```
#
    ##############################################################################################

## code for generating and analyzing samples by lme (for LMMs) or glmmPQL (for
     GLMMs) ##
#
    ##############################################################################################


library(nlme)
library(spaMM) ## as this script may use data sets from the spaMM package

# 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)
}



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) {
## attention a l'appel dans do.simul.binom si on rajoute des parametres ici...
## et aussi   sublist dans l'appel de replicate(repmin-1,  do.call(rbinomHGLM,
    sublist))
  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)
    ## *do not* provide simulation parameter values (phi...) through this code
  } 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)
  eta <- alpha+beta*(1:nb)/nb+u
  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 rHGLM: 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))
}


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,
                     corSt="corMatern",
                     rbdesign="",
                     REMLformula=NULL,
                     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
  d <-cbind(d,dummy=1)
  corFn <- eval(parse(text=corSt))
  if (test=="beta") {
  ## In all cases we help the fit by providing the true range value as the
      starting value
    if (family=="binomial") {
      full.model <- glmmPQL(fixed = cbind(succes,echec) ~ env, data = d,
          random = ~ 1 | dummy,family=binomial,
                  correlation = corFn(form = ~ x + y | dummy,nugget=F))
      lrtpval <- NA
    } else if (family=="poisson") {
      full.model <- glmmPQL(fixed = count ~ env, data = d, random = ~ 1 |
          dummy,family=poisson ,
                  correlation = corFn(form = ~ x + y | dummy,nugget=F))  ## ,
                      control=lmeControl(opt="nlminb"))
      lrtpval <- NA
    } else if (family=="gaussian") {
          null.model <- lme(fixed = resp ~ 1, data = d, random = ~ 1 | dummy,
                  correlation = corFn(form = ~ x + y | dummy,nugget=F), method
                      = "ML")
          full.model <- update(null.model, fixed = resp ~ env)
          lrtpval <- 1-pchisq(2*(full.model$logLik-null.model$logLik),df=1)
    }
    beta.est <- full.model$coefficients$fixed[["env"]]
    needtocallsummary <- summary(full.model) ## call to summary provides the t
        tests
    ttestpval <- needtocallsummary$tTable[2,"p-value"] ## this is the test
        that glmmPQL provides
    #loglambda <- 2*log(full.model$sigma)
    loglambda <- log(as.numeric(VarCorr(full.model)["Residual","Variance"]))
    rangenu <- exp(coef(full.model$modelStruct$corStruct))
    logLik <- full.model$logLik ##
  }
    logphi <- log(as.numeric(VarCorr(full.model)["(Intercept)","Variance"]))
    return(c(beta.est=beta.est,loglambda =loglambda,logphi =logphi,rangenu=
        rangenu,ttestpval=ttestpval,lrtpval=lrtpval,logLik=logLik))
}

## code to process a call through e.g. R --vanilla --repl=1:1000
tmp <- commandArgs()[[3]] ## R puis --vanilla puis --params

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 (arglist$rbdesign=="blackcap" && is.null(arglist$family)) arglist$family <-
    "gaussian"
```

```
##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)
}

if (arglist$corSt=="corMatern") {corparnames <- c("range","nu")} else
    corparnames <- c("range")
dispnames <- c("loglambda","logphi")
resunames <- c("beta.est",dispnames,corparnames,"ttestpval","lrtpval","logLik"
    )
resulen <- length(resunames)
resulme <-as.data.frame(matrix(nrow=nreps,ncol=resulen))
colnames(resulme) <- resunames
for (ii in seq(nreps)) {
  set.seed(123) ## control of bootstrap replicates (at least)
  currentSample <- sampleList[[repmin+ii-1]]
  if (interactive()) {
    resulme[ii,] <- do.call(do.simul,arglist) ## now uses currentSample
  } else {
    essai <- try(do.call(do.simul,arglist)) ## ow uses currentSample
    if (class(essai)=="try-error") {
      resulme[ii,] <- rep(NA,resulen)
    } else resulme[ii,] <- essai
  }
  if(!(ii %%50)) {cat(ii);cat("␣")}
}

save(resulme,file=paste("resulme",rangereps[1],".",rangereps[2],".Rdata",sep="
    "))
```