5-3-2007

A REPRODUCIBLE RESEARCH TOOLKIT FOR R

Roger Peng
Department of Biostatistics, Johns Hopkins Bloomberg School of Public Health, rpeng@jhsph.edu

A Reproducible Research Toolkit for R

Roger D. Peng
Department of Biostatistics
Johns Hopkins Bloomberg School of Public Health

May 3, 2007

Abstract

We present a collection of R packages for conducting and distributing reproducible research using R, Sweave, and LaTeX. The collection consists of the **cacheSweave**, **stashR**, and **SRPM** packages which allow for the caching of computations in Sweave documents and the distribution of those cached computations via remotely accessible key-value databases. We describe the caching mechanism used by the **cacheSweave** package and tools that we have developed for authors and readers for the purposes of creating and interacting with reproducible documents.

Key words: R, Literate programming, Cached Computation, Sweave

1 Introduction

Reproducible research is a phrase that is typically used to describe research that is conducted in such a manner that published results can be recreated by a third party by running the original analysis programs on the original data. Reproducible research is distinguished from replication in that replication requires a third party to obtain similar results using new data and an identical or comparable analytic approach. A minimum requirement for reproducible research is that the data and computer programs used to analyze the data are made available and distributed to others.

The idea of a "compendium" is described by Gentleman and Temple Lang (2007) as a way to publish a reproducible analysis by including multiple levels of detail. Readers with a casual interest in the paper may only read the finished product while more interested readers can dig deeper into the specifics of the data and computation. Users of the R programming language have tools such as Sweave (Leisch, 2002) and Emacs Speaks Statistics (Rossini *et al.*, 2004) to assist them in the development of such compendiums and other reproducible documents.

The distribution of reproducible research is a problem for which the solution varies depending on the nature of the research. Small investigations involving moderately sized datasets and standard computational techniques can be archived and distributed in their entirety. Readers can subsequently re-run the entire analysis from start to finish to see if they can obtain the same results as the authors. Complex investigations involving large or multiple linked datasets and sophisticated statistical computations will be more difficult for readers to reproduce because of the resources and time required for running the analysis. In such a situation a method is needed to give readers without such resources the ability to conduct an initial examination of the details of the investigation and to reproduce or verify some of the results.

Peng and Eckel (2007) describe a framework in which reproducible research can be distributed using *cached computations*. Cached computations are saved results that are stored in a database as an analysis is being conducted. These stored results can be distributed in “shared reproducibility packages” via websites or central repositories so that others may explore the datasets and computer code for a given scientific investigation.

In this paper we describe in detail the design and implementations of the **cacheSweave** and **SRPM** packages. Together with the **stashR** package, these packages implement the framework of distributed reproducible research using cached computations described in Peng and Eckel (2007). We describe the mechanism by which computations in Sweave are cached to key-value databases and describe the organization of “shared reproducibility packages” and how they can be created and explored with the **SRPM** package. Details of the **stashR** package have already been written up in Eckel and Peng (2006) so we do not place as much emphasis on describing that package.

2 Caching Sweave Computations

The Sweave system of Leisch (2002) is a literate programming tool based on ideas of Knuth (1984) and is currently part of the core R installation. Specifically, Sweave is a system for processing documents that mix LaTeX document formatting with R code. R code can be interspersed within the LaTeX markup by indicating “code chunks”. These code chunks are evaluated by the `Sweave` function in R and the code is replaced with the results of the evaluation. For example, the code for fitting a linear model and summarizing the estimated regression coefficients might be replaced by a formatted table of estimated regression coefficients along with standard errors and p-values. Another possibility is for the code to replaced by a plot which shows the data and the fitted regression line. In either case, the author writes the code to generate the output and Sweave runs the code and places the output in the final document.

Given a file written in the Noweb format (Ramsey, 1994), one can generate a LaTeX file by running in R

```
> Sweave("foo.Rnw")
```

where “foo.Rnw” contains both LaTeX markup and R code. Calling `Sweave` in this manner results in a the file “foo.tex” being created, which can subsequently be processed by standard LaTeX tools. In particular, the **tools** package contains the R function `texi2dvi` which calls the system’s `texi2dvi` program if it is available.

