Chapter 11 Solutions

Easy

https://rpubs.com/taixingbi/ana505-s9

1. If an event has probability .35, what are the log-odds of this event?

```
log_odds_event = log(.35/.65)
log_odds_event
```

```
## [1] -0.6190392
```

2. If an event has log-odds 3.2, what is the probability of this event?

```
#3.2 = log(x / 1-x)
#3.2 = log(x) - log(1-x)
#exp(3.2) = x - 1 + x
.5*(exp(3.2) + 1) -> x
x
```

```
## [1] 12.76627
```

3. Suppose that a coefficient in a logistic regression has value 1.7. What does this imply about the proportional change in odds of the outcome?

```
exp(1.7)
```

```
## [1] 5.473947
```

4. Why do Poisson regressions sometimes require the use of an offset?

In some cases, different observations are aggregated over different time frames. For example one observation in the data might be 3 manuscripts (in a day), another observations might be 10 manuscripts (in a week). The offset is used to compare all the observations on an apples-to-apples scale.

Medium

1. Binomial data can be organized in aggregated and disaggregated forms, without any impact on inference. But the likelihood of the data does change when the data are converted between the two formats. Can you explain why?

Because when converting likelihood in the aggregated form to the non-aggregated format, the c(n,m) multiplier is converted to a constant at the log scale

2. If a coefficient in a Poisson regression has value 1.7, what does this imply about the change in the outcome?

It indicates that the change of the predictor by 1 unit increases the lambda parameter of the Poisson distribution by exp(1.7)=5.4739 times.

3. Explain why the logit link is appropriate for the binomial GLM.

The logit link maps a parameter that is defined as a probability mass (therefore constrained to lie between zero and one) onto a linear model that can take any real value.

4. Explain why the log link is appropriate for a Poisson GLM.

The log link constrains lambda to be positive, and poisson models model counts (ie, things that can't be negative)

5. What would it imply to use a logit link for the mean of a Poisson GLM? Can you think of a real research problem for which this would make sense?

Using logit link implies that lambda always falls on [0, +inf]

6. State the constraints for which the binomial and Poisson distributions have maximum entropy. Are the constraints different between the two? Why or why not?

Binomial: Events are discrete and expected value is constant. Poisson: Variance is equal to expected value and both are constant

Poisson distributions have more constraints than binomial because it's a special case of the binomial.

Hard

1. Use quap to construct a quadratic approximate posterior distribution for the chimpanzee model that includes a unique intercept for each actor, m11.4.

```
library(rethinking)
```

```
## Loading required package: rstan
```

```
## Loading required package: StanHeaders
```

```
## Loading required package: ggplot2
```

```
## rstan (Version 2.21.2, GitRev: 2e1f913d3ca3)
```

```
## For execution on a local, multicore CPU with excess RAM we recommend calling
## options(mc.cores = parallel::detectCores()).
## To avoid recompilation of unchanged Stan programs, we recommend calling
## rstan_options(auto_write = TRUE)
```

```
## Loading required package: parallel
```

```
## rethinking (Version 2.13)
```

```
##
## Attaching package: 'rethinking'
```

```
## The following object is masked from 'package:stats':
##
##     rstudent
```

```
data("chimpanzees")
d <- chimpanzees
d$treatment <- 1 + d$prosoc_left + 2*d$condition
d$recipient <- NULL
```

```
# build model using quap
flist <- alist(
  pulled_left ~ dbinom( 1 , p ) ,
  logit(p) <- a[actor] + b[treatment],
  a[actor] ~ dnorm(0,1.5),
  b[treatment] ~ dnorm(0,.5)
)

quap_model <- quap(flist, d)
```

```r
# re-construct the model the MCMC way
dat_list <- list(
pulled_left = d$pulled_left,
actor = d$actor,
treatment = as.integer(d$treatment) )

m11.4 <- ulam(
alist(
pulled_left ~ dbinom( 1 , p ) ,
logit(p) <- a[actor] + b[treatment] ,
a[actor] ~ dnorm( 0 , 1.5 ),
b[treatment] ~ dnorm( 0 , 0.5 )
) , data=dat_list , chains=4 , log_lik=TRUE )
```

```
## Trying to compile a simple C file
## Running /Library/Frameworks/R.framework/Resources/bin/R CMD SHLIB foo.c
## clang -I"/Library/Frameworks/R.framework/Resources/include" -DNDEBUG   -I"/Library/Frameworks/R.frame
## In file included from <built-in>:1:
## In file included from /Library/Frameworks/R.framework/Versions/3.6/Resources/library/StanHeaders/incl
## In file included from /Library/Frameworks/R.framework/Versions/3.6/Resources/library/RcppEigen/inclu
## In file included from /Library/Frameworks/R.framework/Versions/3.6/Resources/library/RcppEigen/inclu
## /Library/Frameworks/R.framework/Versions/3.6/Resources/library/RcppEigen/include/Eigen/src/Core/util/
## namespace Eigen {
## ^
## /Library/Frameworks/R.framework/Versions/3.6/Resources/library/RcppEigen/include/Eigen/src/Core/util/
## namespace Eigen {
##                 ^
##                 ;
## In file included from <built-in>:1:
## In file included from /Library/Frameworks/R.framework/Versions/3.6/Resources/library/StanHeaders/incl
## In file included from /Library/Frameworks/R.framework/Versions/3.6/Resources/library/RcppEigen/inclu
## /Library/Frameworks/R.framework/Versions/3.6/Resources/library/RcppEigen/include/Eigen/Core:96:10: fa
## #include <complex>
##          ^~~~~~~~~
## 3 errors generated.
## make: *** [foo.o] Error 1
##
## SAMPLING FOR MODEL '82480dff1a626a42c2ca9de938d65b9d' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 7.8e-05 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.78 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:   1 / 1000 [  0%]  (Warmup)
## Chain 1: Iteration: 100 / 1000 [ 10%]  (Warmup)
## Chain 1: Iteration: 200 / 1000 [ 20%]  (Warmup)
## Chain 1: Iteration: 300 / 1000 [ 30%]  (Warmup)
## Chain 1: Iteration: 400 / 1000 [ 40%]  (Warmup)
## Chain 1: Iteration: 500 / 1000 [ 50%]  (Warmup)
## Chain 1: Iteration: 501 / 1000 [ 50%]  (Sampling)
## Chain 1: Iteration: 600 / 1000 [ 60%]  (Sampling)
## Chain 1: Iteration: 700 / 1000 [ 70%]  (Sampling)
```

```
## Chain 1: Iteration: 800 / 1000 [ 80%]  (Sampling)
## Chain 1: Iteration: 900 / 1000 [ 90%]  (Sampling)
## Chain 1: Iteration: 1000 / 1000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 0.301687 seconds (Warm-up)
## Chain 1:                0.224243 seconds (Sampling)
## Chain 1:                0.52593 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL '82480dff1a626a42c2ca9de938d65b9d' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 3.9e-05 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0.39 seconds.
## Chain 2: Adjust your expectations accordingly!
## Chain 2:
## Chain 2:
## Chain 2: Iteration:   1 / 1000 [  0%]  (Warmup)
## Chain 2: Iteration: 100 / 1000 [ 10%]  (Warmup)
## Chain 2: Iteration: 200 / 1000 [ 20%]  (Warmup)
## Chain 2: Iteration: 300 / 1000 [ 30%]  (Warmup)
## Chain 2: Iteration: 400 / 1000 [ 40%]  (Warmup)
## Chain 2: Iteration: 500 / 1000 [ 50%]  (Warmup)
## Chain 2: Iteration: 501 / 1000 [ 50%]  (Sampling)
## Chain 2: Iteration: 600 / 1000 [ 60%]  (Sampling)
## Chain 2: Iteration: 700 / 1000 [ 70%]  (Sampling)
## Chain 2: Iteration: 800 / 1000 [ 80%]  (Sampling)
## Chain 2: Iteration: 900 / 1000 [ 90%]  (Sampling)
## Chain 2: Iteration: 1000 / 1000 [100%]  (Sampling)
## Chain 2:
## Chain 2:  Elapsed Time: 0.277461 seconds (Warm-up)
## Chain 2:                0.235716 seconds (Sampling)
## Chain 2:                0.513177 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL '82480dff1a626a42c2ca9de938d65b9d' NOW (CHAIN 3).
## Chain 3:
## Chain 3: Gradient evaluation took 3.7e-05 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0.37 seconds.
## Chain 3: Adjust your expectations accordingly!
## Chain 3:
## Chain 3:
## Chain 3: Iteration:   1 / 1000 [  0%]  (Warmup)
## Chain 3: Iteration: 100 / 1000 [ 10%]  (Warmup)
## Chain 3: Iteration: 200 / 1000 [ 20%]  (Warmup)
## Chain 3: Iteration: 300 / 1000 [ 30%]  (Warmup)
## Chain 3: Iteration: 400 / 1000 [ 40%]  (Warmup)
## Chain 3: Iteration: 500 / 1000 [ 50%]  (Warmup)
## Chain 3: Iteration: 501 / 1000 [ 50%]  (Sampling)
## Chain 3: Iteration: 600 / 1000 [ 60%]  (Sampling)
## Chain 3: Iteration: 700 / 1000 [ 70%]  (Sampling)
## Chain 3: Iteration: 800 / 1000 [ 80%]  (Sampling)
## Chain 3: Iteration: 900 / 1000 [ 90%]  (Sampling)
## Chain 3: Iteration: 1000 / 1000 [100%]  (Sampling)
## Chain 3:
```

```
## Chain 3:  Elapsed Time: 0.283028 seconds (Warm-up)
## Chain 3:                0.293146 seconds (Sampling)
## Chain 3:                0.576174 seconds (Total)
## Chain 3:
##
## SAMPLING FOR MODEL '82480dff1a626a42c2ca9de938d65b9d' NOW (CHAIN 4).
## Chain 4:
## Chain 4: Gradient evaluation took 4.4e-05 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0.44 seconds.
## Chain 4: Adjust your expectations accordingly!
## Chain 4:
## Chain 4:
## Chain 4: Iteration:   1 / 1000 [  0%]  (Warmup)
## Chain 4: Iteration: 100 / 1000 [ 10%]  (Warmup)
## Chain 4: Iteration: 200 / 1000 [ 20%]  (Warmup)
## Chain 4: Iteration: 300 / 1000 [ 30%]  (Warmup)
## Chain 4: Iteration: 400 / 1000 [ 40%]  (Warmup)
## Chain 4: Iteration: 500 / 1000 [ 50%]  (Warmup)
## Chain 4: Iteration: 501 / 1000 [ 50%]  (Sampling)
## Chain 4: Iteration: 600 / 1000 [ 60%]  (Sampling)
## Chain 4: Iteration: 700 / 1000 [ 70%]  (Sampling)
## Chain 4: Iteration: 800 / 1000 [ 80%]  (Sampling)
## Chain 4: Iteration: 900 / 1000 [ 90%]  (Sampling)
## Chain 4: Iteration: 1000 / 1000 [100%]  (Sampling)
## Chain 4:
## Chain 4:  Elapsed Time: 0.293003 seconds (Warm-up)
## Chain 4:                0.207148 seconds (Sampling)
## Chain 4:                0.500151 seconds (Total)
## Chain 4:
```

