Comprehensive Tutorial for the Spatio-Temporal R-package

Silas Bergen
University of Washington

Johan Lindström
University of Washington
Lund University

22nd June 2018

Contents

1 Introduction

The aim of this tutorial is to provide detailed descriptions of funtion outputs and features not covered (or covered only briefly) in the introductory tutorial `vignette("ST_intro", package="SpatioTemporal")`. The reader is encouraged to first study that tutorial.

As always this tutorial can be accesed from R with `vignette("ST_tutorial", package="SpatioTemporal")` and the R-code can be found as `edit( vignette("ST_tutorial", package="SpatioTemporal") )` or `Stangle( vignette("ST_tutorial", package="SpatioTemporal")$file )`

The remainder of this introduction provides some brief comments on common problems that may arise when using the package (subsection 1.1) followed by overviews of the data used in the examples of this tutorial (subsection 1.2) and the theory behind the model (subsection 1.3). Following this background the actual R-tutorial begins in section 2 with an overview of the data structures used by the package to encapsulate data for the model fitting. Functions that do parameter estimation and prediction are introduced in section 4, along with tools for illustration of the results. The last part of the R-tutorial is a cross-validation example in section 5.

The Appendices contain commented code for some additional examples: Appendix A gives an example of predictions at unobserved locations and times, Appendix B provides the outlines of a simulation study, and an MCMC example is given in Appendix C.

1.1 Common Problems — Troubleshooting

Before starting with the full tutorial it seems prudent to discuss some of the common problems that might arise when using the package, along with possible solutions.

If the parameter estimation fails consider:

- Covariate scaling: avoid covariates with *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 points 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 list or 3D-array*.

1.2 Data

The data used in this tutorial consists of a subset of the 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, 2013) and a brief descriptions is given in `vignette("ST_intro", package="SpatioTemporal")`.

1.2.1 NO_x Observations

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

1.2.2 Geographic Covariates

To aid in the prediction at times and locations where we have no measurements a set of spatial and/or spatio-temporal covariates can be used. Covariates included in this example include both geographic covariates — such as 1) distance to a major roads; 2) distance to coast (truncated to be $\leq$15km); and 3) average population density in a 2 km buffer — and a spatio-temporal containing predictions from a deterministic air-pollution model — Caline3QHC (EPA, 1992, Wilton et al., 2010, MESA Air Data Team, 2010).

The covariates, and covariate selection, is described in detail by Mercer et al. (2011), Cohen et al. (2009), and briefly in `vignette("ST_intro", package="SpatioTemporal")`.

1.3 Model and Theory

The model and theory is described in `vignette("ST_intro", package="SpatioTemporal")` or (Szpiro et al., 2010, Sampson et al., 2011, Lindström et al., 2011, 2013) and the reader is referred to those papers for extensive details. We will inly give a very brief overview here.

Denoting the quantity to be modelled (in this example ambient 2-week average log 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 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) temporal basis functions, with $f_1(t) \equiv 1$; and the $\beta_i(s)$ are spatially varying coefficients for the temporal functions.

The $\beta_i(s)$-coefficients in (2) are treated as spatial fields with a universal kriging structure, allowing the temporal structure to vary between locations:

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

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. The X_i matrices often contain geographical covariates and we dentote this component a "land use" regression (LUR). This structure allows for different covariates and covariance structures in the each of the $\beta_i(s)$ fields; the fields are assumed to be apriori independent of each other.

The residual space-time field, $\nu(s,t)$, is assumed to be independent in time

with stationary parametric spatial covariance

$$\nu(s,t) \in \mathsf{N}\left(0, \underbrace{\begin{bmatrix} \Sigma_\nu^1(\theta_\nu) & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & \Sigma_\nu^T(\theta_\nu) \end{bmatrix}}_{\Sigma_\nu(\theta_\nu)}\right), \tag{4}$$

Here the size of each block matrix, $\Sigma_\nu^t(\theta_\nu)$, is the number of observations, n_t, at each time-point.

1.3.1 Model parameters

The parameters of the model consist of the regression parameters for the geographical ($\boldsymbol{\alpha} = (\alpha_1^\top, \ldots, \alpha_m^\top)^\top$) and spatio-temporal covariates ($\boldsymbol{\gamma} = (\gamma_1, \ldots, \gamma_L)^\top$); and covariance parameters for the β_i- and ν-fields

$$\theta_B = (\theta_1, \ldots, \theta_m) \quad \text{and} \quad \theta_\nu.$$

2 Preliminaries

Some of the code in this tutorial takes considerable time to run, in these cases precomputed results have been included in the package as 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 subsection 1.2 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:

```
> library(SpatioTemporal)
> library(Matrix)
> library(plotrix)
> library(maps)
```

2.1 The STdata object

The basic S3-object in this package, collecting covariates and observations, is a an STdata-object. In the following an STdata-object will be created from the data, thereafter the structure and the components of the object are described.

2.1.1 Creating an STdata object from raw data

The data used in this example are contained in data(mesa.data.raw), which we load and examine.

```
> data(mesa.data.raw, package="SpatioTemporal")
> str(mesa.data.raw,1)

List of 3
 $ X            :'data.frame':        25 obs. of  12 variables:
 $ obs          : num [1:280, 1:25] 4.58 3.89 4.01 4.08 3.73 ...
  ..- attr(*, "dimnames")=List of 2
 $ lax.conc.1500: num [1:280, 1:25] 2.32 1.84 1.49 2.59 1.9 ...
  ..- attr(*, "dimnames")=List of 2
```

As we can see mesa.data.raw consists of a list with two matrices and one data.frame; these contain the observations ("obs"), geographic covariates ("X") and spatio-temporal covariates ("lax.conc.1500") of the example.

We will use the createSTdata() function to create the STdata object. The createSTdata() function requires (at least) two arguments: obs and covars. Spatio-temporal covariates can be supplied through the optional argument SpatioTemporal. An example of possible input for the covars argument is given by the X data frame of mesa.data.raw:

```
> 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 log10.m.to.road
1      2.861509      4.100755      2.494956        2.494956
2      3.461672      3.801059      2.471498        2.471498
3      2.561133      3.695772      1.830197        1.830197
4      3.111413      2.737527      2.451927        2.451927
5      2.762193      3.687412      2.382281        2.382281
6      2.760931      4.035977      1.808260        1.808260
  km.to.coast s2000.pop.div.10000
1   15.000000            1.733283
2   15.000000            1.645386
3   15.000000            6.192630
4    1.023311            2.088930
5    6.011075            7.143731
6   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, covariates from a GIS to be used in the LUR, monitor type, longitudes and latitudes. The `covars` argument of `createSTdata()` should, *at a minimum*, include coordinates and covariates for all locations. Observations are matched to the locations by matching the *columnames* of `obs` (see below) to 1) names given by a `ID` field in `covars`; 2) the rownames of `covars`; 3) names infered from the ordering of `covars`, see `stCheckCovars`.

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 denoted 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 fields: date, ID and obs. The format of mesa.data.raw$obs as a matrix is most convenient for data with few (or no) missing observations.

The final element is a spatio-temporal covariate, i.e. the output from the Caline3QHC model (see subsubsection 1.2.2),

```
> 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. Similar to the mesa.data.raw$obs matrix, the row- and column names of the mesa.data.raw$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 createSTdata() function. Given these minimal elements, creation of the STdata structure is easy:

```
> ##matrix of observations
> obs <- mesa.data.raw$obs
> ##data.frame/matrix of covariates
> covars <- mesa.data.raw$X
> ##list/3D-array with the spatio-temporal covariates
> ST.list <- list(lax.conc.1500=mesa.data.raw$lax.conc.1500)
> ##create STdata object
> mesa.data <- createSTdata(obs, covars, SpatioTemporal=ST.list,
                            n.basis=2)
```

A few things to note here: we must first convert the `mesa.data.raw$lax.conc.1500` spatio-temporal covariate matrix to a list (or 3D-array); the length of this list equals the number of spatio-temporal covariates we want to use (in this case, just 1). We also specified `n.basis=2`, which indicates we want to compute 2 temporal trends; for a discussion on how to determine suitable temporal trends (or basis functions) see Section 4.3 in `vignette("ST_intro", package="SpatioTemporal")`.

The resulting `STdata`-object contains a number of elements, described in the following Sections (2.1.2–2.1.6).

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

```
[1] "obs"            "covars"         "SpatioTemporal"
[4] "trend"          "trend.fnc"
```

2.1.2 The `mesa.data$covars` Data Frame

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

```
> head(mesa.data$covars)
```

```
        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 log10.m.to.road
1      2.861509      4.100755      2.494956        2.494956
2      3.461672      3.801059      2.471498        2.471498
3      2.561133      3.695772      1.830197        1.830197
4      3.111413      2.737527      2.451927        2.451927
5      2.762193      3.687412      2.382281        2.382281
6      2.760931      4.035977      1.808260        1.808260
  km.to.coast s2000.pop.div.10000
1   15.000000            1.733283
2   15.000000            1.645386
3   15.000000            6.192630
4    1.023311            2.088930
```

```
5    6.011075            7.143731
6   15.000000            4.766780
```

The covars data frame is a 25×12 data frame. The first field contains the ID, or names, for each of the 25 locations, this is the only *mandatory* field in covars and will be added by createSTdata if missing; the second and third fields contain x- and y-coordinates, which are used to calculate distances between locations. The following fields contain longitude and latitude coordinates; a field describing the type of monitoring system to which each location belongs; and LUR covariates. In this example, the LUR covariates are $\log_{10}$ meters to A1, A2, A3 roads and the minimum of these three measurements; kilometres to the coast; and average population density in a 2 km buffer (divided by 10,000).

In addition to the ID-field the type-field is also special; when it exists it is used to seperate different types of observtions locations (used by e.g. the summary and plot functions). If included, this field should contain factors or strings. 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 locations.

Although we have observations at all the locations in this example, one could also include locations in mesa.data$covars that do not have observations in order to predict at those locations (see Appendix A for a prediction example).

