第三章 物理学中的数值计算：求解常微分方程

物理学中的很多定律、方程都是以微分方程的形式给出来的。在计算机普及之前，能够通过解析的方法近似求解各种复杂微分方程是一个物理学家，特别是理论物理学家科研能力的重要体现之一。然而，在实际工作中，能够解析求解的微分方程非常有限。计算机的普及使得人们可以通过数值计算的方法求解各种复杂微分方程，极大地扩展了人们能够解决的实际物理与工程问题范围。微分方程可分为常微分方程和偏微分方程。常微分方程只包含一个独立变量，而偏微分方程可包含多个独立变量。本章我们重点学习常微分方程的数值解法。

2.1 RC 回路放电：求解常微分方程

图 3.1 给出的是一个 RC 回路。当单刀双掷开关置于“1”处时，电容器开始充电。当电容器充分充电后，其电量$Q_0 = CE$，其中 C 为电容器的电容，E 为直流电源的电动势。这时再将开关置于“2”处，则电容器开始放电。电容器的电量随时间的变化由如下方程给出：

$$IR + \frac{Q}{C} = 0 \qquad (3.1)$$

其中R和I为电阻的阻值和通过电阻的电流值。上式可以进一步写为：

$$\begin{cases} \frac{dQ}{dt} = -\frac{Q}{RC} \\ Q(t=0) = Q_0 \end{cases} \qquad (3.2)$$

式（3.2）给出了电容器电量 Q 随时间演化的常微分方程以及初始条件，称为初值问题。其中独立变量为时间 t。以上微分方程较为简单，可解析求解，其解析解为$Q = Q_0 e^{-\frac{t}{RC}}$。接下来，我们以此简单微分方程为例，学习数值求解常微分方程的方法。

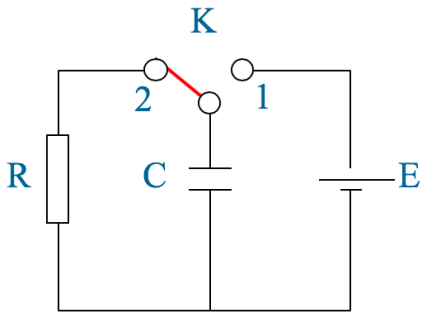

图 3.1 RC 回路电路图。

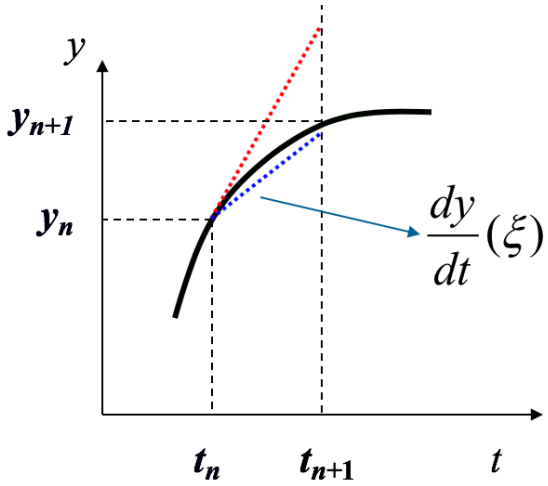

图 3.2 欧拉法示意图。

式（3.2）可以写为一般的常微分方程初值问题：

$$\begin{cases}\frac{dy}{dt}=f(y,t)\\ y(0)=y_0\end{cases} \qquad (3.3)$$

求解常微分方程的目的是得到函数y随时间t的变化规律。常用的数值求解常微分方程的方法有欧拉法、改进的欧拉法以及龙格库塔等。这些求解常微分方程的算法都是在已知当前时间点t_n的函数值y_n的条件下，求得下一时间点t_{n+1}的函数值y_{n+1}，即得到的是马尔可夫链。为此目的，我们将y_{n+1}在t_n处展开，得到：

$$y_{n+1}=y_n+\frac{dy}{dt}(t_n)\Delta t+\frac{d^2y}{dt^2}(t_n)\Delta t^2+\cdots \qquad (3.4)$$

根据中值定理，上式可以写为：

$$y_{n+1}=y_n+\frac{dy}{dt}(t_\xi)\Delta t=y_n+f\big(y(t_\xi),t_\xi\big)\Delta t \qquad (3.5)$$

其中t_ξ为介于区间(t_n,t_{n+1})的某一时刻。由于t_ξ未知，一个最直接的做法是用t_n近似地替代t_ξ（如图 3.2 所示），则(3.5)式可以写为

$$y_{n+1}=y_n+f(y_n,t_n)\Delta t \qquad (3.6)$$

式(3.6)给出了一种常微分方程的数值求解格式，称为**欧拉法**。以上欧拉法等效于将原常微分初值问题式（3.3）转化为如下差分方程：

$$\begin{cases} \frac{y_{n+1}-y_n}{\Delta t} = f(y_n, t_n) \\ y(0) = y_0 \end{cases} \quad (3.7)$$

即微分用向前差分代替，斜率$f(y,t)$用左端点的值$f(y_n,t_n)$代替。根据欧拉法公式，以上 RC 电路的放电问题可以写为：

$$\begin{cases} Q_{n+1} = Q_n - \frac{Q_n}{RC}\Delta t \\ Q(0) = Q_0 = CE \end{cases} \quad (3.8)$$

设$R = 2.0;\ C = 1.0;\ E = 1.0$, code-2-1 给出了 RC 回路放电问题的欧拉法求解过程。

```
1 # -*- coding: utf-8 -*-
2
3 import numpy as np
4 import pylab as pl
5
6 rc = 2.0
7 dt = 0.5
8 n = 5000
9 t = 0.0
10 q = 1.0
11 qt=[]
12 qt_analytic=[]
13 time = []
14
15 for i in range(n):
16     t = t + dt
17     q = q - q*dt/rc
18     q_analytic = np.exp(-t/rc)
19     qt.append(q)
20     qt_analytic.append(q_analytic)
21     time.append(t)
22
23 pl.plot(time,qt,'ro',label='Euler')
24 pl.plot(time,qt_analytic,'k-',label='Analytical')
25 pl.xlabel('Time')
26 pl.ylabel('charge')
27 pl.xlim(0,12)
28 pl.ylim(-0.2,1.0)
29 pl.legend(loc='upper right')
30 pl.show()
31
```

Code-3-1

Code-3-1 的第 6-8 行，设定了确定了基本参数，包括电阻电容值、时间步长以及最大求解步数。9-10 行是初始化，将计时器置零(t=0.0)，并给出初始电量(q=0)。11-13 行建立了三个空列表，分别存储数值求解得到的电量、解析求解得到的电量以及时间信息。15-21 行对应求解过程，其中 16 行对计时器更

新，17 行由欧拉法计算出$t+\Delta t$时刻的电量，18 行由解析公式计算出$t+\Delta t$时刻的电量，19-21 行将得到的计算结果以及时钟信息写入相应的列表中。23-30 行将得到的结果可视化。运行程序，得到的结果如图 3.3 所示。可以看出，数值计算得到的结果能够较好地与解析计算得到的精确解相吻合。其中，数值计算的精度与时间步长Δt有关。对于欧拉算法，其整体截断误差与Δt成正比。因此，减小时间步长Δt可进一步提高计算精度，但会导致计算量的增加。

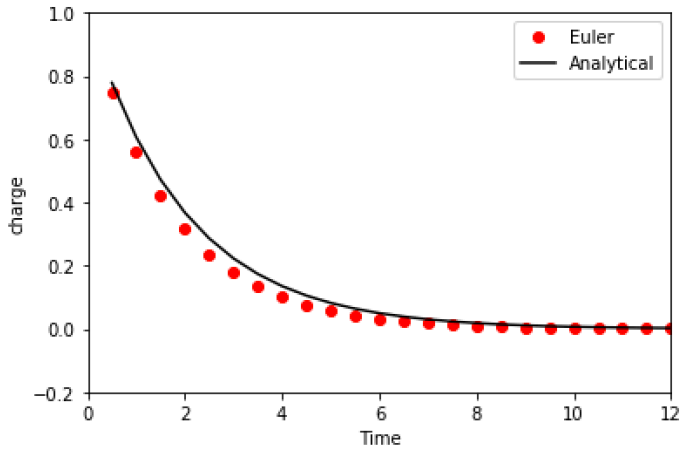

图 3.3 欧拉法（红点）与解析（黑线）得到的电容器电量随时间的变化。

在欧拉法中，我们用t_n时刻的$f(y,t)$代替t_ξ时刻的$f(y,t)$。事实上，不同的求解常微分方程的算法都是尽可能得到比较精确的$f(y_\xi,t_\xi)$，即图 3.3 的蓝色虚线斜率。作为对欧拉算法的改进，我们可以用t_n时刻和t_{n+1}时刻的$f(y,t)$的平均值来代替$f(y_\xi,t_\xi)$（如图 3.4 所示），即$f(y_\xi,t_\xi)\approx\frac{1}{2}\big(f(y_n,t_n)+f(y_{n+1},t_{n+1})\big)$，则(3.5)式可以写为

$$y_{n+1}=y_n+\frac{\Delta t}{2}\big(f(y_n,t_n)+f(y_{n+1},t_{n+1})\big) \quad (3.9)$$

