```r
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)
```

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


library(RWiener)


#original parameter values
th =  4.52
ndt =   1.09
beta =  .5
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=L> participant[N];            // level (participant)

    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;
}
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] + 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(1,5);
    threshold_v ~ normal(1,3);
    ndt_v ~ normal(0,1);


    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,-20,1),
                          threshold_v = runif(1,-0.5,5),
                          ndt_v = runif(1,-1.5, 0)
    )
  }
  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)

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

th_dev <- numeric(n_iter)
theta_dev <- numeric(n_iter)
ndt_dev <- numeric(n_iter)
alpha_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, beta, theta * (stim$evd[n] + alpha * stim$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')
  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=1000,
                chains=4,#If not specified, gives random inits
                init=initFunc(4),
                warmup = 500,  # Stands for burn-in; Default = iter/2
                refresh = 0
                )

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


  # 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)



  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


  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
```

```
}
```

```
## here are whatever error messages were returned
```

```
## [[1]]
## Stan model '4902d0378fbd23c5c32b941e46cb6162' does not contain samples.
```

```
#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
                 )
```

```
#create a table to show all means and true values
df_mean <- data.frame(parameter = c('th', "theta", "ndt", "alpha"),
                      true_value = c(th, theta,ndt, alpha),
                      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_bias = c(mean(df_summary$bias_th), mean(df_summary$bias
_theta),mean(df_summary$bias_ndt),mean(df_summary$bias_alpha)),
                      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))
                 )
df_mean
```

```
##   parameter true_value mean_recovered   mean_bias mean_deviation
## 1        th       4.52     4.63889726  0.02630470     0.07442669
## 2     theta       0.04     0.04007058  0.00176452     0.12651581
## 3       ndt       1.09     1.04889665 -0.03770950     0.11211901
## 4     alpha      -0.59    -0.63190795  0.07103042    -0.13535252
```

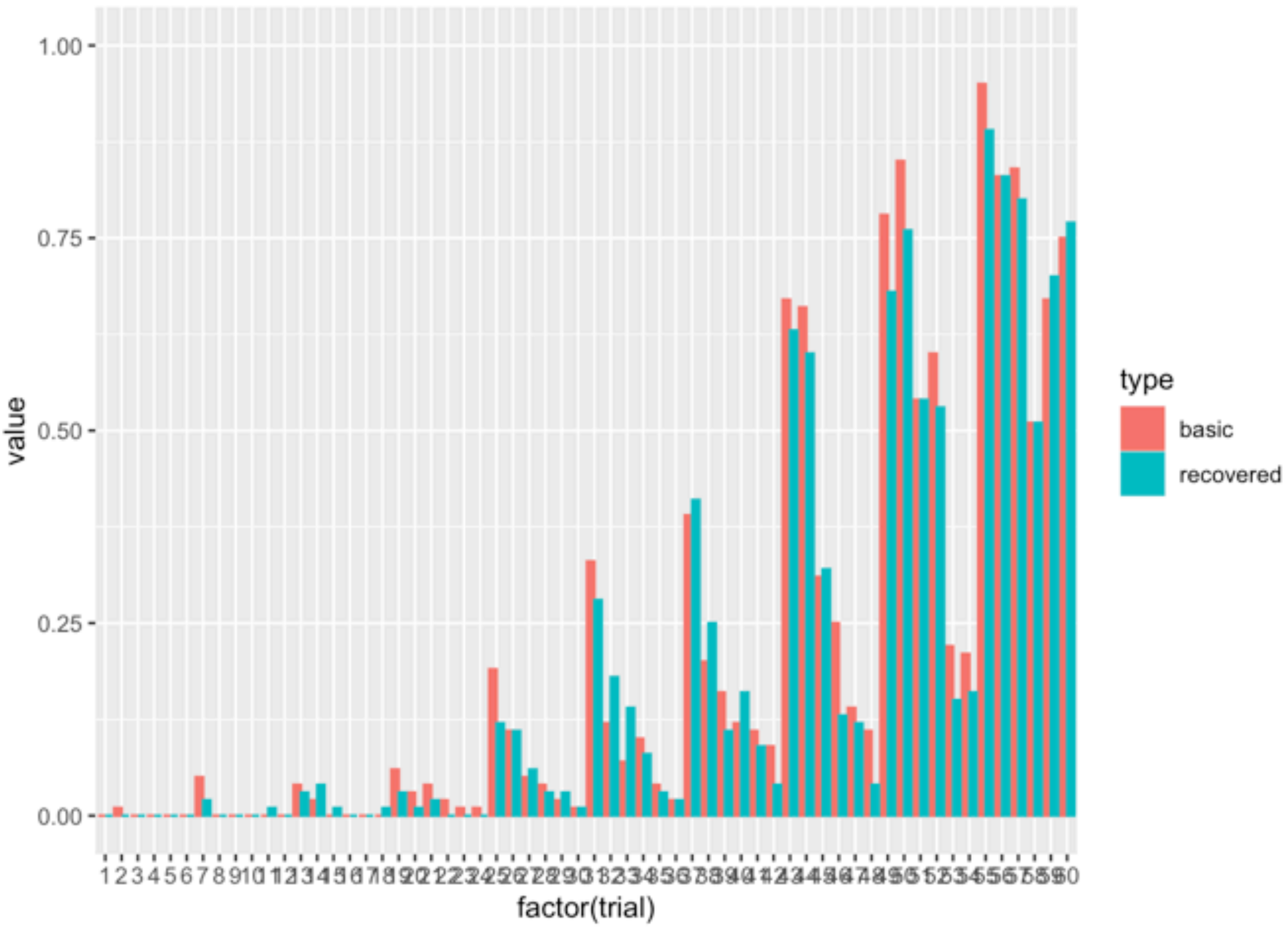
```
df_median <- data.frame(parameter = c('th', "theta", "ndt", "alpha"),
                      true_value = c(th, theta,ndt, alpha),
                      median_recovered = c(median(df_summary$recovered_th), media
n(df_summary$recovered_theta),median(df_summary$recovered_ndt),median(df_summary$r
ecovered_alpha))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

