第 5 章

数学：MATLAB 数学计算

MATLAB 的真正力量蕴藏在其对数学运算的精湛处理之中。无论是微积分、线性代数，还是更为复杂的数学分支，MATLAB 都呈现了卓越的解决方案。精通 MATLAB 的工具和功能，读者的高等数学技能将提升至新的高度。

5.1 初等数学

在初等数学的学习中，我们经常遇到离散数学和多项式问题，这些问题往往需要借助像 MATLAB 这样的计算软件来解决。掌握 MATLAB，就意味着读者能以更高效、更精确的方式处理这些数学挑战。

5.1.1 离散数学

虽然离散数学并非严格意义上的初等数学范畴，但我们这里提到的离散数学内容，主要涉及质数、约数、倍数、有理数等计算问题，这些都是高等数学和计算机科学的基础知识。掌握这些基础概念，对深入理解更高级的数学和计算机科学领域至关重要。离散数学例程如图 5-1 所示。

EX 5-1 离散数学

```
% 分解质因数
primeFactors = factor(20)
% 最大公约数
greatestCommonDivisor = gcd(20, 15)
% 最小公倍数
leastCommonMultiple = lcm(20, 15)
% 确定哪些数组元素为质数
isPrime = isprime([2 3 0 6 10])
% 小于或等于输入值的质数
PrimeNumbers = primes(20)
% 所有可能的排列
permutations = perms([1 2 3])
% 输入的阶乘
inputFactorial = factorial(5)
% 有理分式近似值
rationalFraction = rat(pi,1e-7)
```

```
primeFactors = 1×3
     2     2     5

greatestCommonDivisor = 5
leastCommonMultiple = 60
isPrime = 1×5 logical 数组
   1   1   0   0   0

PrimeNumbers = 1×8
     2     3     5     7    11    13    17    19

permutations = 6×3
     3     2     1
     3     1     2
     2     3     1
     2     1     3
     1     3     2
     1     2     3


inputFactorial = 120

rationalFraction = '3 + 1/(7 + 1/(16 + 1/(-294)))'
```

图 5-1　离散数学例程

说明：

(1) gcd()函数和 lcm()函数分别为求最大公约数(Greatest Common Divisor)和求最小公倍数(Least Common Multiple)的函数。

(2) perms()函数用于得到一个向量中所有元素的所有可能的排列，其实是一个应用场景较广的函数，比如常被应用于穷举算法以及一些集合论计算中。

(3) rat()函数用于取得输入量的有理分式，其参数为计算容差，根据不同的容差得到不同精度的有理分式逼近。

5.1.2　多项式

在 MATLAB 中，多项式的表达采用向量形式，向量中的每个元素代表多项式相应次幂的系数，其中，向量的最后一个元素对应常数项。MATLAB 内置了一些处理多项式的常用函数，例如求值函数 polyval()、求多项式根的函数 roots()、由根构建多项式的函数 poly()，以及将多项式转换为符号表达式的函数 poly2sym()。多项式例程如图 5-2 所示，读者还可以利用 MATLAB 绘制多项式的图形，来丰富多项式的表达形式。

EX 5-2 多项式

$p(x) = x^4 - 10x^3 + 35x^2 - 50x + 24$

```
% 创建上述多项式
p = [1 -10 35 -50 24]
% 计算x为2时多项式的值
polyValue = polyval(p,2.5)
% 同时计算x为从1到5时的值
polyValue2 = polyval(p,1:5)
% 多项式作图
x=0.5:0.1:4.5;
plot(x,polyval(p,x)); yline(0);
% 求多项式的根
r = roots(p)
% 由根反求多项式
pCalc = poly(r)
% 显示多项式
poly2sym(p)
```

```
p = 1×5
     1   -10    35   -50    24

polyValue = 0.5625
polyValue2 = 1×5
     0     0     0     0    24

r = 4×1
    4.0000
    3.0000
    2.0000
    1.0000

pCalc = 1×5
    1.0000  -10.0000   35.0000  -50.0000   24.0000

ans = x^4 - 10 x^3 + 35 x^2 - 50 x + 24
```

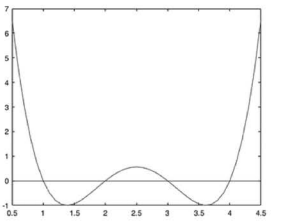


图 5-2　多项式例程

说明：

(1) 在 MATLAB 中表示多项式的向量里，即使某些次幂的系数为 0，也需要用 0 值来正确占位，以保持多项式结构的完整性。

(2) 绘制多项式图形的核心方法实际上是计算多项式在各个点上的值，并据此进行作图。

(3) 多项式显示函数 poly2sym()的本质是将多项式向量转换为符号矩阵。

5.2　线性代数

线性代数，被称为“第二代数学模型”，它是构筑了现代科学的基石之一，并以其矩阵理论为核心，成为了高等数学思维中至关重要的一环。线性代数是研究“线性问题”的代数理

论，线性是线性代数的第一特性。我们可以这样初步地理解线性——“线性就是简单性”，就是“可以任意叠加而不会产生意外”。线性代数无论在数学、自然科学，还是高等数学学习里都具有中心地位。线性代数还是高等数学思维的敲门砖，所谓高等数学，就是将事物抽象到本质结构，再通过结构来反向推知事物规律。因此，线性代数是最值得所有人深度体会与思考的一门数学。我们坚信，线性代数的思想是高等数学中最典型的范例。因此，学习线性代数无疑是进入高等数学世界的最佳途径。

5.2.1 矩阵基础运算

在线性代数的学习中，矩阵的基本运算诸如提取对角线、计算迹（即对角线元素之和）、求秩，以及计算行列式等，是不可或缺的操作。在 MATLAB 中，这些操作的函数可以同时应用于符号矩阵和数值矩阵，如图 5-3 所示。

EX 5-3 矩阵基础运算

```
% 准备符号阵和数值阵
syms a b c d
x = [a b; c d]
y = [1 2; 3 4]
% 对角线向量
diagX = diag(x)
diagY = diag(y)
% 由对角线向量构成对角阵
diagMatrixX = diag(diag(x))
diagMatrixY = diag(diag(y))
% 矩阵的迹
traceX = trace(x)
traceY = trace(y)
% 矩阵的秩
rankX = rank(x)
rankY = rank(y)
% 矩阵的行列式
detX = det(x)
detY = det(y)
```

```
x =
[a, b]
[c, d]
y = 2×2
     1     2
     3     4
diagX =
[a]
[d]
diagY = 2×1
     1
     4
diagMatrixX =
[a, 0]
[0, d]
diagMatrixY = 2×2
     1     0
     0     4
traceX = a + d
traceY = 5
rankX = 2
rankY = 2
detX = a*d - b*c
detY = -2
```

图 5-3 矩阵基本运算例程

说明：

（1）diag()函数对于不同的输入会有不同的输出，如果输入的是矩阵，则输出对角线向量；如果输入的是一个向量，则会依照该向量输出一个对角矩阵。

（2）rank()函数对于符号阵的计算，是按照最大秩获得结果的，即把所有符号都当作非零值来计算，实际意义微弱，因此一般并不常用。数值计算必然是存在精度的，精度大小与算法关系密切，比如 MATLAB 中求秩的算法是基于矩阵的奇异值分解的，在 rank()函数里还可以后跟一个给定误差参数，即为机器精度。

5.2.2　矩阵分解

MATLAB为用户提供了一系列全面的线性代数函数，涵盖了几乎所有的矩阵分解操作，这些函数同样适用于符号矩阵，如图5-4所示。

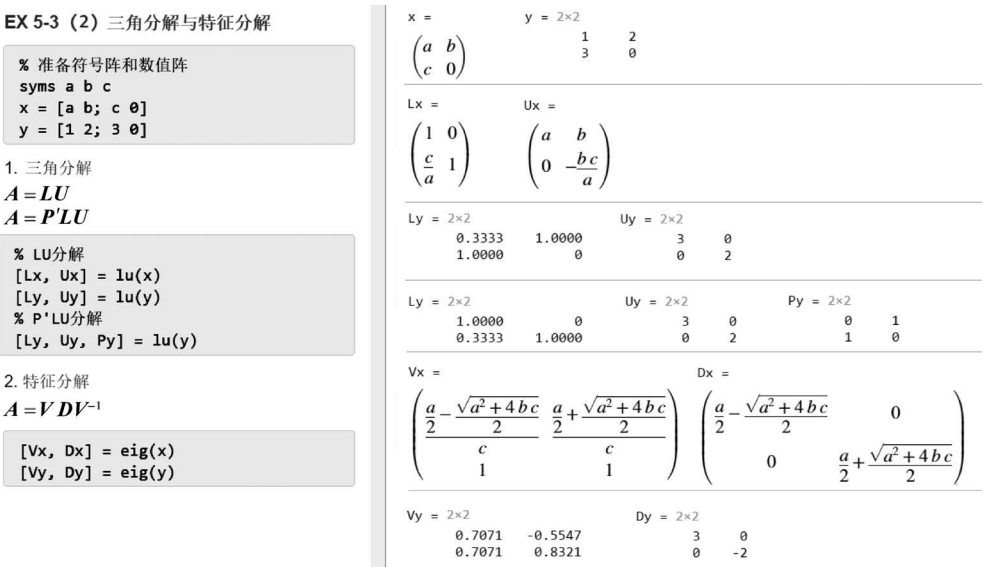


图5-4　三角分解与特征分解例程

说明：

（1）在MATLAB中，对于符号阵的LU分解，可以得出与线性代数教材中一致的结果；然而，由于考虑到算法稳定性的问题，在数值计算LU分解中，得到的所谓下三角阵（$\boldsymbol{L}$）与定义中并不一致，而是需要进行PLU分解才能得到真正的单位下三角阵，这里需要多得到一个置换矩阵$\boldsymbol{P}$，即$\boldsymbol{A}=\boldsymbol{P}'\boldsymbol{L}\boldsymbol{U}$。

（2）特征分解在矩阵分解中至关重要，eig()函数提供分解算法得到特征对角阵$\boldsymbol{D}$和特征向量矩阵$\boldsymbol{V}$，实现满足$\boldsymbol{A}\boldsymbol{V}=\boldsymbol{V}\boldsymbol{D}$的特征阵分解计算。

5.2.3　线性方程及矩阵的逆

线性方程组是许多问题的基础抽象模型，例如，坐标系变换、运动分析、解析几何中直线交点问题等，它们的应用范围极其广泛。MATLAB，这款根植于线性代数学科的科学计算软件，在解决线性方程问题上展现了其领先的技术实力。它以简洁的符号封装了适用于各种特殊情况的算法，并且在求解速度上做到了极致的优化，如图5-5所示。

说明：

（1）对于$\boldsymbol{A}\boldsymbol{x}=\boldsymbol{b}$形式的线性方程求解非常简洁，解即为“$\boldsymbol{A}\backslash\boldsymbol{b}$”，注意斜线的方向，记忆方法是，斜线方向指示$\boldsymbol{A}$为分母项，是从等号左侧移动到右侧的分母项。

EX 5-4 解线性方程

$$\begin{bmatrix}3 & 4\\ 2 & 5\end{bmatrix}\begin{bmatrix}x_1\\ x_2\end{bmatrix}=\begin{bmatrix}11\\ 12\end{bmatrix}$$

```
% 解方程 Ax=b
A = [3 4; 2 5];
b=[11 12]';
% 求解方法一
solve1 = A\b
% 求解方法二
inverseA = inv(A)
solve2 = inv(A)*b
```

$$\begin{bmatrix}a & b\\ c & d\end{bmatrix}\begin{bmatrix}x_1\\ x_2\end{bmatrix}=\begin{bmatrix}e\\ f\end{bmatrix}$$

```
% 解方程 Ax=b
syms a b c d e f
A = [a b; c d];
b=[e f]';
solve1 = A\b
inverseA = inv(A)
solve2 = inv(A)*b
% 化简符号表达式
simpSlolve2 = simplify(solve2)
```

```
solve1 = 2×1
       1
       2

inverseA = 2×2
    0.7143   -0.5714
   -0.2857    0.4286

solve2 = 2×1
    1.0000
    2.0000
```

solve1 =

$$\begin{pmatrix}-\frac{b\bar{f}-d\bar{e}}{ad-bc}\\ \frac{a\bar{f}-c\bar{e}}{ad-bc}\end{pmatrix}$$

inverseA =

$$\begin{pmatrix}\frac{d}{ad-bc} & -\frac{b}{ad-bc}\\ -\frac{c}{ad-bc} & \frac{a}{ad-bc}\end{pmatrix}$$

solve2 =

$$\begin{pmatrix}\frac{d\bar{e}}{ad-bc}-\frac{b\bar{f}}{ad-bc}\\ \frac{a\bar{f}}{ad-bc}-\frac{c\bar{e}}{ad-bc}\end{pmatrix}$$

simpSlolve2 =

$$\begin{pmatrix}-\frac{b\bar{f}-d\bar{e}}{ad-bc}\\ \frac{a\bar{f}-c\bar{e}}{ad-bc}\end{pmatrix}$$

图 5-5　解线性方程例程

（2）inv()函数用于求矩阵的逆(Inverse)，使用前述“斜线求解法”与“矩阵的逆求解法”在求解原理上大相径庭，后者无论从精度、速度、稳定性上都略逊一筹，从图 5-5 中的结果也可看出，前者求解精度足够高，可以显示为整数的形式(当然实际还是 double 类)，而后者显示出了 short 形式，说明求解精度有限，可以使用“format long”命令将显示格式改为长型，则更为明显。

