Final Project 2023

Md Delwar Hossain

19/08/2023

Agreement attraction in sentence processing: Grammaticality-judgment study

In a study focused on grammatical judgment, participants were presented with sentences and asked to determine if each sentence was grammatical (denoted as "F") or ungrammatical (denoted as "J"). Additionally, their response times were recorded along with their judgments. The aim of this research is to investigate the agreement attraction phenomenon in sentence processing, and it involves examining the following example sentences:

a. Attractor condition: The key to the cabinets are rusty.
b. Baseline condition: The key to the cabinet are rusty.

The findings indicate that participants took longer to respond in the attractor condition compared to the baseline condition. The dataset contains responses and response times for various sentences with these conditions, gathered during the experiment.

```
load("project-data.Rda")
head(data)
```

```
##   Condition cond        RT      response
## 1 Attractor    a 0.7851932 Ungrammatical
## 2  Baseline    b 0.9532993 Ungrammatical
## 3 Attractor    a 0.6631340 Ungrammatical
## 4  Baseline    b 1.0253926 Ungrammatical
## 5 Attractor    a 0.8176316 Ungrammatical
## 6  Baseline    b 0.9891965 Ungrammatical
```

Problem 1: Implement a simple accumulator model

To facilitate computation and analysis, I begin by manipulating the data as follows:

1. I create a new column called *ncond*, where the value is "1" if the condition is "Attractor" and "0" if it is any other condition (such as "Baseline" in this case).
2. I also create another column called *nresp* to convert the categorical values of the response column into integers.

After implementing these changes, the dataset will be better suited for numerical computations and statistical analysis.

```r
data_new <- data %>%
  mutate(ncond = ifelse(cond == "a", 1, 0),
         nresp = ifelse(response == "Ungrammatical", 0, 1),
         response_time_ms = RT*1000) # convert rt in milliseconds
data_new = data_new[1:100,] # for faster computation
```

To model the given problem using a simple accumulator model, we introduce two accumulators: one for grammatical decisions and another for ungrammatical decisions. Each accumulator travels a certain distance D, also known as the decision threshold, before making a decision. The drift velocity for each accumulator is assumed to be drawn from a Log-normal distribution.

$$V \sim LogNormal(\mu_v, \sigma_v) \tag{1}$$

We then define location of distribution for drift velocity as following:

$$\mu_v = \alpha + cond\_a_n \cdot \beta_{cond\_a} \tag{2}$$

It should be noted that the equation provided is a general one applicable to both accumulators. There are a total of four parameters to consider: two for the grammatical accumulator, denoted as $\alpha_g rammatical$ and β_g, and two for the ungrammatical accumulator, denoted as α_{ug} and β_{ug}. Additionally, there is a non-decision time ($non_decision_time$) which represents the time taken before the subject reads the sentence or any miscellaneous time not directly related to the grammatical judgment task.

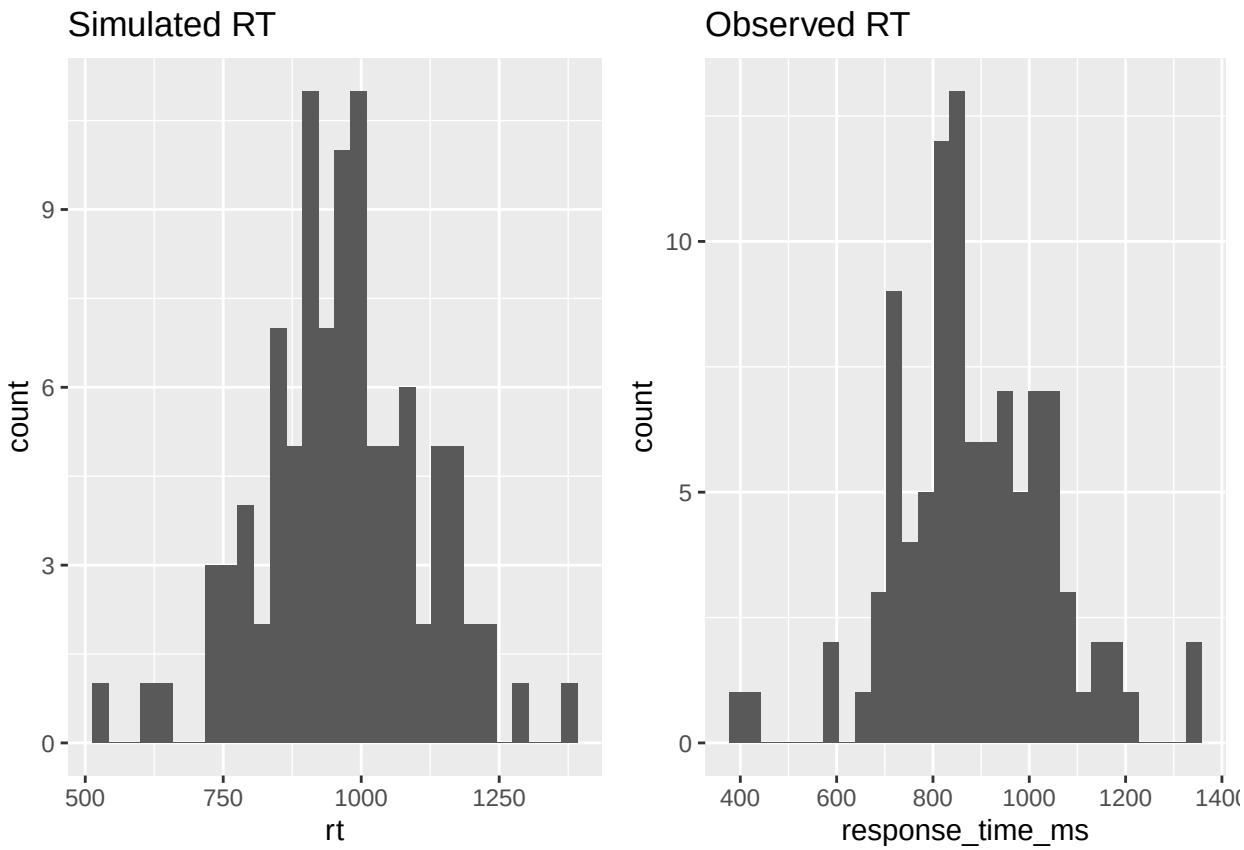
Prior to commencing the fitting process, we assume certain values as potential truth values for the various parameters involved in the accumulator model. Subsequently, we generate simulated data based on these assumed values. To validate the model, we compare the simulated data with the observed data by plotting them side by side.

```r
set.seed(123)
D <- 1800
alpha_ungrammatical <- 0.4 # bias for ungrammatical accumulator
alpha_ungrammatical_a <- 0.2 # effect of condition a
alpha_grammatical <- 0.8 # bias for grammatical accumulator
beta_grammatical_a <- -0.2 # effect of condition a
sigma <- .2
non_decision_time <- 100 # non decision time in ms

mu_ungrammatical <- alpha_ungrammatical + data_new$ncond * alpha_ungrammatical_a # location of rate of
mu_grammatical <- alpha_grammatical + data_new$ncond * beta_grammatical_a # location of rate of accumul
N=100

V_ungrammatical <- rlnorm(N, mu_ungrammatical, sigma) # rate of accumulation of evidence for ungrammati
V_grammatical <- rlnorm(N, mu_grammatical, sigma) # rate of accumulation of evidence for grammatical
T_ungrammatical <-  D / V_ungrammatical # Time taken by ungrammatical accumulator
T_grammatical <- D / V_grammatical # Time taken by grammatical accumulator

T_winner <- pmin(T_ungrammatical, T_grammatical)
accumulator_winner <- ifelse(T_ungrammatical == T_winner,
                             "umgrammatical",
                             "grammatical")

rt_sim = non_decision_time + T_winner
df_rt_sim = data_frame(rt=rt_sim)
```

```
## Warning: ‘data_frame()‘ was deprecated in tibble 1.1.0.
## i Please use ‘tibble()‘ instead.
## This warning is displayed once every 8 hours.
## Call ‘lifecycle::last_lifecycle_warnings()‘ to see where this warning was
## generated.
```

```
rt_sim_plot = ggplot(df_rt_sim, aes(rt)) +
  geom_histogram() + ggtitle('Simulated RT')
rt_obs_plot = ggplot(data_new, aes(response_time_ms)) +
  geom_histogram() + ggtitle('Observed RT')

ggarrange(rt_sim_plot, rt_obs_plot,ncol=2, nrow=1)
```

```
## ‘stat_bin()‘ using ‘bins = 30‘. Pick better value with ‘binwidth‘.
## ‘stat_bin()‘ using ‘bins = 30‘. Pick better value with ‘binwidth‘.
```



