矩阵代数 Matlab 实验

丁广太等

上海大学计算机工程与科学学院

2020 年 4 月

目录

Matlab 7.6 简单入门

Matlab 是 Mathworks 公司推出的科技应用软件.

1 启动

点击 Matlab/bin/matlab.exe，打开 Matlab 的工作窗(或指令窗)

2 Matlab 的使用方式

①指令行操作之直接交互工作方式;

②使用 Matlab 编程语言之程序设计方式.

3 交互方式的使用

在 Matlab 工作窗（如图 1 所示，Command Window）中一般输入以下三种指令行:

①命令

②表达式

③赋值语句:变量=表达式

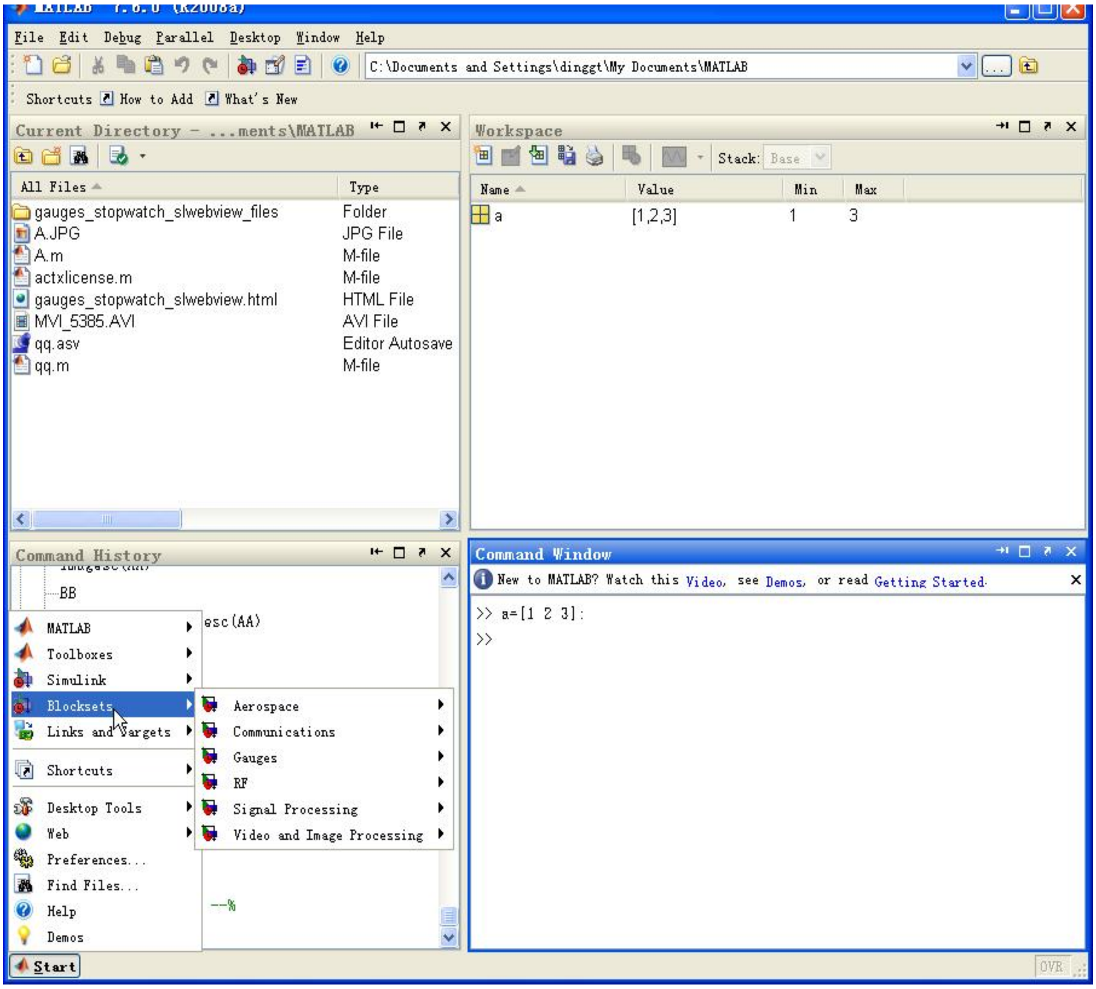


图 1 Matlab 工作窗

4 Matlab 的程序工作方式

step 1. File→New→M-file %打开 Matlab 程序工作窗 Editor/Debugger

step 2. 编写 Matlab 程序；Tools→Run

5 基本命令

```
quit        %退出 Matlab
clc         %清除指令窗口
clear       %从内存中清除变量和函数
clf         %清除当前图
pack        %合并工作内存中的碎块
dir         %列出文件
cd          %改变或显示当前工作目录
disp        %显示矩阵和文字内容
size        %确定矩阵的维数
demo        %演示程序
help        %在线帮助
delete      %删除文件
whos        %列出工作内存中的变量细节
xpimage     %图像处理性能的演示
imagedemo   %同上
```

6 基本函数

```
figure       %创建图形窗口
image        %创建图形窗口
imshow       %显示图像
colormap     %设置配色图
rgb2hsv      %将 rgb 图像转换成 hsv 图像
hsv2rgb      %将 hsv 图像转换成 rgb 图像
rgb2ycbcr    %将 rgb 图像转换成 ycbcr 图像
ycbcr2rgb    %将 ycbcr 图像转换成 rgb 图像
abs          %幅值
fft2         %二维快速 Fourier 变换
ifft2        %二维快速 Fourier 反变换
log          %自然对数
dct2         %二维快速余弦变换
idct2        %二维快速余弦反变换
image        %与 imshow 相似
imfinfo      %显示图像参数
```

7 数据类型及运算

1)Matlab 的基本数据类型为矩阵(实数皆看成 1× 1 的矩阵)，矩阵的基本运算同线性代数

```
A+B     %矩阵相加
A-B     %矩阵相减
A*B     %矩阵相乘
A.*B    %矩阵对应元素相乘
```

2)标量与矩阵进行运算的规定

s+B=sE+B

s-B=sE-B

B-s=B-sE

s*B=sE*B

3)其他运算

inv(B)=B^{-1}

A^n=A^n，A.^n=(a_{ij}^n)

exp(A)=(exp(a_{ij}))

log(A)=(log(a_{ij}))

f(A)=(f(a_{ij}))

A’=A'

4）矩阵的输入

直接输入

```
A=[1 2 3;
   4 5 6;
   7 8 9]
```

矩阵编辑器

edit A

5)指令行结果的输出

(有如下三个要点)

指令行后有分号，不输出结果

指令行后无分号，输出运算结果

表达式后按回车，则 ans=之后，给出结果

6)冒号运算符

设 A 是 m*n 矩阵

B=A(:,r) %由第 r 列元素组成的矩阵

B=A(s,:) %由第 s 行元素组成的矩阵

B=A(s1:s2,r1:r2) %取 A 的子矩阵

B=A([1 3 5],:) %由 A 的 1,3,5 行组成的矩阵

7)给矩阵作标志

L=X<=0.5; %标志矩阵 X 中其值小于 0.5 的元素的位置(小于 0.5 的元素替换为 1，其余为零,作成一个新矩阵)

8)Laplacian 算子

del2( )%五点离散拉普拉斯算子

8 Matlab 程序控制流

1)注释

%号为串首元素的一行字符串

2)循环结构

for-end	while-end

for 循环变量=循环初值：增量值：循环终值
    语句组
end
	while　逻辑表达式
	    语句组
	end

3)分支结构

if-else-end	switch-case-end

if 逻辑表达式
   语句体 1
else
   语句体 2
end
	switch 表达式
	case value1
	   语句体 1
	case valueN
	   语句体 N
	otherwise
	   语句体 N+1
	end

实验一 矩阵的基本运算

1.1 矩阵的输入

通过键盘输入数据创建一个矩阵。

```
A=[ 1 2 3 4;2 3 4 5;6 7 8 9];
disp(A)
>> A=[ 1 2 3 4;2 3 4 5;6 7 8 9];
disp(A)
     1     2     3     4
     2     3     4     5
     6     7     8     9

B=[0.1  0.2  0.3
     -1  2   3
       1  1   1];
disp(B)
>> B=[0.1  0.2  0.3
-1  2   3
1  1   1];
disp(B)
    0.1000    0.2000    0.3000
   -1.0000    2.0000    3.0000
    1.0000    1.0000    1.0000
```

