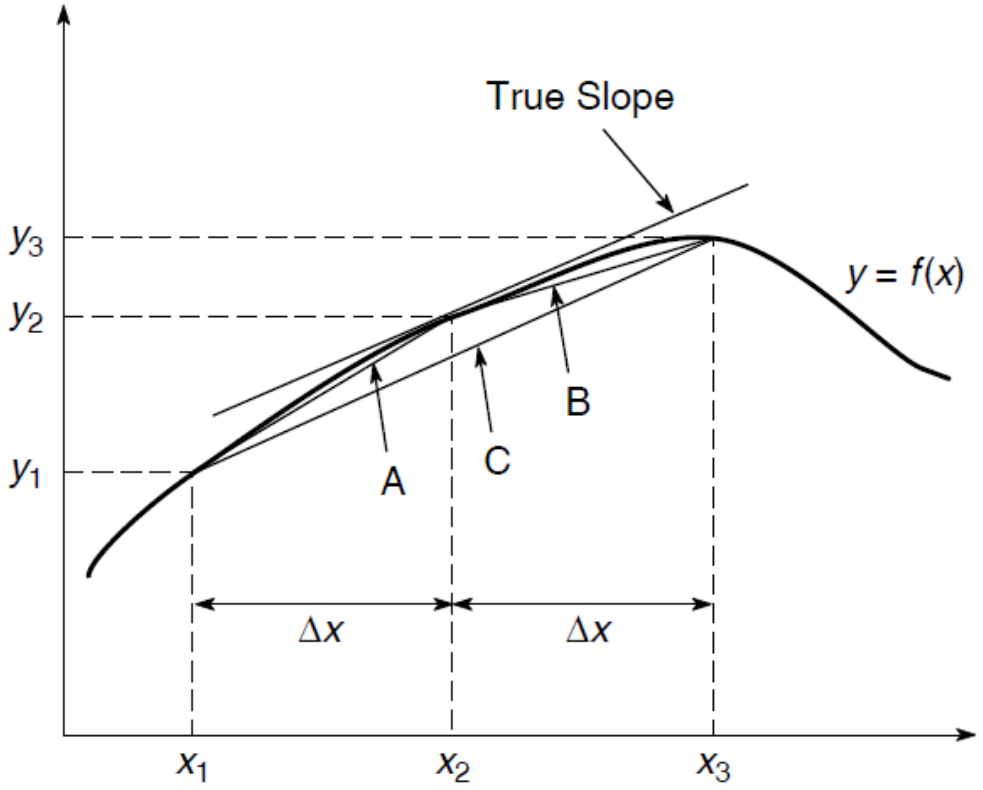
MATLAB 与工程应用

ODE with initial condition

Numerical Differentiation

$$\frac{dy}{dx} = \lim_{\Delta x \to 0} \frac{\Delta y}{\Delta x}$$



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 derivative dy/dx can be estimated from `diff(y)./diff(x)`.

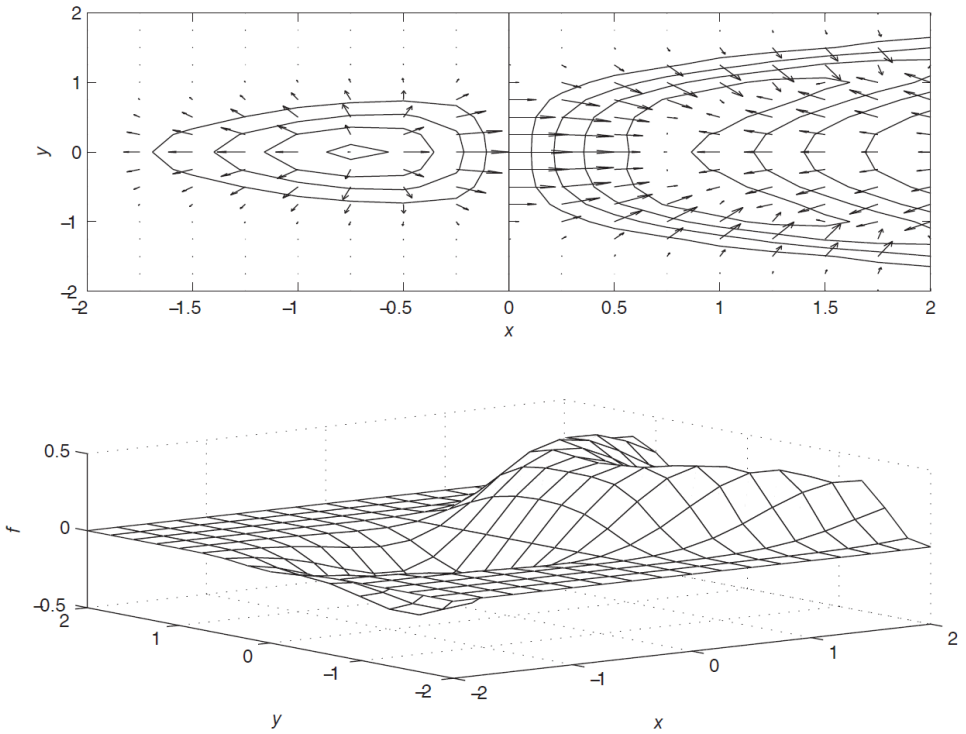
```
x = 0:pi/50:pi;
n = length(x);
% Data-generation function with +/-0.025 random error.
y = sin(x)+.05*(rand(1,51)-0.5);
% Backward difference estimate of dy/dx.
d1 = diff(y)./diff(x);
subplot(2,1,1)
plot(x(2:n),d1,x(2:n),d1,'o')
% Central difference estimate of dy/dx.
d2 = (y(3:n)-y(1:n-2))./(x(3:n)-x(1:n-2));
subplot(2,1,2)
plot(x(2:n-1),d2,x(2:n-1),d2,'o')
```

Gradients

$$\nabla f = \frac{\partial f}{\partial x}\mathbf{i} + \frac{\partial f}{\partial y}\mathbf{j}$$

where **i** and **j** are the unit vectors in the x and y directions, respectively. The concept can be extended to functions of three or more variables.

