第3章 MATLAB数值计算

数值计算是指有效使用数字计算机求数学问题近似解的方法与过程，以及由相关理论构成的学科。从数学类型分，数值运算的研究领域包括数值逼近、数值微分和数值积分、数值代数、最优化方法、常微分方程数值解法、积分方程数值解法、偏微分方程数值解法、计算几何、计算概率统计等。

3.1 数据排序

数据排序是指按一定规则对数据进行整理、排列，为数据的进一步处理做好准备。在MATLAB中，也提供了相关内置函数用于实现数据的排序处理。

3.1.1 最值

已知数据序列求序列的最大值、最小值，是实际工程中经常遇到的问题。对这类数值分析问题，MATLAB提供了强大的支持。MATLAB提供max、min函数分别用于求数据序列的最大值、最小值。函数的调用格式如下。

C = max(A)：如果A为向量，则返回A的最大值；如果A为N维数组，则沿A的第一个长度不为1的维求最大值，返回N－1维数组C；特别的，当N=2时，则返回一个行向量，行向量的元素对应A每列的最大值。

C = max(A,B)：返回与A、B大小相同的数组，数组元素取A、B对应元素的较大者。

C = max(A,[],dim)：沿A第dim维求最大值，如max(A,[],1)为沿A列向取最大值。

[C,I] = max(…)：不仅返回最大值，而且返回最大值对应的索引；如果最大值有重复，则返回第一个最大值的索引。

min函数的调用格式与max函数调用格式类似，只是其用于返回最小值。

【例 3-1】 求创建矩阵的最大值与最小值。

```
>> A = [1.7 1.2 1.5; 1.3 1.6 1.99]                    % 创建矩阵
A =
    1.7000    1.2000    1.5000
    1.3000    1.6000    1.9900
>> M = max(A,[],2)
M =
    1.7000
    1.9900
>> M = max(A)
M =
    1.7000    1.6000    1.9900
>> m = min(A)
m =
    1.3000    1.2000    1.5000
```

3.1.2 平均值与中值

在数据序列中，也经常会遇到求序列的平均值与中值。在 MATLAB 中，提供了 mean 函数用于求序列的平均值，median 函数用于求序列的中值。函数的调用格式如下。

M = mean(A)：返回一个行向量，其第 i 个元素是矩阵 A 的第 i 列的算术平均值。

mean(A, dim)：当 dim 为 1 时，该函数等同于 mean(A)；当 dim 为 2 时，则返回一个列向量，其第 i 个元素是 A 的第 i 行的算术平均值。

M = median(A)：如果 A 为向量，则返回 A 的中位数；如果 A 为 N 维矩阵，则函数沿第一个长度不为 1 的维取中位数，返回 N−1 维矩阵；特别的，当 N=2 时，函数返回一个行向量，行向量元素为对应 A 每列的中位数。

M = median(A,dim)：对 A 沿第 dim 维求中位数，如 median(A,1)为沿 A 列向求中位数。

【例 3-2】 求矩阵的均值与中位数。

```
>> clear all;
A = [0 1 1; 2 3 2; 1 3 2; 4 2 2]
A =
     0     1     1
     2     3     2
     1     3     2
     4     2     2
>> mean(A)                                            % 求矩阵的行均值
ans =
    1.7500    2.2500    1.7500
>> mean(A,2)                                          % 求矩阵的列均值
ans =
    0.6667
    2.3333
    2.0000
```

```
    2.6667
>> M = median(A)                                    % 矩阵的中位数
M =
    1.5000    2.5000    2.0000
>> M2 = median(A,2)                                 % 矩阵的列中位数
M2 =
     1
     2
     2
     2
>> B = gallery('integerdata',10,[1,3,4],1)         % 创建高维矩阵
B(:,:,1) =
    10     8    10
B(:,:,2) =
     6     9     5
B(:,:,3) =
     9     6     1
B(:,:,4) =
     4     9     5
>> M = median(B)                                    % 高维矩阵的中位数
M(:,:,1) =
    10
M(:,:,2) =
     6
M(:,:,3) =
     6
M(:,:,4) =
     5
```

3.1.3 分位数

对数值序列 X,其中 $p(0\leqslant p\leqslant 1)$分位数是满足$\dfrac{\text{count}(X\leqslant q)}{\text{count}(X)}=p$ 的 q,其中 count(X)为数值序列的长度,count$(X\leqslant q)$是 X 中不大于q 的元素个数。通常上式的等号不能成立,可以定义 p 分位数是使$\left|\dfrac{\text{count}(X\leqslant q)}{\text{count}(X)}-p\right|$最小的$q$。

在 MATLAB 中提供了 quantile 函数用于求数值序列的分位数。其调用格式为

```
Y = quantile(X,p)
Y = quantile(X,p,dim)
```

其中 X 为数值序列,p 可以为向量,用法与 max 类似。

【例 3-3】 已知数据序列 $X=[-500,-499,\cdots,500]$,求 X 的 0,1/4,1/2,3/4,1 分位数。

```
>> clear
>> X=-500:500;
>> P=[0,1/4,1/2,3/4,1];
```

```
>> quantile(X,P)
ans =
  -500.0000  -250.2500         0   250.2500   500.0000
```

注意：quantile 将 NaN 视为缺失数据，在计算分位数时会忽略这些缺失数据。

3.1.4 求和(积)

MATLAB 提供的数据序列求和与求积的函数分别为 sum 和 prod。函数的调用格式如下。

B = sum(A)：沿数组第一个非 1 的维进行求和。如果 A 为向量，则返回该向量的和；如果 A 为矩阵，则函数沿列方向求和，返回一个行向量，行向量的元素对应 A 的每一列的和。

B = sum(A,dim)：指定函数沿第 dim 求和。

B = sum(…, 'double')或 B = sum(…, dim,'double')：返回一个双精度求和结果。

B = sum(…, 'native')或 B = sum(…, dim,'native')：返回一个本地数据类型的求和结果，默认值为双精度。

B = prod(A)：计算数据 A 所有元素的乘积。如果 A 为向量，则返回向量所有元素的积；如果 A 为矩阵，则返回各列元素的积。

B = prod(A,dim)：指定维上的数据 A 的元素乘积，dim=1 时计算列元素乘积，dim=2 时计算行元素的乘积。

B = prod(__,datatype)：datatype 为指定返回求积值 B 的数据类型，其类型值可为 double 或 native。

【例 3-4】 求向量及矩阵的和与积。

```
>> X=[1 3 8 9 11 0 5];
>> sum(X)                              %求向量和
ans =
    37
>> A=magic(3)
A =
     8     1     6
     3     5     7
     4     9     2
>> sum(A)                              %求矩阵 A 的列和
ans =
    15    15    15
>> sum(A,2)                            %求矩阵 A 的行和
ans =
    15
    15
    15
>> prod(A)                             %求矩阵各列元素的积
ans =
    96    45    84
```

```
>> B1 = prod(A,2,'double')          % 求矩阵各行元素的积,返回数据类型为 double
B1 =
    48
   105
    72
```

3.1.5 求累积积(和)

设 $\boldsymbol{U}=(u_1,u_2,\cdots,u_n)$ 为一个向量,$\boldsymbol{V}$、$\boldsymbol{W}$ 是与 $\boldsymbol{U}$ 等长的另外两个向量,并且

$$\boldsymbol{V}=\left(\sum_{i=1}^{1}u_i,\sum_{i=1}^{2}u_i,\cdots,\sum_{i=1}^{n}u_i\right)$$

$$\boldsymbol{W}=\left(\prod_{i=1}^{1}u_i,\prod_{i=1}^{2}u_i,\cdots,\prod_{i=1}^{n}u_i\right)$$

则称 $\boldsymbol{V}$ 为 $\boldsymbol{U}$ 的累加和向量,$\boldsymbol{W}$ 为 $\boldsymbol{U}$ 的累乘积向量。

在 MATLAB 中,使用函数 cumsum 和 cumprod 函数可方便地计算数据序列的累和和累积,函数的调用格式如下。

B = cumsum(A):计算数据 A 的累积和。如果 A 为向量,则计算该向量的累积和;如果 A 为矩阵,则计算矩阵各列元素的累和。

B = cumsum(A,dim):当 dim 为 1 时,该函数等同于 comsum(A);当 dim 为 2 时,则返回一个矩阵,其第 i 行是矩阵 A 的第 i 行的累加和向量。

B = cumprod(A):计算数据 A 的累积。如果 A 为向量,则计算该向量的累积积;如果 A 为矩阵,则计算矩阵各列元素的累积。

B = cumprod(A,dim):当 dim 为 1 时,该函数等同于 comprod (A);当 dim 为 2 时,则返回一个矩阵,其第 i 行是矩阵 A 的第 i 行的累乘积向量。

【例 3-5】 对矩阵进行累加和与累乘积运算。

```
>> A = [1 2 3; 4 5 6];
>> cumsum(A)                        % 求矩阵 A 列累加和
ans =
     1     2     3
     5     7     9
>> cumsum(A,2)                      % 求矩阵 A 行累加和
ans =
     1     3     6
     4     9    15
>> cumprod(A,2)                     % 求矩阵 A 行累乘积
ans =
     1     2     6
     4    20   120
>> cumprod(A)                       % 求矩阵 A 列累乘积
ans =
     1     2     3
     4    10    18
```

注意:cumsum、cumprod 不能识别 NaN 数据,用户使用 cumsum、cumprod 时需要首

先去掉数据序列中的 NaN 数据。

3.1.6 方差与标准差

样本方差为样本中各数据与样本平均数的差的平方和的平均数，而样本的算术平方根即为样本标准差。方差与标准差用于反映数据的离散程度，方差或标准差越大，数据的离散程度越大。

方差的计算公式为

$$\sigma^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2$$

其中，n 为样本数，$\bar{x}$ 为样本数据的均值。

标准差的计算公式为

$$\mathrm{std} = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

其中，n 为样本数，$\bar{x}$ 为样本数据的均值。

在 MATLAB 中，提供了 var 函数用于计算样本方差，提供了 std 函数用于计算样本数据的标准差。函数的调用格式如下。

V = var(X)：计算数据 X 的方差。如果 X 为向量，则返回向量的方差；如果 X 为矩阵，则返回矩阵各列数据的方差。

V = var(X,1)：计算样本的方差，前置因子样本数不减 1，即为 1/n。

V = var(X,w)：返回以 w 为权值的数据 X 的方差。

V = var(X,w,dim)：返回以 w 为权值的数据 X 的方差，并指定方差计算的维度 dim。

s = std(X)：计算数据 X 的标准差。如果 X 为向量，则返回该向量的标准差；如果 X 为矩阵，则返回矩阵的各列数据的标准差。

s = std(X,flag)：计算数据 X 的标准差，flag 的值为 0 时前置因子为 1/(n−1)，其他情况下为 1/n，默认时 flag=0。

s = std(X,flag,dim)：计算数据 X 的标准差，参数 dim 指定标准差计算的维度。当 dim=1 时，计算各列数据的标准方差；当 dim=2 时，计算各行数据的标准差。

【例 3-6】 计算矩阵的方差与标准差。

```
>> A = [4 -2 1; 9 5 7; 3 -7 6];          % 矩阵
>> var(A)                                % 矩阵的方差
ans =
   10.3333   36.3333   10.3333
>> std(A)                                % 矩阵的标准差
ans =
    3.2146    6.0277    3.2146
>> var(A,1,1)                            % 计算矩阵维度为 1,权值为 1 的方差
ans =
    6.8889   24.2222    6.8889
>> var(A,1,2)                            % 计算矩阵维度为 1,权值为 2 的方差
```

```
ans =
    6.0000
    2.6667
   30.8889
>> std(A,0,1)                          % 计算矩阵维度为 1,前置因子为 1/(n-1)的标准差
ans =
    3.2146    6.0277    3.2146
>> std(A,0,2)                          % 计算矩阵维度为 2,前置因子为 1/(n-1)的标准差
ans =
    3.0000
    2.0000
    6.8069
```

