Distributed lag non-linear models in R: the package dlnm

Antonio Gasparrini and Ben Armstrong
London School of Hygiene and Tropical Medicine, UK

dlnm version 0.4.0 , 2009-08-03

Contents

1 Preamble **2**

2 Introduction **2**
2.1 Installing the package dlnm 2
2.2 Data 3

3 Distributed lag non-linear models (DLNM's) **3**
3.1 The issue 3
3.2 The concept of basis 4
3.3 Delayed effect: DLM's 4
3.4 The extension to DLNM's 5

4 The functions in the package dlnm **6**
4.1 Internal functions: `mkbasis()` and `mklagbasis()` 6
4.2 The function `crossbasis()` 7
4.3 The function `crosspred()` 9
4.4 The function `crossplot()` 10

5 Some examples **11**
5.1 Examples for `mkbasis()` and `mklagbasis()` 12
5.2 Example 1: a simple DLM 15
5.3 Example 2: a threshold parameterization 17
5.4 Example 3: a complex DLNM 21

6 Conclusions **24**

7 Acknowledgements **25**

[1]This work was supported by the Medical Research Council (UK) through the Research Grant RES-G0707030.

1 Preamble

This document is included as a vignette (a LaTeX document created using the R function `Sweave()`) of the package dlnm. It is automatically dowloaded together with the package and can be accessed through R typing *`vignette("dlnmOverview", package="dlnm")`*, or otherwise directly in the path `library/dlnm/doc` within the R folder. A file named `dlnmOverview.R` containing the R pieces of code (*chunks*) illustrated here can be found in the same folder and run to check the results. Type *`citation("dlnm")`* in R to cite this package. A list of changes included in the current and previous versions can be found typing *`file.show(system.file("dlnmChangeLog", package = "dlnm"))`*.

Please send comments or suggestions and report bugs to `antonio.gasparrini@lshtm.ac.uk`.

2 Introduction

The R package dlnm provides some facilities to run *distributed lag non-linear models* (DLNM's), a modelling framework to describe simultaneously non-linear and delayed effects between predictors and an outcome in time-series data.

The aim of this document is to provide an extended overview of the capabilities of the package, including an detailed summary of the functions included here, with some examples of application to real data. Although these examples refer to the health effects of air pollution and temperature, the purpose of package is fairly general, and it can be used to specify DLNM's in order to investigate the relationship between series of any predictors and outcomes.

The remainder of Section 2 provides some information on the installation of the package dlnm and on the data used throughout this document. The family DLNM's is introduced in Section 3, while the R functions used to specify them are described in Section 4. Three different examples are illustrated in Section 5: users mainly interested in the application of DLNM's can skip the previous sections and start with these examples.

We state beforehand that the goal of the examples included in this document is to describe the functionalities of the package, and that the results should not be used to infer some conclusions on the causal associations of the relationships considered there.

2.1 Installing the package dlnm

The dlnm package is installed in the standard way for CRAN packages, for example using the `install.packages()` function or directly through the menu in R (from version 2.9.0 onwards), clicking on *Packages* and then on *Install package(s)....* The package can be alternatively installed using the .zip file containing the binaries, via *Packages* and then *Install package(s) from local zip files....*

The functionalities of dlnm depend on other packages whose commands are used to specify the dlnm functions or whose data are used throughout the examples. This hierarchy is ruled by the fields *Depends* and *Imports* of the file `description` included in the package. Some functions are imported from the packages splines and tsModel, while the package NMMAPSlite provides the data to run the examples (see Section 2.2). While splines is present in the basic installation of R, the

packages tsModel and NMMAPSlite are automatically downloaded if dlnm is installed through R using the CRAN, but must be independently installed if a .zip file is used.

The package `dlnm` is assumed to be present in the R library and loaded in this session. If not, type:

```
> library(dlnm)
```

2.2 Data

The dlnm package does not contain any data, and the examples illustrated in Section 5 are carried out on the data available through the package NMMAPSlite (type `?NMMAPSlite` in R for further information). The database contains dataframes with air pollution, weather, and mortality data for 108 United States cities in the period 1987-2000. Each city dataframe contains daily time series of mortality counts (for various causes of death), pollution levels, and weather (e.g. temperature and humidity) [12]. The data were assembled from publicly available data sources as part of the National Morbidity, Mortality, and Air Pollution Study (NMMAPS) sponsored by the Health Effects Institute [14, 13].

The examples in Section 5 are carried out using the NMMAPS data for the city of Chicago in the period 1987-2000, with the daily overall mortality series as the outcome and temperature, PM_{10} and ozone (O_3) as predictors. The database is loaded in the R session into the object `data`, and then manipulated by:

```
> initDB()
> data <- readCity("chic", collapseAge=TRUE)
> data$temp <- (data$tmpd-32)*5/9
> data$pm10 <- with(data, pm10tmean+pm10mtrend)
> data$o3 <- with(data, o3tmean+o3mtrend)
```

The variable `temp` represents the conversion in Celsius degree (°C) of the series of mean temperature in Fahrenheit (°F), while the original PM_{10} and O_3 series are approximately recovered summing the de-trended and trend series (see `http://www.ihapss.jhsph.edu` for further information). The argument `collapseAge=TRUE` collapses the age-stratified records in a a single ordered and complete series, as required by the function `crossbasis()` (see Section 4.2). The dataset does not require further manipulation.

3 Distributed lag non-linear models (DLNM's)

The aim of this Section is to provide a summary on the framework of DLNM's. A detailed overview of this family of models have already been published elsewhere [3, 8].

3.1 The issue

The main purpose of a statistical regression model is to define and then estimate the effect of a regressor on an outcome. A first problem arises when the relationship between them is non-linear:

several possible solutions have been proposed, and many of them involve a manipulation of the original variable in order to specify one or more new predictors which must explain, globally, the shape of the dependency: a simple example is the inclusion of a quadratic term in the model.

A further complexity occurs when the effect of a specific occurrence of the predictor is not limited to the period when it is observed, but is *delayed* in time. In this case, more complex models are required to specify the dependency, taking into account the inclusion of the additional time dimension.

3.2 The concept of basis

Several different methods have been adopted to specify non-linear effects in a regression models. A simple solution is to generate strata variables, applying specific cut-off points along the range of the predictor in order to define specific intervals, and then specifying new variables through a dummy parameterization.

Other types of manipulations of the original variable are applied when there are specific assumptions on the shape of the relationship, for example when the effect is likely to exist and be linear only above or below a specific threshold (*hockey-stick* model). An extension of this model assumes two distinct linear effects below a first threshold and above a second threshold, with a null effect in between them.

An alternative to the strata or threshold approaches is to include in the model some terms allowing a true non-linear relationship, describing a smoothed curve between the predictor and the outcome. The traditional methods include a quadratic term or higher degree polynomials. Recently, spline functions have been favoured, especially through a natural cubic parameterization.

A generalization may be provided assuming that all the approaches above imply the choice of a *basis*, defined as a *space of functions* used to define the relationship [19]. The choice of the basis defines the related *basis functions*, completely known transformations of the original predictor generating a new set of transformed variables, defined *basis variables*. Independently from the basis chosen, the final result will be a matrix of transformed variables which can be included in the design matrix of a model in order to estimate the related parameters. The choice of different bases leads to the specification of different matrices, but the mechanism is common.

3.3 Delayed effect: DLM's

In the specific context of time series analysis, given the ordered series of the predictor values, a delayed (or lagged) effect is present when the outcome in a specific day is influenced by the level of the predictor in the days before, up to a maximum lag. Therefore, the presence of delayed effects requires to take into account the *time dimension* of the relationship, specifying the additional *virtual* dimension of the lags.

A very simple model to deal with delayed effects considers the moving average of the predictor up to a certain lag, specifying a transformed predictor which is the average of the values in that specific lag period. Although simple, this model is limited if the purpose is to assess the temporal structure of the effects.

These limitations have been addressed using a more elegant approach based on distributed lag mod-

els (DLM's). The main advantage of this method is the possibility to depict a detailed description of the time-course of the relationship. Originally developed in econometrics [1], this method has recently been used to quantify the health effect in studies on environmental factors [21, 16, 18, 5].

