Joint modeling workshop:

An exploration of simple joint modeling using BayesFlow

Unlocking New Horizons in Cognitive Modeling with Simulation-Based Inference 2026

Michael D. Nunez
University of Amsterdam, Psychological Methods

Google Colab

https://tinyurl.com/ColabDemoIntegrative

Colab example from this preprint

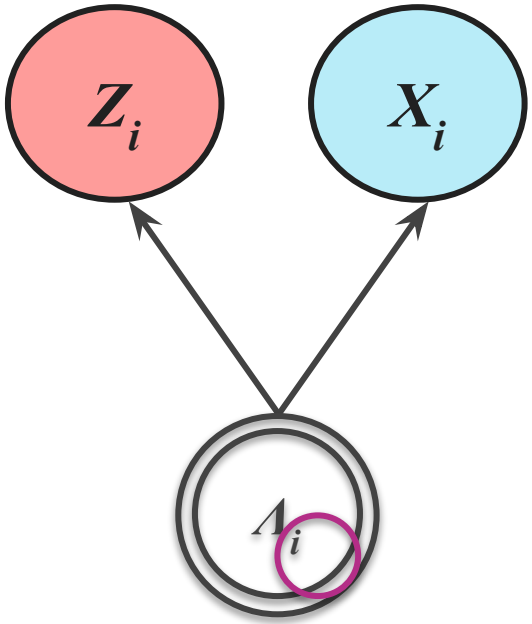
PDF of our CCN 2026 poster

Bruny Krijgsman
PhD candidate at Erasmus University Rotterdam

Required exercise 1 (easy):

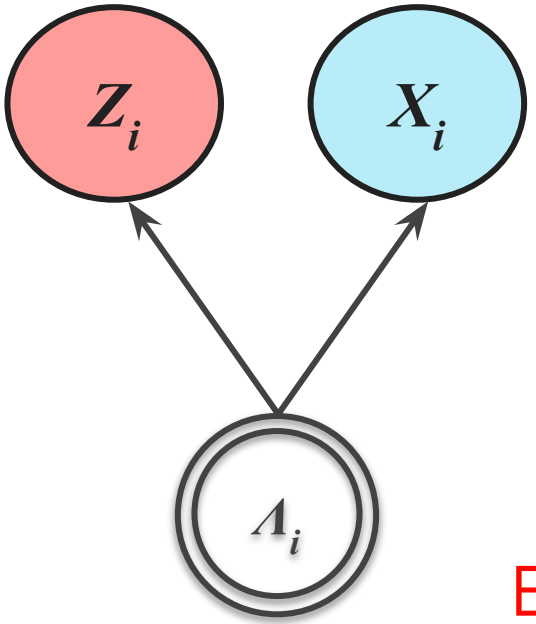
Actually run the code yourself on Colab

(Ask fellow students or me for help)



Single-trial Integrative Models

Ghaderi-Kangavari, Rad, Nunez. 2023; Computational Brain & Behavior



$$x_i \sim DDM(\alpha, \tau, \delta_i, \beta)$$

$$\delta_i \sim N(\mu_\delta, \eta_\delta^2)$$

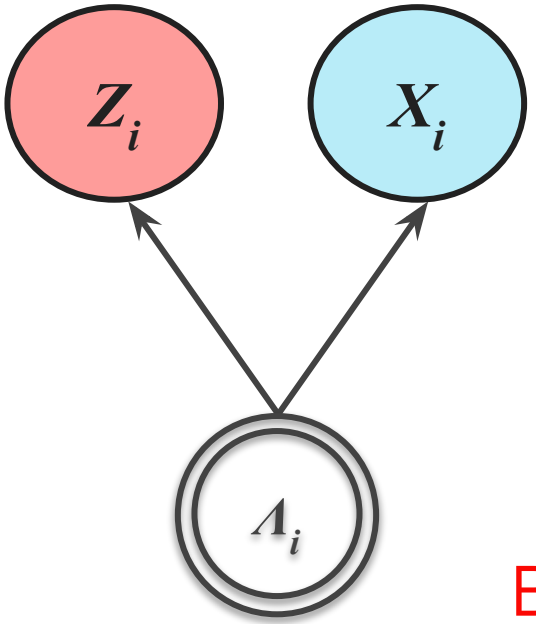
$$z_i \sim N(\gamma\delta_i, \sigma^2)$$

Brain measures and Behavior are both dependent on cognitive parameters

Optional Exercise 2 (medium difficulty):

Change the code in integrative_model/simulation.py and integrative_model/workflow.py to add an *intercept parameter* in the last probabilistic statement

(Ask me or fellow students for help)



$$x_i \sim DDM(\alpha, \tau, \delta_i, \beta)$$

$$\delta_i \sim N(\mu_\delta, \eta_\delta^2)$$

$$z_i \sim N(\gamma\delta_i + b, \sigma^2)$$

Brain measures and Behavior are both dependent on cognitive parameters

Optional Exercise 3 (advanced):

Fork my improved_demo branch and improve the code based on current BayesFlow recommendations (see BayesFlow lectures and the amoritized-workflow agent)

(Ask Stefan and Jerry for help)

I don’t have solution documents!

