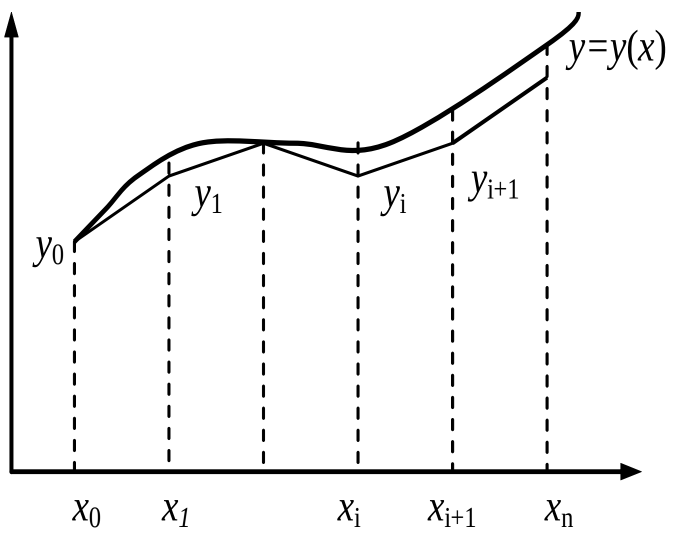
Solving Ordinary Differential Equations

Euler's method

$$\begin{cases} y_0 = \eta \\ y_{k+1} = y_k + hf(t_k, y_k), \quad k = 0, 1, \ldots, n-1 \end{cases}$$



Simplest method
Not very accurate
How to improve?

$$\boxed{\begin{cases} y_0 = \eta \\ y_{k+1} = y_k + hf(t_k, y_k) \end{cases} \quad 或 \begin{cases} y_0 = \eta \\ y_{k+1} = y_k + hf(t_{k+1}, y_{k+1}) \end{cases}}$$

$$\boxed{\begin{cases} y_0 = \eta \\ y_{k+1} = y_k + \frac{h}{2}[f(t_k, y_k) + f(t_{k+1}, y_{k+1})] \end{cases}}$$

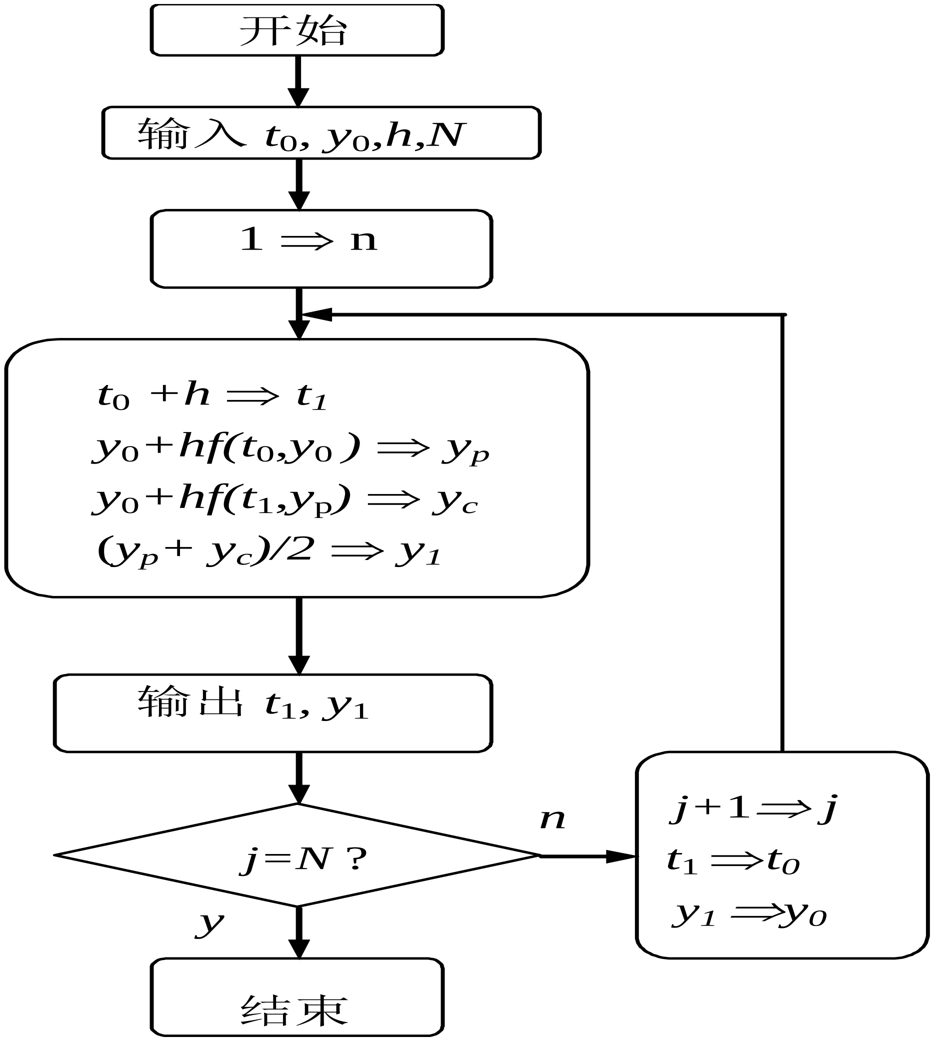
Eq. (4.10)

