Assignment 4

Jasmin Capka

11/28/2021

Exercise 1

To create the plot of the posterior predictions of the Chimpanzee experiment on the bottom panel of Figure 11.4 on page 333, I decided to first understand the code for the plot in the top panel. The following code is copied from chapter 11 of the book to be able to create the plots:

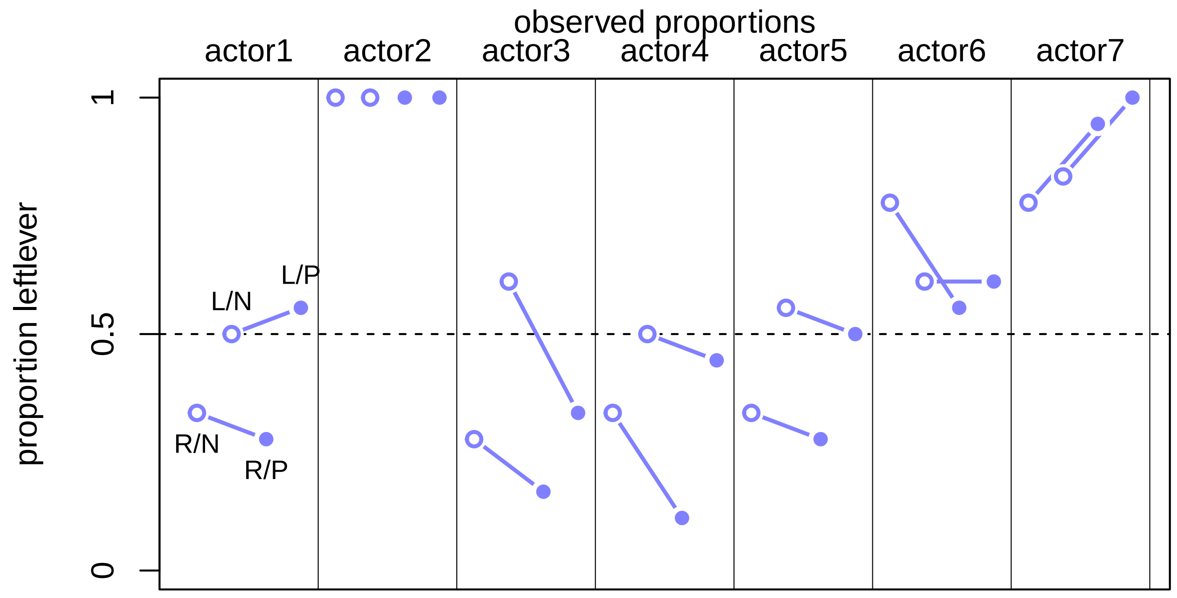
```r
# load data
data(chimpanzees)
d <- chimpanzees

# create treatment variable
d$treatment <- 1 + d$prosoc_left + 2 * d$condition

pl <- by(d$pulled_left, list(d$actor, d$treatment), mean)

# trimmed data list
dat_list <- list(
  pulled_left = d$pulled_left,
  actor = d$actor,
  treatment = as.integer(d$treatment)
)

# fit model
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
```

Then I took the code that was given to visualize the plot in the top panel and tried to understand it:

```r
# create basis graph consisting of x and y axis
plot(NULL, xlim = c(1, 28), ylim = c(0, 1), xlab = "",
     ylab = "proportion leftlever", xaxt = "n", yaxt = "n")

# define y axis
```

```r
axis(2, at = c(0, 0.5, 1), labels = c(0, 0.5, 1))
# plot horizontal line at 0.5
abline(h = 0.5, lty = 2)

# divide the x-axis in 7 parts for 7 actors and label those
for (j in 1:7) abline(v = (j - 1) * 4 + 4.5, lwd = 0.5)
for (j in 1:7) text((j - 1) * 4 + 2.5, 1.1, concat("actor", j), xpd = TRUE)

# print two lines for each actor except the second one
for (j in(1:7)[-2]) {
  # takes the information about the lines from pl
  lines((j - 1) * 4 + c(1, 3), pl[j, c(1, 3)], lwd = 2, col = rangi2)
  lines((j - 1) * 4 + c(2, 4), pl[j, c(2, 4)], lwd = 2, col = rangi2)
}

# plots the points at the end of each of the lines (information from pl)
points(1:28, t(pl), pch = 16, col = "white", cex = 1.7)
points(1:28, t(pl), pch = c(1, 1, 16, 16), col = rangi2, lwd = 2)

# labels the first four points
yoff <- 0.01
text(1, pl[1, 1] - yoff, "R/N", pos = 1, cex = 0.8)
text(2, pl[1, 2] + yoff, "L/N", pos = 3, cex = 0.8)
text(3, pl[1, 3] - yoff, "R/P", pos = 1, cex = 0.8)
text(4, pl[1, 4] + yoff, "L/P", pos = 3, cex = 0.8)

# sets title of plot
mtext("observed proportions\n")
```



The following code was given to compute posterior function in the book. This is needed to create the plot on the bottom panel of Figure 11.4.

```r
dat <- list(actor = rep(1:7, each = 4), treatment = rep(1:4, times = 7))
p_post <- link(m11.4, data = dat)
p_mu <- apply(p_post, 2, mean)
```

```r
p_ci <- apply(p_post, 2, PI)
```

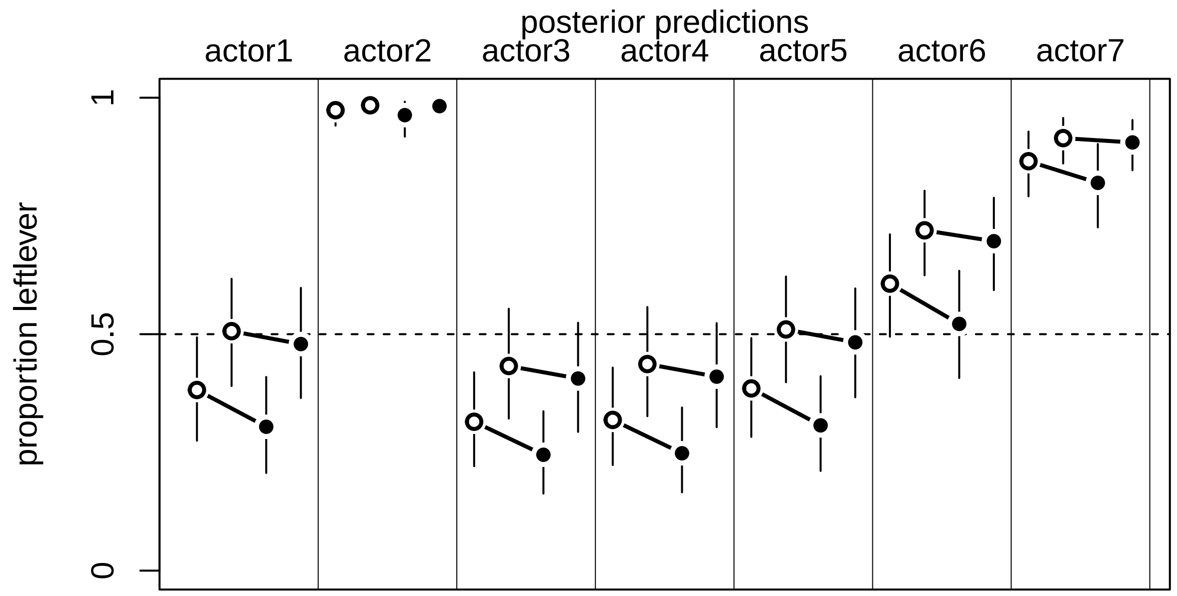
Then I tried to adjust the code from the plot above:

