Clausal Inference

Michael Betancourt

January 2025

Table of contents

How do we parse, and ultimately understand, language? One way of studying comprehension of language is to measure how long it takes for individuals to read different types of writing. Distinct language patterns that are particularly easy, or difficult, to parse will manifest as systematic differences in those reading times. In this case study we'll develop a Bayesian analysis of reading time data using narratively generative modeling.

Our goal here is not to construct estimators of particular quantities or make decisions about different quantities being the same. Rather our objective is to model the entire data generating process, from the latent cognitive behaviors of interest to the final, processed observations. Given that model we can then use Bayes' Theorem to directly infer which cognitive behaviors are most consistent with observed data.

This case study will assume familiarity with probabilistic modeling and Bayesian inference, narratively generative modeling, model evaluation, and Stan. If your experience is limited then this analysis will likely be a bit overwhelming, *and that is absolutely okay.* In that case I recommend treating this case study more as a demonstration of what is possible given more experience with these tools and, ideally, a motivation to study them.

1 Modeling Reading Times

As with most scientific endeavors, designing and modeling reading time experiments requires care. For example, some individuals in in experimental cohort might read faster than others. If faster readers are given one text and slower readers another, then the differences we observe may be due to not difference in the texts themselves but rather these differences in the individuals. Separating individual pace, and other sources of heterogeneity, from language complexity requires a careful analysis.

1.1 Pairwise Comparison Models

One particularly productive way to account for the complexities in an experiment is to directly model the entire data generating process. While this might appear to be an overwhelming task, it becomes much more manageable if we start as simply as possible and then expand as necessary.

How should we model reading times? Let's assume that each observation is the result of one reader parsing a short piece of text, such as a word or phrase, from a longer piece of text, such as a sentence or paragraph. I will refer to this short piece as the **target text**.

The more complex the target text is in the context of the longer text, the longer we would expect reading times to be. At the same time, a more experienced reader would typically be able to achieve shorter reading times than a less experienced reader.

We can model this contrast between an individual's reading skill and the complexity of the assigned text with a bipartite pairwise comparison process. Those unfamiliar with pairwise comparison modeling more generally may have encountered the specific application of this technique in **item response theory**.

A pairwise comparison model begins with an appropriate probabilistic location model for the reading times,

$$p(t \mid \mu, \phi).$$

Here the location parameter μ configures the *centrality* of the reading times while ϕ configures any other behaviors, such as how the model concentrates around that central value. Some common models that are compatible with positive, real-valued reading times include the log normal model, the gamma model, and the inverse gamma model. If you're curious about these models then take a look at this chapter.

Once we have chosen a location model, we replace the location parameter μ by the output of a deterministic function,

$$\mu_{ij} = f(\alpha_i - \beta_j).$$

The parameter α_i quantifies the complexity of the ith target text, the parameter β_j quantifies the experience of jth reader, and the coupling function f determines exactly how the difference

in these parameters moderates μ. Different coupling functions can be more or less useful in different applications.

To demonstrate this construction let's start with the log normal reading time model,

$$\text{log-normal}(t \mid \log(\mu), \phi),$$

with the location parameter μ and scale parameter ϕ. We can then elevate the log normal model to a pairwise comparison model by introducing a coupling function,

$$\begin{aligned} \mu_{ij} &= f(\alpha_i - \beta_j) \\ &= \exp\left(\eta + \alpha_i - \beta_j\right) \\ &= \exp\left(\eta\right) \exp\left(\alpha_i\right) \frac{1}{\exp\left(\beta_j\right)}. \end{aligned}$$

Conveniently, this coupling function factors into three multiplicative terms. The leading term $\exp(\eta)$ defines a baseline reading time behavior which the complexity and experience parameters then scale up and down depending on their sign. Given a particular reader, the more complex a target text is the larger the difference

$$\alpha_i - \beta_j$$

will be. As the difference increases, the location μ_{ij} scales up from the baseline and the reading time model concentrates on longer reading times.

1.2 Capturing Systematic Heterogeneity

The goal of many linguistic experiments is to infer not the behavior of individual texts but rather any behaviors that are *shared* across groups of texts, and the relationships between those shared behaviors. For example, one common objective is to understand how different language patterns influence reader comprehension regardless of particular word choice.

Shared behaviors manifest as *systematic* differences in the observed reading times. These differences, however, take different forms depending on the assumed data generating process. Because our interaction with language is so complex, the possibilities here can be overwhelming.

To make the discussion more manageable we'll have to narrow our scope to a specific language pattern. Then we can consider how variants of that pattern might result in heterogeneous reading time behaviors.

1.2.1 Relative Clauses

Consider a simple sentence with a subject, a verb, and an object, such as

> "The scientist built a model."

This sentence becomes more complex with the introduction of a **relative clause** that endows the subject with additional information. For example we might write

> "The scientist who ignored the statistician built a model."

or

> "The scientist whom the statistician ignored built model."

In the first sentence the subject of the sentence is also the subject of the relative clause, making the relative clause **subject-extracted**. Because the subject of the second sentence is the *object* of the relative clause, however, the relative clause is **object-extracted**.

For more discussion of these different types of relative clauses see Gibson and Wu (2013), Vasishth et al. (2017), Nicenboim and Vasishth (2018), and Nicenboim, Schad, and Vasishth (2025).

We should be able to learn the relative difficulty in parsing these two types of relative clauses by first engineering sentences that are the same up to the behavior of the relative clause and then examining any difference in the subsequent reading time behaviors. That said, even in this narrow scope there are a myriad of ways in which the reading times might behave differently.

Here we will consider two classes of models discussed in Vasishth et al. (2017).

1.2.2 Dependency Locality Model

The dependency locality model hypothesizes that the difficulty in parsing a sentence is influenced by the distance between linguistically related elements. Sentences in which related elements are separated by unrelated elements are predicted to take longer to read than sentences where the related elements appear next to each other.

If subject-extracted and object-extracted relative clauses exhibit different localities then we would expect sentences with one to be systematically more difficult to parse than sentences with the other. Interestingly the type of relative clause that should be easier to parse is not universal, but rather specific to particular languages.

We can incorporate dependency locality into our base reading time model by introducing sentence complexity parameters for each type of relative clause, α_i^{SR} and α_i^{OR}. If

$$\alpha_i^{\mathrm{SR}} > \alpha_i^{\mathrm{OR}}$$

in a particular language, then subject-extracted relative clauses will be more difficult to parse than object-extracted relative clauses.

When the influence of the relative clause structure is independent of the rest of the sentence, we can expand these parameters into text-specific baselines α_i that are modified by relative clause parameters,

$$\begin{aligned} \alpha_i^{\text{SR}} &= \alpha_i + \gamma^{\text{SR}} \\ \alpha_i^{\text{OR}} &= \alpha_i + \gamma^{\text{OR}}. \end{aligned}$$

In this case the relative clause parameters γ^{SR} and γ^{OR} scale the location of the base reading time model,

$$\begin{aligned} \mu_{ij}^{\text{SR}} &= f(\alpha_i^{\text{SR}} - \beta_j) \\ &= \exp\left(\eta + \alpha_i^{\text{SR}} - \beta_j\right) \\ &= \exp\left(\gamma^{\text{SR}} + \eta + \alpha_i - \beta_j\right) \\ &= \exp\left(\gamma^{\text{SR}}\right) \exp\left(\eta + \alpha_i - \beta_j\right) \\ &= \exp\left(\gamma^{\text{SR}}\right) \mu_{ij} \end{aligned}$$

and

$$\begin{aligned} \mu_{ij}^{\text{OR}} &= f(\alpha_i^{\text{OR}} - \beta_j) \\ &= \exp\left(\gamma^{\text{OR}} + \eta + \alpha_i - \beta_j\right) \\ &= \exp\left(\gamma^{\text{OR}}\right) \exp\left(\eta + \alpha_i - \beta_j\right) \\ &= \exp\left(\gamma^{\text{OR}}\right) \mu_{ij} \end{aligned}$$

1.2.3 Direct-Access Model

The direct-access model (Nicenboim and Vasishth 2018) posits that reading comprehension is more of a trial and error process. In this model readers parse text by forming an initial hypothesis for the relationships between all of its elements. If these hypothesized relationships are inconsistent, then the reader needs to spend time to correct it. Text can take longer to parse not only because forming the initial hypothesis might take longer but also because that initial hypothesis might be less likely to be correct.

We can model the time it takes for a reader to form an initial hypothesis using the same pairwise comparison modeling techniques that we've already considered,

$$p_1(t_{ij} \mid \mu_{1,ij}, \phi_1)$$

with

$$\mu_{1,ij} = \exp\left(\eta + \alpha_i - \beta_j\right).$$

Again more complex text slows the time of initial hypothesis formation, while increased reader experience accelerates it.

Correcting an erroneous initial hypothesis takes more time. Here we'll assume that the time to fix an initial hypothesis is proportional to the time taken to form it in the first place,

$$p_2(t_{ij} \mid \mu_{2,ij}, \phi_2)$$

with

$$\mu_{2,ij} = \exp(\omega)\, \mu_{1,ij} = \exp\left(\eta + \omega + \alpha_i - \beta_j\right).$$

To ensure that an initial failure results in longer reading times we'll need to restrict ω to be positive.

When we observe only reading times, however, we will not know if a reader's initial hypothesis was actually successful or not. Consequently neither of these models is sufficient on their own!

To quantify our uncertainty about the success of the initial hypothesis we have to combine both of these models into a single mixture model,

$$\lambda\, p_1(t_{ij} \mid \mu_{1,ij}, \phi_1) + (1 - \lambda)\, p_2(t_{ij} \mid \mu_{2,ij}, \phi_2),$$

where λ quantifies the probability that the initial hypothesis is correct. The more challenging a text is to parse correctly, the smaller λ will be and the more weight will be given to the longer reading times of the second component model.

In general λ could vary across readers and texts. Here, however, we will assume that that any systematic heterogeneity in λ is entirely due to the structure of the relative clause,

$$\lambda_{\text{SR}}\, p_1(t_{ij} \mid \mu_{1,ij}, \phi_1) + (1 - \lambda_{\text{SR}})\, p_2(t_{ij} \mid \mu_{2,ij}, \phi_2)$$

and

$$\lambda_{\text{OR}}\, p_1(t_{ij} \mid \mu_{1,ij}, \phi_1) + (1 - \lambda_{\text{OR}})\, p_2(t_{ij} \mid \mu_{2,ij}, \phi_2).$$

If

$$\lambda_{\text{SR}} > \lambda_{\text{OR}}$$

then adding a subject-extracted relative clause to a sentence makes it more difficult to parse correctly than adding an equivalent object-extracted relative clause.

1.2.4 ¿Por Qué No Los Dos?

One of the beautiful features of probabilistic modeling is that we can often combine separate models together into a consistent *joint model*. This allows us to, for example, directly infer which behaviors from each model are more or less consistent with any observed data.

The dependency locality and direct-access models are not mutually exclusive; indeed integrating them into a consistent joint model is relatively straightforward. All we need to do is introduce the dependency locality model's scaling of the locality parameter to our direct-access mixture model.

For passages with subject-extracted relative clauses this gives

$$\lambda_{\text{SR}}\, p_1(t_{ij} \mid \exp\left(\gamma^{\text{SR}}\right)\, \mu_{1,ij}, \phi_1) + (1 - \lambda_{\text{SR}})\, p_2(t_{ij} \mid \exp\left(\gamma^{\text{SR}}\right)\, \mu_{2,ij}, \phi_2).$$

Equivalently for object-extracted relative clauses we have

$$\lambda_{\text{OR}}\, p_1(t_{ij} \mid \exp\left(\gamma^{\text{OR}}\right)\, \mu_{1,ij}, \phi_1) + (1 - \lambda_{\text{OR}})\, p_2(t_{ij} \mid \exp\left(\gamma^{\text{OR}}\right)\, \mu_{2,ij}, \phi_2).$$

Note that we completely recover the dependency locality model when

$$\lambda_{SR} = \lambda_{OR} = 1.$$

Similarly we recover the direct-access model when

$$\gamma^{\text{SR}} = \gamma^{\text{OR}}.$$

In other words both of the proposed models are *special cases* of this larger joint model.

We can compare how useful these pieces are for explaining any observed data by studying how much our inferences for λ_{SR}, λ_{OR}, γ^{SR}, and γ^{OR} deviate from these special cases. For example if the observed data is consistent with

$$\gamma^{\text{SR}} = \gamma^{\text{OR}}$$

then the role of dependency locality will be negligible.

2 Environment Setup

To prepare for implementing these models let's first set up our local `R` environment.

```
par(family="serif", las=1, bty="l",
    cex.axis=1, cex.lab=1, cex.main=1,
    xaxs="i", yaxs="i", mar = c(5, 5, 3, 1))

library(rstan)
rstan_options(auto_write = TRUE)            # Cache compiled Stan programs
options(mc.cores = parallel::detectCores()) # Parallelize chains
parallel:::setDefaultClusterOptions(setup_strategy = "sequential")
```

To facilitate the implementation of the Bayesian analysis we'll take advantage of my recommended diagnostic and visualization tools.

```
util <- new.env()
```

First we have a suite of Markov chain Monte Carlo diagnostics and estimation tools; this code and supporting documentation are both available on GitHub.

```
source('mcmc_analysis_tools_rstan.R', local=util)
```

Second we have a suite of probabilistic visualization functions based on Markov chain Monte Carlo estimation. Again the code and supporting documentation are available on GitHub.

```
source('mcmc_visualization_tools.R', local=util)
```

3 Data Exploration

A model is only speculation until we can study its interplay with data. For this case study we will analyze data from Gibson and Wu (2013). These data were collected by measuring how quickly readers could parse Chinese sentences with subject-extracted and object-extracted relative clauses.

3.1 The Experimental Design

Gibson and Wu designed sixteen **items**, each of which defined a writing template consisting of two sentences. The first sentence in each item established context, while the last sentence introduced space for a relative clause. Individual written passages were then constructed by taking an item, selecting a noun to be the subject of the relative clause, and then finally choosing whether the relative clause would subject-extracted or object-extracted. The remaining

word choice was carefully engineered to minimize interactions with the subject of the relative clause.

Forty native Chinese speakers, or **subjects**, each read some of these constructed passages on computers. To measure the pace of each subject's reading, the passages were not initially presented in their entirety. Instead each subject had to press a button to reveal fragments sequentially, with the time between presses quantifying the time taken to parse each fragment. The subject of the relative clause was always revealed by itself, and the final reading times were given by the time between the proceeding and subsequent button presses.

This procedure was implemented using the `Linger` software package.

To verify that each subject correctly understood the structure of the relative clause in each passage, the trials concluded with a binary question testing the subject's comprehension. Gibson and Wu (2013) quotes a comprehension accuracy of about 90% for both the subject-extracted and object-extracted passages, indicating that in both cases the subjects were usually taking the time to parse what they were reading.

Gibson graciously shared the data from that experiment in a text file formatted with space-separated values.

```
raw_data <- read.csv('data/gibsonwu2012data.txt',
                     sep="", header=TRUE)

data <- list('N' = nrow(raw_data),
             'N_items' = max(raw_data$item),
             'item' = raw_data$item,
             'N_subjects' = max(raw_data$subj),
             'subject' = raw_data$subj,
             'reading_time' = raw_data$rt,
             'subj_rel' = raw_data$type == 'subj-ext')
```

Altogether the data includes 2735 observations. Consistent with the experiment design in **ref**, these observations spanned 16 items and 40 subjects, both of which are indexed sequentially.

```
cat(sprintf('%i observations\n', data$N))
```

```
2735 observations
```

```
cat(sprintf('%i items\n', data$N_items))
```

```
16 items
```

```
cat(sprintf('%i subjects\n', data$N_subjects))
```

```
40 subjects
```

Not every subject in the experiment was given every constructed passage to read. Instead pairings were determined through a Latin square design (Box, Hunter, and Hunter 2005) which, if implemented correctly, would have ensured good coverage of the possible subject-item pairings.

That said, we don't analyze how an experiment *was supposed* to be implemented. We analyze how it how it *was actually* implemented. Here that requires carefully investigating the observed pairings.

One nice way to quantify the observed pairings is with a little bit of graph theory; for more on these techniques see Sections 4.3.2 and 6.2 of my pairwise comparison modeling chapter. Specifically, we can encode the observations in a graph where the items and subjects are nodes and each observed pairing is an edge. The properties of this graph then allow us to study the properties of the observed pairings.

For example, the connected components of a graph determine which subsets of items and subjects have been directly or indirectly compared to each other. Items and subjects in different connected components cannot be related to each other.

In conflict with the experimental design not every item has been paired with at least one subject, and not every subject has been paired with at least one item. Item 12 and Subjects 10, 13, and 25 all appear as their own isolated components in the graph.

```
source('graph_utility.R', local=util)

adj <- util$build_adj_matrix(data$N_items + data$N_subjects,
                             data$N,
                             data$item,
                             data$N_items + data$subject)

components <- util$compute_connected_components(adj)
length(components)
```

```
[1] 5
```

```
for (k in seq_along(components)) {
  print(paste('Component', k))
  cat('  ')
  cat(sapply(components[[k]],
             function(i)
               ifelse(i > data$N_items,
                      paste0('Subject ', i - data$N_items, ','),
                      paste0('Item ', i, ','))))
  cat('\n')
  print('')
}
```

```
[1] "Component 1"
  Item 1, Subject 1, Item 2, Subject 2, Item 3, Subject 3, Item 4, Subject 4, Item 5, Subject
[1] ""
[1] "Component 2"
  Item 12,
[1] ""
[1] "Component 3"
  Subject 10,
[1] ""
[1] "Component 4"
  Subject 13,
[1] ""
[1] "Component 5"
  Subject 25,
[1] ""
```

Indeed this item and these subjects have not been involved in any observations at all. Item 12 was not read by any subjects, and Subjects 10, 13, and 25 did not read any items.

```
for (k in seq_along(components)) {
  if (length(components[[k]]) > 1) next

  for (i in components[[k]]) {
    if (i > data$N_items) {
      N_obs <- sum(data$subject == (i - data$N_items))
      print(paste('Subject', i %% data$N_items))
    } else {
      N_obs <- sum(data$item == i)
      print(paste('Item', i))
```

```
    }
    print(paste('  N_obs =', N_obs))
    print('')
  }
}
```

```
[1] "Item 12"
[1] "  N_obs = 0"
[1] ""
[1] "Subject 10"
[1] "  N_obs = 0"
[1] ""
[1] "Subject 13"
[1] "  N_obs = 0"
[1] ""
[1] "Subject 9"
[1] "  N_obs = 0"
[1] ""
```

To be fair these gaps are documented in Gibson and Wu (2013). Item 12 was not sampled by the experimental software due to a coding error. Moreover, the three subjects not included here were intentionally excluded due to exceptionally low sentence comprehension accuracy.

Had the data from all of the subjects been included, then we would have had the opportunity to analyze sentence comprehension jointly with reading times. For now, however, the best we can do is acknowledge this processing of the data and analyze the observations that were included.

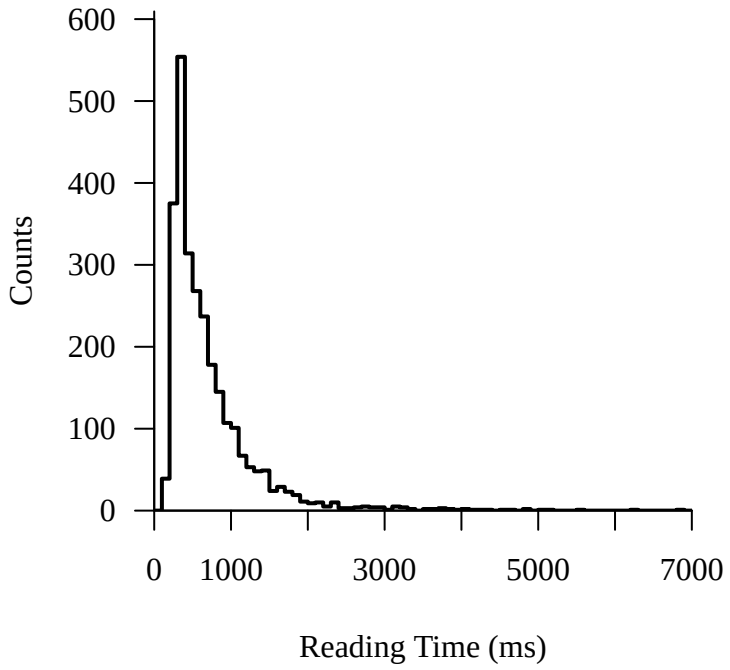
3.2 Summarizing Reading Times

With a thorough understanding of the item/subject pairings, we can finally consider the resulting reading times. Because we cannot productively visualize the data in its entirely, we will have to engineer interpretable **summary statistics** that can be visualized.

My personal favorite summary statistic is the lowly histogram. Across all observations we see a rapid rise in instances of small reading times and then a slower decay as we move up to larger reading times.

```
par(mfrow=c(1, 1), mar = c(5, 5, 2, 1))

util$plot_line_hist(data$reading_time, 0, 7000, 100,
                    xlab='Reading Time (ms)')
```

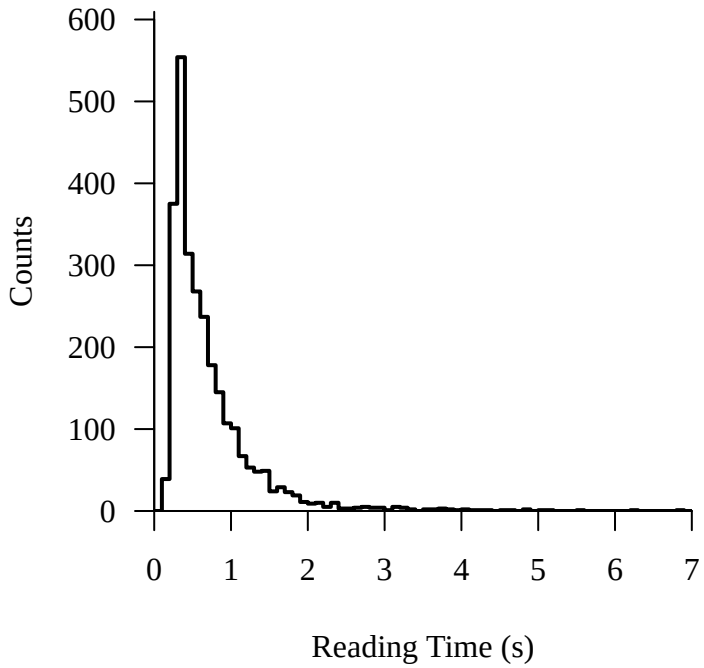
```
Warning in check_bin_containment(bin_min, bin_max, values): 1 value (0.0%) fell
above the binning.
```



Sometimes different choices of units can better connect a summary to our available domain expertise. By moving from milliseconds to seconds, for example, we see that most readers can parse the target noun in under a second. Sometimes, however, readers take over five seconds!

```
par(mfrow=c(1, 1), mar = c(5, 5, 2, 1))

util$plot_line_hist(data$reading_time * 1e-3, 0, 7, 0.1,
                    xlab='Reading Time (s)')
```

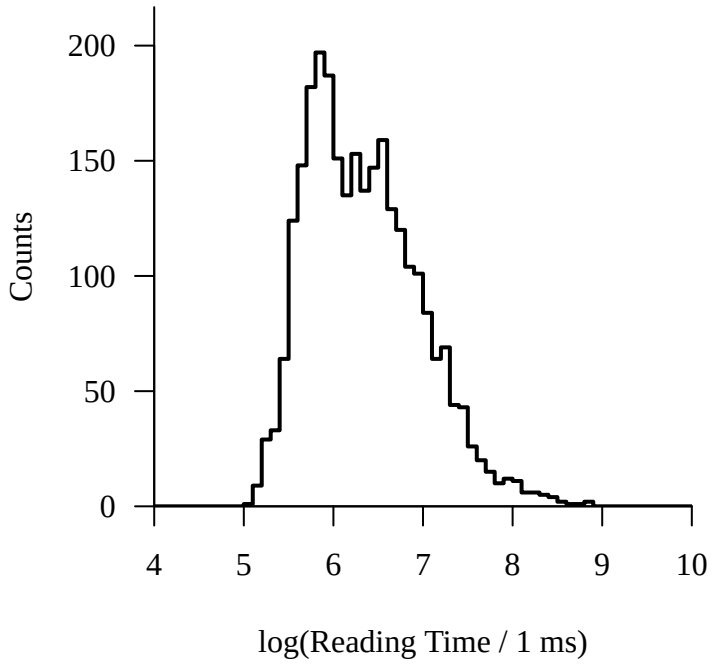
```
Warning in check_bin_containment(bin_min, bin_max, values): 1 value (0.0%) fell
above the binning.
```



Because of this heavy tailed behavior, it can be difficult to perceive all the relevant features of the reading times at the same time. One way to make the observed reading times a bit more compact is to histogram their *logarithms*. Note that we can do this here only because there are no vanishing reading times; `log(1 + epsilon)` heuristics are *never* a good idea. Yes, even for you.

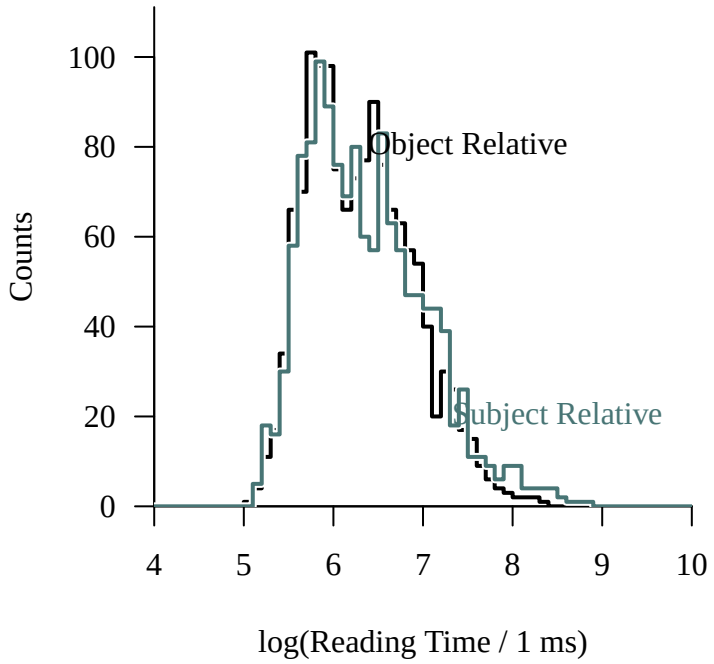
Interestingly, the histogram of the log reading times indicate that there might be some multimodal behavior in the observations. That said, this is only a speculation because of the inherent variability in the measurements.

```
par(mfrow=c(1, 1), mar = c(5, 5, 2, 1))

util$plot_line_hist(log(data$reading_time), 4, 10, 0.1,
                    xlab='log(Reading Time / 1 ms)')
```



Summarizing all of the data in aggregate doesn't tell the entire story here. Remember that the scientific question of interest concerns the *difference* in reading time behavior across the two types of relative clause. Fortunately, we can readily isolate those behaviors with two separate histograms.

```
par(mfrow=c(1, 1), mar = c(5, 5, 2, 1))

util$plot_line_hists(log(data$reading_time[data$subj_rel == FALSE]),
                     log(data$reading_time[data$subj_rel == TRUE]),
                     4, 10, 0.1,
                     xlab='log(Reading Time / 1 ms)')

text(7.5, 80, labels='Object Relative', col="black")
text(8.5, 20, labels='Subject Relative', col=util$c_mid_teal)
```



The peaks of both histograms look similar, with the reading times of subject-extracted relative clauses exhibiting a slightly heavier tail. Note that the similarity of the two histograms doesn't necessarily say anything about the differences in complexity for subject-extracted and objected-extracted relative clauses. For example, if the assignment of passages with the two types of clauses is not balanced across readers of similar experience then the differences in reader behavior could wash out differences in the text behavior.

