第八章

微分方程模型

目录 CONTENTS

01 微分方程模型的求解方法

02 微分方程建模方法

03 微分方程建模实例

04 拉氏变换求常微分方程（组）的符号解

8.1.1 微分方程的数值解

考虑一阶常微分方程的初值问题

$$\begin{cases} \dfrac{dy}{dx} = f(x,y), \\ y(x_0) = y_0 \end{cases} \tag{8.1}$$

在区间$[a,b]$上的解，其中$f(x,y)$为x,y的连续函数，y_0是给定的初始值。将上述问题的精确解记为$y(x)$。

所谓数值解法，就是求初值问题（8.1）的解$y(x)$在若干个点

$$a = x_0 < x_1 < x_2 < \cdots < x_N = b$$

处的近似值$y_n(n = 1,2,\cdots,N)$的方法，$y_n(n = 1,2,\cdots,N)$称为问题（8.1）的数值解，$h_n = x_{n+1} - x_n$称为由x_n到x_{n+1}的步长，一般取等步长$h = (b-a)/N$。

建立数值解法，首先要将微分方程离散化，一般采用以下几种方法：

1.用差商近似导数

若用向前差商$\frac{y(x_{n+1})-y(x_n)}{h}$代替$y'(x_n)$代入（8.1）中的微分方程，则得

$$\frac{y(x_{n+1})-y(x_n)}{h}\approx f(x_n,y(x_n)),\quad n=0,1,\cdots,N-1.$$

化简得

$$y(x_{n+1})\approx y(x_n)+hf(x_n,y(x_n)).$$

用$y(x_n)$的近似值y_n代入上式右端，所得结果作为$y(x_{n+1})$的近似值，记为y_{n+1}则有

$$y_{n+1}=y_n+hf(x_n,y_n),\quad n=0,1,\cdots,N-1.$$

于是问题（8.1）的数值解可通过下列递推公式

$$\begin{cases} y_{n+1} = y_n + hf(x_n, y_n), n = 0,1,\cdots,N-1, \\ y_0 = y(a) \end{cases} \tag{8.2}$$

得到，按式（8.2）由初值y_0可逐次算出$y_1, y_2, \cdots, y_N$。这个求解方法称为欧拉（Euler）方法。式（8.2）是个离散化的问题，也称其为差分方程初值问题。

因为用用差商近似导数的误差为

$$\frac{y(x_{n+1}) - y(x_n)}{h} - y'(x_n) = \frac{1}{2}hy''(\xi_n),$$

此求解公式得到的数值解的截断误差是$O(h)$.

2.用数值积分方法

对问题（8.1）的方程两边从x_n到x_{n+1}积分，得

$$y(x_{n+1})-y(x_n)=\int_{x_n}^{x_{n+1}} f(x,y(x))dx,\quad n=0,1,\cdots,N-1,\quad (8.3)$$

用矩形公式或梯形公式计算右边积分的近似值，得到求数值解的公式。

1° 积分用矩形公式近似，得

$$y(x_{n+1})-y(x_n)\approx hf(x_n,y(x_n)).$$

用$y(x_n)$的近似值y_n代入上式的右边，并把所得的值作为$y(x_{n+1})$的近似值, 即

$$\begin{cases} y_{n+1} = y_n + hf(x_n, y_n), \ n = 0, 1, \cdots, N-1. \\ y_0 = y(a) \end{cases}$$

2° 积分用梯形公式近似，得

$$y(x_{n+1}) - y(x_n) \approx \frac{h}{2}[f(x_n, y(x_n)) + f(x_{n+1}, y(x_{n+1}))].$$

分别用$y(x_n), y(x_{n+1})$的近似值y_n, y_{n+1}代入上式, 得

$$y_{n+1} = y_n + \frac{h}{2}[f(x_n, y_n) + f(x_{n+1}, y_{n+1})], \ n = 0, 1, \cdots, N-1.$$

利用上述公式求解时，当已知y_n时还不能算出y_{n+1}, 因此称为隐式公式. 而前面得到的公式，当已知y_n时直接计算就可得出y_{n+1}，因此称为显式公式. 为了利用上述隐式公式，可做预测-校正处理得到新的求解公式

$$\begin{cases} y_{n+1}^{(p)} = y_n + hf(x_n, y_n), \\ y_{n+1} = y_n + \frac{h}{2}[f(x_n, y_n) + f(x_{n+1}, y_{n+1}^{(p)})], n = 0, 1, \cdots, N-1. \end{cases}$$

上述公式还可以写为

$$\begin{cases} y_{n+1} = y_n + \frac{h}{2}[k_1 + k_2], \\ k_1 = f(x_n, y_n), \\ k_2 = f(x_{n+1}, y_n + hk_1), \end{cases} \qquad n = 0, 1, \cdots, N-1.$$

上述求解公式的截断误差是$O(h^2)$. 类似还有高阶公式

$$\begin{cases} y_{n+1} = y_n + \frac{h}{6}[k_1 + 2k_2 + 2k_3 + k_4], \\ k_1 = f(x_n, y_n), \\ k_2 = f(x_n + \frac{h}{2}, y_n + \frac{h}{2}k_1), \\ k_3 = f(x_n + \frac{h}{2}, y_n + \frac{h}{2}k_2), \\ k_4 = f(x_n + h, y_n + hk_3), \end{cases}$$

此求解公式称为经典龙格-库塔公式, 其截断误差是$O(h^4)$.

3.Taylor 多项式近似

将函数$y(x)$在x_n处展开，取一次 Taylor 多项式近似，则得

$$y(x_{n+1}) \approx y(x_n) + hy'(x_n) = y(x_n) + hf(x_n, y(x_n)),$$

再将$y(x_n)$的近似值y_n代入上式右端，所得结果作为$y(x_{n+1})$的近似值y_{n+1}，得到离散化的计算公式

$$y_{n+1} = y_n + hf(x_n, y_n).$$

以上三种方法都是将微分方程离散化的常用方法，每一类方法又可导出不同形式的计算公式，其中的 Taylor 展开法，不仅可以得到求数值解的公式，而且容易估计截断误差。

4.一阶微分方程组与高阶微分方程的数值解方法

考虑一阶微分方程组的初值问题

$$\begin{cases} y_i' = f_i(x, y_1, y_2, \cdots, y_m), \ a \le x \le b, \\ y_i(a) = \eta_i, i = 1, 2, \cdots, m. \end{cases}$$

令

$$\boldsymbol{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix}, \ \boldsymbol{f} = \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_m \end{bmatrix}, \ \boldsymbol{\eta}_0 = \begin{bmatrix} \eta_1 \\ \eta_2 \\ \vdots \\ \eta_m \end{bmatrix},$$

则一阶微分方程组的初值问题可以表示为向量形式

$$\begin{cases} \boldsymbol{y}' = \boldsymbol{f}(x, \boldsymbol{y}), \ a \le x \le b, \\ \boldsymbol{y}(a) = \boldsymbol{\eta}. \end{cases}$$

类似方程式的数值解公式，可以建立上述方程组的数值求解公式. 例如一阶求解公式

$$\begin{cases} \boldsymbol{y}_{n+1} = \boldsymbol{y}_n + h\boldsymbol{f}(x_n, \boldsymbol{y}_n), \ n = 0, 1, \cdots, N-1. \\ \boldsymbol{y}_0 = \boldsymbol{\eta}, \end{cases}$$

