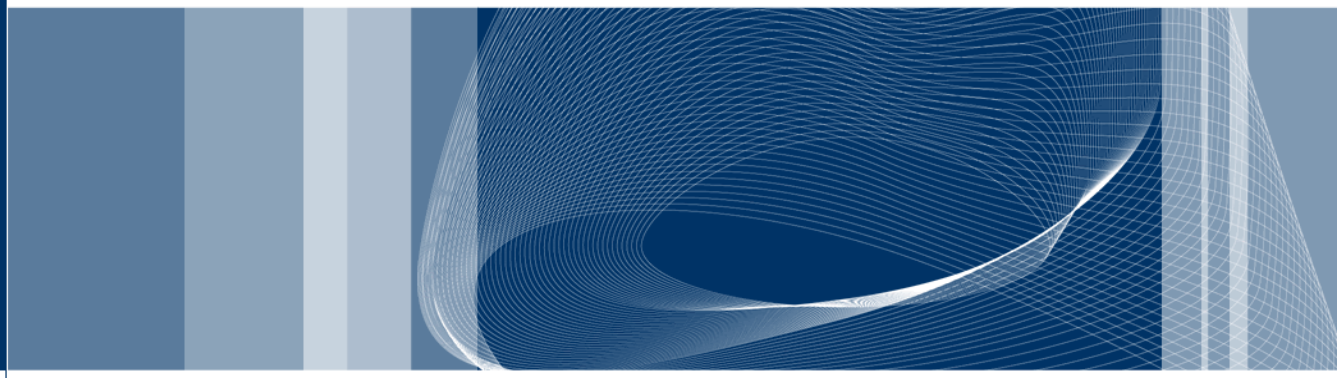
POLITECNICO DI MILANO



POLITECNICO DI MILANO

150°

GEOLaB
Geomatics and Earth Observation lab

Introduction to R

2nd lecture

Alessandro FERMI – Giovanna VENUTI

Outline of the lecture

In this lecture we will introduce

- other specialized functions for matrix objects
- ordered and unordered factors
- R Lists and data frames
- Some R data import functions
- How to access builtin datasets
- Built in probability distributions and statistical tools in R

Remind: arrays and matrices

The functions matrix() and array() are available for defining matrices and arras in R.

- "array()" function: the general syntax is

  Z <- array(data_vector, dim_vector)

  Example

  > x5 <- seq(by=.5, from=.5, to=12)
  > Z <- array(x5, dim=c(4,6))

- "matrix()" function: the general syntax is

  M <- matrix(data_vector, n_rows, n_cols)

Example

> B <- matrix(rnorm(50), 5, 10)
> B <- matrix(rexp(50), 5, 10)

Matrix operations

Recall that all usual matrix operations are available in R. In particular

Transpose: t(M)

Determinant: det(M)

Number of rows and columns: nrows(M) and ncols(M)

Elementwise product: A * B

Matrix multiplication: A %*% B

Example

```
>   nrow(B)
[1] 5
> BC <-B %*% C
> BC
```

Matrix operations

- Recall that a linear system of the form Ax = b, where A is a matrix and b is a known vector, may be solved with the solve() function

```
> x <- solve(A, b)
```

To compute the inverse use the command

```
> solve(A)
```

- Eigenvalues and eigenvectors: the function eigen() computes the eigenvalues and eigenvectors of a square (real and complex) matrix. It returns a list, consisting of two components, namely values and vectors.

Matrix operations

Example

```
> M <- matrix(rbinom(25, 50, 0.1), 5, 5)
> M
> eg <- eigen(M, symmetric=FALSE)
> eg
$values
[1] 25.880561+0.000000i -4.875864+0.000000i -0.166576+2.387233i
[4] -0.166576-2.387233i  1.328456+0.000000i
$vectors
[,1]          [,2]                  [,3]
[1,] 0.5007784+0i -0.74992023+0i -0.03486211-0.04257049i
[2,] 0.3462832+0i  0.38171642+0i  0.61899510-0.12158566i

> evals <- eg$values
> eg <- eigen(M, symmetric=FALSE, only.values=TRUE)
> eg
```

Matrix operations

- **Singular value decomposition**: the function svd(M) takes an arbitrary matrix and calculates its singular value decomposition. This consists of a matrix of orthonormal columns U, a matrix of orthonormal columns V and a diagonal matrix of positive entries D such that

$$M = U \%*\% D \%*\% t(V)$$

svd() returns a list with three components named d, u, v with evident meanings.

