

Programación estadística con R

Ramón Díaz-Uriarte

Bioinformatics Unit
Centro Nacional de Investigaciones Oncológicas (CNIO)
(Spanish National Cancer Centre)
rdiaz@cnio.es
<http://ligarto.org/rdiaz>
Copyright © 2006 Ramón Díaz-Uriarte

Instituto Español de Oceanografía, Santander, Abril 2006

Outline

Introducción

Editores y GUIs para R

R: lenguaje

Importar y exportar datos

Gráficos en R

Programación en R (I)

Ejemplos parseo strings

Métodos y técnicas estadísticas disponibles

Modelos lineales

Modelos lineales generalizados

Problema de programación I: t-test-then-cluster

Problema de programación II. regresión logística, stacking y combinación de modelos

Programación en R (II)

Organización de análisis y programación

Documentación

"Horario"

- Lunes:
 - ▶ Introducción a R
 - ▶ Editores con R: Tinn-R.
 - ▶ R: lenguaje (objetos, operadores, atributos, secuencias, etc).
- Martes
 - ▶ Gráficos en R
 - ▶ Programación en R
- Miércoles
 - ▶ Programación en R (continuación).
 - ▶ Ejemplos con strings y caracteres.

● Jueves

- ▶ Modelos lineales y lineales generalizados
- ▶ Ejemplo corto programación: t-test-then-cluster
- ▶ Empezar ejemplo programación regresión logística.

● Viernes

- ▶ Terminar ejemplo programación regresión logística.
- ▶ Programación en R (II)
- ▶ Organización de análisis y programación
- ▶ Bibliografía

Código

- Todo el código podría estar disponible (en realidad, basta con hacer cut and paste).
- Mucho mejor que lo tecleeis:
 - ▶ Familiarizarse con el uso de las herramientas.
 - ▶ Detectar errores
 - ▶ Cambiar el código para experimentar con él inmediatamente.

Introducción

Qué son R y S
Instalación de R
Primera sesión
Configuración y
mantenimiento
Ayuda y
documentación
Paquetes
adicionales
Actualizando R
Algunas FAQs
habituales

GUIs

R: lenguaje

Import/Export

Gráficos en R

Progr. I

parsing
strings

Métodos y
técnicas
estadísticas
disponibles

Modelos
lineales

GLMs

Problema I

Problema II

Introducción

Qué son R y S

Instalación de R

Primera sesión

Configuración y mantenimiento

Ayuda y documentación

Paquetes adicionales

Actualizando R

Algunas FAQs habituales

Qué son R y S

- " R, also known as "GNU S", is a language and environment for statistical computing and graphics. R implements a dialect of the award-winning language S, developed at Bell Laboratories by John Chambers et al. For newcomers it provides easy access to a wide variety of statistical and graphical techniques. Advanced users are offered a full-featured programming language with which to add functionality by defining new functions." (Del folleto que teneis en las manos).
- " [S] has forever altered the way how people analyze, visualize and manipulate data" (Association of Computer Machinery Software System Award 1998 a John Chambers).
- Probablemente, S y R son los dos lenguajes más usados en investigación en estadística. Otras virtudes en el folletillo.

Qué son R y S (II)

- En pocas palabras, los grandes atractivos de R/S son:
 - ▶ La capacidad de combinar, sin fisuras, análisis "preempaquetados" (ej., una regresión logística) con análisis ad-hoc, específicos para una situación: capacidad de manipular y modificar datos y funciones.
 - ▶ Los gráficos de alta calidad (revelaciones de la visualización de datos y producción de gráficas para papers).
- La comunidad de R es muy dinámica (ej., crecimiento en número de paquetes), integrada por estadísticos de gran renombre (ej., J. Chambers, L. Terney, B. Ripley, D. Bates, etc).
- Extensiones específicas a áreas nuevas (bioinformática, geoestadística, modelos gráficos).
- Un lenguaje orientado a objetos.
- Muy parecido a Matlab y Octave, y con sintaxis que recuerda a C/C++.

R, estad. y comp. estad.

- R es tanto un sistema/lenguaje de programación estadística, como un sistema/paquete/programa estadístico.
- Tenemos a nuestra disposición muchos “módulos” (funciones, paquetes) para el análisis estadístico. Muchos más aún en CRAN.
- Al ser un lenguaje de programación con orientación al manejo y análisis de datos:
 - ▶ Podemos modificar las funciones preexistentes (ej., si no nos gusta el output por defecto del anova).
 - ▶ Podemos desarrollar nuevos métodos de análisis, explorar ideas, etc. (Esto explica, en parte, su popularidad en bioinformática).

Introducción

Qué son R y S
Instalación de R
Primera sesión
Configuración y
mantenimiento
Ayuda y
documentación
Paquetes
adicionales
Actualizando R
Algunas FAQs
habituales

GUIs

R: lenguaje

Import/Export

Gráficos en R

Progr. I

parsing strings

Métodos y técnicas estadísticas disponibles

Modelos lineales

GLMs

Problema I

Problema II

- Nos centraremos en el manejo de datos y programación.
- Veremos por encima la funcionalidad estadística pre-empaquetada existente.

¿Cuánto cuesta R? R es "GNU S"

- R es la implementación GNU de S.
- Filosofía y objetivos del proyecto GNU: www.gnu.org.
- Desarrollar un sistema completo y "libre" (donde "free is free as in freedom, not free as in beer").
- Algunos "GNUs famosos": Emacs, gcc, GNU/Linux, etc.
- R se distribuye con licencia GNU GPL o General Public License (ver <http://www.gnu.org/licenses/gpl.html>.)
- La GPL no pone ninguna restricción al uso de R. Restringe su distribución (ha de ser GPL).
- R se obtiene por 0 euros en <http://cran.r-project.org>

Obtención e instalación de R

Depende del sistema operativo, pero todo se puede encontrar en <http://cran.r-project.org/bin>.

- Windows: bajar ("download") el ejecutable desde <http://cran.r-project.org/bin/windows/base>. Ejecutar el fichero. Instalará el sistema base y los paquetes recomendados.
- GNU/Linux: dos opciones:
 - ▶ Obtener el R-x.y.z.tar.gz y compilar desde las fuentes, y también bajar los paquetes adicionales e instalar. (Buena forma de comprobar que el sistema tiene development tools).
 - ▶ Obtener binarios (ej., *.deb para Debian, *.rpm para RedHat, SuSE, Mandrake).

Instalación R: nuestro caso

- El instalador se encuentra en el CD.
- Clickear, y seguir instrucciones.
- Recomendable usar inglés si nos sentimos cómodos.
- Las opciones por defecto en cuanto a qué instalar son razonables.
- El "path": en lo que sigue asumo
c:\Archivos de programa\R\R-2.2.1. Cambiar
si es necesario.

Cambiar el path

- Es útil añadir **R/bin** al PATH de Windows.
- Nos permitirá ejecutar **R CMD BATCH**.

Inicio de una sesión de R

- Windows:

- ▶ Hacer click dos veces en el icono. Se abrirá "Rgui".
- ▶ Desde una "ventana del sistema" ejecutar "Rterm"; parecido a "R" en Unix o Linux.
- ▶ Iniciar R desde de XEmacs.

- (GNU/Linux:

- ▶ Teclear "R" en una shell.
- ▶ Iniciar R desde (X)Emacs (M-X R.).

Una primera sesión

```
> rnorm(5)
> x <- rnorm(5)
> summary(x)
> w <- summary(x)
> w
> print(w)
```


Introducción

Qué son R y S
Instalación de R

Primera sesión

Configuración y
mantenimiento

Ayuda y
documentación

Paquetes
adicionales

Actualizando R
Algunas FAQs
habituales

GUIs

R: lenguaje

Import/Export

Gráficos en R

Progr. I

parsing strings

Métodos y técnicas estadísticas disponibles

Modelos lineales

GLMs

Problema I

Problema II

```
> x1 <- rnorm(100)
> x2 <- rnorm(100)
> plot(x1, x2)
> rug(x1)
> typo.paciente <- factor(c(rep("e",
+   50), rep("s", 50)))
> plot(x1, x2, type = "n", xlab = "gen A",
+   ylab = "gen B")
> points(x1, x2, col = c("red",
+   "blue")[typo.paciente])
> par(mfrow = c(2, 2))
> typo.paciente <- factor(c(rep("Enfermo",
+   50), rep("Sano", 50)))
```

Introducción

Qué son R y S
Instalación de R

Primera sesión

Configuración y
mantenimiento

Ayuda y
documentación

Paquetes
adicionales

Actualizando R

Algunas FAQs
habituales

GUIs

R: lenguaje

Import/Export

Gráficos en R

Progr. I

parsing strings

Métodos y técnicas estadísticas disponibles

Modelos lineales

GLMs

Problema I

Problema II

```
> plot(x1, x2, type = "n", xlab = "gen A",  
+       ylab = "gen B")  
> points(x1, x2, col = c("red",  
+       "blue")[typo.paciente],  
+        pch = 19)  
> boxplot(x1 ~ typo.paciente,  
+         ylab = "Expresión normalizada",  
+         xlab = "Tipo de paciente")  
> hist(x1)  
> hist(x2, main = "Histograma de la variable B")
```

Un ejemplo más largo

- Vamos a ordenar un conjunto de datos en función del p-value del estadístico de la t. (Simulamos los datos; muestras en columnas, variables en filas.)

```
> data <- matrix(rnorm(50 * 100),  
+               ncol = 50)  
> clase <- factor(c(rep("sano",  
+                   20), rep("enfermo", 30)))  
> tmp <- t.test(data[1, ] ~ clase)  
> tmp  
> attributes(tmp)  
> tmp$p.value  
> resultado <- apply(data, 1,  
+                   function(x) t.test(x ~ clase)$p.value)  
> hist(resultado)  
> order(resultado)  
> which(resultado < 0.05)
```

Ejemplo (II)

- Además de data (ej., medidas de con la técnica A) tenemos dataB (con la técnica B). Queremos seleccionar aquellos pares con alta correlación positiva.

```
> dataB <- t(matrix(rnorm(50 *  
+      100), ncol = 50))  
> dataA <- t(data)  
> correlaciones <- apply(rbind(dataA,  
+      dataB), 2, function(x) cor(x[1:50],  
+      x[51:100]))  
> order(correlaciones)  
> which(correlaciones > 0.7)  
> hist(correlaciones)
```

Configuración y personalización

- Personalizar el aspecto de R: "Edit -> GUI preferences".
 - ▶ Poner "SDI" (no "MDI"), o cambiar y probar.
 - ▶ "Initial left" y "top" (poner 0 y 0).
- Se puede "customizar" como se inicia una sesión de R (ej., que paquetes se cargan, mensajes, etc). Ver sección 10.8 en *An introduction to R*.
- Luego editaremos Rprofile.site.

¿Desde dónde ejecutamos R?

- Hacemos `getwd()`
- Podemos cambiar de directorio: `setwd(lo.que.sea)`
- Más cómodo que ejecutemos R desde un directorio usando un "shortcut".
- Creemos un directorio (ej., CursoR).
- Copiar el shortcut y modificar propiedades.

Ayuda incluida con el programa

- `?rnorm`
- `help(rnorm)`
- `help("+")`
- `help.start()`
- `?help.search`
- `help.search("normal")`
- `?apropos`
- `apropos("normal")`
- Si hay acceso a internet: `RSiteSearch("logistic residuals")`
- Y qué hace eso? `?RSiteSearch`

Ayuda (II)

- `?demo`
- `demo(graphics); demo(persp); demo(lm.glm)`

Documentación

- Referencia más extensa al final. Pero debemos mencionar:
- Toda instalación de R incluye manuales. El primero a leer: “An introduction to R”.
- *An introduction to R: software for statistical modelling and computing*, de P. Kuhnert y W.. Venables, en http://cran.r-project.org/doc/contrib/Kuhnert+Venables-R_course_Notes.zip.
- *A guide for the unwilling S user*, de P. Burns. En http://cran.r-project.org/doc/contrib/Burns-unwilling_S.pdf o <http://www.burns-stat.com/pages/tutorials.html>. Sólo 8 páginas.
- *R para principiantes*, de E. Paradis. En <http://cran.r-project.org/other-docs.html> o http://cran.r-project.org/doc/contrib/rdebuts_es.pdf.

Paquetes adicionales

R consta de un "sistema base" y de paquetes adicionales que extienden la funcionalidad. Distintos "tipos" de paquetes:

- Los que forman parte del sistema base (ej. ctest).
- Los que no son parte del sistema base, pero son **"recommended"** (ej., survival, nlme). En GNU/Linux y Windows ya (desde 1.6.0?) forman parte de la distribución estándar.
- Otros paquetes; ej., car, paquetes de Bioconductor (como multtest, etc). Estos necesitamos seleccionarlos e instalarlos individualmente.

Instalación de paquetes adicionales

Depende del sistema operativo

- Windows:

- ▶ Desde la "GUI".
- ▶ Desde R, con "install.packages()", "update.packages()", etc.

- GNU/Linux:

- ▶ "R CMD INSTALL paquete-x.y.z.tar.gz". Permite instalar aunque uno no sea root (especificando el directorio).
- ▶ Más cómodo, desde R, "install.packages()", "update.packages()", etc. También permiten instalar no siendo root (especificar lib.loc).

Paquetes adicionales: nuestro caso

- Tenemos algunos paquetes adicionales en el CD-ROM
- Lo más sencillo: usar el menú de R: “Packages” -> “Install packages from local zip file(s)”

¿Qué paquetes están disponibles?

- Podemos hacer:

```
> library()
```
- Instalemos alguno de los paquetes en el CD.
- Otra vez:

```
> search()  
> library()
```
- Y ahora

```
> detach(2)  
> search()  
> library()
```
- Repitamos a nuestro antojo.

Actualizando R (Windows)

- Instalar R es sencillo. ¿Qué hacemos si hemos instalado muchos paquetes?
- La opción más simple (ver también la FAQ para Windows, q. 2.8):
- Instalar la nueva versión de R.
- Copiar todos los paquetes de “/library” de la vieja a la nueva versión (i.e., del viejo al nuevo directorio).
- Arrancar la nueva versión de R y hacer
`update.packages()`.
- Podemos entonces borrar la antigua versión.

Actualizando R (Linux)

- Depende de la configuración y distribución de Linux.
- Si usamos Debian, y sólo paquetes de “r-cran-”, entonces basta hacer un apt-get install.

- Para cualquier distribución podemos usar *installed.packages*, especificando el directorio(s) apropiado, si fuera necesario:

```
> paquetes <- installed.packages()[,
+   1]
> save(file = "paquetes.RData",
+   paquetes)
> q()
```

- Arrancamos la nueva versión y:

```
load('paquetes.RData')
install.packages('paquetes')
```

Memoria

- Pregunta frecuente, sobre todo para Windows.
- Sistemas a 32 bits tienen limitación en máxima memoria usable por un proceso.
- Limitación aún más restrictiva en Windows.
- Por regla general, máximo usable es 1 GB. Posible llegar a 3 GB (ver la FAQ, pregunta 2.9 , “There seems to be a limit on the memory it uses!”).
- Si necesitamos más memoria: pasar a sistemas de 64 bits, con límites muuuucho mayores.
- Si 64 bits (ej., AMD Opteron o Intel Itanium) nótese que NO existen (de momento) versiones de R para Windows, por falta de compilador apropiado.

Cómo especificar paths (rutas) en R para Windows

- En Windows es habitual hacer “C:
Un directorio
Un subdirectorio”.
- Pero el caracter “
” tiene que ser “backslashed”, poniendo otro “
” antes.
- En R, entonces:

```
> setwd("C:\\tmp\\otro")
```
- Alternativamente, más sencillo usar la notación como en Unix/Linux:

```
> setwd("C:/tmp/otro")
```
- Fijaos cómo nos devuelve R su actual directorio:

```
> getwd()
```

Compartir datos y programas entre Windows y Linux

- Los ficheros con código por supuesto se pueden abrir tanto en Windows como en Linux. (Todo lo más, algún caracter raro por la diferente codificación del “newline”).
- Pero, los ficheros RData guardados en una versión de R para Windows se pueden abrir en R para Linux, y viceversa.
- Esto es una funcionalidad excepcional: permite que puedan enviarse y compartirse ficheros con independencia del sistema operativo.

Introducción

GUIs

Introducción

Tinn-R

Otros editores y
entornos

R: lenguaje

Import/Export

Gráficos en R

Progr. I

parsing
strings

Métodos y
técnicas
estadísticas
disponibles

Modelos
lineales

GLMs

Problema I

Problema II

Prog. II

Sweave et al.

Docs.

ESS, Emacs

Editores y GUIs para R

Introducción

Tinn-R

Otros editores y entornos

Editores y GUIs para R

- Es conveniente usar un editor que nos permita conservar los scripts: mantenimiento de código ordenado y comentado.
- Mejor si hay coloreado de sintaxis, completado de paréntesis, "call tips", etc.
- Mejor aún si permite interacción directa con R (i.e., evitar el "corta-pegar", usando un "envía estas líneas a R").
- Una página con (casi) todas las posibilidades existentes: http://www.sciviews.org/_rgui/.

