Computational Statistics in R Lab

Computer Science PG Study Material
Paper: CS-292 Module - II

Part-II: Vectors, Matrices, Statistical Operations

Anupam Pattanayak[1]
Assistant Professor,
Department of Computer Science,
Raja N. L. Khan Women's College (Autonomous),
Midnapore, West Bengal

April 21, 2020

[1]anupam.pk@gmail.com

Contents

1

R - Working with Vectors

In the previous lecture note, we have seen usage of some basic commands and doing input-output in R. We will now dee how to work with vectors. Vectors are mathematical objects, which for simplicity we can consider as arrays which we are very much familiar with.

we will refer the book by Cotton[1], and the book by Matloff[2].

1.1 Initializing Vectors

If a vector contains arbitrary elements 2, 3, 5, 11, 12 then we can initialize the vector in the following way:

```
> x<-c(2,3,5,11,12)
> x
[1]  2  3  5 11 12
>
```

If the vector elements has some pattern or they follow a particular sequence like 2,3,4,5,6,7,8 theen we can also create the vector like given below:

```
> y<-2:8
> y
[1] 2 3 4 5 6 7 8
>
```

We can also create the vector with above sequence as given below:

[1]Learning R - A Step by Step Functional Guide by Richard Cotton, Orielly

[2]The Art of R Programming- A Tour by Norman Matloff, No Starch Press

```
 x<-seq.int(2,8)
> x
[1] 2 3 4 5 6 7 8
>
```

We can also create the above vector as follows:

```
> x2<-(2:16)
> x2
 [1]  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16
>
```

If we want a vector whose elelments has the sequence 0,2,4,6,8,10,12,14,16, then we can use *seq.int( )* in the following way.

```
> x1<-seq.int(0,16,2)
> x1
[1]  0  2  4  6  8 10 12 14 16
>
```

Unlike C programming where array index starts from 0, here vector index starts from 1. To extract first element of vector x and initialize it to a variable a, we do the following.

```
> a<-x[1]
> a
[1] 2
>
```

Let us suppose we want to create a vector with the elements 1,4,9,16,25,36 and then extract its 1^{st}, 3^{rd}, and 5^{th} elements and store it into another vector. This we can achieve by the following.

```
> x<-(1:6)^2
> x
[1]  1  4  9 16 25 36
> y<-x[c(1,3,5)]
> y
[1]  1  9 25
>
```

If we want a vector like 1,2,3,4,5,6,1,2,3,4,5,6 that is 1:6 repeated twice, we use the following command.

```
> x<-rep(1:6,2)
> x
 [1] 1 2 3 4 5 6 1 2 3 4 5 6
>
```

If we want to create a vector 1,1,2,2,3,3,4,4,5,5 that is 1:5 with each element repeated twice, we use the *rep( )* command in the following way.

```
> x<-rep(1:5,each=2)
> x
 [1] 1 1 2 2 3 3 4 4 5 5
>
```

If we want to create a vector 1,2,2,3,3,3,4,4,4,4 then we use the *rep( )* command in the following way.

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

If we want to create a vector like 1,2,3,4,5,1,2,3 then we use the *rep( )* command in the following way.

```
> x<-rep(1:5,length.out=8)
> x
[1] 1 2 3 4 5 1 2 3
>
```

1.2 Basic Operations on Vectors

We can do basic operations on vectors like addition, subtraction, multiplication etc. We illustrate these in the following.

To add 1 to each element of vector x, we use the command, as our intuition suggests, as follows.

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

Suppose, we have a vector 1:5 and another vector 11:20. So clearly the vectors are of different length. But R recycles the shorter length vector elements to match the larger-sized vector and then performs addition. But the only condition is that *longer object length has to be multiple of shorter object length*. Otherwise, an error message like *longer object length is not a multiple of shorter object length* would appear. This is illustrated below.

```
> x=1:5
> y=11:20
> x
[1] 1 2 3 4 5
> y
 [1] 11 12 13 14 15 16 17 18 19 20
> x+y
 [1] 12 14 16 18 20 17 19 21 23 25
>
```

Similarly, we can do the subtraction on vectors. For example, see the following:

```
> x
[1] 1 2 3 4 5
> x-1
[1] 0 1 2 3 4
> y
 [1] 11 12 13 14 15 16 17 18 19 20
> y-x
 [1] 10 10 10 10 10 15 15 15 15 15
>
```

