CSP Problem Set 4

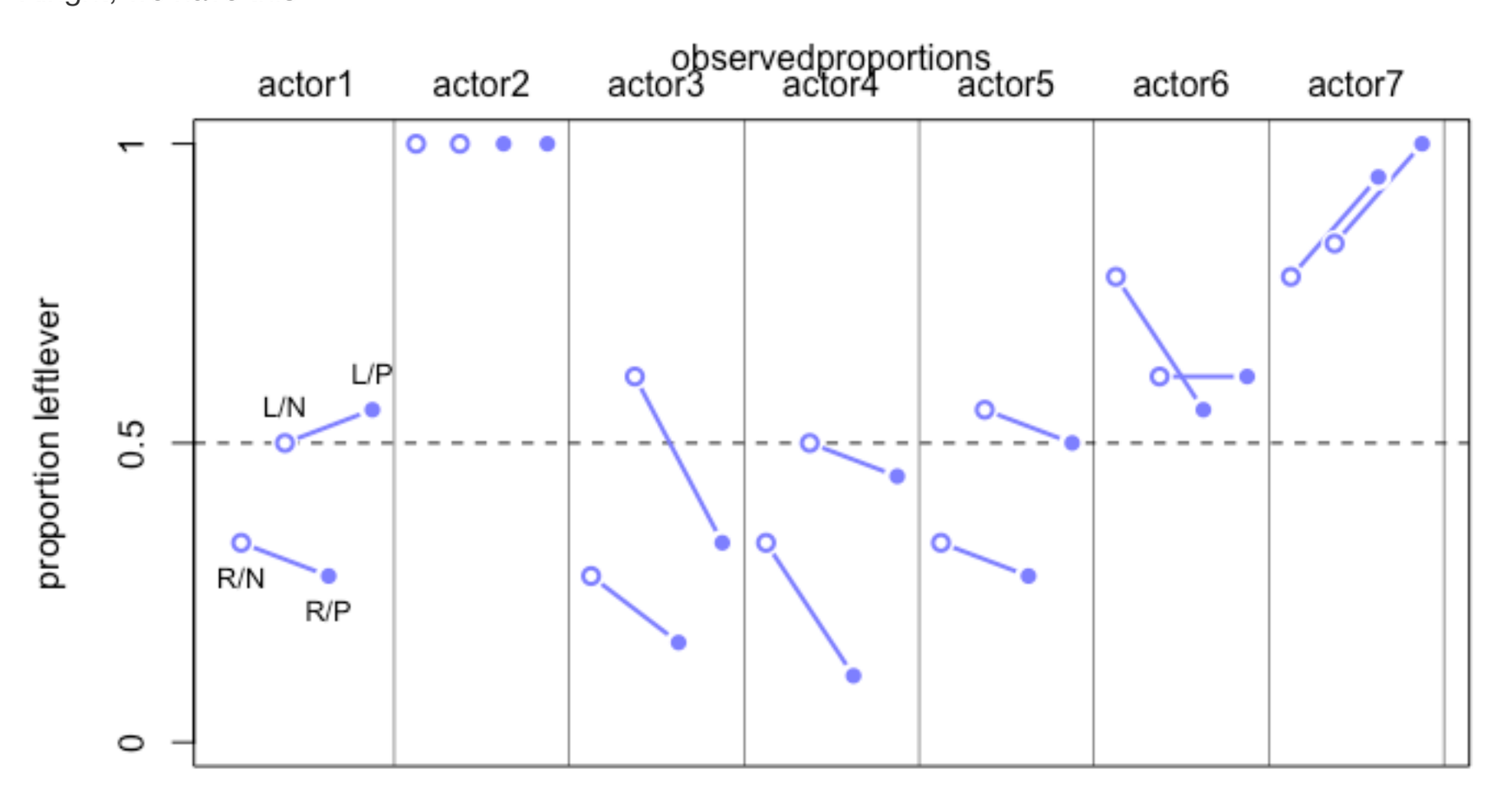
Code

Nicolas Kepper

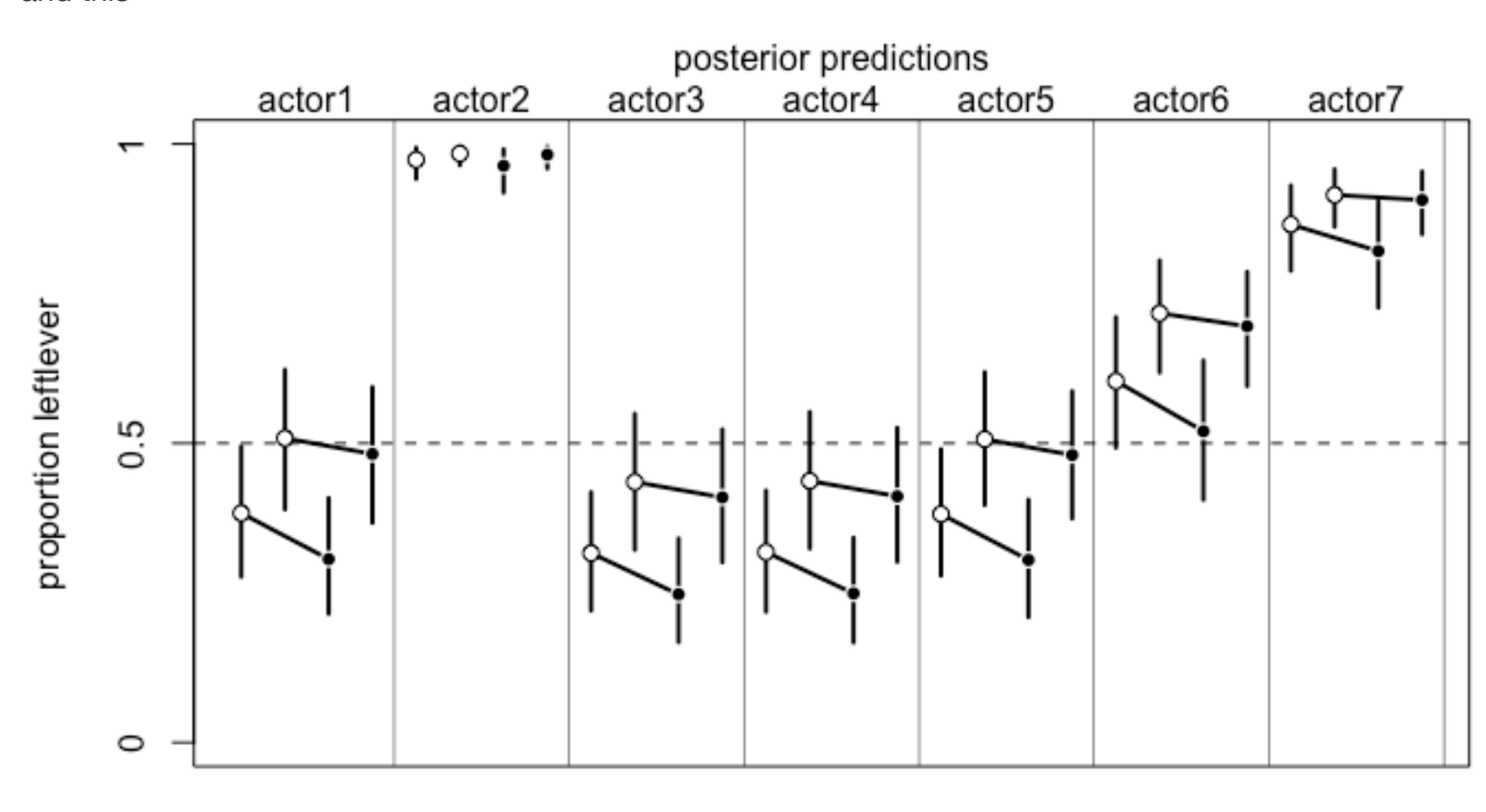
1 12 2021

Exercise 1

Alright, we have this



and this



Let's cover the process of getting from the first one to the second one step by step. Bear with me, this is going to be a lot of the same code, copied and pasted with just small changes everytime.

First, let's have a look at the code producing the first plot

There's quite a lot going on here - obviously, there's like a billion lines on the plot, they have to come from somewhere - and I won't go into detail on every step. I will also not show the new plot after every single step, only when there are major changes. I will however mostly explain the code that will be replaced and, of course, I'll also explain the replacements and why they're necessary to get to the desired output plot.

Let's get rid of the most obvious and easiest changes first. Color and title. The title is specified in the very last line of the code. Let's keep the \n in the title in order to have a bit of space between the title and the plot. To replace the color, just Ctrl + F and replace `col=rangi2` with `col="black"`. We should also remove the R/N etc texts.

Now, the first big step. We need to use different data. The first plot uses observed data. We want to show what our model predicts instead. Let's start with building a vector that we can use as an input for our model. It will contain every combination of actor and treatment. Then we let the model predict each outcome. Similar to previous chapters in the book and previous problem sets we can use the mean and the pi function to get the mean prediction and the 89% confidence interval.

Hide

```
dat <- list(actor=rep(1:7, each=4), treatment=rep(1:4, times=7))
post <- link(m, data=dat)
post.mu <- apply(post, 2, mean)
post.ci <- apply(post, 2, PI)
```

First, let's only replace the dots and circles so we can still see how much they change position compared to before, by looking at the old lines.

Hide

```
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))
```

Hide

```
abline(h=0.5, lty=2)
for(j in 1:7) abline(v=(j - 1) * 4 + 4.5, lwd=0.5)
```

