Quality Control for Statistical Graphics: the graphicsQC package for R

Paul Murrell
The University of Auckland

Stephen Gardiner
The University of Auckland

Abstract

An important component of quality control for statistical graphics software is the ability not only to test that code runs without errors, but also to test that code produces the right result. The simple way to test for the correct result in graphical output is to test whether a test file differs from a control file; this is effective in determining whether a difference exists. However, the test can be significantly enhanced by also producing a graphical image of any difference; this makes it much easier to determine how and why two files differ. This article describes the **graphicsQC** package for R, which provides functions for producing and comparing files of graphical output and for generating a report of the results, including images of any differences.

Keywords: quality control, R, statistical graphics, XML, XSL.

1. Introduction

Quality control involves the maintenance of quality in a product, particularly by comparing the performance of a product with a set of specifications. When the product is software, an important part of quality control is *software testing*, where the software is run to determine whether it produces the desired output (Hower 2009; Horch 2003).

Software testing can take many different forms. For example, the most basic test is that the software runs at all, without "crashing"; a more strict test is that the software produces the correct output for a known set of inputs. The latter example involves comparing the output from one run of the software, the *test* output, with a set of *control* output to check for any differences. This is called *regression testing*.

The purpose of software testing is not only to detect when problems occur, but also to help in diagnosing the nature of the problem.

In the world of statistical computing, the output of software is, in many cases, a numeric result, such as tables of counts, group means and variances, p-values from hypothesis tests, and coefficients from fitting a model. Regression testing can be performed for this sort of output quite easily by recording a set of numeric control values in a text file and then comparing the test results using readily-available tools for comparing text files, such as the GNU diff utility (McKenzie, Eggert, and Stallman 2002, http://www.gnu.org/software/diffutils/diffutils.html). With numeric output in a text format, it is not only straightforward to detect that a problem exists, it is also straightforward to identify the *location* of a problem; it is easy to see which numbers are different from each other.

In the area of statistical graphics, the situation is a little less straightforward. Some graphical output, such as a PNG file, is binary rather than text-based. The diff utility can be used to determine that two binary files are different, but it does not provide useful information about the location of the difference, or *how* the two files differ. Even with text-based graphical output, such as PostScript files, knowing the location of a difference within a file may not be enough to know how the *appearance* of the graphical image has changed, which can make it difficult to locate the source of the problem.

This article describes an approach to performing regression tests for graphical output that overcomes these issues. The core idea is that the use of the diff utility on graphics files can be usefully augmented by the addition of a *visual* comparison of the graphics files. The ImageMagick compare utility (Still 2005, `http://www.imagemagick.org/`), is used to produce an *image* of the difference between two graphics files.

Much of this article will be concerned with describing the **graphicsQC** package, which implements this approach for R (R Development Core Team 2008a). In addition to performing the comparison between two graphics files, this package also provides facilities for generating large quantities of graphics files and for generating reports on the results of comparisons.

2. Quality control in R

This article describes a contribution to the quality control tools *for graphics* in the R system. This section provides the context for that contribution by describing the quality control tools that already exist in R.

R has a number of different facilities for quality control (Hornik 2002). For example, the core system has a number of make targets that run a large suite of software tests.

There are tests that run code just to make sure that it does not crash. One example of this approach involves running many different code examples in an attempt to "exercise" as much of the code base as possible. To this end, one test runs a large number of example code chunks from the system documentation. Another test calls functions within the core R system with a deliberately wide range of arguments to ensure that the functions can cope gracefully with a wide variety of user inputs.

The core quality control facilities in R also include a number of regression tests, where the results of a successful code run are compared with a set of control output and any differences are reported.

Both of the above testing facilities in R provide some coverage of graphics code: some of the code chunk examples from the system documentation exercise some of the graphics code in R; and there is a regression test aimed directly at the core graphics facilities.

The graphics regression test produces about 30 plots and checks for any differences compared to a set of control plots. The plots are produced in PostScript format and the diff utility is used to perform the comparison, so differences are reported in terms of changes in PostScript code.

One way to enhance the software tests for graphics is to produce *images* of differences, which will make it easier to diagnose the source of problems.

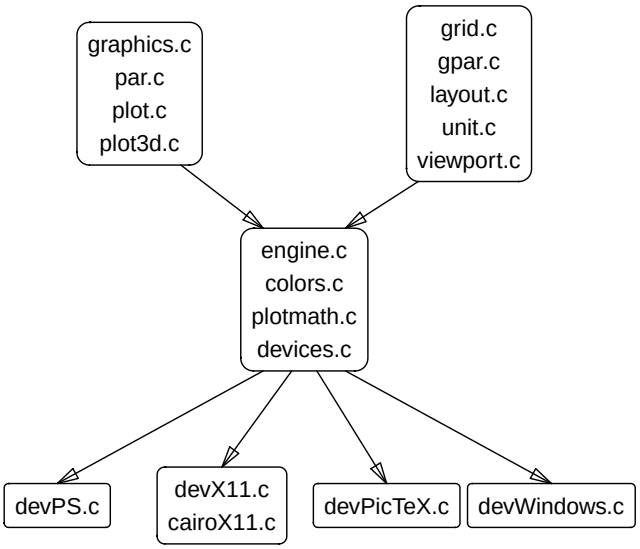

Figure 1: The organisation of graphics C code in the R system. The "traditional" graphics system (top-left) and the **grid** graphics system (top-right) call the graphics engine (centre), which calls graphics devices (bottom).

2.1. The structure of R graphics code

In order to further appreciate the limitations of the existing software tests with respect to the graphics code in R, this section will briefly outline how the R graphics code is structured.

As with most of the R system, the graphics code consists of a combination of R code and C code. This section is only concerned with the structure of the C code for graphics; the testing of R code will be discussed below in Section 2.2.

The C code for R graphics can be divided into three broad categories: there is a central *graphics engine*, two *graphics systems*, and a number of *graphics devices*. All graphics functions that the user sees will pass through either the "traditional" graphics system or the **grid** graphics system (Murrell 2005), both of these send calls to the graphics engine, and the graphics engine calls the graphics devices to produce output in different formats (see Figure 1).

Every graphics call passes through the graphics engine, so this C code will be exercised no matter what R graphics code gets run. Exercising the two graphics systems is also easy to arrange because it is simply a matter of ensuring that code is run from both traditional graphics packages and from packages built on top of the **grid** system. However, in order to exercise the code in different graphics devices, all graphics code must be produced in as many different output formats as possible.

Another way that the software tests in R can be expanded is to produce and compare graphics output in a wide range of formats rather than just as PostScript output.

2.2. Quality control for R packages

In addition to the software tests for the core system, R provides facilities for testing extension packages.

The `R CMD check` facility performs a wide variety of tests (R Development Core Team 2008b), including running all documentation from the package's example code and providing a simple mechanism for the package to specify regression tests for numerical (text output) results, *plus* tests for the existence of documentation, tests of consistency between code and documentation, and much more.

These tests are comprehensive and are an important contributor to the quality and reliability of the Comprehensive R Archive Network (CRAN) package repositories (R Foundation for Statistical Computing 2008). However, this mechanism does not provide support for regression tests of graphical output, so this is an area where quality control tools for *graphics* can contribute.

2.3. Quality assurance in R

In addition to providing *quality control* tools for both the core R system and for extension packages, as described above, the R project for statistical computing also provides *quality assurance* through a number of processes that *maintain* the ongoing quality of the R environment as a whole.

Checks of the core R system are performed almost every day, on a variety of platforms. The entire set of CRAN extension packages (1682 packages as of 2009-02-27) are also tested at least once a week on a variety of Linux systems. A CRAN package is checked on Windows and MacOS X systems whenever there is a new version of the package. (Hornik 2008; Ligges 2008). When a problem arises with an extension package, an email is sent to the package maintainers to notify them of the problem and to request a fix.

Ideally, new quality control tools for graphics should be able to be incorporated into these existing quality assurance processes.

2.4. Packages for quality control

Two extension packages provide some additional software testing facilities for R. The **Runit** package (Burger, Juenemann, and Koenig 2008) provides a *unit testing* framework for R, which is one of the pillars of the Extreme Programming paradigm (Beck and Andres 2004). This package extends the set of available tools for checking that code runs without crashing and for regression testing numeric results, but it does not provide any graphics-specific software testing tools.

Of particular relevance to this article is the experimental **graphicsQC** package (Murrell and Hornik 2003, `http://www.stat.auckland.ac.nz/~paul/`). This package produces plots in multiple formats, but is otherwise limited in its scope because it only runs code from examples in package documentation. It does produce an image of differences for raster formats, but this is simply a logical `xor` of the source images, so the result is a set of pixels that only occur in *one* of the source images and this can be extremely hard to interpret. Finally, it does not produce a convenient report of the results of comparisons. The **graphicsQC** package described in this article represents a major evolution of this earlier effort.

3. The graphicsQC package

This section describes the **graphicsQC** package for testing R code that produces graphical output.

This package can be divided into three parts: there are functions for producing files of graphical output based on user-specified R code; there is a function for comparing sets of graphics files; and there is a function for generating a report from the results of a set of comparisons.

Each of these parts is described in a separate section below.

3.1. Producing plots

The core function for producing plots is called `plotExpr()`.

The first argument to this function is a character vector of R expressions and the second argument is a character vector of graphics format names. For each expression and for each format, the `plotExpr()` function opens an appropriate graphics device and evaluates the expression.

By default, these expressions are evaluated to produce output on all supported graphics devices (currently, PDF, PostScript, PNG, and, on Windows, BMP). For example, the following code loads the **graphicsQC** package and (on Linux) produces a single plot (a scatterplot of the `pressure` data set) in PDF, PostScript, and PNG format.

```
> library(graphicsQC)
```

```
> plotExpr("plot(pressure)")
```

Additional arguments control how the resulting files are named (`prefix`) and where they are created (`path`). For example, the following code is similar to the previous example, but it keeps things more organized by creating the files in a separate directory, called `Demo1`, and it uses a specific prefix, `plot`, to name the files.

```
> demo1Result <- plotExpr("plot(pressure)", path="Demo1", prefix="plot")
```

The R expression in this example produces a single plot so three files are produced, one for each graphics file format.

```
> list.files("Demo1")
```

```
[1] "plot-1.pdf"   "plot-1.png"   "plot-1.ps"
[4] "plot-log.xml"
```

As the above result shows, as well as producing the graphics files, the `plotExpr()` function has created an XML *log file*, which contains a record of the graphics files that were produced. This log file will be discussed in detail shortly, but first there is more to say about the expressions that are passed to `plotExpr()` as the first argument.