We can do the element-wise product of two vectors as shown below.

```
> x
[1] 1 2 3 4 5
> y=11:15
> y
[1] 11 12 13 14 15
> x*y
[1] 11 24 39 56 75
>
```

Similarly, we can do the element-wise division as follows.

```
> y
[1] 11 12 13 14 15
> x
[1] 1 2 3 4 5
> y/x
[1] 11.000000  6.000000  4.333333  3.500000  3.000000
>
```

earlier, we have seen the usage of exponent operator, ^ Let us use it once more, as given below.

```
> y
[1] 11 12 13 14 15
> y^2
[1] 121 144 169 196 225
>
```

To know the size of a vector, we use the *NROW( )* command.

```
> NROW(x)
[1] 5
>
```

With thse, we conclude the chapter on vector. We will use the vectors again when we apply statistical operations on the vectors. You are highly encouraged the given e-books to explore further.

2

R - Working with Matrices

In this chapter we will learn how to create matrix with specific dimension, reshaping the matrix dimension, basic matrix operations such as addition and multiplication, determining matrix determinant, matrix transpose, matrix inverse, creating identity matrix, finding Eigen values and Eigen vectors.

2.1 Creation of a Matrix

To create a 4×3 matrix like $\begin{pmatrix} 1 & 5 & 9 \\ 2 & 6 & 10 \\ 3 & 7 & 11 \\ 4 & 8 & 12 \end{pmatrix}$ we use the *matrix( )* command as follows:

```
> m1<-matrix(1:12,nrow=4)
> m1
     [,1] [,2] [,3]
[1,]    1    5    9
[2,]    2    6   10
[3,]    3    7   11
[4,]    4    8   12
>
```

Observe the output. Note that, matrix has been created column-wise with our supplied values 1:12. To create the matrix row-wise instead of column-wise, we use the following command:

```
> m2<-matrix(1:12,nrow=4,byrow=TRUE)
> m2
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
[4,]   10   11   12
>
```

To know that *m2* is a matrix, we can use *class(m2)* command. To know the *dimension* of this created matrix, we can use *dim(m1)* command. These two commands usage are shown below:

```
> class(m2)
[1] "matrix"
> dim(m2)
[1] 4 3
>
```

2.2 Matrix Addition and Subtraction

To add two matrices, we use the following.

```
> m1
     [,1] [,2] [,3]
[1,]    1    5    9
[2,]    2    6   10
[3,]    3    7   11
[4,]    4    8   12
> m2
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
[4,]   10   11   12
> m1+m2
     [,1] [,2] [,3]
[1,]    2    7   12
[2,]    6   11   16
[3,]   10   15   20
[4,]   14   19   24
>
```

Similarly, we can do matrix subtraction as given below.

```
> m1-m2
     [,1] [,2] [,3]
[1,]    0    3    6
[2,]   -2    1    4
[3,]   -4   -1    2
[4,]   -6   -3    0
>
```

2.3 Matrix Multiplication

If we use ∗to multiply matrices, then we will get element-wise product. To multiply two matrices, we need to use %∗%. However, both the matrices that we have created *m1* and *m2* are of order 4×3 and are not compatible for multiplication. So we first *reshape* one matrix to be of order 3×4 and then do the multiplication as illustrated below.

```
> dim(m1)
[1] 4 3
> dim(m2)
[1] 4 3
> dim(m2) <- c(3,4)
> m2
     [,1] [,2] [,3] [,4]
[1,]    1   10    8    6
[2,]    4    2   11    9
[3,]    7    5    3   12
> m1 % * % m2
     [,1] [,2] [,3] [,4]
[1,]   84   65   90  159
[2,]   96   82  112  186
[3,]  108   99  134  213
[4,]  120  116  156  240
>
```

Please note that, there wouln't be any spaces around ∗to execute the command, but I had to put it to avoid some display issue.

