# 14 Poisson regression with continuous and categorical explanatory variables

# To stress the similarity with the normal linear case, we only slightly alter the inferential setting sketched in chapter 10. We assume that instead of measuring body mass in Asp vipers in three populations in the Pyrenees, Massif Central and the Jura mountains, leading to a normal model, we had instead assessed ectoparasite load in a dragonfly, the Sombre Goldenring (Fig. 14-1), leading to a Poisson model. We are particularly interested in whether there are more or less little red mites on dragonflies of different size (expressed as wing length) and whether this relationship differs among the three mountain ranges. (Actually, dragonflies don’t vary that much in body size, but let’s assume there is sufficient variation to make such a study worthwhile.)

# 14.2 Data generation

# We assemble a data set.

set.seed(14)

nPops <- 3

nSample <- 100

n <- nPops \* nSample

x <- rep(1:nPops, rep(nSample, nPops)) # Population indicator

pop <- factor(x, labels = c("Pyrenees", "Massif Central", "Jura"))

orig.length <- runif(n, 4.5, 7.0) # Wing length (cm)

length <- orig.length - mean(orig.length) # Centre by subtracting mean

# We build the design matrix of an interactive combination of length and population:

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

head(Xmat, 10) # Look at first 10 rows of design matrix

# Select the parameter values

**# Save truth for comparisons**

truth <- beta.vec <- c(-2, 1, 2, 4, -2, -5)

# Here’s the recipe for assembling the mite counts in three steps:

# (1) we add up all components of the linear model to get the linear predictor, which is the expected mite count on the transformed log scale (i.e., at the log link),  
# (2) we exponentiate to get the actual value of the expected mite count on the natural scale and

# (3) we add Poisson noise.

# We again obtain the value of the linear predictor by matrix multiplication of the design matrix (Xmat) and the parameter vector (beta.vec):

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

lambda <- exp(lin.pred) # Poisson mean: expected count

C <- rpois(n = n, lambda = lambda) # Add Poisson noise

**# Inspect what we’ve created**

par(mfrow = c(1, 2), mar = c(5,5,4,3), cex.axis = 1.5, cex.lab = 1.5)

hist(C, col = "grey", breaks = 30, xlab = "Parasite load", main = "", las = 1)

plot(length, C, pch = rep(c("P", "M", "J"), each=nSample), las = 1, col = rep(c("Red", "Green", "Blue"), each=nSample), ylab = "Parasite load", xlab = "Wing length", cex = 1.2, frame = FALSE)

# We have created a data set where parasite load increases with wing length in the South (Pyrenees, Massif Central) but decreases in the North (Jura mountains)

# 14.3 Likelihood analysis with canned functions in R

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

summary(out14.3 <- glm(C ~ pop \* length, family = poisson)) # not shown

**# Compare with truth**

glm\_est <- coef(out14.3)

tmp <- cbind(truth=truth, glm=glm\_est)

print(tmp, 4)

# 14.4 Bayesian analysis with JAGS

**# Bundle and summarize data**

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

**# Write JAGS model file**

cat(file="model14.4.txt", "

model {

**# Priors**

for (i in 1:nPops){

alpha[i] ~ dnorm(0, 0.01) # Intercepts

beta[i] ~ dnorm(0, 0.01) # Slopes

}

**# Likelihood**

for (i in 1:n) {

C[i] ~ dpois(lambda[i]) # The random variable

lambda[i] <- exp(alpha[pop[i]] + beta[pop[i]]\* length[i])

} # Note double-indexing: alpha[pop[i]]

**# Derived quantities**

**# Recover effects relative to baseline level (no. 1)**

a.effe2 <- alpha[2] - alpha[1] # Intercept Massif Central vs. Pyr.

a.effe3 <- alpha[3] - alpha[1] # Intercept Jura vs. Pyr.

b.effe2 <- beta[2] - beta[1] # Slope Massif Central vs. Pyr.

b.effe3 <- beta[3] - beta[1] # Slope Jura vs. Pyr.

