Chapter 11 Questions

Melissa Van Bussel

July 22, 2018

Points for chapter 11: 2 + 4 + 4 = 10 points total.

Easy Questions (2 points total)

11E1

An ordered categorical variable is a categorical variable in which the categories follow some sort of order, either increasing or decreasing. For example, if you were asked to rate how much you liked something on a scale of 1-10, where only discrete values are allowed, this would be an example of a categorical variable. A non-ordered categorical variable is a categorical variable where there is no ordering of the categories. For example, “Male” or “Female” would be considered a non-ordered categorical variable.

11E2

When performing ordered logisitic regression, you would use a cumulative logit link function. The difference is that a cumulative logit link function uses cumulative probability instead of discrete probabilities.

11E3

If you ignore zero-inflation, you will likely end up underestimating the true rate of events, since there will be extra zeroes in the data which will bring down the mean of the data. For example, with the monk example from the chapter, if you didn’t know that the monks could take days off to drink, you might think that they’re really slow and terrible at producing manuscripts, when in reality they just have been taking some days off and they actually work at a much quicker rate than it would initally appear from looking at the data.

11E4

Overdispersion is when there is more variability in a dataset than what we would typically expect (let’s say we have an expected distribution of the data, but then it turns out that the data actually has much higher variance than the distribution you were expecting it to have come from). The following is an example from Wikipedia: we would expect that the probability of having a male vs. a female child is 0.50/0.50. Oddly enough, though, families tend to have uneven ratios of boys and girls. Thus, the resulting variance in the binomial distribution is much larger than what we would *expect*.

Underdispersion on the other hand is when there is less variance than what you would expect. This usually occurs when observations aren’t independent, so as you go along in the dataset, there’s less variance than what you would’ve expected. An example of this would be if you were measuring the number of defects on a piece of equipment over time. Obviously, the number of defects will increase over time and the observed counts will have less variance than what would happen by chance.

Medium Questions

11M1 (2 points)

The log cumulative odds are calculated using the formula

$$\log\left(\frac{p_k}{1-p_k}\right)$$

Thus we have

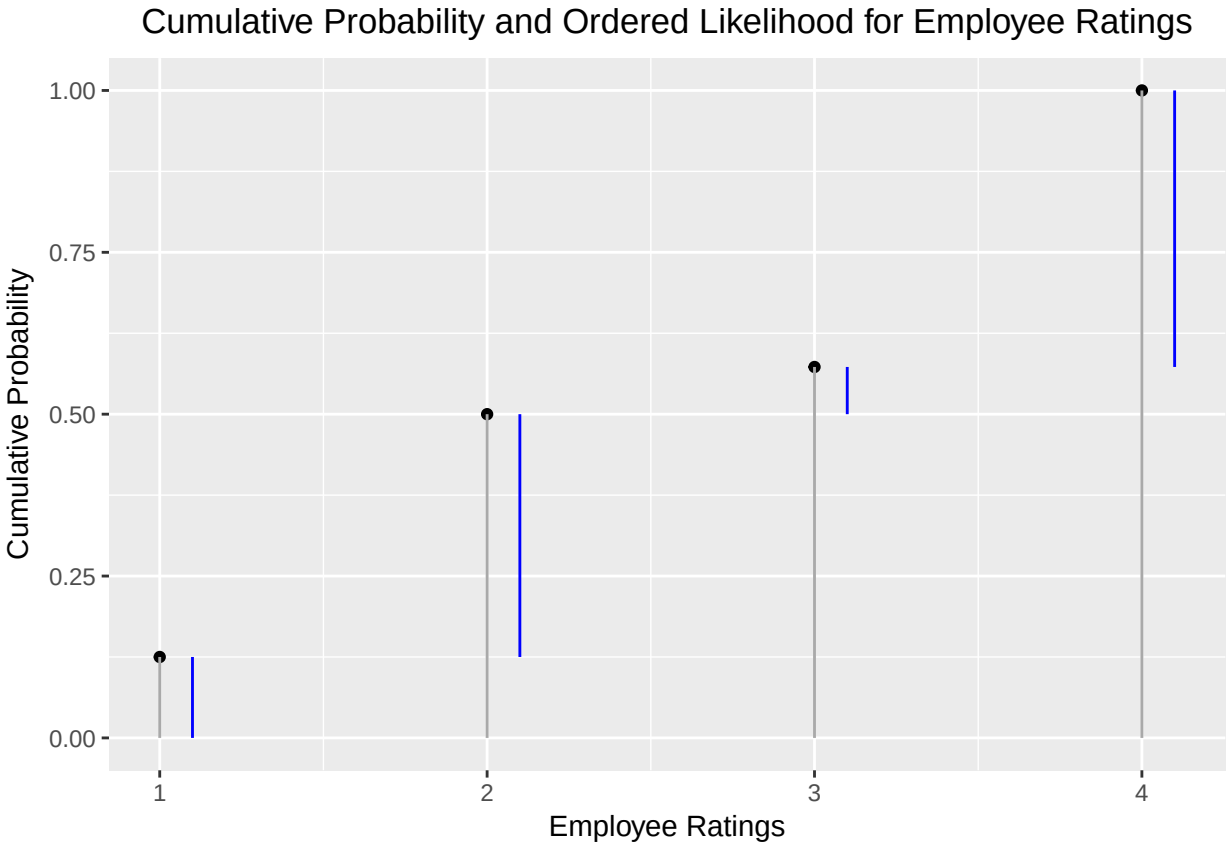
```
ratings <- c(12, 36, 7, 41)
prob_outcome <- ratings / sum(ratings)
p <- cumsum(prob_outcome)
log(p/(1-p))
```

```
## [1] -1.9459101  0.0000000  0.2937611        Inf
```

