矩阵运算 by R

王宁宁

目录

```
rm(list = ls(all = TRUE))
```

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)
## NULL
```

```
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
```

```r
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
```

```r
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
```

```r
rowname=c("r1", "r2", "r3")
rowname
```

```
## [1] "r1" "r2" "r3"
```

```r
colname=c("c1","c2","c3","c4")
colname
```

```
## [1] "c1" "c2" "c3" "c4"
```

3 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
```

```
dim(A)
```

```
## [1] 3 4
```

```
nrow(A)
```

```
## [1] 3
```

```
ncol(A)
```

```
## [1] 4
```

4 求矩阵的秩

```
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
```

```
qr(A)$rank
```

```
## [1] 2
```

5 矩阵的行和、列和、行平均与列平均

```r
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
```

```r
rowSums(A)
```

```
## [1] 22 26 30
```

```r
rowMeans(A)
```

```
## [1] 5.5 6.5 7.5
```

```r
colSums(A)
```

```
## [1]  6 15 24 33
```

```r
colMeans(A)
```

```
## [1]  2  5  8 11
```

上述关于矩阵行和列的操作，还可以使用 apply() 函数实现。

```r
args(apply)
```

```
## function (X, MARGIN, FUN, ...)
## NULL
```

其中:x 为矩阵,MARGIN 用来指定是对行运算还是对列运算,MARGIN = 1 表示对行运算，MARGIN = 2 表示对列运算，FUN 用来指定运算函数, ...用来给定 FUN 中需要的其它的参数，例如：

```
 apply(A,1,sum)
```

```
## [1] 22 26 30
```

```
 apply(A,1,mean)
```

```
## [1] 5.5 6.5 7.5
```

```
apply(A,2,sum)
```

```
## [1]  6 15 24 33
```

```
apply(A,2,mean)
```

```
## [1]  2  5  8 11
```

apply() 函数功能强大，我们可以对矩阵的行或者列进行其它运算，例如：

```
A=matrix(rnorm(10000),2000,5)
 apply(A,2,var)
```

```
## [1] 1.0004521 0.9835358 0.9979132 0.9660764 1.0246545
```

```
 #apply(A,2,function(x,a)x*a,a=2)
```

6 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)
## NULL
```

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

```
##             [,1]        [,2]       [,3]       [,4]
## [1,] -0.13217425  0.29026614 -1.1367271 -0.6979347
## [2,]  0.02181711  2.01297716 -1.4372947 -0.5813031
## [3,] -0.10939907  0.02037605  2.2079625 -1.0996903
## [4,]  1.36559190 -0.12166597  0.4710891 -0.3629068
```

```
 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,] -0.1321742 0.2902661 -1.136727 -0.6979347
## [2,]  0.0000000 2.0129772 -1.437295 -0.5813031
## [3,]  0.0000000 0.0000000  2.207963 -1.0996903
## [4,]  0.0000000 0.0000000  0.000000 -0.3629068
```

```
 A[upper.tri(A)]=0
 A
```

```
##            [,1]     [,2]     [,3]       [,4]
## [1,] -0.1321742 0.000000 0.000000  0.0000000
## [2,]  0.0000000 2.012977 0.000000  0.0000000
## [3,]  0.0000000 0.000000 2.207963  0.0000000
## [4,]  0.0000000 0.000000 0.000000 -0.3629068
```

7 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 $A=[a_{ij}]_{m\times n}$

```
 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
```

```r
row(x)
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    1    1    1
## [2,]    2    2    2    2
## [3,]    3    3    3    3
```

```r
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：

```r
x
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    4    7   10
## [2,]    2    5    8   11
## [3,]    3    6    9   12
```

```r
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
```

```r
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
```

8 矩阵转置

A 为 m×n 矩阵，求 A' 在 R 中可用函数 t()，例如：

```r
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
```

```r
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（矩阵），例如：