The expressions that are given to `plotExpr()` can be arbitrary R code and this presents some complications that the function must be able to cope with:

How many plots? It is impossible to determine *a priori* how many pages of plots a piece of code will produce. The `plotExpr()` function simply generates a separate file for each page and numbers the files appropriately.

Zero plots: A special case of the previous problem is that it is possible for an R expression to produce no plots at all. This will still produce a graphics file, but the file will be "blank". The definition of blank depends on the file format and this can change between versions of R, so the **graphicsQC** package deliberately generates an empty file for each graphics format and `plotExpr()` deletes any graphics files that it produces that are identical to a blank file.

Overwriting files: Another problem that arises from not knowing how many files `plotExpr()` will produce is that it is difficult to predict whether `plotExpr()` will overwrite any existing files. The solution to this problem is that `plotExpr()` will refuse to produce files that start with the specified `prefix`. A further argument, `clear`, is provided which, if `TRUE`, means that all files with the relevant `prefix` are deleted so that `plotExpr()` can create new files without conflict.

Errors and warnings: The code given to `plotExpr()` may not work at all. For this reason, `plotExpr()` uses the `try()` mechanism to run the expressions. In the event of an error, the error message is recorded, but `plotExpr()` itself does not stop execution. Any warning messages are also recorded.

Dangerous code: Some R expressions have the potential to interfere with the workings of the `plotExpr()` function. For example, sending a `dev.off()` expression to `plotExpr()` will prematurely close a graphics device and prevent subsequent expressions from being recorded. The **graphicsQC** package does not attempt to protect itself from such abuse; the user is simply advised not to use expressions that include calls to functions that control graphics devices (e.g., `dev.set()`, `dev.prev()`, `dev.next()`), and any functions that start a new device, such as `postscript()`, `pdf()`, and `dev.new()`).

The following code demonstrates some of these additional details by extending the previous example to produce several files for each format.

The first argument to `plotExpr()` this time is a character vector containing two R expressions. The first expression produces a single plot (this time a scatterplot of pressure to the power of one eighth), but the second expression produces *four* plots (diagnostic plots from a linear model fit).

```
> demo2Result <-
      plotExpr(c("plot(pressure^.125 ~ temperature, data=pressure)",
                 "plot(lm(pressure^.125 ~ temperature, data=pressure))"),
               path="Demo2", prefix="plot")
```

The result is five files for each format, with the suffixes `-1`, `-2`, etc, for a total of 15 graphics files.

```
> list.files("Demo2")
```

```
 [1] "plot-1.pdf"   "plot-1.png"   "plot-1.ps"
 [4] "plot-2.pdf"   "plot-2.png"   "plot-2.ps"
 [7] "plot-3.pdf"   "plot-3.png"   "plot-3.ps"
[10] "plot-4.pdf"   "plot-4.png"   "plot-4.ps"
[13] "plot-5.pdf"   "plot-5.png"   "plot-5.ps"
[16] "plot-log.xml"
```

Again, in addition to the graphics files, there is also an XML log file.

XML log files

The `plotExpr()` function generates an XML log file to record information about the plots that were generated. The log file that was generated in the previous example is shown in Figure 2.

There are several important features in this log file. First of all, the file has an `<info>` element that records information about the version of R that was used to produce the files, plus time and date information (lines 3 to 15). Next, for each graphics format, there is a `<plots>` element containing a list of the files that were produced (e.g., lines 16 to 24). There is a separate `<plots>` element for each graphics format.

The information is recorded in a file like this so that the generation of graphics files can be treated as a separate step from comparing files or reporting on comparisons. For example, a typical approach may involve generating a set of control files using one version of R then a completely different R installation could be used to generate test code. The log files allow information to be shared between different R sessions and between different versions of R.

The choice of the eXtensible Markup Language (Bray, Paoli, Sperberg-McQueen, Maler, and Yergeau 2006) as the format for the log files means that it is easy for the log files to be processed using R functions outside the **graphicsQC** package or by using other software altogether.

The **XML** package (Temple Lang 2008b) is used to generate the XML log files.

Log objects

In addition to generating log files, `plotExpr()` also returns a `qcPlotExprResult` object that contains the same information as the log file about the plots that were produced. There is a print method for this sort of object, as shown below.

```
> demo2Result

plotExpr Result:
Call:      plotExpr(c("plot(pressure^.125 ~ temperature, data=pressure)",
               "plot(lm(pressure^.125 ~ temperature, data=pressure))"),
               path = "Demo2", prefix = "plot")
R version: R version 2.10.0 Under development (unstable) (2009-04-22 r48368)
Directory: /home/staff/paul/Research/Rstuff/QA/graphicsqc/pkg/inst/doc/Demo2
Filename:  plot-log.xml
Formats:
  pdf : Plots: plot-1.pdf, plot-2.pdf, plot-3.pdf, plot-4.pdf, plot-5.pdf
```

```
<?xml version="1.0"?>
<qcPlotExprResult>
 <info>
  <OS>unix</OS>
  <Rver>R version 2.10.0 Under development (unstable) (2009-04-22 r48368)</Rver>
  <date>2009-04-27 10:04:09</date>
  <call>
   <![CDATA[
   plotExpr(c("plot(pressure^.125 ~ temperature, data=pressure)",
    "plot(lm(pressure^.125 ~ temperature, data=pressure))"),
    path = "Demo2", prefix = "plot")   ]]>
  </call>
  <directory>/home/staff/paul/Research/Rstuff/QA/graphicsqc/pkg/inst/doc/Demo2</directory>
  <logFilename>plot-log.xml</logFilename>
 </info>
 <plots type="pdf">
  <warnings/>
  <error/>
  <plot>plot-1.pdf</plot>
  <plot>plot-2.pdf</plot>
  <plot>plot-3.pdf</plot>
  <plot>plot-4.pdf</plot>
  <plot>plot-5.pdf</plot>
 </plots>
 <plots type="png">
  <warnings/>
  <error/>
  <plot>plot-1.png</plot>
  <plot>plot-2.png</plot>
  <plot>plot-3.png</plot>
  <plot>plot-4.png</plot>
  <plot>plot-5.png</plot>
 </plots>
 <plots type="ps">
  <warnings/>
  <error/>
  <plot>plot-1.ps</plot>
  <plot>plot-2.ps</plot>
  <plot>plot-3.ps</plot>
  <plot>plot-4.ps</plot>
  <plot>plot-5.ps</plot>
 </plots>
</qcPlotExprResult>
```

Figure 2: An example of an XML log file that is generated by the `plotExpr()` function.

```
plot(pressure^.125 ~ temperature, data=pressure)
plot(lm(pressure^.125 ~ temperature, data=pressure))
```

Figure 3: The file `demo3.R`, which contains several R expressions.

```
png : Plots: plot-1.png, plot-2.png, plot-3.png, plot-4.png, plot-5.png
ps : Plots: plot-1.ps, plot-2.ps, plot-3.ps, plot-4.ps, plot-5.ps
```

However, the information that is displayed quickly becomes unreadable as the testing becomes more complex. The main purpose for a `qcPlotExprResult` is to allow software testing to occur within a single R session if desired; there will be an example of this later.

Running code from files

Two other convenience functions are built on top of `plotExpr()`. The first of these is called `plotFile()`, which takes the names of one or more files and runs the R code within the files.

All other arguments are the same as for `plotExpr()`, although the `prefix` argument defaults to the name of the file that is being run.

An example of the use of `plotFile()` is shown below. The file `demo3.R` contains the same expressions that were used in the last example (see Figure 3).

```
> demo3Result <- plotFile("demo3.R", path="Demo3")
```

The result is almost identical to the previous example; the R code generates two plots, in three different formats, and there is a log file, `demo3.R-log.xml` that records the files that were generated.

```
> list.files("Demo3")
```

```
 [1] "demo3.R-1.pdf"       "demo3.R-1.png"
 [3] "demo3.R-1.ps"        "demo3.R-2.pdf"
 [5] "demo3.R-2.png"       "demo3.R-2.ps"
 [7] "demo3.R-3.pdf"       "demo3.R-3.png"
 [9] "demo3.R-3.ps"        "demo3.R-4.pdf"
[11] "demo3.R-4.png"       "demo3.R-4.ps"
[13] "demo3.R-5.pdf"       "demo3.R-5.png"
[15] "demo3.R-5.ps"        "demo3.R-fileLog.xml"
[17] "demo3.R-log.xml"
```

However, there is also one additional file, called `demo3.R-fileLog.xml`. The contents of this file are shown in Figure 4.

This log file has an `<info>` element that contains information about the version of R that was used to generate the plots, but rather than storing information about the graphics files that were generated, it just has a `<qcPlotExprResult>` element (lines 14 to 16) that contains the location of the other log file, `demo3.R-log.xml`, which already has the detailed information about which graphics files were generated (just like the log file `plot-log.xml` from the previous example; see Figure 2).

```
<?xml version="1.0"?>
<qcPlotFileResult>
 <info>
  <OS>unix</OS>
  <Rver>R version 2.10.0 Under development (unstable) (2009-04-22 r48368)</Rver>
  <date>2009-04-27 10:04:10</date>
  <call>
   <![CDATA[
   plotFile("demo3.R", path = "Demo3")   ]]>
  </call>
  <directory>/home/staff/paul/Research/Rstuff/QA/graphicsqc/pkg/inst/doc/Demo3</directory>
  <logFilename>demo3.R-fileLog.xml</logFilename>
 </info>
 <qcPlotExprResult>
  /home/staff/paul/Research/Rstuff/QA/graphicsqc/pkg/inst/doc/Demo3/demo3.R-log.xml
 </qcPlotExprResult>
</qcPlotFileResult>
```

Figure 4: The file `demo3.R-fileLog.xml`, which records the results of a call to the `plotFile()` function. This file refers to the `plotExpr()` log file, `demo3.R-log.xml`, which contains detailed information about which graphics files were produced.

