


MATLAB Course
For Engineers
lecture 6


Curve Fitting
Numerical Integration
Numerical Differentiation
Numerical Methods for Differential Equations

Curve Fitting (regression analysis)

Curve fitting is a process of fitting a function to a set of data points. The function can then be used as a mathematical model of the data.
There are many types of functions (linear, polynomial, power, exponential, etc).

Curve fitting with Polynomial, the `polyfit` function is based on the least-squares method.

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

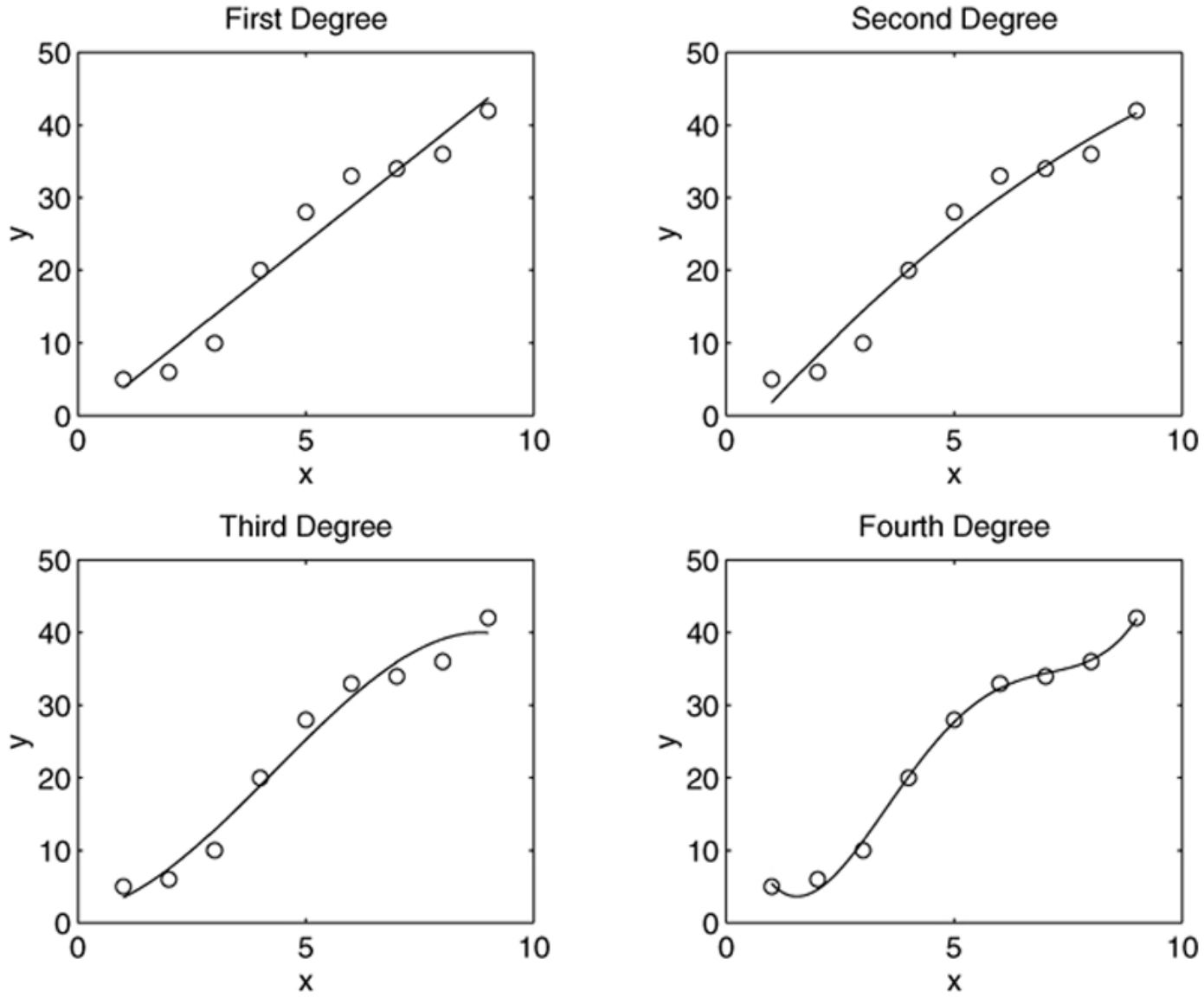
Description

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

Example: Find a curve fitting (best fit) for a set data, where x=1,2,3,4,5,6…9 and y= 5,6,10,20,28,33,34,36,42.

The following script file computes the coefficients of the first – through fourth-degree polynomials for these data.

```
x=[1:9];
y=[5,6,10,20,28,33,34,36,42];
P1=polyfit(x,y,1) % First order
subplot(2,2,1)
plot(x,y,'o',x,polyval(P1,x),'k'),xlabel('x'),ylabel('y')
P2=polyfit(x,y,2) % Second order
subplot(2,2,2)
plot(x,y,'o',x,polyval(P2,x),'k'),xlabel('x'),ylabel('y')
P3=polyfit(x,y,3) % Third order
subplot(2,2,3)
plot(x,y,'o',x,polyval(P3,x),'k'),xlabel('x'),ylabel('y')
P4=polyfit(x,y,4) % Fourth order
subplot(2,2,4)
plot(x,y,'o',x,polyval(P4,x),'k'),xlabel('x'),ylabel('y')
```


First Degree
Second Degree
Third Degree
Fourth Degree
y
x
0
5
10
20
30
40
50

Curve Fitting with Functions Order than Polynomials

Using command `polyfit` to find the curve fitting of these function :

$y = bx^m$	(Power Function)
$y = be^{mx}$ or $y = b(10)^{mx}$	(Exponential Function)
$y = mln(x)+b$ or $y = mlog(x)+b$	(Logarithmic Function)
$y = 1/(mx+b)$	(Reciprocal Function)

All of these functions can easily be fitted to given data with the `polyfit` function. This is done by rewriting the functions in a form that can be fitted with a linear polynomial (n=1) which has the form : $y=mx+b$

The logarithmic function is already in this form, and the power, exponential and reciprocal equations can be rewritten as:

$\ln(y)= m\ln(x)+\ln(b)$	(Power Function)
$\ln(y)= mx+\ln(b)$ or $\log(y)= mx+\log(b)$	(Exponential Function)
$1/y= mx+b$	(Reciprocal Function)

`polyfit` function form

```
p=polyfit(log(x),log(y),1)
p=polyfit(x,log(y),1)
p=polyfit(x,log10(y),1)
p=polyfit(log(x),y,1)
p=polyfit(log10(x),y,1)
p=polyfit(x,1./y,1)
```

