hw11

Christopher Huong

2024-04-20

11E1. If an event has probability 0.35, what are the log-odds of this event?

log odds = log(p / (1-p))

```
p = 0.35
log(p / (1-p))
```

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

11E2. If an event has log-odds 3.2, what is the probability of this event?

e^3.2 = p / (1-p)

```
# after some high school algebra
# p =
exp(1)^3.2 / (1+exp(1)^3.2)
```

```
## [1] 0.9608343
```

11E3. 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

The logit scale = log-odds. So one unit change in x leads to a corresponding change in log-odds of success. So exponentiating the coefficient yields you the change in odds.

```
exp(1.7)
```

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

One unit change in x leads to a 5.5 times higher odds in outcome y

11M2. If a coefficient in a Poisson regression has value 1.7, what does this imply about the change in the outcome?

The outcome scale is probability of success, based on expected value lambda, and the log link function is used to link the outcome to the linear model
Thus, we need to exponentiate the coefficient to get some ratio change in expected value (similar to logistic regression)
Thus, coefficient of 1.7 corresponds to a 5.5 times higher expected value of y

y ~ dpois(lambda)
log(lambda) = a+b*x lambda = e^(a+b*x)

11M7. Use quap to construct a quadratic approximate posterior distribution for the chimpanzee model that includes a unique intercept for each actor, m11.4 (page 330). Compare the quadratic approximation to the posterior distribution produced instead from MCMC. Can you explain both the differences and the similarities between the approximate and the MCMC distributions? Relax the prior on the actor intercepts to Normal(0,10). Re-estimate the posterior using both ulam and quap.

Do the differences increase or decrease? Why?

```r
library(rethinking)
```

```
## Loading required package: cmdstanr
```

```
## This is cmdstanr version 0.7.1
```

```
## - CmdStanR documentation and vignettes: mc-stan.org/cmdstanr
```

```
## - CmdStan path: C:/Users/chris/OneDrive/Documents/.cmdstan/cmdstan-2.34.1
```

```
## - CmdStan version: 2.34.1
## Loading required package: posterior
```

```
## Warning: package 'posterior' was built under R version 4.2.3
```

```
## This is posterior version 1.5.0
```

```
## 
## Attaching package: 'posterior'
```

```
## The following objects are masked from 'package:stats':
## 
##     mad, sd, var
```

```
## The following objects are masked from 'package:base':
## 
##     %in%, match
```

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

```
## rethinking (Version 2.40)
```

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

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

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

```
m1 <- quap(
  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=d
)
```

```r
d_list <- list(pulled_left=d$pulled_left,
               actor=d$actor,
               treatment=d$treatment)
m2 <- 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=d_list,chains=4,cores=4
)
```

```
## Running MCMC with 4 parallel chains, with 1 thread(s) per chain...
##
## Chain 1 Iteration:   1 / 1000 [  0%]  (Warmup)
## Chain 1 Iteration: 100 / 1000 [ 10%]  (Warmup)
## Chain 2 Iteration:   1 / 1000 [  0%]  (Warmup)
## Chain 2 Iteration: 100 / 1000 [ 10%]  (Warmup)
## Chain 3 Iteration:   1 / 1000 [  0%]  (Warmup)
## Chain 3 Iteration: 100 / 1000 [ 10%]  (Warmup)
## Chain 4 Iteration:   1 / 1000 [  0%]  (Warmup)
## Chain 4 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 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 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 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 0.5 seconds.
## Chain 1 Iteration: 900 / 1000 [ 90%]  (Sampling)
## Chain 1 Iteration: 1000 / 1000 [100%]  (Sampling)
## Chain 2 Iteration: 1000 / 1000 [100%]  (Sampling)
## Chain 3 Iteration: 900 / 1000 [ 90%]  (Sampling)
## Chain 3 Iteration: 1000 / 1000 [100%]  (Sampling)
## Chain 1 finished in 0.6 seconds.
## Chain 2 finished in 0.5 seconds.
## Chain 3 finished in 0.5 seconds.
##
## All 4 chains finished successfully.
## Mean chain execution time: 0.5 seconds.
## Total execution time: 0.7 seconds.
```

