The McGraw-Hill Companies

PowerPoint to accompany

Introduction to MATLAB for Engineers, Third Edition

William J. Palm III

Chapter 9
Numerical Methods for Calculus and Differential Equations

Copyright © 2010. The McGraw-Hill Companies, Inc.

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

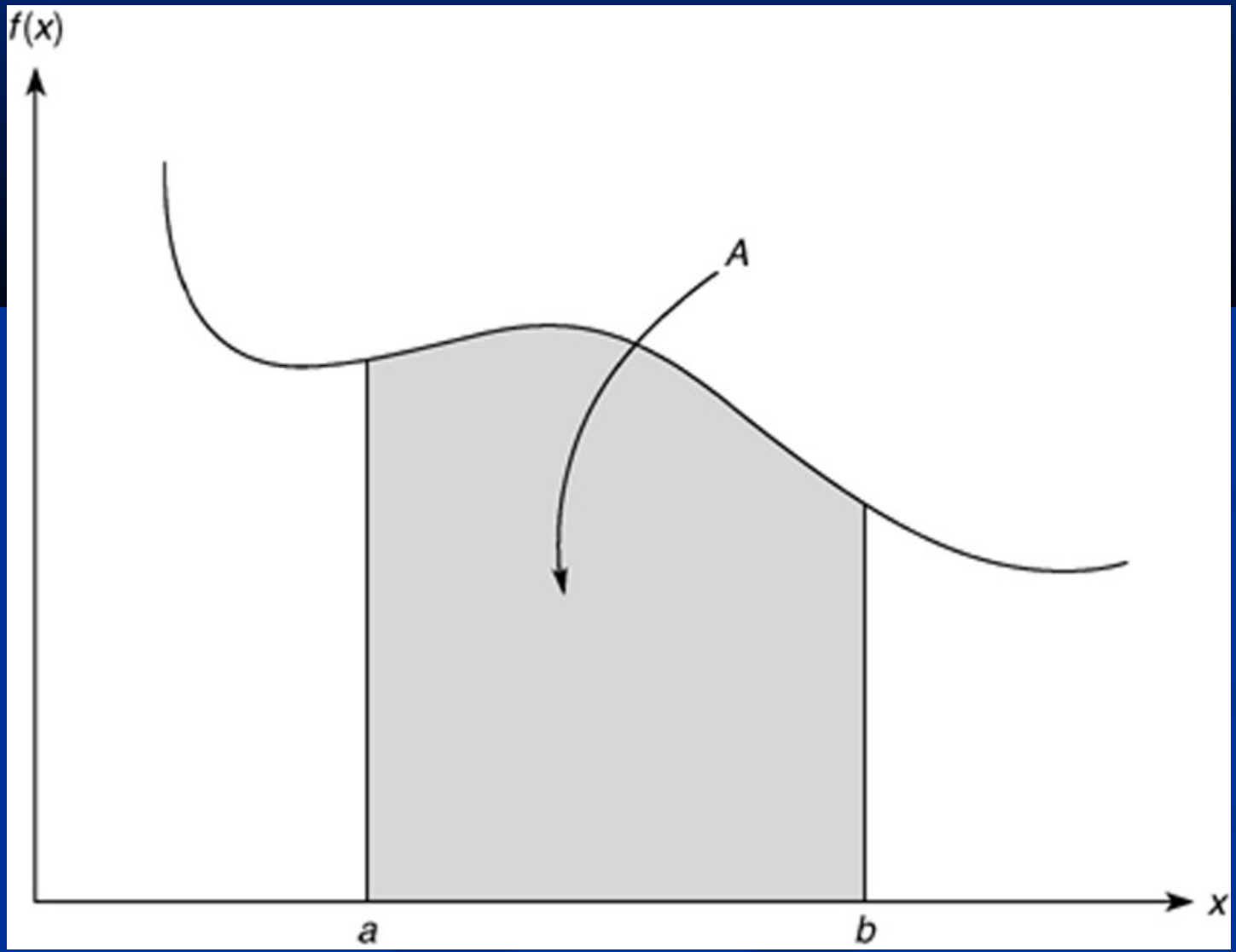
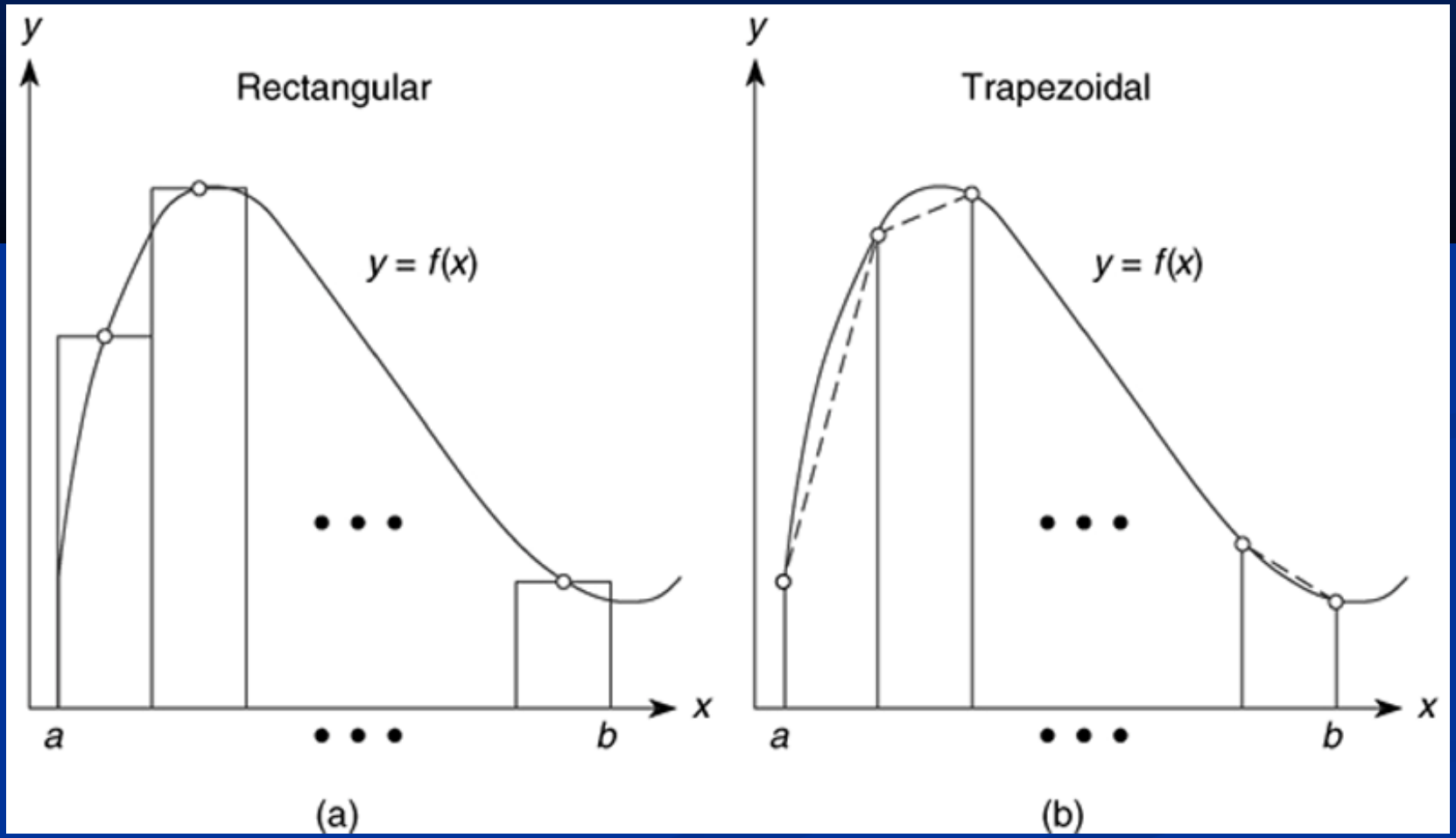


