Comprehensive Tutorial for the Spatio-Temporal R-package

Silas Bergen
University of Washington

Johan Lindström
University of Washington
Lund University

7th March 2012

Contents

1 Introduction

A complex spatio-temporal modelling framework has been developed for the air pollution data collected by the Multi-Ethnic Study of Atherosclerosis and Air Pollution (MESA Air) (Sampson et al., 2009, Szpiro et al., 2010, Lindström et al., 2011). The framework provides a spatio-temporal model that can accommodate arbitrarily missing observations and allows for a complex spatio-temporal correlation structure. To allow for a more wide spread use of the models an R-package, `SpatioTemporal`, has been developed.

The goal of this tutorial is to provide an introduction to the `SpatioTemporal`-package. The tutorial is broadly organised in a theoretical part giving a brief overview of the model and theory (Sections 2-3), followed by a practical part that describes how to use the model for parameter estimation and prediction.

To allow for more realistic examples some real data has been included in the package, this data is described in section 2, with section 3 providing a brief presentation of the theory. Following this background the actual R-tutorial begins in section 4 with an overview of the available data and instructions on how to create the data-structures needed for the model fitting. Estimation and evaluation of smooth temporal trends is described in section 5, and section 6 provides some empirical estimates that, eventually, are compared to corresponding results for the full model. Functions that do parameter estimation and prediction are introduced in section 7, along with tools for illustration of the results. The last part of the R-tutorial is a cross-validation example in section 8.

All the code used in this tutorial has been collected in Appendix A. The Appendices also contain commented code for some additional examples: Appendix B gives an example of predictions at unobserved locations, a case with a spatio-temporal covariate is demonstrated in Appendix D, Appendix E provides the outlines of a simulation study, and an MCMC example is given in Appendix C.

Before starting with the full tutorial it seems prudent to discuss some of the key assumptions in the model along with some common problems that might arise when using the model.

1.1 Key Assumptions

The following two sections provide lists of key assumptions and common problems. The lists are presented here for reference and migth be more understandable *after reading* the tutorial than before.

Some of the key assumptions in the model are:

- Any temporal structure is captured by the smooth temporal trends
- The resulting residuals are *temporal independent.*
- The spatial dependencies can be described using *stationary, exponential* covariance functions.
- No *colocated observations.*

1.2 Common Problems — Troubleshooting

The following gives some common problems – that might arise when using the package – and possible solutions.

If the parameter estimation fails consider:

- Covariate scaling, try to avoid that covariates have *extremely different ranges*; this may cause numerical instabilities.
- The meaning of the parameters, compare the starting values to what occurrs in the actual data.
- Try multiple starting values in the optimisation.
- Changing location coordinates from kilometres to metres will *drastically change the reasonable values of the range.*
- An *over parameterised* (too many covariates) model may cause numerical problems.

Other common problems are:

- Ensure that geographic covariates are provided for *all locations.*
- Ensure that spatio-temporal covariates are provided for *all time-points and locations.*
- The spatio-temporal covariate(s) *must be in a 3D-array.*
- The model does not handle *colocated observations*, however predictions at multiple, unobserved, colocated sites work.

2 Data

The data used in this tutorial consists of a subset of the $\mathrm{NO_x}$ measurements from coastal Los Angeles available to the MESA Air study; as well as a few (geographic) covariates. A detailed description of the full dataset can be found in Cohen et al. (2009), Szpiro et al. (2010), Lindström et al. (2011); short descriptions of the AQS and MESA monitoring can be found in subsubsection 2.1.1 and 2.1.2 below, while subsection 2.2 and 2.3 describes the covariates.

2.1 $\mathrm{NO_x}$ Observations

The data consists of measurements from the national AQS network of regulatory monitors as well as supplementary MESA Air monitoring. The data has been aggregated to *2-week averages*. Since the distribution of the resulting 2-week average $\mathrm{NO_x}$ concentrations (ppb) is skewed, the data has also been *log-transformed*.

2.1.1 Air Quality System (AQS) data

The national AQS network of regulatory monitors consists of a modest number of fixed sites that measure ambient concentrations of several different air pollutants including $\mathrm{NO_x}$ and $\mathrm{PM_{2.5}}$. Many AQS sites provide hourly averages for $\mathrm{NO_x}$, while monitoring of $\mathrm{PM_{2.5}}$ is less frequent. The data in this tutorial includes $\mathrm{NO_x}$ data from 20 AQS sites in and around Los Angeles.

2.1.2 MESA Air data

The AQS monitors provide data with excellent temporal resolution, but only at relatively few locations. As pointed out in Szpiro et al. (2010), potential problems with basing exposure estimates entirely on data from the AQS network are: 1) the number of locations sampled is limited; 2) the AQS network is designed for regulatory rather than epidemiology purposes and does not resolve small scale spatial variability; and 3) the network has siting restrictions that limit its ability to resolve near-road effects. To address these restrictions the MESA Air supplementary monitoring campaign was designed to provide increased diversity in geographic monitoring locations, with specific importance placed on proximity to traffic.

The MESA Air supplementary monitoring consists of three sub-campaigns, Cohen et al. (see 2009, for details): “fixed sites”, “home outdoor”, and “community snapshot”. Only data from the “fixed sites” have been included in this tutorial; this campaign consisted of a few fixed site monitors that provided 2-week averages during the entire MESA Air monitoring period. To allow for comparison of the different monitoring protocols, one of the MESA fixed sites in coastal Los Angeles was colocated with an existing AQS monitor.

2.2 Geographic Information System (GIS)

To predict ambient air pollution at times and locations where we have no measurements MESA Air uses a complex spatio-temporal model that includes regression on geographic covariates (see section 3 for a brief overview). The covariates used in this tutorial are: 1) distance to a major road, i.e., census feature class code A1–A3 (distances truncated to be $\geq$10m and log-transformed) and the minimum of these distances, 2) distance to coast (truncated to be $\leq$15km), and 3) average population density in a 2 km buffer. For details on the variable selection process that lead to these covariates as well as a more complete list of the covariates available to MESA Air the reader is referred to Mercer et al. (2011).

2.3 Caline Dispersion Model for Air Pollution

The geographic covariates described above are fixed in time and provide only spatial information. To aid in the spatio-temporal modelling, covariates that vary in both space and time would be valuable. One option is to integrate output from deterministic air pollution models into the spatio-temporal model. The example in this tutorial contains output from a slightly modified version of Caline3QHC (EPA, 1992, Wilton et al., 2010, MESA Air Data Team, 2010).

Caline is a line dispersion model for air pollution. Given locations of major (road) sources and local meteorology Caline uses a Gaussian model dispersion to predict how nonreactive pollutants travel with the wind away from sources; providing hourly estimates of air pollution at distinct points. The hourly contributions from Caline have then been averaged to produce a 2-week average spatio-temporal covariate. It should be noted that the Caline predictions in this tutorial only includes air pollution due to traffic on major roads (A1, A2, and large A3).

3 Model and Theory

The R-package described here implements the model developed by Szpiro et al. (2010), Lindström et al. (2011), and the reader is referred to those papers for extensive model details. Here we will give a brief description, which hopefully suffices for the purpose of this tutorial.

Denoting the quantity to be modelled (in this example ambient 2-week average log $\mathrm{NO_x}$ concentrations) by $y(s,t)$, we write the spatio-temporal field as

$$y(s,t) = \mu(s,t) + \nu(s,t), \tag{1}$$

where $\mu(s,t)$ is the predictable mean field and $\nu(s,t)$ is the essentially random space-time residual field. The mean field is modelled as

$$\mu(s,t) = \sum_{l=1}^{L} \gamma_l \mathcal{M}_l(s,t) + \sum_{i=1}^{m} \beta_i(s) f_i(t), \tag{2}$$

where the $\mathcal{M}_l(s,t)$ are spatio-temporal covariates; γ_l are coefficients for the spatio-temporal covariates; $\{f_i(t)\}_{i=1}^m$ is a set of smooth basis functions, with $f_1(t) \equiv 1$; and the $\beta_i(s)$ are spatially varying coefficients for the temporal trends.

In (2) the term, $\sum_{i=1}^m \beta_i(s) f_i(t)$, is a linear combination of temporal basis functions weighted by coefficients that vary between locations. Typically the number of basis functions will be small.

We model the spatial fields of β_i-coefficients using universal kriging (Cressie, 1993). The trend in the kriging is constructed as a linear regression on geographical covariates; following Jerrett et al. (2005) we call this component a "land use" regression (LUR). The spatial dependence is assumed to be exponential with *no nugget*. The resulting models for the β-fields are

$$\beta_i(s) \in \mathsf{N}\left(X_i \alpha_i, \Sigma_{\beta_i}(\theta_i)\right) \quad \text{for } i = 1, \ldots, m, \tag{3}$$

where X_i are $n \times p_i$ design matrices, α_i are $p_i \times 1$ matrices of regression coefficients, and $\Sigma_{\beta_i}(\theta_i)$ are $n \times n$ covariance matrices. Note that the design matrices, X_i, can incorporate different geographical covariates for the different spatial fields. The $\beta_i(s)$ fields are assumed to be independent of each other.

Finally the model for the residual space-time field, $\nu(s,t)$, is assumed to be independent in time and have exponential covariance given by

$$\nu(s,t) \in \mathsf{N}\left(0, \Sigma_\nu^t(\theta_\nu)\right) \quad \text{for } t = 1, \ldots, T,$$

where the size of the exponential covariance matrices, $\Sigma_\nu^t(\theta_\nu)$, is the number of observations, n_t, at each time-point. The temporal independence is based on the assumption that the mean model, $\mu(s,t)$, accounts for most of the temporal correlation.

3.1 Model parameters

The parameters of the model consist of the regression parameters for the geographical, and spatio-temporal covariates, respectively

$$\boldsymbol{\alpha} = (\alpha_1^\top, \ldots, \alpha_m^\top)^\top; \qquad\qquad \boldsymbol{\gamma} = (\gamma_1, \ldots, \gamma_L)^\top, \tag{4}$$

spatial covariance parameters for the β_i-fields,

$$\theta_B = (\theta_1, \ldots, \theta_m) \quad \text{where} \quad \theta_i = (\phi_i, \sigma_i^2), \tag{5}$$

and covariance parameters of the spatio-temporal residuals,

$$\theta_\nu = (\phi_\nu, \sigma_\nu^2, \tau_\nu^2). \tag{6}$$

The model assumes exponential covariance functions with range ϕ, partial sill σ^2, and nugget τ^2. The nuggets of the β_i-fields are set to zero.

Other covariance functions are *not currently supported*, but this may change in future versions.

To simplify notation we collect the covariance parameters into Ψ,

$$\Psi = (\theta_1, \ldots, \theta_m, \theta_\nu).$$

Combining (1) and (2) our model is

$$y(s,t) = \sum_{l=1}^{L} \gamma_l \mathcal{M}_l(s,t) + \sum_{i=1}^{m} \beta_i(s) f_i(t) + \nu(s,t). \tag{7}$$

4 Preliminaries

In this tutorial `typewrite text` denotes R-commands or variables, lines prefixed by `##` are comments. To exemplify

```
##A qqplot for some N(0,1) random numbers
qqnorm(rnorm(100))
```

In some cases both function calls and output is provided; the call(s) are prefixed by > while rows without a prefix represent function output, e.g.

```
##Summary of 100 N(0,1) numbers.
> summary(rnorm(100))
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
-2.5870 -0.2815  0.2574  0.2252  0.8056  2.3370
```

Some of the code in this tutorial takes considerable time to run, in these cases precomputed results have been included in package data-files. The tutorial marks time consuming code with the following warning/alternative statements:

WARNING: The following steps are time-consuming.

```
Some time consuming code
```

ALTERNATIVE: Load pre-computed results.

```
An option to load precomputed results.
```

End of alternative

Here we will study NO_x data from Los Angeles. The data are described in section 2 and consist of 25 different monitor locations, with 2-week average log NO_x concentrations measured for 280 2-week periods.

First load the package, along with a few additional packages need by the tutorial:

```
##Load the SpatioTemporal package
library(SpatioTemporal)

##A package used for some plots in this tutorial (plotCI)
library(plotrix)
###The maps to provide reference maps
library(maps)
```

The sample data are contained in `mesa.data`, which is included in the package and can be loaded as follows:

```
###Load the data
data(mesa.data)
##...and the precomputed results
data(mesa.data.res)
```

4.1 Examining the Data

First lets examine the components of `mesa.data`.

```
> names(mesa.data)
[1] "location"   "LUR"     "trend"     "obs"   "SpatioTemp"
```

The structure `mesa.data` contains a list, where each element of the list is a data frame. We will now take a closer look at each one of these data frames.

4.1.1 The `mesa.data$location` Data Frame

We begin our examination of the data by investigating `mesa.data$location`:

```
> head(mesa.data$location)
 ID          x         y        long     lat    type
 60370002 -10861.67 3793.589 -117.923 34.1365  AQS
 60370016 -10854.95 3794.456 -117.850 34.1443  AQS
 60370030 -10888.66 3782.332 -118.216 34.0352  AQS
 60370031 -10891.42 3754.649 -118.246 33.7861  AQS
 60370113 -10910.76 3784.099 -118.456 34.0511  AQS
 60371002 -10897.96 3797.979 -118.317 34.1760  AQS
```

The `location` data frame is a 25 x 6 data frame. The first column contains the ID names for each of the 25 air pollution monitoring sites in the data set as factors or strings; the second and third columns contain x- and y-coordinates, which are used to calculate distances between sites. The fourth and fifth columns contain longitude and latitude coordinates. These coordinates are optional — not required for modelling — but can be included in the data structure to simplify plotting the locations on a map. For this dataset the x- and y-coordinates (given in km) are actually computed from the longitude and latitude as

$$x = 111.13 \cdot \mathrm{long} \cdot \cos(34.021 \cdot \pi/180) \qquad y = 111.13 \cdot \mathrm{lat}.$$

The last column describes the type of monitoring system to which that site belongs. In this example, we have two types: `AQS` refers to the EPA's regulatory monitors that are part of the Air Quality System, while `FIXED` refers to the MESA Air fixed site locations (see subsubsection 2.1.2). If included, this column should contain factors or characters. As with the longitude and latitude coordinates, it too is optional; it can be used in some routines to subset data. Although we have observations at all the locations in this example, one could also include data for locations in `mesa.data$location` that don't have observations in order to predict at those locations (see Appendix B for a prediction example).

The following code plots these locations on a map, shown in Figure 1; the code uses the optional longitude and latitude coordinates.

```
###Plot the locations, see Figure 1
par(mfrow=c(1,1))
plot(mesa.data$location$long,mesa.data$location$lat,
     pch=24,bg=c("red","blue")[mesa.data$location$type],
     xlab="Longitude",ylab="Latitude")

###Add the map of LA
map("county","california",col="#FFFF0055",fill=TRUE,add=TRUE)

##Add a legend
legend("bottomleft",c("AQS","FIXED"),pch=24,bty="n",
       pt.bg=c("red","blue"))
```

4.1.2 The `mesa.data$LUR` Data Frame

Now, study the `LUR` data frame:

```
> head(mesa.data$LUR)
         log10.m.to.a1 log10.m.to.a2 log10.m.to.a3
60370002      2.861509      4.100755      2.494956
60370016      3.461672      3.801059      2.471498
60370030      2.561133      3.695772      1.830197
60370031      3.111413      2.737527      2.451927
60370113      2.762193      3.687412      2.382281
60371002      2.760931      4.035977      1.808260
         log10.m.to.road km.to.coast s2000.pop.div.10000
60370002        2.494956   15.000000            1.733283
60370016        2.471498   15.000000            1.645386
60370030        1.830197   15.000000            6.192630
60370031        2.451927    1.023311            2.088930
60370113        2.382281    6.011075            7.143731
60371002        1.808260   15.000000            4.766780
```

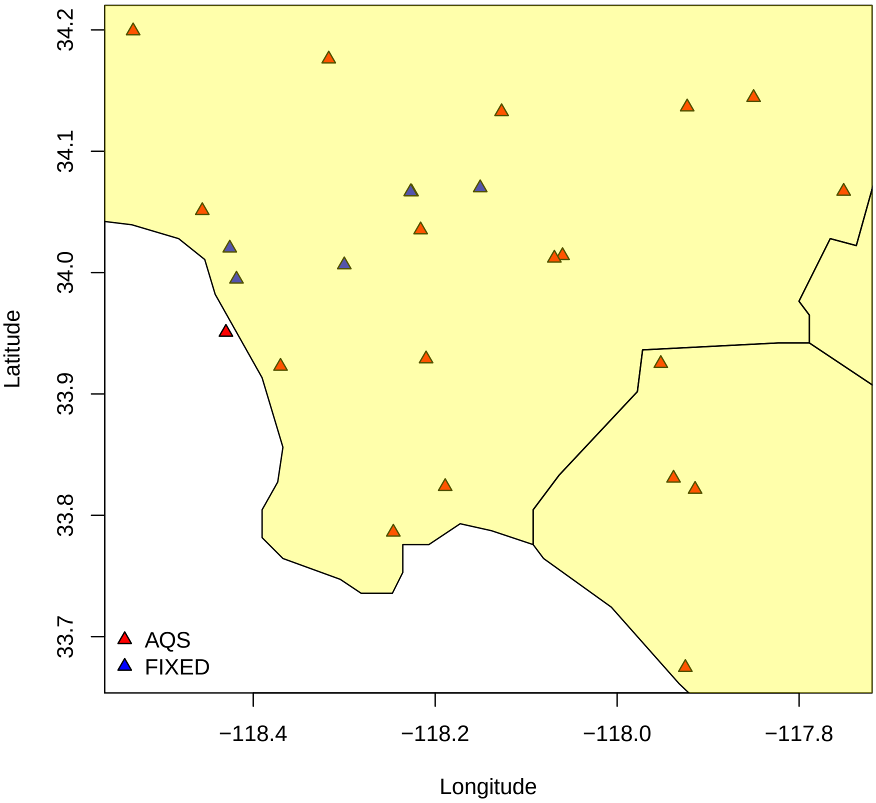

Figure 1: Location of monitors in the Los Angeles area.

The LUR data frame contains GIS covariate values for each of the 25 monitoring locations for use in the land use regression (LUR) part of the model. In this example, the variables in LUR are $\log_{10}$ meters to A1, A2, A3 roads and the minimum of these three measurements (accounting for the four initial columns in LUR); kilometres to the coast; and average population density in a 2 km buffer (divided by 10,000). Note that the row names of LUR are the same as the location ID's in `mesa.data$location`, and are used to link each set of land use regression covariates to a specific site in `mesa.data$location`. As with `mesa.data$location`, one could include LUR's for locations at which one wants to predict.

4.1.3 The `mesa.data$trend` Data Frame

Next, look at the third element, `mesa.data$trend`:

```
> head(mesa.data$trend)
                   V1         V2       date
1999-01-13 -1.8591693 1.20721096 1999-01-13
1999-01-27 -1.5200057 0.90473775 1999-01-27
1999-02-10 -1.1880840 0.62679098 1999-02-10
1999-02-24 -0.8639833 0.38411634 1999-02-24
1999-03-10 -0.5536476 0.19683161 1999-03-10
1999-03-24 -0.2643623 0.08739755 1999-03-24
```

The `trend` data frame consists of 2 smooth temporal basis functions computed using singular value decomposition (SVD), see section 5 for details. These temporal trends corresponds to the $f_i(t)$:s in (2). The spatio-temporal model also includes an intercept, i.e. a vector of 1's; the intercept is added automatically and *should not be included* in `trend`.

The `mesa.data$trend` data frame is 280 × 3, where 280 is the number of time points for which we have NO_x concentration measurements. Here, the first two columns contain smooth temporal trends, and the last column contains dates in the `R date` format. In general, *one of the columns* in `mesa.data$trend` *must* be called `date` and have dates in the `R date` format; the names of the other columns are arbitrary. Studying the `date` component,

```
> range(mesa.data$trend$date)
[1] "1999-01-13" "2009-09-23"
```

we see that measurements are made over a period of about 10 years, from January 13, 1999 until September 23, 2009.

4.1.4 The `mesa.data$obs` Data Frame

The observations are stored in `mesa.data$obs`:

```
> head(mesa.data$obs)
       obs       date       ID
1 4.577684 1999-01-13 60370002
2 4.131632 1999-01-13 60370016
3 4.727882 1999-01-13 60370113
```

```
4 5.352608 1999-01-13 60371002
5 5.281452 1999-01-13 60371103
6 4.984585 1999-01-13 60371201
```

The data frame, `mesa.data$obs`, consists of observations, over time, for each of the 25 locations. The data frame contains three variables: `obs` — the measured log $\mathrm{NO_x}$ concentrations (as 2-week averages); `date` — the date of each observation (here the middle Wednesday of each 2-week period); and `ID` — labels indicating at which monitoring site each measurement was taken. Details regarding the monitoring can be found in Cohen et al. (2009), and a brief introduction is given in subsection 2.1.

The ID values should correspond to the ID of the monitoring locations given in `mesa.data$location$ID`. The dates in `mesa.data$obs` do *not* need to correspond exactly to the dates in `mesa.data$trend$date`; however they *have to be in the range* of `mesa.data$trend$date` (in this case, 1999-01-13 through 2009-09-23).

Note that the number of rows in `mesa.data$obs` is 4577, far fewer than the $280 \times 25 = 7000$ observations there would be if each location had a complete time series of observations.

4.1.5 The `mesa.data$SpatioTemporal` Array

Finally, examine the `mesa.data$SpatioTemporal` data:

```
> dim(mesa.data$SpatioTemp)
[1] 280  25   1

> mesa.data$SpatioTemp[1:5,1:5,]
           60370002 60370016 60370030 60370031 60370113
1999-01-13   2.3188        0   8.0641   0.1467   2.9894
1999-01-27   1.8371        0   7.3568   0.2397   4.7381
1999-02-10   1.4886        0   6.3673   0.2463   4.3922
1999-02-24   2.5868        0   7.1783   0.1140   3.3456
1999-03-10   1.8996        0   6.3159   0.1537   3.8495
```

The `mesa.data$SpatioTemp` element should be a *three dimensional array* containing spatio-temporal covariates. In this example dataset we have only one covariate, which is the output from the Caline3QHC model (see subsection 2.3 for a brief overview; more details can be found in EPA (1992), MESA Air Data Team (2010)) If *no* spatio-temporal covariates are used/needed `mesa.data$SpatioTemp` should be set to `NULL`.

Of the three dimensions of `mesa.data$SpatioTemp`, the first (280) refers to the number of time points where we have spatio-temporal covariate measurements, the second (25) refers to the number of locations, and the third (1) refers to the number of different spatio-temporal covariates. Though the entire array is not shown here, it should be noted that values of the spatio-temporal covariate are specified for all 280-by-25 space-time locations. Again, this array could contain values of the spatio-temporal covariate(s) at times and/or locations that do not have observations, in order to predict at those times/locations.

