geoRglm : a package for generalised linear spatial models
introductory session

Ole F. Christensen & Paulo J. Ribeiro Jr.

Last update: January 31, 2004

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 user has a substantial knowledge about geostatistics, is familar with the **geoR** package (see the introductory session for **geoR**), and has a basic knowledge about Markov chain Monte Carlo methods. We also encourage the reader to study the literature about generalised linear spatial models. A list of books, articles and a short FAQ can be found here).

The objective of this page 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 - functions for geostatistical data analysis
geoR version 1.4-4 (2004-01-30) is now loaded
------------------------------------------------

------------------------------------------------------------
geoRglm - a package for generalised linear spatial models
geoRglm version 0.7-11 (2004-01-31) 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 `p50` and `rongelap` included in the **geoRglm** distribution for the examples presented in this document. These data sets can be loaded by typing:

```
> data(p50)
> data(rongelap)
```

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

```
> help(pois.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 β (OK) or flat prior on β (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.krige(p50, krige = model2, mcmc.input = list(S.scale = 0.2
+     thin = 1))
```

After a few tryouts we decide to use `S.scale = 0.5`. We also need to study how well the chain is mixing. Here the functions in the **coda** package

```
[1] TRUE
```

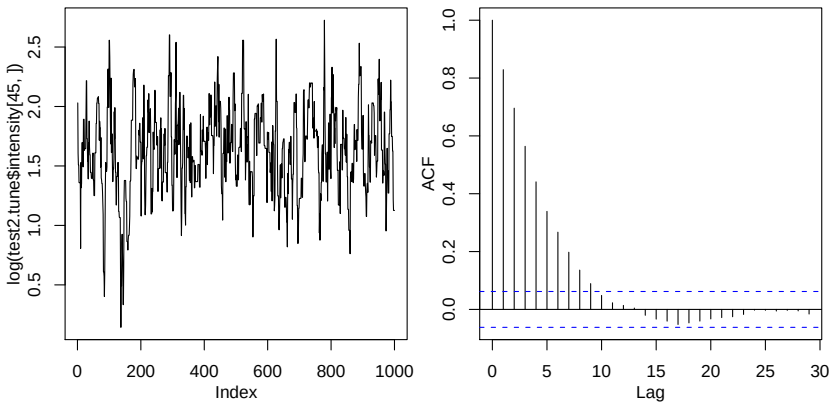

Figure 1: Results from `pois.krige`.

would be useful for assessing the convergence and inspect the mixing of the MCMC algorithm. 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 10 iterations (default); when working with real data sets we may need to make a more extensive subsampling, say, storing only every 1000 iterations.

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.krige(p50, locations = cbind(c(0.5, 0.5),
+     c(1, 0.4)), krige = model2, mcmc.input = mcmc.control(S.scale = 0.5),
+     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.

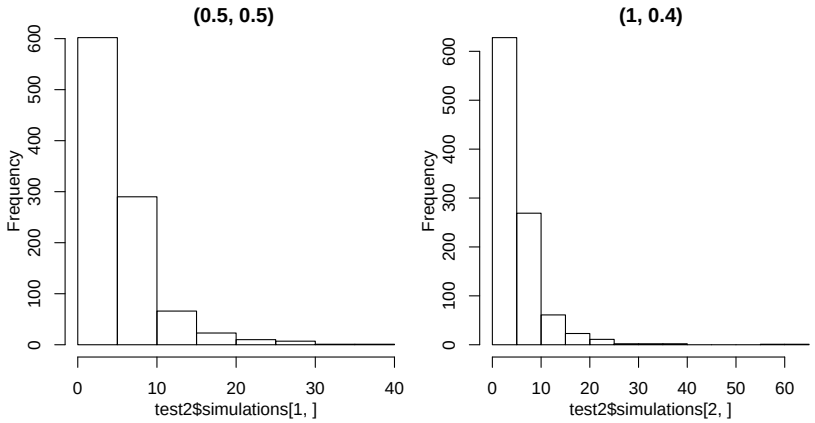

Figure 2: Histograms from 2 simulations.

