第 3 章

科学计算实战

MATLAB 具有出色的数值计算能力，占据世界上数值计算软件的主导地位。数值计算是数学建模重要的一部分内容，本章将对 MATLAB 中的数值计算进行分析。

3.1 数值积分和微分方程

3.1.1 数值积分和微分方程概述

数值积分是工程师和科学家经常使用的基本工具，用来计算无法解析求解的定积分的近似解。例如，$\Phi(x)=\int_0^x \frac{t^3}{e^t-1}dt$ 不存在 $\Phi(x)$的解析解，要求 $\Phi(5)$，就要通过数值积分的方法来计算。数值积分的目的是，通过在有限个采样点上计算 $f(x)$的值来逼近 $f(x)$在区间$[a,b]$上的定积分。

设 $a=x_0<x_1<\cdots<x_M=b$，称形如：

$$Q[f]=\sum_{k=0}^{M} w_k f(x_k)=w_0 f(x_0)+w_1 f(x_1)+\cdots+w_M f(x_M)$$

且具有性质 $\int_a^b f(x)dx=Q[f]+E[f]$ 的公式为数值积分或面积公式。项 $E[f]$称为积分的截断误差，值$\{x_k\}_{k=0}^{M}$ 称为面积节点，$\{w_k\}_{k=0}^{M}$ 称为权。

无论函数表达式是否已知，数值积分函数都可以求积分的近似值。

(1) 当知道如何计算函数时，可以使用 integral 计算具有指定边界的积分。

(2) 要对底层方程未知的一组数据进行积分，可以使用 trapz，它用数据点形成一系列面积易于计算的梯形，以此执行梯形积分。

3.1.2 数值微积分的应用

对于微分，可以使用 gradient 来求数据数组的微分，它用有限差分公式来计算数值导数。

1. 计算弧线长度的积分

【例 3-1】 参数化曲线并使用 integral 计算弧线长度。

将曲线视为带有参数的方程：

$$x(t)=\sin(2t),\quad y(t)=\cos(t),\quad z(t)=t$$

其中，$t \in [0,3\pi]$。

创建此曲线的三维绘图，如图 3-1 所示。

```
>> t = 0:0.1:3 * pi;
plot3(sin(2 * t),cos(t),t)
```

弧线长度公式表明曲线的长度是参数化方程的导数范数的积分。

$$\int_0^{3\pi} \sqrt{4\cos^2(2t)+\sin^2(t)+1}\,\mathrm{d}t$$

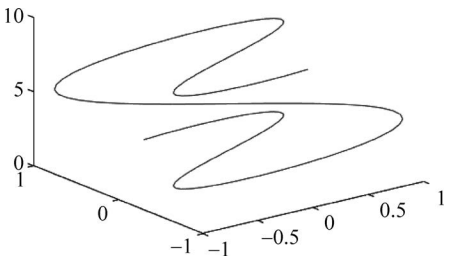

图 3-1　曲线的三维图

将被积函数定义为匿名函数。

```
>> f = @(t) sqrt(4 * cos(2 * t).^2 + sin(t).^2 + 1);
```

通过调用 integral 对此函数进行积分计算。

```
>> len = integral(f,0,3 * pi)
len =
  17.2220
```

由结果可知，此曲线的长度大约为 17.2。

2. 复曲线积分

【例 3-2】 使用 integral 函数的'Waypoints'选项计算复曲线积分。

在 MATLAB 中，可以使用'Waypoints'选项定义直线路径序列，从第一个积分限值到第一个路径点，从第一个路径点到第二个路径点，以此类推，直到从最后一个路径点到第二个积分限值。其实现步骤如下。

(1) 将被积函数定义为匿名函数。

对以下方程求积分：

$$\oint_C \frac{\mathrm{e}^z}{z}\mathrm{d}z$$

其中，C 是一条围绕原点的闭围线的简单极点$\frac{\mathrm{e}^z}{z}$。将被积函数定义为匿名函数：

```
>> fun = @(z) exp(z)./z;
```

(2) 不使用路径点求积分。

可以用参数化计算复值函数的围线积分。一般情况下，指定一条围线，然后将其微分并用于参数化原被积函数。在这种情况下，将围线作为单位圆，但在所有情况下，其结果与所选围线无关。

```
>> g = @(theta) cos(theta) + 1i * sin(theta);
gprime = @(theta) -sin(theta) + 1i * cos(theta);
q1 = integral(@(t) fun(g(t)). * gprime(t),0,2 * pi)
```

这种参数化方法虽然可靠，但难以计算且费时，因为必须先计算导数，然后才能积分。即使是简单函数，也需要写几行代码才能获得正确的结果。由于围绕极点（在本例中为原点）的任何闭围线都有相同的结果，因此可以使用 integral 的'Waypoints'选项构建一个围绕极点的方形或三角形路径。

（3）对不包含极点的围线求积分。

如果路径点向量积分或元素限值为复数，则 integral 会在复平面中针对直线路径序列求积分。围线周围的自然方向为逆时针，指定顺时针围线类似于乘以－1。以这种方式指定围线使其包含一个单函数奇点。如果指定一条不包含极点的围线，则柯西积分定理可保证闭积分环的值是零。

为此，应对远离原点的方围线周围的 fun 求积分。使用相等的积分限值形成一个闭围线。

```
>> C = [2 + i 2 + 2i 1 + 2i];
q = integral(fun,1 + i,1 + i,'Waypoints',C)
q =
  0.0000e + 00 + 2.2204e - 16i
```

其结果数量级为 eps，实际上为零。

（4）对内部包含极点的围线求积分。

指定一个完全在原点包含极点的方围线，然后求积分。

```
>> C = [1 + i - 1 + i - 1 - i 1 - i];
q2 = integral(fun,1,1,'Waypoints',C)
q2 =
   - 0.0000 + 6.2832i
```

这个结果与 q1 的上述计算相符，但使用的代码简单得多。

这个问题的正确答案是 2πi。

```
>> 2 * pi * i
ans =
      0.0000 + 6.2832i
```

3. 积分域内部的奇点

【例 3-3】 拆分积分域以将奇点放在边界上。

实现步骤如下。

（1）将被积函数定义为匿名函数。

复值积分的被积函数：

$$\int_{-1}^{1}\int_{-1}^{1}\frac{1}{\sqrt{x+y}}\mathrm{d}x\,\mathrm{d}y$$

在 $x=y=0$ 时有一个奇点，并通常是 $y=-x$ 线上的奇异值。

将该被积函数定义为匿名函数。

```
>> fun = @(x,y) ((x + y).^( - 1/2));
```

（2）对方形求积分。

对由 $-1\leqslant x\leqslant 1$ 和 $-1\leqslant y\leqslant 1$ 指定的方域中的 fun 求积分。

```
>> format long
q = integral2(fun, - 1,1, - 1,1)
```

警告：非有限结果。积分未成功。可能具有奇异性。

```
q =
                NaN +                  NaNi
```

如果积分区内部有奇异值，则积分不能收敛并返回一个警告。

(3) 将积分域拆分为两个三角形。

可以通过将积分域拆分为互补区并将这些较小的积分加在一起来重新定义积分。可将奇点放在域边界上来避免积分错误和警告。在实例中，可将方积分区域沿着奇异线 $y=-x$ 拆分成两个三角形并将结果相加。

```
>> q1 = integral2(fun, - 1,1, - 1,@(x) - x);
q2 = integral2(fun, - 1,1,@(x) - x,1);      % 求二重积分
q = q1 + q2
q =
    3.771236166328258 - 3.771236166328256i
```

奇异值在边界上时可继续求积分。这个积分的精确值是：

$$\frac{8\sqrt{2}}{3}(1-i)$$

```
>> 8/3 * sqrt(2) * (1 - i)
ans =
     3.771236166328253 - 3.771236166328253i
```

4. 多项式积分的解析解

【例 3-4】 使用 polyint 函数对多项式求解析积分。使用此函数来计算多项式的不定积分。

(1) 定义问题。

考虑实数不定积分：

$$\int(4x^5-2x^3+x+4)\mathrm{d}x$$

被积函数是多项式，解析解是：

$$\frac{2}{3}x^6-\frac{1}{2}x^4+\frac{1}{2}x^2+4x+k$$

其中，k 是积分常量。由于没有指定积分限值，integral 函数族不太适合求解这个问题。

(2) 用向量表示多项式。

创建一个向量，其元素代表各 x 降幂的系数。

```
>> p = [4 0 - 2 0 1 4];
```

(3) 对多项式求解析积分。

使用 polyint 函数求多项式的解析积分。指定第二输入参数的积分常量。

```
>> k = 2;
I = polyint(p,k)
I =
  0.666666666666667   0   - 0.500000000000000   0   0.500000000000000   4.000000000000000
2.000000000000000
```

输出是一个 x 降幂系数向量。这一结果与上述解析解相匹配，但有积分常量 $k=2$。

5. 数值积分

【例 3-5】 对一组离散速度数据进行数值积分以逼近行驶距离。

integral(integral2、integral3 等)族仅接受函数句柄输入，所以这些函数不能用于离散数

据集。当函数表达式不能用于积分时，使用 trapz 或 cumtrapz。

(1) 查看速度数据。

考虑以下速度数据和相应的时间数据。

```
>> vel = [0 .45 1.79 4.02 7.15 11.18 16.09 21.90 29.05 29.05 29.05 29.05 ...
  29.05 22.42 17.9 17.9 17.9 17.9 14.34 11.01 8.9 6.54 2.03 0.55 0];
time = 0:24;
```

这些数据代表汽车的速度(m/s)，间隔为 1s，时间超过 24s。绘制速度数据点并将各点用直线连接，如图 3-2 所示。

```
>> figure
plot(time,vel,'- * ')
grid on
title('汽车速度')
xlabel('时间(s)')
ylabel('速度(m/s)')
```

斜率在加速时为正，恒速时为零，减速时为负。在 $t=0$ 的时间点，车辆处于静止，速度为 vel(1)=0m/s。然后车辆以 vel(9)=29.05m/s 的速度加速，并在 $t=8$s 内达到最大速度，并保持这种速度 4s。然后车辆在 3s 之内减速到 vel(14)=17.9m/s 并最终静止。由于这个速度曲线有多处不连续，因此不能用单一连续函数来描述。

(2) 计算总行驶距离。

trapz 使用数据点进行离散积分以创建梯形，所以它非常适合处理不连续的数据集。这种方法假设在数据点之间为线性行为，当数据点之间的行为是非线性时，精度可能会降低。为了说明这一点，可将数据点作为顶点在图表上画出梯形，如图 3-3 所示。

```
>> xverts = [time(1:end-1); time(1:end-1); time(2:end); time(2:end)];
yverts = [zeros(1,24); vel(1:end-1); vel(2:end); zeros(1,24)];
p = patch(xverts,yverts,'g','LineWidth',2);
```

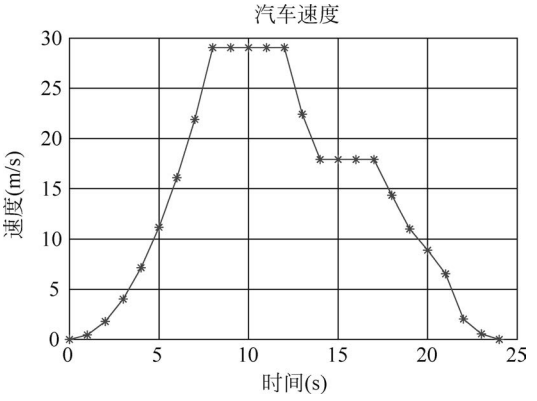

图 3-2 数据点图

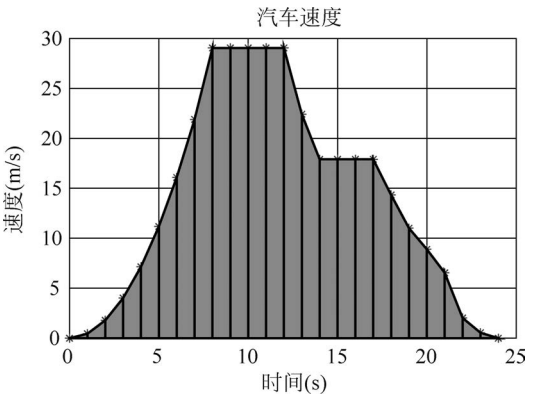

图 3-3 梯形图

trapz 可通过将区域分解成梯形来计算离散数据集下的面积。然后，函数将每个梯形面积累加来计算总面积。

通过使用 trapz 求速度数据积分来计算汽车的总行驶距离(对应的着色区域)。默认情况下，如果使用语法 trapz(Y)，则假定点之间的间距为 1。还可以使用语法 trapz(X,Y)指定不同的均匀或非均匀间距 X。在这种情况下，time 向量中读数之间的间距是 1，因此可以使用默认间距。

```
>> distance = trapz(vel)
distance =
     3.452200000000000e + 02
```

汽车在 $t=24$s 内行驶的距离约为 345.22m。

(3) 绘制累积行驶距离。

cumtrapz 函数与 trapz 密切相关。trapz 仅返回最终的积分值,而 cumtrapz 还在向量中返回中间值。

```
% 计算累积行驶距离并绘制结果
>> cdistance = cumtrapz(vel);
T = table(time',cdistance','VariableNames',{'Time','CumulativeDistance'})
T =
   25 × 2 table
    Time     CumulativeDistance
    ____     __________________
    0                     0
    1                 0.225
    2                 1.345
    3                  4.25
    ……
    22              343.655
    23              344.945
    24               345.22
>> plot(cdistance)          % 如图 3 - 4 所示
title('每秒累计行驶距离')
xlabel('时间(s)')
ylabel('距离(m)')
```

