Methods 4 – Portfolio Assignment 3

Exercise 11H3

The data contained in `library(MASS); data(eagles)` are records of salmon pirating attempts by Bald Eagles in Washington State. See `?eagles` for details. 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.

```
pacman::p_load(MASS, rethinking, tidyverse, tidybayes, tidybayes.rethinking, boot, kn
itr)
devtools::install_github('mjskay/tidybayes.rethinking')
```

```
## Skipping install of 'tidybayes.rethinking' from a github remote, the SHA1 (7da9946
6) has not changed since last install.
##   Use `force = TRUE` to force installation
```

```
set_cmdstan_path('C:/.cmdstan/cmdstan-2.31.0')
```

```
## Warning: Path not set. Can't find directory: C:/.cmdstan/cmdstan-2.31.0
```

```
data(eagles)
eagles <- eagles
```

(a) (MST)

Consider the following model:

$$y_i \sim \text{Binomial}(n_i, p_i)$$

$$\log \frac{p_i}{1 - p_i} = \alpha + \beta_P P_i + \beta_V V_i + \beta_A A_i$$

$$\begin{aligned} \alpha &\sim \text{Normal}(0, 1.5) \\ \beta_P &\sim \text{Normal}(0, 0.5) \\ \beta_V &\sim \text{Normal}(0, 0.5) \\ \beta_A &\sim \text{Normal}(0, 0.5) \end{aligned}$$

where y is the number of successful attempts, n is the total number of attempts, P is a dummy variable indicating whether or not the pirate had large body size, V is a dummy variable indicating whether or not the victim had large body size, and finally A is a dummy variable indicating whether or not the pirate was an adult. Fit the model above to the eagles data, using both `quap` and `ulam`. Is the quadratic approximation okay?

```
m_eagles_quap <- quap(
    alist(
        y ~ dbinom( n , p ) ,
        logit(p) <- a + Bp[P] + Bv[V] + Ba[A],
        a ~ dnorm(0,1.5),
        Bp[P] ~ dnorm(0,0.5),
        Bv[V] ~ dnorm(0,0.5),
        Ba[A] ~ dnorm(0,0.5)
) , data=eagles )

precis( m_eagles_quap , depth=2 )
```

	mean <dbl>	**sd** <dbl>	**5.5%** <dbl>	**94.5%** <dbl>
a	0.5632031	0.5979207	-0.3923897	1.5187959
Bp[1]	1.0757671	0.3908947	0.4510419	1.7004923
Bp[2]	-1.0131895	0.3917740	-1.6393199	-0.3870590
Bv[1]	-1.0632548	0.3956189	-1.6955301	-0.4309794
Bv[2]	1.1258294	0.3992853	0.4876945	1.7639644
Ba[1]	0.4162628	0.3895095	-0.2062485	1.0387742
Ba[2]	-0.3536712	0.3896532	-0.9764123	0.2690699

7 rows

```
m_eagles_ulam <- ulam(
    alist(
        y ~ dbinom( n , p ) ,
        logit(p) <- a + Bp[P] + Bv[V] + Ba[A],
        a ~ dnorm(0,1.5),
        Bp[P] ~ dnorm(0,0.5),
        Bv[V] ~ dnorm(0,0.5),
        Ba[A] ~ dnorm(0,0.5)
    ) , data=eagles , chains=4 , log_lik=TRUE )
```

```
## Warning in '/var/folders/wv/k1c_2q2x52q536wp2_p2kdpm0000gn/T/RtmpZQu5fd/model-70d5
2735c4a2.stan', line 2, column 4: Declaration
##     of arrays by placing brackets after a variable name is deprecated and
##     will be removed in Stan 2.32.0. Instead use the array keyword before the
##     type. This can be changed automatically using the auto-format flag to
##     stanc
## Warning in '/var/folders/wv/k1c_2q2x52q536wp2_p2kdpm0000gn/T/RtmpZQu5fd/model-70d5
2735c4a2.stan', line 3, column 4: Declaration
##     of arrays by placing brackets after a variable name is deprecated and
##     will be removed in Stan 2.32.0. Instead use the array keyword before the
##     type. This can be changed automatically using the auto-format flag to
##     stanc
## Warning in '/var/folders/wv/k1c_2q2x52q536wp2_p2kdpm0000gn/T/RtmpZQu5fd/model-70d5
2735c4a2.stan', line 4, column 4: Declaration
##     of arrays by placing brackets after a variable name is deprecated and
##     will be removed in Stan 2.32.0. Instead use the array keyword before the
##     type. This can be changed automatically using the auto-format flag to
##     stanc
## Warning in '/var/folders/wv/k1c_2q2x52q536wp2_p2kdpm0000gn/T/RtmpZQu5fd/model-70d5
2735c4a2.stan', line 5, column 4: Declaration
##     of arrays by placing brackets after a variable name is deprecated and
##     will be removed in Stan 2.32.0. Instead use the array keyword before the
##     type. This can be changed automatically using the auto-format flag to
##     stanc
## Warning in '/var/folders/wv/k1c_2q2x52q536wp2_p2kdpm0000gn/T/RtmpZQu5fd/model-70d5
2735c4a2.stan', line 6, column 4: Declaration
##     of arrays by placing brackets after a variable name is deprecated and
##     will be removed in Stan 2.32.0. Instead use the array keyword before the
##     type. This can be changed automatically using the auto-format flag to
##     stanc
```

```
## Running MCMC with 4 sequential chains, with 1 thread(s) per chain...
##
## 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 finished in 0.0 seconds.
## 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 finished in 0.0 seconds.
## 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 finished in 0.1 seconds.
## 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 finished in 0.0 seconds.
##
## All 4 chains finished successfully.
## Mean chain execution time: 0.0 seconds.
## Total execution time: 0.5 seconds.
```

