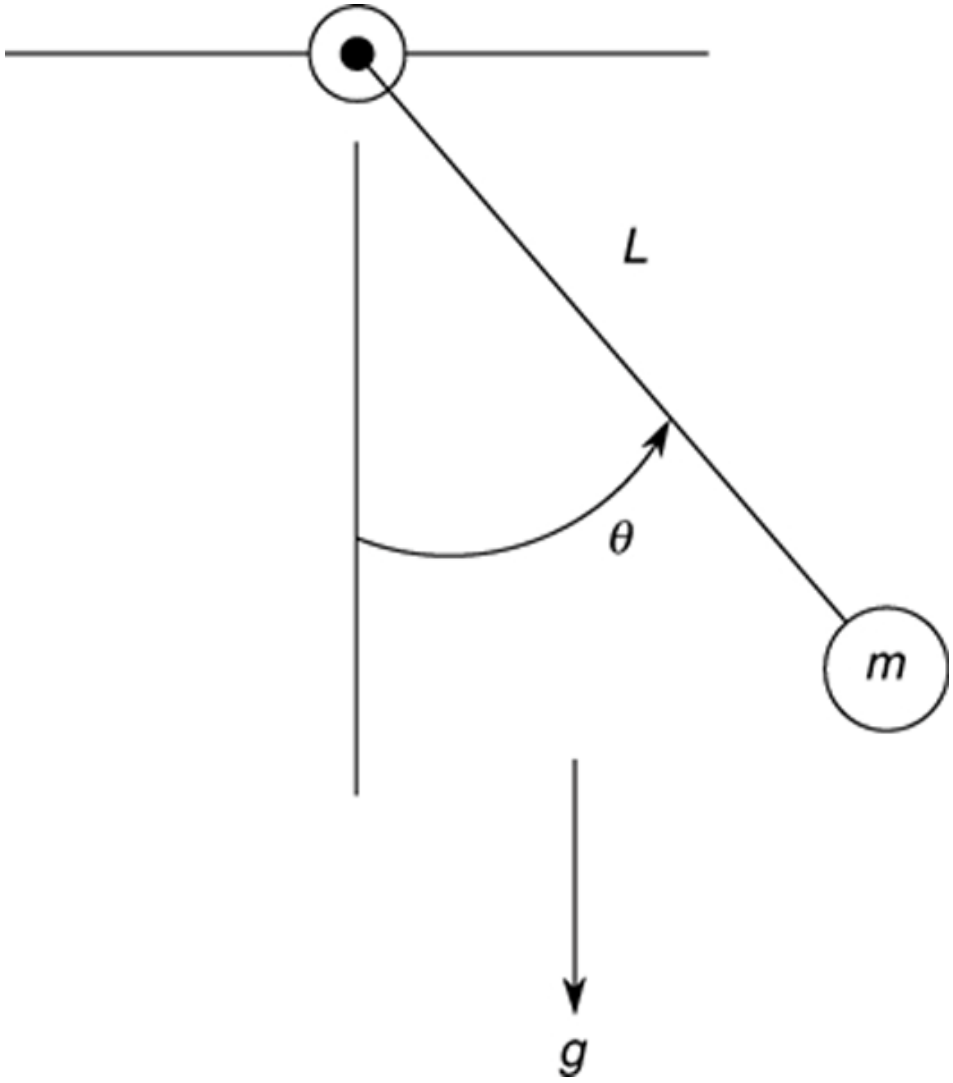
```
[t, x] = ode45(@example_1, [0, 6], [3, 9]);
```

Each row in the vector `x` corresponds to a time returned in the column vector `t`. If you type `plot(t,x)`, you will obtain a plot of both x_1 and x_2 versus t.

Note that `x` is a matrix with two columns; the first column contains the values of x_1 at the various times generated by the solver. The second column contains the values of x_2.

Thus to plot only x_1, type `plot(t,x(:,1))`.

A pendulum example. Example 9.4-1, Figure 9.4-1, page 392



9-29

The equation (9.4-3) is nonlinear and is

$$\ddot{\theta} + \frac{g}{L}\sin\theta = 0$$

It must be rewritten as follows to use `ode45`.

$$\dot{x}_1 = x_2$$

$$\dot{x}_2 = \frac{g}{L}\sin x_1$$

9-30

Create the following function file. Note how we can express `xdot` as a vector in one line, instead of two.

```
function xdot = pendulum(t,x)
g = 9.81; L = 1;
xdot = [x(2); -(g/L)*sin(x(1))];
```

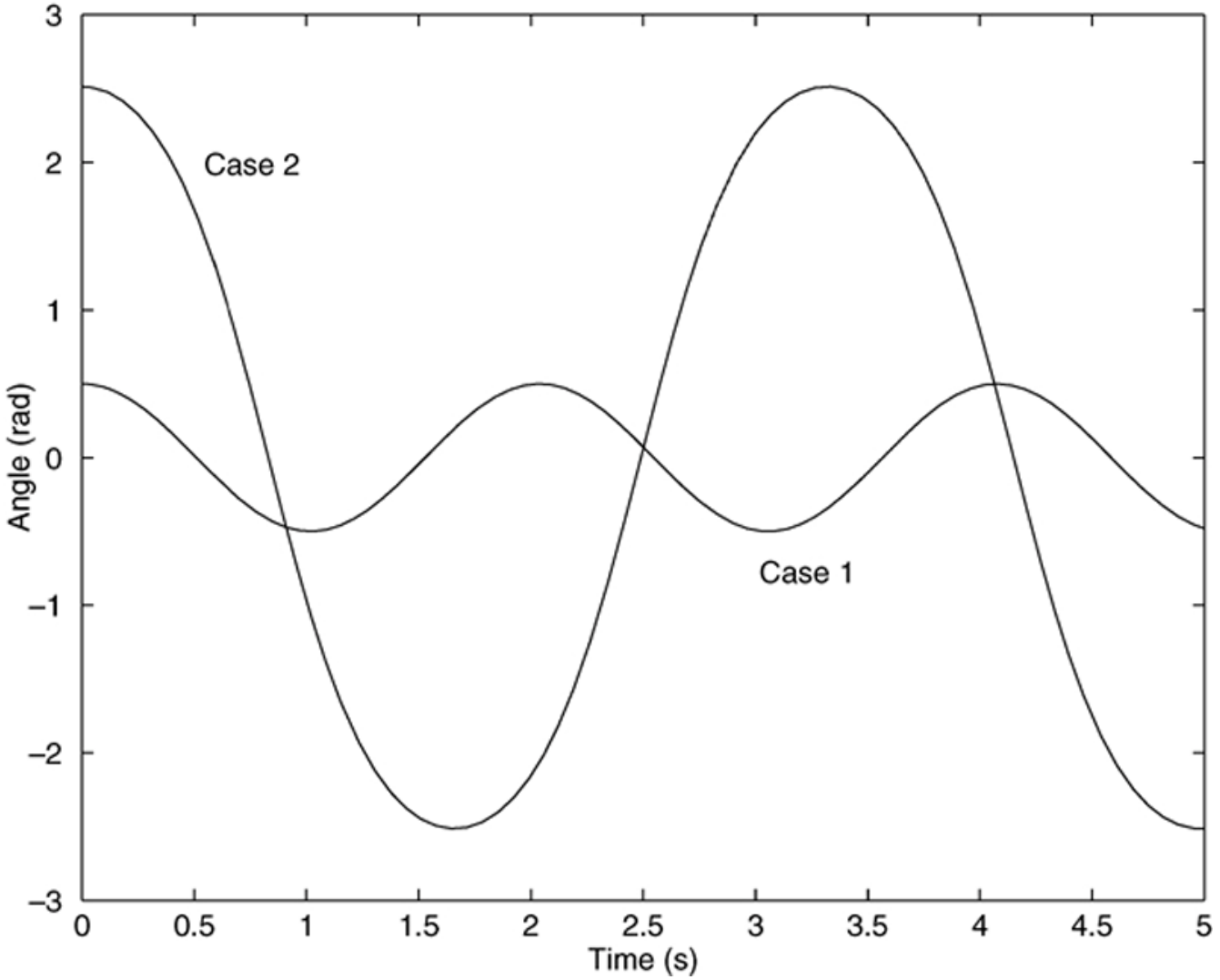
The file is called as follows for the case where $\vartheta(0)=0.5$ rad and the initial angular velocity is zero.

```
[t, x] = ode45(@pendulum, [0, 5], [0.5, 0];
plot(t,x(:,1))
```

The plot is shown by the curve labeled Case 1 on the next slide. See page 392 for how to plot the second case.

9-31

The pendulum angle as a function of time for two starting positions. Figure 9.4-2, page 393.



9-32

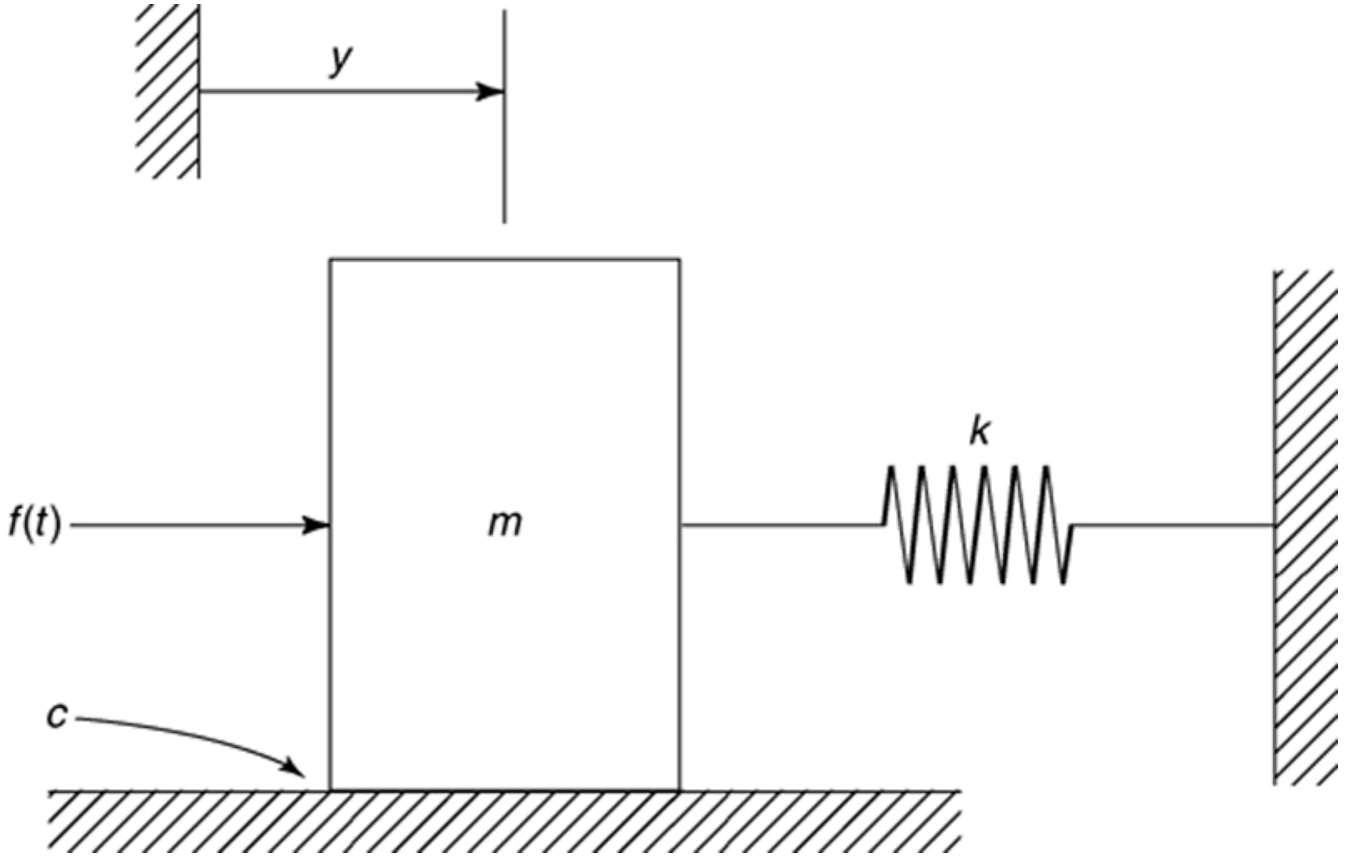
More? See pages 391-394.

The program on pages 393-394 shows how to use a nested function to avoid specifying the values of *g* and *L* within the function file `pendulum`.

9-33

A mass and spring with viscous surface friction. Its equation of motion is

$$m\ddot{y} + c\dot{y} + ky = f(t)$$



9-34

Section 9.5. Special Methods for Linear Differential Equations

The equation of motion can be put into the following state variable form.

$$\dot{x}_1 = x_2$$

$$\dot{x}_2 = \frac{1}{m} f(t) - \frac{k}{m} x_1 - \frac{c}{m} x_2$$

These can be put into matrix form as shown on the next slide.

9-35

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -\frac{k}{m} & -\frac{c}{m} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{1}{m} \end{bmatrix} \mathrm{u(t)}$$

9-36

Create the following function file.

```
function xdot = msd(t,x)
f = 10; m = 1; c = 2; k = 5;
A = [0, 1;-k/m, -c/m];
B = [0; 1/m];
xdot = A*x + b*f;
```

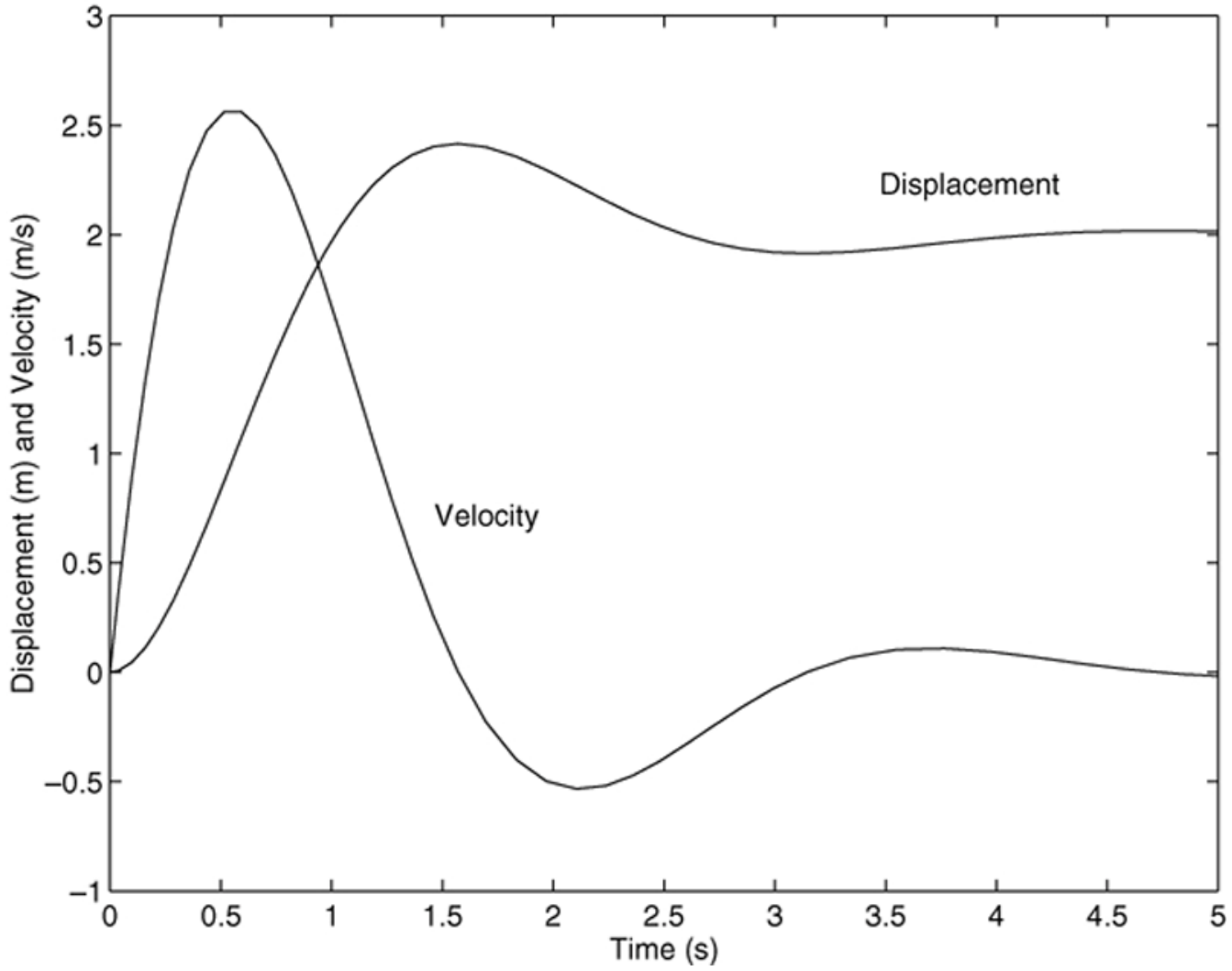
The equations can be solved and the solution plotted follows.

```
[t, x] = ode45(@msd, [0, 5], [0, 0];
plot(t, x(:,1), t, x(:,2))
```

The plot is shown on the next slide.

9-37

Displacement and velocity of the mass as a function of time.
Figure 9.5-1 on page 397.



More? See pages 395-396.

Symbolic Toolbox

In all previous slides, MATLAB always used only numbers. The inputs, outputs, results, etc. were numbers. Sometimes it's better to have more general solutions. One type of general solution is to represent a number by a symbol that represents all numbers.

Symbolic math

- Can give concise statements of problem solutions
- Provides exact solutions with no numerical errors
- Provides exact solutions when numerical solutions aren't possible or feasible
- Can lead to more understanding of a problem and its solution

Consider the quadratic equation $ax^2 + bx + c = 0$ and its solution

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}, \qquad a \neq 0$$

- Concise – one equation gives us all solutions for the infinite number of values of *a, b,* and *c*
- Exact solution - for *a=9, b=0, c=-1*, exact solution is *x=±1/3,* numerical solution is *x=0.333333333333333*

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}, \qquad a \neq 0$$

- Good even if numerical solutions won't work
  - b=10^{200}, a = 1, c = 1 fails because b^2 out of range
- Symbolic solution shows that the equation always has exactly two roots and that b^2 must be at least $4ac$ for the roots to be real
  - These are harder to learn from purely numerical solutions

Symbolic Toolbox in MATLAB performs symbolic math

- Student version of MATLAB comes with Symbolic Toolbox
- Full version of MATLAB doesn't have Symbolic Math Toolbox (must buy it)
- To find out if you have toolbox, type `ver` command and look for “Symbolic Math Toolbox”. If not there, you don’t have it. If there, will also show version and release numbers

A *symbolic object* is made of variables and numbers that MATLAB evaluates symbolically, not numerically

- Can be variables or numbers

A *symbolic expression* is a mathematical expression with at least one symbolic object

- Even if a symbolic expression looks like a numeric expression, MATLAB can tell the difference and evaluates it symbolically

Make a symbolic object with the command

```
object_name = sym('string')
```

`'string'` is the name of the symbolic object and can be

- One or more letters (no spaces)
- A combination of letters and digits that starts with a letter
- A number

`object_name = sym('string')`

It is common (though not required) to make `object_name` be the same as `'string'`

```
>> a=sym('a')
a =
a
>> bb=sym('bb')
bb =
bb
>> x=sym('x');
>>
```

Create a symbolic object a and assign it to a.

The display of a symbolic object is not indented.

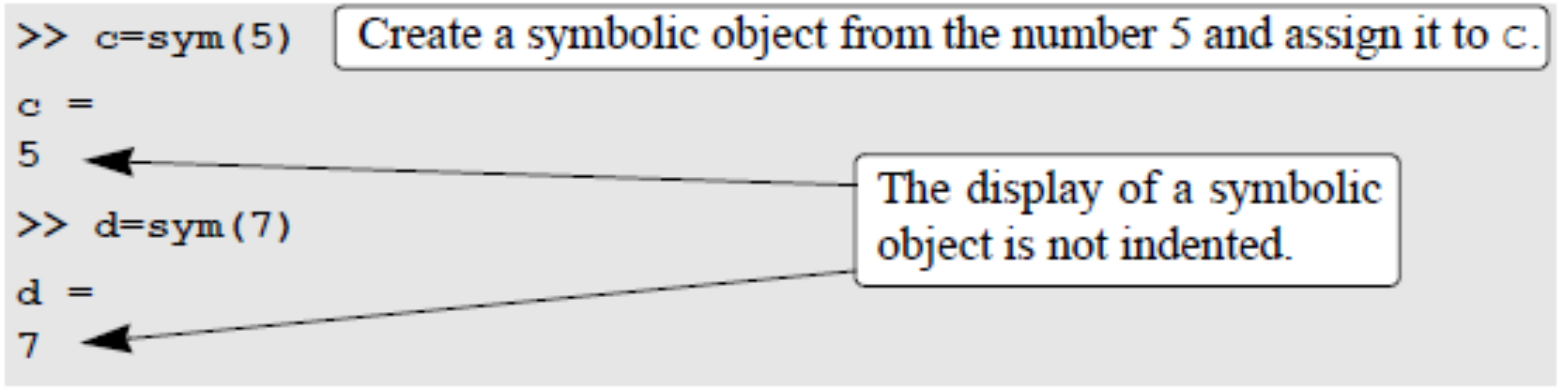
The symbolic variable x is created but not displayed, since a semicolon is typed at the end of the command.

The name of the symbolic object can be different from the name of the variable. For example:

```
>> g=sym('gamma')
g =
gamma
```

The symbolic object is gamma, and the name of the object is g.

When defining a symbolic number, don't need to put the object in quotes



```
syms variable_name1 variable_name2 variable_name3
```

- Creates multiple symbolic objects simultaneously
- Objects have same name as variable names

```
>> syms y z d
>> y
y =
y
```

The variables created by the `syms` command are not displayed automatically. Typing the name of the variable shows that the variable was created,

To create a symbolic expression use

`expression_name = mathematical expression`

A few examples are:

```
>> syms a b c x y
>> f=a*x^2+b*x + c
f =
a*x^2 + b*x + c
```

Define a, b, c, x, and y as symbolic variables.

Create the symbolic expression $ax^2 + bx + c$ and assign it to f.

The display of the symbolic expression is not indented.

When a symbolic expression, which includes mathematical operations that can be executed (addition, subtraction, multiplication, and division), is entered, MATLAB executes the operations as the expression is created. For example:

```
>> g=2*a/3+4*a/7-6.5*x+x/3+4*5/3-1.5
```

$\frac{2a}{3} + \frac{4a}{7} - 6.5x + \frac{x}{3} + 4 \cdot \frac{5}{3} - 1.5$ is entered.

```
g =
(26*a)/21 - (37*x)/6 + 31/6
```

$\frac{26a}{21} - \frac{37x}{6} + \frac{31}{6}$ is displayed.

Notice that all the calculations are carried out exactly with no numerical approximation. In the last example, $\frac{2a}{3}$ and $\frac{4a}{7}$ were added by MATLAB to give $\frac{26a}{21}$, and $-6.5x + \frac{x}{3}$ was added to $\frac{37x}{6}$. The operations with the terms that contain only numbers in the symbolic expression are carried out exactly. In the last example, $4 \cdot \frac{5}{3} + 1.5$ is replaced by $\frac{31}{6}$.

The difference between exact and approximate calculations is demonstrated in the following example, where the same mathematical operations are carried out—once with symbolic variables and once with numerical variables.

```
>> a=sym(3); b=sym(5);
>> e=b/a+sqrt(2)
e =
2^(1/2) + 5/3
>> c=3; d=5;
>> f=d/c+sqrt(2)
f =
    3.0809
```

Define a and b as symbolic 3 and 5, respectively.

Create an expression that includes a and b.

An exact value of e is displayed as a symbolic object (the display is not indented).

Define c and d as numerical 3 and 5, respectively.

Create an expression that includes c and d.

An approximated value of f is displayed as a number (the display is indented).

An expression that is created can include both symbolic objects and numerical variables. However, if an expression includes a symbolic object (or several), all the mathematical operations will be carried out exactly. For example, if *c* is replaced by *a* in the last expression, the result is exact, as it was in the first example.

```
>> g=d/a+sqrt(2)
g =
2^(1/2) + 5/3
```

Additional facts about symbolic expressions and symbolic objects:

- Symbolic expressions can include numerical variables which have been obtained from the execution of numerical expressions. When these variables are inserted in symbolic expressions their exact value is used, even if the variable was displayed before with an approximated value. For example:

```
>> h=10/3
```

h is defined to be 10/3 (a numerical variable).

```
h =
    3.3333
```

An approximated value of h (numerical variable) is displayed.

```
>> k=sym(5); m=sym(7);
```

Define k and m as symbolic 5 and 7, respectively.

```
>> p=k/m+h
```

h, k, and m are used in an expression.

```
p =
85/21
```

The exact value of h is used in the determination of p.
An exact value of p (symbolic object) is displayed.