3.1.7 协方差与相关系数

协方差(covariance)在概率论和统计学中用于衡量两个变量的总体误差。而相关系数用于衡量两个变量间的线性关系的程度。

对于数据序列 X 和 Y,自相关函数定义为 $R(X)=E(X^{\mathrm{T}}X)V$;互相关函数定义为 $R(X,Y)=E(X^{\mathrm{T}}Y)$;协方差定义为 $C(X)=E((X-E(X))^{\mathrm{T}}(X-E(X)))$;互协方差定义为 $C(X)=E((X-E(X))^{\mathrm{T}}(Y-E(Y)))$。

在 MATLAB 中提供了 corr 函数用于相关分析,提供了 cov 函数用于协方差分析。它们的调用格式如下。

cov(X):X 为 n×p 矩阵,返回值 R 为 p×p 矩阵。

cov(X,Y):X、Y 分别为 n×p1、n×p2 矩阵,返回值 R 为 p1×p2 矩阵。

RHO = corr(X):如果 X 为向量,则返回 X 的方差;如果为矩阵,则 cov(x)=$E((X-E(X))^{\mathrm{T}}(X-E(X)))$。

RHO = corr(X,Y):X、Y 为长度相等的向量,cov(X,Y)=cov([X(:),Y(:)])。

对于两组数据序列 $x_i,y_i(i=1,2,\cdots,n)$,可以由下式计算出两组数据的相关系数

$$r=\frac{\sqrt{\sum_i (x_i-\bar{x})(y_i-\bar{y})}}{\sqrt{\sum_i (x_i-\bar{x})^2}\sqrt{\sum_i (y_i-\bar{y})^2}}$$

[RHO,PVAL] = corr(X,Y):同时返回一个非零相关性的相关性矩阵 PVAL。

在 MATLAB 中提供了 corrcoef 函数用于求数据的相关系数矩阵。其调用格式如下。

R = corrcoef(X):返回从矩阵 X 形成的一个相关系数矩阵。此相关系数矩阵的大小与矩阵 X 一样。它把矩阵 X 的每列作为一个变量,然后求它们的相关系数。

R = corrcoef(x,y):在这里,x,y 为向量,它们的作用与 corrcoef([x,y])中一样。

【例 3-7】 计算数据序列的协方差及相关系数。

```
>> clear all;
rng('default')
x = randn(30,4);
```

```
y = randn(30,4);
y(:,4) = sum(x,2); %引入相关
[r1,p1] = corr(x,y)                    %计算x和y中列之间的相关性
r1 =
   -0.1686   -0.0363    0.2278    0.6901
    0.3022    0.0332   -0.0866    0.2617
   -0.3632   -0.0987   -0.0200    0.3504
   -0.1365   -0.1804    0.0853    0.4908
p1 =
    0.3731    0.8489    0.2260    0.0000
    0.1045    0.8619    0.6491    0.1624
    0.0485    0.6039    0.9166    0.0577
    0.4721    0.3400    0.6539    0.0059
>> [r2,p2] = corrcoef(x,y)             %计算x和y之间的相关性
r2 =
    1.0000    0.1252
    0.1252    1.0000
p2 =
    1.0000    0.1729
    0.1729    1.0000
>> c = cov(x,y)                        %计算x和y之间的协方差
c =
    1.3188    0.1914
    0.1914    1.7717
```

3.1.8 排序

数据的排序是理论和实际中经常遇到的问题。MATLAB提供sort和sortrows两个函数用于数据的排序操作。这两个函数不仅可以用于数值数据的排序，而且对字符串数据也可以进行排序，在此主要介绍数值排序。

sort对数组元素按升序或降序进行排列，数组元素的类型既可以是整型、浮点型、逻辑类型等数值类型，也可以是字符、字符串。函数sort对字符或字符数组的排序依据ASCII进行。对复数数值类型，sort函数首先比较各元素的模值，在模值相同的情况下，考虑$(-\pi,\pi)$上的相位值；对于NaN数据，sort函数将其排在最后，不管是按升序还是按降序排列。

sort函数的调用格式如下。

B = sort(A)：其中A为M×N×…×P数组，函数沿第一个长度非1的维进行升序排列。例如当A为M×1(M>1)向量时，函数沿第一维进行升序排列；当A为1×N(N>1)，第一维长度为1，因此函数沿第二维进行升序排列；当A为M×N(M,N>1)，函数沿第一维即列向排序。

B = sort(A,dim)：其中A为M×N×…×P数组，函数沿第dim维进行升序排列。

B = sort(…,mode)：mode为排序模式，可以选择升序ascend或降序descend，默认情况下为升序排列。

[B,IX] = sort(A,…)：函数不仅返回排序后的数组，而且返回B在原来数组中相

应的索引值。

【例 3-8】 对数值矩阵进行排序。

```
>> A = [ 3  7  5; 6  8  3; 0  4  2 ];
>> s = sort(A)                          % 对矩阵 A 进行升序排序
s =
     0     4     2
     3     7     3
     6     8     5
>> s = sort(A,'descend')                % 对矩阵 A 进行降序排序
s =
     6     8     5
     3     7     3
     0     4     2
>> [S,IND] = sort(A)
S =
     0     4     2
     3     7     3
     6     8     5
IND =
     3     3     3
     1     1     2
     2     2     1
```

除了 sort，MATLAB 还提供了 sortrows 排序函数，sort 将每一行作为一个整体，沿列向进行排序。sortrows 函数的调用格式如下。

B = sortrows(A)：将 A 的每一行作为整体沿列向进行升序排列，如果依赖第一列，A 具有相同的两行，那么再依据第二列对这两行进行比较，依此类推。

应注意的是，这里的 A 只能是列向量或者矩阵。

B = sortrows(A,column)：依据第 column 列对 A 进行升序排列，并且 column 可以为向量，如果依据第 coloumn 列，A 的某两行相同，那么 sortrows 不改变其次序。

[B,index] = sortrows(A,…)：返回排序的索引值 index，index 为列向量。当 A 为列向量时，B=A(Index)；当 A 为矩阵时，B=A(index,:)。

【例 3-9】 利用 sortrows 对数据进行排序。

```
>> clear all;
A = floor(gallery('uniformdata',[6 7],0) * 100);
A(1: 4,1) = 95;   A(5: 6,1) = 76;   A(2: 4,2) = 7;   A(3,3) = 73     % 创建矩阵
A =
    95    45    92    41    13     1    84
    95     7    73    89    20    74    52
    95     7    73     5    19    44    20
    95     7    40    35    60    93    67
    76    61    93    81    27    46    83
    76    79    91     0    19    41     1
>> B = sortrows(A)                                  % 沿矩阵列向进行升序排序
B =
    76    61    93    81    27    46    83
```

```
    76    79    91     0    19    41     1
    95     7    40    35    60    93    67
    95     7    73     5    19    44    20
    95     7    73    89    20    74    52
    95    45    92    41    13     1    84
>> C = sortrows(A,2)                          % 沿矩阵第 2 列进行升序排序
C =
    95     7    73    89    20    74    52
    95     7    73     5    19    44    20
    95     7    40    35    60    93    67
    95    45    92    41    13     1    84
    76    61    93    81    27    46    83
    76    79    91     0    19    41     1
>> D = sortrows(A,[1 7])                      % 沿矩阵 1～7 列进行升序排序
D =
    76    79    91     0    19    41     1
    76    61    93    81    27    46    83
    95     7    73     5    19    44    20
    95     7    73    89    20    74    52
    95     7    40    35    60    93    67
    95    45    92    41    13     1    84
>> E = sortrows(A, -4)                        % 沿矩阵第 4 列相反的方向按升序排序
E =
    95     7    73    89    20    74    52
    76    61    93    81    27    46    83
    95    45    92    41    13     1    84
    95     7    40    35    60    93    67
    95     7    73     5    19    44    20
    76    79    91     0    19    41     1
```

3.1.9 偏斜度与峰值

偏斜度可用于对统计数据分析偏斜方向及程度的度量。偏斜度为 0,即可认为数据的分布是对称的;如果大于 0,则数据的分布偏右;如果小于 0,则数据的分布偏左。在 MATLAB 中,提供了 skewness 函数用于计算数据的偏斜度。函数的调用格式如下。

y = skewness(X):计算数据 X 的偏斜度。如果 X 为向量,则返回此向量的偏斜度;如果 X 为矩阵,则返回矩阵各列数据的偏斜度。

y = skewness(X,flag):flag 参数用于设置是否纠正偏离后再返回偏斜度。当 flag =0 时,纠正偏离;当 flag=1 时,默认状态下不纠正偏离。

y = skewness(X,flag,dim):沿指定的 dim 维计算数据的偏斜度。

【例 3-10】 计算数据序列的偏斜度。

```
>> X = randn([5 4])
X =
   -2.9443   -1.7115   -0.8649    1.1093
    1.4384   -0.1022   -0.0301   -0.8637
    0.3252   -0.2414   -0.1649    0.0774
```

```
    -0.7549      0.3192      0.6277    -1.2141
     1.3703      0.3129      1.0933    -1.1135
>> skewness(X)
ans =
    -0.7380    -1.1813    -0.0082     0.7439
>> skewness(X,0,1)                              % 沿列计算偏斜度,并纠正偏离
ans =
    -1.1001    -1.7609    -0.0122     1.1090
>> skewness(X,0,2)                              % 沿行计算偏斜度,纠正偏离
ans =
     0.5886
     1.0506
     0.6666
    -0.1274
    -1.1573
>> skewness(X,1,2)                              % 沿行计算偏斜度,不纠正偏离
ans =
     0.3398
     0.6065
     0.3848
    -0.0735
    -0.6682
>> skewness(X,1,1)                              % 沿列计算偏斜度,不纠正偏离
ans =
    -0.7380    -1.1813    -0.0082     0.7439
```

峰度用于衡量变量的集中程度,正态分布的数据峰度为 0,峰度可以衡量数据偏离正态分布的程度。在 MATLAB 中,提供了 kurtosis 函数用于计算数据的峰度。函数的调用格式如下。

k = kurtosis(X):计算数据 X 的峰度。如果 X 为向量,则返回此向量的峰度;如果 X 为矩阵,则返回矩阵各列数据的峰度。

k = kurtosis(X,flag):flag 参数用于设置是否纠正偏离后再返回峰度。当 flag=0 时,则纠正偏离;当 flag=1 时,默认状态下不纠正偏离。

k = kurtosis(X,flag,dim):计算指定数据维上的数据峰度。

【例 3-11】 计算数据序列的峰度。

```
>> X = randn([5 4]);
>> kurtosis(X)                                  % 计算矩阵各列的峰度
ans =
    2.3779    2.1660    2.2097    2.0843
>> kurtosis(X,0,1)                              % 计算矩阵各列的峰度,并纠正偏离
ans =
    4.5118    3.6641    3.8390    3.3371
>> kurtosis(X,0,2)                              % 计算矩阵各行的峰度,并纠正偏离
ans =
    3.7231
    3.6727
    1.4767
    1.2113
```

```
    0.2929
>> kurtosis(X,1,2)                                  % 计算矩阵各行的峰度,不纠正偏离
ans =
    1.8964
    1.8897
    1.5969
    1.5615
    1.4391
>> kurtosis(X,1,1)                                  % 计算矩阵各列的峰度,不纠正偏离
ans =
    2.3779    2.1660    2.2097    2.0843
```

3.2 符号运算

符号表达式是代数数字、函数、算子和变量的 MATLAB 字符串或字符数组。符号方程式是含有等号的符号表达式。MATLAB 在内部把符号表达式表示成字符串,以与数字变量或运算相区别,否则,这些符号表达式几乎完全像基本的 MATLAB 命令。

MATLAB 中符号运算的核心函数是 sym 函数和 syms 函数,sym 函数用于构造变量和表达式,syms 函数用于构造符号对象。

符号变量确定原则如下:

- 除了 i 和 j 之外,字母为位置最接近 x 的字母;如果距离相等,则取 ASCII 码大的。
- 如果没有除了 i 与 j 以外的字母,则视 x 为默认的符号变量。
- 可利用 findsym(string,N)函数来询问在众多符号中,哪个为符号变量。例如,输入 findsym(2 * a * b+y ^2,1),即可得到答案 y。

3.2.1 符号对象的生成

利用 sym 函数和 syms 函数创建符号对象,具体可以看如下实例。

【例 3-12】 符号常数形成中的差异。

```
>> clear all;
>> a1 = [1/3,pi/7,sqrt(5),pi + sqrt(5)]             % a1 是数值常数,不是符号对象
a1 =
    0.3333    0.4488    2.2361    5.3777
>> a2 = sym([1/3,pi/7,sqrt(5),pi + sqrt(5)])        % 最接近的有理表示
a2 =
[ 1/3, pi/7, 5 ^(1/2), 189209612611719/35184372088832]
>> a3 = sym([1/3,pi/7,sqrt(5),pi + sqrt(5)],'e')    % 带估计误差的有理表示
a3 =
[1/3 - eps/12, pi/7 - (13 * eps)/165, (137 * eps)/280 + 5 ^(1/2), 189209612611719/
35184372088832]
>> a4 = sym('[1/3,pi/7,sqrt(5),pi + sqrt(5)]')      % 绝对准确的符号数值表示
a4 =
[ 1/3, pi/7, 5 ^(1/2), pi + 5 ^(1/2)]
```

```
 >> a = a2 - a4
a =
[ 0, 0, 0, 189209612611719/35184372088832 - 5^(1/2) - pi]
>> whos
  Name       Size             Bytes  Class     Attributes
  a          1x4                 60  sym
  a1         1x4                 32  double
  a2         1x4                 60  sym
  a3         1x4                 60  sym
  a4         1x4                 60  sym
```

【例 3-13】 几种输入下产生矩阵的异同。

```
>> clear all;
>> a1 = sym([1/3,0.25 + sqrt(2),pi])                    % 产生 1 * 3 符号数组
a1 =
[ 1/3, 7494951579368397/4503599627370496, pi]
>> a2 = sym('[1/3,0.25 + sqrt(2),pi]')                  % 产生 1 * 3 符号数组
a2 =
[ 1/3, 2^(1/2) + 0.25, pi]
>> a = a1 - a2                                          % 比较 a1 与 a2
[ 0, 1.41421356237309514547462185873883 - 2^(1/2), 0]
```

【例 3-14】 求矩阵 $\boldsymbol{A}=\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$ 的行列式值、逆和特征根。

```
>> clear all;
>> syms a11 a12 a21 a22;                                % 定义符号变量
>> A = [a11,a12; a21,a22]
A =
[ a11, a12]
[ a21, a22]
>> Da = det(A)                                          % 符号矩阵的行列式
Da =
a11 * a22 - a12 * a21
>> Ea = eig(A)                                          % 符号矩阵的特征根
Ea =
 a11/2 + a22/2 - (a11^2 - 2 * a11 * a22 + a22^2 + 4 * a12 * a21)^(1/2)/2
 a11/2 + a22/2 + (a11^2 - 2 * a11 * a22 + a22^2 + 4 * a12 * a21)^(1/2)/2
>> Ia = inv(A)                                          % 符号矩阵的逆
Ia =
[  a22/(a11 * a22 - a12 * a21), -a12/(a11 * a22 - a12 * a21)]
[ -a21/(a11 * a22 - a12 * a21),  a11/(a11 * a22 - a12 * a21)]
```

3.2.2 符号表达式的操作

符号表达式的操作有很多种，如合并同幂项、因子分解、表达式和简化等，下面进行介绍。

1. 合并同幂项

降幂排列法是各种化简方法中最简单的一种，在 MATLAB 中由 collect 函数完成。函数的调用格式如下。

R = collect(S)：将表达式 S 中相同次幂的项合并，系统默认为按照 x 的相同次幂项进行合并。

R = collect(S,v)：将表达式 S 按照 v 的相同次幂项进行合并。输入参数 S 既可以是一个表达式，也可以是一个符号矩阵。

【例 3-15】 按不同的方式合并同幂项。

```
>> clear all;
>> s = sym('(x^2 + x * exp( - t) + 1) * (x + exp( - t))');    % 创建符号多项式
>> exp1 = collect(s)                                          % 默认合并 x 同幂项系数
exp1 =
x^3 + 2 * exp( - t) * x^2 + (exp( - 2 * t) + 1) * x + exp( - t)
>> exp2 = collect(s,'exp( - t)')                              % 合并 exp( - t)同幂项系数
exp2 =
x * exp( - 2 * t) + (2 * x^2 + 1) * exp( - t) + x * (x^2 + 1)
```

或者

```
>> syms x y;
>> r1 = collect((exp(x) + x) * (x + 2))
r1 =
x^2 + (exp(x) + 2) * x + 2 * exp(x)
>> r2 = collect((x + y) * (x^2 + y^2 + 1),y)
r2 =
y^3 + x * y^2 + (x^2 + 1) * y + x * (x^2 + 1)
>> r3 = collect([(x + 1) * (y + 1),x + y])
r3 =
[ (y + 1) * x + y + 1, x + y]
```

2. 因式分解

在 MATLAB 中提供了 factor 函数用于将表达式进行因式分解，其调用格式如下。

f = factor(n)：其中，n 是多项式或多项式矩阵，系数是有理数，MATLAB 还会将表达式 n 表示成系数为有理数的低阶多项式相乘的形式，如果多项式 n 不能在有理数范围内进行因式分解，则该函数会返回 n 本身，默认 x 为第一变量。

【例 3-16】 利用 factor 进行不同情况的因式分解。

```
% 除 x 外不含其他自由变量的情况
>> clear all;
>> syms a x;
>> f1 = x^4 - 5 * x^3 + 5 * x^2 + 5 * x - 6;
>> factor(f1)
ans =
[ x - 1, x - 2, x - 3, x + 1]
```

```
>> %含有其他自由变量情况之一
>> f2 = x^2 - a^2;
>> factor(f2)
ans =
[ -1, a - x, a + x]
>> %对正整数的质数分解
>> factor(1025)
ans =
     5     5    41
```

3. 重叠法

重叠法是一种很特别的代数式的整理化简方法。它的化简方法是将代数式尽量化为 $ax(bx(cx(\cdots(zx+z')+y')\cdots)+b')+a'$ 的形式。在 MATLAB 中，提供了 horner 函数用于实现重叠法。函数的调用格式如下。

horner(P)：其中 P 是符号多项式矩阵，函数 horner 将其中的每个多项式转换成它们的嵌套形式。

【例 3-17】 将符号表达式变换成重叠形式。

```
>> syms x y
horner(x^3 - 6*x^2 + 11*x - 6)                    %转换重叠形式
ans =
x*(x*(x - 6) + 11) - 6
>> horner([x^2 + x; y^3 - 2*y])
ans =
   x*(x + 1)
 y*(y^2 - 2)
```

4. 分子、分母表达式

如果表达式是一个有理分式(两个多项式之比)，或者可以展开为有理分式(包括那些分母为 1 的分式)，则可利用函数 numden 提取分子或分母。例如，给定如下的表达式

$$m=x^2,\quad f=\frac{ax^2}{b-x},\quad g=\frac{3}{2}x^2+\frac{2}{3}x-\frac{3}{5},$$

$$h=\frac{x^2+3}{2x-1}+\frac{3x}{x-1},\quad k=\begin{pmatrix}\frac{3}{2} & \frac{2x+1}{3}\\ \frac{4}{x^2} & 3x+4\end{pmatrix}$$

在必要时，numden 将表达式合并、有理化并返回所得的分子和分母。函数的调用格式如下。

[N,D] = numden(A)：输入参数 A 可以为符号表达式，也可以为符号矩阵。输出参数 N 为分子、D 为分母。

【例 3-18】 提取矩阵 $\begin{bmatrix}\frac{3}{2} & \frac{x^2+3}{2x-1}+\frac{3x}{x-1}\\ \frac{4}{x^2} & 3x+4\end{bmatrix}$ 各元素的分子、分母多项式。

```
>> syms x;
>> A = [3/2,(x^2 + 3)/(2 * x - 1) + 3 * x/(x - 1); 4/x^2,3 * x + 4]
A =
[   3/2, (3 * x)/(x - 1) + (x^2 + 3)/(2 * x - 1)]
[ 4/x^2,                              3 * x + 4]
>> [n,d] = numden(A)                                    % 提取矩阵的分子分母
n =
[ 3, x^3 + 5 * x^2 - 3]
[ 4,         3 * x + 4]
d =
[   2, (2 * x - 1) * (x - 1)]
[ x^2,                 1]
>> pretty(simplify(A))                                  % 写成习惯形式
/                2     \
|  3   3 x     x   + 3 |
|  -, -----  + ------- |
|  2  x - 1   2 x - 1 |
|                     |
|  4                  |
| --,     3 x + 4     |
|  2                  |
\ x                  /
```

5. 化简

在 MATLAB 中，提供了 simlify 函数和 simple 函数对符号表达式进行化简。

(1) simplify 函数

simplify 函数是一个具有普遍意义的工具，能够对包含和式、方根、分数的乘方、指数函数、对数函数、三角函数等的表达式进行化简。函数的调用格式如下。

simplify(S)：输入参数 S 为符号表达式或符号矩阵。

simplify(S,Name,Value)：指定一个或多个属性名称及其值，对符号表达式或矩阵进行化简。

(2) simple 函数

也可以采用 simple 函数进行符号表达式的化简。该方法比 simplify 函数简单，所得结果也比较合理。函数的调用格式如下。

simple(S)：将显示所有使表达式 S 变短的最简化形式，并返回其中最短的一个表达式。

simple(S,Name,Value)：指定一个或多个参数名及参数值，对表达式进行化简。

r = simple(S)或 r = simple(S,Name,Value)：使用不同的变换简化规则对符号表达式进行简化，返回表达式 S 的最简形式。如果 S 为符号表达式矩阵，则返回表达式矩阵变成最短的形式，而不一定是使每一项都最短。

[r,how]= simple(S)或[r,how]= simple(S,Name,Value)：不显示简化的中间结果，只是显示寻找到的最短形式以及所有可以使用的简化方法。r 为符号表达式的结果，how 则是使用的方法。

【例 3-19】 利用 simple 及 simplify 函数化简符号表达式。

```
>> syms x a b c
simplify(sin(x)^2 + cos(x)^2)
ans =
1
>> simplify([(x^2 + 5*x + 6)/(x + 2),sin(x)*sin(2*x) + cos(x)*cos(2*x);
(exp(-x*i)*i)/2 - (exp(x*i)*i)/2, sqrt(16)])
ans =
[  x + 3, cos(x)]
[ sin(x),      4]
>> f2 = sym('cos(x) + i*sin(x)');
>> simple(f2)
simplify:
cos(x) + sin(x)*i
radsimp:
cos(x) + sin(x)*i
simplify(Steps = 100):
cos(x) + sin(x)*i
combine(sincos):
cos(x) + sin(x)*i
combine(sinhcosh):
cos(x) + sin(x)*i
combine(ln):
cos(x) + sin(x)*i
factor:
cos(x) + sin(x)*i
expand:
cos(x) + sin(x)*i
combine:
cos(x) + sin(x)*i
rewrite(exp):
exp(x*i)
rewrite(sincos):
cos(x) + sin(x)*i
rewrite(sinhcosh):
cosh(x*i) + sinh(x*i)
rewrite(tan):
(tan(x/2)*2*i)/(tan(x/2)^2 + 1) - (tan(x/2)^2 - 1)/(tan(x/2)^2 + 1)
mwcos2sin:
sin(x)*i - 2*sin(x/2)^2 + 1
collect(x):
cos(x) + sin(x)*i
ans =
exp(x*i)
```

