LANGEBIO - BIOSTATISTICS

OCTAVIO MARTÍNEZ DE LA VEGA

1. Aim of the course

The main goal in this course is to make a reflexion about the basis of Statistics. In this short time -20 hours of lecturing, plus the time that you employ in the developing some abilities in computing, it is impossible to cover even a small part of the large field of applications of Statistics to Biology. Fortunately, Statistic is based in a few fundamental concepts, what I call the *Central Core of Statistics*. I will try to demonstrate these ideas, illustrating them with simple examples, which hopefully will be easy to follow and understand, with the aim of rediscovering these concepts for ourselves.

To demonstrate these ideas we are going to use a very powerful computing environment, the R language. In this document you will see the text that is input in R and the output of the commands, you must consult the help for each one of the functions that will be used and explore the results which you can obtain with distinct inputs. Be curious, be inquisitive, be creative. However, this is NOT an R course, we are going to use it just as a tool to understand the concepts; feel free of ask any type of questions, about the concepts which we are exploring, but also about the syntax of the commands needed to do something. The most frustrating part of learning R -or any other computer language, is that the computer do what she (it is a female!) is told to do, which can be different of what we want she to do!.

2. A brief introduction to R

When you run R, you enter into an interactive window, in which you can see only the prompt: ">". That means that R is ready to take your orders, and in contrast with programs with work with the mouse you MUST type your orders (a very important change if you had never done that). Do not worry too much, you will learn R quickly...

What can we do in R?

Well, R knows about arithmetic operations:

```
> 2+3; 3-2; 5*2; 15/3; 2^5 # You can see that ";" is like "return"
```

Date: April 2012.

```
[1] 5
[1] 1
[1] 10
[1] 5
[1] 32
```

There are also functions for all common needs; for example

```
> exp(1); pi; log(exp(1)); sin(1); asin(0.841471) # Etc.
[1] 2.718282
[1] 3.141593
[1] 1
[1] 0.841471
[1] 1
> # Probably you have realized that lines beginning with "#" are comments
```

Importantly, R can work with "chains" of numbers (vectors), and most of the operations can be performed with the whole vector

```
> c(1, 2, 3, 4) # A small vector (chain) of numbers
[1] 1 2 3 4
> c(1:4) # The same an in abbreviated way...
[1] 1 2 3 4
> 4*c(1:4) # The vector multiplied by 4
[1]  4  8 12 16
> sqrt(c(1:4)) # The square root of the numbers in the vector
[1] 1.000000 1.414214 1.732051 2.000000
> c(1:4)*c(4:1) # The product of the vectors (element by element)
[1] 4 6 6 4
```

Of course, the main strength of R is Statistics, thus there are many statistical functions programed in it. For example,

```
> c(8.97, 10.06,  9.29,  7.44,  9.48) # Some data
[1]  8.97 10.06  9.29  7.44  9.48
> mean(c(8.97, 10.06,  9.29,  7.44,  9.48)) # The average
[1] 9.048
> median(c(8.97, 10.06,  9.29,  7.44,  9.48)) # The median
[1] 9.29
> min(c(8.97, 10.06,  9.29,  7.44,  9.48)) # The minimum
```

```
[1] 7.44
> max(c(8.97, 10.06,  9.29,  7.44,  9.48)) # The maximum
[1] 10.06
> sd(c(8.97, 10.06,  9.29,  7.44,  9.48)) # The "Standard deviation"
[1] 0.9824816
> var(c(8.97, 10.06,  9.29,  7.44,  9.48)) # The variance
[1] 0.96527
> summary(c(8.97, 10.06,  9.29,  7.44,  9.48)) # A summary... etc
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  7.440   8.970   9.290   9.048   9.480  10.060
```

R work with *objects*. Objects can be numbers, vectors, functions, etc. In R you can assign a variable to be something... For example, we can assign the data that we have seen to be x, say

```
> x <- c(8.97, 10.06,  9.29,  7.44,  9.48) # Assign x to be those numbers.
> x # Let's see what is in x... Do you see?
[1]  8.97 10.06  9.29  7.44  9.48
```