The following code plots these locations on a map, shown in Figure 1.

```
> ###Plot the locations, see Figure 1
> par(mfrow=c(1,1))
> plot(mesa.data$covars$long, mesa.data$covars$lat,
       pch=24, bg=c("red","blue")[mesa.data$covars$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"))
```

2.1.3 The mesa.data$trend Data Frame

Next, look at mesa.data$trend and mesa.data$trend.fnc:

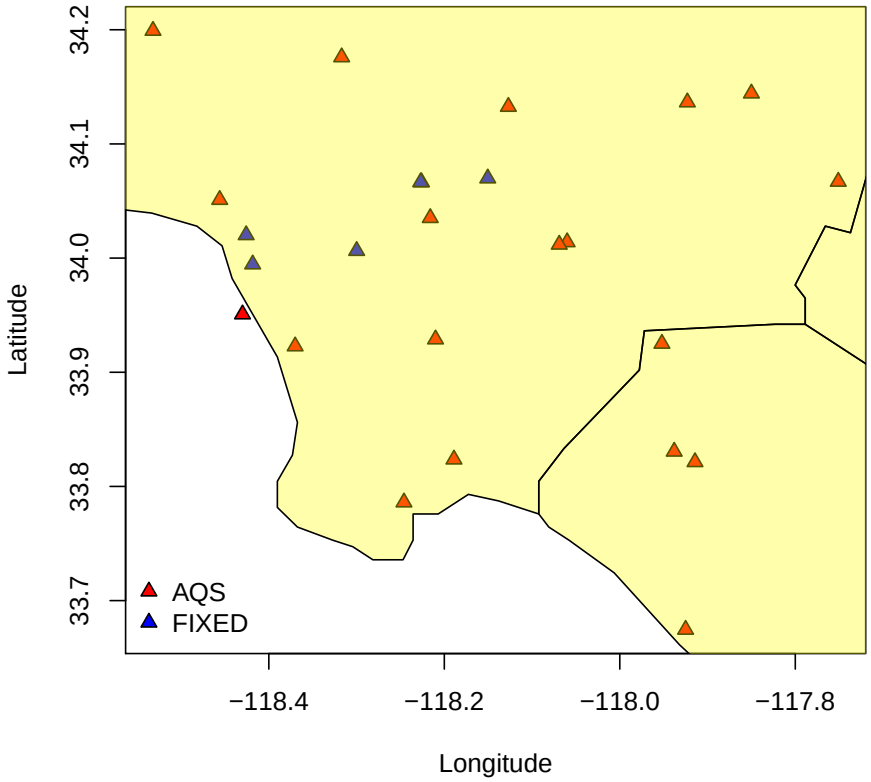

Figure 1: Location of monitors in the Los Angeles area.

```
> head(mesa.data$trend)

           V1          V2       date
1 -1.8591693 1.20721096 1999-01-13
2 -1.5200057 0.90473775 1999-01-27
3 -1.1880840 0.62679098 1999-02-10
4 -0.8639833 0.38411634 1999-02-24
5 -0.5536476 0.19683161 1999-03-10
6 -0.2643623 0.08739755 1999-03-24

> head(mesa.data$trend.fnc)

1 function (x = date.ind)
2 {
3     X.comps <- matrix(NA, length(x), length(spline))
```

```
4     for (i in 1:length(spline)) {
5         X.comps[, i] <- scale(predict(spline[[i]], as.double(x))$y,
6             center = scale.spline[[i]][1], scale = scale.spline[[i]][2])
```

The trend data frame consists of 2 smooth temporal basis functions computed using singular value decomposition (SVD). 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. Additionaly the functions used to compute the smooth trends are stored in trend.fnc and can be used to compute temporal trends at additional time-points, for observed time points trend.fnc returns elements in trend.

```
> cbind(mesa.data$trend.fnc(mesa.data$trend$date[1:5]),
        mesa.data$trend[1:5,])
                   V1        V2         V1        V2
1999-01-13 -1.8591693 1.2072110 -1.8591693 1.2072110
1999-01-27 -1.5200057 0.9047378 -1.5200057 0.9047378
1999-02-10 -1.1880840 0.6267910 -1.1880840 0.6267910
1999-02-24 -0.8639833 0.3841163 -0.8639833 0.3841163
1999-03-10 -0.5536476 0.1968316 -0.5536476 0.1968316
                 date
1999-01-13 1999-01-13
1999-01-27 1999-01-27
1999-02-10 1999-02-10
1999-02-24 1999-02-24
1999-03-10 1999-03-10
```

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.

2.1.4 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 3.889091 1999-01-27 60370002
3 4.013020 1999-02-10 60370002
4 4.080691 1999-02-24 60370002
5 3.728085 1999-03-10 60370002
6 3.751913 1999-03-24 60370002
```

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 NO_x concentrations; `date` — the date of each observation; and `ID` — labels indicating at which monitoring location each measurement was taken. Details regarding the monitoring can be found in Cohen et al. (2009), and a brief introduction is given in subsection 1.2.

The ID values should correspond to the ID of the monitoring locations given in `mesa.data$covars$ID`. The dates in `mesa.data$obs` should correspond to dates in `mesa.data$trend$date`; although as for `mesa.data$covars$ID` additional, unobserved dates, are allowed in `mesa.data$trend$date`.

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.

2.1.5 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,,drop=FALSE]

, , lax.conc.1500

           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$SpatioTemp` element should be a *three dimensional array* containing spatio-temporal covariates. In this example dataset we have only one covariate, which is the output from the Caline3QHC model, see subsection 1.2. If *no* spatio-temporal covariates are used `mesa.data$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 dimnames of the `SpatioTemp` array are used to match covariates with observations, locations, and time-points

```
> str(dimnames(mesa.data$SpatioTemp))

List of 3
 $ : chr [1:280] "1999-01-13" "1999-01-27" "1999-02-10" "1999-02-24" ...
 $ : 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$covars$ID`, and the names of the third dimension

```
> dimnames(mesa.data$SpatioTemp)[[3]]
```

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

identifies the different spatio-temporal covariates.

2.1.6 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:

```
> print(mesa.data)

STdata-object with:
        No. locations: 25 (observed: 25)
        No. time points: 280 (observed: 280)
        No. obs: 4577

Trend with 2 basis function(s):
[1] "V1" "V2"
with dates:
        1999-01-13 to 2009-09-23

12 covariate(s):
 [1] "ID"                  "x"
 [3] "y"                   "long"
 [5] "lat"                 "type"
 [7] "log10.m.to.a1"       "log10.m.to.a2"
 [9] "log10.m.to.a3"       "log10.m.to.road"
[11] "km.to.coast"         "s2000.pop.div.10000"

1 spatio-temporal covariate(s):
[1] "lax.conc.1500"

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 `FIXED` locations in the `mesa.data` structure. There are 20 AQS locations, which correspond to the number of locations marked as `AQS` in `mesa.data$covars$type`, and 5 `FIXED` locations, which correspond to the locations flagged as `FIXED` in `mesa.data$covars$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 locations (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 A 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))
> plot(mesa.data, "loc")
```

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 between different locations.

3 createSTmodel(): Specifying the Spatio-Temporal model

This section discusses how to specify everything we need to fit the Spatio-Temporal model. We need to specify the type of spatial covariance model to use for each β- and ν-field; define which covariates to use for each of β-fields; and specify any spatio-temporal covariates.

The function `createSTmodel()` is used to convert `STdata`-objects to `STmodel`-objects, by adding the covariance and covariate definitions. Geographic covariates for the β-fields are given by a list of formulas

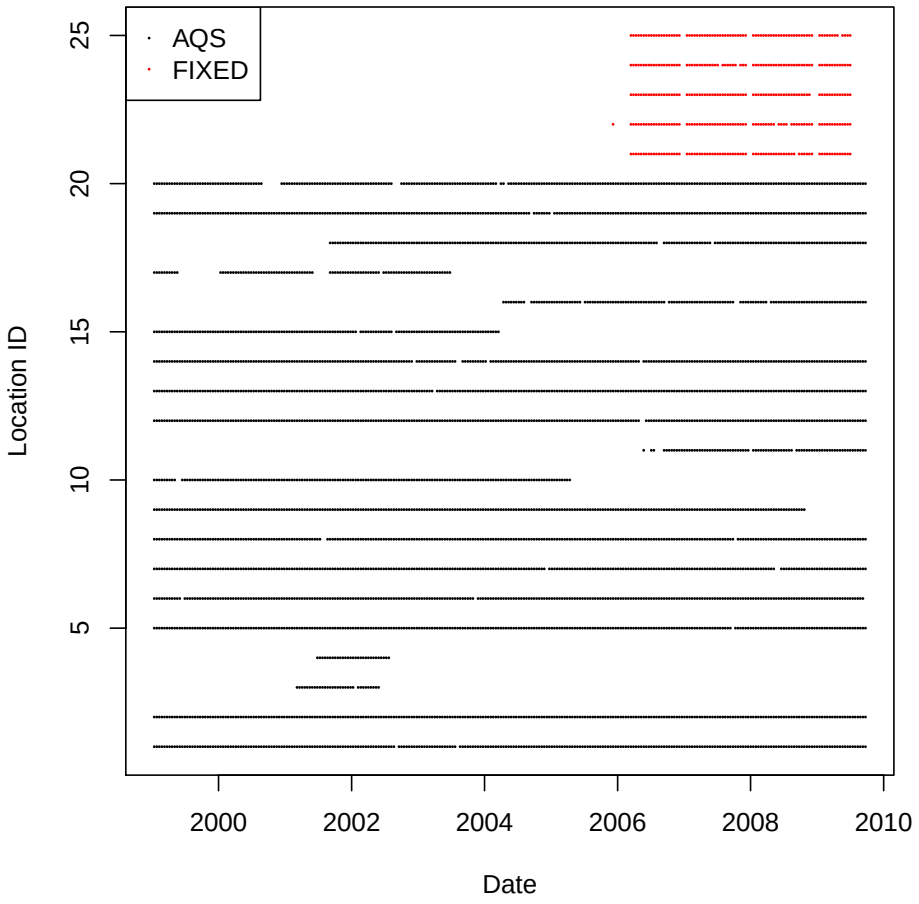

Figure 2: Space-time location of all our observations.

```
> LUR <-  list(~log10.m.to.a1+s2000.pop.div.10000+km.to.coast,
                ~km.to.coast, ~km.to.coast)
```

and the covariace models are given by two lists, with elements that specify covariance function, nugget, etc.

```
> cov.beta <- list(covf="exp", nugget=FALSE)
> cov.nu <- list(covf="exp", nugget=~type)
```

The above code shows that we want to use three LUR variables to model the β_0-field, namely, $\log_{10}$ meters to A1 road, population density, and kilometers to coast. We will use only kilometer to coast to model the β_1- and β_2-fields. An exponential covariance is used for all β-fields and for the ν-field. The β-fields are assumed to have no nugget while the nugget in the ν-field is allowed

to vary between the two types AQS/FIXED of locations, see also Section 4.4 in vignette("ST_intro", package="SpatioTemporal").

Next we specify a list that links variable names in the STdata$covars data frame to locations:

```
> locations <- list(coords=c("x","y"), long.lat=c("long","lat"),
    others="type")
```

Here the coords are used to compute distances between observation locations; long.lat and others are additional fields in mesa.data$covars that we want included in the STmodel object.