6. 查找符号变量

在 MATLAB 中，提供了 findsym 函数用于找出一个表达式中存在哪些符号变量，例如，给定由符号变量定义的符号表达式 f 和 g，其中 $f=e^x$，$g=\sin(ax+b)$，那么，使用

findsym(f)和findsym(g)可以分别找出两个表达式中的符号变量，此外，对于任意表达式s，使用findsym(s,n)可以找出表达式s中n个与x接近的变量。

【例3-20】 符号表达式中变量的确定。

```
>> X = sym('[a b c d; g h i j; m n o p; s t v w]')
X =
[ a, b, c, d]
[ g, h, i, j]
[ m, n, o, p]
[ s, t, v, w]
 %返回符号表达式X中的所有符号变量
>> findsym(X)
ans =
a,b,c,d,g,h,j,m,n,o,p,s,t,v,w
 %返回符号表达式X中第一个符号变量
>> findsym(X,1)
ans =
b
 %如果符号表达式X为下面表达式,注意观察MATLAB选取自由变量的规则
>> X = sym('[a b c d; g h i j; m n o p; x t v w]')
X =
[ a, b, c, d]
[ g, h, i, j]
[ m, n, o, p]
[ x, t, v, w]
 %返回符号表达式X中的前两个符号变量
>> findsym(X,2)
ans =
b,a
```

值得注意的是，符号变量用大小写字母表示是有区别的。如果其中的符号变量为大写字母，则其与S的距离大于所有小写字母与S的距离。

7. 替换求值

在MATLAB中，使用subs函数可以将符号表达式中的字符型变量用数值型变量替换。函数的调用格式如下。

subs(s,old,new)：将符号表达式s中的符号变量old用数值型变量或表达式new进行替换。

subs(s,new)：将符号表达式s中的自由符号变量用数值型变量或表达式new进行替换。所有符号变量用工作区中的变量值进行替换。

subs(s)：将符号表达式s中的所有符号变量，用工作区中的变量值进行替换。

【例3-21】 利用subs函数进行替换和求值。

```
>> syms a b
subs(cos(a) + sin(b), [a, b], [sym('alpha'), 2])
ans =
sin(2) + cos(alpha)
```

```
>> syms x y a
syms f(x, y);
f(x, y) = x + y;
f = subs(f, x, a)
f(x, y) =
a + y
```

8. 反函数

反函数运算是符号运算的重要组成部分，在 MATLAB 中，使用 finverse 函数实现对符号函数的反函数运算。函数的调用格式如下。

g = finverse(f)：g 为符号函数 f 的反函数。f 为一符号函数表达式，单变量为 x，则函数 g 为一符号函数，使得 g(f(x))=x。

g = finverse(f,v)：返回的符号函数表达式的自变量为 v，这里 v 为一符号，是表达式的向量变量，则 g 的表达式要使得 g(f(v))=v。当 f 包括不只一个变量时最好使用此型。

【例 3-22】 利用 finverse 函数求符号表达式的反函数。

```
>> clear all;
>> syms t x
>> f1 = finverse(log(t))                          % 反函数，以默认 x 为自变量
f1 =
exp(t)
>> f2 = finverse(cos(2 * t) + 1)
f2 =
acos(t - 1)/2
>> f3 = finverse(exp(t - 2 * x),t)                % 以 t 为自变量
f3 =
2 * x + log(t)
>> f4 = finverse(exp(x - 2 * t), x)
f4 =
2 * t + log(x)
```

9. 复合函数

在科学计算中，经常要遇到求解复合函数的情况，如函数 $z=f(y)$，而该函数的自变量 y 又是另外一个函数，$y=g(x)$，也就是 $z=f(g(x))$，此时，求 z 对 x 的函数的过程就是求解复合函数的过程。

在 MATLAB 中，提供了 compose 函数用于实现复合函数的运算。函数的调用格式如下。

compose(f,g)：返回当 f=f(x)和 g=g(x)时的复合函数 f(g(y))，这里 x 为自定义的函数 f 的符号变量，y 为自定义的函数 g 的符号变量。

compose(f,g,z)：返回自变量为 z 的复合函数。

compose(f,g,x,z)：返回复合函数 f(g(z))且使得 x 为 f 的独立变量。也就是说，如果 f=cos(x/t)，则 compose(f,g,x,z)返回 cos(g(z)/t)，而 compose(f,g,t,z)返回 cos(x/g(z))。

compose(f,g,x,y,z)：返回复合函数 f(g(z))并使得 x 为 f 的独立变量，y 为 g 的独立变量。即如果 f=cos(x/t)，g=sin(y/u)，则 compose(f,g,x,y,z)返回 cos(sin(z/u)/t)，而 compose(f,g,x,u,z)返回 cos(sin(y/z)/t)。

【例 3-23】 利用 compose 函数对给定的表达式进行复合运算。

```
>> clear all;
>> syms x y z t u
f = 1/(1 + x^2);
g = sin(y);
h = x^t;
p = exp( - y/u);
a = compose(f,g)
b = compose(f,g,t)
c = compose(h,g,x,z)
d = compose(h,g,t,z)
e = compose(h,p,x,y,z)
```

运行程序，输出如下。

```
a =
1/(sin(y)^2 + 1)
b =
1/(sin(t)^2 + 1)
c =
sin(z)^t
d =
x^sin(z)
e =
exp( - z/u)^t
```

3.2.3 符号微积分

微积分是数学分析中的一个十分重要的内容，是高等数学建立的基础和整个微分方程体的基础内容。在 MATLAB 中，能够通过符号函数的计算实现微积分运算。

1. 符号极限

极限在高等数学中占有非常重要的地位，是微积分的基础和出发点。极限的定义为：当自变量趋近某个范围或数值时，函数表达式的数值即为此时的极限。无穷逼近的思想也是符号极限中的求解方式之一。因此，要想学好微积分，就必须先了解极限的求法。在 MATLAB 中，提供了 limit 函数来求符号极限，函数的调用格式如下。

limit(expr，x，a)：对 x 求趋于 a 的极限，但是当左右极限不相同时，极限不存在。

limit(expr，a)：用 findsym(expr)作为独立变量。

limit(expr)：对 x 求右趋于 a=0 的极限。

limit(expr，x，a，'left')：对 x 求左趋于 a 的极限。

limit(expr，x，a，'right')：对 x 求右趋于 a 的极限。

函数 limit 要求第一个输入变量为符号函数，limit 不支持符号函数的句柄，但是对符号函数句柄 f，可以将 f(x)作为输入变量。

【例 3-24】 计算符号表达式的极限值。

```
>> %计算双向极限
>> syms x h a
limit(sin(x)/x)
ans =
1
>> limit((sin(x + h) - sin(x))/h, h, 0)
ans =
cos(x)
>> limit(1/x, x, 0, 'right')                    %计算符号表达式自变量从右边趋近于0
ans =
Inf
>> limit(1/x, x, 0, 'left')                     %计算符号表达式自变量从左边趋近于0
ans =
 - Inf
>> v = [(1 + a/x)^x, exp( - x)];
limit(v, x, inf)
ans =
[ exp(a), 0]
```

2. 符号微分

在 MATLAB 中，使用 diff 函数来进行微分和求导运算。使用 jacobian 函数实现对多元符号函数的求导。diff 函数的调用格式如下。

Y = diff(X)：根据由 findsym(X)命令返回的自变量，求表达式 X 的一阶导数。

Y = diff(X,n)：根据由 findsym(X)命令返回的自变量，求表达式 X 的 n 阶导数，n 必须为自然数。

Y = diff(X,'v',n)：根据由 findsym(x)命令返回的自变量 v，计算 X 的 n 阶导数。

【例 3-25】 利用 diff 函数求符号微分。

```
>> clear all;
>> syms x
>> diff(x^3 - 3 * x^2 + x - 2)
ans =
3 * x^2 - 6 * x + 1
>> diff(sin(x^4),5)
ans =
1024 * x^15 * cos(x^4) - 12480 * x^7 * cos(x^4) - 3360 * x^3 * sin(x^4) + 7680 * x^11 *
sin(x^4)
%对多个变量函数中的某个变量求导
>> syms x y f
>> f = x * y - x^2 + sin(y) - cos(x)
f =
sin(y) - cos(x) + x * y - x^2
>> diff(f,y)
```

```
ans =
x + cos(y)
>> diff(f,x)
ans =
y - 2*x + sin(x)
```

jacobian 函数的调用格式如下。

jacobian(f,x)：其中 f 为符号函数组成的列向量，x 为自变量组成的行向量，如果 x 未指定，则默认 x 为 f 中所有变量依字母顺序组成的行向量。

【例 3-26】 已知 $\boldsymbol{F}(x,y,z)=\begin{bmatrix} xyz \\ y \\ x+z \end{bmatrix}$，求 $J=\dfrac{\partial f}{\partial(x,y)}$。

```
>> syms x y z
f = [x*y*z; y; x + z];
v = [x, y, z];
R = jacobian(f)
R =
[ y*z, x*z, x*y]
[   0,   1,   0]
[   1,   0,   1]
>> R = jacobian(f,v)
R =
[ y*z, x*z, x*y]
[   0,   1,   0]
[   1,   0,   1]
>> b = jacobian(x + z, v)
b =
[ 1, 0, 1]
```

3. 符号积分

在高等数学的研究中，对于积分可以细分为不定积分、定积分、旁义积分和重积分等。这些积分过程比微分过程更为难求。符号积分函数简单，但积分时间可能会更长，给出的结果往往比较冗长，如果积分不能给“闭”解时，积分运行结束时将会给出警告信息。在 MATLAB 中，提供了 int 函数实现符号积分运算，函数的调用格式如下。

int(expr, v)：指定以 v 为自变量，对被积函数或符号表达式 expr 求不定积分。

int(expr,v,Name,Value)：对一个或多个属性名及其对应的属性值求符号表达式的不定积分。

int(expr,v, a, b)：指定定积分的下限参数 b 和上限参数 a，该函数求被积函数在区间[a,b]上的定积分。a 和 b 可以是两个具体的数，也可以是一个符号表达式，还可以是无穷(inf)。

int(expr,v, a, b,Name,Value)：对一个或多个属性名及其对应的属性值求符号表达式的定积分。

【例 3-27】 求符号表达式的积分运算。

```
>> syms x t z
>> int( - 2 * x/(1 + x^2)^2)
ans =
1/(x^2 + 1)
>> int(x/(1 + z^2), z)
ans =
x * atan(z)
>> int(x * log(1 + x), 0, 1)
ans =
1/4
>> alpha = sym('alpha');
int([exp(t), exp(alpha * t)])
ans =
[ exp(t), exp(alpha * t)/alpha]
>> int(acos(sin(x)), x)
ans =
x * acos(sin(x)) + (x^2 * sign(cos(x)))/2
```

3.3 多项式运算

多项式作为线性方程组的表现形式，在运算及应用中具有非常重要的意义。

3.3.1 多项式的四则运算

MATLAB 中，n 次多项式是用一个长度为 $n+1$ 的向量表示，缺少的幂次项系数为 0。例如，如下多项式

$$p(x) = a_n x^n + a_{n-1} x^{n-1} + a_{n-2} x^{n-2}, \cdots + a_1 x + a_0$$

在 MATLAB 中表示为相应的向量

$$[a_n, a_{n-1}, a_{n-2}, \cdots, a_1, a_0]$$

1. 加减法

对于次数相同的若干个多项式，可直接对多项式系数向量进行加、减的运算。对多项式加法，MATLAB 不提供一个直接的函数。如果两个多项式向量大小相同，标准的数组加法有效。

【例 3-28】 实现多项式 $a(x)=x^3+2x^2+3x+4$ 与 $b(x)=x^3+4x^2+9x+16$ 的加减运算。

```
>> clear all;
>> a = [1 2 3 4];
>> b = [1 4 9 16];
>> s = a + b                                  % 加运算
s =
     2     6    12    20
>> poly2sym(s)                                % 转换成多项式形式
```

```
ans =
2*x^3 + 6*x^2 + 12*x + 20
>> s2 = a - b                                    % 减运算
s2 =
     0     -2     -6    -12
>> poly2sym(s2)                                  % 转换成多项式形式
ans =
 -2*x^2 - 6*x - 12
```