Hide

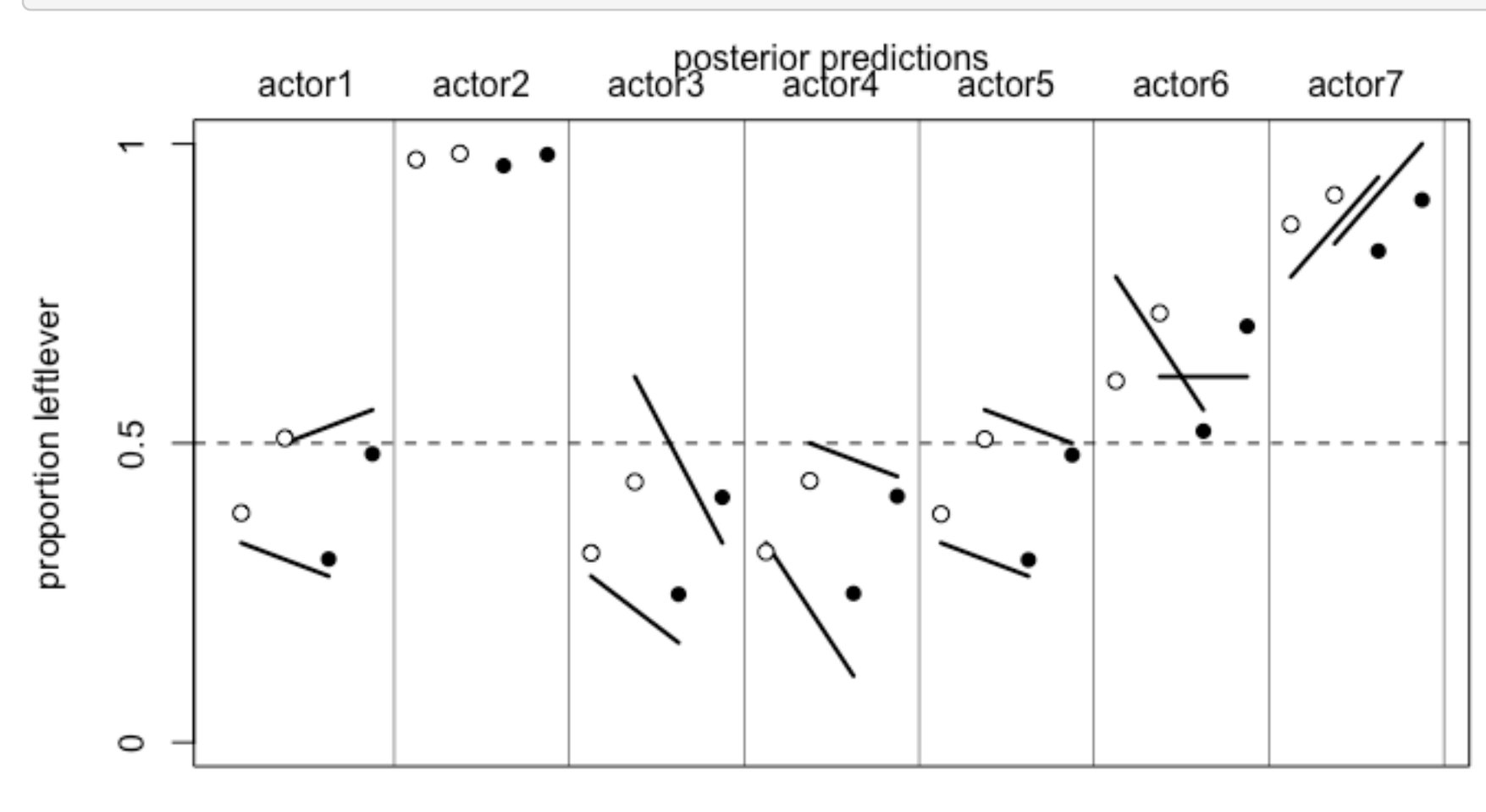
```
for(j in 1:7) text((j - 1) * 4 + 2.5, 1.1, concat("actor", j), xpd=TRUE)
for(j in (1:7)[-2]){
  lines((j - 1) * 4 + c(1,3), pl[j, c(1,3)], lwd=2, col="black")
  lines((j - 1) * 4 + c(2,4), pl[j, c(2,4)], lwd=2, col="black")
}
```

Hide

```
points(1:28, post.mu, pch=16, col="white")
points(1:28, post.mu, pch=c(1,1,16,16), col="black", lwd=1)
```

Hide

```
mtext("posterior predictions\n")
```



Next, let's move the lines, so they connect the matching dots and circles.

Hide

```
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))
```

Hide

```
abline(h=0.5, lty=2)
for(j in 1:7) abline(v=(j - 1) * 4 + 4.5, lwd=0.5)
```

Hide

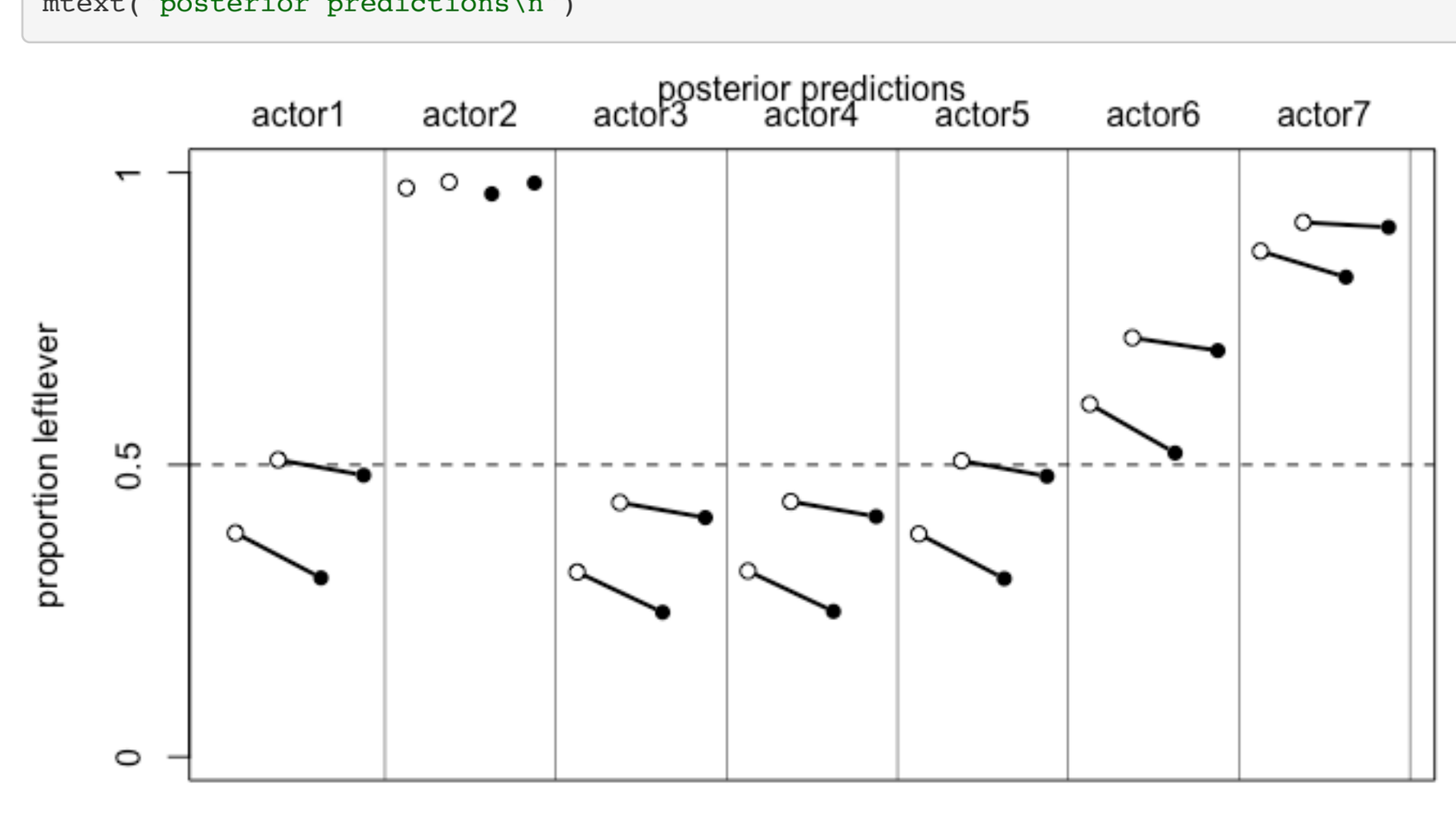
```
for(j in 1:7) text((j - 1) * 4 + 2.5, 1.1, concat("actor", j), xpd=TRUE)
for(j in (1:7)[-2]){
  index_vectors <- cbind((j - 1) * 4 + c(1, 3), (j - 1) * 4 + c(2, 4))

  lines(index_vectors[, 1], post.mu[index_vectors[, 1]], lwd=2, col="black")
  lines(index_vectors[, 2], post.mu[index_vectors[, 2]], lwd=2, col="black")
}
```

