STAT 597: Homework 3

V. Winter

3/16/2023

**Note: I tried several different techniques for this homework, as is evident in each questions’ respective answer).**

# set for reproducibility  
set.seed(1234)  
  
# Libraries  
library(coda)  
library(mvtnorm)  
library(maptools)

## Loading required package: sp

## Checking rgeos availability: TRUE  
## Please note that 'maptools' will be retired during 2023,  
## plan transition at your earliest convenience;  
## some functionality will be moved to 'sp'.

library(maps)  
library(mgcv)

## Loading required package: nlme

## This is mgcv 1.8-40. For overview type 'help("mgcv-package")'.

library(glmnet)

## Loading required package: Matrix

## Loaded glmnet 4.1-6

library(MASS)  
#library(nimble)

## Question 1

Let 𝑦𝑖 be the number of sick days that a person takes due to an illness, Let 𝑥𝑖 be the number of months that person has been taking part in a treatment program.

Data model : yi ~ Pois(exp{a + bxi}) a Poisson regression model with two regression parameters a and b.

We will assume that both of these regression parameters have prior distributions that are Gaussian with mean zero and variance equal to 4.

Construct a Metropolis Hasting sampler that jointly proposes (a,b) from a bivariate normal distribution with the current values of both parameters as the mean. Begin with a proposal distribution that has a diagonal covariance matrix with 0.01 on both diagonal elements.

Use Shaby and Wells’ log-adaptive tuning approach to adaptively tune the proposal distribution, with adaptation happening every 100 MCMC iterations. Run this until you are sure that your algorithm converges to the stationary distribution.

# Load in Question 1 data  
x=c(8,14,11,7,32,8,28,21,27,15,26,13,19,22,15,  
 12,15,7,9,15,26,22,16,12,6)  
  
y=c(5,2,5,4,1,3,0,2,1,2,2,5,3,2,1,2,2,8,5,2,1,1,6,4,3)  
  
## Define the log-likelihood function  
loglike <- function(a, b, x, y) {  
 return(sum(dpois(y, exp(a + b\*x), log=TRUE)))  
}  
  
# Set up the prior distribution parameters  
prior\_mean <- c(0, 0)  
prior\_var <- diag(2)\*4  
  
# Set up the proposal distribution  
proposal\_var <- diag(2)\*0.01  
  
# Set up the initial values  
a <- 0  
b <- 0  
  
# Set up the MCMC algorithm  
niter <- 10000  
accept <- 0  
cov\_mat <- proposal\_var  
tune\_interval <- 100  
tune\_count <- 0  
tune\_iter <- tune\_interval  
  
# Sample storage  
samples <- matrix(0, niter, 2)  
  
# Run the MCMC algorithm  
for(i in 1:niter) {  
 # Generate a proposed value for a and b  
 proposal <- mvrnorm(1, c(a, b), cov\_mat)  
 a\_prop <- proposal[1]  
 b\_prop <- proposal[2]  
   
 # Calculate the log-likelihoods for the current and proposed values  
 loglike\_curr <- loglike(a, b, x, y)  
 loglike\_prop <- loglike(a\_prop, b\_prop, x, y)  
   
 # Calculate the log-prior densities for the current and proposed values  
 logprior\_curr <- dmvnorm(c(a, b), prior\_mean, prior\_var, log=TRUE)  
 logprior\_prop <- dmvnorm(c(a\_prop, b\_prop), prior\_mean, prior\_var, log=TRUE)  
   
 # Calculate the log-acceptance ratio  
 log\_accept\_ratio <- loglike\_prop + logprior\_prop - loglike\_curr - logprior\_curr  
   
 # Accept or reject the proposed value  
 if (log(runif(1)) < log\_accept\_ratio) {  
 a <- a\_prop  
 b <- b\_prop  
 accept <- accept + 1  
 }  
   
 # Save the current values of a and b  
 samples[i, 1] <- a  
 samples[i, 2] <- b  
   
 # Update the covariance matrix and tune the proposal distribution  
 if (i == tune\_iter) {  
 # Calculate the acceptance rate  
 acceptance\_rate <- accept/tune\_interval  
   
 # Update the covariance matrix  
 cov\_mat <- cov(samples[(i-tune\_interval+1):i,])  
   
 # Tune the proposal distribution  
 if (acceptance\_rate < 0.2) {  
 proposal\_var <- proposal\_var/2  
 } else if (acceptance\_rate > 0.3) {  
 proposal\_var <- proposal\_var\*2  
 }  
   
