Spatio-temporal geostatistics using `gstat`

ifgi
Institute for Geoinformatics
University of Münster

Edzer Pebesma

October 30, 2011

1 Introduction

Since `gstat` package version 1.0-0, a dependency of gstat on the R package `spacetime` was introduced, allowing the code in `gstat` to exploit spatio-temporal data structures from that package. This vignette describes the possibilities and limitations of the package for spatio-temporal geostatistics.

To understand some of the possibilities and limitations, some knowledge of the history of the software is needed. The original `gstat` software (Pebesma and Wesseling, 1998) was a standalone computer *program* written in around 25,000 lines of C code, and would do geostatistical modelling, prediction and simulation. The `gstat` R package (Pebesma, 2004) consisted mostly of an R interface to this C code, together with convenience functions to use R's modelling interface (formula's, see `?lm`) and graphic capabilities (trellis graphics in package `lattice` to show cross variogram as matrix plots; interaction with variogram clouds using base plots).

Starting 2003, a group of programmers developed a set of classes and methods for dealing with spatial data in R (points, lines, polygons, grids), which was supported by the publications of the well-known ASDAR book (Bivand et al. 2008; see also `http://www.asdar-book.org/`) and helped convergence in the user community, with in 2011 over 2000 subscribers on the `r-sig-geo` mailing list. Package `gstat` was one of the first packages that adopted and benefited from these classes.

To realize a particular idea, writing code in C typically takes about 10-20 times as long as writing it in R. C code can be more efficient, gives more control over memory usage, but is also more error prone–mistakes in C code make an R session crash, something that is hard to do when writing R code.

The original C code of `gstat` (Pebesma and Wesseling, 1998) provides all kriging varieties (universal, ordinary, simple; univariable, or multivariable as in cokriging) for two- or three-dimensional data. When the spatial domain is constrained to two dimensions (and this might cover over 99% of the use cases!), the third dimension might be used to represent time. As such, the *metric* variogram model, which allows for geometric anisotropy definition in three dimensions, can be used for spatio-temporal kriging. When defining the three-dimensional variogram as the sum of 2 or more nested variogram (summed)

models, one can choose anisotropy coefficients for a single model such that this model is *effectively* zero in some directions, e.g. in space *or* in time; this allows one to approximate the so-called space-time sum model. It should be noted that at the C code there is no knowledge whether a third dimension represents space, or time. As such, particular characteristics of time cannot be taken care of.

Since the second half of 2010, the development of an R package `spacetime` started. It provides methods and classes for spatio-temporal data, and builds on the spatial data classes in `sp` and time series classes in `xts`. This document will explain how data in this form, and methods provided by this package, can be used for spatio-temporal geostatistics.

We will work with a data set with air quality (PM10) measurements over germany, taken from rural background stations available in the data sets provided by the European Environmental Agency.

```
> library(spacetime)
> rm(list = ls())
> data(air)
> ls()
```

```
[1] "DE_NUTS1" "rural"
```

2 Variography

2.1 Temporal autocorrelation and cross correlation

We will look into a subset of the data, ranging from 2005 to 2010, and remove stations that have only missing values in this period:

```
> rr = rural[, "2005::2010"]
> unsel = which(apply(as(rr, "xts"), 2, function(x) all(is.na(x))))
> r5to10 = rr[-unsel, ]
> summary(r5to10)
```

```
Object of class STFDF
 with Dimensions (s, t, attr): (53, 1826, 1)
[[Spatial:]]
Object of class SpatialPoints
Coordinates:
               min      max
coords.x1  6.28107 14.78617
coords.x2 47.80847 54.92497
Is projected: FALSE
proj4string :
[+init=epsg:4326 +proj=longlat +ellps=WGS84 +datum=WGS84 +no_defs
+towgs84=0,0,0]
Number of points: 53
[[Temporal:]]
     Index                ..1
 Min.   :2005-01-01   Min.   :2558
 1st Qu.:2006-04-02   1st Qu.:3014
 Median :2007-07-02   Median :3470
```

```
 Mean   :2007-07-02   Mean   :3470
 3rd Qu.:2008-09-30   3rd Qu.:3927
 Max.   :2009-12-31   Max.   :4383
[[Data attributes:]]
     Min.   1st Qu.    Median      Mean   3rd Qu.      Max.      NA's
    0.560     9.275    13.850    16.260    20.330   269.100 21979.000
```