式(3.9)给出了一种新的常微分方程数值求解格式，称为**改进的欧拉法**，其整体截断误差与Δt^2成正比。需要注意的是，式（3.9）的等号右侧包含待求量y_{n+1}，因此不可以直接用来计算y_{n+1}。解决这一问题的方法式是先采用欧拉法预测y_{n+1}，然后再利用式（3.9）重新计算y_{n+1}，即改进的欧拉算法包含如下两个步骤：1）预测步：$y_{n+1}=y_n+f(y_n,t_n)\Delta t$；2）矫正步：$y_{n+1}=y_n+\frac{\Delta t}{2}\big(f(y_n,t_n)+f(y_{n+1},t_{n+1})\big)$。因此，改进的欧拉法也称为**预测-矫正法**。代码 code-3-2 给出了改进的欧拉算法对 RC 回路放电过程的实现。其中第 17 行是预测步骤，采用欧拉法预测出y_{n+1}，第 18 行采用改进的欧拉算法重新计算y_{n+1}。

运行 code-3-2，得到的结果如图 3.5 所示。相对于图 3-2 中的欧拉法计算结果，尽管时间步长都取了 0.5，但是改进的欧拉算法给出的结果与解析结果更为吻合，表现出更高的精度。当然，这种精度的提高是以计算效率的降低为代价的。欧拉算法中，只需要计算一次$f(y,t)$，而改进的欧拉算法则需要计算两次$f(y,t)$，因此计算效率相比于欧拉算法降低了。

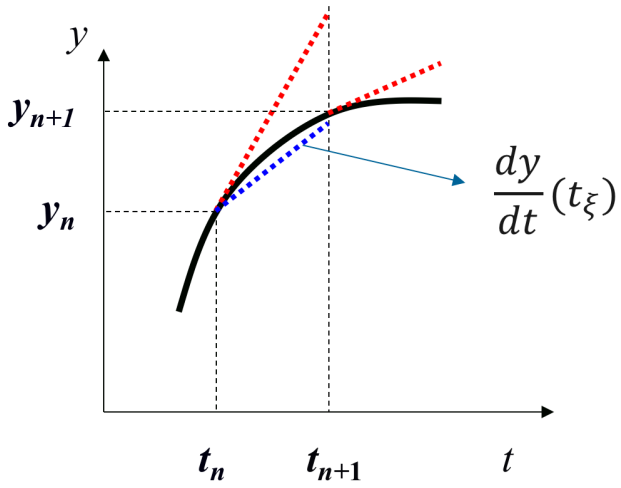

图 3.4 改进的欧拉法示意图。

```python
# -*- coding: utf-8 -*-

import numpy as np
import pylab as pl

rc = 2.0
dt = 0.5
n = 5000
t = 0.0
q = 1.0
qt=[]
qt_analytic=[]
time = []

for i in range(n):
    t = t + dt
    q1 = q - q*dt/rc
    q = q - 0.5*(q1*dt/rc + q*dt/rc)
    q_analytic = np.exp(-t/rc)
    qt.append(q)
    qt_analytic.append(q_analytic)
    time.append(t)

pl.plot(time,qt,'ro',label='Modified-Euler')
pl.plot(time,qt_analytic,'k-',label='Analytical')
pl.xlabel('Time')
pl.ylabel('charge')
pl.xlim(0,12)
pl.ylim(-0.2,1.0)
pl.legend(loc='upper right')
pl.show()

```

Code-3-2

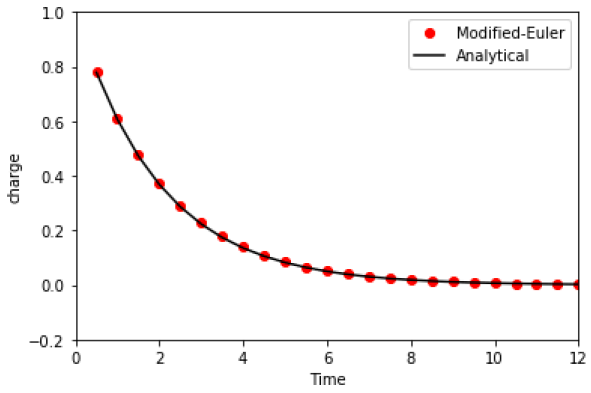

图 3.5 改进的欧拉法(红点)与解析（黑线）得到的电容器电量随时间的变化。

类似地，如果我们用$t_{n+1/2}$时刻的$f(y,t)$来代替$f(y_\xi,t_\xi)$（如图 3.6 所示），即$f(y_\xi,t_\xi)\approx f\left(y_{n+1/2},t_{n+1/2}\right)$，则(3.5)式可以写为

$$y_{n+1}=y_n+f\left(y_{n+1/2},t_{n+1/2}\right)\Delta t \quad (3.10)$$

式(3.10)给出了又一种新的常微分方程数值求解格式，称为**二阶龙格-库塔法**，其整体截断误差与Δt^2成正比。同样地，式（3.10）的等号右侧包含有未知的$y_{n+1/2}$，需要采用欧拉法先预测$y_{n+1/2}$，然后再利用式（3.10）来计算y_{n+1}。代码 code-3-3 给出了二阶龙格-库塔法的实现过程。其中第 17 行采用欧拉法预测出$y_{n+1/2}$，而第 18 行采用二阶龙格-库塔法进一步计算出y_{n+1}。运行 code-3-3 将得到与 3.5 相近的结果。

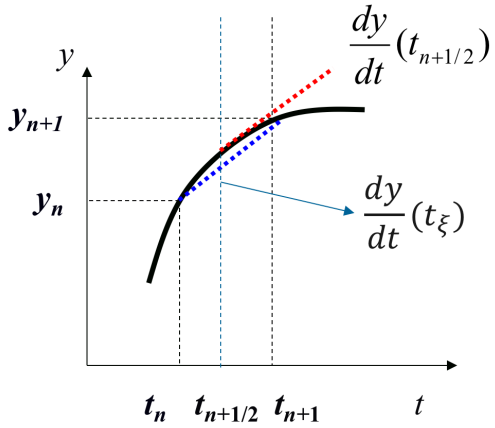

图 3.6 二阶龙格-库塔法示意图。

```python
# -*- coding: utf-8 -*-

import numpy as np
import pylab as pl

rc = 2.0
dt = 0.5
n = 5000
t = 0.0
q = 1.0
qt=[]
qt_analytic=[]
time = []

for i in range(n):
    t = t + dt
    q1 = q - q*dt*0.5/rc
    q = q - q1*dt/rc
    q_analytic = np.exp(-t/rc)
    qt.append(q)
    qt_analytic.append(q_analytic)
    time.append(t)

pl.plot(time,qt,'ro',label='2-order R-K')
pl.plot(time,qt_analytic,'k-',label='Analytical')
pl.xlabel('Time')
pl.ylabel('charge')
pl.xlim(0,12)
pl.ylim(-0.2,1.0)
pl.legend(loc='upper right')
pl.show()

```

Code-3-3

对$f(y_\xi, t_\xi)$进行更精确地估算可以得到高阶的常微分方程求解算法。例如四阶龙格-库塔法将$f(y_\xi, t_\xi)$近似为

$$\bar{k} = \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) \quad (3.11)$$

上式中，$k_1 = f(y_n, t_n)$，$k_2 = f\left(y_n + \frac{k_1 \Delta t}{2}, t_n + \frac{\Delta t}{2}\right)$，$k_3 = f\left(y_n + \frac{k_2 \Delta t}{2}, t_n + \frac{\Delta t}{2}\right)$，$k_4 = f(y_n + k_3 \Delta t, t_n + \Delta t)$。因此，四阶龙格-库塔法用$f\left(y_\xi, t_\xi\right)$的四种不同近似估算值$(k_1, k_2, k_3, k_4)$的加权平均来代替$f(y_\xi, t_\xi)$。其中，$k_1$是$f(y, t)$在左端点的值，$k_2$是由欧拉法计算出的中点处的$f(y, t)$值，$k_3$是基于$k_2$计算出的中点处的$f(y, t)$值，而$k_4$是基于$k_3$计算出的$f(y, t)$在右端点的取值（如图 3.7 所示）。于是，(3.5)式可以写为

$$y_{n+1} = y_n + \frac{\Delta t}{6}(k_1 + 2k_2 + 2k_3 + k_4) \quad (3.12)$$

式(3.12)即为**四阶龙格-库塔法**，其整体截断误差与Δt^4成正比。代码 code-3-4 给出了四阶龙格-库塔法的实现过程。其中第 18-21 行是对k_1, k_2, k_3, k_4的计算，而第 23 行采用四阶龙格-库塔法进一步计算出y_{n+1}。运行 code-3-4 将得到与图 3.4 相近的结果。