Use command `double(S)` to convert a symbolic object or expression S to numerical form

```
>> pN=double(p)
pN =
    4.0476
>> y=sym(10)*cos(5*pi/6)
y =
-5*3^(1/2)
>> yN=double(y)
yN =
   -8.6603
```

p is converted to numerical form (assigned to pN).

Create a symbolic expression y.

Exact value of y is displayed.

y is converted to numerical form (assigned to yN).

- A symbolic object that is created can also be a symbolic expression written in terms of variables that have not been first created as symbolic objects. For example, the quadratic expression $ax^2 + bx + c$ can be created as a symbolic object named `f` by using the `sym` command:

```
>> f=sym('a*x^2+b*x+c')
f =
a*x^2 + b*x +c
```

It is important to understand that in this case, the variables a, b, c, and x included in the object do not exist individually as independent symbolic objects (the whole expression is one object). This means that it is impossible to perform symbolic math operations associated with the individual variables in the object. For example, it will not be possible to differentiate `f` with respect to x. This is different than the way in which the quadratic expression was created in the first example in Section *11.1.2*, where the individual variables are first created as symbolic objects and then used in the quadratic expression.

- Existing symbolic expressions can be used to create new symbolic expressions. This is done by simply using the name of the existing expression in the new expression. For example:

```
>> syms x y
```

Define x, and y as symbolic variables.

```
>> SA=x+y, SB=x-y
```

Create two symbolic expressions SA and SB.

```
SA =
x+y
SB =
x-y
```

$$SA = x + y$$

$$SB = x - y$$

```
>> F=SA^2/SB^3+x^2
```

Create a new symbolic expression F using SA and SB.

```
F =
(x+y)^2/(x-y)^3+x^2
```

$$F = (SA^2)/(SB^3) + x^2 = \frac{(x+y)^2}{(x-y)^3} + x^2$$

11.1.3 The `findsym` Command and the Default Symbolic Variable

The `findsym` command can be used to find which symbolic variables are present in an existing symbolic expression. The format of the command is:

`findsym(S)` or `findsym(S,n)`

The `findsym(S)` command displays the names of all the symbolic variables (separated by commas) that are in the expression `S` in alphabetical order. The `findsym(S,n)` command displays `n` symbolic variables that are in expression `S` in the default order. For one-letter symbolic variables, the default order starts with x, and followed by letters, according to their closeness to x. If there are two letters equally close to x, the letter that is after x in alphabetical order is first (y before w, and z before v). The default symbolic variable in a symbolic expression is the first variable in the default order. The default symbolic variable in an expression `S` can be identified by typing `findsym(S,1)`. Examples:

11.1.3 The `findsym` Command and the Default Symbolic Variable

```
>> syms x h w y d t
```
Define x, h, w, y, d, and t as symbolic variables.

```
>> S=h*x^2+d*y^2+t*w^2
```
Create a symbolic expression S.

```
S =
t*w^2 + h*x^2 + d*y^2
>> findsym(S)
```
Use the `findsym(S)` command.

```
ans =
d, h, t, w, x, y
```
The symbolic variables are displayed in alphabetical order.

```
>> findsym(S,5)
```
Use the `findsym(S,n)` command (n = 5).

```
ans =
x,y,w,t,h
```
5 symbolic variables are displayed in the default order.

```
>> findsym(S,1)
```
Use the `findsym(S,n)` command with n = 1.

```
ans =
x
```
The default symbolic variable is displayed.

Symbolic expressions often not in simplest or preferred form. Can change form by

- Collecting terms with the same power
- Expanding products
- Factoring out common multipliers
- Using mathematical and trigonometric identities
- Many other ways

The collect command:

```
collect(S)   collect(S,variable_name)
```

- Gathers terms in expression that have the variable with the same power
- In new expression, terms ordered in decreasing order of power
- `collect(S)` form works best when expression has only one symbolic variable

```
>> syms x y
>> S=(x^2+x-exp(x))*(x+3)
S =
(x + 3)*(x - exp(x) + x^2)
>> F = collect(S)
F =
x^3+4*x^2+(3-exp(x))*x-3*exp(x)
```

Define x and y as symbolic variables.

Create the symbolic expression $(x+3)(x-e^x+x^2)$ and assign it to S.

Use the `collect` command.

MATLAB returns the expression: $x^3+4x^2+(3--e^x)x-3e^x$.

- If expression has more than one variable MATLAB will collect the terms of one variable first, then of a second variable, etc.
  - Order of variables determined by MATLAB
  - User can specify the first variable by using the form `collect(S, variable_name)`

```
>> T=(2*x^2+y^2)*(x+y^2+3)
T =
(2*x^2+y^2)*(y^2+x+3)
>> G=collect(T)

G =
2*x^3+(2*y^2+6)*x^2+y^2*x+y^2*(y^2+3)
>> H=collect(T,y)

H =
y^4+(2*x^2+x+3)*y^2+2*x^2*(x+3)
```

Create the symbolic expression T $(2x^2 + y^2)(y^2 + x + 3)$.

Use the `collect(T)` command.

MATLAB returns the expression $2x^3 + (2y^2 + 6)x^2 + y^2x + y^2(y^2 + 3)$.

Use the `collect(T,y)` command.

MATLAB returns the expression $y^4 + (2x^2 + x + 3)y^2 + 2x^2(x + 3)$.

Note that when `collect(T)` is used, the reformatted expression is written in order of decreasing powers of x, but when `collect(T,y)` is used, the reformatted expression is written in order of decreasing powers of y.

The expand **command:**

```
expand(S)
```

expand command works in two ways

1. It carries out products of terms that include summation (at least one of the terms)
2. It uses trigonometric identities and exponential and logarithmic laws to expand corresponding terms that include summation

11.2.1 The `collect`, `expand`, and `factor` Commands

```
>> syms a x y
>> S=(x+5)*(x-a)*(x+4)
S =
-(a-x)*(x+4)*(x+5)
>> T=expand(S)
T =
20*x-20*a-9*a*x-a*x^2+9*x^2+x^3
>> expand(sin(x-y))

ans =
cos(y)*sin(x)-cos(x)*sin(y)
```

Define a, x, and y as symbolic variables.

Create the symbolic expression S: $-(a-x)(x+4)(x+5)$.

Use the expand command.

MATLAB returns the expression: $20x-20a-9ax-ax^2+9x^2+x^3$.

Use the expand command to expand $\sin(x-y)$.

MATLAB uses trig identity for the expansion.

The factor **command:**

`factor(S)`

`factor` changes an expression that is a polynomial to be a product of polynomials of a lower degree

```
>> syms x
>> S=x^3+4*x^2-11*x-30
S =
x^3+4*x^2-11*x-30
>> factor(S)
ans =
(x+5)*(x-3)*(x+2)
```

Define x as a symbolic variable.

Create the symbolic expression $x^3 + 4x^2 - 11x - 30$ and assign it to S.

Use the `factor` command.

MATLAB returns the expression $(x+5)(x-3)(x+2)$.

`simplify` and `simple` both general tools for simplifying an expression

- `simplify` uses built-in simplification rules to make simpler form than original expression
- `simple` makes form with least number of characters
  - No guarantee that form with least number of characters is the simplest but in practice often happens

The `simplify` command:

The `simplify` command uses mathematical operations (addition, multiplication, rules of fractions, powers, logarithms, etc.) and functional and trigonometric identities to generate a simpler form of the expression. The format of the `simplify` command is:

```
simplify(S)
```

where either `S` is the name of the existing expression to be simplified, or an expression to be simplified can be typed in for `S`.

Two examples are:

```
>> syms x y
>> S=(x^2+5*x+6)/(x+2)
S =
(x^2+5*x+6)/(x+2)
>> SA = simplify(S)
SA =
x+3
>> simplify((x+y)/(1/x+1/y))

ans =
x*y
```

Define x and y as symbolic variables.

Create the symbolic expression $(x^2+5x+6)/(x+2)$, and assign it to `S`.

Use the `simplify` command to simplify `S`.

MATLAB simplifies the expression to $x+3$.

Simplify $(x+y)/\left(\frac{1}{x}+\frac{1}{y}\right)$.

MATLAB simplifies the expression to xy).

The `simple` command:

`simple`

- Creates forms of expression with least number of characters, which is often simplest form
- Returns only one of forms but you can find out which one it chose

11.2.2 The `simplify` and `simple` Commands

`F = simple(S)`	`simple(S)`	`[F how] = simple(S)`
The shortest form of `S` is assigned to `F`.	All the simplification trails are displayed. The shortest is assigned to `ans`.	The shortest form of `S` is assigned to `F`. The name (string) of the simplification method is assigned to `how`.

The difference between the forms is in the output. The use of two of the forms is shown next.

```
>> syms x
>> S=(x^3-4*x^2+16*x)/(x^3+64)
S =
(x^3-4*x^2+16*x)/(x^3+64)
>> F = simple(S)
F =
x/(x+4)
>> [G how] = simple(S)
G =
x/(x+4)
how =
simplify
```

Define `x` as a symbolic variable.

Create the symbolic expression $\frac{x^3-4x^2+16x}{x^3+64}$, and assign it to `S`.

Use the `F = simple(S)` command to simplify `S`.

The simplest form of `S`, $x/(x+4)$, is assigned to `F`.

Use the `[G how] = simple(S)` command.

The simplest form of `S`, $x/(x+4)$, is assigned to `G`.

The word "simplify" is assigned to `G`, which means that the shortest form was obtained using the `simplify` command.

`pretty(S)` displays a symbolic expression in a form similar to the way you write the expression mathematically

For example:

```
>> syms a b c x
>> S=sqrt(a*x^2 + b*x + c)
S =
(a*x^2+b*x+c)^(1/2)
>> pretty(S)
                    2           1/2
                (a x  + b x + c)
```

Define `a`, `b`, `c`, and `x` as symbolic variables.

Create the symbolic expression: $\sqrt{ax^2 + bx + c}$, and assign it to `S`.

The `pretty` command displays the expression in a math format.

MATLAB can symbolically solve linear equations with one or more unknowns

Solving a single equation:

An algebraic equation can have one or several symbolic variables. If the equation has one variable, the solution is numerical. If the equation has several symbolic variables, a solution can be obtained for any of the variables in terms of the others. The solution is obtained by using the `solve` command, which has the form

`h = solve(eq)` or `h = solve(eq,var)`

- The argument `eq` can be the name of a previously created symbolic expression, or an expression that is typed in. When a previously created symbolic expression `S` is entered for `eq`, or when an expression that does not contain the = sign is typed in for `eq`, MATLAB solves the equation `eq` = 0.
- An equation of the form $f(x) = g(x)$ can be solved by typing the equation (including the = sign) as a string for `eq`.
- If the equation to be solved has more than one variable, the `solve(eq)` command solves for the default symbolic variable (see Section 11.1.3). A solution for any of the variables can be obtained with the `solve(eq,var)` command by typing the variable name for `var`.
- If the user types `solve(eq)`, the solution is assigned to the variable `ans`.
- If the equation has more than one solution, the output `h` is a symbolic column vector with a solution at each element. The elements of the vector are symbolic objects. When an array of symbolic objects is displayed, each row is enclosed with square brackets (see the following examples).

11.3 Solving Algebraic Equations

```
>> syms a b x y z
```
Define a, b, x, y, and z as symbolic variables.

```
>> h=solve(exp(2*z)-5)
```
Use the `solve` command to solve $e^{2z} - 5 = 0$.

```
h =
log(5)/2
```
The solution is assigned to h.

```
>> S=x^2-x-6
S =
x^2-x-6
```
Create the symbolic expression $x^2 - x - 6$, and assign it to S.

```
>> k=solve(S)
```
Use the `solve(S)` command to solve $x^2 - x - 6 = 0$.

```
k =
 -2
  3
```
The equation has two solutions. They are assigned to k, which is a column vector with symbolic objects.

```
>> solve('cos(2*y)+3*sin(y)=2')
```
Use the `solve` command to solve $\cos(2y) + 3\sin(y) = 2$. (The equation is typed as a string in the command.)

```
ans =
     pi/2
     pi/6
 (5*pi)/6
```
The solution is assigned to ans.

```
>> T= a*x^2+5*b*x+20
T =
a*x^2+5*b*x+20
```
Create the symbolic expression $ax^2 + 5bx + 20$, and assign it to T.

```
>> solve(T)
```
Use the `solve(S)` command to solve $T = 0$.

```
ans =
 -(5*b+5^(1/2)*(5*b^2-16*a)^(1/2))/(2*a)
 -(5*b-5^(1/2)*(5*b^2-16*a)^(1/2))/(2*a)
```
The equation $T = 0$ is solved for the variable x, which is the default variable.

```
>> M = solve(T,a)
```
Use the `solve(eq,var)` command to solve $T = 0$.

```
M =
-(5*b*x+20)/x^2
```
The equation $T = 0$ is solved for the variable a.

Solving a system of equations:

Use `solve` to solve a system of equations

- If number of equations same as number of variables, solution is numerical
- If number of variables greater than number of equations, solution is symbolic for desired variables in terms of other variables
- If system has one solution, each solution variable has one numerical value
- If system has multiple solutions, each variable can have several values

The format of the `solve` command for solving a system of n equations is:

```
output = solve(eq1,eq2,....,eqn)
```

or

```
output = solve(eq1,eq2,...,eqn,var1,var2,...,varn)
```

- The arguments `eq1,eq2,...,eqn` are the equations to be solved. Each argument can be a name of a previously created symbolic expression, or an expression that is typed in as a string. When a previously created symbolic expression `S` is entered, the equation is `S = 0`. When a string that does not contain the = sign is typed in, the equation is `expression = 0`. An equation that contains the = sign must be typed as a string.
- In the first format, if the number of equations n is equal to the number of variables in the equations, MATLAB gives a numerical solution for all the variables. If the number of variables is greater than the number of equations n, MATLAB gives a solution for n variables in terms of the rest of the variables. The variables for which solutions are obtained are chosen by MATLAB according to the default order (Section 11.1.3).
- When the number of variables is greater than the number of equations n, the user can select the variables for which the system is solved. This is done by using the second format of the `solve` command and entering the names of the variables `var1,var2,...,varn`.

Output from solve can have two different forms – a cell array or a structure

- *cell array* – an array in which each of the elements can be an array
  - Arrays in elements can be different dimensions
- *structure* – an array in which the elements have text names. Elements are called *fields*

When a cell array is used in the output of the `solve` command, the command has the following form (in the case of a system of three equations):

```
[varA, varB, varC] = solve(eq1,eq2,eq3)
```

- Once the command is executed, the solution is assigned to the variables `varA`, `varB`, and `varC`, and the variables are displayed with their assigned solution. Each of the variables will have one or several values (in a column vector) depending on whether the system of equations has one or several solutions.
- The user can select any names for `varA`, `varB`, and `varC`. MATLAB assigns the solution for the variables in the equations in alphabetical order. For example, if the variables for which the equations are solved are x, u, and t, the solution for t is assigned to `varA`, the solution for u is assigned to `varB`, and the solution for x is assigned to `varC`.

The following examples show how the `solve` command is used for the case where a cell array is used in the output:

```
>> syms x y t
>> S=10*x+12*y+16*t;
>> [xt yt]=solve(S, '5*x-y=13*t')
xt =
2*t
yt =
-3*t
```

Define x, y, and t as symbolic variables.

Assign to S the expression $10x + 12y + 16t$.

Use the `solve` command to solve the system: $10x + 12y + 16t = 0$, $5x - y = 13t$

Output in a cell array with two cells named `xt` and `yt`.

The solutions for x and y are assigned to `xt` and `yt`, respectively.

In the example above, notice that the system of two equations is solved by MATLAB for x and y in terms of t, since x and y are the first two variables in the default order. The system, however, can be solved for different variables. As an example, the system is solved next for y and t in terms of x (using the second form of the `solve` command:

```
>> [tx yx]=solve(S,'5*x-y=13*t',y,t)
tx =
x/2
yx =
-(3*x)/2
```

The variables for which the system is solved (y and t) are entered.

The solutions for the variables for which the system is solved are assigned in alphabetical order. The first cell has the solution for t, and the second cell has the solution for y.

When a structure is used in the output of the `solve` command, the command has the form (in the case of a system of three equations)

```
AN = solve(eq1,eq2,eq3)
```

- `AN` is the name of the structure.
- Once the command is executed the solution is assigned to `AN`. MATLAB displays the name of the structure and the names of the fields of the structure, which are the names of the variables for which the equations are solved. The size and the type of each field is displayed next to the field name. The content of each field, which is the solution for the variable, is not displayed.
- To display the content of a field (the solution for the variable), the user has to type the address of the field. The form for typing the address is: `structure_name.field_name` (see example below).

As an illustration the system of equations solved in the last example is solved again using a structure for the output.

```
>> syms x y t
>> S=10*x+12*y+16*t;
>> AN=solve(S,'5*x-y=13*t')
AN =
  x: [1x1 sym]
  y: [1x1 sym]
>> AN.x
ans =
2*t
>> AN.y
ans =
-3*t
```

Use the `solve` command to solve the system: $10x + 12y + 16t = 0$, $5x - y = 13t$

MATLAB displays the name of the structure `AN` and the names of its fields `x` and `y` (size and type), which are the names of the variables for which the equations are solved.

Type the address of the field `x`.

The content of the field (the solution for x) is displayed.

Type the address of the field `y`.

The content of the field (the solution for y) is displayed.

To do symbolic differentiation, use `diff(S)` or `diff(S,var)`

- `S` is name of existing symbolic expression, or can type in an expression
- For `diff(S)`
  - If only one variable in `S`, command differentiates with respect to that variable
  - If multiple variables in `S`, command differentiates with respect to first variable in default order
- `diff(S,var)` differentiates with respect to `var`

To get the *n*th derivative, use `diff(S,n)` or `diff(S,var,n)`

- e.g., `diff(S,2)` is second derivative, `diff(S,3)` is third derivative, etc.

```
>> syms x y t
```
Define x, y, and t as symbolic variables.

```
>> S=exp(x^4);
```
Assign to S the expression e^{x^4}.

```
>> diff(S)
```
Use the `diff(S)` command to differentiate S.

```
ans =
4*x^3*exp(x^4)
```
The answer $4x^3e^{x^4}$ is displayed.

```
>> diff((1-4*x)^3)
```
Use the `diff(S)` command to differentiate $(1-4x)^3$.

```
ans =
-12*(1-4*x)^2
```
The answer $-12(1-4x)^2$ is displayed.

```
>> R=5*y^2*cos(3*t);
```
Assign to R the expression $5y^2\cos(3t)$.

```
>> diff(R)
```
Use the `diff(R)` command to differentiate R.

```
ans =
10*y*cos(3*t)
```
MATLAB differentiates R with respect to y (default symbolic variable); the answer $10y\cos(3t)$ is displayed.

```
>> diff(R,t)
```
Use the `diff(R,t)` command to differentiate R w.r.t. t.

```
ans =
-15*y^2*sin(3*t)
```
The answer $-15y^2\sin(3t)$ is displayed.

```
>> diff(S,2)
```
Use `diff(S,2)` command to obtain the second derivative of S.

```
ans =
12*x^2*exp(x^4)+16*x^6*exp(x^4)
```
The answer $12x^2e^{x^4}+16x^6e^{x^4}$ is displayed.

Symbolic integration can be carried out by using the `int` command. The command can be used for determining indefinite integrals (antiderivatives) and definite integrals. For indefinite integration the form of the command is:

`int(S)` or `int(S,var)`

- Either `S` can be the name of a previously created symbolic expression, or an expression can be typed in for `S`.
- In the `int(S)` command, if the expression contains one symbolic variable, the integration is carried out with respect to that variable. If the expression contains more than one variable, the integration is carried out with respect to the default symbolic variable (Section 11.1.3).
- In the `int(S,var)` command, which is used for integration of expressions with several symbolic variables, the integration is carried out with respect to the variable `var`.

```
>> syms x y t
```
Define x, y, and t as symbolic variables.

```
>> S=2*cos(x)-6*x;
```
Assign to S the expression $2\cos(x) - 6x$.

```
>> int(S)
```
Use the `int(S)` command to integrate S.

```
ans =
2*sin(x)-3*x^2
```
The answer $2\sin(x) - 3x^2$ is displayed.

```
>> int(x*sin(x))
```
Use the `int(S)` command to integrate $x\sin(x)$.

```
ans =
sin(x)-x*cos(x)
```
The answer $\sin(x) - x\cos(x)$ is displayed.

```
>>R=5*y^2*cos(4*t);
```
Assign to R the expression $5y^2\cos(4t)$.

```
>> int(R)
```
Use the `int(R)` command to integrate R.

```
ans =
(5*y^3*cos(4*t))/3
```
MATLAB integrates R with respect to y (default symbolic variable); the answer $5y^3\cos(4t)/3$ is displayed.

```
>> int(R,t)
```
Use the `int(R,t)` command to integrate R w.r.t. t.

```
ans =
(5*y^2*sin(4*t))/4
```
The answer $5y^2\sin(4t)/4$ is displayed.

For definite integration the form of the command is:

`int(S,a,b)` or `int(S,var,a,b)`

- `a` and `b` are the limits of integration. The limits can be numbers or symbolic variables.

For example, determination of the definite integral $\int_0^{\pi} (\sin y - 5y^2)dy$ with MATLAB is:

```
>> syms y
>> int(sin(y)-5*y^2,0,pi)
ans =
2 - (5*pi^3)/3
```

- It is possible also to use the `int` command by typing the expression to be integrated as a string without having the variables in the expression first created as symbolic objects. However, the variables in the integrated expression do not exist as independent symbolic objects.
- Integration can sometimes be a difficult task. A closed-form answer may not exist, or if it exists, MATLAB might not be able to find it. When that happens MATLAB returns `int(S)` and the message `Explicit integral could not be found.`

A first-order ordinary, differential equation (ODE) contains the first derivative of the dependent variable

- For example, $\frac{dy}{dt} = f(t, y)$, where *t* is the independent variable and *y=y(t)* is the dependent variable

A second-order ODE contains the second derivative (and possibly also the first), i.e., $\frac{dy}{dt} = f(t, y, \frac{dy}{dt})$

A *solution* is a function *y(t)* that satisfies the differential equation

- A *general solution* contains constants whose values are unknown
- In a *particular solution* the values of the constants are set by initial conditions

Use `dsolve` to get a general or particular solution to an ODE

General solution:

For obtaining a general solution, the `dsolve` command has the form:

`dsolve('eq')` or `dsolve('eq','var')`

- `eq` is the equation to be solved. It has to be typed as a string (even if the variables are symbolic objects).
- The variables in the equation don't have to first be created as symbolic objects. (If they have not been created, then, in the solution the variables will not be symbolic objects.)
- Any letter (lowercase or uppercase), except `D` can be used for the dependent variable.
- In the `dsolve('eq')` command the independent variable is assumed by MATLAB to be `t` (default).
- In the `dsolve('eq','var')` command the user defines the independent variable by typing it for `var` (as a string).
- In specifying the equation the letter `D` denotes differentiation. If y is the dependent variable and t is the independent variable, `Dy` stands for $\frac{dy}{dt}$. For example, the equation $\frac{dy}{dt} + 3y = 100$ is typed in as `'Dy + 3*y = 100'`.
- A second derivative is typed as `D2`, third derivative as `D3`, and so on. For example, the equation $\frac{d^2y}{dt^2} + 3\frac{dy}{dt} + 5y = \sin(t)$ is typed in as: `'D2y + 3*Dy + 5*y = sin(t)'`.
- The variables in the ODE equation that is typed in the `dsolve` command do not have to be previously created symbolic variables.
- In the solution MATLAB uses `C1`, `C2`, `C3`, and so on, for the constants of integration.

For example, a general solution of the first-order ODE $\frac{dy}{dt} = 4t + 2y$ is obtained by:

```
>> dsolve('Dy=4*t+2*y')

ans =
C1*exp(2*t) - 2*t - 1
```

The answer $y = C_1 e^{2t} - 2t - 1$ is displayed.

A general solution of the second-order ODE $\frac{d^2x}{dt^2} + 2\frac{dx}{dt} + x = 0$ is obtained by:

```
>> dsolve('D2x+2*Dx+x=0')
```

The following examples illustrate the solution of differential equations that contain symbolic variables in addition to the independent and dependent variables.

```
>> dsolve('Ds=a*x^2')

ans =
a*t*x^2 + C1
```

The independent variable is t (default). MATLAB solves the equation $\frac{ds}{dt} = ax^2$.

The solution $s = ax^2t + C_1$ is displayed.

```
>> dsolve('Ds=a*x^2','x')

ans =
(a*x^3)/3 + C1
```

The independent variable is defined to be x. MATLAB solves the equation $\frac{ds}{dx} = ax^2$.

The solution $s = \frac{1}{3}ax^3 + C_1$ is displayed.

```
>> dsolve('Ds=a*x^2','a')

ans =
(a^2*x^2)/2 + C2
```

The independent variable is defined to be a. MATLAB solves the equation $\frac{ds}{da} = ax^2$.

The solution $s = \frac{1}{2}a^2x^2 + C_1$ is displayed.

