Journal of Statistical Software

MMMMMM YYYY, Volume VV, Code Snippet II. http://www.jstatsoft.org/

hwriterPlus: Extending the hwriter Package

David J. Scott
University of Auckland

Abstract

The R package **hwriter** provides a convenient way of producing HTML documents which incorporate statistical analyses using R. This package extends the capability of **hwriter** to allow the incorporation of SVG graphics which can be displayed in recent versions of common browsers, output from R or complete R sessions and for Firefox and Internet Explorer, the display of mathematical expressions using LaTeX format.

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** has the most functionality. 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 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 addressed in this paper, 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 or Chrome or 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 and Internet Explorer. In addition SVG graphic objects can be incorporated into such documents and displayed in up-to-date versions of Firefox, Internet Explorer, Chrome and Safari. 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 rather than HTML, and of viewing HTML documents using Microsoft Word.

Some understanding of HTML and related concepts such as CSS (Cascading Style Sheets) is required for reading this paper. A suitable introduction which is readily available and includes the use of MathML 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. Experimentation revealed that to successfully display LaTeX expressions in a browser as part of an HTML document a number of conditions must be satisfied:

1. There must be an `onload ="translate()"` statement in the HTML code in the `head` element of the document.

2. The JavaScript code must be available and its location specified in the `head` element of

the document via the inclusion of a `script` element.

3. The browser must be able to display Presentation MathML.

Dealing with the last requirement first, recent versions of Firefox and Internet Explorer can display Presentation MathML without any add-ons being required. Earlier versions of Internet Explorer typically require the installation of MathPlayer (Design Science (2011)). It seems that Chrome and Safari are not able to display Presentation MathML, nor are there addons available for these browsers.

R2HTML fulfills the first two of the requirements list via the `HTMLInitFile` function. This function however is a mysterious black box which creates a number of files in a directory structure and inserts additional head information to enable enhanced interaction with tables of data. (See Section 9 concerning this capability). The corresponding command in **hwriter** is the `openPage` command. This makes explicit the name of the file being opened for writing, any CSS (cascading style sheet) files to be loaded, any JavaScript files to be used, any HTML code to be added to the heading to specify attributes, and any HTML code to be added to the body of the document to specify body attributes. Because of this clarity I preferred to augment **hwriter** rather than **R2HTML** in order to create a package which would allow for mathematical expressions and the inclusion of SVG. **hwriterPlus** contains the function `newPage` which adds some arguments to `openPage`, specifically a `doctype` argument to allow specification of the document type via a `<!DOCTYPE ...>` declaration, and an `xmlns` argument which allows the specification of an XML namespace.

The package **hwriterPlus** contains an extended example of using R to produce an HTML document. The call to the function `newPage` to produce the document `BrowserExample.html` takes the form:

```
### Open file for writing
xmlns <- "xmlns:mml='http://www.w3.org/1998/Math/MathML'"
bodyAttributes <- list(onload = "translate()",
                       bgcolor = "FFFFFF",
                       background = "")
head <- '<object id="mathplayer" classid="clsid:32F66A20-7614-11D4-BD11-00104BD3F987"></object>
<?import namespace="mml" implementation="#mathplayer"?>'
p <- newPage("BrowserExample.html",
             title = "Example of a Document for Display in a Browser",
             doctype = "",
             xmlns = xmlns,
             link.css = c("../css/BrowserExample.css"),
             link.javascript = c("../javascript/ASCIIMathML.js"),
             body.attributes = bodyAttributes,
             head = head)
```

This produces the following heading text in the file `BrowserExample.html`:

```
<html xmlns:mml='http://www.w3.org/1998/Math/MathML' xml:lang = 'en' lang = 'en'><head>
<meta http-equiv="Content-Type" content="text/html; charset = utf-8"></meta>
<title>Example of a Document for Display in a Browser</title>
<object id="mathplayer" classid="clsid:32F66A20-7614-11D4-BD11-00104BD3F987"></object>
<?import namespace="mml" implementation="#mathplayer"?>
<script language="JavaScript" src="../javascript/ASCIIMathML.js"></script>
<link rel="stylesheet" type="text/css" href="../css/BrowserExample.css"></link>

<script>
nequations=0;</script>
</head>
<body onload="translate()" bgcolor="FFFFFF" background=""><br/>
```

Some explanation is required concerning this head information. First of all there is no `<!DOCTYPE ...>` declaration. This is not recommended but worked in the browsers I tried, whereas any `<!DOCTYPE ...>` declaration I tried caused problems with Internet Explorer. The `xmlns` statement provides a namespace for the MathML functions, along with the `<?import ..>` processing instruction. The forms of these components are taken from **R2HTML** code. The `object` element identifies the **MathPlayer** addon for browsers which require it. The first `script` element provides the location of the required JavaScript code, in this case giving a relative address in the file system. The `link` element provides the location of a CSS file which provides overriding styles for the document. Again this gives a relative address in the file system. The second `script` element establishes the name of a variable which will provide equation numbers for displayed equations. The attributes in the `<body>` tag cause the JavaScript to be run. (Note that there is a matching `</body>` tag at the end of the document.)

