第10章　函数、插值和曲线拟合分析

MATLAB可以处理有估值和没有估值的多项式，还有一些强大的数值分析命令，如求零值和最小值。MATLAB中还有数据集合的插值、曲线拟合的命令和函数，还提到了经典的贝赛尔(Bessel)函数。

10.1　MATLAB中的多项式

MATLAB将阶为n的多项式$p(x)$存储在长度为$n+1$的行向量**p**中。元素为多项式的系数，并按x的幂降序排列，表示为：

$$\mathbf{p} = (a_n \quad a_{n-1} \quad \dots \quad a_1 \quad a_0)$$

代表的多项式为：

$$p(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$$

令**A**是一个稀疏矩阵，向量**p**和**q**的长度分别为$n+1$和$m+1$，它们分别表示次数为n和m的多项式。MATLAB中处理多项式的命令如下：

命令集99　多项式

`polyval(p,x)`	计算多项式**p**。如果x是一个标量，则计算出多项式在x点的值；如果**x**是一个向量或者一个矩阵，则计算出多项式在**x**中所有元素上的值。
`[y,err]=polyval(p,x,E)`	计算向量**x**的多项式**p**的值。同上，计算结果在**y**中，同时还根据`polyfit`命令给出的矩阵**E**返回一个误差估计向量**err**。见`help polyval`和`help polyfit`，参见10.4节。
`polyvalm(p,A)`	直接对矩阵**A**进行多项式计算。不是象上个命令一样对每个元素进行多项式计算，而是计算$p(\mathbf{A})=p_1\mathbf{A}^n+p_2\mathbf{A}^{n-1}+\cdots+p_{n+1}\mathbf{I}$。
`poly(A)`	计算矩阵**A**的特征多项式向量。
`poly(x)`	给出一个长度为$n+1$的向量，其中的元素是次数为n的多项式的系数。这个多项式的根是长度为n的向量**x**中元素。
`compan(p)`	计算带有系数p的多项式的友矩阵**A**，这个矩阵的特征多项式为**p**。
`roots(p)`	计算特征多项式**p**的根，是一个长度为n的向量，也就是方程$p(x)=0$的解。表达式`poly(roots(p))=p`为真。结果可以是复数。
`conv(p,q)`	计算多项式**p**和**q**的乘积，也可以认为是**p**和**q**的卷积。
`[k,r]=deconv(p,q)`	计算多项式**p**除**q**。**k**是商多项式，**r**是残数多项式。这个计算等价于**p**和**q**的逆卷积。

`[u v k]=residue(p,q)`	计算$p(x)/q(x)$的部分展开式：$\frac{p(x)}{q(x)}=\frac{u(1)}{x-n(1)}+\frac{u(2)}{x-n(2)}+\cdots+\frac{u(j)}{x-n(j)}+k(x).$ 向量**p**和**q**分别是多项式$p(x)$和$q(x)$的系数。得到的余数在向量**u**中，极点在列向量**v**中，商多项式在向量**k**中。
`[p q]=residue(u,v,x)`	从同上的部分展开式**u**、**v**和**x**计算得到多项式**p**和**q**。
`mpoles`	给出极点多样性的相关信息，见`help mpoles`。
`polyder(p)`	计算得到长度为n的微分多项式向量，多项式的系数在向量**p**中。
`polyder(p,q)`	返回一个向量，它表示由`conv(p,q)`定义的多项式微分。
`[u,v]=polyder(p,q)`	返回两个向量，它们表示由`deconv(p,q)`定义的多项式微分，表达形式为**u**/**v**。

■ **例**10.1

将一些多项式命令应用在下列的多项式上：

$p2(x)=3x^2+2x-4$　　$p3(x)=2x^3-2$

MATLAB用下列向量来表示这两个多项式：

```
p2 = [3 2 -4];
p3 = [2 0 0 -2];
```

(a) 计算多项式在x=1处的值，输入：

```
value2 = polyval(p2,1), ...
value3 = polyval(p3,1)
```

结果为：

```
value2 =
     1

value3 =
     0
```

(b) 很容易对向量或者矩阵计算多项式的值，输入：

```
x = [1 2 3]';
values2 = polyval(p2,x), values3 = polyval(p3,x)
```

结果为：

```
values2 =
     1
    12
    29

values3 =
     0
    14
    52
```

(c) 两个多项式相乘，得到一个新的多项式：

```
p5=conv(p2, p3)
```

给出：

```
p5 =
     6     4    -8    -6    -4     8
```

(d) 用roots命令求多项式的根：

```
roots2=roots(p2), roots3=roots(p3)
```

给出：

```
roots2 =
   -1.5352
    0.8685

roots3 =
  -0.5000 + 0.8660i
  -0.5000 - 0.8660i
   1.0000
```