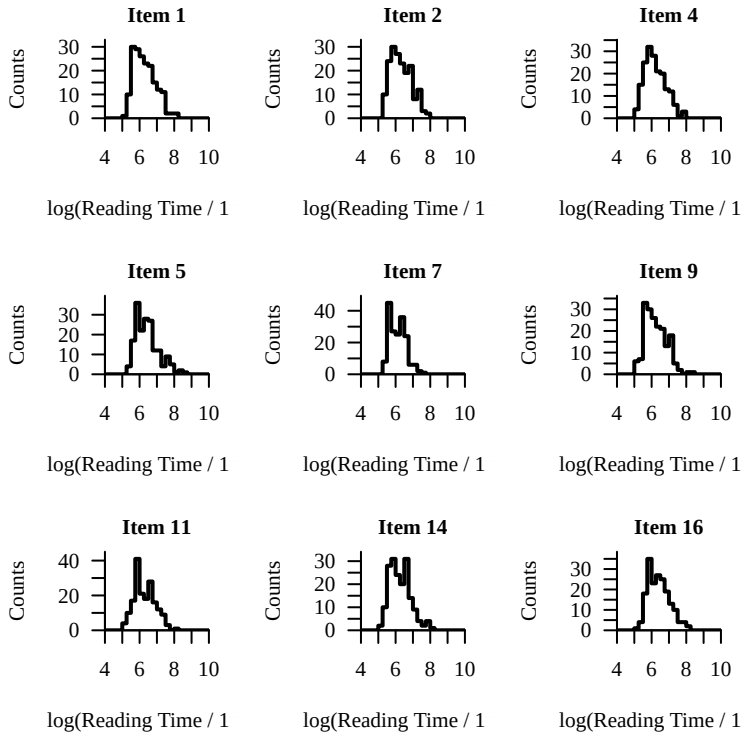
We also see that the multimodal structure persists in both of these histograms. This behavior will be useful to keep in mind as we develop our models.

The type of relative clause is not the only way that we can categorize the observations. We can also examine any systematic heterogeneity in reading time behavior across items and subjects by plotting log reading time histograms for each group. This is also known as **stratifying** the histograms.

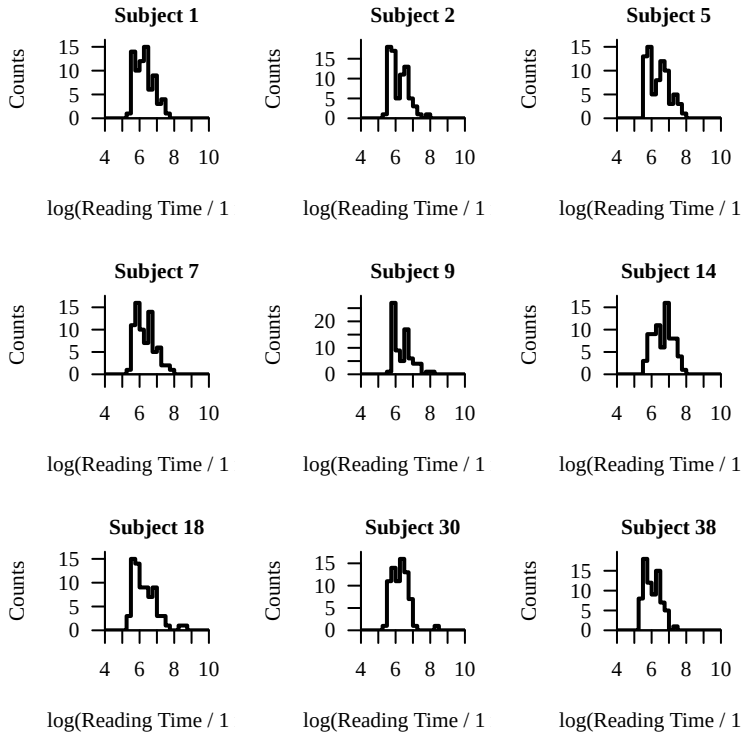
To make the visualization a bit more manageable, I will show the stratified histograms for only nine items and nine subjects.

```
set.seed(483833992)

display_items <- sort(sample(1:data$N_items, size=9))

par(mfrow=c(3, 3), mar = c(5, 5, 2, 1))

for (i in display_items) {
```

```
  reading_times <- data$reading_time[data$item == i]
  util$plot_line_hist(log(reading_times), 4, 10, 0.25,
                      xlab='log(Reading Time / 1 ms)',
                      main=paste('Item', i))
}
```




```
display_subjects <- sort(sample(1:data$N_subjects, size=9))

par(mfrow=c(3, 3), mar = c(5, 5, 2, 1))

for (s in display_subjects) {
  reading_times <- data$reading_time[data$subject == s]
  util$plot_line_hist(log(reading_times), 4, 10, 0.25,
                      xlab='log(Reading Time / 1 ms)',
                      main=paste('Subject', s))
}
```



While informative, these ensemble visualizations can be a bit overwhelming to process when there are a large number of elements. In order to scale to larger groups we would have to consider alternative summaries.

For example we might reduce the reading times in each group to a *scalar* summary, such as an empirical mean or variance, and then histogram all of the scalar summaries into a single, compact visualization. If you are curious about how this might be implemented in practice, then take a look at this case study where I analyze movie reviews.

3.3 Discretization

A well-constructed summary statistic isolates interpretable features of the entire data set. That interpretability ensures that we can compare and contrast the observed behavior to any available domain expertise. Because domain expertise varies from person to person, however, so too will the insights we can pull from any visualization.

Those familiar with histograms of continuous data might have already noticed a "spikeness" in the stratified histograms. As a crude rule of thumb, the expected variation in each bin can be approximated by the square root of the observed counts. Consequently if we're looking at data drawn from a continuous model we would expect neighboring bin counts to be similar up to their square roots.

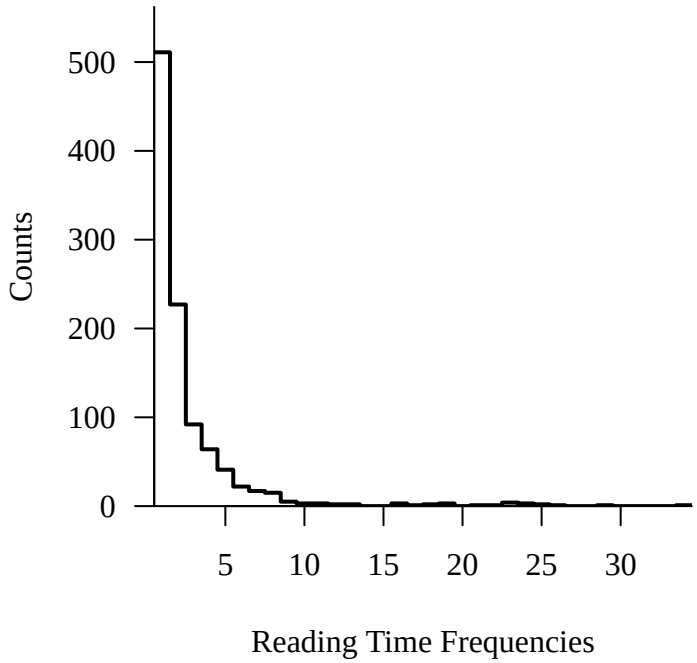
Here, however, there are a decent number of isolated bins whose counts spike up above their neighboring. At this point my concern is entirely speculative. That said, it does motivate me to dig into the data a little bit more carefully.

Observed reading times can, in theory, take any positive real value. In practice we are limited to computers and floating point numbers. Unless our observations are extraordinarily precise, however, floating numbers should well-approximate exact real numbers.

For example, repetitions of particular values is mathematically improbable when working with exactly real-valued observations. Repetitions of particular floating point values is not impossible, but they will be rare if the data generating process is continuous.

This is not, however, the behavior we see in our observed reading times. Most observed values repeat at least once, and some repeat as many as thirty times!

```
par(mfrow=c(1, 1), mar = c(5, 5, 2, 1))

util$plot_line_hist(table(data$reading_time),
                    0.5, 34.5, 1,
                    xlab='Reading Time Frequencies')
```



Looking at the most frequent values reveals an even more striking behavior: the observed reading times are all integer multiples of 1 millisecond.

```
t <- table(data$reading_time)
t[t > 5]
```

```
 189  222  229  230  238  243  245  246  253  254  262  270  277  278  285  286
   7    6    7    9   12    6    9   16    8   16   18   19    7   19    8   23
 287  293  294  298  301  302  309  310  317  318  326  333  334  340  342  344
   6    6   21    8    7   23    6   19    6   34   25    6   24    7   29    6
 349  350  357  358  366  373  374  382  390  398  406  414  422  428  430  438
   9   25    7   23   26   10   24   24   23   13   22   17   13    6   16   18
 446  454  462  470  478  483  484  486  502  508  510  515  524  526  531  532
  10   11    7   12    6    7    6    7   10    8    7    7    8    8    8    6
 534  550  566  582  588  596  603  628  652  660  668  676  684  691  700  708
   8    6    7    8    9    6    6    6    7    8   11    9    6    6    8   11
 716  731  756  780  788  812  820  828  860  876 1028 1044
   6    6    8    7    8    8    7    7    8    6    7    6
```

Indeed *all* of the observed reading times are integer multiples of 1 millisecond.

```
table(data$reading_time %% 1)
```

```
   0
2735
```

In other words, the observed reading times are not as continuous as we had assumed. Instead they are fundamentally *discrete*. The details of this discretization, however, are not immediately clear. How we proceed depends on the available domain expertise.

Fortunately the computer program used to measure the reading times in this experiment is open source. If one were experienced with computer programming then they might look into the `Linger` source code to see that it uses the built-in `Tcl` function `clock` to approximate the time of each button press.

The `clock` function uses CPU cycles to track intervals of time, assuming that each cycle is approximately uniform in time. This isn't always a great approximation, for example excess computation can slow the cycles and introduce awkward timing artifacts, but for now let's assume that it is. While the exact computer hardware used in the experiments was not documented, personal computers of the time typically featured gigahertz processor speeds; at these speeds the resolution of the `clock` function should be closer to microseconds than milliseconds.

Consequently the millisecond discretization is more likely implemented in the code itself, either as truncation or rounding when floating point values are converted to integer values. Because `Tcl` is dynamically typed, however, it can be tricky to track down exactly where in the code this conversion happens, let alone determine whether the reading times are rounded to the nearest integer or truncated to the next lowest or highest integer.

At this point my domain expertise has been exhausted. In order to understand the discretization further I would reach out to collaborators who are more familiar with `Tcl`, if not the `Linger` code itself.

I appreciate that all of this might come across as irrelevant detail. If we can engineer a continuous model that adequately captures the relevant features of the observed data then it will be. On the other hand, if we cannot engineer an adequate continuous model then we will have to incorporate this discretization into our models in order to extract meaningful insights.

In Section 4.6 I demonstrate a few modeling technique that directly account for discretization.

4 Model Development

We have reviewed the relevant theory and carefully explored the observed data. Now we are finally ready to start building some probabilistic models of the data generating process, and then use Bayesian inference to learn what data generating behaviors are consistent with the observed data.

To make model development as manageable as possible we will work iteratively, starting with a simple model and then looking for any inadequacies. If we can learn not only that a model is inadequate but also *why* it is inadequate, then we can productively improve the model. Iterating this process of critique and improvement then allows us to quickly develop a model sophisticated enough to capture all of the relevant features of the observed data.

4.1 Log-Normal Dependency Locality Model

Let's begin with a log-normal dependency locality model for the observed reading times. While this model is relatively straightforward, it still needs a bit of work before we can implement it effectively.

4.1.1 Observational Model

The construction of the observational model begins with a log-normal model for the reading times

$$\text{log-normal}(t \mid \log(\mu), \phi).$$

We will then allow the location parameter to vary depending on the particular passage and reader being paired together.

When subject j has been assigned a passage derived from item i with a subject-extracted relative clause we use the location model

$$\begin{aligned}
\mu_{ij}^{\text{SR}} &= f(\alpha_i^{\text{SR}} - \beta_j) \\
&= \exp\left(\eta + \alpha_i^{\text{SR}} - \beta_j\right) \\
&= \exp\left(\eta + \gamma^{\text{SR}} + \alpha_i - \beta_j\right) \\
&= \exp\left(\gamma^{\text{SR}}\right) \exp\left(\eta + \alpha_i - \beta_j\right) \\
&= \exp\left(\gamma^{\text{SR}}\right) \mu_{ij}
\end{aligned}$$

Similarly for a passage derived from item i but with an object-extracted relative clause we use

$$\begin{aligned}
\mu_{ij}^{\text{OR}} &= f(\alpha_i^{\text{OR}} - \beta_j) \\
&= \exp\left(\eta + \gamma^{\text{OR}} + \alpha_i - \beta_j\right) \\
&= \exp\left(\gamma^{\text{OR}}\right) \exp\left(\eta + \alpha_i - \beta_j\right) \\
&= \exp\left(\gamma^{\text{OR}}\right) \mu_{ij}
\end{aligned}$$

4.1.2 Anchors

One immediate issue with this model is it is inherently **redundant**. There are many distinct configurations of η, α_i, β_j, γ^{SR}, and γ^{OR} that give exactly the same location configuration *for all observations*.

Redundant models always lead to redundant inferences. If one model configuration is reasonably consistent with the observed data then so too will *all* of the other equivalent model configurations. Because of this inferential consequently model redundancy is also known as statistical **non-identifiability**.

In order to quantify inferential uncertainties from a redundant model we will have to consider all of these equivalent model configurations. This is expensive at best and impossible at worst. For much more on identifiability see my chapter on the topic.

Fortunately we can completely eliminate the redundancy in this model with the careful use of **anchors**. To anchor the model we select an arbitrary anchor item, i', and then consider item complexities only *relative* to the complexity of that item,

$$\delta_i = \alpha_i - \alpha_{i'}.$$

If δ_i is greater than zero then the ith item is more complex than the anchor item. By construction the relatively complexity of the anchor item is fixed, or anchored, to zero,

$$\delta_{i'} = \alpha_{i'} - \alpha_{i'} = 0.$$

This eliminates one degree of freedom from the model, reducing its overall redundancy.

In addition to anchoring the item complexities we can also anchor the subject reading skills,

$$\zeta_j = \beta_j - \beta_{j'}.$$

Together these anchors allow us to write the location model as

$$\begin{aligned}
\mu_{ij}^{\mathrm{SR}} &= \exp\left(\eta + \gamma^{\mathrm{SR}} + \alpha_i - \beta_j\right) \\
&= \exp\left(\eta + \gamma^{\mathrm{SR}} + \alpha_{i'} - \beta_{j'} + \delta_i - \zeta_j\right) \\
&= \exp\left(\kappa + \delta_i - \zeta_j\right),
\end{aligned}$$

where

$$\kappa = \eta + \gamma^{\mathrm{SR}} + \alpha_{i'} - \beta_{j'}$$

quantifies the reading time behavior of the anchor subject parsing a written passage derived from the anchor item using a subject-extracted relative clause. By modeling κ as a single parameter we replace four degrees of freedom in our initial implementation of the model with only one.

For object-extracted relative clauses we have

$$\begin{aligned}
\mu_{ij}^{\mathrm{OR}} &= \exp\left(\eta + \gamma^{\mathrm{OR}} + \alpha_i - \beta_j\right) \\
&= \exp\left(\eta + \gamma^{\mathrm{SR}} + \alpha_{i'} - \beta_{j'} + \gamma^{\mathrm{OR}} - \gamma^{\mathrm{SR}} + \delta_i - \zeta_j\right) \\
&= \exp\left(\kappa + \chi + \delta_i - \zeta_j\right).
\end{aligned}$$

Here

$$\chi = \gamma^{\mathrm{OR}} - \gamma^{\mathrm{SR}}$$

quantifies the complexity of object-extracted relative clauses *relative* to subject-extracted relative clauses.

Because all of the included pairings are at least indirectly related to each other, the anchoring that we have implemented completely eliminates the redundancy of the observational model. For much more on anchoring in general, and how to manage pairings that decompose into multiple connected components, see the aforementioned pairwise comparison modeling chapter.

In theory we can anchor any item and subject, even if we have not observed that particular pairing. That said anchoring and item and subject that have been paired as many times as possible can reduce inferential degeneracies, which then reduces computational costs. Because the pairings of the items and subjects in this data set are relatively uniform, any choice of anchors should perform as well as any others. We'll anchor Item 1 and Subject 1 just out of convenience.

4.1.3 Prior Model

Before we can perform Bayesian inference we need to elevate our observational model to a full Bayesian model by specifying a prior model over all of the model configuration variables.

A prior model allows us to incorporates additional domain expertise into our inferences. Translating implicit, and often qualitative domain expertise into an explicit, quantitative prior model, however, takes time and effort. In practice we have only finite resources, so our goal isn't to build a prior model that perfectly incorporates *all* available domain expertise, but rather just the domain expertise that we think might improve any inferences of interest.

Critically we don't have to be precious here. If the domain expertise that we use initially proves to be insufficient, then we can always incorporate more into an updated prior model. In other words prior modeling it itself often iterative.

In general a prior model is defined by a joint probability distribution over all of the model configuration variables. Here we will make a common assumption that our domain expertise for each model configuration variable is independent. This allows the joint prior model to decompose into separate component prior models for each parameter.

To motivate one-dimensional component prior models I like to think in terms of **thresholds** that separate more reasonable behaviors from more extreme behaviors. For example, to avoid unrealistically fast and slow reading times for our baseline scenario we might want to constrain κ to

$$
\begin{aligned}
&100 \text{ milliseconds} \lessapprox \exp(\kappa) \cdot 1 \text{ millisecond} \lessapprox 1000 \text{ milliseconds} \\
&100 \lessapprox \exp(\kappa) \lessapprox 1000 \\
&\log 100 \lessapprox \kappa \lessapprox \log 1000.
\end{aligned}
$$

Note this isn't a constraint on the actual reading times but rather just the baseline location of the reading time distribution.

Model configurations outside of these thresholds are not impossible, just more extreme relative to our available domain expertise. We can capture this with a prior model that concentrates

most, but not all, of its probability within these thresholds,

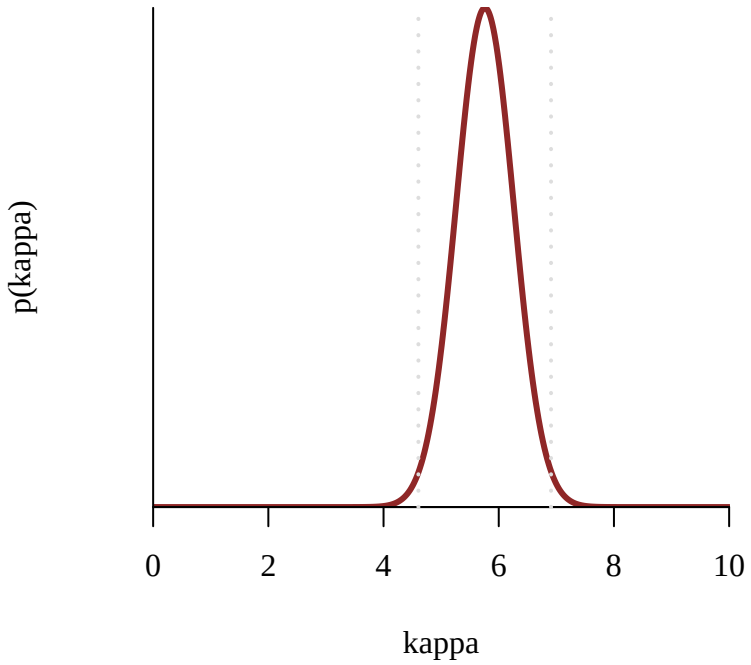
$$\begin{aligned} 1 - \epsilon &= \pi(\, [\log 100, \log 1000] \,) \\ &= \int_{\log 100}^{\log 1000} \mathrm{d}\kappa \, \mathrm{normal}(\kappa \mid m, s). \end{aligned}$$

We don't have to be too precise value about the definition of "most" here. I use $\epsilon = 0.02$ when considering two thresholds.

Conveniently we can analytically derive a normal prior model that satisfies this condition,

$$\begin{aligned} p(\kappa) &= \mathrm{normal} \left(\kappa \, \middle| \, \frac{\log 1000 + \log 100}{2}, \frac{1}{2.32} \frac{\log 1000 - \log 100}{2} \right) \\ &= \mathrm{normal}(\kappa \mid 5.76, 0.50). \end{aligned}$$

```
par(mfrow=c(1, 1), mar = c(5, 4, 2, 1))

xs <- seq(0, 10, 0.01)
plot(xs, dnorm(xs, 5.76, 0.50),
     type='l', yaxt='n',
     lwd=3, col=util$c_dark,
     xlab='kappa', ylab='p(kappa)')

abline(v=log(1e2), lwd=2, lty=3, col="#DDDDDD")
abline(v=log(1e3), lwd=2, lty=3, col="#DDDDDD")
```



To motivate component prior models for the other parameters let's consider their proportional influences on the baseline behavior. If our domain expertise suggests that the item complexities are within an order of magnitude of each other then we would want our prior model to enforce the constraint

$$\begin{aligned} \frac{1}{10} &\lessapprox \exp(\delta_i) &&\lessapprox 10 \\ \log \frac{1}{10} &\lessapprox \delta_i &&\lessapprox \log 10 \\ -\log 10 &\lessapprox \delta_i &&\lessapprox \log 10. \end{aligned}$$

We can achieve this soft containment with

$$\begin{aligned} p(\delta_i) &= \text{normal}\left(\delta_i \;\middle|\; \frac{\log 10 + (-\log 10)}{2}, \frac{1}{2.32}\frac{\log 10 - (-\log 10)}{2}\right) \\ &= \text{normal}\left(\delta_i \;\middle|\; 0, \frac{1}{2.32}\,\log 10\right) \\ &= \text{normal}(\delta_i \mid 0, 0.99). \end{aligned}$$

Note that this is a prior model for the *relative* item complexities. It does *not* imply the same prior model for the *absolute* item qualities,

$$p(\alpha_i) \neq \text{normal}(\alpha_i \mid 0, 0.99)!$$

Here we'll assume similar domain expertise for the relative reader skills and relative complexity between subject-extracted and object-extracted relative clauses,

$$\begin{aligned} p(\zeta_j) &= \text{normal}(\zeta_j \mid 0, 0.99) \\ p(\chi) &= \text{normal}(\chi \mid 0, 0.99). \end{aligned}$$

If have more precise domain expertise for any of these quantities then we should absolutely consider using it to motivate a more informative component prior model.

Lastly we need a component prior model for the scale parameter ϕ. For the log normal model, like most probabilistic models over positive real values, the scale parameter amplifies a baseline variance. When $\phi = 0$ the model collapses to a point and as $\phi \to \infty$ the model becomes infinitely diffuse.

Let's embrace order of magnitude again and use the soft constraints

$$0 \lessapprox \phi \lessapprox 10,$$

which we can enforce with the prior model

$$p(\psi) = \text{half-normal}\left(\psi \;\middle|\; 0, \frac{10}{2.57}\right) = \text{half-normal}\left(\psi \mid 0, 3.89\right).$$

One the main limitations of component prior models is that they ignore the interactions between the parameters as the data generating process evolves. Each component prior model might appear reasonable on their own, but together admit undesired behaviors. Fortunately we can check for any undesired interactions by performing a prior predictive check. In a prior predictive check we sample model configurations from the prior model and then simulate the data generating process, comparing the resulting outcomes to our domain expertise.

For example, what reading times are reasonable? Well, human reaction times are not far from 10^2 milliseconds; observed reading times an order of magnitude faster than this would definitely be a bit suspicious. On the other hand taking longer than

$$10^5 \text{ milliseconds} = 10^2 \text{ seconds} \approx 2 \text{ minutes}$$

to read a single word would also raise questions. Critically we are basing these thresholds on our domain expertise and *not* what we might have seen when we explored the observed data!

We can now use `Stan`'s `Fixed_param` configuration to generate prior predictive samples.

```
fit <- stan(file="stan_programs/dlt1_prior.stan",
            data=data, seed=8438338, algorithm="Fixed_param",
            warmup=0, iter=1024, refresh=0)

samples <- util$extract_expectand_vals(fit)
```

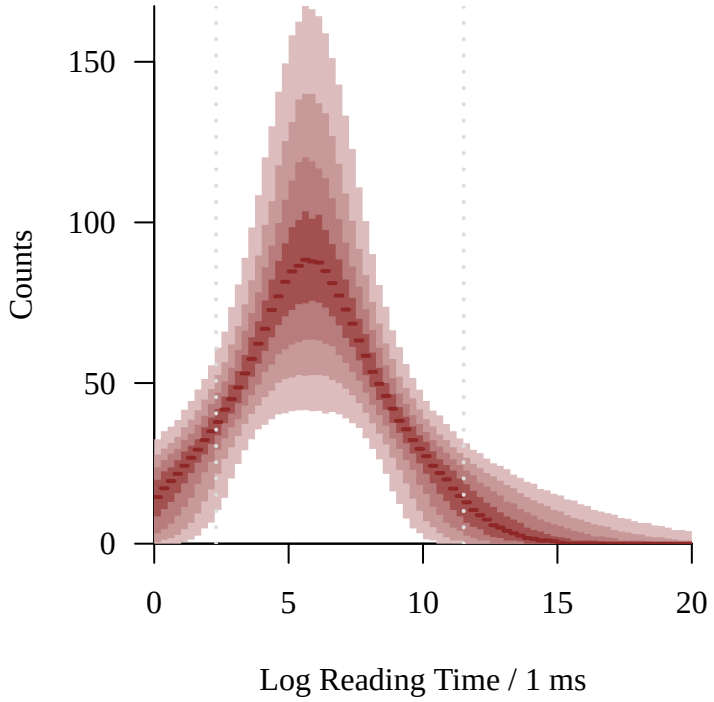
Then we can histogram the results and compare to our elicited thresholds.

```
par(mfrow=c(1, 1), mar=c(5, 5, 1, 1))

util$plot_hist_quantiles(samples, 'log_reading_time_pred',
                         0, 20, 0.25,
                         xlab='Log Reading Time / 1 ms')
```

```
Warning in check_bin_containment(bin_min, bin_max, collapsed_values,
"predictive value"): 715455 predictive values (6.4%) fell below the binning.
```

```
Warning in check_bin_containment(bin_min, bin_max, collapsed_values,
"predictive value"): 52859 predictive values (0.5%) fell above the binning.
```

```
abline(v=log(1e1), lwd=2, lty=3, col="#DDDDDD")
abline(v=log(1e5), lwd=2, lty=3, col="#DDDDDD")
```



The prior predictive reading times weakly concentrate within our thresholds; if anything the prior predictive behavior is too conservative, especially towards smaller reading times. For now we will push ahead, aware that our prior model is relatively weakly informative and could be improved if needed.

In addition to the aggregate histogram we could also examine the prior predictive behavior in the stratified histograms that we considered in Section 3. I will leave that as an exercise for the ambitious reader!

