R - Some Loose Ends

Downloading packages and bundles

```r
> install.packages("rpart")
> library(rpart)
> data(kyphosis)

> kyphosis
         Kyphosis Age Number Start
1      absent    71     3     5
2      absent   158     3   14
3     present   128     4     5
4      absent     2     5     1
5      absent     1     4   15
6      absent     1     2   16
7      absent    61     2   17
8      absent    37     3   16
9      absent   113     2   16
10     present    59     6   12
11     present    82     5   14
12      absent   148     3   16
13      absent    18     5     2
...
```
> install.packages("VR")
> library(MASS)
> data(hills)
> ?hills

hills package:MASS R Documentation

Record Times in Scottish Hill Races

Description:
   The record times in 1984 for 35 Scottish hill races.

Usage:
   data(hills)

Format:
   The components are:
   ‘dist’ distance in miles (on the map)
   ‘climb’ total height gained during the route, in feet.
   ‘time’ record time in minutes.

Color and LaTeX symbols

One of the nice additions to R (relative to Splus) is the easy inclusion of mathematical expressions in plots using the function \texttt{expression}. Take a look at \texttt{help(plotmath)} to see a big list of what you can do; also look at the examples in the help file for the function legend.

x <- rnorm(100)
y <- x+rnorm(100)
plot(x,y,
   xlab=expression(hat(mu)[0]),
ylab=expression(alpha^{beta}),
   main=expression(paste("Plot of ",alpha^{beta}," versus ",hat(mu)[0])))
lines(lowess(x,y),col="green",lwd=2)
Another advantage of R compared to Splus is the much simpler control of colors. There are plenty of pre-defined colors:

```r
> str(colors())
chr [1:657] "white" "aliceblue" "antiquewhite" "antiquewhite1" ...
```

You can define any color you like using the function `rgb`. Check out the help file.

```r
> rgb(0.5,0.2,0.9)
[1] "#8033E6"
```

```r
> col2rgb("aquamarine")
[,1]
red 127
green 255
blue 212
```
options

Options Settings

Description:
Allow the user to set and examine a variety of global "options"
which affect the way in which R computes and displays its results.

Usage:
options(...)
getOption(x)

Arguments:
...: any options can be defined, using 'name = value'.
   However, only the ones below are used in "base R".
   Further, 'options('name') == options()['name']', see the example.
   x: a character string holding an option name.

par

Set or Query Graphical Parameters

Description:
'par' can be used to set or query graphical parameters. Parameters can be set by specifying them as arguments to 'par' in 'tag = value' form, or by passing them as a list of tagged values.

Usage:
par(..., no.readonly = FALSE)
<highlevel plot> (...) , <tag> = <value>)

Arguments:
...: arguments in 'tag = value' form, or a list of tagged values.
The tags must come from the graphical parameters described below.
nо.readonly: logical; if 'TRUE' and there are no other arguments, only parameters are returned which can be set by a subsequent 'par()' call.
Sweep out Array Summaries

Description:
Return an array obtained from an input array by sweeping out a summary statistic.

Usage:
sweep(x, MARGIN, STATS, FUN="-", ...)

Arguments:
x: an array.
MARGIN: a vector of indices giving the extents of ’x’ which correspond to ’STATS’.
STATS: the summary statistic which is to be swept out.
FUN: the function to be used to carry out the sweep.
...: optional arguments to ’FUN’.

> data(attitude)
> attitude
        rating complaints privileges learning raises critical advance
   1       43        51       30       39       61       92       45
   2       63        64       51       54       63       73       47
   3       71        70       68       69       76       86       48
   4       61        63       45       47       54       84       35
   5       81        78       56       66       71       83       47
...

# subtract the column medians
> med.att <- apply(attitude, 2, median)
> sweep(data.matrix(attitude), 2, med.att)
        rating complaints privileges learning raises critical advance
   1   -22.5      -14     -21.5    -17.5      -2.5     14.5       4
   2   -2.5       -1      -0.5     -2.5      -0.5     -4.5       6
   3    5.5        5     16.5     12.5     12.5       8.5       7
   4   -4.5       -2     -6.5     -9.5     -9.5       6.5      -6
   5   15.5       13       4.5      9.5       7.5      5.5       6
...
**match**

`match(x, table, nomatch = NA, incomparables = FALSE)`

**Arguments:**
- `x`: the values to be matched.
- `table`: the values to be matched against.
- `nomatch`: the value to be returned in the case when no match is found. Note that it is coerced to 'integer'.

**Usage:**
- `match(x, table, nomatch = NA, incomparables = FALSE)
- `x %in% table`

**Value Matching**

Description:
- ’match’ returns a vector of the positions of (first) matches of its first argument in its second. ’%in%’ is a more intuitive interface as a binary operator, which returns a logical vector indicating if there is a match or not for its left operand.

**Arguments:**
- `x`: the values to be matched.
- `table`: the values to be matched against.
- `nomatch`: the value to be returned in the case when no match is found. Note that it is coerced to ‘integer’.

