```
############### 0 - safe choice A, 1 - risky choice B #####################
library(rstan); rstan_options(javascript=FALSE)
```

```
## 载入需要的程辑包：StanHeaders
```

```
##
## rstan version 2.26.23 (Stan version 2.26.1)
```

```
## For execution on a local, multicore CPU with excess RAM we recommend calling
## options(mc.cores = parallel::detectCores()).
## To avoid recompilation of unchanged Stan programs, we recommend calling
## rstan_options(auto_write = TRUE)
## For within-chain threading using `reduce_sum()` or `map_rect()` Stan functions,
## change `threads_per_chain` option:
## rstan_options(threads_per_chain = 1)
```

```
## Do not specify '-march=native' in 'LOCAL_CPPFLAGS' or a Makevars file
```

```
options(mc.cores = parallel::detectCores())
rstan_options(auto_write = T)

# Get list of files in 'data_2' folder with the pattern "riskytimed"
files <- dir(path = "data_2", pattern="riskytimed")

# Read all csv files in the list
data_list <- lapply(paste0("data_2/", files), read.table, header = TRUE, skip = 0,
fill = TRUE, sep= ";")

# Concatenate rows of all items in the list into a data frame
dat <- do.call("rbind", data_list)
```

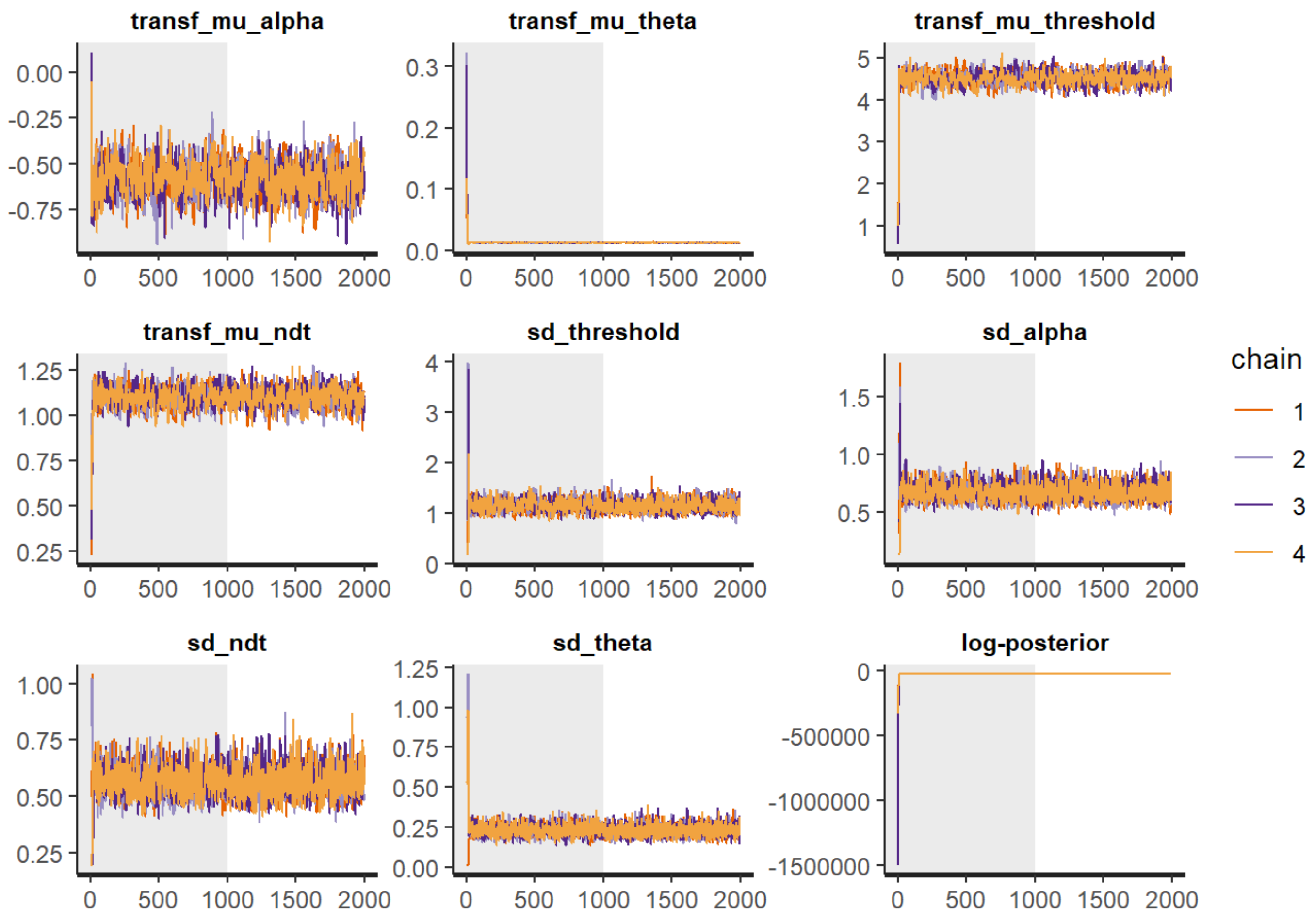
```
# gamble characteristics
dat$eva = dat$oa1*dat$pa1+dat$oa2*dat$pa2 + dat$oa3*dat$pa3+dat$oa4*dat$pa4
dat$evb = dat$ob1*dat$pb1+dat$ob2*dat$pb2 + dat$ob3*dat$pb3+dat$ob4*dat$pb4
dat$evd = dat$evb - dat$eva