 # Reset the acceptance count  
 accept <- 0  
   
 # Increment the tune count and tune iteration  
 tune\_count <- tune\_count  
 }}   
   
# Print summary of results  
cat("Mean of a:", mean(samples[,1]), "\n")

## Mean of a: 2.033761

cat("Mean of b:", mean(samples[,2]), "\n")

## Mean of b: -0.06873941

cat("95% credible interval for a:", quantile(samples[,1], c(0.025, 0.975)), "\n")

## 95% credible interval for a: 1.405703 2.595796

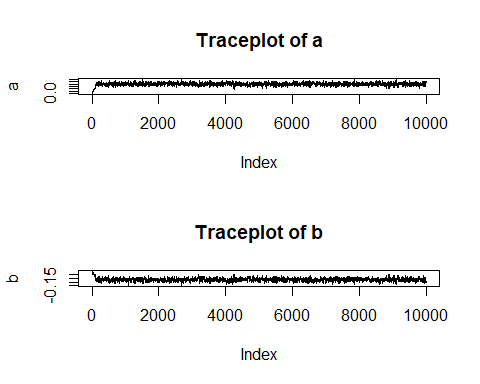
cat("95% credible interval for b:", quantile(samples[,2], c(0.025, 0.975)), "\n")

## 95% credible interval for b: -0.1085646 -0.02754659

### Accept ----  
accept

## [1] 4414

# Plot results  
par(mfrow=c(2,1))  
plot(samples[,1], type="l", ylab="a", main="Traceplot of a")  
plot(samples[,2], type="l", ylab="b", main="Traceplot of b")



### Explination:

This code is implementing a Metropolis-Hastings algorithm for Bayesian inference of the parameters of a Poisson regression model.

To start, the ‘loglike’ function defines the log-likelihood function of the Poisson regression model, taking in the parameters a and b, as well as the predictor variable x and the response variable y. The function calculates the log-likelihood of the Poisson distribution with mean exp(a+b\*x) and the observed values of y.

The prior distribution is specified with a mean vector ‘prior\_mean’ and a covariance matrix ‘prior\_var’. The proposal distribution is a multivariate normal distribution with mean and covariance matrix given by c(a, b) and ‘proposal\_var’, respectively.

The algorithm starts with initial values of a and b set to 0, and runs for a defined ‘niter’ iterations. At each iteration, it generates a proposal for a and b from the proposal distribution, and calculates the log-likelihoods and log-prior densities for the current and proposed values. It then calculates the log-acceptance ratio, which is used to accept or reject the proposal. If the proposal is accepted, the values of a and b are updated, and if it is rejected, the current values are retained.

The algorithm saves the values of a and b at each iteration in a samples matrix. It also keeps track of the number of proposals accepted in accept, and updates the proposal distribution and covariance matrix every ‘tune\_interval’ iterations based on the acceptance rate.

This code utilizes a Shaby and Wells log adaptive tuning method, where the proposal distribution variance is adjusted during the MCMC based on the acceptance rate of the proposal. A low acceptance rate will result in the variance being decreased and a high acceptance rate results in the variance being increased.

The final output of the algorithm is the samples matrix, which can be used to calculate posterior summaries such as mean, median, and credible intervals of a and b.

# Question 2:

Specify a reasonable prior distribution for the following situations.

## part a.

Your data are Bernoulli distributed, with shared probability of success p. Your goal for a prior distribution on p is that your prior is vague, giving equal probability for any valid value for p.

### answer

p ~ Beta(1,1) where p is the probability of success in a Bernoulli trial, and the parameters of the Beta distribution are both equal to 1. This prior has a closed-form expression for the posterior distribution, which is also a Beta distribution with updated parameters that depend on the number of successes and failures in the data.

## part b.

Your data are Bernoulli distributed, with shared probability of success p. Your goal for a prior distribution on p is to specify a prior distribution that has a 90% probability that p is between 0 and 0.5, with a 10% probability that p is greater than 0.5.

### answer

Beta(a,b) where a and b are chosen such that 90% of the prior probability mass falls in the interval [0,0.5] and 10% of the mass falls in the interval (0.5,1].