In MATLAB the gradient of a set of data representing a two-dimensional function $f(x, y)$ can be computed with the `gradient` function. Its syntax is `[df_dx, df_dy] = gradient (f, dx, dy)`, where `df_dx` and `df_dy` represent $\partial f/\partial x$ and $\partial f/\partial y$ and `dx` and `dy` are the spacing in the x and y values

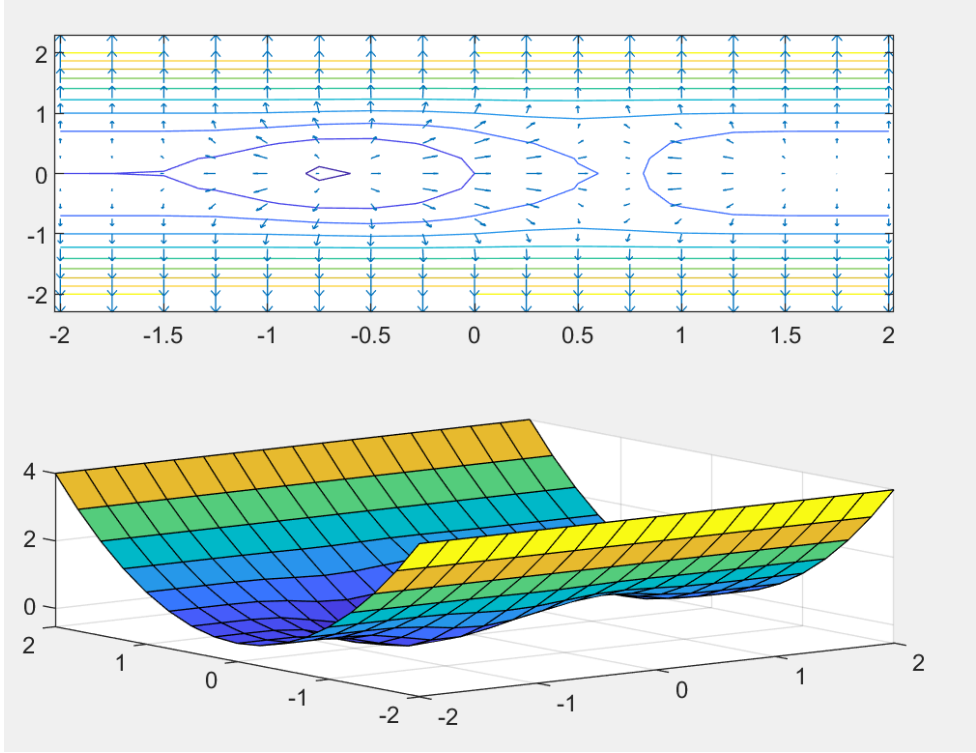


Gradient, contour, and surface plots of the function $f(x, y) = xe^{-(x^2+y^2)^2} + y^2$.

Gradients

$$f(x, y) = xe^{-(x^2+y^2)^2} + y^2$$

```
[x,y]=meshgrid(-2:0.25:2);
f=x.*exp(-(x.^2+y.^2).^2)+y.^2;
dx=x(1,2)-x(1,1);
dy=y(2,1)-y(1,1);
[df_dx,df_dy]=gradient(f,dx,dy);
subplot(2,1,1)
contour(x,y,f)
hold on
quiver(x,y,df_dx,df_dy)
hold off
Subplot(2,2,2)
Surf(x,y,f)
```



Gradient, contour, and surface plots of the function $f(x, y) = xe^{-(x^2+y^2)^2} + y^2$.

First-Order Differential Equations

An ordinary differential equation(ODE) is an equation containing ordinary derivatives of the dependent variable. An equation containing partial derivatives with respect to two or more independent variables is a partial differential equation(PDE).

$$\dot{y}(t) = \frac{dy}{dt} \qquad \ddot{y}(t) = \frac{d^2y}{dt^2}$$

The Euler Method

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

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 definition 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

$$\frac{dy}{dt} \approx \frac{y(t + \Delta t) - y(t)}{\Delta t}$$

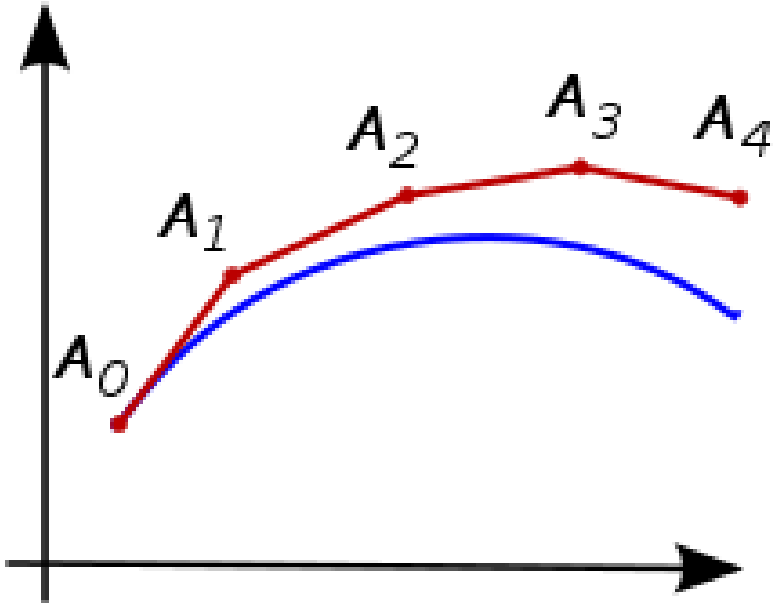
$$\frac{y(t + \Delta t) - y(t)}{\Delta t} = f(t, y) \qquad y(t + \Delta t) = y(t) + f(t, y)\Delta t$$

The Euler Method

STEP SIZE

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

$$t_{k+1} = t_k + \Delta t$$

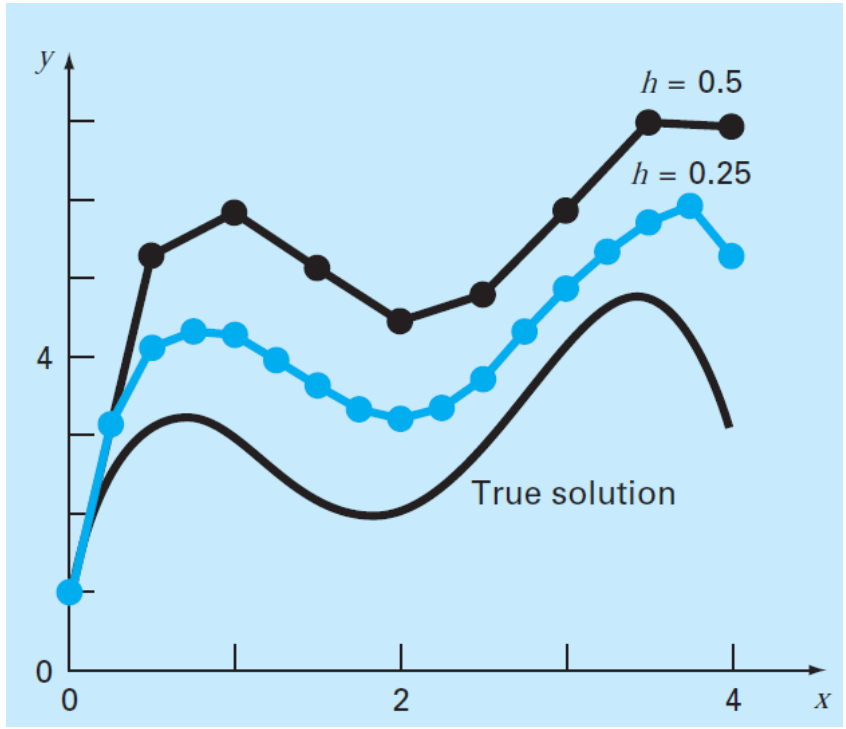


The accuracy of the Euler method can be improved sometimes by using a smaller step size. However, very small step sizes require longer run times and can result in a large accumulated error due to roundoff effects.

