矩阵计算 R 笔记

安宁宁

暨南大学经济学院统计学系

nning_an@sina.com

1　创建一个向量

在 R 中可以用函数 c()来创建一个向量，例如：

```
> x=c(1,2,3,4)
> x
[1] 1 2 3 4
```

2　创建一个矩阵

在 R 中可以用函数 matrix()来创建一个矩阵，应用该函数时需要输入必要的参数值。

```
> args(matrix)
function (data = NA, nrow = 1, ncol = 1, byrow = FALSE, dimnames
= NULL)
```

data 项为必要的矩阵元素，nrow 为行数，ncol 为列数，注意 nrow 与 ncol 的乘积应为矩阵元素个数，byrow 项控制排列元素时是否按行进行，dimnames 给定行和列的名称。例如：

```
> matrix(1:12,nrow=3,ncol=4)
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
> matrix(1:12,nrow=4,ncol=3)
     [,1] [,2] [,3]
[1,]    1    5    9
[2,]    2    6   10
[3,]    3    7   11
[4,]    4    8   12
> matrix(1:12,nrow=4,ncol=3,byrow=T)
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
[4,]   10   11   12
> rowname
[1] "r1" "r2" "r3"
> colname=c("c1","c2","c3","c4")
> colname
[1] "c1" "c2" "c3" "c4"
> matrix(1:12,nrow=3,ncol=4,dimnames=list(rowname,colname))
   c1 c2 c3 c4
```

```
r1  1  4  7 10
r2  2  5  8 11
```

3　矩阵转置

A 为 $m \times n$ 矩阵，求 **A'**在 R 中可用函数 `t()`，例如：

```
> A=matrix(1:12,nrow=3,ncol=4)
> A
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
> t(A)
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
[4,]   10   11   12
```

若将函数 `t()`作用于一个向量 `x`，则 R 默认 `x` 为列向量，返回结果为一个行向量，例如：

```
> x
 [1]  1  2  3  4  5  6  7  8  9 10
> t(x)
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
[1,]    1    2    3    4    5    6    7    8    9    10
> class(x)
[1] "integer"
> class(t(x))
[1] "matrix"
```

若想得到一个列向量，可用 `t(t(x))`，例如：

```
> x
 [1]  1  2  3  4  5  6  7  8  9 10
> t(t(x))
      [,1]
 [1,]    1
 [2,]    2
 [3,]    3
 [4,]    4
 [5,]    5
 [6,]    6
 [7,]    7
 [8,]    8
 [9,]    9
[10,]   10
> y=t(t(x))
> t(t(y))
      [,1]
```

```
 [1,]    1
 [2,]    2
 [3,]    3
 [4,]    4
 [5,]    5
 [6,]    6
 [7,]    7
 [8,]    8
 [9,]    9
[10,]   10
```

4　矩阵相加减

在 R 中对同行同列矩阵相加减，可用符号："＋"、"－"，例如：

```
> A=B=matrix(1:12,nrow=3,ncol=4)
> A+B
     [,1] [,2] [,3] [,4]
[1,]    2    8   14   20
[2,]    4   10   16   22
[3,]    6   12   18   24
> A-B
     [,1] [,2] [,3] [,4]
[1,]    0    0    0    0
[2,]    0    0    0    0
[3,]    0    0    0    0
```

5　数与矩阵相乘

A 为 $m\times n$ 矩阵，$c>0$，在 R 中求 c**A** 可用符号："*"，例如：

```
> c=2
> c*A
     [,1] [,2] [,3] [,4]
[1,]    2    8   14   20
[2,]    4   10   16   22
[3,]    6   12   18   24
```

6　矩阵相乘

A 为 $m\times n$ 矩阵，**B** 为 $n\times k$ 矩阵，在 R 中求 **AB** 可用符号："%*%"，例如：

```
> A=matrix(1:12,nrow=3,ncol=4)
> B=matrix(1:12,nrow=4,ncol=3)
> A%*%B
     [,1] [,2] [,3]
[1,]   70  158  246
[2,]   80  184  288
[3,]   90  210  330
```

若 **A** 为 $n\times m$ 矩阵，要得到 **A'B**，可用函数 crossprod()，该函数计算结果与 t(A)%*%B 相同，但是效率更高。例如：

```
> A=matrix(1:12,nrow=4,ncol=3)
> B=matrix(1:12,nrow=4,ncol=3)
```

```
> t(A)%*%B
     [,1] [,2] [,3]
[1,]   30   70  110
[2,]   70  174  278
[3,]  110  278  446
> crossprod(A,B)
     [,1] [,2] [,3]
[1,]   30   70  110
[2,]   70  174  278
[3,]  110  278  446
```

