6-25-2004

The National Morbidity, Mortality, and Air Pollution Study Database in R

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

Leah J. Welty
Johns Hopkins Bloomberg School of Public Health, lwelty@jhsph.edu

Aidan McDermott
Department of Biostatistics, Johns Hopkins Bloomberg School of Public Health

The National Morbidity, Mortality, and Air Pollution Study Database in R

Roger D. Peng   Leah J. Welty   Aidan McDermott

Department of Biostatistics
Johns Hopkins Bloomberg School of Public Health

Abstract

The `NMMAPSdata` package contains daily mortality, air pollution, and weather data originally assembled as part of the National Morbidity, Mortality, and Air Pollution Study (NMMAPS). The data have recently been updated and are available for 108 United States cities for the years 1987–2000. The package provides tools for building versions of the full database in a structured and reproducible manner. These database derivatives may be more suitable for particular analyses. We describe how to use the package to implement a multi-city time series analysis of mortality and PM_{10}. In addition we demonstrate how to reproduce recent findings based on the NMMAPS data.

1 Introduction and Background

Time series studies of air pollution and health play an important role in understanding the short-term effects of ambient air pollution on mortality and morbidity. Multi-city studies in particular provide strong and consistent evidence of a positive association between daily changes in air pollution levels and daily changes in mortality counts (Katsouyanni et al., 2001; Samoli et al., 2002, 2003; Samet et al., 2000a,b; Dominici et al., 2002a; Daniels et al., 2004; Burnett et al., 1998). The issues of data management become considerably more complex in multi-city studies than in single city studies, necessitating the development of software tools to manage, manipulate, and analyze the data. We focus here on the National Morbidity, Mortality, and Air Pollution Study (NMMAPS), which consists of 108 United States cities, each of which contains 5114 daily measurements on over 70 different variables.

The original NMMAPS examined the relationships between daily mortality/morbidity and air pollution in 90 cities for the years 1987–1994, but has since been updated to include 18 more cities and 6 more years of data. The data were assembled from publicly available sources: mortality data from the National Center for Health Statistics, weather data from the National Climactic Data Center EarthInfo CD-ROM, and air pollution data from the Environmental Protection Agency's Aerometric Information Retrieval System (AIRS). Metadata and details about how the data were processed can be found at the Internet-based Health and Air Pollution Surveillance System (IHAPSS) website maintained by the Johns Hopkins Bloomberg School of Public Health Department of Biostatistics at http://ihapss.biostat.jhsph.edu/. Although the individual city datasets are already publicly available, it requires a considerable investment in time to assemble and prepare the full database for a multi-city analysis.

The NMMAPSdata package, in addition to providing the NMMAPS database as a single entity, includes functions for building "versions" of the database which may be more suitable for different kinds of analyses. The package is a framework for conducting organized, systematic, and reproducible analyses of time series data on air pollution and mortality. The current version contains daily mortality, air pollution, and weather data for 108 U.S. cities covering the years 1987–2000. Note that although the original NMMAPS originally included morbidity outcomes, those data are *not* provided in the NMMAPSdata package. With the need for reproducible research in epidemiologic studies only increasing, the package has been designed to facilitate and encourage such reproducible analyses. Furthermore, the package simplifies distribution of the data and creates a common platform for disseminating results and methodology. It is worth noting that while NMMAPS is currently the largest database linking daily mortality with air pollution exposures, it is small compared to datasets common to fields such as genomics or remote sensing. As technological innovations make possible collecting larger datasets, the packaging of data with accompanying software becomes ever more relevant to its responsible use and distribution.

In the following sections we present the NMMAPSdata package and provide short tutorials on possible ways to use the package to fit time series models. Section 2 describes the core functionality of the package and describes how one can preprocess the database to prepare it for subsequent analyses. Section 3 presents an analysis of PM_{10} and mortality, similar to those done in previous NMMAPS analyses. In addition we demonstrate how to fit single city and multi-city models. Finally, in Section 4 we give examples of how to fit more complex time series models. These examples reproduce some recent findings (Welty and Zeger, 2004; Peng et al., 2004).

2 Overview of NMMAPSdata

The NMMAPSdata package can be downloaded from the IHAPSS website. Upon installation, the NMMAPSdata package can be loaded into an R session using the library function. The package loads the methods package if it has not already been loaded.

The workhorse function of the package, buildDB, can be used to build databases derived from the full database which are suitable for certain analyses. The function buildDB takes the following

```
> ## Load the NMMAPSdata package

> library(NMMAPSdata)
NMMAPS Data (version 0.3)
    Type '?NMMAPS' for a brief introduction to the NMMAPS database.
    Type 'NMMAPScite()' for information on how to cite 'NMMAPSdata' in
    publications.
>
> args(buildDB)

function (procFunc, dbName,
    path = system.file("db", package = "NMMAPSdata"),
```

options:

`procFunc`: a function. This function is either written by the user or can be one of the built-in functions provided in the `NMMAPSdata` package (see `?preprocessEx` for a list of the built-in functions). This function takes care of the processing of each of the city dataframes and should take a dataframe as its first argument. A possible template of such a function is shown to the right. Sections 3 and 4 give further examples of the kinds of functions that may be useful.

`dbName`: a character string containing the name of the database to be created. If `dbName` is not specified, then the name of the new database is taken from the name of the function in `procFunc` (i.e. via `deparse(substitute(procFunc))`). Note that the value for `procFunc` should not be quoted.

`path`: a character string containing the directory in which the new database will be created. The new database will be in a subdirectory of `path`.

`cityList`: a character vector containing abbreviated names of cities to be included in the new database. The default is `NULL`, in which case all 108 cities will be included.

`compress`: a logical flag indicating whether or not the new database should be stored in a com-

```
    cityList = NULL, compress = FALSE, verbose = TRUE, ...)
NULL
>
myProcFunc <-function(dataframe) {

     ## If not all cities are suitable for this analysis, check
     ## city's dataframe and return NULL if it is not suitable

     ## Process dataframe contents, subset variables,
     ## transformations, etc.

     ## Return modified dataframe
}
```

pressed format. The compression used is the `gzip` algorithm. The default is `FALSE`.

`verbose`: a logical flag indicating whether or not messages should be printed to the screen as the database is being built/processed. The default is `TRUE`.

Here is sample call to `buildDB`. This call creates a new database called `myNewDB` by applying the preprocessing function `myProcFunc` to each city dataframe. All 108 cities are included and the new database is created as a subdirectory of `/home/rpeng` (the subdirectory name is `myNewDB`). The database is stored in a (`gzip`-ed) compressed format and messages will be printed to the screen as the database is being built.

```
> ## Example call to 'buildDB'

> buildDB(procFunc = myProcFunc, dbName = "myNewDB", path = "/home/rpeng",
          cityList = NULL, compress = TRUE, verbose = TRUE)
```

`buildDB`, in addition to building the new database, returns an object of class `NMMAPSdbInfo`. This object contains information about the new database that can be used to reconstruct the database, if necessary. The `NMMAPSdbInfo` object contains the preprocessing function, the original call to `buildDB`, and the environment associated with the preprocessing function. There is a preliminary function `rebuildDB` which can be used to rebuild a database using an `NMMAPSdbInfo` object.

Once a database is constructed using `buildDB`, it is *registered* via call to `registerDB`. When `registerDB` is called with no arguments it sets the full NMMAPS database as the currently

```
> ## Register full database

> registerDB()  ## Same as register(dbName = NULL)
> showDB()
```

registered database. The argument `dbName` can be used to register other databases.

The city dataframes of the currently registered database can be loaded using `loadCity`, `readCity`, and `attachCity`. `loadCity` takes a character argument which is the abbreviated name of a city. If that city's dataframe is included in the database, the dataframe is loaded into the environment specified by the `envir` argument to `loadCity`. The name of the loaded object is the abbreviated city name.