In the previous simulated data, the decision threshold for accumulators, denoted as "D," was treated as a constant value. However, in real-world scenarios, this assumption does not hold true. Thus, we propose that "D" follows a lognormal distribution, similar to the distribution of drift velocity for accumulators.

To prepare for model fitting, we conduct prior prediction. This involves defining priors for the various parameters involved. To establish these priors, we draw inspiration from priors used in similar models presented in lectures or exercises. It is essential to maintain a holistic approach to sentence processing rather than focusing solely on individual words.

$$
\begin{aligned}
T_{nd} &\sim Normal(150, 100) \text{ with } 0 < T_{nd} < min(rt_n) \\
\boldsymbol{\alpha} &\sim Normal(7, 1) \\
\boldsymbol{\beta} &\sim Normal(0, .2) \\
\sigma &\sim Normal_+(.5, .2)
\end{aligned} \tag{3}
$$

$non_decision_time$ represents the non-decision time, and its prior is expressed in milliseconds. On the other hand, parameters such as α, β, and σ are represented in the log millisecond scale. For the likelihood function, we employ the lognormal distribution.

$$
T_n \sim LogNormal(\alpha - cond\_a_n * \beta, \sigma) \tag{4}
$$

Once more, we employ a general equation to define the likelihood. Here, α represents the disparity between the locations of two distributions: one for the decision threshold and the other for the drift velocity.

