Lecture 18: Poisson Regression

** This lecture is based on chapter 10 of Statistical Rethinking by Richard McElreath.*

```
library(rstan)
library(shinystan)
library(car)
library(mvtnorm)
library(rethinking)
library(MASS)
rstan_options(auto_write = TRUE)
options(mc.cores = parallel::detectCores())
source("../utilityFunctions.R")
```

When a binomial distribution has a small chance of an event happening (p) and a large number of trials N, an interesting thing occurs. Remember:

- the expectation (i.e., mean) of a Binomial is Np
- the variance is $Np(1-p)$.

When the number of trials is large and the probability of occurrence is small, the mean and the variance become equal.

For example, suppose that you study a very rare trillium. You have funding for an army of undergrads *ca.* (1000) to go out into the field to find specimens.

- On any given day, ≈ 1 finds a trillium individual.

If each of your minions is working independently, and the acquisition of new samples is stochastic, some days you might get 2 or more samples, other days your army might come up empty.

This is a binomial process:

- the mean is $Np = 1000(0.001) = 1$
- the variance is $Np(1-p) = 1000(0.001) \times (1 - 0.001) \approx 1$.

We can simulate this process as well over the course of a graduate career or the first couple of years as faculty:

```
set.seed(3)
trillium <- rbinom(2000, 1000, 1/1000)
c(mean(trillium), var(trillium))
```

```
[1] 1.000000 1.002501
```

The mean and variance are almost identical! This special case of the binomial distribution is the *Poisson distribution*. This distribution allows us to model binomial events when the number of trials N is unknown or too large to count.

Models built with a Poisson likelihood are simpler than models for binomial or normal likelihoods because we only have one parameter that describes the distribution's shape:

λ, the expected value for observation y.

As we discussed two classes ago, we need a link function for the Poisson. The conventional link function is the log link.

To embed a linear model, we use

$$y_i \sim \text{Poisson}(\lambda_i)$$
$$\log(\lambda_i) = \alpha + \beta x_i.$$

The log link keeps λ from becoming negative, which is necessary for the expectation of count outcomes. But, it implies an exponential relationship between predictors and expected values.

- It is important to validate whether the log link provides sensible estimates across the range of predictors.

The parameter λ is interpreted as the mean, but it is also thought of as a rate. This allows us to make models where the *exposure* varies across observations.

Example: suppose another lab studies the same rare trillium species but records the accumulation of new samples by week.

- If we wanted to know the average population size using the concatenated dataset, how could we do this given the counts are aggregated over different periods of time (i.e., different exposures)?

λ is the expected expected number of events μ, per unit time (or distance). This implies

$$\lambda = \frac{\mu}{\tau}.$$

Redefining our link,

$$y_i \sim \text{Poisson}(\lambda_i)$$
$$\log(\lambda_i) = \log \frac{\mu_i}{\tau_i} = \alpha + \beta x_i.$$

The logarithm of a ratio can be rewritten as the difference in logs:

$$\log(\lambda_i) = \log \mu_i - \log \tau_i = \alpha + \beta x_i.$$

The $\tau's$ are the exposures. If different observations have different exposures, the expected value for observation i is

$$\log \mu_i = \alpha + \beta x_i + \log \tau_i.$$

If $\tau = 1$, $\log \tau = 0$ and it drops out. But when the exposure varies with samples, τ correctly scales the expected number of events for the ith sample.

So we can model different exposures by reformulating our likelihood function as

$$y_i \sim \text{Poisson}(\mu_i)$$
$$\log(\mu_i) = \alpha + \beta x_i + \log \tau_i$$

where τ_i is essentially another predictor but without adding another parameter.

Poisson regression in Stan

For this example, we will use a dataset (`kline.csv`) from island societies in Oceania. Different island populations had different tool repertoires (e.g., fish hooks, axes, boats).

- Several theories predict larger populations develop and maintain more diverse tool kits.
- It's also suggested contact rates among populations increase population size, as it's relevant to technological evolution. So variation in contact rates may also be relevant.

```
kline <- read.csv("kline.csv")
head(kline)
```

```
     culture  pop contact tools mean_TU
1   Malekula 1100     low    13     3.2
2    Tikopia 1500     low    22     4.7
3 Santa Cruz 3600     low    24     4.0
4        Yap 4791    high    43     5.0
5   Lau Fiji 7400    high    33     5.0
6  Trobriand 8000    high    19     4.0
```

A map of the societies is in Fig. 1.

