

Bioherramientas en Bioinformática y Bioestadística (1ª ed)

Creación de librerías con R

Juan R González

Insitituto de Salud Global Barcelona (ISGlobal)

Departamento de Matemáticas, Universidad Autónoma de Barcelona (UAB)

juanr.gonzalez@isglobal.org

<http://www.creal.cat/jrgonzalez/software.htm>

<https://github.com/isglobal-brge>

16, 17 y 18 de Mayo de 2017

Parque de Investigación Biomédica de Barcelona
(PRBB)

Creación de librerías con R

Guión

1. Introducción

- Motivación
- Paquetes en Windows: Preparación e instalación de software requerido

2. Programación en R

- Definición de nuevos operadores binarios
- Creación de nuevas funciones
 - Cómo organizar una función
 - Nombres de argumentos y valores por defecto
 - Control de los argumentos
- Uso de 'formula'

3. Conexión con C y Fortran

- Creación de dll's
- Llamadas a dll's

Guión (Cont)

4. Métodos y clases en R

- Programación orientada a objetos
- Creación de Métodos
- Creación de Clases

5. Creación de librerías en R

- Estructura
- El NAMESPACE
- Creación de documentación
- Vignettes
- Envío a CRAN

Sesión 1 – Introducción y preliminares

- Introducción
- Paquetes en Windows (preliminares – instalación y requerimientos)

Creación de librerías con R

Investigación

- Desarrollan un método/modelo estadístico (matemático, físico, ...) para resolver un problema real (datos complejos, nuevo tipo de datos, ...)
- Desarrollan un algoritmo de estimación que puede llegar a ser muy complejo
- Quieren que otros investigadores (biólogos, médicos, ...) utilicen este nuevo método de forma rutinaria y fácil
- Crear una colección de funciones que sean fácil de usar, así como otras aplicaciones que faciliten el análisis y/o interpretación de datos (gráficos, resúmenes, ...)
- CREAR UN PAQUETE EN R

Metodología

- Modelo de regresión lineal (método estimación basado en descomposición QR) [TEORIA]
(adaptado de *Creating R Packages: A Tutorial. Friedrich Leisch, 2008*)
 - Incluye poner unos datos de ejemplo
 - Manual, vignette, ...
- **Ejercicio:** Crear una función “compareGroups” que compare dos grupos de poblaciones (test paramétrico, no paramétrico, datos apareados, ...) [PRACTICA]

Creación de librerías con R

Tipos de paquetes

Base packages: Part of the R source tree, maintained by R Core.

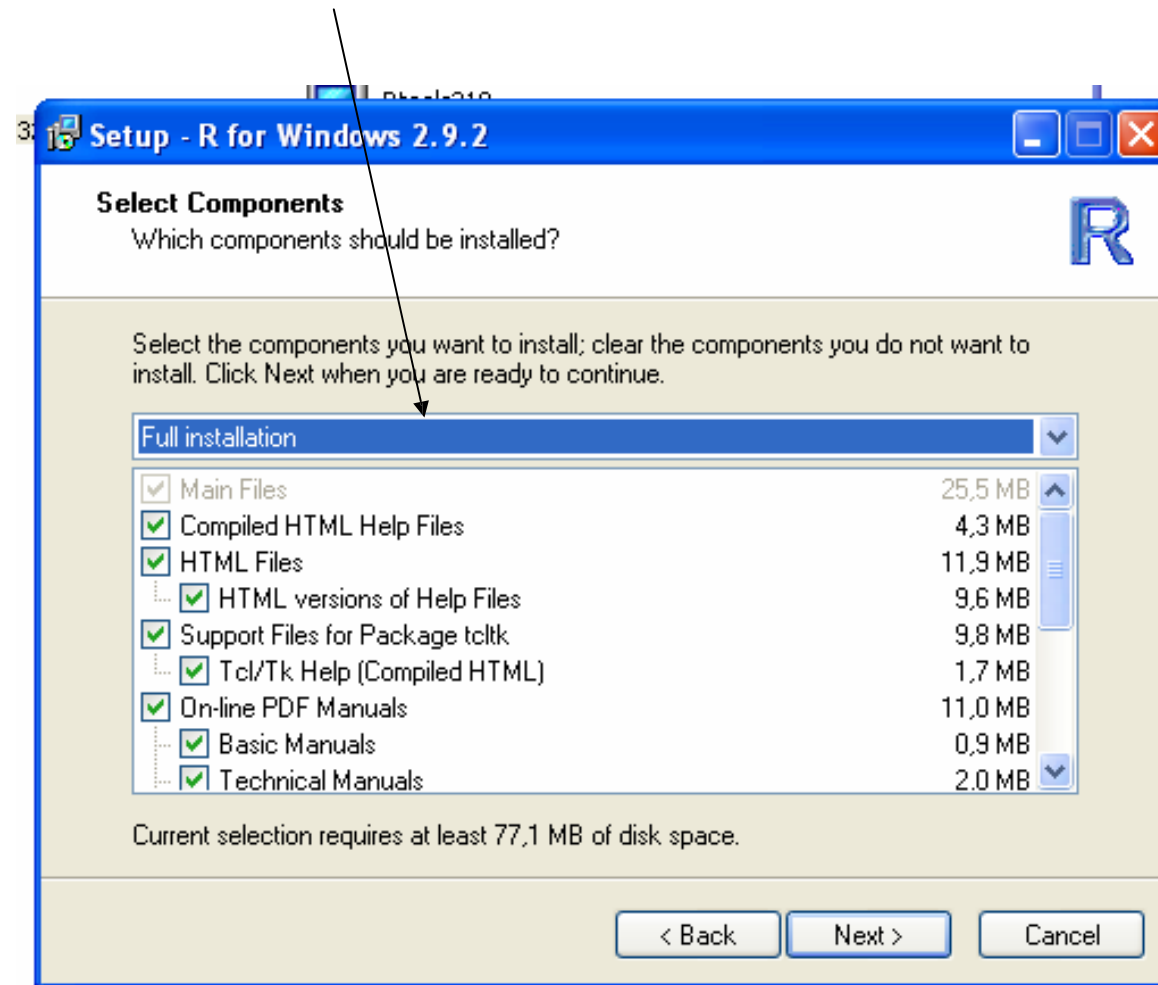
Recommended packages: Part of every R installation, but not necessarily maintained by R Core.

Contributed packages: All the rest. This does not mean that these packages are necessarily of lesser quality than the above, e.g., many contributed packages on CRAN are written and maintained by R Core members. We simply try to keep the base distribution as lean as possible.

Creación de librerías con R

Instalación y requerimientos

Instalar R en version "Full installation"



Instalación y requerimientos

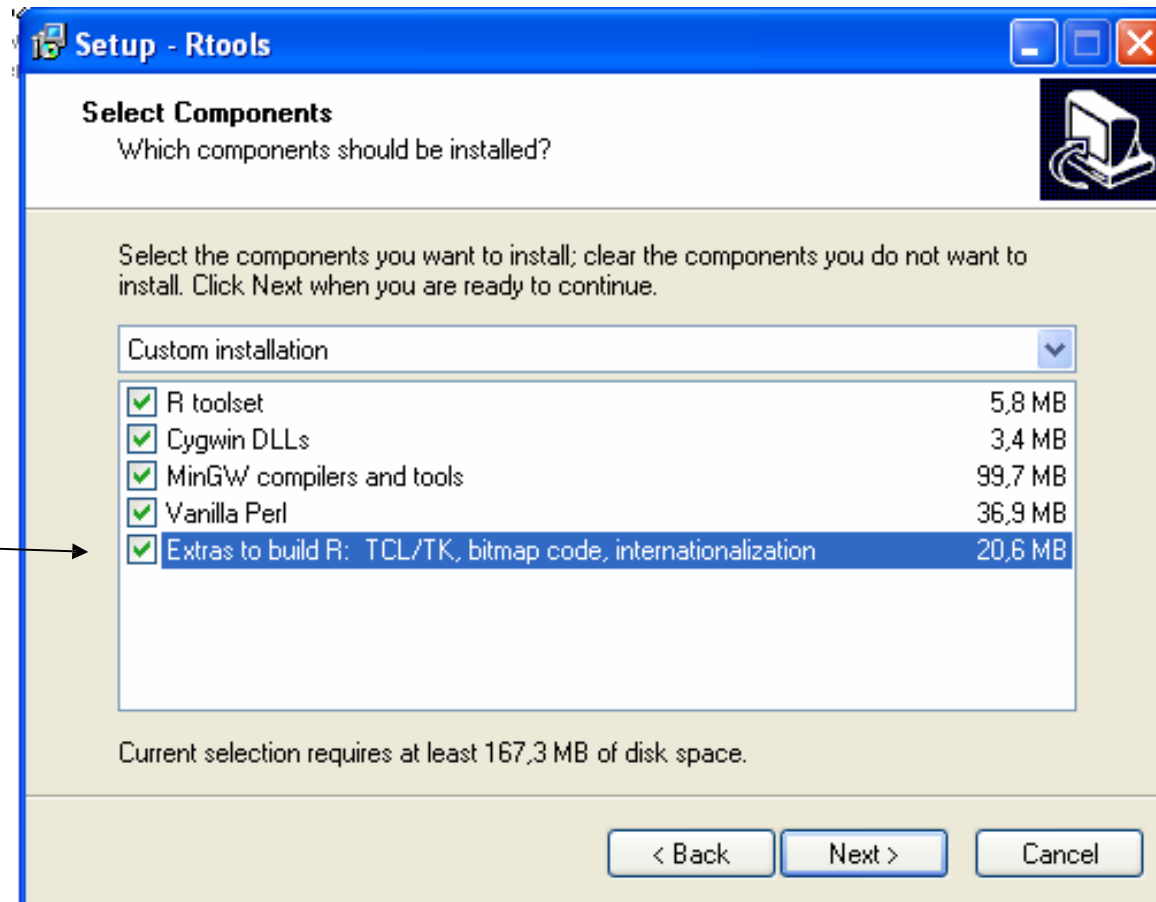
- Se puede mirar en “Windows toolset appendix” y en “R Administration manual”
- Windows: descargar Rtools de: <https://cran.r-project.org/bin/windows/Rtools/>
- Linux y Mac no es necesario estas tools (Rtools emula Linux con Cygwin)

Instala las siguientes “tools”

- unas cuantas herramientas ejecutables mediante lineas de comandos
- Perl
- El compilador MinGW
- También es necesario para las vignettes:
 - Microsoft’s HTML Help Workshop
 - MikTeX o LaTeX

Curso de R avanzado

Instalación y requerimientos



Instalación y requerimientos

Hay que añadir las Rtools y el MinGW al path del ordenador. Desde R se puede con

```
rtools <- "C:\\Rtools\\bin"
gcc <- "C:\\Rtools\\min_gw_64\\bin"
path <- strsplit(Sys.getenv("PATH"), ";")[[1]]
new_path <- c(rtools, gcc, path)
new_path <- new_path[!duplicated(tolower(new_path))]
Sys.setenv(PATH = paste(new_path, collapse = ";"))
```

Ejercicio

Testar que todo está OK ... ejecutar

```
system("R CMD SHLIB -o prueba.dll prueba.f")
```

desde la carpeta donde tengáis el fichero 'prueba.f'

Sesión 2 – Programación en R

- Definición de nuevos operadores binarios
- Creación de nuevas funciones
 - Cómo organizar una función
 - Nombres de argumentos y valores por defecto
 - Control de los argumentos
 - Uso de 'formula'
 - Resultado de una función

Creación de librerías con R

Creación de nuevas funciones

El lenguaje R suele utilizarse como un conjunto de expresiones "sueltas". Muchas veces nos interesará agruparlas bajo un determinado nombre (función) y parametrizarlas (argumentos), para mas tarde crear un grupo de funciones (libreria o paquete). Como en la mayoría de los otros lenguajes de programación, **las asignaciones dentro de funciones son temporales y se pierden cuando acaba su ejecución.**

- Crear funciones en R permite añadir nuevas funcionalidades
- Las funciones que escribe un usuario tiene el mismo status que aquellas que estan escritas en R
- Leer/estudiar las funciones que estan escritas en R es una buena forma de aprender cómo escribir funciones
- Otros usuarios pueden modificar las funciones y adaptarlas a sus necesidades (open source)

Las funciones se definen siguiendo el siguiente patrón:

nombre <- function(arg_1, arg_2, ..., arg_n) expr

expr suele ser una expresión agrupada que usa los argumentos *arg_i* para calcular un valor. El valor de la expresión es el valor retornado por la función.

Las llamadas a funciones suelen tener el siguiente aspecto:

nombre(expr_1, expr_2, ..., expr_n)

Creación de librerías con R

Un ejemplo de función simple.

Vamos a crear una función que, dados los tres coeficientes (a , b , c) de una ecuación de segundo grado, $ax^2 + bx + c = 0$, nos devuelva las dos soluciones, si existen en el conjunto de los reales. Recordemos que la fórmula para encontrar las dos soluciones es:

$$x = (-b \pm \sqrt{b^2 - 4ac}) / 2a$$

```
# Cabecera de la función: le damos un nombre y definimos los tres parámetros
# Como vamos a utilizar más de una instrucción para programar la función, ya añadimos con la llave el inicio
# del grupo de instrucciones
ec_grado2 <- function(a, b, c)
{
  # Calculamos el discriminante (b^2 - 4ac)
  disc <- (b^2)-(4*a*c)
  # Primera solución
  sol1 <- (-b + sqrt(disc))/(2*a)
  # Segunda solución
  sol2 <- (-b - sqrt(disc))/(2*a)
  # Devolvemos un vector con las dos soluciones y cerramos la llave para indicar el final del grupo de
  # instrucciones que forman la función
  ans <- c(sol1,sol2)
  ans
}
```

Creación de librerías con R

Definición de nuevos operadores binarios.

Para funciones que tengan exclusivamente dos parámetros, R permite definirlos de manera que después se puedan llamar con notación infija, es decir, poniendo el nombre de la función entre los dos argumentos. Un ejemplo de una función típicamente infija es la suma (escribimos $3+7$ y no $+(3,7)$ o $\text{suma}(3,7)$).

Para crear una función con notación infija simplemente tenemos que añadir el carácter % delante y detrás del nombre de la función. R ya interpretará de esta manera que tipo de función estamos creando.

```
# Vamos a crear una función que reciba dos numeros y devuelva la división del primero entre la suma de los
# dos multiplicado por 100. Nótese que es obligatorio que el nombre de la función vaya entre comillas.
> "%porcentaje%" <- function(a,b) (a/(a+b))*100
# Ejecutamos la función
> 4 "%porcentaje%" 6
[1] 40
```

Algunos ejemplos de este tipo de funciones son el producto matricial `%*%`, o la inclusión en listas `%in%`.

Creación de librerías con R

Nombres de argumentos y valores por defecto.

Al contrario que la gran mayoría de lenguajes de programación, R permite colocar los argumentos de una llamada a una función en el orden que nosotros queramos. Para poder hacer esto tenemos que especificar para cada valor en la llamada a qué argumento corresponde. En caso que no lo hagamos entonces R sí entenderá que los argumentos de la llamada están en el mismo orden en que han sido especificados en la declaración de las funciones.

```
# Vamos a estudiar los argumentos de la función rnorm
```

```
> args(rnorm)
```

```
function (n, mean = 0, sd = 1)
```

```
# La llamamos sin poner los nombres de los argumentos
```

```
> rnorm(10,1,2)
```

```
# Ahora ponemos los nombres de los argumentos: podemos cambiar su orden
```

```
> rnorm(sd=2,n=10,mean=1)
```

Vemos que los parámetros *mean* y *sd* tienen asociados valores por defecto (0 y 1 respectivamente). En este caso, si aceptamos estos valores por defecto como los deseados para ejecutar la función, podemos obviarlos. Una vez prescindamos de un parámetro, sin embargo, deberemos escribir el nombre de todos los que vengan detrás de él.

```
# Prescindimos de cambiar mean, pero tenemos que especificar que el siguiente es sd.
```

```
> rnorm(10,sd=3)
```


Creación de librerías con R

Partes importantes de una función

```
t.test<-function (x, y = NULL, alternative = c("two.sided", "less", "greater"), mu = 0, paired = FALSE, var.equal = FALSE, conf.level = 0.95, ...)
```

Parámetros o argumentos

```
{
  alternative <- match.arg(alternative)
  if (!missing(mu) && (length(mu) != 1 || is.na(mu)))
    stop("'mu' must be a single number")
  if (!missing(conf.level) && (length(conf.level) != 1 || !is.finite(conf.level) ||
    conf.level < 0 || conf.level > 1))
    stop("'conf.level' must be a single number between 0 and 1")
  if (!is.null(y)) {
    dname <- paste(deparse(substitute(x)), "and", deparse(substitute(y)))
    if (paired)
      xok <- yok <- complete.cases(x, y)
    else {
      yok <- !is.na(y)
      xok <- !is.na(x)
    }
    y <- y[yok]
  }
  else {
    ...
  }
}
```

Control de argumentos

```
if (is.null(y)) {
  df <- nx - 1
  stderr <- sqrt(vx/nx)
  if (stderr < 10 * .Machine$double.eps * abs(mx))
    stop("data are essentially constant")
  tstat <- (mx - mu)/stderr
  method <- ifelse(paired, "Paired t-test", "One Sample t-test")
  names(estimate) <- ifelse(paired, "mean of the differences",
    "mean of x")
}
```

t test caso univariante

Cálculo

Creación de librerías con R

Partes importantes de una función

```
..... Viene de otro if en el que hacía el caso univariante
else {
  # Two sample ny <- length(y)
  # Controlar varianzas iguales o no
  }

  if (alternative == "less") {
    pval <- pt(tstat, df)
    cint <- c(-Inf, tstat + qt(conf.level, df))
  }
  else if (alternative == "greater") {
    pval <- pt(tstat, df, lower = FALSE)
    cint <- c(tstat - qt(conf.level, df), Inf)
  }
  else {
    pval <- 2 * pt(-abs(tstat), df)
    alpha <- 1 - conf.level
    cint <- qt(1 - alpha/2, df)
    cint <- tstat + c(-cint, cint)
  }
  cint <- mu + cint * stderr
  names(tstat) <- "t"
  names(df) <- "df"
  names(mu) <- if (paired || !is.null(y))
    "difference in means"
  else "mean"

  attr(cint, "conf.level") <- conf.level
  rval <- list(statistic = tstat, parameter = df, p.value = pval,
    conf.int = cint, estimate = estimate, null.value = mu,
    alternative = alternative, method = method, data.name = dname)
  class(rval) <- "htest"
  return(rval)
}
```

t test caso dos muestras

P valor e IC

Cálculo

Resultado de la función

Creación de librerías con R

Ejemplo regresión lineal

El investigador propone un método para analizar Y (vector) en función de X (respuesta)

$$y = x'\beta + \epsilon, \quad \epsilon \sim N(0, \sigma^2)$$

No vamos a hacer lo mismo que la función *lm()* sino estimar mediante OLS (e.g., el algoritmo propuesto por el investigador). El estimador viene dado por

$$\hat{\beta} = (X'X)^{-1}X'y$$

Con matriz de covarianza

$$\text{var}(\hat{\beta}) = \sigma^2(X'X)^{-1}$$

Por motivos numéricos el investigador propone utilizar la descomposición QR para resolver el sistema de ecuaciones

Creación de librerías con R

Ejemplo regresión lineal

```
> lmEst <- function(x, y)
+ {
+ ## calcula la descomposicion QR de x
+ qx <- qr(x)
+
+ ## calcula  $(x'x)^{-1} x'y$ 
+ coef <- solve.qr(qx, y)
+
+ ## grados de libertad y desviacion estandard de los residuales
+ df <- nrow(x)-ncol(x)
+ sigma2 <- sum((y - x%*%coef)^2)/df
+
+ ## calcula  $\sigma^2 * (x'x)^{-1}$ 
+ vcov <- sigma2 * chol2inv(qx$qr)
+ colnames(vcov) <- rownames(vcov) <- colnames(x)
+
+ # resultado
+ list(coefficients = coef, vcov = vcov, sigma = sqrt(sigma2), df = df)
+ }
```

Creación de librerías con R

Ejemplo regresión lineal

```
> head(bass)
```

	Lake	Alkalinity	pH	Calcium	Chlorophyll	Mercury
1	Alligator	5.9	low	3.0	0.7	1.23
2	Annie	3.5	low	1.9	3.2	1.33
3	Apopka	116.0	high	44.1	128.3	0.04
4	Blue Cypress	39.4	high	16.4	3.5	0.44
5	Brick	2.5	low	2.9	1.8	1.20
6	Bryant	19.6	high	4.5	44.1	0.27

Creación de librerías con R

Ejemplo regresión lineal

```
> lmEst(cbind(1, bass$Alkalinity), bass$Mercury)
```

```
$coefficients
```

```
[1] 0.726139976 -0.005301603
```

```
$vcov
```

```
          [,1]      [,2]  
[1,] 2.872843e-03 -3.795770e-05  
[2,] -3.795770e-05 1.011391e-06
```

```
$sigma
```

```
[1] 0.2770542
```

```
$df
```

```
[1] 51
```

```
> lm(Mercury~Alkalinity, data=bass)
```

```
Call:
```

```
lm(formula = Mercury ~ Alkalinity, data = bass)
```

```
Coefficients:
```

```
(Intercept)    Alkalinity  
    0.726140    -0.005302
```

Creación de librerías con R

Ejemplo regresión lineal

Los resultados numéricos son exactamente iguales, pero:

- Formato de los resultados son mejores con *lm*
- Existen utilidades para la estimación de un modelo “summary” que da otras medidas de interés como significación estadística o intervalos de confianza
- Tratar los predictores categóricos
- Utilizar formulas para especificar el modelo (e.g., intercept, interacción, ...)
- La programación orientada a objetos ayuda con los puntos 1 y 2 y las ‘formulas con los puntos 3 y 4.

