```
library(rstan)
```

```
## Loading required package: StanHeaders
```

```
## Loading required package: ggplot2
```

```
## rstan (Version 2.21.8, GitRev: 2e1f913d3ca3)
```

```
## 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)
```

```
library(RWiener)


#original parameter values
th =  4.52
ndt =   1.09
beta =  -1
theta = .04
alpha = -0.59


stim = read.csv('Switching-Gambles.csv')

# gamble characteristics
  stim$eva = stim$payoffa1*stim$proba1+stim$payoffa2*stim$proba2

  stim$evb = stim$payoffb1*stim$probb1+stim$payoffb2*stim$probb2
  stim$evd = stim$evb-stim$eva
  stim$sda = sqrt((stim$payoffa1-stim$eva)^2*stim$proba1 + (stim$payoffa2-stim$ev
a)^2*stim$proba2)
  stim$sdb = sqrt((stim$payoffb1-stim$evb)^2*stim$probb1 + (stim$payoffb2-stim$ev
b)^2*stim$probb2)
  stim$sdd = stim$sdb - stim$sda




stim2 = read.csv('Switching-Gambles.csv')
stim3 = read.csv('Switching-Gambles.csv')

# gamble characteristics
  stim3$eva = stim$payoffa1*stim$proba1+stim$payoffa2*stim$proba2
```

```
  stim3$evb = stim$payoffb1*stim$probb1+stim$payoffb2*stim$probb2
  stim3$evd = stim$evb-stim$eva
  stim3$sda = sqrt((stim$payoffa1-stim$eva)^2*stim$proba1 + (stim$payoffa2-stim$ev
a)^2*stim$proba2)
  stim3$sdb = sqrt((stim$payoffb1-stim$evb)^2*stim$probb1 + (stim$payoffb2-stim$ev
b)^2*stim$probb2)
  stim3$sdd = stim$sdb - stim$sda


for(n in 1:nrow(stim2)){

    stim2$simchosum[n] = 0
}

stim4 = read.csv('Switching-Gambles.csv')
for(n in 1:nrow(stim4)){

    stim4$simchosum[n] = 0
}
```

```
sim_ddm <- "
data {
    int<lower=1> N;                                 // number of data items
    int<lower=1> L;                                 // number of participants

    int<lower=-1,upper=1> cho[N];               // accuracy (-1, 1)
    real<lower=0> rt[N];                            // rt
    real evd[N];
    real sdd[N];
    real<lower=0, upper=1> starting_point;          // starting point diffusion mo
del not to estimate
}

parameters {

    real alpha_sbj;
    real theta_v;
    real threshold_v;
    real ndt_v;
    real mu_beta;     // discounting effect parameter

}
transformed parameters {
    real drift_ll[N];                               // trial-by-trial drift rate f
or likelihood (incorporates accuracy)
    real drift_t[N];                                // trial-by-trial drift rate f
or predictions
    real<lower=0> threshold_t[N];                   // trial-by-trial threshold
    real<lower=0> ndt_t[N];                         // trial-by-trial ndt
```

```
    real<lower=0> theta_sbj;
    real<lower=0> threshold_sbj;
    real<lower=0> ndt_sbj;





    theta_sbj = log(1 + exp(theta_v));
    threshold_sbj = log(1 + exp(threshold_v));
    ndt_sbj = log(1 + exp(ndt_v));


    for (n in 1:N) {
        drift_t[n] = theta_sbj * (evd[n] +mu_beta+ alpha_sbj * sdd[n]);
        drift_ll[n] = drift_t[n]*cho[n];
        threshold_t[n] = threshold_sbj;
        ndt_t[n] = ndt_sbj;

    }
}
model {
    alpha_sbj ~ normal(0, 5);