Once this head information 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 simply insert some LaTeX inside backticks, with all backslashes being doubled. Thus

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

will produce the HTML code

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

in the document **BrowserExample.html**, which when opened in a browser will produce

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.

The doubling of backslashes specified above is necessary 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.

In fact ASCIIMathML does not require backslashes at all but then users must be aware of how mathematical expressions will be parsed. For an introduction to ASCIIMathML see Gray (2007). Explicit specification of ASCIIMathML syntax may be found at Miedema (2011).

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 = p,
            table.attributes = "border = '1'",
            tr.attributes = "bgcolor = 'white'")
```

produces (some linebreaks have been added for clarity)

```
<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 = "equation number and label"),
            page = p,
            tr.attributes = "bgcolor = 'white'",
            td.attributes = c("width = '50'", "align = 'center'",
                              "align = 'right' width = '50'"))
```

produces (some linebreaks have been added for clarity)

```
<script>
 nequations = nequations+1;
 document.write("<a name = 'equation"+nequations+"'>&nbsp;</a>")
</script>

<br /><span class = 'equation'>
Equation <script>document.write(nequations);</script> -
equation number and label
<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'>
<script>document.write('('+nequations+')')</script></td>
</tr></table></center><br />
```

which is displayed essentially as

Equation 1 - equation number and label

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

Note the inclusion of the `a` element with a name attribute which if this is the first named equation in the document will have the name `equation1`. This will allow reference to be made to this equation elsewhere in the document, as follows. The R code

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

will produce

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

which creates a hyperlink to Equation 1 with the text ”Numbered Equation”. This is an attempt 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. The present implementation requires the knowledge of the equation number when creating the reference so will not survive the reordering of equations.

3. Incorporating SVG Graphics

Images are usually included in an R document using the `image` element. Thus in **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 `BrowserExample.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", p, height = 600, width = 600, id = "catsSVG",
          center = FALSE, br = TRUE)
```

The code produced in `BrowserExample.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 graphics 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>>=`. Alternatively both commands and output can be captured using the alternative instructions `<<echo=TRUE, results=verbatim>`
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),
                   str(iris),
                   aggregate(Sepal.Length~Species, iris, mean)
                   )
hwriteOutput(aggOut, p, center = FALSE, br = TRUE)
```

inserts the following HTML code in **BrowserExample.html**:

```
<pre style = 'font-size:10pt'>'data.frame':      150 obs. of  5 variables:
 $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
 $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
 $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
 $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
 $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...
     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 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, p, 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.

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, p, center = TRUE, br = TRUE)
```

Note that 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.", p, 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 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"), p,
       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: ",
       p, br = FALSE)
hwrite("Entering Text.", p, 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", p, 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. XML and XHTML

I will not attempt to compare XML (eXtensible Markup Language), 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 of interest to statisticians. If the extension of the document **BrowserExample.html** is changed to **.xhtml** then the document will be treated by browsers as XHTML which is dialect of XML. This reveals many problems with the code in **BrowserExample.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 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 MS 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 graphics be displayed. One approach to deal with embedded LaTeX is to use MathType (see `http://www.dessci.com/en/products/mathtype/`. Provided there are not many instances of mathematical expressions in LaTeX in the document, a search can be made for backticks and the backticks replaced by `$` then use the MathType Toggle TeX facility to render the expressions 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 obvious 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. An alternative to MathML is MathJax, MathJax (2011). This is used by the Org-Babel system (Schulte, Davison, and Dye (2011)) to produce HTML from what is largely LaTeX. **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.

hwriterPlus is currently available on R-Forge, at `https://r-forge.r-project.org/R/?group_id=1269`.

References

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

Gray J (2007). "ASCIIMathML: now everyone can type MathML." *MSOR Connections*, **7**(3), 26–30.

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 (2011). "MathJax." `http://www.mathjax.org/`. Online: accessed November 29, 2011.

Miedema SA (2011). "ASCIIMathML Homepage." `http://www.asciimathml.com/`. Online: accessed November 29, 2011.

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.

Schulte E, Davison D, Dye T (2011). "Babel: active code in Org-mode." `http://orgmode.org/worg/org-contrib/babel/`. 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/`

Journal of Statistical Software `http://www.jstatsoft.org/`
published by the American Statistical Association `http://www.amstat.org/`

Volume VV, Code Snippet II *Submitted:* yyyy-mm-dd
MMMMMM YYYY *Accepted:* yyyy-mm-dd