The function `readCity` takes an abbreviated city name as an argument and return's that city's dataframe from the currently registered database. This function may be more useful when programming functions or writing scripts.

The function `attachCity` works much like the function `attach` in that `attachCity` attaches a city's dataframe to the search list. Note that only one city's dataframe can be attached at a time since all of the city dataframes contain the exact same variable names. `attachCity` (as well as `loadCity`) may be more useful during interactive work.

```
Currently using full NMMAPS database
>

> ## Load New York City dataframe

> loadCity("ny")
> ls()
[1] "ny"
> ny[1:5, 1:10]
  city     date dow agecat accident copd cvd death inf pneinf
1   ny 19870101   5      1       10    3  22    73   0      3
2   ny 19870102   6      1        4    4  20    68   0      1
3   ny 19870103   7      1        5    0  17    56   0      3
4   ny 19870104   1      1        5    1  18    55   0      2
5   ny 19870105   2      1        2    2  14    60   0      2
> ## read New York City dataframe

> dframe <- readCity("ny")
> ls()
[1] "dframe" "ny"
> identical(dframe, ny)
[1] TRUE
> attachCity("ny")
> search()
[1] ".GlobalEnv"        "ny"                 "package:NMMAPSdata"
[4] "package:methods"   "package:graphics"   "package:stats"
[7] "package:utils"     "Autoloads"          "package:base"
> cvd[1:10]  # 10 days of CVD mortality counts from New York City
 [1] 22 20 17 18 14 18 17 16 25 20
> death[1:10]  # 10 days of total non-accidental mortality counts
 [1] 73 68 56 55 60 80 64 63 64 65
>
```

3 Analysis of PM_{10} and Mortality

In this Section we demonstrate how to use the `NMMAPSdata` package to do single-city and multi-city analyses of PM_{10} and non-accidental mortality. The models employed are similar to those of Dominici et al. (2002a,b, 2003).

The basic NMMAPS model for a single city is of the following form:

$$\begin{aligned} \mathrm{Y}_t &\sim \mathrm{Poisson}(\mu_t) \\ \log \mu_t &= \mathrm{DOW}_t + \mathrm{AgeCat} + s(\mathrm{temp}_t, df = 6) + s(\mathrm{temp}_{t,1-3}, df = 6) \\ &\quad + s(\mathrm{dewpt}_t, df = 3) + s(\mathrm{dewpt}_{t,1-3}, df = 3) \\ &\quad + s(t,\ df = 7 \times \ \#\ \mathrm{years}) + s(t,\ df = 0.15 \times 7 \times \ \#\ \mathrm{years}) \times \mathrm{AgeCat} \\ &\quad + \beta \mathrm{PM}_t \\ \mathrm{Var}(Y_t) &= \phi \mu_t \end{aligned} \tag{1}$$

where Y_t is the number of non-accidental deaths on day t for a particular age category, AgeCat is an indicator for the age category, temp_t is the average temperature on day t, $\mathrm{temp}_{t,1-3}$ is a running mean of temperature for the previous 3 days, and PM_t is the PM_{10} level for day t. The variables dewpt_t and $\mathrm{dewpt}_{t,1-3}$ are current day and running mean of dewpoint temperature. The age categories used here are ≥ 75 years old, 65–74, and < 65. Each of the temperature and dewpoint temperature variables are related to mortality in a flexible manner via the smooth function $s()$.

Common choices for the smooth functions are natural splines, smoothing splines, and penalized splines. The smoothness of the functions are controlled by the degrees of freedom (*df*) assigned to each function. We use 6 *df* for temperature and 3 *df* for dewpoint temperature. Also included in the model is a smooth function of time t which is used to control for seasonally varying factors (e.g. winter influenza epidemics) and long-term trends in mortality. This smooth function receives 7 *df* per year of data. In addition to the overall smooth function of time, age-specific mortality trends are accounted for via separate smooth functions of time for the older two age categories (65–74 and ≥ 75). Justification for these choices of *df* for controlling for seasonality, long-term trends, and weather can be found in, e.g. Kelsall et al. (1997); Samet et al. (1998).

For doing an analysis of PM_{10} and mortality one can use the built-in `basicNMMAPS` function for processing the data. First, the dataframe is checked to see if it contains any PM_{10} data. If there is no PM_{10} data, then `NULL` is returned and the city is skipped. For cities with PM_{10} data, days with extreme mortality counts are

```
basicNMMAPS <- function (dataframe) {

    if(all(is.na(dataframe[, "pm10tmean"])))
        return(NULL)

    ## Set extreme mortality values to NA
    is.na(dataframe[, "death"]) <- as.logical(dataframe[, "markdeath"])
```

set to `NA` (missing). Then the function coerces the day-of-week and age category variables to be categorical and creates some age category indicators. Finally, a subset of the pollution (seven lags of PM_{10}), weather (temperature and dewpoint), and mortality (total non-accidental deaths, deaths from cardiovascular disease, and deaths from respiratory diseases) variables are taken and the reduced dataframe is returned.

```
    is.na(dataframe[, "cvd"]) <- as.logical(dataframe[, "markcvd"])
    is.na(dataframe[, "resp"]) <- as.logical(dataframe[, "markresp"])

    ## Create age category indicators
    Age2Ind <- as.numeric(dataframe[, "agecat"] == 2)
    Age3Ind <- as.numeric(dataframe[, "agecat"] == 3)

    ## Coerce day-of-week and age category to factors
    dataframe[, "dow"] <- as.factor(dataframe[, "dow"])
    dataframe[, "agecat"] <- as.factor(dataframe[, "agecat"])

    varList <- c("cvd", "death", "resp", "tmpd", "rmtmpd", "dptp", "rmdptp",
                 "time", "agecat", "dow", "pm10tmean",
                 paste("l", 1:7, "pm10tmean", sep = ""))
    data.frame(dataframe[, varList], Age2Ind = Age2Ind, Age3Ind = Age3Ind)
}
```

The new database can be built with a call to `buildDB`. This process takes approximately 16 minutes on a PC equipped with an AMD Athlon XP 2100+ processor running the Microsoft Windows XP operating system. Note that this only needs to be done once, before the analysis. When `buildDB` is finished building the database it calls `registerDB` to make the newly built database the currently registered one.

```
> buildDB(procFunc = basicNMMAPS)

Creating directory C:/PROGRA~1/R/rw1090/library/NMMAPSdata/db/basicNMMAPS
Creating database: basicNMMAPS
Processing cities...
+ akr ---> C:/PROGRA~1/R/rw1090/library/NMMAPSdata/db/basicNMMAPS/akr.rda
+ albu ---> C:/PROGRA~1/R/rw1090/library/NMMAPSdata/db/basicNMMAPS/albu.rda
+ anch ---> C:/PROGRA~1/R/rw1090/library/NMMAPSdata/db/basicNMMAPS/anch.rda
[...OMITTED...]
Saving city information
Registering database location:
C:/PROGRA~1/R/rw1090/library/NMMAPSdata/db/basicNMMAPS
>
```

A listing of the abbreviated names of the cities included in the new database can be retrieved with the `listDBCities` function. `listDBCities` always lists the names of the cities in the currently registered database. Notice that there are

```
> listDBCities()

 [1] "akr"  "albu" "anch" "arlv" "atla" "aust" "bake" "balt" "batr" "bidd"
[11] "birm" "bost" "buff" "cayc" "cdrp" "char" "chic" "cinc" "clev" "clmg"
[21] "clmo" "colo" "corp" "covt" "dayt" "dc"   "denv" "desm" "det"  "dlft"
```

only 102 cities listed — 6 of the 108 cities in the full database do not contain any PM_{10} data.

```
 [31] "elpa" "evan" "fres" "ftwa" "gdrp" "grnb" "hono" "hous" "hunt" "indi"
 [41] "jcks" "jckv" "jers" "kan"  "kans" "king" "knox" "la"   "lafy" "lasv"
 [51] "lex"  "loui" "ltrk" "lubb" "madi" "memp" "miam" "milw" "minn" "mobi"
 [61] "mode" "nash" "no"   "nor"  "nwk"  "ny"   "oakl" "okla" "olym" "oma"
 [71] "orla" "phil" "phoe" "pitt" "prov" "ral"  "rich" "rive" "roch" "sacr"
 [81] "salt" "sana" "sanb" "sand" "sanf" "sanj" "seat" "shr"  "spok" "staa"
 [91] "stlo" "stoc" "stpe" "syra" "taco" "tamp" "tole" "tope" "tucs" "tuls"
[101] "wich" "wor"
>
```

The file `simple.R` can be downloaded from `http://www.ihapss.jhsph.edu/`. The file contains functions for fitting a Poisson regression model of daily non-accidental mortality and PM_{10}.

