```
knitr::opts_chunk$set(message = FALSE, warning = FALSE)
```

```
############### 0 - safe choice A, 1 - risky choice B ####################
library(rstan); rstan_options(javascript=FALSE)
options(mc.cores = parallel::detectCores())
rstan_options(auto_write = T)
library(dplyr)



dat <- read.csv('final_data.csv')

dat <- dat %>%
  filter(skew != 'control')
dat <- dat %>%
  mutate(cho = ifelse(true_response == 'f', 1, -1))


ids <- unique(dat$Prolific_ID)
for(j in 1:length(ids)){
  dat$tid[dat$Prolific_ID==ids[j]] <- j
}
tids <- unique(dat$tid)

dat <- dat %>%
  filter(test_part == 'cs' | test_part == 'sc')

dat <- dat %>%
  mutate(
    oa_complex = ifelse(test_part == 'cs', 1, -1),
    evd = evd * oa_complex,
    sdd = sdd * oa_complex,
    chose_complex = ifelse((oa_complex == 1 & cho == 1) | (oa_complex == -1 & cho
== -1), 1, -1)
  )
```

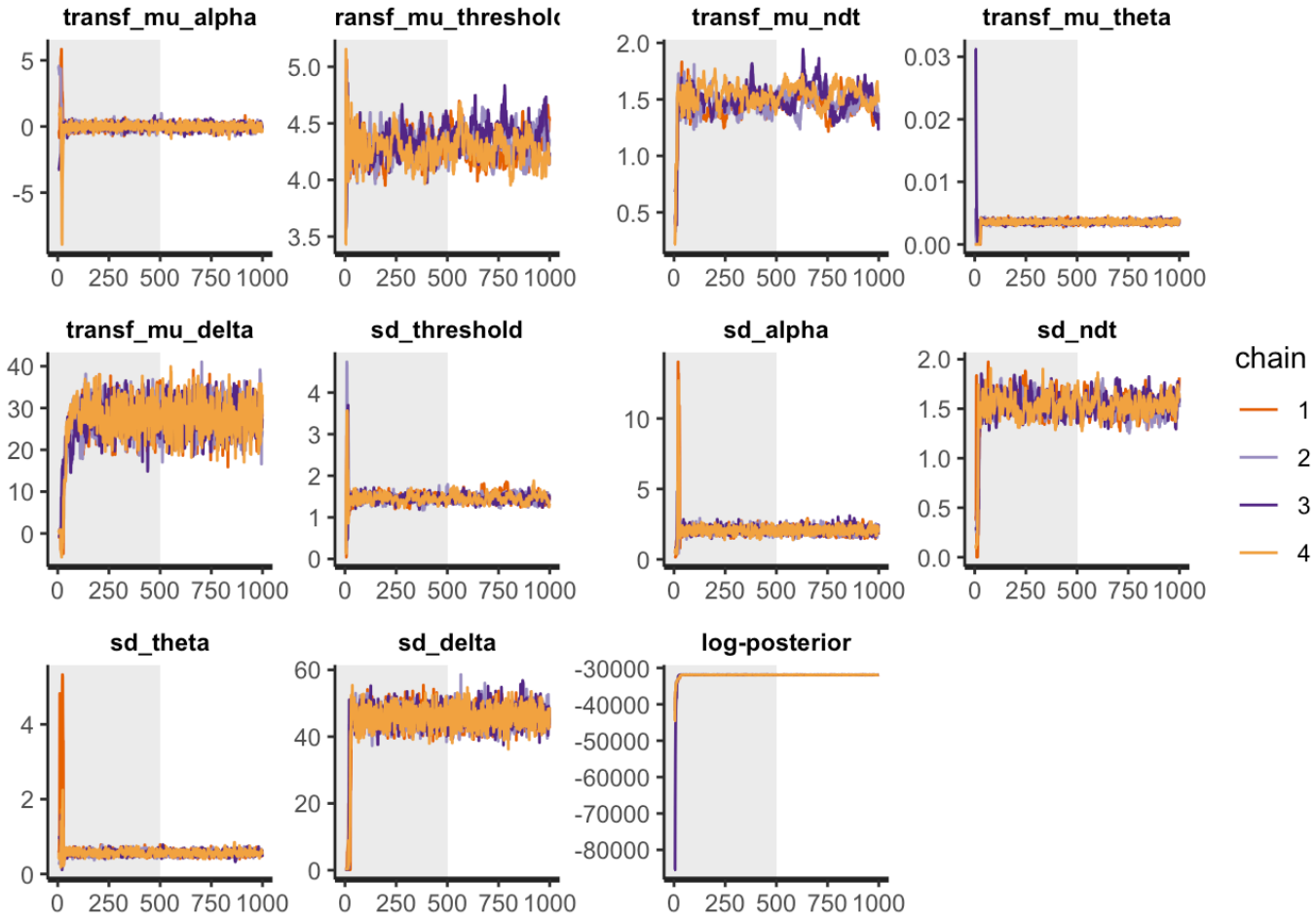
```
dat <- dat %>%
  mutate(
    trialtype = ifelse(skew == 'rl', 1, ifelse(skew == 'lr', -1, 0)) # 1 rl -1 lr