Sweave has many potential uses, but it is particularly useful for creating statistical documents that are *reproducible*, where the results of computation can be reproduced by executing the original code using the original data. Since the code used for analysis is embedded directly into the relevant document, there is a tighter correspondence between the descriptive text and the computational results and a decreased potential for mismatches between the two. In addition, Sweave’s ability to recompute results to reflect changes or updates to the datasets and analytic code is a great benefit to authors who must maintain statistical documents. With Sweave, all of the relevant text and code reside in a master document from which different outputs can be derived by either “weaving” to create a human-readable document or “tangling” to produce a machine-readable code file.

One aspect of Sweave’s default mode of operation is that all code chunks are evaluated whenever the document is read/processed by the `Sweave` function in R (except when an authors explicitly indicates that a code chunk should not be evaluated). While this is generally considered a feature, it can be cumbersome during the development of a document if the code chunks contain calculations that are lengthy or resource intensive. In particular, changes to text portions of the document require that the entire document be re-Sweaved so that the resulting LaTeX file can reflect the changes to the text. In such cases, it might be desirable for the code chunks to either not be evaluated or to be cached in some manner so that subsequent evaluations take less time.

One approximate solution to the problem described above is to indicate that code chunks should not be evaluated (i.e. by setting `eval=false` as an option for each code chunk) so that the `Sweave` function will skip over them and create the LaTeX file. However, such an approach is probably not desirable since then no results can be displayed in the document. Another approach is to separate out the code chunks that contain lengthy computations into a separate file and then include the resulting file via LaTeX's `\input` directive. This way, the file with the expensive code chunks can be Sweaved once while the text can be modified independently in a separate file. This approach has merit and can also benefit greatly from the use of the `make` utility, but it also breaks the principle of including all of the text and code in a single file. The need to manage multiple files has the potential to lead to the same problems that Sweave and other literate programming tools were (in part) designed to solve.

Consider the following code chunk.

```
> set.seed(1)
> x <- local({
+     Sys.sleep(10)
+     rnorm(100)
+ })
> results <- mean(x)
```

Admittedly, this code chunk is not very interesting or realistic but it is useful for demonstrating the basic approach of the **cacheSweave** package. In the code chunk, we (1) set the random number generator seed; (2) generate 100 standard Normal random numbers after sleeping for 10 seconds; and (3) calculate the mean of the Normal random numbers. After executing the code chunk, there are two objects in the user's workspace (i.e. the global environment): `x` and `results` (there is also a hidden object `.Random.seed` that is created by `set.seed`).

On a modern computer executing the code chunk above should take about 10 seconds since the operations other than the call to `Sys.sleep` use a negligible amount of wall clock time. Although the use of `Sys.sleep` here is artificial, one can imagine replacing it with a call to a function that executes a complex or resource intensive statistical calculation. For the purposes of the task at hand, we may only be interested in the mean of the vector `x`, but we have to spend a reasonable amount of time getting there. Repeated evaluation of this code chunk may be neededlessly time consuming if the code and data do not change after the first evaluation.

The **cacheSweave** package allows users to cache the results of evaluating a Sweave code chunk. In the above example, the basic approach would be to cache the objects `x` and `results` in a key-value database with the key being the object name and the value being the R object itself. On future evaluations of this code chunk (assuming the code has not changed otherwise), we could load `x` from the database rather than wait the 10 seconds as we did on the first evaluation. Using the cached value of `x` we could compute various summary statistics. If we were interested in the mean of `x` we could simply load the cached value of `results` from the database (although in this case direct recalculation of the mean would not take much time).

2.1 Expression caching mechanism

A simple code chunk in a Sweave document might appear as follows.

```
<<FitLinearModel>>=
library(datasets)
library(stats)
data(airquality)
fit <- lm(Ozone ~ Temp + Solar.R + Wind, data = airquality)
@
```

This code chunk loads the `airquality` dataset from the **datasets** package and fits a linear model using the `lm` function from the **stats** package. In this case, two objects are created in the workspace: the `airquality` data frame and the `fit` object containing the output from the `lm` call.

To make use of the caching mechanism provided in **cacheSweave**, the user must set the option `cache=true` in the code chunk declaration. The modified code chunk would be

```
<<FitLinearModel,cache=true>>=
library(datasets)
library(stats)
data(airquality)
fit <- lm(Ozone ~ Temp + Solar.R + Wind, data = airquality)
@
```

The user must also modify the standard invocation of `Sweave` by using the `cacheSweaveDriver` function instead of the default `RweaveLatex` driver function. If the above code chunk were contained in the file "foo.Rnw", then one would call

```
> library(cacheSweave)
> Sweave("foo.Rnw", driver = cacheSweaveDriver)
```

to process the file with the caching mechanism.