7　矩阵对角元素相关运算

例如要取一个方阵的对角元素，

```
> A=matrix(1:16,nrow=4,ncol=4)
> A
     [,1] [,2] [,3] [,4]
[1,]    1    5    9   13
[2,]    2    6   10   14
[3,]    3    7   11   15
[4,]    4    8   12   16
> diag(A)
[1]  1  6 11 16
```

对一个向量应用 diag()函数将产生以这个向量为对角元素的对角矩阵，例如：

```
> diag(diag(A))
     [,1] [,2] [,3] [,4]
[1,]    1    0    0    0
[2,]    0    6    0    0
[3,]    0    0   11    0
[4,]    0    0    0   16
```

对一个正整数 z 应用 diag()函数将产生以 z 维单位矩阵，例如：

```
> diag(3)
     [,1] [,2] [,3]
[1,]    1    0    0
[2,]    0    1    0
[3,]    0    0    1
```

8　矩阵求逆

矩阵求逆可用函数 solve()，应用 solve(a, b)运算结果是解线性方程组 $\mathbf{ax} = \mathbf{b}$，若 $\mathbf{b}$ 缺省，则系统默认为单位矩阵，因此可用其进行矩阵求逆，例如：

```
> a=matrix(rnorm(16),4,4)
> a
           [,1]       [,2]       [,3]       [,4]
[1,] 1.6986019  0.5239738  0.2332094  0.3174184
[2,] -0.2010667  1.0913013 -1.2093734  0.8096514
[3,] -0.1797628 -0.7573283  0.2864535  1.3679963
[4,] -0.2217916 -0.3754700  0.1696771 -1.2424030
```

```
> solve(a)
            [,1]        [,2]       [,3]       [,4]
[1,]  0.9096360  0.54057479  0.7234861  1.3813059
[2,] -0.6464172 -0.91849017 -1.7546836 -2.6957775
[3,] -0.7841661 -1.78780083 -1.5795262 -3.1046207
[4,] -0.0741260 -0.06308603  0.1854137 -0.6607851
> solve (a) %*%a
             [,1]          [,2]          [,3]          [,4]
[1,] 1.000000e+00  2.748453e-17 -2.787755e-17 -8.023096e-17
[2,] 1.626303e-19  1.000000e+00 -4.960225e-18  6.977925e-16
[3,] 2.135878e-17 -4.629543e-17  1.000000e+00  6.201636e-17
[4,] 1.866183e-17  1.563962e-17  1.183813e-17  1.000000e+00
```

9　矩阵的特征值与特征向量

矩阵 **A** 的谱分解为 **A=UΛU'**,其中 **Λ** 是由 **A** 的特征值组成的对角矩阵，**U** 的列为 **A** 的特征值对应的特征向量，在 R 中可以用函数 eigen()函数得到 **U** 和 **Λ**，

```
> args(eigen)
function (x, symmetric, only.values = FALSE, EISPACK = FALSE)
```

其中：x 为矩阵，symmetric 项指定矩阵 x 是否为对称矩阵，若不指定，系统将自动检测 x 是否为对称矩阵。例如：

```
> A=diag(4)+1
> A
     [,1] [,2] [,3] [,4]
[1,]    2    1    1    1
[2,]    1    2    1    1
[3,]    1    1    2    1
[4,]    1    1    1    2
> A.eigen=eigen(A,symmetric=T)
> A.eigen
$values
[1] 5 1 1 1

$vectors
     [,1]       [,2]          [,3]       [,4]
[1,]  0.5  0.8660254  0.000000e+00  0.0000000
[2,]  0.5 -0.2886751 -6.408849e-17  0.8164966
[3,]  0.5 -0.2886751 -7.071068e-01 -0.4082483
[4,]  0.5 -0.2886751  7.071068e-01 -0.4082483

> A.eigen$vectors%*%diag(A.eigen$values)%*%t(A.eigen$vectors)
     [,1] [,2] [,3] [,4]
[1,]    2    1    1    1
[2,]    1    2    1    1
[3,]    1    1    2    1
[4,]    1    1    1    2
```

```
> t(A.eigen$vectors)%*%A.eigen$vectors
              [,1]          [,2]          [,3]          [,4]
[1,]  1.000000e+00  4.377466e-17  1.626303e-17 -5.095750e-18
[2,]  4.377466e-17  1.000000e+00 -1.694066e-18  6.349359e-18
[3,]  1.626303e-17 -1.694066e-18  1.000000e+00 -1.088268e-16
[4,] -5.095750e-18  6.349359e-18 -1.088268e-16  1.000000e+00
```