We are now ready to construct a STmodel-object:

```
> mesa.model <- createSTmodel(mesa.data, LUR=LUR,
                              ST="lax.conc.1500",
                              cov.beta=cov.beta,
                              cov.nu=cov.nu,
                              locations=locations)
> print(mesa.model)

STmodel-object with:
        No. locations: 25 (observed: 25)
        No. time points: 280 (observed: 280)
        No. obs: 4577

Trend with 2 basis function(s):
[1] "V1" "V2"
with dates:
        1999-01-13 to 2009-09-23

Models for the beta-fields are:
$const
~log10.m.to.a1 + s2000.pop.div.10000 + km.to.coast

$V1
~km.to.coast

$V2
~km.to.coast
```

```
1 spatio-temporal covariate(s):
[1] "lax.conc.1500"

Covariance model for the beta-field(s):
        Covariance type(s): exp, exp, exp
        Nugget: No, No, No
Covariance model for the nu-field(s):
        Covariance type: exp
        Nugget: ~type
        Random effect: No
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
```

The above output summarizes the model specifications we've made: which LUR covariates we use to model each of the β-fields; exponential covariances with no nugget for each of the β-fields, and an exponential covariance with nugget depending on location for the ν-field.

Note there is quite a bit of flexibility in specification of the β-fields. We can specify different covariance models for each one, and allow some of them to have nuggets. Using the `updateCovf()` function, we can alter the covariance specification for an existing `STmodel`-object. In the following code we change the covariance functions to: an exponential for the β_0-field; a Gaussian/double exponential covariance for the β_1-field; and i.i.d. for the β_2-fields (i.e. only nugget). The help file for `namesCovFuns()` gives a description of the available covariance functions.

```
> cov.beta2 <- list(covf=c("exp","exp2","iid"),
                    nugget=c(FALSE,FALSE,TRUE))
> mesa.model2 <- updateCovf(mesa.model, cov.beta=cov.beta2)
> print(mesa.model2)
```

```
STmodel-object with:
        No. locations: 25 (observed: 25)
        No. time points: 280 (observed: 280)
        No. obs: 4577

Trend with 2 basis function(s):
[1] "V1" "V2"
with dates:
        1999-01-13 to 2009-09-23

Models for the beta-fields are:
$const
~log10.m.to.a1 + s2000.pop.div.10000 + km.to.coast

$V1
~km.to.coast

$V2
~km.to.coast

1 spatio-temporal covariate(s):
[1] "lax.conc.1500"

Covariance model for the beta-field(s):
        Covariance type(s): exp, exp2, iid
        Nugget: No, No, Yes
Covariance model for the nu-field(s):
        Covariance type: exp
        Nugget: ~type
        Random effect: No
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
```

4 Estimating the Model

We are now ready to fit the spatio-temporal model to data. since the estimation is described in Section of `vignette("ST_intro", package="SpatioTemporal")`, we focus here on the details of the output from the estimation functions.

4.1 Parameter Estimation

Before estimating the parameters, we can look at the important dimensions of the model:

```
> model.dim <- loglikeSTdim(mesa.model)
> str(model.dim)

List of 12
 $ T              : int 280
 $ m              : int 3
 $ n              : int 25
 $ n.obs          : int 25
 $ p              : Named int [1:3] 4 2 2
  ..- attr(*, "names")= chr [1:3] "const" "V1" "V2"
 $ L              : int 1
 $ npars.beta.covf: Named int [1:3] 2 2 2
  ..- attr(*, "names")= chr [1:3] "exp" "exp" "exp"
 $ npars.beta.tot : Named int [1:3] 2 2 2
  ..- attr(*, "names")= chr [1:3] "exp" "exp" "exp"
 $ npars.nu.covf  : int 2
 $ npars.nu.tot   : int 4
 $ nparam.cov     : int 10
 $ nparam         : int 19
```

`T` gives us the number of time points; `m` the number of β-fields; `n` the number of locations; `n.obs` the number of observed locations (equal to `n` in this case, since we have no unobserved locations); `p` a vector giving the number of regression coefficients for each of the β-fields; `L` the number of spatio-temporal

covariates. The rest of the output gives numbers of parameters by field, and total number of parameters. An important dimension is `nparam.cov`, which gives us the total number of covariance parameters. Since the regression coefficients are estimated by profile likelihood, in essence only the covariance parameters need starting values specified. We do this accordingly:

```
> x.init <- cbind(c( rep(2, model.dim$nparam.cov-1), 0),
                  c( rep(c(1,-3), model.dim$m+1), -3, 0))
```

Here each column of `x.init` contains a starting value for the optimisation process in estimating the MLE's of the 10 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., 2013, 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 using `loglikeSTnames`, which gives the order of variables in `x.init` and also tells us which of the parameters are logged. Specifying `all=FALSE` gives us only the covariance parameters.

```
> rownames(x.init) <- loglikeSTnames(mesa.model, all=FALSE)
> x.init
                               [,1] [,2]
log.range.const.exp               2    1
log.sill.const.exp                2   -3
log.range.V1.exp                  2    1
log.sill.V1.exp                   2   -3
log.range.V2.exp                  2    1
log.sill.V2.exp                   2   -3
nu.log.range.exp                  2    1
nu.log.sill.exp                   2   -3
nu.log.nugget.(Intercept).exp     2   -3
nu.log.nugget.typeFIXED.exp       0    0
```

We are now ready to estimate the model parameters!

WARNING: The following steps are time-consuming.

```
> est.mesa.model <- estimate(mesa.model, x.init,
                             type="p", hessian.all=TRUE)
```

ALTERNATIVE: Load pre-computed results.

```
> data(est.mesa.model, package="SpatioTemporal")
```

End of alternative

The function `estimate()` (strictly `estimate.STmodel`) estimates all the model parameters. Specifying `type="p"` indicates we want to maximize the profile likelihood. `hessian.all=TRUE` indicates we want the Hessian for *all* model parameters; if we leave this entry blank, the default will compute the Hessian for only the log-covariance parameters.

From the output, which is mainly due to the `R`-function `optim`, we see that the two optimisation consumed 99 and 68 function evaluations each and ended with the same value, 5748.563. 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 `estimate()` is included in the package as `loglikeST`, with `loglikeSTgrad` and `loglikeSTHessian` 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 at the troublesome parameter values.

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

```
> loglikeST(est.mesa.model$res.best$par, mesa.model)

[1] 5748.563

> est.mesa.model$res.best$value

[1] 5748.563
```

4.2 Evaluating the Results

The first step in evaluating the optimisation results is to study the message included in the output from `estimate()`, as well as the converged parameter values from the two starting points:

```
> print(est.mesa.model)
```

```
Optimisation for STmodel with 2 starting points.
  Results: 2 converged, 0 not converged, 0 failed.
  Best result for starting point 1, optimisation has converged

No fixed parameters.

Estimated parameters for all starting point(s):
                                          [,1]          [,2]
gamma.lax.conc.1500              0.0008975546  0.0009008652
alpha.const.(Intercept)          3.7402698246  3.7406058853
alpha.const.log10.m.to.a1       -0.2021288763 -0.2022633497
alpha.const.s2000.pop.div.10000  0.0402182221  0.0401923686
alpha.const.km.to.coast          0.0374363255  0.0374629689
alpha.V1.(Intercept)            -0.7429226257 -0.7411611359
alpha.V1.km.to.coast             0.0174017754  0.0172645957
alpha.V2.(Intercept)            -0.1292573245 -0.1281826343
alpha.V2.km.to.coast             0.0155467684  0.0154883000
log.range.const.exp              2.4245453422  2.4205527321
log.sill.const.exp              -2.7522202309 -2.7568455241
log.range.V1.exp                 2.9175969067  2.9484089871
log.sill.V1.exp                 -3.5231738064 -3.5086290882
log.range.V2.exp                 1.7816565861  1.8062154010
log.sill.V2.exp                 -4.6812486307 -4.6730245593
nu.log.range.exp                 4.3836003026  4.3839197279
nu.log.sill.exp                 -3.2126038230 -3.2123312163
nu.log.nugget.(Intercept).exp   -4.4121007058 -4.4118479775
nu.log.nugget.typeFIXED.exp      0.6766505805  0.6748262774

Function value(s):
[1] 5748.563 5748.561
```

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

The function `estimate()` 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 2nd derivative of the log-likelihood) to determine if the matrix is negative definite; indicating that the optimisation has found a (local) maximum.

Included in the output from `estimate()`

```
> names(est.mesa.model)

[1] "res.best" "res.all"  "summary"
```

is 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(est.mesa.model$res.best)

 [1] "par"         "value"       "counts"      "convergence"
 [5] "message"     "hessian"     "conv"        "par.cov"
 [9] "par.all"     "hessian.all"

> names(est.mesa.model$res.all[[1]])

[1] "par"         "value"       "counts"      "convergence"
[5] "message"     "hessian"     "conv"        "par.cov"
[9] "par.all"

> names(est.mesa.model$res.all[[2]])

[1] "par"         "value"       "counts"      "convergence"
[5] "message"     "hessian"     "conv"        "par.cov"
[9] "par.all"
```

we see that the results include several different fields, several of which are taken directly from the output of the `optim` function —

par The estimated log-covariance parameters.

value The value of the log-likelihood.

counts The number of function evaluations.

convergence and message Convergence information from `optim`.

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

hessian The Hessian of the profile log-likelihood, from `optim`

par.cov A data frame containing estimates, estimated standard errors, initial or fixed values depending on whether we estimated or fixed the various parameters (in this case, all were estimated), and t-statistics for the log-covariance parameters

par.all The same summary as `par.cov`, but for all the parameters of the model. The regression coefficents are computed using generalised least squares (See Lindström et al., 2013, for details.).

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

Refer back to the output from `print(est.mesa.model)`; we consider now the two columns of parameter estimates resulting from the two starting values. 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 parameter uncertaintie, we note large standard-deviations for the β-field covariance parameters.

```
> coef(est.mesa.model, pars="cov")[,1:2]

                                      par         sd
log.range.const.exp             2.4245453 0.59261495
log.sill.const.exp             -2.7522202 0.39343246
log.range.V1.exp                2.9175969 0.72547044
log.sill.V1.exp                -3.5231738 0.53958625
log.range.V2.exp                1.7816566 0.56450965
log.sill.V2.exp                -4.6812486 0.33496982
nu.log.range.exp                4.3836003 0.09642295
nu.log.sill.exp                -3.2126038 0.06162470
nu.log.nugget.(Intercept).exp -4.4121007 0.05636217
nu.log.nugget.typeFIXED.exp     0.6766506 0.11763234
```

This is due to the number of "observations" that go into estimating the β-field covariance parameters; there are only 25 locations. On the other hand, 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 $T = 280$ 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.

4.3 Predictions

Having estimated the model parameters we use `predict.STmodel` to compute the conditional expectations for different parts of the model.

WARNING: The following steps are time-consuming.

```
> pred.mesa.model <- predict(mesa.model, est.mesa.model,
                             pred.var=TRUE)
```

ALTERNATIVE: Load pre-computed results.

```
> data(pred.mesa.model, package="SpatioTemporal")
```

End of alternative

The results from `predict` contains the following elements

```
> names(pred.mesa.model)
```

```
[1] "opts"       "pars"       "beta"       "EX.mu"
[5] "EX.mu.beta" "EX"         "VX"         "VX.pred"
[9] "I"
```

described in detail by the `plot`-function

```
> print(pred.mesa.model)
```

```
Prediction for STmodel.

