

EZ Bayesian Hierarchical Drift Diffusion Model

Based on Joachim's python code

Adriana F. Chavez

Last time knitted: 25 September, 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

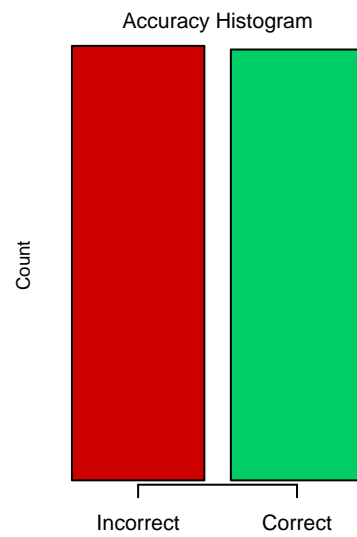
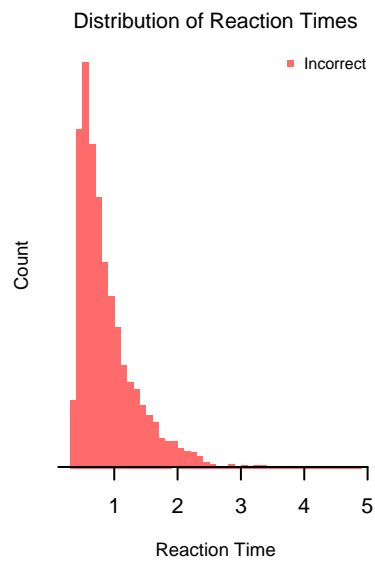
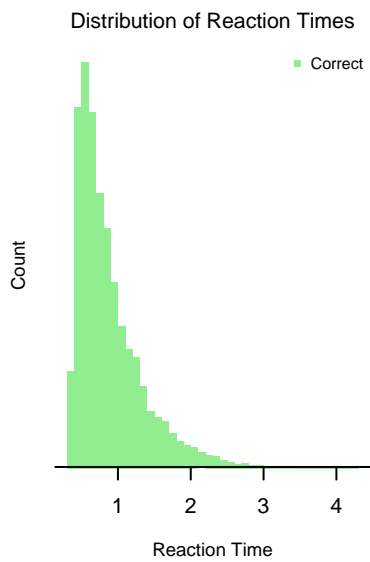
  for(i in 2:max_steps){
    this_step = drift + noise[i]
    x = x + this_step
    if(abs(x)>=(a/2)){ break }
  }
  output <- list("RT" = (i+1)*dt, "C" = x)
  return(output)
}

# 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)  
rt <- data$RT  
accuracy <- data$accuracy
```



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 file).

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. `recoveryPlot`: A function to create a quick recovery plot

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$drift[i],
                    t = parameter_set$nonndt[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))
}

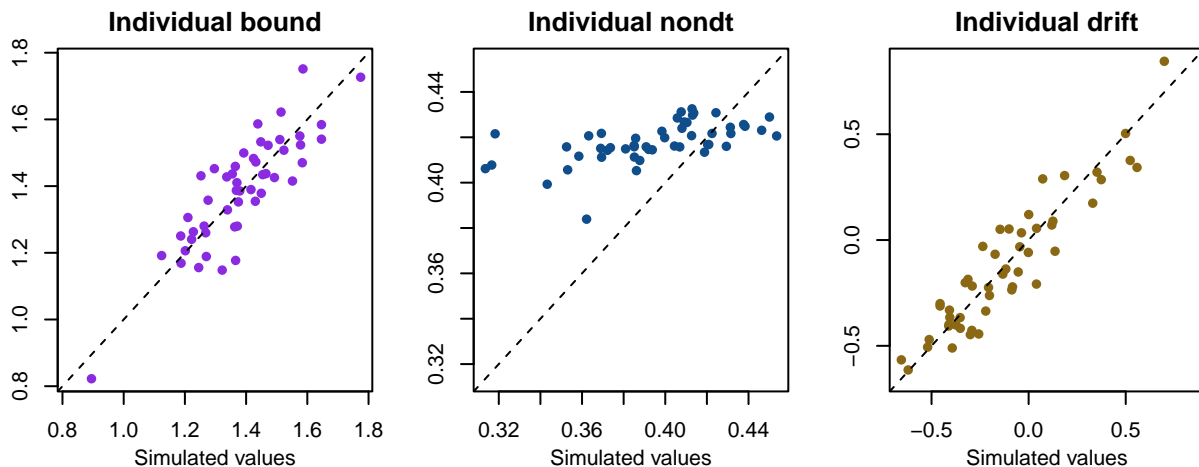
Hddm_simStudy <- function(nSim, nParticipants, nTrials, n.chains = 4,
  outputFile = "./output_simStudy.RData", Show=FALSE, forceSim = FALSE){
  runSim <- TRUE
  if((!forceSim)&(file.exists(outputFile))){runSim <- FALSE}
  if(runSim){
    simOutput <- array(NA, dim=c(nSim,3,3))
    for(k in 1:nSim){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