The way to specify that β 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.krige(p50, krige = model2.u,
+     mcmc.input = list(S.scale = 0.5))
```

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 β and σ^2 parameters (mean and variance of the random effects S, respectively). 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(p50, 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(p50, 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)))

pois.krige.bayes: model with mean being constant
iter. numb. 1000 ; Acc.-rate = 0.67
iter. numb. 2000 ; Acc.-rate = 0.63
iter. numb. 3000 ; Acc.-rate = 0.64
iter. numb. 4000 ; Acc.-rate = 0.64
iter. numb. 5000 ; Acc.-rate = 0.64
iter. numb. 6000 ; Acc.-rate = 0.64
iter. numb. 7000 ; Acc.-rate = 0.66
iter. numb. 8000 ; Acc.-rate = 0.64
iter. numb. 9000 ; Acc.-rate = 0.63
iter. numb. 10000 ; Acc.-rate = 0.65
iter. numb. 11000 ; Acc.-rate = 0.65
iter. numb. 12000 ; Acc.-rate = 0.62
iter. numb. 13000 ; Acc.-rate = 0.63
iter. numb. 14000 ; Acc.-rate = 0.67
iter. numb. 15000 ; Acc.-rate = 0.65
iter. numb. 16000 ; Acc.-rate = 0.66
iter. numb. 17000 ; Acc.-rate = 0.67
iter. numb. 18000 ; Acc.-rate = 0.66
```

```
iter. numb. 19000 ; Acc.-rate = 0.62
iter. numb. 20000 ; Acc.-rate = 0.65
iter. numb. 21000 ; Acc.-rate = 0.64
iter. numb. 22000 ; Acc.-rate = 0.61
iter. numb. 23000 ; Acc.-rate = 0.61
iter. numb. 24000 ; Acc.-rate = 0.64
iter. numb. 25000 ; Acc.-rate = 0.63
iter. numb. 26000 ; Acc.-rate = 0.65
iter. numb. 27000 ; Acc.-rate = 0.63
iter. numb. 28000 ; Acc.-rate = 0.62
iter. numb. 29000 ; Acc.-rate = 0.65
iter. numb. 30000 ; Acc.-rate = 0.64
iter. numb. 31000 ; Acc.-rate = 0.65
iter. numb. 32000 ; Acc.-rate = 0.64
iter. numb. 33000 ; Acc.-rate = 0.65
iter. numb. 34000 ; Acc.-rate = 0.64
iter. numb. 35000 ; Acc.-rate = 0.63
iter. numb. 36000 ; Acc.-rate = 0.65
iter. numb. 37000 ; Acc.-rate = 0.63
iter. numb. 38000 ; Acc.-rate = 0.64
iter. numb. 39000 ; Acc.-rate = 0.65
iter. numb. 40000 ; Acc.-rate = 0.65
iter. numb. 41000 ; Acc.-rate = 0.67
iter. numb. 42000 ; Acc.-rate = 0.63
iter. numb. 43000 ; Acc.-rate = 0.65
iter. numb. 44000 ; Acc.-rate = 0.64
iter. numb. 45000 ; Acc.-rate = 0.65
iter. numb. 46000 ; Acc.-rate = 0.67
iter. numb. 47000 ; Acc.-rate = 0.64
iter. numb. 48000 ; Acc.-rate = 0.63
iter. numb. 49000 ; Acc.-rate = 0.63
iter. numb. 50000 ; Acc.-rate = 0.67
iter. numb. 51000 ; Acc.-rate = 0.65
iter. numb. 52000 ; Acc.-rate = 0.64
iter. numb. 53000 ; Acc.-rate = 0.64
iter. numb. 54000 ; Acc.-rate = 0.61
iter. numb. 55000 ; Acc.-rate = 0.65
iter. numb. 56000 ; Acc.-rate = 0.65
iter. numb. 57000 ; Acc.-rate = 0.61
iter. numb. 58000 ; Acc.-rate = 0.65
iter. numb. 59000 ; Acc.-rate = 0.67
```

```
iter. numb. 60000 ; Acc.-rate = 0.62
iter. numb. 61000 ; Acc.-rate = 0.67
iter. numb. 62000 ; Acc.-rate = 0.67
iter. numb. 63000 ; Acc.-rate = 0.65
iter. numb. 64000 ; Acc.-rate = 0.66
iter. numb. 65000 ; Acc.-rate = 0.62
iter. numb. 66000 ; Acc.-rate = 0.66
iter. numb. 67000 ; Acc.-rate = 0.63
iter. numb. 68000 ; Acc.-rate = 0.67
iter. numb. 69000 ; Acc.-rate = 0.64
iter. numb. 70000 ; Acc.-rate = 0.67
iter. numb. 71000 ; Acc.-rate = 0.65
iter. numb. 72000 ; Acc.-rate = 0.64
iter. numb. 73000 ; Acc.-rate = 0.65
iter. numb. 74000 ; Acc.-rate = 0.64
iter. numb. 75000 ; Acc.-rate = 0.65
iter. numb. 76000 ; Acc.-rate = 0.64
iter. numb. 77000 ; Acc.-rate = 0.65
iter. numb. 78000 ; Acc.-rate = 0.65
iter. numb. 79000 ; Acc.-rate = 0.66
iter. numb. 80000 ; Acc.-rate = 0.65
iter. numb. 81000 ; Acc.-rate = 0.63
iter. numb. 82000 ; Acc.-rate = 0.60
iter. numb. 83000 ; Acc.-rate = 0.65
iter. numb. 84000 ; Acc.-rate = 0.62
iter. numb. 85000 ; Acc.-rate = 0.65
iter. numb. 86000 ; Acc.-rate = 0.64
iter. numb. 87000 ; Acc.-rate = 0.64
iter. numb. 88000 ; Acc.-rate = 0.65
iter. numb. 89000 ; Acc.-rate = 0.69
iter. numb. 90000 ; Acc.-rate = 0.65
iter. numb. 91000 ; Acc.-rate = 0.67
iter. numb. 92000 ; Acc.-rate = 0.62
iter. numb. 93000 ; Acc.-rate = 0.61
iter. numb. 94000 ; Acc.-rate = 0.67
iter. numb. 95000 ; Acc.-rate = 0.63
iter. numb. 96000 ; Acc.-rate = 0.63
iter. numb. 97000 ; Acc.-rate = 0.64
iter. numb. 98000 ; Acc.-rate = 0.65
iter. numb. 99000 ; Acc.-rate = 0.63
iter. numb. 100000 ; Acc.-rate = 0.62
```

```
MCMC performed: n.iter. =  1e+05 ; thinning =  100 ; burn.in =  0
pois.krige.bayes: Prediction performed
pois.krige.bayes: done!
```

