11 Linear mixed-effects model

# We will re-simulate some Asp viper data using R code fairly similar to that in the previous chapter. However, we will now *constrain the values for at least one set of effects* (either the intercepts and/or the slopes) to come from a normal distribution: this is what the random-effects assumption in a traditional mixed model means. There are at least three sets of assumptions that we could make about the random effects for the intercept or the slope of regression lines that are fitted to grouped (here, population-specific) data:

1. Only intercepts are random, but slopes are identical for all groups,
2. both intercepts and slopes are random, but they are independent, and
3. both intercepts and slopes are random and there is a correlation between them.

Model No. 1 is often called a random-intercepts model, and both models No. 2 and 3 are also called random-coefficients models. As we will see, model No. 3 is the default in R’s function lmer() when fitting a random-coefficients model.

We will now first generate a random-coefficients data set under model No. 2, where both intercepts and slopes are uncorrelated random effects. We will then fit both a random-intercepts (No. 1) and a random-coefficients model without correlation (No. 2) to these data (see 11.2–11.4).

This is a key chapter for your understanding of mixed models and we expect its contents to be helpful for the general understanding of mixed models to many ecologists. A close examination of how such data can be assembled (i.e., simulated) will be an invaluable help for your understanding of how analogous data sets are broken down (i.e., analyzed) when using any type of mixed model. Indeed, we believe that very few strategies can be more effective to understand this type of mixed model than the combination of simulating data sets and describing the models fitted in BUGS language.

#11.2 Data generation

# ------------------------------------

set.seed(11)

nPops <- 56 # Number of populations

nSample <- 10 # Number of vipers in each pop

n <- nPops \* nSample # Total number of data points

pop <- gl(n = nPops, k = nSample) # Indicator for population

# We directly normalize covariate length to avoid trouble with JAGS.

**# Body length (cm)**

orig.length <- runif(n, 45, 70)

mn <- mean(orig.length)

sd <- sd(orig.length)

cat("Mean and sd used to normalise original length:", mn, sd, "\n\n")

length <- (orig.length - mn) / sd

hist(length, col = "grey")

# We build a design matrix without intercept.

Xmat <- model.matrix(~pop\*length-1-length)

print(Xmat[1:21,], dig = 2) # Print top 21 rows

# Next, we choose parameter values, but this time, we need to constrain them, i.e., both the values for the intercepts and those for the slopes will now be drawn from two normal distributions for which we will specify four hyperparameters, i.e., two means (corresponding to mu\_alpha  and mu\_beta) and two standard deviations (corresponding to the intecept.sd and the slope.sd). As residual variation we will use a mean-zero normal distribution with a standard deviation of 30.

intercept.mean <- 230 # mu\_alpha

intercept.sd <- 20 # sigma\_alpha

slope.mean <- 60 # mu\_beta

slope.sd <- 30 # sigma\_beta

intercept.effects <-rnorm(n = nPops, mean = intercept.mean, sd = intercept.sd)

slope.effects <- rnorm(n = nPops, mean = slope.mean, sd = slope.sd)

all.effects <- c(intercept.effects, slope.effects) # Put them all together

# We assemble the measurements  as before.

sigma <- 30 # Residual standard deviation

lin.pred <- Xmat[,] %\*% all.effects # Value of lin.predictor

eps <- rnorm(n = n, mean = 0, sd = sigma) # residuals

mass <- lin.pred + eps # response = lin.pred + residual

hist(mass, col = "grey") # Inspect what we’ve created

**# Save true values for comparisons later**

truth <- c(intercept.mean=intercept.mean, slope.mean=slope.mean,

intercept.sd=intercept.sd, slope.sd=slope.sd,  
 residual.sd=sigma)

# Look at data

xyplot(mass ~ length | pop, xlab = 'Length', ylab = 'Mass', main = 'Realized mass-length relationships', pch = 16, cex = 1.2, col = rgb(0, 0, 0, 0.4))