4.1.4 Inferential Computation

At this point we implement our full Bayesian model as a `Stan` program. The `Stan` package can then take this program, evaluate it on the observed data to construct a posterior density function, and then use Hamiltonian Monte Carlo to explore the implied posterior distribution.

```
fit <- stan(file="stan_programs/dlt1.stan",
            data=data, seed=8438338,
            warmup=1000, iter=2024, refresh=0)
```

First and foremost, we have to check for any signs that the posterior computation might be inaccurate.

```
diagnostics1 <- util$extract_hmc_diagnostics(fit)
util$check_all_hmc_diagnostics(diagnostics1)
```

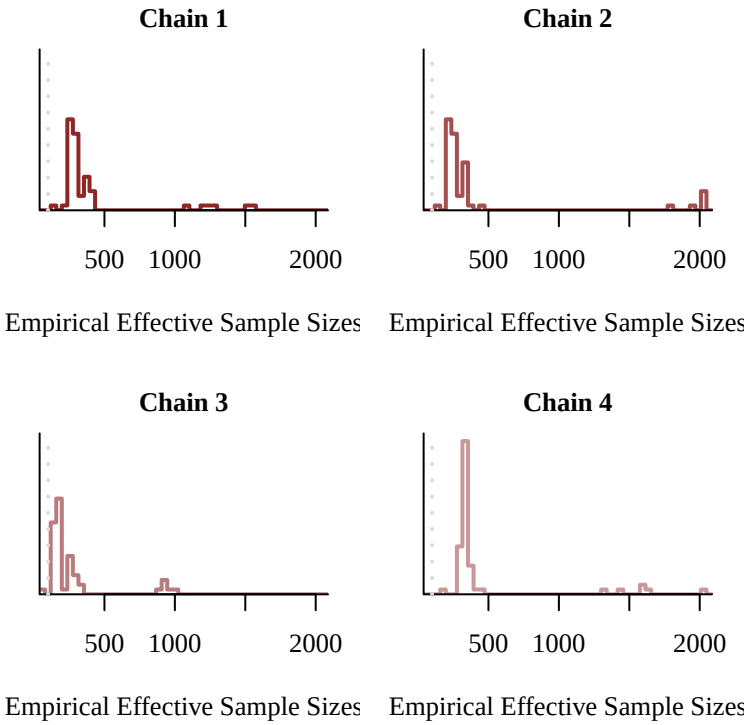
```
  All Hamiltonian Monte Carlo diagnostics are consistent with reliable
Markov chain Monte Carlo.
```

```
samples_dlt1 <- util$extract_expectand_vals(fit)
base_samples <- util$filter_expectands(samples_dlt1,
                                       c('kappa',
                                         'delta_free',
                                         'zeta_free',
                                         'chi', 'phi'),
                                       TRUE)
util$summarize_expectand_diagnostics(base_samples)
```

```
The expectands kappa triggered diagnostic warnings.

The expectands kappa triggered hat{ESS} warnings.

Small empirical effective sample sizes result in imprecise Markov chain
Monte Carlo estimators.
```

The empirical effective sample size warning are not fatal on their own; they just indicate that the Markov chains are exploring the posterior distribution slowly. This is especially true given that most of the small empirical effective sample sizes are coming from only one Markov chains, which indicates that this slow exploration is likely due to an unlucky adaptation of the respective Markov transition.

```
util$plot_ess_hats(base_samples, B=50)
```



4.1.5 Posterior Retrodictive Checks

Before investigating any posterior inferences we need to evaluate how well our full Bayesian model captures the relevant features of the observed data. Inferences drawn from a model that poorly fits the data are fragile at best and completely meaningless at worst.

I strongly recommend evaluating Bayesian models by comparing the behavior of the observed data to the behavior of the posterior predictive distribution with a posterior retrodictive check. Note the terminology here – we're implementing a *retrodictive* check because we're evaluating how well the model retrodicts data that it has already seen. *Predictive* checks require comparisons to new observations.

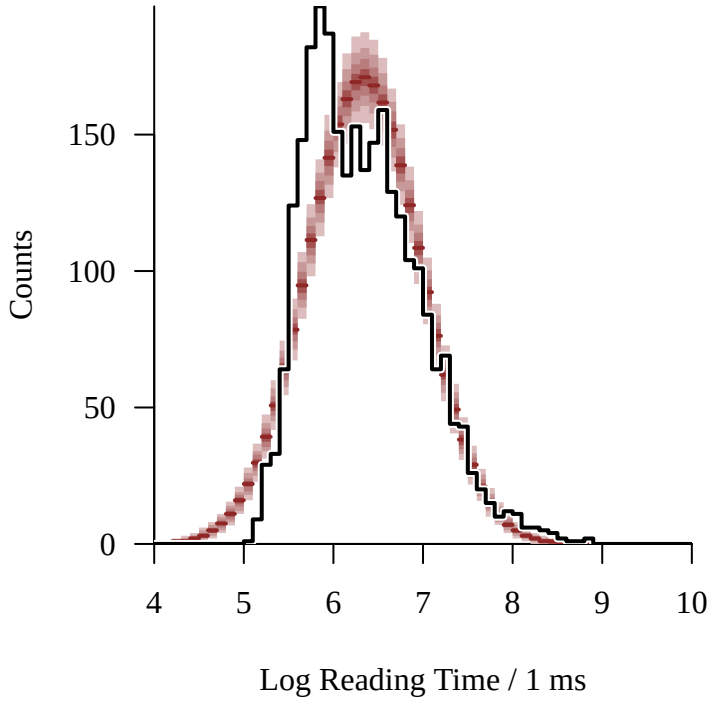
Each posterior retrodictive check starts with a summary statistic that isolates an interpretable, relevant feature of the observational space. Conveniently the summary statistics that are useful for data exploration are also useful for retrodictive checking, so we can build off of the work that we've already done.

For example, we might use the aggregate histogram summary statistic.

```
par(mfrow=c(1, 1), mar=c(5, 5, 1, 1))

util$plot_hist_quantiles(samples_dlt1, 'log_reading_time_pred',
```

```
                        4, 10, 0.1,
                        baseline_values=log(data$reading_time),
                        xlab='Log Reading Time / 1 ms')
```

```
Warning in check_bin_containment(bin_min, bin_max, collapsed_values,
"predictive value"): 1386 predictive values (0.0%) fell below the binning.
```



Unfortunately this comparison reveals substantial retrodictive tension. The posterior predictive behavior is much more symmetric than the observed behavior; specifically it exhibits a heavier left tail and lighter right tail. At the same time the posterior predictive behavior doesn't exhibit the same dip at moderate values that the observed data does.

This aggregate tension could be due to our initial model poorly fitting reading times from passages with either of the relative clauses. Fortunately we can readily investigate this by implementing posterior retrodictive checks for the histograms stratified by relative clause type.

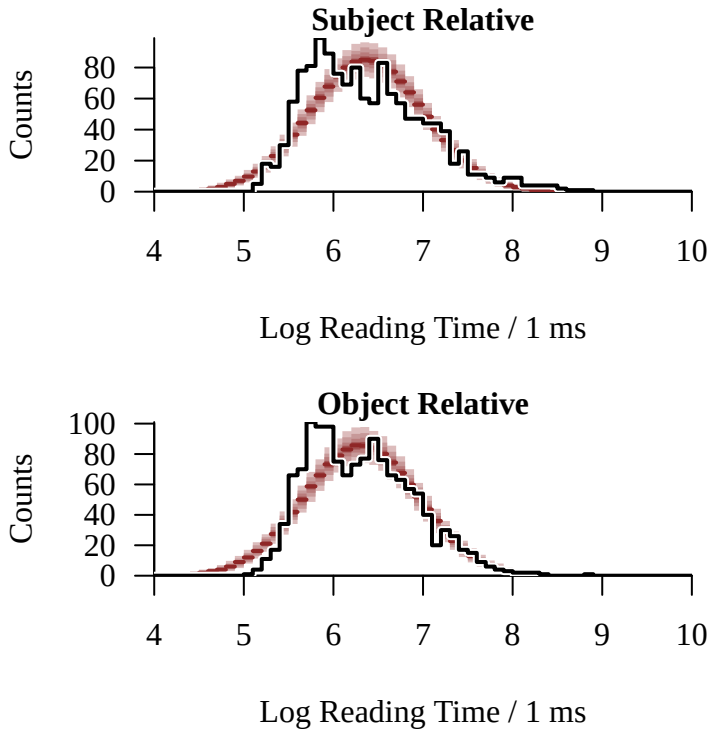
```
par(mfrow=c(2, 1), mar=c(5, 5, 1, 1))

names <- sapply(which(data$subj_rel == 1),
                function(n)
                paste0('log_reading_time_pred[', n, ']'))
filtered_samples <- util$filter_expectands(samples_dlt1, names)
```

```
obs_reading_time <- data$reading_time[data$subj_rel == 1]

util$plot_hist_quantiles(filtered_samples, 'log_reading_time_pred',
                         4, 10, 0.1,
                         baseline_values=log(obs_reading_time),
                         xlab='Log Reading Time / 1 ms',
                         main='Subject Relative')
```

```
Warning in check_bin_containment(bin_min, bin_max, collapsed_values,
"predictive value"): 527 predictive values (0.0%) fell below the binning.
```

```
names <- sapply(which(data$subj_rel == 0),
                function(n)
                paste0('log_reading_time_pred[', n, ']'))
filtered_samples <- util$filter_expectands(samples_dlt1, names)

obs_reading_time <- data$reading_time[data$subj_rel == 0]

util$plot_hist_quantiles(filtered_samples, 'log_reading_time_pred',
                         4, 10, 0.1,
                         baseline_values=log(obs_reading_time),
                         xlab='Log Reading Time / 1 ms',
                         main='Object Relative')
```

```
Warning in check_bin_containment(bin_min, bin_max, collapsed_values,
"predictive value"): 859 predictive values (0.0%) fell below the binning.
```



The aggregate retrodictive tension strongly persists into the object-extracted relative clause histogram. On the other hand, the disagreement appears to be a bit weaker in the subject-extracted relative clause observations.

At this point we could perform more retrodictive checks, for instance over the item-stratified and subject-stratified histograms, to build more and more understanding for *why* our initial model is inadequate, and hence how we can make it better. That said we already have a promising hypothesis – the shape of the tails.

4.2 Inverse Gamma Dependency Locality Model

The retrodictive checks suggest that a more asymmetric measurement variability model, with a lighter lower tail and heavier upper tail, might better fit the observed reading times. Fortunately the mean-dispersion inverse gamma model exhibits exactly this behavior relative to the log normal model.

Perhaps you have never heard of an inverse gamma model, let alone a mean-dispersion inverse gamma model, before? That is entirely reasonable.

Building probabilistic models is similar to writing a story in a foreign language. You may know what you want to write, but not have the vocabulary to express that in the new language. The log normal model, inverse gamma model, as well as pairwise comparison and mixture models

more generally, are all examples of probabilistic vocabulary that expand the kind of data generating stories you can tell.

Just as when learning a foreign language, the stories you can tell when first getting starting with probabilistic modeling will be simple and awkward. As you build your vocabulary by learning new modeling techniques, you will be able to tell richer and more sophisticated stories. Moreover, collaborating with a statistician fluent in this language can gives you a pretty good head start.

The mean-dispersion inverse gamma model uses all of the same parameters as the log normal model. All of these parameters maintain the exact same interpretation except for one: ϕ. While qualitatively similar in the two models, the precise influence of the scale parameter is slightly different.

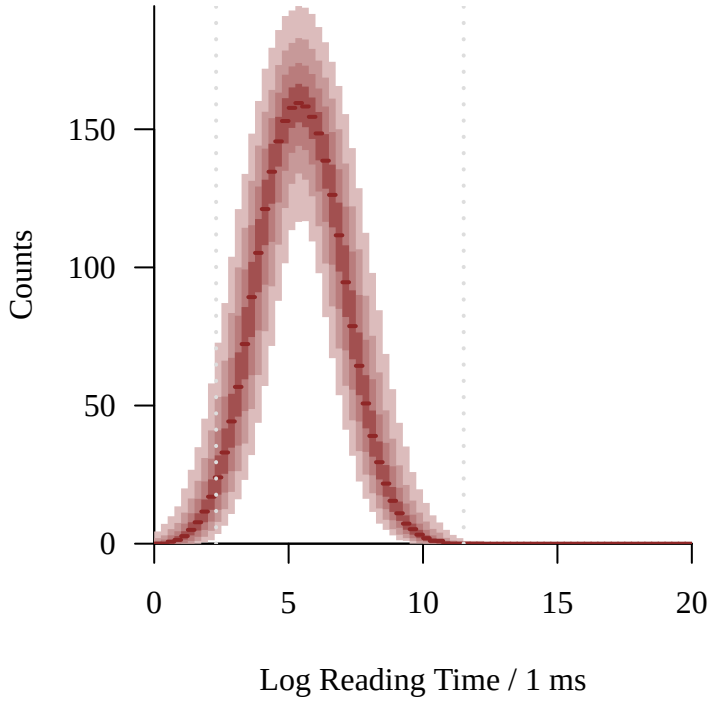
For the persistent parameters we can use the same component prior models. Because the scale parameters behave similarly we can also use the same component prior model for ϕ. We just need to be careful to verify reasonable prior predictive behavior.

```
fit <- stan(file="stan_programs/dlt2_prior.stan",
            data=data, seed=8438338, algorithm="Fixed_param",
            warmup=0, iter=1024, refresh=0)

samples <- util$extract_expectand_vals(fit)
```

```
par(mfrow=c(1, 1), mar=c(5, 5, 1, 1))

util$plot_hist_quantiles(samples, 'log_reading_time_pred',
                         0, 20, 0.25,
                         xlab='Log Reading Time / 1 ms')
```

```
Warning in check_bin_containment(bin_min, bin_max, collapsed_values,
"predictive value"): 12239 predictive values (0.1%) fell below the binning.
```

```
abline(v=log(1e1), lwd=2, lty=3, col="#DDDDDD")
abline(v=log(1e5), lwd=2, lty=3, col="#DDDDDD")
```



Conveniently the prior predictive behavior has become even more consistent with our conservatively-elicited reading time thresholds!

This is all very exciting, but what really matters is the posterior retrodictive performance of this new model. Before we can investigate that, however, we first have to quantify a new posterior distribution.

```
fit <- stan(file='stan_programs/dlt2.stan',
            data=data, seed=8438338,
            warmup=1000, iter=2024, refresh=0)
```

Fortunately the diagnostics are now much cleaner, indicating that our posterior quantification should be trustworthy.

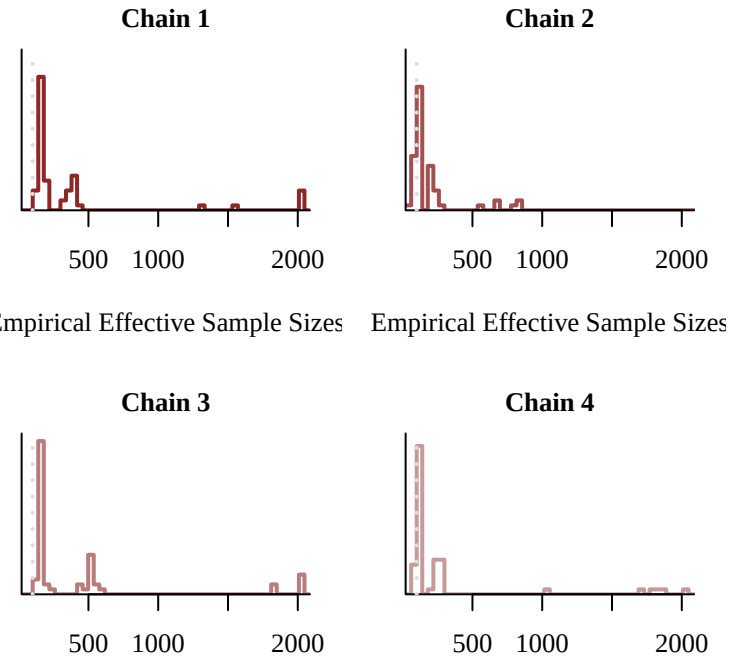
```
diagnostics <- util$extract_hmc_diagnostics(fit)
util$check_all_hmc_diagnostics(diagnostics)
```

```
  All Hamiltonian Monte Carlo diagnostics are consistent with reliable
Markov chain Monte Carlo.
```

```
samples_dlt2 <- util$extract_expectand_vals(fit)
base_samples <- util$filter_expectands(samples_dlt2,
                                       c('kappa',
                                         'delta_free',
                                         'zeta_free',
                                         'chi', 'phi'),
                                       TRUE)
util$summarize_expectand_diagnostics(base_samples)
```

```
The expectands kappa, zeta_free[2], zeta_free[3], zeta_free[8],
zeta_free[13], zeta_free[16], zeta_free[20], zeta_free[22],
zeta_free[27], zeta_free[29], zeta_free[36], zeta_free[37],
zeta_free[38], zeta_free[39] triggered diagnostic warnings.

The expectands kappa, zeta_free[2], zeta_free[3], zeta_free[8],
zeta_free[13], zeta_free[16], zeta_free[20], zeta_free[22],
zeta_free[27], zeta_free[29], zeta_free[36], zeta_free[37],
zeta_free[38], zeta_free[39] triggered hat{ESS} warnings.

Small empirical effective sample sizes result in imprecise Markov chain
Monte Carlo estimators.
```

```
util$plot_ess_hats(base_samples, B=50)
```

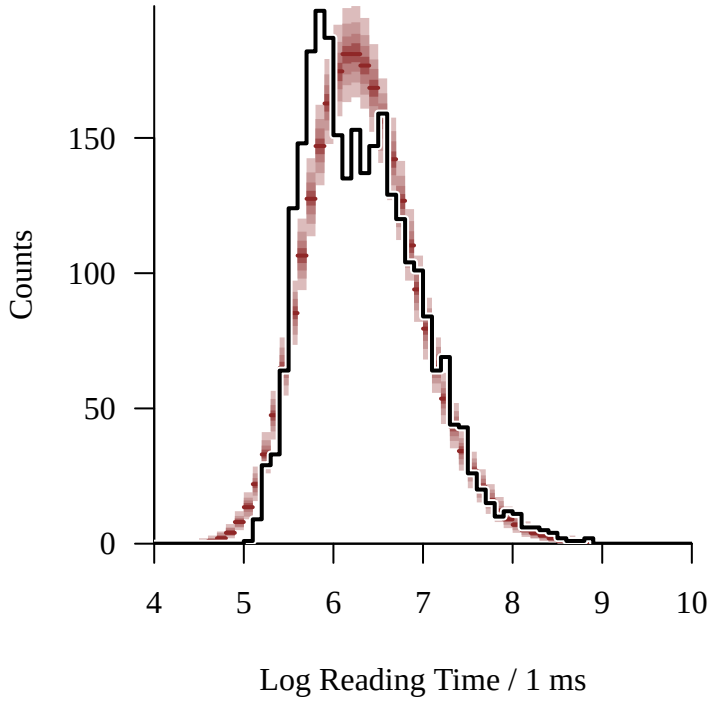


So did we do any better? The retrodictive tension in the upper tail of the aggregate histogram is now slightly better. Frustratingly, however, tension in the lower tail and the peak remain.

```
par(mfrow=c(1, 1), mar=c(5, 5, 1, 1))

util$plot_hist_quantiles(samples_dlt2, 'log_reading_time_pred',
                         4, 10, 0.1,
                         baseline_values=log(data$reading_time),
                         xlab='Log Reading Time / 1 ms')
```

```
Warning in check_bin_containment(bin_min, bin_max, collapsed_values,
"predictive value"): 10 predictive values (0.0%) fell below the binning.
```

```
Warning in check_bin_containment(bin_min, bin_max, collapsed_values,
"predictive value"): 172 predictive values (0.0%) fell above the binning.
```



The tension in the histograms stratified by relative clause is similar.

```
par(mfrow=c(2, 1), mar=c(5, 5, 1, 1))

names <- sapply(which(data$subj_rel == 1),
                function(n) paste0('log_reading_time_pred[', n, ']'))
filtered_samples <- util$filter_expectands(samples_dlt2, names)

obs_reading_time <- data$reading_time[data$subj_rel == 1]

util$plot_hist_quantiles(filtered_samples, 'log_reading_time_pred',
                         4, 10, 0.1,
                         baseline_values=log(obs_reading_time),
                         xlab='Log Reading Time / 1 ms',
                         main='Subject Relative')
```

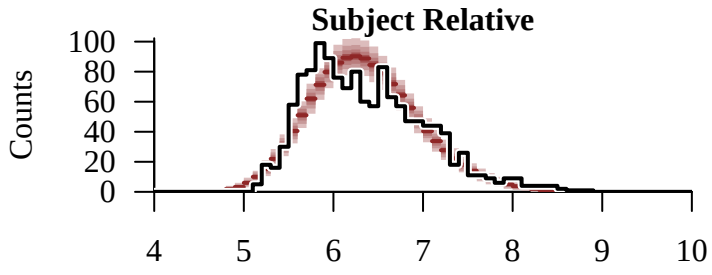
```
Warning in check_bin_containment(bin_min, bin_max, collapsed_values,
"predictive value"): 5 predictive values (0.0%) fell below the binning.
```

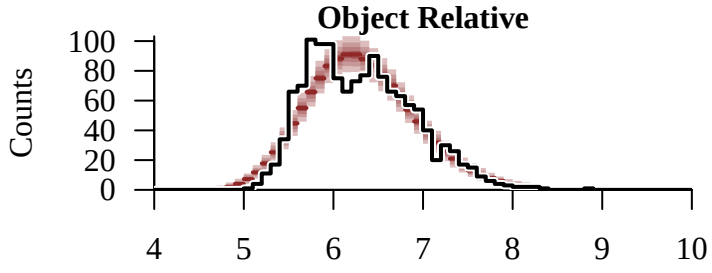
```
Warning in check_bin_containment(bin_min, bin_max, collapsed_values,
"predictive value"): 97 predictive values (0.0%) fell above the binning.
```

```
names <- sapply(which(data$subj_rel == 0),
                function(n) paste0('log_reading_time_pred[', n, ']'))
filtered_samples <- util$filter_expectands(samples_dlt2, names)

obs_reading_time <- data$reading_time[data$subj_rel == 0]


util$plot_hist_quantiles(filtered_samples, 'log_reading_time_pred',
                         4, 10, 0.1,
                         baseline_values=log(obs_reading_time),
                         xlab='Log Reading Time / 1 ms',
                         main='Object Relative')
```

```
Warning in check_bin_containment(bin_min, bin_max, collapsed_values,
"predictive value"): 5 predictive values (0.0%) fell below the binning.
```

```
Warning in check_bin_containment(bin_min, bin_max, collapsed_values,
"predictive value"): 75 predictive values (0.0%) fell above the binning.
```





At this point it's not clear what we can do to save the dependency locality model. The observed data seems to be exhibiting two peaks that the dependency locality model just can't reproduce on its own. Speaking of multiple peaks, however, how about that direct-access model?

4.3 Log Normal Direct-Access Model

So long as there is a chance of initial failure, the direct-access model always results in bimodal reading time behaviors. That might be just want we need to resolve our retrodictive tension.

4.3.1 Observational Model

Recall that the direct-access model implies the observational models

$$\lambda_{\text{SR}}\, p_1(t_{ij} \mid \mu_{1,ij}, \phi_1) + (1 - \lambda_{\text{SR}})\, p_2(t_{ij} \mid \mu_{2,ij}, \phi_2)$$

and

$$\lambda_{\text{OR}}\, p_1(t_{ij} \mid \mu_{1,ij}, \phi_1) + (1 - \lambda_{\text{OR}})\, p_2(t_{ij} \mid \mu_{2,ij}, \phi_2).$$

where

$$\mu_{1,ij} = \exp\left(\eta + \alpha_i - \beta_j\right)$$

and

$$\mu_{2,ij} = \exp\left(\eta + \omega + \alpha_i - \beta_j\right).$$

As we did with the dependency locality model, let's start with a log normal model for both components,

$$\lambda_{\text{SR}}\, \text{log-normal}(t_{ij} \mid \mu_{1,ij}, \phi_1) + (1 - \lambda_{\text{SR}})\, \text{log-normal}(t_{ij} \mid \mu_{2,ij}, \phi_2)$$

and

$$\lambda_{\text{OR}}\, \text{log-normal}(t_{ij} \mid \mu_{1,ij}, \phi_1) + (1 - \lambda_{\text{OR}})\, \text{log-normal}(t_{ij} \mid \mu_{2,ij}, \phi_2).$$

4.3.2 Anchoring

The pairwise-comparison structure of the direct-access model exhibits the same redundancies as the pairwise-comparison structure of the dependency locality model. Fortunately that means that we can eliminate the redundancy with the same anchoring strategy.

Selecting a distinguished anchor item i' allows us to define relative complexities

$$\delta_i = \alpha_i - \alpha_{i'},$$

while choosing a distinguished anchor subject j' allows us to define relative skills,

$$\zeta_j = \beta_j - \beta_{j'}.$$

This allows us to write

$$\begin{aligned}\mu_{1,ij} &= \exp\left(\eta + \alpha_i - \beta_j\right) \\ &= \exp\left(\eta + \alpha_{i'} - \beta_{j'} + \delta_i - \zeta_j\right) \\ &\equiv \exp\left(\nu + \delta_i - \zeta_j\right)\end{aligned}$$

and

$$\begin{aligned}\mu_{2,ij} &= \exp\left(\eta + \omega + \alpha_i - \beta_j\right) \\ &= \exp\left(\eta + \alpha_{i'} - \beta_{j'} + \omega + \delta_i - \zeta_j\right) \\ &= \exp\left(\nu + \omega + \delta_i - \zeta_j\right).\end{aligned}$$

with

$$\nu = \eta + \alpha_{i'} - \beta_{j'}.$$

Once again we'll anchor Item 1 and Subject 1.

4.3.3 Prior Model

The relative complexity and skill parameters in the direct-access model and dependency locality model are equivalent. Consequently we can use the same component prior models for both,

$$\begin{aligned}p(\delta_i) &= \text{normal}(\delta_i \mid 0, 0.99) \\ p(\zeta_j) &= \text{normal}(\zeta_j \mid 0, 0.99).\end{aligned}$$

Similarly we will use the same component prior models for the scale parameters as we did before,

$$\begin{aligned}p(\psi_1) &= \text{half-normal}\left(\psi_1 \mid 0, 3.89\right) \\ p(\psi_2) &= \text{half-normal}\left(\psi_2 \mid 0, 3.89\right).\end{aligned}$$

This leaves the model configuration variables ν and ω which are similar, but not exactly the same, as the variables κ and χ from the dependency locality model. Because our domain expertise elicitation was not particularly precise, however, we would likely end up with equivalent component prior models. Consequently we'll immediately take

$$p(\nu) = \text{normal}(\nu \mid 5.76, 0.50)$$

That said, we have to be a bit more careful about the component prior model for ω which, unlike χ, is constrained to positive values. Instead we'll need

$$\begin{aligned}1 &\lessapprox \exp(\omega) \lessapprox 10 \\ \log 1 &\lessapprox \omega \lessapprox \log 10 \\ 0 &\lessapprox \omega \lessapprox \log 10.\end{aligned}$$

We can accomplish this the half-normal component prior model

$$\begin{aligned} p(\omega) &= \text{half-normal}\left(\omega \mid 0, \frac{1}{2.57}\log 10\right) \\ &= \text{half-normal}\left(\omega \mid 0, 0.90\right) \end{aligned}$$

Lastly we need to consider component prior models for the probabilities λ_{SR} and λ_{OR}. While there is surely wealth of knowledge about the probability of initial hypotheses failing, that knowledge is not available to me at the moment. Given my limited domain expertise I will consider uniform prior models for both probabilities. If we want to be explicit about choosing uniform component prior models, then we can write them as

$$\begin{aligned} p(\lambda_{\text{SR}}) &= \text{beta}(\lambda_{\text{SR}} \mid 1, 1) \\ p(\lambda_{\text{OR}}) &= \text{beta}(\lambda_{\text{OR}} \mid 1, 1). \end{aligned}$$

To double check that this mostly copy-and-pasted prior model behaves reasonably well when passed through the direct-access theory model, let's implement another prior predictive check.