On the first evaluation the `cacheSweaveDriver` function does a number of computations in addition to the standard Sweave processing:

1. For each code chunk, a key-value database is created, by default, in the current working directory for storing data objects. The database implementation comes from the **stashR** package. The name of the database is derived from the name of the code chunk and an MD5 digest (Rivest, 1992) of the entire code chunk. Users can change the location of the key-value database by calling the `setCacheDir` function and providing a path.
2. Within each code chunk, there may be multiple expressions and the each expression is handled separately. For each expression:
   (a) The MD5 digest of the expression is taken and looked up in the key-value database. If the digest does not exist, then the expression is evaluated in a temporary environment that has the global environment as a parent.
   (b) After evaluation, the names of the objects created as a result of the evaluation are stored in the key-value database as a character vector with the digest expression as the key.
   (c) The objects created as a result of the evaluation are then stored separately in the database using their own names as keys.

   (d) The objects are then lazy-loaded (see e.g. Ripley, 2004) into the global environment via the `dbLazyLoad` function from the **filehash** package (Peng, 2006).

3. A "map file" is created which is a text file that contains metadata about the code chunks and any resulting databases or figures produced.

The result of running `Sweave` with the `cacheSweaveDriver` function is a LaTeX file, a collection of **stashR** databases either in the current directory or in a directory specified by `setCacheDir`, and a map file which contains information about each of the code chunks.

On a subsequent evaluation, the processing is slightly different. Namely, for each expression in a code chunk:

1. The MD5 digest of the expression is taken and looked up in the key-value database. If the digest exists (indicating that the same expression has been evaluated previously), the names of the objects associated with this expression are retrieved.

2. Given the names of the objects associated with this expression, the objects are then lazy-loaded into the global environment via the `dbLazyLoad` function.

In this situation, the evaluation of a cached expression is replaced by the lazy-loading of the objects associated with that expression into the global environment.

If a future expression (either within the same code chunk or in a subsequent code chunk) requires an object created in a previous code chunk, then that object will be automatically loaded into the global environment via the lazy-loading mechanism.

2.2 Lazy-loading of objects

The lazy-loading of objects into the global environment once they have been cached is a useful feature of the **cacheSweave** package when large objects are used in a code chunk. For example, one code chunk might read in a large dataset and calculate a summary statistic based on that dataset, e.g.

```
<<loadLargeDataset,cache=true>>=
data <- readLargeDataset("datafile")
x <- computeSummaryStatistic(data)
@
```

With caching turned on for this code chunk, the objects `data` and `x` are stored in the cached computation database for this code chunk. A future code chunk then might simply print the summary statistic `x`, for example,

```
<<printX>>=
print(x)
@
```

If the primary interest is in the summary statistic `x`, then on future evaluations of both of these code chunks, the object `data` is never needed. It is only needed on the first evaluation so that the summary statistic can be calculated and stored in the object `x`. When `data` is lazy-loaded in future Sweave runs, it is never accessed and hence never actually loaded from the database. Therefore, code can be written in the manner shown above and there is no need to worry about the `data` object being loaded repeatedly into R when it is not actually needed.

2.3 Construction of `cacheSweaveDriver`

The construction of the `cacheSweaveDriver` function is modeled on the `RweaveLatex` function from the **utils** package. The `cacheSweaveDriver` function returns a list of five functions:

1. `setup`, creates a list of available options. We add an extra option `cache` for indicating whether a code chunk should be cached. We also add the name of the map file so that it can be updated after evaluating each code chunk.

2. `runcode`, based on the `RweaveLatexRuncode` function in the **utils** package, this function executes code in each code chunk and saves objects to **stashR** databases. While much of the original code is retained, we replace the call to `RweaveEvalWithOpt` with our own `cacheSweaveEvalWithOpt` function, which handles the evaluation of the expression, creation of the **stashR** database, and the saving of objects to the database. We also add a call to the function `writeChunkMetadata` which writes out information to the map file.