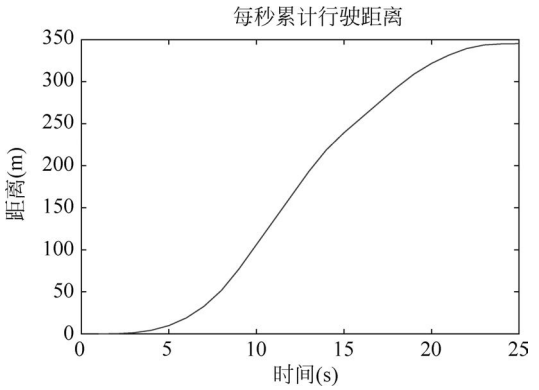

图 3-4　累积行驶距离

6. 计算表面的切平面

【例 3-6】 按有限差分逼近函数梯度,并通过使用这些逼近的梯度,绘制平面上某个点的切平面。

使用函数句柄创建函数 $f(x,y)=x^2+y^2$。

```
>> f = @(x,y) x.^2 + y.^2;
```

使用 gradient 函数,相对 x 和 y 逼近 $f(x,y)$的偏导数。选择与网格大小相同的有限差分长度。

```
>> [xx,yy] = meshgrid( - 5:0.25:5);
[fx,fy] = gradient(f(xx,yy),0.25);
```

曲面上的点 $P=(x_0,y_0,f(x_0,y_0))$的切平面表示为：

$$z=f(x_0,y_0)+\frac{\partial f(x_0,y_0)}{\partial x}(x-x_0)+\frac{\partial f(x_0,y_0)}{\partial y}(y-y_0)$$

$\boldsymbol{f}_x$ 和 $\boldsymbol{f}_y$ 矩阵是偏导数$\frac{\partial f}{\partial x}$和$\frac{\partial f}{\partial y}$的近似值。实例中的相关点(即切平面与函数平面的接合点)为$(x_0,y_0)=(1,2)$。此相关点位置的函数值为 $f(1,2)=5$。为逼近切平面 z,需要求取相关点的导数值。获取该点的索引,并求取该位置的近似导数。

```
>> x0 = 1;
y0 = 2;
t = (xx == x0) & (yy == y0);
indt = find(t);
fx0 = fx(indt);
fy0 = fy(indt);
```

使用切平面 z 的方程创建函数句柄。

```
>> z = @(x,y) f(x0,y0) + fx0 * (x - x0) + fy0 * (y - y0);
```

绘制原始函数 $f(x,y)$、点 P,以及在 P 位置与函数相切的平面 z 的片段,如图 3-5 所示。

```
>> surf(xx,yy,f(xx,yy),'EdgeAlpha',0.7,'FaceAlpha',0.9)
hold on
surf(xx,yy,z(xx,yy))
plot3(1,2,f(1,2),'r * ')
>> % 查看侧剖图,如图 3 - 6 所示
view( - 135,9)
```

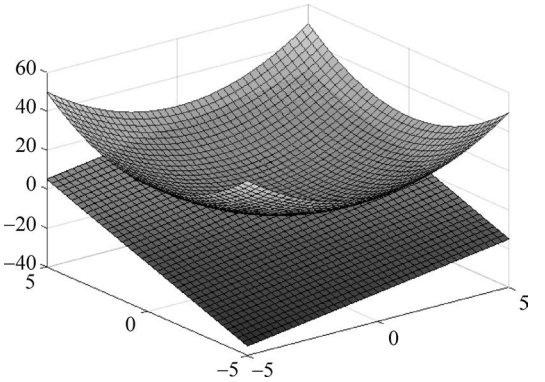

图 3-5　函数的切平面

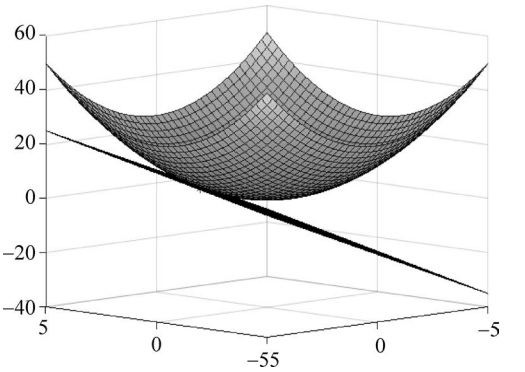

图 3-6　函数切平面的侧剖图

3.2　常微分方程

在高数中常讨论的常微分方程(ODE)求解方法都是一些求典型方程的解析方法。然而,在实际工程中遇到的微分方程往往比较复杂,在很多情况下,都不能给出解析表达式;有些虽然能给出解析表达式,但因计算量太大而不实用。以上说明用求解析解的基本方法来计算微分方程的解往往是不适宜的,甚至很难办到。所以,研究微分方程的数值法就显得十分必要了。

下面考虑微分方程的初值问题的数值解法。用一阶显示的微分方程组来描述为：

$$\dot{\boldsymbol{y}}(t)=y(t,y(t))$$

其中，$\dot{\boldsymbol{y}}(t)$为 n 维列向量，称为状态向量；$\boldsymbol{y}(t)$为 n 维行向量，可以是任意非线性函数。初值问题可做如下理解，已知初始状态 $\boldsymbol{y}_0=[y_1(0),y_2(0),\cdots,y_n(0)]^{\mathrm{T}}$，用数值方法求出某个时间区间 $t\in[0,t_n]$内在步长间隔上的各个时刻状态变量 $y(t)$的数值解。

MATLAB 中的常微分方程求解器可对具有各种属性的初始值问题进行求解。求解器可以处理刚性或非刚性问题、具有质量矩阵的问题、微分代数方程(DAE)或完全隐式问题。

3.2.1　ODE 求解器

1. 常微分方程

常微分方程包含与一个自变量 t(通常称为时间)相关的因变量 y 的一个或多个导数。此处用于表示 y 相对于 t 的导数的表示法，对于一阶导数为 y'，对于二阶导数为 y''，以此类推。ODE 的阶数等于 y 在方程中出现的最高阶导数。

例如，这是一个二阶 ODE：

$$y''=9y$$

在初始值问题中，从初始状态开始解算 ODE。利用初始条件 y_0 以及要在其中求得答案的时间段(t_0,t_f)以迭代方式获取解。在每一步，求解器都对之前各步的结果应用一个特定算法。在第一个这样的时间步，初始条件将提供继续积分所需的必要信息。最终结果是，ODE 求解器返回一个时间步向量 $\boldsymbol{t}=[t_0,t_1,t_2,\cdots,t_f]$以及在每一步对应的解 $\boldsymbol{y}=[y_0,y_1,y_2,\cdots,y_f]$。

MATLAB 中的 ODE 求解器可以解算以下类型的一阶 ODE。

(1) $y'=f(t,y)$形式的显式 ODE。

(2) $\boldsymbol{M}(t,y)y'=f(t,y)$形式的线性隐式 ODE。其中，$\boldsymbol{M}(t,y)$为非奇异质量矩阵。该质量矩阵可能为时间或状态相关的矩阵，也可能为常量矩阵。线性隐式 ODE 涉及在质量矩阵中编码的一阶 y 导数的线性组合。

(3) 线性隐式 ODE 可随时变换为显式形式 $y'=\boldsymbol{M}^{-1}(t,y)f(t,y)$。不过，将质量矩阵直接指定给 ODE 求解器可避免这种既不方便还可能带来大量计算开销的变换操作。

(4) 如果 y'的某些分量缺失，则这些方程称为微分代数方程(DAE)，并且 DAE 方程组会包含一些代数变量。代数变量是导数未出现在方程中的因变量。可通过对方程求导来将 DAE 方程组重写为等效的一阶 ODE 方程组，以消除代数变量。将 DAE 重写为 ODE 所需的求导次数称为微分指数。ode15s 和 ode23t 求解器可解算微分指数为 1 的 DAE。

(5) $f(t,y,y')=0$ 形式的完全隐式 ODE。完全隐式 ODE 不能重写为显式形式，还可能包含一些代数变量。ode15i 求解器专为完全隐式问题(包括微分指数为 1 的 DAE)而设计。

(6) 可通过使用 odeset 函数创建 options 结构体，来针对某些类型的问题为求解器提供附加信息。

1) ODE 方程

可以指定需要解算的任意数量的 ODE 耦合方程，原则上，方程的数量仅受计算机可用内存的限制。如果方程组包含 n 个方程：

$$\begin{pmatrix}y'_1\\y'_2\\\vdots\\y'_n\end{pmatrix}=\begin{pmatrix}f_1(t,y_1,y_2,\cdots,y_n)\\f_2(t,y_1,y_2,\cdots,y_n)\\\vdots\\f_n(t,y_1,y_2,\cdots,y_n)\end{pmatrix}$$

则用于编写该方程组代码的函数将返回一个向量，其中包含 n 个元素，对应于 $y'_1, y'_2, \cdots, y'_n$ 值。例如，考虑以下包含两个方程的方程组：

$$\begin{cases} y'_1 = y_2 \\ y'_2 = y_1 y_2 - 2 \end{cases}$$

用于编写该方程组代码的函数为：

```
function dy = M_ODE(t,y)
dy(1) = y(2);
dy(2) = y(1) * y(2) - 2;
```

2）高阶 ODE

MATLAB ODE 求解器仅可解算一阶方程，因此必须使用常规代换法，将高阶 ODE 重写为等效的一阶方程组：

$$\begin{aligned} y_1 &= y \\ y_2 &= y' \\ y_3 &= y'' \\ &\vdots \\ y_n &= y^{(n-1)} \end{aligned}$$

这些代换将生成一个包含 n 个一阶方程的方程组：

$$\begin{cases} y'_1 = y_2 \\ y'_2 = y_3 \\ \vdots \\ y'_n = f(t, y_1, y_2, \cdots, y_n) \end{cases}$$

例如，考虑三阶 ODE：

$$y''' - y'' y + 1 = 0$$

使用代换法：

$$\begin{aligned} y_1 &= y \\ y_2 &= y' \\ y_3 &= y'' \end{aligned}$$

生成等效的一阶方程组：

$$\begin{cases} y'_1 = y_2 \\ y'_2 = y_3 \\ y'_3 = y_1 y_3 - 1 \end{cases}$$

此方程组的代码则为：

```
function dydt = f(t,y)
dydt(1) = y(2);
dydt(2) = y(3);
dydt(3) = y(1) * y(3) - 1;
```

3）复数 ODE

考虑复数 ODE 方程：

$$y' = f(t, y)$$

其中，$y = y_1 + i y_2$。为解算该方程，需要将实部和虚部分解为不同的解分量，最后重新组合相

应的结果。从概念上讲，这类似于：

$$y_v = [\text{Real}(y) \quad \text{Imag}(y)]$$

$$f_v = [\text{Real}(f(t,y)) \quad \text{Imag}(f(t,y))]$$

例如，如果 ODE 为 $y' = y_t + 2i$，则可以使用函数文件来表示该方程。

```
function f = complex f(t,y)
% 定义接受和返回复数的函数
f = y. * t + 2 * i;
```

然后，分解实部和虚部的代码为：

```
function fv = imaginaryODE(t,yv)
% 从实部和虚部构造 y
y = yv(1) + i * yv(2);
% 评估函数
yp = complex f(t,y);
% 返回实部和虚部
fv = [real(yp); imag(yp)];
```

在运行求解器以获取解时，初始条件 y_0 也会分解为实部和虚部，以提供每个解分量的初始条件。

```
>> y0 = 1 + i;
yv0 = [real(y0); imag(y0)];
tspan = [0 2];
[t,yv] = ode45(@imaginaryODE, tspan, yv0);
```

获得解后，将实部和虚部分量组合到一起可获得最终结果。

```
y = yv(:,1) + i * yv(:,2);
```

2. 捕食者-猎物方程

本节内容说明如何使用 ode23 和 ode45 求解表示捕食者/猎物模型的微分方程。这两个函数用于对使用可变步长大小 Runge-Kutta 积分方法的常微分方程求数值解。ode23 使用一对简单的 2 阶和 3 阶公式实现中等精度，ode45 使用一对 4 阶和 5 阶公式实现更高的精度。

1) Runge-Kutta 积分法

对于一阶微分方程的初值问题，在求解未知函数 y 时，y 在 t_0 点的值 $y(t_0) = y_0$ 是已知的，并且根据高等数学中的中值定理，应有：

$$\begin{cases} y(t_0 + h) = y_1 \approx y_0 + hf(t_0, y_0) \\ y(t_0 + 2h) = y_2 \approx y_1 + hf(t_1, y_1) \end{cases}, \quad h > 0$$

一般地，在任意点 $t_i = t_0 + hi$，有

$$y(t_0 + ih) = y_i \approx y_{i-1} + hf(t_{i-1}, y_{i-1}), \quad i = 1, 2, \cdots, n$$

当 (t_0, y_0) 确定后，根据上述递推式能计算出未知函数 y 在点 $t_i = t_0 + hi$ 的一列数值解：

$$y_i = y_0, y_1, y_2, \cdots, y_n$$

当然，递推过程中有一个误差累计的问题。在实际计算过程中，使用的递推公式一般进行过改造，著名的 Runge-Kutta 公式为：

$$y(t_0 + ih) = y_i \approx y_{i-1} + \frac{h}{6}(k_1 + 2k_2 + 3k_3 + 4k_4)$$

其中，

$$\begin{cases} k_1 = f(t_{i-1}, y_{i-1}) \\ k_2 = f\left(t_{i-1} + \dfrac{h}{2}, y_{i-1} + \dfrac{h}{2}k_1\right) \\ k_3 = f\left(t_{i-1} + \dfrac{h}{2}, y_{i-1} + \dfrac{h}{2}k_2\right) \\ k_4 = f\left(t_{i-1} + \dfrac{h}{2}, y_{i-1} + \dfrac{h}{2}k_3\right) \end{cases}$$