The function `fitSingleCity` in `simple.R` can be used to fit a Poisson regression model for a single city. It is essentially a wrapper which sets up appropriate formulas for the `glm` call. The function takes as its first argument a city dataframe and there are other arguments for specifying the pollutant to use, the response variable (i.e. cause of death), and the degrees of freedom for the various smooth functions.

```
## Function for fitting NMMAPS model to single city dataframe

fitSingleCity <- function(data, pollutant = "l1pm10tmean", cause = "death",
                          dfyr.Time = 7, pdfyr.time = 0.15, df.Temp = 6,
                          df.Dew = 3, extractors = NULL) {
    ## Argument checking
    stopifnot(is.character(pollutant), is.character(cause), length(cause) == 1)

    ## Modify degrees of freedom based on missingness of data
    sub <- data[, c("time", "agecat", "dow", "tmpd", "rmtmpd", "dptp",
                    "rmdptp", cause, paste(pollutant, collapse = "+"))]
    subset <- complete.cases(sub)
    df.Time <- round( numdf(subset, dfyr.Time) )
    df.time <- round( df.Time * pdfyr.time )

    ## Don't setup smooth function of time where there are incomplete cases
    is.na(data[, "time"]) <- !subset

    modelFormula <- setupFormula(cause, pollutant, df.Time, df.time,
```

Once the file `simple.R` has been sourced, fitting model (1) to a single city's dataframe is straightforward. Here we fit the model to Los Angeles data using previous day PM_{10} (`l1pm10tmean`) as the exposure of interest and total non-accidental mortality as the response. Note that you may need to load the `splines` package if it is not already loaded since natural splines are used to represent the smooth function of time in (1).

The estimate for the lag 1 PM_{10} coefficient is 0.00037 with a standard error of approximately 0.00019. This estimate represents a 0.37% increase in mortality associated with a 10 $\mu g/m^3$ increase in PM_{10}.

```
                                  df.Temp, df.Dew)

    ## Fit the model!
    fit <- glm(modelFormula, family = quasipoisson, data = data,
               control = glm.control(epsilon = 1e-10, maxit = 1000),
               na.action = na.omit)

    ## Extract information from the fitted glm model object using the
    ## list of functions in ‘extractors’.  If no extractors are
    ## specified, just return the entire fitted model object.
    rval <- if(is.null(extractors))
        fit
    else
        lapply(extractors, function(f) f(fit))
    invisible(rval)

}
> ## Fit single city model for Los Angeles

> registerDB("basicNMMAPS")
> loadCity("la")
> fit <- fitSingleCity(data = la, pollutant = "l1pm10tmean", cause = "death")
> summary(fit)

Call:
glm(formula = modelFormula, family = quasipoisson, data = data,
    na.action = na.omit, control = glm.control(epsilon = 1e-10,
        maxit = 1000))

Deviance Residuals:
     Min        1Q    Median        3Q       Max
-3.55527  -0.70388  -0.02116   0.66941   5.18345

Coefficients:
```

Results from single city studies tend to be very noisy and sensitive to model specification. With data from multiple cities, one can combine information across cities to gain power and obtain more stable estimates of the pollution effect.

In the first stage of a multi-city analysis one fits separate Poission regressions models to data from each city. The function `cityApply` can be used to apply a function to all of the cities in the currently registered database. Using `cityApply`, we can apply the function

```
                 Estimate Std. Error t value Pr(>|t|)
(Intercept)     3.9597009  0.1119988  35.355  < 2e-16
dow2            0.0394246  0.0107242   3.676 0.000241
dow3            0.0143218  0.0109448   1.309 0.190791
dow4            0.0090323  0.0107891   0.837 0.402567
dow5            0.0063432  0.0103381   0.614 0.539548
dow6            0.0267107  0.0104684   2.552 0.010777
dow7            0.0145003  0.0107050   1.355 0.175673
agecat2        -0.1468729  0.0568662  -2.583 0.009851
agecat3         0.5916656  0.0476910  12.406  < 2e-16
[...OMITTED...]
l1pm10tmean     0.0003722  0.0001875   1.985 0.047209

(Dispersion parameter for quasipoisson family taken to be 1.126420)

    Null deviance: 28946.9  on 2981  degrees of freedom
Residual deviance:  3190.4  on 2826  degrees of freedom
AIC: NA

Number of Fisher Scoring iterations: 4

>
```

`fitSingleCity` from above to all of cities in the database and obtain coefficient estimates and standard errors for each city. The only complication here is that `fitSingleCity`, by default, returns an entire `glm` object, which in this case can be quite large. Returning over 100 `glm` objects from `cityApply` would likely exhaust the memory on an average computer.

The function `fitSingleCity` has an argument `extractors` which, by default is `NULL`. However, one can pass a list of functions via the `extractors` argument and these functions will be applied to the object returned from the call to `glm`.

```
> ## Extractors for 'fitSingleCity'

> extractFun <- list(coef=function(x) summary(x)$coeff["l1pm10tmean",1],
+                    std=function(x) summary(x)$coeff["l1pm10tmean",2])
```

With the list `extractFun`, `fitSingleCity` will only return the coefficient estimate and standard error for the `l1pm10tmean` variable.

```
> ## Fit single city using extractors

> fit <- fitSingleCity(data = la, pollutant = "l1pm10tmean",
+                       cause = "death", extractors = extractFun)
> fit
$coef
[1] 0.0003722355

$std
[1] 0.0001874974

>
```

Now, fitting the model to all cities simply involves calling `cityApply` with `fitSingleCity` and the list of extractor functions in `extractFun`.

```
> ## Do multi-city analysis

> results <- cityApply(fitSingleCity, extractors = extractFun)
```

The multi-city analysis takes approximately 23 minutes. The object `results` is a list of length

```
> ## Extract coefficients and standard errors
```

100. Each element of `results` is a list containing the coefficient and standard error of the coefficient for a city. After retrieving the city-specific coefficients and standard errors, one can pool them to obtain a "national average" effect. Pooling can be done using either a fixed effects model or a full heirarchical model. Both methods of pooling produce simlar estimates.

```
> beta <- sapply(results, "[[", "coef")
> std <- sapply(results, "[[", "std")

> ## Fixed effects model
> weighted.mean(beta, 1 / std^2)  ## Pooled effect
[1] 0.0001817306
> sqrt(1 / sum(1 / std^2))  ## Standard error
[1] 0.00003721123

> ## Hierarchical normal model (requires TLNise software)
> p <- tlnise(beta, std^2, rep(1, 102), prnt = FALSE)
> p[["gamma"]]
          est            se   est/se
0 0.0001799034 0.00004400010 4.088706
>
```

The TLNise (Two Level Normal independent sampling estimation) software used here for the hierarchical model can be downloaded separately from Philip Everson's website at

`http://www.swarthmore.edu/NatSci/peverso1/TLNise/tlnise.htm`

4 More advanced models

4.1 Analysis of PM_{10} and Mortality Using Distributed Lag Models for Temperature

In this section we provide an example use of the `NMMAPSdata` package by reproducing results from Welty and Zeger (2004), which uses distributed lag models to investigate sensitivity of the PM_{10}-mortality relationship to control for weather and season. We use three R functions — `tdlm4MO`, `tdlmSV`, and `tdlmNL` — to fit the models in Welty and Zeger (2004). These three functions are not included as part of the `NMMAPSdata` package, but are listed in the appendix and are available at IHAPSS. They should be sourced into the R workspace. These functions require that the individual city data frames be set up using the `tempDLM` function for the `procFunc` argument in `buildDB`.