Illustration of (a) rectangular and (b) trapezoidal numerical integration. Figure 9.1–1, page 370.



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.

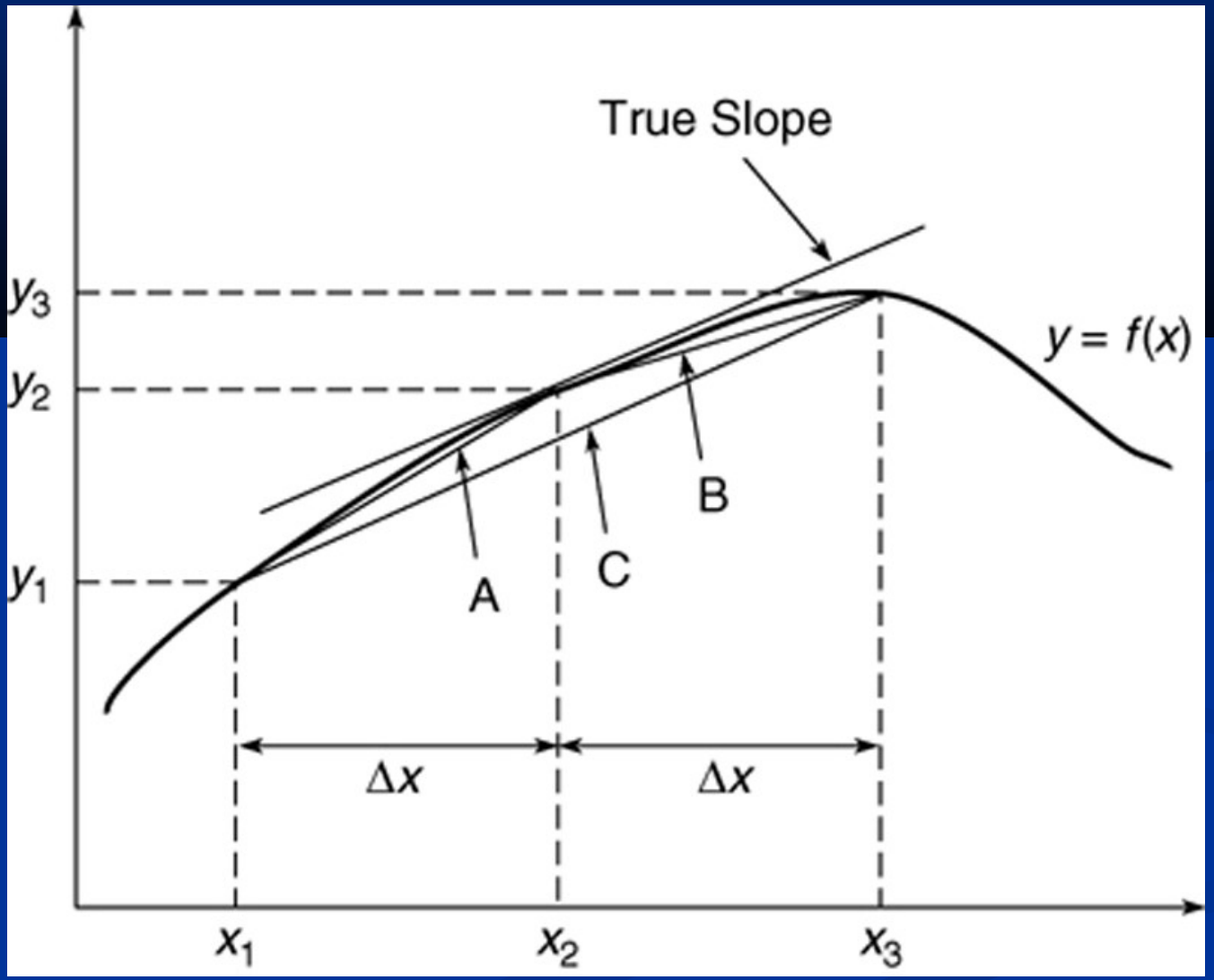
Triple Integrals

`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.

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



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.