dat$sda = sqrt((dat$oa1-dat$eva)^2*dat$pa1 + (dat$oa2-dat$eva)^2*dat$pa2 + (dat$oa
3-dat$eva)^2*dat$pa3 + (dat$oa4-dat$eva)^2*dat$pa4)
dat$sdb = sqrt((dat$ob1-dat$evb)^2*dat$pb1 + (dat$ob2-dat$evb)^2*dat$pb2 + (dat$ob
3-dat$evb)^2*dat$pb3 + (dat$ob4-dat$evb)^2*dat$pb4)
dat$sdd = dat$sdb - dat$sda
dat$evdummy = ifelse(dat$evd>0,1,0)
```

```r
# transform to +/- 1; safe - 1, risky +1
dat$cho <- ifelse(dat$choice==0,-1,ifelse(dat$choice==1,1,NA))
ids <- unique(dat$id)
for(j in 1:length(ids)){
  dat$tid[dat$id==ids[j]] <- j
}
tids <- unique(dat$tid)
# only control data
control_dat <- dat[dat$cond=="control",]
# remove fast RTs
rcontrol_dat <- control_dat[control_dat$rt>1,]
# only condition no time pressure
dataList  = list(cho = rcontrol_dat$cho, rt = rcontrol_dat$rt, participant = rcont
rol_dat$tid, N=nrow(rcontrol_dat),  L = length(tids), starting_point=0.5, evd = rc
ontrol_dat$evd, sdd = rcontrol_dat$sdd)
```

```r
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_s
bj","ndt_sbj",'theta_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, 0.5),
                          sd_threshold = runif(1,0,1),
                          mu_ndt = runif(1, -1.5, 0),
                          sd_ndt = runif(1, 0, 1),
                          mu_theta = runif(1,-0.5, -0.5),
                          sd_theta = 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)


    )
  }