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

	mean <dbl>	sd <dbl>	5.5% <dbl>	94.5% <dbl>	n_eff <dbl>	Rhat4 <dbl>
a	0.6084341	0.6085243	-0.3857814	1.5963115	887.4251	1.005812
Bp[1]	1.0872098	0.3853811	0.4808891	1.7198065	1369.4657	1.001873
Bp[2]	-1.0399817	0.4008788	-1.6962070	-0.4048653	1443.7638	1.000347
Bv[1]	-1.0927238	0.3959441	-1.7225519	-0.4400346	1179.6936	1.005827
Bv[2]	1.1493537	0.4016688	0.5214061	1.7991543	1273.3468	1.001441
Ba[1]	0.4058532	0.3873639	-0.2213873	1.0442360	1405.8730	0.999834
Ba[2]	-0.3741589	0.3820879	-0.9697655	0.2380597	1370.4341	1.000955

7 rows

```
compare(m_eagles_quap, m_eagles_ulam)
```

```
## Warning in compare(m_eagles_quap, m_eagles_ulam): Not all model fits of same clas
s.
## This is usually a bad idea, because it implies they were fit by different algorith
ms.
## Check yourself, before you wreck yourself.
```

	WAIC <dbl>	SE <dbl>	dWAIC <dbl>	dSE <dbl>	pWAIC <dbl>	weight <dbl>
m_eagles_ulam	45.17854	6.913244	0.000000	NA	6.133150	0.7165406
m_eagles_quap	47.03327	7.405123	1.854732	0.790555	6.846136	0.2834594

2 rows

Comparing the ulam and quap model, we see that the ulam is slightly better. We can conclude, that the estimates are very similar and thus the quadratic approximation is okay.

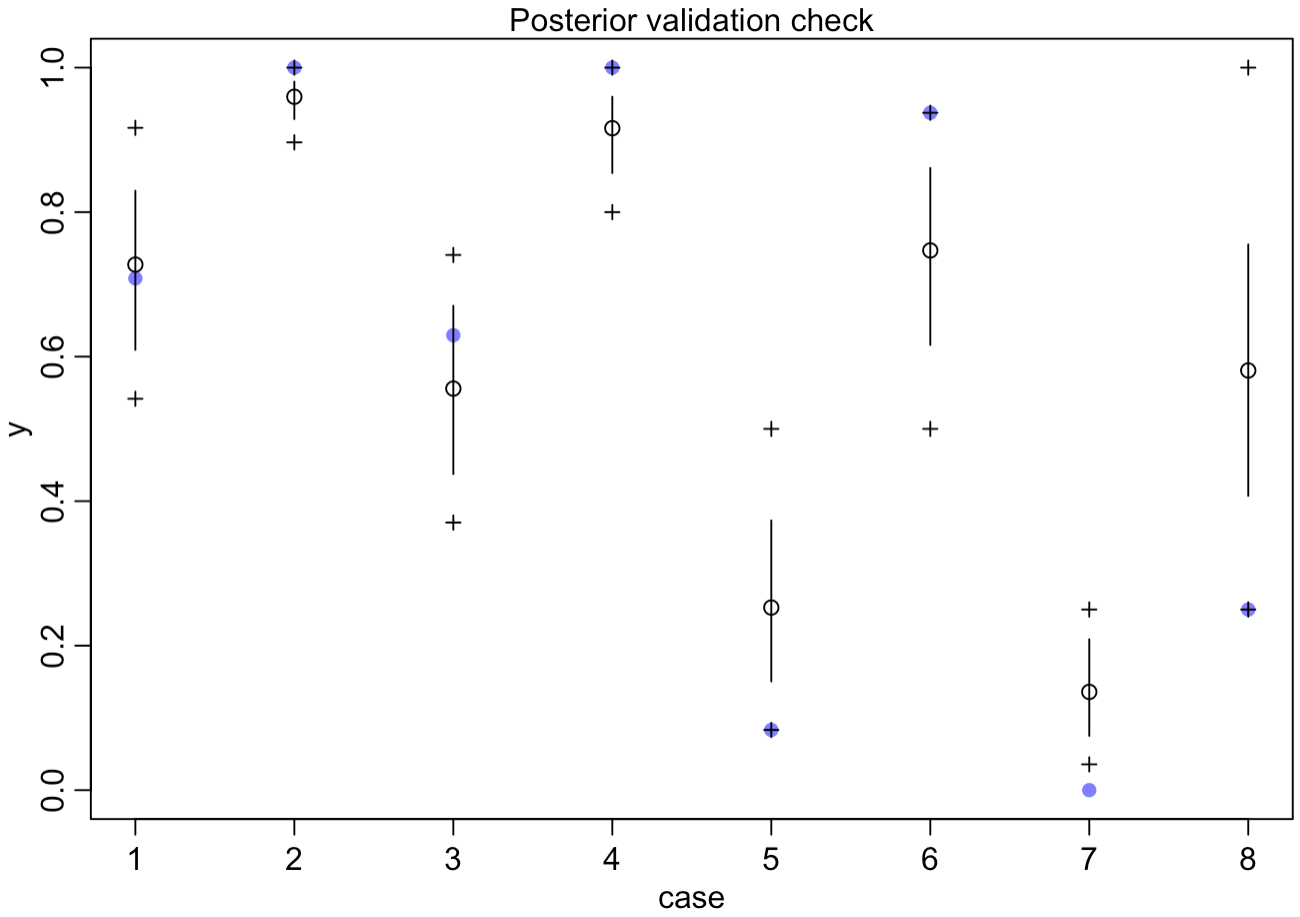
(b) (LR)

