12-20-2006

INTERACTING WITH LOCAL AND REMOTE DATA RESPOSITORIES USING THE stashR PACKAGE

Sandrah P. Eckel
Johns Hopkins Bloomberg School of Public Health, Department of Biostatistics, seckel@jhsph.edu

Roger Peng
Johns Hopkins Bloomberg School of Public Health, Department of Biostatistics

Interacting with local and remote data repositories using the **stashR** package

Sandrah P. Eckel Roger D. Peng

December 20, 2006

Abstract

The **stashR** package (a Set of Tools for Administering SHared Repositories) for R implements a simple key-value style database where character string keys are associated with data values. The key-value databases can be either stored locally on the user's computer or accessed remotely via the Internet. Methods specific to the **stashR** package allow users to share data repositories or access previously created remote data repositories. In particular, methods are available for the S4 classes 'localDB' and 'remoteDB' to insert, retrieve, or delete data from the database as well as to synchronize local copies of the data to the remote version of the database. Users efficiently access information from a remote database by retrieving only the data files indexed by user-specified keys and caching this data in a local copy of the remote database. The local and remote counterparts of the **stashR** package offer the potential to enhance reproducible research by allowing users of `Sweave` to cache their R computations for a research paper in a 'localDB' database. This database can then be stored on the Internet as a 'remoteDB' database. When readers of the research paper wish to reproduce the computations involved in creating a specific figure or calculating a specific numeric value, they can access the 'remoteDB' database and obtain the R objects involved in the computation.

1 Overview

We conduct scientific research by collecting data, analyzing and summarizing the evidence in the data and then publishing substantive results in a paper. To verify scientific results we must either replicate or, at a minimum, reproduce the findings of a previous study. Replication is the act of collecting an independent data set in a similar manner to the original study and then using the data to address a similar scientific question of interest. Reproduction of scientific research is the

act of using the same data as the original study and performing additional statistical analyses. Replication is the highest standard of verification. Through replication, researchers can address the uncertainty inherent in collecting a data sample from a larger population and improve upon any shortcomings of previous data collection designs. Reproducibility allows researchers to evaluate the sensitivity of the results from the initial statistical analysis of a study. Although replication is ideal, in many cases reproduction is the most practical way to verify the results of a study. In some instances, such as epidemiologic studies of national databases, reproduction is the only way to verify the scientific findings of a study.

Truly reproducible research can be achieved by making the data used for statistical analysis available to prospective analysts. Given the framework of reproduction, we need to be able to distribute potentially large data sets and computations. To disseminate accurate data, we need a system for physical data distribution that manages data updates by automatically synchronizing each prospective analyst's local copy of the data to the remote, master copy of the data. It follows that we need software to help manage both the distribution and synchronization/caching of this data. We introduce stashR to fill this need.

2 The stashR Package

The **stashR** package is an extension to local and remote databases of the **filehash** package by (Peng, 2006), which allows users to work interactively with data sets too large to be loaded into R as a single object by alternatively using a key-value database. A key-value database is a collection of data files, each indexed by a character string "key". One example of a key-value database is a multi-center study consisting of data from 4 cities (New York, Los Angeles, Chicago and Seattle) where data for each city is stored in a file named 'ny', 'la', 'chicago', and 'seattle' respectively. In this case, the key-value database allows a researcher to download data from a particular city of interest instead of downloading the entire dataset at once. The **stashR** package can be used to create a local 'localDB' key-value database and to download data from a 'remoteDB' key-value database stored remotely on the internet. A local key-value database, 'localDB', is stored locally on the user's computer so that the user has control over the content of the database. A remote key-value database, 'remoteDB', is stored on a remote server. A 'remoteDB' user can download data from the remote server to a local copy of the database on his or her own computer. This dual functionality of the **stashR** package addresses the need for managing data distribution and synchronizing cached

data copies in fully reproducible research.

The **stashR** package adds important functionalities to data handling and distribution in R. These contributions include:

- the ability to access remote databases efficiently
- a set of tools for creating a local database for exporting to a remotely accessible server
- a tool for synchronizing local copies of a database to the remote version
- an abstract interface for interacting with local and remote databases.