### 11.3 Analysis under a random-intercepts model

### ---------------------------------------------------------------

# 11.3.1 ML and REML estimates using canned functions in R

# --------------------------------------------------------------------------------------------

# We first assume that the slope of the mass-length relationship is identical in all populations and that only the intercepts differ randomly from one population to another.

library('lme4')

out11.3.ML <- lmer(mass ~ length + (1 | pop), REML = FALSE) # with ML

out11.3.REML <- lmer(mass ~ length + (1 | pop), REML = TRUE) # with REML

summary(out11.3.ML)

summary(out11.3.REML)

**# Compare estimates with truth**

lmeML\_est <- c(fixef(out11.3.ML), as.data.frame(VarCorr(out11.3.ML))$sdcor)

lmeREML\_est <- c(fixef(out11.3.REML), as.data.frame(VarCorr(out11.3.REML))$sdcor)

**# Remove parameter 4 from truth (slope sd) since we didn't estimate it**

tmp <- cbind(truth=truth[-4], lmeML=lmeML\_est, lmeREML=lmeREML\_est)

print(tmp, 4)

11.3.2 Bayesian analysis with JAGS

# ---------------------------------------------------

**# Bundle and summarize data**

str(dataList <- list(mass = as.numeric(mass), pop = as.numeric(pop), length = length, nPops = nPops, n = n) )

**# Write JAGS model file**

cat(file="model11.3.2.txt", "

model {

**# Priors**

for (i in 1:nPops){

intercept[i] ~ dnorm(intercept.mean, intercept.prec) # Random intercepts

}

intercept.mean ~ dnorm(0, 0.001) # Mean hyperparameter for random intercepts

intercept.prec <- pow(intercept.sd, -2)

intercept.sd ~ dunif(0, 100) # SD hyperparameter for random intercepts

slope.mean ~ dnorm(0, 0.001) # Common slope

prec <- pow(residual.sd, -2) # Residual precision

residual.sd ~ dunif(0, 100) # Residual standard deviation

**# 'Likelihood'**

for (i in 1:n) {

mass[i] ~ dnorm(mu[i], prec) # The observed random variables

mu[i] <- intercept[pop[i]] + slope.mean\* length[i] # Expectation

}

}

")

**# Function to generate starting values**

inits <- function(){list(intercept.mean = rnorm(1, 0, 1),   
 slope.mean = rnorm(1, 0, 1), intercept.sd = rlnorm(1),   
 residual.sd = rlnorm(1)) }

**# Parameters to estimate**

params <- c("intercept.mean", "slope.mean", "intercept.sd",  
 "residual.sd", "intercept")

**# MCMC settings**

na <- 1000 ; ni <- 3000 ; nb <- 1000 ; nc <- 3 ; nt <- 1

**# Call JAGS (ART <1 min), check convergence and summarize posteriors**

out11.3.2 <- jags(dataList, inits, params, "model11.3.2.txt",   
 n.iter = ni, n.burnin = nb, n.chains = nc, n.thin = nt, n.adapt = na,  
 parallel = TRUE)

par(mfrow=c(2, 2)); jagsUI::traceplot(out11.3.2) # not shown

print(out11.3.2, 2)

**# Compare likelihood with Bayesian estimates and with truth**

jags\_est <- out11.3.2$summary[c(1:4),1]

tmp <- cbind(truth=truth[-4], lmeML=lmeML\_est, JAGS=jags\_est)

print(tmp, 4)

# Do you see anything odd ? Explain ....

# 11.3.4 Bayesian analysis with Stan

# -----------------------------------------------------

library(rstan)

**# Summarize data set again**

str(dataList)

**# Write Stan model**

cat(file="model11\_3\_4.stan", "

data {

int n; //Number of observations

int nPops; //Number of populations

vector[n] mass; //Response variable

vector[n] length; //Covariate

int pop[n]; //Population assignment of each obs

}

parameters {

real intercept\_mean;

real slope\_mean;

real<lower=0> intercept\_sd;

real<lower=0> residual\_sd;

vector[nPops] intercept;

}

model {

vector[n] mu; //Expected value

//Priors

//Note: no priors specified for SDs, so they are implicitly flat

//If you try to specify them as uniform(0, 100),   
 // you will have a bad time!