    theta_v ~ normal(0,5);
    threshold_v ~ normal(0,5);
    ndt_v ~ normal(0,5);
    mu_beta ~ normal(0, 5);



    rt ~ wiener(threshold_t, ndt_t, starting_point, drift_ll);
}
generated quantities {
    vector[N] log_lik;

    {for (n in 1:N) {
        log_lik[n] = wiener_lpdf(rt[n] | threshold_t[n], ndt_t[n], starting_point,
drift_ll[n]);
    }
}
}

"
```

```
initFunc <-function (i) {
  initList=list()
  for (ll in 1:i){
    initList[[ll]] = list(
                          alpha_sbj = runif(1,-5,5),
                          theta_v = runif(1,-6,1),
                          threshold_v = runif(1,-0.5,10),
                          ndt_v = runif(1,-1.5,0),
                          mu_beta = runif(1,-5,5)
    )
  }
  return(initList)
}
```

```
# Set the number of iterations
n_iter <- 100


`%+=%` = function(e1,e2) eval.parent(substitute(e1 <- e1 + e2))



# Create empty vectors to store the outcome parameters for each iteration
th_recover <- numeric(n_iter)
theta_recover <- numeric(n_iter)
ndt_recover <- numeric(n_iter)
alpha_recover <- numeric(n_iter)
beta_recover <- numeric(n_iter)

th_bias <- numeric(n_iter)
theta_bias <- numeric(n_iter)
ndt_bias <- numeric(n_iter)
alpha_bias <- numeric(n_iter)
beta_bias <- numeric(n_iter)

th_dev <- numeric(n_iter)
theta_dev <- numeric(n_iter)
ndt_dev <- numeric(n_iter)
alpha_dev <- numeric(n_iter)
beta_dev <- numeric(n_iter)

# Run the model for n_iter iterations
for (i in 1:n_iter) {


  for(n in 1:nrow(stim)){
    cres <- rwiener(1, th, ndt, 0.5, theta * (stim$evd[n] + beta + alpha * stim$sd
d[n]))
    stim$simrt[n] <- as.numeric(cres[1])
```

```
    stim$simcho[n] <- ifelse(cres[2]=="upper",1,-1)
    stim$cho2[n] <- ifelse(stim$simcho[n] == 1, 0, ifelse(stim$simcho[n] == -1, 1,
NA))

  }



  for(n in 1:nrow(stim2)){

    stim2$simchosum[n]  %+=% ifelse(stim$simcho[n]==1,1,0)
    }




  parameters = c("alpha_sbj","threshold_sbj","ndt_sbj",'theta_sbj', 'mu_beta')
  dataList  = list(cho = stim$simcho, starting_point = 0.5, rt = stim$simrt, N=60,
L = 1,  evd = stim$evd, sdd = stim$sdd)





  # Run the diffusion model for the current iteration
  dsamples <- stan(model_code = sim_ddm,
                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
                refresh = 0
                )

  samples <- rstan::extract(dsamples, pars = c('alpha_sbj', 'theta_sbj', 'threshol
d_sbj', 'ndt_sbj', 'mu_beta'))


  # Store the outcome parameters for the current iteration
  th_recover[i] <- mean(samples$threshold_sbj)
  theta_recover[i] <- mean(samples$theta_sbj)
  ndt_recover[i] <- mean(samples$ndt_sbj)
  alpha_recover[i] <- mean(samples$alpha_sbj)
  beta_recover[i] <- mean(samples$mu_beta)


  th_bias[i] <- (mean(samples$threshold_sbj)-th)/th
  theta_bias[i] <- (mean(samples$theta_sbj)-theta)/theta
  ndt_bias[i] <- (mean(samples$ndt_sbj)-ndt)/ndt
```

```
  alpha_bias[i] <- (mean(samples$alpha_sbj)-alpha)/alpha
  beta_bias[i] <- (mean(samples$mu_beta)-beta)/beta


