Matrices

- Matrices are vectors with a dimension attribute. The dimension attribute is itself an integer vector of length 2 (nrow, ncol)
- All columns in a matrix must have the same mode (numeric, character, etc.) and the same length.
- The general format is:

```
> m <- matrix(nrow = 2, ncol = 3)
> m
[,1] [,2] [,3]
[1,] NA NA NA
[2,] NA NA NA
```

Matrices

- Matrices are constructed column-wise, so entries can be thought of starting in the “upper left" corner and running down the columns.
- **byrow=TRUE** indicates that the matrix should be filled by rows.
- **byrow=FALSE** indicates that the matrix should be filled by columns (the default).
- **dimnames** provides optional labels for the columns and rows.

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

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

> rownames(a)<-c("A","B","C")
> a
  [,1] [,2] [,3] [,4]
A    1    4    7   10
B    2    5    8   11
C    3    6    9   12

> colnames(a)<-c("1","2","x","y")
> a
  1 2 x  y
A 1 4 7 10
B 2 5 8 11
C 3 6 9 12
```

Matrices: looking at properties

- To look at the structure of an object using the **`str()`** function. It looks similar to the output for a vector, with the difference that R gives the indices for the rows and for the columns.
- To look at the number of rows and columns without looking at the structure, use the **`dim()`**or the **`attributes()`** functions.
- To find the total number of values in a matrix use the **`length()`** function.

```
➢ m<-matrix(1:12, ncol=4, byrow=TRUE)

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

> str(m)
 int [1:3, 1:4] 1 5 9 2 6 10 3 7 11
4 ...

 > dim(m)
 [1] 3 4

 > attributes(m)
 $dim
 [1] 3 4
 > length(m)
 [1] 12
```

Matrices: Combining vectors into a matrix

- Matrices can be created by column-binding or row-binding with cbind() and rbind().The rows take the names of the original vectors.

```
> objects <- 1:3
> counts <- 10:12

> cbind(objects, counts)
 objects counts
[1,]       1     10
[2,]       2     11
[3,]       3     12

>rbind(objects, counts)
        [,1] [,2] [,3]
objects    1    2    3
counts    10   11   12
```

Indexing Matrices

Indexing allows to directly extract elements

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

```
> # Indexing a matrix
> a[1,1]
[1] 1

# First row
>a[1,]
 1  2  3  4
 # First column
>a[,1]
 1  5 9
```

Indexing Matrices

Extract the values of the last two columns for the first two rows

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

```
> a[1:2,3:4]
     [,1] [,2]
[1,]    3    4
[2,]    7    8
```

R returns you a matrix again. **Pay attention:** the indices of this new matrix are not the indices of the original matrix anymore.

Dropping values

Get all the values except the second row and the third column

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

```
> a[-2,-3]
     [,1] [,2] [,3]
[1,]    1    2    4
[2,]    9   10   12
```

R returns you a matrix again. **Pay attention:** the indices of this new matrix are not the indices of the original matrix anymore.

R and dimensions

Get all the values except the second row and the third column

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

```
> a[-c(1, 3), ]
```

```
> a[-c(1, 3), ]

[1] 5 6 7 8
```

By default, R always tries to simplify the objects to the smallest number of dimensions possible!!!. So, if you extract from a matrix only one column or row, R will make a vector

R and dimensions

To force R to keep all dimensions use the extra argument `drop` from the indexing function.

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

```
> a[2, , drop=FALSE]
     [,1] [,2] [,3] [,4]
[1,]    5    6    7    8
```

First position: row index
Second position: column index
Third position: argument drop

Replacing values in a Matrix

- Change a value
- Change a entire row or column of values by not specifying the other dimension. Note that values are recycled
- Replace a subset of values

```
#Change a subset
> a[1:2, 3:4] <- c(8,4,2,1)
> a
     [,1] [,2] [,3] [,4]
[1,]    1    2    8    2
[2,]    1    3    4    1
[3,]    9    4   11   12
```

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

# Change a value
> a[3, 2] <- 4
> a
     [,1] [,2] [,3] [,4]
[1,]    1    2    3    4
[2,]    5    6    7    8
[3,]    9    4   11   12

# Change a row
> a[2, ] <- c(1,3)
> a
     [,1] [,2] [,3] [,4]
[1,]    1    2    3    4
[2,]    1    3    1    3
[3,]    9    4   11   12
```

Transposing a Matrix

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

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

Inverting a Matrix

```
> square.matrix <- matrix(c(1,0,3,2,2,4,3,2,1),ncol=3)

> square.matrix
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    0    2    2
[3,]    3    4    1

> square.matrix^(-1)
          [,1] [,2]      [,3]
[1,] 1.0000000 0.50 0.3333333
[2,]       Inf 0.50 0.5000000
[3,] 0.3333333 0.25 1.0000000

> solve(square.matrix)
     [,1]       [,2]       [,3]
[1,]  0.5 -0.8333333  0.1666667
[2,] -0.5  0.6666667  0.1666667
[3,]  0.5 -0.1666667 -0.1666667
```