11M2 (2 points)

I'll do this one using ggplot instead of base R graphics.

```
library(ggplot2)
df <- data.frame("Ratings" = 1:4, "p" = p)
qplot(df$Ratings, df$p) +
  labs(title = "Cumulative Probability and Ordered Likelihood for Employee Ratings",
       x = "Employee Ratings",
       y = "Cumulative Probability") +
  geom_segment(aes(x = 1, y = 0, xend = 1, yend = p[1], col = I("darkgray"))) +
  geom_segment(aes(x = 2, y = 0, xend = 2, yend = p[2], col = I("darkgray"))) +
  geom_segment(aes(x = 3, y = 0, xend = 3, yend = p[3], col = I("darkgray"))) +
  geom_segment(aes(x = 4, y = 0, xend = 4, yend = p[4], col = I("darkgray"))) +
  geom_segment(aes(x = 1.1, y = p[1] - prob_outcome[1],
                   xend = 1.1, yend = p[1], col = I("blue"))) +
  geom_segment(aes(x = 2.1, y = p[2] - prob_outcome[2],
                   xend = 2.1, yend = p[2], col = I("blue"))) +
  geom_segment(aes(x = 3.1, y = p[3] - prob_outcome[3],
                   xend = 3.1, yend = p[3], col = I("blue"))) +
  geom_segment(aes(x = 4.1, y = p[4] - prob_outcome[4],
                    xend = 4.1, yend = p[4], col = I("blue"))) +
  theme(plot.title = element_text(hjust = 0.5))
```



Hard Questions (4 points total)

11H1 (2 points)

```
library(rethinking)
data(Hurricanes)
d <- Hurricanes
d$femininity <- (d$femininity - mean(d$femininity)) / sd(d$femininity)
lmod11h1 <- map2stan(
      alist(
        deaths ~ dpois(lambda),
        log(lambda) <- a + bf * femininity,
        a ~ dnorm(0,10),
        bf ~ dnorm(0,1)
      ), data = list(
                  deaths = d$deaths,
                  femininity = d$femininity),
      chains = 4)
```

Next is the intercept-only model:

```
lmod11h1intercept <- map2stan(
      alist(
        deaths ~ dpois(lambda),
```

```
        log(lambda) <- a,
        a ~ dnorm(0,10)
      ), data = list(
                  deaths = d$deaths,
                  fmnnty = d$femininity),
      chains = 4)
```

Now we can compare the two models:

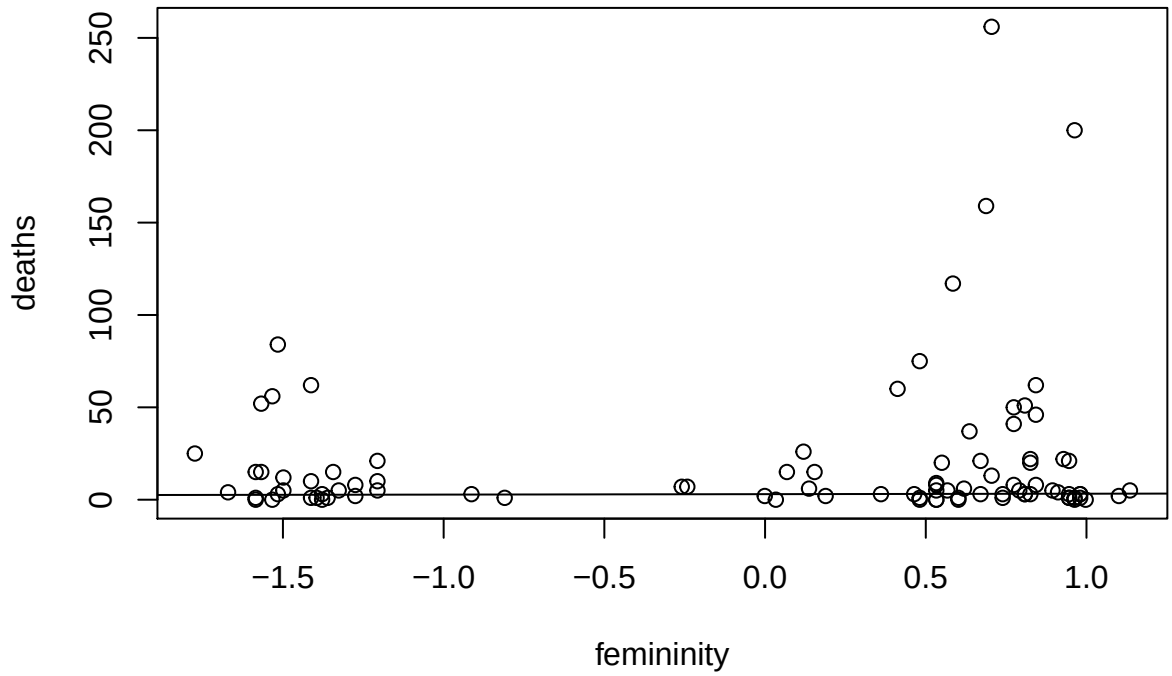
```
compare(lmod11h1, lmod11h1intercept)
```

```
##                      WAIC pWAIC dWAIC weight      SE    dSE
## lmod11h1           4414.8 136.9     0      1  997.10     NA
## lmod11h1intercept 4438.8  79.4    24      0 1070.06 141.54
```

We can see that all of the Akaike weight is given to the model where femininity is included as a predictor variable, so there is evidence that the femininity of a hurricane should be included in our model.

To see which storms are fit well and fit poorly to the model we made, we can plot the raw data.

```
my_seq <- seq(-2, 2, length.out = 100)
plot(d$femininity, d$deaths, xlab = "femininity", ylab = "deaths")
lines(my_seq, coef(lmod11h1)[1] + coef(lmod11h1)[2]*my_seq)
```



The ones that our line did a poor job of predicting are the data points where *deaths* is greater than 100:

```
d[d$deaths > 100,]
```

```
##       name year deaths category min_pressure damage_norm female femininity
## 10   Diane 1955    200        1          987       14730      1  0.9630864
```

```
## 29 Camille 1969    256        5          909       23040      1  0.7048674
## 34   Agnes 1972    117        1          980       20430      1  0.5843644
## 92   Sandy 2012    159        2          942       75000      1  0.6876514
```

11H6 (2 points)

