hwriterPlus: Extending the hwriter Package

David J. Scott
University of Auckland

Abstract

The R package **hwriter** provides a convenient way of producing hypertext markup language documents which incorporate statistical analyses using R. This package extends the capability of **hwriter** to allow the incorporation of scalable vector graphics, output from R or complete R sessions and the display of mathematical expressions using LaTeX format. The resulting documents may be successfully viewed in recent versions of common browsers. Additionally such documents may also be opened in recent versions of Microsoft Word.

Keywords: R, HTML, XML, Microsoft Word.

1. Introduction

For producing documents in pdf or postscript formats which include the results of statistical analysis using R, **Sweave** is commonly used. Because **Sweave** uses LaTeX for document production it can produce documents of almost infinite complexity incorporating text, tables, and graphics. The myriad styles and add-on packages available for LaTeX can also be used. However **Sweave** cannot produce HTML (hypertext markup language) or Microsoft Word documents directly and the requirement that LaTeX be installed is a problem in some situations. To produce HTML there are currently two packages on CRAN, **hwriter** (Pau 2010) and **R2HTML** (Lecoutre 2003). In both cases the only software which needs to be installed is R and the relevant package. The source files for both **hwriter** and **R2HTML** are pure R code and there is only one processing step, consisting of running the R code. These aspects are in contrast to the use of **Sweave** where the source file alternates chunks of LaTeX and of R, and there are at least two steps in the process with a number of additional files being produced. Use of **hwriter** (or **R2HTML**) is in many ways a simpler, more lightweight approach to automated document production than the use of **Sweave**. The challenge though is to enhance the capability of **hwriter** to try and match the ability of **Sweave** to produce complex documents. This is the motivation behind **hwriterPlus**.

As an example of the sort of problem which this software addresses, the author was tasked with providing a program to produce a weekly report taking data from a spreadsheet to produce summary statistics and graphs in a convenient format. This was relatively easy using R and the package **hwriter** which could easily be installed on the client's computer. The document was produced in HTML format with graphs in Windows metafile format. The document could be read by Microsoft Word or viewed in Internet Explorer, so besides R and **hwriter**, no additional software needed to be installed on the client's computer.

Using R with **hwriter** has also proved valuable in providing reports to statistical consulting clients who wish to include results from the report in papers for publication.

hwriter though, has limitations in what it can produce. There is the capability for incorporating individual mathematical symbols, superscripts and subscripts, but not more complex mathematical expressions without the use of additional software outside of R. Images may be included also in various formats, but no cross-platform vector graphic format can be used. Windows metafile is specific to Windows computers and may only be displayed by Internet Explorer, not Firefox nor Chrome nor Safari. Other available image formats are bitmaps, including jpg and png. The only cross-platform vector image format which can be displayed in Firefox and Internet Explorer is SVG (scalable vector graphics).

A further useful capability already available in **Sweave** is the ability to show the output produced by a sequence of R commands or both the commands and output, reproducing part (or even all) of an R session.

The small package **hwriterPlus** extends the capability of **hwriter** to enable the incorporation of LaTeX expressions in documents in HTML format when displayed in up-to-date versions of Firefox, Internet Explorer, Chrome and Safari. In addition SVG objects can be incorporated into such documents and displayed in those browsers. Finally the ability to display the result of a sequence of R commands or part of an R session has also been included in **hwriterPlus**.

This paper describes the features and implementation of **hwriterPlus** and gives examples of its use. There is also a discussion of the problems of using XML (eXtensible markup language) rather than HTML, and of viewing HTML documents using Microsoft Word. In the package **hwriterPlus**, which is available on CRAN at `cran.r-project.org/web/packages/hwriterPlus/index.html` there are two example files showing the capability of the package. These are called BrowserExampleMathJax.R and NumberedHeadingsExample.R, both located in the inst/examples subdirectory. There is also a version of this document (a vignette) in the inst/doc subdirectory.

