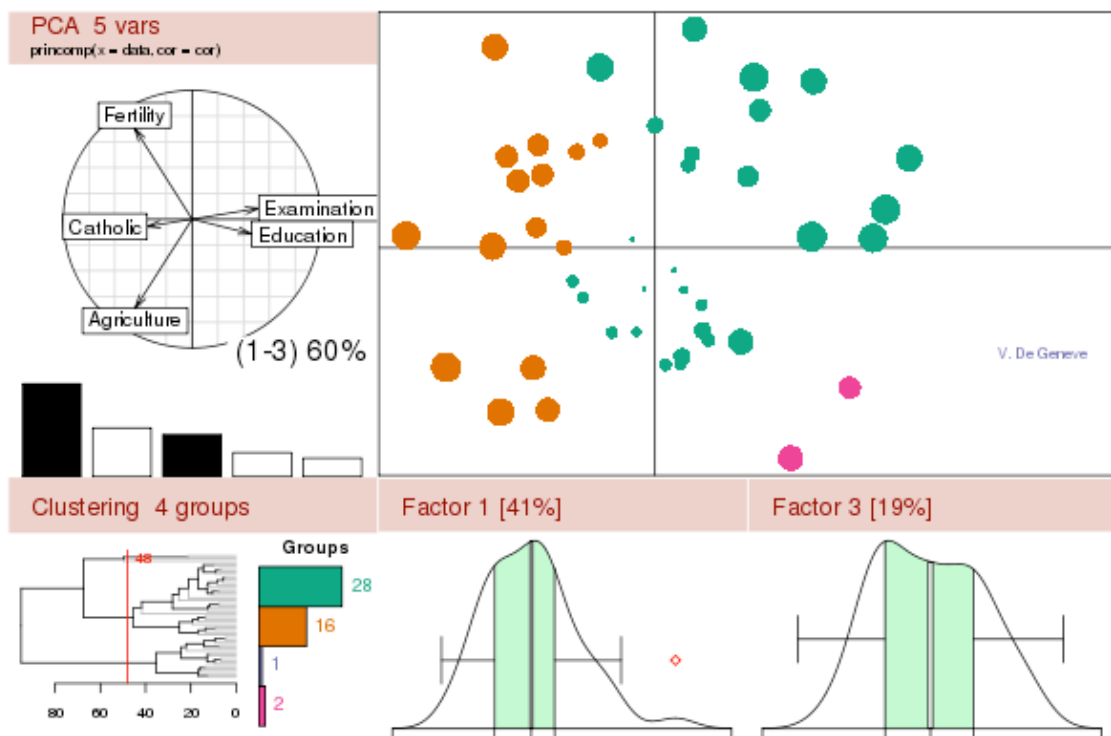


# 행렬에 대해서

## Useful Data Objects

유 충현

블로그 모음 10탄(<http://blog.naver.com/bdboys>) • (주)오픈베이스 • 2010년 10월 10일



## 행렬의 생성

R에서 행렬은 동일한 데이터 유형을 갖는  $m$  by  $n$ 의 자료형을 의미한다. 수학 세계에서는 수치형만 정의하였는데 R에서는 논리형과 문자형의 행렬도 존재한다.

그리고 집합 관계로 따지면 스칼라  $\subset$  벡터  $\subset$  행렬 의 관계가 성립한다.

행의 수가 1인  $1$  by  $n$  행렬은 행벡터이고, 열의 수가 1인  $m$  by  $1$ 의 행렬은 열벡터이다. 그리고 원소의 개수가 1인 벡터는 스칼라라고 할 수 있기 때문이다. 이처럼 행렬과 벡터는 밀접한 관계에 놓여 있고, 행렬을 정의할 때는 벡터를 먼저 정의하고 행렬을 정의하는 것이 일반적이다.

그럼 행렬을 정의하는 방법에 대해서 알아 보자.