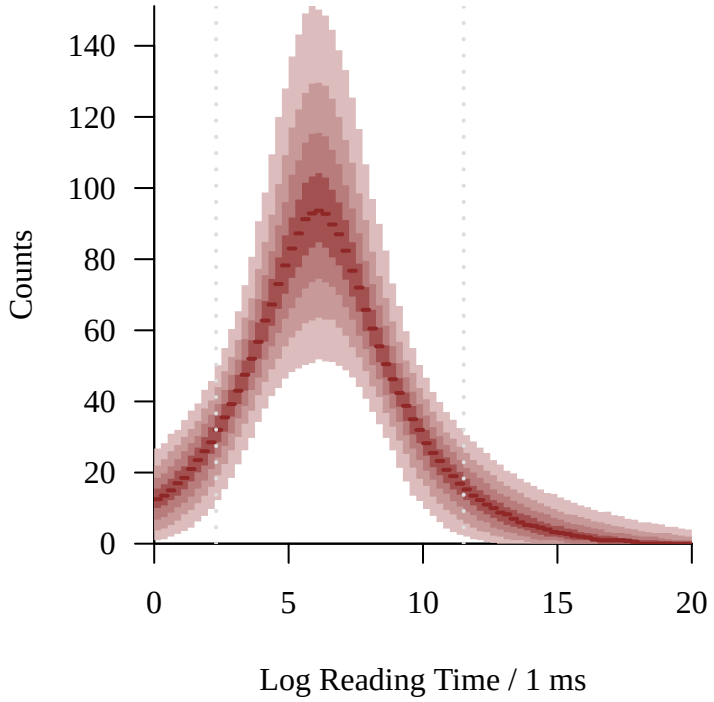
```
fit <- stan(file="stan_programs/dat1_prior.stan",
            data=data, seed=8438338, algorithm="Fixed_param",
            warmup=0, iter=1024, refresh=0)

samples <- util$extract_expectand_vals(fit)
```

```
par(mfrow=c(1, 1), mar=c(5, 5, 1, 1))

util$plot_hist_quantiles(samples, 'log_reading_time_pred',
                         0, 20, 0.25,
                         xlab='Log Reading Time / 1 ms')
```

```
Warning in check_bin_containment(bin_min, bin_max, collapsed_values,
"predictive value"): 643174 predictive values (5.7%) fell below the binning.
```

```
Warning in check_bin_containment(bin_min, bin_max, collapsed_values,
"predictive value"): 62182 predictive values (0.6%) fell above the binning.
```

```
abline(v=log(1e1), lwd=2, lty=3, col="#DDDDDD")
abline(v=log(1e5), lwd=2, lty=3, col="#DDDDDD")
```



Fortunately there don't seem to be any surprises. The prior predictive behavior is very similar to what we saw with the log-normal locality dependency model.

4.3.4 Posterior Quantification

Having taken care to both implement the observational model without redundancy and consider all our prior modeling assumptions, we are ready for inference.

```
fit <- stan(file='stan_programs/dat1.stan',
            data=data, seed=8438338,
            warmup=1000, iter=2024, refresh=0)
```

Unfortunately, multiple diagnostics indicate that our posterior computation is not to be trusted.

```
diagnostics <- util$extract_hmc_diagnostics(fit)
util$check_all_hmc_diagnostics(diagnostics)
```

```
  All Hamiltonian Monte Carlo diagnostics are consistent with reliable
Markov chain Monte Carlo.
```

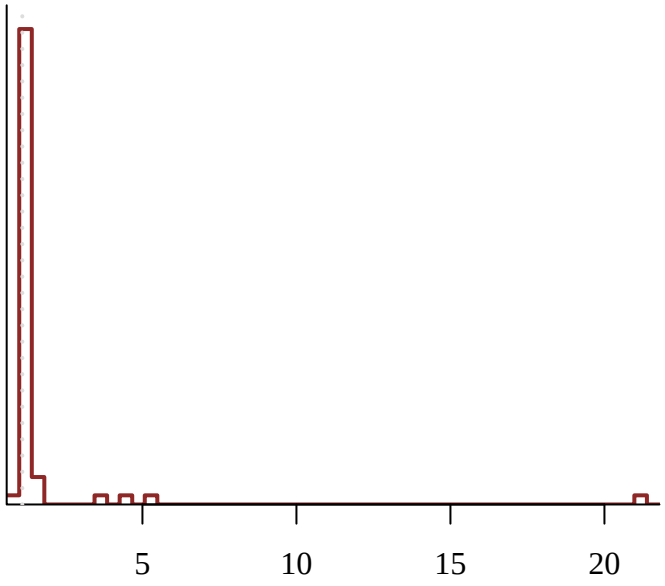
```
samples_dat1 <- util$extract_expectand_vals(fit)
base_samples <- util$filter_expectands(samples_dat1,
                                       c('nu',
                                         'delta_free',
                                         'zeta_free',
                                         'omega',
                                         'phi1', 'phi2',
                                         'lambda_SR',
                                         'lambda_OR'),
                                       TRUE)
util$summarize_expectand_diagnostics(base_samples)
```

```
The expectands nu, delta_free[6], delta_free[7], delta_free[9],
delta_free[14], delta_free[15], zeta_free[1], zeta_free[2],
zeta_free[3], zeta_free[4], zeta_free[5], zeta_free[6], zeta_free[7],
zeta_free[8], zeta_free[10], zeta_free[11], zeta_free[13],
zeta_free[14], zeta_free[15], zeta_free[16], zeta_free[17],
zeta_free[18], zeta_free[19], zeta_free[20], zeta_free[22],
zeta_free[23], zeta_free[25], zeta_free[27], zeta_free[28],
zeta_free[29], zeta_free[30], zeta_free[31], zeta_free[32],
zeta_free[33], zeta_free[34], zeta_free[35], zeta_free[36],
zeta_free[37], zeta_free[38], zeta_free[39], omega, phi1, lambda_OR
triggered diagnostic warnings.

The expectands nu, delta_free[6], delta_free[7], delta_free[9],
delta_free[14], delta_free[15], zeta_free[1], zeta_free[5],
zeta_free[8], zeta_free[13], zeta_free[14], zeta_free[16],
zeta_free[18], zeta_free[25], zeta_free[27], zeta_free[28],
zeta_free[29], zeta_free[30], zeta_free[31], zeta_free[32],
zeta_free[37], zeta_free[38], zeta_free[39], omega, phi1, lambda_OR
triggered hat{R} warnings.

Split Rhat larger than 1.1 suggests that at least one of the Markov
chains has not reached an equilibrium.

The expectands nu, zeta_free[1], zeta_free[2], zeta_free[3],
zeta_free[4], zeta_free[5], zeta_free[6], zeta_free[7], zeta_free[8],
zeta_free[10], zeta_free[11], zeta_free[14], zeta_free[15],
zeta_free[16], zeta_free[17], zeta_free[18], zeta_free[19],
zeta_free[20], zeta_free[22], zeta_free[23], zeta_free[25],
zeta_free[27], zeta_free[30], zeta_free[31], zeta_free[32],
zeta_free[33], zeta_free[34], zeta_free[35], zeta_free[36],
```

```
zeta_free[37], zeta_free[38] triggered hat{ESS} warnings.

Small empirical effective sample sizes result in imprecise Markov chain
Monte Carlo estimators.
```

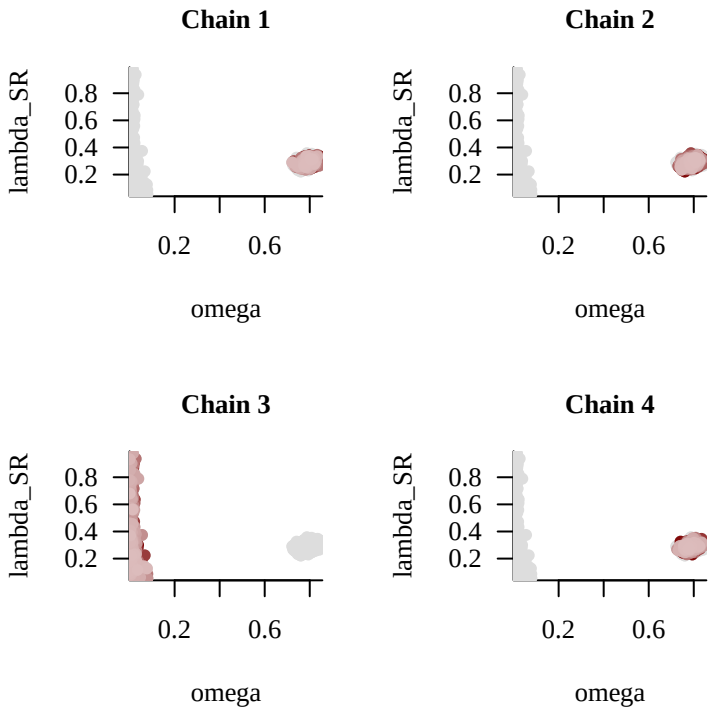
The immediate concern here are the split $\hat{R}$ warnings which suggest inferential multimodality.

```
util$plot_rhats(base_samples, B=50)
```



Indeed the first and third Markov chains appear to be exploring one set of behaviors while the second and fourth Markov chains are exploring an entirely different set of behaviors. The latter two Markov chains are exploring model configurations where ω is consistent with zero. In this case the two mixture components are equivalent, and every value of λ_{SR} and λ_{OR} yields the same reading time behaviors.

```
util$plot_pairs_by_chain(samples_dat1[['omega']], 'omega',
                         samples_dat1[['lambda_SR']], 'lambda_SR')
```



There is no guarantee that these are all of the posterior modes, nor that either of these modes actually provide a non-negligible influence to posterior inferences. Multimodal posterior distributions are notoriously difficult to quantify, but at least Markov chain Monte Carlo provides useful diagnostic information.

4.3.5 Posterior Retrodictive Checks

One way to qualify the relative importance of each mode is to compare how consistent they are with the observed data. In other words we can perform posterior retrodictive checks separately for each pair of Markov chains.

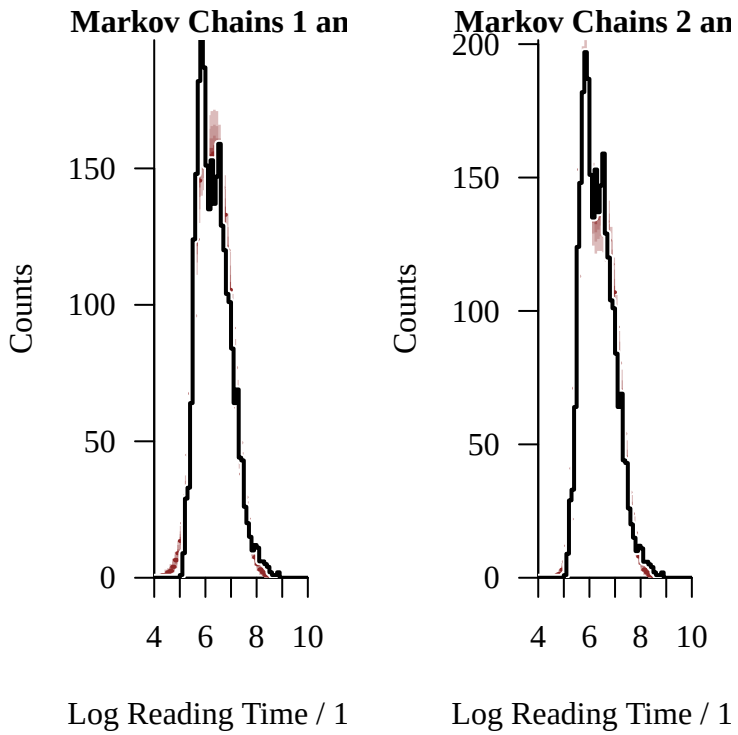
```
par(mfrow=c(1, 2), mar=c(5, 5, 1, 1))

util$plot_hist_quantiles(lapply(samples_dat1,
                                function(s) s[c(1, 3),]),
                         'log_reading_time_pred', 4, 10, 0.1,
                         baseline_values=log(data$reading_time),
                         xlab='Log Reading Time / 1 ms',
                         main='Markov Chains 1 and 3')
```

```
Warning in check_bin_containment(bin_min, bin_max, collapsed_values,
"predictive value"): 1437 predictive values (0.0%) fell below the binning.
```

```
Warning in check_bin_containment(bin_min, bin_max, collapsed_values,
"predictive value"): 23 predictive values (0.0%) fell above the binning.
```

```
util$plot_hist_quantiles(lapply(samples_dat1,
                                function(s) s[c(2, 4),]),
                         'log_reading_time_pred', 4, 10, 0.1,
                         baseline_values=log(data$reading_time),
                         xlab='Log Reading Time / 1 ms',
                         main='Markov Chains 2 and 4')
```

```
Warning in check_bin_containment(bin_min, bin_max, collapsed_values,
"predictive value"): 18 predictive values (0.0%) fell below the binning.
```



Because ω is so close to zero in the model configurations explored by the second and forth Markov chains, the posterior predictive reading time behavior exhibits only a single peak which clashes with the observed reading time behavior. On the other hand, the first and third Markov chains appear to be exploring behaviors that are much more consistent with the observed data. Formally quantifying the relative consistency, however, is difficult.

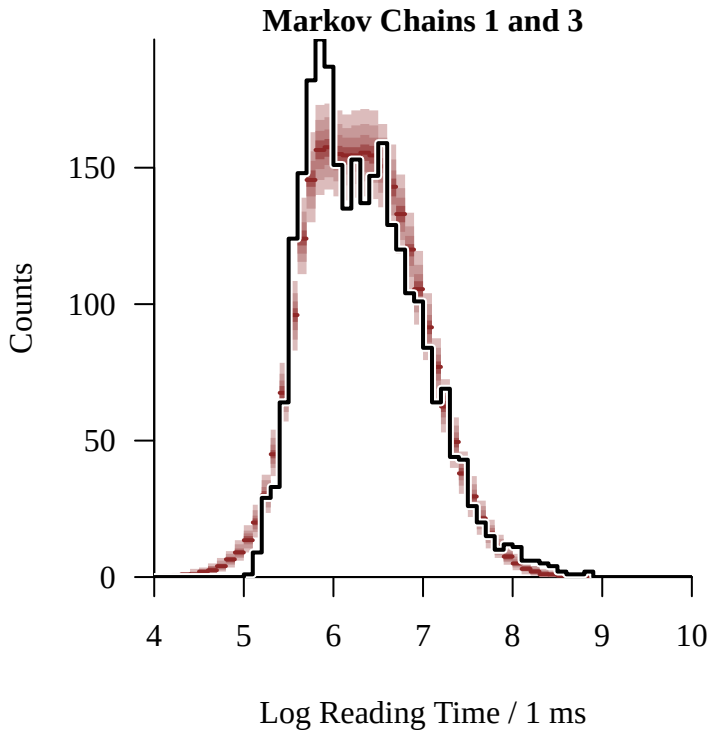
All of that said, the retrodictive performance for the first and third Markov chains still leaves something to be desired. If we zoom in we see that, while the posterior predictive behavior is

able to replicate the observed peaks, the posterior predictive tails are not quite right. Similar to what we saw with the log-normal dependency locality model, the posterior predictive lower tail is too heavy and the posterior predictive upper tail is too light.

```
par(mfrow=c(1, 1), mar=c(5, 5, 1, 1))

util$plot_hist_quantiles(lapply(samples_dat1,
                                function(s) s[c(1, 3),]),
                         'log_reading_time_pred', 4, 10, 0.1,
                         baseline_values=log(data$reading_time),
                         xlab='Log Reading Time / 1 ms',
                         main='Markov Chains 1 and 3')
```

```
Warning in check_bin_containment(bin_min, bin_max, collapsed_values,
"predictive value"): 1437 predictive values (0.0%) fell below the binning.
```

```
Warning in check_bin_containment(bin_min, bin_max, collapsed_values,
"predictive value"): 23 predictive values (0.0%) fell above the binning.
```



Even if the mode explored by the second and forth Markov chains is irrelevant, the posterior retrodictive tension in the mode explored by the first and third Markov chains suggests that

we can improve the direct-access model by adding a bit more asymmetry to the component models.

4.4 Inverse Gamma Direct-Access Model

To improve our initial direct-access model, let's try the same strategy that we attempted with the dependency locality model: replacing the log-normal measurement variability models with inverse gamma models.

4.4.1 Prior Model

As we did in Section 4.2 we will carry over our initial prior model to this second model, even though the inverse gamma scale parameters behave slightly differently from the log normal scale parameters. To double check that this doesn't cause any problems we'll run another quick prior predictive check.

Fortunately everything seems reasonably consistent with our elicited domain expertise.

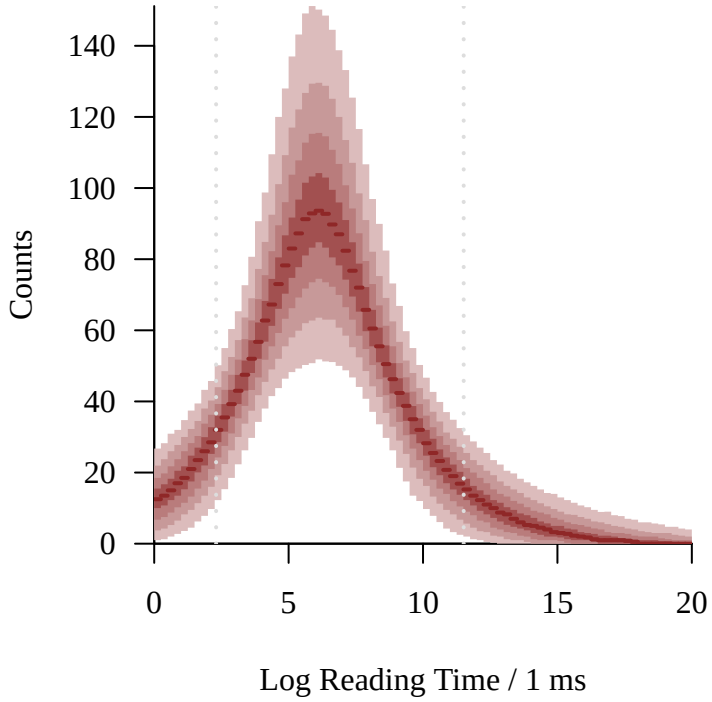
```
fit <- stan(file="stan_programs/dat2_prior.stan",
            data=data, seed=8438338, algorithm="Fixed_param",
            warmup=0, iter=1024, refresh=0)

samples <- util$extract_expectand_vals(fit)
```

```
par(mfrow=c(1, 1), mar=c(5, 5, 1, 1))

util$plot_hist_quantiles(samples, 'log_reading_time_pred',
                         0, 20, 0.25,
                         xlab='Log Reading Time / 1 ms')
```

```
Warning in check_bin_containment(bin_min, bin_max, collapsed_values,
"predictive value"): 643174 predictive values (5.7%) fell below the binning.
```

```
Warning in check_bin_containment(bin_min, bin_max, collapsed_values,
"predictive value"): 62182 predictive values (0.6%) fell above the binning.
```

```
abline(v=log(1e1), lwd=2, lty=3, col="#DDDDDD")
abline(v=log(1e5), lwd=2, lty=3, col="#DDDDDD")
```



4.4.2 Posterior Quantification

We now set our Markov chains free and cross our fingers.

```
fit <- stan(file='stan_programs/dat2.stan',
            data=data, seed=8438338,
            warmup=1000, iter=2024, refresh=0)
```

Has posterior quantification become any better with this new model?

```
diagnostics <- util$extract_hmc_diagnostics(fit)
util$check_all_hmc_diagnostics(diagnostics)
```

```
  All Hamiltonian Monte Carlo diagnostics are consistent with reliable
Markov chain Monte Carlo.
```

```
samples_dat2 <- util$extract_expectand_vals(fit)
base_samples <- util$filter_expectands(samples_dat2,
                                       c('nu',
                                         'delta_free',
```

```
                                          'zeta_free',
                                          'omega',
                                          'phi1', 'phi2',
                                          'lambda_SR', 'lambda_OR'),
                                        TRUE)
util$check_all_expectand_diagnostics(base_samples)
```

```
nu:
  Chain 1: hat{ESS} (95.858) is smaller than desired (100).

zeta_free[10]:
  Chain 1: hat{ESS} (90.622) is smaller than desired (100).

zeta_free[15]:
  Chain 1: hat{ESS} (94.220) is smaller than desired (100).

zeta_free[33]:
  Chain 1: hat{ESS} (86.969) is smaller than desired (100).

zeta_free[35]:
  Chain 1: hat{ESS} (97.021) is smaller than desired (100).


Small empirical effective sample sizes result in imprecise Markov chain
Monte Carlo estimators.
```

The split $\hat{R}$ warnings are nowhere to be seen, and only a few empirical effective sample size warnings persist. Because those low empirical effective sample sizes are all close to the desired threshold, and no other diagnostics are indicating concern, we can be relatively confident in the accuracy of our posterior computation.

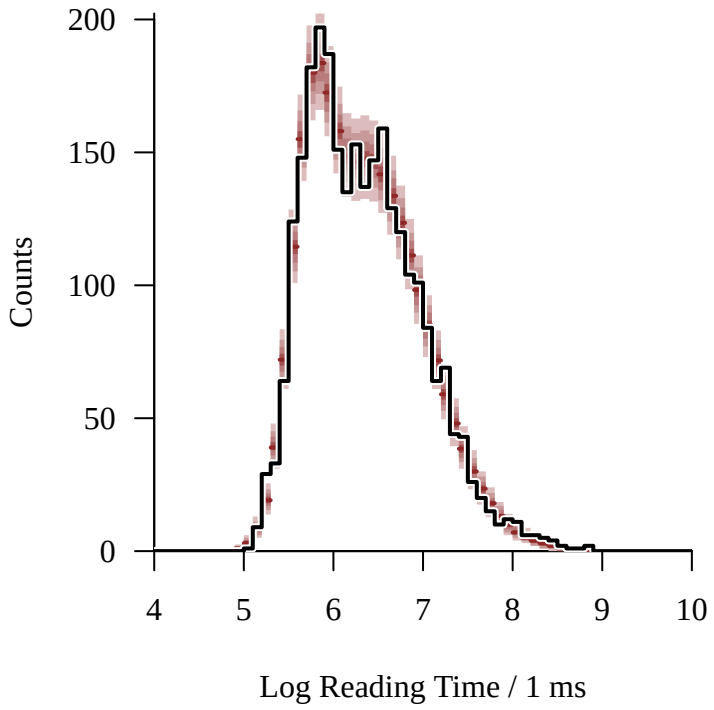
4.4.3 Retrodictive Checks

The diagnostics indicate that our estimated posterior predictive behavior should be faithful to the exact posterior predictive behavior. This allows us to make meaningful posterior retrodictive checks. And what beautiful checks they are.

```
par(mfrow=c(1, 1), mar=c(5, 5, 1, 1))

util$plot_hist_quantiles(samples_dat2, 'log_reading_time_pred',
```

```
                          4, 10, 0.1,
                          baseline_values=log(data$reading_time),
                          xlab='Log Reading Time / 1 ms')
```

```
Warning in check_bin_containment(bin_min, bin_max, collapsed_values,
"predictive value"): 73 predictive values (0.0%) fell above the binning.
```



The posterior predictive reading times now match the observed reading times, in both the peak and the tails. This agreement persists even if we stratify by variant.