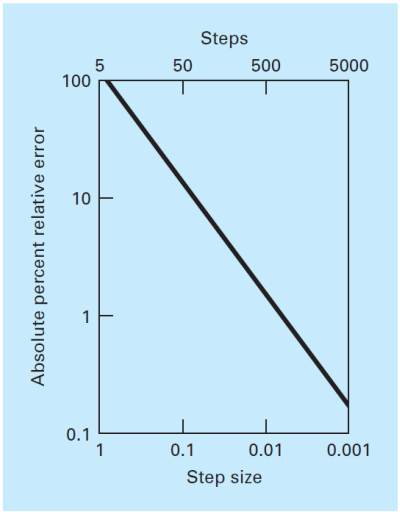
The Euler Method

TABLE 25.1 Comparison of true and approximate values of the integral of $y' = -2x^3 + 12x^2 - 20x + 8.5$, with the initial condition that $y = 1$ at $x = 0$. The approximate values were computed using Euler's method with a step size of 0.5. The local error refers to the error incurred over a single step. It is calculated with a Taylor series expansion as in Example 25.2. The global error is the total discrepancy due to past as well as present steps.

			Percent Relative Error	
x	y_{true}	y_{Euler}	Global	Local
0.0	1.00000	1.00000		
0.5	3.21875	5.25000	−63.1	−63.1
1.0	3.00000	5.87500	−95.8	−28.0
1.5	2.21875	5.12500	131.0	−1.41
2.0	2.00000	4.50000	−125.0	20.5
2.5	2.71875	4.75000	−74.7	17.3
3.0	4.00000	5.87500	46.9	4.0
3.5	4.71875	7.12500	−51.0	−11.3
4.0	3.00000	7.00000	−133.3	−53.0



Effect of step size on the global truncation error of Euler's method for the integral of $y' = -2x^3 + 12x^2 - 20x + 8.5$. The plot shows the absolute percent relative global error at $x = 5$ as a function of step size.

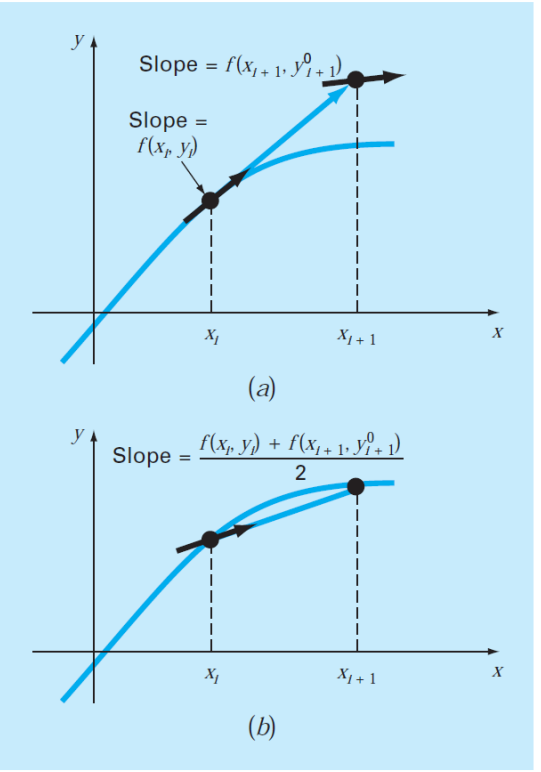


The Predictor-Corrector Method （Heun's method）

The Euler method can have a serious deficiency in problems where the variables are rapidly changing, because the method assumes the variables are constant over the time interval Δt. One way to reduce the error of Euler's method would be to include higher-order terms of the Taylor series expansion in the solution.

$$y_{i+1}^0 = y_i + f(x_i, y_i)h$$

Graphical depiction of Heun's method. (*a*) Predictor and (*b*) corrector.



Euler predictor $\quad y_{i+1}^0 = y_i + f(x_i, y_i)h \qquad E_t = \frac{f'(x_i, y_i)}{2!}h^2 + \frac{f''(x_i, y_i)}{3!}h^3 + \frac{f^{(3)}(x_i, y_i)}{4!}h^4$

Trapezoidal corrector $\quad y_{i+1} = y_i + \frac{f(x_i, y_i) + f(x_{i+1}, y_{i+1}^0)}{2}h \qquad E_t = -\frac{f''(\xi)}{12}h^3$

