EZ Bayesian Hierarchical Drift Diffusion Model

Based on Joachim's python code

Adriana F. Chavez De la Pena

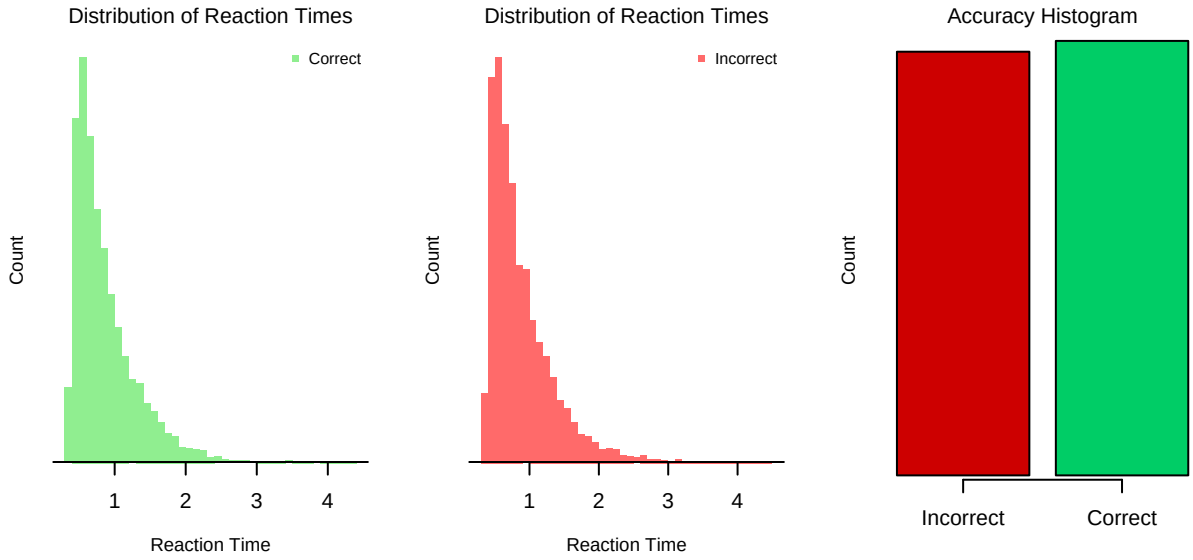
Last time knitted: 23 October, 2023

Basic functions to generate DDM data

```
# Part 1: Simulate single trial outcome
simulate_ddm <- function(a, v, dt, max_steps){
     x <- 0
     random_dev <- rnorm(max_steps)
     # Scale step changes by dt
     noise <- random_dev * sqrt(dt)
     drift <- v * dt

     for(i in 2:max_steps){
        this_step = drift + noise[i]
        x = x + this_step
            if(abs(x)>=(a/2)){  break  }
     }
     output <- list("RT" = (i+1)*dt, "C"  = x)
     return(output)
}

# Part 2: Simulate over 'n' trials
wdmrnd <- function(a,v,t,n){
    dt = 0.001
    max_steps = 10 / dt
    rt = rep(NA,n)
    accuracy = rep(NA,n)

    for(i in 1:n){
        X <- simulate_ddm(a, v, dt, max_steps)
        rt[i] <- X$RT
        if(X$C>0){  accuracy[i] <- 1
          }else{    accuracy[i] <- 0  }
    }
    output <- data.frame("RT" = rt + t, "accuracy" = accuracy)
    return(output)
}
```

Example: Generate some data

```
a = 1.50
v = 0.00
t = 0.30
n = 10000

data <- wdmrnd(a, v, t, n)
```



Simulation Study environment and variables

Auxiliary functions

The code for the auxiliary functions listed below is hidden from this pdf file (but can be checked on the .Rmd).

1. `design_summary`: A function to print the settings used in the simulation.
2. `default_priors`: A function to load and print default prior values.
3. `sample_parameters`: A function to sample true parameter values from the priors specified.
4. `write_JAGSmodel`: A function to write the JAGS model using the prior values.
5. `data_toJAGS`: A function to create a list with all the data objects to be passed to JAGS.
6. `default_inits`: A function to create an object containing initial values for the drift.
7. `extractSamples`: A function to extract all samples associated with a `parameter.name`
8. `plot.Chain`: A function to plot the merging chains for hierarchical parameters
9. `getError`: A function to compute the difference between the true value and estimate retrieved for every parameter.
10. `nameOutput` : A function to generate an appropriate output name based on the number of trials and participants