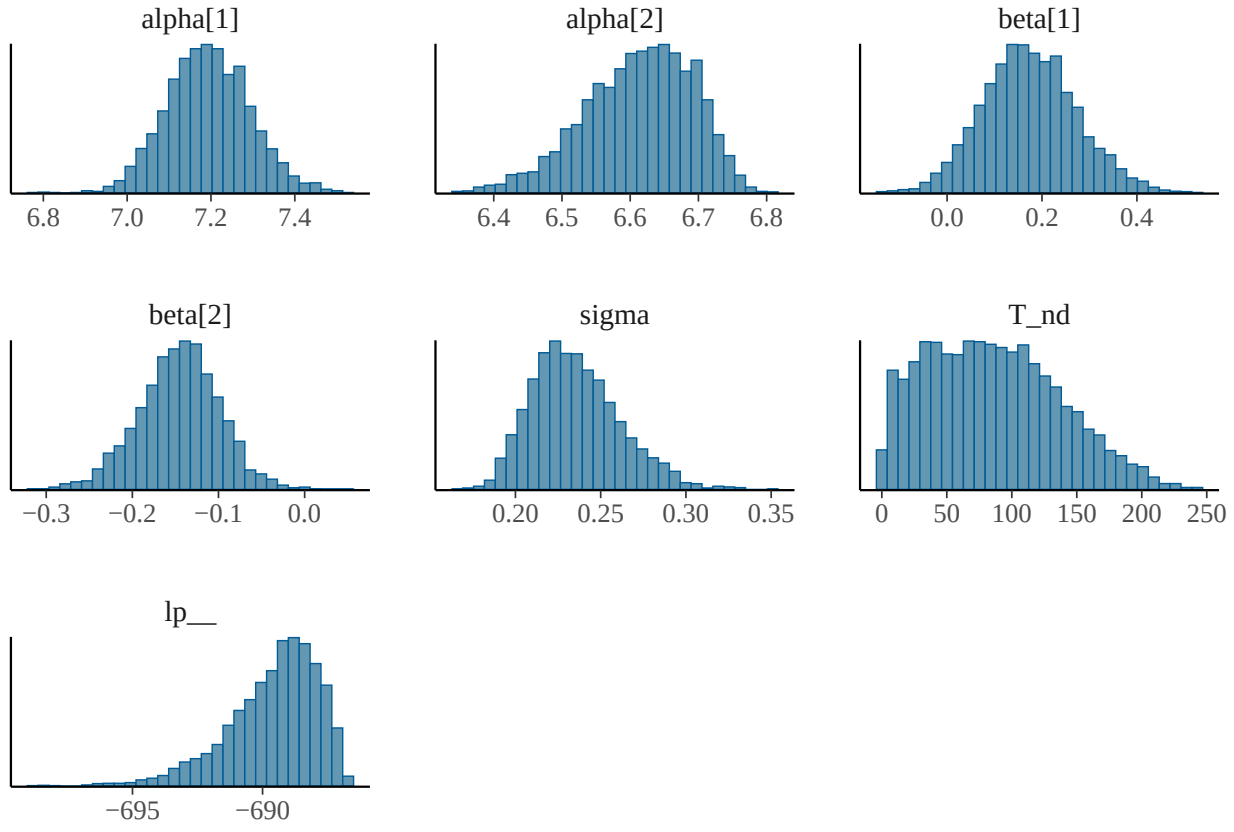
Stan Model for "attraction.stan" :

```
data {
  int<lower = 1> N;
  vector[N] ncond;
  vector[N] RT;
  int nresp[N];
}
parameters {
  real alpha[2];
  real beta[2];
  real<lower = 0> sigma;
  real<lower = 0, upper = min(RT)> T_nd;
}
model {
  vector[N] T = RT - T_nd;
  target += normal_lpdf(alpha | 7, 1);
  target += normal_lpdf(beta | 0, .2);
  target += normal_lpdf(sigma | .5, .2)
    - normal_lccdf(0 | .5, .2);
    target += normal_lpdf(T_nd | 150, 100)
    - log_diff_exp(normal_lcdf(min(RT) | 150, 100),
                   normal_lcdf(0 | 150, 100));
  for(n in 1:N){
    if(nresp[n] == 1) // grammatical
      target += lognormal_lpdf(T[n] | alpha[1] -
                               ncond[n] * beta[1], sigma)  +
        lognormal_lccdf(T[n] | alpha[2] -
                        ncond[n] * beta[2] , sigma);
    else
       target += lognormal_lpdf(T[n] | alpha[2] -
                                ncond[n] * beta[2], sigma) +
        lognormal_lccdf(T[n] | alpha[1] -
                        ncond[n] * beta[1], sigma);
  }
}
```

```
attractor_model_onlyprior <- list(onlyprior=1,
                     N = nrow(data_new),
                     RT = data_new$response_time_ms,
                     nresp = data_new$nresp,
                     ncond = data_new$ncond)
prior_attractor_model <- stan("attraction.stan",
                      data = attractor_model_onlyprior,
                      control = list(max_treedepth = 15,adapt_delta = .99999),
                     )
```

```
prior_estimates <- as.data.frame(prior_attractor_model)
mcmc_hist(prior_estimates)
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



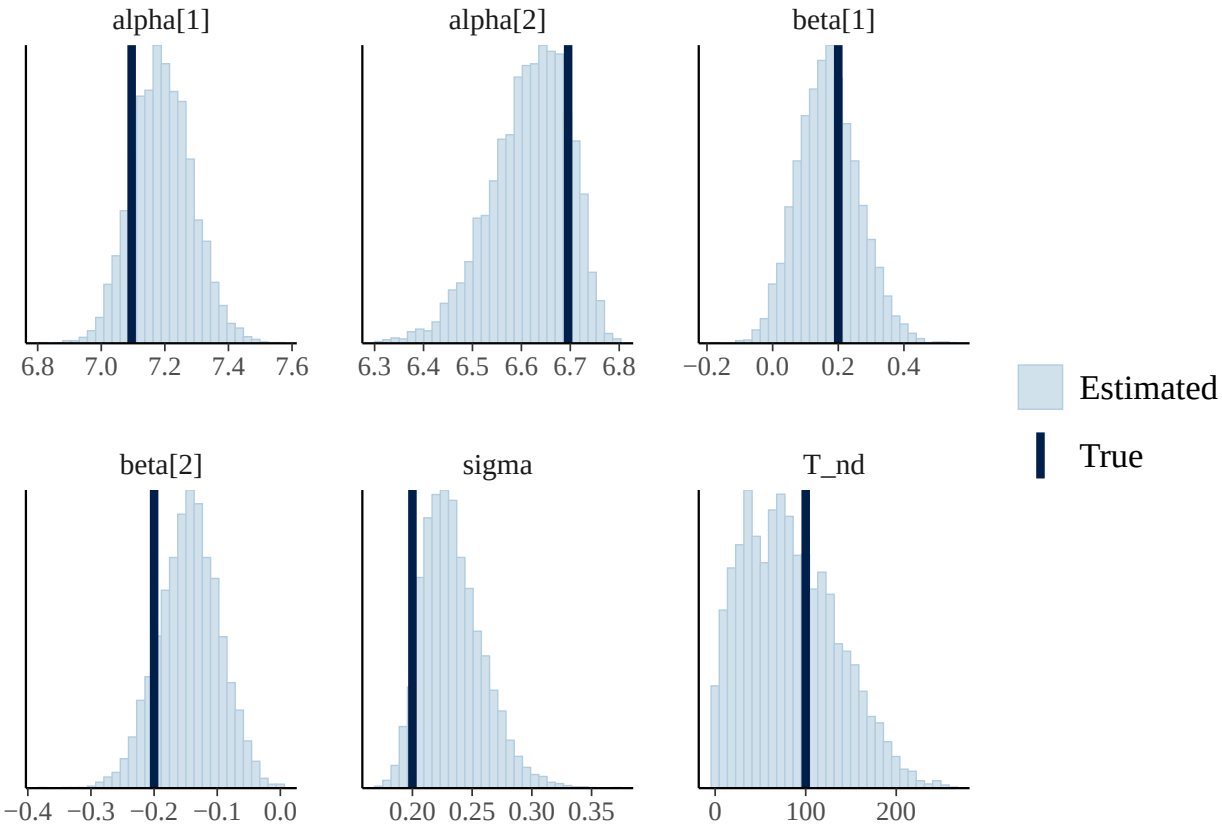
Now we fit attraction effect model,

