members and contributors to the R Foundation, and recent news from the Bioconductor project.

Finally, I would like to remind everyone that the next DSC conference is taking place in Auckland, New Zealand, next February 15 & 16. I hope to see you here!

Paul Murrell
The University of Auckland, New Zealand
paul@stat.auckland.ac.nz

Sweave and the Open Document Format – The odfWeave Package

by Max Kuhn

Introduction

When documenting the results of a data analysis, creating the report can easily take more time than the analysis. There are at least two approaches to this

- Create and format tables and plots, then manually add text around these elements. In this case, the analysis code has no connection to the report.
- Take a combined approach, where the analysis results and the report are created at the same time using the same source code.

In the latter case, the `Sweave` function (Leisch, 2002) is a powerful component of R that can be used to combine R code with LaTeX so that the output is embedded in the processed document.

The user could write a document in LaTeX that contains `Sweave` tags. These tags encapsulate R commands. For example, an in–line `Sweave` tag might look like `\Sexpr{sqrt(2)}`. When the LaTeX document is processed by R's `Sweave` function, the `\Sexpr` tag is a signal to R to insert the results of the command `sqrt(2)` in place of the tag.

When "weaving" with LaTeX, the user has the ability to embed textual R output, as well as more complex types of R output (such as tables). After weaving, standard LaTeX tools can be used to generate a final document file, usually in postscript or PDF format.

For example, R package vignettes are created using LaTeX and `Sweave` (Leisch, 2003). For those readers who are new to `Sweave`, typing `vignette()` at the R prompt will display a list of installed packages that contain manuals that were written using LaTeX and `Sweave`. The package sources contain the underlying `Sweave`/LaTeX files and are a good resource for getting started with `Sweave` and LaTeX.

The capabilities of `Sweave` were later extended to HTML format in the **R2HTML** package. This provided the same functionality, but uses the HTML markup language.

While `Sweave` is highly effective at producing beautiful documents that show the many features of R, there are at least two limitations created by using LaTeX or HTML as the markup language. First, it requires the user to have a moderate knowledge of the markup languages to create and produce the documents.

Secondly, using these particular markup languages limits the main file formats to Postscript, PDF or HTML. If the user is collaborating with scientists or researchers who are not comfortable with editing LaTeX or HTML, it is difficult for them to add their subject-specific insight to the document. For example, at Pfizer, R is used as a computational engine for gene expression and computational chemistry data analysis platforms. Scientists can upload their data, specify models and generate results. We would like to provide them with the results of the analysis in a format that enables them to frame the statistical output in the context of their problem

The Open Document Format and The `odfWeave` Package

The Open Document Format (ODF) is an XML–based markup language. ODF is an open, non–proprietary format that encompasses text documents, presentations and spreadsheets. Version 1.0 of the specification was finalized in May of 2005 (OASIS, 2005). One year later, the format was approved for release as an ISO and IEC International Standard.

ODF has the advantage of being the default format for the OpenOffice (version 2.0 and above). OpenOffice is is a free, open source office suite and can export files to many different formats, including MS Word, rich text format, HTML, PDF and others. Other applications, such as Koffice, use the Open Document Format. Also, Microsoft has recently indicated that it will host an open source project converting documents to ODF from within Microsoft Office.

If OpenOffice documents were to support `Sweave` tags, reports could be edited and formatted in a sophisticated GUI environment. This would allow R users to include `Sweave` tags without directly editing the underlying markup. This would eliminate the two previously discussed limitations related to using LaTeX and HTML.

The **odfWeave** package was created so that the functionality of `Sweave` can used to generate documents in the Open Document Format. The package is currently limited to creating text documents using OpenOffice. Processing of presentations and spreadsheets should be considered to be experimental (but should be supported in subsequent versions of **odfWeave**).

Getting Started with `odfWeave`

First, install the most recent version of **odfWeave** using

```
install.packages("odfWeave")
```

at the R command prompt. Once the package is installed, there is a sub-directory called 'examples' that contains several prototype text documents files (with the extension ODT).