Plotting functions

```
recoveryPlot <- function(x.true, y.estimated, parameterName){
  allValues <- c(x.true,y.estimated)
  param.is.bound <- length(which(grepl("ound",parameterName)))!=0
  param.is.nondt <- length(which(grepl("no",parameterName)))!=0
  if(param.is.bound){   color <- "blueviolet"
  }else{if(param.is.nondt){   color <- "dodgerblue4"
  }else{  color <- "goldenrod4" }}
  par(bty="o")
  plot(x.true,y.estimated, ann=F, col=color, pch=16,
       xlim=c(min(allValues),max(allValues)),
       ylim=c(min(allValues),max(allValues)))
  abline(0,1, lty=2)
  mtext("Simulated values", 1, line=2.1, cex=0.7)
  mtext(parameterName, 3, line=0.5, cex=0.8, f=2)
}

showRecovery <- function(simStudy, nParticipants, nTrials){
  par(mfrow = c(2, 3))#, mar =c(5.1,2,4.1,2))
  x <- simStudy$sim_means
  recoveryPlot(x[,"true","drift_mean"],x[,"est","drift_mean"],"Group mean drift")
  recoveryPlot(x[,"true","bound_mean"],x[,"est","bound_mean"],"Group mean bound")
  recoveryPlot(x[,"true","nondt_mean"],x[,"est","nondt_mean"],"Group mean nondt")
  y <- simStudy$sim_indiv
  recoveryPlot(y[,"true","drift"],y[,"est","drift"],"Individual drifts")
  recoveryPlot(y[,"true","bound"],y[,"est","bound"],"Individual bounds")
  recoveryPlot(y[,"true","nondt"],y[,"est","nondt"],"Individual nondt")
  title <- paste(nParticipants, " participants, ", nTrials, " trials each")
mtext(title, side = 3, line =-1.7, outer = TRUE, f=2, col="red3")
}
```

Core functions

```
# Sample data using simulation settings and true parameter values sampled
sample_data <- function(settings, parameter_set){
  nObs <- settings$nPart*settings$nTrials
  data <- matrix(NA,ncol=3,nrow=nObs)
  data[,1] <- rep(1:settings$nPart, each=settings$nTrials)
  for(i in 1:settings$nP){      # Get data for every Participant
      this.sub <- which(data[,1]==i)
      accuracy = 0
      while(sum(accuracy)==0){
            temp <- wdmrnd(a = parameter_set$bound[i], v = parameter_set$drift[i],
                           t = parameter_set$nondt[i], n = settings$nTrials)
            accuracy = temp$accuracy
      }
      data[this.sub,3] <- accuracy
      data[this.sub,2] <- temp$RT
  }
  data <- as.data.frame(data)
  colnames(data) <- c("sub", "rt", "accuracy")
  return(data)
}

# Get individual statistics from raw data: mean accuracy and mean and variance of correct-RT
get_Statistics <- function(data){
  if(is.null(data$accuracy)|is.null(data$rt)){
        error.msg = "Data not available."
        return(print(error.msg))
    }
  subID <- unique(data$sub)
  sum_correct <- tapply(data$accuracy, data$sub, sum)
  # Remove participants with no correct answer
  always_0 <- which(sum_correct==0)
  if(length(always_0)!=0){
        bad_participants <- (data$sub %in% always_0)
        data <- data[-bad_participants,]
        sum_correct <- tapply(data$accuracy, data$sub, sum)
  }
  # Get proportion of correct responses
  mean_accuracy <- tapply(data$accuracy, data$sub, mean)
  # Get mean and variance of correct RT
  keep.correct <- which(data$accuracy==1)
  correct_only <- data[keep.correct,]
  mean_rt_correct <- tapply(correct_only$rt, correct_only$sub, mean)
  var_rt_correct  <- tapply(correct_only$rt, correct_only$sub, var)
  # Create a data.frame with just summary statistics
  data_statistics <- cbind(subID, sum_correct, mean_accuracy, mean_rt_correct, var_rt_correct)
  data_statistics <- as.data.frame(data_statistics)
  colnames(data_statistics) = c("sub", "sum_correct","meanAccuracy", "meanRT_correct", "varRT_correct")
  return(data_statistics)
}
```

Main functions

```
# A function to load priors and true values to use in simulation
Hddm_Parameter_Set <-function(nParticipants, nTrials, Show=TRUE){
```