图10-1中显示出两个多项式的图形。

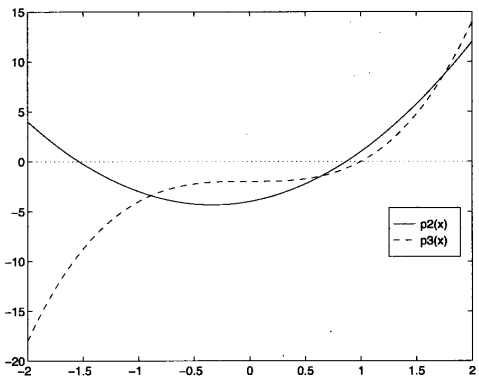

图10-1　多项式$p2(x)=3x^2+2x-4$和$p3(x)=2x^3-2$

(e) 多项式$p(x)$的牛顿-拉普森迭代，多项式的系数在向量**p**中：

```
q = polyder(p);
xnext = x - polyval(p,x)/polyval(q,x);
```

(f) 命令roots(poly(A))求得矩阵**A**的特征值。假设矩阵**A**为：

$$\mathbf{A}=\begin{pmatrix}-9 & -3 & -16\\ 13 & 7 & 16\\ 3 & 3 & 10\end{pmatrix}$$

运行命令：

```
usedRoots=roots(poly(A))
```

结果为：

```
usedRoots =
   10.0000
    4.0000
   -6.0000
```

然而这样得到的特征值没有用 MATLAB命令eig(A)得到的特征值的精度高，而且有效性也差些：

```
usedEig=eig(A)
```

给出：

```
usedEig =
   -6.0000
   10.0000
    4.0000
```

结果是一样的，但是顺序正好相反。

(g) 对于所有的矩阵**A**都有：polyvalm(poly(A), A)=0

这是Cayley-Hamilton法则。这个法则对于秩为5的方阵来说：

```
Magical = magic(5);
AlmostZero = polyvalm(poly(Magical),Magical)

AlmostZero =
   1.0e-07 *
    0.2794    0.3551    0.1723    0.1770    0.2654
    0.2765    0.2887    0.2049    0.2142    0.2561
    0.1775    0.2468    0.2701    0.3073    0.2375
    0.1942    0.2744    0.2759    0.2608    0.2282
    0.2082    0.3120    0.2608    0.2515    0.2049
```

■

10.2 函数的零值

MATLAB的M文件可以表示数学函数；参见2.9节。函数：

$$g(x)=\frac{5x-6.4}{(x-1.3)^2+0.002}+\frac{9x}{x^3+0.03}-\frac{x-0.4}{(x-0.92)^2+0.005}$$

如果输入下面的M文件g.m，这个函数就可以在MATLAB中调用：

```
function y = g(x)

y = (5.*x-6.4)./((x-1.3).^2+0.002) + ...
    (9.*x)./(x.^3+0.03) - ...
    (x-0.4)./((x-0.92).^2+0.005);
```

使用元素运算符.*、./、.^、+和－定义MATLAB函数**g**。结果是如果这个函数被一个向量调用，那么得到的结果也是一个向量。本章中提到的所有MATLAB函数需要以这种方式来定义数学函数。

用plot命令可以画出函数的图形：

```
x=linspace(0, 2);              % 生成向量x
plot(x,g(x));                  % 画g(x)图形
grid;                          % 画格栅
title('The g(x) function')     % 给出图标题
```

或者使用fplot命令：

```
fplot('g', [0 2]);             % 画g(x)图形
grid;                          % 画格栅
title('The g(x) function')     % 给出图标题
```

结果如图10-2所示。命令plot和fplot都定义在13.1 节中。

求函数$f(x)$的零值就等于求方程 $f(x)$=0的解。单变量函数的零值可以用 MATLAB命令fzero来求。对于多项式可以用 roots命令来求，参见10.1节。fzero用迭代法来求解，使得初始的估计值接近理想的零值。