3 Design Rationale

3.1 Remote vs. Local

The **stashR** package is designed for interacting with both local and remote data repositories. A repository is structured set of files in which data and metadata are stored to facilitate data accessibility. Each of the main user interface functions in **stashR** is a generic function with specific methods defined for repository objects of class 'localDB', the local version of a key-value database, and for objects of class 'remoteDB', the remote version of a key-value database. A 'localDB' data repository is stored on a user's local disk whereas a 'remoteDB' data repository is stored on a remote server on which the user typically does not have the right to edit files. When interacting with a 'localDB' data repository, the user can insert, fetch, delete and list keys of the available data files. When interacting with a 'remoteDB' data repository, the user creates a copy of the repository on his or her local disk that contains only the desired data files from the remote repository. The user interface functions for the 'remoteDB' repository are similar to those for the 'localDB' repository. When the user fetches data from a remote repository, it is either accessed using the local cache, or downloaded from the remote repository if it has not previously been downloaded. The **stashR** package also has a feature to synchronize the local copy of a repository to the remote repository.

3.2 Repository Layout

The 'remoteDB' repository consists of a root directory containing a data directory, a text file 'keys' that lists of each of the character keys corresponding to a data file in the data directory, and a text file 'url' that lists the repository's URL on the remote server. The data directory contains compressed

data files labelled according to their corresponding character key. Each data file has a corresponding '.SIG' text file that lists the 32-byte MD5 checksum from running `md5sum()` on the data file (see the R package **tools** for more details) and the data file's identifying character key. The '.SIG' files allow for synchronization of the local data copies to the master version of the data repository on the remote server. The 'localDB' repository has an identical layout to the 'remoteDB' repository except that it does not have a 'url' file.

3.3 Caching and Synchronization

The **stashR** package allows users to cache or access cached data in a 'localDB' or 'remoteDB' repository. A key feature of the **stashR** package, is the ability for a user to download desired data from a 'remoteDB' repository in a local directory and, at a later date, synchronize their locally cached data to the data in the remote 'remoteDB' data repository. The synchronization feature allows a user to efficiently maintain an up-to-date local cache of remotely stored data by downloading updated versions of the remote data files only when needed.

As noted in Section 3.2, each data file has a corresponding '.SIG' file that contains the MD5 checksum of the data file along with the key indexing the data file. The MD5 checksum is theoretically a nearly unique character string that identifies a file. If a small change is made to a file, its corresponding MD5 checksum will change dramatically. Sychronization in the package **stashR** is acheived by comparing the MD5 checksum in the '.SIG' file corresponding to the local copy of the data and the MD5 checksum in the'.SIG' file corresponding to the remote data file. If a this data file has been modified on the remote data repository, the MD5 checksums will not match, and the new data file and '.SIG' file will be downloaded to the local copy of the repository to synchronize the local copy of the data to the remote version of the data.

4 Interface

4.1 Creating a 'localDB' repository or a local copy of a 'remoteDB' repository

There are two steps to creating a 'localDB' or a 'remoteDB' object. The first step is to call `new("localDB", dir, name)` or `new("remoteDB", dir, url, name)` where 'dir' is a character string specifying the local directory in which to create the new 'localDB' repository of the local copy of the 'remoteDB' repository. The 'url' argument is unique to objects of the 'remoteDB' class and it spec-

ifies as a character string the URL of the root directory of the remote key-value database. The 'name' argument is a character string specifying the label that will be associated with the 'localDB' or 'remoteDB' repository. Upon calling `new`, the appropriate `initialize` method will be called to create the local directories needed for either storing the cached copy of the database for 'remoteDB' objects or for storing the data for 'localDB' objects.

4.2 Accessing a remoteDB database

The user-end interfaces to 'localDB' or 'remoteDB' databases are of the functions `dbFetch`, `dbInsert`, `dbList`, `dbExists`, `dbDelete` and `dbSync`. Each of these functions is a generic function defined in the **filehash** package and has a specific method for objects of the 'remoteDB' and 'localDB' classes (with the exception of `dbSync`, which is only available for the 'remoteDB' class). The first argument for any of the above functions is an object of class 'remoteDB' or 'localDB'.

4.2.1 `dbFetch`

The function `dbFetch` takes two arguments. The first argument is either a 'localDB' or 'remoteDB' object and second argument is a character string key indexing a data object.