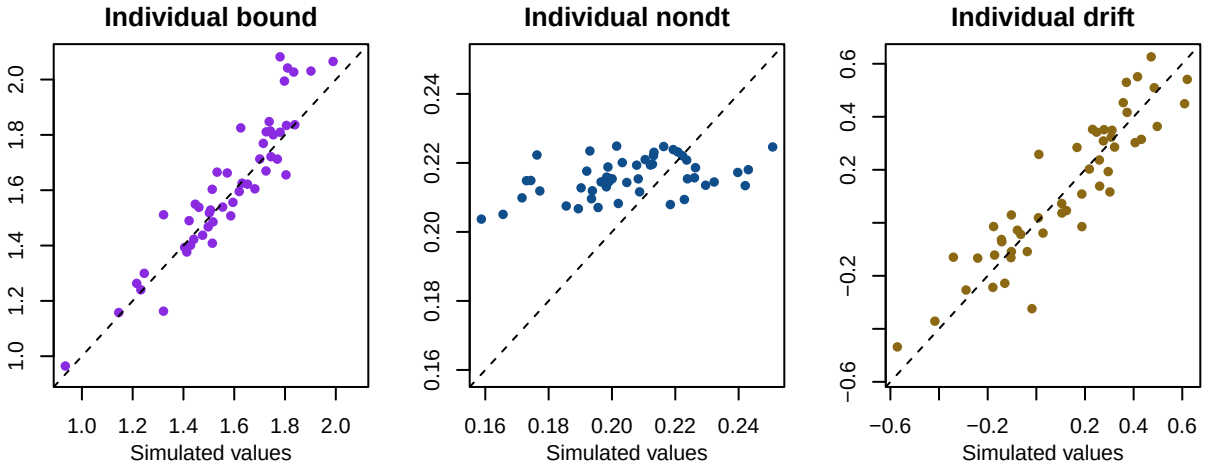
```
    if(Show){  design_summary(nParticipants,nTrials)  }
    prior <- default_priors(Show)
    settings <- list("nPart"= nParticipants, "nTrials"= nTrials, "prior"= prior)
    parameter_set <- sample_parameters(settings, Show)
    return(list("settings" = settings, "parameter_set" = parameter_set))
}

# A function to generate raw data, summary statistics and data to be passed on JAGS
Hddm_Data <- function(settings, parameter_set){
    rawData = sample_data(settings,parameter_set)
    sumData = get_Statistics(rawData)
    jagsData = data_toJAGS()
    return(list("rawData" = rawData, "sumData" = sumData, "jagsData" = jagsData))
}

# A function to run JAGS model
Hddm_runJAGS <- function(getData, settings, n.chains, modelFile="./EZHBDDM.bug", plot.Chains = FALSE){
    # Write model
    write_JAGSmodel(settings$prior)
    # Load settings
    parameters <- c("bound_mean", "drift_mean", "nondt_mean", "bound", "nondt",
                    "drift_sdev", "nondt_sdev", "bound_sdev", "drift")
    myinits    <- default_inits(n.chains, settings$nPart)
    data <- getData$jagsData
    # Prepare data
    sub     <- getData$sumData$sub
    correct <- getData$sumData$sum_correct
    varRT   <- getData$sumData$varRT_correct
    meanRT  <- getData$sumData$meanRT_correct
    nTrialsPerPerson <- as.numeric(unique(tapply(getData$rawData$accuracy,getData$rawData$sub,length)))
    nParticipants    <- length(getData$sumData$sub)
    # Run model and get samples
    suppressMessages(library(R2jags))
    suppressMessages(samples <- jags(data=data,
                                parameters.to.save=parameters,
                                model=modelFile,
                                n.chains=n.chains,
                                n.iter=1000,
                                n.burnin=200,
                                n.thin=1,
                                DIC=T,
                                inits=myinits))
    if(plot.Chains){  plot.Chain(samples)   }
return(list("drift" = apply(extractSamples("drift", samples),3,mean),
            "drift_mean" = mean(extractSamples("drift_mean", samples)),
            "drift_sdev" = mean(extractSamples("drift_sdev", samples)),
            "bound" = apply(extractSamples("bound", samples),3,mean),
            "bound_mean" = mean(extractSamples("bound_mean", samples)),
            "bound_sdev" = mean(extractSamples("bound_sdev", samples)),
            "nondt" = apply(extractSamples("nondt", samples),3,mean),
            "nondt_mean" = mean(extractSamples("nondt_mean", samples)),
            "nondt_sdev" = mean(extractSamples("nondt_sdev", samples))))
}

# Main function: A function to run a complete simulation for nParticipants and nTrials per participant
Hddm_runSim <- function(nParticipants, nTrials, n.chains = 4, Show=TRUE){
```

```
      design.parameters <- Hddm_Parameter_Set(nParticipants,nTrials, Show=Show)
      settings <- design.parameters$settings
      parameter_set <- design.parameters$parameter_set
      getData <- Hddm_Data(settings,parameter_set)
      estimates <- Hddm_runJAGS(getData=getData, n.chains = n.chains, settings = settings)
      error <- getError(estimates, parameter_set)
return(list("trueValues" = parameter_set, "estValues" = estimates, "error" = error))
}


Hddm_simStudy <- function(nSim, nParticipants, nTrials, n.chains = 4,
                          Show=FALSE, forceSim = FALSE){
    outputFile <- nameOutput(nTrials, nParticipants)
    runSim <- TRUE
    if((!forceSim)&(file.exists(outputFile))){runSim <- FALSE}
    if(runSim){
      sim_means <- array(NA, dim=c(nSim,3,3))
      sim_indiv <- array(NA, dim=c(nSim*nParticipants,3,3))
      indiv_init = 1
      for(k in 1:nSim){
          set.seed(k)
          cat("Iteration", k, "of", nSim,"\n")
          tryCatch({
            sim <- Hddm_runSim(nParticipants = 50, nTrials = 150, Show = FALSE)
            sim_means[k,,1] <- c(sim$trueValues$drift_mean, sim$estValues$drift_mean, sim$error$drift_mean)
            sim_means[k,,2] <- c(sim$trueValues$bound_mean, sim$estValues$bound_mean, sim$error$bound_mean)
            sim_means[k,,3] <- c(sim$trueValues$nondt_mean, sim$estValues$nondt_mean, sim$error$nondt_mean)
            Last = nParticipants*k
            sim_indiv[indiv_init:Last,,1] = c(sim$trueValues$drift, sim$estValues$drift, sim$error$drift)
            sim_indiv[indiv_init:Last,,2] = c(sim$trueValues$bound, sim$estValues$bound, sim$error$bound)
            sim_indiv[indiv_init:Last,,3] = c(sim$trueValues$nondt, sim$estValues$nondt, sim$error$nondt)
            indiv_init = Last + 1
            }, error=function(e){cat("ERROR :",conditionMessage(e), "\n")})
      }
      sim_means <- sim_means[which(!is.na(sim_means[,1,1])),,]
      sim_indiv <- sim_indiv[which(!is.na(sim_indiv[,1,1])),,]
      dimnames(sim_means) <- list(NULL, c("true","est","error"), c("drift_mean","bound_mean","nondt_mean"))
      dimnames(sim_indiv) <- list(NULL, c("true","est","error"), c("drift","bound","nondt"))
      simOutput = list("sim_means" = sim_means, "sim_indiv" = sim_indiv)
      save(simOutput,file=outputFile)
    }else{    load(outputFile)     }
    return(simOutput)
}
```