  th_dev[i] <- abs(mean(samples$threshold_sbj)-th)/th
  theta_dev[i] <- abs(mean(samples$theta_sbj)-theta)/theta
  ndt_dev[i] <- abs(mean(samples$ndt_sbj)-ndt)/ndt
  alpha_dev[i] <- abs(mean(samples$alpha_sbj)-alpha)/alpha
  beta_dev[i] <- abs(mean(samples$mu_beta)-beta)/beta




}
```

```
#create a summary df of all parameters
df_summary <- data.frame(original_th = th,
                 recovered_th = th_recover,
                 bias_th = th_bias,
                 deviation_th = th_dev,
                 original_theta = theta,
                 recovered_theta = theta_recover,
                 bias_theta = theta_bias,
                 deviation_theta = theta_dev,
                 original_ndt = ndt,
                 recovered_ndt = ndt_recover,
                 bias_ndt = ndt_bias,
                 deviation_ndt = ndt_dev,
                 original_alpha = alpha,
                 recovered_alpha = alpha_recover,
                 bias_alpha = alpha_bias,
                 deviation_alpha = alpha_dev,
                 original_beta = beta,
                 recovered_beta = beta_recover,
                 bias_beta = beta_bias,
                 deviation_beta = beta_dev

                 )
```

```
#create a table to show all means and true values
df_mean <- data.frame(parameter = c('th', "theta", "ndt", "alpha","beta"),
                      true_value = c(th, theta,ndt, alpha, beta),
                      mean_recovered = c(mean(df_summary$recovered_th), mean(df_su
mmary$recovered_theta),mean(df_summary$recovered_ndt),mean(df_summary$recovered_al
pha), mean(df_summary$recovered_beta)),
                      mean_bias = c(mean(df_summary$bias_th), mean(df_summary$bias
_theta),mean(df_summary$bias_ndt),mean(df_summary$bias_alpha), mean(df_summary$bia
s_beta)),
                      mean_deviation = c(mean(df_summary$deviation_th), mean(df_su
mmary$deviation_theta),mean(df_summary$deviation_ndt),mean(df_summary$deviation_al
pha), mean(df_summary$deviation_beta))
                 )
df_mean
```

```
##   parameter true_value mean_recovered   mean_bias mean_deviation
## 1        th       4.52     4.59180630  0.01588635     0.06280041
## 2     theta       0.04     0.03909366 -0.02265842     0.16019338
## 3       ndt       1.09     1.06141995 -0.02622023     0.10227418
## 4     alpha      -0.59    -0.66303200  0.12378305    -0.24660512
## 5      beta      -1.00    -0.63507352 -0.36492648    -2.03873307
```

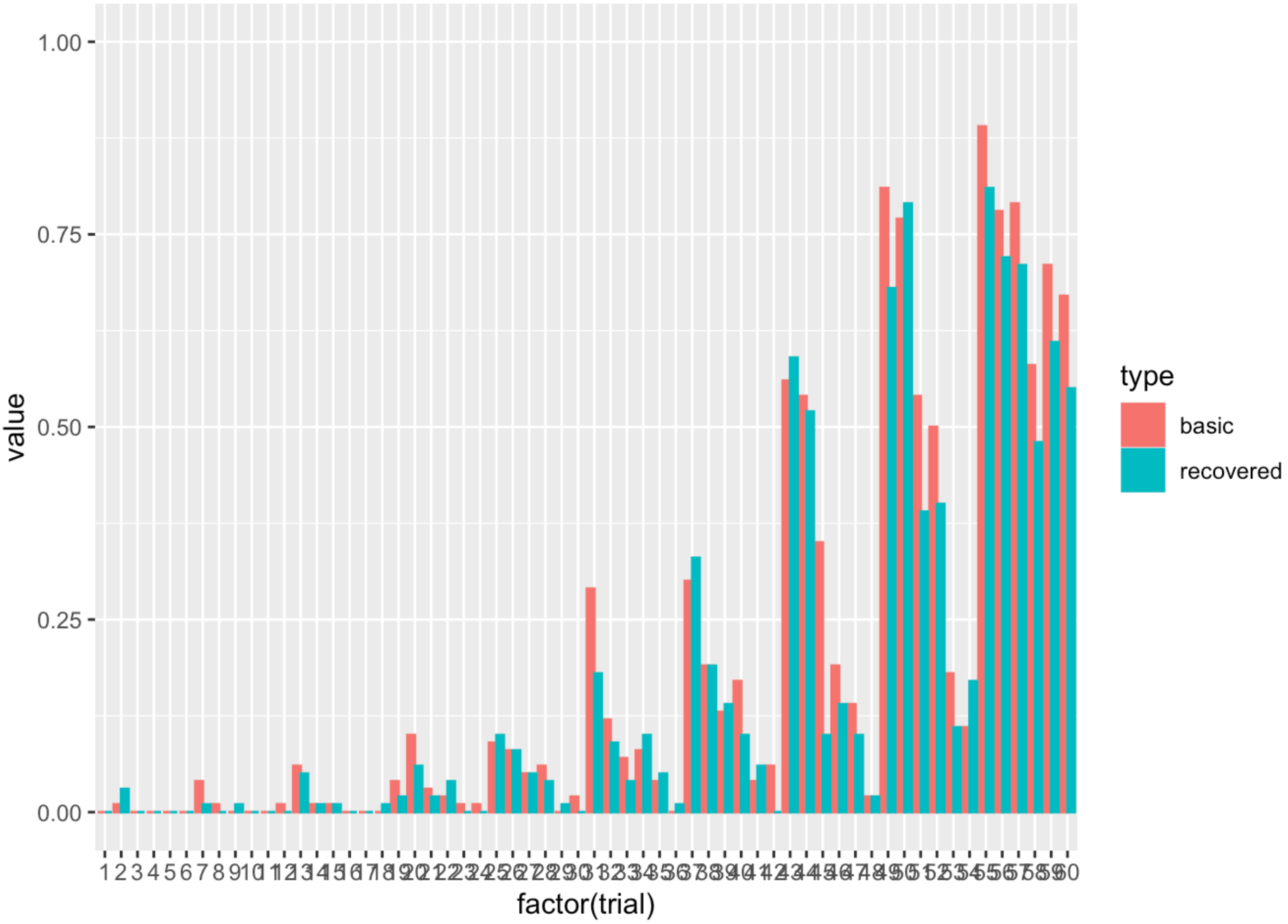
```
df_median <- data.frame(parameter = c('th', "theta", "ndt", "alpha","beta"),
                      true_value = c(th, theta,ndt, alpha, beta),
                      median_recovered = c(median(df_summary$recovered_th), media
n(df_summary$recovered_theta),median(df_summary$recovered_ndt),median(df_summary$r
ecovered_alpha), median(df_summary$recovered_beta))
                      )

df_median
```

```