intercept\_mean ~ normal(0, 1000);

slope\_mean ~ normal(0, 1000);

for (i in 1:nPops){

intercept[i] ~ normal(intercept\_mean, intercept\_sd);

}

//Likelihood

for (i in 1:n){

mu[i] = intercept[pop[i]] + slope\_mean \* length[i];

mass[i] ~ normal(mu[i], residual\_sd);

}

}

")

**# HMC settings**

ni <- 1000 ; nb <- 500 ; nc <- 3 ; nt <- 1

**# Call STAN (ART 80 sec / 16 sec)**

system.time(

out11.3.4 <- stan(file = "model11\_3\_4.stan", data = dataList,

warmup = nb, iter = ni, chains = nc, thin = nt) )

rstan::traceplot(out11.3.4) # not shown

print(out11.3.4, dig = 3) # not shown

**# Compare estimates with truth**

stan\_est <- summary(out11.3.4)$summary[1:4,1]

tmp <- cbind(truth=truth[-4], lmeML=lmeML\_est, JAGS=jags\_est, NIMBLE=nimble\_est, Stan=stan\_est)

print(tmp, 4)

# 11.3.5 Do-it-yourself MLEs

# For the homegrown MLEs we have to integrate over all possible values for the random effect, which here is the set of intercepts.

**# Bundle and summarize data set (note contains nSample now)**

str(dataList <- list(mass=as.numeric(mass), length=length,  
 pop=as.numeric(pop), nPops=nPops, nSample=nSample, n=n) )

**# Definition of NLL for random-intercepts model with Gaussian errors**

NLL <- function(pars, data) {

intercept.mean <- pars[1]

slope.mean <- pars[2]

intercept.sd <- exp(pars[3])

residual.sd <- exp(pars[4])

nll <- 0

for (i in 1:data$nPops){

# Subset data mass and length to just pop i

ysub <- data$mass[pop == i]

lengthsub <- data$length[pop == i]

lik <- integrate(function(x){

tot <- 1 # Starting value for product over J

# Iterate over each sample j in pop i

for (j in 1:data$nSample){

mu <- (intercept.mean + x) + slope.mean \* lengthsub[j]

tot <- tot \* dnorm(ysub[j], mu, residual.sd)

}

tot <- tot \* dnorm(x, 0, intercept.sd)

tot

}, lower=-Inf, upper=Inf, subdivisions=20)$value

nll <- nll - log(lik)

}

return(nll)

}

**# Minimize that NLL to find MLEs and also get SEs**

inits <- c('intercept.mean' = mean(mass), 'slope.mean' = 0,

'intercept.sd' = 0, 'residual.sd' = 3)

out11.3.5 <- optim(inits, NLL, hessian=TRUE, method = "BFGS", data=dataList)

getMLE(out11.3.5, 4)

# Nice (but don't forget that the variances are estimated on the log scale ).

**# Compare estimates with truth and previous estimates**

diy\_est <- out11.3.5$par

diy\_est[3:4] <- exp(diy\_est[3:4]) # Get variances on ilog scale

tmp <- cbind(truth=truth[-4], lmeML=lmeML\_est, JAGS=jags\_est, NIMBLE=nimble\_est, Stan=stan\_est, DIY=diy\_est)

print(tmp, 4)

### 11.4 Analysis under a random-coefficients model without correlation between intercept and slope

### ---------------------------------------------------------------------------------------------

# Next, we assume that both slopes and intercepts of the mass-length relationship differ among populations in the fashion of two independent random variables. That is, we will declare both to be random, but assume the absence of a correlation between intercept and slope. Thus, we will analyze the data under the same model that we used to generate our data set.

