**Matrices and Arrays**

A ***matrix***is a vector with two additional attributes: the number of rows and the number of columns. Since matrices are vectors, they also have modes, such as numeric and character. (Vectors are ***not***one column or one-row matrices.)

Matrices are special cases of a more general R type of object: *arrays*. Arrays can be multidimensional. For example, a three-dimensional array would consist of rows, columns, and layers, not just rows and columns as in the matrix case. Each value of the matrix can be located by its row and column numbers.

**Creating Matrices**

The basic **R** command to define a matrix requires a list of elements **c(.,.,.,.,.,.,**) and the number of rows **nrow** in the matrix

Consider the matrix ****

To enter this matrix in **R**, we first have to write this as a single list, going down

each column, i.e., **c(1,2,3,4,5,6,7,8,9)** To use **R** to set the variable **C** equal to the matrix **C**, we would use

**> C <- matrix(c(1,2,3,4,5,6,7,8,9),nrow=3)**

**> C**

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

[1,] 1 4 7

[2,] 2 5 8

[3,] 3 6 9

**R** uses the **nrow** command to set the dimension of the matrix. For example, if we enter

**> C <- matrix(c(1,2,3,4,5,6,7,8,9),nrow=1)**

This set **C** equal to the matrix with a single row

**C**= (1 2 3 4 5 6 7 8 9)

By typing **C** and hitting return, **R** displays the matrix **C**.

Conversely, you can instruct **R** to enter rows first by adding the command **byrow=T**, which enters the elements of the list as rows (the default is setting this option to false, entering this as columns). Thus entering

> D <- matrix(c(1,2,3,4,5,6,7,8,9),nrow=3,byrow=T)

> D

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

[1,] 1 2 3

[2,] 4 5 6

[3,] 7 8 9

Individual elements can be extracted from a matrix **D** by using command **D[i,j]**, which extracts the element in the *i*th row and *j*th column of **D**.

For example

**> D[2,3]**

**[1] 6**

One can specify the number of rows and number of columns as below

**> y <- matrix(c(1,2,3,4),nrow=2,ncol=2)**

**> y**

**[,1] [,2]**

**[1,] 1 3**

**[2,] 2 4**

Note that when we then print out y, R shows us its notation for rows and columns. For instance, [,2] means the entirety of column 2, as can be seen in this check:

**> y[,2]**

**[1] 3 4**

Another way to build y is to specify elements individually:

> y <- matrix(nrow=2,ncol=2)

> y[1,1] <- 1

> y[2,1] <- 2

> y[1,2] <- 3

> y[2,2] <- 4

> y

[,1] [,2]

[1,] 1 3

[2,] 2 4

**Tip:** The dimension of a matrix can be checked using ***dim()***or ***attributes()***

We can add optional character ***“dimnames”*** giving the row and column names respectively, list names will be used as names for the dimensions

> x <- matrix(c(5,0,6,1,3,5,9,5,7,1,5,3), nrow=3, ncol=4, byrow=TRUE, dimnames=list(rows=c("r.1", "r.2", "r.3"), cols=c("c.1", "c.2", "c.3", "c.4")))

> x

cols

rows c.1 c.2 c.3 c.4

r.1 5 0 6 1

r.2 3 5 9 5

r.3 7 1 5 3

Matrices have numbers naming their rows and columns. Consider a 4×5 matrix of random integers from a Poisson distribution with mean = 1.5

>X<-matrix(rpois(20,1.5),nrow=4)

X

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

[1,] 1 0 2 5 3

[2,] 1 1 3 1 3

[3,] 3 1 0 2 2

[4,] 1 0 2 1 0

**Suppose that the rows refer to four different trials and we want to label the rows ‘Trial.1’, ‘Trial.2” etc. We employ the function rownames to do this.**

>X<-matrix(rpois(20,1.5),nrow=4)

> rownames(X)<-rownames(X,do.NULL=FALSE,prefix="Trial.")

> X

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

Trial.1 5 1 1 3 1

Trial.2 2 1 2 1 2

Trial.3 1 1 2 2 1

Trial.4 1 0 2 0 3

**For the columns we want to supply a vector of different names for the five drugs involved in the trial, and use this to specify the colnames(X):**

> drug.names<-c("aspirin", "paracetamol", "nurofen", "hedex", "placebo")

> colnames(X)<-drug.names

> X

aspirin paracetamol nurofen hedex placebo

Trial.1 5 1 1 3 1

Trial.2 2 1 2 1 2

Trial.3 1 1 2 2 1

Trial.4 1 0 2 0 3

**One more Example**

> x <- matrix(c(5,0,6,1,3,5,9,5,7,1,5,3), nrow=3, ncol=4, byrow=TRUE)