```
attractor_model <- list(N = nrow(data_new),
                     RT = data_new$response_time_ms,
                     nresp = data_new$nresp,
                     ncond = data_new$ncond)
fit_attractor_model <- stan("attraction.stan",
                      data = attractor_model,
                      control = list(max_treedepth = 15,adapt_delta = .99999),
                     )
```

Here we plot parameters of attractor model.

```
true_values <- c(log(D) - alpha_ungrammatical,
                 log(D) - alpha_grammatical,
                 alpha_ungrammatical_a,
                 beta_grammatical_a,
                 sigma,
                 non_decision_time)
estimates <- as.data.frame(fit_attractor_model) %>%
  select(- lp__)
mcmc_recover_hist(estimates, true_values)
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



Now we create a null model, where we donot consider any effect of attractor condition.

Stan Model for "attraction_null.stan" :

```
data {
  int<lower = 1> N;
  vector[N] ncond;
  vector[N] RT;
  int nresp[N];
}
parameters {
  real alpha[2];
```

```
  real<lower = 0> sigma;
  real<lower = 0, upper = min(RT)> T_nd;
}
model {
  vector[N] T = RT - T_nd;
  target += normal_lpdf(alpha | 7, 1);
  target += normal_lpdf(sigma | .5, .2)
    - normal_lccdf(0 | .5, .2);
  target += normal_lpdf(T_nd | 100, 50)
    - log_diff_exp(normal_lcdf(min(RT) | 100, 50),
                   normal_lcdf(0 | 100, 50));
  for(n in 1:N){
    if(nresp[n] == 1)
      target += lognormal_lpdf(T[n] | alpha[1] , sigma)  +
        lognormal_lccdf(T[n] | alpha[2], sigma);
    else
       target += lognormal_lpdf(T[n] | alpha[2], sigma) +
        lognormal_lccdf(T[n] | alpha[1], sigma);
  }
}
```

```
fit_null_attractor_model <- stan("attraction_null.stan",
                     data = attractor_model,
                     control = list(max_treedepth = 15,adapt_delta = .99999))
```

Bayes factor used to compare the null model and attraction effect model.

```
lml_attraction <- bridge_sampler(fit_attractor_model, silent = TRUE)
lml_null_attraction <- bridge_sampler(fit_null_attractor_model, silent = TRUE)
bayes_factor_attraction <- bridgesampling::bf(lml_attraction, lml_null_attraction)
bayes_factor_attraction
```

```
## Estimated Bayes factor in favor of lml_attraction over lml_null_attraction: 40.61888
```

Comparing the models clearly demonstrates the significant influence of attraction, backed by overwhelming evidence in favor of the attractor model.

Problem 2: Implement a feature migration model

Based on the feature migration theory, the plural feature of "the cabinets" can transfer to the subject noun "the key" in condition (a) with a certain probability denoted as θ. Conversely, condition (b) does not involve such feature migration. In θ x N trials, the plural feature of "the cabinet" migrates to the subject, resulting in a plural subject. As a consequence, the sentence appears grammatically correct, leading to incorrect and slower judgments about its grammaticality. However, in the remaining (1 - θ) x N trials, the judgment times are similar to those in condition (b).

To implement this feature migration assumption, the model employs a mixture of lognormals, with 6 specified parameters. The priors for these parameters are defined as follows:

$$
\begin{aligned}
\mu &\sim Normal_{lb=5.2}(5.5, 0.5) \\
\delta &\sim Normal_{lb=0.1}(0.2, .05) \\
\theta &\sim Normal_{lb=0,ub=1}(0, .25) \\
\sigma &\sim Normal_{+}(.1, .05) \\
P1 &\sim Beta(8, 2) \\
P2 &\sim Beta(2, 8)
\end{aligned}
\tag{5}
$$

The problem's definition establishes that $P1 > P2$ due to the feature migration condition, which increases the likelihood of the subject making mistakes and considering a sentence grammatical.

Subsequently, a prior predictive check is conducted.

Stan Model for "feature_migration_mixture.stan" :

```
data {
  int<lower = 1> N;
  vector[N] ncond;
  vector[N] RT;
  int nresp[N];
}
parameters {
  real<lower = 5.2> mu;
  real<lower = 0> delta;
  real<lower = 0> sigma;
  real<lower = 0.1, upper = 1> theta;
  real<lower = 0.1, upper = 1> p1;
  real<lower = 0.1, upper = p1> p2;
}
model {
  // priors for the task component
  target += normal_lpdf(mu | 5.5, 0.5)
    - normal_lccdf(5.2 | 5.5, .2);
  target += normal_lpdf(sigma | .1, .05)
    - normal_lccdf(0 | .1, .05);
  target += normal_lpdf(delta | .2, .05)
    - normal_lccdf(0 | .2, .05);
  target += normal_lpdf(theta | 0, 0.25);
  target += beta_lpdf(p1 | 8, 2);
  target += beta_lpdf(p2 | 2, 8);
  for(n in 1:N) {
    if(nresp[n] == 1){
      target += log_sum_exp(log(theta) +
                            lognormal_lpdf(RT[n] | mu + ncond[n] * delta, sigma) +
                            bernoulli_lpmf(nresp[n] | p1),
                            log1m(theta) +
                            lognormal_lpdf(RT[n] | mu, sigma) +
                            bernoulli_lpmf(nresp[n] | p2));
    }
    else{
      target += lognormal_lpdf(RT[n] | mu, sigma) +
                            bernoulli_lpmf(nresp[n] | p2) ;
    }
```