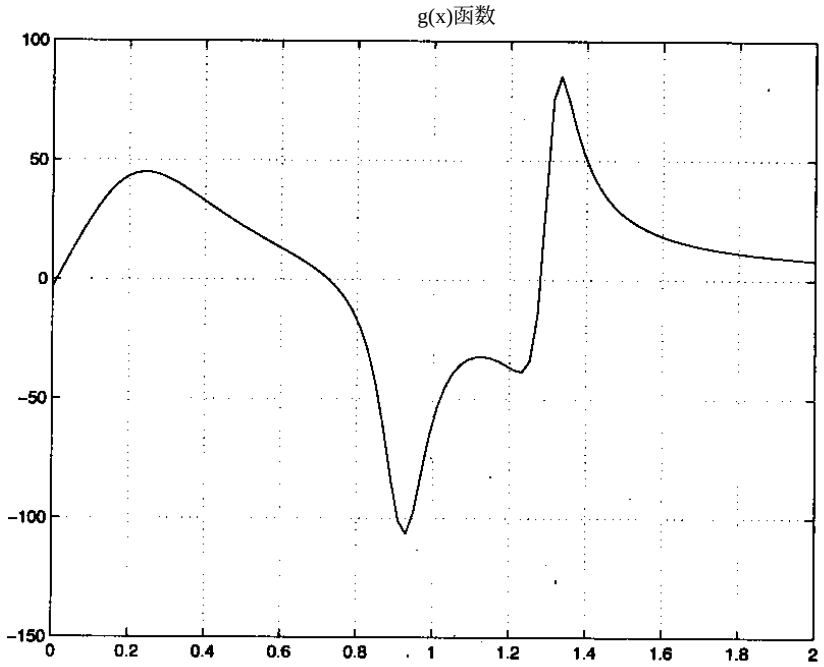

图10-2 用fplot 画的$g(x)$图形

命令集100 函数的零值

fzero(fcn,x0)	求函数的一个零值， **fcn**为函数的名字。要求给出一个初始值x0，近似值的相对误差为eps。
fzero(fcn,x0,tol)	求函数的一个零值， **fcn**为函数的名字。要求给出一个初始值x0，由用户定义近似值的相对误差 *tol*。
fzero(fcn,x0,tol,pic)	求函数的一个零值，同上。如果pic不为零，则给出迭代过程。
fzero(fcn,x0,tol, pic p1 p2,...)	求多变量函数的零值，**fcn=fcn**(x0, p1, p2,…)。如果tol和pic没有给出，则令它们为空矩阵，如 fzero(fcn,x0,[],[],p1)。

zerodemo命令给出了一个演示实列。

■ **例**10.2

(a) 求本节开头定义的函数$g(x)$的零值：

```
x1 = fzero('g',0), x2 = fzero('g',0.5), x3 = fzero('g',2)
```

结果为：

```
x1 =
    0.0112

x2 =
    0.7248

x3 =
    1.2805
```

(b) 求函数sin*x*和2*x*－2的交集，也就是求方程 sin *x*=2*x*-2的解。先定义函数*sinm(x)*，将它存放在M文件sinm.m中，如下：

```
function s = sinm(x)


s = sin(x) - 2.*x + 2;
```

画出曲线是找到初始值的一个好方法，所以：

```
fplot('sinm',[-10 10]);
grid on;
title('The sin(x) - 2.*x + 2 function');
```

结果如图10-3所示。可以看出2是一个可接受的估计值，输入：

```
xzero=fzero('sinm', 2)
```

结果为：

```
xzero =
   1.4987
```

这就是方程sin*x*=2*x*－2的解。

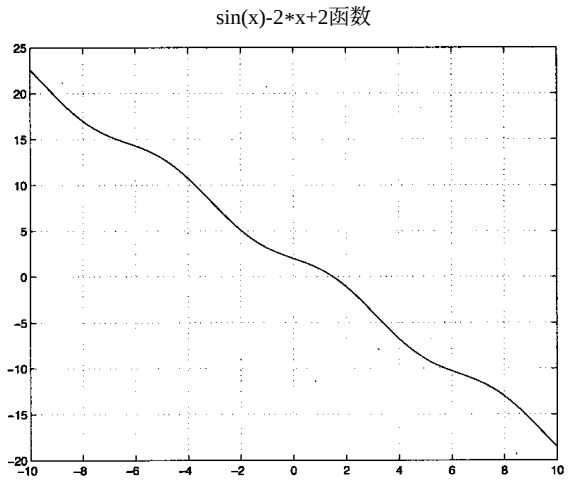

图10-3 *sinm(x)*函数

10.3 函数的最小值和最大值

最优化是求最优解，也就是在某个区间内有条件约束或者无条件约束地找到函数的最大值或者最小值。MATLAB使用数字方法求函数的最小值。使用迭代算法，也就是有些步骤要重复许多次。现在，假设要求函数 f 在某个区间内的最小值 x_{min}。

$$f(x_{\min}) = \min_x f(x)$$

迭代方法需要一个初始估计值 x_0。从 x_0 开始找到一个更接近 x_{min} 的新值 x_1，这个值的好坏取决于使用的数学方法。直到找到有足够精度的近似值 x_i 才停止迭代，也就是绝对值 $|x_{min}-x_i|$ 足够小。

