Una introducción a R & RStudio

Daniel Barráez, Alejandro Labarca

Mayo, 2018

Contenido

1. Unidad I: El entorno de R & RStudio

- Tema 1: Fundamentos básicos
  - Entorno, instalación y comandos básicos
  - Manipulaciones simples: números y vectores.
  - Objetos, sus modos y atributos.
  - Arreglos y matrices.
  - Listas y data frames.
  - Estructuras básicas de programación: Bucles, condicionantes y funciones.
  - Packages destacados: base, ggplot2, astsa, R Markdown, dplr, Shiny.
  - Importación y exportación de datos en distintos formatos.

Contenido

2. Unidad II: Análisis estadísticos en R

- Tema2: Estadística descriptiva y elaboración básica de gráficos.
- Tema3: Pruebas de hipótesis. Listas cuáles.
- Tema4: Modelo lineal uni-variado y multivariado. Estimación y diagnóstico.
- Tema5: Series de tiempo. La librería astsa. Modelos ARMA, ARIMA, SARIMA. Estimación y diagnóstico.

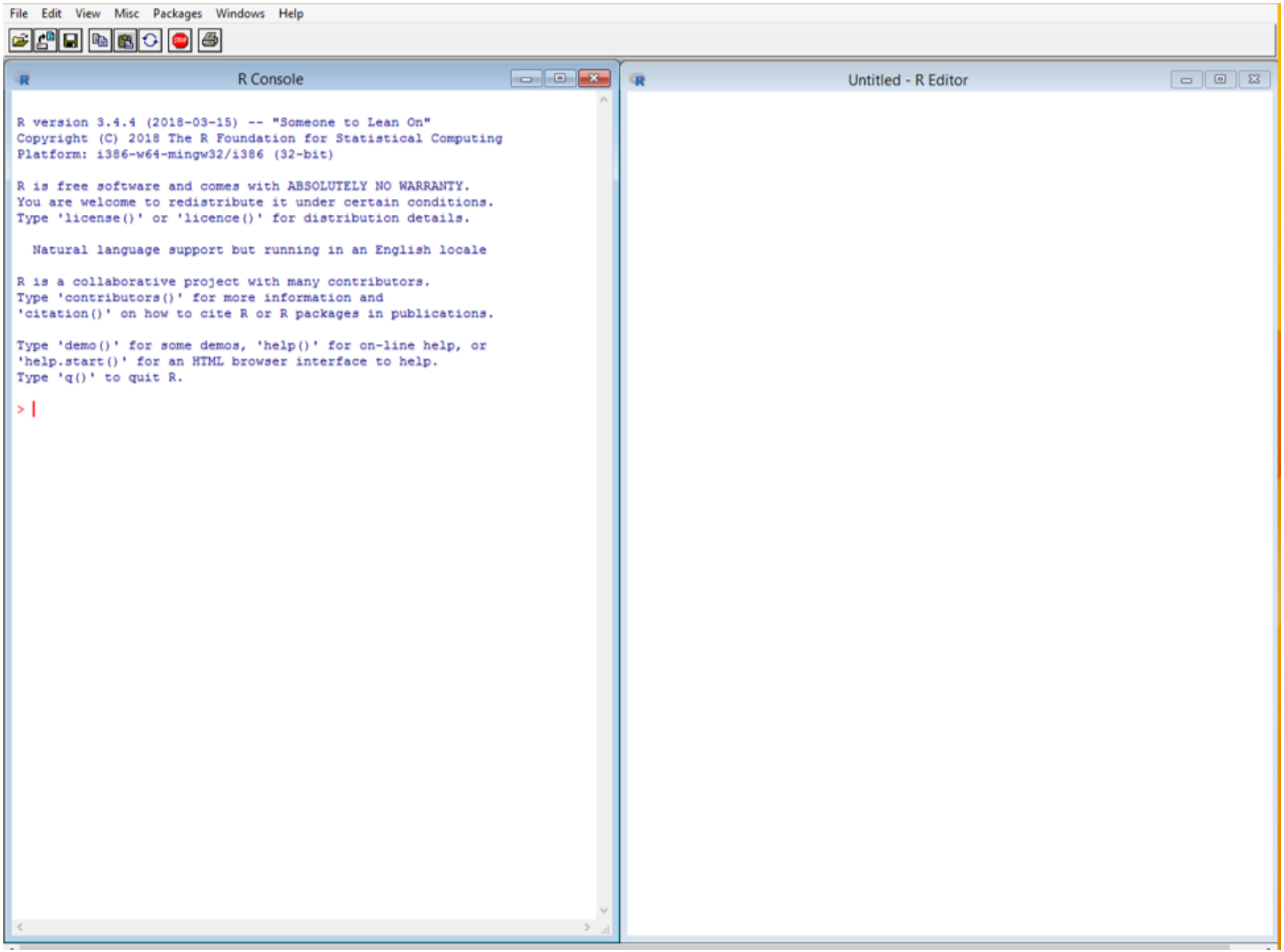
Historia

- Ross Isaka: "... descubrí un maravilloso libro de Hal Abelson y Gerald Sussman llamado THE STRUCTURE AND INTERPRETATION OF COMPUTER PROGRAMS..."
- Ross Isaka y el lenguaje S de Rick Becker y John Chambers: similitudes y diferencias entre S y Scheme.
- Ross Isaka y Robert Gentleman: Universidad de Auckland. Interés en la informática estadística.
- En agosto de 1993 fué el primer anuncio en el laboratorio de enseñanza. En junio de 1995 lanzan el código fuente R como "software libre"

El entorno de R

R es un entorno de software libre para computación y gráficos estadísticos. Entre otras cosas, posee:

1. Una efectiva manipulación y facilidad de almacenamiento de los datos.
2. Operadores para el cálculo en arreglos, en particuar matrices.
3. Una amplia, coherente, colección integrada de herramientas para el análisis de los datos.
4. Facilidades gráficas para el análisis y presentación de los datos.
5. Un buen desarrollador, simple y efectivo lenguaje de programación (llamado 'S') que incluye condicionales, bucles, funciones recursivas definidas por el usuario y entradas y salidas con gran facilidad. (en efecto, la mayoría de las funciones suministradas por el sistema esta escrito en lenguaje S).


File Edit View Misc Packages Windows Help
R Console
R version 3.4.4 (2018-03-15) -- "Someone to Lean On"
Copyright (C) 2018 The R Foundation for Statistical Computing
Platform: i386-w64-mingw32/i386 (32-bit)
R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.
Natural language support but running in an English locale
R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.
Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.
>
Untitled - R Editor

Ranking peers

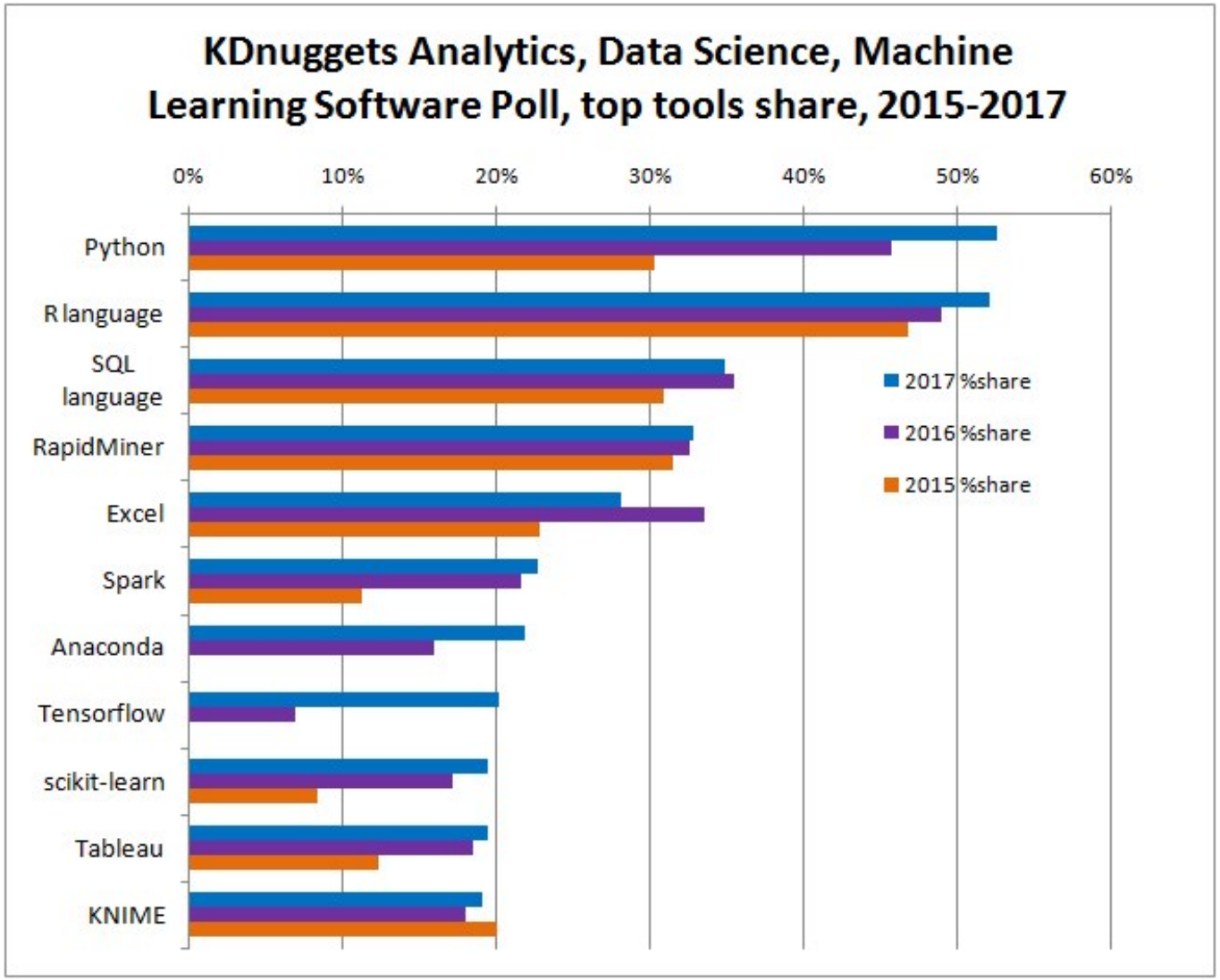


Figure 2:

La distribución R posee una versión HTML (https://stat.ethz.ch/R-manual/) actualizada donde se disponen los manuales de referencia:

1.- An Introduction to R.

2.- R Data Import/Export.

3.- R Installation and Administration.

4.- Writing R Extensions.

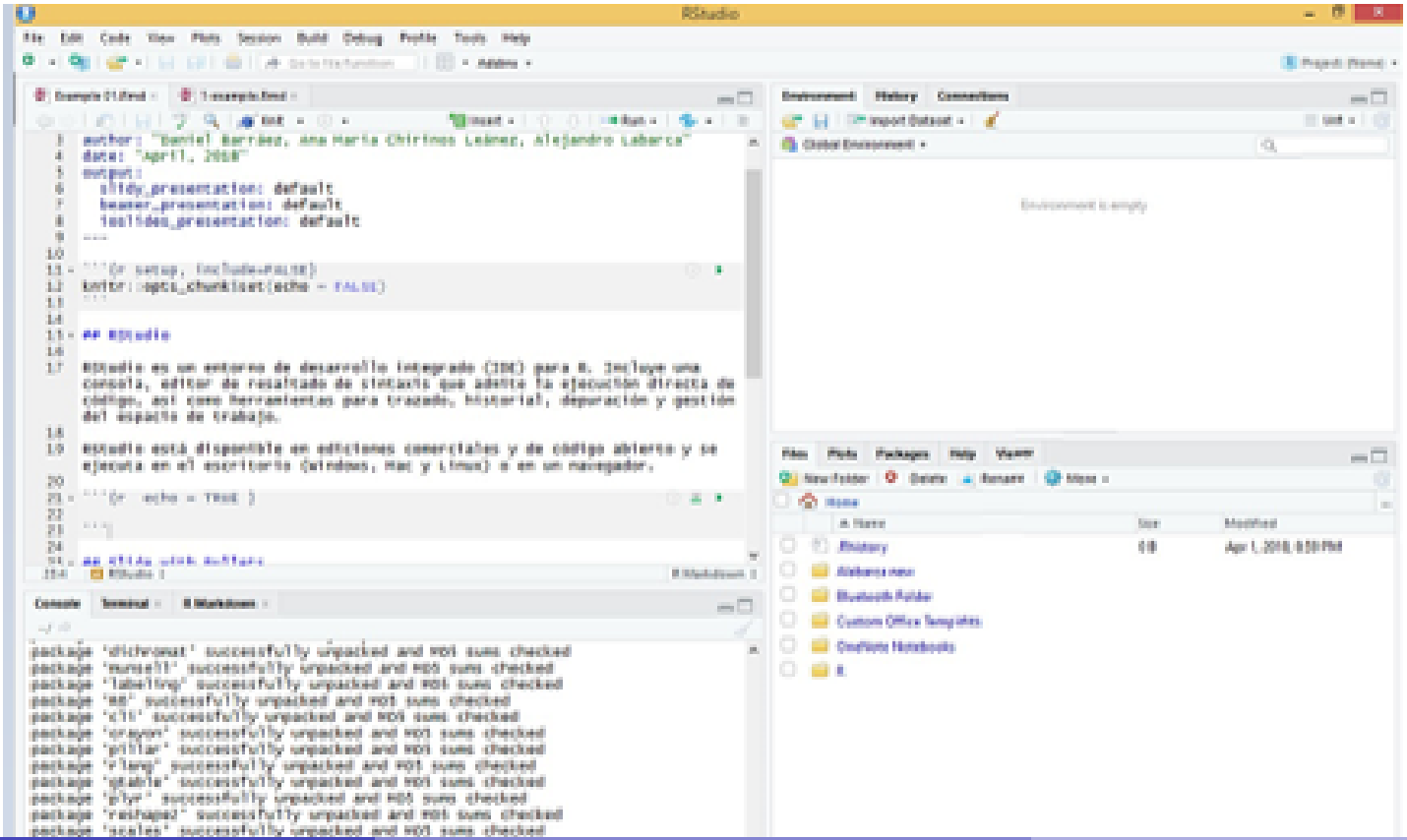
5.- R Internals.

6.- The R Reference Index.

Foros, blogs, grupos, seminarios: #rstats hashtag, R-Ladie, Rweekly, R-bloggers, DataCarpentry y Software Carpentry, Stack Overflow, R Conferences, Github, The R Consortium.

RStudio

RStudio es un entorno de desarrollo integrado (IDE) para R. Incluye una consola, editor de resaltado de sintaxis que admite la ejecución directa de código, así como herramientas para trazado, historial, depuración y gestión del espacio de trabajo.



Consideraciones previas

Para instalar los programas, es necesario comenzar con R y luego RStudio. Debemos descargar las últimas versiones en archivos ejecutables que proporcionan las páginas oficiales, a través de los siguientes enlaces: https://www.r-project.org/ ; https://www.rstudio.com/

Luego de la instalación, es necesario comprender algunos requerimientos básicos para comenzar a programar:

1. En la consola de R el símbolo '>', indica que R está listo para recibir un comando.
2. R es un lenguaje Orientado a Objetos, siendo la sintáxis muy simple e intuitiva. Ejemplo (regresión lineal): lm(y~x).
3. El nombre de un objeto debe comenzar con una letra (A-Z and a-z) y puede incluir letras,dígitos (0-9), y puntos (.). R discrimina entre letras mayúsculas y minúsculas para el nombre de un objeto, de tal manera que x y X se refiere a objetos diferentes (inclusive bajo Windows).

4. El usuario puede modificar o manipular estos objetos con operadores (aritméticos, lógicos, y comparativos) y funciones (que a su vez son objetos)
5. Las funciones disponibles estan guardadas en una librería localizada en el directorio R HOME/library (R HOME es el directorio donde R esta instalado).
6. El paquete (conjunto de funciones) denominado base constituye el nucleo de R y contiene las funciones básicas del lenguaje para leer y manipular datos, algunas funciones gráficas y algunas funciones estadísticas (regresión lineal y análisis de varianza)
7. Para que una función sea ejecutada en R debe estar siempre acompañada de paréntesis, inclusive en el caso que no haya nada dentro de los mismos (por ej., ls()).

Comandos básicos

Creación, listado y remoción de objetos en memoria

1. El comando más simple es escribir el nombre de un objeto para visualizar su contenido. Por ejemplo, si un objeto n contiene el valor 10:

```
n<-10
n
```

```
## [1] 10
```

El dígito 1 indica que la visualizacion del objeto comienza con el primer elemento de n. Este comando constituye un uso implícito de la función print, y el ejemplo anterior es similar a print(n).