The names of of the `SpatioTemp` array are used to match covariates with observations

```
> str(dimnames(mesa.data$SpatioTemp))
List of 3
 $ : chr [1:280] "1999-01-13" "1999-01-27" "1999-02-10" ...
 $ : chr [1:25] "60370002" "60370016" "60370030" "60370031" ...
 $ : chr "lax.conc.1500"
```

The rownames should match the dates of observations and the temporal trends, i.e. they should be given by

```
as.character(sort(unique(c(mesa.data$obs$date,
    mesa.data$trend$date))))
```

the column names should match the location ID's in `mesa.data$location$ID`, and the names of the third dimension

```
> dimnames(mesa.data$SpatioTemp)[[3]]
[1] "lax.conc.1500"
```

identifies the different spatio-temporal covariates.

To simplify, the main model fitted in this tutorial *does not* include any spatio-temporal covariate. However, Appendix D demonstrates predictions with a spatio-temporal covariate.

4.2 Summaries of `mesa.data`

Now that we have gone over a detailed description of what is in the `mesa.data` object, we can use the following function to examine a summary of the observations:

```
> printMesaDataNbrObs(mesa.data)
Nbr locations: 25 (observed: 25)
Nbr time points: 280 (observed: 280)
Nbr obs: 4577
Trend dates: 1999-01-13 to 2009-09-23
Observed dates: 1999-01-13 to 2009-09-23
All sites:
  AQS FIXED
   20     5
Observed:
  AQS FIXED
   20     5
For AQS:
  Number of obs: 4178
  Dates: 1999-01-13 to 2009-09-23
For FIXED:
  Number of obs: 399
  Dates: 2005-12-07 to 2009-07-01
```

Here we can see the number of AQS and MESA FIXED sites in the `mesa.data` structure. There are 20 AQS sites, which correspond to the number of locations marked as AQS in `mesa.data$location$type`, and 5 FIXED sites, which correspond to the 5 sites flagged as FIXED in `mesa.data$location$type`. We can also see that the observations are made over the same range of time as the temporal trends; this is appropriate, as discussed above. The summary also indicates the total number of sites (and time points) as well as how many of these that have been observed, `Nbr locations: 25 (observed: 25)`. In this example all of our locations have been observed; Appendix B provides an example with unobserved locations.

To graphically depict where and when our observation occurred we plot the monitor locations in time and space.

```
###Plot when observations occurr, see Figure 2

par(mfcol=c(1,1),mar=c(4.3,4.3,1,1))
plotMonitoringLoc(mesa.data)
```

From Figure 2 we see that the MESA monitors only sampled during the second half of the period. We also note that the number of observations vary greatly for different locations.

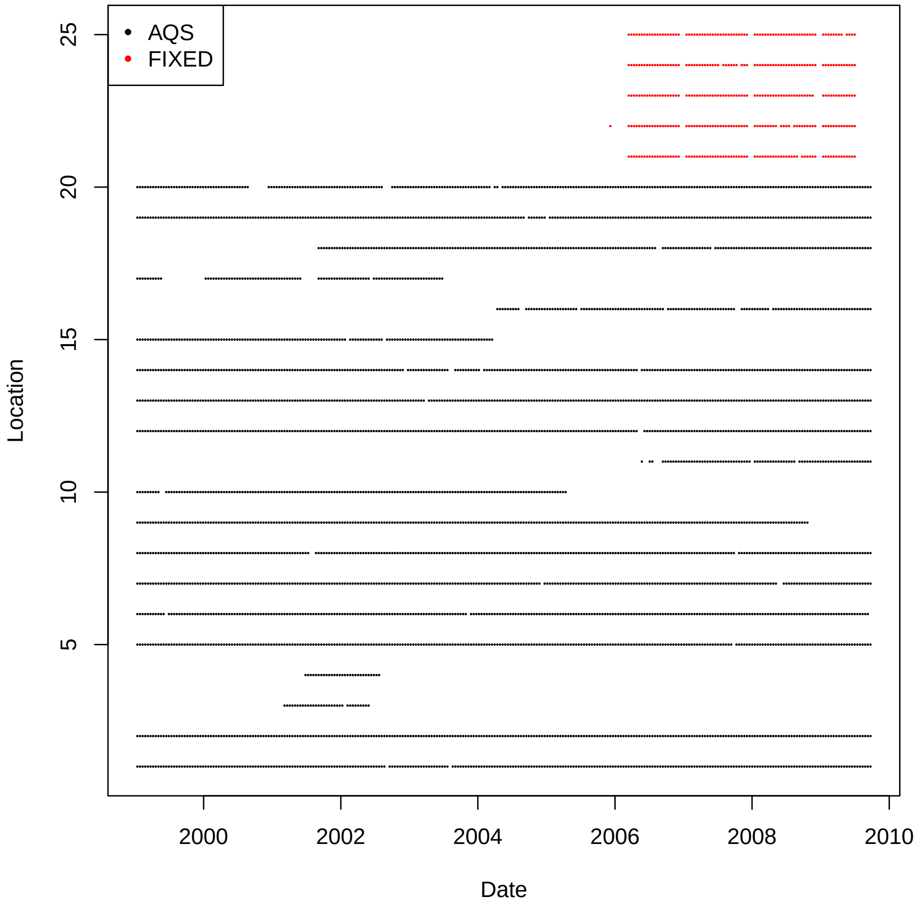

Figure 2: Space-time location of all our observations.

4.3 Creating `mesa.data` from Raw Data

The previous Sections (4.1–4.2) describe the expected elements of the `mesa.data` data structure. This data structure is used by the functions of the SpatioTemporal package; it will be assumed to follow the format outlined above.

In practise, however, data that users of the package have might not be in the exact format of the `mesa.data` structure. The following section describes how to create this structure from one possible raw dataset. The SpatioTemporal package contains an example of a raw data structure, which we now load and examine.

```
> data(mesa.data.raw)
> names(mesa.data.raw)
```

```
[1] "X"              "obs"            "lax.conc.1500"
```

As we can see, the raw data contains a list of three data sets: "X", "obs", and "lax.conc.1500". For comparison, recall the elements of the mesa.data structure:

```
> names(mesa.data)  ###Recall the data frames in mesa.data
[1] "location"   "LUR"     "trend"    "obs"  "SpatioTemp"
```

We will extract information from mesa.data.raw to create the necessary components of the mesa.data structure. But first, a description of what raw data might look like.

```
> head(mesa.data.raw$X)
        ID         x        y     long     lat type
1 60370002 -10861.67 3793.589 -117.923 34.1365  AQS
2 60370016 -10854.95 3794.456 -117.850 34.1443  AQS
3 60370030 -10888.66 3782.332 -118.216 34.0352  AQS
4 60370031 -10891.42 3754.649 -118.246 33.7861  AQS
5 60370113 -10910.76 3784.099 -118.456 34.0511  AQS
6 60371002 -10897.96 3797.979 -118.317 34.1760  AQS
  log10.m.to.a1 log10.m.to.a2 log10.m.to.a3
1      2.861509      4.100755      2.494956
2      3.461672      3.801059      2.471498
3      2.561133      3.695772      1.830197
4      3.111413      2.737527      2.451927
5      2.762193      3.687412      2.382281
6      2.760931      4.035977      1.808260
  log10.m.to.road km.to.coast s2000.pop.div.10000
1        2.494956   15.000000            1.733283
2        2.471498   15.000000            1.645386
3        1.830197   15.000000            6.192630
4        2.451927    1.023311            2.088930
5        2.382281    6.011075            7.143731
6        1.808260   15.000000            4.766780
```

Above we can see an excerpt of mesa.data.raw$X. In this example, mesa.data.raw$X contains information about the monitoring locations, including: names (or ID's), x- and y-coordinates, and covariates from a GIS to be used in the LUR. Recall that when we discussed the mesa.data structure in the above section we noted that latitude and longitude coordinates, as well as monitor type, are optional. This is where one would include coordinates and LUR information regarding locations where one does not have any observations but wants predictions.

Next, examine the $obs part of the raw data.

```
> mesa.data.raw$obs[1:6,1:5]
           60370002 60370016 60370030 60370031 60370113
1999-01-13 4.577684 4.131632       NA       NA 4.727882
1999-01-27 3.889091 3.543566       NA       NA 4.139332
1999-02-10 4.013020 3.632424       NA       NA 4.054051
1999-02-24 4.080691 3.842586       NA       NA 4.392799
1999-03-10 3.728085 3.396944       NA       NA 3.960577
1999-03-24 3.751913 3.626161       NA       NA 3.958741
```

In this example the observations are stored as a (number of time-points)-by-(number of locations) matrix with missing observations denote by NA, the row- and columnames identify the location and time point of each observation. Alternatively, one could have the observations as a data frame with three columns: date, location ID and observations; the mesa.data$obs data frame, described in subsubsection 4.1.4, follows this format. The format of mesa.data.raw$obs is more likely for data with only a few missing observations, and in the following we will demonstrate how to transform the observation matrix to a mesa.data$obs data frame.

The final element is a spatio-temporal covariate, i.e. the output from the Caline3QHC model (see subsection 2.3),

```
> mesa.data.raw$lax.conc.1500[1:6,1:5]
           60370002 60370016 60370030 60370031 60370113
1999-01-13   2.3188        0   8.0641   0.1467   2.9894
1999-01-27   1.8371        0   7.3568   0.2397   4.7381
1999-02-10   1.4886        0   6.3673   0.2463   4.3922
1999-02-24   2.5868        0   7.1783   0.1140   3.3456
1999-03-10   1.8996        0   6.3159   0.1537   3.8495
1999-03-24   2.0162        0   6.3277   0.1906   3.2170
```

This matrix contains spatio-temporal covariate values for all locations and times. Here, as in mesa.data.raw$X, we could include spatio-temporal covariate values for locations and times where we want to predict. Similar to the mesa.data.raw$obs matrix, the row- and columnames of the mesa.data.raw$lax.conc.1500 matrix contain the dates and location ID's of the spatio-temporal covariate.

The measurement locations, LUR information, observations and spatio-temporal covariates (optional) above constitute the basic raw data needed by the SpatioTemporal package. In the following Sections we will use this data to create the necessary components of the `mesa.data` object.

4.3.1 Creating `location`

We begin by creating the `mesa.data$location` data frame from the raw data with the following `R` commands.

```
##extract components from mesa.data.raw
##to create the location data.frame
location <- mesa.data.raw$X[,c("ID", "x", "y", "long",
                               "lat", "type")]
##ensure that it's a data.frame
location <- as.data.frame(location)
##ensure that ID and type are factors
location$ID <- factor(as.character(location$ID))
location$type <- factor(as.character(location$type))
```

This code creates the `$location` data frame of the `mesa.data` structure by extracting the relevant components from `mesa.data.raw$X`. Recall that `mesa.data$location` is a data frame containing location ID, x- and y-coordinates, as well as optional longitude and latitude coordinates and monitor type.

4.3.2 Creating `LUR`

The following code will create the `$LUR` element of the `mesa.data` structure:

```
##create a covariate LUR matrix
LUR <- mesa.data.raw$X
##ensure that it's a data.frame
LUR <- as.data.frame(LUR)
##take rownames as the IDs and drop location information
rownames(LUR) <- as.character(LUR$ID)
LUR <- LUR[,!(names(LUR) %in% c("ID", "x", "y", "long",
           "lat", "type"))]
```

The above code takes the `mesa.data.raw$X` matrix and drops all the variables except for the LUR's. It also makes the *location ID's the new rownames*, allowing the code to match elements in location with the corresponding covariates in LUR.

4.3.3 Creating obs

To create the $obs element of mesa.data we use the following R-code.

```
##create a data.frame of observations
T <- t(matrix(rownames(mesa.data.raw$obs),
              nrow=dim(mesa.data.raw$obs)[2],
              ncol=dim(mesa.data.raw$obs)[1],byrow=TRUE))
ID <- matrix(colnames(mesa.data.raw$obs),
             nrow=dim(mesa.data.raw$obs)[1],
             ncol=dim(mesa.data.raw$obs)[2],byrow=TRUE)
obs <- data.frame(obs=c(mesa.data.raw$obs), date=c(as.Date(T)),
                  ID=c(ID))
##drop unmonitored locations
obs <- obs[!is.na(obs$obs),,drop=FALSE]
##sort the locations (strictly not needed)
obs <- obs[order(obs$date,obs$ID),,drop=FALSE]
```

The above code begins by creating the two matrices T and ID. The matrices are both 280-by-25 and contain repetitions of the dates and locations ID's of the observations.

```
> T[1:3,1:3]
     [,1]         [,2]         [,3]
[1,] "1999-01-13" "1999-01-13" "1999-01-13"
[2,] "1999-01-27" "1999-01-27" "1999-01-27"
[3,] "1999-02-10" "1999-02-10" "1999-02-10"

> ID[1:3,1:3]
     [,1]       [,2]       [,3]
[1,] "60370002" "60370016" "60370030"
[2,] "60370002" "60370016" "60370030"
[3,] "60370002" "60370016" "60370030"
```

The code then creates a data frame by column-binding the observations (including the NA's) with the dates and location ID's. We then drop all the rows containing missing observations. The code also includes an optional line that sorts the observations by date and location.

4.3.4 Creating SpatioTemp

Next, we create the SpatioTemp array of the mesa.data structure.

```
##create a 3D-array for the spatio-temporal covariate
ST <- array(mesa.data.raw$lax.conc.1500, dim =
           c(dim(mesa.data.raw$lax.conc.1500),1))
##add names for time, location and spatio-temporal covariate
dimnames(ST) <- list(rownames(mesa.data.raw$lax.conc),
                     colnames(mesa.data.raw$lax.conc),
                     "lax.conc.1500")
```

The above code creates a *3-dimensional array*, with each 3rd dimension of the array containing a separate spatio-temporal covariate (i.e., to obtain the i^{th} spatio-temporal covariate, we would type `ST[, , i]`). In this case, there is only one spatio-temporal covariate, so the dimensions of the array ST for this example are 280 × 25 × 1.

4.3.5 Creating trend, and Combining it all

Finally, we create the trend element (see section 5) and combine everything we've created into the mesa.data structure, and we're done!

```
##compute the smooth trends
trend <- calc.smooth.trends(obs=obs$obs, date=obs$date,
                            ID=obs$ID, n.basis = 2)$svd

##Combining all the elements
mesa.data.alt <- list(location=location, LUR=LUR, trend=trend,
                      obs=obs, SpatioTemp=ST)
```

The function `calc.smooth.trends()` and the creation of the smooth trends are described in the following section.

5 Estimating the Smooth Temporal Trends

The first step in analysing the data is to determine, using cross-validation, how many smooth temporal trends we need to capture the seasonal variability.

5.1 Data Set-Up for Trend Estimation

In order to estimate the smooth trends, we need a data matrix that is (number of time-points)-by-(number of locations) in dimension. Here, the dimensions refer to time-points and locations where we have observations. Since our `mesa.data$obs` data frame has a single column for the dates, locations, and observations, we use the `create.data.matrix()` function to get the data into the format needed for trend estimation:

```
##extract a data matrix
> D <- create.data.matrix(mesa.data)
##and study the data
> dim(D)
[1] 280  25
> D[1:6,1:5]
           60370002 60370016 60370030 60370031 60370113
1999-01-13 4.577684 4.131632       NA       NA 4.727882
1999-01-27 3.889091 3.543566       NA       NA 4.139332
1999-02-10 4.013020 3.632424       NA       NA 4.054051
1999-02-24 4.080691 3.842586       NA       NA 4.392799
1999-03-10 3.728085 3.396944       NA       NA 3.960577
1999-03-24 3.751913 3.626161       NA       NA 3.958741
> colnames(D)
 [1] "60370002" "60370016" "60370030" "60370031" "60370113"
 [6] "60371002" "60371103" "60371201" "60371301" "60371601"
[11] "60371602" "60371701" "60372005" "60374002" "60375001"
[16] "60375005" "60590001" "60590007" "60591003" "60595001"
[21] "L001"     "L002"     "LC001"    "LC002"    "LC003"
```

`D` is now a 280 × 25 matrix where each column represents observations at one location, and the rows represents points in time. The matrix elements marked as `NA` indicate that there are no observation for those locations and times.

As a brief aside, note that we can also subset locations in the `create.data.matrix()` function. As an example, we create a data matrix for only the AQS monitoring locations:

```
##subset the data
> ID.subset <-mesa.data$location[mesa.data$location$type ==
                                 "AQS",]$ID
> D2 <- create.data.matrix(mesa.data, subset=ID.subset)
##and study the result
```

```
> dim(D2)
[1] 280  20
> colnames(D2)
 [1] "60370002" "60370016" "60370030" "60370031" "60370113"
 [6] "60371002" "60371103" "60371201" "60371301" "60371601"
[11] "60371602" "60371701" "60372005" "60374002" "60375001"
[16] "60375005" "60590001" "60590007" "60591003" "60595001"
```

Note that D2 is now a 280 × 20 matrix: we have created a data matrix for only the AQS locations, dropping the 5 Mesa FIXED locations.

5.2 Cross-Validated Trend Estimations

The displayed portion of the data matrix D shows what we have already seen in Figure 2: not every location has a complete time series, and the times at which observations are made are not consistent across location. When determining the best number of smooth temporal trends and computing these trends we need to address the missing data. Fuentes et al. (2006) outlines a procedure for computing smooth temporal trends from incomplete data matrices, a brief description is given in Sections 5.2.1–5.2.3 below. The procedure, including a cross validation to determine the best number of trends, is implemented in:

```
##Run leave one out cross-validation to find smooth trends
SVD.cv <- SVD.smooth.cv(D,1:5)
```

The function SVD.smooth.cv performs a sequence of procedures, the end goal of which is to assist the user in choosing the optimal number of smooth temporal trends that describe the seasonal variations in the data.
SVD.smooth.cv relies on two other functions: SVD.miss, that completes the data matrix, and SVD.smooth, that computes smooth temporal trends from the completed data matrix. In this example, we will evaluate 1 through 5 smooth trends in *each*, as specified by 1:5 in SVD.smooth.cv(D,1:5).

To clarify: It is important to understand that the following procedures (filling in the data matrix, finding the m smooth temporal trends, and running the leave one out cross-validation) are done for *each* of the values $m = (1, 2, \ldots, 5)$.

5.2.1 `SVD.miss`: Completing the Data Matrix

The function `SVD.miss` completes a data matrix by iterating over the following steps (See Fuentes et al., 2006, for details.):

Step 0. Regression through the origin of the columns of the data matrix `D` (with the NA's replaced by zeroes) on a length 280 vector, u_1, consisting of the mean concentration at each time point taken across all locations that are not `NA`. The missing observations are then replaced by the fitted values of that regression. For example, `D[i,j]` would be replaced by multiplying the i^{th} element of u_1 with the regression coefficient obtained by regressing the j^{th} column of `[`D`]` on u_1; missing values are ignored in the regression.

For this step to be well defined the data matrix must have *at least one* observation in each *row and column.*

Step 1. Compute the SVD of the new data matrix with the missing values imputed.

Step 2. Do regression of each column of the new data matrix on the first m orthogonal basis functions found in Step 1. The missing values are then replaced by the fitted values of this regression.

Step 3. Repeat from Step 1 until convergence; convergence being measured by the change in the imputed values between iterations.

5.2.2 `SVD.smooth`: Smoothing

To obtain m smooth temporal trends, `SVD.smooth` first standardises the data matrix so that each column is mean zero and variance one. The function then uses `SVD.miss` to complete the data matrix. Finally the function uses `smooth.spline` to smooth the m first orthogonal basis functions of the competed data matrix; as computed by `SVD.miss`.

5.2.3 `SVD.smooth.cv`: Cross-Validating

Finally, the function `SVD.smooth.cv` does leave-one-column-out (i.e. leave-one-site-out) cross-validation, and computes cross-validated statistics. These statistics are given in order to provide user guidance in choosing the number of temporal trends that are needed to describe the seasonal variations. The cross-validation procedure is carried out thus:

1. Leave out one column of D, call the reduced matrix $\widetilde{\mathtt{D}}$

2. Impute the missing values and obtain m smooth temporal trends for $\widetilde{\mathtt{D}}$ by calling SVD.smooth.

3. Do regression of the left-out column of D on the set of m smooth basis functions; use the residuals from the regression to compute cross-validation statistics

An important thing to keep in mind is that the value of m is fixed across the functions that are calling each other. That is, SVD.smooth.cv first does a cross-validation for m=1 (and hence calls SVD.smooth with $m = 1$, which calls SVD.miss with m=1) and calculates cross-validation statistics; then it does a new cross-validation for $m = 2$ (which in turn calls SVD.smooth with $m = 2$, etc.). From the output we can then compare the cross-validation statistics for each of the 5 sets of cross-validations (where m goes from 1 to 5.)

5.2.4 Evaluating the Cross-Validated Results

The output of SVD.smooth.cv consists of the cross-validated RMSE, R^2, and Bayesian Information Criteria (BIC). The number of basis functions that lead to low BIC and point to where the RMSE stops decreasing rapidly is what we are looking for.

To investigate this, we plot the results of this cross-validation, shown in.

```
##Plotting the smooth trends CV results, see Figure 3
par(mfcol=c(2,2),mar=c(4,4,.5,.5))
plot(SVD.cv$CV.stat$RMSE, type="l", ylab="RMSE", xlab="")
plot(SVD.cv$CV.stat$R2, type="l", ylab="R2", xlab="")
plot(SVD.cv$CV.stat$BIC, type="l", ylab="BIC", xlab="")
```

Figure 3 shows the cross-validated statistics. As would be expected in any regression scenario, increasing the number of basis functions increases the R^2 and decreases the RMSE, but the increase in R^2 and decrease in RMSE is by far the most dramatic going from 1 to 2 basis functions; the RMSE stops decreasing rapidly after 2 basis functions. The BIC is also lowest when 2 basis functions are used to model the temporal variations. This indicates that 2 basis functions would provide the most efficient description of the seasonal variability.