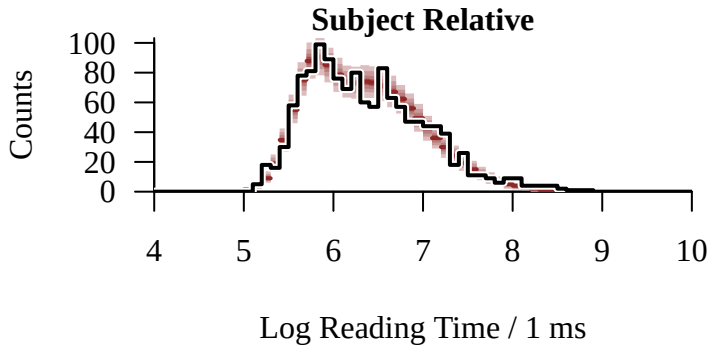
```
par(mfrow=c(2, 1), mar=c(5, 5, 1, 1))

names <- sapply(which(data$subj_rel == 1),
                function(n)
                paste0('log_reading_time_pred[', n, ']'))
filtered_samples <- util$filter_expectands(samples_dat2, names)

obs_reading_time <- data$reading_time[data$subj_rel == 1]

util$plot_hist_quantiles(filtered_samples,
                         'log_reading_time_pred',
```

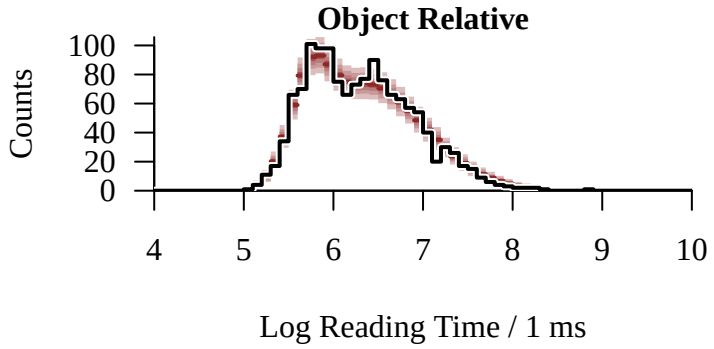
```
                         4, 10, 0.1,
                         baseline_values=log(obs_reading_time),
                         xlab='Log Reading Time / 1 ms',
                         main='Subject Relative')
```

```
Warning in check_bin_containment(bin_min, bin_max, collapsed_values,
"predictive value"): 43 predictive values (0.0%) fell above the binning.
```

```
names <- sapply(which(data$subj_rel == 0),
                function(n)
                paste0('log_reading_time_pred[', n, ']'))
filtered_samples <- util$filter_expectands(samples_dat2, names)

obs_reading_time <- data$reading_time[data$subj_rel == 0]


util$plot_hist_quantiles(filtered_samples,
                         'log_reading_time_pred',
                         4, 10, 0.1,
                         baseline_values=log(obs_reading_time),
                         xlab='Log Reading Time / 1 ms',
                         main='Object Relative')
```

```
Warning in check_bin_containment(bin_min, bin_max, collapsed_values,
"predictive value"): 30 predictive values (0.0%) fell above the binning.
```

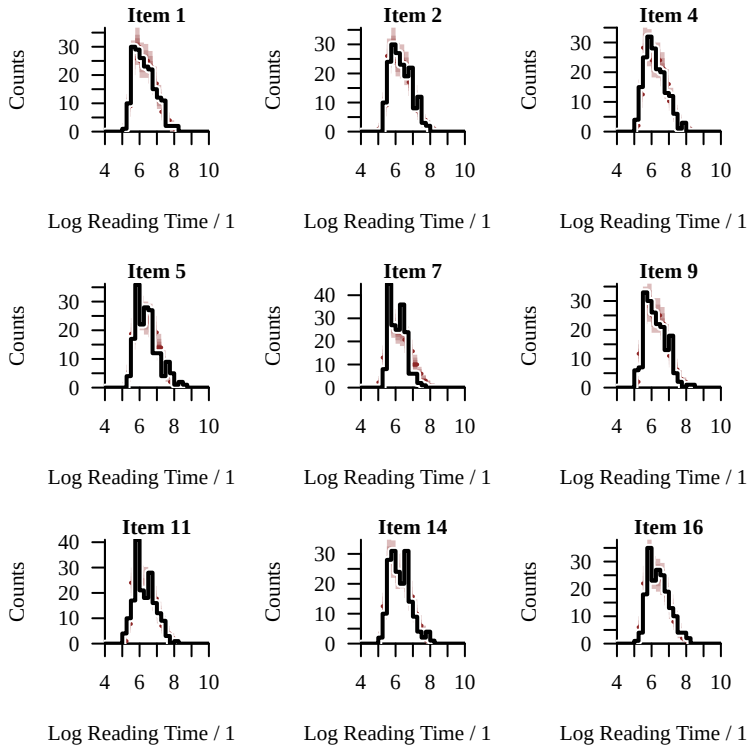




When we stratify by item and subject the observed behavior becomes a bit spikier, perhaps at artifact of discretization. Overall, however, the posterior predictive behavior is pretty consistent with the observed behavior.

```
par(mfrow=c(3, 3), mar=c(5, 5, 1, 1))

for (i in display_items) {
  names <- sapply(which(data$item == i),
                  function(n)
                  paste0('log_reading_time_pred[', n, ']'))
  filtered_samples <- util$filter_expectands(samples_dat2, names)

  obs_reading_time <- data$reading_time[data$item == i]

  util$plot_hist_quantiles(filtered_samples,
                           'log_reading_time_pred',
                           4, 10, 0.25,
                           baseline_values=log(obs_reading_time),
                           xlab='Log Reading Time / 1 ms',
                           main=paste('Item', i))
}
```

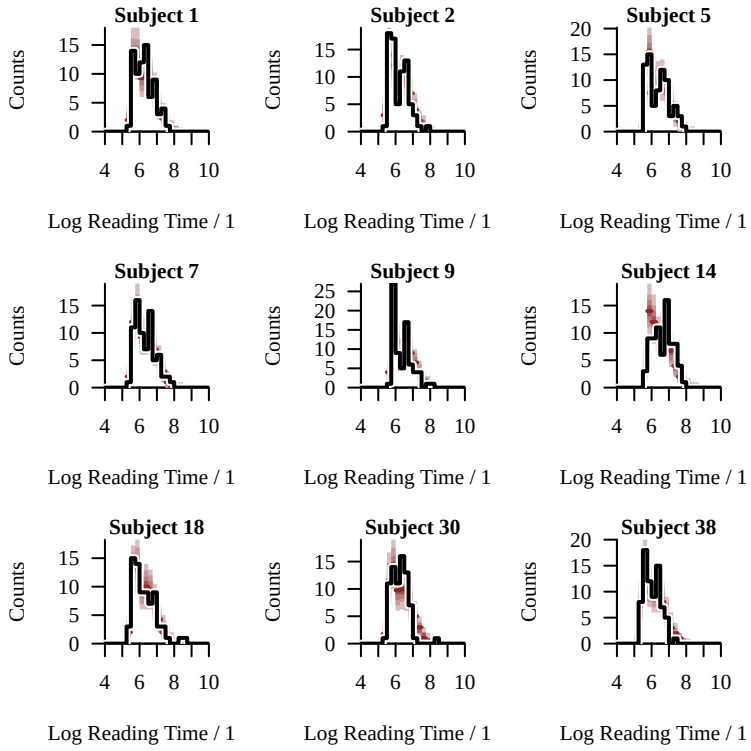
```
Warning in check_bin_containment(bin_min, bin_max, collapsed_values,
"predictive value"): 5 predictive values (0.0%) fell above the binning.

Warning in check_bin_containment(bin_min, bin_max, collapsed_values,
"predictive value"): 4 predictive values (0.0%) fell above the binning.

Warning in check_bin_containment(bin_min, bin_max, collapsed_values,
"predictive value"): 5 predictive values (0.0%) fell above the binning.

Warning in check_bin_containment(bin_min, bin_max, collapsed_values,
"predictive value"): 6 predictive values (0.0%) fell above the binning.

Warning in check_bin_containment(bin_min, bin_max, collapsed_values,
"predictive value"): 5 predictive values (0.0%) fell above the binning.

Warning in check_bin_containment(bin_min, bin_max, collapsed_values,
"predictive value"): 4 predictive values (0.0%) fell above the binning.

Warning in check_bin_containment(bin_min, bin_max, collapsed_values,
"predictive value"): 4 predictive values (0.0%) fell above the binning.

Warning in check_bin_containment(bin_min, bin_max, collapsed_values,
"predictive value"): 3 predictive values (0.0%) fell above the binning.

Warning in check_bin_containment(bin_min, bin_max, collapsed_values,
"predictive value"): 2 predictive values (0.0%) fell above the binning.
```




```
par(mfrow=c(3, 3), mar=c(5, 5, 1, 1))

for (s in display_subjects) {
  if (s == 13) {
    plot.new()
    next
  }

  names <- sapply(which(data$subject == s),
                  function(n)
                  paste0('log_reading_time_pred[', n, ']'))
  filtered_samples <- util$filter_expectands(samples_dat2, names)

  obs_reading_time <- data$reading_time[data$subject == s]

  util$plot_hist_quantiles(filtered_samples,
                           'log_reading_time_pred',
                            4, 10, 0.25,
                            baseline_values=log(obs_reading_time),
                            xlab='Log Reading Time / 1 ms',
                            main=paste('Subject', s))
```

```
}
```

```
Warning in check_bin_containment(bin_min, bin_max, collapsed_values,
"predictive value"): 1 predictive value (0.0%) fell above the binning.

Warning in check_bin_containment(bin_min, bin_max, collapsed_values,
"predictive value"): 1 predictive value (0.0%) fell above the binning.

Warning in check_bin_containment(bin_min, bin_max, collapsed_values,
"predictive value"): 1 predictive value (0.0%) fell above the binning.


Warning in check_bin_containment(bin_min, bin_max, collapsed_values,
"predictive value"): 3 predictive values (0.0%) fell above the binning.


Warning in check_bin_containment(bin_min, bin_max, collapsed_values,
"predictive value"): 5 predictive values (0.0%) fell above the binning.


Warning in check_bin_containment(bin_min, bin_max, collapsed_values,
"predictive value"): 2 predictive values (0.0%) fell above the binning.


Warning in check_bin_containment(bin_min, bin_max, collapsed_values,
"predictive value"): 1 predictive value (0.0%) fell above the binning.
```

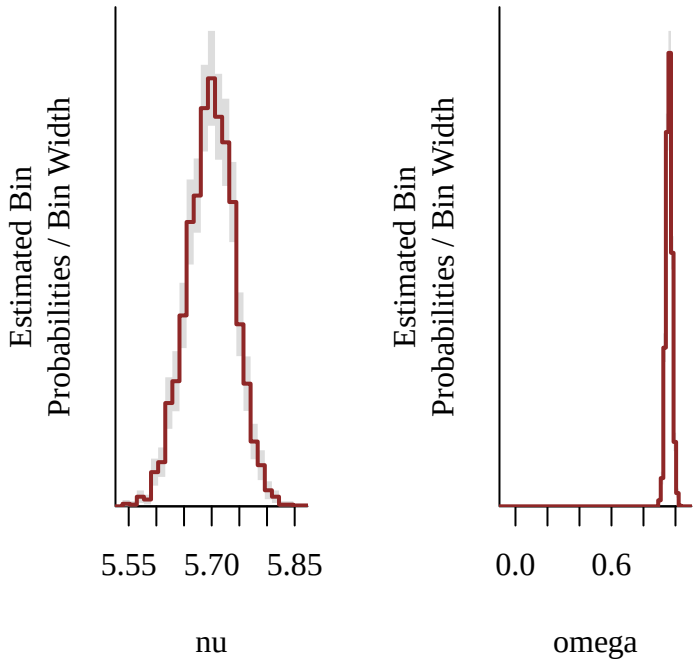


4.4.4 Posterior Inferences

After all of this work, we finally have a model with no apparent retrodictive tension to suggest model inadequacy. This doesn't imply that our model is "true" in any real sense. Indeed if we collected more data then we would likely be able to resolve even more detailed behaviors that could invalidate this relatively simple model. Our posterior inferences, however, will accurately describe the latent data generating process as well as we can resolve it at the moment.

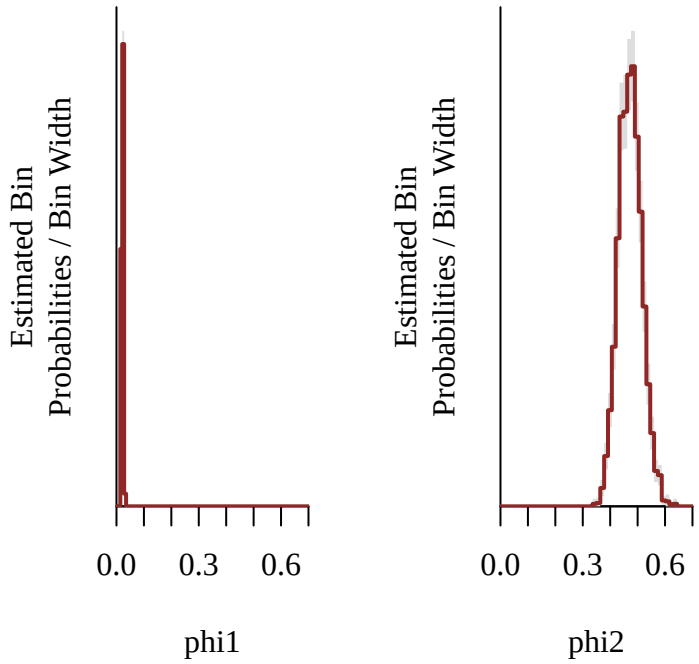
What do those posterior inferences have to say? Firstly, we can see that the reading times after a failed initial hypothesis are clearly larger than those after a successful initial hypothesis.

```
par(mfrow=c(1, 2), mar = c(5, 4, 2, 1))

util$plot_expectand_pushforward(samples_dat2[["nu"]], 25,
                                display_name="nu")

util$plot_expectand_pushforward(samples_dat2[["omega"]],
                                75, flim=c(-0.1 ,1.1),
                                display_name="omega")
```



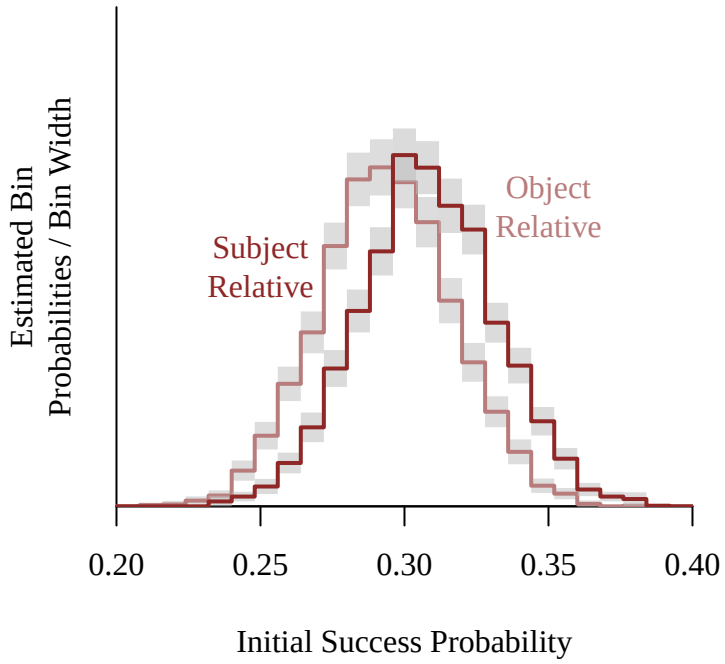
Moreover, the initial failure component model is more diffuse than the initial success component model.

```
par(mfrow=c(1, 2), mar = c(5, 4, 2, 1))

util$plot_expectand_pushforward(samples_dat2[["phi1"]],
                                100, flim=c(0, 0.7),
                                display_name="phi1")

util$plot_expectand_pushforward(samples_dat2[["phi2"]],
                                50, flim=c(0, 0.7),
                                display_name="phi2")
```



Sentences with an object-extracted relative clause favor the faster reading times of the initial success component model more than the sentences with a subject-extracted relative clause. This suggests that subject-extracted relative clauses take more attempts to correctly parse. At least, that is, for Chinese.

```
par(mfrow=c(1, 1), mar = c(5, 4, 2, 1))

util$plot_expectand_pushforward(samples_dat2[['lambda_SR']], 25,
                                flim=c(0.2, 0.4),
                                ylim=c(0, 25),
                                display_name="Initial Success Probability",
                                col=util$c_mid)
text(0.35, 15, 'Object\nRelative', col=util$c_mid)

util$plot_expectand_pushforward(samples_dat2[['lambda_OR']], 25,
                                flim=c(0.2, 0.4),
                                col=util$c_dark,
                                border="#BBBBBB88",
                                add=TRUE)
text(0.25, 12, 'Subject\nRelative', col=util$c_dark)
```



We can even quantify the preference for the first component model by computing the posterior probability that

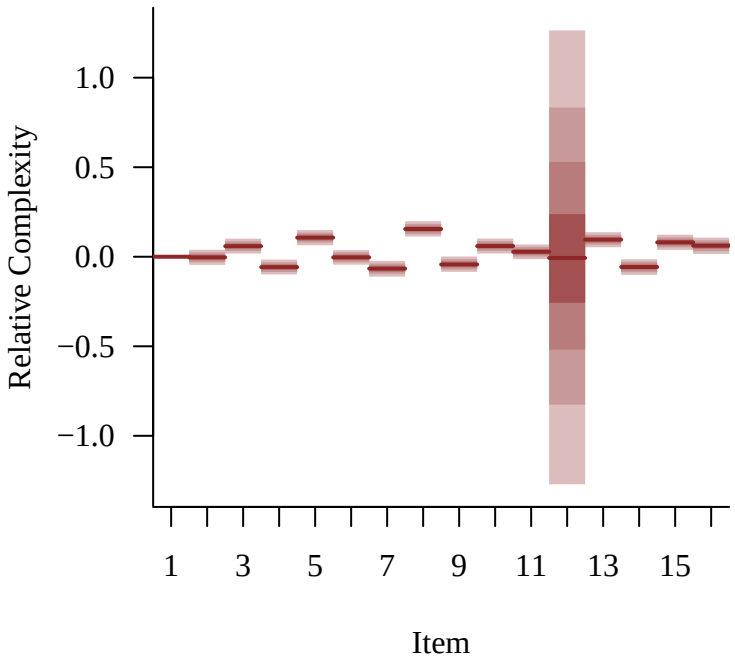
$$\lambda_{\text{SR}} < \lambda_{\text{OR}}.$$

```
var_repl <- list('l1' = 'lambda_SR',
                 'l2' = 'lambda_OR')

p_est <- util$implicit_subset_prob(samples_dat2,
                                   function(l1, l2) l1 < l2,
                                   var_repl)

format_string <- paste0("Posterior probability that lambda_SR ",
                        "< lambda_OR = %.3f +/- %.3f.")
cat(sprintf(format_string, p_est[1], 2 * p_est[2]))
```

```
Posterior probability that lambda_SR < lambda_OR = 0.741 +/- 0.014.
```

Finally, we can examine the inferred behavior of the individual items and subjects. Let's start with the relative complexity of the items.

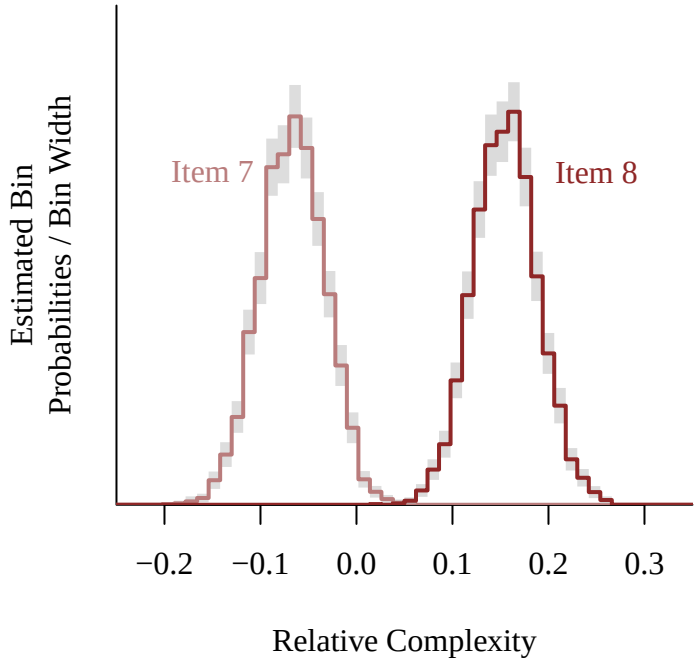
```
par(mfrow=c(1, 1), mar = c(5, 4, 2, 1))

names <- sapply(1:data$N_items,
                function(i) paste0('delta[', i, ']'))
util$plot_disc_pushforward_quantiles(samples_dat2, names,
                                     xlab="Item",
                                     xticklabs=c(1:data$N_items),
                                     ylab="Relative Complexity")
```



Because Item 12 did not appear in any observations its inferences are determined entirely by the prior model, resulting in much stronger posterior uncertainties. Otherwise Item 8 appears to be the most complex question with Item 7 the simplest.

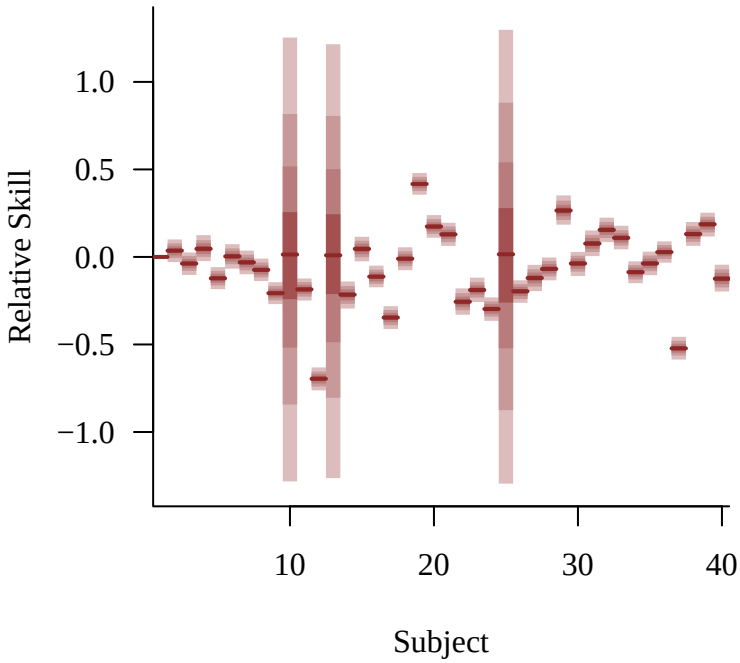
```
par(mfrow=c(1, 1), mar = c(5, 4, 2, 1))

util$plot_expectand_pushforward(samples_dat2[['delta[7]']],
                                50, flim=c(-0.25, 0.35),
                                ylim=c(0, 15),
                                display_name="Relative Complexity",
                                col=util$c_mid)
text(-0.15, 10, 'Item 7', col=util$c_mid)

util$plot_expectand_pushforward(samples_dat2[['delta[8]']],
```

```
                                    50, flim=c(-0.25, 0.35),
                                    col=util$c_dark,
                                    border="#BBBBBB88",
                                    add=TRUE)
text(0.25, 10, 'Item 8', col=util$c_dark)
```



What about the relative subject skills?

```
par(mfrow=c(1, 1), mar = c(5, 4, 2, 1))

names <- sapply(1:data$N_subjects,
                function(s) paste0('zeta[', s, ']'))
util$plot_disc_pushforward_quantiles(samples_dat2, names,
                                     xlab="Subject",
                                     ylab="Relative Skill")
```

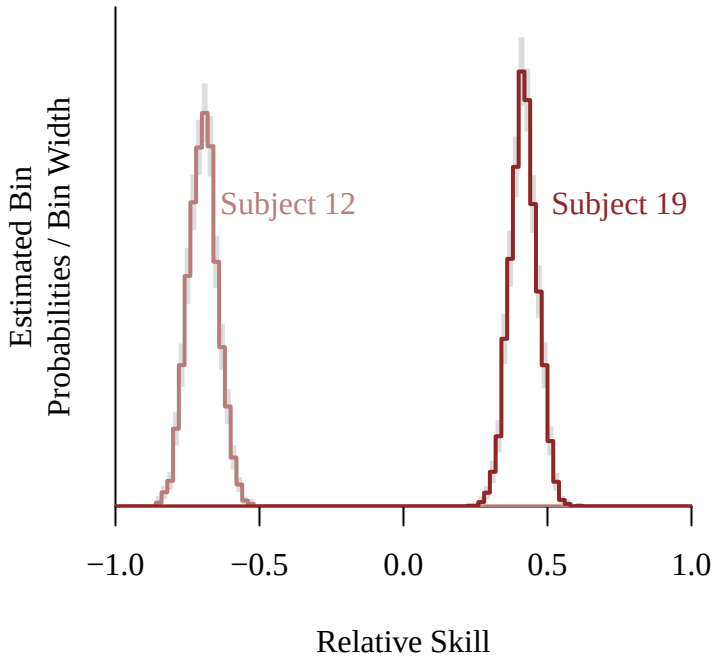


The lack of any observations involving Subjects 10, 12, and 25 once again manifests in larger posterior uncertainties. For those subjects whose reading times were recorded, Subject 19 seems to be about

$$\exp(1.25) \approx 3.5$$

times better at parsing the target texts than Subject 12.

```
par(mfrow=c(1, 1), mar = c(5, 4, 2, 1))

util$plot_expectand_pushforward(samples_dat2[['zeta[12]']],
                                100, flim=c(-1, 1),
                                ylim=c(0, 10),
                                display_name="Relative Skill",
                                col=util$c_mid)
text(-0.4, 6, 'Subject 12', col=util$c_mid)

util$plot_expectand_pushforward(samples_dat2[['zeta[19]']],
                                100, flim=c(-1, 1),
                                col=util$c_dark,
                                border="#BBBBBB88",
                                add=TRUE)
text(0.75, 6, 'Subject 19', col=util$c_dark)
```



4.5 Joint Dependency Locality and Direct-Access Model

We can from their respective posterior retrodictive behaviors that the dependency locality model is less consistent with the observed reading times than the direct-access model. That said, this comparison is a mostly qualitative one. Many are satisfied by only more quantitative comparisons.

There is no end of **predictive performance scores** that claim to be able to quantify how much better of a fit one model is than other. Unfortunately the theoretical construction and practical implementation of these scores inherently results in fragile, if not entirely untrustworthy, results. At least, that is, in the opinion of this author. For a more detailed discussion of posterior predictive performance scores and their limitations see Sections 1.4.1 and 1.4.2 of my workflow chapter, as well as the references therein.

Quantitative comparisons, however, become ordinary posterior inferences if we can integrate both of these models into a single joint model. As we saw in Section 1.2.4, this integration is straightforward for the dependency locality and direct-access models.

4.5.1 Observational Model

To combine these two models we start with the direct-access mixture model but then allow for clausal shifts in the location variables,

$$\lambda_{\text{SR}}\, p_1(t_{ij} \mid \mu^{\text{SR}}_{1,ij}, \phi_1) + (1 - \lambda_{\text{SR}})\, p_2(t_{ij} \mid \mu^{\text{SR}}_{2,ij}, \phi_2)$$

and

$$\lambda_{\text{OR}}\, p_1(t_{ij} \mid \mu^{\text{OR}}_{1,ij}, \phi_1) + (1 - \lambda_{\text{OR}})\, p_2(t_{ij} \mid \mu^{\text{OR}}_{2,ij}, \phi_2).$$

with

$$\begin{aligned}
\mu^{\text{SR}}_{1,ij} &= \exp\left(\eta + \gamma^{\text{SR}} + \alpha_i - \beta_j\right) \\
\mu^{\text{SR}}_{2,ij} &= \exp\left(\eta + \gamma^{\text{SR}} + \omega + \alpha_i - \beta_j\right).
\end{aligned}$$

and

$$\begin{aligned}
\mu^{\text{OR}}_{1,ij} &= \exp\left(\eta + \gamma^{\text{OR}} + \alpha_i - \beta_j\right) \\
\mu^{\text{OR}}_{2,ij} &= \exp\left(\eta + \gamma^{\text{OR}} + \omega + \alpha_i - \beta_j\right).
\end{aligned}$$

If

$$\lambda_{\text{SR}} = \lambda_{\text{OR}} = 1$$

then the joint model reduces to the dependency locality model. Similarly if

$$\gamma^{\text{SR}} = \gamma^{\text{OR}} = \gamma$$

then the joint model effectively reduces to the direct-access model provided we absorb γ into η.

Based on our previous results we'll jump past log normal component models to inverse gamma component models.

4.5.2 Anchoring

At this point anchoring is becoming more routine. We start by selecting a distinguished anchor item i' to define relative complexities,

$$\delta_i = \alpha_i - \alpha_{i'},$$

and then choosing a distinguished anchor subject j' to define relative skills,

$$\zeta_j = \beta_j - \beta_{j'}.$$

Because there are four different location variables in the joint model, however, our next step will have to be a bit more careful. We start by writing

$$\begin{aligned}
\mu^{\text{SR}}_{1,ij} &= \exp\left(\eta + \gamma^{\text{SR}} + \alpha_i - \beta_j\right) \\
&= \exp\left(\eta + \gamma^{\text{SR}} + \alpha_{i'} - \beta_{j'} + \delta_i - \zeta_j\right) \\
&\equiv \exp\left(\tau + \delta_i - \zeta_j\right)
\end{aligned}$$

and

$$\begin{aligned}\mu_{2,ij}^{\mathrm{SR}} &= \exp\left(\eta + \gamma^{\mathrm{SR}} + \omega + \alpha_i - \beta_j\right) \\ &= \exp\left(\eta + \gamma^{\mathrm{SR}} + \alpha_{i'} - \beta_{j'} + \omega + \delta_i - \zeta_j\right) \\ &= \exp\left(\tau + \omega + \delta_i - \zeta_j\right).\end{aligned}$$

Then in contrast we can write

$$\begin{aligned}\mu_{1,ij}^{\mathrm{OR}} &= \exp\left(\eta + \gamma^{\mathrm{OR}} + \alpha_i - \beta_j\right) \\ &= \exp\left(\eta + \gamma^{\mathrm{SR}} + \alpha_{i'} - \beta_{j'} + \gamma^{\mathrm{OR}} - \gamma^{\mathrm{SR}} + \delta_i - \zeta_j\right) \\ &\equiv \exp\left(\tau + \chi + \delta_i - \zeta_j\right)\end{aligned}$$

and

$$\begin{aligned}\mu_{2,ij}^{\mathrm{OR}} &= \exp\left(\eta + \gamma^{\mathrm{OR}} + \omega + \alpha_i - \beta_j\right) \\ &= \exp\left(\eta + \gamma^{\mathrm{SR}} + \alpha_{i'} - \beta_{j'} + \gamma^{\mathrm{OR}} - \gamma^{\mathrm{SR}} + \omega + \delta_i - \zeta_j\right) \\ &= \exp\left(\tau + \chi + \omega + \delta_i - \zeta_j\right),\end{aligned}$$

The new baseline

$$\tau = \eta + \gamma^{\mathrm{SR}} + \alpha_{i'} - \beta_{j'}.$$

quantifies the nominal behavior of a successful initial hypothesis parsing subject-extracted relative clauses. The deviations

$$\chi = \gamma^{\mathrm{OR}} - \gamma^{\mathrm{SR}}$$

and ω are the same parameters that appear in the dependency locality and direct-access models, respectively.