Editor incorporado

- R para Windows incorpora un pequeño editor de scripts.
- Vamos a “File” -> “New Script”
- Permite enviar código a una sesión en R (evita el muy oneroso “corta-pega”).
- Bastante para trabajos rápidos, pero carece de syntax highlighting, call-tips, etc.
- Examinaremos otras alternativas.

Introducción

GUIs

Introducción

Tinn-R

Otros editores y
entornos

R: lenguaje

Import/Export

Gráficos en R

Progr. I

parsing
strings

Métodos y
técnicas
estadísticas
disponibles

Modelos
lineales

GLMs

Problema I

Problema II

Prog. II

Sweave et al.

Docs.

ESS, Emacs

- Distinción entre GUI y editor.
- GUI vs. no GUI una compleja discusión.
- Aquí: usaremos un editor (Tinn-R) y probaremos una GUI (R commander).

Tinn-R

- Editor muy pequeño, con funcionalidad extra para edición de código en R.
- Permite enviar código a R.
- Se obtiene en <http://www.sciviews.org/Tinn-R>
- Tinn-R es también parte del SciViews bundle.
- Muy bien documentado (incluido en la distribución).

Instalación

- Descargar el instalador de sourceforge:
<https://sourceforge.net/projects/tinn-r>
- 0
- Usar el instalador existente en el CD.

Configuración

- Podemos arrancar Tinn-R de forma automática cada vez que arranca R:
- Arrancamos Tinn-R y abrimos 'c:/Archivos de Programa/R/R-2.2.1/etc/Rprofile.site'
- Escribimos

```
options(IDE = 'C:/Archivos de Programa/Tinn-R/  
++ bin/Tinn-R.exe') ##### en una sólo linea  
options(use.DDE = TRUE)  
library(svIDE)  
library(svSocket)  
library(svIO)  
guiDDEInstall()
```

Configuración (II)

- Aseguramos que R se ejecuta en SDI (ver "Edit -> GUI preferences").
- Cerramos R y arrancamos R de nuevo.

Configuración (II)

- Aseguramos que R se ejecuta en SDI (ver "Edit -> GUI preferences").
- Cerramos R y arrancamos R de nuevo.
- ¡Ooops, problemas! Falta svIDE.
- Oportunidad para instalarlo. Desde un local zip file.
- Cerrar R y arrancar de nuevo. (Y faltará R2HTML; instalarlo).

Configuración (III)

- Movemos la ventana de Tinn-R hacia abajo.
- Cerramos R y Tinn-R.
- Arrancamos R.

Configuración (III)

- Movemos la ventana de Tinn-R hacia abajo.
- Cerramos R y Tinn-R.
- Arrancamos R.
- Tinn-R se debería haber arrancado.
- (Si ya tenemos arrancado Tinn-R y R se arranca despues, R usará el Tinn-R arrancado).

Un ejemplo

- Tecleemos el código anterior.

```
> x1 <- rnorm(100)
> x2 <- rnorm(100)
> plot(x1, x2)
> rug(x1)
```

- Marcar, ej., las dos primeras líneas
- Vamos a "R" -> "Send to R" -> "Send region"

Ejemplo (II)

- Guardar la figura como metafile (ej., pegar en word) o postscript.
- Parar un cálculo.

Configuración (IV)

- Más importante: shortcuts. R -> Hotkeys of R.
- Yo uso:
 - Send: selection Ctrl + Enter
 - Send: line Ctrl + L
- Uso de shortcuts facilita mucho enviar código a R.

Otros

- (Probemos todos los siguientes!).
- R card
- R explorer
- File browser
- Call tips (pero veáse luego para posibles problemas).
- Match brackets (Alt + B).

Configuración

- A mi no me gustan las toolbars (quitan espacio precioso).
- Return focus after sending to R (Options)
- Options -> Main -> Application -> Active line highlighted.
- Options -> Editor -> Show line numbers
- View -> Tabs -> Position
- Guardamos la configuración: Tools -> Backup -> System configuration.

Problemas con servidor?

- Options -> Application -> R
- Problema potencial con comunicación con servidor en calltip completion.
- Cambiar el retraso hasta la sincronización.
- Usar sólo TCP/IP (después de poner `library(svSocket)`).
- Deshabilitar: 'no afecta a call tips del sistema base, pero si de algunas librerías y nuestras user-defined functions.

Otras cosas útiles de Tinn-R

- Comparar ficheros (Tools → Differences)
- Organización de proyectos.

JEdit

- Hay varios plugins para JEdit. Uno de Zed Shaw, otro de José Claudio Faria y otro de Tobias Elze. El de JC Faria parece el más reciente. Trucos de instalación en <http://finzi.psych.upenn.edu/R/Rhelp02a/archive/0735.html>.
- Múltiples sistemas operativos (es Java).
- Pero sólo provee syntax highlighting, no posibilidad de enviar a un running R process. (Podría cambiar en el futuro).

JGR

- JGR

<http://stats.math.uni-augsburg.de/JGR/>

incluye interacción con R, ayuda, calltips, editor y browser de objetos, etc.

- Múltiples sistemas operativos (es Java), aunque yo he tenido problemas en Linux.
- No permite usar R commander
- No posible el uso de keyboard shortcuts.

¿R con XEmacs?

- Colorea sintaxis, completa paréntesis, etc.
- Una interfaz común para R en distintos sistemas operativos.
- Una interfaz común para otros paquetes estadísticos (ej., SAS, XLispStat, Arc, etc) y numéricos (ej., Octave).
- Pero aunque (X)Emacs es MUCHO más que un editor...
 - ▶ (X)Emacs con ESS no es "familiar" para los usuarios de Windows (pero no tanto con las modificaciones de J. Fox).
 - ▶ Problemas no resueltos en interacción R, XEmacs, Windows.

R, ESS, XEmacs (a la J.Fox)

- Toda la información (para Windows) está en <http://socserv.mcmaster.ca/jfox/Books/Companion/ESS/>.
- Para Linux ESS se distribuye en cómodo paquete para muchas distribuciones. Todo sobre ESS en <http://stat.ethz.ch/ESS/>.
- Es posible usar ESS y (X)Emacs en Linux de forma parecida a Windows.

Eclipse

- Eclipse: <http://www.eclipse.org/>.
- Plugin en <http://www.walware.de/goto/statet>, por Stephan Wahlbrink.
- Múltiples sistemas operativos (es Java).
- Pero es muy pesado (el fichero a descargar son 100 MB, y tiene un alto consumo de memoria y CPU), la interacción con R no funciona bien en Linux, no existen calltips, y Eclipse tiene sus peculiaridades en cuanto a modus operandi.

Otros

- WinEdt (<http://www.winedt.com> y <http://cran.r-project.org/contrib/extra/winedt>). Pero WinEdt no es GPL ni gratuito (es shareware): problemas de licencia (por ej., para nosotros aquí) y no está disponible para otros sistemas operativos.
- Coloreado de sintaxis (y completado de paréntesis?) para Vim y Nedit, pero nada similar a la interacción con un buffer con R.
- Opciones para otros editores (Syn, TextPad, UltraEdit, CrimsonEditor) pero sólo coloreado, no envío de código (CrimsonEditor sí, con customización?).

Introducción

GUIs

R: lenguaje

Objetos: tipos,
atributos, nombres

Listas

NA et al.

Vectores de
caracteres

Factores

Atributos de los
objetos

Operadores

Generación de
secuencias

Ordenación e
indexación de
vectores

Matrices y arrays

data frames

La family apply

Import/Export

Gráficos en R

Progr. I

parsing
strings

Métodos y
técnicas
estadísticas
disponibles

Modelos
lineales

R: lenguaje

Objetos: tipos, atributos, nombres

Listas

NA et al.

Vectores de caracteres

Factores

Atributos de los objetos

Operadores

Generación de secuencias

Ordenación e indexación de vectores

Matrices y arrays

data frames

La family apply

Tipos de objetos (I)

- (Casi) todo en R es un objeto.
- **vector** Colección ordenada elementos del mismo tipo.

```
> x <- c(1, 2, 3); y <- c("a", "b", "Hola")  
> z1 <- c(TRUE, TRUE, FALSE)
```
- **array** Generalización multidimensional de vector.
Elementos del mismo tipo.
- **data frame** Como array, pero permiten elementos (columnas) de distintos tipos. El objeto más habitual para manejo de datos procedentes de experimentos.

```
> my.data.frame <- data.frame(ID = c("samp1",  
+   "samp2", "samp3"), variable1 = c(10,  
+   25, 33), subj2 = c(NA, 34,  
+   15), sitio1 = c(TRUE, TRUE,  
+   FALSE), loc = c(1, 30, 125))
```

Tipos de objetos (II)

- **funciones** Código.
- **factor** Un tipo de vector para datos categóricos.
- **list** Un "vector generalizado". Cada lista está formada por componentes (que pueden ser otras listas), y cada componente puede ser de un tipo distinto. Son unos "contenedores generales".

```
> una.lista <- c(un.vector = 1:10,  
+   una.palabra = "Hola", una.matriz = matrix(rnorm(20),  
+       ncol = 5), otra.lista = c(a = 5,  
+       b = factor(c("a", "b"))))
```

Listas

- Las data.frames son, en realidad, tipos especiales de listas.
- Las listas son contenedores sin estructura determinada.
- Por tanto muy flexibles, pero sin estructura.
- Muchas funciones devuelven listas: devuelven un conjunto de resultados de distinta longitud y distinto tipo.

```
● > d3 <- data.frame(g1 = runif(10),  
+                   g2 = rnorm(10), id1 = c(rep("a",  
+                   3), rep("b", 2), rep("c",  
+                   2), rep("d", 3)))  
> my.fun <- function(x) {  
+   las.medias <- mean(x[, -3])  
+   las.vars <- var(x[, -3])  
+   max.total <- max(x[, -3])  
+   tabla.clases <- table(x[,  
+   3])  
+   return(list(row.means = las.medias,  
+   row.vars = las.vars,  
+   maximum = max.total,  
+   factor.classes = tabla.clases))  
+ }  
> my.fun(d3)
```

Introducción

GUIs

R: lenguaje

Objetos: tipos,
atributos, nombres

Listas

NA et al.

Vectores de
caracteres

Factores

Atributos de los
objetos

Operadores

Generación de
secuencias

Ordenación e
indexación de
vectores

Matrices y arrays

data frames

La family apply

Import/Export

Gráficos en R

Progr. I

parsing strings

Métodos y técnicas estadísticas disponibles

Modelos lineales

```
> una.lista <- my.fun(d3)
> una.lista
> attributes(una.lista)
> names(una.lista)
> length(una.lista)
> una.lista[[4]]
> una.lista[4]
> una.lista$factor.classes
> una.lista[[3]] <- list(NULL)
> una.lista
> una.lista[[3]] <- NULL
> una.lista
> unlist(una.lista)
> otra.lista <- list(cucu = 25,
+   una.lista)
> unlist(otra.lista)
> unlist(otra.lista, recursive = FALSE)
> una.lista <- c(una.lista, otro.elemento = "una frase")
```


NA, NaN, Inf

- "NA" es el código de "Not available".
- `v <- c(1:3, NA)`
- `is.na(v); which(is.na(v))`
- `v == NA` # No funciona! Por qué?
- Sustituir NA por, p.ej., 0: `v[is.na(v)] <- 0.`
- Infinito y NaN (not a number). Son distintos de NA.
- `5/0; -5/0; 0/0`
- `is.infinite(-5/0); is.nan(0/0); is.na(5/0)`

```
• xna <- c(1, 2, 3, NA, 4); mean(xna)  
mean(xna, na.rm = TRUE)
```

• Lo mismo con otras funciones.

• Para "modelling functions" (ej., lm) lo mejor es usar "na.omit" o "na.exclude" ("na.exclude" más conveniente para generar predicciones, residuos, etc).

• Eliminar todos los NA:

```
> XNA <- matrix(c(1, 2, NA, 3, NA, 4), nrow = 3)  
> XNA  
> X.no.na <- na.omit(XNA)
```

Vectores de caracteres

- `codigos <- paste(c("A", "B"), 2:3, sep = "")`
- `codigos <- paste(c("A", "B"), 2:3, sep = ".")`
- `juntar <-
paste(c("una", "frase", "tonta"), collapse
= "")`
- `columna.a <- LETTERS[1:5]; columna.b <- 10:15;
juntar <- paste(columna.a, columna.b, sep = "")`
- `substr("abcdef", 2, 4)`
- `x <- paste(LETTERS[1:5], collapse="")
substr(x, 3, 5) <- c("uv")`
- Otras muchas funciones de manipulación de caracteres, que pueden hacer poco necesario recurrir a Awk, Python, o Perl. Ver `grep`, `pmatch`, `match`, `tolower`, `toupper`, `sub`, `gsub`, `regexpr`.

Factores

- `codigo.postal <- c(28430, 28016, 28034);
mode(codigo.postal)`
- Pero *no deberíamos* usar el código postal en, por ej., un ANOVA, como si fuera un vector numérico. El usar códigos (aparentemente) numéricos en análisis estadísticos es una frecuente fuente de errores.
- `codigo.postal <- factor(codigo.postal) # mejor.`
- Si tenemos un vector con caracteres, y lo queremos usar para un análisis, necesitamos convertirlo en un factor. Si no, R, sabiamente, no se deja.
`y <- rnorm(10); x <- rep(letters[1:5], 2);
aov(y ~ x) # error!
aov(y ~ factor(x)) # funciona.`

- A veces, al leer datos, un vector de números se convierte en factor. O queremos convertir un vector factor en numérico.

```
x <- c(34, 89, 1000); y <- factor(x); y  
as.numeric(y) # Mal:  
# los valores han sido recodificados.  
as.numeric(as.character(y)) # Bien
```

- `as.numeric(levels(f))[as.integer(f)]` es más eficiente, pero más difícil de recordar.

- Podemos fijar el orden de las etiquetas.

```
ftrl <- factor(c("alto", "bajo", "medio"))  
ftrl  
ftrl <- factor(c("alto", "bajo", "medio"),  
+           levels = c("bajo", "medio", "alto"))
```

Introducción

GUIs

R: lenguaje

Objetos: tipos,
atributos, nombres

Listas

NA et al.

Vectores de
caracteres

Factores

Atributos de los
objetos

Operadores

Generación de
secuencias

Ordenación e
indexación de
vectores

Matrices y arrays
data frames
La family apply

Import/Export

Gráficos en R

Progr. I

parsing
strings

Métodos y
técnicas
estadísticas
disponibles

Modelos
lineales

- R también cuenta con "ordered factors". Sólo difieren de los factors "normales" en algunos análisis (tipo de contrastes usados con modelos lineales —treatment vs. polinomial). No lo veremos.
- Como *medida de "higiene mental y analítica"* convirtamos en factores lo que debe serlo. Eliminará muchos errores y es "buena práctica estadística".

- Para **discretizar** datos: usar `cut`:

```
vv <- c(1, 2, 3, 7, 8, 9)
cut1 <- cut(vv, 3); summary(cut1)
cut2 <- cut(vv, quantile(vv, c(0, 1/3, 2/3, 1)),
+         include.lowest = TRUE)
summary(cut2)
```

- Ver también `split(x, f)`: divide datos del vector `x` usando los grupos definidos en `f`.

Atributos de los objetos

- `x <- 1:15; length(x)`
- `y <- matrix(5, nrow = 3, ncol = 4); dim(y)`
- `is.vector(x); is.vector(y); is.array(x)`
- `mode(x); mode(y); z <- c(TRUE, FALSE); mode(z)`
- `attributes(y); w <- list(a = 1:3, b = 5);
attributes(w)`
- `y <- as.data.frame(y); attributes(y)`
- `f1 <- function(x) {return(2 * x)}; attributes(f1);
is.function(f1)`
- `x1 <- 1:5; x2 <- c(1, 2, 3, 4, 5); typeof(x2);
typeof(x3)`
- Los atributos podemos verlos, pero también podemos cambiarlos.

Nombres de objetos

- Los nombres válidos para un objeto son combinaciones de letras, números, y el punto (".").
- Los nombres no pueden empezar con un número.
- R es "case-sensitive". $x \neq X$.
- Hay nombres reservados ("function", "if", etc).
- Otras consideraciones:
 - ▶ El uso del "." es diferente al de C++.
 - ▶ Mejor evitar nombres que R usa (ej., "c") (pero no es dramático si nos confundimos: podemos arreglarlo).
 - ▶ Las asignaciones se hacen con "<=", y es buen estilo el rodear "<=" por un espacio a cada lado.

```
c <- 4; x <- c(3, 8); x; rm(c); c(7, 9)
```

```
x<-1:5 # Mal estilo
```

```
x <- 1:5 # Mucho mejor.
```

- Las "reglas" habituales de nombrar objetos en programación (usar estilo consistente, uso razonable del ".", nombres que tengan sentido —equilibrio entre longitud y frecuencia de uso, etc). Por ej., suelo nombrar data frames con la inicial en mayúscula, pero las variables siempre en minúscula. Si varias funciones hacen cosas parecidas a objetos distinto, separo con "." (más fácil luego usar clases).
- Sobre el uso de ";". No es bueno abusar, porque hace el código *muy difícil* de leer. (Pero en estas transparencias, si no lo usara ocuparíamos muchísimo espacio).