> x

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

[1,] 5 0 6 1

[2,] 3 5 9 5

[3,] 7 1 5 3

> x[2,3] # Row 2, Column 3

[1] 9

> x[1,] # Row 1

[1] 5 0 6 1

> x[,2] # Column 2

[1] 0 5 1

> x[c(1,3),] # Rows 1 and 3, all Columns

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

[1,] 5 0 6 1

[2,] 7 1 5 3

> x[3,] # Row 3 in the form of a vector

[1] 7 1 5 3

> x[3,,drop=F] # Row 3 in the form of a matrix

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

[1,] 7 1 5 3

> x[-1,] # matrix x without its first row

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

[1,] 3 5 9 5

[2,] 7 1 5 3

> x[1:2, -1]#two first rows of matrix x without its first column

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

[1,] 0 6 1

[2,] 5 9 5

**Choose columns of matrix x in which the value in 1st line is greater than 4**

>x[,x[1,]>4]

[,1] [,2]

[1,] 5 6

[2,] 3 9

[3,] 7 5

**Construct a vector containing the values from matrix x which are greater than 4**

>x[x>4]

[1] 5 7 5 6 9 5 5

**Replace the values of x which are greater than 3 with NA**

> x

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

[1,] 5 0 6 1

[2,] 3 5 9 5

[3,] 7 1 5 3

> x[x>3]<-NA

> x

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

[1,] NA 0 NA 1

[2,] 3 NA NA NA

[3,] NA 1 NA 3

**Matrix Operations**

When matrices are used in math expressions the operations are performed element by element.

We could use subscripts to select parts of the matrix, with a blank meaning ‘all of the rows’

or ‘all of the columns’

Example: Suppose we have a matrix

> X

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

[1,] 1 0 2 5 3

[2,] 1 1 3 1 3

[3,] 3 1 0 2 2

[4,] 1 0 2 1 0

Therefore,

> mean(X[,5])

[1] 2

>var(X[4,])

[1] 0.7

>rowSums(X) # Note the uppercase S

[1] 11 9 8 4

>colSums(X)

[1] 6 2 7 9 8

>rowMeans(X)# Note the uppercase M

[1] 2.2 1.8 1.6 0.8

>colMeans(X)

[1] 1.50 0.50 1.75 2.25 2.00

The ***apply*** function is used for applying functions to the rows or columns of matrices or

dataframes. In a matrix margin 1 refers to rows and margin 2 refers to the column.

> X=matrix(1:24, nrow=4)

> X

[,1] [,2] [,3] [,4] [,5] [,6]

[1,] 1 5 9 13 17 21

[2,] 2 6 10 14 18 22

[3,] 3 7 11 15 19 23

[4,] 4 8 12 16 20 24

> rowSums(X)

[1] 66 72 78 84

> colSums(X)

[1] 10 26 42 58 74 90

> apply(X,1,sum)

[1] 66 72 78 84

> apply(X,2, sum)

[1] 10 26 42 58 74 90

We can shuffle the elements of each columns of a matrix independently.

> apply(X,2,sample)

[,1] [,2] [,3] [,4] [,5] [,6]

[1,] 4 7 9 13 18 21

[2,] 1 5 10 16 19 22

[3,] 3 6 12 14 20 24

[4,] 2 8 11 15 17 23

Similarly we can shuffle the elements of each row as well.

Matrix Functions in R

|  |  |
| --- | --- |
| Function | Description |
| A+B | Addition of matrices |
| A-B | Subtraction of matrices |
| A%\*%B | Product of matrices |
| t(A) | Transposition of a matrix |
| diag(5) | Identity matrix of order 5 |
| diag(A) | Vector with the values of the diagonal elements |
| crossprod(A,B) | Cross product (t(A)%\*%B) |
| det(A) | Determinant of matrix A |
| svd(A) | Singular value decomposition |
| eigen(A) | Matrix diagonalisation |
| solve(A) | Matrix inversion |
| solve(A,b) | Solving linear systems |
| chol(A) | Cholesky decomposition |
| qr(A) | QR decomposition |

**Examples:**

> C <- matrix(c(1,2,3,4,5,6,7,8,9),nrow=3)

> D <- matrix(c(1,2,3,4,5,6,7,8,9),nrow=3,byrow=T)

> C

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

[1,] 1 4 7

[2,] 2 5 8

[3,] 3 6 9

> D

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

[1,] 1 2 3

[2,] 4 5 6

[3,] 7 8 9

> C+D

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

[1,] 2 6 10

[2,] 6 10 14

[3,] 10 14 18

> C-D

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

[1,] 0 2 4

[2,] -2 0 2

[3,] -4 -2 0

**Multiplication by a scalar**

**> y<-matrix(c(1,2,3,4,5,6), nrow=3)**

**> y**

**[,1] [,2]**

**[1,] 1 4**

**[2,] 2 5**

**[3,] 3 6**

**> 3\*y**

**[,1] [,2]**

**[1,] 3 12**

**[2,] 6 15**

**[3,] 9 18**

**Matrix Multiplication**

> C <- matrix(c(1,2,3,4,5,6,7,8,9),nrow=3)