2. 乘法

函数 conv 支持多项式乘法(执行两个数组的卷积)。函数 conv(P1,P2)用于求多项式 P1 和 P2 的乘积。这里,P1、P2 为两个多项式的系数向量。

【例 3-29】 求多项式 $f(x)=x^4+4x^3-2x^2+7x+11$ 和 $g(x)=9x^4-11x^3+5x^2+8$ 的乘法运算。

```
>> clear all;
f = [1 4 -2 7 11];
g = [9 -11 5 0 8];
>> c = conv(f,g)
c =
     9    25   -57   105    20   -54    39    56    88
>> poly2sym(c)
ans =
9*x^8 + 25*x^7 - 57*x^6 + 105*x^5 + 20*x^4 - 54*x^3 + 39*x^2 + 56*x + 88
```

3. 除法

某些特殊情况下,一个多项式需要除以另一个多项式。在 MATLAB 中,可由 deconv 函数实现。函数的调用格式如下。

[q,r] = deconv(v,u):求多项式 v、u 的除法运算,其中 q 为返回多项式 v 除以 u 的商式,r 为返回 v 除以 u 的余式。返回的 q 与 r 仍为多项式系数向量。

【例 3-30】 多项式的除法运算。

```
>> a = [1   2   3   4];
>> b = [10   20   30 -10];
>> [q,r] = deconv(a,b)                           % 多项式的除运算
q =                                              % 商
    0.1000
r =                                              % 余数
         0         0   -0.0000    5.0000
```

3.3.2 多项式的导数

多项式导数的概念大家很清楚,对于 n 阶多项式 $p(x)=a_nx^n+a_{n-1}x^{n-1}$,其导数为 $n-1$阶多项式,为 $dp(x)=na_nx^{n-1}+(n-1)a_{n-1}x^{n-2}+\cdots+a_1$。原多项式及其导数多项

式表示分别为 $\boldsymbol{p}=[a_n,a_{n-1},\cdots,a_0]$，$dp=[na_n,(n-1)a_{n-1},\cdots a_1]$。

在 MATLAB 中提供了 polyder 函数用于求多项式的导数，其调用格式如下。

k = polyder(p)：p、k 分别为原多项式及导数多项式的多项式表示。

k = polyder(a,b)：求多项式 a 与多项式 b 乘积的导函数多项式。

[q,d] = polyder(b,a)：求多项式 b 与多项式 a 相除的导函数，导函数的分子存入 q，分母存入 d。

其中，参数 a 和 b 是多项式的系数向量，返回结果 q 和 d 是多项式的系数向量。

【例 3-31】 对多项式进行求导。

```
>> clear all;
>> P = [1];
>> Q = [1,0,5];
>> [p,q] = polyder(P,Q)
p =
    -2     0
q =
     1     0    10     0    25
>> g = [1 6 20 48 69 72 44]                          % 多项式
g =
     1     6    20    48    69    72    44
>> h = polyder(g)
h =
     6    30    80   144   138    72
>> num = 10 * [1 2];                                 % 分子
>> den = poly([ - 1; 3; 5]);
>> [b,a] = polyder(num,den)
b =
   -20    10   280    10
a =
     1   -14    63   -68  -161   210   225
```

3.3.3 多项式的求值

MATLAB 提供了两种求多项式值的函数，即 polyval 与 polyvalm，它们的输入参数均为多项式系数向量 P 和自变量 x。两者的区别在于，前者是代数多项式求值，而后者是矩阵多项式求值。

1. 代数多项式求值

polyval 函数用来求代数多项式的值，函数的调用格式如下。

y = polyval(p,x)：p 为多项式的系数向量，x 为矩阵，它是按数组运算规则求多项式的值的。

[y,delta] = polyval(p,x,S)：使用可选的结构数组 S 产生由 polyfit 函数输出的估计参数值，delta 是预测未来的观测估算的误差标准偏差。

y = polyval(p,x,[],mu)或[y,delta] = polyval(p,x,S,mu)：使$\hat{x}=(x-\mu_1)/\mu_2$ 替

代 x，在等式中，μ_1=mean(x)，μ_2=std(x)，其中心点与坐标值 mu=[μ_1，μ_2]可由 polyfit 函数计算得出。

【例 3-32】 求多项式 x^2-x-6 在点 1.5，3，4.5，6 的值。

```
>> clear all;
p = [1 -1 -6];                                  % 多项式的系数
x = 1.5: 1.5: 6;
y = polyval(p,x)                                % 求多项式在 x 处的值
```

运行程序，输出如下。

```
y =
   -5.2500         0    9.7500   24.0000
```

2. 矩阵多项式求值

polyvalm 函数用来求矩阵多项式的值，其调用格式与 polyval 相同，但两者含义不同。polyvalm 函数要求 x 为方阵，它以方阵为自变量求多项式的值。设 A 为方阵，P 代表多项式 x^3-5x^2+8，那么 polyvalm(P,A)的含义如下。

$$A*A*A-5*A*A+8*\mathrm{eye}(\mathrm{size}(A))$$

而 polyval(P,A)的含义为

$$A.*A.*A-5*A.*A+8*\mathrm{ones}(\mathrm{size})$$

【例 3-33】 已知矩阵，分别利用 polyval 及 polyvalm 对多项式求值。

```
>> clear all;
X = pascal(4)                                   % 建立四阶 pascal 矩阵
X =
     1     1     1     1
     1     2     3     4
     1     3     6    10
     1     4    10    20
>> p = poly(X)
p =
    1.0000   -29.0000   72.0000   -29.0000    1.0000
>> P = poly2str(p,'x')
P =
   x^4 - 29 x^3 + 72 x^2 - 29 x + 1
>> y = polyval(p,X)                             % 利用 polyval 求解多项式的值
y =
  1.0e+004 *
    0.0016    0.0016    0.0016    0.0016
    0.0016    0.0015   -0.0140   -0.0563
    0.0016   -0.0140   -0.2549   -1.2089
    0.0016   -0.0563   -1.2089   -4.3779
>> Y = polyvalm(p,X)                            % 利用 polyvalm 求解多项式的值
Y =
  1.0e-010 *
   -0.0013   -0.0063   -0.0104   -0.0242
   -0.0048   -0.0218   -0.0360   -0.0798
```

```
  -0.0115   -0.0512   -0.0822   -0.1812
  -0.0229   -0.0973   -0.1560   -0.3410
```

3.3.4 多项式求根

找出多项式的根(即多项式为零的值),可能是许多学科需要解决的共同问题。MATLAB可求解这个问题,并提供其他的多项式操作工具。在MATLAB中,多项式由一个行向量表示,它的系数按降序排列。

在MATLAB中,提供了roots函数求多项式的根,函数的调用格式如下。

r = roots(c):c为多项式的系数向量,r为返回多项式c的所有根。

【例3-34】 求多项式 $x^4-12x^3+25x+116$ 的根。

```
>> clear all;
>> p=[1 -12 0 25 116]
p =
     1   -12     0    25   116
```

注意:必须包括具有零系数的项。除非特别地辨认,不然MATLAB无法知道哪一项为零。

```
>> r=roots(p)
r =
  11.7473 + 0.0000i
   2.7028 + 0.0000i
  -1.2251 + 1.4672i
  -1.2251 - 1.4672i
```

因为在MATLAB中,无论是一个多项式,还是多项式的根,都是向量,MATLAB按惯例规定,多项式是行向量,根是列向量。给出一个多项式的根,也可以构造相应的多项式。在MATLAB中,利用poly函数完成这个操作。

```
>> pp=poly(r)
pp =
    1.0000   -12.0000    -0.0000   25.0000  116.0000
```

3.3.5 有理多项式

在许多应用中,如傅里叶(Fourier)、拉普拉斯(Laplace)和Z变换中,会出现有理多项式或两个多项式之比。在MATLAB中,有理多项式由它们的分子多项式和分母多项式表示。对有理多项式进行运算的两个函数是residue和polyder。函数residue执行部分分式展开。

residue函数的调用格式如下。

[r,p,k] = residue(b,a):b、a分别为分母与分母多项式系数的行向量,r为留数行向量,p为极点行向量,k为直项行向量。

[b,a] = residue(r,p,k)：p为极点行向量，k为直项行向量，b、a分别为分母与分母多项式系数的行向量。

【例 3-35】 将多项式$\frac{10(s+3)}{(s-1)(s+4)(s+6)}$展开成几个最简单多项式的和。

```
>> clear all;
num = 10 * [1,3];                                    % 分子多项式
den = poly([1; -4; -6]);                              % 分母多项式
[res,poles,k] = residue(num,den)
```

运行程序，输出为

```
res =
   -2.1429
    1.0000
    1.1429
poles =
   -6.0000
   -4.0000
    1.0000
k =
     [ ]
```

结果是余数、极点和部分分式展开的常数项。上面的结果说明了

$$\frac{10(s+3)}{(s-1)(s+4)(s+6)}=\frac{-2.1429}{s+6}+\frac{1.0000}{s+4}+\frac{1.1429}{s-1}+0$$

这个函数也执行逆运算。

```
>> [n,d] = residue(res,poles,k)
n =
         0   10.0000   30.0000
d =
    1.0000    9.0000   14.0000   -24.0000
>> roots(d)
ans =
   -6.0000
   -4.0000
    1.0000
```

3.4 数据插值

插值法是一种古老的数学方法。插值问题的数学定义为

由实验或测量的方法得到函数$y=f(x)$在互异点$x_0,x_1,x_2,\cdots,x_n$处的数值$y_0,y_1,y_2,\cdots,y_n$，然后构造一个函数$\varphi(x)$作为$y=f(x)$的近似表达式，即

$$y=f(x)\approx\varphi(x)$$

使得$\varphi(x_0)=y_0,\varphi(x_1)=y_1,y_2,\cdots,\varphi(x_n)=y_n$。这类问题称为插值问题。$y=f(x)$称为被插值函数，$\varphi(x)$称为插值函数，$x_0,x_1,x_2,\cdots,x_n$称为插值节点。

插值的任务是由已知的观测点为物理量建立一个简单的、连续的解析模型，以便能根据该模型推测物理量在非观测点处的特性。

3.4.1 一维插值

当被插值函数 $y=f(x)$ 为一元函数时，为一维插值。在 MATLAB 中，提供了 interp1 函数实现一维插值，函数的调用格式如下。

yi = interp1(x,Y,xi)：x 必须为向量，Y 可以是向量也可以是矩阵。如果 Y 是向量，则必须与变量 x 具有相同的长度，这时 xi 可以是标量，也可以是向量或者矩阵。

yi = interp1(Y,xi)：在默认情况下，x 变量选择为 1：n，其中 n 是向量 Y 的长度。

yi = interp1(x,Y,xi,method)：输入变量 method 用于指定插值的方法，其取值如下。

(1) method=nearest：最邻近插值(Nearest Neighbor Interpodlation)。这种插值方法在已知数据的最邻近点设置插值点，对插值点的数值进行四舍五入，对超出范围的数据点返回 NaN。

(2) method=linear：线性插值(Linear Interpolation)。这是 method 的默认值。该方法采用直线将相邻的数据点相连，对超出数据范围的数据点返回 NaN。

(3) method=spline：三次样条插值(Cubic Spline Interpolation)。该方法采用三次样条函数获取插值数据点，在已知点为端点的情况下，插值函数至少具有相同的一阶和二阶导数。