Creación de librerías con R

Control de argumentos

```
t.test<-function(x, y = NULL, alternative = c("two.sided", "less", "greater"), mu = 0,
paired = FALSE, var.equal = FALSE, conf.level = 0.95, ...)
{
  alternative <- match.arg(alternative)
  if (!missing(mu) && (length(mu) != 1 || is.na(mu)))
    stop("'mu' must be a single number")
  if (!missing(conf.level) && (length(conf.level) != 1 || !is.finite(conf.level) ||
    conf.level < 0 || conf.level > 1))
    stop("'conf.level' must be a single number between 0 and 1")
  if (!is.null(y)) {
    dname <- paste(deparse(substitute(x)), "and", deparse(substitute(y)))
    if (paired)
      xok <- yok <- complete.cases(x, y)
    else {
      yok <- !is.na(y)
      xok <- !is.na(x)
    }
    y <- y[yok]
  }
  else {
    ...
  }
}
```


Creación de librerías con R

Control de argumentos (method=c("OLS", "lm"))

Aproximación sencilla

A

```
lmEst<-function(x, y, method)
{
  if (is.missing(method))
    stop("method should be 'OLS' or 'lm'")
  ...
}
```

B

```
lmEst<-function(x, y, method=NULL)
{
  if (is.null(method))
    stop("method should be 'OLS' or 'lm'")
  ...
}
```

Valores por defecto en los argumentos

```
method.type<-c("OLS", "lm")
m<-charmatch(method, method.type, nomatch=0)
if (m==0)
```

Posibilidad 1: stop("method should be 'OLS' or 'lm'")

Posibilidad 2: warning(paste("method=", method, "is not supported. Using 'OLS'"))
 method<-1

Despues: if (method==1) { llama lmEst }
 if (method==2) { llama lm }

Creación de librerías con R

```
> lmEst1 <- function(x, y, method="OLS")
+ {
+   #control errores
+   method.type<-c("OLS", "lm")
+   m<-charmatch(method, method.type, nomatch=0)
+   if (m==0)
+     stop("method should be 'OLS' or 'lm'")
+   if (m==1) {
+     ... Hace lo de antes
+   # resultado
+   list(coefficients = coef, vcov = vcov, sigma = sqrt(sigma2), df = df)
+ }
+ if (m==2) {
+   ans <-lm(x ~ y)
+ }
+   ans
+ }
> lmEst1(cbind(1, bass$Alkalinity), bass$Mercury, method="qr")
Error en lmEst1(cbind(1, bass$Alkalinity), bass$Mercury, method = "qr") :
  method should be 'OLS' or 'lm'
```

Creación de librerías con R

```
> lmEst2 <- function(x, y, method="OLS")
+ {
+
+ #control errores
+ method.type<-c("OLS", "lm")
+ m<-charmatch(method, method.type, nomatch=0)
+ if (m==0)
+ warning(paste("method=", method, "is not supported. Using 'OLS'"))
+ m<-1
+
+ ...
+
+ # resultado
+ list(coefficients = coef, vcov = vcov, sigma = sqrt(sigma2), df = df)
+ }
```

Creación de librerías con R

```
> lmEst2(cbind(1, bass$Alkalinity), bass$Mercury, method="qr")
$coefficients
[1] 0.726139976 -0.005301603

$vcov
           [,1]      [,2]
[1,] 2.872843e-03 -3.795770e-05
[2,] -3.795770e-05 1.011391e-06

$sigma
[1] 0.2770542

$df
[1] 51

Warning message:
In lmEst2(cbind(1, bass$Alkalinity), bass$Mercury, method = "qr") :
  method= qr is not supported. Using 'OLS'
```

Creación de librerías con R

```
> lmEst3 <- function(x, y, method="OLS")
+ {
+   #control errores
+   method.type<-c("OLS", "lm")
+   m<-charmatch(method, method.type, nomatch=0)
+   if (m==0)
+     warning(paste("method=", method, "is not supported. Using 'OLS'"))
+   method<-1
+
+   if (missing(x) | missing(y))
+     stop("x and y arguments are required")
+
+   if (nrow(x) !=length(y))
+     stop("x and y should have the same length")
+
+   if (any(is.na(x)) | any(is.na(y)))
+   {
+     warning("There are missing values. Only complete cases have been
analyzed")
+     o<-complete.cases(x,y)
+     x<-x[o,]
+     y<-y[o]
+   }
```

Creación de librerías con R

```
> lmEst3(cbind(1, bass$Alkalinity))
Error en lmEst3(cbind(1, bass$Alkalinity)) :
  x and y arguments are required

> lmEst3(cbind(1, bass$Alkalinity), bass$Mercury[-1])
Error en lmEst3(cbind(1, bass$Alkalinity), bass$Mercury[-1]) :
  x and y should have the same length

> lmEst3(cbind(1, bass$Alkalinity), bass$Mercury)
$coefficients
[1] 0.707061260 -0.005071191
$vcov
      [,1]      [,2]
[1,] 2.797887e-03 -3.686621e-05
[2,] -3.686621e-05  9.666413e-07
$sigma
[1] 0.2690298
$df
[1] 50
Warning message:
In lmEst3(cbind(1, bass$Alkalinity), y) :
  There are missing values. Only complete cases have been analyzed
```

Creación de librerías con R

Uso de `...`

Se utiliza para no poner todos los argumentos de otras funciones pero que podamos cambiarlos si fuera necesario. Por ejemplo descomposición QR

```
qr.solve(a, b, tol = 1e-7)
```

Se podría intentar cambiar este argumento:

```
> lmEst(cbind(1, bass$Alkalinity), bass$Mercury, tol=1e-8)
Error en lmEst(cbind(1, bass$Alkalinity), bass$Mercury, tol
= 1e-08) :
  unused argument(s) (tol = 1e-08)
```

SOLUCION: en la función lmEst poner

```
# calcula  $(x'x)^{-1}x'y$ 
coef <- solve.qr(qx, y, ...)
```

RECOMENDACION: En nuestra función añadir este argumento si se cree que se va a llamar desde otra función

```
lmEst <- function(x, y, ...)
```

Ejercicio

Crear una función (compareGroups) que:

1. Tenga como argumentos: 1) 'X' un data.frame con muchas variables resultados (distintos tipos: continuas normales, continuas discretas, categóricas, ..) y 2) un vector 'y' que defina dos grupos de individuos
2. La función debe empezar por comprobar que:
 1. Las dimensiones cuadran: número de filas de 'X' igual al tamaño de 'y'
 2. 'y' es factor
 3. El número de niveles de 'y' es 2 como máximo
3. Para cada variable de X haga:
 1. La media y desviación típica para cada grupo de 'y' para las variables continuas en 'X' y el número de casos y % para cada grupo de 'y' para las variables categóricas
 2. Calcule el p-valor usando el test adecuado
4. Pensar qué debería devolver la función ...

Creación de librerías con R

Uso de 'formula'

Elegante en cuanto a formulación ya que es similar a como escribiríamos el modelo incluyendo covariables o interacciones

Hace que tu función sea fácil de usar ya que utiliza la sintaxis similar a otras funciones que hay por defecto en R (lm, glm, coxph, rpart, ...)

Evita tener que especificar el vector o matriz de datos, ya que los toma del argumento 'data'

Útil para analizar datos completos sin tener que controlarlo dentro de la función

Su formulación es la siguiente:

```
lmEst<-function(formula, data, contrasts=NULL,...)
```

Creación de librerías con R

Uso de 'formula'

```
cl <- match.call()
```

match.call se usa para:

- guardar una llamada y usarla después

```
> lm(Mercury~Alkalinity, data=bass)
```

Call:

```
lm(formula = Mercury ~ Alkalinity, data = bass)
```

Coefficients:

(Intercept)	Alkalinity
0.726140	-0.005302

- El objeto más importante es "model.frame" que es un 'data frame' con sólo las variables que aparecen en la formula y que tiene un atributo que se llama 'terms' que nos dice cuál es la variable dependiente, el intercept, ...

```
mf <- model.frame(formula=formula, data=data)
```

```
mt <- attr(mf, "terms")
```

```
x <- model.matrix(mt, data=mf)
```

```
y <- model.response(mf)
```

Hay mas cosas pero esto es suficiente en la mayoría de ocasiones

Creación de librerías con R

```
> unclass(mf.copia)
$Mercury
 [1] 1.23 1.33 0.04 0.44
...
$Alkalinity
 [1] 5.9 3.5 116.0
39.4 ...
attr(,"terms")
Mercury ~ Alkalinity
attr(,"variables")
list(Mercury, Alkalinity)
attr(,"factors")
      Alkalinity
Mercury          0
Alkalinity       1
attr(,"term.labels")
[1] "Alkalinity"
attr(,"order")
[1] 1
attr(,"intercept")
[1] 1
attr(,"response")
[1] 1
```

```
attr(,".Environment")
<environment: R_GlobalEnv>
attr(,"predvars")
list(Mercury, Alkalinity)
attr(,"dataClasses")
      Mercury Alkalinity
"numeric"    "numeric"
attr(,"row.names")
 [1] 1 2 3 4 5 6 7 8 9
10 11 12 13 14 15 16 17 18 19 20
21 22 23 24
[25] 25 26 27 28 29 30 31 32 33
34 35 36 37 38 39 40 41 42 43 44
45 46 47 48
[49] 49 50 51 52 53
```

↑
Controla los datos completos. Interesante para comparar modelos.

Creación de librerías con R

Uso de 'formula'

```
> lmEst4(Mercury~Alkalinity, bass)
Error en qr(x) : objeto 'x' no encontrado
```

Ahora debemos recuperar los datos que nos interesan (x, y) match.call se usa para:

```
y <- model.response(mf)
x <- model.matrix(mt, mf, contrasts)
```

Existen otras para otros argumentos que hay útiles cuando se usa 'formula'

model.weights, model.offset, model.frame, ...

model.extract(frame, component) -> component: "weights", "offset"

Creación de librerías con R

Uso de 'formula'

1ª ventaja !!!

```
> lmEst4(Mercury~Alkalinity, bass)
```

```
$coefficients
```

```
(Intercept)    Alkalinity
```

```
0.726139976 -0.005301603
```

```
$vcov
```

	(Intercept)	Alkalinity
(Intercept)	2.872843e-03	-3.795770e-05
Alkalinity	-3.795770e-05	1.011391e-06

```
$sigma
```

```
[1] 0.2770542
```

```
$df
```

```
[1] 51
```

Creación de librerías con R

Uso de 'formula'

```
> lmEst4(Mercury~Alkalinity-1, bass)
```

```
$coefficients
```

```
Alkalinity
```

```
0.004292588
```

```
$vcov
```

```
Alkalinity
```

```
Alkalinity 2.299709e-06
```

```
$sigma
```

```
[1] 0.5883974
```

```
$df
```

```
[1] 52
```

Creación de librerías con R

Uso de 'formula' – variable factor

```
> lmEst4(Mercury~pH, bass)
```

```
$coefficients
```

(Intercept)	pHlow
0.3800000	0.3545455

```
$vcov
```

	(Intercept)	pHlow
(Intercept)	0.002802243	-0.002802243
pHlow	-0.002802243	0.006750857

```
$sigma
```

```
[1] 0.2947364
```

```
$df
```

```
[1] 51
```

Creación de librerías con R

Uso de 'formula' – argumentos especiales (strata en survival)

Uso de 'formula' – control del término interacción

Uso de 'formula' – inclusion de todas las variables en data

Creación de librerías con R

Uso de 'formula' – inclusion de todas las variables en data

Podemos hacer que el modelo incluya todas las variables que hay en el argumento 'data' usando ~.

```
> lmEst4(Mercury~., bass[, -1])
$coefficients
  (Intercept)   Alkalinity      pHlow      Calcium Chlorophyll
  0.644868568 -0.005606862  0.131985790  0.004529969 -0.002709375
$vcov
              (Intercept)   Alkalinity      pHlow      Calcium
(Intercept)  8.077564e-03 -6.889535e-05 -7.119053e-03 -2.226835e-05
Alkalinity   -6.889535e-05  3.569650e-06  6.285224e-05 -3.601438e-06
pHlow        -7.119053e-03  6.285224e-05  9.423046e-03  1.405910e-05
Calcium      -2.226835e-05 -3.601438e-06  1.405910e-05  6.884240e-06
Chlorophyll  -3.283876e-05 -4.847165e-07  2.321261e-05 -5.398429e-08
              Chlorophyll
(Intercept) -3.283876e-05
Alkalinity   -4.847165e-07
pHlow        2.321261e-05
Calcium      -5.398429e-08
Chlorophyll  1.842515e-06
$sigma
[1] 0.2607964
$df
[1] 48
```

Creación de librerías con R

Resultado de una función

La función devuelve lo último que se ejecuta si no se asigna ningún objeto

```
list(coefficients = coef, vcov = vcov, sigma =  
sqrt(sigma2), df = df)
```

Si se asigna algún objeto luego hay que llamarlo o usar 'return'

```
ans<-list(coefficients = coef, vcov = vcov, sigma =  
sqrt(sigma2), df = df)  
ans
```

ó

```
ans<-list(coefficients = coef, vcov = vcov, sigma =  
sqrt(sigma2), df = df)  
return(ans)
```

Creación de librerías con R

Resultado de una función

Veremos que se puede definir una clase al resultado para luego usar algunos métodos (print, summary, plot, anova, ...)

El resultado puede ser

Un número

Un vector

Una lista (vectores, matrices, números, ...)

```
> mod<-lm(Mercury~Alkalinity, bass)
> names(mod)
[1] "coefficients" "residuals"      "effects"        "rank"
[5] "fitted.values" "assign"          "qr"             "df.residual"
[9] "xlevels"       "call"           "terms"          "model"
> mod$coef
(Intercept)    Alkalinity
0.726139976 -0.005301603
> mod$residuals[1:10]
      1      2      3      4      5      6
0.53513948 0.62241564 -0.07115402 -0.07725681 0.48711403 -0.35222856
      7      8      9     10
-0.21857164 -0.15760551 0.24382235 0.10930772
> mod$call
lm(formula = Mercury ~ Alkalinity, data = bass)
```

Creación de librerías con R

Resultado de una función

Pero puede tener atributos

```
> attributes(mod)
$names
[1] "coefficients" "residuals"      "effects"        "rank"
[5] "fitted.values" "assign"          "qr"             "df.residual"
[9] "xlevels"       "call"           "terms"          "model"

$class
[1] "lm"
```

En una función se hace

```
ans<-list(coefficients = coef, vcov = vcov, sigma = sqrt(sigma2), df = df)
attr(ans, "contrasts") <- contrasts
ans
```

**La utilidad es para luego hacer cosas con ese objeto (print, summary, ...)
en función de los valores del atributo**

Creación de librerías con R

Resultado de una función

Otra forma es guardar el valor de una variable en el Workspace (variable global??)

```
> ff<-function(x, y)
+ {
+   xx <- x+y
+   assign("xx", xx, env=.GlobalEnv)
+   z <- xx - 9
+   z
+ }
>
>
> ff(5,6)
[1] 2
> xx
[1] 11
```

No muy recomendable (también funciona `xx <- x+y`), sólo útil para 'debugging'

Ejercicio

Mejorar la función `compareGroups`:

1. Que el argumento de entrada sea una fórmula
2. Ya no es necesario que tenga 'X' e 'y' ya que 'y' puede estar en el mismo `data.frame` (argumento 'data')
3. Extrae 'X' e 'y' del argumento 'formula' y 'data'
4. El resto sigue igual

Sesión 3 – Conexión con C y Fortran

- Creación de dll's
- Llamadas a dll's

Creación de librerías con R

Mejora de la eficiencia en R: uso de "dlls"

- Velocidad
- Eficiencia en el uso de memoria
- Uso de librerías existente en Fortran o C (BLAS, LAPACK, ...)
- Se usan las siguientes funciones en R para llamar a estas dlls:
 - .Fortran ó .C
 - .Call
 - .External
- Uso de .Fortran ó .C
 - Llamar a Fortran o C con vectores de diferente tipo
- Uso de .Call
 - Enviar vectores enteros/carácteres de R a Fortran ó C
 - Conseguir un vector de enteros/carácteres o una lista de Fortran ó C
- Para compilar un código Fortran ó C
 - R CMD SHLIB prog.c y crea prog.so (acepta varias .c ó .f)
 - R CMD SHLIB -o prog.dll prog.f
 - g77 -o prog.dll -shared prog.f
 - gcc -o prog.dll -shared prog.f
- Se carga con `dyn.load("nombre.dll")` y se descarga con `dyn.unload("nombre.dll")`
- Para ver si hay alguna subrutina `is.loaded("nombre")`

Creación de librerías con R

Mejora de la eficiencia en R: uso de "dlls"

- Los tipos de variables deben coincidir (coerción)

