geoRglm : a package for generalised linear spatial models

introductory session

Ole F. Christensen & Paulo J. Ribeiro Jr.
9 February 2002

The package **geoRglm** provides functions for inference in generalised linear spatial models using the software **R**. This document illustrates some of the capabilities of the package.

We assume that the reader is familiar with the **geoR** package (see the introductory session for **geoR**).

The objective is to introduce the reader to the **geoRglm** commands and show how they can be used. The commands used here are basic examples of the package handling, where we typically use default arguments for the function calls. We encourage the user also to inspect other function arguments.

For further details on the functions included in **geoRglm**, we refer to the **geoRglm** documentation.

1. STARTING A SESSION AND LOADING DATA

After starting an **R** session, we first load **geoR** and **geoRglm** with the commands:

```
library(geoR)
library(geoRglm)
```

If the installation directories for the packages are not the default locations for **R** packages, type:

```
library(geoR, lib.loc = "PATH_TO_geoR")
library(geoRglm, lib.loc = "PATH_TO_geoRglm")
```

where "PATH_TO_geoR" and "PATH_TO_geoRglm" are the paths to the directories where **geoR** and **geoRglm** are installed, respectively. If the packages are correctly loaded the following messages will be displayed:

```
--------------------------------------------------
geoR - a package for geostatistical analysis in R
geoR version 1.2-1 is now loaded
--------------------------------------------------
```

```
------------------------------------------------------------
geoRglm - a package for generalised linear spatial models
geoRglm version 0.3-7 is now loaded
------------------------------------------------------------
```

Typically, data are stored as an object (a list) of class `"geodata"` (see the **geoR** introductory session for more details on this). For the data sets considered here, the object will sometimes include a vector `units.m` consisting of observation times (for the Poisson distribution) or numbers `N` in `bi(N,p)` (for the binomial distribution).

We use the data sets `y50` and `rongelap` included in the **geoRglm** distribution for the examples presented in this document. These data sets can be loaded by typing:

```
data(y50)
data(rongelap)
```

Helpfiles are available for `geoRglm`. For getting help on the function `pois.log.krige`, just type:

```
help(pois.log.krige)
```

2. CONDITIONAL SIMULATION and SPATIAL PREDICTION

Here we describe conditional simulation using MCMC and spatial prediction in the Poisson-log normal model, when covariance parameters are fixed. Full Bayesian methods are also implemented and will be presented in Section 3.

The nugget effect parameter (microscale variation) in the underlying Gaussian field can be set to a fixed value. The same applies for the smoothness and anisotropy parameters. Options for taking covariates (trends) into account are also included.

Conditional simulation and prediction with fixed covariance parameters in the Poisson-log normal model can be performed with options for either fixed `beta` (OK) or flat prior on `beta` (SK). The function uses a Langevin-Hastings MCMC algorithm for simulating from the conditional distribution.

An example where all parameters are fixed is shown below (for illustration purposes, some parameter values are just taken).

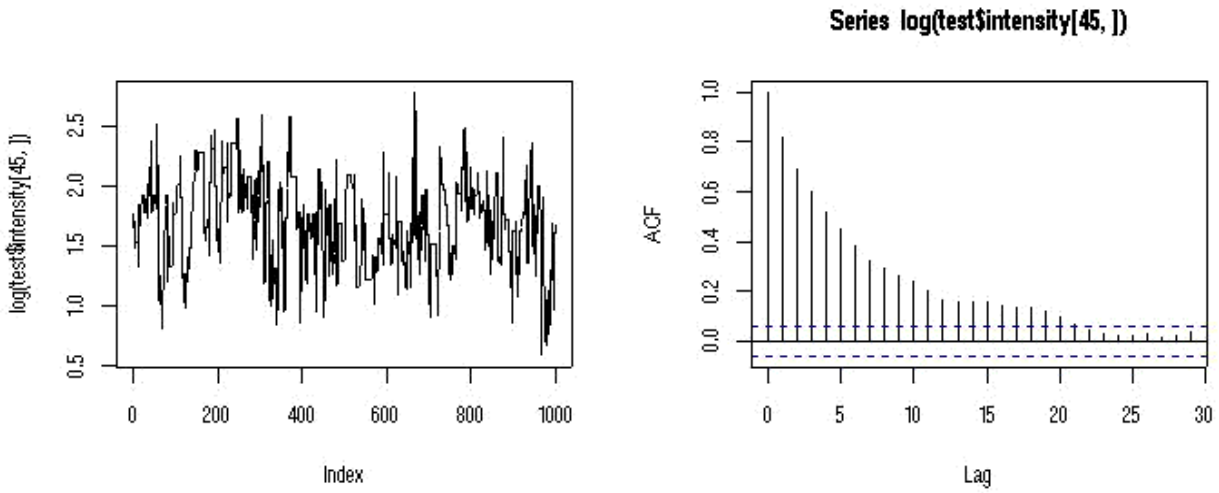
First we need to tune the algorithm by scaling the proposal variance so that acceptance rate is approximately 60 percent (optimal acceptance rate for Langevin-Hastings algorithm). This is done by trial and error.

```
model2 <- krige.glm.control(cov.pars = c(1,1), beta = 1)
test2.tune <- pois.log.krige(y50, krige = model2, mcmc.input =
list(S.scale = 0.09, thin = 1))
```