Initial guess?
Solve by iteration

Flow chart

Trapezoidal method

$$
\begin{cases}
y_p = y_k + hf(t_k, y_k) \\
y_c = y_k + hf(t_k, y_p) \\
y_{k+1} = \dfrac{1}{2}(y_p + y_c)
\end{cases}
$$



Predictor-Corrector Methods

- Use a less accurate algorithm (e.g., Euler) to predict y_{i+1}
- Then use a better algorithm (e.g., Picard) to compute the new y_{i+1}
- Only one iteration

$$\hat{y}_{k+1} = y_k + hf(t_k, y_k)$$

$$y_{k+1} = y_k + \frac{h}{2}[f(t_k, y_k) + f(t_{k+1}, \hat{y}_{k+1})]$$

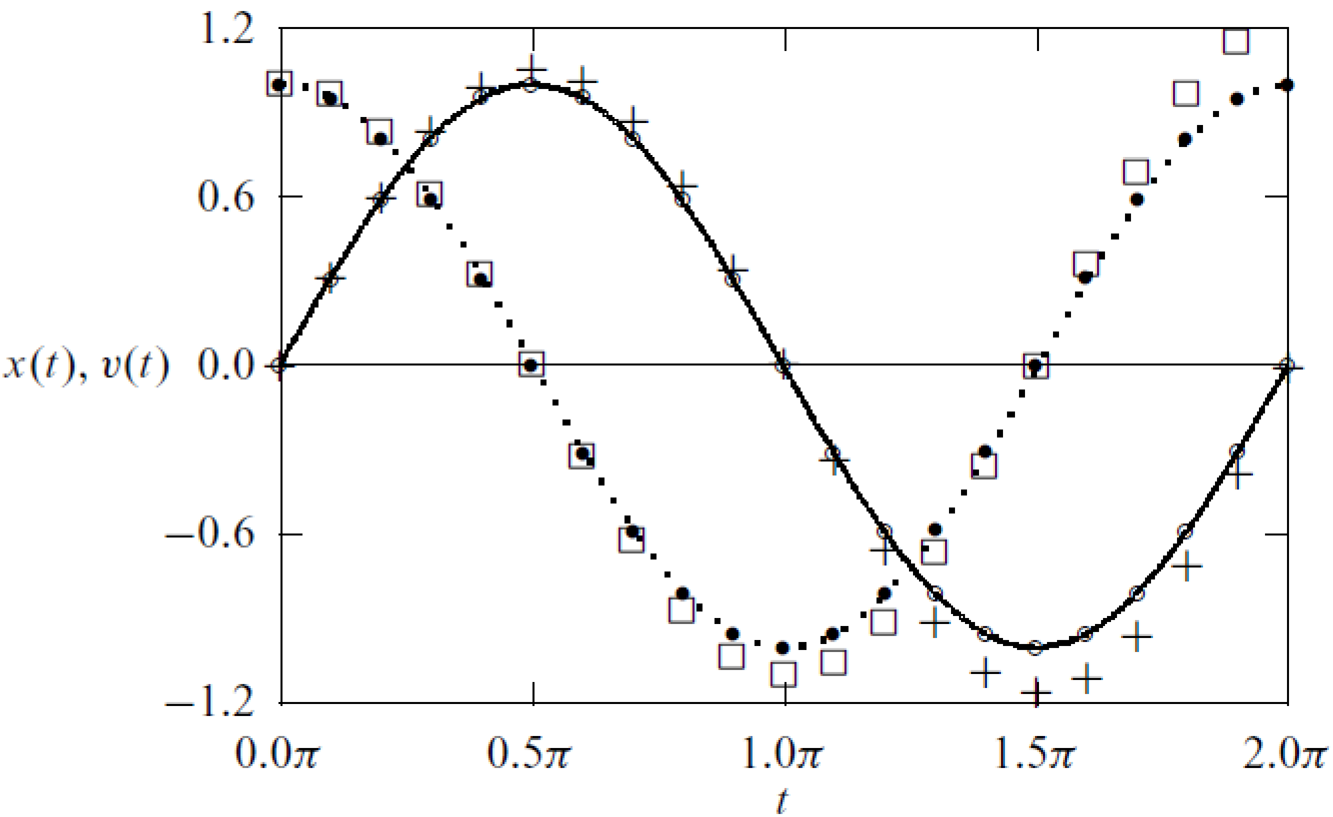


Fig. 4.1 The position (+) and velocity (□) of the particle moving in a one-dimensional space under an elastic force calculated using the Euler method with a time step of 0.02π compared with the position (○) and velocity (•) calculated with the predictor–corrector method and the exact results (solid and dotted lines).

The Runge-Kutta Method