R storage mode	C type	FORTRAN type
logical	int *	INTEGER
integer	int *	INTEGER
double	double *	DOUBLE PRECISION
complex	Rcomplex *	DOUBLE COMPLEX
character	char **	CHARACTER*255
raw	char *	none

- Hay que guardar en memoria los vectores que pasaremos a .Fortran en R creando vectores con el tamaño correcto
- El primer argumento de .Fortran es un carácter con el nombre de la subrutina
- El resto de argumentos son objetos de R que se pasan a la función
- .Fortran devuelve un objeto tipo lista

```
subroutine fun(n,b,c,d)
  integer n
  double precision b(n),c(2), d(10)
```

```
> n<-3
> b<-c(23,11,-9)
> c<-c(1,4)
> out <- .Fortran("fun", as.integer(n), as.double(b), as.double(c), as.double(rep(0,10)))
```

Creación de librerías con R

Mejora de la eficiencia en R: uso de "dlls"

¿cómo hacer el programa en Fortran? (subroutine) ¿y en C (void)?

ejemplo, el archivo prog.f contiene el código:

```
subroutine prod(a,b,c)
double precision a,b,c
c=a*b
end subroutine
```

Creamos la .dll con `system("gcc -shared -o prog.dll prog.f")` [Linux .so]
`system("R SHLIB -o prog.dll prog.f")`

- En R:

```
> dyn.load("fortran/prog.dll")
> a<-4
> b<-6
> out <- .Fortran("prod", a = as.double(a),
b=as.double(b), c=as.double(0))
> out$c
[1] 24
> out[[3]] [1] 24
```

Creación de librerías con R

Mejora de la eficiencia en R: uso de "dlls"

- ¿y si tenemos una función? -> hay que hacer un "wrapper"

ejemplo, el archivo prog2.f contiene el código:

```
double precision function fun(a,b)
double precision a, b
  fun=a*b
end function
```

```
subroutine prod(a,b,c)
double precision a,b,c
c=fun(a,b)
end subroutine
```

- En R:

```
> dyn.load("fortran/prog2.dll")      #nota: prog2.so para Linux!!!
> a<-4
> b<-6
> out <- .Fortran("prod", a = as.double(a), b=as.double(b),
c=as.double(0))
> out$c
[1] 24
> out[[3]]
[1] 24
```

Creación de librerías con R

Mejora de la eficiencia en R: uso de "dlls"

```
/* use.c                                     */

void ex(int *i, double *d, char **c, int *l) {
    i[0] = 11;
    d[0] = 2.333;
    c[1] = "g";
    l[0] = 0;
}

> dyn.load("use.so")
> i <- 1:10                # integer vector
> d <- seq(length=3,from=1,to=2) # real
number vector
> c <- c("a", "b", "c")    # string vector
> l <- c("TRUE", "FALSE")  # logical vector
> i
[1] 1 2 3 4 5 6 7 8 9 10
> d
[1] 1.0 1.5 2.0
> c
[1] "a" "b" "c"
> l
[1] "TRUE" "FALSE"
```

```
> out <- .C("ex",
            i1 = as.integer(a),
            d1 = as.numeric(d),
            c1 = as.character(c),
            l1 = as.logical(l))
> out
$i1
[1] 11 2 3 4 5 6 7 8 9 10
$d1
[1] 2.333 1.500 2.000
$c1
[1] "a" "g" "c"
$l1
[1] FALSE FALSE
```

Creación de librerías con R

Mejora de la eficiencia en R: uso de "dlls"

- A la hora de desarrollar un paquete en R:
 - Copiar el código fuente (*.c ó *.f) en myPackage/src/
 - El usuario no debe cargar manualmente el código compilado con `dyn.load()` añadir un archivo `firstlib.R` ó `zzz.R` a myPackage/R con el código:

```
.First.lib <-function (lib, pkg)
{
  library.dynam("myPackage", pkg, lib)
}
```

- Modificar la llamada a `.Fortran` añadiendo `PACKAGE="archivo_compilado"`.
Por ejemplo, si el archivo compilado se llama `prog.dll`
`out <- .Fortran("suma", a = as.double(a), b=as.double(b),
c=as.double(0), PACKAGE="prog")`

Creación de librerías con R

Aplicación a nuestra función

- Existe una subrutina en C que hace lo mismo que qr.solve (**dqrcf**)

```
> is.loaded("dqrcf")
```

TRUE

En Fortran subroutine dqrcf(x, n, k, graux, y, ny, b, info)

```
subroutine dqrcf(x, n, k, graux, y, ny, b, info)

    integer n, k, ny, info
    double precision x(n,k), graux(k), y(n,ny), b(k,ny)
    integer j
    double precision dummy(1)
    do 10 j = 1,ny
        call dqrs1(x, n, n, k, graux, y(1,j), dummy,
&                y(1,j), b(1,j), dummy, dummy, 100, info)
    10 continue
    return
end
```

Creación de librerías con R

Aplicación a nuestra función

```
lmEst5 <- function(formula, data, contrasts=NULL, ...)  
{  
  ...  
  # calcula la descomposicion QR de x  
  qx <- qr(x)  
  
  # calcula  $(x'x)^{-1} x'y$   
  n <- as.integer(nrow(qx$qr))  
  k <- as.integer(qx$rank)  
  ny <- as.integer(1)  
  
  z <- .Fortran("dqrclf", as.double(qx$qr), n, k, as.double(qx$qlraux),  
    as.double(matrix(y, nrow=n, ncol=ny)), as.integer(ny), coef = matrix(0,  
    nrow = k, ncol = ny), info = as.integer(0),  
    NAOK = TRUE, PACKAGE = "base")  
  
  coef <- z$coef  
  info <- z$info  
  
  ...  
  
  # resultado  
  list(coefficients = coef, vcov = vcov, sigma = sqrt(sigma2), df = df, info=info)
```

Creación de librerías con R

Aplicación a nuestra función

```
> lmEst5(Mercury ~ Alkalinity, bass)
```

```
$coefficients [,1]  
[1,] 0.726139976  
[2,] -0.005301603
```

```
$vcov  
              (Intercept) Alkalinity  
(Intercept) 0.0028728426 -3.795770e-05  
Alkalinity -0.0000379577 1.011391e-06
```

```
$sigma [1] 0.2770542  
$df [1] 51
```

```
$info [1] 0
```


Creación de librerías con R

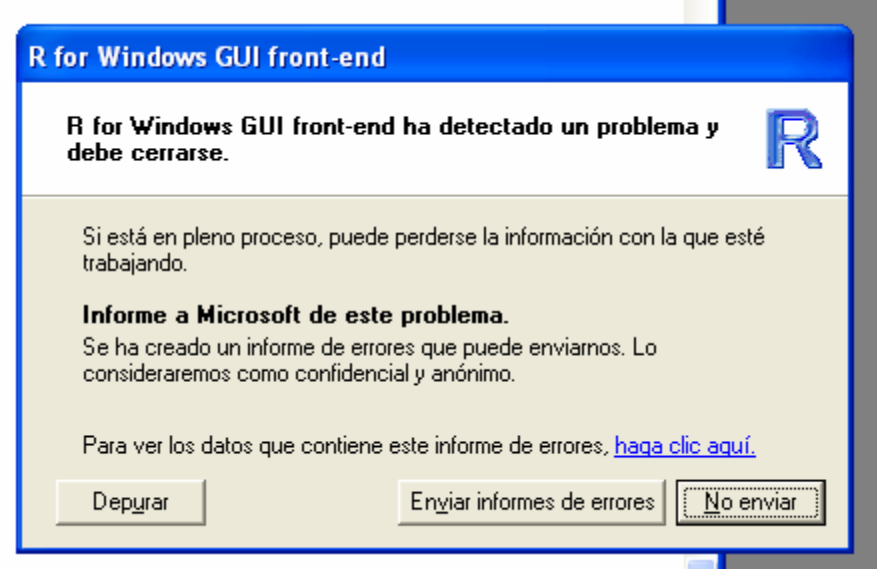
Debugging

```
lmEst6 <- function(formula, data, contrasts=NULL, ...)  
{  
  ...  
  
  z <- .Fortran("dqrcf", as.integer(qx$qr), n, k, as.double(qx$qraux),  
    as.double(matrix(y, nrow=n, ncol=ny)), as.integer(ny), coef = matrix(0,  
    nrow = k, ncol = ny), info = as.integer(0),  
    NAOK = TRUE, PACKAGE = "base")  
  
  ...  
}  
  
> lmEst6(Mercury~Alkalinity, bass)  
$coefficients  
      [,1]  
[1,]  NaN  
[2,]  NaN  
  
$vcov  
  
      (Intercept) Alkalinity  
(Intercept)    NaN        NaN  
Alkalinity     NaN        NaN  
  
$sigma  
[1] NaN
```

Creación de librerías con R

Debugging

```
lmEst7 <- function(formula, data, contrasts=NULL, ...)  
{  
  ...  
  
  z <- .Fortran("dqrcf", as.double(qx$qqr), n, k, as.double(qx$qqraux),  
    as.double(matrix(y, nrow=n, ncol=ny)), as.integer(n), coef = matrix(0,  
    nrow = k, ncol = ny), info = as.integer(0),  
    NAOK = TRUE, PACKAGE = "base")  
  
  ...  
}  
  
> lmEst7(Mercury~Alkalinity, bass)
```



Creación de librerías con R

Cuando falla !!!!

Windows !!!!!

Linux !!!!

Emulador !!!!

Chivatos !!!!

Sesión 4 – Métodos y clases en R

- Introducción
- Orientación a objetos
 - Clases
 - Objetos
 - Métodos
 - Ejemplos

Ejercicio

1. Crea una dll con el código implementado en la función 'prog.f'
2. Carga la dll en R
3. Llama a esa dll para calcular cuanto vale $1345 + 456$

Creación de librerías con R

Introducción

Preguntas:

- Cuando tenemos un objeto en el que hemos estimado un modelo de regresión lineal ¿qué parámetros estamos interesados en conocer?
- Después podemos estar interesados en otra información (parámetros) ¿por ejemplo?
- ¿Finalmente qué hacemos para validar un modelo?
- ¿sabéis cómo se consigue esta información y qué tipo de información nos da?
print, summary, plot
- Y lo más importante. Si el objeto que quiero inspeccionar es por ejemplo un modelo de supervivencia, ¿obtenemos la misma información?
- Esto es lo que permite hacer la Programación Orientada a Objetos

Creación de librerías con R

```
> mod<-lm(uptake~conc, data=CO2)
> mod
```

Call:

```
lm(formula = uptake ~ conc, data = CO2)
```

Coefficients:

(Intercept)	conc
19.50029	0.01773

```
> summary(mod)
```

Call:

```
lm(formula = uptake ~ conc, data = CO2)
```

Residuals:

Min	1Q	Median	3Q	Max
-22.831	-7.729	1.483	7.748	16.394

Coefficients:

	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	19.500290	1.853080	10.523	< 2e-16 ***
conc	0.017731	0.003529	5.024	2.91e-06 ***

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 9.514 on 82 degrees of freedom

Multiple R-Squared: 0.2354, Adjusted R-squared: 0.2261

F-statistic: 25.25 on 1 and 82 DF, p-value: 2.906e-06

Creación de librerías con R

```
> mod<-lm(uptake~conc, data=CO2)
> mod
```

Call:

```
lm(formula = uptake ~ conc, data = CO2)
```

Coefficients:

(Intercept)	conc
19.50029	0.01773

```
> mod2<-coxph(Surv(time,status)~nodes,data=colon)
> mod2
```

Call:

```
coxph(formula = Surv(time, status) ~ nodes, data = colon)
```

	coef	exp(coef)	se(coef)	z	p
nodes	0.0868	1.09	0.00626	13.9	0

Likelihood ratio test=133 on 1 df, p=0 n=1822

Creación de librerías con R

Ejemplo: la función summary

```
> names(CO2)
```

```
[1] "Plant"      "Type"      "Treatment" "conc"      "uptake"
```

```
> summary(CO2$uptake)
```

```
Min. 1st Qu. Median Mean 3rd Qu. Max.
 7.70  17.90  28.30 27.21  37.12 45.50
```

```
> class(CO2$uptake)
```

```
[1] "numeric"
print.summary.numeric
```

```
> summary(CO2$Plant)
```

```
Qn1 Qn2 Qn3 Qc1 Qc3 Qc2 Mn3 Mn2 Mn1 Mc2 Mc3 Mc1
  7   7   7   7   7   7   7   7   7   7   7   7
```

```
> class(CO2$Plant)
```

```
[1] "factor"
print.summary.factor
```

```
> summary(CO2)
```

	Plant	Type	Treatment	conc	uptake
Qn1	: 7	Quebec	:42	nonchilled:42	Min. : 95
Qn2	: 7	Mississippi	:42	chilled :42	1st Qu.: 175
Qn3	: 7				Median : 350
Qc1	: 7				Mean : 435
Qc3	: 7				3rd Qu.: 675
Qc2	: 7				Max. : 1000
(Other)	:42				

Creación de librerías con R

Orientación a objetos

- Es un sistema para abstraer datos en programas
- La forma de implementar este sistema puede ser muy distinto entre lenguajes
- Lenguajes que soportan programación orientada a objetos pueden ser: Java, C++, Python, Lisp, Perl
- R tiene implementados dos sistemas de orientación a objetos "S3" y "S4" *classes and methods*
- S3 y S4 son sistemas independientes y funcionan de forma separada (S4 esta implementada en la librería métodos, es muy usada en Bioconductor)

- Una clase es una descripción de una "cosa"
 - ej., *modelo lineal, data frame, matriz, proceso puntual, anova, ...*
- Un objeto es una instancia de una clase
 - ej., `x<-matrix(1:9, nrow=3, ncol=3)` `y<-matrix(5, nrow=4, ncol=2)`
x e y son dos objetos de clase "matrix"
- ¿cómo definir la clase de un objeto? Mediante la función `class`
 - ej., *En una función al final al objeto que devuelva le damos un nombre*
`miFuncion<-function(x)`
 {
 `ans<-x[length(x)]-x[1]`
 `class(ans)<-"range"`
 `ans`
 }

Creación de librerías con R

Orientación a objetos

- Una función genérica permite una “interface” para un cálculo particular
 - El cálculo es “genérico” ya que puede tener diferentes significados para diferentes clases
 - ej. print, summary, confidence intervals, prediction, mean
- Un método implementa un cálculo para una clase particular
- La idea básica es:
 - Si *pepe()* es una función genérica, entonces si llamamos a *pepe()* con un objeto *x* puede producir una salida completamente diferente a la salida obtenida llamando a la misma función con un objeto *y*, siempre y cuando *x* e *y* sean de clases diferentes
 - ej. En el ejemplo de “lm” y “coxph” la función print() daba cosas distintas
- ¿cómo identificamos funciones genéricas?

```
> print
function (x, ...)
  UseMethod("print")
<environment: namespace:base>
> summary
function (object, ...)
  UseMethod("summary")
<environment: namespace:base>
> deviance
function (object, ...)
  UseMethod("deviance")
<environment: namespace:stats>
```

Creación de librerías con R

Orientación a objetos

- Funciones útiles
 - *methods()* muestra los métodos existentes para una función genérica o para una clase dada
 - ej. `> methods(deviance)`
`[1] deviance.default* deviance.glm* deviance.lm* deviance.mlm*`
`[5] deviance.nls* (probar methods(summary))`
 - Non-visible functions are asterisked
 - *getAnywhere()* busca en cualquier lugar una función deseada
- Búsqueda de métodos
 - Cuando una función genérica *ff* es llamada por un objeto de clase *car*, R busca la función *ff.car()*
 - Cuando el método apropiado no se encuentra, entonces R busca la función *ff.default()*
 - Si no se encuentra ningún método por defecto, entonces aparece un error
- Hay otros requerimientos que se hacen por convención (se “chequean”):
 - Un método debe tener todos los argumentos que el genérico, incluyendo “...”
 - Si un genérico tiene argumentos por defecto, todos los métodos deben tener esos mismos argumentos (e.g. si hacemos `print.myFunction` debe tener `x`, ‘...’) (*investigar deviance*)
 - Un método debe tener todos los argumentos exactamente en el mismo orden que el genérico

Creación de librerías con R

Orientación a objetos

- IDEA: Si creamos una función que devuelva un objeto de una clase nueva (p.e. `myFunction`) que hemos creado, entonces estaría bien crear “como mínimo” las funciones `print.myFunction`, `summary.myFunction`, `plot.myFunction` que son las más usuales
- NOTA sobre `summary()`: los métodos para `summary()` no deberían “printar” nada. Se utiliza para calcular mas cosas (**ver `summary.lm`**). Debemos crear una función `print.summary.myFunction()`

Creación de librerías con R

Orientación a objetos (ejemplo)

```
myrange<-function(x)
```

```
{  
  x<-sort(x)  
  ans<-x[length(x)]-x[1]  
  out<-list(x=x, rango=ans)  
  class(out)<-"myrange"  
  out  
}
```

```
print.myrange<-function(x)
```

```
{  
  cat("El rango de tus valores es: \n")  
  cat(x$rango)  
  cat("\n")  
}
```

```
> myrange(c(1,5,6,9,2,19,3))
```

```
El rango de tus valores es:  
18
```

```
out  
$x  
[1] 1 2 3 5 6 9 19  
  
$rango  
[1] 18
```

Creación de librerías con R

Definición de tus propias clases y métodos

Supongamos que nos “inventamos” un nuevo test para evaluar asociación entre dos variables categóricas (y que lo tenemos implementado en una función que se llama “fisher.test”)

Los datos a los que podemos pasar este test podrían ser:

- Dos vectores
- Una tabla
- Una matriz

Simplemente aplicando ‘myTest’ a: dos vectores, una tabla o una matriz, debería hacer el cálculo

```
myTest<-function(x,...)
  UseMethod("myTest")

myTest.default<-function(x, y, ...)
{
  # ... algunos controles
  xx<-table(x,y)
  myTest.table(xx, ...)
}

myTest.table<-function(x,...) {
  # ... algunos controles
  myTest.matrix(x, ...)
}

myTest.matrix<-function(x, ...) {
  # ...programa con todos los cálculos
  # implementa la teoría ...
  ans<-fisher.test(x, ...)
  class(ans)<-"myTest"
  ans
}

print.myTest<-function(x,...)
{
  cat("Results for my new test \n")
  cat(" p-value of 'new test':",
x$p.value)
  cat("\n")
}
```

Creación de librerías con R

```
> set.seed(123456)
> casco<-sample(c("ca","co"), 200, replace=TRUE)
> fuma<-sample(c("si","no"), 200, replace=TRUE)
>
> tt<-table(casco, fuma)
> tt
```

	fuma	
casco	no	si
ca	52	46
co	43	59

```
> myTest(casco, fuma)
Results for my new test
p-value of 'new test': 0.1565684

> myTest(tt)
Results for my new test
p-value of 'new test': 0.1565684

> mm<-matrix(tt, nrow=2, ncol=2)
> myTest(mm)
Results for my new test
p-value of 'new test': 0.1565684
```


Creación de librerías con R

Las funciones genéricas también deberían tener controles (o si no NAMESPACE ...)

```
print.myTest<-function(x,...)
{
  if(!inherits(x, "myTest"))
    stop("x should be of class 'myTest'")

  cat("Results for my new test \n")
  cat(" p-value of 'new test':", x$p.value)
  cat("\n")
}
```