Compare the quadratic approximation to the posterior distribution produced instead from MCMC.

```
compare(quap_model, m11.4)
```

```
## Warning in compare(quap_model, m11.4): Not all model fits of same class.
## This is usually a bad idea, because it implies they were fit by different algorithms.
## Check yourself, before you wreck yourself.
```

```
##                WAIC       SE    dWAIC       dSE    pWAIC    weight
## m11.4      531.4295 18.89486 0.000000        NA 8.119138 0.6877546
## quap_model 533.0088 18.53519 1.579285 0.4356427 8.364023 0.3122454
```

```
precis(quap_model, depth = 2)
```

```
##             mean        sd        5.5%       94.5%
## a[1] -0.43920225 0.3276017 -0.96277307  0.08436857
## a[2]  3.70604586 0.7217457  2.55255687  4.85953484
## a[3] -0.73274934 0.3329757 -1.26490875 -0.20058993
## a[4] -0.73274967 0.3329757 -1.26490910 -0.20059025
## a[5] -0.43920193 0.3276017 -0.96277275  0.08436888
## a[6]  0.46895248 0.3317747 -0.06128761  0.99919257
## a[7]  1.90506497 0.4136438  1.24398234  2.56614760
## b[1] -0.04066249 0.2837319 -0.49412086  0.41279587
## b[2]  0.47213915 0.2842163  0.01790667  0.92637162
## b[3] -0.37836011 0.2852441 -0.83423532  0.07751511
## b[4]  0.36201136 0.2838328 -0.09160825  0.81563096
```

```
precis(m11.4, depth = 2)
```

```
##             mean        sd        5.5%       94.5%    n_eff    Rhat4
## a[1] -0.43953196 0.3269034 -0.97669704  0.07849967 615.9055 1.004556
## a[2]  3.92782394 0.7575390  2.82399274  5.25392706 825.4970 1.001542
## a[3] -0.73402586 0.3392238 -1.29590991 -0.19296168 742.1706 1.004517
## a[4] -0.73842566 0.3416373 -1.29574682 -0.20652500 676.2151 1.002353
## a[5] -0.43687694 0.3302756 -0.96653000  0.09257161 666.1891 1.001581
## a[6]  0.48305744 0.3324883 -0.04932631  1.03401130 620.9175 1.004164
## a[7]  1.97049942 0.4159340  1.31369949  2.63979557 913.8363 1.003975
## b[1] -0.05058911 0.2882775 -0.51575335  0.40681499 577.9657 1.004907
## b[2]  0.46814649 0.2895971 -0.01158589  0.91609430 537.6749 1.003561
## b[3] -0.39574390 0.2888728 -0.84371993  0.08032173 562.8976 1.004555
## b[4]  0.36217717 0.2814648 -0.07753713  0.80329855 564.8794 1.005621
```

Can you explain both the differences and the similarities between the approximate and the MCMC distributions?

Similarities: The precis() output shows that the mean and distributions of the various parameters are very close to each other. This makes sense since the model structure is basically the same through out.

Differences: The MCMC distribution appears to do a slightly better job of approximating the posterior, out-of-sample, compared to the approximate. I am not really quite sure why though...

2. Use WAIC to compare the chimpanzee model that includes a unique intercept for each actor, to the simpler models fit in the same section.

```
m11.1 <- quap(
alist(
pulled_left ~ dbinom( 1 , p ) ,
logit(p) <- a ,
a ~ dnorm( 0 , 1.5 )
) , data=d )
```

```
m11.2 <- quap(
alist(
pulled_left ~ dbinom( 1 , p ) ,
logit(p) <- a + b[treatment] ,
a ~ dnorm( 0 , 1.5 ),
b[treatment] ~ dnorm( 0 , .5 )
) , data=d )
```

```
m11.3 <- quap(
alist(
pulled_left ~ dbinom( 1 , p ) ,
logit(p) <- a + b[treatment] ,
a ~ dnorm( 0 , 1.5 ),
b[treatment] ~ dnorm( 0 , 0.5 )
) , data=d )
```

```
compare(m11.1, m11.2, m11.3, m11.4)
```

```
## Warning in compare(m11.1, m11.2, m11.3, m11.4): Not all model fits of same class.
## This is usually a bad idea, because it implies they were fit by different algorithms.
## Check yourself, before you wreck yourself.
```

```
##           WAIC        SE    dWAIC      dSE    pWAIC       weight
## m11.4 531.4295 18.894862   0.0000       NA 8.119138 1.000000e+00
```

```
## m11.2 682.5153  9.128488 151.0858 18.37805 3.649611 1.556465e-33
## m11.3 682.5361  9.166202 151.1066 18.37839 3.664398 1.540333e-33
## m11.1 687.9419  7.129697 156.5124 18.92593 1.000923 1.032176e-34
```

3. Bald eagle salmon pirating. While one eagle feeds, sometimes another will swoop in and try to steal the salmon from it. Call the feeding eagle the 'victim' and the thief the 'pirate.' Use the available data to build a binomial GLM of successful pirating attempts.

Solutions here: http://xcelab.net/rm/wp-content/uploads/2012/01/week07-ch8-solutions.pdf

```
library(MASS)
data(eagles)
d <- eagles
# y = # successful attempts
# n = total attempts
# P = size of pirating eagle (L = large, S = small)
# A = age of pirating eagle (I = immature, A = adult)
# V = size of victime eagle (L = large, S = small)
d$big_pirate = ifelse(d$P == "L", 1, 0)
d$big_victim = ifelse(d$V == "L", 1, 0)
d$adult_pirate = ifelse(d$A == "A", 1, 0)
head(d, 3)
```

```
##    y  n P A V big_pirate big_victim adult_pirate
## 1 17 24 L A L          1          1            1
## 2 29 29 L A S          1          0            1
## 3 17 27 L I L          1          1            0
```

a. Fit the given model to the eagles data, using both quap and ulam. Is the quadratic approximation okay?

To me it seems like the quadratic approximation gets pretty stuck. Bad start values it seems.

```
# fit the quap model first
#flist <- alist(y ~ dbinom(n, pr),
 #              logit(pr) <- a + bp*P + bv*V + ba*A,
  #             a ~ dnorm(0, 1.5),
   #            bp ~ dnorm(0, .5),
    #          ba ~ dnorm(0, .5))

#quap_model = quap(flist, d)
```

```
# try again with ulam
dat_list <- list(
y = d$y,
n = d$n,
big_pirate = d$big_pirate,
adult_pirate = d$adult_pirate,
big_victim = d$big_victim)

mcmc_model <- ulam(
  alist(
    y ~ dbinom(n, pr),
    logit(pr) <- a + bp*big_pirate + bv*big_victim + ba*adult_pirate,
    a ~ dnorm(0, 1.5),
    bp ~ dnorm(0, .5),
    bv ~ dnorm(0, .5),
```

```
    ba ~ dnorm(0, .5)),
    data = dat_list, chains = 4, log_lik = TRUE)
```

```
## Trying to compile a simple C file
## Running /Library/Frameworks/R.framework/Resources/bin/R CMD SHLIB foo.c
## clang -I"/Library/Frameworks/R.framework/Resources/include" -DNDEBUG   -I"/Library/Frameworks/R.frame
## In file included from <built-in>:1:
## In file included from /Library/Frameworks/R.framework/Versions/3.6/Resources/library/StanHeaders/incl
## In file included from /Library/Frameworks/R.framework/Versions/3.6/Resources/library/RcppEigen/includ
## In file included from /Library/Frameworks/R.framework/Versions/3.6/Resources/library/RcppEigen/includ
## /Library/Frameworks/R.framework/Versions/3.6/Resources/library/RcppEigen/include/Eigen/src/Core/util/
## namespace Eigen {
## ^
## /Library/Frameworks/R.framework/Versions/3.6/Resources/library/RcppEigen/include/Eigen/src/Core/util/
## namespace Eigen {
##                ^
##                ;
## In file included from <built-in>:1:
## In file included from /Library/Frameworks/R.framework/Versions/3.6/Resources/library/StanHeaders/incl
## In file included from /Library/Frameworks/R.framework/Versions/3.6/Resources/library/RcppEigen/includ
## /Library/Frameworks/R.framework/Versions/3.6/Resources/library/RcppEigen/include/Eigen/Core:96:10: fa
## #include <complex>
##          ^~~~~~~~~
## 3 errors generated.
## make: *** [foo.o] Error 1
##
## SAMPLING FOR MODEL 'c42ab10bad9cab5c4be9df23c7c9d028' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 1.8e-05 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.18 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:   1 / 1000 [  0%]  (Warmup)
## Chain 1: Iteration: 100 / 1000 [ 10%]  (Warmup)
## Chain 1: Iteration: 200 / 1000 [ 20%]  (Warmup)
## Chain 1: Iteration: 300 / 1000 [ 30%]  (Warmup)
## Chain 1: Iteration: 400 / 1000 [ 40%]  (Warmup)
## Chain 1: Iteration: 500 / 1000 [ 50%]  (Warmup)
## Chain 1: Iteration: 501 / 1000 [ 50%]  (Sampling)
## Chain 1: Iteration: 600 / 1000 [ 60%]  (Sampling)
## Chain 1: Iteration: 700 / 1000 [ 70%]  (Sampling)
## Chain 1: Iteration: 800 / 1000 [ 80%]  (Sampling)
## Chain 1: Iteration: 900 / 1000 [ 90%]  (Sampling)
## Chain 1: Iteration: 1000 / 1000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 0.030619 seconds (Warm-up)
## Chain 1:                0.030367 seconds (Sampling)
## Chain 1:                0.060986 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL 'c42ab10bad9cab5c4be9df23c7c9d028' NOW (CHAIN 2).
## Chain 2:
```

```
## Chain 2: Gradient evaluation took 7e-06 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0.07 seconds.
## Chain 2: Adjust your expectations accordingly!
## Chain 2:
## Chain 2:
## Chain 2: Iteration:   1 / 1000 [  0%]  (Warmup)
## Chain 2: Iteration: 100 / 1000 [ 10%]  (Warmup)
## Chain 2: Iteration: 200 / 1000 [ 20%]  (Warmup)
## Chain 2: Iteration: 300 / 1000 [ 30%]  (Warmup)
## Chain 2: Iteration: 400 / 1000 [ 40%]  (Warmup)
## Chain 2: Iteration: 500 / 1000 [ 50%]  (Warmup)
## Chain 2: Iteration: 501 / 1000 [ 50%]  (Sampling)
## Chain 2: Iteration: 600 / 1000 [ 60%]  (Sampling)
## Chain 2: Iteration: 700 / 1000 [ 70%]  (Sampling)
## Chain 2: Iteration: 800 / 1000 [ 80%]  (Sampling)
## Chain 2: Iteration: 900 / 1000 [ 90%]  (Sampling)
## Chain 2: Iteration: 1000 / 1000 [100%]  (Sampling)
## Chain 2:
## Chain 2:  Elapsed Time: 0.025729 seconds (Warm-up)
## Chain 2:                0.028048 seconds (Sampling)
## Chain 2:                0.053777 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL 'c42ab10bad9cab5c4be9df23c7c9d028' NOW (CHAIN 3).
## Chain 3:
## Chain 3: Gradient evaluation took 8e-06 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0.08 seconds.
## Chain 3: Adjust your expectations accordingly!
## Chain 3:
## Chain 3:
## Chain 3: Iteration:   1 / 1000 [  0%]  (Warmup)
## Chain 3: Iteration: 100 / 1000 [ 10%]  (Warmup)
## Chain 3: Iteration: 200 / 1000 [ 20%]  (Warmup)
## Chain 3: Iteration: 300 / 1000 [ 30%]  (Warmup)
## Chain 3: Iteration: 400 / 1000 [ 40%]  (Warmup)
## Chain 3: Iteration: 500 / 1000 [ 50%]  (Warmup)
## Chain 3: Iteration: 501 / 1000 [ 50%]  (Sampling)
## Chain 3: Iteration: 600 / 1000 [ 60%]  (Sampling)
## Chain 3: Iteration: 700 / 1000 [ 70%]  (Sampling)
## Chain 3: Iteration: 800 / 1000 [ 80%]  (Sampling)
## Chain 3: Iteration: 900 / 1000 [ 90%]  (Sampling)
## Chain 3: Iteration: 1000 / 1000 [100%]  (Sampling)
## Chain 3:
## Chain 3:  Elapsed Time: 0.027386 seconds (Warm-up)
## Chain 3:                0.026361 seconds (Sampling)
## Chain 3:                0.053747 seconds (Total)
## Chain 3:
##
## SAMPLING FOR MODEL 'c42ab10bad9cab5c4be9df23c7c9d028' NOW (CHAIN 4).
## Chain 4:
## Chain 4: Gradient evaluation took 9e-06 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0.09 seconds.
## Chain 4: Adjust your expectations accordingly!
## Chain 4:
```

```
## Chain 4:
## Chain 4: Iteration:   1 / 1000 [  0%]  (Warmup)
## Chain 4: Iteration: 100 / 1000 [ 10%]  (Warmup)
## Chain 4: Iteration: 200 / 1000 [ 20%]  (Warmup)
## Chain 4: Iteration: 300 / 1000 [ 30%]  (Warmup)
## Chain 4: Iteration: 400 / 1000 [ 40%]  (Warmup)
## Chain 4: Iteration: 500 / 1000 [ 50%]  (Warmup)
## Chain 4: Iteration: 501 / 1000 [ 50%]  (Sampling)
## Chain 4: Iteration: 600 / 1000 [ 60%]  (Sampling)
## Chain 4: Iteration: 700 / 1000 [ 70%]  (Sampling)
## Chain 4: Iteration: 800 / 1000 [ 80%]  (Sampling)
## Chain 4: Iteration: 900 / 1000 [ 90%]  (Sampling)
## Chain 4: Iteration: 1000 / 1000 [100%]  (Sampling)
## Chain 4:
## Chain 4:  Elapsed Time: 0.03042 seconds (Warm-up)
## Chain 4:                0.024944 seconds (Sampling)
## Chain 4:                0.055364 seconds (Total)
## Chain 4:
```