10 矩阵的 Choleskey 分解

对于正定矩阵 **A**，可对其进行 Choleskey 分解，即：**A=P'P**，其中 **P** 为上三角矩阵，在 R 中可以用函数 `chol()`进行 Choleskey 分解，例如：

```
> A
     [,1] [,2] [,3] [,4]
[1,]    2    1    1    1
[2,]    1    2    1    1
[3,]    1    1    2    1
[4,]    1    1    1    2
> chol(A)
         [,1]      [,2]      [,3]      [,4]
[1,] 1.414214 0.7071068 0.7071068 0.7071068
[2,] 0.000000 1.2247449 0.4082483 0.4082483
[3,] 0.000000 0.0000000 1.1547005 0.2886751
[4,] 0.000000 0.0000000 0.0000000 1.1180340
> t(chol(A))%*%chol(A)
     [,1] [,2] [,3] [,4]
[1,]    2    1    1    1
[2,]    1    2    1    1
[3,]    1    1    2    1
[4,]    1    1    1    2
> crossprod(chol(A),chol(A))
     [,1] [,2] [,3] [,4]
[1,]    2    1    1    1
[2,]    1    2    1    1
[3,]    1    1    2    1
[4,]    1    1    1    2
```

若矩阵为对称正定矩阵，可以利用 Choleskey 分解求行列式的值，如：

```
> prod(diag(chol(A))^2)
[1] 5
> det(A)
[1] 5
```

若矩阵为对称正定矩阵，可以利用Choleskey分解求矩阵的逆，这时用函数 `chol2inv()`，这种用法更有效。如：

```
> chol2inv(chol(A))
     [,1] [,2] [,3] [,4]
[1,]  0.8 -0.2 -0.2 -0.2
[2,] -0.2  0.8 -0.2 -0.2
```

```
[3,] -0.2 -0.2  0.8 -0.2
[4,] -0.2 -0.2 -0.2  0.8
> solve(A)
     [,1] [,2] [,3] [,4]
[1,]  0.8 -0.2 -0.2 -0.2
[2,] -0.2  0.8 -0.2 -0.2
[3,] -0.2 -0.2  0.8 -0.2
[4,] -0.2 -0.2 -0.2  0.8
```

11 矩阵奇异值分解

A为$m\times n$矩阵，rank(A)= r, 可以分解为：**A**=**UDV'**,其中**U'U**=**V'V**=**I**。在R中可以用函数scd()进行奇异值分解，例如：

```
> A=matrix(1:18,3,6)
> A
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    4    7   10   13   16
[2,]    2    5    8   11   14   17
[3,]    3    6    9   12   15   18
> svd(A)
$d
[1] 4.589453e+01 1.640705e+00 3.627301e-16
$u
           [,1]        [,2]       [,3]
[1,] -0.5290354  0.74394551  0.4082483
[2,] -0.5760715  0.03840487 -0.8164966
[3,] -0.6231077 -0.66713577  0.4082483
$v
            [,1]       [,2]        [,3]
[1,] -0.07736219 -0.7196003 -0.18918124
[2,] -0.19033085 -0.5089325  0.42405898
[3,] -0.30329950 -0.2982646 -0.45330031
[4,] -0.41626816 -0.0875968 -0.01637004
[5,] -0.52923682  0.1230711  0.64231130
[6,] -0.64220548  0.3337389 -0.40751869
> A.svd=svd(A)
> A.svd$u%*%diag(A.svd$d)%*%t(A.svd$v)
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    4    7   10   13   16
[2,]    2    5    8   11   14   17
[3,]    3    6    9   12   15   18
> t(A.svd$u)%*%A.svd$u
              [,1]          [,2]          [,3]
[1,]  1.000000e+00 -1.169312e-16 -3.016793e-17
[2,] -1.169312e-16  1.000000e+00 -3.678156e-17
[3,] -3.016793e-17 -3.678156e-17  1.000000e+00
```

```
> t(A.svd$v)%*%A.svd$v
              [,1]          [,2]          [,3]
[1,]  1.000000e+00  8.248068e-17 -3.903128e-18
[2,]  8.248068e-17  1.000000e+00 -2.103352e-17
[3,] -3.903128e-18 -2.103352e-17  1.000000e+00
```