如果函数有多个局部最小值，`fmin`可以找到它们中的一个。也可用 MATLAB的最优化工具箱来求得，参见附录C。

这里提到了标准 MATLAB系统的两个最优化命令，`fmin`命令可以求单变量函数的最小值；`fmins`命令可以求多变量函数的最小值，同时它还要求有一个初始向量。

没有求函数 f 的最大值的命令，相反函数 $h=-f$ 的最小值可以求得。

命令集101 函数的最小值

`fmin(fcn,x1,x2)`	求函数在区间($x1$，$x2$)内的最小值，**fcn**是目标函数名。如果没有局部最小值，则返回区间内的最小 x 值。相对误差小于10^{-4}。
`fmin(fcn,x1,x2, options)`	求函数在区间($x1$，$x2$)内的最小值，**fcn**是目标函数名。如果没有局部最小值，则返回区间内的最小 x 值。向量**options**为控制参数，如**options**(1)=1，显示中间结果；**options**(2)表示得到的结果 x 的精度，缺省为 10^{-4}。输入`help foptions`可得更多信息。
`fmins(fcn,x0)`	求函数**fcn**的最小值。由用户自己给出一个初始估计向量**x0**，相对误差为10^{-4}。
`fmins(fcn,x0, options)`	带优化参数求函数**fcn**的最小值，同上。输入`help fmins` 和`help foptions`可得更多信息。例如，优化参数可以控制迭代次数和计算结果的精度。

■ **例10.3**

(a) 在区间[0，2π]内求函数cos的最小值：

```
cosmin=fmin('cos', 0,*pi)    %求cos的最小值
cosmin=
       3.1416
```

这就是期望得到的结果。

(b) 同样可以简单地求高级函数的最小值。对定义在 10.2节中的函数 $g(x)$，求在区间[0，2]内的最小值。

```
gmin = fmin('g',0,2)

gmin =
    1.2277
```

注意，这是一个局部最小值，不一定是函数在这个区间内的最小值。从图 10-2上可以看出在一个更小区间内可以得到第二个最小值，这个值比第一个值还小：

```
gmin2 = fmin('g',0,1)

gmin2 =
    0.9260
```

(c) 还可以用fmin命令来求函数的最大值，但是要先编写一个返回－ $g(x)$的函数，这个函数保存在M文件minusg.m中。

```
function y = minusg(x)

y = -g(x);
```

求这个函数的最小值就等于求函数 g的最大值。

```
gmax=fmin('minusg', 0, 2)
```

结果为：

```
gmax =
    0.2433
```

在这个区间内有若干个最大值， MATLAB求出的最大值不一定是函数的全局最大值。

(d) 用fmins命令来求多个变量函数的最小值，假设函数为：

$$f(x_1, x_2) = x_1^2 + x_2^2 - 0.5x_1x_2 - \sin x_1$$

编写M文件fx1x2.m：

```
function f = fx1x2(x)

f = x(1).^2 + x(2).^2 - 0.5.*x(1).*x(2) - sin(x(1));
```

函数fmins要求有一个初始估计向量，假设给 (1，0)：

```
fx1x2min = fmins('fx1x2',[1,0])
```

结果为：

```
fx1x2min =
    0.4744    0.1186
```

用下面的程序画出函数的图形来：

```
x=linspace(－1, 1 50);     % 新建向量x，假设y=x
for i=1: 50                % 计算fx1x2在每一点的值
     for j=1: 50
       Z(i, j)=fx1x2([x(i)  x(j)]);
     end
  end
meshc(x ,x, Z);            % 带有基本等值线的网格图
view(80, 10);              % 指定观察点
```

命令meshc画出函数的表面图形，同时在*xy*平面画出图形的等值线。命令meshc和view定义在13.5节中，在4.2节中提到了命令linspace。结果如图10-4所示，从图上可以看出最小值。

