Simulate and fit your own drift-duffison models (DDMs)!

Hierarchical DDMs in brms

1. (5min) A random walk process is a model in which the next step of a process comes from a distribution. This next step is then added to the result of the previous step.

Let us assume that a participant collects evidence for the target during one experimental trial in a process that follows a random walk. The random walk is *simulated* by the following R code:

```
set.seed(1)
nwalks = 50; nsteps = 100; step_length = .01; standard_deviation = 0.1
time = seq(0, 1, length.out=nsteps)
random_walks = matrix(0, nrow=nsteps, ncol=nwalks)
for(w in 1:nwalks)
{
    this_random_walk = 0.5
    for(s in 2:nsteps)
    {
        evidence_units = rnorm(1, 0.01, standard_deviation)
        this_random_walk[s] = this_random_walk[s-1] + evidence_units
    }
    random_walks[,w] = this_random_walk
}

matplot(time, random_walks,type='l')
```

1a. Change the standard deviation of the random walk to 0.001, what happens? How would you describe this graph?

1b. Change the standard deviation of the random walk to 1, what happens? What has changed compared to the original figure?

Now let's *simulate* the PDM data set from one participant. This simulation is based on a random walk process that passes one of two boundaries that correspond to a correct and incorrect choice respectively. This time of the boundary crossing also describes the response time (when also accounting for *non-decision time*).

```
ntrials = 200; nsteps = 300; step_length = .01
time = seq(0, nsteps*step_length, length.out=nsteps)
boundary = 1.2; drift = 1.5; ndt = 0.2; dc = 1
random_walks = matrix(0, nrow=nsteps, ncol=ntrials)
rts = rep(0, ntrials); correct = rep(1, ntrials)
set.seed(1)
for(n in 1:ntrials)
{
    random_walks[1, n] = 0.5*boundary  # relative start point = 0.5
    for(s in 2:nsteps)
    {
        random_walks[s, n] = random_walks[s-1, n] + rnorm(1, drift*step_length, dc*sqrt(step_length))
        if(random_walks[s, n] >= boundary)
        {
            random_walks[-(1:(s-1)), n] = boundary
            rts[n] = s*step_length + ndt
            correct[n] = 1
            break
        } else if (random_walks[s, n] <= 0) {
            random_walks[-(1:(s-1)), n] = 0
            rts[n] = s*step_length + ndt
            correct[n] = 0
            break
        } else if (s == nsteps) {
            correct[n] = NA; rts[n] = NA
            break
        }
    }
}
x11(); matplot(time, random_walks,type='l')
x11(); hist((correct * 2 - 1) * rts, breaks=20)
simulate_data = data.frame(RT = rts, accuracy = correct)
```

Now let's *fit* the simulated data. Below is the code to fit the EZ Diffusion model to the simulated data.

```
ezdiff = function(rt, correct, s=1.0)
{
    if (length(rt) <= 0) stop('length(rt) <= 0')
    if (length(rt) != length(correct)) stop('RT and correct unequal lengths')
    if (max(correct, na.rm=TRUE) > 1) stop('Correct values larger than 1')
    if (min(correct, na.rm=TRUE) < 0) stop('Correct values smaller than 0')
    pc = mean(correct, na.rm=TRUE)
    if (pc <= 0) stop('Mean accuracy less than or equal to 0')
    # subtract or add 1/2 an error to prevent division by zero
    if (pc == 1.0) {pc=1 - 1/(2*length(correct))}
    if (pc == 0.5) {pc=0.5 + 1/(2*length(correct))}
    MRT = mean(rt[correct == 1], na.rm=TRUE)
    VRT = var(rt[correct == 1], na.rm=TRUE)
    if (VRT <= 0) stop('RT variance less than or equal to 0')
    r=(qlogis(pc)*(((pc**2)*qlogis(pc)) - pc*qlogis(pc) + pc - 0.5))/VRT
    drift=sign(pc-0.5)*s*(r)**0.25
    boundary=(s**2 * qlogis(pc))/drift
    y=(-1*drift*boundary)/(s**2)
    MDT=(boundary/(2*drift))*((1-exp(y))/(1+exp(y)))
    ndt=MRT-MDT
    return(list(boundary, drift, ndt))
}
est_params = ezdiff(simulate_data$RT, simulate_data$accuracy)

# The estimate of the drift rate in evidence units per second
est_params[2]

# The estimate of the boundary in evidence units
est_params[1]

# The estimate of the non-decision time in seconds
est_params[3]
```

Below is the code to fit the Bayesian DDM using brms to the simulated data.

```
library(brms)

#Define the formula for brms
ddm_formula <- bf(RT | dec(accuracy) ~ 1,
                  bs ~ 1,
                  ndt ~ 1,
                  bias = 0.5)

our_prior <- c(
 set_prior("normal(0, 2)", class = "Intercept"),
 set_prior("normal(1, 0.25)", class = "Intercept", dpar = "bs"),
 set_prior("exponential(4)", class = "Intercept", dpar = "ndt")
)

# Define the initial values for each Markov chain.
```

```r
# Note that brms and Stan do not have good default start values for DDMs.
mcmc_initials <- function() {
  list(
    Intercept = runif(1, -4, 4),
    Intercept_bs = runif(1, 0.5, 2),
    Intercept_ndt = runif(1, 0, min(simulate_data$RT))
  )
}

bayes_ddm <- brm(ddm_formula,
                 family=wiener(link_bs ="identity",link_ndt="identity"),
                 init=mcmc_initials,
                 prior=our_prior,
                 data=simulate_data)
summary(bayes_ddm)
x11()
plot(bayes_ddm)
```

2. (10min) Fit this cognitive model to the simulated data using both EZ Diffusion and a Bayesian analysis with brms. Do both fitted models recover the true simulated cognitive parameters? What do you learn from the Bayesian DDM that you do not learn from the EZ Diffusion?

3. (Bonus) Fit the EZ Diffusion and brms DDM to one participant from the real data set in previous sessions. Note any differences between the two procedures. Which procedure is more trustworthy?