(4) method = pchip：分段三次厄米特多项式插值(Piecewise Cubic Hermite Interpolation)。

(5) method=cubic：三次多项式插值，与分段三次厄米特多项式插值方法相同。

(6) method=v5cubic：MATLAB 5 中使用的三次多项式插值。

yi = interp1(x,Y,xi,method,'extrap')：对超出数据范围的插值运算使用外推方法。

yi = interp1(x,Y,xi,method,extrapval)：对超出数据范围的插值数据返回 extrapval 数值，一般为 NaN 或 0。

pp = interp1(x,Y,method,'pp')：返回数值 pp 为数据 Y 的分段多项式形式，method 指定产生分段多项式 pp 的方法。

以上的 method 方法中有以下特点：

(1) Nearest 方法速度最快，占用内存最小，但一般来说误差最大，插值结果最不光滑。

(2) Linear 分段线性插值方法则在速度和误差之间取得了比较好的均衡，其插值函数具有连续性，但在已知数据点处的斜率一般都会改变，因此不是光滑的。

(3) Spline 三次样条插值方法是所有插值方法中运行耗时最长的，其插值函数及插值函数的一阶、二阶导数函数都连续，因此是最光滑的插值方法，占用内存上比 Cubic 方法小，但在已知数据点不均匀分布时可能会出现异常结果。

(4) Cubic 三次多项式插值法中插值函数及其一阶导数都是连续的，因此其插值结

果也比较光滑，运算速度比 Spline 方法稍快，但占用内存最多。在实际应用中，应根据实际需求和运算条件选择合适的算法。

下面介绍两种较为常用的一维插值方法。

1. 分段线性插值(Linear)

分段线性插值的算法是在每个小区间$[x_i, x_{i+1}]$上采用简单的线性插值。在区间$[x_i, x_{i+1}]$上的子插值多项式为

$$F_i = \frac{x - x_{i+1}}{x_i - x_{i+1}} f(x_i) + \frac{x - x_i}{x_{i+1} - x_i} f(x_{i+1})$$

在整个区间$[x_i, x_n]$上的值函数为

$$F(x) = \sum_{i=1}^{n} F_i l_i(x)$$

其中，$l_i(k)$的定义为

$$l_i(x) = \begin{cases} \dfrac{x - x_{i-1}}{x_i - x_{i-1}} & x \in [x_{i-1}, x_i](i = 0) \\ \dfrac{x - x_{i+1}}{x_i - x_{i+1}} & x \in [x_i, x_{i+1}](i = 0) \\ 0 & x \notin [x_{i-1}, x_{i+1}] \end{cases}$$

【例 3-36】 采用 interp1 函数对 $y=\sin x$ 进行分段线性插值。

```
>> clear all;
x = 0: 2 * pi;
y = sin(x);
xx = 0: 0.5: 2 * pi;
yy = interp1(x, y, xx);
plot(x, y, 'rs', xx, yy);
```

运行程序，效果如图 3-1 所示。

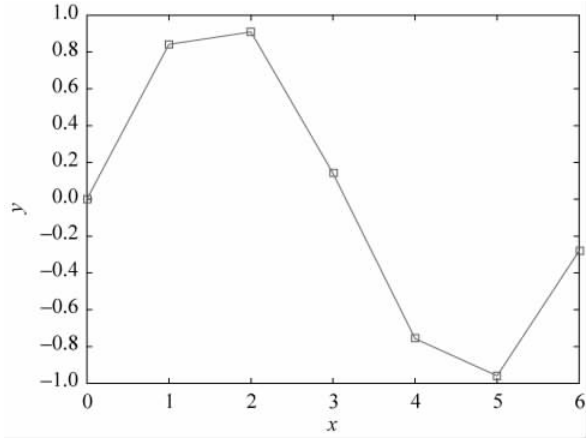

图 3-1 分段线性插值效果

2. 一维快速傅里叶插值

一维快速傅里叶插值通过函数 interpft 实现。该函数用傅里叶变换把输入数据变换到频域，然后用更多点的傅里叶逆变换变回时域，其结果是对数据进行增采样。函数的调用格式如下。

y = interpft(x,n)：对 x 进行傅里叶变换，然后采用 n 点傅里叶反变换，变回到时域。如果 x 为一个向量，数据 x 的长度为 m，采样间隔为 dx，则数据 y 的采样间隔为 dx * m/n，其中 n 必须大于 m。如果 x 为矩阵，该函数对矩阵 x 的列进行操作，则其返回的结果 y 与 x 具有相同的列，行数为 n。

注意：n 必须大于 m。如果 x 为矩阵，则函数操作在 x 的列上，返回结果与 x 具有相同的列数，但其行数为 n。

y = interpft(x,n,dim)：在 dim 指定的维度上进行操作。

【例 3-37】 利用一维快速傅里叶插值实现数据增采样。

```
>> clear all;
y = [0 .5 1 1.5 2 1.5 1 .5 0 -.5 -1 -1.5 -2 -1.5 -1 -.5 0];   % 插值数据
N = length(y);
L = 5;
M = N*L;                                                      % 将采样提到了 5 倍
x = 0: L: L*N-1;
xi = 0: M-1;
yi = interpft(y,M);                                           % 采用一维快速傅里叶插值
plot(x,y,'o',xi,yi,'r: ')
legend('原始数据','插值后数据');
```

运行程序，效果如图 3-2 所示。

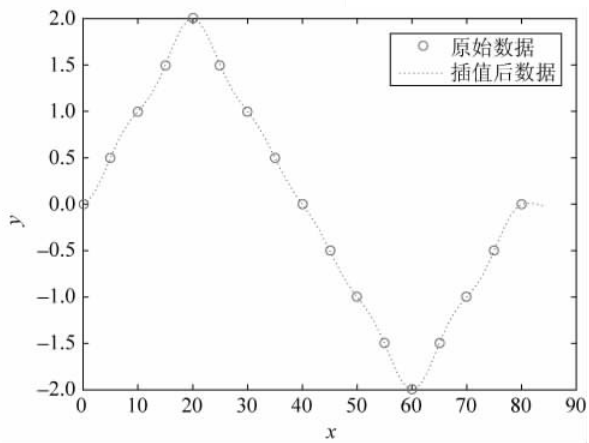

图 3-2 一维快速傅里叶插值

3. 快速傅里叶算法

当数据点呈现周期分布时，用上面的几种插值算法效果都不是很好，这时可以使用

interpft 函数进行插值，此函数使用快速傅里叶算法作一维插值，其调用格式如下。

```
y = interpft(x,n)
```

提示：它返回周期函数在重采样的 n 个等距点的插值。注意 n 必须大于 x 的长度。

【例 3-38】 采用 interpft 函数对 sinx 函数插值。

```
>> clear all;
x = 0: 2 * pi;
y = sin(x);
z = interpft(y,15);
xx = linspace(0,2 * pi,15);                    % 生成 0～2 * pi 之间的 15 个线性等分点
plot(x,y,'r - o',xx,z,': s');
```

运行程序，效果如图 3-3 所示。

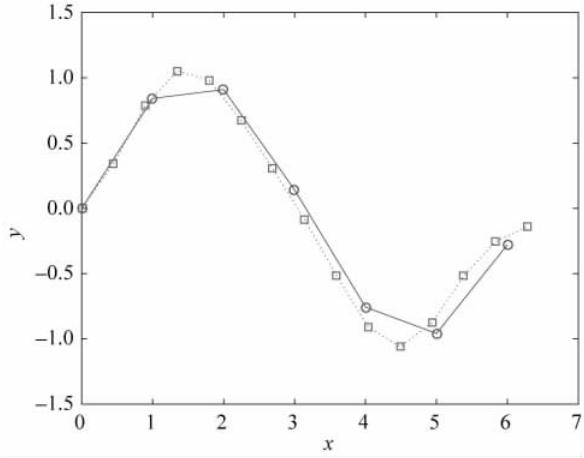

图 3-3 快速傅里叶插值效果

3.4.2 二维插值

二维插值是高维插值的一种，主要应用于图像处理和数据的可视化，其基本思想与一维插值大致相同，它是对两个变量的函数 $z=f(x,y)$ 进行插值。

在 MATLAB 中，提供了 interp2 函数实现二值插值。函数的调用格式如下。

ZI = interp2(X,Y,Z,XI,YI)：原始数据 X、Y 与 Z 决定插值函数 Z=f(X,Y)，返回的数值 ZI 是(XI,YI)在函数 Z=f(X,Y)上的数值。

ZI = interp2(Z,XI,YI)：如果 Z 的维度是 n×m，则有 X=1：n，Y=1：m。

ZI = interp2(Z,ntimes)：在两点之间递归地进行 ntimes 次插值。

ZI = interp2(X,Y,Z,XI,YI,method)：使用选定的插值方法进行二维插值，method 的取值如下。

- mehod=nearest：最近邻插值法，将插值点周围的四个数据点中离该插值点最近的数据点函数值作为该插值点的函数值估计。

- method=linear：双线性插值，将插值点周围四个数据点的函数值线性组合作为插值点的函数值估计。双线性插值是 interp2 的默认项。
- method=cubic：双立方插值，该方法利用插值点周围的 16 个数据点，相对于前两种方法，双立方插值得到的曲面更加光滑，但是也消耗更多的内存和时间。
- mehod=spline：三次样条插值，该方法是实际过程中经常使用的插值方法，得到的曲面光滑，并且具有很高的效率。

ZI = interp2(…,method, extrapval)：进行外插值。

关于上面命令中的参数，应注意以下的内容：

(1) 对于其他高维插值的命令，使用格式与此大致相同。

(2) 在上面的函数中，X、Y 与 Z 是进行插值的基准数据。X1、Y1 是待求插补函数 Z1 的自变量对组，虽然随着维度的增加，插值的具体操作变得越来越复杂，但是基本原理和一维插值的情况相同，调用名称也类似。

(3) 参数 X 和 Y 必须满足一定的条件，首先 X 和 Y 必须是同维矩阵，同时矩阵中的元素，无论是行向还是列向，都必须是单调排列。最后，X 和 Y 必须是 Plaid 格式的矩阵，所谓 Plaid 格式的矩阵，是指 X 矩阵每一行的元素依照单调次序排列，并且任何两行都是相同的；Y 矩阵每一列的元素依照单调次序排列，并且任何两列都是相同的。

(4) 对于 X1 和 Y1 数据系列，一般需要使用 meshgrid 函数创建。具体的方法为：给定两个量的分向量，然后通过 meshgrid 函数产生对应的数据系列。

【例 3-39】 利用 interp2 对给出的数据进行不同二维网格插值。

```
>> clear all;
[X,Y] = meshgrid(-3:.25:3);
Z = peaks(X,Y);
[XI,YI] = meshgrid(-3:.125:3);
Z1 = interp2(X,Y,Z,XI,YI);
subplot(2,2,1); mesh(X,Y,Z);
hold on;
mesh(XI,YI,Z1+15)
axis([-3 3 -3 3 -5 20]);
title('二维网格双线性插值');
Z2 = interp2(X,Y,Z,XI,YI,'nearest');
subplot(2,2,2); mesh(X,Y,Z);
hold on;
mesh(XI,YI,Z2+15)
axis([-3 3 -3 3 -5 20]);
title('二维网格最近邻插值');
Z3 = interp2(X,Y,Z,XI,YI,'cubic');
subplot(2,2,3); mesh(X,Y,Z);
hold on;
mesh(XI,YI,Z3+15)
axis([-3 3 -3 3 -5 20]);
title('二维网格双三次插值');
Z4 = interp2(X,Y,Z,XI,YI,'spline');
subplot(2,2,4); mesh(X,Y,Z);
hold on;
```

```
mesh(XI,YI,Z4 + 15)
axis([ - 3 3  - 3 3  - 5 20]);
title('二维网格样条插值');
set(gcf,'Color','w');
```