Hide

```
points(1:28, post.mu, pch=16, col="white")
points(1:28, post.mu, pch=c(1,1,16,16), col="black", lwd=1)
```

Hide

```
mtext("posterior predictions\n")
```



The `index_vectors` variable always contains the four x-values, i.e. indices in the post.mu vector in form of a matrix. There would have been more readable approaches of doing this, but where's the fun of programming when you can't slam everything into a single variable.

Only the lines that show the 89% CI are missing now. Adding them is no witchcraft either. Let's throw in some quick style changes as well. In the "source-plot" the lines didn't touch the circles and dots. In the "target-plot" the lines touch the circles and barely don't touch the dots. Also, I'll move the "actor" texts down a touch, because they're interfering with the title a little - at least in my test knittings.

Hide

```
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))
```

Hide

```
abline(h=0.5, lty=2)
for(j in 1:7) abline(v=(j - 1) * 4 + 4.5, lwd=0.5)
```

Hide

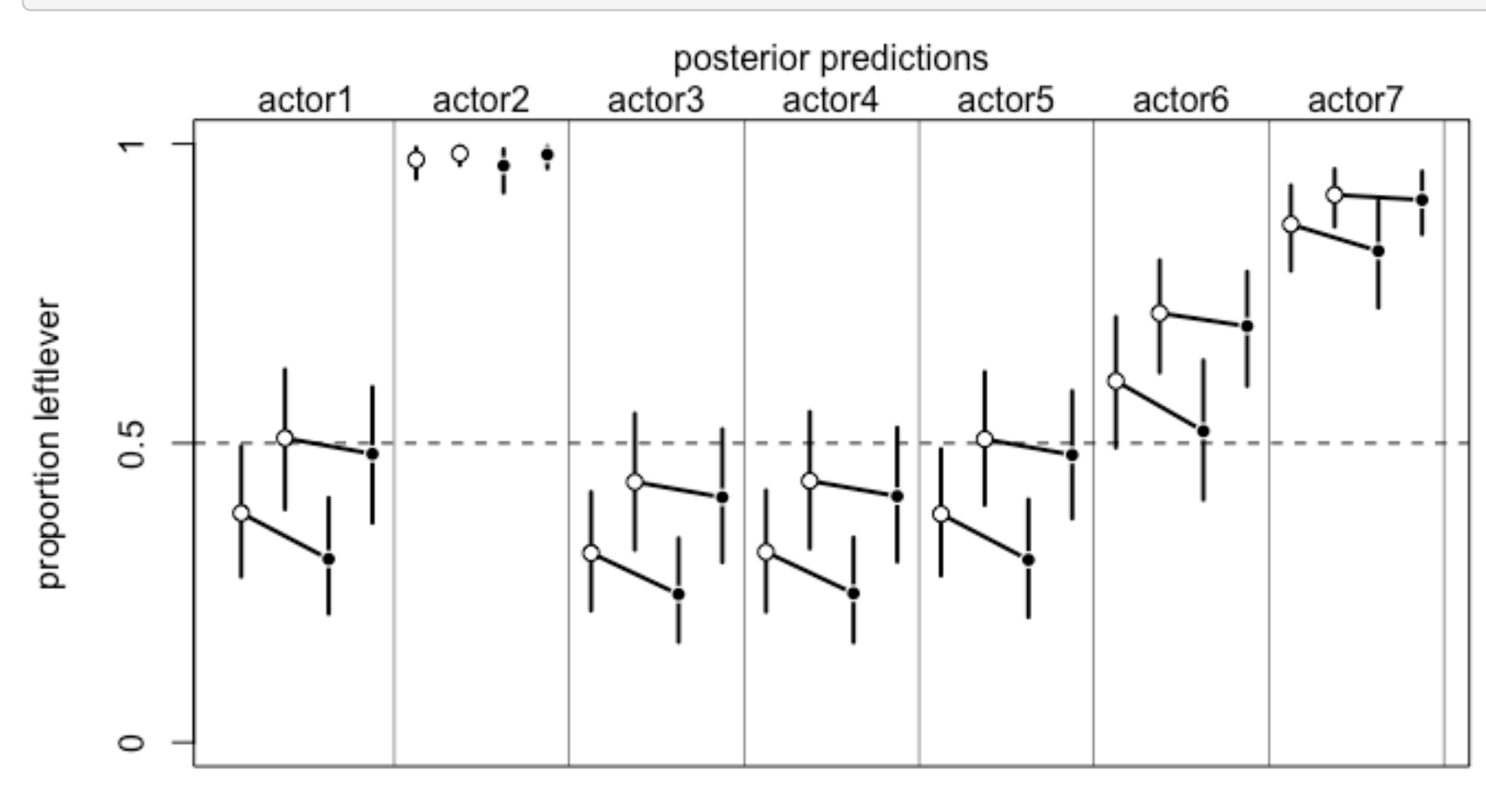
```
for(j in 1:7) text((j - 1) * 4 + 2.5, 1.075, concat("actor", j), xpd=TRUE)
for(j in (1:7)[-2]){
  index_vectors <- cbind((j - 1) * 4 + c(1, 3), (j - 1) * 4 + c(2, 4))

  lines(index_vectors[, 1], post.mu[index_vectors[, 1]], lwd=2, col="black")
  lines(index_vectors[, 2], post.mu[index_vectors[, 2]], lwd=2, col="black")
}
```