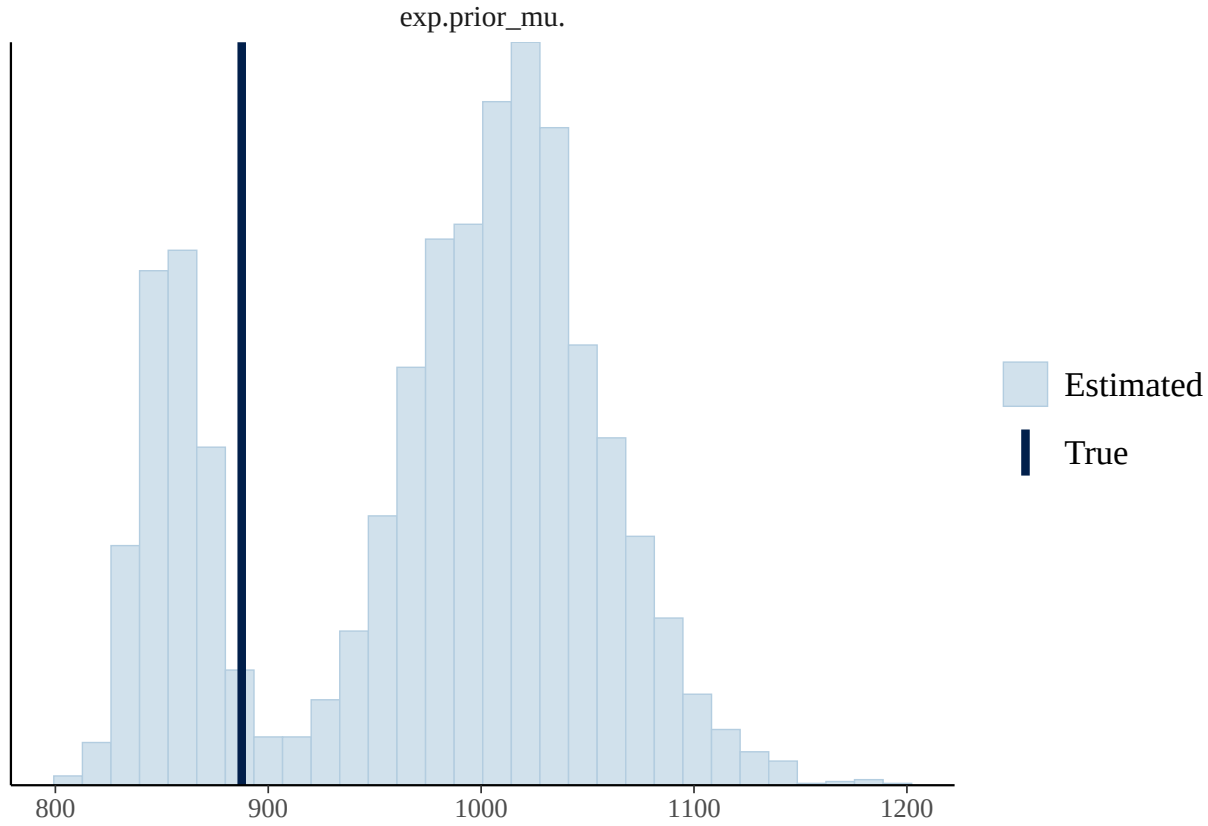
```
}
}
```

```
feature_migrature_mixture_model_prioronly <- list(prioronly=1,
                                    N = nrow(data_new),
                                    RT = data_new$response_time_ms,
                                    nresp = data_new$nresp,
                                    ncond = data_new$ncond)
prior_mix_rt = stan(file = 'feature_migration_mixture.stan',
               data = feature_migrature_mixture_model_prioronly,
               control = list(max_treedepth = 15,adapt_delta = .99999) )
```