### 1. matrix 함수의 이용

행렬을 만드는 함수로 가장 많이 사용되는 함수로 함수의 원형은 다음과 같다.

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

`data`는 행렬의 원소를 포함하는 벡터값으로 선택 인수이다. `nrow`는 행의 수를 지정하며, `ncol`은 열의 수를 지정한다. `byrow`는 행우선 배치의 여부를 지정한다.

`dimnames`는 차원의 이름을 지정한다. 이 함수는 필수 인수가 없고, 모두 선택인수이다.

이 함수의 기본값을 보자. 원소가 NA인  $1$  by  $1$  행렬을 만든다.

```
> matrix()
     [,1]
[1,] NA
```

원소가 1부터 4인 정수를 가지고 행의 수가 2인 행렬을 만든다. 자동적으로 열의 수도 2가 된다.

```
> matrix(1:4,nrow=2)
     [,1] [,2]
[1,]  1   3
[2,]  2   4
```

원소가 1부터 4인 정수를 가지고 행의 수가 2인 행렬을 만드는데 행을 우선적으로 배치한다.

```
> matrix(1:4,nrow=2, byrow=T)
     [,1] [,2]
[1,]  1   2
[2,]  3   4
```

원소가 1부터 4인 정수를 가지고 행의 수가 4인 행렬을 만든다.

```
> matrix(1:4,nrow=4)
     [,1]
[1,]  1
[2,]  2
[3,]  3
[4,]  4
```

4개의 원소를 가지고 행이 3인 행렬을 만들려고하니 오류가 발생했다.

nrow \* ncol = length(data)를 만족해야 한다.

```
> matrix(1:4,nrow=3)
     [,1] [,2]
[1,]  1   4
[2,]  2   1
[3,]  3   2
```

Warning message:

```
data length [4] is not a sub-multiple or multiple of the number of rows
[3] in matrix
```

2 by 3의 행렬을 만들고 차원의 이름을 지정하였다.

```
> matrix(c(1,2,3, 11,12,13), nrow = 2, ncol=3, byrow=TRUE,
+ dimnames = list(c("row1", "row2"), c("C.1", "C.2", "C.3")))
     C.1 C.2 C.3
row1  1  2  3
row2 11 12 13
```

동일하게 2 by 3의 행렬을 만들었다.

```
> mat = matrix(c(1,2,3, 11,12,13), nrow = 2, ncol=3, byrow=TRUE)
> mat
     [,1] [,2] [,3]
[1,]  1   2   3
[2,] 11  12  13
```

차원의 이름을 matrix 함수의 밖에서 만들 수도 있다.

```
> dimnames(mat) = list(c("row1", "row2"), c("C.1", "C.2", "C.3"))
> mat
     C.1 C.2 C.3
```

```
row1  1  2  3
row2 11 12 13
```

## 2. as.matrix 함수의 이용

as.matrix도 행렬을 생성하는 함수이다. 생성이라기 보다는 전환이라는 것이 더 맞는 말이다.

1부터 6의 정수를 원소로 갖는 벡터

```
> 1:6
[1] 1 2 3 4 5 6
```

1부터 6의 정수를 원소로 갖는 벡터 를 as.matrix함수를 이용해서 행렬로 전환한 뒤 x에 할당한다.

```
> x = as.matrix(1:6)
```

역시 6 by 1(n by 1)행렬로 변환되었다.

```
> x
      [,1]
[1,]    1
[2,]    2
[3,]    3
[4,]    4
[5,]    5
[6,]    6
```

6 by 1 행렬인 x를 2 by 3인 dim 함수를 이용해서 행렬로 변환하였다.

```
> dim(x) = c(2,3)
```

```
> x
      [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
```

## 3. t 함수의 이용

t함수는 전치행렬(Transpose Matrix)을 구하는 함수이다. 이 함수를 이용해서 제한적이거나 벡터를 행렬로 만들 수 있다.