##   parameter true_value median_recovered
## 1        th       4.52       4.58493131
## 2     theta       0.04       0.03839063
## 3       ndt       1.09       1.04122366
## 4     alpha      -0.59      -0.63541222
## 5      beta      -1.00      -0.62258991
```

```
#check whether the risky choice proportion can be successfully recovered by the me
an-variance model
#firstly, use recovered parameter values to simulation choice data
for (i in 1:n_iter) {

  for(n in 1:nrow(stim3)){
      cres <- rwiener(1,mean(df_summary$recovered_th), mean(df_summary$recovered_n
dt), 0.5, mean(df_summary$recovered_theta) * ((stim3$evd[n]+df_summary$recovered_b
eta) + mean(df_summary$recovered_alpha) * stim3$sdd[n]))
      stim3$simrt[n] <- as.numeric(cres[1])
      stim3$simcho[n] <- ifelse(cres[2]=="upper",1,-1)
      stim3$cho2[n] <- ifelse(stim3$simcho[n] == 1, 0, ifelse(stim3$simcho[n] == -
1, 1, NA))

  }

  for(n in 1:nrow(stim4)){

    stim4$simchosum[n]  %+=% ifelse(stim3$simcho[n]==1,1,0)
    }
}
```

```
#create summary dataframe
label <- c(rep("basic", 60), rep("recovered", 60))
df <- data.frame(trial = rep(stim2$num),
                 value = c(stim2$simchosum/n_iter, stim4$simchosum/n_iter),
                 type = rep(label))
#display the first n trials
subset_data <- df[df$trial <= 60, ]
```

```
ggplot(subset_data, aes(x = factor(trial), y = value, fill = type, colour = type))
+
  geom_bar(stat = "identity", position = "dodge")+
  ylim(0,1)
```




```
library(rstan)


library(RWiener)


#original parameter values
th =  4.52
ndt =   1.09
beta =  -3
theta = .04
alpha = -0.59


stim = read.csv('Switching-Gambles.csv')

# gamble characteristics
  stim$eva = stim$payoffa1*stim$proba1+stim$payoffa2*stim$proba2

  stim$evb = stim$payoffb1*stim$probb1+stim$payoffb2*stim$probb2
  stim$evd = stim$evb-stim$eva
  stim$sda = sqrt((stim$payoffa1-stim$eva)^2*stim$proba1 + (stim$payoffa2-stim$ev
a)^2*stim$proba2)
```

```
  stim$sdb = sqrt((stim$payoffb1-stim$evb)^2*stim$probb1 + (stim$payoffb2-stim$ev
b)^2*stim$probb2)
  stim$sdd = stim$sdb - stim$sda




stim2 = read.csv('Switching-Gambles.csv')
stim3 = read.csv('Switching-Gambles.csv')

# gamble characteristics
  stim3$eva = stim$payoffa1*stim$proba1+stim$payoffa2*stim$proba2

  stim3$evb = stim$payoffb1*stim$probb1+stim$payoffb2*stim$probb2
  stim3$evd = stim$evb-stim$eva
  stim3$sda = sqrt((stim$payoffa1-stim$eva)^2*stim$proba1 + (stim$payoffa2-stim$ev
a)^2*stim$proba2)
  stim3$sdb = sqrt((stim$payoffb1-stim$evb)^2*stim$probb1 + (stim$payoffb2-stim$ev
b)^2*stim$probb2)
  stim3$sdd = stim$sdb - stim$sda


for(n in 1:nrow(stim2)){

    stim2$simchosum[n] = 0
}

stim4 = read.csv('Switching-Gambles.csv')
for(n in 1:nrow(stim4)){

    stim4$simchosum[n] = 0
}
```

```
initFunc <-function (i) {
  initList=list()
  for (ll in 1:i){
    initList[[ll]] = list(
                          alpha_sbj = runif(1,-5,5),
                          theta_v = runif(1,-6,1),
                          threshold_v = runif(1,-0.5,10),
                          ndt_v = runif(1,-1.5,0),