Analyzing the distribution of the predictive data's location relative to the mean of the observed data allows us to comprehend the effectiveness of capturing the prior predictive data. The graph representation confirms that the selected prior is indeed successful in capturing the data.

```
true_value <- c(mean(data_new$response_time_ms))
prior_theta = as.data.frame(prior_mix_rt)$theta
prior_mu = rep(0,4000)
for(i in 1:4000){
  theta = rbern(1,prior_theta[1])
  if(theta==0){
    prior_mu[i] = sample(as.data.frame(prior_mix_rt)$mu, 1)
  }else{
    prior_mu[i] = sample(as.data.frame(prior_mix_rt)$mu, 1) + sample(as.data.frame(prior_mix_rt)$delta,
  }
}
estimates = data.frame(exp(prior_mu))
mcmc_recover_hist(estimates, true_value)
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



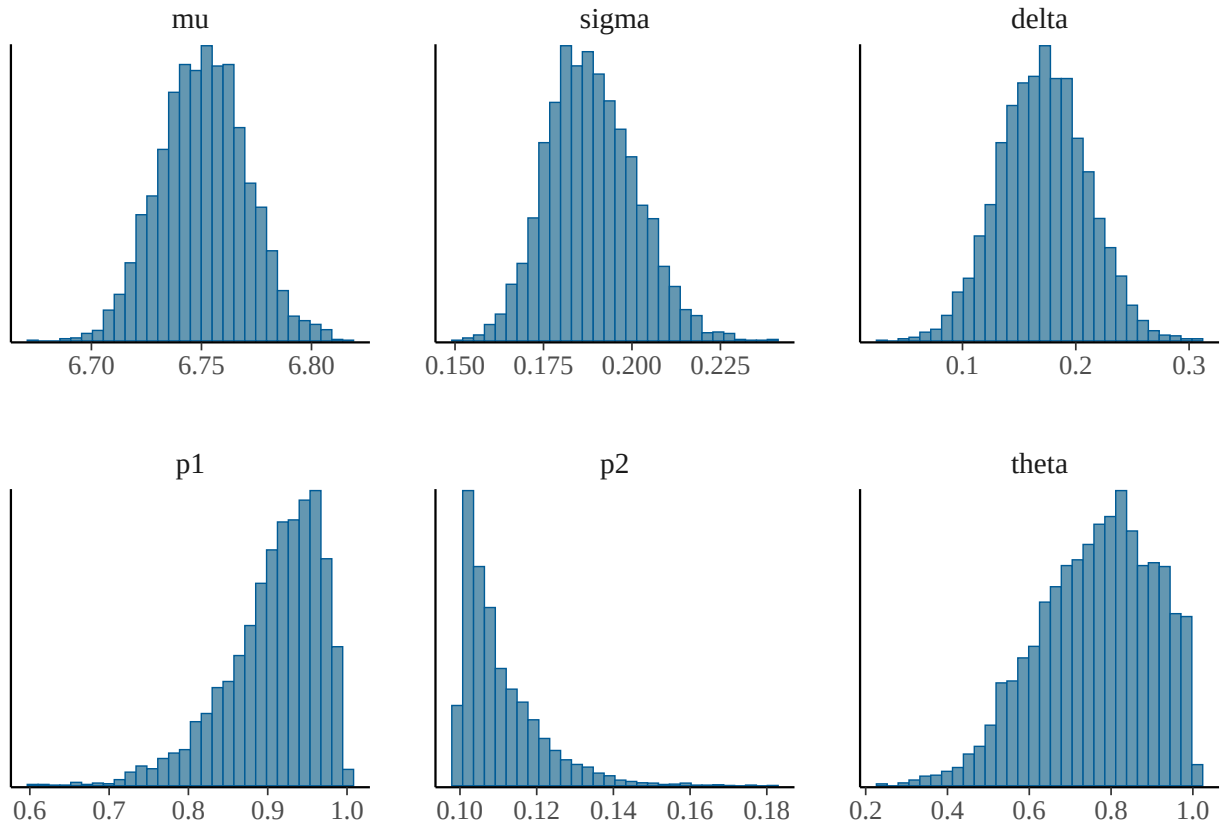
Finally, we successfully apply the data to the model.

```
feature_migrature_mixture_model <- list(N = nrow(data_new),
                     RT = data_new$response_time_ms,
                     nresp = data_new$nresp,
                     ncond = data_new$ncond)
fit_mix_rt = stan(file = 'feature_migration_mixture.stan',
               data = feature_migrature_mixture_model,
               control = list(max_treedepth = 15,adapt_delta = .99999) )
```

Graphs representing the posterior distribution of parameters.

```
df_fit_mix_rt <- as.data.frame(fit_mix_rt)
mcmc_hist(df_fit_mix_rt, pars = c("mu", "sigma", "delta", "p1", "p2", "theta"))
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



The migration rate feature, denoted by $\theta = 0$, is indicative of a null feature migration model.

Stan Model for "feature_migration_mixture_null.stan" :

```
data {
  int<lower = 1> N;
  vector[N] ncond;
  vector[N] RT;
  int nresp[N];
}
parameters {
  real<lower = 5.2> mu;
  real<lower = 0> delta;
  real<lower = 0> sigma;
  real<lower = 0.1, upper = 1> p2;
}
model {
  // priors for the task component
  target += normal_lpdf(mu | 5.5, 0.5)
    - normal_lccdf(5.2 | 5.5, .2);
  target += normal_lpdf(sigma | .1, .05)
    - normal_lccdf(0 | .1, .05);
  target += normal_lpdf(delta | .2, .05)
    - normal_lccdf(0 | .2, .05);
  target += beta_lpdf(p2 | 2, 8);
  for(n in 1:N) {
      target += lognormal_lpdf(RT[n] | mu, sigma) +
```

```
                          bernoulli_lpmf(nresp[n] | p2) ;
}
}
```

```
fit_mix_rt_null = stan(file = 'feature_migration_mixture_null.stan',
               data = feature_migrature_mixture_model,
               control = list(max_treedepth = 15,adapt_delta = .99999) )
```

Contrasting the null model against the feature migration model.

```
lml_mix <- bridge_sampler(fit_mix_rt, silent = TRUE)
lml_null_mix <- bridge_sampler(fit_mix_rt_null, silent = TRUE)
bayes_factor_attraction <- bridgesampling::bf(lml_mix, lml_null_mix)
bayes_factor_attraction
```

```
## Estimated Bayes factor in favor of lml_mix over lml_null_mix: 31473912.67864
```

The Bayes Factor value strongly supports the idea of a mixture model with a feature migration effect.

Problem 3: Model comparison

Once more, we employ Bayes factor to compare a mixture model that incorporates both feature migration and attractor condition with an accumulator model that relies solely on the attractor condition.

```
bayes_factor_attraction <- bridgesampling::bf(lml_mix, lml_attraction)
bayes_factor_attraction
```

```
## Estimated Bayes factor in favor of lml_mix over lml_attraction: 33590.55273
```

The evidence in favor of the mixture model appears to outweigh that of the accumulator model.

Problem 4: Multinomial processing tree

In the context of the MPT lecture on guessing and retrieval processes, two processing trees can be considered, one for condition a and another for condition b. This concept can also be applied to sentence processing.

Condition a:

The subject has the option to either guess with a probability of θ_g or perform a memory retrieval task with a probability of θ_r . The subject takes μ_1 time to decide whether to guess or not. If they choose to guess, it takes them μ_2 time to respond regarding the sentence's grammaticality. Alternatively, if the subject engages in the retrieval task, it takes them μ_3 time. Note: $\mu_2 < \mu_3$ indicates that the response time after guessing is faster than the response time after engaging in the retrieval task.