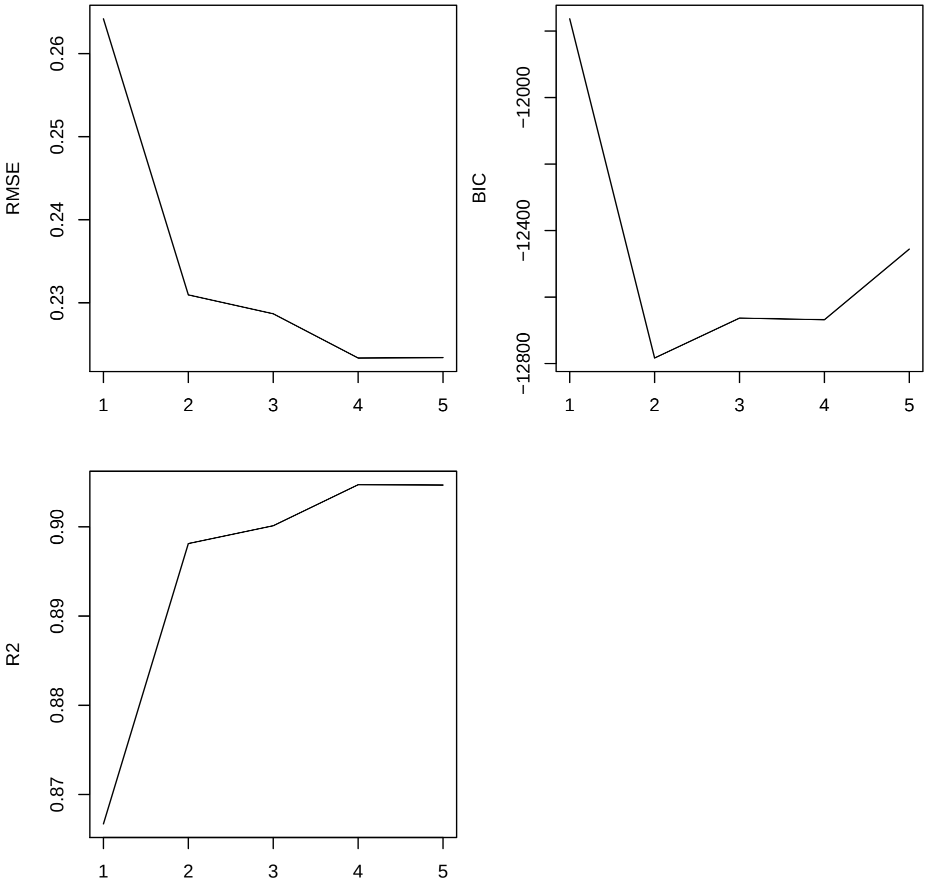

Figure 3: Cross-validation results for different numbers of smooth temporal trends

However just looking at overall statistics might be misleading. We can also do scatter plots of BIC for different numbers of trends at all the sites, as shown in Figure 4.

```
##Plotting BIC values, see Figure 4
par(mfcol=c(1,1),mar=c(4,4,.5,.5))
pairs(SVD.cv$BIC.all, pane =
      function(x,y){points(x,y);abline(0,1)})
```

Note from Figure 4 that as we increase the number of trends all sites don't behave equally. Some sites require many trends and some fewer. In particular, there are three sites whose seasonal trends appear very well described by using only one basis function.

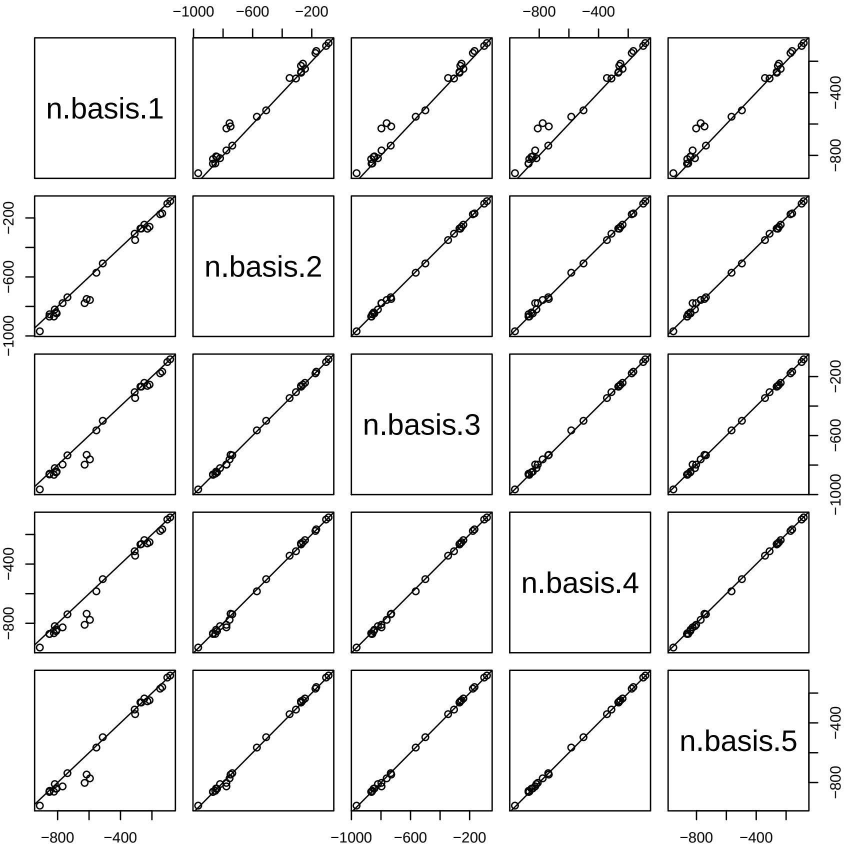

Figure 4: BIC at all sites for different numbers of temporal trends

5.3 Creating and Investigating the Trends

Based on the cross-validation statistics, using two smooth temporal basis functions appears to be appropriate. We now compute two smooth trends, and add them to the data structure.

```
##compute new temporal smooths
F <- calc.smooth.trends(mesa.data, n.basis=2)
##and add the new trends to the data structure
mesa.data$trend <- F$svd
```

Given smooth trends we fit the time series of observations to the trends at each site, and study the residuals (see Figure 5).

```
##plot the observations at two of the locations along
##with the fitted smooth trends, see Figure 5
par(mfrow=c(4,1),mar=c(2.5,2.5,2,.5))
plotMesaData(mesa.data, 5, type="obs")
plotMesaData(mesa.data, 18, type="obs")

##also plot the residuals
plotMesaData(mesa.data, 5, type="res")
plotMesaData(mesa.data, 18, type="res")
```

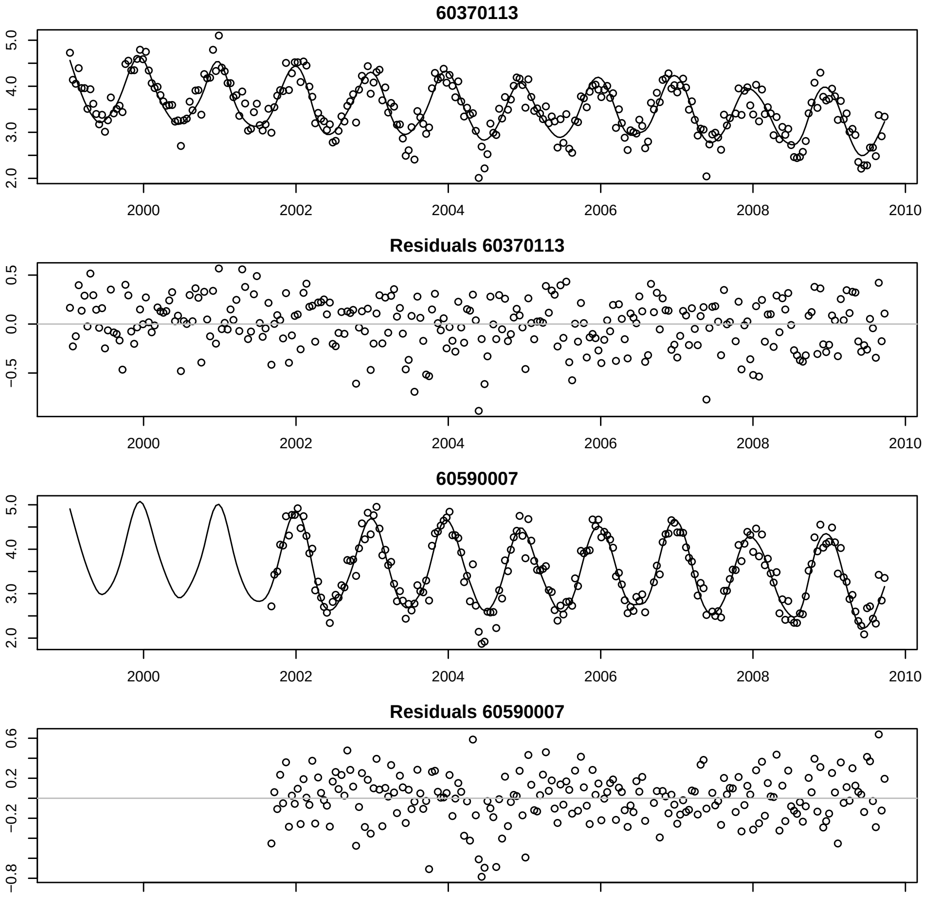

Figure 5: Smooth temporal trends and residuals for two locations

The function plotMesaData fits a linear regression of observations for a particular site on the smooth basis functions, and plots either fitted values, residuals or the auto-correlation function. From Figure 5 we can see that the fitted seasonal variability is different for the two sites (with greater seasonal variability in log $\mathrm{NO_x}$ for Site 60590007), and furthermore that for Site 60590007, the fit extends earlier in time than observations are made. Both of the fits elicit residuals that are centred around 0, indicating that the temporal trends do a good job of describing the temporal variability of the observed log $\mathrm{NO_x}$ concentrations.

Since we want the temporal trends to capture the temporal variability we also study the auto correlation function of the residuals to determine how much temporal dependence remains after fitting the temporal trends. If the temporal trends do a good job of capturing temporal variability, there should be little to no temporal correlation in the residuals. To investigate how well the trends have captured the temporal variability we study the auto-correlation plots in Figure 6.

```
##Autocorrelation of the residuals at four locations,
##see Figure 6
par(mfcol=c(2,2),mar=c(2.5,2.5,3,.5))
plotMesaData(mesa.data, 1, type="acf")
plotMesaData(mesa.data, 5, type="acf")
plotMesaData(mesa.data, 13, type="acf")
plotMesaData(mesa.data, 18, type="acf")
```

The plots in Figure 6 show us the auto-correlation functions at four different locations. Taken as a whole, these plots show us what we want to see: little to no correlation (rarely stronger than ±0.2) in the residuals at any time lag. These plots, together with the ones above it, give credence to the assumptions of the SpatioTemporal model: residual fields that have mean 0 and are uncorrelated over time. It appears that the temporal trends are aptly capturing the seasonal variability.

6 Empirical Estimation of the β–Fields

The advantage of the spatio-temporal model is the ability to allow the temporal fluctuation of $\mathrm{NO_x}$ to vary based on the geographic characteristics of various locations (2). This is done by allowing the coefficients of the temporal trends in the model to depend on land-use regression covariates (3). The coefficients of the temporal trends are what we refer to as the β–fields, which we will now estimate using an empirical approach (see Szpiro et al., 2010, for more details).

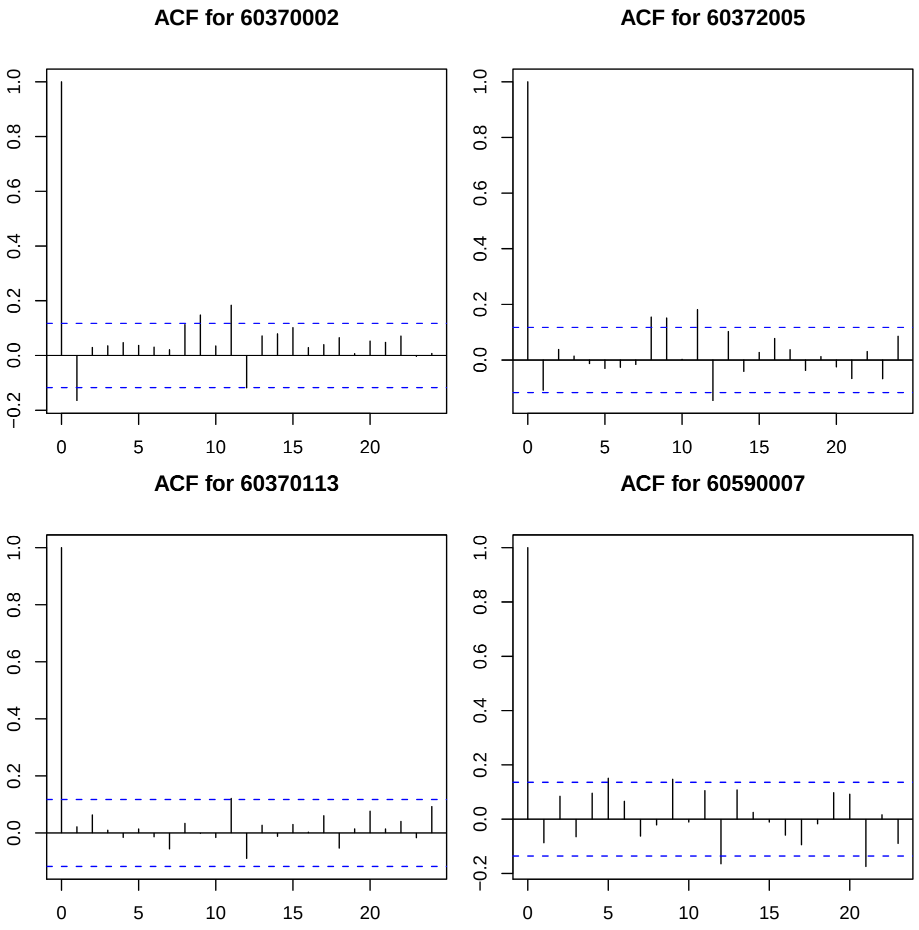

Figure 6: Auto-correlation functions for four locations

Recall D, which was the 280×25 data matrix of the $n = 25$ locations with the (up to) $T = 280$ observations over time. Given smooth temporal trends we fit each of the times series of observations to the smooth trends and extract the regression coefficients.

```
##create data matrix
D <- create.data.matrix(mesa.data)
beta <- matrix(NA,dim(D)[2], dim(mesa.data$trend)[2])
beta.std <- beta
##extact the temporal trends
F <- mesa.data$trend
##drop the date column
F$date <- NULL
```

```
##estimate the beta-coeficients at each location
for(i in 1:dim(D)[2]){
  tmp <- summary(lm(D[,i] ~ as.matrix(F)))
  beta[i,] <- tmp$coefficients[,1]
  beta.std[i,] <- tmp$coefficients[,2]
}

##Add names to the estimated betas
colnames(beta) <- c("const",colnames(F))
rownames(beta) <- colnames(D)
dimnames(beta.std) <- dimnames(beta)
```

Finally we study the estimated regression coefficients:

```
##examine the beta fields
> head(beta)
            beta0       beta1       beta2
60370002 3.698932 -0.20189383  0.25751993
60370016 3.372849 -0.07064596  0.24127792
60370030 4.206123 -0.48859878  0.15939586
60370031 3.719557 -0.64148029 -0.04067102
60370113 3.527936 -0.51028700  0.07050822
60371002 4.173345 -0.46945537  0.12826088
```

And the uncertainty in the estimation

```
> head(beta.std)
              const         V1         V2
60370002 0.01447987 0.01447860 0.01456190
60370016 0.01332344 0.01335100 0.01335100
60370030 0.04656093 0.03761209 0.04388509
60370031 0.07273316 0.04202223 0.06416097
60370113 0.01547441 0.01547949 0.01547938
60371002 0.01278047 0.01278508 0.01279324
```

These are the β–fields that will be modelled using geographic covariates in the full spatio-temporal model. Here, `beta` is a 25×3 matrix with $\beta_0(s), \beta_1(s)$, and $\beta_2(s)$ representing regression coefficients for the intercept and two temporal trends at each of the 25 locations.

The selection of geographic covariates could be done by comparing these fields to the available covariates. However, the variable selection is outside the scope of this tutorial (see Mercer et al., 2011, for some possible alternatives). For now we just keep the values and will (eventually) compare them to results from the full model.

7 Estimating the Model

We are now ready to fit the entire spatio-temporal model (7) to data.

Recall that the covariance matrices of each β_i-field is characterised by two parameters, ϕ_i and σ_i^2, where ϕ_i is the range and σ_i^2 is the partial sill (5); we are assuming a zero nugget for the β-fields. The covariance for the spatio-temporal residuals are characterised by three parameters, ϕ_ν, σ_ν^2, and τ_ν^2 (6); respectively denoting range, partial sill and nugget (Lindström et al., 2011, Sampson et al., 2009, Szpiro et al., 2010). These are the parameters that we need to estimate using maximum-likelihood. The regression parameters (4) are implicitly estimated since we will be using a profile likelihood formulation, see Lindström et al. (2011) for details.

7.1 Parameter Estimation

In the section 5 above we have defined two smooth temporal trends, and we have also seen that these temporal trends vary by location. Thus we want to define coefficients for these temporal trends that also vary by location. We have just the covariates we need to provide this flexibility in the coefficients: the land use regression covariates (LUR's) in `mesa.data$LUR`. We can take a look at the names of these covariates:

```
> names(mesa.data$LUR)
[1] "log10.m.to.a1"   "log10.m.to.a2"  "log10.m.to.a3"
[4] "log10.m.to.road" "km.to.coast"    "s2000.pop.div.10000"
```

and available spatio-temporal covariates:

```
> dimnames(mesa.data$SpatioTemp)[[3]]
[1] "lax.conc.1500"
```

Now we need to define which of these covariates will be used for our three temporal trend coefficients (including the intercept). We will create a model with three covariates for the temporal intercept, and one covariate for each of the two temporal trends:

```
mesa.data.model <- create.data.model(mesa.data, LUR =
    list(c("log10.m.to.a1", "s2000.pop.div.10000",
           "km.to.coast"), "km.to.coast", "km.to.coast"),
    ST.Ind=NULL)
```

Note that the LUR's we have specified match the names of the LUR's in `mesa.data$LUR`. This data structure, `mesa.data.model`, is what we will use to fit the model and estimate its parameters. Here we have defined the covariates that will allow for the flexibility in the temporal coefficients we want due to locational variation.

```
> names(mesa.data.model)
 [1] "obs"        "location"        "trend"    "F"
 [5] "X"          "dist"            "dates"    "nt"
 [9] "SpatioTemp" "SpatioTemp.all" "LUR.list" "ST.Ind"
```

Along with the contents of the `mesa.data` object described earlier, `mesa.data.model` contains several additional objects; the most interesting of these are `LUR.list` and `ST.Ind`

```
> mesa.data.model$LUR.list
[[1]]
[1] "log10.m.to.a1"  "s2000.pop.div.10000"  "km.to.coast"

[[2]]
[1] "km.to.coast"

[[3]]
[1] "km.to.coast"

> mesa.data.model$ST.Ind
NULL
```

In `mesa.data.model$ST.Ind` we see that the model includes *no* spatio-temporal covariates. Further, in `mesa.data.model$LUR.list` we see the covariates that will be used to model each of the β-fields. These are the same ones we defined when we created `mesa.data.model`.

Studying the elements of `mesa.data.model$X`:

```
> head(mesa.data.model$X$const)
         const log10.m.to.a1 s2000.pop.div.10000 km.to.coast
60370002     1      2.861509            1.733283   15.000000
```

```
60370016      1      3.461672                 1.645386   15.000000
60370030      1      2.561133                 6.192630   15.000000
60370031      1      3.111413                 2.088930    1.023311
60370113      1      2.762193                 7.143731    6.011075
60371002      1      2.760931                 4.766780   15.000000

> head(mesa.data.model$X$V1)
         const km.to.coast
60370002     1   15.000000
60370016     1   15.000000
60370030     1   15.000000
60370031     1    1.023311
60370113     1    6.011075
60371002     1   15.000000

> head(mesa.data.model$X$V2)
         const km.to.coast
60370002     1   15.000000
60370016     1   15.000000
60370030     1   15.000000
60370031     1    1.023311
60370113     1    6.011075
60371002     1   15.000000
```

We see that the function has created design matrices for each of the β-fields in mesa.data.model$X, this is the first step in allowing the temporal coefficients to vary by location.

One could also look at the matrix mesa.data.model$dist (not displayed), which is a 25×25 distance matrix containing Euclidean distances between the 25 locations. These distances were derived from the $x and $y coordinates of the mesa.data$location data structure. The distances in mesa.data.model$dist are used to create the exponential covariance matrices for the β– and ν–fields.

We now look at some other important aspects of the model:

```
##Some important dimensions of the model
> dim <- loglike.dim(mesa.data.model)
> print(dim)
$T
[1] 280

$m
```

```
[1] 3

$n
[1] 25

$p
[1] 4 2 2

$L
[1] 0

$nparam
[1] 17

$nparam.cov
[1] 9
```

The object `dim` gives us the important dimensions of our model: `T`, the number of temporal observations; `m`, the number of smooth temporal basis functions (including f_0, the temporal intercept); `n`, the number of locations; `p`, a vector that gives the number of covariates (including intercept) for each of the β-fields; `L`, the number of spatio-temporal covariates (there are none in this example); `nparam`, which gives the total number of parameters, including both the regression coefficients and the covariance parameters; and `nparam.cov` which is the number of covariance parameters: there are two for each of the β-fields (as discussed above) and three for the spatio-temporal residuals, for a total of 9 in this example.

In order to estimate the covariance parameters we need to specify starting points for the maximisation procedure:

```
##Setting up the initial parameter values for optimization
x.init <- cbind(rep(2,dim$nparam.cov),
                c(rep(c(1,-3),dim$m+1),-3))
```

Here the matrix `x.init` contains two starting values for the optimisation process in estimating the MLE's of the 9 covariance parameters. Note that these are starting values for only the optimisation of the covariance parameters; once those have been optimised, the maximum-likelihood estimate of the regression coefficients can be inferred using generalised least squares (see Lindström et al., 2011, for details). In general `x.init` should be a (`nparam.cov`)-by-(number of starting points) matrix, or just a vector of length `nparam.cov` vector if only one starting point is desired.

What parameters are we specifying the starting points for? We can verify this question using `loglike.var.names`:

```
##Names of the covariance parameters
> loglike.var.names(mesa.data.model,all=FALSE)
[1] "log.range.const" "log.sill.const" "log.range.V1"
[4] "log.sill.V1"     "log.range.V2"   "log.sill.V2"
[7] "log.range.nu"    "log.sill.nu"    "log.nugget.nu"
```

This gives the order of variables in `x.init` and also tells us which of the parameters that are logged. Specifying `all=FALSE` gives us only the covariance parameters.

```
##Add names to the initial values
> rownames(x.init) <- loglike.var.names(mesa.data.model,
+                                        all=FALSE)
> x.init
                [,1] [,2]
log.range.const    2    1
log.sill.const     2   -3
log.range.V1       2    1
log.sill.V1        2   -3
log.range.V2       2    1
log.sill.V2        2   -3
log.range.nu       2    1
log.sill.nu        2   -3
log.nugget.nu      2   -3
```

We are now ready to estimate the model parameters!