Operaciones aritméticas con vectores (I)

- **FUNDAMENTAL:** R puede operar *sobre vectores enteros de un golpe*.

```
> x <- 1:10
```

```
> y <- x/2
```

```
> z <- x^2
```

```
> w <- y + z
```

- Si un elemento es más corto, se "recicla". Es "intuitivo" si la operación es escalar con vector. Pero también ocurre en operaciones entre vectores.

```
> x + 15
```

```
> x2 <- 1:3
```

```
> x + x2
```

- El reciclado cuando (vector, escalar) suele ser lo que queremos. Entre vectores no siempre: cuidado. (R da un warning, de momento –S4 classes dan un error).

Operaciones aritméticas con vectores (II)

- Operar sobre vectores enteros es *MUCHO MEJOR* que usar loops:
 - ▶ Mucho más claro:
 - ▶ Es la forma natural de operar sobre objetos enteros.
 - ▶ Código más fácil de entender.
 - ▶ Más sencillo de modificar y mantener.
 - ▶ Más fácil hacer debugging.
 - ▶ Más rápido de escribir (y no sólo porque muchas menos líneas!).
 - ▶ Más eficiente (en tiempo y memoria).

Operaciones aritméticas con vectores (III)

- `+`, `-`, `*`, `/`, `^`, `%%`, `%/%`.
- `log`, `log10`, `log2`, `log(x, base)`, `exp`, `sin`,
`cos`, `tan`, `sqrt`
- Otras:
 - ▶ `max`, `min`, `range`, `mean`, `var`, `sd`, `sum`, `prod`
 - ▶ `which.max`, `which.min`
 - ▶ `x <- c(3, 1, 9, 5) which.min(x) which.max(x)`
 - ▶ Y si hacemos `which(x == min(x))`
 - ▶ Menos eficiente. Y los reales?

- `pmax`, `pmin`. `max` and `min` devuelven el máximo y el mínimo de un conjunto de números; devuelven un solo número aunque les pasemos varios vectores. `pmax`, `pmin` devuelven un vector que contiene el elemento `max`, `min` en esa posición.

```
> xa <- c(1, 4, 3, 2)
> xb <- c(2, 2, 1, 3)
> xc <- c(1, 9)
> pmin(xa, xb)
> pmin(xa, xc)
> pmax(xa, xb, xc)
```

- `cumsum`, `cumprod`, `diff`.

Operadores comparativos y lógicos

- `<`, `>`, `<=`, `>=`, `==`, `!=`

- `!`, `&`, `|`, `xor()` y los parecidos `&&`, `||`

- `x <- 5; x < 5; x >= 5; x == 6; x != 5`

- `y <- c(TRUE, FALSE); !y; z <- c(TRUE, TRUE)`
`xor(y, z)`

- `y & z; y | z`

- Las formas `&&`, `||` son "short-circuit operators" que se suelen usar dentro de `if` statements. Se aplican a vectores de longitud uno y sólo evalúan el segundo argumento si es preciso.

`if (is.numeric(x) && min(x) > 0) {lo que sea....`
`min(x)` no tiene sentido si `x` no es numérico.

- `0 + y; as.numeric(y); mode(y) <- "numeric"; y`

Interludio: comparando números reales

- De la FAQ, 7.31. “Why doesn’t R think these numbers are equal?”
- Sólo los enteros (y fracciones cuyo denominador es una potencia de 2) se pueden representar exactamente.
- Todos los demás son redondeados.
- Todos los demás incluyen cosas como $2/17$, o 1.0 .
- Por tanto “two floating point numbers will not reliably be equal unless they have been computed by the same algorithm, and not always even then. For example

```
> a <- sqrt(2)
```

```
> a * a == 2
```

```
> a * a - 2
```


- Para comparar números reales podemos usar *all.equal*.
- Una referencia clásica: David Goldberg (1991), “What Every Computer Scientist Should Know About Floating-Point Arithmetic”, ACM Computing Surveys, 23, 5 http://docs.sun.com/source/806-3568/ncg_goldberg.html.
- (Problemas causados por la comparación indebida de números reales son habituales en las listas de R.)

Operaciones de conjuntos

- `x <- 1:5; y <- c(1, 3, 7:10)`
- `union(x, y)`
- `intersect(x, y)`
- `setdiff(y, x)`
- `v <- c("bcA1", "bcA2", "blX1")`
`w <- c("bcA2", "xA3")`
- `union(v, w)`
`intersect(v, w)`
`setdiff(w, v)`
`setdiff(v, w)`

Generación de secuencias aleatorias

- Muestrear un vector (imprescindible en bootstrapping y cross-validation):

```
> sample(5)
> sample(5, 3)
> x <- 1:10
> sample(x)
> sample(x, replace = TRUE)
> sample(x, size = 2 * length(x),
+       replace = TRUE)
> probs <- x/sum(x)
> sample(x, prob = probs)
```

Ej: validación cruzada

- Validación cruzada con aprox. mismo número en cada grupo. (De Venables y Ripley, "S programming", p. 175).
- ¿Qué es validación cruzada?

```
> x2 <- rnorm(200)
> N <- length(x2)
> knumber <- 10
> index.select <- sample(rep(1:knumber,
+      length = N), N, replace = FALSE)
```

- `index.select` es el vector de índices. Ahora podemos hacer:

```
> hace.algo.con.train.y.test <- function(train,  
+   test) {  
+ }  
  
> for (sample.number in 1:knumber) {  
+   x2.train <- x2[index.select !=  
+     sample.number]  
+   x2.test <- x2[index.select ==  
+     sample.number]  
+   hace.algo.con.train.y.test(x2.train,  
+     x2.test)  
+ }
```

- ¿Por qué funciona?

Número aleatorios

- Números aleatorios: usar *rFuncionDistribucion* (con sus parámetros).

```
> rnorm(10)
> rnorm(10, mean = 13, sd = 18)
> runif(15)
> runif(8, min = 3, max = 59)
```
- Muchas funciones de distribución: *lognormal*, *t*, *F*, χ^2 , *exponential*, *gamma*, *beta*, *poisson*, *binomial*, etc. Ver, por ejemplo, sección 8.1 (p. 34) de *An introduction to R* y p. 16 de *R para principiantes*, de E. Paradis. (Con esas funciones podemos también obtener la densidad, quantiles y función cumulativa de todas estas distribuciones).

Introducción

GUIs

R: lenguaje

Objetos: tipos,
atributos, nombres

Listas

NA et al.

Vectores de
caracteres

Factores

Atributos de los
objetos

Operadores

**Generación de
secuencias**

Ordenación e
indexación de
vectores

Matrices y arrays

data frames

La family apply

Import/Export

Gráficos en R

Progr. I

parsing
strings

Métodos y
técnicas
estadísticas
disponibles

Modelos
lineales

Generación de secuencias: *seq*

```
> x <- c(1, 2, 3, 4, 5)
> x <- 1:10
> y <- -5:3
> x <- seq(from = 2, to = 18,
+         by = 2)
> x <- seq(from = 2, to = 18,
+         length = 30)
> x <- 5:10
> y <- seq(along = x)
> z2 <- c(1:5, 7:10, seq(from = -7,
+         to = 5, by = 2))
```

Generación de secuencias: *rep*

- `rep(1, 5)`
- `x <- 1:3; rep(x, 2)`
- `y <- rep(5, 3); rep(x, y)`
- `rep(1:3, rep(5, 3))`
- `rep(x, x)`
- `rep(x, length = 8)`
- `gl(3, 5) # como rep(1:3, rep(5, 3))`
- `gl(4, 1, length = 20) #Ojo: gl genera factores`
- `gl(3, 4, label = c("Sano", "Enfermo", "Muerto"))`
- `expand.grid(edad = c(10, 18, 25),
sexo = c("Macho", "Hembra"), loc = 1:3)`
- Podemos combinar: `z5 <- c(1:5, rep(8, 3))`

Indexación de vectores

- Una de las gran virtudes de R es la flexibilidad en el acceso a los elementos (de vectores, arrays, data frames, etc).
- `x <- 1:5; x[1]; x[3]`
- `x[x > 3]`
`x > 3`
`y <- x > 3`
`x[y]`
- `x[-c(1, 4)]; y <- c(1, 2, 5); x[y]`
- `names(x) <- c("a", "b", "c", "d", "patata")`
- `x[c("b", "patata")]`

- Podemos indexar un vector de cuatro formas:
 - ▶ Un vector lógico.
 - ▶ Un vector de enteros (integers) positivos.
 - ▶ Un vector de enteros negativos.
 - ▶ Un vector de cadenas de caracteres (character strings).
- También, por supuesto, podemos indexar un vector usando cualquier expresión o función que resulte en una de las cuatro anteriores.
- No solo podemos acceder (devolver) el/los valores, sino también asignarlos:

```
x[c(1, 3)] <- c(25, 79); x[x > 3] <- 97
```

Ordenando vectores

- `x1 <- c(5, 1, 8, 3)`
- `order(x1)`
- `sort(x1)`
- `rank(x1)`
- `x1[order(x1)]`
- `x2 <- c(1, 2, 2, 3, 3, 4); rank(x2)`
- `min(x1); which.min(x1); which(x1 == min(x1))`
- `y <- c(1, 1, 2, 2); order(y, x)`
- `order` y `sort` admiten "decreasing = TRUE".

Matrices y arrays (I)

- Generalizaciones de vectores. Nos centraremos en arrays de 2 dimensiones, pero podemos usar un número arbitrario. Con dos dimensiones, `array` y `matrix` proveen similar funcionalidad, pero `matrix` es más cómoda.
- ```
a1 <- array(9, dim = c(5, 4))
```
- ```
a2 <- matrix(1:20, nrow = 5) # column-major-order,  
# como en FORTRAN.
```

```
a3 <- matrix(1:20, nrow = 5, byrow = TRUE)
```
- ```
a4 <- 1:20; dim(a4) <- c(5, 4)
```

## Matrices y arrays (II)

- Indexado de arrays es similar a vectores.
- `a4[1, 4]; a4[1, ]; a4[, 2]; a4[c(1, 3), c(2, 4)]`
- Podemos usar los cuatro métodos para vectores y también **indexar con una matriz**. En la matriz con los índices cada fila extrae un elemento; su fila (primera dimensión) se especifica por el elemento en la primera columna, y su columna por el elemento en la segunda columna.

```
> im <- matrix(c(1, 3, 2, 4), nrow = 2)
```

```
> im
```

```
> a4[im]
```

```
> # Por qué difiere de a4[c(1, 3), c(2, 4)]?
```

- El indexado con matrices se extiende también a arrays de más dos dimensiones.

- Ordenar un array usando una de las columnas:

```
o.array <- matrix(rnorm(20), ncol = 4)
o.array <- o.array[order(o.array[, 1]),]
```

- Selección aleatoria de filas:

```
sample.of.rows <-
+ o.array[sample(1:dim(o.array)[1],
+ replace = TRUE),]
```

# Matrices y arrays (III)

- `a4[1, ]` *#es un vector!*
- Para evitar "dropping indices" usar  
`a4[1, , drop = FALSE]`  
`a4[, 3, drop = FALSE]`.
- Esto mismo se aplica a arrays de más de dos dimensiones.
- "drop = FALSE" generalmente debe usarse, de forma defensiva, cuando escribamos funciones (si no, nos encontraremos con errores como *"Error in apply(a6, 2, mean) : dim(X) must have a positive length"*.)

- Operaciones matriciales: muchas disponibles incluyendo factorizaciones como `svd` y `qr`, determinantes, rango, solución de sistemas, etc. Sólo mencionamos `diag`, producto de matrices (`%*%`) y traspuesto (`t`):

```
a6 <- diag(6); diag(a4) <- -17
```

```
a4 %*% a2; a2 %*% t(a2)
```



# Matrices *rbind*, *cbind* y otros

- Para combinar vectores o arrays para obtener arrays, usamos `rbind`, `cbind`.
- ```
x1 <- 1:10; x2 <- 10:20  
a7 <- cbind(x1, x2)  
a8 <- cbind(x1, x2)  
a12 <- cbind(a2, a4)  
a9 <- matrix(rnorm(30), nrow = 5)  
cbind(a4, a6) # no funciona  
rbind(a4, a6)
```

- Las matrices tiene atributos. En particular:

```
attributes(a4)
colnames(a4) <- paste("v", 1:4, sep = "")
rownames(a4) <- paste("id", 1:5, sep = ".")
a4[, c("v1", "v3")]
attributes(a4)
colnames(a4) <- NULL
```

- Eliminar nombres de filas y columnas (dropping) a veces aumenta velocidad de algunos procedimientos (accesos a “named components” pueden ser más lentos).
- “Dropping col and row names” sólo justificado en ocasiones.

data frames

- Y si son de distinto tipo?

```
x3 <- letters[1:10]
```

```
a9 <- cbind(x1, x2, x3)
```

data frames

- Y si son de distinto tipo?

```
x3 <- letters[1:10]  
a9 <- cbind(x1, x2, x3)
```

- No es eso lo que quiero. Quizás quería un **data.frame**

```
a10 <- data.frame(x1, x2, x3)
```

- El indexado y subsetting de data frames como el de arrays y matrices. (Por supuesto, no hallaremos el determinante de un data frame que tenga factores)

Pero sí podemos hacer:

```
prcomp(a10[, c(1,2)])  
prcomp(a10[, c("x1", "x2")])  
prcomp(a10[, -3])
```

- Además, al hacer data.frame, los "character vectors" son convertidos a factors (lo que es una ventaja).

- Podemos convertir matrices a data frames con `as.data.frame()`.
- Por estas razones, las data frames suelen ser objetos más útiles que las arrays. Permiten lo mismo que las arrays, pero se puede guardar/acceder a otra información.
- Las data.frames también tienen `rownames`,
`colnames`.

```
attributes(a10) # pero nosotros no habíamos  
# fijado los row.names
```
- También `dimnames(a10)`.

Ordenar data frames

- (De la ayuda de “order”)

```
> x <- c(1, 1, 3:1, 1:4, 3)
> y <- c(9, 9:1)
> z <- c(2, 1:9)
> dd <- transform(data.frame(x,
+   y, z), z = factor(z, labels = LETTERS[9:1]))
> dd[order(x, -y, z), ]
> dd[do.call(order, dd), ]
> dd[order(x, y, z), ]
```

- Y qué hace `do.call`?

data.frames: \$ y attach

- Usar \$ facilita el acceso y creación de nuevas columnas.

```
> set.seed(1) # fijar la semilla
> ## del random number generator
> d1 <- data.frame(g1 = runif(10), g2 = rnorm(10))
> d1$edad <- c(rep(20, 5), rep(40, 5))
>
> set.seed(1)
> d2 <- cbind(g1 = runif(10), g2 = rnorm(10))
> d2[, 3] <- c(rep(20, 5), rep(40, 5)) # error
> d2 <- cbind(d2, edad = c(rep(20, 5), rep(40,
5)))
> #OK
```

Introducción

GUIs

R: lenguaje

Objetos: tipos,
atributos, nombres

Listas

NA et al.

Vectores de
caracteres

Factores

Atributos de los
objetos

Operadores

Generación de
secuencias

Ordenación e
indexación de
vectores

Matrices y arrays

data frames

La family apply

Import/Export

Gráficos en R

Progr. I

parsing
strings

Métodos y
técnicas
estadísticas
disponibles

Modelos
lineales

- Si usamos mucho los datos de un data.frame, podemos acceder a ellos directamente:

```
> attach(d1)
> g1
> edad
> plot(g1, g2) # en vez de plot(d1$g1, d1$g2)
```


Precauciones con `attach()`

- Con `attach()` ponemos el data frame nombrado en el **search path** (en la posición 2).

- Cierta cuidado, por masking.

```
> edad <- 25
```

```
> edad # usando la que está antes
```

- `search()` #el search path

- Además, la actualización no es dinámica:

```
> d1$g3 <- "nueva columna"
```

```
> d1 # aquí está
```

```
> g3 # pero aquí no
```

```
> detach() # detach(d1) aquí es equivalente
```

```
> attach(d1)
```

```
> g3
```

```
> edad # y esto qué?
```

Introducción

GUIs

R: lenguaje

Objetos: tipos,
atributos, nombres

Listas

NA et al.

Vectores de
caracteres

Factores

Atributos de los
objetos

Operadores

Generación de
secuencias

Ordenación e
indexación de
vectores

Matrices y arrays

data frames

La family apply

Import/Export

Gráficos en R

Progr. I

parsing
strings

Métodos y
técnicas
estadísticas
disponibles

Modelos
lineales

En "entornos confusos" (ej., un análisis que se prolonga 2 semanas) mejor evitar `attach()` y acceder siempre a las variables usando su localización explícita y completa.

subset

- ```
d3 <- data.frame(g1 = runif(10), g2 = rnorm(10),
 id1 = c(rep("a", 3), rep("b", 2), rep("c", 2),
 rep("d", 3)))
```
- Los que no son "a" ni "b"  

```
d3[!(d3$id1 %in% c("a", "b")),]
```
- O usar subset  

```
subset(d3, !(id1 %in% c("a", "b")))
```
- Por supuesto, podemos usar reglas más complejas (ej., que sea igual a una variable, distinto en otra, y no missing en una tercera).
- subset también permite la selección de variables (columnas); el argumento "select".