3. `writedoc`, handles writing of output LaTeX file; we import the `RweaveLatexWritedoc` function from **utils**.

4. `finish`, closes the output connection and prints some final messages; we import the `RweaveLatexFinish` function from **utils**.

5. `checkopts`, checks that code chunk options are valid; we import the `RweaveLatexOptions` function from **utils**.

The bulk of the work is done in the `runcode` function, which handles the evaluation of the expressions in each code chunk. The code in that function is based on the code from R version 2.5.0 (R Development Core Team, 2007).

2.4 Expressions with side effects

Simple expressions, such as assignments, will typically result in a single object being created in the global environment. For example, the expression

```
> x <- 1:100
```

results in an object named `x` being created in the global environment whose value is an integer sequence from 1 to 100.

However, there are other types of expressions which can result in either multiple objects being created in the user's workspace or no objects being created. For example, the `source` function is often used to load objects from an R code file. Unless the `local` argument is set to `TRUE`, these objects will by default be created in the global environment. When the `cacheSweaveDriver` function evaluates an expression that contains a call to `source`, there will be objects created outside of the temporary environment in which the expression is evaluated (again, unless the argument `local = TRUE` is specified in the call to `source`). The `set.seed` function behaves in a similar way by modifying (or creating) the `.Random.seed` object in the global environment.

In order to handle the effects of functions like `source` the function `evalAndDumpToDB`, which evalutes an expression and saves the results to the **stashR** database, first obtains a character vector of the names of all the objects in the global environment. After evaluating the expression in a temporary environment, a check is made to see if any new objects have been created or modified in the global environment. If so, those objects are saved to the database as well as any objects that were created in the temporary environment. Note that we currently make a special case of the global environment. If the code being evaluated creates objects in some other environment, then **cacheSweave** will not be able to cache those objects.

Another example of a function with side effects is the `plot` function (and related functions) from the **graphics** package. Since `plot` does not create any objects in the global environment, but rather creates a plot on a graphics device, there is nothing for `cacheSweaveDriver` to cache. Currently, it is not clear what is the best way to handle this behavior and so calls to plotting functions cannot be cached using the **cacheSweave** package. In the future, we may attempt to detect the creation of a graphics file (e.g. a PDF or EPS file) and store that file along with the cached computations.

There are many other types of expressions that have side effects and do not result in the creation of objects in the global environment. Expressions such as calls to `system` or functions which write out files (e.g. `save`, `save.image`, `write.table`, `dput`, etc.) all result in objects being created outside of R. In general, these expressions cannot yet take advantage of the caching mechanism in **cacheSweave** and must be executed every time `Sweave` is run.

When caching is used, it is useful to divide the code into chunks which setup the data and results (and can use caching) and chunks that present or display the results (and cannot use caching). For example, with the linear model example from the previous section, one might have one code chunk for loading the data and fitting the model

```
<<FitLinearModel,cache=true>>=
library(datasets)
library(stats)
data(airquality)
fit <- lm(Ozone ~ Temp + Solar.R + Wind, data = airquality)
@
```

and another code chunk for summarizing the results in a standard table of regression coefficients.

```
<<LinearModelTable,results=tex>>=
library(xtable)
print(xtable(fit))
@
```

Here, we use the **xtable** package to create a formatted LaTeX table of the regression output. A similar approach could be used for plots by separating out the code that generates the plot, e.g.

```
\begin{figure}
  \centering
<<LinearModelDiagnosticPlot,fig=true>>=
par(mfcol = c(2, 2))
plot(fit)
@
\caption{Linear model diagnostic plots}
\end{figure}
```

3 Packaging Reproducible Documents

While one can certainly use the **cacheSweave** package as a standalone tool for developing Sweave documents, its primary purpose is to create the cached computation databases and metadata so that they can be distributed to others. We describe the mechanism for distributing reproducible research via cached computations in this section.

The **SRPM** package ("Shared Reproducibility Package Management") provides tools for creating and interacting with what we call "shared reproducibility packages". These shared reproducibility packages (SRPs) are *not* true R packages but rather contain information related to a Sweave document that can be distributed to a wide audience and used to reproduce the results. The format of an SRP is meant to be simple so that it can be used on different systems.