Now you can work with x as you wish...

```
> sd(x) # It's standar deviation...
[1] 0.9824816
> summary(x) # It's summary...
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  7.440   8.970   9.290   9.048   9.480  10.060
> 3*x # x times 3, etc. Get it?
[1] 26.91 30.18 27.87 22.32 28.44
```

R has good helping facilities. If you want you can start an HTML window with help typing "help.start()". Also, every function existent in R has help documentation. For example, type "? mean" (plus Return) to get help about the function "mean". You can also do a fuzzy search for a word or sentence using double "?"; for example to see if the word Poisson is included somewhere, you can type "?? Poisson", etc.

In R there are many datasets already "included"; type data() to see a list. Let's work with a dataset named DNase (Elisa assay of DNase).

```
> data(DNase) # Make the data available
> # Type "? DNase" to see the help for this dataset
> summary(DNase) # Get a summary
```

```
      Run          conc              density
 10     :16   Min.   : 0.04883   Min.   :0.0110
 11     :16   1st Qu.: 0.34180   1st Qu.:0.1978
 9      :16   Median : 1.17188   Median :0.5265
 1      :16   Mean   : 3.10669   Mean   :0.7192
 4      :16   3rd Qu.: 3.90625   3rd Qu.:1.1705
 8      :16   Max.   :12.50000   Max.   :2.0030
 (Other):80
```

From the help of the dataset we know that "The DNase data frame has 176 rows and 3 columns of data obtained during development of an ELISA assay for the recombinant protein DNase in rat serum.", we have three variables: "Run", "conc" and "density". The data are arranged in a "data.frame", that is like a matrix (you can think of it as an Excel file) where rows are data and columns are variables. Let's see:

```
> DNase[1:5, ] # This means first 5 rows and all columns
  Run       conc density
1   1 0.04882812   0.017
2   1 0.04882812   0.018
3   1 0.19531250   0.121
4   1 0.19531250   0.124
5   1 0.39062500   0.206
> DNase[2, 2:3] # Second row, columns 2 and 3
        conc density
2 0.04882812   0.018
> DNase[1, 1] # First row, first column
[1] 1
Levels: 10 < 11 < 9 < 1 < 4 < 8 < 5 < 7 < 6 < 2 < 3
> DNase$conc[1:10] # The first ten concentrations
 [1] 0.04882812 0.04882812 0.19531250 0.19531250 0.39062500 0.39062500 0.78125000
 [8] 0.78125000 1.56250000 1.56250000
> names(DNase) # The names of the columns of "DNase"
[1] "Run"     "conc"    "density"
> DNase[DNase$Run == "1", ] # All the columns of first Run
   Run        conc density
1    1  0.04882812   0.017
2    1  0.04882812   0.018
3    1  0.19531250   0.121
4    1  0.19531250   0.124
```