Particular solution:

A particular solution of an ODE can be obtained if boundary (or initial) conditions are specified. A first-order equation requires one condition, a second-order equation requires two conditions, and so on. For obtaining a particular solution, the `dsolve` command has the form

First-order ODE:

```
dsolve('eq','cond1','var')
```

Higher-order ODE:

```
dsolve('eq','cond1','cond2',....,'var')
```

- For solving equations of higher order, additional boundary conditions have to be entered in the command. If the number of conditions is less than the order of the equation, MATLAB returns a solution that includes constants of integration (`C1`, `C2`, `C3`, and so on).
- The boundary conditions are typed in as strings in the following:

Math form	MATLAB form
$y(a) = A$	`'y(a)=A'`
$y'(a) = A$	`'Dy(a)=A'`
$y''(a) = A$	`'D2y(a)=A'`

- The argument `'var'` is optional and is used to define the independent variable in the equation. If none is entered, the default is t.

For example, the first-order ODE $\frac{dy}{dt} + 4y = 60$, with the initial condition $y(0) = 5$ is solved with MATLAB by:

```
>> dsolve('Dy+4*y=60','y(0)=5')
ans =
15 - 10/exp(4*t)
```

The answer $y = 15 - (10/e^{4t})$ is displayed.

The second-order ODE $\frac{d^2y}{dt^2} - 2\frac{dy}{dt} + 2y = 0$, $y(0) = 1$, $\left.\frac{dy}{dt}\right|_{t=0} = 0$, can be solved with MATLAB by:

```
>> dsolve('D2y-2*Dy+2*y=0','y(0)=1','Dy(0)=0')
ans =
exp(t)*cos(t)-exp(t)*sin(t)
>> factor(ans)
ans =
exp(t)*(cos(t)-sin(t))
```

The answer $y = e^t\cos(t) - e^t\sin(t)$ is displayed.

The answer can be simplified with the `factor` command.

The simplified answer $y = e^t(\cos(t) - \sin(t))$ is displayed.

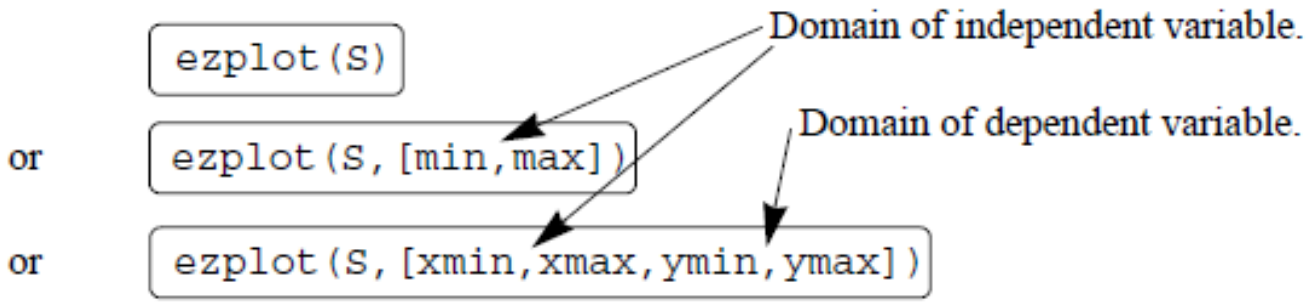
Additional examples of solving differential equations are shown in Sample Problem 11-5.

If MATLAB cannot find a solution, it returns an empty symbolic object and the message `Warning: explicit solution could not be found.`

Can easily plot symbolic expressions with `ezplot`

- If symbolic expression `S` contains one symbolic variable `var`, `ezplot` plots `S(var)` vs `var`
- If it contains two symbolic variables, `ezplot` graphs `S(var1,var2)=0` in the plane

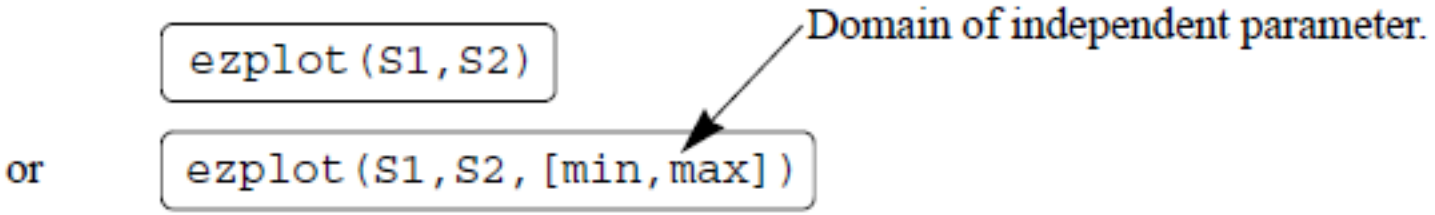
To plot a symbolic expression `S` that contains one or two variables, the `ezplot` command is:



- `S` is the symbolic expression to be plotted. It can be the name of a previously created symbolic expression, or an expression can be typed in for `S`.
- It is also possible to type the expression to be plotted as a string without having the variables in the expression first created as symbolic objects.
- If `S` has one symbolic variable, a plot of $S(var)$ versus (var) is created, with the values of var (the independent variable) on the abscissa (horizontal axis), and the values of $S(var)$ on the ordinate (vertical axis).
- If the symbolic expression `S` has two symbolic variables, `var1` and `var2`, the expression is assumed to be a function with the form $S(var1,var2) = 0$. MATLAB creates a plot of one variable versus the other variable. The variable that is first in alphabetic order is taken to be the independent variable. For example, if the variables in S are x and y, then x is the independent variable and is plotted on the abscissa and y is the dependent variable plotted on the ordinate. If the variables in S are u and v, then u is the independent variable and v is the dependent variable.

- In the `ezplot(S)` command, if `S` has one variable ($S(var)$), the plot is over the domain $-2\pi < var < 2\pi$ (default domain) and the range is selected by MATLAB. If `S` has two variables ($S(var1,var2)$), the plot is over $-2\pi < var1 < 2\pi$ and $-2\pi < var2 < 2\pi$.
- In the `ezplot(S,[min,max])` command the domain for the independent variable is defined by `min` and `max`:—$min < var < max$—and the range is selected by MATLAB.
- In the `ezplot(S,[xmin,xmax,ymin,ymax])` command the domain for the independent variable is defined by `xmin` and `xmax`, and the domain of the dependent variable is defined by `ymin` and `ymax`.

Can also use `ezplot` to plot a 2-D parametric curve, e.g., *x=x(t)* and *y=y(t)*



- `S1` and `S2` are symbolic expressions containing the same single symbolic variable, which is the independent parameter. `S1` and `S2` can be the names of previously created symbolic expressions, or expressions can be typed in.
- The command creates a plot of $S2(var)$ versus $S1(var)$. The symbolic expression that is typed first in the command (`S1` in the definition above) is used for the horizontal axis, and the expression that is typed second (`S2` in the definition above) is used for the vertical axis.
- In the `ezplot(S1,S2)` command the domain of the independent variable is $0 < var < 2\pi$ (default domain).
- In the `ezplot(S1,S2,[min,max])` command the domain for the independent variable is defined by `min` and `max`: $min < var < max$.

Additional comments:

Once you create plot with `ezplot`, can format same way as plots made with `plot`

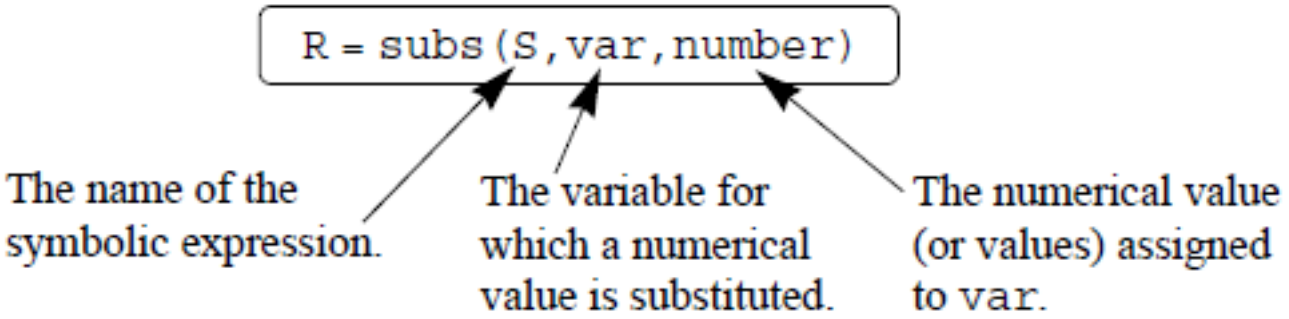
- When plot symbolic expression with `ezplot` it automatically displays expression at top of plot

Table 11-1 in book shows several examples of symbolic-expression plots

After manipulating symbolic expression, often need to evaluate it numerically. Can do this with the MATLAB command `subs`, which substitutes numerical values for symbolic ones

Substituting a numerical value for one symbolic variable:

A numerical value (or values) can be substituted for one symbolic variable when a symbolic expression has one or more symbolic variables. In this case the `subs` command has the form:



- `number` can be one number (a scalar), or an array with many elements (a vector or a matrix).
- The value of `S` is calculated for each value of `number` and the result is assigned to `R`, which will have the same size as `number` (scalar, vector, or matrix).
- If `S` has one variable, the output `R` is numerical. If `S` has several variables and a numerical value is substituted for only one of them, the output `R` is a symbolic expression.

An example with an expression that includes one symbolic variable is:

```
>> syms x
```

Define x, as a symbolic variable.

```
>> S=0.8*x^3+4*exp(0.5*x)
S =
4/5*x^3+4*exp(1/2*x)
```

Assign to S the expression: $0.8x^3 + 4e^{(0.5x)}$.

```
>> SD=diff(S)
SD =
12/5*x^2+2*exp(1/2*x)
```

Use the `diff(S)` command to differentiate S.

The answer $12x^2/5 + 2e^{(0.5x)}$ is assigned to SD.

```
>> subs(SD, x, 2)
ans =
   15.0366
```

Use the `subs` command to substitute $x = 2$ in SD.

The value of SD is displayed.

```
>> SDU=subs(SD, x, [2:0.5:4])
SDU =
   15.0366   21.9807   30.5634   40.9092   53.1781
```

Use the `subs` command to substitute $x = [2, 2.5, 3, 3.5, 4]$ (vector) in SD.

The values of SD (assigned to SDU) for each value of x are displayed in a vector.

In the last example, notice that when the numerical value of the symbolic expression is calculated, the answer is numerical (the display is indented). An example of substituting numerical values for one symbolic variable in an expression that has several symbolic variables is:

```
>> syms a g t v
>> Y=v^2*exp(a*t)/g
Y =
v^2*exp(a*t)/g
>> subs(Y,t,2)
ans =
v^2*exp(2*a)/g
>> Yt=subs(Y,t,[2:4])
Yt =
[ v^2*exp(2*a)/g, v^2*exp(3*a)/g, v^2*exp(4*a)/g]
```

Define a, g, t and v as symbolic variables.

Create the symbolic expression: $v^2e^{(at)}/g$ and assign it to Y.

Use the subs command to substitute $t = 2$ in SD.

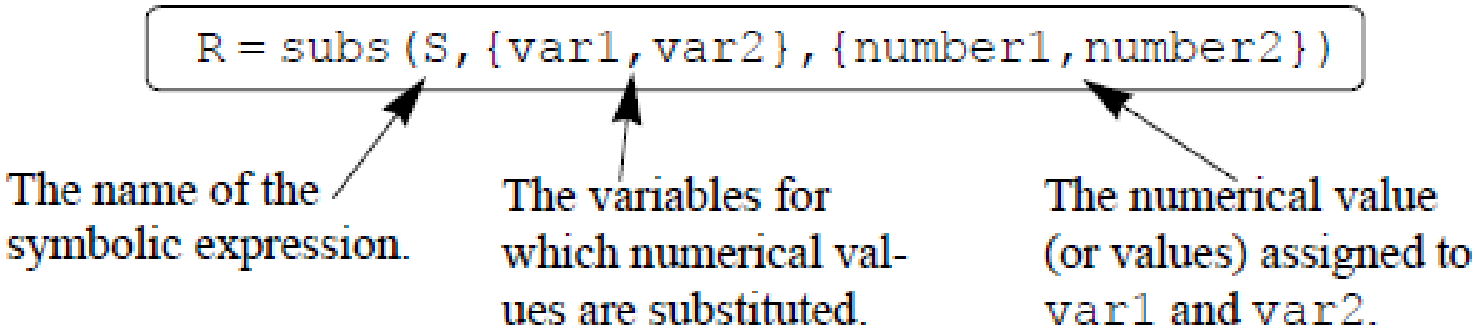
The answer $v^2e^{(2a)}/g$ is displayed.

Use the subs command to substitute $t = [2, 3, 4]$ (vector) in Y.

The answer is a vector with elements of symbolic expressions for each value of t.

Substituting a numerical value for two or more symbolic variables:

Can use `subs` to substitute for two or more symbolic variables. For example, for two symbolic variables



- The variables `var1` and `var2` are the variables in the expression `S` for which the numerical values are substituted. The variables are typed as a cell array (inside curly braces `{ }`). Cell array is an array of cells where each cell can be an array of numbers or text.

- The variables `var1` and `var2` are the variables in the expression `S` for which the numerical values are substituted. The variables are typed as a cell array (inside curly braces `{ }`). Cell array is an array of cells where each cell can be an array of numbers or text.
- The numbers `number1,number2` substituted for the variables are also typed as a cell array (inside curly braces `{ }`). The numbers can be scalars, vectors, or matrices. The first cell in the numbers cell array (`number1`) is substituted for the variable that is in the first cell of the variable cell array (`var1`), and so on.
- If all the numbers that are substituted for variables are scalars, the outcome will be one number or one expression (if some of the variables are still symbolic).
- If, for at least one variable, the substituted numbers are an array, the mathematical operations are executed element-by-element and the outcome is an array of numbers or expressions. It should be emphasized that the calculations are performed element-by-element even though the expression `S` is not typed in the element-by-element notation. This also means that all the arrays substituted for different variables must be of the same size.
- It is possible to substitute arrays (of the same size) for some of the variables and scalars for other variables. In this case, in order to carry out element-by-element operations, MATLAB expands the scalars (array of ones times the scalar) to produce an array result.

```
>> syms a b c e x
>> S=a*x^e+b*x+c
S =
a*x^e+b*x+c
>> subs(S,{a,b,c,e,x},{5,4,-20,2,3})
ans =
    37
>> T=subs(S,{a,b,c},{6,5,7})
T =
5*x+ 6*x^e+7
>> R=subs(S,{b,c,e},{[2 4 6],9,[1 3 5]})
R =
[  2*x+a*x+9, a*x^3+4*x+9, a*x^5+6*x+9]
>> W=subs(S,{a,b,c,e,x},{[4 2 0],[2 4 6],[2 2 2],[1 3 5],[3 2 1]})
W =
    20    26     8
```

Define a, b, c, e, and x as symbolic variables.

Create the symbolic expression $ax^e + bx + c$ and assigned it to S.

Substitute in S scalars for all the symbolic variables.

Cell array.

Cell array.

The value of S is displayed.

Substitute in S scalars for the symbolic variables a, b, and c.

The result is an expression with the variables x and e.

Substitute in S a scalar for c, and vectors for b and e.

The result is a vector of symbolic expressions.

Substitute in S vectors for all the variables.

The result is a vector of numerical values.

Can also substitute numerical values into a symbolic expression by first setting symbolic variables in the expression to numerical values and then calling `subs`

```
R = subs(S)
```

- Note that once you redefine symbolic variables to numerical variables, you can't use them as symbolic variables any more

```
>> syms A c m x y
>> S=A*cos(m*x)+c*y
S =
c*y+A*cos(m*x)
>> A=10; m=0.5; c=3;
>> subs(S)
ans =
3*y + 10*cos(x/2)
>> x=linspace(0,2*pi,4);
>> T = subs(S)
T =
[ 3*y+10, 3*y+5, 3*y-5, 3*y-10]
```

Define A, c, m, x, and y as symbolic variables.

Create the symbolic expression $A\cos(mx) + cy$ and assign it to S.

Assign numerical values to variables A, m, and c.

Use the subs command with the expression S.

The numerical values of variables A, m, and c are substituted in S.

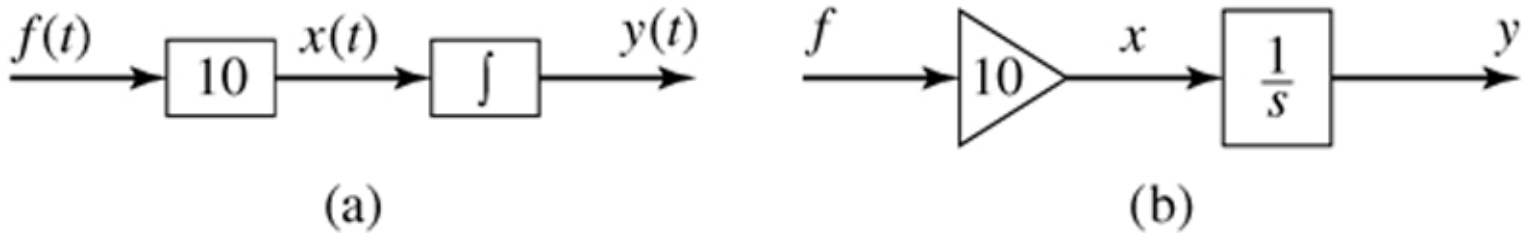
Assign numerical values (vector) to variable x.

Use the subs command with the expression S.

The numerical values of variables A, m, c, and x are substituted. The result is a vector of symbolic expressions.

Simulink

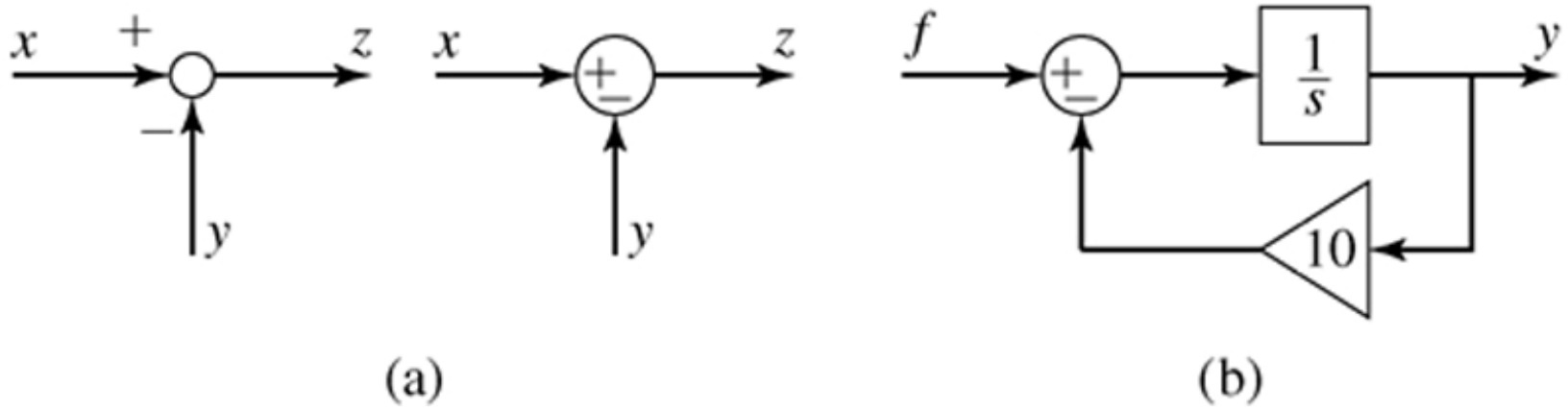
Simulation diagrams for $\mathbf{y = \int 10\, f(t)\, dt}$. Figure 10.1–1 on page 420



10-2

$$y(t) = \int [f(t) - 10y]\,dt$$

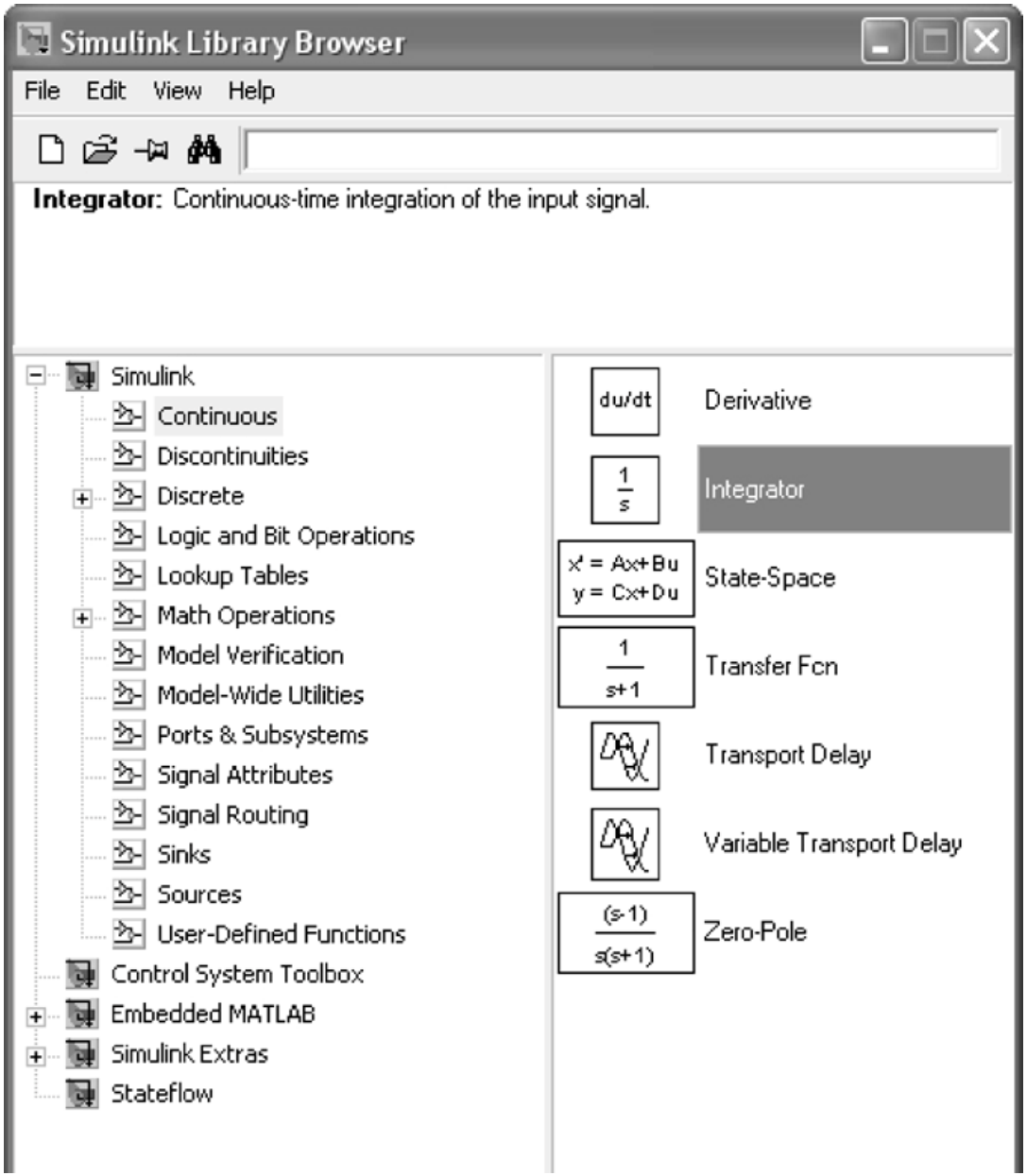
$$y = \frac{1}{s}(f - 10y)$$



(a) The summer element. (b) Simulation diagram for $\dot{y} = f(t) - 10y$.

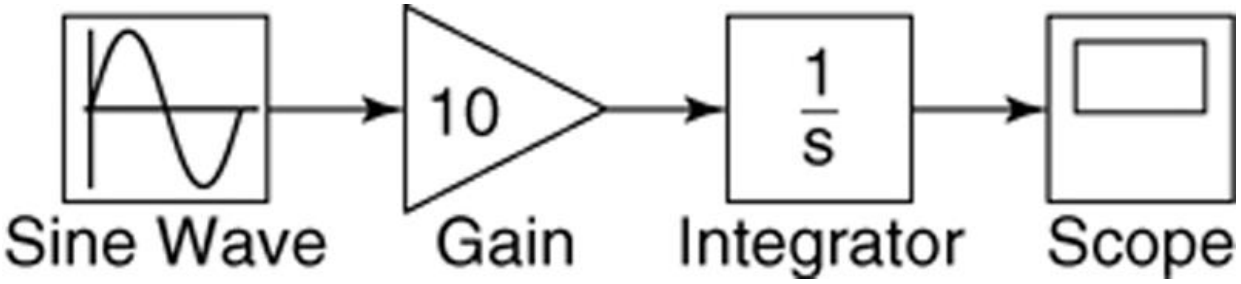
10-3

The Simulink Library Browser. Figure 10.2–1 on page 422



10-4

Simulink model for $y = 10 \sin t$. Figure 10.2–2 on page 423



For the steps needed to construct this model, see Example 10.2-1 on pages 422-424.

Note that blocks have a Block Parameters window that opens when you double-click on the block.

This window contains several items, the number and nature of which depend on the specific type of block.