Hide

```
for(i in 1:28) {
  lines(rep(i, 2), post.ci[,i], lwd=2)
}

points(1:28, post.mu, pch=16, col="white", cex=c(1, 1, 1.1, 1.1))
```

Hide

```
points(1:28, post.mu, pch=c(1,1,16,16), col="black", lwd=1, cex=c(1, 1, 0.8, 0.8))

mtext("posterior predictions\n")
```



And there we go. Looks pretty similar to the desired outcome.

Exercise 2

Let's use a variable j with 9 different indices (jid) for the judges and w with 20 indices (wid) for the wines.

The score variable is standardized. Looking back at prior predictive simulations from previous problem sets, book chapters and exercises, we can remember that a prior $\mathcal{N}(0, 1)$ can yield very wild priors. Let's use some mean-faithful priors - $\mathcal{N}(0, 0.5)$ have proven themselves worthy in the past. Since we don't really have much prior information, these seem pretty well suited for this model.

Hide

```
data(Wines2012)
d <- Wines2012

dat_list <- list(
  S = standardize(d$score),
  jid = as.integer(d$judge),
  wid = as.integer(d$wine)
)

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

	mean	sd	5.5%	94.5%	n_eff	Rhat4
	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
j[1]	-0.28	0.19	-0.59	0.03	2407	1
j[2]	0.22	0.20	-0.09	0.54	2108	1
j[3]	0.21	0.20	-0.11	0.53	2332	1
j[4]	-0.54	0.20	-0.86	-0.21	2410	1
j[5]	0.80	0.19	0.49	1.10	2375	1
j[6]	0.48	0.19	0.16	0.78	2347	1
j[7]	0.13	0.20	-0.18	0.45	2341	1
j[8]	-0.66	0.20	-0.97	-0.35	2300	1
j[9]	-0.34	0.20	-0.67	-0.03	2578	1
w[1]	0.11	0.26	-0.30	0.53	3435	1

1-10 of 30 rows Previous 1 2 3 Next

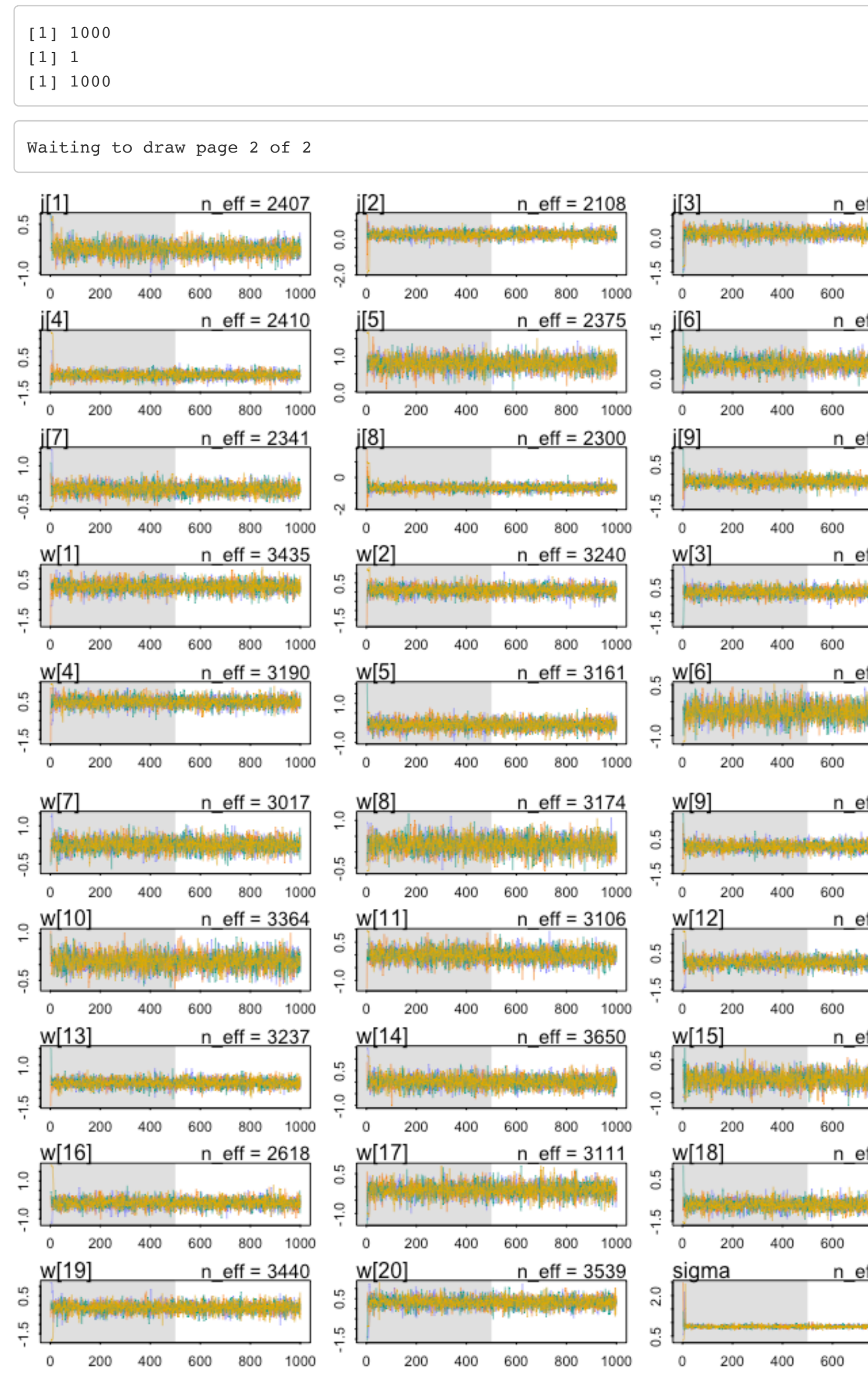
R-Hat is 1 for all wines and judges which means the chains all converged fine. We would need to take a deeper look at this if one of the R-Hat values had been higher than 1. Also the traceplots…

Hide