Regression parameters:
         0 Spatio-temporal covariate(s).
         8 beta-fields regression parameters in x$pars.
```

```
Regression parameters are assumed to be known and
        prediction variances do NOT include
        uncertainties in regression parameters.

Prediction of beta-fields, (x$beta):
List of 3
 $ mu: num [1:25, 1:3] 3.79 3.67 4.03 3.23 3.69 ...
  ..- attr(*, "dimnames")=List of 2
 $ EX: num [1:25, 1:3] 3.7 3.37 4.19 3.67 3.52 ...
  ..- attr(*, "dimnames")=List of 2
 $ VX: num [1:25, 1:3] 0.000186 0.000186 0.001112 0.002197 0.000186 ...
  ..- attr(*, "dimnames")=List of 2

Predictions for 280 times at 25 locations.
List of 3
 $ EX.mu     : num [1:280, 1:25] 4.82 4.62 4.43 4.25 4.08 ...
  ..- attr(*, "dimnames")=List of 2
 $ EX.mu.beta: num [1:280, 1:25] 4.39 4.24 4.1 3.97 3.86 ...
  ..- attr(*, "dimnames")=List of 2
 $ EX        : num [1:280, 1:25] 4.55 4 4.03 4.18 3.72 ...
  ..- attr(*, "dimnames")=List of 2

Variances have been computed.
List of 2
 $ VX     : num [1:280, 1:25] 0.00509 0.00505 0.00501 0.00499 0.00497 ...
  ..- attr(*, "dimnames")=List of 2
 $ VX.pred: num [1:280, 1:25] 0.0172 0.0172 0.0171 0.0171 0.0171 ...
  ..- attr(*, "dimnames")=List of 2
```

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:

opts Options used in the call to `predict`, or implicitly assumed.

pars The regression parameters in (2) and (3), computed using generalised least squares (see Lindström et al., 2013, for details).

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); this includes the conditional expectations of the β-fields,

$$\mu_\beta(s,t) = \sum_{l=1}^{L} \gamma_l \mathcal{M}_l(s,t) + \sum_{i=1}^{m} f_i(t) \mathsf{E}(\beta_i|\Psi, \text{observations}).$$

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

VX.pred The predictive conditional variance for the spatio-temporal process in `EX` (essentially `VX`, plus the nugget in the ν-field).

beta A structure containing reconstructions and uncertainties for the latent β-fields.

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

First we compare the β-fields computed by fitting each of the times series of observations to the smooth trends,

```
> beta <- estimateBetaFields(mesa.model)
```

with the β-fields obtained from the full model, see Figure 3.

```
> par(mfrow=c(2,2), mar=c(3.3,3.3,1.5,1), mgp=c(2,1,0), pty="s")
> for(i in 1:3){
    plotCI(x=beta$beta[,i], y=pred.mesa.model$beta$EX[,i],
           uiw=1.96*beta$beta.sd[,i], err="x",
           main=paste("Beta-field for f", i, "(t)", sep=""),
           xlab="Empirical estimate",
           ylab="Spatio-Temporal Model",
           pch=NA, sfrac=0.005, asp=1)
```

```
  plotCI(x=beta$beta[,i], y=pred.mesa.model$beta$EX[,i],
         uiw=1.96*sqrt(pred.mesa.model$beta$VX[,i]),
         add=TRUE, pch=NA, sfrac=0.005)
  abline(0, 1, col="grey")
}
```

We can see from Figure 3 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.

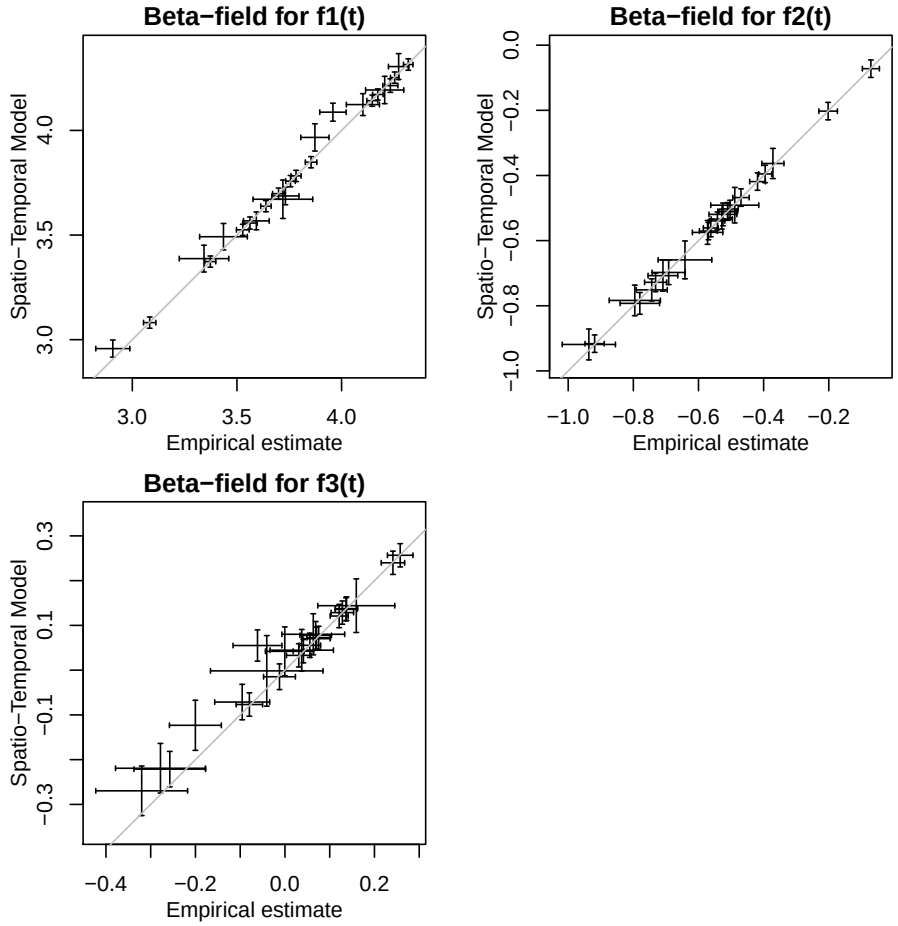

Figure 3: Comparing the two estimates of the β–field for the constant temporal trend and for the two smooth temporal trends.

In addition to predictions of the β-fields, `predict` also computes the conditional expectation at all the 280×25 space-time locations. As an example we study 4 of these locations, see Figure 4.

```
> par(mfrow=c(4,1),mar=c(2.5,2.5,2,.5))
> for(i in c(1,10,17,22)){
    plot(pred.mesa.model, ID=i, STmodel=mesa.model,
         col=c("black","red","grey"), lwd=1)
    plot(pred.mesa.model, ID=i, pred.type="EX.mu",
         col="green", lwd=1, add=TRUE)
    plot(pred.mesa.model, ID=i, pred.type="EX.mu.beta",
         col="blue", lwd=1, add=TRUE)
  }
```

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 4. The important thing to note here is that the predictions are computed as the conditional expectation of a *latent field* given observations. For unobserved locations this distinction *does not* matter, but for observed locations this implies smoothing over the nugget in the ν-fields resulting in $\mathsf{E}(x(s,t)|y(s,t)) \neq y(s,t)$, where $y(s,t)$ is an observations of the latent field, $x(s,t)$ at time t and locations s. Thus *predictions do not* coincides with 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 the observations.

5 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:

```
> Ind.cv <- createCV(mesa.model, groups=10, min.dist=.1)
> Ind.cv[1:10]
 [1]  1  3  8  2 10  4  3 10 10  4
```

Here `Ind.cv` is a 4577 vector defining the CV-groups. Each element of the vector indicates for which of the 10 CV-groups that the corresponding observation should be left out. For the i^{th} group, we are going to use our

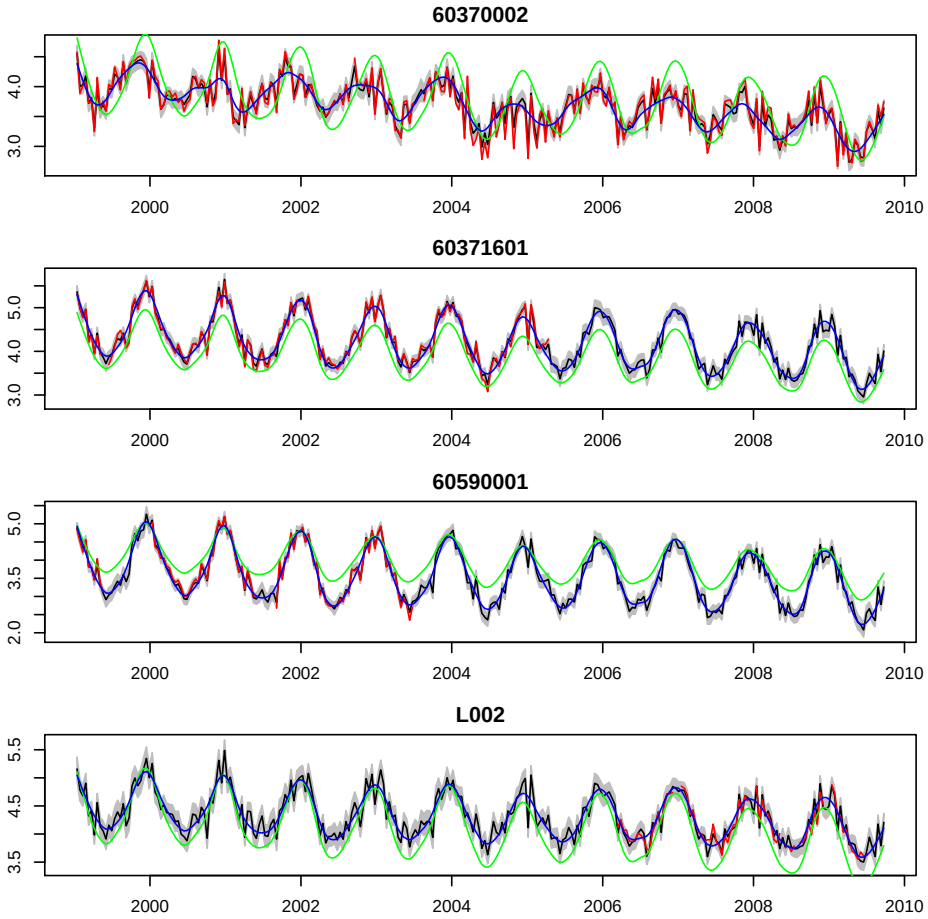

Figure 4: The predicted and observed data for 4 of the 25 locations. The red-lines denote observations, the black line and grey shading give predictions and 95% confidence intervals at unobserved time-points. The green and blue give the contribution to the predictions from the regression and regression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

model to predict at the observations marked by the number of that group (i.e. Ind.cv $= i$), using all other observations (i.e. Ind.cv $\neq i$). Once we have done this for each CV-group 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.