1부터 4의 원소를 갖는 벡터를 만들어 보자. 열벡터일까? 행벡터일까?

```
> 1:4
```

```
[1] 1 2 3 4
```

일단은 행렬이 아니라고 한다.

```
> is.matrix(1:4)
```

```
[1] FALSE
```

그러면 이 벡터의 전치행렬을 구해보자. 행벡터로 결과가 나왔다. 그러면 1:4는 열벡터인가? 왜냐하면 행벡터의 전치행렬은 열벡터이고, 열벡터의 전치행렬은 행벡터이기 때문이다. 엄밀히 말해서 R의 자료형에 의하면 벡터는 행렬이 아니므로 열벡터, 행벡터의 의미가 없이 그냥 벡터인 것이다. 즉 1 by n 행렬인 것이다.

```
> t(1:4)
```

```
 [1] [2] [3] [4]
```

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

이 전치행렬은 행렬이다.

```
> is.matrix(t(1:4))
```

```
[1] TRUE
```

그럼 벡터의 전치행렬의 전치행렬은 무엇일까? n by 1 행렬이다.

```
> t(t(1:4))
```

```
 [1]
```

```
[1,] 1
```

```
[2,] 2
```

```
[3,] 3
```

```
[4,] 4
```

이것 또한 행렬이다.

```
> is.matrix(t(t(1:4)))
```

```
[1] TRUE
```

이방법의 단점은 1 by n이나 n by 1의 행렬만 만들 수 있다. 그러므로 제한적인 용도로만 사용할 수 있다.

#### 4. cbind, rbind의 이용 (Combine Rows & Cols)

벡터들을 행으로 묶거나 열로 묶어 행렬을 만들 수 있다. 이 때 사용하는 함수가 cbind, rbind이다. 함수의 원형은 다음과 같다.

```
cbind(..., deparse.level = 1)
rbind(..., deparse.level = 1)
```

1이라는 스칼라와 1:4이라는 벡터를 열로 붙혀 4 by 2 행렬을 만들었다. 여기서 중요한 점은 R에서 길이가 다른 두 자료를 이용한 작업에서는 길이가 짧은 쪽의 자료를 긴쪽의 자료의 길이와 맞추려는 성향이 있다.

그래서 스칼라 1이 rep(1,4)의 자료로 변환되어 계산되었다.

```
> cbind(1, 1:4)
  [,1] [,2]
[1,]  1   1
[2,]  1   2
[3,]  1   3
[4,]  1   4
```

행렬이 맞다.

```
> is.matrix(cbind(1, 1:4))
[1] TRUE
```

1이라는 스칼라와 1:4이라는 벡터를 행으로 붙혀 2 by 4 행렬을 만들었다.

```
> rbind(1, 1:4)
  [,1] [,2] [,3] [,4]
[1,]  1   1   1   1
[2,]  1   2   3   4
```

1:2이라는 벡터와 1:4이라는 벡터를 열로 붙혀 4 by 2 행렬을 만들었다.

벡터 1:2가 rep(1:2,2)의 자료로 변환되어 계산되었다.

```
> cbind(1:2, 1:4)
  [,1] [,2]
[1,]  1   1
[2,]  2   2
[3,]  1   3
[4,]  2   4
```

1:3이라는 벡터와 1:4이라는 벡터를 옆로 붙혀 4 by 2 행렬을 만들었다.  
1:3가 길이 4인 벡터로 변환되어 계산되었지만 경고 메시지가 출력되었다.  
큰 쪽의 길이가 작은 쪽의 길이에 배수가 아닐경우에 발생하는 경고 메시지다.

```
> cbind(1:3, 1:4)
```

```
  [,1] [,2]
```

```
[1,]  1  1
```

```
[2,]  2  2
```

```
[3,]  3  3
```

```
[4,]  1  4
```

Warning message:

number of rows of result

is not a multiple of vector length (arg 1) in: cbind(1:3, 1:4)

벡터뿐만 아니라 행렬도 묶을 수 있다.