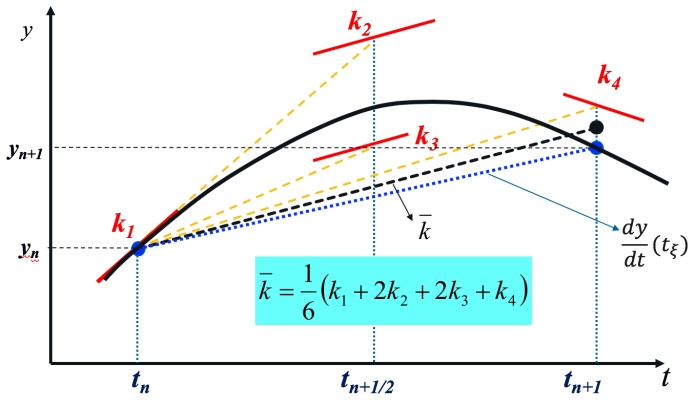

图 3.7 四阶龙格-库塔法示意图。

```
1  # -*- coding: utf-8 -*-
2
3  import numpy as np
4  import pylab as pl
5
6  rc = 2.0
7  dt = 0.5
8  n = 5000
9  t = 0.0
10 q = 1.0
11 qt=[]
12 qt_analytic=[]
13 time = []
14
15 for i in range(n):
16     t = t + dt
17
18     k1 = -q/rc
19     k2 = -(q+k1*0.5*dt)/rc
20     k3 = -(q+k2*0.5*dt)/rc
21     k4 = -(q+k3*dt)/rc
22
23     q = q + dt/6*(k1+2*k2+2*k3+k4)
24     q_analytic = np.exp(-t/rc)
25
26     qt.append(q)
27     qt_analytic.append(q_analytic)
28     time.append(t)
29
30 pl.plot(time,qt,'ro',label='4-order R-K')
31 pl.plot(time,qt_analytic,'k-',label='Analytical')
32 pl.xlabel('Time')
33 pl.ylabel('charge')
34 pl.xlim(0,12)
35 pl.ylim(-0.2,1.0)
36 pl.legend(loc='upper right')
37 pl.show()
38
```

Code-3-4

3.2 单摆运动：求解二阶常微分方程

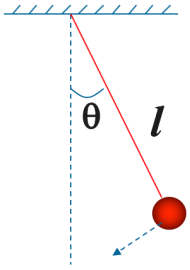

图 3.8 单摆运动示意图。

如图 3.8 给示的单摆，其摆长为l，在小摆角情况下运动方程为：

$$\begin{cases} \frac{d^2\theta}{dt^2} = -\frac{g}{l}\theta \\ \theta(0) = \theta_0; \frac{d\theta}{dt}(0) = w_0 \end{cases} \quad (3.13)$$

以上单摆运动方程为二阶常微分方程。如果引入变量$\omega = \frac{d\theta}{dt}$，则上式可以写成由两个一阶常微分方程组成的微分方程组，即：

$$\begin{cases} \frac{d\omega}{dt} = -\frac{g}{l}\theta \\ \frac{d\theta}{dt} = \omega \\ \theta(0) = \theta_0; \frac{d\theta}{dt}(0) = \omega_0 \end{cases} \quad (3.14)$$

以上方程的解析解为简谐振动$\theta = Asin(\Omega t + \phi_0)$。这里我们用四阶龙格-库塔法求解以上微分方程组。代码 code-3-5 给出了四阶龙格-库塔法对单摆运动的求解过程。其中 4-5 行给出了重力加速度常数和摆长的数值。第 6 行设定时间步长。第 7-8 行设置了初始角度和角速度。16-26 行利用四阶龙格-库塔法计算出下一时间步的角度和角速度。35 行画出角度随时间的演化轨迹，26 行画出了由角速度和角度组成的二维相平面内的轨迹。分别取初始角度为 0.1 和 0.2，运行 code-3-5，得到的结果如图 3.9 所示。可以看到，在小摆角无阻尼情况下，数值计算给出单摆的运动呈简谐振动，其振幅依赖于初始条件，初始摆角越大，振幅越大，但振动频率不变。这与解析计算结果相符。单摆在相平面中的轨迹呈闭环形状，展现出周期性运动规律。

```python
# -*- coding: utf-8 -*-
import pylab as pl

g=9.8
xl=9.8
dt=0.04
theta=0.1
omiga=0.
n = 4000
ThetaT0 = []
OmigaT0 = []
Time0 = []

t = 0.0
for i in range(n):
    xk1=-(g/xl)*theta
    xl1=omiga
    xk2=-(g/xl)*(theta+dt/2.*xl1)
    xl2=omiga+dt/2.*xk1
    xk3=-(g/xl)*(theta+dt/2.*xl2)
    xl3=omiga+dt/2.*xk2
    xk4=-(g/xl)*(theta+dt*xl3)
    xl4=omiga+dt*xk3

    omiga=omiga+dt/6.*(xk1+2*xk2+2*xk3+xk4)
    theta=theta+dt/6.*(xl1+2*xl2+2*xl3+xl4)
    t=t+dt
    ThetaT0.append(theta)
    OmigaT0.append(omiga)
    Time0.append(t)

fig = pl.figure(figsize=(10,4))
ax1 =fig.add_subplot(1,2,1)
ax2 =fig.add_subplot(1,2,2)
ax1.plot(Time0, ThetaT0, 'r-', label='theta0=0.1',linewidth=2.0)
ax2.plot(ThetaT0, OmigaT0, 'r-', label='theta0=0.1',linewidth=2.0)

pl.subplots_adjust(hspace=0.35,wspace=0.3)
ax1.set_ylabel(r'Theta', fontsize=20)
ax1.set_xlabel(r'Time', fontsize=20)
ax1.set_xlim(0,40)
ax1.set_ylim(-0.5,0.5)
ax2.set_xlabel(r'Theta', fontsize=20)
ax2.set_ylabel(r'Omega', fontsize=20)
ax2.set_xlim(-0.5,0.5)
ax2.set_ylim(-0.5,0.5)
pl.legend(loc='upper right')
#pl.show()
```

Code-3-5

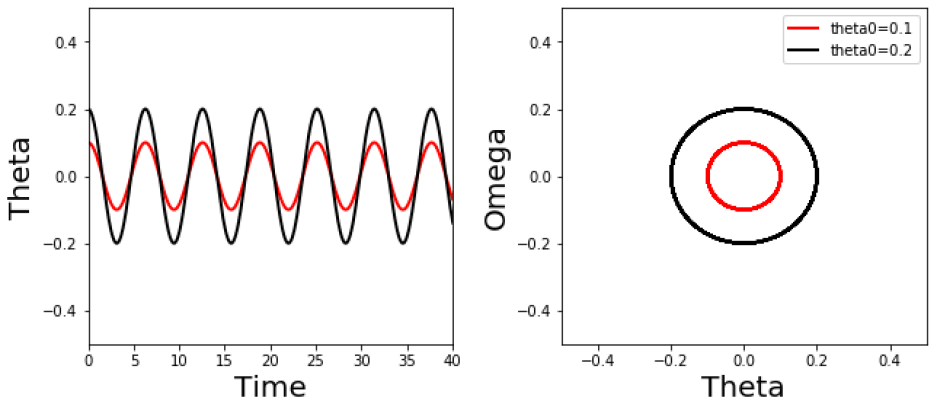

图 3.9 无阻尼小角度单摆运动轨迹和相图。

考虑单摆运动受到阻力时，通常在摆动速度较小时，阻力的大小与角速度成正比，方向与运动方向相反，即$F_{阻}=-q\omega$，其中比例系数 q 称为阻尼系数。有阻尼情况下单摆的运动方程由下式给出：

$$\begin{cases} \dfrac{d\omega}{dt}=-\dfrac{g}{l}\theta-q\omega \\ \dfrac{d\theta}{dt}=\omega \\ \theta(0)=\theta_0;\ \dfrac{d\theta}{dt}(0)=\omega_0 \end{cases} \qquad (3.15)$$