注：蓝色表示在控制台运行的结果（命令行方式）

1.2 矩阵的生成

(1) 特殊矩阵

- 单位矩阵

```
I=eye(5)
I=eye(3,5)
>> I=eye(5)
I =
     1     0     0     0     0
     0     1     0     0     0
     0     0     1     0     0
     0     0     0     1     0
     0     0     0     0     1
>> I=eye(3,5)
I =
     1     0     0     0     0
     0     1     0     0     0
     0     0     1     0     0
```

eye 的帮助文档

```
>> help eye
 EYE Identity matrix.
    EYE(N) is the N-by-N identity matrix.

    EYE(M,N) or EYE([M,N]) is an M-by-N matrix with 1's on
    the diagonal and zeros elsewhere.

    EYE(SIZE(A)) is the same size as A.

    EYE with no arguments is the scalar 1.

    EYE(M,N,CLASSNAME) or EYE([M,N],CLASSNAME) is an M-by-N matrix with 1's
    of class CLASSNAME on the diagonal and zeros elsewhere.

    Note: The size inputs M and N should be nonnegative integers.
    Negative integers are treated as 0.

    Example:
       x = eye(2,3,'int8');

    See also speye, ones, zeros, rand, randn.

    Overloaded methods:
       distributionReplicated/eye
       distribution1d/eye

    Reference page in Help browser
       doc eye
```

- 零矩阵

A=zeros(n);

B=zeros(n,m);

```
>> A=zeros(3)
A =
     0     0     0
     0     0     0
     0     0     0
>> B=zeros(3,3)
B =
     0     0     0
     0     0     0
     0     0     0
```

```
>> C=zeros(1,3)
C =
    0    0    0
```

● 元素全 1 的矩阵

A=ones(n);

B=ones(n,m);

语法同 zeros

● 稀疏单位矩阵

用表示稀疏数据的数据结构表示的单位阵

```
>> A=speye(4)
A =
  (1,1)        1
  (2,2)        1
  (3,3)        1
  (4,4)        1

>> whos
  Name      Size            Bytes  Class     Attributes
  A         4x4               104  double    sparse
  B         3x3                72  double
  C         1x3                24  double
  I         3x5               120  double
  ans       2x2                56  double    sparse

>> help speye
 SPEYE  Sparse identity matrix.
    SPEYE(M,N) forms an M-by-N sparse matrix with 1's on
    the main diagonal.  SPEYE(N) abbreviates SPEYE(N,N).

    SPEYE(SIZE(A)) is a space-saving SPARSE(EYE(SIZE(A))).

    See also spones, spdiags, spalloc, sprand, sprandn.

    Overloaded methods:
       distributionReplicated/speye
       distribution1d/speye

    Reference page in Help browser
       doc speye

>> A(1,1)
ans =
```

```
     1

>> A(1,2)
ans =
     0

>> B=A
B =
   (1,1)        1
   (2,2)        1
   (3,3)        1
   (4,4)        1
```

(2) 随机矩阵

- 服从均匀分布的随机矩阵
  rand(n);
  rand(n,m)
- 服从标准正态分布的随机矩阵
  randn(n);
  randn(n,m)

1.3 矩阵的四则运算

A+B　　%矩阵相加
C-B　　%矩阵相减
A*B　　%矩阵相乘
A.*B　%矩阵对应元素相乘

1.4 矩阵与标量的运算

s+B=sE+B
s-B=sE-B
B-s=B-sE
s*B=sE*B

1.5 矩阵的函数运算

inv(B)=B^{-1}
A^n=A^n, A.^n=(a_{ij}^n)
exp(A)=(exp(a_{ij}))
log(A)=(log(a_{ij}))
f(A)=(f(a_{ij}))
A'=$A^{/}$

1.6 矩阵的引用

- **用位置标号引用**
  设 A 是 m*n 矩阵
  B=A(i,j)
- **冒号运算符**
  设 A 是 m*n 矩阵

B=A(:,r) %由第 r 列元素组成的矩阵
B=A(s,:) %由第 s 行元素组成的矩阵
B=A(s1:s2,r1:r2) %取 A 的子矩阵
B=A([1 3 5],:) %由 A 的 1,3,5 行组成的矩阵

练习

(1) 生成随机矩阵 A,B

(2) 计算 $A+A^2+A^3+A^4+A^5+A^6$

(3) 计算 sin(A),exp(A)

(4) 计算 sin.(A),exp.(A)

思考

引进矩阵这种数据对象，有何必要？目的何在？

（数学家，工程人员如何想？）

实验二 矩阵的性能指标和范数

2.1 矩阵相似

判断 A~B，只需求方程 AX-XB=0 是否有非奇异解。

```
X=lyap(A,-B,0);
det(X)
```

2.2 行列式

det(A)

2.3 秩

rank(A)

conj(x): x 的共轭。

2.4 二次型

quadprog

2.5 特征值

eig(A)

例：$eig(A-cI)=eig(A)-c$

```
A=rand(5,5);
c=20;
I=eye(5,5);
eig(A+c*I)
eig(A)
ans =
  23.2146 + 0.0000i
  20.3121 + 0.0000i
  19.6940 + 0.0000i
  19.4406 + 0.1624i
  19.4406 - 0.1624i
ans =
   3.2146 + 0.0000i
   0.3121 + 0.0000i
  -0.3060 + 0.0000i
  -0.5594 + 0.1624i
  -0.5594 - 0.1624i
```

2.6 迹

trace(A)

例：写出矩阵乘积 ABCD 的等迹关系式！（最好使用递归）

```
i=0;
A=zeros(24,4);
for i1=1:4
   for i2=1:4
      for i3=1:4
         for i4=1:4
            if i1~=i2 && i1~=i3 && i1~=i4 && i2~=i3 && i2~=i4 && i3~=i4
               i=i+1;
               A(i,1)=i1;A(i,2)=i2;A(i,3)=i3;A(i,4)=i4;
```

```
                    disp(A(i,:));
                end
            end
        end
    end
End


B=zeros(4,5,5);
for ii=1:4
    BB=rand(5,5);
    B(ii,:,:)=BB(:,:);
%     C(:,:)= B(ii,:,:);
%     disp(C);
End

for ii=1:24
    D1(:,:)=B(A(ii,1),:,:);
    D2(:,:)=B(A(ii,2),:,:);
    D3(:,:)=B(A(ii,3),:,:);
    D4(:,:)=B(A(ii,4),:,:);
    D=D1*D2*D3*D4;
    E(ii)=trace(D);
End

for ii=1:24
    for jj=(ii+1):24
        if abs(E(ii)-E(jj))<0.00001
            E(jj)=0;
        end
    end
end
for ii=1:24
    if E(ii)~=0
        F(ii)=E(ii);
    else
        break;
    end
end
disp(F);
```

2.7 初等变换

```
m=5;
I=eye(m,m);
A=randn(m,m);
%某行乘以常数倍---------
c=0.5;%常数
col=3;%行号
I1=I;
I1(col,col)=1*c;

%某两行互换--------
col1=2;%行号 1
col2=4;%行号 2
I2=I;
```

```
I2(col1,col1)=0;
I2(col2,col2)=0;
I2(col1,col2)=1;
I2(col2,col1)=1;

%一行乘以常数倍加另一行----------
c3=10;%常数
col3=1;%目标排行的行号
col4=4;%乘以常数倍的行
I3=I;
I3(col3,col4)=c3;
%---------------------------------
A1=I1*A;
A2=I2*A;
A3=I3*A;
disp(A)
disp(A1)
disp(A2)
disp(A3)


-1.0056   -0.4049   -0.3038    0.3970    0.5684
   0.4340    0.5279   -0.0087   -0.4828   -1.2060
   0.5201   -1.0070    0.5066   -0.2315    0.4331
  -1.0922    1.0890    1.2033    0.6134   -0.0921
  -0.2258    1.7849    0.5220    1.6829   -0.2441

  -1.0056   -0.4049   -0.3038    0.3970    0.5684
   0.4340    0.5279   -0.0087   -0.4828   -1.2060
   0.2601   -0.5035    0.2533   -0.1157    0.2165
  -1.0922    1.0890    1.2033    0.6134   -0.0921
  -0.2258    1.7849    0.5220    1.6829   -0.2441

  -1.0056   -0.4049   -0.3038    0.3970    0.5684
  -1.0922    1.0890    1.2033    0.6134   -0.0921
   0.5201   -1.0070    0.5066   -0.2315    0.4331
   0.4340    0.5279   -0.0087   -0.4828   -1.2060
  -0.2258    1.7849    0.5220    1.6829   -0.2441

 -11.9280   10.4853   11.7288    6.5309   -0.3528
   0.4340    0.5279   -0.0087   -0.4828   -1.2060
   0.5201   -1.0070    0.5066   -0.2315    0.4331
  -1.0922    1.0890    1.2033    0.6134   -0.0921
  -0.2258    1.7849    0.5220    1.6829   -0.2441
```