0 ns
  )

dat$rt <- dat$rt/1000


# Assuming your dataframe is named 'df'
dat$P_A1 <- dat$P_A1 / 100
dat$P_A2 <- dat$P_A2 / 100
dat$P_B1 <- dat$P_B1 / 100
dat$P_B2 <- dat$P_B2 / 100
```

```
dataList  = list(cho = dat$cho, rt = dat$rt, participant = dat$tid,N=nrow(dat),  L
= length(tids),starting_point=0.5, evd = dat$evd, sdd = dat$sdd, trialtype = dat$t
rialtype)
```

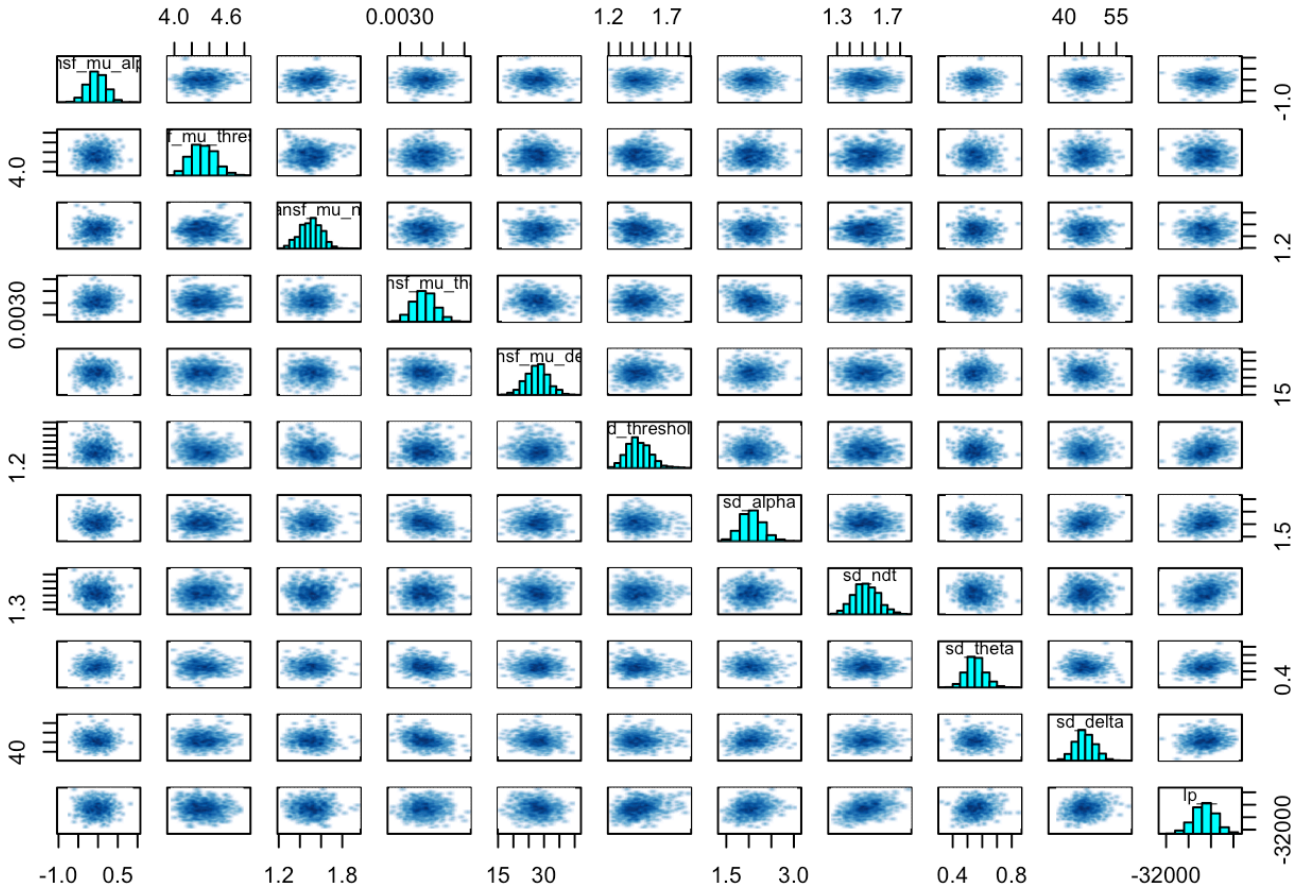
```
parameters = c("transf_mu_alpha","transf_mu_threshold","transf_mu_ndt", "transf_mu
_theta",'transf_mu_delta', 'sd_threshold',"sd_alpha","sd_ndt", 'sd_theta', 'sd_del
ta', "alpha_sbj","threshold_sbj","ndt_sbj",'theta_sbj', 'delta_sbj', "log_lik")

initFunc <-function (i) {
  initList=list()
  for (ll in 1:i){
    initList[[ll]] = list(
                          mu_alpha = runif(1,-5,5),
                          sd_alpha = runif(1,0,1),
                          mu_threshold = runif(1,-0.5,5),
                          sd_threshold = runif(1,0,1),