If several filenames are provided, `plotFile()` makes a separate call to `plotExpr()` for each filename, each of those calls generates a separate `-log.xml` file, and the overall `-fileLog.xml` files contains a pointer to each of the `-log.xml` files.

Running function examples

The other function that builds upon `plotExpr()` is called `plotFunction()`. This function takes the names of one or more functions as its first argument and runs the example code from the help page for the appropriate functions (by constructing a call to `plotExpr()`). Again, the other arguments are the same as for `plotExpr()`, but the `prefix` argument defaults to the function name(s).

The following code demonstrates the use of this function to produce plots from the example code on the help pages of the `boxplot()` and `hist()` functions.

```
> demo4Result <- plotFunction(c("boxplot", "hist"), path="Demo4")
```

The example code for `barplot()` produces six plots (in three formats) and there is a log file, `barplot-log.xml`, that records the names of these files and how they were produced. The example code for `hist()` produces three plots (in three formats) and a log file, `hist-log.xml`, records the names of these files and how they were produced. The overall log file, `barplot-funLog.xml`, just contains pointers to the individual log files (see lines 14 to 19 in Figure 5). The prefix for the overall log file from a call to `plotFunction()` is named after just the first function name (`boxplot` in this case) to avoid having a ridiculously long name for the overall log file.

```
> list.files("Demo4")

 [1] "boxplot-1.pdf"      "boxplot-1.png"
 [3] "boxplot-1.ps"       "boxplot-2.pdf"
```

```
<?xml version="1.0"?>
<qcPlotFunResult>
 <info>
  <OS>unix</OS>
  <Rver>R version 2.10.0 Under development (unstable) (2009-04-22 r48368)</Rver>
  <date>2009-04-27 10:04:11</date>
  <call>
   <![CDATA[
   plotFunction(c("boxplot", "hist"), path = "Demo4")   ]]>
  </call>
  <directory>/home/staff/paul/Research/Rstuff/QA/graphicsqc/pkg/inst/doc/Demo4</directory>
  <logFilename>boxplot-funLog.xml</logFilename>
 </info>
 <qcPlotExprResult>
  /home/staff/paul/Research/Rstuff/QA/graphicsqc/pkg/inst/doc/Demo4/boxplot-log.xml
 </qcPlotExprResult>
 <qcPlotExprResult>
  /home/staff/paul/Research/Rstuff/QA/graphicsqc/pkg/inst/doc/Demo4/hist-log.xml
 </qcPlotExprResult>
</qcPlotFunResult>
```

Figure 5: The file `boxplot-funLog.xml`, which records the results of a call to the `plotFunction()` function. This log file refers to two further log files, `boxplot-log.xml` and `hist-log.xml`, which are produced by calls to the `plotExpr()` function and contain detailed information about the graphics files that were produced.

```
 [5] "boxplot-2.png"      "boxplot-2.ps"
 [7] "boxplot-3.pdf"      "boxplot-3.png"
 [9] "boxplot-3.ps"       "boxplot-4.pdf"
[11] "boxplot-4.png"      "boxplot-4.ps"
[13] "boxplot-5.pdf"      "boxplot-5.png"
[15] "boxplot-5.ps"       "boxplot-6.pdf"
[17] "boxplot-6.png"      "boxplot-6.ps"
[19] "boxplot-funLog.xml" "boxplot-log.xml"
[21] "hist-1.pdf"         "hist-1.png"
[23] "hist-1.ps"          "hist-2.pdf"
[25] "hist-2.png"         "hist-2.ps"
[27] "hist-3.pdf"         "hist-3.png"
[29] "hist-3.ps"          "hist-log.xml"
```

In summary, the functions `plotExpr()`, `plotFile()`, and `plotFunction()` allow an arbitrary number of R expressions, R script files, or example code from R function help pages to be run. Each separate page of graphics output produces a separate graphics file and the resulting file names, plus details of the R session that produced them, are recorded in log files for later processing.

These functions can be used to generate both control and test output.

3.2. Comparing plots

Having generated a set of control plots and a set of test plots, the next step is to compare the sets of plots in order to detect any differences.

This step is performed by the `compare()` function.

The first two arguments to this function specify the set of test plots and the set of control plots that are to be compared. Each set of plots can be specified in one of three ways: as an R object, which is the result of a call to `plotExpr()`, `plotFile()`, or `plotFunction()`; as the full path to a log file; or as a directory that contains one or more log files.

As a simple example, the following code compares the results of the two simple `plotExpr()` examples from Section 3.1. The code specifies the set of test plots as an R object, `demo1Result`, and the set of control plots as a log file, `"Demo2/plot-log.xml"`.

```
> compareResult <- compare(demo1Result, "Demo2/plot-log.xml")
```

The `compare()` function will only work if it is comparing similar types of results, for example, a `plotExpr()` result against a `plotExpr()` result, but not a `plotExpr()` result against a `plotFunction()` result.

The call to the `compare()` function shown above generates several more files and these are placed by default in the directory of test files. This directory originally just contained three graphics files, `plot-1.pdf`, `plot-1.ps`, and `plot-1.png`, and one log file, `plot-log.xml`. The new contents of this directory, following the call to `compare()`, are shown below.

```
> list.files("Demo1/")

 [1] "plot-1.pdf"                    "plot-1-pdf+plot-1-pdf.diff"
 [3] "plot-1-pdf+plot-1-pdf.png"     "plot-1.png"
 [5] "plot-1-png+plot-1-png.png"     "plot-1.ps"
 [7] "plot-1-ps+plot-1-ps.diff"      "plot-1-ps+plot-1-ps.png"
 [9] "plot-log.xml"                  "plot+plot-compareExprLog.xml"
```

The original graphics files and the original log file have not been touched, but there are now several additional files. One of these is a *comparison* log file, `plot+plot-compareExprLog.xml`, which contains the results of the comparison. The contents of this file are shown in Figure 6.

As demonstrated by the size of this log file, which is from the most simple sort of comparison, a significant amount of information is recorded from the call to `compare()`. Following the order in the file, there is initially an `<info>` element (lines 3 to 15) that contains information about the R version used to generate this comparison, a `<testInfo>` element (lines 16 to 26) that points to the log file for the test set of plots (i.e., information about which plots were generated in the test set), and a `<controlInfo>` element (lines 27 to 39) with a pointer to the log file for the control set of plots.

The "meat" of the comparison result comes next, with several `<compare>` elements, one for each file format (e.g., lines 40 to 48). Within each of these are `<comparison>` elements containing the results of individual comparisons between graphics files. The first of these is reproduced below.

```
<?xml version="1.0"?>
<qcCompareExprResult>
 <info>
  <OS>unix</OS>
  <Rver>R version 2.10.0 Under development (unstable) (2009-04-22 r48368)</Rver>
  <date>Mon Apr 27 10:04:13 2009</date>
  <call>
   <![CDATA[
   compare(demo1Result, "Demo2/plot-log.xml")   ]]>
  </call>
  <path>/home/staff/paul/Research/Rstuff/QA/graphicsqc/pkg/inst/doc/Demo1</path>
  <testDirectory>/home/staff/paul/Research/Rstuff/QA/graphicsqc/pkg/inst/doc/Demo1</testDirectory>
  <controlDirectory>/home/staff/paul/Research/Rstuff/QA/graphicsqc/pkg/inst/doc/Demo2</controlDirectory>
  <logFilename>plot+plot-compareExprLog.xml</logFilename>
 </info>
 <testInfo>
  <OS>unix</OS>
  <Rver>R version 2.10.0 Under development (unstable) (2009-04-22 r48368)</Rver>
  <date>2009-04-27 10:04:09</date>
  <call>
   <![CDATA[
   plotExpr("plot(pressure)", path = "Demo1", prefix = "plot")   ]]>
  </call>
  <directory>/home/staff/paul/Research/Rstuff/QA/graphicsqc/pkg/inst/doc/Demo1</directory>
  <logFilename>plot-log.xml</logFilename>
 </testInfo>
 <controlInfo>
  <OS>unix</OS>
  <Rver>R version 2.10.0 Under development (unstable) (2009-04-22 r48368)</Rver>
  <date>2009-04-27 10:04:09</date>
  <call>
   <![CDATA[
   plotExpr(c("plot(pressure^.125 ~ temperature, data=pressure)",
    "plot(lm(pressure^.125 ~ temperature, data=pressure))"),
    path = "Demo2", prefix = "plot")   ]]>
  </call>
  <directory>/home/staff/paul/Research/Rstuff/QA/graphicsqc/pkg/inst/doc/Demo2</directory>
  <logFilename>plot-log.xml</logFilename>
 </controlInfo>
 <compare type="pdf">
  <comparison
   controlFile="/home/staff/paul/Research/Rstuff/QA/graphicsqc/pkg/inst/doc/Demo2/plot-1.pdf"
   testFile="/home/staff/paul/Research/Rstuff/QA/graphicsqc/pkg/inst/doc/Demo1/plot-1.pdf">
   <result>different</result>
   <diffFile>plot-1-pdf+plot-1-pdf.diff</diffFile>
   <diffPlot>plot-1-pdf+plot-1-pdf.png</diffPlot>
  </comparison>
 </compare>
 <compare type="png">
  <comparison
   controlFile="/home/staff/paul/Research/Rstuff/QA/graphicsqc/pkg/inst/doc/Demo2/plot-1.png"
   testFile="/home/staff/paul/Research/Rstuff/QA/graphicsqc/pkg/inst/doc/Demo1/plot-1.png">
   <result>different</result>
   <diffFile></diffFile>
   <diffPlot>plot-1-png+plot-1-png.png</diffPlot>
  </comparison>
 </compare>
 <compare type="ps">
  <comparison
   controlFile="/home/staff/paul/Research/Rstuff/QA/graphicsqc/pkg/inst/doc/Demo2/plot-1.ps"
   testFile="/home/staff/paul/Research/Rstuff/QA/graphicsqc/pkg/inst/doc/Demo1/plot-1.ps">
   <result>different</result>
   <diffFile>plot-1-ps+plot-1-ps.diff</diffFile>
   <diffPlot>plot-1-ps+plot-1-ps.png</diffPlot>
  </comparison>
 </compare>
 <unpaired>
  <test/>
  <control>
   <pdf>
    <plot>plot-2.pdf</plot>
    <plot>plot-3.pdf</plot>
    <plot>plot-4.pdf</plot>
    <plot>plot-5.pdf</plot>
   </pdf>
   <png>
    <plot>plot-2.png</plot>
    <plot>plot-3.png</plot>
    <plot>plot-4.png</plot>
    <plot>plot-5.png</plot>
   </png>
   <ps>
    <plot>plot-2.ps</plot>
    <plot>plot-3.ps</plot>
    <plot>plot-4.ps</plot>
    <plot>plot-5.ps</plot>
   </ps>
  </control>
 </unpaired>
</qcCompareExprResult>
```