Next, we will (rather arbitrarily) select four stations, which have the following labels:

```
> rn = row.names(r5to10@sp)[4:7]
> rn

[1] "DEBE056" "DEBE032" "DEHE046" "DENW081"
```

In the following, autocorrelation functions are computed and plotted. The resulting plot is shown in figure 1.

```
> par(mfrow = c(2, 2))
> for (i in rn) acf(na.omit(r5to10[i, ]), main = i)
> par(mfrow = c(1, 1))
```

Auto- and cross correlations can be computed when a multivariate time series object is passed to `acf`:

```
> acf(na.omit(as(r5to10[rn, ], "xts")))
```

The resulting plot is shown in figure 2. From these graphs one should be able to observe the following

- autocorrelations for lag 0 are always 1
- cross correlations for lag 0 are not always 1
- cross correlations can be asymmetric, meaning that when $\rho_{AB}(h)$ is the correlation between $Z(s_A, t)$ and $Z(s_B, t + h)$,

  $$\rho_{AB}(h) = \rho_{BA}(-h) \neq \rho_{AB}(-h)$$

  with s_A and s_B the two stations between which a cross correlation is computed, and h the (directional!) lag between the series.

The plot further more shows that for these four stations the asymmetry is not very strong, but that cross correlations are fairly strong and of a similar form of autocorrelations.

This kind of plot does not work very well in layouts of e.g. 10 x 10 sub-plots; `acf` automatically chooses 4 x 4 as the maximimum a single plot. To try this out, do a 7 x 7 plot

```
> acf(na.omit(as(r5to10[4:10, ], "xts")))
```

and note that here we see in the last figure (DESH & DESN04) a pair of plots with nearly no cross correlation. This might have to do with the spatial distance between these two stations:

```
> par(mfrow = c(2, 2))
> rn = row.names(r5to10@sp)[4:7]
> for (i in rn) acf(na.omit(r5to10[i, ]), main = i)
```

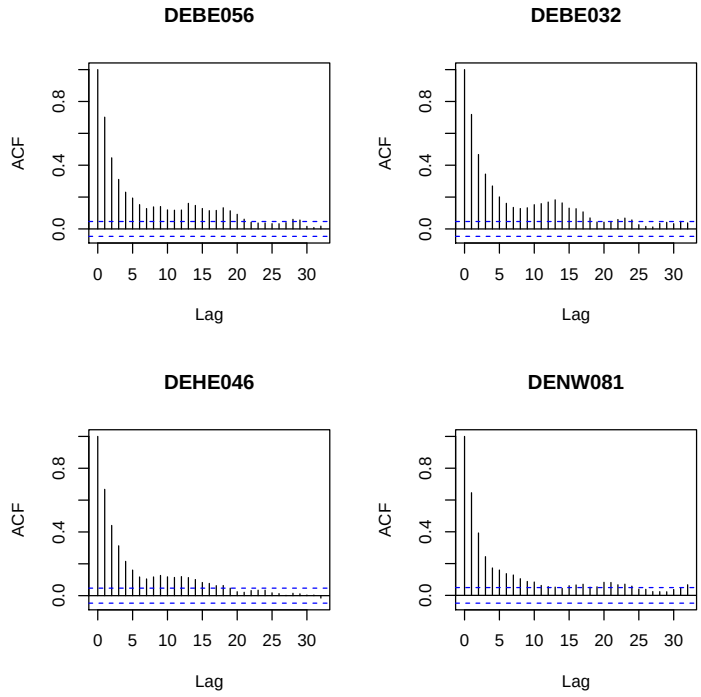

Figure 1: Autocorrelations for PM10; time lag unit in days.

```
> print(spDists(r5to10[4:10, ]@sp), digits = 3)
```

```
      [,1]  [,2] [,3] [,4] [,5] [,6] [,7]
[1,]   0.0  28.8  344  468  292  235  135
[2,]  28.8   0.0  317  440  269  231  131
[3,] 343.6 317.3    0  150  309  282  285
[4,] 467.9 439.7  150    0  336  432  430
[5,] 291.7 268.5  309  336    0  438  362
[6,] 235.5 231.0  282  432  438    0  101
[7,] 134.6 130.9  285  430  362  101    0
```

(What is the spatial distance between stations DESH and DESN04?)