  return(initList)
}
```

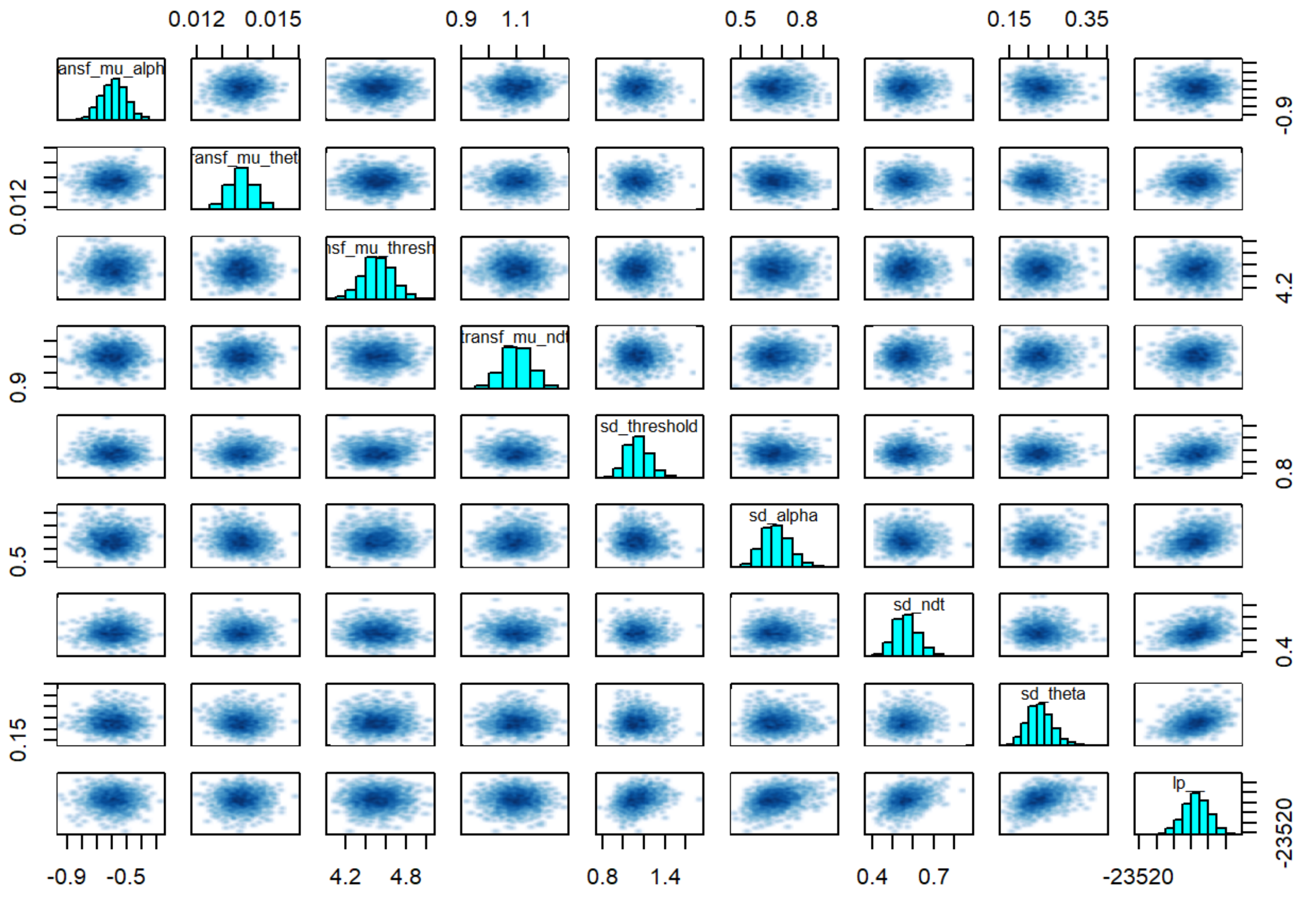
```
m <- stan_model("MV_Baseline.stan")
dsamples <- sampling(m,
                data=dataList,
                pars=parameters,
                iter=2000,
                chains=4,#If not specified, gives random inits
                init = initFunc(4),
                warmup = 1000,  # Stands for burn-in; 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_m
u_theta",'sd_threshold',"sd_alpha","sd_ndt", 'sd_theta', "alpha_sbj","threshold_sb
j","ndt_sbj",'theta_sbj',"log_lik")

rstan::traceplot(dsamples, pars=c("transf_mu_alpha","transf_mu_theta", "transf_mu_
threshold","transf_mu_ndt", 'sd_threshold',"sd_alpha","sd_ndt", 'sd_theta', "lp_
_"), inc_warmup = TRUE, nrow = 3)
```



```
pairs(dsamples, pars = c("transf_mu_alpha","transf_mu_theta","transf_mu_threshol
d","transf_mu_ndt", 'sd_threshold',"sd_alpha","sd_ndt", 'sd_theta', "lp__"))
```

```
## Warning in par(usr): argument 1 does not name a graphical parameter

## Warning in par(usr): argument 1 does not name a graphical parameter

## Warning in par(usr): argument 1 does not name a graphical parameter

## Warning in par(usr): argument 1 does not name a graphical parameter

## Warning in par(usr): argument 1 does not name a graphical parameter

## Warning in par(usr): argument 1 does not name a graphical parameter

## Warning in par(usr): argument 1 does not name a graphical parameter

## Warning in par(usr): argument 1 does not name a graphical parameter

## Warning in par(usr): argument 1 does not name a graphical parameter
```