Creación de librerías con R

Volvemos al ejemplo de regresión lineal

definimos un método nuevo

```
lmMod <- function(x, ...) UseMethod("lmMod")
```

añadimos un metodo por defecto 'lmMod.default'

```
lmMod.default <- function(x, y, ...)
```

```
{
```

controles !!!

```
x <- as.matrix(x)
```

```
y <- as.numeric(y)
```

```
ans <- lmEst(x, y)
```

mas cálculos utiles

```
ans$fitted.values <- as.vector(x %*% est$coefficients)
```

```
ans$residuals <- y - ans$fitted.values
```

```
ans$call <- match.call()
```

definimos la clase

```
class(ans) <- "lmMod"
```

ans # devuelve una lista!!!

```
}
```

Creación de librerías con R

Volvemos al ejemplo de regresión lineal

definimos 'print'

```
print.lmMod <- function(x, ...)
{
  cat("Call:\n")
  print(x$call)
  cat("\nCoefficients:\n")
  print(x$coefficients)
}
```

```
> lmMod(cbind(1, bass$Alkalinity), bass$Mercury)
```

Call:

```
lmMod.default(x = cbind(1, bass$Alkalinity), y = bass$Mercury)
```

Coefficients:

```
[1] 0.726139976 -0.005301603
```

```
> lmMod(cbind(Const=1, Alk=bass$Alkalinity), bass$Mercury)
```

Call:

```
lmMod.default(x = cbind(Const = 1, Alk = bass$Alkalinity), y = bass$Mercury)
```

Coefficients:

Const	Alk
0.726139976	-0.005301603

Creación de librerías con R

Otras ventajas de usar métodos/classes

Se pueden usar otras funciones genéricas si se adaptan al 'default'

```
> mod<-lmMod(cbind(Const=1, Alk=bass$Alkalinity),  
bass$Mercury)  
> coef(mod)  
          Const          Alk  
0.726139976 -0.005301603  
> fitted(mod)  
[1] 0.69486052 0.70758436 0.11115402 0.51725681 ...  
> resid(mod)  
[1] 0.53513948 0.62241564 -0.07115402 -0.07725681 ...
```

Creación de librerías con R

Otras ventajas de usar métodos/classes

Se pueden usar otras funciones genéricas si se adaptan al 'default'

```
> getAnywhere(coef.default)
A single object matching 'coef.default' was found
It was found in the following places
  registered S3 method for coef from namespace stats
  namespace:stats with value

function (object, ...)
  object$coefficients

<environment: namespace:stats>
```

Creación de librerías con R

Summary y print.summary

```
summary.lmMod <- function(object, ...)  
{  
  se <- sqrt(diag(object$vcov))  
  tval <- coef(object) / se  
  TAB <- cbind(Estimate = coef(object),  
    StdErr = se,  
    t.value = tval,  
    p.value = 2*pt(-abs(tval), df=object$df))  
  res <- list(call=object$call,  
    coefficients=TAB)  
  class(res) <- "summary.lmMod"  
  res  
}
```

```
print.summary.lmMod <- function(x, ...)  
{  
  cat("Call:\n")  
  print(x$call)  
  cat("\n")  
  printCoefmat(x$coefficients, P.value=TRUE, has.Pvalue=TRUE)  
}
```

Función útil (Ver ayuda)



Creación de librerías con R

Summary

```
> mod<-lmMod(cbind(Const=1, Alk=bass$Alkalinity), bass$Mercury)
```

```
> summary(mod)
```

Call:

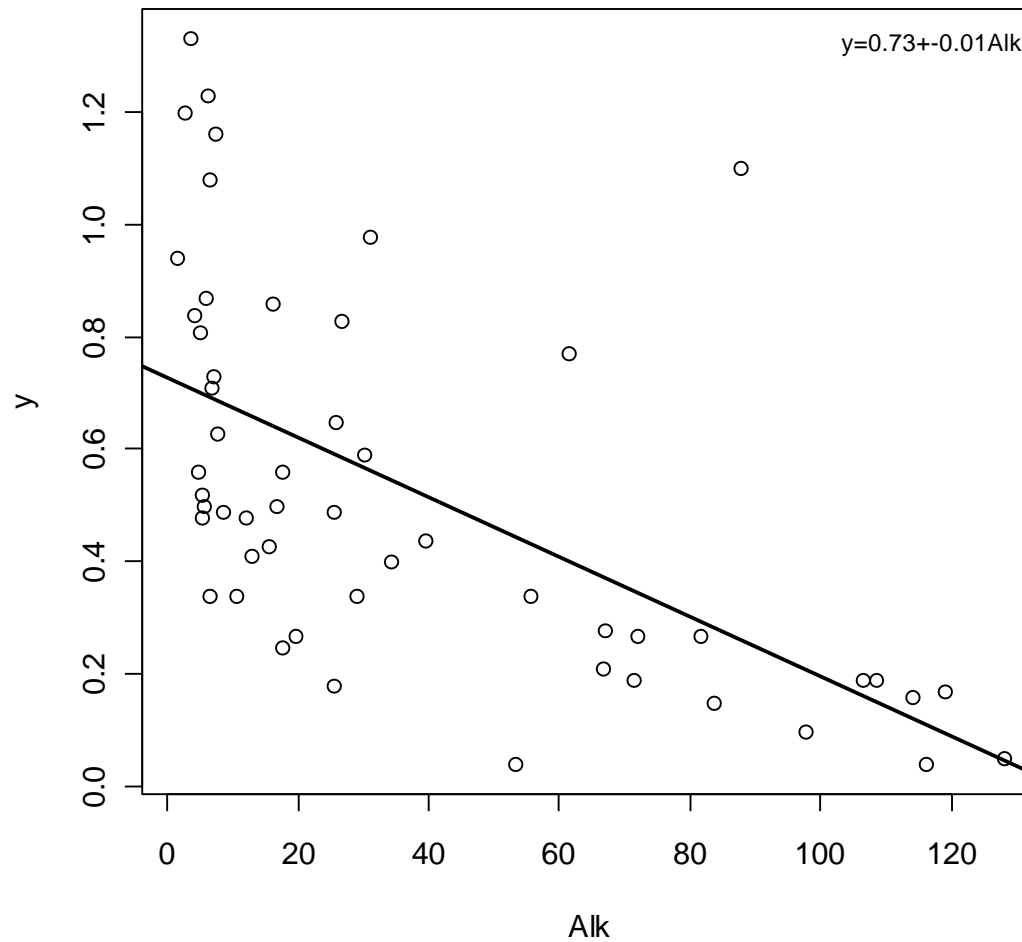
```
lmMod.default(x = cbind(Const = 1, Alk = bass$Alkalinity), y =  
bass$Mercury)
```

	Estimate	StdErr	t.value	p.value
Const	0.7261400	0.0535989	13.5477	< 2.2e-16 ***
Alk	-0.0053016	0.0010057	-5.2717	2.763e-06 ***

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Creación de librerías con R

plot



Creación de librerías con R

plot

```
> names(mod)
```

```
[1] "coefficients" "vcov" "sigma" "df"  
"fitted.values" "residuals" "call"
```

```
lmMod.default <- function(x, y, ...)
```

```
{
```

```
x <- as.matrix(x)
```

```
y <- as.numeric(y)
```

```
ans <- lmEst(x, y)
```

```
ans$fitted.values <- as.vector(x %*% ans$coefficients)
```

```
ans$residuals <- y - ans$fitted.values
```

```
ans$call <- match.call()
```

```
ans$data <- cbind(y, x)
```

```
class(ans) <- "lmMod"
```

```
ans
```

```
}
```

Creación de librerías con R

plot

```
> names(mod)
```

```
[1] "coefficients" "vcov"          "sigma"         "df"  
[5] "fitted.values" "residuals"     "call"          "data"
```

```
plot.lmMod <- function(x, ...)  
{  
  data <- x$data  
  coefs <- coef(x)  
  lab <- colnames(data)  
  plot(data[,3], data[,1], xlab=lab[3], ylab=lab[1], ...)  
  abline(coefs[1], coefs[2], lwd=2)  
  ff <- paste(lab[1], "=", round(coefs[1],2) , "+",  
              round(coefs[2],2), lab[3], sep="")  
  legend("topright", ff, bty="n", cex=0.8)  
}
```

Y si mas de una variable? Y el + -? Y la posición de la etiqueta? ...

Creación de librerías con R

Clases y métodos S4

- La implementación S4 de clases y métodos se pueden encontrar en el paquete “method” (e.g. via `library(methods)`)
- Referencia:
Programming with Data by J. Chambers (1998), the “green book”
- S4 classes/methods se utilizan muy a menudo en Bioconductor

Ejercicio

Al resultado de la función 'compareMean' asígnale una clase y:

1. Crea una función para el método 'print'
2. Crea una función para el método 'summary' (y su 'print')
3. ¿Se te ocurre alguna visualización gráfica?

NOTA: Antes debes pensar qué quieres mostrar

Sesión 5 – Creación de librerías en R

basado en *Writing R Extensions* v2.5.0

- Cómo crear un paquete en R
 - El archivo 'DESCRIPTION'
 - El archivo 'INDEX'
 - El archivo 'NAMESPACE'
 - Subdirectorios de la librería
 - Documentación
 - Vignette
- Chequeo de la librería
- Creación de la librería
- Envío a CRAN

Creación de librerías con R

Cómo crear un paquete en R

- Un paquete en R consiste en un subdirectorio con los archivos `DESCRIPTION` y los subdirectorios `R`, `data`, `demo`, `exec`, `inst`, `man`, `po`, `src`, y `tests` (algunos de los cuales pueden omitirse)
- El subdirectorio del paquete también puede incluir los archivos `INDEX`, `NAMESPACE`, `configure`, `cleanup`, y `COPYING`. Otros archivos como `README`, `NEWS` o `ChangeLog` son omitidos por R pero pueden ser muy útiles para otros usuarios

El archivo `DESCRIPTION`

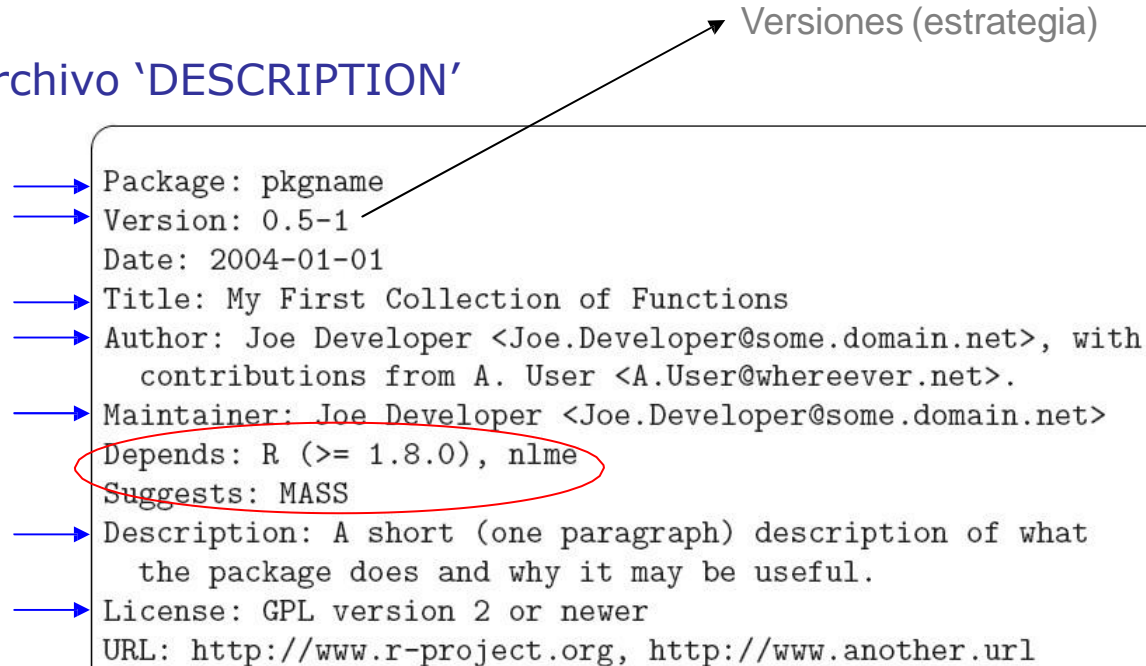


Diagram illustrating the structure of the `DESCRIPTION` file, with arrows pointing to specific fields:

- Package: pkgname
- Version: 0.5-1 (An arrow points from this field to the text "Versiones (estrategia)" above it.)
- Date: 2004-01-01
- Title: My First Collection of Functions
- Author: Joe Developer <Joe.Developer@some.domain.net>, with contributions from A. User <A.User@whereever.net>.
- Maintainer: Joe Developer <Joe.Developer@some.domain.net>
- Depends: R (>= 1.8.0), nlme (This line is circled in red in the original image.)
- Suggests: MASS
- Description: A short (one paragraph) description of what the package does and why it may be useful.
- License: GPL version 2 or newer
- URL: <http://www.r-project.org>, <http://www.another.url>

Creación de librerías con R

Cómo crear un paquete en R

El archivo 'INDEX' (opcional) contiene una línea para cada uno de los objetos que son suficientemente interesantes en el paquete, junto a su nombre y su descripción.

Normalmente este archivo no se crea y la información correspondiente se genera automáticamente de los códigos fuente de la documentación cuando se instala el paquete o bien cuando se crea el mismo

En vez de editar este archivo, es preferible poner una información sobre el paquete como un 'overview' y/o una viñeta.

INDEX del paquete Hmisc

ols	Ordinary least squares linear model
lrm	Binary and ordinal logistic regression model
psm	Accelerated failure time parametric survival model
cph	Cox proportional hazards regression
bj	Buckley-James censored least squares linear model
specs	Detailed specifications of fit
robcov	Robust covariance matrix estimates
bootcov	Bootstrap covariance matrix estimates
summary	Summary of effects of predictors
plot.summary	Plot continuously shaded confidence bars for results of summary
anova	Wald tests of most meaningful hypotheses

Creación de librerías con R

Cómo crear un paquete en R

El archivo 'NAMESPACE' (opcional) se usa para manejar los nombres de variables/funciones entre paquetes. Permite especificar qué variables/funciones en el paquete deberían *exportarse* para que las pudiera usar otro usuario, y cuales deberían ser *importadas* de otros paquetes (ventaja open source)

- Usar `export(f, g)` para exportar en el archivo NAMESPACE (aquí `f` y `g`)
- Si el paquete tiene muchas funciones `exportPattern`
 - `exportPattern("^[^\\.]")` exporta todas las funciones que no empiezan con [
- Usar `import(pkg1, pkg2)` para importar funciones de un paquete (aquí `pkg1` y `pkg2`)
 - `importFrom(pkg1, f, g)` para importar funciones `f` y `g` del paquete `pkg1`
- Mediante `pkg1:::f` se puede acceder a objetos no exportados
- Usar `S3method(print, f)` para Registrar métodos S3 (aquí `print.f`)
- Cargar "hooks"

`useDynLib(pkg1)` registra el objeto `pkg1` para que sea cargado con `library.dynam`

Creación de librerías con R

Ejemplo NAMESPACE

Funciones paquete 1 (pgk1)

```
x <- 1
f <- function(y) c(x,y)
foo <- function(x) .Call("pgk1", x, PACKAGE="pgk1")
print.foo <- function(x, ...) cat("<a foo>\n")
```

NAMESPACE

```
useDynLib(foo)
export(f, foo)
S3method(print, foo)
```

Funciones paquete 2 (pgk2)

```
c <- function(...) sum(...)
g <- function(y) f(c(y, 7))
h <- function(y) y+9
```

NAMESPACE

```
import(pgk1)
export(g, h)
```

Creación de librerías con R

Archivo first.lib.R o zzz.R

```
##### First.lib #####  
  
.onLoad <- function(lib, pkg){  
  library.dynam("mypackage", pkg, lib)  
}  
  
.onUnload <- function(libpath)  
  library.dynam.unload(" mypackage ", libpath)  
  
##### End of .First.lib #####
```

Se debe indicar si se necesitan otras librerías (require) o si se quiere poner algún mensaje de inicio cuando se carga la librería

Creación de librerías con R

Archivo first.lib.R o zzz.R

```
> library(mclust)
```

by using mclust, invoked on its own or through another package,
you accept the license agreement in the mclust LICENSE file
and at <http://www.stat.washington.edu/mclust/license.txt>

Creación de librerías con R

Archivo LICENCE (library MASS)

Software and datasets to support 'Modern Applied Statistics with S', fourth edition, by W. N. Venables and B. D. Ripley. Springer, 2002.

From the text (pp. 464):

These datasets and software are provided in good faith, but none of the authors, publishers nor distributors warrant their accuracy nor can be held responsible for the consequences of their use.

This file is intended to clarify ownership and copyright: where possible individual files also carry brief copyright notices.

Copyrights

=====

File MASS/R/profiles.R copyright (C) 1996 D. M. Bates and W. N. Venables.
port to R by B. D. Ripley copyright (C) 1998
corrections copyright (C) 2000,3,6 B. D. Ripley

Our understanding is that the dataset files MASS/data/*.rda are not copyright.

....