2）求解非刚性 ODE

MATLAB 拥有三个非刚性 ODE 求解器，分别为 ode45、ode23、ode113 函数。对于大多数非刚性问题，ode45 的性能最佳。但对于允许较宽松的误差容限或刚度适中的问题，建议使用 ode23。同样，对于具有严格误差容限的问题，ode113 可能比 ode45 更加高效。

如果非刚性求解器需要很长时间才能解算问题或总是无法完成积分，则该问题可能是刚性问题。

【例 3-7】 以名为 Lotka-Volterra 方程，也即捕食者-猎物模型的一对一阶常微分方程为例：

$$\frac{\mathrm{d}x}{\mathrm{d}t} = x - \alpha xy$$

$$\frac{\mathrm{d}y}{\mathrm{d}t} = -y + \beta xy$$

变量 x 和 y 分别计算猎物和捕食者的数量。二次交叉项表示物种之间的交叉。当没有捕食者时，猎物数量将增加，当猎物匮乏时，捕食者数量将减少。

（1）编写方程代码。

为了模拟系统，需要创建一个函数，以返回给定状态和时间值时的状态导数的列向量。在 MATLAB 中，两个变量 x 和 y 可以表示为向量 $\boldsymbol{y}$ 中的前两个值。同样，导数是向量 $\boldsymbol{y}_p$ 中的前两个值。函数必须接受 t 和 y 的值，并在 $\boldsymbol{y}_p$ 中返回公式生成的值。

```
yp(1) = (1 - alpha * y(2)) * y(1)
yp(2) = (-1 + beta * y(1)) * y(2)
```

在实例中，公式包含在名为 lotka.m 的文件中。文件使用 $\alpha=0.01$ 和 $\beta=0.02$ 的参数值：

```
>> type lotka
function yp = lotka(t,y)
 % LOTKA Lotka - Volterra 捕食者 - 猎物模型。
yp = diag([1 - .01 * y(2), -1 + .02 * y(1)]) * y;
```

（2）模拟系统。

使用 ode23 在区间 $0<t<15$ 中求解 lotka 中定义的微分方程。使用初始条件 $x(0)=y(0)=20$，使捕食者和猎物的数量相等。

```
>> t0 = 0;
tfinal = 15;
y0 = [20; 20];
[t,y] = ode23(@lotka,[t0 tfinal],y0);
```

（3）绘制结果。

绘制两个种群数量对时间图，如图 3-7 所示。

```
>> plot(t,y)
title('捕食者/猎物种群随时间的变化')
```

```
xlabel('时间')
ylabel('种群')
legend('猎物','捕食者','Location','North')
```

下面绘制两个种群数量的相对关系图(相平面图),如图 3-8 所示。生成的相平面图非常清晰地表明了二者数量之间的循环关系。

```
>> plot(y(:,1),y(:,2))
title('相平面图')
xlabel('猎物种群')
ylabel('捕食者种群')
```

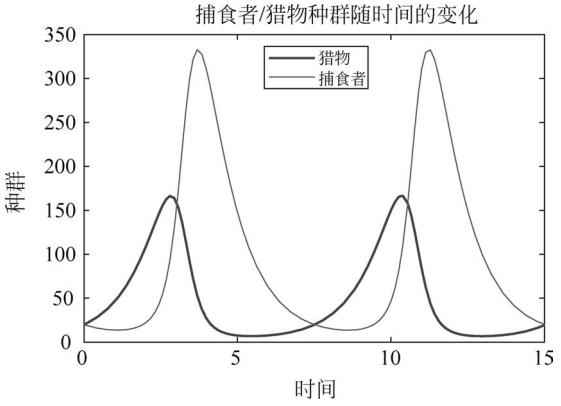

图 3-7　两个种群数量对时间图

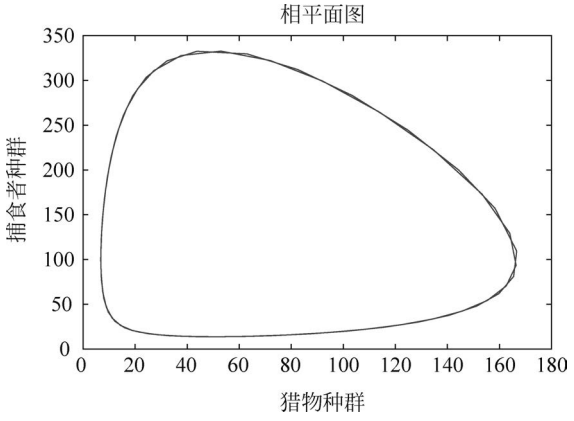

图 3-8　两个种群的相平面图

(4) 比较不同求解器的结果。

使用 ode4 再次求解该方程组,ode45 求解器的每一步都需要更长的时间,但它的步长也更大。然而,ode45 的输出是平滑的,因为在默认情况下,此求解器使用连续展开公式在每个步长范围内的四个等间距时间点生成输出(可以使用'Refine'选项调整时间点数)。绘制两个解进行比较,如图 3-9 所示。

```
>> [T,Y] = ode45(@lotka,[t0 tfinal],y0);
plot(y(:,1),y(:,2),'-.',Y(:,1),Y(:,2),'-');
title('相平面图')
legend('ode23','ode45')
```

由图 3-9 可看出,使用不同的数值方法求解微分方程会产生略微不同的答案。

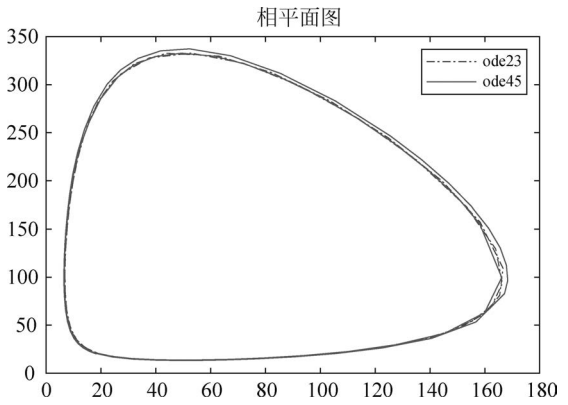

图 3-9　比较两个求解器的相平面图

3. 解算刚性 ODE

MATLAB 拥有四个专用于刚性 ODE 的求解器，分别为 ode15s、ode23s、ode23t、ode23tb 函数。对于大多数刚性问题，ode15s 的性能最佳。但如果问题允许较宽松的误差容限，则 ode23s、ode23t 和 ode23tb 可能更加高效。

1）什么是刚性 ODE

对于一些 ODE 问题，求解器采用的步长被强制缩小为与积分区间相比过小的级别，甚至在解曲线平滑的区域也是如此。这些步长可能过小，以至于遍历很短的时间区间都可能需要数百万次计算。这可能导致求解器积分失败，即使积分成功也需要花费很长时间。

导致 ODE 求解器出现此行为的方程称为刚性方程。刚性 ODE 造成的问题是，显式求解器（例如 ode45）获取解的速度慢得令人无法忍受。这是将 ode45 与 ode23 和 ode113 一同归类为非刚性求解器的原因所在。

专用于刚性 ODE 的求解器称为刚性求解器，它们通常在每一步中完成更多的计算工作。这样做的好处是，它们能够采用大得多的步长，并且与非刚性求解器相比提高了数值稳定性。

2）求解器选项

对于刚性问题，使用 odeset 指定 Jacobian 矩阵尤为重要。刚性求解器使用 Jacobian 矩阵 $\frac{\partial f_i}{\partial y_i}$ 来预测 ODE 在积分过程中的局部行为，因此提供 Jacobian 矩阵（或者对于大型稀疏方程组提供其稀疏模式）对于提高效率和可靠性至关重要。使用 odeset 的 Jacobian、JPattern 或 Vectorized 选项来指定 Jacobian 的相关信息。如果没有提供 Jacobian，则求解器将使用有限差分对其进行数值预测。

【例 3-8】 van der Pol 方程为二阶 ODE：

$$y'' - \mu(1 - y_1^2)y'_1 + y_1 = 0$$

其中，$\mu > 0$ 为标量参数。当 $\mu = 1$ 时，生成的 ODE 方程组为非刚性方程组，可以使用 ode45 轻松求解。但如果将 μ 增大至 1000，则解会发生显著变化，并会在明显更长的时间段中显示振荡。求初始值问题的近似解变得更加复杂。由于此特定问题是刚性问题，因此专用于非刚性问题的求解器（如 ode45）的效率非常低下且不切实际。针对此问题应改用 ode15s 等刚性求解器。

通过执行代换 $y'_1 = y_2$，将该 van der Pol 方程重写为一阶 ODE 方程组，为：

$$y'_1 = y_2$$

$$y'_2 = \mu(1 - y_1^2)y_2 - y_1$$

vdp1000 函数使用 $\mu=1000$ 计算 van der Pol 方程。

```
function dydt = vdp1000(t,y)
% vdp1000 计算 mu = 1000 的 van der Pol ODE
dydt = [y(2); 1000*(1-y(1)^2)*y(2)-y(1)];
```

使用 ode15s 函数和初始条件向量[2; 0],在时间区间[0 3000]上解算此问题。由于是标量,因此仅绘制解的第一个分量,如图 3-10 所示。

```
[t,y] = ode15s(@vdp1000,[0 3000],[2; 0]);
plot(t,y(:,1),'-o');
title('van der Pol 方程的解, \mu = 1000');
xlabel('时间 t');
ylabel('解 y_1');
```

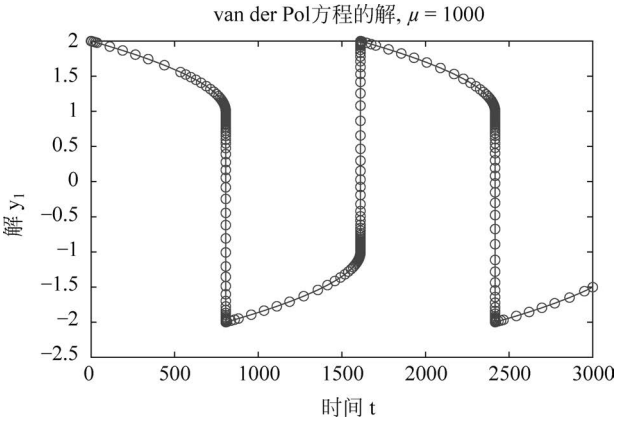

图 3-10　第一个分量解

vdpode 函数也可以求解同一问题,但它接受的是用户指定的 μ 值。随着 μ 的增大,该方程组的刚性逐渐增强。

3.2.2　边界值问题

边界值问题(BVP)是受限于边界条件的常微分方程。与初始值问题不同,BVP 可以有一个有限解、无解或有无限多个解。解的初始估计值是求解 BVP 必不可少的一部分,估计值的质量对于求解器性能乃至计算成功与否都至关重要。bvp4c 和 bvp5c 求解器适用于具有两点边界条件、多点条件、奇异值或未知参数的边界值问题。

在边界值问题(BVP)中,目标是求常微分方程(ODE)的解,该解还需满足某些指定的边界条件。边界条件指定积分区间中两个或多个位置处的解的值之间的关系。在最简单的情形中,边界条件适用于区间的开始和结束(即边界)。

MATLAB BVP 求解器 bvp4c 和 bvp5c 用于处理以下形式的 ODE 方程组:

$$y'=f(x,y)$$

其中,x 是自变量,y 是因变量,y'表示 y 关于 x 的导数,也写为$\frac{\mathrm{d}y}{\mathrm{d}x}$。

1. 边界条件

在两点 BVP 的最简单情形中,ODE 的解在区间[a,b]中求得,并且必须满足边界条件:

$$g(y(a),y(b))=0$$

要指定给定 BVP 的边界条件,必须编写 res=bcfun(ya,yb)形式的函数,如果涉及未知参

数,则使用 res=bcfun(ya,yb,p)形式。将此函数作为第二个输入参数提供给求解器。该函数返回 res,这是在边界点处的解的残差值。例如,如果 $y(a)=1$ 且 $y(b)=0$,则边界条件函数为:

```
function res = bcfun(ya,yb)
res = [ya(1) - 1
    yb(1)];
end
```

在解的初始估计值中,网格中的第一个和最后一个点指定强制执行边界条件的点。对于上述边界条件,可以指定 bvpinit(linspace(a,b,5),yinit)来对 a 和 b 强制执行边界条件。

MATLAB 中的 BVP 求解器还适用于包含下列各项的其他类型问题:未知参数 p、解具有奇异性、多点条件(将积分区间分成若干区域的内边界)。

在多点边界条件的情形下,边界条件应用于积分区间中两个以上的点。例如,可能要求在区间的开始处、中间处和结束处的解为零。

2. 解的初始估计值

与初始值问题不同,边界值问题的解可以是:无解、有限个解、无限多个解。

求解 BVP 过程的重要部分是提供所需解的估计值。估计值的准确与否对求解器性能甚至是能否成功计算来说至关重要。

使用 bvpinit 函数为解的初始估计值创建结构体。求解器 bvp4c 和 bvp5c 接受此结构体作为第三个输入参数。

3. 查找未知参数

通常,BVP 会涉及需要同时求解的未知 p 参数。ODE 和边界条件变为:

$$y'=f(x,y,p)$$
$$g(y(a),y(b),p)=0$$

在此情况下,边界条件必须足以确定参数 p 的值,并必须为求解器提供任何未知参数的初始估计值。当调用 bvpinit 来创建结构体 solinit 时,请在第三个输入参数 parameters 中指定初始估计值向量。

```
solinit = bvpinit(x,v,parameters)
```

此外,用于编写 ODE 方程和边界条件代码的函数 odefun 和 bcfun 都必须具有第三个参数。

```
dydx = odefun(x,y,parameters)
res = bcfun(ya,yb,parameters)
```