运行程序,效果如图 3-4 所示。

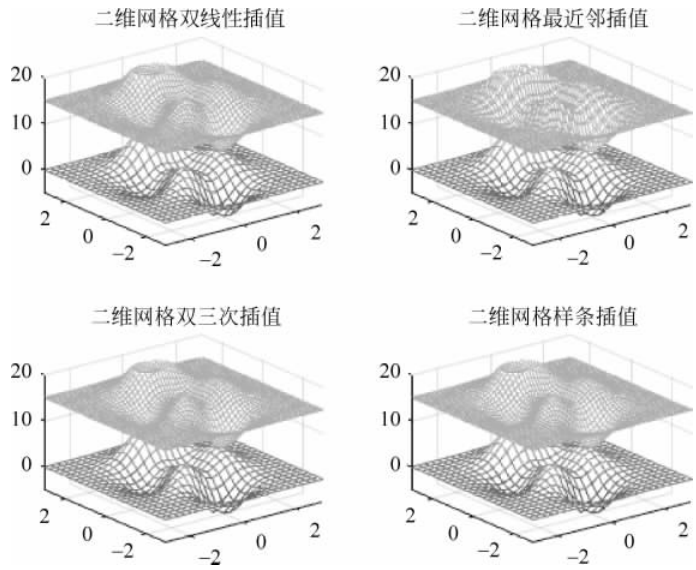

图 3-4　二值插值效果

下面通过一个实际的应用例子演示二维插值的实际应用。

【例 3-40】 假设有一组海底深度数据,采用插值方式绘制海底形状图。

```
% 用以下模型产生一组分度稀疏的"海底深度测量数据"
>> randn('state',2);                                    % 为仿真的可重复性而设
x = - 5: 5;
y = - 5: 5;
[X,Y] = meshgrid(x,y);                                  % 产生"经纬"矩阵
% 以下两条指令本可写成一条,因 Notebook 无法正确处理续行号
zz = 1.2 * exp( - ((X - 1).^2 + (Y - 2).^2)) - 0.7 * exp( - ((X + 2).^2 + (Y + 1).^2));
Z = - 500 + zz + randn(size(X)) * 0.05;                 % 使用带标准差为 0.05 的随机误差
surf(X,Y,Z);
view( - 25,25);
```

运行程序,效果如图 3-5 所示。

通过插值绘制更细致的海底图,代码为:

```
>> xi = linspace( - 5,5,50);
yi = linspace( - 5,5,50);
[XI,YI] = meshgrid(xi,yi);
ZI = interp2(X,Y,Z,XI,YI,' * cubic');
surf(XI,YI,ZI);
view( - 25,25);
```

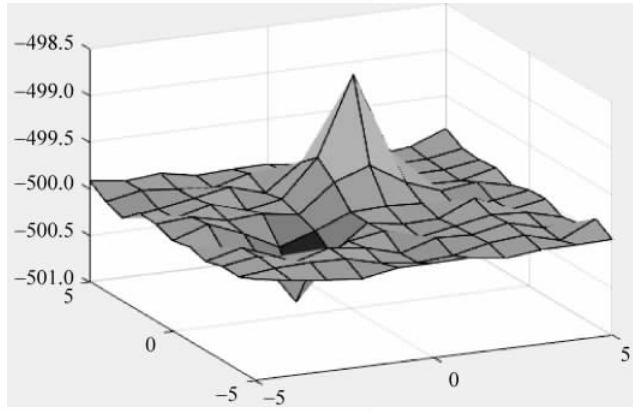

图 3-5　根据基准数据绘制的曲面图

运行程序，效果如图 3-6 所示。

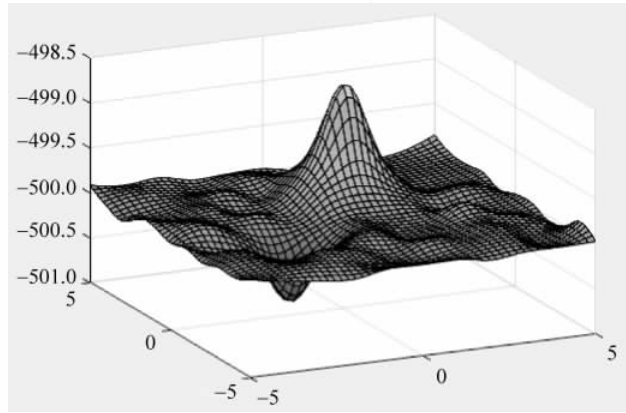

图 3-6　由插值数据生成的曲面

3.4.3　三次样条插值

在生产和科学实验中，所作的插值曲线既要简单，又要在曲线的连接处比较光滑，即所作的分段插值函数在分段上要求多项式次数低，而在节点上不仅连续，还应存在连续的低阶导数，把满足这样条件的插值函数称为样条插值函数。它所对应的曲线称为样条曲线，其节点称为样点，这种插值方法称为样条插值。

目前，有多种消除病态的方法。在这些方法中，三次样条是最常用的一种。在 MATLAB 中，实现基本的三次样条插值的函数有 spline、ppval、mkpp 和 unmkpp。首先给出三次样条函数的定义。

设对 $y=f(x)$在区间$[a,b]$上给定一组节点 $a=x_0<x_1<x_2<\cdots<x_n=b$ 和相应的函数值 $y_0,y_1,y_2,\cdots,y_n$，如果 $s(x)$具有如下性质：

(1) 在每个子区间$[x_{i-1},x_i](i=1,2,\cdots,n)$上 $s(x)$是不高于三次的多项式。

(2) $s(x)$、$s'(x)$、$s''(x)$在$[a,b]$上连续，满足(1)、(2)则称 $s(x)$为三次样条函数。

(3) 如果再有 $s(x_i)=y_i(i=0,1,2,\cdots,n)$，则称 $s(x)$ 为 $y=f(x)$ 的三次样条插值函数。

在三次样条中，要寻找三次多项式，以逼近每对数据点间的曲线。在样条语句中，这些数据点称为断点。因为两点只能决定一条直线，而在两点间的曲线可用无限多的三次多项式近似，因此，为使结果具有唯一性，在三次样条中，增加了三次多项式的约束条件。通过限定每个三次多项式的一阶和二阶层数，使其在断点处相等，就可以较好地确定所有内部三次多项式。此外，近似多项式通过这些断点的斜率和曲率是连续的。然而，第一个和最后一个三次多项式在第一个和最后一个断点以外，没有伴随多项式。因此，必须通过其他方法确定其余的约束。最常用的方法(也是函数 spline 所采用的方法)就是采用非扭结(not-a-knot)条件。这个条件强迫第一个和第二个三次多项式的三阶导数相等，对最后一个和倒数第二个三次多项式也作同样的处理。

基于上述描述，人们可能猜想到，寻找三次样条多项式需要求解大量的线性方程。实际上，给定 N 个断点，就要寻找 $N-1$ 个三次多项式，每个多项式有 4 个未知系数。这样，所求解的方程组就包含有 $4*(N-1)$ 个未知数。而每个三次多项式的特殊形式通过求解 N 个具有 N 个未知系数的方程组，并且运用各种约束就能确定三次多项式。这样，如果有 50 个断点，就有 50 个具有 50 个未知系数的方程组。

三次样条函数的调用格式如下。

yy = spline(x,y,xx)：用三次样条插值计算出由向量 x 与 y 确定的一元函数 y=f(x)在点 xx 处的值。若参量 y 是一矩阵，则以 y 的每一列和 x 配对，再分别计算由它们确定的函数在点 xx 处的值，故 yy 是一阶数为 length(xx) * size(y,2)的矩阵。

pp = spline(x,y)：返回由向量 x 与 y 确定的分段样条多项式的系数矩阵 pp，它可用于命令 ppval、unmkpp 的计算。

yi=ppval(pp,xi)：计算 xi 中各点的分段多项式。

[break,coefs,nopolys,ncoefs]=unmkpp(pp)：分解分段多项式的表示。

pp=mkpp(break,coefs)：形成分段多项式的表示。

最简单的用法：spline 获取数据 x 和 y 以及期望值 xi，寻找拟合 x 和 y 的三次样条内插多项式，然后计算这些多项式，对每个 xi 的值寻找相应的 yi。例如，

```
>> x = 0: 12;
>> y = tan(pi * x/25);
>> xi = linspace(0,12);
>> yi = spline(x,y,xi);
>> plot(x,y,'o',xi,yi);
>> title('样条拟合');
```

运行程序，效果如图 3-7 所示。

这种方法适合于只需要一组内插值的情况。不过，如果需要从相同数据集中获取另一组内插值，再次计算三次样条系数是没有意义的。在这种情况下，可以调用仅带前两个参量的 spline 函数。

当采用这种方式调用时，spline 返回一个称为三次样条的 pp 形式或分段多项式形式的数组。这个数组包含了对于任意一组所期望的内插值和计算三次样条所必需的全部

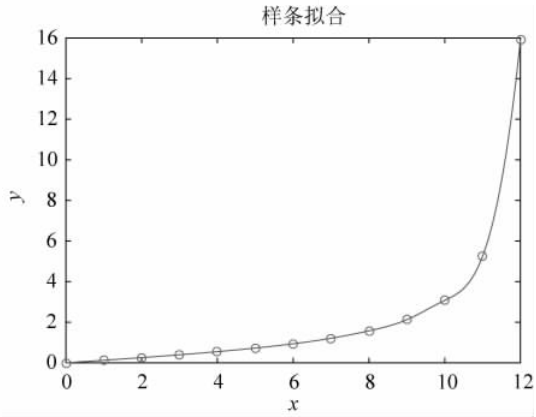

图 3-7　分段多项式拟合

信息。给定 pp 形式，函数 ppval 可计算该三次样条。

【例 3-41】 利用 spline 函数对给定的数据进行三次样条插值。

```
>> clear all;
x = pi*[0:.5:2];
y = [0  1  0 -1  0  1  0;1  0  1  0 -1  0  1];
pp = spline(x,y)                                    %产生插值函数
yy = ppval(pp, linspace(0,2*pi,101));
plot(yy(1,:),yy(2,:),'-b',y(1,2:5),y(2,2:5),'or');
axis equal
```

运行程序，输出如下，效果如图 3-8 所示。

```
pp =
      form: 'pp'
    breaks: [0 1.5708 3.1416 4.7124 6.2832]
     coefs: [8x4 double]
    pieces: 4
     order: 4
       dim: 2
```

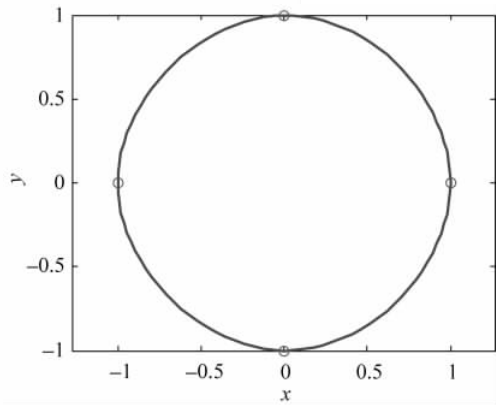

图 3-8　利用样条插值绘图

3.4.4 高维插值

高维插值指的是二维、三维的数据插值，前面已经介绍了二维插值，现在介绍三维插值。在MATLAB中，提供了interp3函数用于实现三维插值。函数的调用格式如下。

VI=interp3(X,Y,Z,V,XI,YI,ZI)：求出由参量X、Y、Z决定的三元函数V=V(X,Y,Z)在点(XI,YI,ZI)的值。参量XI,YI,ZI是同型阵列或向量。若向量参量XI、YI、ZI是不同长度、不同方向(行或列)的向量，则这时输出参量VI与Y1、Y2、Y3为同型矩阵。Y1、Y2、Y3为用函数meshgrid(XI,YI,ZI)生成的同型阵列。若插值点(XI,YI,ZI)中有位于点(X,Y,Z)之外的点，则相应地返回特殊变量值NaN。

VI=interp3(V,XI,YI,ZI)：默认的，X=1：N,Y=1：M,Z=1：P,其中,[M,N,P]=size(V),再按上面的情形计算。

VI=interp3(V,n)：作n次递归计算，在V的每两个元素之间插入它们的三维插值。这样，V的阶数将不断增加。interp3(V)等价于interp3(V,1)。

VI=interp3(…,method)：用指定的算法method作插值计算。linear为线性插值(默认算法)，cubic为三次插值，spline为三次样条插值，nearest为最邻近插值。

【例3-42】 在生成的水流数据处利用interp3进行三维插值。

```
>> clear all;
[x,y,z,v] = flow(15);                          % 创建水流效果
[xi,yi,zi] = meshgrid(.1: .25: 10, -3: .25: 3, -3: .25: 3);
vi = interp3(x,y,z,v,xi,yi,zi);                % vi 为一个 25×40×25 的数组
slice(xi,yi,zi,vi,[6 9.5],2,[-2 .2]);
shading flat
set(gcf,'color','w')
```