In the basic formulation, a DLM is fitted including a parameter for each lagged predictor occurrence. An estimate of the overall effect is given by the sum of the single lag effects upon the whole lag period considered [15, 9].

This *unconstrained* version of DLM does not require any assumption on the shape of the effect along lags, and consequently on the relationship between parameters. In order to define a more parsimonious model, it is possible to specify some assumptions on the shape of the distributed effect, applying some constraint. The simplest solution is to group the lags in different strata [18, 10], while a more complex option is to force the curve along lags to follow a specific smooth function, for example polynomials [20, 17, 4] or splines [21].

Following the general approach used in Section 3.2, it may be shown [8] that all the different DLM's above can be described by the same equation, where different models are specified through different basis functions to be applied to the virtual vector of lags, building a new basis matrix. Again, the choice of different bases generates different matrices, but the mechanism is general.

3.4 The extension to DLNM's

A general approach to specify non-linear but un-lagged effects has been introduced in Section 3.2, while the methods to define distributed lag functions for simple linear effects have been presented in Section 3.3. An obvious extensions is to combine these approaches to define distributed lag non-linear models (DLNM's), a family of models which can deal at the same time with non-linear and delayed effects [3, 8].

The different issues of non-linearity and delayed effects share a common feature: in both cases the solution is to choose a basis to describe the shape of the relationship in the relative dimension. This step leads to the concept of *cross-basis*: following the idea of basis in 3.2, a cross-basis can be imagined as a bi-dimensional space of functions describing on the same time the shape of the relationship and the distributed lag effects. The algebraic notation to define the cross-basis and then the DLNM can be quite complex, involving tensor products of 3-dimensional arrays, and has been presented elsewhere [8]. Nonetheless, the basic concept is straightforward: choosing a cross-basis amounts to choosing two independent set of basis functions, which will be combined to generate the specific cross-basis functions. The DLM's described in 3.3 can be considered as special cases of DLNM's with a simple linear function in the dimension of the predictor.

The result of a DLNM can be interpreted building a grid of predictions for each lag and for suitable values of the predictor, using three dimensional plots to provide an overall picture of the effects varying along the two dimensions. In addition, it is possible to compute the effects for single predictor levels or lags, simply cutting a "slice" of the grid along specific values of predictor or lags, respectively. Finally, an estimate of the overall effect can be computed by summing all the contributions at different lags. The effects are usually reported versus a reference value of the predictor, centering the basis functions for this space to their corresponding transformed values [6].

The choice of the two set of basis functions for each space is perfectly independent, and should be based on a-priori assumptions or on a compromise between complexity and generalizability. Linear, threshold, strata, polynomial or splines functions can be used to define the relationship along the

space of predictor, while unconstrained, strata, polynomial or splines functions can be applied to specify the shape along lags.

The package dlnm provides a function named `crossbasis()` to specify the cross-basis matrix, given the choices on the two bases among the options listed above, and two other functions `crosspred()` and `crossplot()` to predict and plot the estimated effects, respectively.

4 The functions in the package dlnm

The functions included in the package can be used to complete all the steps required to specify and interpret a DLNM.

First, the internal functions `mkbasis()` and `mklagbasis()` are called in order to build the basis matrices for the dimension of the predictor and lags, respectively. In concrete terms, they apply a transformation to the vector of predictor and to the vector of lags, and stored the transformed variables in two matrix objects. Details on the basis specification are given in Section 3. These two internal commands are called directly by `crossbasis()`, and they are not meant to be run by the users.

The main function in the package dlnm is `crossbasis()`. It calls the internal functions `mkbasis()` and `mklagbasis()` and combines the two basis matrices in order to create the cross-basis matrix which specifies the dependency simultaneously in the two dimensions. The function `summary.crossbasis()` provides a summary of the choices made for the two bases and the final cross-basis. The cross-basis matrix should be included in the model formula of default model commands to estimate the parameters defining the shape of the effects along the space of the predictor and along lags. The package dlnm has been tested with the model function `glm()`; the accuracy of the results obtained with other commands is not guaranteed and should be carefully checked.

The function `crosspred()` generates the predicted effects for a set of values of the original predictor, given the applied cross-basis functions and the parameters estimated by the model. It stores them in matrices with specific effects for each combination of predictor values and lags, and in vectors of overall effects (summed up along lags).

Finally, the function `crossplot()` provides some options to visualize the predicted effects.

4.1 Internal functions: `mkbasis()` and `mklagbasis()`

These two internal functions are called by `crossbasis()` to build the basis matrices for the two dimensions of predictor and lags. Even if they are not expected to be directly run by the users, they are included in the `namespace` of the package and therefore made accessible, with the intention to keep the process more transparent and give the opportunity to change or improve them. Type *`?mkbasis`* or *`?mklagbasis`* in R for additional information.

The result of these two functions are two list objects containing the basis matrices and other values corresponding to the arguments of the functions specified below.

The syntax of the two commands is:

```
mkbasis(var, type="ns", df=1, degree=1, knots=NULL, bound=range(var),
```

```
  int=FALSE, cen=TRUE, cenvalue=mean(var))

mklagbasis(maxlag=0, type="ns", df=1, degree=1, knots=NULL,
  bound=c(0, maxlag), int=TRUE)
```

The first function, `mkbasis()`, applies a set basis functions on the vector of the predictor specified in `var`, i.e. it transforms the vector in a new set of basis variables. The choice of the basis is given by the arguments `type`, `df`, `degree`, `knots` and `bound`. The logical argument `int=TRUE` adds an intercept (where allowed) to the basis, while if `cen=TRUE` centers the basis variables for `var` are centered on `cenvalue`. See Section 4.2 or type *`?crossbasis`* in R for additional information.

The function `mklagbasis()` calls `mkbasis()` to build the basis matrix for the space of lags. Basically, it creates a new vector `0:maxlag` (a vector of integers from 0 to the maximum lag allowed) and then apply the related basis functions specified through the same arguments seen above. The basis variables for this dimension are never centered.

See the examples in Section 5.1 for further information.

4.2 The function `crossbasis()`

The purpose of this function is to generate the cross-basis matrix for a predictor vector, given the the two bases chosen to specify the relationship along the dimensions of the predictor and the lags. The values of the predictor vector are expected to be equally spaced (with that space defining a lag unit) and ordered in time, although `NA` values are allowed. Type *`?crossbasis`* in R for additional information.

The function `crossbasis()` returns a matrix object of class *"crossbasis"* which can be included in a model formula in order to fit a DLNM.

The syntax of the command is:

```
crossbasis(var, vartype="ns", vardf=1, vardegree=1, varknots=NULL,
  varbound=range(var), varint=FALSE, cen=TRUE, cenvalue=mean(var),
  maxlag=0, lagtype="ns", lagdf=1, lagdegree=1, lagknots=NULL,
  lagbound=c(0,maxlag), lagint=TRUE)
```

The argument `var` includes the predictor vector, while the other arguments `type`, `df`, `degree`, `knots`, `bound` and `int`, with specific stub `var-` and `lag-`, defines the bases for the two spaces, respectively, creating the related basis variables. See below for a detailed explanation of these arguments. The values in `cen` and `cenvalue` specify if and where the basis variables for the space of the predictor should be centered on the relative transformation of `cenvalue` [6]: this value is used as the reference point for the effects predicted by `crosspred()`. `maxlag` sets the maximum lag in the related dimension. The two basis matrices are built calling the two internal functions `mkbasis()` and `mklagbasis()` and passing the specific arguments.

Different independent choices regarding the basis for each one of the two dimensions are available through the argument `type`, with specific stub `var-` or `lag-`. The first basis is related to `var`, in order to describe the relationship in the space of the predictor. The second one is applied to a new vector `0:maxlag`, in order to describe the relationship in the space of lags. The possible options for `type` are:

- `"ns"`: natural cubic B-splines (constrained to be linear beyond the boundary knots). This is the default `type` for both the dimensions. Specified by `knots` (internal knots) and `bound` (boundary or external knots), calling the functions `ns()` of the package splines (type *?ns* in R for additional information). If `knots` is provided, the dimension `df` is set to `length(knots)+int+1`. An intercept is included if `int=TRUE`. For the basis in the space of the predictor, the transformed variables can be centered at `cenvalue` if `cen=TRUE` (the default).