以及四阶经典龙格-库塔公式

$$\begin{cases} \boldsymbol{y}_{n+1} = \boldsymbol{y}_n + \frac{h}{6}[\boldsymbol{k}_1 + 2\boldsymbol{k}_2 + 2\boldsymbol{k}_3 + \boldsymbol{k}_4], \\ \boldsymbol{k}_1 = \boldsymbol{f}(x_n, \boldsymbol{y}_n), \\ \boldsymbol{k}_2 = \boldsymbol{f}(x_n + \frac{h}{2}, \boldsymbol{y}_n + \frac{h}{2}\boldsymbol{k}_1), \\ \boldsymbol{k}_3 = \boldsymbol{f}(x_n + \frac{h}{2}, \boldsymbol{y}_n + \frac{h}{2}\boldsymbol{k}_2), \\ \boldsymbol{k}_4 = \boldsymbol{f}(x_n + h, \boldsymbol{y}_n + h\boldsymbol{k}_3), \end{cases}$$

对于高阶微分方程组的初值问题

$$\begin{cases} y^{(m)} = f(x, y, y', y'', \cdots, y^{(m-1)}), \ a \le x \le b, \\ y(a) = \eta_1, y'(a) = \eta_2, \cdots, y^{(m-1)}(a) = \eta_m. \end{cases}$$

引入新变量，把问题化为一阶方程组：

$$y_1 = y, y_2 = y', y_3 = y'', \cdots, y_m = y^{(m-1)},$$

则相应一阶方程组的初值问题为

$$\begin{cases} y_i' = f_i(x, y_1, y_2, \cdots, y_m) = y_{i+1}, i = 1, 2, \cdots, m-1, \\ y_m' = f(x, y, y', \cdots, y^{(m-1)}) = f_m(x, y_1, y_2, \cdots, y_m), \ a \le x \le b, \\ y_i(a) = \eta_i, i = 1, 2, \cdots, m. \end{cases}$$

8.1.2 用Python求解微分方程

1.符号解法

第 3 章中我们已经介绍过微分方程的符号解，下面再举几个例子。

例 8.1　求下述微分方程的特解

$$\begin{cases} \dfrac{d^2y}{dx^2}+4\dfrac{dy}{dx}+2y=0, \\ y(0)=0, \quad y'(0)=1. \end{cases}$$

```
#程序文件 Pex8_1.py
from sympy.abc import x
from sympy import diff, dsolve, simplify, Function
y=Function('y')
eq=diff(y(x),x,2)+2*diff(y(x),x)+2*y(x)    #定义方程
con={y(0): 0, diff(y(x),x).subs(x,0): 1}    #定义初值条件
y=dsolve(eq, ics=con)
print(simplify(y))
```

求得符号解为 $y(x)=e^{-x}\sin x$。

例 8.2　求下述微分方程的解：

$$\begin{cases} \dfrac{d^2y}{dx^2} + 2\dfrac{dy}{dx} + 2y = \sin x, \\ y(0) = 0, \quad y'(0) = 1. \end{cases}$$

```
#程序文件 Pex8_2.py
from sympy.abc import x    #引进符号变量 x
from sympy import Function, diff, dsolve, sin
y=Function('y')
eq=diff(y(x),x,2)+2*diff(y(x),x)+2*y(x)-sin(x)    #定义方程
con={y(0): 0, diff(y(x), x).subs(x,0): 1}    #定义初值条件
y=dsolve(eq, ics=con)
print(y)
```

求得的符号解为 $y(x)=\left(\frac{6\sin x}{5}+\frac{2\cos x}{5}\right)e^{-x}+\frac{\sin x}{5}-\frac{2\cos x}{5}$。

例 8.3　求下列微分方程组的解：

$$\begin{cases} \dfrac{dx_1}{dt} = 2x_1 - 3x_2 + 3x_3, & x_1(0) = 1, \\ \dfrac{dx_2}{dt} = 4x_1 - 5x_2 + 3x_3, & x_2(0) = 2, \\ \dfrac{dx_3}{dt} = 4x_1 - 4x_2 + 2x_3, & x_3(0) = 3. \end{cases}$$

```
#程序文件 Pex8_3_1.py
import sympy as sp
t=sp.symbols('t')
x1,x2,x3=sp.symbols('x1,x2,x3',cls=sp.Function)
eq=[x1(t).diff(t)-2*x1(t)+3*x2(t)-3*x3(t),
      x2(t).diff(t)-4*x1(t)+5*x2(t)-3*x3(t),
      x3(t).diff(t)-4*x1(t)+4*x2(t)-2*x3(t)]
con={x1(0):1, x2(0):2, x3(0):3}
s=sp.dsolve(eq, ics=con); print(s)
```

或者编写如下简洁的 Python 程序:

```
#程序文件 Pex8_3_2.py
import sympy as sp
t=sp.symbols('t')
x1,x2,x3=sp.symbols('x1:4',cls=sp.Function)
x=sp.Matrix([x1(t),x2(t),x3(t)])
A=sp.Matrix([[2,-3,3],[4,-5,3],[4,-4,2]])
eq=x.diff(t)-A*x
s=sp.dsolve(eq,ics={x1(0):1,x2(0):2,x3(0):3})
print(s)
```

求得的符号解为

$$\begin{cases} x_1(t) = 2e^{2t} - e^{-t}, \\ x_2(t) = 2e^{2t} - e^{-t} + e^{-2t}, \\ x_3(t) = 2e^{2t} + e^{-2t}. \end{cases}$$

2.数值解法

Python 对常微分方程的数值求解是基于一阶方程进行的，高阶微分方程必须化成一阶方程组，通常采用龙格—库塔方法。scipy.integrate 模块的 odeint 函数求常微分方程的数值解，其基本调用格式为：

```
sol=odeint(func, y0, t)
```

其中 func 是定义微分方程的函数或匿名函数，y0 是初始条件的序列，t 是一个自变量取值的序列（t 的第一个元素一定为初始时刻），返回值 sol 是对应于序列 t 中元素的数值解，如果微分方程组中有n个函数，返回值 sol 是n列的矩阵，第$i(i=1,2,\cdots,n)$列对应于第i个函数的数值解。

例 8.4　求微分方程

$$\begin{cases} y' = -2y + x^2 + 2x, \\ y(1) = 2. \end{cases}$$

在$1 \le x \le 10$上，并且步长为 0.5 的数值解。

```
#程序文件 Pex8_4.py
from scipy.integrate import odeint
from numpy import arange
dy=lambda y, x: -2*y+x**2+2*x
x=arange(1, 10.5, 0.5)
sol=odeint(dy, 2, x)
print("x={}\n 对应的数值解 y={}".format(x, sol.T))
```

数值解 y=[2, 2.08484933, 2.9191691, 4.18723381, 5.77289452, 7.63342241, 9.75309843, 12.12613985, 14.75041934, 17.62515427, 20.75005673, 24.12502089, 27.7500077, 31.62500278, 35.75000104, 40.1250004, 44.75000015, 49.62500006, 54.75000002]

例 8.5 求例 8.1 的数值解，其中步长为 0.1，并在同一个图形界面上画出符号解和数值解的曲线。

解 引进$y_1 = y$，$y_2 = y'$，则可以把原来的二阶微分方程化为如下一阶微分方程组：

$$\begin{cases} y_1' = y_2, & y_1(0) = 0, \\ y_2' = -2y_1 - 2y_2, & y_2(0) = 1. \end{cases}$$