Run simple example

```
sim <- Hddm_runSim(nParticipants = 50, nTrials = 150)
```

```
## ========== EZBHDDM Design Parameters: =======
## Number of Participants:     50
## Trials Per Person:          150
## ========== EZBHDDM Priors: ==================
## Bound Mean Mean:    1.5
## Bound Mean Std Dev: 0.2
## Drift Mean Mean:    0
## Drift Mean Std Dev: 0.5
## Non-decision Time Mean Mean: 0.3
## Non-decision Time Mean Std:  0.06
## Bound Std Dev Shape: 0.1
## Bound Std Dev Scale: 0.2
## Drift Std Dev Shape: 0.2
## Drift Std Dev Scale: 0.4
## Non-decision Time Shape: 0.01
## Non-decision Time Scale: 0.05
## ===== EZBHDDM True Parameters: ==============
## Bound Mean:    1.570147
## Bound SD:      0.1977124
## Drift Mean:    0.1629902
## Drift SD:      0.2687111
## Non-decision Time Mean: 0.2058321
## Non-decision Time SD:   0.02227891
## ============================================
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 150
##    Unobserved stochastic nodes: 156
##    Total graph size: 1579
##
## Initializing model
```



Full smulation studies (200 repetitions)

150 trials

```
nSim <- 200

nT <- 150
nP1 <- 10
nP2 <- 80
nP3 <- 200
nP4 <- 500
```

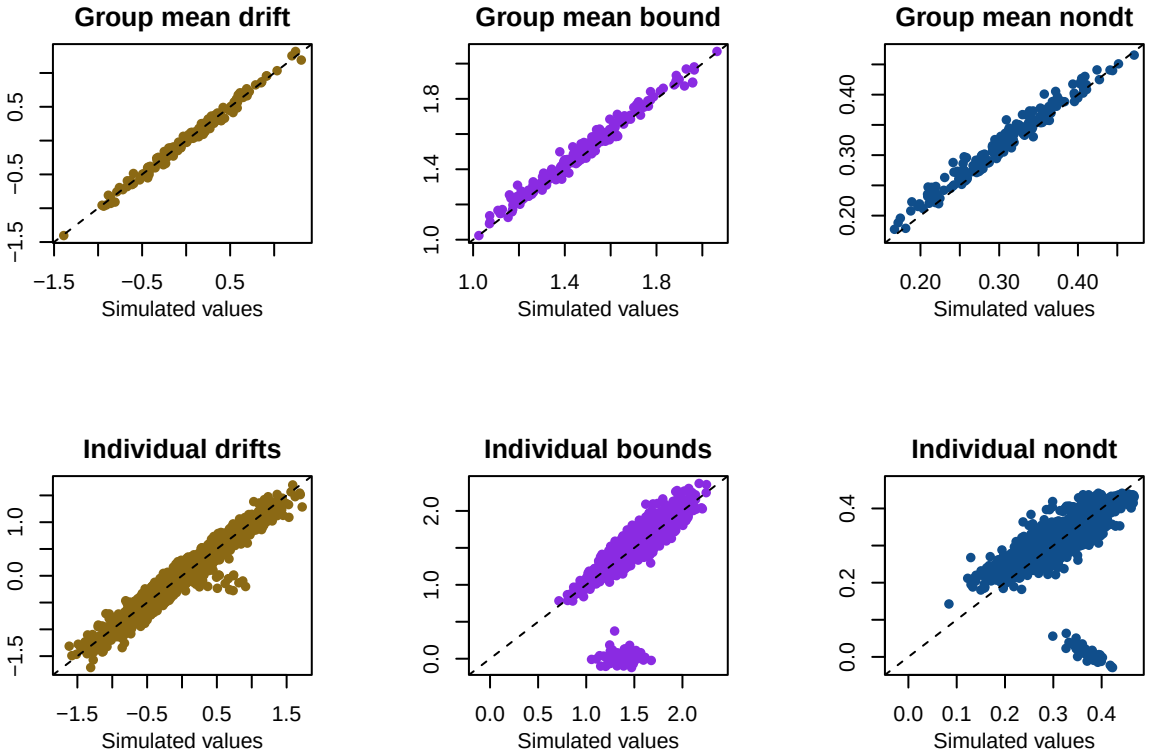
```
simStudy1 <- Hddm_simStudy(nSim = nSim, nParticipants = nP1, nTrials = nT)
simStudy2 <- Hddm_simStudy(nSim = nSim, nParticipants = nP2, nTrials = nT)
simStudy3 <- Hddm_simStudy(nSim = nSim, nParticipants = nP3, nTrials = nT)
simStudy4 <- Hddm_simStudy(nSim = nSim, nParticipants = nP4, nTrials = nT)
```