（3）利用符号形式进行求解往往会得到不错的效果，并且可以避免计算精度的问题，simplify()函数是用于对符号表达式进行自动整理与化简的函数，可以看到化简后的解 simpSlolve2 与 solve1 完全一致。

5.3　微积分

微积分是现代科学的核心数学基础之一，对于经典物理世界的建模几乎完全依靠在微积分的港湾里，这其中的底层逻辑在于，人类对于经典物理世界的理解就是“连续的”且“有因果关系的”，这两个特点正与微积分的特点相符，因而，微积分是对物理世界建模的第一工具。

微积分的英文 Calculus 其实就是“计算方法”之意，在没有计算机的时代，科学家与工程师把微积分看作很有效的计算工具，因此也诞生了许多计算微积分的手算算法，当然这些手算算法也不是万能的，而计算机软件工具促使这一切向前飞跃，MATLAB 强大的符号计算引擎，已经可以瞬间得到许多人工无法得到的解析解，让微积分的计算从此不再是科学家与工程师需要考虑的难点。

5.3.1　极限的艺术

微积分的基础是连续与极限，在 MATLAB 中有 limit()函数可以用于求极限，且兼容符号计算与数值计算，如图 5-6 所示。

EX 5-5 极限

$f=\left(1+\frac{a}{x}\right)^x$，求：$\lim_{x\to 1} f(x)$，$\lim_{x\to 0^+} f(x)$，$\lim_{x\to \infty^-} f(x)$

```
% 解方程 Ax=b
syms x a
f = (1+a/x)^x;
limF1 = limit(f,x,1)
limF0 = limit(f,x,0, 'right')
limFinf = limit(f,x,-inf)
% 作a=1时的图像
fplot(@(x) (1+1/x)^x, [-50 50]);
yline(exp(1)); % 渐近线
```

limF1 = $a+1$

limF0 = 1

limFinf = e^a

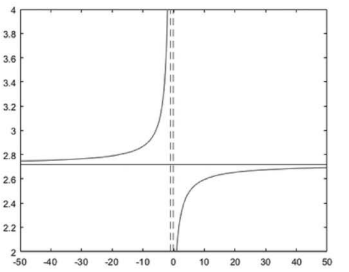


图 5-6　极限例程

说明：

（1）极限函数 limit()的第三个参数为极限点的位置，可以是正无穷(inf)或负无穷(－inf)，第四个参数可以定义所求极限为右单边极限(right)还是左单边极限(left)。

（2）在本例中取 a 为 1 后使用 fplot()函数作图时，会提示警告函数处理数组输入时行为异常，原因是在 0 点附近的虚线范围内，函数值的绝对值过大，导致无法正常作图。

（3）对于多元函数的极限求解，仍然是使用 limit()函数，原理是先对一元求极限，再对另一元求极限，下面两种代码形式等价：

```
limit(limit(f,x,x0), y, y0) 和 limit(limit(f,y,y0), x, x0)
```

5.3.2　导数：原函数的“因”

“导数”是微积分的核心概念，“导”的含义从组词中隐约可见，比如“导致”“引导”“导向”等，《史记·孙子吴起列传》中讲“善战者，因其势而利导之”，导的含义其实是“引领方向改变”，所以导数大则原函数方向改变多，导数小则原函数方向改变少，导数为零则方向不变。因而可以说，“导数是原函数的原因”。

MATLAB 中求解导数使用 diff()函数，极为简洁实用，如图 5-7 所示，这里与导数的英文 Derivative 并不一致，原因是 diff()同时也是差分(Differences)函数，在 3.4 节的矩阵运算中提及过。

说明：

（1）diff()函数的第三个参数表示要求的是几阶导数，此项为空即为 1 阶导数。

（2）本例展示了单引号与双引号在文本中的显示方法，单引号文本使用双引号引用，而双引号的文本使用单引号引用，这样不会引发代码歧义。

EX 5-6 导数

$f=2x^4+x$，求：f' f''

```
syms x
f = 2*x^4+x
% 一阶导数
diffx1 = diff(f,x)
% 二阶导数
diffx2 = diff(f,x,2)
% 绘图
hold on; box on;
p(1) = fplot(f);
p(2) = fplot(diffx1,'--');
p(3) = fplot(diffx2,':');
[p.LineWidth] = deal(2.5);
l = legend('f', "f'", 'f"');
l.FontSize = 15;
```

f = $2x^4+x$

diffx1 = $8x^3+1$

diffx2 = $24x^2$

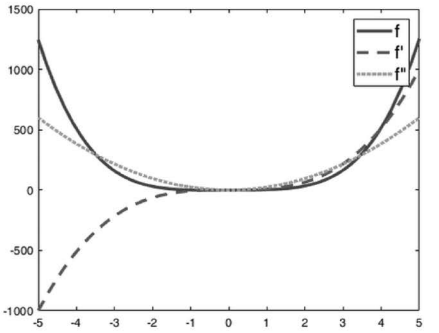


图 5-7　导数例程

（3）对于多元函数的偏导数，与前述极限的思路一致，代码形式如下：

```
diff(diff(f,x,m), y, n) 或 diff(diff(f,y,n), x, m)
```

5.3.3　积分：原函数的“果”

积分（Integral）与导数是相反的运算，导函数是原函数的原因，而积分是原函数的结果。积分包括不定积分与定积分，不定积分是求导函数的原函数，求得结果是由一个函数代表的一族函数，而定积分是求导函数在一个区间内的面积，求得结果是一个值，代表着导函数在该区间内引发的变化，如图 5-8 所示。

EX 5-7 积分

$f=ax^2$，求：$\int f\,\mathrm{d}(x)$ 与 $\int\left(\int f\,\mathrm{d}(x)\right)\mathrm{d}(x)$

```
syms x a
f = a*x^2
intf1 = int(f,x)
intf2 = int(int(f,x),x)
```

$f=ax^2$，求：$\int_0^1 f\,\mathrm{d}(x)$

```
intf = int(f,x,0,1)
```

f = ax^2

intf1 =

$\frac{ax^3}{3}$

intf2 =

$\frac{ax^4}{12}$

intf =

$\frac{a}{3}$

图 5-8　积分例程

说明：

（1）积分函数 int()对于不定积分与定积分通用，且并没有多次积分的输入参数，只能使用函数嵌套多层来实现。

（2）不定积分的结果函数省略了一个常数 C。

（3）对于不可积的函数，即使是 MATLAB 也无能为力。

（4）当定积分区间的一边是无穷值时，称为无穷积分，只需要将区间输入值设置为－inf 或 inf 即可。

5.3.4　泰勒展开：多项式仿真工具

泰勒展开(Taylor Expansion)是微积分体系提供的又一伟大工具，它可以将任意包裹着外壳的函数都展开为多项式函数，可以说泰勒展开是用于模拟任意函数的一个多项式仿真工具，与此同时，由于展开的结果中各项对应着导数次数，因此相当于将原函数的变化原因依照主次进行了分解，对于了解原函数的本质有拨云见日的功效。对函数 $f(x)$ 在 x_0 点的泰勒展开的公式为

$$f(x)=f(x_0)+f'(x_0)(x-x_0)+\frac{f''(x-x_0)^2}{2!}+\cdots+\frac{f^{(n)}(x_0)}{n!}(x-x_0)^n$$

等号右侧即为“泰勒多项式”，其最后一项为“多项式仿真的误差项”，是 $(x-x_0)^n$ 的高阶无穷小项，如图 5-9 所示。

EX 5-8 泰勒展开

展开：$f=\sin(x)$

```
syms x
f = sin(x)
f1 = taylor(f,x,0,'order',3)
f2 = taylor(f,x,0,'order',5)
f3 = taylor(f,x,0,'order',7)
% 作图
fplot(f,'LineWidth',2); hold on;
fplot(f1); fplot(f2); fplot(f3);
```

f = $\sin(x)$

f1 = x

f2 =

$$x-\frac{x^3}{6}$$

f3 =

$$\frac{x^5}{120}-\frac{x^3}{6}+x$$

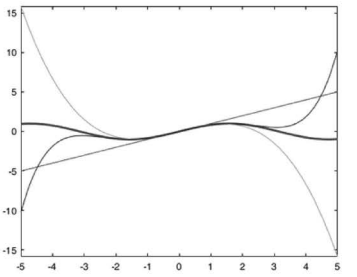


图 5-9　泰勒展开例程

说明：

(1) 泰勒展开函数 taylor()的截断阶数(order)项如果不予输入，则默认截断阶数为 6，该阶数为绝对阶数，从示例中可见。

(2) 如绘制的图形所示，阶数越多仿真结果越准确，离展开点越近仿真结果越准确，这也是为什么 $\sin(x)$ 在 x 接近 0 时可以用 x 来近似表示的原因了。

5.3.5　傅里叶展开：频域上的简谐波仿真

“周期”是一个显然而又隐秘的概念，周期性的变化称之为“波”，波是物理世界的基本组成，而所有的波都以“简谐波”(正余弦)为宗，也就是说所有的周期函数，都可以分解为简谐波的组合。傅里叶展开(Fourier Expansion)就是对周期性函数的简谐波分解，也可以称其为简谐波动仿真系统。5.3.4 节讲的泰勒展开相当于对函数在时域上的多项式仿真，而傅里叶展开则相当于对函数在频域上的简谐波仿真，两者均是解析物理世界的重要模型。

对于一个周期为 T 的函数 $f(x)$，在其一个周期范围 $[-L,L]$ 内，可以展开为如下级数形式：

$$f(x)=\frac{a_0}{2}+\sum_{n=1}^{\infty}\left(a_n\cos\frac{n\pi}{L}x+b_n\sin\frac{n\pi}{L}x\right)$$

其中，

$$\begin{cases} a_n = \dfrac{1}{L}\displaystyle\int_{-L}^{L} f(x)\cos\frac{n\pi x}{L}\mathrm{d}x, \quad n = 0,1,2,\cdots \\ b_n = \dfrac{1}{L}\displaystyle\int_{-L}^{L} f(x)\sin\frac{n\pi x}{L}\mathrm{d}x, \quad n = 1,2,3,\cdots \end{cases}$$

对于非周期函数也一样可以进行傅里叶展开，展开的效果其实就是把［$-L$，L］范围认为是周期函数的一个周期，其余部分都默认自动进行了周期性拓展。在 MATLAB 中并没有专门的傅里叶展开函数，不过完全可以根据上述公式以及前述积分函数写出自己的傅里叶展开算法，如图 5-10 所示。

EX 5-9 傅里叶展开

```
syms x
f = abs(x)/x; % 构造一个方波
p = 10; % 设置展开项数
L = pi; % 原函数半周期
fs=int(f,x,-L,L)/L/2;
for n=1:p
    a=int(f*cos(n*pi*x/L),x,-L,L)/L;
    b=int(f*sin(n*pi*x/L),x,-L,L)/L;
    fs=fs+a*cos(n*pi*x/L)+b*sin(n*pi*x/L);
end
fs % 显示傅里叶展开结果
fplot(f);hold on;
fplot(fs);
```

```
fs =
```

$$\frac{4\sin(3x)}{3\pi}+\frac{4\sin(5x)}{5\pi}+\frac{4\sin(7x)}{7\pi}+\frac{4\sin(9x)}{9\pi}+\frac{4\sin(x)}{\pi}$$

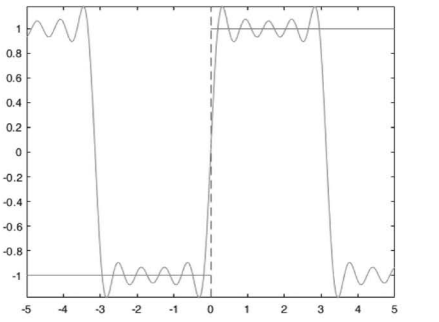


图 5-10　傅里叶展开例程

说明：

（1）例程中 p 表示展开的项数，在算法中对应于循环的次数；L 为函数的半周期，即函数周期为 $2L$。

（2）如图形中所示，在一个周期范围内，傅里叶展开可以将函数值很好地逼近，级数的项数越多仿真精度就越高。

5.4　插值与拟合

在科学及工程研究的过程中，有时会得到一些“成组数据”，这些数据可能是一维的或者多维的，并且代表着某几个变量之间的逻辑关系，这些成组数据被称为“样本点数据”。样本点数据可能不够密集，或者其中恰好没有想要得到的位点的数据，用户可以通过算法，向已知数据中“插入新的位点以得到新值”，称为“插值”。插值算法不改变输入数据，而是尽可能地使插入的新值“和谐”，让得到的新值从逻辑上“最可能”符合数据的内存逻辑关系。

从已知数据中得到新数据，往往并不足够，用户还希望得到数据与数据之间内存逻辑关系——函数关系，这样就需要一套算法“模拟出一个函数使之与样本数据相合”，因此称为“拟合算法”，拟合算法首先需要假设一个可能的“原型函数”，再根据样本点数据求出一套

“可行参数”，保证与所有样本点都尽可能地“接近”，而且一般来说，都只是接近而不能完全通过样本点，但这样拟合的结果函数已经完全足够用于代表数据之间的真实规律了。

5.4.1　一维插值

在已知点范围内进行插值称为“内插”，在范围外进行插值则称为“外插”；从时间概念上讲，如果要插值的位点处于已知点之后，则称为“预报”。一维插值面向的是一维的已知数据的插值方法，如图 5-11 所示，输入的数据为向量形式。