Second, since ODF files are compressed archives of files and directories, the user may need to download tools to extract and re-package the files. By default, `odfWeave` uses the `zip` and `unzip` tools which are free and can be found at `http://www.info-zip.org/`. Other applications, such as `jar`, can also be used. See the manual for the `odfWeaveControl` function for more information on using other utility programs.

`odfWeave` uses almost the exact same syntax as `Sweave` via LaTeX. Examples of common `Sweave` tags are illustrated in the next few sections. Once an ODF file is created containing `Sweave` tags, the document can be processed in R using:

```
odfWeave(inFile, outFile, workDir, control)
```

where `inFile` and `outFile` are the source and destination file names, `workDir` is an optional path where the files are processed and `control` is an optional object that can be used to specify how `odfWeave` runs. While the functionality of `odfWeave` is described in more detail in the next three sections, Figures 1 and 2 have screenshots of ODT files from OpenOffice before and after `Sweave` processing.

In–line Tags

The simplest way to start working with `Sweave` is to use in-line tags. These tags can be used to write single lines of text into the document. R code can be encapsulated in the tag `\Sexpr{}`. For example, putting the line

```
   There were data collected on
   \Sexpr{length(levels(iris$Species))}
   iris species.
```

in a document would produce the output:

```
   There were data collected on 3 iris species.
```

One interesting feature of using `odfWeave` is the preservation of the text formatting. If all the characters of an in-line `Sweave` tag have the same format, the formatting carries over into the results. Figures 1 and 2 show an example of formatting that persists. Note that partial formatting of the tag may not affect the output or might generate an error.

Code Chunks

Code chunks are used in much the same way as `Sweave` with LaTeX. Once exception is that code chunks must use the `<<>>=` convention (i.e `\begin{Scode}` is not currently supported).

For example, several R packages produce output on initial startup. To avoid sending this information into the document, a code chunk similar to

```
<<loadLibs, results = hide, echo = FALSE>>=
library("randomForest")
@
```

