Lesson 11: Hierarchical modeling

Lesson 11.2

Data

Let's fit our hierarhical model for counts of chocolate chips. The data can be found in `cookies.dat`.

```
dat = read.table(file="cookies.dat", header=TRUE)
head(dat)
```

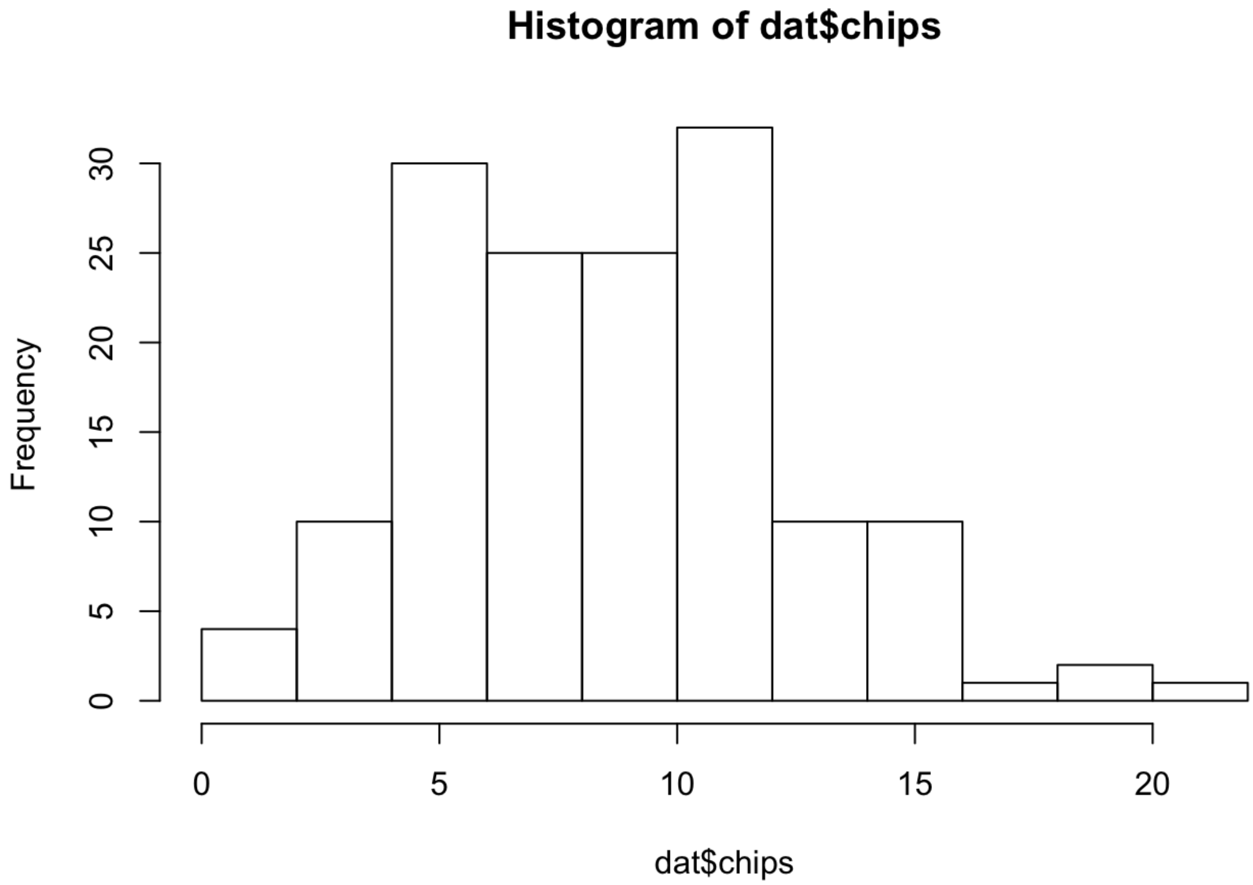
```
##   chips location
## 1    12        1
## 2    12        1
## 3     6        1
## 4    13        1
## 5    12        1
## 6    12        1
```

```
table(dat$location)
```

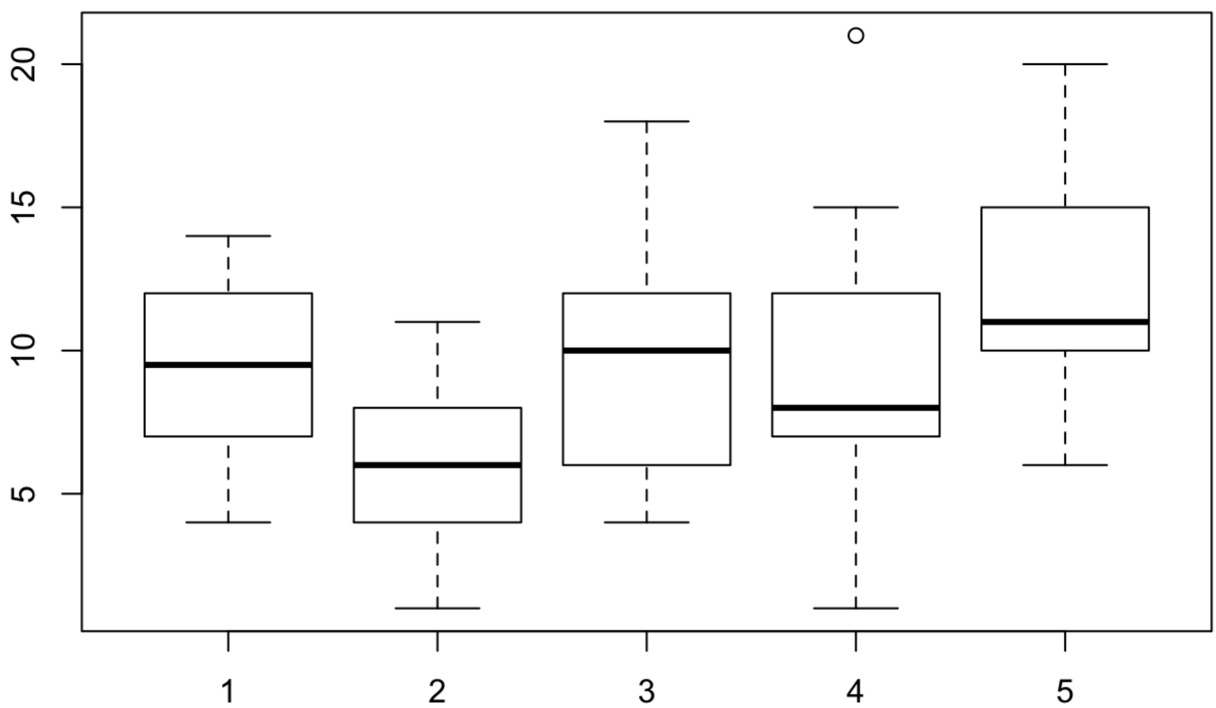
```
##
##  1  2  3  4  5
## 30 30 30 30 30
```