- The first part that creates the axis and the separation for different actors was copied without adjustment.
- Then I changed the title to the title that was visible in the plot in the bottom panel.
- The values for the points and the lines in the upper plot were stored in the variable *pl*, which has a shape of (7, 4). For the posterior, all values are in *p_mu* which consists of one row with 28 values. Nevertheless, it follows the same logic as *pl* when splitting those 28 values in 7 parts, containing 4 values each. Those 4 values in each part of *p_mu* can be treated the same as the 4 values in each of the 7 rows of *pl* when plotting the posterior.
- Therefore, in order to draw the connecting lines, for each of the seven parts of *p_mu*, the first and third value and the second and forth value are connected, following the logic of the upper plot, still without plotting them for actor 2.
- To draw the points the code from the to plot could be copied almost completely. Only $t(pl)$ was exchanged with *p_mu*.
- Additionally, for all points and lines, the color setting to blue was removed to draw it in black.
- The last thing to add are the CI lines. The information about the CI intervals are stored in *p_ci*, following a similar structure than *p_mu* only having two rows instead of one. Therefore, I adapted the for-loop of the connecting lines only this time including actor two. Then for each of the seven parts, we can split *p_ci* into, four CI lines are drawn. For each of the four columns, connecting the value from the first row with the one from the second row. The line width is set to the default value to make the lines a bit thinner as can be seen in the bottom plot of Figure 11.4.
- In the end I ordered the code to first draw the connecting lines, then the CI lines and in the end the points to achieve the same overlapping of those as in the given plot.

This is the result:

```r
# adopted as is
plot(NULL, xlim = c(1, 28), ylim = c(0, 1), xlab = "",
     ylab = "proportion leftlever", xaxt = "n", yaxt = "n")
axis(2, at = c(0, 0.5, 1), labels = c(0, 0.5, 1))
abline(h = 0.5, lty = 2)
for (j in 1:7) abline(v = (j - 1) * 4 + 4.5, lwd = 0.5)
for (j in 1:7) text((j - 1) * 4 + 2.5, 1.1, concat("actor", j), xpd = TRUE)

# change title
mtext("posterior predictions\n")

# draw connecting lines
for (j in(1:7)[-2]) {
  lines((j - 1) * 4 + c(1, 3), c(p_mu[j * 4 - 3], p_mu[j * 4 - 1]), lwd = 2)
  lines((j - 1) * 4 + c(2, 4), c(p_mu[j * 4 - 2], p_mu[j * 4 - 0]), lwd = 2)
}

# draw ci lines
for (j in(1:7)) {
  lines((j - 1) * 4 + c(1, 1), p_ci[c(1, 2), j * 4 - 3])
  lines((j - 1) * 4 + c(2, 2), p_ci[c(1, 2), j * 4 - 2])
  lines((j - 1) * 4 + c(3, 3), p_ci[c(1, 2), j * 4 - 1])
  lines((j - 1) * 4 + c(4, 4), p_ci[c(1, 2), j * 4 - 0])
}

# set points
```

```
points(1:28, p_mu, pch = 16, col = "white", cex = 1.7)
points(1:28, p_mu, pch = c(1, 1, 16, 16), lwd = 2)
```



Exercise 2

First I imported the data:

```
data(Wines2012)
d <- Wines2012
precis(d)
```

```
##                  mean        sd 5.5% 94.5%     histogram
## judge             NaN        NA   NA    NA
## flight            NaN        NA   NA    NA
## wine              NaN        NA   NA    NA
## score      14.2000000 2.6639535   10    18
## wine.amer   0.6000000 0.4912645    0     1
## judge.amer  0.5555556 0.4982901    0     1
```

Then I took the provided code from the exercise to standardize and create IDs to distinguish between American and French judges and American and French wines:

```
# standardize scores and create IDs for
# American / French judges and wines
dat_list <- list(
  S = standardize(d$score),
  jid = as.integer(d$judge),
  wid = as.integer(d$wine)
)
str(dat_list)
```

```
## List of 3
##  $ S  : num [1:180] -1.5766 -0.4505 -0.0751 0.3003 -2.3274 ...
##   ..- attr(*, "scaled:center")= num 14.2
##   ..- attr(*, "scaled:scale")= num 2.66
##  $ jid: int [1:180] 4 4 4 4 4 4 4 4 4 4 ...
##  $ wid: int [1:180] 1 3 5 7 9 11 13 15 17 19 ...
```

Part 2.a)

Side note: I interpreted the exercise 2.a) in such a way, that I was already supposed to construct a linear regression model with quap. In hindsight, I realized that I might have been supposed to only do preparations under 2.a) and train the one ulam model. As I already did this quap model, I decided to leave it in. The ulam model can be found beneath at section 2.c).

I constructed a linear regression model with index variables. `judge` and `wine` in their representation as `jid` and `wid` are already index variables. Each judge and wine gets its individual parameter $\beta Ji[jid]$ or $\beta Wi[wid]$.

```r
m2.1 <- quap(
  alist(
    S ~ dnorm(mu, sigma),
    mu <- bJi[jid] + bWi[wid],
    bJi[jid] ~ dnorm(0, 0.5),
    bWi[wid] ~ dnorm(0, 0.5),
    sigma ~ dexp(1)
  ),
  data = dat_list,
)
precis(m2.1, depth = 2)
```

```
##                  mean         sd        5.5%       94.5%
## bJi[1]  -0.283984898 0.18589977 -0.58108864  0.01311885
## bJi[2]   0.217287570 0.18588748 -0.07979653  0.51437166
## bJi[3]   0.208866826 0.18588626 -0.08821532  0.50594897
## bJi[4]  -0.551292909 0.18598157 -0.84852738 -0.25405844
## bJi[5]   0.810389594 0.18611053  0.51294902  1.10783017
## bJi[6]   0.484590657 0.18595617  0.18739678  0.78178454
## bJi[7]   0.133740887 0.18587677 -0.16332609  0.43080786
## bJi[8]  -0.668227209 0.18603386 -0.96554525 -0.37090917
## bJi[9]  -0.350815631 0.18591531 -0.64794420 -0.05368706
## bWi[1]   0.121004435 0.24509155 -0.27069921  0.51270808
## bWi[2]   0.088301859 0.24508421 -0.30339004  0.47999376
## bWi[3]   0.235647868 0.24513504 -0.15612528  0.62742101
## bWi[4]   0.481096070 0.24532274  0.08902295  0.87316919
## bWi[5]  -0.108079063 0.24508830 -0.49977751  0.28361938
## bWi[6]  -0.320859008 0.24518553 -0.71271284  0.07099482
## bWi[7]   0.251967242 0.24514364 -0.13981965  0.64375413
## bWi[8]   0.235618999 0.24513510 -0.15615423  0.62739223
## bWi[9]   0.071943025 0.24508140 -0.31974439  0.46363044
## bWi[10]  0.104630728 0.24508761 -0.28706661  0.49632807
## bWi[11] -0.009802131 0.24507597 -0.40148087  0.38187661
## bWi[12] -0.026273075 0.24507659 -0.41795280  0.36540665
## bWi[13] -0.091684182 0.24508484 -0.48337709  0.30000872
## bWi[14]  0.006521211 0.24507591 -0.38515742  0.39819985
## bWi[15] -0.189891464 0.24511432 -0.58163149  0.20184856
## bWi[16] -0.173497389 0.24510802 -0.56522734  0.21823256
## bWi[17] -0.124401994 0.24509240 -0.51610699  0.26730300
## bWi[18] -0.746283775 0.24566963 -1.13891129 -0.35365626
## bWi[19] -0.140814097 0.24509698 -0.53252641  0.25089822
## bWi[20]  0.333812556 0.24519474 -0.05805599  0.72568110
## sigma    0.785740924 0.04174314  0.71902732  0.85245453
```

As can be seen, there are 9 `judge` parameters and 20 `wine` parameters.