Each package is simply a directory which contains the following subdirectories:

- `article/`: contains the original Sweave file and a "weaved" version of the file (e.g. in PDF format)
- `figures/`: contains files corresponding to any figures in the document
- `cacheDB/`: contains a collection of **stashR** databases storing any cached computations from code chunks
- `code/`: contains code files corresponding to the R code for each code chunk in the Sweave document (these are produced with the `Stangle` function with the argument `split = TRUE`).

Each SRP also has a file called `metadata.dcf` which is a text file containing information about each of the code chunks in the document. This file is written in the Debian Control File format.

In addition, a package may contain a file named `REMOTE` which contains the URL of the location of any remote **stashR** databases containing cached computations. If the size of the cached computation databases is large, an author may wish to post them on a webserver rather than distribute them with the SRP itself. The `REMOTE` file indicates the location of the cached computation databases and configures the other tools in the **SRPM** package to retrieve data from this location using the functionality in the **stashR** package.

3.1 Author tools

A shared reproducibility package can be constructed with the `makeSRP` function from the **SRPM** package which takes as arguments the name of the package to create and the name of the Sweave file for the original document. The `makeSRP` function

1. creates the necessary directories for the SRP;
2. calls `Stangle` to create individual code files for each code chunk and copies the files into the `code/` subdirectory;
3. copies the **stashR** databases containing the cached computations into the `cacheDB/` subdirectory;
4. copies graphics files corresponding to figures into the `figures/` subdirectory;
5. copies the article PDF file and Sweave file into the `article/` subdirectory;

6. creates the metadata file by reading the map file produced by the **cacheSweave** package and writes it to the `metadata.dcf` file.

Currently, the **SRPM** package requires that both the graphics files for the figures and the weaved version of the article be in PDF format, however we hope to remove this limitation in the near future.

Another function that is available to authors is the `makeWebpage` function which produces a simple webpage corresponding to an SRP. The webpage lists all of the code chunks in a document with links to the code itself. Also, there are links to cache databases as well as the PDF versions of figures so that readers can browse an SRP using a web browser and without having to have R installed.

3.2 Reader tools

Sweave is an example of a tool that is useful to authors of statistical or scientific documents in that it assists in the development of documents by ensuring that the text and data analysis are closely integrated into a single document. However, readers of reproducible documents also need tools to assist them with interacting with the data analyses therein and reproducing key results.

In the **SRPM** package we provide some basic tools for readers of Sweave documents that allow them to interact with the code and data provided by a shared reproducibility package created by the author. The basic functions are

- `code`: When called with no arguments, a listing of all the code chunks in the article is printed to the console. The `code` function can also take a numeric argument corresponding to the code chunk sequence number of a character argument corresponding to the code chunk name. When `code` is passed a numeric or character argument, it returns an object of class "codeObject" which contains the code and pointers to any cached computation databases or figures associated with the code chunk.

- `article`: This function takes no arguments; when called it launches the article PDF document in the PDF viewer.

- `figure`: This function must be given a numeric argument corresponding to the figure number in the original article. When called, it displays in the PDF viewer the figure corresponding to the figure number.

- `cache`: This function takes a code chunk sequence number or a code chunk name (character) as an argument and returns an object of class "localDB" or "remoteDB" depending on whether the SRP is using local or remote cached computation databases. This object can be explored with the methods defined in the **stashR** package (see also details in Eckel and Peng, 2006).

- `loadcache`: This function lazy-loads cached computation databases into the global environment. It takes a numeric vector of code chunk sequence numbers or a character vector of code chunk names and loads the cached computation databases associated with those code chunks in the order that they are specified. Once a database is lazy-loaded, the object names appear in the environment into which the database was loaded, but they do not occupy any extra memory until they are first accessed. If a specified code chunk does not have a database associated with it, no action is taken.

- `runcode`: The `runcode` function takes as input a numeric vector of code chunk sequence numbers or a character vector of code chunk names and executes the code in those code chunks. Each code chunk is evaluated in the order in which it appears in the input vector. By default, if a cached computation database is associated with a code chunk, then the database is lazy-loaded via `loadcache` rather than executed. In order to force evaluation of code in a code chunk with a cache database, one needs to set `useCache = FALSE` when calling `runcode`. If an error occurs when executing the code in a code chunk, a message is printed to the console indicating the error and the code chunk is skipped.