> D <- matrix(c(1,2,3,4,5,6,7,8,9),nrow=3,byrow=T)

> C%\*%D

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

[1,] 66 78 90

[2,] 78 93 108

[3,] 90 108 126

Similarly,

> D%\*%C

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

[1,] 14 32 50

[2,] 32 77 122

[3,] 50 122 194

**Example:** Consider the vector. Use **R** to compute  and .

> A <- matrix(c(1,2,3),nrow=3)

> AT<-t(A)

> A

[,1]

[1,] 1

[2,] 2

[3,] 3

> AT

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

[1,] 1 2 3

> A%\*%AT

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

[1,] 1 2 3

[2,] 2 4 6

[3,] 3 6 9

> AT%\*%A

[,1]

[1,] 14

Note: ***col()*** returns the column number of its argument, and row() returns the row number

> y<-matrix(c(1,2,3,4,5,6), nrow=3)

> y

[,1] [,2]

[1,] 1 4

[2,] 2 5

[3,] 3 6

> row(y)

[,1] [,2]

[1,] 1 1

[2,] 2 2

[3,] 3 3

> col(y)

[,1] [,2]

[1,] 1 2

[2,] 1 2

[3,] 1 2

**Concatenation**

By column using the function ***cbind***

By row using the function ***rbind***

> A<-matrix(c(1,4,5,6,7,8, 6,8),nrow=2)

> A

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

[1,] 1 5 7 6

[2,] 4 6 8 8

> B<-matrix(1:12,nrow=3)

> B

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

[1,] 1 4 7 10

[2,] 2 5 8 11

[3,] 3 6 9 12

> rbind(A,B)

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

[1,] 1 5 7 6

[2,] 4 6 8 8

[3,] 1 4 7 10

[4,] 2 5 8 11

[5,] 3 6 9 12

**The Inverse of a Matrix**

The inverse of A is obtained using the solve command, **solve (A).**

Example: Consider a system of equations



> A <- matrix(c(3,1,4,6),nrow=2)

> y <- matrix(c(4,2),nrow=2)

> x<- solve(A)%\*%y

> x

[,1]

[1,] 1.1428571

[2,] 0.1428571

We can check this by looking at the first equation**,**

> 3\*x[1,1]+4\*x[2,1]

[1] 4

**Arrays**

An array is a multi-dimensional generalization of a vector. To create an array we use the following syntax

***array(data = NA, dim = length(data), dimnames = NULL)***

Note that

* Values are entered by columns
* Like with vectors and matrices, when arrays are used in math expressions

the operations are performed element by element.

* Also like vectors and matrices, the elements of an array must all be of the

same type (numeric, character, logical, etc.)

**Example**

Sample 2 x 3 x 2 array:

> w <- array(1:12, dim=c(2,3,2), dimnames=list(c("A","B"), c("X","Y","Z"), c("N","M")))

> w

, , N

X Y Z

A 1 3 5

B 2 4 6

, , M

X Y Z

A 7 9 11

B 8 10 12

**Useful Array Functions**

|  |  |
| --- | --- |
| Function | Description |
| apply() | Apply a function to the margins of an array |
| aperm() | Transpose an array by permuting its dimensions |
| dim(x) | Dimensions of x |
| dimnames(x) | Get or set the dimension names of x |

We can use the ***apply()*** function for more then one dimension.

For a 3-dimensional array there are now three margins to apply the function

to: 1=rows, 2=columns, and 3=matrices.

> X <- array(1:12, dim=c(2,3,2),dimnames=list(c("A","B"), c("X","Y","Z"), c("N","M")))

> X

, , N

X Y Z

A 1 3 5

B 2 4 6

, , M

X Y Z

A 7 9 11

B 8 10 12

> apply(X, 2, sum)# column sums

X Y Z

18 26 34

> apply(X, 1, sum)# row sums

A B

36 42

> apply(X, c(1,3), sum)# row and matrix sums

N M

A 9 27

B 12 30

**Example: aperm()**

> X <- array(1:12, dim=c(2,3,2),dimnames=list(c("A","B"), c("X","Y","Z"), c("N","M")))

> X

, , N

X Y Z

A 1 3 5

B 2 4 6

, , M

X Y Z

A 7 9 11

B 8 10 12

> dim(X)

[1] 2 3 2

> aperm(X)

, , A

X Y Z

N 1 3 5

M 7 9 11

, , B

X Y Z

N 2 4 6

M 8 10 12

**Lists**

A list is a general form of a vector, where the elements don't need to be of the same type or dimension.