Now interpret the estimates. If the quadratic approximation turned out okay, then it's okay to use the quap

estimates. Otherwise stick to `ulam` estimates. Then plot the posterior predictions. Compute and display both (1) the predicted probability of success and its 89% interval for each row (i) in the data, as well as (2) the predicted success count and its 89% interval. What different information does each type of posterior prediction provide?

First, we plot the predicted probability of successes and their 89% intervals.

```
postcheck(m_eagles_ulam, prob = 0.89, window = 20, n = 1000, col=rangi2)
```



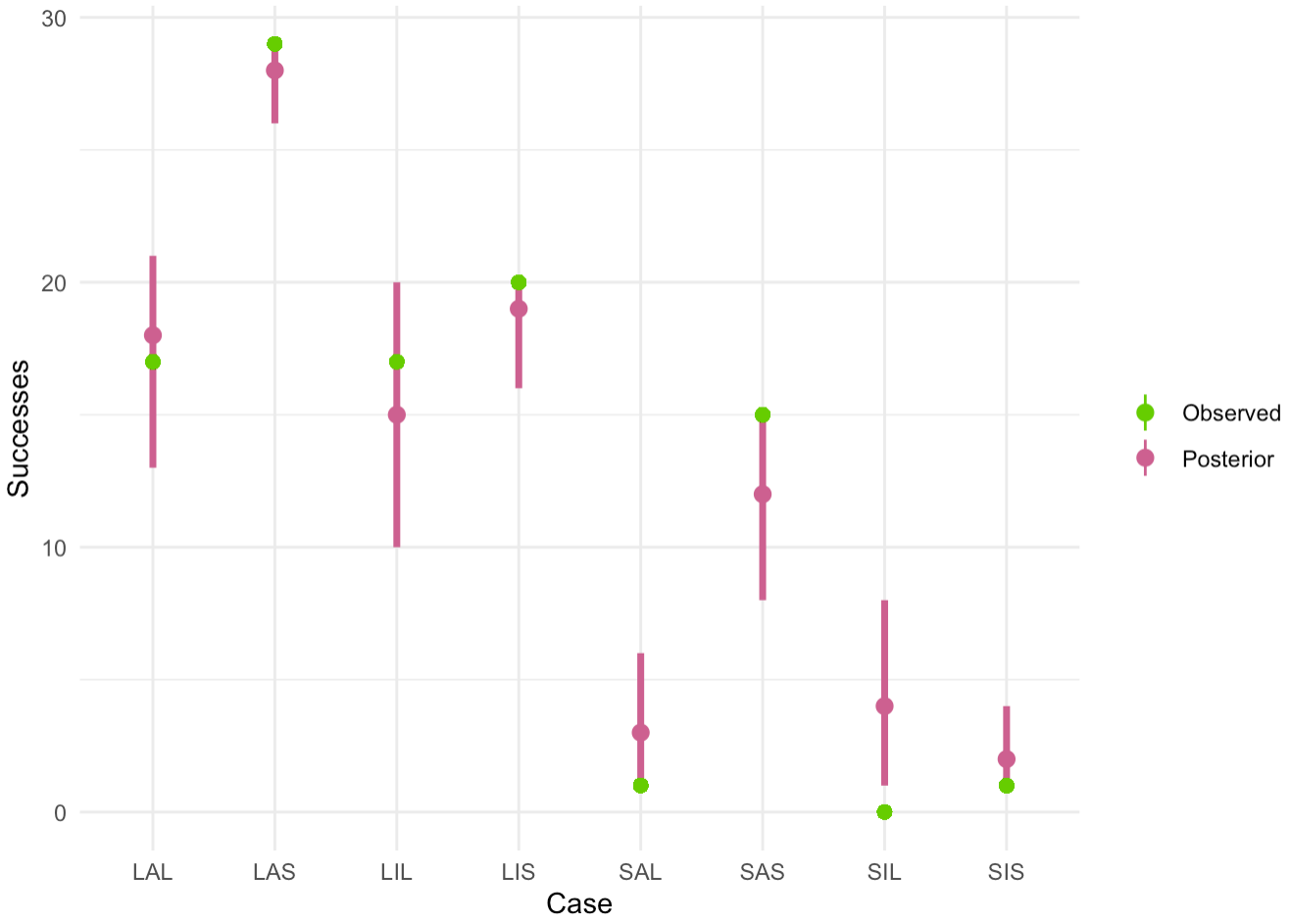
Blue dots = observed data White dots = posterior estimates Lines at white dots = 89 % intervals