12 矩阵QR分解

A为$m \times n$矩阵可以进行QR分解，**A**=**QR**，其中：**Q'Q**＝**I**，在R中可以用函数qr()进行QR分解，例如：

```
> A=matrix(1:16,4,4)
> qr(A)
$qr
           [,1]        [,2]          [,3]          [,4]
[1,] -5.4772256 -12.7801930 -2.008316e+01 -2.738613e+01
[2,]  0.3651484  -3.2659863 -6.531973e+00 -9.797959e+00
[3,]  0.5477226  -0.3781696  2.641083e-15  2.056562e-15
[4,]  0.7302967  -0.9124744  8.583032e-01 -2.111449e-16

$rank
[1] 2

$qraux
[1] 1.182574e+00 1.156135e+00 1.513143e+00 2.111449e-16

$pivot
[1] 1 2 3 4

attr(,"class")
[1] "qr"
```

rank项返回矩阵的秩，qr项包含了矩阵Q和R的信息，要得到矩阵**Q**和**R**，可以用函数qr.Q()和qr.R()作用qr()的返回结果，例如：

```
> qr.R(qr(A))
          [,1]       [,2]          [,3]          [,4]
[1,] -5.477226 -12.780193 -2.008316e+01 -2.738613e+01
[2,]  0.000000  -3.265986 -6.531973e+00 -9.797959e+00
[3,]  0.000000   0.000000  2.641083e-15  2.056562e-15
[4,]  0.000000   0.000000  0.000000e+00 -2.111449e-16
> qr.Q(qr(A))
           [,1]          [,2]       [,3]        [,4]
[1,] -0.1825742 -8.164966e-01 -0.4000874 -0.37407225
[2,] -0.3651484 -4.082483e-01  0.2546329  0.79697056
[3,] -0.5477226 -8.131516e-19  0.6909965 -0.47172438
[4,] -0.7302967  4.082483e-01 -0.5455419  0.04882607
> qr.Q(qr(A))%*%qr.R(qr(A))
     [,1] [,2] [,3] [,4]
```

```
[1,]    1    5    9   13
[2,]    2    6   10   14
[3,]    3    7   11   15
[4,]    4    8   12   16
> t(qr.Q(qr(A)))%*%qr.Q(qr(A))
              [,1]          [,2]          [,3]          [,4]
[1,]  1.000000e+00 -1.457168e-16 -6.760001e-17 -7.659550e-17
[2,] -1.457168e-16  1.000000e+00 -4.269046e-17  7.011739e-17
[3,] -6.760001e-17 -4.269046e-17  1.000000e+00 -1.596437e-16
[4,] -7.659550e-17  7.011739e-17 -1.596437e-16  1.000000e+00
> qr.X(qr(A))
     [,1] [,2] [,3] [,4]
[1,]    1    5    9   13
[2,]    2    6   10   14
[3,]    3    7   11   15
[4,]    4    8   12   16
```

13 矩阵广义逆(Moore-Penrose)

$n\times m$矩阵$\mathbf{A}^{+}$称为$m\times n$矩阵$\mathbf{A}$的Moore-Penrose逆，如果它满足下列条件：

① $\mathbf{A}\mathbf{A}^{+}\mathbf{A}=\mathbf{A}$；②$\mathbf{A}^{+}\mathbf{A}\mathbf{A}^{+}=\mathbf{A}^{+}$；③$(\mathbf{A}\mathbf{A}^{+})^{H}=\mathbf{A}\mathbf{A}^{+}$；④$(\mathbf{A}^{+}\mathbf{A})^{H}=\mathbf{A}^{+}\mathbf{A}$

在R的MASS包中的函数ginv()可计算矩阵**A**的Moore-Penrose逆，例如：

```
library(“MASS”)
> A
     [,1] [,2] [,3] [,4]
[1,]    1    5    9   13
[2,]    2    6   10   14
[3,]    3    7   11   15
[4,]    4    8   12   16
> ginv(A)
       [,1]    [,2]  [,3]    [,4]
[1,] -0.285 -0.1075  0.07  0.2475
[2,] -0.145 -0.0525  0.04  0.1325
[3,] -0.005  0.0025  0.01  0.0175
[4,]  0.135  0.0575 -0.02 -0.0975
```

验证性质1：

```
> A%*%ginv(A)%*%A
     [,1] [,2] [,3] [,4]
[1,]    1    5    9   13
[2,]    2    6   10   14
[3,]    3    7   11   15
[4,]    4    8   12   16
```

验证性质2：

```
> ginv(A)%*%A%*%ginv(A)
       [,1]    [,2]  [,3]    [,4]
[1,] -0.285 -0.1075  0.07  0.2475
```

```
[2,] -0.145 -0.0525  0.04  0.1325
[3,] -0.005  0.0025  0.01  0.0175
[4,]  0.135  0.0575 -0.02 -0.0975
```

验证性质3:

```
> t(A%*%ginv(A))
     [,1] [,2] [,3] [,4]
[1,]  0.7  0.4  0.1 -0.2
[2,]  0.4  0.3  0.2  0.1
[3,]  0.1  0.2  0.3  0.4
[4,] -0.2  0.1  0.4  0.7
> A%*%ginv(A)
     [,1] [,2] [,3] [,4]
[1,]  0.7  0.4  0.1 -0.2
[2,]  0.4  0.3  0.2  0.1
[3,]  0.1  0.2  0.3  0.4
[4,] -0.2  0.1  0.4  0.7
```