150 participants

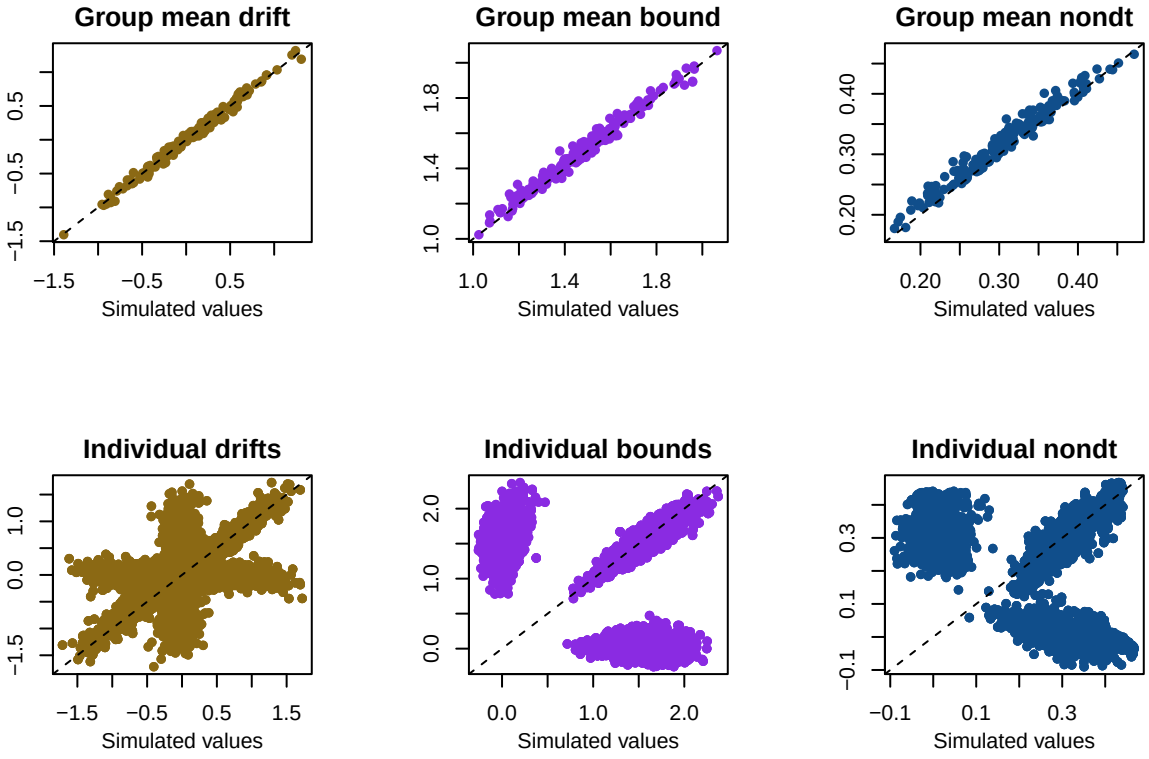
```
nSim <- 200

nP <- 150
nT1 <- 10
nT2 <- 80
nT3 <- 200
nT4 <- 500
```

```
simStudy5 <- Hddm_simStudy(nSim = nSim, nParticipants = nP, nTrials = nT1)
simStudy6 <- Hddm_simStudy(nSim = nSim, nParticipants = nP, nTrials = nT2)
simStudy7 <- Hddm_simStudy(nSim = nSim, nParticipants = nP, nTrials = nT3)
simStudy8 <- Hddm_simStudy(nSim = nSim, nParticipants = nP, nTrials = nT4)
```