- `edit`: A method is provided for the `edit` generic function for objects of class "codeObject" which can be used to edit the R code corresponding to a code chunk. The modified "codeObject" object can be executed with the `runcode` function. The `edit` method returns the modified copy of the object so that the original code is not modified. The editor used is that which is launched by the `file.edit` function and will be system dependent.

These functions consist of the primary user interface for readers to interact with shared reproducibility package. Certain SRPs may also require that other R packages be installed in order to execute the code in the code chunks and these should be installed before attempting to execute the code with `runcode`.

Other utility functions available to the user are `currentPackage`, which shows the currently registered SRP, `getRemoteURL`, which returns the URL of the remote cached computation databases (if any), and `getLocalDir`, which returns the path to the directory where local copies of the remote cache databases will be stored.

3.3 Example

The **SRPM** package depends on the **methods** and **stashR** packages and additionally imports the **utils**, **filehash**, and **cacheSweave** packages. Once those dependencies are installed, the **SRPM** package can be loaded using `library` in the usual way.

The first thing a user must do is register a shared reproducibility package (SRP) using the `setPackage` function. We will use as an example the package `srp_seasonal` which can be downloaded from

```
http://www.biostat.jhsph.edu/~rpeng/RR/srp_seasonal.zip
```

This package corresponds to the article Peng *et al.* (2005), a national study of air pollution and mortality in the United States.

Once the package is unzipped, it can be registered by passing the name of the directory to the `setPackage` function.

```
> library(SRPM)
> setPackage("srp_seasonal")
```

Upon registering a package one can call the `article` function (with no arguments) to open a PDF copy of the full article in the PDF viewer (as identified by `getOption("pdfviewer")`). Another useful function to begin with is the `code` function. Called with no arguments, `code` lists all of the code chunks in the article.

```
> code()
```

```
1 SetupCacheSweave
2 mortalityTop10setup  [C]
3 mortalityTop10plot [Figure 1]
4 seasonRegionPM10setup  [C]
5 seasonRegionPM10plot [Figure 2]
6 ComputeNonSeasonalEstimates  [C]
7 ComputeSeasonalStepFunctionEstimates  [C]
8 ComputeNationalAverageEstimates  [C]
9 nationalAverageEstimates
10 ComputeSeasonalPeriodic  [C]
11 ComputePeriodicSeasonByRegion  [C]
12 periodicSeasonByRegionPlot [Figure 3]
13 ComputeOrthogonalEstimates  [C]
14 ComputePeriodicPosteriorBeta  [C]
15 periodicPosteriorBeta [Figure 4]
16 SensitivityAnalysisSetup  [C]
17 ComputeSensitivityLag0  [C]
18 ComputeSensitivityLag1  [C]
19 ComputeSensitivityLag2  [C]
20 CombineSensitivityAnalysisResults  [C]
21 ComputePeriodicDfTimeSensitivity  [C]
22 periodicDfTimeSensitivity [Figure 5]
23 setupCopollComputation  [C]
24 CopollNO2  [C]
25 CopollO3  [C]
26 CopollSO2  [C]
27 CopollPM10  [C]
28 ComputeCopollutantModelTable  [C]
29 copollutantModelTable
30 ComputePeriodicLags012  [C]
31 periodicLags012 [Figure 6]
```

The code chunk listing is annotated with three different types of tags. The first is the code chunk sequence number which appears to the left of the code chunk name. This number can be used to identify a code chunk in other operations. The second is a `[C]` which indicates caching has been turned on and that the corresponding code chunk gives rise to a cached computation database. Lastly, code chunks with the tag `[Figure ?]` produce figures or plots. For example, code chunk 5 produces Figure 2 in the original article.

Code in any of the code chunks can be executed with the `runcode` function by passing the code chunk name or sequence number. Sequences of code chunks can be executed by passing a numeric or character vector to `runcode`. For example, to execute the code in chunks 1 through 3 to create the boxplot in Figure 1 of the original article, we can call

```
> runcode(1:3)
```

which prints the following messages

```
running code in code chunk 1
```

```
ERROR: unable to run code chunk 1
could not find function "setCacheDir"
loading cache for code chunk 2
running code in code chunk 3
```

and produces a series a boxplots on the graphics device. Here, there was an error in executing code chunk 1. This code chunk calls `setCacheDir` and sets the directory for storing the cached computation databases. The error occurs because `setCacheDir` is from the **cacheSweave** package which is not currently loaded. However, the error is irrelevant in this case and does not prevent other code chunks from being executed. Code chunk 2 has a cached computation database associated with it so the database is lazy-loaded into the workspace instead of the code being executed. Code chunk 3 creates the plot and it is executed successfully.