After a few tryouts we decide to use `S.scale = 0.07`. We also need to study how well the chain

is mixing.

```
     test2.tune <- pois.log.krige(y50, krige = model2, mcmc.input =
list(S.scale = 0.07, thin = 1))
     plot(log(test2.tune$intensity[45,]), type = "l")
     require(ts)
     acf(log(test2.tune$intensity[45,]), type = "correlation", plot = TRUE)
```



Here the functions in the `coda` package would be useful for assessing the convergence and inspect the mixing of the MCMC algorithm (consider installing this R-package !). For a small demonstration of a few CODA functions used on the output above, see here.

To reduce the autocorrelation of the samples we decide to subsample every 50 iteration.

Now we make (minimal mean square error) prediction of the intensity at the two locations (0.5, 0.5) and (1, 0.4).

```
     test2 <- pois.log.krige(y50, locations = cbind(c(0.5,0.5),c(1,0.4)), krige
= model2, mcmc.input = mcmc.control(S.scale = 0.07, thin = 50), output =
output.glm.control(sim.predict = TRUE))
```

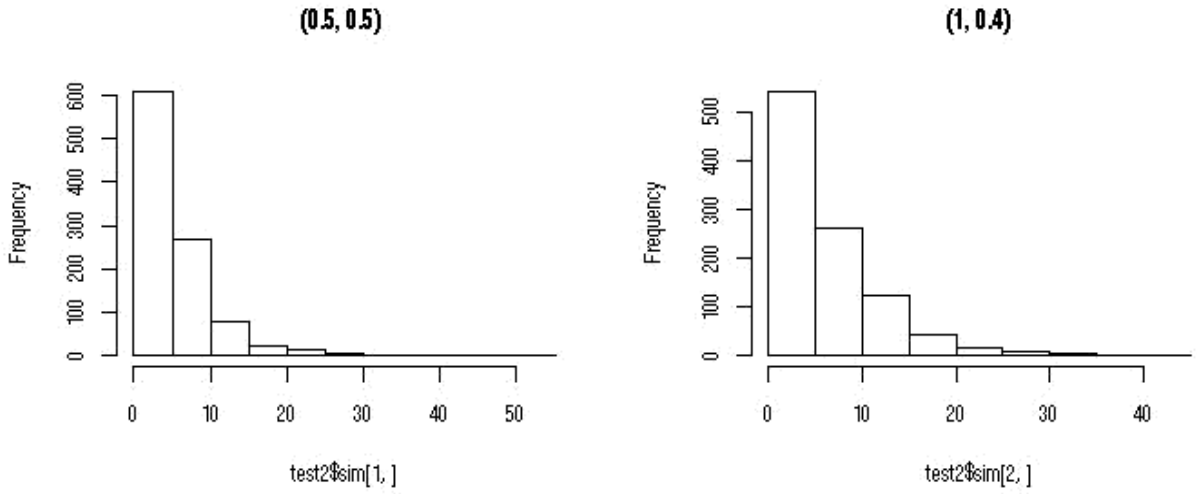
The output is a list including the predicted values (`test2$predict`), the prediction variances (`test2$krige.var`) and the estimated Monte Carlo standard errors on the predicted values (`test2$mcmc.error`). Please consider printing out the predicted values and the associated Monte Carlo standard errors:

```
     test2$predict
     test2$mcmc.error
```

Note that the Monte Carlo standard errors (the errors due to the MCMC-simulation) are small compared to predicted values, which is very satisfactory. (Monte Carlo standard errors on the prediction variances is not implemented yet).