We can also visualize the distribution of chips by location.

```
hist(dat$chips)
```




```
boxplot(chips ~ location, data=dat)
```



Prior predictive checks

Before implementing the model, we need to select prior distributions for α and β, the hyperparameters governing the gamma distribution for the λ parameters. First, think about what the λ's represent. For location j, λ_j is the expected number of chocolate chips per cookie. Hence, α and β control the distribution of these means between locations. The mean of this gamma distribution will represent the overall mean of number of chips for all cookies. The variance of this gamma distribution controls the variability between locations. If this is high, the mean number of chips will vary widely from location to location. If it is small, the mean number of chips will be nearly the same from location to location.

To see the effects of different priors on the distribution of λ's, we can simulate. Suppose we try independent exponential priors for α and β.

```
set.seed(112)
n_sim = 500
alpha_pri = rexp(n_sim, rate=1.0/2.0)
beta_pri = rexp(n_sim, rate=5.0)
mu_pri = alpha_pri/beta_pri
sig_pri = sqrt(alpha_pri/beta_pri^2)

summary(mu_pri)
```

```
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
##    0.021    2.983    9.852   61.130   29.980 4859.000
```

```
summary(sig_pri)
```

```
##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
##    0.1834    3.3660    8.5490   41.8100   22.2200 2866.0000
```

After simulating from the priors for α and β, we can use those samples to simulate further down the hierarchy:

```
lam_pri = rgamma(n=n_sim, shape=alpha_pri, rate=beta_pri)
summary(lam_pri)
```

```
##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
##     0.000     1.171     7.667    83.060    28.620 11010.000
```

Or for a prior predictive reconstruction of the original data set:

```
(lam_pri = rgamma(n=5, shape=alpha_pri[1:5], rate=beta_pri[1:5]))
```

```
## [1] 66.444084  9.946688  6.028319 15.922568 47.978587
```

```
(y_pri = rpois(n=150, lambda=rep(lam_pri, each=30)))
```

```
##   [1] 63 58 64 63 70 62 61 48 71 73 70 77 66 60 72 77 69 62 66 71 49 80 66
##  [24] 75 74 55 62 90 65 57 12  9  7 10 12 10 11  7 14 13  9  6  6 13  7 10
##  [47] 12  9  9 10  7  8  6  9  7 10 13 13  8 12  6 10  3  6  7  4  6  7  5
##  [70]  5  4  3  6  2  8  4  8  4  5  7  1  4  5  3  8  8  3  1  7  3 16 14
##  [93] 13 17 17 12 13 13 16 16 15 14 11 10 13 17 16 19 16 17 15 16  7 17 21
## [116] 16 12 15 14 13 52 44 51 46 39 40 40 44 46 59 45 49 58 42 31 52 43 47
## [139] 53 41 48 57 35 60 51 58 36 34 41 59
```

Because these priors have high variance and are somewhat noninformative, they produce unrealistic predictive distributions. Still, enough data would overwhelm the prior, resulting in useful posterior distributions. Alternatively, we could tweak and simulate from these prior distributions until they adequately represent our prior beliefs. Yet another approach would be to re-parameterize the gamma prior, which we'll demonstrate as we fit the model.

Lesson 11.3

`JAGS` Model

```r
library("rjags")
```

```
## Loading required package: coda
```

```
## Linked to JAGS 4.2.0
```

```
## Loaded modules: basemod,bugs
```

```
mod_string = " model {
for (i in 1:length(chips)) {
  chips[i] ~ dpois(lam[location[i]])
}

for (j in 1:max(location)) {
  lam[j] ~ dgamma(alpha, beta)
}

alpha = mu^2 / sig^2
beta = mu / sig^2

mu ~ dgamma(2.0, 1.0/5.0)
sig ~ dexp(1.0)

} "

set.seed(113)

data_jags = as.list(dat)

params = c("lam", "mu", "sig")

mod = jags.model(textConnection(mod_string), data=data_jags, n.chains=3)
update(mod, 1e3)

mod_sim = coda.samples(model=mod,
                       variable.names=params,
                       n.iter=5e3)
mod_csim = as.mcmc(do.call(rbind, mod_sim))

## convergence diagnostics
plot(mod_sim)

gelman.diag(mod_sim)
autocorr.diag(mod_sim)
autocorr.plot(mod_sim)
effectiveSize(mod_sim)

## compute DIC
dic = dic.samples(mod, n.iter=1e3)
```