In general, you can use the default values of these parameters, except where we have explicitly indicated that they should be changed.

You can always click on **Help** within the Block Parameters window to obtain more information.

See page 424 for more information.

Note that most blocks have default labels.

You can edit text associated with a block by clicking on the text and making the changes.

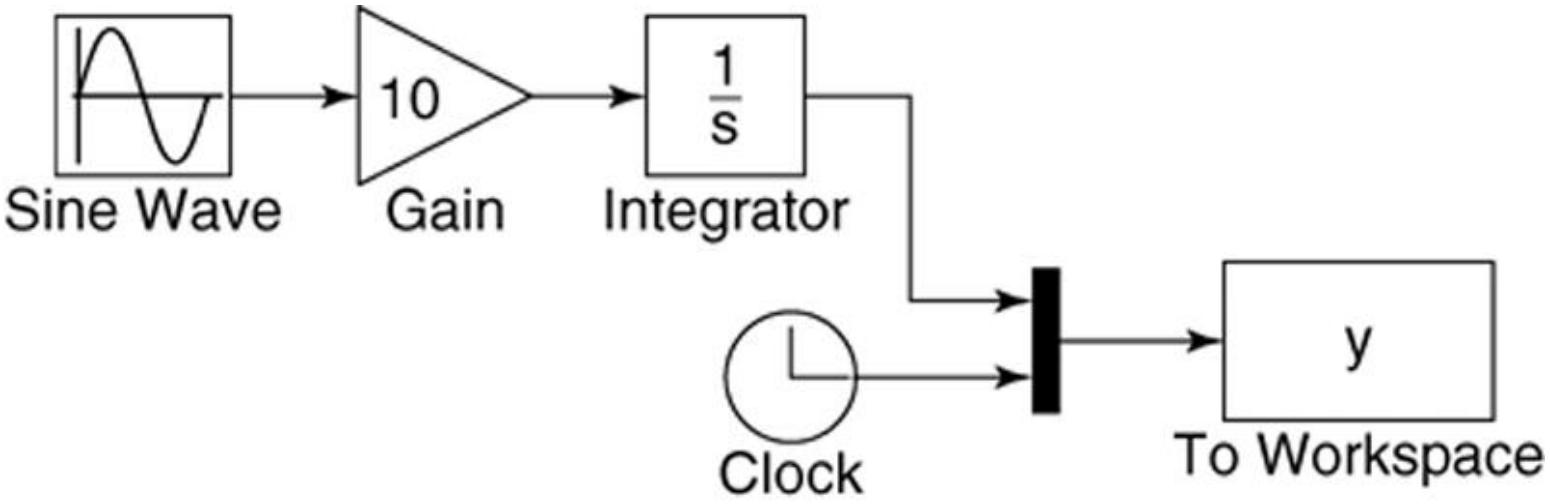
You can save the Simulink model as an `.mdl` file by selecting **Save** from the **File** menu in Simulink.

The model file can then be reloaded at a later time.

You can also print the diagram by selecting **Print** on the **File** menu.

Simulink model using the Clock and To Workspace blocks.

Figure 10.2–5 on page 425



For the steps needed to construct this model, see Example 10.2-2 on pages 425-426.

Double-click on the To Workspace block. You can specify any variable name you want as the output; the default is `simout`. Change its name to `y`.

The output variable `y` will have as many rows as there are simulation time steps, and as many columns as there are inputs to the block.

The second column in our simulation will be time, because of the way we have connected the Clock to the second input port of the Mux.

Specify the Save format as Array. Use the default values for the other parameters (these should be `inf`, `1`, and `-1` for Maximum number of rows, Decimation, and Sample time, respectively). Click on **OK.**

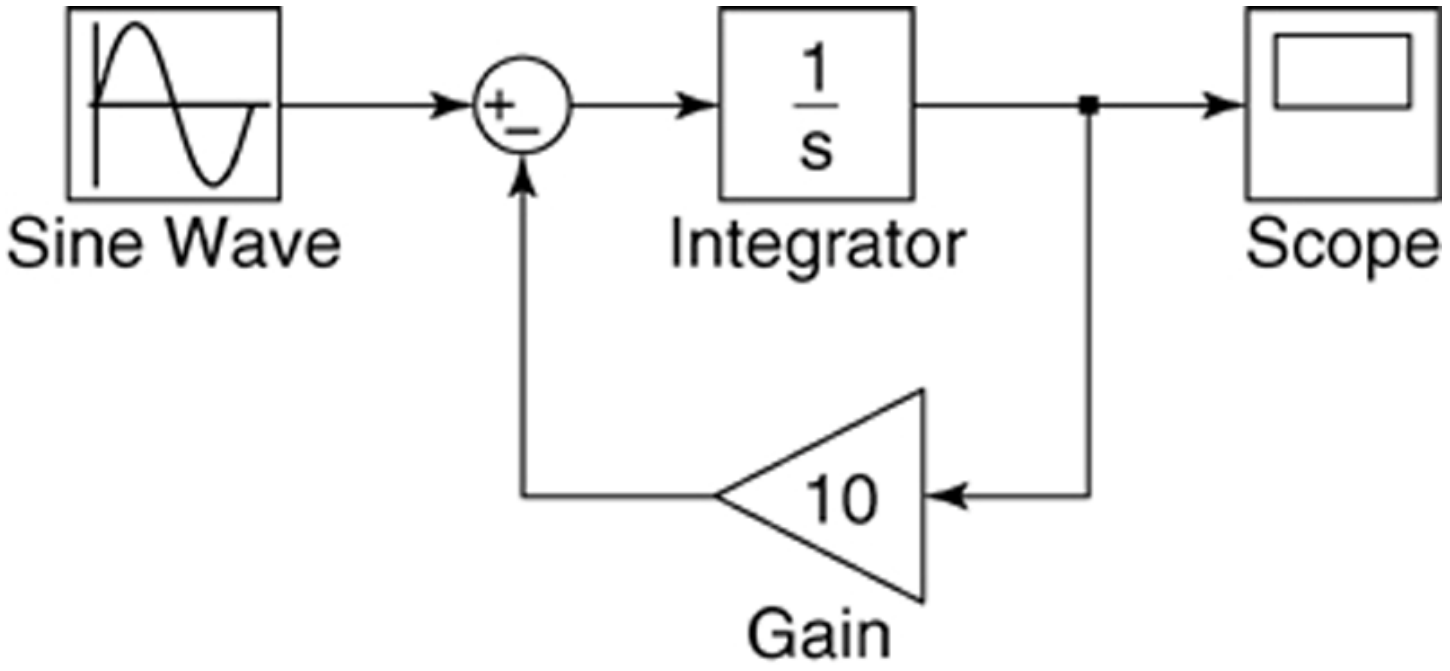
Simulink can be configured to put the time variable `tout` into the MATLAB workspace automatically when you are using the To Workspace block.

This is done with the Data I/O tab under **Configuration Parameters** on the Simulation menu.

The alternative is to use the Clock block to put `tout` into the workspace.

The Clock block has one parameter, Decimation. If this parameter is set to 1, the Clock block will output the time every time step; if set to 10 for example, the block will output every 10 time steps, and so on.

Simulink model for $y = -10y + f(t)$. Figure 10.2–6 on page 426

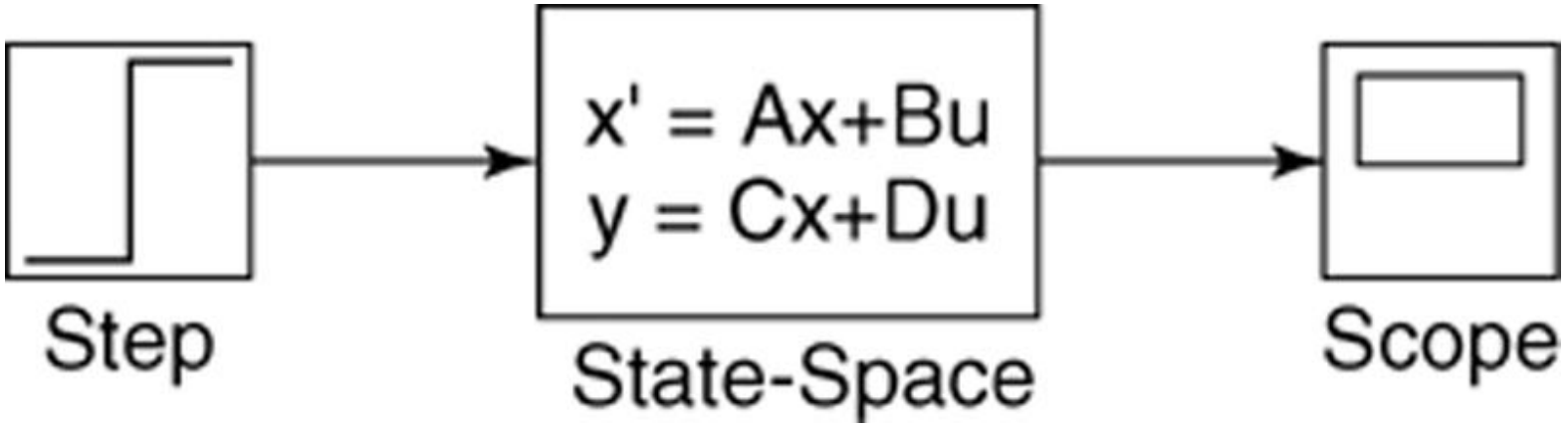


For the steps needed to construct this model, see Example 10.2-3 on pages 426-427.

A vibrating system having two masses. Figure 10.3–1 on page 428. The state-variable model and simulation construction steps are given in Example 10.3-1 on pages 427-430.

10-12

Simulink model of the system having two masses. Use of the State-Space block and the Step block. Figure 10.3–2 on page 429.

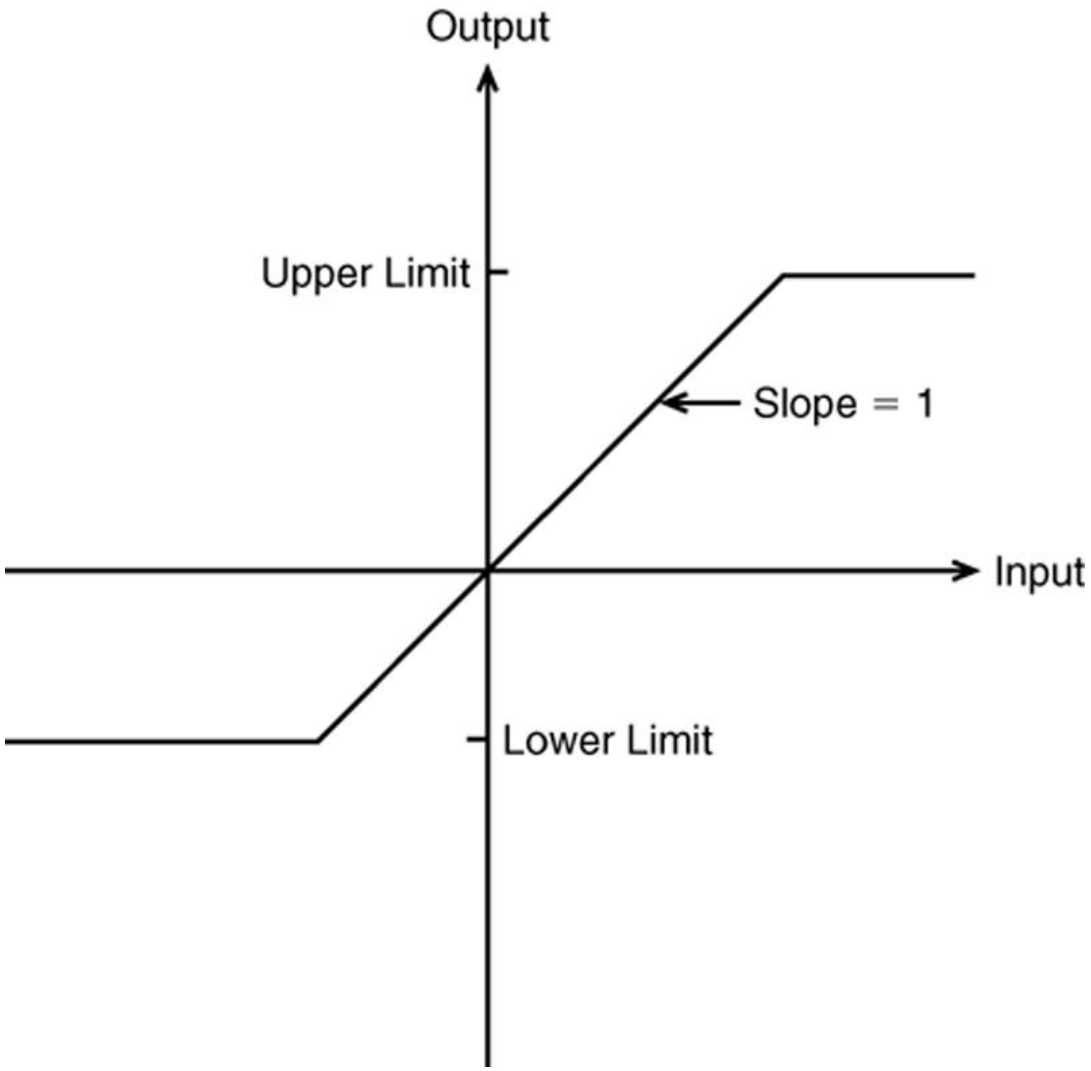


10-13

When you are connecting inputs to the State-Space block, care must be taken to connect them in the proper order.

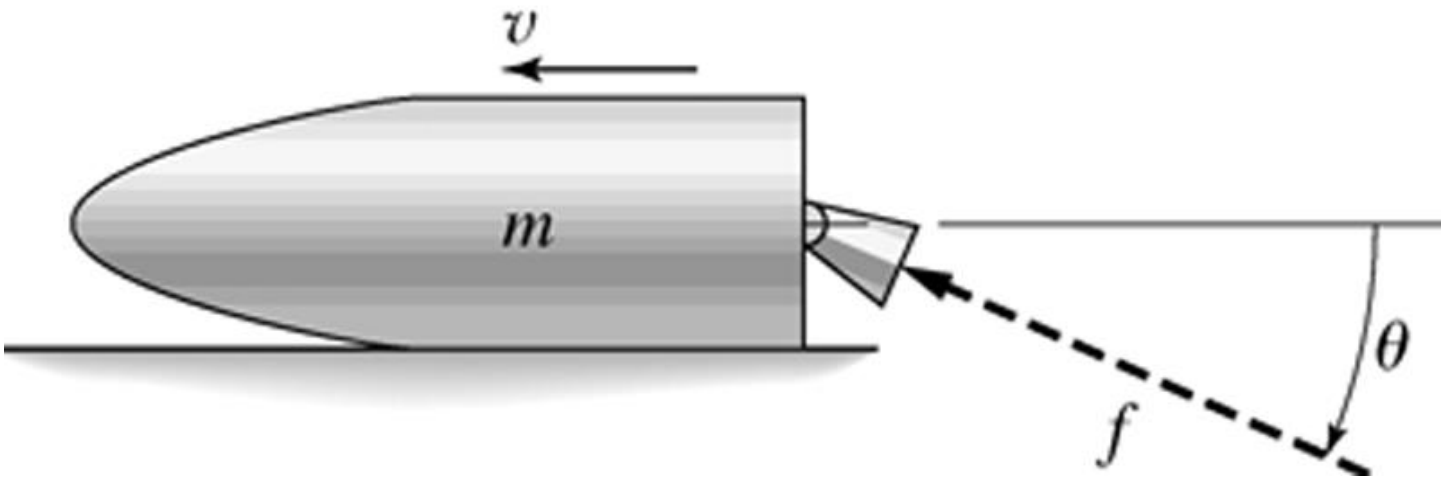
Similar care must be taken when connecting the block's outputs to another block.

The saturation nonlinearity. Figure 10.4–1 on page 431.



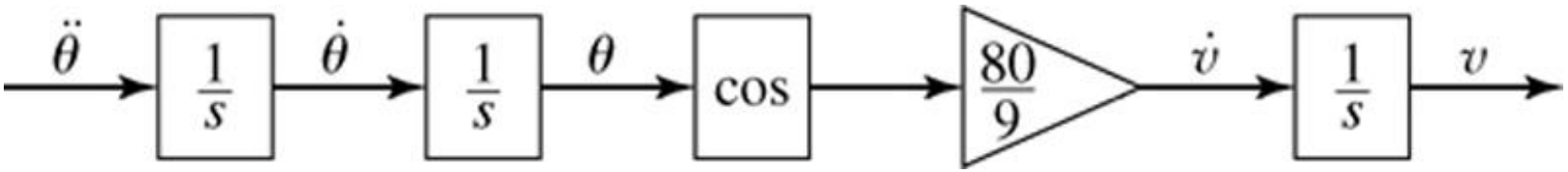
10-15

Example 10.4-1: A rocket-propelled sled. Figure 10.4–2 on page 431.

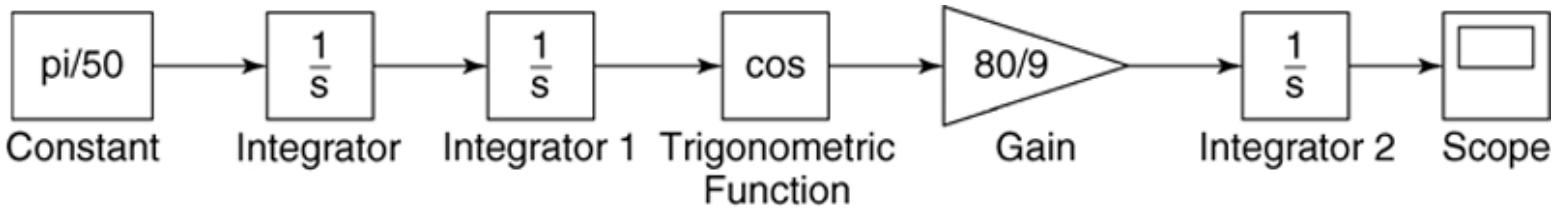


10-16

Rocket-propelled sled example. Simulation diagram for $v = (80/9)\cos(\pi t^2/100)$. Example 10.4-1 and Figure 10.4–3 on page 432.

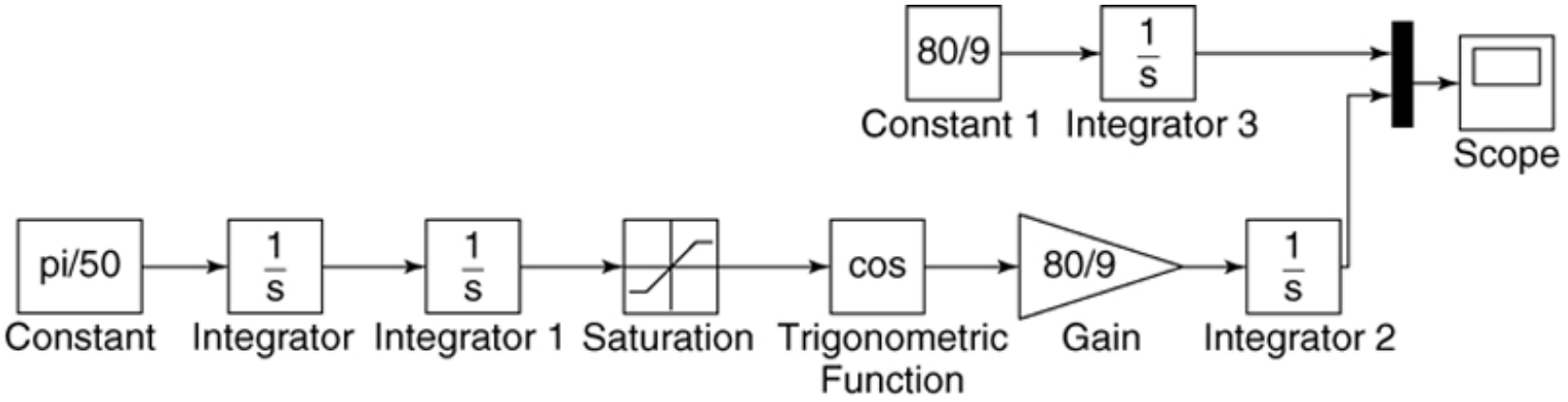


Simulink model for $v = (80/9)\cos(\pi t^2/100)$. Figure 10.4–4 on page 433.

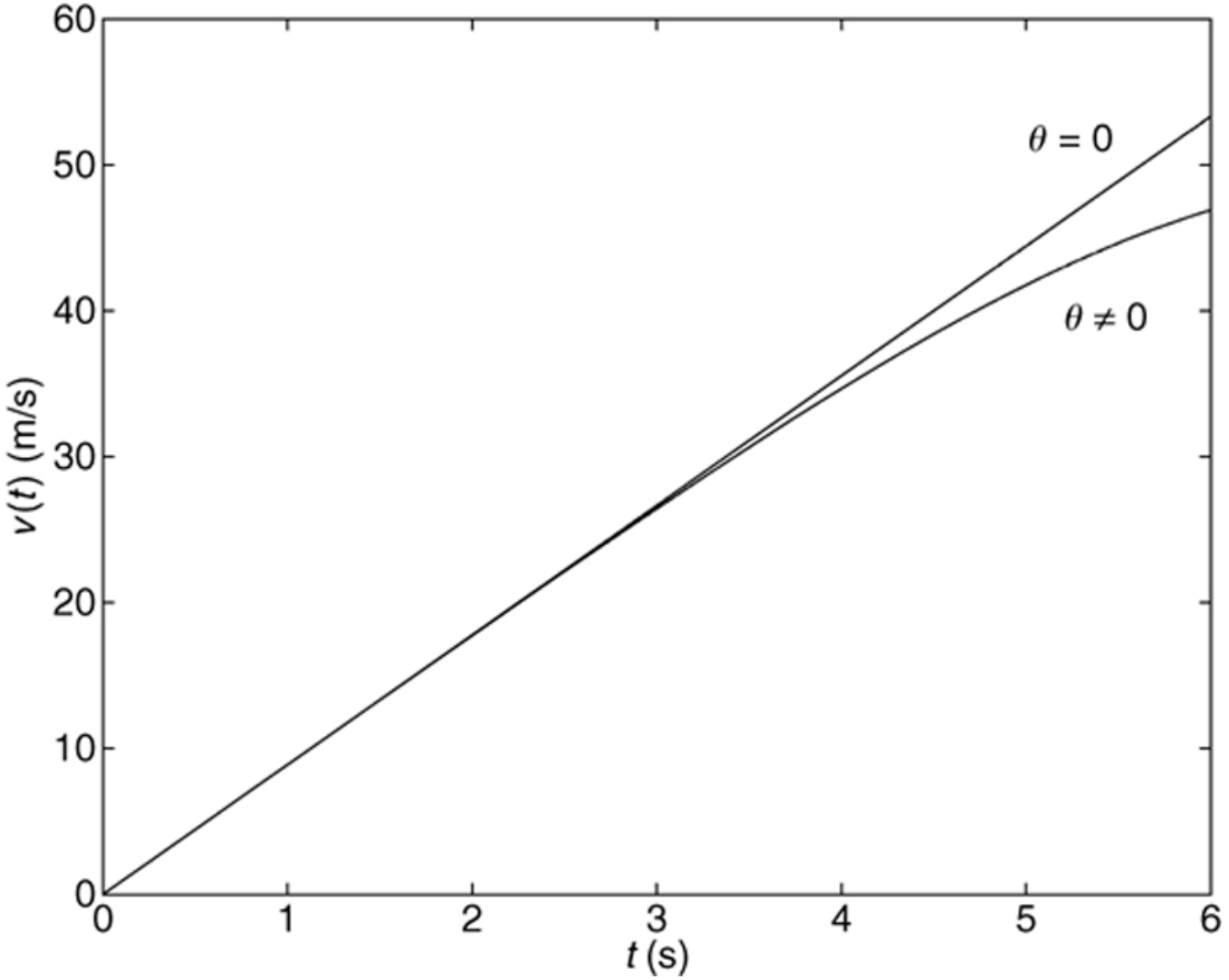


10-18

Simulink model for $\upsilon = (80/9)\cos(\pi t^2/100)$ with a Saturation block. Figure 10.4–5 on page 433.