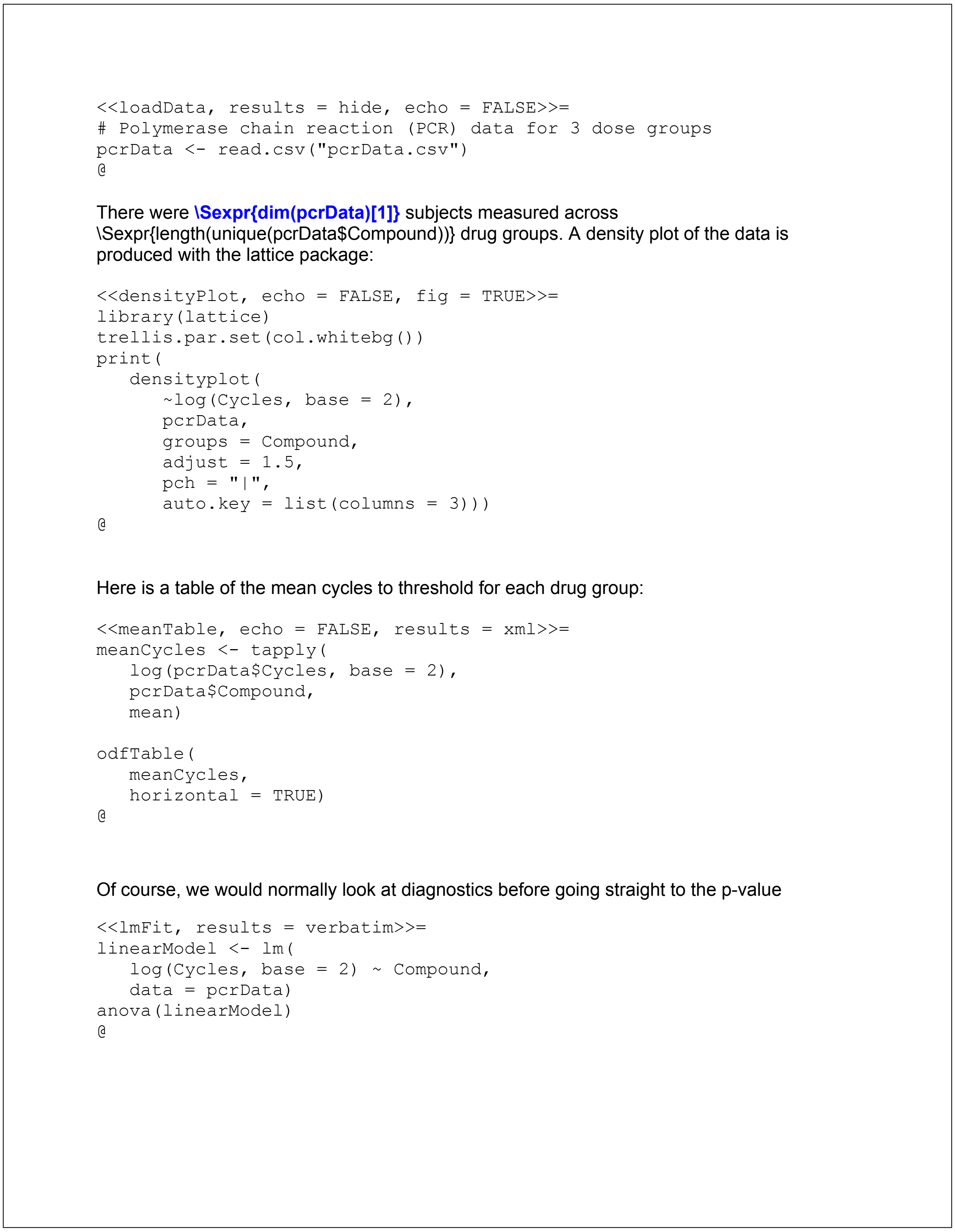
can be used to initialize the package. The first argument provides an optimal label for the chunk. The `results` argument controls the destination of the output. A value of `hide` prevents any output from being sent to the document, whereas a value of `xml` is used when the code chunk generates output that is formatted in the Open Document format (see the next section for an example). Using `results = verbatim` formats the output of the chunk similar to how it would appear when printed at the command line. The `echo` argument specifies whether the R code should also be printed in the output.

These, and other options for code chunks in `odfWeave` are documented with the `Sweave` driver function, called `RweaveOdf`. There are some differences in options when compared to `Sweave` with LaTeX. For example, the `odfWeave` has more image file format options when compared to those available using LaTeX, so the `eps` and `pdf` arguments do not exist for `odfWeave`.

Tables

Tables can be created from vectors, matrices or data frames in R. The odfTable class can be used in code chucks to create a table in ODF format. For example, to create a table containing the iris data, the code chunk might look like:

```
<<irisTable, results = xml>>=
   # create nice column labels for illustration
   irisCols <- gsub("\\.", " ", names(iris))
   odfTable(
      iris,
      useRowNames = FALSE,
      colnames = irisCols)
@
```



```
<<loadData, results = hide, echo = FALSE>>=
# Polymerase chain reaction (PCR) data for 3 dose groups
pcrData <- read.csv("pcrData.csv")
@
```

There were **\Sexpr{dim(pcrData)[1]}** subjects measured across \Sexpr{length(unique(pcrData$Compound))} drug groups. A density plot of the data is produced with the lattice package:

```
<<densityPlot, echo = FALSE, fig = TRUE>>=
library(lattice)
trellis.par.set(col.whitebg())
print(
   densityplot(
      ~log(Cycles, base = 2),
      pcrData,
      groups = Compound,
      adjust = 1.5,
      pch = "|",
      auto.key = list(columns = 3)))
@
```

