Ecography

E5171

Dormann, F. C., McPherson, J. M., Araújo, M. B., Bivand, R., Bolliger, J., Carl, G., Davies, R. G., Hirzel, A., Jetz, W., Kissling, W. D., Kühn, I., Ohlemüller, R., Peres-Neto, P. R., Reineking, B., Schröder, B., Schurr, F. M. and Wilson, R. 2007. Methods to account for spatial autocorrelation in the analysis of species distributional data: a review. – Ecography 30: 609–628.

Introduction

The publication and this supplement serve the purpose of making specific statistical methods available to ecologists. These users are usually not statisticians, and we attempt to relate sometimes rather sophisticated methodologies to the desperate analyst. However, analysing species distribution data is a tricky thing, with many potential pitfalls along the way. Neither do we attempt to address all open questions, nor will we be able to produce a cookbook recipe for all types of analyses. What we do attempt is a) a decision tree about which spatial autocorrelation modelling method to use when, and b) software implementation aids for these methods. We opted for using the software package R (www.r-project.org), which is extremely flexible, versatile - and free.

The following pages cannot be understood without some advanced statistical knowledge or without the paper this code accompanies. We assume a basic understanding of generalised linear models for non-normally distributed data. Most details on the methods are provided in the main paper, while these pages are primarily for implementing the methods. The person mainly responsible for the implementation of that method is given in the parentheses in the section title.

Decision tree

The first partition in our decision tree is dictated by the type of response variable to be analysed. More methods are available for data derived from a normal distribution (data whose residuals are normally distributed) than data of alternative distributional form (e.g. binomial, Poisson). Typical examples of ecological data with normally distributed errors include abundance, species richness, or functional diversity per unit area, crop yield and catch per unit effort. Examples of data following a binomial distribution are the presence or absence of a species and the success or failure of seed germination. In contrast, whale sightings, counts of herd size or the number of ectoparasites per sheep are data more likely to follow a Poisson (or negative binomial) distribution.

The second partition refers to computational efficiency. Some methods are so computer-intensive that they may not be suitable for use with very large data sets.

method	residuals	computational intensity
GAM	normal, Poisson, binomial	low
autogressive models (SAR/CAR)	normal	medium-high
GLS	normal	medium-high
GEE	normal, Poisson, binomial	low
autocovariate regression	normal, Poisson, binomial	low
spatial GLMM	normal, Poisson, binomial	very high
Spatial Eigenvector Mapping	normal, Poisson, binomial	very high

1.1. Preparing the data and non-spatial analysis (Carsten F. Dormann)

All following analyses are illustrated using data organized as an XYZ-table (or, in the R nomenclature), data.frame. X and Y are the spatial coordinates, while Z is the response variable. Additionally, we have explanatory variables, here "rain" and "djungle". The data we used for the analysis are available as supplementary material ("snouterdata.txt").

To get into the right mood, we start with *non-spatial* models, both for normally and non-normally distributed residuals:

Reading the data

```
#set your path here!
setwd <- "C:/Data/..."
#read in data:
snouter.df <- read.table("snouterdata.txt", header=T, sep="\t")
```

Run non-spatial models

GLMs

```
summary(ols1 <- lm(snouter1.1 ~ rain + djungle, data=snouter.df))
summary(binom1 <- glm(snouter2.1 ~rain+djungle, family=binomial,
data=snouter.df))
summary(pois1 <- glm(snouter3.1 ~rain+djungle, family=poisson,
data=snouter.df))
```

GAMs (Janine Bolliger & Ralf Ohlemüller)

There are two commonly used GAM-packages in R: **mgcv** (by Simon Woods) and **gam** (by Trevor Hastie). They are different, since there is no single definition of what a GAM is, but they are also similar (see the comparison in Faraway's book: Extending the Linear Model). Here, we use **mgcv**. We use the spline only for the geographical component, treating rain and djungle as fixed effects. This practically amounts to a trend-surface regression with rain and djungle as covariates.

```
library(mgcv)
summary(gam.norm <- gam(snouter1.1 ~ rain + djungle + s(X, Y),
data=snouter.df, family=gaussian))
summary(gam.bino <- gam(snouter2.1 ~ rain + djungle + s(X, Y),
data=snouter.df, family=binomial))
summary(gam.pois <- gam(snouter3.1 ~ rain + djungle + s(X, Y),
data=snouter.df, family=poisson))
```

The degree of spatial autocorrelation in the residuals can be assessed using correlograms. They depict Moran's across all distance classes.