Figure 6: The file `plot+plot-compareExprLog.xml`, which records the results of a call to the `compare()` function.

```
5,6c5,6
< /CreationDate (D:20090427100408)
< /ModDate (D:20090427100408)
---
> /CreationDate (D:20090427100409)
> /ModDate (D:20090427100409)
43c43
< /F1 1 Tf 1 Tr 7.48 0 0 7.48 92.77 84.61 Tm (l) Tj 0 Tr
---
> /F1 1 Tf 1 Tr 7.48 0 0 7.48 92.77 99.77 Tm (l) Tj 0 Tr
46c46
< /F1 1 Tf 1 Tr 7.48 0 0 7.48 114.10 84.61 Tm (l) Tj 0 Tr
---
> /F1 1 Tf 1 Tr 7.48 0 0 7.48 114.10 116.62 Tm (l) Tj 0 Tr
49c49
< /F1 1 Tf 1 Tr 7.48 0 0 7.48 135.44 84.62 Tm (l) Tj 0 Tr
---
> /F1 1 Tf 1 Tr 7.48 0 0 7.48 135.44 137.21 Tm (l) Tj 0 Tr
52c52
< /F1 1 Tf 1 Tr 7.48 0 0 7.48 156.77 84.64 Tm (l) Tj 0 Tr
```

Figure 7: The first few lines of the file `plot-1-pdf+plot-1-pdf.diff`, which records the difference in the raw PDF code between two graphics files.

```
<comparison

 controlFile="/home/staff/paul/Research/Rstuff/QA/graphicsqc/pkg/inst/doc/Demo2/plot-1.p

 testFile="/home/staff/paul/Research/Rstuff/QA/graphicsqc/pkg/inst/doc/Demo1/plot-1.pdf"
 <result>different</result>
 <diffFile>plot-1-pdf+plot-1-pdf.diff</diffFile>
 <diffPlot>plot-1-pdf+plot-1-pdf.png</diffPlot>
</comparison>
```

This `<comparison>` element records which graphics files were compared via the attributes `controlFile` and `testFile`, and the result of the comparison is a `<result>` element. In this case, there is a difference between the files, which means that the `compare()` function has generated a file to show the difference in the underlying text (PDF) code, `plot-1-pdf+plot-1-pdf.diff`, *and* it has generated an *image* of the difference, `plot-1-pdf+plot-1-pdf.png`. The names of these files are recorded in `<diffFile>` and `<diffPlot>` elements in the log file and they can also be seen in the file listing of the `Demo1` directory that was shown above.

The first few lines of the file `plot-1-pdf+plot-1-pdf.diff` are shown in Figure 7 and the image `plot-1-pdf+plot-1-pdf.png` is shown in Figure 8.

The `.diff` file shows the low-level detail of the difference in terms of PDF code and the `.png` file shows a high-level view of the difference. The latter makes it very easy to identify and understand what has actually changed in the image, and the former can be useful for tracking down exactly which piece of C code is reponsible for the difference. Together, they provide very powerful debugging information. In this example, the difference has been deliberately introduced—the plots differ in terms of the y-values (one is a plot of the raw pressure values

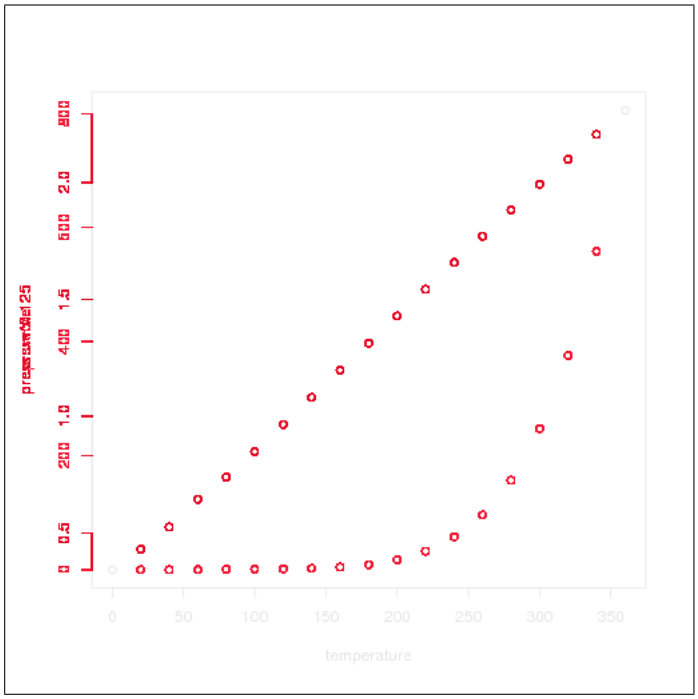

Figure 8: An image of the difference between two graphics files. Areas where the original files are the same are drawn in light grey and areas of difference are in red. This complements the difference in the raw PDF code that is shown in Figure 7.

and one is a plot of the transformed pressure values).

Going back to the log file `plot+plot-compareExprLog.xml` (Figure 6), the results of further comparisons are recorded as well, with more `.diff` files and `.png` files. One detail to note is that the comparison of PNG files does *not* produce a `.diff` file (line 54) because PNG files are binary rather than text-based files.

There is also one final section in the log file for the comparison, which records any files that were "unpaired". In this comparison, the test set of plots consisted of three plots, but the control set consisted of fifteen plots. Consequently, twelve of the control plots have nothing to be compared to, so these graphics files are recorded in the log file within an `<unpaired>` element.

In addition to the log file, the results of a call to the `compare()` function are also returned as an R object. In the above example, this has been assigned to the symbol `compareResult`. We will make use of this R object later.

Auto-detecting log files

In the above example of the `compare()` function, the test set of plots were specified using an R object—the result of a call to `plotExpr()`—and the control set of plots were specified as the name of a log file.

It is also possible to specify just a directory that contains a set of plots. In this case, the `compare()` function attempts to find the highest-level log file within that directory. If there is only a log file from a call to `plotExpr()` that will be used, but if there is a log file from a call to `plotFile()` or `plotFunction()`, that will be used instead (such a log file will point to one or more log files from one or more calls to `plotExpr()`).

In summary, the `compare()` function is used to perform a comparison between two sets of plots. Any differences are recorded as `.diff` files (for text-based file formats) and visually as `.png` files, and the full set of results for the comparisons is recorded in a log file.

3.3. Reporting on comparisons

The previous section described how to perform a comparison between two sets of plots and the set of difference files and log files that are generated by the comparison. It should be clear that a large amount of information is recorded for even a simple comparison and sifting through the XML within a log file to determine the result of a comparison is not very convenient.

This section describes the function `writeReport()`, which is provided to present the results of a call to the `compare()` function in a much more convenient format, as an HTML report.

The `writeReport()` function has a single important argument, which specifies the information from a comparison, either as an R object that was produced by a call to `compare()`, or the name of a log file, or the name of a directory that contains a log file. The following code produces a report on the comparison example from the previous section by specifying the R object that resulted from the call to `compare()`.

```
> writeReport(compareResult)

[1] "/home/staff/paul/Research/Rstuff/QA/graphicsqc/pkg/inst/doc/Demo1/plot+plot-compareExprLog.htm
```