对 code-3-5 稍作改进，可得到有阻尼情况下单摆运动的代码 code-3-5。其中，第 6 行设定了阻尼系数，第 17-27 行利用四阶龙格-库塔法求出有阻尼情况下$t_n+\Delta t$时刻的角度和角速度。图 3.10 给出的是在阻尼系数分别为 0.5,1.6，5.0 时运行 code-3-5 得到的角度随时间的演化以及相平面轨迹图。在阻尼系数较小时（0.5，红线），单摆仍做简谐振动，但振幅随时间越来越小。大约 2 个振动周期后，振幅变为零，停止振动，对应的运动模式为欠阻尼振动。在阻尼系数较大时，单摆缓慢地弛豫到平衡位置，并停止运动，对应的运动模式为过阻尼振动。在阻尼系数约为 1.6 时，单摆很快地恢复到平衡位置，对应的运动模式为临界阻尼振动。在这三种情况下，由于阻尼的存在，单摆在相空间中的轨迹最后都收敛到一个不动点，即平衡位置。

```python
# -*- coding: utf-8 -*-
import pylab as pl

g=9.8
xl=9.8
q = 0.5
dt=0.04
theta=0.2
omiga=0.
n = 4000

ThetaT0 = []
OmigaT0 = []
Time0 = []
t = 0.0
for i in range(n):
    xk1=-(g/xl)*theta-q*omiga
    xl1=omiga
    xk2=-(g/xl)*(theta+dt/2.*xl1)-q*(omiga+dt/2.*xk1)
    xl2=omiga+dt/2.*xk1
    xk3=-(g/xl)*(theta+dt/2.*xl2)-q*(omiga+dt/2.*xk2)
    xl3=omiga+dt/2.*xk2
    xk4=-(g/xl)*(theta+dt*xl3)-q*(omiga+dt*xk3)
    xl4=omiga+dt*xk3

    omiga=omiga+dt/6.*(xk1+2*xk2+2*xk3+xk4)
    theta=theta+dt/6.*(xl1+2*xl2+2*xl3+xl4)
    t=t+dt
    ThetaT0.append(theta)
    OmigaT0.append(omiga)
    Time0.append(t)

fig = pl.figure(figsize=(10,4))
ax1 =fig.add_subplot(1,2,1)
ax2 =fig.add_subplot(1,2,2)

ax1.plot(Time0, ThetaT0, 'r-', label='q=0.5',linewidth=2.0)
ax2.plot(ThetaT0, OmigaT0, 'r-', label='q=0.5',linewidth=2.0)

pl.subplots_adjust(hspace=0.35,wspace=0.3)
ax1.set_ylabel(r'Theta', fontsize=20)
ax1.set_xlabel(r'Time', fontsize=20)
ax1.set_xlim(0,20)
ax1.set_ylim(-0.25,0.25)
ax2.set_xlabel(r'Theta', fontsize=20)
ax2.set_ylabel(r'Omega', fontsize=20)
ax2.set_xlim(-0.25,0.25)
ax2.set_ylim(-0.25,0.25)
pl.legend(loc='upper right')
#pl.show()
```

Code-3-6

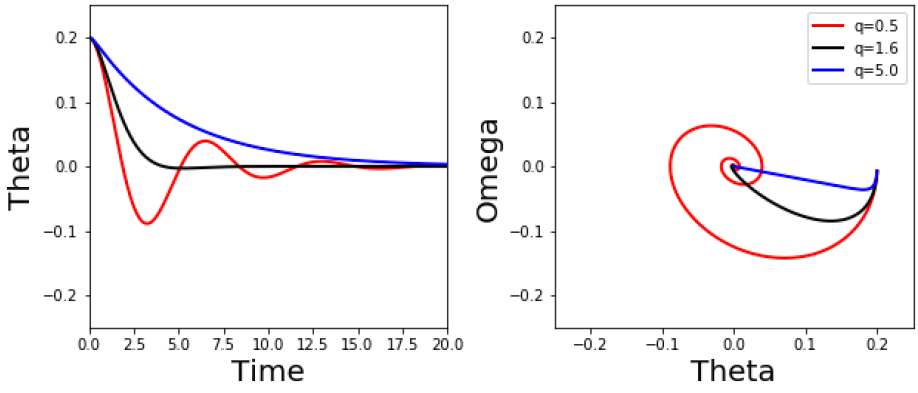

图 3.10 有阻尼情况下单摆运动轨迹和相图。

当有阻尼的单摆在周期性外力驱动下摆动时，其运动方程由下式给出：

$$\begin{cases} \frac{d\omega}{dt} = -\frac{g}{l}\theta - q\omega + F \cdot sin(Wt) \\ \frac{d\theta}{dt} = \omega \\ \theta(0) = \theta_0; \frac{d\theta}{dt}(0) = \omega_0 \end{cases} \quad (3.16)$$

其中 F 为外力强度，W 为外力频率。在 code-3-6 的基础上，第 20-27 行计算 k1, k2, k3, k4 的过程增加$F \cdot sin(Wt)$项，即得到有周期性外力驱动下的单摆运动求解代码，即 code-3-7。运行结果如图 3.11 所示。可以看到，在周期性外力驱动下，单摆的运动趋向于周期性简谐振动，其振动周期与驱动外力周期相等，而振幅依赖于多个因素，包括周期外力的强度、频率、阻尼系数以及单摆的固有频率等。在相平面中，单摆的运动轨迹收敛到极限环，显示出典型的周期运动特征。

```
# -*- coding: utf-8 -*-
import numpy as np
import pylab as pl

g=9.8
xl=9.8
q = 0.5
F=0.5
W=2.0/3.0
dt=0.04
theta=0.2
omiga=0.
n = 2000

ThetaT0 = []
OmigaT0 = []
Time0 = []
t = 0.0
for i in range(n):
    xk1=-(g/xl)*theta-q*omiga+F*np.sin(W*t)
    xl1=omiga
    xk2=-(g/xl)*(theta+dt/2.*xl1)-q*(omiga+dt/2.*xk1)+F*np.sin(W*(t+dt/2))
    xl2=omiga+dt/2.*xk1
    xk3=-(g/xl)*(theta+dt/2.*xl2)-q*(omiga+dt/2.*xk2)+F*np.sin(W*(t+dt/2))
    xl3=omiga+dt/2.*xk2
    xk4=-(g/xl)*(theta+dt*xl3)-q*(omiga+dt*xk3)+F*np.sin(W*(t+dt))
    xl4=omiga+dt*xk3

    omiga=omiga+dt/6.*(xk1+2*xk2+2*xk3+xk4)
    theta=theta+dt/6.*(xl1+2*xl2+2*xl3+xl4)
    t=t+dt
    ThetaT0.append(theta)
    OmigaT0.append(omiga)
    Time0.append(t)

fig = pl.figure(figsize=(10,4))
ax1 =fig.add_subplot(1,2,1)
ax2 =fig.add_subplot(1,2,2)

ax1.plot(Time0, ThetaT0, 'r-', label='q=0.5',linewidth=2.0)
ax2.plot(ThetaT0, OmigaT0, 'r-', label='q=0.5',linewidth=2.0)

pl.subplots_adjust(hspace=0.35,wspace=0.3)
ax1.set_ylabel(r'Theta', fontsize=20)
ax1.set_xlabel(r'Time', fontsize=20)
ax1.set_xlim(0,60)
ax1.set_ylim(-1.0,1.0)
ax2.set_xlabel(r'Theta', fontsize=20)
ax2.set_ylabel(r'Omega', fontsize=20)
ax2.set_xlim(-1.0,1.0)
ax2.set_ylim(-0.8,0.8)
pl.legend(loc='upper right')
pl.show()
```

Code-3-7

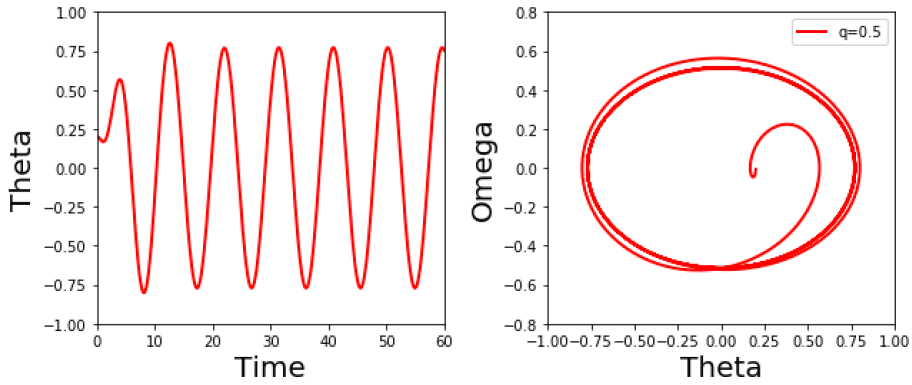

图 3.11 有阻尼和周期性外力驱动下单摆运动轨迹和相图。

以上讨论都是假设小角度运动。如果允许大角度摆动，则运动方程中的 $\sin(\theta)$不能近似为θ，运动方程如下：

$$\begin{cases} \frac{d\omega}{dt} = -\frac{g}{l}\sin(\theta) - q\omega + F \cdot sin(Wt) \\ \frac{d\theta}{dt} = \omega \\ \theta(0) = \theta_0; \frac{d\theta}{dt}(0) = \omega_0 \end{cases} \quad (3.17)$$