EX 5-10 一维插值

```
x = 0:pi/4:2*pi;
v = sin(x); % 定义已知数据点
xq = 0:pi/16:2*pi; % 定义将查询点（更精细）
% 默认线性插值
ax(1) = subplot(1,2,1);
vq1 = interp1(x,v,xq);
plot(x,v,'o',xq,vq1,':.');
title('(Default) Linear Interpolation');
% 三次样条插值
ax(2) = subplot(1,2,2);
vq2 = interp1(x,v,xq,'spline');
plot(x,v,'o',xq,vq2,':.');
title('Spline Interpolation');
axis(ax, "square")
```

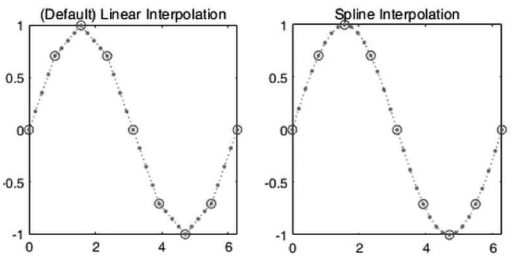


图 5-11　一维插值例程

说明：

（1）一维插值函数 interp1()对于输入已知点的向量并不要求单调，只要长度一致即可。

（2）不同的插值方法对于计算结果与计算速度有很大的不同，如图 5-11 所示，一般来说，三次样条插值(spline)插值效果最好，计算速度最慢。主要的插值方法及说明如表 5-1 所示。

表 5-1　一维插值主要的插值方法及说明

方　法	说　明	连续性	最少已知点
'linear'	邻点的线性插值(默认方法)	C^0	2
'nearest'	距样本网格点最近的值	不连续	2
'next'	下一个抽样网格点的值	不连续	2
'previous'	上一个抽样网格点的值	不连续	2
'pchip'	邻点网格点处数值的分段三次插值	C^1	4
'makima'	基于阶数最大为 3 的多项式的分段函数	C^1	2
'spline'	邻点网格点处数值的三次插值 (比'pchip'需要更多内存和计算时间)	C^2	4

5.4.2　二维网格数据插值

对于二维网格数据插值采用 interp2()函数，同样也有几种插值方法，如图 5-12 所示，与一维稍有不同，代码格式为

```
interp2(x0, y0, z0, x1, y1, 'method')
```

EX 5-11 二维网格数据插值

```
[X,Y] = meshgrid(-3:3);
V = peaks(X,Y);
subplot(2,2,1);
surf(X,Y,V)
title('Original Sampling');
[Xq,Yq] = meshgrid(-3:0.2:3);
%
Vq = interp2(X,Y,V,Xq,Yq);
subplot(2,2,2);
surf(Xq,Yq,Vq);
title('Linear');
%
Vq = interp2(X,Y,V,Xq,Yq,'cubic');
subplot(2,2,3);
surf(Xq,Yq,Vq);
title('Cubic');
%
Vq = interp2(X,Y,V,Xq,Yq,'spline');
subplot(2,2,4);
surf(Xq,Yq,Vq);
title('Spline');
```

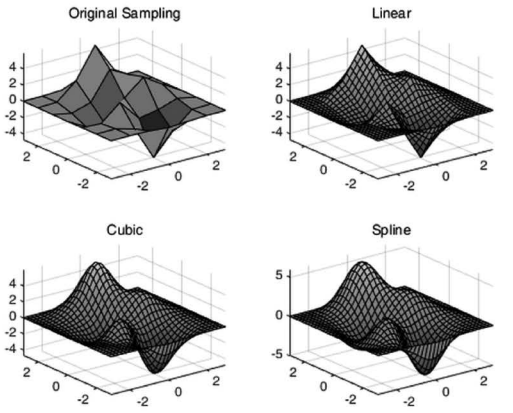


图 5-12　二维网格数据插值例程

说明：interp2()函数只应用于“网格数据”，即要求在平面某范围内，所有网格点上均有数据。该函数常用方法及说明如表 5-2 所示。

表 5-2　二维网格数据插值常用设置及说明

方　法	说　　明	连续性	每维度最少已知点
'linear'	邻点的线性插值(默认方法)	C^0	2
'nearest'	距样本网格点最近的值	不连续	2
'cubic'	邻点网格点处数值的三次卷积插值	C^1	4
'makima'	基于阶数最大为 3 的多项式的分段函数	C^1	2
'spline'	邻点网格点处数值的三次插值 (比'cubic'需要更多内存和计算时间)	C^2	4

5.4.3　二维一般数据插值

对于非网格数据，MATLAB 提供了一个面向更为一般数据的插值函数 griddata()，如图 5-13 所示，代码格式为

```
griddata(x0, y0, z0, x, y, 'method')
```

说明：

(1) 四种插值方法如图 5-13 所示，其中，'natural'方法是基于三角剖分的自然邻点插值，支持二维和三维插值，该方法在线性与立方之间达到有效的平衡。另外还有一种'v4'方法，采用双调和样条插值方法，仅支持二维插值，且不是基于三角剖分，目前公认效果较好。

(2) 对于三维的网格样本点，可以使用函数 interp3()或者更一般的 n 维插值函数 interpn()；而对于多维的非网格样本点，则使用与之对应的 griddata3()和 griddatan()函数。

EX 5-12 二维一般数据插值

```
x = -3 + 6*rand(30,1);
y = -3 + 6*rand(30,1);
v = sin(x).^4 .* cos(y); % 构造已知数据
[xq,yq] = meshgrid(-3:0.1:3); % 构造插值网
subplot(2,2,1);
z1 = griddata(x,y,v,xq,yq,'nearest');
plot3(x,y,v,'o'); hold on
mesh(xq,yq,z1)
title('Nearest Neighbor')
subplot(2,2,2);
z2 = griddata(x,y,v,xq,yq,'linear');
plot3(x,y,v,'o'); mesh(xq,yq,z2)
title('Linear')
subplot(2,2,3);
z3 = griddata(x,y,v,xq,yq,'natural');
plot3(x,y,v,'o'); mesh(xq,yq,z3)
title('Natural Neighbor')
subplot(2,2,4);
z4 = griddata(x,y,v,xq,yq,'cubic');
plot3(x,y,v,'o'); mesh(xq,yq,z4)
title('Cubic')
```

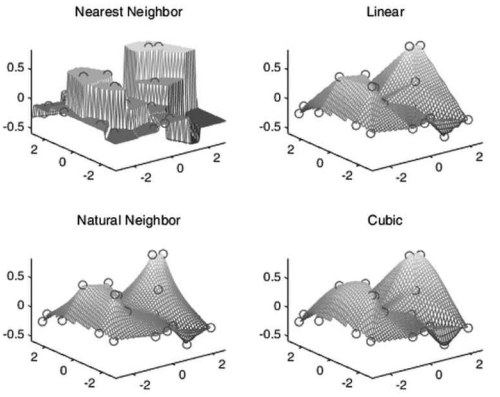


图 5-13　二维一般数据插值例程

5.4.4　多项式拟合

多项式拟合是一种最常用和实用的拟合手法，前述的泰勒展开本质上也属于一种多项式拟合，只不过要求有已知函数表达式，而拟合行为只需要样本点数据作为输入即可，如图 5-14 所示。

EX 5-13 多项式拟合

```
% 准备样本数据并作图
x = linspace(0,1,5);
y = 1./(1+x);
plot(x,y,'o'); hold on
% 原始函数图像
fplot(@(x) 1./(1+x),[-0.5 2],'--')
% 拟合4次多项式并作图
p = polyfit(x,y,4)
% 按三位精度显示多项式
fitFunc = poly2sym(vpa(p,3))
fplot(poly2sym(p),[-0.5 2])
legend('Sample','Origin','Fit')
```

```
p = 1×5
    0.1524   -0.5333    0.8667   -0.9857    1.0000
```

$$\text{fitFunc} = 0.152x^4 - 0.533x^3 + 0.867x^2 - 0.986x + 1.0$$

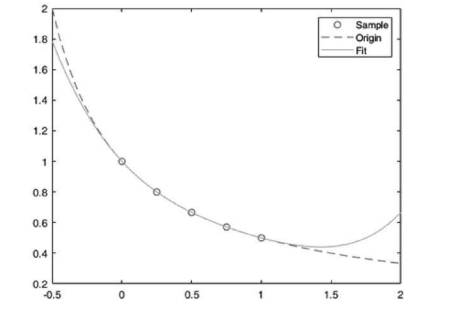


图 5-14　多项式拟合例程

说明：

（1）多项式拟合函数 polyfit()的第三个参数代表多项式拟合的阶数，阶数越多，精度越高。

（2）vpa()函数用于将输入数据转换为变精度数据（Variable-precision Arithmetic），第二个参数表示保留小数点后的数字位数，如若此处不采用 vpa()函数，则会输出无必要的长分式数字，影响对于多项式的快速感观。

（3）从图形中可见，在样本数据点范围内拟合的效果相当好，但是在此范围之外逐渐远

离了原始的函数图像，这是拟合动作所无能为力的，如果想得到足够贴近真相的模型，就必须获得足够范围的样本点数据作为输入。

5.4.5 最小二乘拟合：拟合的优化之路

许多场景下，用户希望拟合的函数形式并不一定总是多项式，对于更一般的原型函数，MATLAB的优化工具箱（Optimization Toolbox）提供了函数lsqcurvefit()，基于最小二乘（Least-squares，LSQ）原理实现曲线拟合（Curve-fitting），该函数的输入原型可以是M函数名及匿名函数，可以高效地求解出要拟合的参数，如图5-15所示。

EX 5-14 最小二乘拟合

$$f(x)=\frac{a_1}{a_2+a_3x}$$

```
% 准备样本数据并作图
x = linspace(0,1,5);
y = 1./(2+3*x);
plot(x,y,'o'); hold on
% 原始函数图像
fplot(@(x) 1./(2+3*x),[-0.5 2],'--')
% 定义原型函数
func = @(a,x) a(1)./(a(2)+a(3)*x)
% 最小二乘拟合
a = lsqcurvefit(func, [1 1 1], x, y)
% 作图 - 系数a已知
fplot(@(x) a(1)./(a(2)+a(3)*x),[-0.5 2])
legend('Sample','Origin','Fit')
```

```
func = 包含以下值的 function_handle:
    @(a,x)a(1)./(a(2)+a(3)*x)
a = 1×3
    3.2783    6.5565    9.8348
```

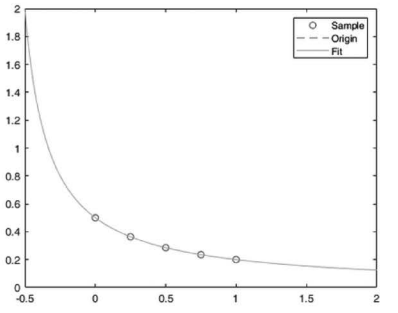


图5-15 最小二乘拟合例程

说明：

（1）定义原型函数时，注意要将待定参数定义到函数的自变量集中。而在使用fplot()函数对拟合函数作图时，参数a已经不是未知量，需要将其从自变量集中取出。

（2）lsqcurvefit()函数的第二个参数为用户给定的参数初值，一般来说可以随意给定，如果希望提高运算速度，则可以尽量给出预测参数值。

（3）从图5-15可见，在原型函数的结构选择准确的前提下，拟合算法可以很好地得到准确的拟合函数，且在样本数据范围外的部分也能很好地进行预测。

5.5 代数方程与优化

由已知数与未知数通过代数运算组成的方程称为代数方程，线性方程作为简单的一次代数方程（组）已在5.2节线性代数中展示过解法，本节重点介绍非线性方程的求解问题。

优化（Optimization）问题是运筹学（Operations Research）的主要组成部分，以数学分析和组合学为主要工具，提到数学分析就离不开数学建模，而优化问题的数学模型往往就是代数方程（包括等式与不等式方程），因此优化计算问题最终大多转化为解代数方程问题。

规划（Programming）与优化向来是运筹学中概念区别不清晰的两个词，根据中国运筹

学会引用《数学大辞典》的说法来看，优化与规划所包含的内容基本一致，因此也不加以区分；习惯上宏观概念多称为优化，而具体的类别问题多称为规划，如线性规划、非线性规划等。

5.5.1　代数方程的求解

解析求解函数为 solve()，数值求解函数为 vpasolve()，具体应用如图 5-16 所示。

EX 5-15 代数方程

解一元方程：$x^2+x-1=0$

```
syms x
eqn = x^2+x-1 % 方程定义
solveSym1 = solve(eqn,x) % 解析解
solveNum1 = vpasolve(eqn) % 数值解
```

解多元方程：$\begin{cases} x+y=0 \\ x^2+y^2=1 \end{cases}$

```
syms x y
eqn1 = x+y, eqn2 = x^2+y^2-1 % 方程定义
[xs,ys] = solve(eqn1,eqn2) % 解析解
[xn,yn] = vpasolve(eqn1,eqn2) % 数值解
```

eqn = x^2+x-1

solveSym1 = $\begin{pmatrix} -\frac{\sqrt{5}}{2}-\frac{1}{2} \\ \frac{\sqrt{5}}{2}-\frac{1}{2} \end{pmatrix}$　　solveNum1 = $\begin{pmatrix} -1.62 \\ 0.618 \end{pmatrix}$

eqn1 = $x+y$　　eqn2 = x^2+y^2-1

xs = $\begin{pmatrix} \frac{\sqrt{2}}{2} \\ -\frac{\sqrt{2}}{2} \end{pmatrix}$　　ys = $\begin{pmatrix} -\frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} \end{pmatrix}$

xn = $\begin{pmatrix} 0.707 \\ -0.707 \end{pmatrix}$　　yn = $\begin{pmatrix} -0.707 \\ 0.707 \end{pmatrix}$

