HW 10

Problem 1

(1) & (2)

We will skip the single run in (1) and do 100 repeat runs directly.

```
library(rstan)
options(mc.cores = parallel::detectCores())
rstan_options(auto_write = TRUE)

lr <- stan_model(file = 'LR.stan') # compile the model

beta <- c(3, .1, .5)
coverage <- matrix(NA, nrow=100, ncol=3)

for (i in 1:100) {
  x1 <- seq(1, 100, 1)
  x2 <- rbinom(100, 1, .5)
  err <- rt(100, df=4) * 5

  y <- 3 + .1*x1 + .5*x2 + err

  lr_dat <- list(N=100, x1=x1, x2=x2, y=y)

  chain <- sampling(object = lr, data = lr_dat, seed = 0)
  stats <- summary(chain)$summary
  coverage[i, ] <- (stats[1:3, 5] < beta) & (stats[1:3, 7] > beta)
}

colMeans(coverage, na.rm = T)
```

```
## [1] 0.47 0.53 0.53
```

(3)

```
lr_t <- stan_model(file = 'LR_t.stan') # compile the model

beta <- c(3, .1, .5)
coverage_t <- matrix(NA, nrow=100, ncol=3)

for (i in 1:100) {
  x1 <- seq(1, 100, 1)
  x2 <- rbinom(100, 1, .5)
  err <- rt(100, df=4) * 5

  y <- 3 + .1*x1 + .5*x2 + err
```

```r
  lr_dat <- list(N=100, x1=x1, x2=x2, y=y)

  chain <- sampling(object = lr_t, data = lr_dat, seed = 0)
  stats <- summary(chain)$summary
  coverage_t[i, ] <- (stats[1:3, 5] < beta) & (stats[1:3, 7] > beta)
  if (i/10 == round(i/10))  print(i)
}
```

```
## [1] 10
## [1] 20
## [1] 30
## [1] 40
## [1] 50
## [1] 60
## [1] 70
## [1] 80
## [1] 90
## [1] 100
```

```r
colMeans(coverage_t, na.rm = T)
```

```
## [1] 0.52 0.53 0.55
```

Problem 2

Denote the number of shock avoidances made by dog j before trial i as W_{ij} and whether the dog j received the shock in trail j as $S_{ij} \in \{0, 1\}$ where 0 means no shock. Let T_{ij} be a latent variable indicating the waiting time for dog j in trial i to jump when presented the CS.

The model is $S_{ij} = I(T_{ij} \leq 10)$ where $T_{ij} \sim \text{Exp}(\lambda_{ij})$ and $\log \lambda_{ij} = \alpha i + \beta W_{ij}$. The prior distribution for α and β are both uniform distribution on $(0, 100)$. Please note this is a weakly informative prior because we are enforcing that the training effect is positive and also the effect is not dramatically strong.

The posterior is

$$\begin{aligned}
p(\alpha, \beta, T|S) &\propto p(\alpha, \beta)p(T|\alpha, \beta)p(S|T) \\
&\propto (\alpha i + \beta W_{ij}) \exp\{-(\alpha i + \beta W_{ij})T_{ij}\} \, I\{T_{ij}S_{ij} \in \{0\} \cup (10, +\infty)\}.
\end{aligned}$$

```r
dogs_model <- stan_model(file = 'Dogs.stan') # compile the model

dogs <- read.table('dogs.txt', header=F, fill=T)[-(1:2), -1]
dogs[dogs == 'S'] <- 1
dogs[dogs == '.'] <- 0
W <- t(apply(dogs, 1, cumsum))
IND <- t(apply(matrix(1, 30, 25), 1, cumsum))
```

```r
dogs_dat <- list(J=30, I=25, S=dogs, W=W, IND=IND)
chain <- sampling(object = dogs_model, data = dogs_dat,
                  seed = 0, iter=5000, chains=1)
```

```
##
## SAMPLING FOR MODEL 'Dogs' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 0.000245 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 2.45 seconds.
```

```
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: Iteration:    1 / 5000 [  0%]  (Warmup)
## Chain 1: Iteration:  500 / 5000 [ 10%]  (Warmup)
## Chain 1: Iteration: 1000 / 5000 [ 20%]  (Warmup)
## Chain 1: Iteration: 1500 / 5000 [ 30%]  (Warmup)
## Chain 1: Iteration: 2000 / 5000 [ 40%]  (Warmup)
## Chain 1: Iteration: 2500 / 5000 [ 50%]  (Warmup)
## Chain 1: Iteration: 2501 / 5000 [ 50%]  (Sampling)
## Chain 1: Iteration: 3000 / 5000 [ 60%]  (Sampling)
## Chain 1: Iteration: 3500 / 5000 [ 70%]  (Sampling)
## Chain 1: Iteration: 4000 / 5000 [ 80%]  (Sampling)
## Chain 1: Iteration: 4500 / 5000 [ 90%]  (Sampling)
## Chain 1: Iteration: 5000 / 5000 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 451.801 seconds (Warm-up)
## Chain 1:                455.362 seconds (Sampling)
## Chain 1:                907.163 seconds (Total)
## Chain 1:
```

```r
stats <- summary(chain)$summary
stats[1:2, 9:10]
```

```
##            n_eff     Rhat
## alpha 12.569297 1.007875
## beta   9.935438 1.548232
```