Second, we plot the predicted success count and its 89% intervals.

```
predictions <- tidybayes::predicted_draws(
  m_eagles_ulam,
  eagles,
  value = ".prediction",
  ndraws = NULL,
  seed = NULL,
  re_formula = NULL
)
```

```
predictions %>%
  mutate(label = paste0(P, A, V)) %>%
  ggplot(aes(x = label, y = .prediction)) +
  stat_pointinterval(aes(color = "Posterior"), .width = 0.89, size = 5) +
  geom_point(data = predictions, size = 2,
             aes(x = paste0(P, A, V), y = y, color = "Observed")) +
  scale_color_manual(values = c("Posterior" = "hotpink3",
                                "Observed" = "chartreuse3"),
                     name = NULL) +
  labs(x = "Case", y = "Successes") + theme_minimal()
```

```
## Warning: Using the `size` aesthietic with geom_segment was deprecated in ggplot2
3.4.0.
## ℹ Please use the `linewidth` aesthetic instead.
```



The two posterior prediction plots provide different information. The plot showing the predicted probability of success visualizes the relative values in terms of probability, where the plot for predicted success counts visualizes absolute values.

(c) (SM & TI)

Now try to improve the model. Consider an interaction between the pirate's size and age (immature or adult). Compare this model to the previous one, using WAIC. Interpret.

```
set.seed(123)
eagles <- eagles %>%
  mutate(dP = ifelse(P == "L", 1, 2),
         dV = ifelse(V == "L", 1, 2),
         dA = ifelse(A == "A", 1, 2))

m2_eagles_ulam <- ulam(
    alist(
        y ~ dbinom( n , p ) ,
        logit(p) <- a[dP,dA] + Bv[dV] ,
        Bv[dV] ~ dnorm(0,0.5),
        matrix[dP,dA]:a ~ normal(0,1)
    ) , data=eagles , chains=4 , log_lik=TRUE )
```

```
## Warning in '/var/folders/wv/k1c_2q2x52q536wp2_p2kdpm0000gn/T/RtmpZQu5fd/model-70d5
730e1de7.stan', line 2, column 4: Declaration
##     of arrays by placing brackets after a variable name is deprecated and
##     will be removed in Stan 2.32.0. Instead use the array keyword before the
##     type. This can be changed automatically using the auto-format flag to
##     stanc
## Warning in '/var/folders/wv/k1c_2q2x52q536wp2_p2kdpm0000gn/T/RtmpZQu5fd/model-70d5
730e1de7.stan', line 3, column 4: Declaration
##     of arrays by placing brackets after a variable name is deprecated and
##     will be removed in Stan 2.32.0. Instead use the array keyword before the
##     type. This can be changed automatically using the auto-format flag to
##     stanc
## Warning in '/var/folders/wv/k1c_2q2x52q536wp2_p2kdpm0000gn/T/RtmpZQu5fd/model-70d5
730e1de7.stan', line 4, column 4: Declaration
##     of arrays by placing brackets after a variable name is deprecated and
##     will be removed in Stan 2.32.0. Instead use the array keyword before the
##     type. This can be changed automatically using the auto-format flag to
##     stanc
## Warning in '/var/folders/wv/k1c_2q2x52q536wp2_p2kdpm0000gn/T/RtmpZQu5fd/model-70d5
730e1de7.stan', line 5, column 4: Declaration
##     of arrays by placing brackets after a variable name is deprecated and
##     will be removed in Stan 2.32.0. Instead use the array keyword before the
##     type. This can be changed automatically using the auto-format flag to
##     stanc
## Warning in '/var/folders/wv/k1c_2q2x52q536wp2_p2kdpm0000gn/T/RtmpZQu5fd/model-70d5
730e1de7.stan', line 6, column 4: Declaration
##     of arrays by placing brackets after a variable name is deprecated and
##     will be removed in Stan 2.32.0. Instead use the array keyword before the
##     type. This can be changed automatically using the auto-format flag to
##     stanc
## Warning in '/var/folders/wv/k1c_2q2x52q536wp2_p2kdpm0000gn/T/RtmpZQu5fd/model-70d5
730e1de7.stan', line 7, column 4: Declaration
##     of arrays by placing brackets after a variable name i
```

```
## s deprecated and
##     will be removed in Stan 2.32.0. Instead use the array keyword before the
##     type. This can be changed automatically using the auto-format flag to
##     stanc
## Warning in '/var/folders/wv/k1c_2q2x52q536wp2_p2kdpm0000gn/T/RtmpZQu5fd/model-70d5
730e1de7.stan', line 8, column 4: Declaration
##     of arrays by placing brackets after a variable name is deprecated and
##     will be removed in Stan 2.32.0. Instead use the array keyword before the
##     type. This can be changed automatically using the auto-format flag to
##     stanc
## Warning in '/var/folders/wv/k1c_2q2x52q536wp2_p2kdpm0000gn/T/RtmpZQu5fd/model-70d5
730e1de7.stan', line 9, column 4: Declaration