Model checking

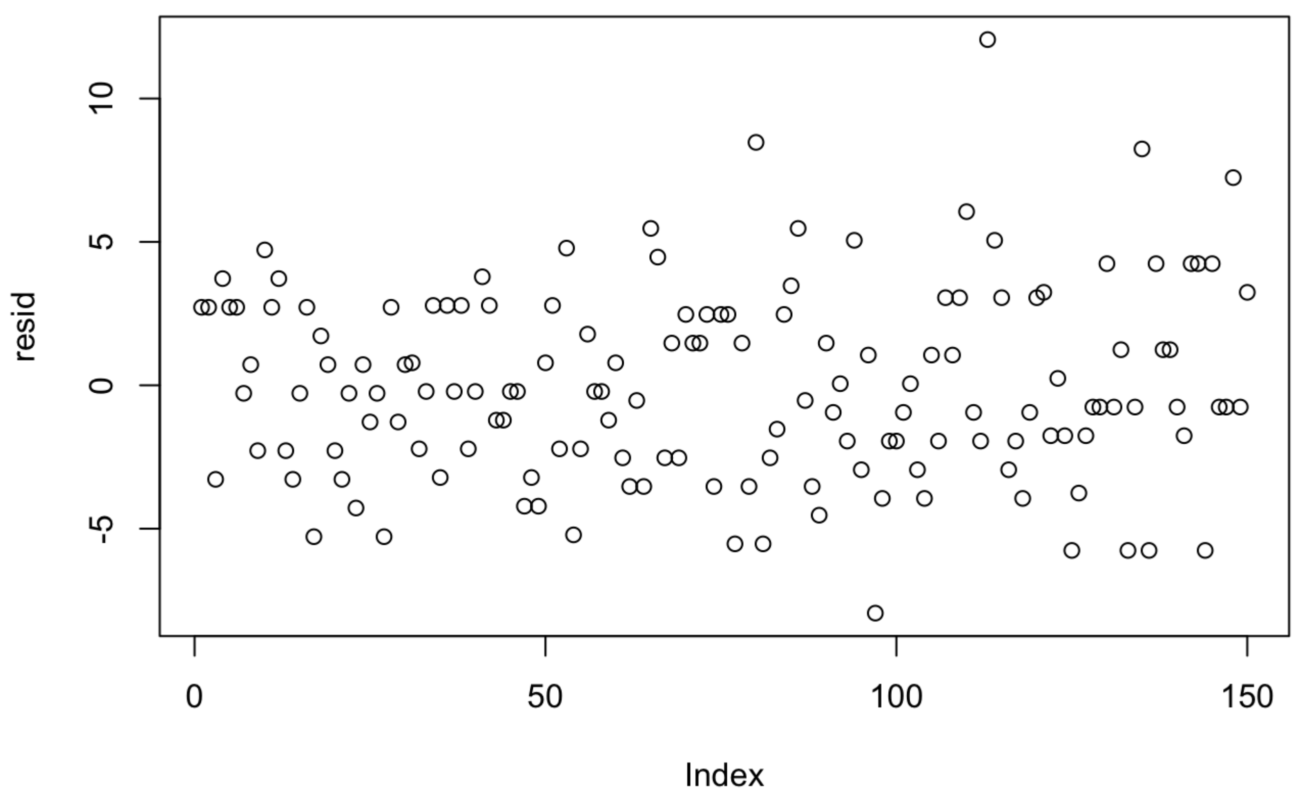
After assessing convergence, we can check the fit via residuals. With a hierarhcical model, there are now two levels of residuals: the observation level and the location mean level. To simplify, we'll look at the residuals associated with the posterior means of the parameters.

First, we have observation residuals, based on the estimates of location means.

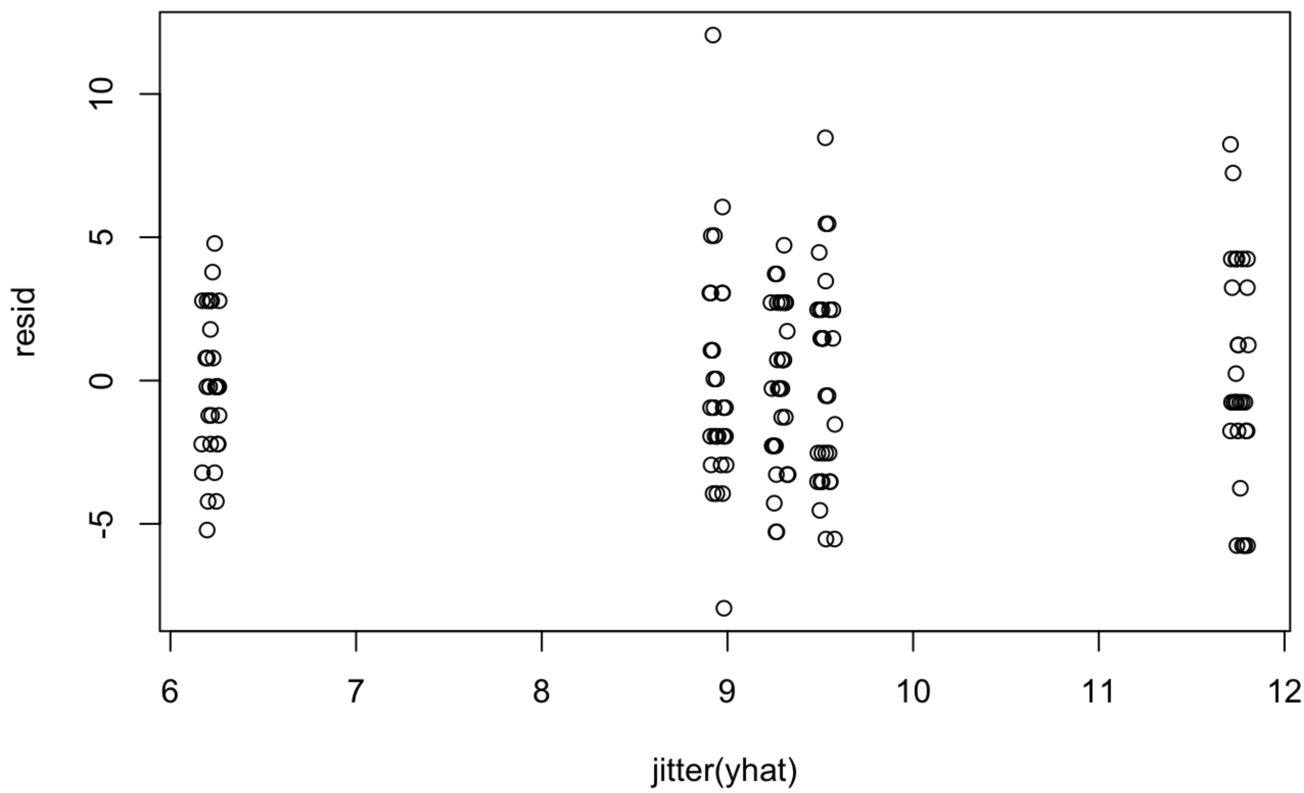
```
## observation level residuals
(pm_params = colMeans(mod_csim))
```

```
##     lam[1]     lam[2]     lam[3]     lam[4]     lam[5]         mu        sig
##   9.279690   6.216018   9.529345   8.944948  11.758629   9.114266   2.095705
```

```
yhat = rep(pm_params[1:5], each=30)
resid = dat$chips - yhat
plot(resid)
```




```
plot(jitter(yhat), resid)
```




```
var(resid[yhat<7])
```