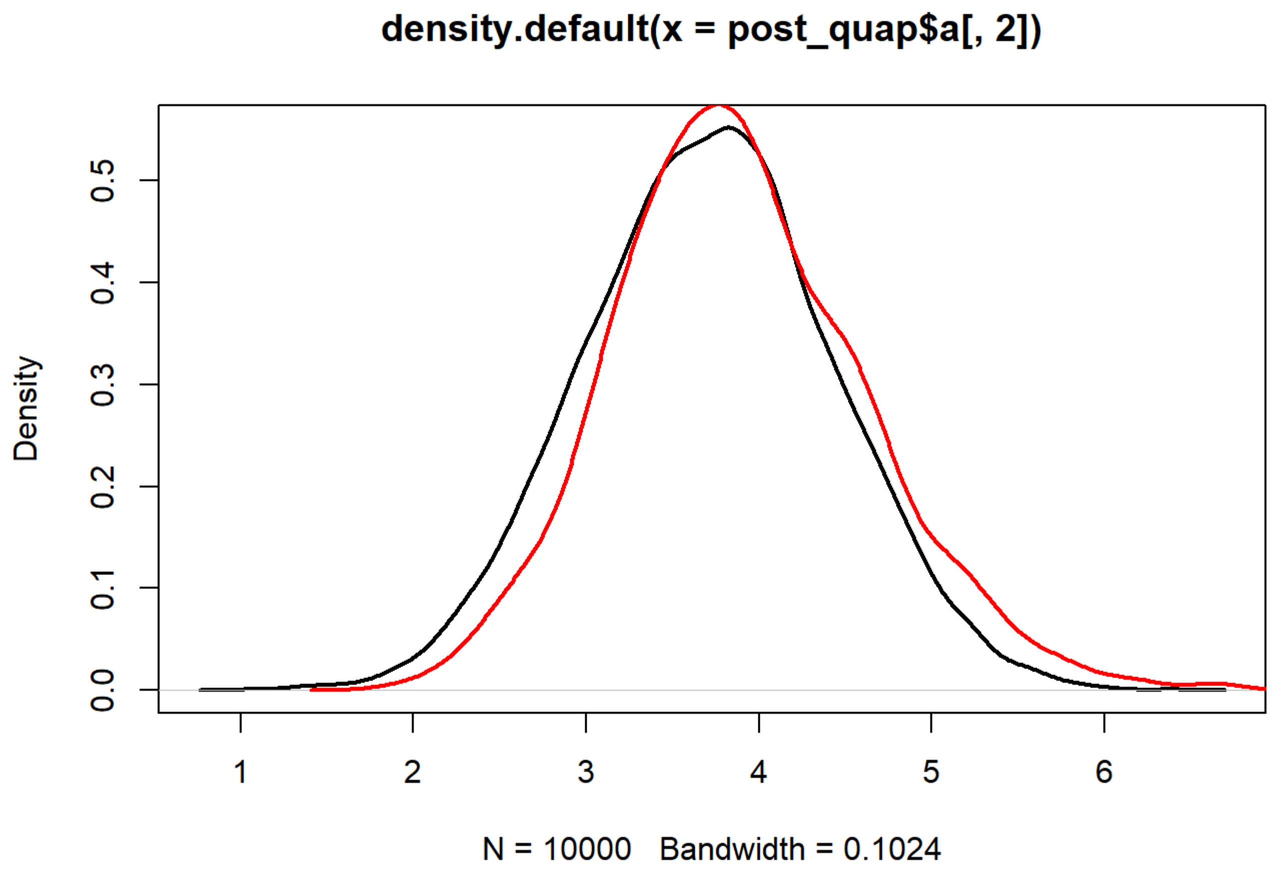
```
compare1 <- data.frame(quap_mean = precis(m1,2)$mean,
                       quap_sd = precis(m1,2)$sd,
                       mcmc_mean = precis(m2,2)$mean,
                       mcmc_sd = precis(m2,2)$sd) |> round(2)

compare1 <- cbind(parameter = rownames(precis(m1, 2)), compare1)

compare1
```

```
##    parameter quap_mean quap_sd mcmc_mean mcmc_sd
## 1       a[1]     -0.44    0.33     -0.45    0.33
## 2       a[2]      3.71    0.72      3.89    0.73
## 3       a[3]     -0.73    0.33     -0.75    0.35
## 4       a[4]     -0.73    0.33     -0.76    0.35
## 5       a[5]     -0.44    0.33     -0.45    0.34
## 6       a[6]      0.47    0.33      0.47    0.33
## 7       a[7]      1.91    0.41      1.94    0.41
## 8       b[1]     -0.04    0.28     -0.04    0.29
## 9       b[2]      0.47    0.28      0.49    0.29
## 10      b[3]     -0.38    0.29     -0.38    0.30
## 11      b[4]      0.36    0.28      0.38    0.29
```