                          mu_ndt = runif(1, -1.5, 0),
                          sd_ndt = runif(1, 0, 1),
                          mu_theta = runif(1,-20, 1),
                          sd_theta = runif(1,0,1),
                          mu_delta = runif(1,-1, 1),
                          sd_delta = runif(1, 0, 1),
                          z_alpha = runif(length(tids),-0.1,0.1),
                          z_theta = runif(length(tids),-0.1,0.1),
                          z_threshold = runif(length(tids),-0.1,0.1),
                          z_ndt = runif(length(tids),-0.1,0.1),
                          z_delta = runif(length(tids),-0.1,0.1)


    )
  }

  return(initList)
}
```

```
m <- stan_model("MV_prob.stan")
dsamples <- sampling(m,
                data=dataList,
                pars=parameters,
                iter=1000,
                chains=4,#If not specified, gives random inits
                init = initFunc(4),
                warmup = 500,  # Stands for burn-in; Default = iter/2
                seed = 12,  # Setting seed; Default is random seed
                refresh = 0
                )
```

```
#parameters = c("transf_mu_alpha","transf_mu_threshold","transf_mu_ndt", "transf_mu_theta",'sd_threshold',"sd_alpha","sd_ndt", 'sd_theta', "alpha_sbj","threshold_sbj","ndt_sbj",'theta_sbj',"log_lik")

rstan::traceplot(dsamples, pars=c("transf_mu_alpha","transf_mu_threshold","transf_mu_ndt", "transf_mu_theta",'transf_mu_delta', 'sd_threshold',"sd_alpha","sd_ndt", 'sd_theta', 'sd_delta', "lp__"), inc_warmup = TRUE, nrow = 3)
```




```
pairs(dsamples, pars = c( "transf_mu_alpha","transf_mu_threshold","transf_mu_ndt", "transf_mu_theta",'transf_mu_delta', 'sd_threshold',"sd_alpha","sd_ndt", 'sd_theta', 'sd_delta', "lp__"))
```

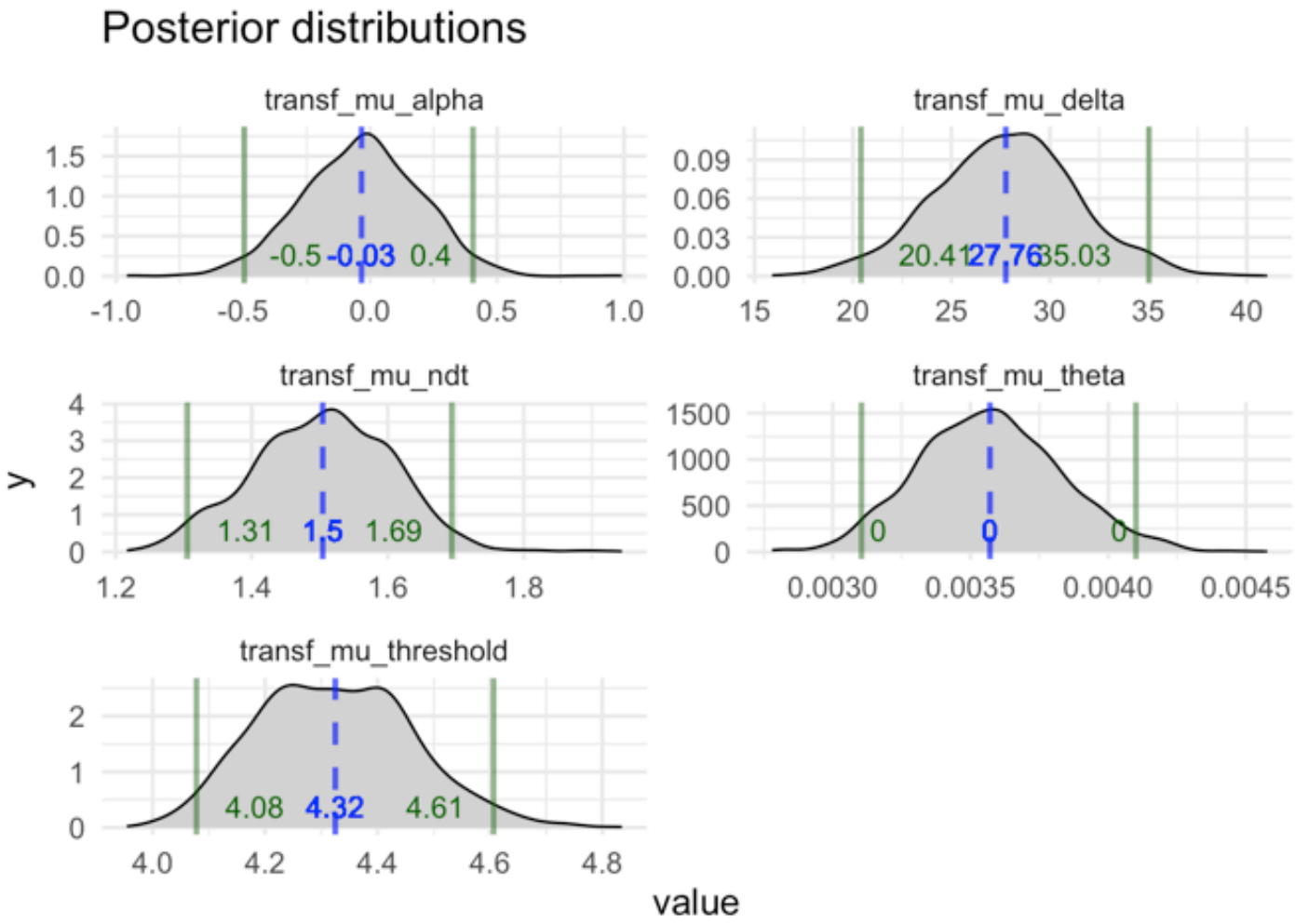



```
print(dsamples, pars = c("transf_mu_alpha","transf_mu_threshold","transf_mu_ndt",
"transf_mu_theta",'transf_mu_delta', 'sd_threshold',"sd_alpha","sd_ndt", 'sd_thet
a', 'sd_delta', "lp__"))
```