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.

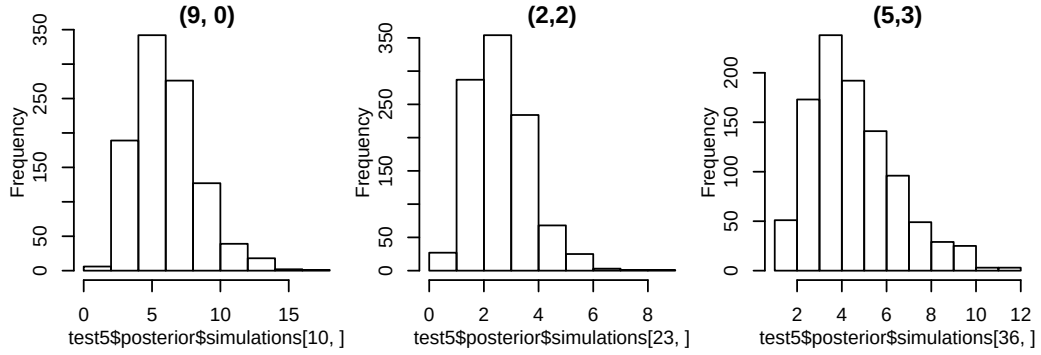

Figure 3: Histograms

Now we consider an example with a random correlation scale parameter phi and a positive nugget for the random effects S. The program is using a discretised prior fpr phi, where the discretisation is given by the argument `phi.discrete`). The argument `tausq.rel = 0.05` gives the relative nugget for S, i.e. 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(p50, 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 = 4e+05,
+     thin = 200, burn.in = 5000, phi.scale = 0.12, phi.start = 0.5)
> test6 <- pois.krige.bayes(p50, 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.