For objects of the class 'remoteDB', `dbFetch` first checks to see if the provided key's data file and .SIG file exist in the local copy of the 'remoteDB' repository. If the data and .SIG files indexed by the key do not exist, then `dbFetch` downloads the two files from the remote repository to the local copy and reads the data file. If the data and .SIG file do exist in the local copy of the repository, then `dbFetch` compares the MD5 checksum stored in the .SIG file from the local repository to the MD5 checksum stored in the .SIG file in the remote repository. If the MD5 checksums are the same, then `dbFetch` reads the file from the local repository. Otherwise, `dbFetch` downloads the updated version of the data and .SIG files from the remote repository and reads the data file. The object associated with the key, that was stored in the corresponding data file, is returned by `dbFetch`.

Similarly, for objects of the class 'localDB', `dbFetch` checks if the provided character value key's data file and '.SIG' file exist in the local repository. If the files exist, then `dbFetch` reads the data file from the local directory and returns the R object stored in the data file. If the corresponding files do not exist, then `dbFetch` returns an error.

4.2.2 dbInsert

The function `dbInsert)` takes four arguments. The first argument, like any of the other user-end interfaces is either a 'localDB' or 'remoteDB' object. The second argument is a character string key indexing the file that will be created to store the object indicated by the 'value' argument. The third argument, value, is any R object that the user wishes to store in the repository.

Calling `dbInsert` on a 'remoteDB' object returns an error message. Thus the user cannot write to a remote repository or write to his or her local copy of the remote repository, which would make the two versions of the repository out of sync.

On the other hand, calling `dbInsert` on a 'localDB' object writes the value to a data file corresponding to the specified key within the local data directory. Also, `dbInsert` appends the specified key to the end of the 'keys' file if the key is not already included in the 'keys' file. The fourth argument of the `dbInsert` function allows the user to specify whether or not they will allow `dbInsert` to overwrite a pre-existing file with the same key. The defalt is set to 'TRUE' so that `dbInsert` will overwrite a pre-existing file indexed by the same key as the file that the user is trying to insert with `dbInsert`.

4.2.3 dbList

The function `dbList(db = "localDB")` or `dbList(db = "remoteDB", save = FALSE)` takes a 'localDB' or 'remoteDB' object as its argument. For 'remoteDB' objects, there is also an option to save the 'keys' file from the remote repository to the analogous location in the local copy of the repository. For both classes of objects, `dbList` reads the character string key values stored in the 'keys' file of the repository and returns a vector of the keys.

4.2.4 dbExists

In general terms, the function `dbExists` allows a user to determine which elements of a vector of character string keys are contained in the repository. The function `dbExists(db = "localDB" or "remoteDB", key = key)` has a second argument, 'key', which takes a vector of character strings. The logical vector returned by `dbExists` is of the same length as the vector of character keys.

For both objects of class 'remoteDB' and objects of class 'localDB', `dbExists` returns TRUE for each key that indexes a data file contained the repository (as indicated in the 'keys' file of the

repository). If a key in the vector of keys specified as the key argument to `dbExists` indexes a file that is not contained 'keys' file of the repository, `dbExists` returns FALSE in the corresponding position of the output vector of logical values.

4.2.5 dbDelete

The function `dbDelete` allows a user to delete both the data and '.SIG' file indexed by a particular key from the repository. The function call is `dbDelete(db = "remoteDB" or "localDB", key = "character")`. Calling `dbDelete` on a 'remoteDB' object returns an error message since the user does not have access to the remote repository to delete the specified files. On the other hand, calling `dbDelete` on a 'localDB' object results in the deletion of the specified data and '.SIG' file from the data directory of the local repository. The specified key is also deleted from the 'keys' file in the top-level directory of the repository.

4.2.6 dbSync

The **stashR** function for synchronizing local copies of data stored on a remote repository is the generic function `dbSync`. Currently, `dbSync` only has a method for objects of the 'remoteDB' class because one would only need to synchronize a local copy of a remote database. The `dbSync` function takes as arguments a 'remoteDB' object and a (possibly null) character vector of keys, called 'key'. If the 'key' vector contains a character string key that corresponds to a data file that has not yet been downloaded to the local copy of the repository, `dbSync` returns an error message. If the 'key' vector is null, then `dbSync` obtains a list of the data files that have been locally cached, checks if these data files have changed on the remote repository, and then updates the necessary data files. Similarly, if the 'key' vector contains only keys for data files that have been locally downloaded, `dbSync` will only check and, if necessary, update the files specified in the 'key' vector.