2.8 向量的内积与范数

(1)内积

dot

表示两个向量的内积

(2)范数

norm(x, 1)，norm(x, 2)，norm(x, inf)，norm(x, -inf)，norm(x, p)

(3)距离（相似性）

```
pdist2
help pdist2
pdist2 - Pairwise distance between two sets of observations

   This MATLAB function returns the distance between each pair of observations in X
   and Y using the metric specified by Distance.

   D = pdist2(X,Y,Distance)
   D = pdist2(X,Y,Distance,DistParameter)
   D = pdist2(___,Name,Value)
   [D,I] = pdist2(___,Name,Value)

   另请参阅 ExhaustiveSearcher, KDTreeSearcher, createns, knnsearch, pdist

   pdist2 的参考页
    名为 pdist2 的其他函数

d1 = pdist2(x,y,'Euclidean');
d2 = pdist2(x,y,'seuclidean');
d3 = pdist2(x,y,'mahalanobis');
d4 = pdist2(x,y,'cityblock');
d5 = pdist2(x,y,'minkowski',p);
d6 = pdist2(x,y,'chebychev');
d7 = pdist2(x,y,'cosine');
d8 = pdist2(x,y,'correlation');
d9 = pdist2(x,y,'hamming');
d10 = pdist2(x,y,'jaccard');
d11 = pdist2(x,y,'spearman');
```

S=dist(X,0)

Distance Metrics

A distance metric is a function that defines a distance between two observations. `pdist2` supports various distance metrics: Euclidean distance, standardized Euclidean distance, Mahalanobis distance, city block distance, Minkowski distance, Chebychev distance, cosine distance, correlation distance, Hamming distance, Jaccard distance, and Spearman distance.

Given an mx-by-n data matrix X, which is treated as mx (1-by-n) row vectors x_1, x_2, ..., x_{mx}, and an my-by-n data matrix Y, which is treated as my (1-by-n) row vectors y_1, y_2, ...,y_{my}, the various distances between the vector x_s and y_t are defined as follows:

- Euclidean distance

  $$d_{st}^2 = (x_s - y_t)(x_s - y_t)'.$$

  The Euclidean distance is a special case of the Minkowski distance, where $p = 2$.

- Standardized Euclidean distance

$$d_{st}^2 = (x_s - y_t)V^{-1}(x_s - y_t)',$$

where V is the n-by-n diagonal matrix whose jth diagonal element is $(S(j))^2$, where S is a vector of scaling factors for each dimension.

- Mahalanobis distance

$$d_{st}^2 = (x_s - y_t)C^{-1}(x_s - y_t)',$$

where C is the covariance matrix.

- City block distance

$$d_{st} = \sum_{j=1}^{n} |x_{sj} - y_{tj}|.$$

The city block distance is a special case of the Minkowski distance, where $p = 1$.

- Minkowski distance

$$d_{st} = \sqrt[p]{\sum_{j=1}^{n} |x_{sj} - y_{tj}|^p}.$$

For the special case of $p = 1$, the Minkowski distance gives the city block distance. For the special case of $p = 2$, the Minkowski distance gives the Euclidean distance. For the special case of $p = \infty$, the Minkowski distance gives the Chebychev distance.

- Chebychev distance

$$d_{st} = \max_j\{|x_{sj} - y_{tj}|\}.$$

The Chebychev distance is a special case of the Minkowski distance, where $p = \infty$.

- Cosine distance

$$d_{st} = \left(1 - \frac{x_s y'_t}{\sqrt{(x_s x'_s)(y_t y'_t)}}\right).$$

- Correlation distance

$$d_{st} = 1 - \frac{(x_s - \bar{x}_s)(y_t - \bar{y}_t)'}{\sqrt{(x_s - \bar{x}_s)(x_s - \bar{x}_s)'}\sqrt{(y_t - \bar{y}_t)(y_t - \bar{y}_t)'}},$$

where

$$\bar{x}_s = \frac{1}{n}\sum_j x_{sj}$$

and

$$\bar{y}_t = \frac{1}{n}\sum_j y_{tj}.$$

- Hamming distance

$$d_{st} = (\#(x_{sj} \neq y_{tj})/n).$$

- Jaccard distance

$$d_{st} = \frac{\#[(x_{sj} \neq y_{tj}) \cap ((x_{sj} \neq 0) \cup (y_{tj} \neq 0))]}{\#[(x_{sj} \neq 0) \cup (y_{tj} \neq 0)]}.$$

- Spearman distance

$$d_{st} = 1 - \frac{(r_s - \bar{r}_s)(r_t - \bar{r}_t)'}{\sqrt{(r_s - \bar{r}_s)(r_s - \bar{r}_s)'}\sqrt{(r_t - \bar{r}_t)(r_t - \bar{r}_t)'}},$$

where

- r_{sj} is the rank of x_{sj} taken over x_{1j}, x_{2j}, ...$x_{mx,j}$, as computed by `tiedrank`.
- r_{tj} is the rank of y_{tj} taken over y_{1j}, y_{2j}, ...$y_{my,j}$, as computed by `tiedrank`.
- r_s and r_t are the coordinate-wise rank vectors of x_s and y_t, i.e., r_s = (r_{s1}, r_{s2}, ... r_{sn}) and r_t = (r_{t1}, r_{t2}, ... r_{tn}).
- $\bar{r}_s = \frac{1}{n}\sum_j r_{sj} = \frac{(n+1)}{2}.$
- $\bar{r}_t = \frac{1}{n}\sum_j r_{tj} = \frac{(n+1)}{2}.$

Distance metric, specified as a character vector, string scalar, or function handle, as described in the following table.

Value	Description
`'euclidean'`	Euclidean distance (default).
`'squaredeuclidean'`	Squared Euclidean distance. (This option is provided for efficiency only. It does not satisfy the triangle inequality.)
`'seuclidean'`	Standardized Euclidean distance. Each coordinate difference between observations is scaled by dividing by the corresponding element of the standard deviation, `S = nanstd(X)`. Use `DistParameter` to specify another value for `S`.
`'mahalanobis'`	Mahalanobis distance using the sample covariance of `X`, `C = nancov(X)`. Use `DistParameter` to specify another value for `C`, where the matrix `C` is symmetric and positive definite.
`'cityblock'`	City block distance.
`'minkowski'`	Minkowski distance. The default exponent is 2. Use `DistParameter` to specify a different exponent `P`, where `P` is a positive scalar value of the exponent.
`'chebychev'`	Chebychev distance (maximum coordinate difference).
`'cosine'`	One minus the cosine of the included angle between points (treated as vectors).
`'correlation'`	One minus the sample correlation between points (treated as sequences of values).

Value	Description
'hamming'	Hamming distance, which is the percentage of coordinates that differ.
'jaccard'	One minus the Jaccard coefficient, which is the percentage of nonzero coordinates that differ.
'spearman'	One minus the sample Spearman's rank correlation between observations (treated as sequences of values).