- `"bs"`: B-splines characterized by `degree` (degree of the piecewise polynomial). Specified by `knots` (internal knots) and `bound` (boundary or external knots), calling the functions `bs()` of the package splines (type *?bs* in R for additional information). If `knots` is provided, the dimension `df` is set to `length(knots)+degree+int`, with the constraint that `df>=degree+int`. An intercept is included if `int=TRUE`. For the basis in the space of the predictor, the transformed variables can be centered at `cenvalue` if `cen=TRUE` (the default).

- `"strata"`: strata variables (dummy parameterization) determined by internal cut-off values specified in `knots`, which represent the lower boundaries for the right-open intervals. Intervals containing no observation are automatically discarded. If `knots` is provided, the dimension `df` is set to `length(knots)+int`. A dummy variable for the reference stratum (the first one by default) is included if `int=TRUE`, generating a full rank basis. The related basis functions are never centered.

- `"poly"`: polynomial with power specified by `degree`. The dimension `df` is set to to `degree+int`. An intercept, corresponding to a vector of 1's (the power 0 of the polynomial) is included if `int=TRUE`. For the basis in the space of the predictor, the transformed variables can be centered at `cenvalue` if `cen=TRUE` (the default).

- `"integer"`: strata variables (dummy parameterization) for each integer values. This choice is explicitly created to specify an unconstrained function in the space of lags, i.e. a function with a parameter for each lag. `df` is set automatically to the number of integer values minus 1 plus `int`. A dummy variable for the reference stratum (the first one by default) is included if `int=TRUE`, generating a full rank basis. The related basis functions are never centered.

- `"hthr"`, `"lthr"`: high and low threshold parameterization, with a linear relationship above or below the threshold, respectively, and flat otherwise. The threshold is chosen by `knots`: if more than one is provided, a piecewise linear relationship is applied above the first knot or below the last one, respectively, with the slope changing at each further knot. `df` is automatically set to `length(knots)+int`. An intercept (corresponding to a vector of 1's) is included if `int=T`. The related basis function is never centered.

- `"dthr"`: double threshold parameterization, assuming two independent linear relationships above the second and below the first threshold, and a null effect between them. The thresholds are chosen by `knots`. If only one is provided, the threshold is unique (V-model). If more than 2 are provided, the first and the last ones are chosen. `df` is automatically set to `2+int`. An intercept (corresponding to a vector of 1's) is included if `int=T`. The related basis functions are never centered.

- `"lin"`: linear relationship (untransformed apart from optional centering). `df` is automatically set to 1. An intercept (corresponding to a vector of 1's) is included if `int=T`. For the basis in

the space of the predictor, the transformed variables can be centered at `cenvalue` if `cen=TRUE` (the default).

The values in `knots`, if provided, are automatically ordered and made unique, and determine the value of `df`. The `knots` must be set within the range of the variable (`var` or `0:maxlag`). The value of `df` always corresponds to the dimension of the relative basis, and can be interpreted as the number of degrees of freedom spent to define the relationship in that space. If only `df` is provided, `varknots` are placed at equally spaced quantiles (in the space of predictor), and `lagknots` at equally spaced values on the log scale of lags: their number is given reversing the equations above. Some arguments may be automatically changed by the function for combinations not sensible, or set to `NULL` if not required: it is recommended to check the resulting basis specifications with the function `suummary.crossbasis()`.

For continuous functions specified with `vartype` equal to `"ns"`, `"bs"`, `"poly"` or `"lin"`, the reference for the effects predicted by `crosspred()` (see Section 4.3) is set at `cenvalue`. For the other choices, the reference is automatic: for `vartype` equal to `"strata"` and `"integer"`, the reference is the first interval, while for `vartype` equal to `"hthr"`, `"lthr"` and `"dthr"`, the reference is the region of null effect below, above or between the threshold(s), respectively.

It is strongly recommended to avoid the inclusion of an intercept in the basis for the space of the predictor, otherwise the presence of the additional intercept (when included) in the model used to fit the data will cause some of the cross-basis variables to be excluded. Conversely, an intercept should always be included in the basis for the space of lags when `lagtype` is equal to `"ns"`, `"bs"`, `"strata"` or `"poly"`.

The arguments specified for the two bases are included as attributes in the matrix object of class *"crossbasis"* created by the function. The objects must retain these attributes and the class when included in the model formula (see Section 4.3). The choices about the bases for the two dimensions are returned by the function `summary.crossbasis()`. The name of the `crossbasis` object will be used by `crosspred()` in order to extract the estimated parameters. This names must not match the names of other predictors in the model formula. In addition, if more than one variable is transformed by cross-basis functions in the same model, different names for the related `crossbasis` objects must be specified.

4.3 The function `crosspred()`

This function generates the predicted effects for a model containing the cross-basis matrix built by `crossbasis()` (see Section 4.2). The prediction is carried out for a set of values of the original predictor. The range of these values must contain the `varknots` specified in the function `crossbasis()` related to the base for the space of the predictor. Type *`?crosspred`* in R for additional information.

The function returns a list object of class *"crosspred"* with values containing the matrices with predicted effects for each combination of predictor values and lags, and the vectors with overall effects (summed up along lags).

The syntax of the command is:

```
crosspred(crossbasis, model, at=NULL, from=NULL, to=NULL, by=NULL)
```

The object for the argument `crossbasis` must be the same containing the cross-basis matrix included in the formula of the model object `model`. The `crossbasis` object must retain all its attributes and class. The set of values for which the effects must be computed can be specified by `at` or alternatively by `from/to/by`. If specified by `at`, the values are automatically ordered and made unique. By default, `from` and `to` correspond to the range of the original vector of observation stored in the `crossbasis` object (see Section 4.2). If `by` is not provided, 30 equally spaced values are returned.

The results are included in the list objects of class *"crosspred"*, and specifically:

- `predvar`: vector of observations used for prediction.
- `maxlag`: a positive value defining the maximum lag.
- `coef, vcov`: related coefficients and variance-covariance matrix from `model`.
- `matfit, matse`: matrices of effects and related standard errors for each predictor value in `predvar` and each lag in `0:maxlag`.
- `allfit, allse`: vectors of total effects and related standard errors for each predictor value in `predvar`. `allfit` is obtained summing the effects in `matfit` upon lags.
- `matRRfit`: exponentiated effects from `matfit`.
- `matRRlow, matRRhigh`: matrices with low and high 95% confidence intervals for `matRRfit`.
- `allRRfit`: exponentiated total effects from `allfit`.
- `allRRlow, allRRhigh`: vectors with low and high 95% confidence intervals for `allRRfit`.

All the effects are reported versus a reference value corresponding to the centering point for continuous functions (arguments `vartype` in `crossbasis()` equal to `"ns"`, `"bs"`, `"poly"` or `"lin"`) or to the default values for the other options (see Section 4.2 for further information). The exponentiated effects above are included if `model` has link equal to `log` or `logit`. The objects `coef` and `vcov` are extracted from `model` using the name of `crossbasis`: therefore, the name must not match other predictor names included in the model formula of `model`. If more than one variable included in the model is transformed by cross-basis functions, different names must be specified.

4.4 The function `crossplot()`

The function `crossplot()` produces several types of graphs of the predicted effects for a DLNM stored in an objects of class *"crosspred"*. The possible choices are 3-D graphs, plots of effects for specific predictor values or lags and overall effects. Type *`?crossplot`* in R for additional information.

The syntax of the command is:

```
crossplot(crosspred, type="3d", var=NULL, lag=NULL,
  ylim=NULL, title=NULL, label="var")
```

The argument `crosspred` must contain an object of class *"crosspred"* with the predicted values for a DLNM. The arguments `label` and `title` add a label to the x-axis and a title to the plot, respectively, while `xlim` sets the range of the y-axis. The arguments `var` and `lag` are used if `type="slices"` (see below). The type of plot specified in `type` are the following:

- `"3d"`: a 3-D plot generated by calling the function `persp()` in the package graphics.
- `"contour"`: a contour/level plot generated by calling the function `filled.contour()` in the package graphics.
- `"overall"`: a plot of the overall effects (summed up all the contributions at each lag).
- `"slices"`: a multiple plot of effects at specific values of predictor or lags, chosen by `var` and `lag`, respectively. Up to 4 plots for each dimension are allowed.