---

**paste**

`paste(..., sep = " ", collapse = NULL)`

**Arguments:**
- `...`: one or more R objects, to be coerced to character vectors.
- `sep`: a character string to separate the terms.
- `collapse`: an optional character string to separate the results.

**Concatenate Strings**

Description:
- Concatenate vectors after converting to character.

Usage:
- `paste(..., sep = " ", collapse = NULL)`
source

source package:base R Documentation

Read R Code from a File or a Connection

Description:

‘source’ causes R to accept its input from the named file (the name must be quoted). Input is read from that file until the end of the file is reached. ‘parse’ is used to scan the expressions in, they are then evaluated sequentially in the chosen environment.

Usage:

source(file, local = FALSE, echo = verbose, print.eval = echo, verbose = getOption("verbose"), prompt.echo = getOption("prompt"), max.deparse.length = 150, chdir = FALSE)

Arguments:

file: a connection or a character string giving the name of the file or URL to read from.

You can use source to read in R code:

source("somecode.R")

The commands in somecode.R will be executed, and the objects specified will be created in your current .RData file. For example, this is quite convenient when you want to create a fairly fancy plot, and still need to tinker with the layout. You can also use source to read in functions you wrote:

myfunction <- source("myfunction.R")

The file you read in can contain more than one function statement. For example, if your main function calls some subfunctions, they can all be included in myfunction.R, and read in at the same time.
The BATCH mode

The command `source` reads in R code, but it is generally not a good idea to use it to run large simulations. This should be done using the `R BATCH` mode:

```
R BATCH inputfile outputfile &
```

In R, you have to use the `&` to run the job in the background. In Splus, this was not necessary.

If you run your job(s) on a cluster and share CPU time with other users, don’t forget to `nice` it (them)!

→ See Rafa’s notes from week one.

Using sink

```r
make.tex <- function(xx, fl="output.tex", nms=FALSE, r.nms=FALSE) {
    file <- fl
    sink(file)
    n1 <- dim(xx)[1]; n2 <- dim(xx)[2]
    if(nms){
        if(r.nms) cat(" & ")
        for(k in 1:n2){
            cat(names(xx)[k])
            if(k<n2) cat(" & ")
                else cat(" \rightarrow\n")
        }
    for(j in 1:n1){
        if(r.nms){
            cat(row.names(xx)[j])
            cat(" & ")
        for(k in 1:n2){
                cat(xx[j,k])
                if(k<n2) cat(" & ")
                    else cat(" \rightarrow\n")
        }
    sink()
    invisible()}
```
> source("make.tex.R")
> data(iris)
> make.tex(iris,nms=T,r.nms=T)

This creates a file output.tex that looks like this:

\begin{tabular}{cccccc}
1 & 5.1 & 3.5 & 1.4 & 0.2 & 1 \\
2 & 4.9 & 3 & 1.4 & 0.2 & 1 \\
3 & 4.7 & 3.2 & 1.3 & 0.2 & 1 \\
4 & 4.6 & 3.1 & 1.5 & 0.2 & 1 \\
5 & 5 & 3.6 & 1.4 & 0.2 & 1 \\
6 & 5.4 & 3.9 & 1.7 & 0.4 & 1 \\
... 
\end{tabular}

which can be included in the tabular environment of a LaTeX document using:

\input{output}

### Grouped data

The file fermentation.txt contains data on the effect of oxygen levels on the fermentation end product.

ethanol oxygen sugar
0.59 0 Galactose
0.30 0 Galactose
0.25 0 Glucose
0.03 0 Glucose
0.44 46 Galactose
0.18 46 Galactose
...

You can create a grouped data object:

fermentation <- groupedData(ethanol~oxygen|sugar, data=read.table("fermentation.txt",header=T), labels=list(x="Oxygen",y="Ethanol"))
The function `system` allows you execute Unix/Linux commands inside an R session. For example,

```r
> system("ls")
> system("pwd")
> system("ping www.google.com")
```

lists the files in your current directory, shows you the path, and 'pings' Google.

The function `system.time` (see also `proc.time`) lets you determines how much time the currently running R process has already consumed.
Debugging

R has some built-in program debugging tools. Check out the help files for:

- `browser`
- `debug`
- `trace`
- `traceback`
- `recover`

Customizing your R environment

Creating a `.Renviron` file in your root directory is a good idea. For example, when you use the `postscript` command to create a figure, the default for the paper format is ’A4’. You can change it to format ’letter’ by typing

```r
postscript("test.ps",paper="letter")
```

but you had to do this every time you want to create a postscript file. However, if you put the line

```r
R_PAPERSIZE=letter
```

into your `.Renviron` file, R will use the ’letter’ format as default. And it works in all subdirectories!
Creating a `.Rprofile` file in your root directory is an equally good idea. For example, the commands

```r
source("/home/ingo/code/R/image.foof.R")
source("/home/ingo/code/R/make.tex.R")
library(rpart)
```

ensure that the functions `image.foof` and `make.tex` are available when you start your R session, and that the library `rpart` is loaded. Works in all subdirectories!