式（3.17）是典型的非线性常微分方程，解析求解很困难。因此，对这类方程的求解通常需要用到数值方法。Code-3-8 是对式（3.17）的求解过程。相对于 code-3-7，第 20-27 行代码中的θ用$\sin(\theta)$代替。运行结果如图 3.12 所示。可以看到，在给定的阻尼系数 q=0.5 时，单摆的运动行为依赖于周期性外力的强度。当外力强度为零时，表现为欠阻尼运动。而在外力比较弱时，表现为周期性受迫振动。在外力强度为 1.2 时，表现为非周期不规则运动，进入混沌运动区域。这种简单的非线性动态模型能够展现出复杂的混沌特性一直是令人惊讶的事情，也是 20 世纪最令人兴奋的发现之一。混沌是非线性系统普遍存在的现象。由于现实生活和实际工程技术问题通常都是非线性系统，因此混沌是无处不在的。近年来，对于混沌的研究理论已被用于处理包括湍流、复杂化学反应、股票市场、心律不齐、甚至生态系统等，取得了重要进展。

混沌的显著特征是对初始条件的极端敏感性[1]。为了展示这一特征，图 3.13 给出了在初始角度为 0 和 0.001 时，运行代码 code-3-8 得到的结果。可以看出，在短时间尺度内（Time<40），初始条件相近的两条轨迹重叠度很高，几乎用肉眼区分不出明显差别。但在长时间尺度时(Time>60)，两条轨迹逐渐分开，差别越来越大，表现出对初值的敏感性。这种运动轨迹对初值的敏感性只有在非线性系统中才会出现。我们的现实物理系统通常都是非线性体系。这种初值敏感性告诉我们，对这类非线性物理体系的长时间尺度行为的预测是困难的。例如，长时间的天气预报很难实现的主要原因就是非线性系统的混沌运动的初值敏感性。事实上，混沌现象最早是 Lorenz 在天气预报研究中发现的，并

将混沌运动的初值敏感性用蝴蝶效应来比喻，即亚马逊热带雨林中的蝴蝶偶尔扇动翅膀，可以导致两周后美国得克萨斯州的一场龙卷风。

```python
# -*- coding: utf-8 -*-
import numpy as np
import pylab as pl

g=9.8
xl=9.8
q = 0.5
F=0.5
W=2.0/3.0
dt=0.04
theta=2.5
omiga=0.
n = 5000
twopi = np.arctan(1.0)*8.0
ThetaT0 = []
OmigaT0 = []
Time0 = []
t = 0.0
for i in range(n):
    xk1=-(g/xl)*np.sin(theta)-q*omiga+F*np.sin(W*t)
    xl1=omiga
    xk2=-(g/xl)*np.sin(theta+dt/2.*xl1)-q*(omiga+dt/2.*xk1)+F*np.sin(W*(t+dt/2))
    xl2=omiga+dt/2.*xk1
    xk3=-(g/xl)*np.sin(theta+dt/2.*xl2)-q*(omiga+dt/2.*xk2)+F*np.sin(W*(t+dt/2))
    xl3=omiga+dt/2.*xk2
    xk4=-(g/xl)*np.sin(theta+dt*xl3)-q*(omiga+dt*xk3)+F*np.sin(W*(t+dt))
    xl4=omiga+dt*xk3

    omiga=omiga+dt/6.*(xk1+2*xk2+2*xk3+xk4)
    theta=theta+dt/6.*(xl1+2*xl2+2*xl3+xl4)
    if(theta > twopi/2):
        theta=theta-twopi
    if(theta < -twopi/2):
        theta=theta+twopi

    t=t+dt
    ThetaT0.append(theta)
    OmigaT0.append(omiga)
    Time0.append(t)

fig = pl.figure(figsize=(10,4))
ax1 =fig.add_subplot(1,2,1)
ax2 =fig.add_subplot(1,2,2)
ax1.plot(Time0, ThetaT0, 'r-', label='F=0.5',linewidth=2.0)
ax2.plot(ThetaT0, OmigaT0, 'r.', label='F=0.5',ms=2.0)

pl.subplots_adjust(hspace=0.35,wspace=0.3)
ax1.set_ylabel(r'Theta', fontsize=20)
ax1.set_xlabel(r'Time', fontsize=20)
ax1.set_xlim(0,60)
ax1.set_ylim(-5.0,5.0)
ax2.set_xlabel(r'Theta', fontsize=20)
ax2.set_ylabel(r'OmigaT', fontsize=20)
ax2.set_xlim(-3.5,3.5)
ax2.set_ylim(-3.5,3.5)
pl.legend(loc='upper right')
pl.show()
```

