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

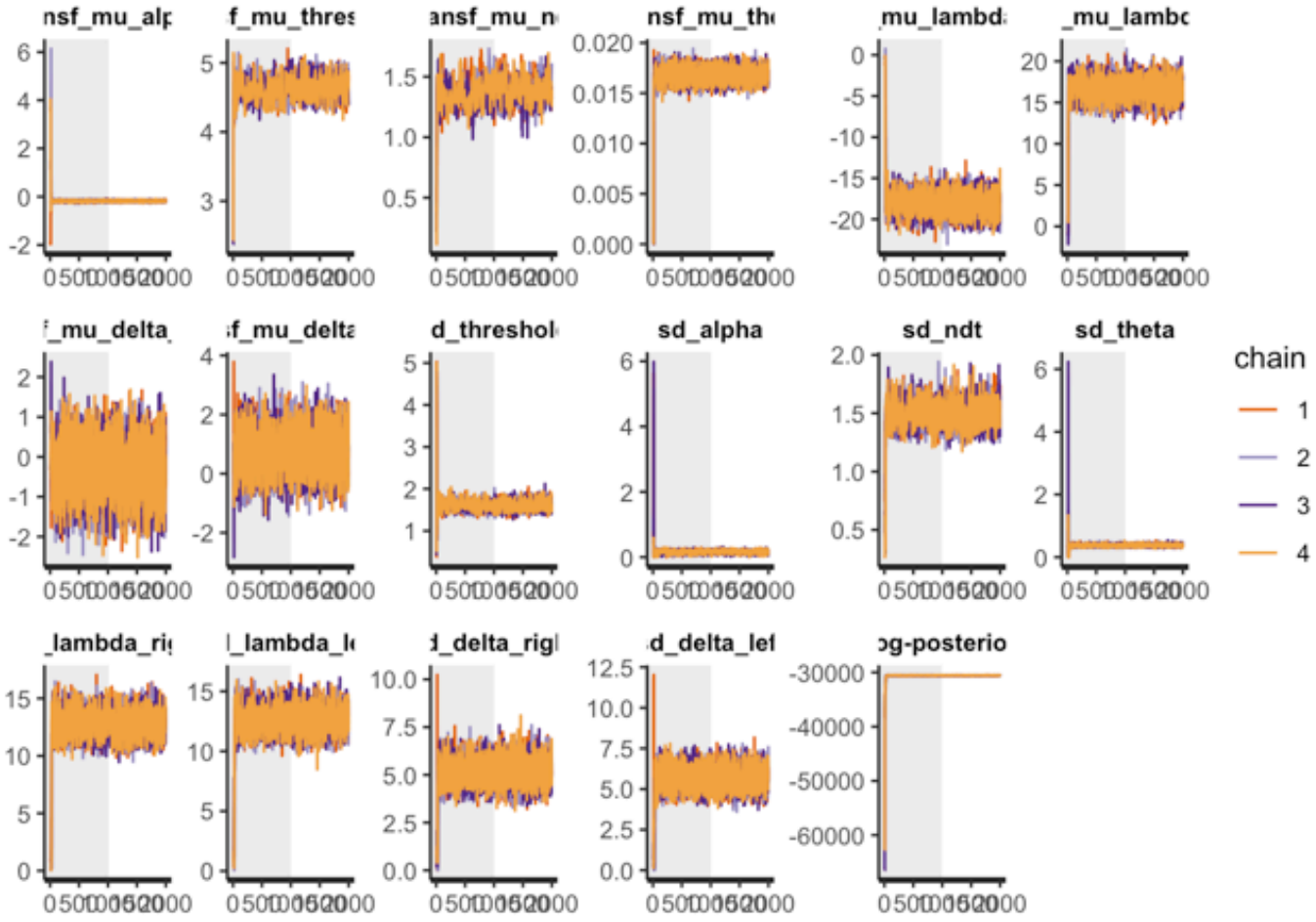
```
############### 0 - safe choice A, 1 - risky choice B ####################
library(rstan); rstan_options(javascript=FALSE)
options(mc.cores = parallel::detectCores())
rstan_options(auto_write = T)
library(dplyr)



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 == 'cc' | test_part == 'ss')

dat <- dat %>%
  mutate(con = ifelse(test_part == 'cc', 1, -1))
```

```
dat$trialtype1 <- ifelse(dat$skew == "ns", -1, ifelse(dat$skew == "lr", 1, ifelse(
dat$skew == "rl", 0, NA)))
dat$trialtype2 <- ifelse(dat$skew == "ns", -1, ifelse(dat$skew == "lr", 0, ifelse(
dat$skew == "rl", 1, NA)))


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$sdd, trialtype1 = dat$
trialtype1, trialtype2 = dat$trialtype2, con = dat$con)
```

```
parameters = c("transf_mu_alpha","transf_mu_threshold","transf_mu_ndt", "transf_mu
_theta",'transf_mu_lambda_right','transf_mu_lambda_left', 'transf_mu_delta_right',
'transf_mu_delta_left', 'sd_threshold',"sd_alpha","sd_ndt", 'sd_theta', 'sd_lambda
_right','sd_lambda_left', 'sd_delta_right','sd_delta_left', "alpha_sbj","threshold
_sbj","ndt_sbj",'theta_sbj', 'lambda_right_sbj','lambda_left_sbj','delta_right_sbj
', 'delta_left_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_lambda_right = runif(1,-1, 1),
                          sd_lambda_right = runif(1, 0, 1),
                          mu_lambda_left = runif(1,-1, 1),
                          sd_lambda_left = runif(1, 0, 1),
                          mu_delta_right = runif(1,-1, 1),
                          sd_delta_right = runif(1, 0, 1),
                          mu_delta_left = runif(1,-1, 1),
                          sd_delta_left = 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_lambda_right = runif(length(tids),-0.1,0.1),
                          z_lambda_left = runif(length(tids),-0.1,0.1),
                          z_delta_right = runif(length(tids),-0.1,0.1),
                          z_delta_left = runif(length(tids),-0.1,0.1)


    )
  }

  return(initList)
}
```

```
#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_lambda_right','transf_mu_lambda_left', 'transf_mu_delta_right','transf_mu_delta_left', 'sd_threshold',"sd_alpha","sd_ndt", 'sd_theta', 'sd_lambda_right','sd_lambda_left', 'sd_delta_right','sd_delta_left', "lp__"), inc_warmup = TRUE, nrow = 3)
```




```
pairs(dsamples, pars = c("transf_mu_alpha","transf_mu_threshold","transf_mu_ndt", "transf_mu_theta",'transf_mu_lambda_right','transf_mu_lambda_left', 'transf_mu_delta_right','transf_mu_delta_left', 'sd_threshold',"sd_alpha","sd_ndt", 'sd_theta', 'sd_lambda_right','sd_lambda_left', 'sd_delta_right','sd_delta_left', "lp__"))
```


```
print(dsamples, pars = c("transf_mu_alpha","transf_mu_threshold","transf_mu_ndt",
"transf_mu_theta",'transf_mu_lambda_right','transf_mu_lambda_left', 'transf_mu_del
ta_right','transf_mu_delta_left', 'sd_threshold',"sd_alpha","sd_ndt", 'sd_theta',