Plotting/calculating spatial autocorrelation

```
# First, install the library ncf (http://onb.ent.psu.edu/onb1/);
# for windows:
install.packages("ncf",contriburl="http://asi23.ent.psu.edu/onb1/R/windows"
)
# for linux:
#install.packages("ncf",contriburl="http://asi23.ent.psu.edu/onb1/R/src")
require(ncf)
?correlog
model <- ols1 # or binom1 or pois1
correlog1.1 <- correlog(snouter.df$X, snouter.df$Y, residuals(model),
na.rm=T, increment=1, resamp=0)

# now plot only the first 20 distance classes:
par(mar=c(5,5,0.1, 0.1))
plot(correlog1.1$correlation[1:20], type="b", pch=16, cex=1.5, lwd=1.5,
xlab="distance", ylab="Moran's I", cex.lab=2, cex.axis=1.5); abline(h=0)

# make a map of the residuals:
plot(snouter.df$X, snouter.df$Y, col=c("blue",
"red")[sign(resid(model))/2+1.5], pch=19,
cex=abs(resid(model))/max(resid(model))*2, xlab="geographical x-
coordinates", ylab="geographical y-coordinates")

# calculate Moran's I values explicitly for a certain distance,
# and to test for its significance:
require(spdep)
snouter.nb <- dnearneigh(as.matrix(snouter.df[1:2]), 0, 20) #give lower and
upper distance class here!
snouter.listw <- nb2listw(snouter.nb) #turns neighbourhood object into a
weighted list
#this next step takes often several minutes to run:
GlobMT1.1<- moran.test(residuals(model), listw=snouter.listw)
```

Now we are set to start with the spatial analysis.

1.2. Methods for normally distributed residuals

The best established methods here are autoregressive models and Generalized Linear Models. Spatial filtering is a very new method. For details on GEE and autocovariate regression models see next section.

Autoregressive Models in R

AR are implemented in the library **spdep**. Several functions can be invoked for the regression itself, depending on which assumptions are made about the cause of spatial autocorrelation (errorsarlm, lagsarlm, spautolm). A comparison of these different autoregressive models is very advisable, either using model selection procedures (e.g. Kissling & Carl 2007) or the Lagrange multiplier test (see SAR below).

Simultaneous Autoregressive Models (SAR) (W. Daniel Kissling)

1. spatial SAR error model ("SARerr"): function `errorsarlm()`
2. spatial SAR lag model ("SARlag"): function `lagsarlm()` with `type="lag"`
3. spatial SAR mixed model ("SARmix"): function `lagsarlm()` with `type="mixed"`

All SAR models require normally distributed errors. To use the SAR functions, one first needs to specify a spatial weights matrix which is to be incorporated in the SAR functions as a "listw" object. This spatial weights matrix is constructed by first defining the neighbourhood with the dnearneigh() function and then weighting the neighbours with the nb2listw() function by choosing a certain coding scheme (e.g., "B" = binary coding, "W" = row standardised, or "S" = variance-stabilising coding scheme).
Here we only use a neighbourhood distance of 1.5 and a coding scheme "W", but other spatial weights matrices can be defined and implemented. The Lagrange Multiplier diagnostics function "lm.LMtests()") is used to compare the different SAR model types. Rather than entering the regression formula directly, we here call a previous non-spatial model (ols1) for this. A model selection procedure can also be used (Kissling & Carl 2007).

```
require(spdep)

# Data preparation
snouter.df <- read.table("snouterdata.txt", header=T, sep="\t")

# Define coordinates, neighbours, and spatial weights
coords<-as.matrix(cbind(snouter.df$X,snouter.df$Y))

#Define neighbours up to a certain distance
nb1.5<-dnearneigh(coords,0,1.5) #you can use e.g. distance 1 or 2 instead

#Spatial weights
nb1.5.w<-nb2listw(nb1.5, glist=NULL, style="W", zero.policy=FALSE)

#1. Spatial SAR error model
sem.nb1.5.w <- errorsarlm(ols1, listw=nb1.5.w)
summary(sem.nb1.5.w) #Gives a summary of the SAR error model

# 2. Spatial SAR lag model
slm.nb1.5.w <- lagsarlm(ols1, listw=nb1.5.w, type="lag")
summary(slm.nb1.5.w) #Gives a summary of the SAR lag model

# 3. Spatial SAR mixed model
smm.nb1.5.w <- lagsarlm(ols1, listw=nb1.5.w, type="mixed")
summary(smm.nb1.5.w) #Gives a summary of the SAR mixed model


# Lagrange multiplier diagnostics for model comparison
#To test which model is most appropriate:

lm.LMtests(lm1, nb1.5.w, test="all")
```