                          mu_beta = runif(1,-5,5)
    )
  }
  return(initList)
}
```

```
# Set the number of iterations
n_iter <- 100
```

```
`%+=%` = function(e1,e2) eval.parent(substitute(e1 <- e1 + e2))



# Create empty vectors to store the outcome parameters for each iteration
th_recover <- numeric(n_iter)
theta_recover <- numeric(n_iter)
ndt_recover <- numeric(n_iter)
alpha_recover <- numeric(n_iter)
beta_recover <- numeric(n_iter)

th_bias <- numeric(n_iter)
theta_bias <- numeric(n_iter)
ndt_bias <- numeric(n_iter)
alpha_bias <- numeric(n_iter)
beta_bias <- numeric(n_iter)

th_dev <- numeric(n_iter)
theta_dev <- numeric(n_iter)
ndt_dev <- numeric(n_iter)
alpha_dev <- numeric(n_iter)
beta_dev <- numeric(n_iter)

# Run the model for n_iter iterations
for (i in 1:n_iter) {


  for(n in 1:nrow(stim)){
    cres <- rwiener(1, th, ndt, 0.5, theta * (stim$evd[n]+beta + alpha * stim$sdd[
n]))
    stim$simrt[n] <- as.numeric(cres[1])
    stim$simcho[n] <- ifelse(cres[2]=="upper",1,-1)
    stim$cho2[n] <- ifelse(stim$simcho[n] == 1, 0, ifelse(stim$simcho[n] == -1, 1,
NA))

  }



  for(n in 1:nrow(stim2)){

    stim2$simchosum[n]  %+=% ifelse(stim$simcho[n]==1,1,0)
    }




  parameters = c("alpha_sbj","threshold_sbj","ndt_sbj",'theta_sbj', 'mu_beta')
```

```
  dataList  = list(cho = stim$simcho, starting_point = 0.5, rt = stim$simrt, N=60,
L = 1,  evd = stim$evd, sdd = stim$sdd)






  # Run the diffusion model for the current iteration
  dsamples <- stan(model_code = sim_ddm,
                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
                refresh = 0
                )

  samples <- rstan::extract(dsamples, pars = c('alpha_sbj', 'theta_sbj', 'threshol
d_sbj', 'ndt_sbj', 'mu_beta'))


  # Store the outcome parameters for the current iteration
  th_recover[i] <- mean(samples$threshold_sbj)
  theta_recover[i] <- mean(samples$theta_sbj)
  ndt_recover[i] <- mean(samples$ndt_sbj)
  alpha_recover[i] <- mean(samples$alpha_sbj)
  beta_recover[i] <- mean(samples$mu_beta)


  th_bias[i] <- (mean(samples$threshold_sbj)-th)/th
  theta_bias[i] <- (mean(samples$theta_sbj)-theta)/theta
  ndt_bias[i] <- (mean(samples$ndt_sbj)-ndt)/ndt
  alpha_bias[i] <- (mean(samples$alpha_sbj)-alpha)/alpha
  beta_bias[i] <- (mean(samples$mu_beta)-beta)/beta


  th_dev[i] <- abs(mean(samples$threshold_sbj)-th)/th
  theta_dev[i] <- abs(mean(samples$theta_sbj)-theta)/theta
  ndt_dev[i] <- abs(mean(samples$ndt_sbj)-ndt)/ndt
  alpha_dev[i] <- abs(mean(samples$alpha_sbj)-alpha)/alpha
  beta_dev[i] <- abs(mean(samples$mu_beta)-beta)/beta




}
```

```
#create a summary df of all parameters
df_summary <- data.frame(original_th = th,
                 recovered_th = th_recover,
                 bias_th = th_bias,
                 deviation_th = th_dev,
                 original_theta = theta,
                 recovered_theta = theta_recover,
                 bias_theta = theta_bias,
