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*

**Suggested Citation**  
http://biostats.bepress.com/jhubiostat/paper44
The original NMMAPS examined the relationships between daily mortality/morbidity and air pollution. The data were assembled from publicly available sources: mortality data from the National Center for Health Statistics, weather data from the National Climatic Data Center, and air pollution data from the Environmental Protection Agency's Aerometric Information Retrieval System (AIRS). The NMMAPS database contains daily measurements on over 70 different variables.

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 mortality and morbidity. Multi-city studies are more complex in multi-city studies than in single city studies. The development of software tools to manage, manipulate, and analyze the data becomes more important. 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.

1.1 Data Sources

The data in NMMAPS were collected from various sources. Mortality data were obtained from the National Center for Health Statistics, weather data from the National Climatic Data Center, and air pollution data from the Environmental Protection Agency's Aerometric Information Retrieval System (AIRS).

1.2 Data Processing

The AIRS data were processed to include 18 more cities and 6 more years of data. The data were then assembled into a structured database using the R programming language. The R package NMMAPS 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 use the package to reproduce recent findings based on the NMMAPS data.

Abstract

The NMMAPS data package contains daily mortality and air pollution data, and weather data, originally assembled as part of the National Mortality and Air Pollution Study.
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 NMMAPS data 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 NMMAPS Data (version 0.3) package loads the NMMAPS database 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 that are suitable for certain kinds of analyses. The function `buildDB()` takes the following arguments:

- `procFunc` (character): the name of the processing function to be used.
- `dbName` (character): the name of the database.
- `path` (character): the path to the NMMAPS database.
- `filename` (character): the name of the file containing the database.
- `procArgs` (list): a list of arguments to be passed to the processing function.

In the following sections, we present the core functionality of the package and provide short introductions to possible ways of using the package to fit time series models. Section 2 describes how to 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 one can preprocess the database to prepare it for more complex analyses, such as those required to fit more complex time series models. These examples reproduce some recent findings (Welty and Zeger, 2004; Peng et al., 2004).
options:

- **cityList**: NULL, compress = FALSE, verbose = TRUE, ...

```r
myProcFunc <- function(dataframe) {
  # Process dataframe contents, subset variables, transformations, etc.
  # Check if city's dataframe and return NULL if it is not suitable
  # If not all cities are suitable for this analysis, check
  # NULL

  procFunc(dataframe) # Function to process each city's dataframe

  NULL
}
```

- **dbName**: a character string containing the name of the database to be created. If not specified, the name of the function in `procFunc` (i.e., via `deparse(substitute(procFunc))`) is used. 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`.

- **compress**: a logical flag indicating whether or not the new database should be stored in a compressed file. The default is NULL, in which case all 108 cities will be included.

- **verbose**: a logical flag indicating whether or not to print diagnostic information. The default is TRUE.

The `preprocessEx` function is used to check if all cities are suitable for the analysis. If not, the function returns NULL. Sections 3 and 4 give further examples of the kinds of functions that may be useful.
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:

```r
> ## 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 the same preprocessing function as the original buildDB call. The NMMAPSdbInfo object is used to recreate the database from the NMMAPSdbInfo object.

Once a database is constructed using buildDB, it is registered in the current environment with a call to registerDB. When registerDB is called with no arguments it sets the full NMMAPS database as the currently registered database. registerDB is called with no arguments if it is registered in the call to registerDB when the database is constructed. The default is TRUE.

http://biostats.bepress.com/jhubiostat/paper44
The argument dbName can be used to register other databases. Currently using full NMMAPS database.

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.

Load New York City dataframe

```r
loadCity("ny")
```

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.

```r
ny[1:5, 1:10]
```

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

```r
dframe <- readCity("ny")
```

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.

```r
attachCity("ny")
```

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.