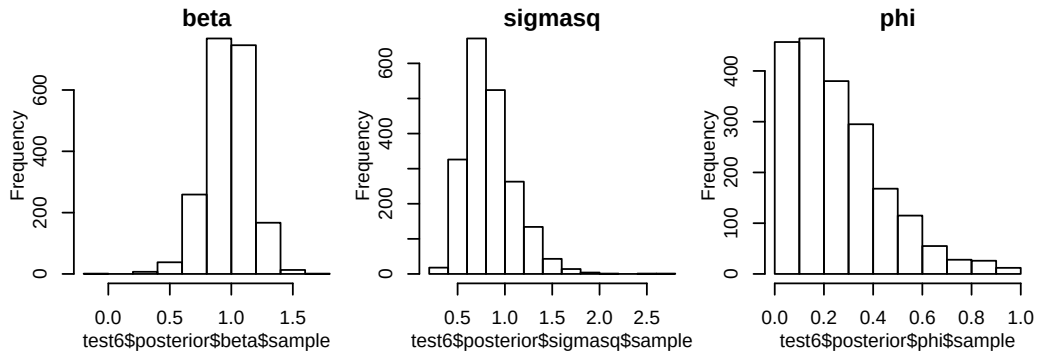

Figure 4: Samples from the posterior

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)

[1] 0.003987273

> sqrt(asympvar(test6$posterior$sigmasq$sample)/2000)

[1] 0.005988311

> sqrt(asympvar(test6$posterior$phi$sample)/2000)

[1] 0.004674492

> sqrt(asympvar(log(test6$posterior$simulations))/2000)
```

```
 [1] 0.009374290 0.012638003 0.011794051 0.014555765 0.015347837
 [6] 0.011045295 0.014465286 0.012491128 0.006027526 0.008469001
[11] 0.005531818 0.013874580 0.015976958 0.015250717 0.009199150
[16] 0.009210645 0.005880092 0.016010136 0.012233969 0.009458540
[21] 0.010637662 0.009375096 0.009345658 0.010245975 0.006142211
[26] 0.014449004 0.008758485 0.010099277 0.007804221 0.008338349
[31] 0.009768063 0.012269127 0.011481620 0.012702738 0.013745384
[36] 0.009780933 0.012138821 0.011836702 0.014334527 0.012199694
[41] 0.012523938 0.008098801 0.014382872 0.012506506 0.008908307
[46] 0.005264276 0.015292651 0.013220800 0.012310278 0.009545884
```

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))
```

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 ADDITIONAL INFORMATION

1. Empirical covariogram for the Poisson-log normal model. See here.
2. A short demonstration shown at my Ph.D. viva is found here.
3. The commands from the example in Diggle, Ribeiro Jr and Christensen (2003) [*bookchapter*], and Christensen and Ribeiro Jr (2002) [*R-news*] are found here.

```
> plot(sim$coords[, 1], sim$coords[, 2], type = "n")
> text(sim$coords[, 1], sim$coords[, 2], format(sim$data))
```

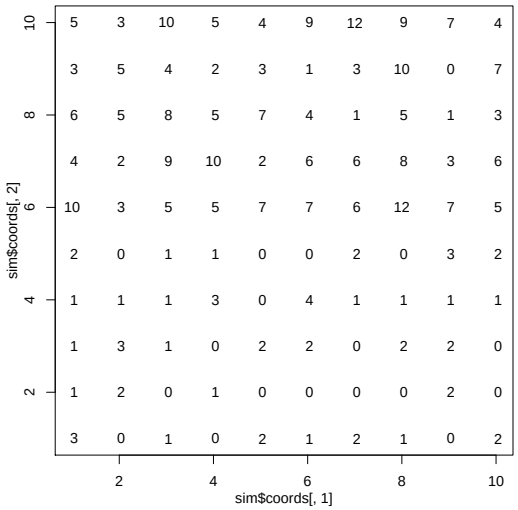

Figure 5: Simulated data