```
traceplot(m2)
```

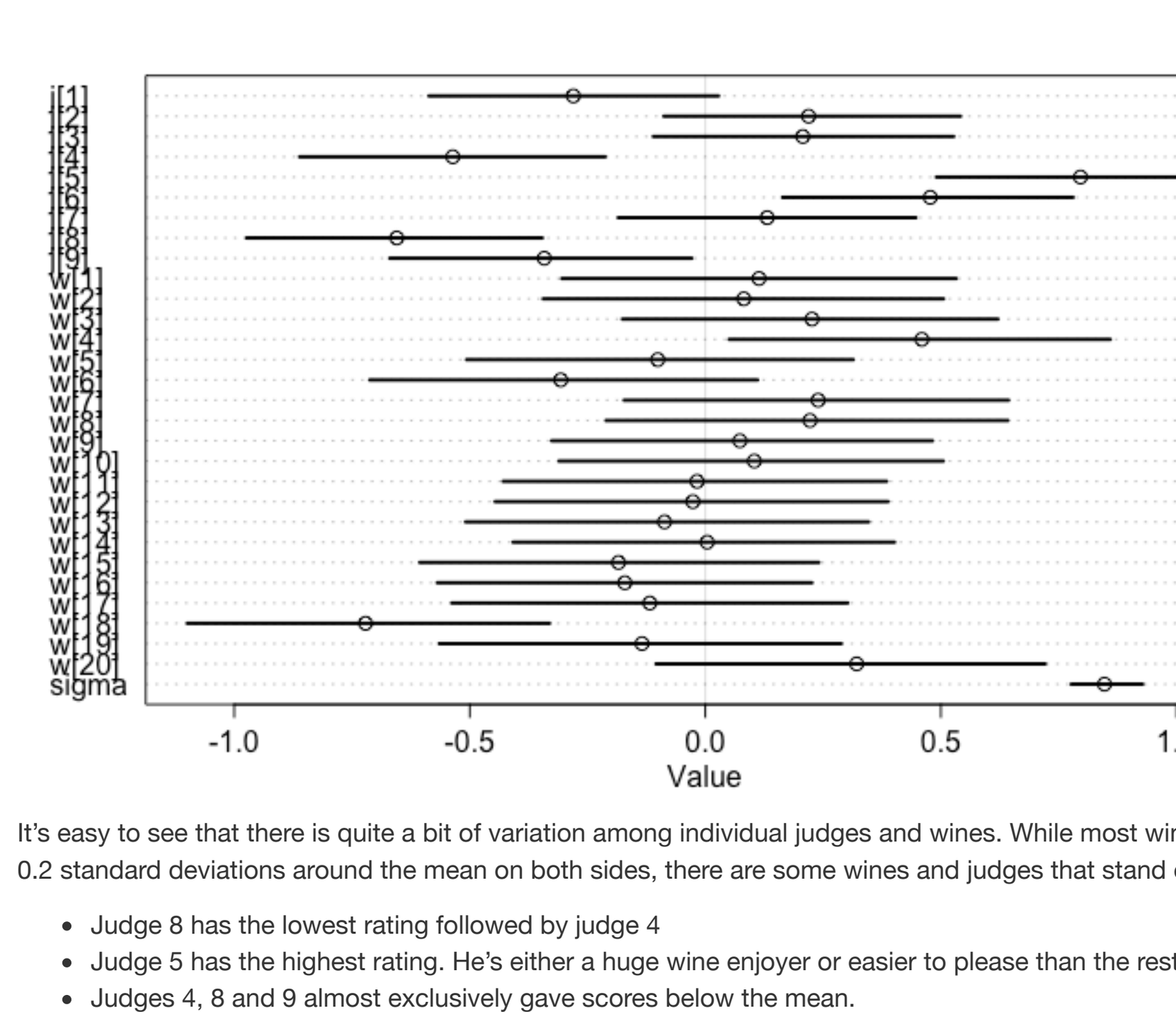
```
[1] 1000
[1] 1
[1] 1000
```

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



… confirm, that everything looks good. Additionally to the R-Hat values all being one, we can see here, that the two other conditions for healthy chains mentioned in the book are also fulfilled. The chains all look stationary with a lot of zig-zagging.

To answer the remaining questions, let's plot the precis. It'll be much easier to just read off the answers from that plot.



It's easy to see that there is quite a bit of variation among individual judges and wines. While most wines and judges receive/give scores about 0.2 standard deviations around the mean on both sides, there are some wines and judges that stand out.

- Judge 8 has the lowest rating followed by judge 4
- Judge 5 has the highest rating. He's either a huge wine enjoyer or easier to please than the rest of the judges.
- Judges 4, 8 and 9 almost exclusively gave scores below the mean.
- Most wines gather somewhere around the average, only wine 4, 20 and 18 stand out by a big margin. 4 is the best rated wine followed by 20 while 18 is by far the worst rated wine.

Exercise 3

Let's start with rebuilding our training data list. All three variables `R`, `W` and `J` are now either 0 or 1 for each dataset. R = 1 indicates that a wine is red, W = 1 indicates that the wine comes from america and J = 1 indicates that the judge is american.

Hide