运行程序，效果如图3-9所示。

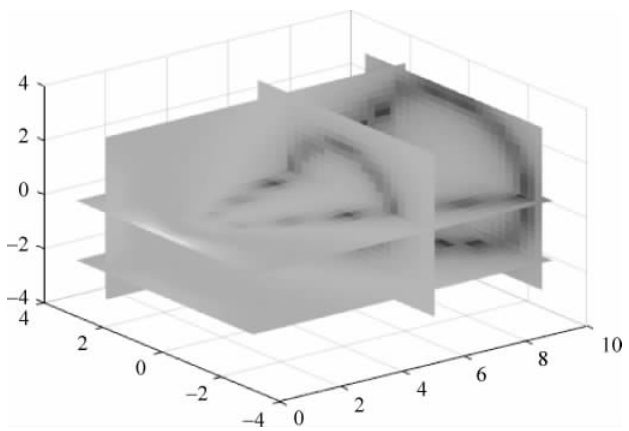

图3-9 三维插值图

3.5 数据拟合

在科学和工程领域，曲线拟合的主要功能是寻求平滑的曲线来最好地表现带有噪声的测量数据，从这些测量数据中寻求两个函数变量之间的关系或变化趋势，最后得到曲线拟合的函数表达式 $y=f(x)$。

由于认为在进行曲线拟合时所有测量数据中已经包含了噪声，因此，最后的拟合曲线并不要求通过每一个已知数据点，衡量拟合数据的标准是整体数据拟合的误差最小。一般情况下，MATLAB 的曲线拟合方法使用的是“最小方差”函数，其中方差的数值是拟合曲线和已知数据之间的垂直距离。

3.5.1 多项式拟合

一般多项式拟合的目标是找出一组多项式系数 $a_i(i=1,2,\cdots,n+1)$，使得多项式 $\psi(x)=a_1x^n+a_2x^{n+1}+\cdots+a_nx+a_{n+1}$ 能够较好地拟合原始数据。在 MATLAB 中提供了 polyfit 函数用于计算多项式拟合系数，其设定曲线拟合的目标是最小方差(Leat Squares)或者被称为最小二乘法，polyfit 函数的调用格式如下。

p = polyfit(x,y,n)：对 x 与 y 进行 n 维多项式的曲线拟合，输出结果 p 为含有n+1个元素的行向量，该向量以维数递减的形式给出拟合多项式的系数。

[p,S] = polyfit(x,y,n)：结果中的 S 包括 R、df 与 normr，分别表示对 x 进行 QR 分解的三角元素、自由度、残差。

[p,S,mu] = polyfit(x,y,n)：在拟合过程中，首先对 x 进行数据标准化处理，以在拟合中消除量纲等的影响，mu 包含两个元素，分别是标注化处理过程中使用的 x 的均值与标准差。

与 polyfit 函数配合使用的函数是 polyval，这个函数根据拟合出来的多项式系数 p 计算给定数据 x 处的 y 值。函数的调用格式如下。

y = polyval(p,x)：y 是根据多项式系数 p 计算出来的 x 处的多项式值。

[y,delta] = polyval(p,x,S)：delta 是利用结构体 S 计算出来的误差估计，y 的95%置信区间为[y−delta,y+delta]，其中假设 polyfit 函数数据输入的误差是独立正态的，并且方差为常数。

y=polyval(p,x,[],mu)或[y,delta]= polyval(p,x,S,mu)：使 $\hat{x}=(x-\mu_1)/\mu_2$ 替代 x，在等式中，μ_1=mean(x)，μ_2=std(x)，其中心点与坐标值 mu=[μ_1,μ_2]可由 polyfit 函数计算得出。

【例 3-43】 使用 polyfit 函数进行多项式的数值拟合，并分析曲线拟合的误差情况。

```
>> clear all;
x = (0: 0.1: 5)';
y = erf(x);
% 计算多项式拟合的参数
[p,s] = polyfit(x,y,5);
[yp,delta] = polyval(p,x,s);
```

```
plot(x,y,'rp',x,yp,'.',x,yp + 2 * delta,' - .',x,yp - 2 * delta);
grid on;
axis([0 5 0 1.2]);
legend('原始曲线','拟合曲线','yp + 2 * delta','yp - 2 * delta');
```

运行程序,效果如图 3-10 所示。

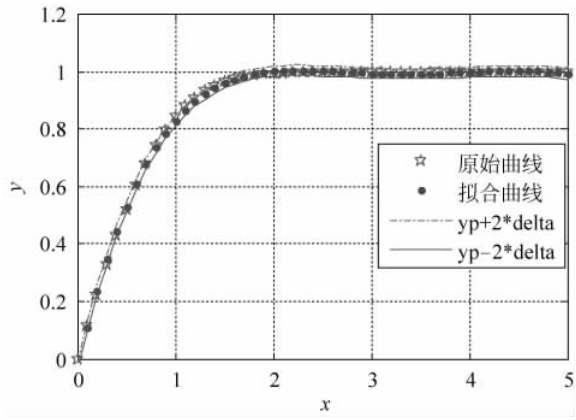

图 3-10　多项式拟合效果图

3.5.2　最小二乘拟合

逼近实验数据的基本方法就是拟合,其中最常用的就是最小二乘拟合。最小二乘拟合适用的数学模型可以是线性函数与非线性函数。对于一组给定的数据(x_i,y_i),设计与确定一个函数模型 $y=f(x)$,使得函数 $f(x)$在某种误差函数标准下与左右数据点(x_i,y_i)之间的距离最近,也就是寻找一个最佳拟合函数。

在 MATLAB 中,提供了 lsline 函数实现线性最小二乘拟合。函数的调用格式如下。

lsline:在当前坐标轴中叠加绘制每个散点的最小二乘拟合曲线。

h = lsline:返回最小二乘拟合曲线的句柄值 h。

【例 3-44】 利用 lsline 对给定包含噪声的信号进行拟合。

```
>> clear all;
x = 1: 10;
y1 = x + randn(1,10);                    % 包含噪声
scatter(x,y1,25,'b','*')                 % 散点图
hold on
y2 = 2 * x + randn(1,10);
plot(x,y2,'mo')
y3 = 3 * x + randn(1,10);
plot(x,y3,'rx: ')
y4 = 4 * x + randn(1,10);
plot(x,y4,'g + -- ')
lsline
xlabel('x'); ylabel('y');
```

运行程序,效果如图 3-11 所示。

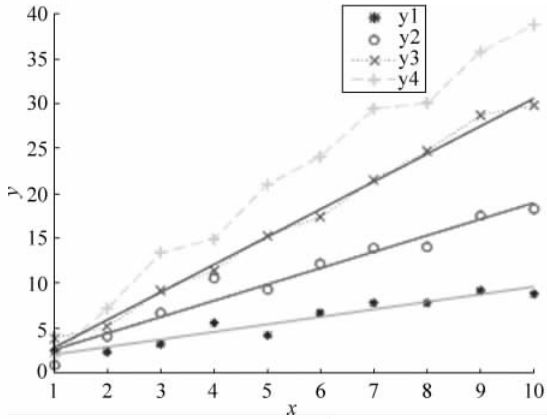

图 3-11　线性最小二乘拟合效果

3.5.3　非线性最小二乘拟合

非线性最小二乘可以应用于曲线拟合问题，对于给定的一系列的散点 $P_1(x_1,y_1)$，$P_2(x_2,y_2)$，…，$P_n(x_n,y_n)$，选择拟合曲线 $y=f(x)$描述散点所确定的曲线形状。对于拟合效果的衡量可以使用以下目标函数描述：

$$\min_x \frac{1}{2}\| f(x_i)-y_i \|_2^2 = \min_x \sum_i \frac{1}{2}[f(x_i)-y_i]^2$$

使用非线性最小二乘求解函数 lsqnonlin 或者函数 lsqcurvefit 即可进行求解。lsqcurvefit 函数的调用格式如下。

x = lsqcurvefit(fun,x0,xdata,ydata)：fun 为拟合函数；(xdata,ydata)为一组观测数据，满足 ydata=fun(xdata,x)；以 x0 为初始点求解该数据的拟合问题。

x = lsqcurvefit(fun,x0,xdata,ydata,lb,ub)：以 x0 为初始点求解该数据拟合问题，lb、ub 为向量，分别是变量 x 的下界与上界。

x = lsqcurvefit(fun,x0,xdata,ydata,lb,ub,options)：options 为指定优化参数。

[x,resnorm] = lsqcurvefit(…)：在上面命令功能的基础上，输出变量 resnorm=$\| r(x) \|_2^2$。

[x,resnorm,residual] = lsqcurvefit(…)：输出变量 residual=r(x)。

[x,resnorm,residual,exitflag] = lsqcurvefit(…)：exitflag 为终止迭代的条件信息。

[x,resnorm,residual,exitflag,output] = lsqcurvefit(…)：output 为输出的关于变量的信息。

[x,resnorm,residual,exitflag,output,lambda] = lsqcurvefit(…)：lambda 为输出的 Lagrange 乘子。

[x, resnorm, residual, exitflag, output, lambda, jacobian] = lsqcurvefit(…)：jacobian 为输出的在解 x 处的 Jacobian 矩阵。

【例 3-45】 已知数据(x_i,y_i)满足 $y_i=0.12e^{-0.213x_i}+0.54e^{-0.17x_i}\sin(1.23x_i)$，其中 $x_i=10(i-1)/100$，$i=1,2,\cdots,101$。并已知该数据满足函数原型 $y(x)=a_1e^{-a_2x}+a_3e^{-a_4x}\sin(a_5x)$，其中 a_i 为待定系数。

采用非线性最小二乘曲线拟合的目的是获得这些待定系数，使得目标函数的值最小。

```
>> clear all;
%根据原型函数建立的内联函数
f = inline('a(1) * exp( - a(2) * x) + a(3) * exp( - a(4) * x). * sin(a(5) * x)','a','x');
x = 0: 0.1: 10;
y = 0.12 * exp( - 0.213 * x) + 0.54 * exp( - 0.17 * x). * sin(1.23 * x);
[x,resnorm] = lsqcurvefit(f,[1,1,1,1,1],x,y);
x',resnorm
ans =
    0.1200
    0.2130
    0.5400
    0.1700
    1.2300
resnorm =
   1.7928e - 016
```

可以看出这样得出的待定系数精度较高，接近于理论值 a＝[0.12 0.213 0.54 0.17 1.23]'。如果想进一步提高精度，则需要修改最优化的选项，这时函数的调用格式也将发生变化。

```
>> ff = optimset;
ff.TolFun = 1e - 20;
ff.TolX = 1e - 15;                                    %修改精度限制
[x,resnorm] = lsqcurvefit(f,[1,1,1,1,1],x,y,[],[],ff);
x',resnorm
ans =
    0.1200
    0.2130
    0.5400
    0.1700
    1.2300
resnorm =
          0
>> x1 = 0: 0.01: 10; y1 = f(xx,x1);
plot(x1,y1,x,y,'rp');
legend('拟合曲线','样本点');
```

运行程序，效果如图 3-12 所示。

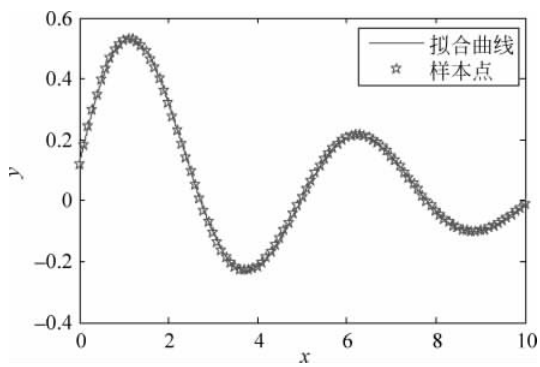

图 3-12　非线性曲线拟合效果