Begin by loading in the Fish data

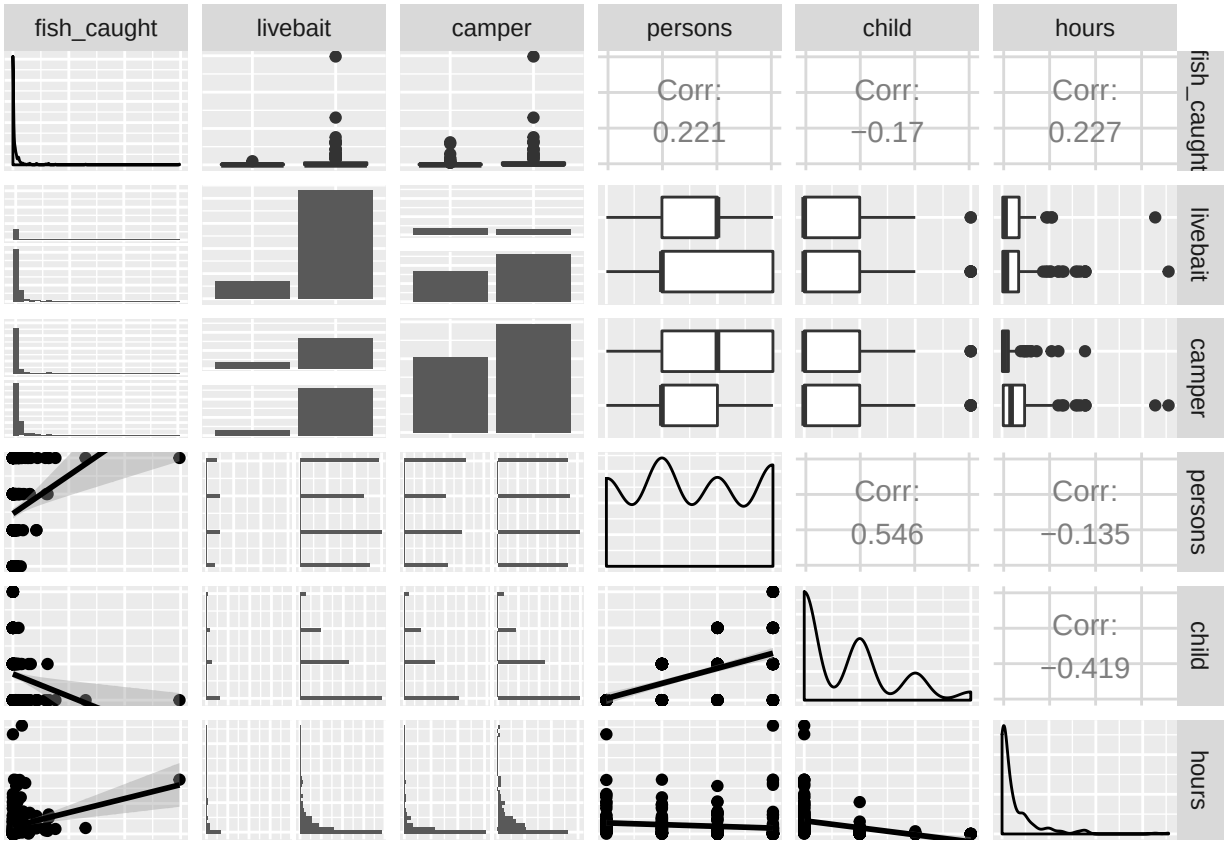
```
library(rethinking)
data(Fish)
f <- Fish
str(f)
```

The variables in this dataset are as follows:

- **fish_caught:** Number of fish caught during visit
- **livebait:** Whether or not group used livebait to fish
- **camper:** Whether or not group had a camper
- **persons:** Number of adults in group
- **child:** Number of children in group
- **hours:** Number of hours group spent in park

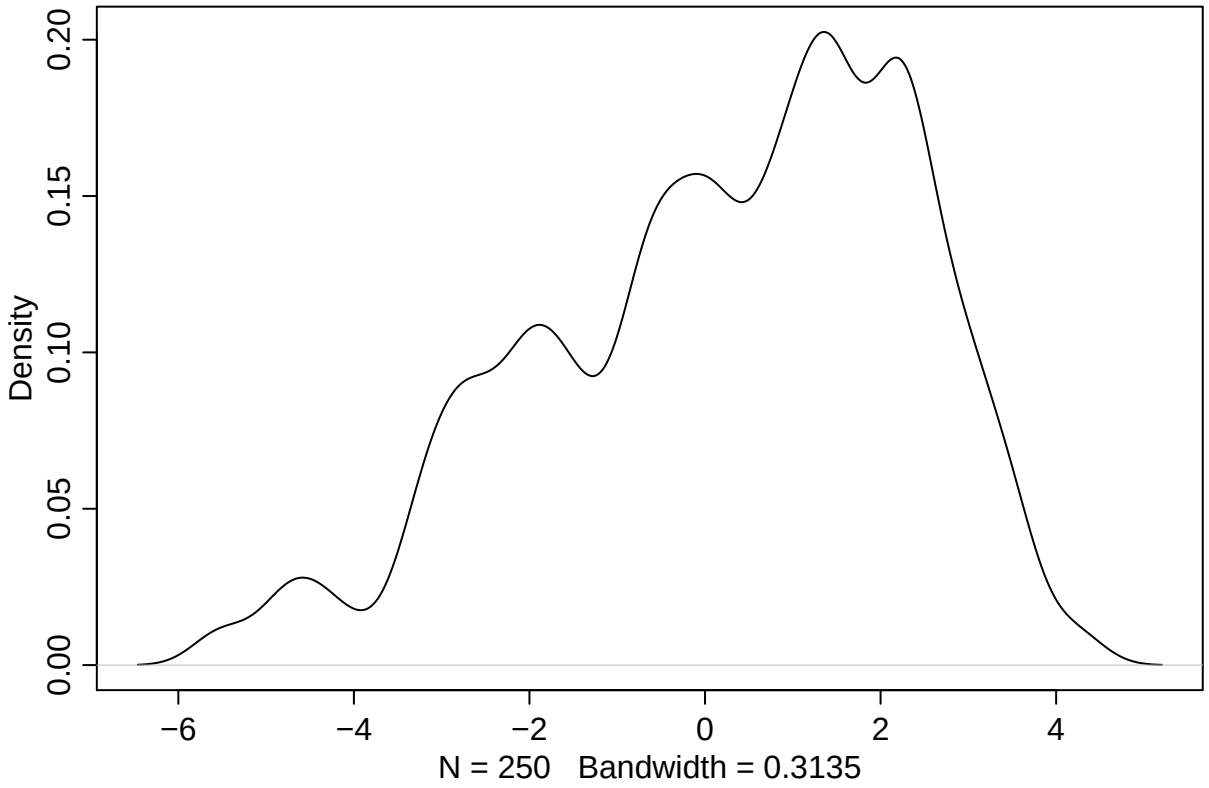
The **fish_caught** variable will be our response, and the other variables are candidates for predictors. Let's take a look at the distributions of the other variables so we can make a better decision about which predictors to include in our model. (Note that in order to do this, we'll need to create a second copy of the data frame where we convert the factor variables to factors so that **ggpairs()** can interpret properly.)

```
library(ggplot2)
library(GGally)
f_factor <- f
f_factor$livebait <- as.factor(f$livebait)
f_factor$camper <- as.factor(f$camper)
ggpairs(f_factor, lower = list(continuous = "smooth"), diag = list(continuous = "density"),
    axisLabels = "none")
```