**# Custom test**

test1 <- beta[3] - beta[2] # Slope Jura vs. Massif Central

}

")

**# Function to generate starting values**

inits <- function(){list(alpha=rlnorm(nPops, 3, 1), beta=rlnorm(nPops, 2, 1))}

**# Parameters to estimate**

params <- c("alpha", "beta", "a.effe2", "a.effe3", "b.effe2", "b.effe3", "test1")

**# MCMC settings**

na <- 2000 ; ni <- 25000 ; nb <- 5000 ; nc <- 3 ; nt <- 5

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

out14.4 <- jags(dataList, inits, params, "model14.4.txt", n.iter = ni, n.burnin = nb, n.chains = nc, n.thin = nt, n.adapt = na, parallel = TRUE)

par(mfrow=c(2, 2)); traceplot(out14.4) # not shown

print(out14.4, 3) # not shown

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

jags\_est <- unlist(out14.4$mean)[c(1,7,8,4,9,10)]

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

print(tmp, 4)

# Remember that alpha[1] and beta[1] in JAGS correspond to the intercept and the length main effect in the analysis in R and a.effe2, a.effe3. b.effe2, b.effe3 to the remaining terms of the analysis in R.

# 14.4.2 Forming predictions

Finally, let’s summarize our main findings from the analysis in a graph. We illustrate the posterior distribution of the relationship between mite load and wing length for each of the three study areas. To do that, we predict mite load for 100 dragonflies in each of the three mountain ranges and plot these estimates along with their uncertainty. We compute the predicted relationship between mite count and wing-length for a sample of 100 of all MCMC draws of the involved parameters and plot that (Fig. 14-3).

**# Create a vector with 100 wing lengths**

orig.wlength <- sort(orig.length)

wlength <- orig.wlength - mean(orig.length)

**# Create matrices to contain prediction for each winglength and MCMC** **iteration**

nsamp <- out14.4$mcmc.info$n.samples # Get size of posterior sample

sel.sample <- sample(1:nsamp, size = 100)

mite.load.Pyr <- mite.load.MC <- mite.load.Ju <- array(dim = c(300, 100))

**# Fill in these vectors: this is clumsy, but it works**

for(i in 1:300) {

for(j in 1:100) {

mite.load.Pyr[i,j] <- exp(out14.4$sims.list$alpha[sel.sample[j],1] + out14.4$sims.list$beta[sel.sample[j],1] \* wlength[i])

mite.load.MC[i,j] <- exp(out14.4$sims.list$alpha[sel.sample[j],2] + out14.4$sims.list$beta[sel.sample[j],2] \* wlength[i])

mite.load.Ju[i,j] <- exp(out14.4$sims.list$alpha[sel.sample[j],3] + out14.4$sims.list$beta[sel.sample[j],3] \* wlength[i])

}

}

**# Two variants of a plot**

par(mfrow = c(1, 2))

matplot(orig.wlength, mite.load.Pyr, col = "grey", type = "l", las = 1, ylab = "Expected mite load", xlab = "Wing length (cm)", frame = FALSE)

for(j in 1:100){

points(orig.wlength, mite.load.MC[,j], col = "grey", type = "l")

points(orig.wlength, mite.load.Ju[,j], col = "grey", type = "l")

}

points(orig.wlength, exp(out14.4$mean$alpha[1] + out14.4$mean$beta[1] \* wlength), col = "red", type = "l", lwd = 3)

points(orig.wlength, exp(out14.4$mean$alpha[2] + out14.4$mean$beta[2] \* wlength), col = "green", type = "l", lwd = 3)

points(orig.wlength, exp(out14.4$mean$alpha[3] + out14.4$mean$beta[3] \* wlength), col = "blue", type = "l", lwd = 3)

# I find Fig. 14-3 (left) a rather nice plot, but if an editor asks for more conventional 95% credible intervals instead, you can give also use this code, which results in Fig. 14-3 (right)

**# Compute 95% Bayesian prediction intervals**

LCB.Pyr <- apply(mite.load.Pyr, 1, quantile, prob=0.025)