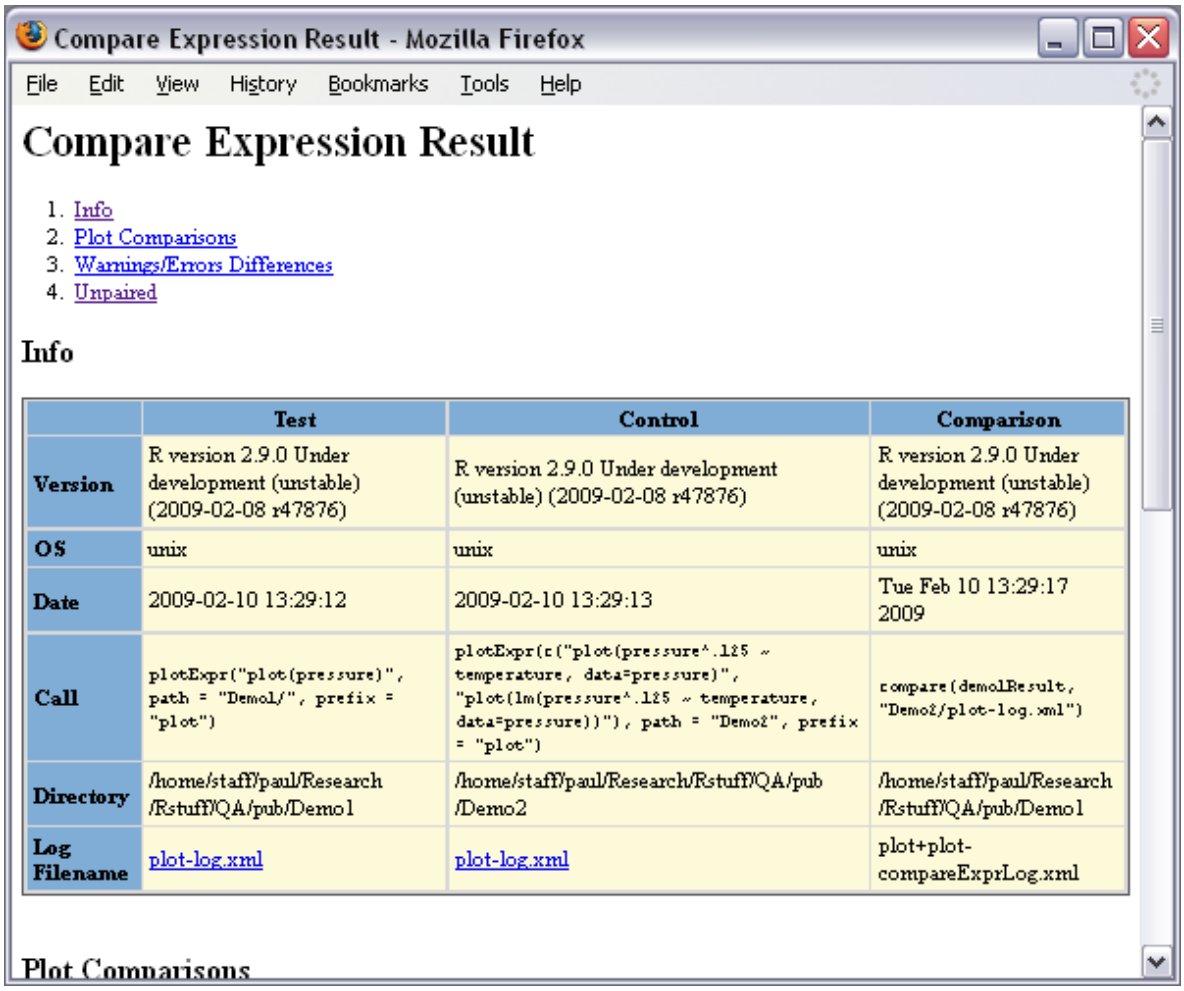

Figure 9: The top part of an HTML report produced by `writeReport()`.

The top of the resulting HTML report is shown in Figure 9 and the bottom of the report is shown in Figure 10.

The basic structure of this report mirrors the structure of the original log file; there is a section titled "Info" with information about how the sets of plots were generated and about how the comparison was performed, then there is a section titled "Plot Comparisons" with information about each individual file comparison, and finally several sections containing extra details about any graphics files that were unpaired, and a comparison of warning and error messages.

This report is just a more digestible form of the log file, but it is *much* more convenient. It also has the benefit of providing links to the information that it records. For example, in the section on plot differences (Figure 10), it is possible to click on links to view the original graphics files (e.g., the link labelled `plot-1.pdf`), the `.diff` files (e.g., the link labelled `plot-1-pdf+plot-1-pdf.diff`), and the `.png` images of the differences (e.g., the link labelled `plot-1-pdf+plot-1-pdf.png`).

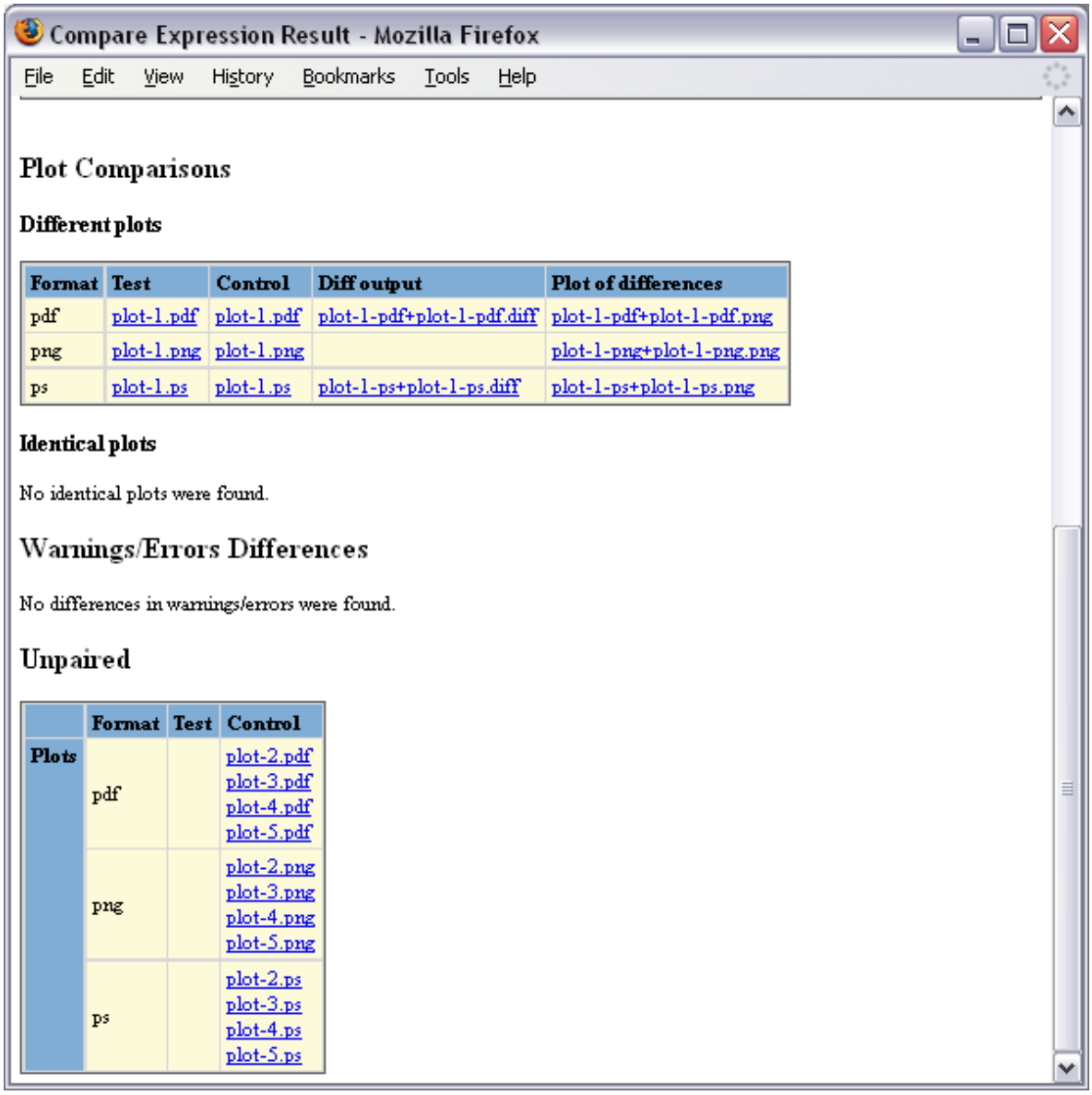

Figure 10: The bottom part of an HTML report produced by `writeReport()`.

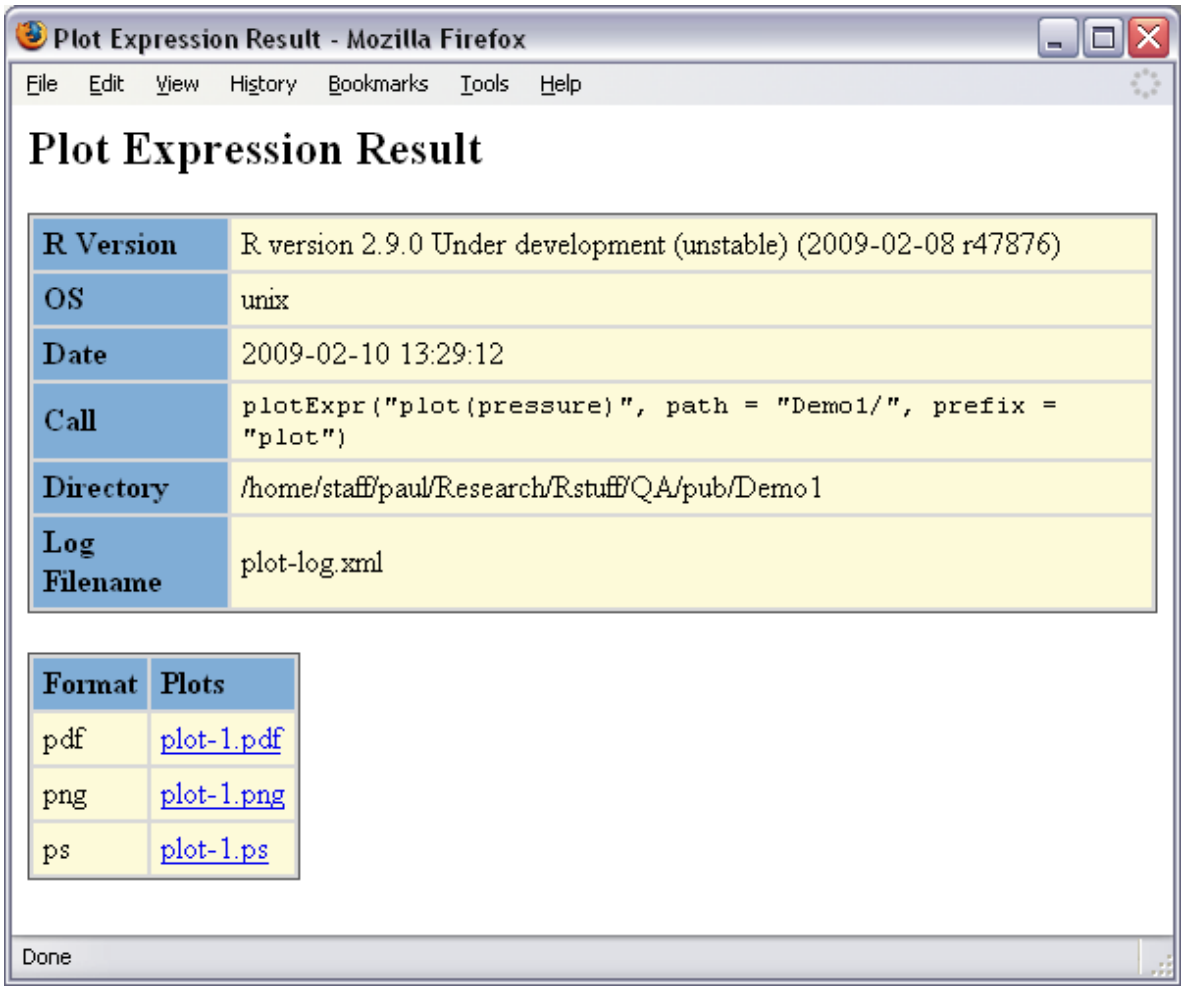

Figure 11: An HTML report, produced by `writeReport()`, on a set of plots that were generated by a call to `plotExpr()`.

