Randomized Time Trial

Michael Betancourt

August 2024

Table of contents

Modeling the outcome of competitions, for example games between competing sports teams or tests between students and a standardized set of questions, is a common statistics application. Different types of competitions, however, are more or less compelling to certain audiences. In this case study I consider a Bayesian analysis of a somewhat niche competition that also happens to be particular compelling to the author – racing to see who can finish a modified version of a thirty year old video game as quickly as possible.

1 Super Metroid Map Randomizer Races

In the era of the Super Nintendo Entertainment System® the assets comprising each video game – such as code, visuals, and music – were stored in the read-only memory, or **ROM**, of physical cartridges. **ROM hacks** rearrange the assets of a particular game, and in some cases include assets from other games or even entirely new assets, to create novel gaming experiences. Many popular ROM hacks, for example, add quality of life features to make older games less frustrating to play. Others focus on drastically increasing the difficulty of existing games while still others offer the ability to randomize locations, items, and more so that each new playthrough is unique.

Super Metroid® was first released for the Super Nintendo Entertainment System® in 1994. The game's well-designed core mechanics interact surprisingly well with unintended bugs, resulting in fast-paced and highly-technical game play that has long been a favorite target of the ROM hacking community.

One prominent example is the Super Metroid Map Randomizer ("Super Metroid Map Rando," n.d.), or *MapRando* for short. The Super Metroid Map Randomizer is an open source project started in 2021 that randomizes individual rooms, items, objectives, and more while also avoiding any inconsistencies that would prevent players from completing the game. Randomization options are extensive and can be configured to control everything from the topology of the room placement to the difficulty of the techniques needed for completion. Each realized map is

referred to as a *seed* for the seed that initializes the behavior of the underlying pseudo-random number generator.

By 2024 the project had stimulated a passionate user community that not only played the games individually but also started to race against each other to see who could finish a particular seed the fastest. Because some seeds end up being easier to finish than others the races tend to be a bit chaotic and consistently entertaining.

Community races are even organized and recorded on the non-commercial speed running website racetime.gg ("Super Metroid Randomizer | Racetime.gg," n.d.). Conveniently this organization makes data on previous races, including individual entrants and their race outcomes, readily accessible. The availability of this data in turn puts us in a position to infer and compare the skill of those entrants and predict the outcome of future races.

2 Environment Setup

Before exploring that data we'll need to 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 Bayesian inference we'll also need my recommended diagnostics and visualization tools.

```
util <- new.env()
source('mcmc_analysis_tools_rstan.R', local=util)
source('mcmc_visualization_tools.R', local=util)
```

3 Data Exploration

To assemble the full data set I programmatically scraped `https://racetime.gg/smr` for all races with the title `Map Rando` that also include a link to the MapRando configuration in their description. This then allowed me to collect additional information on the MapRando

version while also restricting consideration to only those seeds with a `Hard` skill assumption and `Tricky` item progression setting.

For each valid race I then scraped information on the individual participants, in particular whether they finished the race or forfeited and, if they finished, what their finish time was in seconds. Forfeits are also referred to as "did not finish" or "DNF". Although forfeit times are available on `https://racetime.gg/smr` interactively they are difficult to extract programmatically and I consequently did not include them.

Following the `racetime.gg` terminology I will refer to individual participants in a race as **entrants** and their participation into a particular race as an **entrance**. For programming convenience I encoded the individual entrant usernames into sequential numerical labels that can also be used for indexing.

Each race consists of a variable number of entrances, with each entrance resulting in either a finish time or a forfeit. To accommodate this ragged structure I organized the finish entrances and forfeit entrances for all races into single arrays that are complemented with indexing arrays for straightforward retrieval of individual race information.

The data collection scripts, translation between entrant indices and usernames, and final data are all accessible in the GitHub repository for this chapter.

```
entrant_info <- read.csv("data/entrant_level_defs.csv")

race_info <- read.csv("data/race_info.csv")
race_entrant_f_info <- read.csv("data/race_entrant_f_info.csv")
race_entrant_dnf_info <- read.csv("data/race_entrant_dnf_info.csv")

data <- list("N_races" = nrow(race_info),
             "N_entrants" = nrow(entrant_info),
             "race_N_entrants_f" = race_info$race_N_entrants_f,
             "race_N_entrants_dnf" = race_info$race_N_entrants_dnf,
             "race_f_start_idxs" = race_info$race_f_start_idxs,
             "race_f_end_idxs" = race_info$race_f_end_idxs,
             "race_dnf_start_idxs" = race_info$race_dnf_start_idxs,
             "race_dnf_end_idxs" = race_info$race_dnf_end_idxs,
             "race_entrant_f_idxs" = race_entrant_f_info$race_entrant_f_idxs,
             "race_entrant_f_times" =  race_entrant_f_info$race_entrant_f_times,
             "N_entrances_fs" = length(race_entrant_f_info$race_entrant_f_idxs),
             "race_entrant_dnf_idxs" = race_entrant_dnf_info$race_entrant_dnf_idxs,
             "N_entrances_dnfs" = length(race_entrant_dnf_info$race_entrant_dnf_idxs))
```

Altogether the data spans 192 races across the first eight months of 2024.

```
cat(sprintf('%i total races', data$N_races))
```

```
192 total races
```

```
cat(sprintf('First Race: %s', race_info$race_datetimes[1]))
```

```
First Race: 2024-01-09T21:39:16.610866+00:00
```

```
cat(sprintf('Last Race: %s',
            race_info$race_datetimes[length(race_info$race_datetimes)]))
```

```
Last Race: 2024-07-27T23:28:49.606056+00:00
```

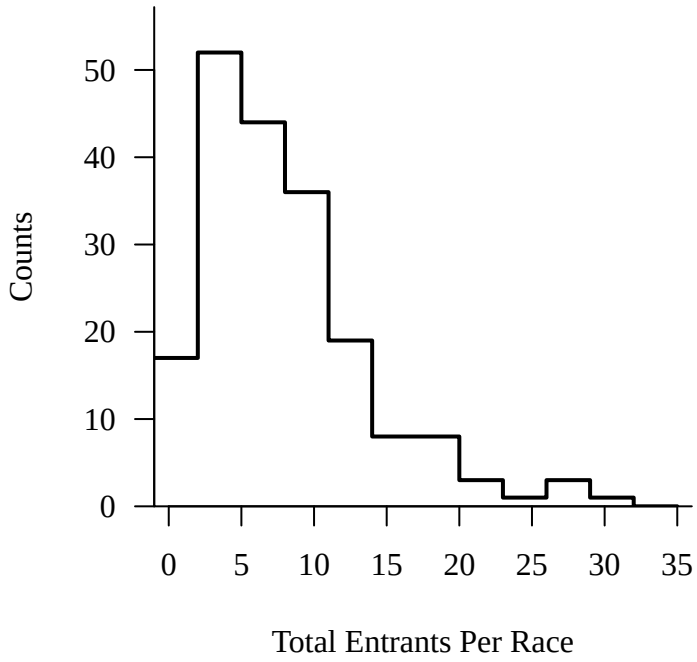
Those races included 107 distinct entrants.

```
cat(sprintf('%i total entrants', data$N_entrants))
```

```
107 total entrants
```