# 11.4.1 REML and ML estimates using canned functions in R

#library('lme4')

**# the || notation means no correlation between random effects**

out11.4.ML <- lmer(mass ~ length + ( 1 + length || pop), REML = FALSE)

out11.4.REML <- lmer(mass ~ length + ( 1+ length || pop), REML = TRUE)

summary(out11.4.ML)

summary(out11.4.REML)

**# Compare estimates with truth**

lmeML\_est <- c(fixef(out11.4.ML), as.data.frame(VarCorr(out11.4.ML))$sdcor)

lmeREML\_est <- c(fixef(out11.4.REML), as.data.frame(VarCorr(out11.4.REML))$sdcor)

tmp <- cbind(truth=truth, lmeML=lmeML\_est, lmeREML=lmeREML\_est)

print(tmp, 4)

# 11.4.2 Bayesian analysis with JAGS

# -----------------------------------------------------------

**# Bundle and summarize data**

str(dataList <- list(mass = as.numeric(mass), pop = as.numeric(pop), length = length, nPops = nPops, n = n) )

**# Write JAGS model file**

cat(file="model11.4.2.txt", "

model {

**# Priors**

for (i in 1:nPops){

intercept[i] ~ dnorm(intercept.mean, intercept.prec) # Random intercepts

slope[i] ~ dnorm(slope.mean, slope.prec) # Random slopes

}

intercept.mean ~ dnorm(0, 0.001) # Mean hyperparameter for random intercepts

intercept.prec <- pow(intercept.sd, -2)

intercept.sd ~ dunif(0, 100) # SD hyperparameter for random intercepts

slope.mean ~ dnorm(0, 0.001) # Mean hyperparameter for random slopes

slope.prec <- pow(slope.sd, -2)

slope.sd ~ dunif(0, 100) # SD hyperparameter for slopes

prec <- pow(residual.sd, -2) # Residual precision

residual.sd ~ dunif(0, 100) # Residual standard deviation

# 'Likelihood'

for (i in 1:n) {

mass[i] ~ dnorm(mu[i], prec)

mu[i] <- intercept[pop[i]] + slope[pop[i]] \* length[i]

}

}

")

**# Function to generate starting values**

inits <- function(){ list(intercept.mean = rnorm(1, 0, 1),   
 intercept.sd = rlnorm(1), slope.mean = rnorm(1, 0, 1),   
 slope.sd = rlnorm(1), residual.sd = rlnorm(1)) }

**# Parameters to estimate**

params <- c("intercept.mean", "slope.mean", "intercept.sd", "slope.sd",

"residual.sd", "intercept", "slope")

**# MCMC settings**

na <- 1000 ; ni <- 3000 ; nb <- 1000 ; nc <- 3 ; nt <- 1

**# Call JAGS (ART <1 min), check convergence and summarize posteriors**

out11.4.2 <- jags(dataList, inits, params, "model11.4.2.txt",   
 n.iter = ni, n.burnin = nb, n.chains = nc, n.thin = nt, n.adapt = na,  
 parallel = TRUE)

par(mfrow=c(2, 2)); jagsUI::traceplot(out11.4.2) # not shown

print(out11.4.2, 2)

**# Compare likelihood with Bayesian estimates and with truth**

jags\_est <- out11.4.2$summary[c(1:5),1]

tmp <- cbind(truth=truth, lmeML=lmeML\_est, JAGS=jags\_est)

print(tmp, 4)

# If you want to see the random effects, do this:

ranef(out11.4.ML) # For the analysis in R

print(out11.4.2, 2) # In JAGS

# 11.4.4 Bayesian analysis with Stan

library(rstan)

**# Summarize data set again**

str(dataList)

**# Write Stan model**

cat(file="model11\_4\_4.stan", "

data {

int n; //Number of observations

int nPops; //Number of populations

vector[n] mass; //Response variable

vector[n] length; //Covariate

int pop[n]; //Population assignment of each obs

}

parameters {

real intercept\_mean;

real slope\_mean;

real<lower=0> intercept\_sd;

real<lower=0> slope\_sd;

real<lower=0> residual\_sd;

vector[nPops] intercept;

vector[nPops] slope;

}

model {

vector[n] mu; //Expected value

//Priors

//Note: no priors specified for SDs, so they are implicitly flat

//If you try to specify them as uniform(0,100), you will have a bad time!