Code-3-8

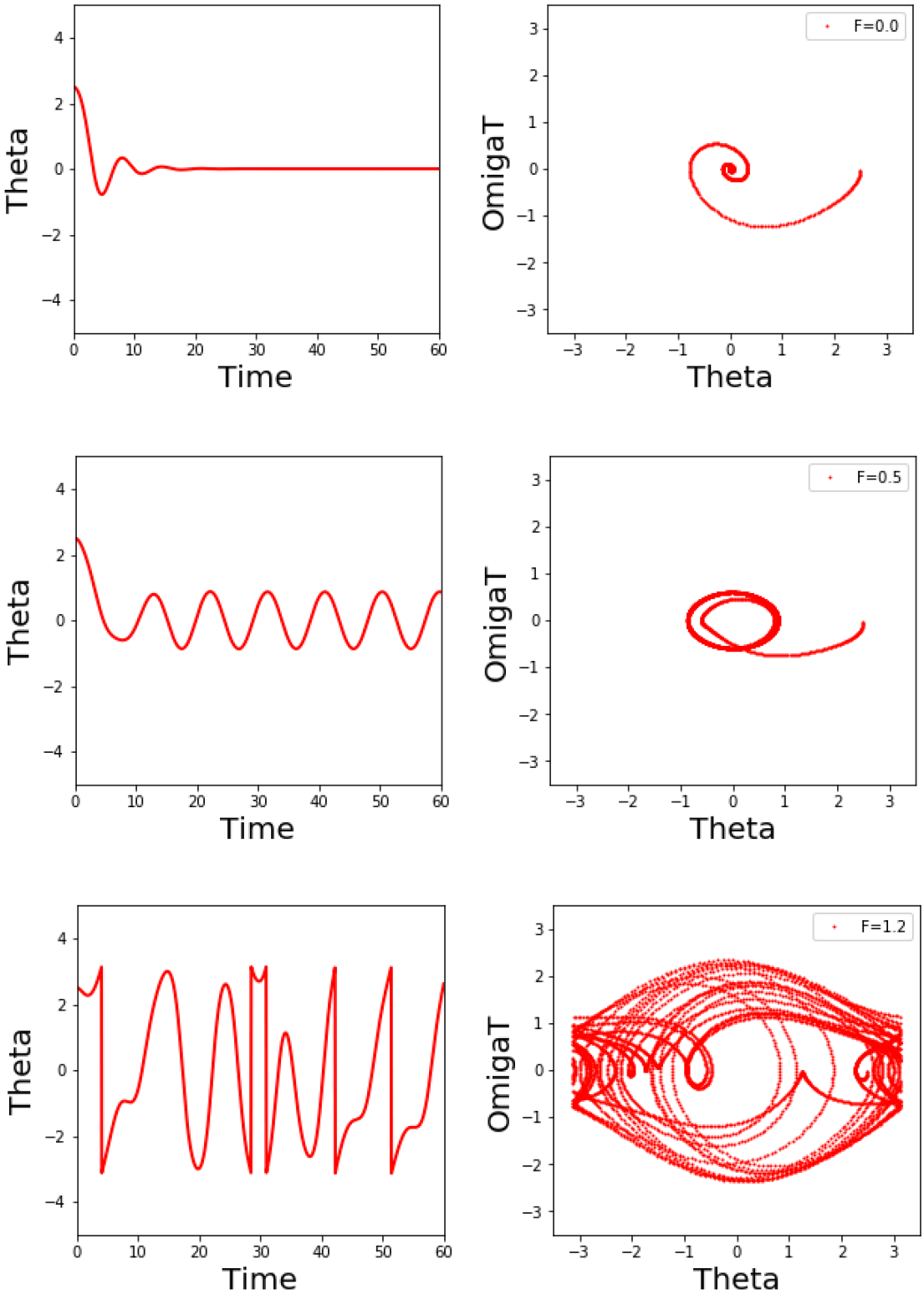

图 3.12 不同外力强度下单摆的运动轨迹和相图（上图：F=0;中图：F=0.5;下图：F=1.2）。

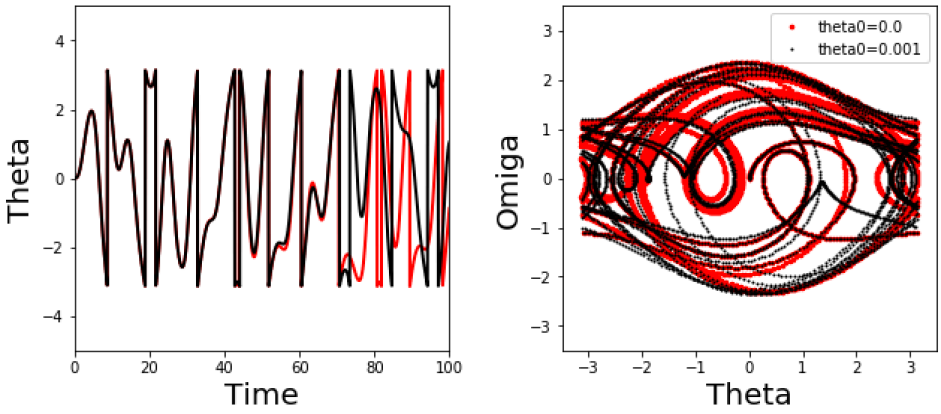

图 3.13 单摆运动轨迹对初值的敏感性。

```python
# -*- coding: utf-8 -*-
import numpy as np
import pylab as pl

g=9.8
xl=9.8
q = 0.5
F=1.2
W=2.0/3.0
dt=0.04
theta=0.0
omiga=0.
n = 5000000
twopi = np.arctan(1.0)*8.0
ThetaT0 = []
OmigaT0 = []
Time0 = []
t = 0.0
for i in range(n):
    xk1=-(g/xl)*np.sin(theta)-q*omiga+F*np.sin(W*t)
    xl1=omiga
    xk2=-(g/xl)*np.sin(theta+dt/2.*xl1)-q*(omiga+dt/2.*xk1)+F*np.sin(W*(t+dt/2))
    xl2=omiga+dt/2.*xk1
    xk3=-(g/xl)*np.sin(theta+dt/2.*xl2)-q*(omiga+dt/2.*xk2)+F*np.sin(W*(t+dt/2))
    xl3=omiga+dt/2.*xk2
    xk4=-(g/xl)*np.sin(theta+dt*xl3)-q*(omiga+dt*xk3)+F*np.sin(W*(t+dt))
    xl4=omiga+dt*xk3

    omiga=omiga+dt/6.*(xk1+2*xk2+2*xk3+xk4)
    theta=theta+dt/6.*(xl1+2*xl2+2*xl3+xl4)

    if(theta > 3.1415927):
        theta=theta-3.1415927*2.
    if(theta < -3.1415927):
        theta=theta+3.1415927*2.

    t=t+dt
    if abs(W*t/twopi-int(W*t/twopi+1.0e-6))<0.005:

       ThetaT0.append(theta)
       OmigaT0.append(omiga)
       Time0.append(t)

fig = pl.figure(figsize=(5,5))
pl.plot(ThetaT0, OmigaT0, 'r.', label='theta0=0.0',ms=3.0,linewidth=0.5)
pl.ylabel(r'Omega', fontsize=20)
pl.xlabel(r'Theta', fontsize=20)
pl.show()
```

Code-3-9

需要注意的是混沌运动与随机运动是不同的概念。混沌运动是由于非线性系统运动轨迹对初始条件的敏感性导致的，支配其运动的仍然是确定性方程。而随机运动是非确定性的。另外，混动运动轨迹并非完全无规则。描述混沌运动相平面轨迹的一个重要工具是庞加莱截面，即按驱动外力的周期记录相平面中的轨迹（通常只记录去除初始暂态阶段的稳态轨迹），每个周期记录一次相平面位置，所画出的轨迹即为庞加莱截面。对于周期性运动，庞加莱截面只有一个不动点和少数离散点；准周期运动的庞加莱截面呈一封闭曲线；而处于混沌状态的庞加莱截面呈现出具有层次结构的密集点。

Code-3-9 给出的是庞加莱截面的构建过程。与 code-3-8 的区别只是在记录轨迹时增加了条件：即第 39 行设定当 W*t 为2π的整数倍时记录轨迹。运行 code-3-9 得到的结果如图 3.14 所示。可以看到，尽管混沌运动表现为不可预测性，但是在庞加莱截面上仍可以看出一些规则。例如，运动轨迹只有在相平面特定的区域才会出现，而在一些区域轨迹重来未出现，称为禁戒区域。另外，庞加莱截面具有分形特征，即在小尺度范围看到的形貌与大尺度范围看到的形貌具有相似性。庞加莱截面的分形结构是混沌运动的又一重要特征。

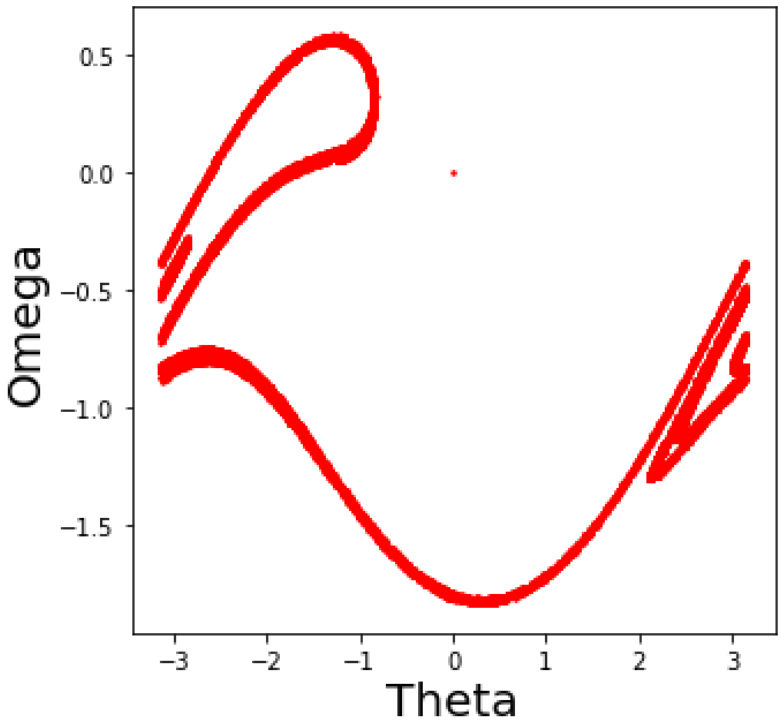

图 3.14 单摆运动的庞加莱截面。

从以上的讨论可知，非线性耗散系统在外力强度逐渐增加时，其运动轨迹由规则运动变为混沌运动。人们关注的一个重要问题是这类系统如何随控制参量的变化由规则运动进入混沌运动。为此，我们可以在不同的外力强度下观察系统运动轨迹特征。在 Code-3-10 中，我们在 0.5 到 2.0 的外力强度区间内，以 0.01 的间隔增加外力强度，在不同的外力强度下计算运动轨迹，并每隔一个外力周期记录一次θ角的位置。显然，如果运动轨迹具有固定的周期，则每一个外力周期

记录得到的θ角的位置是重叠的，表现为一个点。类似地，如果运动轨迹是无规则的（或具有无限长周期），则每一个外力周期记录得到的θ角的位置呈条带状。运行 code-3-10 得到的结果如图 3.15 所示。可以看到，单摆的运动通过倍周期分岔的途径由周期性运动过渡到混沌运动。例如，在 F<1.0 时，角度θ在不同的外力周期出现在相同的位置，表现为单周期运动。而在 1.0<F<1.08 范围，角度θ在不同的外力周期出现在两个固定的位置，表现为两周期运动。而在 1.08<F<1.1 范围，运动轨迹由两周期变为四周期、八周期等，最后到达混沌运动区间。除倍周期分岔途径外，非线性系统还可以通过阵发途径和准周期途径由规则运动过渡到混沌运动。

```python
# -*- coding: utf-8 -*-
import numpy as np
import pylab as pl

g=9.8
xl=9.8
q = 0.5
W=2.0/3.0
dt=0.01
n = 100000
twopi = np.arctan(1.0)*8.0

ThetaT0 = []
OmigaT0 = []
F0 = []
for F in np.arange(0.5,2.0,0.01):
    theta=0.1
    omiga=0.1
    t=0.0
    for i in range(n):
        xk1=-(g/xl)*np.sin(theta)-q*omiga+F*np.sin(W*t)
        xl1=omiga
        xk2=-(g/xl)*np.sin(theta+dt/2.*xl1)-q*(omiga+dt/2.*xk1)+F*np.sin(W*(t+dt/2))
        xl2=omiga+dt/2.*xk1
        xk3=-(g/xl)*np.sin(theta+dt/2.*xl2)-q*(omiga+dt/2.*xk2)+F*np.sin(W*(t+dt/2))
        xl3=omiga+dt/2.*xk2
        xk4=-(g/xl)*np.sin(theta+dt*xl3)-q*(omiga+dt*xk3)+F*np.sin(W*(t+dt))
        xl4=omiga+dt*xk3

        omiga=omiga+dt/6.*(xk1+2*xk2+2*xk3+xk4)
        theta=theta+dt/6.*(xl1+2*xl2+2*xl3+xl4)

        if(theta > 3.1415927):
            theta=theta-3.1415927*2.
        if(theta < -3.1415927):
            theta=theta+3.1415927*2.
        t=t+dt

        if abs(W*t/twopi-int(W*t/twopi+1.0e-6))<0.005 and t>100:
            ThetaT0.append(theta)
            OmigaT0.append(omiga)
            F0.append(F)

fig = pl.figure(figsize=(8,5))
pl.plot(F0, ThetaT0, 'r.', label='F=0.0',ms=3.0)

pl.ylabel(r'Theta', fontsize=20)
pl.xlabel(r'F', fontsize=20)
pl.show()
```