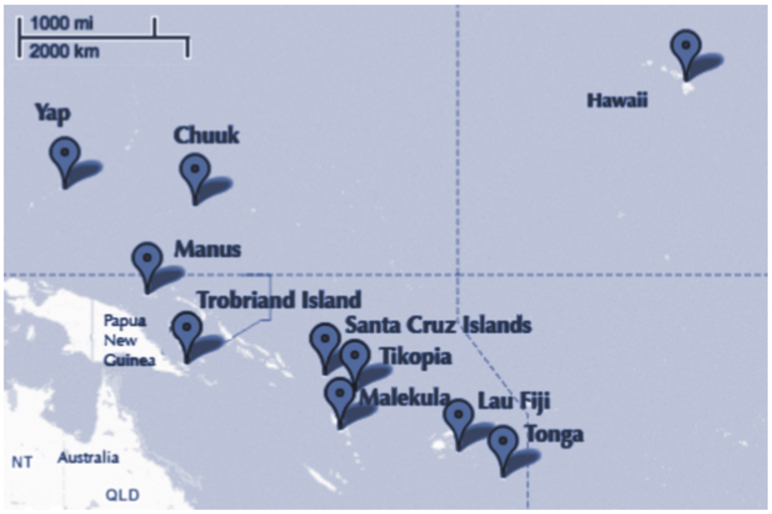


Figure 1: Locations of societies in the Kline data.

This dataset has three variables:

1. `tools`: the total number of tools (response variable).
2. `pop`: the population size. We would predict tool number would increase with the log of population size because only the magnitude really matters.
3. `contact`: the contact rate. More networked islands should have more tool types.

We might also expect an interaction between population size and contact rate because larger populations probably have more contact.

```r
# sort dataframe by pop size.
dat <- kline[order(kline$pop),]
obs <- dat$tools
contact <- ifelse(dat$contact=="low", 0, 1)
pop <- log(dat$pop)
nObs <- nrow(dat)
# make design matrix
predMat <- model.matrix(~pop*contact)

# make design matrix for new simulated data
newPop <- rep(seq(6, 13, length=30), 2)
newCont <- rep(0:1, each=30)
newMat <- model.matrix(~newPop*newCont)
nNew <- nrow(newMat)
```

Here is our model:

$$
\begin{aligned}
y_i &\sim \text{Poisson}(\lambda_\text{i}) \\
\log \lambda_i &= \beta_0 + \beta_p \log P_i + \beta_c C_i + \beta_{pc} C_i \log P_i \\
\beta_0 &\sim \text{Normal}(0, 10) \\
\beta_{k>0} &\sim \text{Normal}(0, 1).
\end{aligned}
$$

- Because the sample size is small, we will use stronger regularizing priors on the slopes.
- The intercept has weaker priors because we don't necessarily know where our intercept might end up, especially with this sample size.

Fitting this model in Stan is easy. However, I have added some things for more flexibility.

1. We are still using the design matrix parameterization of the linear model such that we are pre-multiplying a matrix of predictors by a vector of $\beta's$.

- However, in the model block, we can use indexing to specify distinct priors for the intercept.

```
...

parameters {
  vector[nVar] beta;
}

transformed parameters {
  vector[nObs] lambda;

  lambda = X * beta;
}
model {
  beta[1] ~ normal(bMu, b0SD);
  beta[2:nVar] ~ normal(bMu, bSD);

  obs ~ poisson_log(lambda);
}
```

2. In the generated quantities block, we are estimating fitted values for new data (`newX`)—an `nNew` × `nVarN` design matrix—that covers a more complete sequence of `log(pop)` values for high and low contact.

- These are new counterfactual data that make prettier plots (among other things).