5 Examples

5.1 Objects of the class 'localDB'

For objects of the class 'localDB', we start out by defining a local directory in which we will create the repository.

```
> library(stashR)
```

```
Simple key-value database (version 0.8-1 2006-09-25)
A Set of Tools for Administering SHared Repositories
(version 0.1 2006-12-11)
```

```
> wd <- getwd()
> dir <- file.path(wd, "localDBExample")
```

Next, we perform a two-step process to create the 'localDB' object, which we will call 'fhLocal'.

```
> fhLocal <- new("localDB", dir = dir, name = "localDB Example")
```

We now insert different types of R objects into the local repository to create a basic 'localDB' database. Note that each time we call `dbList`, we see the keys indexing all of the data files we have inserted.

```
> v <- 1:10
> dbInsert(fhLocal, key = "vector", value = v, overwrite = TRUE)
> m <- matrix(1:20, 5, 4)
> dbInsert(fhLocal, key = "matrix", value = m, overwrite = TRUE)
> d <- data.frame(cbind(id = 1:5, age = c(12, 11,
+     15, 11, 14), sex = c(1, 1, 0, 1, 0)))
> dbInsert(fhLocal, key = "dataframe", value = d,
+     overwrite = TRUE)
> dbList(fhLocal)
```

```
[1] "vector"    "matrix"    "dataframe"
```

```
> l <- list(v = v, m = m, df = d)
> dbInsert(fhLocal, key = "list", value = l, overwrite = TRUE)
> dbList(fhLocal)
```

```
[1] "vector"    "matrix"    "dataframe" "list"
```

We can fetch any of the R objects saved in our local repository.

```
> dbFetch(fhLocal, "dataframe")
```

```
  id age sex
1  1  12   1
2  2  11   1
3  3  15   0
4  4  11   1
5  5  14   0
```

If we delete a data file from the local repository, `dbList` or `dbExists` can be used to confirm the deletion.

```
> dbDelete(fhLocal, "vector")
> dbExists(fhLocal, "vector")

[1] FALSE

> dbList(fhLocal)

[1] "matrix"    "dataframe" "list"
```

5.2 Objects of the class 'remoteDB'

The same data used in the previous example for 'localDB' has been stored in a 'remoteDB' repository on the internet at:

```
> myurl <- "http://www.biostat.jhsph.edu/~seckel/remoteDBExample"
```

In this example, we will use the 'remoteDB' methods for the **stashR** package interface functions: `dbFetch`, `dbList`, `dbExists` and `dbSync`. Note that we will not use `dbInsert` and `dbDelete` functions because these methods simply return error messages for 'remoteDB' objects.

Again, we start off with the two-step process of creating a 'remoteDB' object. The local copy of the database will be located in our working directory under a directory called 'remoteDBExample'.

```
> wd <- getwd()
> dir <- file.path(wd, "remoteDBExample")
> fhRemote <- new("remoteDB", url = myurl, dir = dir,
+     name = "remoteDB Example")
```

When we run `dbList` on the 'remoteDB' object, 'fhRemote', we see the same four character string keys corresponding to the data values from the previous example. Using the `save = TRUE` option in `dbList` saves a copy of the 'keys' file from the remote version of the database to the local copy of the database. The function `dbExists` can be used as a shortcut, when the list of keys is long, to see which elements of a vector of keys are contained in the database.

```
> dbList(fhRemote, save = TRUE)

[1] "matrix"    "dataframe" "list"      "vector"

> dbExists(fhRemote, c("vector", "array", "list",
+     "function"))

[1]  TRUE FALSE  TRUE FALSE
```