Condition b:

Similar to condition a, the subject can guess with a probability of θ_g or perform a memory retrieval task with a probability of θ_r. The subject takes μ_1 time to decide whether to guess or not. After guessing, it takes them μ_2 time to respond regarding the sentence's grammaticality. Alternatively, if the subject chooses the retrieval task, it takes them μ_4 time. It is important to note that the time taken to engage in the retrieval task differs between condition a and condition b. In condition b, the subject may experience feature

migration, leading to a larger (slower) response time. For simplicity, it can be assumed that the probability of retrieval is the same in both cases.

Note: $\mu_2 < \mu_4$
The priors for all the parameters mentioned above are defined as follows:

$$\begin{aligned}
\theta_g &\sim Beta(2, 2) \\
\theta_r &\sim Beta(2, 2) \\
\mu_1 &\sim Normal(7, 1) \\
\mu_3 &\sim Normal(4, 1) \\
\mu_2 &\sim Normal_{ub=\mu_4}(2, 1) \\
\mu_4 &\sim Normal_{ub=\mu_3}(3, 1) \\
\sigma &\sim Normal_{+}(.1, .05)
\end{aligned} \tag{6}$$

We initiate the process by conducting a prior predictive check.

Stan Model for "mpt.stan" :

```
data {
  int<lower=1> N_obs;
  real RT[N_obs];
  int<lower=0,upper=1> n_resp[N_obs];
  int<lower=0,upper=1> n_cond[N_obs];
}
parameters {
  real<lower=0.1,upper=1> theta_guess;
  real<lower=0.1,upper=1> theta_retr;
  real mu_1; // time to decide guess or not
  real mu_3; // time to retrieve if condition attractor
  real<upper=mu_3> mu_4; // time to retrieve if baseline condition
  real <upper=mu_4> mu_2; // time to guess

  real<lower = 0> sigma;
}
transformed parameters {
  simplex[2] theta[N_obs];
  simplex[2] theta_G[N_obs];
  simplex[2] theta_UG[N_obs];
  for(n in 1:N_obs){
    //Pr_Grammatical:
    theta[n, 1] = theta_guess*0.5 + (1-theta_guess)*theta_retr;
    //Pr_Ungrammatical:
    theta[n, 2] = theta_guess*0.5 + (1-theta_guess)*(1-theta_retr);
    //Pr_guess_when_grammatical
    theta_G[n, 1] = (theta_guess*0.5)/theta[n, 1];
    //Pr_retrieval_when_grammatical
    theta_G[n, 2] = ((1-theta_guess)*theta_retr)/theta[n, 1];
    //Pr_guess_when_ungrammatical
    theta_UG[n, 1] = (theta_guess*0.5)/theta[n, 2];
    //Pr_retrieval_when_ungrammatical
    theta_UG[n, 2] = ((1-theta_guess)*(1-theta_retr))/theta[n, 2];
  }
}
```

```
model {
  target += beta_lpdf(theta_guess | 2, 2);
  target += beta_lpdf(theta_retr | 2, 2);
  target += normal_lpdf(mu_1 | 7, 1);
  target += normal_lpdf(mu_3 | 4, 1);
  target += normal_lpdf(mu_4 | 3, 1)
    - normal_lcdf(mu_3 | 4, 1);
  target += normal_lpdf(mu_2 | 2, 1)
    - normal_lcdf(mu_4 | 3, 1);

  target += normal_lpdf(sigma | .1, .05)
    - normal_lccdf(0 | .1, .05);
  for(n in 1:N_obs)
  {
    if (n_cond[n] == 0) // no attraction effect
    {
         if (n_resp[n]==0) // if response is Ungrammatical
      {
        target += log(theta[n,2]) + log_sum_exp(log(theta_UG[n, 1])+
        lognormal_lpdf(RT[n] | mu_1 + mu_2, sigma), log(theta_UG[n, 2])+
        lognormal_lpdf(RT[n] | mu_1 + mu_4, sigma)) ;
      }
      else {
        target += log(theta[n,1]) + log_sum_exp(log(theta_G[n, 1])+
        lognormal_lpdf(RT[n] | mu_1 + mu_2, sigma), log(theta_G[n, 2])+
        lognormal_lpdf(RT[n] | mu_1 + mu_4, sigma)) ;
      }
    }
    else // attraction effect
    {
        if (n_resp[n]==0) // if response is Ungrammatical
      {
        target += log(theta[n,2]) + log_sum_exp(log(theta_UG[n, 1])+
        lognormal_lpdf(RT[n] | mu_1 + mu_2, sigma), log(theta_UG[n, 2])+
        lognormal_lpdf(RT[n] | mu_1 + mu_3, sigma)) ;
      }
      else {
        target += log(theta[n,1]) + log_sum_exp(log(theta_G[n, 1])+
        lognormal_lpdf(RT[n] | mu_1 + mu_2, sigma), log(theta_G[n, 2])+
        lognormal_lpdf(RT[n] | mu_1 + mu_3, sigma)) ;
      }
    }

  }
}
```