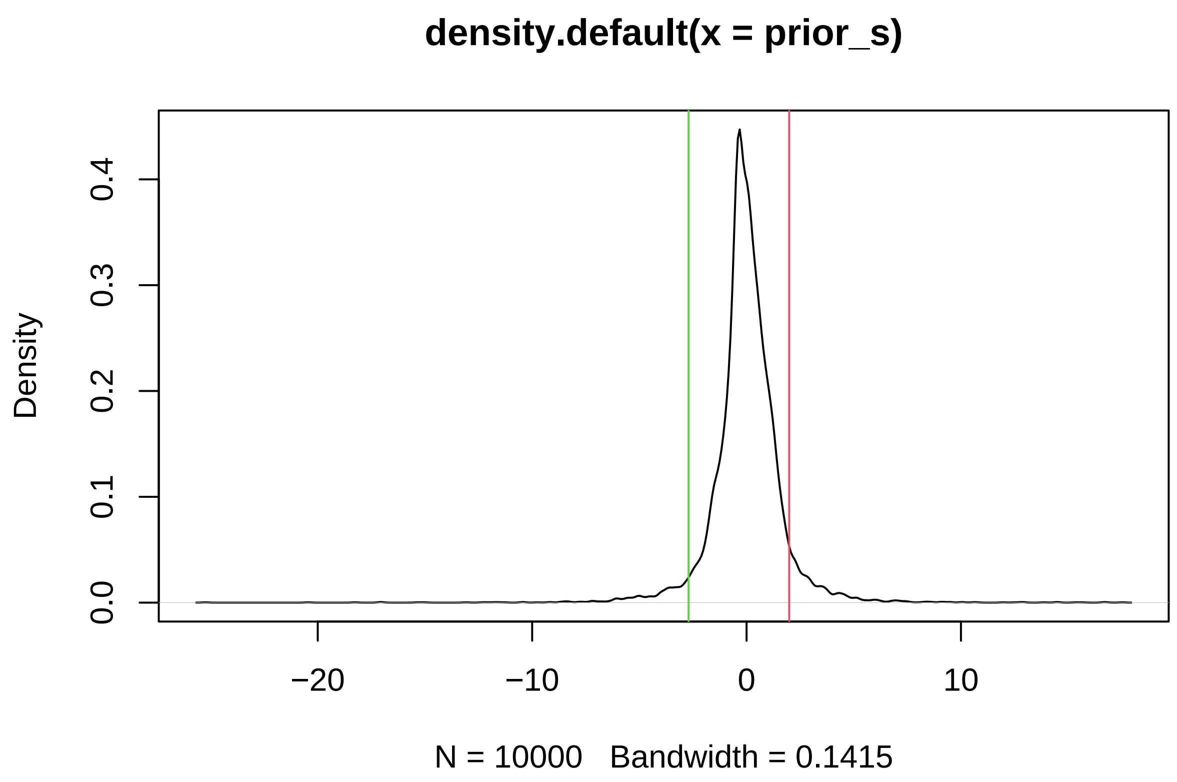
Part 2.b)

I used the same priors for the parameters $\beta Ji[jid]$ and $\beta Wi[wid]$:

$$\beta Ji[jid] \sim Normal(0, 0.5)$$
$$\beta Wi[wid] \sim Normal(0, 0.5)$$

The score S is standardized with a mean of 0 and a standard deviation of 1, meaning 95% of values are between 2 and -2. Therefore, I used normal distributions with a mean of 0 and a standard deviation of 0.5 as priors. This means when adding both, the mean values are added and the variances are added, resulting in a normal distribution with a mean of 0 and a standard deviation of 1. This is approximately the distribution of S. Lokking at the sum of these priors, 95% of values are between 2 and -2. The density plot of the prior simulation can be seen below with the min and max value of the standardized score S represented by a green and red line respectively:

```r
N = 180

# define distribution of a and b
bJi <- rnorm(N, 0, 0.5)
bWi <- rnorm(N, 0, 0.5)

# sigma and mu
sample_sigma <- rexp(N, 1)
sample_mu <- bJi + bWi

# plot prior predictive simulation
prior_s <- rnorm(1e4, sample_mu, sample_sigma)
plot(density(prior_s))

# plot min and max line
abline(v = max(dat_list$S), col = 2)
abline(v = min(dat_list$S), col = 3)
```



I did not include an intercept α in the model as the mean value of the standardized score S is 0, which means that the intercept should be zero as well.

Part 2.c)

This is the model with `ulam` instead of `quap`:

```
m2.3 <- ulam(
  alist(
    S ~ dnorm(mu, sigma),
    mu <- bJi[jid] + bWi[wid],
    bJi[jid] ~ dnorm(0, 0.5),
    bWi[wid] ~ dnorm(0, 0.5),
    sigma ~ dexp(1)
  ),
  data = dat_list, chains = 4, cores = 4
)
```

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

```
precis(m2.3, depth = 2)
```

```
##                 mean         sd        5.5%       94.5%    n_eff     Rhat4
## bJi[1]  -0.278309561 0.19696735 -0.58788082  0.04368315 2774.261 0.9993111
## bJi[2]   0.205327804 0.19963110 -0.11096636  0.52211027 2849.549 0.9993790
## bJi[3]   0.199908607 0.19110925 -0.09263340  0.51062696 2352.701 0.9992492
## bJi[4]  -0.543883651 0.19642620 -0.85132285 -0.22913783 2411.195 1.0000450
## bJi[5]   0.795485436 0.19462601  0.48241986  1.09302297 2582.156 0.9987509
## bJi[6]   0.467540638 0.19519764  0.15372207  0.76875899 2457.373 0.9987262
## bJi[7]   0.124800808 0.19145699 -0.19298370  0.43419498 2408.710 1.0004641
## bJi[8]  -0.665345519 0.19634242 -0.97270803 -0.34269037 2451.091 0.9995529
## bJi[9]  -0.358093341 0.18674962 -0.64791719 -0.05767710 2155.781 0.9994498
## bWi[1]   0.119573614 0.25553055 -0.30326189  0.51059382 2838.025 0.9999840
## bWi[2]   0.090241485 0.26857268 -0.32662411  0.52819311 3851.711 0.9986427
## bWi[3]   0.233093274 0.24642602 -0.16616173  0.63533615 2429.831 0.9992442
## bWi[4]   0.469336140 0.24030641  0.07735774  0.85866635 3205.480 1.0000740
## bWi[5]  -0.098835460 0.26525350 -0.52774210  0.32684679 3002.713 0.9986214
## bWi[6]  -0.303726769 0.24765622 -0.69628801  0.08340660 3163.795 0.9992789
## bWi[7]   0.249025253 0.26085821 -0.15574065  0.66843604 3366.800 0.9984768
## bWi[8]   0.228594258 0.25674032 -0.19363006  0.63241711 3163.423 0.9986336
## bWi[9]   0.074731682 0.25467416 -0.33996823  0.47725954 2882.547 1.0008027
## bWi[10]  0.110789170 0.25403995 -0.30076507  0.51781415 3519.139 1.0000120
## bWi[11] -0.004533708 0.26371542 -0.42761638  0.40819241 4027.110 0.9986613
## bWi[12] -0.014212297 0.25394660 -0.41279843  0.39396086 3193.788 0.9988272
## bWi[13] -0.079224448 0.25240536 -0.48223034  0.34139157 3296.817 0.9990937
## bWi[14]  0.015565979 0.26828304 -0.42214636  0.43737692 3031.374 0.9988593
## bWi[15] -0.177999512 0.25924059 -0.60588845  0.24028754 3084.242 0.9989882
## bWi[16] -0.165053718 0.26305534 -0.58457875  0.25194245 3827.260 0.9989719
## bWi[17] -0.120030250 0.26104528 -0.53913176  0.30665763 3294.314 0.9996794
## bWi[18] -0.717245090 0.25285401 -1.11540548 -0.32249902 3424.941 0.9990878
## bWi[19] -0.124885160 0.26138733 -0.53266959  0.29259623 3075.204 0.9985004
## bWi[20]  0.328571683 0.26685881 -0.08008723  0.74897858 3429.890 0.9997276
## sigma    0.846791146 0.04672328  0.77401656  0.92177587 3061.552 0.9997578
```

