Chapter I. Introduction to R: the basic objects

- In order to work with a language we need to know its syntax and its objects.
- In terms of the syntax, we have seen how to use variables and functions. We will learn more as we progress.
- This section focuses on the objects in R that we will routinely work with: vectors, matrices, factors, data frames and lists.

Vectors

- A vector is a collection of objects all of the same data type or mode.
- If necessary, R will coerce your input in order to enforce that rule.
- To create a vector use the function "vector", or simply create a new variable.
- Another common way of creating a vector is using the concatenation function 'c' or use a function that generates a vector.
- A vector has two basic properties: its mode/type, and its length.

Vectors

```
x1 = numeric(20)# a 20-components vector of real numbers
##all 20 components of x1 are zero
x2=c(1:10) # a vector of integers 1 to 10
##the function c concatenates its arguments
s1=seq(from=1,to=10,by=1) #same as x2
v=rnorm(n=10,mean=1,sd=1) #returns a vector
```

Vectors

```
x3=c(T,F,FALSE,TRUE) # a vector of logical values
##we can use either T/F or TRUE/FALSE
x4=c('MY','NAME', 'IS') # a vector of characters
x5=c(2+2i,complex(real=cos(pi/3),imaginary=sin(pi/3)))
# a vector of complex numbers
##notice the difference between 2i and 2*i
typeof(x5)
length(x4)
```

Vectors

Calculus is vectorized in R and obeys the following two basic rules:

1. A function that can operate on a data type will operate on a vector of such data type.
2. recycling rule: In an operation that involves two vectors of different lengths, the shortest vector is "recycle" to match the length of the longest.

```
x=c(1,3)
exp(x)
x + 2 #same as x+ c(2,2)
```

Vectors

The recycling rule: consider the following example.

```
x = c(1,2);  y=1:5
x+y #same as c(1,2,1,2,1) + y
```

In formal calculus, the last expression is not valid because the vectors are not of the same size. R uses the recycling rule: repeat the elements of shortest vector until the two vector have the same length.

Vectors

There are special behaviors worth knowing. Try this and explain.

```
x = c(1,2,3,5);
x + 1;
x + numeric(0);
```

Logical vectors

- Let us look a bit more closely at vectors of logical.
- Logical can be either TRUE, FALSE or NA (for missing).
- These variables obey the rules of Boolean algebra with operators AND, OR and NOT.
- In R, AND is &, OR is | and NOT is !

Logical vectors

```
A=c(TRUE,TRUE);B=c(TRUE,FALSE)
!A  # gives FALSE, FALSE
A&B # gives TRUE, FALSE
A|B # gives TRUE, TRUE
```

Logical vectors

- Logical vectors arise typically as the result of comparison operations.

```
x=1:5
x>2 #same as x>c(2,2,2,2,2). Returns a vector of logicals
x<=2
x!=2
x==2
all(x==2); any(x==2); which(x==2)
```

Logical vectors

- Many functions in R operate on vectors of logical. We just saw 3 of them: all, any, which
- R allows to do usual calculations on logical vectors. In this case, the value TRUE is taken as 1 and FALSE as 0. This can be very useful in practice. Consider the following:

```
x=rnorm(n=100)
mean(x>0) # what is this computing ?
##Can you guess the answer ?
```

Logical vectors

Activity: Generate a vector X of length 100 filled with independent standard normal random variables. Find the proportion of those values that fall between -1 and 1.

Subsetting

- Accessing elements of collections of objects is an important operation.
- R provides a very powerful and flexible facility for this.
- We can select subsets of a vector by inclusion, exclusion, by name and by logical indexing. The result is another vector.