intercept\_mean ~ normal(0, 1000);

slope\_mean ~ normal(0, 1000);

for (i in 1:nPops){

intercept[i] ~ normal(intercept\_mean, intercept\_sd);

slope[i] ~ normal(slope\_mean, slope\_sd);

}

//Likelihood

for (i in 1:n){

mu[i] = intercept[pop[i]] + slope[pop[i]] \* length[i];

mass[i] ~ normal(mu[i], residual\_sd);

}

}

")

**# HMC settings**

ni <- 1000 ; nb <- 500 ; nc <- 3 ; nt <- 1

**# Call STAN (ART 53 sec / 30 sec – timings very variable here)**

system.time(

out11.4.4 <- stan(file = "model11\_4\_4.stan", data = dataList,

warmup = nb, iter = ni, chains = nc, thin = nt) )

rstan::traceplot(out11.4.4) # not shown

print(out11.4.4, dig = 3) # not shown

**# Compare estimates with truth**

stan\_est <- summary(out11.4.4)$summary[1:5,1]

tmp <- cbind(truth=truth, lmeML=lmeML\_est, JAGS=jags\_est, NIMBLE=nimble\_est, Stan=stan\_est)

print(tmp, 4)

# 11.4.5 Do-it-yourself MLEs

# As before, to write the likelihood required for parameter estimation, we must integrate out the random effects from the likelihood that we use in the optimisation. This means a double integral here, and for this we use functionality in the R package pracma (though others would be possible, too). Below you will see that our fitting method is very sensitive to the choice of initial values. We offer considerable 'help' to the algorithm by our choice of starting values since initializing the optimisation algorithm at very general places (such as 0 or 1) will invariably fail.

library(pracma)

**# Definition of NLL for random-slopes model with Gaussian errors**

NLL <- function(pars, response = mass, cov = length) {

intercept.mean <- pars[1]

slope.mean <- pars[2]

intercept.sd <- exp(pars[3])

slope.sd <- exp(pars[4])

residual.sd <- exp(pars[5])

nll <- 0

for (i in 1:nPops){

# Subset data mass and length to just pop i

masssub <- response[pop == i]

lengthsub <- cov[pop == i]

# Double integral function from pracma library

lik <- integral2(function(x, y){

tot <- 1 # Starting value for product over J

# Iterate over each sample j (1-12) in pop i

for (j in 1:nSample){

mu <- intercept.mean + x + (slope.mean + y) \* lengthsub[j]

tot = tot \* dnorm(masssub[j], mu, residual.sd)

}

tot <- tot \* dnorm(x, 0, intercept.sd) \* dnorm(y, 0, slope.sd)

tot

}, xmin=-200, xmax=200, ymin=-100, ymax=100)$Q

nll <- nll - log(lik)

}

return(nll)

}

**# Minimize that NLL to find MLEs and also get SEs (ART 123 sec)**

inits <- c('intercept.mean' = mean(mass), 'slope.mean' = 60,  
 'log.intercept.sd' = log(20), 'log.slope.sd' = log(30),   
 'log.residual.sd' = log(30)) # Pretty 'informative' inits

inits <- c('intercept.mean' = mean(mass), 'slope.mean' = 60,  
 'log.intercept.sd' = 3, 'log.slope.sd' = 3,   
 'log.residual.sd' = 3) # Try less 'informative inits'

system.time(

out11.4.5 <- optim(inits, NLL, hessian=TRUE, method = 'BFGS',

control=list(trace=1, REPORT=5)) )

getMLE(out11.4.5, 4)

# When starting the optimisation kind of right on target, it works, but when using the second set of starting values (which are already pretty close to the target !), optim will declare successful convergence, but this is in fact not true. You can try that out. Here is the solution with the first sets of inits above.

We do our usual comparison.

**# Compare estimates with truth and previous estimates**

diy\_est <- out11.4.5$par

diy\_est[3:5] <- exp(diy\_est[3:5]) # Get variances on ilog scale

tmp <- cbind(truth=truth, lmeML=lmeML\_est, JAGS=jags\_est, NIMBLE=nimble\_est, Stan=stan\_est, DIY=diy\_est)

print(tmp, 4)