10 participants, 150 trials each

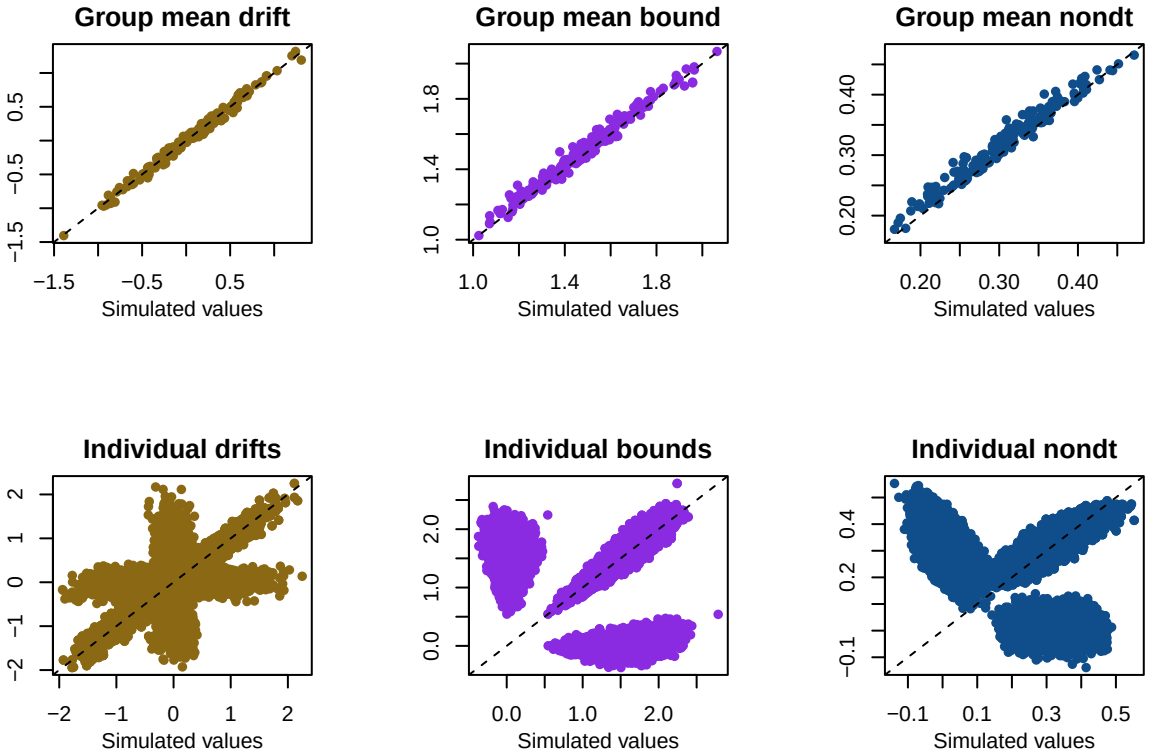


80 participants, 150 trials each

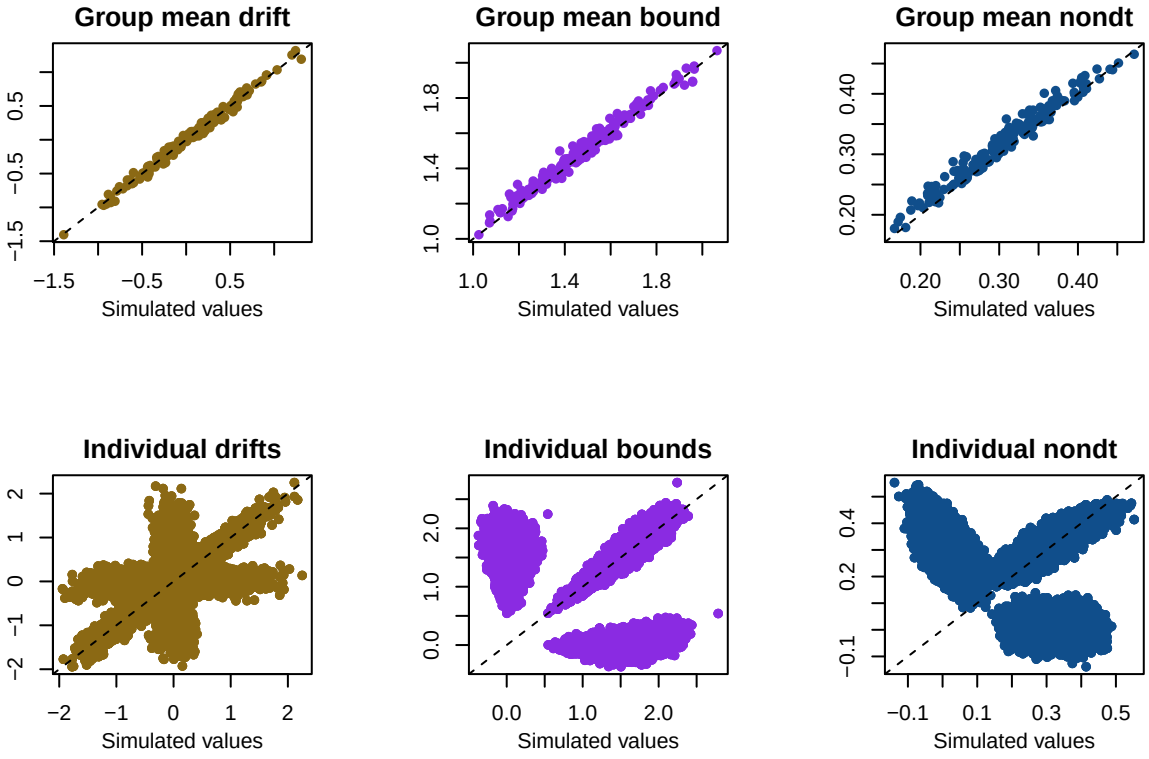


9