## aggregate, etc

- Escenario: un data frame (datos); varias réplicas de la misma especie (especie); queremos "colapsar" por especie. Si es una variable numérica, hallar la mediana; si categórica, sólo debería haber un valor: devolver ese (si hay más valores, señalarlo.)

```
colapsar <- function(x) {
 + if(is.numeric(x))
 + return(median(x, na.rm = TRUE))
 + else {
 + tmp <- unique(x)
 + if(length(tmp) > 1) return("Varios!")
 + else return(as.character(tmp[1]))
 + }
}
```

- ```
d.collapse.by.species <- aggregate(datos,  
  list(id = datos[, "species"]),  
  FUN = colapsar)  
d.collapse.by.species <- d.collapse.by.species[,  
-1]  
# Asegurarse que se ha echo lo que queríamos!!!
```
- Ahora, preparemos un fichero apropiado y usemos esa función.
- Otras funciones para objetivos similares: ver *?reshape*,
?merge

La familia `apply`

```
> ax <- matrix(rnorm(20), ncol = 5)
> medias.por.fila <- apply(ax,
+   1, mean)
> por.si.na <- apply(ax, 1, mean,
+   na.rm = TRUE)
> mi.fl <- function(x) {
+   return(2 * x - 25)
+ }
> mi.fl.por.fila <- apply(ax,
+   1, mi.fl)
> mas.simple <- apply(ax, 1, function(x) {
+   return(2 * x - 25)
+ })
```

Introducción

GUIs

R: lenguaje

Objetos: tipos,
atributos, nombres

Listas

NA et al.

Vectores de
caracteres

Factores

Atributos de los
objetos

Operadores

Generación de
secuencias

Ordenación e
indexación de
vectores

Matrices y arrays
data frames

La family apply

Import/Export

Gráficos en R

Progr. I

parsing
strings

Métodos y
técnicas
estadísticas
disponibles

Modelos
lineales

```
> media.por.columna <- apply(ax,  
+   2, mean)  
> sample.rows <- apply(ax, 1,  
+   sample)  
> dos.cosas <- function(y) {  
+   return(c(mean(y), var(y)))  
+ }  
> apply(ax, 1, dos.cosas)  
> t(apply(ax, 1, dos.cosas))
```

Introducción

GUIs

R: lenguaje

Objetos: tipos,
atributos, nombres

Listas

NA et al.

Vectores de
caracteres

Factores

Atributos de los
objetos

Operadores

Generación de
secuencias

Ordenación e
indexación de
vectores

Matrices y arrays

data frames

La family apply

Import/Export

Gráficos en R

Progr. I

parsing
strings

Métodos y
técnicas
estadísticas
disponibles

Modelos
lineales

- Operar con apply generalmente mucho más eficiente que loops (además de más claro, más fácil, etc, etc).
- `sapply`, `lapply` son como `apply` pero no hay que especificar el "margin"; `sapply` intenta simplificar el resultado a un vector o a una matriz (la "s" es de "simplify"), pero `lapply` siempre devuelve una lista. Ambas pueden aplicarse a vectores, listas, arrays.

Un ejemplo inútil de apply

- Una forma tonta de generar datos de distribuciones normales con una media y desviación típicas dadas:

```
> parameters <- cbind(mean = -5:5, sd = 2:12)
> data <- t(apply(parameters, 1, function(x)
+           rnorm(10000, x[1], x[2]))))
> apply(data, 1, mean); apply(data, 1, sd)
```
- Este es un ejemplo tonto, por supuesto. Es más sencillo (y matemáticamente equivalente) hacer:

```
> z.data <- matrix(rnorm(10000 * 11), nrow = 11)
> data2 <- (z.data * parameters[, 2]) +
+         parameters[, 1]
>
> apply(data2, 1, mean); apply(data2, 1, sd)
```

tapply, table

- ```
> # see MASS, p. 38
> library(MASS)
> data(quine) # absentismo escolar
> attach(quine)
> tapply(Days, Age, mean)
> tapply(Days, list(Sex, Age), mean)
> tapply(Days, list(Sex, Age),
+ function(y) sqrt((var(y)/length(y))))
```
- **Una tabla**  

```
> table(Sex, Age)
```

- Algunas funciones directamente hacen un "apply".

```
> m1 <- matrix(1:20, ncol = 5)
> d1 <- as.data.frame(m1)
> mean(x1); mean(d1); sd(x1); sd(d1); median(m1);
median(d1)
```

- `apply`, `sapply`, `lapply` y `tapply` son funciones *muy útiles* que contribuyen a hacer el código más legible, fácil de entender, y facilitan posteriores modificaciones y aplicaciones.

- ¿Y "mapply"?
- Cada vez que vayamos a usar un "loop" explícito, intentemos substituirlo por algún miembro de *la ilustre familia apply*.

## Miscel. (cov, cor, outer)

- Dos funciones que se aplican directamente sobre matrices:

```
> cov(m1)
> cor(m1) #demasiados dígitos ...
> round(cor(m1), 3)
> # cor.test hace otra cosa
> cor.test(m1[, 1], m1[, 2])
> cor(apply(m1, 2, rank)) # corr. rangos
(Spearman)
```

- `tabla.multiplicación <- outer(11:15, 11:20, "*")`
- A `outer` se pueden pasar funciones mucho más complejas; por ej., `outer` se usa en el ejemplo de `persp`.

```
?persp # y ejecutemos el primero ejemplo
par(mfrow(1,2)); ?image # y otro ejemplo
```

- `outer` requiere funciones vectorizadas (`apply` no). Ver FAQ, 7.19 para más detalles.

# tapply para bootstrap

- Generar bootstrap samples con mismo número de muestras por grupo que en muestra original.

```
> x <- 1:25
> y <- factor(c(rep("a", 5), rep("b",
+ 10), rep("c", 15)))
> N <- length(x)
> sample.again <- TRUE
> while (sample.again) {
+ bootsample <- unlist(tapply(1:N,
+ y, function(x) sample(x,
+ size = length(x),
+ replace = TRUE)))
+ nobootsample <- setdiff(1:N,
+ bootsample)
+ if (!length(nobootsample))
+ sample.again <- TRUE
+ else sample.again <- FALSE
+ }
```

## Introducción

## GUIs

## R: lenguaje

Objetos: tipos,  
atributos, nombres

Listas

NA et al.

Vectores de  
caracteres

Factores

Atributos de los  
objetos

Operadores

Generación de  
secuencias

Ordenación e  
indexación de  
vectores

Matrices y arrays

data frames

**La family apply**

## Import/Export

## Gráficos en R

## Progr. I

parsing  
strings

Métodos y  
técnicas  
estadísticas  
disponibles

Modelos  
lineales

```
> bootsample
> nobootsample
```

# Resumen (y apología de R)

- Una vez que los datos están en R, su manipulación es *mu*y flexible.
- Podemos seleccionar variables, casos, subsecciones de datos, etc, de acuerdo con criterios arbitrarios (que usan, además, condiciones que pueden implicar a un número arbitrario de variables y casos).
- Los data frames y las matrices pueden separarse, juntarse, cambiarse de forma (reshape), etc.
- El indexado y selección de casos pueden usar números, factores, cadenas de caracteres, etc.
- Podemos preparar código que repita las mismas operaciones con datos semejantes (i.e., podemos automatizar el proceso con sencillez).

Introducción

GUIs

R: lenguaje

Objetos: tipos,  
atributos, nombres

Listas

NA et al.

Vectores de  
caracteres

Factores

Atributos de los  
objetos

Operadores

Generación de  
secuencias

Ordenación e  
indexación de  
vectores

Matrices y arrays  
data frames

La family apply

Import/Export

Gráficos en R

Progr. I

parsing  
strings

Métodos y  
técnicas  
estadísticas  
disponibles

Modelos  
lineales

- Podemos verificar "on-the-fly" que estas transformaciones hacen lo que queremos que hagan (mirando selectivamente los resultados, o "emulando" el proceso en unos datos artificiales más pequeños).
- Por tanto, una vez que los datos están en R, no hay muchas razones para exportarlos y hacer la selección y manipulación con otros lenguajes (ej., Python, Perl) para luego volver a leerlos en R.



## Importar y exportar datos

# Importando y exportando datos

- `read.table`
- `write.table`
- `save.image`
- `data`

## Importando datos

- R es un tanto inflexible en la importación de datos. En ocasiones, puede ser necesario un preprocesado de los datos con otro programa (Python, Perl).
- Hay funciones para importar datos en formato binario de SAS, SPSS, Minitab, Stata, S3 (ver package foreign (<http://cran.r-project.org/src/contrib/PACKAGES.html#foreign>)). También de bases de datos. Ver *R data import/export*.
- No hay, por el momento, formas sencillas de importar directamente datos en formato binario Excel.
- Por tanto, datos en Excel habrán de ser exportados o salvados desde Excel como texto (u ODBC, pero no lo veremos).
- (Para usuarios de GNU/Linux: con ficheros de Excel de tamaño moderado, Gnumeric generalmente es más rápido que OpenOffice. Con ficheros muy grandes, Gnumeric parece atragantarse.)

## Importando datos: read.table

- Queremos leer un fichero simple, separado por tabuladores.
- ```
my.data.frame <- read.table("mi.fichero",  
+   header = TRUE, sep = "\t",  
+   comment.char = "")
```
- Si el carácter decimal no es un punto sino, por ej., una coma, usar: `dec = ",", "`.
- Se pueden saltar líneas (`skip`) o leer un número fijo de líneas (`nrows`).
- Otro separador habitual de columnas es `sep = " "`.
- Hay funciones especializadas para otros ficheros (ej., `read.csv`) pero son casos específicos de `read.table`.

- Ojo con `comment.char`. El "default" es "#" lo que puede causar problemas si ese carácter se usa en nombres (como en muchos ficheros de arrays).
- Podemos especificar el código para "missing data" (por defecto usa NA y el que no haya valor —columna vacía).
- Muchos errores de lectura relacionados con distinto número de columnas. Usar `count.fields` para examinar donde está el problema.
- `scan` es una función más general, útil con conjuntos de datos grandes o cuando poca memoria.

scan: mini-problema de programación

- Comprobar si scan es más o menos rápido que read.table.
- Generemos un fichero largo y exportemoslo como texto.
- Leamos tanto con scan como con read.table.
- Podemos evaluar velocidad usando "system.time".

Importando de Excel

- Lo mejor es exportar desde Excel como fichero de texto separado por tabuladores.
- Ojo con las últimas columnas y missing data (Excel elimina los "trailing tabs"). Dos formas de minimizar problemas:
 - ▶ Usar "NA" para missing.
 - ▶ Poner una última columna con datos arbitrarios (ej., una columna llena de 2s).
- Cuidado también con líneas extra al final del fichero.
- Salvamos como texto (sólo salvamos una de las hojas).
- Importamos en R con `read.table`.

Exportando datos

- Lo más sencillo es exportar una tabla.
- ```
write.table(my.data.frame, file =
"mi.output.txt",
+ sep = "\t", row.names = FALSE,
+ col.names = TRUE)
```
- `row.names` y `col.names`: TRUE o FALSE?



# Guardando datos

- Guardar datos, funciones, etc, para ser usados en otras sesiones de R.
- Datos pueden compartirse entre sesiones de R en distintos sistemas operativos.
- ```
> a1 <- rnorm(10)
> a2 <- 1:10
> a3 <- letters[10:20]
> save("unos.datos.guardados.RData", a1, a2, a3)
```
- Los leemos con

```
> load("unos.datos.salvados.RData")
```
- Podemos salvar todos los objetos con

```
> save.image() # salvado como ".RData"
> save.image(file = "un.nombre.RData")
```
- El fichero ".RData" es cargado al iniciarse R.

Cargando built-in data

- R, y muchos paquetes, incorporan ficheros con datos.
- Se cargan con `load(nombre.fichero)`.
- ```
rm(list = ls()) # borrar todo del workspace
para ver efectos de siguientes comandos
library(car)
data(Davis)
o data(Davis, package = "car") si no
hemos attached la librería
search()
?Davis
```

Introducción

GUIs

R: lenguaje

Import/Export

Gráficos en R

Introducción a los  
gráficos

Boxplots

Jittering

Identificación  
interactiva de

puntos

Múltiples gráficos  
por ventana

Conditioning plots y  
datos multivariantes

Guardando gráficos

Otros

Progr. I

parsing  
strings

Métodos y  
técnicas  
estadísticas  
disponibles

Modelos  
lineales

GLMs

Problema I

Problema II

## Gráficos en R

Introducción a los gráficos

Boxplots

Jittering

Identificación interactiva de puntos

Múltiples gráficos por ventana

Conditioning plots y datos multivariantes

Guardando gráficos

Otros

# Introducción a los gráficos

- R incluye *muchas y variadas* funciones para hacer gráficos.
- El sistema permite desde simples plots a figuras de calidad para incluir en artículos y libros.
- Sólo examinaremos la superficie. Más detalles en el capítulo 4 de *Modern applied statistics with S*; los capítulos 3 y 7 de *An R and S-PLUS companion to applied regression*; el capítulo 4 de *R para principiantes*; el capítulo 12 de *An introduction to R*.
- También *demo (graphics)*.

# plot()

- `plot()` función gráfica básica.
- *# Modificado de "Introductory statistics with R"*  

```
x <- runif(50, 0, 4); y <- runif(50, 0, 4)
plot(x, y, main = "Título principal",
 + sub = "subtítulo", xlab = "x label",
 + ylab = "y label", xlim = c(-1, 5),
 + ylim = c(1, 5))
abline(h = 0, lty = 1); abline(v = 0, lty = 2)
text(1, 4, "Algo de texto")
mtext("mtext", side = 1)
mtext("mtext en side 2", side = 2, line = -3,
 + cex = 2)
```

## ● Variaciones de plot

```
plot(x)
z <- cbind(x,y)
plot(z)
plot(y ~ x)
plot(log(y + 1) ~ x) # transformacion de y
plot(x, y, type = "p")
plot(x, y, type = "l")
plot(x, y, type = "b")
plot(c(1,5), c(1,5))
legend(1, 4, c("uno", "dos", "tres"), lty = 1:3,
 + col = c("red", "blue", "green"),
 + pch = 15:17, cex = 2)
```

Introducción

GUIs

R: lenguaje

Import/Export

Gráficos en R

Introducción a los  
gráficos

Boxplots

Jittering

Identificación  
interactiva de  
puntos

Múltiples gráficos  
por ventana

Conditioning plots y  
datos multivariantes

Guardando gráficos  
Otros

Progr. I

parsing  
strings

Métodos y  
técnicas  
estadísticas  
disponibles

Modelos  
lineales

GLMs

Problema I

Problema II

- Con `text` podemos representar caracteres de texto directamente:

```
sexo <- c(rep("v", 20), rep("m", 30))
plot(x, y, type = "n")
text(x, y, labels = sexo)
```

# plot: pch, col, lty

- `plot(x, y, type = "n")`  
`points(x, y, pch = 3, col = "red")`
- Tipos de puntos:  
`plot(c(1, 10), c(1, 3), type = "n", axes = FALSE,`  
`+ xlab = "", ylab="")`  
`points(1:10, rep(1, 10), pch = 1:10, cex = 2,`  
`+ col = "blue")`  
`points(1:10, rep(2, 10), pch = 11:20, cex = 2,`  
`+ col = "red")`  
`points(1:10, rep(3, 10), pch = 21:30, cex = 2,`  
`+ col = "blue", bg = "yellow")`



- Tipos de líneas:

```
plot(c(0, 10), c(0, 10), type = "n", xlab = "",
 + ylab = "")
for(i in 1:10)
 + abline(0, i/5, lty = i, lwd = 2)
```

- `lty` permite especificaciones más complejas (longitud de los segmentos que son alternativamente dibujados y no dibujados).
- `par` controla muchos parámetros gráficos. Por ejemplo, `cex` puede referirse a los "labels" (`cex.lab`), otro, `cex.axis`, a la anotación de los ejes, etc.
- Hay muchos más colores. Ver `palette`, `colors`.

# Boxplots

- Muy útiles para ver rápidamente las características de una variable, o comparar entre variables.
- ```
attach(Y)
boxplot(weight)
plot(sexo, weight)
detach()
boxplot(weight ~ sexo, data = Y,
        + col = c("red", "blue"))
```
- `boxplot` tiene muchas opciones; se puede modificar el aspecto, mostrarlos horizontalmente, en una matriz de boxplots, etc. Vea la ayuda (`?boxplot`).

Jittering en scatterplots

- Los datos cuantitativos discretos pueden ser difíciles de ver bien:

```
dc1 <- sample(1:5, 500, replace = TRUE)
dc2 <- dc1 + sample(-2:2, 500, replace = TRUE,
+      prob = c(1, 2, 3, 2, 1)/9)
plot(dc1, dc2)
plot(jitter(dc1), jitter(dc2))
```

- También útil si sólo una de las variables está en pocas categorías.