$y = bx^m$	(Power Function)
$y = be^{mx}$	(Exponential Function)
$y = b(10)^{mx}$	
$y = m\ln(x)+b$	(Logarithmic Function)
$y = m\log(x)+b$	
$y =1/(mx+b)$	(Reciprocal Function)

For given data it is possible to foresee, to some extant, which of the function has the potential for providing a good fit. This is done by plotting the data using different combinations of linear and logarithmic axes. If the data points in one of the plots appear to fit a straight line, the corresponding function can provide a good fit according to the list below.

x axis	*y* axis	Function
linear	linear	linear $y = mx + b$
logarithmic	logarithmic	power $y = bx^m$
linear	logarithmic	exponential $y = be^{mx}$ or $y = b10^{mx}$
logarithmic	linear	logarithmic $y = m\ln(x) + b$ or $y = m\log(x) + b$
linear	linear (plot $1/y$)	reciprocal $y = \frac{1}{mx + b}$

Considerations when choosing a function are:

- **Exponential functions can not pass through the origin.**
- **Exponential functions can only fit data with all positive y's, or all negative y's.**
- **Logarithmic functions can not model x=0, or negative values of x.**
- **For the power function y=0 when x=0.**
- **The reciprocal equation can not model y=0.**

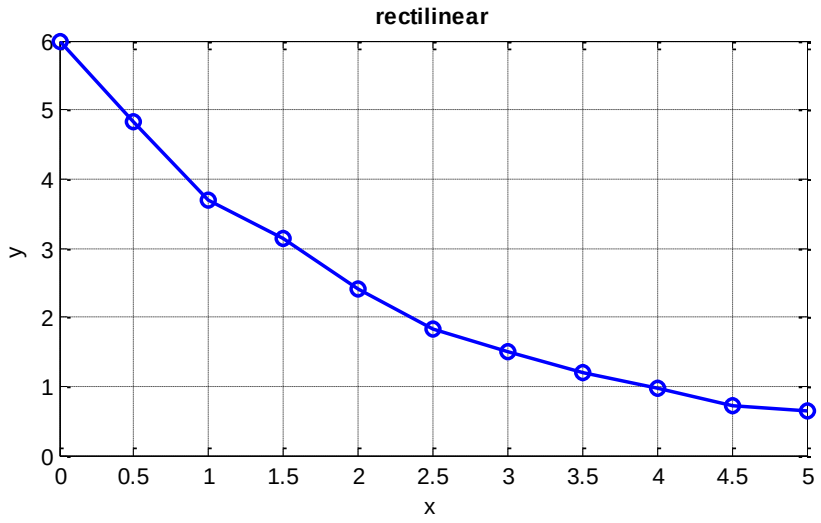
Example: The following data points are given. Determine a function $w=f(t)$ where is t independent variable, w is the dependent variable.

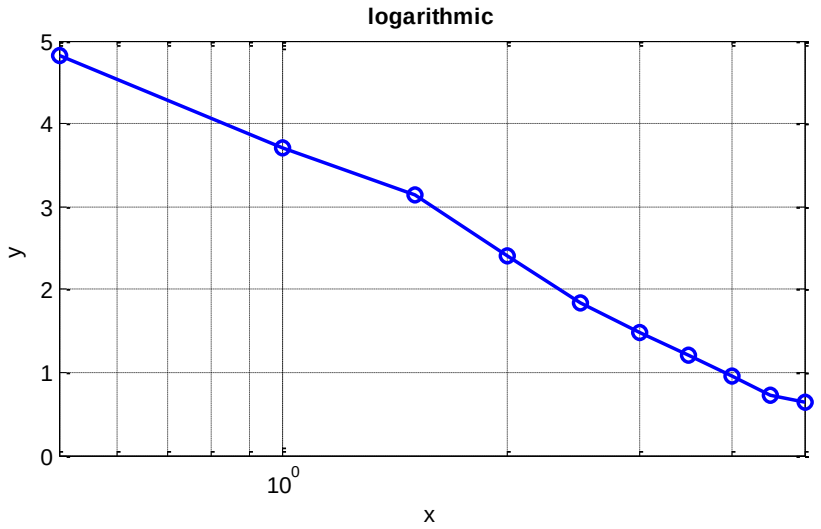
t	0	0.5	1.0	1.5	2.0	2.5	3.0	3.5	4.0	4.5	5.0
w	6	4.3	3.7	3.15	2.41	1.83	1.49	1.21	0.96	0.73	0.64

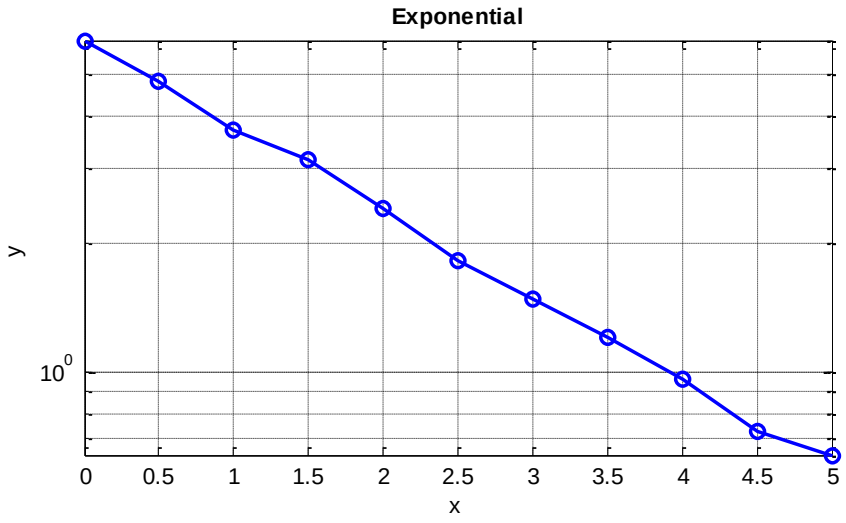
The following script file computes the coefficients of the function:

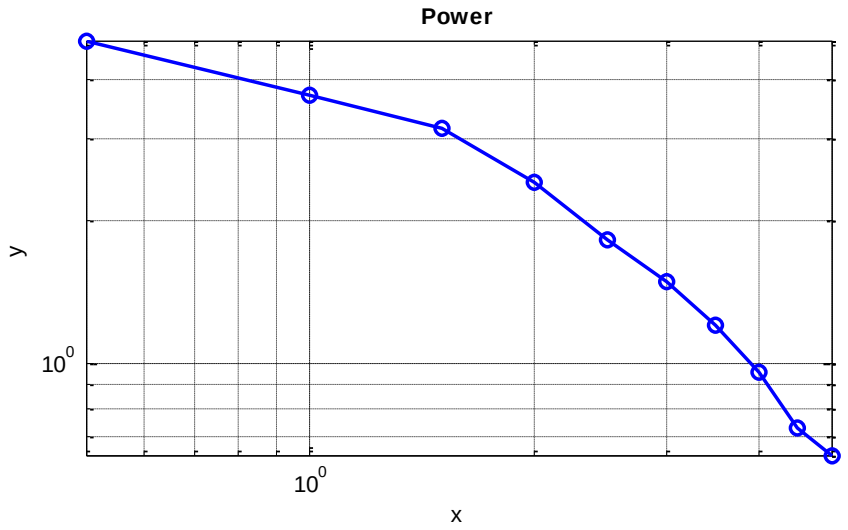
```
t=[0:.5:5];
w=[6,4.83,3.7,3.15,2.41,1.83,1.49,1.21,…
0.96,0.73,0.64];
subplot(2,2,1)
plot(t,w,'o-')
subplot(2,2,2)
semilogx(t,w,'o-')
subplot(2,2,3)
semilogy(t,w,'o-')
subplot(2,2,4)
loglog(t,w,'o-')
```

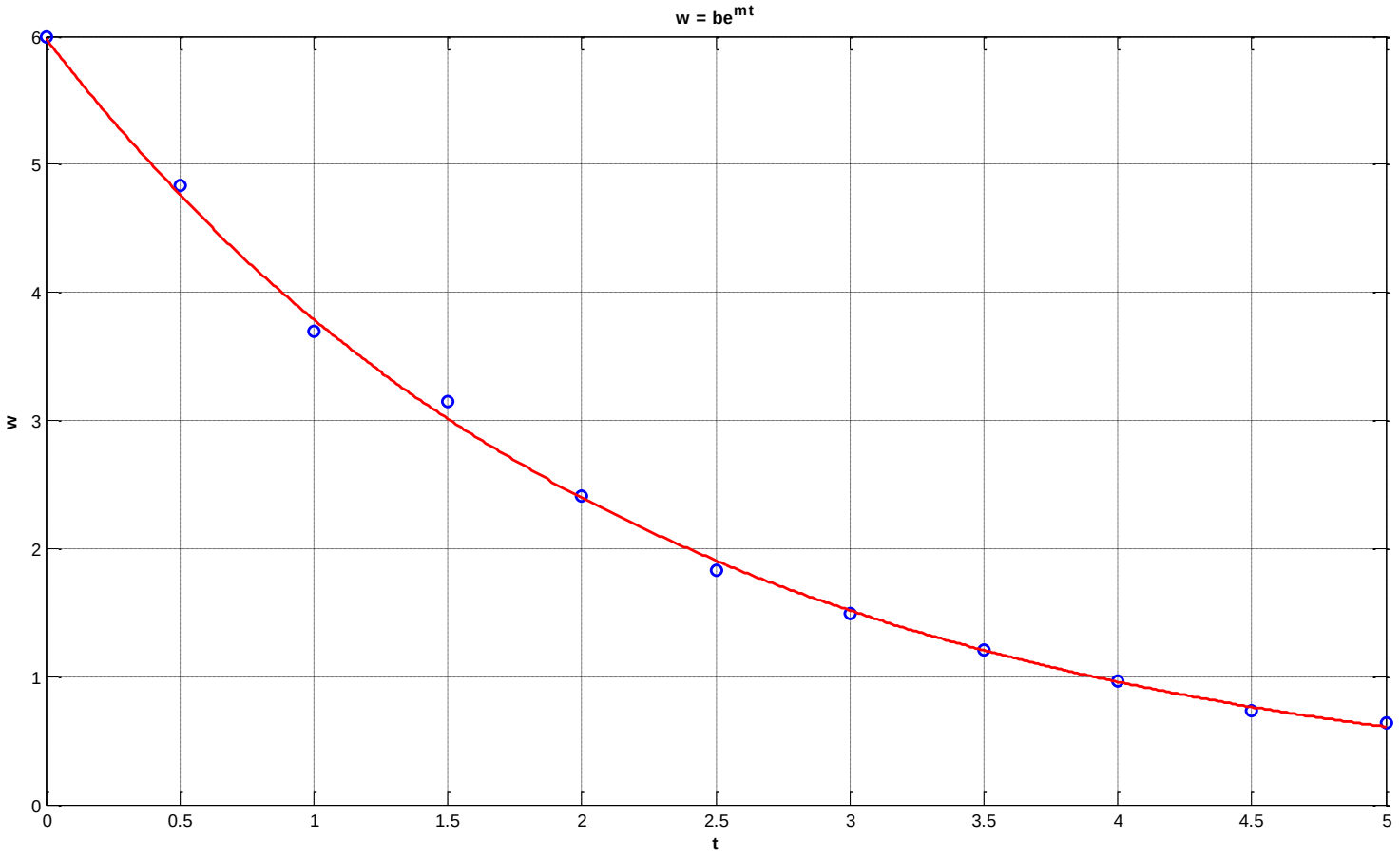
```
%The curve fitting of the function is…
exponential from graph
p=polyfit(t,log(w),1) %w=be^(mt)
m=p(1)
b=exp(p(2))
t1=[0:0.01:5];
w1=b*exp(m*t1);
figure
plot(t,w,'o',t1,w1,'r')
title('w = be^{mt}')
xlabel('t')
ylabel('w')
```


rectilinear
y
x
0
1
2
3
4
5
6
0.5
1.5
2.5
3.5
4.5


logarithmic
y
x
0
1
2
3
4
5
10^0


Exponential
y
x
10^0
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5


Power
y
x
10^0
10^0


w = be^mt
w
t
0
1
2
3
4
5
6
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5