2.4 Matrix Transpose

To obtain transpose of a matrix, we use the command . For example, to transpose the matrix *m1*, we give the command as given below.

```
> t(m1)
     [,1] [,2] [,3] [,4]
[1,]    1    2    3    4
[2,]    5    6    7    8
[3,]    9   10   11   12
>
```

2.5 Matrix Determinant

We use *det( )* to obtain *determinant* of a matrix. In the following, we first create a square matrix whose determinant is 0, and then another square matrix and obtain it's determinant.

```
> m1<-matrix(1:9,nrow=3)
> det(m1)
[1] 0
> m1<-matrix((1:9)^2,nrow=3)
> m1
     [,1] [,2] [,3]
[1,]    1   16   49
[2,]    4   25   64
[3,]    9   36   81
> det(m1)
[1] -216
>
```

2.6 Identity Matrix Creation

To create an identity matrix, say of order 4, we use the command *diag( )*.

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

2.7 Matrix Inverse

To find the inverse of a non-singular matrix, we use the command *solve( )*. To be sure that it's indeed the matrix inverse, we can multiply the matrix and it's inverse and check if we obtain identity matrix or not. In some cases, we may get values very very close to 1 or 0 when we do such multiplication.

```
> solve(m1)
           [,1]       [,2]       [,3]
[1,]  1.291667 -2.166667  0.9305556
[2,] -1.166667  1.666667 -0.6111111
[3,]  0.375000 -0.500000  0.1805556
>
```

2.8 Eigen Values and Eigen Vector

Eigen value and Eigen vectors are very important which has been used in many applications in science and engineering. We use the command *eigen( )* to obtain both eigen values and eigen vectors.

```
> eigen(m1)
$values
[1] 112.9839325  -6.2879696   0.3040371

$vectors
           [,1]       [,2]       [,3]
[1,] -0.3993327 -0.8494260  0.7612507
[2,] -0.5511074 -0.4511993 -0.6195403
[3,] -0.7326760  0.2736690  0.1914866

>
```

3

R - Statistical Commands

Here, we will some basic statistical commands to compute minimum, maximum, cumulative min, cumulative sum, cumulative product, mean, median, variance, and standard deviation.

3.1 Minimum and Maximum

We use the commands *min( )* and *max( )* to find minimum and maximum respectively.

```
> x<-c(2,3,2,4,5,4,6,4)
> min(x)
[1] 2
> max(x)
[1] 6
>
```

3.2 Cumulative Sum, Cumulative Product, Cumulative Min

Cumulative means what we have seen so far as we continue scanning the elements. We use the commands *cumsum( )*, *cumprod( )*, and *cummin( )* to find cumulative minimum, cumulative product, and cumulative minimum respectively.

```
> x
[1] 2 3 2 4 5 4 6 4
> cumsum(x)
[1]  2  5  7 11 16 20 26 30
> cumprod(x)
[1]    2    6   12   48  240  960 5760 23040
> cummin(x)
[1] 2 2 2 2 2 2 2 2
>
```

3.3 Mean, Median

Mean and median are two important statistical parameter to examine a given population. Both give a single quantity to represent the given population. Mean finds the average of the population. But mean can skew towards a very extreme value present in the population. To get rid of this one or two extreme values present in the population, median is used. We use the commands *mean( )*, and *median( )* to find mean and median respectively.

```
> x
[1] 2 3 2 4 5 4 6 4
> mean(x)
[1] 3.75
> median(x)
[1] 4
>
```

3.4 Variance, Covariance, Standard Deviation

In simpler words, variance is a single quantity that measures the average variation of each element in the population from their mean. Because the differences may get cancelled due to values with equal magnitude but opposite sign, the differences are squared. The command *var( )* is used to compute variance. Similarly, covariance is a measure of the joint variability of two populations. The command *cov( )* is used to compute covariance. When the two arguments of *cov( )* is same then it basically returns *variance*. Standard deviation also measures the amount of variation of the population. It is the

square root of variance. This is more commonly used term than variance. The command *sd( )* is used to compute standard deviation.

```
> x
[1] 2 3 2 4 5 4 6 4
> var(x)
[1] 1.928571
> y<-c(10,11,5,7,2,3,8,1)
> cov(x,y)
[1] -1.607143
> cov(x,y=x)
[1] 1.928571
> sd(x)
[1] 1.38873
>
```

With this, we end this chapter. There are lot more statistical functions in R that you can further explore consulting help pages and refering the books suggested.