图 5-16　代数方程例程

说明：

(1) 将方程整理为等号右侧为零的形式后，只需要输入等式左侧即可，等号和等号右侧零的部分会自动补充。

(2) 对于解析解函数 solve()，如果所求方程无法求出解析解，系统会提示，并且自动转为使用 vpasolve()函数求得数值解。

5.5.2　无约束优化

无约束优化是最简单的一类优化问题，其目标就是求某一函数的最小值(最大值只需要乘一个负号即化为最小值问题)；在计算之前，对函数进行作图有一个直观的印象往往是有必要的，在 MATLAB 中，既有软件主体自带的函数 fminsearch()，也有优化工具箱提供的等效函数 fminunc()，函数名意为“find minimum of unconstrained multivariable function”，下面用著名的 Rosenbrock 香蕉函数来举例，如图 5-17 所示，该函数是一个常用来测试最优化算法性能的非凸函数。

说明：

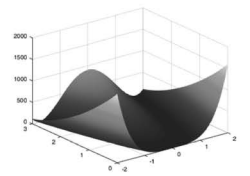
(1) 设置函数时，注意将多元自变量写为 x(1)、x(2)这样的形式。

(2) 从图 5-17 中可见，fminsearch()函数不如优化工具箱提供的 fminunc()函数收敛的速度快，对于绝大部分其他类型函数也是类似的效果，因此后者拥有更优异的计算性能，是优化问题的首选函数。

EX 5-16 无约束优化问题

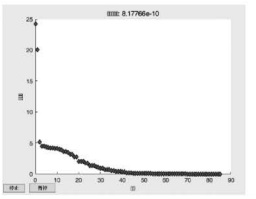
$f(x)=100(x_2-x_1^2)^2+(1-x_1)^2$，求：$\min\limits_x f(x)$

```
% 作surf图-直观印象
[x,y]=meshgrid(-2:.1:2,0:.1:3);
z = 100*(y - x.^2).^2 + (1 - x).^2;
surf(x,y,z);
shading interp;
% 设置函数
fun = @(x) 100*(x(2) - x(1)^2)^2 + (1 - x(1))^2;
% 设置优化初值
x0 = [-1.2,1];
% 设置显示迭代目标函数图
options = optimset('PlotFcns',@optimplotfval);
% 方法1-MATLAB
x1 = fminsearch(fun,x0,options)
% 方法2-优化工具箱
x2 = fminunc(fun,x0,options)
```



```
x1 = 1×2
    1.0000    1.0000

x2 = 1×2
    1.0000    1.0000
```



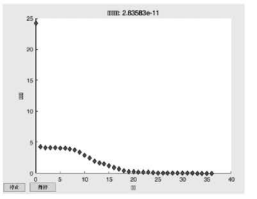


图 5-17　无约束优化例程

5.5.3　线性规划：高效决策工具

线性规划问题(Linear Programming)，是“有约束优化问题”中最简单的一类问题，在线性规划中，目标函数和约束函数都是线性的，约束函数可能是不等式或等式及自变量上下界，数学描述为

$$\min \boldsymbol{f}^{\mathrm{T}}\boldsymbol{x} \quad \text{s.t.}\begin{cases}\boldsymbol{A}\boldsymbol{x}\leqslant\boldsymbol{b}\\ \boldsymbol{A}_{\mathrm{eq}}\boldsymbol{x}=\boldsymbol{b}_{\mathrm{eq}}\\ \boldsymbol{lb}\leqslant\boldsymbol{x}\leqslant\boldsymbol{ub}\end{cases}$$

其中，记号“s.t.”翻译为“使得”，是“subject to”的简写，是极为常用的数学符号。MATLAB为线性规划问题提供了基于单纯形法的函数 linprog()，如图 5-18 所示，常用代码格式为

```
[x, fval] = linprog(f,A,b,Aeq,beq,lb,ub)
```

EX 5-17 边界约束优化——线性规划

$f(x)=-2x_1-x_2$ 其中：$2x_2\leqslant 7$；$x_1+4x_2\leqslant 9$；$x_1\leqslant 4$；$x_2\leqslant 8$，

求：$\min\limits_x f(x)$

```
f = [-2 -1]'; % 函数定义
A = [0 2; 1 4]; b = [7; 9];
Aeq = []; beq = [];
lb = []; ub = [4; 8];
[x, fval] = linprog(f,A,b,Aeq,beq,lb,ub)
```

```
Optimal solution found.
x = 2×1
    4.0000
    1.2500

fval = -9.2500
```

图 5-18　线性规划例程

说明：

(1) 计算结果显示，当 x(1)为 4 且 x(2)为 1.25 时，函数将在满足约束条件的前提下达到最小值 fval 为－9.25。

(2) 计算优化问题的前提是问题合理，如果在已知的约束条件下，函数本身并没有最值，那么即使是 MATLAB 也无能为力，一般会提示“问题是欠约束的”(Problem is

unbounded)。

(3) 对于函数的参数中并有的参数，应使用空矩阵符号“[]”来占位。

5.5.4　非线性规划

非线性规划问题，是“有约束优化问题”中较为复杂的一类，约束中不仅包含线性不等式、等式、自变量上下界，还可能包括非线性的不等式及等式，数学表达形式为

$$\min f(\boldsymbol{x}) \text{ s.t.} \begin{cases} \boldsymbol{A}\boldsymbol{x} \leqslant \boldsymbol{b} \\ \boldsymbol{A}_{\text{eq}}\boldsymbol{x} = \boldsymbol{b}_{\text{eq}} \\ \boldsymbol{lb} \leqslant \boldsymbol{x} \leqslant \boldsymbol{ub} \\ \boldsymbol{C}(\boldsymbol{x}) \leqslant \boldsymbol{0} \\ \boldsymbol{C}_{\text{eq}}(\boldsymbol{x}) = \boldsymbol{0} \end{cases}$$

这一类问题都可以用 MATLAB 优化工具箱中提供的边界约束优化函数 fmincon()来解决，函数名意为“find minimum of constrained nonlinear multivariable function”，例程如图 5-19 所示，函数常用代码形式为

```
[x, fval] = fmincon(fun,x0,A,b,Aeq,beq,lb,ub)
```

EX 5-18 边界约束优化——非线性规划

$f(x) = -x_1x_2x_3$ 其中：$0 \leqslant x_1 + 2x_2 + 4x_3 \leqslant 12$，求：$\min\limits_{x} f(x)$

```
% 不等式约束
func = @(x) -x(1)*x(2)*x(3);
x0 = [1; 1; 1];
A = [-1 -2 -4; 1 2 4];
b = [0; 12];
[x1, val1] = fmincon(func, x0, A, b)
```

```
x1 = 3×1
    4.0000
    2.0000
    1.0000

val1 = -8.0000
```

$f(x) = -x_1x_2x_3$ 其中：$x_1 + 2x_2 + 4x_3 = 24$，求：$\min\limits_{x} f(x)$

```
% 等式约束
func = @(x) -x(1)*x(2)*x(3);
x0 = [1; 1; 1];
A = [1 2 4];
b = 24;
[x2, val2] = fmincon(func, x0, [], [], A, b)
```

```
x2 = 3×1
    8.0000
    4.0000
    2.0000

val2 = -64.0000
```

图 5-19　非线性规划例程

说明：

(1) fmincon()函数的第二个参数为计算初值；第三个参数 A 和第四个参数 b 为一组，表示约束边界为 $\boldsymbol{A}\boldsymbol{x} \leqslant \boldsymbol{b}$，第五个参数 Aeq 与第六个参数 beq 为一组，表示约束边界为 $\boldsymbol{A}_{\text{eq}}\boldsymbol{x} = \boldsymbol{b}_{\text{eq}}$。当没有第三个和第四个参数时，使用空矩阵符号“[]”占位。

(2) 注意该函数要求目标函数与约束函数必须都是连续的，否则可能会给出局部最优解而不是全局最优解。

5.5.5 最大值最小化问题

最大值最小化问题(Minimax Constraint Problem)其实是一个实际应用中比较常见的问题，却由于问题类型不易被理解而很容易被忽略。举一个具体的例子来理解该问题，比如在一个战场中对于急救中心的选址，要求它到各阵地的运输伤员的时间 $f_i(x)$不可以太长，对于所有选址方案中能保证到达所有阵地中所需最长时间 $\max f_i(x)$最小的，就是最佳方案。也就是说，伤员在运输过程中所消耗的每分钟都是损失，耽误时间越久就越危险，如何让最大危险降至最小，就是最大值最小化问题。该问题的数学描述与前述非线性规划非常类似：

$$\min_{x}\max_{i} if(\boldsymbol{x}) \quad \text{s.t.}\begin{cases}\boldsymbol{A}\boldsymbol{x}\leqslant\boldsymbol{b}\\ \boldsymbol{A}_{\text{eq}}\boldsymbol{x}=\boldsymbol{b}_{\text{eq}}\\ \boldsymbol{lb}\leqslant\boldsymbol{x}\leqslant\boldsymbol{ub}\\ \boldsymbol{C}(\boldsymbol{x})\leqslant\boldsymbol{0}\\ \boldsymbol{C}_{\text{eq}}(\boldsymbol{x})=\boldsymbol{0}\end{cases}$$

而且在 MATLAB 中求解计算的形式也非常类似，如图 5-20 所示，函数使用 fminimax()，其代码格式为

```
[x,fval] = fminimax(fun,x0,A,b,Aeq,beq,lb,ub)
```

EX 5-19 最大值最小化问题

$f(x)=\begin{bmatrix}-x_1x_2x_3\\ x_1-x_2+x_3\end{bmatrix}$ 其中：$0\leqslant x_1+2x_2+4x_3\leqslant 12$，求：$\min_{x}\max_{i} f_i(x)$

```
% 不等式约束
func = @(x) [-x(1)*x(2)*x(3); x(1)-x(2)+x(3)];
x0 = [1; 1; 1];
A = [-1 -2 -4; 1 2 4];
b = [0; 12];
[x1, val1] = fminimax(func, x0, A, b)
```

```
Local minimum possible. Constraints satisfied.

x1 = 3×1
    1.1786
    4.3726
    0.5191

val1 = 2×1
   -2.6749
   -2.6749
```

图 5-20 最大值最小化例程

说明：fminimax()函数与前述 fmincon()函数的参数几乎完全一致，不同的是，fminimax()函数要求输入目标函数组，这样才能对一组函数取最大值。

5.6 微分方程

微分方程(Differential Equation，DE)分为常微分方程(Ordinary Differential Equation，ODE)和偏微分方程(Partial Differential Equation，PDE)。常微分方程中的“常”并不是“常系数”的意思，而是“正常”(Ordinary，通常的、普通的)的常，相对于“偏”(Partial，局部的、片面的)，理解记忆两种微分方程的区别可以将关注点放在“偏”字上，即包含偏导数的微分方程为偏微分方程，不包含偏导数的微分方程即为常微分方程。目前，人类对于经典物理世界

的认知，基本都是建立在微分方程的基础上的，这其中的底层逻辑在于，物理量的导数代表着该量的原因，积分代表该量的结果，而微分方程正是物理量的因果关系，这也是为什么许多学科的数学模型都是微分方程，而对于这些学科而言，解决变量之间的影响关系，就等同于求解微分方程。

5.6.1　常微分方程解析解

常微分方程相比于偏微分方程更简单易于求解，对于线性常微分方程和一些特殊的低阶非线性常微分方程一般可以求得解析解，而绝大多数的非线性常微分方程是没有解析解的，这时需要退而求其次，求得方程的数值解，得到了求解区域的数值解，也就可以做出该解函数的图像，这无论是对理解函数性质还是分析函数数值都可以提供极大帮助，甚至可以再利用数据进行函数拟合，得到一个近似的解析解。

对常微分方程求解析解首先需要定义符号变量与符号函数，进而将微分方程定义为符号方程，如果方程有初始条件则也需要定义，然后直接使用 dsolve()函数即可完成求解，如图 5-21 所示。

EX 5-20 常微分方程解析解

1. 无初始条件微分方程

$$\frac{d^2}{dt^2}y = ay$$

```
syms a t y(t) z(t)
eqn = diff(y,t,2) == a*y % 符号方程
ySol(t) = dsolve(eqn)
```

2. 有初始条件的微分方程

$$\frac{dy}{dt} = ay, y(0) = 5$$

```
eqn = diff(y,t) == a*y % 符号方程
cond = y(0) == 5; % 初始条件
ySol(t) = dsolve(eqn,cond)
```

3. 微分方程组

$$\begin{cases} \frac{dy}{dt} = z \\ \frac{dz}{dt} = -y \end{cases}$$

```
eqns = [diff(y,t) == z, diff(z,t) == -y]
sol = dsolve(eqns);
solY = sol.y, solZ = sol.z
```

```
eqn(t) =
∂²/∂t² y(t) = a y(t)
ySol(t) = C4 e^(-√a t) + C5 e^(√a t)

eqn(t) =
∂/∂t y(0) = a y(t)
ySol(t) = 5 e^(a t)

eqns(t) =
(∂/∂t y(t) = z(t)   ∂/∂t z(t) = -y(t))
solY = C8 cos(t) + C7 sin(t)
solZ = C7 cos(t) - C8 sin(t)
```