求解微分方程时,求解器会调整未知参数的值以满足边界条件。求解器会在 sol.parameters 中返回未知参数的最终值。

4. 奇异 BVP

bvp4c 和 bvp5c 可对以下形式的一类奇异 BVP 求解:

$$y'=\frac{1}{x}\boldsymbol{S}y+f(x,y)$$
$$0=g(y(a),y(b))$$

该求解器还可以接受以下形式的问题的未知参数。

$$y'=\frac{1}{x}\boldsymbol{S}y+f(x,y,p)$$

$$0=g(y(a),y(b),p)$$

奇异问题必须位于[0,b]区间上，且$b>0$。使用 bpset 将常量矩阵 $\boldsymbol{S}$ 作为'SingularTerm'选项的值传递给求解器。$x=0$ 时的边界条件必须与平滑解 $\boldsymbol{S}y_0=0$ 的必要条件一致。解的初始估计值也应该满足此条件。

当求解奇异 BVP 时，求解器要求函数 odefun(x,y)只返回方程中 $f(x,y)$项的值。涉及 $\boldsymbol{S}$ 的项由求解器使用'SingularTerm'选项单独处理。

【例 3-9】 使用 bvp4c 和两个不同的初始估计值来估计 BVP 的两个解。

假设有微分方程 $y''+e^{y}=0$，边界条件为：$y(0)=y(1)=0$。

要在 MATLAB 中对该方程求解，需要先编写方程和边界条件的代码，然后为解生成合适的初始估计值，再调用边界值问题求解器 bvp4c。

(1) 编写方程代码。

根据需要，创建一个函数以编写方程代码，函数名为 bvpfun。

```
function dydx = bvpfun(x,y)
dydx = [y(2)
        -exp(y(1))];
end
```

求解器自动将这些输入传递给该函数，在实例中，可以将二阶方程重写为一阶方程组：

$$y'_1=y_2$$

$$y'_2=-e^{y_1}$$

(2) 编写边界条件。

对于像此问题中的两点边界值条件，边界条件函数应为 res=bcfun(ya,yb)。其中，ya 和 yb 是求解器自动传递给函数的列向量，bcfun 返回边界条件中的残差。

```
function res = bcfun(ya,yb)
res = [ya(1)
       yb(1)];
end
```

(3) 具体求解过程。

调用 bvpinit 以生成解的初始估计值。x 的网格不需要有很多点，但是第一个点必须为 0，而最后一个点必须为 1，以正确指定边界条件。对 y 使用初始估计值，其中第一个分量为稍大于零的正数，第二个分量为零。

```
>> xmesh = linspace(0,1,5);
solinit = bvpinit(xmesh, [0.1 0]);
```

使用 bvp4c 求解器求解 BVP。

```
>> sol1 = bvp4c(@bvpfun, @bcfun, solinit);
```

使用解的不同初始估计值第二次求解 BVP。

```
>> solinit = bvpinit(xmesh, [3 0]);
sol2 = bvp4c(@bvpfun, @bcfun, solinit);
```

绘制 bvp4c 针对不同的初始条件所计算的解，如图 3-11 所示。这两个解都满足规定的边界条件，但它们之间有不同行为。由于解并不始终唯一，不同行为展现出为解提供良好初始估计值的重要性。

```
>> plot(sol1.x,sol1.y(1,:),'-v',sol2.x,sol2.y(1,:),'-o')
```

```
title('不同解的边值问题取决于初始值估计')
xlabel('x')
ylabel('y')
legend('解 1','解 2')
```

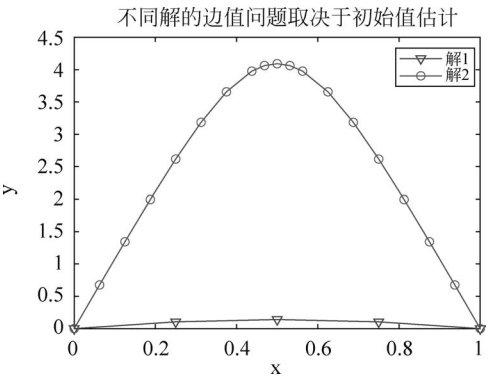

图 3-11　不同的初始条件所计算的解

【例 3-10】 求解多点边界值问题，其中关注的解满足积分区间内的条件。

对于[0,λ]中的 x，考虑以下方程：

$$v' = \frac{C-1}{n}$$

$$C' = \frac{vC - \min(x,1)}{\eta}$$

问题的已知参数是 $n,\kappa,\lambda>1$ 和 $n,\kappa,\lambda>1\eta=\frac{\lambda^2}{n\cdot\kappa^2}$。$C'(x)$的方程中，项 $\min(x,1)$在 $x=1$ 处不平滑，因此该问题不能直接求解。在这种情况下，可以将问题分成两部分：一部分在区间[0,1]内，另一部分在区间[1,λ]内。这两个区域之间的联系是在 $x=1$ 处的解必须为连续的。解还必须满足边界条件：

$$v(0)=0$$

$$C(\lambda)=1$$

每个区域的方程如下。

区域 1：$0\leqslant x\leqslant 1$

$$v' = \frac{C-1}{n}$$

$$C' = \frac{vC - x}{\eta}$$

区域 2：$1\leqslant x\leqslant \lambda$

$$v' = \frac{C-1}{n}$$

$$C' = \frac{vC - 1}{\eta}$$

交界点 $x=1$ 同时包含在这两个区域中。在此交界点上，求解器会产生左解和右解，这两个解必须相等，以确保解的连续性。

(1) 编写方程代码。

$v'(x)$和 $C'(x)$的方程取决于正在求解的区域。对于多点边界值问题，导数函数必须接

受第三个输入参数 region，该参数用于标识正在计算导数的区域。求解器从左到右对区域进行编号，从 1 开始。

创建一个函数 f1，代码为：

```
function dydx = f1(x,y,region,p)
% x 是自变量
% y 是因变量
% dydx(1)给出 v(x)的方程，dydx(2)给出 C'(x)的方程
% region 是计算导数的区域编号(本例中问题分为两个区域，region 为 1 或 2)
% p 是向量，包含常量参数[n,κ,λ,η]的值
n = p(1);
eta = p(4);
dydx = zeros(2,1);
dydx(1) = (y(2) - 1)/n;
switch region
    case 1                                  % x 范围为[0 1]
        dydx(2) = (y(1) * y(2) - x)/eta;
    case 2                                  % x 范围为[1 λ]
        dydx(2) = (y(1) * y(2) - 1)/eta;
end
end
```

(2) 编写边界代码。

在两个区域中求解两个一阶微分方程需要四个边界条件。这些条件中有两个来自原始问题：

$$v(0)=0$$

$$C(\lambda)-1=0$$

另外两个条件强制交界点 $x=1$ 处的左解和右解具备连续性：

$$v_{\mathrm{L}}(1)-v_{\mathrm{R}}(1)=0$$

$$C_{\mathrm{L}}(1)-C_{\mathrm{R}}(1)=0$$

对于多点 BVP，边界条件函数 YL 和 YR 的参数会是矩阵。具体来说，第 k 列 YL(:,k)是第 k 个区域左边界的解，YR(:,k)则是第 k 个区域右边界的解。

在此问题中，$y(0)$通过 YL(:,1)来逼近，而 $y(\lambda)$通过 YR(:,end)来逼近。解在 $x=1$ 处的连续性要求 YR(:,1) = YL(:,2)。

```
function res = bc(YL,YR)
res = [YL(1,1)                      % v(0) = 0
       YR(1,1) - YL(1,2)            % v(x)在 x = 1 处
       YR(2,1) - YL(2,2)            % C(x)在 x = 1 处
       YR(2,end) - 1];              % C(lambda) = 1
end
```

(3) 获取初始估计值。

对于多点 BVP，边界条件自动应用于积分区间的开始处和结束处。但是，必须在 xmesh 中为其他交界点分别指定双重项。满足边界条件的简单估计值是常量估计值 $y=[1;1]$。

```
>> xc = 1;
xmesh = [0 0.25 0.5 0.75 xc xc 1.25 1.5 1.75 2];
yinit = [1; 1];
sol = bvpinit(xmesh,yinit);
```

(4) 求解方程。

定义常量参数的值，并将其放入向量 ***p*** 中。计算 κ 的几个值的解，其中使用每个解作为

下一个解的初始估计值。对于 κ 的每个值，计算渗透性 $O_s=\frac{1}{v(\lambda)}$ 的值。对于循环的每次迭代，将计算值与近似解析解进行比较。

```
>> lambda = 2;
n = 5e-2;
for kappa = 2:5
  eta = lambda^2/(n*kappa^2);
  p = [n kappa lambda eta];
  sol = bvp5c(@(x,y,r) f1(x,y,r,p), @bc, sol);
  K2 = lambda*sinh(kappa/lambda)/(kappa*cosh(kappa));
  approx = 1/(1 - K2);
  computed = 1/sol.y(1,end);
  fprintf(' %2i     %10.3f      %10.3f \n',kappa,computed,approx);
end
  2         1.462         1.454
  3         1.172         1.164
  4         1.078         1.071
  5         1.039         1.034
```

(5) 对解进行绘图。

绘制 $v(x)$ 和 $C(x)$ 的解分量，以及在交界点 $x=1$ 处的垂直线。显示的 $\kappa=5$ 的解是循环的最后一次迭代的结果，效果如图 3-12 所示。

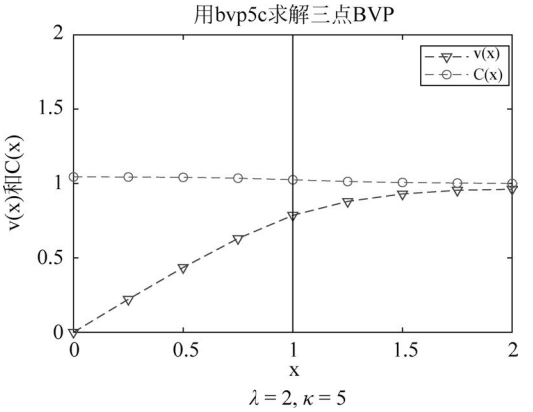

图 3-12　绘制 $v(x)$ 和 $C(x)$ 的解分量

3.2.3　时滞微分方程

时滞微分方程包含的项的值依赖于先前时间的解。时滞可以固定不变、与时间相关或与状态相关，而求解器函数(dde23、ddesd 或 ddensd)的选择取决于方程中的时滞类型。通常，时滞将导数的当前值与某个先前时间的解的值联系起来，但对于中立型方程，导数的当前值依赖于先前时间的导数值。由于方程依赖于先前时间的解，因此有必要提供一个历史记录函数，该函数传递初始时间 t_0 之前的解的值。

1. 时滞微分方程的概述

时滞微分方程(DDE)是当前时间的解与过去时间的解相关的常微分方程。该时滞可以固定不变、与时间相关、与状态相关或与导数相关。要开始积分，通常必须提供历史解，以便求解器可以获取初始积分点之前的时间的解。

1）常时滞 DDE

具有常时滞的微分方程组的形式如下。

$$y'(t)=f(t,\boldsymbol{y}(t),\boldsymbol{y}(t-\tau_1),\cdots,\boldsymbol{y}(t-\tau_k))$$

此处，t 为自变量，$\boldsymbol{y}$ 为因变量的列向量，而 y' 表示 y 关于 t 的一阶导数。时滞 $\tau_1,\tau_2,\cdots,\tau_k$ 是正常量。

dde23 函数用于求解具有历史解 $y(t)=S(t)$（其中，$t<t_0$）的常时滞 DDE。DDE 的解通常是连续的，但其导数不连续。dde23 函数跟踪低阶导数的不连续性，并与 ode23 使用的同一显式 Runge-Kutta(2,3)对和插值求微分方程的积分。对于大于时滞的步长而言，Runge-Kutta 公式是隐式的。当 $y(t)$ 足够平滑以证明此大小的步长时，使用预测-校正迭代法计算隐式公式。

2）时间相关和状态相关的 DDE

常时滞 DDE 是一种特殊情况，更为一般的 DDE 形式为：

$$y'(t)=f(t,y(t),y(\mathrm{d}y_1),\cdots,y(\mathrm{d}y_p))$$

时间相关和状态相关的 DDE 涉及可能依赖于时间 t 和 y 的时滞 $\mathrm{d}y_1,\mathrm{d}y_2,\cdots,\mathrm{d}y_k$。时滞 $\mathrm{d}y_j(t,y)$ 必须满足 $\mathrm{d}y_j(t,y)\leqslant t$（在区间 $[t_0,t_f]$ 上，其中，$t_0<t_f$）。

ddesd 函数用于求具有历史解 $y(t)=S(t)$（其中，$t<t_0$）的时间相关和状态相关 DDE 的解 $y(t)$。ddesd 函数使用标准的四阶显式 Runge-Kutta 法来求积分，并控制自然插值的余值大小。它使用迭代来采用超过时滞的步长。

3）计算特定点的解

使用 deval 函数和任何 DDE 求解器的输出来计算积分区间中的特定点处的解。例如，y=deval(sol,0.5*(sol.x(1)+sol.x(end)))计算积分区间中点处的解。

4）历史解和初始值

对 DDE 求解时，将在区间 $[t_0,t_f]$（其中，$t_0<t_f$）上来逼近解。DDE 表明 $y(t)$ 如何依赖于 t 之前的时间的解（及其可能的导数）的值。例如，具有常时滞时，$y'(t_0)$ 依赖于 $y(t_0-\tau_1)$，$\cdots,y(t_0-\tau_k)$，其中，τ_j 为正常量。因此，$[t_0,\tau_k]$ 上的解依赖于其在 $t\leqslant t_0$ 处具有的值。必须使用历史解函数 $y(t)=S(t)$（其中，$t<t_0$）定义这些值。

2. 具有常时滞的 DDE

【例 3-11】 使用 dde23 对具有常时滞的 DDE（时滞微分方程）方程组求解。

$$y_1'(t)=y_1(t-1)$$
$$y_2'(t)=y_1(t-1)+y_2(t-0.2)$$
$$y_3'(t)=y_2(t)$$

$t\leqslant 0$ 的历史解函数是常量 $y_1'(t)=y_2(t)=y_3(t)=1$。方程中的时滞仅存在于 y 项中，并且时滞本身是常量，因此各方程构成常时滞方程组。