Numerical Integration

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

$$A = \int_a^b f(x).dx$$

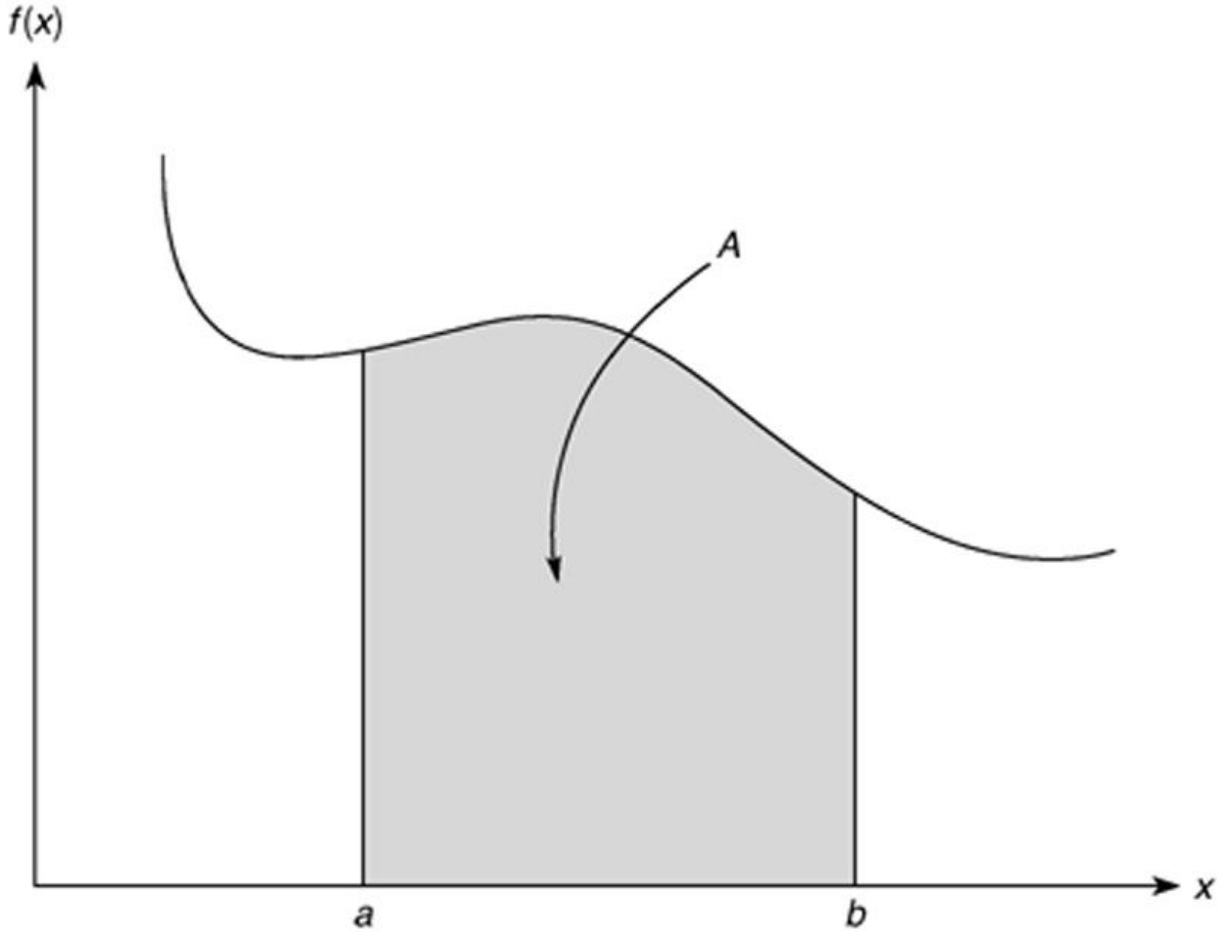
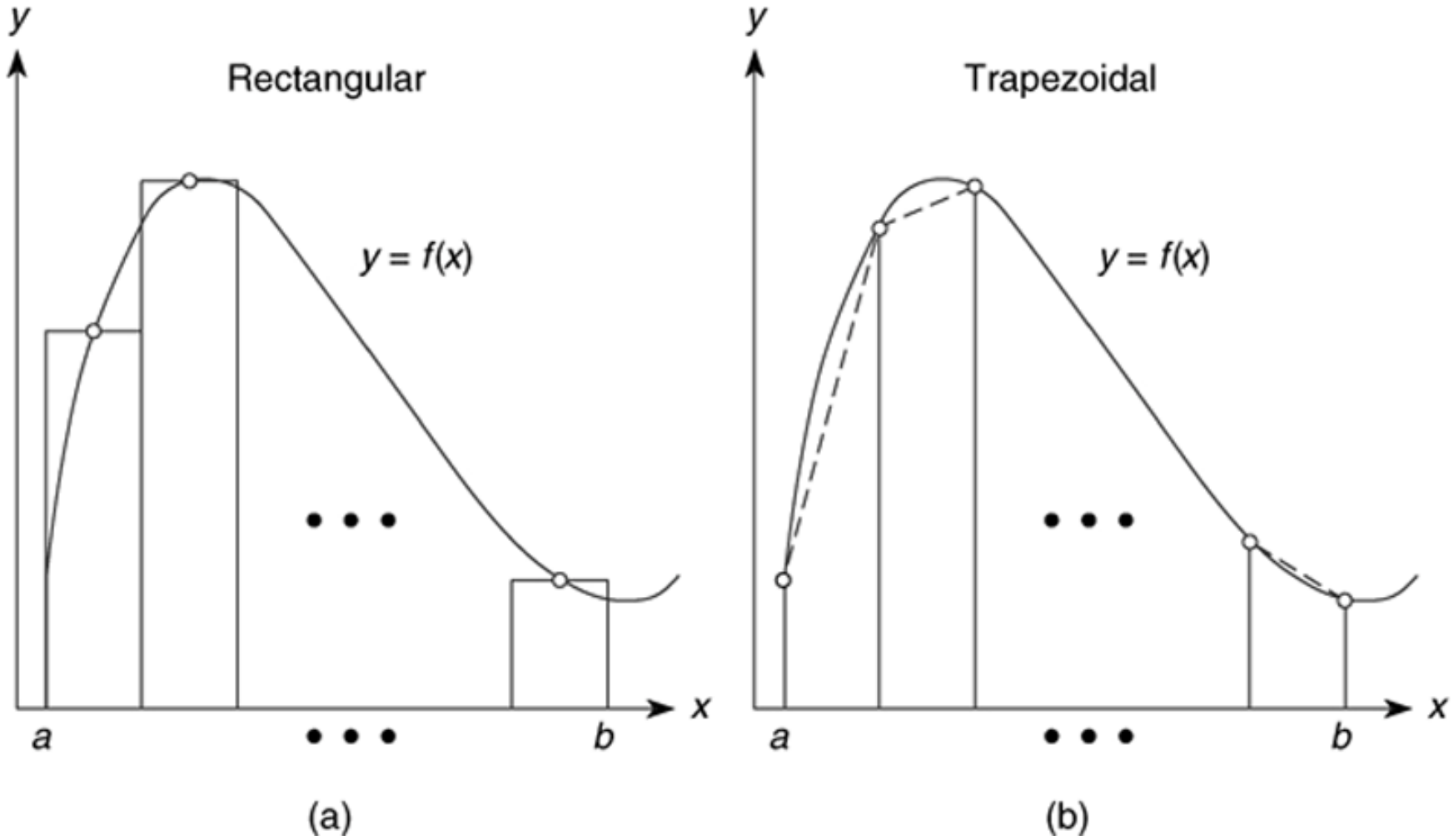


Illustration of (a) rectangular and (b) trapezoidal numerical integration.