Conditional Autoregressive Models (CAR) (Boris Schröder)

```
require(spdep)

#Make a matrix of coordinates
coords<-as.matrix(cbind(data$X,data$Y))
```

```
#Define neighbourhood
nb<-dnearneigh(coords, 0, 2) # 2nd order neighbourhood

# define spatial weights matrix
w<-nb2listw(nb, style="W", zero.policy=FALSE)

# specify and run CAR model

cem<-spautolm(snouter1.1 ~ rain + djungle, data=snouter.df, listw=w,
family="CAR") # CAR error model
summary(cem)
```

Generalized Least Square Models in R (Björn Reineking)

GLS are fitted using the function gls {**nlme**} or gls.fit {**MASS**}. Internally, also SAR and CAR methods call one of them. gls {**nlme**} offers to specify the expected form of autocorrelation in the correlation argument. With the correlation option in the gls-call, the different spatial correlation structures can be specified.

```
require(nlme)
?gls #for syntax help
?corClasses #for help with correlation functions available

summary(gls.exp <- gls(snouter1.1 ~ rain + djungle, data=snouter.df,
correlation=corExp(form=~X+Y)))
summary(gls.gauss <- gls(snouter1.1 ~ rain + djungle, data=snouter.df,
correlation=corGaus(form=~X+Y)))
summary(gls.spher <- gls(snouter1.1 ~ rain + djungle, data=snouter.df,
correlation=corSpher(form=~X+Y)))
```

1.3. Methods also for non-normally distributed residuals (e.g. Poisson or binomial)

Autocovariate regression in R (Jana McPherson)

Autocovariate regression was conceived for binary data (as "autologistic regression") by Augustin et al. (1996). Here we offer the identical methodology to be extended to any type of data, since the logic behind the approach is independent of the error distribution. Note, however, that to date only autologistic regressions have found their way into literature, and we cannot warrant for the statistical correctness of this approach. In R, a function autocov_dist {**spdep**} provides the means to create a new explanatory variable which is entered in addition to the other explanatory variables in the model.

```
require(spdep)
?autocov_dist

# prepare neighbour lists for spatial autocorrelation analysis
nb.list <- dnearneigh(as.matrix(snouter.df[,c("X", "Y")]), 0, 5)
nb.weights <- nb2listw(nb.list)

#Make a matrix of coordinates
coords<-as.matrix(cbind(data$X,data$Y))

# compute the autocovariate based on the above distance and weight
ac <- autocov_dist(snouter1.1,  coords,  nbs = 1,  type = "inverse")
```

```
# now run a linear model with the autocovariate as additional explanatory
variable:
fm <- lm(snouter1.1 ~ rain + djungle + ac,  data=snouter.df)
summary(fm)
```

Generalized Estimation Equations in R (Gudrun Carl)

Two different GEE packages are available in R: gee {**gee**} and geese {**geepack**}. Data preparation for the analysis is a significant part of the exercise, while the models themselves work fast and efficient. The following code and helper functions shall aid with the data preparation.