UCB.Pyr <- apply(mite.load.Pyr, 1, quantile, prob=0.975)

LCB.MC <- apply(mite.load.MC, 1, quantile, prob=0.025)

UCB.MC <- apply(mite.load.MC, 1, quantile, prob=0.975)

LCB.Ju <- apply(mite.load.Ju, 1, quantile, prob=0.025)

UCB.Ju <- apply(mite.load.Ju, 1, quantile, prob=0.975)

mean.rel <- cbind(exp(out14.4$mean$alpha[1] + out14.4$mean$beta[1] \* wlength), exp(out14.4$mean$alpha[2] + out14.4$mean$beta[2] \* wlength), exp(out14.4$mean$alpha[3] + out14.4$mean$beta[3] \* wlength))

covar <- cbind(orig.wlength, orig.wlength, orig.wlength)

matplot(orig.wlength, mean.rel, col = c("red", "green", "blue"), type = "l", lty = 1, lwd = 2, las = 1, ylab = "Expected mite load", xlab = "Wing length (cm)", frame = FALSE)

polygon(c(orig.wlength, rev(orig.wlength)), c(LCB.Pyr, rev(UCB.Pyr)), col = rgb(0,0,0,0.3), border = NA)

polygon(c(orig.wlength, rev(orig.wlength)), c(LCB.MC, rev(UCB.MC)), col = rgb(0,0,0,0.3), border = NA)

polygon(c(orig.wlength, rev(orig.wlength)), c(LCB.Ju, rev(UCB.Ju)), col = rgb(0,0,0,0.3), border = NA)

matplot(orig.wlength, mean.rel, col = c("red", "green", "blue"), type = "l", lty = 1, lwd = 3, add = TRUE)

# 14.6 Bayesian analysis with Stan

library(rstan)

**# Bundle and summarize data**

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

**# Write Stan model**

cat(file="model14\_6.stan", "

data{

int n;

int nPops;

int C[n];

vector[n] length;

int pop[n];

}

parameters{

vector[nPops] alpha;

vector[nPops] beta;

}

model{

vector[n] lambda;

for (i in 1:nPops){

alpha[i] ~ normal(0, 1000);

beta[i] ~ normal(0, 1000);

}

for (i in 1:n){

lambda[i] = exp(alpha[pop[i]] + beta[pop[i]] \* length[i]);

C[i] ~ poisson(lambda[i]);

}

}

generated quantities{

real a\_effe2 = alpha[2] - alpha[1];

real a\_effe3 = alpha[3] - alpha[1];

real b\_effe2 = beta[2] - beta[1];

real b\_effe3 = beta[3] - beta[1];

real test1 = beta[3] - beta[2];

}

")

**# HMC settings**

ni <- 2000 ; nb <- 1000 ; nc <- 3 ; nt <- 1

**# Call STAN (ART 40/10 sec), assess convergence and print results table**

system.time(

out14.6 <- stan(file = "model14\_6.stan", data = dataList,

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

rstan::traceplot(out14.6) # not shown

print(out14.6, dig = 3) # not shown

**# Compare estimates with truth**

stan\_est <- summary(out14.6)$summary[c(1,7,8,4,9,10),1]

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

Stan=stan\_est)

print(tmp, 4)

# 14.7 Do-it-yourself MLEs

And doing it on foot...

**# Define NLL for general Poisson regression**

NLL <- function(beta, y, Xmat) {

mu <- exp(Xmat %\*% beta)

L <- dpois(y, mu) # Likelihood contribution for 1 observation

LL <- log(L) # Log-likelihood contribution for 1 observation

NLL <- -sum(LL) # NLL for all observations in data set

return(NLL)

}

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

inits <- rep(0, 6) # This does not work for all

names(inits) <- rownames(tmp)

out14.7 <- optim(inits, NLL, y = C, Xmat = Xmat, hessian=TRUE, method = "BFGS")

getMLE(out14.7, 4)

**# Compare estimates with truth**

diy\_est <- out14.7$par

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

Stan=stan\_est, DIY=diy\_est)

print(tmp, 4)

# Marvelous !