4.5.3 Prior Model

Except for τ, all of the parameters in the joint model directly correspond to parameters in either the dependency locality model or the direct-access model. Consequently we can use the same component prior models that we had used previously when building a prior for the joint model.

Because we used the same component prior models for the baselines κ and ν in those two models, it's also reasonable to continue that choice for τ. We're effectively assuming that the slight difference in these quantities is negligible compared to the uncertainties in our domain expertise elicitation. Again, more precise domain expertise is always welcome.

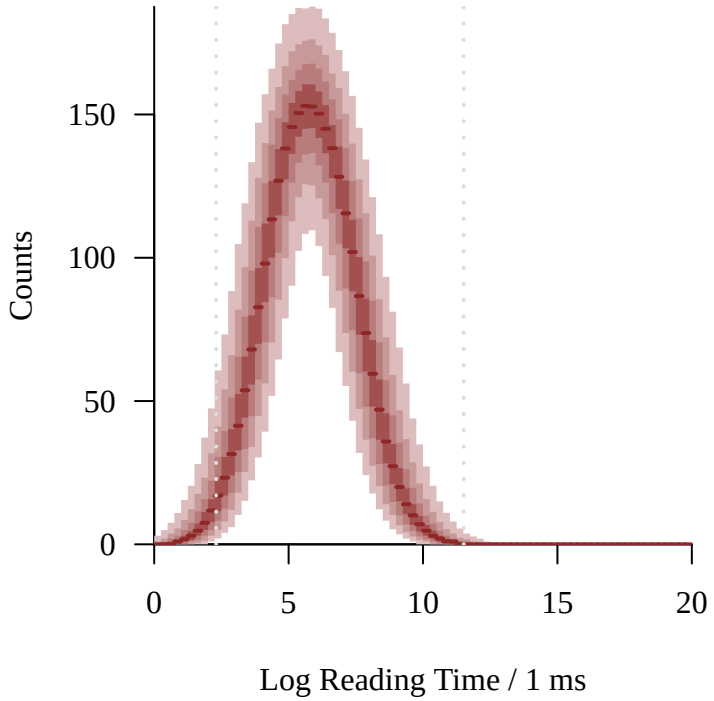
To be diligent we should check the prior predictive behavior of this new joint model. Fortunately, we don't see any new awkward behaviors that clash with our domain expertise.

```
fit <- stan(file="stan_programs/joint_prior.stan",
            data=data, seed=8438338, algorithm="Fixed_param",
            warmup=0, iter=1024, refresh=0)

samples <- util$extract_expectand_vals(fit)
```

```
par(mfrow=c(1, 1), mar=c(5, 5, 1, 1))

util$plot_hist_quantiles(samples, 'log_reading_time_pred',
                         0, 20, 0.25,
                         xlab='Log Reading Time / 1 ms')
```

```
Warning in check_bin_containment(bin_min, bin_max, collapsed_values,
"predictive value"): 8270 predictive values (0.1%) fell below the binning.
```

```
abline(v=log(1e1), lwd=2, lty=3, col="#DDDDDD")
abline(v=log(1e5), lwd=2, lty=3, col="#DDDDDD")
```



4.5.4 Posterior Quantification

Now we hope Hamiltonian Monte Carlo is up to the task of exploring the posterior distribution derived from this joint model.

```
fit <- stan(file='stan_programs/joint.stan',
            data=data, seed=8438338,
            warmup=1000, iter=2024, refresh=0)
```

Well the diagnostics are almost completely quiet. This suggests that the exploration of the Markov chains was exhaustive.

```
diagnostics <- util$extract_hmc_diagnostics(fit)
util$check_all_hmc_diagnostics(diagnostics)
```

```
  All Hamiltonian Monte Carlo diagnostics are consistent with reliable
Markov chain Monte Carlo.
```

```
samples_joint <- util$extract_expectand_vals(fit)
base_samples <- util$filter_expectands(samples_joint,
                                       c('tau',
                                         'delta_free',
                                         'zeta_free',
                                         'omega', 'chi',
                                         'phi1', 'phi2',
                                         'lambda_SR', 'lambda_OR'),
                                       TRUE)
util$check_all_expectand_diagnostics(base_samples)
```

```
tau:
  Chain 1: hat{ESS} (87.566) is smaller than desired (100).
  Chain 4: hat{ESS} (99.453) is smaller than desired (100).

zeta_free[2]:
  Chain 2: hat{ESS} (88.870) is smaller than desired (100).

zeta_free[8]:
  Chain 2: hat{ESS} (97.598) is smaller than desired (100).

zeta_free[10]:
  Chain 2: hat{ESS} (86.756) is smaller than desired (100).
```

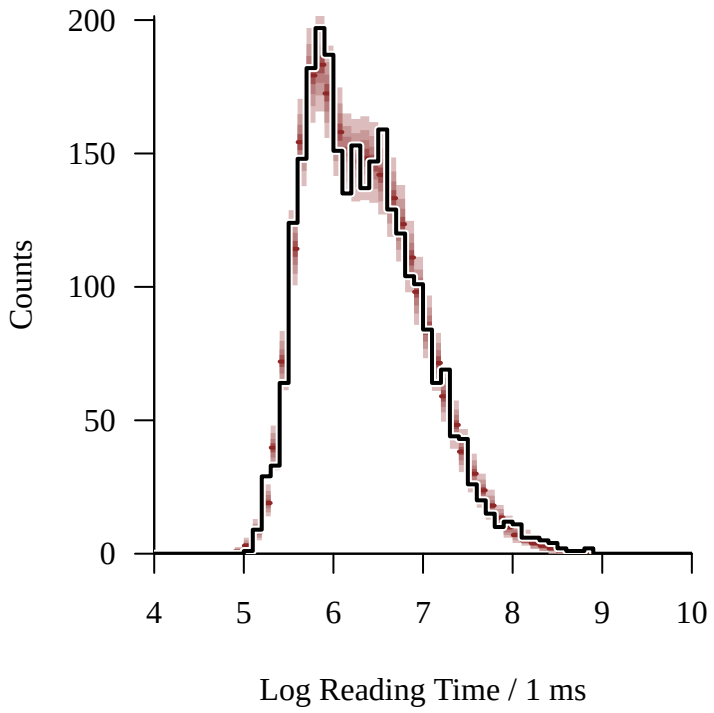
```
zeta_free[14]:
  Chain 2: hat{ESS} (94.936) is smaller than desired (100).

zeta_free[18]:
  Chain 2: hat{ESS} (94.783) is smaller than desired (100).

zeta_free[32]:
  Chain 2: hat{ESS} (92.060) is smaller than desired (100).

zeta_free[33]:
  Chain 2: hat{ESS} (92.746) is smaller than desired (100).

zeta_free[34]:
  Chain 2: hat{ESS} (98.846) is smaller than desired (100).


Small empirical effective sample sizes result in imprecise Markov chain
Monte Carlo estimators.
```

4.5.5 Retrodictive Checks

Because the joint model is a strict generalization of the direct-access model, we should be able to achieve the same posterior retrodictive performance. Indeed the retrodictive checks are similarly gorgeous for all of the summary statistics that we have considered.

```
par(mfrow=c(1, 1), mar=c(5, 5, 1, 1))

util$plot_hist_quantiles(samples_joint, 'log_reading_time_pred',
                         4, 10, 0.1,
                         baseline_values=log(data$reading_time),
                         xlab='Log Reading Time / 1 ms')
```

```
Warning in check_bin_containment(bin_min, bin_max, collapsed_values,
"predictive value"): 73 predictive values (0.0%) fell above the binning.
```



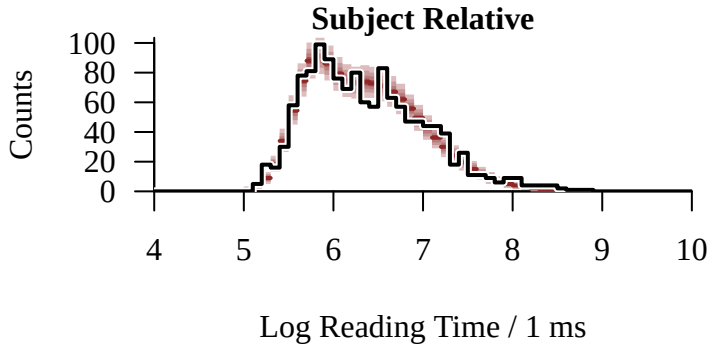

```
par(mfrow=c(2, 1), mar=c(5, 5, 1, 1))

names <- sapply(which(data$subj_rel == 1),
                function(n)
                paste0('log_reading_time_pred[', n, ']'))
filtered_samples <- util$filter_expectands(samples_joint, names)

obs_reading_time <- data$reading_time[data$subj_rel == 1]

util$plot_hist_quantiles(filtered_samples,
                         'log_reading_time_pred',
                         4, 10, 0.1,
                         baseline_values=log(obs_reading_time),
                         xlab='Log Reading Time / 1 ms',
                         main='Subject Relative')
```

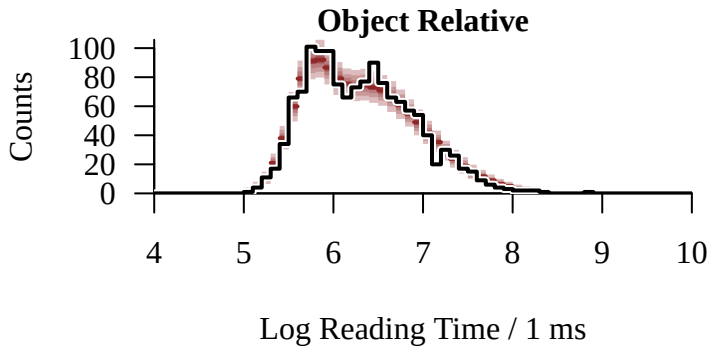
```
Warning in check_bin_containment(bin_min, bin_max, collapsed_values,
"predictive value"): 37 predictive values (0.0%) fell above the binning.
```

```
names <- sapply(which(data$subj_rel == 0),
                function(n)
```

```
                paste0('log_reading_time_pred[', n, ']'))
filtered_samples <- util$filter_expectands(samples_joint, names)

obs_reading_time <- data$reading_time[data$subj_rel == 0]

util$plot_hist_quantiles(filtered_samples,
                         'log_reading_time_pred',
                         4, 10, 0.1,
                         baseline_values=log(obs_reading_time),
                         xlab='Log Reading Time / 1 ms',
                         main='Object Relative')
```

```
Warning in check_bin_containment(bin_min, bin_max, collapsed_values,
"predictive value"): 36 predictive values (0.0%) fell above the binning.
```

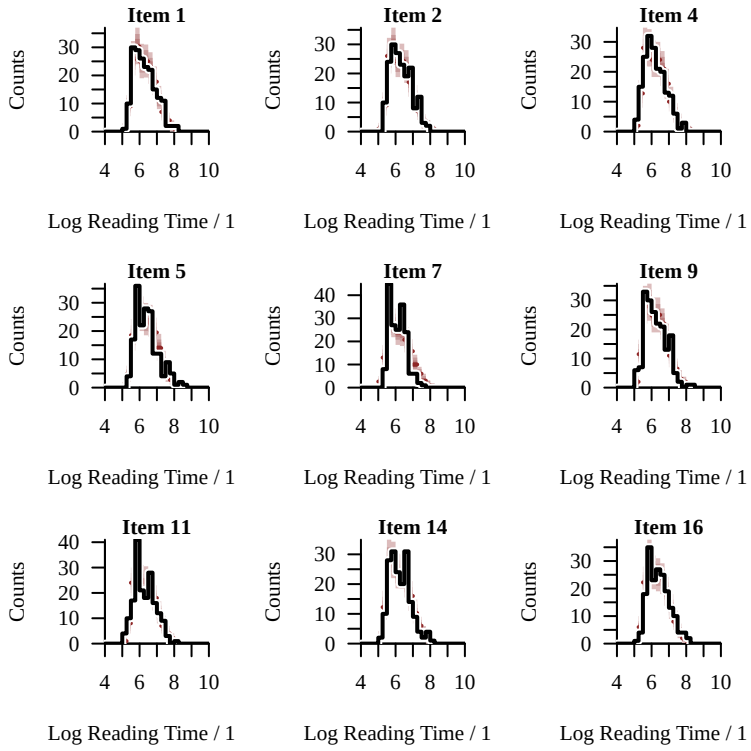





```
par(mfrow=c(3, 3), mar=c(5, 5, 1, 1))

for (i in display_items) {
  names <- sapply(which(data$item == i),
                  function(n)
```

```
                  paste0('log_reading_time_pred[', n, ']'))
  filtered_samples <- util$filter_expectands(samples_joint, names)

  obs_reading_time <- data$reading_time[data$item == i]

  util$plot_hist_quantiles(filtered_samples,
                           'log_reading_time_pred',
                           4, 10, 0.25,
                           baseline_values=log(obs_reading_time),
                           xlab='Log Reading Time / 1 ms',
                           main=paste('Item', i))
}
```

```
Warning in check_bin_containment(bin_min, bin_max, collapsed_values,
"predictive value"): 3 predictive values (0.0%) fell above the binning.

Warning in check_bin_containment(bin_min, bin_max, collapsed_values,
"predictive value"): 7 predictive values (0.0%) fell above the binning.

Warning in check_bin_containment(bin_min, bin_max, collapsed_values,
"predictive value"): 3 predictive values (0.0%) fell above the binning.

Warning in check_bin_containment(bin_min, bin_max, collapsed_values,
"predictive value"): 4 predictive values (0.0%) fell above the binning.

Warning in check_bin_containment(bin_min, bin_max, collapsed_values,
"predictive value"): 3 predictive values (0.0%) fell above the binning.

Warning in check_bin_containment(bin_min, bin_max, collapsed_values,
"predictive value"): 6 predictive values (0.0%) fell above the binning.

Warning in check_bin_containment(bin_min, bin_max, collapsed_values,
"predictive value"): 6 predictive values (0.0%) fell above the binning.

Warning in check_bin_containment(bin_min, bin_max, collapsed_values,
"predictive value"): 3 predictive values (0.0%) fell above the binning.

Warning in check_bin_containment(bin_min, bin_max, collapsed_values,
"predictive value"): 8 predictive values (0.0%) fell above the binning.
```




```
par(mfrow=c(3, 3), mar=c(5, 5, 1, 1))

for (s in display_subjects) {
  if (s == 13) {
    plot.new()
    next
  }

  names <- sapply(which(data$subject == s),
                  function(n)
                  paste0('log_reading_time_pred[', n, ']'))
  filtered_samples <- util$filter_expectands(samples_joint, names)

  obs_reading_time <- data$reading_time[data$subject == s]

  util$plot_hist_quantiles(filtered_samples,
                           'log_reading_time_pred',
                           4, 10, 0.25,
                           baseline_values=log(obs_reading_time),
                           xlab='Log Reading Time / 1 ms',
                           main=paste('Subject', s))
```

```
}
```

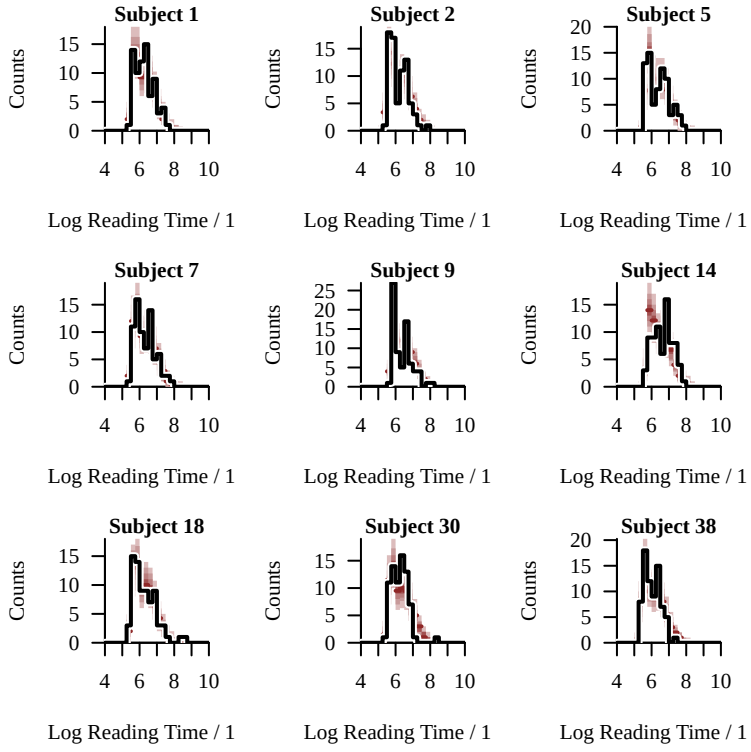
```
Warning in check_bin_containment(bin_min, bin_max, collapsed_values,
"predictive value"): 1 predictive value (0.0%) fell above the binning.
```

```
Warning in check_bin_containment(bin_min, bin_max, collapsed_values,
"predictive value"): 2 predictive values (0.0%) fell above the binning.
```

```
Warning in check_bin_containment(bin_min, bin_max, collapsed_values,
"predictive value"): 3 predictive values (0.0%) fell above the binning.
```

```
Warning in check_bin_containment(bin_min, bin_max, collapsed_values,
"predictive value"): 1 predictive value (0.0%) fell above the binning.
```

```
Warning in check_bin_containment(bin_min, bin_max, collapsed_values,
"predictive value"): 2 predictive values (0.0%) fell above the binning.
```

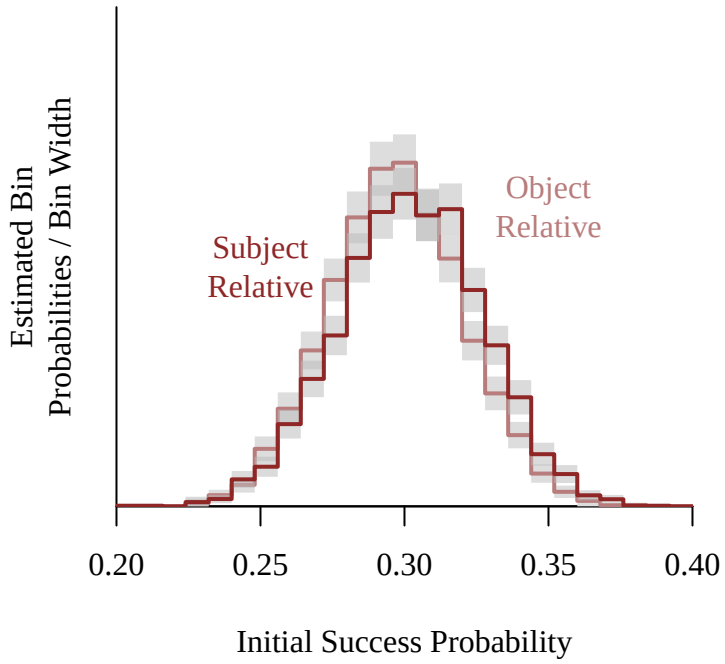


4.5.6 Posterior Inferences

We can now use posterior inferences to quantify the contribution from the dependency locality and direct-access model behaviors.

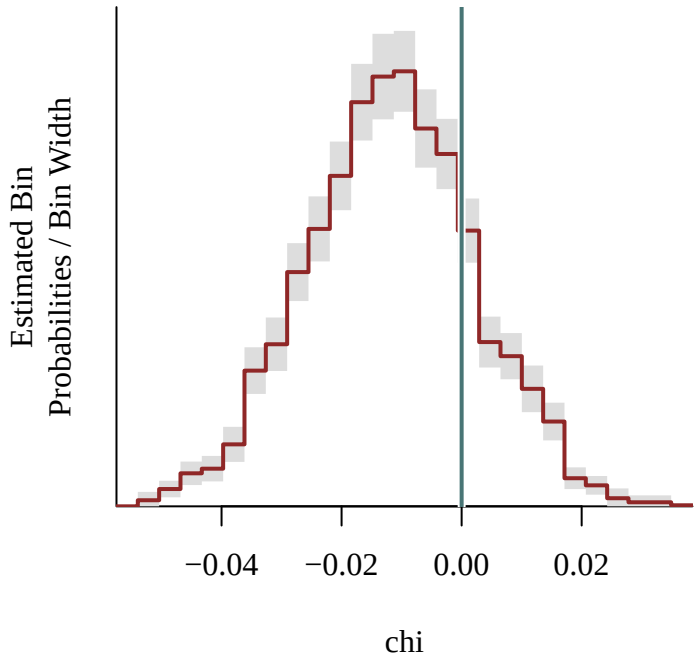
The initial success probabilities for the subject-extracted and object-extracted relative clauses are both far from unity, indicating that the direct-access model is doing a lot of work. While the difference between the two appears to be smaller than what we saw in the inverse gamma direct-access model alone, this is an artifact of slightly larger posterior uncertainties caused by the increased flexibility of the joint model.

```
par(mfrow=c(1, 1), mar = c(5, 4, 2, 1))

util$plot_expectand_pushforward(samples_joint[['lambda_SR']], 25,
                                flim=c(0.2, 0.4),
                                ylim=c(0, 25),
                                display_name="Initial Success Probability",
                                col=util$c_mid)
text(0.35, 15, 'Object\nRelative', col=util$c_mid)

util$plot_expectand_pushforward(samples_joint[['lambda_OR']], 25,
                                flim=c(0.2, 0.4),
                                col=util$c_dark,
                                border="#BBBBBB88",
                                add=TRUE)
text(0.25, 12, 'Subject\nRelative', col=util$c_dark)
```



This doesn't, however, imply that the behaviors from the dependency locality model aren't also contributing. To quantify that we need to look at our posterior inferences for χ.

```
par(mfrow=c(1, 1), mar = c(5, 4, 2, 1))

util$plot_expectand_pushforward(samples_joint[["chi"]], 25,
                                display_name="chi",
                                baseline=0,
                                baseline_col=util$c_mid_teal)
```



Because the marginal posterior distribution for χ strongly concentrates around zero, the contribution from the dependency locality model appears to be negligible. These behaviors just don't help the joint model fit the observed data.

If we can specify thresholds t_l and t_u that quantify what a "non-negligible" contribution is, then we can summarize the suppression of the dependency locality model with the posterior probability that

$$t_l < \chi < t_u.$$

For example if 5% proportional changes to the location behavior are considered negligible for a given application then we could compute

$$\begin{aligned}
\log\left(\frac{1}{1.05}\right) < \chi &< \log(1.05) \\
-\log(1.05) < \chi &< \log(1.05) \\
-\log(1.05) < \chi &< \log(1.05) \\
-0.049 < \chi &< 0.049.
\end{aligned}$$

```
p_est <- util$implicit_subset_prob(samples_joint,
                                   function(x)
                                   -0.049 < x && x < +0.049,
                                   list('x' = 'chi'))

format_string <- paste0("Posterior probability that -0.049 < chi ",
```

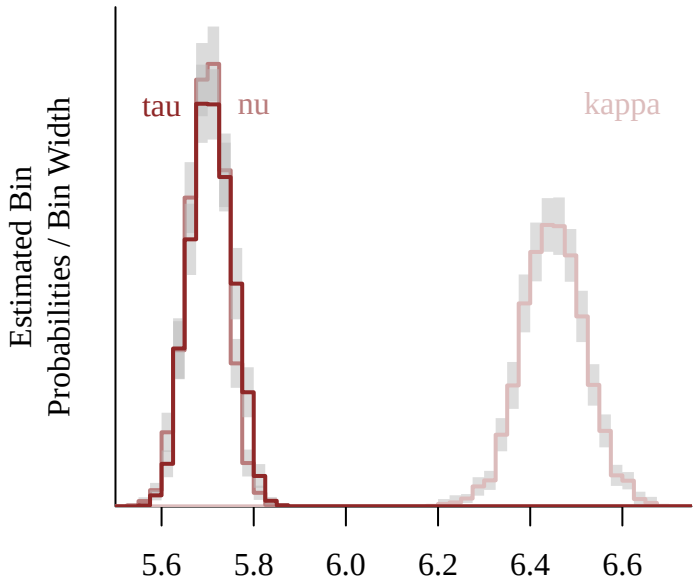
```
                          "< +0.049 = %.3f +/- %.3f.")
cat(sprintf(format_string, p_est[1], 2 * p_est[2]))
```

```
Posterior probability that -0.049 < chi < +0.049 = 0.997 +/- 0.002.
```

In this case, almost *all* of the posterior probability concentrates on negligible model configurations.

Finally the joint baseline τ is consistent with the direct-access baseline ν.

```
par(mfrow=c(1, 1), mar = c(5, 4, 2, 1))

util$plot_expectand_pushforward(samples_dlt2[["kappa"]],
                                50, flim=c(5.5, 6.75),
                                ylim=c(0, 10),
                                display_name="",
                                col=util$c_light)
text(6.6, 8, 'kappa', col=util$c_light)

util$plot_expectand_pushforward(samples_dat2[["nu"]],
                                50, flim=c(5.5, 6.75),
                                col=util$c_mid,
                                border="#BBBBBB88",
                                add=TRUE)
text(5.8, 8, 'nu', col=util$c_mid)

util$plot_expectand_pushforward(samples_joint[["tau"]],
                                50, flim=c(5.5, 6.75),
                                col=util$c_dark,
                                border="#BBBBBB88",
                                add=TRUE)
text(5.6, 8, 'tau', col=util$c_dark)
```



4.6 Modeling Discretization

Once we adopted an inverse gamma direct-access model, the posterior predictive distribution spanned the range of observed behaviors. In particular, the spikes in the observed data that I speculated might be discretization artifacts don't really transcend the posterior predictive uncertainties until we start stratifying the reading times into small groups. Consequently it appears that, in this case, a continuous model is sufficient to learn from the discrete observations.

What would we have done, however, if potential discretization artifacts in the observed data were less amenable with the posterior predictive distribution? In this section we'll review a few techniques for accommodating discrete observations.

4.6.1 Modeling Rounding

One strategy that we might consider is to treat the discretization of the observed reading times as part of the data generating process itself.

Let's say, for example, that continuous reading times are rounded to the nearest integer. More formally, for any integer $k \in \mathbb{Z}$ all continuous reading times in the interval

$$\mathsf{r}_k = [k - 0.5, k + 0.5)$$

are mapped to k.

To model this censoring process we compute the probability of each interval, and hence each integer value, from the latent continuous model,

$$\begin{aligned} q_k &= \pi(\mathsf{r}_k \mid \theta) \\ &= \int_{k-0.5}^{k+0.5} \mathrm{d}t \, p(t \mid \theta). \end{aligned}$$

The observational model for any discrete observation k is then just a categorical model,

$$p(k) = \text{categorical}(k \mid q_1, ...) = q_k.$$

We can use this basic approach to also model reading times that have been rounded down to the next smallest integer or up to the next highest integer. All we have to do is change the continuous interval that maps to each integer reading time. For example, when modeling continuous reading times that are rounded up to the next highest integer we would map all reading times in the interval

$$(k - 1, k]$$

to the integer $k \in \mathbb{Z}$.

These censored models are straightforward to implement if we can directly calculate the interval probabilities. This, in turn, usually requires being able to evaluate all of the relevant cumulative distribution functions.