图 5-21　常微分方程解析解例程

说明：

（1）符号函数的定义即为函数字母加括号带自变量，如 $y(t)$等，与数学中的书写完全一致；注意符号方程中的等号是双等号（==），而不是赋值符号（=）；仍然遵从一切都是矩阵的原则，方程组本质是也是方程的矩阵，需要用方括号括起来。

（2）对于方程组的解，dsolve()函数将输出一个结构体，结构体的字段即为求解的函数。

（3）常数 C 的角标完全取决于该程序之前的使用情况，角标值仅区分意义，同一个角标代表同一个常数。

（4）对于无法求出解析解的方程，软件会提示“无法找到解析解”（Unable to find explicit solution），这时就要考虑使用数值解法了。

5.6.2 常微分方程数值解

科学家很早就意识到解析解不是大多数微分方程的解决方案，于是纷纷研究探索数值解法，比如欧拉法、龙格-库塔法、亚当斯-莫尔顿法等，MATLAB 内置的求解 ODE 的函数就是基于这些方法的，比如其中最常用而强大的 ode45()函数就是基于四五阶龙格-库塔法的算法。

MATLAB 共内置 8 个 ODE 函数，对于初学者以及不专门研究它的用户，建议不必深究其中的区别，实际使用时选取方法是：首选 ode45()函数，如图 5-22 所示；如果无法解算或速度极慢，则可尝试改用 ode15s()函数；如果 ode45()函数可以求解，但是速度略慢而精度并不要求那么高，则可尝试改用 ode23()函数。更为精微的函数区分可进入帮助文档中搜索“选择 ODE 求解器”查看。

EX 5-21 常微分方程数值解

1. 一个分量的微分方程

$y' = 2t$

```
tspan = 0:0.1:2; % 求解范围
y0 = 0; % 初值条件
[t,y] = ode45(@(t,y) 2*t, tspan, y0);
plot(t,y,'-o')
```

2. 微分方程组

$$\begin{cases} y_1' = y_2 \\ y_2' = (1-y_1^2)y_2 - y_1 \end{cases}$$

```
f = @(t,y) [y(2); (1-y(1)^2)*y(2)-y(1)];
[t,y] = ode45(f, 0:0.2:16,[2; 0]);
plot(t,y(:,1),'-o',t,y(:,2),'-o')
legend('y_1','y_2')
```

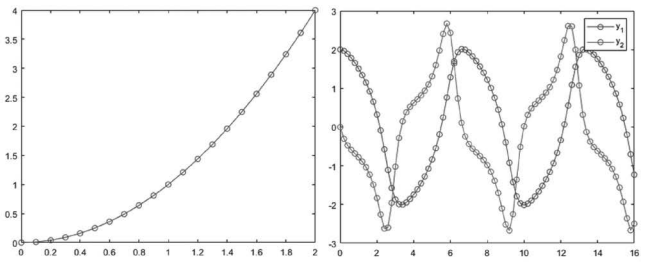


图 5-22 常微分方程数值解例程

说明：

（1）设定的求解范围 **tspan** 是一个向量，向量的步长即为求解步长，步长越小则精度越高、算量越大。

（2）注意设定微分方程时，必须将求解函数（如 y）和自变量（如 t）均列入参数列表中。

5.6.3 微分方程 Simulink 求解

Simulink 既是 MATLAB 的一个工具箱，同时也是几乎可以与 MATLAB 相并列的一款独立软件，就是因为其实在过于强劲，在某些方面甚至掩盖了 MATLAB 的光芒。由于 Simulink 也可以用于解一些微分方程，并且还十分方便，因此在这里作简要介绍。

Simulink 将所有数据看成可以传递的信号，从一个模块单元中输出并输入另一个模块单元中，Simulink 打包了各行各业可能会用到的大量的模块，以至于几乎没有可以新增的模块了，用户所要做的，就是将模块单元拖入模型中，连线并进行设置，即可运行计算。Simulink 计算的对象称为“模型”，其意为对于现实物理系统的“数字孪生”，擅长所有可以

称为系统的问题，微分方程(组)就是最简单且典型的一类系统，下面举一实例，解微分方程组：

$$\begin{cases}\dot{x}_1 = -x_1 + x_2 \\ \dot{x}_2 = -2x_2 + 1 \\ \dot{x}_3 = -x_1 x_2\end{cases}$$

首先进入 Simulink 模块，方法是在命令行中输入 simulink 再按下 Enter 键即可，选择新建空模型(blank model)，单击窗口上方模块浏览器按钮(library browser)，选择需要的模块向模型中拖动，如图 5-23 所示，左侧即为搭建的模型，三个积分器的初值均设为 0，使用 Mux 混路器模块将三个输入组合为一个向量输出，再使用 Fcn 函数模块计算方程等式的右侧，再将计算输出回路连接到对应积分器上，并且可以在 $x(t)$ 输出端插入一个示波器 Scope，这样便可以看到计算的输出结果。在本书配套的代码文件中，模型文件为 pdeModel.slx，读者可以直接打开运行。

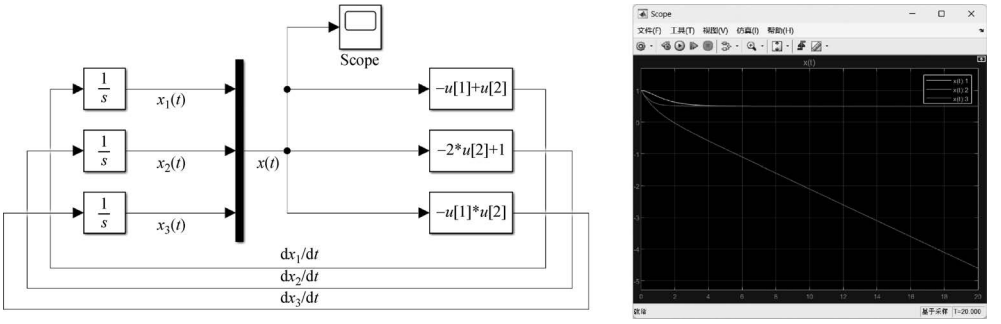


图 5-23　微分方程 Simulink 求解

从示波器图中可以看出，x_1 与 x_2 均稳定在 0.5 的位置，而 x_3 稳定为一条直线，这样即相当于求得了微分方程的数值解。Simulink 的建模求解方式极为强大，虽然对于简单的小型问题，使用 MATLAB 解方程函数会更为快捷，然而对于求解较大规模的问题、模块化系统问题时，则更显示出 Simulink 的四两拨千斤，尤其对于时间延迟性微分方程来说，更是只有用 Simulink 才能实现求解。

5.6.4　抛物-椭圆型偏微分方程

常微分方程的解析解已经比较难求，到了偏微分方程，连求解数值解都困难了，在 MATLAB 的核心组件中，只对一类偏微分方程提供了数值解求解函数，这一类称为一阶抛物-椭圆型 PDE，数学形式如下：

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

实际求解例程如图 5-24 所示。

EX 5-22 偏微分方程数值解

方程：$\pi^2\frac{\partial}{\partial t}u=\frac{\partial}{\partial x}\left(\frac{\partial}{\partial x}u\right)$

方程区间：$x\in[0,1],\ t\in[0,\infty]$

初始条件：$u(x,0)=\sin\pi x$

边界条件：$u(0,t)\equiv 0,\ \pi e^{-t}+\frac{\partial}{\partial x}u(1,t)=0$

```
m = 0; x=0:0.03:1; t=0:0.2:2;
u = pdepe(m,@pdeeqn,@pdeic,@pdebc,x,t);
surf(x,t,u); title('Numerical solution')
xlabel('x'); ylabel('t'); zlabel('u');
plot(x,u(end,:)); title('Solution at t = 2')
xlabel('x'); ylabel('u(x,2)')
```

FUNCTION-EX-5-23

方程（equation, eqn）

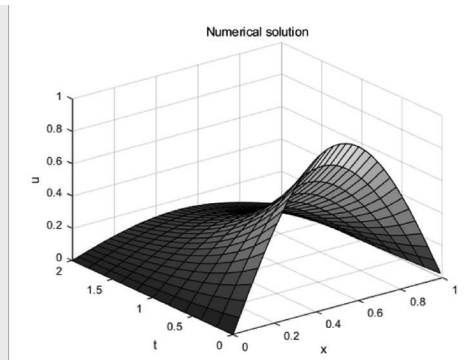
```
function [c,f,s] = pdeeqn(x,t,u,DuDx)
c = pi^2; f = DuDx; s = 0;
end
```

初始条件（initial condition, ic）

```
function u0 = pdeic(x)
u0 = sin(pi*x);
end
```

边界条件（boundary condition, bc）

```
function [pl,ql,pr,qr] = pdebc(xl,ul,xr,ur,t)
pl = ul; ql = 0; pr = pi * exp(-t); qr = 1;
end
```



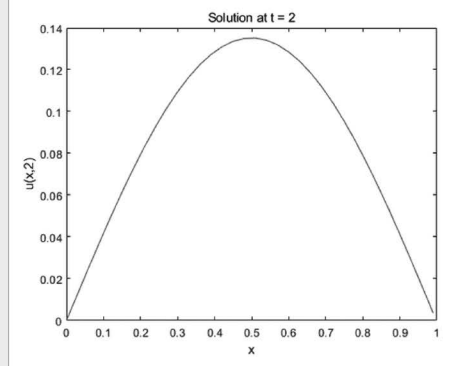


图 5-24　抛物-椭圆型偏微分方程例程

一阶抛物-椭圆型 PDE 只是诸多偏微分方程中的一类，因此，MATLAB 核心组件对于偏微分方程的求解还远远不够，实际上，MATLAB 将其强大的偏微分方程求解功能集成在了优秀的“偏微分方程工具箱”（Partial Differential Equation Toolbox，PDET）中，而且还拥有图形用户界面，可以让用户非常方便地求解偏微分方程，因此建议初学者或不以此为研究的用户首选 PDET。

5.6.5　偏微分方程工具箱

一个标量物理量在空间中的表示为

$$u=u(x,y,z,t)$$

这也是目前人类对于世界的认知，即三维空间加一维时间（其中，时间单方向流动），如果把该物理量的值按颜色绘制在空间中，则会显示出一幅“云图”，这就是“三维空间中的物理场”，而场中每点的量值还会随着时间变化，这就是所谓的“瞬态物理场”。物理场是关于时间与空间的函数，导数表示物理量的原因，积分表示物理量的结果，因而该函数所要满足的客观规律即为偏微分方程，这就是为什么物理学几乎全部由偏微分方程所支撑起来，也是偏微分方程为什么会成为科学工程领域至关重要的环节的原因。

物理学中常见的偏微分方程最多包含二阶偏导数，而 MATLAB 提供的偏微分方程工

具箱可以非常简易有效地求解“二阶偏微分方程”，还包括多元 PDE 以及 PDE 组，二阶 PDE 的数学形式如下：

$$m\frac{\partial^2 u}{\partial t^2}+d\frac{\partial u}{\partial t}-\nabla\cdot(c\nabla u)+au=f$$

其中∇为 Nabla 算子，∇表示梯度，∇·表示散度，如果 c 为常数，则梯度的散度其实是代表“所有非混合二阶偏导数”之意，即

$$\nabla\cdot(c\nabla u)=c\left(\frac{\partial^2}{\partial x_1^2}+\frac{\partial^2}{\partial x_2^2}+\cdots+\frac{\partial^2}{\partial x_n^2}\right)u=c\Delta u$$

其中，Δ 为 Laplace 算子。

二阶 PDE 与二阶代数方程有所类似，借用按圆锥曲线的分类来划分，包含如下三大类方程形式。

(1) 椭圆型方程(Elliptic)，此时 $m=0$，$d=0$，数学形式为

$$-\nabla\cdot(c\nabla u)+au=f(\boldsymbol{x},t)$$

椭圆型方程没有对于时间的一阶及二阶偏导，因此定解问题中只有边界条件而没有初值条件，主要用来描述物理中的定常、平衡、稳定状态，最典型的就是泊松方程及二维拉普拉斯方程(梯度的散度恒为零)，物理中常用的有定常状态的电磁场、引力场和反应扩散现象等。

(2) 抛物线方程(Parabolic)，此时 $m=0$，数学形式为

$$d\frac{\partial u}{\partial t}-\nabla\cdot(c\nabla u)+au=f$$

包含场对于时间的一阶偏导，因此不仅需要边界条件，还需要一个初值条件，一般用于描述能量耗散系统，物理中常见的抛物线方程有一维热传导方程。

(3) 双曲线方程(Hyperbolic)，此时 $d=0$，数学形式为

$$m\frac{\partial^2 u}{\partial t^2}-\nabla\cdot(c\nabla u)+au=f$$

包含场对于时间的二阶偏导，因此需要边界条件和两个初值条件，没有一阶时间偏导，可以理解为能量不耗散，一般描述能量守恒系统，常见的比如一维弦振动(波动)方程，波动方程的扰动是以有限速度传播的，因而其影响区和依赖区是锥体状的。

PDET 对于上述这些类型的二阶 PDE 可以很方便地在二维或三维几何空间上完成求解，基本原理是采用三角形及四面体网格划分，采用有限单元法求解并对结果后处理得到物理场云图。