Furthermore, in addition to generating an HTML version of the log file generated by `compare()`, the `writeReport()` function generates an HTML version of the log files that recorded the original sets of plots—the log files that were generated by `plotExpr()`. In the "Info" section of the report (Figure 9), it is possible to click on a link labelled `plot-log.xml` to view a report of the files that were generated by `plotExpr()`. This report for the control set of plots is shown in Figure 11.

XSLT

The generation of HTML report files from the XML log files is performed using eXtensible Stylesheet Language Transformations (Clark 1999).

The **graphicsQC** package provides a default set of XSLT templates for transforming the information in the XML log files into HTML elements, but these can be modified or replaced to produce a different style of report. The `writeReport()` function has an `xslStyleSheets`

argument that allows new XSLT templates to be specified.

The **Sxslt** package (Temple Lang 2008a) is used to apply the XSLT templates that transform the XML log files into HTML report files.

In summary, the `writeReport()` function generates HTML files based on the results of a call to `compare()`. This set of HTML files allows for an efficient and detailed exploration of the results of a comparison.

4. Applications and examples

The examples up to this point have been very simple, with deliberate differences between plots, in order to clearly demonstrate how the functions within the **graphicsQC** package work.

This section demonstrates a realistic application of the package.

The quality control scenario is that a change was made to the C code in the core R graphics system to account for anisotropy (pixels that are not square). This change was expected to have an effect on the output of some specific R functions on some raster formats, for example PNG, but no effect on vector formats such as PDF and PostScript.

The software testing serves two purposes: first of all, it makes it possible to observe whether the code change has any of the anticipated effects on raster output; secondly, it makes it possible to check that the code change has not had any unintended consequences.

The first step is to generate a set of plots using the version of R immediately prior to the code change (this was SVN revision 44416). This specific version of R was checked out of the SVN repository and built, the **XML**, **Sxslt**, and **graphicsQC** packages were installed, then the following code was run.

```
> plotFunction(ls("package:grid"),
               path="R44416/QC")
```

The purpose of this call is to run all example code for all functions in the **grid** graphics package. The resulting graphics files and log files are stored in the directory `R44416/QC`.

The next step is to generate a set of plots using the version of R immediately after the code change (revision 44417). Having checked out and built that version, and having installed the required packages, the following code was run.

```
> plotFunction(ls("package:grid"),
               path="R44417/QC")
```

Again, the result is a set of plots from running example code for functions in the **grid** package, this time storing the files in the directory `R44417/QC`.

The comparison of the two sets of plots is performed with the following code (run in either version of R).

```
> compare("R44417/QC", "R44416/QC")
```

The `compare()` function automatically detects the appropriate log files and compares all plots, producing files of differences where appropriate and generating log files to record the results.

The final step is to generate an HTML report so that the results of the comparison can be viewed. This is achieved using the following code.

```
> writeReport("R44417/QC")
```

The top of the resulting report is shown in Figure 12.

The first important thing to note about this report is that the table of contents at the top includes a summary of the number of files that were different (9 files were different and these files were generated from the example code of 6 different **grid** functions), the number of files that were identical (372 files, representing the output of example code from 67 different functions), and the number of functions that produced no graphical output at all (86 functions).

Figure 13 shows the "Different plots" section from the report. This reveals which functions produced different plots, along with links to the original plots and links to pictures of the differences. The difference for the `arcCurvature` output is shown in Figure 14. This shows that some lines are drawn in slightly different locations, which is exactly the effect that was expected from the anisotropy code change.

It is worth noting that, due to the sheer number of plots that need to be produced and compared, this is a good demonstration of the usefulness of automating this sort of check. It is also a good demonstration of the usefulness of producing an *image* of the difference—this tiny change would be impossible to detect by eye and, even having determined that the images were different, it would be hard to pinpoint the exact nature of the change.

It is also worth noting that this report only shows that there is a difference. It does not make any statements about whether the control output or the test output represents the "correct" output (it is possible that neither does). The results of **graphicsQC** comparisons provide support for diagnosing problems, but an expert viewer is required to determine the meaning of any differences.

Another important point about the table of "Different plots" (Figure 13) is that it provides links to more detailed information about the comparison for each **grid** function. For example, there is a link, labeled `arcCurvature+arcCurvature`, to a separate HTML report on the comparison just for the `arcCurvature()` example plots. This report is shown in Figure 15.

The usefulness of this lower level report is that it clearly shows that there were differences in the graphics output only for the PNG format. The output of the `arcCurvature()` function was identical for PostScript and PDF formats.

It is also possible to drill down even further to reports that detail how the test and control sets of plots were generated.

Going back to Figure 13, not all of the differences that are reported here are ones that were expected. The differences were expected to be seen in the output of functions that draw special curves called X-splines (Blanc and Schlick 1995). This accounts for the differences reported for the `arcCurvature()` function, the `curveGrob()` and `grid.curve()` functions, and the `grid.xspline()` and (not visible) `xsplineGrob()` functions. However, a difference in the output for the `grid.gedit()` function is a surprise. Figure 16 shows the image of the difference in this output.

This output does not involve any X-splines; it is just straight lines and text. Furthermore the difference is in the positioning of the text.

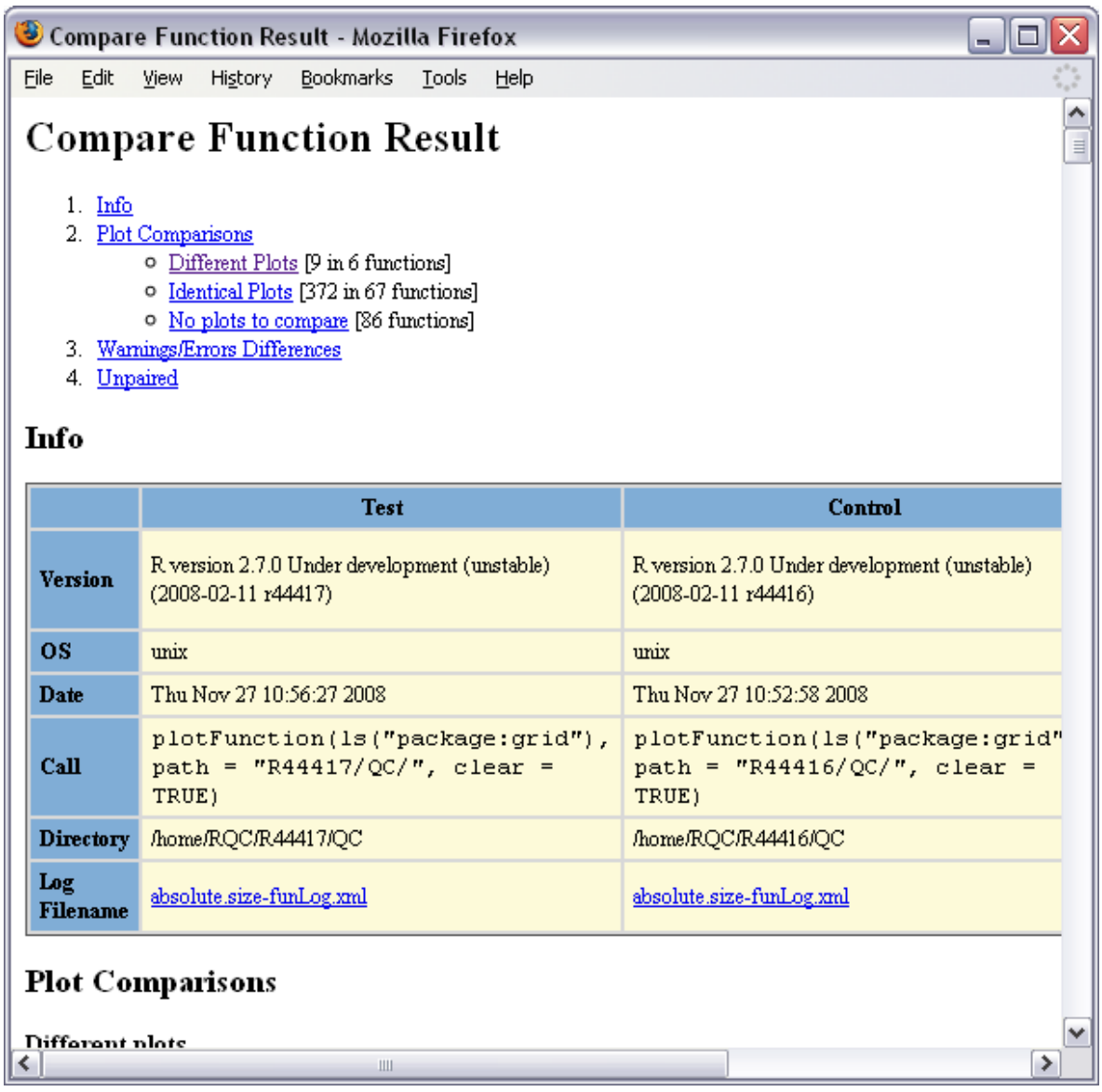

Figure 12: An HTML report, produced by `writeReport()`, to check for differences in the output of **grid** functions following a change in the C code of the R graphics engine.