x라는 3 by 2 행렬을 만든다.

```
> x = matrix(1:6, nrow=3)
```

```
> x
```

```
  [,1] [,2]
```

```
[1,]  1  4
```

```
[2,]  2  5
```

```
[3,]  3  6
```

x 행렬과 7:9의 벡터를 옆로 묶어 3 by 3 행렬을 만들었다.

```
> cbind(x,7:9)
```

```
  [,1] [,2] [,3]
```

```
[1,]  1  4  7
```

```
[2,]  2  5  8
```

```
[3,]  3  6  9
```

## 행렬의 연산

행렬의 기본적인 연산에 대해서 살펴보자. 행렬대수는 통계계산의 핵심이라고 할 수 있을 정도로 많은 분석 방법에 이용되고 있다.

### 1. 전치행렬 구하기

전치행렬은 행과 열의 값이 바뀐 행렬을 의미한다. 앞서 살펴본대로 t함수를 사용한다.

3 by 2 행렬 x를 만든다.

```
> x = matrix(1:6, nrow=3)
> x
  [1,] [2,]
[1,]  1  4
[2,]  2  5
[3,]  3  6
```

t함수를 이용해서 2 by 3의 전치행렬을 만든다.

```
> t(x)
  [1,] [2,] [3,]
[1,]  1  2  3
[2,]  4  5  6
```

전치행렬의 전치행렬을 만든다. 원래의 행렬이 된다.

```
> t(t(x))
  [1,] [2,]
[1,]  1  4
[2,]  2  5
[3,]  3  6
```

그러면 전치행렬을 만드는 함수를 transpose라는 이름으로 정의해 보자.

```
transpose <- function (x)
{
  if (!is.matrix(x)) stop("입력값은 행렬이 아닙니다.")

  # 행과 열을 바꿔 초기화 하였다.
  tmp <- matrix(as.vector(x), ncol=nrow(x))
  for(i in seq(nrow(x)))
    tmp[, i] <- x[i, ]
}
```



```
return (tmp)
}
```

앞서 만든 행렬 x의 전치행렬을 구한다.

```
> transpose(x)
  [,1] [,2] [,3]
[1,]  1  3  5
[2,]  2  4  6
```

전치행렬의 전치행렬을 만든다. 원래의 행렬이 된다.

```
> transpose(transpose(x))
  [,1] [,2]
[1,]  1  4
[2,]  2  5
[3,]  3  6
```

벡터를 가지고 전치행렬을 만든다. 오류가 발생한다.

```
> transpose(1:5)
Error in transpose(1:5) : 입력값은 행렬이 아닙니다.
```

## 2. 행렬의 덧셈과 뺄셈

차원이 같은 두개 이상의 행렬에 대해서 덧셈과 뺄셈이 가능하다. 동일한 행과 열의 위치에 있는 원소들을 더하거나 빼면 된다.

임의의 수로 a라는 3 by 2 행렬을 만들었다.

```
> a = matrix(sample(6,replace=T),nrow=3)
> a
  [,1] [,2]
[1,]  4  6
[2,]  6  3
[3,]  3  1
```

임의의 수로 b라는 3 by 2 행렬을 만들었다.

```
> b = matrix(sample(6,replace=T),nrow=3)
> b
  [,1] [,2]
[1,]  5  5
```

```
[2,] 1 4
[3,] 3 5
```

두행렬의 합

```
> a + b
      [,1] [,2]
[1,]  9  11
[2,]  7   7
[3,]  6   6
```

두 행렬의 차

```
> b - a
      [,1] [,2]
[1,]  1  -1
[2,] -5   1
[3,]  0   4
```

-와 +의 중복연산

```
> b - a + a
      [,1] [,2]
[1,]  5   5
[2,]  1   4
[3,]  3   5
```