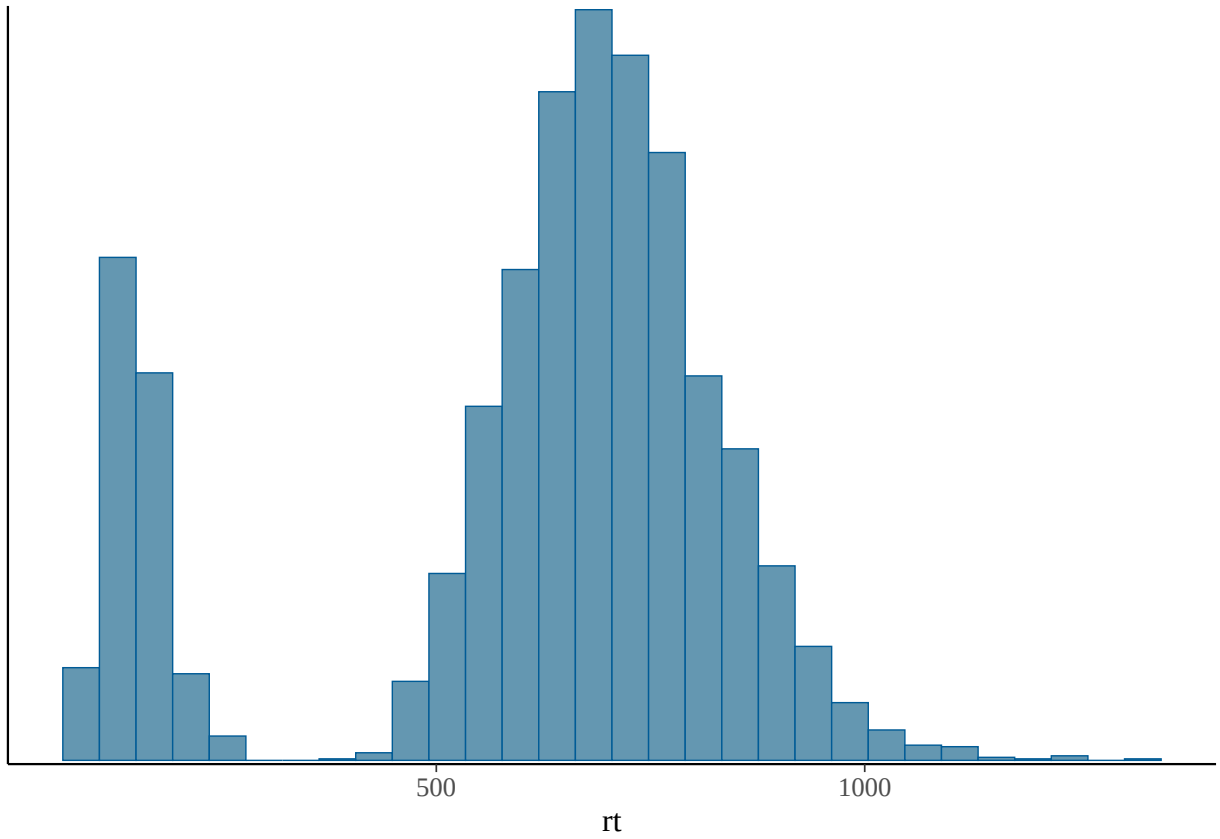
```
mpt_model_prior <- list(prioronly=1,
                    N_obs = nrow(data_new),
                    RT = data_new$response_time_ms,
                    n_resp = data_new$nresp,
                    n_cond = data_new$ncond)
prior_mpt = stan(file = 'mpt.stan',
               data = mpt_model_prior,
```

```
                control = list(max_treedepth = 15,adapt_delta = .99999) )
```

In this plot, we display the response time derived from the prior predictive model.

```
Probability_Grammatical = function(a,b) #Probability that response is Grammatical
  a*0.5 + (1-a)*b
Probability_Ungrammatical = function(a,b) #Probability that response is ungrammatical
  a*0.5 + (1-a)*(1-b)

P_Guess_G_UG = function(a,b) #Probability of Guess given response is un/grammatical
  (a*0.5)/b

P_Retr_G = function(a,b,c) #Probability of retrieval given response is grammatical
  ((1-a)*b)/c

P_Retr_UG = function(a,b,c) #Probability of retrieval given response is ungrammatical
 ((1-a)*(1-b))/c


theta_g = as.data.frame(prior_mpt)$theta_guess
theta_r = as.data.frame(prior_mpt)$theta_retr
mu_1 = as.data.frame(prior_mpt)$mu_1
mu_2 = as.data.frame(prior_mpt)$mu_2
mu_3 = as.data.frame(prior_mpt)$mu_3
mu_4 = as.data.frame(prior_mpt)$mu_4
sigma = as.data.frame(prior_mpt)$sigma

Theta = tibble(UG = Probability_Ungrammatical(theta_g, theta_r),
               G =  Probability_Grammatical(theta_g, theta_r),
               Guess_G= P_Guess_G_UG(theta_g,G),
               Guess_UG= P_Guess_G_UG(theta_g,UG),
               Retr_G= P_Retr_G(theta_g,theta_r,G),
               Retr_UG= P_Retr_UG(theta_g,theta_r,UG),
               )

N = 4000
nresp = rbern(N,Theta$G)
ncond = rbern(N)
rt = if_else(ncond == 1, ## attractor condtion
                if_else(nresp == 0, ## ungrammatical
                        Theta$UG*(Theta$Guess_UG * rlnorm(N,mu_1+mu_2,sigma) + Theta$Retr_UG *       rl
                        Theta$G*(Theta$Guess_G * rlnorm(N,mu_1+mu_2,sigma) + Theta$Retr_G *       rlno
                        ),
                if_else(nresp == 0, ## ungrammatical
                        Theta$UG*(Theta$Guess_UG * rlnorm(N,mu_1+mu_2,sigma) + Theta$Retr_UG *       rl
                        Theta$G*(Theta$Guess_G * rlnorm(N,mu_1+mu_2,sigma) + Theta$Retr_G *       rlno
                        )
                )
rt_estimates = data.frame(rt)
mcmc_hist(rt_estimates)
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



Here,we are applying the model's fitting process.

```
#Fit data
mpt_model <- list(N_obs = nrow(data_new),
                  RT = data_new$response_time_ms,
                  n_resp = data_new$nresp,
                  n_cond = data_new$ncond)
fit_mpt = stan(file = 'mpt.stan',
               data = mpt_model,
               control = list(max_treedepth = 15,adapt_delta = .99999) )
```

Since the mixture model outperformed the accumulator model, we solely compare the MPT model with the mixture model.

```
lml_mpt <- bridge_sampler(fit_mpt, silent = TRUE)
bayes_factor_attraction <- bridgesampling::bf(lml_mix, lml_mpt)
bayes_factor_attraction
```

```
## Estimated Bayes factor in favor of lml_mix over lml_mpt: 172306657.01243
```

Since the Bayes factor for the mentioned combination is considerably high, it indicates that the mixture model surpasses the MPT model. One potential explanation for this could be that the MPT model is either inadequately defined or its priors are not appropriately specified, resulting in poorer performance compared to the mixture model.