Eyeballing the quap estimates and MCMC estimates, they seem relatively similar, given narrow priors. a[2] has the largest magnitude of difference. Can plot

```
post_quap <- extract.samples(m1)
post_mcmc <- extract.samples(m2)
```

```
plot(density(post_quap$a[,2]), lwd=2)
lines(density(post_mcmc$a[,2]), lwd=2,col="red")
```



The mcmc density plot (red) has more weight on more extreme densities (left skew), probably because mcmc has no assumed distribution?

Now with flat priors for the actors

```
m3 <- quap(
  alist(
    pulled_left ~ dbinom(1, p),
    logit(p) <- a[actor] + b[treatment],
    a[actor] ~ dnorm(0, 10),
    b[treatment] ~ dnorm(0, 0.5)
  ),
  data=d
)
```

```
m4 <- ulam(
  alist(
    pulled_left ~ dbinom(1, p),
    logit(p) <- a[actor] + b[treatment],
    a[actor] ~ dnorm(0, 10),
    b[treatment] ~ dnorm(0, 0.5)
  ),
  data=d_list,chains=4,cores=4
)
```

```
## Running MCMC with 4 parallel chains, with 1 thread(s) per chain...
##
## Chain 1 Iteration:   1 / 1000 [  0%]  (Warmup)
## Chain 1 Iteration: 100 / 1000 [ 10%]  (Warmup)
## Chain 2 Iteration:   1 / 1000 [  0%]  (Warmup)
## Chain 2 Iteration: 100 / 1000 [ 10%]  (Warmup)
## Chain 3 Iteration:   1 / 1000 [  0%]  (Warmup)
## Chain 4 Iteration:   1 / 1000 [  0%]  (Warmup)
## Chain 1 Iteration: 200 / 1000 [ 20%]  (Warmup)
## Chain 1 Iteration: 300 / 1000 [ 30%]  (Warmup)
## Chain 1 Iteration: 400 / 1000 [ 40%]  (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 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 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 1 Iteration: 500 / 1000 [ 50%]  (Warmup)
## Chain 1 Iteration: 501 / 1000 [ 50%]  (Sampling)
## Chain 1 Iteration: 600 / 1000 [ 60%]  (Sampling)
## Chain 2 Iteration: 600 / 1000 [ 60%]  (Sampling)
## Chain 3 Iteration: 600 / 1000 [ 60%]  (Sampling)
## Chain 3 Iteration: 700 / 1000 [ 70%]  (Sampling)
## Chain 4 Iteration: 500 / 1000 [ 50%]  (Warmup)
## Chain 4 Iteration: 501 / 1000 [ 50%]  (Sampling)
## Chain 4 Iteration: 600 / 1000 [ 60%]  (Sampling)
## Chain 1 Iteration: 700 / 1000 [ 70%]  (Sampling)
## Chain 2 Iteration: 700 / 1000 [ 70%]  (Sampling)
## Chain 3 Iteration: 800 / 1000 [ 80%]  (Sampling)
## Chain 3 Iteration: 900 / 1000 [ 90%]  (Sampling)
## Chain 4 Iteration: 700 / 1000 [ 70%]  (Sampling)
## Chain 1 Iteration: 800 / 1000 [ 80%]  (Sampling)
## Chain 2 Iteration: 800 / 1000 [ 80%]  (Sampling)
## Chain 3 Iteration: 1000 / 1000 [100%]  (Sampling)
## Chain 4 Iteration: 800 / 1000 [ 80%]  (Sampling)
## Chain 4 Iteration: 900 / 1000 [ 90%]  (Sampling)
## Chain 3 finished in 0.7 seconds.
## Chain 1 Iteration: 900 / 1000 [ 90%]  (Sampling)
## Chain 1 Iteration: 1000 / 1000 [100%]  (Sampling)
## Chain 2 Iteration: 900 / 1000 [ 90%]  (Sampling)
## Chain 4 Iteration: 1000 / 1000 [100%]  (Sampling)
## Chain 1 finished in 0.9 seconds.
## Chain 4 finished in 0.8 seconds.
## Chain 2 Iteration: 1000 / 1000 [100%]  (Sampling)
## Chain 2 finished in 0.9 seconds.
##
## All 4 chains finished successfully.
## Mean chain execution time: 0.8 seconds.
## Total execution time: 1.1 seconds.
```