Creación de librerías con R

Archivo CITATION

```
citHeader("To cite the MASS package in publications use:")

citEntry(entry="Book",
  title = "Modern Applied Statistics with S",
  author = personList(as.person("W. N. Venables"),
    as.person("B. D. Ripley")),
  publisher = "Springer",
  edition = "Fourth",
  address = "New York",
  year = 2002,
  note = "ISBN 0-387-95457-0",
  url = "http://www.stats.ox.ac.uk/pub/MASS4",

  textVersion =
    paste("Venables, W. N. & Ripley, B. D. (2002)",
      "Modern Applied Statistics with S.",
      "Fourth Edition. Springer, New York. ISBN 0-387-95457-0")
)
```

Creación de librerías con R

Archivo CITATION

```
> citation("MASS")
```

To cite the MASS package in publications use:

Venables, W. N. & Ripley, B. D. (2002) Modern Applied Statistics with S. Fourth Edition. Springer, New York. ISBN 0-387-95457-0

A BibTeX entry for LaTeX users is

```
@Book{,  
  title = {Modern Applied Statistics with S},  
  author = {W. N. Venables and B. D. Ripley},  
  publisher = {Springer},  
  edition = {Fourth},  
  address = {New York},  
  year = {2002},  
  note = {ISBN 0-387-95457-0},  
  url = {http://www.stats.ox.ac.uk/pub/MASS4},  
}
```

Creación de librerías con R

El subdirectorio R

- Contiene archivos con el código R.
- Debe empezar con una letra (mayúscula o minúscula) o un dígito y debe tener las extensiones ``.R'`, ``.S'`, ``.q'`, ``.r'` ó ``.s'` (recomendado ``.R'`)
- Debe ser posible leer los archivos utilizando la función `source()`. Esto significa que los objetos deben ser creados mediante asignación

```
ej., el archivo maxstat.default.R contendría
maxstat.default<-function(x, y, ...)
{
# ... algunos controles
xx<-table(x,y)
maxstat.table(xx, ...)
}
```

- Se pueden usar varias funciones de R para inicializar y borrar. Para paquetes sin "name space" estas son `.First.lib` y `.Last.lib`.
- Normalmente se definen estas funciones en un archivo que se llama ``zzz.R'`

```
##### First.lib #####
.onLoad <- function(lib, pkg){
  require(haplo.stats)
  require(survival)
  require(mvtnorm)
  library.dynam("SNPassoc", pkg, lib)
}
.onUnload <- function(libpath)
  library.dynam.unload("SNPassoc", libpath)
##### End of .First.lib #####
```

Creación de librerías con R

El subdirectorio man

- Debería contener sólo archivos de documentación para los objetos que se han creado en el paquete en el formato *R documentation* (Rd) que veremos más adelante
- Debe empezar con una letra (mayúscula o minúscula) o un dígito y debe tener las extensiones ``.Rd'` (por defecto) ó ``.rd'` (recomendado ``.R'`)

El subdirectorio src

- Contiene el fuente y las cabeceras para el código compilado, y opcionalmente el archivo `Makevars` o `Makefile`.
- Cuando un paquete se instala con R CMD INSTALL, Make se utiliza como control de compilación.
- Las reglas y variables generales están definidas en `~R_HOME/etc/Makeconf` y dan soporte para C, C++, FORTRAN 77, Fortran 9x, Objective C y Objective C++ con las extensiones ``.c'`, ``.cc'` ó ``.cpp'` ó ``.C'`, ``.f'`, ``.f90'` ó ``.f95'`, ``.m'`, y ``.mm'` ó ``.M'`, respectivamente.
- Es recomendable utilizar ``.h'` para las cabeceras

Creación de librerías con R

El subdirectorio data

- Se usa para incluir archivos de datos adicionales que serán cargados con la función *data()*
- Actualmente, los archivos de datos pueden ser de tres tipos
 - plain R code (`.R` ó `.r`)
 - tablas (`.tab`, `.txt`, ó `.csv`)
 - imágenes *save()* (`.Rdata` ó `.rda`) (se recomienda usar *save()*, *compress=TRUE*)

El subdirectorio demo

- Es para R scripts (que se ejecutan via *demo()*) que demuestran la funcionalidad del paquete
- Debe empezar con una letra (mayúscula o minúscula) o un dígito y debe tener las extensiones *.R` ó `.r`*
- Si existe dicho archivo, el subdirectorio debe tener otro archivo *'00Index'* con una línea para cada demo dando su nombre y una descripción separada por un espacio (NOTA: no es posible generar este fichero de forma automática)

Los subdirectorios inst, tests, exec y po

Creación de librerías con R

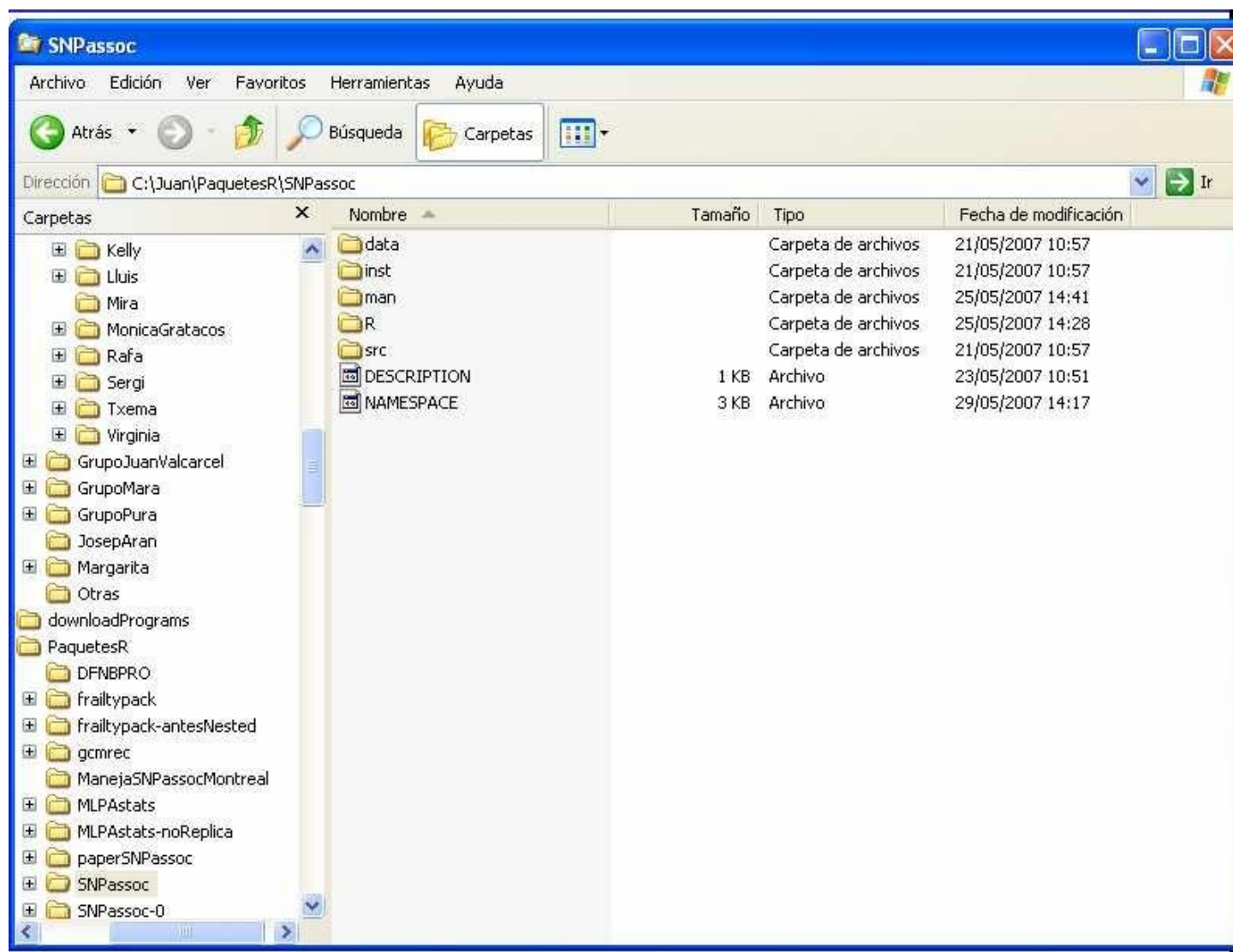
El subdirectorio exec

- Desde R se pueden llamar programas ejecutables (.exe, Perl, .sh, ...) como si se hiciera desde la consola usando la función `system()`
 - por ejemplo, `system("prog.exe")`
- PROBLEMA: cada usuario debería poner el archivo "prog.exe" en el directorio de trabajo
- SOLUCIÓN: `ff<-system.file("exec/prog.exe", package="myPackage")` `system(ff)`

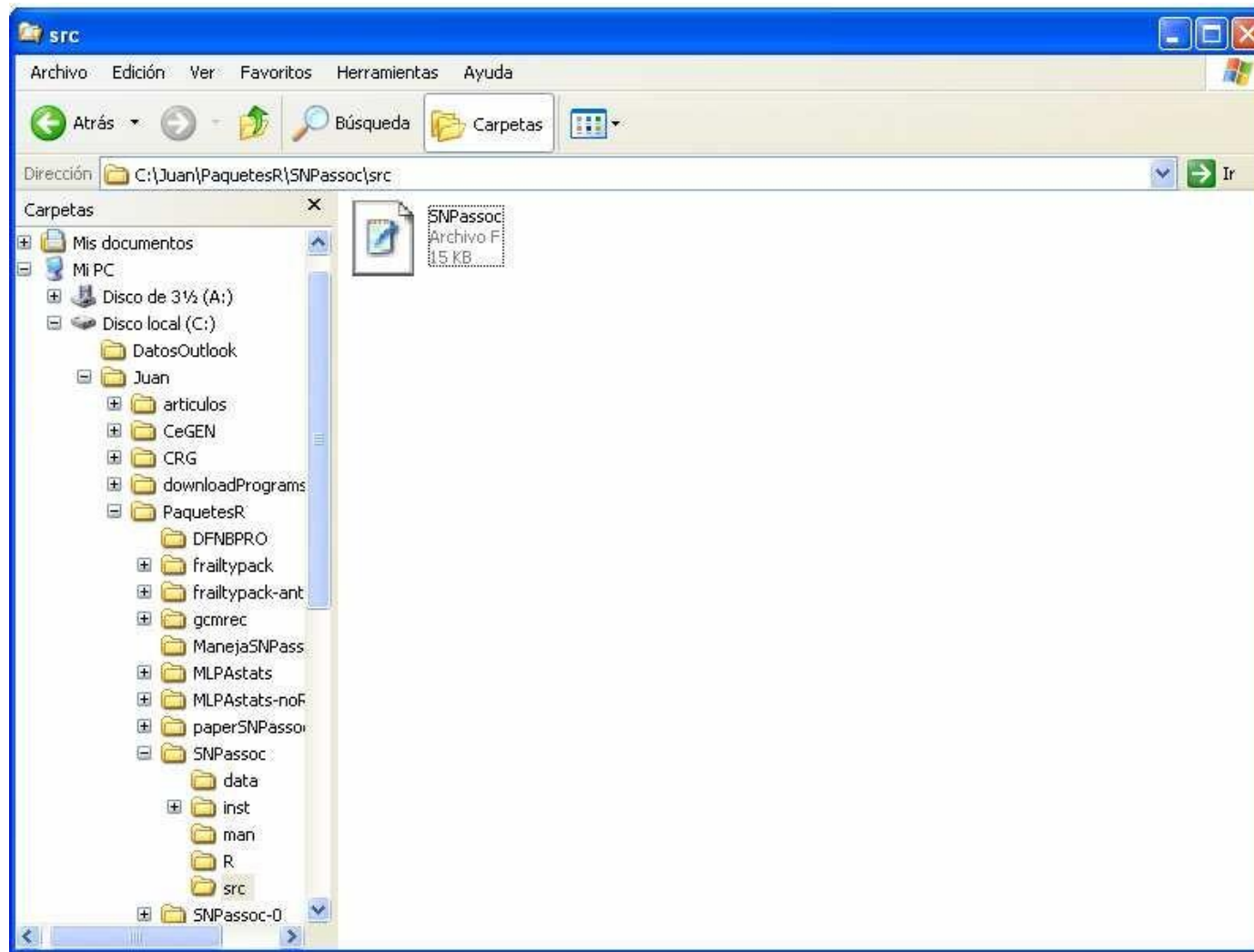
NOTAs:

- Se podría pensar en hacer un programa a la "vieja usanza" y usar `system()`
 - Ejemplo: Crear un ejecutable en Fortran que lea un fichero (en formato muy rígido) y devolver otro fichero que luego se lea con R
- PROBLEMAS:
 - No reproducibilidad del código (open source)
 - No poder usar la función en otro paquete
 - ...
- SOLUCIÓN: Crear dll
- Cuando usarlo: Cuando sea mejor que R (ejemplo partir un archivo -> Perl)

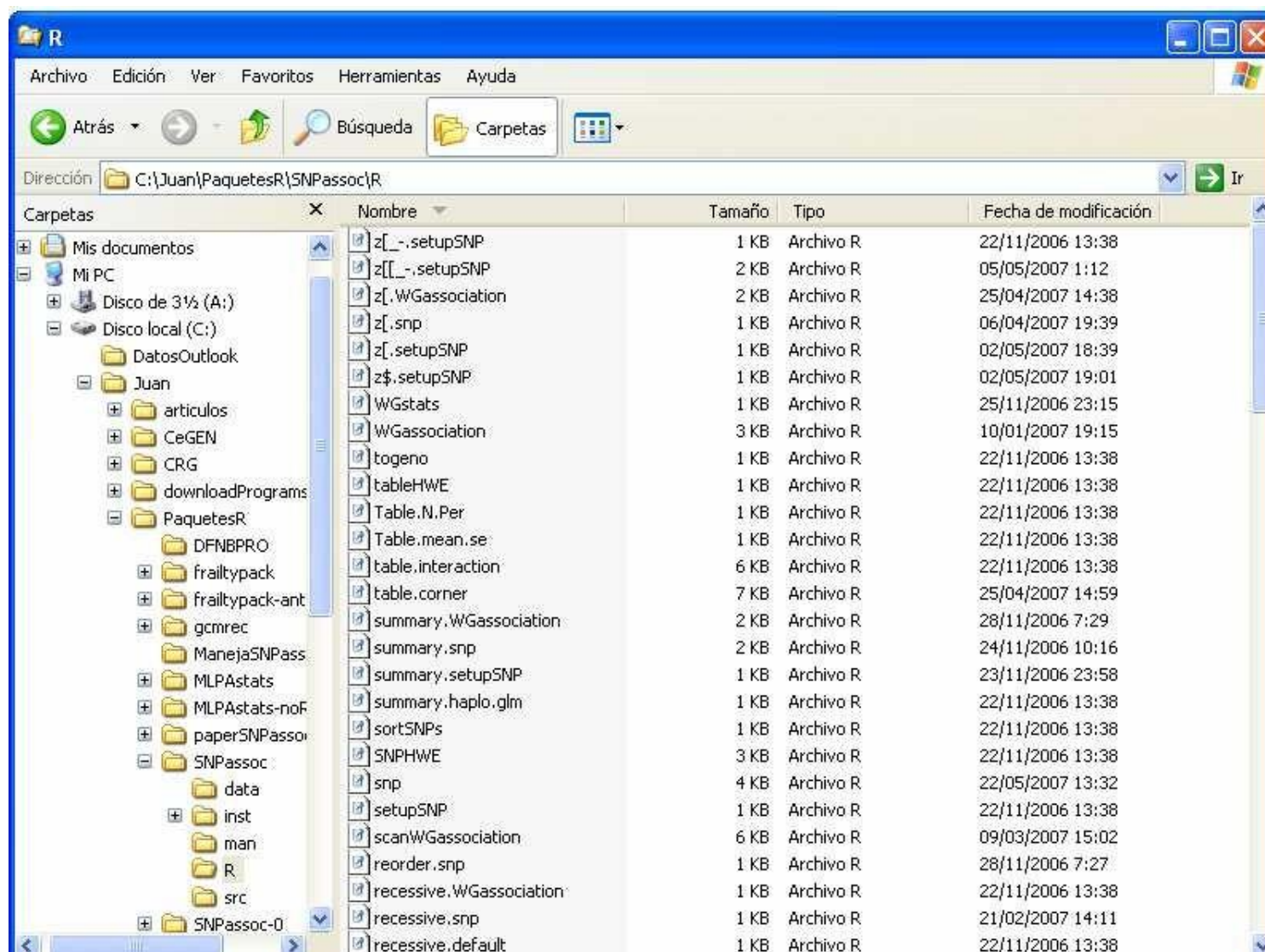
Creación de librerías con R



Creación de librerías con R



Creación de librerías con R



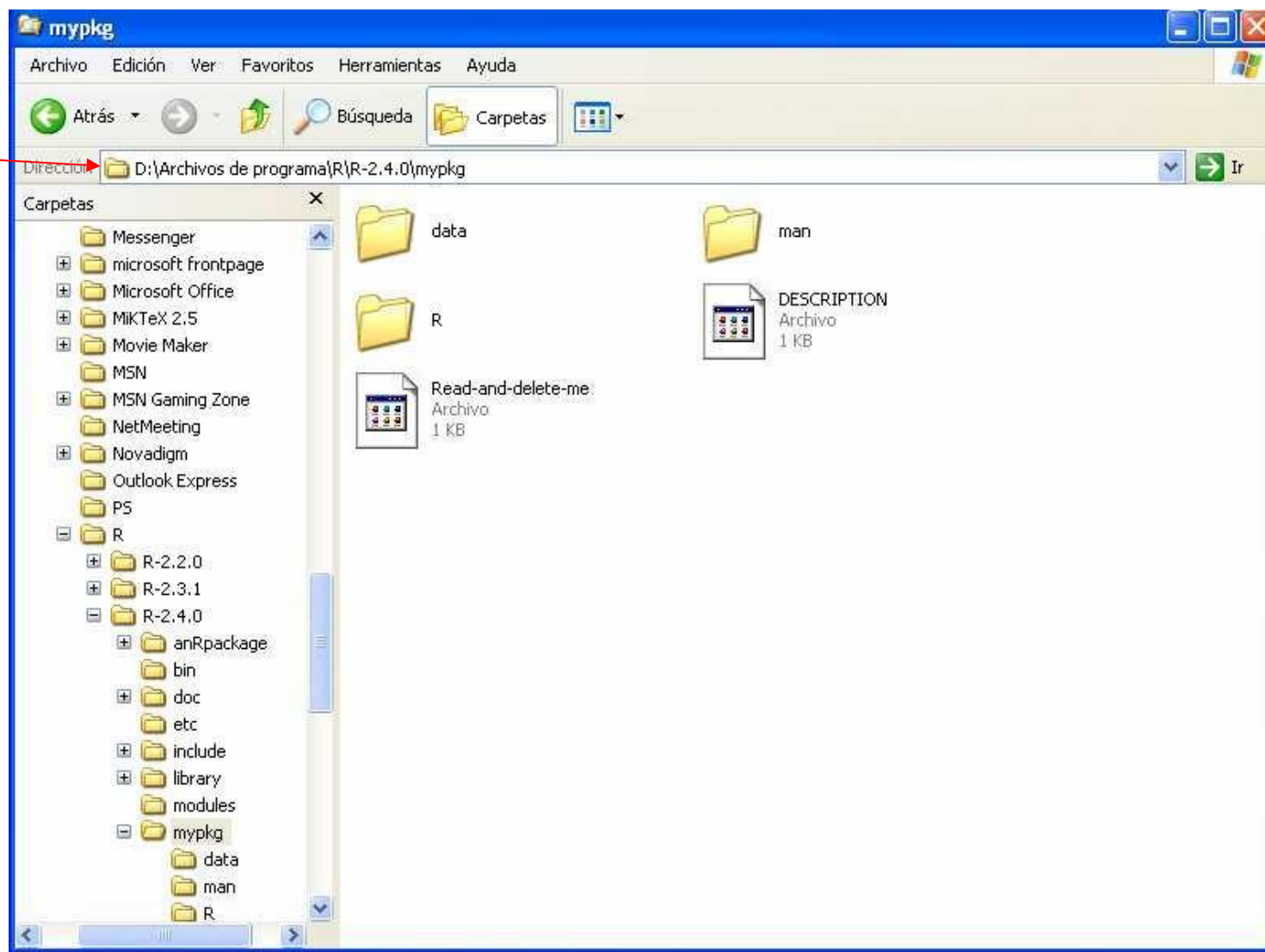
Creación de librerías con R

La función `package.skeleton()`

Ejemplo

```
## dos funciones y dos "data sets" :  
>f <- function(x,y) x+y  
>g <- function(x,y) x-y  
>d <- data.frame(a=1, b=2)  
>e <- rnorm(1000)  
  
>package.skeleton(list=c("f","g","d","e"), name="mypkg")  
Creating directories ...  
Creating DESCRIPTION ...  
Creating Read-and-delete-me ...  
Saving functions and data ...  
Making help files ...  
Done.  
Further steps are described in './mypkg/Read-and-delete-me'.
```

Creación de librerías con R



Creación de librerías con R

La función `package.skeleton()`

Further steps are described in `'./mypkg/Read-and-delete-me'`.

- * Edit the help file skeletons in `'man'`, possibly combining help files for multiple functions.
- * Put any C/C++/Fortran code in `'src'`.
- * If you have compiled code, add a `.First.lib()` function in `'R'` to load the shared library.
- * Run R CMD build to build the package tarball.
- * Run R CMD check to check the package tarball.