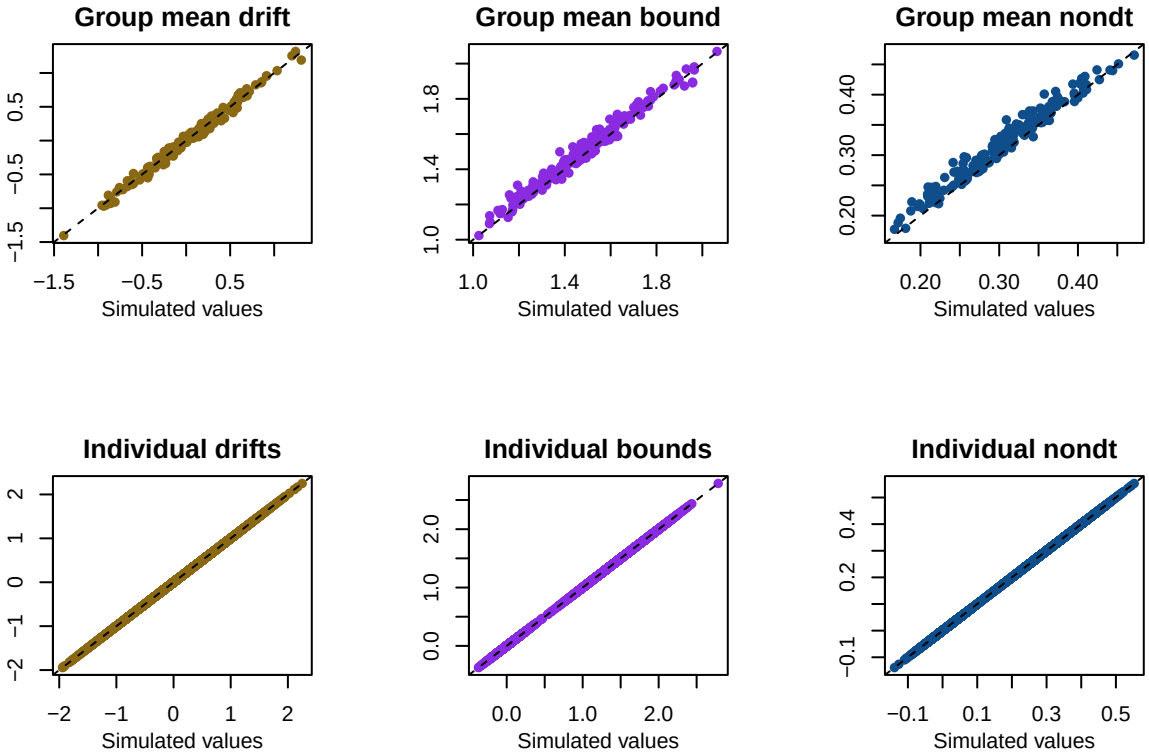
200 participants, 150 trials each



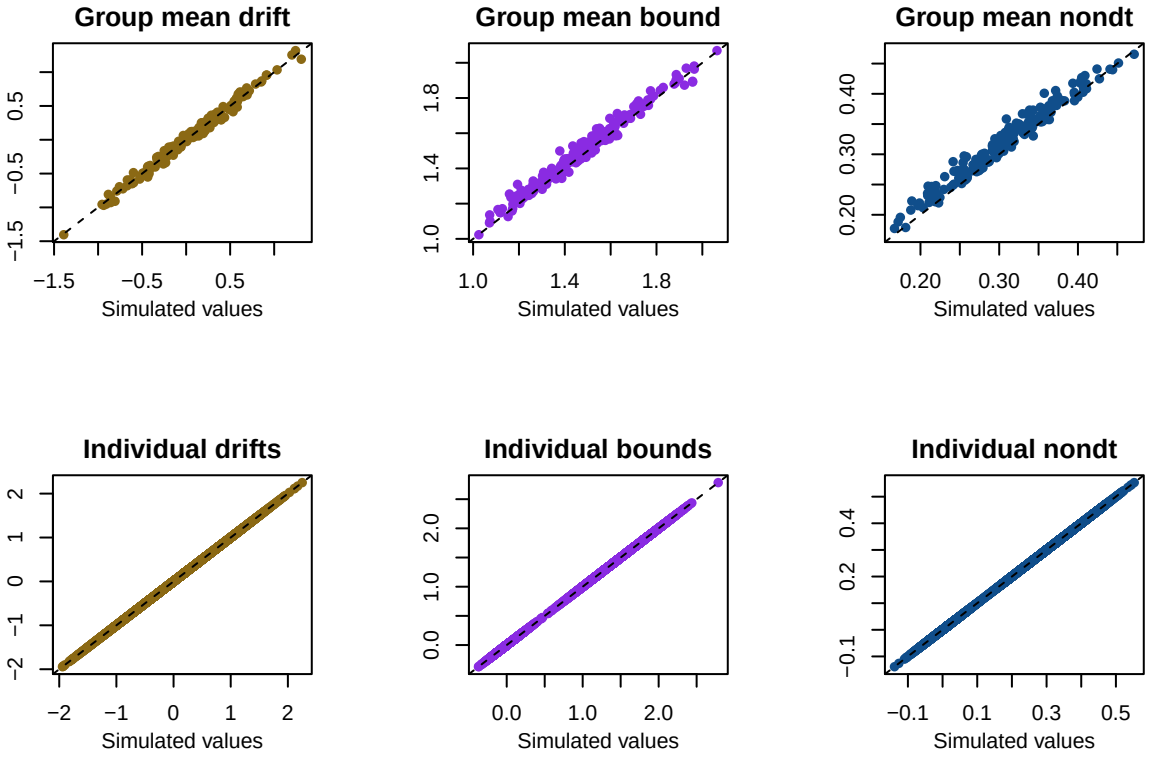
500 participants, 150 trials each