Throughout the present paper, in the R code and the resulting HTML, lines have been broken or otherwise reformatted to ensure that the code would fit within the width of the page, or to enhance readability. In most cases this doesn't matter. Where there is confusion, the examples in the package **hwriterPlus** can be consulted.

Some understanding of HTML and related concepts such as CSS (cascading style sheets) is desirable when reading this paper. A suitable introduction which is readily available and includes the use of MathML (Mathematical Markup Language) is Siegrist (2007). A thorough introduction (excluding MathML) is given in Murrell (2009).

2. Incorporating LaTeX in HTML

When considering how to add the ability to display mathematical expressions in LaTeX format to **hwriter**, it is natural to examine the package **R2HTML** which already has this facility. In **R2HTML** the LaTeX expressions are processed using the JavaScript code ASCIIMathML from Jipsen (2005) which converts LaTeX expressions to Presentation MathML expressions. Initially I reverse-engineered **R2HTML** to use ASCIIMathML to display mathematical expressions in LaTeX format. There were problems with this approach however. First of all, ASCIIMathML is poorly supported across different browsers. Firefox works best. Recent versions of Internet Explorer are also satisfactory, but less up-to-date versions require the installation of MathPlayer (Design Science 2011). Neither Chrome nor Safari appear to support ASCIIMathML, nor is there an add-on available for them. Secondly, the head element specification required to ensure that Internet Explorer properly processes the JavaScript for ASCIIMathML is complex and rather unforgiving. It required considerable experimentation to get it to work properly.

Most importantly though, ASCIIMathML has been supplanted by the JavaScript implementation provided by MathJax, `http://www.mathjax.org/`. MathJax "is a joint project of the American Mathematical Society, Design Science, Inc., and the Society for Industrial and Applied Mathematics" (see the Sponsorship tab on the website MathJax (2012)). It appears to live up to its claim to display properly in all modern browsers.

For these reasons **hwriterPlus** uses MathJax for display mathematical expressions and equations.

The first step in creating an HTML document is the insertion of the head element. In **hwriter** this is done via the `openPage` function. For **hwriterPlus** rather than modify `openPage`, I wrote a new function `newPage` which has some additional arguments, but also removes some arguments present in `openPage`. To enable the use of MathJax, the minimal evocation of `newPage` is R code similar to the following (formatting changed to fit on this page):

```
pg <- newPage("file.html",
             title = "Example of a Document for Display in a Browser",
             link.javascript =
             c("http://cdn.mathjax.org/mathjax/latest/
                MathJax.js?config=TeX-AMS-MML_HTMLorMML"))
```

This will open a text document with the name file.html in the current folder and prepare it to accept HTML code making up the body element. It will also place essentially the following head element in the file:

```
<!DOCTYPE html>
<html ><head>
<title>Example of a Document for Display in a Browser</title>

<script type = "text/javascript"
    src = "http://cdn.mathjax.org/mathjax/latest/
           MathJax.js?config=TeX-AMS-MML_HTMLorMML">
</script>

</head>
<body>
```

Use of MathJax as indicated in the above example requires an active internet connection so that the latest version of MathJax can be accessed. Alternatively a local copy of `MathJax.js` can be downloaded and accessed. All that is required is that the argument `link.javascript` should specify the location of `MathJax.js`. The required format is
`link.javascript = "path-to-MathJax/MathJax.js"`.

Once this head element has been inserted in the document it is possible to insert LaTeX code into the document to produce both inline and displayed mathematical expressions. For an inline expression MathJax requires the delimiters to be `\(...\)`. Because `hwrite` is essentially a wrapper around `cat` and it must be made explicit that backslashes be produced rather than the backslash being treated as an escape symbol, all backslashes must be doubled. Thus

```
hwrite("Here is an inline expression:
       \\(\\int_{-\\infty}^{1}f(x)dx\\),
       page = pg, br = TRUE)
```

will produce the HTML code

```
Here is an inline expression: \(\int_{-\infty}^{1}f(x)dx\)
```

in the document file.html. When the file is opened in a browser this will appear as

Here is an inline expression: $\int_{-\infty}^{1} f(x)dx$

although the exact typsetting of the expression will differ from what is shown on this page, and from browser to browser.