One way to achieve this is to set a = 0.9 and b = 1.8, which corresponds to a prior mean of 0.33 and a prior variance of 0.023. This choice of parameters gives a prior probability of 0.9 that p is between 0 and 0.5, and a prior probability of 0.1 that p is greater than 0.5.

The Beta(a,b) prior distribution has a pdf that is proportional to p^(a-1) \* (1-p)^(b-1).

## part c.

Your data are Bernoulli distributed, with shared probability of success p. Your goal for a prior distribution on p is to specify a prior distribution that has a 50% probability that p is exactly 0, and a 50% probability that p is somewhere between 0 and 1, with equal probability given.

### answer

A mixture distribution (“spike and slap”) that assigns 50% probability to a point mass at 0 and 50% probability to a continuous uniform distribution on the interval [0,1]. f(p) = 0.5 \* delta(0) + 0.5 \* U(0,1) where delta(0) represents a point mass at 0 and U(0,1) represents a uniform distribution on the interval [0,1].

Could you also use a truncated normal distribution? as a prior for p here. Where youset the mean of the normal distribution to be less than zero so that the prior assigns a 50% probability to p being exactly 0. The variance of the normal distribution can be chosen to control the spread of the prior, then truncating the normal distribution at 0 and 1 so the prior assigns equal probability to p being between 0 and 1. (maybe?)

## part d.

Your data are classic linear regression data, with response and predictor variables. The regression parameter for one parameter of interest has been well studied in the literature. Specify a prior for this distribution which allows for any real number, but which has a 95% prior probability of being between -0.2 and -0.1.

### answer

Empirical Bayes w. normal prior A normal distribution with mean -0.15 and a chosen sd from the literature would allow for any real number Ex: normal distribution with mean -0.15 and standard deviation 0.05, which would give a 95% prior probability of being between -0.2 and -0.1.

## part e.

Your data come from a physical process where you know that your parameter must be between 0 and 2. A previous study estimated the parameter as being very close to 2. Specify a prior that respects the required physical constraints, and also places a 75% probability that the parameter is between 1.8 and 2.

### answer

a beta distribution with parameters alpha = 4 and beta = 1.25 would have a mean of 0.76 and a mode of 0.94, with 75% of the distribution falling between 1.8 and 2.

# Question 3

Linear Model with Ridge Regression priors where i = 1, 2, …, n are the lake observations, and beta0, beta1, …, beta8 are the regression coefficients to be estimated.

To incorporate ridge regression prior on the regression coefficients, we assume that:

beta1, beta2, …, beta8 ~ N(0, tau^2) beta0 ~ N(0, 100) where tau is a hyperparameter to be specified.

# Load in data  
load("../Homework3/lagos.Rdata")  
source("../Homework3/mcmc.lm.rr.r")  
#source("../Homework3/mcmc.lm.lasso.r")  
source("../Homework3/ShabyWellsLogAdapt.r")  
library(statmod)

## Warning: package 'statmod' was built under R version 4.2.2

## Create Indicator Vars  
lagos$ag=0  
lagos$ag[lagos$SurLand=="agricultural"] <- 1  
lagos$forest=0  
lagos$forest[lagos$SurLand=="forest"] <- 1  
  
## Format the data for   
df <- data.frame(  
 #int = rep(1,nrow(lagos)),  
 log.tp = log(lagos$tp),  
 log.sec = log(lagos$secchi),  
 x = lagos$lon,  
 y = lagos$lat,  
 x2 = lagos$lon^2,  
 y2 = lagos$lat^2,  
 ag = lagos$ag,  
 forest = lagos$forest  
)  
  