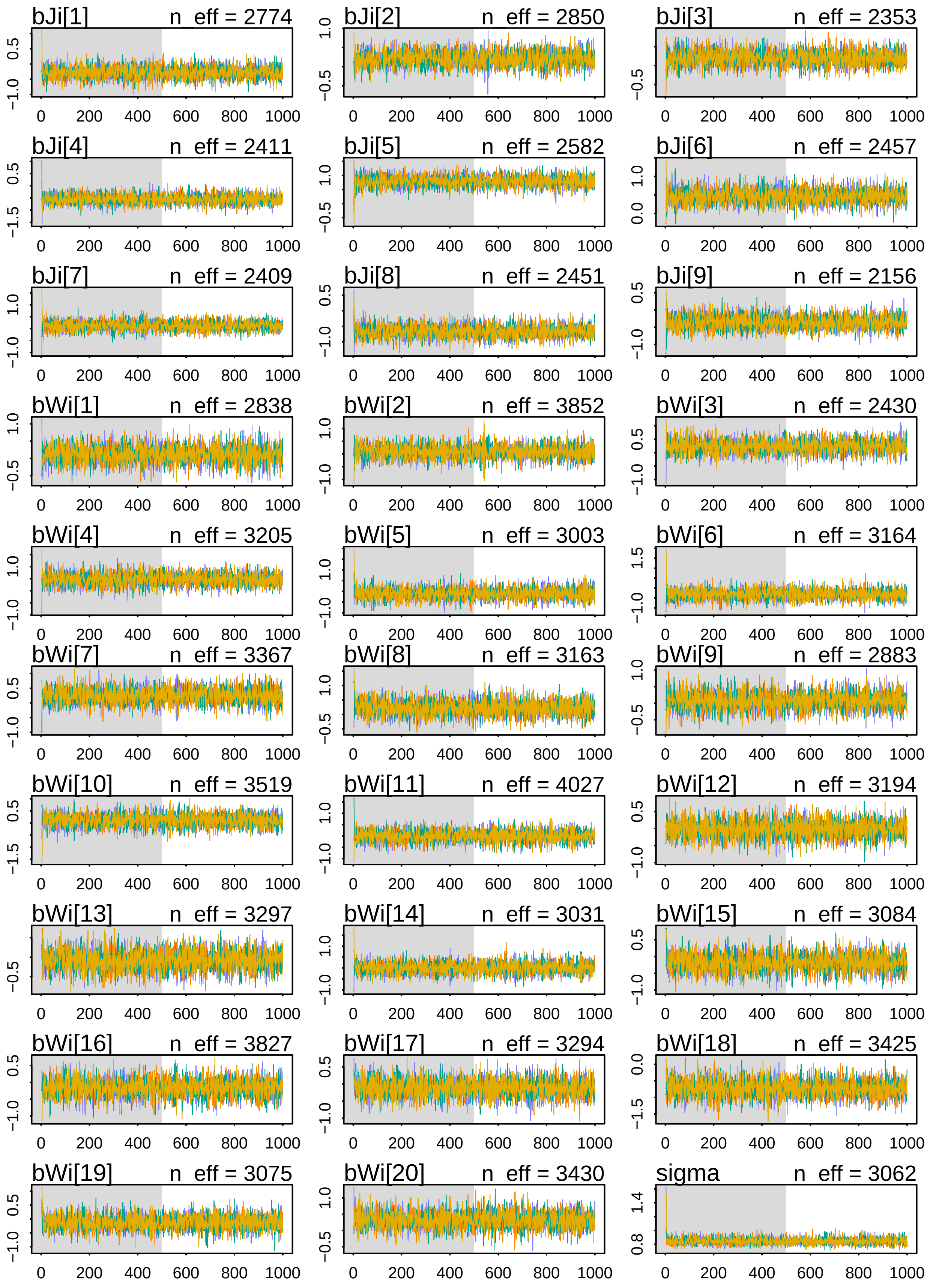
Part 2.d)

In order to check the chains for convergence, I use the `traceplot` and `trankplot` functions:

```r
traceplot(m2.3)
```

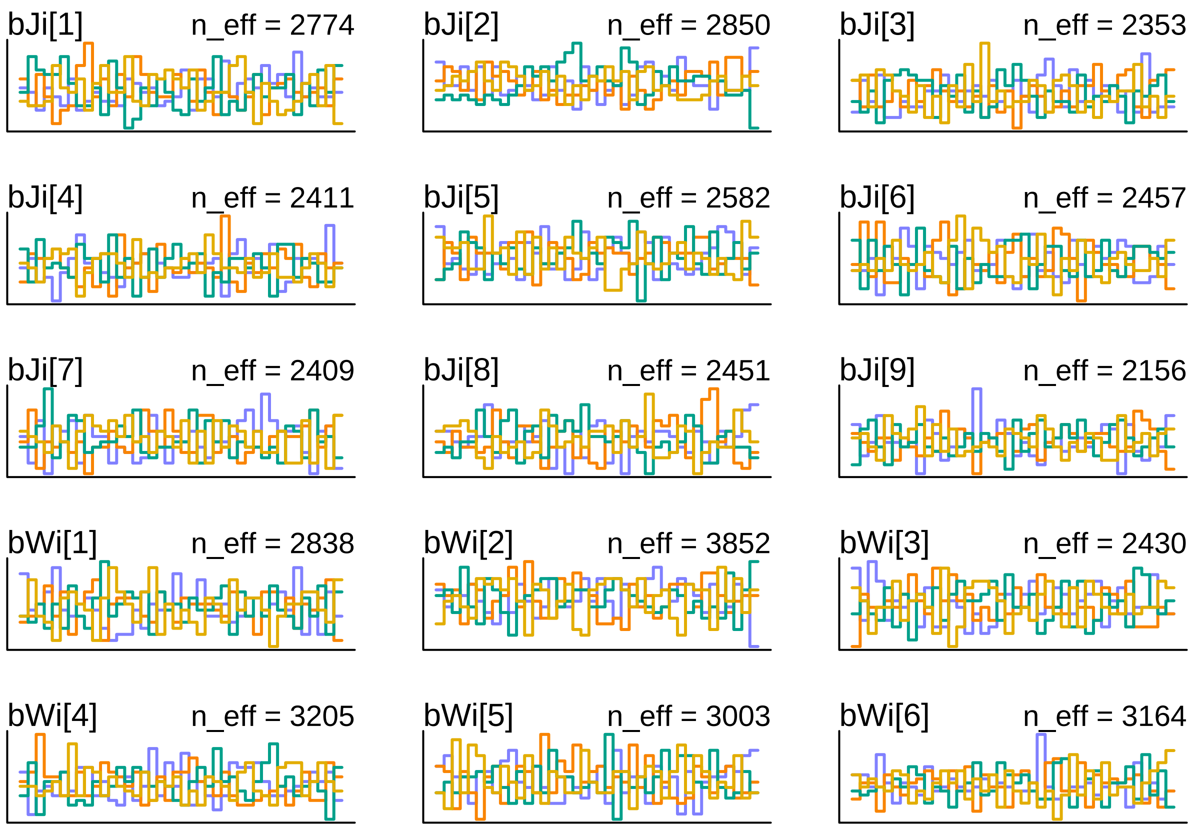
```
## [1] 1000
## [1] 1
## [1] 1000
```