```r
dframe$cvd[1:10] # 10 days of CVD mortality counts from New York City
dframe$death[1:10] # 10 days of total non-accidental mortality counts from New York City
```
In this Section we demonstrate how to use the NMMAPS 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:

$$Y_t \sim \text{Poisson}(\lambda_t)$$

$$\log(\lambda_t) = \text{DOW}_t + \text{AgeCat} + s(\text{temp}_t; df=6) + s(\text{temp}_t; 1-3; df=6) + s(\text{dewpt}_t; df=3) + s(\text{dewpt}_t; 1-3; df=3) + s(\text{t}; df=7 \times \# \text{years}) + s(\text{t}; 0:15 \times \# \text{years}) + \text{AgeCat} + \beta$$

where $Y_t$ is the number of non-accidental deaths on day $t$ for a particular age category, $\beta$ is an intercept for the age category, and $\beta$.

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

$$\text{log}(\lambda_t) = \text{DOW}_t + \text{AgeCat} + s(\text{temp}_t; df=6) + s(\text{temp}_t; 1-3; df=6) + s(\text{dewpt}_t; df=3) + s(\text{dewpt}_t; 1-3; df=3) + s(\text{t}; df=7 \times \# \text{years}) + s(\text{t}; 0:15 \times \# \text{years}) + \text{AgeCat} + \beta$$

where $Y_t$ is the number of non-accidental deaths on day $t$ for a particular age category, $\beta$ is an intercept for the age category, and $\beta$.

For doing an analysis of PM$_{10}$ and mortality basicNMMAPS <- function (dataframe) {
  if(all(is.na(dataframe[, "pm10tmean"]))){
    return(NULL)
  } else {
    ## Set extreme mortality values to NA
    is.na(dataframe[, "death"] <- as.logical(dataframe[, "markdeath"])
    
    ## Set extreme PM$_{10}$ data, if any
    if(any(dataframe[, "pm10tmean"] > 10000))
      dataframe[, "pm10tmean"] <- NA
  }
}

For more details on the smooth functions and their implementation, see Kelsall et al. (1997) and Samet et al. (1998).
The function coerces `is.na(dataframe[, "cvd"] <- as.logical(dataframe[, "markcvd"])

The day-of-week and age category variables to factors. Finally, a subset of the pollution indicators included in the new database can be retrieved with the `listDBCities` function.

A listing of the abbreviated names of the cities always helps the names of the cities in the current database. Notice that there are 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. The new database can be built with a call to `buildDB`.

The process takes approximately 16 minutes on a PC equipped with an AMD Athlon XP 2100+ processor running the Microsoft Windows XP operating system.
The \textit{simple.R} file 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 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.

The function returns a fitted NMMAPS model for a single city. It is essentially a wrapper which sets up appropriate formulas for the \texttt{glm} call and uses an appropriate formulation for the \texttt{gam} call. The function returns an object that can be used to fit a Poisson regression model for the city. The function \texttt{fitSingleCity} in \textit{simple.R} can be used to fit a Poisson regression model for the city.

The model of daily non-accidental mortality and cause-specific PM$_{10}$ data can be downloaded from http://biostats.bepress.com/jhubiostat/paper44. Only 102 cities listed — 6 of the 108 cities in the full database do not contain any PM$_{10}$ data.
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)

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 PM10 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 PM10 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 μg/m³ increase in PM10. The estimate for the lag 1 PM10 coefficient is
Results from single city studies tend to be very noisy and sensitive to model specification. With more stable estimates of the pollution effect, estimates from multiple cities 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 Poisson regression models to data from each city. The function cityApply can be used to apply a function to all of the data from each city. The separate Poisson regression models to the separate city data can be used to produce a multi-city analysis.
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` 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 extractor functions via the `extractors` argument and these functions will be applied to the object returned from the call to `glm`. Over, one can pass a list of functions via the `extractors` which, by default, is `NULL`. However, the extractor function in `fitSingleCity` takes, approximately 23 minutes. The object returned is a list of length 11.

\[
\begin{align*}
\text{coef} & \rightarrow \text{summary}(x)[,\text{"coefficient"}, \text{"empmeans"}, 2] \\
\text{std} & \rightarrow \text{summary}(x)[,\text{"coefficient"}, \text{"empmeans"}, 1]
\end{align*}
\]

The function `fitSingleCity` takes an argument `extractors` which, by default, is `NULL`. However, one can pass a list of extractor functions via the `extractors` argument and these functions will be applied to the object returned from the call to `glm`. Over, one can pass a list of functions via the `extractors` which, by default, is `NULL`. However, the extractor function in `fitSingleCity` takes, approximately 23 minutes. The object returned is a list of length 11.
In this section we provide an example use 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 functions have been modified for use in R. We have distributed the modified functions 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). We have distributed the modified functions to investigate sensitivity of the PM_{10}-mortality relationship to control for weather and season.

The TLNise (Two Level Normal independent sampling estimation) software used here for the hierarchical model can be downloaded separately from Phillip Everson's website at http://www.swarthmore.edu/NatSci/peverso1/TLNise/tlnise.htm.
of generalized linear models for mortality on PM0\textsubscript{10} and distributed lags of temperature. The basic form for these models is
\begin{align*}
Y_t & \sim \text{Poisson}(\mu_t) \\
\log \mu_t &= DOW_t + DOM_t + \text{AgeCat} + r(dewpt_t) + s(dewpt_{t-1}, df = 4) + s(dewpt_{t-2}, df = 1) + s(temp_t, df = 4, \# \text{years}) + s(temp_{t-1}, df = 1) + s(temp_{t-2}, df = 7) + s(temp_{t-3}, df = 14) + \bar{PM}_t
\end{align*}
where temp\textsubscript{t}, temp\textsubscript{t-1}, temp\textsubscript{t-2}, and temp\textsubscript{t-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 \(r(dewpt_t)\) and \(s(dewpt_{t-1}, df = 4)\) denote the residuals from regressing current day dew point and the average of previous two days dew points on all temperature attributes. 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 NMMAPS\textsubscript{Data} package, begin