All the effects are reported versus a reference value corresponding to the centering point for continuous functions (arguments `vartype` in `crossbasis()` equal to `"ns"`, `"bs"`, `"poly"` or `"lin"`) or to the default values for the other options (see Section 4.2 for further information). If the model to carry out the estimates has link equal to `log` or `logit`, the exponentiated effect stored in the object `crosspred` are automatically plotted. The values in `var` and `lag` must match those specified in the object `crosspred` (see Section 4.3).

This function creates plots with default settings (i.e. perspective in 3-D plot, colours etc.). Refer to the original estimates stored in the `crosspred` object in order to personalize the output with generic plot commands.

5 Some examples

This Section provides some examples of the use of the various functions included in the dlnm package, described in Section 4.

First, some simple examples of the internal functions `mkbasis()` and `mklagbasis()` are showed in Section 5.1. Although these commands are not meant to be performed directly by the user, but are commonly called through `crossbasis()`, these codes can shine a light on the process to build the basis functions for the two dimensions (predictor and lags), and clarify the meaning of the arguments of the function `crossbasis()`.

Then, 3 different examples of the application of DLNM's are illustrated in the Sections 4.2-4.4, using the NMMAPS dataset for the city of Chicago in the period 1987-2000 described in Section 2.2. These different cases cover most of the functionalities of the package, providing a detailed overview of its capabilities and a basis to perform analyses on this dataset or on other data sources.

The models included in the examples do not contain other important confounders which are commonly accounted for in the time series analysis of the effect of environmental factors [7, 11, 2]. However, as already stated above, the examples are included only with the aim to illustrate the use of the functions in the dlnm package, and the results should not be considered as scientific evidences. In spite of this, these confouders can be easily added to the model formula in order to perform a proper statistical analysis, without any changes to the specification of the dlnm functions.

5.1 Examples for `mkbasis()` and `mklagbasis()`

As a first step, we provide an example of the use of the function `mkbasis()`. We build a basis matrix applying the selected basis functions to the vector of integers going form 1 to 5. We leave many of the arguments at their default values, apart from the selection of the degrees of freedom `df`.

```
> basis.var <- mkbasis(1:5, knots=3)
> basis.var

$basis
              b1          b2
[1,] -0.56626284  0.21084190
[2,] -0.20921622 -0.00635585
[3,]  0.00000000  0.00000000
[4,] -0.03716777  0.37894518
[5,] -0.22216593  0.98144395

$type
[1] "ns"

$df
[1] 2

$knots
[1] 3

$bound
[1] 1 5

$int
[1] FALSE

$cen
[1] TRUE

$cenvalue
[1] 3
```

The chosen basis is a natural cubic B-splines (default `type="ns"`) with the 1 knots and `df=2` (`df` is equal to `length(knots)+1+int` for `type="ns"`). Apart from the fact that the basis variables are centered at `cenvalue=3` (the mean of the predictor values, the default for this argument), the same results could be obtained by the command *`ns(1:5,knots=3)`*. The basis matrix is stored in the object `basis.var$basis`, while the arguments specifying it are included as other objects in the list, and can be called directly (for example, try *`basis.var$knots`*).

Alternative choices may be specified through the following code (results not shown, the user can try to run the commands):

```
> mkbasis(1:5, type="bs", df=4, degree=2)
> mkbasis(1:5, type="lin", cenvalue=4)
```

In the first case the result is a quadratic spline where the number and location of `knots` are chose automatically, and fixed to 2 (`df` is `length(knots)+degree+int` for this `type`) and at equally spaced quantiles, respectively. The second line returns a simple linear function, where the only transformation is the centering at the value of 4.

As explained in Section 4.1, the function `mklagbasis()` calls `mkbasis()` to create a basis matrix for the space of the lag. The basis functions are applied to the vector `0:maxlag` expressly created by the function. These are two examples of application:

```
> mklagbasis(maxlag=5, type="poly", degree=3)

$basis
     b1 b2 b3  b4
lag0  1  0  0   0
lag1  1  1  1   1
lag2  1  2  4   8
lag3  1  3  9  27
lag4  1  4 16  64
lag5  1  5 25 125

$type
[1] "poly"

$df
[1] 4

$degree
[1] 3

$int
[1] TRUE

$maxlag
[1] 5

> mklagbasis(maxlag=5, type="integer")

$basis
     b1 b2 b3 b4 b5 b6
lag0  1  0  0  0  0  0
lag1  0  1  0  0  0  0
lag2  0  0  1  0  0  0
lag3  0  0  0  1  0  0
lag4  0  0  0  0  1  0
```

```
lag5  0  0  0  0  0  1

$type
[1] "integer"

$df
[1] 6

$int
[1] TRUE

$maxlag
[1] 5
```

The first line specifies a 3rd degree polynomial. Differently from the bases for the space of the predictor build above, this matrix contains an intercept (`int=TRUE` by default), in this case a vector of 1's (see Section 4.2), and it is never centered. As stated above (Section 4.2), `df` is equal to the degree of the polynomial plus 1 when an intercept is included. In this case, for a polynomial basis, the argument `knots` is not included. The second example refers to an specific transformation in the space of lags in order to define unconstrained distributed lag effects (see Section 4.2), simply returning an identity matrix.

Other choices may consider a threshold parameterization, with the following code (results not shown):

```
> mkbasis(1:5, type="dthr", knots=c(2,3))
> mkbasis(1:5, type="hthr", knots=3)
> mkbasis(1:5, type="hthr", knots=c(2,3))
```

In the first example, the result is a double threshold basis which can be applied to describe linear effects below 2 and above 3, with a null effect in between them. In the second case, the choice is for a simple threshold with a linear effect above 3. In the last example, a piecewise linear basis is returned, which can describe a relationship with a first threshold at 2 and then an additional change in slope at 3.

A basis matrix of `type="strata"` with and without intercept is created by (results not shown):

```
> mklagbasis(maxlag=10, type="strata", knots=c(4,7))
> mklagbasis(maxlag=10, type="strata", knots=c(4,7), int=FALSE)
```

In this case, the intercept is represented by the dummy variable for the first stratum (see Section 4.2). The values in `knots` specify the cut-off point for the strata, and represent the lower boundaries for the right-open intervals.

The effect of centering is illustrated below (results not shown):

```
> mkbasis(0:10, type="poly", degree=3)
> mkbasis(0:10, type="poly", degree=3, cen=FALSE)
```

Each basis function is centered on the relative transformation of `cenvalue`, which is placed at the mean of the predictor values by default [6].

5.2 Example 1: a simple DLM

The dataset used in the next examples has already been loaded in the R session and prepared for the analysis in Section 2.2.

In this first example, we specify a simple DLM, assessing the effect of PM_{10} on overall mortality, while adjusting for the effect of temperature. In order to do so, we first build two cross-basis matrices for the two predictors, and then include them in a model formula of the command `glm()`. The effect of PM_{10} is assumed as linear in the dimension of the predictor, so, from this point of view, we can define this as a simple DLM even if it estimates also the distributed lag function for temperature, which is included as a non-linear term.

First, we run `crossbasis()` to build the two cross-basis matrices, saving them in two objects. The names of the two objects must be different in order to predict the effects separately for each of them (see Section 4.3). This is the code:

```
> basis.pm <- crossbasis(data$pm10, vartype="lin", lagtype="poly",
+   lagdegree=4,cen=FALSE,maxlag=15)
> basis.temp <- crossbasis(data$temp, vardf=5, lagtype="strata",
+   lagknots=1, cenvalue=21, maxlag=3)
```

The function `crossbasis()` calls the two internal functions `mkbasis()` and `mklagbasis()` to build the basis matrices. It passes the arguments with stub `var-` to the former, in order to specify the basis functions for the predictor (in this case `data$pm10` and `data$temp`), and the arguments with stub `lag-` to the latter, specifying the basis functions for the expressly created vector `0:maxlag`. Then it combines the two basis matrices to create the final cross-basis variables included in the objects of class *"crossbasis"* (`basis.pm` and `basis.temp`).