b. Now interpret the estimates.

```
precis(mcmc_model)
```

```
##          mean        sd       5.5%      94.5%    n_eff    Rhat4
## a   0.2971145 0.3810319 -0.3133629  0.9065942 1022.554 1.001262
## bp  1.6302904 0.3022682  1.1485337  2.1103483 1175.289 1.001038
## bv -1.6922084 0.3099360 -2.1861298 -1.1995647 1270.661 1.000287
## ba  0.6710458 0.3072357  0.1911051  1.1692760 1273.005 1.001137
```

The intercept log-odds, a, indicates the log-odds probability of a successful attempt for a *small and immature pirate* attacking a small victim. To convert this log-odds back to the "outcome scale" – ie the non-log-odds scale, we need to apply the *inverse logit*, otherwise known as the *logistic* function.

```
# probability of small immature pirate successfully stealing from a small victim
exp(.32) / (1 + exp(.32))
```

```
## [1] 0.5793243
```

58% of attempts by immature small pirates on small victims are expected to succeed.

```
# probability of a big immature pirate successfully stealing from a small victim
exp(.32 + 1.62)/(1 + exp(.32 + 1.62))
```

```
## [1] 0.8743521
```

87% of attempts by large immature pirates on small victims are expected to succeed.

```
# probability of a big adult pirate successfully stealing from a small victim
exp(.32 + 1.62 + .66)/(1 + exp(.32 + 1.62 + .66))
```

```
## [1] 0.9308616
```

93% of attemps by large adult pirates on small victims are expected to succeed.

```
# probability of a big adult pirate successfully stealing from a big victim
exp(.32 + 1.62 + .66 - 1.7)/(1 + exp(.32 + 1.62 + .66 - 1.7))
```

```
## [1] 0.7109495
```

71% of attempts by large adult pirates on large victims are expected to succeed.

So the odds of success go down when the victim gets bigger, and they go up when the pirate gets bigger.

We can also interpret the results another way by looking at the *proportional odds*. We can see this by exponentiating each beta estiamte.

```
exp(coef(mcmc_model))
```

```
##         a        bp        bv        ba
## 1.3459694 5.1053569 0.1841125 1.9562821
```

So this says that the odds (p/(1-p)) of a successful attempt are multiplied by about 5 when a pirate becomes large. The odds of success are multiplied by .18 when the victim becomes large. When the pirate and victim become large, the proportional change in odds is 5.07 x .18 = .91 (the net effect is that it gets harder to make the steal). Finally, the effect of being an adult is to roughly double the odds, multiplying them by 1.92. These are all *relative* risk measures. For the absolute measures, it'll be nice to plot them.

3b. Plot the posterior predictions

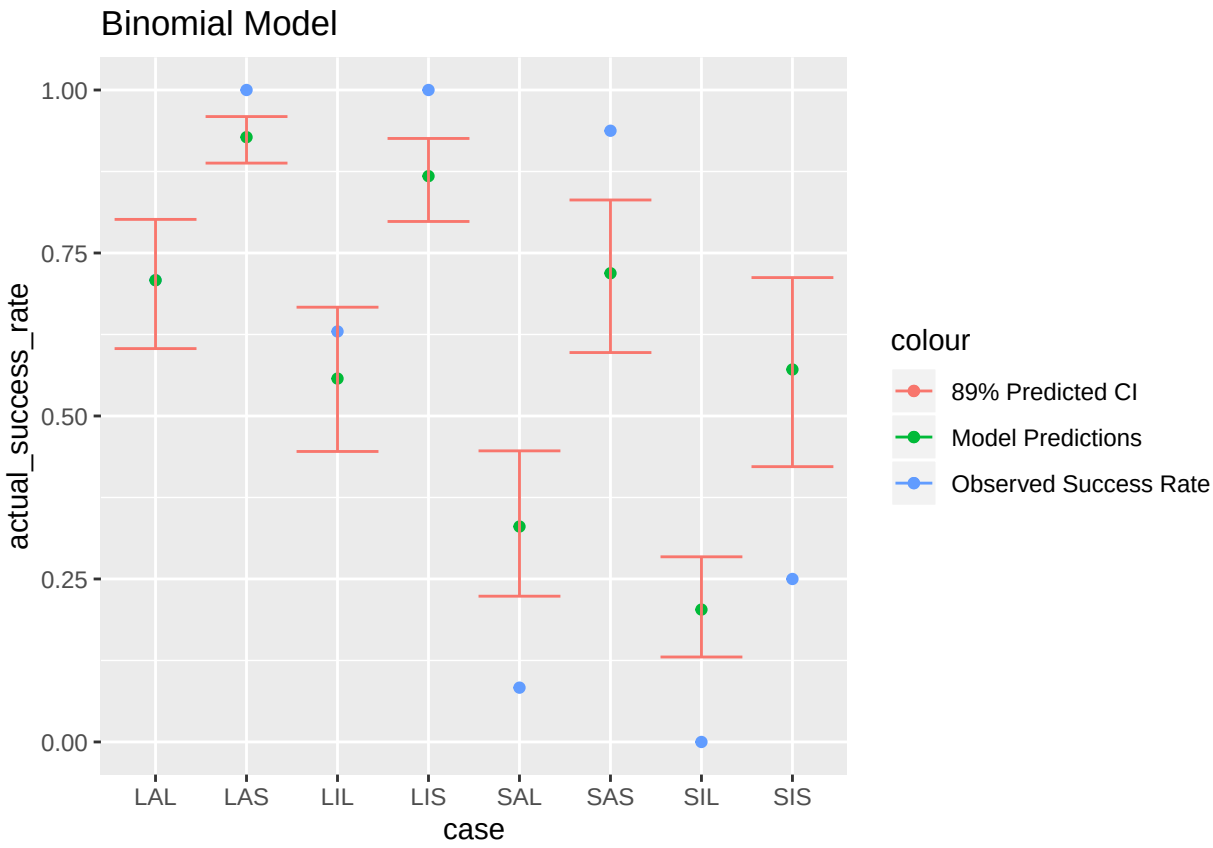
Compute and display the predicted probability of success and its 89% interval for each row in the data

```
# matrix of predictions for each row of data
y.preds = link(mcmc_model, data = d)
y.mean = apply(y.preds, 2, mean)
y.ci = apply(y.preds, 2, PI, prob = .89)
```

```
d$success_rate = d$y / d$n
preds = data.frame(case = c('LAL','LAS','LIL','LIS','SAL','SAS','SIL','SIS'),
                   actual_success_rate = d$success,
                   predicted = y.mean,
                   lower_ci = y.ci[1,],
                   upper_ci = y.ci[2,])
head(preds, 3)
```

```
##   case actual_success_rate predicted  lower_ci  upper_ci
## 1  LAL           0.7083333 0.7082224 0.6033775 0.8015530
## 2  LAS           1.0000000 0.9276364 0.8878669 0.9592693
## 3  LIL           0.6296296 0.5573842 0.4455466 0.6668553
```

```
ggplot(data = preds, aes(x = case, y = actual_success_rate, color = 'Observed Success Rate'))+
  geom_point()+
  geom_point(aes(x = case, y = y.mean, color = 'Model Predictions'))+
  geom_errorbar(aes(ymin = lower_ci, ymax = upper_ci, color = '89% Predicted CI'))+
  ggtitle('Binomial Model')
```