Identificación interactiva de datos

- ```
> x <- 1:10
> y <- sample(1:10)
> nombres <- paste("punto", x, ".", y, sep = "")
>
> plot(x, y)
> identify(x, y, labels = nombres)
```
- **locator devuelve la posición de los puntos.**

```
> plot(x, y)
> locator()
> text(locator(1), "el marcado", adj = 0)
```

# Múltiples gráficos por ventana

- Podemos mostrar muchos gráficos en el mismo dispositivo gráfico. La función más flexible y sofisticada es `split.screen`, bien explicada en *R para principiantes*, secc. 4.1.2 (p. 30).
- Mucho más sencillo es `par(mfrow=c(filas, columnas))`
- ```
par(mfrow = c(2, 2))  
plot(rnorm(10))  
plot(runif(5), rnorm(5))  
plot(runif(10))  
plot(rnorm(10), rnorm(10))
```

Datos multivariantes

- Una "pairwise scatterplot matrix":

```
> X <- matrix(rnorm(1000), ncol = 5)
> colnames(X) <- c("a", "id", "edad", "loc",
+   "weight")
> pairs(X)
```

- "Conditioning plots" (revelan, entre otros, interacciones):

```
> Y <- as.data.frame(X)
> Y$sexo <- as.factor(c(rep("Macho", 80),
+   rep("Hembra", 120)))
> coplot(weight ~ edad | sexo, data = Y)
> coplot(weight ~ edad | loc, data = Y)
> coplot(weight ~ edad | loc + sexo, data = Y)
```

- La librería **lattice** permite lo mismo, y mucho más, que coplot. Ver secc. 4.6 de *R para principiantes*.

Añadición de rectas de regresión

- Podemos añadir muchos elementos a un gráfico, además de leyendas y líneas rectas:
- ```
> x <- rnorm(50)
> y <- rnorm(50)
> plot(x, y)
> lines(lowess(x, y), lty = 2)
> plot(x, y)
> abline(lm(y ~ x), lty = 3)
```
- Podemos añadir otros elementos con "panel functions" en otras funciones (como pairs, lattice, etc).

Introducción

GUIs

R: lenguaje

Import/Export

Gráficos en R

Introducción a los  
gráficos

Boxplots

Jittering

Identificación  
interactiva de

puntos

Múltiples gráficos  
por ventana

Conditioning plots y  
datos multivariantes

Guardando gráficos

Otros

Progr. I

parsing  
strings

Métodos y  
técnicas  
estadísticas  
disponibles

Modelos  
lineales

GLMs

Problema I

Problema II

- Lo más sencillo es usar `panel.car` y `scatterplot.matrix`, en el paquete "car".

```
> library(car)
> coplot(a ~ edad | loc, panel = panel.car,
+ data = X)
> scatterplot.matrix(X, diagonal = "density")
```



# Guardando los gráficos

- En Windows, podemos usar los menús y guardar con distintos formatos.
- Podemos guardar al “portapapeles”.
- También podemos especificar donde queremos guardar el gráfico:

```
> pdf(file = "f1.pdf", width = 8, height = 10)
> plot(rnorm(10))
> dev.off()
```

- O bien, podemos copiar una figura a un fichero: >  

```
plot(runif(50))
> dev.copy2eps()
```

- También podemos usar comandos como `dev.copy()`, `dev.copy2eps()`, `pdf()` (ver ?`dev.copy`).
- Guardar la figura (en formato pdf o emf o lo que sea) es más útil que copiar al clipboard para:
  - ▶ Incluir en documentos de  $\text{\LaTeX}$  (en ps o pdf) y en algunos casos de Word (como wmf; ej., si nos hacen usar macros que piden inserción de figuras).
  - ▶ Para mantener una copia de la figura en formato no Window-céntrico (ej., en formato pdf) que podemos mandar por email, etc.
  - ▶ Para añadir figuras en páginas Web.
  - ▶ Recordad que formatos como pdf, o ps son más "portables" que metafile.
- Por defecto, yo siempre uso `dev.copy2eps()` y `pdf()`.
- En GNU/Linux no hay un "File/Save As" para las figuras.

# Otros

- Podemos modificar márgenes exteriores de figuras y entre figuras (vease `?par` y búsquense `oma`, `omi`, `mar`, `mai`; ejemplos en *An introduction to R*, secc. 12.5.3 y 12.5.4.
- También **gráficos 3D**: `persp`, `image`, `contour`; **histogramas**: `hist`; **gráficos de barras**: `barplot`; **gráficos de comparación de cuantiles**, usados para **comparar la distribución** de dos variables, o la distribución de unos datos frente a un estándar (ej., distribución normal): `qqplot`, `qqnorm` y, en paquete "car", `qq.plot`.

- Notación matemática (`plotmath`) y expresiones de texto arbitrariamente complejas.
- Gráficos tridimensionales dinámicos con XGobi y GGobi. (Ver <http://cran.r-project.org/src/contrib/PACKAGES.html#xgobi>, <http://www.ggobi.org>, <http://www.stat.auckland.ac.nz/~kwan022/pub/gguide.pdf>). El paquete JGR et al. pueden (?) ofrecer alternativas similares o mejores.

Introducción

GUIs

R: lenguaje

Import/Export

Gráficos en R

Progr. I

Definición de  
funciones

Control de  
ejecución

Cuando algo va  
mal: browser y  
debug

Ambito: lexical  
scope

Try

Profiling

parsing  
strings

Métodos y  
técnicas  
estadísticas  
disponibles

Modelos  
lineales

GLMs

Problema I

Problema II

Prog. II

## Programación en R (I)

Definición de funciones

Control de ejecución

Cuando algo va mal: browser y debug

Ambito: lexical scope

Try

Profiling

# Definición de funciones

- Ya hemos definido varias funciones. Aquí una más:

```
my.f2 <- function(x, y) {
 + z <- rnorm(10)
 + y2 <- z * y
 + y3 <- z * y * x
 + return(y3 + 25)
}
```

- Lo que una función devuelve puede ser un simple número o vector, o puede producir una gráfica, o devolver una lista o un mensaje.

# Argumentos

- ```
otra.f <- function(a, b, c = 4, d = FALSE) {  
  +   x1 <- a * z ...}
```
- Los argumentos "a" y "b" tienen que darse en el orden debido o, si los nombramos, podemos darlos en cualquier orden:

```
otra.f(4, 5)  
otra.f(b = 5, a = 4)
```
- Pero los argumentos con nombre siempre se tienen que dar después de los posicionales

```
otra.f(c = 25, 4, 5) # error
```
- Los argumentos "c" y "d" tienen "default values". Podemos especificarlos nosotros, o no especificarlos (i.e., usar los valores por defecto).
- `args(nombre.funcion)` nos muestra los argumentos (de cualquier función).

Introducción

GUIs

R: lenguaje

Import/Export

Gráficos en R

Progr. I

Definición de
funciones

Control de
ejecución

Cuando algo va
mal: browser y
debug

Ámbito: lexical
scope

Try
Profiling

parsing
strings

Métodos y
técnicas
estadísticas
disponibles

Modelos
lineales

GLMs

Problema I

Problema II

Prog. II

- "... " permite pasar argumentos a otra función:

```
> f3 <- function(x, y, label = "la x", ...) {  
+   plot(x, y, xlab = label, ...)  
+ }  
>  
> f3(1:5, 1:5)  
> f3(1:5, 1:5, col = "red")
```


Control de ejecución: *if*

- Condicional: *if*
- *if (condicion.logica) instruccion* donde "*instruccion*" es cualquier expresión válida (incluida una entre `{ }`).
- *if (condicion.logica) instruccion else instruccion.alternativa.*
- ```
> f4 <- function(x) {
+ if(x > 5) print("x > 5")
+ else {
+ y <- runif(1)
+ print(paste("y is ", y))
+ }
+ }
```

## Progr. I

Definición de  
funciones

Control de  
ejecución

Cuando algo va  
mal: browser y  
debug

Ámbito: lexical  
scope

Try  
Profiling

parsing  
strings

Métodos y  
técnicas  
estadísticas  
disponibles

Modelos  
lineales

GLMs

Problema I

Problema II

Prog. II

- **ifelse es una versión vectorizada:**

```
> #from Thomas Unternährer, R-help, 2003-04-17
> odd.even <- function(x) {
+ ifelse(x %% 2 == 1, "Odd", "Even")
+ }
```

- ```
mtf <- matrix(c(TRUE, FALSE, TRUE, TRUE),  
+   nrow = 2)  
ifelse(mtf, 0, 1)
```

Control de ejecución: *while, for*

- `while (condicion.logica) instruccion`
- `for (variable.loop in valores) instruccion`
- `for(i in 1:10) cat("el valor de i es", i, "\n")`
- `continue.loop <- TRUE`
`x <- 0`
`while(continue.loop) {`
`+ x <- x + 1`
`+ print(x)`
`+ if(x > 10) continue.loop <- FALSE`
`}`
- `repeat`, `switch` también están disponibles.
- `break`, para salir de un loop.

Cuando algo va mal: browser y debug

- Cuando se produce un **error**, `traceback()` nos informa de la secuencia de llamadas antes del crash de nuestra función. Util cuando se produce mensajes de error incomprensibles.
- Cuando se producen **errores** o la función da **resultados incorrectos o warnings indebidos** podemos seguir la ejecución de la función.
- `browser` interrumpe la ejecución a partir de ese punto y permite seguir la ejecución o examinar el entorno; con "n" paso a paso, si otra tecla sigue ejecución normal. "Q" para salir.
- `debug` es como poner un "browser" al principio de la función, y se ejecuta la función paso a paso. Se sale con "Q".

```
> debug(my.buggy.function)
> ...
> undebug(my.buggy.function)
```

```
> my.f2 <- function(x, y) {  
+   z <- rnorm(10)  
+   y2 <- z * y  
+   y3 <- z * y * x  
+   return(y3 + 25)  
+ }
```

- Hagamos debugging

```
my.f2(runif(3), 1:4)  
debug(my.f2)  
my.f2(runif(3), 1:4)  
undebug(my.f2)  
  
# insertar un browser() y correr de nuevo
```

- Dentro del entorno de debugging podemos cambiar variables, crear variables, etc.
- Probémoslo.

debug y mvbutils

- Mark Bravington ha desarrollado dos paquetes, **mvbutils** y **debug**, que pueden ayudar mucho con debugging.
- Yo no tengo experiencia directa: no son fáciles de usar desde (X)Emacs, y los mecanismos que ofrecen **browser**, **debug**, y el uso de **cat** me han resultado suficientes.
- Los incluyo en los ficheros, por si quereis probarlos.

Lexical scope: introducción

```
> ff <- function(a, b, c = 4,
+   d = FALSE) {
+   x1 <- a * z
+   return(x1)
+ }
```

- "z" es una "free variable": ¿cómo se especifica su valor? **Lexical scoping**.
- Documentación: *Frames, environments, and scope in R and S-PLUS* de J. Fox (en <http://cran.r-project.org/doc/contrib/Fox-Companion/appendix.html>) y sección 10.7 en *An introduction to R*. También `demo(scoping)`. El que seguiré.
- *Lexical scope and statistical computing* de R. Gentleman y R. Ihaka, J. Comput Graph. Stats, 2000, 9: 491–508. Más detalles, y sus implicaciones generales en statistical computing.

- Ojo: las “scoping rules” son algo distintas entre R y S-PLUS. Nos limitaremos a R. Ver documentos mencionados para diferencias con S-PLUS.
- ¿Por qué importa? Porque hace la programación en R mucho más potente y sencilla.

Definiciones

- **Bound to**: una variable **is bound to** si se le ha asignado un valor. `x <- 5` o `f1(x = 2)`.
- **Frame**: un conjunto de bindings. En un frame dado, una variable sólo puede tener un valor asignado (ej., podemos hacer: `x <- 2`; `x <- 5`; `x` primero tiene el valor 2, y luego el 5, pero no ambos). En distintos frames, una variable puede tener distintos bindings (veremos ejemplos en un momento).
- **Free variable**: una variable que, en un frame dado, no está “bound to” a ningún valor. En `f3 <- function(x) x * y`; `f3(5)`, `y` es una variable libre en el frame local de la función `f3` (pero `x` tiene el valor 5).
- **global frame, global environment**: al arrancar R, si hacemos `x <- 13`, hemos asignado 13 a `x` en el “global frame” o “global environment”.

- **Scoping rules**: determinan como el intérprete (de R, o de cualquier lenguaje) busca el valor de las free variables.
- **environment**: una secuencia de frames. Ojo la idea de secuencia.
 - ▶ *frame 1* luego *frame 2*
 - ▶ *frame 2* luego *frame 1*
 - ▶ Pueden tener consecuencias MUY distintas. Si “x” está definido tanto en frame 1 como en frame 2, en el primero caso x en el frame 1 “shadows” (enmascara) el valor de x en el frame 2.
 - ▶ **enclosure**: función + environment.

Interludio: search, librerías, enmascaramientos

- Nótese que `search()` nos informa de como buscamos valores y funciones.
- Creemos un fichero “misFunciones.R” que contiene sólo `lm <- function(x, y) x * y`
- En R hacemos: `source('misFunciones.R')`.

```
> lm  
> rm(lm)  
> lm
```

- Habíamos enmascarado “lm” (pero nadie nos dijo nada!).

Introducción

GUIs

R: lenguaje

Import/Export

Gráficos en R

Progr. I

Definición de
funciones

Control de
ejecución

Cuando algo va
mal: browser y
debug

Ambito: lexical
scope

Try

Profiling

parsing
strings

Métodos y
técnicas
estadísticas
disponibles

Modelos
lineales

GLMs

Problema I

Problema II

Prog. II

```
> library(car)
> library(Hmisc)
> ls(pos = 2)
> rm(xless, pos = 2)
> ls(pos = 2)
```

Scoping rules en R

- El environment de una función. Ej., “f1”, definida dentro de “f1Padre”, dentro de “f1Abuelo”, ...
- *frame local de la función* luego *frame donde se definición función (f1Padre)* luego *f1Abuelo ... global environment.*

```
> f1 <- function(x) x + a
```

```
> a <- 15
```

```
> x <- 7
```

```
> f1(x = 1)
```

```
> f1(3)
```

```
> f2 <- function(x) {
```

```
+   a <- 3
```

```
+   f3(x)
```

```
+ }
```

```
> f3 <- function(y) y + a
```

```
> f2(2)
```

```
> f3(2)
```

Introducción

GUIs

R: lenguaje

Import/Export

Gráficos en R

Progr. I

Definición de
funciones

Control de
ejecución

Cuando algo va
mal: browser y
debug

Ámbito: lexical
scope

Try
Profiling

parsing
strings

Métodos y
técnicas
estadísticas
disponibles

Modelos
lineales

GLMs

Problema I

Problema II

Prog. II

```
> f4 <- function(x) {  
+   a <- 3  
+   f5 <- function(y) y + a  
+   f5(x)  
+ }  
> f4(2)
```

- Esta funcionalidad facilita muchísimo el escribir funciones dentro de otras funciones. Lo volveremos a ver luego.

Closures: definiendo funciones

- (Seguimos siguiendo a J. Fox).

```
> make.power <- function(power) {  
+   function(x) x^power  
+ }  
  
> square <- make.power(2)  
> cuberoot <- make.power(1/3)  
  
> square  
> cuberoot  
  
> square(2)  
> square(9)  
> cuberoot(64)
```

- Al hacer `square <- make.power(2)` el valor 2 es bound a la variable local “power”, y la asignación a “square” devuelve una “closure” (función + environment).

environments

- Verifiquemos qué es lo que “hay dentro de” square y cuberoot:

```
> environment(square)
> ls(envir = environment(square))
> get("power", envir = environment(square))
> get("power", envir = environment(cuberoot))
```

- square y cuberoot son, por tanto, la definición de la función y un conjunto de bindings (en este caso, el binding de power), que son distintos entre ambas funciones.

try

- Qué hacemos si algo falla?
- En R, un mecanismo sencillo de hacer frente a “excepciones” es con try.
- Varios ejemplos lo usarán luego. De momento:

```
> ffail <- function(x, y) {  
+   tf1 <- try(lm(y ~ x))  
+   if (class(tf1) == "try-error")  
+     print("Vaya, un error")  
+   else print(summary(tf1))  
+ }  
  
> x <- rnorm(10)  
> y <- rnorm(10)  
> ffail(x, y)  
  
> y <- 34  
> ffail(x, y)
```

Profiling: `unix.time` y `Rprof`

- Nuestro objetivo aquí no es producir las funciones más eficientes, sino funciones que hagan lo que deben.
- Las administraciones habituales sobre no optimizar y jamás optimizar antes de tiempo.
- Pero a veces útil saber cuanto dura la ejecución de una función: `unix.time(my.f2(runif(10000), rnorm(1000)))`. En vez de "unix.time" podemos usar "system.time".
- `Rprof`: un profiler para ver cuantas veces es llamada cada función y cuanto tiempo se usa en esa función. Ver la ayuda.
- Garbage collection: desde R 1.2.0 hay garbage collection. Se puede ver el status con `gc()`, que sirve también para despejar las cosas después de operaciones con manejo de grandes objetos.