PDET 提供两种使用方法，一是打开 GUI(相当于一个软件界面)进行单击与输入操作，二是使用命令代码输入。后者的功能涵盖了前者，对于初学者可以使用 GUI 实现初步功能，再使用代码自动生成功能得到与操作对应的代码并在其基础上修改。下面按照操作步骤举一实际例子。

1. 打开 PDET 界面

方法是输入命令 pdetool 并按下 Enter 键，或在 App 界面中找到 PDE Modeler。界面

的菜单栏就代表着操作顺序：选项(Options)、几何(Draw)、边界(Boudary)、方程(PDE)、网格(Mesh)、求解(Solve)、作图(Plot)，如图 5-25 所示。

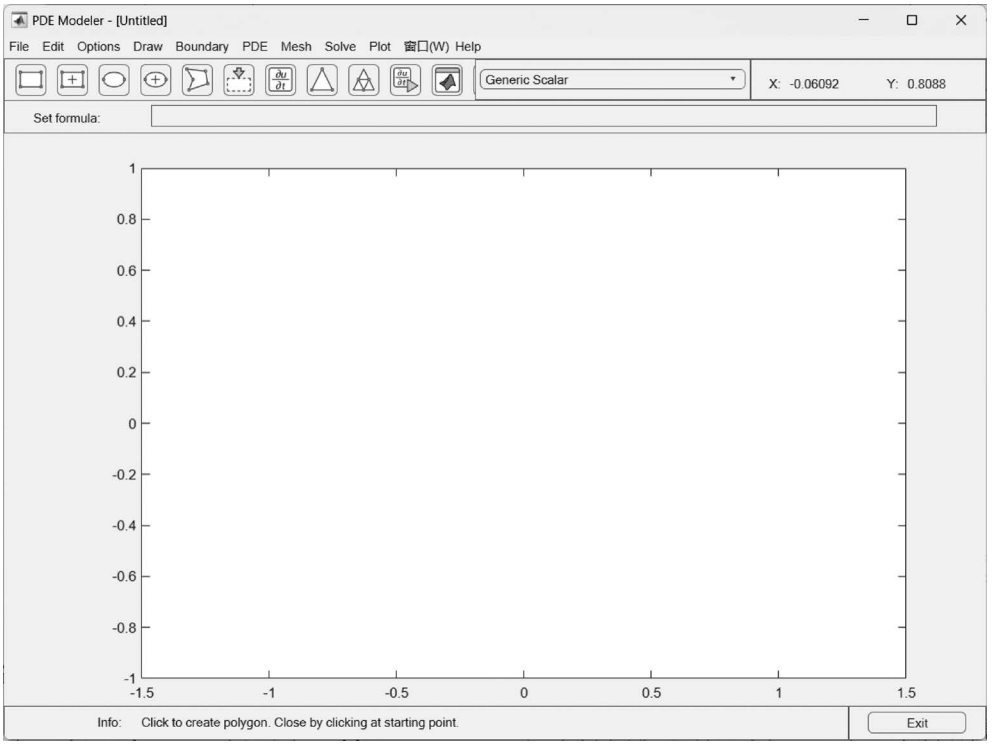


图 5-25 PDET 初始界面

2. 选择 PDE 类型

位于选项(Option)菜单栏下的应用(Application)选项，也位于界面按钮行右侧，10 个选项分别意为：通用标量方程(Generic scalar)、通用系统方程组(Generic system)、机械结构平面应力场(Structural mechanics, plane stress)、机械结构平面应变场(Structural mechanics, plane strain)、静态电场(Electrostatics)、静态磁场(Magnetostatics)、交流电磁场(AC power electromagnetics)、直流电场(Conductive media DC)、热传导温度场(Heat transfer)、扩散(Diffusion)。

正确选择这些物理场，可以在界面上简化参数的输入，获得正确的引导提示。

3. 确定几何域

在 GUI 中指画出平面形状，有 5 项工具：长方形、中心长方形、(椭)圆、中心(椭)圆和多边形，还有一项旋转工具，每创建一个图形，界面上 Set formula 后的文本框内都会出现该图形的“字母＋序号”，字母有 R(矩形)、Q(正方形)、E(椭圆形)、C(圆形)、P(多边形)，加减号表示图形之间的布尔运算，可以自行修改顺序与符号以得到目标求解区域。

双击图形可以打开图形位置尺寸的准确设置窗口，如图 5-26 所示设置了中心为零点半径为 1 的正圆。

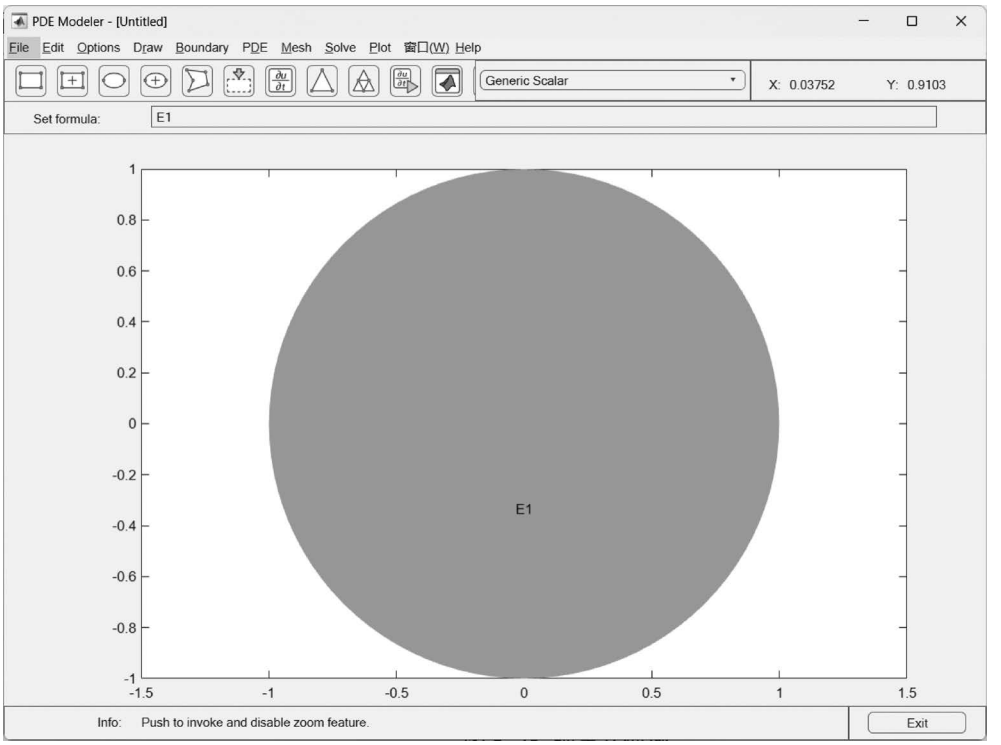


图 5-26　确定几何域

对应代码为

```
pdeellip(0,0,1,1,0,'E1');
```

画图函数只有三个：画(椭)圆函数 pdeellip()、画矩形函数 pderect()、画多边形函数 pdepoly()。还可以载入三维的标准模板库(Standard Template Library,STL)模型,作为三维几何域,此功能并不包含在界面上,需要使用代码来载入,函数为 importGeometry(),代码形式如下：

```
importGeometry(model,'BracketWithHole.stl');
```

4. 设置边界条件

进入边界(Boundary)菜单栏,选择边界模式(Boundary mode)进入状态,这时,边界上会显示颜色与箭头,再进入边界条件设置(Specify boundary conditions),对于单个偏微分方程的边界条件有如下两类。

(1) 狄利克雷(Dirichlet)边界条件,也称第一类边界条件,该边界条件定义了边界处的值,数学形式为

$$hu = r$$

其中,h 与 r 是关于 $\boldsymbol{x}$,t,u,$\partial u/\partial \boldsymbol{x}$ 的函数,更为常见的是 h 与 r 都比较简单,可以将方程向右整理,令 h 为 1 即可。

（2）诺依曼（Neumann）边界条件，也称第二类边界条件，定义了场在边界处的导数（变化率），数学形式为

$$\frac{\partial}{\partial \boldsymbol{n}}(c\nabla u)+qu=g$$

其中，q 与 g 是关于 $\boldsymbol{x}$，t，u，$\partial u/\partial \boldsymbol{x}$ 的函数，$\partial u/\partial \boldsymbol{n}$ 表示 $\boldsymbol{x}$ 向量法向的偏导数。

如果是求解偏微分方程组的话，当然也不排除其中有不同的方程使用不同类型的边界条件，这时称为“混合边界条件”。

本例设置最简单的狄利克雷边界条件，让场量在边界处均为 0，如图 5-27 所示。

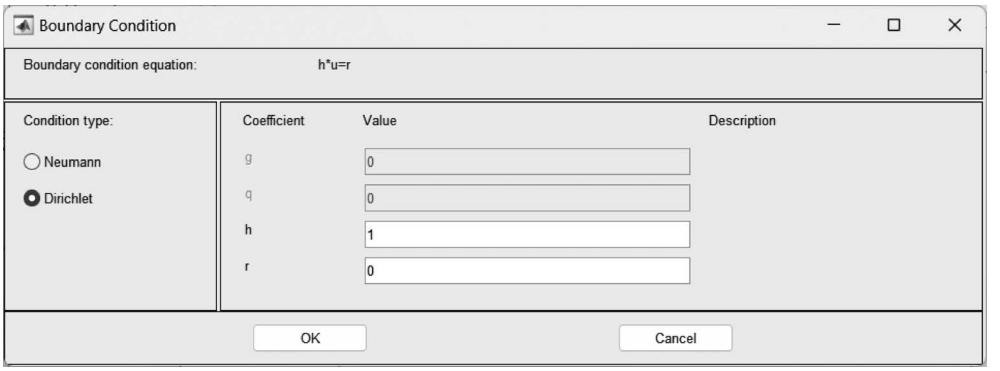


图 5-27　边界条件对话框

不同的边界条件由不同的颜色区分，狄利克雷边界条件为红色，诺依曼边界条件为蓝色。

对于本例的设置，对应代码为

```
applyBoundaryCondition(model,'dirichlet','Edge',1:model.Geometry.NumEdges,'u',0)
```

5. 设置偏微分方程

选择菜单栏中的偏微分方程（PDE），选择设置偏微分方程（PDE Specification），进入 PDE 的设置界面，本例选择椭圆型方程（Elliptic），设置 c=1，a=0，f=10，这也是软件的默认参数，如图 5-28 所示。

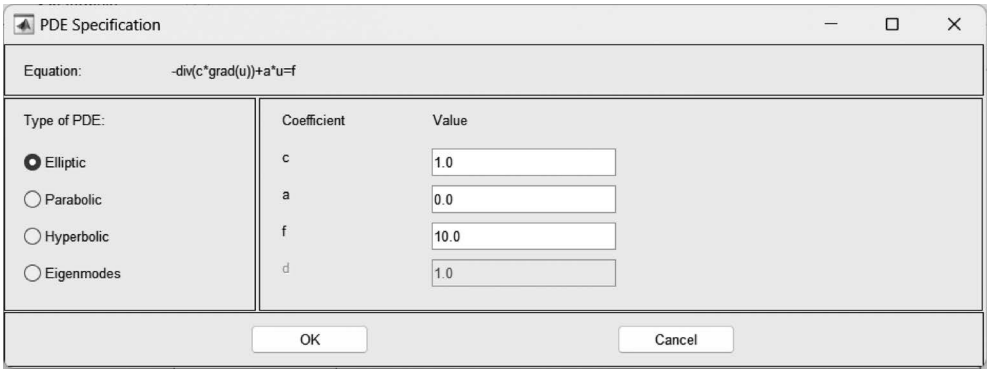


图 5-28　偏微分方程设置

如图 5-28 所示的界面上方已将方程的形式列出，可以参考对应位置的字母，方便设置。对应代码为

```
specifyCoefficients(model,'m',0,'d',0,'c',1,'a',0,'f',1);
```

6. 设置网格

单击网格(Mesh)菜单栏，进入网格模式(Mesh mode)可以看到软件自动为区域划分了三角形网格，可以对网格进行一些更详细的设置和处理，比如细化网格(Refine mesh)，细化前后的网格如图 5-29 所示。