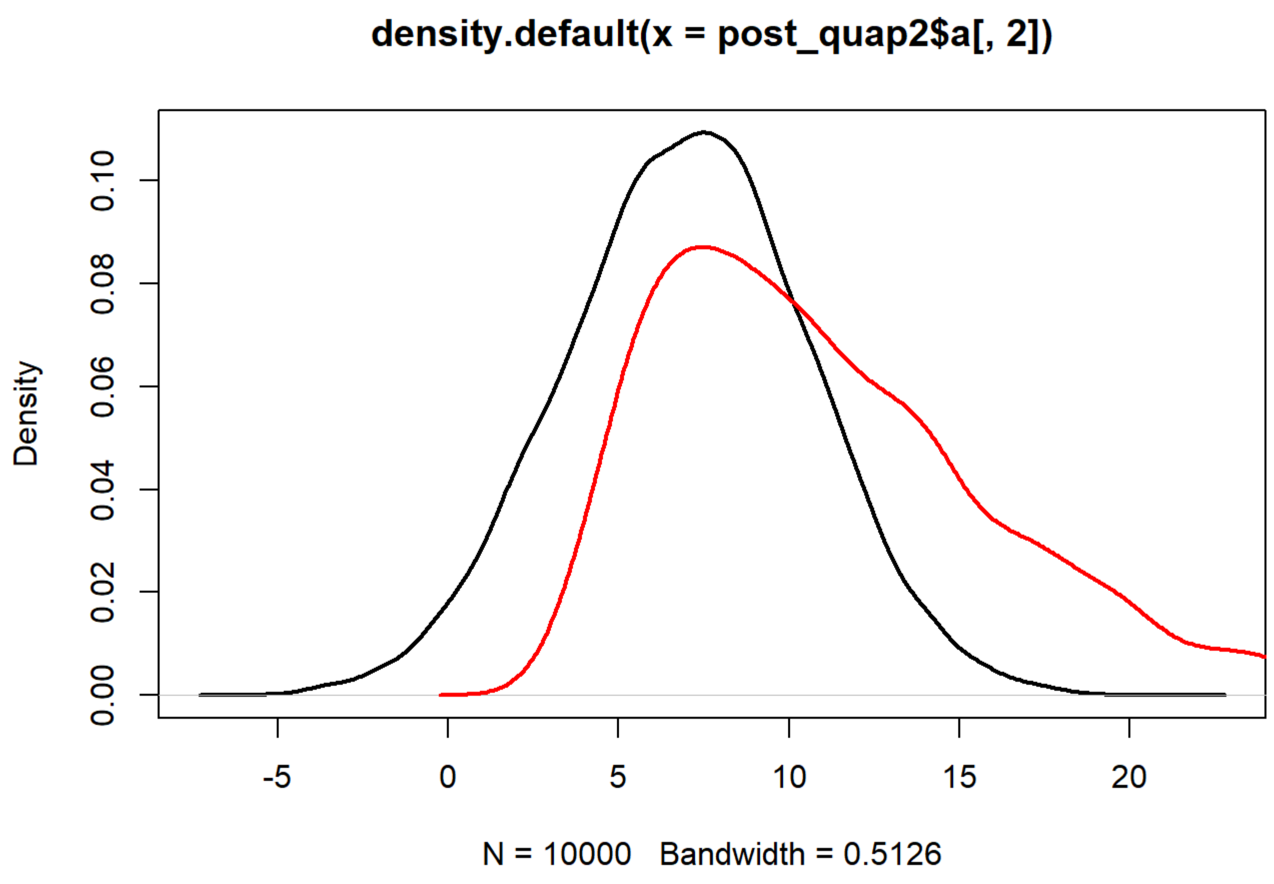
```
compare2 <- data.frame(quap_mean = precis(m3,2)$mean,
                       quap_sd = precis(m3,2)$sd,
                       mcmc_mean = precis(m4,2)$mean,
                       mcmc_sd = precis(m4,2)$sd) |> round(2)

compare2 <- cbind(parameter = rownames(precis(m3, 2)), compare1)

compare2
```

```
##    parameter parameter quap_mean quap_sd mcmc_mean mcmc_sd
## 1       a[1]      a[1]     -0.44    0.33     -0.45    0.33
## 2       a[2]      a[2]      3.71    0.72      3.89    0.73
## 3       a[3]      a[3]     -0.73    0.33     -0.75    0.35
## 4       a[4]      a[4]     -0.73    0.33     -0.76    0.35
## 5       a[5]      a[5]     -0.44    0.33     -0.45    0.34
## 6       a[6]      a[6]      0.47    0.33      0.47    0.33
## 7       a[7]      a[7]      1.91    0.41      1.94    0.41
## 8       b[1]      b[1]     -0.04    0.28     -0.04    0.29
## 9       b[2]      b[2]      0.47    0.28      0.49    0.29
## 10      b[3]      b[3]     -0.38    0.29     -0.38    0.30
## 11      b[4]      b[4]      0.36    0.28      0.38    0.29
```