In order to explicitly lazy-load a cached computation database into the workspace one can use the `loadcache` function. For example, to load the database for code chunk 11, one can call

```
> loadcache(11)
> ls()
```

```
 [1] "b"               "B"               "bc"              "cityRegion"
 [5] "conf"            "curve"           "curveRegion"     "exclude"
 [9] "g"               "ndays"           "nRegions"        "pooledRegion"
[13] "pooledRegionSD"  "r"               "regionFullNames" "regionInd"
[17] "regionNames"     "rng"             "V"               "x"
[21] "y"               "Y"               "zero"
```

The “localDB” object representing the cached computation database can be explicitly retrieved by calling the `cache` function and accessed using the methods defined in the **stashR** package. The above code fragment is roughly equivalent to

```
> library(stashR)
> db <- cache(11)
> show(db)
```

```
'localDB' database 'ComputePeriodicSeasonByRegion'
```

```
> dbList(db)
```

```
 [1] "exclude"         "r"               "cityRegion"      "regionNames"
 [5] "regionInd"       "ndays"           "B"               "bc"
 [9] "pooledRegion"    "pooledRegionSD"  "zero"            "curveRegion"
[13] "V"               "b"               "curve"           "regionFullNames"
[17] "Y"               "conf"            "rng"             "y"
[21] "nRegions"        "x"               "g"
```

```
> dbLazyLoad(db)
```

3.4 Remote packages

Shared reproducibility packages can be structured in such a way that the cached computation databases can be stored on a remote server and accessed over the web using R’s Internet capabilities. In that case the SRP that is distributed does not have any of the **stashR** databases in it but rather contains a REMOTE file containing the URL indicating the location of the cached computation databases on the web. A remote version of the srp_seasonal package called srp_seasonal_R is available from

```
http://www.biostat.jhsph.edu/~rpeng/RR/srp_seasonal_R.zip
```

The REMOTE file for this package contains the following information:

```
> remote <- readLines("srp_seasonal_R/REMOTE")
> writeLines(remote)

RemoteURL: http://www.biostat.jhsph.edu/~rpeng/RR/seasonal
```

The RemoteURL field in the file contains the URL for the cached computation databases. If a REMOTE file exists it is automatically read by the setPackage function. The remote URL can be retrieved in an R session via the getRemoteURL function. Once a remote package is registered with setPackage all of the functions mentioned above will work, however, depending on the speed of the network connection there may be some delay in downloading large objects when they are accessed.

3.5 Package websites

If the author has created a webpage via the makeWebpage function, the reader may prefer to view that first. The webpage created by makeWebpage corresponding to the srp_seasonal package can be found at

```
http://www.biostat.jhsph.edu/~rpeng/RR/seasonal/html/
```

Currently, the webpage created by makeWebpage resembles the output provided by the R functions at the console. The cached computation databases can be browsed in a limited fashion—smaller objects can be viewed in their entirety while for larger objects a summary is provided by showing the output from str.

4 Discussion

We have described the design and implementation of the **cacheSweave** and **SRPM** packages which, together with the **stashR** package, provide authors tools for conducting and distributing reproducible research. The **SRPM** and **stashR** packages also provide tools for readers to interact with the research behind a document via shared reproducibility packages (SRPs). These SRPs contain the code and cached computation databases corresponding to an article. They can be distributed widely to others over the web and the cached computation databases can be served over the web separately if they are too large to be efficiently distributed in their entirety.

The functionality of the **cacheSweave** package will likely be useful to those running very lengthy or resource intensive computations. For example, the analysis conducted for the article described in Section 3.3 takes approximately 6 hours to run on an AMD Athlon 64 X2 4800+ machine running Fedora Core 5 Linux. An interested reader of the article may not initially be interested in reproducing the analysis in its entirety but may want to explore certain parts of the analysis or dataset. The cached computation databases allow the reader to conduct this type of preliminary examination prior to fully reproducing an analysis.