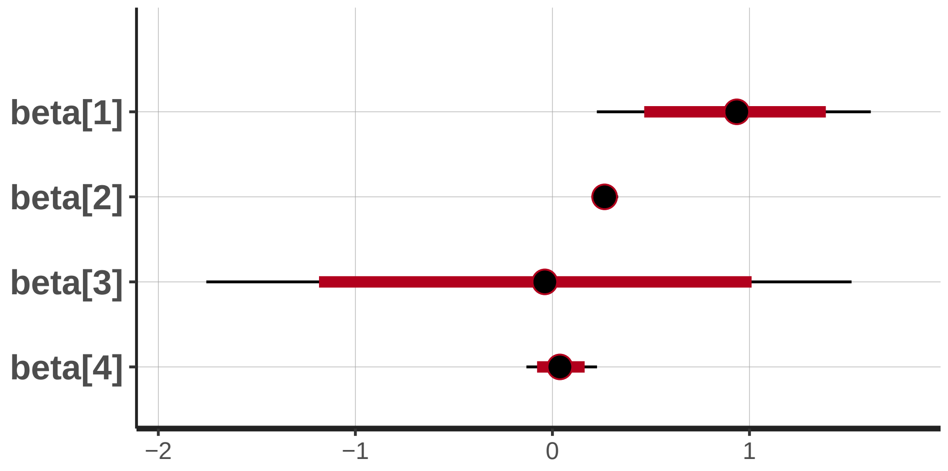
```
...

generated quantities {
  vector[nObs] log_lik;
  vector[nNew] newLam;

  for(n in 1:nObs)
  log_lik[n] = poisson_log_lpmf(obs[n]|lambda[n]);

  newLam = newX * beta;
}
```

Now, let's run the complete model:

```
#Specify design matrices
X <- predMat
newX <- newMat

# declare data list
d1 <- list(nObs=nObs, nVar=ncol(X), nNew=nNew, nVarN=ncol(newX),
  obs=obs, X=X, newX=newX, b0SD=10, bMu=0, bSD=1)

# run model
m1 <- stan(file="poissonMod.stan", data=d1, iter=2000, chains=4,
  seed=867.5309, pars="lambda", include=FALSE)

# extract betas and print result
beta <- as.matrix(m1, "beta")
print(m1,"beta")
```

```
Inference for Stan model: poissonMod.
4 chains, each with iter=2000; warmup=1000; thin=1;
post-warmup draws per chain=1000, total post-warmup draws=4000.

         mean se_mean   sd  2.5%   25%   50%  75% 97.5% n_eff Rhat
beta[1]  0.93    0.01 0.36  0.23  0.69  0.94 1.17  1.62  1459    1
beta[2]  0.27    0.00 0.03  0.20  0.24  0.26 0.29  0.33  1434    1
beta[3] -0.07    0.02 0.85 -1.76 -0.67 -0.04 0.52  1.52  1239    1
beta[4]  0.04    0.00 0.09 -0.13 -0.02  0.04 0.11  0.23  1272    1

Samples were drawn using NUTS(diag_e) at Sun Apr 22 12:22:11 2018.
For each parameter, n_eff is a crude measure of effective sample size,
and Rhat is the potential scale reduction factor on split chains (at
convergence, Rhat=1).
```

```
par(mar=c(3,3,0.1,0.5))
plotBeta <- plot(m1, pars="beta")
plotBeta + theme(text=element_text(family="ArialMT"))
```



A quick visual inspection of the results shows that, although the slope for log population size (`beta[2]`) is precise and positive, both `beta[3]`(difference between low and high contact) and `beta[4]` (interaction) overlap zero substantially.

Thus, on first glance, we might assume population size reliably indicates an increase in tool complexity, but contact rate has no impact.

However, this is a misnomer. Let's consider two islands with the same population size (8) but different contact rates (i.e., one high, one low).

- We will calculate λ, the expected tool count for each one by using the posterior estimates of beta and then inverting the link through exponentiation.

```
lamLo <- exp(beta[,1] + beta[,2]*8)
lamHi <- exp(beta[,1] + beta[,3] + (beta[,2] + beta[,4])*8)
```

Now, lets calculate the difference in distributions between the two hypothetical islands.