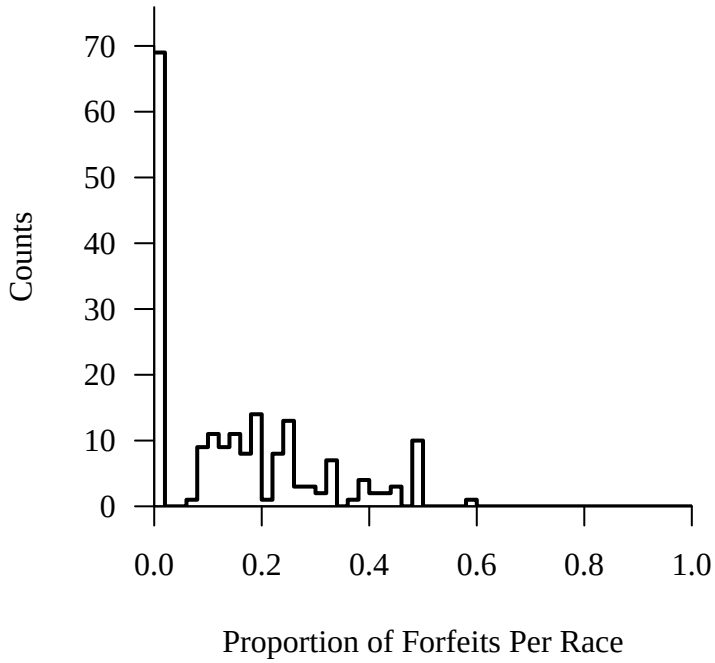
While the majority of races include at least five entrants some include over thirty.

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

util$plot_line_hist(data$race_N_entrants_f + data$race_N_entrants_dnf,
                    0, 35, 3,
                    xlab="Total Entrants Per Race")
```



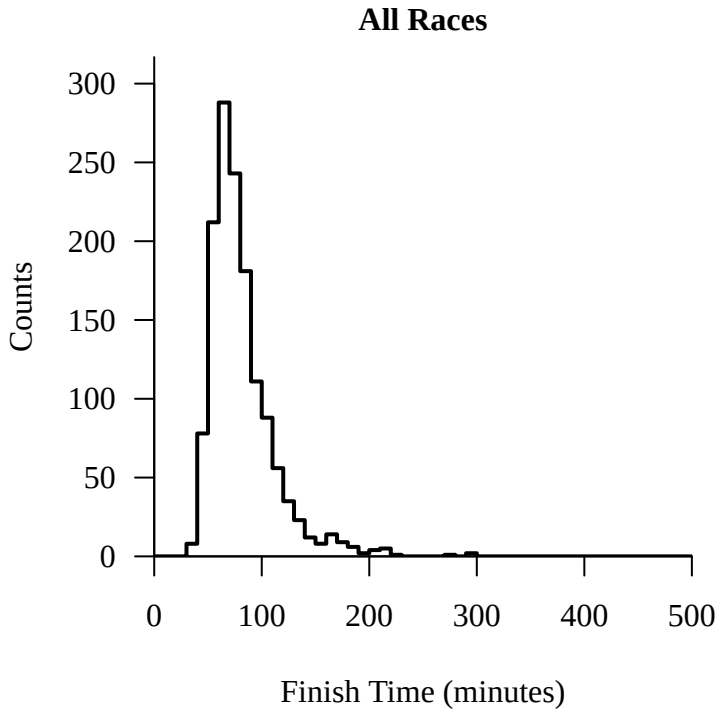
Similarly most races see most entrants finishing but some races, presumable using more difficult seeds, can see over half of the entrants forfeit.

```
par(mfrow=c(1, 1), mar=c(5, 5, 2, 1))
util$plot_line_hist(data$race_N_entrants_dnf /
                    (data$race_N_entrants_f + data$race_N_entrants_dnf),
                    0, 1, 0.02,
                    xlab="Proportion of Forfeits Per Race")
```



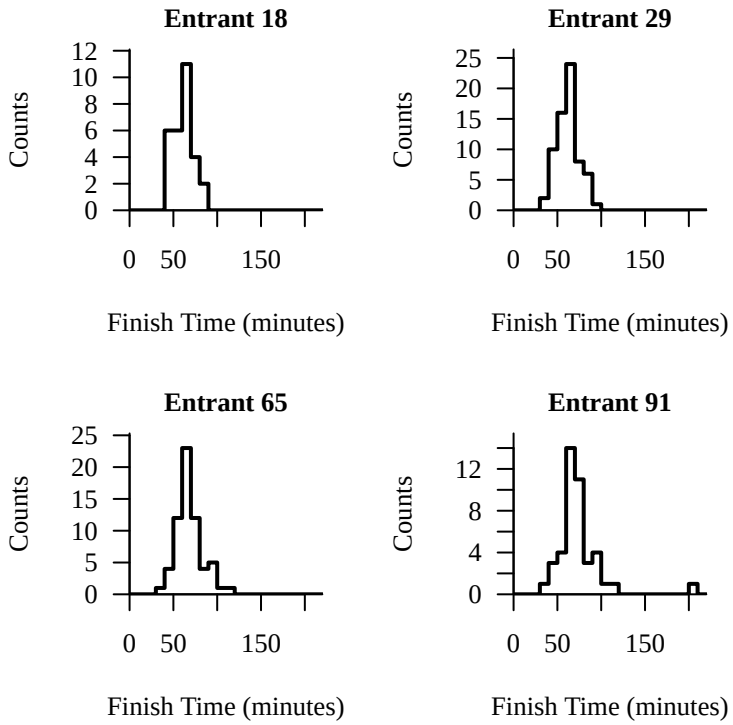
The finish times across races vary substantially, peaking near an hour but stretching from half an hour all the way to multiple hours. This suggests that player skill, seed difficulty, or some combination of the two, is highly variable from race to race.

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

util$plot_line_hist(data$race_entrant_f_times / 60, 0, 500, 10,
                    xlab="Finish Time (minutes)", main="All Races")
```



If we isolate the finish times for a few entrants near the top of the `https://racetime.gg/smr` leader boards then we see much less variability, especially in the upper tail.