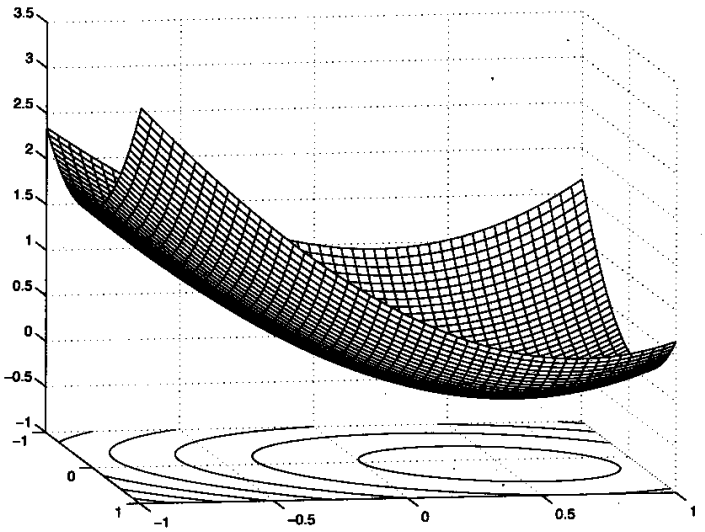

图10-4 函数$x_1^2+x_2^2-0.5x_1x_2-\sin x_1$在区间[－1，1]×[－1，1]上的图形

■

10.4 插值、曲线拟合和曲面拟合

如果在有限个数据点内给出函数，那么利用插值的方法就可以找到中间点的近似值。最简单的插值就是对两个相邻数据点进行线性插值。 interp1和interp2命令用特殊的算法来进行等距离数据点的快速插值。使用时，必须在方法的名字前加上一个星号，'*'，如interp1(x, Y, xx ,'*cubic') 。

MATLAB中有几个函数可以用不同的方法来进行数据插值。

命令集102 插值

interp1(x,y,xx)	返回一个长度和向量**xx**相同的向量**f(xx)**。函数**f**由向量**x**和**y**定义，形式为**y=f(x)**，用线性插值的方法来计算值。为了得到正确的结果，向量**x**必须按升序或降序排列。
interp1(x,Y,xx)	返回一个相应向量的矩阵**F(xx)**，同上。矩阵**Y**的每列是一个关于**x**的函数，对于每个这样的函数**xx**的值通过插值得到。矩阵**F(xx)**的行数和向量**xx**的长度相同，列数和矩阵**Y**的相同。
interp1(x,y,xx, metstr)	进行一维插值，字符串**metstr**规定不同的插值方法，可用的方法有：

	‘linear’ 线性插值。 ‘nearest’ 最邻近插值。 ‘spline’ 三次样条插值。也叫外推法。 ‘cubic’ 三次插值，要求x的值等距离。 所有插值方法均要求x是单调的。
interp1q(x,y,xx)	和interp1相同，但是对于非均匀空间的数据插值更快。
interp2(X,Y,Z,Xx,Yy)	进行矩阵**Xx**和**Yy**的二维插值，并由**X**、**Y**和**Z**所描述的函数**Z**=**f**(*X*,*Y*)内的插值所决定。如果**X**、**Y**和**Z**中任何一个是一个向量，则它的元素被认为应用于相应的行和列。
interp2(X,Y,Z,Xx,Yy, metstr)	进行二维插值，字符串**metstr**规定了不同的插值方法，可用的方法有： ‘linear’ 线性插值。 ‘nearest’ 最邻近插值。 ‘spline’ 三次样条插值。 ‘cubic’ 三次插值。
VV=interp3(X,Y,Z,V,XX,YY,ZZ,metstr)	进行由**X**，**Y**和**Z**所描述的函数**V**的插值，**XX**、**YY**和**ZZ**是插值点。字符串**metstr**规定了不同的插值方法，可用的方法有： ‘nearest’ 使用最邻近点的值。 ‘linear’ 使用8个最邻近点进行线性插值。 ‘spline’ 三次样条插值。 ‘cubic’ 使用64个最邻近点进行三次插值。
VV=interpn(X1,X2,X3, ···,V,Y1,, Y2,Y3,···,method)	和interp3相同，但是V和VV可以是多维数组。如果**X**1，**X**2，**X**3，…是等距离的，使用星号*如’*cubic’可以加快计算速度。
Interpft(y,n)	快速傅利叶变换插值，返回一个长度为*n*，从*y*计算得到的向量。要求*y*的值是等距离的，结果在与*y*相同区间内计算。
griddata(x,y,z,Xx,Yy,’method’)	返回相同大小的矩阵**Xx**和**Yy**，它们表示一个网格，由函数*z*=*f*(*x*, *y*)的插值得到。向量**x**、**y**和**z**包含的是三维空间的 *x*、*y*和*z*坐标。字符串**method**规定了不同的插值方法，可用的方法如下： ‘linear’ 基于三角的线性插值。 ‘nearest’ 最邻近插值。

	‘cubic’　基于三角的三次插值。 ‘v4’使用MATLAB 4的插值方法。
[X1,X2,X3,···]=ndgrid(x1,x2,x3,···)	变换由向量**x1**，**x2**，**x3**…给出的域。对于矩阵**X1**，**X2**，**X3**，…来说，可以用做多变量函数的估计值和多维插值。矩阵 **Xn**的第*n*维和向量**xn**的元素相同。
[X1X2,···]=ndgrid(x)	等同于[X1,X2,···]= ndgrid(x,x,x,···)。

■ **例**10.4

做一个函数$\sin x^2$在区间[0，2π]上40个函数值的表。

```
x = linspace(0,2*pi,40); y = sin(x.^2);
```