Consider, for instance, the direct-access model

$$p(t_{ij} \mid \theta)\lambda \, p_1(t_{ij} \mid \mu_{1,ij}, \phi_1) + (1 - \lambda) \, p_2(t_{ij} \mid \mu_{2,ij}, \phi_2)$$

where

$$\theta = (\lambda, \mu_{1,ij}, \phi_1 \mu_{2,ij}, \phi_2).$$

The probability allocated to the interval $[k - 0.5, k + 0.5)$ can be evaluated as

$$\begin{aligned} q_k &= \int_{k-0.5}^{k+0.5} \mathrm{d}t_{ij} \, p(t_{ij} \mid \theta) \\ &= \quad \lambda \int_{k-0.5}^{k+0.5} \mathrm{d}t_{ij} \, p_1(t_{ij} \mid \mu_{1,ij}, \phi_1) \\ &\quad + (1 - \lambda) \int_{k-0.5}^{k+0.5} \mathrm{d}t_{ij} \, p_2(t_{ij} \mid \mu_{2,ij}, \phi_2) \\ &= \qquad \lambda \ \left[\Pi_1(k + 0.5 \mid \mu_{1,ij}, \phi_1) - \Pi_1(k - 0.5 \mid \mu_{1,ij}, \phi_1)\right] \\ &\quad + (1 - \lambda) \left[\Pi_2(k + 0.5 \mid \mu_{2,ij}, \phi_2) - \Pi_2(k - 0.5 \mid \mu_{2,ij}, \phi_2)\right]. \end{aligned}$$

If p_1 and p_2 are both inverse gamma models then Π_1 and Π_2 are given by different configurations of the inverse gamma cumulative distribution function. Conveniently this function is

available in the `Stan` modeling language, making it straightforward to implement the censored models as Stan programs.

The only problem with this approach is that many cumulative distribution functions are expensive to evaluate. Moreover, they are not always the most numerically stable functions. Stan's implementation of the inverse gamma function, for example, can be problematic at larger inputs. The resulting numerical errors then frustrate accurate Hamiltonian Monte Carlo.

To demonstrate the implementation of a censored model we'll instead go back to our first direct-access model. The log normal cumulative distribution function is much more computationally robust.

```
fit <- stan(file='stan_programs/dat1_rounding.stan',
            data=data, seed=8438338,
            warmup=1000, iter=2024, refresh=0)
```

Interestingly, we see similar split $\hat{r}$ diagnostic warnings that we encountered when trying to explore the uncensored log normal direct-access model. In addition a stray divergence has popped up.

```
diagnostics <- util$extract_hmc_diagnostics(fit)
util$check_all_hmc_diagnostics(diagnostics)
```

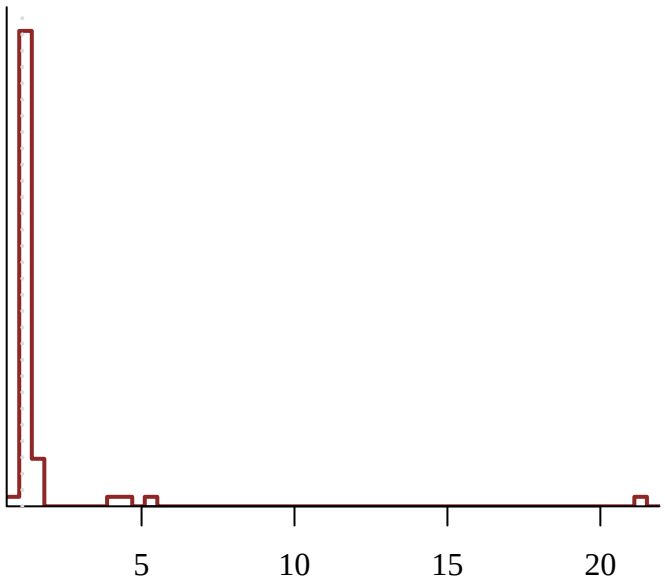
```
  All Hamiltonian Monte Carlo diagnostics are consistent with reliable
Markov chain Monte Carlo.
```

```
samples_dat1r <- util$extract_expectand_vals(fit)
base_samples <- util$filter_expectands(samples_dat1r,
                                       c('nu',
                                         'delta_free',
                                         'zeta_free',
                                         'omega',
                                         'phi1', 'phi2',
                                         'lambda_SR', 'lambda_OR'),
                                       TRUE)
util$summarize_expectand_diagnostics(base_samples)
```

```
The expectands nu, delta_free[6], delta_free[7], delta_free[9],
delta_free[14], delta_free[15], zeta_free[1], zeta_free[2],
zeta_free[3], zeta_free[4], zeta_free[5], zeta_free[6], zeta_free[7],
zeta_free[8], zeta_free[10], zeta_free[11], zeta_free[13],
zeta_free[14], zeta_free[15], zeta_free[16], zeta_free[17],
```

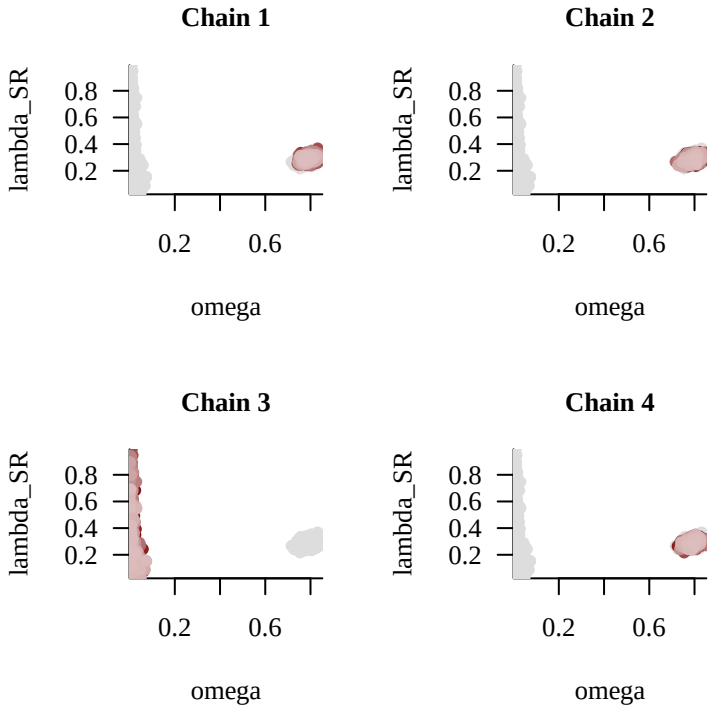
```
zeta_free[18], zeta_free[19], zeta_free[20], zeta_free[22],
zeta_free[23], zeta_free[25], zeta_free[27], zeta_free[29],
zeta_free[30], zeta_free[32], zeta_free[33], zeta_free[34],
zeta_free[35], zeta_free[36], zeta_free[38], zeta_free[39], omega,
phi1, lambda_OR triggered diagnostic warnings.

The expectands nu, delta_free[6], delta_free[7], delta_free[9],
delta_free[14], delta_free[15], zeta_free[5], zeta_free[8],
zeta_free[13], zeta_free[14], zeta_free[16], zeta_free[18],
zeta_free[25], zeta_free[27], zeta_free[30], zeta_free[32],
zeta_free[38], zeta_free[39], omega, phi1, lambda_OR triggered hat{R}
warnings.

Split Rhat larger than 1.1 suggests that at least one of the Markov
chains has not reached an equilibrium.

The expectands nu, zeta_free[1], zeta_free[2], zeta_free[3],
zeta_free[4], zeta_free[6], zeta_free[7], zeta_free[8], zeta_free[10],
zeta_free[11], zeta_free[14], zeta_free[15], zeta_free[16],
zeta_free[17], zeta_free[18], zeta_free[19], zeta_free[20],
zeta_free[22], zeta_free[23], zeta_free[25], zeta_free[27],
zeta_free[29], zeta_free[32], zeta_free[33], zeta_free[34],
zeta_free[35], zeta_free[36], zeta_free[38], zeta_free[39] triggered
hat{ESS} warnings.

Small empirical effective sample sizes result in imprecise Markov chain
Monte Carlo estimators.
```

```
util$plot_rhats(base_samples, B=50)
```



Indeed the posterior multimodality is nearly identical.

```
util$plot_pairs_by_chain(samples_dat1r[['omega']], 'omega',
                         samples_dat1r[['lambda_SR']], 'lambda_SR')
```



Moreover, the posterior retrodictive behavior seems to be unaffected by the rounding.

```
par(mfrow=c(2, 2), mar=c(5, 5, 3, 1))

util$plot_hist_quantiles(lapply(samples_dat1,
                                function(s) s[c(1, 3),]),
                         'log_reading_time_pred', 4, 10, 0.1,
                         baseline_values=log(data$reading_time),
                         xlab='Log Reading Time / 1 ms',
                         main='Markov Chains 1 and 3')
```

```
Warning in check_bin_containment(bin_min, bin_max, collapsed_values,
"predictive value"): 1437 predictive values (0.0%) fell below the binning.
```

```
Warning in check_bin_containment(bin_min, bin_max, collapsed_values,
"predictive value"): 23 predictive values (0.0%) fell above the binning.
```

```
util$plot_hist_quantiles(lapply(samples_dat1,
                                function(s) s[c(2, 4),]),
                         'log_reading_time_pred', 4, 10, 0.1,
                         baseline_values=log(data$reading_time),
```

```
                          xlab='Log Reading Time / 1 ms',
                          main='Markov Chains 2 and 4')
```

```
Warning in check_bin_containment(bin_min, bin_max, collapsed_values,
"predictive value"): 18 predictive values (0.0%) fell below the binning.
```

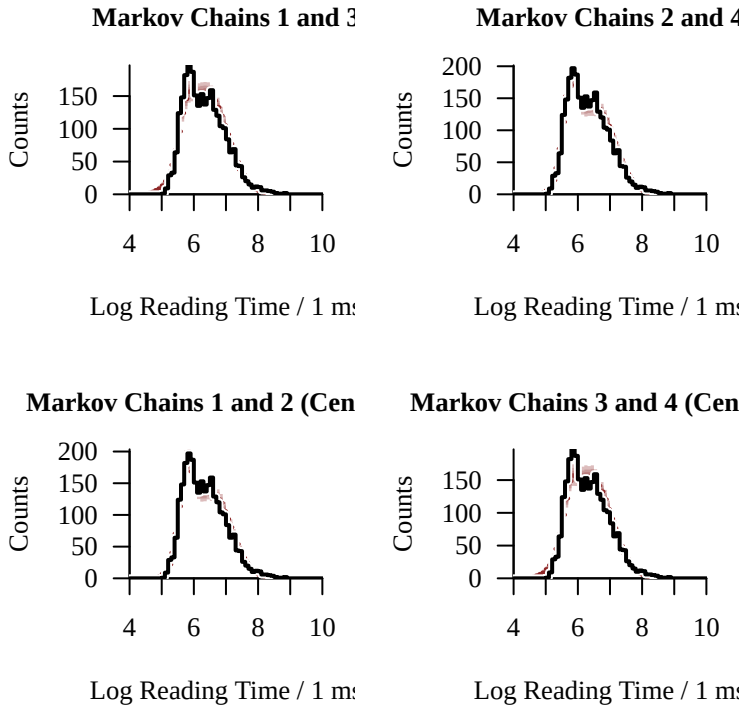
```
util$plot_hist_quantiles(lapply(samples_dat1r,
                                function(s) s[c(1, 2),]),
                         'log_reading_time_pred', 4, 10, 0.1,
                         baseline_values=log(data$reading_time),
                         xlab='Log Reading Time / 1 ms',
                         main='Markov Chains 1 and 2 (Censored)')
```

```
Warning in check_bin_containment(bin_min, bin_max, collapsed_values,
"predictive value"): 21 predictive values (0.0%) fell below the binning.
```

```
util$plot_hist_quantiles(lapply(samples_dat1r,
                                function(s) s[c(3, 4),]),
                         'log_reading_time_pred', 4, 10, 0.1,
                         baseline_values=log(data$reading_time),
                         xlab='Log Reading Time / 1 ms',
                         main='Markov Chains 3 and 4 (Censored)')
```

```
Warning in check_bin_containment(bin_min, bin_max, collapsed_values,
"predictive value"): 1505 predictive values (0.0%) fell below the binning.
```

```
Warning in check_bin_containment(bin_min, bin_max, collapsed_values,
"predictive value"): 18 predictive values (0.0%) fell above the binning.
```



In this case rounding doesn't appreciably change the behavior of the continuous model.

4.6.2 Modeling Discretization Directly

Another approach that we might consider is to replace any family of continuous probability models with a qualitatively similar family of discrete probability models.

For example, in the direct-access model,

$$p(t_{ij} \mid \theta)\lambda\, p_1(t_{ij} \mid \mu_{1,ij}, \phi_1) + (1 - \lambda)\, p_2(t_{ij} \mid \mu_{2,ij}, \phi_2)$$

we might consider using a negative binomial model for p_1 and p_2 instead of an inverse gamma model. Like the inverse gamma model, the negative binomial model skews towards larger values. Unlike the inverse gamma model, however, the negative binomial model is defined over the positive integers.

The only issue with the negative binomial model is that it tends to be implemented differently in every software package. To mimic the location-scale parameterization of the log normal and inverse gamma models we'll need to use

```
neg_binomial_2_lpdf(y | mu, 1 / phi)
```

or, equivalently,

```
neg_binomial_2_log_lpdf(y | log_mu, 1 / phi)
```

in the Stan modeling language. This ensures that $\phi \to 0$ configures the narrowest, most symmetric models while $\phi \to \infty$ configures the widest, most skewed models.

```
fit <- stan(file='stan_programs/dat_nb.stan',
            data=data, seed=8438338,
            warmup=1000, iter=2024, refresh=10)
```

Unfortunately, the computational diagnostics show strong signs of posterior multimodality.

```
diagnostics <- util$extract_hmc_diagnostics(fit)
util$check_all_hmc_diagnostics(diagnostics)
```

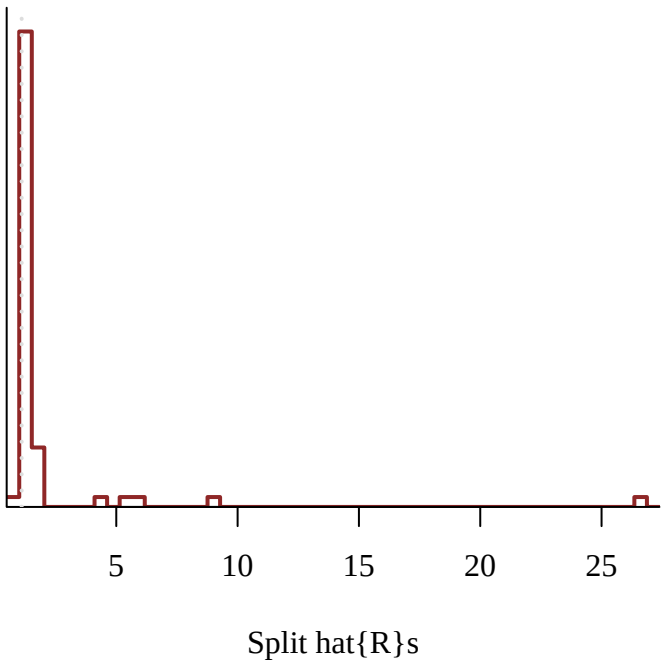
```
  All Hamiltonian Monte Carlo diagnostics are consistent with reliable
Markov chain Monte Carlo.
```

```
samples_dat_nb <- util$extract_expectand_vals(fit)
base_samples <- util$filter_expectands(samples_dat_nb,
                                       c('nu',
                                         'delta_free',
                                         'zeta_free',
                                         'omega',
                                         'phi1', 'phi2',
                                         'lambda_SR',
                                         'lambda_OR'),
                                       TRUE)
util$summarize_expectand_diagnostics(base_samples)
```

```
The expectands nu, delta_free[5], delta_free[6], delta_free[7],
delta_free[9], delta_free[14], delta_free[15], zeta_free[1],
zeta_free[2], zeta_free[4], zeta_free[5], zeta_free[8], zeta_free[13],
zeta_free[14], zeta_free[15], zeta_free[16], zeta_free[18],
zeta_free[19], zeta_free[22], zeta_free[25], zeta_free[26],
zeta_free[27], zeta_free[28], zeta_free[29], zeta_free[30],
zeta_free[31], zeta_free[32], zeta_free[33], zeta_free[37],
zeta_free[39], omega, phi1, lambda_SR, lambda_OR triggered diagnostic
warnings.

The expectands nu, delta_free[5], delta_free[6], delta_free[7],
```

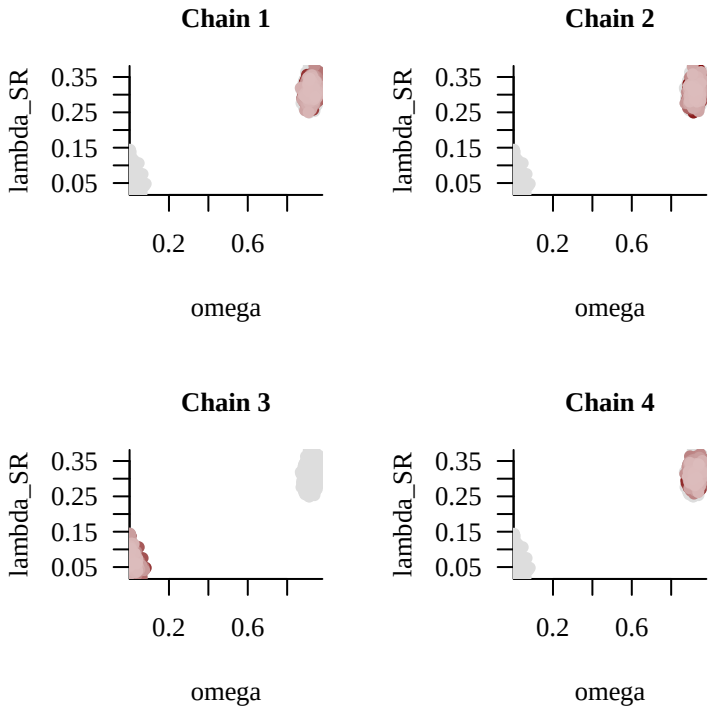
```
delta_free[9], delta_free[14], delta_free[15], zeta_free[1],
zeta_free[2], zeta_free[5], zeta_free[8], zeta_free[13], zeta_free[14],
zeta_free[16], zeta_free[18], zeta_free[22], zeta_free[25],
zeta_free[26], zeta_free[27], zeta_free[28], zeta_free[29],
zeta_free[30], zeta_free[31], zeta_free[32], zeta_free[33],
zeta_free[37], zeta_free[39], omega, phi1, lambda_SR, lambda_OR
triggered hat{R} warnings.

Split Rhat larger than 1.1 suggests that at least one of the Markov
chains has not reached an equilibrium.

The expectands nu, zeta_free[1], zeta_free[2], zeta_free[4],
zeta_free[8], zeta_free[15], zeta_free[18], zeta_free[19],
zeta_free[25], zeta_free[27], zeta_free[32], zeta_free[33] triggered
hat{ESS} warnings.

Small empirical effective sample sizes result in imprecise Markov chain
Monte Carlo estimators.
```

```
util$plot_rhats(base_samples, B=50)
```



The multimodality we see here is very similar to the multimodalities we saw in the uncensored and censored log normal direct-access models. In one of the modes $\omega \to 0$ so that the two component models collapse on top of each other to form a single peak.

```
util$plot_pairs_by_chain(samples_dat_nb[['omega']], 'omega',
                         samples_dat_nb[['lambda_SR']], 'lambda_SR')
```



It appears that the negative binomial model is not as skewed as the inverse gamma model. Instead it's retrodictive performance is more similar to the log normal model.

```
par(mfrow=c(2, 2), mar=c(5, 5, 3, 1))

util$plot_hist_quantiles(lapply(samples_dat1,
                                function(s) s[c(1, 3),]),
                         'log_reading_time_pred', 4, 10, 0.1,
                         baseline_values=log(data$reading_time),
                         xlab='Log Reading Time / 1 ms',
                         main='Markov Chains 1 and 3 (Log Normal)')
```

```
Warning in check_bin_containment(bin_min, bin_max, collapsed_values,
"predictive value"): 1437 predictive values (0.0%) fell below the binning.
```

```
Warning in check_bin_containment(bin_min, bin_max, collapsed_values,
"predictive value"): 23 predictive values (0.0%) fell above the binning.
```

```
util$plot_hist_quantiles(lapply(samples_dat1,
                                function(s) s[c(2, 4),]),
                         'log_reading_time_pred', 4, 10, 0.1,
                         baseline_values=log(data$reading_time),
                         xlab='Log Reading Time / 1 ms',
                         main='Markov Chains 2 and 4 (Log Normal)')
```

```
Warning in check_bin_containment(bin_min, bin_max, collapsed_values,
"predictive value"): 18 predictive values (0.0%) fell below the binning.
```

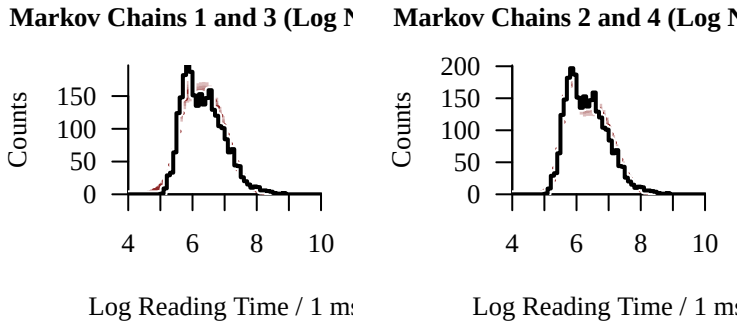
```
util$plot_hist_quantiles(lapply(samples_dat_nb,
                                function(s) s[c(1, 4),]),
                         'log_reading_time_pred', 4, 10, 0.1,
                         baseline_values=log(data$reading_time),
                         xlab='Log Reading Time / 1 ms',
                         main='Markov Chains 1 and 4 (Negative Binomial)')
```

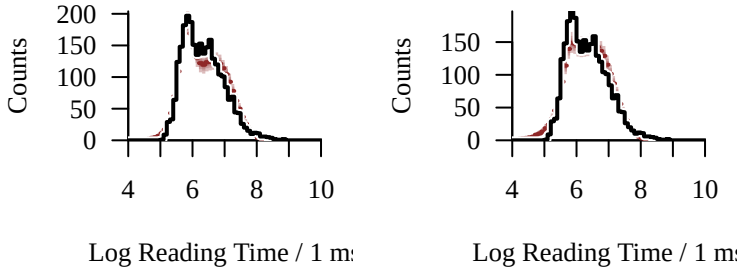
```
Warning in check_bin_containment(bin_min, bin_max, collapsed_values,
"predictive value"): 2649 predictive values (0.0%) fell below the binning.
```

```
util$plot_hist_quantiles(lapply(samples_dat_nb,
                                function(s) s[c(2, 3),]),
                         'log_reading_time_pred', 4, 10, 0.1,
                         baseline_values=log(data$reading_time),
                         xlab='Log Reading Time / 1 ms',
                         main='Markov Chains 2 and 3 (Negative Binomial)')
```

```
Warning in check_bin_containment(bin_min, bin_max, collapsed_values,
"predictive value"): 28651 predictive values (0.5%) fell below the binning.
```

```
Warning in check_bin_containment(bin_min, bin_max, collapsed_values,
"predictive value"): 1 predictive value (0.0%) fell above the binning.
```

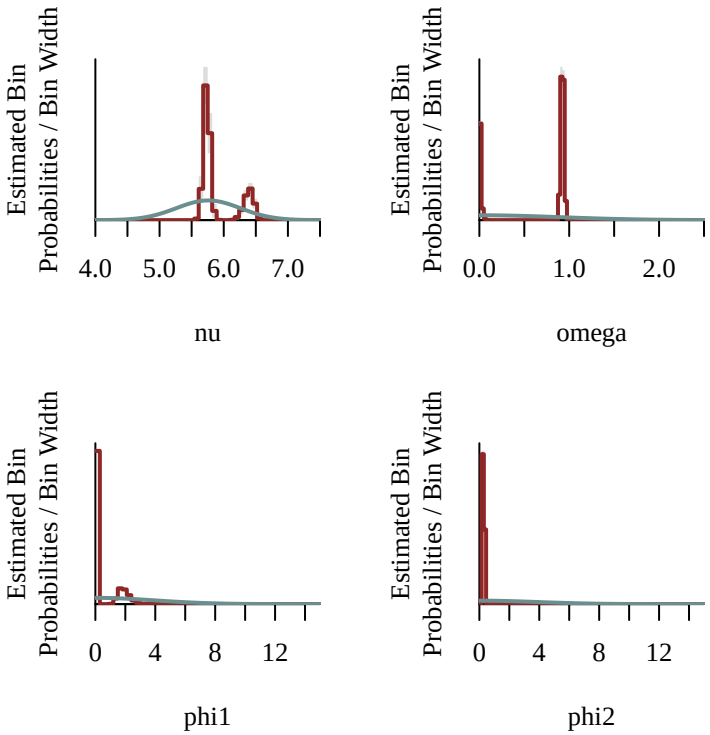




Whenever a model appears to be too rigid to adequately fit the observed data we may want to double check that the prior model isn't excluding useful model configurations. In this case, the marginal posterior distributions all contract well within the scope of the component prior models. Consequently the negative binomial model really does appear to be too rigid.

```
par(mfrow=c(2, 2), mar = c(5, 4, 2, 1))

util$plot_expectand_pushforward(samples_dat_nb[["nu"]],
                                50, flim=c(4, 7.5),
                                display_name="nu")
xs <- seq(4, 7.5, 0.01)
lines(xs, dnorm(xs, 5.76, 0.50),
      lwd=2, col=util$c_light_teal)

util$plot_expectand_pushforward(samples_dat_nb[["omega"]],
                                100, flim=c(0, 2.5),
                                display_name="omega")
xs <- seq(0, 2.5, 0.01)
lines(xs, dnorm(xs, 0, 0.90),
      lwd=2, col=util$c_light_teal)

util$plot_expectand_pushforward(samples_dat_nb[["phi1"]],
```

```
                                  50, flim=c(0, 15),
                                  display_name="phi1")
xs <- seq(0, 15, 0.01)
lines(xs, dnorm(xs, 0, 3.89),
      lwd=2, col=util$c_light_teal)

util$plot_expectand_pushforward(samples_dat_nb[["phi2"]],
                                  100, flim=c(0, 15),
                                  display_name="phi2")
xs <- seq(0, 15, 0.01)
lines(xs, dnorm(xs, 0, 3.89),
      lwd=2, col=util$c_light_teal)
```



Over the decades statisticians have developed a wealth of discrete models that we might consider instead of the negative binomial model. As we move beyond more common models, however, these alternatives become more obscure and require more statistical expertise to identify and then implement robustly.

This leaves us with one last demonstration of the practicalities of probabilistic model building. When our probabilistic vocabulary is limited we can tell only simple stories. Sometimes these stories are sufficient for the analysis at hand, and sometimes they're not. As we become more

fluent with probabilistic modeling techniques, however, we can tell richer stories that allow us to implement increasing more sophisticated analyses.

5 Next Steps

Although we were eventually able to develop an adequate model for the observe reading times, our analysis does not have to end here.

For example the pairwise-comparison structure of our models is ripe for generalization. If we wanted to account for poorly written items, for example, then we could allow for item-specific **discrimination parameters**,

$$\mu_{2,ij} = \exp\left(\eta + \rho_i \cdot (\omega + \alpha_i - \beta_j)\right).$$

The smaller ρ_i is, the less sensitive the reading times of passages based on that template will be to the individual reader skills.