                 deviation_theta = theta_dev,
                 original_ndt = ndt,
                 recovered_ndt = ndt_recover,
                 bias_ndt = ndt_bias,
                 deviation_ndt = ndt_dev,
                 original_alpha = alpha,
                 recovered_alpha = alpha_recover,
                 bias_alpha = alpha_bias,
                 deviation_alpha = alpha_dev,
                 original_beta = beta,
                 recovered_beta = beta_recover,
                 bias_beta = beta_bias,
                 deviation_beta = beta_dev

                 )
```

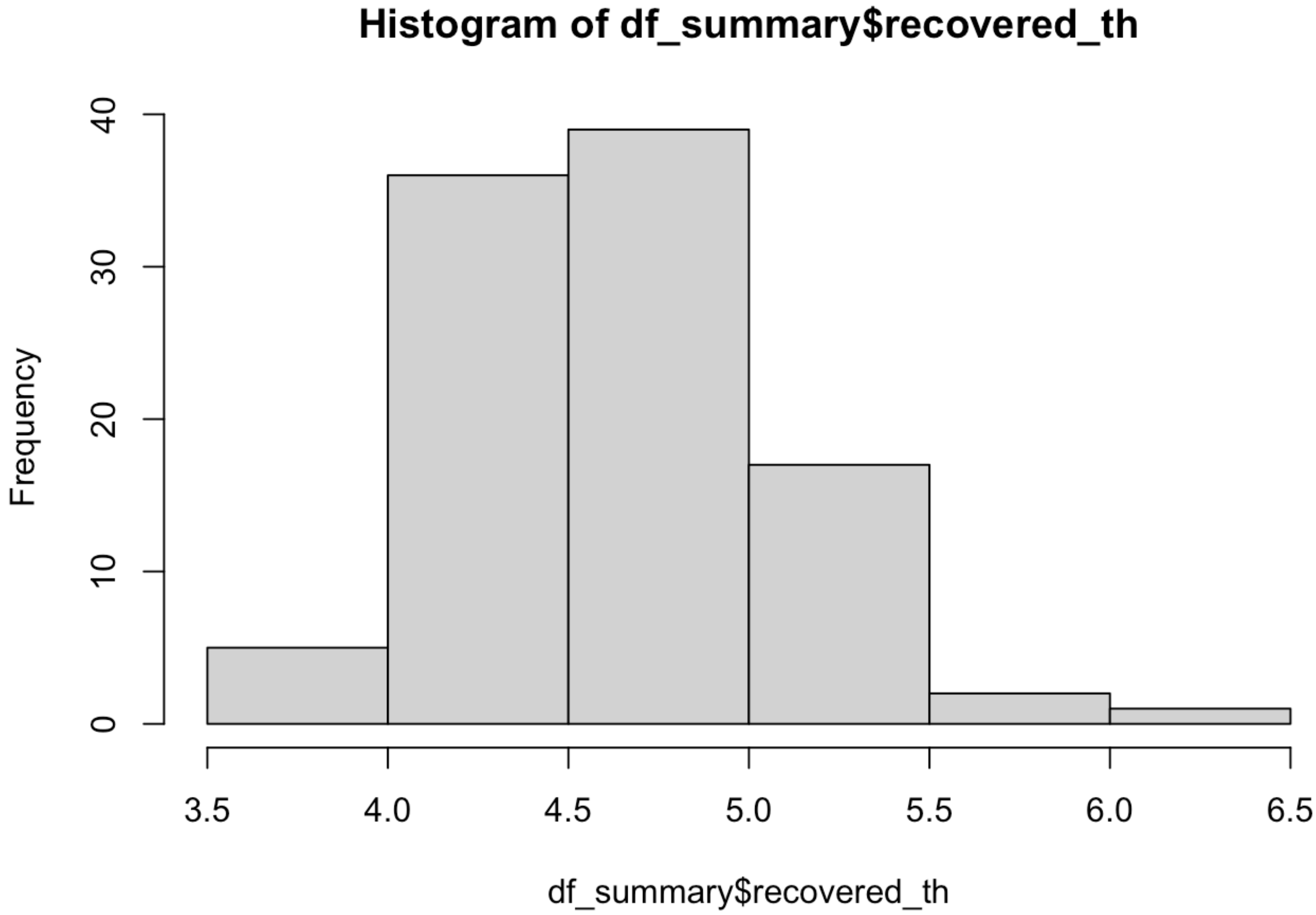
```
#create a table to show all means and true values
df_mean <- data.frame(parameter = c('th', "theta", "ndt", "alpha","beta"),
                      true_value = c(th, theta,ndt, alpha, beta),
                      mean_recovered = c(mean(df_summary$recovered_th), mean(df_su
mmary$recovered_theta),mean(df_summary$recovered_ndt),mean(df_summary$recovered_al
pha), mean(df_summary$recovered_beta)),
                      mean_bias = c(mean(df_summary$bias_th), mean(df_summary$bias
_theta),mean(df_summary$bias_ndt),mean(df_summary$bias_alpha), mean(df_summary$bia
s_beta)),
                      mean_deviation = c(mean(df_summary$deviation_th), mean(df_su
mmary$deviation_theta),mean(df_summary$deviation_ndt),mean(df_summary$deviation_al
pha), mean(df_summary$deviation_beta))
                 )
df_mean
```

```
##   parameter true_value mean_recovered   mean_bias mean_deviation
## 1        th       4.52     4.62156123  0.02246930     0.07510133
## 2     theta       0.04     0.04009834  0.00245849     0.15591037
## 3       ndt       1.09     1.07593687 -0.01290196     0.10492037
## 4     alpha      -0.59    -0.71711736  0.21545316    -0.27924795
## 5      beta      -3.00    -1.01693672 -0.66102109    -0.80282666
```

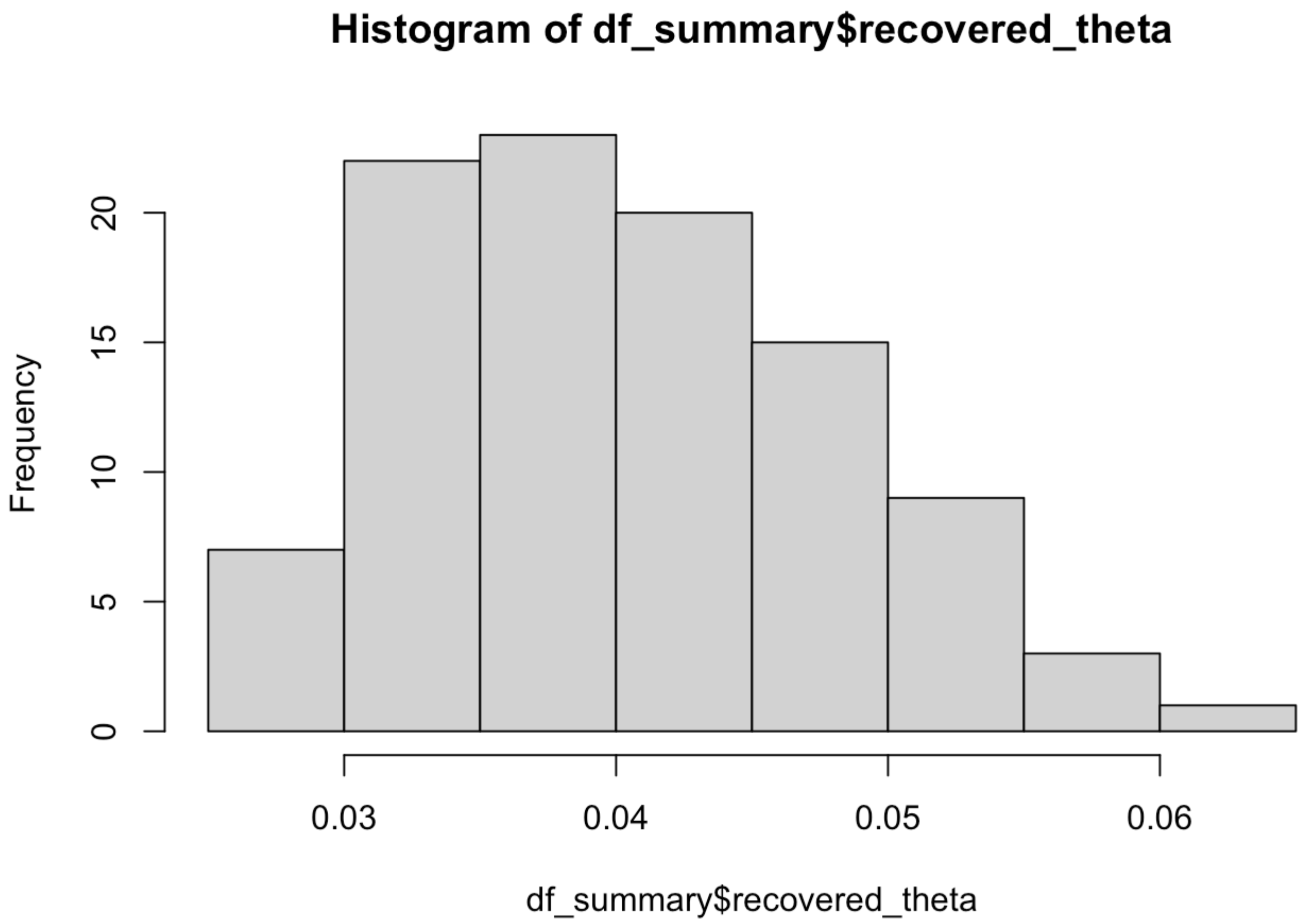
```
df_median <- data.frame(parameter = c('th', "theta", "ndt", "alpha","beta"),
                      true_value = c(th, theta,ndt, alpha, beta),
                      median_recovered = c(median(df_summary$recovered_th), media
n(df_summary$recovered_theta),median(df_summary$recovered_ndt),median(df_summary$r
ecovered_alpha), median(df_summary$recovered_beta))
                      )

df_median
```

```
##   parameter true_value median_recovered
## 1        th       4.52       4.62299033
## 2     theta       0.04       0.03861467
## 3       ndt       1.09       1.06684646
## 4     alpha      -0.59      -0.70591223
## 5      beta      -3.00      -0.89936916
```

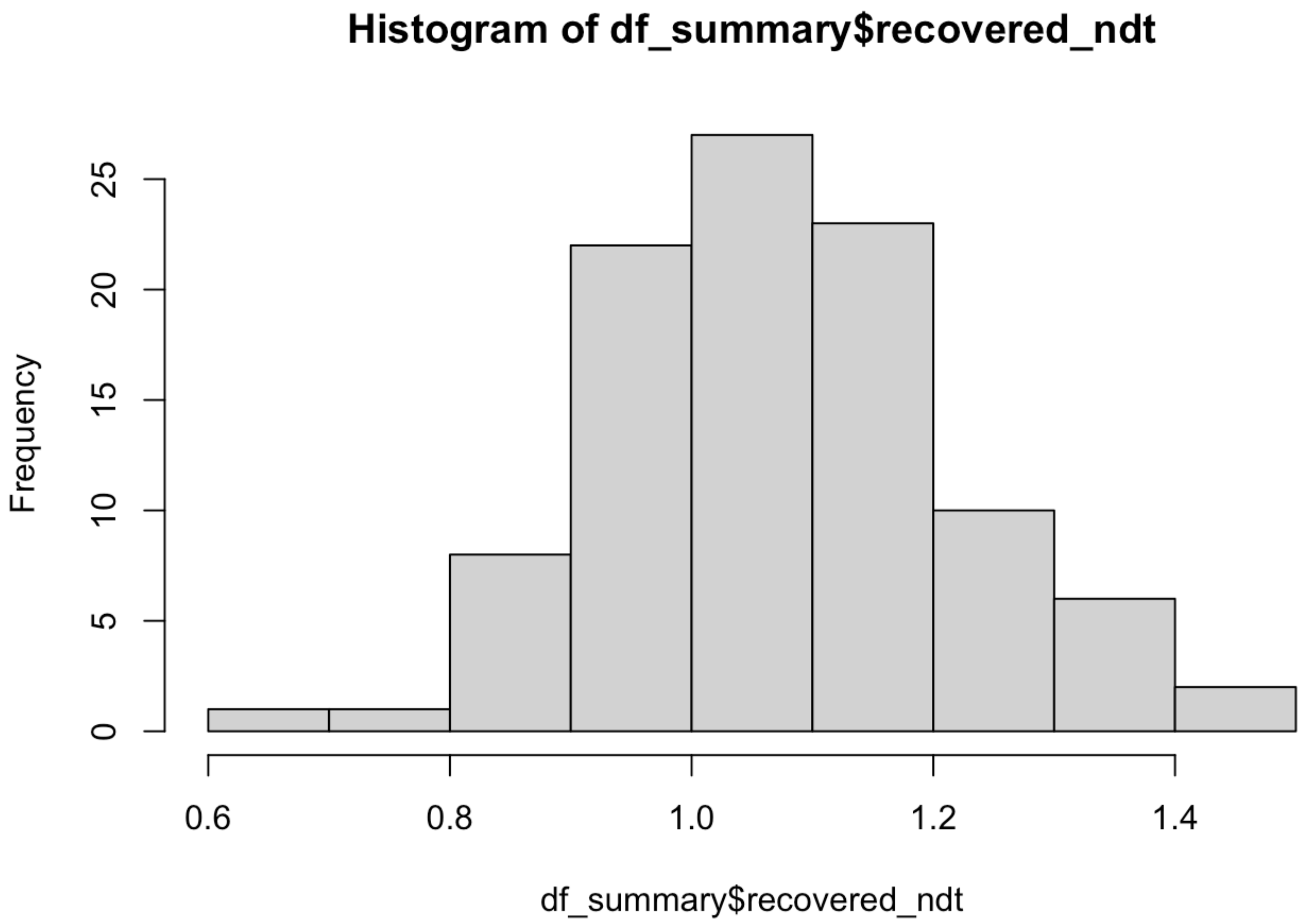
```
hist(df_summary$recovered_th)
```



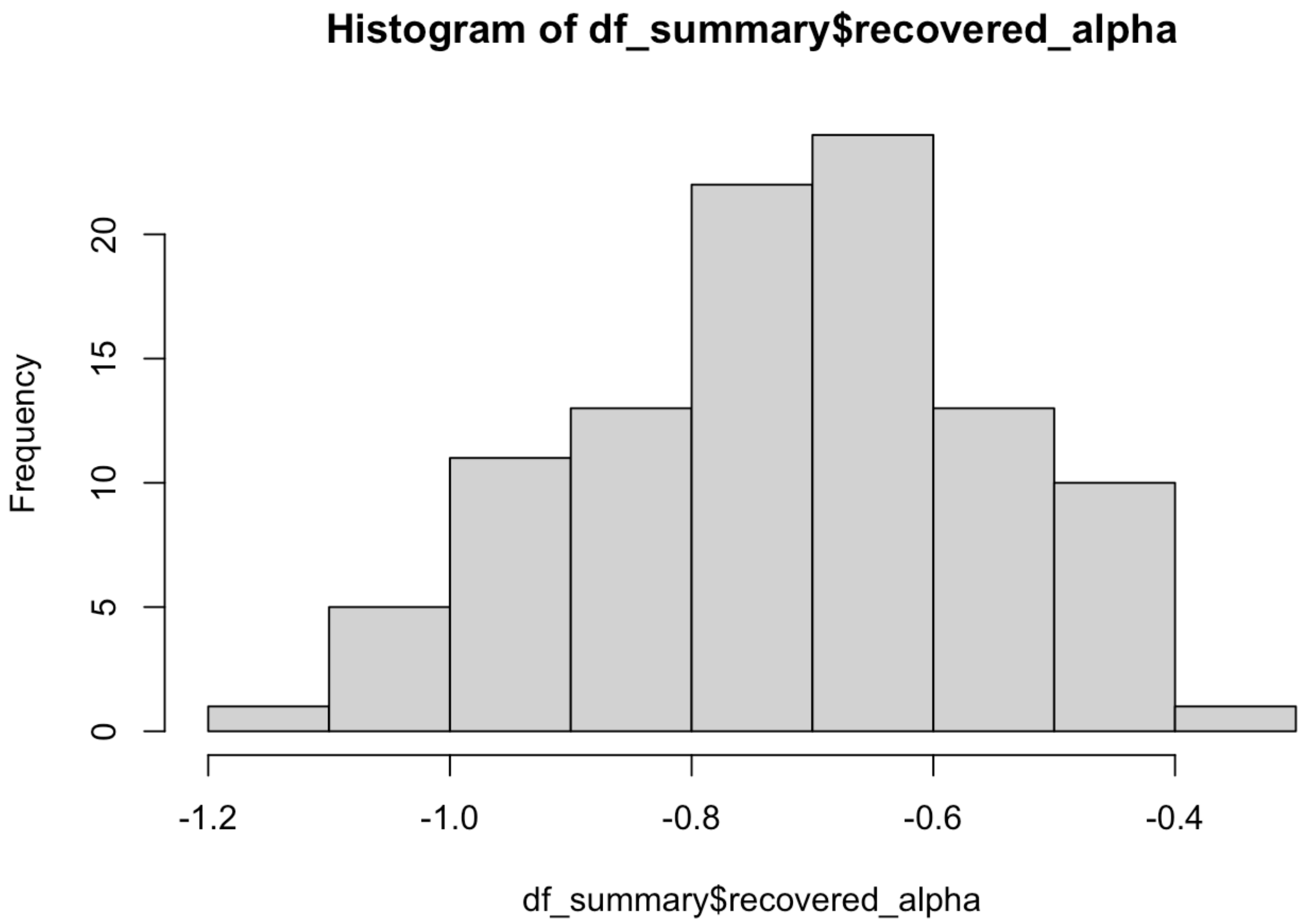

```
hist(df_summary$recovered_theta)
```



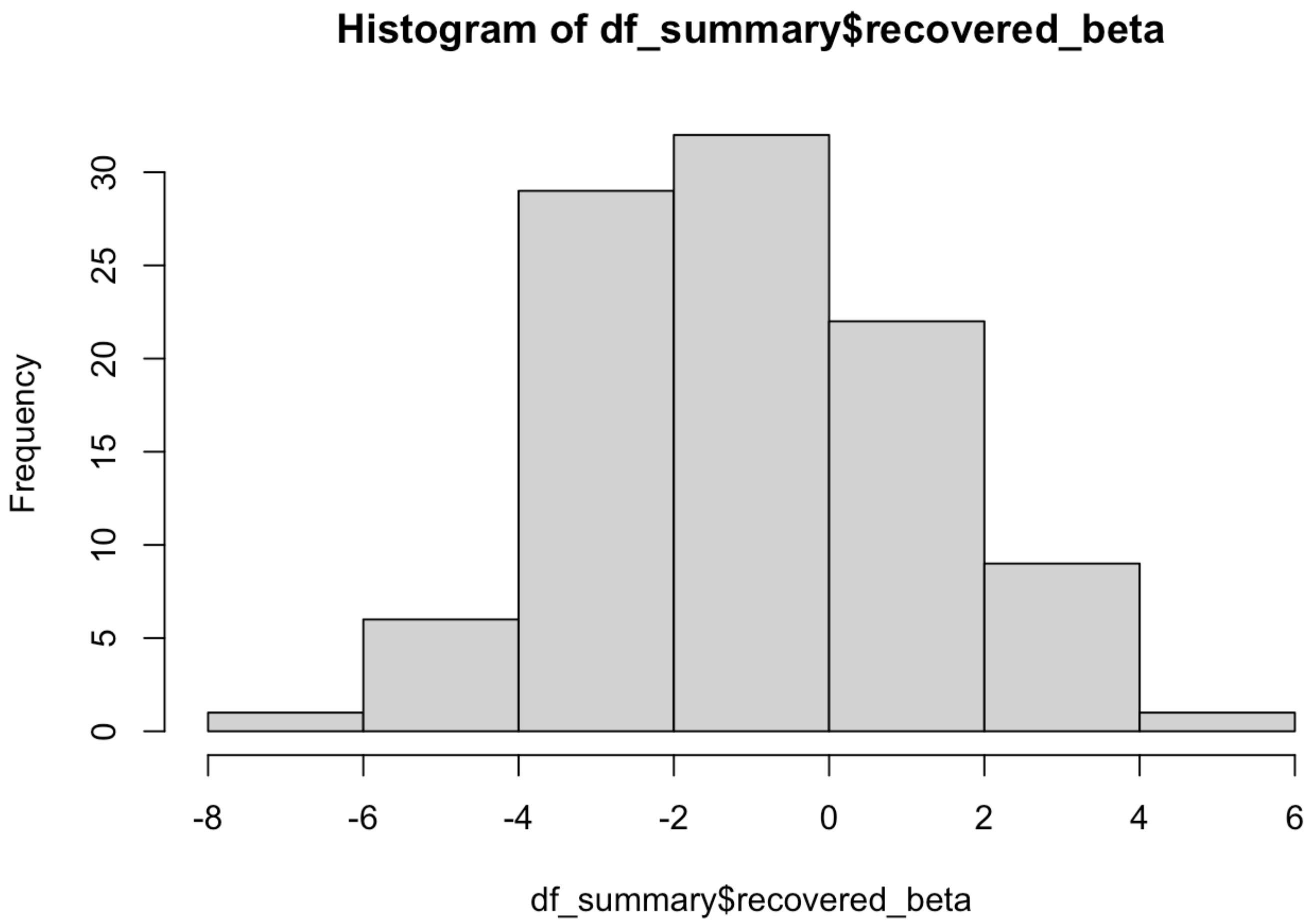

```
hist(df_summary$recovered_ndt)
```




```
hist(df_summary$recovered_alpha)
```



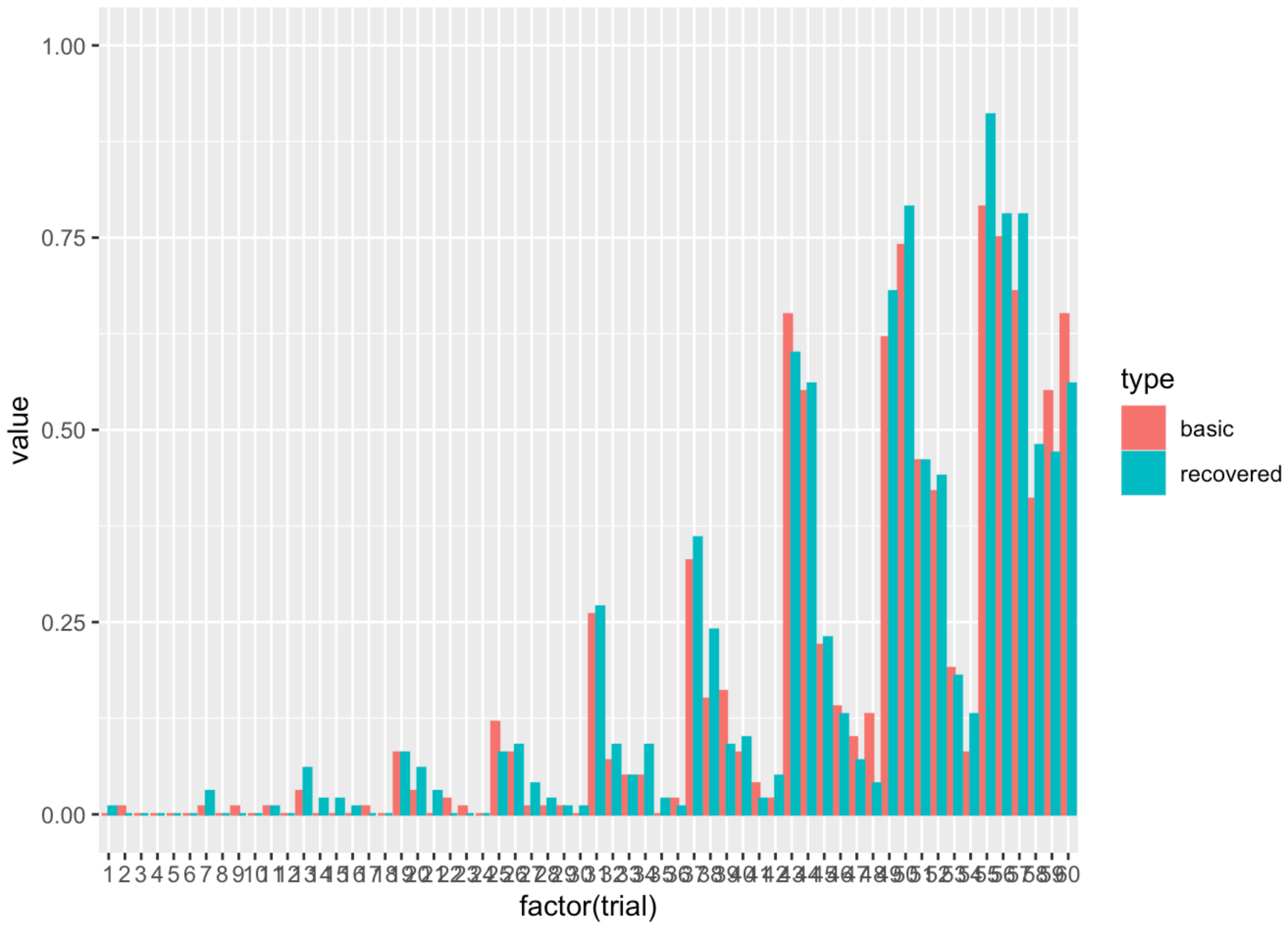

```
hist(df_summary$recovered_beta)
```




```