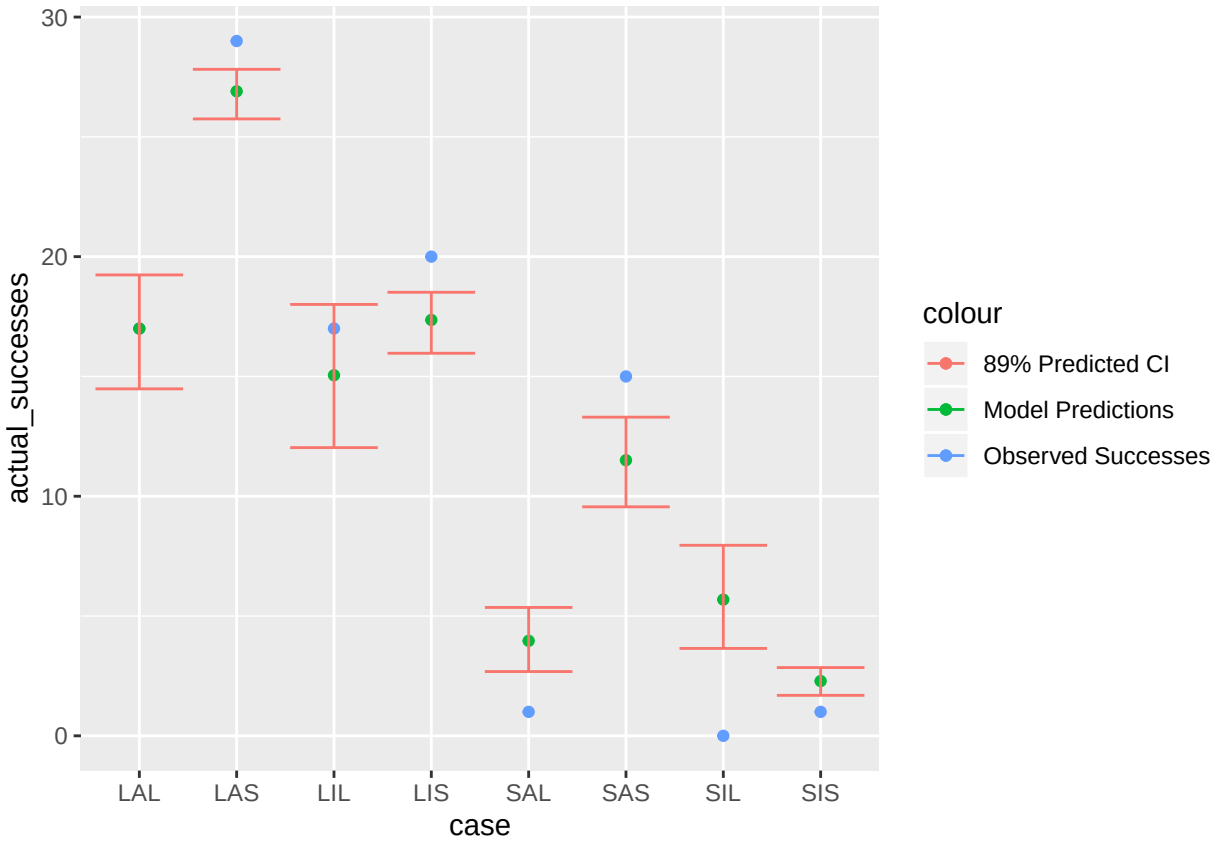


In only 2 out of 8 cases is the observation within the model's 89% CI. This is definitely a disappointing result.

Compute and display the predicted success count and its 89% interval.

```
preds$trials = d$n
preds$actual_successes = d$y
preds$predicted_success_count = preds$trials * preds$predicted
preds$success_lower = preds$trials * preds$lower_ci
preds$success_upper = preds$trials * preds$upper_ci
```

```
ggplot(data = preds, aes(x = case, y = actual_successes, color = 'Observed Successes'))+
  geom_point()+
  geom_point(aes(x = case, y = predicted_success_count, color = 'Model Predictions'))+
  geom_errorbar(aes(ymin = success_lower, ymax = success_upper, color = '89% Predicted CI'))
```



What different information does each type of posterior prediction provide?

The top plot is more useful because it makes the probabilities comparable. We can quickly see that cases when the attacker is larger than the victim are predicted to be more successful. The count plot gives us a quick way to double check sample size for each of the predictions. For example, we can be most "confident" in our predictions for the LAS case, since it occurred the most times, and least "confident" in the reliability of our predictions for the SIS cases, since it occurred the least times.

3c. Now try to improve the model. Consider an interaction between the pirate's size and age.

prior for a: average success rate should always be positive...It should be somewhere between 0 and 1 so might make sense to center it at 50%.... a = 0 on log-odds scale corresponds to 50% on outcome scale. We can increase the sigma on a to 3 to give the outcome a range from 5% to 95% as plausible values prior for bp: this should improve success rate, so also should be postive prior for ba: should be positive prior for bv: should be negative (big victims decrease success rate) prior for bpa: not really sure, give it a range centered at 0?

```
interaction_mod <- ulam(
  alist(
    y ~ dbinom(n, pr),
    logit(pr) <- a + bp*big_pirate + bv*big_victim + ba*adult_pirate + bpa * adult_pirate * big_pirate,
    a ~ dnorm(0, 3),
    bp ~ dnorm(3, 1),
    bv ~ dnorm(-3, 1),
    ba ~ dnorm(3, 1),
    bpa ~ dnorm(0, 1.5)),
    data = dat_list, chains = 4, log_lik = TRUE)
```

```
## Trying to compile a simple C file
## Running /Library/Frameworks/R.framework/Resources/bin/R CMD SHLIB foo.c
## clang -I"/Library/Frameworks/R.framework/Resources/include" -DNDEBUG   -I"/Library/Frameworks/R.frame
## In file included from <built-in>:1:
## In file included from /Library/Frameworks/R.framework/Versions/3.6/Resources/library/StanHeaders/incl
## In file included from /Library/Frameworks/R.framework/Versions/3.6/Resources/library/RcppEigen/inclu
## In file included from /Library/Frameworks/R.framework/Versions/3.6/Resources/library/RcppEigen/inclu
## /Library/Frameworks/R.framework/Versions/3.6/Resources/library/RcppEigen/include/Eigen/src/Core/util/
## namespace Eigen {
## ^
## /Library/Frameworks/R.framework/Versions/3.6/Resources/library/RcppEigen/include/Eigen/src/Core/util/
## namespace Eigen {
##                ^
##                ;
## In file included from <built-in>:1:
## In file included from /Library/Frameworks/R.framework/Versions/3.6/Resources/library/StanHeaders/incl
## In file included from /Library/Frameworks/R.framework/Versions/3.6/Resources/library/RcppEigen/inclu
## /Library/Frameworks/R.framework/Versions/3.6/Resources/library/RcppEigen/include/Eigen/Core:96:10: fa
## #include <complex>
##          ^~~~~~~~
## 3 errors generated.
## make: *** [foo.o] Error 1
##
## SAMPLING FOR MODEL '73cda34b6dc79f7a8bd5f3acaf59f157' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 3e-05 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.3 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:   1 / 1000 [  0%]  (Warmup)
## Chain 1: Iteration: 100 / 1000 [ 10%]  (Warmup)
## Chain 1: Iteration: 200 / 1000 [ 20%]  (Warmup)
## Chain 1: Iteration: 300 / 1000 [ 30%]  (Warmup)
## Chain 1: Iteration: 400 / 1000 [ 40%]  (Warmup)
## Chain 1: Iteration: 500 / 1000 [ 50%]  (Warmup)
## Chain 1: Iteration: 501 / 1000 [ 50%]  (Sampling)
## Chain 1: Iteration: 600 / 1000 [ 60%]  (Sampling)
## Chain 1: Iteration: 700 / 1000 [ 70%]  (Sampling)
## Chain 1: Iteration: 800 / 1000 [ 80%]  (Sampling)
## Chain 1: Iteration: 900 / 1000 [ 90%]  (Sampling)
## Chain 1: Iteration: 1000 / 1000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 0.061903 seconds (Warm-up)
## Chain 1:                0.049382 seconds (Sampling)
## Chain 1:                0.111285 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL '73cda34b6dc79f7a8bd5f3acaf59f157' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 1.1e-05 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0.11 seconds.
## Chain 2: Adjust your expectations accordingly!
```

```
## Chain 2:
## Chain 2:
## Chain 2: Iteration:   1 / 1000 [  0%]  (Warmup)
## Chain 2: Iteration: 100 / 1000 [ 10%]  (Warmup)
## Chain 2: Iteration: 200 / 1000 [ 20%]  (Warmup)
## Chain 2: Iteration: 300 / 1000 [ 30%]  (Warmup)
## Chain 2: Iteration: 400 / 1000 [ 40%]  (Warmup)
## Chain 2: Iteration: 500 / 1000 [ 50%]  (Warmup)
## Chain 2: Iteration: 501 / 1000 [ 50%]  (Sampling)
## Chain 2: Iteration: 600 / 1000 [ 60%]  (Sampling)
## Chain 2: Iteration: 700 / 1000 [ 70%]  (Sampling)
## Chain 2: Iteration: 800 / 1000 [ 80%]  (Sampling)
## Chain 2: Iteration: 900 / 1000 [ 90%]  (Sampling)
## Chain 2: Iteration: 1000 / 1000 [100%]  (Sampling)
## Chain 2:
## Chain 2:  Elapsed Time: 0.0563 seconds (Warm-up)
## Chain 2:                0.054973 seconds (Sampling)
## Chain 2:                0.111273 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL '73cda34b6dc79f7a8bd5f3acaf59f157' NOW (CHAIN 3).
## Chain 3:
## Chain 3: Gradient evaluation took 9e-06 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0.09 seconds.
## Chain 3: Adjust your expectations accordingly!
## Chain 3:
## Chain 3:
## Chain 3: Iteration:   1 / 1000 [  0%]  (Warmup)
## Chain 3: Iteration: 100 / 1000 [ 10%]  (Warmup)
## Chain 3: Iteration: 200 / 1000 [ 20%]  (Warmup)
## Chain 3: Iteration: 300 / 1000 [ 30%]  (Warmup)
## Chain 3: Iteration: 400 / 1000 [ 40%]  (Warmup)
## Chain 3: Iteration: 500 / 1000 [ 50%]  (Warmup)
## Chain 3: Iteration: 501 / 1000 [ 50%]  (Sampling)
## Chain 3: Iteration: 600 / 1000 [ 60%]  (Sampling)
## Chain 3: Iteration: 700 / 1000 [ 70%]  (Sampling)
## Chain 3: Iteration: 800 / 1000 [ 80%]  (Sampling)
## Chain 3: Iteration: 900 / 1000 [ 90%]  (Sampling)
## Chain 3: Iteration: 1000 / 1000 [100%]  (Sampling)
## Chain 3:
## Chain 3:  Elapsed Time: 0.059975 seconds (Warm-up)
## Chain 3:                0.04228 seconds (Sampling)
## Chain 3:                0.102255 seconds (Total)
## Chain 3:
##
## SAMPLING FOR MODEL '73cda34b6dc79f7a8bd5f3acaf59f157' NOW (CHAIN 4).
## Chain 4:
## Chain 4: Gradient evaluation took 1.2e-05 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0.12 seconds.
## Chain 4: Adjust your expectations accordingly!
## Chain 4:
## Chain 4:
## Chain 4: Iteration:   1 / 1000 [  0%]  (Warmup)
## Chain 4: Iteration: 100 / 1000 [ 10%]  (Warmup)
```

```
## Chain 4: Iteration: 200 / 1000 [ 20%]  (Warmup)
## Chain 4: Iteration: 300 / 1000 [ 30%]  (Warmup)
## Chain 4: Iteration: 400 / 1000 [ 40%]  (Warmup)
## Chain 4: Iteration: 500 / 1000 [ 50%]  (Warmup)
## Chain 4: Iteration: 501 / 1000 [ 50%]  (Sampling)
## Chain 4: Iteration: 600 / 1000 [ 60%]  (Sampling)
## Chain 4: Iteration: 700 / 1000 [ 70%]  (Sampling)
## Chain 4: Iteration: 800 / 1000 [ 80%]  (Sampling)
## Chain 4: Iteration: 900 / 1000 [ 90%]  (Sampling)
## Chain 4: Iteration: 1000 / 1000 [100%]  (Sampling)
## Chain 4:
## Chain 4:  Elapsed Time: 0.051866 seconds (Warm-up)
## Chain 4:                0.047892 seconds (Sampling)
## Chain 4:                0.099758 seconds (Total)
## Chain 4:
```