Heun's method

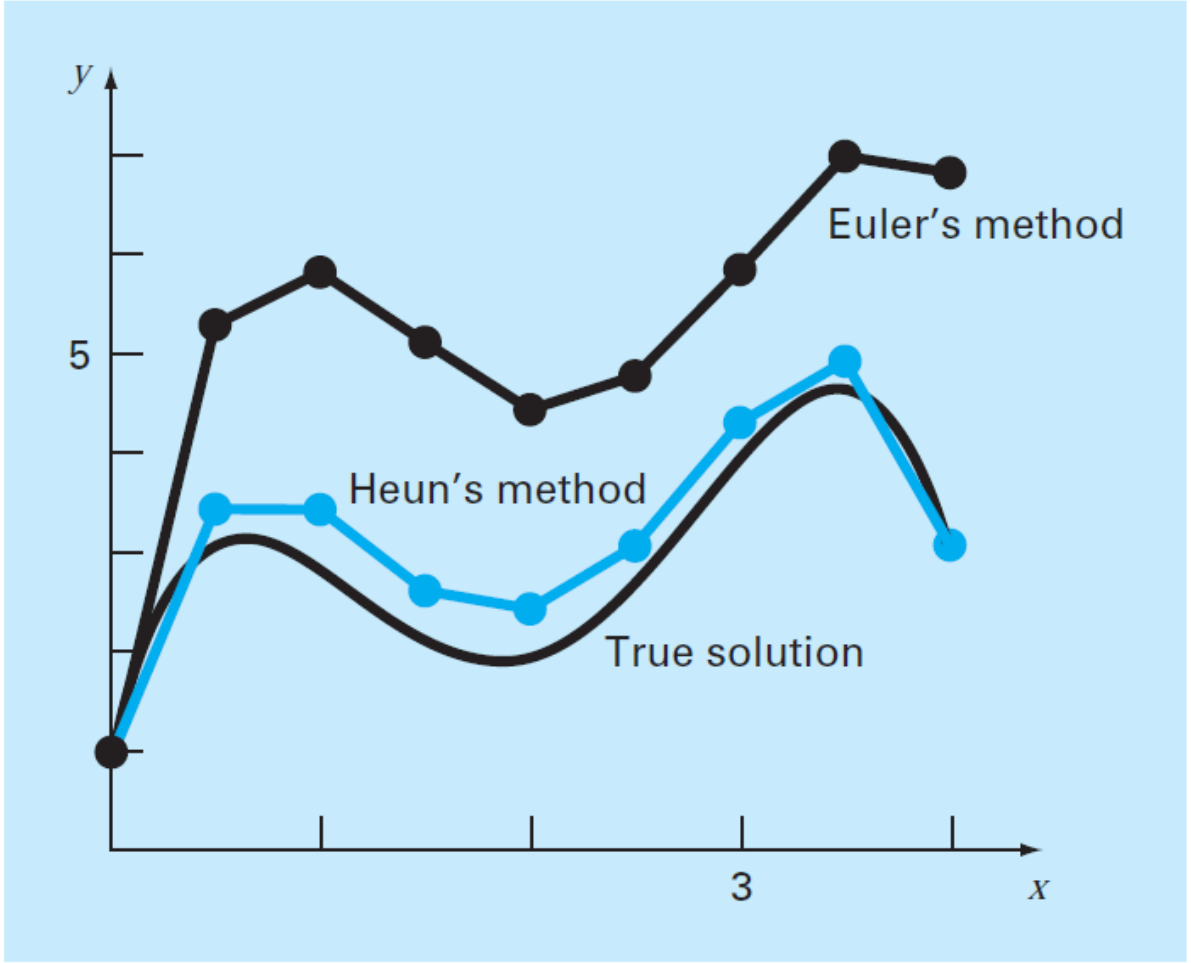


FIGURE 25.11
Comparison of the true solution with a numerical solution using Euler's and Heun's methods for the integral of $y' = -2x^3 + 12x^2 - 20x + 8.5$.

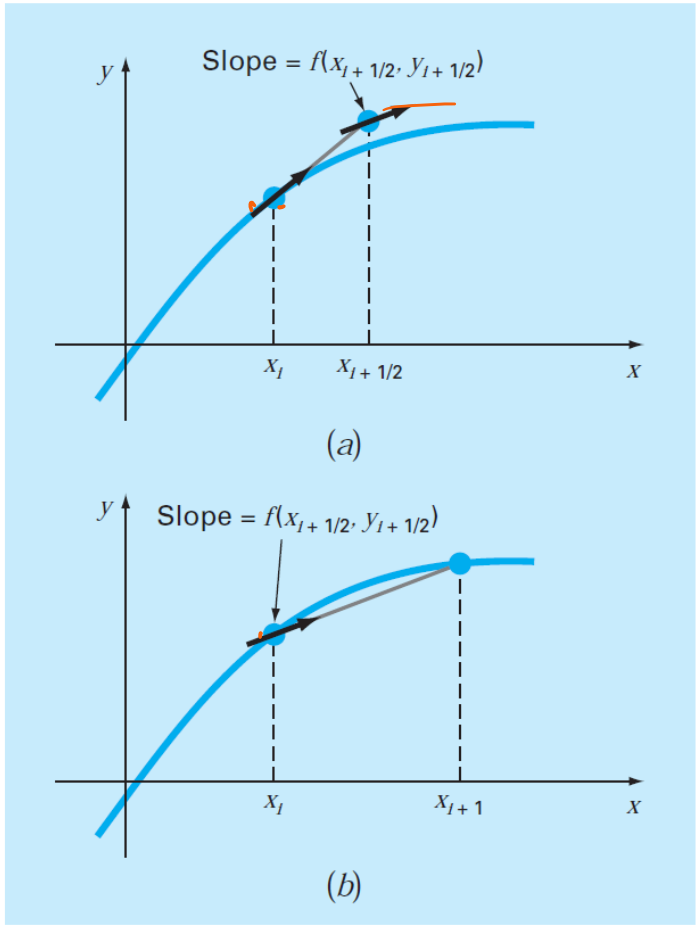
The Midpoint Method

$$y_{i+1/2} = y_i + f(x_i, y_i)\frac{h}{2}$$

$$y'_{i+1/2} = f(x_{i+1/2}, y_{i+1/2})$$

The slope at the midpoint is assumed to represent a valid approximation of the average slope for the entire interval

$$y_{i+1} = y_i + f(x_{i+1/2}, y_{i+1/2})h$$



Runge-Kutta Methods

Runge-Kutta(RK) methods achieve the accuracy of a Taylor series approach without requiring the calculation of higher derivatives.

$$y_{i+1} = y_i + \phi(x_i, y_i, h)h$$

$\phi(x_i, y_i, h)$. Increment function, which can be interpreted as a representative slope over the interval.

$$\phi = a_1 k_1 + a_2 k_2 + \cdots + a_n k_n$$

$$\begin{aligned}
k_1 &= f(x_i, y_i) \\
k_2 &= f(x_i + p_1 h, y_i + q_{11} k_1 h) \\
k_3 &= f(x_i + p_2 h, y_i + q_{21} k_1 h + q_{22} k_2 h) \\
&\vdots \\
k_n &= f(x_i + p_{n-1} h, y_i + q_{n-1,1} k_1 h + q_{n-1,2} k_2 h + \cdots + q_{n-1,n-1} k_{n-1} h)
\end{aligned}$$

Second-Order Runge-Kutta Methods

Box 25.1 Derivation of the Second-Order Runge-Kutta Methods

The second-order version of Eq. (25.28) is

$$y_{i+1} = y_i + (a_1 k_1 + a_2 k_2)h \tag{B25.1.1}$$

where

$$k_1 = f(x_i, y_i) \tag{B25.1.2}$$

and

$$k_2 = f(x_i + p_1 h, y_i + q_{11} k_1 h) \tag{B25.1.3}$$

To use Eq. (B25.1.1) we have to determine values for the constants a_1, a_2, p_1, and q_{11}. To do this, we recall that the second-order Taylor series for y_{i+1} in terms of y_i and $f(x_i, y_i)$ is written as [Eq. (25.11)]

$$y_{i+1} = y_i + f(x_i, y_i)h + \frac{f'(x_i, y_i)}{2!}h^2 \tag{B25.1.4}$$

where $f'(x_i, y_i)$ must be determined by chain-rule differentiation (Sec. 25.1.3):

$$f'(x_i, y_i) = \frac{\partial f(x, y)}{\partial x} + \frac{\partial f(x, y)}{\partial y}\frac{dy}{dx} \tag{B25.1.5}$$

Substituting Eq. (B25.1.5) into (B25.1.4) gives

$$y_{i+1} = y_i + f(x_i, y_i)h + \left(\frac{\partial f}{\partial x} + \frac{\partial f}{\partial y}\frac{dy}{dx}\right)\frac{h^2}{2!} \tag{B25.1.6}$$

The basic strategy underlying Runge-Kutta methods is to use algebraic manipulations to solve for values of a_1, a_2, p_1, and q_{11} that make Eqs. (B25.1.1) and (B25.1.6) equivalent.