By specifying `sim.predict = TRUE`, simulations are drawn from the predictive intensity at the two prediction locations (test2$simulations). These simulations are plotted below.

```
par(mfrow = c(1,2))
hist(test2$simulations[1,], main = "(0.5, 0.5) ")
hist(test2$simulations[2,], main ="(1, 0.4)")
```



The way to specify that `beta` should follow a uniform prior would be:

```
    model2.u <- krige.glm.control(cov.pars = c(1,1), beta = 1, type.krige =
"ok"))
    test2.unif.beta <- pois.log.krige(y50, krige = model2.u, mcmc.input =
list(S.scale = 0.07, thin = 20))
```

3. BAYESIAN ANALYSIS

Bayesian analysis for the Poisson-log normal model and the binomial-logit model is implemented by the functions `pois.krige.bayes` and `binom.krige.bayes`, respectively. Model parameters can be treated as fixed or random.

As an example consider first a model without nugget and including uncertainty in the `beta` and `sigma.sq` parameters. A Bayesian analysis is made by typing commands like:

```
    prior5 <- prior.glm.control(phi.prior = "fixed", phi = 0.1)
    mcmc5.tune <- mcmc.control(S.scale = 0.01, thin = 1)
    test5.tune <- pois.krige.bayes(y50, prior = prior5, mcmc.input =
mcmc5.tune)
```

Now chose `S.scale` (Acc-rate=0.60 is preferable). After having adjusted the parameters for the MCMC algorithm and checking the output we run an analysis.

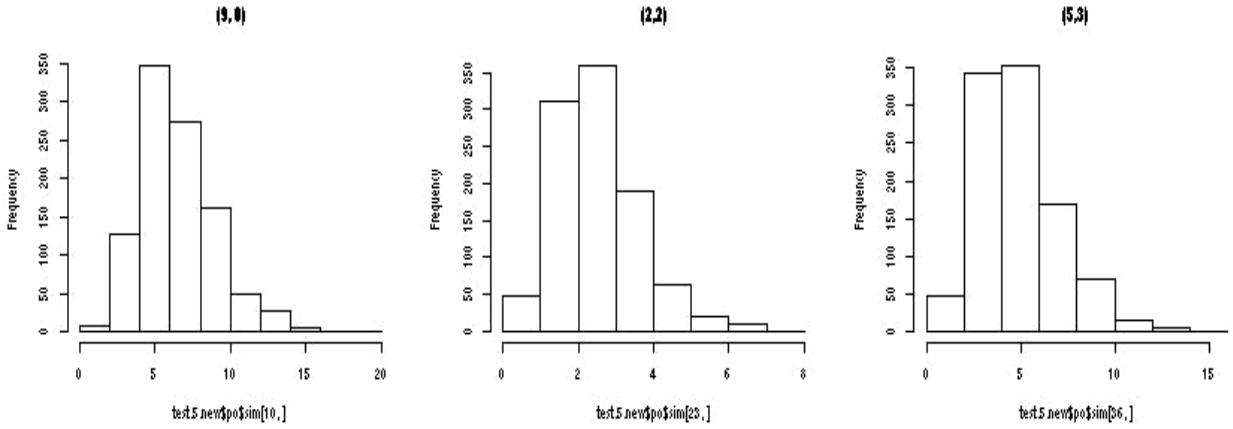
```
    mcmc5 <- mcmc.control(S.scale = 0.075, thin = 100)
    test5 <- pois.krige.bayes(y50, locations =
t(cbind(c(2.5,3),c(-6050,-3270))), prior = prior5, mcmc.input = mcmc5,
    output = list(threshold = 10, quantile = c(0.05,0.99)))
```

The output is a list which contains the five arguments `posterior`, `predictive`, `model`, `prior` and

`mcmc.input`.
The `posterior` contains information on the posterior distribution of the parameters, and the conditional simulations of the signal `g^{-1}(S)` at the data locations.
The `predictive` contains information on the predictions, where `predictive$median` is the predicted signal and `predictive$uncertainty` is the associated uncertainty.
The `threshold = 10` argument gives probabilities of the predictive distribution of the signal being less than 10 (test5$predictive$probability).
The `quantiles = c(0.05,0.99)` gives the 0.05 and 0.99 quantiles of the predictive distribution of the signal (test5$predictive$quantiles).