```

Run simple example

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

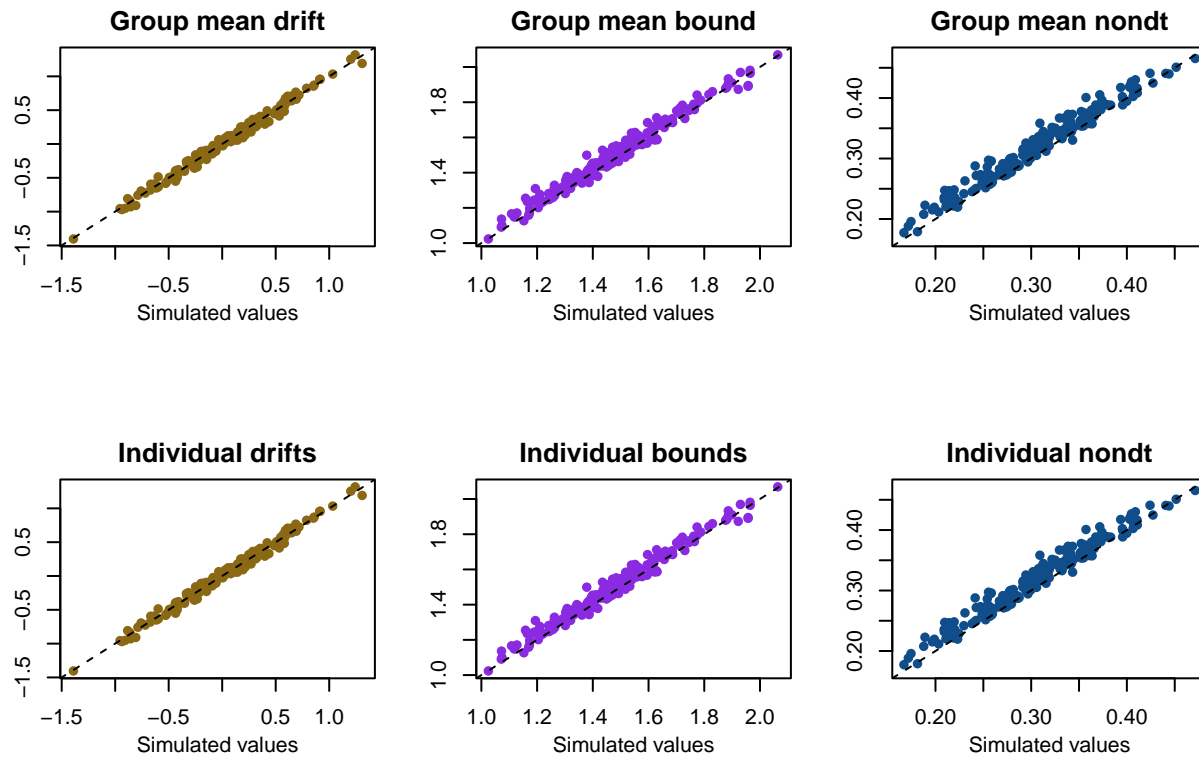
```
## ===== EZBHDDM Design Parameters: =====
## Number of Participants:      50
## Trials Per Person:          150
## ===== EZBHDDM Priors: =====
## Bound Mean Mean:            1.5
## Bound Mean Std Dev: 0.2
## Drift Mean Mean:            0
## Drift Mean Std Dev: 0.5
## Non-decision Time Mean Mean: 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:      1.387905
## Bound SD:         0.1528105
## Drift Mean:       -0.1150887
## Drift SD:         0.3784838
## Non-decision Time Mean: 0.3935225
## Non-decision Time SD:  0.0320574
## =====
## Compiling model graph
##   Resolving undeclared variables
##   Allocating nodes
## Graph information:
##   Observed stochastic nodes: 150
##   Unobserved stochastic nodes: 156
##   Total graph size: 1579
##
## Initializing model
```