```r
# load NMMAPS\textsubscript{Data} and splines packages
library(NMMAPSdata)

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 the NMMAPS\textsubscript{Data} and splines packages. The splines package is required for fitting the distributed lag models.

```
## examine data for New York City

```r
> data <- readCity("ny")
> data[1:2,]
```

```
   var.lag:
   # lagged by days
   # date mon yr doy
   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

   var.lmean:
   # average of the past days of var
   # date mon yr doy
   1 NA NA NA NA NA NA
   2 NA NA NA NA NA NA

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

We may now re-create the exploratory analysis presented in Welty and Zeger, Distributed lag
relations of the mortality - temperature relationship.
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.); \( \theta_0 \) is the coefficient for \( \text{tmpd} \), \( \theta_1 \) is the coefficient for \( \text{tmpd.lmean2} \), and so on. For additional details on month numbering or the \( \theta_s \) see the comments in tdlm4MO.

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

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

We now switch to Los Angeles and fit several models, using the function tdlmSV. The function tdlmSV takes as arguments the city-specific dataframe `data <- readCity("la")`, degrees of freedom for the smooth function of time, degrees of freedom for the time by age category interaction, the lag of \( \text{PM}_{10} \) to use as exposure, and whether or not to include interactions of the distributed lags of temperature.

```r
> results <- tdlmSV(data = data, degf = 28, degfage = 14,
+ pol.lag = 1, inter = FALSE)
```

In the full model, the estimated coefficient and its standard error as well as the coefficient of the distributed lags of temperature are

- 0.0006091162, SE = 0.0001949552
- 0.003991299, SE = 0.0007882620
- 0.002467549, SE = 0.0039741800
- 0.003422143, SE = 0.0054526861
- 0.001162100, SE = 0.0113166819

Estimates of the \( \theta_s \) values can be found in the output of the function tdlmSV. The function tdlmSV takes as arguments the city-specific dataframe of the seasonally varying distributed lag models for the seasonality varying temperature that was used to fit the models for temperature section, for New York City. The code to the right specified all degrees of freedom, not as-de-

```r
> names(results)
[1] "pol.ef"  "pol.se"  "model"
```

degrees of freedom per year. The code to the right specified all degrees of freedom, not as-de-

```r
> results$pol.ef
value
- 0.0006091162
0.003991299
0.002467549
0.003422143
0.001162100
```

Note that degrees of freedom are

```r
> results$pol.se
value
- 0.0001949552
0.0007882620
0.0039741800
0.0054526861
0.0113166819
```

Note that degrees of freedom are

```r
> results$pol.se
value
- 0.0001949552
0.0007882620
0.0039741800
0.0054526861
0.0113166819
```
fits the model referred to as SV \( \text{I}^2 \) in Welty and Zeger (2004) for Los Angeles. We may analogously fit the model \( \text{SV} \text{I}^8 \) for Los Angeles. The results for \( \text{SV} \text{I}^2 \) above and \( \text{SV} \text{I}^8 \) match those presented in Figure 4 of Welty and Zeger (2004).