Producing displayed mathematical expressions is more complicated. They are in fact produced by inserting a table in the HTML document. In **hwriterPlus** the command used is `hwriteLatex` along with `as.latex`, derived from the **R2HTML** functions `HTML.latex` and `as.latex`. As an example

```
hwriteLatex(as.latex("\\int_{-\\infty}^{1}f(x)dx",
                     inline = FALSE, count = FALSE),
            page = pg,
            table.attributes = "border = '1'",
            tr.attributes = "bgcolor = 'white'")
```

produces

```
<br /><center><table border = '1'>
<tr bgcolor = 'white'>
<td align = 'center'>\[\int_{-\infty}^{1}f(x)dx\]</td>
</tr>
</table></center><br />
```

displayed in a browser more or less as

$$\boxed{\int_{-\infty}^{1} f(x)dx}$$

If numbered equations are required then there is additional complexity. For example

```
hwriteLatex(as.latex("\\{ 26.119 < \\sum_{i=1}^n(X_i-\\bar{X})^2\\}
            \\bigcup\\ \\{ 5.629 > \\sum_{i=1}^n (X_i-\\bar{X})^2 \\}.",
            inline = FALSE, label = "equation1"),
            page = pg,
            tr.attributes = "bgcolor = 'white'",
            td.attributes = c("width = '50'", "align = 'center'",
                              "align = 'right' width = '50'"))
```

produces

```
<br /><center><table border = '0' width = '90%'>
<tr bgcolor = 'white'>
<td width = '50'>&nbsp;</td>
<td align = 'center'>
\[\{ 26.119 < \sum_{i=1}^n(X_i-\bar{X})^2\}
\bigcup\ \{ 5.629 > \sum_{i=1}^n (X_i-\bar{X})^2 \}.\]
</td>
<td align = 'right' width = '50' id = 'eq:equation1'>(1)</td>
</tr>
</table></center><br />
```

which is displayed essentially as

$$\{26.119 < \sum_{i=1}^n (X_i - \bar{X})^2\} \bigcup \{5.629 > \sum_{i=1}^n (X_i - \bar{X})^2\} \tag{1}$$

Note the inclusion of the `id` element with a name attribute. In this the name attribute will be `eq:equation1`, as a result of the equation label being specifed as equation1. This permits reference to be made to this equation elsewhere in the document, as follows. The R code

```
hwrite("Here is the link to the equation: ",
       pg, br = FALSE)
hwrite("Numbered Equation.", pg, br = TRUE,
       link = "#eq:equation1")
```

will produce

```
Here is the link to the equation:
<a href="#eq:equation1">Numbered Equation.</a><br/>
```

This creates a hyperlink to equation 1 with the text "Numbered Equation". This is an attempt at an HTML equivalent of the LaTeX approach of creating a labelled equation and a reference to that equation elsewhere in the document using the `\ref` command. There is a convenience function which will produce the equation number when supplied with the equation label, this is eqRef. In the example above if the equation with the label equation1 is the first numbered equation in the document, then `eqRef{"equation1"}` returns the number 1, so eqRef performs the same function as `\ref` in LaTeX.

Whereas **R2HTML** uses JavaScript to implement equation numbering, in **hwriterPlus**, only R code is used. This is in part because for many R users, JavaScript will be very foreign. In addition the **hwriterPlus** implementation allows equations to be referenced by name as in LaTeX rather than the writer having to remember the equation number. When newPage is called initially, a vector of length one is created called hwriterEquation, with the initial value of 0. A second vector, a character vector of length zero is also created with the name hwriterEquationList, which will hold the names of any numbered equations created. Each time a numbered equation is inserted into the document using hwriteLatex, hwriterEquation is incremented and the resulting number is inserted into the table which displays the numbered equation. If a label for the equation has been supplied, then the label prefixed by `eq:` is appended to hwriterEquationList. If no label was supplied, then a default label is created by concatenating the prefix `eq:` and the equation number.