Copy the following code (starting and ending with #%%%%%%%%%%) into a text-file and save as "geefunctions.R". It is also provided as supplementary material, which you can include into an R-session writing: `source("geefunctions.R")`.

```
#%%%%%%%%%%
# Helper functions for the spatial usage of gee and geepack
# Code written by Gudrun Carl, 2005

##############################################################

########################################################################
dat.nn<-function(data,n){
########################################################################
# A function to generate clusters and order variables and
# to produce a data frame with response, predictors, coordinates, and
# 3 new parameters:
# o for order
# id for identifying clusters and
# waves for identifying members of clusters
#
# Arguments
# data      a data frame with response and predictors and
#           in the last columns with ordered cartesian coordinates
# n         for maximal cluster size  n*n
########################################################################

l<-dim(data)[2]
OST<-data[,l-1]
NORD<-data[,l]
ko<-OST-min(OST)
idx<-(ko-(ko%%(n)))/n+1
ks<-NORD-min(NORD)
idy<-(ks-(ks%%(n)))/n+1
ie<-(idy-1)*max(idx)+idx
idwx<-ko%%(n)+1
idwy<-ks%%(n)+1
wav<-(idwy-1)*n+idwx
data<-as.matrix(data)
o<-order(ie,wav)
id<-ie[o]
waves<-wav[o]
dat.new1<-data[o,]
dat.new2<-cbind(dat.new1,o,id,waves)
dat.new<-as.data.frame(dat.new2)
}
```

```
##########################################################################
a.gee<-function(mgee,n,type="glm",corstr="independence",quad=T) {
##########################################################################
# A function to order correlation parameters of Generalized Estimating
# Equation Models
# Arguments
# mgee       matrix or vector of correlation parameters according to model
# n          for maximal cluster size n*n
# type       type of model
#            "glm", "gee", "geese" are allowed
# corstr     correlation structure
#            "independence", "exchangeable", "userdefined" are allowed
# quad       by default quadratic correlation structure
#            for model "geese" and "userdefined" correlation only
##########################################################################

if(n==2)n3<-6
if(n==3)n3<-36
if(n==4)n3<-120
a<-rep(0,n3)
if(type=="glm") a<-a
if(type=="gee"){
  if(corstr=="exchangeable") a[c(1:n3)]<-mgee[1,2]
  if(corstr=="independence") a<-a
}
a<-as.vector(a)

if(type=="geese") {
 if(corstr=="userdefined"){
if(quad) {
if(n==2)  {
 a<-rep(0,6)
 a[c(1,2,5,6)]<-mgee[1]
 a[c(3,4)]<-mgee[2]
}
if(n==3)  {
 a<-rep(0,36)
 a[c(1,3,9,11,18,22,24,27,29,33,34,36)]<-mgee[1]
 a[c(2,6,14,21,23,35)]<-mgee[2]
 a[c(4,10,12,17,25,28,30,32)]<-mgee[3]
 a[c(5,7,13,15,16,20,26,31)]<-mgee[4]
 a[c(8,19)]<-mgee[5]
}
if(n==4)  {
 a<-rep(0,120)
 a[c(1,4,16,19,30,33,46,55,58,66,69,76,79,88,93,96,100,103,106,109,
114,115,118,120)]<-mgee[1]
 a[c(2,8,17,23,37,50,56,62,67,73,83,92,94,101,116,119)]<-mgee[2]
 a[c(3,12,27,41,54,57,95,117)]<-mgee[3]
 a[c(5,18,20,32,34,45,59,68,70,78,80,87,97,102,104,108,110,113)]<-mgee[4]
 a[c(6,9,21,22,24,31,36,38,44,49,60,63,71,72,74,77,82,84,86,91,98,
105,107,112)]<-mgee[5]
 a[c(7,13,26,28,40,42,43,53,61,85,99,111)]<-mgee[6]
 a[c(10,25,35,48,64,75,81,90)]<-mgee[7]