Compare this model to the previous one using WAIC

```
compare(mcmc_model, interaction_mod, n = sum(d$n))
```

```
##                      WAIC        SE    dWAIC      dSE    pWAIC       weight
## interaction_mod 24.22338  5.370717  0.00000       NA 2.740529 1.000000e+00
## mcmc_model      58.30593 11.049718 34.08255 14.90241 7.883505 3.972539e-08
```

This looks like either strong support for the interaction, or possibly just for the better priors...

```
precis(interaction_mod)
```

```
##           mean        sd      5.5%      94.5%    n_eff    Rhat4
## a   -0.2221164 0.6291739 -1.245650  0.8029585 835.9413 1.005689
## bp   4.3628041 0.6242105  3.383901  5.3488816 960.5949 1.005116
## bv  -3.8442430 0.5969734 -4.805374 -2.9143677 967.8236 1.001713
## ba   2.3174809 0.6144979  1.334283  3.2884533 772.8814 1.009579
## bpa -1.4784460 0.7236705 -2.605979 -0.2744948 699.7265 1.004578
```

log-odds for a is -.19 which corresponds to about 45% probability on outcome scale. This seems fairly reasonable! The 89% CI ranges from 23% to 69% as "average success rate." Also seems reasonable.

Interpret.

```
# matrix of predictions for each row of data
y.preds = link(interaction_mod, data = d)
y.mean = apply(y.preds, 2, mean)
y.ci = apply(y.preds, 2, PI, prob = .89)
```

```
preds = data.frame(case = c('LAL','LAS','LIL','LIS','SAL','SAS','SIL','SIS'),
                   actual_success_rate = d$success,
                   predicted = y.mean,
                   lower_ci = y.ci[1,],
                   upper_ci = y.ci[2,])
```

```
ggplot(data = preds, aes(x = case, y = actual_success_rate, color = 'Observed Success Rate'))+
  geom_point()+
  geom_point(aes(x = case, y = y.mean, color = 'Model Predictions'))+
  geom_errorbar(aes(ymin = lower_ci, ymax = upper_ci, color = '89% Predicted CI'))+
  ggtitle('Visualizing Interaction Model Predictions')
```

This looks much better! All of the situations are now contained in the 89% CI, and in most cases the mean value is pretty close to the actual observation.

```
no_int_mod <- ulam(
  alist(
    y ~ dbinom(n, pr),
    logit(pr) <- a + bp*big_pirate + bv*big_victim + ba*adult_pirate,
    a ~ dnorm(0, 3),
    bp ~ dnorm(3, 1),
    bv ~ dnorm(-3, 1),
    ba ~ dnorm(3, 1)),
    data = dat_list, chains = 4, log_lik = TRUE)
```

```
## Trying to compile a simple C file
```

```
## Running /Library/Frameworks/R.framework/Resources/bin/R CMD SHLIB foo.c
## clang -I"/Library/Frameworks/R.framework/Resources/include" -DNDEBUG   -I"/Library/Frameworks/R.frame
## In file included from <built-in>:1:
## In file included from /Library/Frameworks/R.framework/Versions/3.6/Resources/library/StanHeaders/incl
## In file included from /Library/Frameworks/R.framework/Versions/3.6/Resources/library/RcppEigen/inclu
## In file included from /Library/Frameworks/R.framework/Versions/3.6/Resources/library/RcppEigen/inclu
## /Library/Frameworks/R.framework/Versions/3.6/Resources/library/RcppEigen/include/Eigen/src/Core/util/
## namespace Eigen {
## ^
## /Library/Frameworks/R.framework/Versions/3.6/Resources/library/RcppEigen/include/Eigen/src/Core/util/
## namespace Eigen {
##                 ^
##                 ;
```

```
## In file included from <built-in>:1:
## In file included from /Library/Frameworks/R.framework/Versions/3.6/Resources/library/StanHeaders/incl
## In file included from /Library/Frameworks/R.framework/Versions/3.6/Resources/library/RcppEigen/inclu
## /Library/Frameworks/R.framework/Versions/3.6/Resources/library/RcppEigen/include/Eigen/Core:96:10: fa
## #include <complex>
##          ^~~~~~~~
## 3 errors generated.
## make: *** [foo.o] Error 1
##
## SAMPLING FOR MODEL 'd9f9e231f0fe88d4c565cde65b7458ea' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 1.8e-05 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.18 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:   1 / 1000 [  0%]  (Warmup)
## Chain 1: Iteration: 100 / 1000 [ 10%]  (Warmup)
## Chain 1: Iteration: 200 / 1000 [ 20%]  (Warmup)
## Chain 1: Iteration: 300 / 1000 [ 30%]  (Warmup)
## Chain 1: Iteration: 400 / 1000 [ 40%]  (Warmup)
## Chain 1: Iteration: 500 / 1000 [ 50%]  (Warmup)
## Chain 1: Iteration: 501 / 1000 [ 50%]  (Sampling)
## Chain 1: Iteration: 600 / 1000 [ 60%]  (Sampling)
## Chain 1: Iteration: 700 / 1000 [ 70%]  (Sampling)
## Chain 1: Iteration: 800 / 1000 [ 80%]  (Sampling)
## Chain 1: Iteration: 900 / 1000 [ 90%]  (Sampling)
## Chain 1: Iteration: 1000 / 1000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 0.033463 seconds (Warm-up)
## Chain 1:                0.031468 seconds (Sampling)
## Chain 1:                0.064931 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL 'd9f9e231f0fe88d4c565cde65b7458ea' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 6e-06 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0.06 seconds.
## Chain 2: Adjust your expectations accordingly!
## Chain 2:
## Chain 2:
## Chain 2: Iteration:   1 / 1000 [  0%]  (Warmup)
## Chain 2: Iteration: 100 / 1000 [ 10%]  (Warmup)
## Chain 2: Iteration: 200 / 1000 [ 20%]  (Warmup)
## Chain 2: Iteration: 300 / 1000 [ 30%]  (Warmup)
## Chain 2: Iteration: 400 / 1000 [ 40%]  (Warmup)
## Chain 2: Iteration: 500 / 1000 [ 50%]  (Warmup)
## Chain 2: Iteration: 501 / 1000 [ 50%]  (Sampling)
## Chain 2: Iteration: 600 / 1000 [ 60%]  (Sampling)
## Chain 2: Iteration: 700 / 1000 [ 70%]  (Sampling)
## Chain 2: Iteration: 800 / 1000 [ 80%]  (Sampling)
## Chain 2: Iteration: 900 / 1000 [ 90%]  (Sampling)
## Chain 2: Iteration: 1000 / 1000 [100%]  (Sampling)
## Chain 2:
```

```
## Chain 2:  Elapsed Time: 0.039621 seconds (Warm-up)
## Chain 2:                0.034259 seconds (Sampling)
## Chain 2:                0.07388 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL 'd9f9e231f0fe88d4c565cde65b7458ea' NOW (CHAIN 3).
## Chain 3:
## Chain 3: Gradient evaluation took 1e-05 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0.1 seconds.
## Chain 3: Adjust your expectations accordingly!
## Chain 3:
## Chain 3:
## Chain 3: Iteration:   1 / 1000 [  0%]  (Warmup)
## Chain 3: Iteration: 100 / 1000 [ 10%]  (Warmup)
## Chain 3: Iteration: 200 / 1000 [ 20%]  (Warmup)
## Chain 3: Iteration: 300 / 1000 [ 30%]  (Warmup)
## Chain 3: Iteration: 400 / 1000 [ 40%]  (Warmup)
## Chain 3: Iteration: 500 / 1000 [ 50%]  (Warmup)
## Chain 3: Iteration: 501 / 1000 [ 50%]  (Sampling)
## Chain 3: Iteration: 600 / 1000 [ 60%]  (Sampling)
## Chain 3: Iteration: 700 / 1000 [ 70%]  (Sampling)
## Chain 3: Iteration: 800 / 1000 [ 80%]  (Sampling)
## Chain 3: Iteration: 900 / 1000 [ 90%]  (Sampling)
## Chain 3: Iteration: 1000 / 1000 [100%]  (Sampling)
## Chain 3:
## Chain 3:  Elapsed Time: 0.037524 seconds (Warm-up)
## Chain 3:                0.034082 seconds (Sampling)
## Chain 3:                0.071606 seconds (Total)
## Chain 3:
##
## SAMPLING FOR MODEL 'd9f9e231f0fe88d4c565cde65b7458ea' NOW (CHAIN 4).
## Chain 4:
## Chain 4: Gradient evaluation took 6e-06 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0.06 seconds.
## Chain 4: Adjust your expectations accordingly!
## Chain 4:
## Chain 4:
## Chain 4: Iteration:   1 / 1000 [  0%]  (Warmup)
## Chain 4: Iteration: 100 / 1000 [ 10%]  (Warmup)
## Chain 4: Iteration: 200 / 1000 [ 20%]  (Warmup)
## Chain 4: Iteration: 300 / 1000 [ 30%]  (Warmup)
## Chain 4: Iteration: 400 / 1000 [ 40%]  (Warmup)
## Chain 4: Iteration: 500 / 1000 [ 50%]  (Warmup)
## Chain 4: Iteration: 501 / 1000 [ 50%]  (Sampling)
## Chain 4: Iteration: 600 / 1000 [ 60%]  (Sampling)
## Chain 4: Iteration: 700 / 1000 [ 70%]  (Sampling)
## Chain 4: Iteration: 800 / 1000 [ 80%]  (Sampling)
## Chain 4: Iteration: 900 / 1000 [ 90%]  (Sampling)
## Chain 4: Iteration: 1000 / 1000 [100%]  (Sampling)
## Chain 4:
## Chain 4:  Elapsed Time: 0.033351 seconds (Warm-up)
## Chain 4:                0.037124 seconds (Sampling)
## Chain 4:                0.070475 seconds (Total)
## Chain 4:
```

Out of curiosity, I want to see if the interaction model is also considerably better if we re-train the original model using the same priors, but remove the interaction:

```
compare(no_int_mod, interaction_mod, mcmc_model)
```

```
##                      WAIC        SE    dWAIC       dSE    pWAIC       weight
## interaction_mod 24.22338  5.370717  0.00000        NA 2.740529 9.709923e-01
## no_int_mod      31.24490  8.355303  7.02152  3.914503 4.893372 2.900761e-02
## mcmc_model      58.30593 11.049718 34.08255 14.902407 7.883505 3.857305e-08
```

Ah, so the original model has terrible priors. It is significantly improved by the addition of better priors, and we can see that the interaction seems to make a big difference improving it even still. Nice!