```
> table(Ind.cv)

Ind.cv
  1   2   3   4   5   6   7   8   9  10
```

```
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. If the groups are computed using a logical matrix (option `Icv.vector=FALSE`) instead of a vector, it is possible to combine the resulting CV-groups. As an example, group locations with more than 270 observations separately.

```
> n.obs <- table(mesa.model$obs$ID)
> ID1 <- names( n.obs[n.obs>270] )
> ID2 <- names( n.obs[n.obs<=270] )
> Ind.cv1 <- createCV(mesa.model, groups=10,
                      subset=ID1, Icv.vector=FALSE)
> Ind.cv2 <- createCV(mesa.model, groups=10,
                      subset=ID2, Icv.vector=FALSE)
```

Study the number of observations in each group for the two CV-grouping,

```
> colSums(Ind.cv1)

 [1] 275 279 278 277 277 280 277 279 278 278

> colSums(Ind.cv2)

 [1] 111 242 215 165 348 256 208  94  80  80
```

combine to one grouping,

```
> Ind.cv.final <- Ind.cv1 | Ind.cv2
> colSums(Ind.cv.final)

 [1] 386 521 493 442 625 536 485 373 358 358
```

and compare with the previous grouping.

```
> table(Ind.cv)
```

```
Ind.cv
  1   2   3   4   5   6   7   8   9  10
438 389 811 556 546 165 228 487 160 797

> ##easier if we sort by number of observations in each group
> rbind(sort(table(Ind.cv)), sort(colSums(Ind.cv.final)))

      9   6   7   2   1   8   5   4  10   3
[1,] 160 165 228 389 438 487 546 556 797 811
[2,] 358 358 373 386 442 485 493 521 536 625
```

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

```
> ID.cv <- sapply(split(mesa.model$obs$ID, Ind.cv),unique)
> print(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 locations in the 10^{th} group is due to the fact that `60371103` and `L001` are colocated.

```
> mesa.model$D.beta[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 locations in the 10^{th} group we see that the `60371103` 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 locations together, treating them as "one" location when creating the CV-grouping.

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

```
> I.col <- apply(sapply(ID.cv,
                        function(x) mesa.model$locations$ID
                        %in% x), 1,
                  function(x) if(sum(x)==1) which(x) else 0)
> names(I.col) <- mesa.model$locations$ID
> print(I.col)

60370002 60370016 60370030 60370031 60370113 60371002
       1        3        2        6        8        2
60371103 60371201 60371301 60371601 60371602 60371701
      10        4        3       10        9       10
60372005 60374002 60375001 60375005 60590001 60590007
       4        3        7        6        7        8
60591003 60595001     L001     L002    LC001    LC002
       5        5       10        1        1        9
   LC003
       2
```

Using this vector we can plot the locations on a map, colour-coded by which CV-group they belong to, see Figure 5.

```
> par(mfrow=c(1,1))
> plot(mesa.model$locations$long,
       mesa.model$locations$lat,
       pch=23+floor(I.col/max(I.col)+.5), bg=I.col,
       xlab="Longitude", ylab="Latitude")
> map("county", "california", col="#FFFF0055",
      fill=TRUE, add=TRUE)
```

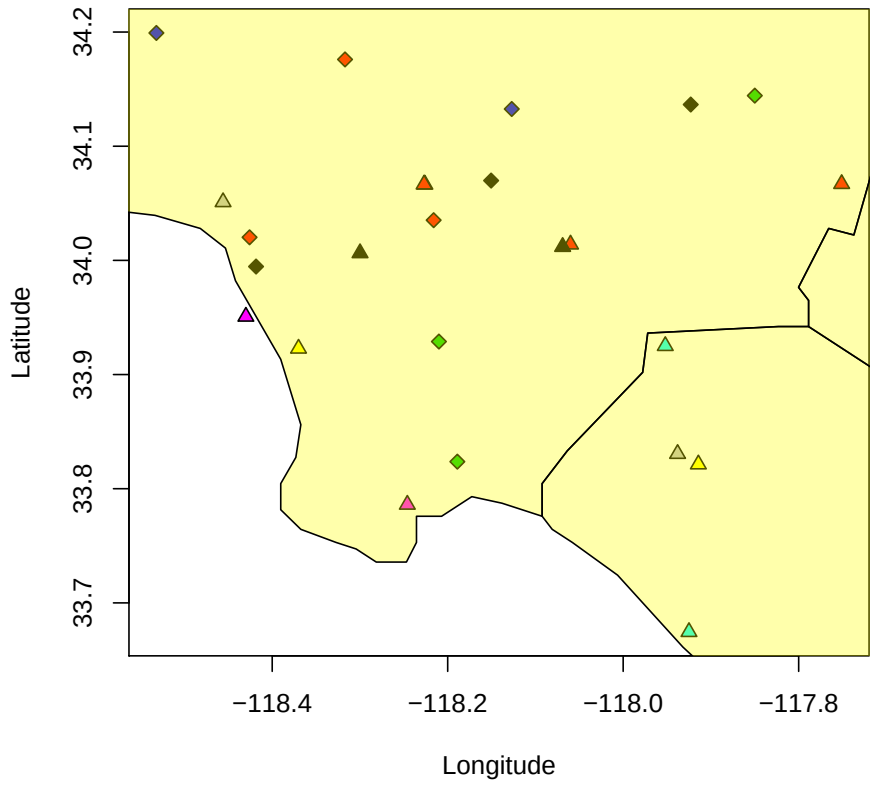

Figure 5: Location of monitors in the Los Angeles area. The different cross-validation groups are indicated by colour and shape of the points, i.e. all points of the same colour and shape belong to the same cross-validation group.

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 5.1 and 5.2 below.

5.1 Cross-Validated Estimation

Parameter estimation for each of the CV-groups is done by the `estimateCV` function, which calls `estimate.STmodel`. The inputs to `estimateCV` are similar to those of `estimate.STmodel`. As described in subsection 4.1 the estimation function require at least a `STmodel` and a matrix (or vector) of initial values, in addition to these `estimateCV` also requires a vector (or matrix) describing the CV-grouping, e.g. `Ind.cv`. Since the parameter estimation for 10 CV-groups using 2 initial values takes some considerable time the results have been pre-computed.

WARNING: The following steps are time-consuming.

```
> x.init <- coef(est.mesa.model, pars="cov")[,c("par","init")]
> est.cv.mesa <- estimateCV(mesa.model, x.init, Ind.cv)
```

ALTERNATIVE: Load pre-computed results.

```
> data(est.cv.mesa, package="SpatioTemporal")
```

End of alternative

We first study the results of the estimation.

```
> print(est.cv.mesa)

Cross-validation parameter estimation for STmodel
  with 10 CV-groups and 2 starting points.
  Results: 10 converged, 0 not converged.

No fixed parameters.

Estimated function values and convergence info:
      value convergence conv  eigen.min eigen.all.min
1  5185.641        TRUE TRUE 0.05786443            NA
2  5175.569        TRUE TRUE 1.56679419            NA
3  4699.504        TRUE TRUE 0.24456506            NA
```

```
4  4939.732        TRUE TRUE 0.95850244           NA
5  5002.363        TRUE TRUE 1.72437665           NA
6  5748.226        TRUE TRUE 0.17356445           NA
7  5423.787        TRUE TRUE 1.33945612           NA
8  5182.118        TRUE TRUE 0.90769121           NA
9  5513.730        TRUE TRUE 1.42290499           NA
10 4488.807        TRUE TRUE 1.05472374           NA
```

Here we see that the parameter estimates for all 10 groups have converged, it also gives the optimal log-likelihood value for each estimate along with convergence information and the smallest eigenvalue of the Hessian; very small eigenvalues would indicate that some parameters in that CV-group have large uncertainties. This information can also be obtained from `est.cv.mesa$status`.

The estimated parameters for each CV-group can be found in

```
> head( coef(est.cv.mesa) )
```

```
                                      [,1]          [,2]
gamma.lax.conc.1500             0.001318065 -0.0007460844
alpha.const.(Intercept)         3.786553974  4.0181346049
alpha.const.log10.m.to.a1      -0.221173875 -0.2860374441
alpha.const.s2000.pop.div.10000 0.033846619  0.0407220627
alpha.const.km.to.coast         0.041109873  0.0340335295
alpha.V1.(Intercept)           -0.712590209 -0.7532094907
                                      [,3]          [,4]
gamma.lax.conc.1500             0.003210611  0.0008843482
alpha.const.(Intercept)         3.687813713  3.6754117871
alpha.const.log10.m.to.a1      -0.171452696 -0.1809776106
alpha.const.s2000.pop.div.10000 0.024720783  0.0386846530
alpha.const.km.to.coast         0.040562327  0.0405810400
alpha.V1.(Intercept)           -0.774256633 -0.7313169921
                                      [,5]         [,6]
gamma.lax.conc.1500             0.001057007  0.00119386
alpha.const.(Intercept)         3.864372729  3.71435248
alpha.const.log10.m.to.a1      -0.211665468 -0.24307135
alpha.const.s2000.pop.div.10000 0.037559088  0.04270922
alpha.const.km.to.coast         0.032878863  0.04930008
alpha.V1.(Intercept)           -0.685680715 -0.78571072
                                      [,7]         [,8]
gamma.lax.conc.1500            -0.003488516  0.001383858
alpha.const.(Intercept)         3.857006261  3.745298736
```

```
alpha.const.log10.m.to.a1       -0.237677919 -0.211036625
alpha.const.s2000.pop.div.10000  0.038461583  0.047587969
alpha.const.km.to.coast          0.038261372  0.037393106
alpha.V1.(Intercept)            -0.751222333 -0.755570973
                                        [,9]        [,10]
gamma.lax.conc.1500              0.001417108  0.00214366
alpha.const.(Intercept)          3.705864017  3.51272830
alpha.const.log10.m.to.a1       -0.192903505 -0.14685413
alpha.const.s2000.pop.div.10000  0.042350283  0.04179399
alpha.const.km.to.coast          0.037603582  0.03313854
alpha.V1.(Intercept)            -0.766977363 -0.73636091
```

with uncertainties stored in `est.cv.mesa$par.cov.sd` and `est.cv.mesa$par.all.sd`.

If the option `verbose.res=TRUE` is used in the call to `estimateCV` the results of each `estimate.STmodel` call are stored in the `est.cv.mesa$res.all`.

5.2 Cross-Validated Prediction

Once parameters have been estimated `predictCV()` computes predictions for each of the CV-groups. This is done by computing the conditional expectations of the left-out observations, as indicated in by the columns in `Ind.cv`, given all other observations and the estimated parameters. Details can be found in (Lindström et al., 2013, Szpiro et al., 2010) or Section 4.6 of `vignette("ST_intro", package="SpatioTemporal")`

We now compute cross-validation predictions Gaussian, along with temporal averages based on only the observed time-points.