WARNING: The following steps are time-consuming.

```
##Estimate parameters
>  par.est <- fit.mesa.model(x.init, mesa.data.model, type="p",
+      hessian.all=TRUE, control=list(trace=3,maxit=1000))
In 'fit.mesa.model': Optimisation using starting value 1 of 2
N = 9, M = 5 machine precision = 2.22045e-16
At X0, 0 variables are exactly at the bounds
At iterate     0  f=       6266.3  |proj g|=          17
At iterate    10  f =      -5723.4  |proj g|=       19.483
At iterate    20  f =      -5727.4  |proj g|=       4.1733
At iterate    30  f =      -5731.4  |proj g|=        4.989
At iterate    40  f =      -5732.2  |proj g|=      0.32831
Bad direction in the line search;
   refresh the lbfgs memory and restart the iteration.

iterations 48
function evaluations 91
segments explored during Cauchy searches 52
BFGS updates skipped 0
active bounds at final generalized Cauchy point 0
norm of the final projected gradient 0.0678904
final function value -5732.2

F = -5732.2
l(0) > u(0).  No feasible solutionfinal  value -5732.201905
converged
In 'fit.mesa.model': Optimisation using starting value 2 of 2
N = 9, M = 5 machine precision = 2.22045e-16
At X0, 0 variables are exactly at the bounds
At iterate     0  f=      -3987.4  |proj g|=          14
At iterate    10  f =      -5727.4  |proj g|=       7.6344
At iterate    20  f =      -5731.6  |proj g|=       7.1354
At iterate    30  f =      -5732.1  |proj g|=       1.0735
At iterate    40  f =      -5732.2  |proj g|=      0.89313
At iterate    50  f =      -5732.2  |proj g|=      0.21688

iterations 58
function evaluations 72
segments explored during Cauchy searches 61
BFGS updates skipped 0
active bounds at final generalized Cauchy point 0
norm of the final projected gradient 0.3795
final function value -5732.2

F = -5732.2
final  value -5732.201567
converged
```

ALTERNATIVE: Load pre-computed results.

```
##Get the precomputed optimisation results instead.
par.est <- mesa.data.res$par.est
```

End of alternative

From the output, which is mainly due to the internal function `optim`, we see that the two optimisation consumed 91 and 72 function evaluations each and ended with the same value, −5732.2. The exact behaviour, including amount of progress information, of `optim` is controlled by the pass-through argument `control = list(trace=3, maxit=1000)`.

The log-likelihood function called by `fit.mesa.model` is included in the package as `loglike`, with `loglike.grad` and `loglike.hessian` computing the finite difference gradient and hessian of the log-likelihood functions. In case of trouble with the optimisation the user is recommended to study the behaviour of the log-likelihood.

Here we just verify that the log-likelihood value given parameters from the optimisation actually equals the maximum reported from the optimisation.

```
> loglike(par.est$res.best$par, mesa.data.model)
[1] 5732.202
> par.est$res.best$value
[1] 5732.202
```

7.2 Evaluating the Results

The first step in evaluating the optimisation results is to study the message included in the output from `fit.mesa.model`:

```
##Optimisation status message
> par.est$message
[1] "Optimisation: 2 converged, 0 not converged, 0 failed.
     Best result found for starting point 1 (optimisation has
     converged). Best converged result for starting point 1."
```

The message indicates that of our 2 starting points both converged, and the best overall result was found for the first starting value.

The function `fit.mesa.model` determines convergence for a given optimisation by studying the `convergence` field in the output from `optim`, with 0 indicating a successful completion; followed by an evaluation of the eigenvalues of the Hessian (the 2$^{\text{nd}}$ derivative of the log-likelihood) to determine if the matrix is negative definite; indicating that the optimisation has found a (local) maximum.

Studying `par.est`

```
> names(par.est)
[1] "res.best" "res.all"  "message"
```

we see that `fit.mesa.model`, in addition to the message above, returns the results from all the optimisations and the best possible result. Here `res.all` is a list with the optimisation results for each starting point, and `res.best` contains the "best" optimisation results.

Examining the optimisation results

```
> names(par.est$res.best)
 [1] "par"     "value" "counts"   "convergence" "message"
 [6] "hessian" "conv"  "par.init" "par.all"     "hessian.all"

> names(par.est$res.all[[1]])
 [1] "par"     "value" "counts"   "convergence" "message"
 [6] "hessian" "conv"  "par.init" "par.all"

> names(par.est$res.all[[2]])
 [1] "par"     "value" "counts"   "convergence" "message"
 [6] "hessian" "conv"  "par.init" "par.all"
```

we see that the results include several different fields —

`par` The estimate parameters.

`value` The value of the log-likelihood.

`counts` The number of function evaluations.

`convergence` **and** `message` Convergence information from `optim`.

`conv` An indicator convergence that combines `convergence` with a check if the Hessian is negative definite

`hessian` The Hessian of the profile log-likelihood.

`par.init` The starting point for the optimisation.

`par.all` All the parameters of the model, computed using generalised least squares (See Lindström et al., 2011, for details.).

`hessian.all` The Hessian of the full log-likelihood, this is *only* computed for the best starting point, `par.est$res.best`.

Given all the results we can compare the estimated parameters and final values for the two different starting values.

```
> cbind(par.est$res.all[[1]]$par.all,
+       par.est$res.all[[2]]$par.all)
                                     [,1]        [,2]
alpha.const.const                 3.76831400  3.76908005
alpha.const.log10.m.to.a1        -0.21088919 -0.21115277
alpha.const.s2000.pop.div.10000   0.04060766  0.04056836
alpha.const.km.to.coast           0.03780148  0.03784215
alpha.V1.const                   -0.74459989 -0.74299999
alpha.V1.km.to.coast              0.01752189  0.01739811
alpha.V2.const                   -0.14002081 -0.14023731
alpha.V2.km.to.coast              0.01609303  0.01610390
log.range.const                   2.27289032  2.26417580
log.sill.const                   -2.74896409 -2.75346065
log.range.V1                      2.89745785  2.92480706
log.sill.V1                      -3.53291852 -3.51741390
log.range.V2                      1.54690291  1.54125633
log.sill.V2                      -4.62574529 -4.62045553
log.range.nu                      4.41073138  4.41007669
log.sill.nu                      -3.22895866 -3.22964405
log.nugget.nu                    -4.30031734 -4.29991896

> cbind(par.est$res.all[[1]]$value,
+       par.est$res.all[[2]]$value)
         [,1]     [,2]
[1,] 5732.202 5732.202
```

The parameters are similar but not identical, with the biggest difference being for `log.range.V1`. The differences have to do with where and how the numerical optimisation stopped/converged. Due to the few locations (only 25) the log-likelihood is flat, implying that even with some variability in the parameter values we will still obtain *very* similar log-likelihood values.

The flat log-likelihood implies that some parameter estimates will be rather uncertain. Extracting the estimated parameters and computing the parameter uncertainties

```
##extract the estimated parameters
x <- par.est$res.best$par.all
```

```
##and approximate uncertainties from the hessian
x.sd <- sqrt(diag(-solve(par.est$res.best$hessian.all)))
names(x.sd) <- names(x)
```

we can now plot the estimated parameters along with approximate confidence intervals, see Figure 7.

```
##Plot the estimated parameters with uncertainties, see Figure 7
par(mfrow=c(1,1),mar=c(13.5,2.5,.5,.5))
plotCI(x,uiw=1.96*x.sd,ylab="",xlab="",xaxt="n")
points(par.est$res.all[[2]]$par.all, col=2, pch=4)
abline(h=0, col="grey")
axis(1,1:length(x),names(x),las=2)
```

In the above `x` contains the 17 estimated parameters including the spatial coefficients for the β-fields, the covariance parameters for the β-fields, and the covariance parameters for the spatial-temporal residual fields, ν. `par.sd` contains the approximate standard errors of those estimated parameters, computed using the Hessian (i.e. the observed information matrix) and assuming standard asymptotic properties for ML-estimators (Chap. 10, Casella and Berger, 2002).

From Figure 7 we note that the confidence intervals of the covariance parameters for the spatial-temporal residual field are much smaller than those of the covariance parameters that characterise the β-fields.

The wide confidence intervals for the β-field covariance parameters is because the effective number of "observations" that go into estimating the β-field covariance parameters is only 25, while the entire contingent of observations (4577 in this data set) can be used to estimate the covariance parameters of the spatial-temporal residual fields. Another way of seeing this is that we have *only one replicate* of each β-field — given by the regression of observations on the smooth-temporal basis functions — but T replicates of the residual field, *one for each timepoint* — given by the residuals from the regression.

Either way, the larger sample size for the residual field is making the standard error for those covariance parameters smaller, leading to tighter confidence intervals.

7.3 Predictions

Having estimated the model parameters we use `cond.expectation` to compute the conditional expectations of different parts of the model.

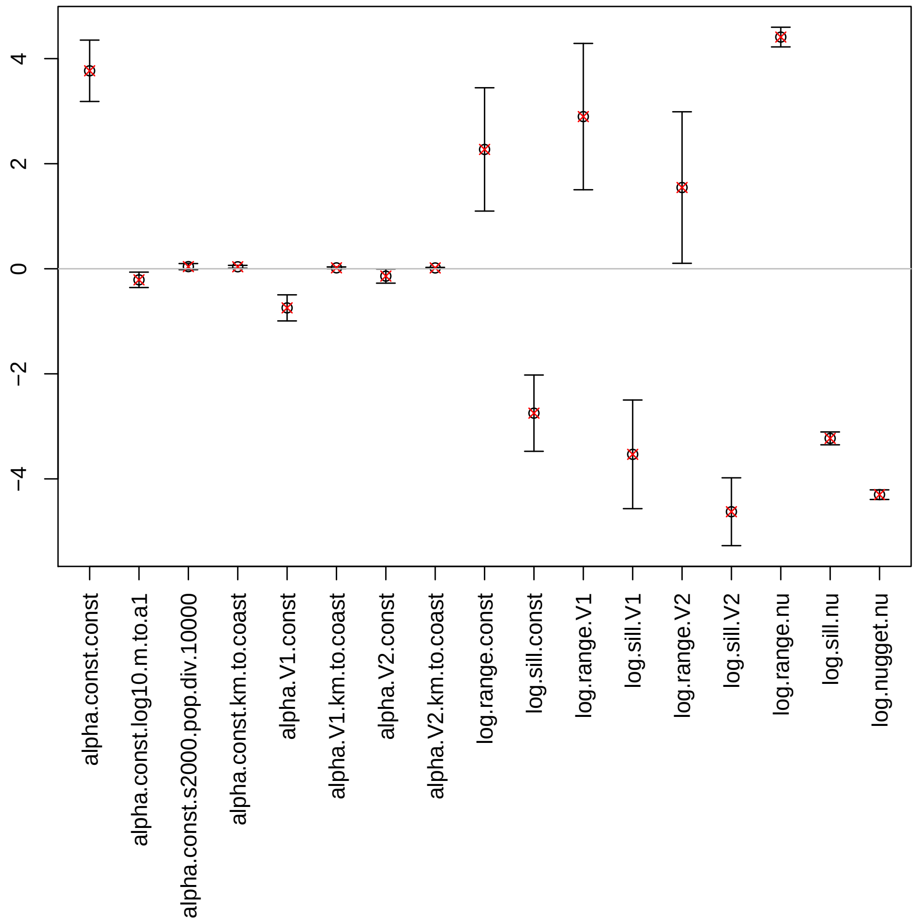

Figure 7: Estimated parameters and their 95% confidence intervals, red crosses mark the parameter values from the second starting point.

```
##compute conditional expectations (takes roughly 1 min)
EX <- cond.expectation(par.est$res.best$par, mesa.data.model,
                       compute.beta = TRUE)
```

The results from `cond.expectation` contains the following elements

```
> names(EX)
[1] "pars"       "EX.beta.mu" "EX.beta"  "VX.beta" "EX.mu"
[6] "EX.mu.beta" "EX"         "VX"       "VX.full" "I"
```

The most important components of these results are the estimated β-fields and their variances (`EX.beta` and `VX.beta`); as well as the conditional expectations and variances at all the 280 × 25 space-time locations (`EX` and `VX`).

All the components of `EX` are compute conditional on the estimated parameters and observed data; the components are:

`pars` The regression parameters in (4), computed using generalised least squares (see Lindström et al., 2011, for details).

`EX.beta` The expected latent β-fields.

`EX.beta.mu` The mean component of the latent β-fields, i.e. the $X_i\alpha_i$ part of (3).

`VX.beta` The conditional variance of the latent β-fields in `EX.beta`.

`EX` The expected spatio-temporal process (1), or
$\mathsf{E}(y(s,t)|\Psi, \text{observations})$.

`EX.mu` The regression component of the spatio-temporal process,

$$\mu(s,t) = \sum_{l=1}^{L} \gamma_l \mathcal{M}_l(s,t) + \sum_{i=1}^{m} X_i\alpha_i f_i(t).$$

Note that this differs from (2).

`EX.mu.beta` The mean part (2) of the spatio-temporal process (1).

`VX` The conditional variance of the spatio-temporal process in `EX`.

`VX.full` A list containing the temporal covariance matrix for each reconstructed time-series. This is needed to correctly compute confidence intervalls for long term average pollution at each location.

`I` An index vector that can be used to extract the observed spatio-temporal locations from `EX`, `EX.mu`, and/or `EX.mu.beta`.

The decomposition of the predictions could potentially be used for model evaluation.

First we compare the β-fields computed by fitting each of the times series of observations to the smooth trends, with the β-fields obtained from the full model. Recall that the simple, or empirical, estimate of the β-fields was computed in section 6 above with the results are contained in the `beta`.

```
##Plot that compares two estimated beta-fields, see Figure 8
```

```
par(mfcol=c(1,1), mar=c(4.5,4.5,2,.5), pty="s")
plotCI(x=beta[,1], y=EX$EX.beta[,1],
       uiw=1.96*beta.std[,1], err="x",
       main="Intercept", pch=NA, sfrac=0.005,
       xlab="Empirical estimate",
       ylab="Spatio-Temporal Model")
plotCI(x=beta[,1],y=EX$EX.beta[,1],
       uiw=1.96*sqrt(EX$VX.beta[,1]),
       add=TRUE, pch=NA, sfrac=0.005)
abline(0,1,col="grey")
```

The above code compares the two estimates of the β-field for $f_1(t) \equiv 1$ and produces Figure 8. We can of course also look at the β-fields for the two smooth temporal trend, f_2 and f_3.

```
##Plot that compares the other beta-fields, see Figure 9
par(mfcol=c(1,2), mar=c(4.5,4.5,2,.5), pty="s")
for(i in 2:3){
  plotCI(x=beta[,i], y=EX$EX.beta[,i],
         uiw=1.96*beta.std[,i], err="x",
         main=colnames(beta)[i], pch=NA, sfrac=0.005,
         xlab="Empirical estimate",
         ylab="Spatio-Temporal Model")
  plotCI(x=beta[,i],y=EX$EX.beta[,i],
         uiw=1.96*sqrt(EX$VX.beta[,i]),
         add=TRUE, pch=NA, sfrac=0.005)
  abline(0,1,col="grey")
}
```

We can see from Figure 8 and 9 that the two ways of computing the β-fields lead to very comparable results. The largest discrepancies lie with the coefficient for the second temporal trend, where it appears the coefficients calculated via conditional expectation are larger than those calculated by fitting the time series to the temporal trend. However, the uncertainty in these coefficients is large.

In addition to predictions of the β-fields, `cond.expectation` also computes the conditional expectation at all the 280 × 25 space-time locations.

```
##plot predictions and observations for 4 locations, see Figure 10
par(mfrow=c(4,1),mar=c(2.5,2.5,2,.5))
plotPrediction(EX,  1, mesa.data.model)
lines(as.Date(rownames(EX$EX.mu)),EX$EX.mu[,1],col=3)
```

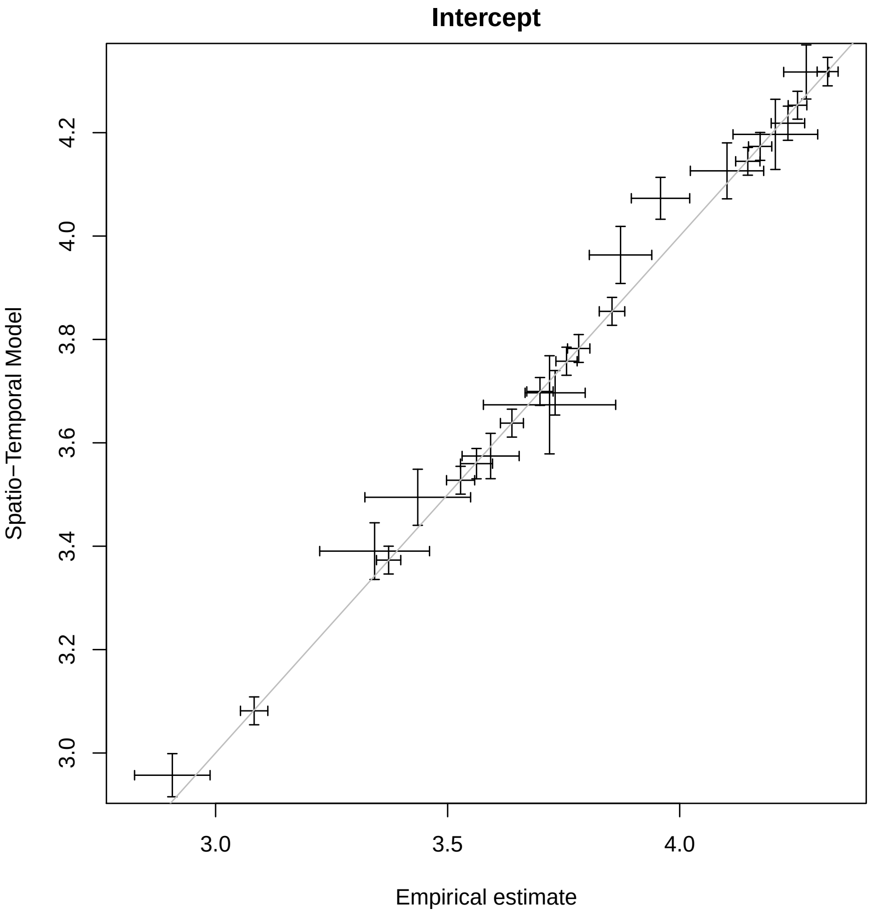

Figure 8: Comparing the two estimates of the β–field for the constant temporal trend.

```
lines(as.Date(rownames(EX$EX.mu.beta)),
      EX$EX.mu.beta[,1],col=4)
plotPrediction(EX, 10, mesa.data.model)
lines(as.Date(rownames(EX$EX.mu)),EX$EX.mu[,10],col=3)
lines(as.Date(rownames(EX$EX.mu.beta)),
      EX$EX.mu.beta[,10],col=4)
plotPrediction(EX, 17, mesa.data.model)
lines(as.Date(rownames(EX$EX.mu)),EX$EX.mu[,17],col=3)
lines(as.Date(rownames(EX$EX.mu.beta)),
      EX$EX.mu.beta[,17],col=4)
plotPrediction(EX, 22, mesa.data.model)
lines(as.Date(rownames(EX$EX.mu)),EX$EX.mu[,22],col=3)
```

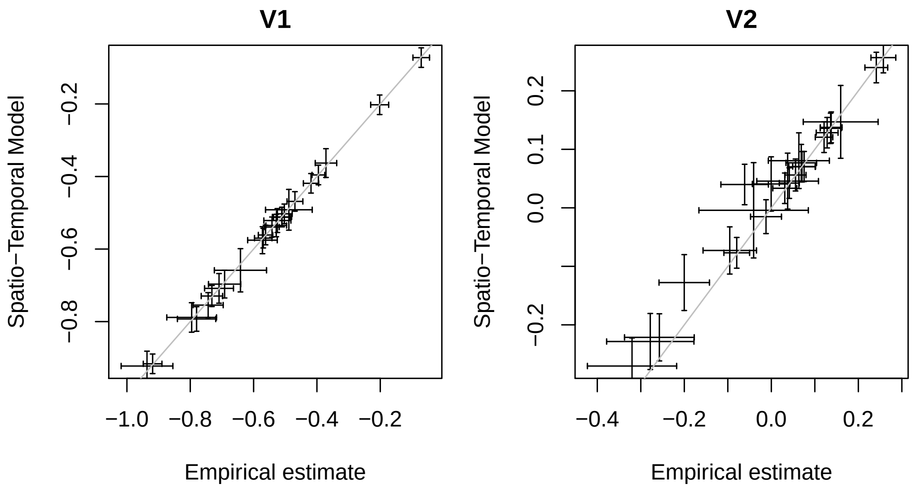

Figure 9: Comparing the two estimates of the β–field for the two smooth temporal trends.

```
lines(as.Date(rownames(EX$EX.mu.beta)),
      EX$EX.mu.beta[,22],col=4)
```

Plotting these predictions along with 95% confidence intervals, the components of the predictions, and the observations at 4 different locations indicates that the predictions capture the seasonal variations in the data, see Figure 10. Since the predictions are conditional on the observations (and estimated parameters) predictions at the observed locations of course coincides with the observations. Adding the components of the predictions that are due to only the regression (green) and both regression and β-fields (blue) allows us to investigate how the different parts of the model capture observations.