3. Incorporating SVG

Images are usually included in an R document using the `image` element. In the package **hwriter** the function `hwriteImage` essentially surrounds the location of an image file with an `<image></image>` tag pair. SVG images are different however and use an `object` element. Moreover the required attributes of the `object` element vary according to browser. SVG can be used to actually create an image when the file is browsed, as opposed to the browser simply displaying a previously created SVG format image. The ability to include SVG images in a file has been added to **hwriterPlus** via the `hwriteSVG` function. This produces a complicated piece of HTML which was taken from the Quick Start (`http://codinginparadise.org/projects/svgweb/docs/QuickStart.html`) available on the web page of the svgWeb project (svgWeb 2011). To insert the SVG image with the filename `helloworld.svg` into an HTML document, the following HTML code is recommended:

```
 <!--[if !IE]>-->
  <object data="../svg-files/helloworld.svg" type="image/svg+xml"
          width="200" height="200" id="mySVGObject"> <!--<![endif]-->
<!--[if lt IE 9]>
  <object src="../svg-files/helloworld.svg" classid="image/svg+xml"
          width="200" height="200" id="mySVGObject"> <![endif]-->
<!--[if gte IE 9]>
  <object data="../svg-files/helloworld.svg" type="image/svg+xml"
          width="200" height="200" id="mySVGObject"> <![endif]-->
  </object>
```

For those unfamiliar with HTML code this is quite strange, but certainly works. The following snippet of R code creates a lattice plot of data concerning cats provided in the **MASS** package and inserts it into the document file.html.

```
library(MASS)
library(Cairo)
CairoSVG("cats.svg", width = 4, height = 4)
lattice.options(theme = "col.whitebg")
print(xyplot(Hwt ~ Bwt|Sex, data = cats, type = c("p", "r")))
dev.off()
hwriteSVG("cats.svg", pg, height = 600, width = 600, id = "catsSVG",
          center = FALSE, br = TRUE)
```

The code produced in file.html is

```
<!--[if !IE]>-->
              <object data = 'cats.svg' type = 'image/svg+xml'
```

```
                width = '600' height = '600'
                id = catsSVG border = '0'> <!--<![endif]-->
<!--[if lt IE 9]>
<object src = 'cats.svg' classid = 'image/svg+xml'
                width = '600' height = '600'
                id = catsSVG border = '0'> <![endif]-->
<!--[if gte IE 9]>
<object data = 'cats.svg' type = 'image/svg+xml'
                width = '600' height = '600'
                id = catsSVG border = '0'> <![endif]-->
</object><br/>
```

One problem with SVG objects in web documents is that they are treated differently by different browsers. Firefox is the most lenient in what it requires and will produce an image if the width and height arguments are omitted. If either of these arguments is too small to accommodate the image, the image will be produced with slider bars. Internet Explorer requires both the width and height arguments to be specified, or no image is produced. Also, the actual size of the displayed image can vary wildly from browser to browser.

4. Capturing R output and R sessions

Sweave has the ability to capture the output of a sequence of R commands using the instructions starting a chunk of code, `<<echo=FALSE, results=verbatim>>=`. To capture both commands and output the instructions `<<echo=TRUE, results=verbatim>>=` are used. These facilities have been incorporated into **hwriterPlus**.

To capture output from R the commands producing the output are given as an argument of the `capture.output` function, and the resulting output is inserted into the document using the **hwriterPlus** function `hwriteOutput`. Thus

```
aggOut <-
    capture.output(data(iris),
                   aggregate(Sepal.Length~Species, iris, mean)
                   )
hwriteOutput(aggOut, pg, center = FALSE, br = TRUE)
```

inserts the following HTML code in file.html:

```
<pre style = 'font-size:10pt'>
     Species Sepal.Length
1     setosa        5.006
2 versicolor        5.936
3  virginica        6.588
</pre><br/>
```

The element `pre` means the text is to be treated as preformatted, and is akin to the LaTeX `verbatim` environment. Note that there are commas between the commands whose output is to be captured by `capture.output`. This requirement is explained and amplified in the help and examples for the function `capture.output`.