@distfun	Custom distance function handle. A distance function has the form
	`function D2 = DISTFUN(ZI,ZJ)`
	`% calculation of distance`
	`...`
	where
	`ZI` is a `1`-by-`n` vector containing a single observation.
	`ZJ` is an `m2`-by-`n` matrix containing multiple observations. `distfun` must accept a matrix `ZJ` with an arbitrary number of observations.
	`D2` is an `m2`-by-`1` vector of distances, and `D2(k)` is the distance between observations `ZI` and `ZJ(k,:)`.
	If your data is not sparse, you can generally compute distance more quickly by using a built-in distance instead of a function handle.

```
A=rand(4,5);
B=rand(4,5);
ab=dot(A,B);
X(:)=A(:,1);
Y(:)=B(:,1);
xy=dot(X,Y);
disp(ab);
disp(xy);
disp(['X=',num2str(X)]);
normX=norm(X);
normX1=norm(X,1);
normX2=norm(X,2);
normXinf=norm(X,Inf);
normXfro=norm(X,'fro');
disp(normX);
disp(normX1);
disp(normX2);
disp(normXinf);
disp(normXfro);
```

2.9 矩阵范数

列和范数： norm(A, 1)

行和范数：norm(A, inf)
谱范数：norm(A, 2)
Frobenius 范数 ： norm(A, ‘fro’)

Syntax

```
n = norm(v)
n = norm(v,p)
n = norm(X)
n = norm(X,p)
n = norm(X,'fro')
```

Description

`n = norm(v)` returns the Euclidean norm of vector v. This norm is also called the 2-norm, vector magnitude, or Euclidean length.

`n = norm(v,p)` returns the generalized vector p-norm.

`n = norm(X)` returns the 2-norm or maximum singular value of matrix X, which is approximately `max(svd(X))`.

`n = norm(X,p)` returns the *p*-norm of matrix X, where p is 1, 2, or Inf:

If p = 1, then n is the maximum absolute column sum of the matrix.
If p = 2, then n is approximately `max(svd(X))`. This is equivalent to `norm(X)`.
If p = Inf, then n is the maximum absolute row sum of the matrix.

`n = norm(X,'fro')` returns the Frobenius norm of matrix X.

```
A=rand(4,5);
B=rand(4,5);
ab=dot(A,B);
disp(ab);
%disp(['A=',num2str(A)]);
normA=norm(A);
normA1=norm(A,1);
normA2=norm(A,2);
normAinf=norm(A,Inf);
normAfro=norm(A,'fro');
disp(normA);
disp(normA1);
disp(normA2);
disp(normAinf);
disp(normAfro);
```

练习

(1)生成一个矩阵 A,求其行列式、迹、秩等。

(2)研究 quadprog，生成一个二次型问题，并对其求解。

(3)编写程序，实现矩阵的初等变换，三种类型都考虑。

思考

判断两个向量的相似性，用哪种距离函数最好？

实验三 特殊矩阵

3.1 三角阵和上三角阵的生成

diag, triu, chol

3.2 基本矩阵和向量的外积

$$a \wedge b = ab^T$$

3.3 酉矩阵和正交矩阵

傅里叶矩阵、Hadamard 矩阵是正交矩阵

3.4 矩阵的向量化

```
A =1     2     3
     4     5     6
>> B=reshape(A,1,[])
B =
   1     4     2     5     3     6
>> C=reshape(B,2,3)
C =
     1     2     3
     4     5     6
```

3.5 位移矩阵

```
A=eye(3);
B=circshift(A,-1);
C=B*[ 10,11,12]'
```

3.6 傅里叶矩阵和范德蒙矩阵

```
clear all
N=4;
w=exp(-1i*2*pi/N);
V=zeros(1,N);
for ii=1:N
   V(ii)=w^(ii-1);
end
A=vander(V);
B=fliplr(A);
F=B;
for ii=1:N
   for jj=1:N
      k=((ii-1)*(jj-1));
      FF(ii,jj)=w^k;
   end
```

```
end
F-FF
C=real(F);
S=imag(F);
C*C+S*S
F*F
```

3.7 各种对称矩阵

对称矩阵、反对称矩阵、斜对称矩阵、中心对称矩阵

3.8 拓普利兹矩阵(Toeplitz matrix)和汉克矩阵(Hankel matrix)

```
>> help hankel
hankel - Hankel 矩阵
    此 MATLAB 函数 返回其第一列是 c 并且其第一个反对角线下方的元素为零的
Hankel 方阵.
    H = hankel(c)
    H = hankel(c,r)
>> help  toeplitz
toeplitz - Toeplitz matrix
    This MATLAB function returns a nonsymmetric Toeplitz matrix with c as its
first column and r as its first row.
    T = toeplitz(c,r)
    T = toeplitz(r)
```

3.9 最小二乘拟合

```
clear all
close all
N=7;%阶数，N+1 个系数
M=20;%数据采样点个数
x=rand(M,1);
a=zeros(1,2*N+1);y=sin(pi/2*x);%真实信号
K=0.02;%干扰信号的放大系数
errors=K*randn(M,1);%干扰信号
y=y+errors;%采样信号（叠加了噪声, 加性噪声）
%y=y.*(1+errors)；%采样信号（叠加了噪声, 一种乘性噪声）
b=zeros(N+1,1);
for k=1:2*N+1
   for ii=1:M
      a(k)=a(k)+x(ii)^(k-1);
   end
end
for k=1:N+1
```

```
    for ii=1:M
        b(k)=b(k)+y(ii)*x(ii)^(k-1);
    end
end
H=zeros(N+1,N+1);
for ii=1:N+1
    for jj=1:N+1
        H(ii,jj)=a(ii+jj-1);
    end
end
disp(H); A=H\b; disp(A');  tt=0:0.01:1;
yy=zeros(1,length(tt));
for ii=1:length(tt)
    for jj=1:N+1
         yy(ii)=yy(ii)+A(jj)*tt(ii)^(jj-1);
    end
end
plot(tt,0.05+sin(tt*pi/2),'--','Color','b','LineWidth',1);
hold on
plot(tt,yy,'Color','r');
legend('真实信号','多项式拟合','Location','SouthEast' );
```

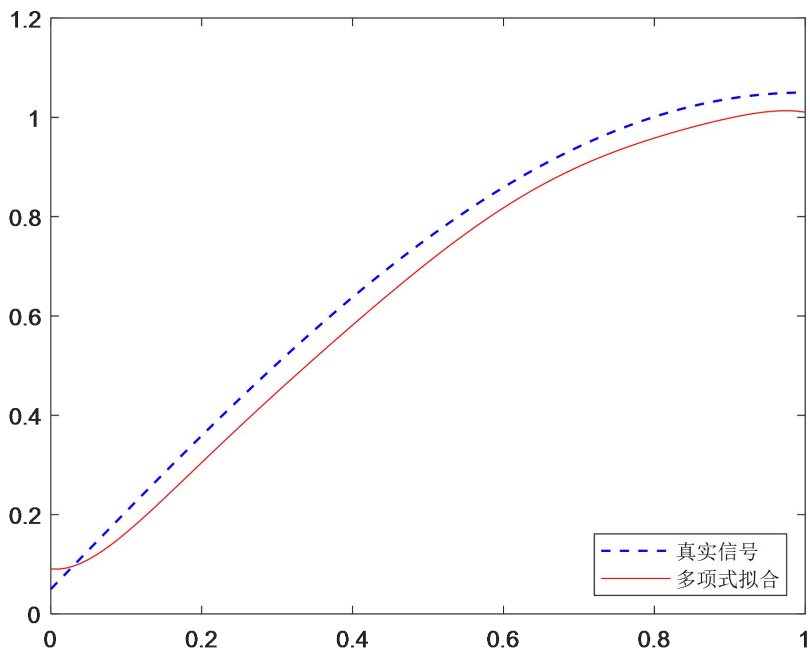


图 3.1 曲线拟合

3.10 Kronecker 积

验证 $K_{mn} vec(A) = vec(A^T)$，其中 $K_{mn} = \sum_{j=1}^{n} (e_j^T \otimes I_m \otimes e_j)$

```
function main()
clear all
n=3;
m=3;
K=comm(n,m);
disp(K)
A=rand(3,3);
B=vec(A);
C=vec(A');
D=K*B-C;
disp(D');
end
 %子程序:
function V=vec(A) %向量化
  V=reshape(A,[],1);
end
function V=unvec(A,m,n)%反向量化
  V=reshape(A,m,n);
end
function K=comm(n,m)%生成交换矩阵
   I1=eye(n);
   I2=eye(m);
   K=zeros(m*n);
   for ii=1:n
      temp1(:)=I1(:,ii);
      e_ii=temp1';%列向量
      temp2=kron(e_ii',I2);
      K=K+kron(temp2,e_ii);
   end
end
```

```
>> main
   1   0   0   0   0   0   0   0   0
   0   0   0   1   0   0   0   0   0
   0   0   0   0   0   0   1   0   0
   0   1   0   0   0   0   0   0   0
   0   0   0   0   1   0   0   0   0
   0   0   0   0   0   0   0   1   0
   0   0   1   0   0   0   0   0   0
```