df_median
```

```
##   parameter true_value median_recovered
## 1        th       4.52       4.61109967
## 2     theta       0.04       0.03916862
## 3       ndt       1.09       1.03977598
## 4     alpha      -0.59      -0.61577317
```

```
#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), beta, mean(df_summary$recovered_theta) * (stim3$evd[n] + mean(df_summary$reco
vered_alpha) * stim3$sdd[n]))
      stim3$simrt[n] <- as.numeric(cres[1])
      stim3$simcho[n] <- ifelse(cres[2]=="upper",1,-1)

  }
  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, ]
```

```
library(ggplot2)
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
beta =  .5
theta = .04
```

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

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
}
```

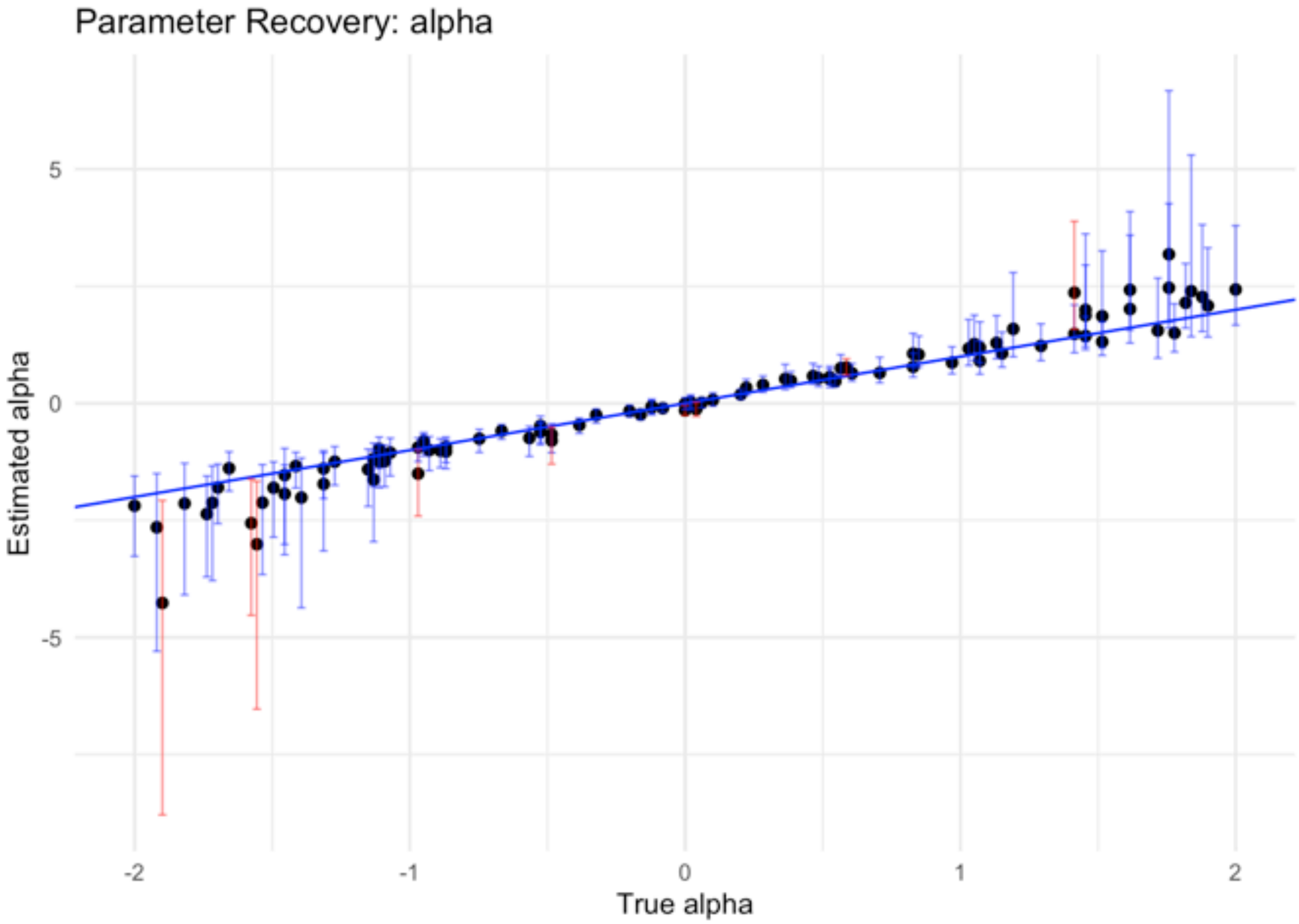
```
# Set the number of iterations
n_iter <- 100
```

```
`%+=%` = 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)

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

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

# Storage for results
results_df <- data.frame(
  True_alpha = numeric(n_iter),
  Estimated_alpha = numeric(n_iter),
  CI_alpha_Lower = numeric(n_iter),
  CI_alpha_Upper = numeric(n_iter)
)

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

  # Set the range (minimum and maximum values)
  min_value <- -2
  max_value <- 2


   # 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_v
alue, length.out = 100)), 1)
  }
  alpha_set[i] = alpha


  for(n in 1:nrow(stim)){
    cres <- rwiener(1,th, ndt, beta, theta * (stim$evd[n] + alpha * stim$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')
  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=1000,