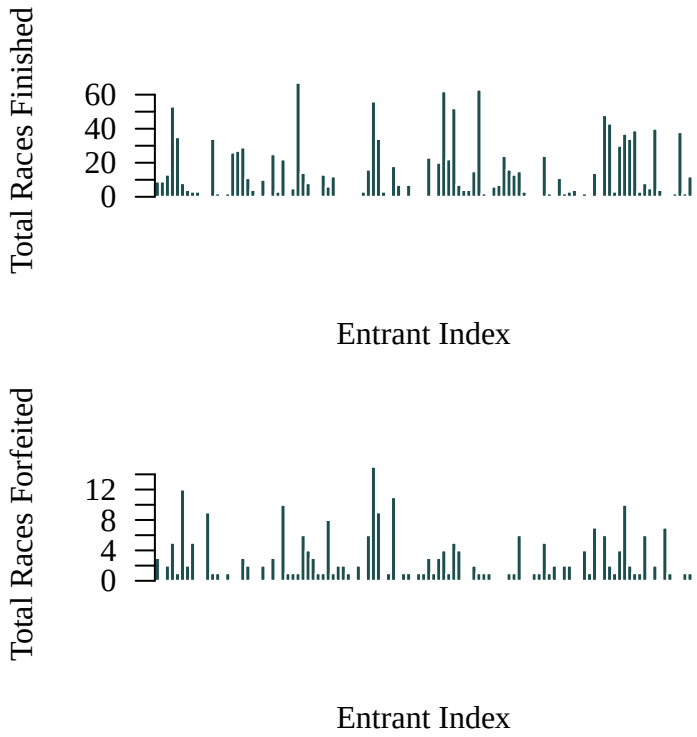
```
par(mfrow=c(2, 2), mar=c(5, 5, 2, 1))

for (e in c(18, 29, 65, 91)) {
  times <- data$race_entrant_f_times[which(data$race_entrant_f_idxs == e)]
  util$plot_line_hist(times / 60, 0, 220, 10,
                      xlab="Finish Time (minutes)",
                      main=paste("Entrant", e))
}
```



Overall participation and proportion of forfeits exhibits its own heterogeneity across the individual entrants.

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

entrant_f_idxs <- function(e) {
  which(data$race_entrant_f_idxs == e)
}

entrant_dnf_idxs <- function(e) {
  which(data$race_entrant_dnf_idxs == e)
}

N_entrant_f_races <- sapply(1:data$N_entrants,
                            function(e) length(entrant_f_idxs(e)))
N_entrant_dnf_races <- sapply(1:data$N_entrants,
                              function(e) length(entrant_dnf_idxs(e)))

barplot(N_entrant_f_races,
        space=0, col=util$c_dark_teal, border="white",
        xlab="Entrant Index", ylab="Total Races Finished")
```

```
barplot(N_entrant_dnf_races,
        space=0, col=util$c_dark_teal, border="white",
        xlab="Entrant Index", ylab="Total Races Forfeited")
```



There are many more ways that we could dive further into the data here, but without domain expertise to guide us it's easy to get lost. Instead let's leverage what understanding we've developed into an initial model.

4 Model Development

To make the modeling process as manageable as possible we will start simple and then add features iteratively.

4.1 Model 1

To begin let's ignore forfeits altogether and instead focus on modeling the finish times of the entrants who do not forfeit in each race.

4.1.1 Observational Model

One of the challenging aspects of modeling races in general is that individual entrant behavior can depend on their position at any given time. For example an entrant in first place might slow their pace to stay just ahead of the entrant in second place, while racers in lower places might play more aggressively in an attempt to catch up. While some MapRando race entrants live-stream their play publicly the community largely follows the rules of `https://racetime.gg/smr` which disallow entrants from following the progress of any other entrants.

Consequently entrants are mostly ignorant of their position during races, at least until the first entrants finish and post their finish times on the active `https://racetime.gg/smr` race page. This suggest that modeling the finish time of each entrant independently of the other entrants, without having to worry about interactions within each race, is a reasonable approximation to the true race dynamics.

Now a particularly crude model of independent finish times might assume that the finish times across all races concentrate around some common baseline value,

$$\mu = t_{\text{baseline}} = \exp(\eta) \cdot 1 \text{ second}.$$

Not all races, however, are the same.

For example some race seeds result in map layouts that require traveling longer distances than others and some require executing difficult techniques that usually take longer to complete than others. We could account for this heterogeneity with a separate baseline finish time for each seed, but we could also account for it by proportionally modifying the common baseline,

$$\begin{aligned}\mu_r &= t_{\text{baseline}} \cdot \delta_{\text{difficulty},r} \\ &= \exp(\eta) \cdot \exp(\lambda_{\text{difficulty},r}) \cdot 1 \text{ second} \\ &= \exp(\eta + \lambda_{\text{difficulty},r}) \cdot 1 \text{ second}.\end{aligned}$$

Similarly not all entrants are the same. The entrants who are more experienced with Super Metroid® game play in general, and MapRando game play in particular, should be able to finish a random seed faster than those who are less experienced. Again we can model this with a proportional modification to the common baseline,

$$\begin{aligned}\mu_{re} &= t_{\text{baseline}} \cdot \delta_{\text{difficulty},r} \cdot \frac{1}{\delta_{\text{skill},e}} \\ &= \exp(\eta) \cdot \exp(\lambda_{\text{difficulty},r}) \cdot \frac{1}{\exp(\lambda_{\text{skill},e})} \cdot 1 \text{ second} \\ &= \exp(\eta + \lambda_{\text{difficulty},r} - \lambda_{\text{skill},e}) \cdot 1 \text{ second}.\end{aligned}$$

To complete our observational model we need to model the variation of finish times around these baselines. One immediate possibility is the gamma family of probability density functions, not in its conventional parameterization but rather in its mean-dispersion parameterization. For example the gamma family of probability density functions is typically parameterized in terms of a shape parameter α and a scale parameter β. We can also parameterize this same family, however, in terms of a location parameter

$$\mu = \text{mean}(\alpha, \beta) = \frac{\alpha}{\beta},$$

and a dispersion parameter

$$\begin{aligned} \psi &= \frac{\text{variance}(\alpha, \beta)}{\text{mean}^2(\alpha, \beta)} \\ &= \frac{\alpha}{\beta^2} \left(\frac{\beta}{\alpha} \right)^2 \\ &= \frac{1}{\alpha}. \end{aligned}$$

We can then define an appropriate observational model by replacing the location parameter μ with the race-entrant baseline μ_{re} for each entrant in a particular race.

4.1.2 Prior Model

To elevate our observational model to a full Bayesian model we need to specify a prior model over our model configuration variables. Here we'll assume that the prior model is built up from independent component prior models for each parameter.

To avoid unrealistically fast and slow races let's constrain the baseline finish time to

$$\begin{aligned} 1800 \text{ seconds} &\lessapprox & t_{\text{baseline}} & \lessapprox 5400 \text{ seconds} \\ 1800 \text{ seconds} &\lessapprox & \exp(\eta) \cdot 1 \text{ second} & \lessapprox 5400 \text{ seconds} \\ 1800 &\lessapprox & \exp(\eta) & \lessapprox 5400 \\ \log 1800 &\lessapprox & \eta & \lessapprox \log 5400. \end{aligned}$$

We can ensure that 98% of the prior probability is contained within these bounds with the prior model

$$\begin{aligned} p(\eta) &= \text{normal}\left(\eta \,\middle|\, \frac{\log 5400 + \log 1800}{2}, \frac{1}{2.32} \frac{\log 5400 - \log 1800}{2} \right) \\ &= \text{normal}(\eta \mid 8.045, 0.237). \end{aligned}$$

Note that this prior model doesn't suppress finish times below 30 minutes and above 90 minutes, just the baseline around which finish times concentrate. The variation of the gamma observational model will allow for much smaller and much larger finish times.

Similarly it would be a bit extreme if seed difficulty or entrant skill modified the baseline by more than a factor of two,

$$\begin{aligned} \frac{1}{2} &\lessapprox \delta \lessapprox 2 \\ \log \frac{1}{2} &\lessapprox \lambda \lessapprox \log 2 \\ -\log 2 &\lessapprox \lambda \lessapprox \log 2. \end{aligned}$$

A reasonable prior that achieves this soft containment is then

$$\begin{aligned} p(\lambda) &= \text{normal}\left(\lambda \,\middle|\, \frac{\log 2 + (-\log 2)}{2}, \frac{1}{2.32} \frac{\log 2 - (-\log 2)}{2}\right) \\ &= \text{normal}\left(\lambda \,\middle|\, 0, \frac{1}{2.32} \log 2\right) \\ &= \text{normal}(\lambda \mid 0, 0.299). \end{aligned}$$

Lastly we need to consider the dispersion strength ψ. Here let's suppress model configurations where the variance would exceed the squared mean,

$$\begin{aligned} 0 &\lessapprox \frac{\text{variance}(\alpha, \beta)}{\text{mean}^2(\alpha, \beta)} \lessapprox 1 \\ 0 &\lessapprox \quad \psi \quad \lessapprox 1. \end{aligned}$$

One way to achieve this soft containment is with the prior model

$$p(\psi) = \text{half-normal}\left(\psi \,\middle|\, 0, \frac{1}{2.57}\right) = \text{half-normal}\left(\psi \,\middle|\, 0, 0.389\right).$$

4.1.3 Anchors

Incidentally this simplified model of the data generating process is a pairwise comparison model. In particular the race seeds and entrants define bipartide items with the race seed difficulty and entrant skills their respective qualities. Moreover the location of the finish times is coupled to these item qualities with a baseline-inducing function.

One advantage of this categorization is that it can motivate productive model expansions. For example we could model how sensitive different race seeds are to entrant skill by introducing a discrimination parameter for each race,

$$\mu_{se} = \exp(\gamma + \gamma_r \cdot (\lambda_{\text{difficulty},r} - \lambda_{\text{skill},e})) \cdot 1 \text{ second}.$$

The smaller γ_r is the less sensitive the finish times for the given race seed will be to the contrast between race seed difficulty and entrant skill.

Another benefit of this categorization is a recognition of potential inferential degeneracies and guidance on how to robustify our model to them. For example because we are using a baseline-inducing function and the items being compared are bipartide we need to anchor one of each item type within each connected component to avoid degeneracies.

Interestingly there is more than one connected component here.

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

cat_race_idxs <- unlist(sapply(1:data$N_races,
                               function(r)
                               rep(r, data$race_N_entrants_f[r])))

cat_entrant_idxs <- data$N_races + data$race_entrant_f_idxs

adj <- util$build_adj_matrix(data$N_races + data$N_entrants,
                             data$N_entrances_fs,
                             cat_race_idxs, cat_entrant_idxs)

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