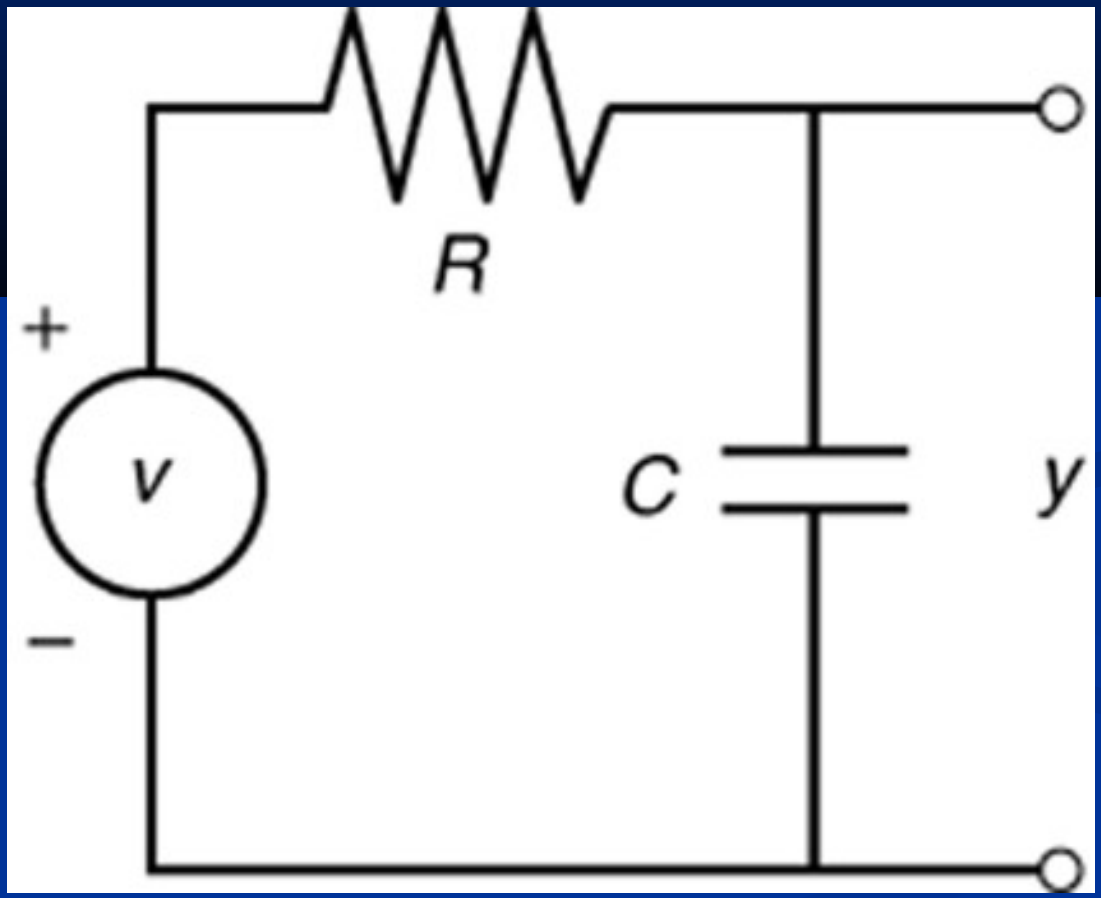
Solving First-Order Differential Equations, Section 9.3.

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.

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



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;
```

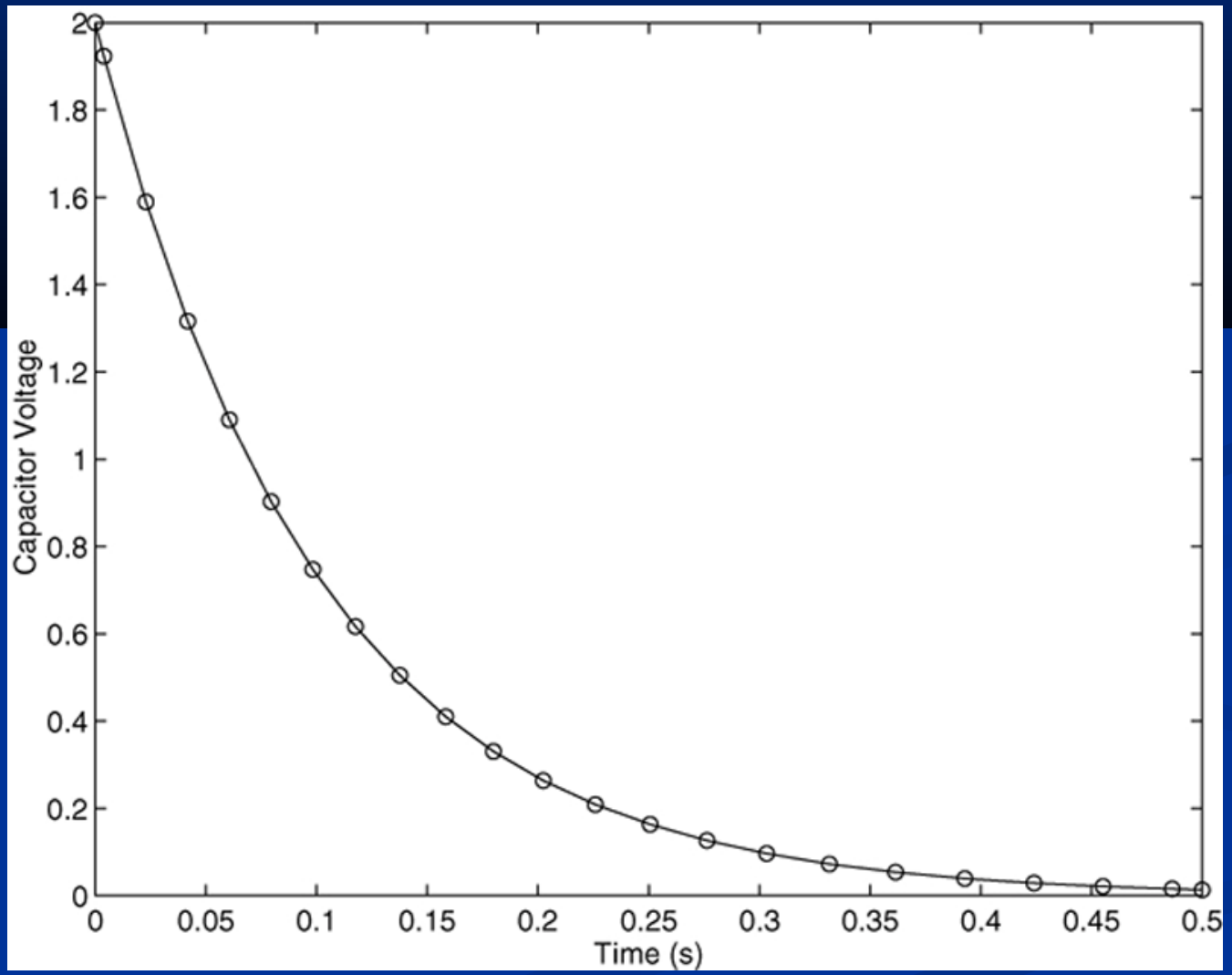
The function is called as follows, and the solution plotted along with the analytical solution `y_true`. The intial condition is $y(0)=2$.