求数值解和画图的程序如下：

```
#程序文件 Pex8_5.py
from scipy.integrate import odeint
from sympy.abc import t
import numpy as np
import matplotlib.pyplot as plt
def Pfun(y,x):
    y1, y2=y;
    return np.array([y2, -2*y1-2*y2])
x=np.arange(0, 10, 0.1)    #创建时间点
```

```
sol1=odeint(Pfun, [0.0, 1.0], x)    #求数值解
plt.rc('font',size=16); plt.rc('font',family='SimHei')
plt.plot(x, sol1[:,0],'r*',label="数值解")
plt.plot(x, np.exp(-x)*np.sin(x), 'g', label="符号解曲线")
plt.legend(); plt.savefig("figure8_5.png"); plt.show()
```

所画出的数值解和符号解的图形如图 8.1 所示。

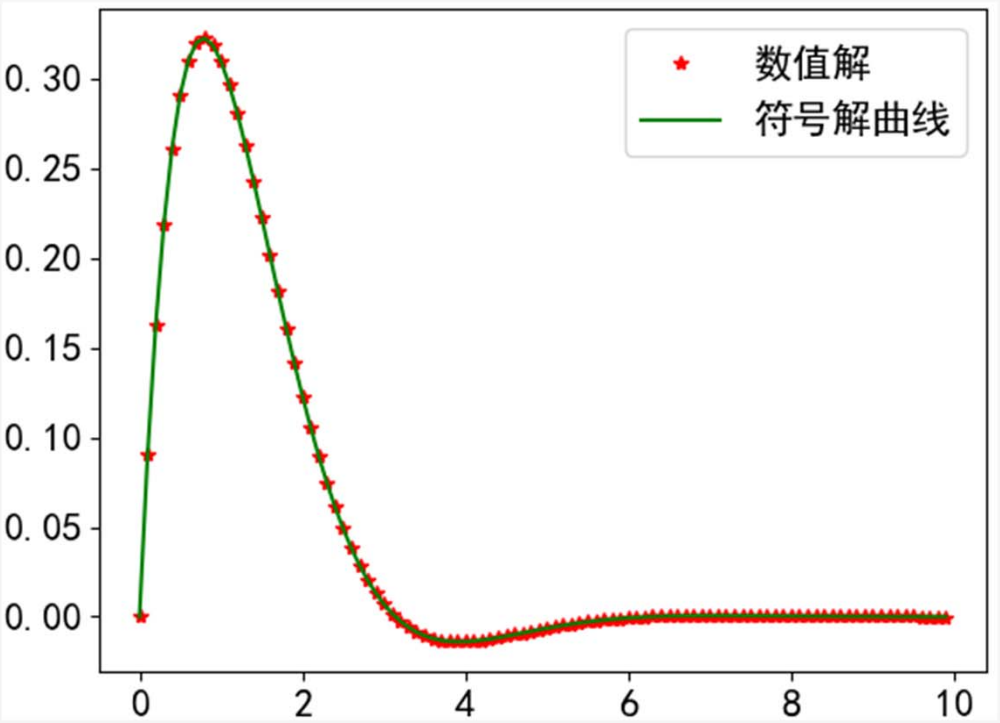


图 8.1　数值解和符号解的对比图

例 8.6　Lorenz 模型的混沌效应。

Lorenz 模型是由美国气象学家 Lorenz 在研究大气运动时，通过简化对流模型，只保留 3 个变量提出的一个完全确定性的一阶自治常微分方程组（不显含时间变量），其方程为

$$\begin{cases} \dot{x} = \sigma(y - x), \\ \dot{y} = \rho x - y - xz, \\ \dot{z} = xy - \beta z. \end{cases}$$

其中参数σ为 Prandtl 数，ρ为 Rayleigh 数，β为方向比。

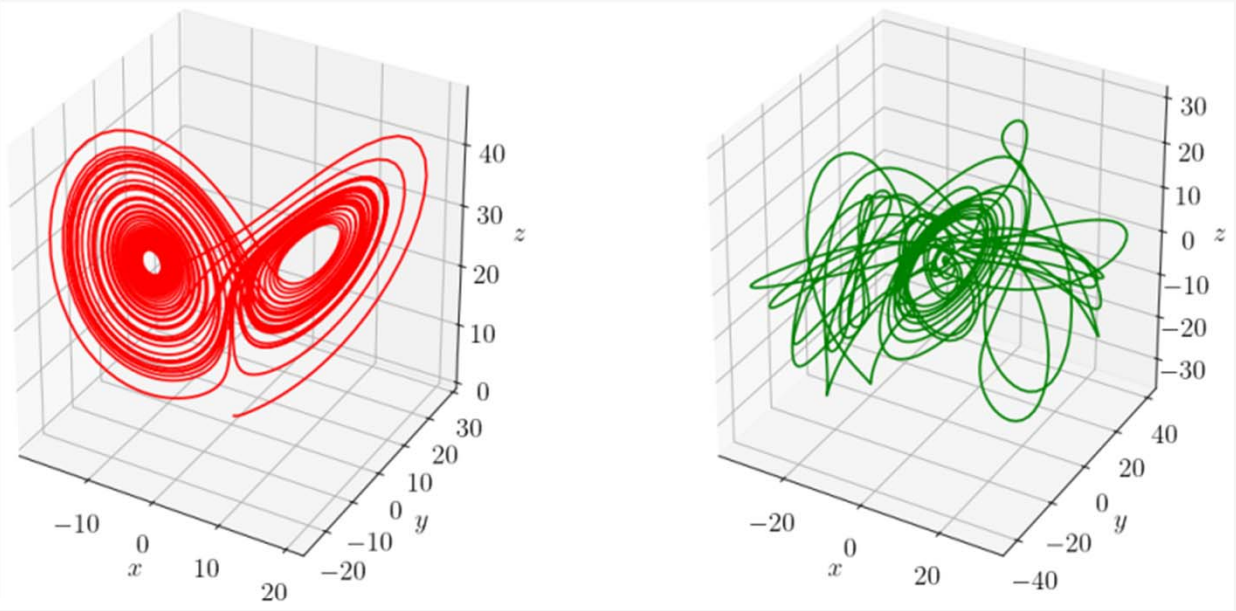
Lorenz 模型如今已经成为混沌领域的经典模型，第一个混沌吸引子—Lorenz 吸引子也是在这个系统中被发现的。系统中三个参数的选择对系统会不会进入混沌状态起着重要的作用。图 8.2（a）给出了 Lorenz 模型在 $\sigma=10$，$\rho=28$，$\beta=8/3$时系统的三维演化轨迹。由图 8.2（a）可见，经过长时间运行后，系统只在三维空间的一个有限区域内运动，即在三维相空间里的测度为零。图 8.2（a）显示出我们经常听到的“蝴蝶效应”。图 8.2（b）给出了系统从两个靠的很近的初值出发（相差仅 0.0001）后，解的偏差演化曲线。