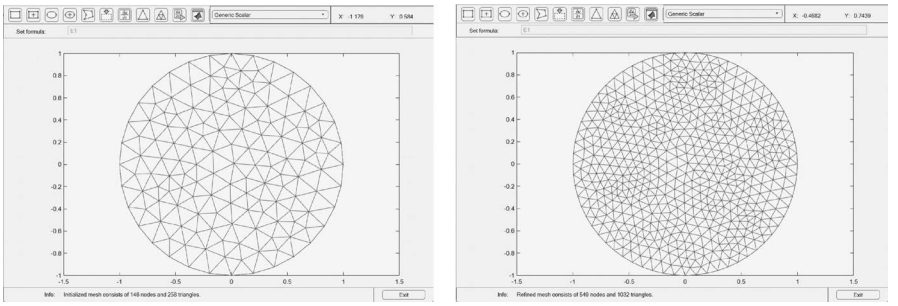


图 5-29　网格细化前后对比

设置最大网格尺寸为 0.1，对应代码为

```
generateMesh(model,'Hmax', 0.1);
```

如需显示网格划分的情况，可以使用如下代码对网格作图：

```
pdemesh(model);
```

7. 求解方程

单击求解(Solve)菜单栏下的求解偏微分方程(Solve PDE)，默认求解结果如图 5-30 所示。

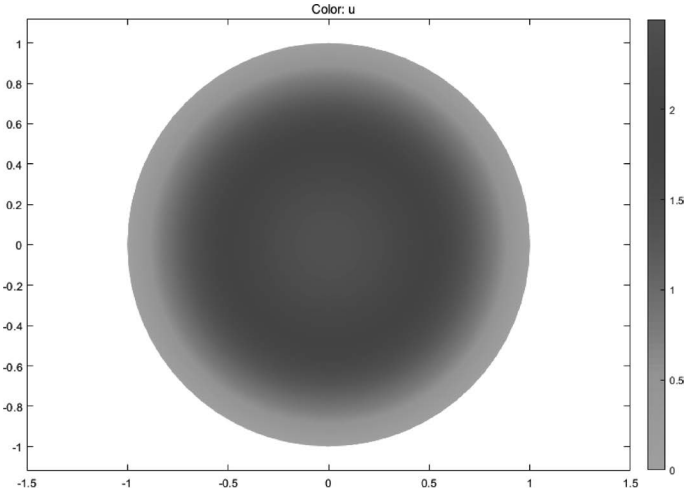


图 5-30　方程解的云图

如图 5-30 所示，边缘上确实如边界条件设置的那样均为零。对应的代码如下，注意求解的直接结果为一个结构体：

```
results = solvepde(model);
u = results.NodalSolution;
```

如需绘图，使用 pdeplot()函数，代码如下：

```
pdeplot(model,'XYData',u)
```

8. 后处理作图

自动生成的图像往往不满足要求，这时可以单击作图选项(Plot selection)，在界面上有多种功能可选，而且可以对作图的对象进行选择甚至输入，如图 5-31 所示为选择使用箭头标注场量的梯度方向，并使用 jet 作为配色方案，同时勾选了三维绘图选项。

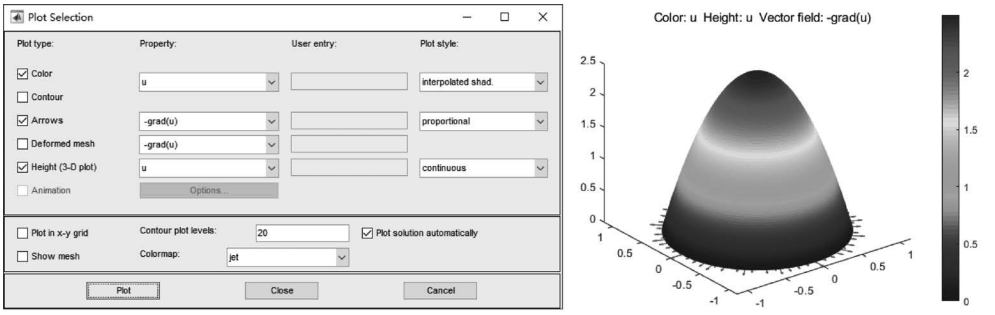


图 5-31　多维数组/矩阵 1

PDET 既然可以解偏微分方程，就可以解决许多物理问题，比如热传递问题，如图 5-32 所示。

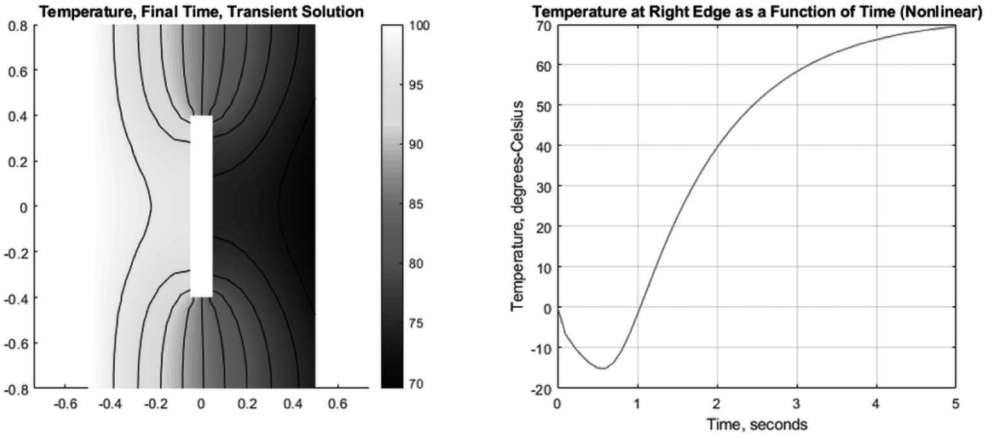


图 5-32　多维数组/矩阵 2

还可以导入三维的 STL 模型，实现三维空间分析，如图 5-33 所示为一个支架结构的变形分析。

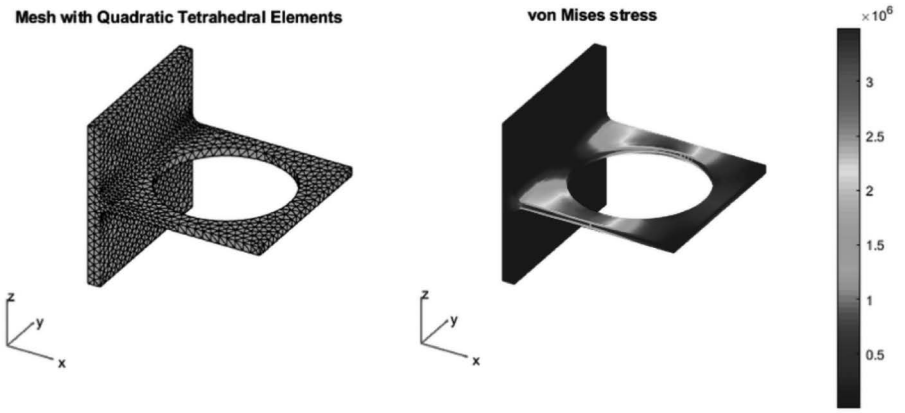


图 5-33　多维数组/矩阵 3

PDET 实质上是一个专为有限元分析而设计的仿真软件。在本质上，单一物理场问题可以归结为偏微分方程的求解，而多物理场耦合问题则涉及偏微分方程组的处理。作为多物理场有限元仿真的先驱，COMSOL Multiphysics 实际上是 MATLAB PDET 的一个高度发展和演化形式，因此 COMSOL 不仅继承了求解偏微分方程的数学功能，还扩展了这些能力。如果遇到一些 MATLAB 难以攻克的偏微分方程挑战，可以转向使用 COMSOL 软件进行求解。如图 5-34 所示，COMSOL 软件提供了一个数学接口，该接口能解决的偏微分方程类型比 MATLAB 更加丰富多样。在物理数学方程的数值求解方面，特别是在二维平面和三维空间问题上，更推荐使用像 COMSOL 这类专业的有限元软件。

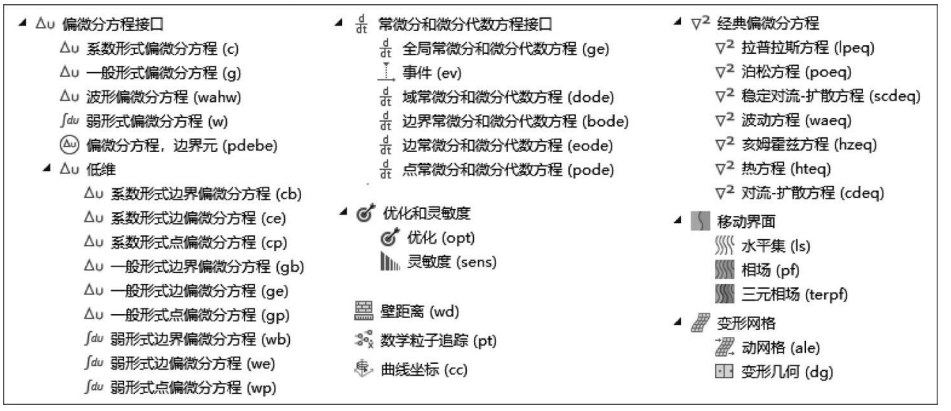


图 5-34　COMSOL 软件中提供的数学工具模块

5.7　概率统计

统计学(Statistics)主要有两大方面的内容：统计描述和统计推断。统计描述，用数字、函数、图像等描述一个概率分布，这个分布可能是总体的，也可能是样本的，用数字来描述分

布的，称为分布度量。统计推断，是用来解释“样本分布”与“总体分布”之间的关系的，它分为两大问题：参数估计(Parameter Estimation)和假设检验(Hypothesis Test)。

5.7.1 概率分布

连续随机变量的概率分布函数 $F(x)$，表示随机变量 ξ 满足 $\xi<x$ 的概率，即 $F(x)=P(\xi<x)$，因此也被更形象地称为累积分布函数(Cumulative Distribution Function，CDF)；由于它可以通过简单减法得到随机变量落入任何范围内的概率，所以 CDF 是最为易用的概率描述函数。而概率密度函数(Probability Density Function，PDF)用于描述随机变量的输出值在某个取值点附近的可能性，一般记为 $p(x)$，两者之间存在简单的积分关系：

$$F(x)=\int_{-\infty}^{x} p(t)\,\mathrm{d}t$$

MATLAB 提供了强大的统计和机器学习工具箱(Statistics and Machine Learning Toolbox，SMLT)，其中关于概率统计的相关函数均有涉及，如图 5-35 所示绘制了最典型的离散和连续概率分布——泊松分布和正态分布。

EX 5-23 概率分布与概率密度

1. 泊松分布（离散）

```
x=0:20; lamda=10;
pdf_P=pdf('Poisson',x,lamda);
cdf_P=cdf('Poisson',x,lamda);
ax(1)=subplot(2,2,1);stem(x,pdf_P),line(x,pdf_P);
title('Poisson PDF');
ax(2)=subplot(2,2,2);plot(x,cdf_P);
title('Poisson CDF');
```

2. 正态分布（连续）

```
x=-5:0.02:5; mu=0; sigma=sqrt(1);
pdf_N=pdf('Normal',x,mu,sigma);
cdf_N=cdf('Normal',x,mu,sigma);
ax(3)=subplot(2,2,3);plot(x,pdf_N);
title('Normal PDF');
ax(4)=subplot(2,2,4);plot(x,cdf_N);
title('Normal CDF');
axis(ax,"square")
```

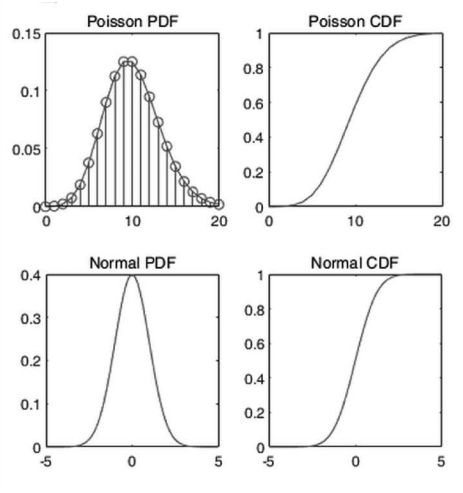


图 5-35 概率分布与概率密度例程

说明：

(1) 概率密度函数 pdf()与概率分布函数 cdf()的输入格式非常清晰：

```
pdf('name',x,A,B,C,D)
cdf('name',x,A,B,C,D)
```

其中，name 表示分布的名称，A～D 即为对应分布的参数设置，输出向量与输入的向量 x 长度一致，因此可以直接作图。

(2) MATLAB 共提供 33 种概率分布选项，基本上兼容了所有可见的概率分布，最常用的 3 种离散型及 8 种连续型分布如表 5-3 所示，如需查看所有分布选项，可以在帮助文档中

搜索 pdf 或 cdf。

表 5-3　最常用的 3 种离散型及 8 种连续型分布

类　别	分　布	字　段	参数 A	参数 B
离散型	二项分布	'Binomial'	n(试验次数)	p(单次成功率)
	泊松分布	'Poisson'	λ(均值)	—
	几何分布	'Geometric'	p(单次成功率)	—
连续型	均匀分布	'Uniform'	a(下界)	b(上界)
	指数分布	'Exponential'	μ(均值)	—
	正态分布	'Normal'	μ(均值)	σ(标准差)
	瑞利分布	'Rayleigh'	b(规模参数)	—
	卡方分布	'Chisquare'	v(自由度)	—
	伽马分布	'Gamma'	a(形状参数)	λ(规模参数)
	学生分布	'T'	v(自由度)	—
	贝塔分布	'Beta'	a(形状参数)	b(形状参数)

其中，均匀分布也有离散型，即“等概率模型”(古典概型)，由于比较简单因此并没有对应函数。

如果需要快速地可视化概率分布与概率密度曲线，MATLAB 提供了一个概率分布函数 App(Probability Distribution Function App)，相当于一个方便快捷的小软件，打开方式是在命令行中输入 disttool 并按下 Enter 键即可。如图 5-36 所示为正态分布的 CDF 与 PDF 图像，界面上可以任意选择分布并输入参数，还可以使用鼠标交互式地捕捉坐标位置。