Familiarity with Welty and Zeger (2004) is assumed in the commentary that follows. In brief, Welty and Zeger consider two variations of generalized linear models for mortality on PM_{10} and distributed lags of temperature. The basic form for these models is

$$
\begin{aligned}
\mathrm{Y}_t &\sim \mathrm{Poisson}(\mu_t) \\
\log \mu_t &= \mathrm{DOW}_t + \mathrm{DOM}_t + \mathrm{AgeCat} \\
&\quad + r(\mathrm{dewpt}_t) + r(\mathrm{dewpt}_{t,1-2}) \\
&\quad + s(t,\ df = 4 \times \#\ \text{years}) + s(t,\ df = 1 \times \#\ \text{years}) \times \mathrm{AgeCat} \\
&\quad + \mathrm{temp}_t + \mathrm{temp}_{t,1-2} + \mathrm{temp}_{t,1-7} + \mathrm{temp}_{t,1-14} \\
&\quad + \beta \mathrm{PM}_t \\
\mathrm{Var}(Y_t) &= \phi \mu_t
\end{aligned}
\tag{2}
$$

where temp_t, $\mathrm{temp}_{t,1-2}$, $\mathrm{temp}_{t,1-7}$, and $\mathrm{temp}_{t,1-14}$ are current day temperature, the average of the past two days' temperatures, the average of the past seven days' temperatures, and the average of the past fourteen days' temperatures. The expressions $r(\mathrm{dewpt}_t)$ and $r(\mathrm{dewpt}_{t,1-2})$ denote the residuals from regressing current day dew point and the average of previous two days' dew points on all temperature covariates. The first variation on (2) in Welty and Zeger (2004) is to allow the coefficients on the distributed lags of temperature to vary seasonally. The second variation is to allow for non-linear functions of the distributed lags of temperature. In what follows, we step through the commands to replicate the exploratory analysis in Welty and Zeger (2004) and to fit several of the seasonally varying and non-linear distributed lag models.

After installing the `NMMAPSdata` package, begin by loading the `NMMAPSdata` and `splines` packages. The `splines` package is required for fitting the distributed lag models.

```
## load NMMAPSdata and splines packages

> library(NMMAPSdata)
NMMAPS Data (version 0.3)
    Type '?NMMAPS' for a brief introduction to the NMMAPS database.
    Type 'NMMAPScite()' for information on how to cite 'NMMAPSdata' in
    publications.
>
```

Next, we build the databases for New York and Los Angeles using the processing function `tempDLM` in `buildDB`. Processing with `tempDLM` creates the appropriate dataframes for use with `tdlm4MO`, `tdlmSV`, and `tdlmNL`. These three functions will not work on the default database build.

```
## build databases for ny and la using tempDLM

>  buildDB(procFunc = tempDLM, dbName = "tempDLM",
+           path = "C:/mydoc/nmmaps_pub/R/",
+           cityList = c("ny", "la"))
Creating database: tempDLM
Processing cities...
+ ny ---> C:/mydoc/nmmaps_pub/R//tempDLM/ny.rda
+ la ---> C:/mydoc/nmmaps_pub/R//tempDLM/la.rda
```

The city databases created by `tempDLM` include several new variables not originally part of the NMMAPS database.

`var.lag#`: `var` lagged by `#` days
`var.lmean#`: average of the past `#` days of `var`
`days`: days numbered consecutively (1-5114)
`mon`: month number (1-12)
`yr`: year (1987-2000)
`doy`: day of year (1-365 or 366)
`ndays`: no. of days in the year (either 365 or 366)

The variables `tmpd.lmean2`, `tmpd.lmean7`, and `tmpd.lmean14` in particular are important for the temperature distributed lag models.

```
Saving city information
Registering database location: C:/mydoc/nmmaps_pub/R//tempDLM
 ## examine data for New York City

> data <- readCity("ny")
> data[1:2,]
      date dow agecat death tmpd pm10tmean    dptp dom tmpd.lag1
1 19870101   5      1    73 34.5        NA 33.1875   1        NA
2 19870102   6      1    68 36.5        NA 29.8125   2      34.5
  tmpd.lag2 tmpd.lag3 tmpd.lag4 tmpd.lag5 tmpd.lag6 tmpd.lag7
1        NA        NA        NA        NA        NA        NA
2        NA        NA        NA        NA        NA        NA
  tmpd.lag8 tmpd.lag9 tmpd.lag10 tmpd.lag11 tmpd.lag12 tmpd.lag13
1        NA        NA         NA         NA         NA         NA
2        NA        NA         NA         NA         NA         NA
  tmpd.lag14 tmpd.lag15 tmpd.lmean2 tmpd.lmean7 tmpd.lmean14
1         NA         NA          NA          NA           NA
2         NA         NA          NA          NA           NA
  pm10.lag1 pm10.lag2 pm10.lag3 pm10.lag4 pm10.lag5 pm10.lag6
1        NA        NA        NA        NA        NA        NA
2        NA        NA        NA        NA        NA        NA
  pm10.lag7 pm10.lag8 pm10.lag9 pm10.lag10 pm10.lag11 pm10.lag12
1        NA        NA        NA         NA         NA         NA
2        NA        NA        NA         NA         NA         NA
  pm10.lag13 pm10.lag14 dptp.lag1 dptp.lag2 dptp.lag3 dptp.lmean2
1         NA         NA        NA        NA        NA          NA
2         NA         NA   33.1875        NA        NA          NA
  days mon   yr doy ndays
1    1   1 1987   1   365
2    2   1 1987   2   365

 ## fit tdlm4MO to New York data

> results <- tdlm4MO(data)
```

We may now re-create the exploratory analyses of the mortality – temperature relationship presented in Welty and Zeger, *Distributed lag* *models for temperature* section, for New York City. The function `tdlm4MO` fits a quasipoisson GLM with distributed lags of temperature on log expected mortality over moving four month windows of the data. `tdlm4MO` requires the city dataframe as an argument, and returns a matrix with 5 columns. The first column designates the center month of the four month period (3 for March of 1987, 15 for March of 1988, etc.); `thet0` is the coefficient for `tmpd`, `thet1` is the coefficient for `tmpd.lmean2`, and so on. For additional details on month numbering or the thetas see the comments in `tdlm4MO`.

```
> results[1:4,]
  month       thet0        thet1        thet2         thet3
1     3 0.001295988 0.0002889866 -0.005356287  0.0008976634
2     4 0.003991299 0.0007882620 -0.003133897 -0.0028361116
3     5 0.002467549 0.0039741800 -0.001541874 -0.0055862979
4     6 0.003422143 0.0054526861  0.001162100 -0.0113166819
[...]
```

It is now possible to re-create Figure 2 of Welty and Zeger (2004).

```
 ## boxplot of mortality attributable to temperature by month

> plot(results$month - 12 * (results$month - 1) %/% 12,
+        1000 * rowSums(results[,2:5]))
```

We now switch to Los Angeles and fit several of the seasonally varying distributed lag models, using the function `tdlmSV`. The function `tdlmSV` takes as arguments the city-specific data frame (created using `buildDB` with `procFunc = tempDLM`), degrees of freedom for the smooth function of time, degrees of freedom for the time by age category interaction, the lag of PM_{10} to use as exposure, and whether or not to include interactions of the distributed lags of temperature. `tdlmSV` returns a list with the PM_{10} coefficient and its standard error as well as the full model. Note that degrees of freedom are specified as total degrees of freedom, not as degrees of freedom per year. The code to the right

```
 ## seasonally varying temperature dist lag models for la

> data <- readCity("la")
> results <- tdlmSV(data = data, degf = 28, degfage = 14,
+                       pol.lag = 1, inter = FALSE)
> names(results)
[1] "pol.ef" "pol.se" "model"
> results$pol.ef
    Estimate
0.0006091162
> results$pol.se
  Std. Error
0.0001949552
```

fits the model referred to as SV_2 in Welty and Zeger (2004) for Los Angeles.