Rprof

Progr. I

Definición de
funciones
Control de
ejecución
Cuando algo va
mal: browser y
debug
Ambito: lexical
scope
Try
Profiling

parsing strings

Métodos y técnicas estadísticas disponibles

Modelos lineales

GLMs

Problema I

Problema II

Prog. II

```
> fC <- function(z, rep = 10) {  
+   for (i in 1:rep) {  
+     x <- rnorm(2000)^2  
+     y <- mean(x) - runif(length(x))  
+     w <- sample(y, z)  
+   }  
+ }  
  
> Rprof()  
> fC(300, 500)  
> Rprof(NULL)  
> summaryRprof()
```

eval(parse(text=x))

- Un conjunto de objetos (generalmente nombres) con una raíz común que deseamos extraer, generalmente en bucles. Por ejemplo:

```
> sitio1 <- "Asturias"  
> sitio2 <- "Cantabria"  
> sitio3 <- "Galicia"  
> par(mfrow = c(1, 3))  
> for (i in 1:3) {  
+   x <- paste("sitio", i, sep = "")  
+   plot(rnorm(10))  
+   title(eval(parse(text = x)))  
+ }
```

- "x" podría ser el título de una figura, un fichero a leer (ej., capturas en cada sitio), etc.

- (En efecto, mucho más simple si todos los nombres existieran como los elementos de un vector o lista, pero en muchos casos no es así).
- Miremos la ayuda de **eval** y **parse**.

- (En efecto, mucho más simple si todos los nombres existieran como los elementos de un vector o lista, pero en muchos casos no es así).
- Miremos la ayuda de **eval** y **parse**.
- Y ahora dos ejemplos no de libro.

- `varSelRF.R`, líneas 49 a 83 (elimino mucho, para que quepa):

```
> basicClusterInit <- function(clusterNumberNodes = 1,  
+   nameCluster = "TheCluster",  
+   typeCluster = "MPI") {  
+   assign(nameCluster, makeCluster(clusterNumberNodes,  
+     type = typeCluster),  
+     env = .GlobalEnv)  
+   clusterSetupSPRNG(eval(parse(text = nameCluster)),  
+     seed = sprng.seed)  
+   clusterEvalQ(eval(parse(text = nameCluster)),  
+     library(randomForest))  
+   clusterEvalQ(eval(parse(text = nameCluster)),  
+     library(varSelRF))  
+ }
```

- Procede del código para el programa SignS (<http://signs.bioinfo.cnio.es>).
- Tenemos una matriz de datos cuyos nombres pueden ser cualquier cosa.

```
> library(survival)
> subject <- Surv(rexp(10), rbinom(10,
+      1, 0.7))
> mdf <- matrix(rnorm(50), ncol = 5)
> colnames(mdf) <- sapply(1:5,
+      function(x) paste(sample(letters,
+      5), collapse = ""))
> mdf <- as.data.frame(mdf)
```


- Queremos un análisis de supervivencia con todos los conjuntos de dos variables (la primera más cada una de las demás) y con variable labels que sean inteligibles.

```
> attach(mdf)
> for (i in 2:5) {
+   print(coxph(sobject ~ mdf[,
+               1] + mdf[, i]))
+ }
```

- No, eso no nos dice fácilmente que variables son las evaluadas. (Un ejemplo de "computing on the language").

```
> attach(mdf)
> for (i in 2:5) {
+   print(coxph(eval(parse(text = paste("sobject ~",
+   paste(colnames(mdf)[c(1,
+   i)], sep = "", collapse = " + "))))))
+ }
```

Métodos y técnicas estadísticas disponibles

Métodos y técnicas estadísticas disponibles

- Estadística descriptiva y tabulación.
- Contrastes de hipótesis (test de la t, wilcoxon, etc).
- Modelos y métodos multivariantes
 - ▶ PCA
 - ▶ Análisis factorial
 - ▶ MANOVA
 - ▶ Análisis discriminante
 - ▶ Análisis de componentes independientes
 - ▶ Clustering (aglomerados), incluyendo modelos de mezclas (mixture models).

- Modelos lineales y derivados
 - ▶ Modelos lineales
 - ▶ Modelos lineales generalizados
 - ▶ Modelos lineales multivariantes (MANOVA)
 - ▶ Modelos lineales mixtos
 - ▶ Modelos no-lineales
 - ▶ Modelos no-lineales mixtos
 - ▶ Modelos lineales generalizados mixtos (GLMM)
 - ▶ Modelos aditivos generalizados (GAM)

- Modelos para datos de supervivencia
 - ▶ Cox y modelos paramétricos
 - ▶ Modelos con frailties
- Modelos para datos categóricos (no solo GLM, sino también análisis de correspondencias, etc).

- Métodos de clasificación y “machine learning”
 - ▶ Redes neuronales.
 - ▶ Máquinas de vector soporte (SVM).
 - ▶ Árboles de clasificación y random forest.
 - ▶ Bagging, boosting.
- Paquetes y funcionalidad varia para “métodos bayesianos”.

Introducción

GUIs

R: lenguaje

Import/Export

Gráficos en R

Progr. I

parsing
strings

Métodos y
técnicas
estadísticas
disponibles

**Modelos
lineales**

Modelos lineales:
introducción

Regresión lineal y
regresión múltiple

Anova y otros
modelos

Diagnósticos

Selección de
variables

GLMs

Problema I

Problema II

Prog. II

Sweave et al.

Modelos lineales

Modelos lineales: introducción

Regresión lineal y regresión múltiple

Anova y otros modelos

Diagnósticos

Selección de variables

Modelos lineales: introducción

- Piedra angular de la estadística.
- Expresión general:

$$y_i = \sum_{j=1}^{j=p} x_{ij} \beta_j + e_i$$

- Generalmente (suponemos) $e_i \text{ Normal}(0, \sigma^2)$
- Modelo de regresión de una sola variable
 $y_i = \alpha + \beta x_i + e_i$ es un caso especial de la expresión anterior.
- También lo es la regresión múltiple.
- También lo es el ANOVA (análisis de la varianza).
- Otros, como los modelos lineales generalizados, se derivan del modelo lineal.

Modelos lineales y derivados

“Roadmap” en la fig. 57 de Kuhnert y Venables (p. 141).

- Modelos lineales
- Modelos lineales generalizados
- Modelos lineales multivariantes (MANOVA)
- Modelos lineales mixtos
- Modelos no-lineales
- Modelos no-lineales mixtos
- Modelos lineales generalizados mixtos (GLMM)
- Modelos aditivos generalizados (GAM)

Modelos lineales et al. en R

- Todos los incluidos arriba pueden usarse en R.
- Para los lineales y lineales generalizados hay, generalmente, más de un paquete y muchas opciones y funciones auxiliares.
- Usaremos, además de las funciones existentes en el paquete base de R, la librería **car** de J. Fox.
- Para otros problemas con modelos lineales, algunos modelos lineales generalizados y análisis de supervivencia, muy interesante también las librerías **Hmisc** y **Design**, de F. Harrell.
- Nuestra exposición seguirá muy de cerca la de J. Fox en “An R and S-Plus companion to applied regression”.
- Nuestra exposición es **MUY** rápida. Es sólo para abrir el apetito y ver el amplísimo rango de herramientas a nuestra disposición.

Introducción

GUIs

R: lenguaje

Import/Export

Gráficos en R

Progr. I

parsing
strings

Métodos y
técnicas
estadísticas
disponibles

Modelos
lineales

Modelos lineales:
introducción

**Regresión lineal y
regresión múltiple**

Anova y otros
modelos

Diagnósticos

Selección de
variables

GLMs

Problema I

Problema II

Prog. II

Sweave et al.

```
> library(car)
> data(Davis)
> names(Davis)
> plot(repwt, weight, data = Davis)
> plot(weight ~ repwt, data = Davis)
> mod.davis <- lm(weight ~ repwt,
+   data = Davis)
> attach(Davis)
> mod.davis2 <- lm(weight ~ repwt)
> summary(mod.davis)
> plot(weight ~ repwt)
> abline(mod.davis, col = "blue")
> abline(0, 1, col = "green",
+   lty = 2)
> identify(repwt, weight)
> detach(Davis)
```

Regresión múltiple

- Podemos expresar

$$y_i = \sum_{j=1}^{j=p} x_{ij} \beta_j + e_i$$

como

$$y_i = \alpha + \sum_{j=1}^{j=p} x_{ij} \beta_j + e_i$$

o, por ej.,

$$y_i = \alpha + \beta_1 x_{i1} + \beta_2 x_{i2} + \beta_3 x_{i3} + e_i$$

cuando tenemos tres variables independientes.

Introducción

GUIs

R: lenguaje

Import/Export

Gráficos en R

Progr. I

parsing
strings

Métodos y
técnicas
estadísticas
disponibles

Modelos
lineales

Modelos lineales:
introducción

**Regresión lineal y
regresión múltiple**

Anova y otros
modelos

Diagnósticos
Selección de
variables

GLMs

Problema I

Problema II

Prog. II

Sweave et al.

```
> data(Prestige)
> Prestige <- na.omit(Prestige)
> attach(Prestige)
> names(Prestige)
> par(mfrow = c(1, 3))
> plot(prestige ~ education)
> plot(prestige ~ income)
> plot(prestige ~ women)
> pairs(cbind(prestige, education,
+             income, women))
> mod.prestige <- lm(prestige ~
+                   education + income + women)
> summary(mod.prestige)
```

Introducción

GUIs

R: lenguaje

Import/Export

Gráficos en R

Progr. I

parsing
strings

Métodos y
técnicas
estadísticas
disponibles

Modelos
lineales

Modelos lineales:
introducción

**Regresión lineal y
regresión múltiple**

Anova y otros
modelos

Diagnósticos

Selección de
variables

GLMs

Problema I

Problema II

Prog. II

Sweave et al.

```
> anova(mod.prestige)
> Anova(mod.prestige)
> "?" (Anova)
```

(La función **Anova** es parte de **car**, no del sistema base.)

"Ancova"

- Vamos a introducir tanto variables continuas.

```
> mod.prestige2 <- lm(prestige ~
+     education + income + type)
> contrasts(type)
> mod.prestige3 <- lm(prestige ~
+     -1 + education + income +
+     type)
> coefs <- mod.prestige2$coeff
> coefs
> coefs[1] + coefs[4]
> coefs[1] + coefs[5]
```

- Otros tipos de contrastes disponibles (sum, helmert, poly).
- No entraremos en ello. Se pueden cambiar, por ej., con `options(contrasts= c('contr.helmert', 'contr.poly'))`, donde el primero se refiere a factores sin ordenar y el segundo a factores ordenados.

Tests en "Ancova"

```
> anova(mod.prestige2)
> Anova(mod.prestige2)
> mod.prestige4 <- lm(prestige ~
+   type + income + education)
> anova(mod.prestige4)
> Anova(mod.prestige4)
> Anova(mod.prestige3)
```


Interacciones

```
> mod.prestige5 <- lm(prestige ~  
+   education + income + type +  
+   income:type + education:type)  
> mod.prestige6 <- update(mod.prestige2,  
+   . ~ . + income:type + education:type)  
> mod.prestige7 <- lm(prestige ~  
+   education * type + income *  
+   type)  
> Anova(mod.prestige5)  
> Anova(mod.prestige6)  
> Anova(mod.prestige7)  
> summary(mod.prestige5)  
> summary(mod.prestige6)  
> summary(mod.prestige7)
```

- Principio de marginalidad.
- Y las sumas de cuadrado de Type III???

Anova y otros modelos

- Hemos visto el uso de " ", "+", ":", "*", "-1", para especificar modelos.
- Rango completo de modelos es más amplio; "/" para modelos enjados o anidados (nested).
- A veces se separan el Anova de la regresión. En realidad, ambos son casos específicos de modelos lineales. Anidamiento, factores fijos y aleatorios, split-plot designs, modelos de medidas repetidas, etc, se recogen más cómodamente en el contexto de los modelos lineales mixtos.
- No entramos aquí en más detalles.

Diagnósticos

- Las funciones en el paquete básico:

```
> par(mfrow = c(1, 1))
```

```
> plot(mod.prestige)
```

- Qué es cada cosa?
- Vamos con calma... uno a uno.

```

> "?"(influence.measures)
> hatvalues(mod.prestige)
> dfbeta(mod.prestige)
> dfbetas(mod.prestige)
> plot(cookd(mod.prestige))
> identify(1:length(cookd(mod.prestige)),
+         cookd(mod.prestige), row.names(Prestige))
> plot(rstudent(mod.prestige) ~
+      fitted.values(mod.prestige))
> abline(h = 0, lty = 2)

```

- El último, al usar studentized residuals, minimiza el problema de que los residuos no tienen la misma varianza.
- (cookd no es del paquete básico, sino de car).

Diagnósticos: car

- La librería car ofrece ciertas adaptaciones y adiciones de plots diagnósticos que son muy cómodas.

```
> qq.plot(mod.prestige, simulate = TRUE,  
+         labels = row.names(Prestige))  
> plot(hatvalues(mod.prestige))  
> identify(1:length(hatvalues(mod.prestige)),  
+         hatvalues(mod.prestige),  
+         row.names(Prestige))  
> plot(dfbetas(mod.prestige)[,  
+         c(2, 3)])
```

Added variable plots y ceres plots

- Added variable plots : Muy útiles para detectar leverage e influencia, minimizando el enmascaramiento.
- Ceres plots: detección de relaciones no lineales.

```
> av.plots(mod.prestige, labels = row.names(Prestige))  
> ceres.plots(mod.prestige, ask = TRUE)
```

VIF: variance inflation factors

- Y la multicolinealidad?

```
> vif(mod.prestige)
```

```
> vif(mod.prestige2)
```

- La $\sqrt{vif_j}$ indica cuanto se ensancha el intervalo de confianza para β_j en comparación con una situación en la que no hubiera predictores correlacionados.

Otros

- No hemos mencionado las transformaciones Box-Cox.
- Casi todos los métodos mencionados sirven para modelos lineales generalizados.
- No se puede enfatizar bastante la necesidad de mirar con mimo y cuidado los plots de diagnóstico!!

Selección de variables

- ¿Necesitamos todas esas variables en el modelo?
- La selección de variables es terreno **muy peligroso**.
- En general, después de la selección de variables, los p-valores no son de fiar (subestimados) y se subestima también la variabilidad en la obtención de soluciones.
- Frank Harrell tiene una detallada discusión de estos temas, y posibles formas de examinarlo (ej., via bootstrap del proceso de selección de variables).
- No entramos ahora en más detalles. Suponemos, en lo que sigue, que haremos selección por "motivos heurísticos".

- A veces popular el uso de métodos "stepwise" usando p-valores. En general, aun más desaconsejables.
- Usemos **stepAIC**.

```
> data(Ericksen)
> census.mod <- lm(undercount ~
+      ., data = Ericksen)
> library(MASS)
> summary(census.mod)
> stepAIC(census.mod)
> stepAIC(mod.prestige)
> stepAIC(mod.prestige2)
```
- Podemos hacer también "forward selection", empezando con un modelo con sólo el intercepto, o acotar los modelos a usar.
- Podemos obtener los subconjuntos óptimos de predictores para modelos de todos los tamaños (ver **regsubsets** en librería **leaps** y función **subsets** en car).

GLM: introducción

- Generalización modelo lineal.
- Distribución del error cambia.
- La relación entre la esperanza de la respuesta y el predictor lineal cambia.
- Lo que sigue igual: un predictor lineal, que es una combinación lineal de los variables independientes.
$$\sum_{j=1}^p x_{ij} \beta_j$$
- $\eta_i = \beta_1 x_{i1} + \beta_2 x_{i2} + \dots + \beta_p x_{ip}$
- "Link function": $g(\mu_i) = \eta_i$. $\mu_i = g^{-1}(\eta_i)$ (g^{-1} a veces llamada "mean function").

Introducción

GUIs

R: lenguaje

Import/Export

Gráficos en R

Progr. I

parsing
strings

Métodos y
técnicas
estadísticas
disponibles

Modelos
lineales

GLMs

Modelos lineales
generalizados:
introducción

Datos dicotómicos.
Regresión logística

Modelos de
Poisson

Problema I

Problema II

Prog. II

Sweave et al.

Docs.

- Datos normales: "identity link"; datos binomiales: "logit link"; Poisson: "log link".
- Los anteriores son los links comunes. Hay otros para esas distribuciones, así como links para otras distribuciones.
- Y la varianza? Para familias exponenciales, la varianza es una función de la media y de un parámetro de dispersión.
- Esta es otra introducción, **AUN MAS RAPIDA**. Seguimos libro de Fox y notas de Kuhnert y Venables.

Regresión logística

- $\mu = \frac{e^{\eta}}{1+e^{\eta}}, \eta = \log \frac{\mu}{1-\mu}$
- Usando R: distinguir entre datos binarios (0, 1) y datos binomiales.