그러면 행렬의 덧셈과 뺄셈의 함수를 만들어 보자.

```
add.matrix <- function (a, b)
{
  if (!is.matrix(a)) stop("입력값중 행렬이 아닌 것이 있습니다.")
  if (!is.matrix(b)) stop("입력값중 행렬이 아닌 것이 있습니다.")

  if(!prod(dim(a)==dim(b))) stop("두 행렬의 차원이 다릅니다.")

  tmp <- a
  for(i in seq(nrow(a)))
    for(j in seq(ncol(a)))
      tmp[i,j] <- a[i,j] + b[i,j]
  #      tmp[i,j] <- a[i,j] - b[i,j] 뺄셈의 경우
}
```

```

        return (tmp)
    }

> a = matrix(1:8, nrow=4)
> a
  [,1] [,2]
[1,]  1  5
[2,]  2  6
[3,]  3  7
[4,]  4  8

> b = matrix(8:1, nrow=4)
> b
  [,1] [,2]
[1,]  8  4
[2,]  7  3
[3,]  6  2
[4,]  5  1

> a+b
  [,1] [,2]
[1,]  9  9
[2,]  9  9
[3,]  9  9
[4,]  9  9

```

행렬의 덧셈

```

> add.matrix(a,b)
  [,1] [,2]
[1,]  9  9
[2,]  9  9
[3,]  9  9
[4,]  9  9

```

행렬과 벡터의 연산에서는 에러가 난다.

```

> add.matrix(a,1:8)
Error in add.matrix(a, 1:8) : 입력값중 행렬이 아닌 것이 있습니다.

```

차원이 다른 행렬의 연산에서도 에러가 난다.

```
> add.matrix(a, rbind(b, b))
```

Error in add.matrix(a, rbind(b, b)) : 두 행렬의 차원이 다릅니다.

### 3. 행렬의 곱

행렬의 곱은 행렬의 덧셈과 뺄셈과는 다르다.

일단 두 행렬에서 앞의 행렬의 열의 수와 뒤의 행렬의 행의 수가 같아야 된다.

R에서는 `%*%` 연산자를 사용한다.

3 by 2의 행렬을 만든다.

```
> a = matrix(1:6, nrow=3)
```

```
> a
```

```
  [,1] [,2]
```

```
[1,]  1  4
```

```
[2,]  2  5
```

```
[3,]  3  6
```

2 by 3의 행렬을 만든다.

```
> b = matrix(1:6, nrow=2)
```

```
> b
```

```
  [,1] [,2] [,3]
```

```
[1,]  1  3  5
```

```
[2,]  2  4  6
```

3 by 2 행렬과 2 by 3행렬은 곱은 3 by 3의 정방행렬이 된다.

```
> a %*% b
```

```
  [,1] [,2] [,3]
```

```
[1,]  9 19 29
```

```
[2,] 12 26 40
```

```
[3,] 15 33 51
```

2 by 3 행렬과 3 by 2행렬은 곱은 2 by 2의 정방행렬이 된다.

```
> b %*% a
```

```
  [,1] [,2]
```

```
[1,] 22 49
```

```
[2,] 28 64
```

2 by 3 행렬과 2 by 3행렬은 곱은 성립하지 않는다.

```
> b %*% b
```

```
Error in b %*% b : non-conformable arguments
```

원소의 갯수가 5인 열벡터를 만든다.

```
> x = matrix(1:5, ncol=1)
```

```
> x
```

```
  [,1]
```

```
[1,]  1
```

```
[2,]  2
```

```
[3,]  3
```

```
[4,]  4
```

```
[5,]  5
```

벡터의 외적을 구한다.

```
> x %*% t(x)
```

```
  [,1] [,2] [,3] [,4] [,5]
```

```
[1,]  1  2  3  4  5
```

```
[2,]  2  4  6  8 10
```

```
[3,]  3  6  9 12 15
```

```
[4,]  4  8 12 16 20
```

```
[5,]  5 10 15 20 25
```