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

We now fit two models with non-linear distributed lags of temperature, \( \text{NL} \text{I}^4 \) (1; 4) and \( \text{NL} \text{I}^4 \) (2; 4), using \( \text{tdlmNL} \). Similar to \( \text{tdlmSV} \) (but with slightly different names), \( \text{tdlmNL} \) 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\( \text{I}^0 \) exposure. Again, the degree of freedom per year, the arguments \( \text{df.time} \) and \( \text{df.timeage} \) designate the number of distributed lags and local degrees of freedom, rather than by degrees of freedom per year. The arguments \( \text{df.dlm} \) and \( \text{nlags} \) specify the number of distributed lags to include in the model and their (natural spline) degrees of freedom. The comments in \( \text{tdlmNL} \) provide greater detail.

We now fit two models with non-linear distributed lags of temperature, \( \text{NL} \text{I}^4 \) (1; 4) and \( \text{NL} \text{I}^4 \) (2; 4), using \( \text{tdlmNL} \). Similar to \( \text{tdlmSV} \) (but with slightly different names), \( \text{tdlmNL} \) 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\( \text{I}^0 \) exposure. Again, the degree of freedom per year, the arguments \( \text{df.time} \) and \( \text{df.timeage} \) designate the number of distributed lags and local degrees of freedom, rather than by degrees of freedom per year. The arguments \( \text{df.dlm} \) and \( \text{nlags} \) specify the number of distributed lags to include in the model and their (natural spline) degrees of freedom. The comments in \( \text{tdlmNL} \) provide greater detail.

Lastly, in preparation for obtaining national results from those used in Welty and Zeger (2004), we create a table with city name and the corresponding PM\( \text{I}^0 \) coefficient estimates, for as many cities in NMMAPS as possible. The function \( \text{tempDLMCities} \) returns the list of cities used in Welty and Zeger (2004).
two columns of the table (other appropriate co-
cord error in the table results table. The
the city name and the coefficient and the stan-
efficient from $N(3,2)$ for each city and show-

The loop to the right computes the pollution co-

Building the databases for cities takes approx-

approximately 13 minutes on a Pentium M 1.7 GHz

> 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
+ 
+ each ---> C:/mydoc/nmmaps_pub/R//tempDLM/mich.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 co-

# 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)
+ results.table[i,2] <- results$coefficients$df.dlm$coefficients
+ results.table[i,3] <- results$coefficients$df.dlm$coefficients.sd.error
+ }

> results.table
         No.1        No.2
akr  -0.05732054  0.00571004
albu  0.00348482  0.00348482
anch  0.00000000  0.00000000

city information to R-normal hierarchical model can
be found in Everson and Morris (2000).
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 potential confounders are the same as in model (1) however now the effect of PM$_{10}$ on mortality is a periodic function of time. The potential confounders are the same as in seasonal variation is plausible because of the changing sources of PM throughout the year. The potential confounders are the same as in seasonal variation of the effect of PM$_{10}$ on mortality.

The seasonal interaction model described in this section is expressed as

$$
\log t = \bar{\beta}_0 + \bar{\beta}_1 \sin(2\pi t/365) + \bar{\beta}_2 \cos(2\pi t/365) \pm PM_t + \text{confounders}
$$

where $\bar{\beta}_0, \bar{\beta}_1, \bar{\beta}_2$ 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 in this section.

In this section we reproduce some of the analyses presented in Peng et al. (2004). The models presented here incorporate seasonal variation.
In this portion of the function we setup the seasonal interaction with PM$_{10}$. There are two choices: smooth, which will use the default temperature model, or all (2004) for sensitivity analyses. For now we will use the default temperature model.