The only noticeably different estimates are for a[2] again

```
post_quap2 <- extract.samples(m3)
post_mcmc2 <- extract.samples(m4)
```

```
plot(density(post_quap2$a[,2]), lwd=2)
lines(density(post_mcmc2$a[,2]), lwd=2,col="red")
```



Without regularizing (narrow priors), the data is allowed to dominate the posterior, and we still a very non-normal distribution with the MCMC posterior

---

H2

```
rm(list=ls())
library(MASS)
data("eagles"); d <- eagles; rm(eagles)

d$P <- ifelse(d$P == "L", 1, 0)
d$A <- ifelse(d$A == "A", 1, 0)
d$V <- ifelse(d$V == "L", 1, 0)
```

```
m1 <- quap(
  alist(
    y ~ dbinom(n, p),
    logit(p) <- a + P*bP + A*bA + V*bV,
    a ~ dnorm(0, 1.5),
    c(bP, bA, bV) ~ dnorm(0, 0.5)
  ), data=d
)
```

```
m2 <- ulam(
  alist(
    y ~ dbinom(n, p),
    logit(p) <- a + P*bP + A*bA + V*bV,
    a ~ dnorm(0, 1.5),
    c(bP, bA, bV) ~ dnorm(0, 0.5)
  ),data=d, chains=4
)
```

```
## 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 0.1 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 0.1 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 0.1 seconds.
##
## All 4 chains finished successfully.
## Mean chain execution time: 0.1 seconds.
## Total execution time: 0.6 seconds.
```

```
precis(m1)
```

```
##          mean        sd       5.5%      94.5%
## a   0.2936066 0.3681378 -0.2947487  0.8819618
## bP  1.6211229 0.3063650  1.1314925  2.1107534
## bA  0.6517834 0.3054247  0.1636558  1.1399111
## bV -1.6730560 0.3191365 -2.1830978 -1.1630142
```

```
precis(m2)
```

```
##          mean        sd       5.5%      94.5%     rhat  ess_bulk
## a   0.2870354 0.3538319 -0.2649727  0.8625281 1.004131  920.4897
## bV -1.6812698 0.3068709 -2.1822608 -1.1909030 1.006575  989.7566
## bA  0.6637550 0.2923481  0.1841378  1.1286550 1.000291 1221.8875
## bP  1.6455095 0.3117081  1.1373667  2.1243272 1.001843 1223.0086
```

a. Pretty much identical posterior means and CIs for quap and mcmc estimates.

b. The estimates are on the scale of log-odds.

The mean estimate for the intercept is not reliability different from zero. This can be interpreted as the average probability of a small-body non-adult pirate on a small-body victim is around 50/50.

To interpret the other estimates we should extract samples from the posterior and convert to the outcome scale. We will use the quap estimates since they're the same

```
post <- extract.samples(m1)
```

Ugliest for loop of all time

```
posterior <- list()
for(i in 1:8){
  a <- as.matrix(post)
  b <- as.matrix(d[i, 3:5])
  c <- matrix(0, ncol=4, nrow=nrow(post))
    for(v in 1:3){
    c[, 1] <- a[, 1]
    c[, (v+1)] <- a[, (v+1)] * b[, v]
    }
  posterior[[i]] <- inv_logit(rowSums(c))
}
```

```
estimates <- matrix(0, ncol=2, nrow=nrow(d))
colnames(estimates) <- c("means", "sd")
for(i in 1:8){
  estimates[i, 1] <- mean(posterior[[i]])
}

for(i in 1:8){
  estimates[i, 2] <- sd(posterior[[i]])
}

estimates <- as.data.frame(estimates) |> round(2)
```

PROBABILITY OF SUCCESS ESTIMATES FOR EACH COMBINATION OF P + A + V

```
estimates <- cbind(d[, 3:5], estimates)
estimates
```