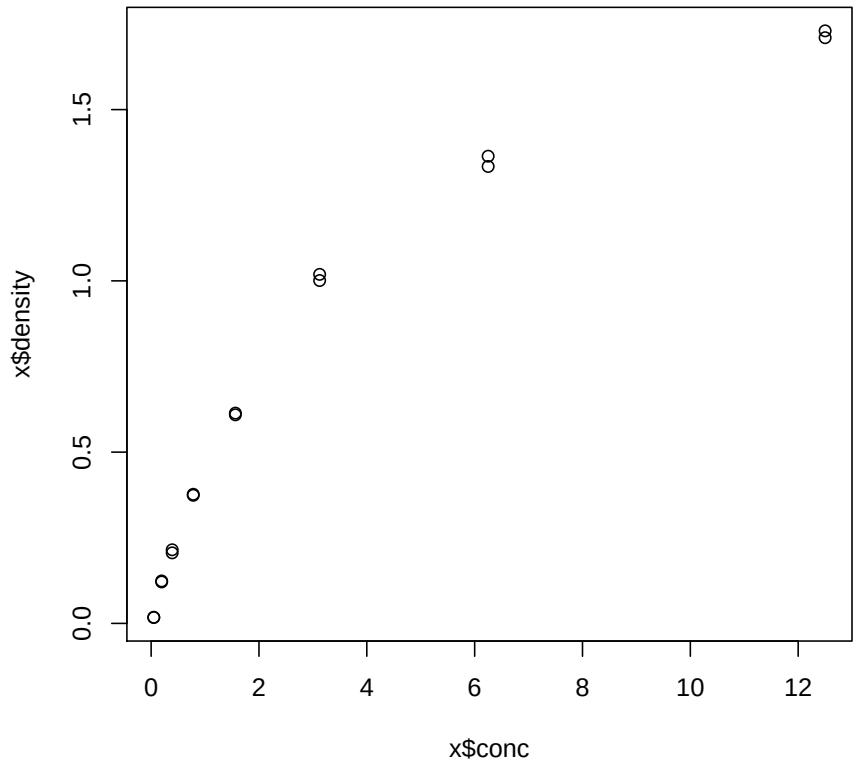
```
5    1  0.39062500   0.206
6    1  0.39062500   0.215
7    1  0.78125000   0.377
8    1  0.78125000   0.374
9    1  1.56250000   0.614
10   1  1.56250000   0.609
11   1  3.12500000   1.019
12   1  3.12500000   1.001
13   1  6.25000000   1.334
14   1  6.25000000   1.364
15   1 12.50000000   1.730
16   1 12.50000000   1.710
```

```
> x <- DNase[DNase$Run == "1", ] # What am I doing?
> x
```

```
   Run        conc density
1    1  0.04882812   0.017
2    1  0.04882812   0.018
3    1  0.19531250   0.121
4    1  0.19531250   0.124
5    1  0.39062500   0.206
6    1  0.39062500   0.215
7    1  0.78125000   0.377
8    1  0.78125000   0.374
9    1  1.56250000   0.614
10   1  1.56250000   0.609
11   1  3.12500000   1.019
12   1  3.12500000   1.001
13   1  6.25000000   1.334
14   1  6.25000000   1.364
15   1 12.50000000   1.730
16   1 12.50000000   1.710
```

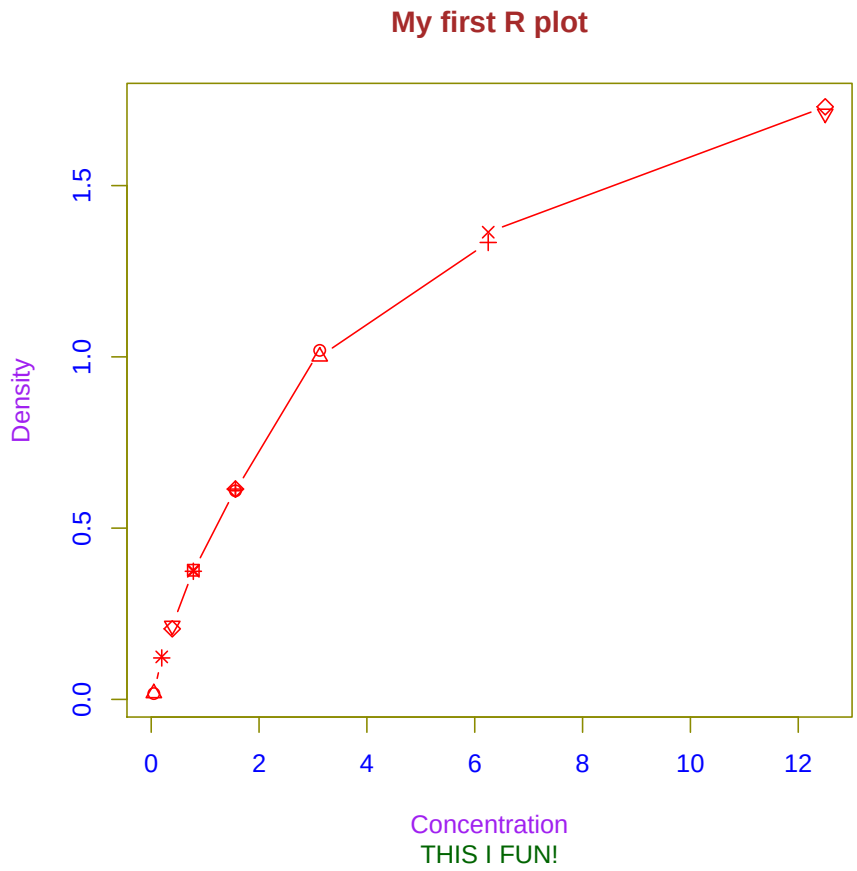
R has also extensive graphing capabilities; for example, let's see a plot of concentration versus density of our object x:

```
> plot(x$conc, x$density)
```