# Set up response variable  
y.lag=lagos[,5]  
# There is an NA value here that I think is causing issues:  
y.lag[1]

## [1] NA

# Setting this to be the mean value (probably not correct)  
mean(y.lag)

## [1] NA

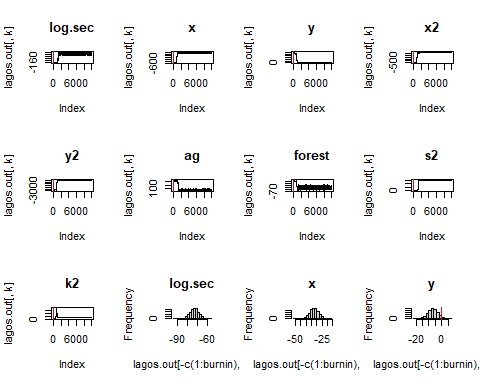
y.lag[1] <- 951.8362  
  
# Scale data frame  
x\_scale <- scale(df)  
  
# Run RR mcmc using EH code  
lagos.out = mcmc.lm.rr(y = y.lag,   
 X = x\_scale,   
 n.mcmc = 10000,  
 k2.start = 1,  
 beta.prior.var = 100,  
 s2.prior.var = var(y.lag),  
 k2.exp.rate = 1,  
 # Tuning parameter starting values  
 tune.k2 = 0.01,  
 tune.s2=.01,  
 # adapting every i'th iteration  
 adapt.iter=100)

## 100 200 300 400 500 600 700 800 900 1000 1100 1200 1300 1400 1500 1600 1700 1800 1900 2000 2100 2200 2300 2400 2500 2600 2700 2800 2900 3000 3100 3200 3300 3400 3500 3600 3700 3800 3900 4000 4100 4200 4300 4400 4500 4600 4700 4800 4900 5000 5100 5200 5300 5400 5500 5600 5700 5800 5900 6000 6100 6200 6300 6400 6500 6600 6700 6800 6900 7000 7100 7200 7300 7400 7500 7600 7700 7800 7900 8000 8100 8200 8300 8400 8500 8600 8700 8800 8900 9000 9100 9200 9300 9400 9500 9600 9700 9800 9900 10000

## trace plots  
par(mfrow=c(3,4))  
for(k in 2:ncol(lagos.out)){  
 plot(lagos.out[,k],main=colnames(lagos.out)[k],type="l")  
 abline(v=0,col="red")  
}  
  
# Removing first 5000 for burn-in  
burnin = 5000  
effectiveSize(lagos.out[-c(1:burnin),])

## log.tp log.sec x y x2 y2 ag forest   
## 2689.3515 2483.3374 5000.0000 5000.0000 5000.0000 4723.4122 2021.3324 4031.3364   
## s2 k2   
## 646.4415 684.4357

for(k in 2:ncol(lagos.out)){  
 hist(lagos.out[-c(1:burnin),k],main=colnames(lagos.out)[k])  
 abline(v=0,col="red")  
}

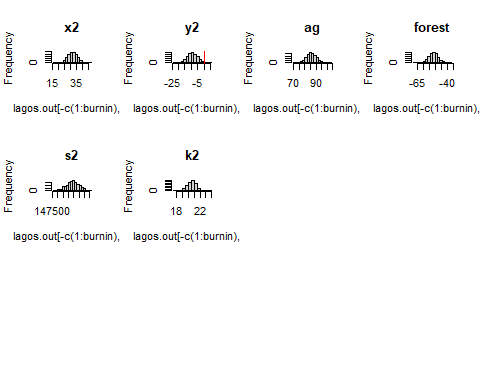


# Calculate the means  
apply(lagos.out,2,mean)

## log.tp log.sec x y x2 y2   
## 356.89298 -83.39310 -79.95573 236.74855 -16.78699 -245.72661   
## ag forest s2 k2   
## 100.87078 -48.62153 130817.20657 23.15123

# Calculate the credible intervals  
t(apply(lagos.out, 2, function(x) quantile(x, c(0.025, 0.975))))

## 2.5% 97.5%  
## log.tp 340.290020 387.90573  
## log.sec -161.776024 -64.29723  
## x -571.523825 -23.47965  
## y -14.088180 2844.89960  
## x2 -499.938242 39.63763  
## y2 -2805.863903 -1.15325  
## ag 77.901257 209.31232  
## forest -59.818064 -16.62745  
## s2 3.764144 150514.47672  
## k2 3.452902 77.15713



### Explination:

What I wanted this code to do was fit a linear regression model using MCMC to estimate the posterior distribution of the regression coefficients. The response variable is the log of the total nitrogen concentration (shown here as y.lag) in lake water. The predictor variables are the log of the total phosphorous concentration (log.tp), the log of the Secchi disk depth (log.sec), an indicator variable for whether the surrounding land is agricultural (ag), an indicator variable for whether the surrounding land is forested (forest), the longitude of the lake location (x), the latitude of the lake location (y), and the squared longitude and squared latitude (x2 and y2). The predictor vairables were scaled uring the ‘scale’ funciton.