随着时间的增大，可以看到两个解的差异越来越大，这正是动力学系统对初值敏感性的直观表现，由此可断定此系统的这种状态为混沌态。混沌运动是确定性系统中存在随机性，它的运动轨道对初始条件极端敏感。

```
#程序文件 Pex8_6.py
from scipy.integrate import odeint
import numpy as np
from mpl_toolkits import mplot3d
import matplotlib.pyplot as plt
def lorenz(w,t):
    sigma=10; rho=28; beta=8/3
    x, y, z=w;
    return np.array([sigma*(y-x), rho*x-y-x*z, x*y-beta*z])
t=np.arange(0, 50, 0.01)    #创建时间点
sol1=odeint(lorenz, [0.0, 1.0, 0.0], t)    #第一个初值问题求解
sol2=odeint(lorenz, [0.0, 1.0001, 0.0], t)    #第二个初值问题求解
```

```
plt.rc('font',size=16); plt.rc('text',usetex=True)
ax1=plt.subplot(121,projection='3d')
ax1.plot(sol1[:,0], sol1[:,1], sol1[:,2],'r')
ax1.set_xlabel('$x$'); ax1.set_ylabel('$y$'); ax1.set_zlabel('$z$')
ax2=plt.subplot(122,projection='3d')
ax2.plot(sol1[:,0]-sol2[:,0], sol1[:,1]-sol2[:,1], sol1[:,2]-sol2[:,2],'g')
ax2.set_xlabel('$x$'); ax2.set_ylabel('$y$'); ax2.set_zlabel('$z$')
plt.savefig("figure8_6.png", dpi=500); plt.show()
print("sol1=",sol1, '\n\n', "sol1-sol2=", sol1-sol2)
```

所画出的图形如图 8.2 所示。



（A）Lorenz 相轨线　　（B）两个解的偏差曲线

图 8.2 混沌效应图

目录 CONTENTS

01 微分方程模型的求解方法

02 微分方程建模方法

03 微分方程建模实例

04 拉氏变换求常微分方程（组）的符号解

建立微分方程模型一般可分以下三步：

（1）根据实际要求确定研究的量（自变量、未知函数、必要的参数等），并确定坐标系。

（2）找出这些量所满足的基本规律。

（3）运用这些规律列出方程和定解条件。

下面通过实例介绍几类常用的利用微分方程建立数学模型的方法。

1.按物理规律建立方程

例 8.7　建立物体冷却过程的数学模型。

将某物体放置于空气中，在时刻$t=0$时，测量得它的温度为$u_0=150℃$，10 分钟后测量得它的温度为$u_1=100℃$. 要求建立此物体的温度u和时间t的关系，并计算 20 分钟后物体的温度。假设空气的温度保持为$\tilde{u}=24℃$。

解　Newton 冷却定律是温度高于周围介质的物体向周围介质传递热量逐渐冷却时所遵循的规律：当物体表面与周围存在温度差时，单位时间从单位面积散失的热量与温度差成正比，比例系数称为热传递系数。

假设该物体在时刻t时的温度为$u = u(t)$，则由 Newton 冷却定律，得到

$$\frac{du}{dt} = -k(u - \tilde{u}), \tag{8.4}$$

其中热传递系数$k > 0$，方程（8.4）就是物体冷却过程中温度满足的数学模型。根据问题的叙述，初始条件为 u(0)=150。

注意到$\tilde{u}=24$为常数，$u-\tilde{u}>0$，可将方程（8.4）改写为

$$\frac{d(u-24)}{u-24}=-kdt.$$

两边积分得到

$$\int_{150}^{u}\frac{d(u-24)}{u-24}=\int_{0}^{t}-kdt,$$

化简得

$$u=24+126e^{-kt}. \tag{8.5}$$

把条件$t=10$时，$u=u_1=100$代入式（8.5），得$k=\frac{1}{10}\ln\frac{126}{76}=0.0506$，所以此物体的温度$u$和时间$t$的关系为$u=24+126e^{-0.0506t}$。20 分钟后物体的温度为 69.8413℃。

计算的 Python 程序如下

```
#程序文件 Pex8_7.py
import sympy as sp
sp.var('t, k')    #定义符号变量 t,k
u = sp.var('u', cls=sp.Function)    #定义符号函数
eq = sp.diff(u(t), t) + k * (u(t) - 24)    #定义方程
uu = sp.dsolve(eq, ics={u(0): 150}) #求微分方程的符号解
print(uu)
```

```
kk = sp.solve(uu, k)    #kk 返回值是列表，可能有多个解
k0 = kk[0].subs({t: 10.0, u(t): 100.0})
print(kk, '\t', k0)
u1 = uu.args[1]    #提出符号表达式
u0 = u1.subs({t: 20, k: k0})    #代入具体值
print("20 分钟后的温度为：", u0)
```

2.微元分析法

该方法的基本思想是通过分析研究对象的有关变量在一个很短时间内的变化规律，寻找一些微元之间的关系式。

例 8.8　有高为 1m 的半球形容器，开始时容器内盛满了水，水从它的底部小孔流出，小孔横截面积为 $1cm^2$。求水从小孔流出过程中容器里水面的高度h（水面与孔口中心的距离）随时间t变化的规律。

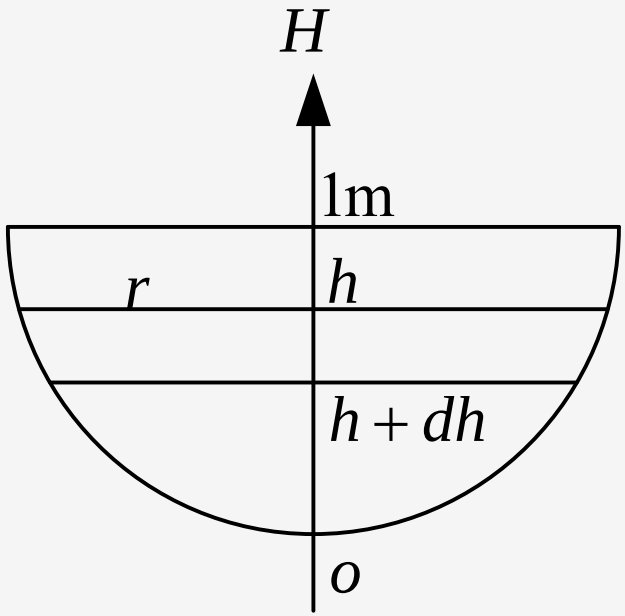


图 8.3　半球形容器及坐标系

解　如图 8.3 所示，以底部中心为坐标原点，垂直向上为坐标轴的正向建立坐标系。

由水力学知，水从孔口流出的流量Q为“通过孔口横截面的水的体积V对时间t的变化率”，满足

$$Q=\frac{dV}{dt}=0.62S\sqrt{2gh}\,, \tag{8.6}$$

式中 0.62 为流量系数，g为重力加速度（取 9.8m/s^2），S为孔口横截面积（单位：m^2），h为t时刻水面高度（单位：cm）。

当$S=1\text{cm}^2=0.0001\text{m}^2$时，

$$dV=0.000062\sqrt{2gh}dt. \tag{8.7}$$

在微小时间间隔$[t,t+dt]$内，水面高度由h降到$h+dh$（这里$dh<0$），容器中水的体积改变量近似为

$$dV=-\pi r^2dh, \tag{8.8}$$

式中，r为t时刻的水面半径，右端置负号是由于$dh<0$，而$dV>0$；这里

$$r^2=[1^2-(1-h)^2]=2h-h^2. \tag{8.9}$$

由式(8.7)-(8.9)，得

$$0.000062\sqrt{2gh}dt = \pi(h^2 - 2h)dh.$$

考虑到初始条件当 t=0 时 h=1，得到如下的微分方程模型

$$\begin{cases} \dfrac{dt}{dh} = \dfrac{10000\pi}{0.62\sqrt{2g}}(h^{\frac{3}{2}} - 2h^{\frac{1}{2}}), \\ t(1) = 0, \end{cases}$$