4. Salamanders

https://www.samples-of-thoughts.com/projects/statistical-rethinking/chapter_10/chp10-ex/

Counts of salamanders from 47 different plots in CA.

```
data("salamanders")
d <- salamanders
head(d)
```

```
##   SITE SALAMAN PCTCOVER FORESTAGE
## 1    1      13       85       316
## 2    2      11       86        88
## 3    3      11       90       548
## 4    4       9       88        64
## 5    5       8       89        43
## 6    6       7       83       368
```

4a. Model the relationship between density and percent cover

Use a log-link (same as the example in the book and lecture). Use weakly informative priors of your choosing.

Let a be centered at -5 (which is essentially 0 on the outcome scale). My guess is that when cover is at 0, salamanders won't be very happy.

The coefficient on cover should probably be positive.

```
dat <- list(
cover = d$PCTCOVER ,
density = d$SALAMAN
)

msal <- ulam(
alist(
density ~ dpois( lambda ),
log(lambda) <- a + b*cover,
a ~ dnorm(-1,2),
b ~ dnorm(-1, 3)
), data=dat , chains=4 , iter = 4000, log_lik=TRUE )
```

```
## Trying to compile a simple C file
```

```
## Running /Library/Frameworks/R.framework/Resources/bin/R CMD SHLIB foo.c
## clang -I"/Library/Frameworks/R.framework/Resources/include" -DNDEBUG   -I"/Library/Frameworks/R.frame
## In file included from <built-in>:1:
```

```
## In file included from /Library/Frameworks/R.framework/Versions/3.6/Resources/library/StanHeaders/incl
## In file included from /Library/Frameworks/R.framework/Versions/3.6/Resources/library/RcppEigen/inclu
## In file included from /Library/Frameworks/R.framework/Versions/3.6/Resources/library/RcppEigen/inclu
## /Library/Frameworks/R.framework/Versions/3.6/Resources/library/RcppEigen/include/Eigen/src/Core/util/
## namespace Eigen {
## ^
## /Library/Frameworks/R.framework/Versions/3.6/Resources/library/RcppEigen/include/Eigen/src/Core/util/
## namespace Eigen {
##                ^
##                ;
## In file included from <built-in>:1:
## In file included from /Library/Frameworks/R.framework/Versions/3.6/Resources/library/StanHeaders/incl
## In file included from /Library/Frameworks/R.framework/Versions/3.6/Resources/library/RcppEigen/inclu
## /Library/Frameworks/R.framework/Versions/3.6/Resources/library/RcppEigen/include/Eigen/Core:96:10: fa
## #include <complex>
##          ^~~~~~~~
## 3 errors generated.
## make: *** [foo.o] Error 1
##
## SAMPLING FOR MODEL '0b7a0b4a2c4de2485c2c9905faae061b' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 4.1e-05 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.41 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:    1 / 4000 [  0%]  (Warmup)
## Chain 1: Iteration:  400 / 4000 [ 10%]  (Warmup)
## Chain 1: Iteration:  800 / 4000 [ 20%]  (Warmup)
## Chain 1: Iteration: 1200 / 4000 [ 30%]  (Warmup)
## Chain 1: Iteration: 1600 / 4000 [ 40%]  (Warmup)
## Chain 1: Iteration: 2000 / 4000 [ 50%]  (Warmup)
## Chain 1: Iteration: 2001 / 4000 [ 50%]  (Sampling)
## Chain 1: Iteration: 2400 / 4000 [ 60%]  (Sampling)
## Chain 1: Iteration: 2800 / 4000 [ 70%]  (Sampling)
## Chain 1: Iteration: 3200 / 4000 [ 80%]  (Sampling)
## Chain 1: Iteration: 3600 / 4000 [ 90%]  (Sampling)
## Chain 1: Iteration: 4000 / 4000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 0.266029 seconds (Warm-up)
## Chain 1:                0.152861 seconds (Sampling)
## Chain 1:                0.41889 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL '0b7a0b4a2c4de2485c2c9905faae061b' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 9e-06 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0.09 seconds.
## Chain 2: Adjust your expectations accordingly!
## Chain 2:
## Chain 2:
## Chain 2: Iteration:    1 / 4000 [  0%]  (Warmup)
## Chain 2: Iteration:  400 / 4000 [ 10%]  (Warmup)
## Chain 2: Iteration:  800 / 4000 [ 20%]  (Warmup)
```

```
## Chain 2: Iteration: 1200 / 4000 [ 30%]  (Warmup)
## Chain 2: Iteration: 1600 / 4000 [ 40%]  (Warmup)
## Chain 2: Iteration: 2000 / 4000 [ 50%]  (Warmup)
## Chain 2: Iteration: 2001 / 4000 [ 50%]  (Sampling)
## Chain 2: Iteration: 2400 / 4000 [ 60%]  (Sampling)
## Chain 2: Iteration: 2800 / 4000 [ 70%]  (Sampling)
## Chain 2: Iteration: 3200 / 4000 [ 80%]  (Sampling)
## Chain 2: Iteration: 3600 / 4000 [ 90%]  (Sampling)
## Chain 2: Iteration: 4000 / 4000 [100%]  (Sampling)
## Chain 2:
## Chain 2:  Elapsed Time: 0.208812 seconds (Warm-up)
## Chain 2:                0.130454 seconds (Sampling)
## Chain 2:                0.339266 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL '0b7a0b4a2c4de2485c2c9905faae061b' NOW (CHAIN 3).
## Chain 3:
## Chain 3: Gradient evaluation took 9e-06 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0.09 seconds.
## Chain 3: Adjust your expectations accordingly!
## Chain 3:
## Chain 3:
## Chain 3: Iteration:    1 / 4000 [  0%]  (Warmup)
## Chain 3: Iteration:  400 / 4000 [ 10%]  (Warmup)
## Chain 3: Iteration:  800 / 4000 [ 20%]  (Warmup)
## Chain 3: Iteration: 1200 / 4000 [ 30%]  (Warmup)
## Chain 3: Iteration: 1600 / 4000 [ 40%]  (Warmup)
## Chain 3: Iteration: 2000 / 4000 [ 50%]  (Warmup)
## Chain 3: Iteration: 2001 / 4000 [ 50%]  (Sampling)
## Chain 3: Iteration: 2400 / 4000 [ 60%]  (Sampling)
## Chain 3: Iteration: 2800 / 4000 [ 70%]  (Sampling)
## Chain 3: Iteration: 3200 / 4000 [ 80%]  (Sampling)
## Chain 3: Iteration: 3600 / 4000 [ 90%]  (Sampling)
## Chain 3: Iteration: 4000 / 4000 [100%]  (Sampling)
## Chain 3:
## Chain 3:  Elapsed Time: 0.326336 seconds (Warm-up)
## Chain 3:                0.168398 seconds (Sampling)
## Chain 3:                0.494734 seconds (Total)
## Chain 3:
##
## SAMPLING FOR MODEL '0b7a0b4a2c4de2485c2c9905faae061b' NOW (CHAIN 4).
## Chain 4:
## Chain 4: Gradient evaluation took 7e-06 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0.07 seconds.
## Chain 4: Adjust your expectations accordingly!
## Chain 4:
## Chain 4:
## Chain 4: Iteration:    1 / 4000 [  0%]  (Warmup)
## Chain 4: Iteration:  400 / 4000 [ 10%]  (Warmup)
## Chain 4: Iteration:  800 / 4000 [ 20%]  (Warmup)
## Chain 4: Iteration: 1200 / 4000 [ 30%]  (Warmup)
## Chain 4: Iteration: 1600 / 4000 [ 40%]  (Warmup)
## Chain 4: Iteration: 2000 / 4000 [ 50%]  (Warmup)
## Chain 4: Iteration: 2001 / 4000 [ 50%]  (Sampling)
```

```
## Chain 4: Iteration: 2400 / 4000 [ 60%]  (Sampling)
## Chain 4: Iteration: 2800 / 4000 [ 70%]  (Sampling)
## Chain 4: Iteration: 3200 / 4000 [ 80%]  (Sampling)
## Chain 4: Iteration: 3600 / 4000 [ 90%]  (Sampling)
## Chain 4: Iteration: 4000 / 4000 [100%]  (Sampling)
## Chain 4:
## Chain 4:  Elapsed Time: 0.308674 seconds (Warm-up)
## Chain 4:                0.176146 seconds (Sampling)
## Chain 4:                0.48482 seconds (Total)
## Chain 4:
```

```
precis(msal)
```