(a) 用interp1来计算中间点的$\sin x^2$函数值，而不用sin求，命令为：

```
values = interp1(x,y,[0 pi/2 3])
```

结果为：

```
values =
         0    0.6050    0.3559
```

和用sin计算得到的正确结果比较：

```
correct = sin([0 pi/3 3].^2)

correct =
         0    0.8897    0.4121
```

在表中用更多的数据点可以使精度更好。

(b) 用样条插值可得更高精度的结果。假设向量 x和y定义如上，那么：

```
better = interp1(x,y,[0 pi/2 3],'spline')
```

结果为：

```
better =
         0    0.6241    0.4098
```

这是一个更好的近似值。

■

命令griddata可以在三维空间内建立任意数据点外的函数。

■ **例**10.5

先生成三个有10个元素的向量，它们的值随机分布在0～1之间：

```
x = rand(10,1); y = rand(10,1); z = rand(10,1);
```

建立一个网格来计算内部曲面之后，可以用命令 griddata来对这些点之间的曲面进行插值。下面的图10-5给出了不同的插值方法之间的区别：

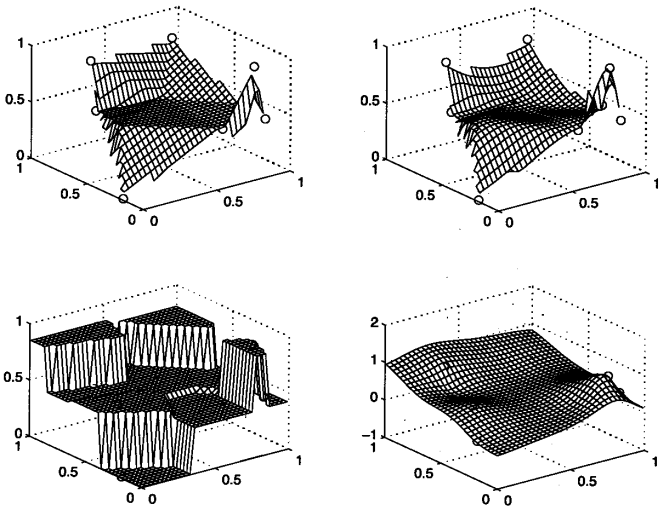

图10-5 用griddata 对10个随机点的插值，左上图用的是‘linear ’，右上图用的是‘cubic ’，右下图用的是‘nearest ’，右下图用的是‘v4’

```
stps=0:0.03:1;                          % 向量A的值在[0, 1]之间
[X,Y]=meshgrid(stps);                   % 生成一个[0,1]×[0,1]坐标网格
Z1=griddata(x, y, z, X, Y);             % 线性插值
Z2=griddata(x, y, z, X, 'cubic');       % 三次插值
Z3=griddata(x, y, z, X, 'nearest');     % 最邻近插值
Z4=griddata(x, y, z, X, 'v4');          % MATLAB 4中的插值方法

subplot(2, 2, 1);                       % 画第1个子图
mesh(X, Y, Z1);                         % 画曲面网格
hold on                                 % 保持当前图形
plot3(x, y, z,'o');                     % 画出数据点
hold off                                % 释放图形

subplot(2 2 2);                         % 画第2个子图
mesh(X, Y ,Z2);                         % 画曲面网格
hold on                                 % 保持当前图形
plot3(x, y, z,'o');                     % 画出数据点
hold off                                % 释放图形

subplot(2, 2, 3);                       % 画第3个子图
mesh(X, Y, Z3);                         % 画曲面网格
hold on;
plot3(x, y, z,'o');                     % 画出数据点
hold off

subplot(2, 2, 4);                       % 画第4个子图
mesh(X, Y, Z4);                         % 画曲面网格
hold on
plot3(x, y, z,'o');                     % 画出数据点
hold off
```

命令hold和subplot在13.3节进行了说明，命令meshgrid在13.4节中，命令mesh和plot3可以在13.5节中找到。结果如图10-5所示。

■

用spline命令可以求得三次样条的近似值，还可以求得三次样条插值 pp形式的向量，向量的元素是三次样条函数的系数。用 ppval命令可以求得三次样条函数的估计值。