利用分离变量法，可以求得微分方程的解为

$$t(h) = -15260.5042h^{\frac{3}{2}} + 4578.1513h^{\frac{5}{2}} + 10682.3530.$$

上式表达了水从小孔流出的过程中容器内水面高度h与时间t之间的关系。

计算的 Python 程序如下：

```
#程序文件 Pex8_8.py
import sympy as sp
sp.var('h')    #定义符号变量
sp.var('t', cls=sp.Function)    #定义符号函数
g = 9.8
```

```
    eq = t(h).diff(h) -10000*sp.pi/0.62/sp.sqrt(2*g)*(h**(3/2)-2*h**(1/2))    #定义方程
    t = sp.dsolve(eq, ics={t(1): 0}) #求微分方程的符号解
    t = sp.simplify(t)
    print(t.args[1].n(9))
```

运行结果如下:

```
-15260.5042*h**1.5 + 4578.15127*h**2.5 + 10682.353
```

3.模拟近似法

该方法的基本思想是对实际现象做一些近似假设模拟，建立近似的微分方程，从数学上求解方程或分析解的性质，再去和实际情况作对比，观察这个模型能否模拟、近似某些实际的现象。

例 8.9（交通管理问题） 在交通十字路口，都会设置红绿灯。为了让那些正行驶在交叉路口或离交叉路口太近而无法停下的车辆通过路口，红绿灯转换中间还要亮起一段时间的黄灯。那么，黄灯应亮多长时间才最为合理呢?

分析：黄灯状态持续的时间包括驾驶员的反应时间、车通过交叉路口的时间以及通过刹车距离所需的时间。

解　记v_0是法定速度，I是交通路口的长度，L是典型的车身长度，则车通过路口的时间为$\dfrac{I+L}{v_0}$。

下面计算刹车距离，刹车距离就是从开始刹车到速度$v=0$时汽车驶过的距离。设W为汽车的重量，μ为摩擦系数，那么，地面对汽车的摩擦力为μW，其方向与运动方向相反。由牛顿第二定律，汽车在停车过程中，行驶的距离x与时间t的关系可由下面的微分方程表示：

$$\frac{W}{g}\frac{d^2x}{dt^2}=-\mu W\ , \tag{8.10}$$

其中，g为重力加速度。化简(8.10)式，得

$$\frac{d^2x}{dt^2}=-\mu g. \tag{8.11}$$

再考虑初始条件，建立如下的二阶微分方程模型

$$\begin{cases} \dfrac{d^2x}{dt^2} = -\mu g, \\ x\big|_{t=0} = 0, \quad \left.\dfrac{dx}{dt}\right|_{t=0} = v_0. \end{cases} \tag{8.12}$$

先求解二阶微分方程(8.11)式，对(8.11)式两边从 0 到t积分，利用初始条件得

$$\frac{dx}{dt} = -\mu g t + v_0. \tag{8.13}$$

再积分(8.13)式一次，求得二阶微分方程初值问题(8.12)式的解为

$$x(t)=-\frac{1}{2}\mu g t^2+v_0 t. \tag{8.14}$$

在（8.13）式中令$\frac{dx}{dt}=0$，可得刹车所用时间$t_0=\frac{v_0}{\mu g}$，从而得到刹车距离$x(t_0)=\frac{v_0^2}{2\mu g}$。

设T_0是驾驶员的反应时间，则黄灯状态的时间

$$T=\frac{x(t_0)+I+L}{v_0}+T_0, \tag{8.15}$$

把代入$x(t_0)$上式，得

$$T=\frac{v_0}{2\mu g}+\frac{I+L}{v_0}+T_0. \tag{8.16}$$

设$T_0=1\text{s}$，$L=4.5\text{m}$，$I=9\text{m}$。另外，取具有代表性的$\mu=0.7$，当$v_0=45$ km/h、60 km/h 以及 80 km/h 时，黄灯时间T如表 8.1 所列。

表 8.1　不同速度下计算的黄灯时长

v_0/(m/s)	12.5	16.667	22.222
T /s	2.991	3.025	3.227

计算黄灯时间的 Python 程序如下：

```
#程序文件 Pex8_9.py
from numpy import array
v0=array([12.5, 16.667, 22.222])
T0=1; L=4.5; I=9; mu=0.7; g=9.8
T=v0/(2*mu*g)+(I+L)/v0+T0
print(T)
```

目录 CONTENTS

01 微分方程模型的求解方法

02 微分方程建模方法

03 微分方程建模实例

04 拉氏变换求常微分方程（组）的符号解

8.3.1 Malthus模型

1789 年，英国神父 Malthus 在分析了一百多年人口统计资料之后，提出了 Malthus 模型。

1.模型假设

（1）设$x(t)$表示t时刻的人口数，且$x(t)$连续可微。

（2）人口的增长率r是常数（增长率=出生率-死亡率）。

（3）人口数量的变化是封闭的，即人口数量的增加与减少只取决于人口中个体的生育和死亡，且每一个体都具有同样的生育能力与死亡率。

2.建模与求解

由假设(2)，t时刻到$t+\Delta t$时刻人口的增量为$x(t+\Delta t)-x(t)=rx(t)\Delta t$，于是得

$$\begin{cases}\dfrac{dx}{dt}=rx,\\ x(0)=x_0,\end{cases} \tag{8.17}$$

其解为

$$x(t)=x_0e^{rt}. \tag{8.17}$$

3.模型评价

考虑二百多年来人口增长的实际情况，1961 年世界人口总数为 3.06×10^9，在 1961～1970 年这段时间内，每年平均的人口自然增长率为 2%，则（8.18）式可写为

$$x(t)=3.06\times10^9\cdot e^{0.02(t-1961)}. \tag{8.19}$$

根据 1700～1961 年间世界人口统计数据，发现这些数据与(8.19)式的计算结果相当符合。因为在这期间地球上人口大约每 35 年增加 1 倍，而(8.19)式算出每 34.6 年增加 1 倍。

但是，利用(8.19)式对世界人口进行预测，也会得出惊异的结论，当 $t=2670$年时，$x(t)=4.4\times10^{15}$，即 4400 万亿，这相当于地球上每平方米要容纳至少 20 人。

显然，用这一模型进行预测的结果远高于实际人口增长，误差的原因是对增长率r的估计过高。由此，可以对r是常数的假设提出疑问。

8.3.2 Logistic模型

地球上的资源是有限的，它只能提供一定数量的生命生存所需的。当人口数量的增加时，自然资源、环境条件等对人口再增长的限制作用也加强，人口的增长会减慢。当人口较少时，自然资源、环境条件等对人口再增长的限制作用会减弱，人口会快速增长。可见，增长率r不是常数，而是随着人口的增加而减小的量，即增长率r是人口数$x(t)$的函数$r(x)$，且是x的减函数。当时间 t 比较短时，因为人口数量变化不大，可将增长率r近似为常数。