Subsetting

Try out and explain

```
x=rbinom(n=10,size=20,prob=0.5)
x[1]  # Returns first component of x
#this is selection by inclusion
x[-c(1,2,9,10)]  #selection by exclusion
#we keep all of x but components 1,2,9,10
```

Subsetting

We continue from last slide:

```
x[x>5]  # select components of x larger 5
## selection by logical indexing
u=x>5; x[u] # same as previous line
x[c(TRUE,FALSE)] # explain the result
```

Subsetting

- When we do subsetting by logical indexing, the vector of logicals needs to be of the same length. Otherwise the recycling rule is applied.

```
x[x>5]  # select components of x larger 5
## selection by logical indexing
u=x>5; x[u] # same as previous line
x[c(TRUE,FALSE)] # explain the result
```

Subsetting

Activity:

```
x=rbinom(n=10,size=20,prob=0.5)
#explain the difference between
u1=(x>5); (u1=x)>5; (u1==x)>5;
```

Understanding "NULL","NA","NaN"

- NULL is a special object called the NULL object.
- NA is R symbol for missing data.
- NaN in R means "not a number". Typically the result of an undetermined numerical operation. NaN is often treated by many functions as a special case of NA.

Understanding "NULL","NA","NaN"

```
z1=character(0) # empty variable of mode character
z2=NULL  #$empty object
y=1/0; y-y #\infty-\infty return NaN
z3=c(1,3,NA,23,NaN,4);
## notice we did not write 'NA' or 'NaN'
is.na(z3) #NaN is a special case of NA
is.nan(z3).
#NA is not a special case of NaN
u=is.na(z3)
z3[!u]
```

Factors

- Factors are special vectors designed to hold categorical or ordinal variables.
- In other words, a factor is a character vector but where each entry of the vector can take only a limited number of values.
- Example: in a survey you record the gender of individuals in the sample ('male', or 'female'). You can code this in R as either a character vector or as a factor.
- A factors handled such data better.

Factors

- We create a factor using the function factor.
- Also when loading a data file into R using read.table and alike, variables that contains characters are loaded as factor by default. More on this later.

Factors

```
gender_1 = c('male','male', 'female',
     'male', 'female','female','female','male');
typeof(gender_1); #character
is.factor(gender_1);#should be FALSE
gender_2=factor(gender_1); # create a factor
gender_2
levels(gender_2); #alphab. order by default
# we can set the order of the levels
gender_3=factor(gender_1,levels=c('male','female'));
gender_3
table(gender_2); # produce a count table
```

Factors

- Internally factors are represented as integer vectors.
- Which means that R can coerce a factor to an integer vector (often without warning).

Factors

```
#suppose we want to append a new value
c(gender_2, 'male')# does not work
factor(c(as.character(gender_2),'male'),
           levels=c('male','female'))
#Suppose we want a subset of the data
male_only = gender_2[gender_2=='male']
male_only; table(male_only); #level 'female' hangs around
male_only = gender_2[gender_2=='male',drop=T]
male_only; table(male_only);#better
```

Matrix

- A matrix is a rectangular table where all entries have the same type. Most of the time we work with numerical matrices.
- R supports matrices and has a good numerical linear algebra library.
- You create matrix in R using the function 'matrix'.
- By default the matrix is filled by column. Use the argument 'byrow' to fill the matrix by row.

```
M1=matrix(data=1:8,ncol=4, nrow=2)
M2=matrix(data=1:8,ncol=4,nrow=2,byrow=T)
rownames(M1)=letters[1:2] #this is rarely useful
colnames(M1)=letters[1:4] #this is rarely useful
dim(M1); ncol(M1); nrow(M1)
```

Matrix

- To index a matrix, use the same techniques as for vectors but with the first index for rows and the second index for column.

```
M1=matrix(data=1:8,ncol=4, nrow=2)
rownames(M1)=letters[1:2]
colnames(M1)=letters[1:4]
M1[1,1] #entry (1,1)
M1[-1,2] #column 2 without first element
M1['a',] #row named 'a'
M1[,c(TRUE,TRUE,FALSE)] #select all columns but the third
# Question: what is the output of
M1[,c(TRUE,FALSE,FALSE)]
```