验证性质4:

```
> t(ginv(A)%*%A)
     [,1] [,2] [,3] [,4]
[1,]  0.7  0.4  0.1 -0.2
[2,]  0.4  0.3  0.2  0.1
[3,]  0.1  0.2  0.3  0.4
[4,] -0.2  0.1  0.4  0.7
> ginv(A)%*%A
     [,1] [,2] [,3] [,4]
[1,]  0.7  0.4  0.1 -0.2
[2,]  0.4  0.3  0.2  0.1
[3,]  0.1  0.2  0.3  0.4
[4,] -0.2  0.1  0.4  0.7
```

14 矩阵Kronecker积

$n\times m$矩阵$\mathbf{A}$与$h\times k$矩阵$\mathbf{B}$的kronecker积为一个$nh\times mk$维矩阵，公式为：

$$\mathbf{A}_{m\times n}\otimes\mathbf{B}_{h\times k}=\begin{pmatrix} a_{11}\mathbf{B} & \cdots & a_{1n}\mathbf{B} \\ \vdots & \vdots & \vdots \\ a_{m1}\mathbf{B} & \cdots & a_{mn}\mathbf{B} \end{pmatrix}_{mh\times nk}$$

在R中kronecker积可以用函数kronecker()来计算，例如：

```
> A=matrix(1:4,2,2)
> B=matrix(rep(1,4),2,2)
> A
     [,1] [,2]
[1,]    1    3
[2,]    2    4
> B
     [,1] [,2]
```

```
[1,]    1    1
[2,]    1    1
> kronecker(A,B)
     [,1] [,2] [,3] [,4]
[1,]    1    1    3    3
[2,]    1    1    3    3
[3,]    2    2    4    4
[4,]    2    2    4    4
```

15 矩阵的维数

在R中很容易得到一个矩阵的维数，函数dim()将返回一个矩阵的维数，nrow()返回行数，ncol()返回列数，例如：

```
> A=matrix(1:12,3,4)
> A
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
> nrow(A)
[1] 3
> ncol(A)
[1] 4
```

16 矩阵的行和、列和、行平均与列平均

在R中很容易求得一个矩阵的各行的和、平均数与列的和、平均数，例如：

```
> A
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
> rowSums(A)
[1] 22 26 30
> rowMeans(A)
[1] 5.5 6.5 7.5
> colSums(A)
[1]  6 15 24 33
> colMeans(A)
[1]  2  5  8 11
```

17 矩阵X'X的逆

在统计计算中，我们常常需要计算这样矩阵的逆，如OLS估计中求系数矩阵。R中的包“strucchange”提供了有效的计算方法。

```
> args(solveCrossprod)
function (X, method = c("qr", "chol", "solve"))
```

其中：method指定求逆方法，选用“qr”效率最高，选用“chol”精度最高，选用“slove”与slove(crossprod(x,x))效果相同，例如：

```
> A=matrix(rnorm(16),4,4)
```

```
> solveCrossprod(A,method="qr")
           [,1]       [,2]       [,3]       [,4]
[1,]  0.6132102 -0.1543924 -0.2900796  0.2054730
[2,] -0.1543924  0.4779277  0.1859490 -0.2097302
[3,] -0.2900796  0.1859490  0.6931232 -0.3162961
[4,]  0.2054730 -0.2097302 -0.3162961  0.3447627
> solveCrossprod(A,method="chol")
           [,1]       [,2]       [,3]       [,4]
[1,]  0.6132102 -0.1543924 -0.2900796  0.2054730
[2,] -0.1543924  0.4779277  0.1859490 -0.2097302
[3,] -0.2900796  0.1859490  0.6931232 -0.3162961
[4,]  0.2054730 -0.2097302 -0.3162961  0.3447627
> solveCrossprod(A,method="solve")
           [,1]       [,2]       [,3]       [,4]
[1,]  0.6132102 -0.1543924 -0.2900796  0.2054730
[2,] -0.1543924  0.4779277  0.1859490 -0.2097302
[3,] -0.2900796  0.1859490  0.6931232 -0.3162961
[4,]  0.2054730 -0.2097302 -0.3162961  0.3447627
> solve(crossprod(A,A))
           [,1]       [,2]       [,3]       [,4]
[1,]  0.6132102 -0.1543924 -0.2900796  0.2054730
[2,] -0.1543924  0.4779277  0.1859490 -0.2097302
[3,] -0.2900796  0.1859490  0.6931232 -0.3162961
[4,]  0.2054730 -0.2097302 -0.3162961  0.3447627
```

18 取矩阵的上、下三角部分

在R中，我们可以很方便的取到一个矩阵的上、下三角部分的元素，函数lower.tri()和函数upper.tri()提供了有效的方法。

```
> args(lower.tri)
function (x, diag = FALSE)
```