要在 MATLAB 中求解此方程组，需要先编写方程组、时滞和历史解的代码，然后再调用时滞微分方程求解器 dde23，该求解器适用于具有常时滞的方程组。

（1）编写时滞代码。

首先，创建一个向量来定义方程组中的时滞。此方程组有以下两种不同时滞。

① 在第一个分量 $y_1(t-1)$ 中的时滞为 1。

② 在第二个分量 $y_2(t-0.2)$ 中的时滞为 0.2。

dde23 接受时滞的向量参数，其中每个元素是一个分量的常时滞。

```
lags = [1 0.2];
```

(2) 编写方程代码。

创建一个函数来编写方程的代码。此函数名为 ddex1de,代码为:

```
function dydt = ddex1de(t,y,Z)
% t 是时间(自变量)
% y 是解(因变量)
% Z(:,j)用于逼近时滞 y(t - τj),其中,常时滞 τj 由 lags(j)给定
% 求解器会自动将这些输入传递给该函数,但是变量名称决定如何编写方程代码。在这种情况下:
% Z(:,1)→y1(t - 1)
% Z(:,2)→y2(t - 0.2)
  ylag1 = Z(:,1);
  ylag2 = Z(:,2);
  dydt = [ylag1(1);
          ylag1(1) + ylag2(2);
          y(2)];
end
```

(3) 编写历史解代码。

接下来,创建一个函数来定义历史解。历史解是时间 $t \leqslant t_0$ 的解。

```
function s = history(t)
  s = ones(3,1);
end
```

(4) 求解方程。

最后,定义积分区间$[t_0, t_f]$并使用 dde23 求解器对 DDE 求解。

```
tspan = [0 5];
sol = dde23(@ddefun, lags, @history, tspan);
```

(5) 对解进行绘图。

解结构体 sol 具有字段 sol.x 和 sol.y,这两个字段包含求解器在这些时间点所用的内部时间步和对应的解(如果需要在特定点的解,可以使用 deval 来计算在特定点的解)。绘制三个解分量对时间的图,如图 3-13 所示。

```
plot(sol.x,sol.y,'-o')
xlabel('时间 t');
ylabel('解 y');
legend('y_1','y_2','y_3','Location','NorthWest');
```

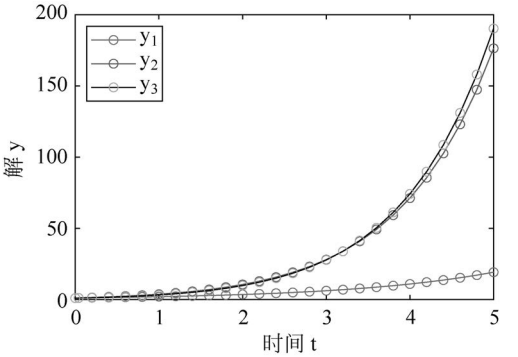

图 3-13　三个解分量对时间的图

【例 3-12】 使用 dde23 对具有不连续导数的心血管模型求解。方程组为:

$$\dot{P}_a(t) = -\frac{1}{c_a R}P_a(t) + \frac{1}{c_a R}P_v(t) + \frac{1}{c_a R}V_{\mathrm{str}}(P_a^{\tau}(t))H(t)$$

$$\dot{P}_v(t)=\frac{1}{c_vR}P_a(t)-\left(\frac{1}{c_vR}+\frac{1}{c_vr}\right)P_v(t)$$

$$\dot{H}(t)=\frac{aHT_s}{1+\gamma HT_p}-\beta HT_p$$

T_s 和 T_p 的项分别是同一方程在有时滞和没有时滞状态下的变体。P_a^τ 和 P_a 分别代表在有时滞和没有时滞状态下的平均动态压。

$$T_s=\frac{1}{1+\left(\frac{P_a^\tau}{a_s}\right)^{\beta_s}}$$

$$T_p=\frac{1}{1+\left(\frac{P_a}{a_p}\right)^{-\beta_p}}$$

此问题有许多物理参数：

- 动脉顺应性 $c_a=1.55\text{ml/mmHg}$。
- 静脉顺应性 $c_v=519\text{ml/mmHg}$。
- 外周阻力 $R=1.05(0.84)\text{mmHg s/ml}$。
- 静脉流出阻力 $r=0.068\text{mmHg s/ml}$。
- 心搏量 $V_{str}=6.79(77.9)\text{ml}$。
- 典型平均动脉压 $P_0=93\text{mmHg}$。
- $a_0=a_s=a_p=93(121)\text{mmHg}$。
- $a_H=0.84\text{sec}^{-2}$。
- $\beta_0=\beta_s=\beta_p=7$。
- $\beta H=1.17$。
- $rH=0$。

该方程组受外周压的巨大影响，外周压会从 $R=1.05$ 急剧减少到 $R=0.84$，从 $t=600$ 处开始。因此，该方程组在 $t=600$ 处的低阶导数具有不连续性。

常历史解由以下物理参数定义：

$$P_a=P_0,\quad P_v(t)=\frac{1}{1+\frac{R}{r}}P_0,\quad H(t)=\frac{1}{RV_{str}}\frac{1}{1+\frac{r}{R}}P_0$$

要在 MATLAB 中求解此方程组，需要先编写方程组、参数、时滞和历史解的代码，然后再调用时滞微分方程求解器 dde23，该求解器适用于具有常时滞的方程组。

(1) 编写方程代码。

创建一个函数来编写方程的代码，函数名为 ddefun。

```
function dydt = ddefun(t,y,Z,p)
% t 为时间(自变量)
% y 是解(因变量)
% Z(n,j)对时滞 y_n(d(j))求近似值,其中,时滞 d(j)由 dely(t,y)的分量 j 给出
% p 是可选的第四个输入,用于传入参数值
    if t <= 600
      p.R = 1.05;
    else
      p.R = 0.21 * exp(600 - t) + 0.84;
    end
```

```
    ylag = Z(:,1);
    Patau = ylag(1);
    Paoft = y(1);
    Pvoft = y(2);
    Hoft = y(3);
    dPadt = - (1 / (p.ca * p.R)) * Paoft ...
            + (1/(p.ca * p.R)) * Pvoft ...
            + (1/p.ca) * p.Vstr * Hoft;
    dPvdt = (1 / (p.cv * p.R)) * Paoft...
            - ( 1 / (p.cv * p.R)...
            + 1 / (p.cv * p.r) ) * Pvoft;
    Ts = 1 / ( 1 + (Patau / p.alphas)^p.betas );
    Tp = 1 / ( 1 + (p.alphap / Paoft)^p.betap );
    dHdt = (p.alphaH * Ts) / (1 + p.gammaH * Tp) ...
            - p.betaH * Tp;
    dydt = [dPadt; dPvdt; dHdt];
end
```

求解器自动将前三个输入传递给函数，变量名称决定如何编写方程代码。调用求解器时，参数结构体 p 将传递给函数。在实例中，时滞表示为：

$$Z(:,1) \rightarrow P_a(t-\tau)$$

(2) 定义物理参数。

将问题的物理参数定义为结构体中的字段。

```
>> p.ca    = 1.55;
p.cv       = 519;
p.R        = 1.05;
p.r        = 0.068;
p.Vstr     = 67.9;
p.alpha0   = 93;
p.alphas   = 93;
p.alphap   = 93;
p.alphaH   = 0.84;
p.beta0    = 7;
p.betas    = 7;
p.betap    = 7;
p.betaH    = 1.17;
p.gammaH   = 0;
```

(3) 编写时滞代码。

接下来，创建变量 tau 来表示项 $P_a^{\tau}(t)=P_a(t-\tau)$的方程中的常时滞 τ。

```
>> tau = 4;
```

(4) 编写历史解代码。

接下来，创建一个向量来定义三个分量 P_a、P_v 和 H 的常历史解。历史解是时间 $t \leqslant t_0$ 的解。

```
>> P0 = 93;
Paval = P0;
Pvval = (1 / (1 + p.R/p.r)) * P0;
Hval = (1 / (p.R * p.Vstr)) * (1 / (1 + p.r/p.R)) * P0;
history = [Paval; Pvval; Hval];
```

(5) 求解方程。

使用 ddeset 来指定在 $t=600$ 处存在不连续性。最后，定义积分区间$[t_0, t_f]$并使用

dde23 求解器对 DDE 求解。使用匿名函数指定 ddefun 以传入参数结构体 p。

```
>> options = ddeset('Jumps',600);
tspan = [0 1000];
sol = dde23(@(t,y,Z) ddefun(t,y,Z,p), tau, history, tspan, options);
```

(6) 对解进行绘图。

解结构体 sol 具有字段 sol.x 和 sol.y,这两个字段包含求解器在这些时间点所用的内部时间步和对应的解(如果需要在特定点的解,可以使用 deval 来计算在特定点的解)。

```
>> % 绘制第三个解分量(心率)对时间的图,如图 3-14 所示
plot(sol.x,sol.y(3,:))
title('压力反射-反馈机制的心率')
xlabel('时间 t')
ylabel('H(t)')
```

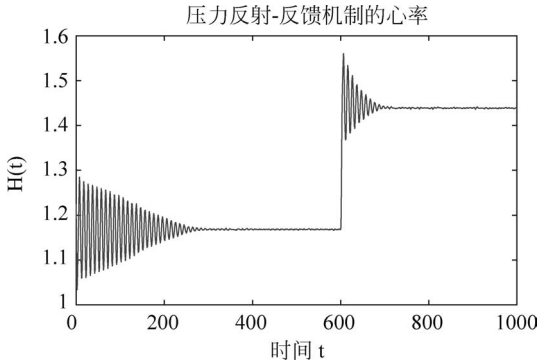

图 3-14　第三个解分量(心率)对时间的图

3.2.4　偏微分方程

偏微分方程包含依赖于若干变量的函数的偏导数。可以利用 MATLAB 中提供的函数,求解时间和一个空间变量的抛物线和椭圆 PDE。

1. 求解偏微分方程

在偏微分方程(PDE)中,要求解的函数取决于几个变量,微分方程可以包括关于每个变量的偏导数。偏微分方程可用于对波浪、热流、流体扩散和其他空间行为随时间变化的现象建模。

1) 求解哪些类型的 PDE

MATLAB PDE 求解器 pdepe 使用一个空间变量 x 和时间 t 对 PDE 方程组的初始边界值问题求解。可以将这些看作一个变量的 ODE,它们也会随着时间而变化。

pdepe 要求解的一维方程大概可分为以下两类。

(1) 带时间导数的方程是抛物形方程。例如,热方程$\dfrac{\partial u}{\partial t}=\dfrac{\partial^2 u}{\partial x^2}$。

(2) 不带时间导数的方程是椭圆形方程。例如,拉普拉斯方程$\dfrac{\partial^2 u}{\partial x^2}=0$。

pdepe 要求方程组中存在至少一个抛物形方程。也就是说,方程组中至少一个方程必须包含时间导数。pdepe 还可求解某些二维和三维问题,这些问题由于角对称而简化为一维问题。

2）求解一维 PDE

一维 PDE 包含函数 $u(x,t)$，该函数依赖于时间 t 和一个空间变量 x。求解器 pdepe 求解以下形式的一维抛物形和椭圆形 PDE 的方程组。

$$c\left(x,t,u,\frac{\partial u}{\partial x}\right)\frac{\partial u}{\partial t}=x^{-m}\frac{\partial}{\partial x}\left(x^{m}f\left(x,t,u,\frac{\partial u}{\partial x}\right)\right)+s\left(x,t,u,\frac{\partial u}{\partial x}\right)$$

方程具有以下属性。

（1）PDE 在 $t_0\leqslant t\leqslant t_f$ 和 $a\leqslant x\leqslant b$ 时成立。

（2）空间区间$[a,b]$必须为有限值。

（3）m 可以是 0、1 或 2，分别对应平面、柱状或球面对称性。如果 $m>0$，则 $a\geqslant 0$ 也必须成立。

（4）系数 $f\left(x,t,u,\frac{\partial u}{\partial x}\right)$是能量项，$s\left(x,t,u,\frac{\partial u}{\partial x}\right)$是源项。

（5）能量项必须取决于偏导数$\frac{\partial u}{\partial x}$。

关于时间的偏导数耦合只限于与对角矩阵 $c\left(x,t,u,\frac{\partial u}{\partial x}\right)$相乘。此矩阵的对角线元素为零或正数。为零的元素对应椭圆形方程，任何其他元素对应抛物形方程。必须至少存在一个抛物形方程。如果 x 的某些孤立值是网格点（即计算解的位置），那么在这些值处，抛物形方程对应的 c 元素可能消失。当物质界面上有网格点时，允许 c 和 s 中出现界面导致的不连续点。

（1）求解过程。

要使用 pdepe 求解 PDE，必须定义 c、f 和 s 的方程系数、初始条件、解在边界处的行为以及在其上计算解的点网格。调用函数 sol=pdepe(m,pdefun,icfun,bcfun,xmesh,tspan)使用以下信息计算指定网格上的一个解。其中，m 是对称常量，pdefun 定义要求解的方程，icfun 定义初始条件，bcfun 定义边界条件，xmesh 是 x 的空间值向量，tspan 是 t 的时间值向量。xmesh 和 tspan 向量共同构成一个二维网格，pdepe 在该网格上计算解。

（2）方程。

必须按照 pdepe 所需的标准形式表示 PDE。以这种形式编写，可以读取系数 c、f、s 的值。在 MATLAB 中可以用以下形式的函数编写方程代码。

```
function [c,f,s] = pdefun(x,t,u,dudx)
c = 1;
f = dudx;
s = 0;
end
```