To do this, we first use a Taylor series to expand Eq. (25.1.3). The Taylor series for a two-variable function is defined as [recall Eq. (4.26)]

$$g(x + r, y + s) = g(x, y) + r\frac{\partial g}{\partial x} + s\frac{\partial g}{\partial y} + \cdots$$

Applying this method to expand Eq. (B25.1.3) gives

$$f(x_i + p_1 h, y_i + q_{11} k_1 h) = f(x_i, y_i) + p_1 h\frac{\partial f}{\partial x} + q_{11} k_1 h\frac{\partial f}{\partial y} + O(h^2)$$

This result can be substituted along with Eq. (B25.1.2) into Eq. (B25.1.1) to yield

$$y_{i+1} = y_i + a_1 h f(x_i, y_i) + a_2 h f(x_i, y_i) + a_2 p_1 h^2\frac{\partial f}{\partial x} + a_2 q_{11} h^2 f(x_i, y_i)\frac{\partial f}{\partial y} + O(h^3)$$

or, by collecting terms,

$$y_{i+1} = y_i + [a_1 f(x_i, y_i) + a_2 f(x_i, y_i)]h + \left[a_2 p_1\frac{\partial f}{\partial x} + a_2 q_{11} f(x_i, y_i)\frac{\partial f}{\partial y}\right]h^2 + O(h^3) \tag{B25.1.7}$$

Now, comparing like terms in Eqs. (B25.1.6) and (B25.1.7), we determine that for the two equations to be equivalent, the following must hold:

$$a_1 + a_2 = 1$$

$$a_2 p_1 = \frac{1}{2}$$

$$a_2 q_{11} = \frac{1}{2}$$

These three simultaneous equations contain the four unknown constants. Because there is one more unknown than the number of equations, there is no unique set of constants that satisfy the equations. However, by assuming a value for one of the constants, we can determine the other three. Consequently, there is a family of second-order methods rather than a single version.

Second-Order Runge-Kutta Methods

Heun Method with a Single Corrector ($a_2 = 1/2$).

$$y_{i+1} = y_i + \left(\frac{1}{2}k_1 + \frac{1}{2}k_2\right)h$$

$$k_1 = f(x_i, y_i)$$

$$k_2 = f\left(x_i + \frac{1}{2}h, y_i + \frac{1}{2}k_1 h\right)$$

The Midpoint Method ($a_2 = 1$).

$$y_{i+1} = y_i + k_2 h$$

$$k_1 = f(x_i, y_i)$$

$$k_2 = f\left(x_i + \frac{1}{2}h, y_i + \frac{1}{2}k_1 h\right)$$

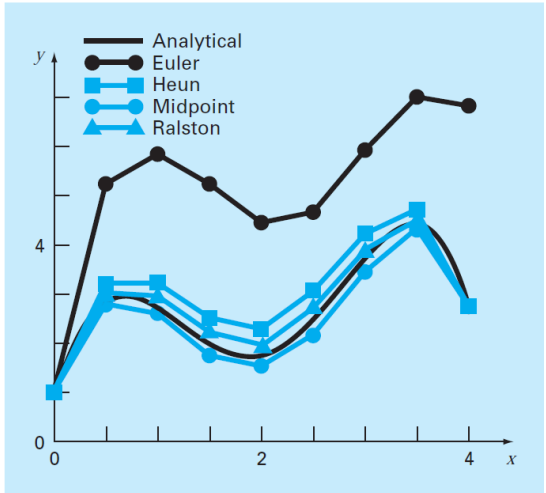
Ralston's Method ($a_2 = 2/3$).

$$y_{i+1} = y_i + \left(\frac{1}{3}k_1 + \frac{2}{3}k_2\right)h$$

$$k_1 = f(x_i, y_i)$$

$$k_2 = f\left(x_i + \frac{3}{4}h, y_i + \frac{3}{4}k_1 h\right)$$

Comparison of the true solution with numerical solutions using three second-order RK methods and Euler's method.