```
[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.

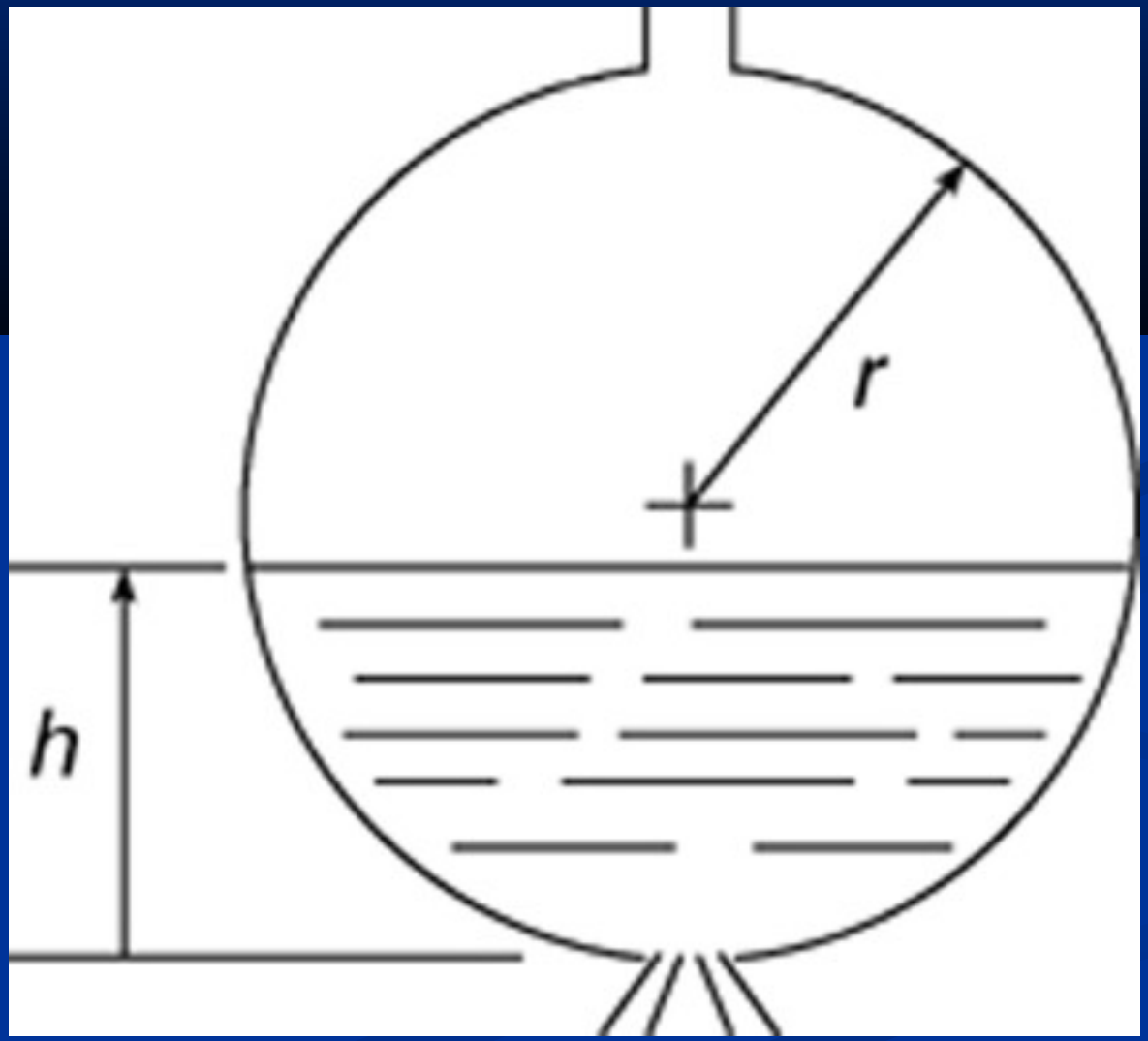
The plot is shown on the next slide and in Figure 9.3-2 on page 387..

Free response of an RC circuit. Figure 9.3-2



More? See pages 498-500.

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



The equation for the height is

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

First create the following function file.

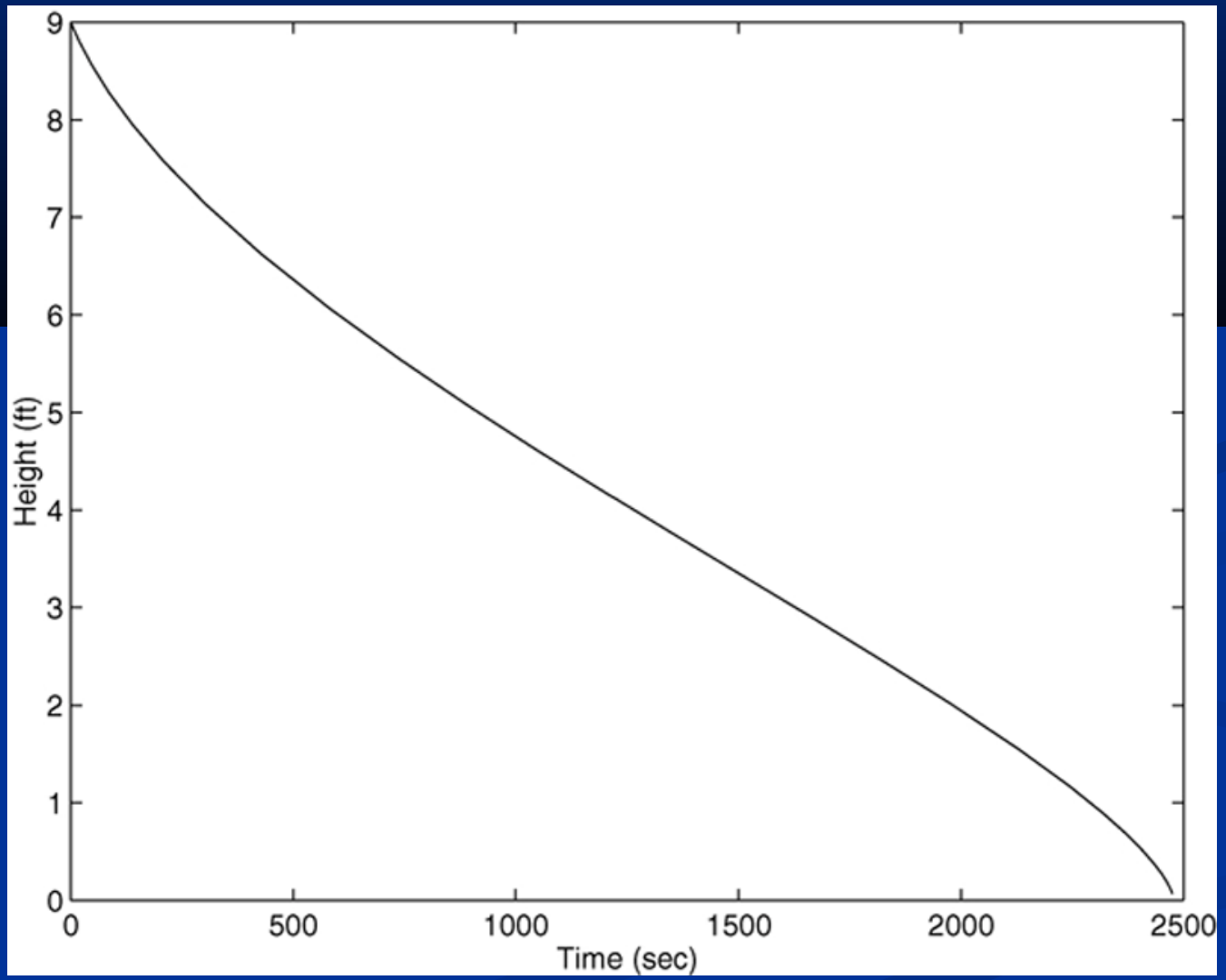
```
function hdot = height(t,h)
Hdot = -(0.0344*sqrt(h))/(10*h-h^2);
```