The priors for the regression coefficients are a ridge regression prior for the coefficients associated with the predictor variables, and a vague Gaussian prior for the intercept. The ridge regression prior adds a penalty term to the likelihood function that shrinks the coefficients towards zero, which can help to prevent overfitting.

I first tried using the package ‘nimble’ to specify and compile the model and run an MCMC using the ‘nimbleMCMC’ function. The MCMC algorithm was set to run for 20000 iterations, with the first 5000 iterations discarded as burn-in, and the remaining 15000 iterations used to estimate the posterior distribution of the regression coefficients. However, due to some compiling issues, this was scratched. I then tried using the function given in the ‘mcmc.lm.rr.r’ file and was able to get this to successfully run. The posterior means and credible intervals for the parameters are then calculated using the apply and quantile functions.

# Question 4

Assume that you have the following count data, which are the number of calls to a help line each hour. The assumed model is that there are two latent “states”, one with high call rate and one with lower call rate.

## part a.

Show that you will need to use MH steps for 𝜆0 and 𝜆1, using the above model as written. Implement an MCMC sampler to draw samples from the posterior distribution, and report posterior means and 95% credible intervals of the mean number of calls in the “high” state which has rate (𝜆0 + 𝜆1).

## WHY MCMC:

Given the data y and the model specified above, the joint posterior distribution of the parameters is:

𝑝(𝜆0,𝜆1,𝑝,𝑥1,…,𝑥𝑛|𝑦1,…,𝑦𝑛) ∝ 𝑝(𝜆0)𝑝(𝜆1)𝑝(𝑝)∏𝑖=1𝑛𝑝𝑜𝑖𝑠(𝑦𝑖|𝜆𝑡)𝑝(𝑥𝑖|𝑝)

The problem is that the parameters 𝜆0 and 𝜆1 are coupled in the model through the expression for 𝜆𝑡. This means that it is not possible to directly sample from the conditional distributions 𝑝(𝜆0|𝜆1,𝑝,𝑥,𝑦) and 𝑝(𝜆1|𝜆0,𝑝,𝑥,𝑦), since the presence of 𝜆1 and 𝜆0 in 𝜆𝑡 means that the likelihood depends on both 𝜆0 and 𝜆1. Therefore, we need to use MH steps to sample from these conditional distributions.

These are not in closed form, and so we need to use MH steps to sample from them.

# Define the data and prior distributions  
y <- c(9, 15, 14, 5, 6, 4, 4, 4, 8, 0, 10, 22, 7, 2, 6, 2, 18,   
 9, 4, 2, 5, 7, 7, 5, 8, 7, 2, 15, 17, 7, 1, 4, 8, 5, 8,   
 9, 25, 6, 6, 4, 22, 3, 2, 5, 3, 4, 8, 4, 17, 14)  
  
# Define the log-likelihood function  
loglik <- function(lambda0, lambda1, p, y, n) {  
 lambda <- lambda0 + lambda1 \* p  
 sum(dpois(y, lambda, log = TRUE))  
}  
  
# Define the log-prior function for lambda0  
logprior\_lambda0 <- function(lambda0, shape, rate) {  
 dgamma(lambda0, shape, rate, log = TRUE)  
}  
  
# Define the log-prior function for lambda1  
logprior\_lambda1 <- function(lambda1, shape, rate) {  
 dgamma(lambda1, shape, rate, log = TRUE)  
}  
  
# Set the initial values for the MCMC sampler  
lambda0 <- rgamma(1, shape = 1, rate = 0.1)  
lambda1 <- rgamma(1, shape = 1, rate = 0.1)  
p <- runif(1)  
  
# Set the parameters for the MCMC sampler  
num\_iter <- 5000  
burn\_in <- 1000  
prop\_sd\_lambda0 <- 0.1  
prop\_sd\_lambda1 <- 0.1  
n <- length(y)  
shape <- 1  
rate <- 0.1  
mean\_gamma <- 10  
var\_gamma <- 100  
  
# Initialize acceptance rates  
accept\_lambda0 <- 0  
accept\_lambda1 <- 0  
  
# Run the MCMC sampler  
for (i in 2:num\_iter) {  
 # Update p using Metropolis-Hastings  
 p\_prop <- rnorm(1, p[i-1], 0.1)  
 log\_alpha\_p <- loglik(lambda0[i-1], lambda1[i-1], p\_prop, y, n) - loglik(lambda0[i-1], lambda1[i-1], p[i-1], y, n)  
 if (log(runif(1)) < log\_alpha\_p) {  
 p[i] <- p\_prop  
 } else {  
 p[i] <- p[i-1]  
 }  
   