2. Un objeto puede ser creado con el operador "asignar" el cual se denota como una flecha con el signo menos y el símbolo "$>$" o "$<$" dependiendo de la dirección en que asigna el objeto:

```
n<-15
n
```

```
## [1] 15
```

```
5->n
n
```

```
## [1] 5
```

```
x<-1
X<-10
x
```

```
## [1] 1
```

Si el objeto ya existe, su valor anterior es borrado después de la asignación (la modificación afecta solo objetos en memoria, no a los datos en el disco). El valor asignado de esta manera puede ser el resultado de una operación y/o de una función:

```
n<- 10 + 2
n
```

```
## [1] 12
```

```
n<-3 + rnorm(1)
n
```

```
## [1] 3.311168
```

La función rnorm(1) genera un dato aleatorio cuya distribución de probabilidad es una normal con media 0 y varianza 1.

3. La función ls lista los objetos en memoria: sólo arroja los nombres de éstos.

```
name<-"Carmen"; n1<-10; n2<-100; m<-0.5
ls()
```

```
## [1] "m"     "n"     "n1"    "n2"    "name" "x"     "X"
```

4. Note el uso del punto y coma para separar comandos diferentes en la misma línea. Si se quiere listar solo aquellos objetos que contengan un caracter en particular, se puede usar la opción pattern (que se puede abreviar como pat):

```
ls(pat="m")
```

```
## [1] "m"     "name"
```

5. La función ls.str() muestra algunos detalles de los objetos en memoria:

```
ls.str()
```

```
## m :  num 0.5
## n :  num 3.31
## n1 :  num 10
## n2 :  num 100
## name :  chr "Carmen"
## x :  num 1
## X :  num 10
```

La opción pattern se puede usar de la misma manera con ls.str(). Otra opción útil en esta función es max.level la cual especifica el nivel de detalle para la visualización de objetos compuestos. Por defecto, ls.str() muestra todos los detalles de los objetos en memoria, incluyendo las columnas de los marcos de datos ("data frames"), matrices y listas.

```r
M<-data.frame(n1,n2,m)
ls.str(pat="M")
```

```
## M : 'data.frame':    1 obs. of  3 variables:
##  $ n1: num 10
##  $ n2: num 100
##  $ m : num 0.5
```

6. Para borrar objetos en memoria, utilizamos la función rm(): rm(x) elimina el objeto x, rm(x,y) elimina ambos objetos x y y, y rm(list=ls()) elimina todos los objetos en memoria; las mismas opciones mencionadas para la función ls() se pueden usar para borrar selectivamente algunos objetos: rm(list=ls(pat="^m"))

La ayuda en línea

Proporciona información muy útil de cómo utilizar las funciones. La ayuda se encuentra disponible directamente para una función dada. Por ejemplo: ?lm o help(lm) mostrará dentro de R, ayuda para la función lm() (modelo lineal)

Manipulaciones simples: Números y vectores

Vectores. Asignación

R tiene cinco clases básicas o "atomic" para objetos:

- character
- numeric (real numbers)
- integer
- complex
- logical (True/False)

El tipo de objeto más simple en R es un vector. En realidad, hay una sola regla sobre vectores en R: Un vector solo puede contener objetos de la misma clase. La excepción a la regla previa la contienen las listas, que veremos un poco más adelante. Una lista se representa como un vector, pero puede contener objetos de diferentes clases. De hecho, esa es la razón por la que los usamos!

La función c() se puede usar para crear vectores de objetos mediante la concatenación de elementos.

```
x <- c(0.5, 0.6)        ## numérico
x <- c(TRUE, FALSE)     ## lógico
x <- c(T, F)            ## lógico
x <- c("a", "b", "c")   ## caracter
x <- c(1+0i, 2+4i)      ##complejo
x <- 4:9               ## entero (secuencia de números)
```

Con la siguiente asignación, crea un vector con 11 elementos consistentes en dos copias de x con un cero entre ambas.

```
y <- c(x, 0, x)
```

El comando as.factor() es útil para "etiquetar" los elementos de un objeto. Un comando que permite visualizar los elementos de un objeto, excluyendo las repeticiones, es a través del comando unique()

Si queremos extraer elementos de un vector utilizamos [ ]:

```
y[1:3]
```

```
## [1] 4 5 6
```

Generación de sucesiones

Si queremos que la secuencia de números se realice cada 2 pasos, utilizamos el comando seq():

```
seq(from=1,to=15,by=2)
```

```
## [1]  1  3  5  7  9 11 13 15
```

Para mostrar vectores cuyas entradas son iguales, se puede utilizar el comando rep(). Es útil, para reducir tiempo de programación.

```
rep(x=5,times=15)
```

```
##  [1] 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
```

Mezcla de objetos

Cuando diferentes objetos se mezclan en un vector, la coerción ocurre de modo que cada elemento en el vector es de la misma clase:

```
c(1.7, "a")    ## caracter
```

```
## [1] "1.7" "a"
```

```
c(TRUE, 2)     ## numérico
```

```
## [1] 1 2
```

```r
c("a", TRUE)  ## caracter
```

```
## [1] "a"    "TRUE"
```

Los objetos pueden ser forzados explícitamente de una clase a otra usando las funciones as. *, Si están disponibles.

```r
x <- 0:6
class(x)
```

```
## [1] "integer"
```

```r
as.numeric(x)
```

```
## [1] 0 1 2 3 4 5 6
```

```r
as.logical(x)
```

```
## [1] FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
```

```r
as.character(x)
```

```
## [1] "0" "1" "2" "3" "4" "5" "6"
```

Aritmética vectorial

- Los vectores pueden usarse en expresiones aritméticas, en cuyo caso las operaciones se realizan elemento a elemento.
- Los operadores aritméticos elementales son los habituales +, -, *, / y ^ para elevar a una potencia. Además están disponibles las funciones log, exp, sin, cos, tan, sqrt, bien conocidas.

Objetos: modos y atributos

En general, los objetos R pueden tener "attributes", que son como metadatos para el objeto. Estos metadatos pueden ser muy útiles ya que ayudan a describir el objeto. Por ejemplo, los nombres de una columna en un marco de datos ayudan a decirnos qué datos están contenidos en cada una de las columnas. Algunos ejemplos de atributos de objetos R son:

- names, dimnames
- dimensions (por ejemplo, matrices, arreglos)
- class (por ejemplo, entero, numérico)
- length
- Otros atributos / metadatos definidos por el usuario

Se puede acceder a los atributos de un objeto (si corresponde) utilizando la función attributes(). No todos los objetos R contienen atributos, en cuyo caso la función attributes() devuelve NULL.

Ejercicios

1. Construir un vector de clase caracter compuesto por los siguientes elementos : 4,5,6,7,. . . ,90,92,94,96,98,100,1,4,6,“a”,“ab”,“abc”. Contar el número de elementos menores a 13. Sugerencia: Utilizar el operador lógico $<$ ó $>$ y la función sum()
2. Considerar el vector de tipo numérico compuesto por los siguientes elementos: 34,56,55,87,NA,4,77,NA,21,NA,21. (i) Contar el número de elementos iguales a NA y eliminarlos. (ii) Determinar cuántos niveles “etiquetas” tiene el vector. Sugerencia: Utilizar la función is.na() en (i).

Matrices y arreglos

Matrices

Las matrices son vectores que posee la dimensión de un vector de tamaño 2 como atributo, el cual especifica el número de filas y columnas.

```
m <- matrix(nrow = 2, ncol = 3)
m
```

```
##      [,1] [,2] [,3]
## [1,]   NA   NA   NA
## [2,]   NA   NA   NA
```

```
m <- matrix(1:6,nrow=2,ncol=3)
m
```

```
##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    6
```

Pueden ser creadas a través de funciones alternativas:

- Vectores, añadiendo la dimensión atributo.

```
m <- 1:10
m
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

```
dim(m)<-c(2,5)
m
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    3    5    7    9
## [2,]    2    4    6    8   10
```

- Enlace de columnas y filas: funciones cbind() y rbind()

```
x <- 1:3
y <- 10:12
cbind(x, y)
```

```
##      x  y
## [1,] 1 10
## [2,] 2 11
## [3,] 3 12
```

```
rbind(x, y)
```

```
##   [,1] [,2] [,3]
## x    1    2    3
## y   10   11   12
```

Arreglos

Un arreglo es una colección de datos indexada por varios índices. Las matrices, son un caso particular de esta colección de datos.En R, hay dos maneras de crear arreglos:

- Definiendo un vector y luego estableciendo las dimensiones a través del comando dim(),

```
z<-1:24
dim(z)<-c(3,4,2)
z
```

```
## , , 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
```

- Utilizando el comando array(),

```
array(z,dim = c(3,4,2))
```

```
## , , 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
```

Ejercicios

3. Crear un vector x,y,z de tipo entero de tres elementos cada uno. (i) Combinar los tres vectores en una matriz 3x3, que llamaremos A, donde cada columna es un vector. (ii) Cambiar el nombre de las columnas por a,b,c. (iii) Visualizar la segunda fila y el elemento A_{3X2}. Sugerencia: En (ii) utilizar el comando dimnames()[[2]].
4. Crear un vector con 12 enteros. Convertir el vector en una matriz 4x3, denominada B usando el comando matrix(). (i) Cambiar el nombre de las columnas por x,y,z y las filas por a,b,c,d. (ii) Realizar la transpuestas de B y calcular la multiplicación de matrices t(B)xB. (iii) Extraer los elementos distintos de cero. Sugerencia: utilizar el comando dimnames() en (i), el operador aritmético %*% en (ii), el operador lógico != en (iii).

Listas y data frames

En R, una lista es un objeto cuyos elementos son una colección ordenada de objetos, conocidos como componentes. No es necesario que los componentes sean del mismo modo.El siguiente es un ejemplo de una lista:

```
Lst<-list(nombre="Pedro", esposa="María",no.hijos=3)
Lst
```

```
## $nombre
## [1] "Pedro"
##
## $esposa
## [1] "María"
##
## $no.hijos
## [1] 3
```

- Los componentes siempre están numerados y pueden ser referidos por dicho número. En este ejemplo, Lst es el nombre de una lista con cuatro componentes, cada uno de los cuales puede ser referido, respectivamente, por Lst[[1]], Lst[[2]], Lst[[3]] y Lst[[4]]. Como, además, Lst[[4]] es un vector, Lst[[4]][1] refiere su primer elemento. La función length() aplicada a una lista devuelve el número de componentes (del primer nivel) de la lista.
- La función length() aplicada a una lista devuelve el número de componentes (del primer nivel) de la lista.
- Los componentes de una lista pueden tener nombre, en cuyo caso pueden ser referidos también por dicho nombre, mediante una expresión de la forma nombre de lista$nombre de componente. También es posible utilizar los nombres de los componentes entre dobles corchetes [[]]

- Es muy importante distinguir claramente entre Lst[[1]] y Lst[1]. '[[]]' es el operador utilizado para seleccionar un sólo elemento, mientras que '[]' es un operador general de indexado. Esto es, Lst[[1]] es el primer objeto de la lista Lst, y si es una lista con nombres, el nombre no está incluido. Por su parte, Lst[1], es una sublista de la lista Lst consistente en la primera componente. Si la lista tiene nombre, éste se transfiere a la sublista.

Hoja de datos (Data frames)

La hoja de datos se usan para almacenar datos tabulares en R. Son un tipo importante de objeto en R y se usan en una variedad de aplicaciones de modelado estadístico. El paquete dplyr de Hadley Wickham tiene un conjunto optimizado de funciones diseñadas para funcionar eficientemente con hojas de datos, y las funciones de trazado ggplot2 funcionan mejor con datos almacenados en hojas de datos.

- Se representan como un tipo especial de lista donde cada elemento de la lista debe tener la misma longitud. Cada elemento de la lista se puede considerar como una columna y la longitud de cada elemento de la lista es el número de filas.
- A diferencia de las matrices, las hojas de datos pueden almacenar diferentes clases de objetos en cada columna, a diferencia de las matrices que deben tener cada elemento de la misma clase (por ejemplo, todos los enteros o todos los numéricos).
- Puede construir una hoja de datos utilizando la función data.frame():

```
cont <- data.frame(A=cars$speed, B=cars$dist)
```

Puede forzar que las componentes una lista cumplan las restricciones para ser una hoja de datos, mediante la función as.data.frame(). La manera más sencilla de construir una hoja de datos es utilizar la función read.table() o read.csv, para leerla desde un archivo del sistema operativo. Esta forma se tratará más adelante.

Bucles, condicionantes y funciones

Una ventaja de R comparado con otros programas estadísticos con "menus y botones" es la posibilidad de programar de una manera muy sencilla una serie de análisis que se puedan ejecutar de manera sucesiva.

Supongamos que tenemos un vector x, y para cada elemento de x con valor igual a b, queremos asignar el valor 0 a otra variable y, o sino asignarle 1. En R:

```
y <- numeric(length(x))

for (i in 1:length(x)){

if (x[i] == b) { y[i] <- 0}

else{y[i] <- 1}

}
```

Otra posibilidad es ejecutar una instruccion siempre y cuando se cumpla una cierta condición:

while (myfun $>$ minimum) $\{ \ldots \}$

Sin embargo, este tipo de bucles y estructuras se pueden evitar gracias a una característica clave en R: vectorización. La vectorizacion hace los bucles y condicionantes implícitos en las expresiones. En el ejemplo previo basta con ejecutar:

```
y[x == b] <- 0
y[x != b] <- 1
```

Tambien existen varias funciones del tipo "apply" que evitan el uso de bucles. apply() actua sobre las filas o columnas de una matriz, y su sintaxis es apply(X, MARGIN, FUN,...), donde X es una matriz, MARGIN indica si se van a usar las filas (1), las columnas (2), o ambas (c(1, 2)), FUN es una funcion a ser aplicada, y ... son posibles argumentos opcionales de FUN.

```r
x <- rnorm(10, -5, 0.1)
y <- rnorm(10, 5, 2)
# las columnas de X mantienen los nombres "x" y "y"
X <- cbind(x, y)
apply(X, 2, mean)
```

```
##          x          y
## -5.001751  5.085742
```

```r
apply(X, 2, sd)
```

```
##          x          y
## 0.1020764 2.4768365
```

Otras funciones que actúan de manera similar: lapply(), sapply(), mapply()

Para definir una función debe realizar una asignación de la forma

`NombreDeFuncion <-function(arg 1, arg 2, ...) {exp.}`

donde exp., es una expresión de R que utiliza los argumentos, arg i, para calcular un valor que es devuelto por la función. Consideremos una función que calcule el estadístico t de Student para dos muestras:

```
DosMuestras <- function(y1, y2) {
n1 <- length(y1); n2 <- length(y2)
yb1 <- mean(y1); yb2 <- mean(y2)
s1 <- var(y1); s2 <- var(y2)
s <- ((n1-1)*s1 + (n2-1)*s2)/(n1+n2-2)
tst <- (yb1 - yb2)/sqrt(s2*(1/n1 + 1/n2))
gl<-(((s1/n1)+(s2/n2))^2)/((((s1/n1)^2)+((s2/n2)^2))/(n2-1))
pvalor=dt(tst,df=gl)
M=matrix(nrow=1,ncol=3,data = c(tst,gl,pvalor))
dimnames(M) = list(c("test-t"),c("t-statistic","gl","p-value"))
return(M)
}
```

Una vez definida esta función, puede utilizarse para realizar un contraste de t de Student para dos muestras, del siguiente modo:

```
tStudent <- DosMuestras(sleep[sleep$group==1,]$extra,
sleep[sleep$group==2,]$extra)
tStudent
```

```
##         t-statistic       gl    p-value
## test-t    -1.76451 17.77647 0.08644974
```

Importación y exportación de datos

R brinda una amplia variedad de comandos para trabajar con información que se encuentra en distintos tipos de formatos, permitiendo un alto grado de flexibilidad al momento de realizar análisis específicos con la herramienta. Con el siguiente link descargamos la data VE_INE.csv: https://github.com/alabarca/Curso-R/upload/master

1. .txt/.csv:

   - package base: Si utilizamos la instalación recomendada en R, este paquete ya viene incluido. Comandos:

```
read.table(file = "VE_INE.csv",sep=",",header=TRUE)#importar archivo

write.table(VEN_INE,file="VE_INE2.csv",sep=",",dec=".")
#exportar archivo
```

- package readr: es una buena idea especificar los tipos de columnas explícitamente. Esto descarta cualquier posible error de adivinación por parte de read.table.:

```
read_csv(file = "VE_INE.csv",col_types="ccn",nmax=10)
```

2. .xlsx:

- package readxl: es un paquete diseñado para hacer una sola tarea, importar hojas de Excel a R. Esto hace que sea un paquete ligero y eficiente, a cambio de no contar con funciones avanzadas.

Es compatible con hojas de cálculo de Excel 97-03, con extensión .xls, y con hojas de cálculo de las versiones más recientes de Excel, con extensión .xlsx. Si una celda de una pestaña contiene una fórmula, se importa es el resultado de esa fórmula.

Antes de empezar, necesitamos conocer el contenido de nuestra hoja de cálculo. Podemos usar la función excel_sheets() para conocer qué pestañas contiene nuestra hoja de cálculo sin salir de R