0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0

练习

(1)验证酉矩阵的性质。

(2)研究最小二乘法曲线拟合问题。

(3)验证矩阵各种新运算是否满足：交换律、结合律、分配律。

思考

（1）如何随机生成正交矩阵？

（2）验证矩阵各种新运算是否满足：交换律、结合律、分配律。

（3）如何用增量采样数据，改进曲线拟合？

实验四 矩阵分解

4.1 常用矩阵分解

矩阵分解是矩阵分析的出发点。注：蓝色标注的矩阵分解是针对方阵的。

（1）任意矩阵都与其等价标准型等价

$$A_{m\times n}=P_{m\times m}\begin{pmatrix} I_{r\times r} & O_{r\times(n-r)} \\ O_{(m-r)\times r} & O_{(m-r)\times(n-r)} \end{pmatrix}Q_{n\times n}, r=rank(A), Q,P\text{是非奇异的.}$$

（2）三角(LU)分解

```
[L,U,P]=lu(A);
```

（3）Cholesky 分解(乔里斯基)

```
R=chol(X);
```

（4）C.Schur 分解(舒尔)

```
[U,R]=schur(X，'complex');
[U,R]=schur(X,'real');
```

（5）QR 分解

```
[Q,R]=qr(X);
[Q,R,E]=qr(X);
```

（6）若当标准型

$$U^{-1}AU=J$$

$$J=\begin{pmatrix} J_1 & & O \\ & \ddots & \\ O & & J_p \end{pmatrix}, J_k=\begin{pmatrix} \lambda & 1 & & 0 \\ & \lambda & 1 & \\ & & \ddots & 1 \\ 0 & & & \lambda \end{pmatrix}\in C^{L_k\times L_k}$$

Jordan 标准型理论价值比较大。很多性质，都可基于 Jordan 标准型理论推导。

（7）特征值分解

```
[V,D]=eig(X);
```

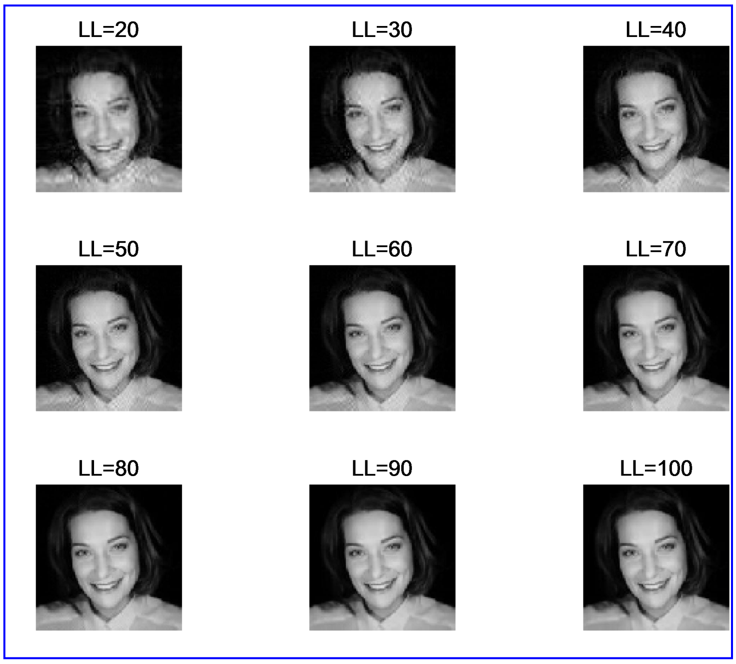
（8）奇异值分解

[U, S, V]=svd(X)；

4.2 案例 1：图像压缩

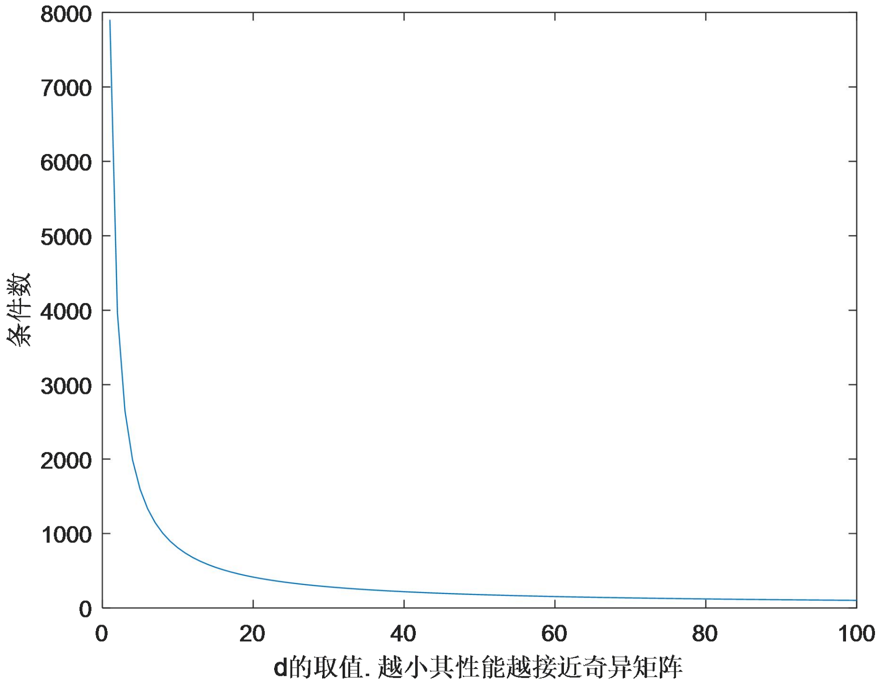
```
%读原图像
X=imread('test.jpg');
Y=rgb2gray(X);
figure(1);imshow(Y);

%将原图像整理为正方图像，宽高为 H
H=100;
S=size(Y);%得图像的宽和高
Z=zeros(H,H);

for ii=1:H
    for jj=1:H
        Z(ii,jj)=Y(fix((S(1)-1)/H*ii)+1,fix((S(2)-1)/H*jj)+1);
    end
end
figure(2); imshow(uint8(Z)); title('原图');

%特征分解
[U,D]=eig(double(Z));
M=zeros(H,H);

for LL=1:H/10-1;
MM=M;
for ii=1:(LL+1)*10
    MM(ii,ii)=1;
end

%基于特征分解进行图像重建（还原）
GG=U*MM*D/U;
figure(3),subplot(3,3,LL),
imshow(uint8(abs(GG))); title(['LL=',num2str(LL*10+10)]);
end

ZZ=zeros(10,10);
flag=1;
for ii=1:10
    for jj=1:10
        ZZ(ii,jj)=abs(D(flag,flag));
```

```
        flag=flag+1;
    end
end
disp(ZZ);
```



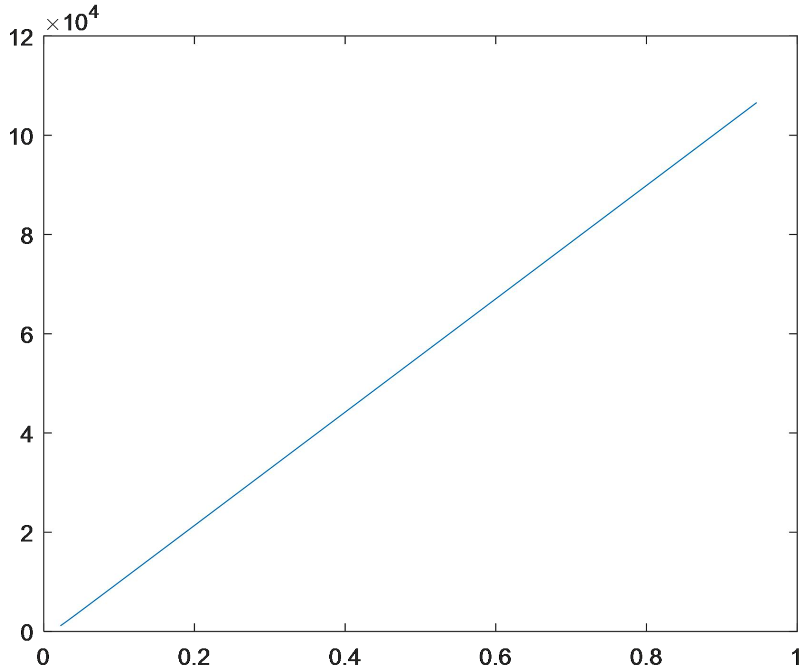
4.3 条件数的计算

```
%设置一个 小参数
d=0.01;
%生成两个满秩矩阵
U=rand(4,4);
V=rand(4,4);
%基于 d 生成一个矩阵序列
for I=1:100
    dd=d*I;
    D=diag([4,3,2,dd]);
    A=U*D*V;
    %计算矩阵条件数
    Y(I)=cond(A);
end
%画 Y 的函数图象
plot(Y);
xlabel('d 的取值. 越小其性能越接近奇异矩阵');
ylabel('条件数');
```