The function `setupTemp` can be used to setup built-in preprocessing function `match.arg` which can be used to prepare the NMMAPS database built-in preprocessing function `match.arg` which can be used to prepare the NMMAPS database.
uses the sine/cosine basis and stepfun, which else {
switch(season, how many sine/cosine terms to include in the
model (defaults to one of each). The function
nfreq <- df.Season
periodicBasis is reproduced in the Appendix
paste("periodicBasis(SeasonTime," nfreq,
and is needed to construct the sine/cosine basis
which interacts with the PM
10 variable.
}
stepfun = past("Season", pollutant)
(formula = form.str)
" + " + sep = "pollutant, f = "time.f, temp.f, weather.f,
form.str = paste(formula, pollutant, Sep, f = "time.f, temp.f, weather.f,

smooth = smooth(Season, f = "time.f)
stepfun = past("Season", pollutant, sep = "pollutant")
(model = model(formula = as.formula(form.str),
control = control(eps = 1e-10, maxit = 1000),
fit = E1m(formula = formula, data = data,
fit = glm(formula = formula, family = quasipoisson, data = data,
control = glm.control(epsilon = 1e-10, maxit = 1000),
na.action = na.omit)
fit = glm(formula = formula, family = quasipoisson, data = data,
control = glm.control(epsilon = 1e-10, maxit = 1000),
na.action = na.omit)

(b) <- coef(fit) # Get estimated coefficients
b <- periodicBasis(1:365, nfreq = 1, period = 365, intercept = TRUE)

We can use the periodicBasis function to do
seasonal analysis of Detroit.

When doing multi-city analyses,
which interacts with the PM
10 variable.
we can be passed if we do not want to return the
fit function is called to fit the model. Which

type extractor function
and is needed to construct the sine/cosine basis
and the seasonal model is
to do the PM
10 analysis is total non-
accidental mortality, and the seasonal model is
to do the PM
10 analysis is total non-
accidental mortality, and the seasonal model is
when doing multi-city analyses.
which interacts with the PM
10 variable.
we can be passed if we do not want to return the
fit function is called to fit the model. Which

type extractor function
and is needed to construct the sine/cosine basis
and the seasonal model is
to do the PM
10 analysis is total non-
accidental mortality, and the seasonal model is
when doing multi-city analyses.
which interacts with the PM
10 variable.
we can be passed if we do not want to return the
fit function is called to fit the model. Which

type extractor function
and is needed to construct the sine/cosine basis
and the seasonal model is
to do the PM
10 analysis is total non-
accidental mortality, and the seasonal model is
when doing multi-city analyses.
which interacts with the PM
10 variable.
we can be passed if we do not want to return the
fit function is called to fit the model. Which

type extractor function
and is needed to construct the sine/cosine basis
and the seasonal model is
to do the PM
10 analysis is total non-
accidental mortality, and the seasonal model is
when doing multi-city analyses.
which interacts with the PM
10 variable.
we can be passed if we do not want to return the
fit function is called to fit the model. Which

type extractor function
and is needed to construct the sine/cosine basis
and the seasonal model is
to do the PM
10 analysis is total non-
accidental mortality, and the seasonal model is
when doing multi-city analyses.
which interacts with the PM
10 variable.
we can be passed if we do not want to return the
fit function is called to fit the model. Which

type extractor function
and is needed to construct the sine/cosine basis
and the seasonal model is
to do the PM
10 analysis is total non-
accidental mortality, and the seasonal model is
when doing multi-city analyses.
which interacts with the PM
10 variable.
we can be passed if we do not want to return the
fit function is called to fit the model. Which

type extractor function
and is needed to construct the sine/cosine basis
and the seasonal model is
to do the PM
10 analysis is total non-
accidental mortality, and the seasonal model is
when doing multi-city analyses.
which interacts with the PM
10 variable.
we can be passed if we do not want to return the
fit function is called to fit the model. Which

type extractor function
and is needed to construct the sine/cosine basis
and the seasonal model is
to do the PM
10 analysis is total non-
accidental mortality, and the seasonal model is
when doing multi-city analyses.
which interacts with the PM
10 variable.
we can be passed if we do not want to return the
fit function is called to fit the model. Which

type extractor function
and is needed to construct the sine/cosine basis
and the seasonal model is
to do the PM
10 analysis is total non-
accidental mortality, and the seasonal model is
when doing multi-city analyses.
which interacts with the PM
10 variable.
we can be passed if we do not want to return the
fit function is called to fit the model. Which

type extractor function
and is needed to construct the sine/cosine basis
and the seasonal model is
to do the PM
10 analysis is total non-
accidental mortality, and the seasonal model is
when doing multi-city analyses.
which interacts with the PM
10 variable.
we can be passed if we do not want to return the
fit function is called to fit the model. Which