If you feel your code has a solution (e.g. your github fork) post the link in Discord, I’ll look at it!

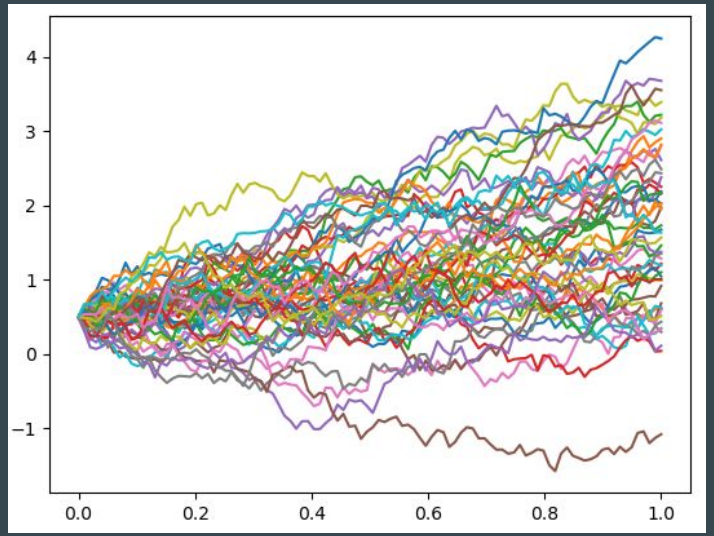
This could be soon, later today, or next week.

Random Walks (Python code)

A random walk process is a model in which the next step of a process comes from a distribution which is then added to the result of the previous step.

Financial markets are a good example of random walk processes

```
plt.figure()
np.random.seed(1)
nwalks = 50
nsteps = 100
step_length = .01
time = np.linspace(0, 1, num=nsteps)
for w in range(nwalks):
    random_walk = np.empty(nsteps)
    random_walk[0] = 0.5
    for s in range(1,nsteps):
        evidence_units = stats.norm.rvs(loc=0.01, scale=0.1)
        random_walk[s] = random_walk[s-1] + evidence_units
    plt.plot(time, random_walk)
```

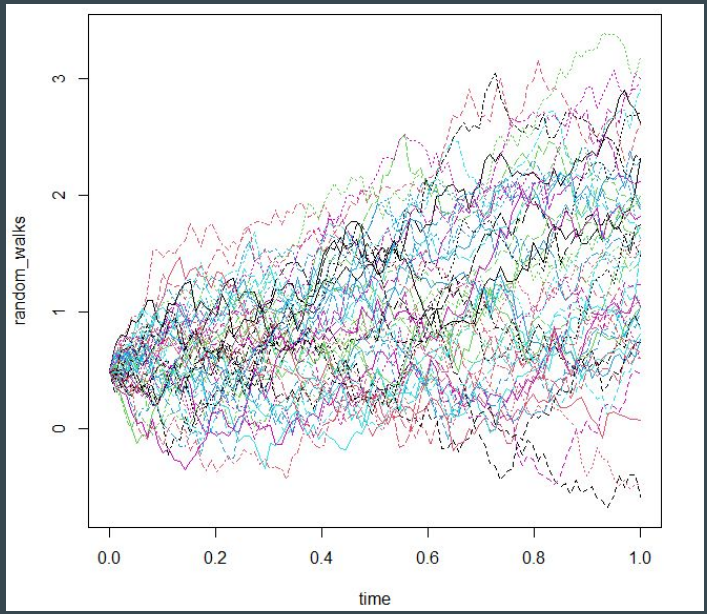


Random Walks (R code)

A random walk process is a model in which the next step of a process comes from a distribution which is then added to the result of the previous step.

Financial markets are a good example of random walk processes

```
set.seed(1)
nwalks = 50; nsteps = 100; step_length = .01
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, 0.1)
      this_random_walk[s] = this_random_walk[s-1] + evidence_units
   }
   random_walks[,w] = this_random_walk
}
x11()
matplot(time, random_walks,type='l')
```