8 Cross-validation

As a last step in the tutorial we will study a cross-validation (CV) example. The first step is to define 10 CV groups:

```
##create the CV structure defining 10 different CV-groups
> Ind.cv <- createCV(mesa.data.model, groups=10, min.dist=.1)
> head(Ind.cv)
      [,1]  [,2]  [,3]  [,4]  [,5]  [,6]  [,7]  [,8]  [,9] [,10]
[1,]  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
```

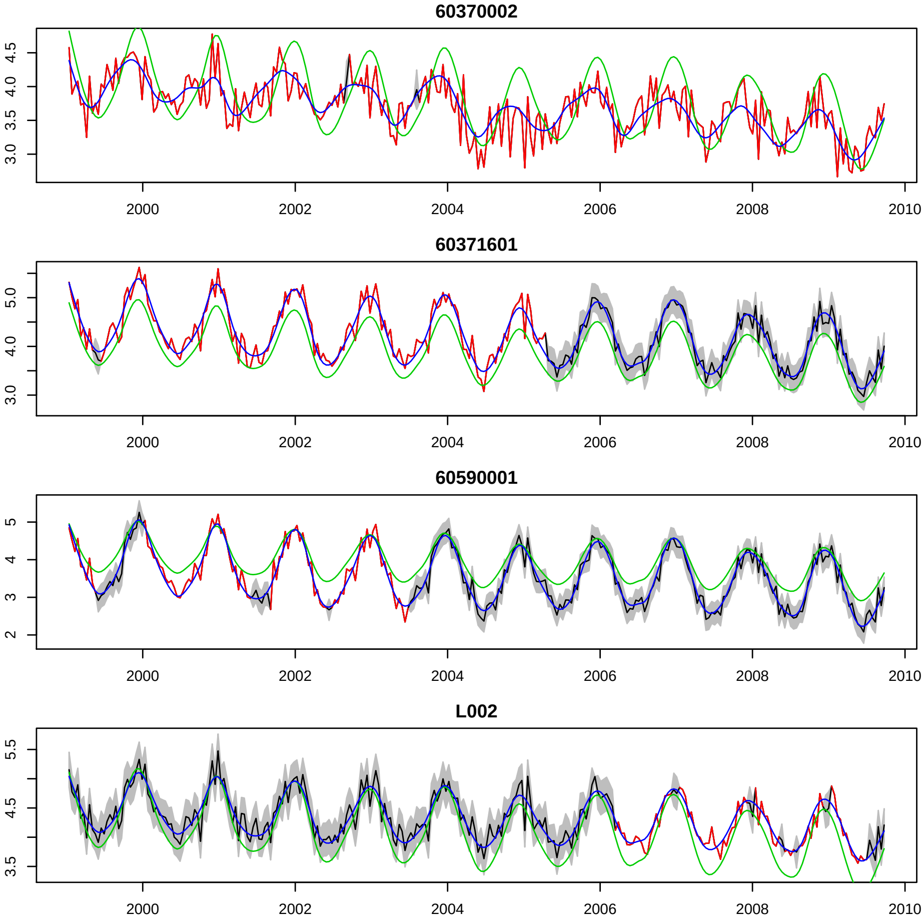

Figure 10: The predicted and observed data for 4 of the 25 locations. The red-lines denote observations, the black line and grey shading give predictions and 95% confidence intervals at unobserved time-points. The green and blue give the contribution to the predictions from the regression and regression + β-fields respectively.

```
[2,] FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[3,] FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE
[4,] FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[5,] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE
[6,] FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE
```

Here `Ind.cv` is a 4577 × 10 matrix where each column defines a CV-group. The columns contain logical values indicating which of the observations to leave out. For each column, we are going to use our model to predict at the observations marked by `TRUE` using all of the observations marked `FALSE`.

Once we have done this for each CV-group/column we can compare our predictions to the truth and calculate cross-validated statistics such as RMSE and R^2.

However first we will take a closer look at the CV-groupings.

```
##number of observations in each CV-group
> colSums(Ind.cv)
 [1] 438 389 811 556 546 165 228 487 160 797
```

We see that the number of observations left out of each group is rather uneven; the main goal of `createCV` is to create CV-groups such that the groups contain roughly the same *number of locations* ignoring the number of observations at each location. If there are large differences in the number of observations at different locations one could use the `subset` option to create different CV-groupings for different types of locations, possibly combining the resulting CV-groups by `Ind.cv.final = Ind.cv1 | Ind.cv2`.

If we instead look at which sites that will be excluded from which CV-group:

```
##Which sites belong to which groups?
> ID.cv <- lapply(apply(Ind.cv,2,list),function(x)
+                 unique(mesa.data.model$obs$ID[x[[1]]]))
> ID.cv
[[1]]
[1] "60370002" "L002"     "LC001"

[[2]]
[1] "60371002" "60370030" "LC003"

[[3]]
[1] "60370016" "60371301" "60374002"

[[4]]
[1] "60371201" "60372005"

[[5]]
[1] "60591003" "60595001"

[[6]]
[1] "60370031" "60375005"
```

```
[[7]]
[1] "60375001" "60590001"

[[8]]
[1] "60370113" "60590007"

[[9]]
[1] "LC002"    "60371602"

[[10]]
[1] "60371103" "60371601" "60371701" "L001"
```

We see that the groups are a lot more even. The four sites in the 10th group is due to the fact that 60371701 and L001 are colocated.

```
##Distances between the sites in the 10th CV-group
> mesa.data.model$dist[ID.cv[[10]],ID.cv[[10]]]
            60371103 60371601 60371701        L001
60371103  0.00000000 16.36527 43.75141  0.08363892
60371601 16.36527030  0.00000 29.06447 16.44669372
60371701 43.75141252 29.06447  0.00000 43.83429713
L001      0.08363892 16.44669 43.83430  0.00000000
```

By studying the distance between the sites in the 10th group we see that the 60371701 and L001 are only 0.084 km apart, which is less than the `min.dist=.1` specified in the `createCV`-call above. This causes `createCV` to lump the two sites together, treating them as "one" site when creating the CV-grouping.

Instead of creating a list with which site(s) get dropped in each CV-group is might be more useful with a vector that, for each site, indicates which CV-group it belongs to.

```
##Find out which location belongs to which cv group
> I.col <- apply(sapply(ID.cv,
+     function(x) mesa.data.model$location$ID %in% x),1,
+     function(x) if(sum(x)==1) which(x) else 0)
> names(I.col) <- mesa.data.model$location$ID
> I.col
60370002 60370016 60370030 60370031 60370113 60371002 60371103
       1        3        2        6        8        2       10
60371201 60371301 60371601 60371602 60371701 60372005 60374002
```

```
       4        3       10        9       10        4        3
60375001 60375005 60590001 60590007 60591003 60595001     L001
       7        6        7        8        5        5       10
    L002    LC001    LC002    LC003
       1        1        9        2
```

Using this vector we can plot the sites on a map, colour-coded by which CV-group they belong to, see Figure 11.

```
##Plot the locations, colour coded by CV-grouping, see Figure 11
par(mfrow=c(1,1))
plot(mesa.data$location$long,mesa.data$location$lat,
     pch=23+floor(I.col/max(I.col)+.5), bg=I.col,
     xlab="Longitude",ylab="Latitude")

##Add the map of LA
map("county","california",col="#FFFF0055",fill=TRUE,add=TRUE)
```

Having created the CV-grouping we need to estimate the parameters for each of the CV-groups and then predict the left out observations given the estimated parameters. The estimation and prediction is described in subsection 8.1 and 8.2 below.

8.1 Cross-validated Estimation

Parameter estimation for each of the CV-groups is done by the `estimateCV` function, which calls `fit.mesa.model`. The inputs to `estimateCV` are similar to those of `fit.mesa.model`. As described in subsection 7.1 the estimation function require at least a `mesa.data.model` and a matrix (or vector) of initial values, in addition to these `estimateCV` also requires a matrix describing the CV-grouping, e.g. `Ind.cv`.

WARNING: The following steps are time-consuming.

```
##estimate different parameters for each CV-group
dim <- loglike.dim(mesa.data.model)
x.init <- cbind(rep(2,dim$nparam.cov),
                c(rep(c(1,-3),dim$m+1),-3))
par.est.cv <- estimateCV(x.init, mesa.data.model, Ind.cv)
```

ALTERNATIVE: Load pre-computed results.

```
##Get the precomputed CV-estimation results instead.
par.est.cv <- mesa.data.res$par.est.cv
```

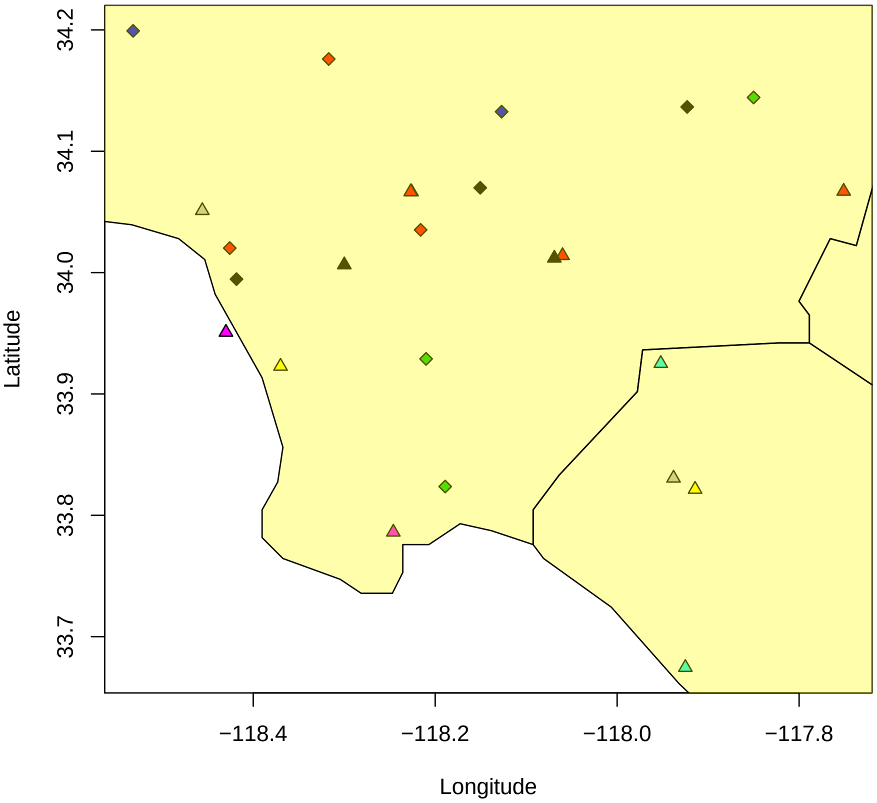

Figure 11: Location of monitors in the Los Angeles area. The different cross-validation groups are indicated by colour and shape of the points, i.e. all points of the same colour and shape belong to the same cross-validation group.

End of alternative

Since the parameter estimation for 10 CV-groups using two initial values takes some considerable time the results have been pre-computed.

We first study the results of the estimation.

```
#lets examine the results
> names(par.est.cv)
[1] "par"     "res.all"
```

Here `par` is a matrix containing the estimated parameters for each of the CV-groups,

```
> par.est.cv$par
                      [,1]      [,2]      [,3]      [,4]      [,5]
log.range.const  1.827772  2.343013  0.853828  2.335368  2.094403
log.sill.const  -2.653389 -2.782688 -2.811791 -2.682343 -2.796057
log.range.V1     3.036171  2.748023  2.552579  3.039957  2.689855
log.sill.V1     -3.494985 -3.491094 -3.797869 -3.377405 -3.655132
log.range.V2    -1.806757  1.897147 -2.618593  1.536771  1.563515
log.sill.V2     -4.809735 -4.627417 -4.696091 -4.546989 -4.557741
log.range.nu     4.578942  4.506398  4.705063  4.488070  4.220442
log.sill.nu     -3.299564 -3.262670 -3.230799 -3.214886 -3.254663
log.nugget.nu   -4.221845 -4.206128 -4.220209 -4.216576 -4.338722
                      [,6]      [,7]      [,8]      [,9]     [,10]
log.range.const  1.051494  1.335265  2.264926  1.673407  3.011853
log.sill.const  -2.894746 -2.735022 -2.771816 -2.665607 -2.696889
log.range.V1     2.834853  2.809474  3.197457  2.811640  2.922013
log.sill.V1     -3.548835 -3.511821 -3.508199 -3.596125 -3.409934
log.range.V2    -1.973050 -1.621618  2.305343 -1.661328  1.679843
log.sill.V2     -4.718497 -4.562417 -4.841537 -4.781818 -4.556341
log.range.nu     4.371704  4.424812  4.312572  4.409028  4.226999
log.sill.nu     -3.198788 -3.263056 -3.211409 -3.231683 -3.175562
log.nugget.nu   -4.524629 -4.260814 -4.443111 -4.283517 -4.276639
```

and `res.all` is a list storing the `res.best` field from the output of `fit.mesa.model` for each of the CV-groups.

```
> str(par.est.cv$res.all[[1]])
List of 9
 $ par        : Named num [1:9] 1.83 -2.65 3.04 -3.49 -1.81 ...
  ..- attr(*, "names")= chr [1:9] "log.range.const" ...
 $ value      : num 5170
 $ counts     : Named int [1:2] 103 103
  ..- attr(*, "names")= chr [1:2] "function" "gradient"
 $ convergence: int 0
 $ message    : chr "CONVERGENCE: REL_REDUCTION_OF_F ..."
 $ hessian    : num [1:9, 1:9] -2.2384 1.4527 -0.0766 ...
 $ conv       : logi TRUE
 $ par.init   : Named num [1:9] 2 2 2 2 2 2 2 2 2
  ..- attr(*, "names")= chr [1:9] "log.range.const" ...
 $ par.all    : Named num [1:17] 3.8571 -0.2426 0.033 ...
  ..- attr(*, "names")= chr [1:17] "alpha.const.const" ...
```

Studying the `conv`-field in `res.all`

```
> sapply(par.est.cv$res.all,function(x) x$conv)
```

```
 [1] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
```

we see that all optimisations have converged for all CV-groups. The field `par.init` gives information about which initial value gave the best optimisation result, for each CV-group

```
> sapply(par.est.cv$res.all,function(x) x$par.init)
               [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
log.range.const   2    2    2    2    2    1    1    1    1     1
log.sill.const    2    2    2    2    2   -3   -3   -3   -3    -3
log.range.V1      2    2    2    2    2    1    1    1    1     1
log.sill.V1       2    2    2    2    2   -3   -3   -3   -3    -3
log.range.V2      2    2    2    2    2    1    1    1    1     1
log.sill.V2       2    2    2    2    2   -3   -3   -3   -3    -3
log.range.nu      2    2    2    2    2    1    1    1    1     1
log.sill.nu       2    2    2    2    2   -3   -3   -3   -3    -3
log.nugget.nu     2    2    2    2    2   -3   -3   -3   -3    -3
```

and we see that different initial values worked best for different CV-groups.

Finally we can study the difference in the estimated parameters for the different CV-groups using a box-plot.

```
##Boxplot of the different estimates from the CV, see Figure 12
par(mfrow=c(1,1), mar=c(7,2.5,2,.5), las=2)
boxplot(t(par.est.cv$par))
points(par.est$res.best$par, pch=4, col=2)
```

The result in Figure 12, shows some considerable spread in the estimated variables, which can be compared to the parameter estimates based on all the data in Figure 7.

8.2 Cross-Validated Prediction

Once the parameters have been estimated, `predictCV()` does column-by-column predictions of the left-out observations denoted by `TRUE` in the `Ind.cv` data set.

```
##Do cross-validated predictions using the newly
##estimated paremeters (takes roughly 1 minute)
> pred.cv <- predictCV(par.est.cv$par, mesa.data.model,
+                      Ind.cv, silent = FALSE)
[1] "Predicting cv-set 1/10"
```

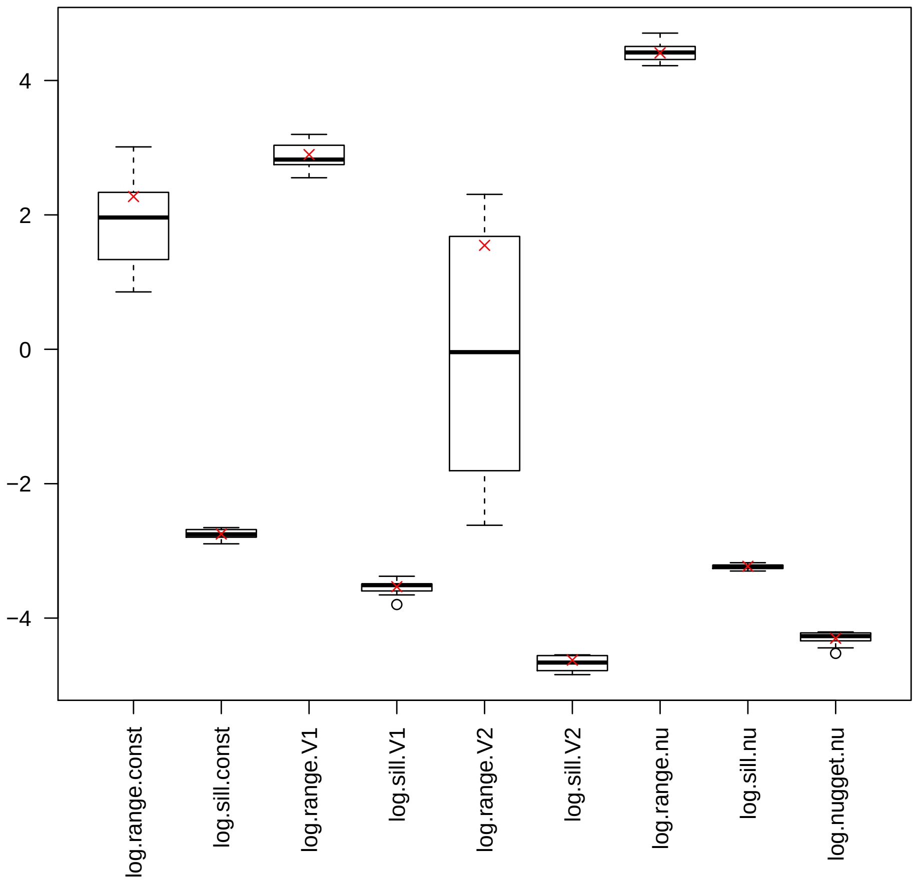

Figure 12: The estimated parameters for the 10 cross-validation groups, the red cross marks the parameter estimates obtained from the entire data in Figure 7, subsection 7.1.

```
[1] "Predicting cv-set 2/10"
[1] "Predicting cv-set 3/10"
[1] "Predicting cv-set 4/10"
[1] "Predicting cv-set 5/10"
[1] "Predicting cv-set 6/10"
[1] "Predicting cv-set 7/10"
[1] "Predicting cv-set 8/10"
[1] "Predicting cv-set 9/10"
[1] "Predicting cv-set 10/10"
```

This is done by computing the conditional expectation of the left-out observations, as indicated in by the columns in `Ind.cv`, given all other observations and the estimated parameters. The conditional expectation is computed under the assumption that the observed and “unobserved” (i.e., left out) locations follow a joint multivariate normal distribution characterised by the parameters we have estimated. Further details can be found in (Lindström et al., 2011, Szpiro et al., 2010). It should be noted that the cross-validated predictions are conceptually identical to the predictions of unobserved locations descriped in Appendix B.

We first examine the results of the predictions:

```
> names(pred.cv)
[1] "pred.obs"   "pred.by.cv" "pred.all"
```

Here the `pred.obs` contains a data frame with predictions, prediction variances and observations for each observed space-time location (this matches the observations in `mesa.data.model$obs$obs`)

```
> head(pred.cv$pred.obs)
          obs     pred   pred.var
[1,] 4.577684 4.407115 0.14053531
[2,] 4.131632 4.518086 0.14223372
[3,] 4.727882 4.881646 0.09687876
[4,] 5.352608 5.110592 0.17340530
[5,] 5.281452 5.208341 0.08265215
[6,] 4.984585 4.864277 0.20100392
```

Since `pred.obs` *only gives predictions at observed* space-time locations the full predictions are collected in the list `pred.all`

```
> str(pred.cv$pred.all[[1]])
 num [1:280, 1:3, 1:3] 4.58 3.89 4.01 4.08 3.73 ...
 - attr(*, "dimnames")=List of 3
  ..$ : chr [1:280] "1999-01-13" "1999-01-27" "1999-02-10" ...
  ..$ : chr [1:3] "60370002" "L002" "LC001"
  ..$ : chr [1:3] "obs" "pred" "pred.var"
```

Each element in the list contains a 3D-array with predictions, prediction variances and observations for the sites left out in the corresponding CV-group.

Some standard cross-validation statistics can be obtained through:

```
> pred.cv.stats <- summaryStatsCV(pred.cv, lta=TRUE, trans=0)
> names(pred.cv.stats)
[1] "Stats"    "res"      "res.norm" "lta"      "p"
```

Here `summaryStatsCV` computes prediction residuals, standardised residuals, cross-validation statistics, and the long term average at each site. Cross-validation statistics as well as the averages are computed on the *natural scale* (our observations are log NO_x), with the *exponential back-transformation* being requested by the `trans=0` option.

```
> pred.cv.stats$Stats
        RMSE        R2  coverage
obs 17.99432 0.7969228 0.9217828
lta 12.44381 0.5433923        NA
```

For cross-validation statistics the function gives us RMSE, R^2, and the coverage of the observations by a calculated 95% prediction intervals. The cross-validated statistics show a RMSE that is reasonably small, while the cross-validated R^2 is reasonably large. Also, the 95% prediction intervals provide coverage that is close to 95%, at 0.922.

As discussed above, the cross-validation function `predictCV` leaves one site out at a time and uses the estimated parameters to predict the left-out data. To assess the model's predictive ability we plot four predicted time series (with 95% confidence intervals), and the left out observations (in red), shown in Figure 13.

```
##Plot observations with CV-predictions and
##95% prediction intervals, see Figure 13
par(mfcol=c(4,1),mar=c(2.5,2.5,2,.5))
plotCV(pred.cv,  1, mesa.data.model)
plotCV(pred.cv, 10, mesa.data.model)
plotCV(pred.cv, 17, mesa.data.model)
plotCV(pred.cv, 22, mesa.data.model)
```

The object `pred.cv` contains the observations and predictions made using `predictCV`. Remember that this function takes our (number of observations)-by-(number of groups) data frame of logical values and predicts at the locations and times denoted by `TRUE`. The function `plotCV` plots the observations and predictions with corresponding 95% prediction intervals for each location specified; this is what we are specifying by the integers in e.g. `plotCV(pred.cv, 1, mesa.data.model)`. We can see the plots below:

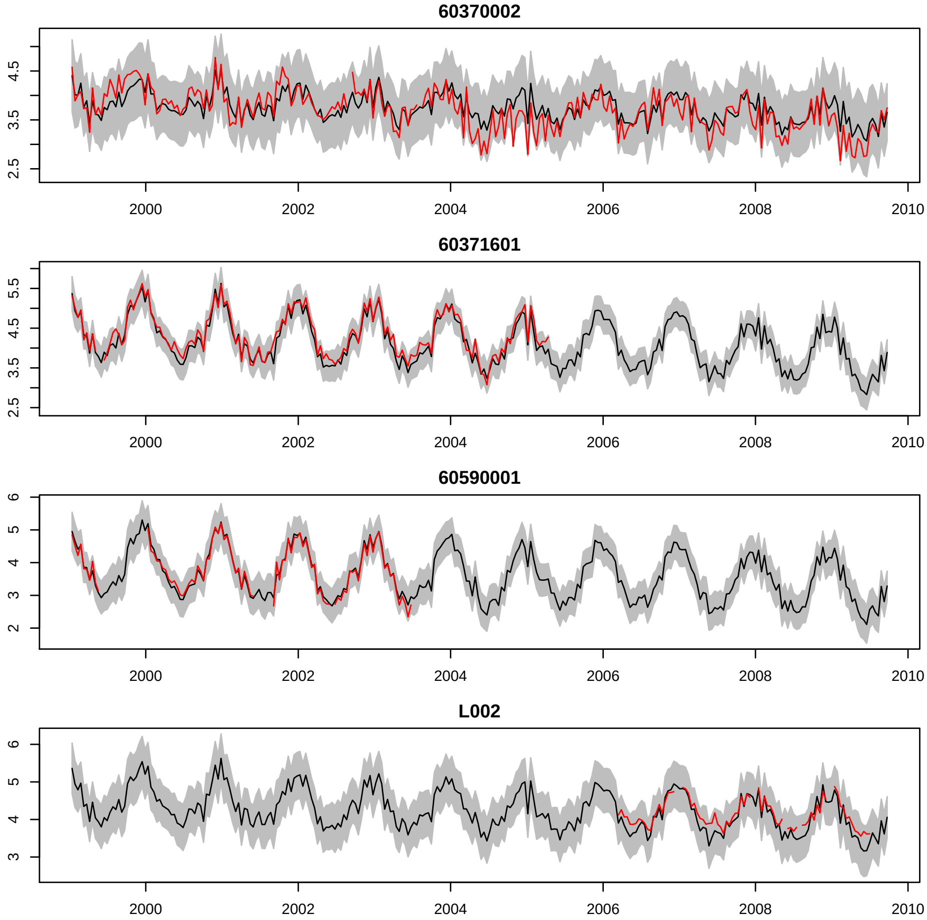

Figure 13: Observed and predicted time series and 95% prediction intervals at 4 locations

In Figure 13 we can see that the observations lie reasonably close to the predicted log NO_x concentrations. The 95% prediction intervals also do a good job of covering the observations, and it seems the model has performed well.