For the summary we can tell how well it converged using n_eff and Rhat. Although the n_eff and Rhat are not satisfying, we will use 4000 iterations at this time because the program is running too slow for larger sample size.

Vectorization will help speed up but I could not find convenient way to vectorize this program and remove the for loop at this point.

Following is the posterior inference on (α, β).

```r
stats[1:2, c(1, 4:8)]
```

```
##              mean        2.5%         25%         50%         75%      97.5%
## alpha 0.020596677 0.020385227 0.020484875 0.020586324 0.020671442 0.02095566
## beta  0.009187564 0.007054933 0.008103831 0.009364539 0.009966576 0.01152375
```

Posterior inference on T_1 (mean and median only).

```r
stats[3:27, c(1, 6)]
```

```
##                mean          50%
## T[1,1]  41.75953910 41.393792356
## T[1,2]   2.35453912  2.237662569
## T[1,3]   5.10452635  5.105817052
## T[1,4]  15.79432810 15.704713677
## T[1,5]   1.67372394  1.655419089
## T[1,6]  27.24510895 27.887315058
## T[1,7]   0.68327047  0.699448637
## T[1,8]   5.66276154  5.643688848
## T[1,9]   0.33140361  0.334146627
## T[1,10]  2.31671671  2.312742968
```

```
## T[1,11]  4.18057826  4.258493729
## T[1,12]  1.15175788  1.129763969
## T[1,13]  4.85814431  4.840310104
## T[1,14]  0.90584605  0.905724569
## T[1,15]  0.21650298  0.200198477
## T[1,16]  9.15181639  9.207317028
## T[1,17]  0.36949873  0.359432960
## T[1,18]  4.47524874  4.520395080
## T[1,19]  0.83457638  0.844777769
## T[1,20]  1.05374010  1.051074735
## T[1,21]  3.56121346  3.698212388
## T[1,22]  3.82027084  3.769465958
## T[1,23]  2.15468285  2.171327547
## T[1,24]  0.35080644  0.361198969
## T[1,25]  0.01013905  0.009890074
```

Problem 3

Since the total number of 3×3 binary matrices with grand total equals 4 is $\binom{9}{4=126}$, the proposal is a uniform distribution on $\{1, \ldots, 126\}$ and our target is uniform on $\{1, \ldots, 5\}$ which represent the five possible arrangements of matrices.

```
samp <- rep(0, 1e4)
accept <- c()
target <- seq(1, 5, 1)
i <- 1
while (i <= 1e4) {
  ind <- sample(1:choose(9, 4), 1)
  if (ind %in% target) {
    samp[i] <- ind
    accept <- c(accept, 1)
    i <- i + 1
  } else {
    accept <- c(accept, 0)
  }
}
mean(accept)
```

```
## [1] 0.03942487
```

The acceptance rate is expected to be low because we are trying to sample 5 out of 126 uniformly. And the experiment shows that the acceptance rate is indeed low.