1.模型假设

(1) 设$r(x)$为x的线性函数，$r(x)=r-sx$（工程师原则，首先用线性）。

(2) 自然资源与环境条件所能容纳的最大人口数为x_m，即当$x=x_m$时，增长率$r(x_m)=0$，因此 s=r/x_m。

2.建模与求解

由假设（1），（2）可得$r(x)=r(1-\frac{x}{x_m})$，则有

$$\begin{cases}\dfrac{dx}{dt}=r(1-\dfrac{x}{x_m})x,\\ x(t_0)=x_0.\end{cases} \tag{8.20}$$

(8.20)式是一个可分离变量的方程，其解为

$$x(t)=\frac{x_m}{1+(\frac{x_m}{x_0}-1)e^{-r(t-t_0)}} \tag{8.21}$$

3.模型检验

由(8.20)式，计算可得

$$\frac{d^2x}{dt^2}=r^2\left(1-\frac{x}{x_m}\right)\left(1-\frac{2x}{x_m}\right)x. \tag{8.22}$$

人口总数$x(t)$有如下规律：

（1） 只要人口初值$x_0>0$，人口数 x(t)>0，并且当 $0<x_0<x_m$ 时总有 $x(t)<x_m$，当 $x_0>x_m$ 时，总有 $x(t)>x_m$，当 $x_0=x_m$ 时，有 $x(t)=x_m$，同时极限 $\lim\limits_{t\to+\infty} x(t)=x_m$，即无论人口初值$x_0$如何，人口总数以$x_m$为极限。

(2) 当$0 < x_0 < x_m$时，$\frac{dx}{dt} = r(1 - \frac{x}{x_m})x > 0$，这说明$x(t)$是单调增加的，又由(8.22)式知，当$x < \frac{x_m}{2}$时，$\frac{d^2x}{dt^2} > 0$，$x = x(t)$为凹函数，当$x > \frac{x_m}{2}$时，$\frac{d^2x}{dt^2} < 0$，$x = x(t)$为凸函数。

（3）人口变化率$\frac{dx}{dt}$在$x=\frac{x_m}{2}$时取到最大值，即人口总数达到极限值一半以前是加速生长时期，经过这一点之后，增长速率会逐渐变小，最终达到零。

8.3.3 美国人口的预报模型

例 8.10　利用表 8.2 给出的近两个世纪的美国人口统计数据（以百万为单位），建立人口预测模型，最后用它预报 2010 年美国的人口。

表 8.2　美国人口统计数据

年	1790	1800	1810	1820	1830	1840	1850	1860
人口	3.9	5.3	7.2	9.6	12.9	17.1	23.2	31.4
年	1870	1880	1890	1900	1910	1920	1930	1940
人口	38.6	50.2	62.9	76.0	92.0	106.5	123.2	131.7
年	1950	1960	1970	1980	1990	2000		
人口	150.7	179.3	204.0	226.5	251.4	281.4		

1. 建模与求解

记$x(t)$为第t年的人口数量，设人口年增长率$r(x)$为x的线性函数，$r(x)=r-sx$。自然资源与环境条件所能容纳的最大人口数为x_m，即当$x=x_m$时，增长率$r(x_m)=0$，可得$r(x)=r(1-\frac{x}{x_m})$，建立 Logistic 人口模型

$$\begin{cases}\dfrac{dx}{dt}=r(1-\dfrac{x}{x_m})x,\\ x(t_0)=x_0,\end{cases}$$

其解为

$$x(t)=\frac{x_m}{1+(\frac{x_m}{x_0}-1)e^{-r(t-t_0)}}. \tag{8.23}$$

2. 参数估计

把表 8.2 中的全部数据保存到文本文件 Pdata8_10_1.txt 中。

（1）非线性最小二乘估计

把表8.2中的第1个数据作为初始条件，利用余下的数据拟合式(8.23)中的参数x_m和r，编写的 Python 程序如下

```
#程序文件 Pex8_10_1.py
import numpy as np
from scipy.optimize import curve_fit
a=[]; b=[];
with open("Pdata8_10_1.txt") as f:        #打开文件并绑定对象 f
    s=f.read().splitlines()    #返回每一行的数据
for i in range(0, len(s),2):    #读入奇数行数据
    d1=s[i].split("\t")
```

```
        for j in range(len(d1)):
                if d1[j]!="": a.append(eval(d1[j]))    #把非空的字符串转换为
年代数据
 for i in range(1, len(s), 2):    #读入偶数行数据
        d2=s[i].split("\t")
        for j in range(len(d2)):
                if d2[j] != "": b.append(eval(d2[j])) #把非空的字符串转换为
人口数据
 c=np.vstack((a,b))    #构造两行的数组
```

```
np.savetxt("Pdata8_10_2.txt", c)   #把数据保存起来供下面使用
x=lambda t, r, xm: xm/(1+(xm/3.9-1)*np.exp(-r*(t-1790)))
bd=((0, 200), (0.1,1000))   #约束两个参数的下界和上界
popt, pcov=curve_fit(x, a[1:], b[1:], bounds=bd)
print(popt); print("2010 年的预测值为： ", x(2010, *popt))
```

注 8.1　用 with 语句打开数据文件并把它绑定到对象 f。不必操心在操作完资源后去关闭数据文件，因为 with 语句的上下文管理器会帮助处理。这在操作资源型文件时非常方便，因为它能确保在代码执行完毕后资源会被释放掉（比如关闭文件）。

求得$r = 0.0274$，$x_m = 342.4419$,，2010 年人口的预测值为 282.6798 百万。

（2）线性最小二乘法

为了利用简单的线性最小二乘法估计这个模型的参数r和x_m，把Logistic 方程表示为

$$\frac{1}{x}\cdot\frac{dx}{dt}=r-sx,\quad s=\frac{r}{x_m}. \tag{8.24}$$

记$1790,1800,\cdots,2000$年分别用$k=1,2,\cdots,22$表示，利用向前差分，得到差分方程

$$\frac{1}{x(k)}\frac{x(k+1)-x(k)}{\Delta t}=r-sx(k),\quad k=1,2,\cdots,21. \tag{8.25}$$

其中步长$\Delta t=10$，下面先拟合其中的参数r和s。编写 Python 程序如下：

```
#程序文件 Pex8_10_2.py
import numpy as np
d=np.loadtxt("Pdata8_10_2.txt")   #加载文件中的数据
t0=d[0]; x0=d[1]   #提取年代数据及对应的人口数据
b=np.diff(x0)/10/x0[:-1]   #构造线性方程组的常数项列
a=np.vstack([np.ones(len(x0)-1),-x0[:-1]]).T #构造线性方程组系数矩阵
rs=np.linalg.pinv(a)@b;    r=rs[0]; xm=r/rs[1]
```

print("人口增长率 r 和人口最大值 xm 的拟合值分别为", np.round([r,xm],4))

xhat=xm/(1+(xm/3.9-1)*np.exp(-r*(2010-1790))) #求预测值

print("2010 年的预测值为： ",round(xhat,4))

求得$r=0.0325$，$x_m=294.3860$。2010 年人口的预测值为277.9634百万。

从上面的两种拟合方法可以看出，拟合同样的参数，方法不同可能结果相差很大。

8.3.4 传染病模型

流行病动力学是用数学模型研究某种传染病在某一地区是否蔓延下去成为当地的“地方病”，或最终该病将消除。下面以 Kermack 和 Mckendrick 提出的阈值模型为例说明流行病学数学模型的建模过程。

