EZ Bayesian Hierarchical Drift Diffusion Model

Based on Joachim's python code

Adriana F. Chavez

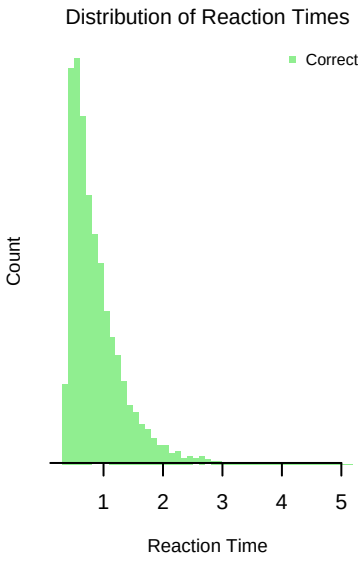
Last time knitted: 20 September, 2023

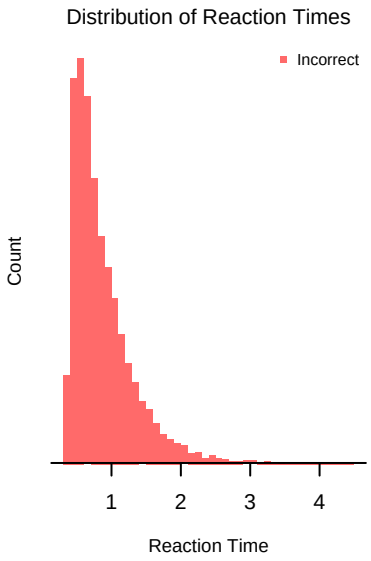
Basic functions to generate DDM data

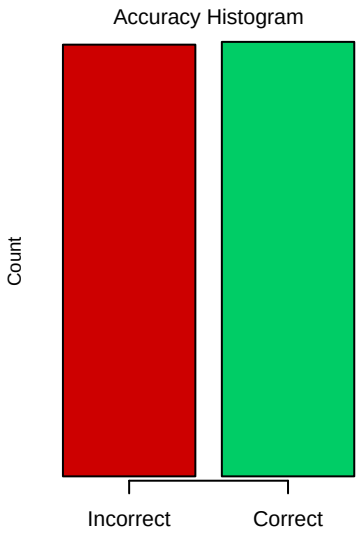
```
# Part 1: Simulate single trial outcome
simulate_ddm <- function(a, v, dt, max_steps){
     x <- 0
     random_dev <- rnorm(max_steps)
     # Scale step changes by dt
     noise <- random_dev * sqrt(dt)
     drift <- v * dt

     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 is hidden from this .pdf file (but can be checked on the .Rmd file). The auxiliary functions are:

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. `write_JAGSmodel`: A function to write the JAGS model using the prior values
4. `data_toJAGS`: A function to create a list with all the data objects in the JAGS model
5. `default_inits`: A function to create an object (list) containing initial values for the drift
6. `extractSamples`: A function to extract individual samples for any parameter
7. `plot.Chain`: A function to plot the merging chains for the hierarchical parameters
8. `getError`: A function to compute the difference between the true value and estimate retrieved for every parameter.

Core functions

```
# Sample 'true' parameter values from the priors specified
sample_parameters <- function(settings){
    prior <- settings$prior
    bound_mean <- rnorm(1,prior$bound_mean_mean,prior$bound_mean_sdev)
    drift_mean <- rnorm(1,prior$drift_mean_mean,prior$drift_mean_sdev)
    nondt_mean <- rnorm(1,prior$nondt_mean_mean,prior$nondt_mean_sdev)
    bound_sdev <- runif(1,prior$bound_sdev_lower,prior$bound_sdev_upper)
    drift_sdev <- runif(1,prior$drift_sdev_lower,prior$drift_sdev_upper)
    nondt_sdev <- runif(1,prior$nondt_sdev_lower,prior$nondt_sdev_upper)
    bound <- rnorm(settings$nPart,bound_mean, bound_sdev)
    drift <- rnorm(settings$nPart,drift_mean, drift_sdev)
    nondt <- rnorm(settings$nPart,nondt_mean, nondt_sdev)
    parameter_set <- list("bound_mean" = bound_mean, "drift_mean" = drift_mean, "nondt_mean" = nondt_mean,
                          "bound_sdev" = bound_sdev, "drift_sdev" = drift_sdev, "nondt_sdev" = nondt_sdev,
                          "bound" = bound,   "drift" = drift,   "nondt" = nondt)
  return(parameter_set)
}

# Sample data using simulation settings and 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$nP){
      this.sub <- which(data[,1]==i)
      accuracy = 0
      while(sum(accuracy)==0){
            temp <- wdmrnd(a = parameter_set$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 full data: mean acc and correct-rt mean and var
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$sub, sum)
  always_0 = which(sum_correct==0)
  if(length(always_0)!=0){
    bad_participants = (data$sub %in% always_0)
    data = data[-bad_participants,]
    sum_correct     = tapply(data$accuracy, data$sub, sum)
  }
  mean_accuracy   = tapply(data$accuracy, data$sub, mean)

  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)