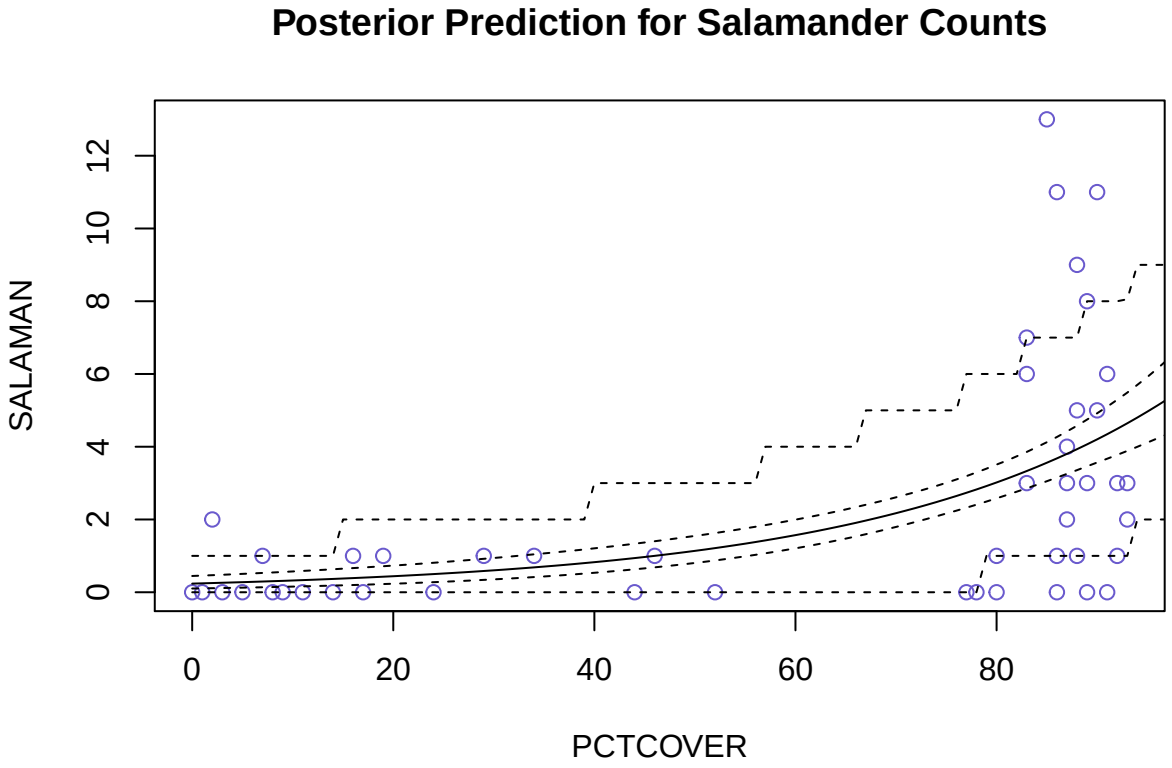
```
##          mean          sd       5.5%       94.5%    n_eff    Rhat4
## a -1.54587774 0.461394475 -2.3621752 -0.85814714 1187.358 1.000978
## b  0.03308208 0.005437621  0.0248549  0.04270703 1197.783 1.000936
```

The estimate for b is positive, which means counts are predicted to increase with cover. But by how much? Hard to say, until you convert back to the count scale. So let's plot the predictions, now.

```
post <- sample.naive.posterior( msal)
x.seq <- 0:100
mu <- sapply( x.seq , function(z)
mean( exp( post$a + post$b*z ) ) )
mu.ci <- sapply( x.seq , function(z)
PCI( exp( post$a + post$b*z ) ) )
y.ci <- sapply( x.seq , function(z)
PCI( rpois( 10000 , exp( post$a + post$b*z ) ) ) )
```

4a. Plot the expected counts and their 89% CI against percent cover. In what ways does the model do a good job? In what ways does the model do a bad job?

```
plot( SALAMAN ~ PCTCOVER , data=d , col='slateblue' , main = 'Posterior Prediction for Salamander Counts
lines( x.seq , mu )
lines( x.seq , mu.ci[1,] , lty=2 )
lines( x.seq , mu.ci[2,] , lty=2 )
lines( x.seq , y.ci[1,] , lty=2 )
lines( x.seq , y.ci[2,] , lty=2 )
```



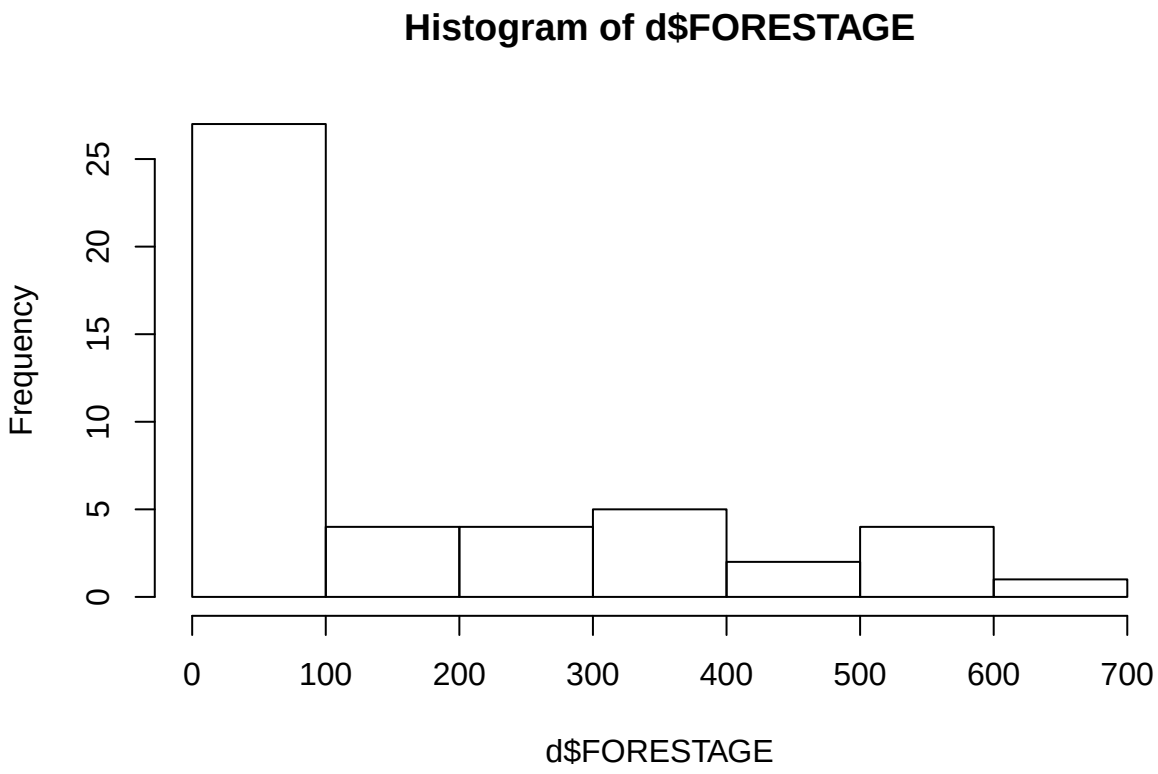
Good job: Model figured out that more cover corresponds to more salamanders Bad job: We can see tha the data seems to have two clusters, cover up to 80% vs cover above 80%. That seems to be the threshold for happy salamanders. The model is trying to capture this continuously but it seems like it might be better captured through some sort of discrete relationship.

4b. Add forestage to the model

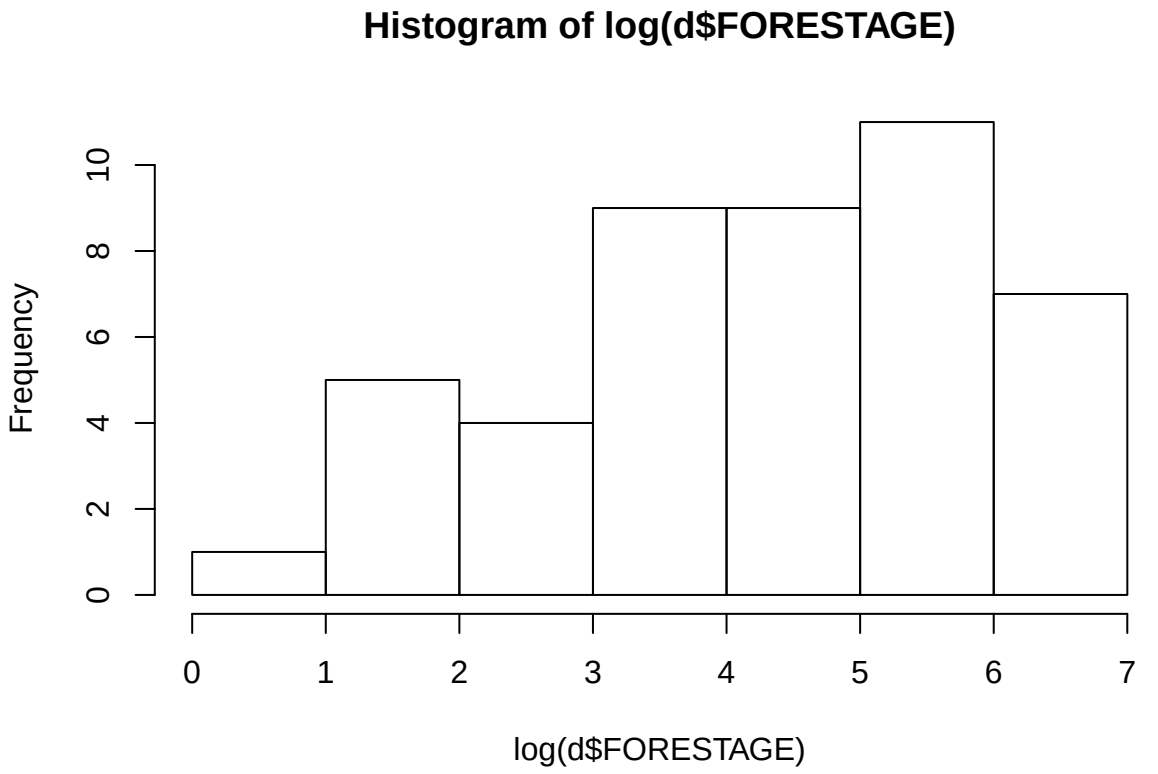
```
cor(d$FORESTAGE, d$PCTCOVER)
```

```
## [1] 0.62546
```

```
hist(d$FORESTAGE)
```




```
hist(log(d$FORESTAGE))
```