```
[1] 12
```

That said all but the first connected component are singleton components consisting of a single entrant.

```
format_bipartide_item <- function(i) {
  ifelse(i > data$N_race,
         paste('Entrant', i - data$N_races),
         paste('Race', i))
}

for (c in seq_along(components)) {
  cat(sprintf('Component %s:\n', c))
  items <- sprintf('%s,', sapply(components[[c]],
                                 format_bipartide_item))
  cat(paste0(strwrap(do.call(paste, as.list(items)), 70, 2),
      collapse='\n'))
  cat('\n\n')
}
```

Component 1:
Race 1, Entrant 18, Race 2, Entrant 7, Race 3, Entrant 70, Race 6, Entrant 12, Race 7, Entrant 4, Race 8, Entrant 29, Race 4, Race 5, Entrant 91, Race 9, Race 10, Entrant 36, Race 16, Entrant 5, Race 12, Entrant 3, Race 31, Entrant 43, Race 32, Entrant 1, Race 19, Entrant 13, Race 186, Entrant 30, Race 11, Entrant 28, Race 15, Entrant 6, Race 51, Entrant 17, Race 27, Entrant 57, Race 17, Entrant 64, Race 22, Entrant 44, Race 20, Entrant 88, Race 24, Entrant 34, Race 23, Entrant 65, Race 14, Race 33, Entrant 16, Race 64, Entrant 26, Race 21, Entrant 45, Race 30, Entrant 14, Entrant 58, Race 25, Entrant 60, Race 29, Entrant 19, Race 37, Race 42, Entrant 94, Race 28, Entrant 68, Race 36, Entrant 96, Race 50, Entrant 107, Race 39, Entrant 90, Race 13, Entrant 69, Race 18, Race 63, Entrant 2, Race 34, Entrant 95, Race 38, Race 58, Entrant 31, Race 74, Entrant 61, Race 75, Entrant 100, Race 26, Entrant 73, Race 61, Entrant 51, Race 48, Entrant 49, Race 49, Race 52, Entrant 24, Race 45, Entrant 82, Race 80, Entrant 22, Race 76, Entrant 23, Race 87, Entrant 48, Race 55, Entrant 78, Race 43, Entrant 79, Entrant 83, Race 105, Entrant 55, Race 90, Race 93, Entrant 81, Race 53, Race 84, Entrant 98, Race 71, Race 102, Entrant 8, Race 86, Entrant 93, Race 88, Race 91, Entrant 105, Race 59, Entrant 20, Race 172, Entrant 71, Race 110, Entrant 42, Race 103, Entrant 72, Race 68, Race 69, Race 97, Entrant 59, Race 89, Race 92, Race 94, Race 104, Race 107, Race 117, Race 132, Race 139, Race 140, Entrant 25, Race 143, Entrant 35, Race 62, Race 101, Race 106, Entrant 10, Race 127, Race 128, Entrant 62, Race 147, Entrant 9, Race 154, Entrant 41, Entrant 97, Entrant 104, Race 155, Race 157, Entrant 86, Entrant 74, Race 142, Entrant 32, Entrant 46, Race 149, Entrant 92, Race 148, Entrant 11, Entrant 39, Entrant 99, Race 56, Entrant 106, Race 179, Race 141, Entrant 101, Race 123, Entrant 89, Race 160, Race 165, Race 171, Entrant 50, Race 173, Race 183, Race 184, Race 188, Race 192, Race 98, Race 112, Race 115, Race 134, Race 136, Entrant 75, Race 137, Race 166, Race 174, Race 177, Race 191, Entrant 15, Race 65, Race 66, Entrant 33, Race 72, Race 73, Race 78, Race 82, Race 99, Race 100, Race 111, Race 121, Race 151, Race 152, Race 169, Race 95, Race 96, Entrant 40, Race 118, Race 120, Race 124, Race 125, Race 126, Race 130, Race 156, Entrant 21, Entrant 52, Race 162, Race 114, Race 133, Race 144, Race 185, Race 77, Race 81, Race 60, Race 145, Entrant 76, Race 46, Race 122, Race 161, Race 176, Race 178, Race 54, Race 70, Race 158, Race 159, Race 40, Race 41, Race 44, Race 47, Race 67, Entrant 84, Race 83, Race 85, Race 135, Race 164, Entrant 54, Race 35, Race 79, Race 180, Race 131, Race 138, Race 57, Race 108, Race 113, Race 150, Race 116, Race 129, Race 168, Race 190, Entrant 63, Race 182, Race 153, Race 170, Race 189, Entrant 66, Race