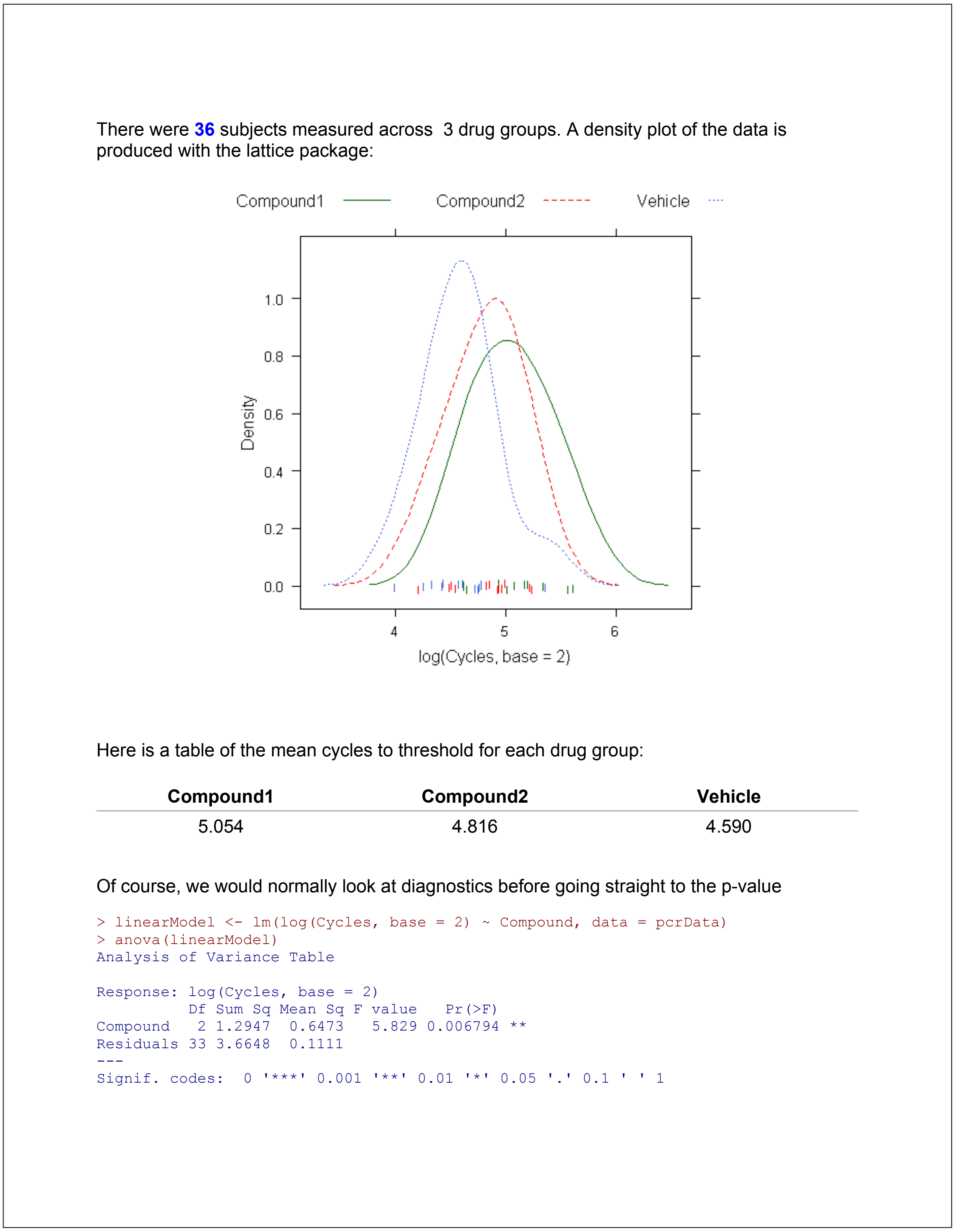
Here is a table of the mean cycles to threshold for each drug group:

```
<<meanTable, echo = FALSE, results = xml>>=
meanCycles <- tapply(
   log(pcrData$Cycles, base = 2),
   pcrData$Compound,
   mean)

odfTable(
   meanCycles,
   horizontal = TRUE)
@
```

Of course, we would normally look at diagnostics before going straight to the p-value

```
<<lmFit, results = verbatim>>=
linearModel <- lm(
   log(Cycles, base = 2) ~ Compound,
   data = pcrData)
anova(linearModel)
@
```

Figure 1: An example of an ODF document containing Sweave tags as viewed in OpenOffice.