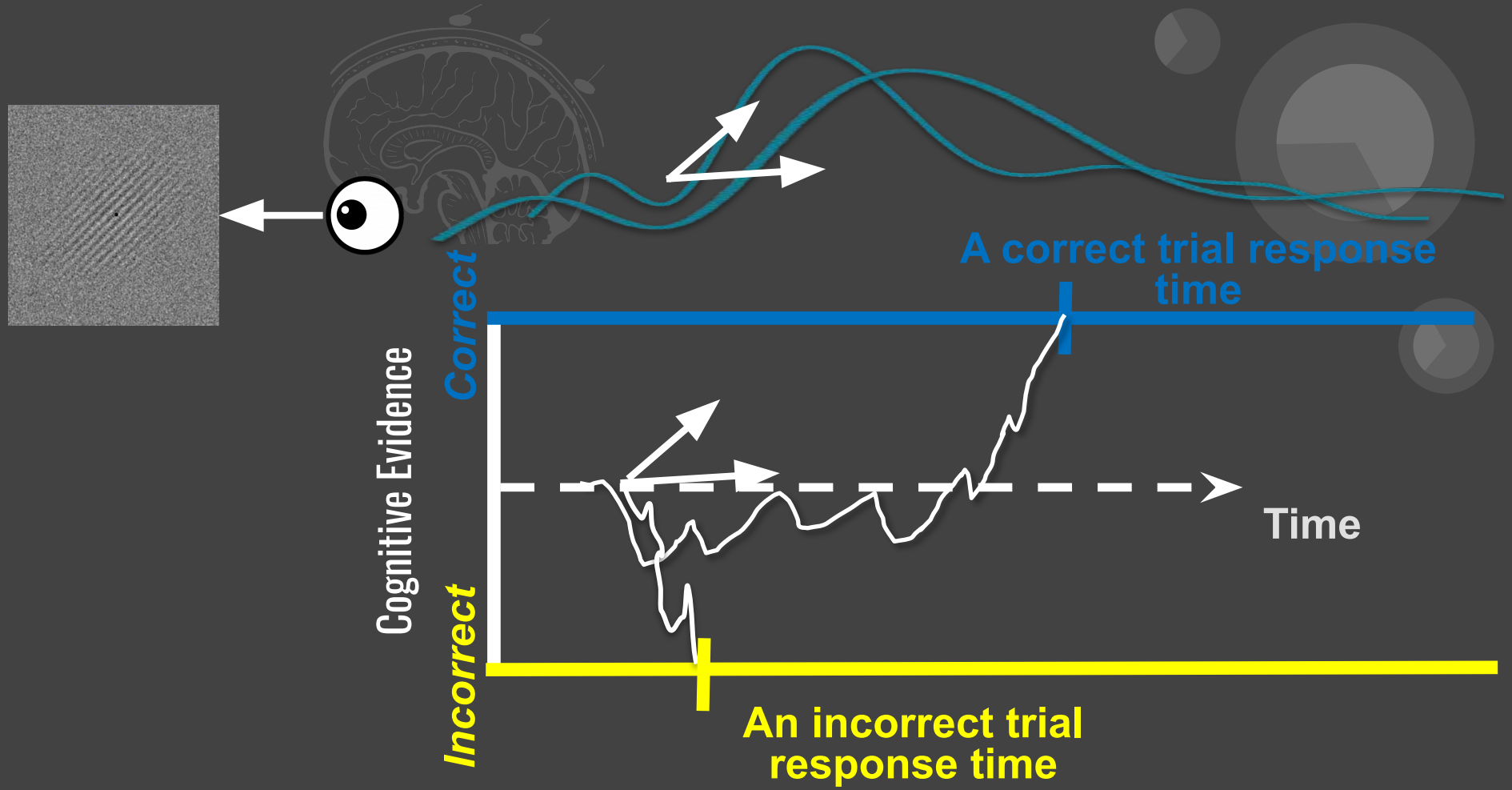


Simulating Drift Diffusion Model (DDM) in Python

```
ntrials = 200; nsteps = 300; step_length = .01
boundary = 1.2; drift = 1.5; ndt = 0.2; dc = 1
rts = np.empty(ntrials)  # Simulated correct response times
correct = np.empty(ntrials)
plt.figure()
time = np.linspace(0, step_length*nsteps, num=nsteps)
np.random.seed(1)  # Try different random seeds
for n in range(ntrials):
    random_walk = np.empty(nsteps)
    random_walk[0] = 0.5*boundary  # relative start point = 0.5
    for s in range(1,nsteps):
        random_walk[s] = random_walk[s-1] + stats.norm.rvs(loc=drift*step_length, scale=dc*np.sqrt(step_length))
        if random_walk[s] >= boundary:
            random_walk[s:] = boundary
            rts[n] = s*step_length + ndt
            correct[n] = 1
            break
        elif random_walk[s] <= 0:
            random_walk[s:] = 0
            rts[n] = s*step_length + ndt
            correct[n] = 0
            break
        elif s == (nsteps-1):
            correct[n] = np.nan; rts[n] = np.nan
            break
    plt.plot(time + ndt, random_walk)
plt.xlim([0, 2]); plt.xlabel('Time (secs)'); plt.ylabel('Cognitive / Neural Evidence')
plt.figure()
plt.hist((correct * 2 - 1) * rts, bins=20)  # Typical method of plotting choice-RTs
plt.xlabel('Choice * Response Time (sec)')
print(np.nanmean(correct)); print(np.nanmean(rts))
```

Simulating Drift Diffusion Model (DDM) in R

```
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)
mean(correct, na.rm=TRUE); mean(rts, na.rm=TRUE)
```


A correct trial response time
Correct
Cognitive Evidence
Time
Incorrect
An incorrect trial response time