WARNING: The following steps are time-consuming.

```
> pred.cv.mesa <- predictCV(mesa.model, est.cv.mesa, LTA=TRUE)
```

ALTERNATIVE: Load pre-computed results.

```
> data(pred.cv.mesa, package="SpatioTemporal")
```

End of alternative

First we examine the results of the predictions:

```
> print(pred.cv.mesa)
```

```
Cross-validation prediction for STmodel with 10 CV-groups.
  Predictions for 25 locations and 280 time points.
  Temporal averages predicted for 25 locations.
  Variances have been computed.
        Regression parameters are assumed to be known,
        prediction variances do NOT include
        uncertainties in regression parameters.
```

```
> names(pred.cv.mesa)
```

```
[1] "opts"     "Ind.cv"   "pred.obs" "pred.LTA" "pred.all"
```

Here the `pred.obs` contains a data frame with observations, predictions, prediction variances, and residuals for each observed space-time location

```
> str(pred.cv.mesa$pred.obs)
```

```
'data.frame':        4577 obs. of  10 variables:
 $ obs       : num  4.58 4.13 4.73 5.35 5.28 ...
 $ date      : Date, format: "1999-01-13" ...
 $ ID        : chr  "60370002" "60370016" "60370113" "60371002" ...
 $ EX.mu     : num  4.85 4.77 4.89 4.95 4.79 ...
 $ EX.mu.beta: num  4.24 4.44 4.71 4.95 5.06 ...
 $ EX        : num  4.38 4.51 4.87 5.11 5.21 ...
 $ VX        : num  0.115 0.116 0.0824 0.1537 0.0679 ...
 $ VX.pred   : num  0.1282 0.1288 0.0923 0.1673 0.0802 ...
 $ res       : num  0.2011 -0.3793 -0.1414 0.2397 0.0744 ...
 $ res.norm  : num  0.562 -1.057 -0.465 0.586 0.263 ...
```

Only predictions at observed locations are given in `pred.obs`, and full predictions for all space-time locations are collected in `pred.all`

```
> str(pred.cv.mesa$pred.all,1)
```

```
List of 6
 $ EX.mu     : num [1:280, 1:25] 4.85 4.66 4.47 4.29 4.12 ...
  ..- attr(*, "dimnames")=List of 2
 $ EX.mu.beta: num [1:280, 1:25] 4.24 4.13 4.04 3.94 3.86 ...
  ..- attr(*, "dimnames")=List of 2
 $ EX        : num [1:280, 1:25] 4.38 3.97 3.97 4.22 3.71 ...
  ..- attr(*, "dimnames")=List of 2
 $ VX        : num [1:280, 1:25] 0.115 0.0955 0.0809 0.0706 0.0642 ...
```

```
  ..- attr(*, "dimnames")=List of 2
 $ VX.pred   : num [1:280, 1:25] 0.1282 0.1087 0.0941 0.0838 0.0774 ...
  ..- attr(*, "dimnames")=List of 2
 $ beta      :List of 3
```

which contains fields coresponding to those in `pred.mesa.model` from `predict.STmodel`

```
> names(pred.mesa.model)
```

```
[1] "opts"       "pars"       "beta"       "EX.mu"
[5] "EX.mu.beta" "EX"         "VX"         "VX.pred"
[9] "I"
```

If requested — `LTA=TRUE` in the call to `predictCV` — the temporal averages are collected in

```
> str(pred.cv.mesa$pred.LTA)
```

```
'data.frame':        25 obs. of  9 variables:
 $ obs       : num  3.7 4.15 3.24 4.32 4.17 ...
 $ ID        : chr  "60370002" "L002" "LC001" "60370030" ...
 $ EX.mu     : num  3.83 3.9 3.24 4.11 3.93 ...
 $ EX.mu.beta: num  3.72 4.04 2.97 4.28 3.94 ...
 $ EX        : num  3.72 4.02 2.95 4.3 3.94 ...
 $ VX        : num  0.0526 0.0489 0.0333 0.0265 0.0525 ...
 $ VX.pred   : num  0.0526 0.0492 0.0336 0.0269 0.0526 ...
 $ res       : num  -0.0211 0.1278 0.2833 0.0184 0.2314 ...
 $ res.norm  : num  -0.0919 0.576 1.5452 0.1121 1.0091 ...
```

Some standard cross-validation statistics can be obtained through:

```
> summary(pred.cv.mesa)
```

```
Cross-validation predictions for STmodel with 10 CV-groups.
  Predictions for 4577 observations.
  Temporal averages for 25 locations.

RMSE:
            EX.mu EX.mu.beta        EX
obs     0.4262461  0.3574069 0.3163159
```

```
average 0.3166756  0.2381370 0.2381013

R2:
            EX.mu EX.mu.beta        EX
obs     0.6530150  0.7560416 0.8089127
average 0.3541589  0.6347837 0.6348930

Coverage of 95% prediction intervals:
               EX
obs     0.9224383
average 0.9200000
```

Here `summary.predCVSTmodel` computes RMSE, R^2, and coverage of prediction intervalls for the observations and for the long term average at each location.

If temporal averages were *not* computed by `predictCV`, the option `LTA=TRUE` can be used in `summary.predCVSTmodel` to obtain some statistics for the averages (not coverage since the variance will not be computed). We can also ask `summary` to perform a transformation of observations and predictions before computing the CV-statistics. This transform is *not biased corrected*, and for the exponential case the better option is often to use the `transform` option of `predict` and `predictCV`.

5.2.1 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.

```
> I.season <- as.factor(as.POSIXlt(pred.cv.mesa$pred.obs$date)$mon+1)
> levels(I.season) <- c(rep("Winter",2), rep("Spring",3),
                        rep("Summer",3), rep("Fall",3), "Winter")
```

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

```
> table(I.season)

I.season
Winter Spring Summer   Fall
  1096   1223   1172   1086
```

We now take a look at the residuals from the prediction, to do this we use the `pred.cv.mesa` 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 6.

```
> par(mfrow=c(1,2), mar=c(3,2,1,1), pty="s")
> qqnorm(pred.cv.mesa, col=I.season, line=2)
> qqnorm(pred.cv.mesa, norm=TRUE, main="Normalised residuals",
        col=I.season)
> legend("bottomright", legend=as.character(levels(I.season)),
        pch=1, col=1:nlevels(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.

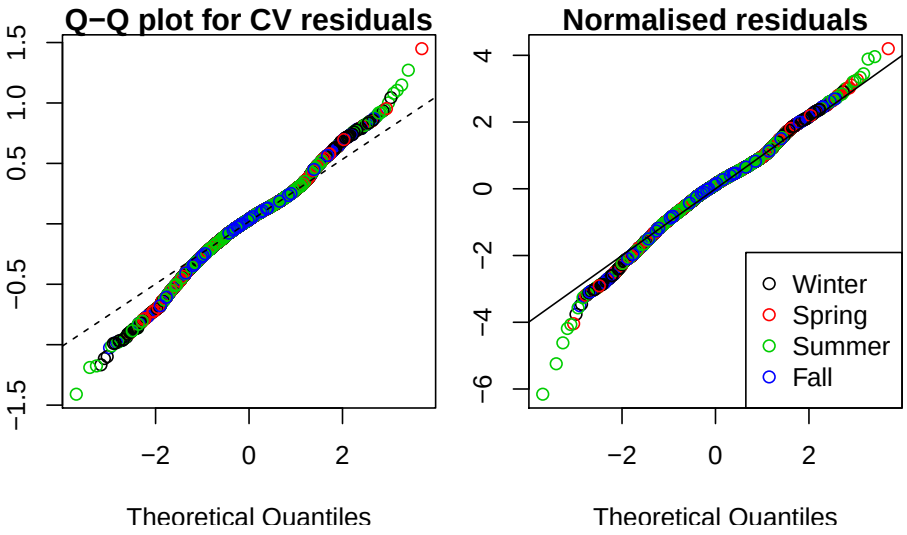

Figure 6: QQ-plots for the residuals, colour-coded by season.

In Figure 6 the raw residuals are on the left and the normalised, $(y - \mathsf{E}(y))/\sqrt{\mathsf{V}(y)}$, on the right. Both are close to normal, but have slightly heavier tails than expected. Though the departure from normality is not drastic for either, the normalised residuals are noticeably better (i.e. closer to Gaussianity).

We can also plot the residuals versus the temporal trends or versus any of the land use regression variables. In Figure 7 we plot the residuals against the first temporal trend and one of the LUR variables used to model the

$\beta_0(s)$-field; the additional argument `STdata=mesa.model` is used to define the covariates and temporal trends.

```
> par(mfcol=c(2,1),mar=c(4.5,4.5,2,2))
> scatterPlot(pred.cv.mesa, trend=1, group=I.season, col=c(2:5,1),
              xlab="First temporal smooth", type="res",
              STdata=mesa.model, main="CV Residuals - All data")
> scatterPlot(pred.cv.mesa, covar="log10.m.to.a1", group=I.season,
              col=c(2:5,1), STdata=mesa.model, type="res",
              main="CV Residuals - All data")
> legend("topleft", levels(I.season), col=c(2:5), pch=1, cex=.75)
```

Finally we can use the `mesa.data$covars$type` field to distinguish between `AQS` and `FIXED` locations, doing separate plots for separate types of locations, see Figure 8.

