```
#
   ################################################################################
## code for generating and analyzing samples by the procedures implemented in
   spaMM ##
#
   ################################################################################

# the code assumes that the spaMM package has been installed from CRAN
library(spaMM)

if(interactive()) options(error=recover)

# mvrnorm code has changed between R 2.15.1 and 2.15.2.
# This version aims to reproduce the behaviour of the old version
# for consistency between different simulation tests.
mvrnorm <- function (n = 1, mu, Sigma, tol = 1e-06, empirical = FALSE)
{
    p <- length(mu)
    if (!all(dim(Sigma) == c(p, p)))
        stop("incompatible arguments")
    eS <- suppressWarnings(eigen(Sigma, symmetric = TRUE,EISPACK=T))  ##
        suppressWarnings
    ev <- eS$values
    if (!all(ev >= -tol * abs(ev[1L])))
        stop("'Sigma' is not positive definite")
    X <- matrix(rnorm(p * n), n)
    if (empirical) {
        X <- scale(X, TRUE, FALSE)
        X <- X %*% svd(X, nu = 0)$v
        X <- scale(X, FALSE, TRUE)
    }
    X <- drop(mu) + eS$vectors %*% diag(sqrt(pmax(ev, 0)), p) %*%
        t(X)
    nm <- names(mu)
    if (is.null(nm) && !is.null(dn <- dimnames(Sigma)))
        nm <- dn[[1L]]
    dimnames(X) <- list(nm, NULL)
    if (n == 1)
        drop(X)
    else t(X)
}


## Simulation of samples
rHGLM <- function(nb,rho,nu=0.5,sigma2_u=0.1,size=10,base=0,alpha=-1,beta=0.1,
    spread=2/rho,rbdesign="",family="binomial",phi=0.1) {
  if (rbdesign=="Loaloa") {
    data(Loaloa)
    x <- Loaloa$longitude
    y <- Loaloa$latitude
    nb <- nrow(Loaloa)
    size <- Loaloa$ntot
    env <- Loaloa$elev1
    loc <- seq(nrow(Loaloa))
  } else if (rbdesign=="blackcap") {
    data(blackcap)
    x <- blackcap$longitude
    y <- blackcap$latitude
    family <- "gaussian"
    env <- blackcap$means ## because per-individual values are not available
        for migratory behaviour
    nb <- nrow(blackcap)
    loc <- seq(nb)
  } else {
    x <- spread*(rnorm(nb))
    y <- spread*(rnorm(nb))
    loc  <- (1:nb)/nb
    env <- loc
  }
  names(x)<-loc ## to end up with names on the distm rows and cols
  distm <- as.matrix(dist(cbind(x,y)))
  m <- Matern.corr(rho*distm,nu=nu)
  u <- sqrt(sigma2_u) * mvrnorm(1,rep(0,nb),m) ## gaussian ie GLMM ## this
      would be better called v=Lu
  eta <- alpha+beta*(1:nb)/nb+u ## linear predictor for freq/count without the
       size factor
  obs <- switch(family,
                binomial= rbinom(nb,size=size,prob=1/(1+exp(-eta)))  ,
                 poisson= rpois(nb,exp(log(base)+eta)),
                gaussian= rnorm(nb,mean=eta,sd=sqrt(phi)),
                stop("(!) From HGLM: unknown 'family' argument. I exit.")
               )
  if (family=="binomial") return(data.frame(succes=obs,echec=size-obs,x,y,loc=
      loc,env=env,U=u))
  if (family=="poisson") return(data.frame(count=obs,x,y,loc=loc,env=env,U=u))
  if (family=="gaussian") return(data.frame(resp=obs,x,y,loc=loc,env=env,U=u))
}

## function to analyse a sample contained in global variable currentSample
do.simul <- function(nb=100,rho=1,nu=0.5,size=100,sigma2_u=1,spread=2/rho,beta
    =0.1,base=0,family="binomial",phi=0.1,
                         test="beta",
                         maxit=100,outer.eps=1e-05,verbose=T,trace=F,
                         rbdesign="",LRTfn=corrMM.LRT,
                         REMLformula=NULL, ## to control *non-default* REML
                             correction
                         always.refit=T,HLmethod="HL(0,1)",...
                   ) {
  zut <- match.call()
  zut <- as.list(zut[-1])
  d <- currentSample
  if (family=="binomial" && size==1 && nb==100) { ## catch poor binary samples
    sumsucces <- sum(d$succes)
    if (sumsucces>90 || sumsucces<10) return(list(notEnoughInfo=T))
  }
  nb <- nrow(d) ## makes a difference for e.g. nbdesign="Loaloa" where the
```

```
    default nb is ignored
if (test=="beta") {
  if (family=="binomial") {
    br <- do.call(LRTfn, list(null.predictor=Predictor(cbind(succes,echec)~1
        +(1|loc)), ## null hypo being tested
         predictor=Predictor(cbind(succes,echec)~env +(1|loc)),REMLformula=
             REMLformula,
         data=d,family=do.call(family,list()),# BinomialDen=d$succes+d$echec,
         always.prof.fixed=always.refit ,trace=trace,
         HLmethod=HLmethod,verbose=F,...))