Code-3-10

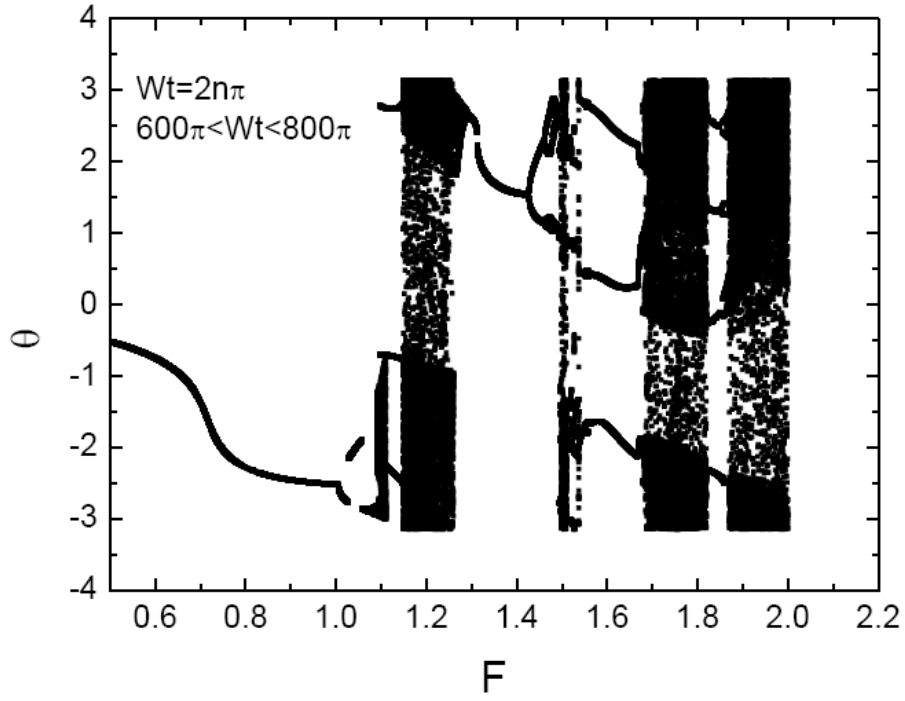

图 3.15 单摆运动的倍周期分岔图。

3.3 生物钟：包含多个方程的常微分方程组

生物体 24 小时生物钟是我们非常熟悉的生物现象。例如我们的睡眠、体温、情绪等都表现出 24 小时的生物节律。事实上，任何生物，包括植物、动物、甚至单细胞生物都有 24 小时生物钟现象，这是因为细胞层次存在 24 小时的内禀振荡。关于 24 小时生物节律产生的分子机制一直是人们感兴趣的问题。目前，人们已经清楚，这种 24 小时生物钟现象是由细胞内具有负反馈回路特性的基因调控网络产生的。

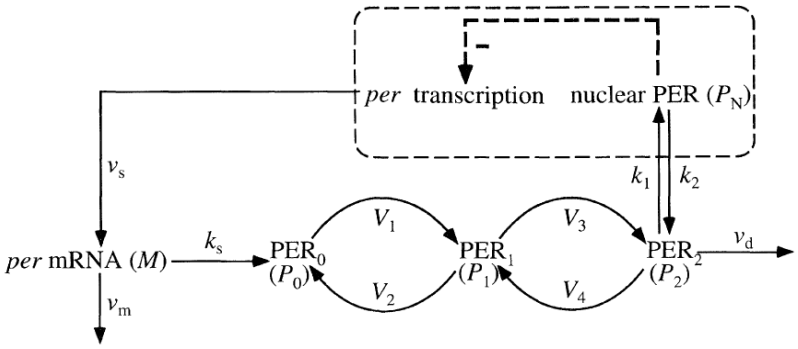

图 3.16 生物钟相关基因调控网络示意图。

1995 年，Goldbeter 提出一个简单的模型来描述 24 小时生物节律的产生，这样模型考虑了周期蛋白 PER 的表达、磷酸化、降解、入核等步骤（如图 3.16 所示）[2]。具体地，编码周期蛋白 PER 的 mRNA 表达产生周期蛋白PER_0。然后，PER_0在激酶的催化下经过连续两次磷酸化反应（磷酸化反应指的是对 PER 蛋白的特定氨基酸进行磷酸化修饰），分别产生PER_1和PER_2。PER_2进入细胞核，抑制PER蛋白的基因转录，产生 mRNA。其中，mRNA 和PER_2可发生降解。因此，整个过程形成一个负反馈回路，其动力学行为受到相关参数的调节，包括转录速率v_s,翻译速率k_s， 两次磷酸化速率V_1和V_3，及其逆反应速率V_2和V_4，PER_2入核和出核速率k_1和k_2，以及 mRNA 与PER_2的降解速率v_m和v_d等。根据化学反应动力学理论，如果反应过程是酶催化反应，则反应速率由米氏方程（Michaelis-Menten 方程）描述：

$$v = \frac{v_{max}[S]}{K_M+[S]} \quad (3.18)$$

其中，v_{max}为饱和底物浓度下的反应速率，即最大反应速率。K_M为描述底物分子与酶结合强度的米氏常数，[S]为催化反应的底物分子浓度。v_{max}和K_M对给定的催化反应是基本参数，可由实验测定。底物浓度[S]是可变量。在细胞周期基因调控网络中，磷酸化和降解过程皆为酶催化下的反应，可由米氏方程描述。另外，如果一个动力学过程需要受到 n（n=4 对应形成四聚体）个单体组成复合体的调控，则其速率与调控因子浓度呈 n 次方的关系。例如，细胞核内 n 个PER_2共同作用，抑制PER基因的转录，则转录速率由如下函数描述：

$$v = \frac{v_{max}K^n}{K^n+[A]^n} \quad (3.19)$$

其中K为解离常数。设 mRNA，PER_0，PER_1，PER_2和核内PER_2的浓度为M，P_0，P_1，P_2，和P_N，则各组分的浓度变化由以下微分方程组出：

$$\frac{dM}{dt} = v_s \frac{K_I^n}{K_I^n+P_N^n} - v_m \frac{M}{K_m+M} \quad (3.20a)$$

$$\frac{dP_0}{dt} = k_s M - v_1 \frac{P_0}{K_1+P_0} + v_2 \frac{P_1}{K_2+P_1} \quad (3.20b)$$

$$\frac{dP_1}{dt} = v_1 \frac{P_0}{K_1+P_0} - v_2 \frac{P_1}{K_2+P_1} - v_3 \frac{P_1}{K_3+P_1} + v_4 \frac{P_2}{K_4+P_2} \quad (3.20c)$$

$$\frac{dP_2}{dt} = v_3 \frac{P_1}{K_3+P_1} - v_4 \frac{P_2}{K_4+P_2} - k_1 P_2 + k_2 P_N - v_d \frac{P_2}{K_d+P_2} \quad (3.20d)$$