Read "Writing R Extensions" for more information.

Creación de librerías con R

Cómo escribir la documentación (capítulo 2 *Writing R Extensions v2.5.0*)

- `package.skeleton()` crea directamente los archivos .Rd
- Si añadimos otro objeto podemos usar la función `prompt()` para crear su manual

```
\name{load}
\alias{load}
\title{Reload Saved Datasets}
\description{
  Reload the datasets written to a file with the function
  \code{save}.
}
\usage{
load(file, envir = parent.frame())
}
\arguments{
  \item{file}{a connection or a character string giving the
    name of the file to load.}
  \item{envir}{the environment where the data should be
    loaded.}
}
\seealso{
  \code{\link{save}}.
}
\examples{
## save all data
save(list = ls(), file= "all.Rdata")

## restore the saved values to the current environment
load("all.Rdata")

## restore the saved values to the workspace
load("all.Rdata", .GlobalEnv)
}
\keyword{file}
```

documentación
de una función

Creación de librerías con R

Cómo escribir la documentación (capítulo 2 *Writing R Extensions v2.5.0*)

documentación
de un dataset

```
\name{rivers}
\docType{data}
\alias{rivers}
\title{Lengths of Major North American Rivers}
\description{
  This data set gives the lengths (in miles) of 141 \dQuote{major}
  rivers in North America, as compiled by the US Geological
  Survey.
}
\usage{rivers}
\format{A vector containing 141 observations.}
\source{World Almanac and Book of Facts, 1975, page 406.}
\references{
  McNeil, D. R. (1977) \emph{Interactive Data Analysis}.
  New York: Wiley.
}
\keyword{datasets}
```

Creación de librerías con R

Cómo escribir la documentación (capítulo 2 *Writing R Extensions v2.5.0*)

- Existen más “markups” (ver manual) [warnings, author, ...]
- Se pueden incluir listas y tablas (ver manual)
- Se pueden incluir fórmulas matemáticas (ver manual)

Usar `\eqn{latex}{ascii}` y `\deqn{latex}{ascii}`

`\eqn` similar a `$... $` en LaTeX (“inline” formulas)

`\deqn` similar a `$$... $$` en LaTeX (ambiente *displaymath*)

ej.: `\eqn{\chi^2}{chi^2}` → χ^2 en el pdf y `chi^2` en el texto de ayuda (? en R)

- Se puede procesar el formato Rd para ver cómo queda el manual (LaTeX)
 - R CMD Rd2txt crea un archivo de texto
 - R CMD Rd2dvi genera un archivo DVI (o PDF con la opción -pdf) ej. R CMD Rd2dvi --pdf --output=snpassoc.pdf
SNPassoc

(dentro de esta carpeta debe estar el DESCRIPTION y /man)

Creación de librerías con R

Guidelines for Rd files (en CRAN Manuals)

- \title sections should be capitalized and not end in a period.
- **Argument lists need to cover all the arguments in all the functions covered, including '...'**.
- Do provide text alternatives for all but the very simplest \eqn and \deqn commands.
- **In \source and \references sections**
 - Use separate paragraphs (separated by a blank line) for each reference.
 - Write authors' names in the form Author, A. B. and separate them by a comma or and (but not both).
 - Give a date immediately after the author(s), and do not put a period after the date.
 - Enclose titles of books and journals (but *not* articles) in \emph{...}.
 - Enclose volume numbers for journals in \bold{...} and follow them by a comma.
 - Use -- for page ranges.
 - If you give an address for a publisher (probably unnecessary) use the form New York: Springer-Verlag.
- In \usage and \examples sections
 - Watch the line length: 65 characters is a reasonable limit.
 - Use TRUE and FALSE rather than T and F.
 - **Readability is greatly enhanced by adding spaces around binary operators and after commas in argument lists. In particular, <- needs spaces around it.**
- **Always use <- rather than = for assignments.**

Creación de librerías con R

Guidelines for Rd files (en CRAN Manuals)

- Follow the 'R core' indentation guidelines for example code (which basically means an indentation level of 4 spaces and aligning continuation lines suitably — but copy the code into a .R Emacs buffer and reformat it there).
- Either ensure that the `\usage` section exactly matches the function definition, or include a `\synopsis` section with the actual definition.
- Not only make sure that examples are directly executable, also make sure that they are system-independent (do not use `system`) and do not require special facilities (for example Internet access or write permission to specific directories).
- Use quotation marks sparingly: quotation marks are used for R objects in text rendition. If you must use them, make them double (`\dQuote{a quotation}`), and use `\sQuote` and `\dQuote<>`
- **Do not use tabs to indent** (as these do not render correctly on all possible pagers).
- Please put two spaces after the end of a sentence in sections with running text (`\description`, `\details`, ...).
- For consistency, aim to use British (rather than American) spelling. (NB: British spelling often uses -ize as in 'capitalize'. The view of spell -b of British spelling is a flawed American one. There are English/Scottish differences too.)
- Do not use `&` except where quoting (for example, the publisher that has been called Chapman & Hall fairly recently).

Creación de librerías con R

Cómo escribir la documentación – Funciones internas

archivo myPackage-internal.Rd

```
\name{myPackage-inl}  
\alias{myPackage-internal}  
\alias{fun1}  
\alias{fun2}  
\alias{fun3}  
...  
  
\title{Internal myPackage functions}  
\description{Internal myPackage functions}  
\usage{  
  fun1(x, y)  
  fun2(object, y, ...)  
  fun3(x, y, data, method, ...)  
}  
  
\details{These are not to be called by the user}  
\keyword{internal}
```

Creación de librerías con R

Cómo escribir la documentación – Descripción del paquete

archivo myPackage-package.Rd

```
\name{lmPackage-package}  
\alias{lmPackage-package}  
\alias{lmPackage}  
\docType{package}  
\title{  
What the package does (short line)  
~~ package title ~~  
}  
\description{  
More about what it does (maybe more than one line)  
~~ A concise (1-5 lines) description of the package ~~  
}  
  
...
```

Ver ejemplo [samplesize.pdf](#) con todo lo que hace según el tipo de .Rd

Creación de librerías con R

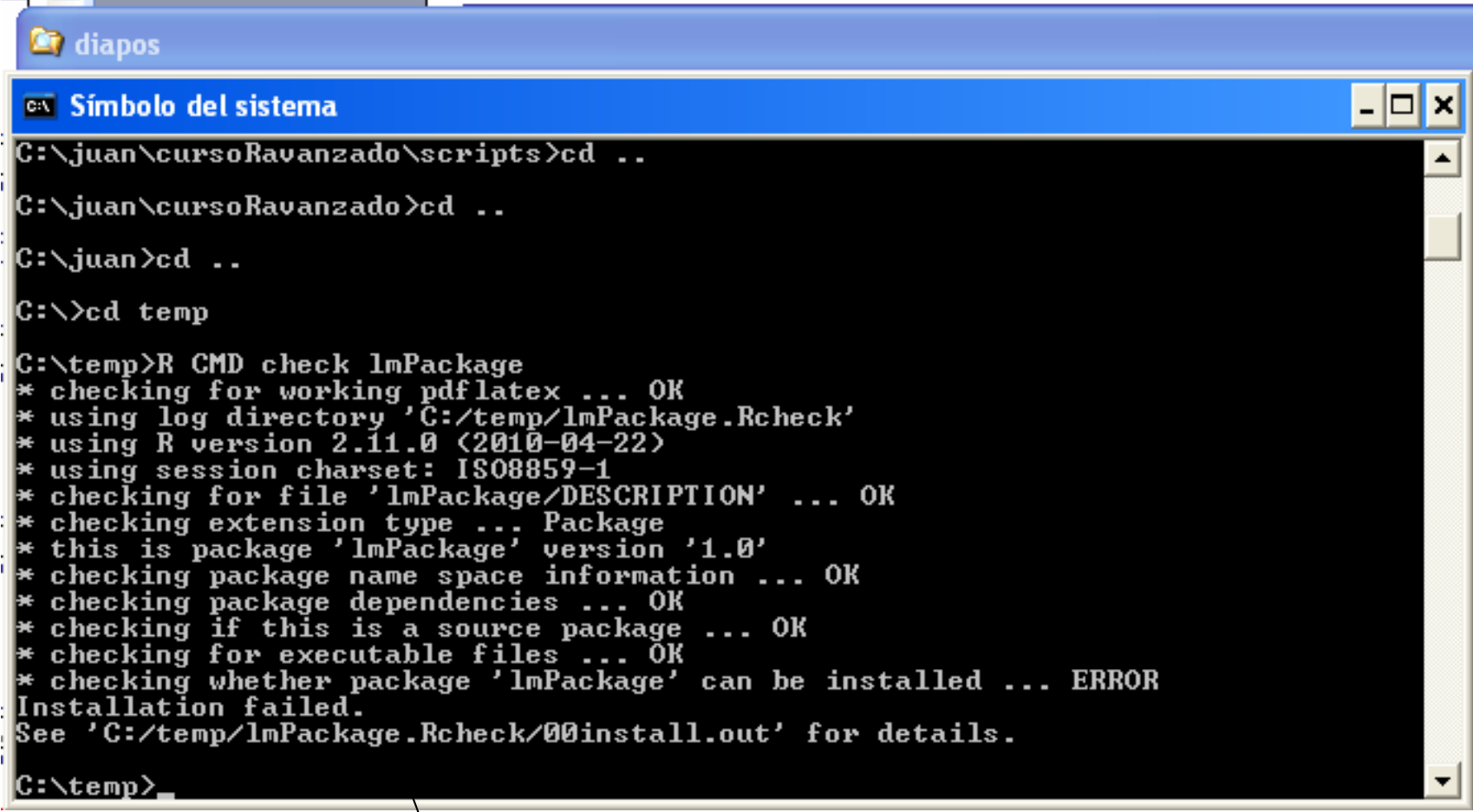
Ejemplo – modelo regresión lineal con descomposición QR

```
> ls()  
[1] "bass"           "lmEst"          "lmMod"  
[4] "lmMod.default" "plot.lmMod"     "print.lmMod"  
[7] "print.summary.lmMod" "summary.lmMod"
```

```
> package.skeleton("lmPackage", path="c:/temp/")  
Creating directories ...  
Creating DESCRIPTION ...  
Creating Read-and-delete-me ...  
Saving functions and data ...  
Making help files ...  
Done.  
Further steps are described in 'c:/temp//lmPackage/Read-and-  
delete-me'.
```


Creación de librerías con R

Ejemplo – modelo regresión lineal con descomposición QR



```
diapos
C:\> Símbolo del sistema
C:\juan\cursoRavanzado\scripts>cd ..
C:\juan\cursoRavanzado>cd ..
C:\juan>cd ..
C:\>cd temp
C:\temp>R CMD check lmPackage
* checking for working pdflatex ... OK
* using log directory 'C:/temp/lmPackage.Rcheck'
* using R version 2.11.0 (2010-04-22)
* using session charset: ISO8859-1
* checking for file 'lmPackage/DESCRIPTION' ... OK
* checking extension type ... Package
* this is package 'lmPackage' version '1.0'
* checking package name space information ... OK
* checking package dependencies ... OK
* checking if this is a source package ... OK
* checking for executable files ... OK
* checking whether package 'lmPackage' can be installed ... ERROR
Installation failed.
See 'C:/temp/lmPackage.Rcheck/00install.out' for details.
C:\temp>
```

lmPackage.Rcheck

Creación de librerías con R

Ejemplo – modelo regresión lineal con descomposición QR

*** install options are ' --no-html'**

*** installing *source* package 'lmPackage' ...**

**** R**

**** data**

**** preparing package for lazy loading**

**** help**

Aviso: ./man/lmPackage-package.Rd:34: All text must be in a section

Aviso: ./man/lmPackage-package.Rd:35: All text must be in a section

***** installing help indices**

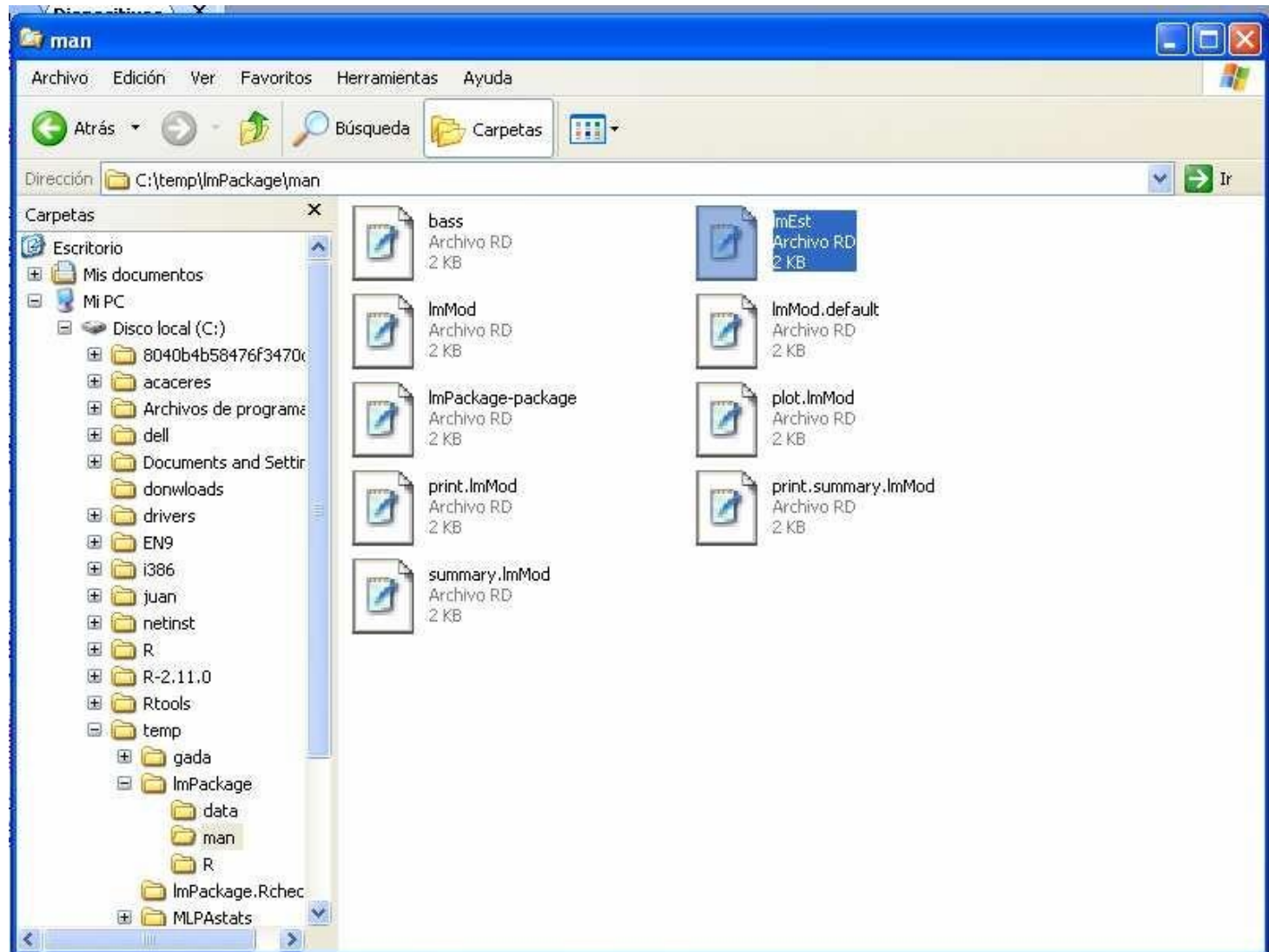
Error en Rd_info(db[[i]]) : Rd files must have a non-empty \title.

See chapter 'Writing R documentation' in manual 'Writing R Extensions'.

*** removing 'C:/temp/lmPackage.Rcheck/lmPackage'**

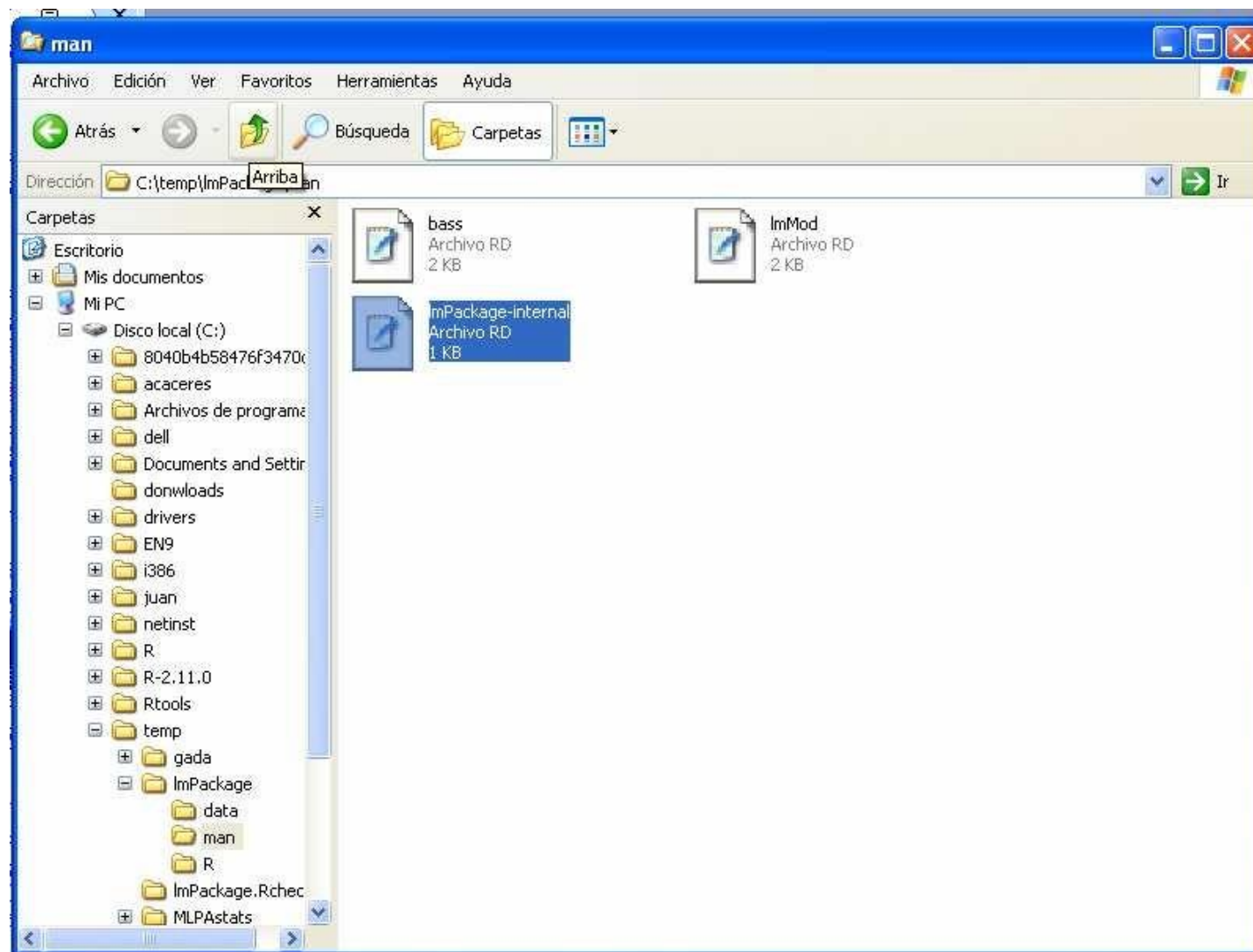
Creación de librerías con R

Ejemplo – modelo regresión lineal con descomposición QR



Creación de librerías con R

Ejemplo – modelo regresión lineal con descomposición QR



Creación de librerías con R

Archivo lmPackage-internal.Rd

```
\name{lmPackage-internal}  
\alias{lmPackage-internal}  
\alias{lmEst}  
  
\title{Internal lmPackage functions}  
\description{Internal lmPackage functions}  
\usage{  
  lmEst(x, y)  
}  
  
\details{These are not to be called by the user}  
\keyword{internal}
```

Creación de librerías con R

Archivo lmMod.Rd

```
\name{lmMod}
\alias{lmMod}
\alias{lmMod.default}
\alias{print.lmMod}
\alias{summary.lmMod}
\alias{print.summary.lmMod}
\alias{plot.lmMod}

\title{ Linear regresion using QR decomposition }

\description{ This package performs ... }

\usage{
  lmMod(x, ...)

  \method{lmMod}{default}(x, y, ...)
  \method{print}{lmMod}(x, ...)
  \method{summary}{lmMod}(object, ...)
  \method{print}{summary.lmMod}(x, ...)
  \method{plot}{lmMod}(x, ...)
}

\arguments{
  \item{x}{model matrix}
  \item{y}{dependent variable}
  \item{object}{object of class 'lmMod'}
  \item{\dots}{other arguments to be passed to print, plot, summary}
}
```

Creación de librerías con R

Archivo ImPackage-internal.Rd (cont.)

```
\details{ To be supplied }

\value{ %% ~Describe the value returned
%% If it is a LIST, use
%% \item{comp1 }{Description of 'comp1'}
%% \item{comp2 }{Description of 'comp2'} %% ...}

\references{ %% ~put references to the literature/web site here ~ }

\author{ %% ~~who you are~~ }

\note{ %% ~~further notes~~ }

%% ~Make other sections like Warning with \section{Warning }{....} ~

\seealso{ %% ~~objects to See Also as \code{\link{help}}, ~~~ }

\examples{
data(bass)
mod<-lmMod(cbind(Const=1, Alk=bass$Alkalinity), bass$Mercury)
coef(mod)
fitted(mod)
resid(mod)
summary(mod)
plot(mod)
}

% Add one or more standard keywords, see file 'KEYWORDS' in the R documentation directory.
\keyword{ ~kwd1 }
\keyword{ ~kwd2 }
```

Creación de librerías con R

File KEYWORDS (C:\R-2.14.0\doc)

GROUPED Keywords

Graphics

aplot	&	Add to Existing Plot / internal plot
dplot	&	Computations Related to Plotting

...