2.2 Spatial correlation, variograms

In the next steps, we will sample 100 time instances randomly,

```
> rs = sample(dim(r5to10)[2], 100)
```

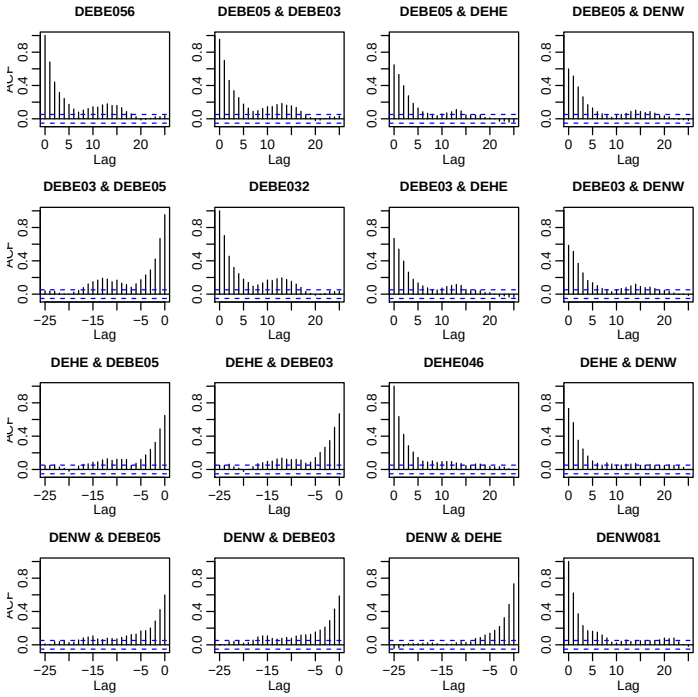

Figure 2: autocorrelations (diagional) and cross correlations (off-diagional) for the four stations selected; time lag unit in days.

we select these instances as a `SpatialPointsDataFrame` and add a time index to them. After this we bind them together in a single `SpatialPointsDataFrame` which has a time index `ti`:

```
> lst = lapply(rs, function(i) {
+     x = r5to10[, i]
+     x$ti = i
+     x
+ })
> pts = do.call(rbind, lst)
```

Then, we can compute the pooled variogram

```
> library(gstat)
> v = variogram(PM10 ~ ti, pts[!is.na(pts$PM10), ], dX = 0)
```

and plot it (figure 3):

```
> plot(v, fit.variogram(v, vgm(1, "Exp", 200, 1)))
> vmod = fit.variogram(v, vgm(1, "Exp", 200, 1))
> plot(v, vmod)
```

The fitted model is this:

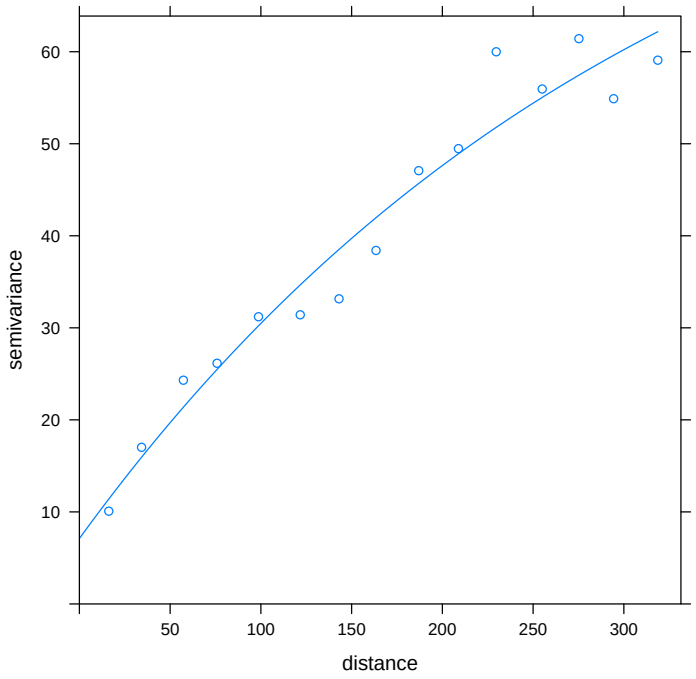

Figure 3: sample spatial variogram, averaged over 100 time randomly chosen time steps

```
> vmod

  model     psill   range
1   Nug  7.104396   0.000
2   Exp 87.807481 323.073
```