在实例中，pdefun 定义方程$\frac{\partial u}{\partial t}=\frac{\partial^2 u}{\partial x^2}$。如果有多个方程，则 $\boldsymbol{c}$、$\boldsymbol{f}$ 和 $\boldsymbol{s}$ 均为向量，其中每个元素对应一个方程。

（3）初始条件。

在初始时间 $t=t_0$ 时，针对所有 x，解分量均满足以下格式的初始条件。

$$u(x,t_0)=u_0(x)$$

在 MATLAB 中，可以用以下形式的函数对初始条件进行编码。

```
function u0 = icfun(x)
```

```
u0 = 1;
end
```

在实例中，$u_0=1$ 定义 $u_0(x,t_0)=1$ 的初始条件。如果有多个方程，则 $\boldsymbol{u}_0$ 是一个向量，其中每个元素定义一个方程的初始条件。

(4) 边界条件。

在边界 $x=a$ 或 $x=b$ 时，针对所有 t，解分量满足以下形式的边界条件。

$$p(x,t,u)+q(x,t)f\left(x,t,u,\frac{\partial u}{\partial x}\right)=0$$

$q(x,t)$是对角线矩阵，其元素全部是零或全部是非零。请注意，边界条件以能量 f（而非关于 x 的 u 的偏导数）形式表示。同时，在 $p(x,t,u)$和 $q(x,t)$这两个系数之间，只有 p 可以依赖于 u。在 MATLAB 中，可以用以下形式的函数对边界条件进行编码。

```
function [pL,qL,pR,qR] = bcfun2(xL,uL,xR,uR,t)
pL = uL;
qL = 0;
pR = uR - 1;
qR = 0;
end
```

q_L 和 p_L 是左边界的系数，p_R 和 q_R 是右边界的系数。其中每个元素定义一个方程的边界条件。

$$u_L(x_L,t)=0$$

$$u_R(x_R,t)=0$$

如果有多个方程，则输出 q_L、p_L、p_R 和 q_R 是向量，其中每个元素定义一个方程的边界条件。

(5) 积分选项。

可以选择 MATLAB PDE 求解器中的默认积分属性来处理常见问题。在某些情况下，可以通过覆盖这些默认值来提高求解器的性能。为此，请使用 odeset 创建一个 options 结构体。然后，将该结构体作为最后一个输入参数传递给 pdepe。

```
sol = pdepe(m,pdefun,icfun,bcfun,xmesh,tspan,options)
```

(6) 解的计算。

在用 pdepe 求解方程后，MATLAB 将以三维数组 sol 返回解，其中，sol(i,j,k)包含在 $t(i)$和 $x(j)$处计算的解的第 k 个分量。通常，可以使用命令 u=sol(:,:,k)提取第 k 个解分量。

指定的时间网格仅用于输出目的，不影响求解器采用的内部时间步。但是，指定的空间网格会影响解的质量和速度。求解方程后，可以使用 pdeval 计算 pdepe 采用不同空间网格返回的解结构体。

2. 求解具有不连续性的 PDE

本节实例说明如何求解涉及物质界面的 PDE。物质界面使得问题在 $x=0.5$ 处具有不连续点，初始条件在右边界 $x=1$ 处具有不连续点。

【例 3-13】 以分段 PDE $\begin{cases}\dfrac{\partial u}{\partial t}=x^{-2}\dfrac{\partial}{\partial x}\left(x^2 5\dfrac{\partial u}{\partial x}\right)-1000\mathrm{e}^u, & 0\leqslant x\leqslant 0.5\\ \dfrac{\partial u}{\partial t}=x^{-2}\dfrac{\partial}{\partial x}\left(x^2\dfrac{\partial u}{\partial x}\right)-\mathrm{e}^u, & 0.5\leqslant x\leqslant 1\end{cases}$ 为例。

初始条件为：

$$u(x,0)=0(0\leqslant x\leqslant 1)$$
$$u(1,0)=1(x=1)$$

边界条件为：

$$\frac{\partial u}{\partial x}=0(x=0)$$
$$u(1,t)=1(x=1)$$

在 MATLAB 中求解该方程，需要对方程、初始条件和边界条件编写代码，然后在调用求解器 pdepe 之前选择合适的解网格。

(1) 编写方程。

在编写方程代码前，需要确保它的形式符合 pdepe 求解器的要求。pdepe 所需的标准形式为：

$$c\left(x,t,u,\frac{\partial u}{\partial x}\right)\frac{\partial u}{\partial t}=x^{-m}\frac{\partial}{\partial x}\left(x^{m}f\left(x,t,u,\frac{\partial u}{\partial x}\right)\right)+s\left(x,t,u,\frac{\partial u}{\partial x}\right)$$

在实例中，PDE 采用了正确形式，因此可以读取系数的值。

$$\begin{cases}\dfrac{\partial u}{\partial t}=x^{-2}\dfrac{\partial}{\partial x}\left(x^{2}5\dfrac{\partial u}{\partial x}\right)-1000\mathrm{e}^{u}, & 0\leqslant x\leqslant 0.5\\ \dfrac{\partial u}{\partial t}=x^{-2}\dfrac{\partial}{\partial x}\left(x^{2}\dfrac{\partial u}{\partial x}\right)-\mathrm{e}^{u}, & 0.5\leqslant x\leqslant 1\end{cases}$$

能量项 $f\left(x,t,u,\dfrac{\partial u}{\partial x}\right)$ 和源项 $s\left(x,t,u,\dfrac{\partial u}{\partial x}\right)$ 的值根据 x 的值而变化。系数是：

$$m=2$$
$$c\left(x,t,u,\frac{\partial u}{\partial x}\right)=1$$
$$\begin{cases}f\left(x,t,u,\dfrac{\partial u}{\partial x}\right)=5\dfrac{\partial u}{\partial x}, & (0\leqslant x\leqslant 0.5)\\ f\left(x,t,u,\dfrac{\partial u}{\partial x}\right)=\dfrac{\partial u}{\partial x}, & (0.5\leqslant x\leqslant 1)\end{cases}$$
$$\begin{cases}s\left(x,t,u,\dfrac{\partial u}{\partial x}\right)=-1000\mathrm{e}^{u}, & (0\leqslant x\leqslant 0.5)\\ s\left(x,t,u,\dfrac{\partial u}{\partial x}\right)=-\mathrm{e}^{u}, & (0.5\leqslant x\leqslant 1)\end{cases}$$

可以创建一个函数 pdex2pde 以编写方程代码，函数代码为：

```
function [c,f,s] = pdex2pde(x,t,u,dudx)
% x 是独立的空间变量
% t 是独立的时间变量
% u 是关于 x 和 t 微分的因变量
% dudx 是偏空间导数∂u/∂x
% 输出 c、f 和 s 对应于 pdepe 所需的标准 PDE 形式中的系数
c = 1;
if x <= 0.5
    f = 5 * dudx;
    s = -1000 * exp(u);
else
    f = dudx;
    s = -exp(u);
```

```
end
end
```

(2) 编写初始条件。

接下来,编写一个返回初始条件的函数。初始条件应用在第一个时间值处,并为 x 的任何值提供 $u(x,t_0)$ 的值。对应函数为:

```
function u0 = pdex2ic(x)
if x < 1
    u0 = 0;
else
    u0 = 1;
end
end
```

(3) 编写边界条件。

编写一个计算边界条件的函数。对于区间 $a \leqslant x \leqslant b$ 上的问题,边界条件应用于所有 t 以及 $x=a$ 或 $x=b$ 的情形。求解器所需的边界条件的标准形式是:

$$p(x,t,u)=q(x,t)f\left(x,t,u,\frac{\partial u}{\partial x}\right)=0$$

由于实例具有球面对称性($m=2$),pdepe 求解器会自动强制执行左边界条件以约束在原点的解,并忽略在边界函数中为左边界指定的任何条件。因此,对于左边界条件,可以指定 $p_L=q_L=0$。对于右边界条件,可以用标准形式重写边界条件,并读取 p_R 和 q_R 的系数值。

对于 x,方程为 $u(1,t)=1\rightarrow(u-1)+0\cdot\dfrac{\partial u}{\partial x}=0$。系数是:

$$p_R(1,t,u)=u-1$$

$$q_R(1,t)=0$$

边界函数应使用函数 pdex2bc。函数代码为:

```
function [pl,ql,pr,qr] = pdex2bc(xl,ul,xr,ur,t)
% 对于左边界,输入 xl 和 ul 对应于 u 和 x
% 对于右边界,输入 xr 和 ur 对应于 u 和 x
% t 是独立的时间变量
% 对于左边界,输出 pl 和 ql 对应于 pL(x,t,u)和 qL(x,t)(对于此问题,x = 0)
% 对于右边界,输出 pr 和 qr 对应于 pR(x,t,u)和 qR(x,t)(对于此问题,x = 1)
pl = 0;
ql = 0;
pr = ur - 1;
qr = 0;
end
```

(4) 选择解网格。

空间网格应包括 $x=0.5$ 附近的几个值以表示不连续界面,并包括 $x=1$ 附近的点,因为在该点上具有不一致的初始值($u(1,0)=1$)和边界值($u(1,t)=0$)。对于较小的 t,解的变化很快,因此请使用可以解析这种急剧变化的时间步。

```
x =[0 0.1 0.2 0.3 0.4 0.45 0.475 0.5 0.525 0.55 0.6 0.7 0.8 0.9 0.95 0.975
    0.99 1];
t =[0 0.001 0.005 0.01 0.05 0.1 0.5 1];
```

(5) 求解方程。

使用对称性值 m、PDE 方程、初始条件、边界条件以及 x 和 t 的网格来求解方程。

```
m = 2;
sol = pdepe(m,@pdex2pde,@pdex2ic,@pdex2bc,x,t);
```

pdepe 以三维数组 sol 形式返回解，其中，sol(i,j,k)是在 $t(i)$和 $x(j)$处计算的解 u_k 的第 k 个分量的逼近值。sol 的大小是 length(t)×length(x)×length(u_0)，因为 u_0 为每个解分量指定初始条件。对于此问题，u 只有一个分量，因此 sol 是 8×18 矩阵，但通常可以使用命令 u=sol(:,:,k)提取第 k 个解分量。

从 sol 中提取第一个解分量。

```
u = sol(:,:,1);
```

(6) 对解进行绘图。

创建在 x 和 t 的所选网格点上绘制的解 u 的曲面图，如图 3-15 所示。由于 m=2 问题是在具有球面对称性的球面几何中提出的，因此解仅在径向 x 方向上变化。

```
surf(x,t,u)
title('非均匀网格的数值解')
xlabel('距离 x')
ylabel('时间 t')
zlabel('解 u')
```

现在，只需绘制 x 和 u 即可获得曲面图中等高线的侧视图，如图 3-16 所示。在 x=0.5 处添加一条线，以突出材料接口的效果。

```
plot(x,u,x,u,'*')
line([0.5 0.5], [-3 1], 'Color', 'k')
ylabel('时间 t')
ylabel('解 u')
title('侧视图')
```

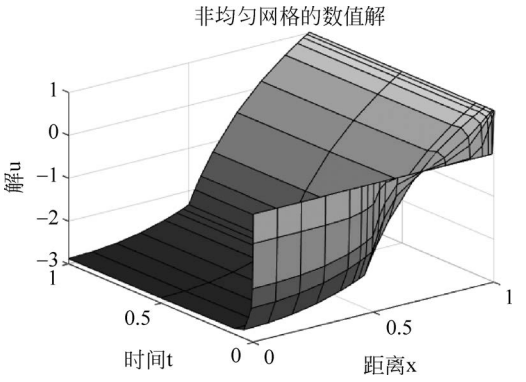

图 3-15　解 u 的曲面图

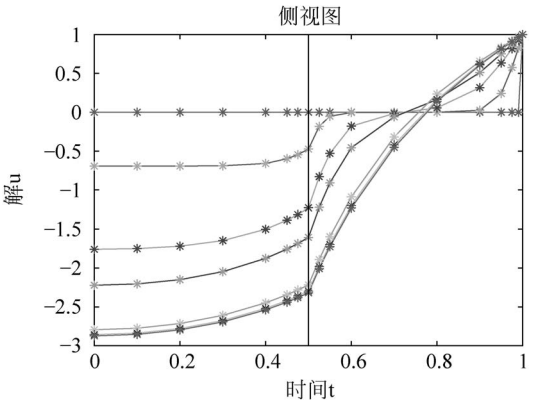

图 3-16　等高线侧视图

3. 求解 PDE 并计算偏导数

本节实例说明如何求解一个晶体管偏微分方程(PDE)，并使用结果获得偏导数，这是求解更大型问题的一部分。

【例 3-14】 以如下 PDE 为例：

$$\frac{\partial u}{\partial t}=D\frac{\partial^2 u}{\partial x^2}-\frac{D_\eta}{L}\frac{\partial u}{\partial x}$$

方程中的 $u(x,t)$是描述 PNP 晶体管基极中过剩电荷载流子(或空穴)浓度的函数。D 和 η 是物理常量。该公式在区间 $0\leqslant x\leqslant L$ 上对于时间 $t\geqslant 0$ 成立。

初始条件包括常量 K，由下式给出：

$$u(x,0)=\frac{KL}{D}\left(\frac{1-\mathrm{e}^{-\eta(1-x/L)}}{\eta}\right)$$

该问题具有由下式给出的边界条件。

$$u(0,t)=u(L,t)=0$$

对于固定 x，方程 $u(x,t)$的解将过剩电荷的坍塌描述为 $t\to\infty$。这种坍塌产生一种电流，称为发射极放电电流，它还有另一种常量 I_p：

$$I(t)=\left[\frac{I_pD}{K}\frac{\partial}{\partial x}u(x,t)\right]_{x=0}$$

由于在 $t=0$ 和 $t>0$ 时，$x=0$ 处的边界值不一致，该公式对 $t>0$ 有效。由于 PDE 对 $u(x,t)$有闭型级数解，可以通过解析方式和数值方式计算发射极放电电流，并对结果进行比较。