Numerical integration functions

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.
`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.

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.

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.**

For example: the integral of $12x^3 + 9x^2 + 8x + 5$ **is obtained from** `q =polyint([12,9,8,5], 10).`

The answer is `q = [3, 3, 4, 5, 10],`

which corresponds to $3x^4 + 3x^3 + 4x^2 + 5x + 10$.

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 $A = \int_0^1 \int_1^3 xy^2 \, dx \, dy$ **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.

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

$$A = \int_1^2 \int_0^2 \int_1^3 \left(\frac{xy - y^2}{z} \right) dx\, dy\, dz$$

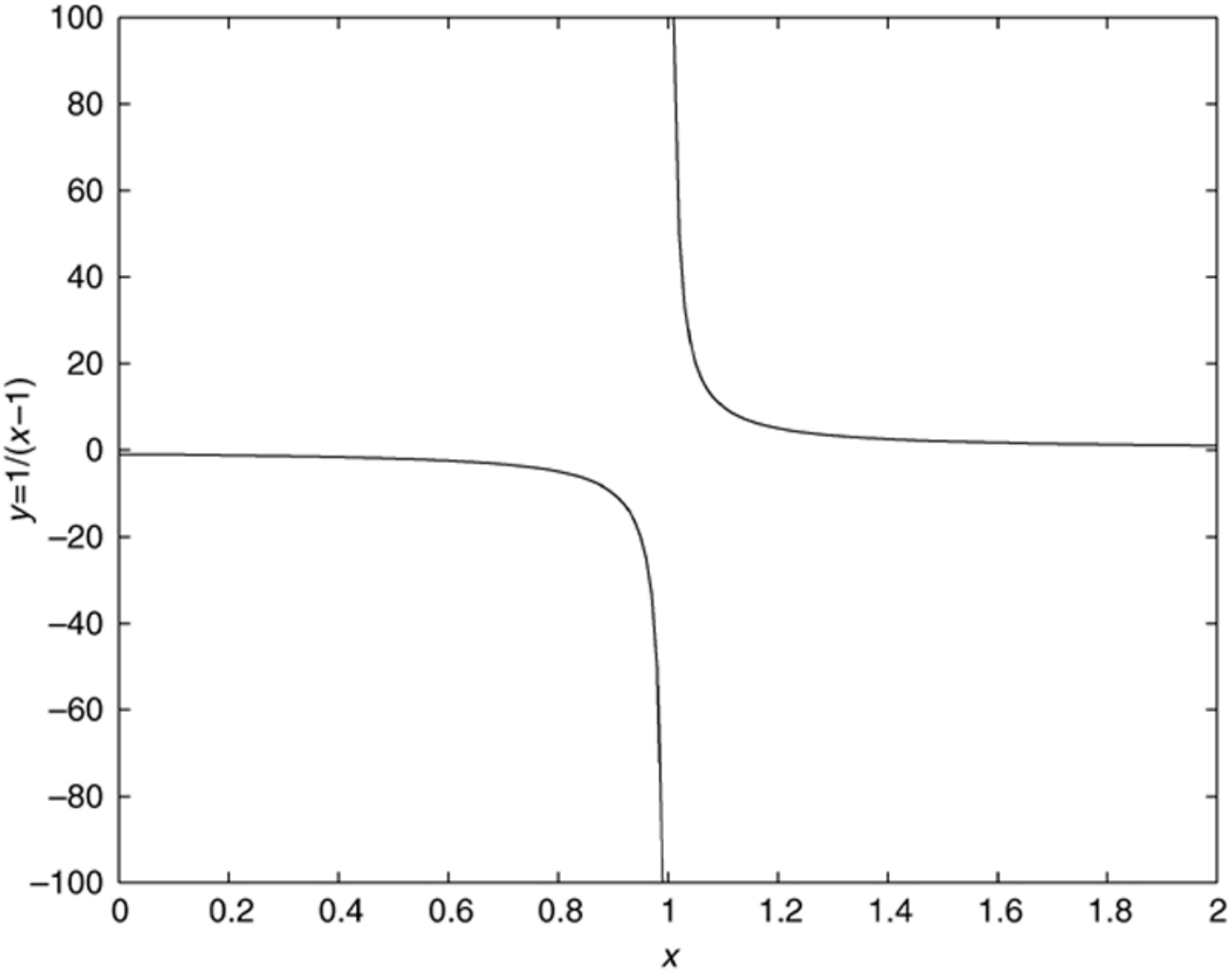
using an anonymous function to represent $f(x,y,z) = (xy - y^2)/z$.