We can see that the distribution of the **hours** variable looks like it could use a logarithmic transformation in order to be more normal.

```
f$hours <- log(f$hours)
dens(f$hours)
```



This still doesn't look very Normal, but it looks better than it did before we applied the transformation.

Since the question doesn't ask us to use specific predictors, we have some freedom to experiment and see which predictors create the best model. Personally, I think that **camper** is the only variable there which probably doesn't matter much at all. The rest seem like they could have some sort of effect on how many fish are caught by a group of campers. We can compute a couple of models and compare them by using model diagnostics. Either way, since this is a zero-inflated dataset, we'll need to use the **dzipois()** distribution.

Keep in mind, that a zero-inflated Poisson regression model always takes the general form

$$\begin{aligned} y_i &\sim \text{ZIPoisson}(p_i, \lambda_i) \\ \text{logit}(p_i) &\sim \alpha_p + \beta_p x_i \\ \log(\lambda_i) &\sim \alpha_\lambda + \beta_\lambda x_i \end{aligned}$$

and in this type of model, we have two linear models and two link functions (one for each process in the ZIPoisson). We're allowed to use different predictors in the two models, so there are a lot more possible combinations than in a typical regression problem like in the previous chapters. Thus, model exploration this could go on for a really long time, so I'll just try out a couple since we weren't asked to create a specific one.

The first model will be

$$\begin{aligned} \text{fish} &\sim \text{ZIPoisson}(p, \mu) \\ \text{logit}(p) &\sim \alpha_p + \beta_{pp}\text{persons} + \beta_{pc}\text{child} \\ \log(\mu) &\sim \alpha_\mu + \beta_{\mu p}\text{persons} + \beta_{\mu c}\text{child} + \beta_{\mu h}\log(\text{hours}) \end{aligned}$$

```
lmod11h1_1 <- map2stan(alist(
  fish_caught ~ dzipois(p, mu),
```

```
  logit(p) <- ap + bpp*persons + bpc*child,
  log(mu) <- am + bmp*persons + bmc*child + bmh*hours,
  c(ap,am) ~ dnorm(0,10),
  c(bpp, bpc, bmp, bmc, bmh) ~ dnorm(0,1)
), data = f, iter = 10000, chains = 1, cores = 1)
```

In the second model, I'll include the **livebait** variable. Thus, the second model will be:

$$
\begin{aligned}
\text{fish} &\sim \text{ZIPoisson}(p, \mu) \\
\text{logit}(p) &\sim \alpha_p + \beta_{pp}\text{persons} + \beta_{pc}\text{child} + \beta_{pl}\text{livebait} \\
\log(\mu) &\sim \alpha_\mu + \beta_{\mu p}\text{persons} + \beta_{\mu p} + \beta_{\mu c}\text{child} + \beta_{\mu l}\text{livebait} + \beta_{\mu h}\log(\text{hours})
\end{aligned}
$$

```
lmod11h1_2 <- map2stan(alist(
  fish_caught ~ dzipois(p, mu),
  logit(p) <- ap + bpp*persons + bpc*child + bpl*livebait,
  log(mu) <- am + bmp*persons + bmc*child + bml*livebait + bmh*hours,
  c(ap,am) ~ dnorm(0,10),
  c(bpp, bpc, bmp, bmc, bmh, bpl, bml) ~ dnorm(0,1)
), data = f, iter = 10000, chains = 1, cores = 1)
```

Now we can compare the two models.