命令集103 三次样条数据插值

spline(x,y,xx)	等同于interp1(x, y, xx, 'spline')，但是参数必须是向量。
spline(x,y)	返回三次样条插值向量的 **pp形式**，它是函数$y=f(x)$的近似值。pp是 ' piecewise polynomial ' 的缩写，得到的向量元素包含计算的三次样条系数。这个命令可以被 ppval函数使用。
YI=splncore(X,Y,XI)	在细胞数组**X**的函数**Y**内进行n维数据的插值，得到一个新集合**XI**。函数可以被interp2、interp3和interpn使用。
ppval(pp,xx)	计算三次样条函数。如果三次样条定义为pp=spline(x, y)，那么ppval(pp,xx)得到的结果和spline(x,y,xx)一样。
p=mkpp(points, coeff,d)	通过用给定点建立 pp函数来求分段多项式，其中 **coeff**(i, :)包含第i个多项式的系数。多项式的个数由 l=length(points)－1确定，第i个多项式的阶数为n=length(coeff(:))/l。
[points,coeff,l, n,d]=unmkpp(p)	给出分段多项式相关信息，见上。

多项式可以使用最小二乘法来进行数据拟合 (也可见7.7节)，用的命令是polyfit。

命令集104 多项式曲线拟合

polyfit(x,y,n)	找到次数为n的多项式系数，对于数据集合$\{(x_i, y_i)\}$，满足差的平方和最小。
[p,E]=polyfit(x,y,n)	返回同上的多项式 **P**和矩阵**E**。多项式系数在向量 **p**中，矩阵**E**用在polyval函数中来计算误差。

■ **例**10.6

下面给出了对向量**x**和**y**拟合不同阶的多项式图形，阶分别为 3，4，5。

```
x = [-3 -1 0 2 5.5 7];
y = [3.3 4.5 2.0 1.5 2.5 -1.2];

p3=polyfit(x, y, 3);                % 用向量x和y中元素拟合不同
p4=polyfit(x, y, 4);                % 次数的多项式
p5=polyfit(x, y, 5);
xcurve= -3.5:0.1:7.2;               % 生成x值
p3curve=polyval(p3, xcurve);        % 计算在这些x点的多项式值
p4curve=polyval(p4, xcurve);
p5curve=polyval(p5, xcurve);

plot(xcurve,p3curve,'--',xcurve,p4curve,'-.', ...
     xcurve,p5curve,'-',x,y,'*');
```

```
lx = [-1 1.5]; ly = [0 0]; hold on;

plot(lx,ly,'--',lx,ly-1.3,'-.',lx,ly-2.6,'-');

text(2,   0,'degree 3');
text(2,-1.3,'degree 4');
text(2,-2.6,'degree 5');
hold off;
```

结果如图10-6所示。

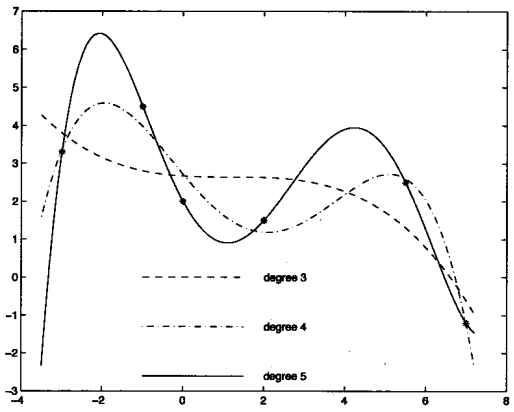

图10-6 不同次数的多项式拟合曲线图

可以看出，次数越高的多项式精度越好。 5次的多项式经过所有的 6个点。对于这个特殊的数据集合来说，这个多项式可称为内部插值多项式。

■

在MATLAB中，用legendre命令来计算标量或者向量的**勒让德函数**，它是在选定的区间内形成完全直交集合的直交多项式系统。**勒让德多项式**是阶为零的勒让德函数，可以在给定的数据集合内进行曲线拟合。**贝塞尔函数**是一个经典的特殊函数，它用在数学物理学中。

命令集105 勒让德函数和贝塞尔函数

legendre(n,x)	返回一个矩阵，它的元素是在 x内计算得到的 *n*阶，*m*=0,1，…，*n*的相关勒让德函数值。x的元素要求在区间[－1，1]内。矩阵的第1行对应*m*=0，并包含根据x计算得到的*n*阶勒让德多项式。
besselj(order,z)	计算第1类贝塞尔函数，变量 *order*指定函数的阶， z用来进行函数运算。
bessely(n,x)	计算第2类贝塞尔函数，变量 *order*指定函数的阶， z用来进行函数运算。
besselh(order,k,z)	计算向量z中元素的Hankel函数值(第3类贝塞尔函数)，变量*k*定义使用哪一类Hankel函数。
besseli(order,z)	计算改良型第 1类贝塞尔函数，变量 *order*指定函数的阶，z用来进行函数运算。

besselk(order,z)	计算改良型第2类贝塞尔函数，变量 *order*指定函数的阶，**z**用来进行函数运算。
w=airy(*k*,z)	*k*=0或者不给出*k*，计算Airy函数w=Ai(z)； *k*=1，计算导数Ai′(z)；*k*=2，计算第2类Airy函数Bi(z)； *k*=3，计算导数Bi′(z)。
[w,err]=airy(...)	返回一个错误标志数组*err*。