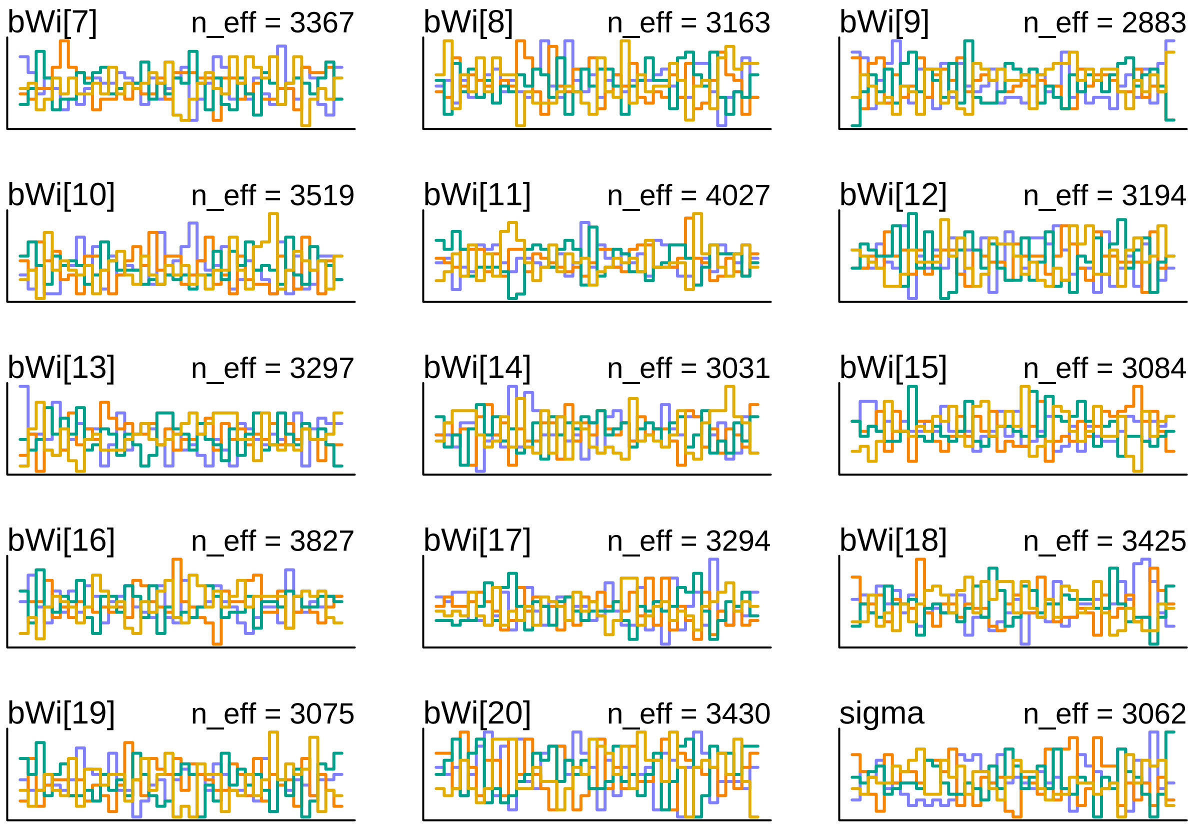
```
## Waiting to draw page 2 of 2
```


bJi[1] n_eff = 2774
bJi[2] n_eff = 2850
bJi[3] n_eff = 2353
bJi[4] n_eff = 2411
bJi[5] n_eff = 2582
bJi[6] n_eff = 2457
bJi[7] n_eff = 2409
bJi[8] n_eff = 2451
bJi[9] n_eff = 2156
bWi[1] n_eff = 2838
bWi[2] n_eff = 3852
bWi[3] n_eff = 2430
bWi[4] n_eff = 3205
bWi[5] n_eff = 3003
bWi[6] n_eff = 3164
bWi[7] n_eff = 3367
bWi[8] n_eff = 3163
bWi[9] n_eff = 2883
bWi[10] n_eff = 3519
bWi[11] n_eff = 4027
bWi[12] n_eff = 3194
bWi[13] n_eff = 3297
bWi[14] n_eff = 3031
bWi[15] n_eff = 3084
bWi[16] n_eff = 3827
bWi[17] n_eff = 3294
bWi[18] n_eff = 3425
bWi[19] n_eff = 3075
bWi[20] n_eff = 3430
sigma n_eff = 3062
0 200 400 600 800 1000

```
trankplot(m2.3)
```

```
## Waiting to draw page 2 of 2
```