Below we show the simulations from the posterior distribution of the signal at a few data locations.

```
par(mfrow = c(1,3))
hist(test5$posterior$simulations[10,], main = "(9, 0)")
hist(test5$posterior$simulations[23,], main = "(2,2)")
hist(test5$posterior$simulations[36,], main ="(5,3)")
```



Now we consider an example with a random phi (the program is using a discretised prior, where the discretisation is given by the argument phi.discrete) and a positive nugget (tausq.rel = 0.05 gives the relative nugget, the relative microscale variation).

```
    mcmc6.tune <- mcmc.control(S.scale = 0.075, n.iter = 2000, thin = 100,
phi.scale = 0.01)
    prior6 <- prior.glm.control(phi.prior = "uniform", phi.discrete =
seq(0.02, 1, 0.02), tausq.rel = 0.05)
    test6.tune <- pois.krige.bayes(y50, prior = prior6, mcmc.input =
mcmc6.tune)
```

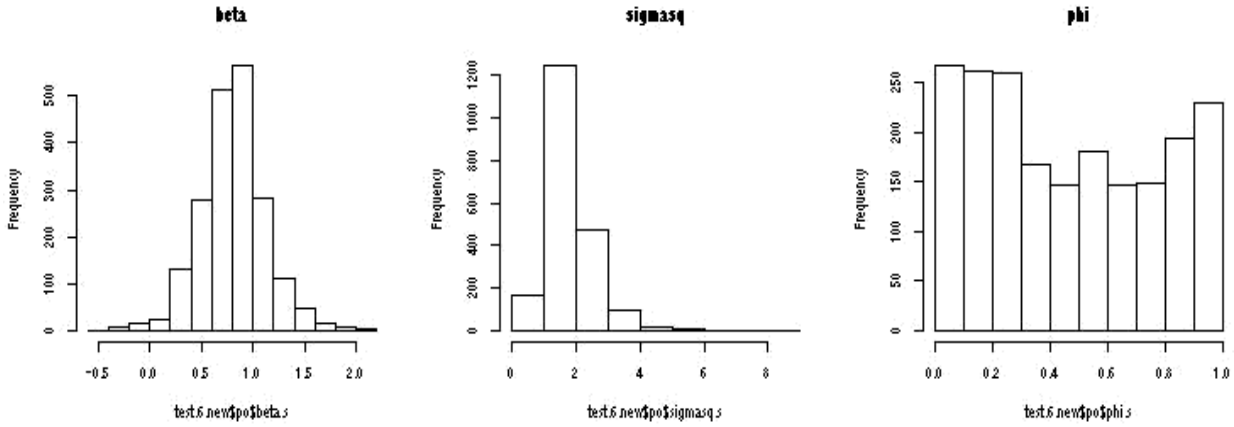
Acc-rate=0.60 , acc-rate-phi = 0.25-0.30 are preferable. After having adjusted the parameters for the MCMC algorithm and checking the output we run an analysis.

WARNING: RUNNING THE NEXT COMMAND CAN BE TIME-CONSUMING

```
    mcmc6 <- mcmc.control(S.scale = 0.075, n.iter = 400000, thin = 200,
burn.in = 5000, phi.scale = 0.12, phi.start = 0.5)
    test6 <- pois.krige.bayes(y50, locations =
t(cbind(c(2.5,3.5),c(-60,-37))), prior = prior6, mcmc.input = mcmc6)
```

Below we show the posterior distribution of the two covariance parameters and the beta parameter.

```
par(mfrow = c(1,3))
hist(test6$posterior$beta$sample, main ="beta")
hist(test6$posterior$sigmasq$sample, main = "sigmasq")
hist(test6$posterior$phi$sample, main = "phi")
```



To calculate the Monte Carlo standard errors on the posterior means of the parameters, we use the function asympvar.

```
sqrt(asympvar(test6$posterior$beta$sample)/2000)
sqrt(asympvar(test6$posterior$sigmasq$sample)/2000)
sqrt(asympvar(test6$posterior$phi$sample)/2000)
sqrt(asympvar(log(test6$posterior$simulations))/2000)
```