One should note that the fit is rather poor, and not forget that we only have 53 stations selected. The time resolution is rich (1862 days) but the number of stations is small:

```
> dim(r5to10)

[1]   53 1826    1
```

We can fit a spatio-temporal variogram the usual way, by passing an object of class `STFDF`:

```
> vv = variogram(PM10 ~ 1, r5to10, width = 20, cutoff = 200)
```

Alternatively, if this takes too long, a temporal subset can be taken, e.g. using the first 200 days:

```
> vv = variogram(PM10 ~ 1, r5to10[, 1:200], width = 20, cutoff = 200)
```

taking random days from the full period will lead to the a wrong assumption that every time index increment reflect a constant lag increase. As an alternative, we will here load the precomputed S/T variogram:

```
> data(vv)
```

Plotting this object can be done in two ways, both shown in figure 4:

```
> plot(vv, ylab = "time lag (days)")
> plot(vv, map = FALSE, ylab = "time lag (days)")
```

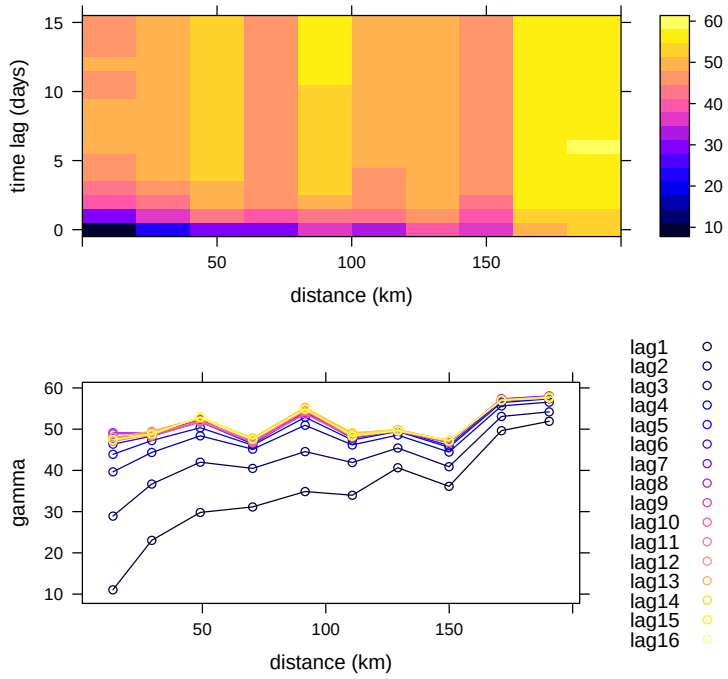

Figure 4: Spatio-temporal sample variogram map (top) and sample variograms for each time lag (bottom); both figures depict the information of object `vv`

2.3 Fitting a spatio-temporal variogram model

```
> ExpVgmMetric = function(x, s, t, nugget = 0) {
+     h = sqrt(s^2 + (x[3] * as.numeric(t))^2)
+     ifelse(h == 0, 0, nugget + x[1] * (1 - exp(-h/x[2])))
+ }
> ExpFitFn = function(x, gfn, v, trace = FALSE, ...) {
+     mod = gfn(x, v$spacelag, v$timelag, ...)
+     resid = v$gamma - mod
+     if (trace)
+         print(c(x, MSE = mean(resid^2)))
```

```
+     mean(resid^2)
+ }
> pars.M = optim(c(sill = 50, range = 100, anis = 50), ExpFitFn,
+     gfn = ExpVgmMetric, v = vv, nugget = 10)
> pars.M$par

    sill    range     anis
41.33730 83.17767 74.32199
```