Datos binarios

```
> data(Mroz)
> attach(Mroz)
> mroz.mod1 <- glm(lfp ~ k5 +
+       k618 + age + wc + hc + lwg +
+       inc, family = "binomial")
> summary(mroz.mod1)
> Anova(mroz.mod1)
> stepAIC(mroz.mod1)
> mroz.mod2 <- update(mroz.mod1,
+       . ~ . - k5)
> summary(mroz.mod2)
> anova(mroz.mod1, mroz.mod2,
+       test = "Chisq")
> Anova(mroz.mod1)
```

Introducción

GUIs

R: lenguaje

Import/Export

Gráficos en R

Progr. I

parsing
strings

Métodos y
técnicas
estadísticas
disponibles

Modelos
lineales

GLMs

Modelos lineales
generalizados:
introducción

Datos dicotómicos.
Regresión logística

Modelos de
Poisson

Problema I

Problema II

Prog. II

Sweave et al.

Docs.

```
> mroz.mod3 <- update(mroz.mod1,  
+ . ~ . - k618 - hc)  
> anova(mroz.mod1, mroz.mod3,  
+ test = "Chisq")  
> mroz.mod4 <- update(mroz.mod1,  
+ . ~ . - k618 - hc - k5)  
> anova(mroz.mod1, mroz.mod4,  
+ test = "Chisq")
```

Datos binomiales

- Creamos los "famosos" budworm data.

```
> ldose <- rep(0:5, 2)
> numdead <- c(1, 4, 9, 13, 18,
+             20, 0, 2, 6, 10, 12, 16)
> sex <- factor(rep(c("M", "F"),
+                  each = 6))
> SF <- cbind(numdead, numalive = 20 -
+             numdead)
> Budworms <- data.frame(ldose,
+                          sex)
> Budworms$SF <- SF
> rm(sex, ldose, SF)
```


Introducción

GUIs

R: lenguaje

Import/Export

Gráficos en R

Progr. I

parsing
strings

Métodos y
técnicas
estadísticas
disponibles

Modelos
lineales

GLMs

Modelos lineales
generalizados:
introducción

Datos dicotómicos.
Regresión logística

Modelos de
Poisson

Problema I

Problema II

Prog. II

Sweave et al.

Docs.

```
> bud.mod1 <- glm(SF ~ sex/ldose,  
+   family = "binomial", data = Budworms,  
+   trace = TRUE)  
> summary(bud.mod1, cor = FALSE)  
> bud.mod2 <- glm(SF ~ sex * ldose,  
+   family = "binomial", data = Budworms,  
+   trace = TRUE)  
> summary(bud.mod2, cor = FALSE)  
> anova(bud.mod1, bud.mod2)  
> bud.mod3 <- glm(SF ~ -1 + sex +  
+   ldose, family = "binomial",  
+   data = Budworms, trace = TRUE)  
> anova(bud.mod3, bud.mod1, test = "Chisq")
```

Poisson

- Dos usos fundamentales: para conteos y para analizar tablas de contingencia.
- Sólo mostraremos un ejemplo con conteos. (El examen de las tablas de contingencia requiere extenderse sobre modelos loglineares, etc).

Conteos

```
> data(Ornstein)
> summary(Ornstein)
> attach(Ornstein)
> plot(as.numeric(iltable) ~ as.numeric(names(iltable)),
+      type = "h", xlab = "Number of interlocks",
+      ylab = "Frecuencia")
> orns.mod1 <- glm(interlocks ~
+      assets + nation + sector,
+      family = "poisson")
> summary(orns.mod1)
> Anova(orns.mod1)
```

Otros

- Nos hemos dejado mucho en el tintero de los GLMs.
- Modelos para tablas de contingencia (via Poisson), modelos multinomiales (con `nnet: multinom`), modelos para respuestas ordenadas (proportional odds models), modelos para dicotomías anidadas; sobredispersión en modelos binomiales y de poisson.
- Diagnóstico: muchos de los vistos antes con modelos lineales se pueden usar con GLMs, pero la interpretación de los residuos a veces es complicada (existen varios tipos de residuos, que en los modelos gaussianos coinciden, pero no en los demás); particularmente complicado con datos binarios.
- Modelos de supervivencia, que pueden verse como un tipo de GLMs.

Problema I: introducción

- Supongamos una variable discreta (ej., sexo, o país de origen, o tipo de enfermedad, o subespecie) y una gran cantidad de variables continuas que pueden estar relacionadas con esa variable discreta.
- ¿Cuáles son las que se relacionan? ¿Cómo?
- En muchas áreas existe una lógica “retorcida” que hace lo siguiente:
- Hacemos un test (ej., test de la t si dos clases, Anova si 3 o más), variable a variable.
- Ordenamos las variables de “más importante” a “menos importante” en función de, por ej., el p-valor.
- Hacemos un cluster (dendrograma), usando sólo las variables “más importantes”.
- Afirmamos tener una muestra obvia de la relación cuando el dendrograma separa limpiamente los grupos.

Problema I: objetivo

- Desarrollar código que haga precisamente lo de arriba.
- Para simplificar, empecemos por sólo dos clases (ej., hombres vs. mujeres, o dos subespecies de pez).
- Mostremos como, con datos al azar (sobre todo si número de variables es grande) siempre encontramos resultados formidables.

Problema I: herramientas

- Test de la t.
- La función hclust.

Problema II: introducción

- Un escenario que no es distinto del anterior, pero ahora...
- Queremos usar una regresión logística.
- Hay muchas más variables que sujetos, o bien queremos modelos muy simples.
- Decidimos que sólo vamos a usar regresiones logísticas con un número pequeño de variables (ej., 2 a 4).
- Sabemos que es posible que no encontremos “El” modelo. Usaremos la lógica de la combinación de modelos. Usaremos un ponderado de modelos con AIC y stacking.
- Queremos evaluar el funcionamiento de esos modelos ponderados.

Problema II: objetivo y métodos

- Desarrollar un algoritmo de “stacking” de las predicciones de regresiones logísticas.
- Desarrollar un algoritmo para obtener modelos (y predicciones) ponderadas usando AIC.
- Evaluar el funcionamiento de estos métodos (ej., con validación cruzada).
- La programación será muy “defensiva”: el problema es complejo y muchas cosas pueden fallar, por lo que usaremos “try”, “browser”, y frecuentes asignaciones de objetos al entorno global.

Introducción

GUIs

R: lenguaje

Import/Export

Gráficos en R

Progr. I

parsing
strings

Métodos y
técnicas
estadísticas
disponibles

Modelos
lineales

GLMs

Problema I

Problema II

Prog. II

source y BATCH

Creación de
paquetes para uso
propio

Llamando a
funciones en C/C++

Crear paquetes
para R Windows
desde Linux

NAMESPACE y
"funciones ocultas"

Paralelización

Programación en R (II)

source y BATCH

Creación de paquetes para uso propio

Llamando a funciones en C/C++

Crear paquetes para R Windows desde Linux

NAMESPACE y "funciones ocultas"

Paralelización

source y BATCH

- Para la ejecución no interactiva de código.
- Con `source` abrimos una sesión de R y hacemos `> source("mi.fichero.con.codigo.R")`.
- Con BATCH: `% R CMD BATCH mi.fichero.con.codigo.R`.
- `source` es en ocasiones más útil porque informa inmediatamente de errores en el código. BATCH no informa, pero no requiere tener abierta una sesión (se puede correr en el background).
- Ver la ayuda: `R CMD BATCH -help`.
- Puede que necesitemos explícitos `print` statements o hacer `source(my.file.R, echo = TRUE)`.
- `sink` es el inverso de `source` (manda todo a un fichero).

Creación de paquetes para uso propio

- Si tenemos unos ficheros con funciones, etc, ¿cómo lo usamos repetidas veces?
- Podemos usar `"source"`.
- Llena el espacio global de funciones, etc.
- Alternativa: usar un paquete.

- Detalles sobre como escribir un paquete: en manual “Writing R extensions”.
- Si sólo queremos “paquete de andar por casa” basta con:
- Crear un directorio; ej. “MisFuncs”.
- Crear subdirectorio: **R** y poner ahí el código.
- Crear fichero DESCRIPTION (ej., copiar de otra librería).
- Crear usando R CMD build. (Herramientas adicionales necesarias; ver <http://www.murdoch-sutherland.com/Rtools/>).

Llamando a funciones en C/C++

- Se puede llamar con facilidad a código escrito en C/C++ (y en Fortran).
- Util si existen librerías con la funcionalidad deseada.
- Util si algo en nuestro código es muy lento y fácil de reescribir en C/C++.
- Las herramientas necesarias son muy fáciles de configurar en Unix/Linux (en Linux, son parte de muchas instalaciones por defecto).
- Requerimientos más complicados en Windows. Ver:
 - ▶ R Installation and Administration: Appendix F, "The Windows toolset".
 - ▶ R Installation and Administration: Ch. 3, "Building from source".
 - ▶ R for Windows FAQ, Ch. 8, "Building from source".
- Se puede hacer, pero no es trivial ni para uso casual. No lo cubriremos aquí.

Introducción

GUIs

R: lenguaje

Import/Export

Gráficos en R

Prog. I

parsing
strings

Métodos y
técnicas
estadísticas
disponibles

Modelos
lineales

GLMs

Problema I

Problema II

Prog. II

source y BATCH

Creación de
paquetes para uso
propio

Llamando a
funciones en C/C++

Crear paquetes
para R Windows
desde Linux

NAMESPACE y
"funciones ocultas"

Revolución

- Desde R dos maneras de llamarlo: ".C" y ".Call".
- Detalles en "Writing R extensions", Ch. 4, "System and foreign language interfaces".
- .C es mucho más fácil, y para muchos usos basta.
- .Call nos permite manipular objetos de R desde C. Más complicado.

Ejemplo con “.C”

- Usaré como ejemplo mi paquete geSignatures.
- Se puede bajar de <http://ligarto.org/rdiaz/Software/Software.html>
- Existe tanto en versiones para Windows como en la versión original para Unix/Linux.
- Incluidos ambos ficheros en “Example-files”.
- En Windows, instalamos usando el instalador habitual.
- En Windows, podríamos instalar usando el fichero tar.gz, compilando el código, si dispusiéramos de las herramientas necesarias. (No lo cubriremos aquí).

- Si descomprimos el fichero zip, vemos que existe el directorio “libs”, y en él el fichero “geSignatures.dll”.
- Los ficheros “dll” son ficheros de código que es “dynamically loaded” en windows. Un Windows Explorer, o similar, nos dirá que es una “Extensión de la aplicación”.
- (Si miramos las propiedades del fichero, en el Tab “Versión”, nos dice que fué compilado bajo R.)
- Los ficheros con extensión dll son los que usan las aplicaciones que utilizan código en C/C++ (o Fortran) llamado desde R.

- Otra información sobre que este fichero se utiliza lo vemos en el fichero "NAMESPACE":
`useDynLib(geSignatures):`
- nos dice que este paquete usa una librería llamada geSignatures (el fichero geSignatures.dll) que se carga dinámicamente.
- (Hay otras formas de hacer que eso ocurra. Esta es una de ellas).
- Si no estamos en una librería, sino desde la línea de comandos, es habitual usar `dyn.load` directamente.

```
> help(dyn.load)
```

Introducción

GUIs

R: lenguaje

Import/Export

Gráficos en R

Progr. I

parsing
strings

Métodos y
técnicas
estadísticas
disponibles

Modelos
lineales

GLMs

Problema I

Problema II

Prog. II

source y BATCH

Creación de
paquetes para uso
propio

**Llamando a
funciones en C/C++**

Crear paquetes
para R Windows
desde Linux

NAMESPACE y
"funciones ocultas"

Revolución

- Cualquier otro paquete que use código compilado compartirá estos aspectos:
- Uno o más ficheros con extensión dll
- Algún mecanismo que cargue ese código en los dll

.C: Linux/Unix

- El paquete original no contiene ningún fichero con extensión dll, ni directorio libs.
- Sin embargo, existe un directorio "src", que no existía en Windows.
- Los paquetes tar.gz tienen las fuentes. Si existe código compilado, no está listo para ser instalado.
- El proceso de instalación (R CMD INSTALL o similares) llamará al compilador apropiado (e.g., gcc) que compilará el fichero apropiado.
- La compilación generará ficheros .so; equivalente al .dll.
- En nuestro caso: `dlda.cpp`, que será compilado, y luego cargado (via `useDynLib` en `NAMESPACE`).

Introducción

GUIs

R: lenguaje

Import/Export

Gráficos en R

Progr. I

parsing
strings

Métodos y
técnicas
estadísticas
disponibles

Modelos
lineales

GLMs

Problema I

Problema II

Prog. II

source y BATCH

Creación de
paquetes para uso
propio

Llamando a
funciones en C/C++

Crear paquetes
para R Windows
desde Linux

NAMESPACE y
"funciones ocultas"

Revolución

- La librería para windows se obtuvo del fichero original tar.gz.
- Unicos cambios: generar dll (y borrar src).
- De nuevo: **.so** [Unix/Linux] equivalente al **.dll** [Windows].

Usando .C: el código de R

- Abrimos el fichero geSignatures_0.7-1.tar.gz y buscamos geSignatures.R (en el directorio R) o el geSignatures_0.7-1.zip y buscamos geSignatures (en el directorio R).
- Buscar la primera función: dlda (comentarios eliminados)

```
> dlda <- function(ls, cll, ts) {  
+   ls <- as.matrix(ls)  
+   ts <- as.matrix(ts)  
+   .C("dlda", as.double(ls),  
+       as.integer(cll), as.integer(as.vector(table(cll))),  
+       as.double(t(ts)), as.integer(max(cll) -  
+           min(cll) + 1), as.integer(length(cll)),  
+       as.integer(ncol(ls)),  
+       as.integer(nrow(ts)),  
+       predictions = as.integer(rep(-9,  
+           nrow(ts))), PACKAGE = "geSignatures")$predicti  
+ }
```

Introducción

GUIs

R: lenguaje

Import/Export

Gráficos en R

Progr. I

parsing
strings

Métodos y
técnicas
estadísticas
disponibles

Modelos
lineales

GLMs

Problema I

Problema II

Prog. II

source y BATCH

Creación de
paquetes para uso
propio

**Llamando a
funciones en C/C++**

Crear paquetes
para R Windows
desde Linux

NAMESPACE y
"funciones ocultas"

Revolución

- Es una llamada desde R a una función en C que usa la interfaz `.C (.C("dlda", ...))`.

Usando .C: el código en C/C++

- Del fichero tar.gz, al descomprimir, en el directorio src buscamos dlda.cpp.
- El uso de **extern "C"** es porque se trata de código en C++.
- La función empieza así:

```
void dlda(double *ls, int *c1l, int *nk,  
          double *ts, int *num_classes,  
          int *nsubjects_ls,  
          int *num_genes, int *nsubjects_ts,  
          int *predictions) {
```

- Es una función con una correspondencia 1-a-1 con los argumentos de la llamada desde R.
- La función es de tipo "void", o sea que no devuelve explícitamente nada; el resultado de retorno va en el vector (el array) "predictions".

Crear paquetes para R desde Linux

- Si sólo hay código en R simplemente necesario hacer un fichero zip a partir del paquete instalado en Linux.
- Si usamos código en C/C++ o Fortran...

Crear paquetes para R desde Linux

- Si sólo hay código en R simplemente necesario hacer un fichero zip a partir del paquete instalado en Linux.
- Si usamos código en C/C++ o Fortran...
- Podemos preferir desarrollar en Linux, pero hacer nuestros paquetes disponibles en Windows.
- Podemos usar Windows, con todo un entorno de desarrollo, y compilar nuestro paquete para Windows.
- También posible directamente desde Linux compilar para Windows. Varias opciones. Trivial si usamos funcionalidad provista por J. Yan y A. Rossini (<http://cran.r-project.org/doc/contrib/cross-build.pdf>).

Funciones ocultas y NAMESPACE

```
> library(geSignatures)
> search()
> ls(pos = 2)
```

- Dónde está dlda?

```
> geSignatures:::dlda
```

- Vayamos a ver el NAMESPACE que existe en la librería.
- Qué hay y qué no hay?
- Podemos hacer lo mismo en cualquier otro paquete que queráis.

Paralelización

- Muchas tareas se prestan a ser ejecutadas simultáneamente sobre varias CPUs.
- Ganancia de velocidad. En muchos problemas, y con pocas CPUs, el escalado puede ser (casi) lineal.
- Casos más simples “embarrassingly parallelizable”. Bootstrap, validación cruzada, muchas simulaciones idénticas excepto por el valor de algun(os) parametro(s), etc.
- Se necesita la infraestructura; en R: PVM o MPI.
- MPI un estándar más moderno que PVM, y con desarrollo muy activo. A menos que exista una razón en contra, probablemente mejor usar MPI que PVM en igualdad de condiciones.

MPI

- Hay varias implementaciones del estándar MPI.
- La más fácil de usar con R es LAM/MPI
<http://www.lam-mpi.org>. Difícil en Windows. Sí, parcialmente, con Cygwin. Ver <http://www.lam-mpi.org/faq/category12.php3#question2>. En Linux es trivial.
- OpenMP (<http://www.open-mpi.org>); sucesor de LAM/MPI. De momento no bajo Windows, pero lo hará pronto.
- MPICH (<http://wwwunix.mcs.anl.gov/mi/mpich>) sí funciona en Windows, pero su uso con R puede requerir ciertas adaptaciones (ver documentación de Rmpi).