```
> par(mfcol=c(1,2),mar=c(4.5,4.5,2,2))
> scatterPlot(pred.cv.mesa, covar="log10.m.to.a1", group=I.season,
              subset=with(mesa.data$covars, ID[type=="AQS"]),
              col=c(2:5,1), lty=c(rep(2,4),1), type="res",
              STdata=mesa.model, main="AQS sites")
> legend("topleft", levels(I.season), col=c(2:5), pch=1, cex=.75)
> ##and for the FIXED sites
> scatterPlot(pred.cv.mesa, covar="log10.m.to.a1", group=I.season,
              subset=with(mesa.data$covars, ID[type=="FIXED"]),
              col=c(2:5,1), lty=c(rep(2,4),1), type="res",
              STdata=mesa.model, main="FIXED sites")
```

The plots in Figure 7 and 8 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 NO_x in the data set.

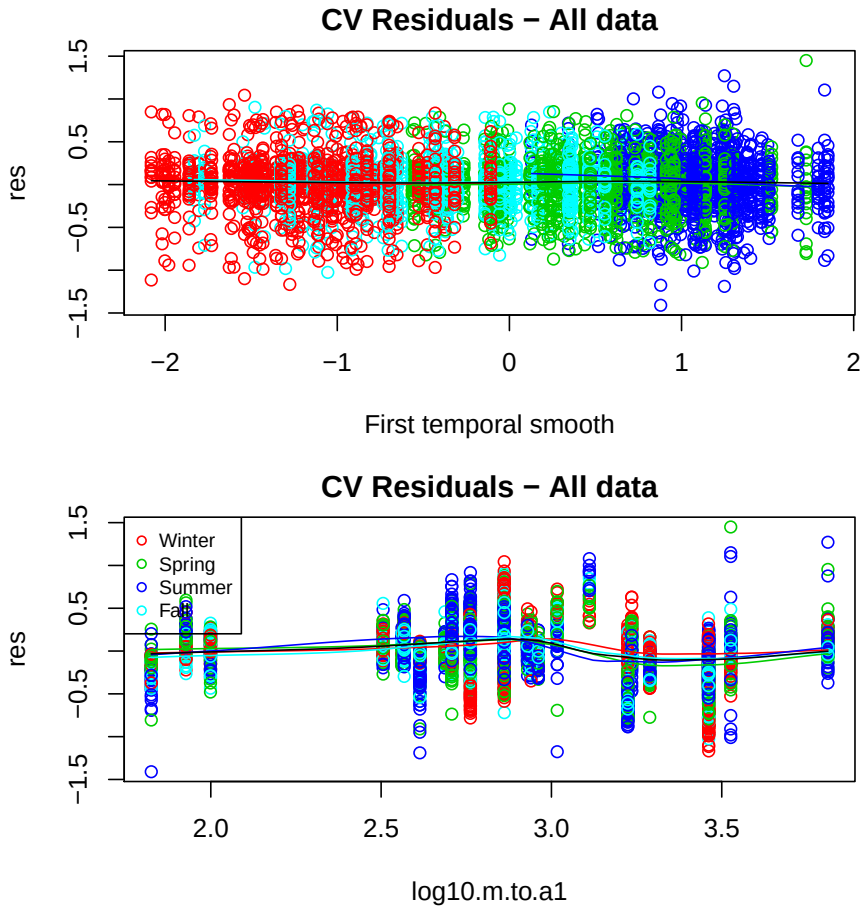

Figure 7: Residual scatter plots

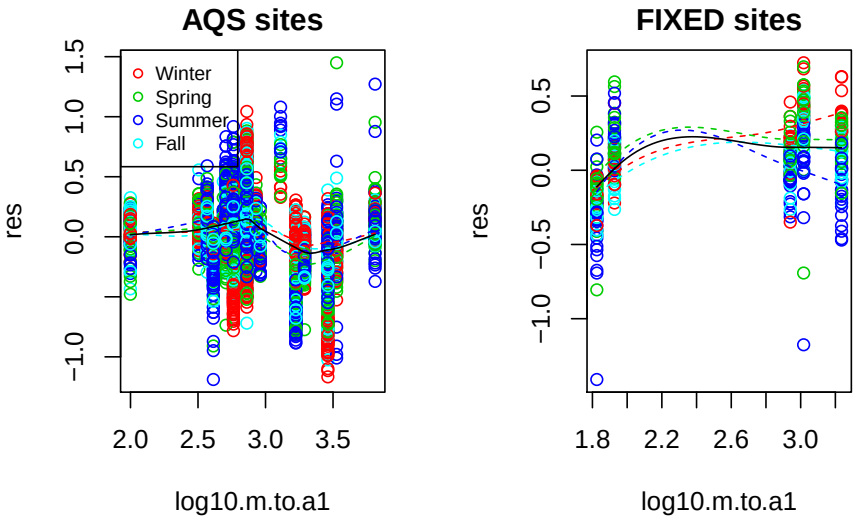

Figure 8: Residual scatter plots, by location type.

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

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.

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.

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

Lindström, J., Szpiro, A. A., Sampson, P. D., Oron, A., Richards, M., Larson, T., and Sheppard, L. (2013). A flexible spatio-temporal model for air pollution with spatial and spatio-temporal covariates. *Under revision for Environmental and Ecological Statistics*, TBD:?–?

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. D., Szpiro, A. A., Sheppard, L., Lindström, J., Adar, S. D., Allen, R. W., Avol, E. L., Oron, A. P., Larson, T., Liu, L.-J. S., and Kaufman, J. D. (2011). Comparing universal kriging and land-use regression for predicting concentrations of gaseous oxides of nitrogen (NOx) for the multi-ethnic study of atherosclerosis and air pollution (MESA air). *Atmo. Environ.*, 45(26):4412–4420.

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. (2011). Pragmatic estimation of a spatio-temporal air quality model with irregular monitoring data. *Atmo. Environ.*, 45(36):6593–6606.

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. *Sci. Total Environ.*, 408(5):1120–1130.

A Prediction at Unobserved Locations

The following is an example of predictions at unobserved locations and times. For brevity, most of the function outputs have been omitted in the Appendices.

A.1 Load Data

Let us first load relevant libraries and data.

```
> ##libraries
> library(SpatioTemporal)
> library(plotrix)
> ##load data
> data(mesa.data.raw)
> data(mesa.model)
> data(est.mesa.model)
```

A.2 Setup and Study the Data

We then setup data structures — dropping the spatio-temporal covariate and all observations at two sites — and study the resulting data.

```
> mesa.data.raw$obs <-
    mesa.data.raw$obs[,!(colnames(mesa.data.raw$obs) %in%
                         c("60595001", "LC003"))]
> mesa.data <- with(mesa.data.raw,
                    createSTdata(obs, X, n.basis=2))
```

Let us also expand the temporal trends to every week instead of every 2-weeks, giving us unobserved time-points at which to predict.

```
> mesa.data.org <- mesa.data
> T <- with(mesa.data$trend, seq(min(date), max(date), by=7))
> mesa.data <- updateTrend(mesa.data, n.basis=2, extra.dates=T)
```

Studying the reduced data structure, we see that 2 locations lack observations, there are additional time-points, and no spatio-temporal covariates.

```
> print(mesa.data)

STdata-object with:
        No. locations: 25 (observed: 23)
        No. time points: 559 (observed: 280)
        No. obs: 4229

Trend with 2 basis function(s):
[1] "V1" "V2"
with dates:
        1999-01-13 to 2009-09-23

12 covariate(s):
 [1] "ID"                  "x"
 [3] "y"                   "long"
 [5] "lat"                 "type"
 [7] "log10.m.to.a1"       "log10.m.to.a2"
 [9] "log10.m.to.a3"       "log10.m.to.road"
[11] "km.to.coast"         "s2000.pop.div.10000"

No spatio-temporal covariates.

All sites:
  AQS FIXED
   20     5
Observed:
  AQS FIXED
   19     4

For AQS:
  Number of obs: 3910
  Dates: 1999-01-13 to 2009-09-23
For FIXED:
  Number of obs: 319
  Dates: 2005-12-07 to 2009-07-01

> print(mesa.data.org)

STdata-object with:
        No. locations: 25 (observed: 23)
        No. time points: 280 (observed: 280)
        No. obs: 4229
```

```
Trend with 2 basis function(s):
[1] "V1" "V2"
with dates:
        1999-01-13 to 2009-09-23

12 covariate(s):
 [1] "ID"                  "x"
 [3] "y"                   "long"
 [5] "lat"                 "type"
 [7] "log10.m.to.a1"       "log10.m.to.a2"
 [9] "log10.m.to.a3"       "log10.m.to.road"
[11] "km.to.coast"         "s2000.pop.div.10000"

No spatio-temporal covariates.

All sites:
  AQS FIXED
   20     5
Observed:
  AQS FIXED
   19     4

For AQS:
  Number of obs: 3910
  Dates: 1999-01-13 to 2009-09-23
For FIXED:
  Number of obs: 319
  Dates: 2005-12-07 to 2009-07-01
```

We can also compare the smooth temporal trends in the two models

```
> par(mfrow=c(2,1), mar=c(2,2,2,1))
> plot(mesa.data$trend$date, mesa.data$trend$V1,
       xlab="", ylab="", main="Trend 1")
> points(mesa.data.org$trend$date, mesa.data.org$trend$V1,
         col="red", pch=3)
> plot(mesa.data$trend$date, mesa.data$trend$V2,
       xlab="", ylab="", main="Trend 2")
> points(mesa.data.org$trend$date, mesa.data.org$trend$V2,
         col="red", pch=3)
```

The trends are identical, since they are based on observations from the same 23 locations.

We then create a STmodel for the reduced data-set (Using the model specification in `mesa.model`).

```
> mesa.model.1 <- createSTmodel(mesa.data,
    LUR=mesa.model$LUR.list, cov.beta=mesa.model$cov.beta,
    cov.nu=mesa.model$cov.nu,
    locations=mesa.model$locations.list)
> mesa.model.2 <- createSTmodel(mesa.data.org,
    LUR=mesa.model$LUR.list, cov.beta=mesa.model$cov.beta,
    cov.nu=mesa.model$cov.nu,
    locations=mesa.model$locations.list, strip=TRUE)
```

Studying the models,

```
> print(mesa.model.1)
> print(mesa.model.2)
```

we note that they contain different number of locations

```
> str(loglikeSTdim(mesa.model.1))
```

```
List of 12
 $ T              : int 280
 $ m              : int 3
 $ n              : int 25
 $ n.obs          : int 23
 $ p              : Named int [1:3] 4 2 2
  ..- attr(*, "names")= chr [1:3] "const" "V1" "V2"
 $ L              : int 0
 $ npars.beta.covf: Named int [1:3] 2 2 2
  ..- attr(*, "names")= chr [1:3] "exp" "exp" "exp"
 $ npars.beta.tot : Named int [1:3] 2 2 2
  ..- attr(*, "names")= chr [1:3] "exp" "exp" "exp"
 $ npars.nu.covf  : int 2
 $ npars.nu.tot   : int 4
 $ nparam.cov     : int 10
 $ nparam         : int 18
```

```
> str(loglikeSTdim(mesa.model.2))
```

```
List of 12
 $ T              : int 280
 $ m              : int 3
 $ n              : int 23
 $ n.obs          : int 23
 $ p              : Named int [1:3] 4 2 2
  ..- attr(*, "names")= chr [1:3] "const" "V1" "V2"
 $ L              : int 0
 $ npars.beta.covf: Named int [1:3] 2 2 2
  ..- attr(*, "names")= chr [1:3] "exp" "exp" "exp"
 $ npars.beta.tot : Named int [1:3] 2 2 2
  ..- attr(*, "names")= chr [1:3] "exp" "exp" "exp"
 $ npars.nu.covf  : int 2
 $ npars.nu.tot   : int 4
 $ nparam.cov     : int 10
 $ nparam         : int 18
```

and different number of unobserved time-points

```
> dim(mesa.model.1$trend)

[1] 559   3

> dim(mesa.model.2$trend)

[1] 280   3
```

A.3 Predictions

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

In a real world application we would first use `estimate.STmodel` to estimate parameters. Here we will just use the covariance-parameters previously estimated in subsection 4.1,

```
> x <- coef(est.mesa.model, pars="cov")$par
```

and compute predictions for the two models (this may take several seconds, we have omitted variance computations to save time).