type extractor function
and is needed to construct the sine/cosine basis
and the seasonal model is
to do the PM
10 analysis is total non-
accidental mortality, and the seasonal model is
when doing multi-city analyses.
which interacts with the PM
10 variable.
we can be passed if we do not want to return the
fit function is called to fit the model. Which

type extractor function
and is needed to construct the sine/cosine basis
and the seasonal model is
to do the PM
10 analysis is total non-
accidental mortality, and the seasonal model is
when doing multi-city analyses.
which interacts with the PM
10 variable.
we can be passed if we do not want to return the
fit function is called to fit the model. Which

type extractor function
and is needed to construct the sine/cosine basis
and the seasonal model is
to do the PM
10 analysis is total non-
accidental mortality, and the seasonal model is
when doing multi-city analyses.
which interacts with the PM
10 variable.
we can be passed if we do not want to return the
fit function is called to fit the model. Which

type extractor function
and is needed to construct the sine/cosine basis
and the seasonal model is
to do the PM
10 analysis is total non-
accidental mortality, and the seasonal model is
when doing multi-city analyses.
which interacts with the PM
10 variable.
we can be passed if we do not want to return the
fit function is called to fit the model. Which

type extractor function
and is needed to construct the sine/cosine basis
and the seasonal model is
to do the PM
10 analysis is total non-
accidental mortality, and the seasonal model is
when doing multi-city analyses.
which interacts with the PM
10 variable.
we can be passed if we do not want to return the
fit function is called to fit the model. Which

type extractor function
and is needed to construct the sine/cosine basis
and the seasonal model is
to do the PM
10 analysis is total non-
accidental mortality, and the seasonal model is
when doing multi-city analyses.
which interacts with the PM
10 variable.
we can be passed if we do not want to return the
fit function is called to fit the model. Which

type extractor function
and is needed to construct the sine/cosine basis
and the seasonal model is
to do the PM
10 analysis is total non-
accidental mortality, and the seasonal model is
when doing multi-city analyses.
which interacts with the PM
10 variable.
we can be passed if we do not want to return the
fit function is called to fit the model. Which

type extractor function
and is needed to construct the sine/cosine basis
and the seasonal model is
to do the PM
10 analysis is total non-
accidental mortality, and the seasonal model is
when doing multi-city analyses.
which interacts with the PM
10 variable.
we can be passed if we do not want to return the
fit function is called to fit the model. Which

type extractor function
and is needed to construct the sine/cosine basis
and the seasonal model is
to do the PM
10 analysis is total non-
accidental mortality, and the seasonal model is
when doing multi-city analyses.
which interacts with the PM
10 variable.
we can be passed if we do not want to return the
fit function is called to fit the model. Which

http://biostats.bepress.com/jhubiostat/paper44
seas.curve <- B %*% b[grep("pm10", names(b))]

## Estimated curve

fitCitySeason returns the full glm object by default, from which we extract the coefficients.

V <- vcov(fit)

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.

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 RO1ES012054, 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


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 = "+")
  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, pollutant), collapse = "+")
  as.formula(form.str)
}
```
B Code for Section 4.1: Analysis of PM10 and Mortality Using Distributed Lag Models for Temperature

```r
# Function to calculate lagged temperature coefficients

tdlm4MO <- function(data) {
  # Create variable month, consecutive numbering of months in dataset
  month <- as.numeric(substring(data$date, 5, 6))
  month <- month + (as.numeric(substring(data$date, 1, 4)) - 1987) * 12

  # Create matrix of distributed lag temperature coefficients, 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)) {
    # Fit generalized linear model with quasipoisson distribution
    mod <- glm(death ~ as.factor(agecat) + as.factor(dow) + dom +
               tmpd + tmpd.lmean2 + tmpd.lmean7 + tmpd.lmean14,
               data = data, family = quasipoisson)

    # Extract coefficients for months 3 to 6
    coeffs[i, 3] <- coefficients(mod)[4]
    coeffs[i, 4] <- coefficients(mod)[5]
    coeffs[i, 5] <- coefficients(mod)[6]
    # Extract coefficients for months 7 to 10
    coeffs[i, 1] <- coefficients(mod)[7]
    coeffs[i, 2] <- coefficients(mod)[8]