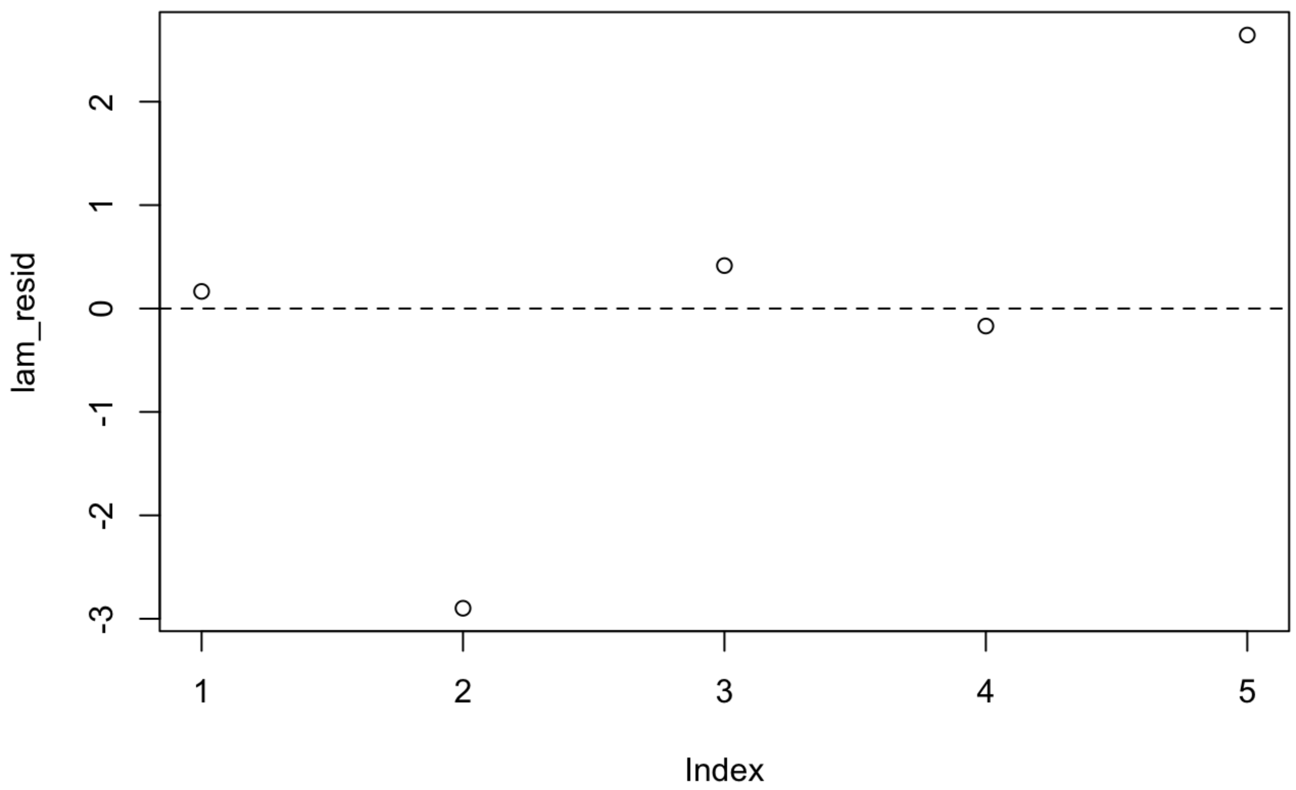
```
## [1] 6.447126
```

```
var(resid[yhat>11])
```

```
## [1] 13.72414
```

Also, we can look at how the location means differ from the overall mean \(\mu\).

```
## location level residuals
lam_resid = pm_params[1:5] - pm_params["mu"]
plot(lam_resid)
abline(h=0, lty=2)
```



We don’t see any obvious violations of our model assumptions.

Results

```
summary(mod_sim)
```

```
##
## Iterations = 2001:7000
## Thinning interval = 1
## Number of chains = 3
## Sample size per chain = 5000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##           Mean     SD Naive SE Time-series SE
## lam[1]  9.280 0.5344 0.004363       0.004363
## lam[2]  6.216 0.4641 0.003789       0.004294
## lam[3]  9.529 0.5439 0.004441       0.004503
## lam[4]  8.945 0.5266 0.004300       0.004300
## lam[5] 11.759 0.6246 0.005100       0.005660
## mu      9.114 0.9933 0.008110       0.012151
## sig     2.096 0.7180 0.005862       0.012142
##
## 2. Quantiles for each variable:
##
##           2.5%    25%    50%    75%  97.5%
## lam[1]  8.274  8.911  9.266  9.631 10.364
## lam[2]  5.327  5.897  6.215  6.523  7.142
## lam[3]  8.501  9.159  9.520  9.887 10.640
## lam[4]  7.954  8.580  8.935  9.291 10.005
## lam[5] 10.576 11.324 11.743 12.175 13.026
## mu      7.210  8.498  9.079  9.690 11.256
## sig     1.104  1.587  1.955  2.457  3.910
```