We may analogously fit the model SV_8^I for Los Angeles. The results for SV_2 above and SV_8^I match those presented in Figure 4 of Welty and Zeger (2004).

```
## another seasonally varying model for la

>   results <- tdlmSV(data = data, degf = 112, degfage = 14,
+                     pol.lag = 1, inter = FALSE)
>   results$pol.ef
    Estimate
0.0003023730
```

We now fit two models with non-linear distributed lags of temperature, $NL_4^I(1,4)$ and $NL_4^I(2,4)$, using `tdlmNL`. Similar to `tdlmSV` (but with slightly different names), `tdlmNL` takes as arguments the city-specific data frame, degrees of freedom for smooths of time and time by age category, and the lag of PM_{10} exposure. Again note that the smoothness of the time and time by age category splines are specified by their total degrees of freedom, rather than by degrees of freedom per year. The arguments `nlags` and `df.dlm` designate the number of distributed lags to include in the model and their (natural spline) degrees of freedom. The comments in `tdlmNL` provide greater detail.

```
## non linear temperature dist lag models for la

>   results <- tdlmNL(data = data, nlags = 1, df.dlm = 4,
+                     pol.lag = 1, df.time = 56, degfage = 14,
+                     inter = TRUE)
>   results$pol.ef
    Estimate
0.0003751928
>   results <- tdlmNL(data = data, nlags = 4, df.dlm = 2,
+                     pol.lag = 1, df.time = 56, degfage = 14,
+                     inter = TRUE)
>   results$pol.ef
    Estimate
0.0005284077
```

Lastly, in preparation for obtaining national average PM_{10} coefficient estimates, we create a table with city name and the corresponding PM_{10} coefficient and standard error, for as many cities in NMMAPS as possible. The function `tempDLMCities` with argument `pmmiss = TRUE` returns the list of cities used in Welty and Zeger

```
## list of all cities with sufficient PM10

>   cities <- tempDLMCities(pmmiss = TRUE)
>   cities
  [1] "akr"  "albu" "anch" "arlv" "atla" "aust" "bake" "balt" "batr"
 [10] "birm" "bost" "buff" "cayc" "cdrp" "char" "chic" "cinc" "clev"
 [19] "clmg" "clmo" "colo" "corp" "covt" "dayt" "dc"   "denv" "desm"
```

(2004) to create such a table. These cities contain PM_{10} at least every sixth day for more than four years. Welty and Zeger (2004) uses the table information to fit normal hierarchical models (using `TLNise`) that estimate a national average PM_{10} coefficient. The code for `TLNise` is not provided with the NMMAPS package or on the IHAPSS website, but the source is available on the web (see reference in Section 3). More information on the normal hierarchical model can be found in Everson and Morris (2000).

```
 [28] "det"  "dlft" "elpa" "evan" "fres" "ftwa" "gdrp" "grnb" "hono"
 [37] "hous" "hunt" "indi" "jcks" "jckv" "jers" "kan"  "kans" "king"
 [46] "knox" "la"   "lasv" "lex"  "loui" "ltrk" "lubb" "madi" "memp"
 [55] "miam" "milw" "minn" "mobi" "mode" "nash" "no"   "nor"  "nwk"
 [64] "ny"   "oakl" "okla" "olym" "oma"  "orla" "phil" "phoe" "pitt"
 [73] "prov" "ral"  "rich" "rive" "roch" "sacr" "salt" "sana" "sanb"
 [82] "sand" "sanf" "sanj" "seat" "shr"  "spok" "staa" "stlo" "stoc"
 [91] "stpe" "syra" "taco" "tamp" "tole" "tope" "tucs" "tuls" "wich"
[100] "wor"
```

Building the databases for `cities` takes approximately 13 minutes on a Pentium M 1.7 GHz processor running Microsoft Windows XP.

```
 ## build databases for ‘cities’

> buildDB(procFunc = tempDLM, dbName = "tempDLM",
+         path = "C:/mydoc/nmmaps_pub/R/", cityList = cities)
+ Creating database: tempDLM Processing cities...
+ akr ---> C:/mydoc/nmmaps_pub/R//tempDLM/akr.rda
+ albu ---> C:/mydoc/nmmaps_pub/R//tempDLM/albu.rda
+ anch ---> C:/mydoc/nmmaps_pub/R//tempDLM/anch.rda

[...]

+ wich ---> C:/mydoc/nmmaps_pub/R//tempDLM/wich.rda
+ wor ---> C:/mydoc/nmmaps_pub/R//tempDLM/wor.rda
Saving city information Registering database location:
C:/mydoc/nmmaps_pub/R//tempDLM
```

The loop to the right computes the pollution coefficient from $\mathrm{NL}_4^I(3,2)$ for each city and stores the city name and the coefficient and its standard error in the table `results.table`. The loop takes approximately 14 minutes. The right two columns of the table (after appropriate coer-

```
 # fitting NL(3,2) to all cities

>   for (i in 1:ncity) {
+     results.table[i,1] <- cities[i]
+     data <- readCity(cities[i])
+     results <- tdlmNL(data = data, nlags = 3, df.dlm = 2,
```

cion to numeric type) may be used in a normal hierarchical model as in TLNise to estimate a national pollution coefficient.

```
+                         pol.lag = 1, df.time = 56, degfage = 14,
+                         inter = TRUE)
+     results.table[i,2:3] <- c(results$pol.ef,results$pol.se)
+   }
> results.table
       [,1]   [,2]                   [,3]
  [1,] "akr"  "0.000726542986445253"  "0.000771758494338229"
  [2,] "albu" "-0.0002369830547286"   "0.000527971772081411"
  [3,] "anch" "-4.22076249831615e-05" "0.000197063679738634"

    [...]

 [98,] "tuls" "0.000483106752628049"  "0.000936662050099774"
 [99,] "wich" "0.000101311205052070"  "0.000878399055295233"
[100,] "wor"  "0.00142129615964940"   "0.00101806822667225"
```

4.2 Seasonal Analyses of PM_{10} and Mortality

In this section we recreate some of the analyses presented in Peng et al. (2004). The models presented there incorporate seasonal interactions of the PM_{10} series with periodic functions of time, allowing for seasonal variation of the effect of PM_{10} on mortality. Such seasonal variation is plausible because of the changing sources of PM throughout the year. The potential confounders are the same as in model (1) however now the effect of PM_{10} on mortality is a periodic function of time:

$$\begin{aligned} \log \mu_t &= \beta(t) \times \mathrm{PM}_t + \text{confounders} \\ &= [\beta_0 + \beta_1 \sin(2\pi t/365) + \beta_2 \cos(2\pi t/365)] \times \mathrm{PM}_t + \text{confounders} \end{aligned}$$

where $\beta_0, \beta_1, \beta_2$ are coefficients to be estimated.