```r
x
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    4    7   10
```

```
## [2,]    0    5    8   11
## [3,]    0    0    9   12
```

```r
t(x)
```

```
##      [,1] [,2] [,3]
## [1,]    1    0    0
## [2,]    4    5    0
## [3,]    7    8    9
## [4,]   10   11   12
```

```r
class(x)
```

```
## [1] "matrix"
```

```r
class(t(x))
```

```
## [1] "matrix"
```

可用 t(t(x)) 得到一个列向量：

```r
x
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    4    7   10
## [2,]    0    5    8   11
## [3,]    0    0    9   12
```

```r
t(t(x))
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    4    7   10
## [2,]    0    5    8   11
## [3,]    0    0    9   12
```

```
y=t(t(x))
t(t(y))
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    4    7   10
## [2,]    0    5    8   11
## [3,]    0    0    9   12
```

9 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,]  1.0417890 -0.7306732  1.2048320  0.9723702
## [2,]  0.8022733 -2.2449170 -0.6517073 -0.6026585
## [3,] -1.0621420 -1.0367986  0.8017808 -0.4837199
## [4,]  0.8139164 -1.2551169 -1.0616901 -1.2185154
```

```
det(x)
```

```
## [1] 4.64723
```

10 矩阵加减

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,]    1    4    7   10
## [2,]    2    5    8   11
## [3,]    3    6    9   12
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    4    7   10
## [2,]    2    5    8   11
## [3,]    3    6    9   12
```

```
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
```

11 矩阵数乘

A 为 m×n 矩阵，c>0，在 R 中求 cA 可用符号：“*”，例如：

```
c=2
A
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    4    7   10
## [2,]    2    5    8   11
## [3,]    3    6    9   12
```

```
c*A
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    2    8   14   20
## [2,]    4   10   16   22
## [3,]    6   12   18   24
```

12 矩阵相乘

A 为 m×n 矩阵，B 为 n×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×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);A
```

```
##      [,1] [,2] [,3]
## [1,]    1    5    9
## [2,]    2    6   10
## [3,]    3    7   11
## [4,]    4    8   12
```

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

```
##      [,1] [,2] [,3]
## [1,]    1    5    9
## [2,]    2    6   10
## [3,]    3    7   11
## [4,]    4    8   12
```

```
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
```

13 向量化算子

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
```

14 矩阵 Hadamard 积

若 A={aij}m×n, B={bij}m×n, 则矩阵的 Hadamard 积定义为: A B={aij bij }m×n,R 中 Hadamard 积可以直接运用运算符 "*" 例如:

```
A=matrix(1:16,4,4)
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
```

```
B=A
A*B
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1   25   81  169
```

```
## [2,]    4   36  100  196
## [3,]    9   49  121  225
## [4,]   16   64  144  256
```

15 矩阵 Kronecker 积

n×m 矩阵 A 与 h×k 矩阵 B 的 kronecker 积为一个 nh×mk 维矩阵，在 R 中 kronecker 积可以用函数 kronecker() 来计算，例如：

```
A=matrix(1:4,2,2);A
```

```
##      [,1] [,2]
## [1,]    1    3
## [2,]    2    4
```

```
B=matrix(rep(1,4),2,2);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
```

注：这两种积在矩阵求导里很常用，更多信息可以看这里。

16 矩阵对角元素相关运算

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
```

17 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 $ax = b$，若 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,]  0.6009152 -1.5732064  0.1468049  0.4462512
## [2,] -0.7712984 -0.4610141  0.4108327  0.2259612
## [3,]  1.0116070  0.9637897 -0.3051251 -0.5726910
## [4,]  1.0932814 -1.1849586  0.1249404  0.5726790
```

```
solve(a)
```

```
##            [,1]       [,2]        [,3]      [,4]
## [1,] -0.2182523  0.3546345  0.65167897 0.6818346
## [2,] -1.5076959  0.1630261 -0.08664279 1.0238788
## [3,] -0.6987859  3.9427547  2.16942364 1.1582996
## [4,] -2.5505378 -1.1998788 -1.89667378 2.3103666
```