Lesson 11.4

Posterior predictive simulation

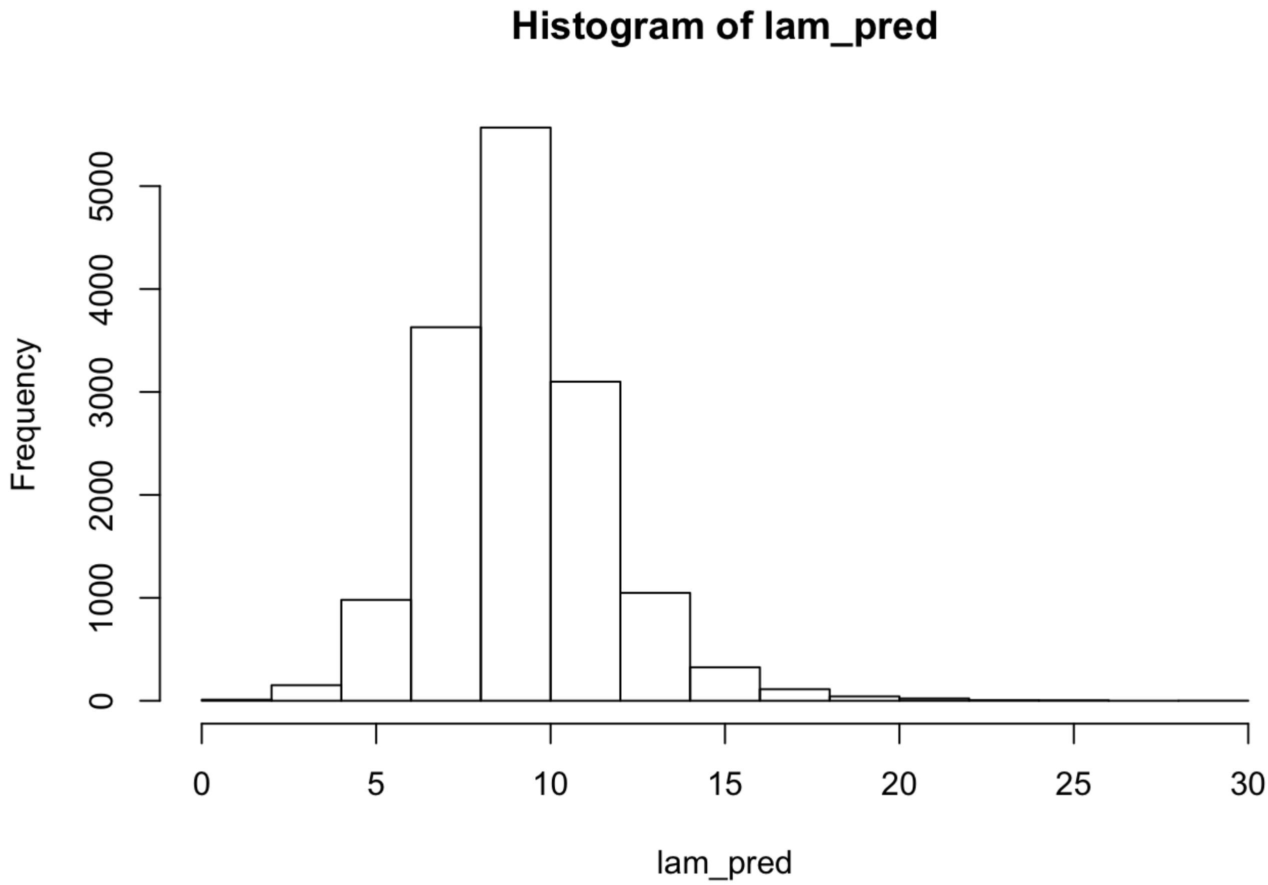
Just as we did with the prior distribution, we can use these posterior samples to get Monte Carlo estimates that interest us from the posterior predictive distribution.

For example, we can use draws from the posterior distribution of \(\mu\) and \(\sigma\) to simulate the posterior predictive distribution of the mean for a new location.

```
(n_sim = nrow(mod_csim))
```

```
## [1] 15000
```

```
lam_pred = rgamma(n=n_sim, shape=mod_csim[,"mu"]^2/mod_csim[,"sig"]^2,
                  rate=mod_csim[,"mu"]/mod_csim[,"sig"]^2)
hist(lam_pred)
```

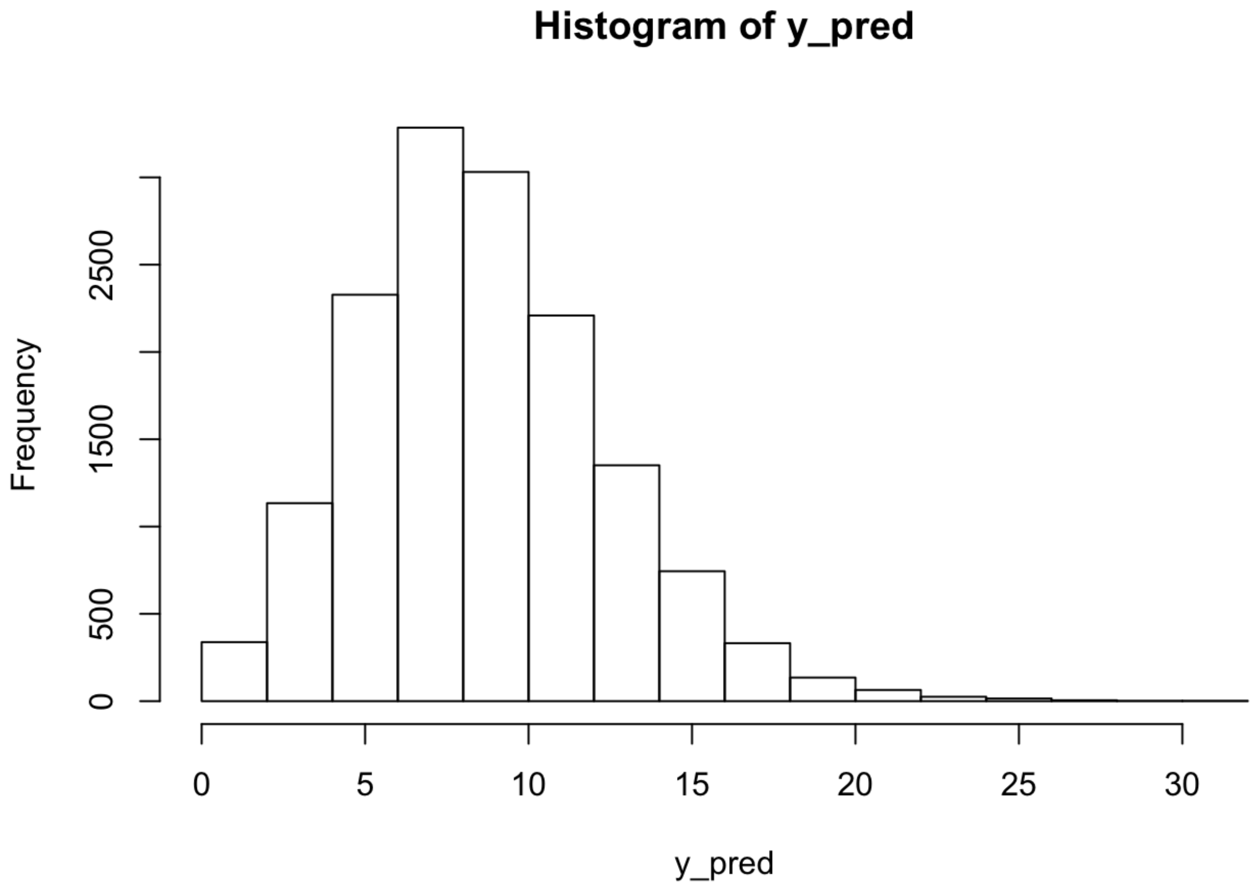



```
mean(lam_pred > 15)
```

```
## [1] 0.02073333
```