```
print(dsamples, pars = c("transf_mu_alpha", "transf_mu_theta", "transf_mu_threshol
d","transf_mu_ndt", 'sd_threshold',"sd_alpha","sd_ndt", 'sd_theta', "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         -0.59    0.00  0.09     -0.76     -0.65     -0.59
## transf_mu_theta          0.01    0.00  0.00      0.01      0.01      0.01
## transf_mu_threshold      4.52    0.01  0.15      4.23      4.42      4.52
## transf_mu_ndt            1.10    0.00  0.05      1.00      1.07      1.10
## sd_threshold             1.14    0.00  0.11      0.95      1.07      1.14
## sd_alpha                 0.67    0.00  0.07      0.55      0.62      0.67
## sd_ndt                   0.57    0.00  0.06      0.46      0.53      0.56
## sd_theta                 0.23    0.00  0.03      0.17      0.21      0.23
## lp__                -23454.41    0.62 15.72 -23486.53 -23464.41 -23454.21
##                           75%     97.5% n_eff Rhat
## transf_mu_alpha         -0.53     -0.41   517 1.00
## transf_mu_theta          0.01      0.01  2690 1.00
## transf_mu_threshold      4.62      4.83   463 1.00
## transf_mu_ndt            1.13      1.20   563 1.01
## sd_threshold             1.21      1.38   888 1.00
## sd_alpha                 