                 chains=4,#If not specified, gives random inits
                 init=initFunc(4),
                 warmup = 500,  # Stands for burn-in; Default = iter/2
                 refresh = 0
                 )

  samples <- extract(dsamples, pars = c('alpha_sbj', 'theta_sbj', 'threshold_sbj',
'ndt_sbj'))
  extracted_params <- extract(dsamples)
  Estimated_alpha = mean(extracted_params$alpha_sbj)
  CI_alpha = quantile(extracted_params$alpha_sbj, probs = c(0.025, 0.975))

  # 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)



  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
```

```
  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

    # Store the results in the data frame
  results_df[i, ] <- c(
    alpha,
    Estimated_alpha,
    CI_alpha[1],
    CI_alpha[2]
  )


}
```

```
library(ggplot2)
# Create scatterplots for True vs. Estimated Intercepts with color-coded error bar
s
plot_alpha <- ggplot(results_df, aes(x = True_alpha, y = Estimated_alpha)) +
  geom_point(shape = 16, size = 2, color = "black", fill = "white") +
  geom_abline(intercept = 0, slope = 1, color = "blue") +
  geom_errorbar(
    aes(ymin = results_df$CI_alpha_Lower, ymax = results_df$CI_alpha_Upper),
    width = 0.03,
     color = ifelse(results_df$CI_alpha_Lower > results_df$True_alpha | results_df
$CI_alpha_Upper < results_df$True_alpha, "red", "blue"),
    linetype = "solid",
    linewidth = 0.4,
    alpha = 0.5
  ) +
  labs(
    title = "Parameter Recovery: alpha",
    x = "True alpha",
    y = "Estimated alpha"
  ) +
  theme_minimal()  # Change to a minimal theme

# Print the plot
print(plot_alpha)
```


Parameter Recovery: alpha
5
0
-5
Estimated alpha
-2
-1
0
1
2
True alpha