Using these \(\lambda\) draws, we can go to the observation level and simulate the number of chips per cookie, which takes into account the uncertainty in \(\lambda\):

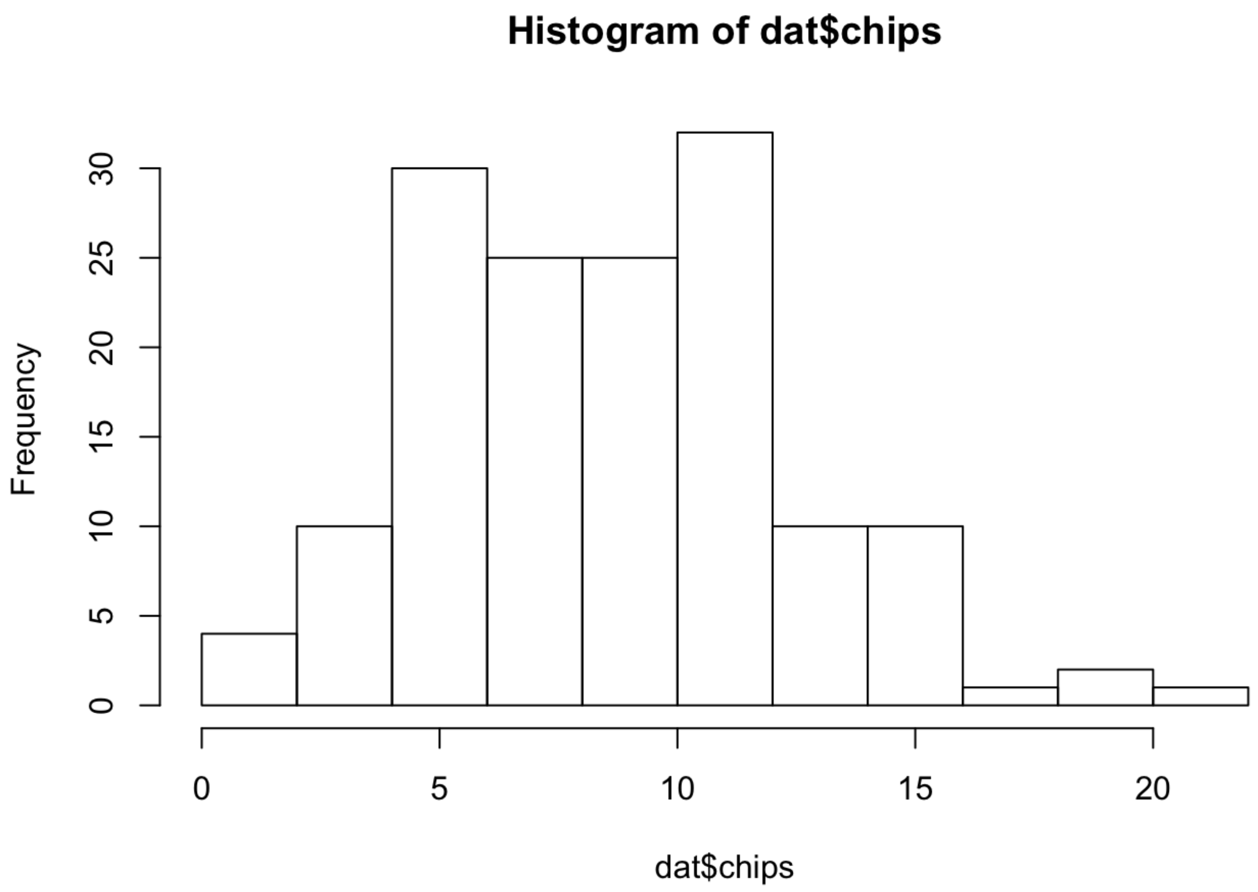
```
y_pred = rpois(n=n_sim, lambda=lam_pred)
hist(y_pred)
```




```
mean(y_pred > 15)
```

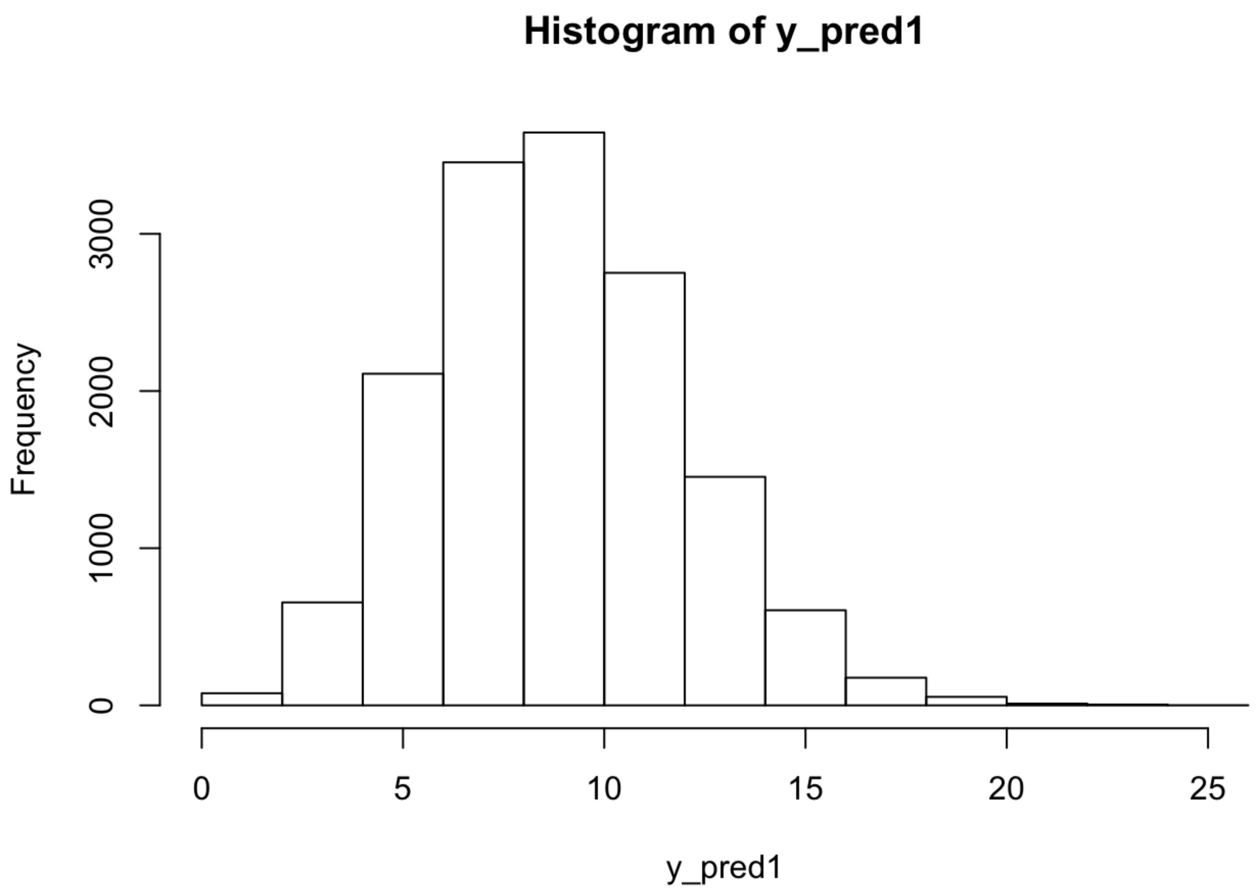
```
## [1] 0.05746667
```

```
hist(dat$chips)
```