```
excel_sheets("VE_INE.xlsx")
```

```
## [1] "VE_INE"
```

Usaremos la función read_excel() indicando la ruta del documento que queremos importar a nuestro espacio de trabajo de R, sin parámetros adicionales.Asignaremos el resultado de esta función al objeto excel_INE

```
#excel_INE <- read_excel("VE_INE.xlsx",n_max=100)
```

Es posible explorar el objeto a través de los comandos str(), summary(), head(), tail().

Estadística descriptiva con R

Nombre Comando	Explicación
summary(*data*)	Resumen estadístico
min(*data*)	Mínimo
max(*data*)	Máximo
range(*data*)	Rango
mean(*data*)	Media aritmética
median(*data*)	Mediana
length(*data*)	Tamaño
sd(*data*)	Desviación típica
var(*data*), cov(*data*)	Varianza
cor(*data*)	Correlación
quantile(*data*, 0.25)	Cuantil Q1
quantile(*data*, 0.75)	Cuantil Q3
IQR(*data*)	Rango intercuartílico
sort(*data*)	Ordenar
table(*data*)	Tabla de frecuencias absolutas

Figura 4:

Procedimientos gráficos

Las posibilidades gráficas son un componente de R muy importante y versátil. Es posible utilizarlas para mostrar una amplia variedad de gráficos estadísticos y también para construir nuevos tipos de gráficos. Utilizaremos las herramientas gráficas contenidas en el package base, y daremos unos ejemplos del alcance que posee el paquete ggplot2, en lo que algunos usuarios de la comunidad han denominado una "revolución gráfica".

Las órdenes gráficas se dividen en tres grupos básicos:

- Alto nivel: Son funciones que crean un nuevo gráfico, posiblemente con ejes, etiquetas, títulos, etc..
- Bajo nivel: Son funciones que añaden información a un gráfico existente, tales como puntos adicionales, líneas y etiquetas.
- Interactivas: Son funciones que permiten interactuar con un gráfico, añadiendo o eliminando información, utilizando un dispositivo apuntador, como un ratón.

Funciones gráficas de nivel alto

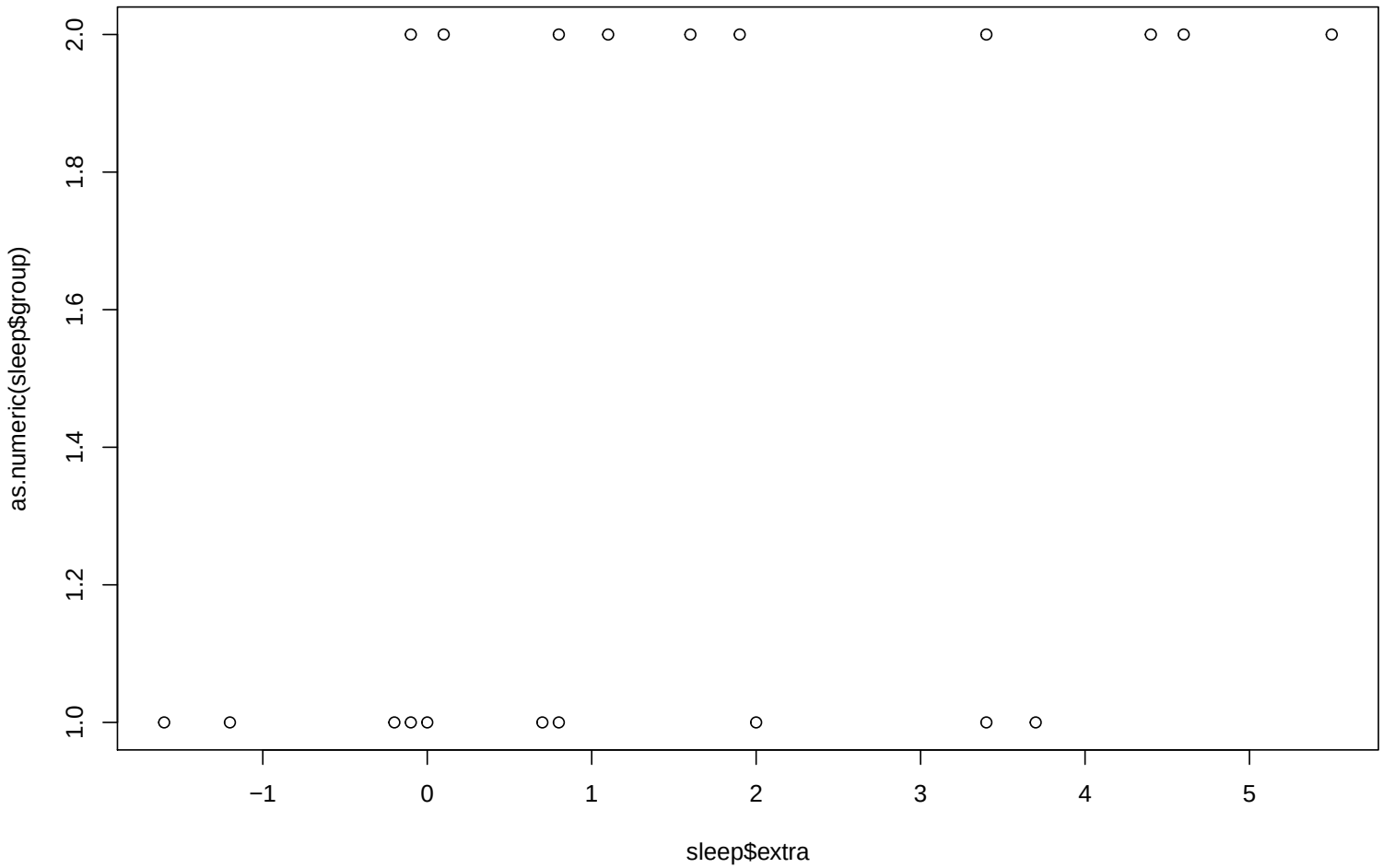
Las órdenes gráficas de nivel alto están diseñadas para generar un gráfico completo a partir de unos datos pasados a la función como argumento. Cuando es necesario se generan automáticamente ejes, etiquetas o títulos (salvo que se especifique lo contrario). Estas órdenes comienzan siempre un nuevo gráfico, borrando el actual si ello es necesario.

1. La función *plot*

```
#plot(x, y)
#plot(xy)
```

Si x e y son vectores, *plot(x, y)* produce un diagrama de dispersión de y sobre x. El mismo efecto se consigue suministrando un único argumento (como se ha hecho en la segunda forma) que sea bien una lista con dos elementos, x e y, bien una matriz con dos columnas.

```r
plot(sleep$extra,as.numeric(sleep$group))
```



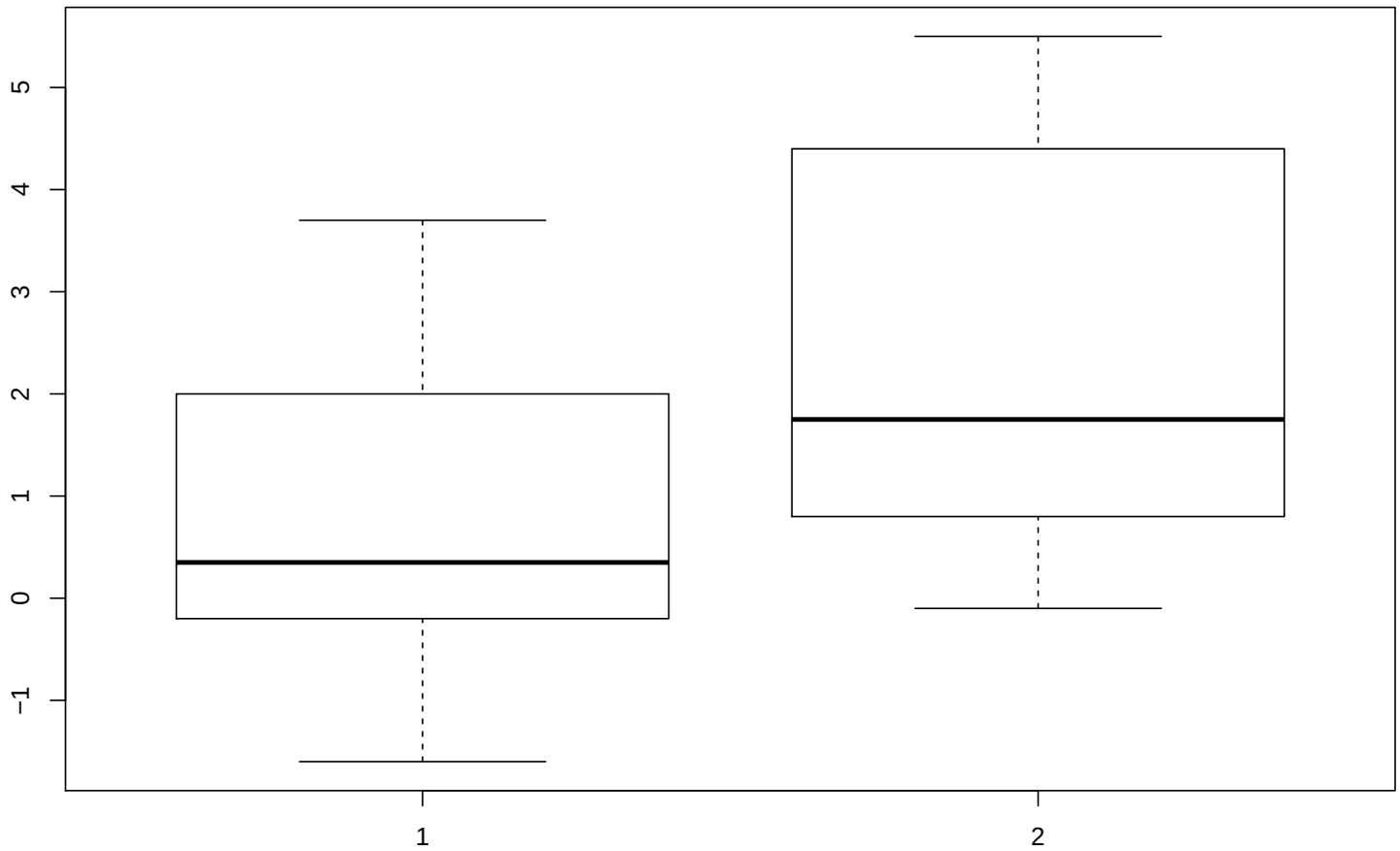
```
#plot(x)
```

Si x es una serie temporal, produce un gráfico temporal, si x es un vector numérico, produce un gráfico de sus elementos sobre el índice de los mismos, y si x es un vector complejo, produce un gráfico de la parte imaginaria sobre la real de los elementos del vector.

```
#plot(f)
#plot(f, y)
```

Sean f un factor e y un vector numérico. La primera forma genera un diagrama de barras de f; la segunda genera diagramas de cajas de y para cada nivel de f.

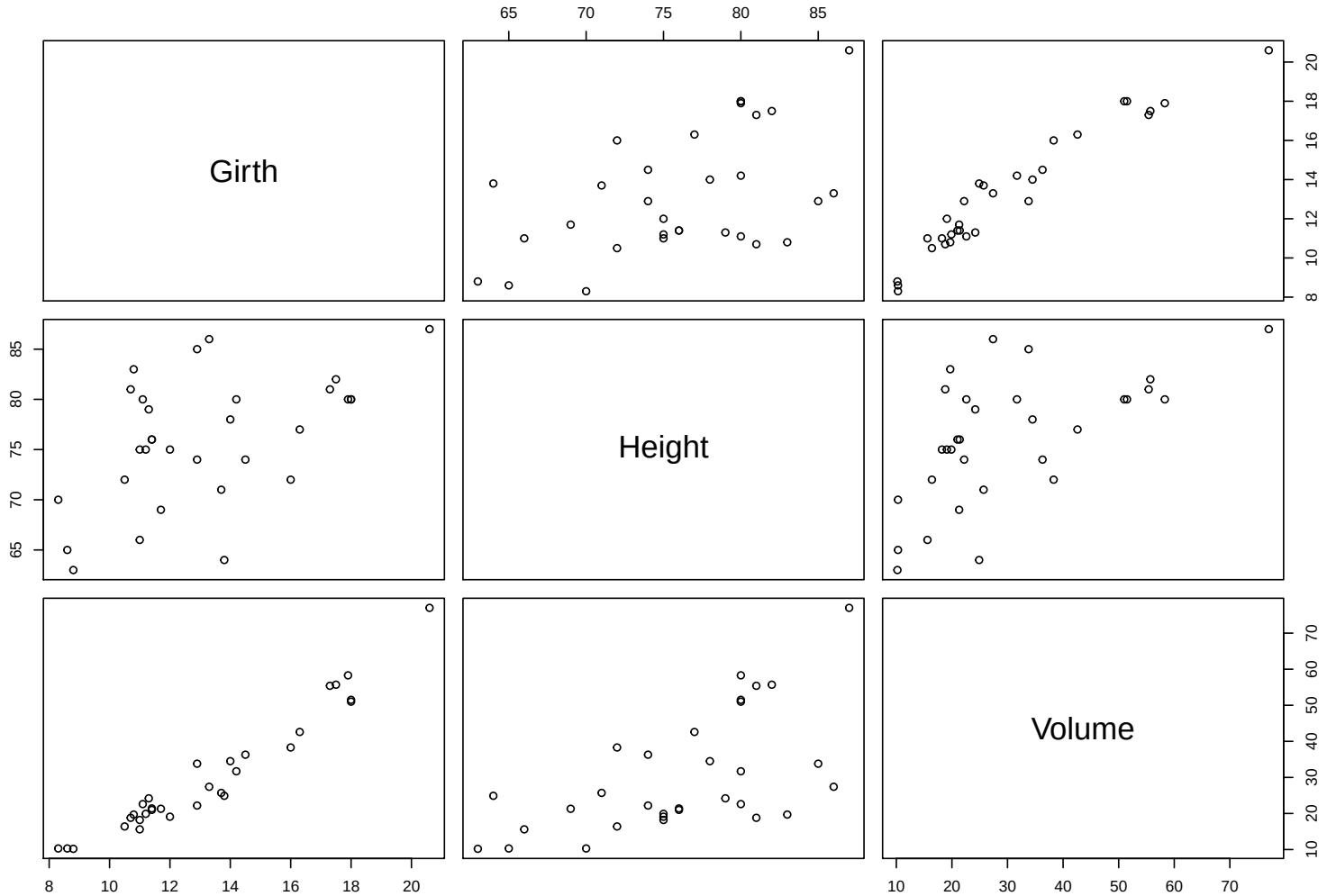
```r
plot(sleep$group,sleep$extra)
```



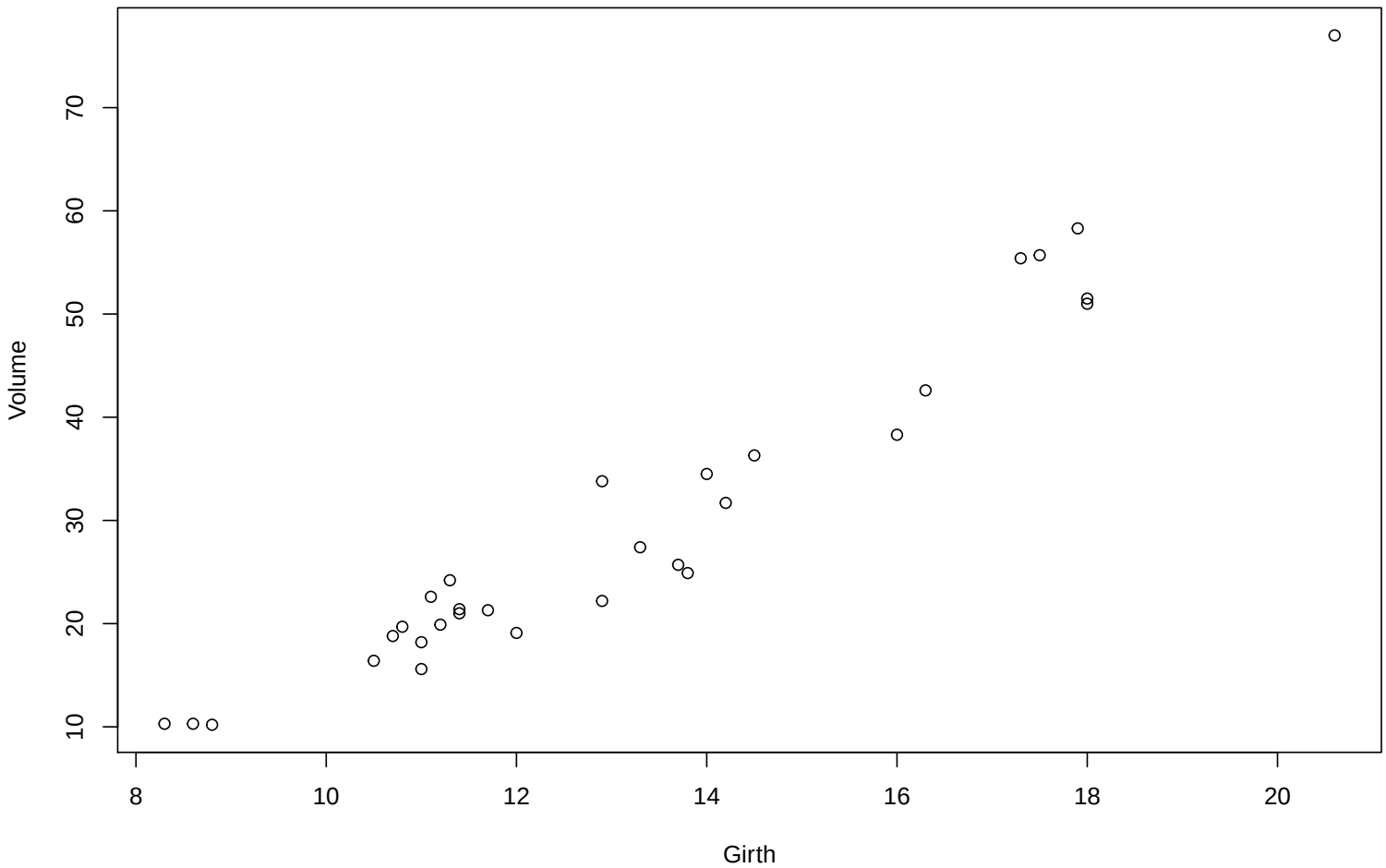
```
#plot(hd)
#plot(~ expr)
```

Sea hd una hoja de datos; y, un objeto cualquiera; y expr, una lista de nombres de objetos separados por símbolos '+' (por ejemplo, a + b + c). Las dos primeras formas producen diagramas de todas las parejas de variables de la hoja de datos hd (en el primer caso) y de los objetos de la expresión expr (en el segundo caso).