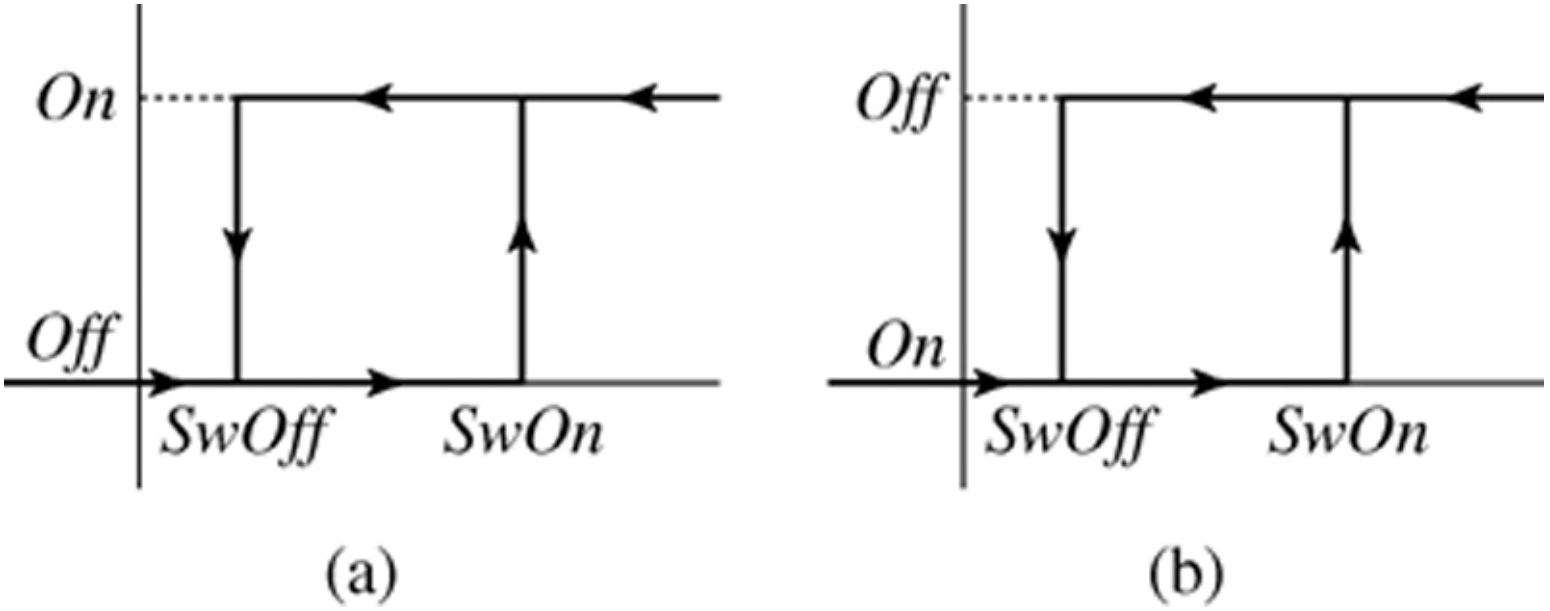


Speed response of the sled for $\theta = 0$ and $\theta \neq 0$. Figure 10.4–6 on page 434.



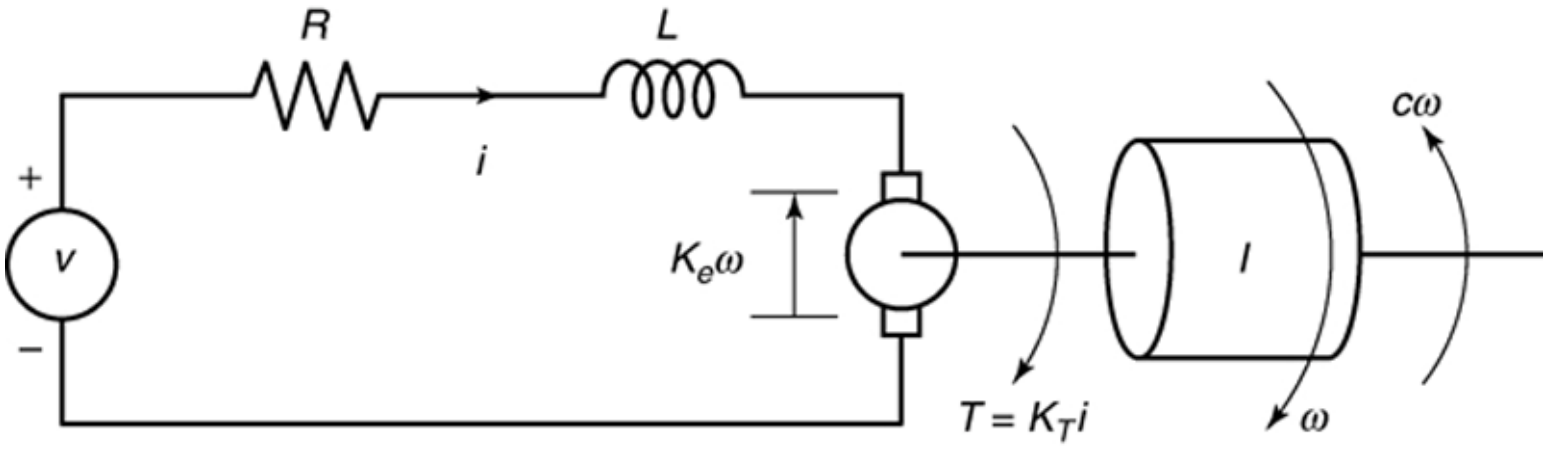
10-20

The relay function. (a) The case where *On* > *Off*. (b) The case where *On* < *Off*. Figure 10.4–7 on page 434.



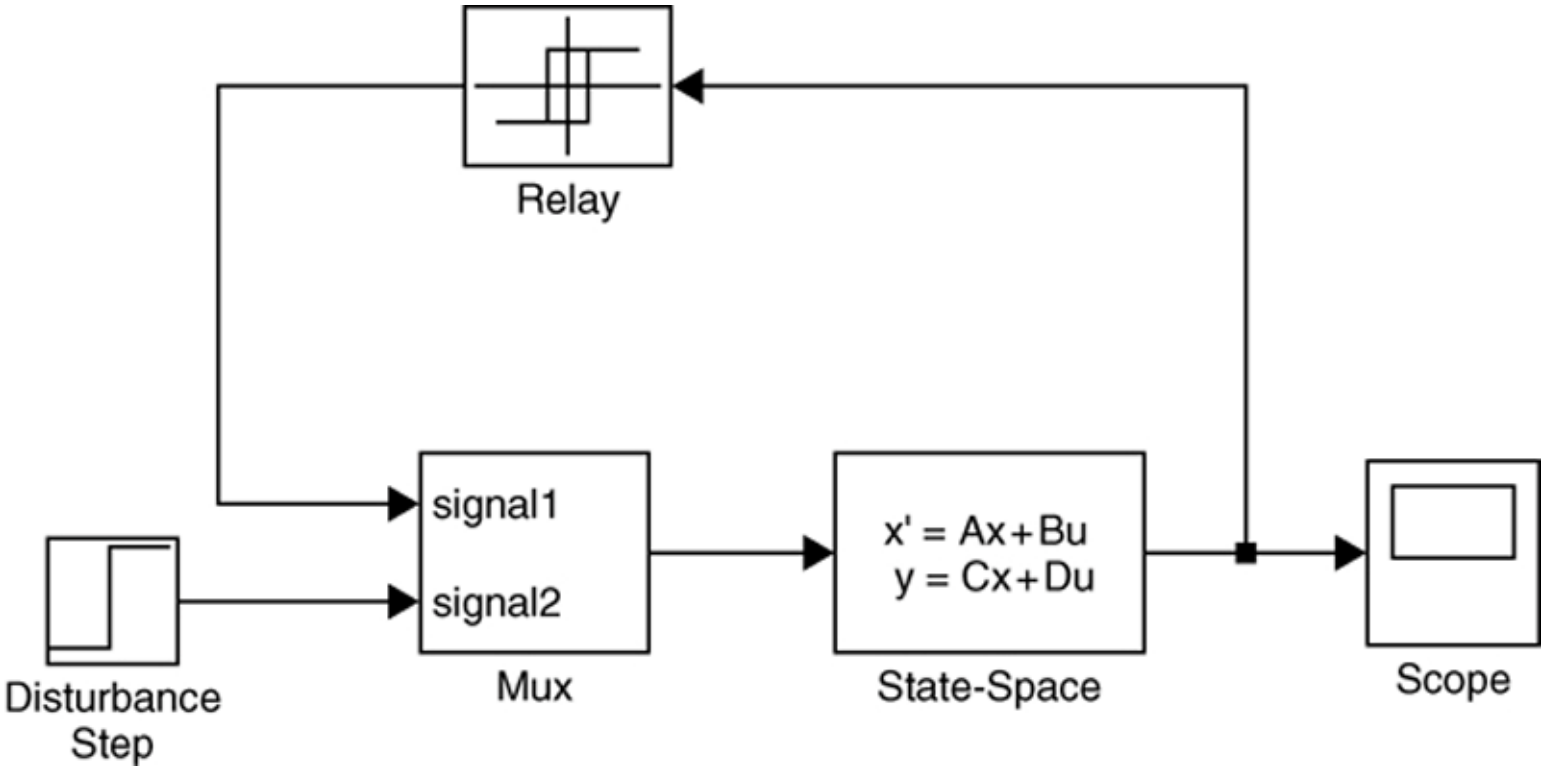
10-21

An armature-controlled dc motor, Example 10.4-2. Figure 10.4–8 on page 435.



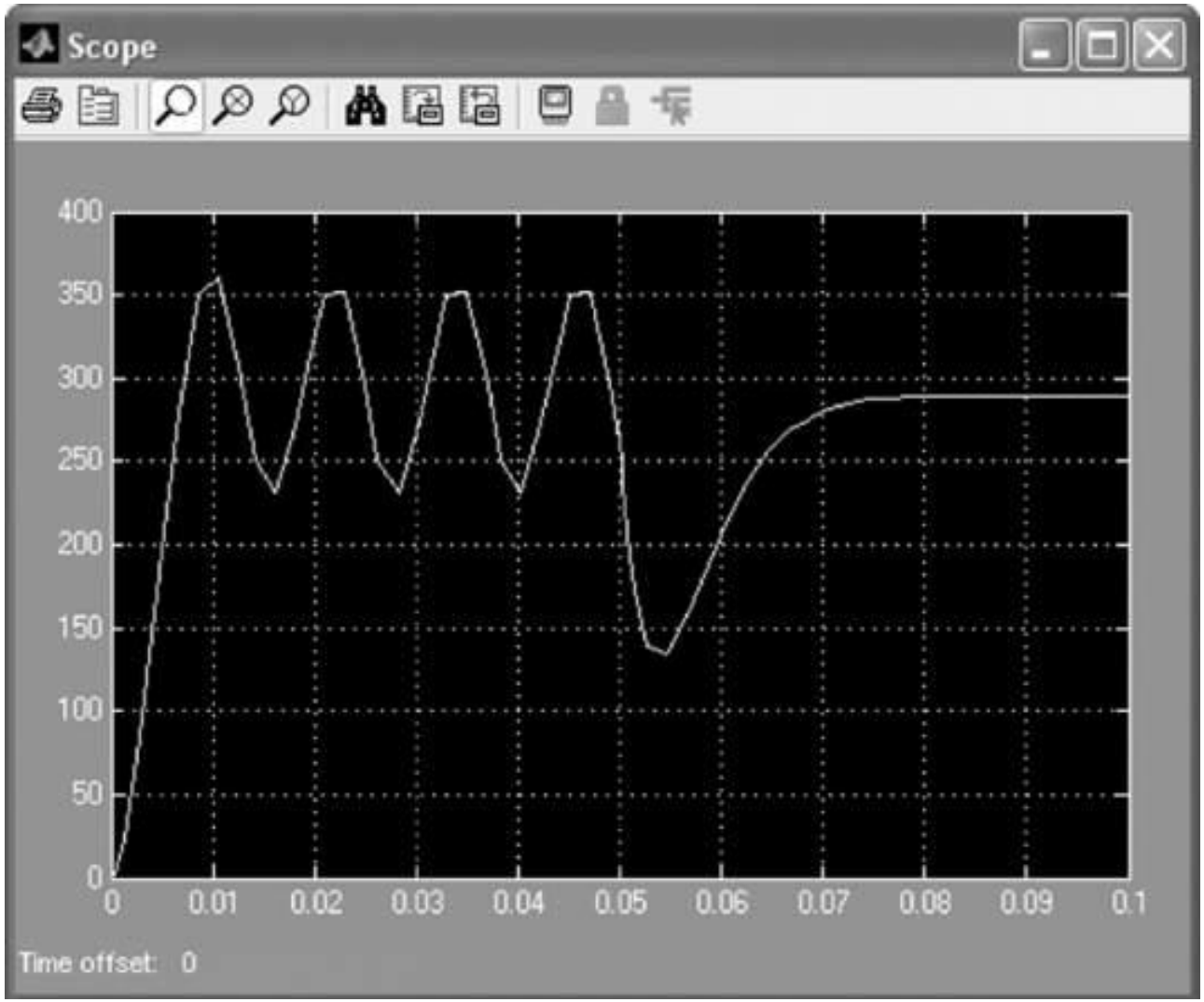
10-22

Simulink model of a relay-controlled motor. Figure 10.4–9 on page 436.



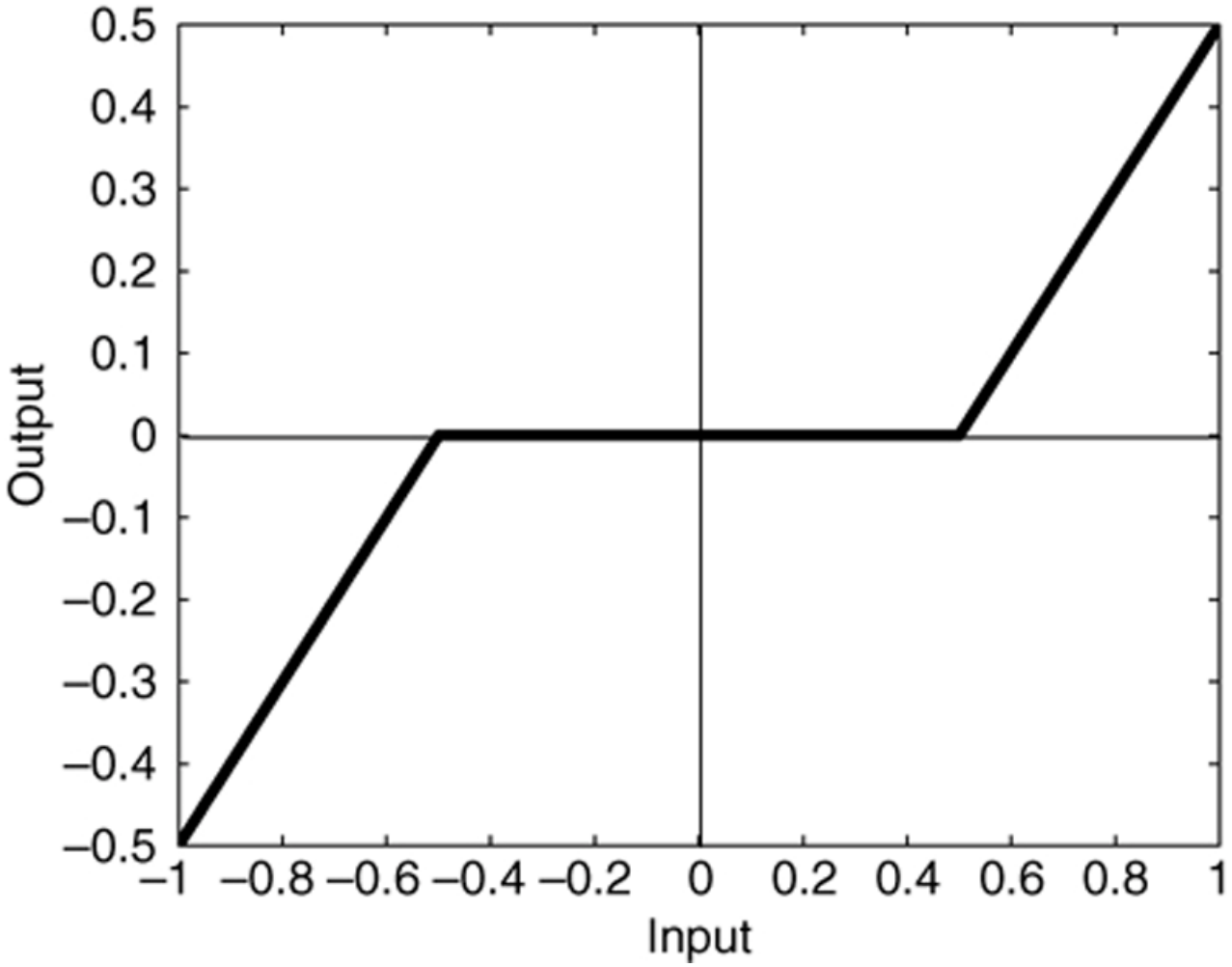
For the steps needed to construct this model, see Example 10.4-2 on pages 434-437.

Scope display of the speed response of a relay-controlled motor. Figure 10.4–10 on page 437.



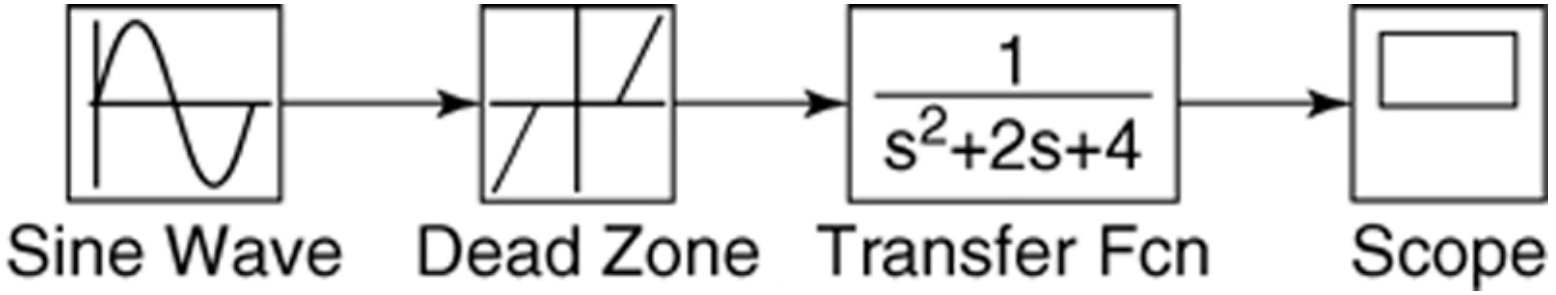
10-24

A dead-zone nonlinearity. Figure 10.5–1 on page 438.



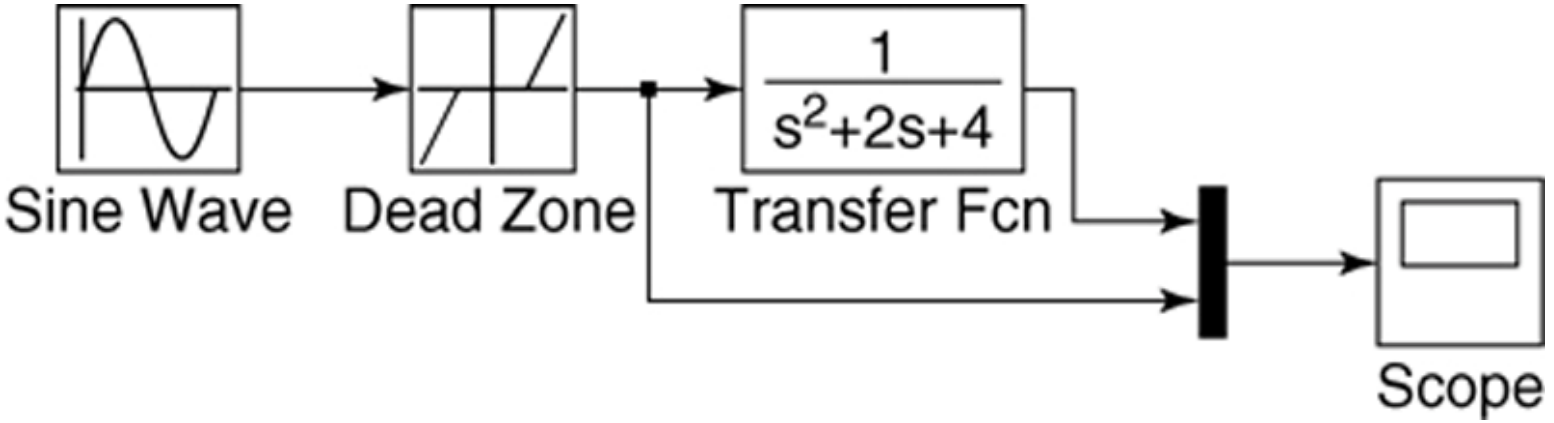
10-25

The Simulink model of dead-zone response. Figure 10.5–2



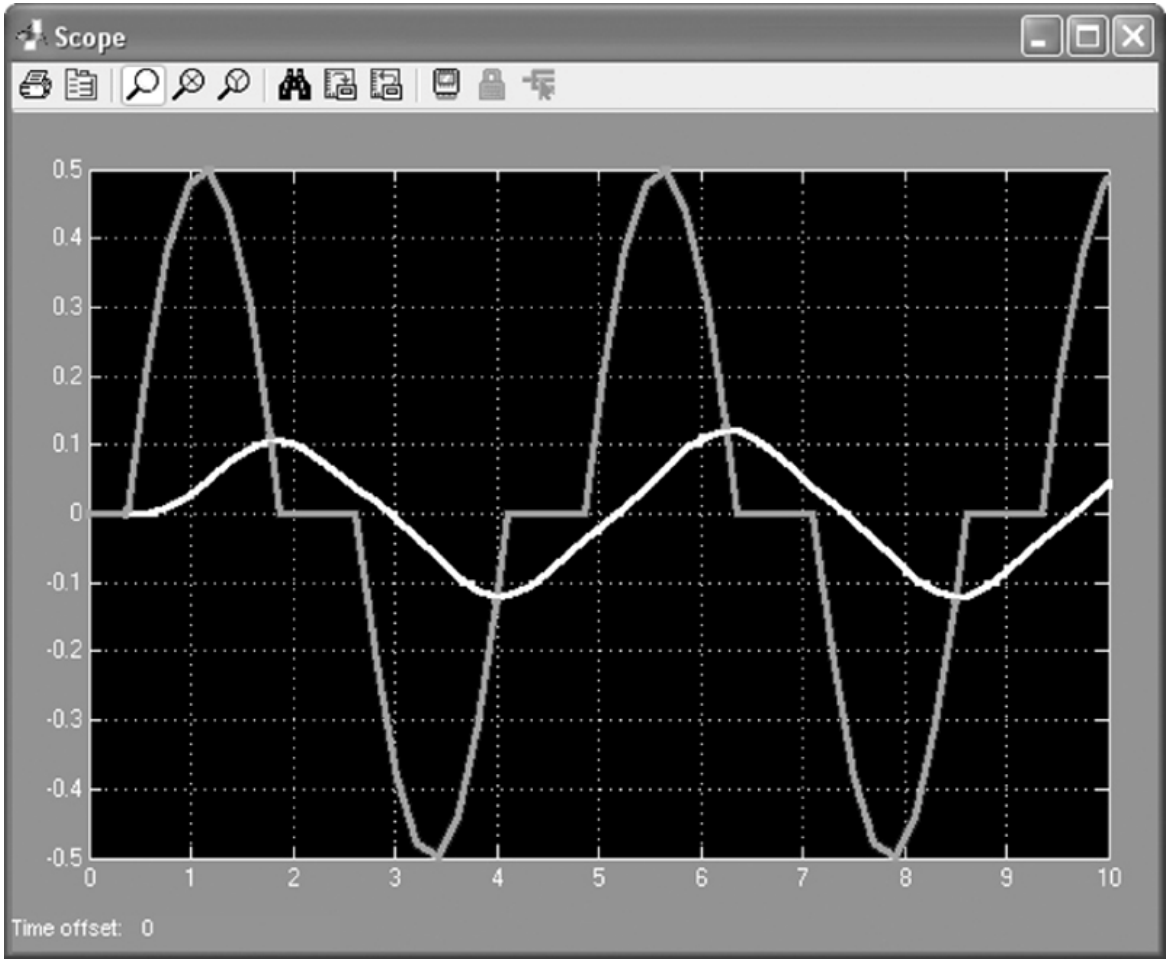
For the steps needed to construct this model, see Example 10.5-1 on pages 438-439.

Modification of the dead-zone model to include a Mux block.
Figure 10.5–3 on page 439.