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

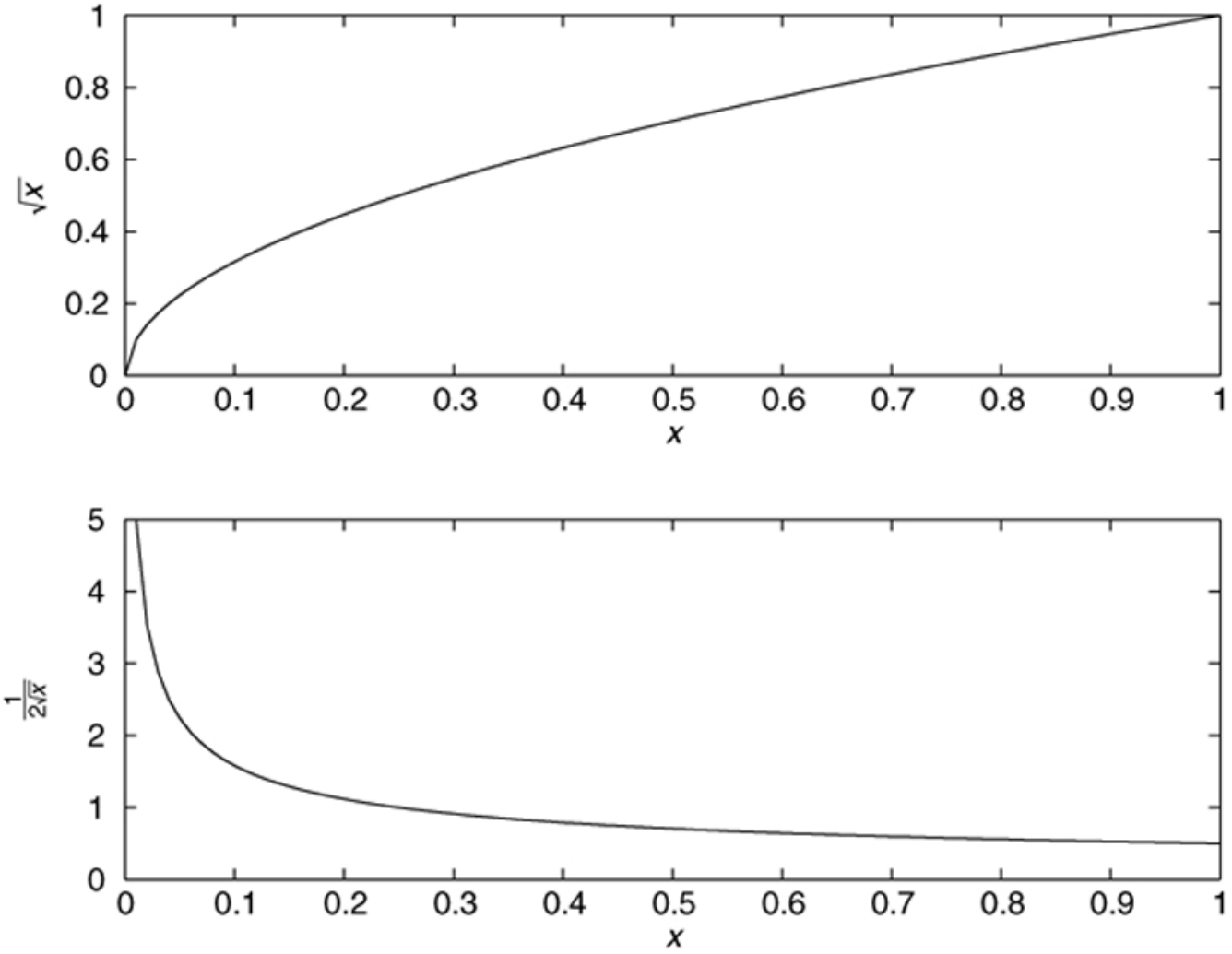
The answer is A = 1.8484. Note that the function must accept a vector x, but scalar y and z.

A potential problem with integration.
A function having a singularity at $x = 1$.

The plot of
y=1/(x-1)



Potential problem with integration: a function having a singularity in its slope function. The top graph shows the function $y = \sqrt{x}$. The bottom graph shows the derivative of y. The slope has a singularity at $x = 0$.



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

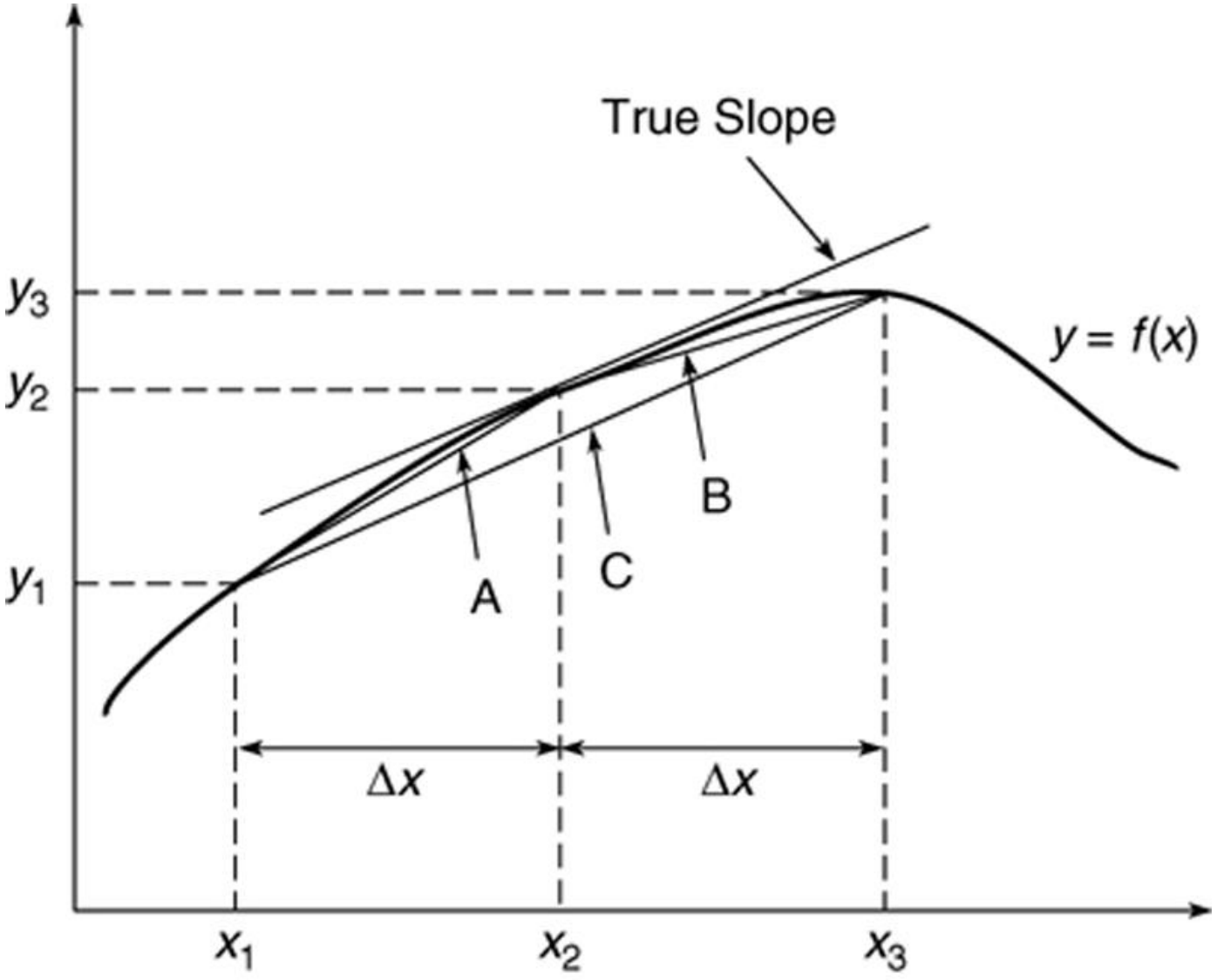
$$\int_0^{\pi} \sin x \, dx$$

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%.