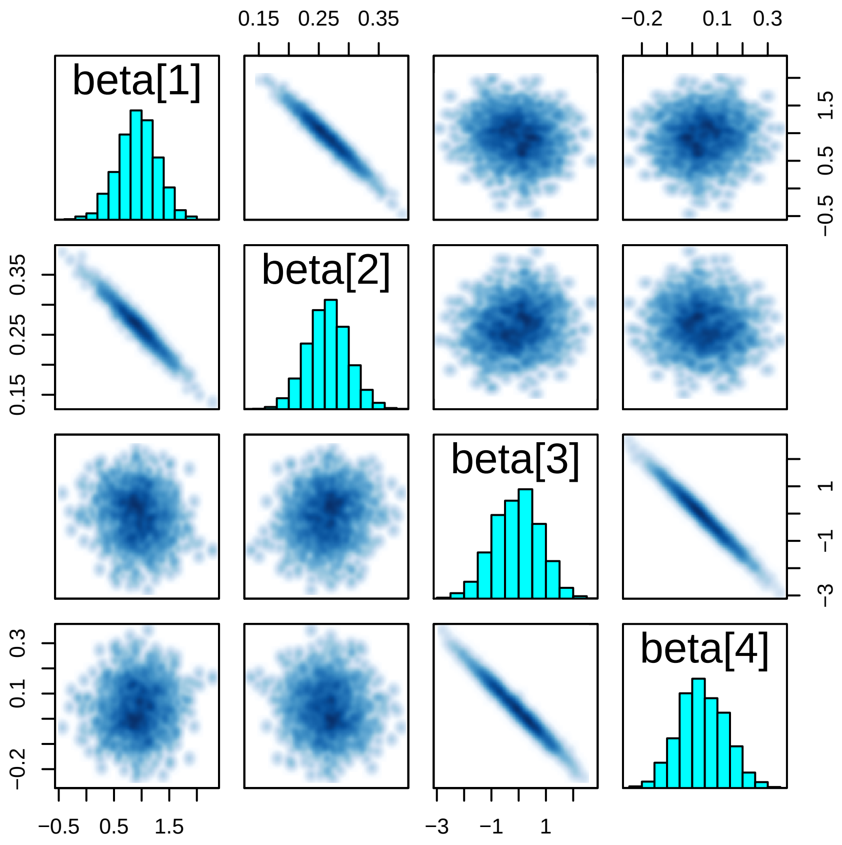
```
diff <- lamHi - lamLo
round(sum(diff>0)/length(diff),3)
```

```
[1] 0.954
```

There is 0.95% plausibility that the high contact island has more tools, despite substantial overlap in the marginal distributions of the slopes!

One reason for this counterintuitive result can be found by looking at the correlations between parameters:

```
pairs(m1, pars="beta")
```



The correlation between `beta[3]` and `beta[4]` are strongly negative. Therefore we cannot get an accurate understanding of what is happening by looking at the marginal distributions.

One way we can assess the importance of predictors such as contact rate is to use model comparison.

- Because they are on the scale of predicted outcomes, they account for correlations among parameters.

Let's fit some less complex models:

1. A model without an interaction:

```
X <- predMat[,1:3]
newX <- newMat[,1:3]

d2 <- list(nObs=nObs, nVar=ncol(X), nNew=nNew, nVarN=ncol(newX),
  obs=obs, X=X, newX=newX, b0SD=10, bMu=0, bSD=1)

m2 <- stan(file="poissonMod.stan", data=d2, iter=2000, chains=4,
  seed=867.5309, pars="lambda", include=FALSE)

beta1 <- as.matrix(m1, "beta")
```

2. A model with just population size

```r
X <- predMat[,1:2]

newX <- newMat[,1:2]

d3 <- list(nObs=nObs, nVar=ncol(X), nNew=nNew, nVarN=ncol(newX),
  obs=obs, X=X, newX=newX, b0SD=10, bMu=0, bSD=1)

m3 <- stan(file="poissonMod.stan", data=d3, iter=2000, chains=4,
  seed=867.5309, pars="lambda", include=FALSE)
```