 a[c(11,14,29,39,47,52,65,89)]<-mgee[8]
 a[c(15,51)]<-mgee[9]
}}
if(!quad) a<-mgee
}
if(corstr=="exchangeable") a[c(1:n3)]<-mgee
if(corstr=="independence") a<-a
```

```
 }
a<-as.vector(a)
}

#########################################################################
clus.sz<-function(id){
#########################################################################
# A function to calculate sizes of clusters
# Argument
# id     vector which identifies the clusters
#########################################################################

clus<-rep(0,length(id))
k0<-0
k1<-1
for(i in 2:length(id)) { i1<-i-1
if(id[i]==id[i1]) {k1<-k1+1
if(i==length(id)) {k0<-k0+1
                   clus[k0]<-k1}}
if(id[i]!=id[i1]) {k0<-k0+1
                   clus[k0]<-k1
                   k1<-1
if(i==length(id)) {k0<-k0+1
                   clus[k0]<-k1 }}}
clusz<-clus[clus>0]
}

#########################################################################
zcor.quad<-function(zcor,n,quad=TRUE) {
#########################################################################
# A function to create a quadratic correlation structure
# zcor    an object of class "genZcor" (see: geepack)
# n       for maximal cluster size n*n
# quad    by default quadratic correlation structure
#########################################################################

if(quad) {
if(n==2)  {
 zcorn<-matrix(0,dim(zcor)[1],2)
 zcorn[,1]<-zcor[,1]+zcor[,2]+zcor[,5]+zcor[,6]
 zcorn[,2]<-zcor[,3]+zcor[,4]
}
if(n==3)  {
 zcorn<-matrix(0,dim(zcor)[1],5)
 zcorn[,1]<-zcor[,1]+zcor[,3]+zcor[,9]+zcor[,11]+zcor[,18]+zcor[,22]+
 zcor[,24]+zcor[,27]+zcor[,29]+zcor[,33]+zcor[,34]+zcor[,36]
 zcorn[,2]<-zcor[,2]+zcor[,6]+zcor[,14]+zcor[,21]+zcor[,23]+zcor[,35]
 zcorn[,3]<-zcor[,4]+zcor[,10]+zcor[,12]+zcor[,17]+zcor[,25]+zcor[,28]+
 zcor[,30]+zcor[,32]
 zcorn[,4]<-zcor[,5]+zcor[,7]+zcor[,13]+zcor[,15]+zcor[,16]+zcor[,20]+
 zcor[,26]+zcor[,31]
 zcorn[,5]<-zcor[,8]+zcor[,19]
}
if(n==4)  {
 zcorn<-matrix(0,dim(zcor)[1],9)
  zcorn[,1]<-zcor[,1]+zcor[,4]+zcor[,16]+zcor[,19]+zcor[,30]+zcor[,33]+
zcor[,46]+zcor[,55]+zcor[,58]+zcor[,66]+zcor[,69]+zcor[,76]+
zcor[,79]+zcor[,88]+zcor[,93]+zcor[,96]+zcor[,100]+zcor[,103]+
zcor[,106]+zcor[,109]+zcor[,114]+zcor[,115]+zcor[,118]+zcor[,120]
  zcorn[,2]<-zcor[,2]+zcor[,8]+zcor[,17]+zcor[,23]+zcor[,37]+zcor[,50]+
zcor[,56]+zcor[,62]+zcor[,67]+zcor[,73]+zcor[,83]+zcor[,92]+
zcor[,94]+zcor[,101]+zcor[,116]+zcor[,119]
  zcorn[,3]<-zcor[,3]+zcor[,12]+zcor[,27]+zcor[,41]+zcor[,54]+zcor[,57]+
zcor[,95]+zcor[,117]
  zcorn[,4]<-zcor[,5]+zcor[,18]+zcor[,20]+zcor[,32]+zcor[,34]+zcor[,45]+
zcor[,59]+zcor[,68]+zcor[,70]+zcor[,78]+zcor[,80]+zcor[,87]+
zcor[,97]+zcor[,102]+zcor[,104]+zcor[,108]+zcor[,110]+zcor[,113]
  zcorn[,5]<-zcor[,6]+zcor[,9]+zcor[,21]+zcor[,22]+zcor[,24]+zcor[,31]+
zcor[,36]+zcor[,38]+zcor[,44]+zcor[,49]+zcor[,60]+zcor[,63]+
zcor[,71]+zcor[,72]+zcor[,74]+zcor[,77]+zcor[,82]+zcor[,84]+
zcor[,86]+zcor[,91]+zcor[,98]+zcor[,105]+zcor[,107]+zcor[,112]
  zcorn[,6]<-zcor[,7]+zcor[,13]+zcor[,26]+zcor[,28]+zcor[,40]+zcor[,42]+
zcor[,43]+zcor[,53]+zcor[,61]+zcor[,85]+zcor[,99]+zcor[,111]
  zcorn[,7]<-zcor[,10]+zcor[,25]+zcor[,35]+zcor[,48]+zcor[,64]+zcor[,75]+
zcor[,81]+zcor[,90]
  zcorn[,8]<-zcor[,11]+zcor[,14]+zcor[,29]+zcor[,39]+zcor[,47]+zcor[,52]+
zcor[,65]+zcor[,89]
  zcorn[,9]<-zcor[,15]+zcor[,51]