In this case, we assume that the effect of PM_{10} is linear (`vartype="lin"`), while we model the relationship with temperature through a natural cubic spline with 5 degrees of freedom (`vartype="ns"`, chosen by default). In this space, the internal knots (if not provided) are located by default at equally spaced quantiles, while the boundary knots are located at the range of the observed values, so we need to specify only `vardf`. We did not center PM_{10}, in order to describe the effect versus a reference value of 0 μgr/m^3 (the same results could be reached setting `cen=TRUE` and `cenvalue=0`). The reference value for temperature is set to 21°C.

The basis for the space of the lags is chosen through the same arguments but with stub `lag-`. We specify the lagged effect of PM_{10} up to 15 days of lag with a 4^{th} degree polynomial function (setting `lagdf=5`, i.e. the degree of the polynomial plus the intercept). The delayed effect of temperature are defined by two lag strata (0 and 1-3), assuming the effects as constant within each stratum. The argument `varknots=1` define the lower boundary of the second interval.

An overview of the specifications for the two cross-bases and bases is provided by the function `summary.crossbasis`, which calls the attributes of the `crossbasis` object:

```
> summary(basis.pm)
```

```
CROSSBASIS FUNCTIONS
Observations: 5114
Range: -3.104157 , 358.1367
Total df: 5
maxlag: 15

BASIS FOR VAR:
type: lin
df: 1

BASIS FOR LAG:
type: poly with degree 4
df: 5
with intercept
```

Now the two `crossbasis` objects can be included in a model formula in order to fit the DLM. In this case we model the effect assuming an overdispersed Poisson distribution:

```
> model <- glm(death ~  basis.pm + basis.temp, family=quasipoisson(), data)
```

The effects of specific levels of PM_{10} on overall mortality, predicted by the model above, can be computed by the function `crosspred()` and saved in an object with the same class:

```
> pred.pm <- crosspred(basis.pm, model,at=0:20)
```

The argument `at=0:20` states that the prediction must be computed for each integer value from 0 to 20 μgr/m^3. Now that the predicted effects have been stored in `pred.pm`, they can be plot by the function `crossplot()`. The first plot in Figure 1a summarizes the effect along lags of a 10-unit increases in PM_{10}, and it is created by:

```
> crossplot(pred.pm, "slices", var=10,
+   title="Effect of a 10-unit increase in PM10 along lags")
```

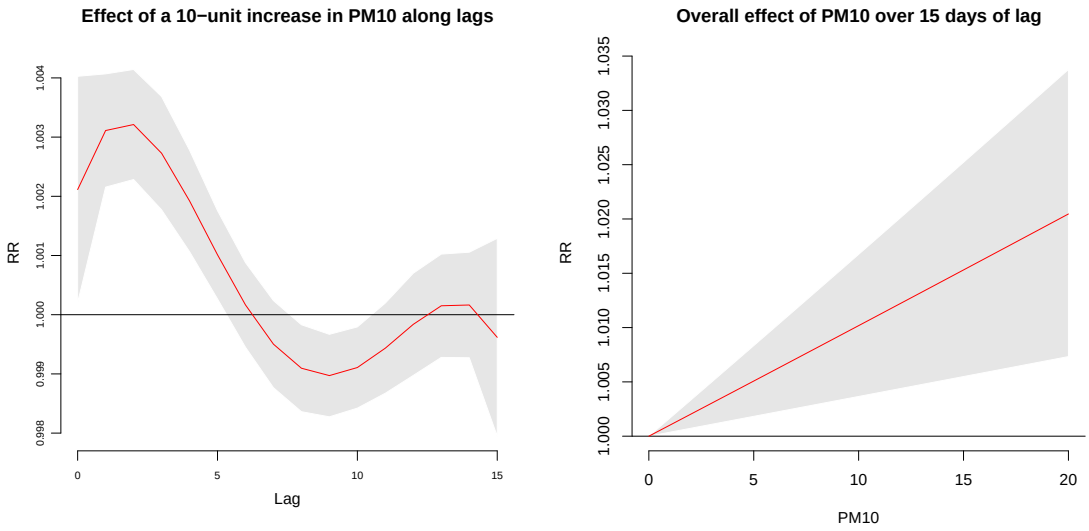
The argument `"slices"` defines that we want to graph the relationship at specific values of the two dimensions (predictor and lag). With `var=10` we specify this relationship along lags for a specific value of PM_{10}, i.e. 10 μgr/m^3. This effect is related to the reference value of 0 μgr/m^3, giving the effect for a 10-unit increase.

Then, we can compute the overall effect for a 10-unit increase in PM_{10} over 15 days of lag (i.e. summing all the effects up to the maximum lag), together with its 95% confidence intervals. These results are stored in the objects `allRRfit`, `allRRhigh` and `allRRlow` included in `pred.pm`, and can be extracted by:

```
> pred.pm$allRRfit["10"]

      10
1.010179
```

Figure 1

(a) (b)




```
> cbind(pred.pm$allRRlow, pred.pm$allRRhigh)["10",]
```

```
[1] 1.003657 1.016742
```

The overall effects for the selected range of PM_{10} versus 0 μgr/m^3 can be then plotted using the argument `type="overall"` in the function `crossplot()` (Figure 1b):

```
> crossplot(pred.pm, "overall", label="PM10",
+   title="Overall effect of PM10 over 15 days of lag")
```

5.3 Example 2: a threshold parameterization

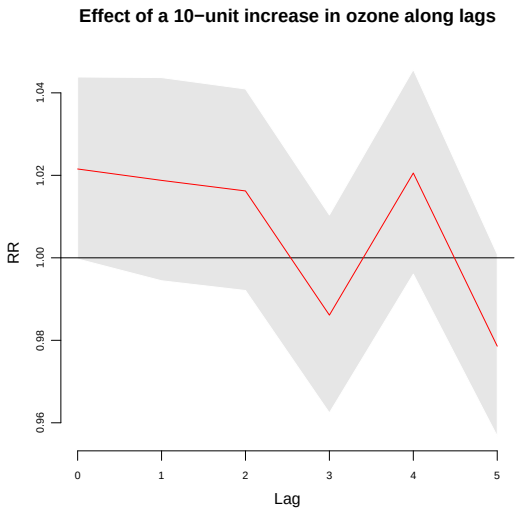
The purpose of the second example is to illustrate the use of the threshold parameterization. We assess the effect of ozone and temperature on overall mortality up to 5 and 15 days of lag, respectively, using the same steps already seen in Section 5.2.

Again, we first create the cross-basis matrices:

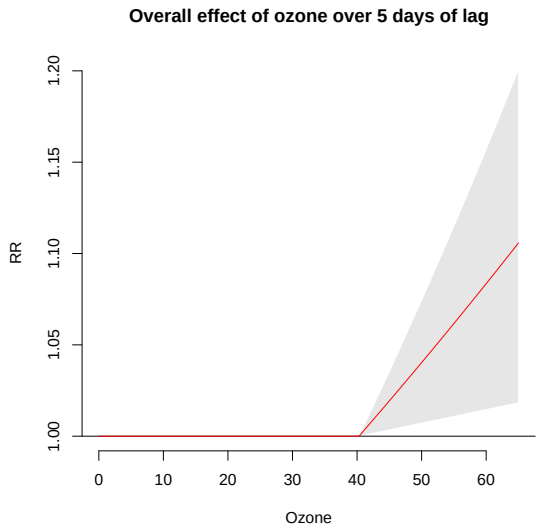
```
> basis.o3 <- crossbasis(data$o3, vartype="hthr", varknots=40.3,
+         lagtype="integer", maxlag=5)
> basis.temp <- crossbasis(data$temp, vartype="dthr", varknots=c(10,25),
+         lagtype="strata", lagknots=c(2,7), maxlag=15)
```

Figure 2

(a)



(b)



Here we make the assumption that the effect of O_3 is null up to 40.3 μgr/m^3 and then linear, applying an high threshold parameterization. For temperature, we use a double threshold with the assumption that the effect is linear below 10°C and above 25°C, and null in between. Regarding the lag dimension, we specify an unconstrained function for O_3, applying one parameter for each lag (`lagtype="integer"`) up to a 5 days. For temperature, we define 3 strata intervals at lag 0-1, 2-6, 7-15. A summary of the choices made for the cross-bases can be shown by the function `summary.crossbasis()`.