#check whether the risky choice proportion can be successfully recovered by the me
an-variance model
#firstly, use recovered parameter values to simulation choice data
for (i in 1:n_iter) {

  for(n in 1:nrow(stim3)){
      cres <- rwiener(1,mean(df_summary$recovered_th), mean(df_summary$recovered_n
dt), 0.5, mean(df_summary$recovered_theta) * ((stim3$evd[n]+df_summary$recovered_b
eta) + mean(df_summary$recovered_alpha) * stim3$sdd[n]))
      stim3$simrt[n] <- as.numeric(cres[1])
      stim3$simcho[n] <- ifelse(cres[2]=="upper",1,-1)
      stim3$cho2[n] <- ifelse(stim3$simcho[n] == 1, 0, ifelse(stim3$simcho[n] == -
1, 1, NA))

  }

  for(n in 1:nrow(stim4)){

    stim4$simchosum[n]  %+=% ifelse(stim3$simcho[n]==1,1,0)
    }
}
```

```
#create summary dataframe
label <- c(rep("basic", 60), rep("recovered", 60))
df <- data.frame(trial = rep(stim2$num),
                 value = c(stim2$simchosum/n_iter, stim4$simchosum/n_iter),
                 type = rep(label))
#display the first n trials
subset_data <- df[df$trial <= 60, ]
```

```
ggplot(subset_data, aes(x = factor(trial), y = value, fill = type, colour = type))
+
  geom_bar(stat = "identity", position = "dodge")+
  ylim(0,1)
```




```
library(rstan)
library(RWiener)
th =  4.52
ndt =   1.09
theta = .04
alpha = -0.59
```

```
stim = read.csv('Switching-Gambles.csv')

# gamble characteristics
  stim$eva = stim$payoffa1*stim$proba1+stim$payoffa2*stim$proba2

  stim$evb = stim$payoffb1*stim$probb1+stim$payoffb2*stim$probb2
  stim$evd = stim$evb-stim$eva
  stim$sda = sqrt((stim$payoffa1-stim$eva)^2*stim$proba1 + (stim$payoffa2-stim$ev
a)^2*stim$proba2)
  stim$sdb = sqrt((stim$payoffb1-stim$evb)^2*stim$probb1 + (stim$payoffb2-stim$ev
b)^2*stim$probb2)
  stim$sdd = stim$sdb - stim$sda




stim2 = read.csv('Switching-Gambles.csv')
stim3 = read.csv('Switching-Gambles.csv')

# gamble characteristics
  stim3$eva = stim$payoffa1*stim$proba1+stim$payoffa2*stim$proba2

  stim3$evb = stim$payoffb1*stim$probb1+stim$payoffb2*stim$probb2
  stim3$evd = stim$evb-stim$eva
  stim3$sda = sqrt((stim$payoffa1-stim$eva)^2*stim$proba1 + (stim$payoffa2-stim$ev
a)^2*stim$proba2)
  stim3$sdb = sqrt((stim$payoffb1-stim$evb)^2*stim$probb1 + (stim$payoffb2-stim$ev
b)^2*stim$probb2)
  stim3$sdd = stim$sdb - stim$sda


for(n in 1:nrow(stim2)){

    stim2$simchosum[n] = 0
}

stim4 = read.csv('Switching-Gambles.csv')
for(n in 1:nrow(stim4)){

    stim4$simchosum[n] = 0
}
```

```
# Set the number of iterations
n_iter <- 200


`%+=%` = function(e1,e2) eval.parent(substitute(e1 <- e1 + e2))
```

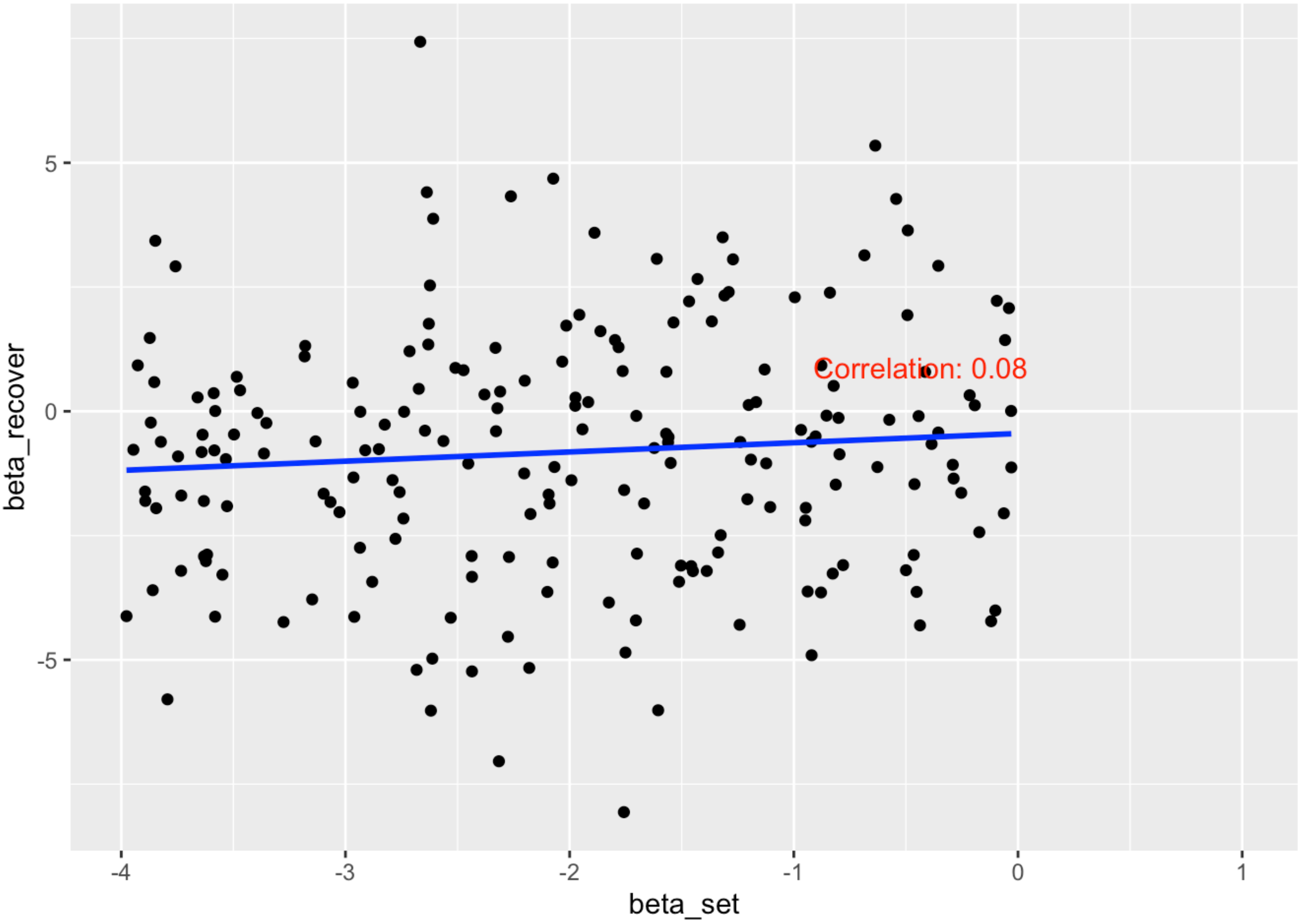
```
# Create empty vectors to store the outcome parameters for each iteration
th_recover <- numeric(n_iter)
theta_recover <- numeric(n_iter)
ndt_recover <- numeric(n_iter)
alpha_recover <- numeric(n_iter)
beta_recover <- numeric(n_iter)

th_bias <- numeric(n_iter)
theta_bias <- numeric(n_iter)
ndt_bias <- numeric(n_iter)
alpha_bias <- numeric(n_iter)
beta_bias <- numeric(n_iter)