```
## Inference for Stan model: anon_model.
## 4 chains, each with iter=1000; warmup=500; thin=1;
## post-warmup draws per chain=500, total post-warmup draws=2000.
##
##                          mean se_mean    sd      2.5%       25%       50%
## transf_mu_alpha         -0.03    0.01  0.23     -0.50     -0.19     -0.03
## transf_mu_threshold      4.32    0.02  0.14      4.08      4.22      4.32
## transf_mu_ndt            1.50    0.02  0.10      1.31      1.43      1.51
## transf_mu_theta          0.00    0.00  0.00      0.00      0.00      0.00
## transf_mu_delta         27.76    0.12  3.63     20.41     25.40     27.78
## sd_threshold             1.46    0.01  0.10      1.28      1.39      1.45
## sd_alpha                 2.07    0.01  0.24      1.63      1.90      2.05
## sd_ndt                   1.53    0.01  0.10      1.35      1.46      1.53
## sd_theta                 0.56    0.00  0.06      0.44      0.51      0.55
## sd_delta                45.96    0.08  2.88     40.51     44.01     45.80
## lp__                -31914.21    1.35 24.70 -31962.68 -31930.17 -31914.25
##                          75%     97.5% n_eff Rhat
## transf_mu_alpha          0.12      0.40   791 1.00
## transf_mu_threshold      4.42      4.61    39 1.08
## transf_mu_ndt            1.58      1.69    44 1.12
## transf_mu_theta          0.00      0.00  1022 1.00
## transf_mu_delta         30.14     35.03   989 1.00
## sd_threshold             1.52      1.69   120 1.02
## sd_alpha                 2.23      2.57   679 1.01
## sd_ndt                   1.60      1.73   192 1.02
## sd_theta                 0.59      0.69   636 1.00
## sd_delta                47.83     51.89  1381 1.00
## lp__                -31897.42 -31864.10   334 1.01
##
## Samples were drawn using NUTS(diag_e) at Mon Jan  8 13:54:06 2024.
## For each parameter, n_eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor on split chains (at
## convergence, Rhat=1).
```

```
library(ggplot2)
library(tidyverse) # for the gather function

samples_matrix <- as.matrix(dsamples)
means <- colMeans(samples_matrix)
hpd_interval <- t(apply(samples_matrix, 2, function(x) quantile(x, probs=c(0.025,
0.975))))



parameters <- c("transf_mu_alpha","transf_mu_threshold","transf_mu_ndt", "transf_m
u_theta",'transf_mu_delta')

# Reshape data to a long format
df_long <- as.data.frame(samples_matrix) %>%
```

```
  gather(key = "parameter", value = "value", parameters)

# Convert hpd_interval to a data frame and name the columns
hpd_interval_sub <- hpd_interval[parameters, ]
hpd_df <- as.data.frame(hpd_interval_sub)
colnames(hpd_df) <- c("lower", "upper")
rownames(hpd_df) <- parameters
hpd_df$parameter <- rownames(hpd_df)



# Aesthetic enhancements
theme_set(theme_minimal(base_size = 14)) # Set the default theme

custom_palette <- c("density_fill" = "lightgray",
                    "mean_line" = "blue",
                    "hpd_line" = "darkgreen")

# Add text labels for mean, lower, and upper HPD values
df_long <- df_long %>%
  group_by(parameter) %>%
  mutate(mean = means[parameter])

hpd_df <- hpd_df %>%
  mutate(mid = (lower + upper) / 2)

p <- ggplot(df_long, aes(x = value)) +
  geom_density(aes(fill = "density_fill")) +
  scale_fill_manual(values = custom_palette, guide = FALSE) +
  geom_vline(aes(xintercept = mean, color = "mean_line"), linetype = "dashed", siz
e = 1, alpha = 0.7) +
  geom_text(data = df_long, aes(x = mean, y = 0, label = round(mean, 2)), vjust =
-0.5, hjust = 0.5, size = 4, color = custom_palette["mean_line"]) +
  geom_vline(data = hpd_df, aes(xintercept = lower, color = "hpd_line"), linetype
= "solid", size = 1, alpha = 0.5) +
  geom_text(data = hpd_df, aes(x = lower, y = 0, label = round(lower, 2)), vjust =
-0.5, hjust = -0.5, size = 4, color = custom_palette["hpd_line"]) +
  geom_vline(data = hpd_df, aes(xintercept = upper, color = "hpd_line"), linetype
= "solid", size = 1, alpha = 0.5) +
  geom_text(data = hpd_df, aes(x = upper, y = 0, label = round(upper, 2)), vjust =
-0.5, hjust = 1.5, size = 4, color = custom_palette["hpd_line"]) +
  facet_wrap(~ parameter, scales = "free", ncol = 2) +
  scale_color_manual(values = custom_palette, guide = 'none') +
  labs(title = "Posterior distributions")

print(p)
```


Posterior distributions
transf_mu_alpha
-0.5 -0.03 0.4
transf_mu_delta
20.41 27.76 35.03
transf_mu_ndt
1.31 1.5 1.69
transf_mu_theta
0 0 0
transf_mu_threshold
4.08 4.32 4.61
y
value