We can fetch any of the data values indexed by the keys resulting from `dbFetch`. Once we have downloaded a data file to the local cache, `dbFetch` simply looks in the local cache for the data file rather than downloading the file again over the internet.

```
> dbFetch(fhRemote, "vector")
trying URL 'http://www.biostat.jhsph.edu/~seckel/remoteDBExample/data/vector'
Content type 'text/plain; charset=UTF-8' length 59 bytes
opened URL
downloaded 59 bytes

trying URL 'http://www.biostat.jhsph.edu/~seckel/remoteDBExample/data/vector.SIG
Content type 'text/plain; charset=UTF-8' length 42 bytes
opened URL
downloaded 42 bytes

 [1]  1  2  3  4  5  6  7  8  9 10

> dbFetch(fhRemote, "matrix")
trying URL 'http://www.biostat.jhsph.edu/~seckel/remoteDBExample/data/matrix'
Content type 'text/plain; charset=UTF-8' length 97 bytes
```

```
opened URL
downloaded 97 bytes

trying URL 'http://www.biostat.jhsph.edu/~seckel/remoteDBExample/data/matrix.SIG
Content type 'text/plain; charset=UTF-8' length 42 bytes
opened URL
downloaded 42 bytes

     [,1] [,2] [,3] [,4]
[1,]    1    6   11   16
[2,]    2    7   12   17
[3,]    3    8   13   18
[4,]    4    9   14   19
[5,]    5   10   15   20

> dbFetch(fhRemote, "matrix")

     [,1] [,2] [,3] [,4]
[1,]    1    6   11   16
[2,]    2    7   12   17
[3,]    3    8   13   18
[4,]    4    9   14   19
[5,]    5   10   15   20
```

As mentioned previously, the function `dbSync` allows a user to synchronize a local copy of a remote database to the remote version of the database. Using the `key = NULL` option synchronizes all data files in the local copy of the database, while specifying a vector of keys synchronizes only the specified keys.

```
> dbSync(fhRemote, key = NULL)
> dbSync(fhRemote, key = c("matrix", "vector"))
```

6 Application: NMMAPS database

The National Morbidity, Mortality, and Air Pollution Study (NMMAPS) is a large multi-city time series study of the short-term effects of ambient air pollution on health (Samet et al., 2000a,b; Peng et al., 2005). The primary aims of the study were to develop methods for integrating national-level databases for the purposes of assessing the health effects of air pollution. Another aim was to disseminate the assembled datasets as well as software implementing the methodology developed. To this end, the Internet-based Health and Air Pollution Surveillance System (iHAPSS) website was created (http://www.ihapss.jhsph.edu/) to distribute the data, methods, and documentation.

Currently, the multi-city NMMAPS database of weather, air pollution, and mortality time series data are available from the iHAPSS website in two formats. The first format consists of separate city-specific comma-separated-value (CSV) files and is the most generic, suitable for reading into almost any statistical package. The second format is specific to the R software system and is the **NMMAPSdata** package (Peng and Welty, 2004). This R package combines all of the data into a single bundle and provides a few functions for reading the data into R.

In practice, one may not be interested in purely reproducing the multi-city analyses of the original NMMAPS study. Certain users may be interested in city-specific analyses or in pooling results over a small number of cities in a region. For example, an official working in a local public health office may be interested only in the air pollution health risks for his or her city. Such an analysis may not depend on data from other cities. These users would therefore only require a subset of the available data.

As an alternative to the **NMMAPSdata** package we have developed the **NMMAPSlite** package which takes advantage of the functionality made available by the **stashR** package. This package, when loaded, establishes a connection to three separate 'remoteDB' databases containing NMMAPS data. The three databases are

- `outcome`: daily mortality counts for 108 U.S. cities spanning the years 1987–2000;
- `exposure`: daily time series of weather and air pollution variables;
- `Meta`: metadata for the NMMAPS cities and other information.

The data from each database is associated with a key that is derived from the name of the city.

The primary functions in the **NMMAPSlite** package for accessing the NMMAPS data are `readCity`, `getMetaData`, `listCities`, and `initDB`. The `readCity` function in **NMMAPSlite** is used for

reading the mortality, air pollution, and weather data from the `exposure` and `outcome` databases. This function has one required argument, which is the abbreviated name of the cities. A character vector of all the names available from the database can be obtained by calling the `listCities` function. The `readCity` function fetches the outcome and exposure data and by default, merges them together in to one data frame and returns it to the user. If the user sets the option `asDataFrame = FALSE`, then a list of two data frames named "outcome" and "exposure" is returned. In addition, there is an option to collapse the age categories of the mortality/outcome data before merging it with the exposure data.

The `getMetaData` function can be used to obtain objects from the `Meta` database. Called with default arguments, `getMetaData` returns a character vector with the names of the metadata objects that are available from the database. If `getMetaData` is passed a non-`NULL` `name` argument, then it returns the object associated with that name from the `Meta` database.