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.

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



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.*

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.

Suppose we want to solve (9.4-1) for $0 \leq t \leq 6$ 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

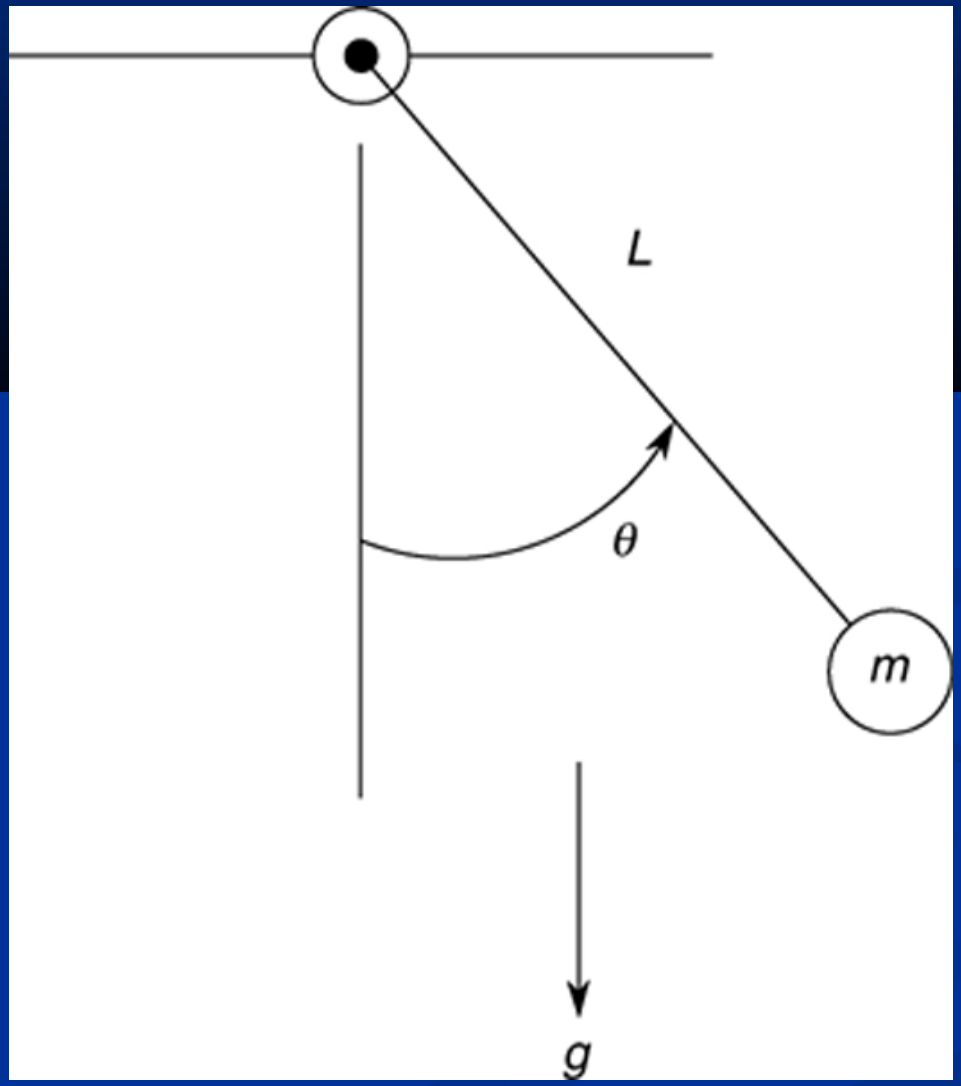
```
[t, x] = ode45(@example_1, [0, 6], [3, 9]);
```

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



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`.