  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

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

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))
}

Hddm_runJAGS <- function(getData, settings, n.chains,
                              modelFile="./EZHBDDM.bug",
                              samplesFile=NA, plot.Chains = FALSE){
    parameters <- c("bound_mean", "drift_mean", "nondt_mean", "bound", "nondt",
                    "drift_sdev", "nondt_sdev", "bound_sdev", "drift")
    myinits    <- default_inits(n.chains, settings$nPart)

    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)

    write_JAGSmodel(settings$prior)
    data <- getData$jagsData

    library(R2jags)
    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(!is.na(samplesFile)){
       save(samples,file=samplesFile)
    }
    if(plot.Chains){
       plot.Chain(samples)
    }

    # Isolate samples related to the drift parameter
    samples.drift = extractSamples("drift", samples)
    samples.drift_mean = extractSamples("drift_mean", samples)
    samples.drift_sdev = extractSamples("drift_sdev", samples)
    # Isolate samples related to the bound parameter
    samples.bound = extractSamples("bound", samples)
    samples.bound_mean = extractSamples("bound_mean", samples)
    samples.bound_sdev = extractSamples("bound_sdev", samples)
    # Isolate samples related to the nondt parameter
    samples.nondt = extractSamples("nondt", samples)
    samples.nondt_mean = extractSamples("nondt_mean", samples)
    samples.nondt_sdev = extractSamples("nondt_sdev", samples)
    estimates <- list("drift" = apply(samples.drift,3,mean), "drift_sdev" = mean(samples.drift_sdev),
                      "bound" = apply(samples.bound,3,mean), "drift_mean" = mean(samples.drift_mean),
                      "nondt" = apply(samples.nondt,3,mean), "bound_mean" = mean(samples.bound_mean),
                      "bound_sdev" = mean(samples.bound_sdev), "nondt_mean" = mean(samples.nondt_mean),
                      "nondt_sdev" = mean(samples.nondt_sdev))
return(estimates)
}

Hddm_runSim <- function(nParticipants, nTrials,
                        n.chains = 4, samplesFile = NA){

      design.parameters <- Hddm_Parameter_Set(nParticipants,nTrials)
      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))
```

```
}
```

Run simulations

Simple example

```
set.seed(123)

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

design.sample_parameters()
design.sample_data()
design.estimate_parameters()
```

Simulation study (200 repetitions)

```
nSim <- 200
prior <- default_priors()
nParticipants <- 50
nTrials <- 150
n.chains = 4
save_sampleFiles = FALSE
samplesFileName = "samples.RData"
settings <- list("nPart"   = nParticipants,
                 "nTrials" = nTrials,
                 "prior"   = prior)

nSim <- 200
prior <- default_priors()
settings <- list("nPart"   = 50,
                 "nTrials" = 150,
                 "prior"   = prior)


tru = [Hddm_Parameter_Set()] * K
est = [Hddm_Parameter_Set()] * K
err = [Hddm_Parameter_Set()] * K

for(k in 1:nSim){
    set.seed(k)
    cat("Iteration", k+1, "of", nSim)
    design = Hddm_Design(participants=20, trials=50, prior=prior)
    design.sample_parameters()
    design.sample_data()
    #print(design.parameter_set)
    #design.data.summary()
    design.estimate_parameters()
    tru[k] = design.parameter_set
    est[k] = design.estimate
    if design.estimate is not None:
        err[k] = (design.estimate - design.parameter_set)
```

```
    else:
        err[k] = None
    if (k+1) % 100 == 0:
        print(f'. {k+1} of {K}\n', end='')
    else:
        print('.', end='')
}
```

```
def recovery_plot(x, y, parameterName, ttl):
    fontsize = 10

    plt.figure(figsize=(2, 2))

    plt.scatter(x, y, color='b', s=3)
    plt.grid()
    plt.gca().set_aspect('equal')

    xax = np.linspace(min(x), max(x), 100)

    plt.plot(xax, xax, '--')

    plt.xlabel('Simulated value', fontsize=10)
    plt.title('Group mean ' + parameterName, fontsize=10)

    output_path = "ezrecovery_" + parameterName + ".pdf"
    plt.savefig(output_path, format='pdf', bbox_inches='tight')

    plt.show()
```

```
x = [np.nan] * K
y = [np.nan] * K
for k in range(K):
    if err[k] is not None:
       x[k] = tru[k].nondt_mean
       y[k] = est[k].nondt_mean

recovery_plot(x, y, 'nondt', 'Group mean nondt')

x = [np.nan] * K
y = [np.nan] * K
for k in range(K):
    if err[k] is not None:
       x[k] = tru[k].drift_mean
       y[k] = est[k].drift_mean

recovery_plot(x, y, 'drift', 'Group mean drift')

x = [np.nan] * K
y = [np.nan] * K
for k in range(K):
    if err[k] is not None:
       x[k] = tru[k].bound_mean
       y[k] = est[k].bound_mean

recovery_plot(x, y, 'bound', 'Group mean bound')
```

```
x = np.empty(0)
y = np.empty(0)
for k in range(K):
    if err[k] is not None:
       x = np.append(x, tru[k].drift)
       y = np.append(y, est[k].drift)

recovery_plot(x, y, 'drift', 'Individual drift rates')
```