Example

```
> D <- matrix(c(1,0,0,0,0,0,0,4,0,3,0,0,0,0,0,0,2,0,0,0), 4,5)
> svd(D)
$d
[1] 4.000000 3.000000 2.236068 0.000000
```

Matrix operations

Least square fitting: the function lsfit() returns a list giving results of a least squares fitting procedure. An assignment such as

```
>   b <- lsfit(A, y, intercept=FALSE)
```

gives the results of a least squares fit, where y is the vector of observations and A is the design matrix (without considering an intercept term).

Another interesting function for a least square solution is the qr() function. Consider

```
> Xplus <- qr(X)
> b <- qr.coef(Xplus, y)
> fit <- qr.fitted(Xplus, y)
> res <- qr.resid(Xplus, y)
```

Matrix operations

These assignments compute the orthogonal projection of y onto the range of X in fit, the projection onto the orthogonal complement in res and the coefficient vector for the projection in b.
Remark: even though they may be still useful, these functions have been replaced by the statistical models features, which we will see in a forthcoming lecture.

Example. Given the design matrix

- A =

	[,1]	[,2]	[,3]
[1,]	2	3	5
[2,]	-4	2	3
[3,]	-1	1	1
[4,]	1	0	1

b = (10, 1, 0, 0)

Solve the least square problem using the lsfit() and qr() funtions.

Least square problem

```
>    data <- c(2,-4,-1,1,3,2,1,0,5,3,1,1)
>    Data <- matrix(data, 4,3)
>    Data
>    b <- c(10,1,0,0)
>    lsq1 <- lsfit(Data,b, intercept=FALSE)
>    lsq1
>    lsq1$coefficients
          X1        X2        X3
 1.0925926  2.7962963 -0.1759259
>   DataQR <- qr(Data)
>   DataQR
>   lsq1$qr$qr – DataQR$qr       # lsq1$qr is another list!!
>   par <- qr.coef(DataQR,b)
>   y_model <- qr.fitted(DataQR,b)
>   res <- qr.resid(DataQR,b)
> par
[1]  1.0925926  2.7962963 -0.1759259
```

R factors

A factor is vector object used to specify a discrete classification (grouping) of the component of a given data vector.

It has a “level” attribute.

Example. We have a sample of 30 tax accountants

```
>    state <- c("tas", "sa", "qld", "nsw", "nsw", "nt", "wa", "wa",
   "qld", "vic", "nsw", "vic", "qld", "qld", "sa", "tas", "sa", "nt",
   "wa", "vic", "qld", "nsw", "nsw", "wa", "sa", "act", "nsw", "vic",
   "vic", "act")
>    statef <- factor(state)
>    statef
[1] tas sa  qld nsw nsw nt  wa  wa  qld vic nsw vic qld qld sa
   tas sa  nt  wa
[20] vic qld nsw nsw wa  sa  act nsw vic vic act
Levels: act nsw nt qld sa tas vic wa
```

R factors – tapply() function

To continue the previous example, suppose to have vector

```
>  c(60, 49, 40, 61, 64, 60, 59, 54, 62, 69, 70, 42, 56, 61, 61, 61, 58, 51,
   48, 65, 49, 49, 41, 48, 52, 46, 59, 46, 58, 43)  -> incomes
>  n_acc <- tapply(incomes, statef, length)
>  n_acc
  act nsw  nt qld  sa tas vic  wa
    2   6   2   5   4   2   5   4
> inc_mean <- tapply(incomes, statef, mean)
> inc_mean
   act     nsw      nt     qld      sa     tas     vic      wa
 44.50   57.33   55.50   53.60   55.00   60.50   56.00   52.25
```

R factors – table() function

We already encountered the function table().
It allowed us to compute the absolute frequency distribution of one or more data vectors.

Actually, the table() function has factors as its arguments. In case simple vectors are applied to this function, then they are coerced to factors.