And we can annotate the plot, for example

```
> plot(x$conc, x$density,
+ xlab="Concentration", ylab="Density",
+ type="b", col="red",
+ main="My first R plot",
+ sub="THIS I FUN!", col.axis="blue",
+ col.lab="purple", col.main="brown",
+ col.sub="darkgreen", fg="yellow4", pch=c(1:10))
```



For a nice demonstration of the abilities of R to do plots type demo("graphics").

So far we have been working with the facilities already present in R, but an interesting point is that we can extend those facilities by programing our own functions. Assume that we want to program the linear function Y = a + bX. Well, that is straight forward in R:

```
> my.fun <- function(x, a=0, b=1){a + b*x} # That is it!
```

The values for default for the parameters a and b are zero and one, respectively. Now we can use our function:

```
> my.fun(0) # Evaluating Y = 0 + 1*0
[1] 0
> my.fun(5) # Evaluating Y = 0 + 1*5
[1] 5
> my.fun(5, a=1, b=2 ) # Evaluating Y = 1 + 2*5
```

```
[1] 11
```

```
> my.fun(5, 1, 2) # Same than above
```

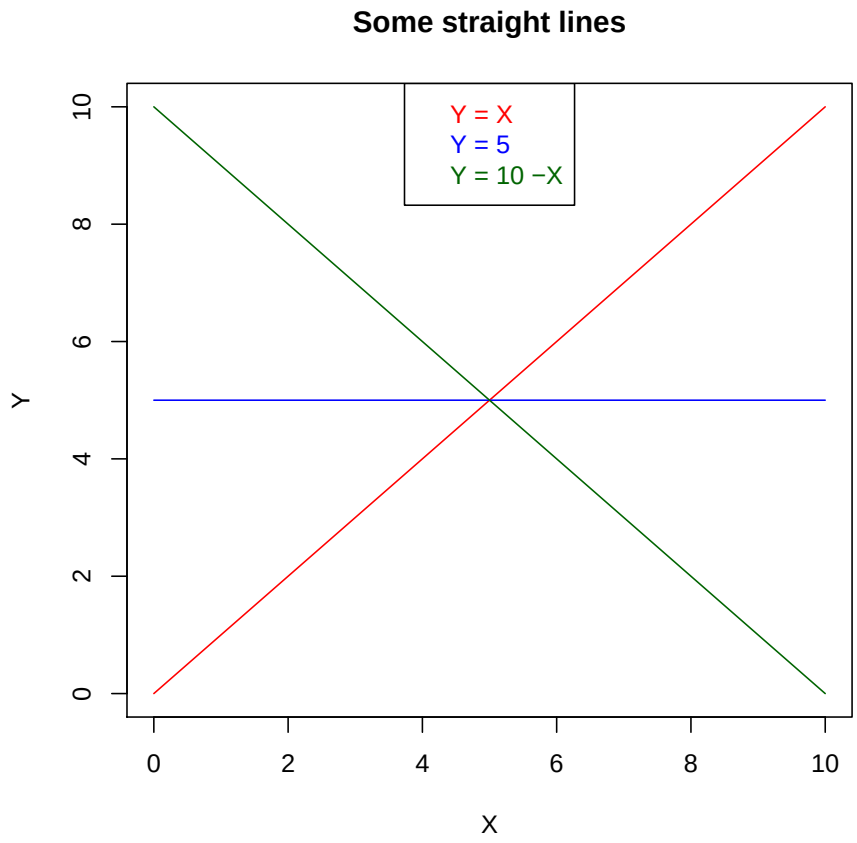
```
[1] 11
```

```
> my.fun(c(1:10), a=1, b=2 ) # Evaluating Y = 1 + 2*(1, 2, ... 10)
```

```
 [1]  3  5  7  9 11 13 15 17 19 21
```