##     of arrays by placing brackets after a variable name is deprecated and
##     will be removed in Stan 2.32.0. Instead use the array keyword before the
##     type. This can be changed automatically using the auto-format flag to
##     stanc
```

```
## Running MCMC with 4 sequential chains, with 1 thread(s) per chain...
##
## 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 finished in 0.0 seconds.
## 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 finished in 0.0 seconds.
## 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 finished in 0.0 seconds.
## 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 finished in 0.0 seconds.
##
## All 4 chains finished successfully.
## Mean chain execution time: 0.0 seconds.
## Total execution time: 0.5 seconds.
```

```
m3_eagles_ulam <- ulam(
    alist(
        y ~ dbinom( n , p ) ,
        logit(p) <- a[dP, dV] + Ba[dA],
        Ba[dA] ~ dnorm(0,0.5),
        matrix[dP,dV]:a ~ normal(0,1)
    ) , data=eagles , chains=4 , log_lik=TRUE )
```

```
## Warning in '/var/folders/wv/k1c_2q2x52q536wp2_p2kdpm0000gn/T/RtmpZQu5fd/model-70d5
33831ec.stan', line 2, column 4: Declaration
##     of arrays by placing brackets after a variable name is deprecated and
##     will be removed in Stan 2.32.0. Instead use the array keyword before the
##     type. This can be changed automatically using the auto-format flag to
##     stanc
## Warning in '/var/folders/wv/k1c_2q2x52q536wp2_p2kdpm0000gn/T/RtmpZQu5fd/model-70d5
33831ec.stan', line 3, column 4: Declaration
##     of arrays by placing brackets after a variable name is deprecated and
##     will be removed in Stan 2.32.0. Instead use the array keyword before the
##     type. This can be changed automatically using the auto-format flag to
##     stanc
## Warning in '/var/folders/wv/k1c_2q2x52q536wp2_p2kdpm0000gn/T/RtmpZQu5fd/model-70d5
33831ec.stan', line 4, column 4: Declaration
##     of arrays by placing brackets after a variable name is deprecated and
##     will be removed in Stan 2.32.0. Instead use the array keyword before the
##     type. This can be changed automatically using the auto-format flag to
##     stanc
## Warning in '/var/folders/wv/k1c_2q2x52q536wp2_p2kdpm0000gn/T/RtmpZQu5fd/model-70d5
33831ec.stan', line 5, column 4: Declaration
##     of arrays by placing brackets after a variable name is deprecated and
##     will be removed in Stan 2.32.0. Instead use the array keyword before the
##     type. This can be changed automatically using the auto-format flag to
##     stanc
## Warning in '/var/folders/wv/k1c_2q2x52q536wp2_p2kdpm0000gn/T/RtmpZQu5fd/model-70d5
33831ec.stan', line 6, column 4: Declaration
##     of arrays by placing brackets after a variable name is deprecated and
##     will be removed in Stan 2.32.0. Instead use the array keyword before the
##     type. This can be changed automatically using the auto-format flag to
##     stanc
## Warning in '/var/folders/wv/k1c_2q2x52q536wp2_p2kdpm0000gn/T/RtmpZQu5fd/model-70d5
33831ec.stan', line 7, column 4: Declaration
##     of arrays by placing brackets after a variable name is depr
```

```
## ecated and
##     will be removed in Stan 2.32.0. Instead use the array keyword before the
##     type. This can be changed automatically using the auto-format flag to
##     stanc
## Warning in '/var/folders/wv/k1c_2q2x52q536wp2_p2kdpm0000gn/T/RtmpZQu5fd/model-70d5
33831ec.stan', line 8, column 4: Declaration
##     of arrays by placing brackets after a variable name is deprecated and
##     will be removed in Stan 2.32.0. Instead use the array keyword before the
##     type. This can be changed automatically using the auto-format flag to
##     stanc
## Warning in '/var/folders/wv/k1c_2q2x52q536wp2_p2kdpm0000gn/T/RtmpZQu5fd/model-70d5
33831ec.stan', line 9, column 4: Declaration
##     of arrays by placing brackets after a variable name is deprecated and
##     will be removed in Stan 2.32.0. Instead use the array keyword before the
##     type. This can be changed automatically using the auto-format flag to
##     stanc
```

```
## Running MCMC with 4 sequential chains, with 1 thread(s) per chain...
##
## 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 finished in 0.0 seconds.
## 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 finished in 0.0 seconds.
## 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 finished in 0.0 seconds.
## 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)
```