For instance

```
>   statefr <- table(statef)
>   statefr
statef
act nsw  nt qld  sa tas vic  wa
  2   6   2   5   4   2   5   4
```

is the same as statefr <- table(state) and to
tapply(incomes, statef, length)

R factors – table() function

A pair of factors defines a two way cross classification, and so on. The function table() allows then to compute frequency tables from equal length factors. If there are k factor arguments, the result is a k-way array of frequencies.

Example

```
>  factor(cut(incomes, breaks = 35+10*(0:7))) -> incomef
>  incomef
>  state_VS <- table(incomef, statef)
>  state_VS
          statef
incomef   act nsw nt qld sa tas vic wa
  (35,45]   1   1  0   1  0   0   1  0
  (45,55]   1   1  1   1  2   0   1  3
  (55,65]   0   3  1   3  2   2   2  1
  (65,75]   0   1  0   0  0   0   1  0
```

Extension to higher-way frequency tables is possible!

R lists

An R list is an object consisting of an ordered collection of objects known as its components.

The components do not have to be of the same type (or mode), but can be any R objects.

Example.
Lst <- list(name="Fred", wife="Mary", no.children=3, child.ages=c(4,7,9), name.children=c(“Tizio”,”Caio”,”Sempronio”))

The components of a list are always numbered and can be accessed through the “[[.]]” operator,

```
>     Lst[[1]]
[1] "Fred"
```

R lists

Otherwise they can be named and in this case the component may be referred to either by giving the component name as a character string in place of the number in double square brackets, or, more conveniently, by giving an expression of the form

> name$component_name

Example.

```
>    Lst$name
[1] "Fred"
>    Lst$name.children[2]
[1] "Caio"
>    Lst$name.children
[1] «Tizio» «Caio» «Sempronio»
```

R lists

R lists may be defined by the list() function, whose general syntax is

```
>   Lst <- list(name_1=object_1, ..., name_m=object_m)
```

Furthermore, lists can be extended with new elements; e.g.

```
>    Lst[6] <- list(matrix=Mat)
```

and can be concatenated with the function c()

```
>    list.ABC <- c(list.A, list.B, list.C)
```

R lists

Example

- Explore the lists of the eigenvalues and eigenvectors of a square matrix as obtained by the function “eigen()”

- Explore the lists by the functions “lsfit()” and qr()

R data frames

A data frame is a list with class "data.frame".
There are restrictions on lists that may be made into data frames:

- The components must be vectors (numeric, character, or logical), factors, numeric matrices, lists, or data frames.

- Matrices, lists, and data frames provide as many variables to the new data frame as they have columns, elements, or variables, respectively.

- Numeric vectors, logicals and factors are included as is, and by default character vectors are coerced to be factors, whose levels are the unique values appearing in the vector.

- Vector structures appearing as variables of the data frame must all have the same length, and matrix structures must all have the same row size.

R data frames

Objects satisfying the restrictions placed on the components of a data frame may be used to form one using the function data.frame()

```
>   accountants <- data.frame(home=statef, loot=incomes,
shot=incomef)
>   accountants
```

A list whose components conform to the restrictions of a data frame may be coerced into a data frame using the function “as.data.frame()”

R data frames

The simplest way to construct a data frame from scratch is to use the "read.table()" function to read an entire data frame from an external file.

R input facilities are simple and rather inflexible. The files to be read must have already a specific form obtained by using other editors.
Input file form with names and row labels:

	Price	Floor	Area	Rooms	Age	Cent.heat
01	52.00	111.0	830	5	6.2	no
02	54.75	128.0	710	5	7.5	no
03	57.50	101.0	1000	5	4.2	no
04	57.50	131.0	690	6	8.8	no
05	59.75	93.0	900	5	1.9	yes

R data frames – read.table()

If the file to be read has this format, then the “read.table()” function can be used directly

```
> House <- read.table("houses.data")
```

If the row labels must be omitted, one may use the logical argument “header” in read.table()

```
> House <- read.table("houses.data", header=TRUE)
```

Remark: there are other input functions, e.g. The scan() function, for which we refer to the literature

R data frames

To illustrate the read.table() function, let us consider the following example: in the package “datasets” there is a data frame called “Morley”.
To check that the “datasets” package is loaded, first use the command

```
>   search()
[1] ".GlobalEnv"        "package:stats"     "package:graphics"
[4] "package:grDevices" "package:utils"
   "package:datasets"
[7] "package:methods"   "Autoloads"         "package:base"
```

R data frames

Now to check which datasets are stored in the “datasets” package write

```
>   data(package=”datasets”)
>   morley
> filepath <- system.file("data", "morley.tab" ,
  package="datasets")
> filepath
[1] "C:/PROGRA~1/R/R-2~1.0/library/datasets/data/morley.tab"
>  morley <- read.table(filepath)
> morley
class(morley)
[1] "data.frame"
```

R data frames

When working with data frames or lists, the notation $ notation is not always convenient.