Ejemplos: snow

- Tres paquetes fundamentalmente **Rmpi**, **papply**, **snow**.
- snow es un mecanismo general de acceder a funcionalidad para la paralelización (provista, por ej., por Rmpi, o por PVM).
- snow facilita mucho la tarea de paralelizar: sustituir los lapply por funciones equivalentes (**clusterApply**, **clusterApplyLB**).

snow (II)

- varSelRF contiene tanto versiones paralelizadas como no paralelizadas de varias funciones.
- línea 590 de varSelRF.R. Versión serie (no paralelizada): un bucle sobre bootnumber podríamos haber usado `sapply`).

```
> for (nboot in 1:bootnumber) {  
+   cat(".")  
+   boot.runs[[nboot]] <- bootTrainTest(nboot,  
+     c.sd = c.sd, mtryFactor = mtryFactor,  
+     ntree = ntree, ntreeIterat = ntreeIterat,  
+     whole.range = whole.range,  
+     recompute.var.imp = recompute.var.imp)  
+ }
```

- Donde `bootTrainTest` definida desde línea 492 a 557:

```
> bootTrainTest <- function(dummy,  
+   c.sd, mtryFactor, ntree,  
+   ntreeIterat, whole.range,  
+   recompute.var.imp, ...) {  
+ }
```

- Versión paralelizada: línea 572.

```
> boot.runs <- clusterApplyLB(TheCluster,  
+ 1:bootnumber, bootTrainTest,  
+ c.sd = c.sd, mtryFactor = mtryFactor,  
+ ntree = ntree, ntreeIterat = ntreeIterat,  
+ whole.range = whole.range,  
+ recompute.var.imp = recompute.var.imp)
```

- Llamaremos a `bootTrainTest` `bootnumber` número de veces, con balanceo de carga sobre las CPUs. Una especie de `sapply` (sobre “1:bootnumber”) a ejecutar sobre las CPUs.
- Usar la misma función básica con y sin paralelización facilita debugging y paralelización de funciones previamente seriales.
- (Hemos obviado muchos detalles sobre uso de `snow`).

Rmpi

- Acceso a las funciones de MPI desde R (send, recv, etc).
- Uso directo más complicado que snow.
- Debugging y captura de mensajes de error más complicada.
- Instrucciones para su uso en Windows en:
<http://www.stats.uwo.ca/faculty/yu/Rmpi>

- Lógica parecida a la anterior, aunque detalles varían.
- Definir una función, que exportamos a los nodos (via, ej., `mpi.bcast.Robj2slave`).
- Usar `mpi.remove.exec`.
- (x, y, z, ya pasados a cada elemento del universo MPI via `mpi.bcast`)

```
> flInternalMPI <- function() {  
+   tmp <- f2(x, y, z)  
+   return(list(scoresTest = tmp$testPred,  
+             fmObject = tmp$bestTrain))  
+ }  
  
resultados <- mpi.remote.exec(flInternalMPI())
```

papply

- Una versión de `apply` que se ejecuta paralelizadamente (con balanceo de carga) si es posible, y si no ejecuta una versión serial.
- Documentación y ejemplos en <http://ace.acadiau.ca/math/ACMMaC/software/papply/>.
- Quizás una de las formas más sencillas de proceder, con razonablemente claros “error messages”

- A y BB en cada nodo.

```
> funpap3 <- function(x) {  
+   tmp1 <- f11(x, A)  
+   tmp2 <- f12(x, BB)  
+   return(c(tmp1, tmp2$z))  
+ }  
  
tmp1 <- papply(as.list(1:nfold),  
funpap3,  
papply_commdata = list(A = esteA,  
BB = otraCosa))
```

Paralelización: resumen

- Uso de Rmpi, snow, y papply para tareas “embarrassingly parallelizable” no es difícil.
- Rmpi permite acceso a gran parte de MPI: mucho más que embarrassingly parallelizable.
- Debugging más (o mucho más) complicado en todos los casos (snow, Rmpi, papply).
- Oportunidades uso paralelización en aumento (dual-cores cada vez más frecuente; workstations con 2 dual cores cada vez más frecuentes).
- Linux posiblemente entorno más cómodo que Windows.
- No es trivial. Pero una vez cómodos, es fácil explotarlo en muchos casos.

Organización de análisis y programación

Notas generales

- Crucial mantener código comentado en ficheros: permite repetir análisis, automatizar acciones, entender secuencia de acciones.
- De gran ayuda si los comentarios contienen fechas.
- Más sofisticado el uso de sistemas de control de versiones (RCS, CVS, subversion), disponibles tanto en Linux/Unix, como Unix. (Por ejemplo, RCS es estándar en Emacs).
- Incorporar comentarios y código en el estilo “Literate Programming”...

Sweave

- Sweave (estándar con R) facilita el uso de un estilo tipo “Literate Programming”. Tenemos código y comentarios mezclados por completo.
- Ahora mismo, Sweave usa \LaTeX (pero podría usar otros lenguajes).
- Toda esta presentación escrita usando Sweave.
- Sweave especialmente útil para “análisis reproducibles”, elaboración de informes con datos cambiantes (pero análisis similares), etc.

Sweave y otros

- Sweave sólo necesita una instalación funcional de \LaTeX .
- Estándar en Linux (tetex). En Windows, muy recomendable MikTeX.
- Otros interesantes:
 - ▶ Usar la funcionalidad de “Projects” y “Groups” en Tinn-R.
 - ▶ Usar “Leo: literate editor with outlines” (<http://webpages.charter.net/edreamleo/front.html>).
- Podemos ver ejemplos de ambos (en realidad, esta presentación usa Sweave Y Leo).

Documentación sobre R (I)

Los "manuales" de R, incluidos en todas las instalaciones.
Son:

- *An introduction to R.* De lectura requerida.
- *Writing R extensions.* Creación de paquetes, interfaz con C.
- *R data import/export.*
- *The R language definition.*
- *R installation and administration.*

Documentación sobre R (II)

Documentación general:

- Ver, como lugar de entrada, <http://cran.r-project.org/other-docs.html>.
- *A guide for the unwilling S user*, de P. Burns. En http://cran.r-project.org/doc/contrib/Burns-unwilling_S.pdf o <http://www.burns-stat.com/pages/tutorials.html>. Sólo 8 páginas.
- *R para principiantes*, de E. Paradis. En <http://cran.r-project.org/other-docs.html> o http://cran.r-project.org/doc/contrib/rdebuts_es.pdf.
- *An introduction to R: software for statistical modelling and computing*, de P. Kuhnert y W.. Venables, en http://cran.r-project.org/doc/contrib/Kuhnert+Venables-R_course_Notes.zip.
- *FAQ*.

Documentación general:

- *S Programming*, de W. Venables y B. Ripley. (Ver también <http://www.stats.ox.ac.uk/pub/MASS3/Sprog.>)
- *S poetry* de P. Burns. En <http://www.burns-stat.com/pages/spoetry.html>.
- Otros documentos en la página de J. Fox (<http://cran.r-project.org/doc/contrib/Fox-Companion/appendix.html>), ej. sobre Frames, etc).
- El site de Paul Johnson (<http://lark.cc.ukans.edu/~pauljohn/R/statsRus.html>).
- Los libros azul , blanco, y verde de Chambers et al.

Documentación sobre R (III)

- Muchos libros disponibles; ver: <http://www.r-project.org/doc/bib/R-books.html>.
Lo que sigue una selección sesgada (otros nuevos aun no los conozco).
- *Introductory statistics with R* de P. Dalgaard.
- *An R and S-PLUS companion to applied regression*, de J. Fox.
- *Modern applied statistics with S, 4th ed.* de W. Venables y B. Ripley. (Ver también <http://www.stats.ox.ac.uk/pub/MASS4>.)
- Otros documentos en <http://cran.r-project.org/other-docs.html>.
- *S-PLUS 6.0 for Unix. Guide to statistics*. Vol. I & II. En <http://www.insightful.com/support/documentation.asp?DID=3>.

Documentación sobre R (IV)

- *Data analysis and graphics using R*, de J. Maindonald y J. Braum.
- *Linear models with R*, de J. Faraway.(Ver <http://www.stat.lsa.umich.edu/~faraway/book/> para scripts y pdf de versión previa).
- *Mixed-effects models in S and S-PLUS*, de J. Pinheiro y D. Bates.
- *Regression modelling strategies*, de F. Harrell.
- Site con documentación sobre análisis para datos categóricos (site para libro de A. Agresti *Categorical data analysis*).
<http://www.stat.ufl.edu/~aa/cda/cda.html>.
- *Modeling survival data: extending the Cox model*, de T. M. Therneau y P. M. Grambsch.
- Documentos misceláneos en página de J. Fox.
(<http://cran.r-project.org/doc/contrib/Fox-Companion/appendix.html>.)

R, ESS, XEmacs, (a la J. Fox)

- Toda la información está en <http://socserv.mcmaster.ca/jfox/Books/Companion/ESS/>
- Las últimas páginas de *An Introduction to ESS + XEmacs for Windows Users of R* tiene una **clarísima y detallada** información de como instalar y configurar ESS y XEmacs con las modificaciones de J. Fox. En resumen:
- Descargar XEmacs para Windows (la versión binaria para Windows, no para Cygwin) de <http://www.xemacs.org/Download/win32>. Si es posible, instalar todos los paquetes seleccionados por defecto (si no, ver documento J. Fox "An introduction to ESS (...)") para paquetes requeridos.)
- Instalar R (si no está ya instalado).

- Añadir `c:/Archivos de Programa/R/rwxxxx/bin` al search path the windows (o modificar `init.el` para que XEmacs sepa donde encontrar `rterm.exe`).
- Descargar el fichero "fox-ess-config.zip" de <http://www.socsi.mcmaster.ca/jfox/Books/Companion/ESS/>. Descomprimir.
- Crear un directorio `.xemacs` en el "home directory".
- Copiar "init.el" a `./.xemacs`.
- Copiar los demás ficheros de "fox-ess-.." a `c:/Archivos de Programa/XEmacs/XEmacs-x.y.z/etc/toolbar`.
- Listo. Ya se puede ejecutar XEmacs + ESS + R.

Modificaciones (XEmacs + ESS) (I)

- Muy conveniente mantener cada proyecto separado. Separamos código, datos, etc. Para eso:
 - ▶ Abrir en el Explorador de Windows el/los directorios donde queremos trabajar.
 - ▶ Right-click en el icono de XEmacs, y arrastrar hasta el directorio.
 - ▶ Seleccionar "crear iconos de acceso directo aquí".
- El icono de cada directorio inicia R en ese directorio.
- (Si nos olvidamos, siempre podemos cambiar directorios con "getwd").
- También podemos editar `init.el` y donde pone
`(setq ess-ask-for-ess-directory nil)` poner
`(setq ess-ask-for-ess-directory t)`.

Modificaciones (XEmacs + ESS) (II)

- Si preferimos los keybindings de XEmacs: cambiar la línea "(defconst pc-behaviour-level 2)" a "(defconst pc-behaviour-level 0)" (Y algún otro cambio menor).
- Para que indique en que fila y columna nos encontramos usar el "custom.el" provisto.
- Para lo primero, copiar ramon.init.el o fox.init.el como init.el.
- Para lo segundo: ya instalado por defecto en vuestras máquinas.
- Mis modificaciones están en mi página web (<http://bioinfo.cnio.es/~rdiaz/#cursosR>).

Modificaciones (XEmacs + ESS) (III)

- Podríamos querer usar XEmacs y ESS de la forma "tradicional", con el estilo genuino de XEmacs.
- Toda la funcionalidad de XEmacs (que se ve disminuida un poco por los keybindings tipo Windows al no permitir "C-x" como inicio de secuencia de teclas).
- Funciona si nos movemos a otro sistema operativo/otra máquina.

- Algunas limitaciones usando "modo windows":
 - ▶ No podemos usar shortcuts para evaluar línea, región o buffer (C-c C-j, C-c C-r, C-c C-b).
 - ▶ No podemos usar shortcuts para abrir ficheros (C-x C-f), para guardarlos (C-x C-s), para dividir la pantalla en 2 (C-x 2, C-x 3), para seleccionar el buffer que estamos editando (C-x b), o para cambiar de ventana (C-x o).
 - ▶ Limita un uso completo y sofisticado, tanto de ESS como de (X)Emacs.
 - ▶ ESS es un sistema grande y complejo con muchas opciones. No se aprende en 2 días. (¡(X)Emacs mucho menos!).

Si preferimos los keybindings de XEmacs: cambiar la linea "(defconst pc-behaviour-level 2)" a "(defconst pc-behaviour-level 0)" (Y algún otro cambio menor).

- Para que indique en que fila y columna nos encontramos usar el "custom.el" provisto.
- Para lo primero, copiar ramon.init.el o fox.init.el como init.el.
- Para lo segundo: ya instalado por defecto en vuestras máquinas.
- Mis modificaciones están en mi página web (<http://bioinfo.cnio.es/~rdiaz/#cursosR>).

R, ESS + XEmacs bajo GNU/Linux

- Obtener XEmacs y ESS.
- En algunas distribuciones, ESS precompilado (ej. Debian). En otras hay que instalar desde fichero tar.gz (<http://software.biostat.washington.edu/statsoft/ess/>).
- Si se quiere replicar comportamiento bajo windows:
 - ▶ Crear .xemacs en home.
 - ▶ Obtener xemacs-files.tar.gz de mi página y descomprimir.
 - ▶ Copiar RDU.Linux.like.Windows.init.el a `/.xemacs/init.el`.
 - ▶ Copiar también custom.el.
 - ▶ Poner iconos (xpm) en su sitio (ej., `/usr/lib/xemacs-x.y.z/etc/toolbar.`)

R, ESS + XEmacs bajo GNU/Linux (II)

Si usamos el método "como en Windows", para **mantener los proyectos diferenciados** necesitaremos iniciar xemacs desde directorios distintos.

O editar el fichero init.el, y donde pone `(setq ess-ask-for-ess-directory nil)` poner `(setq ess-ask-for-ess-directory t)`.

Esto difiere del comportamiento habitual si no usamos el Linux.like.Windows.

Uso básico de ESS + XEmacs (I)

Tomado de *An introduction to ESS + XEmacs for Windows users of R* de J. Fox.

- **Frame:** la ventana de Xemacs.
- **Menu bar, toolbar:** pues eso.
- **Window:** cada una de las ventanas.
- **Minibuffer:** muestra mensajes, se introducen comandos (ej., "C-s", o seleccionar "Edit/Find" en Menubar).
- **Buffer:** lo que Xemacs está editando. Dos importantes:
 - ▶ **Inferior R process:** el buffer cuyo **buffer name** es **"*R*"**.
 - ▶ **"*scratch buffer*":** donde introduciremos comandos y código para R. Lo renombraremos **lo-que-sea.R**.
- Podemos tener muchos otros buffers (ej., podemos editar código HTML, o C++, o Python, o Perl, o \LaTeX).

Uso básico de ESS + XEmacs (II)

- **Mode line:** información sobre el buffer que está justo encima. Información sobre **buffer name**, **major mode**, **font**, **column and row**. (Ojo: la numeración de la columna empieza por 0 por defecto).
- El buffer que estamos editando es lo que está en memoria, no es el fichero mismo. Los cambios sólo se guardan si salvamos el buffer.
- Se crean ficheros "nombre~ " que son copias del anterior estado del buffer. También se puede seleccionar para que Xemacs haga copias periódicas (útil en caso de crash.)
- La **extensión** de un fichero ayuda a (X)Emacs a elegir el modo de edición apropiado. Para nuestros ficheros con código en R usaremos **".R"** (o ".r" o ".s" o ".S"). Pero, ¿qué pasa si abrimos (o creamos) un fichero con extensión Py, cpp, tex, html?

Uso básico de ESS + XEmacs (III)

- Iniciamos XEmacs en el directorio de nuestro interés.
- Abrimos o creamos un fichero con código en R.
- El fichero de configuración de J. Fox reserva el buffer inferior para R, así que otros buffers sólo por arriba.
- Escribimos algo de código en R...

(Cosas a probar)

- Edición de código.
- Envío de una línea.
- Envío de una región marcada.
- Envío de un buffer entero.
- Guardar una figura como Metafile y pegarla en Word.
- Completar paréntesis.
- Uso de "STOP" para detener un cálculo.
- Trabajar directamente en ***R***, editando comandos, completando comandos, etc.
- Podemos instalar paquetes desde "ESS/R" (tenemos que estar en un buffer con código R).
- Salir de R. Es preferible hacer explícito la salida de R via "q()" (en vez de usar el menú de "File/ExitR").

Shortcuts útiles en ESS y XEmacs

- Los siguientes funcionan tanto con como sin las modificaciones de J. Fox.
- "M-x comment-region" ("M" es la "Meta key" que corresponde a "Alt").
- "C-s": incremental forward search.
- "C-r": incremental backward search.
- "M-Shift-5": query replace.
- "M-g": Ir a linea.