The final model can be added, as fitted model to the sample variogram, and plotted by (figure 5):

```
> vv$model = ExpVgmMetric(pars.M$par, vv$spacelag, vv$timelag,
+     nugget = 10)
> plot(vv)
```

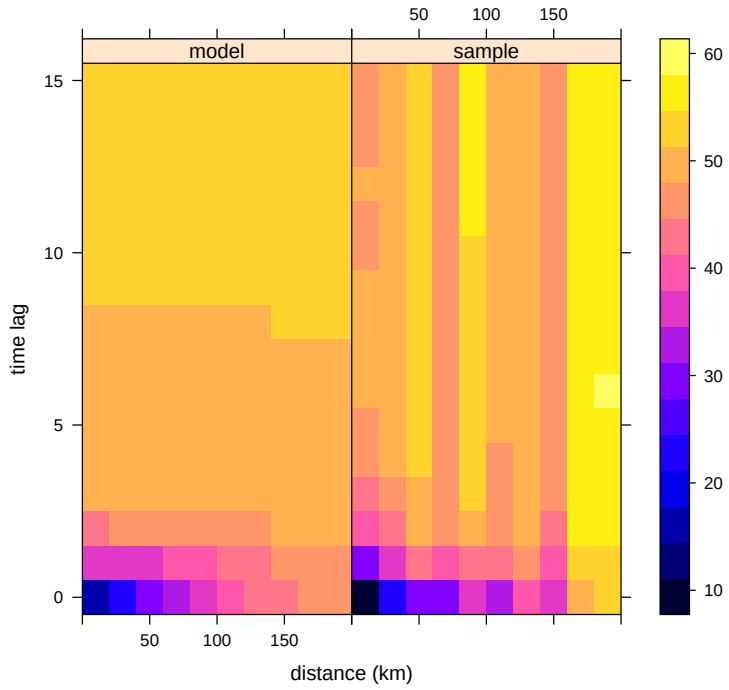

Figure 5: sample variogram map (right) and fitted metric model (left).

Now, let us try to fit and plot a separable model (figure 6):

```
> ExpVgmSeparable = function(x, s, t, nugget = 0) {
+     h = s/x[2] + as.numeric(t)/x[3]
+     ifelse(h == 0, 0, nugget + x[1] * (1 - exp(-h)))
+ }
> pars.S = optim(c(sill = 40, srange = 90, trange = 2), ExpFitFn,
+     gfn = ExpVgmSeparable, v = vv, nugget = 10)
```

```
> vv$model = ExpVgmSeparable(pars.S$par, vv$spacelag, vv$timelag,
+     nugget = 10)
> pars.S$par

     sill    srange    trange
41.482486 90.238746  1.631836

> plot(vv)
```

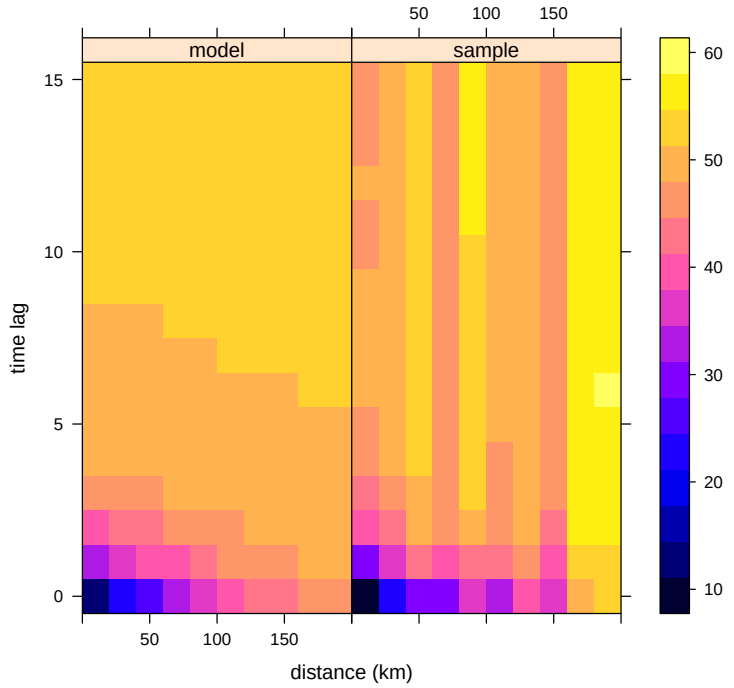

Figure 6: sample variogram map (right) and fitted seperable model (left).