Formally, we can expand $y(t+\tau)$ in terms of the quantities at t with the Taylor expansion

$$\begin{aligned} y(t+\tau) &= y + \tau y' + \frac{\tau^2}{2} y'' + \frac{\tau^3}{3!} y^{(3)} + \cdots \\ &= y + \tau g + \frac{\tau^2}{2}(g_t + g g_y) + \frac{\tau^3}{6}\left(g_{tt} + 2 g g_{ty} + g^2 g_{yy} + g g_y^2 + g_t g_y\right) + \cdots, \end{aligned} \tag{4.22}$$

where the subscript indices denote partial derivatives for example, $g_{yt} = \partial^2 g/\partial y \partial t$. We can also formally write the solution at $t+\tau$ as

$$y(t+\tau) = y(t) + \alpha_1 c_1 + \alpha_2 c_2 + \cdots + \alpha_m c_m, \tag{4.23}$$

with

$$\begin{aligned} c_1 &= \tau g(y, t), \\ c_2 &= \tau g(y + \nu_{21} c_1, t + \nu_{21}\tau), \\ c_3 &= \tau g(y + \nu_{31} c_1 + \nu_{32} c_2, t + \nu_{31}\tau + \nu_{32}\tau), \\ &\vdots \\ c_m &= \tau g\left(y + \sum_{i=1}^{m-1} \nu_{mi} c_i, t + \tau \sum_{i=1}^{m-1} \nu_{mi}\right), \end{aligned} \tag{4.24}$$