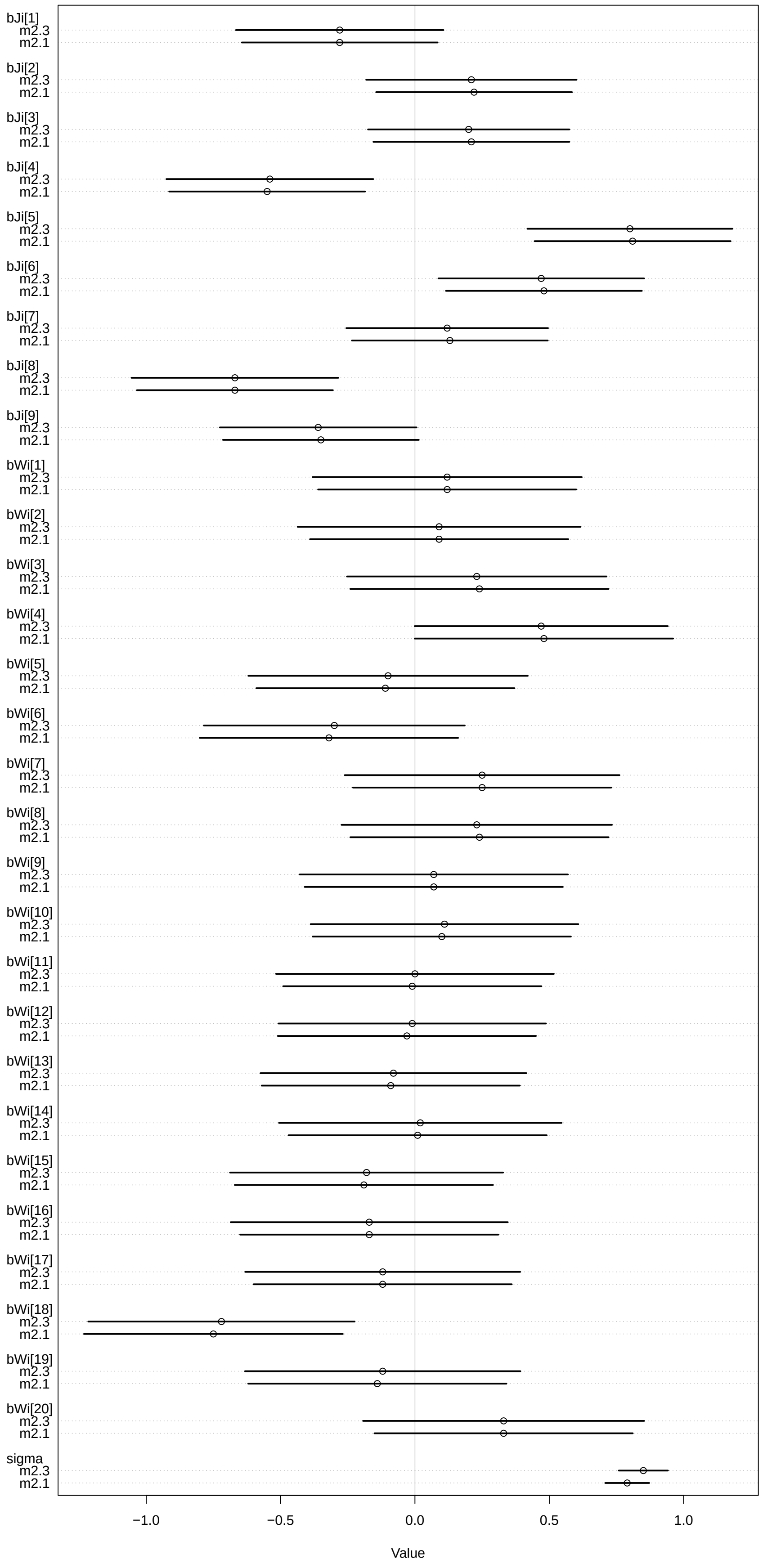


In the traceplot, the four independent chains stick around the same region of high probability. In the trankplot, the histograms overlap and are within the same range. Therefore, the chains are converging. Another indicator for the convergence is that the Rhat values are all 1.

Further questions:

To further inspect the implications of the parameters, I plotted them for both models:

```
plot(coeftab(m2.1, m2.3))
```


bJi[1]
m2.3
m2.1
bJi[2]
m2.3
m2.1
bJi[3]
m2.3
m2.1
bJi[4]
m2.3
m2.1
bJi[5]
m2.3
m2.1
bJi[6]
m2.3
m2.1
bJi[7]
m2.3
m2.1
bJi[8]
m2.3
m2.1
bJi[9]
m2.3
m2.1
bWi[1]
m2.3
m2.1
bWi[2]
m2.3
m2.1
bWi[3]
m2.3
m2.1
bWi[4]
m2.3
m2.1
bWi[5]
m2.3
m2.1
bWi[6]
m2.3
m2.1
bWi[7]
m2.3
m2.1
bWi[8]
m2.3
m2.1
bWi[9]
m2.3
m2.1
bWi[10]
m2.3
m2.1
bWi[11]
m2.3
m2.1
bWi[12]
m2.3
m2.1
bWi[13]
m2.3
m2.1
bWi[14]
m2.3
m2.1
bWi[15]
m2.3
m2.1
bWi[16]
m2.3
m2.1
bWi[17]
m2.3
m2.1
bWi[18]
m2.3
m2.1
bWi[19]
m2.3
m2.1
bWi[20]
m2.3
m2.1
sigma
m2.3
m2.1
−1.0
−0.5
0.0
0.5
1.0
Value

One can directly see, that both models calculated almost identical posterior distributions.

Is there variation among individual judges and individual wines? Are there patterns you can notice just by plotting the differences?

- The parameters $\beta Ji[jid]$ explain the average influence of the individual judges on the scores of the wines. This means, judges with lower parameter values distribute on average lower scores, the ones with higher parameter values judge the wines on average with higher scores. Looking at the graph above, I definitely see variations in the $\beta Ji[jid]$ parameters and I am able to tell some of the judges apart as well.
- The parameters $\beta Wi[wid]$ indicate the average individual score of the wines across all judges. These parameters vary only a bit. Wine 18 has a uniquely identifiable parameter distribution, but other than that, the wines cannot easily be told apart.

Which judges have the lowest and highest ratings? Which wines are rated worst and which best on average?

- Looking at the mean values for the parameters $\alpha[jid]$ and at the plot, judge 8 gave on average the lowest ratings followed by judge 4 and judge 5 gave the highest ratings followed by judge 6. Looking at all judges, judge 1 and 9 still gave on average scores slightly below the average score, whereas judges 2, 3, and 7 gave on average scores slightly above the average score.
- Comparing the same for the parameters $\beta[wid]$, wine 18 had the lowest ratings. As the other wines do not vary that much, wine 4 had the highest average ratings, closely followed by and wine 20 and others.

Excercise 3

First, I created indicator variables containing values of 0 and 1 for `flight`, `wine.amer` and `judge.amer`:

```r
dat_list <- list(
  S = standardize(d$score),
  R = ifelse(d$flight == "red ", 1L, 0L),
  WA = as.integer(d$wine.amer),
  JA = as.integer(d$judge.amer)
)
str(dat_list)
```

```
## List of 4
##  $ S : num [1:180] -1.5766 -0.4505 -0.0751 0.3003 -2.3274 ...
##   ..- attr(*, "scaled:center")= num 14.2
##   ..- attr(*, "scaled:scale")= num 2.66
##  $ R : int [1:180] 0 0 0 0 0 0 0 0 0 0 ...
##  $ WA: int [1:180] 1 1 0 0 1 1 1 0 1 0 ...
##  $ JA: int [1:180] 0 0 0 0 0 0 0 0 0 0 ...
```

Then I used `ulam` to compute the posterior distribution and analyze the effects of `R` (flight), `WA` (wine.amer) and `JA` (judge.amer) on `S`:

```r
m3 <- ulam(
  alist(
    S ~ dnorm(mu, sigma),
    mu <- a + bR * R + bW * WA + bJ * JA,
    a ~ dnorm(0, 0.5),
    bR ~ dnorm(0, 0.33),
    bW ~ dnorm(0, 0.33),
    bJ ~ dnorm(0, 0.33),
    sigma ~ dexp(1)
  ),
  data = dat_list, chains = 4, cores = 4
)
```