There were **36** subjects measured across 3 drug groups. A density plot of the data is produced with the lattice package:

Here is a table of the mean cycles to threshold for each drug group:

Compound1	Compound2	Vehicle
5.054	4.816	4.590

Of course, we would normally look at diagnostics before going straight to the p-value

```
> linearModel <- lm(log(Cycles, base = 2) ~ Compound, data = pcrData)
> anova(linearModel)
Analysis of Variance Table

Response: log(Cycles, base = 2)
          Df Sum Sq Mean Sq F value   Pr(>F)
Compound   2 1.2947  0.6473   5.829 0.006794 **
Residuals 33 3.6648  0.1111
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Figure 2: The processed ODF document as viewed in OpenOffice.

First, note that the results argument is set to `xml` since `odfTable` will create ODF markup for the table. To illustrate some of the options in `odfTable`, the column names for the data frame are altered (e.g., "Sepal.Length" becomes "Sepal Length") for the table headings. The argument `useRowNames` prevents extra column of row names from being created (if `useRowNames` were set to `TRUE` (the default), we would have had to create an extra entry in `irisCols` to account for the column of row names).

Tables for vectors and matrices are handled similarly. See the documentation for `odfTable` for the details (by typing `?odfTable` at the command prompt).

Images

Figures can be created with code chunks by using the `fig = TRUE` argument. Using the iris data as an example again, we can create a lattice scatterplot matrix of the iris measurements with distinct symbols for each species using

```
<<irisPlot, fig = TRUE>>=
   trellis.par.set(col.whitebg())
   out <- splom(iris[,-5],
      groups = iris$Species,
      pscales = 0,
      auto.key = list(columns = 3))
   # remember, with lattice graphics, we
   # must explicitly print the plot object
   print(out)
@
```

This creates the image file, copies it to the appropriate directory and inserts required markup in place of this code chunk.

The default image format produced is png, although there are many options within OpenOffice. Figure 3 is a screenshot of the available formats. The image format can be specified in the function `getImageDefs` and `setImageDefs`. There are options for the image file extension and driver. For example, to specify postscript graphics, the `type` attribute would be set to "eps" and the `device` attribute would be set to "postscript". Also, the image file size can be controlled independently of the size of the image as displayed in the document. The image attributes can be changed throughout the document.

As with regular `Sweave`, `Sweavehooks` can be used to set graphical parameters before code chunks.

If the need arises to insert a pre–existing image, the `odfInsertPlot` can be used from within a code chunk:

```
<<insertImage, echo = FALSE, results = xml>>=
   imageMarkup <- odfInsertPlot(
      "myImage.gif",
      height = 2.7,
      width = 2.8,
      externalFile = TRUE)
   cat(imageMarkup)
@
```

This function will copy the image from the original location to the appropriate destination, but unlike `odfTable`, the `odfInsertPlot` function doesn't automatically print the output and `cat` must be used.

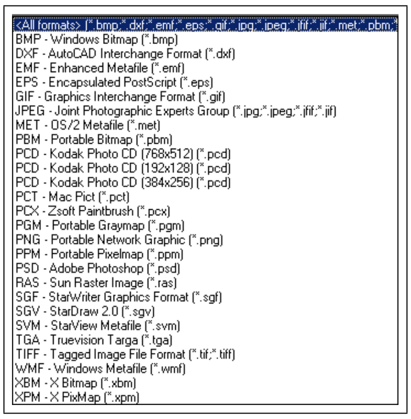
Formatting the Output

There are two main components to specifying output formats: style definitions and style assignments. The definition has the specific components (such as a table cell) and their format values (e.g., boxed with solid black lines). The function `getStyleDefs` can fetch the pre–existing styles in the package. For example:

```
> getStyleDefs()$ArialNormal
$type
[1] "Paragraph"

$parentStyleName
[1] ""