```
dat_list <- list(
  S = standardize(d$score),
  R = ifelse(d$flight == "red", 1L, 0L),
  W = d$wine.amer,
  J = d$judge.amer
)
```

We can now train a pretty standard multivariate linear model with these predictors. Since S is standardized, the intercept should be very close to 0, so we can use a rather tight prior. For the three β's I'll use the same priors as in exercise 2.

Hide

```
m3 <- ulam(
  alist(
    S ~ dnorm(mu, sigma),
    mu <- a + bR * R + bW * W + bJ * J,
    a ~ dnorm(0, 0.2),
    c(bR, bW, bJ) ~ dnorm(0, 0.5),
    sigma ~ dexp(1)
  ), data=dat_list, chains=4
)
```

Hide

```
precis(m3)
```

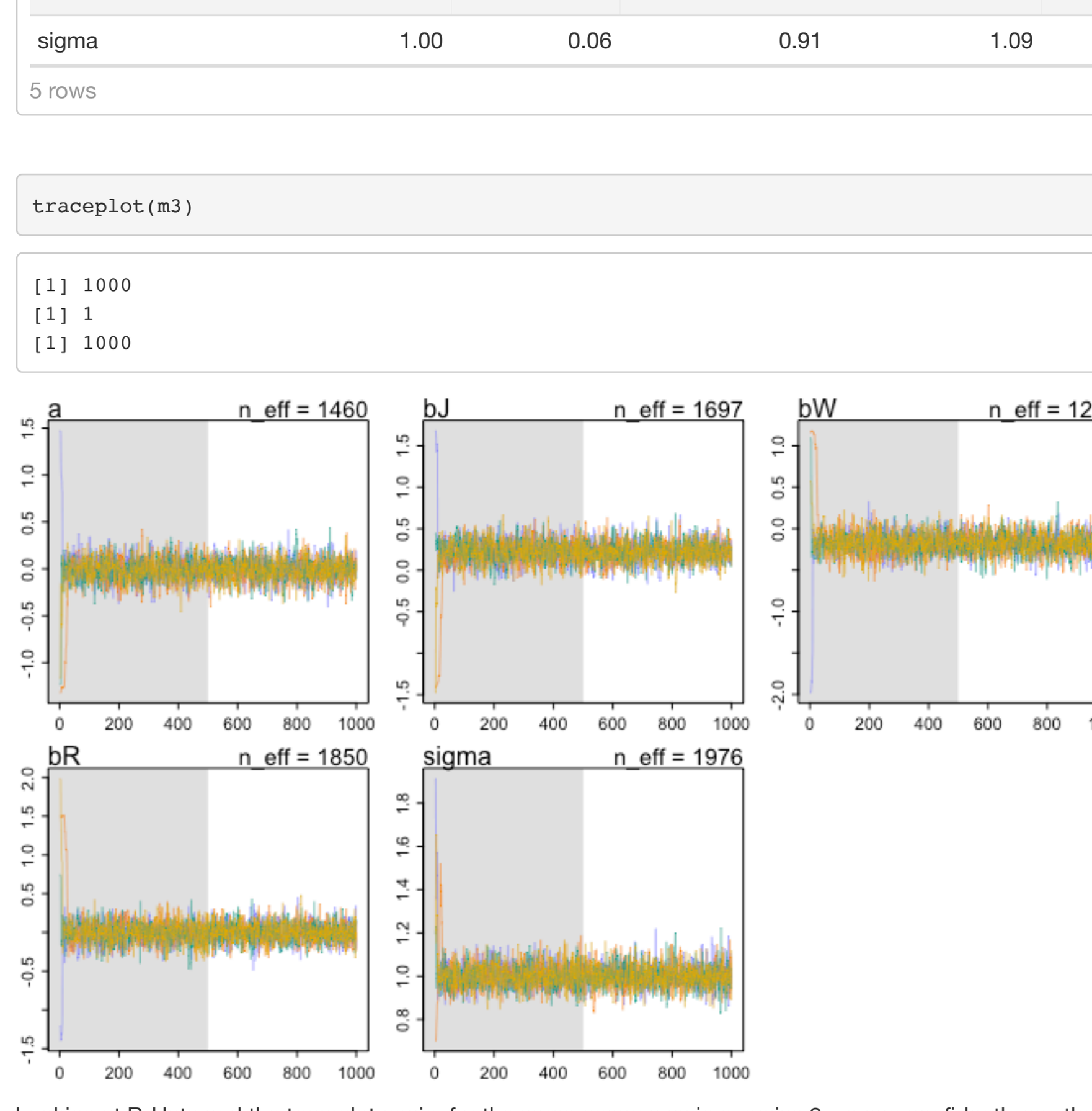
	mean	sd	5.5%	94.5%	n_eff	Rhat4
	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
a	-0.01	0.12	-0.21	0.18	1460	1
bJ	0.22	0.13	0.01	0.43	1697	1
bW	-0.18	0.14	-0.40	0.04	1290	1
bR	0.00	0.13	-0.21	0.21	1850	1
sigma	1.00	0.06	0.91	1.09	1976	1

5 rows

Hide