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

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

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

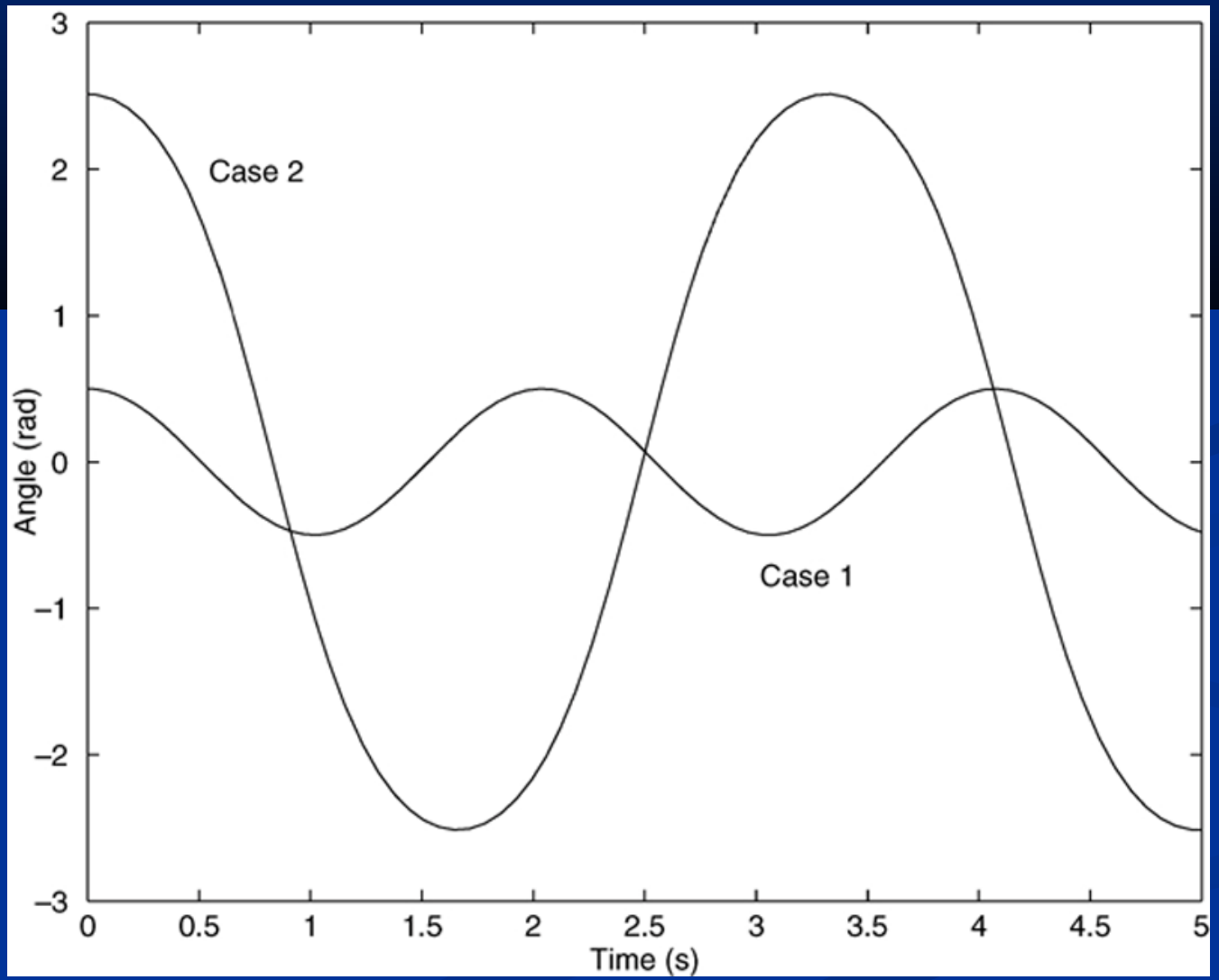
```
function xdot = pendulum(t,x)
g = 9.81; L = 1;
xdot = [x(2); -(g/L)*sin(x(1))];
```

The file is called as follows for the case where $\theta(0)=0.5$ 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.

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

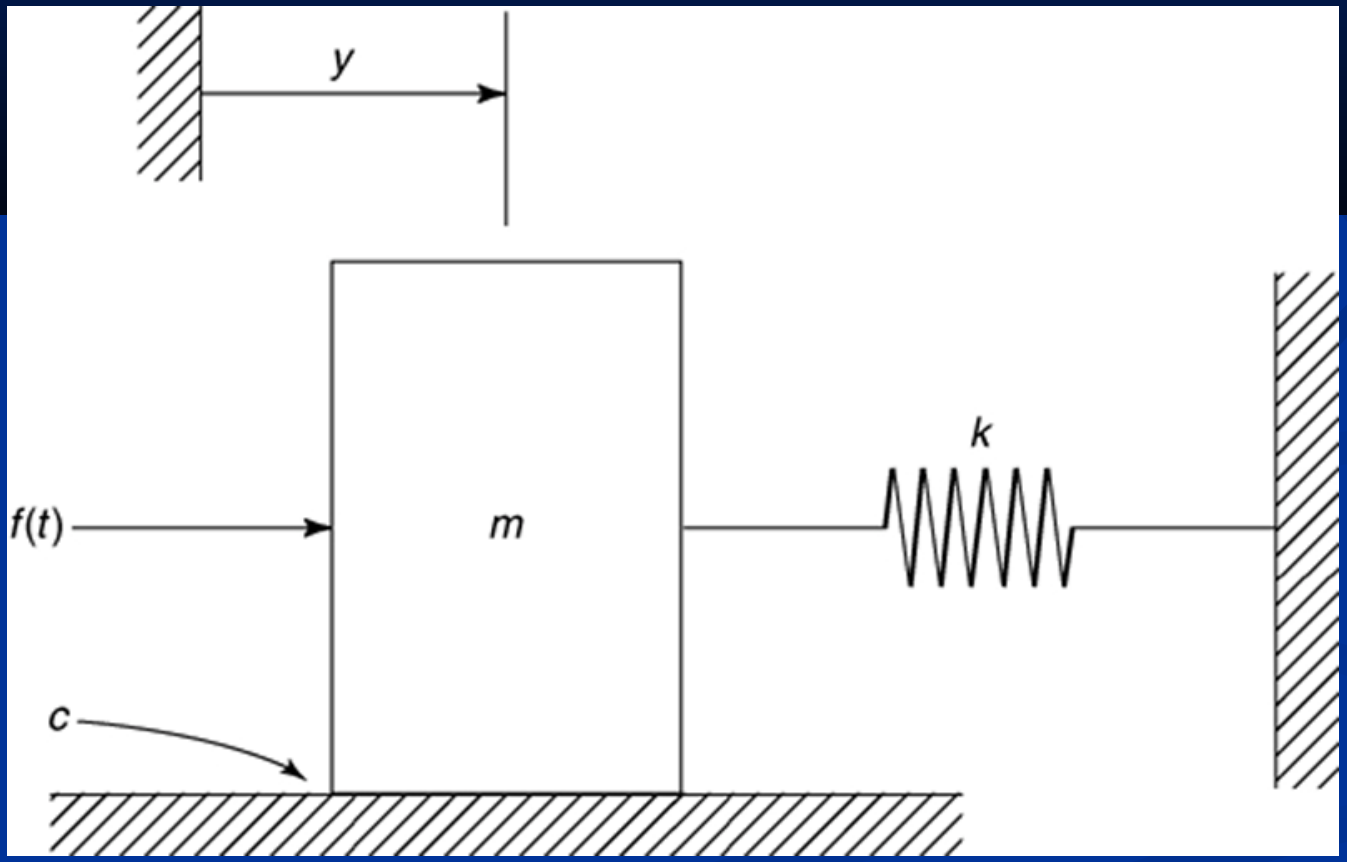


More? See pages 391-394.