1.模型假设

（1）被研究人群是封闭的，总人数为n。$s(t),i(t)$和$r(t)$分别表示t时刻人群中易感染者、感染者（病人）和免疫者的人数。起始条件为s_0个易感者，i_0个感染者，免疫者$n-s_0-i_0$个。

（2）易感人数的变化率与当时的易感人数和感染人数之积成正比，比例系数为λ。

（3）免疫者人数的变化率与当时的感染者人数成正比，比例系数为μ。

（4）三类人总的变化率代数和为零。

2.模型建立

根据上述假设，可以建立如下模型：

$$\begin{cases} \dfrac{ds}{dt} = -\lambda si, \\ \dfrac{di}{dt} = \lambda si - \mu i, \\ \dfrac{dr}{dt} = \mu i, \\ s(t) + i(t) + r(t) = n. \end{cases} \tag{8.26}$$

以上模型又称 Kermack-Mckendrick 方程。

3.模型求解与分析

对于方程（8.26）无法求出$s(t)$、$i(t)$和$r(t)$的解析解，转到平面$s-i$上来讨论解的性质。由方程（8.26）中的前两个方程消去dt，可得

$$\begin{cases} \dfrac{di}{ds}=\dfrac{1}{\sigma s}-1, \\ i\big|_{s=s_0}=i_0, \end{cases} \tag{8.27}$$

其中$\sigma=\lambda/\mu$，是一个传染期内每个患者有效接触的平均人数，称为接触数。用分离变量法可求出（8.27）式的解为

$$i=(s_0+i_0)-s+\frac{1}{\sigma}\ln\frac{s}{s_0}. \tag{8.28}$$

s与i的关系如图 8.4 所示，从图中可以看出，当初始值$s_0 \le 1/\sigma$时，传染病不会蔓延。患者人数一直在减少并逐渐消失。而当$s_0 > 1/\sigma$时，患者人数会增加，传染病开始蔓延，健康者的人数在减少。当$s(t)$减少至$1/\sigma$时，患者在人群中的比例达到最大值，然后患者数逐渐减少至零。由此可知，$1/\sigma$是一个阈值，要想控制传染病的流行，应控制s_0使之小于此阈值。

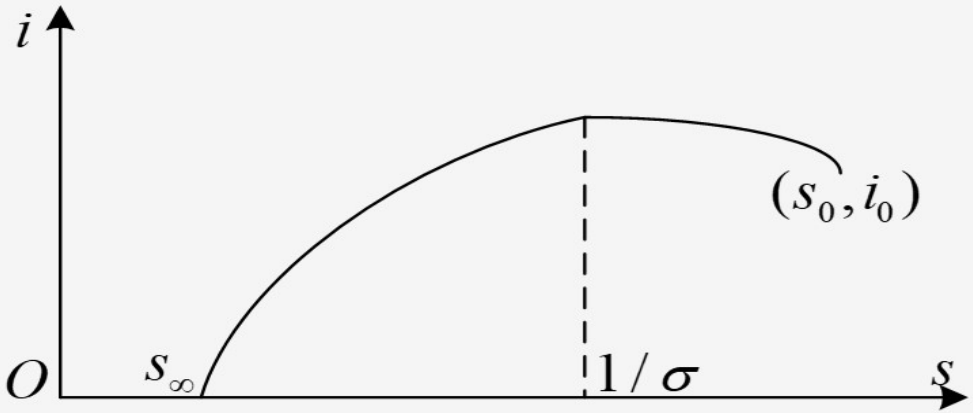


图 8.4　s与i关系

由上述分析可知，要控制疫后有免疫力的此类传染病的流行可通过两个途径：一是提高卫生和医疗水平，卫生水平越高，传染性接触率λ就越小；医疗水平越高，恢复系数μ就越大。这样，阈值$1/\sigma$就越大，因此提高卫生和医疗水平有助于控制传染病的蔓延。另一条途径是通过降低s_0来控制传染病的蔓延。由$s_0+i_0+r_0=n$可知，要想减少s_0可通过提高r_0来实现，而这又可通过预防接种和群体免疫等措施来实现。

4.参数估计

参数σ的值可由实际数据估计得到，记s_∞、i_∞分别是传染病流行结束后的健康者人数和患者人数。当流行结束后，患者都将转为免疫者。所以，$i_\infty = 0$。则由（8.28）式可得

$$i_\infty = 0 = s_0 + i_0 - s_\infty + \frac{1}{\sigma}\ln\frac{s_\infty}{s_0},$$

解出σ得

$$\sigma = \frac{\ln s_0 - \ln s_\infty}{s_0 + i_0 - s_\infty}. \tag{8.29}$$

于是，当已知某地区某种疾病流行结束后的s_∞，那么可由（8.29）计算出σ的值，而此σ的值可在今后同种传染病和同类地区的研究中使用。

5.模型应用

这里以 1950 年上海市某全托幼儿所发生的一起水痘流行过程为例，应用 K-M 模型进行模拟，并对模拟结果进行讨论。该所儿童总人数n为 196 人；既往患过水痘而此次未感染者 40 人；查不出水痘患病史而本次流行期间感染水痘者 96 人，既往无明确水痘史，本次又未感染的幸免者 60 人。全部流行期间 79 天，病例成代出现，每代间隔约 15 天。各代病例数、易感染者数及间隔时间如表 8.3 所示。

表 8.3　某托儿所水痘流行过程中各代病例数

代	病例数	易感染者	间隔时间/天
1	1	155	
2	2	153	15
3	14	139	32
4	38	101	46
5	34	67	
6	7	60	
合计	96		

以初始值$s_0 = 155, s_\infty = 60$代入(8.29)式可得$\sigma = 0.0099$。将σ代入(8.28)可得该流行过程的模拟结果所表 8.4 所列。

表 8.4　用 K_M 模型模拟水痘的数值解

易感者数S	155	153	139	101
病例数i	1	1.7	6.0	11.7

从计算结果来看，模型模拟的效果与实际数据差别较大，说明所建模型不够理想，需进一步完善。

计算的 Python 程序如下：

```
#程序文件 Pex8_11.py
import numpy as np
s0=155.0;    i0=1.0;    s_inf=60.0;
sigma=(np.log(s0)-np.log(s_inf))/(s0+i0-s_inf)
print("sigma=",sigma)
S=np.array([155, 153, 139, 101])
I=(s0+i0)-S+1/sigma*np.log(S/s0)
print("所求的解为：\n",I)
```

目录 CONTENTS

01 微分方程模型的求解方法

02 微分方程建模方法

03 微分方程建模实例

04 拉氏变换求常微分方程（组）的符号解

拉普拉斯变换定义如下：设函数$f(t)$定义在$[0,\infty)$, 则广义积分

$$L[f](s)=\int_0^{\infty} f(t)e^{-st}\mathrm{d}t$$

所定义复函数$L[f](s)$称为函数$f(t)$的拉普拉斯变换或拉普拉斯像函数, 同时称$f(t)$为$L[f](s)$的拉普拉斯逆变换或拉普拉斯像原函数.