```
109, Race 146, Race 181, Entrant 85, Race 119, Race 175, Race 187,
Race 163, Entrant 102, Race 167,

Component 2:
  Entrant 27,

Component 3:
  Entrant 37,

Component 4:
  Entrant 38,

Component 5:
  Entrant 47,

Component 6:
  Entrant 53,

Component 7:
  Entrant 56,

Component 8:
  Entrant 67,

Component 9:
  Entrant 77,

Component 10:
  Entrant 80,

Component 11:
  Entrant 87,

Component 12:
  Entrant 103,
```

All of these isolated entrants have never finished a race, with their participation limited to forfeits.

```
for (component in components) {
  if (length(component) == 1) {
    e <- component[1] - data$N_races
```

```
    cat(sprintf('Entrant %s: %i finished races\n',
        e, N_entrant_f_races[e]))
  }
}
```

```
Entrant 27: 0 finished races
Entrant 37: 0 finished races
Entrant 38: 0 finished races
Entrant 47: 0 finished races
Entrant 53: 0 finished races
Entrant 56: 0 finished races
Entrant 67: 0 finished races
Entrant 77: 0 finished races
Entrant 80: 0 finished races
Entrant 87: 0 finished races
Entrant 103: 0 finished races
```

When modeling finish times we need to anchor only a single race seed and entrant. Here we'll anchor the most popular race seed and most prolific entrant.

```
data$anchor_race_idx <-
  which(data$race_N_entrants_f == max(data$race_N_entrants_f))

data$anchor_entrant_idx <-
  which(N_entrant_f_races == max(N_entrant_f_races))
```

The last thing we need to do is push forward our normal prior model over the race difficulties and entrant skills,

$$\text{normal}(\lambda \mid 0, \log 2 \,/\, 2.32)$$

to an appropriately inflated prior model over the relative difficulties and entrant skills,

$$\text{normal}(\delta \mid 0, \sqrt{2} \cdot \log 2 \,/\, 2.32).$$

4.1.4 Inferential Computation

We can now implement our full Bayesian model as a Stan program, plug in the observed data, and give Stan's Hamiltonian Monte Carlo sampler a chance at exploring the posterior distribution.

```
fit <- stan(file="stan_programs/model1.stan",
            data=data, seed=8438338,
            warmup=1000, iter=2024, refresh=0)
```

The empirical effective sample size warnings suggest some nontrivial autocorrelations, especially in the baseline η, but nothing large enough to compromise the faithfulness of Markov chain Monte Carlo integration.

```
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.
```

```
samples1 <- util$extract_expectand_vals(fit)
base_samples <- util$filter_expectands(samples1,
                                       c('eta',
                                         'rel_difficulties_free',
                                         'rel_skills_free',
                                         'psi'),
                                       check_arrays=TRUE)
util$check_all_expectand_diagnostics(base_samples)
```

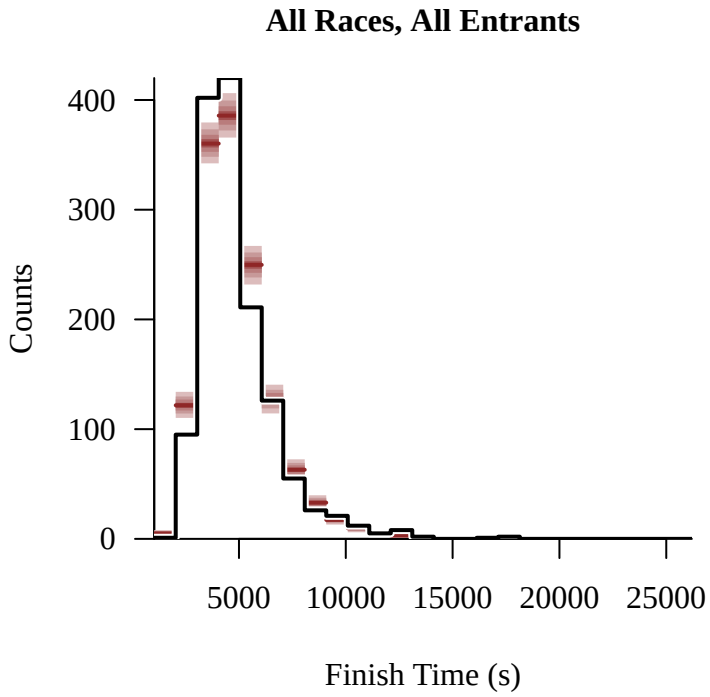
```
eta:
  Chain 1: hat{ESS} (34.284) is smaller than desired (100).
  Chain 2: hat{ESS} (58.063) is smaller than desired (100).
  Chain 3: hat{ESS} (52.912) is smaller than desired (100).
  Chain 4: hat{ESS} (64.830) is smaller than desired (100).

rel_skills_free[64]:
  Chain 1: hat{ESS} (95.800) is smaller than desired (100).


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

4.1.5 Posterior Retrodictive Checks

There are a lot of summary statistics that we might consider for our visual posterior retrodictive checks. Indeed we explored some candidates in Section 3.

For example we could use a histogram summary statistic that aggregates the finish times across all races. Here we see a pretty strong retrodictive tension, with the observed finish times exhibiting stronger skewness than what the posterior predictive distribution can accommodate.