函数将返回一个逻辑值矩阵，其中下三角部分为真，上三角部分为假，选项diag为真时包含对角元素，为假时不包含对角元素。upper.tri()的效果与之孑然相反。例如：

```
> A
     [,1] [,2] [,3] [,4]
[1,]    1    5    9   13
[2,]    2    6   10   14
[3,]    3    7   11   15
[4,]    4    8   12   16
> lower.tri(A)
      [,1]  [,2]  [,3]  [,4]
[1,] FALSE FALSE FALSE FALSE
[2,]  TRUE FALSE FALSE FALSE
[3,]  TRUE  TRUE FALSE FALSE
[4,]  TRUE  TRUE  TRUE FALSE
> lower.tri(A,diag=T)
```

```
     [,1]  [,2]  [,3]  [,4]
[1,] TRUE FALSE FALSE FALSE
[2,] TRUE  TRUE FALSE FALSE
[3,] TRUE  TRUE  TRUE FALSE
[4,] TRUE  TRUE  TRUE  TRUE
> upper.tri(A)
      [,1]  [,2]  [,3]  [,4]
[1,] FALSE  TRUE  TRUE  TRUE
[2,] FALSE FALSE  TRUE  TRUE
[3,] FALSE FALSE FALSE  TRUE
[4,] FALSE FALSE FALSE FALSE
> upper.tri(A,diag=T)
      [,1]  [,2]  [,3] [,4]
[1,]  TRUE  TRUE  TRUE TRUE
[2,] FALSE  TRUE  TRUE TRUE
[3,] FALSE FALSE  TRUE TRUE
[4,] FALSE FALSE FALSE TRUE
> A[lower.tri(A)]=0
> A
     [,1] [,2] [,3] [,4]
[1,]    1    5    9   13
[2,]    0    6   10   14
[3,]    0    0   11   15
[4,]    0    0    0   16
> A[upper.tri(A)]=0
> A
     [,1] [,2] [,3] [,4]
[1,]    1    0    0    0
[2,]    2    6    0    0
[3,]    3    7   11    0
[4,]    4    8   12   16
```

19 backsolve&fowardsolve函数

这两个函数用于解特殊线性方程组，其特殊之处在于系数矩阵为上或下三角。

```
> args(backsolve)
function (r, x, k = ncol(r), upper.tri = TRUE, transpose = FALSE)
> args(forwardsolve)
function (l, x, k = ncol(l), upper.tri = FALSE, transpose = FALSE)
```

其中：r或者l为$n \times n$维三角矩阵，x为$n \times 1$维向量，对给定不同的upper.tri和transpose的值，方程的形式不同，如：

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1,n} \\ a_{21} & a_{22} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix} \mathbf{b} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}$$

对于函数backsolve()而言，

upper.tri=T&transpose=T

$$\left(\begin{array}{ccccc|c} a_{11} & 0 & \cdots & \cdots & 0 & b_1 \\ a_{12} & a_{22} & 0 & \cdots & 0 & b_2 \\ \vdots & \vdots & \ddots & \ddots & \vdots & \vdots \\ a_{1,n-1} & a_{2,n-1} & \cdots & \ddots & 0 & \vdots \\ a_{1,n} & a_{2,n} & \cdots & \cdots & a_{nn} & b_n \end{array}\right)$$

upper.tri=T&transpose=F

$$\left(\begin{array}{ccccc|c} a_{11} & a_{12} & \cdots & a_{1,n-1} & a_{1,n} & b_1 \\ 0 & a_{22} & \cdots & a_{2,n-1} & a_{2,n} & b_2 \\ \vdots & 0 & \ddots & \ddots & \vdots & \vdots \\ \vdots & \vdots & \ddots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & \cdots & a_{nn} & b_n \end{array}\right)$$

upper.tri=F&transpose=T

$$\left(\begin{array}{ccccc|c} a_{11} & a_{21} & \cdots & a_{n-1,1} & a_{n,1} & b_1 \\ 0 & a_{22} & \cdots & a_{n-1,2} & a_{n,2} & b_2 \\ \vdots & 0 & \ddots & \ddots & \vdots & \vdots \\ \vdots & \vdots & \ddots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & \cdots & a_{nn} & b_n \end{array}\right)$$

upper.tri=F&transpose=F

$$\left(\begin{array}{ccccc|c} a_{11} & 0 & \cdots & \cdots & 0 & b_1 \\ a_{12} & a_{22} & 0 & \cdots & 0 & b_2 \\ \vdots & \vdots & \ddots & \ddots & \vdots & \vdots \\ a_{1,n-1} & a_{2,n-1} & \cdots & \ddots & 0 & \vdots \\ a_{1,n} & a_{2,n} & \cdots & \cdots & a_{nn} & b_n \end{array}\right)$$