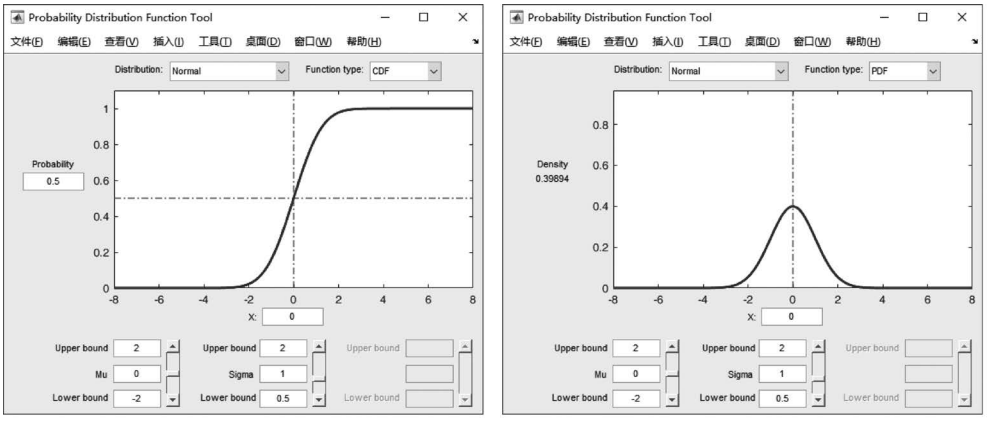


图 5-36　概率分布函数 App

5.7.2　伪随机数的生成与应用

在科学研究和统计分析活动中经常要用到随机数据，然而由计算机生成的随机数据，是通过算法并按照给定的分布规律计算出来的，称为“伪随机数”。

在 MATLAB 核心函数集中，提供了三个最常用的随机数生成函数，分别为产生均匀分

布随机数的函数 rand()、产生均匀分布随机整数的函数 randi()、产生标准正态分布随机数的函数 randn()。在统计和机器学习工具箱(SMLT)中还提供了更为通用和强大的随机数产生函数 random(),它可以按任意概率分布规律产生随机数,其代码形式为

```
R = random('name',A,B,C,D)
R = random('name',A,B,C,D,[size])
```

可见如上代码与前述 pdf()和 cdf()函数所能处理的概率分布类型及分布参数输入形式完全一致,后跟向量为输出矩阵的尺寸规模。利用随机数产生函数绘制一些直方图如图 5-37 所示。

EX 5-24 伪随机数

1. 均匀分布和正态分布

```
ax(1) = subplot(2,2,1);
histogram(rand(1e4,1),20);
ax(2) = subplot(2,2,2);
histogram(randn(1e4,1),20);
```

2. 指数分布和卡方分布

```
ax(1) = subplot(2,2,3);
histogram(random('Exponential', 2, [1e4 1]),20);
ax(2) = subplot(2,2,4);
histogram(random('Chisquare', 2, [1e4 1]),20);
axis(ax,'square')
```

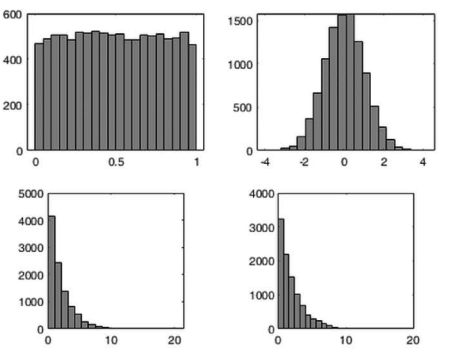


图 5-37　伪随机数例程

说明：1e4 为科学记数法,等价于 10000,这里设置目的是产生一个由随机数组成的特定尺寸的向量。

5.7.3　统计量分析：数据的解码器

对于一组数据或一个分布,用一些数字量来作为某些整体性质的度量,这些量称为统计量,比如均值(mean)、方差(var)、标准差(std)、原点矩(无对应函数)、中心矩(moment)等,如图 5-38 所示为一个实例,其中构造了一个概率分布对象,并使用圆点表示法提取了参数,可以理解为解析解,是精确的统计量；而通过随机量计算的统计量一定会存在偏差,当随机量数目越多时,偏差会有变小的趋势。

说明：

(1) 均值函数 mean()、方差函数 var()、标准差函数 std()的输入参数不仅可以是向量,还可以是矩阵,这时相当于对矩阵中所有的列向量进行计算,因而输出的是一个行向量,如果需要对矩阵中所有元素进行计算,则可以采用简单的代码形式,如下：

```
mean(x(:))
```

(2) 函数 makedist()用于创建概率分布对象,该分布对象的属性中包含特定的分布参数,如 mu 和 sigma 等,还包含一些方法(函数),也可以直接使用圆点表示法调用,比如 mean、var 及 std 等,与前述对于数据的分析函数完全一致。

EX 5-25 统计量分析

1. 均值、方差、标准差

```
% 构造概率分布随机数
x = random('Normal',2,4,[1e4,1]);
meanX = mean(x) % 均值
s2X = var(x) % 方差
sX = std(x) % 标准差
% 构造概率分布对象
pd = makedist('Normal','mu',2,'sigma',4);
meanX2 = pd.mean % 均值
meanX2_mu = pd.mu % 均值
s2X2 = pd.var % 均值
sX2 = pd.std % 标准差
sX2_sigma = pd.sigma % 标准差
```

```
meanX = 2.0023
s2X = 15.8531
sX = 3.9816


meanX2 = 2
meanX2_mu = 2
s2X2 = 16
sX2 = 4
sX2_sigma = 4
```

2. 原点矩与中心矩

```
A1 = sum(x.^1)/length(x) % 一阶原点矩
A2 = sum(x.^2)/length(x) % 二阶原点矩
B1 = moment(x,1) % 一阶中心矩
B2 = moment(x,2) % 二阶中心矩
```

```
A1 = 2.0023
A2 = 19.8608
B1 = 0
B2 = 15.8515
```

图 5-38　统计量分析例程

(3) MATLAB 没有提供原点矩计算函数，但根据定义可知 k 阶原点矩的代码为

```
Ak = sum(x.^k)/length(x)
```

5.7.4　参数估计：统计的预言家

有时需要根据一组数据(样本数据)的基本形态做出推测，它应该是满足某分布规律的，比如正态分布规律，因此有理由推定总体数据也是满足正态分布规律的，那么这个规律表示成函数形式具体是什么样的，或者说分布函数里的参数应该是多少呢？这个计算过程就是参数估计(Parameter Estimation)。

本书观点认为“参数估计”这个名称略有晦涩，其实际含义比较接近于概率分布函数的拟合(Probability Distribution Fit)，只不过有一点在概念上需要注意，参数估计是用样本数据来估计总体的函数，因此样本数据的个数也是决定算法结果的关键。

对参数的估计，并不是只得到一个值，而是得到一个估计值加一个估计值的变化范围，称为置信区间(Confidence Interval，CI)，与这个区间相对应的可信度称为置信水平(Confidence Level)，置信水平越接近 1，则置信区间就会越小，一般最常用的置信水平是 95%，因此 MATLAB 工具箱中提供的通用参数估计函数 fitdist()即默认为 95%的置信水平，工具箱还提供一些对应于特定分布的参数估计函数，比如正态分布估计函数 normfit()、泊松分布估计函数 poissfit()、均匀分布估计函数 unifit()等，如图 5-39 所示，这些函数可以对置信水平进行设置，得到不同的置信区间。

EX 5-26 参数估计

```
% 构造概率分布随机数
x = random('Normal',2,4,[1e4,1]);
% 方法1
pd = fitdist(x,'Normal')
% 方法2
[muHat,sigmaHat,muCI,sigmaCI] = normfit(x,0.05)
```

```
pd =
  Normal distribution
       mu = 2.01604   [1.93761, 2.09448]
    sigma = 4.00123   [3.94654, 4.05747]

muHat = 2.0160     sigmaHat = 4.0012
muCI = 2×1         sigmaCI = 2×1
    1.9376             3.9465
    2.0945             4.0575
```

图 5-39　参数估计例程

说明：

（1）通用参数估计函数 fitdist()的常用代码形式为

```
pd = fitdist(x,distname)
```

（2）normfit()函数的第二个参数意为显著性水平，显著性水平与置信水平的和为 1。

对于参数估计计算，MATLAB 还提供了专用 App——参数估计器（Distribution Fitter），打开方式是在命令行中输入 distributionFitter 并按下 Enter 键即可，其界面友好、功能强大，如图 5-40 所示为本节例程中数据 x 的估计结果 PDF 及 CDF。

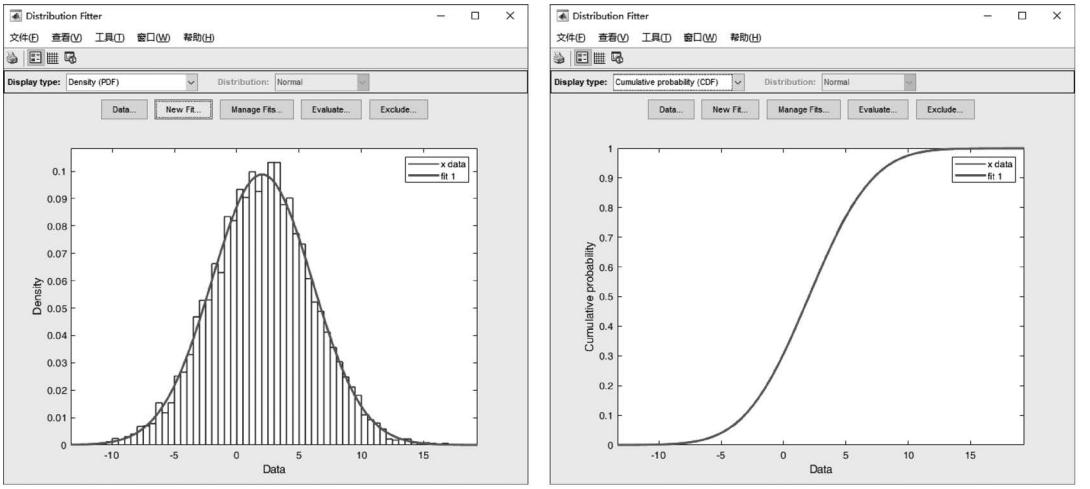


图 5-40　参数估计器 App

5.7.5　假设检验：验证数据的真相

如果一个射击运动员说："我射击的平均成绩是 8 环"，但是对于分析者来说，他说的话不知真假，只能称之为"假设"，如何证明这个假设呢，可以根据"大数定律"让他射击很多很多次取均值来判定，但实际情况很可能是样本数量并不足够多，这时就需要特定的检验算法，这也就是假设检验的过程。

假设检验（Hypothesis Test）先假设总体分布的参数，再用样本来检验这个参数的可信

度(置信水平),是一种持怀疑态度的基于反证法思想的验证,实例如图 5-41 所示。

EX 5-27 假设检验

```
% 构造概率分布随机数
x = random('Normal',2,4,[1e4,1]);
sX = std(x); % 标准差
% Z检验-已知标准差
[hZ,pZ,ciZ,~]= ztest(x,2,sX)
% T检验-不知标准差
[hT,pT,ciT,~] = ttest(x,2)
```

```
hZ = 0
pZ = 0.2547
ciZ = 2×1
    1.8768
    2.0326

hT = 0
pT = 0.2548
ciT = 2×1
    1.8768
    2.0327
```

图 5-41　假设检验例程

说明：对于疑似符合正态分布的数据,当已知标准差时,可以采用 Z 检验,函数为 ztest(),若标准差是未知的,可以使用 T 检验。h 为检验结论,为零时表明不拒绝假设,也就是接受假设；p 为该检验的显著性水平；ci 为置信区间。

常见问题解答

(1) MATLAB 在高等数学的学习中会不会起到辅助作用?

MATLAB 在高等数学的学习中扮演了十分重要的角色。它为我们提供了一个实验平台,将抽象的高等数学问题转换为具体的函数和代码,并通过图形展示,极大地促进了我们对高等数学概念和原理的理解。强烈建议想深入掌握高等数学思想和方法的读者,积极利用 MATLAB 作为学习辅助。

(2) 为什么要强调插值、拟合和优化这些概念?

插值、拟合和优化是数学应用中至关重要的概念。插值允许我们在已知数据点之间准确预测未知值,从而在图像处理、气象预测等领域实现数据的连续性和平滑性；拟合帮助我们通过实验或观测数据建立数学模型,从而在统计回归、生物学等学科中揭示和理解潜在规律；优化则是寻找最有效率和最经济的解决方案的过程,它在机器学习参数调整、成本削减、系统设计改进等多个领域中至关重要。这三种数学工具是理解复杂现象、提高决策质量和推动技术进步的关键。

(3) 为什么说微分方程的求解意义重大?

微分方程的求解意义重大,主要因为微分方程是描述自然界和工程技术中各种现象和过程的数学模型。无论是物理学中的运动定律,化学反应的速率,生物种群的增长,还是经济学中模型的构建,许多自然规律和科学问题都可以用微分方程来表达。通过求解微分方程,我们能够预测系统的未来行为,理解不同变量之间的相互关系,以及控制或优化现实世界中的过程。微分方程的求解意义包括但不限于以下几点：理解系统动态、揭示因果关系、指导实验和设计、优化和控制。

本章精华总结

本章精心梳理了 MATLAB 在数学领域的广泛应用，涵盖了初等数学、线性代数、微积分、插值与拟合、代数方程与优化、微分方程以及概率统计这七大核心主题。更重要的是，本章强调了深刻理解这些数学概念的重要性。通过本章的学习，读者将能够把数学知识从简单记忆转变为深刻理解和实际应用，这不仅丰富了读者对数学的认识，也为解决实际问题提供了强大的工具。