The file seasonal.R is available from the IHAPSS website and contains functions for fitting the seasonal interaction model described above. The function fitCitySeason is reproduced to the right and follows a similar format to the fitSingleCity function described in Section 3. The NMMAPSdata package comes with a

```
## Function for fitting single city seasonal interaction model

fitCitySeason <- function(data, pollutant = "l1pm10tmean", cause = "death",
                          season = c("none", "smooth", "stepfun"),
                          tempModel = c("default", "rm7", "tempInt"),
                          dfyr.Time = 7, pdfyr.time = 0.15, df.Temp = 6,
                          df.Dew = 3, df.Season = 1, extractors = NULL) {
```

built-in preprocessing function `seasonal` which can be used to prepare the NMMAPS database for fitting such seasonal models.

The function `setupTemp` can be used to setup the alternate temperature models used in Peng et al. (2004) for sensitivity analyses. For now we will use the default temperature model.

```
season <- match.arg(season)
tempModel <- match.arg(tempModel)

## Specify temperature model

temp.info <- setupTemp(data, df.Temp, tempModel)
data <- temp.info[["adj.data"]]
temp.f <- temp.info[["temp.f"]]

## Need to modify degrees of freedom based on missingness of data
sub <- data[, c("time", "agecat", "tmpd", "rmtmpd", "dptp", "rmdptp",
                temp.info[["addedVars"]], cause,
                paste(pollutant, collapse = "+"))]
subset <- complete.cases(sub)
df.Time <- round( numdf(subset, dfyr.Time) )
df.time <- round( df.Time * pdfyr.time )

## Don't setup smooth function of time where there are incomplete cases
is.na(data[, "time"]) <- !subset

## Setup other formulas
covariates.f <- paste(cause, "~ dow + agecat + Season")
weather.f <- paste(c(paste("ns(dptp,", df.Dew, ")"),
                     paste("ns(rmdptp,", df.Dew, ")")),
                   collapse = " + ")

## Setup smooth functions of time
time.f <- paste(c(paste("ns(time,", df.Time, ")"),
                  paste("I(ns(time,", df.time, ")*Age2Ind)"),
                  paste("I(ns(time,", df.time, ")*Age3Ind)")),
                collapse = "+")
```

In this portion of the function we setup the model formulas for the seasonal interaction with PM_{10}. There are two choices: `smooth`, which

```
if(season == "none")
    poll.f <- paste(pollutant, collapse = " + ")
```

uses the sine/cosine basis and `stepfun`, which uses a simple step function to model the seasonal effects. The argument `df.Season` indicates how many sine/cosine terms to include in the model (defaults to one of each). The function `periodicBasis` is reproduced in the Appendix and is needed to construct the sine/cosine basis which interacts with the PM_{10} variable.

```
    else {
        poll.f <-
            switch(season,
                   smooth = {
                       nfreq <- df.Season
                       paste("periodicBasis(SeasonTime,", nfreq,
                             ",365, intercept = TRUE):", pollutant)
                   },
                   stepfun = paste("Season", pollutant, sep = ":")

                   )
    }
```

Once the formulas have been constructed the `glm` function is called to fit the model. Much like `fitSingleCity`, a list of extractor functions can be passed if one does not want to return the entire `glm` object. This will most likely be useful when doing multi-city analyses.

```
    form.str <- paste(covariates.f, time.f, temp.f, weather.f,
                      poll.f, sep = " + ")
    modelFormula <- as.formula(form.str)

    ## Fit the model!
    fit <- glm(modelFormula, family = quasipoisson, data = data,
               control = glm.control(epsilon = 1e-10, maxit = 1000),
               na.action = na.omit)

    rval <- if(is.null(extractors))
        fit
    else
        lapply(extractors, function(f) f(fit))
    invisible(rval)
}
```

We can use the `fitCitySeason` function to do a seasonal analysis of the Detroit data. After loading the city dataframe with `loadCity`, we call `fitCitySeason` and specify the pollutant to be lag 1 PM_{10}, the outcome is total non-accidental mortality, and the seasonal model is `smooth`.

```
> ## Seasonal analysis of Detroit

> loadCity("det")
> fit <- fitCitySeason(data = det, pollutant = "l1pm10tmean",
+                      cause = "death", season = "smooth")
> b <- coef(fit)  ## Get estimated coefficients
> B <- periodicBasis(1:365, nfreq = 1, period = 365, intercept = TRUE)
```

`fitCitySeason` returns the full `glm` object by default, from which we extract the coefficients. To visualize the estimated seasonal effect, we can reconstruct the sine/cosine basis with the `periodicBasis` function. We can plot pointwise standard errors by extracting the variance-covariance matrix from the `glm` object.

```
> seas.curve <- B %*% b[grep("pm10", names(b))]  ## Estimated curve

> V <- vcov(fit)
> i <- grep("pm10", colnames(V))
> std <- sqrt(diag(B %*% V[i, i] %*% t(B)))

> ## Plot estimated curve +/- 2 standard errors (pointwise)
> mat <- cbind(seas.curve, seas.curve - 2*std, seas.curve + 2*std)
> matplot(mat, type = "l", lty = c(1, 2, 2), col = 1)
```

A plot of the estimated curve (not shown) indicates a peak in the effect of PM_{10} on mortality in the summer and decrease during the winter. This pattern is consistent with the other midwest and northeast cities in the database.

5 Bug Reports

Please send any bug reports or comments to `rpeng@jhsph.edu`.

6 Acknowledgments

This research was supported in part by NIH/NHLBI grant T32HL07024, the CDC Center for Excellence in Environmental Public Health Tracking (CDC grant U50CCU322417) at Johns Hopkins Bloomberg School of Public Health, NIEHS grant R01ES012054, and Health Effects Institute grant HEI025. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of these funding agencies

References

Burnett, R. T., Cakmak, S., and Brook, J. R. (1998), "The Effect of the Urban Ambient Air Pollution Mix on Daily Mortality Rates in 11 Canadian Cities," *Canadian Journal of Public Health*, 89, 152–156.

Daniels, M. J., Dominici, F., Zeger, S. L., and Samet, J. M. (2004), *The National Morbidity, Mortality, and Air Pollution Study, Part III: Concentration-Response Curves and Thresholds for the 20 Largest US Cities*, Health Effects Institute, Cambridge MA.

Dominici, F., Daniels, M., Zeger, S. L., and et al. (2002a), "Air Pollution and Mortality: Estimating Regional and National Dose-Response Relationships," *Journal of the American Statistical Association*, 97, 100–111.

Dominici, F., McDermott, A., Zeger, S. L., and et al. (2002b), "On the Use of Generalized Additive Models in Time-Series Studies of Air Pollution and Health," *American Journal of Epidemiology*, 156, 193–203.

Dominici, F., McDermott, A., Zeger, S. L., and et al. (2003), "Mortality Among Residents of 90 Cities," in *Revised Analysis of the National Morbidity Mortality Air Pollution Study: Part II*, The Health Effects Institute, Cambridge, MA, pp. 9–24.

Everson, P. J. and Morris, C. N. (2000), "Inference for Multivariate Normal Hierarchical Models," *Journal of the Royal Statistical Society, Series B*, 62, 399–412.

Katsouyanni, K., Toulomi, G., Samoli, E., Gryparis, A., LeTertre, A., Monopolis, Y., Rossi, G., Zmirou, D., Ballester, F., Boumghar, A., and Anderson, H. R. (2001), "Confounding and effect modification in the short-term effects of ambient particles on total mortality: Results from 29 European cities within the APHEA2 project," *Epidemiology*, 12, 521–513.

Kelsall, J. E., Samet, J. M., Zeger, S. L., and Xu, J. (1997), "Air Pollution and Mortality in Philadelphia, 1974–1988," *American Journal of Epidemiology*, 146, 750–762.