To capture a full R session, that is including commands and output, it appears to be necessary to write to a file, then read back the contents of the file. The package **TeachingDemos** has this facility via the command `txtStart`. Here is an example of some R code

```
tmpFile <- tempfile("Session")
txtStart(tmpFile)
clotting <-
    data.frame(
               u = c(5,10,15,20,30,40,60,80,100),
               lot1 = c(118,58,42,35,27,25,21,19,18),
               lot2 = c(69,35,26,21,18,16,13,12,12)
               )
clotting
coef(glm(lot1 ~ log(u), data=clotting, family=Gamma))
txtStop()
sessionOut <- readLines(tmpFile)
hwriteOutput(sessionOut, pg, center = FALSE, br = TRUE)
```

which produces

```
<pre style = 'font-size:10pt'>>
clotting <- data.frame(u = c(5, 10, 15, 20, 30, 40, 60, 80, 100),
+ lot1 = c(118, 58, 42, 35, 27, 25, 21, 19, 18), lot2 = c(69,
+
+ 35, 26, 21, 18, 16, 13, 12, 12))
> clotting
    u lot1 lot2
1   5  118   69
2  10   58   35
3  15   42   26
4  20   35   21
5  30   27   18
6  40   25   16
7  60   21   13
8  80   19   12
9 100   18   12
> coef(glm(lot1 ~ log(u), data = clotting, family = Gamma))
(Intercept)      log(u)
-0.01655438  0.01534311
</pre><br/>
```

Note that unfortunately `txtStart` does not capture the line breaks correctly for a multiline input command.

I approached Professor Ross Ihaka with this problem and he provided me the function `script` which will properly capture an R session. Here is an example of its use. The R code

```
tmpFile <- tempfile("Session")
script(tmpFile)
clotting <-
    data.frame(
               u = c(5,10,15,20,30,40,60,80,100),
               lot1 = c(118,58,42,35,27,25,21,19,18),
               lot2 = c(69,35,26,21,18,16,13,12,12)
               )
clotting
coef(glm(lot1 ~ log(u), data=clotting, family=Gamma))
q()
sessionOut <- readLines(tmpFile)
sessionOut
hwriteOutput(sessionOut, pg, br = TRUE)
```

produces the following

```
<pre style = 'font-size:10pt'>Script started on Sat Sep 01 19:17:13 2012
> clotting <-
>     data.frame(
>                u = c(5,10,15,20,30,40,60,80,100),
>                lot1 = c(118,58,42,35,27,25,21,19,18),
>                lot2 = c(69,35,26,21,18,16,13,12,12)
>                )
> clotting
    u lot1 lot2
1   5  118   69
2  10   58   35
3  15   42   26
4  20   35   21
5  30   27   18
6  40   25   16
7  60   21   13
8  80   19   12
9 100   18   12
> coef(glm(lot1 ~ log(u), data=clotting, family=Gamma))
(Intercept)      log(u)
-0.01655438  0.01534311
> q()
Script done on Sat Sep 01 19:17:13 2012 </pre><br/>
```

The function `script` places some information about the session being recorded at the beginning and at the end of the session. In many cases that information is not needed. The function `hwriteScript` is derived from `hwriteOutput`, the only difference being that by default it will drop the information lines (one at the start and two at the end) which `script` places in the temporary file.

5. Displaying R objects

hwriter is able to display R objects such as vectors, matrices, and dataframes. An extensive example is given in the help to the function `hwrite`. This example may also be seen on Gregoire Pau's website at http://www.embl.de/~gpau/hwriter/.