```
solve (a) %*%a
```

```
##      [,1]         [,2]          [,3]          [,4]
## [1,]    1 1.110223e-16 -5.551115e-17 -1.665335e-16
## [2,]    0 1.000000e+00 -5.551115e-17  0.000000e+00
## [3,]    0 0.000000e+00  1.000000e+00 -2.220446e-16
## [4,]    0 0.000000e+00 -5.551115e-17  1.000000e+00
```

18 矩阵的特征值与特征向量

```
args(eigen)
```

```
## function (x, symmetric, only.values = FALSE, EISPACK = FALSE)
## NULL
```

```
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  0.0000000  0.0000000
## [2,] -0.5 -0.2886751 -0.5773503 -0.5773503
## [3,] -0.5 -0.2886751 -0.2113249  0.7886751
## [4,] -0.5 -0.2886751  0.7886751 -0.2113249
```

```
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
```

19 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：

```r
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
```

```r
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
```

```r
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
```

```r
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
```

20 矩阵奇异值分解

A 为 m×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 1.366522e-15
##
## $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 -0.71960032 -0.4076688
## [2,] -0.19033085 -0.50893247  0.5745647
## [3,] -0.30329950 -0.29826463 -0.0280114
## [4,] -0.41626816 -0.08759679  0.2226621
## [5,] -0.52923682  0.12307105 -0.6212052
## [6,] -0.64220548  0.33373889  0.2596585
```

```
A.svd=svd(A);A.svd
```

```
## $d
## [1] 4.589453e+01 1.640705e+00 1.366522e-15
##
## $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 -0.71960032 -0.4076688
## [2,] -0.19033085 -0.50893247  0.5745647
## [3,] -0.30329950 -0.29826463 -0.0280114
## [4,] -0.41626816 -0.08759679  0.2226621
## [5,] -0.52923682  0.12307105 -0.6212052
## [6,] -0.64220548  0.33373889  0.2596585
```

```
(A.svd$u)%*%diag(A.svd$d)
```

```
##           [,1]        [,2]          [,3]
## [1,] -24.27983  1.22059535  5.578802e-16
## [2,] -26.43853  0.06301108 -1.115760e-15
## [3,] -28.59724 -1.09457320  5.578802e-16
```

```
t(A.svd$u)
```

```
##            [,1]        [,2]       [,3]
## [1,] -0.5290354 -0.57607152 -0.6231077
## [2,]  0.7439455  0.03840487 -0.6671358
## [3,]  0.4082483 -0.81649658  0.4082483
```

```
t(A.svd$v)
```

```
##             [,1]       [,2]       [,3]        [,4]       [,5]       [,6]
## [1,] -0.07736219 -0.1903308 -0.3032995 -0.41626816 -0.5292368 -0.6422055
## [2,] -0.71960032 -0.5089325 -0.2982646 -0.08759679  0.1230711  0.3337389
## [3,] -0.40766875  0.5745647 -0.0280114  0.22266214 -0.6212052  0.2596585
```

21 矩阵 QR 分解

A 为 m×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：

```r
A=matrix(1:16,4,4);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
```

```r
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  1.601186e-15  2.217027e-15
## [4,]  0.7302967  -0.9124744 -5.547002e-01 -1.478018e-15
##
## $rank
## [1] 2
##
## $qraux
## [1] 1.182574e+00 1.156135e+00 1.832050e+00 1.478018e-15
##
## $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：

```r
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  1.601186e-15  2.217027e-15
## [4,]  0.000000   0.000000  0.000000e+00 -1.478018e-15
```

```
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 -1.665335e-16  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 -5.551115e-17  0.000000e+00  2.081668e-17
## [2,] -5.551115e-17  1.000000e+00 -2.775558e-17 -6.938894e-17
## [3,]  0.000000e+00 -2.775558e-17  1.000000e+00  2.775558e-17
## [4,]  2.081668e-17 -6.938894e-17  2.775558e-17  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
```