例如：

```
> A=matrix(1:9,3,3)
> A
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
> x=c(1,2,3)
> x
[1] 1 2 3
> B=A
> B[upper.tri(B)]=0
> B
     [,1] [,2] [,3]
[1,]    1    0    0
```

```
[2,]    2    5    0
[3,]    3    6    9
> C=A
> C[lower.tri(C)]=0
> C
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    0    5    8
[3,]    0    0    9
> backsolve(A,x,upper.tri=T,transpose=T)
[1]  1.00000000 -0.40000000 -0.08888889
> solve(t(C),x)
[1]  1.00000000 -0.40000000 -0.08888889
> backsolve(A,x,upper.tri=T,transpose=F)
[1] -0.8000000 -0.1333333  0.3333333
> solve(C,x)
[1] -0.8000000 -0.1333333  0.3333333
> backsolve(A,x,upper.tri=F,transpose=T)
[1] 1.111307e-17 2.220446e-17 3.333333e-01
> solve(t(B),x)
[1] 1.110223e-17 2.220446e-17 3.333333e-01
> backsolve(A,x,upper.tri=F,transpose=F)
[1] 1 0 0
> solve(B,x)
[1]  1.000000e+00 -1.540744e-33 -1.850372e-17
```

对于函数forwardsolve()而言，

$$\underline{\texttt{upper.tri=T\&transpose=T}}$$

$$\left(\begin{array}{ccccc|c} a_{11} & 0 & \cdots & \cdots & 0 & b_1 \\ a_{12} & a_{22} & 0 & \cdots & 0 & b_2 \\ \vdots & \vdots & \ddots & \ddots & \vdots & \vdots \\ a_{1,n-1} & a_{2,n-1} & \cdots & \ddots & 0 & \vdots \\ a_{1,n} & a_{2,n} & \cdots & \cdots & a_{nn} & b_n \end{array}\right)$$

$$\underline{\texttt{upper.tri=T\&transpose=F}}$$

$$\left(\begin{array}{ccccc|c} a_{11} & a_{12} & \cdots & a_{1,n-1} & a_{1,n} & b_1 \\ 0 & a_{22} & \cdots & a_{2,n-1} & a_{2,n} & b_2 \\ \vdots & 0 & \ddots & \ddots & \vdots & \vdots \\ \vdots & \vdots & \ddots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & \cdots & a_{nn} & b_n \end{array}\right)$$

upper.tri=F&transpose=T

$$\left(\begin{array}{ccccc|c} a_{11} & a_{21} & \cdots & a_{n-1,1} & a_{n,1} & b_1 \\ 0 & a_{22} & \cdots & a_{n-1,2} & a_{n,2} & b_2 \\ \vdots & 0 & \ddots & \ddots & \vdots & \vdots \\ \vdots & \vdots & \ddots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & \cdots & a_{nn} & b_n \end{array}\right)$$

upper.tri=F&transpose=F

$$\left(\begin{array}{ccccc|c} a_{11} & 0 & \cdots & \cdots & 0 & b_1 \\ a_{12} & a_{22} & 0 & \cdots & 0 & b_2 \\ \vdots & \vdots & \ddots & \ddots & \vdots & \vdots \\ a_{1,n-1} & a_{2,n-1} & \cdots & \ddots & 0 & \vdots \\ a_{1,n} & a_{2,n} & \cdots & \cdots & a_{nn} & b_n \end{array}\right)$$

例如：

```
> A
    [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
> B
     [,1] [,2] [,3]
[1,]    1    0    0
[2,]    2    5    0
[3,]    3    6    9
> C
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    0    5    8
[3,]    0    0    9
> x
[1] 1 2 3
> forwardsolve(A,x,upper.tri=T,transpose=T)
[1]  1.00000000 -0.40000000 -0.08888889
> solve(t(C),x)
[1]  1.00000000 -0.40000000 -0.08888889
> forwardsolve(A,x,upper.tri=T,transpose=F)
[1] -0.8000000 -0.1333333  0.3333333
> solve(C,x)
[1] -0.8000000 -0.1333333  0.3333333
> forwardsolve(A,x,upper.tri=F,transpose=T)
[1] 1.111307e-17 2.220446e-17 3.333333e-01
> solve(t(B),x)
[1] 1.110223e-17 2.220446e-17 3.333333e-01
```

```
> forwardsolve(A,x,upper.tri=F,transpose=F)
[1] 1 0 0
> solve(B,x)
[1]  1.000000e+00 -1.540744e-33 -1.850372e-17
```