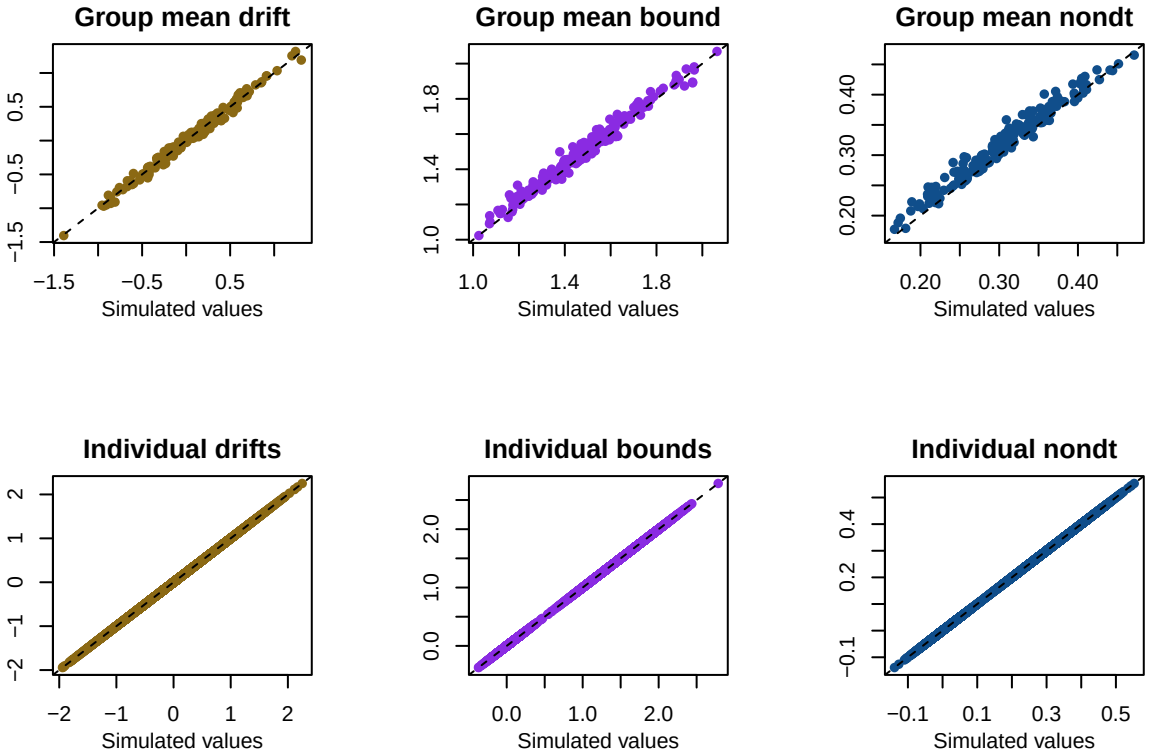
150 participants, 10 trials each



150 participants, 80 trials each



150 participants, 200 trials each



150 participants, 500 trials each