3. A model with just contact rate

```r
X <- predMat[,c(1,3)]
newX <- newMat[,c(1,3)]

d4 <- list(nObs=nObs, nVar=ncol(X), nNew=nNew, nVarN=ncol(newX),
  obs=obs, X=X, newX=newX, b0SD=10, bMu=0, bSD=1)

m4 <- stan(file="poissonMod.stan", data=d4, iter=2000, chains=4,
  seed=867.5309, pars="lambda", include=FALSE)
```

```r
compare(m1, m2, m3, m4)
```

```
    WAIC pWAIC dWAIC weight    SE   dSE
m2  79.1   4.3   0.0   0.59 11.26    NA
m1  80.1   4.8   1.0   0.36 11.21  1.36
m3  84.0   3.5   4.9   0.05  8.88  8.45
m4 151.4  17.4  72.3   0.00 45.85 48.13
```

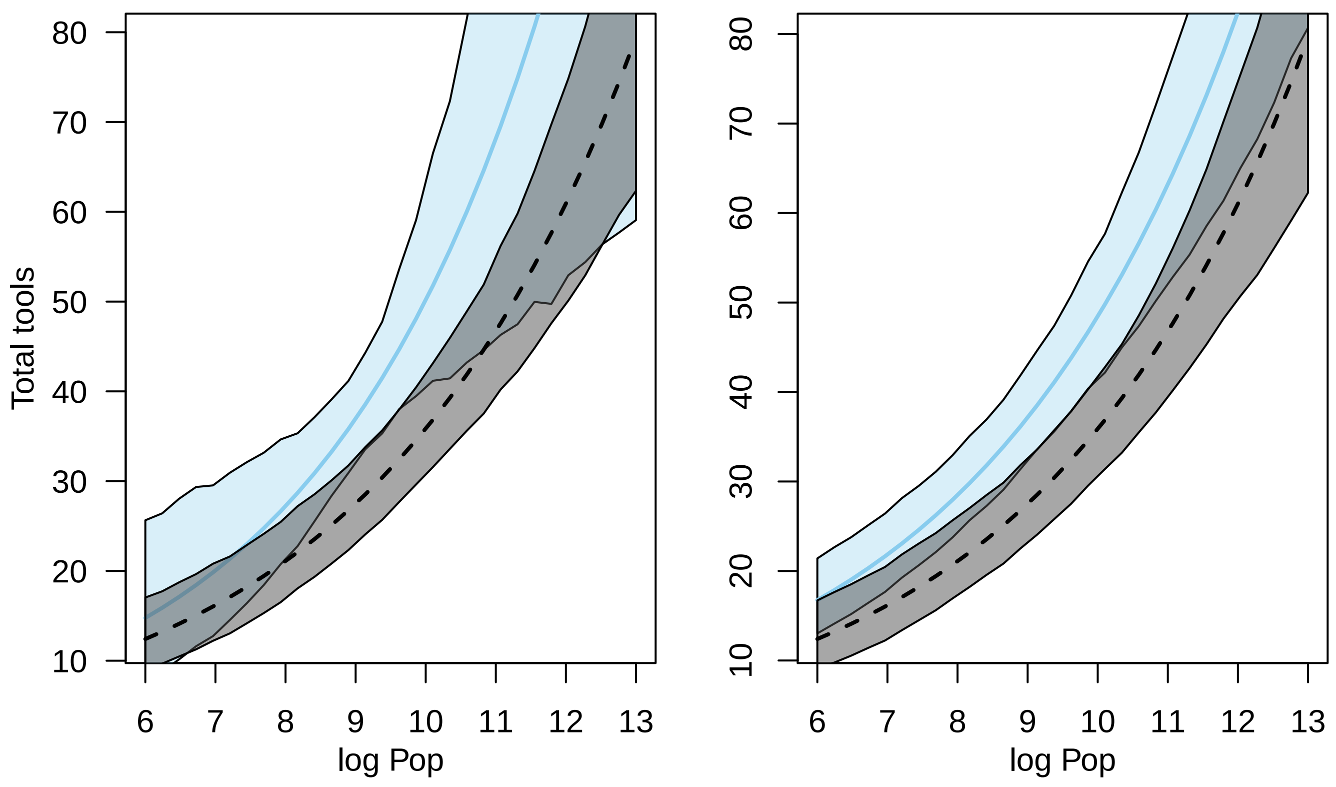
The best model is a model with both predictors but no interaction (`m2`). However, the model with the interaction has about $\frac{1}{3}$ of the model weight.