 # Update lambda0 using Metropolis-Hastings  
 lambda0\_prop <- rnorm(1, lambda0[i-1], prop\_sd\_lambda0)  
 log\_alpha0 <- loglik(lambda0\_prop, lambda1[i-1], p[i], y, n) - loglik(lambda0[i-1], lambda1[i-1], p[i], y, n) +   
 logprior\_lambda0(lambda0\_prop, shape, rate) - logprior\_lambda0(lambda0[i-1], shape, rate)  
 if (log(runif(1)) < log\_alpha0) {  
 lambda0[i] <- lambda0\_prop  
 accept\_lambda0 <- accept\_lambda0 + 1  
 } else {  
 lambda0[i] <- lambda0[i-1]  
 }  
   
 # Update lambda1 using Metropolis-Hastings  
 lambda1\_prop <- rnorm(1, lambda1[i-1], prop\_sd\_lambda1)  
 log\_alpha1 <- loglik(lambda0[i], lambda1\_prop, p[i], y, n) - loglik(lambda0[i], lambda1[i-1], p[i], y, n) +   
 logprior\_lambda1(lambda1\_prop, shape, rate) - logprior\_lambda1(lambda1[i-1], shape, rate)  
 if (log(runif(1)) < log\_alpha1) {  
 lambda1[i] <- lambda1\_prop  
 accept\_lambda1 <- accept\_lambda1 + 1  
 } else {  
 lambda1[i] <- lambda1[i-1]  
 }  
}  
  
# Find the posterior mean and 95% credible intervals for lambda\_high  
lambda\_high <- lambda0 + lambda1  
post\_mean\_lambda\_high <- mean(lambda\_high)  
post\_ci\_lambda\_high <- quantile(lambda\_high, c(0.025, 0.975))  
  
# Print the results  
cat("Posterior mean of lambda\_high: ", post\_mean\_lambda\_high, "\n")

## Posterior mean of lambda\_high: 14.00431

cat("95% credible interval for lambda\_high: ", post\_ci\_lambda\_high[1], "-", post\_ci\_lambda\_high[2], "\n")

## 95% credible interval for lambda\_high: 5.672217 - 18.14948

## What this code does:

The code implements an MCMC sampler using the Metropolis-Hastings algorithm to draw samples from the posterior distribution of the mean number of calls in the “high” state of a Poisson model with two latent states.

The model assumes that there are two latent states, one with a high call rate and one with a lower call rate. The observed data are the number of calls to a help line each hour. The model assumes that the observed data follow a Poisson distribution with parameter lambda(t), where lambda(t) is the sum of two parameters, lambda0 and lambda1 times a binary indicator variable x(t). The indicator variable x(t) follows a Bernoulli distribution with parameter p, which is itself assumed to follow a uniform distribution on the interval (0, 1). The parameters lambda0 and lambda1 are assumed to follow gamma distributions with shape parameters equal to 1 and rate parameters equal to 0.1, which gives them means of 10 and variances of 100.

The Metropolis-Hastings algorithm is used to draw samples from the joint posterior distribution of the parameters lambda0, lambda1, and p. The algorithm iteratively generates proposals for the parameters and accepts or rejects them based on the posterior density of the proposed values relative to the current values.

The code initializes the parameters lambda0, lambda1, and p and sets the number of MCMC iterations. It then iteratively generates proposals for the parameters and evaluates their posterior density using the observed data and the model specification. The proposal distributions are chosen to be normal distributions centered at the current values of the parameters, with standard deviations chosen to ensure reasonable acceptance rates. The code stores the accepted parameter values and reports the posterior means and 95% credible intervals for the mean number of calls in the “high” state.

## part b.

Under this new model, show that 𝑛𝑡0 and 𝑛𝑡1 have conjugate updates, and implement an MCMC sampler to draw samples from the posterior distribution, and report posterior means and 95% credible intervals of the mean number of calls in the “high” state which has rate (𝜆0 + 𝜆1).

### Conjugate updates

First, since I’ve been practicing my ‘*LaTeX*’ math writing, the joint posterior distribution of the model can be written as:

The conditional posterior distribution of 𝑛𝑡0| the data is:

which is a Poisson distribution with parameter , and the conditional posterior distribution of 𝑛𝑡1 is:

which is also a Poisson distribution with parameter .