Matrix

When we take a one-dimensional subset of a matrix, the result by default is coerced into a vector, unless we use the argument drop.

```
M1[,1]
# compare with
M1[,1,drop=F]
```

Matrix calculus

- As with vectors, any operation on numerics will also work on matrix of numerics, one entry of the matrix at the time.
- R supports all the common matrix operations.
- It is important to know that in R, $A + B$, $A - B$, A/B, $A * B$ are all element by element operations.
- This is all fine, except for $A * B$. To do the usual matrix multiplication in R, we use $A\% * \%B$.

Matrix calculus

An example:

```
A=matrix(1:4,2,2);B=matrix(5:8,2,2);
A+B
A/B
A*B
A%*%B
log(exp(A))
```

- The inverse is obtained with the function 'solve'.
- Other useful function: 'eigen', 'det'. The transpose is 't'.

Matrix calculus

-
  ```
  eig=eigen(A)
  C=diag(2)
  det(A-eig$values[1]*C);
  #should return approx. zero, by definition
  B=solve(A) # return the inverse of A
  ```
- Equivalently to the usual determinant definition, λ is an eigenvalue for A with eigenvector $u \neq 0$ is $Au = \lambda u$. Check that this definition holds (approx.) on the example in the code above.

Matrix calculus

Activity: Use R to solve the following linear system $Ax = b$ where

$$A = \begin{pmatrix} 1 & 0 & 6 & 2 \\ 8 & 0 & -2 & -2 \\ 2 & 9 & 1 & 3 \\ 2 & 1 & -3 & 10 \end{pmatrix}, \quad \text{and} \quad b = \begin{pmatrix} 6 \\ -2 \\ -8 \\ -4 \end{pmatrix}.$$

Lists

- Lists are collections, like vectors. But unlike vectors, the same list can hold different types of objects.
- You create a list with the function list.
- You can also access the components using the operator '[[ ]]' not '[ ]' as for vectors and matrices.
- Why use lists? it allows you to put together objects of different types. Vectors don't.

Lists

```
V=c('A','A','C','B','B');
X=1:5;
Y=rnorm(n=5,mean=0,sd=1);
L=list(X,Y,V)
#You can name the entries of the list
names(L)
names(L)=c('V1','V2','V3')
L$V1
L[[1]] #same as L$V1
```

Lists

But we can also subset a list using a single bracket. In which case, we get another list. For instance:

```
L2=L[2] # this is not the vector 1:5
L2[3] # will return empty. What we want is
L2=L[[2]]; L2[3]
```

Data frames

- Roughly a data frame is a rectangular table with rows and columns. The columns represent variables and has its own type (numeric, integer, factor, character); a major difference from matrices. The rows are records or cases.
- Typically we create data frames by reading data from files. We can also create data frames from vectors with the function *data.frame*.
- You can convert a data frame to a matrix with 'data.matrix'.

```
V=c('A','A','C','B','B');
X=1:5;
Y=rnorm(n=5,mean=0,sd=1);
dt1=data.frame(X,Y,V)
names(dt1)=c('meas1','meas2','meas3')
#notice how the character vector is automatically
#transform into a factor
```

- We think of data frames both as a matrix (a rectangular table with rows and columns) and as a list (a list of statistical variables on which we have observations).

```
#Continuing with the previous example:
dt1[1,1]# returns 1
dt1$meas1 # same as dt1[[1]]
```

Working with data frames:

```
dt2=airquality
dim(dt2);names(dt2)
edit(dt2)
#Compare and understand the following
mean(dt2$Ozone); mean(dt2$Ozone,na.rm=T)
#better yet: 'complete.cases'
Ind=complete.cases(dt2)
dt=dt2[Ind,]
mean(dt$Ozone) #or mean(dt[,1]) or mean(dt[[1]])
plot(dt$Ozone,type='l')
```