Before any data can be downloaded after loading/attaching the package, one must first call the `initDB` function to create the local directory where for caching the remote data. By default, `initDB` creates a directory called "NMMAPS" in which it will cache local copies of the data. The `initDB` function also initializes the 'remoteDB' class objects with the correct URLs of the databases.

```
> library(NMMAPSlite)
> initDB(basedir = "NMMAPS")
```

After initializing the database, one can access the data using the above-mentioned functions.

```
> listCities()[1:10]
> data <- readCity("akr")
> getMetaData()
```

Census data is available for each of the cities in the database and can be obtained by calling `getMetaData("census")`. In addition, a short description of all of the variables available for each city can be found by typing `?variables`.

7 Discussion

7.1 Related R packages

The **filehash** package allows an individual to interactively access a (potentially quite large) dataset through a key-value database located locally on the user's disk. As **stashR** is an extension of

filehash, **stashR** is related to many of the large data handling packages mentioned in Peng (2006). One such package is **g.data** by David Brahm (Brahm, 2002). One of the main ideas of **filehash** and hence **stashR** is that by using keys to index the data, we can efficiently download only the necessary data objects on an interactive level. The **g.data** package provides interfaces to delayed-data packages (DDPS). DDPS efficiently accesses data by exploiting the lazy evaluation feature of R to load the specified data into memory only once the data is requested. An individual can interface with one of the common relational databases (Oracle, MySQL or SQLite) in R by using the S4 methods and generics in the **DBI** package created by the R Special Interest Group on Databases (R-SIG-DB) (2006), along with **ROracle**, **RMySQL** or **RSQLite** respectively.

The **reposTools** package on Bioconductor by Gentry and Gentleman (2004), which mainly serves to create and interfaced with repositories of R packages, has several features whose purposes are similar to features of the **stashR** package. Briefly, the **reposTools** package allows a user post the contents of a local directory to a server for distribution. While creating the remote repository, **resposTools** adds files to the repository that contain information about the repository's contents. On the client side, **reposTools** allows users to access the packages or objects in the remote repository and download, install or update to the user's specified local directory. The automatic library management feature of **reposTools** keeps track of which R packages are installed on the user's system as well as the corresponding version numbers which facilitates the updating packages feature of **reposTools**. Both the **reposTools** and **stashR** packages help an individual to create a remote repository to distribute the contents to other R users, who are provided with specific methods to interface with the remote repository. Both packages allow a user to download files from the remote repository to a local repository and later to 'synchronize' or 'update' local copies of files or packages to the remote version. Unlike **reposTools**, the **stashR** package contains the tools to construct local repositories, but **stashR** does not yet automate the posting of this repository to a server as does **reposTools**. The key difference between the two packages is that **reposTools** is intended for repositories of R packages whereas **stashR** is designed for repositories of key-value databases which can store arbitrary R data objects.

7.2 Extensions and future work

The **stashR** package has been designed as a tool for both authors and readers of statistical documents in the context of streamlining and enhancing reproducible research documents created, for example, by using `Sweave`. There is a need for future work linking, on the producer's end, the

results of R code chunks of `Sweave` documents to a localDB database. This local database would then need to be transferred in an automatic way as a remoteDB database to a repository on the internet. Ideally, this internet repository would be a central database that all statistical researchers could use, although individuals could alternatively choose to store their data on their own server. On the consumer's end, we need to develop a method for allowing a user to 'click' on a figure or numerical result in a pdf document produced by `Sweave` and then have returned to them the R objects (stored in the remoteDB database) used in the computation of their result of interest as well as the R code chunk that operates on these objects. In this case, the R objects used in the computation of each figure or numerical value would be indexed by a key that is the name of each R code chunk in the `Sweave` document.

References

Brahm, D. E. (2002), "Delayed Data Packages," *R News*, 2, 11–12.

Gentry, J. and Gentleman, R. (2004), *reposTools: Repository tools for R*, R package version 1.5.2.

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

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

Peng, R. D. and Welty, L. J. (2004), "The NMMAPSdata Package," *R News*, 4, 10–14.

R Special Interest Group on Databases (R-SIG-DB) (2006), *DBI: R Database Interface*, R package version 0.1-10.

Samet, J. M., Zeger, S. L., Dominici, F., and et al. (2000a), *The National Morbidity, Mortality, and Air Pollution Study, Part I: Methods and Methodological Issues*, Health Effects Institute, Cambridge MA.

— (2000b), *The National Morbidity, Mortality, and Air Pollution Study, Part II: Morbidity and Mortality from Air Pollution in the United States*, Health Effects Institute, Cambridge MA.