Finally, we could answer questions like: what is the posterior probability that the next cookie produced in Location 1 will have fewer than seven chips?

```
y_pred1 = rpois(n=n_sim, lambda=mod_csim[,"lam[1]"])
hist(y_pred1)
```




```
mean(y_pred1 < 7)
```

```
## [1] 0.1894667
```

Lesson 11.6

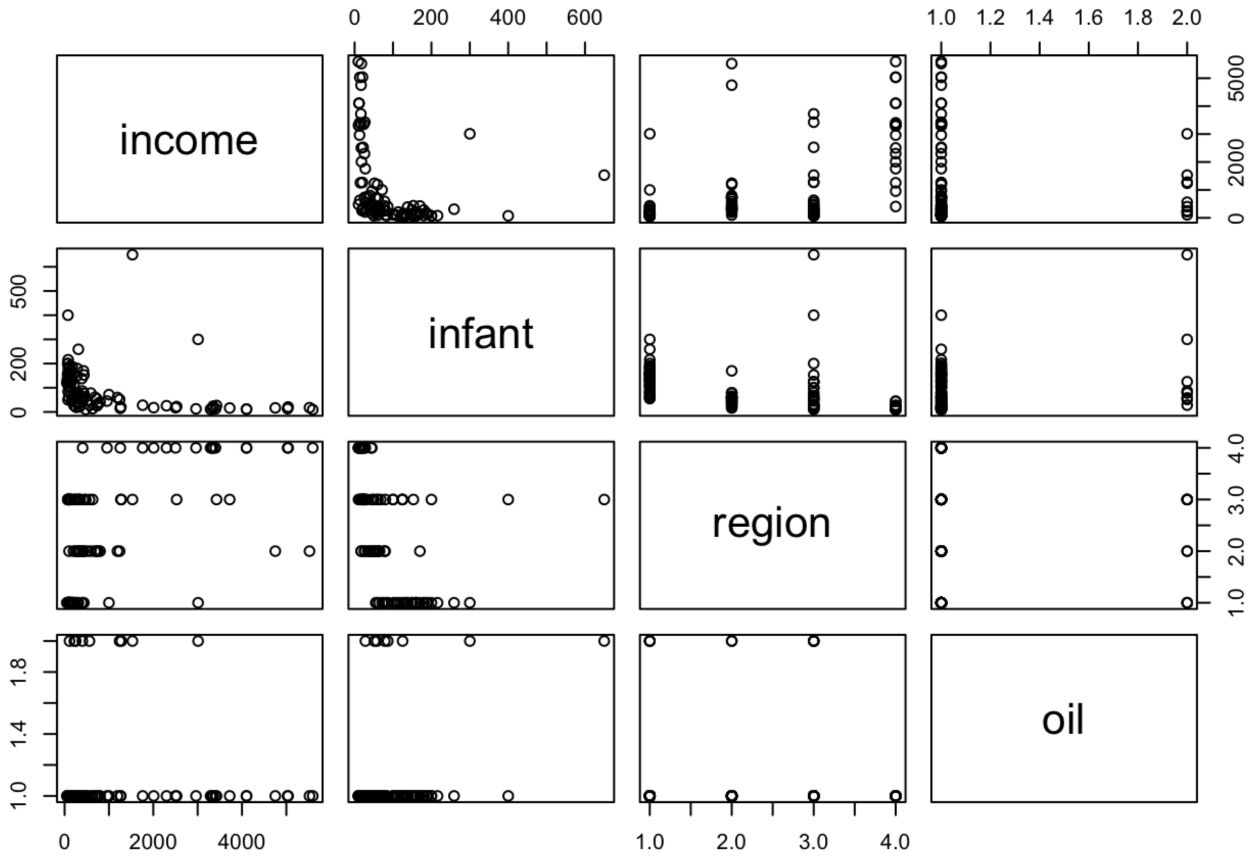
Random intercept linear model

We can extend the linear model for the Leinhardt data on infant mortality by incorporating the `region` variable. We'll do this with a hierarhcical model, where each region has its own intercept.

```
library("car")
data("Leinhardt")
?Leinhardt
str(Leinhardt)
```

```
## 'data.frame':    105 obs. of  4 variables:
##  $ income: int  3426 3350 3346 4751 5029 3312 3403 5040 2009 2298 ...
##  $ infant: num  26.7 23.7 17 16.8 13.5 10.1 12.9 20.4 17.8 25.7 ...
##  $ region: Factor w/ 4 levels "Africa","Americas",..: 3 4 4 2 4 4 4 4 4 4 ...
##  $ oil   : Factor w/ 2 levels "no","yes": 1 1 1 1 1 1 1 1 1 1 ...
```

```
pairs(Leinhardt)
```




```
head(Leinhardt)
```

```
##            income infant   region oil
## Australia    3426   26.7     Asia  no
## Austria      3350   23.7   Europe  no
## Belgium      3346   17.0   Europe  no
## Canada       4751   16.8 Americas  no
## Denmark      5029   13.5   Europe  no
## Finland      3312   10.1   Europe  no
```