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`.

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

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



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.

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

Create the following function file.

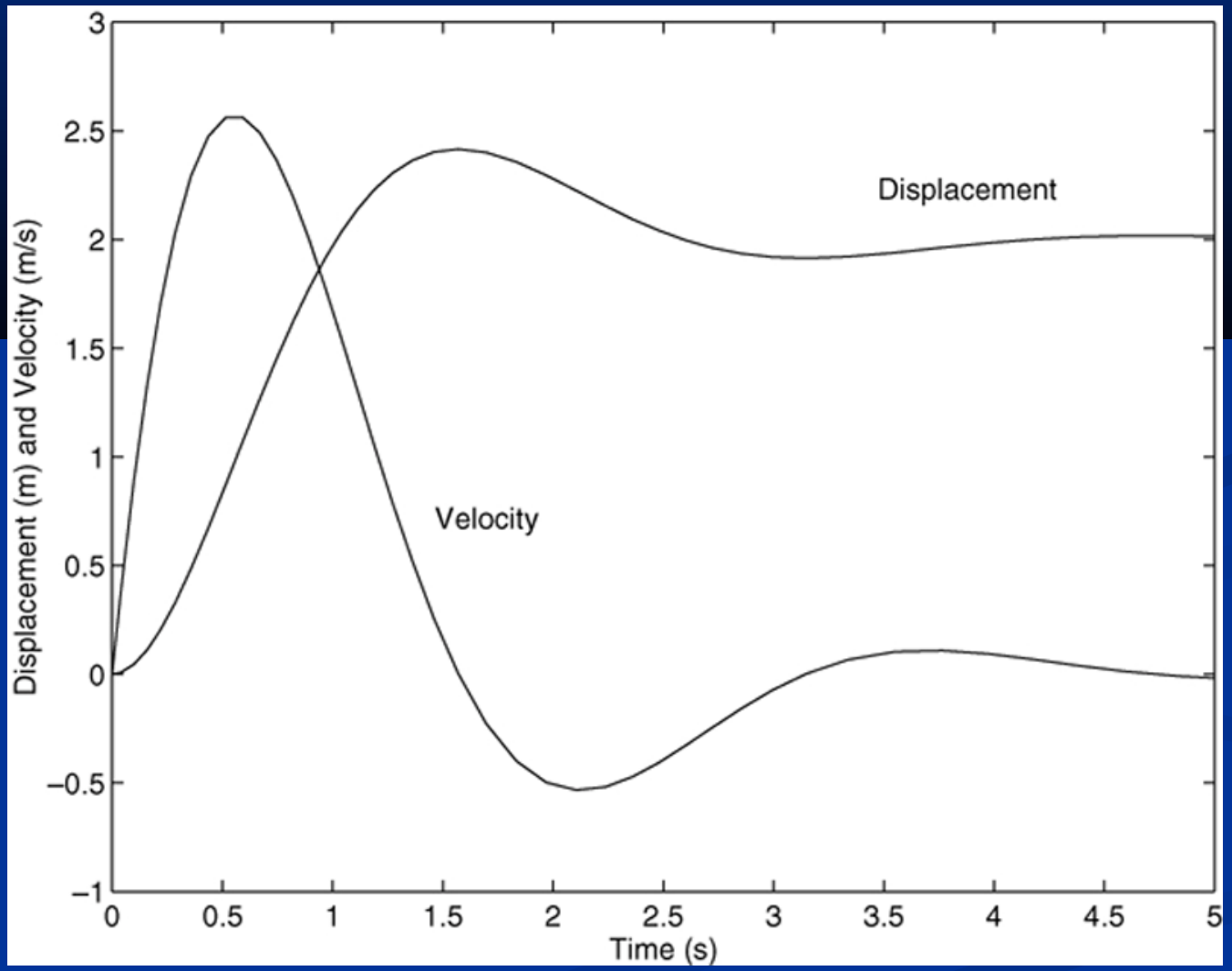
```
function xdot = msd(t,x)
f = 10; m = 1; c = 2; k = 5;
A = [0, 1;-k/m, -c/m];
B = [0; 1/m];
xdot = A*x + b*f;
```

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.

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



More? See pages 395-396.

ODE Solvers in the Control System Toolbox, page 398.

Transfer function form: It is created by typing `tf(right, left)` where the vector `right` contains the coefficients on the right side of the equation and the vector `left` contains the coefficients on the left side. Consider the equation (9.5-10).

$$5\ddot{y} + 7\dot{y} + 5y = 5\dot{f} + f(t)$$

You create the transfer function model form named `sys1` by typing `sys = tf([5, 1],[5, 7, 5]);` You can plot the free response for the initial conditions 5 and -2 by typing

```
initial(sys, [5, -2])
```

You can plot the unit step response for zero initial conditions by typing `step(sys1)`. See pages 399 – 402 and Tables 9.5-1 and 9.5-2 for more information about the `tf`, `initial`, and `step` functions.

LTI object functions. Table 9.5–1 on page 400.

Command	Description
`sys = ss(A, B, C, D)`	Creates an LTI object in state-space form, where the matrices `A`, `B`, `C`, and `D` correspond to those in the model ***dx/dt*** = **Ax** + **Bu**, **y** = **Cx** + **Du**.
`[A, B, C, D] = ssdata(sys)`	Extracts the matrices `A`, `B`, `C`, and `D` corresponding to those in the model ***dx/dt*** = **Ax** + **Bu**, **y** = **Cx** + **Du**.
`sys = tf(right,left)`	Creates an LTI object in transfer-function form, where the vector `right` is the vector of coefficients of the right-hand side of the equation, arranged in descending derivative order, and `left` is the vector of coefficients of the left-hand side of the equation, also arranged in descending derivative order.
`[right, left] = tfdata(sys)`	Extracts the coefficients on the right- and left-hand sides of the reduced-form model.

Basic syntax of the LTI ODE solvers. Table 9.5–2 on page 401.