  } else if (family=="poisson") {
    br <- do.call(LRTfn, list(null.predictor=Predictor(count~1 +(1|loc)), ##
        null hypo being tested
         predictor=Predictor(count~env +(1|loc)),REMLformula=REMLformula,
         data=d,family=do.call(family,list()),
         always.prof.fixed=always.refit ,trace=trace,
         HLmethod=HLmethod,verbose=F,...))
  } else if (family=="gaussian") {
    br <- do.call(LRTfn, list(null.predictor=Predictor(resp~1 +(1|loc)), ##
        null hypo being tested
         predictor=Predictor(resp~env +(1|loc)),REMLformula=REMLformula,
         data=d,family=do.call(family,list()),
         always.prof.fixed=always.refit ,trace=trace,
         HLmethod=HLmethod,verbose=F,...))
  }
} else {
  stop("From do.simul.binom(): 'test' option must be implemented ")
}
rescorrpars <- br$fullfit$corrPars ## new version
if (is.null(rescorrpars)) rescorrpars <- br$fullfit$corrpars ## back
    compatibility
## some inelegant code to keep information in any place some earlier
    postprocessing code assumes it is
if (is.null(rescorrpars$loglambda)) rescorrpars$loglambda <- log(br$fullfit$
    lambda[1])
if (family=="gaussian") {
  if (is.null(rescorrpars$logphi)) rescorrpars$logphi <- log(br$fullfit$phi
      [1])
}
resu <- list(beta.est=br$fullfit$fixef[2], full.p_v=br$fullfit$APHLs$p_v,
    corrpars=rescorrpars,df=br$df,trace.info=br$trace.info)
resu$null$loglambda <- log(br$nullfit$lambda[1]) ## added 12/01/2013
resu$null$corrpars <- br$nullfit$corrPars ## added 13/02/2013
## info
thisFormals <- formals(do.simul) ## list with default values !
namesWOdots <- names(thisFormals)[names(thisFormals)!="..."]
argvec <- sapply(namesWOdots,function(v) {get(v)}) ## vector with local
    values !! (more direct way ?)
subsublist <- as.list(argvec[intersect(names(argvec),c("rho","nu","nb","
    sigma2_u","phi"))])
subsublist$s2 <- subsublist$sigma2_u; subsublist$sigma2_u <-NULL
chaine <- sapply(seq(length(subsublist)),function(v) {paste(names(subsublist
    [v]),subsublist[v],sep="=")})
chaine <- paste(chaine,collapse=", ")
```

```
  resu$args <- chaine
  resu$data <- d
  resu$family <- family
  ## end info
  if ( ! is.null(br$LRTprof) ) { ## ie for test of fixed effects
    resu$LRTnaif<-br$LRTori ## not so naive in fact
    resu$LRTalwprof <- br$LRTprof ## else no $alwprofcolumn in the output
  } else resu$LRT <- br$LRT
  if ( ! is.null(br$meanbootLRT)) { ##
    resu$meanbootLRT <- br$meanbootLRT
    resu$bootreps <- br$bootreps
  }
  return(resu)
}

# tmp <- "params=list(replicat=1,nb=40,nu=0.5,rho=10,size=40,sigma2_u=0.1,beta
    =0,fixed=list(nu=0.5),HLmethod='HL(1,1)',init.corrHLfit=list(lambda=0.05))
    "


## code to process a call through e.g. R --vanilla --repl=1:1000
tmp <- commandArgs()[[3]]
replicats <- tmp[substr(tmp,0,7)=="--repl="]
replicats <- eval(parse(text=substring(replicats,8)))
rangereps <- range(replicats)
repmin <- rangereps[1]
nreps <- rangereps[2]-rangereps[1]+1

## reading the parameter file
source("arglist.R")

## implementing some defaults
if ( ! is.null(arglist$rbdesign) && arglist$rbdesign=="blackcap" && is.null(
    arglist$family)) {
  arglist$family <- "gaussian"
} else if (is.null(arglist$family)) arglist$family <- "binomial"
if ( ! is.null(arglist$nrepl)) {
  arglist$boot.repl <- arglist$nrepl
  arglist$nrepl <- NULL
}

##arguments sufficient for generating the samples
sublist <- arglist[which(names(arglist) %in% names(formals(rHGLM)))]


## compute the samples once for all
sampleList <- list()
set.seed(123)
if (repmin-1 >0) silent<-replicate(repmin-1,  do.call(rHGLM, sublist))
for (ii in seq(nreps)) {
  sampleList[[repmin-1+ii]] <- do.call(rHGLM, sublist)
}

for (ii in seq(nreps)) {
  set.seed(123) ## control of bootstrap replicates (at least)
  currentSample <- sampleList[[repmin+ii -1]]
  ## checking for separation in binary data
  if (arglist$family=="binomial" && length(unique(currentSample$succes)) == 2)
      {
    XX <- cbind(1,currentSample$env)
    separation <- separator(XX, currentSample$succes, purpose = "test")$
        separation
  } else separation <- FALSE
  if(separation) {
    print(paste("separation for replicat ",ii))
    resu <- list(rep=ii,separation=TRUE)
  } else {
    if (interactive()) {
      resu <- do.call(do.simul,arglist) ## now uses currentSample
    } else resu <- try(do.call(do.simul,arglist)) ## now uses currentSample
  }
  ## saves result for each data set in a distinct file
  save(resu,file=paste("resu",repmin-1+ii,".Rdata",sep=""))
}
```