Fourth-Order Runge-Kutta Methods

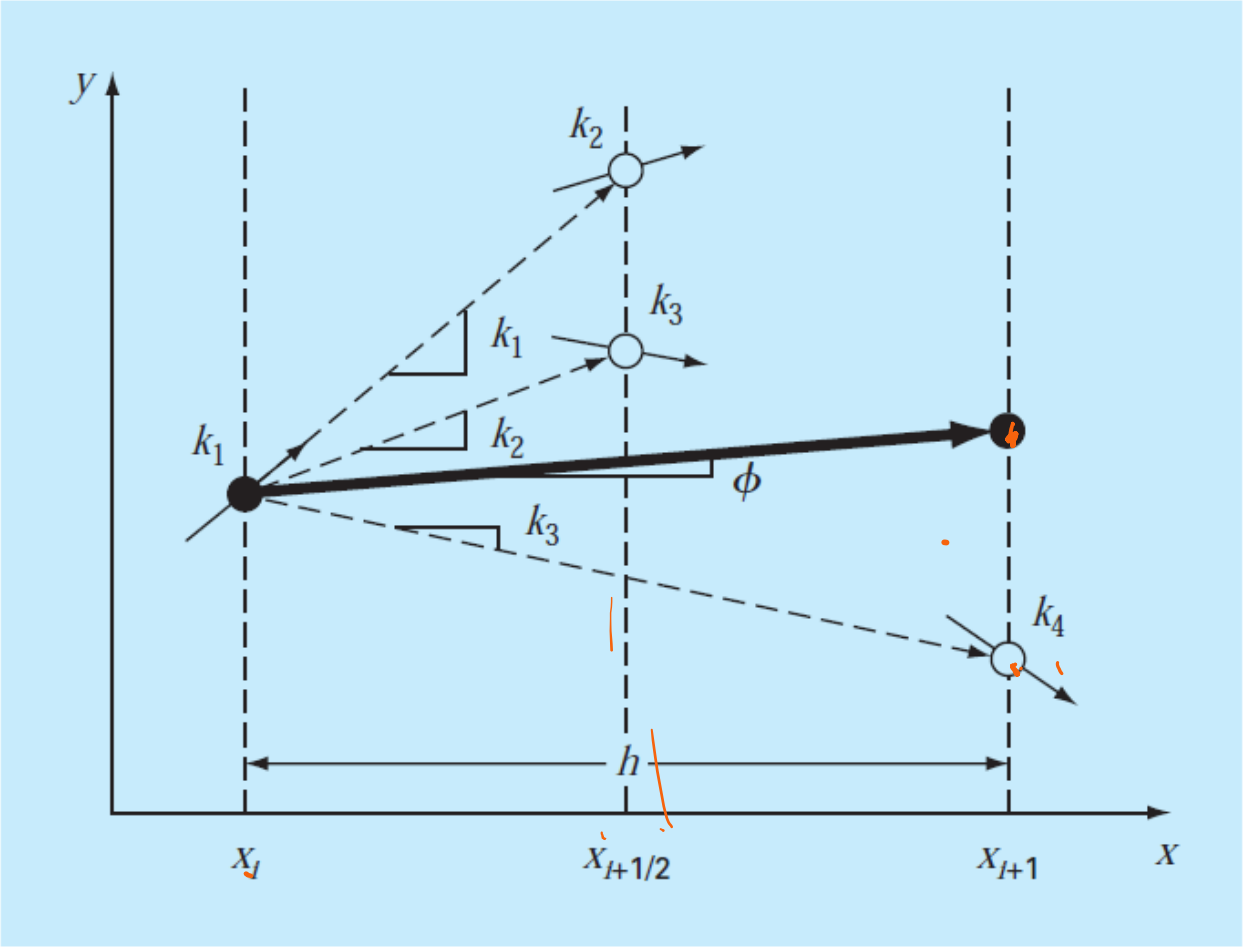
$$y_{i+1} = y_i + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)h$$

$$k_1 = f(x_i, y_i)$$

$$k_2 = f\left(x_i + \frac{1}{2}h, y_i + \frac{1}{2}k_1 h\right)$$

$$k_3 = f\left(x_i + \frac{1}{2}h, y_i + \frac{1}{2}k_2 h\right)$$

$$k_4 = f(x_i + h, y_i + k_3 h)$$



fifth-Order Runge-Kutta Methods (Butcher's method)

$$y_{i+1} = y_i + \frac{1}{90}(7k_1 + 32k_3 + 12k_4 + 32k_5 + 7k_6)h$$

where

$$k_1 = f(x_i, y_i)$$

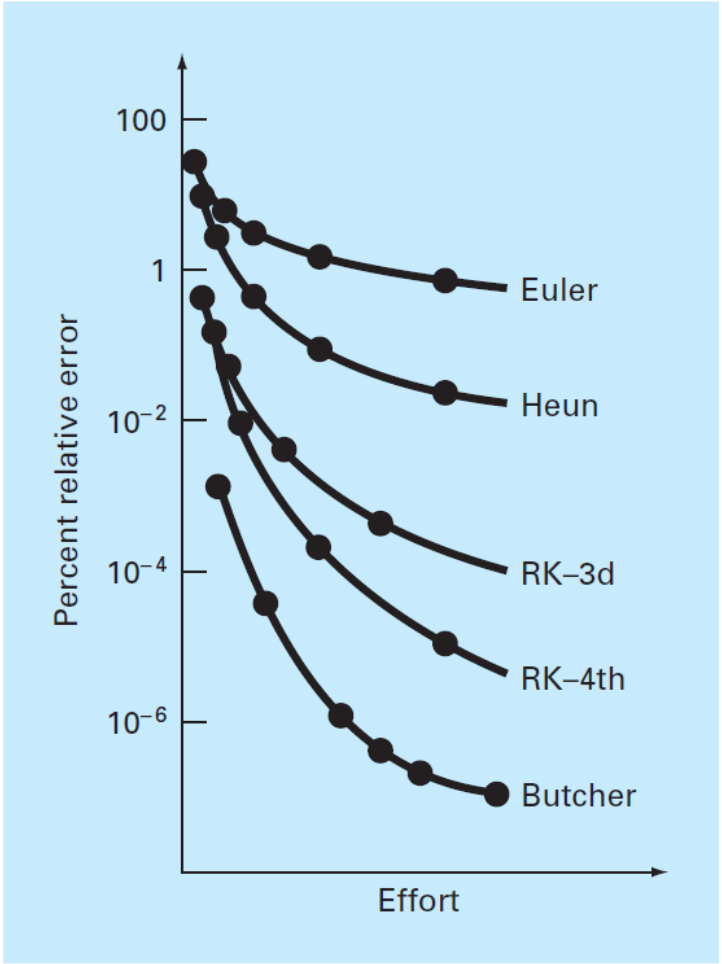
$$k_2 = f\left(x_i + \frac{1}{4}h, y_i + \frac{1}{4}k_1 h\right)$$

$$k_3 = f\left(x_i + \frac{1}{4}h, y_i + \frac{1}{8}k_1 h + \frac{1}{8}k_2 h\right)$$

$$k_4 = f\left(x_i + \frac{1}{2}h, y_i - \frac{1}{2}k_2 h + k_3 h\right)$$

$$k_5 = f\left(x_i + \frac{3}{4}h, y_i + \frac{3}{16}k_1 h + \frac{9}{16}k_4 h\right)$$

$$k_6 = f\left(x_i + h, y_i - \frac{3}{7}k_1 h + \frac{2}{7}k_2 h + \frac{12}{7}k_3 h - \frac{12}{7}k_4 h + \frac{8}{7}k_5 h\right)$$



MATLAB ODE Solvers

In addition to the many variations of the predictor-corrector and Runge-Kutta algorithms that have been developed, there are more-advanced algorithms that use a variable step size. These "adaptive" algorithms use larger step sizes when the solution is changing more slowly. MATLAB provides several functions, called *solvers*, that implement the Runge-Kutta and other methods with variable step size. Two of these are the `ode45` and `ode15s` functions. The `ode45` function uses a combination of fourth- and fth-order Runge-Kutta methods. It is a general-purpose solver whereas `ode15s` is suitable for more-dif cult equations called "stiff" equations. These solvers are more than suf cient to solve the problems in this text. It is recommended that you try `ode45` rst. If the equation proves dif cult to solve (as indicated by a lengthy solution time or by a warning or error message), then use `ode15s`.

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

Stiffness

A problem is stiff if the solution being sought varies slowly, but there are nearby solutions that vary rapidly, so the numerical method must take small steps to obtain satisfactory results.