벡터의 내적을 구한다.

```
> t(x) %*% x
```

```
  [,1]
```

```
[1,] 55
```

행렬의 곱을 구하는 prod.matrix라는 함수도 하나 만들어 보자

```
prod.matrix <- function (a, b)
```

```
{
```

```
  if (!is.matrix(a)) stop("입력값중 행렬이 아닌 것이 있습니다.")
```

```
  if (!is.matrix(b)) stop("입력값중 행렬이 아닌 것이 있습니다.")
```

```
  if(dim(a)[1]!=dim(b)[2]) stop("행렬의 곱이 성립하지 않습니다.")
```

```
  prd <- matrix(1:dim(a)[1]^2 , nrow=dim(a)[1])
```

```

for(i in 1:nrow(a)) {
  for(j in 1:nrow(a)) {
    tmp <- 0
    for(k in 1:ncol(a)) {
      tmp <- tmp + a[i,k] * b[k,j]
    }
    prd[i, j] <- tmp
  }
}
return (prd)
}

```

3 by 2 행렬과 2 by 3행렬은 곱은 3 by 3의 정방행렬이 된다.

```

> prod.matrix(a,b)
  [,1] [,2] [,3]
[1,]  9  19  29
[2,] 12  26  40
[3,] 15  33  51

```

2 by 3 행렬과 3 by 2행렬은 곱은 2 by 2의 정방행렬이 된다.

```

> prod.matrix(b,a)
  [,1] [,2]
[1,] 22  49
[2,] 28  64

```

2 by 3 행렬과 2 by 3행렬은 곱은 성립하지 않는다.

```

> prod.matrix(b,b)
Error in prod.matrix(b, b) : 행렬의 곱이 성립하지 않습니다.

```

벡터의 외적을 구한다.

```

> prod.matrix(x,t(x))
  [,1] [,2] [,3] [,4] [,5]
[1,]  1   2   3   4   5
[2,]  2   4   6   8  10
[3,]  3   6   9  12  15
[4,]  4   8  12  16  20
[5,]  5  10  15  20  25

```

벡터의 내적을 구한다.

```
> prod.matrix(t(x),x)
      [,1]
[1,] 55
```

#### 4. 대각(단위)행렬 만들기

정방행렬 중에 대각요소를 제외한 원소들이 0인 것을 대각행렬이라 한다. 또한 대각행렬 중에서 대각요소의 값이 모두 1인 것을 단위행렬이라 한다.

R에서는 `diag` 함수를 이용해서 대각행렬을 만든다.

대각요소가 1,2,3인 3by3 대각행렬을 만든다

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

3by3의 단위행렬을 만든다

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

정방행렬이 아닌 대각행렬도 가능하다.

```
> diag(1:3, ncol=4)
      [,1] [,2] [,3] [,4]
[1,]  1  0  0  0
[2,]  0  2  0  0
[3,]  0  0  3  0
```

다음과 같은 방법도 있다.

```
> m <- matrix(0,3,3)
> m
      [,1] [,2] [,3]
[1,]  0  0  0
```

```

[2,] 0 0 0
[3,] 0 0 0
> diag(m) <- 1:3
> m
  [,1] [,2] [,3]
[1,]  1  0  0
[2,]  0  2  0
[3,]  0  0  3

```

## 5. 삼각행렬 만들기

대각 원소 기준으로 아래가 모두 0인 행렬을 상삼각행렬, 위가 모두 0인 행렬을 하삼각행렬이라고 한다.

R에서는 `upper.tri`와 `lower.tri` 함수를 이용해서 대각행렬을 만들 수 있다.