20 row()与col()函数

在R中定义了的这两个函数用于取矩阵元素的行或列下标矩阵，例如矩阵$\mathbf{A}=\{a_{ij}\}_{m\times n}$，row()函数将返回一个与矩阵$\mathbf{A}$有相同维数的矩阵，该矩阵的第$i$行第$j$列元素为$i$，函数col()类似。例如：

```
> x=matrix(1:12,3,4)
> row(x)
     [,1] [,2] [,3] [,4]
[1,]    1    1    1    1
[2,]    2    2    2    2
[3,]    3    3    3    3
> col(x)
     [,1] [,2] [,3] [,4]
[1,]    1    2    3    4
[2,]    1    2    3    4
[3,]    1    2    3    4
```

这两个函数同样可以用于取一个矩阵的上下三角矩阵，例如：

```
> x
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
> x[row(x)<col(x)]=0
> x
     [,1] [,2] [,3] [,4]
[1,]    1    0    0    0
[2,]    2    5    0    0
[3,]    3    6    9    0
> x=matrix(1:12,3,4)
> x[row(x)>col(x)]=0
> x
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    0    5    8   11
[3,]    0    0    9   12
```

21 行列式的值

在R中，函数det(x)将计算方阵x的行列式的值，例如：

```
> x=matrix(rnorm(16),4,4)
> x
           [,1]       [,2]       [,3]        [,4]
[1,] -1.0736375  0.2809563 -1.5796854  0.51810378
```

```
[2,] -1.6229898 -0.4175977  1.2038194 -0.06394986
[3,] -0.3989073 -0.8368334 -0.6374909 -0.23657088
[4,]  1.9413061  0.8338065 -1.5877162 -1.30568465
> det(x)
[1] 5.717667
```

22 向量化算子

记矩阵 $\mathbf{A}=\left\{a_{ij}\right\}_{m\times n}$， $vec(\mathbf{A})=\left(a_{11},\cdots,a_{m1},a_{12},\cdots a_{m2},\cdots,a_{1,n},\cdots,a_{mn}\right)'$

记矩阵 $\mathbf{B}=\left\{b_{ij}\right\}_{n\times n}$， $vech(\mathbf{B})=\left(b_{11},\cdots,b_{n1},b_{22},\cdots a_{n2},\cdots,a_{nn}\right)'$

在R中可以很容易的实现向量化算子，例如：

```
vec<-function (x)
    {
        t(t(as.vector(x)))
    }
vech<-function (x)
    {
       t(x[lower.tri(x,diag=T)])
    }
> x=matrix(1:12,3,4)
> x
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
> vec(x)
      [,1]
 [1,]    1
 [2,]    2
 [3,]    3
 [4,]    4
 [5,]    5
 [6,]    6
 [7,]    7
 [8,]    8
 [9,]    9
[10,]   10
[11,]   11
[12,]   12
> vech(x)
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    2    3    5    6    9
```

23 时间序列的滞后值

在时间序列分析中，我们常常要用到一个序列的滞后序列，R中的包“fMultivar”中

的函数tslag()提供了这个功能。

```
> args(tslag)
function (x, k = 1, trim = FALSE)
```

其中：x为一个向量，k指定滞后阶数，可以是一个自然数列，若trim为假，则返回序列与原序列长度相同，但含有NA值；若trim项为真，则返回序列中不含有NA值，例如：

```
> x=1:20
> tslag(x,1:4,trim=F)
      [,1] [,2] [,3] [,4]
 [1,]   NA   NA   NA   NA
 [2,]    1   NA   NA   NA
 [3,]    2    1   NA   NA
 [4,]    3    2    1   NA
 [5,]    4    3    2    1
 [6,]    5    4    3    2
 [7,]    6    5    4    3
 [8,]    7    6    5    4
 [9,]    8    7    6    5
[10,]    9    8    7    6
[11,]   10    9    8    7
[12,]   11   10    9    8
[13,]   12   11   10    9
[14,]   13   12   11   10
[15,]   14   13   12   11
[16,]   15   14   13   12
[17,]   16   15   14   13
[18,]   17   16   15   14
[19,]   18   17   16   15
[20,]   19   18   17   16
> tslag(x,1:4,trim=T)
      [,1] [,2] [,3] [,4]
 [1,]    4    3    2    1
 [2,]    5    4    3    2
 [3,]    6    5    4    3
 [4,]    7    6    5    4
 [5,]    8    7    6    5
 [6,]    9    8    7    6
 [7,]   10    9    8    7
 [8,]   11   10    9    8
 [9,]   12   11   10    9
[10,]   13   12   11   10
[11,]   14   13   12   11
[12,]   15   14   13   12
[13,]   16   15   14   13
[14,]   17   16   15   14
```

```
[15,]   18   17   16   15
[16,]   19   18   17   16
```