4.4 条件数与方程解的精度之间的关系

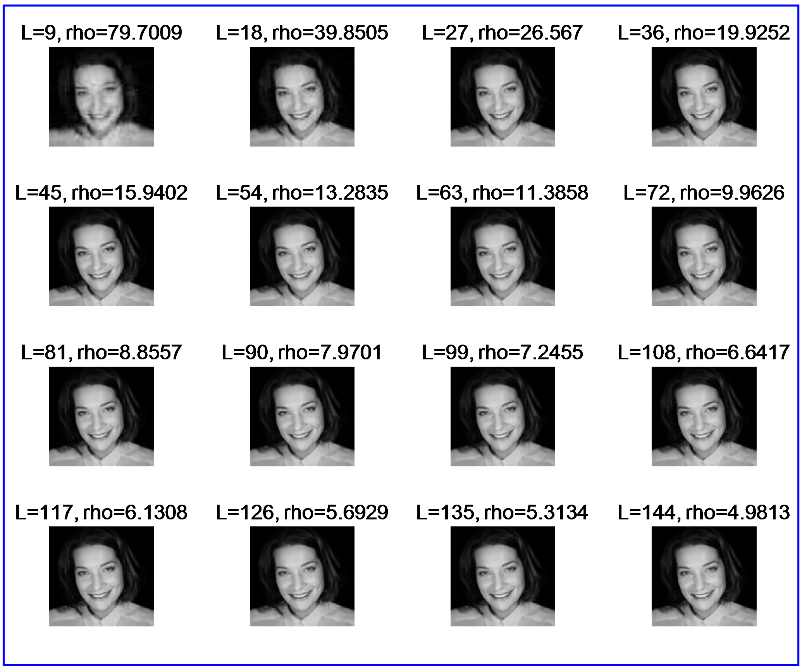
```
clear all
dd=0.01;
b=rand(4,1);
dA=0.01*rand(4,4);
U=rand(4,4);
V=rand(4,4);
for I=1:100
    d=dd*I;
    D=diag([4,3,2,d]);
    A=U*D*V;
    Y(I)=cond(A);
    X=A\b;
    XX=(A+dA)\b;
    a1=norm(XX);
    a2=norm(XX-X);
    YY(I)=a2/a1;
end
whos
plot(YY,Y);
```



4.5 奇异值分解（图像压缩）

```
%读原图像
X=imread('test.jpg');
Y=rgb2gray(X);
figure(1);imshow(Y);

%奇异值分解
[U,D,V]=svd(double(Y));

SS=size(D);

M=zeros(SS(1),SS(2));
H=min(SS(1),SS(2));
dd=fix(H/9);
d=fix(sqrt(dd));

for L=1:d*d
MM=M;
for ii=1:(L)*9
   MM(ii,ii)=1;
end
rho=SS(1)*SS(2)/((SS(1)+SS(2)+1)*L);
%基于特征值分解进行图像重建（还原）
GG=U*(MM.*D)*V';
figure(3),subplot(d,d,L),
```

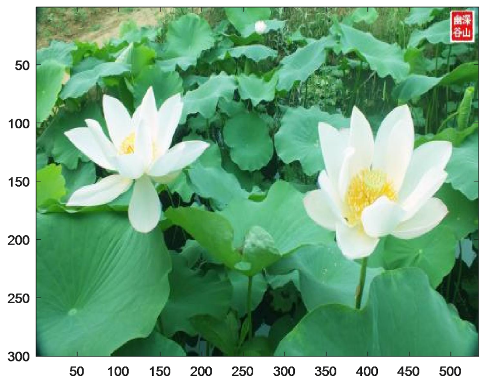
```
imshow(uint8(abs(GG)));
title(['L=',num2str(L*9),', rho=',num2str(rho)]);
end

ZZ=zeros(d,d);
flag=1;
for ii=1:d
    for jj=1:d
        ZZ(ii,jj)=abs(D(flag,flag));
        flag=flag+1;
    end
end
disp(ZZ);
```

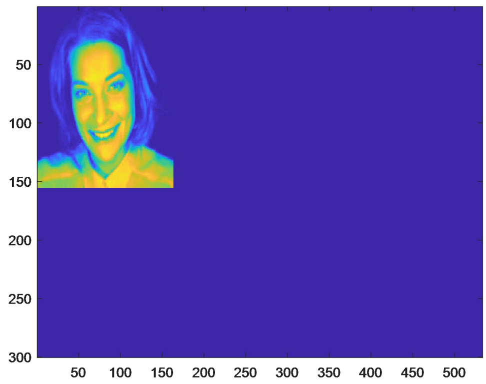


4.6 数字水印

```
AA=imread('A.jpg'); %原图
WW=imread('test.jpg');%水印
A(:,:)=AA(:,:,1);
W=rgb2gray(WW);
SW=size(W);
figure(1),imagesc(AA);
[U,S,V]=svd(double(A));
SS=size(S);
WWW=zeros(SS(1),SS(2));
WWW(1:SW(1),1:SW(2))=W(:,:);
figure(2),imagesc(WWW);
a=0.1;
```

```
%嵌入数字水印
L=a*double(WWW)+S;
[U1,S1,V1]=svd(double(L));
AW=U*S1*V';
AA(:,:,1)=AW(:,:);
figure(3),imagesc(AA);
%提取数字水印
[Up,Sp,Vp]=svd(double(AW));
F=U1*Sp*V1';
WE=(F-S)/a;
figure(4),imagesc(WE);
```





4.7 主成分分析

```
load hald;
%用 Matlab 函数 pca( )进行主成分分析
[coeff,score,latent,tsquared,explained] =
pca(ingredients,'Algorithm','svd','Centered',true)
disp('--用 Matlab 函数 pca( )进行主成分分析---结束');
%自编程序验证 pca()
S=size(ingredients);%求数据维数
EX=zeros(1,S(2)); %用于存放数据期望值
%（1）求期望值
for II=1:S(1)
   EX=EX+ingredients(II,:);
end
EX=EX/S(1);
%（2）求协方差矩阵
DX=zeros(S(2),S(2));
for II=1:S(1)
   DX=DX+(ingredients(II,:)-EX)'*(ingredients(II,:)-EX);
end
DX=DX/(S(1)-1);
disp(EX);
```

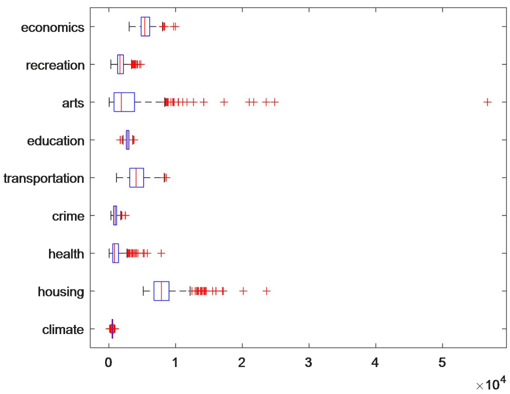
```
disp('以上是数学期望---------------');
disp(DX);
disp('以上是协方差矩阵-------------');
%（3）特征值分解或奇异值分解
[U,SS,V]=svd(DX);
%[U,SS]=eig(DX)
Y=U'*(ingredients'-EX'*ones(1,S(1)));%Hotelling 变换
%coeff
disp('Coeff=');
disp(U);
%latent
disp('Latent=');
disp(diag(SS));
%score
disp('Score=');
disp(Y');
Explained=diag(SS);
Explained=100*Explained/sum(Explained);
%explained
disp('Explained=');
disp(Explained);
S1=DX;
n=S(2);
for II=1:S(1)
   X(II,:)=ingredients(II,:)-EX;
end
Tsquared=X*pinv(S1)*X';%Matlab 如此计算 Tsquared!!!
T2=diag(Tsquared);
%tsquared
disp('Tsquared=');
disp(T2);
disp('---自编程序验证 pca()结束----');
```