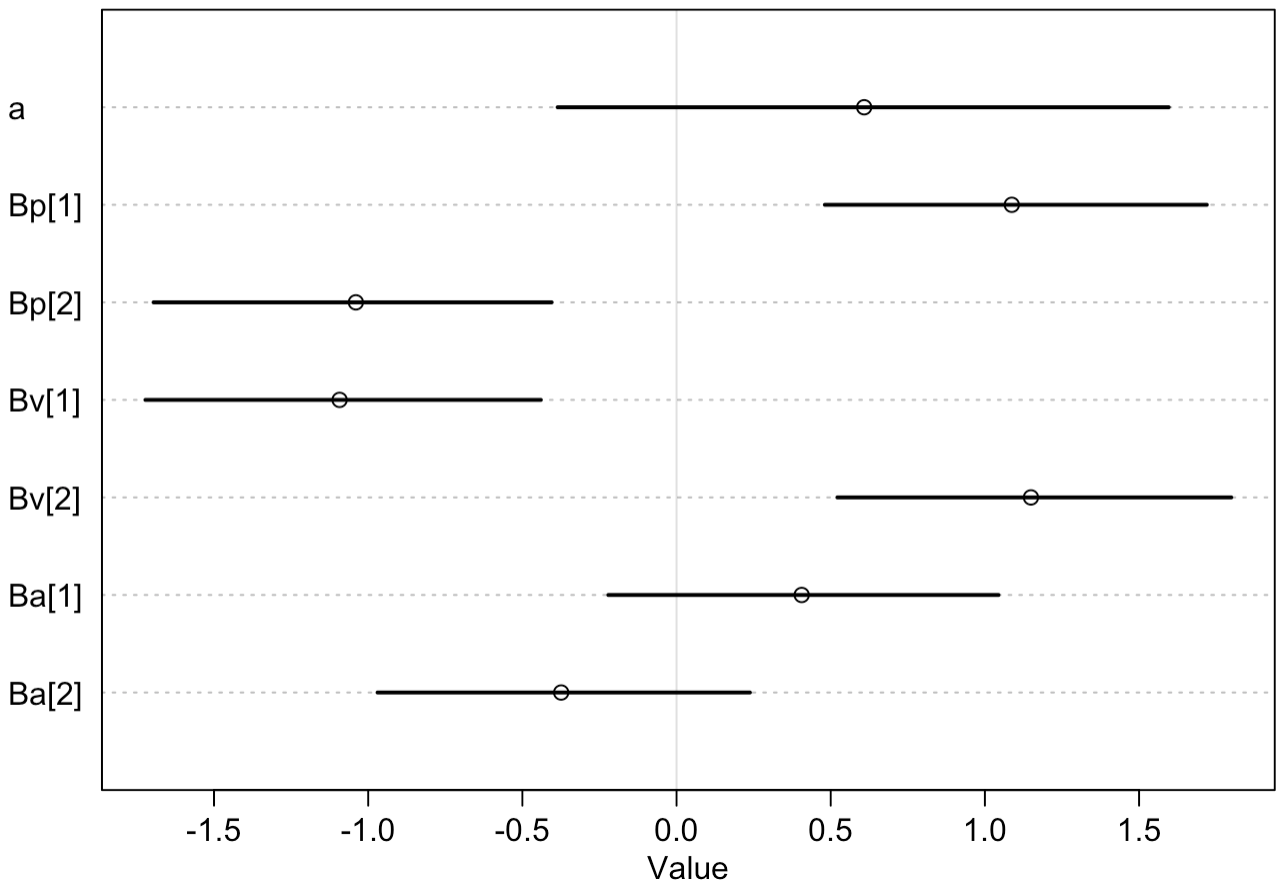
```
## Chain 4 Iteration: 900 / 1000 [ 90%]  (Sampling)
## Chain 4 Iteration: 1000 / 1000 [100%]  (Sampling)
## Chain 4 finished in 0.0 seconds.
##
## All 4 chains finished successfully.
## Mean chain execution time: 0.0 seconds.
## Total execution time: 0.5 seconds.
```

```
compare(m_eagles_ulam, m2_eagles_ulam, m3_eagles_ulam)
```