    # Adjust for reference month
    month[i] <- month[i] - 3
  }

  return(coeffs)
}
```

---

**Temporale**

Temperature

**Code for Section 4.1: Analysis of PM10 and Mortality Using Distributed Lag Models for Temperature**

```r
# Function to calculate lagged temperature coefficients

tdlm4MO <- function(data) {
  # Create variable month, consecutive numbering of months in dataset
  month <- as.numeric(substring(data$date, 5, 6))
  month <- month + (as.numeric(substring(data$date, 1, 4)) - 1987) * 12

  # Create matrix of distributed lag temperature coefficients, 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)) {
    # Fit generalized linear model with quasipoisson distribution
    mod <- glm(death ~ as.factor(agecat) + as.factor(dow) + dom +
               tmpd + tmpd.lmean2 + tmpd.lmean7 + tmpd.lmean14,
               data = data, family = quasipoisson)

    # Extract coefficients for months 3 to 6
    coeffs[i, 3] <- coefficients(mod)[4]
    coeffs[i, 4] <- coefficients(mod)[5]
    coeffs[i, 5] <- coefficients(mod)[6]
    # Extract coefficients for months 7 to 10
    coeffs[i, 1] <- coefficients(mod)[7]
    coeffs[i, 2] <- coefficients(mod)[8]

    # Adjust for reference month
    month[i] <- month[i] - 3
  }

  return(coeffs)
}
```
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$coefficients[substring(names(mod$coefficients),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 havetemporal:seasonal variation
  ## degf is total df for smooth of time
  ## degfage is total df for interaction of agecat and time
  ## 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
  
  ######################################################################

  bas0 <- ns(data$days, df = degf)
  bas1 <- cbind(sin(2 * pi * data$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 <- function(data, df, degree, lag)
          {
              basis <- function(data, df, degree, lag)
              {

              } # basis function

  
  bas0 <- ns(data$days, df = degf)
  bas1 <- cbind(sin(2 * pi * data$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))
    
  return(coefficients)
  
  

  return(coefficients)
bas2 <- ns(data$days, df = 3)
## basis for interaction btw season and time in dist lag temperature coeffs
bas3 <- matrix(nrow = nrow(bas1), ncol = 12)
for (i in 1:ncol(bas1)) {
  for (j in 1:ncol(bas2)) {
    bas3[(i-1)*ncol(bas2) + j] = bas1[,i]*bas2[,j]
  }
}
## temperature covaritates for model
t.base <- c("tmpd", "tmpd.lmean2", "tmpd.lmean7", "tmpd.lmean14")
t_vars <- c(t.base, paste(t.base, ":bas1", sep = ""), paste(t.base, ":bas2", sep = ""), paste(t.base, ":bas3", sep = ""))
## interactions of dist lags of temperature, if appropriate
if (inter == TRUE) {
t.inter <- c("tmpd:tmpd.lmean2", "tmpd:tmpd.lmean7", "tmpd.lmean2:tmpd.lmean7")
t_vars <- c(t_vars, t.inter, paste(t_inter, ":bas1", sep = ""), paste(t_inter, ":bas2", sep = ""), paste(t_inter, ":bas3", sep = ""))
}
## compute residualized dew point covariates to include in model
mod.form <- formula(paste("dptp ~", paste(t_vars, collapse = " + ")))
mod <- lm(mod.form, data = data, na.action = na.omit)
mod$r.dptp <- vector(length = nrow(data))
mod$r.dptp[as.numeric(names(residuals(mod)))] <- as.numeric(residuals(mod))
mod$r.dptp[as.numeric(unique(mod$na.action))] <- rep(NA, length(unique(mod$na.action)))
mod.form <- formula(paste("dptp.lmean2 ~", paste(t_vars, collapse = " + ")))
## temperature covariates for model

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
## to include(i.e. nlags = 2 means use tmpd and tmpd.lmean2)
## df.dlm determines the df of smooth function of the distributed
## lags of temperature to use in model
## 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 = ", df.tmpd", sep = "\n\n")

## 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, \(degfage\)), 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])
C Code for Section 4.2: Seasonal Analyses

```c
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)
}
```
```r
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, "\n                    ")"},
                    default = default.temp.f,
                    tempModel = switch(tempModel)

  list(adj.data = dataframe, temp.f = temp.f,
       addedVars = setdiff(names(dataframe), orig.namelist))
```