Peng, R. D., Dominici, F., Pastor-Barriuso, R., Zeger, S. L., and Samet, J. M. (2004), "Seasonal Analyses of PM_{10} and Mortality," Tech. Rep. 41, Johns Hopkins University Department of Biostatistics, `http://www.bepress.com/jhubiostat/paper41`.

Samet, J., Zeger, S., Kelsall, J., and et al. (1998), "Does Weather Confound or Modify the Association of Particulate Air Pollution with Mortality?" *Environmental Research, Section A*, 77, 9–19.

Samet, J. M., Dominici, F., Zeger, S. L., Schwartz, J., and Dockery, D. W. (2000a), *The National Morbidity, Mortality, and Air Pollution Study, Part 1: Methods and Methodological Issues*, Health Effects Institute, Cambridge MA.

Samet, J. M., Zeger, S. L., Dominici, F., Curriero, F., Coursac, I., Dockery, D. W., Schwartz, J., and Zanobetti, A. (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.

Samoli, E., Schwartz, J., Wojtyniak, B., Touloumi, G., Spix, C., Balducci, F., and Medina, S. (2002), "Investigating regional differences in short-term effects of air pollution on daily mortality in the APHEA project: a sensitivity analysis for controlling long-term trends and seasonality." *Environmental Health Perspectives*, 109, 349–353.

Samoli, E., Touloumi, G., Zanobetti, A., Le Tertre, A., Schindler, C., Atkinson, R., Vonk, J., Rossi, G., Saez, M., Rabczenko, D., Schwartz, J., and Katsouyanni, K. (2003), "Investigating the dose-response relation between air pollution and total mortality in the APHEA-2 multicity project," *Occupational and Environmental Medicine*, 60, 977–982.

Welty, L. J. and Zeger, S. L. (2004), "Flexible Distributed Lag Models: Are the Acute Effects of PM_{10} on Mortality the Result of Inadequate Control for Weather and Season?" Tech. Rep. 38, Johns Hopkins University Department of Biostatistics, http://www.bepress.com/jhubiostat/paper38.

A Code for Section 3: Analysis of PM_{10} and Mortality

```
setupFormula <- function(cause, pollutant, df.Time, df.time, df.Temp, df.Dew) {
    covariates.f <- paste(cause, "~ dow + agecat")
    weather.f <- paste(c(paste("ns(tmpd,", df.Temp, ")"),
                         paste("ns(rmtmpd,", df.Temp, ")"),
                         paste("ns(dptp,", df.Dew, ")"),
                         paste("ns(rmdptp,", df.Dew, ")")),
                       collapse = "+")
    poll.f <- paste(pollutant, collapse = "+")
    time.f <- paste(c(paste("ns(time,", df.Time, ")"),
                      paste("I(ns(time,", df.time, ")*Age2Ind)"),
                      paste("I(ns(time,", df.time, ")*Age3Ind)")),
                    collapse = "+")
    form.str <- paste(c(covariates.f, time.f, weather.f, poll.f),
                      collapse = "+")
    as.formula(form.str)
}

## Return a discounted df (based on 12 consecutive days of missings)

numdf <- function(usedata, num = 7){
    n <- length(usedata)
    use <- usedata[1:(n/3)]
    ll  <- round(length(use)/12)

    ## this is to eliminate the warning message the length of use is
    ## not a multiple of 12
    usenew <- use[1:(ll*12)]
```

```
    mat <- matrix(usenew, ncol=12,byrow=T)
    m   <- sum(ceiling(apply(mat,1,sum,na.rm=T)/12)) ##-365.25/12
    df  <- round(12*m/365.25*num)
    max(1,df)
}
```