Numerical Differentiation: Illustration of three methods for estimating the derivative *dy/dx*.



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]`.

The Euler Method

The *Euler method* is the simplest algorithm for numerical solution of a differential equation. Consider the equations

$$\frac{dy}{dt} = f(t, y) \qquad y(0) = y_0 \tag{9.3–1}$$

where $f(t, y)$ is a known function and y_0 is the initial condition, which is the given value of $y(t)$ at $t = 0$. From the de nition of the derivative,

$$\frac{dy}{dt} = \lim_{\Delta t \to 0} \frac{y(t + \Delta t) - y(t)}{\Delta t}$$

If the time increment Δt is chosen small enough, the derivative can be replaced by the approximate expression

$$\frac{dy}{dt} \approx \frac{y(t + \Delta t) - y(t)}{\Delta t} \tag{9.3–2}$$

Assume that the function $f(t, y)$ in Equation (9.3–1) remains constant over the time interval $(t, t + \Delta t)$, and replace Equation (9.3–1) by the following approximation:

$$\frac{y(t + \Delta t) - y(t)}{\Delta t} = f(t, y)$$

or

$$y(t + \Delta t) = y(t) + f(t, y)\Delta t \tag{9.3–3}$$

The smaller Δt is, the more accurate are our two assumptions leading to Equation (9.3–3). This technique for replacing a differential equation with a difference equation is the *Euler method*. The increment Δt is called the *step size*.

Equation (9.3–3) can be written in more convenient form as

$$y(t_{k+1}) = y(t_k) + \Delta t f[t_k, y(t_k)] \tag{9.3–4}$$

where $t_{k+1} = t_k + \Delta t$. This equation can be applied successively at the times t_k by putting it in a `for` loop. The accuracy of the Euler method can be improved sometimes by using a smaller step size. However, very small step sizes require longer run times and can result in a large accumulated error due to roundoff effects.

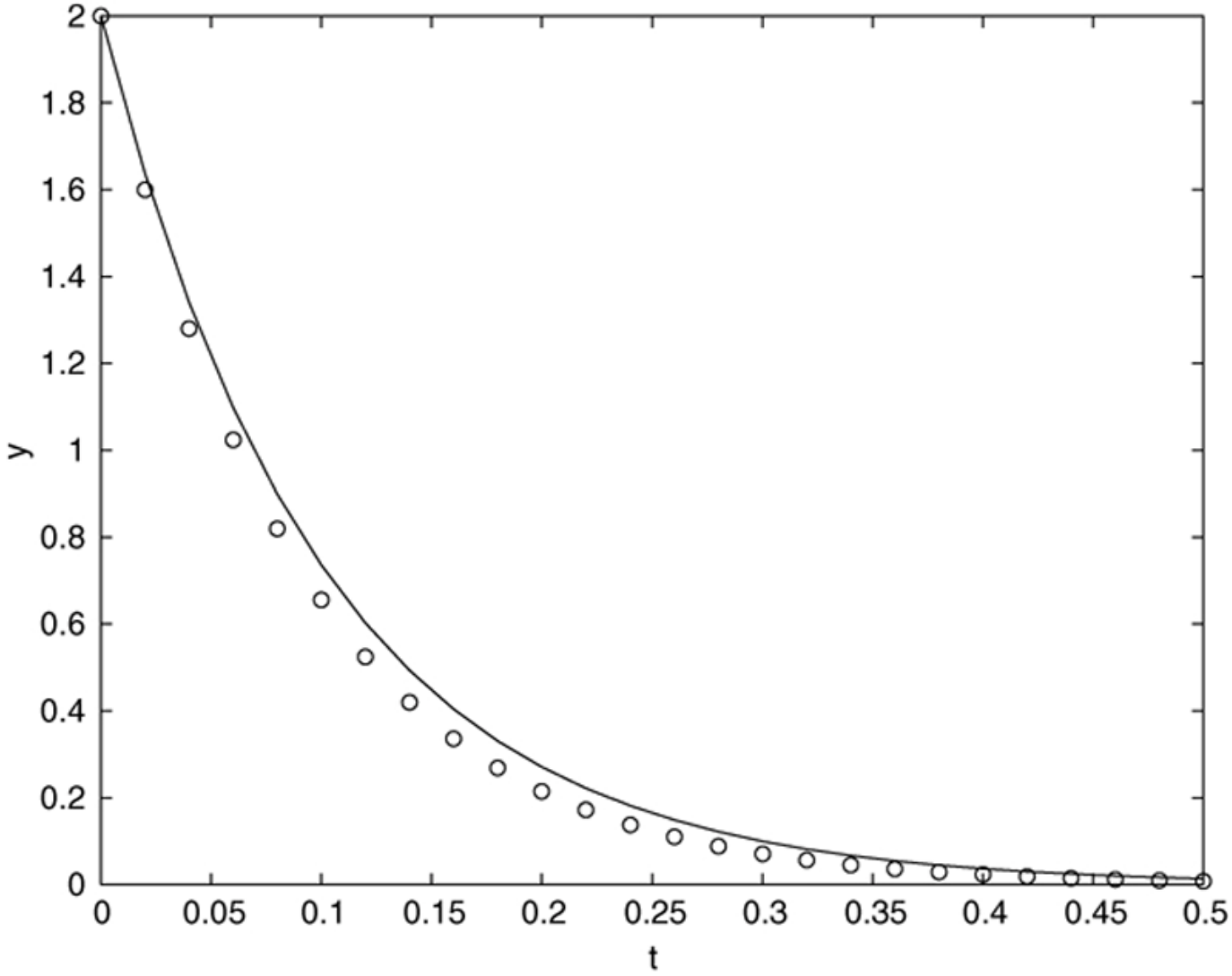
Numerical Methods for Differential Equations

Let $dy/dt = -10y$, $y(0) = 2$.and $0 \le t \le 0.5$.

The true solution is $y(t) = 2\,e^{-10t}$.
The following script files solves and plots the solution by using Euler method.

```
r = -10; delta = 0.02; y(1) = 2;
k=0;
for time = [delta:delta:0.5]
k = k + 1;
y(k+1) = y(k) + r*y(k)*delta;
end
t = [0:delta:0.5]; % for true solution.
y_t= 2*exp(-10*t); % true solution.
plot(t,y,'o',t,y_t),xlabel('t'),
...ylabel('y')
```

Euler method solution for the *free response* of $dy/dt = -10y$, $y(0) = 2$.