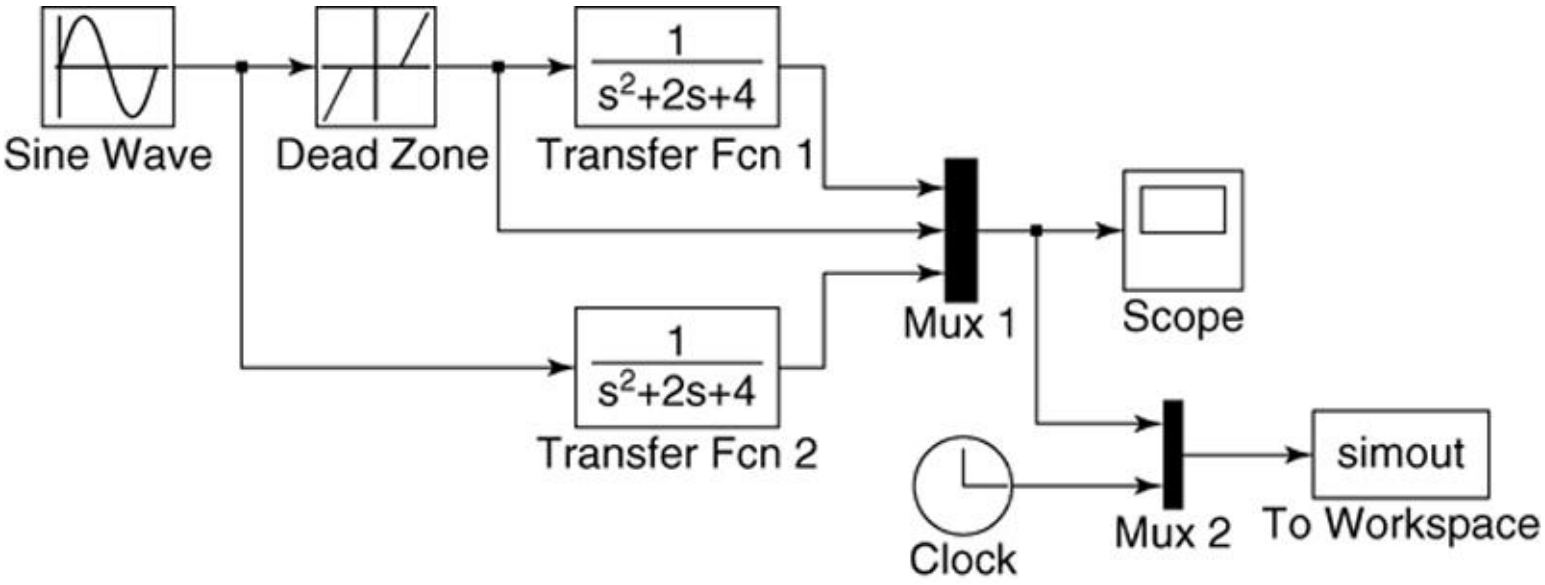
For the steps needed to construct this model, see Example 10.5-1 on page 439.

The response of the dead-zone model. Figure 10.5–4 on page 440.



10-28

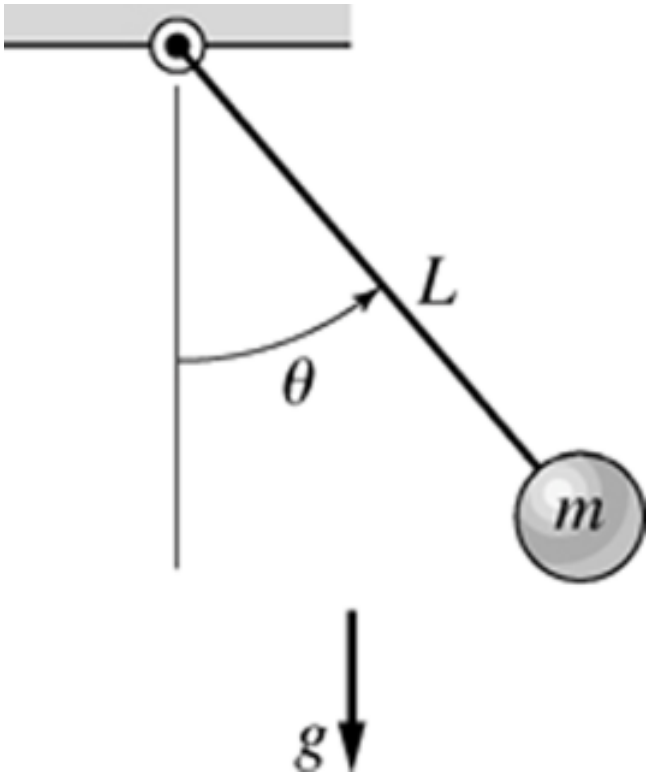
Modification of the dead-zone model to export variables to the MATLAB workspace. Figure 10.5–5 on page 440.



For the steps needed to construct this model, see pages 440-441.

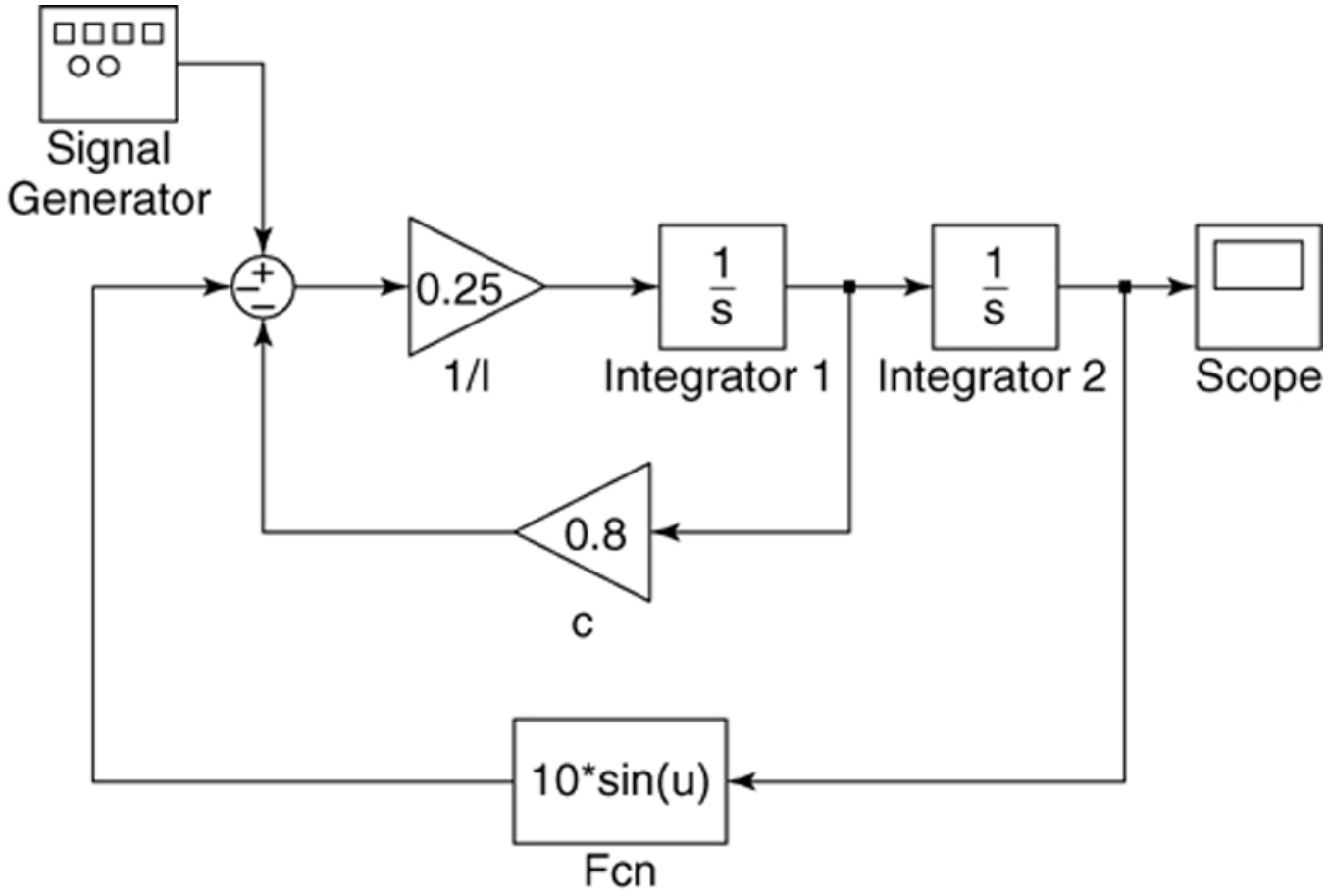
10-29

A pendulum. Figure 10.6–1 on page 441.



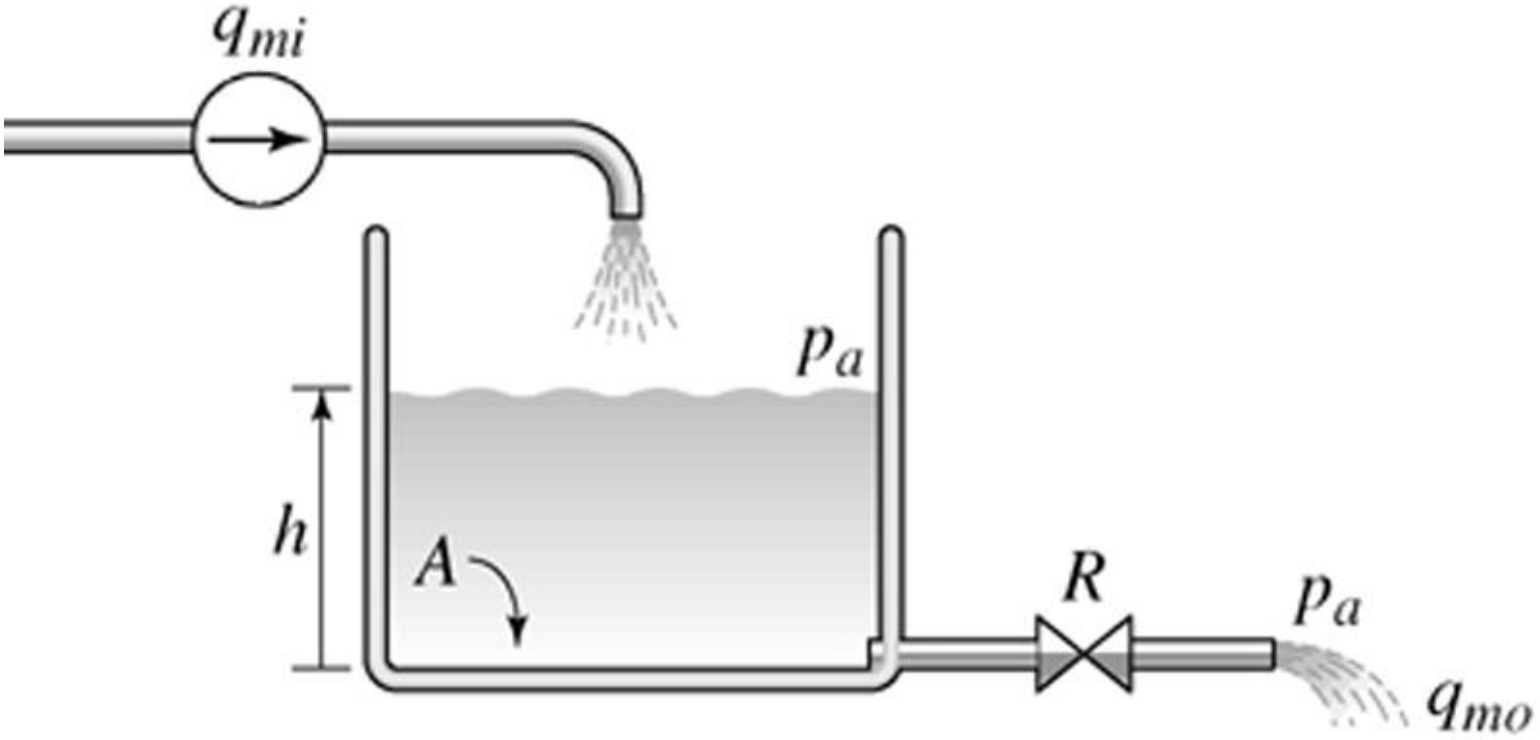
10-30

Simulink model of nonlinear pendulum dynamics. Figure 10.6–2 on page 442.



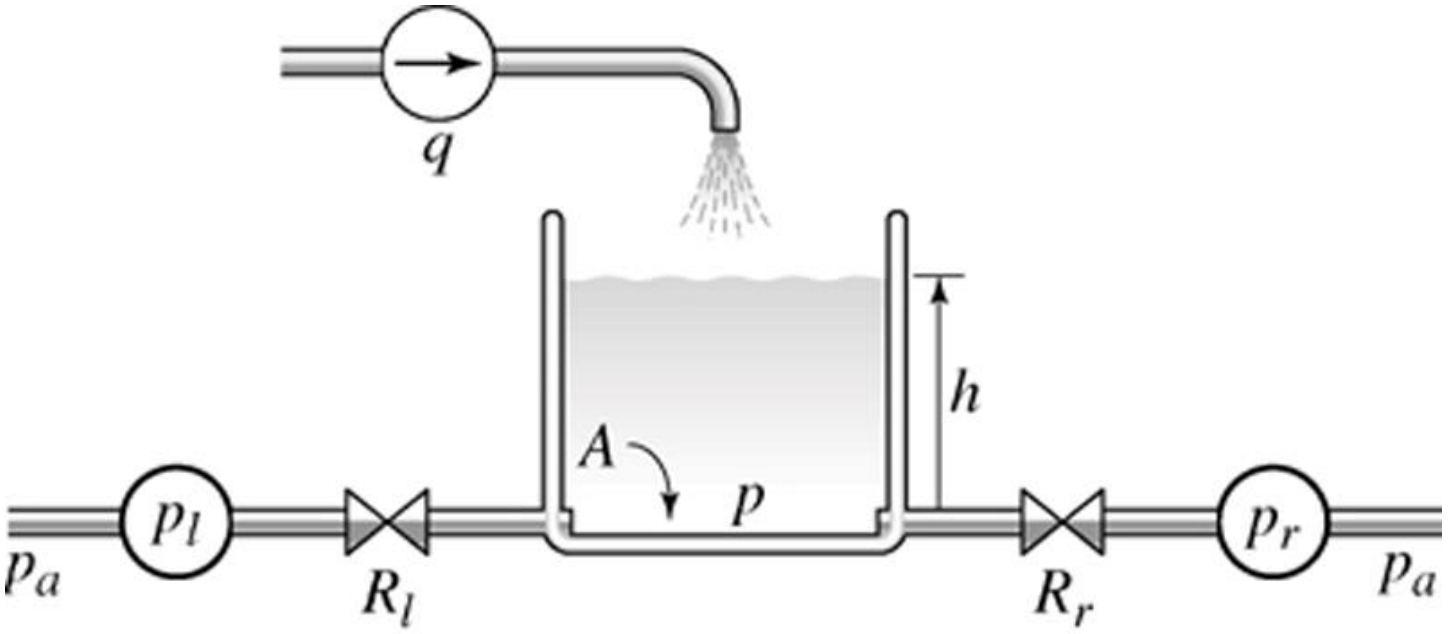
For the steps needed to construct this model, see pages 442-443.

Subsystem blocks. A hydraulic system with a flow source.
Figure 10.7–1 on page 443.



This model is discussed on pages 443-444.

A hydraulic system with a flow source and two pumps. Figure 10.7–2 on page 444.



This model is discussed on pages 444-445.

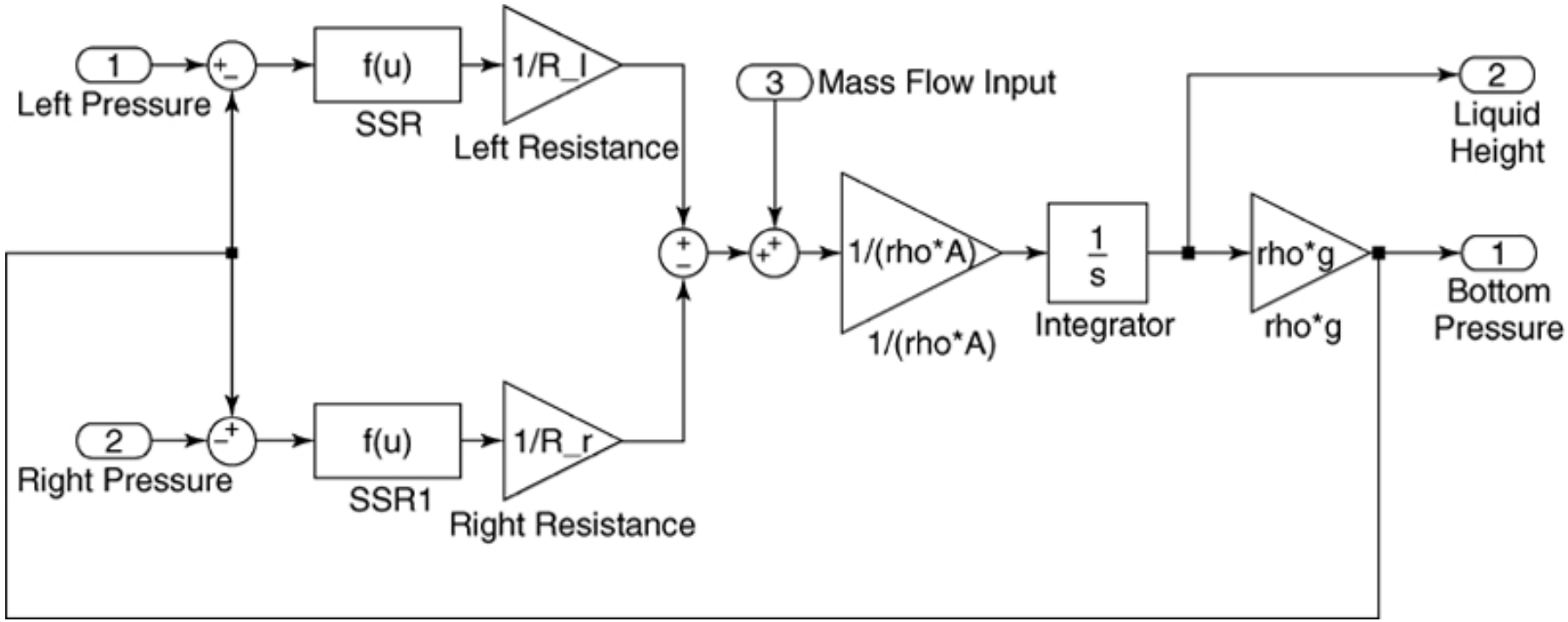
10-33

You can create a subsystem block in one of two ways:

1. by dragging the Subsystem block from the library to the model window, or

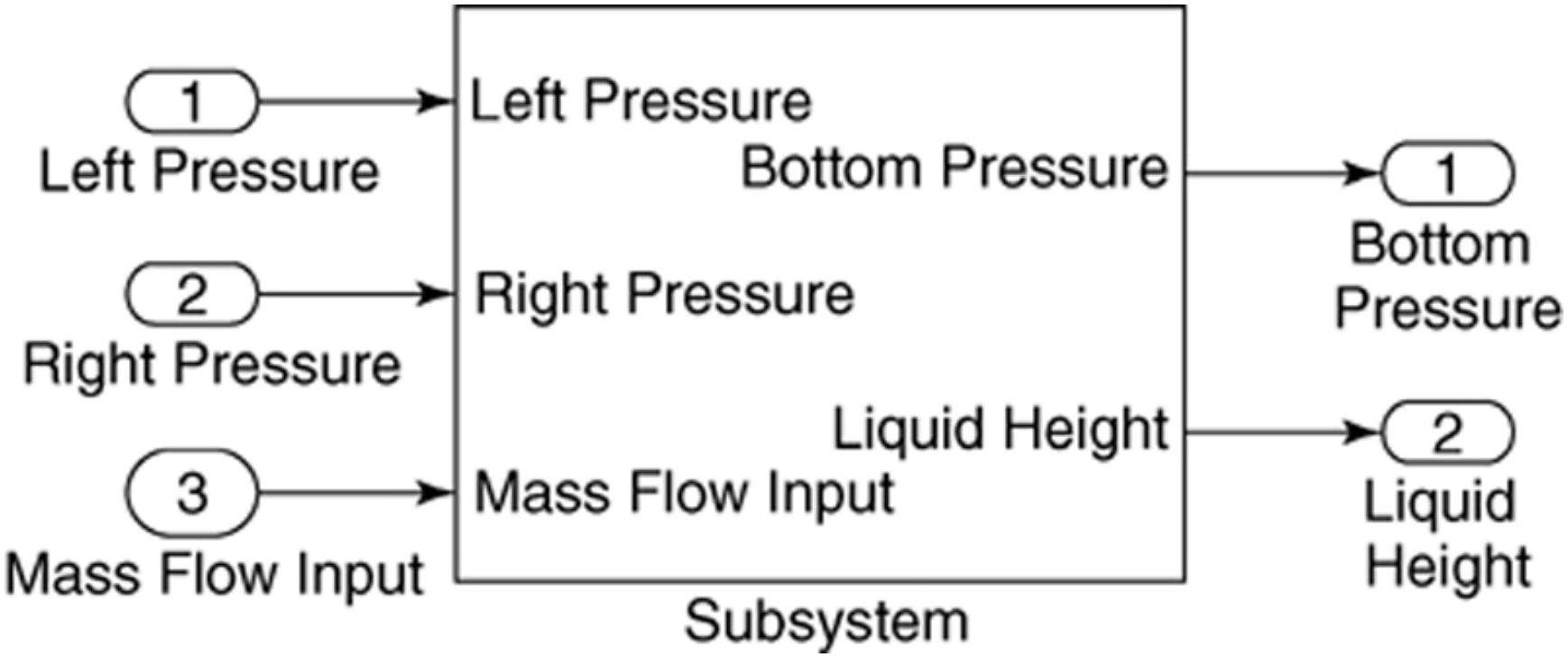
2. by first creating a Simulink model and then “encapsulating” it within a bounding box.

Simulink model of the hydraulic system shown in Figure 10.7–2. Figure 10.7–3 on page 445.



10-35

The Subsystem block. Figure 10.7–4 on page 446.



10-36

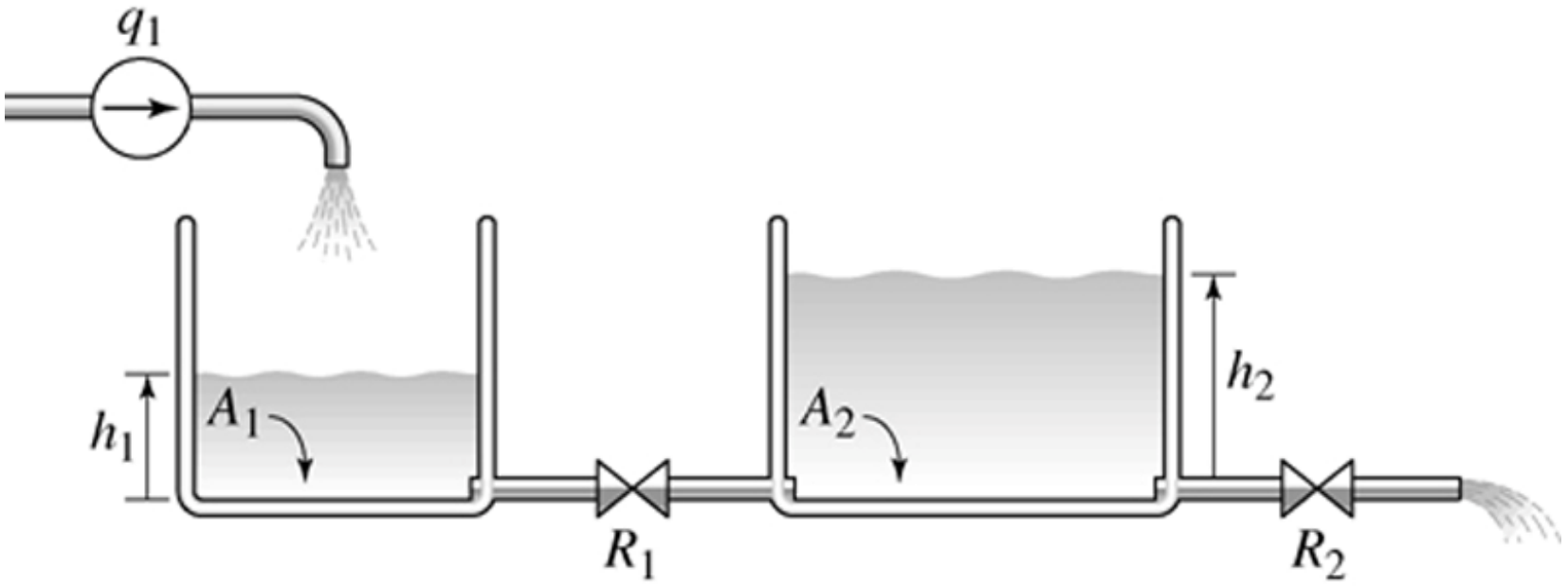
To create a subsystem block, create a “bounding box” surrounding the diagram.

Do this by placing the mouse cursor in the upper left, holding the mouse button down, and dragging the expanding box to the lower right to enclose the entire diagram.

Then choose **Create Subsystem** from the **Edit** menu. Simulink will then replace the diagram with a single block having as many input and output ports as required and will assign default names.