Statistics

datagen	&	Functions for generating data sets
distribution	&	Probability Distributions and Random Numbers
univar	&	simple univariate statistics [!= S]
htest	&	Statistical Inference
models	&	Statistical Models
& regression&		Regression
& &nonlinear&		Non-linear Regression (only?)
robust	&	Robust/Resistant Techniques
design	&	Designed Experiments
multivariate	&	Multivariate Techniques
ts	&	Time Series
survival &		Survival Analysis
nonparametric	&	Nonparametric Statistics [w/o 'smooth']

...

Creación de librerías con R

Creación de ayuda con la librería devtools y roxygen2

1. Creamos una librería vacía

```
library(devtools)  
library(roxygen2)  
create("ImPackage")
```

Crea: Carpeta R, DESCRIPTION y NAMESPACE

2. Movemos los ficheros .R a la carpeta R (e.g., fichero ImModFinal.R) (NOTA: ver cabecera de la función)

3. Creamos el .Rd (carpeta man) con

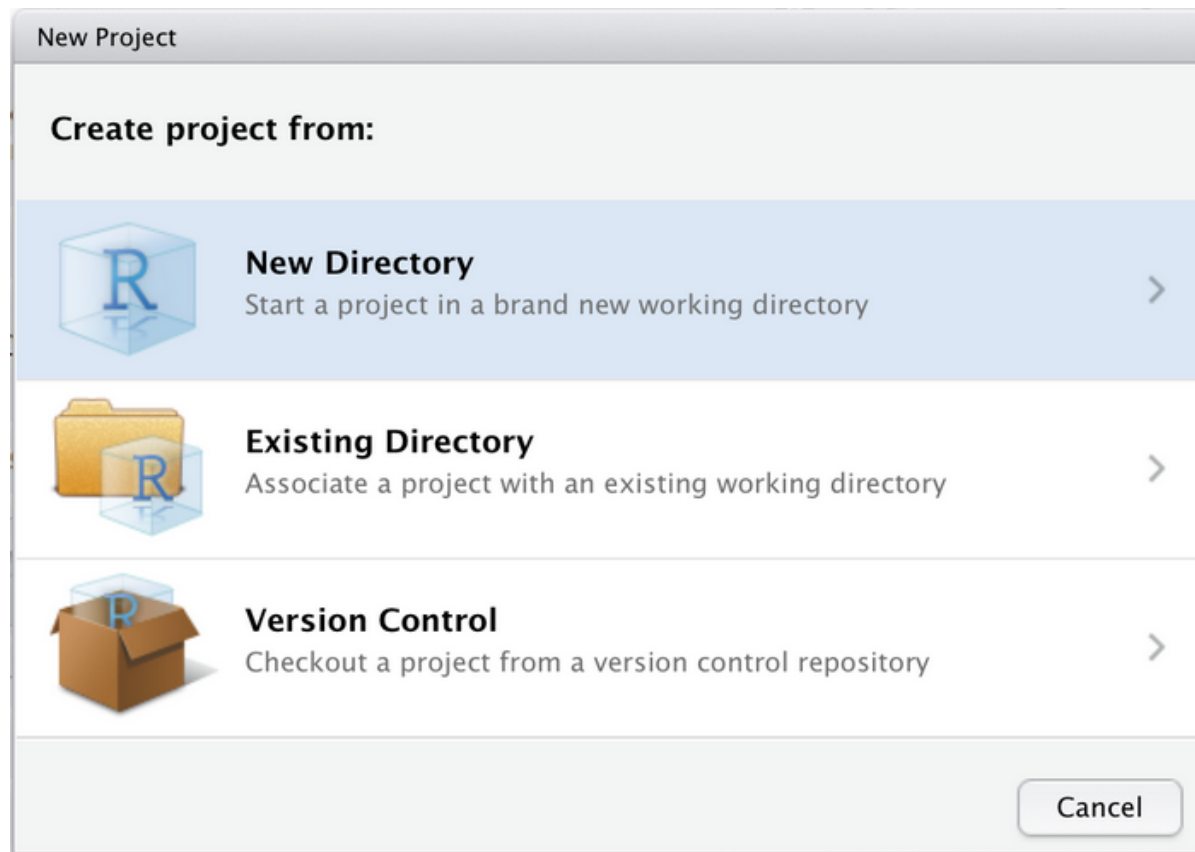
```
setwd("ImPackage")  
document()
```

NOTA: a veces hay que ejecutar esto dos veces

Creación de librerías con R

Con RStudio

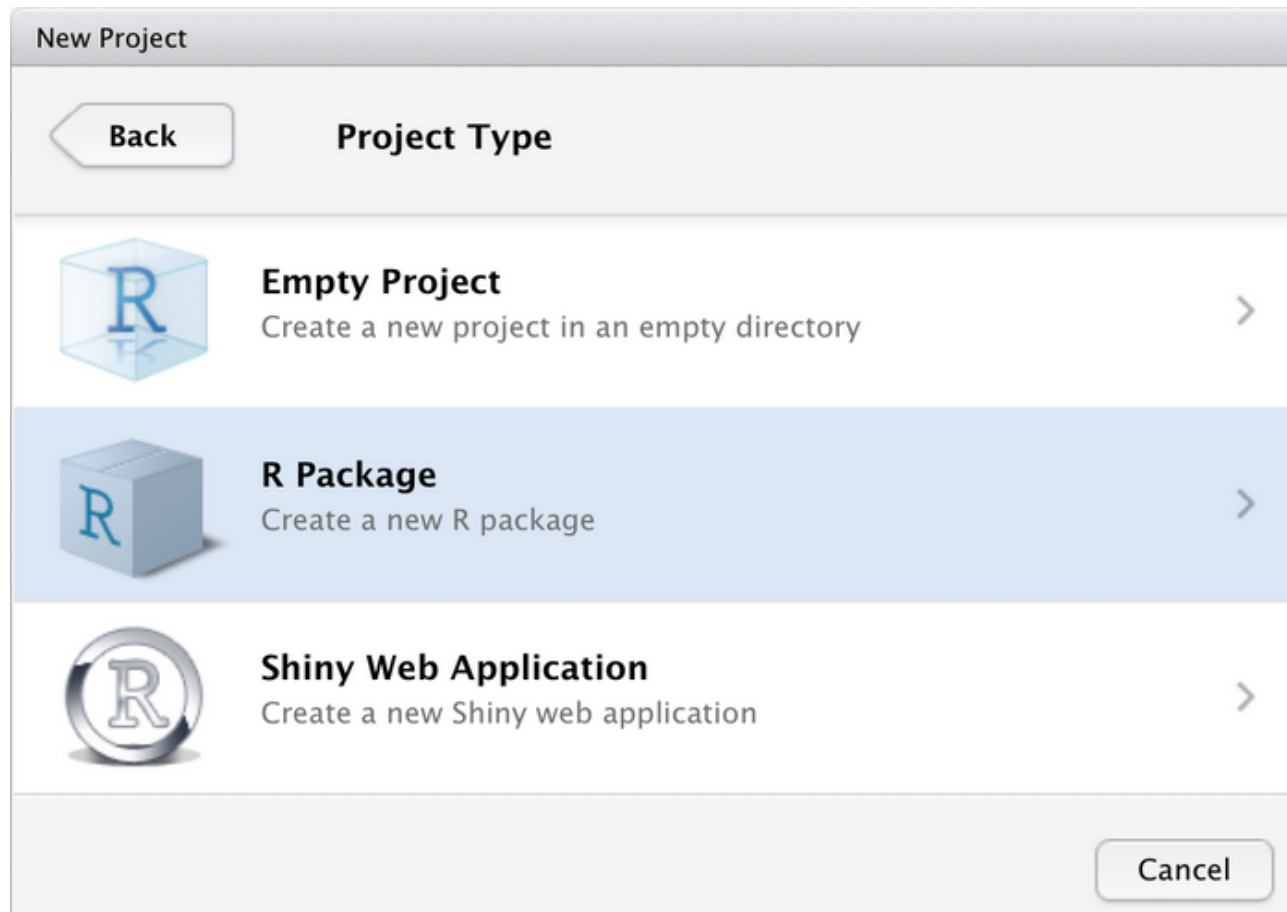
1. Pinchar en File | New project
2. Escoger "New Directory"



Creación de librerías con R

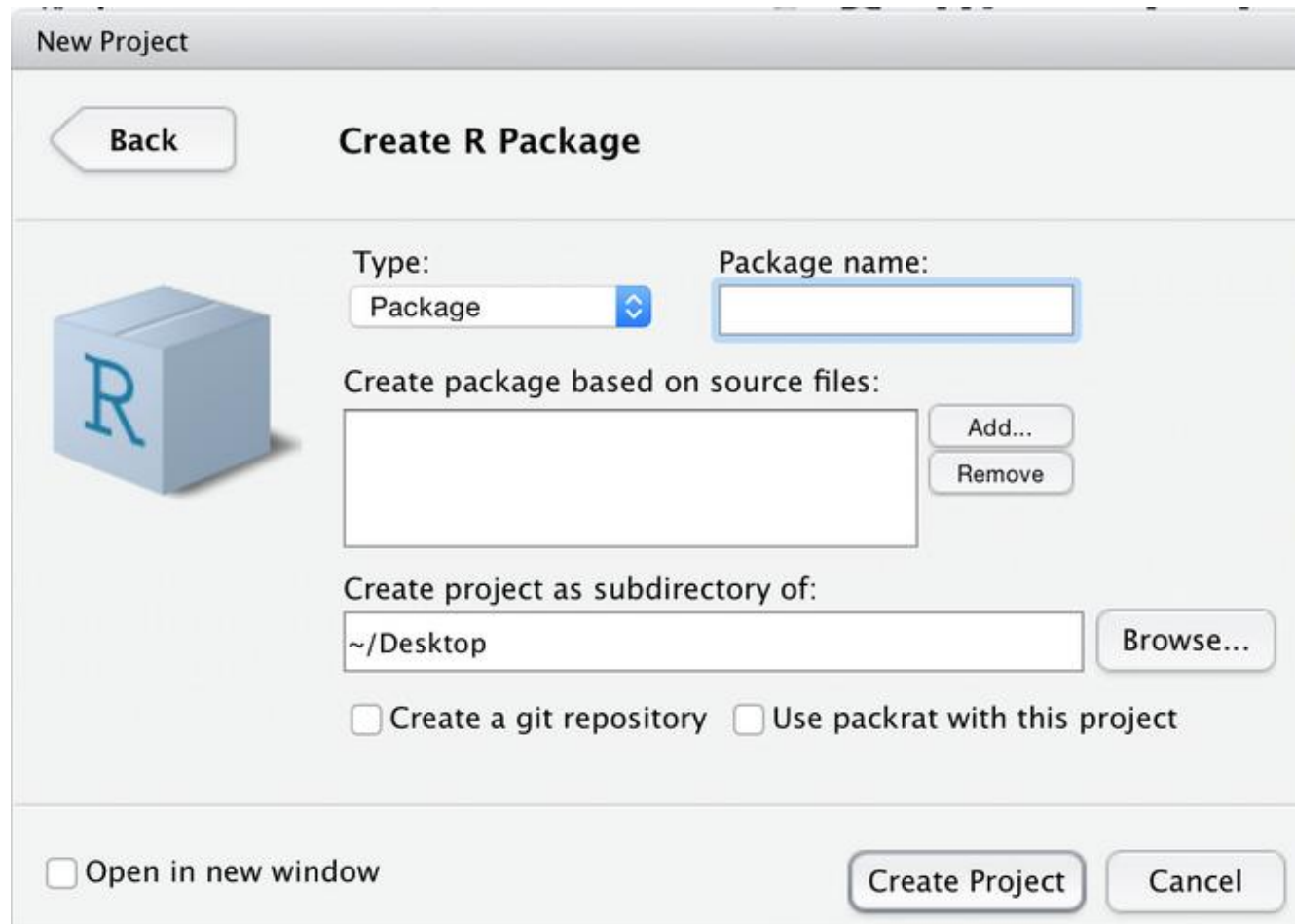
Con RStudio

3. Despues "R package"



Creación de librerías con R

4. Después pon un nombre al paquete y pincha "Create R package"



The screenshot shows the 'New Project' dialog box in R Studio, with the 'Create R Package' tab selected. The dialog includes a 'Back' button, a 'Type' dropdown menu set to 'Package', and a 'Package name' text field. Below these, there is a section for 'Create package based on source files' with an 'Add...' button and a 'Remove' button. The 'Create project as subdirectory of:' section shows a text field with '~/'Desktop' and a 'Browse...' button. At the bottom, there are two checkboxes: 'Create a git repository' and 'Use packrat with this project'. The 'Open in new window' checkbox is also present. The 'Create Project' and 'Cancel' buttons are at the bottom right.

New Project

Back

Create R Package

Type: Package

Package name:

Create package based on source files:

Add...

Remove

Create project as subdirectory of:

~/Desktop

Browse...

☐ Create a git repository ☐ Use packrat with this project

☐ Open in new window

Create Project Cancel

Escribir un vignette para la librería

- Además de la ayuda en formato .Rd, los paquetes de R permiten la inclusión de otros documentos en otros formatos
- ANTES debían estar en la carpeta 'inst/doc' del código fuente. AHORA en la carpeta 'vignettes'
- Pueden estar en cualquier formato pero se recomienda tenerlo en formato PDF o HTML para que los usuarios puedan leerlos desde cualquier plataforma
- Normalmente se ponen 'technical reports' o se suelen hacer un manual específico
- **VIGNETTE:** Compilado con Sweave / Knitr / Markdown -> Garantiza reproducibilidad

Escribir un vignette para la librería

- Knitr: Ver material en
- Markdown: Ver material en

Ejercicio (Después de ver el código siguiente):

1. Crea la librería 'lmPackage' [vacía] como hemos visto anteriormente
2. Añade el fichero 'lmModFinal.R' a la carpeta 'R' de la librería
3. Añade los datos 'bass.txt' a la carpeta 'extdata'
4. Crea la documentación
5. Añade el fichero 'lmPackage.Rnw' a una carpeta 'vignettes'
6. Compila la librería
7. Abre el fichero lmPackage.Rnw y crea la vignette

Creación de librerías con R

Una posible vignette para nuestro ejemplo

```
\documentclass[11pt]{article}

\begin{document}

\title{\bf Ejemplo creación de una vignette: modelo lineal}
\vspace{1cm}
\author{Juan R González}

\maketitle

\begin{center}
  Instituto de Salud Global Barcelona (ISGlobal)
  \vspace{1cm}
  {\tt juan.rgonzalez@isglobal.org}
  {\tt http://www.creal.cat/jrgonzalez/software.htm}
\end{center}

\tableofcontents

%%%%%%%%%%%%%%
\section{Introduction}

Este documento da una visión general del paquete {\tt lmPackage} que está
accesible en CRAN. Mediante esta librería se puede llevar a cabo ...

\section{Getting started}
\noindent El paquete {\tt lmPackage} utiliza la librería {\tt survival}. Empezamos cargando las librerías necesarias.
```

Creación de librerías con R

```
<<load_library>>=  
library(survival)  
@
```

\noindent Despues, la librería puede cargarse ejecutando

```
<<load_library2>>=  
library(lmPackage)  
@
```

\section{The data}

\noindent Los datos pueden importarse usando la función `read.delim` de forma usual. El paquete tiene unos datos (sobre unos lagos) que pueden usarse como ejemplo. Se cargan usando la función `data`

```
<<load_data>>=  
bass <- read.delim("../extdata/bass.txt")  
@
```


Creación de librerías con R

```
\section{Model parameter estimates}
```

```
\noindent El modelo se puede estimar usando la función {\tt lmMod}
```

```
<<fit_model>>=  
mod <- lmModFinal(Mercury ~ Alkalinity, data=bass)  
names(mod)  
@
```

```
\noindent Se puede crear una figura con el modelo mediante la siguiente instrucción:
```

```
<<fig_mod, fig.cap='Modelo de regresion lineal estimado mediante descomposicion QR'>>=  
plot(mod$data[,2], mod$data[,3], ylab="Mercury", xlab="Alkalinity")  
abline(mod)  
@
```

```
\end{document}
```

Creación de librerías con R

The screenshot displays the RStudio interface with a file named `ImPackage.Rnw` open. The editor shows LaTeX code for a document class and a vignette. A blue arrow points to the `Compile PDF` button in the toolbar. The console at the bottom shows the R welcome message.

```
1 \documentclass[11pt]{article}
2
3 \begin{document}
4
5 \title{\bf Ejemplo creaci'on de una vignette: modelo lineal}
6 \vspace{1cm}
7 \author{Juan R Gonz'alez}
8
9 \maketitle
10
11 \begin{center}
12 Instituto de salud Global Barcelona (ISGlobal)
13 \vspace{1cm}
14 {\tt juan.rgonzalez@isglobal.org}
15 {\tt http://www.creal.cat/jrgonzalez/software.htm}
16 \end{center}
```

Console output:

```
C:/Juan/CREAL/cursos/BioTools/Day_1/ImPackage/
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

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.

> |
```

Ejemplo creación de una vignette: modelo lineal

Juan R González

May 10, 2017

Instituto de Salud Global Barcelona (ISGlobal)
`juan.rgonzalez@isglobal.org`

`http://www.creal.cat/jrgonzalez/software.htm`

Contents

1	Introduction	1
2	Getting started	1
3	The data	2
4	Model parameter estimates	2

1 Introduction

Este documento da una visión general del paquete `lmPackage` que está accesible en CRAN. Mediante esta librería se puede llevar a cabo ...