The ode solvers.

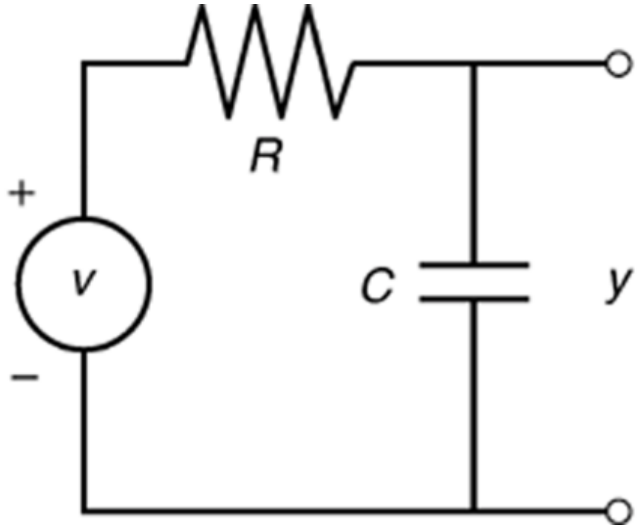
When used to solve the ODE equation $dy/dt = f(t, y)$, the basic syntax is (using `ode23` or `ode45` as the example):

```
[t,y] = ode23('ydot',[tspan], y0)
```

where ydot is the name 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.

Application of an ode solver to find the response of an RC circuit .

The circuit model for zero input voltage v is



$$RC\, dy/dt + y = 0$$

$RC=0.1s$, $v(t)=0$ $y(0)=2$.

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.

The func.name `rccirc.m`.

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

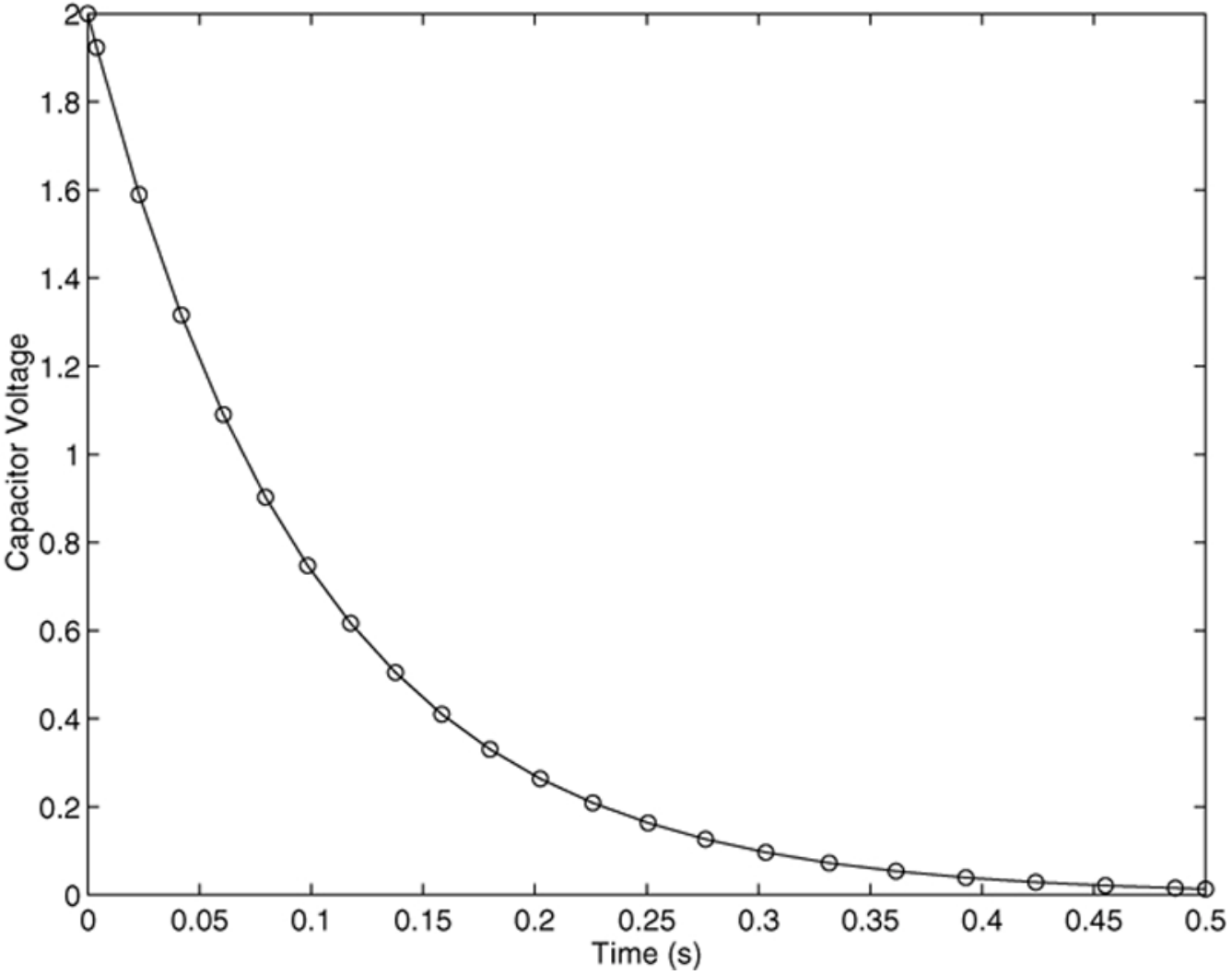
The function is called as follows, and the solution plotted along with the analytical solution y_true.

```
[t,y] = ode45('rccirc',[0,0.4],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. 

Free response of an RC circuit.



Higher – Order Differential Equations

To use the ODE solvers to solve an equation higher than order 1, you must first write the equation as a set of first-order equations.

For example,

$$5d^2y/dt^2 + 7dy/dt + 4y = f(t)$$

Set : $x_1 = y$, **and** $x_2 = dy/dt$

$$dx_1/dt = x_2$$

$$dx_2/dt_2 = (1/5)f(t) - (4/5)x_1 - (7/5)x_2$$

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

Solve: $5d^2y/dt^2 + 7dy/dt + 4y = \sin t$

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

```
function xdot= example1(t,x)
% Computes derivatives of two equations
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 pervious ODE equation for $0 \le t \le 6$ **with the initial conditions** $x_1(0) = 3$, $x_2(0) = 9$.

Then the initial condition for the vector `x` **is** `[3, 9]`. **To use** `ode45`, **you type**

```
[t,x] = ode45('example1',[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),t,x(:,2))`.