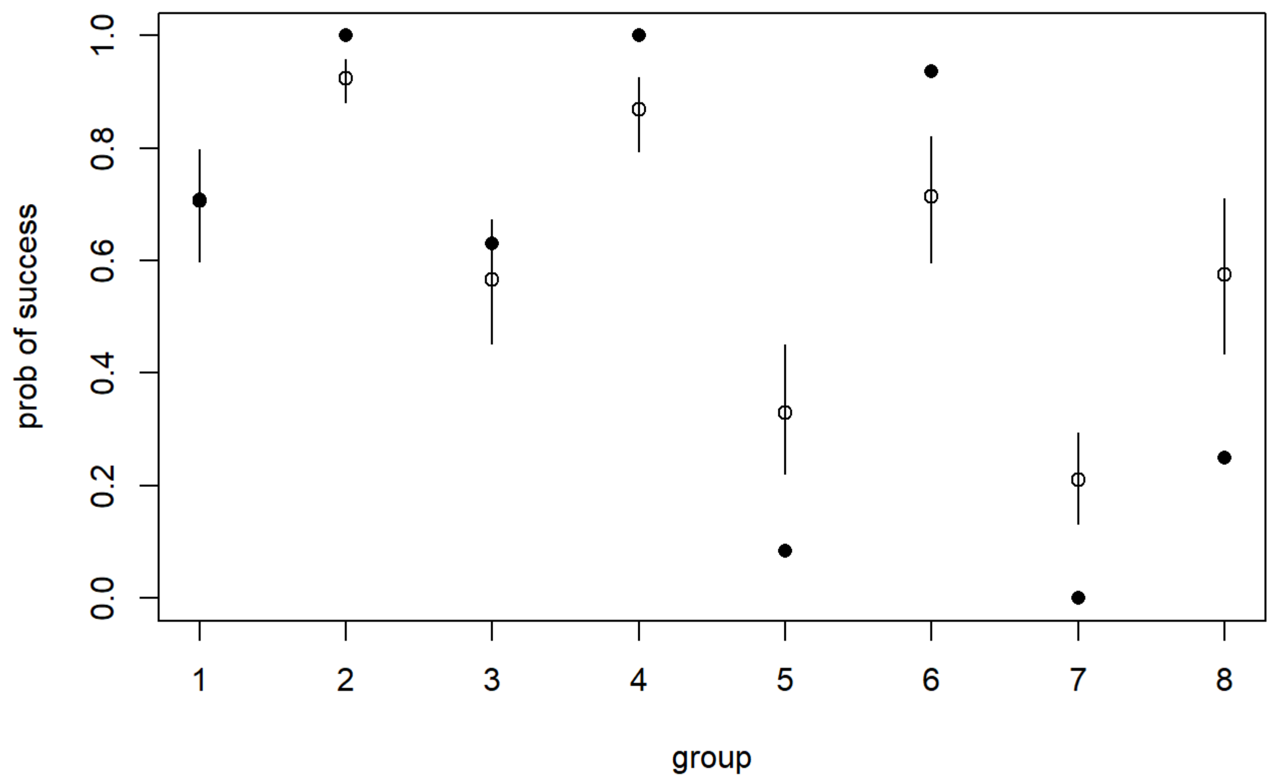
```
##   P A V means   sd
## 1 1 1 1  0.71 0.06
## 2 1 1 0  0.93 0.02
## 3 1 0 1  0.56 0.07
## 4 1 0 0  0.87 0.04
## 5 0 1 1  0.33 0.07
## 6 0 1 0  0.71 0.07
## 7 0 0 1  0.21 0.05
## 8 0 0 0  0.57 0.09
```

Large-body adult pirates on small-body victims have highest probability of success (p=0.92), and the inverse has the smallest (p=0.21). This makes sense.

Now plotting posterior predictions

```
postpred <- link(m1)
means <- apply(postpred, 2, mean)
ci <- apply(postpred, 2, PI)
```

```
plot(d$y/d$n , ylab="prob of success" , xlab="group" , xaxt="n" , xlim=c(1,8) , pch=16 )
axis(1 , at=1:8 ,labels=c(1:8) )
points(1:8, means)
for(i in 1:8){
  lines(c(i,i) , ci[,i] )
}
```



c. Add interaction between pirate size and age and compare with WAIC

```
m3 <- quap(
  alist(
    y ~ dbinom(n, p),
    logit(p) <- a + P*bP + A*bA + V*bV + bPA*P*A,
    a ~ dnorm(0, 1.5),
    c(bP, bA, bV, bPA) ~ dnorm(0, 0.5)
  ), data=d
)
```

```
precis(m3)
```

```
##            mean        sd        5.5%      94.5%
## a     0.3248780 0.3692525 -0.26525880  0.9150148
## bP    1.5393874 0.3243977  1.02093725  2.0578375
## bA    0.5489343 0.3340237  0.01509987  1.0827686
## bV   -1.6860295 0.3205111 -2.19826817 -1.1737909
## bPA   0.2933414 0.3807134 -0.31511222  0.9017949
```

```
compare(m1, m3, func=WAIC)
```

```
##        WAIC       SE     dWAIC      dSE    pWAIC    weight
## m1 60.52459 12.17193 0.0000000       NA 8.906066 0.5983016
## m3 61.32138 11.97554 0.7967869 1.424824 9.080314 0.4016984
```

No difference, there does not seem to be a meaningful interaction between pirate age and pirate size