The estimates and predictions are carried out in the same way as in Section 5.2. The prediction range approximately equals the range of the observed values. The code is:

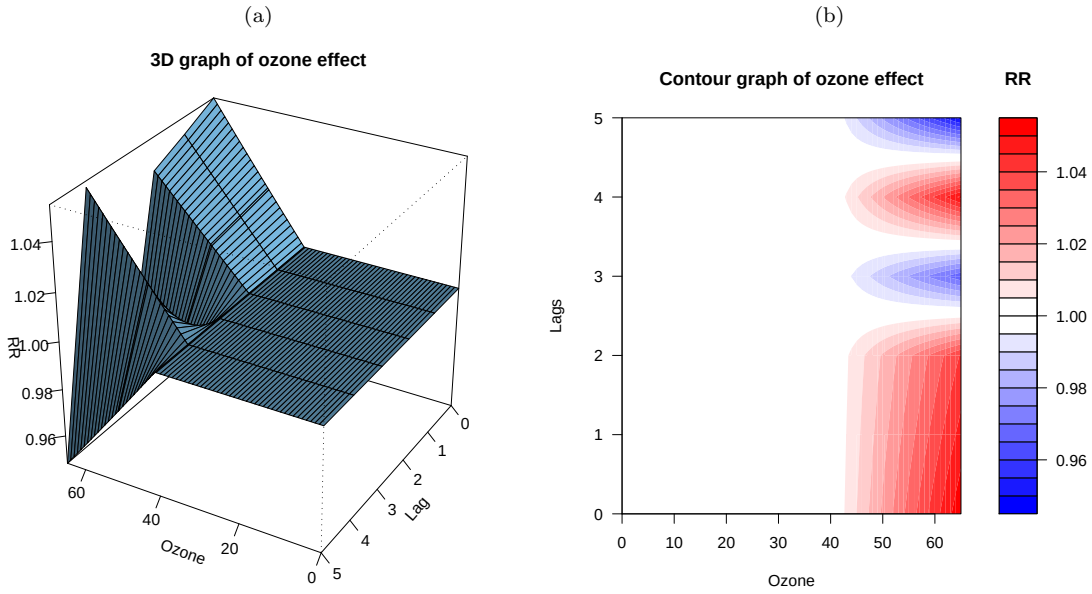
```
> model <- glm(death ~  basis.o3 + basis.temp, family=quasipoisson(), data)
> pred.o3 <- crosspred(basis.o3, model,at=c(0:65,40.3,50.3))
```

The values for which the prediction must be computed are specified in `at`: here we define the integers from 0 to 65 μgr/m^3 (approximately the range), plus the threshold and the value 50.3 μgr/m^3 corresponding to a 10-unit increase above the threshold, which is set as the reference point (see Section 4.2). The vector is automatically ordered. The command is (results in Figure 2a):

```
> crossplot(pred.o3, "slices", var=50.3,
+         title="Effect of a 10-unit increase in ozone along lags")
```

Similarly to Section 5.2, we can estimate and plot (Figure 5) the overall effect of a 10-unit increase in O_3 with 95% confidence intervals. In addition, we plot the three-dimensional effects (simultaneously

Figure 3



along the spaces of O_3 and lags) using two other options of the function `crossplot()`. The argument `type="3d"` specifies a 3-D plot, while `type="contour"` builds a contour/level plot (Figure 3a and 3b):

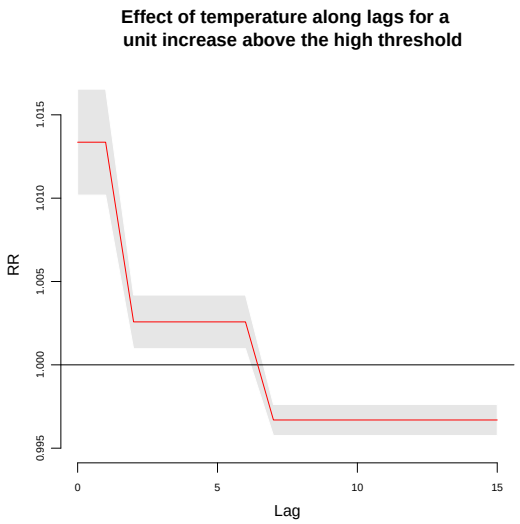
```
> pred.o3$allRRfit["50.3"]

    50.3
1.041538

> cbind(pred.o3$allRRlow, pred.o3$allRRhigh)["50.3",]

[1] 1.007359 1.076878

> crossplot(pred.o3,"overall",label="Ozone",
+         title="Overall effect of ozone over 5 days of lag")
> crossplot(pred.o3, label="Ozone", title="3D graph of ozone effect")
> crossplot(pred.o3, "contour", label="Ozone", title="Contour graph of ozone effect")
```

From the same model, we can predict and plot the effect of temperature, always in the range of the observed values:

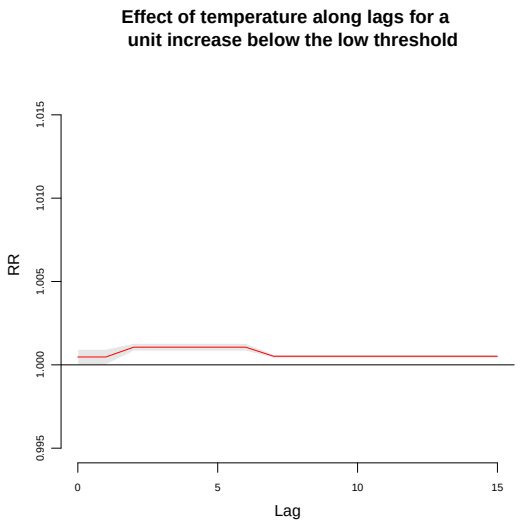
```
> pred.temp <- crosspred(basis.temp, model, at=-26:33)
> crossplot(pred.temp, "slices", var=26, ylim=c(0.995,1.017),
```

Figure 4

(a)



(b)




```
+         title="Effect of temperature along lags for a
+         unit increase above the high threshold")
> crossplot(pred.temp, "slices", var=9, ylim=c(0.995,1.017),
+         title="Effect of temperature along lags for a
+         unit increase below the low threshold")
```

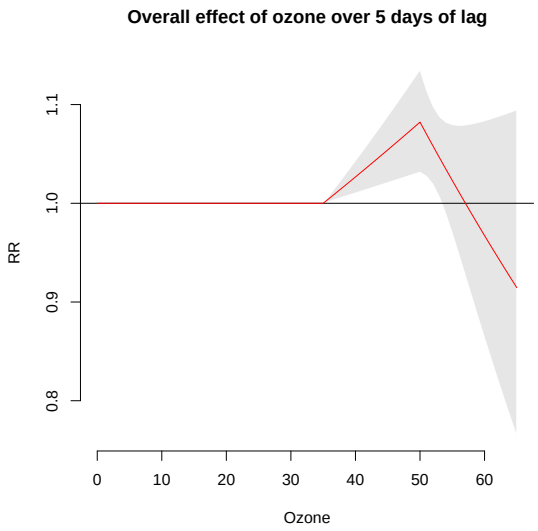
The effects along lags for 1°C above the high threshold (25°C) and below the low threshold (10°C) are showed in Figures 4a and 4b. The argument `ylim` is specified to set the same range in the y-axis, making the two plots comparable.

In order to illustrate a possible extension of the threshold models presented so far, we can assume that the O_3 effect is null up to 35 μgr/m^3 and then shows an additional change at 50 μgr/m^3. We can parameterize this effect with a piecewise linear function above a first threshold, changing slope at a specific cut-off point:

```
> basis.o3 <- crossbasis(data$o3, vartype="hthr", varknots=c(35,50),
+         lagtype="integer", maxlag=5)
> model <- update(model)
> pred.o3 <- crosspred(basis.o3, model, at=c(0:65))
```

Simply, we changed the cross-basis for O_3 including 2 `knots` which specify the threshold and the cut-off point for the change in slope, then we updated the model and predicted the results under the new assumptions. The overall effect can be plot by (Figure 5):

Figure 5




```
> crossplot(pred.o3,"overall",label="Ozone",
+         title="Overall effect of ozone over 5 days of lag")
```

This result can be compared with the same plot obtained under the simple threshold model in Figure 2b.