(1) 定义物理常量。

要跟踪物理常量，请创建一个结构体数组，其中每个常量都有一个对应的字段。当稍后为方程、初始条件和边界条件定义函数时，可以将此结构体作为额外的参数传入，以便函数可以访问常量。

```
>> C.L = 1;
C.D = 0.1;
C.eta = 10;
C.K = 1;
C.Ip = 1;
```

(2) 编写方程。

在编写方程代码前，需要确保它的形式符合 pdepe 求解器的要求：

$$c\left(x,t,u,\frac{\partial u}{\partial x}\right)\frac{\partial u}{\partial t}=x^{-m}\frac{\partial}{\partial x}\left(x^m f\left(x,t,u,\frac{\partial u}{\partial x}\right)\right)+s\left(x,t,u,\frac{\partial u}{\partial x}\right)$$

此形式的 PDE 为：

$$\frac{\partial u}{\partial t}=x^0\frac{\partial}{\partial x}\left(x^0D\frac{\partial u}{\partial x}\right)-\frac{D_\eta}{L}\frac{\partial u}{\partial x}$$

因此，方程中的系数的值为：

- $m=0$(没有角对称性的笛卡儿坐标)
- $c\left(x,t,u,\frac{\partial u}{\partial x}\right)=1$
- $f\left(x,t,u,\frac{\partial u}{\partial x}\right)=D\frac{\partial u}{\partial x}$
- $s\left(x,t,u,\frac{\partial u}{\partial x}\right)=-\frac{D_\eta}{L}\frac{\partial u}{\partial x}$

根据需要，建立函数 transistorPDE 以编写方程代码，函数代码为：

```
function [c,f,s] = transistorPDE(x,t,u,dudx,C)
% x 是独立的空间变量
% t 是独立的时间变量
% u 是关于 x 和 t 微分的因变量
% dudx 是偏空间导数∂u/∂x
% C 是包含物理常量的额外输入
% 输出 c、f 和 s 对应于 pdepe 所需的标准 PDE 形式中的系数
D = C.D;
eta = C.eta;
```

```
L = C.L;
c = 1;
f = D * dudx;
s = -(D * eta/L) * dudx;
end
```

(3) 初始条件。

编写一个返回初始条件的函数 transistorIC。初始条件应用在第一个时间值外,并为任意 x 提供 $u(x,t_0)$ 的值。

初始条件为:

$$u(x,0)=\frac{KL}{D}\left(\frac{1-e^{-\eta(1-x/L)}}{\eta}\right)$$

对应的函数代码为:

```
function u0 = transistorIC(x,C)
K = C.K;
L = C.L;
D = C.D;
eta = C.eta;
u0 = (K * L/D) * (1 - exp( - eta * (1 - x/L)))/eta;
end
```

(4) 编写边界条件。

编写一个计算边界条件 $u(0,t)=u(1,t)=0$ 的函数。对于在区间 $a\leqslant x\leqslant b$ 上的问题,边界条件将应用于所有 t 以及 $x=a$ 或 $x=b$ 的情形。求解器所需的边界条件的标准形式是:

$$p(x,t,u)+q(x,t)f\left(x,t,u,\frac{\partial u}{\partial x}\right)=0$$

以这种形式编写此问题的边界条件是:

① 对于 $x=0$,方程为 $u+0\cdot \mathrm{d}\dfrac{\partial u}{\partial x}=0$。系数为:

- $p_{\mathrm{L}}(x,t,u)=u$
- $q_{\mathrm{L}}(x,t)=0$

② 同样,对于 $x=1$,方程为 $u+0\cdot \mathrm{d}\dfrac{\partial u}{\partial x}=0$。系数为:

- $p_{\mathrm{R}}(x,t,u)=u$
- $q_{\mathrm{R}}(x,t)=0$

边界函数 transistorBC 的代码为:

```
function [pl,ql,pr,qr] = transistorBC(xl,ul,xr,ur,t)
% 对于左边界,输入 xl 和 ul 对应于 x 和 u
% 对于右边界,输入 xr 和 ur 对应于 x 和 u
% t 是独立的时间变量
% 对于左边界,输出 pl 和 ql 对应于 pL(x,t,u)和 qL(x,t)(对于此问题,x = 0)
% 对于右边界,输出 pr 和 qr 对应于 pR(x,t,u)和 qR(x,t)(对于此问题,x = 1)
pl = ul;
ql = 0;
pr = ur;
qr = 0;
end
```

(5) 选择解网格。

解网格定义 x 和 t 的值,求解器基于它们来计算解。由于此问题的解变化很快,可使用一

个相对精细的网格，其中包含 50 个位于 $0 \leqslant x \leqslant L$ 区间中的空间点和 50 个位于 $0 \leqslant t \leqslant 1$ 区间中的时间点。

```
>> x = linspace(0,C.L,50);
t = linspace(0,1,50);
```

(6) 求解方程。

最后，使用对称性值 m、PDE 方程、初始条件、边界条件以及 x 和 t 的网格来求解方程。由于 pdepe 函数需要使用 PDE 方程的四个输入作为初始条件，因此需要创建函数句柄，将由物理常量组成的结构体作为额外输入来传入。

```
>> m = 0;
eqn = @(x,t,u,dudx) transistorPDE(x,t,u,dudx,C);
ic = @(x) transistorIC(x,C);
sol = pdepe(m,eqn,ic,@transistorBC,x,t);
```

pdepe 以三维数组 sol 形式返回解，其中，sol(i,j,k)是在 $t(i)$ 和 $x(j)$ 处计算的解 u_k 的第 k 个分量的逼近值。对于此问题，u 只有一个分量，但通常可以使用命令 u = sol(:,:,k)提取第 k 个解分量。

```
>> u = sol(:,:,1);
```

(7) 对解进行绘图。

创建在 x 和 t 的所选网格点上绘制的解 u 的曲面图，如图 3-17 所示。

```
>> surf(x,t,u)
title('数值解(50 个网格点)')
xlabel('距离 x')
ylabel('时间 t')
zlabel('解 u(x,t)')
```

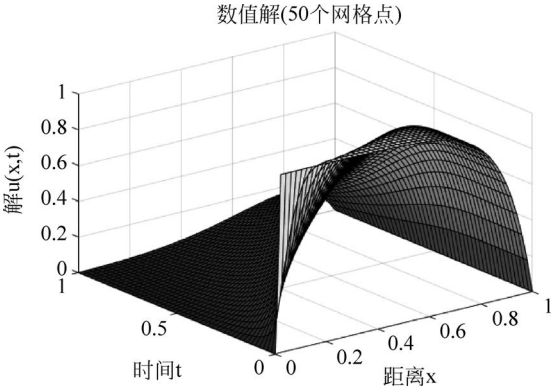

图 3-17　数值解曲面图

下面，只需绘制 x 和 u 即可获得曲面图中等高线的侧视图，如图 3-18 所示。

```
>> plot(x,u)
xlabel('距离 x')
ylabel('解 u(x,t)')
title('侧视图')
```

(8) 计算发射极放电电流。

使用 $u(x,t)$ 的级数解，发射极放电电流可以表示为无穷级数：

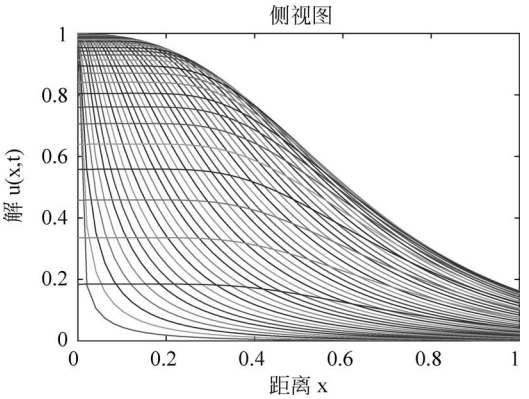

图 3-18　侧视图

$$I(t)=2\pi^2 I_p\left(\frac{1-\mathrm{e}^{-\eta}}{\eta}\right)\sum_{n=1}^{\infty}\frac{n^2}{n^2\pi^2+\eta^2/4}\mathrm{e}^{-\frac{dt}{L^2}(n^2\pi^2+\eta^2/4)}$$

编写一个函数，以使用级数中的 40 个项计算 $I(t)$的解析解。唯一的变量是时间，但要将常量结构体指定为函数的另一个输入。

```
function It = serex3(t,C)          % 用级数展开近似 I(t)
Ip = C.Ip;
eta = C.eta;
D = C.D;
L = C.L;
It = 0;
for n = 1:40                       % 40 个项
  m = (n * pi)^2 + 0.25 * eta^2;
  It = It + ((n * pi)^2 / m) * exp( - (D/L^2) * m * t);
end
It = 2 * Ip * ((1 - exp( - eta))/eta) * It;
end
```

使用 pdepe 计算 $u(x,t)$的数值解，还可以通过以下方程计算在 $x=0$ 处的 $I(t)$的数值逼近：

$$I(t)=\left[\frac{I_p D}{K}\frac{\partial}{\partial x}u(x,t)\right]_{x=0}$$

计算 $I(t)$的解析解和数值解，并对结果绘图，如图 3-19 所示。使用 pdeval 计算$\frac{\partial u}{\partial x}$在 $x=0$ 处的值。

```
>> nt = length(t);
I = zeros(1,nt);
seriesI = zeros(1,nt);
iok = 2:nt;
for j = iok
    % 在时间 t(j),计算 x = 0 时的 du/dx
    [~,I(j)] = pdeval(m,x,u(j,:),0);
    seriesI(j) = serex3(t(j),C);
end
% 数值解的形式为 I(t) = (I_p * D/K) * du(0,t)/dx
I = (C.Ip * C.D/C.K) * I;
plot(t(iok),I(iok),'o',t(iok),seriesI(iok))
legend('From PDEPE + PDEVAL','From series')
```

```
title('数值解 I(t)')
xlabel('时间 t')
```

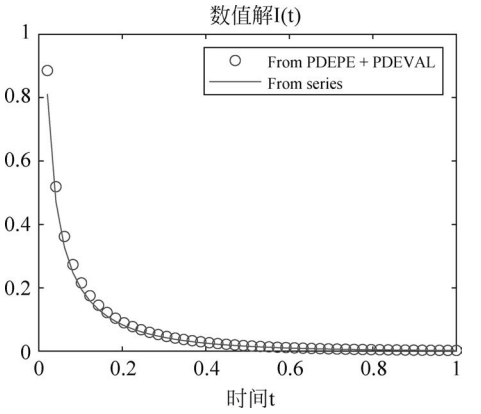

图 3-19　$I(t)$的解析解与数值解

从图 3-19 可看出，结果相当吻合。通过使用更精细的解网格，可以进一步改进 pdepe 得出的数值结果。

3.3　傅里叶变换与滤波

变换和滤波器是用于处理和分析离散数据的工具，常用在信号处理应用和计算数学中。当数据表示为时间或空间的函数时，傅里叶变换会将数据分解为频率分量。fft 函数使用快速傅里叶变换算法，相对于其他算法直接实现，这种方式能够减少计算成本。

3.3.1　傅里叶变换

傅里叶变换是将按时间或空间采样的信号与按频率采样的相同信号进行关联的数学公式。在信号处理中，傅里叶变换可以揭示信号的重要特征(即其频率分量)。

对于包含 n 个均匀采样点的向量 x，其傅里叶变换定义为：

$$y_{k+1}=\sum_{j=0}^{n-1}\omega^{jk}x_j+1$$

$\omega=e^{-2\pi i/n}$ 是 n 个复单位根之一，其中，i 是虚数单位。对于 x 和 y，索引 j 和 k 的范围为 0～$n-1$。在 MATLAB 中使用快速傅里叶变换算法 fft 函数来计算数据的傅里叶变换。

1. 含噪信号

在科学应用中，信号经常遭到随机噪声破坏，掩盖其频率分量。傅里叶变换可以清除随机噪声并显现频率。

【例 3-15】 以正弦信号 x 为例，该信号是时间 t 的函数，频率分量为 15Hz 和 20Hz。使用在 10s 周期内以 $\frac{1}{50}$s 为增量进行采样的时间向量。计算并绘制以零频率为中心的含噪信号的功率谱。

```
>> x = sin(2 * pi * 15 * t) + sin(2 * pi * 20 * t);
y = fft(x);
n = length(x);
```

```
fshift = (-n/2:n/2-1)*(50/n);
yshift = fftshift(y);
```

在原始信号 x 中注入高斯噪声，创建一个新信号 xnoise。

```
>> xnoise = x + 2.5*gallery('normaldata',size(t),4);
```

频率函数形式的信号功率是信号处理中的一种常用度量。功率是信号的傅里叶变换按频率样本数进行归一化后的平方幅值，如图 3-20 所示。

```
ynoise = fft(xnoise);
ynoiseshift = fftshift(ynoise);
power = abs(ynoiseshift).^2/n;
plot(fshift,power)
title('功率')
```

2. 计算效率

直接使用傅里叶变换公式分别计算 y 的 n 个元素需要 n^2 数量级的浮点运算。使用快速傅里叶变换算法，则只需要 $n\log n$ 数量级的运算。在处理包含成百上千万个数据点的数据时，这一计算效率会带来很大的优势。在 n 为 2 的幂时，许多专门的快速傅里叶变换实现可进一步提高效率。

【例 3-16】 载入包含太平洋蓝鲸鸣声的文件 bluewhale.au，并对其中一部分数据进行格式化。可使用命令 sound(x,fs)来收听完整的音频文件。

```
>> whaleFile = 'bluewhale.au';
[x,fs] = audioread(whaleFile);
whaleMoan = x(2.45e4:3.10e4);
t = 10*(0:1/fs:(length(whaleMoan)-1)/fs);
plot(t,whaleMoan)                                    % 效果如图 3-21 所示
xlabel('时间(seconds)')
ylabel('幅度')
xlim([0 t(end)])
```