}
}
if(!quad) zcorn<-zcor
zcorn<-as.matrix(zcorn)
}

########################################################################
res.gee<-function(formula,data,n,clusz=NA,zcor=NA,a=NA,b,R=NA,
fam="b",type="resid")  {
########################################################################
# A function to calculate fitted values and residuals
# for Generalized Estimating Equation Models
# for gaussian or binary data (with logit link) or Poisson data (log link)
# Arguments
# formula     a formula expression
# data        a data frame
# n           for maximal cluster size n*n
# clusz       an object of class "clus.sz"
# zcor        an object of class "genZcor"
# a           a vector of correlation parameters
#             for clusters only
#             as an object of class "a.gee"
# b           a vector of regression parameters beta
# R           a square matrix of correlation parameters
#             for full dimension (=number of observations)  only
# fam         family
#             "g", "b", "p" (gaussian, binary, Poisson) are allowed
# type        "fitted" calculates fitted values
#             "resid" calculates residuals
#
########################################################################

l<-dim(data)[2]
ieo<-data[,l-1]
if(n!=dim(data)[1]) {
n2<-n*n
n3<-n2*(n2-1)/2
n4<-n2-1
n5<-n2-2
for(i in 1:dim(zcor)[1]){
for(k in 1:n3){
if(zcor[i,k]==1) zcor[i,k]<-a[k]  }}
lc<-length(clusz)
z2<-matrix(0,lc,n3)
```

```
for( j in 1:n3) {
k3<-0
k2<-0
for(i in 1:lc) {
if(clusz[i]!=1) {
k2<-k3+1
k3<-clusz[i]*(clusz[i]-1)/2+k3
for(k in k2:k3)
z2[i,j]<-zcor[k,j]+z2[i,j] }}}
if(n==2)
iod<-c(1,1,2)
if(n==3)
iod<-c(1,1,1,1,1,1,1,2,2,2,2,2,2,3,3,3,3,3,4,4,4,4,5,5,5,6,6,7)
if(n==4)
iod<-c(1,1,1,1,1,1,1,1,1,1,1,1,1,1,2,2,2,2,2,2,2,2,2,2,2,2,2,
3,3,3,3,3,3,3,3,3,3,3,3,4,4,4,4,4,4,4,4,4,4,4,5,5,5,5,5,5,5,5,5,5,
6,6,6,6,6,6,6,6,6,7,7,7,7,7,7,7,7,8,8,8,8,8,8,8,9,9,9,9,9,9,10,10,10,10,10,
11,11,11,11,12,12,12,13,13,14)
cs<-0
v<-matrix(0,length(ieo),length(ieo))
vgl<-rep(0,n2)
for(i in 1:lc) {clu<-clusz[i]
if(clu!=1) {
v1<-matrix(0,n2,n2)
if(n==2)
{  v1[1,2:4]<-z2[i,1:3]
   v1[2,3:4]<-z2[i,4:5]
   v1[3,4]<-z2[i,6]  }
if(n==3)
{  v1[1,2:9]<-z2[i,1:8]
   v1[2,3:9]<-z2[i,9:15]
   v1[3,4:9]<-z2[i,16:21]
   v1[4,5:9]<-z2[i,22:26]
   v1[5,6:9]<-z2[i,27:30]
   v1[6,7:9]<-z2[i,31:33]
   v1[7,8:9]<-z2[i,34:35]
   v1[8,9]<-z2[i,36]  }
if(n==4)
{  v1[1,2:16]<-z2[i,1:15]
   v1[2,3:16]<-z2[i,16:29]
   v1[3,4:16]<-z2[i,30:42]
   v1[4,5:16]<-z2[i,43:54]
   v1[5,6:16]<-z2[i,55:65]
   v1[6,7:16]<-z2[i,66:75]
   v1[7,8:16]<-z2[i,76:84]
   v1[8,9:16]<-z2[i,85:92]
   v1[9,10:16]<-z2[i,93:99]
   v1[10,11:16]<-z2[i,100:105]
   v1[11,12:16]<-z2[i,106:110]
   v1[12,13:16]<-z2[i,111:114]
   v1[13,14:16]<-z2[i,115:117]
   v1[14,15:16]<-z2[i,118:119]
   v1[15,16]<-z2[i,120]   }
for(i1 in 1:length(iod)) {
i2<-iod[i1]
if(var(v1[i2,1:n2])==0) {for(k in i2:n5) {k1<-k+1
                                          v1[k,]<-v1[k1,]
                                          v1[k1,]<-vgl[]}}}
for(i1 in 1:length(iod)){
i3<-iod[i1]+1
if(var(v1[1:n2,i3])==0) {for(k in i3:n4) {k1<-k+1
```

```
                                          v1[,k]<-v1[,k1]
                                          v1[,k1]<-vgl[]}}}

clu1<-clu-1
for(k in 1:clu1) {csk<-cs+k
f1<-2
for(k1 in f1:clu) {k2<-cs+f1