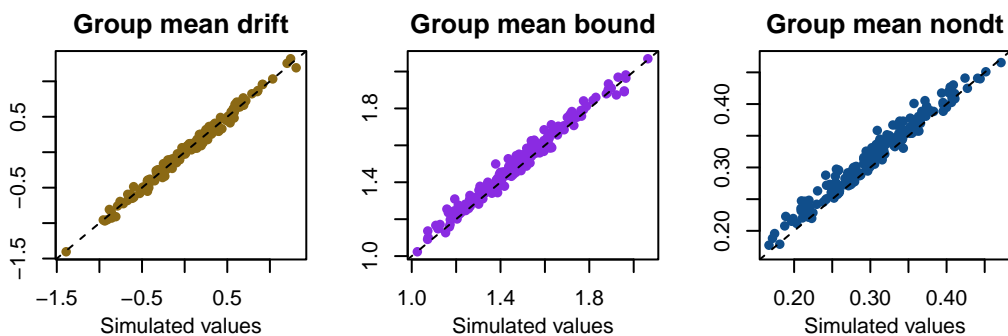
Simulation study (150 trials - 200 repetitions)

```
nT <- 150
nP <- 50
simStudy <- Hddm_simStudy(nSim = 200, nParticipants = nP, nTrials = nT, outputFile = "./sim-P50T150.RData")
showRecovery(simStudy, nP, nT)
```

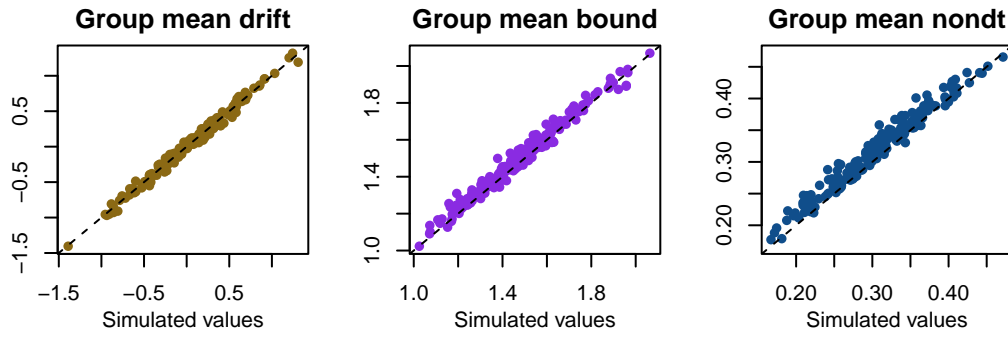
50 participants, 150 trials each



50 participants, 150 trials each



50 participants, 50 trials each



500 participants, 10 trials each