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

```
precis(m3, depth = 2)
```

```
##                mean         sd        5.5%      94.5%    n_eff     Rhat4
## a     -0.016419631 0.13292331 -0.22880722 0.19920413 1300.229 0.9996076
## bR    -0.008392505 0.33439541 -0.52876266 0.52401321 1706.445 1.0010991
## bW    -0.158578782 0.14003554 -0.37959740 0.06496342 1433.879 0.9997560
## bJ     0.203877064 0.13701334 -0.01132046 0.41847453 1625.302 0.9982784
## sigma  0.995044037 0.05028861  0.91729476 1.08066228 1919.214 1.0010445
```

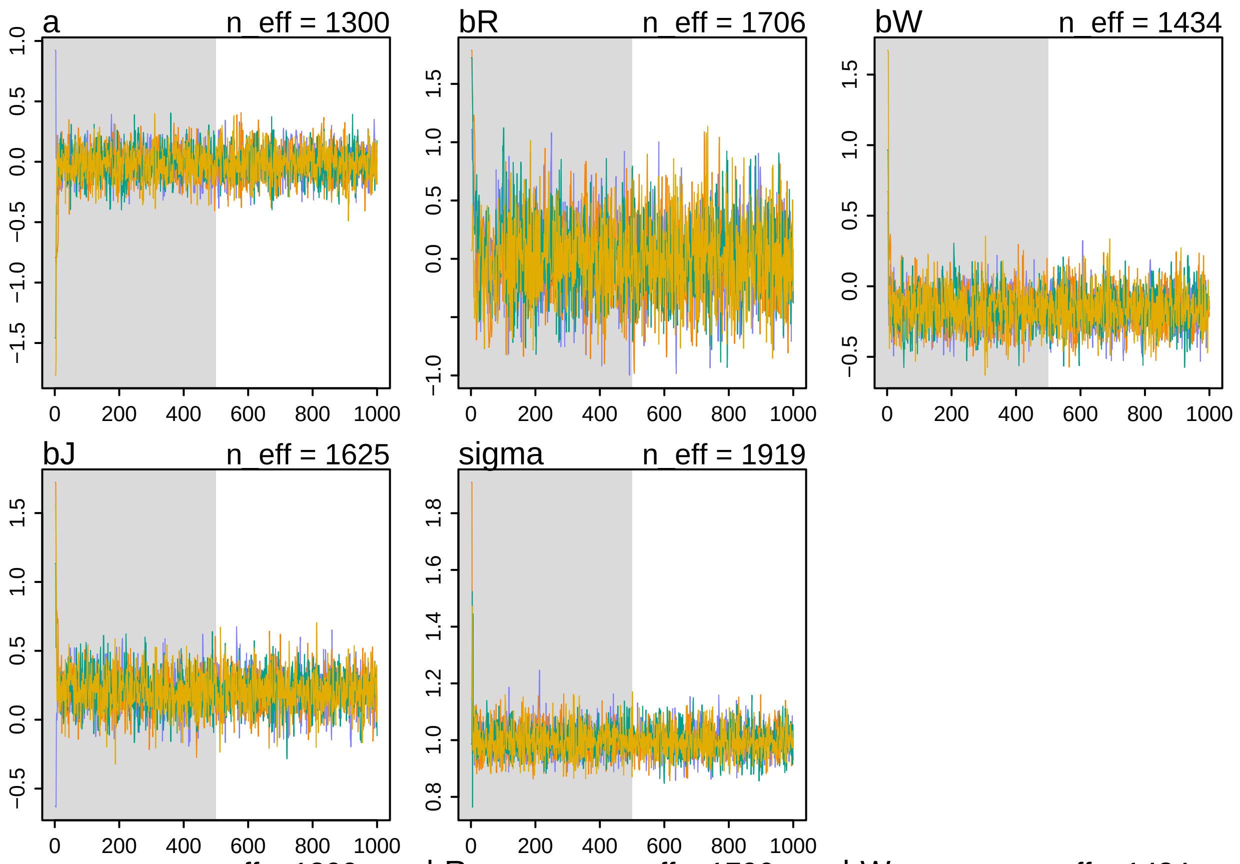
For the same reasons as in Exercise 2, I picked normal distributions with a mean of 0, but a standard deviation of 0.33 for the parameters `bR`, `bW` and `bJ`, . This is because we now have three indicator variables that are able to build the score. I gave each the same share of the standard deviation of the score, because I did not have any more information about their impact. This time I also included an intercept α to check its influence. I gave it a mean value of 0 and a standard deviation of 0.5 to be able to span across the entire range of standardized score values.

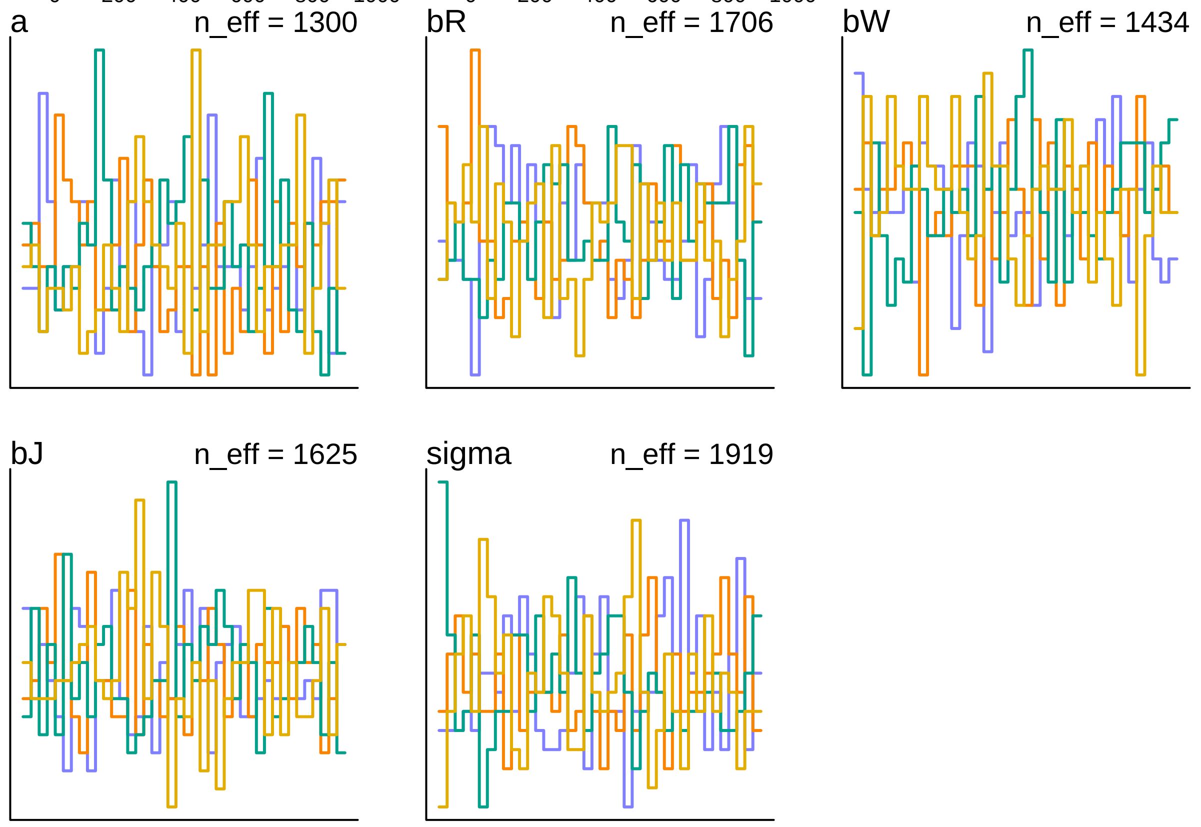
In order to check the chains for convergence, I use the `traceplot` and `trankplot` functions again:

```
traceplot(m3)
```

```
## [1] 1000
## [1] 1
## [1] 1000
```

```
trankplot(m3)
```