2 Getting started

El paquete `lmPackage` utiliza la librería `survival`. Empezamos cargando las librerías necesarias.

Creación de librerías con R

```
library(survival)
```

Despues, la librería puede cargarse ejecutando

```
library(lmPackage)
```

3 The data

Los datos pueden importarse usando la función `read.delim` de forma usual. El paquete tiene unos datos (sobre unos lagos) que pueden usarse como ejemplo. Se cargan usando la función `data`

```
bass <- read.delim("../extdata/bass.txt")
```

4 Model parameter estimates

El modelo se puede estimar usando la función `lmMod`

```
mod <- lmModFinal(Mercury ~ Alkalinity, data=bass)
names(mod)

## [1] "coefficients" "vcov"          "sigma"          "df"
## [5] "data"
```

Se puede crear una figura con el modelo mediante la siguiente instrucción:

```
plot(mod$data[,2], mod$data[,3], ylab="Mercury", xlab="Alkalinity")
abline(mod)
```

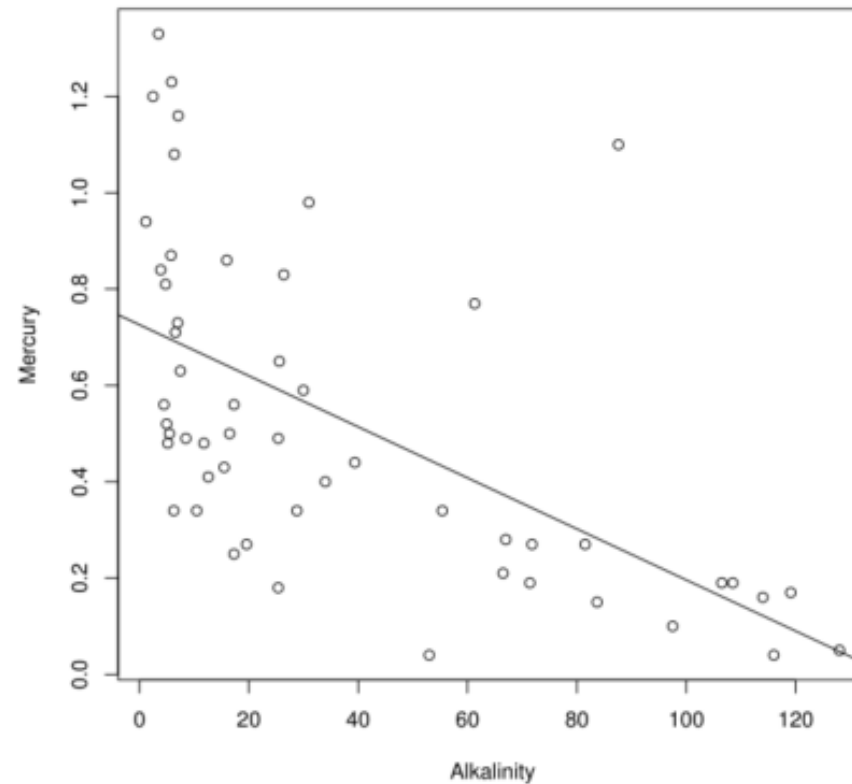


Figure 1: Modelo de regresion lineal estimado mediante descomposicion QR

Creación de librerías con R

Otras ... (inst/doc/changelog.txt)

v1.4-8

2007-05-21

- mvtnorm is required (change in DESCRIPTION, NAMESPACE and firstlib.R)
- added: new generic function maxstat (to compute asymptotic p values for max-statistic) and some methods
- added: function for inspecting overdispersion due to population substructure (qqpval)
- added: function for correcting the p values using genomic control (GenomiControl)
- added: example in the Rd file for odds function
- added: argument nIndiv in association.fit (to deal with genotypeRate properly). association has also been changed (look for 'nSubject')
- minor change in the function snp (when snp is applied to an object of class snp)
- added: reference of SNPassoc to some .Rd files

v1.4-7

2007-05-17

- bug fixed: overdominant returned wrong levels (thanks to Nicholas Orr)

2007-05-02

- bug fixed: [\leftarrow].setupSNP added any variable as new SNP, but could be a covariate
- bug fixed: odds returns NA for monomorphic SNPs
- bug fixed: association.fit detected monomorphic SNPs that were not so

...

Creación de librerías con R

Chequear la librería

R CMD check lleva a cabo las siguientes verificaciones

```
jrgonzalez@PT13:~> R CMD check SNPassoc
* checking for working latex ... OK
* using log directory '/home/jrgonzalez/SNPassoc.Rcheck'
* using R version 2.4.0 (2006-10-03)
* checking for file 'SNPassoc/DESCRIPTION' ... OK
* this is package 'SNPassoc' version '1.4-8'
* checking package dependencies ... OK
* checking if this is a source package ... OK
* checking whether package 'SNPassoc' can be installed ... OK
* checking package directory ... OK
* checking for portable file names ... OK
* checking for sufficient/correct file permissions ... OK
* checking DESCRIPTION meta-information ... OK
* checking top-level files ... OK
* checking index information ... OK
* checking package subdirectories ... OK
* checking R files for syntax errors ... OK
* checking R files for non-ASCII characters ... OK
* checking whether the package can be loaded ... OK
* checking whether the package can be loaded with stated dependencies ... OK
* checking whether the name space can be loaded with stated dependencies ... OK
* checking S3 generic/method consistency ... OK
* checking replacement functions ... OK
* checking foreign function calls ... OK
* checking R code for possible problems ... OK
* checking Rd files ... OK
* checking Rd cross-references ... OK
* checking for missing documentation entries ... OK
* checking for code/documentation mismatches ... OK
* checking Rd \usage sections ... OK
* checking for CRLF line endings in C/C++/Fortran sources/headers ... OK
* checking for portable use of $BLAS_LIBS ... OK
* creating SNPassoc-Ex.R ... OK
* checking examples ... OK
* creating SNPassoc-manual.tex ... OK
* checking SNPassoc-manual.tex ... OK
```

Creación de librerías con R

Chequear la librería

Ver algunos errores

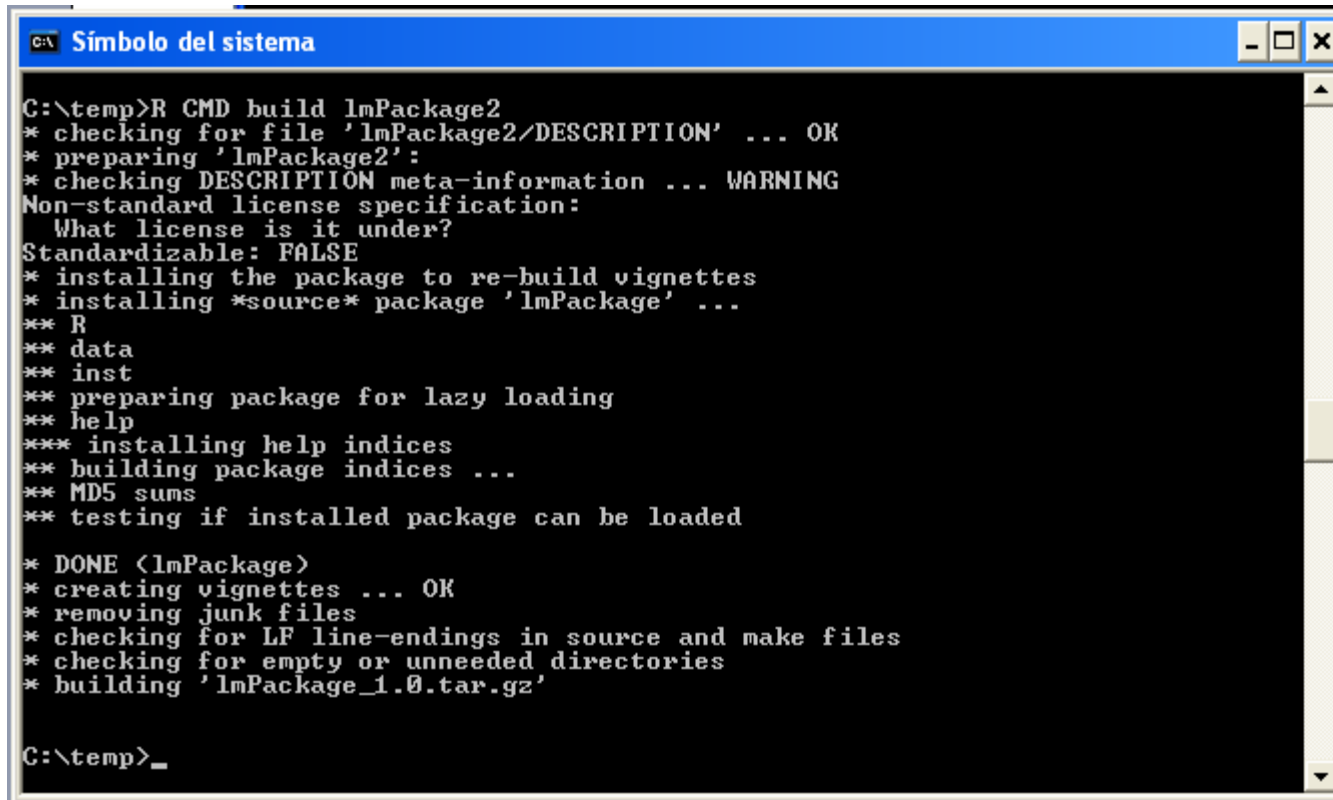
- que no cuadre el .Rd con .R
- que haya una función sin .Rd
- Consistencia funciones genéricas (el primer argumento de 'summary' es x y no 'object')
- métodos en .Rd
- petar ejemplos

Ver `ImPackage.Rcheck`

Creación de librerías con R

Crear una librería (.tar.gz) desde la consola

R CMD build



```
C:\temp>R CMD build lmPackage2
* checking for file 'lmPackage2/DESCRIPTION' ... OK
* preparing 'lmPackage2':
* checking DESCRIPTION meta-information ... WARNING
Non-standard license specification:
What license is it under?
Standardizable: FALSE
* installing the package to re-build vignettes
* installing *source* package 'lmPackage' ...
** R
** data
** inst
** preparing package for lazy loading
** help
*** installing help indices
** building package indices ...
** MD5 sums
** testing if installed package can be loaded

* DONE (lmPackage)
* creating vignettes ... OK
* removing junk files
* checking for LF line-endings in source and make files
* checking for empty or unneeded directories
* building 'lmPackage_1.0.tar.gz'

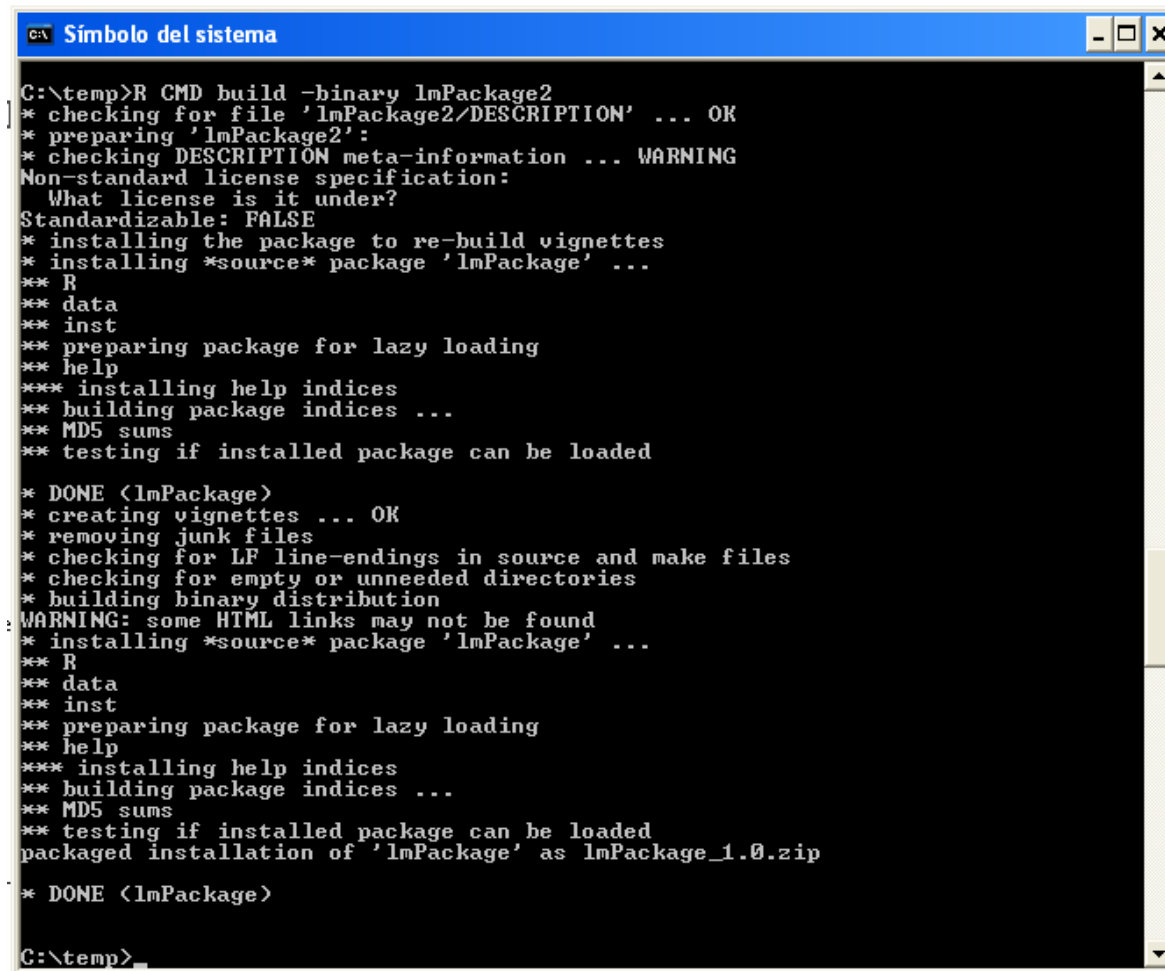
C:\temp>_
```

Ejecutar: R CMD build lmModFinal

Curso de R avanzado

Crear una librería para Windows (.zip) desde la consola

R CMD build -binary



```
C:\temp>R CMD build -binary lmPackage2
* checking for file 'lmPackage2/DESCRIPTION' ... OK
* preparing 'lmPackage2':
* checking DESCRIPTION meta-information ... WARNING
Non-standard license specification:
What license is it under?
Standardizable: FALSE
* installing the package to re-build vignettes
* installing *source* package 'lmPackage' ...
** R
** data
** inst
** preparing package for lazy loading
** help
*** installing help indices
** building package indices ...
** MD5 sums
** testing if installed package can be loaded

* DONE (lmPackage)
* creating vignettes ... OK
* removing junk files
* checking for LF line-endings in source and make files
* checking for empty or unneeded directories
* building binary distribution
WARNING: some HTML links may not be found
* installing *source* package 'lmPackage' ...
** R
** data
** inst
** preparing package for lazy loading
** help
*** installing help indices
** building package indices ...
** MD5 sums
** testing if installed package can be loaded
packaged installation of 'lmPackage' as lmPackage_1.0.zip
* DONE (lmPackage)

C:\temp>
```

Ejecutar: R CMD build -binary lmModFinal

Curso de R avanzado

Testar (check) una librería desde RStudio

The screenshot shows the RStudio interface with the following components:

- Console:** Displays the R version (3.3.2) and the 'Sincere Pumpkin Patch' message. It also shows the R license and contributors information.
- Build Menu:** The 'Build' menu is open, showing options like 'Build & Reload', 'Check', and 'More'. The 'Check' option is highlighted with a blue circle and an arrow pointing to it from the text '2º Ir a Build'.
- File Explorer:** The file explorer shows the project files. The file 'ImPackage.Rproj' is highlighted with a blue circle and an arrow pointing to it from the text '1º Abrir proyecto'.
- Environment Panel:** The 'Environment' panel is visible, showing the 'devtools::check()' command being executed. It also shows the output of the command, including the R flags and the package name 'ImPackage'.

Annotations:

- 2º Ir a Build:** Points to the 'Check' option in the Build menu.
- 3º check:** Points to the 'Check' option in the Build menu.
- 1º Abrir proyecto:** Points to the 'ImPackage.Rproj' file in the file explorer.

Curso de R avanzado

Compilar/Cargar una librería desde RStudio

2º Ir a Build

3º Build & Reload

1º Abrir proyecto

Console C:/Juan/CREAL/cursos/BioTools/Day_1/lmModFinal/

```
R version 3.3.2 (2016-10-31) -- "Sincere Pumpkin Patch"
Copyright (C) 2016 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-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.

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.

Restarting R session...

> library(lmModFinal)
> |
```

Environment History Build

Build & Reload Check More

```
==> Rcmd.exe INSTALL --no-multiarch --with-keep.source lmModFinal

* installing to library 'C:/Users/jrgonzalez/Documents/R/win-library/3.3'
* installing *source* package 'lmModFinal' ...
No man pages found in package 'lmModFinal'
** help
*** installing help indices
```

Files Plots Packages Help Viewer

New Folder Delete Rename More

	Name	Size	Modif
	..		
	.gitignore	32 B	May
	.Rbuildignore	30 B	May
	DESCRIPTION	354 B	May
	lmModFinal.Rproj	406 B	May
	man		
	NAMESPACE	48 B	May
	R		

Creación de librerías con R

Envío a CRAN

- CRAN es una red de sitios WWW que contienen el código distribuido y sus distribuciones, especialmente los paquetes de R
 - Antes de someter un paquete ejecuta R CMD check para **verificar** que el paquete se instalará y que los ejemplos funcionan, así como que la documentación está completa y puede ser procesada
 - Crea el archivo ``.tar.gz'` mediante R CMD build
 - Asegúrate que se puede ejecutar todo el proceso sólo con **warnings** que entiendes y ten razones para no corregirlos
 - Después de testar todo esto, sube el archivo ``.tar.gz'` utilizando ``anonymous'` como log-in y tu dirección de e-mail como password a <ftp://cran.R-project.org/incoming>
- y envía un mensaje a cran@r-project.org sobre él. Los ``maintainers'` de CRAN (principalmente Kurt Hornik) ejecutarán estos tests antes de ponerlo en el repositorio principal

Ejercicio: Hacer una vignette de la librería 'ImMod' (ver fichero ImMod.Rnw) usando Markdown