```
> E.1 <- predict(mesa.model.1, x, pred.var=FALSE)
> E.2 <- predict(mesa.model.2, x, pred.var=FALSE)
```

Studying the resulting structures, we see that the first case has computed predictions at all locations, while the second case only computed predictions at the 23 locations, and 280 time-points

```
> colnames(E.1$EX)
```

```
 [1] "60370002" "60370016" "60370030" "60370031" "60370113"
 [6] "60371002" "60371103" "60371201" "60371301" "60371601"
[11] "60371602" "60371701" "60372005" "60374002" "60375001"
[16] "60375005" "60590001" "60590007" "60591003" "L001"
[21] "L002"     "LC001"    "LC002"    "60595001" "LC003"
```

```
> str(E.1$EX)
```

```
 num [1:559, 1:25] 4.55 4.31 4 4.17 4.03 ...
 - attr(*, "dimnames")=List of 2
  ..$ : chr [1:559] "1999-01-13" "1999-01-20" "1999-01-27" "1999-02-03" ...
  ..$ : chr [1:25] "60370002" "60370016" "60370030" "60370031" ...
```

```
> colnames(E.2$EX)
```

```
 [1] "60370002" "60370016" "60370030" "60370031" "60370113"
 [6] "60371002" "60371103" "60371201" "60371301" "60371601"
[11] "60371602" "60371701" "60372005" "60374002" "60375001"
[16] "60375005" "60590001" "60590007" "60591003" "L001"
[21] "L002"     "LC001"    "LC002"
```

```
> str(E.2$EX)
```

```
 num [1:280, 1:23] 4.55 4 4.03 4.18 3.72 ...
 - attr(*, "dimnames")=List of 2
  ..$ : chr [1:280] "1999-01-13" "1999-01-27" "1999-02-10" "1999-02-24" ...
  ..$ : chr [1:23] "60370002" "60370016" "60370030" "60370031" ...
```

The predictions are equal (to within numerical precision).

```
> range(E.1$EX[rownames(E.2$EX),colnames(E.2$EX)] - E.2$EX)
```

```
[1] 0 0
```

We could also use the original data structure to define additional prediction locations.

```
> E.3 <- predict(mesa.model.2, x, STdata=mesa.data, pred.var=FALSE)
```

Here, the trend from `mesa.model.2` is used for prediction, with values at additional time-points computed by `mesa.model.2$trend.fnc`. Having obtained predictions at all locations and times in `mesa.data`, we compare these to those in `E.1`.

```
> colnames(E.3$EX)

 [1] "60370002" "60370016" "60370030" "60370031" "60370113"
 [6] "60371002" "60371103" "60371201" "60371301" "60371601"
[11] "60371602" "60371701" "60372005" "60374002" "60375001"
[16] "60375005" "60590001" "60590007" "60591003" "L001"
[21] "L002"     "LC001"    "LC002"    "60595001" "LC003"

> str(E.3$EX)

 num [1:559, 1:25] 4.55 4.31 4 4.17 4.03 ...
 - attr(*, "dimnames")=List of 2
  ..$ : chr [1:559] "1999-01-13" "1999-01-20" "1999-01-27" "1999-02-03" ...
  ..$ : chr [1:25] "60370002" "60370016" "60370030" "60370031" ...

> all.equal(E.3,E.1)

[1] TRUE
```

A.3.1 Temporal Averages

The `predict` function provides an option for computing temporal averages, and variances of the temporal averages. The averages can either be over all time points or over only a few time-points. Here we will ilustrate by computing temporal averages for each year. The first step is to create a list where each component gives the dates over which to average, e.g. by splitting the observations dates between each year.

```
> LTA <- with(mesa.model.1$trend, split(date,as.POSIXlt(date)$year+1900))
> str(LTA)

List of 11
 $ 1999: Date[1:51], format: "1999-01-13" ...
 $ 2000: Date[1:52], format: "2000-01-05" ...
 $ 2001: Date[1:52], format: "2001-01-03" ...
```

```
 $ 2002: Date[1:52], format: "2002-01-02" ...
 $ 2003: Date[1:53], format: "2003-01-01" ...
 $ 2004: Date[1:52], format: "2004-01-07" ...
 $ 2005: Date[1:52], format: "2005-01-05" ...
 $ 2006: Date[1:52], format: "2006-01-04" ...
 $ 2007: Date[1:52], format: "2007-01-03" ...
 $ 2008: Date[1:53], format: "2008-01-02" ...
 $ 2009: Date[1:38], format: "2009-01-07" ...
```

```
> lapply(LTA[1:3], range)
```

```
$`1999`
[1] "1999-01-13" "1999-12-29"

$`2000`
[1] "2000-01-05" "2000-12-27"

$`2001`
[1] "2001-01-03" "2001-12-26"
```

To allow for averaging over different time at each location, we need a list with each element named by the location and containing the temporal list constructed above.

```
> ID <- mesa.model.1$locations$ID
> LTA <- rep(list(LTA), length(ID))
> names(LTA) <- ID
```

The `LTA` object can now be used as an argument to `predict`, specifying which times, at each location, that we should average over.

```
> E.1.LTA <- predict(mesa.model.1, x, pred.var=FALSE, LTA=LTA)
```

The resulting predictions now contain a `LTA` field that provides the averages and (not shown here) variances.

```
> head(E.1.LTA$LTA)
```

```
              EX.mu EX.mu.beta       EX
60370002.1 4.156455   4.080497 4.072983
60370002.2 4.060119   3.952038 3.956382
60370002.3 3.972387   3.922065 3.904138
```

```
60370002.4 3.919124   3.881751 3.876226
60370002.5 3.882752   3.824432 3.827193
60370002.6 3.729945   3.578364 3.562589
```

The element `E.1.LTA`*opts*`LTA.list` contains a copy of the `LTA` option provided in the call to `predict` that can be used to check over which time points each average has been computed.

In this simple case the averages are the same as those obtained by just averaging the predictions; the main need for seperate computation of averages are to: 1) get correct variances, accounting for temporal dependencies due to the temporal basis functions; and 2) account for non-linearities when averaging over log-Gaussian fields.

```
> E.1.LTA.alt <- sapply(split(E.1$EX[,1], as.POSIXlt(rownames(E.1$EX))$year),mean)
> cbind(E.1.LTA.alt, E.1.LTA$LTA$EX[1:11])
```

```
    E.1.LTA.alt
99     4.072983 4.072983
100    3.956382 3.956382
101    3.904138 3.904138
102    3.876226 3.876226
103    3.827193 3.827193
104    3.562589 3.562589
105    3.628571 3.628571
106    3.624021 3.624021
107    3.540999 3.540999
108    3.400794 3.400794
109    3.159187 3.159187
```

B 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 4, and compare predictions to the simulated data.

B.1 Load Data

Let us first load relevant libraries and data.

```
> ##Load libraries
> library(SpatioTemporal)
> library(plotrix)
> library(maps)
> ##and data
> data(mesa.model)
> data(est.mesa.model)
```

B.2 Simulating some Data

First we simulate 4 samples of new data, using the parameters previously estimated in subsection 4.1.

```
> x <- coef(est.mesa.model)$par
> sim.data <- simulate(mesa.model, nsim=4, x=x)
```

Examine the result

```
> names(sim.data)

[1] "param" "B"     "X"     "obs"

> str(sim.data,1)

List of 4
 $ param: Named num [1:19] 0.000898 3.74027 -0.202129 0.040218 0.037436 ...
  ..- attr(*, "names")= chr [1:19] "gamma.lax.conc.1500" "alpha.const.(Inte
 $ B    : num [1:25, 1:3, 1:4] 3.98 3.89 3.97 3.61 3.8 ...
  ..- attr(*, "dimnames")=List of 3
```

```
$ X    : num [1:280, 1:25, 1:4] 4.91 4.53 4.43 4.11 4.12 ...
 ..- attr(*, "dimnames")=List of 3
$ obs  :List of 4
```

Here `sim.data$X` contains the 4 simulations, `sim.data$B` contains the simulated beta fields and `sim.data$obs` contains observations data.frames that can be used to replace `mesa.model$obs`.

Let's create model structures based on the simulated observations

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

Compute predictions for the 4 simulated datasets We do the computations for only one location to save time, thus we need to define a `STdata`-object with one unobserved-site and all covariates.

```
> data(mesa.data.raw)
> mesa.data.raw$X <- mesa.data.raw$X[mesa.data.raw$X[,"ID"]=="60590001",]
> mesa.data <- createSTdata(obs=NULL, covars=mesa.data.raw$X,
      extra.dates=as.Date(mesa.model$trend$date),
      SpatioTemporal=list(lax.conc.1500=mesa.data.raw$lax.conc.1500))
```

For these predicitons we'll just use the known parameters, however one could easily estimate new parameters based on the simulated data using `estimate.STmodel` (although this would take more time). *Please note that following the predictions may take a few minutes.*

```
> E <- list()
> for(i in 1:length(sim.data$obs)){
    E[[i]] <- predict(mesa.data.sim[[i]], x, STdata=mesa.data)
  }
```

B.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.

Let's compare the predicted values and the simulated data for all four simulations

```
> par(mfrow=c(2,2),mar=c(2.5,2.5,2,.5))
> for(i in 1:4){
    ##plot predictions, but not the observations
    plot(E[[i]])
    ##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[,mesa.data$covars$ID,i], col="red")
  }
```

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.

```
> library(SpatioTemporal)
> library(plotrix)
> ##load data
> data(mesa.model)
> data(est.mesa.model)
```

C.2 Running the MCMC

In addition to the standard model fitting described in subsection 4.1 the model (and parameter uncertainties) can be estimated using a simple Metropolis-Hastings algorithm.

Here we run a standard random-walk MCMC starting at the mode found in subsection 4.1 and using a proposal matrix based on the Hessian, as suggested in Roberts et al. (1997).

```
> ##parameters
> x <- coef(est.mesa.model)
> ##and Hessian
> H <- est.mesa.model$res.best$hessian.all
```

WARNING: The following steps are time-consuming.

```
> MCMC.mesa.model <- MCMC(mesa.model, x$par, N = 2500,
                          Hessian.prop = H)
```

ALTERNATIVE: Load pre-computed results.

```
> data(MCMC.mesa.model)
```

End of alternative

C.3 Results

We start by looking at the status of the MCMC-runs, and components of the result structure.

```
> print(MCMC.mesa.model)

MCMC for STmodel, results over 2500 iterations.
  19 unknown parameters
  Acceptance rate: 0.2780721

No fixed parameters.

> names(MCMC.mesa.model)

[1] "par"        "log.like"   "acceptance" "Sigma.prop"
[5] "chol.prop"  "x.fixed"
```

as well as some summaries of the results (e.g. which values the parameters took).

```
> summary(MCMC.mesa.model)
```

C.3.1 Plotting the Results

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

```
> par(mfrow=c(4,1),mar=c(2,2,2.5,.5))
> for(i in c(4,9,13,15)){
    plot(MCMC.mesa.model, i, ylab="", xlab="", type="l")
  }
```

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.

```
> dens <- density(MCMC.mesa.model, estSTmodel=x)
> ##plots for all covariance parameters
> par(mfrow=c(3,3),mar=c(4,4,2.5,.5))
> for(i in 9:17){
    plot(dens, i, norm.col="red")
  }
```

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 4.2). 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.