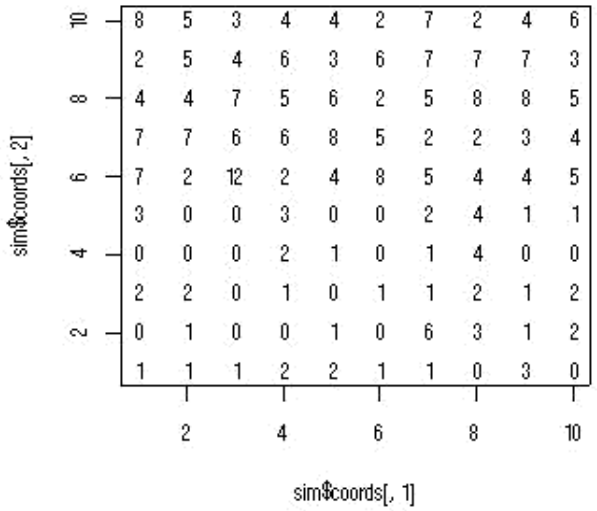
Exercise

Construct similar commands using binom.krige.bayes on the data set b50 yourself (you load the data set by typing data(b50)).

4. SIMULATION of a GENERALISED LINEAR SPATIAL MODEL

The geoR function grf generates a simulation from a Gaussian random field. This function can be used to generate a simulation from a generalised linear spatial model as follows.

```
sim <- grf(grid = expand.grid(x = seq(1, 10, l = 10), y = seq(1, 10, l = 10)), cov.pars = c(0.1, 0.2))
attr(sim,"class") <- "geodata"
sim$units.m <- c(rep(1, 50), rep(5, 50))
sim$data <- rpois(100, lambda = sim$units.m*exp(sim$data))
plot(sim$coords[,1], sim$coords[,2], type = "n")
text(sim$coords[,1], sim$coords[,2], format(sim$data))
```



Observe that the upper part of the figure corresponds to observation times equal to 5. Therefore the simulated counts are larger.

Exercise

Generate a simulation from a spatial binomial random field.

5. EXPLORATORY TOOLS

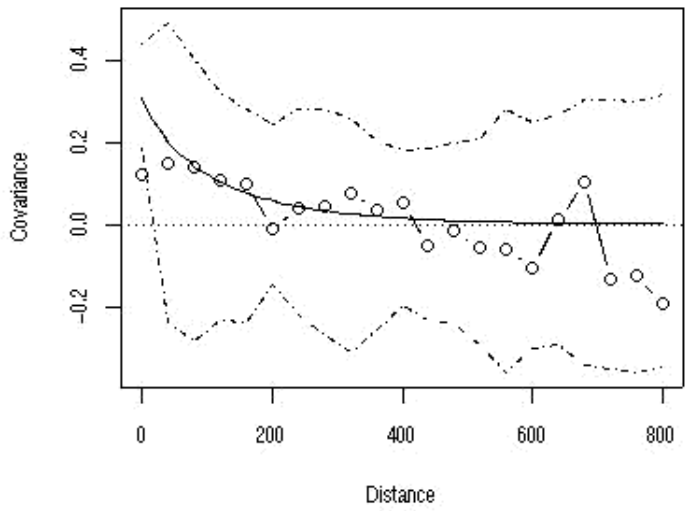
1. Empirical covariogram for the Poisson-log normal model

An empirical covariogram can be calculated using the function covariog.

```
covar <- covariog(rongelap, uvec = c((1:20)*40))
```

Theoretical and empirical variograms can be plotted and visually compared. For example, the figure below shows the estimated covariogram, a theoretical covariogram model (which in fact was estimated) and an envelope (2.5 and 97.5 percent quantiles) based on simulating the covariogram for the model having parameters given in parms.R.

```
parmR <- list(cov.model = "powered.exponential", kappa = 0.843,
cov.pars = c(0.31,6702/61.9), beta = 1.836)
class(parmR) <- "covariomodel"
konvol <- covariog.model.env(rongelap, obj.covariog = covar,
model.pars = parmR)
plot(covar, envelope.obj = konvol)
lines(parmR, max.dist = 800, lty = 1)
```



6. MISC.

A short demonstration shown at my Ph.D. viva is found here.

Site maintained by: Ole F. Christensen. (o.christensen@lancaster.ac.uk)