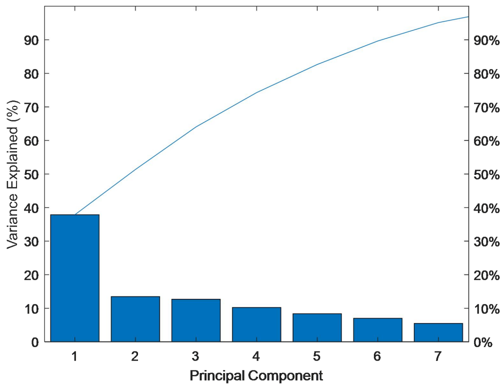
4.8 城市生活指数排名

（此实例来自 Matlab 帮助）

```
load cities
figure()
boxplot(ratings,'Orientation','horizontal','Labels',categories)
[wcoeff,score,latent,tsquared,explained] = pca(ratings,...
'VariableWeights','variance');
```

```
figure()
pareto(explained)
xlabel('Principal Component')
ylabel('Variance Explained (%)')
[st2,index] = sort(tsquared,'descend');
% sort in descending order
extreme = index(1);
names(extreme,:)
for II=1:329
    extreme = index(II);
    disp(names(extreme,:));
end
```





练习

（1）随机生成正交阵、酉矩阵、对称阵

（2）讨论条件数对矩阵分解精度的影响

思考

（1）使用 pca 函数进行主成分分析，算法可选择 svd 或 eig。算法选择是否影响数据分析结论？

实验五 逆矩阵与矩阵方程求解

5.1 彭罗斯逆的奇异值分解(SVD)法

$$A^{+} = V\Sigma^{+}U^{H}$$

```
A=rand(3,4);
[U,S,V]=svd(A)
SS=size(S);
SSS=min(SS(1),SS(2));
S1=S;
for II=1:SSS
   if S(II,II)>eps
      S1(II,II)=1/S(II,II);
   end
end
PA=(U*S1*V')'
PA1=pinv(A)

U =
  -0.7350  -0.0625   0.6751
  -0.4743  -0.6643  -0.5778
  -0.4846   0.7449  -0.4586
S =
   1.9188        0        0        0
        0   0.7543        0        0
        0        0   0.2598        0
V =
  -0.6132   0.4201   0.4111  -0.5277
  -0.5262   0.1633   0.1102   0.8273
  -0.1764   0.4704  -0.8599  -0.0905
  -0.5621  -0.7587  -0.2818  -0.1702
PA =
   1.2685  -1.1327  -0.1561
   0.4745  -0.2589   0.0995
  -2.2061   1.5419   2.0271
  -0.4541   1.4337  -0.1099
PA1 =
   1.2685  -1.1327  -0.1561
   0.4745  -0.2589   0.0995
  -2.2061   1.5419   2.0271
```

-0.4541 1.4337 -0.1099

5.2 方程的普通最小二乘解

```
clear all
MU=1.5;
SIGMA=1;
A= normrnd(MU,SIGMA,3,4)
b= normrnd(MU,SIGMA,3,1)
XLS1=(A'*A)\(A'*b)
XLS2=pinv(A'*A)*(A'*b)
```

5.3 吉洪诺夫正则化方法

```
clear all
MU=1.5;
SIGMA=1;
A= normrnd(MU,SIGMA,4,3)
b= normrnd(MU,SIGMA,4,1)
l=1
T=(A'*A+l.*eye(3,3))\A'*b
l=2
T=(A'*A+l.*eye(3,3))\A'*b
l=10
T=(A'*A+l.*eye(3,3))\A'*b
l=100
T=(A'*A+l.*eye(3,3))\A'*b
```

5.4 总体最小二乘法

若增广矩阵$B=[A,b]$的奇异值为$\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_{n+1}$,则总体最小二乘解可表示为

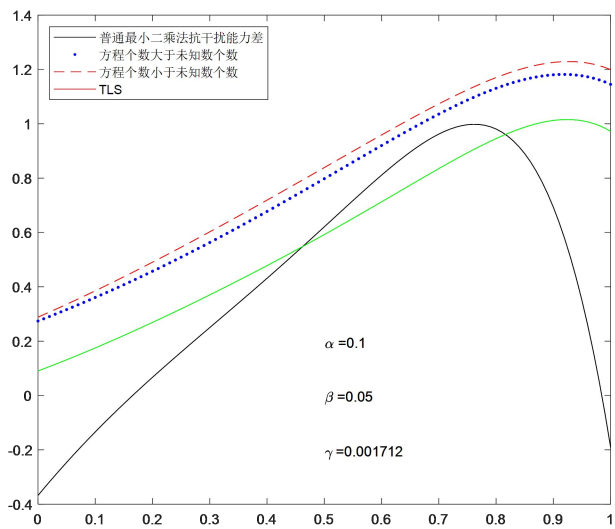
$$X_{TLS}=(A^H A-\sigma_{n+1}^2 I)^{-1} A^H b.$$

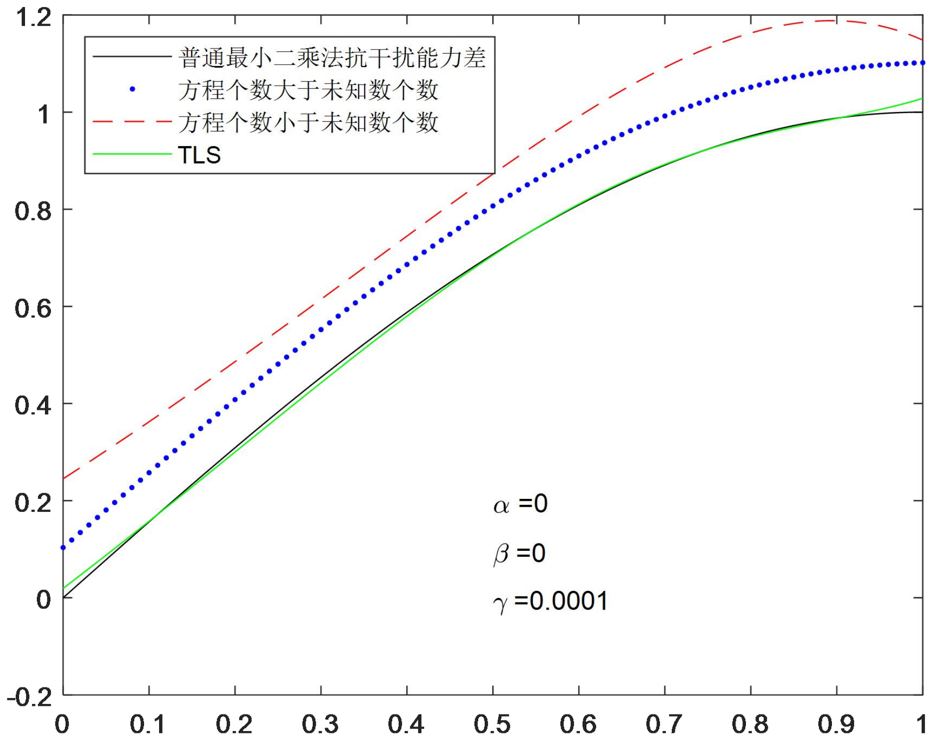
若增广矩阵具有比较好的性质，则总体最小二乘解等价于特殊参数的吉洪诺夫反正则化方法(deregularizd method)解.