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

util$plot_hist_quantiles(samples1, 'race_entrant_f_times_pred',
                         baseline_values=data$race_entrant_f_times,
                         xlab="Finish Time (s)",
                         main="All Races, All Entrants")
```



This tension could be due to an inadequacy of the gamma observational model, but it could also be a consequence of poorly modeling the heterogeneity in seed difficulties and entrant skills. One way to explore these possibilities is to separate the histogram summary statistic by race and entrant.

Here there doesn't seem to be any substantial retrodictive tension in the finish times for a few arbitrarily selected races.

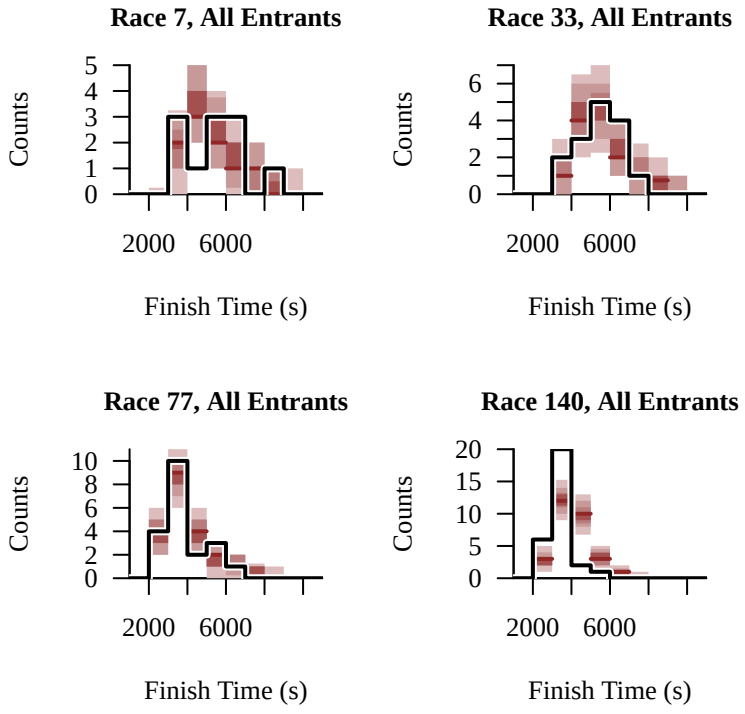
```
par(mfrow=c(2, 2), mar=c(5, 5, 3, 1))

for (r in c(7, 33, 77, 140)) {
```

```
  idxs <- data$race_f_start_idxs[r]:data$race_f_end_idxs[r]
  names <- sapply(idxs,
                  function(n) paste0('race_entrant_f_times_pred[', n, ']'))
  filtered_samples <- util$filter_expectands(samples1, names)
  util$plot_hist_quantiles(filtered_samples, 'race_entrant_f_times_pred',
                           1000, 11000, 1000,
                           baseline_values=data$race_entrant_f_times[idxs],
                           xlab="Finish Time (s)",
                           main=paste0("Race ", r, ", All Entrants"))
}
```

```
Warning in check_bin_containment(bin_min, bin_max, collapsed_values,
"predictive value"): 65 predictive values (0.1%) fell above the binning.
```

```
Warning in check_bin_containment(bin_min, bin_max, collapsed_values,
"predictive value"): 100 predictive values (0.2%) 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.
```

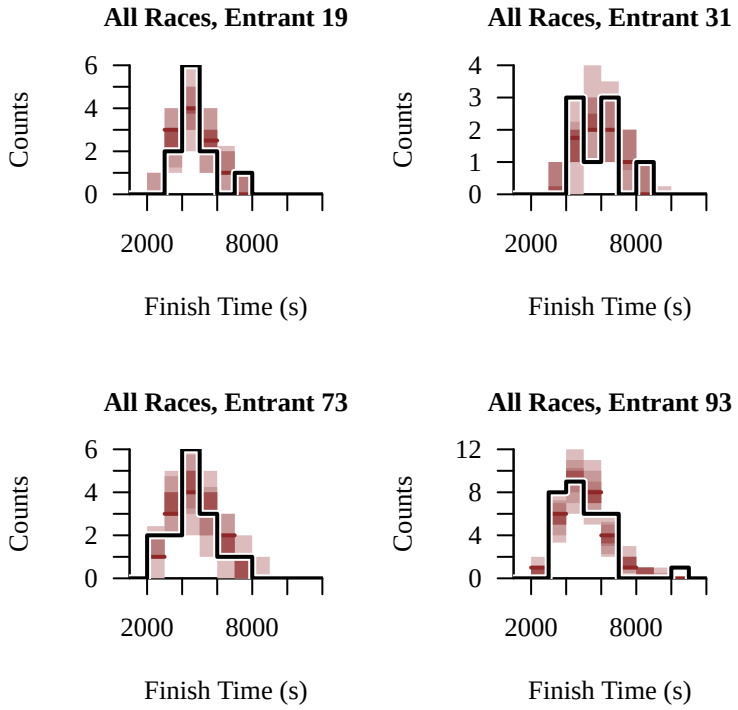


Similarly the finish time behaviors for a few spot-checked entrants are consistent between the observed data and our posterior predictions. One might argue that the observed behavior for entrant 93 is slightly heavier-tailed than the posterior predictions, but the disagreement is relatively weak.

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

for (e in c(19, 31, 73, 93)) {
  idxs <- which(data$race_entrant_f_idxs == e)
  names <- sapply(idxs,
                  function(n) paste0('race_entrant_f_times_pred[', n, ']'))
  filtered_samples <- util$filter_expectands(samples1, names)
  util$plot_hist_quantiles(filtered_samples, 'race_entrant_f_times_pred',
                           1000, 12000, 1000,
                           baseline_values=data$race_entrant_f_times[idxs],
                           xlab="Finish Time (s)",
                           main=paste0("All Races, Entrant ", e))
}
```

```
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"): 23 predictive values (0.0%) fell above the binning.
```



If we really wanted to be thorough then we would need to examine the behavior of the hundreds of finish time histograms across all of the individual races and all of the individual entrants. Based on the reasonable behavior of the few spot checks that we've performed here, however, let's see if changing the observational model addresses the issue.

4.2 Model 2

The gamma family of probability density functions are naturally complemented with the inverse gamma family of probability density functions. Because the gamma probability density functions exhibit heavier tails towards zero and lighter tails towards infinity their *peaks* skew towards larger values. On the other hand the inverse gamma probability density functions exhibit lighter tails towards zero and heavier tails towards infinity, resulting in *peaks* that skew towards smaller values. This conveniently contrasting behavior might be exactly what we need to address the retrodictive tension in our first model.

In order to build an inverse gamma observational model we need to engineer a location-dispersion parameterization. The inverse gamma family, like the gamma family, is typically parameterized in terms of a shape parameter α and a scale parameter β but we can also parameterize the family in terms of a location parameter

$$\mu = \text{mean}(\alpha, \beta) = \frac{\beta}{\alpha - 1}$$

and a dispersion parameter

$$\begin{aligned}\psi &= \frac{\text{variance}(\alpha, \beta)}{\text{mean}^2(\alpha, \beta)} \\ &= \frac{1}{\alpha - 2} \left(\frac{\beta}{\alpha - 1} \right)^2 \left(\frac{\alpha - 1}{\beta} \right)^2 \\ &= \frac{1}{\alpha - 2}.\end{aligned}$$

Let's try swapping the gamma observational model with an inverse gamma observational model.

```
fit <- stan(file="stan_programs/model2.stan",
            data=data, seed=8438338,
            warmup=1000, iter=2024, refresh=0)
```

The diagnostics again suggest some larger autocorrelations for the baseline η but nothing too problematic.

```
diagnostics2 <- util$extract_hmc_diagnostics(fit)
util$check_all_hmc_diagnostics(diagnostics2)
```