- `F=@(t,y) y^2-y^3`
- `y0=1e-6;`
- `tfinal=2/y0;`
- `opts=odeset('reltol',1.e-5);`
- `ode45(F,[0 tfinal],y0,opts)`
- `ode23s(F,[0 tfinal],y0,opts)`

Example

$$y_{i+1} = y_i + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)h$$

- Consider an 80 kg paratrooper falling from 600 meters.
- The trooper is accelerated by gravity, but decelerated by drag on the parachute

erning Equation

paratrooper mass (kg)

- g=acceleration of gravity (m/s^2)
- V=trooper velocity (m/s)
- Initial velocity is assumed to be zero

$$m\frac{dV}{dt} = -mg - \frac{4}{15}V * |V|$$

Preparing to solve numerically

Approach

- Choose a time step
- Set the initial condition
- Run a series of steps
- Adjust time step
- Continue

- First, we put the equation in the form

$$\frac{dy}{dt} = f(t, y)$$

- For our example, the equation becomes:

$$\frac{dV}{dt} = -g - \frac{4}{15}\frac{V * |V|}{m}$$

How ode45 works

- ode45 takes two steps, one with a different error order than the other
- Then it compares results
- If they are different, time step is reduced and process is repeated
- Otherwise, time step is increased

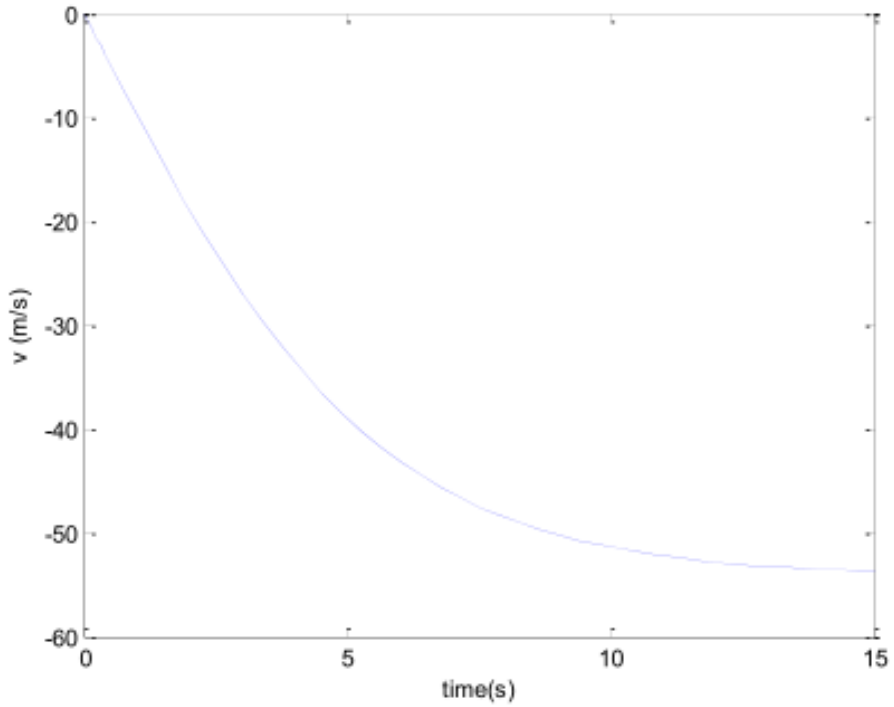
The solution

```
clear all
timerange=[0 15]; %seconds
initialvelocity=0; %meters/second
[t,y]=ode45(@f,timerange, initialvelocity)
plot(t,y)
ylabel('velocity (m/s)')
xlabel('time(s)')
```

The Function

```
function rk=f(t,y)
mass=80;
g=9.81;
rk=-g-4/15*y.*abs(y)/mass;
```

- Download the file **odeexample.m**
- Run it to reproduce my result
- Run again out to t=30 seconds
- Run again for an initial velocity of 10 meters/second
- Change to k=0 and run again (gravity only)



Practice

- The outbreak of an insect population can be modeled with the equation below.
- R=growth rate
- C=carrying capacity
- N=# of insects
- N_c=critical population
- Second term is due to bird predation

$$\frac{dN}{dt} = RN\left(1-\frac{N}{C}\right)-\frac{rN^2}{N_c^2+N^2}$$

Parameters

- 0<t<50 days
- R=0.55 /day
- N(0)=10,000
- C=10,000
- N_c=10,000
- r=10,000 /day

- What is steady state population?
- How long does it take to get there?

$$\frac{dN}{dt} = RN\left(1-\frac{N}{C}\right)-\frac{rN^2}{N_c^2+N^2}$$

- Note: this is a first order ode
- Skeleton script is in file: insects.m

Insects.m

```
function insects
clear all
tr=[0 ??];
initv=??;
[t,y]=ode45(@f, tr, initv);
plot(t,y)
ylabel('Number of Insects')
xlabel('time')
%
function rk=f(t,y)
rk= ??;
```

Practice

- Let h be the depth of water in a spherical tank
- If we open a drain at the tank bottom, the pressure at the bottom will decrease as the tank empties, so the drain rate decreases with h
- Find the time to empty the tank

Parameters

- R=5 ft; Initial height=9 ft
- 1 inch hole for drain

$$\frac{dh}{dt} = -\frac{0.0334\sqrt{h}}{10h - h^2}$$

- How long does it take to drain the tank?

Rockets

- A rocket's mass decreases as it burns fuel
- Find the final velocity of a rocket if:
- T=48000 N; m_0=2200 kg
- R=0.8; g=9.81 m/s^2; b=40 s

$$m\frac{dv}{dt} = T - mg$$

$$m = m_0\left(1 - \frac{rt}{b}\right)$$

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.