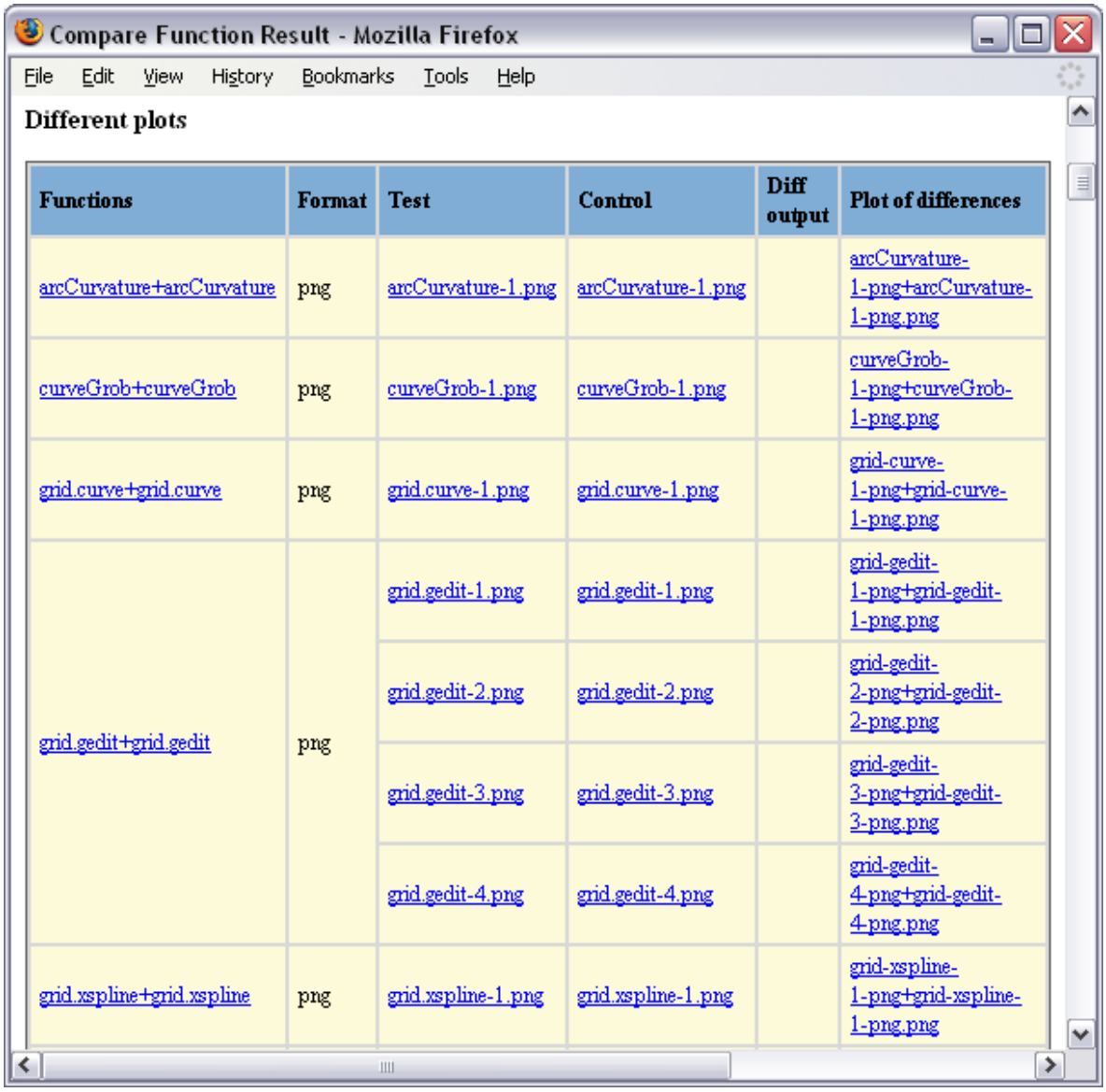

Figure 13: An HTML report, produced by `writeReport()`, to check for differences in the output of **grid** functions following a change in the C code of the R graphics engine. This shows the "Different plots" section of the report.

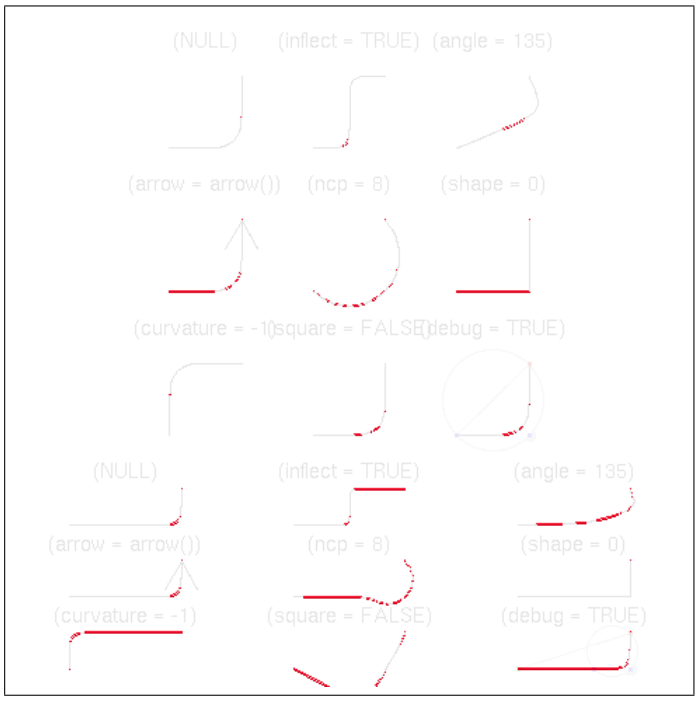

Figure 14: An image of the difference in output from the **grid** function `arcCurvature()` following a change in the C code of the R graphics engine. Differences are colored red.

It turns out that this is the result of a small and hard-to-find bug in the core R graphics code that occurs essentially at random. However, the details of this bug are not important to this article.

What is important is that this is a vivid demonstration of the value of this sort of quality control tool. This bug had gone unnoticed for 9 months precisely because the sort of tests that the **graphicsQC** package provides were not conducted. One of the important purposes of running these sorts of software tests is to find changes like this that were not anticipated.

5. Limitations

This section discusses some of the known limitations of the **graphicsQC** package.

One major issue is the fact that **graphicsQC** depends on the **XML** and **Sxslt** packages. This is an issue because some people have reported problems with the **XML** package on the Mac OS platform and the **Sxslt** package is only known to work on Linux systems. These problems are perhaps less serious than they would be for other R packages because it is anticipated that **graphicsQC** will be used mostly by package developers and it is generally assumed that the majority of package development occurs on Linux systems. Furthermore, the **Sxslt** package is only required for generating reports via the `writeReport()` function, so the other graphics quality control tools can still be used on other platforms.

The **graphicsQC** package also depends on the ImageMagick software library for generating images of differences between graphics files. This imposes a burden on the user on non-Linux

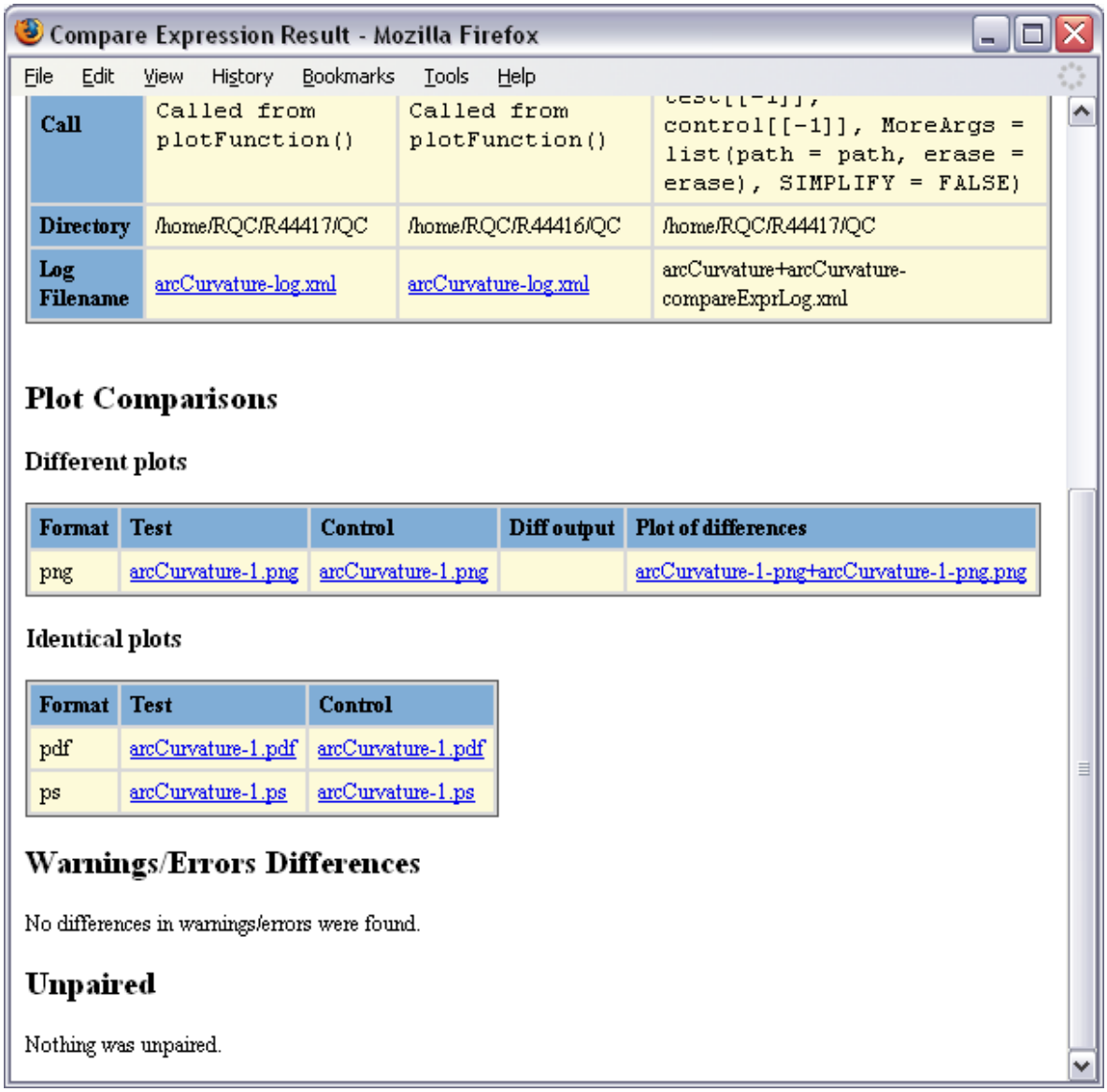

Figure 15: An HTML report, produced by `writeReport()`, to check for differences in the output of the **grid** function `arcCurvature()`, following a change in the C code of the R graphics engine. This shows the "Different plots" section of the report. This report is obtained by "drilling down" from the main report shown in Figure 13.