- Suggests we may be overfitting by including the interaction term.

We could average them, but it might also make sense to plot the results of the two models side by side to see how they differ:

```r
par(mar=c(3,3.2,0.1,0.5))
par(mfrow=c(1,2))

# interaction model
lam <- as.matrix(m1, pars="newLam")
avLam <- exp(colMeans(lam))
hdiLam <- exp(apply(lam, 2, HDI, credMass=0.95))

avLo <- avLam[newCont==0]
avHi <- avLam[newCont==1]

hdiLo <- hdiLam[, newCont==0]
hdiHi <- hdiLam[, newCont==1]
```

```r
x <- newPop[1:30]
plot(x, avLo, type="n", ann=FALSE, las=1)
mtext("log Pop", side=1, line=2)
mtext("Total tools", side=2, line=2.2)

polygon(c(x, rev(x)), c(hdiHi[1, ], rev(hdiHi[2,])),
  col="#88CCEE50")
lines(x,avHi, lwd=2, col="#88CCEE")

polygon(c(x, rev(x)), c(hdiLo[1, ], rev(hdiLo[2,])),
  col="#50505080")
lines(x,avLo, lwd=2, lty=2)


lam <- as.matrix(m2, pars="newLam")
avLam <- exp(colMeans(lam))
hdiLam <- exp(apply(lam, 2, HDI, credMass=0.95))

avLo <- avLam[newCont==0]
avHi <- avLam[newCont==1]

hdiLo <- hdiLam[, newCont==0]
hdiHi <- hdiLam[, newCont==1]

plot(x, avLo, type="n", ann=FALSE)
mtext("log Pop", side=1, line=2)

polygon(c(x, rev(x)), c(hdiHi[1, ], rev(hdiHi[2,])),
  col="#88CCEE50")
lines(x,avHi, lwd=2, col="#88CCEE")

polygon(c(x, rev(x)), c(hdiLo[1, ], rev(hdiLo[2,])),
  col="#50505080")
lines(x,avLo, lwd=2, lty=2)
```



As you can see, the results are almost identical, although the uncertainty is much lower in the model without the interaction term (right plot).

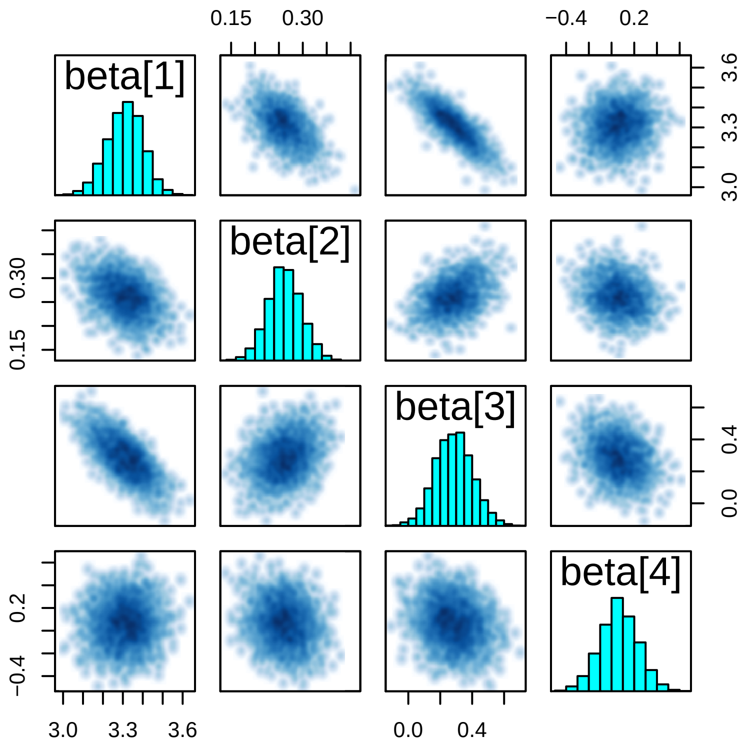
Even though HMC is pretty good at dealing with correlated predictors, it isn't immune. One solution is to center. Let's recenter `log Pop` and look at how things change:

```r
cX <- model.matrix(~scale(pop, scale = FALSE) * contact)
cnewX <- model.matrix(~scale(newPop, scale=FALSE) * newCont)

dc <- list(nObs=nObs, nVar=ncol(cX), nNew=nNew, nVarN=ncol(cnewX),
  obs=obs, X=cX, newX=cnewX, b0SD=10, bMu=0, bSD=1)

mc <- stan(file="poissonMod.stan", data=dc, iter=2000, chains=4,
  seed=867.5309, pars="lambda", include=FALSE)

cBeta <- as.matrix(mc, "beta")
print(mc, "beta")
```

```
Inference for Stan model: poissonMod.
4 chains, each with iter=2000; warmup=1000; thin=1;
post-warmup draws per chain=1000, total post-warmup draws=4000.

        mean se_mean   sd  2.5%   25%  50%  75% 97.5% n_eff Rhat
beta[1] 3.31       0 0.09  3.13  3.25 3.31 3.37  3.48  1521    1
beta[2] 0.26       0 0.04  0.19  0.24 0.26 0.29  0.33  2191    1
beta[3] 0.29       0 0.12  0.06  0.21 0.28 0.36  0.52  1777    1
beta[4] 0.07       0 0.17 -0.26 -0.05 0.07 0.18  0.39  3181    1
```

```
Samples were drawn using NUTS(diag_e) at Mon Apr 23 09:53:16 2018.
For each parameter, n_eff is a crude measure of effective sample size,
and Rhat is the potential scale reduction factor on split chains (at
convergence, Rhat=1).
```

```r
pairs(mc, pars="beta")
```