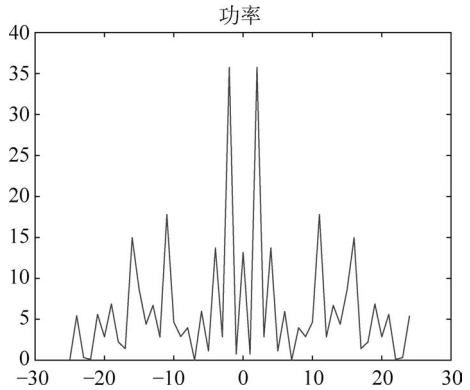

图 3-20 功率图

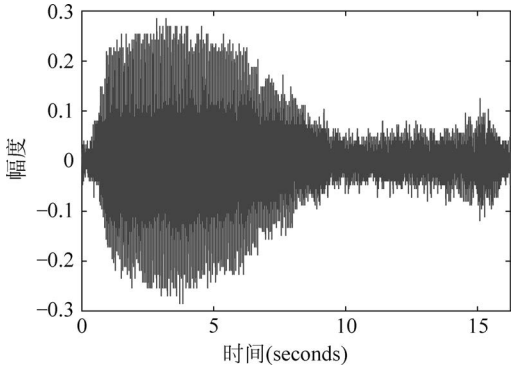

图 3-21 信号图

指定新的信号长度，该长度是大于原始长度的最邻近的 2 的幂。然后使用 fft 和新的信号长度计算傅里叶变换。fft 会自动用零填充数据，以增加样本大小。此填充操作可以大幅提高变换计算的速度，对于具有较大质因数的样本大小更是如此。

```
>> m = length(whaleMoan);
n = pow2(nextpow2(m));
y = fft(whaleMoan,n);
```

绘制信号的功率谱，如图 3-22 所示。绘图指示，鸣音包含约 17Hz 的基本频率和一系列谐波（其中强调了第二个谐波）。

```
>> f = (0:n-1)*(fs/n)/10;        % 频率向量
power = abs(y).^2/n;             % 功率谱
plot(f(1:floor(n/2)),power(1:floor(n/2)))
xlabel('频率')
ylabel('功率')
```

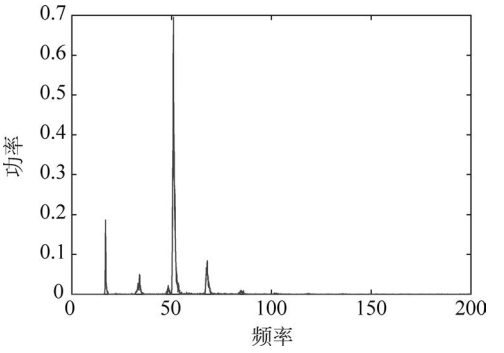

图 3-22　功率谱

3.3.2　二维傅里叶变换

fft 函数将二维数据变换为频率空间。例如，可以变换二维光学掩模以揭示其衍射模式。

1. 二维傅里叶定义

以下公式定义 $m \times n$ 矩阵 X 的离散傅里叶变换 Y。

$$Y_{P+1,q+1} = \sum_{j=0}^{m-1} \sum_{k=0}^{n-1} \omega_m^{jp} \omega_n^{kq} X_{j+1,k+1}$$

ω_m 和 ω_n 是以下方程所定义的复单位根。

$$\omega_m = \mathrm{e}^{-2\pi \mathrm{i}/m}$$

$$\omega_n = \mathrm{e}^{-2\pi \mathrm{i}/n}$$

i 是虚数单位，p 和 j 是值范围从 0 到 $m-1$ 的索引，q 和 k 是值范围从 0 到 $n-1$ 的索引。在此公式中，X 和 Y 的索引平移 1 位，以反映 MATLAB 中的矩阵索引。

计算 X 的二维傅里叶变换等同于首先计算 X 每列的一维变换，然后获取每行结果的一维变换。换言之，函数 fft2(X)等同于 Y=fft(fft(X).').'。

2. 二维衍射模式

在光学领域，傅里叶变换可用于描述平面波入射到带有小孔的光学掩模上所产生的衍射模式。

【例 3-17】 对光学掩模使用 fft2 函数来计算其衍射模式。

```
% 创建用于定义带有小圆孔的光学掩模的逻辑数组。
>> n = 2^10;                     % 掩模大小
M = zeros(n);
I = 1:n;
x = I-n/2;                       % 掩模 x 坐标
y = n/2-I;                       % 掩模 y 坐标
[X,Y] = meshgrid(x,y);           % 创建二维掩模网格
```

```
R = 10;                          %孔径半径
A = (X.^2 + Y.^2 <= R^2);        %半径为R的圆孔
M(A) = 1;                        %将孔径内的掩码元素设置为1
imagesc(M)                       %绘制掩模,如图3-23所示
axis image
```

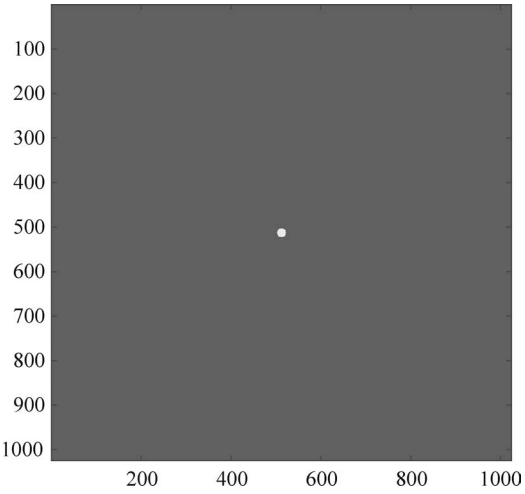

图 3-23　掩模图

使用 fft2 计算掩模的二维傅里叶变换，并使用 fftshift 函数重新排列输出，从而使零频率分量位于中央。绘制生成的衍射模式频率，如图 3-24 所示。蓝色指示较小的幅值，黄色指示较大的幅值。

```
>> DP = fftshift(fft2(M));
imagesc(abs(DP))
axis image
```

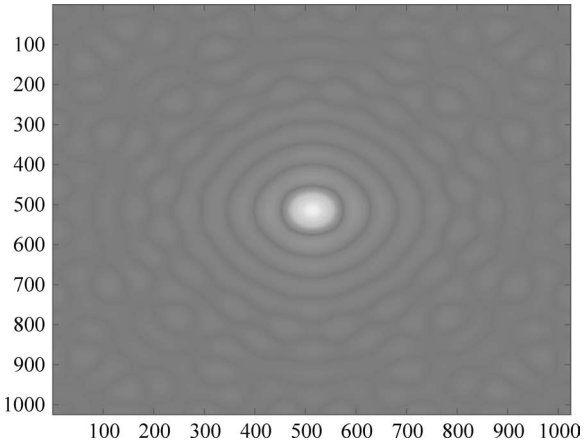

图 3-24　衍射模式频率图

为增强小幅值区域的细节，需绘制衍射模式的二维对数(如图 3-25 所示)。极小的幅值会受数值舍入误差影响，而矩形网格则会导致径向非对称性。

```
>> imagesc(abs(log2(DP)))
axis image
```

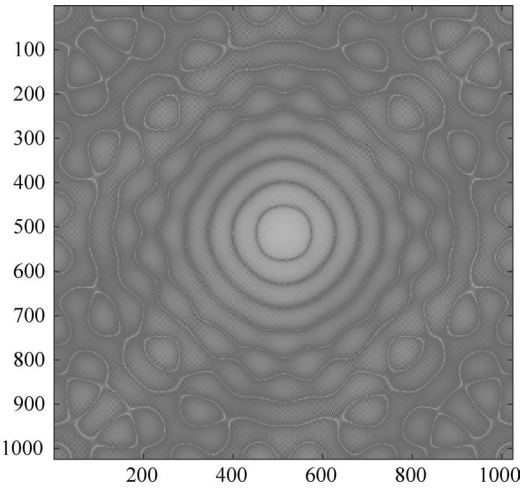

图 3-25　衍射模式的二维对数

3.3.3　滤波数据

滤波器是一种数据处理技术，可滤掉数据中的高频波动部分使之平滑或从数据中删除特定频率的周期趋势。在 MATLAB 中，filter 函数会根据以下差分方程对数据 $\boldsymbol{x}$ 的向量进行滤波，该差分方程描述一个抽头延迟线滤波器。

$$\boldsymbol{a}(1)\boldsymbol{y}(n)=\boldsymbol{b}(1)\boldsymbol{x}(n)+\boldsymbol{b}(2)\boldsymbol{x}(n-1)+\cdots+\boldsymbol{b}(N_b)\boldsymbol{x}(n-N_b+1)-$$
$$\boldsymbol{a}(2)\boldsymbol{y}(n-1)-\cdots-\boldsymbol{a}(N_a)\boldsymbol{y}(n-N_a+1)$$

在方程中，$\boldsymbol{a}$ 和 $\boldsymbol{b}$ 是滤波器系数的向量，N_a 是反馈滤波器阶数，N_b 是前馈滤波器阶数。n 是 $\boldsymbol{x}$ 的当前元素的索引。输出 $\boldsymbol{y}(n)$ 是 $\boldsymbol{x}$ 和 $\boldsymbol{y}$ 的当前元素和前面元素的线性组合。

filter 函数使用指定的系数向量 $\boldsymbol{a}$ 和 $\boldsymbol{b}$ 对输入数据 $\boldsymbol{x}$ 进行滤波。它是实现移动平均值滤波器的一种方式，是一种常见的数据平滑技术。

【例 3-18】 以下差分方程描述一个滤波器，它对关于当前小时和前三个小时的数据的时间相关数据求平均值。

$$\boldsymbol{y}(n)=\frac{1}{4}\boldsymbol{x}(n)+\frac{1}{4}\boldsymbol{x}(n-1)+\frac{1}{4}\boldsymbol{x}(n-2)+\frac{1}{4}\boldsymbol{x}(n-3)$$

导入描述交通流量随时间变化的数据，并将第一列车辆计数分配给向量 $\boldsymbol{x}$。

```
>> load count.dat
x = count(:,1);
% 创建滤波器系数向量
a = 1;
b = [1/4 1/4 1/4 1/4];
% 计算数据的 4 小时移动平均值，同时绘制原始数据和滤波后的数据，如图 3-26 所示
y = filter(b,a,x);
t = 1:length(x);
plot(t,x,'--',t,y,'-')
legend('原始数据','滤波数据')
```

此外，还可以修改数据振幅。在数字信号处理中，滤波器通常由传递函数表示。以下差分方程的 Z 变换：

$$\boldsymbol{a}(1)\boldsymbol{y}(n)=\boldsymbol{b}(1)\boldsymbol{x}(n)+\boldsymbol{b}(2)\boldsymbol{x}(n-1)+\cdots+\boldsymbol{b}(N_b)\boldsymbol{x}(n-N_b+1)-$$
$$\boldsymbol{a}(2)\boldsymbol{y}(n-1)-\cdots-\boldsymbol{a}(N_a)\boldsymbol{y}(n-N_a+1)$$

对应的传递函数为

$$\boldsymbol{Y}(z)=\boldsymbol{H}(z^{-1})\boldsymbol{X}(z)=\frac{\boldsymbol{b}(1)+\boldsymbol{b}(2)z^{-1}+\cdots+\boldsymbol{b}(N_b)z^{-N_b+1}}{\boldsymbol{a}(1)+\boldsymbol{a}(2)z^{-1}+\cdots+\boldsymbol{a}(N_a)z^{-N_a+1}}\boldsymbol{X}(z)$$

实例中使用传递函数

$$\boldsymbol{H}(z^{-1})=\frac{\boldsymbol{b}(z^{-1})}{\boldsymbol{a}(z^{-1})}=\frac{2+3z^{-1}}{1+0.2z^{-1}}$$

【例 3-19】 通过应用传递函数来修改数据向量的振幅。

```
>> % 加载数据并将第一列分配到向量 x
load count.dat
x = count(:,1);
% 根据传递函数 H(Z^-1)创建滤波器系数向量
a = [1 0.2];
b = [2 3];
% 计算滤波后的数据,同时绘制原始数据和滤波后的数据,如图 3-27 所示。此滤波器主要修改原始
% 数据的振幅
y = filter(b,a,x);
t = 1:length(x);
plot(t,x,'--',t,y,'-')
legend('原始数据','滤波数据')
```

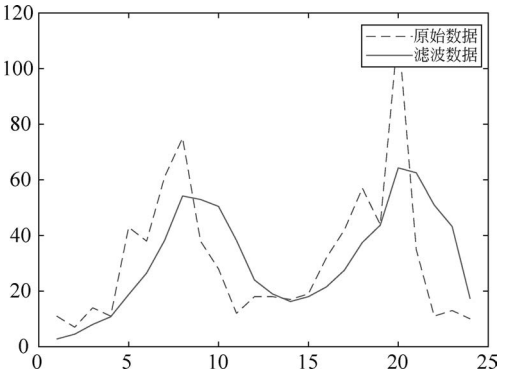

图 3-26 原始数据和滤波数据

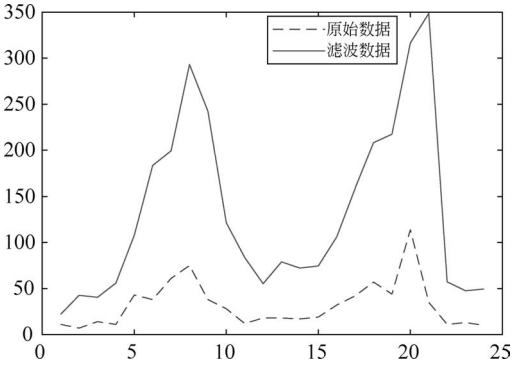

图 3-27 修改后的原始数据与滤波数据的振幅