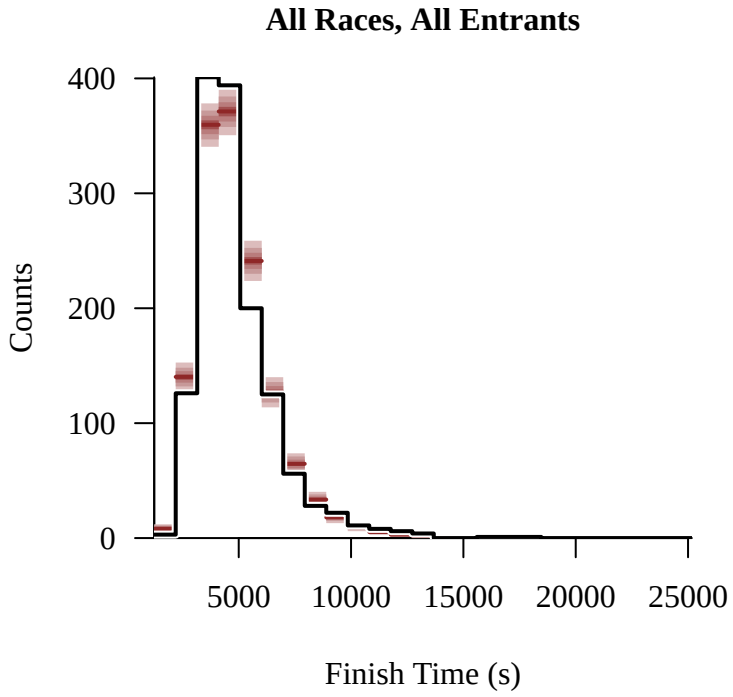
```
  All Hamiltonian Monte Carlo diagnostics are consistent with reliable
Markov chain Monte Carlo.
```

```
samples2 <- util$extract_expectand_vals(fit)
base_samples <- util$filter_expectands(samples2,
                                       c('eta',
                                         'rel_difficulties_free',
                                         'rel_skills_free',
                                         'psi'),
                                       check_arrays=TRUE)
util$check_all_expectand_diagnostics(base_samples)
```

```
eta:
  Chain 1: hat{ESS} (44.072) is smaller than desired (100).
  Chain 2: hat{ESS} (73.652) is smaller than desired (100).
  Chain 3: hat{ESS} (93.127) is smaller than desired (100).


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

It looks like this tweak to our model may have done the trick. The observed and posterior predictive behavior of the aggregate finish time histogram is a bit more consistent than it was in our first model.

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

util$plot_hist_quantiles(samples2, 'race_entrant_f_times_pred',
                         baseline_values=data$race_entrant_f_times,
                         xlab="Finish Time (s)",
                         main="All Races, All Entrants")
```



The retrodictive agreement in the individual race finish time histograms is similar to what we saw above. In particular no new retrodictive tensions have arisen.

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

for (r in c(7, 33, 77, 140)) {
  idxs <- data$race_f_start_idxs[r]:data$race_f_end_idxs[r]
  names <- sapply(idxs,
                  function(n) paste0('race_entrant_f_times_pred[', n, ']'))
  filtered_samples <- util$filter_expectands(samples2, names)
  util$plot_hist_quantiles(filtered_samples, 'race_entrant_f_times_pred',
                           1000, 11000, 1000,
```

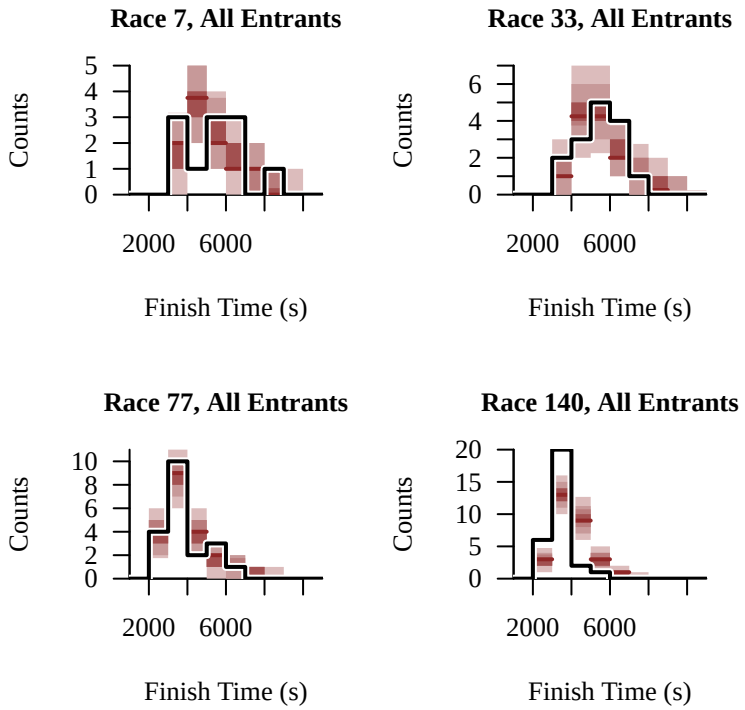
```
                            baseline_values=data$race_entrant_f_times[idxs],
                            xlab="Finish Time (s)",
                            main=paste0("Race ", r, ", All Entrants"))
}
```

```
Warning in check_bin_containment(bin_min, bin_max, collapsed_values,
"predictive value"): 84 predictive values (0.2%) fell above the binning.
```

```
Warning in check_bin_containment(bin_min, bin_max, collapsed_values,
"predictive value"): 137 predictive values (0.2%) 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"): 1 predictive value (0.0%) fell above the binning.
```

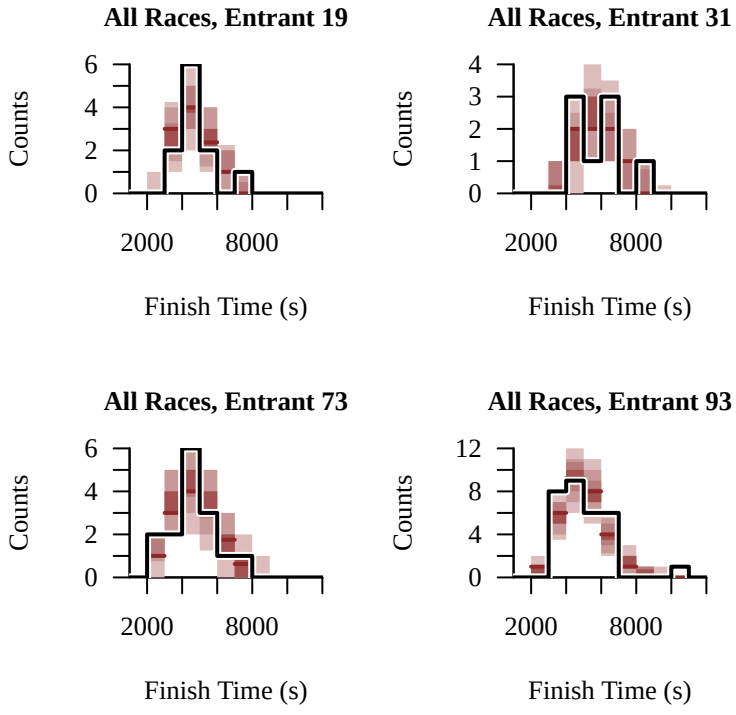


Interestingly the heavier tail of the inverse gamma family appears to allow the posterior predictive behavior for entrant 93 to spread out further and better match the observed behavior.

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

for (e in c(19, 31, 73, 93)) {
  idxs <- which(data$race_entrant_f_idxs == e)
  names <- sapply(idxs,
                  function(n) paste0('race_entrant_f_times_pred[', n, ']'))
  filtered_samples <- util$filter_expectands(samples2, names)
  util$plot_hist_quantiles(filtered_samples, 'race_entrant_f_times_pred',
                           1000, 12000, 1000,
                           baseline_values=data$race_entrant_f_times[idxs],
                           xlab="Finish Time (s)",
                           main=paste0("All Races, Entrant ", e))
}
```