Let's do a plot of some straight lines...

```
> plot(c(0:10), my.fun(c(0:10)),
+ type="l", col="red", xlab="X", ylab="Y",
+ main="Some straight lines")
> points(c(0:10), my.fun(c(0:10), 5, 0),
+ type="l", col="blue")
> points(c(0:10), my.fun(c(0:10), 10, -1), type="l",
+ col="darkgreen")
> legend("top", legend=c("Y = X", "Y = 5", "Y = 10 -X"),
+ text.col=c("red", "blue", "darkgreen"))
```



A very nice facility in R is that we can **simulate** *random* numbers with various distributions. For example, let's assume that you want to sort 20 plants, numbered 1, 2, ... 10 into two treatments. We can use the function **sample**, say

```
> sample(c(1:20), 10) # Get 10 numbers at random form c(1:20)
 [1]  9  2 12 13  6 11 19  5 20  4
```

Then you can assign that plants to the treatment one. If you run this code in your machine, very likely your own results will be different; each time that the function is run you will get a random sample.

In summary, with R you have the most powerful statistical tool that there is, and it is free!. There are "packages" (collections of functions and datasets) almost for every statistical application. Learning R is a good investment of your time if you ever are going to analyze data, and I am pretty sure you will.

2.1. **Homework.** How to input data into R?

There are various ways, the most straight forward is to input the data from a text file. You need to have the data into a pure ASCII file; one that is generated with a **pure text** application, like "vi" in Linux or the notepad in Windows or TextEdit in MacOS etc. DO NOT use Words or, if you use it, save it as text.

The next table has data of number of gene tags in an experiment with the fungus Phycomyces (Unpublish data from the Lab. of Alfredo Herrera-Estrella in Langebio). There were four libraries: $S.R_1$ = Sporangiophores, replicate 1; $S.R_2$ = Sporangiophores, replicate 2; $M.R_1$ = Mycelia, replicate 1 and $S.R_2$ = Mycelia, replicate 2. The data are for three genes (17, 1661 and 7433). The row "Others" has the sum of the total number of gene tags which were sequenced from other genes different to 17, 1661 or 7433.

Your mission is to input the data (number of tags, NOT the transcripts per million, which are the numbers in **bold face** between parenthesis) into an structure say "phyco". You can use the function **read.table**. See the help of that function. Note that you need to put the file to be read INTO the "working directory" of your R session. To see which is that directory use *getwd()*.

Gene	$S.R_1$	$S.R_2$	$M.R_1$	$M.R_2$
17	22 **(40)**	6 **(10)**	30 **(46)**	18 **(19)**
1661	56 **(101)**	43 **(71)**	1,440 **(2,205)**	2,163 **(2,317)**
7433	40 **(72)**	24 **(39)**	141 **(216)**	64 **(69)**
Others	551,957	608,085	651,479	931,356
Total	552,075	608,158	653,090	933,604

If you have some small dataset, generated by you, or in your lab or from a paper, input into R using the procedure that you just learned.