How much does adding forest age help, once we already know cover?

```
d$logFORESTAGE <- log(d$FORESTAGE + 1 )
d$logFORESTAGE_c <- d$logFORESTAGE - mean(d$logFORESTAGE)
```

```
dat <- list(
cover = d$PCTCOVER ,
density = d$SALAMAN,
forest = d$logFORESTAGE_c
)

fc <- ulam(
alist(
density ~ dpois( lambda ),
log(lambda) <- a + b*cover + c*forest,
a ~ dnorm(-1.5, .4),
b ~ dnorm(.03, .1),
c ~ dnorm(.03, .1)
), data=dat , chains=4 , iter = 8000, log_lik=TRUE )
```

```
## Trying to compile a simple C file
## Running /Library/Frameworks/R.framework/Resources/bin/R CMD SHLIB foo.c
## clang -I"/Library/Frameworks/R.framework/Resources/include" -DNDEBUG   -I"/Library/Frameworks/R.frame
## In file included from <built-in>:1:
## In file included from /Library/Frameworks/R.framework/Versions/3.6/Resources/library/StanHeaders/incl
## In file included from /Library/Frameworks/R.framework/Versions/3.6/Resources/library/RcppEigen/inclu
## In file included from /Library/Frameworks/R.framework/Versions/3.6/Resources/library/RcppEigen/inclu
## /Library/Frameworks/R.framework/Versions/3.6/Resources/library/RcppEigen/include/Eigen/src/Core/util/
## namespace Eigen {
## ^
## /Library/Frameworks/R.framework/Versions/3.6/Resources/library/RcppEigen/include/Eigen/src/Core/util/
## namespace Eigen {
##                 ^
##                 ;
## In file included from <built-in>:1:
## In file included from /Library/Frameworks/R.framework/Versions/3.6/Resources/library/StanHeaders/incl
## In file included from /Library/Frameworks/R.framework/Versions/3.6/Resources/library/RcppEigen/inclu
## /Library/Frameworks/R.framework/Versions/3.6/Resources/library/RcppEigen/include/Eigen/Core:96:10: fa
## #include <complex>
##          ^~~~~~~~
## 3 errors generated.
## make: *** [foo.o] Error 1
##
## SAMPLING FOR MODEL 'ddec7c18cbb782e9d451fd847876173a' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 2.4e-05 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.24 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:    1 / 8000 [  0%]  (Warmup)
## Chain 1: Iteration:  800 / 8000 [ 10%]  (Warmup)
## Chain 1: Iteration: 1600 / 8000 [ 20%]  (Warmup)
## Chain 1: Iteration: 2400 / 8000 [ 30%]  (Warmup)
## Chain 1: Iteration: 3200 / 8000 [ 40%]  (Warmup)
## Chain 1: Iteration: 4000 / 8000 [ 50%]  (Warmup)
## Chain 1: Iteration: 4001 / 8000 [ 50%]  (Sampling)
```

```
## Chain 1: Iteration: 4800 / 8000 [ 60%]  (Sampling)
## Chain 1: Iteration: 5600 / 8000 [ 70%]  (Sampling)
## Chain 1: Iteration: 6400 / 8000 [ 80%]  (Sampling)
## Chain 1: Iteration: 7200 / 8000 [ 90%]  (Sampling)
## Chain 1: Iteration: 8000 / 8000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 0.554088 seconds (Warm-up)
## Chain 1:                0.431581 seconds (Sampling)
## Chain 1:                0.985669 seconds (Total)
## Chain 1:
##
## SAMPLING FOR MODEL 'ddec7c18cbb782e9d451fd847876173a' NOW (CHAIN 2).
## Chain 2:
## Chain 2: Gradient evaluation took 8e-06 seconds
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0.08 seconds.
## Chain 2: Adjust your expectations accordingly!
## Chain 2:
## Chain 2:
## Chain 2: Iteration:    1 / 8000 [  0%]  (Warmup)
## Chain 2: Iteration:  800 / 8000 [ 10%]  (Warmup)
## Chain 2: Iteration: 1600 / 8000 [ 20%]  (Warmup)
## Chain 2: Iteration: 2400 / 8000 [ 30%]  (Warmup)
## Chain 2: Iteration: 3200 / 8000 [ 40%]  (Warmup)
## Chain 2: Iteration: 4000 / 8000 [ 50%]  (Warmup)
## Chain 2: Iteration: 4001 / 8000 [ 50%]  (Sampling)
## Chain 2: Iteration: 4800 / 8000 [ 60%]  (Sampling)
## Chain 2: Iteration: 5600 / 8000 [ 70%]  (Sampling)
## Chain 2: Iteration: 6400 / 8000 [ 80%]  (Sampling)
## Chain 2: Iteration: 7200 / 8000 [ 90%]  (Sampling)
## Chain 2: Iteration: 8000 / 8000 [100%]  (Sampling)
## Chain 2:
## Chain 2:  Elapsed Time: 0.622405 seconds (Warm-up)
## Chain 2:                0.429004 seconds (Sampling)
## Chain 2:                1.05141 seconds (Total)
## Chain 2:
##
## SAMPLING FOR MODEL 'ddec7c18cbb782e9d451fd847876173a' NOW (CHAIN 3).
## Chain 3:
## Chain 3: Gradient evaluation took 8e-06 seconds
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0.08 seconds.
## Chain 3: Adjust your expectations accordingly!
## Chain 3:
## Chain 3:
## Chain 3: Iteration:    1 / 8000 [  0%]  (Warmup)
## Chain 3: Iteration:  800 / 8000 [ 10%]  (Warmup)
## Chain 3: Iteration: 1600 / 8000 [ 20%]  (Warmup)
## Chain 3: Iteration: 2400 / 8000 [ 30%]  (Warmup)
## Chain 3: Iteration: 3200 / 8000 [ 40%]  (Warmup)
## Chain 3: Iteration: 4000 / 8000 [ 50%]  (Warmup)
## Chain 3: Iteration: 4001 / 8000 [ 50%]  (Sampling)
## Chain 3: Iteration: 4800 / 8000 [ 60%]  (Sampling)
## Chain 3: Iteration: 5600 / 8000 [ 70%]  (Sampling)
## Chain 3: Iteration: 6400 / 8000 [ 80%]  (Sampling)
## Chain 3: Iteration: 7200 / 8000 [ 90%]  (Sampling)
```

```
## Chain 3: Iteration: 8000 / 8000 [100%]  (Sampling)
## Chain 3:
## Chain 3:  Elapsed Time: 0.650123 seconds (Warm-up)
## Chain 3:                0.432158 seconds (Sampling)
## Chain 3:                1.08228 seconds (Total)
## Chain 3:
##
## SAMPLING FOR MODEL 'ddec7c18cbb782e9d451fd847876173a' NOW (CHAIN 4).
## Chain 4:
## Chain 4: Gradient evaluation took 1.1e-05 seconds
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0.11 seconds.
## Chain 4: Adjust your expectations accordingly!
## Chain 4:
## Chain 4:
## Chain 4: Iteration:    1 / 8000 [  0%]  (Warmup)
## Chain 4: Iteration:  800 / 8000 [ 10%]  (Warmup)
## Chain 4: Iteration: 1600 / 8000 [ 20%]  (Warmup)
## Chain 4: Iteration: 2400 / 8000 [ 30%]  (Warmup)
## Chain 4: Iteration: 3200 / 8000 [ 40%]  (Warmup)
## Chain 4: Iteration: 4000 / 8000 [ 50%]  (Warmup)
## Chain 4: Iteration: 4001 / 8000 [ 50%]  (Sampling)
## Chain 4: Iteration: 4800 / 8000 [ 60%]  (Sampling)
## Chain 4: Iteration: 5600 / 8000 [ 70%]  (Sampling)
## Chain 4: Iteration: 6400 / 8000 [ 80%]  (Sampling)
## Chain 4: Iteration: 7200 / 8000 [ 90%]  (Sampling)
## Chain 4: Iteration: 8000 / 8000 [100%]  (Sampling)
## Chain 4:
## Chain 4:  Elapsed Time: 0.591246 seconds (Warm-up)
## Chain 4:                0.464963 seconds (Sampling)
## Chain 4:                1.05621 seconds (Total)
## Chain 4:
```

```
precis(fc)
```

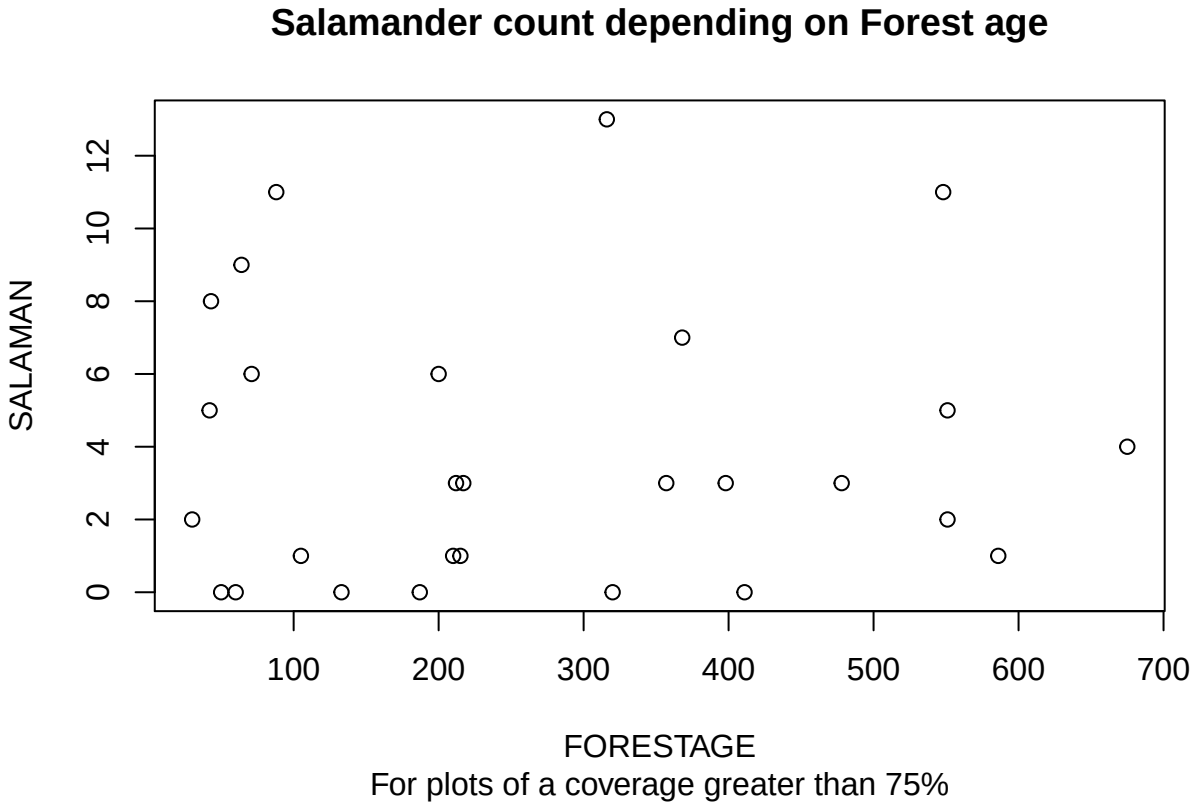
```
##          mean          sd        5.5%       94.5%    n_eff    Rhat4
## a -1.52611488 0.306238840 -2.01791973 -1.04256722 4350.030 1.001509
## b  0.03298363 0.003959863  0.02671656  0.03933417 4391.306 1.001269
## c -0.01254600 0.069456059 -0.12312145  0.09700181 6198.211 1.000013
```

```
compare(fc, msal)
```

```
##          WAIC       SE     dWAIC       dSE    pWAIC    weight
## msal 213.3825 26.44639 0.0000000        NA 4.744302 0.5500819
## fc   213.7845 26.80428 0.4020031 0.9574185 5.321481 0.4499181
```

4b. Can you explain why FORESTAGE helps or does not help with prediction?

```
plot(SALAMAN ~ FORESTAGE, data=d[d$PCTCOVER > 75,],
     main="Salamander count depending on Forest age",
     sub="For plots of a coverage greater than 75%")
```



In summary, the older a forest is, the more likely it has a high ground coverage. If the ground coverage is high, then the age of the forest doesn't add any additional information that help predict the salamder count.