Another option is to do a scatter plot of the left out data against the predicted (points colour-coded by site), shown in Figure 14.

```
##Plot predicted vs.  observed, see Figure 14
par(mfcol=c(1,1),mar=c(4.5,4.5,2,2))
plot(pred.cv$pred.obs[,"obs"], pred.cv$pred.obs[,"pred"],
     xlab="Observations", ylab="Predictions", pch=19,
     cex=.1, col=mesa.data.model$obs$idx)
abline(0,1,col="grey")
```

Or a scatter plot of the average at each of the 25 sites against the predicted averages, shown in Figure 15.

```
##Predicted long term average against observations
##(on the natural scale), see Figure 15

par(mfcol=c(1,1),mar=c(4.5,4.5,3.5,2))
plot(pred.cv.stats$lta[,"obs"], pred.cv.stats$lta[,"pred"],
     xlab="Observations", ylab="Predictions", main="Averages",
     ylim=c(25,95), xlim=c(25,95))
abline(0,1,col="grey")
```

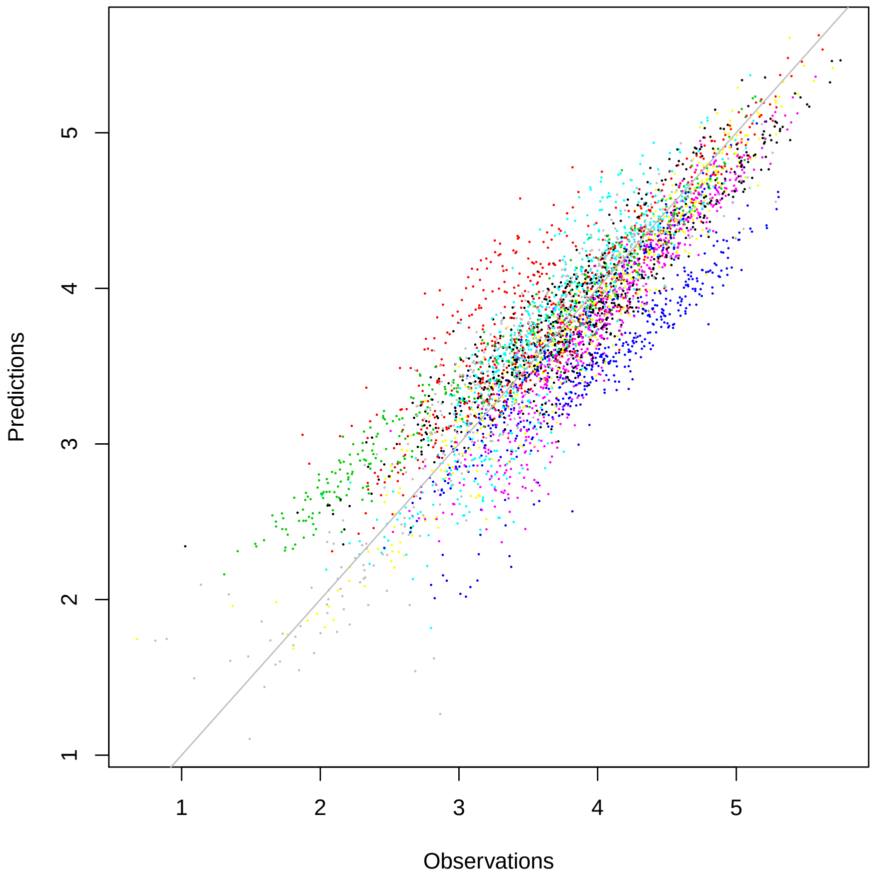

Figure 14: Plot of observations vs. predictions, the points are colour-coded by site.

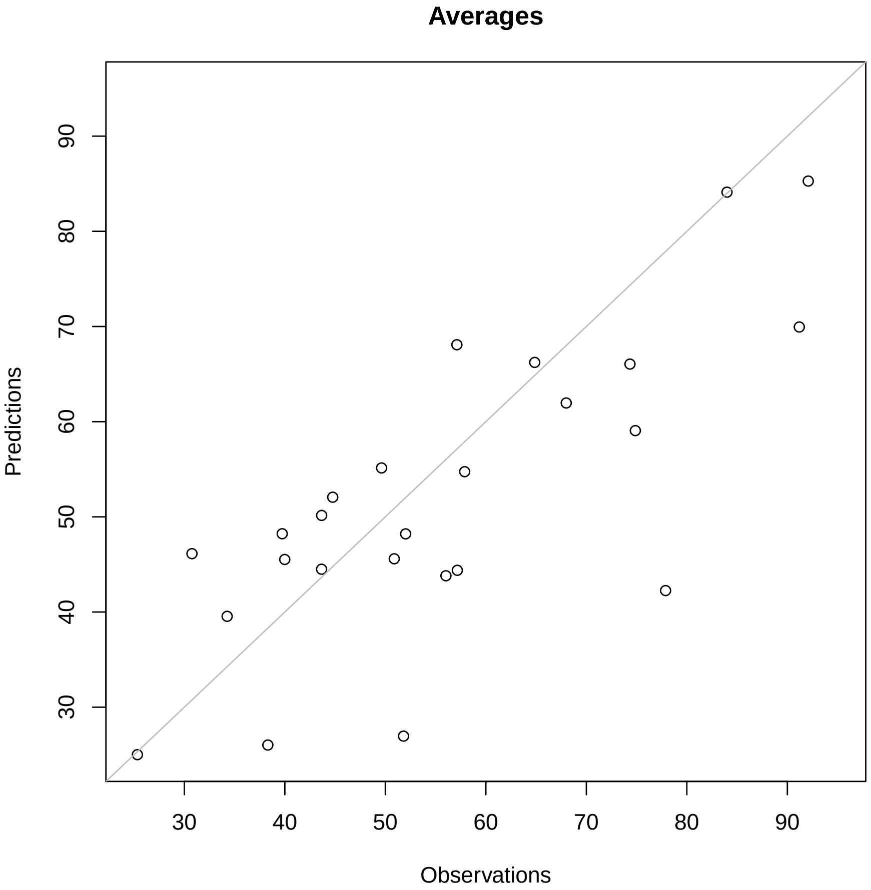

Figure 15: Plot of the observed long term average at each site against the cross-validated predictions. The data has been back-transformed to the original scale by taking exponents *before* computing the averages.

8.2.1 Residual Analysis

Before we start with a more thorough residual analysis, we need to create an indicator vector for which season each observation belongs to.

```
##create a vector dividing data into four seasons
I.season <- matrix(NA,length(mesa.data.model$obs$date),1)
I.season[months(mesa.data.model$obs$date) %in%
         c("December","January","February"),1] <- "DJF"
I.season[months(mesa.data.model$obs$date) %in%
         c("March","April","May"),1] <- "MAM"
I.season[months(mesa.data.model$obs$date) %in%
```

```
         c("June","July","August"),1] <- "JJA"
I.season[months(mesa.data.model$obs$date) %in%
         c("September","October","November"),1] <- "SON"
I.season <- factor(I.season,levels=c("DJF","MAM","JJA","SON"))
```

Given this we can investigate how many observations we have during each season.

```
> table(I.season)
I.season
 DJF  MAM  JJA  SON
1096 1223 1172 1086
```

We now take a look at the residuals from the prediction, to do this we use the `pred.cv.stats` object computed above which contains prediction residuals. First we'll examine a residual QQ-plot to assess the normality of the residuals, both raw and normalised, shown in Figure 16.

```
##Residual QQ-plots, see Figure 16
par(mfrow=c(1,2),mar=c(3,2,1,1),pty="s")
##Raw Residual QQ-plot
CVresiduals.qqnorm(pred.cv.stats$res, I.season=I.season)
##Normalized Residual QQ-plot
CVresiduals.qqnorm(pred.cv.stats$res.norm, norm=TRUE,
                   I.season=I.season)
```

Here we have used the `I.season` indicator to colour code our observations, this should help us to detect any seasonal effects on the predictions.

In Figure 16 the raw residuals are on the left and the normalised on the right. Both appear close to normal, though both have slightly heavier tails than a normal distribution. Though the departure from normality is not drastic for either, the normalised residuals are noticeably closer to being approximately normally distributed.

We can also plot the residuals versus the temporal trends or versus any of the land use regression variables. The residuals of the prediction are contained in `pred.cv.stats$res`. In Figure 17 we will plot the residuals against the first temporal trend and one of the LUR variables used to model the $\beta_0(s)$-field .

```
##Residual Scatter Plots, see Figure 17
par(mfcol=c(2,1),mar=c(4.5,4.5,2,2))
CVresiduals.scatter(pred.cv.stats$res, mesa.data.model$F[,2],
     I.season=I.season, xlab="First temporal smooth",
```

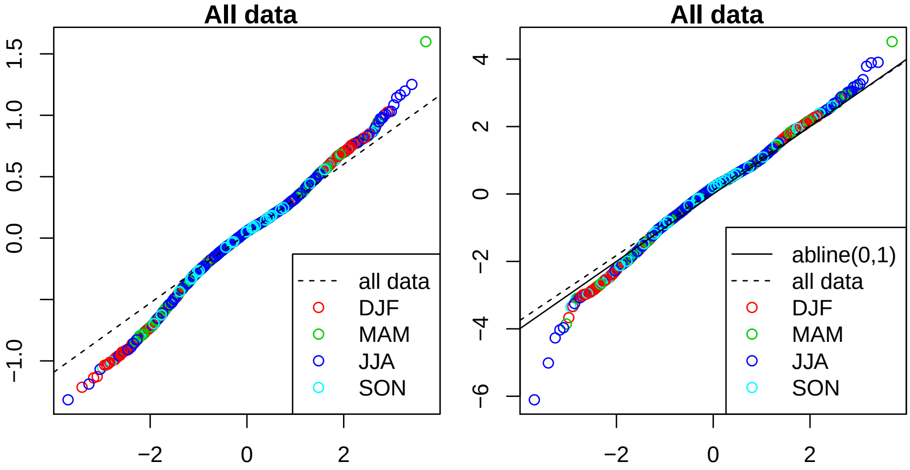

Figure 16: QQ-plots for the residuals, colour-coded by season.

```
     main="CV Residuals - All data")
CVresiduals.scatter(pred.cv.stats$res,
     mesa.data.model$X[[1]][ mesa.data.model$obs$idx, 2],
     I.season=I.season, main="CV Residuals - All data",
     xlab=colnames(mesa.data.model$X[[1]])[2])
```

Finally we can use the `mesa.data.model$location$type` field to distinguish between AQS and Mesa FIXED sites, doing separate plots for separate types of sites, see Figure 18.

```
##Residual Scatter Plots by type of site, see Figure 18
par(mfcol=c(2,2),mar=c(4.5,4.5,2,2))
CVresiduals.scatter(pred.cv.stats$res,
  mesa.data.model$X[[1]][ mesa.data.model$obs$idx, 2], I.type=
  mesa.data.model$location$type[mesa.data.model$obs$idx],
  I.season=I.season, xlab=colnames(mesa.data.model$X[[1]])[2],
  main="CV Residuals - ")
```

The plots in Figure 17 and 18 show us residuals that are roughly centred around zero and that are relatively constant over space and time. This is good to see: the model seems to be capturing the spatio-temporal relationships of $\mathrm{NO_x}$ in the data set.

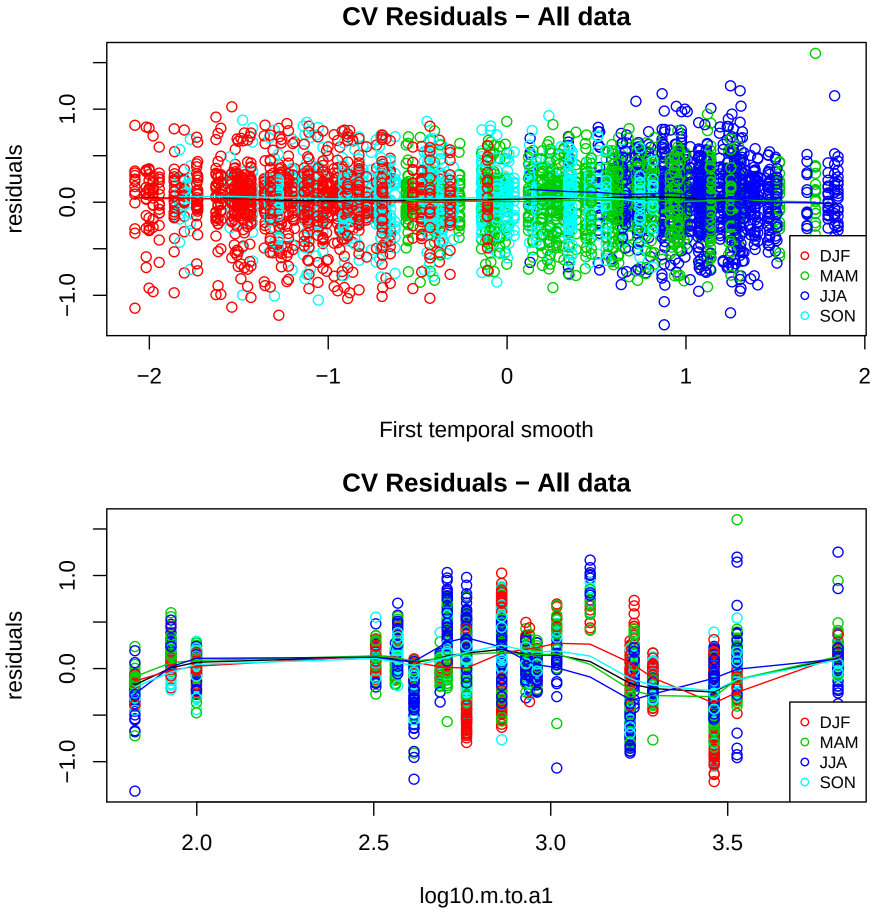

Figure 17: Residual scatter plots

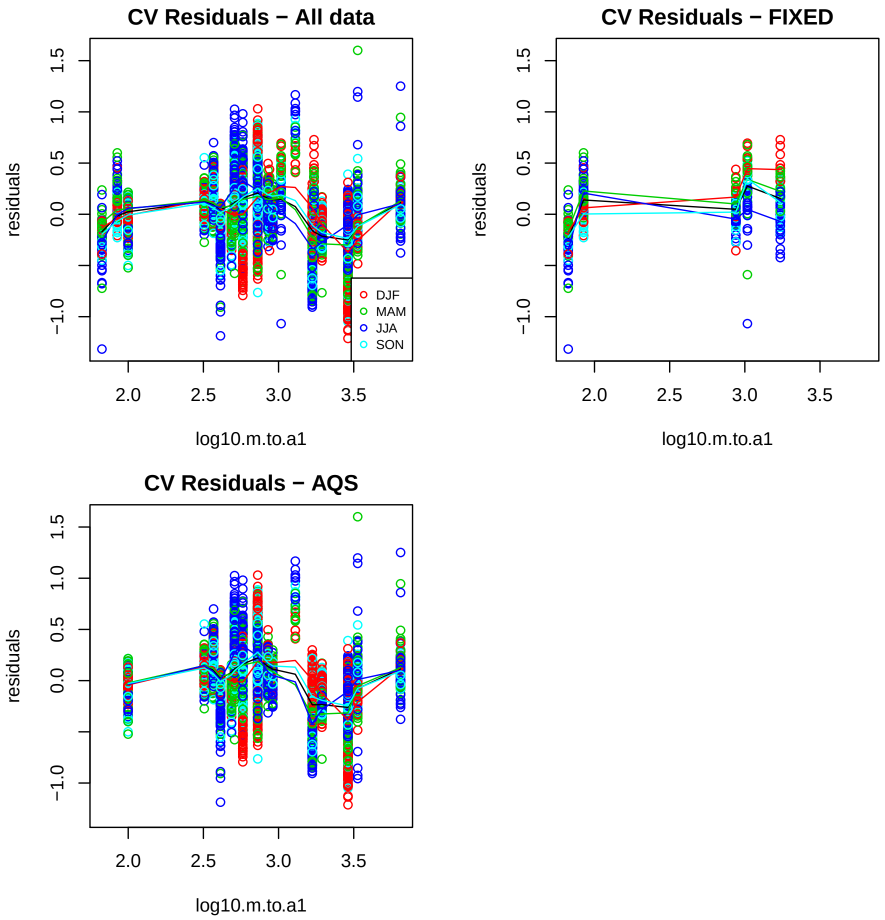

Figure 18: Residual scatter plots

Acknowledgements

Data used in the examples has been provided by **the Multi-Ethnic Study of Atherosclerosis and Air Pollution (MESA Air)**. Details regarding the data can be found in Cohen et al. (2009), Wilton et al. (2010).

Although this tutorial and development of the package described there in has been funded wholly or in part by the United States Environmental Protection Agency through **assistance agreement CR-834077101-0** and **grant RD831697** to the University of Washington, it has not been subjected to the Agency's required peer and policy review and therefore does not necessarily reflect the views of the Agency and no official endorsement should be inferred.

Travel for Johan Lindström has been paid by **STINT Grant IG2005-2047**.

Additional funding was provided by grants to the University of Washington from the **National Institute of Environmental Health Sciences (P50 ES015915)** and the **Health Effects Institute (4749-RFA05-1A/06-10)**.

References

Casella, G. and Berger, R. L. (2002). *Statistical Inference*. Duxbury, second edition.

Cohen, M. A., Adar, S. D., Allen, R. W., Avol, E., Curl, C. L., Gould, T., Hardie, D., Ho, A., Kinney, P., Larson, T. V., Sampson, P. D., Sheppard, L., Stukovsky, K. D., Swan, S. S., Liu, L.-J. S., and Kaufman, J. D. (2009). Approach to estimating participant pollutant exposures in the Multi-Ethnic Study of Atherosclerosis and air pollution (MESA air). *Environmental Science & Technology*, 43(13):4687–4693.

Cressie, N. (1993). *Statistics for Spatial Data*. John Wiley & Sons Ltd, revised edition.

EPA (1992). User's guide to CAL3QHC version 2.0: A modeling methodology for predicting pollutant concentrations near roadway intersections. Technical Report EPA-454/R-92-006, U.S. Environmental Protection Agency, Research Triangle Park, NC, USA.

Fuentes, M., Guttorp, P., and Sampson, P. D. (2006). Using transforms to analyze space-time processes. In Finkenstadt, B., Held, L., and Isham, V., editors, *Statistical Methods for Spatio-Temporal Systems*, pages 77–150. CRC/Chapman and Hall.

Hastings, W. (1970). Monte Carlo sampling methods using Markov chains and their applications. *Biometrika*, 57:97–109.

Jerrett, M., Burnett, R. T., Ma, R., Pope, C. A., Krewski, D., Newbold, K. B., Thurston, G., Shi, Y., Finkelstein, N., Calle, E. E., and Thun, M. J. (2005). Spatial analysis of air pollution mortality in Los Angeles. *Epidemiology*, 16(6):727–736.

Lindström, J., Szpiro, A. A., Sampson, P. D., Sheppard, L., Oron, A., Richards, M., and Larson, T. (2011). A flexible spatio-temporal model for air pollution: Allowing for spatio-temporal covariates. Technical Report Working Paper 370, UW Biostatistics Working Paper Series.

Mercer, L., Szpiro, A. A., Sheppard, L., Adar, S., Allen, R., Avol, E., Lindström, J., Oron, A., Larson, T., Liu, L.-J. S., and Kaufman, J. (2011). Predicting concentrations of oxides of nitrogen in Los Angeles, CA using universal kriging. *Submitted*.

MESA Air Data Team (2010). Documentation of MESA air implementation of the Caline3QHCR model. Technical report, University of Washington, Seattle, WA, USA.

Metropolis, N., Rosenbluth, A., Rosenbluth, M., Teller, A., and Teller, E. (1953). Equations of state calculations by fast computing machines. *J. Chem. Phys.*, 21:1087–1092.

Roberts, G., Gelman, A., and Gilks, W. (1997). Weak convergence and optimal scaling of random walk metropolis algorithms. *Ann. Appl. Probab.*, 7:110–120.

Sampson, P. D., Szpiro, A. A., Sheppard, L., Lindström, J., and Kaufman, J. D. (2009). Pragmatic estimation of a spatio-temporal air quality model with irregular monitoring data. Technical Report Working Paper 353, UW Biostatistics Working Paper Series.

Szpiro, A. A., Sampson, P. D., Sheppard, L., Lumley, T., Adar, S., and Kaufman, J. (2010). Predicting intra-urban variation in air pollution concentrations with complex spatio-temporal dependencies. *Environmetrics*, 21(6):606–631.

Wilton, D., Szpiro, A. A., Gould, T., and Larson, T. (2010). Improving spatial concentration estimates for nitrogen oxides using a hybrid meteorological dispersion/land use regression model in Los Angeles, CA and Seattle, WA. *SciTotalEnviron*, 408(5):1120–1130.