```
traceplot(m3)
```

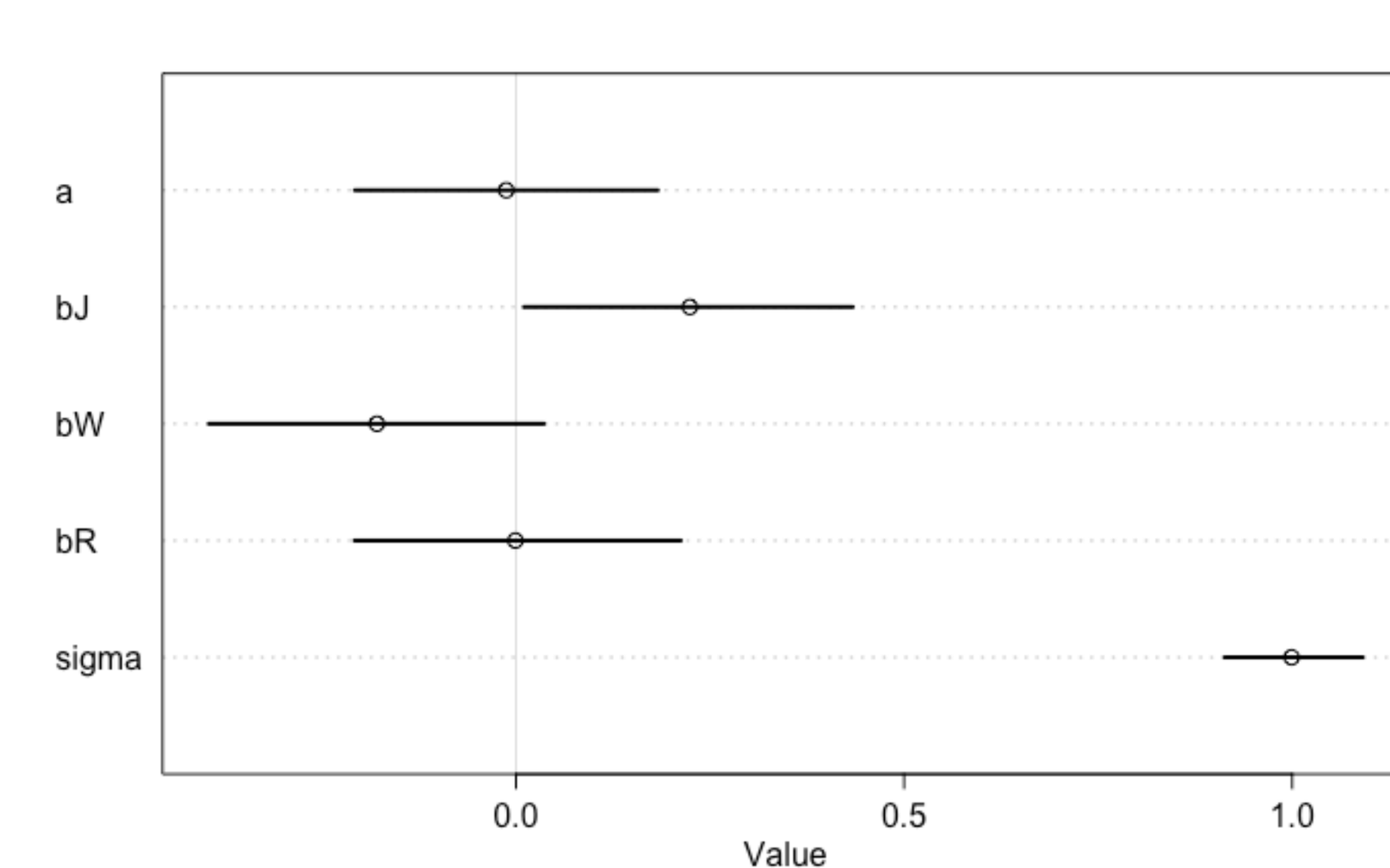
```
[1] 1000
[1] 1
[1] 1000
```



Looking at R-Hats and the traceplot again, for the same reasons as in exercise 2 we can confidently say that everything looks good.

Hide

```
plot(precis(m3))
```



Looking at the posterior values for the coefficients of this model we can conclude that generally there's no difference in scoring based on whether the wine is a red or a white wine. Therefore, american judges rate wines significantly higher than average judges while american wines quite a bit lower rated than average.

This partially matches the results of Problem 2. Three out of the 5 judges whose scores are better than average are american. However, out of the 4 judges giving scores below average 2 are american and 2 are not. This is a little weak when compared to the completely above average 89% confidence interval of bJ.

On the wines, it's a little bit clearer. In Ex. 2, most wines were more or less hugging the mean. The outliers, Wines 4, 18 and 20 however clearly resemble what the model from this exercise shows. 4 is the best wine followed by 20. They're both not american. Wine 18 - by far the worst rated wine- is an american wine.

Comparing the two is tough though. We're looking at the data from a completely different perspective. These different models definitely help with getting a deeper insight into the data and help with understanding the dataset better.