The Forth-Order Runge-Kutta Method

$$y(t+\tau) = y(t) + \frac{1}{6}(c_1 + 2c_2 + 2c_3 + c_4), \tag{4.33}$$

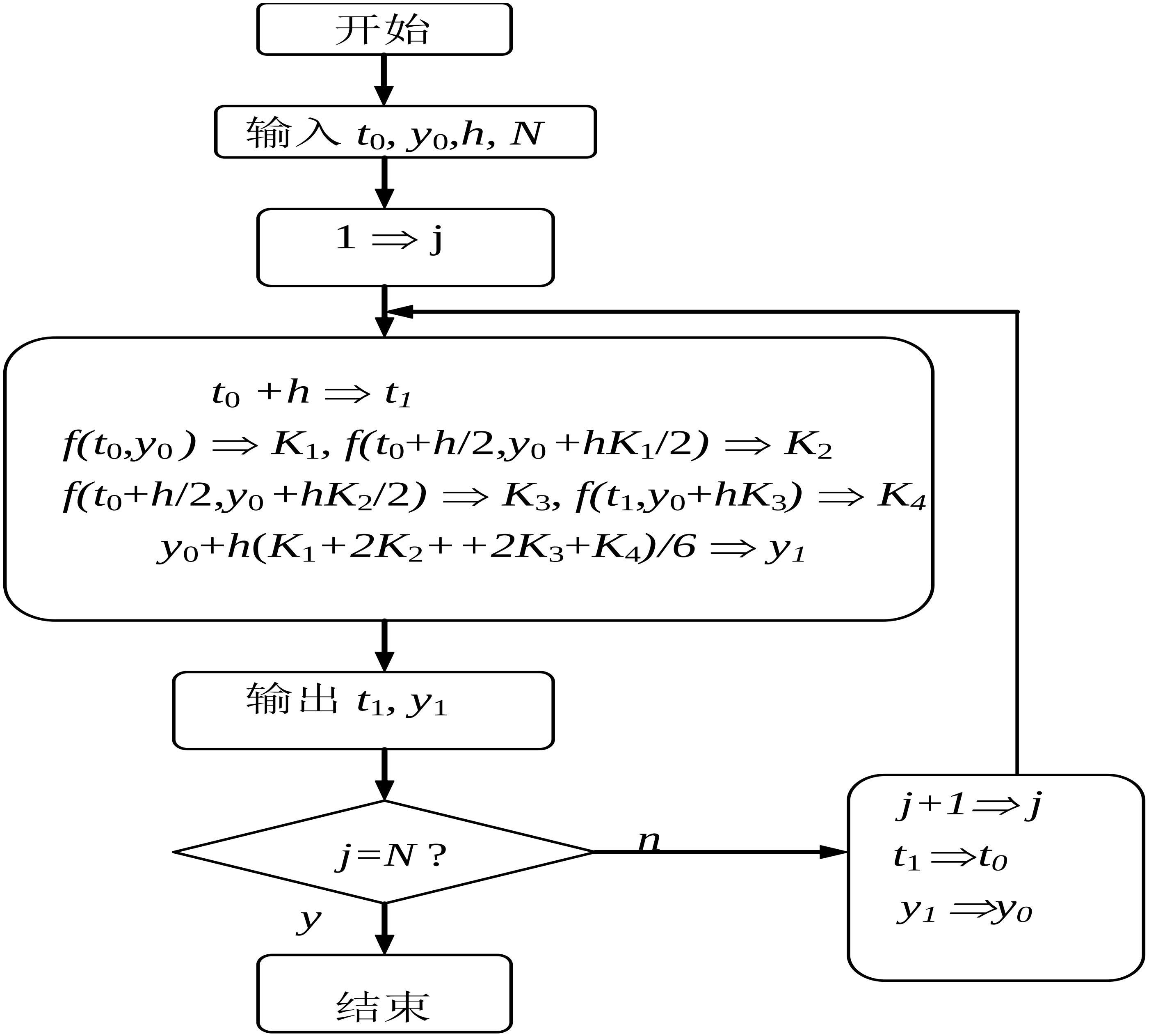
$$c_1 = \tau g(y, t), \tag{4.34}$$

$$c_2 = \tau g\left(y + \frac{c_1}{2}, t + \frac{\tau}{2}\right), \tag{4.35}$$

$$c_3 = \tau g\left(y + \frac{c_2}{2}, t + \frac{\tau}{2}\right), \tag{4.36}$$

$$c_4 = \tau g(y + c_3, t + \tau). \tag{4.37}$$

流程图


开始
输入 t_0, y_0, h, N
$1 \Rightarrow j$
$t_0 + h \Rightarrow t_1$
$f(t_0, y_0) \Rightarrow K_1$, $f(t_0+h/2, y_0+hK_1/2) \Rightarrow K_2$
$f(t_0+h/2, y_0+hK_2/2) \Rightarrow K_3$, $f(t_1, y_0+hK_3) \Rightarrow K_4$
$y_0+h(K_1+2K_2++2K_3+K_4)/6 \Rightarrow y_1$
输出 t_1, y_1
$j=N$?
n
y
$j+1 \Rightarrow j$
$t_1 \Rightarrow t_0$
$y_1 \Rightarrow y_0$
结束

```
SUBROUTINE RK4(Y,DYDX,N,X,H,YOUT,DERIVS)
```

Given values for N variables Y and their derivatives DYDX known at X, use the fourth-order Runge-Kutta method to advance the solution over an interval H and return the