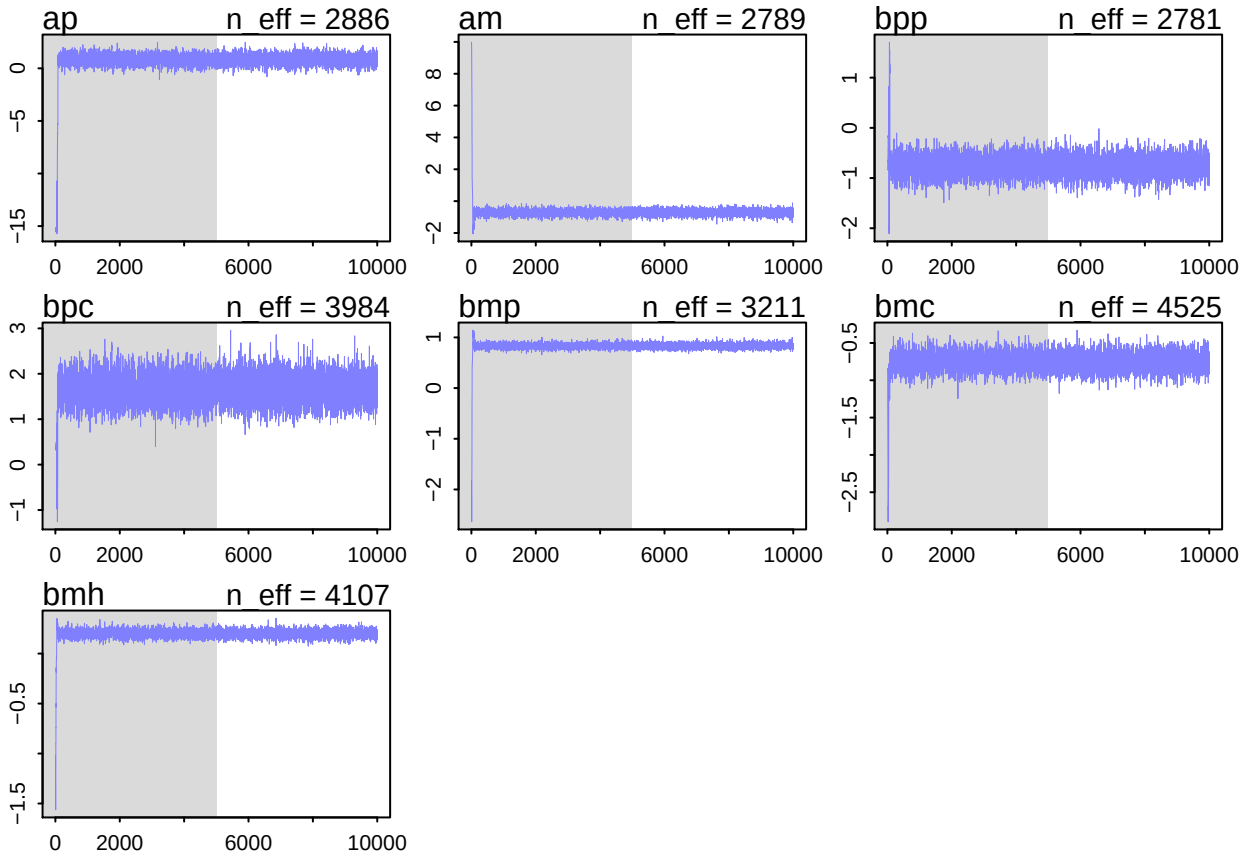
```
precis(lmod11h1_1)
```

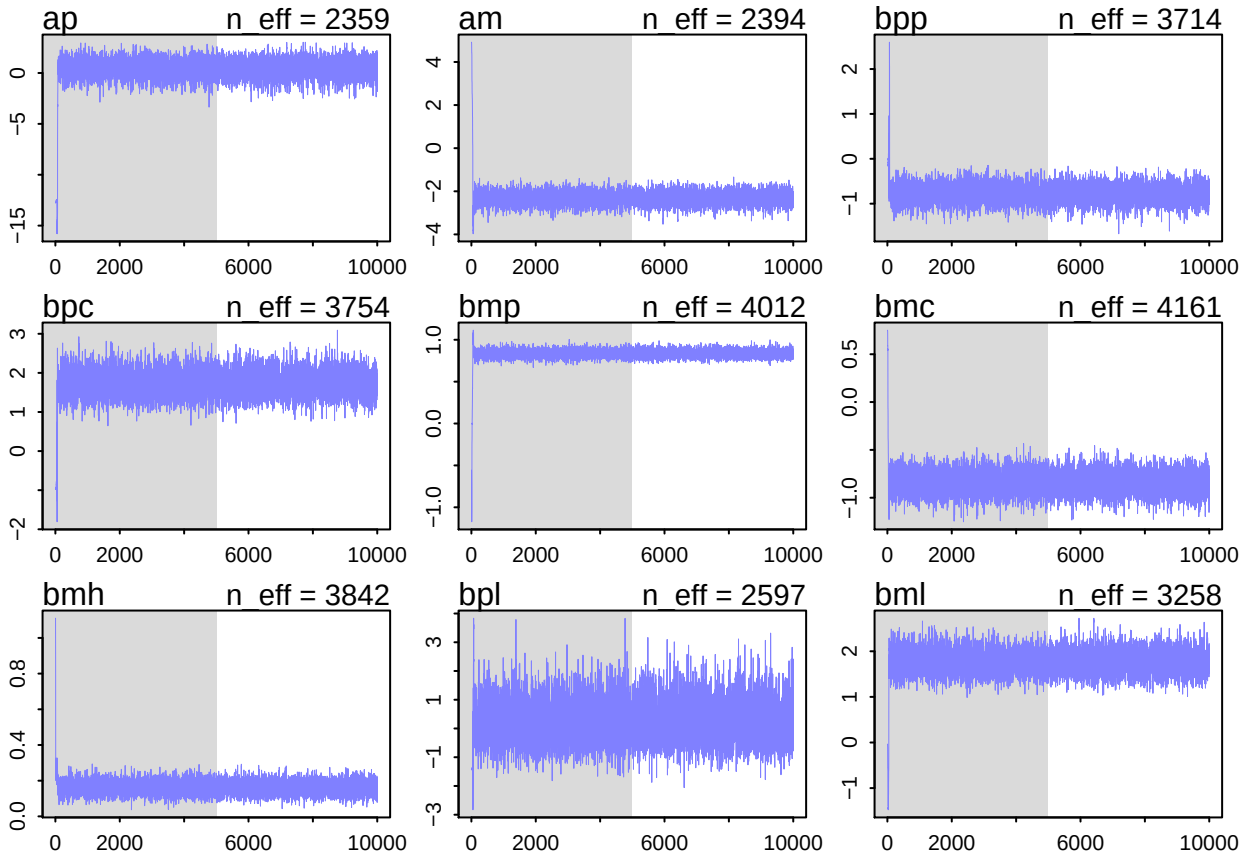
```
##      Mean StdDev lower 0.89 upper 0.89 n_eff Rhat
## ap   0.85   0.44       0.15       1.54  2886    1
## am  -0.71   0.17      -0.99      -0.44  2789    1
## bpp -0.76   0.18      -1.06      -0.47  2781    1
## bpc  1.66   0.29       1.22       2.16  3984    1
## bmp  0.83   0.04       0.76       0.90  3211    1
## bmc -0.75   0.11      -0.93      -0.58  4525    1
## bmh  0.20   0.03       0.15       0.25  4107    1
```

```
precis(lmod11h1_2)
```

```
##      Mean StdDev lower 0.89 upper 0.89 n_eff Rhat
## ap   0.46   0.86      -0.87       1.85  2359    1
## am  -2.32   0.29      -2.78      -1.84  2394    1
## bpp -0.78   0.19      -1.08      -0.47  3714    1
## bpc  1.73   0.31       1.21       2.20  3754    1
## bmp  0.84   0.04       0.77       0.91  4012    1
## bmc -0.84   0.11      -1.01      -0.66  4161    1
## bmh  0.16   0.04       0.11       0.22  3842    1
## bpl  0.30   0.73      -0.85       1.41  2597    1
## bml  1.78   0.24       1.39       2.17  3258    1
```

```
plot(lmod11h1_1)
plot(lmod11h1_2)
```


ap
n_eff = 2886
am
n_eff = 2789
bpp
n_eff = 2781
bpc
n_eff = 3984
bmp
n_eff = 3211
bmc
n_eff = 4525
bmh
n_eff = 4107
0
2000
6000
10000




```
compare(lmod11h1_1, lmod11h1_2)
```

```
##              WAIC pWAIC dWAIC weight     SE   dSE
## lmod11h1_2 1601.5 103.8     0      1 365.83    NA
## lmod11h1_1 1695.5 105.3    94      0 383.65 48.53
```

The Rhat values and n_eff values are pretty good for both models, but we can see that the traceplots for the second model are a bit better, and the second model took 100% of the Akaike weight. Thus we can conclude that our second model was the "better" model.