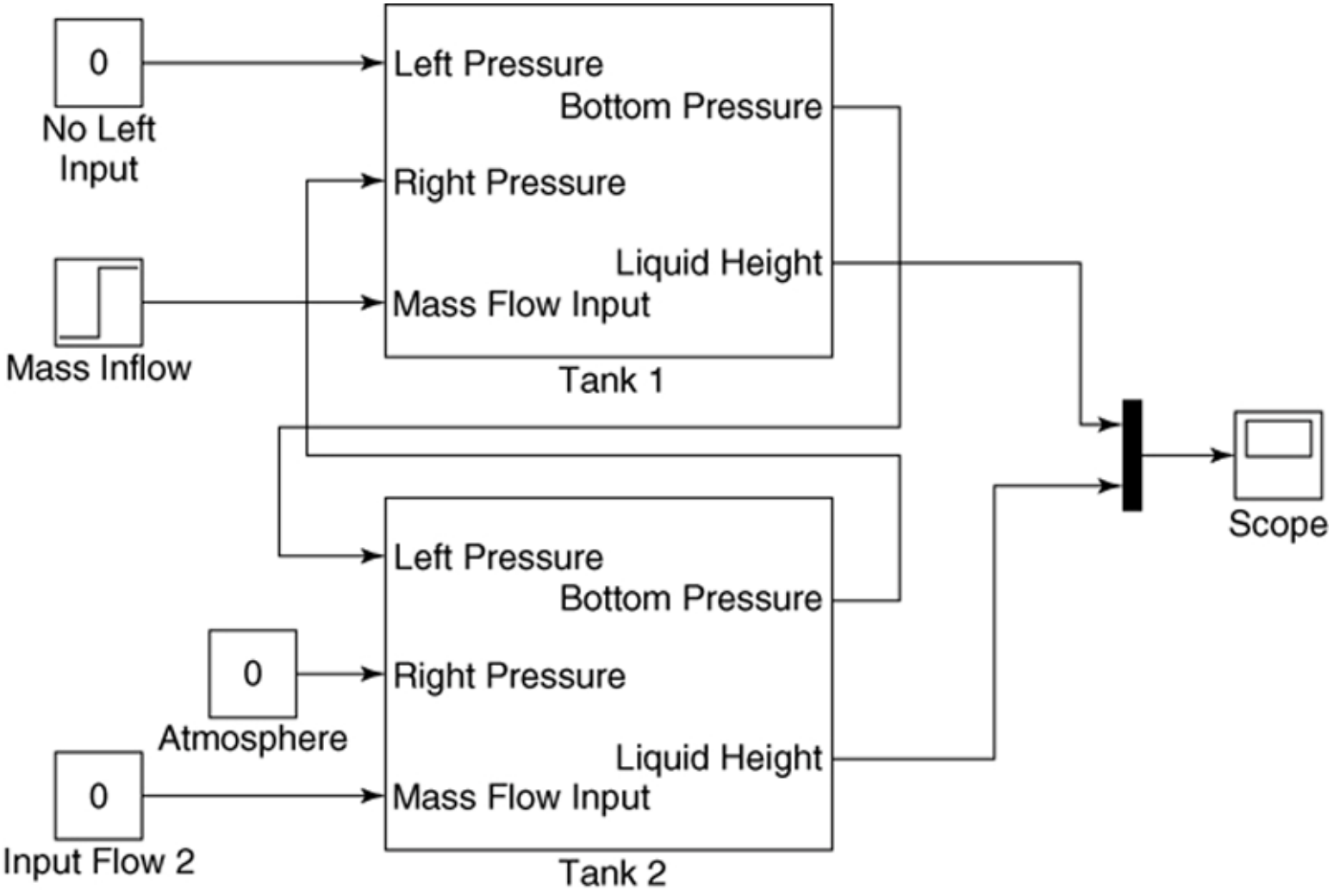
You can resize the block to make the labels readable. You may view or edit the subsystem by double-clicking on it.

A hydraulic system with two tanks. Figure 10.7–5 on page 446.



10-38

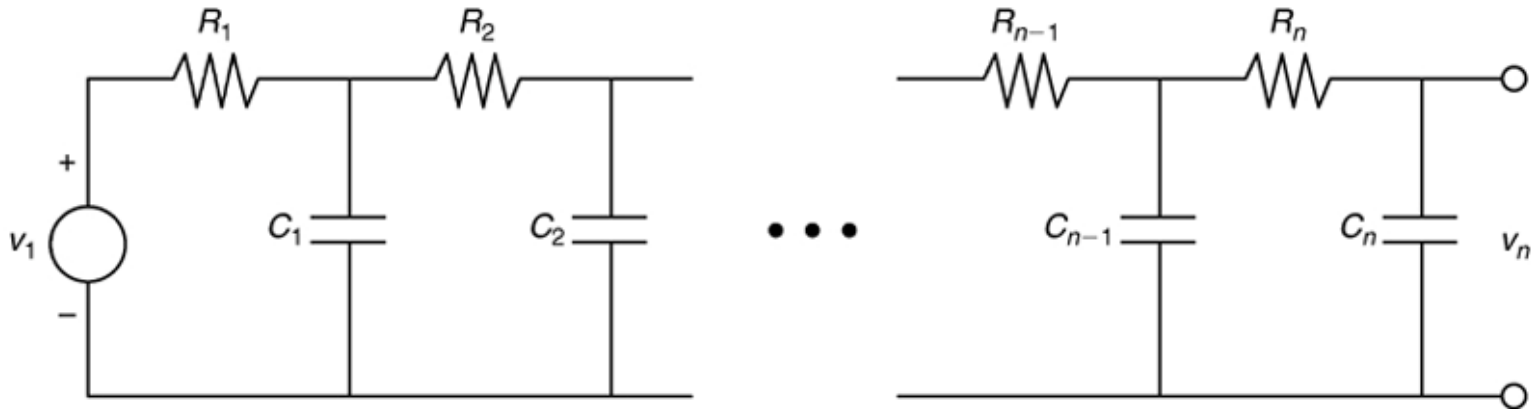
Simulink model of the system shown in Figure 10.7–5. Figure 10.7–6 on page 447.



More? See pages 445-447.

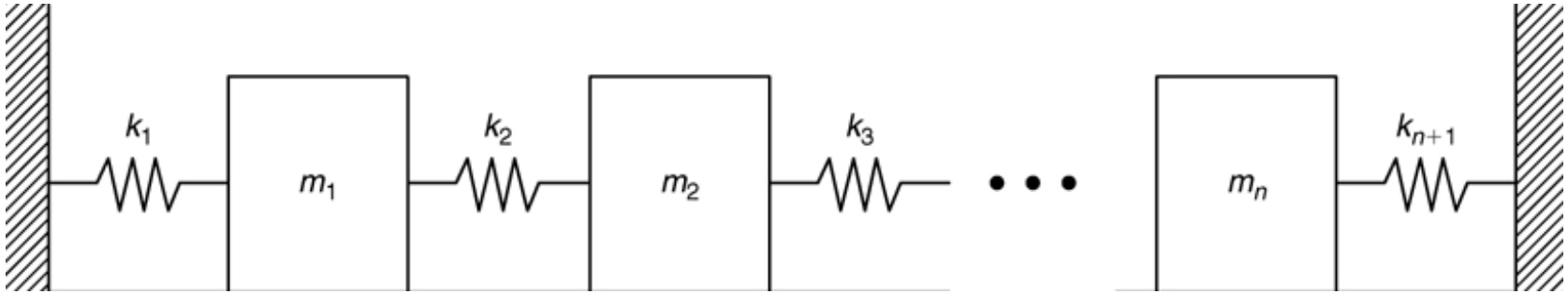
Other applications of subsystem blocks.

A network of RC loops. Figure 10.7–7 on page 448.



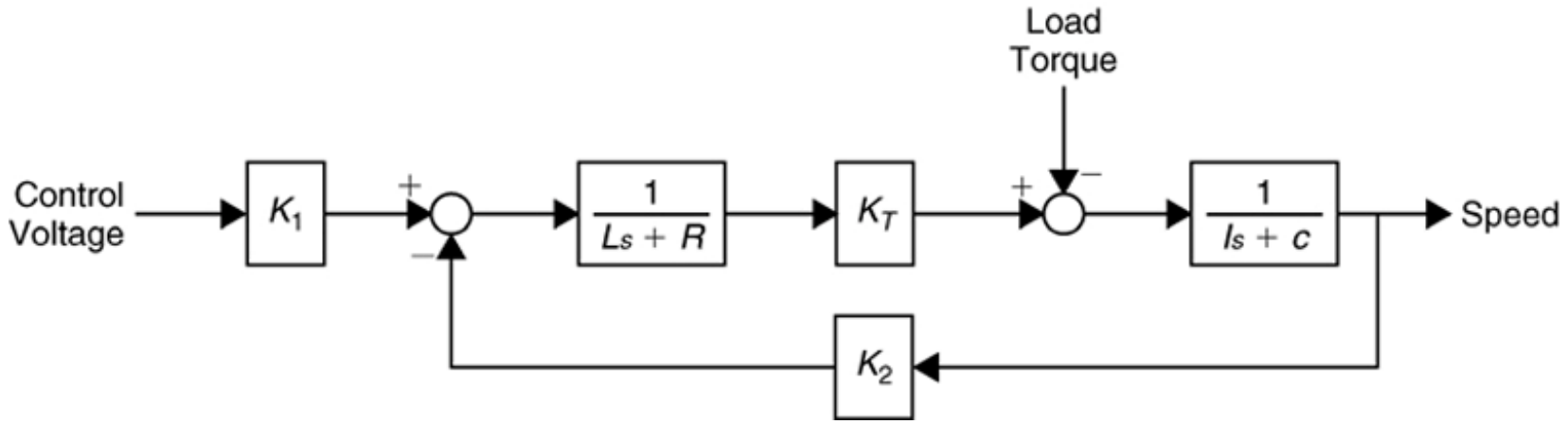
10-40

A vibrating system. Figure 10.7–8 on page 448.



10-41

An armature-controlled dc motor. Figure 10.7–9 on page 448.



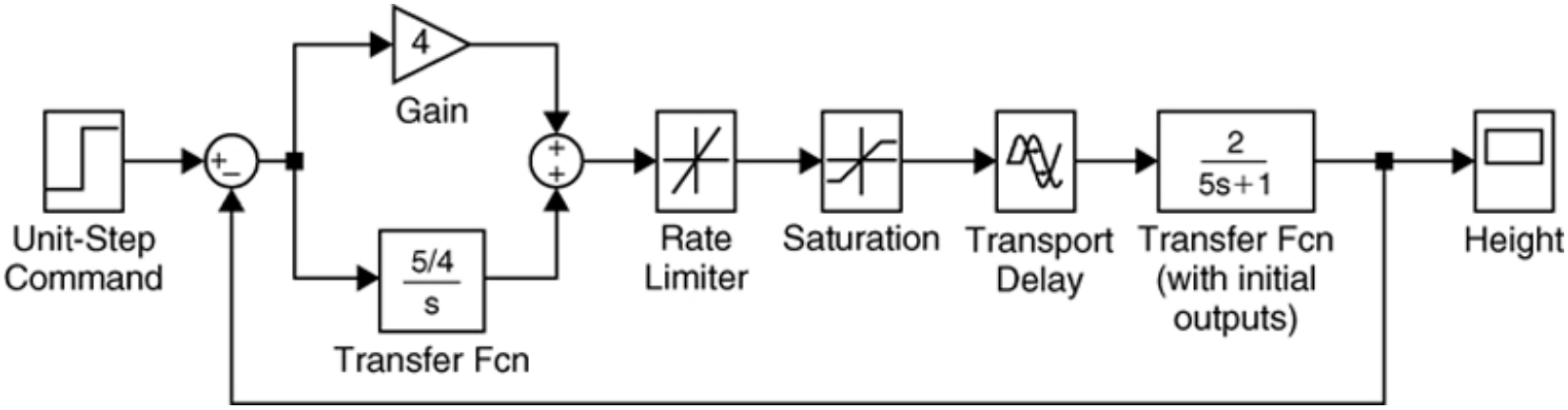
10-42

The “Transfer Fcn (with initial outputs)” block

The “Transfer Fcn (with initial outputs)” block, so-called to distinguish it from the Transfer Fcn block, enables us to set the initial value of the block output.

The “Transfer Fcn (with initial outputs)” block is equivalent to adding the free response to the block output, with all the block’s state variables set to zero except for the output variable.

Simulink model of a hydraulic system with dead time.
Figure 10.8–1 on page 450.



More? See pages 448-451.

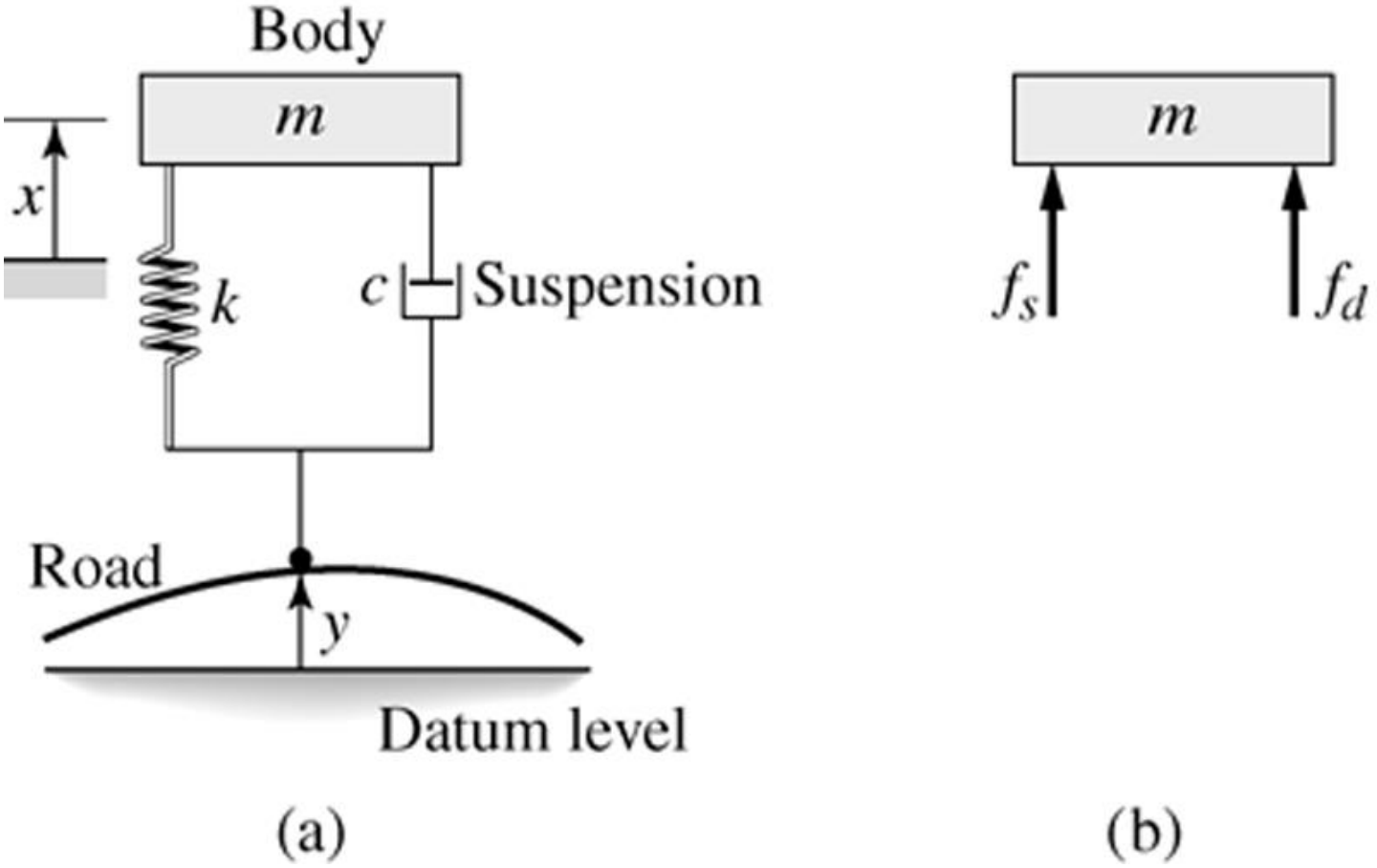
10-44

Four additional Simulink elements that enable us to model a wide range of nonlinearities and input functions, namely,

- the Derivative block,
- the Signal Builder block,
- the Look-Up Table block, and
- the MATLAB Fcn block.

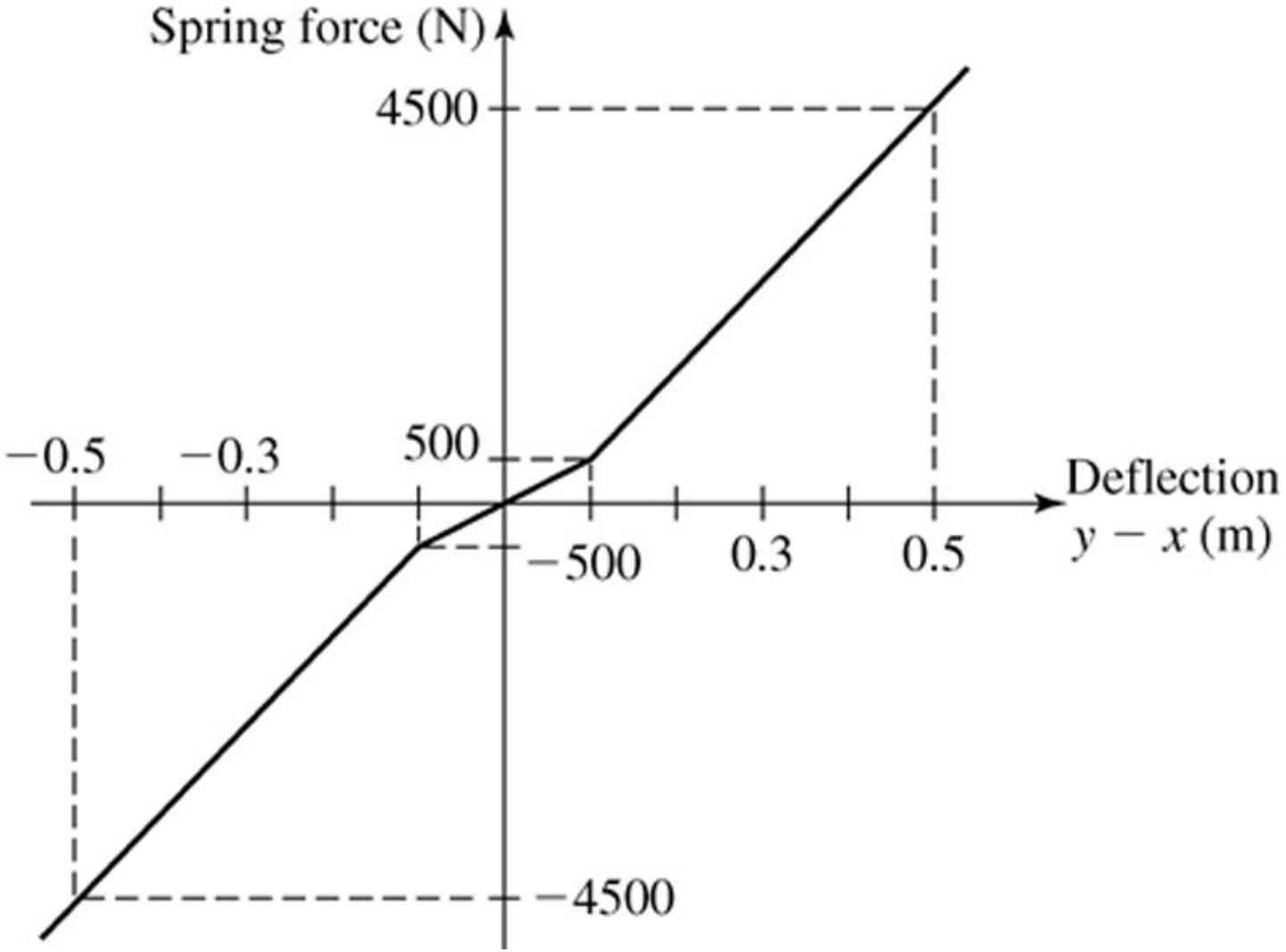
- These are discussed in Section 10.9 along with a simulation of a nonlinear vehicle suspension model (pages 451-455).

Single-mass model of a vehicle suspension. Figure 10.9–1 on page 451.



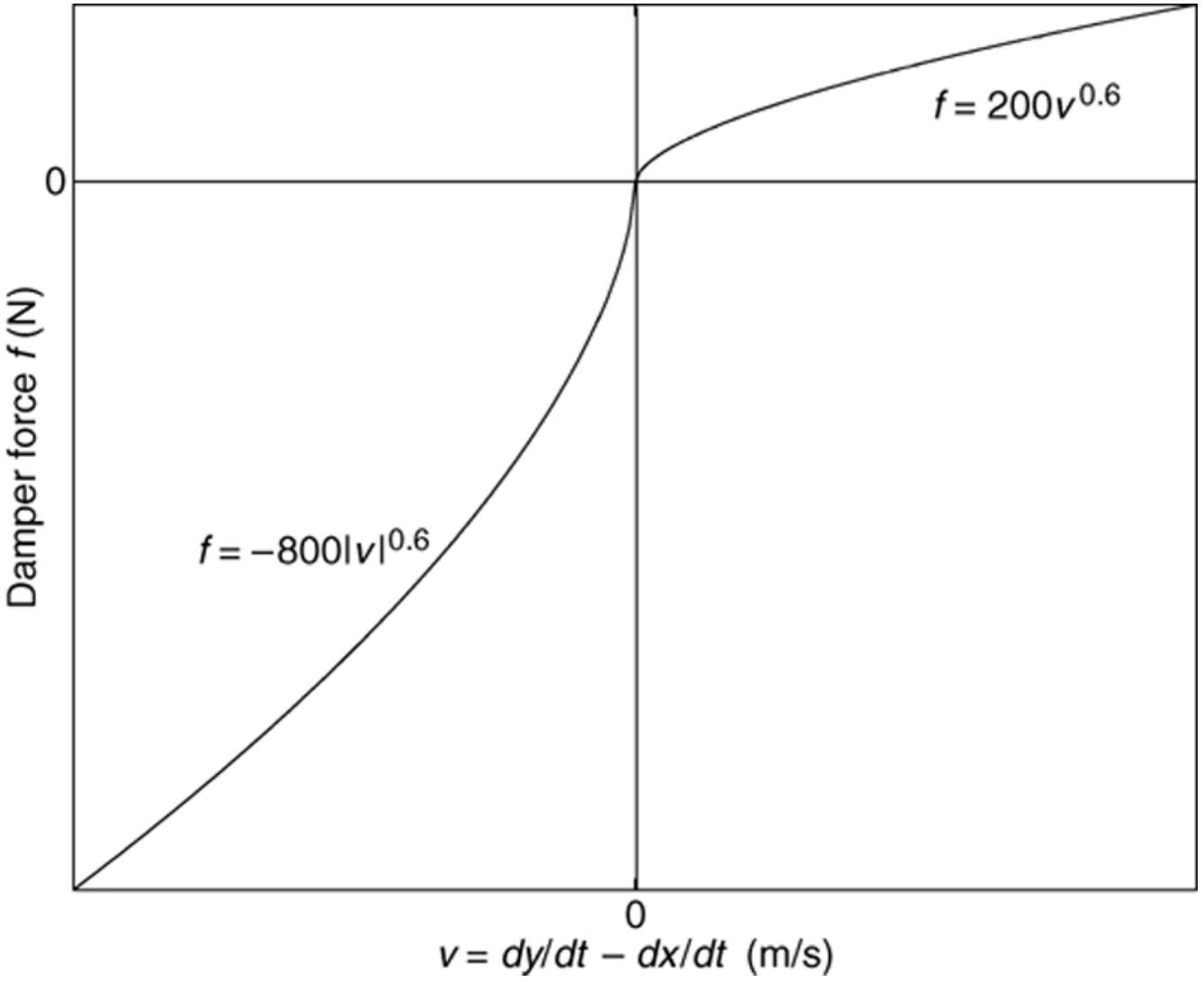
10-46

Nonlinear spring function. Figure 10.9–2 on page 452.