By centering, we have almost doubled our effective sample size and (at least partially) removed the correlation among parameters.

Offsets

When the length of observation, area of sampling, or sampling intensity varies, the counts we observe can vary as well.

- A well known example of this is in the estimation of species richness or species abundance.

Lets simulate an example and make a model to answer it.

Suppose we are sampling trillium in GA with an average abundance of $\lambda = 5$ ind., sampled from 30 sites over the course of a day per site.

We can simulate a dataset of 30 days:

```r
yD <- rpois(30, 5)
```

Now suppose we want to compare the abundance of our sample from a bio-blitz in TN, where 4 volunteers looked for the species over the course of a week.

Suppose the abundance in TN is actually $\lambda = 3$. To simulate the data from this dataset, we just multiply this average by 7, the exposure.

```
yW <- rpois(4, 3*7)
```

To analyze both the daily abundance counts and weekly abundance counts, we just add the log of exposure to the linear model. Let's build the predictors:

```
obs <- c(yD, yW)

expose <- c(rep(1,30), rep(7, 4))

state <- c(rep(0,30), rep(1, 4))
```

Now, we can make a model and estimate the average trillium abundance in each state.

- We do this by computing the log of each exposure and including the variable in a linear model, but without estimating a parameter.

```
data {
  int<lower=0> nObs;
  int<lower=0> obs[nObs];
  vector<lower=0>[nObs] expose;
  vector[nObs] state;
}

parameters {
  real alpha;
  real beta;
}

transformed parameters {
  vector[nObs] lambda;

  lambda =alpha + beta*state + log(expose);
}
model {
 alpha ~ normal(0, 10);
  beta ~ normal(0, 1);

  obs ~ poisson_log(lambda);
}
```

Our parameter estimates from the model now have the offset included in them, and thus they are averages on the same scale.

```
dex <- list(nObs=length(obs), expose=expose, state=state,
  obs=obs)

mEx <- stan(file="exposeMod.stan", data=dex, iter=2000, chains=4,
  seed=867.5309, pars="lambda", include=FALSE)

pars <- as.data.frame(mEx, c("alpha", "beta"))

GA <- quantile(exp(pars$alpha), c(0.025, 0.5, 0.975))
TN <- quantile(exp(rowSums(pars)), c(0.025, 0.5, 0.975))
rbind(GA,TN)
```

```
       2.5%      50%    97.5%
GA 4.327588 5.070874 5.936835
TN 2.368304 2.936806 3.596880
```