Another potential direction for expanding our model is to couple the individual item complexities and reader skills together into **hierarchical models**. Provided that the individual behaviors are approximately exchangeable, hierarchical models might allow us to extract more precise inferences for the quantities of interest.

The opportunities only increase when we look beyond the reading time data alone. The answers to the comprehension questions, for instance, could be modeled jointly with the reading times. This would allow us to capture how reading time behaviors are coupled to reader comprehension and avoid excluding any subjects from the analysis at all. Sadly the these responses have replaced by a placeholder character in the available data.

```
table(raw_data$correct)
```

```
    -
2735
```

To proceed in this direction we would first need to do a bit of statistical archaeology.

6 Conclusion

When an experiment is carefully designed, the resulting data generating process might be reasonably well-approximated by black-box analysis tools such as regression estimators. We can also derive well-behaved inferences in these ideal settings by modeling the data generating process directly.

More importantly, probabilistic modeling allows us to derive well-behaved inferences when the experimental design or its implementation falters and when the system that we're studying is more complex then we might have initially assumed. In other words, probabilistic modeling in uniquely suited to adapt to meet our scientific goals. This way we don't have to adapt our scientific goals to meet our analysis techniques.

Acknowledgements

I thank Ted Gibson for graciously making the data available and Shravan Vasishth for generously sharing his domain expertise as well as many helpful comments.

A very special thanks to everyone supporting me on Patreon: Adam Fleischhacker, Alejandro Morales, Alessandro Varacca, Alex D, Alexander Noll, Amit, Andrea Serafino, Andrew Mascioli, Andrew Rouillard, Ara Winter, Ari Holtzman, Austin Rochford, Aviv Keshet, Avraham Adler, Ben Matthews, Ben Swallow, Benoit Essiambre, boot, Brendan Galdo, Bryan Chang, Brynjolfur Gauti Jónsson, Cameron Smith, Canaan Breiss, Cat Shark, Cathy Oliveri, Charles Naylor, Chase Dwelle, Chris Jones, Christina Van Heer, Christopher Mehrvarzi, Colin Carroll, Colin McAuliffe, Damien Mannion, dan mackinlay, Dan W Joyce, Dan Waxman, Dan Weitzenfeld, Danny Van Nest, David Burdelski, Doug Rivers, Dr. Jobo, Dr. Omri Har Shemesh, Dylan Maher, Dylan Spielman, Ebriand, Ed Cashin, Edgar Merkle, Edoardo Marcora, Eric LaMotte, Erik Banek, Eugene O'Friel, Felipe González, Fergus Chadwick, Finn Lindgren, Francesco Corona, Geoff Rollins, Granville Matheson, Gregor Gorjanc, Guilherme Marthe, Hamed Bastan-Hagh, haubur, Hector Munoz, Henri Wallen, hs, Hugo Botha, Håkan Johansson, Ian Costley, idontgetoutmuch, Ignacio Vera, Ilaria Prosdocimi, Isaac Vock, Isidor Belic, jacob pine, Jair Andrade, James C, James Hodgson, James Wade, Janek Berger, Jarrett Byrnes, Jason Martin, Jason Pekos, Jason Wong, jd, Jeff Burnett, Jeff Dotson, Jeff Helzner, Jeffrey Erlich, Jerry Lin , Jessica Graves, Joe Sloan, John Flournoy, Jonathan H. Morgan, Josh Knecht, JU, Julian Lee, June, Justin Bois, Karim Naguib, Karim Osman, Konstantin Shakhbazov, Kristian Gårdhus Wichmann, Kádár András, Lars Barquist, lizzie , LOU ODETTE, Luís F, Mads Christian Hansen, Marek Kwiatkowski, Mariana Carmona, Mark Donoghoe, Markus P., Daniel Edward Marthaler, Matthew, Matthew Kay, Matthieu LEROY, Mattia Arsendi, Matěj, Maurits van der Meer, Max, Michael Colaresi, Michael DeWitt, Michael Dillon, Michael Lerner, Mick Cooney, MisterMentat , Márton Vaitkus, N Sanders, N.S. , Nathaniel Burbank, Nicholas Cowie, Nick S, Nikita Karetnikov, Octavio Medina, Ole Rogeberg, Oliver Crook, Patrick Kelley, Patrick Boehnke, Pau Pereira Batlle, Peter Johnson, Pieter van den Berg , ptr, quasar,

Ramiro Barrantes Reynolds, Ravin Kumar, Raúl Peralta Lozada, Rex, Riccardo Fusaroli, Richard Nerland, Robert Frost, Robert Goldman, Robert kohn, Robin Taylor, Ryan Gan, Ryan Grossman, Ryan Kelly, S Hong, Sean Wilson, Sergiy Protsiv, Seth Axen, shira, Simon Duane, Simon Lilburn, Simon Steiger, Simone, Spencer Boucher, sssz, Stefan Lorenz, Stephen Lienhard, Steve Harris, Steven Forrest, Stew Watts, Stone Chen, Susan Holmes, Svilup, Tate Tunstall, Tatsuo Okubo, Teresa Ortiz, Theodore Dasher, Thomas Siegert, Thomas Vladeck, Tobychev , Tomas Capretto, Tony Wuersch, Virgile Andreani, Virginia Fisher, Vitalie Spinu, Vladimir Markov, VO2 Maximus Decius, Will Farr, Will Lowe, Will Wen, woejozney, yolhaj , yureq , Zach A, and Zhengchen Cai.

References

Box, George E. P., J. Stuart Hunter, and William G. Hunter. 2005. *Statistics for Experimenters*. Second. Wiley Series in Probability and Statistics. Wiley-Interscience [John Wiley & Sons], Hoboken, NJ.

Gibson, Edward, and H.-H. Iris Wu. 2013. "Processing Chinese Relative Clauses in Context." *Language and Cognitive Processes* 28 (1-2): 125–55.

Nicenboim, Bruno, Daniel J Schad, and Shravan Vasishth. 2025. *Introduction to Bayesian Data Analysis for Cognitive Science*. 1st ed. Chapman & Hall/CRC Statistics in the Social and Behavioral Sciences. London, England: Chapman; Hall.

Nicenboim, Bruno, and Shravan Vasishth. 2018. "Models of Retrieval in Sentence Comprehension: A Computational Evaluation Using Bayesian Hierarchical Modeling." *Journal of Memory and Language* 99: 1–34.

Vasishth, Shravan, Nicolas Chopin, Robin Ryder, and Bruno Nicenboim. 2017. "Modelling Dependency Completion in Sentence Comprehension as a Bayesian Hierarchical Mixture Process: A Case Study Involving Chinese Relative Clauses," February.

License

A repository containing all of the files used to generate this chapter is available on GitHub.

Original Computing Environment

```
writeLines(readLines(file.path(Sys.getenv("HOME"), ".R/Makevars")))
```

```
CC=clang

CXXFLAGS=-O3 -mtune=native -march=native -Wno-unused-variable -Wno-unused-function -Wno-macr
CXX=clang++ -arch x86_64 -ftemplate-depth-256

CXX14FLAGS=-O3 -mtune=native -march=native -Wno-unused-variable -Wno-unused-function -Wno-ma
CXX14=clang++ -arch x86_64 -ftemplate-depth-256
```

```
sessionInfo()
```

```
R version 4.3.2 (2023-10-31)
Platform: x86_64-apple-darwin20 (64-bit)
Running under: macOS 15.7.3

Matrix products: default
BLAS:   /Library/Frameworks/R.framework/Versions/4.3-x86_64/Resources/lib/libRblas.0.dylib
LAPACK: /Library/Frameworks/R.framework/Versions/4.3-x86_64/Resources/lib/libRlapack.dylib;

locale:
[1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8

time zone: America/New_York
tzcode source: internal

attached base packages:
[1] stats     graphics  grDevices utils     datasets  methods   base

other attached packages:
[1] colormap_0.1.4     rstan_2.32.6       StanHeaders_2.32.7

loaded via a namespace (and not attached):
 [1] gtable_0.3.4       jsonlite_1.8.8     compiler_4.3.2     Rcpp_1.0.11
 [5] stringr_1.5.1      parallel_4.3.2     gridExtra_2.3      scales_1.3.0
 [9] yaml_2.3.8         fastmap_1.1.1      ggplot2_3.4.4      R6_2.6.1
[13] curl_5.2.0         knitr_1.45         tibble_3.2.1       munsell_0.5.0
[17] pillar_1.9.0       rlang_1.1.2        utf8_1.2.4         V8_4.4.1
```

```
[21] stringi_1.8.3      inline_0.3.19      xfun_0.41          RcppParallel_5.1.7
[25] cli_3.6.2          magrittr_2.0.3     digest_0.6.33      grid_4.3.2
[29] lifecycle_1.0.4    vctrs_0.6.5        evaluate_0.23      glue_1.6.2
[33] QuickJSR_1.0.8     codetools_0.2-19   stats4_4.3.2       pkgbuild_1.4.3
[37] fansi_1.0.6        colorspace_2.1-0   rmarkdown_2.25     matrixStats_1.2.0
[41] tools_4.3.2        loo_2.6.0          pkgconfig_2.0.3    htmltools_0.5.7
```

Stan
Program 1 `dlt1\_prior.stan`

```
data {
  int<lower=1> N; // Number of observations

  // Item configuration
  int<lower=1> N_items;
  array[N] int<lower=1, upper=N_items> item;

  // Subject configuration
  int<lower=1> N_subjects;
  array[N] int<lower=1, upper=N_subjects> subject;

  // Item variant
  //   Object Relative:  subj_rel = 0
  //   Subject Relative: subj_rel = 1
  array[N] int<lower=0, upper=1> subj_rel;

  vector<lower=0>[N] reading_time; // Reading times (ms)
}


generated quantities {
  // Log Reading Time Baseline
  real kappa = normal_rng(5.76, 0.50);

  // Relative item difficulties
  vector[N_items - 1] delta_free
    = to_vector(normal_rng(zeros_vector(N_items - 1), 0.99));

  // Relative subject skills
  vector[N_subjects - 1] zeta_free
    = to_vector(normal_rng(zeros_vector(N_subjects - 1), 0.99));

  // Subject Relative Difference
  real chi = normal_rng(0, 0.99);

  // Measurement scale
  real<lower=0> phi = abs(normal_rng(0, 3.89));

  // Relative skills for all items and subjects
  vector[N_items] delta
    = append_row([0]', delta_free);
  vector[N_subjects] zeta
    = append_row([0]', zeta_free);

  array[N] real log_reading_time_pred;

  for (n in 1:N) {
    int i = item[n];
    int s = subject[n];
```

Stan
Program 2 `dlt1.stan`

```
data {
  int<lower=1> N; // Number of observations

  // Item configuration
  int<lower=1> N_items;
  array[N] int<lower=1, upper=N_items> item;

  // Subject configuration
  int<lower=1> N_subjects;
  array[N] int<lower=1, upper=N_subjects> subject;

  // Item variant
  //   Object Relative:  subj_rel = 0
  //   Subject Relative: subj_rel = 1
  array[N] int<lower=0, upper=1> subj_rel;

  vector<lower=0>[N] reading_time; // Reading times (ms)
}

parameters {
  // Log Reading Time Baseline
  real kappa;

  // Relative item difficulties
  vector[N_items - 1] delta_free;

  // Relative subject skills
  vector[N_subjects - 1] zeta_free;

  // Subject Relative Difference
  real chi;

  // Measurement scale
  real<lower=0> phi;
}

transformed parameters {
  // Relative skills for all items and subjects
  vector[N_items] delta
    = append_row([0]', delta_free);
  vector[N_subjects] zeta
    = append_row([0]', zeta_free);
}
```



```
model {
  // Location configurations
  vector[N] log_mu =   kappa
                     + delta[item] - zeta[subject]
                     + (1 - to_vector(subj_rel)) * chi;
```

Stan
Program 3 `dlt2\_prior.stan`

```
functions {
  // Mean-dispersion parameterization of inverse gamma family
  real inv_gamma_md_lpdf(real x, real log_mu, real psi) {
    return inv_gamma_lpdf(x | inv(psi) + 2,
                              exp(log_mu) * (inv(psi) + 1));
  }

  real inv_gamma_md_rng(real log_mu, real psi) {
    return inv_gamma_rng(inv(psi) + 2,
                         exp(log_mu) * (inv(psi) + 1));
  }
}

data {
  int<lower=1> N; // Number of observations

  // Item configuration
  int<lower=1> N_items;
  array[N] int<lower=1, upper=N_items> item;

  // Subject configuration
  int<lower=1> N_subjects;
  array[N] int<lower=1, upper=N_subjects> subject;

  // Item variant
  //   Object Relative:  subj_rel = 0
  //   Subject Relative: subj_rel = 1
  array[N] int<lower=0, upper=1> subj_rel;

  vector<lower=0>[N] reading_time; // Reading times (ms)
}

generated quantities {
  // Log Reading Time Baseline
  real kappa = normal_rng(5.76, 0.50);

  // Relative item difficulties
  vector[N_items - 1] delta_free
    = to_vector(normal_rng(zeros_vector(N_items - 1), 0.99));

  // Relative subject skills
  vector[N_subjects - 1] zeta_free
    = to_vector(normal_rng(zeros_vector(N_subjects - 1), 0.99));

  // Subject Relative Difference
  real chi = normal_rng(0, 0.99);

  // Measurement scale
  real<lower=0> phi = abs(normal_rng(0, 3.89));
```

Stan
Program 4 `dlt2.stan`

```
functions {
  // Mean-dispersion parameterization of inverse gamma family
  real inv_gamma_md_lpdf(real x, real log_mu, real psi) {
    return inv_gamma_lpdf(x | inv(psi) + 2,
                              exp(log_mu) * (inv(psi) + 1));
  }

  real inv_gamma_md_rng(real log_mu, real psi) {
    return inv_gamma_rng(inv(psi) + 2,
                         exp(log_mu) * (inv(psi) + 1));
  }
}

data {
  int<lower=1> N; // Number of observations

  // Item configuration
  int<lower=1> N_items;
  array[N] int<lower=1, upper=N_items> item;

  // Subject configuration
  int<lower=1> N_subjects;
  array[N] int<lower=1, upper=N_subjects> subject;

  // Item variant
  //   Object Relative:  subj_rel = 0
  //   Subject Relative: subj_rel = 1
  array[N] int<lower=0, upper=1> subj_rel;

  vector<lower=0>[N] reading_time; // Reading times (ms)
}

parameters {
  // Log Reading Time Baseline
  real kappa;

  // Relative item difficulties
  vector[N_items - 1] delta_free;

  // Relative subject skills
  vector[N_subjects - 1] zeta_free;

  // Subject Relative Difference
  real chi;

  // Measurement scale
  real<lower=0> phi;
}

transformed parameters {
```

Stan
Program 5 `dat1\_prior.stan`

```
data {
  int<lower=1> N; // Number of observations

  // Item configuration
  int<lower=1> N_items;
  array[N] int<lower=1, upper=N_items> item;

  // Subject configuration
  int<lower=1> N_subjects;
  array[N] int<lower=1, upper=N_subjects> subject;

  // Item variant
  //   Object Relative:  subj_rel = 0
  //   Subject Relative: subj_rel = 1
  array[N] int<lower=0, upper=1> subj_rel;

  vector<lower=0>[N] reading_time; // Reading times (ms)
}

generated quantities {
  // Log reading time baseline
  real nu = normal_rng(5.76, 0.50);

  // Relative item difficulties
  vector[N_items - 1] delta_free
    = to_vector(normal_rng(zeros_vector(N_items - 1), 0.99));

  // Relative subject skills
  vector[N_subjects - 1] zeta_free
    = to_vector(normal_rng(zeros_vector(N_subjects - 1), 0.99));

  // Initial failure difference
  real<lower=0> omega = abs(normal_rng(0, 0.90));

  // Measurement scales
  real<lower=0> phi1 = abs(normal_rng(0, 3.89));
  real<lower=0> phi2 = abs(normal_rng(0, 3.89));

  // Initial failure probabilities
  real<lower=0, upper=1> lambda_SR = beta_rng(1, 1);
  real<lower=0, upper=1> lambda_OR = beta_rng(1, 1);

  // Relative skills for all items and subjects
  vector[N_items] delta
    = append_row([0]', delta_free);
  vector[N_subjects] zeta
    = append_row([0]', zeta_free);

  array[N] real log_reading_time_pred;
```

Stan
Program 6 `dat1.stan`

```
data {
  int<lower=1> N; // Number of observations

  // Item configuration
  int<lower=1> N_items;
  array[N] int<lower=1, upper=N_items> item;

  // Subject configuration
  int<lower=1> N_subjects;
  array[N] int<lower=1, upper=N_subjects> subject;

  // Item variant
  //   Object Relative:  subj_rel = 0
  //   Subject Relative: subj_rel = 1
  array[N] int<lower=0, upper=1> subj_rel;

  vector<lower=0>[N] reading_time; // Reading times (ms)
}

parameters {
  // Log reading time baseline for successful retrieval
  real nu;

  // Relative item difficulties
  vector[N_items - 1] delta_free;

  // Relative subject skills
  vector[N_subjects - 1] zeta_free;

  // Log offset for initial failure before successful retrieval
  real<lower=0> omega;

  // Measurement scales
  real<lower=0> phi1;
  real<lower=0> phi2;

  // Mixture probabilities
  real<lower=0, upper=1> lambda_SR;
  real<lower=0, upper=1> lambda_OR;
}

transformed parameters {
  // Relative skills for all items and subjects
  vector[N_items] delta
    = append_row([0]', delta_free);
  vector[N_subjects] zeta
    = append_row([0]', zeta_free);
}

model {
```

Stan
Program 7 `dat2\_prior.stan`

```
data {
  int<lower=1> N; // Number of observations

  // Item configuration
  int<lower=1> N_items;
  array[N] int<lower=1, upper=N_items> item;

  // Subject configuration
  int<lower=1> N_subjects;
  array[N] int<lower=1, upper=N_subjects> subject;

  // Item variant
  //   Object Relative:  subj_rel = 0
  //   Subject Relative: subj_rel = 1
  array[N] int<lower=0, upper=1> subj_rel;

  vector<lower=0>[N] reading_time; // Reading times (ms)
}

generated quantities {
  // Log reading time baseline
  real nu = normal_rng(5.76, 0.50);

  // Relative item difficulties
  vector[N_items - 1] delta_free
    = to_vector(normal_rng(zeros_vector(N_items - 1), 0.99));

  // Relative subject skills
  vector[N_subjects - 1] zeta_free
    = to_vector(normal_rng(zeros_vector(N_subjects - 1), 0.99));

  // Initial failure difference
  real<lower=0> omega = abs(normal_rng(0, 0.90));

  // Measurement scales
  real<lower=0> phi1 = abs(normal_rng(0, 3.89));
  real<lower=0> phi2 = abs(normal_rng(0, 3.89));

  // Initial failure probabilities
  real<lower=0, upper=1> lambda_SR = beta_rng(1, 1);
  real<lower=0, upper=1> lambda_OR = beta_rng(1, 1);

  // Relative skills for all items and subjects
  vector[N_items] delta
    = append_row([0]', delta_free);
  vector[N_subjects] zeta
    = append_row([0]', zeta_free);

  array[N] real log_reading_time_pred;
```

Stan
Program 8 `dat2.stan`

```
functions {
  // Mean-dispersion parameterization of inverse gamma family
  real inv_gamma_md_lpdf(real x, real log_mu, real psi) {
    return inv_gamma_lpdf(x | inv(psi) + 2,
                              exp(log_mu) * (inv(psi) + 1));
  }

  real inv_gamma_md_rng(real log_mu, real psi) {
    return inv_gamma_rng(inv(psi) + 2,
                         exp(log_mu) * (inv(psi) + 1));
  }
}

data {
  int<lower=1> N; // Number of observations

  // Item configuration
  int<lower=1> N_items;
  array[N] int<lower=1, upper=N_items> item;

  // Subject configuration
  int<lower=1> N_subjects;
  array[N] int<lower=1, upper=N_subjects> subject;

  // Item variant
  //   Object Relative:  subj_rel = 0
  //   Subject Relative: subj_rel = 1
  array[N] int<lower=0, upper=1> subj_rel;

  vector<lower=0>[N] reading_time; // Reading times (ms)
}

parameters {
  // Log reading time baseline for successful retrieval
  real nu;

  // Relative item difficulties
  vector[N_items - 1] delta_free;

  // Relative subject skills
  vector[N_subjects - 1] zeta_free;

  // Log offset for initial failure before successful retrieval
  real<lower=0> omega;

  // Measurement scales
  real<lower=0> phi1;
  real<lower=0> phi2;

  // Mixture probabilities
```

Stan
Program 9 `joint\_prior.stan`

```
functions {
  // Mean-dispersion parameterization of inverse gamma family
  real inv_gamma_md_lpdf(real x, real log_mu, real psi) {
    return inv_gamma_lpdf(x | inv(psi) + 2,
                              exp(log_mu) * (inv(psi) + 1));
  }

  real inv_gamma_md_rng(real log_mu, real psi) {
    return inv_gamma_rng(inv(psi) + 2,
                         exp(log_mu) * (inv(psi) + 1));
  }
}

data {
  int<lower=1> N; // Number of observations

  // Item configuration
  int<lower=1> N_items;
  array[N] int<lower=1, upper=N_items> item;

  // Subject configuration
  int<lower=1> N_subjects;
  array[N] int<lower=1, upper=N_subjects> subject;

  // Item variant
  //   Object Relative:  subj_rel = 0
  //   Subject Relative: subj_rel = 1
  array[N] int<lower=0, upper=1> subj_rel;

  vector<lower=0>[N] reading_time; // Reading times (ms)
}

generated quantities {
  // Log reading time baseline
  real tau = normal_rng(5.76, 0.50);

  // Relative item difficulties
  vector[N_items - 1] delta_free
    = to_vector(normal_rng(zeros_vector(N_items - 1), 0.99));

  // Relative subject skills
  vector[N_subjects - 1] zeta_free
    = to_vector(normal_rng(zeros_vector(N_subjects - 1), 0.99));

  // Initial failure difference
  real<lower=0> omega = abs(normal_rng(0, 0.90));

  // Subject Relative Difference
  real chi = normal_rng(0, 0.99);
```

Stan
Program 10 `joint.stan`

```
functions {
  // Mean-dispersion parameterization of inverse gamma family
  real inv_gamma_md_lpdf(real x, real log_mu, real psi) {
    return inv_gamma_lpdf(x | inv(psi) + 2,
                              exp(log_mu) * (inv(psi) + 1));
  }

  real inv_gamma_md_rng(real log_mu, real psi) {
    return inv_gamma_rng(inv(psi) + 2,
                         exp(log_mu) * (inv(psi) + 1));
  }
}

data {
  int<lower=1> N; // Number of observations

  // Item configuration
  int<lower=1> N_items;
  array[N] int<lower=1, upper=N_items> item;

  // Subject configuration
  int<lower=1> N_subjects;
  array[N] int<lower=1, upper=N_subjects> subject;

  // Item variant
  //   Object Relative:  subj_rel = 0
  //   Subject Relative: subj_rel = 1
  array[N] int<lower=0, upper=1> subj_rel;

  vector<lower=0>[N] reading_time; // Reading times (ms)
}

parameters {
  // Log reading time baseline for successful retrieval
  real tau;

  // Relative item difficulties
  vector[N_items - 1] delta_free;

  // Relative subject skills
  vector[N_subjects - 1] zeta_free;

  // Log offset for initial failure before successful retrieval
  real<lower=0> omega;

  // Subject Relative Difference
  real chi;

  // Measurement scales
  real<lower=0> phi1;
```

Stan
Program 11 `dat2\_rounding.stan`

```
functions {
  // Mean-dispersion parameterization of inverse gamma family
  real inv_gamma_md_lpdf(real x, real log_mu, real psi) {
    return inv_gamma_lpdf(x | inv(psi) + 2,
                             exp(log_mu) * (inv(psi) + 1));
  }

  real inv_gamma_md_cdf(real x, real log_mu, real psi) {
    return inv_gamma_cdf(x | inv(psi) + 2,
                            exp(log_mu) * (inv(psi) + 1));
  }

  real inv_gamma_md_rng(real log_mu, real psi) {
    return inv_gamma_rng(inv(psi) + 2,
                         exp(log_mu) * (inv(psi) + 1));
  }
}

data {
  int<lower=1> N; // Number of observations

  // Item configuration
  int<lower=1> N_items;
  array[N] int<lower=1, upper=N_items> item;

  // Subject configuration
  int<lower=1> N_subjects;
  array[N] int<lower=1, upper=N_subjects> subject;

  // Item variant
  //   Object Relative:  subj_rel = 0
  //   Subject Relative: subj_rel = 1
  array[N] int<lower=0, upper=1> subj_rel;

  array[N] int<lower=0> reading_time; // Reading times (ms)
}

parameters {
  // Log reading time baseline for successful retrieval
  real nu;

  // Relative item difficulties
  vector[N_items - 1] delta_free;

  // Relative subject skills
  vector[N_subjects - 1] zeta_free;

  // Log offset for initial failure before successful retrieval
  real<lower=0> omega;
```

Stan
Program 12 `dat1\_rounding.stan`

```
data {
  int<lower=1> N; // Number of observations

  // Item configuration
  int<lower=1> N_items;
  array[N] int<lower=1, upper=N_items> item;

  // Subject configuration
  int<lower=1> N_subjects;
  array[N] int<lower=1, upper=N_subjects> subject;

  // Item variant
  //   Object Relative:  subj_rel = 0
  //   Subject Relative: subj_rel = 1
  array[N] int<lower=0, upper=1> subj_rel;

  array[N] int<lower=0> reading_time; // Reading times (ms)
}

parameters {
  // Log reading time baseline for successful retrieval
  real nu;

  // Relative item difficulties
  vector[N_items - 1] delta_free;

  // Relative subject skills
  vector[N_subjects - 1] zeta_free;

  // Log offset for initial failure before successful retrieval
  real<lower=0> omega;

  // Measurement scales
  real<lower=0> phi1;
  real<lower=0> phi2;

  // Mixture probabilities
  real<lower=0, upper=1> lambda_SR;
  real<lower=0, upper=1> lambda_OR;
}

transformed parameters {
  // Relative skills for all items and subjects
  vector[N_items] delta
    = append_row([0]', delta_free);
  vector[N_subjects] zeta
    = append_row([0]', zeta_free);
}

model {
```

Stan
Program 13 `dat\_nb.stan`

```
data {
  int<lower=1> N; // Number of observations

  // Item configuration
  int<lower=1> N_items;
  array[N] int<lower=1, upper=N_items> item;

  // Subject configuration
  int<lower=1> N_subjects;
  array[N] int<lower=1, upper=N_subjects> subject;

  // Item variant
  //   Object Relative:  subj_rel = 0
  //   Subject Relative: subj_rel = 1
  array[N] int<lower=0, upper=1> subj_rel;

  array[N] int<lower=0> reading_time; // Reading times (ms)
}

parameters {
  // Log reading time baseline for successful retrieval
  real nu;

  // Relative item difficulties
  vector[N_items - 1] delta_free;

  // Relative subject skills
  vector[N_subjects - 1] zeta_free;

  // Log offset for initial failure before successful retrieval
  real<lower=0> omega;

  // Measurement scales
  real<lower=0> phi1;
  real<lower=0> phi2;

  // Mixture probabilities
  real<lower=0, upper=1> lambda_SR;
  real<lower=0, upper=1> lambda_OR;
}

transformed parameters {
  // Relative skills for all items and subjects
  vector[N_items] delta
    = append_row([0]', delta_free);
  vector[N_subjects] zeta
    = append_row([0]', zeta_free);
}

model {
```