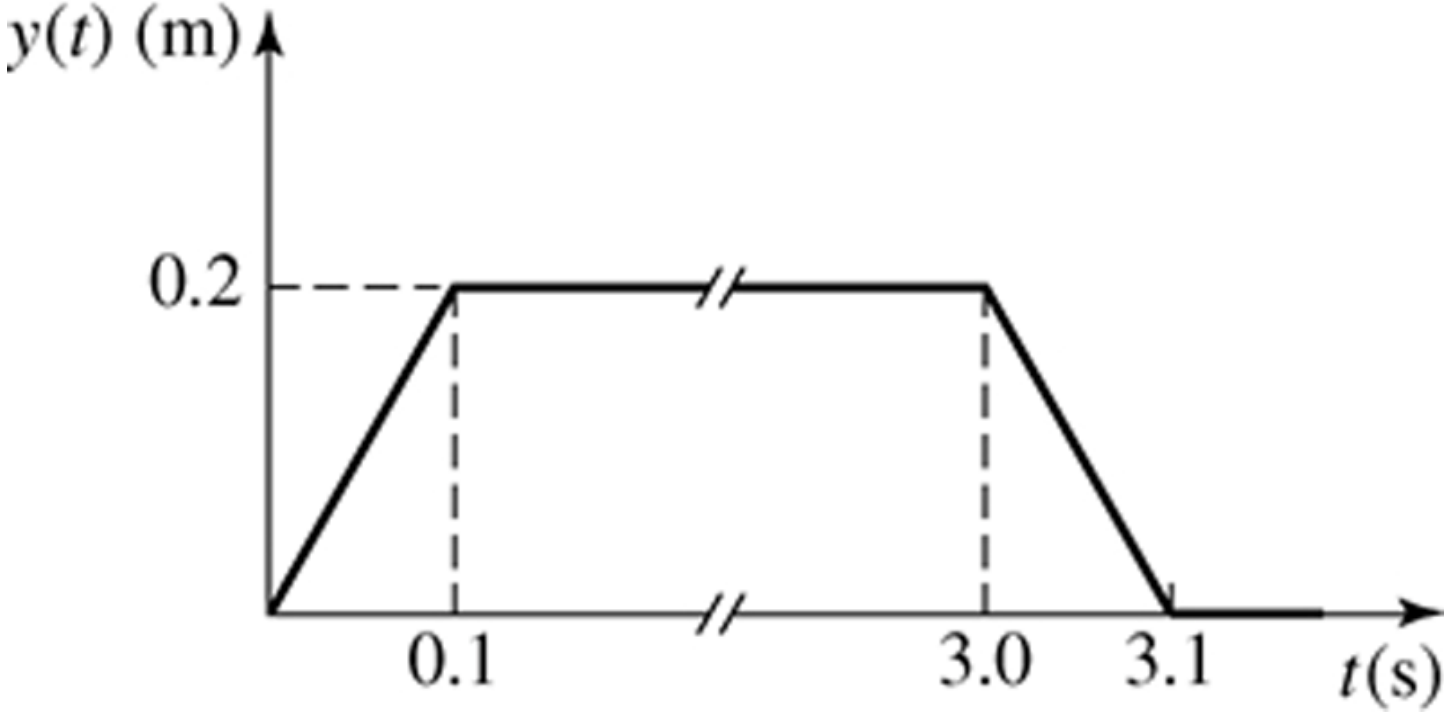


10-47

Nonlinear damping function. Figure 10.9–3 on page 452.

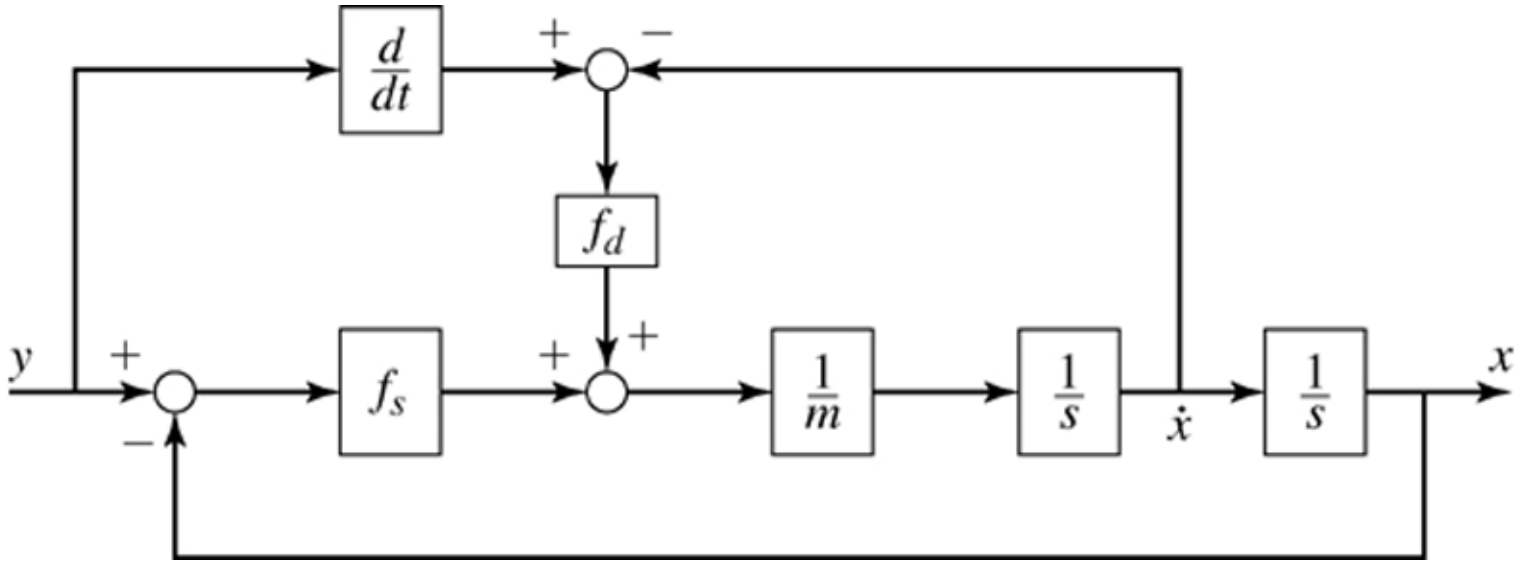


Road surface profile. Figure 10.9–4 on page 452.



10-49

Simulation diagram of a vehicle suspension model. Figure 10.9–5 on page 453.



10-50

The simulation diagram shows that we need to compute the derivative *dydt*.

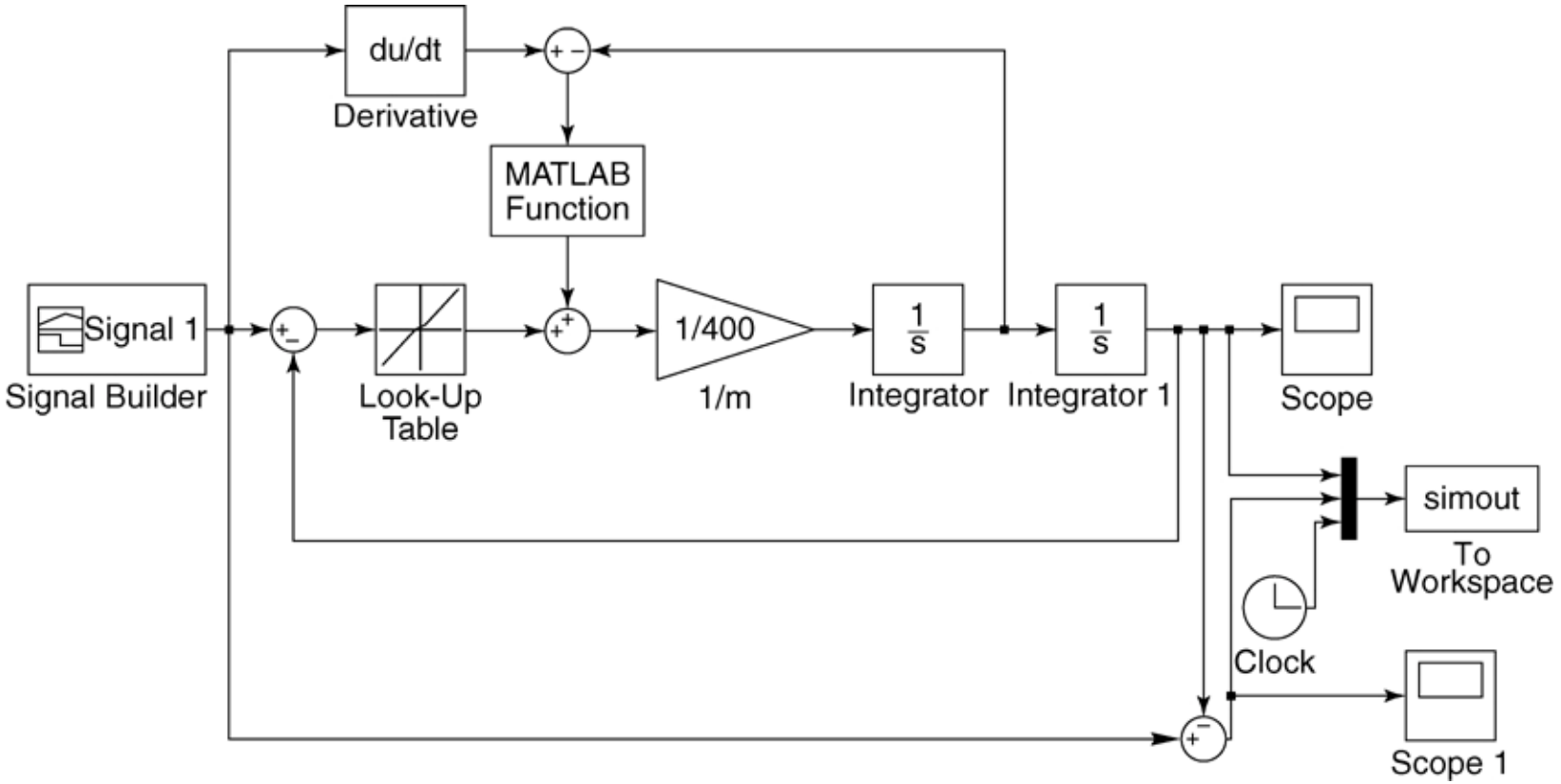
Because Simulink uses numerical and not analytical methods, it computes derivatives only approximately, using the Derivative block.

We must keep this approximation in mind when using rapidly changing or discontinuous inputs.

10-51

Simulink model of a vehicle suspension system.

Figure 10.9–6 on page 453.



10-52

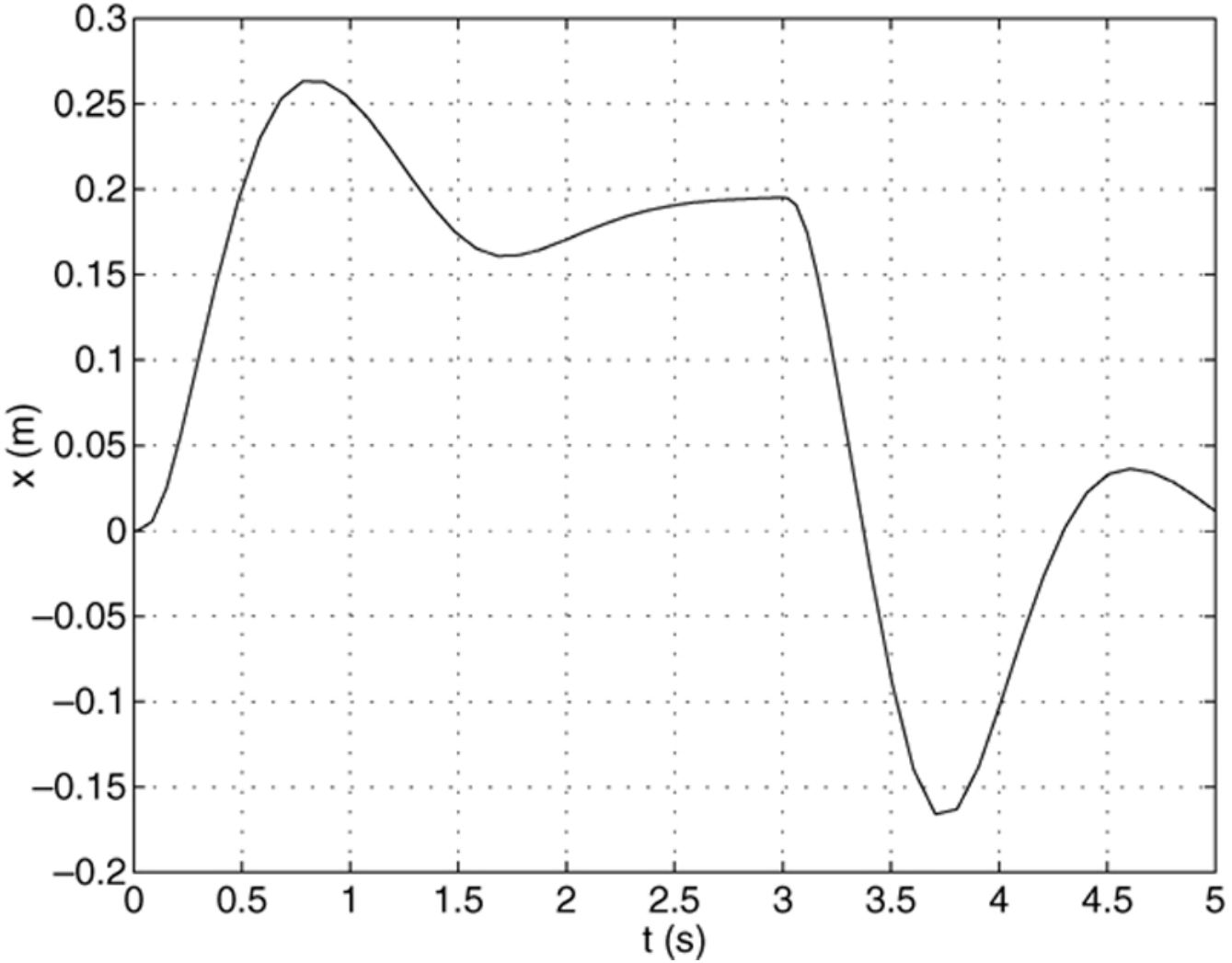
The Fcn, MATLAB Fcn, Math Function, and S-Function blocks can be used to implement functions, but each has its advantages and limitations.

The Fcn block can contain an expression, but its output must be a scalar, and it cannot call a function file.

The MATLAB Fcn block is slower than the Fcn block, but its output can be an array, and it can call a function file.

The Math Function block can produce an array output, but it is limited to a single MATLAB function and cannot use an expression or call a file. The S-Function block provides more advanced features, such as the ability to use C language code.

Output of the Simulink model shown in Figure 10.9–6. Figure 10.9–7 on page 455.

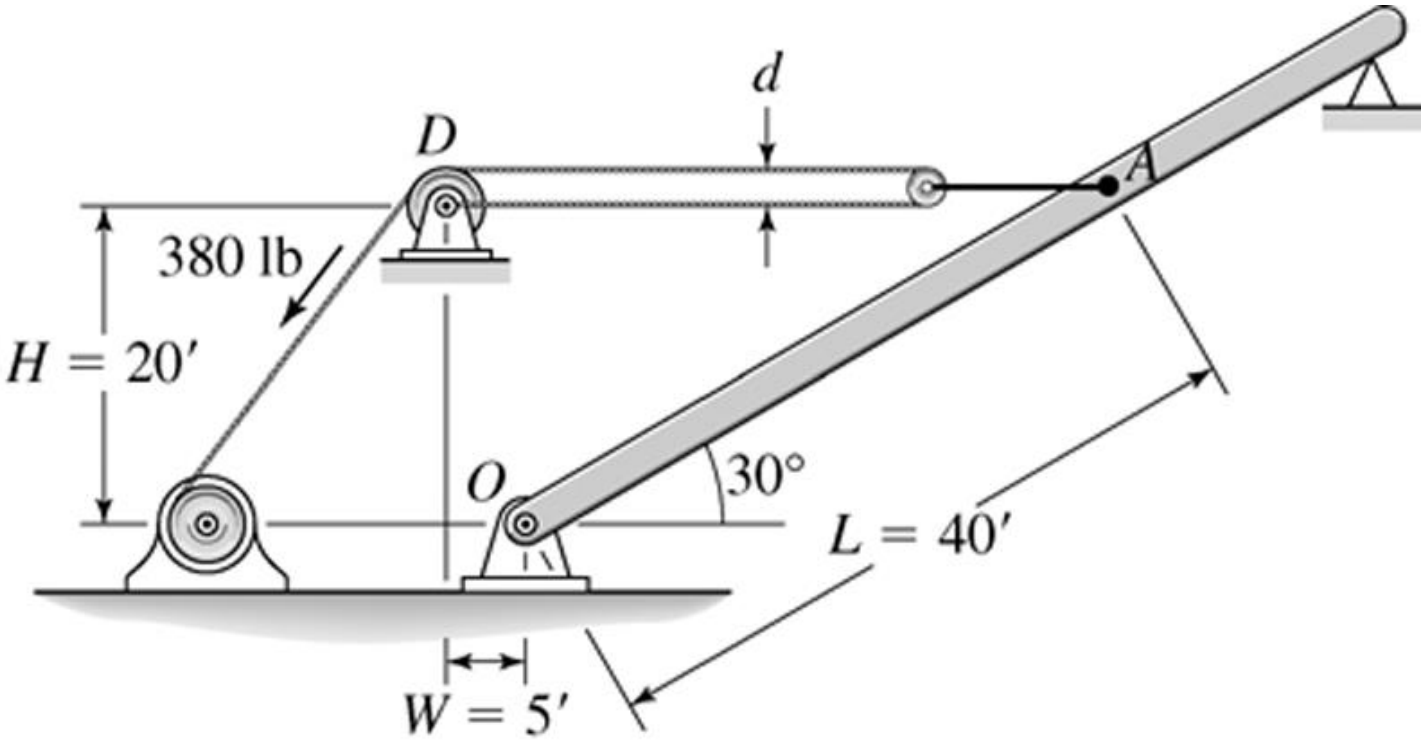


10-54

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

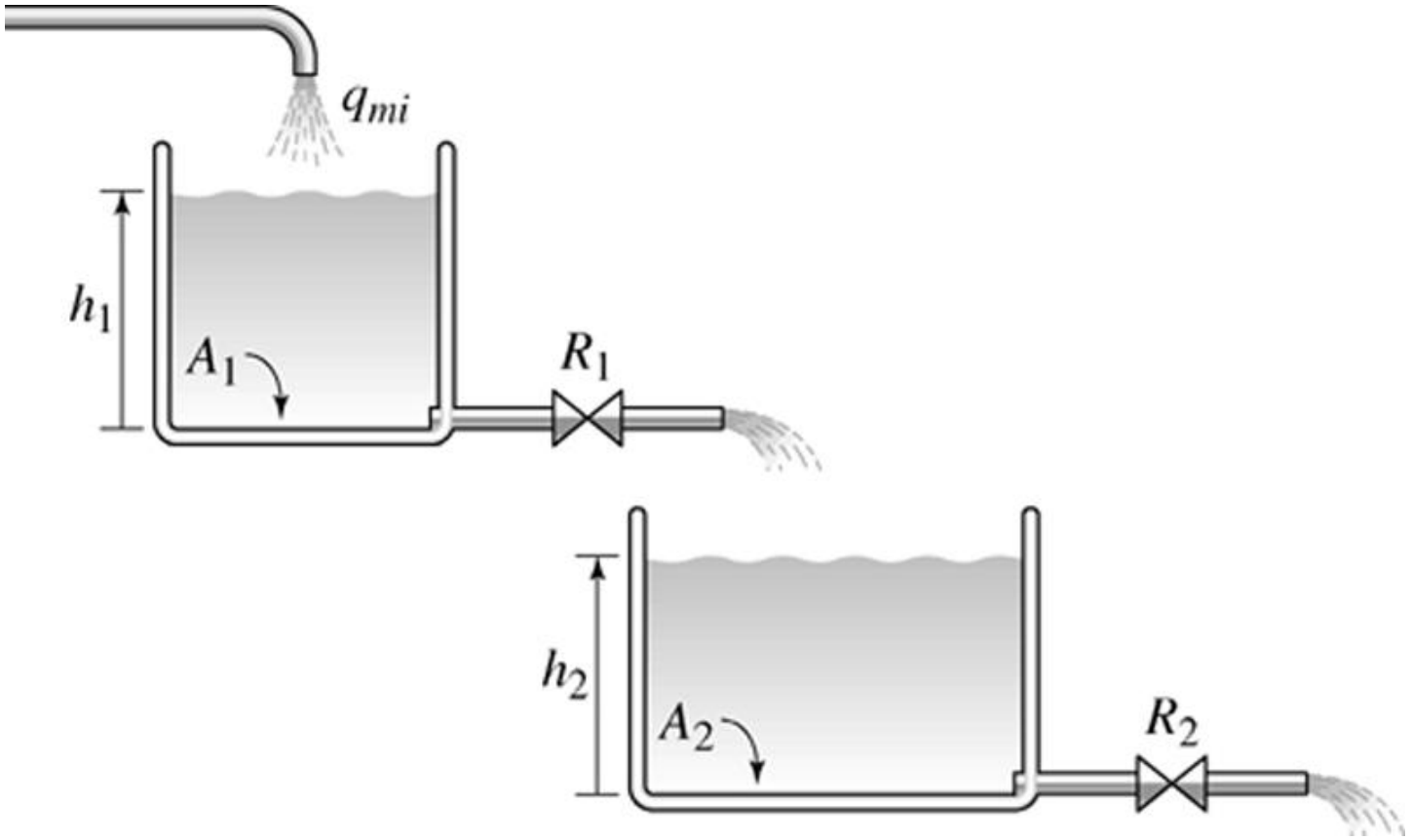
10-55

Figure P26 on page 460.



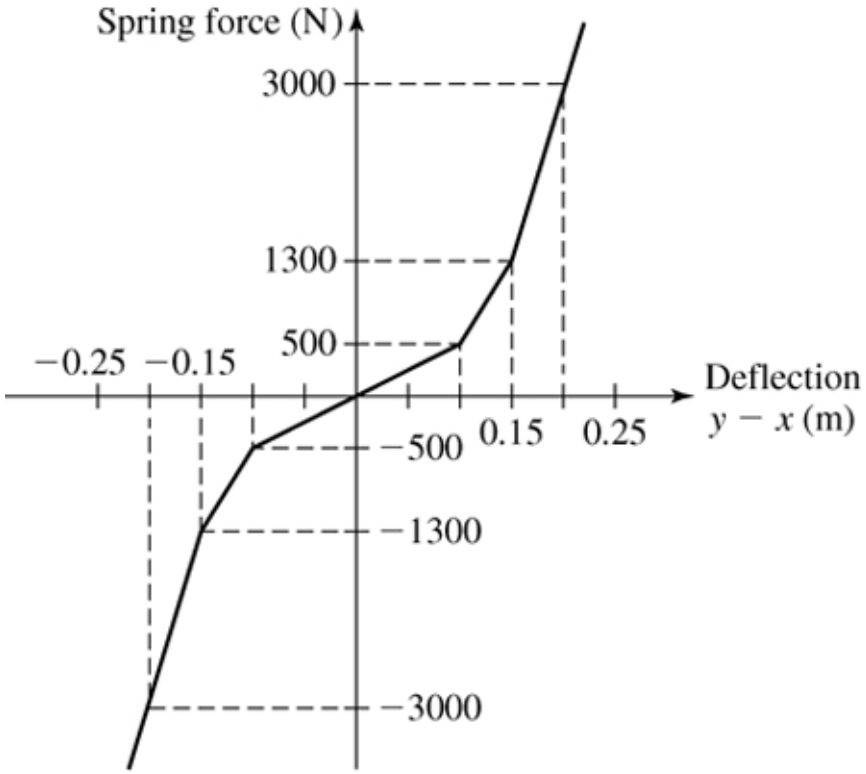
10-56

Figure P30 on page 462.

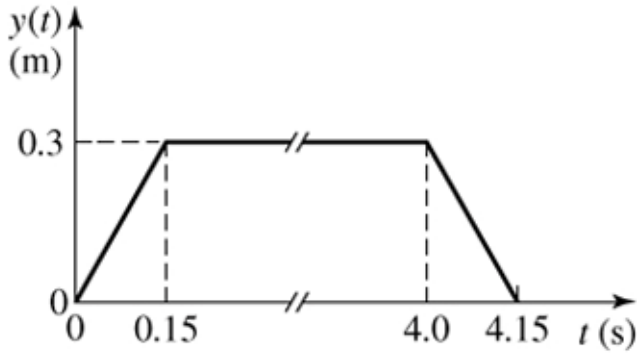


10-57

Figure P34 on page 463.



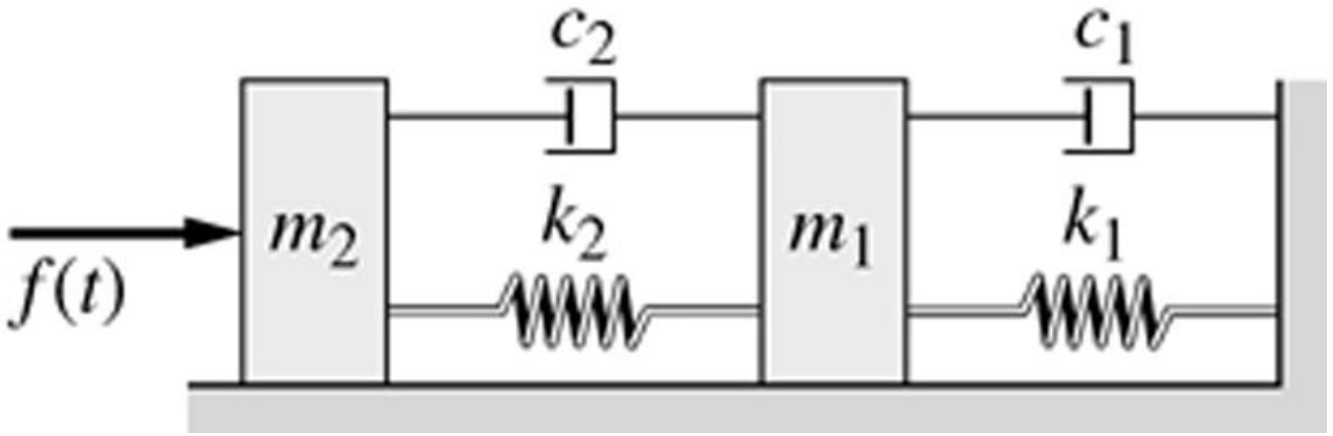
(a)



(b)

10-58

Figure P35 on page 463.



10-59