For output of tables, **xtable** is useful since the function `print.xtable` can produce HTML. Here is some code which will produce a nicely formatted analysis of variance table for example.

```
require(xtable)
data(tli)
fm1 <- aov(tlimth ~ sex + ethnicty + grade + disadvg, data=tli)
fm1.table <- print(xtable(fm1), type="html")
hwrite(fm1.table, pg, center = TRUE, br = TRUE)
```

Note that the way that R objects are formatted in the HTML document can be controlled by the use of a CSS file. In particular the appearance of the **hwriter** example document is governed by the CSS style sheet which comes with **hwriter**.

6. Links in documents

Internal and external links are easy to achieve in HTML documents. External links use the `a` element. So a link to the Department of Statistics website at the University of Auckland requires the HTML code

```
 <a href="http://www.stat.auckland.ac.nz/uoa/">
 The Department of Statistics.</a>
```

which is easily produced by the R code

```
hwrite("The Department of Statistics.", pg, br = TRUE,
        link = 'http://www.stat.auckland.ac.nz/uoa/')
```

Internal links within an HTML document are very useful since they allow cross-referencing. The mechanism is to create anchors, which are named objects within the HTML document, then other parts of the document can link to an anchor using an `a` element. To name an object, either an `a` element can be used or an `id` attribute may be added to the object's definition.

Here is an example of using the 'a' element approach. The R code

```
hwrite(hwrite("Entering Text", name = "intro"), pg,
       heading = 1, center = FALSE, br = TRUE)
```

produces the HTML

```
<h1><a name="intro">Entering Text</a></h1><br/>
```

which is a level 1 heading with the name `"intro"`. To create a reference to this heading we can use the R code

```
hwrite("Here is a link to the heading named using the
        <font face='monospace'>a</font> element: ",
       pg, br = FALSE)
hwrite("Entering Text.", pg, br = TRUE,
       link = "#intro")
```

which produces

```
Here is a link to the heading named using the
<font face='monospace'>a</font> element:
<a href="#intro">Entering Text.</a><br/>
```

As an example of the use of an `id` attribute, the R code

```
hwrite("Rendering Mathematics", pg, id = "mathematics",
       heading = 1, center = FALSE, br = TRUE)
```

produces

```
<h1 id="mathematics">Rendering Mathematics</h1><br/>
```

which has the anchor `#mathematics`.

The code to produce numbered equations given in Section 2 includes the creation of an anchor. If that equation is the first in the document the anchor is then `#equation1` and a link can be created in the usual way.

7. Extensible Markup Language

I will not attempt to compare XML, and HTML. An authoritative description of XML is available at `http://www.w3schools.com/xml/xml_whatis.asp`. See also Murrell (2009). It is of interest however to see if **hwriter** and **hwriterPlus** can produce XML rather than HTML, not least because XML is used by recent versions of Microsoft Word, but also because XML is intended as a data description language, and hence is of interest to statisticians. If the extension of the document BrowserExampleMathJax.html is changed to .xhtml then the document will be treated by browsers as XHTML (eXtensible HTML) which is dialect of XML. This reveals many problems with the code in BrowserExampleMathJax.xhtml. Some problems which arose in earlier versions of **hwriterPlus** have been eliminated, such as the requirement that attribute values be quoted. Browsers are happy to accept `border = 0` when reading HTML, but `border = '0'` is required in XML. XML also doesn't accept a number of escapes such as `&gt;`, so in that case `>` must be used. The assignment in R, `<-`, needs to be identified as literal rather than the start of an XML element, so must be enclosed inside `<![CDATA[]]>` which is like a LaTeX `\verb` or `\verbatim` environment. The output from `print.xtable` also caused problems because it produces tags in upper case: so `TABLE`, `TR`, `TD` have to be changed to lower case.