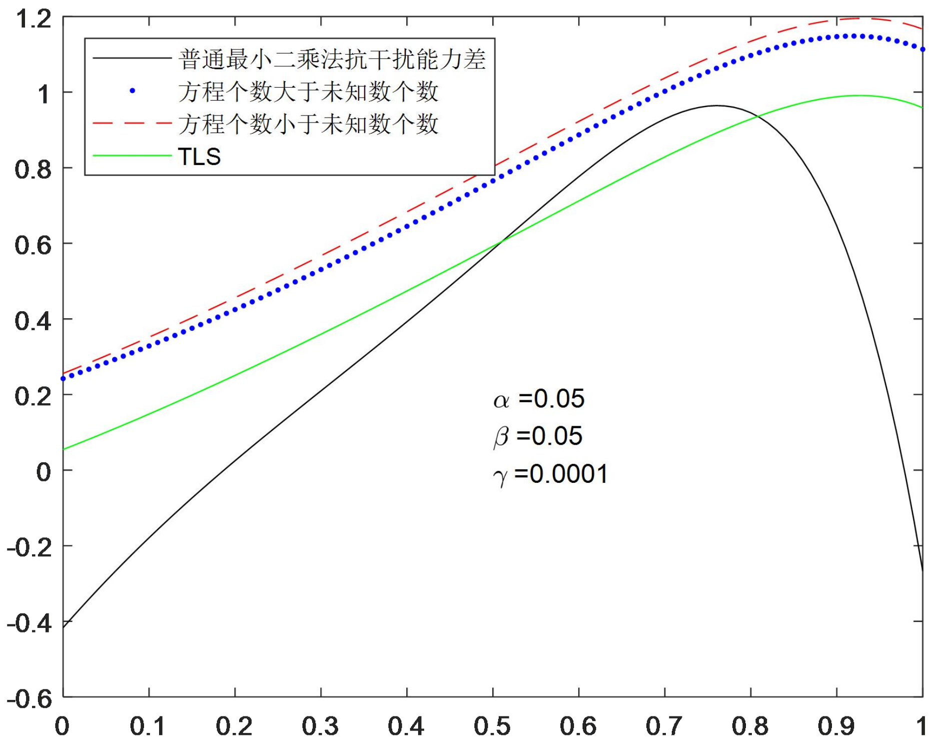
```
clear all; rng default
N=7;%阶数，N+1 个系数
M=20;%数据采样点个数
x=rand(M,1);
alpha=0.1;
beta=0.05;
b=sin(pi/2*x)+randn*alpha;%取值有噪声!
a=zeros(N+1,1);
A=zeros(M,N+1);
```

```
for ii=1:M
    for jj=1:N+1
        A(ii,jj)=x(ii)^(jj-1)+randn*beta;%有噪声！
    end
end
BltN=A(1:N-1,:);
BeqN=A(1:N+1,:);
BgtN=A;
%%
%(1):方程个数等于未知数个数
B=BeqN;
bb=b(1:N+1); a=B\bb;
%a=(B'*B+.00012*eye(1+N,1+N))\B'*bb;
tt=0:0.01:1;
yy=zeros(1,length(tt));
for ii=1:length(tt)
    for jj=1:N+1
          yy(ii)=yy(ii)+a(jj)*tt(ii)^(jj-1);
    end
end
plot(tt,yy,'-k');
hold on
%%
%(2)方程个数大于未知数个数
B=BgtN;
bb=b;
a=(B'*B+.00012*eye(1+N,1+N))\B'*bb;
yy=zeros(1,length(tt));
for ii=1:length(tt)
    for jj=1:N+1
          yy(ii)=yy(ii)+a(jj)*tt(ii)^(jj-1);
    end
end
plot(tt,yy+0.1,'.b');
hold on
%%
%(3)方程个数小于未知数个数
B=BltN;
bb=b(1:N-1);
a=(B'*B+.0012*eye(1+N,1+N))\B'*bb;
```

```
yy=zeros(1,length(tt));
for ii=1:length(tt)
   for jj=1:N+1
        yy(ii)=yy(ii)+a(jj)*tt(ii)^(jj-1);
   end
end
plot(tt,yy+0.2,'--r');
%%
%(4)方程个数小于未知数个数
B=BltN;
bb=b(1:N-1);
%这是总体最小二乘法的简单处理方案，调整参数 gama
gamma=.001712;
a=(B'*B-gamma*eye(1+N,1+N))\B'*bb;
yy=zeros(1,length(tt));
for ii=1:length(tt)
   for jj=1:N+1
        yy(ii)=yy(ii)+a(jj)*tt(ii)^(jj-1);
   end
end
plot(tt,yy,'-g');
legend('普通最小二乘法抗干扰能力差','方程个数大于未知数个数','方程个数小于未知数个数','TLS','Location','NorthWest');
text(0.5,0.2,['\alpha =' ,num2str(alpha)]);
text(0.5,0,['\beta =' ,num2str(beta)]);
text(0.5,-0.2,['\gamma =' ,num2str(gamma)]);
```







5.5 稀疏矩阵方程求解

X=normrnd(0,1,3,5)

```
Y=rand(3,1)
B=lasso(X,Y)
```

```
>> B(:,3)
ans =
         0
         0
    0.1941
    1.3347
         0
>> B(:,39)
ans =
     0
     0
     0
     0
     0
>> B(:,33)
ans =
         0
         0
    0.1079
    0.3278
         0
>>
```

实验六 矩阵微积分

6.1 用 MATLAB 函数 logm()计算 log(tA)的导数

```
rng default;
A=rand(3,3);
disp(A);
t=3;
dt=0.001;
dA=logm((t+dt)*A)-logm(t*A);
B=dA/dt;
disp(B);
  0.8147    0.9134    0.2785
    0.9058    0.6324    0.5469
    0.1270    0.0975    0.9575
   0.3333 + 0.0000i   0.0000 - 0.0000i  -0.0000 + 0.0000i
   0.0000 - 0.0000i   0.3333 + 0.0000i   0.0000 - 0.0000i
   0.0000 + 0.0000i   0.0000 - 0.0000i   0.3333 + 0.0000i
```

6.2 函数 f(X)=aTXb 的导函数

```
a=rand(1,5)';
b=rand(1,5)';
X=rand(5,5);
delta=0.01;
Z=zeros(5,5);
for ii=1:5
    for jj=1:5
        XX=X;       XX(ii,jj)=XX(ii,jj)+delta;
        f=a'*XX*b-a'*X*b;
        Z(ii,jj)=f/delta;
    end
end
disp(Z);
disp(a*b');
```

6.3 Tikhonov 方法的最速下降法实现

```
A=rand(4,4);
b=rand(4,1);
X=zeros(4,1);
lamda=.2;
Xt=(A'*A+lamda*eye(4,4))\A'*b;
disp([0,Xt']);

lamda1=0.15;
```

```
Df=zeros(4,4);
jj=10;kk=1;
for ii=1:200
    df=2*(A*X-b)'*A+2*lamda*X';
    Df=-df';
    X=X+lamda1*Df;
    if jj==ii
        disp([ii,X']);
        kk=kk+1;
         jj=kk*40;
    end
end
disp([ii,X']);
```

lamda1=0.1 (迭代两百次，与理论值就基本相同了）

```
         0    0.1264   -0.0274    0.0519    0.0388
   10.0000    0.0909    0.0032    0.0519    0.0436
   80.0000    0.1262   -0.0273    0.0523    0.0385
  120.0000    0.1264   -0.0274    0.0520    0.0387
  160.0000    0.1264   -0.0274    0.0519    0.0387
  200.0000    0.1264   -0.0274    0.0519    0.0388
  200.0000    0.1264   -0.0274    0.0519    0.0388
```

6.4 Tikhonov 方法的牛顿法实现

```
A=rand(4,4);
b=rand(4,1);
X=zeros(4,1);
lamda=.2;
%Tikhonov 解
Xt=(A'*A+lamda*eye(4,4))\A'*b;
disp([0,Xt']);
lamda1=1.2;%学习率
Df=zeros(4,4);
jj=5;kk=1;
for ii=1:200
    df=2*(A*X-b)'*A+2*lamda*X';%协梯度
    Df=2*(A'*A+lamda*eye(4,4));%H 矩阵
    X=X-lamda1*Df\df';
    if jj==ii
        disp([ii,X']);
        kk=kk+1;
         jj=kk*5;
    end
```

```
end
disp([ii,X']);
 lamda=.2，lamda1=1.2
 0   -0.0838    0.2688    0.0570    0.1199
 5   -0.0838    0.2687    0.0570    0.1198
10   -0.0838    0.2688    0.0570    0.1199
15   -0.0838    0.2688    0.0570    0.1199
20   -0.0838    0.2688    0.0570    0.1199
25   -0.0838    0.2688    0.0570    0.1199
```

5 步收敛到一个比较精确的值！