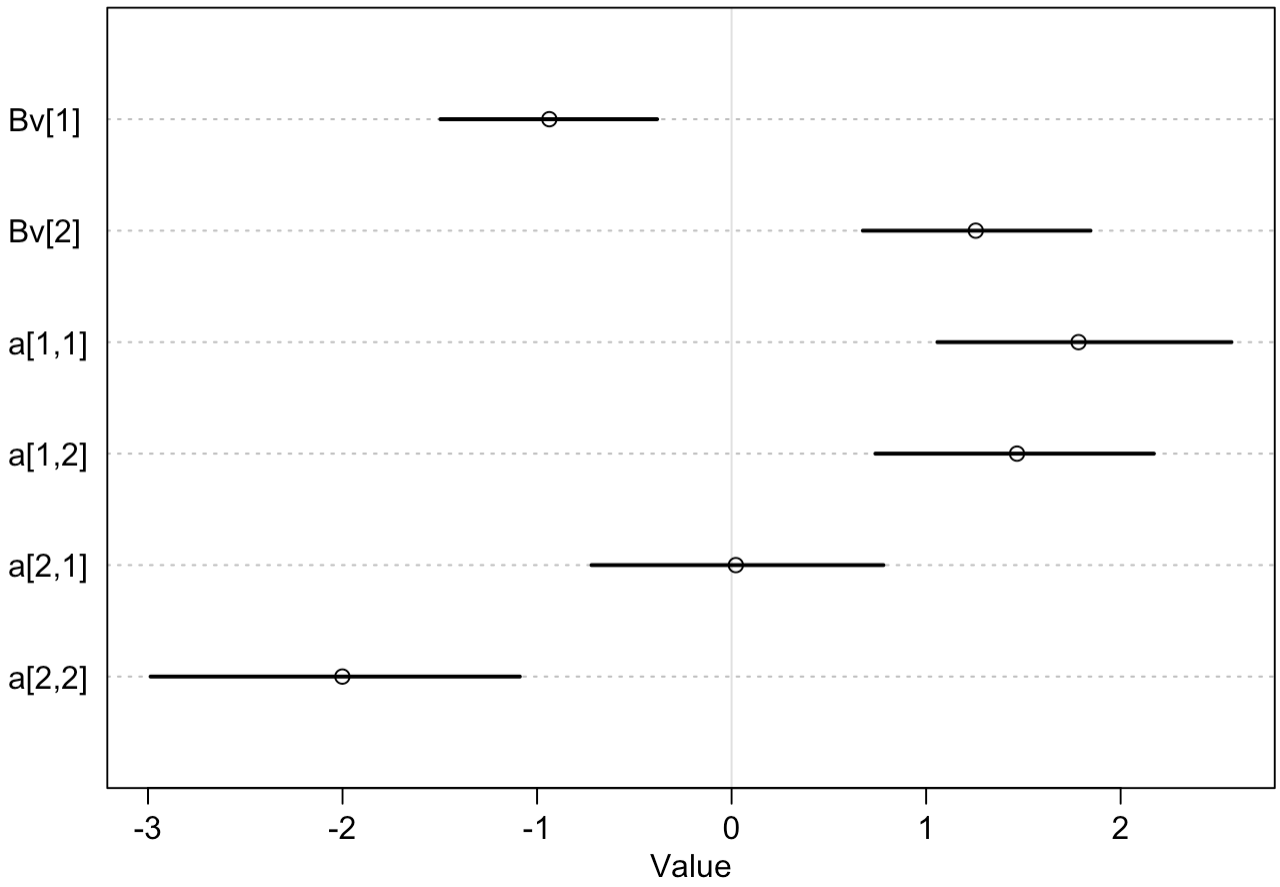
	WAIC <dbl>	SE <dbl>	dWAIC <dbl>	dSE <dbl>	pWAIC <dbl>	weight <dbl>
m2_eagles_ulam	37.45531	4.691664	0.0000000	*NA*	5.144304	0.54789190
m3_eagles_ulam	37.89127	3.658230	0.4359551	6.506374	5.432340	0.44058371
m_eagles_ulam	45.17854	6.913244	7.7232247	7.153980	6.133150	0.01152439