This question doesn't actually ask us to do anything aside from creating the models, but it would still be interesting to explore our model a bit to see how zero-inflated problems are a bit different than the models we've used previously.

```
f$fish_caught
```

```
##   [1]   0   0   0   0   1   0   0   0   0   1   0   0   1   2   0   1   0
##  [18]   0   1   0   1   5   0   3  30   0  13   0   0   0   0   0  11   5
##  [35]   0   1   1   7   0  14   0  32   0   1   0   0   0   1   5   0   1
##  [52]   0  22   0  15   0   0   0   5   4   2   0   2  32   0   0   1   0
##  [69]   0   0   7   0   0   0   0   0   0   0   0   2   3   1   5   0   2
##  [86]   1   0   1 149   0   1   0   0   1   0   0   0   2   2  29   3   0
## [103]   0   5   0   0   0   0   0   1   7   1   0   2   0   2   0   0   0
## [120]   1   0   0   0   0   0   3   4   3   3   8   2   1   6   0   0   5
## [137]   3  31   0   2   0   0   0   0   0   0   6   9   0   0   0   0   0
## [154]   2  15   1   2   3   0  65   5   0   0   0   0   1   8   0   0   0
## [171]   2   4   5   9   0   0   0   0  21   0   6   0   0   0   0  16   0
## [188]   0   4   2  10   0   0   0   2   1   3   0   0  21   0   0   2   0
## [205]   3   0  38   0   0   0   1   3   0   1   0   0   0   0   5   0   0
```

```
## [222]   2   0   0   0   1   4   0   0   2   3   0   0   0   0   1   2   0
## [239]   6   4   1   1   0   1   0   0   0   0   0   0
```

As you can see, there are a LOT of zeroes. But, using our model from **map2stan()**, we can simulate some counterfactual data just with using the **link()** function. Let's see how many fish we would expect a group of 4 adults to catch if they went fishing for 3 hours using livebait, since this is a pretty typical situation in real life. We have to keep in mind, though, that we applied a logarithmic transformation to the **hours** variable, so we have to apply this same transformation to the 3 hours.

```
new_data <- list(hours = log(3), persons = 4, child = 0, livebait = 1)
counterfactual_data <- link(1mod11h1_2, new_data)
```

```
## [ 100 / 1000 ]
[ 200 / 1000 ]
[ 300 / 1000 ]
[ 400 / 1000 ]
[ 500 / 1000 ]
[ 600 / 1000 ]
[ 700 / 1000 ]
[ 800 / 1000 ]
[ 900 / 1000 ]
[ 1000 / 1000 ]
```

```
p <- counterfactual_data$p
mu <- counterfactual_data$mu
```

Now, according to Wikipedia, the mean of the Zero-Inflated Poisson Distribution is

$$(1-p)\mu$$

Thus we have

```
mean((1-p)*mu)
```

```
## [1] 17.93279
```

Based on this simulation, we can see that a group of 4 adults fishing for 3 hours with live bait would be expected to catch 17.9327867 fish on average (this accounts for the zero-inflation). This sounds pretty reasonable, especially when we take a look at the distribution of the **fish_caught** variable.

```
f[order(-f$fish_caught)[1:5],]
```

```
##     fish_caught livebait camper persons child    hours
## 89          149        1      1       4     0 3.572121
## 160          65        1      1       4     0 1.362258
## 207          38        1      1       4     0 1.189367
## 42           32        1      1       4     0 2.355652
## 64           32        1      1       4     0 2.547960
```

We can see that in the data, there are quite a few groups of 4 adult fishers with livebait who caught quite a lot more than our estimate – but keep in mind that our data was inflated with zeroes, which pulled down the expected mean. (It's not terrrrrrrribly far off though; you can see there's one group who was there for ~2.5h and caught 32 fish, so we're at least on the right order of magnitude. This might suggest that there could've been better models though.)

References

1. https://en.wikipedia.org/wiki/Zero-inflated_model#Zero-inflated_Poisson