B Code for Section 4.1: Analysis of PM_{10} and Mortality Using Distributed Lag Models for Temperature

```
tdlm4MO <- function(data) {
######################################################################
## quasipoisson glm of log mean mortality on distributed lags of
## temperature (plus dow, dom, and age cat), est over four month
periods ## as described in (insert reference) ## returns dataframe
of coefficients indexed by the third ## month in the four month
period, i.e. temperature dist lag coefficients ## for January -
April of the first year of the data ## will be identified by month
= 3, those for February - May of the first year ## will be
identified by month = 4, and those for January - April of the ##
second year will be identified by month = 15, etc. ## coefficients
for tmpd, tmpd.lmean2, tmpd.lmean7, and tmpd.lmean14 are ##
referred to as thet0, thet1, thet2, and thet3, respectively
######################################################################

  ## create variable month, consecutive numbering of months in data set
  month <- as.numeric(substring(data$date, 5,6))
  month <- month + (as.numeric(substring(data$date,1,4)) - 1987) * 12

  # create matrix of dist lag temperature coeffs estimated over 4 month periods
  # coefficients are identified by third month out of four month period

  coeffs <- matrix(nrow = max(month) - 3, ncol = 5)
  for (i in 1:(max(month)-3)) {
    mod <- glm(death ~ as.factor(agecat) + as.factor(dow) + dom +
               tmpd + tmpd.lmean2 + tmpd.lmean7 + tmpd.lmean14,
```

```
              family = quasipoisson, na.action = na.omit, data = data,
              subset = (month == i | month == (i+1) | month == (i+2) |
                        month == (i + 3)))
    coeffs[i,] <- c(i+2,  mod$coeff[substring(names(mod$coeff),1,4) == "tmpd"])
  }

  coeffs <- data.frame(coeffs)
  names(coeffs) <- c("month", "thet0", "thet1", "thet2", "thet3")

  return(coeffs)
}

tdlmSV <- function(data, degf, degfage, pol.lag = -1, inter =
FALSE) {
######################################################################
## quasipoisson glm of log mean mortality on distributed lags of
## temperature + covariates (possibly a lag of pm10) as described
in ## (insert reference) ## dist lag coefficients have
temporal:seasonal variation ## degf is total df for smooth of time
covariate (usually 1,2,4 or 8/year) ## degfage is total df for
interaction of agecat and time (usually 1/year) ## pol.lag must be
an integer identifying lag of PM10 to include ## pol.lag = -1
(default) is model without any PM10 covariate ## inter = T
includes interactions of temperature covariates, ## up to avg of
past week ## function returns list of pm10 coefficient and std
error (NA's if ## no pm10 in model) and model output from call to
'glm'
######################################################################

  ## basis for long term time and seasonality main effects
  bas0 <- ns(data$days, df = degf)

  ## basis for seasonality in dist lag temperature coeffs, using harmonics
  bas1 <- cbind(sin(2 * pi * data$doy / data$ndays),
                cos(2 * pi * data$doy / data$ndays),
                sin(2 * pi * data$doy / (data$ndays/2)),
                cos(2 * pi * data$doy / (data$ndays/2)))

  ## basis for time trends in dist lag temperature coefficients
```

```
bas2 <- ns(data$days, df = 3)

## basis for interaction btw season and time in dist lag temperature coeffs
bas3 <- matrix(nrow = nrow(bas1), ncol = 12)
for (i in 1:ncol(bas1)) {
  for (j in 1:ncol(bas2)) {
    bas3[,(i-1)*ncol(bas2) + j] = bas1[,i]*bas2[,j]
  }
}

## temperature covariates for model
t.base <- c("tmpd", "tmpd.lmean2", "tmpd.lmean7", "tmpd.lmean14")
tmpd.vars <- c(t.base,
               paste(t.base, ":bas1", sep = ""),
               paste(t.base, ":bas2", sep = ""),
               paste(t.base, ":bas3", sep = ""))

## interactions of dist lags of temperature, if appropriate
if (inter == TRUE) {
  t.inter <- c("tmpd:tmpd.lmean2", "tmpd:tmpd.lmean7",
               "tmpd.lmean2:tmpd.lmean2:tmpd.lmean7")
  tmpd.vars <- c(tmpd.vars, t.inter,
               paste(t.inter, ":bas1", sep = ""),
               paste(t.inter, ":bas2", sep = ""),
               paste(t.inter, ":bas3", sep = ""))
}

## compute residualized dew point covariates to include in model

mod.form <- formula(paste("dptp ~", paste(tmpd.vars, collapse = " + ")))
mod <- lm(mod.form, data = data, na.action = na.omit)

r.dptp <- vector(length = nrow(data))
r.dptp[as.numeric(names(residuals(mod)))] <- as.numeric(residuals(mod))
r.dptp[as.numeric(unique(mod$na.action))] <-
  rep(NA, length(unique(mod$na.action)))

mod.form <- formula(paste("dptp.lmean2 ~",
                          paste(tmpd.vars, collapse = " + ")))
```

```
mod <- lm(mod.form, data = data, na.action = na.omit)

r.dptp.lmean2 <- vector(length = nrow(data))
r.dptp.lmean2[as.numeric(names(residuals(mod)))] <-
  as.numeric(residuals(mod))
r.dptp.lmean2[as.numeric(unique(mod$na.action))] <-
  rep(NA, length(unique(mod$na.action)))

## spline for interaction between age category and time
agetime <- ns(data$days, degfage)

## list covariates to go in model
mod.vars <- c("as.factor(agecat)", "as.factor(agecat):agetime",
              "as.factor(dow)", "dom", "bas0", tmpd.vars,
              "r.dptp", "r.dptp.lmean2")

## set up and include pollution variable if appropriate
if (pol.lag > 0) {
  p.var <- paste("pm10.lag", pol.lag, sep = "")
  mod.vars <- c(p.var, mod.vars)
}
if (pol.lag == 0) {
  p.var <- "pm10tmean"
  mod.vars <- c(p.var, mod.vars)
}

## fit model
mod.form <- formula(paste("death ~", paste(mod.vars, collapse = " + ")))
mod <- glm(mod.form, data = data, family = quasipoisson,
           na.action = na.omit)

## extract pollution coeff and se, if applicable
pol.res <- c(NA, NA)
if (pol.lag >= 0) {
  pol.res <- summary(mod)$coefficients[names(mod$coeff)[!is.na(mod$coeff)] ==
                          p.var,1:2]
}

return(list(pol.ef = pol.res[1], pol.se = pol.res[2], model = mod))
```

```
}


tdlmNL <- function(data, nlags, df.dlm, pol.lag, df.time, degfage,
                  inter = FALSE) {
#######################################################################
## quasipoisson glm of log mean mortality on smooths of
distributed lags of ## temperature + PM10(possibly lagged) +
covaraites, as described in ## (insert reference) ## nlags is how
many of tmpd, tmpd.lmean2, tmpd.lmean7, tmpd.lmean14 ## to include
(i.e. nlags = 2 means use tmpd and tmpd.lmean2) ## df.dlm
determines the df of smooth function of the distributed lags ##
(usually either 1, 2, 3, or 4) ## pol.lag is pollution lag to
include in the model (must be 0, 1, or 2) ## df.time is total
degrees of freedom in smooth of time covariate (usu 4/yr) ##
degfage is total df for interaction of agecat and time (usually
1/year) ## inter is logical, TRUE includes appropriate
interactions up to tmpd.lmean7 ## returns pollution effect
estimate and standard error ## in addition to output of call to
'glm'
#######################################################################

  ## potential distributed lags of temperature to use in model
  t.lags <- c("tmpd", "tmpd.lmean2", "tmpd.lmean7", "tmpd.lmean14")

  ## potential lags of pollution to use in model
  p.lags <- c("pm10tmean", "pm10.lag1", "pm10.lag2")
  pol.lag <- pol.lag + 1 # lag zero corresponds to first elt, etc.

  ## temp variables, as nat spline w/ df = df.tmpd
  tmpd.vars <- paste("ns(", t.lags[1:nlags], ", df = ", df.dlm, ")",
                     sep = "")

  ## interactions up through through tmpd.lmean7
  if (inter == TRUE) {
    mlag <- min(nlags, 3)
    if (mlag > 1) {
      for(i in 1:(mlag-1)) {
```

```
      for (j in (i+1):(mlag)) {
        tmpd.vars <- c(tmpd.vars,
                       paste("ns(I(", t.lags[i], "*", t.lags[j], "), df = ",
                             df.dlm, ")", sep = ""))
      }
    }
  }
}

## residualized dptp to go in model
mod.form <- formula(paste("dptp ~", paste(tmpd.vars, collapse = " + ")))
mod <- lm(mod.form, data = data, na.action = na.omit)

r.dptp <- vector(length = nrow(data))
r.dptp[as.numeric(names(residuals(mod)))] <- as.numeric(residuals(mod))
r.dptp[as.numeric(unique(mod$na.action))] <-
  rep(NA, length(unique(mod$na.action)))

mod.form <- formula(paste("dptp.lmean2 ~",
                          paste(tmpd.vars, collapse = " + ")))
mod <- lm(mod.form, data = data, na.action = na.omit)

r.dptp.lmean2 <- vector(length = nrow(data))
r.dptp.lmean2[as.numeric(names(residuals(mod)))] <-
  as.numeric(residuals(mod))
r.dptp.lmean2[as.numeric(unique(mod$na.action))] <-
  rep(NA, length(unique(mod$na.action)))

## base variables in model
mod.vars <- c("as.factor(agecat)", "as.factor(dow)", "dom",
              paste("as.factor(agecat):ns(days,", degfage, ")"),
              paste("ns(days, df = ", df.time, ")", sep = ""))

## add temperature, dew point vars to model
mod.vars <- c(mod.vars, tmpd.vars, "r.dptp", "r.dptp.lmean2")

## add pollution variable
mod.vars <- c(mod.vars, p.lags[pol.lag])
```

```
  ## specify and fit model
  mod.form <- formula(paste("death ~", paste(mod.vars, collapse = " + ")))
  mod <- glm(mod.form, data = data, family = quasipoisson,
             na.action = na.omit)

  if (mod$converged == FALSE) {
    print("error: glm did not converge")
    return(list(pol.ef = NA, pol.se = NA, model = NA))
  }
  else {
    pol.res <- summary(mod)$coefficients[names(mod$coeff)[!is.na(mod$coeff)] ==
                         p.lags[pol.lag],1:2]
    return(list(pol.ef = pol.res[1], pol.se = pol.res[2],
                model = mod))
  }
}
```

C Code for Section 4.2: Seasonal Analyses

```
periodicBasis <- function(x, nfreq, period, intercept = FALSE) {
    pi <- base::pi
    stopifnot(nfreq > 0)
    x <- as.numeric(x)
    nax <- is.na(x)
    if (nas <- any(nax))
        x <- x[!nax]
    N <- seq(0, nfreq - 1)
    k <- 2^N * 2 * pi / period
    M <- outer(x, k)
    sinM <- apply(M, 2, sin)
    cosM <- apply(M, 2, cos)

    if(!intercept)
        cbind(sinM, cosM)
```

```
    else
        cbind(1, sinM, cosM)
}

setupTemp <- function(dataframe, df.Temp, tempModel) {
    default.temp.f <- paste(c(paste("ns(tmpd,", df.Temp, ")"),
                              paste("ns(rmtmpd,", df.Temp, ")")),
                            collapse = " + ")
    orig.namelist <- names(dataframe)

    temp.f <- switch(tempModel,
                     default = default.temp.f,
                     rm7 = paste(default.temp.f, "ns(rm7tmpd, 3)", sep = "+"),
                     tempInt = {
                         paste("ns(tmpd,", df.Temp, "):",
                               "ns(rmtmpd,", df.Temp, ")", sep = "")
                     })
    list(adj.data = dataframe, temp.f = temp.f,
         addedVars = setdiff(names(dataframe), orig.namelist))
}

## Function ‘numdf’ can be found in the file ‘simple.R’
```