A R–Code

The following is all the R–code used in the tutorial:

```
### section 4 Preliminaries ###
##A qqplot for some N(0,1) random numbers
qqnorm(rnorm(100))

##Summary of 100 N(0,1) numbers.
summary(rnorm(100))

##Load the spatio-temporal package
library(SpatioTemporal)

##A package used for some plots in this tutorial (plotCI)
library(plotrix)
###The maps to provide reference maps
library(maps)

##Load the data
data(mesa.data)
##...and the precomputed results
data(mesa.data.res)

### subsection 4.1 Examining the Data ###
names(mesa.data)

### subsubsection 4.1.1 The mesa.data$location Data Frame ###
head(mesa.data$location)

###Plot the locations, see Figure 1
par(mfrow=c(1,1))
plot(mesa.data$location$long,mesa.data$location$lat,
     pch=24,bg=c("red","blue")[mesa.data$location$type],
     xlab="Longitude",ylab="Latitude")

###Add the map of LA
map("county","california",col="#FFFF0055",fill=TRUE,add=TRUE)

##Add a legend
legend("bottomleft",c("AQS","FIXED"),pch=24,bty="n",
       pt.bg=c("red","blue"))
```

```
### subsubsection 4.1.2 The mesa.data$LUR Data Frame ###
head(mesa.data$LUR)

### subsubsection 4.1.3 The mesa.data$trend Data Frame ###
head(mesa.data$trend)
range(mesa.data$trend$date)

### subsubsection 4.1.4 The mesa.data$obs Data Frame ###
head(mesa.data$obs)

### subsubsection 4.1.5 The mesa.data$SpatioTemporal Array ###
dim(mesa.data$SpatioTemp)
mesa.data$SpatioTemp[1:5,1:5,]

str(dimnames(mesa.data$SpatioTemp))
as.character(sort(unique(c(mesa.data$obs$date,
    mesa.data$trend$date))))
dimnames(mesa.data$SpatioTemp)[[3]]

### subsection 4.2 Summaries of mesa.data ###
printMesaDataNbrObs(mesa.data)

###Plot when observations occurr, see Figure 2
par(mfcol=c(1,1),mar=c(4.3,4.3,1,1))
plotMonitoringLoc(mesa.data)

### subsection 4.3 Creating mesa.data from Raw Data ###
data(mesa.data.raw)
names(mesa.data.raw)
names(mesa.data)  ###Recall the data frames in mesa.data

head(mesa.data.raw$X)

mesa.data.raw$obs[1:6,1:5]

mesa.data.raw$lax.conc.1500[1:6,1:5]

### subsubsection 4.3.1 Creating location ###
```

```
##extract components from mesa.data.raw
##to create the location data.frame
location <- mesa.data.raw$X[,c("ID", "x", "y", "long",
                               "lat", "type")]
##ensure that it's a data.frame
location <- as.data.frame(location)
##ensure that ID and type are factors
location$ID <- factor(as.character(location$ID))
location$type <- factor(as.character(location$type))

### subsubsection 4.3.2 Creating LUR ###
##create a covariate LUR matrix
LUR <- mesa.data.raw$X
##ensure that it's a data.frame
LUR <- as.data.frame(LUR)
##take rownames as the IDs and drop location information
rownames(LUR) <- as.character(LUR$ID)
LUR <- LUR[,!(names(LUR) %in% c("ID", "x", "y", "long",
           "lat", "type"))]

### subsubsection 4.3.3 Creating obs ###
##create a data.frame of observations
T <- t(matrix(rownames(mesa.data.raw$obs),
              nrow=dim(mesa.data.raw$obs)[2],
              ncol=dim(mesa.data.raw$obs)[1],byrow=TRUE))
ID <- matrix(colnames(mesa.data.raw$obs),
             nrow=dim(mesa.data.raw$obs)[1],
             ncol=dim(mesa.data.raw$obs)[2],byrow=TRUE)
obs <- data.frame(obs=c(mesa.data.raw$obs), date=c(as.Date(T)),
                  ID=c(ID))
##drop unmonitored locations
obs <- obs[!is.na(obs$obs),,drop=FALSE]
##sort the locations (strictly not needed)
obs <- obs[order(obs$date,obs$ID),,drop=FALSE]

##examine the time and location matrices
T[1:3,1:3]
ID[1:3,1:3]

### subsubsection 4.3.4 Creating SpatioTemp ###
##create a 3D-array for the spatio-temporal covariate
```

```
ST <- array(mesa.data.raw$lax.conc.1500, dim =
            c(dim(mesa.data.raw$lax.conc.1500),1))
##add names for time, location and spatio-temporal covariate
dimnames(ST) <- list(rownames(mesa.data.raw$lax.conc),
                     colnames(mesa.data.raw$lax.conc),
                     "lax.conc.1500")

### subsubsection 4.3.5 Creating trend, and Combining it all ###
##compute the smooth trends
trend <- calc.smooth.trends(obs=obs$obs, date=obs$date,
                            ID=obs$ID, n.basis = 2)$svd

##Combining all the elements
mesa.data.alt <- list(location=location, LUR=LUR, trend=trend,
                      obs=obs, SpatioTemp=ST)

### subsection 5.1 Data Set-Up for Trend Estimation ###
##extract a data matrix
D <- create.data.matrix(mesa.data)
##and study the data
dim(D)
D[1:6,1:5]
colnames(D)

##subset the data
ID.subset <-mesa.data$location[mesa.data$location$type ==
                               "AQS",]$ID
D2 <- create.data.matrix(mesa.data, subset=ID.subset)
##and study the result
dim(D2)
colnames(D2)

### subsection 5.2 Cross-Validated Trend Estimations ###
##Run leave one out cross-validation to find smooth trends
SVD.cv <- SVD.smooth.cv(D,1:5)

### subsubsection 5.2.4 Evaluating the Cross-Validated Results ###

##Plotting the smooth trends CV results, see Figure 3
par(mfcol=c(2,2),mar=c(4,4,.5,.5))
plot(SVD.cv$CV.stat$RMSE, type="l", ylab="RMSE", xlab="")
plot(SVD.cv$CV.stat$R2, type="l", ylab="R2", xlab="")
plot(SVD.cv$CV.stat$BIC, type="l", ylab="BIC", xlab="")
```

```
##Scatterplots of BIC values for different number of trends,
##see Figure 4
par(mfcol=c(1,1),mar=c(4,4,.5,.5))
pairs(SVD.cv$BIC.all, pane =
      function(x,y){points(x,y);abline(0,1)})

### subsection 5.3 Creating and Investigating the Trends ###
##compute new temporal smooths
F <- calc.smooth.trends(mesa.data, n.basis=2)
##and add the new trends to the data structure
mesa.data$trend <- F$svd

##plot the observations at two of the locations along
##with the fitted smooth trends, see Figure 5
par(mfrow=c(4,1),mar=c(2.5,2.5,2,.5))
plotMesaData(mesa.data, 5, type="obs")
plotMesaData(mesa.data, 18, type="obs")

##also plot the residuals
plotMesaData(mesa.data, 5, type="res")
plotMesaData(mesa.data, 18, type="res")

##Autocorrelation of the residuals at four locations,
##see Figure 6
par(mfcol=c(2,2),mar=c(2.5,2.5,3,.5))
plotMesaData(mesa.data, 1, type="acf")
plotMesaData(mesa.data, 5, type="acf")
plotMesaData(mesa.data, 13, type="acf")
plotMesaData(mesa.data, 18, type="acf")

### section 6 Empirical Estimation of the β-Fields ###
##create data matrix
D <- create.data.matrix(mesa.data)
beta <- matrix(NA,dim(D)[2], dim(mesa.data$trend)[2])
beta.std <- beta
##extact the temporal trends
F <- mesa.data$trend
##drop the date column
F$date <- NULL
```

```
##estimate the beta-coeficients at each location
for(i in 1:dim(D)[2]){
  tmp <- summary(lm(D[,i] ~ as.matrix(F)))
  beta[i,] <- tmp$coefficients[,1]
  beta.std[i,] <- tmp$coefficients[,2]
}

##Add names to the estimated betas
colnames(beta) <- c("const",colnames(F))
rownames(beta) <- colnames(D)
dimnames(beta.std) <- dimnames(beta)

##examine the beta fields
head(beta)
head(beta.std)

### subsection 7.1 Parameter Estimation ###
##Examine available spatial, ...
names(mesa.data$LUR)
##... and spatio-temporal covariates
dimnames(mesa.data$SpatioTemp)[[3]]

##specify the model
mesa.data.model <- create.data.model(mesa.data, LUR =
    list(c("log10.m.to.a1", "s2000.pop.div.10000",
           "km.to.coast"), "km.to.coast", "km.to.coast"),
    ST.Ind=NULL)

##examine the model
names(mesa.data.model)
mesa.data.model$LUR.list
mesa.data.model$ST.Ind

##covariates for the temporal intercept
head(mesa.data.model$X$const)
##...and the two smooth temporal trends
head(mesa.data.model$X$V1)
head(mesa.data.model$X$V2)

##Some important dimensions of the model
dim <- loglike.dim(mesa.data.model)
```

```
print(dim)

##Setting up the initial parameter values for optimization
x.init <- cbind(rep(2,dim$nparam.cov),
                c(rep(c(1,-3),dim$m+1),-3))

##Names of the covariance parameters
loglike.var.names(mesa.data.model,all=FALSE)

##Add names to the initial values
rownames(x.init) <- loglike.var.names(mesa.data.model,
                                      all=FALSE)
x.init

##estimate parameters
if(FALSE){
  ##This may take a while...
  par.est <- fit.mesa.model(x.init, mesa.data.model, type="p",
      hessian.all=TRUE, control=list(trace=3,maxit=1000))
}else{
  ##Get the precomputed optimisation results instead.
  par.est <- mesa.data.res$par.est
}

##compare function values
loglike(par.est$res.best$par, mesa.data.model)
par.est$res.best$value

### subsection 7.2 Evaluating the Results ###
##Optimisation status message
par.est$message

##the optimisation results
names(par.est)

##examine optimisation results
names(par.est$res.best)
names(par.est$res.all[[1]])
names(par.est$res.all[[2]])

##lets compare the estimated parameters
```

```
cbind(par.est$res.all[[1]]$par.all,
      par.est$res.all[[2]]$par.all)
##and values of the likelihood
cbind(par.est$res.all[[1]]$value,
      par.est$res.all[[2]]$value)

##extract the estimated parameters
x <- par.est$res.best$par.all

##and approximate uncertainties from the hessian
x.sd <- sqrt(diag(-solve(par.est$res.best$hessian.all)))
names(x.sd) <- names(x)

##Plot the estimated parameters with uncertainties, see Figure 7
par(mfrow=c(1,1),mar=c(13.5,2.5,.5,.5))
plotCI(x,uiw=1.96*x.sd,ylab="",xlab="",xaxt="n")
points(par.est$res.all[[2]]$par.all, col=2, pch=4)
abline(h=0, col="grey")
axis(1,1:length(x),names(x),las=2)

### subsection 7.3 Predictions ###
##compute conditional expectations (takes roughly 1 min)
EX <- cond.expectation(par.est$res.best$par, mesa.data.model,
                       compute.beta = TRUE)

##study the results
names(EX)

##Plot that compares two estimated beta-fields, see Figure 8
par(mfcol=c(1,1), mar=c(4.5,4.5,2,.5), pty="s")
plotCI(x=beta[,1], y=EX$EX.beta[,1],
       uiw=1.96*beta.std[,1], err="x",
       main="Intercept", pch=NA, sfrac=0.005,
       xlab="Empirical estimate",
       ylab="Spatio-Temporal Model")
plotCI(x=beta[,1],y=EX$EX.beta[,1],
       uiw=1.96*sqrt(EX$VX.beta[,1]),
       add=TRUE, pch=NA, sfrac=0.005)
abline(0,1,col="grey")

##...and the other two beta-fields, see Figure 9
```

```
par(mfcol=c(1,2), mar=c(4.5,4.5,2,.5), pty="s")
for(i in 2:3){
  plotCI(x=beta[,i], y=EX$EX.beta[,i],
         uiw=1.96*beta.std[,i], err="x",
         main=colnames(beta)[i], pch=NA, sfrac=0.005,
         xlab="Empirical estimate",
         ylab="Spatio-Temporal Model")
  plotCI(x=beta[,i],y=EX$EX.beta[,i],
         uiw=1.96*sqrt(EX$VX.beta[,i]),
         add=TRUE, pch=NA, sfrac=0.005)
  abline(0,1,col="grey")
}

##plot predictions and observations for 4 locations, see Figure 10
par(mfrow=c(4,1),mar=c(2.5,2.5,2,.5))
plotPrediction(EX,  1, mesa.data.model)
lines(as.Date(rownames(EX$EX.mu)),EX$EX.mu[,1],col=3)
lines(as.Date(rownames(EX$EX.mu.beta)),
      EX$EX.mu.beta[,1],col=4)
plotPrediction(EX, 10, mesa.data.model)
lines(as.Date(rownames(EX$EX.mu)),EX$EX.mu[,10],col=3)
lines(as.Date(rownames(EX$EX.mu.beta)),
      EX$EX.mu.beta[,10],col=4)
plotPrediction(EX, 17, mesa.data.model)
lines(as.Date(rownames(EX$EX.mu)),EX$EX.mu[,17],col=3)
lines(as.Date(rownames(EX$EX.mu.beta)),
      EX$EX.mu.beta[,17],col=4)
plotPrediction(EX, 22, mesa.data.model)
lines(as.Date(rownames(EX$EX.mu)),EX$EX.mu[,22],col=3)
lines(as.Date(rownames(EX$EX.mu.beta)),
      EX$EX.mu.beta[,22],col=4)

### section 8 Cross-validation ###
##create the CV structure defining 10 different CV-groups
Ind.cv <- createCV(mesa.data.model, groups=10, min.dist=.1)
head(Ind.cv)

##number of observations in each CV-group
colSums(Ind.cv)

##Which sites belong to which groups?
```

```
ID.cv <- lapply(apply(Ind.cv,2,list),function(x)
               unique(mesa.data.model$obs$ID[x[[1]]]))
ID.cv

##Distances between the sites in the 10th CV-group
##Note that the sites with distance 0.084<min.dist
##are grouped together.
mesa.data.model$dist[ID.cv[[10]],ID.cv[[10]]]

##Find out which location belongs to which cv group
I.col <- apply(sapply(ID.cv,
    function(x) mesa.data.model$location$ID %in% x),1,
    function(x) if(sum(x)==1) which(x) else 0)
names(I.col) <- mesa.data.model$location$ID
I.col

##Plot the locations, colour coded by CV-grouping, see Figure 11
par(mfrow=c(1,1))
plot(mesa.data$location$long,mesa.data$location$lat,
     pch=23+floor(I.col/max(I.col)+.5), bg=I.col,
     xlab="Longitude",ylab="Latitude")

##Add the map of LA
map("county","california",col="#FFFF0055",fill=TRUE,add=TRUE)

### subsection 8.1 Cross-validated Estimation ###
if(FALSE){
  ##estimate different parameters for each CV-group
  dim <- loglike.dim(mesa.data.model)
  x.init <- cbind(rep(2,dim$nparam.cov),
                  c(rep(c(1,-3),dim$m+1),-3))
  ##This may take a while...
  par.est.cv <- estimateCV(x.init, mesa.data.model, Ind.cv)
}else{
  ##Get the precomputed results instead.
  par.est.cv <- mesa.data.res$par.est.cv
}

##lets examine the results
names(par.est.cv)
par.est.cv$par
```

```
##res.all contains a list with the optimisation results
##for each CV-group
str(par.est.cv$res.all[[1]])

##Studying the conv-field of all the groups
sapply(par.est.cv$res.all,function(x) x$conv)
##we see that all optimisations have converged.

##We can also see that different starting values
##where best for different cv-groups
sapply(par.est.cv$res.all,function(x) x$par.init)

##Boxplot of the different estimates from the CV, see Figure 12
par(mfrow=c(1,1), mar=c(7,2.5,2,.5), las=2)
boxplot(t(par.est.cv$par))
points(par.est$res.best$par, pch=4, col=2)

### subsection 8.2 Cross-Validated Prediction ###
##Do cross-validated predictions using the newly
##estimated paremeters (takes roughly 1 minute)
pred.cv <- predictCV(par.est.cv$par, mesa.data.model,
                     Ind.cv, silent = FALSE)

##examine the results
names(pred.cv)
head(pred.cv$pred.obs)
str(pred.cv$pred.all[[1]])

##compute CV-statistics (see also predictNaive)
##this uses compute.ltaCV internally to compute the
##long term averages if we want just lta:s then use
##compute.ltaCV directly.
pred.cv.stats <- summaryStatsCV(pred.cv, lta=TRUE, trans=0)
names(pred.cv.stats)
pred.cv.stats$Stats

##Plot observations with CV-predictions and
##95% prediction intervals, see Figure 13
par(mfcol=c(4,1),mar=c(2.5,2.5,2,.5))
plotCV(pred.cv,  1, mesa.data.model)
plotCV(pred.cv, 10, mesa.data.model)
plotCV(pred.cv, 17, mesa.data.model)
plotCV(pred.cv, 22, mesa.data.model)

##Plot predicted vs.  observed, see Figure 14
par(mfcol=c(1,1),mar=c(4.5,4.5,2,2))
plot(pred.cv$pred.obs[,"obs"], pred.cv$pred.obs[,"pred"],
     xlab="Observations", ylab="Predictions", pch=19,
     cex=.1, col=mesa.data.model$obs$idx)
abline(0,1,col="grey")

##Predicted long term average against observations
##(on the natural scale), see Figure 15
par(mfcol=c(1,1),mar=c(4.5,4.5,3.5,2))
plot(pred.cv.stats$lta[,"obs"], pred.cv.stats$lta[,"pred"],
     xlab="Observations", ylab="Predictions", main="Averages",
     ylim=c(25,95), xlim=c(25,95))
abline(0,1,col="grey")

### subsubsection 8.2.1 Residual Analysis ###
##create a vector dividing data into four seasons
I.season <- matrix(NA,length(mesa.data.model$obs$date),1)
I.season[months(mesa.data.model$obs$date) %in%
         c("December","January","February"),1] <- "DJF"
I.season[months(mesa.data.model$obs$date) %in%
         c("March","April","May"),1] <- "MAM"
I.season[months(mesa.data.model$obs$date) %in%
         c("June","July","August"),1] <- "JJA"
I.season[months(mesa.data.model$obs$date) %in%
         c("September","October","November"),1] <- "SON"
I.season <- factor(I.season,levels=c("DJF","MAM","JJA","SON"))

##No. observations in each season
table(I.season)

##Residual QQ-plots, see Figure 16
par(mfrow=c(1,2),mar=c(3,2,1,1),pty="s")
##Raw Residual QQ-plot
CVresiduals.qqnorm(pred.cv.stats$res, I.season=I.season)
##Normalized Residual QQ-plot
CVresiduals.qqnorm(pred.cv.stats$res.norm, norm=TRUE,
                   I.season=I.season)
```

```
##Residual Scatter Plots, see Figure 17
par(mfcol=c(2,1),mar=c(4.5,4.5,2,2))
CVresiduals.scatter(pred.cv.stats$res, mesa.data.model$F[,2],
     I.season=I.season, xlab="First temporal smooth",
     main="CV Residuals - All data")
CVresiduals.scatter(pred.cv.stats$res,
     mesa.data.model$X[[1]][ mesa.data.model$obs$idx, 2],
     I.season=I.season, main="CV Residuals - All data",
     xlab=colnames(mesa.data.model$X[[1]])[2])

##Residual Scatter Plots by type of site, see Figure 18
par(mfcol=c(2,2),mar=c(4.5,4.5,2,2))
CVresiduals.scatter(pred.cv.stats$res,
  mesa.data.model$X[[1]][ mesa.data.model$obs$idx, 2], I.type=
  mesa.data.model$location$type[mesa.data.model$obs$idx],
  I.season=I.season, xlab=colnames(mesa.data.model$X[[1]])[2],
  main="CV Residuals - ")
```

B Prediction at Unobserved Locations

The following is an example of predictions at unobserved locations and times.

B.1 Load Data

Let us first load relevant libraries and data.

```
##Load the spatio-temporal package
library(SpatioTemporal)
##And additional packages for plotting
library(plotrix)

##load data
data(mesa.data)
##...and optimisation results
data(mesa.data.res)
```

B.2 Setup and Study the Data

Then we setup the data structures — dropping some observations to simulate a case with observed and unobserved locations — and study the available data.

```
##store the original data structure
mesa.data.org <- mesa.data

##keep only observations from the AQS sites
##This gives us 5 "unobserved" sites at which to predict
ID.AQS <- mesa.data$location$ID[mesa.data$location$type=="AQS"]
mesa.data$obs <- mesa.data$obs[mesa.data$obs$ID %in% ID.AQS,]

##Let us also expand the temporal trends to be every week
##instead of every 2-weeks, giving us some unobserved
##time-points at which to predict.
T <- seq(min(mesa.data$trend$date), max(mesa.data$trend$date),
         by=7)
mesa.data$trend <- data.frame(
    V1=spline(x=mesa.data$trend$date, y=mesa.data$trend$V1,
             xout=T)$y,
    V2=spline(x=mesa.data$trend$date, y=mesa.data$trend$V2,
              xout=T)$y, date=T)

##study the reduced data structure, we see
##that the 5 FIXED sites lack observations.
printMesaDataNbrObs(mesa.data)
##compate this to the original data.
printMesaDataNbrObs(mesa.data.org)

##create an object containing only the unmonitored sites
mesa.unmon <- mesa.data
ID.miss <- !(mesa.unmon$location$ID %in%
             unique(mesa.unmon$obs$ID))
mesa.unmon$location <- mesa.unmon$location[ID.miss,]
mesa.unmon$LUR <- mesa.unmon$LUR[mesa.unmon$location$ID,]
mesa.unmon$SpatioTemp <-
    mesa.unmon$SpatioTemp[,mesa.unmon$location$ID,,drop=FALSE]
##drop observations from the unmonitored data
mesa.unmon$obs <- NULL

##study the data structure for the unobserved locations
printMesaDataNbrObs(mesa.unmon)

##create a model object, dropping unobserved locations.
##Dropping the unobserved locations will speed up the parameter
##estimations and is thus often a good idea.
mesa.data.model <- create.data.model(mesa.data,
    LUR = list(c("log10.m.to.a1", "km.to.coast",
      "s2000.pop.div.10000"), "km.to.coast","km.to.coast"),
    ST.Ind = NULL)

##create a model object containing all the sites (strip=FALSE)
mesa.data.model.all <- create.data.model(mesa.data,
    LUR = list(c("log10.m.to.a1", "km.to.coast",
      "s2000.pop.div.10000"), "km.to.coast","km.to.coast"),
    ST.Ind = NULL, strip=FALSE)

##note that this drops unobserved sites from the data structure
printMesaDataNbrObs(mesa.data.model)
printMesaDataNbrObs(mesa.data.model.all)
```

B.3 Predictions

Having created data structures that contain some unobserved locations and time-points we are now ready to compute predictions at these unobserved points.