It should be noted that the SRPs can be created without any caching being used. In particular, for shorter documents whose computations do not take substantial time to run, there is no need to cache any of the code chunks. In that case, one can still generate an SRP with the **SRPM** package for distribution and it will contain the code, figures, and article. One still needs to use the `cacheSweaveDriver` function in the **cacheSweave** package because the information written to the map file is used in creating the SRP.

Currently, the **cacheSweave** package works for fairly standard analyses but cannot handle more complex or nonstandard ones, for example, when objects are created in environments other than the global environment. Another feature missing from **cacheSweave** is code dependency checking between code chunks. That is, if code chunk 2 depends on the results of code chunk 1, then any changes to code chunk 1 will likely result in the need to re-run code chunk 2 if code chunk 2 has been cached. However, **cacheSweave** cannot detect the dependence of code chunk 2 on code chunk 1 and will not re-run code chunk 2 in this situation.

The **weaver** package (Falcon, 2007) from the Bioconductor Project implements similar functionality to **cacheSweave** and makes use of the **codetools** package (Tierney, 2007) to detect dependencies between code chunks (and re-run code chunks when necessary). The **weaver** package caches the results of expressions in separate files stored in the R workspace format. These results are then loaded (rather than lazy-loaded) into R upon subsequent evaluation of the same expression.

The **SRPM** package provides rudimentary tools for interacting with a shared reproducibility package and further development is needed to create more useful interfaces for readers. Also, more tools are needed for authors to develop "views" of their research that are amenable to different types of readers (see e.g. Gentleman and Temple Lang, 2007). Currently, the `makeWebpage` function creates an interface that is browsable with a web browser, but there may be better ways to present the information to other readers.

There are substantial benefits to scientists and statisticians if future research is made reproducible. By expediting the dissemination of ideas and publishing the full details of a scientific investigation, researchers can more easily adapt the findings of others and build off of existing knowledge. However, progress in reproducible research will be impeded if the proper tools are not developed and made available to both authors and readers.

5 Acknowledgements

This research was supported in part by a Faculty Innovation Fund award from the Johns Hopkins Bloomberg School of Public Health, grant ES012054-03 from the National Institute of Environmental Health Sciences.

References

Eckel SP, Peng RD (2006). "Interacting with local and remote data repositories using the stashR package for R." *Technical Report 127*, Johns Hopkins University Department of Biostatistics. http://www.bepress.com/jhubiostat/paper127.

Falcon S (2007). *weaver: Tools and extensions for processing Sweave documents*. R package version 1.2.0.

Gentleman R, Temple Lang D (2007). "Statistical Analyses and Reproducible Research." *Journal of Computational and Graphical Statistics*, **16**(1), 1–23.

Knuth DE (1984). "Literate Programming." *Computer Journal*, **27**(2), 97–111.

Leisch F (2002). "Sweave: Dynamic generation of statistical reports using literate data analysis." In W Härdle, B Rönz (eds.), "Compstat 2002 — Proceedings in Computational Statistics," pp. 575–580. Physika Verlag, Heidelberg, Germany. ISBN 3-7908-1517-9.

Peng RD (2006). "Interacting with data using the filehash package." *R News*, **6**(4), 19–24.

Peng RD, Dominici F, Pastor-Barriuso R, Zeger SL, Samet JM (2005). "Seasonal Analyses of Air Pollution and Mortality in 100 US Cities." *American Journal of Epidemiology*, **161**, 585–594.

Peng RD, Eckel SP (2007). "Distributed reproducible research using cached computations." *Technical report*, Johns Hopkins University Department of Biostatistics.

R Development Core Team (2007). *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`.

Ramsey N (1994). "Literate Programming Simplified." *IEEE Software*, **11**(5), 97–105.

Ripley BD (2004). "Lazy Loading and Packages in R 2.0.0." *R News*, **4**(2), 2–4. URL `http://CRAN.R-project.org/doc/Rnews/`.

Rivest RL (1992). *The MD5 Message-Digest Algorithm*. RFC 1321. http://tools.ietf.org/html/rfc1321.

Rossini AJ, Heiberger RM, Sparapani RA, Mächler M, Hornik K (2004). "Emacs Speaks Statistics: A Multiplatform, Multipackage Development Environment for Statistical Analysis." *Journal of Computational and Graphical Statistics*, **13**(1), 247–261.

Tierney L (2007). *codetools: Code Analysis Tools for R*. R package version 0.1-1.