```
plot(trees)
```


Girth
Height
Volume
65
70
75
80
85
8
10
12
14
16
18
20
65
70
75
80
85
10
20
30
40
50
60
70

```r
plot(~Girth+Volume,data=trees)
```



2 Representación de datos multivariantes

R posee dos funciones muy útiles para representar datos multivariantes. Si X es una matriz numérica o una hoja de datos, la orden

```
#pairs(X)
```

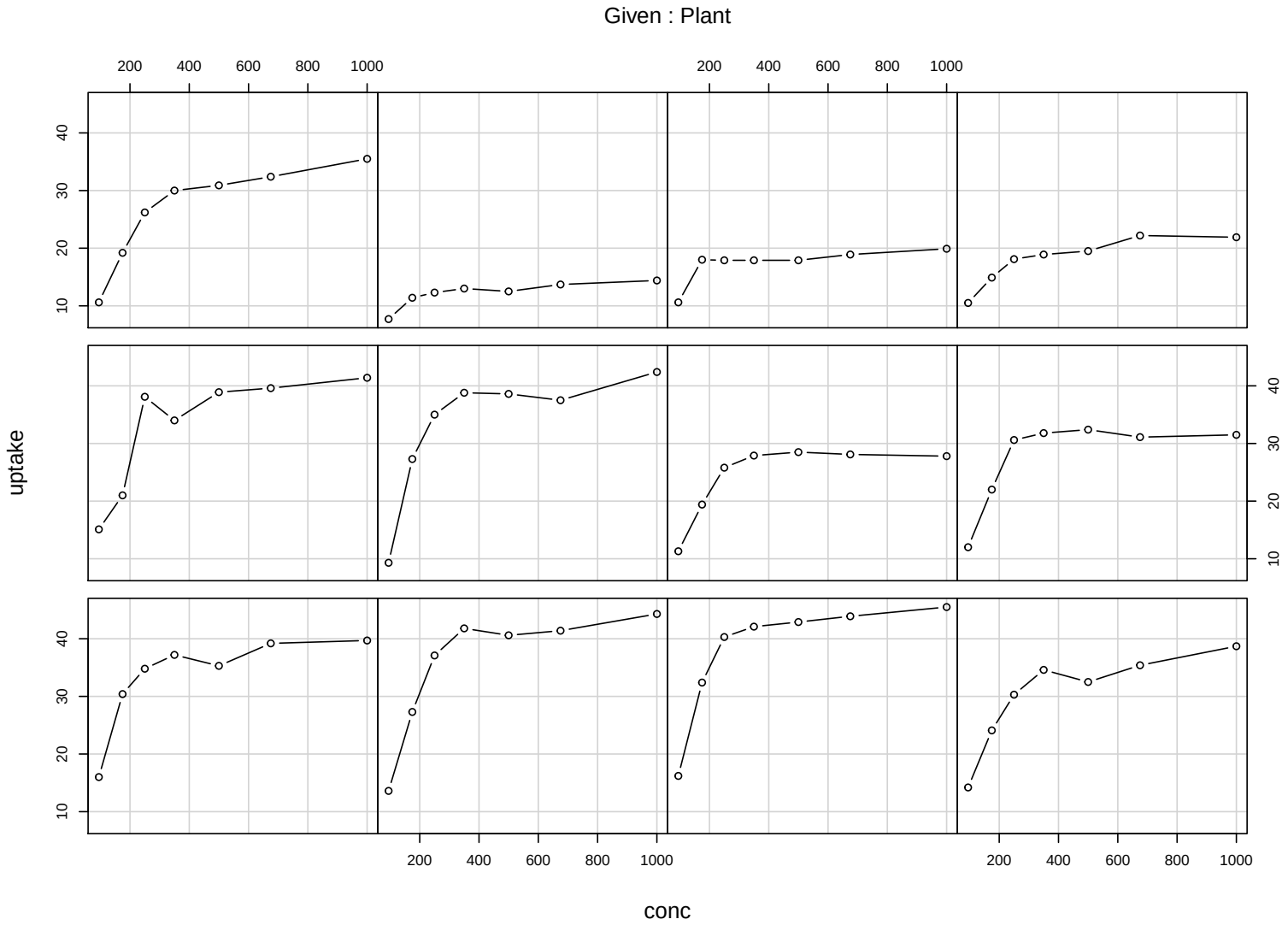
produce una matriz de gráficos de dispersión para cada pareja de variables definidas por las columnas de X, esto es, cada columna de X se representa frente a cada una de las restantes columnas, y los gráficos se presentan en una matriz con escalas constantes sobre las filas y columnas.

Cuando se trabaja con tres o cuatro variables, la función coplot() puede ser más apropiada. Si a y b son vectores numéricos y c es un vector numérico o un factor (todos de la misma longitud) entonces la orden

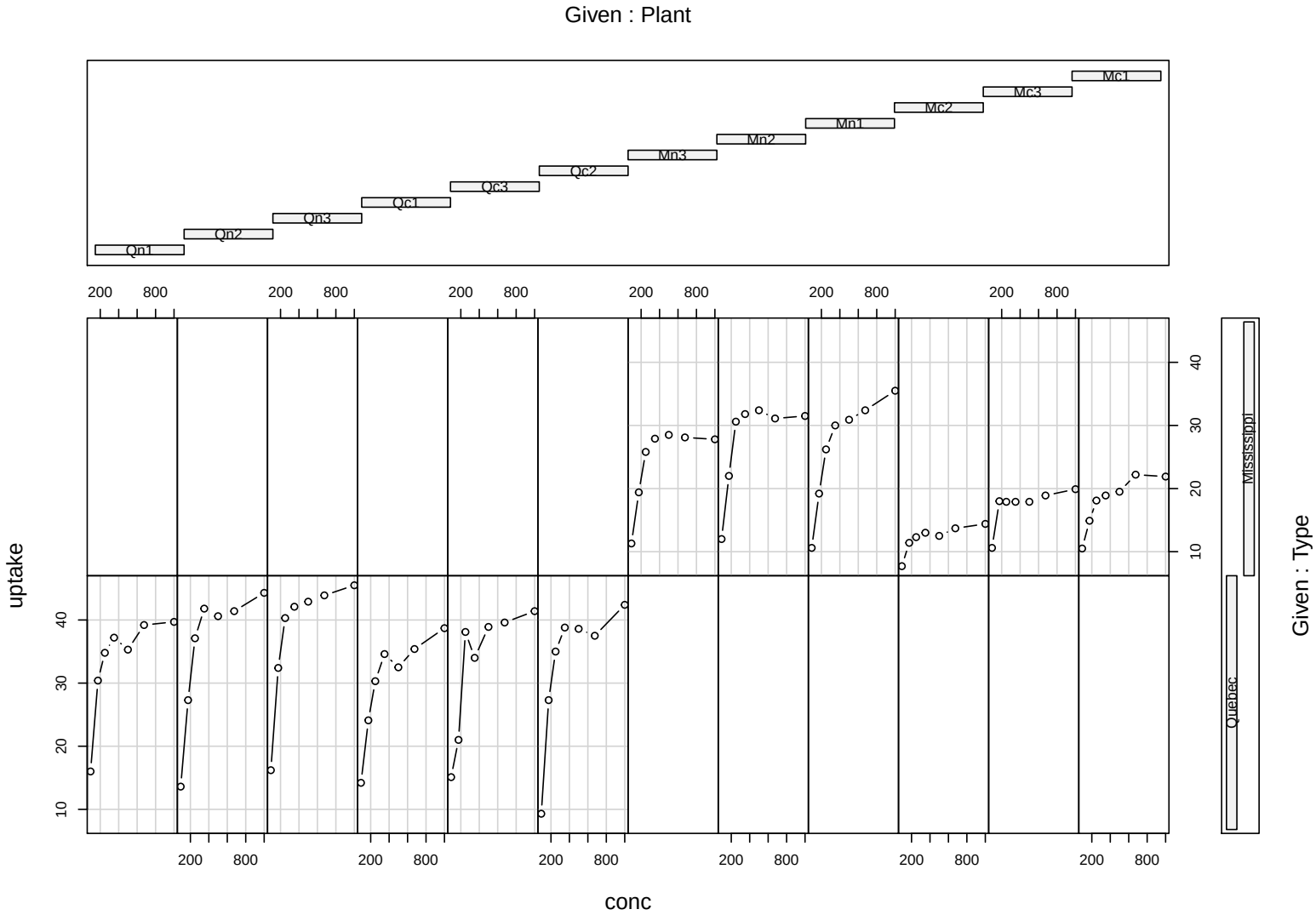
```
#coplot(a ~ b | c)
```

produce diagramas de dispersión de a sobre b para cada valor de c.

```
coplot(uptake ~ conc | Plant, data = CO2, show.given = FALSE
, type = "b")
```



```
coplot(uptake ~ conc | Plant + Type, data = CO2,type="b")
```



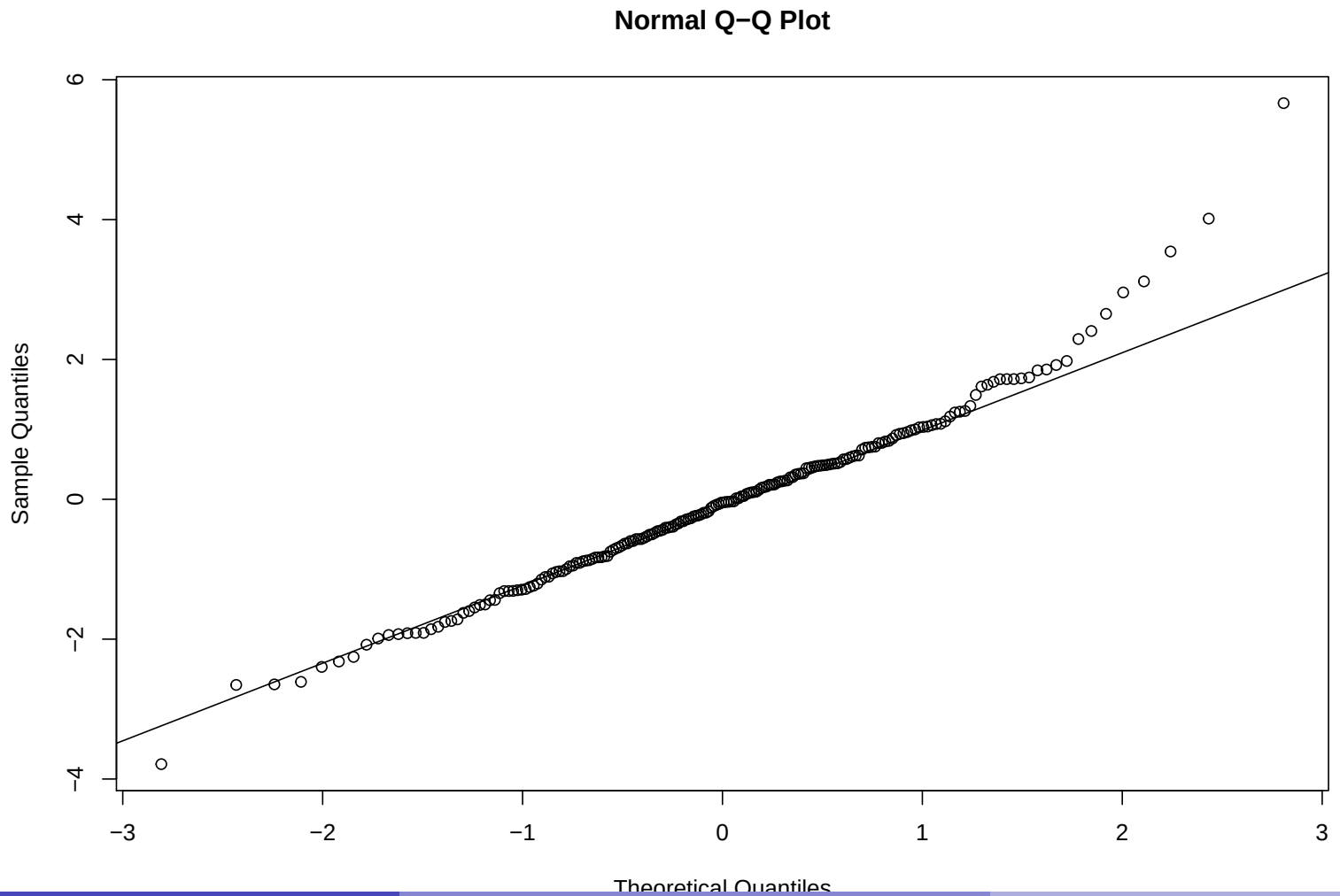
3 Otras representaciones gráficas

Existen otras funciones gráficas de nivel alto que producen otros tipos de gráficos. Algunas de ellas son las siguientes:

```
#qqnorm(x)
#qqline(x)
#qqplot(x, y)
```

Gráficos de comparación de distribuciones. El primero representa el vector x sobre los valores esperados normales. El segundo le añade una recta que pasa por los cuartiles de la distribución y de los datos. El tercero representa los cuantiles de x sobre los de y para comparar sus distribuciones respectivas.

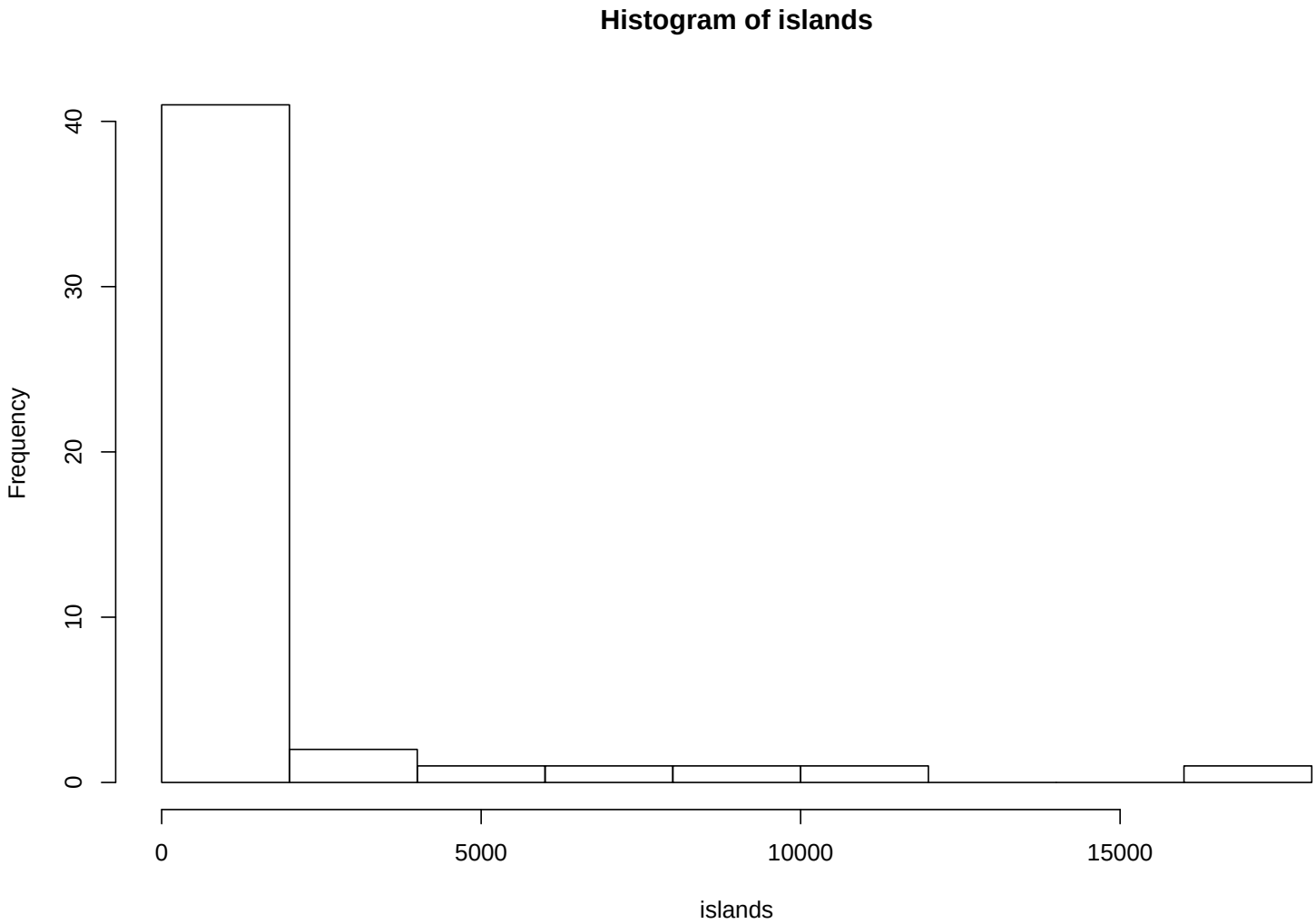
```r
y<-rt(200,df=5)
qqnorm(y)
qqline(y)
```



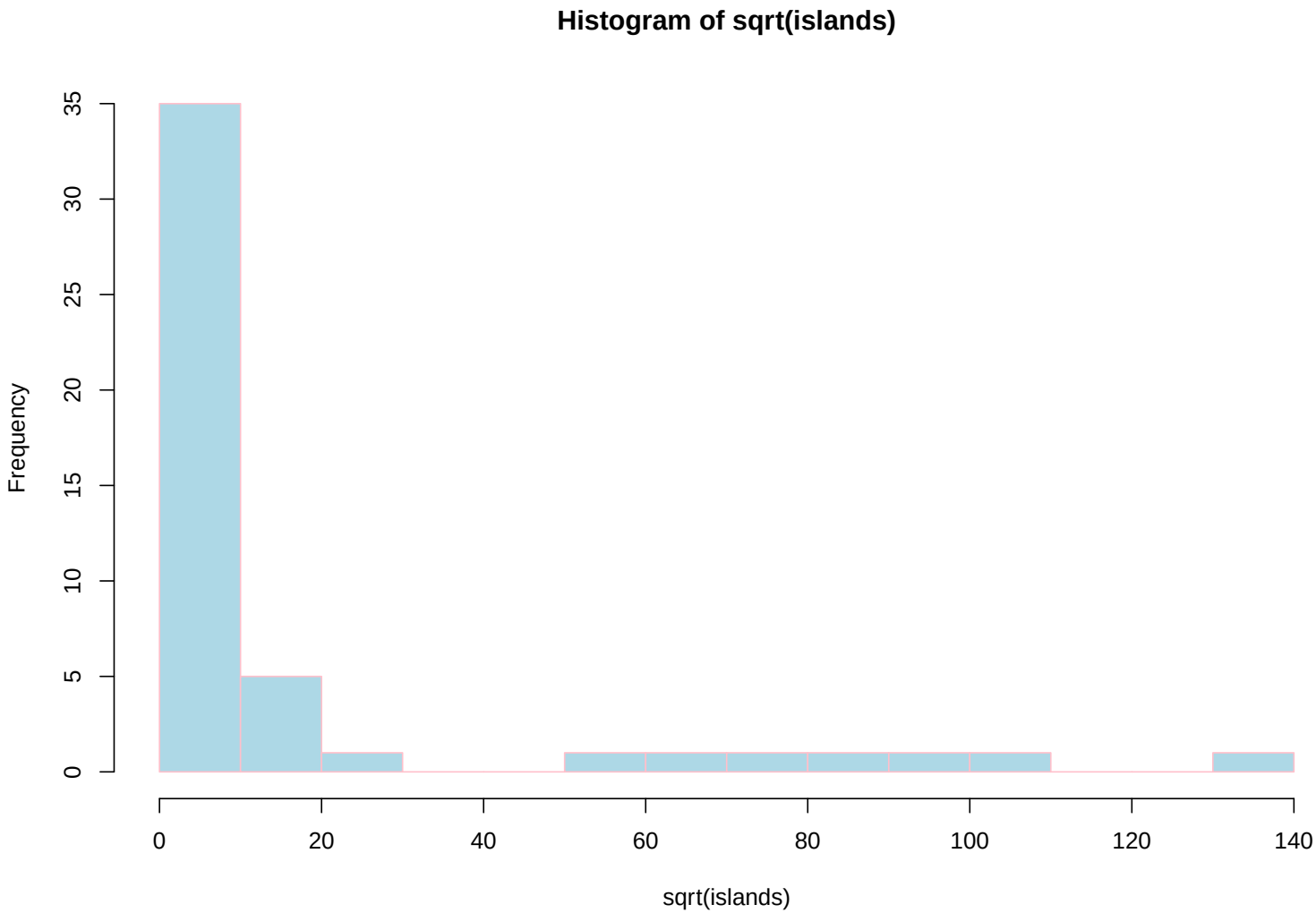
```
#hist(x)
#hist(x, nclass=n)
#hist(x, breaks=b, ...)
```

Produce un histograma del vector numérico x. El número de clases se calcula habitualmente de modo correcto, pero puede elegir uno con el argumento nclass, o bien especificar los puntos de corte con el argumento breaks. Si está presente el argumento probability=TRUE, se representan frecuencias relativas en vez de absolutas.

```r
hist(islands)
```



```
hist(sqrt(islands), breaks = 12, col = "lightblue",
border = "pink")
```



4. Argumentos de las funciones gráficas de nivel alto

Existen una serie de argumentos que pueden pasarse a las funciones gráficas de nivel alto, entre los que se destacan:

- add=TRUE, Obliga a la función a comportarse como una función a nivel bajo, de modo que el gráfico que genere se superpondría al gráfico actual, en vez de borrarlo (sólo en algunas funciones)
- axes=FALSE, Suprime la generación de ejes. Util para crear ejes personalizados con la función axis. El valor predeterminado es axes=TRUE, que incluye los ejes.
- type= , este argumento controla el tipo de gráfico producido, de acuerdo a las siguientes posibilidades:
- type="p" Dibuja puntos individuales. Este es el valor predeterminado
- type="l" Dibuja líneas
- type="b" Dibuja puntos y líneas que los unen
- type="o" Dibuja puntos y líneas que los unen, cubriéndolos
- type="h" Dibuja líneas verticales desde cada punto al eje X

- xlab=cadena ; ylab=cadena, Definen las etiquetas de los ejes x e y, en vez de utilizar las etiquetas predeterminadas, que normalmente son los nombres de los objetos utilizados en la llamada a la función gráfica de nivel alto.
- main=cadena, título del gráfico, aparece en la parte superior con tamaño de letra grande.

Funciones gráficas de nivel bajo

A veces las funciones gráficas de nivel alto no producen exactamente el tipo de gráfico deseado. En este caso pueden añadirse funciones gráficas de nivel bajo para añadir información adicional (tal como puntos, líneas o texto) al gráfico actual. Algunas de las funciones gráficas de nivel bajo más usuales son:

- points(x, y) ; lines(x, y) , añaden puntos o líneas conectadas al gráfico actual.

- text(x, y, etiquetas, . . .) añade texto al gráfico en las coordenadas x, y.
- abline(a, b) ; abline(h=y) ; abline(v=x) , añade al gráfico actual, una recta de pendiente b y ordenada en el origen a. La forma h=y representa una recta horizontal de altura y, y la forma v=x, una similar, vertical.

Uso de parámetros gráficos

- Al crear gráficos, especialmente con fines de presentación o publicación, es posible que R no produzca de modo automático la apariencia exacta que se desea. Ahora bien, es posible personalizar cada aspecto del gráfico utilizando parámetros gráficos. R dispone de muchos parámetros gráficos que controlan aspectos tales como estilo de línea, colores, disposición de las figuras y justificación del texto entre otros muchos. Cada parámetro gráfico tiene un nombre (por ejemplo, 'col', que controla los colores)

- Los parámetros gráficos pueden indicarse de dos modos; bien de modo permanente a través del comando par(), lo que afectará a todas las funciones gráficas que accedan al dispositivo gráfico, bien temporalmente, indicando los argumentos dentro de la función de nivel alto utilizada.

Modelo lineal univariado y multivariado

En este apartado, suponemos al lector familiarizado con la terminología estadística, en particular con el análisis de regresión y el análisis de varianza.

R contiene un conjunto de posibilidades que hace que el ajuste de modelos estadísticos sea muy simple. La salida básica es mínima, y es necesario utilizar las funciones extractoras para obtener todos los detalles.

Definición de modelos estadísticos. Fórmulas

El ejemplo básico de un modelo estadístico es un modelo de regresión lineal con errores independientes y homoscedásticos.

$$y_i = \sum_{j=0}^{p} \beta_j x_{ij} + \epsilon_i$$

$\epsilon \sim NID(0, \sigma^2), i = 1, ..., n$

La función primaria para el ajuste de modelos múltiples ordinarios es lm() y una versión resumida de su uso es la siguiente:

```
#modelo.ajustado <- lm(formula.de.modelo, data=hoja.de.datos)
```

Por ejemplo

```
#fm2 <- lm(y ~ x1 + x2, data=produccion)
```

ajustará un modelo de regresión múltiple de y sobre x1 y x2 (con término independiente)

El término data=produccion, pese a ser opcional, es importante y especifica que cualquier variable necesaria para la construcción del modelo debe provenir en primer lugar de la hoja de datos produccion.

El operador $\sim$ se utiliza para definir una "fórmula de modelo" en R. Los operadores de fórmula son similares a la notación de Wilkinson y Rogers utilizada en los programas Glim y Genstat. Un resumen de la notación básica:

- $Y \sim M$, Y se modeliza como M
- $M_1 + M_2$, incluye M_1 y M_2
- $M_1 - M_2$, incluye M_1 exceptuando los términos de M_2

El valor de lm() es el objeto modelo ajustado; que consiste en una lista de resultados de clase lm. La información acerca del modelo ajustado puede imprimirse, extraerse, dibujarse, etc. utilizando funciones genéricas orientadas a objetos de clase lm. Algunas de ellas con sus descripción:

- anova(objeto 1, objeto 2), Compara un submodelo con un modelo externo y produce una tabla de análisis de la varianza.
- coefficients(objeto), Extrae la matriz de coeficientes de regresión. Forma reducida: coef(objeto).
- deviance(objeto), suma de cuadrados residual.
- formula(objeto), extrae la fórmula del modelo.
- plot(objeto) crea cuatro gráficos que muestran los residuos, los valores ajustados y algunos diagnósticos.
- predict(objeto, newdata=hoja.de.datos) la nueva hoja de datos que se indica debe tener variables cuyas etiquetas coincidan con las de la original. El resultado es un vector o matriz de valores predichos correspondiente a los valores de las variables de hoja de datos.
- print(objeto) imprime una versión concisa del objeto. A menudo se utiliza implícitamente.

- residuals(objeto) extrae la matriz de residuos, ponderada si es necesario. La forma reducida es resid(objeto).
- step(objeto) Selecciona un modelo apropiado añadiendo o eliminando términos y preservando las jerarquías. Se devuelve el modelo que en este proceso tiene el máximo valor de AIC (Acrónimo del criterio de información Akaike).
- summary(objeto) imprime un resumen estadístico completo de los resultados del análisis de regresión.

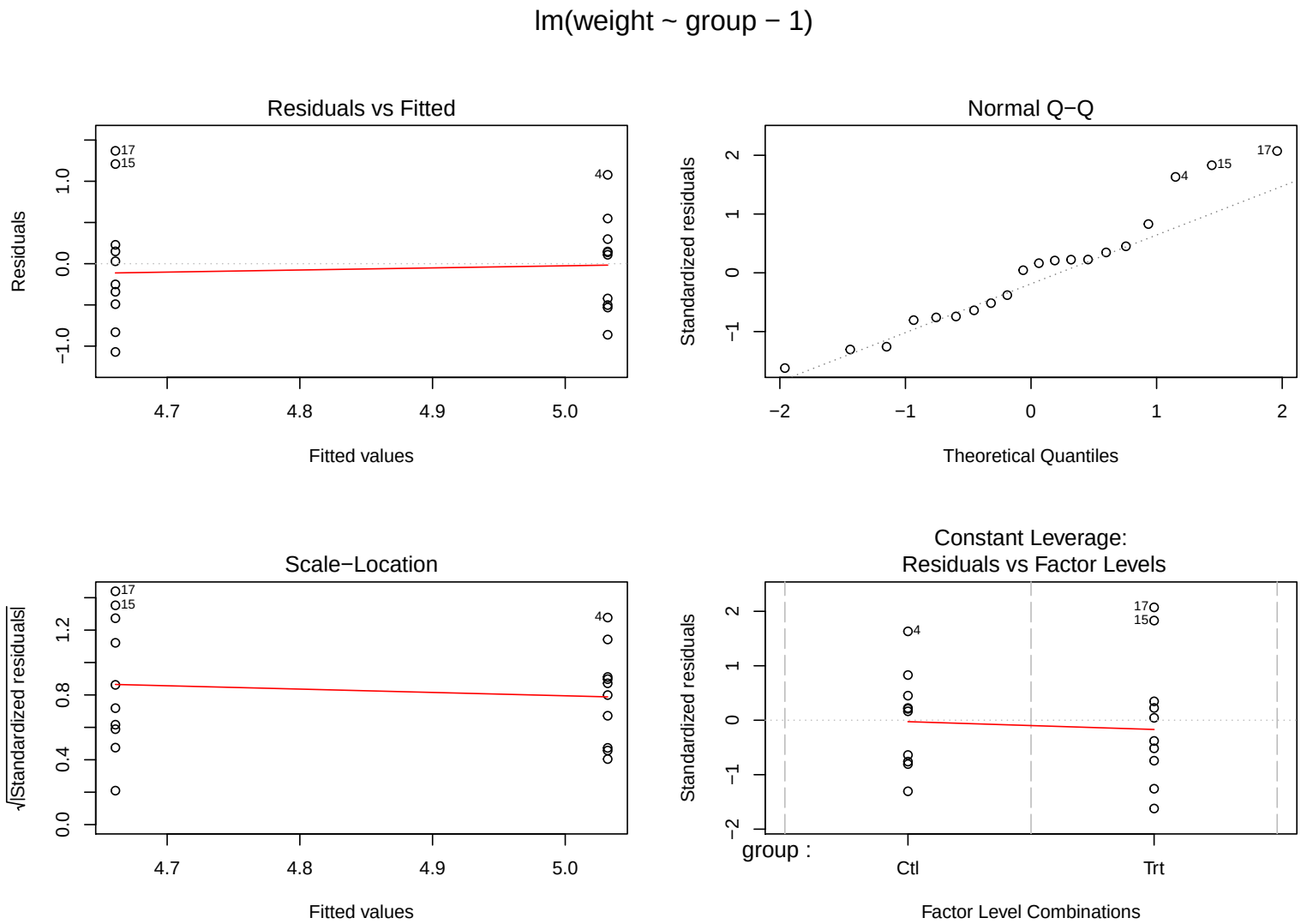
Ejemplo: Annette Dobson (1990) "An Introduction to Generalized Linear Models". Page 9: Plant Weight Data.