v[csk,k2]<-v1[k,k1]
f1<-f1+1 }}
for(k in 1:clu) {csk<-cs+k
v[csk,csk]<- 0.5 } }
if(clu==1) {cs1<-cs+1
v[cs1,cs1]<-0.5 }
cs<- cumsum(clusz)[i]  }
v<-v+t(v)
}
if(n==dim(data)[1]) v<-R
ww<-solve(v)

s.geese<-svd(ww,LINPACK=T)
d.geese<-diag(sqrt(s.geese$d))
w<-s.geese$u%*%d.geese%*%t(s.geese$u)

x.matrix<-model.matrix(formula)
fitted<-x.matrix%*%b
fitted<-fitted[1:length(ieo)]
if(fam=="p") fitted<-exp(fitted)
if(fam=="b") fitted<-exp(fitted)/(1+exp(fitted))

if(type=="fitted")resgeeseo<-fitted
if(type=="resid"){
if(fam=="g") rgeese<-data[,1]-fitted
if(fam=="p") rgeese<-(data[,1]-fitted)/sqrt(fitted)
if(fam=="b") rgeese<-(data[,1]-fitted)/sqrt(fitted*(1-fitted))
rsgeese<-w%*%rgeese
resgeeseo<-rsgeese[1:length(ieo)]
}

resgeeseo<-as.vector(resgeeseo)
}


#####################################################

#%%%%%%%%%%


# here the real analysis with GEE starts:
# define dependent variable "snouter" (response)
response <- "snouter1.1"

# load necessary libraries and codes
require(gee)
require(geepack)
source("geefunctions.R") # attached as text file


# GEE -----------------------------------------------
# gee model with fixed correlation structure

# first prepare a dataframe
```

```
data <-
data.frame(snouter.df[,c("snouter1.1",names(snouter.df[c(3:4,1:2)]))])
attach(data)
nn <- nrow(data)
coord <- cbind(X, Y)

# next compute glm model:
mglm <- glm(snouter1.1 ~ rain + djungle, family=gaussian, data=data)
resglm <- resid(mglm)
corglm <- correlog(X, Y, resglm, na.rm=T, increment=1,
resamp=1,legacy=FALSE)

# fit of autocorrelation for gee fixed model
alpha <- corglm$correlation[1]
idn <- rep(1,nn)
D <- as.matrix(dist(coord))
R <- alpha^D

# then gee model
mgee <- gee(snouter1.1 ~ rain + djungle, family=gaussian, data=data,
id=idn, R=R, corstr="fixed")
summary(mgee)[1:7]

# residuals and fitted values extracted by homemade functions
resgee <-res.gee(snouter1.1 ~ rain + djungle, data=data, nn, b=mgee$coeff,
R=R, fam="g", type="resid")
fitgee <-res.gee(snouter1.1 ~ rain + djungle, data=data, nn, b=mgee$coeff,
R=R, fam="g", type="fitted")
detach(data)

# GEESE ------------------------------------------------
# geese model with user defined correlation structure
# cluster size, nr=3 --> 3*3 cells in a cluster
nr <- 3

# first prepare a new dataframe
data.cluster <- dat.nn(snouter.df[,c("snouter1.1",
names(snouter.df[c(3:4,1:2)]))], n=nr)
attach(data.cluster)

# prepare cluster sizes, waves within clusters and quadratic correlation
structure
# by functions from geepack and homemade functions
clusz <- clus.sz(id)
zcor <- genZcor(clusz=clusz, waves=waves, corstrv="unstructured")
zcorq <- zcor.quad(zcor, n=nr, quad=T)

# then geese model
mgeese <- geese(snouter1.1 ~ rain + djungle, family=gaussian,
data=data.cluster, id=id, corstr="userdefined", zcor=zcorq)
summary(mgeese)

# residuals and fitted values extracted by homemade functions
ageese <- a.gee(mgeese$a,n=nr,type="geese",corstr="userdefined",quad=T)
resgeese.cluster <- res.gee(snouter1.1 ~ rain + djungle,
data=data.cluster,n=nr,clusz,zcor,ageese,mgeese$b,fam="g",type="resid")
fitgeese.cluster <- res.gee(snouter1.1 ~ rain + djungle,
data=data.cluster,n=nr,clusz,zcor,ageese,mgeese$b,fam="g",type="fitted")
resgeese <- resgeese.cluster[order(o)]
fitgeese <- fitgeese.cluster[order(o)]
detach(data.cluster)
```