Command	Description
`impulse(sys)`	Computes and plots the unit-impulse response of the LTI object `sys`.
`initial(sys,x0)`	Computes and plots the free response of the LTI object `sys` given in state-model form, for the initial conditions specified in the vector `x0`.
`lsim(sys,u,t)`	Computes and plots the response of the LTI object `sys` to the input specified by the vector `u`, at the times specified by the vector `t`.
`step(sys)`	Computes and plots the unit-step response of the LTI object `sys`.

The state variable form can be created with the `ss` function. Consider the model

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

Create the state space model `sys` for $m = 2$, $c = 5$ and $k = 3$, and plot the unit step response of the first variable by typing

```
m = 2; c = 5; k = 3;
A = [0, 1;-k/m, -c/m];B = [0; 1/m];
C = [1,0]; D = 0;
sys = ss(A,B,C,D);step(sys)
```

See pages 399 – 402 and Table 9.5-1 for more information about the `ss` function.

The `impulse` function computes and plots the impulse response. The `lsim` function computes and plots the solution for a user-defined input function. Both can be used with either the transfer function or the state variable forms. See Table 9.5-2 on page 401. Here is an example for the following equation with $f(t) = 10 \sin 3t$ and $y(0)=2$ for $t = 0$ to $t = 10$.

$$5\ddot{y} + 7\dot{y} + 4y = f(t)$$

```
sys = tf(1,[5,7,4]);
t = linspace(0,10,500);
f = 10*sin(3*t);
lsim(sys,f,t,2)
```

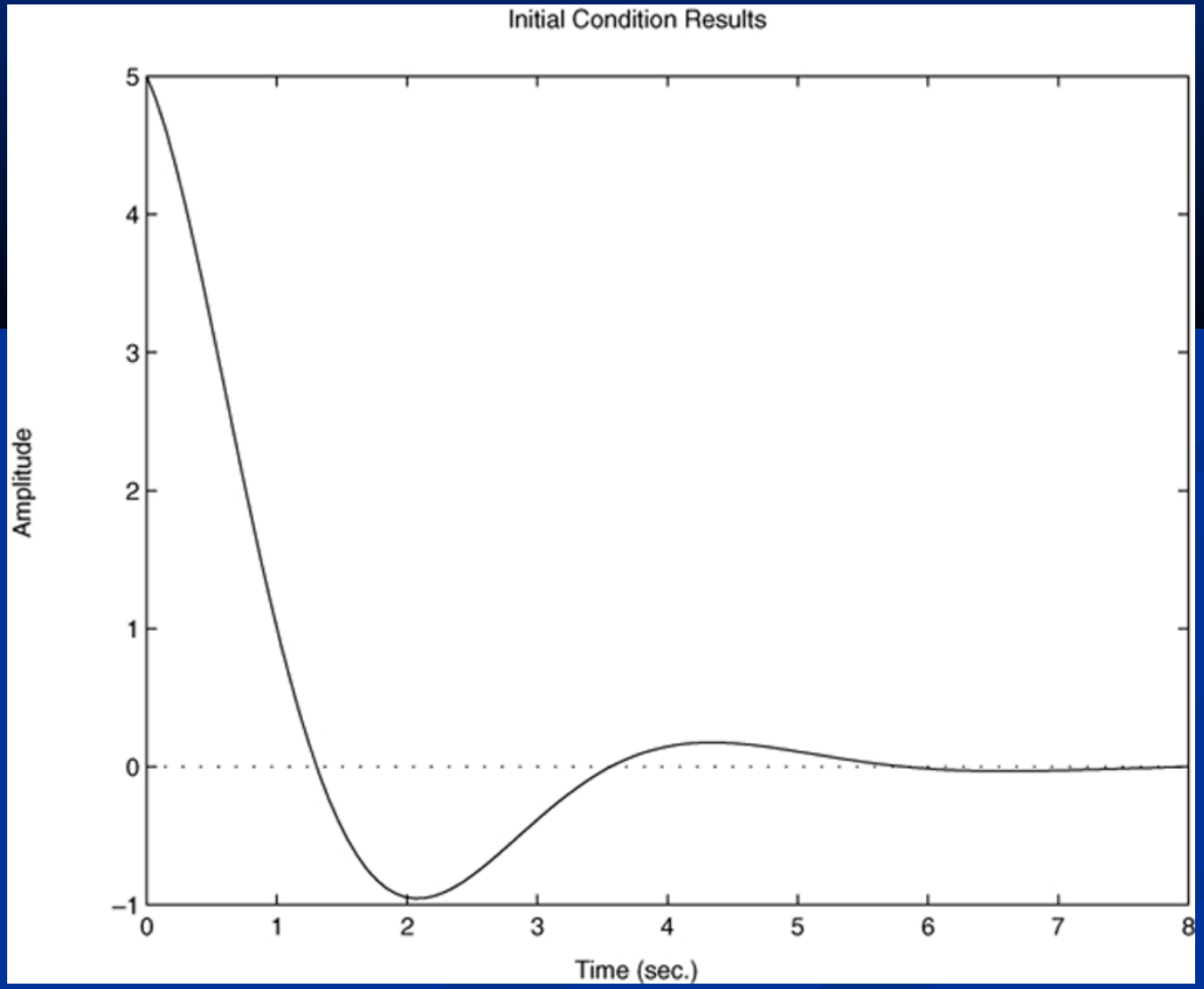

See pages 402 – 407 for more information.

The command `initial(sys,x0)`computes and plots the free response of the LTI object `sys` given in state-model form, for the initial conditions specified in the vector `x0`. For example,

```
m = 2; c = 5; k = 3;
A = [0, 1;-k/m, -c/m];B = [0; 1/m];
C = [1,0]; D = 0;
sys = ss(A,B,C,D);
initial(sys, [5, -2])
```

The response is shown on the next slide.

Free response of the model given by (9.5-5) through (9.5-8) for $x_1(0) = 5$ and $x_2(0) = -2$. Figure 9.5-2 on page 401.



Predefined Input Functions (pages 407-408)

The `gensig` function makes it easy to construct periodic input functions. The syntax is

```
[u,t] = gensig(type, period)
```

where `type` can be ‘sin’, ‘square’, or ‘pulse’ and `period` is the desired period of the input. The vector `t` contains the times and the vector u contains the input values at those times. The next slide gives an example using a square wave,

Square-wave response of the model $\ddot{x} + 2\dot{x} + 4x = 4f$.
Figure 9.5-6 on page 408.