incremented variables as YOUT, which need not be a distinct array from Y. The user supplies the subroutine DERIVS(X,Y,DYDX) which returns derivatives DYDX at X.

```
PARAMETER (NMAX=10)              Set to the maximum number of functions
DIMENSION Y(N),DYDX(N),YOUT(N),YT(NMAX),DYT(NMAX),DYM(NMAX)
HH=H*0.5
H6=H/6.
XH=X+HH
DO 11 I=1,N                      First step
   YT(I)=Y(I)+HH*DYDX(I)
11 CONTINUE
CALL DERIVS(XH,YT,DYT)           Second step
DO 12 I=1,N
   YT(I)=Y(I)+HH*DYT(I)
12 CONTINUE
CALL DERIVS(XH,YT,DYM)           Third step
DO 13 I=1,N
   YT(I)=Y(I)+H*DYM(I)
   DYM(I)=DYT(I)+DYM(I)
13 CONTINUE
CALL DERIVS(X+H,YT,DYT)          Fourth step
DO 14 I=1,N                      Accumulate increments with proper weights.
   YOUT(I)=Y(I)+H6*(DYDX(I)+DYT(I)+2.*DYM(I))
14 CONTINUE
RETURN
END
```

Chaotic Dynamics of a Driven Pendulum

- **The motion can be either regular/periodic or**

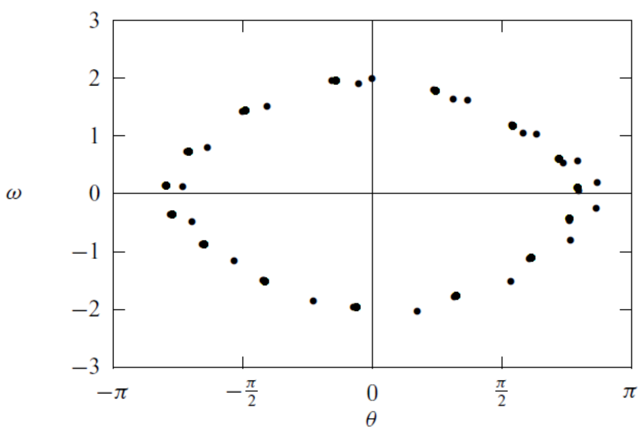
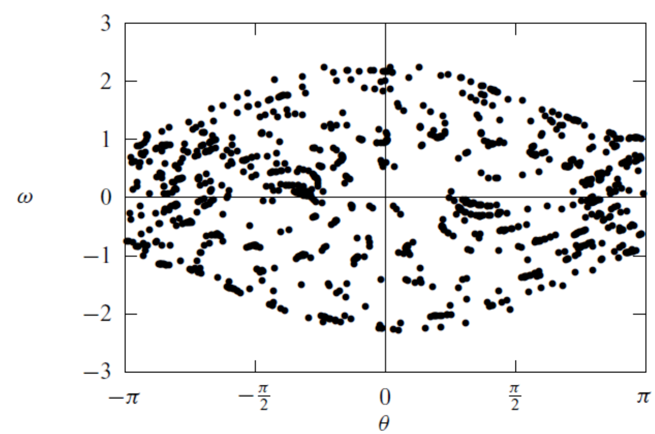