8. Microsoft Word

Recent versions of Microsoft Word (where the extension is `docx`) use a modified and compressed XML format. From 2007, Word will also open an HTML document, so that **hwriter**, **hwriterPlus** and **R2HTML** can be used to produce documents which can be opened with Microsoft Word. Some aspects of an HTML document are recognised by Word, for example, heading styles transfer appropriately, and links created using an `a` element or an `id` attribute are recognised. Numbered equations and embedded LaTeX are not recognised, nor can SVG objects be displayed. One approach to deal with embedded LaTeX is to use MathType (see `http://www.dessci.com/en/products/mathtype/`). Unfortunately, for inline mathematical expressions, MathType expects `$...$` delimiters, whereas MathJax requires `\(...\)`. Once the `\(...\)` delimiters have been replaced by `$...$`, the MathType Toggle TeX facility will enable both inline and displayed expressions to be rendered appropriately.

A further problem with using MS Word when the document contains images is that the images are not stored within the document. They can be embedded easily however by following the instructions on this webpage: `http://www.onemanwrites.co.uk/2011/09/13/how-to-embed-linked-images-in-word-2010/`. In this form it is easy to send the document to someone else.

9. Discussion

There are many extensions which may be considered for **hwriterPlus**. One possible extension is to allow LaTeXMathML, see for example Knisley (2007). This permits the inclusion of larger LaTeX structures such as theorem environments, but also tables. Not all of the rich features of LaTeX tables are available however.

R2HTML includes grid JavaScript and a grid CSS file. This is of interest for producing rich data representations in an HTML file. According to Permessur (2010):

> A data grid can help address concerns of HTML tables with large data sets by providing features like sorting, filtering, searching, pagination and even in-line editing for your tables.

hwriter already includes some limited interaction with tables as can be seen on the **hwriter** examples page Pau (2010). The addition of grid JavaScript would allow much richer interactions. Grid JavaScript is an example of AJAX (Asynchronous JavaScript and XML) technologies whereby ”web applications can send data to, and retrieve data from, a server asynchronously (in the background) without interfering with the display and behavior of the existing page” (Wikipedia 2011).

10. Conclusion

hwriterPlus is able to produce useful documents in HTML format using only R code and small CSS and JavaScript files and is very simple to use. The files produced can be opened in Microsoft Word which will interpret most aspects of the files correctly.

References

Design Science (2011). "MathPlayer." http://www.dessci.com/en/products/mathplayer/. Online: accessed November 29, 2011.

Jipsen P (2005). http://www1.chapman.edu/~jipsen/mathml/asciimath.xml. Online: accessed November 29, 2011.

Knisley J (2007). "A Brief Description of LaTeXMathML." http://math.etsu.edu/LaTeXMathML/. Online: accessed December 5, 2011.

Lecoutre E (2003). "The **R2HTML** Package." *R News*, **3**(3), 33–36. URL http://cran.r-project.org/doc/Rnews/Rnews_2003-3.pdf.

MathJax (2012). http://www.mathjax.org/. Online: accessed October 22, 2012.

Murrell P (2009). *Introduction to Data Technologies*. Computer Science and Data Analysis Series. CRC Press, Boca Raton, FL.

Pau G (2010). ***hwriter**: HTML Writer - Outputs R objects in HTML format*. R package version 1.3, URL http://CRAN.R-project.org/package=hwriter.

Permessur A (2010). "15 JavaScript Data Grids to Enhance your HTML Tables." http://www.hotscripts.com/blog/15-javascript-data-grids-enhance-html-tables/. Online: accessed December 5, 2011.

Siegrist K (2007). "Mathematics with Structure and Style." *The Journal of Online Mathematics and Its Applications*, **7**. http://www.maa.org/joma/Volume7/Siegrist/StructureStyle.html.

svgWeb (2011). "svgweb - Scalable Vector Graphics for Web Browsers using Flash." http://code.google.com/p/svgweb/. Online: accessed November 30, 2011.

Wikipedia (2011). "Ajax (programming)." http://en.wikipedia.org/wiki/Ajax_(programming). Online: accessed December 8, 2011.

Affiliation:

David J. Scott
Department of Statistics
The University of Auckland
PB 92019

Auckland
New Zealand
Telephone: +64/9/923-5055
E-mail: `d.scott@auckland.ac.nz`
URL: `http://www.stat.auckland.ac.nz/~dscott/`