정방행렬 `m`을 만들어 보자.

```

> m = matrix(1:9, nrow=3)
> m
  [,1] [,2] [,3]
[1,]  1  4  7
[2,]  2  5  8
[3,]  3  6  9

```

`upper.tri` 함수의 결과는 논리형의 행렬이다.

```

> upper.tri(m)
  [,1] [,2] [,3]
[1,] FALSE TRUE TRUE
[2,] FALSE FALSE TRUE
[3,] FALSE FALSE FALSE

```

`m`의 상삼각행렬을 만든다.

```

> upper.tri(m)*m
  [,1] [,2] [,3]
[1,]  0  4  7
[2,]  0  0  8
[3,]  0  0  0

```



lower.tri 함수의 결과도 논리형의 행렬이다.

```
> lower.tri(m)
      [,1] [,2] [,3]
[1,] FALSE FALSE FALSE
[2,]  TRUE FALSE FALSE
[3,]  TRUE  TRUE FALSE
```

m의 하삼각행렬을 만든다.

```
> lower.tri(m)*m
      [,1] [,2] [,3]
[1,]    0    0    0
[2,]    2    0    0
[3,]    3    6    0
```

## 6. 행렬식(Determinant) 구하기

행렬식을 구하는 함수는 det이다.

다음과 같은 2 by 2행렬에 대해

$$\begin{matrix} a & b \\ c & d \end{matrix}$$

m =  $\begin{matrix} a & b \\ c & d \end{matrix}$ 의 행렬식은  $a*d-b*c$ 이다. 3차원은 더 복잡하므로 생략한다.

```
> x <- matrix(1:4, ncol=2)
> x
      [,1] [,2]
[1,]    1    3
[2,]    2    4
```

```
> det(x)
[1] -2
```

```
> y <- matrix(1:9, ncol=3)
> y
      [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
```

```
> det(y)
```

```
[1] 0
```

단위행렬의 행렬식은 항상 1이다.

```
> det(diag(rep(1,3)))
```

```
[1] 1
```

## 7. 역행렬 구하기

정방행렬  $a$ 에 정방행렬  $b$ 를 곱한 것이 단위행렬이 된다면  $b$ 는  $a$ 의 역행렬이라 한다. R에서는 solve함수를 이용해서 구한다.

```
> x
```

```
  [,1] [,2]
```

```
[1,]  1  3
```

```
[2,]  2  4
```

$x$ 의 역행렬을 구한다.

```
> solve(x)
```

```
  [,1] [,2]
```

```
[1,] -2  1.5
```

```
[2,]  1 -0.5
```

$x$ 와  $x$ 의 역행렬의 곱은 단위행렬이 된다.

```
> x%%solve(x)
```

```
  [,1] [,2]
```

```
[1,]  1  0
```

```
[2,]  0  1
```

## 8. 행렬(벡터)의 외적 구하기

행렬  $m$ 의 전치행렬과 행렬  $m$ 의 곱을 행렬의 외적이라고 한다.

즉,  $(m) \%*\% m$ 이다.

```
> x
  [,1] [,2]
[1,]  1  3
[2,]  2  4
```

외적구하기

```
> t(x)%*%x
  [,1] [,2]
[1,]  5 11
[2,] 11 25
```

crossprod 함수로 외적 구하기

```
> crossprod(x)
  [,1] [,2]
[1,]  5 11
[2,] 11 25
```

```
> y <- matrix(1:4, ncol=2, byrow=T)
```

```
> y
  [,1] [,2]
[1,]  1  2
[2,]  3  4
```

행렬 x와 y의 외적

```
> crossprod(x,y)
  [,1] [,2]
[1,]  7 10
[2,] 15 22
```

행렬 x와 y의 외적

```
> t(x)%*%y
  [,1] [,2]
[1,]  7 10
[2,] 15 22
```

```
> x = 1:4
```

```
> y = 3:6
```

outer함수를 이용한 외적 구하기

```
> outer(x,y)
```

```
  [,1] [,2] [,3] [,4]  
[1,]  3   4   5   6  
[2,]  6   8  10  12  
[3,]  9  12  15  18  
[4,] 12  16  20  24
```

outer 연산자 %o%를 이용한 외적 구하기

```
> x%o%y
```

```
  [,1] [,2] [,3] [,4]  
[1,]  3   4   5   6  
[2,]  6   8  10  12  
[3,]  9  12  15  18  
[4,] 12  16  20  24
```