Fig. 4.4 The angular velocity ω versus the angle θ, with parameters $\omega_0 = 2/3$, $q = 0.5$, and $b = 0.9$. Under the given condition the system is apparently periodic. Here 1000 points from 10 000 time steps are shown.

Fig. 4.5 The same plot as in Fig. 4.4, with parameters $\omega_0 = 2/3$, $q = 0.5$, and $b = 1.15$. The system at this point of the parameter space is apparently chaotic. Here 1000 points from 10 000 time steps are shown.



Problem 4.5

```
// A program to study the driven pendulum under damping
// via the fourth-order Runge-Kutta algorithm.

import java.lang.*;
public class Pendulum {
  static final int n = 100, nt = 10, m = 5;
  public static void main(String argv[]) {
    double y1[] = new double[n+1];
    double y2[] = new double[n+1];
    double y[] = new double[2];

 // Set up time step and initial values
    double dt = 3*Math.PI/nt;
    y1[0] = y[0] = 0;
    y2[0] = y[1] = 2;

 // Perform the 4th-order Runge-Kutta integration
    for (int i=0; i<n; ++i) {
      double t = dt*i;
      y = rungeKutta(y, t, dt);
      y1[i+1] = y[0];
      y2[i+1] = y[1];

   // Bring theta back to the region [-pi, pi]
      int np = (int) (y1[i+1]/(2*Math.PI)+0.5);
      y1[i+1] -= 2*Math.PI*np;
    }

 // Output the result in every m time steps
    for (int i=0; i<=n; i+=m) {
      System.out.println("Angle: " + y1[i]);
      System.out.println("Angular velocity: " + y2[i]);
      System.out.println();
```

```
// Method to complete one Runge-Kutta step.

  public static double[] rungeKutta(double y[],
    double t, double dt) {
    int l = y.length;
    double c1[] = new double[l];
    double c2[] = new double[l];
    double c3[] = new double[l];
    double c4[] = new double[l];
    c1 = g(y, t);
    for (int i=0; i<l; ++i) c2[i] = y[i] + dt*c1[i]/2;
    c2 = g(c2, t+dt/2);
    for (int i=0; i<l; ++i) c3[i] = y[i] + dt*c2[i]/2;
    c3 = g(c3, t+dt/2);
    for (int i=0; i<l; ++i) c4[i] = y[i] + dt*c3[i];
    c4 = g(c4, t+dt);
    for (int i=0; i<l; ++i)
      c1[i] = y[i] + dt*(c1[i]+2*(c2[i]+c3[i])+c4[i])/6;
    return c1;
  }

// Method to provide the generalized velocity vector.

  public static double[] g(double y[], double t) {
    int l = y.length;
    double q = 0.5, b = 0.9, omega0 = 2.0/3;
    double v[] = new double[l];
    v[0] = y[1];
    v[1] = -Math.sin(y[0])+b*Math.cos(omega0*t);
    v[1] -= q*y[1];
    return v;
  }
}
```