```
ctl <- c(4.17,5.58,5.18,6.11,4.50,4.61,5.17,4.53,5.33,5.14)
trt <- c(4.81,4.17,4.41,3.59,5.87,3.83,6.03,4.89,4.32,4.69)
group <- gl(2, 10, 20, labels = c("Ctl","Trt"))
weight <- c(ctl, trt)
lm.D9 <- lm(weight ~ group)
lm.D90 <- lm(weight ~ group - 1) # omitting intercept
```

```r
summary(lm.D90)
```

```
##
## Call:
## lm(formula = weight ~ group - 1)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.0710 -0.4938  0.0685  0.2462  1.3690
##
## Coefficients:
##          Estimate Std. Error t value Pr(>|t|)
## groupCtl   5.0320     0.2202   22.85 9.55e-15 ***
## groupTrt   4.6610     0.2202   21.16 3.62e-14 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.6964 on 18 degrees of freedom
## Multiple R-squared:  0.9818, Adjusted R-squared:  0.9798
## F-statistic: 485.1 on 2 and 18 DF,  p-value: < 2.2e-16
```

```r
opar <- par(mfrow = c(2,2), oma = c(0, 0, 1.1, 0))
plot(lm.D90)      # Residuals, Fitted, ...
```



Ejercicio:

1. Generar una variable x de 20 números aleatorios gaussianos con media 0 y varianza 1. Considerar la variable

```
#y<-3*x+5+rnorm(20,sd=0.3)
```

Ajustar el modelo lineal $y = b_0 + b_1 x + e$ y comparar los resultados.

Modelos lineales generalizados (GLM)

Constituyen una extensión de los modelos lineales, para tomar en consideración tanto distribuciones de respuestas no normales como transformaciones para conseguir linealidad, de una forma directa. Un modelo lineal generalizado puede describirse según las siguientes suposiciones:

- Existe una variable respuesta, y, y unas variables estímulo, x1, x2, . . . , cuyos valores influyen en la distribución de la respuesta.
- Las variables estímulo influyen en la distribución de y mediante una función lineal que recibe el nombre de predictor lineal, y viene dada por

$$\nu = \beta_1 x_1 + \beta_2 x_2 + ... + \beta_p x_p$$

- La distribución de y es de la forma:

$$f_Y(y; \mu, \phi) = \exp\left[\frac{A}{\phi}\{\gamma(\lambda(\mu))\} + \tau(y, \phi)\right]$$

1. Familias

- La clase de GLM que pueden ser tratados en R incluye las distribuciones de respuesta gaussian (normal), binomial, poisson, inversa gaussiana (normal inversa) y gamma así como los modelos de quasi-likelihood (cuasi-verosimilitud) cuya distribución de respuesta no está explícitamente definida. En este último caso debe especificarse la función de varianza como una función de la media, pero en el resto de casos esta función está implícita en la distribución de respuesta.

2. La función glm

Permite ajustar un GLM con la siguiente expresión:

```
#modelo.ajustado <- glm(formula.de.modelo, family =
#                        generador.de.familia, data=hoja.datos)
```

La única característica nueva es "generador.de.familia" que es el instrumento mediante el que se describe la distribución del error y la función enlace a ser usada en el modelo.

Veamos algunos ejemplos:

- Dobson (1990) Page 93: Randomized Controlled Trial :

```
counts <- c(18,17,15,20,10,20,25,13,12)
outcome <- gl(3,1,9)
treatment <- gl(3,3)
d.AD <- data.frame(treatment, outcome, counts)
glm.D93 <- glm(counts ~ outcome + treatment, family = poisson())
```

```
summary(glm.D93)
```

```
##
## Call:
## glm(formula = counts ~ outcome + treatment, family = poisson())
##
## Deviance Residuals:
##       1         2         3         4         5         6         7
## -0.67125   0.96272  -0.16965  -0.21999  -0.95552   1.04939   0.84715
##       8         9
## -0.09167  -0.96656
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)  3.045e+00  1.709e-01  17.815   <2e-16 ***
## outcome2    -4.543e-01  2.022e-01  -2.247   0.0246 *
## outcome3    -2.930e-01  1.927e-01  -1.520   0.1285
## treatment2   1.338e-15  2.000e-01   0.000   1.0000
## treatment3   1.421e-15  2.000e-01   0.000   1.0000
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for poisson family taken to be 1)
##
##     Null deviance: 10.5814  on 8  degrees of freedom
## Residual deviance:  5.1291  on 4  degrees of freedom
## AIC: 56.761
##
## Number of Fisher Scoring iterations: 4
```

- Venables & Ripley (2002, p.189)

```
utils::data(anorexia, package = "MASS")
anorex.1 <- glm(Postwt ~ Prewt + Treat + offset(Prewt),
                family = gaussian, data = anorexia)
```

```r
summary(anorex.1)
```

```
##
## Call:
## glm(formula = Postwt ~ Prewt + Treat + offset(Prewt), family = gaussian,
##     data = anorexia)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -14.1083   -4.2773   -0.5484    5.4838   15.2922
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  49.7711    13.3910   3.717 0.000410 ***
## Prewt        -0.5655     0.1612  -3.509 0.000803 ***
## TreatCont    -4.0971     1.8935  -2.164 0.033999 *
## TreatFT       4.5631     2.1333   2.139 0.036035 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 48.69504)
##
##     Null deviance: 4525.4  on 71  degrees of freedom
## Residual deviance: 3311.3  on 68  degrees of freedom
## AIC: 489.97
##
## Number of Fisher Scoring iterations: 2
```

Pruebas de Hipótesis

t de student

La prueba t de student se utiliza para comprobar la igualdad de las medias de dos muestras o para comprobar si la media de una muestra es igual a una media teórica determinada. Los datos deben tener una distribución normal. La función más utilizada en R es t.test(), del package "stats".

- En el caso de una muestra:

```
#t.test(x,alternative=c("two.sided","less","greater"),mu=0,)
```

donde:

- x : vectores de datos tipo numéricos (muestras a comparar)
- two.sided : si $H_1 : \mu \neq \mu_0$
- less : si $H_1 : \mu < \mu_0$
- greater : si $H_1 : \mu > \mu_0$
- μ : donde se especifica μ_0

```r
t.test(sleep$extra,alternative="two.sided",
       mu=mean(sleep$extra))
```

```
##
##  One Sample t-test
##
## data:  sleep$extra
## t = 0, df = 19, p-value = 1
## alternative hypothesis: true mean is not equal to 1.54
## 95 percent confidence interval:
##  0.5955845 2.4844155
## sample estimates:
## mean of x
##      1.54
```

- En el caso de dos muestras:

```
#t.test(x,y,alternative=c("two.sided","less","greater")
#,mu=0,var.equal =TRUE)
```

donde:

- x ; y : vectores de datos tipo numéricos (muestras a comparar)
- two.sided : si $H_1 : \mu_x - \mu_y \neq \delta_0$
- less : si $H_1 : \mu_x - \mu_y < \delta_0$
- greater : si $H_1 : \mu_x - \mu_y > \delta_0$
- μ : donde se especifica δ_0
- paired : TRUE si son pares de muestras.

```r
x1<-sleep[sleep$group==1,]$extra
y1<-sleep[sleep$group==2,]$extra
t.test(x=x1,y=y1,alternative="two.sided",mu=0,
       var.equal=FALSE)
```

```
##
##  Welch Two Sample t-test
##
## data:  x1 and y1
## t = -1.8608, df = 17.776, p-value = 0.07939
## alternative hypothesis: true difference in means is not equ
## 95 percent confidence interval:
##  -3.3654832  0.2054832
## sample estimates:
## mean of x mean of y
##      0.75      2.33
```

```r
#Voto para el candidato A en 2 distritos
x<- c(120,334)
n<-c(450,1067)
prop.test(x,n,correct=FALSE)
```

```
##
##  2-sample test for equality of proportions without continui
##  correction
##
## data:  x out of n
## X-squared = 3.2439, df = 1, p-value = 0.07169
## alternative hypothesis: two.sided
## 95 percent confidence interval:
##  -0.095793043  0.003072018
## sample estimates:
##    prop 1    prop 2
## 0.2666667 0.3130272
```

chi-cuadrado

Mide la discrepancia entre una distribución observada y otra teórica (bondad de ajuste), indicando en qué medida la diferencias entre ellas, de haberlas, se deben al azar en el contraste de hipótesis.
También se utiliza para probar la independencia entre dos variables entre sí, mediante la presentación de los datos en tablas de contingencia.

```
titanic <- matrix(c(212, 202, 118, 178, 673, 123,167, 528),
                  nrow=2, ncol=4, byrow=TRUE)
titanic
```

```
##      [,1] [,2] [,3] [,4]
## [1,]  212  202  118  178
## [2,]  673  123  167  528
```

```
chisq.test(titanic)
```

```
##
##  Pearson's Chi-squared test
##
## data:  titanic
## X-squared = 187.79, df = 3, p-value < 2.2e-16
```

test para igualdad de varianzas

Es utilizada para evaluar si las varianzas de dos poblaciones con distribución normal son iguales.

```
#var.test(x, y, alternative=c("two.sided" "less", "greater"))
```

donde:

- two.sided : si $H_1 : \sigma_X^2 \neq \sigma_Y^2$
- less : si $H_1 : \sigma_X^2 < \sigma_Y^2$
- greater : si $H_1 : \sigma_X^2 > \sigma_Y^2$

Análisis de Series de Tiempo

Requerimientos básicos en R

- Lectura de data (formato ts) en series de tiempo.
- Herramientas exploratorias para data ts.
- Metodología Box-Jenkins para series de tiempo lineales.

Data ts para Box-Jenkins

El conjunto de datos debe ser:

- Continuo.
- Discreto pero aproximable por un conjunto de datos continuo.
  - Ejemplo: Relación mensual de manchas solares
- Periódico.
  - Ejemplo: diario, semanal, mensual, trimestral, anual.

Paquetes disponibles en CRAN

- Utilizaremos el paquete astsa, desarrollado por David Stoffer y el paquete stats.
- Consultar "Time Series Analysis and Its Applications: With R Examples" by Shumway and Stoffer.
- Un link útil para el análisis ts puede ser encontrado en: http://cran.r-project.org/web/views/TimeSeries.html

Data ts en R

Los objetos más comunes en R son las listas y los marco de datos. Para manipular series de tiempo utilizamos el comando ts()

```
#ts(data = NA, start = 1, end = numeric(), frequency = 1)
```

donde:

- data : vector o matrix de valores observados.
- start : tiempo de la primera observación.
- end : tiempo de la última observación.
- frequency : número de observaciones por unidad de tiempo.

Observación: La construcción de objetos de la clase ts requiere que los datos de partida estén distribuidos regularmente en la escala temporal utilizada (Ejemplo: siempre se cumpla un dato por día)

```r
#conjunto de datos
(mydata = c(1,2,3,2,1))
```

```
## [1] 1 2 3 2 1
```

```r
#transformar en serie de tiempo
(my.data=as.ts(mydata))
```

```
## Time Series:
## Start = 1
## End = 5
## Frequency = 1
## [1] 1 2 3 2 1
```

```r
#serie de tiempo anual que comience en 1950
(mydata =ts(mydata,start=1950))
```

```
## Time Series:
## Start = 1950
## End = 1954
## Frequency = 1
## [1] 1 2 3 2 1
```

```
#serie de tiempo trimestral que comience en 1950-III
(mydata=ts(mydata,start=c(1950,3),frequency=4))
```

```
##      Qtr1 Qtr2 Qtr3 Qtr4
## 1950                1    2
## 1951    3    2    1
```

```
time(mydata) #ver el tiempo en la muestra
```

```
##          Qtr1    Qtr2    Qtr3    Qtr4
## 1950                    1950.50 1950.75
## 1951 1951.00 1951.25 1951.50
```

```
#usar parte del objeto de serie de tiempo
(x=window(mydata,start=c(1951,1),end=c(1951,3)))
```

```
##      Qtr1 Qtr2 Qtr3
## 1951    3    2    1
```

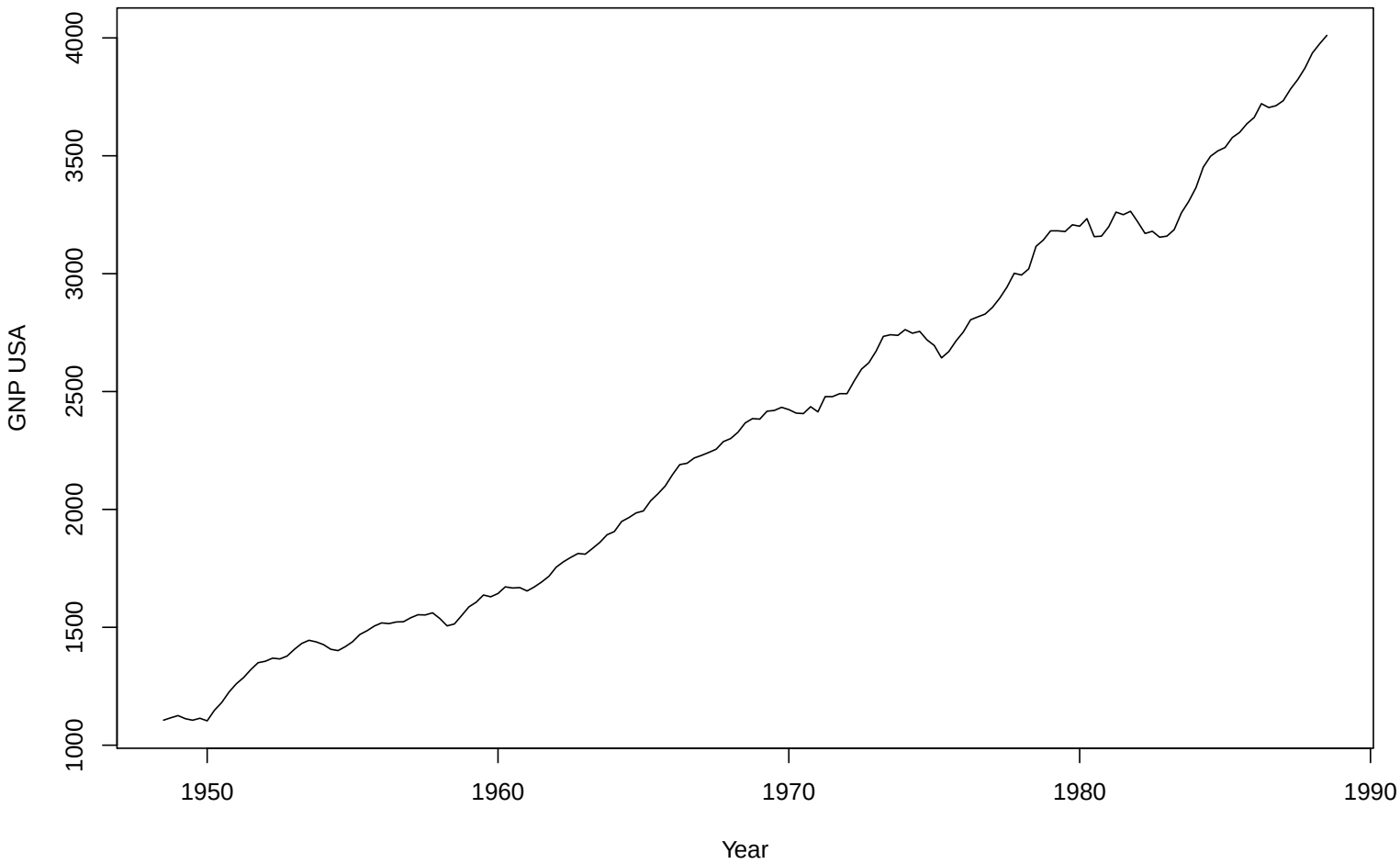
- Conjunto de datos que contiene el desempleo trimestral, GNP, consumo y la inversión gubernamental y privada, de III-1948 a II-1988

```
gnp<-ts(as.matrix(econ5$gnp),start=c(1948,3),
        frequency=4)
str(gnp)
```

```
##  Time-Series [1:161, 1] from 1948 to 1988: 1106 1116 1126 1112 1
##  - attr(*, "dimnames")=List of 2
##   ..$ : NULL
##   ..$ : chr "Series 1"
```

Graficación de series de tiempo