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

Solve it for the highest derivative:

$$\dddot{y} = \frac{1}{5}f(t) - \frac{4}{5}\dot{y} - \frac{7}{5}\ddot{y}$$

De ne two new variables x_1 and x_2 to be y and its derivative $\dot{y}$. That is, de ne $x_1 = y$ and $x_2 = \dot{y}$. This implies that

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

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

Second Order Equations

- Consider a falling object with drag

$$\ddot{y} = -g - \frac{4}{15m}\dot{y}|\dot{y}|$$
$$y(0) = h$$
$$\dot{y}(0) = 0$$

Preparing for Solution

- We must break second order equation into set of first order equations
- We do this by introducing new variable (z=dy/dt)

$$z = \dot{y}$$
$$\dot{z} = \ddot{y}$$
$$\dot{z} = -\frac{4}{15m} z|z| - g$$

$$\dot{z} = -\frac{4}{15m} z|z| - g$$
$$\dot{y} = z$$
$$y(0) = h; \quad z(0) = 0$$

Solving

- Now we have to send a set of equations and a set of initial values to the ode45 routine
- We do this via vectors
- Let w be vector of solutions: w(1)=y and w(2)=z
- Let r be vector of equations: r(1)=dy/dt and r(2)=dz/dt

Function to Define Equation

$$\frac{dy}{dt} = z = w(2)$$

$$\frac{dz}{dt} = -\frac{4}{15m} w(2) * |w(2)| - g$$

```
function r=rkfalling(t,w)
...
r=zeros(2,1);
r(1)=w(2);
r(2)= -k*w(2).*abs(w(2))-g;
```

The Routines

```
tr=[0 15]; %seconds
initv=[600 0]; %start 600 m high
[t,y]=ode45(@rkfalling, tr, initv)
plot(t,y(:,1))
ylabel('x (m)')
xlabel('time(s)')
figure
plot(t,y(:,2))
ylabel('velocity (m/s)')
xlabel('time(s)')
```

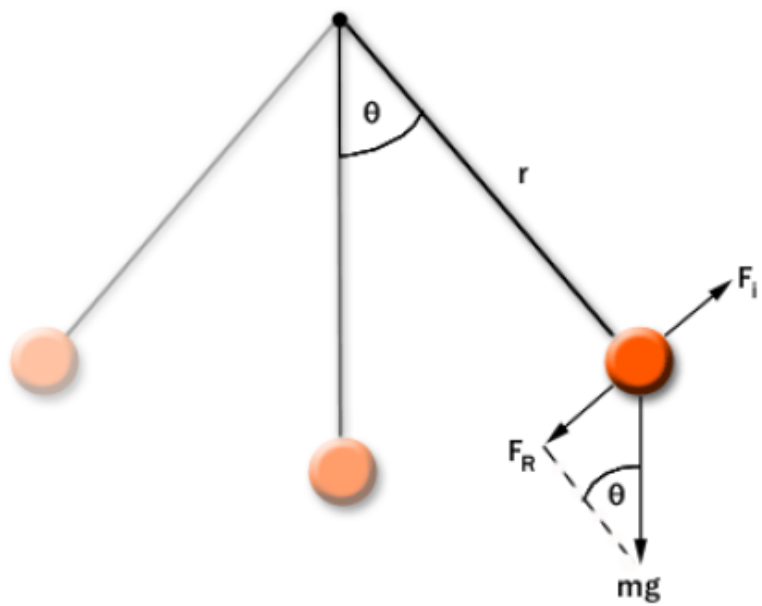
```
clear all
tr=[0 15]; %seconds
initv=[600 0]; %start 600 m high
[t,y]=ode45(@rkfalling, tr, initv);
plot(t,y(:,1))
ylabel('x (m)')
xlabel('time(s)')
figure
plot(t,y(:,2))
ylabel('velocity (m/s)')
xlabel('time(s)')

function r=rkfalling(t,w)
mass=80;
k=4/15/mass;
g=9.81;
r=zeros(2,1);
r(1)=w(2);
r(2)= k*w(2)^2-g;
```

Practice-nonlinear pendulum

- r=1 m; g=9.81 m/s^2
- Initial angle =π/8, π/2, π-0.1

$$\frac{d^2\theta}{dt^2} = -\frac{g}{r}\sin(\theta)$$



```
%pendulum problem
tr=[0 5]; %seconds
init=[pi/8 0]; %start at pi/8
[t,y]=ode45(@rkpend, tr, init);
plot(t,y(:,1))
ylabel('theta')
xlabel('time(s)')
hold on
init=[pi/2 0]; %start at pi/2
[t,y]=ode45(@rkpend, tr, init);
plot(t,y(:,1))
init=[pi-0.1 0]; %start at pi-0.1
[t,y]=ode45(@rkpend, tr, init);
plot(t,y(:,1))
hold off
disp('press any key to continue')
pause
```

```
function p=rkpend(t,w)
r=1;
g=9.81;
p=zeros(2,1);
p(1)=w(2);
p(2)= -g/r*sin(w(1));
```

Systems

- For systems of first order ODEs, just define both equations.

Practice

- Consider an ecosystem of rabbits r and foxes f. Rabbits are fox food.
- Start with 300 rabbits and 150 foxes
- α=0.01

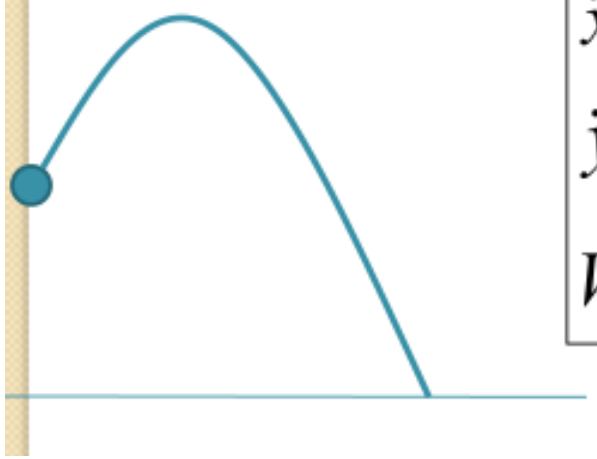
$$\frac{dr}{dt} = 2r - \alpha rf$$
$$\frac{df}{dt} = -f + \alpha rf$$

- r=w(1)
- f=w(2)

```
function z=rkfox(t,w)
alpha=0.01;
r=zeros(2,1);
z(1)=2*w(1)-alpha*w(1)*w(2);
z(2)= -w(2)+alpha*w(1)*w(2);
```

Higher Order Equations

Suppose we want to model a projectile



$$\ddot{x} = -k\,\dot{x}V$$
$$\ddot{y} = -k\,\dot{y}V - g$$
$$V = \sqrt{\dot{x}^2 + \dot{y}^2}$$

Now we need 4 1st order ODEs

$$\dot{x} = s$$
$$\dot{s} = -k\,sV$$
$$\dot{y} = z$$
$$\dot{z} = -k\,zV - g$$
$$V = \sqrt{s^2 + z^2}$$

The Code

```
clear all;
tspan=[0 1.1]
wnot(1)=0; wnot(2)=10;
wnot(3)=0; wnot(4)=10;
[t,y]=ode45('rkprojectile',tspan,wnot);
plot(t,y(:,1),t,y(:,3))
figure
plot(y(:,1),y(:,3))
```

```
function r=rkprojectile(t,w)
g=9.81;
x=w(1); s=w(2); y=w(3); z=w(4);
vel=sqrt(s.^2+z.^2);
r=zeros(4,1);
r(1)=s;
r(2)=-s*vel;
r(3)=z;
r(4)=-z*vel-g;
```