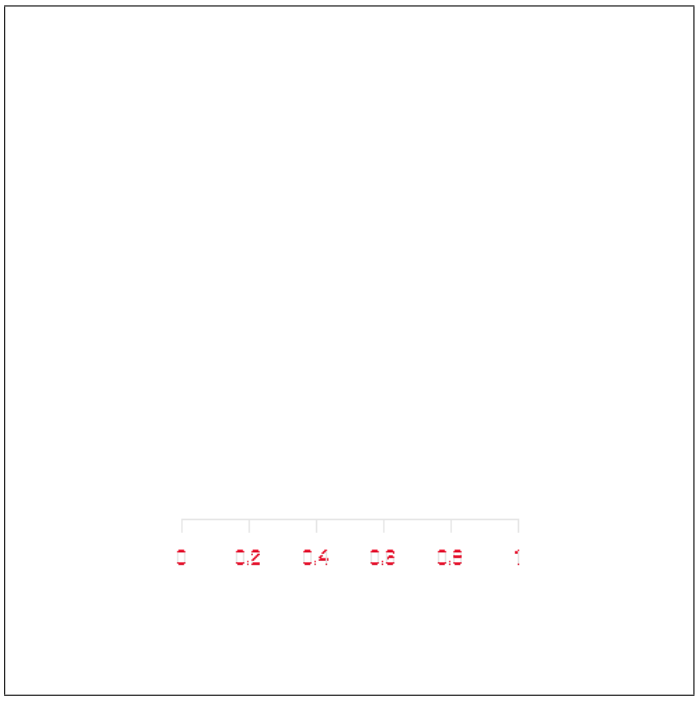

Figure 16: An image of the difference in output from the **grid** function `grid.gedit()` following a change in the C code of the R graphics engine. Differences are colored red.

systems, who will have to install the additional software, but ImageMagick is available for all major platforms.

The graphics testing tools provided by the **graphicsQC** package are not part of the standard `R CMD check` suite of tests. However, they can be easily incorporated via a file in the package `tests` directory for a package. All that is needed is a file of R code that generates test plots—Figure 17 shows an example—plus a set of control files to compare the test plots to. The control files can be generated once and included as part of the package directory structure. This approach has been succesfully used to automate graphics quality control checks for the package **gridDiagram** (http://www.stat.auckland.ac.nz/~paul/R/Diagram/gridDiagram_0.2.tar.gz).

The limitation is that this approach is only appropriate for private testing of a package. This would not necessarily work for a package submitted to CRAN because of the dependencies on the **XML** and **Sxslt** package, *plus* the fact that this sort of check may generate large numbers of files and take a long time to run, which may take too much of a toll on the resources used for daily testing of CRAN packages. In practice, the authors have so far mostly used the package for one-off testing, with control files generated using one version of R or a one version of a particular package (prior to a bug fix) and test files generated using a different version of R or a different version of the package (which includes the bug fix).

One final limitation of the software testing tools provided by the **graphicsQC** package is that they do not allow *interactive* graphics devices to be tested. For example, not all code that supports X11 or Windows on-screen windows will be exercised by **graphicsQC** tests. Also, functions that require user input, such as `locator()` and `identify()` cannot be tested with

```
library(gridDiagram)
library(graphicsQC)

# Control plots generated by:
#
# plotFile(c(system.file("demo", "books.R", package="gridDiagram"),
#            system.file("demo", "swimmers.R", package="gridDiagram")),
#          path="ControlPlots")

plotFile(c(system.file("demo", "books.R", package="gridDiagram"),
           system.file("demo", "swimmers.R", package="gridDiagram")),
         path="TestPlots")
compare("TestPlots", "ControlPlots")
writeReport("TestPlots")
```

Figure 17: An example of a file that could be placed in the `tests` sub-directory of a package in order to run **graphicsQC** tests as part of `R CMD check`. This code checks the graphics output from R code in two files from the `demo` sub-directory of a package called **gridDiagram**.

the **graphicsQC** package.

6. Availability

The **graphicsQC** package is available from CRAN. This article describes version 1.0 of the package.

A development version of the package is available from R-Forge (R-Forge Development and Administration Team 2008, http://R-Forge.R-project.org/) and the R-Forge site also has a link to the complete set of files from the report that was described in Section 4. The random nature of the graphics bug that was discovered in that example means that some of the differences between graphics files that are recorded in the report on R-Forge differ from what is shown in Section 4 of this article.

7. Conclusion

The **graphicsQC** package provides quality control tools for graphics software testing.

The functions `plotExpr()`, `plotFunction()`, and `plotFile()` can be used to generate sets of graphics files from explicit R code; from the example code of help pages for specified function names; or from the names of files that contain R code.

The `compare()` function compares a set of test graphics files with a set of control graphics files and, where there are differences, it generates new files containing both text-based and visual representations of the differences.

The `writeReport()` function produces an HTML report that conveniently displays the results of a graphics software test.

The **graphicsQC** package is being used as a quality control tool in the development of the core R graphics system and can be easily incorporated into the testing of extension packages for R.

A possible future development of the package could include a function that generates graphics files based simply on the name of a package. This would be a simple wrapper around `plotFunction()` and `plotFile()` that would run all examples from all help files in the package *plus* any files of R code in the `demo` and `tests` directories of the package.

It would also be possible to write a single overall function that performed a complete software test, including producing both test and control sets of plots by running separate R versions via `system()` calls, followed by calls to `compare()` and `writeReport()`.

Finally, the range of supported graphics formats could also be extended.

Further experience with using the package will dictate whether these enhancements are necessary or worthwhile.

Acknowledgements

Yun Hong provided proof-of-concept code for some of the ideas in this package during a Faculty of Science Summer Scholarship 2005–2006 at the University of Auckland.

The valuable suggestions and criticisms from the assistant editor and the anonymous reviewers are greatly appreciated.

References

Beck K, Andres C (2004). *Extreme Programming Explained: Embrace Change*. Addison-Wesley Professional, 2nd edition. ISBN 9780321278654.

Blanc C, Schlick C (1995). "X-splines: A Spline Model Designed for the End-User." In "SIGGRAPH '95: Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques," pp. 377–386. ACM, New York, NY, USA. ISBN 0-89791-701-4. doi:http://doi.acm.org/10.1145/218380.218488.

Bray T, Paoli J, Sperberg-McQueen CM, Maler E, Yergeau F (2006). *Extensible Markup Language (XML) 1.0*. World Wide Web Consortium (W3C). URL http://www.w3.org/TR/2006/REC-xml-20060816/.

Burger M, Juenemann K, Koenig T (2008). ***RUnit**: R Unit Test Framework*. R package version 0.4.19, URL https://sourceforge.net/projects/runit/.

Clark J (1999). *XSL Transformations (XSLT)*. World Wide Web Consortium (W3C). URL http://www.w3.org/TR/xslt/.

Horch JW (2003). *Practical Guide to Software Quality Management*. Artech House Publishers, 2nd edition. ISBN 9781580535274.

Hornik K (2002). "Tools and Strategies for Managing Software Library Repositories." In "Statistics in an Era of Technological Change, Proceedings of the 2002 Joint Statistical Meetings," pp. 1490–1493.

Hornik K (2008). "CRAN Package Check Results." http://cran.r-project.org/web/checks/check_summary.html.

Hower R (2009). "Software QA and Testing Frequently-Asked-Questions." http://www.softwareqatest.com/qatfaq1.html.

Ligges U (2008). "CRAN Windows Binaries' Package Check." http://cran.r-project.org/bin/windows/contrib/checkSummaryWin.html.

McKenzie D, Eggert P, Stallman R (2002). *Comparing and Merging Files*. URL http://www.gnu.org/software/diffutils/manual/diff.html.

Murrell P (2005). *R Graphics*. Chapman & Hall/CRC, Boca Raton, FL. ISBN 1-584-88486-X, URL http://www.stat.auckland.ac.nz/~paul/RGraphics/rgraphics.html.

Murrell P, Hornik K (2003). "Quality Assurance for Graphics in R." In "Proceedings of the 3rd International Workshop on Distributed Statistical Computing," pp. 20–22. Vienna, Austria. ISSN 1609-395X. Edited by Hornik K., Leisch, F. & Zeileis, A.

R Development Core Team (2008a). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, URL http://www.R-project.org.

R Development Core Team (2008b). *Writing R Extensions*. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-11-9, URL http://cran.stat.auckland.ac.nz/doc/manuals/R-exts.html.

R-Forge Development and Administration Team (2008). *R-Forge User's Manual*. URL http://R-Forge.R-Project.org/R-Forge_Manual.pdf.

R Foundation for Statistical Computing (2008). *Comprehensive R Archive Network—CRAN*. URL http://cran.r-project.org/.

Still M (2005). *The Definitive Guide to ImageMagick*. Apress. ISBN 9781590595909.

Temple Lang D (2008a). ***Sxslt**: R extension for libxslt*. R package version 0.8-0, URL http://www.omegahat.org/Sxslt.

Temple Lang D (2008b). ***XML**: Tools for Parsing and Generating XML within R and S-Plus*. R package version 1.95-3.

Affiliation:

Paul Murrell
Department of Statistics
The University of Auckland
Private Bag 92019
Auckland
New Zealand
64 9 3737599 x85392
E-mail: paul@stat.auckland.ac.nz
URL: http://www.stat.auckland.ac.nz/~paul/