Spatial Generalized Linear Mixed Model in R (Frank Schurr)

This is an inofficial abuse of a Generalized Linear Mixed Model function (glmmPQL {**MASS**}), which is a wrapper function for lme {**nlme**}, which in turn internally calls gls {**nlme**}. Hence the presented method is actually a generalization of gls to non-normal data. It requires a "cheat" to make it do what we want, however. This code produces the identical results as an official spatial GLMM in SAS (proc glimmix) and can hence be trusted.

```
require(MASS)
?glmmPQL
?corClasses

snouter.df <- read.table("snouterdata.txt", head=T)

#define a grouping factor that assigns all observations to the same group
group <- factor(rep("a",nrow(snouter.df)))
snouter.df <- cbind(snouter.df, group)


attach(snouter.df) #For some reason, the data have to be attached AND
specified in the formula!
# GLMM fits ----------------------
#exponential correlation structure

model.e <- glmmPQL(snouter2.1 ~ rain + djungle, random=~1|group,
data=snouter.df,
correlation=corExp(form=~X+Y), family=binomial))

#Gaussian correlation structure
model.g <- glmmPQL(snouter2.1 ~ rain + djungle, random=~1|group,
data=snouter.df, correlation=corGaus(form=~X+Y), family=binomial))

#spherical correlation structure
model.s <- glmmPQL(snouter2.1 ~ rain + djungle, random=~1|group,
data=snouter.df, correlation=corSpher(form=~X+Y), family=binomial))

detach(snouter.df)
```

Spatial Eigenvector Mapping in R (Pedro Peres-Neto)

This method has only recently seen the light of day. Two steps are involved in R: (1) setting up the correct neighbourhood, (2) running the acutal eigenvector generator and selector. You may also want to check ?SpatialFiltering which "only" corrects standard errors of an OLS but does not affect parameter estimates.

```
require(spdep)
?ME

snouter_sp <- SpatialPixelsDataFrame(as.matrix(snouter.df[,2:1]),
snouter.df)
nb1.0 <- dnearneigh(coordinates(snouter_sp), 0, 1.0)
nb1.0_dists <- nbdists(nb1.0, coordinates(snouter_sp))
nb1.0_sims <- lapply(nb1.0_dists, function(x) (1-((x/4)^2)) )
ME.listw <- nb2listw(nb1.0, glist=nb1.0_sims, style="B")
```

```
sevm1 <- ME(snouter1.1 ~ rain + djungle, data=snouter.df, family=gaussian,
listw=ME.listw)

# modify the arguments "family" according to your error distribution
```

Text file: <http://www.oikos.ekol.lu.se/appendixdown/snouterdata.txt>.
Text file: <http://www.oikos.ekol.lu.se/appendixdown/geefunctions.R>.