In a real world application we would probably use `fit.mesa.model` to estimate parameters. However, here we will just use the parameters previously estimated in subsection 7.1 using all available data.

```
##Strictly we should now estimate the relevant parameters.
##However since this is time consuming, we instead use the
##parameters estimated for the model using all observations.
x <- mesa.data.res$par.est$res.best$par

##Now let's use these parameters to compute som predictions
##Please note that the predictions take roughly 30s each on a
##decent laptop.

##predict at times and locations in mesa.data.model (AQS-sites)
E.AQS <- cond.expectation(x, mesa.data.model,
                          compute.beta = TRUE)

##predict at times and locations in mesa.unmon (FIXED-sites)
E.FIXED <- cond.expectation(x, mesa.data.model,
    mesa.data = mesa.unmon, compute.beta = TRUE)

##predict at times in mesa.data.model and locations in both
##mesa.data.model and mesa.unmon (AQS + FIXED-sites)
##using the "combine.data" option.
E.ALL <- cond.expectation(x, mesa.data.model,
    mesa.data = mesa.unmon, compute.beta = TRUE,
    combine.data = TRUE)

##predict at times and locations in mesa.data.model.all (the
##one with strip=FALSE), this gives predictions att all
##locations and *times*.
E.ALL2 <- cond.expectation(x, mesa.data.model.all,
                           compute.beta = TRUE)

##To predict at unmonitored sites for only 2000, we first
##pick out the relevant time points in mesa.unmon$trend
```

```
mesa.unmon$trend <- mesa.data$trend
mesa.unmon$trend <-
    mesa.unmon$trend[mesa.unmon$trend$date>="2000-01-01" &
                     mesa.unmon$trend$date<"2001-01-01",]
##and then predict at these locations
E.2000 <- cond.expectation(x, mesa.data.model,
    mesa.data = mesa.unmon, compute.beta = TRUE)
```

B.4 Results

Having computed predictions in a number of different ways we now want to investigate the results.

B.4.1 Studying the Results

We start by looking at the predictions produced for the five different cases.

```
##First look at the components of the prediction structure
names(E.AQS)

##Here E.AQS$EX is a matrix containing predictions at all
##relevant times and locations. The dimension of the matrix
dim(E.AQS$EX)
colnames(E.AQS$EX)
rownames(E.AQS$EX)[1:5]
##And we see that E.AQS has predicted at the 20 locations
##and 280 time-points in mesa.data.model.

##While E.FIXED contains predictions at the 5 locations
##and 559 time-points (weekly data) in mesa.unmon
dim(E.FIXED$EX)

##Using the "combine.data" option gives predictions at
##the 20+5=25 locations in mesa.data.model and mesa.unmon
##BUT this uses the temporal trends from mesa.data.model
##resulting in predictions at ONLY 280 time-points
dim(E.ALL$EX)

##If we instead use mesa.data.model.all which contains all
##the 25 locations as well as all the 559 time-points we
```

```
##obtain predictions at ALL locations and time-points.
dim(E.ALL2$EX)

##Finally we also see that restricting mesa.unmon$trend to
##only 2000 dates gives the expected 5 locations and 52
##(weekly trend in mesa.unmon) timepoints
dim(E.2000$EX)
```

B.4.2 Plotting the Results

Having studied the number of locations and time-points at which we have predictions we now investigate the actual predictions.

Recall that we dropped all observations for the FIXED-sites to simulate a case where predictions are desired at some locations given observations at other locations. However, we still have the observations at the FIXED-sites in the `mesa.data.org` structure, this allows us to add the known but *u*nused observations to the following plots, providing a measure of how good the predictions are.

```
##Start by plotting predictions and observations
##for 3 observed locations
par(mfrow=c(3,1),mar=c(2.5,2.5,2,.5))
plotPrediction(E.AQS, "60371103", mesa.data.model)
plotPrediction(E.AQS, "60371601", mesa.data.model)
plotPrediction(E.AQS, "60590001", mesa.data.model)

##Using the E.ALL strucutre that has predictions at
##all locations and on the 2-week time scale gives
##an identical result.
par(mfrow=c(3,1),mar=c(2.5,2.5,2,.5))
plotPrediction(E.ALL, "60371103", mesa.data.model.all)
plotPrediction(E.ALL, "60371601", mesa.data.model.all)
plotPrediction(E.ALL, "60590001", mesa.data.model.all)

##Before plotting predictions for the unobserved
##locations we do a few minor things

##studying the distance between the observed and
##unobserved locations
apply(create.data.model(mesa.data.org)$dist[1:20,21:25],2,min)
```

```
##we note that L001 is very close (0.083 km) to an AQS site.

create.data.model(mesa.data.org)$dist[,"L001"]
##Which on closer inspection turns out to be site 60371103.

##Let's also extract the data matrix from the original
##data so that we can compare the predicted values with
##the known, LEFT OUT observations
D <- create.data.matrix(mesa.data.org)

##We are now ready to plot predictions for three unobserved
##locations along with the left out obserbations (and for L001
##the data from the colocated site). Plotting the left out
##observations allows us to investigate how well the
##predictions are doing

par(mfrow=c(3,1),mar=c(2.5,2.5,2,.5))
#site L001
plotPrediction(E.FIXED, "L001", mesa.unmon)
##add the colocated data
lines(as.Date(rownames(D)), D[,"60371103"], col="green")
##also add the left out observations
lines(as.Date(rownames(D)), D[,"L001"], col="red")

##and the other two sites
plotPrediction(E.FIXED, "LC002", mesa.unmon)
lines(as.Date(rownames(D)), D[,"LC002"], col="red")
plotPrediction(E.FIXED, "LC003", mesa.unmon)
lines(as.Date(rownames(D)), D[,"LC003"], col="red")

##Now plotting with the prediction structure that has computed
##predictions on the weekly time-scale. Since we now have
##observed only every other point standard usage of
##plotPreditcion will not show any observations (a line with
##only every other point...)
par(mfrow=c(4,1),mar=c(2.5,2.5,2,.5))
plotPrediction(E.ALL2, "60371103", mesa.data.model.all)
plotPrediction(E.ALL2, "60590001", mesa.data.model.all)
##Instead we need to plot the observations as points.
plotPrediction(E.ALL2, "60371103", mesa.data.model.all,
               lty=c(1,NA), pch=c(NA,19), cex=c(NA,.5))
```

```
plotPrediction(E.ALL2, "60590001", mesa.data.model.all,
               lty=c(1,NA), pch=c(NA,19), cex=c(NA,.5))
```

C MCMC

The following is an example of estimation using the Metropolis-Hastings algorithm (Metropolis et al., 1953, Hastings, 1970).

C.1 Load Data

Let us first load relevant libraries and data.

```
##Load the spatio-temporal package
library(SpatioTemporal)
##And additional packages for plotting
library(plotrix)

##load data
data(mesa.data.model)
##...and optimisation results
data(mesa.data.res)
```

C.2 Running the MCMC

In addition to the standard model fitting described in subsection 7.1 the model (and parameter uncertainties) can be estimated using Metropolis-Hastings.

Here we run an the MCMC starting at the mode found in subsection 7.1 and using a proposal matrix based on the Hessian, as suggested in Roberts et al. (1997).

```
##Extract parameters
par <- mesa.data.res$par.est$res.best$par.all
##and Hessian
H <- mesa.data.res$par.est$res.best$hessian.all
##parameter uncertainty
par.sd <- sqrt(diag(solve(-H)))

if(FALSE){
  ##run the MCMC, this may take a while...
  MCMC.res <- run.MCMC(par, mesa.data.model, N = 5000,
                       Hessian.prop = H, silent = FALSE)
}else{
  ##Get the precomputed results instead.
  MCMC.res <- mesa.data.res$MCMC.res
}
```

C.3 Results

Having run the MCMC algorithm for the model we now investigate the results.

C.4 Studying the Results

We start by looking at the components of the result structure, and some summaries of the results.

```
##components of the MCMC results
names(MCMC.res)

##The acceptance probability (alpha) for each step
##in the Metropolis-Hastings algorithm.
summary(MCMC.res$acceptance)

##The MCMC-estimated parameters
summary(MCMC.res$par)
```

C.4.1 Plotting the Results

Having studied the elements of the result structure we now plot the parameter tracks and MCMC estimates of the parameter densities.

```
##MCMC tracks for four of the parameters
par(mfrow=c(4,1),mar=c(2,2,2.5,.5))
for(i in c(4,9,13,15)){
  plot(MCMC.res$par[,i], ylab="", xlab="", type="l",
       main=colnames(MCMC.res$par)[i])
}

##And estimated densities for the log-covariance parameters.
##The red line is the approximate normal distribution given by
##the maximum-likelihood estimates, e.g. ML-estimate and
##standard deviation from the observed information matrix.
```

```
par(mfrow=c(3,3),mar=c(4,4,2.5,.5))
for(i in 9:17){
  x <- sort(unique(MCMC.res$par[,i]))
  y <- dnorm(x, mean=par[i],sd=par.sd[i])
  dens <- density(MCMC.res$par[,i])
  plot(dens,ylim=c(0,max(c(dens$y,y))),main=names(par)[i])
  lines(x,y,col=2)
}
```

The large uncertainties (and bad mixing) for some of the log-covariance parameters are not unexpected. Recall that we only have 25 locations to base the estimates of the range and sill for the β-fields on (see subsection 7.2). For the residual ν-fields, on the other hand, the estimates are essentially based on $T = 280$ replicates of the residual field; implying that the estimates of range, sill and nugget for the residual field are much more certain.

D Modelling with a Spatio-Temporal covariate

The following is an example of modelling with a spatio-temporal covariate.

D.1 Load Data

Let us first load relevant libraries and data.

```
##Load the spatio-temporal package
library(SpatioTemporal)
##And additional packages for plotting
library(plotrix)

##load data
data(mesa.data)
data(mesa.data.model)
##...and optimisation results
data(mesa.data.res)
```

D.2 Setup and Study the Data

Then we setup the data structures, creating a new mesa.data.model that contains the spatio-temporal covariate.

```
##create model structure with ST-covariate
mesa.data.model.ST <- create.data.model(mesa.data,
    LUR = mesa.data.model$LUR.list,
    ST.Ind="lax.conc.1500")

##An alternative is to seperate the spatio-temporal
##covariate into an average over time, and a mean-zero
##spatio-temporal covariate.
##First we create a new data object
mesa.data.mean0 <- remove.ST.mean(mesa.data)

##with mean-zero spatio-temporal covariate.
colMeans(mesa.data.mean0$SpatioTemp)

##The mean has been added as a geographic covariate
```

```
names(mesa.data.mean0$LUR)

##create model structure with mean-zero ST-covariate
mesa.data.model.ST.mean0 <- create.data.model(mesa.data.mean0,
    LUR = list(c("log10.m.to.a1", "s2000.pop.div.10000",
                 "km.to.coast", "mean.lax.conc.1500"),
               "km.to.coast", "km.to.coast"),
    ST.Ind="lax.conc.1500")

##note that the models have different number of covariates
dim <- loglike.dim(mesa.data.model)
dim.ST <- loglike.dim(mesa.data.model.ST)
dim.ST0 <- loglike.dim(mesa.data.model.ST.mean0)

##number of spatio-temporal covariates
c(dim$L, dim.ST$L, dim.ST0$L)

##number of geogrpahic covariates for the
##two temporal trends+intercept
cbind(dim$p, dim.ST$p, dim.ST0$p)

##The models require the same number of log-covariance
##parameters (same number of beta-fields)
c(dim$nparam.cov, dim.ST$nparam.cov, dim.ST0$nparam.cov)

##... but different number of regression parameters
c(dim$nparam, dim.ST$nparam, dim.ST0$nparam)
```

D.3 Parameter Estimation

Given the two models that contain spatio-temporal covariates we estimate parameters for the models (or load precomputed results).

```
##create initial values
dim <- loglike.dim(mesa.data.model.ST)
x.init <- c(rep(c(1,-3),dim$m+1),-3)

##estimate parameters
if(FALSE){
  ##This may take a while...
  par.est.ST <- fit.mesa.model(x.init, mesa.data.model.ST,
        hessian.all=TRUE, control=list(trace=3,maxit=1000))

  par.est.ST.mean0 <- fit.mesa.model(x.init,
        mesa.data.model.ST.mean0, hessian.all=TRUE,
        control=list(trace=3,maxit=1000))
}else{
  ##Get the precomputed optimisation results instead.
  par.est.ST <- mesa.data.res$par.est.ST
  par.est.ST.mean0 <- mesa.data.res$par.est.ST.mean0
}
##and the reference model
par.est <- mesa.data.res$par.est
```

D.4 Predictions

Having created two models and estimated parameters we are now ready to compute some predictions for the models.

```
##extract the estimated parameters
x.ST <- par.est.ST$res.best$par.all
x.ST0 <- par.est.ST.mean0$res.best$par.all

##as well as parameters for the model without a
##spatio-temporal covariate
x <- par.est$res.best$par.all

##compute some predictions for these three models
EX <- cond.expectation(x, mesa.data.model,
                       compute.beta = TRUE)
EX.ST <- cond.expectation(x.ST, mesa.data.model.ST,
                          compute.beta = TRUE)
EX.ST.0 <- cond.expectation(x.ST0, mesa.data.model.ST.mean0,
                            compute.beta = TRUE)
```

D.5 Results

Having estimated the models and computed predictions we now want to investigate the results.

D.5.1 Estimation Results

We start by looking at estimated parameters for the two different cases with spatio-temporal covariates, and compare to the model without spatio-temporal covariates.

```
##first that both optimisations have converged
par.est.ST$message
par.est.ST.mean0$message

##extract the estimated parameters
x.ST <- par.est.ST$res.best$par.all
x.ST0 <- par.est.ST.mean0$res.best$par.all
x <- par.est$res.best$par.all

##and parameter uncertainties
x.ST.sd <- sqrt(diag(-solve(par.est.ST$res.best$hessian.all)))
x.ST0.sd <-
  sqrt(diag(-solve(par.est.ST.mean0$res.best$hessian.all)))
x.sd <- sqrt(diag(-solve(par.est$res.best$hessian.all)))

##plot the estimated parameters
par(mfrow=c(1,1),mar=c(13.5,2.5,.5,.5))
plotCI(1:19, x.ST0, uiw=1.96*x.ST0.sd,
       ylab="", xlab="", xaxt="n")
plotCI(c(1:5,7:19)-.2, x.ST, uiw=1.96*x.ST.sd,
       add=TRUE, col=2)
plotCI(c(2:5,7:19)+.2, x, uiw=1.96*x.sd,
       add=TRUE, col=3)
abline(h=0, col="grey")
axis(1,1:length(x.ST0),names(x.ST0),las=2)
legend("bottomleft", col = c(1,2,3), pch = 1,
       legend=c("par.est.ST","par.est.ST.mean0","par.est.ST"))
```

In this case the including of a spatio-temporal covariate hardly affects the parameter estimates at all. It should be noted that this spatio-temporal covariate has been included in the data *to provide an example* of how a covariate can be used by the model.

D.5.2 Prediction Results

Having studied the estimated parameters we now investigate the resulting predictions.

```
##Start by plotting predictions and observations
##for 1 locations, for the three models
par(mfrow=c(3,1),mar=c(2.5,2.5,2,.5))
plotPrediction(EX, "60590001", mesa.data.model)
plotPrediction(EX.ST, "60590001", mesa.data.model)
plotPrediction(EX.ST.0, "60590001", mesa.data.model)
```

```
##Start by plotting predictions and observations
##for 3 observed locations
par(mfrow=c(3,1),mar=c(2.5,2.5,2,.5))
plotPrediction(E.AQS, "60371103", mesa.data.model)
plotPrediction(E.AQS, "60371601", mesa.data.model)
plotPrediction(E.AQS, "60590001", mesa.data.model)

##Due to the minimal influence of the spatio-temporal
##covariate there is little difference between the
##different predictions, both for the observations
range(EX$EX-EX.ST$EX)
range(EX$EX-EX.ST.0$EX)

##...and for the latent beta-fields
range(EX$EX.beta-EX.ST$EX.beta)
range(EX$EX.beta-EX.ST.0$EX.beta)
```

E Simulation

Another option for evaluating model behaviour is to use simulated data. Instead of using actual observations and comparing predictions to actual observations, we simulate log NO_x observations using the same model as in section 7, and comparing predictions to the simulated data.

E.1 Load Data

Let us first load relevant libraries and data.

```
##Load the spatio-temporal package
library(SpatioTemporal)
##And additional packages for plotting
library(plotrix)
library(maps)

##load the data model (same as before)
data(mesa.data)
data(mesa.data.model)
##...and optimisation results
data(mesa.data.res)
```

E.2 Simulating some Data

First we simulate 4 samples of new data, using the parameters previously estimated in subsection 7.1.

```
##Extract parameters
x <- mesa.data.res$par.est$res.best$par

##And simulate new data using previously estimated parameters
sim.data <- simulateMesaData(x,  mesa.data.model, rep=4)

##examine the result
names(sim.data)
str(sim.data,1)

##Here sim.data$X contains the 4 simulations, sim.data$B
##contains the simulated beta fields and sim.data$obs
```

```
##contains observations data.frames that can be used to
##replace mesa.data.model$obs.

##lets create model structures that contain the simulated data
mesa.data.sim <- list()
for(i in 1:length(sim.data$obs)){
  ##copy the mesa.data.model object
  mesa.data.sim[[i]] <- mesa.data.model
  ##replace observations with the simulated data
  mesa.data.sim[[i]]$obs <- sim.data$obs[[i]]
}

##Compute predictions for the 4 simulated datasets.
##Here we'll just use the known parameters, however one could
##easily estimate new parameters based on the simulated data
##using fit.mesa.model (although this would take more time)

##Please note that following the predictions take roughly
##3 minutes on a decent laptop.
E <- list()
for(i in 1:length(sim.data$obs)){
  E[[i]] <- cond.expectation(x, mesa.data.sim[[i]],
                             compute.beta = TRUE)
}
```

E.3 Studying the Results

Given simulated datasets and predictions based on the simulated data we study how well the estimates agree with the simulated data.

```
##First we compare the simulated, known values of the first
##beta-field with the predictions.
par(mfrow=c(2,2),mar=c(4.5,4.5,2,.5))
for(i in 1:4){
  ##plot confidence intervalls for the predicted data
  plotCI(E[[i]]$EX.beta[,"const"],
         uiw = 1.96*sqrt(E[[i]]$VX.beta[,"const"]),
         ylab="Beta-field", xlab="Locations",
         main=sprintf("Simulation set %d",i))
  ##and add the known values from the simulation.
  points(sim.data$B[,"const",i],col="red",pch=19,cex=.5)
```

```
}

##Let's compare the predicted values and the simulated data for
##all four simulations at three different sites
##First site
par(mfrow=c(2,2),mar=c(2.5,2.5,2,.5))
for(i in 1:4){
  ##plot predictions, but not the observations
  plotPrediction(E[[i]], "60590001", mesa.data.sim[[i]],
                 lty=c(1,NA))
  ##add the simulated data (i.e. observations +
  ##simulated values at points where we've predicted)
  lines(as.Date(rownames(sim.data$X)),
        sim.data$X[,"60590001",i], col="red")
}

##Second site
par(mfrow=c(2,2),mar=c(2.5,2.5,2,.5))
for(i in 1:4){
  plotPrediction(E[[i]], "60371602", mesa.data.sim[[i]],
                 lty=c(1,NA))
  lines(as.Date(rownames(sim.data$X)),
        sim.data$X[,"60371602",i], col="red")
}

##Third site
par(mfrow=c(2,2),mar=c(2.5,2.5,2,.5))
for(i in 1:4){
  plotPrediction(E[[i]], "L002", mesa.data.sim[[i]],
                 lty=c(1,NA))
  lines(as.Date(rownames(sim.data$X)),
        sim.data$X[,"L002",i], col="red")
}

##Finally we also compare the predicted long term average
##at each site with the average of the simulated data.
par(mfrow=c(2,2),mar=c(4.5,4.5,2,.5))
for(i in 1:4){
  plot(apply(sim.data$X[,,i],2,mean),apply(E[[i]]$EX,2,mean),
       xlab="Simulated data", ylab="Predictions",
       main = sprintf("Long term average for simulation %d",i))
  abline(0,1,col="grey")
}
```

E.4 Simulation at Unobserved Locations

Then we setup the data structures — dropping some observations to simulate a case with observed and unobserved locations — and study the available data.

```
##store the original data structure
mesa.data.org <- mesa.data

##keep only observations from the AQS sites
##This gives us 5 "unobserved" sites at which to predict
ID.AQS <- mesa.data$location$ID[mesa.data$location$type=="AQS"]
mesa.data$obs <- mesa.data$obs[mesa.data$obs$ID %in% ID.AQS,]

##study the reduced data structure, we see
##that the 5 FIXED sites lack observations.
printMesaDataNbrObs(mesa.data)
##compate this to the original data.
printMesaDataNbrObs(mesa.data.org)

##create an object containing only the unmonitored sites
mesa.unmon <- mesa.data
ID.miss <- !(mesa.unmon$location$ID %in%
             unique(mesa.unmon$obs$ID))
mesa.unmon$location <- mesa.unmon$location[ID.miss,]
mesa.unmon$LUR <- mesa.unmon$LUR[mesa.unmon$location$ID,]
mesa.unmon$SpatioTemp <-
    mesa.unmon$SpatioTemp[,mesa.unmon$location$ID,,drop=FALSE]
##drop observations from the unmonitored data
mesa.unmon$obs <- NULL

##study the data structure for the unobserved locations
printMesaDataNbrObs(mesa.unmon)

##create a model object, dropping unobserved locations.
##Dropping the unobserved locations will speed up the parameter
##estimations and is thus often a good idea.
mesa.data.model <- create.data.model(mesa.data,
  LUR = mesa.data.model$LUR.list,
  ST.Ind = mesa.data.model$ST.Ind)

##note that this drops unobserved sites from the data structure
printMesaDataNbrObs(mesa.data.model)

##And now simulate new data using previously
##estimated parameters
sim.data.obs <- simulateMesaData(x, mesa.data.model)
sim.data.unobs <- simulateMesaData(x, mesa.data.model,
                                   mesa.data=mesa.unmon)
sim.data.all <- simulateMesaData(x, mesa.data.model,
      mesa.data=mesa.unmon, combine.data=TRUE)

##This results in simulations at
##All sites
colnames(sim.data.all$X)
##... only observed sites
colnames(sim.data.obs$X)
##... and only unobserved sites
colnames(sim.data.unobs$X)

##note that the unobserved sites do not have a $obs vector
sim.data.unobs$obs
```