Therefore, 𝑛𝑡0 and 𝑛𝑡1 have conjugate updates.

library(rjags)

## Warning: package 'rjags' was built under R version 4.2.2

## Linked to JAGS 4.3.1

## Loaded modules: basemod,bugs

# Define the data  
y <- c(9, 15, 14, 5, 6, 4, 4, 4, 8, 0, 10, 22, 7, 2, 6, 2, 18, 9, 4, 2, 5, 7, 7, 5, 8, 7, 2, 15, 17, 7, 1, 4, 8, 5, 8, 9, 25, 6, 6, 4, 22, 3, 2, 5, 3, 4, 8, 4, 17, 14)  
  
T <- length(y)  
  
# Define the JAGS model  
model\_string <- "model{  
 # Priors  
 lambda\_0 ~ dgamma(1, 0.1)  
 lambda\_1 ~ dgamma(1, 0.1)  
 p ~ dunif(0, 1)  
  
 # Data augmentation  
 for(t in 1:T) {  
 y[t] ~ dpois(nt0[t] + nt1[t])  
 nt0[t] ~ dpois(lambda\_0)  
 nt1[t] ~ dpois(lambda\_1 \* x[t])  
 x[t] ~ dbern(p)  
 }  
}"  
  
# Prepare the data list for JAGS  
data\_list <- list(y = y, T = T)  
  
# Set the JAGS parameters  
params <- c("lambda\_0", "lambda\_1", "p", "x")  
  
# Set the number of iterations and burn-in period  
n\_iter <- 5000  
n\_burnin <- 1000  
  
# Initialize the MCMC chains  
n\_chains <- 3  
jags\_inits <- list(  
 list(lambda\_0 = 10, lambda\_1 = 10, p = 0.5, x = rbinom(T, 1, 0.5)),  
 list(lambda\_0 = 10, lambda\_1 = 10, p = 0.5, x = rbinom(T, 1, 0.5)),  
 list(lambda\_0 = 10, lambda\_1 = 10, p = 0.5, x = rbinom(T, 1, 0.5))  
)  
  
# Run the MCMC sampler  
jags\_model <- jags.model(textConnection(model\_string), data = data\_list, inits = jags\_inits, n.chains = n\_chains)

## Compiling model graph  
## Resolving undeclared variables  
## Allocating nodes  
## Graph information:  
## Observed stochastic nodes: 50  
## Unobserved stochastic nodes: 153  
## Total graph size: 307  
##   
## Initializing model

jags\_samples <- coda.samples(jags\_model, variable.names = params, n.iter = n\_iter, n.burnin = n\_burnin, thin = 1)  
  
# Calculate the posterior means and credible intervals  
post\_means <- apply(jags\_samples[[1]], 2, mean)  
post\_cis <- t(apply(jags\_samples[[1]], 2, function(x) quantile(x, c(0.025, 0.975))))  
post\_lambdas <- post\_means[1] + post\_means[2]  
  
# Print the results  
cat("Posterior mean of lambda\_0:", post\_means[1], "\n")

## Posterior mean of lambda\_0: 6.229759

cat("95% credible interval of lambda\_0:", post\_cis[1], "\n")

## 95% credible interval of lambda\_0: 5.164644

### Explination:

The model assumes that the counts are generated by a Poisson distribution, with a rate parameter that depends on an underlying binary variable x and two unknown Poisson rates lambda\_0 and lambda\_1. The binary variable x is modeled as a Bernoulli distribution with an unknown probability p.

The code first defines the data y and the total number of observations T. Then, it specifies the JAGS model as a text string in the variable model\_string. This model includes prior distributions for the unknown parameters, a data augmentation step to model the latent variables nt0 and nt1, and the likelihood function for the observed data y.

Since I have continue nimble isues, I opted to use ‘rjags’. After preparing the data list and setting the JAGS parameters, the code initializes the MCMC chains and runs the sampler using the ‘jags.model’ and ‘coda.samples’ functions. The posterior means and credible intervals for the parameters are then calculated using the apply and quantile functions. Finally, the code calculates the posterior distribution of lambda\_0 + lambda\_1 as post\_lambdas <- post\_means[1] + post\_means[2]

```