```
plot(gnp,xlab="Year",ylab="GNP USA")
```



Ajuste modelo Box-Jenkins

1. Débilmente estacionario: (i) Media constante; (ii) La función de auto-covarianza depende únicamente del rezago.

Nota: (ii) implica que la varianza es una constante.

2. Gráficamente, observamos la media y varianza.

- Si es constante, procedemos con el ajuste del modelo.
- En otro caso, exploramos transformaciones y luego diferenciamos para validar de nuevo el item 1. Repetimos este paso tantas veces se requiera

Modelo moving average (MA)

- En análisis de series de tiempo, el moving average es el enfoque común para modelar series de tiempo univariada.
- X_t es un proceso moving average de orden q si

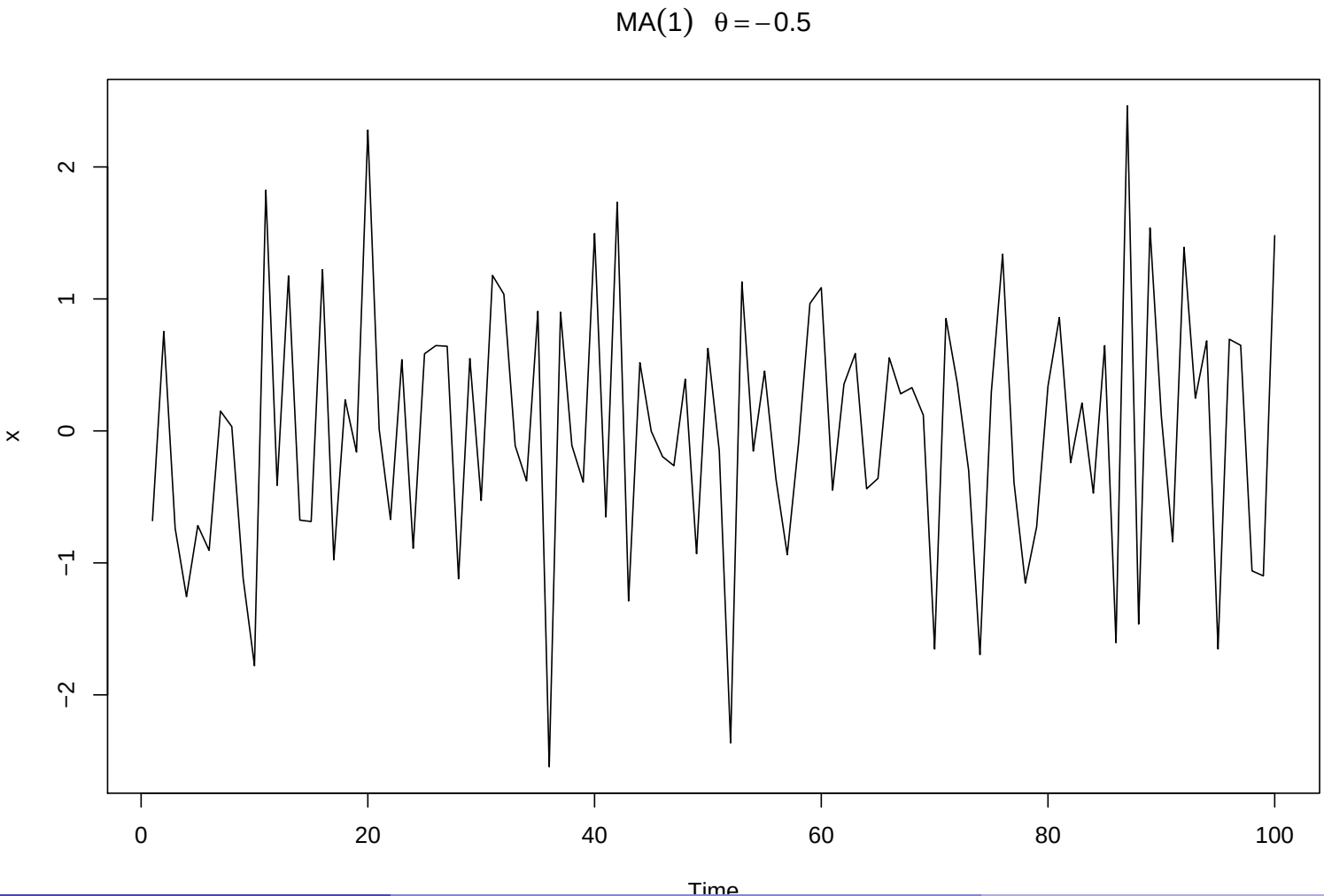
$$X_t = \mu + \epsilon_t + \theta_1 \epsilon_{t-1} + ... + \theta_q \epsilon_{t-q}$$

donde:

- μ : es la media de la serie
- $\theta_1, ..., \theta_q$ son los parámetros del modelo
- $\epsilon_t, \epsilon_{t-1}, ..., \epsilon_{t-q}$ son términos del error "ruido blanco".
- El modelo puede ser alternativamente escrito en términos del operador backshift B como

$$X_t = \mu + (1 + \theta_1 B + ... + \theta_q B^q)\epsilon_t$$

```
#MA(1)
out<-arima.sim(list(order=c(0,0,1), ma=-.5),n=100)
plot(out,ylab="x",main=(expression(MA(1)~~~theta==-.5)))
```



Funciones ACF Y PACF

1. La autocorrelación es la dependencia lineal de una variable con ella misma entre dos puntos en el tiempo. Para procesos estacionarios, autocorrelación entre 2 o más observaciones únicamente depende del intervalo de tiempo h entre ellas. Viene dada por la siguiente expresión:

$$\rho(h) = \frac{\gamma(t+h,t)}{\sqrt{\gamma(t+h,t+h)\gamma(t,t)}} = \frac{\gamma(h)}{\gamma(0)}$$

donde:

- $\gamma(h) = cov(x_{t+h}, x_t) = E[(x_{t+h} - \mu)(x_t - \mu)]$; h fijo.

2. La función de autocorrelación (ACF) es la sucesión $\rho(h)_{h=1}^{N-1}$

3. La autocorrelación parcial es la autocorrelación entre x_t y x_{t-h} sin la dependencia lineal en $x_1, x_2, ..., x_{t-h-1}$:

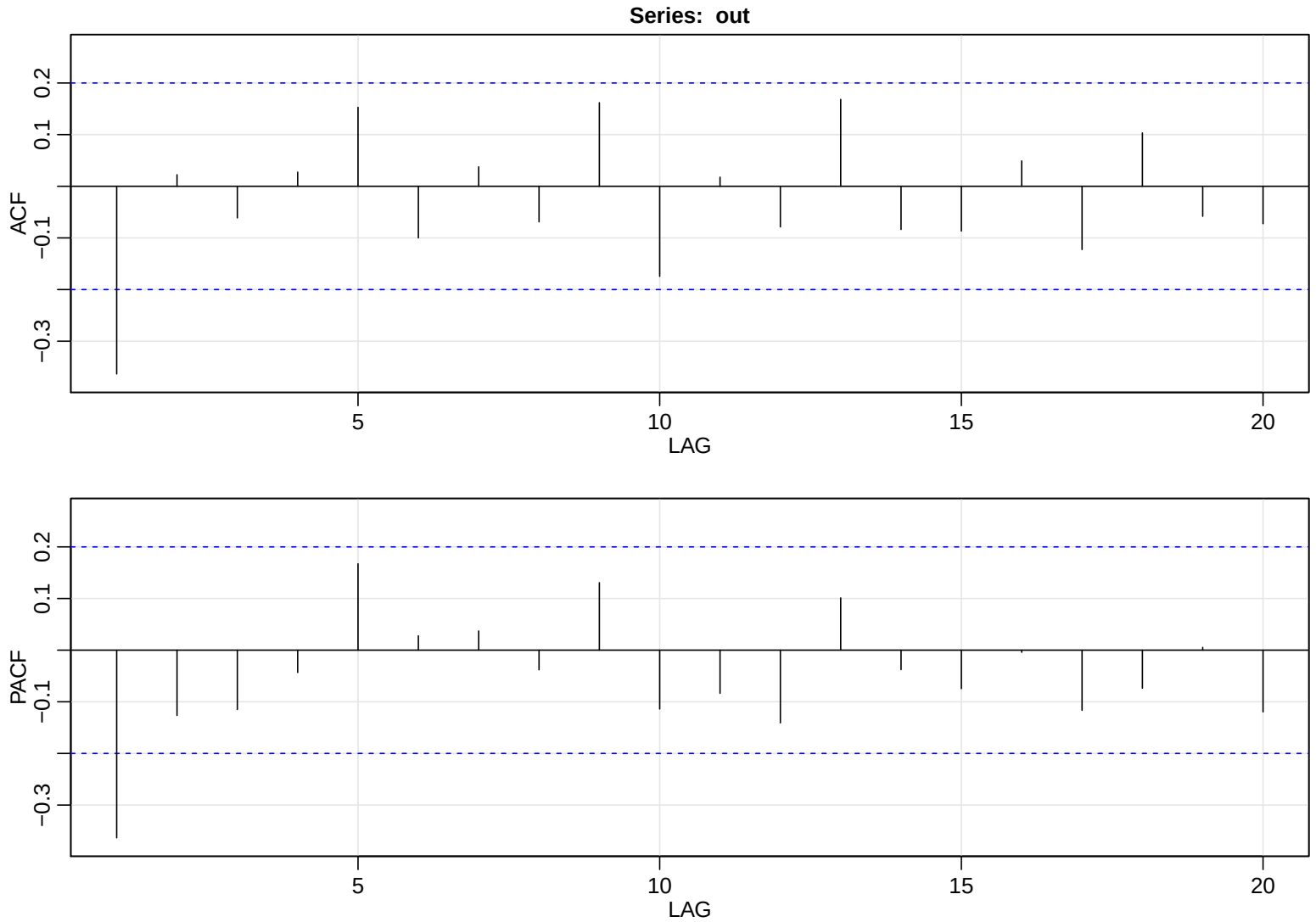
$$\phi_{hh} = corr(x_{t+h} - \widehat{x}_{t+h}, x_t - \widehat{x}_t)$$

donde:

- $\widehat{x}_t$: regresión de x_t en $x_{t+1}, x_{t+2}, ..., x_{t+h-1}$

4. La función de autocorrelación parcial (PACF) de un proceso estacionario, x_t, es la sucesión $\{\phi_{hh}\}_{h \geq 1}$

```
acf2(out)
```




```
##      ACF  PACF
```

Modelo autorregresivo (AR)

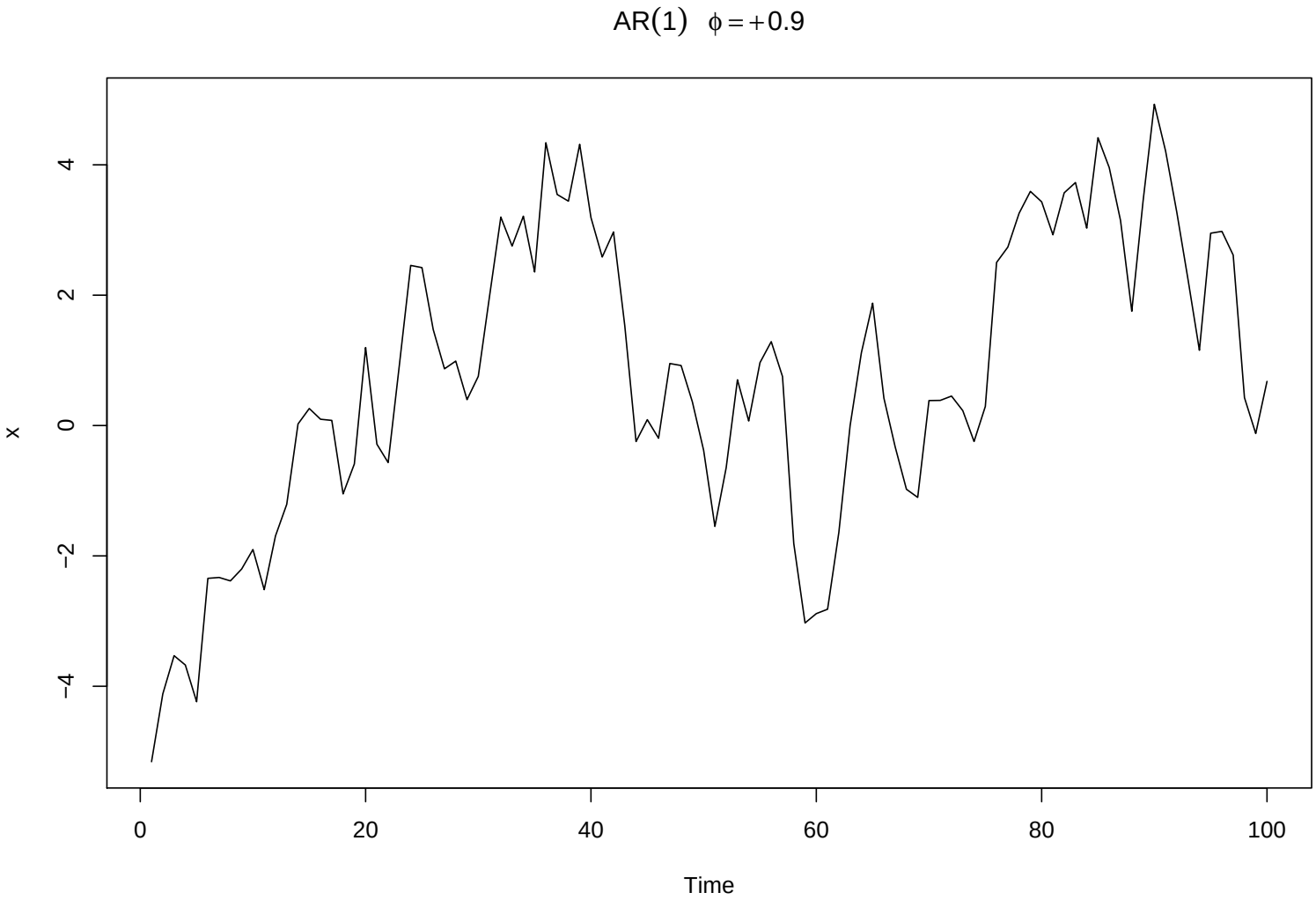
Un modelo autorregresivo de orden p, AR(p), es de la forma

$$x_t = \phi_1 x_{t-1} + \phi_2 x_{t-2} + ... + \phi_i x_{t-p} + w_t$$

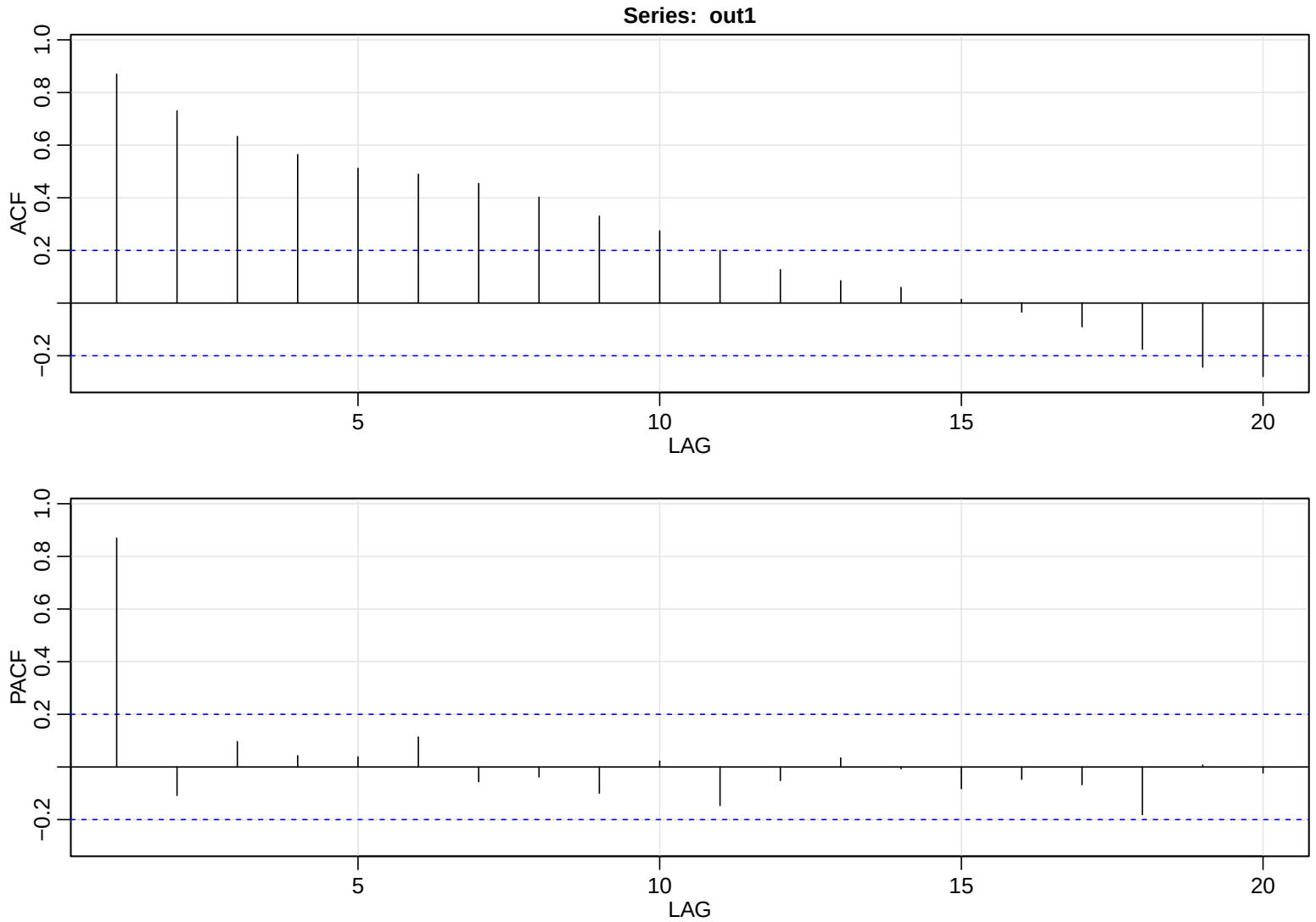
donde:

- x_t : es un modelo estacionario
- $\phi_1, \phi_2, ..., \phi_p$ son constantes $\phi \neq 0$

```
out1=arima.sim(list(order=c(1,0,0),ar=.9), n=100)
plot(out1,ylab="x",main=(expression(AR(1)~~~phi==+.9)))
```