0.72      0.82  1054 1.00
## sd_ndt                   0.60      0.70   958 1.00
## sd_theta                 0.25      0.30  1783 1.00
## lp__                -23443.74 -23424.85   653 1.01
##
## Samples were drawn using NUTS(diag_e) at Tue Oct  3 23:17:58 2023.
## 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(bayesplot)
```

```
## This is bayesplot version 1.10.0
```

```
## - Online documentation and vignettes at mc-stan.org/bayesplot
```

```
## - bayesplot theme set to bayesplot::theme_default()
```

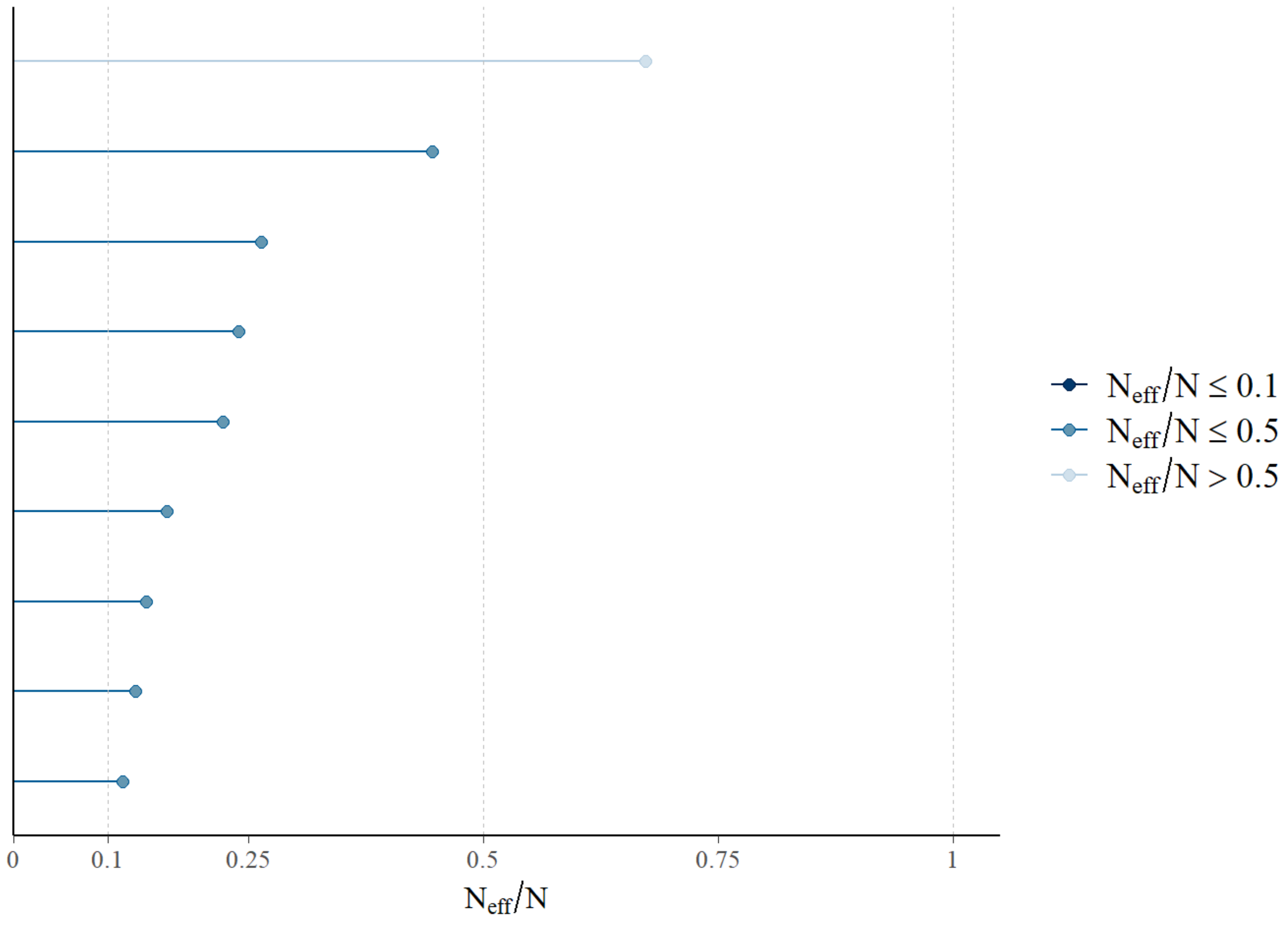
```
##    * Does _not_ affect other ggplot2 plots
```

```
##    * See ?bayesplot_theme_set for details on theme setting
```

```
ratios_cp <- neff_ratio(dsamples, pars = c("transf_mu_alpha","transf_mu_theta", "t
ransf_mu_threshold","transf_mu_ndt", 'sd_threshold',"sd_alpha","sd_ndt", 'sd_thet
a', "lp__"))
df_ratios_cp <- as.data.frame(ratios_cp)
print(df_ratios_cp)
```

```
##                       ratios_cp
## transf_mu_alpha       0.1291516
## transf_mu_theta       0.6725804
## transf_mu_threshold   0.1156301
## transf_mu_ndt         0.1406975
## sd_threshold          0.2221016
## sd_alpha              0.2633997
## sd_ndt                0.2395300
## sd_theta              0.4457589
## lp__                  0.1632010
```

```
mcmc_neff(ratios_cp, size = 2)
```


N_{eff}/N
$N_{eff}/N \leq 0.1$
$N_{eff}/N \leq 0.5$
$N_{eff}/N > 0.5$