The **graphicsqc** package
Quality Control for Graphics in R

Stephen Gardiner

Supervised by Dr. Paul Murrell

Department Of Statistics
University of Auckland
BSc(Hons) Project

November 2008

Abstract

The major achievements of this research has been to identify and create a new Quality Control function for graphics in R which produces results of differences including pictures highlighting the differences (when available), is capable of testing any arbitrary code, and has convenience functions for plotting other functions and files, which is easily extendable to plot entire packages.

Contents

Chapter 1

Introduction

There has been much work into the concept of Quality Control for software. There are currently tools in R which ensure that R code will run without catastrophic failure, but fewer tools to check that the output from the code is correct, especially graphics output. Thus, the aim of this research has been to establish new methods of doing regression testing for graphics output in R.

1.1 Measuring software quality: QC and Regression testing

Quality Control (QC), or *testing*, is used to ensure quality of software. The type of testing used to assess whether graphics output has changed or not is called *regression testing*. This type of testing involves creating *control output* before a change is made to the code, then creating *test output* after the change is made, and finally comparing the outputs to see if the change in the code affected the graphics output. This method is useful to identify changes in graphics output, but in order to determine whether graphics output is correct or not, an 'expert observer' is required to initially validate the control output. Once the control output has been validated it becomes the *model output*, and then by using regression testing we can validate the test output as being correct [Ihaka and Gentleman, 1996] and [R F, 2008] and [Murrell and Hornik, 2003].

1.2 Validating graphics output

1.3 Existing QC functions for graphics in R

An early attempt at graphics testing for R is a package called graphicsQC written by Dr. Paul Murrell (2003). It is very limited in usability and functionality; it can only run on the Unix platform, can only do regression testing on the example code in given functions in R, and only identifies which plots had differences with no further information about the differences. Further research is therefore required in order to extend the package to be platform independent, and to identify new ways to test R

code such that any arbitrary code can be tested, and more information about the differences to be reported.

The goal of this research was to identify new interfaces and methods of doing regression testing for graphics output in R.

Chapter 2

Plotting Arbitrary Expressions

2.1 Plotting expressions

In order to test the correctness of output, it follows that output must have initially been produced. For this task, the function `plotExpr` was created. Using error handling functions in R, `plotExpr` was designed to be able to produce the output from any abitrary code passed to it as an argument, with the ability to note and safely recover from any warnings and to also correctly stop on any errors. As arbitrary code is accepted by the function it can be easily seen that plotting the code from files, functions, or even packages in R is possible, and thus the convenience functions plotFile, and plotFun were also created.

The output produced from a plotting function would simply be a series of plots which the code might (or might not) have produced. In order to store these plots, a file format (or file formats) must be chosen and given as an argument to the function. Using many file formats is a distinct advantage in that all graphics formats can be tested against themselves to help identify whether changes in graphics output are due to the underlying code used to produce the respective graphics, or whether there is a problem or change with a specific graphics driver.
For example, we might wish to create plots of the `plot` function. In R:

```
> plotFunction("plot", filetype=c("png", "pdf"), path="plotControl")
```

Then to view the files created in the directory:

```
$ ls plotControl/
plot-1.pdf  plot-2.pdf  plot-3.pdf  plot-4.pdf  plot-funLog.xml
plot-1.png  plot-2.png  plot-3.png  plot-4.png  plot-log.xml
```

As we can see, files from the `plot` function's example code have been produced, where each file has the prefix of the function name, and a numbered suffix to identify it. Also, as specified, pdf and png files have been produced, along with two xml files; `plot-log.xml` and `plot-funLog.xml`. The creation of the XML files is described in the next section.