R applies the arithmetic operators element-wise on the matrix. So, the command `square.matrix^(-1)` does not give the inverse of the matrix but the inverse of the elements

Funciones para trabajar con matrices

Función	Utilidad
ncol(x)	Número de columnas de x.
nrow(x)	Número de filas de x.
t(x)	Transpuesta de x
cbind(…)	Combina secuencias de vectores/matrices por col's.
rbind(…)	Combina secuencias de vectores/matrices por filas.
diag(x)	Extrae diagonal de matriz o crea matriz diagonal.
col(x)	Crea una matriz con elemento ij igual al valor j
row(x)	Crea una matriz con elemento ij igual al valor i
apply(x,margin,FUN,)	Aplica la función FUN a la dimensión especificada en margin 1 indica filas, 2 indica columnas. NB.
outer(x,y,fun="*") otra forma x %o %y	Para dos vectores x e y , crea una matriz A[i,j]=FUN(x[i],y[j] Por defecto crea el producto externo.

*Tabla de David Conesa, GE**e**i **tE**ma (UV)*

Arrays in R

- Arrays are the R data objects which can store data in more than two dimensions.
- An array is created using the **array()** function. It takes vectors as input and uses the values in the **dim** parameter to create an array.

```
> my.array <- array(1:24, dim=c(3,4,2))
> my.array
, , 1
  [,1] [,2] [,3] [,4]
[1,] 1 4 7 10
[2,] 2 5 8 11
[3,] 3 6 9 12
, , 2
  [,1] [,2] [,3] [,4]
[1,] 13 16 19 22
[2,] 14 17 20 23
[3,] 15 18 21 24
```

Naming columns and rows in an Array

We can give names to the rows, columns and matrices in the array by using the **dimnames** parameter.

```
# Create two vectors of different lengths.

vector1 <- c(5,9,3)
vector2 <- c(10,11,12,13,14,15)

column.names <- c("COL1","COL2","COL3")
row.names <- c("ROW1","ROW2","ROW3")
matrix.names <- c("Matrix1","Matrix2")

# Take these vectors as input to the array.
result <- array(
            c(vector1,vector2),
            dim = c(3,3,2),
            dimnames = list(row.names,column.names, matrix.names)
          )
```

Assesing Array elements

```
# Print the third row of the second matrix of the array.
 print(result[3,,2])

# Print the element in the 1st row and 3rd column of the 1st
matrix.

 print(result[1,3,1])

 # Print the 2nd Matrix.

print(result[,,2])
```

Factors a special vector to work with categories

It is common to have *categorical data* in statistical data analysis (e.g. Male/ Female). In R such variables are referred to as factors. This makes it possible to assign meaningful names to categories.

```
Pain <- c(0,3,2,2,1)

SevPain <- factor(c(0,3,2,2,1),
levels=c(0,1,2,3),labels=c("none","mild","medium","severe"))

> SevPain
[1] none   severe medium medium mild
Levels: none mild medium severe
```

Factors a special vector to work with categories

A factor has a set of levels and labels and this can be confusing.

Levels: refers to the input values

Labels: refers to the output values of the new factor.

```
Pain <- c(0,3,2,2,1)

SevPain <- factor(c(0,3,2,2,1),
levels=c(0,1,2,3),labels=c("none","mild","medium","severe"))

> SevPain
[1] none   severe medium medium mild
Levels: none mild medium severe
```

Factors a special vector to work with categories

The levels of the new factor does not contain the value **“West”**
However, it makes sense to have all possible levels of your factor.

To add the missing level specify the **levels** arguments of **factor**:

```
> directions <- c("North", "East", "South", "South")

> factor(directions)
[1] North East  South South
Levels: East North South

> factor(directions, levels= c("North", "East", "South",
"West"),labels=c("N","E","S","W”))
[1] N E S S
Levels: N E S W
```

Summarizing factors

In factors values are repeated and it is interesting to have summarized information. `table()`

```
> head(state.region)
[1] South West  West  South West  West
Levels: Northeast South North Central West

> table(state.region)
state.region
    Northeast         South North Central          West
            9            16            12            13
```

Working with ordered factors

Sometimes categorical data has some kind of order. An example:

- Project status is described as low, medium, or high.
- A traffic light that can be red, yellow, or green.
- A gene underexpressed or overexpressed

The name for this type of data, where rank ordering is important is **ordinal data**. In R, there is a special data type for ordinal data called **ordered factors**

Working with ordered factors

```
> status <- c("Lo", "Hi", "Med", "Med", "Hi")
>ordered.status <- factor(status, levels=c("Lo", "Med",
"Hi"),ordered=TRUE)

> ordered.status
[1] Lo  Hi  Med Med Hi
Levels: Lo < Med < Hi

> table(status)
status
 Hi  Lo Med
  2   1   2

> table(ordered.status)
ordered.status
 Lo Med  Hi
  1   2   2
```