* The function list(...) creates a list of the arguments
* Arguments have the form name=value. Arguments can be specified with and without names.

**Example**

> y <- list(num=c(3,4,5), "Jack", identity=diag(2))

> y

$num

[1] 3 4 5

[[2]]

[1] "Jack"

$identity

[,1] [,2]

[1,] 1 0

[2,] 0 1

**Example**

> j <- list(name="Jack", salary=55000, union=T)

> j

$name

[1] "Jack"

$salary

[1] 55000

$union

[1] TRUE

Names of list components can be abbreviated to whatever extent is possible without causing ambiguity:

> j$s

[1] 55000

Elements of a list can be referenced using [ ] as well as [[ ]] or $.

> x <- list(num=c(1,2,3), "Nick", identity=diag(2))

> x[[2]] # Second element of x

[1] "Nick"

> x[["num"]] # Element named "num"

[1] 1 2 3

> x$identity # Element named "identity"

[,1] [,2]

[1,] 1 0

[2,] 0 1

> x[[3]][1,] # First row of the third element

[1] 1 0

> x[1:2] # Create a sublist of the first two elements

$num

[1] 1 2 3

[[2]]

[1] "Nick"

**Useful List Functions**

|  |  |
| --- | --- |
| Function | Description |
| lapply() | Apply a function to each element of a list, returns a list |
| sapply() | Same as lapply(), but returns a vector or matrix by default |
| vapply() | Similiar to sapply(), but has a pre-specied type of return value |
| replicate() | Repeated evaluation of an expression, useful for replicating lists |
| unlist(x) | Produce a vector of all the components that occur in x |
| length(x) | Number of objects in x |
| names(x) | Names of objects in x |

**Data Frames**

A *data frame* is like a matrix, with a two-dimensional rows and columns structure. However, it differs from a matrix in that each column may have a different mode. For instance, one column may consist of numbers, and another column might have character strings. A data frame is the fundamental data structure used by R 's statistical modeling functions In this sense, just as lists are the heterogeneous analogs of vectors in one dimension, data frames are the heterogeneous analogs of matrices for two-dimensional data.

**Creating Data Frames**

> names<-c("Joe", "Peter","William")

> age<-c(45,78,60)

> data<-data.frame(names,age)

> data

names age

1 Joe 45

2 Peter 78

3 William 60

**Accessing Data Frames**

> names<-c("Joe", "Peter","William")

> age<-c(45,78,60)

> data<-data.frame(names,age)

> data

names age

1 Joe 45

2 Peter 78

3 William 60

> data[2,2] # extracting the information of the data

[1] 78

> data[[1]]

[1] Joe Peter William

Levels: Joe Peter William

> data[[2]]

[1] 45 78 60

> data[1]

names

1 Joe

2 Peter

3 William

> data[2]

age

1 45

2 78

3 60

**Data Frame Restrictions**

* Components must be vectors (numeric, character, logical) or factors.
* All vectors and factors must have the same lengths.

**Expanding Data Frames**

Components can be added easily to a data frame in the natural way.

> data$Major<-c("Math", "Biology", "Statistics")

> data

names age Major

1 Joe 45 Math

2 Peter 78 Biology

3 William 60 Statistics

We can add more rows using ***‘rbind***” and columns using ***cbind***

Suppose we want to add two more entries

Jim 23 Physics

Lucas 54 Math

So we create a new data frame

> new=data.frame(names=c("Jim", "Lucas"), age=c(23,54), Major=c("Physics","Math"))

> new

names age Major

1 Jim 23 Physics

2 Lucas 54 Math

The combined data is as below

> newdata<-rbind(data,new)

names age Major

1 Joe 45 Math

2 Peter 78 Biology

3 William 60 Statistics

4 Jim 23 Physics

5 Lucas 54 Math

**Select Rows Based Variable Values**

Often data are better viewed when sorted. The function order sorts a column and gives output that can sort the rows of a data.frame.

> databyage<-newdata[order(newdata$age),]

> databyage

names age Major

4 Jim 23 Physics

1 Joe 45 Math

5 Lucas 54 Math

3 William 60 Statistics

2 Peter 78 Biology

**Reverse order**

> databyage<-data[rev(order(data$age)),]

> databyage

names age Major

2 Peter 78 Biology

3 William 60 Statistics

5 Lucas 54 Math

1 Joe 45 Math

4 Jim 23 Physics

**Eliminating Duplicate Rows from a Dataframe**

Suppose a data set ***students*** has several observations with multiple variables. If some data are replicated with same information then we can use ***unique(students)*** in order to create a dataframe with all the duplicate rows stripped out