10.5 信号分析

这里简短地介绍一些MATLAB中用作信号分析的命令。通过help和demo命令可以得到更多信息，也可参见‘信号过程工具箱’和它的手册。复数命令(2.4节)和卷积命令(10.1节)也要涉及。

命令集106 信号分析

fft(x)	进行向量**x**的离散傅立叶变换。如果 **x**的长度是2的幂，则用快速傅立叶变换，FFT。注意，变换没有规格化。
fft(x,n)	得到一个长度为*n*的向量。它的元素是**x**中前*n*个元素离散傅立叶变换值。如果**x**有*m*<*n*个元素，则令最后的*m*+1，…，*n*元素等于零。
fft(A)	求矩阵**A**的列离散傅立叶变换矩阵。
fft(A,n,dim)	求多维数组*A*中*dim*维内列离散傅立叶变换矩阵。
ifft(x)	求向量**x**的离散逆傅立叶变换。用因子1/*n*进行规格化，*n*为向量的长度。也可象fft命令一样对矩阵或者固定长度的向量进行变换。
fft2(A)	求矩阵**A**的二维离散傅立叶变换矩阵。这个矩阵没有正交化。如果**A**=**a**是一个向量，则这个命令等同于 fft(a)命令。
fft2(A,m,n)	求矩阵**A**中相应元素的二维离散傅立叶变换矩阵。这个矩阵大小为*m*×*n*，没有正交化。如果**A**是一个小矩阵，则余下的元素用零补上。如果可能，MATLAB在计算时用快速傅立叶变换，即FFT。
ifft2(A)	求矩阵**A**的二维离散逆傅立叶变换矩阵。用因子1/*mn*进行规格化。可以象ff2一样改变变换矩阵的维数。
fftn(A,Size)	求*n*维数组**A**的*n*维离散傅立叶变换数组。如果 **A**是一个向量，则得到的结果也一个向量。如果给定了数组的大小 **Size**，则**X**重构成与**Size**一样大小。
ifftn(A,Size)	求*n*维数组**A**的*n*维离散逆傅立叶变换数组。如果 **A**是一个向量，则得到的结果也一个向量。如果给定了数组的大小 **Size**，则**X**重构成与**Size**一样大小。
fftshift(A)	返回一个将矩阵 **A**的第1象限和第3象限、第2象限和第4象限互换的数组。如果**A**是一个向量，则返回一个左半部和右半部互换的向量。如果**A**是一个*n*维数组(*n*>2)，则返回一个同等大小的数组，将**A**的每一维内的左右半部互换。

`ifftshift(A)`	`fftshift(A)`命令的逆变换。
`filter(b,a,x)`	由向量**a**和**b**所描述形成的滤波器对x向量进行数字滤波，产生滤波后的数据。输入`help filter`可得更多信息。
`Y=filter2(h,X)`	用在矩阵**h**中的**FIR**滤波器处理**x**中的数据。结果**Y**由二维卷积计算得到，包含卷记的中心部分，且与**X**的大小相同。
`Y=filter2(h, X,form)`	Y由二维卷积计算得出，其维数由参数**form**规定，*form*是下列字符串之一： '`full`' 返回二维卷积的全部，这样**Y**就比**X**大。 '`same`' 等同于`Y=filter2(h,X)`。 '`valid`' 仅仅返回卷积的一部分，这部分是不带零插值边缘计算的。这样**Y**就比**X**小。

■ **例**10.7

新建一个名为'hat funtion' 的函数，之后对其进行傅立叶变换。这个函数在 0,1处值为0，在0.5处值为1。

用linspace建立这个函数：

```
x = linspace(0,1,100);
y = [linspace(0,1,50) linspace(1,0,50)];
```

用下列命令画出函数的图形，如图 10-7所示：

```
subplot(1,3,1); plot(x,y);
title('A hat function');
```

进行傅立叶变换：

```
subplot(1,3,2); plot(x,fft(y));
title('The Fourier transform');
```

这个傅立叶变换得到复数值，但是只显示出实数部分。最后，对它进行逆傅立叶变换得到原来函数：

```
subplot(1,3,3); plot(x,ifft(fft(y)));
title('Retransformed hat function');
```

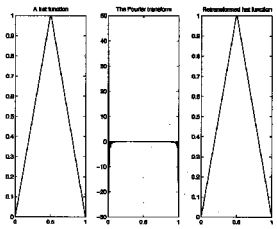

图10-7 函数的傅立叶变换图

所有的画图命令都定义在第 13章中。

■