th_dev <- numeric(n_iter)
theta_dev <- numeric(n_iter)
ndt_dev <- numeric(n_iter)
alpha_dev <- numeric(n_iter)
beta_dev <- numeric(n_iter)

#alpha_set <- numeric(n_iter)
beta_set <- numeric(n_iter)
# Run the model for n_iter iterations
for (i in 1:n_iter) {




   # Generate a single random non-zero value within the range
  #alpha <- 0
 # while (alpha == 0) {
  #  alpha <- sample(c(seq(min_value, -0.0001, length.out = 100), seq(0.0001, max_
value, length.out = 100)), 1)
 # }
 # alpha_set[i] = alpha

  beta <- runif(1, -4, 0)
  beta_set[i] = beta


  for(n in 1:nrow(stim)){
    cres <- rwiener(1,th, ndt, 0.5, theta * (stim$evd[n]+ beta + alpha * stim$sdd[
n]))
    stim$simrt[n] <- as.numeric(cres[1])
    stim$simcho[n] <- ifelse(cres[2]=="upper",1,-1)
  }



  for(n in 1:nrow(stim2)){
```

```
    stim2$simchosum[n]  %+=% ifelse(stim$simcho[n]==1,1,0)
    }




  parameters = c("alpha_sbj","threshold_sbj","ndt_sbj",'theta_sbj','mu_beta')
  dataList  = list(cho = stim$simcho,rt = stim$simrt, N=60,  L = 1, starting_point
=0.5, evd = stim$evd, sdd = stim$sdd)






  # Run the diffusion model for the current iteration
  dsamples <- stan(model_code = sim_ddm,
                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
                refresh = 0
                )

  samples <- extract(dsamples, pars = c('alpha_sbj', 'theta_sbj', 'threshold_sbj',
'ndt_sbj', 'mu_beta'))


  # Store the outcome parameters for the current iteration
  th_recover[i] <- mean(samples$threshold_sbj)
  theta_recover[i] <- mean(samples$theta_sbj)
  ndt_recover[i] <- mean(samples$ndt_sbj)
  alpha_recover[i] <- mean(samples$alpha_sbj)
  beta_recover[i] <- mean(samples$mu_beta)



  th_bias[i] <- (mean(samples$threshold_sbj)-th)/th
  theta_bias[i] <- (mean(samples$theta_sbj)-theta)/theta
  ndt_bias[i] <- (mean(samples$ndt_sbj)-ndt)/ndt
  alpha_bias[i] <- (mean(samples$alpha_sbj)-alpha)/alpha
  beta_bias[i] <- (mean(samples$mu_beta)-beta)/beta


  th_dev[i] <- abs(mean(samples$threshold_sbj)-th)/th
  theta_dev[i] <- abs(mean(samples$theta_sbj)-theta)/theta
  ndt_dev[i] <- abs(mean(samples$ndt_sbj)-ndt)/ndt
  alpha_dev[i] <- abs(mean(samples$alpha_sbj)-alpha)/alpha
```

```
  beta_dev[i] <- abs(mean(samples$mu_beta)-beta)/beta




}
```

```
# Load the required library
library(ggplot2)



# Create a data frame with the vectors
data <- data.frame(beta_set, beta_recover)

# Calculate the correlation coefficient
correlation <- cor(beta_set, beta_recover, method = "spearman")

# Create the scatter plot with correlation line using ggplot2
ggplot(data, aes(x = beta_set, y = beta_recover)) +
  geom_point() +
  geom_smooth(method = "lm", se = FALSE, color = "blue") +
  labs(x = "beta_set", y = "beta_recover") +
  annotate("text", x = 1, y = 1, label = paste0("Correlation: ", round(correlatio
n, 2)), hjust = 2, vjust = 0.8, color = "red")
```

```
## `geom_smooth()` using formula = 'y ~ x'
```


Correlation: 0.08
beta_recover
beta_set
5
0
-5
-4
-3
-2
-1
0
1