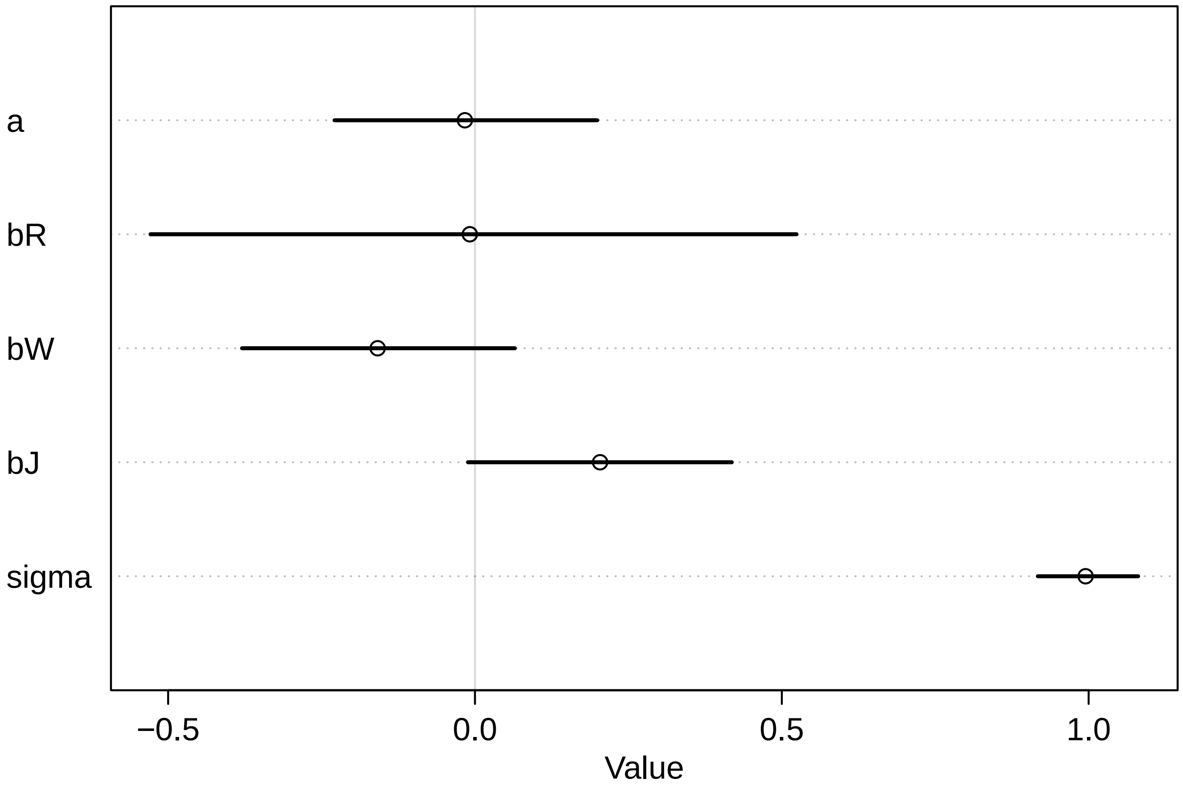




The outcomes are similar to the plots for m2.3: In the traceplot, the four independent chains stick around the same region of high probability. In the trankplot, the histograms overlap and are within the same range. Therefore, the chains are converging. Another indicator for the convergence is that the Rhat values are all 1.

To further analyze the parameters, I plotted them:

```r
plot(precis(m3, depth=2))
```



The intercept α is distributed on both sides of zero, having a mean close to zero. This means, as already expected in Exercise 2, α has no significant influence on the standardized score.

What do you conclude about the differences among the wines and judges?

- The parameter `bR` has a mean close to zero and a high standard deviation, spreading from values below to values above zero. This means that on average red and white wines are scored the same. Therefore, the flight of a wine has no significant influence on the average score a wine receives.
- The parameter `bW` is mostly below zero. This means that on average French wines receive better scores than American wines.
- The parameter `bJ` is slightly above zero. This means that on average American judges tend to give wines higher scores than French judges.

Compare your results to the results from Problem 2.

- Comparing the models in general, the model from exercise 2 tries to display the effects of individual judges and individual wines on the average score, whereas the model from exercise 3 tries to display the effects of features of those wines and judges on the average score.
- Additionally, it is possible to compare the actual results with each other. In order to do that I tried to derive the specific features for judges that give high/low scores on average and wines that get high/low scores on average from exercise 2:

```r
wine <- levels(d$wine)
judge <- levels(d$judge)
```

```
# get row for wines
print("Wine 18:")
```

```
## [1] "Wine 18:"
```

```
d[which(d$wine == wine[18]), ][1, ]
```

```
##                judge flight wine score wine.amer judge.amer
## 99 Jean-M Cardebat    red   I2    10         1          0
```

```
print("Wine 4:")
```

```
## [1] "Wine 4:"
```

```
d[which(d$wine == wine[4]), ][1, ]
```

```
##                judge flight wine score wine.amer judge.amer
## 92 Jean-M Cardebat    red   B2    11         0          0
```

```
print("Wine 20:")
```

```
## [1] "Wine 20:"
```

```
d[which(d$wine == wine[20]), ][1, ]
```

```
##                 judge flight wine score wine.amer judge.amer
## 100 Jean-M Cardebat    red   J2  14.5         0          0
```

```
# get row for judges
print("Judge 8:")
```

```
## [1] "Judge 8:"
```

```
d[which(d$judge == judge[8]), ][1, ]
```

```
##               judge flight wine score wine.amer judge.amer
## 41 Robert Hodgson  white   A1    17         1          1
```

```
print("Judge 4:")
```

```
## [1] "Judge 4:"
```

```
d[which(d$judge == judge[4]), ][1, ]
```

```
##               judge flight wine score wine.amer judge.amer
## 1 Jean-M Cardebat  white   A1    10         1          0
```

```
print("Judge 5:")
```

```
## [1] "Judge 5:"
```

```
d[which(d$judge == judge[5]), ][1, ]
```

```
##        judge flight wine score wine.amer judge.amer
## 21 John Foy  white   A1    16         1          1
```

```
print("Judge 6:")
```

```
## [1] "Judge 6:"
```

```
d[which(d$judge == judge[6]), ][1, ]
```

```
##             judge flight wine score wine.amer judge.amer
## 51 Linda Murphy  white   A1  15.5         1          1
```

- The result from exercise 3 was, that on average French wines receive better scores than American wines. Looking at the worst performing wine from exercise 2, which was wine 18, it is actually an american wine. The wines with high ratings, wine 4 and wine 20 are both french wines. Therefore, the conclusions of both exercises match.
- The second result from exercise 3 was, that on average American judges tend to give wines higher scores than French judges. Comparing this with the results form exercise 2, the judges 5 and 6 gave the highest ratings and are both American. The judges 8 and 4 gave the worst ratings. Judge 4 is actually a French judge, but judge 8 is a American one. Nevertheless, as all conclusions are about average tendencies, the conclusions of both exercises still match.