For many purposes, it could be useful to make the components of the data frame or list temporarily visible in the current workspace.

This is achieved by means of the “attach()” function, e.g.

```
> attach(morley)
```

This makes the data frame visible in the search path at position 2 (or above) and the components can be used as variables in their own right.

R data frames

More precisely, write in the console the following commands

```
>   morley
>   search()
>   ls()
```

The data frame 'morley' is loaded, but it is not present in the search path. Attaching it means making it visible in the search path

```
>   attach(morley)
>   search()
    >   ls(2)
```

R data frames

At this point, an assignment like the following is possible

```
>    aux <- Speed / 1000
```

This does not change permanently the “morley” data frame.
If permanent changes have to be stored, then one has to use the $ notation specifying the name of the data frame.

Example

```
>   aux <- Speed / 1000
>   morley$Speed <- aux
```

However the new value is not visible until the data frame is detached with the “detach()” function.

R data frames

Remark. When invoked on a data frame or matrix, the “edit()” function brings up a separate spreadsheet-like environment
for editing.

This is useful for making small changes once a data set has been read. The command

> xnew <- edit(xold)

will allow you to edit your data set xold, and on completion the changed object is assigned
to xnew.

Example

As an example with an available data frame, let us examine its distribution; first of all let us find the 'faithful' data frame

```
>   data(package=”datasets”)
>   faithful
>   class(faithful)
[1] "data.frame"
>   attach(faithful)
>   ls(2)
[1] "eruptions" "waiting"
>   summary(eruptions)
```

Example

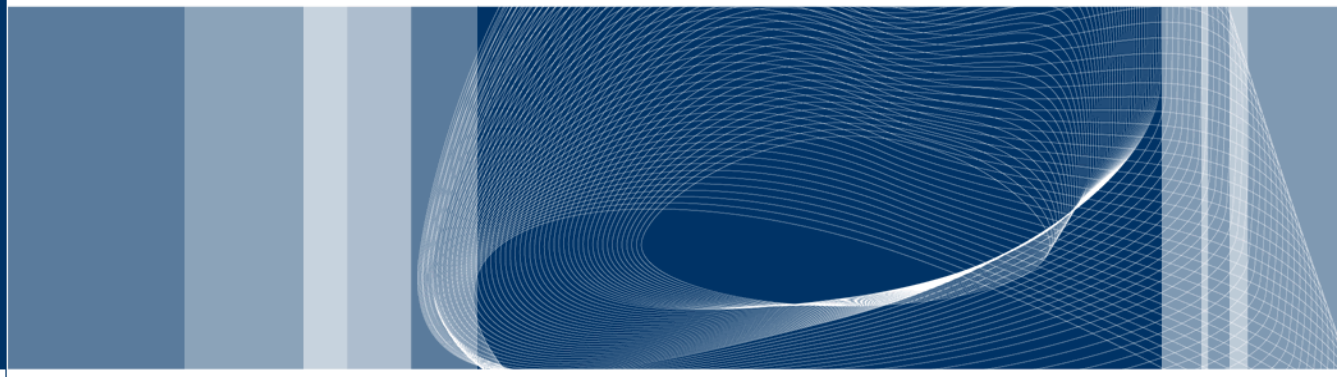
Continuing the example above, R offers a “hist()” function to display the data of interest

```
>   hist(eruptions)
```

To customize the size of the bins and make a plot of density, use the commands

```
>  hist(eruptions, seq(1.6, 5.2, 0.2), prob=TRUE)
> lines(density(eruptions, bw=0.1))
> rug(eruptions) # show the actual data points
```

POLITECNICO DI MILANO



POLITECNICO DI MILANO

150°

GEOLaB
Geomatics and Earth Observation lab

THANK YOU FOR YOUR ATTENTION!