'sd_lambda_right','sd_lambda_left', 'sd_delta_right','sd_delta_left', "lp__"))
```

```
## Inference for Stan model: anon_model.
## 4 chains, each with iter=2000; warmup=1000; thin=1;
## post-warmup draws per chain=1000, total post-warmup draws=4000.
##
##                            mean se_mean    sd      2.5%       25%       50%
## transf_mu_alpha           -0.18    0.00  0.03     -0.24     -0.20     -0.18
## transf_mu_threshold        4.65    0.01  0.14      4.39      4.55      4.65
## transf_mu_ndt              1.39    0.01  0.10      1.19      1.32      1.38
## transf_mu_theta            0.02    0.00  0.00      0.02      0.02      0.02
## transf_mu_lambda_right   -18.06    0.03  1.24    -20.51    -18.89    -18.05
## transf_mu_lambda_left     16.69    0.03  1.20     14.32     15.87     16.71
## transf_mu_delta_right     -0.36    0.01  0.61     -1.55     -0.77     -0.35
## transf_mu_delta_left       0.77    0.01  0.64     -0.50      0.35      0.77
## sd_threshold               1.60    0.00  0.10      1.42      1.53      1.60
## sd_alpha                   0.16    0.00  0.04      0.07      0.14      0.16
## sd_ndt                     1.48    0.00  0.10      1.30      1.42      1.48
## sd_theta                   0.38    0.00  0.04      0.32      0.35      0.38
## sd_lambda_right           12.69    0.03  0.99     10.87     12.01     12.63
## sd_lambda_left            12.54    0.03  0.97     10.78     11.85     12.50
## sd_delta_right             5.17    0.02  0.63      4.03      4.73      5.14
## sd_delta_left              5.65    0.02  0.62      4.47      5.22      5.62
## lp__                  -30568.11    1.35 34.70 -30637.44 -30591.41 -30567.43
##                             75%     97.5% n_eff Rhat
## transf_mu_alpha           -0.16     -0.12  5815 1.00
## transf_mu_threshold        4.75      4.93   343 1.00
## transf_mu_ndt              1.45      1.58   215 1.01
## transf_mu_theta            0.02      0.02  2463 1.00
## transf_mu_lambda_right   -17.21    -15.64  1461 1.00
## transf_mu_lambda_left     17.51     19.02  1223 1.00
## transf_mu_delta_right      0.05      0.83  3364 1.00
## transf_mu_delta_left       1.20      2.04  3319 1.00
## sd_threshold               1.67      1.82   719 1.00
## sd_alpha                   0.19      0.25   946 1.00
## sd_ndt                     1.54      1.70   668 1.01
## sd_theta                   0.40      0.46  1664 1.00
## sd_lambda_right           13.34     14.74  1350 1.00
## sd_lambda_left            13.18     14.54  1357 1.00
## sd_delta_right             5.59      6.46  1631 1.00
## sd_delta_left              6.06      6.94  1506 1.00
## lp__                  -30544.18 -30502.37   660 1.00
##
## Samples were drawn using NUTS(diag_e) at Sun Jan 14 20:32:58 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_lambda_right','transf_mu_lambda_left', 'transf_mu_delta_right'
,'transf_mu_delta_left')

# 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)
```