A wireframe (3D) plot of the sample variogram can be obtained e.g. by

```
> library(lattice)
> wireframe(gamma ~ spacelag * timelag, vv, drape = TRUE, col.regions = bpy.colors)
```

which is shown in figure 7.

3 Spatio-temporal prediction

The vignette in package `spacetime` gives an example of using the gstat function `krigeST` for spatio-temporal kriging of the Irish wind data. The variogram there needs to be specified as two separate variogram models, the product of which needs to be the target spatio-temporal variogram. The `krigeST` function

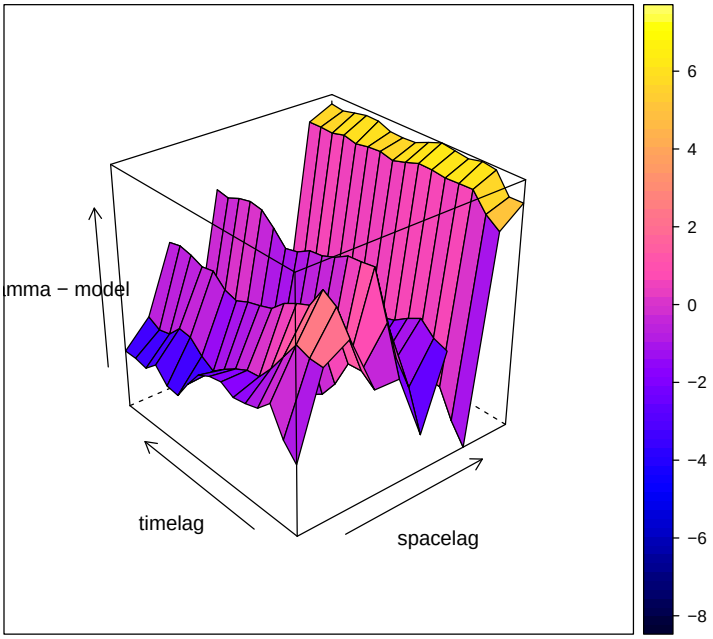

Figure 7: Wireplot of sample space-time variogram

uses global kriging, but only needs to invert the purely spatial and purely time covariance matrices.

For more generic spatio-temporal kriging where space is two-dimensional, one could use `krige`, defining the observations and prediction locations as three-dimensional data sets, see for an example

```
> demo(gstat3D)
```

It needs to be pointed out that in that case, the time (typically the third dimension) needs to be numeric, and three-dimensional anisotropy needs to be defined properly (see `?vgm`).

In case the data set is too large for global kriging, one could try to use local kriging, and select data within some distance, or by specifying `nmax` (the nearest n observations). In both cases, it is advisable to transform time such that one can use an *isotropic* variogram model in the three dimensions, as only in that case the nearest n observations correspond to the n most correlated observations.

An additional consideration is that in space-time, observations may not be regularly spaced. In some cases, the nearest n obervations may come from a single measurement location, which may lead to sharp jumps/boundaries in the interpolated values. This might be solved by using larger neighbourhoods, or by setting the `omax` in `krige` or `gstat` calls to the neighbourhood size to select *per octant* (this should be combined with specifying `maxdist`).

References

- Bivand, R., E. Pebesma and V. Gomez-Rubio, 2008. Applied Spatial Data Analysis with R. Springer.
- Cressie, N.A.C., 1993. Statistics for Spatial Data. Wiley.
- Cressie, N. and C. Wikle, 2011. Statistics for Spatio-temporal Data. Wiley.
- Pebesma, E.J., Wesseling, C.G., 1998. Gstat, a program for geostatistical modelling, prediction and simulation. Computers & Geosciences, 24 (1), pp. 17–31.
- Pebesma, E.J., 2004. Multivariable geostatistics in S: the gstat package. Computers & Geosciences 30: 683-691
- Ver Hoef, J.M., Cressie, N.A.C, 1993. Multivariable Spatial Prediction. Mathematical Geology, 25 (2), pp. 219–240.