2.2 Plotting files and functions

2.3 Logs of plots created

2.3.1 XML logging

As we will want to compare the plots to previously produced plots, we will need to know information about each set of plots which has been produced. Information about the plots produced are stored in an e**X**tensible **M**arkup **L**anguage file. The main advantages of storing this information in XML is that it is self-documenting and also platform independent. It is relatively human-legible, and since it is self-documenting, the process at this point can be taken out of R and the plots and corresponding information used by other programs if desired.
Continuing the example in 3.1, the log file produced is shown:

```
$ cat plotControl/plot-log.xml
<?xml version="1.0"?>
<qcPlotExprResult>
 <info>
  <OS>unix</OS>
  <Rver>2.6.1</Rver>
  <date>Sat Feb 23 11:32:49 2008</date>
  <call>
   <![CDATA[
   plotFunction("plot", filetype = c("png", "pdf"), path = "plotControl")   ]]>
  </call>
  <filetype>png</filetype>
  <filetype>pdf</filetype>
  <directory>/home/stephen/plotControl</directory>
 </info>
 <warnings/>
 <errors/>
 <filenames>
  <filename>plot-1.pdf</filename>
  <filename>plot-1.png</filename>
  <filename>plot-2.pdf</filename>
  <filename>plot-2.png</filename>
  <filename>plot-3.pdf</filename>
  <filename>plot-3.png</filename>
  <filename>plot-4.pdf</filename>
  <filename>plot-4.png</filename>
 </filenames>
</qcPlotExprResult>
```

As XML is self-documenting, the log file produced does not require much explanation. The `funLog` file is simply a pointer to all of the log files that the function might have produced for ease of comparison later.

2.4 Example: Plotting the `grid` function

Chapter 3

Comparing plots

The next step in the process is to compare a set of plots to another; typically between two different versions of R to establish if the graphics outputs have changed, but could also be used to detect whether changes made to the graphics system or to specific functions are different to previously known correct control group outputs.

GNU diff is used.

To detect the differences, the utility `diff` is used, which is assumed on all supported platforms according to the R Coding Standards (Writing R Extensions). An extension to this has been to use the ImageMagick software to also create plots of the differences when available. A drawback to this is that the ImageMagick software must first be installed, but it is open-source, free, and is readily available for many operating systems or even from source. While it is a distinct advantage to use the ImageMagick software, it is also not a requirement as the differences will still be detected by `diff`, but plots of the differences will not be created.
For example, to create two plots which will be different from each other:

```
> control <- plotExpr("hist(rep(1:10,1:10), breaks=6)", "png", "histControl")
> test <- plotExpr("hist(rep(10:1,1:10), breaks=6)", "png", "histTest")
> compare(test, control, erase="none")
```

Which informs us that the plots are indeed different:

```
/home/stephen/histControl-1.png
                    "different"
```

And an appropriate plot of the differences is also produced, with the differences highlighted in red:

While this was a trivial example of a single expression, the ability to compare example code from entire functions has also been implemented, with lists of the identical and different files and corresponding plots of the differences produced and also the ability to handle extreme cases such as when new plots have been added in the test group which don't have a corresponding pair in the control group.

3.1 Text Based vs. Bitmap Formats

3.2 Auto-detection of plot logs

3.3 Extensibility for new file formats

3.4 Example: Comparing the `grid` function

Chapter 4

Reporting

4.1 The XSL Language

4.2 Transforming XML logs into HTML reports

Chapter 5

Conclusions

A powerful method of doing regression testing for graphics has been implemented in R. The current method consists of generating sets of plots for the control and test groups, and then comparing them. It is a major improvement on earlier work as it is now platform independent, able to run any arbitrary code, and able to produce plots of differences. It significantly improves the reliability and quality of graphics testing in R.

The current implementation still has much room for improvement. Using the current implementation as a base, the package could be extended to include a function to plot and compare entire packages, which will simply be an extension of the current functions which plot and compare other functions. A function to write a report (such as an HTML report) based on the comparisons would also be a useful addition to the package. Lastly a wrapper class to combine plotting, comparing, and reporting into one step would be a clear finalisation of the package.

Appendix A

Documentation?

This is the documentation.

References

Ross Ihaka and Robert Gentleman. R: A language for data analysis and graphics. *Journal of Computational and Graphical Statistics*, 5(3):299–314, 1996.

P. Murrell and K. Hornik. Quality assurance for graphics in R. In *Proceedings of the 3rd International Workshop on Distributed Statistical Computing*, pages 20–22, Vienna, Austria, March 2003. ISSN 1609-395X. Edited by Hornik K., Leisch, F. & Zeileis, A.

R: Writing R Extensions. R Foundation for Statistical Computing, Vienna, Austria, 2008.