5.4 Example 3: a complex DLNM

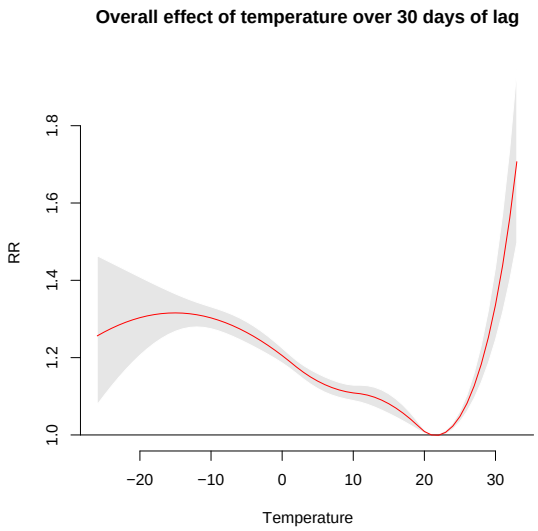
In this last example we specify a more complex DLNM, where the effects in both dimensions are estimated using smooth non-linear functions. Despite the higher complexity of the relationship, we will see how the steps required to specify and fit the model and predict and plot the results are exactly the same as for the simpler models see before in Sections 5.2-5.3. We apply this model to investigate the effects of temperature and PM_{10} on overall mortality up to lag 30 and 1, respectively.

These are the cross-basis matrices:

```
> basis.pm <- crossbasis(data$pm10,vartype="lin", lagtype="strata",
+   cen=FALSE, maxlag=1)
> basis.temp <- crossbasis(data$temp, vartype="bs", vardf=5, vardegree=2,
+   lagdf=5, cenvalue=21, maxlag=30)
```

The chosen basis functions for the space of the predictor are a linear function for the effect of PM_{10} and a quadratic B-spline (`vartype="bs"`) with 5 degrees of freedom for temperature (with `varknots` placed by default at equally spaced quantiles in the space of the predictor). The basis for temperature is centered at 21°C, which will represent the reference point for the predicted effects.

Figure 6



Regarding the space of lags, we assume a simple lag 0-1 parameterization for PM_{10} (i.e. a single strata up to lag 1, keeping the default values of `lagdf=1`), while we define another cubic spline, this time with the natural constraint (`vartype="ns"` by default) for the lag dimension of temperature. For this space, `lagknots` are located by default at equally spaced values in the log scale of lags, while the boundary knots are set to 0 and `maxlag`.

Now we can fit the model and carry out the prediction:

```
> model <- glm(death ~  basis.pm + basis.temp, family=quasipoisson(), data)
> pred.temp <- crosspred(basis.temp, model, at=-26:33)
```

Again, the prediction is performed at each integer value of temperature from -26 to 33°C, corresponding approximately to the range of the observed values.

The 3-D and contour plots equivalent to Figure 3a-3b and the overall effect summed all the contributions along lags is carried out by:

```
> crossplot(pred.temp, label="Temperature",
+   title="3D graph of temperature effect")
> crossplot(pred.temp, "contour", label="Temperature",
+   title="Contour graph of temperature effect")
> crossplot(pred.temp, "overall", label="Temperature",
+         title="Overall effect of temperature over 30 days of lag")
```

These results are showed in Figures 6 and 8a-8b.

Finally, a more detailed representation of the relationship is given plotting the effects at specific values of temperature and lags (Figure 7):

Figure 7

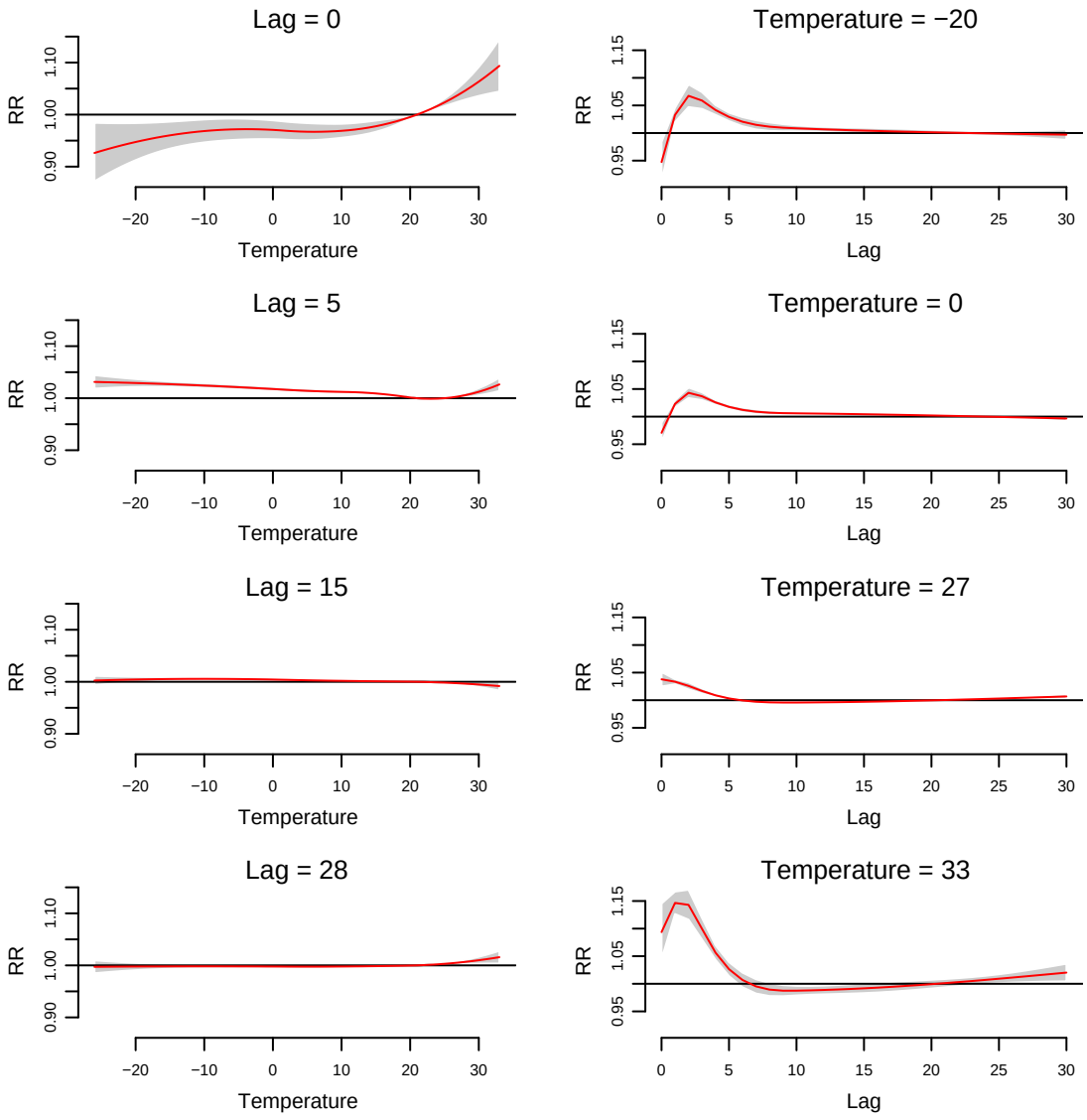
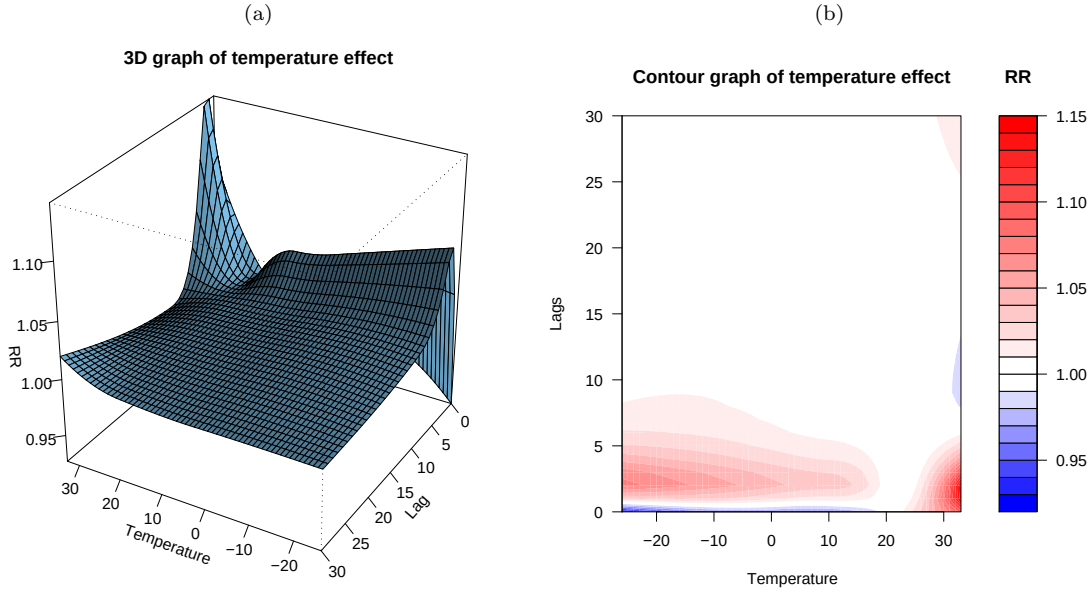