Previously, we worked with infant mortality and income on the logarithmic scale. Recall also that we had to remove some missing data.

```
dat = na.omit(Leinhardt)
dat$logincome = log(dat$income)
dat$loginfant = log(dat$infant)
str(dat)
```

```
## 'data.frame':    101 obs. of  6 variables:
##  $ income   : int  3426 3350 3346 4751 5029 3312 3403 5040 2009 2298 ...
##  $ infant   : num  26.7 23.7 17 16.8 13.5 10.1 12.9 20.4 17.8 25.7 ...
##  $ region   : Factor w/ 4 levels "Africa","Americas",..: 3 4 4 2 4 4 4 4 4 4 ..
.
##  $ oil      : Factor w/ 2 levels "no","yes": 1 1 1 1 1 1 1 1 1 1 ...
##  $ logincome: num  8.14 8.12 8.12 8.47 8.52 ...
##  $ loginfant: num  3.28 3.17 2.83 2.82 2.6 ...
##  - attr(*, "na.action")=Class 'omit'  Named int [1:4] 24 83 86 91
##   .. ..- attr(*, "names")= chr [1:4] "Iran" "Haiti" "Laos" "Nepal"
```

Now we can fit the proposed model:

```r
library("rjags")

mod_string = " model {
  for (i in 1:length(y)) {
    y[i] ~ dnorm(mu[i], prec)
    mu[i] = a[region[i]] + b[1]*log_income[i] + b[2]*is_oil[i]
  }

  for (j in 1:max(region)) {
    a[j] ~ dnorm(a0, prec_a)
  }

  a0 ~ dnorm(0.0, 1.0/1.0e6)
  prec_a ~ dgamma(1/2.0, 1*10.0/2.0)
  tau = sqrt( 1.0 / prec_a )

  for (j in 1:2) {
    b[j] ~ dnorm(0.0, 1.0/1.0e6)
  }

  prec ~ dgamma(5/2.0, 5*10.0/2.0)
  sig = sqrt( 1.0 / prec )
} "

set.seed(116)
data_jags = list(y=dat$loginfant, log_income=dat$logincome,
                  is_oil=as.numeric(dat$oil=="yes"), region=as.numeric(dat$region)
)
data_jags$is_oil
table(data_jags$is_oil, data_jags$region)

params = c("a0", "a", "b", "sig", "tau")

mod = jags.model(textConnection(mod_string), data=data_jags, n.chains=3)
update(mod, 1e3) # burn-in

mod_sim = coda.samples(model=mod,
                       variable.names=params,
                       n.iter=5e3)

mod_csim = as.mcmc(do.call(rbind, mod_sim)) # combine multiple chains

## convergence diagnostics
plot(mod_sim)

gelman.diag(mod_sim)
autocorr.diag(mod_sim)
autocorr.plot(mod_sim)
effectiveSize(mod_sim)
```

Results

Convergence looks okay, so let’s compare this with the old model from Lesson 7 using DIC:

```
dic.samples(mod, n.iter=1e3)
```

```
## Mean deviance:  214 
## penalty 6.899 
## Penalized deviance: 220.8
```

```
# nonhierarchical model: 230.1
```

It appears that this model is an improvement over the non-hierarchical one we fit earlier. Notice that the penalty term, which can be interpreted as the “effective” number of parameters, is less than the actual number of parameters (nine). There are fewer “effective” parameters because they are “sharing” information or “borrowing strength” from each other in the hierarhical structure. If we had skipped the hierarchy and fit one intercept, there would have been four parameters. If we had fit separate, independent intercepts for each region, there would have been seven parameters (which is close to what we ended up with).

Finally, let’s look at the posterior summary.

```
summary(mod_sim)
```

```
## 
## Iterations = 1001:6000
## Thinning interval = 1 
## Number of chains = 3 
## Sample size per chain = 5000 
## 
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
## 
##         Mean      SD  Naive SE Time-series SE
## a[1]  6.5539 0.55172 0.0045048      0.0429737
## a[2]  6.0100 0.69213 0.0056513      0.0536065
## a[3]  5.8481 0.61824 0.0050479      0.0481237
## a[4]  5.5364 0.84626 0.0069097      0.0669336
## a0    5.9883 1.31893 0.0107690      0.0491634
## b[1] -0.3414 0.10461 0.0008541      0.0085398
## b[2]  0.6462 0.35042 0.0028612      0.0039220
## sig   0.9183 0.06494 0.0005303      0.0005924
## tau   2.0420 1.04278 0.0085142      0.0114027
## 
## 2. Quantiles for each variable:
## 
##          2.5%     25%     50%     75%   97.5%
## a[1]  5.49401  6.1784  6.5327  6.9151  7.6849
## a[2]  4.68032  5.5470  5.9737  6.4617  7.4473
## a[3]  4.64803  5.4353  5.8211  6.2497  7.1140
## a[4]  3.92103  4.9715  5.4934  6.0856  7.2902
## a0    3.47652  5.1829  5.9858  6.7717  8.5699
## b[1] -0.55847 -0.4100 -0.3357 -0.2704 -0.1386
## b[2] -0.04912  0.4141  0.6494  0.8782  1.3285
## sig   0.80286  0.8727  0.9138  0.9595  1.0567
## tau   0.97505  1.4062  1.7772  2.3481  4.6814
```

In this particular model, the intercepts do not have a real interpretation because they correspond to the mean response for a country that does not produce oil and has $0 log-income per capita (which is $1 income per capita). We can interpret \(a_0\) as the overall mean intercept and \(\tau\) as the standard deviation of intercepts across regions.

Other models

We have not investigated adding interaction terms, which might be appropriate. We only considered adding hierarchy on the intercepts, but in reality nothing prevents us from doing the same for other terms in the model, such as the coefficients for income and oil. We could try any or all of these alternatives and see how the DIC changes for those models. This, together with other model checking techniques we have discussed could be used to identify your *best* model that you can use to make inferences and predictions.