$textAlign
[1] "center"

$fontName
[1] "Arial"

$fontSize
[1] "12pt"

$fontType
[1] "normal"

$fontColor
[1] "#000000"
```

These can be modified and new definitions can be added. The function `setStyledefs` "registers" the style changes with the package. When `odfWeave` is called, these definitions are written to the style sections of the XML files. There is a second mechanism to assign styles to specific output elements. The functions `getStyles` and `setStyles` can be used to tell `odfWeave` which style definition to use for a particular output:

```
> currentStyles <- getStyles()
> currentStyles
$paragraph
[1] "ArialNormal"

$input
[1] "ttRed"

$output
[1] "ttBlue"
```



```
<All formats> (*.bmp;*.dxf;*.emf;*.eps;*.gif;*.jpg;*.jpeg;*.jfif;*.jif;*.met;*.pbm;*
BMP - Windows Bitmap (*.bmp)
DXF - AutoCAD Interchange Format (*.dxf)
EMF - Enhanced Metafile (*.emf)
EPS - Encapsulated PostScript (*.eps)
GIF - Graphics Interchange Format (*.gif)
JPEG - Joint Photographic Experts Group (*.jpg;*.jpeg;*.jfif;*.jif)
MET - OS/2 Metafile (*.met)
PBM - Portable Bitmap (*.pbm)
PCD - Kodak Photo CD (768x512) (*.pcd)
PCD - Kodak Photo CD (192x128) (*.pcd)
PCD - Kodak Photo CD (384x256) (*.pcd)
PCT - Mac Pict (*.pct)
PCX - Zsoft Paintbrush (*.pcx)
PGM - Portable Graymap (*.pgm)
PNG - Portable Network Graphic (*.png)
PPM - Portable Pixelmap (*.ppm)
PSD - Adobe Photoshop (*.psd)
RAS - Sun Raster Image (*.ras)
SGF - StarWriter Graphics Format (*.sgf)
SGV - StarDraw 2.0 (*.sgv)
SVM - StarView Metafile (*.svm)
TGA - Truevision Targa (*.tga)
TIFF - Tagged Image File Format (*.tif;*.tiff)
WMF - Windows Metafile (*.wmf)
XBM - X Bitmap (*.xbm)
XPM - X PixMap (*.xpm)
```

Figure 3: A screenshot of available image formats in OpenOffice (version 2.0).

```
$table
[1] "Rtable1"

$cell
[1] "noBorder"

$header
[1] "lowerBorder"

$cellText
[1] "ArialCentered"

$headerText
[1] "ArialCenteredBold"

$bullet
[1] "Rbullet"
```

For example, the `input` and `output` elements control how R code and command–line output look. To change either of these, an existing definition can be assigned to these entries and reset using `setStyles(currentStyles)`. Unlike the style definitions, the style assignments can be modified throughout the R code (although only the style definitions available when `odfWeave` is called can be used).

The package also contains a function, `tableStyles`, that can be used to differentially format specific cells and text in tables.

Other Functionality

`odfWeave` can also be used to created bulleted lists from vectors via the `odfItemize` and formatted text can be written using `odfCat`.

There are a few miscellaneous functions also included in the package. `pkgVersions` can create a summary of the packages and versions in the search path. The function `listString` takes a vector and returns a textual listing. For example,

```
listString(letters[1:4])
```

would become "a, b, c and d". Also, `matrixPaste` can take a series of character matrices with the same dimensions and perform an element–wise paste.

Summary

odfWeave provides functionality for weaving R into documents without dealing with the details of the underlying markup language. The usage of `odfWeave` closely mirrors that of standard `Sweave` so current `Sweave` users should be able to move between the two formats easily.

Appendix: Under the Hood of an ODF File

Open Document Format files are compressed archives of several files and folders which can be decompressed using standard tools such as `unzip`, `jar` or `WinZip`. A typical document will have a structure similar to:

```
Name
----
content.xml
layout-cache
META-INF/
META-INF/manifest.xml
meta.xml
mimetype
Pictures/
Pictures/rplot.png
settings.xml
styles.xml
Thumbnails/
Thumbnails/thumbnail.png
```

The main points of interest are:

- 'content.xml' contains the text, tables, lists and any other user–supplied content. It contains limited formatting.

- 'style.xml' contains most of the formatting definitions, including fonts, table formats, and page layout.
- the 'Pictures' directory contains any images that were inserted into the document. When images are added, the image files are saved in the archive in their native format

`Sweave` tags inserted into the body of a document would primarily effect 'content.xml'.

`OdfWeave` requires a zip/unzip utility to gain access to these files, process them and re-assemble the ODT file so that OpenOffice can re-open it.

Bibliography

F. Leisch. Sweave, Part I: Mixing R and LaTeX. *R News*, 2(3):28–31, 2002. URL http://cran.r-project.org/doc/Rnews.

F. Leisch. Sweave, Part II: Package Vignettes. *R News*, 3(2):21–34, 2003. URL http://cran.r-project.org/doc/Rnews.

Organization for the Advancement of Structured Information Standards (OASIS). *Open Document Format for Office Applications (OpenDocument)* v1.0. May, 2005. URL http://www.oasis-open.org/committees/download.php/12572/OpenDocument-v1.0-os.pdf.

Max Kuhn
Research Statistics
Pfizer Global Research and Development
max.kuhn@pfizer.com

plotrix

A package in the red light district of R

by Jim Lemon

The social science of statistics

It is firmly established that statistics can assist the deceptive, but less well recognized that it can also aid those practising one of the other ancient arts. Some statisticians do attain the position of courtesans to the mighty, where they may spurn the lowlier tasks of their trade. Most of us must entertain the whims of the paying customer.

Like many R users, I did not aspire to be a statistician. The combination of a bit of relevant talent and the disinclination of others led to my recruitment. Readers are probably well aware that legions of reluctant statisticians desperately scan their sometimes inadequate knowledge in the effort to produce what others with considerably less knowledge demand.

In this spirit I began to program the functions that were included in the initial version of **plotrix** (Lemon, 2006). Inspiration was found not only in my own efforts to use R as I had previously used other statistical or graphical packages, but in the pleas of others like me that found their way to the R help list. Commercial software has lent a sympathetic, if not always disinterested, ear to such requests. You want a 3D pie chart with clip art symbols floating in the transparent sectors and you get it, for a price. There is a strong desire for the conventional in the average consumer of statistics, even if the conventional is not the most informative.

I would like to introduce **plotrix** using three topics in plotting that have been the subject of some discussion on the R help list as well as a little parade of the functions.

How to slice the pie

Consider the aforementioned pie chart. There have been several discussions upon the wisdom of using such illustrations, typically initiated by a query from an R user. Despite the limitations of the pie chart, they seem common. To test this impression, I asked our friend Google to inform me on the phrase "division of wealth" and then stepped through the listing until I found an illustration of this phrase. On the twelfth hit, there was ... a pie chart (Ng, 2006). As an example of the way in which the **plotrix** package may be used, I reproduced it in Figure 1. The code I used is shown below.

```
x11(width=7, height=4)
par(mar=c(0,0,0,0), xpd=FALSE)
plot(0, xlim=c(-2,2.2), ylim=c(-1,1),
    xlab="", ylab="", type="n")
rect(-2.5, -1.5, 2.5, 1.5, col="#ffffcb")
wealth.pct<-c(41, 15, 9.4, 6.4, 4.8, 3.7,
             2.9, 2.4, 2, 1.6, 12.1)
sector.colors<-c("#fe0000", "#ffff00", "#0000fe",
                 "#ff00fe", "#00ffff", "#fc6866",
                 "#666632", "#d2ffff", "#fdffcd",
                 "#ffcb65", "#ffffff")
floating.pie(0, 0, wealth.pct, col=sector.colors,
            startpos=pi/2)
text(-1.55, 0.2,
    "wealth owned by the\nrichest one percent
of the population")
text(-1.4, -0.9,
```