$$\frac{dP_N}{dt} = k_1 P_2 - k_2 P_N \quad (3.20e)$$

上式中的相关参数如下：$v_s=0.76$，$v_m=0.65$，$v_d=0.95$，$k_s=0.38$，$k_1=1.9$，$k_2=1.3$，$v_1=3.2$，$v_2=1.58$，$v_3=5.0$，$v_4=2.5$，$K_1=2.0$，$K_2=2.0$，$K_3=2.0$，$K_4=2.0$，$K_I=1.0$，$K_m=0.5$，$K_d=0.2$,n=4。Code-3-11 给出了以上过程的实现代码。其中，5-21 行设定基本的参数。24-31 行设定了时间步长，计算的最大步数，以及对各组分浓度进行初始化。34-54 行定义了 5 个子函数，对应式(3.20a-e)等号右边的 5 个函数表达式。66-97 行基于四阶龙格-库塔法求解微分方程组。107-129 对结果进行可视化，分别画出各组分蛋白浓度和总蛋白浓度随时间的变化，以及由 mRNA 和蛋白总浓度构成的相平面内的轨迹。

```python
# -*- coding: utf-8 -*-
import pylab as pl

# some parameters
vs = 0.76
vm = 0.65
vd = 0.95
ks = 0.38
k1 = 1.9
k2 = 1.3
v1 = 3.2
v2 = 1.58
v3 = 5.0
v4 = 2.5
K1 = 2.0
K2 = 2.0
K3 = 2.0
K4 = 2.0
Ki = 1.0
Km = 0.5
Kd = 0.2

# Initialization ...
t = 0.0
dt = 0.1
n = 10000
M = 1.9
P0 = 0.8
P1 = 0.8
P2 = 0.8
Pn = 0.8

# Defining functions....
def fM(M, Pn):
    freturn = vs * Ki**4/(Ki**4 + Pn**4) - vm * M/(Km + M)
    return freturn

def fP0(M, P0, P1):
    freturn = ks * M - v1 * P0/(K1 + P0) + v2 * P1/(K2 + P1)
    return freturn

def fP1(P0, P1, P2):
    freturn = v1 * P0/(K1 + P0) - v2 * P1/(K2 + P1) \
    - v3 * P1/(K3 + P1) + v4 * P2/(K4 + P2)
    return freturn

def fP2(P1, P2, Pn):
    freturn = v3 * P1/(K3+ P1) - v4 * P2/(K4 + P2) \
    - k1 * P2 + k2 * Pn -vd * P2/(Kd + P2)
    return freturn

def fPn(P2, Pn):
    freturn = k1 * P2 - k2 * Pn
    return freturn

#Defining empty list
MT = []
P0T = []
P1T = []
P2T = []
PnT = []
PtT = []
Time = []
```

```python
# 4th order R-K
for i in range(n):
    k1_m = fM(M, Pn)
    k1_0 = fP0(M, P0, P1)
    k1_1 = fP1(P0, P1, P2)
    k1_2 = fP2(P1, P2, Pn)
    k1_n = fPn(P2, Pn)

    k2_m = fM(  M+k1_m*dt/2, Pn+k1_n*dt/2)
    k2_0 = fP0( M+k1_m*dt/2, P0+k1_0*dt/2, P1+k1_1*dt/2)
    k2_1 = fP1(P0+k1_0*dt/2, P1+k1_1*dt/2, P2+k1_2*dt/2)
    k2_2 = fP2(P1+k1_1*dt/2, P2+k1_2*dt/2, Pn+k1_n*dt/2)
    k2_n = fPn(P2+k1_2*dt/2, Pn+k1_n*dt/2)

    k3_m = fM(  M+k2_m*dt/2, Pn+k2_n*dt/2)
    k3_0 = fP0( M+k2_m*dt/2, P0+k2_0*dt/2, P1+k2_1*dt/2)
    k3_1 = fP1(P0+k2_0*dt/2, P1+k2_1*dt/2, P2+k2_2*dt/2)
    k3_2 = fP2(P1+k2_1*dt/2, P2+k2_2*dt/2, Pn+k2_n*dt/2)
    k3_n = fPn(P2+k2_2*dt/2, Pn+k2_n*dt/2)

    k4_m = fM(  M+k3_m*dt, Pn+k3_n*dt)
    k4_0 = fP0( M+k3_m*dt, P0+k3_0*dt, P1+k3_1*dt)
    k4_1 = fP1(P0+k3_0*dt, P1+k3_1*dt, P2+k3_2*dt)
    k4_2 = fP2(P1+k3_1*dt, P2+k3_2*dt, Pn+k3_n*dt)
    k4_n = fPn(P2+k3_2*dt, Pn+k3_n*dt)

    M =M +dt/6.*(k1_m+2*k2_m+2*k3_m+k4_m)
    P0=P0+dt/6.*(k1_0+2*k2_0+2*k3_0+k4_0)
    P1=P1+dt/6.*(k1_1+2*k2_1+2*k3_1+k4_1)
    P2=P2+dt/6.*(k1_2+2*k2_2+2*k3_2+k4_2)
    Pn=Pn+dt/6.*(k1_n+2*k2_n+2*k3_n+k4_n)
    Pt = P0 + P1 + P2 + Pn
    t=t+dt

    MT.append(M)
    P0T.append(P0)
    P1T.append(P1)
    P2T.append(P2)
    PnT.append(Pn)
    PtT.append(Pt)
    Time.append(t)

fig = pl.figure(figsize=(10,4))
ax1 =fig.add_subplot(1,2,1)
ax2 =fig.add_subplot(1,2,2)

ax1.plot(Time, MT, 'r-', label='M',linewidth=1.0)
ax1.plot(Time, P0T, 'b-', label='M',linewidth=1.0)
ax1.plot(Time, P1T, 'c-', label='M',linewidth=1.0)
ax1.plot(Time, P2T, 'g-', label='M',linewidth=1.0)
ax1.plot(Time, PnT, 'y-', label='M',linewidth=1.0)
ax1.plot(Time, PtT, 'k-', label='M',linewidth=1.0)

ax2.plot(PtT, MT, 'r-', label='phase',linewidth=1.0)

ax1.set_ylabel(r'PER forms or M', fontsize=20)
ax1.set_xlabel(r'Time/h', fontsize=20)
ax1.set_xlim(200,300)
ax1.set_ylim(0,5)

ax2.set_ylabel(r'mRNA', fontsize=20)
ax2.set_xlabel(r'Total PER', fontsize=20)
ax2.set_xlim(0,6)
ax2.set_ylim(0,4)
pl.show()
```

Code-3-11

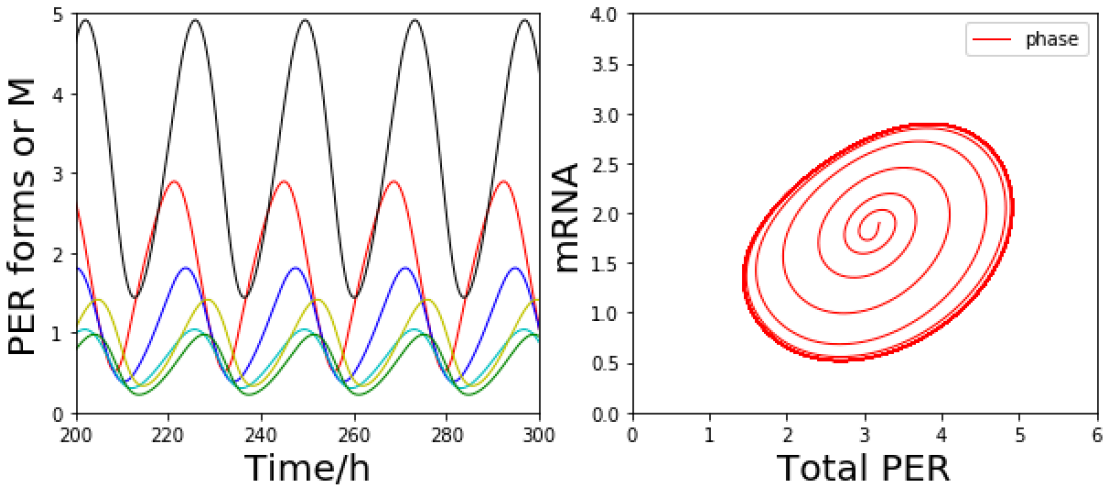

图 3.17 （左）各蛋白组分随时间的变化；（右）mRNA 和蛋白总浓度构成的相平面内的轨迹。

运行 code-3-11 得到的结果如图 3.17 所示（只显示了 200-300 小时范围内的结果）。可以看出，各组分蛋白浓度呈周期性震荡。另外，尽管各组分蛋白最高浓度出现的时间不同，但震荡周期相同，都为 24 小时。相应地，相平面的轨迹收敛到环形轨迹，展示出周期运动特征。以上讨论尽管对实际情况做了很大简化，但给出了 24 小时生物节日产生的一个基本的物理机制。除了事实上，细胞的各种生命过程都可以通过基因调控网络动力学和代谢网络动力学来理解，包括细胞凋亡、周期阻断、细胞分裂等命运抉择过程，而求解常微分方程是研究这类细胞动力学问题的基本工具。

参考文献：

[1] 郝柏林，从抛物线谈起--混沌动力学引论(第二版)，北京大学出版社，2013.

[2] Goldbeter A. A model for circadian oscillations in the Drosophila period protein (PER), Proc. R. Soc. Lond. B, 1995, 261:319-324.