3 rows

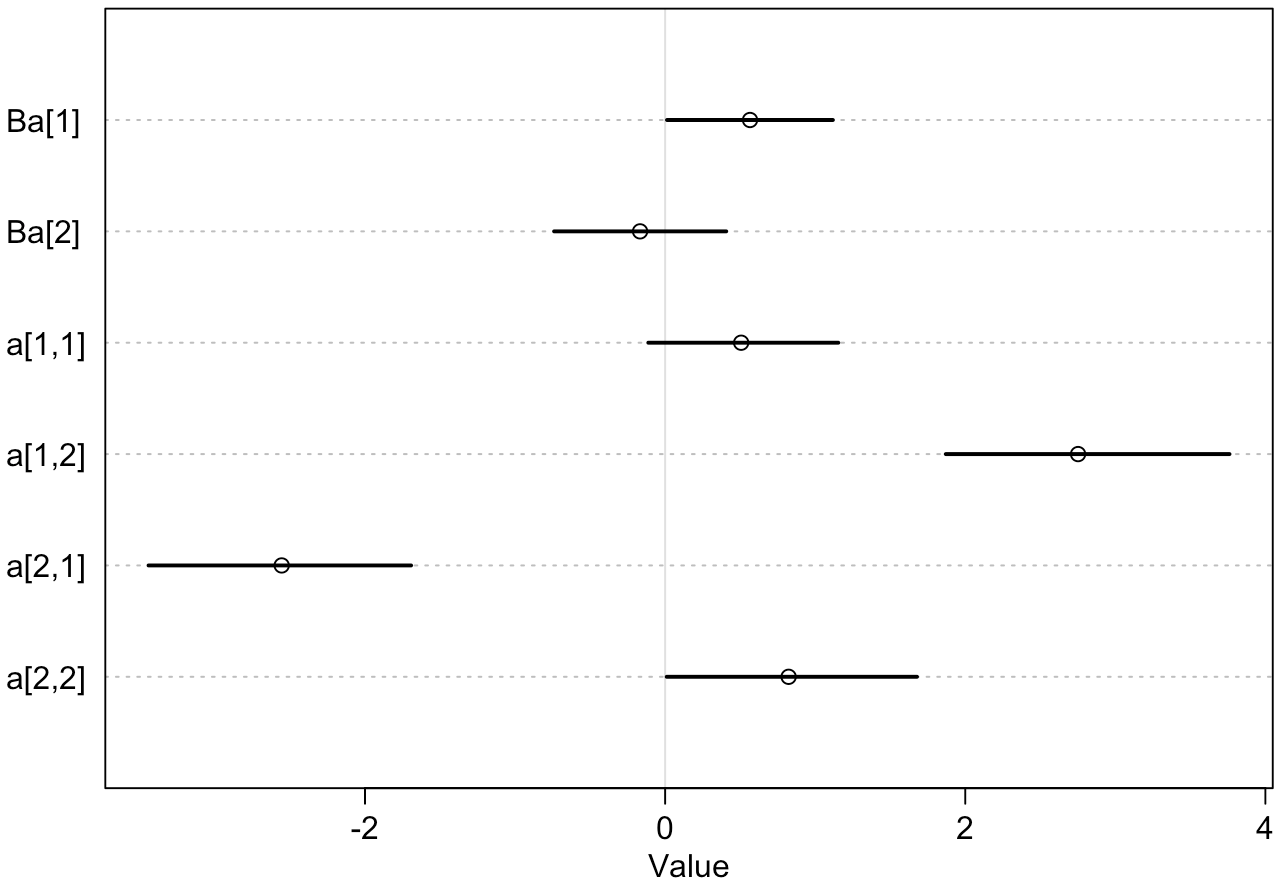
```
plot(precis(m_eagles_ulam, depth = 2))
```



```
plot(precis(m2_eagles_ulam, depth = 4))
```




```
plot(precis(m3_eagles_ulam, depth = 4))
```



From comparing the models using WAIC, we can conclude that model 2, which accounts for an interaction between age and body size of the pirate, has lowest WAIC and therefore the best performance. However, model 3, which accounts for an interaction between body size of the pirate and body size of the victim, performs almost as well. The difference between them is below 1 point on the WAIC scale and they have similar standard deviations.