拉普拉斯变换有很多性质，其中微分性质在求解常微分方程(组)时积极重要，该性质如下：

$$L[f^{(k)}(t)](s)=s^kL[f](s)-s^{k-1}f(0)-s^{k-2}f'(0)-\cdots-f^{(k-1)}(0).$$

例 8.11　利用拉氏变换解初值问题

$$y^{(4)} + 2y'' + y = 4te^{t};$$

$$y(0) = y'(0) = y''(0) = y^{(3)}(0) = 0.$$

解　设$L(y(t))=Y(s)$，利用初值条件，对微分方程两边取拉氏变换变换得

$$s^4Y(s)+2s^2Y(s)+Y(s)=\frac{4}{(s-1)^2},$$

解之得

$$Y(s)=\frac{4}{(s-1)^2(s^2+1)^2}=\frac{1}{(s-1)^2}-\frac{2}{s-1}+\frac{2s}{(s^2+1)^2}+\frac{2s+1}{s^2+1}.$$

两边取拉氏逆变换，得

$$y(t)=(t-2)e^t+(t+1)\sin t+2\cos t.$$

计算的Python程序如下：

```
#程序文件Pex8_11-2.py
import sympy as sp
sp.var('t',positive=True); sp.var('s')   #定义符号变量
sp.var('Y',cls=sp.Function)   #定义符号函数
g=4*t*sp.exp(t)
Lg=sp.laplace_transform(g,t,s)   #方程右端项的拉氏变换
d=s**4*Y(s)+2*s**2*Y(s)+Y(s)
de=d-Lg[0]      #定义取拉氏变换后的代数方程
```

```
Ys=sp.solve(de,Y(s))[0]   #求像函数
Ys=sp.factor(Ys)
yt=sp.inverse_laplace_transform(Ys,s,t)
print("y(t)=",yt); yt=yt.rewrite(sp.exp)
#这里的变换只是为了把解化成指数函数，并且不出现虚数
yt=yt.as_real_imag(); print("y(t)=",yt)
yt=sp.simplify(yt[0]); print("y(t)=",yt)
```

例 8.12　解方程组

$$\begin{cases} 2x'' = -6x + 2y, \\ y'' = 2x - 2y + 40\sin 3t. \end{cases}$$

初值条件为

$$x(0) = x'(0) = y(0) = y'(0) = 0.$$

并画出解曲线 $x(t), y(t)$。

解　记$X(s)=L(x(t))$，$Y(s)=L(y(t))$，利用初值条件，对微分方程组两边取拉氏变换得

$$\begin{cases} 2s^2X(s)=-6X(s)+2Y(s), \\ s^2Y(s)=2X(s)-2Y(s)+\dfrac{120}{s^2+9}. \end{cases}$$

解之，得

$$\begin{cases} X(s)=\dfrac{120}{(s^2+1)(s^2+4)(s^2+9)}=\dfrac{5}{s^2+1}+\dfrac{8}{s^2+4}+\dfrac{3}{s^2+9}, \\ Y(s)=\dfrac{120(s^2+3)}{(s^2+1)(s^2+4)(s^2+9)}=\dfrac{10}{s^2+1}+\dfrac{8}{s^2+4}-\dfrac{18}{s^2+9}. \end{cases}$$

取拉氏逆变换得

$$\begin{cases} x(t) = 5\sin t - 4\sin 2t + \sin 3t, \\ y(t) = 10\sin t + 4\sin 2t - 6\sin 3t. \end{cases}$$

解曲线$x(t), y(t)$的图形见图8.5。

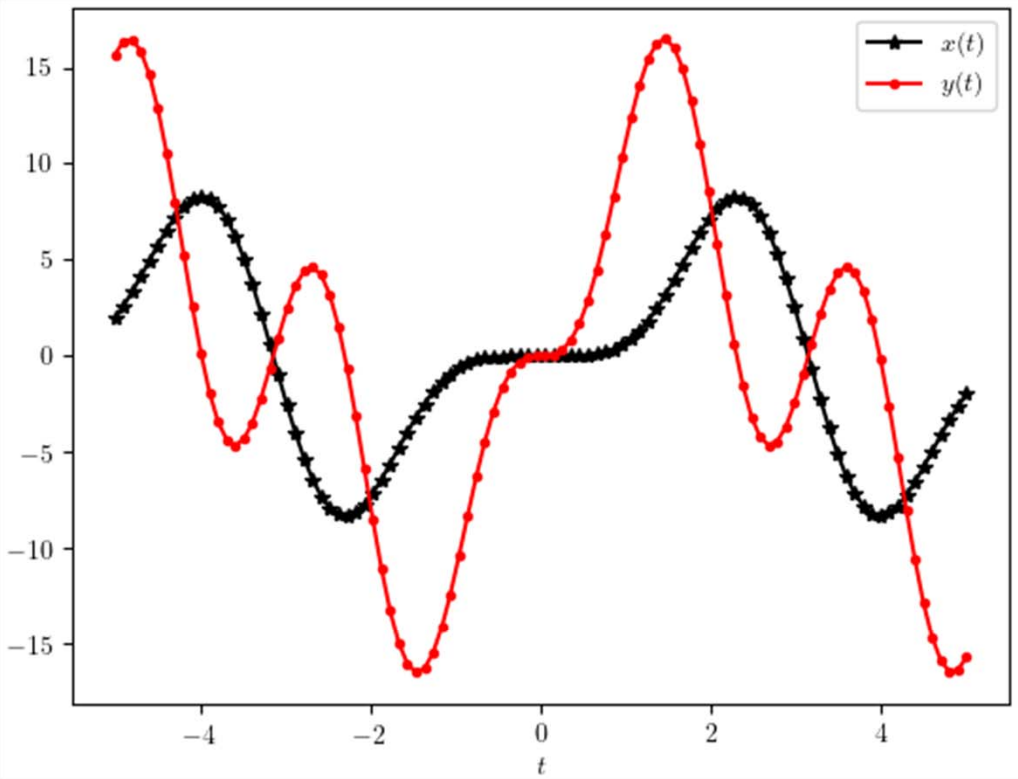


图 8.5　$x(t)$和$y(t)$的解曲线

计算及画图的Python程序如下：

```
#程序文件Pex8_12.py
import sympy as sp
import pylab as plt
import numpy as np
sp.var('t',positive=True); sp.var('s')   #定义符号变量
sp.var('X,Y',cls=sp.Function)   #定义符号函数
g=40*sp.sin(3*t)
Lg=sp.laplace_transform(g,t,s)
```

```
eq1=2*s**2*X(s)+6*X(s)-2*Y(s)
eq2=s**2*Y(s)-2*X(s)+2*Y(s)-Lg[0]
eq=[eq1,eq2]      #定义取拉氏变换后的代数方程组
XYs=sp.solve(eq,(X(s),Y(s)))   #求像函数
Xs=XYs[X(s)]; Ys=XYs[Y(s)]
Xs=sp.factor(Xs); Ys=sp.factor(Ys)
xt=sp.inverse_laplace_transform(Xs,s,t)
yt=sp.inverse_laplace_transform(Ys,s,t)
print("x(t)=",xt);
```

```
print("y(t)=",yt)
fx=sp.lambdify(t,xt,'numpy')   #转换为匿名函数
fy=sp.lambdify(t,yt,'numpy')
t=np.linspace(-5,5,100)
plt.rc('text',usetex=True)
plt.plot(t,fx(t),'*-k',label='$x(t)$')
plt.plot(t,fy(t),'.-r',label='$y(t)$')
plt.xlabel('$t$'); plt.legend(); plt.show()
```