Lag = 0
RR
0.90
1.00
1.10
−20
−10
0
10
20
30
Temperature
Temperature = −20
RR
0.95
1.05
1.15
0
5
10
15
20
25
30
Lag
Lag = 5
RR
0.90
1.00
1.10
−20
−10
0
10
20
30
Temperature
Temperature = 0
RR
0.95
1.05
1.15
0
5
10
15
20
25
30
Lag
Lag = 15
RR
0.90
1.00
1.10
−20
−10
0
10
20
30
Temperature
Temperature = 27
RR
0.95
1.05
1.15
0
5
10
15
20
25
30
Lag
Lag = 28
RR
0.90
1.00
1.10
−20
−10
0
10
20
30
Temperature
Temperature = 33
RR
0.95
1.05
1.15
0
5
10
15
20
25
30
Lag

Figure 8




```
> crossplot(pred.temp, "slices", var=c(-20,0,27,33),
+   lag=c(0,5,15,28), label="Temperature")
```

The values in `var` and `lag` are used to define the specific predictor and lag values for which the effects must be computed. As a result, the plot in Figure 7 shows the effect by temperature at specific lags (left) and the effect by lag at specific temperatures (right). These plots can be imagined as the results of cutting "slices" on the effect surface shown in Figure 8a at the specific values of temperature and lags.

6 Conclusions

This document illustrates the functionalities of the dlnm package, providing a detailed overview of the process to specify and run a DLNM and then to predict and plot its results. The main advantage of this family of models is to unify many of the previous methods to deal with delayed effects in a unique framework, also providing more flexible alternatives regarding the shape of the relationships. Section 3 provides a brief summary of the theory underpinning DLNM's: a more detailed overview has been published elsewhere [3, 8], together with a complete specification of the algebra [8].

The flexibility is kept when this framework is implemented in the dlnm package: several different models with an increasing level of complexity can be performed using a simple and general procedure, as showed in the examples in Section 5. As already stated before, this method is not limited

to the examples on the effect of air pollution and temperature on mortality, but can be applied to investigate the relationship between any predictor and outcomes in time-series data.

7 Acknowledgements

This work was supported by the Medical Research Council (UK) through the Research Grant RES-G0707030.

We gratefully acknowledge the valuable suggestions of Fabio Frascati regarding the procedures to build and document this package, and the authors of other package vignettes used as examples (especially Heather Turner and David Firth for the package gnm). Moreover, we are thankful to the colleagues who tested the beta-version of this package and suggested some important improvements included in the current version (in particular Marie-France Valois).

Finally, we express our gratitude to all the people working to develop and maintain the R Project.

References

[1] S. Almon. The distributed lag between capital appropriations and expenditures. *Econometrica*, 33:178–196, 1965.

[2] B. G. Anderson and M. L. Bell. Weather-Related mortality: how heat, cold, and heat waves affect mortality in the United States. *Epidemiology*, 20(2):205–213, 2009.

[3] B. Armstrong. Models for the relationship between ambient temperature and daily mortality. *Epidemiology*, 17(6):624–31, 2006.

[4] M. Baccini, A. Biggeri, G. Accetta, T. Kosatsky, K. Katsouyanni, A. Analitis, H. R. Anderson, L. Bisanti, D. D'Ippoliti, J. Danova, B. Forsberg, S. Medina, A. Paldy, D. Rabczenko, C. Schindler, and P. Michelozzi. Heat effects on mortality in 15 European cities. *Epidemiology*, 19(5):711–9, 2008.

[5] A. L. Braga, A. Zanobetti, and J. Schwartz. The time course of weather-related deaths. *Epidemiology*, 12(6):662–7, 2001.

[6] J. Cao, M. F. Valois, and M. S. Goldberg. An S-Plus function to calculate relative risks and adjusted means for regression models using natural splines. *Comput Methods Programs Biomed*, 84(1):58–62, 2006.

[7] F. Dominici and R. T. Burnett. Risk models for particulate air pollution. *J Toxicol Environ Health A*, 66(16-19):1883–9, 2003.

[8] A. Gasparrini, B. Armstrong, and Kenward M. G. Distributed lag non-linear models. *Biostatistcs*, in submission.

[9] S. Hajat, B. G. Armstrong, N. Gouveia, and P. Wilkinson. Mortality displacement of heat-related deaths: a comparison of Delhi, Sao Paulo, and London. *Epidemiology*, 16(5):613–20, 2005.

[10] S. Pattenden, B. Nikiforov, and B. G. Armstrong. Mortality and temperature in Sofia and London. *J Epidemiol Community Health*, 57(8):628–33, 2003.

[11] R. D. Peng, F. Dominici, and T. A. Louis. Model choice in time series studies of air pollution and mortality. *J R Stat Soc Ser A*, 169(2):179–203, 2006.

[12] Roger D. Peng and Leah J. Welty. The NMMAPSdata Package. *R News*, 4(2):10–14, 2004. URL http://CRAN.R-project.org/doc/Rnews/.

[13] J. M. Samet, S. L. Zeger, F. Dominici, F. Curriero, I. Coursac, and D. W. Dockery. The National Morbidity, Mortality, and Air Pollution Study (NMMAPS). Part 2. Morbidity and mortality from air pollution in the United States. Technical report, Health Effects Institute, 2000.

[14] J. M. Samet, S. L. Zeger, F. Dominici, D. Dockery, and J. Schwartz. The National Morbidity, Mortality, and Air Pollution Study (NMMAPS). Part 1. Methods and methodological issues. Technical report, Health Effects Institute, 2000.

[15] J. Schwartz. The distributed lag between air pollution and daily deaths. *Epidemiology*, 11(3): 320–6, 2000.

[16] J. Schwartz. Is there harvesting in the association of airborne particles with daily deaths and hospital admissions? *Epidemiology*, 12(1):55–61, 2001.

[17] J. Schwartz, J. M. Samet, and J. A. Patz. Hospital admissions for heart disease: the effects of temperature and humidity. *Epidemiology*, 15(6):755–61, 2004.

[18] L. J. Welty and S. L. Zeger. Are the acute effects of particulate matter on mortality in the National Morbidity, Mortality, and Air Pollution Study the result of inadequate control for weather and season? A sensitivity analysis using flexible distributed lag models. *Am J Epidemiol*, 162(1):80–8, 2005.

[19] S.N. Wood. *Generalized additive models: an introduction with R*. Chapman & Hall/CRC, 2006.

[20] A. Zanobetti and J. Schwartz. Mortality displacement in the association of ozone with mortality: an analysis of 48 cities in the United States. *Am J Respir Crit Care Med*, 177(2):184–9, 2008.

[21] A. Zanobetti, M. P. Wand, J. Schwartz, and L. M. Ryan. Generalized additive distributed lag models: quantifying mortality displacement. *Biostatistics*, 1(3):279–92, 2000.