```
acf2(out1)
```




```
##      ACF  PACF
```

Modelo moving average autorregresivo (ARMA)

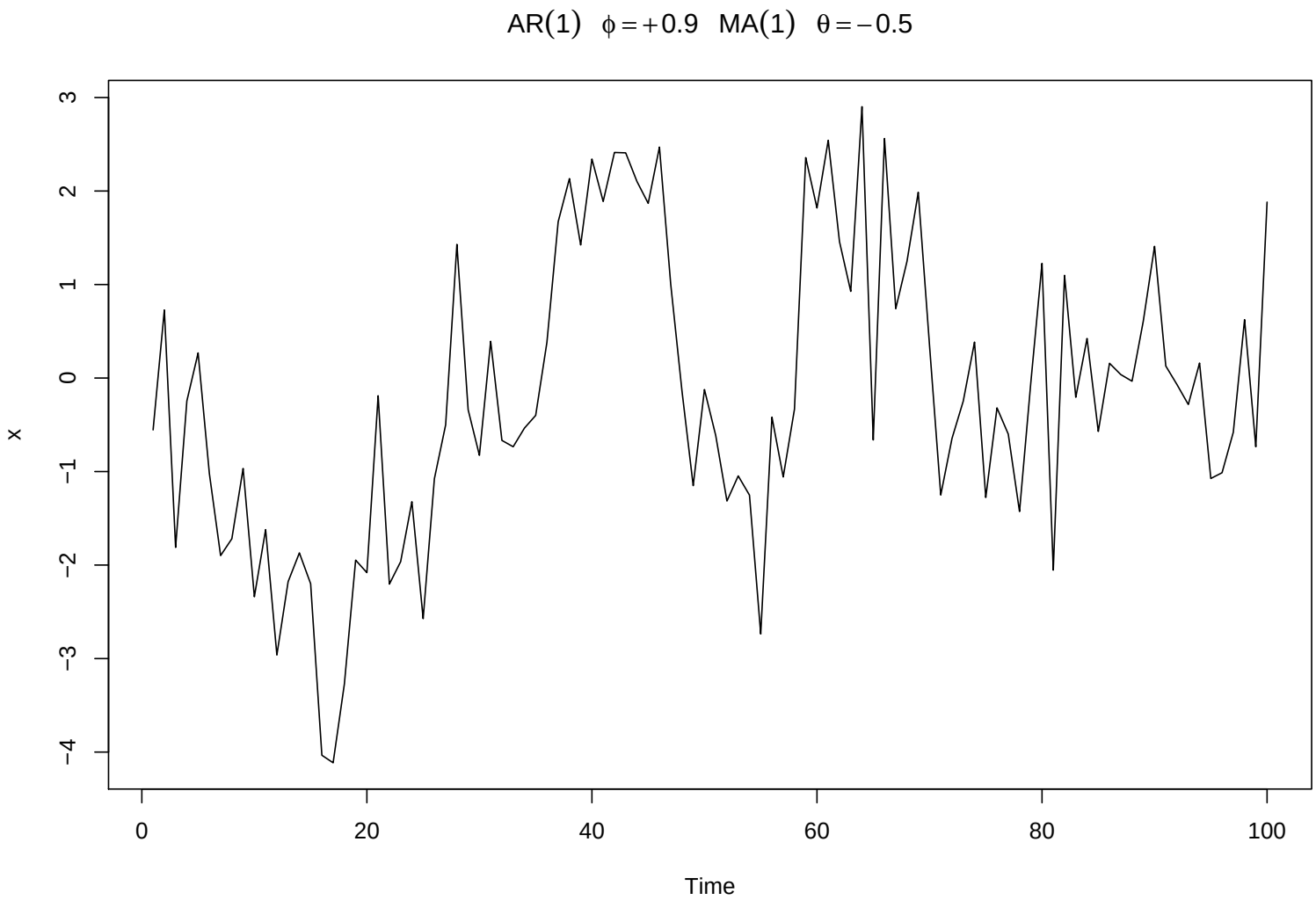
Una serie de tiempo $\{x_t\}$ es ARMA(p,q) si este es estacionario y

$$x_t = \phi_1 x_{t-1} + ... + \phi_p x_{t-p} + w_t + \theta_1 w_{t-1} + ... + \theta_q w_{t-q}$$

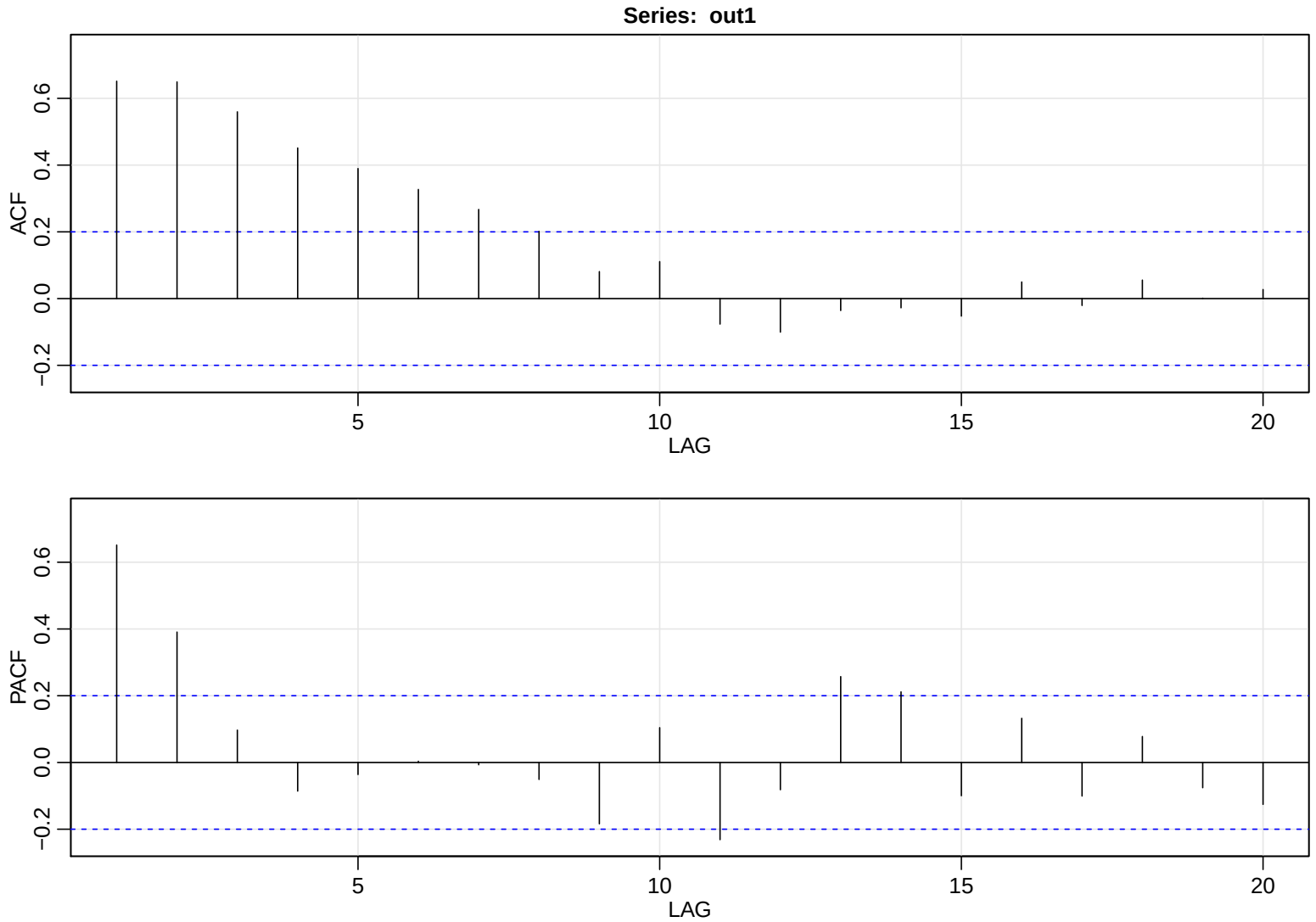
donde,

- $\phi_p \neq 0$, $\theta_q \neq 0$ y $\sigma_w^2 > 0$.
- Los parámetros p y q establecen el orden del modelo autorregresivo y moving average respectivamente.

```
out1=arima.sim(list(order=c(1,0,1),      ar=0.9,ma=-.5),n=100)
plot(out1, ylab="x", main=(expression(AR(1)
~~~phi==+.9~~~MA(1)~~~theta==-.5)))
```



```
acf2(out1)
```



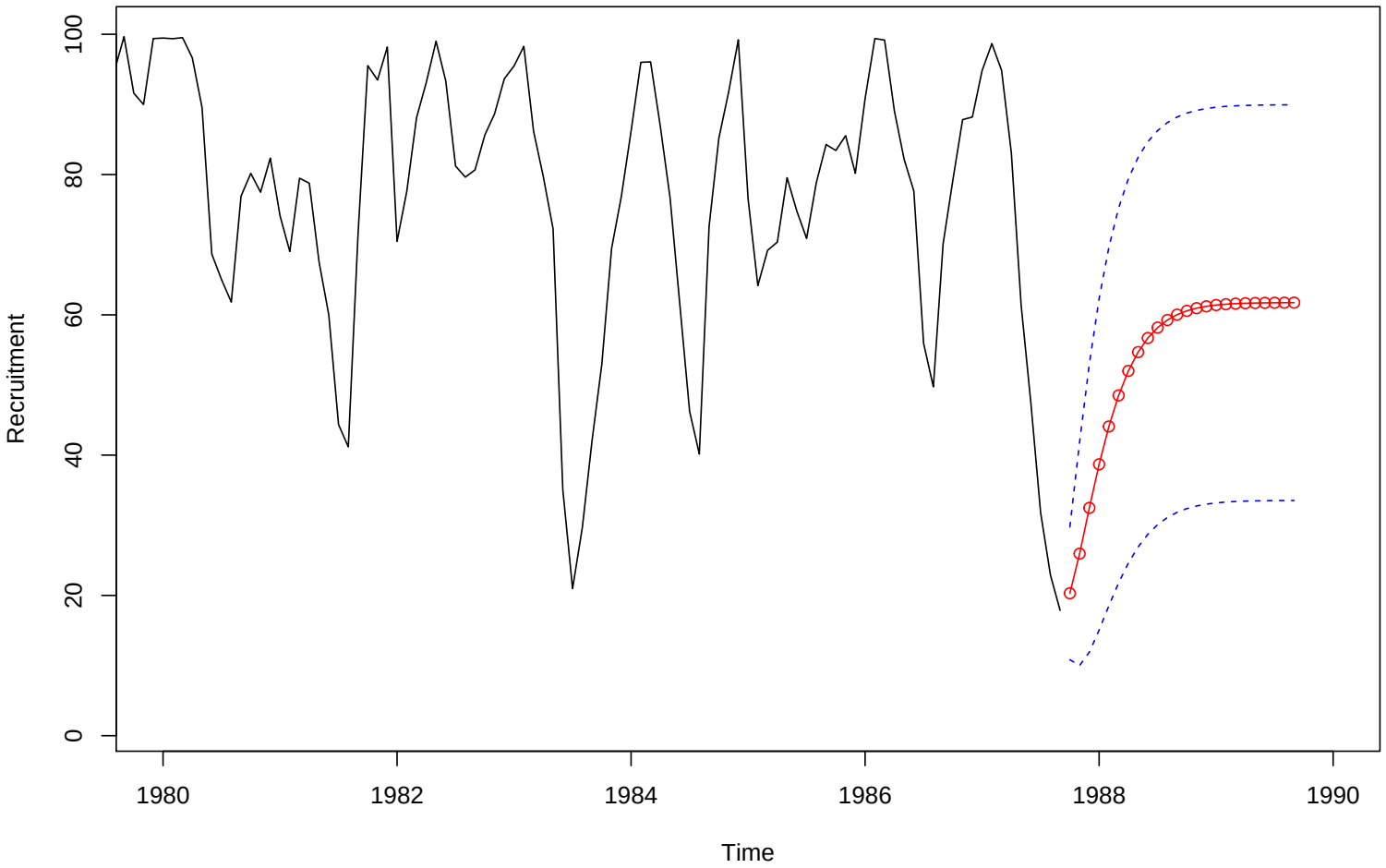

```
##      ACF  PACF
```

Estimación y predicción

OLS

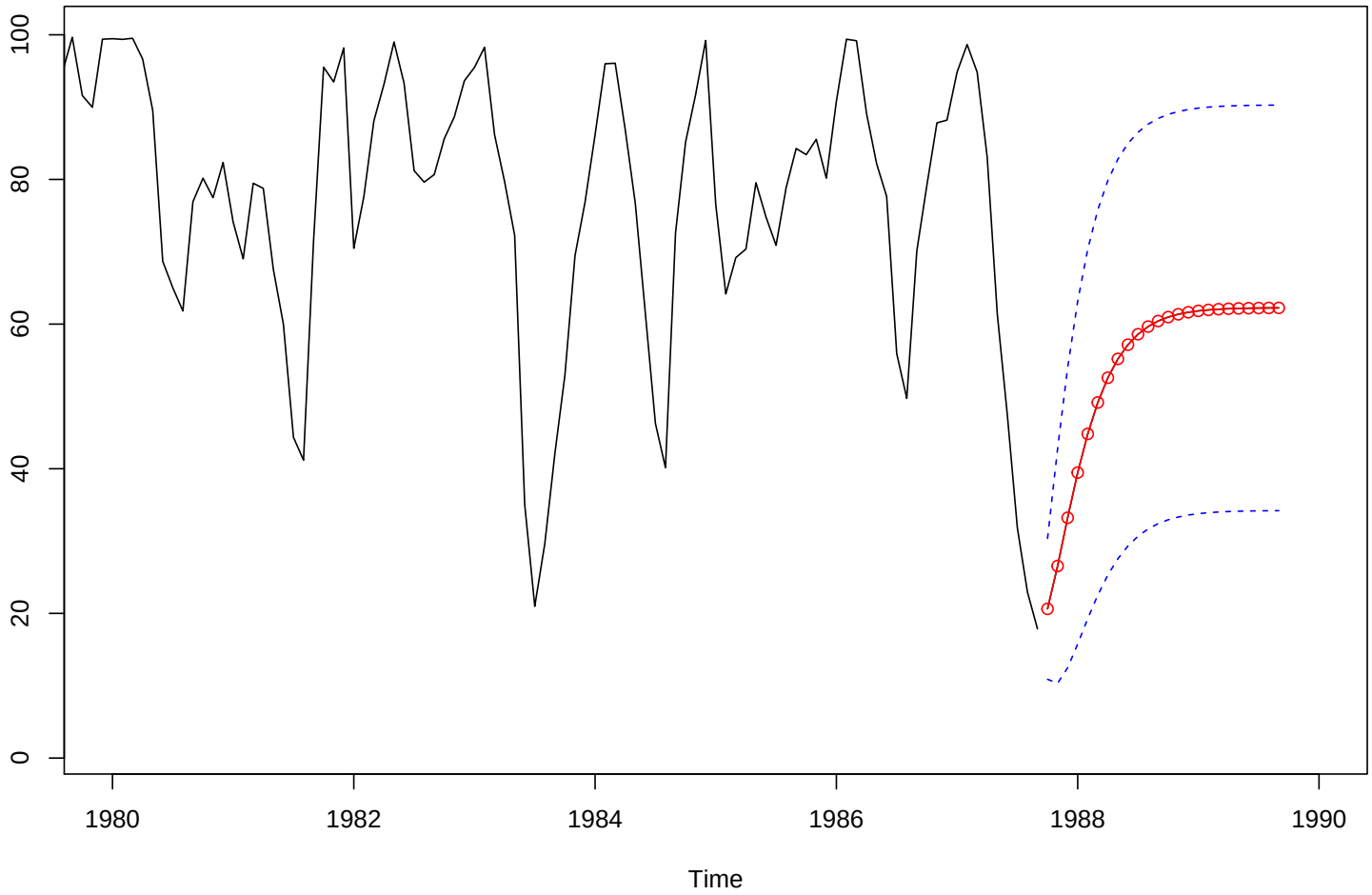
```
#regr = ar.ols(rec, order=2,demean=FALSE, intercept=TRUE)
#fore = predict(regr, n.ahead=24)
#ts.plot(rec, fore$pred, col=1:2, xlim=c(1980,1990),
#          ylab="Recruitment")
#lines(fore$pred, type="p", col=2)
#lines(fore$pred+fore$se, lty="dashed", col=4)
#lines(fore$pred-fore$se, lty="dashed", col=4)
```

```
## Warning in object$var.pred * vars: Recycling array of lengt
##   Use c() or as.vector() instead.
```



Yule–Walker

```
#rec.yw = ar.yw(rec, order=2)
# rec.yw$x.mean # = 62.26 (mean estimate)
# rec.yw$ar # = 1.33, -.44 (parameter estimates)
# sqrt(diag(rec.yw$asy.var.coef)) # = .04, .04 (standard errors)
# rec.yw$var.pred # = 94.80 (error variance estimate)
#rec.pr = predict(rec.yw, n.ahead=24)
#U = rec.pr$pred + rec.pr$se
#L = rec.pr$pred - rec.pr$se
# minx = min(rec,L); maxx = max(rec,U)
#ts.plot(rec, rec.pr$pred, xlim=c(1980,1990), ylim=c(minx,maxx))
#lines(rec.pr$pred, col="red", type="o")
#lines(U, col="blue", lty="dashed")
#lines(L, col="blue", lty="dashed")
```


0
20
40
60
80
100
1980
1982
1984
1986
1988
1990
Time

Modelos ARIMA

Un proceso $\{x_t\}$ se dice ARIMA(p,d,q) si

$$diff_t^d = (1 - B)^d x_t$$

es ARMA(p,q)