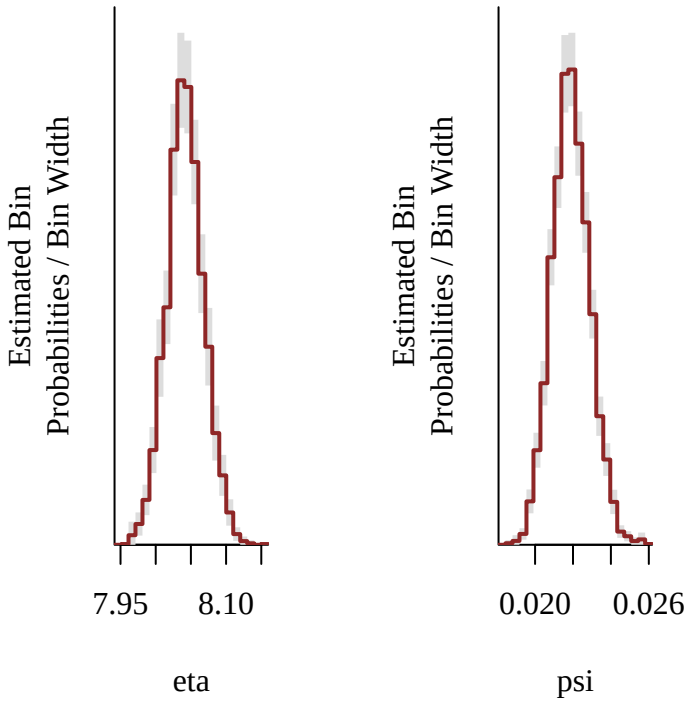
```
Warning in check_bin_containment(bin_min, bin_max, collapsed_values,
"predictive value"): 6 predictive values (0.0%) fell above the binning.
```

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



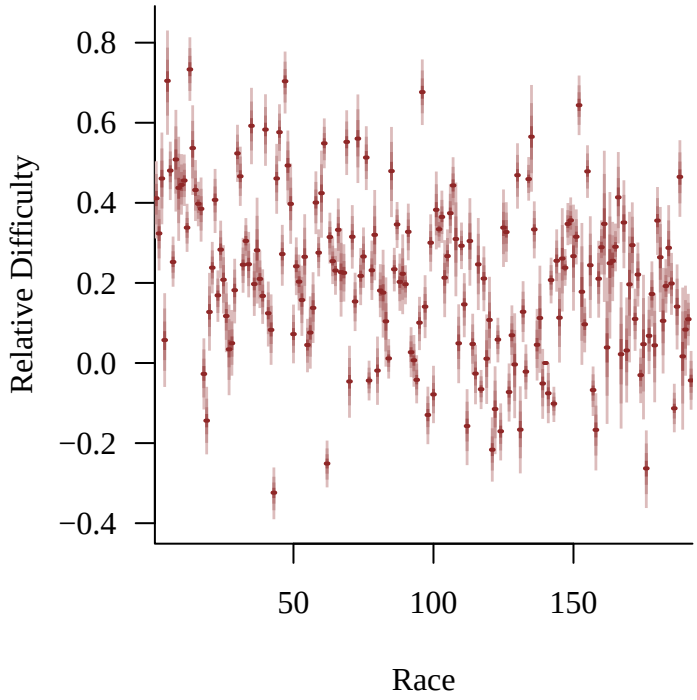
With no immediate reason to doubt our modeling assumptions we can finally move on to investigating our posterior inferences. The marginal posterior distributions for η and ψ look reasonable, with both strongly contracting within the prior model.

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

util$plot_expectand_pushforward(samples2[['eta']], 20,
                                display_name="eta")

util$plot_expectand_pushforward(samples2[['psi']], 20,
                                display_name="psi")
```



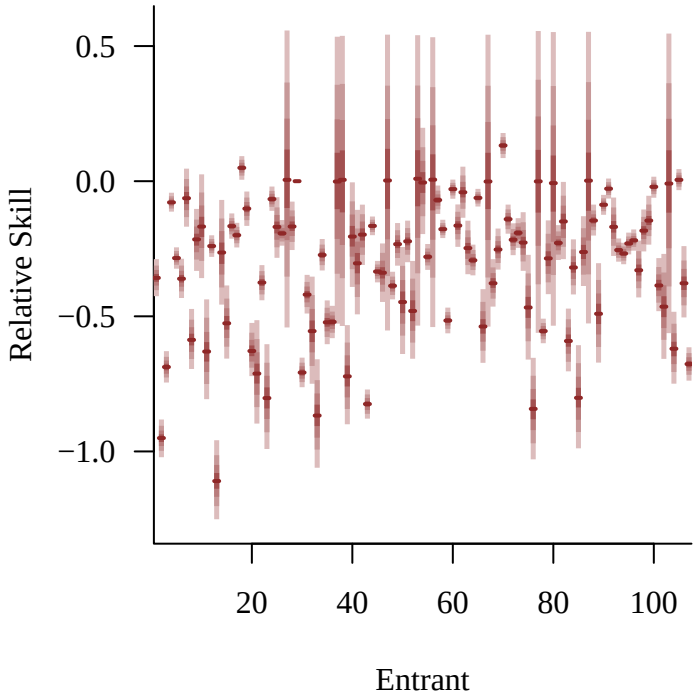
While the values of the relative seed difficulties all seem reasonable there does appear to be an unexpected pattern across the races. Initially the difficulties systematically decay before flatting out.

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

names <- sapply(1:data$N_races,
                function(r) paste0('rel_difficulties[', r,']'))
util$plot_disc_pushforward_quantiles(samples2, names,
                                     xlab="Race",
                                     ylab="Relative Difficulty")
```



On the other hand the relative entrant skills exhibit both reasonable values and no systematic patterns.

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

names <- sapply(1:data$N_entrants,
                function(n) paste0('rel_skills[', n,']'))
util$plot_disc_pushforward_quantiles(samples2, names,
                                     xlab="Entrant",
                                     ylab="Relative Skill")
```



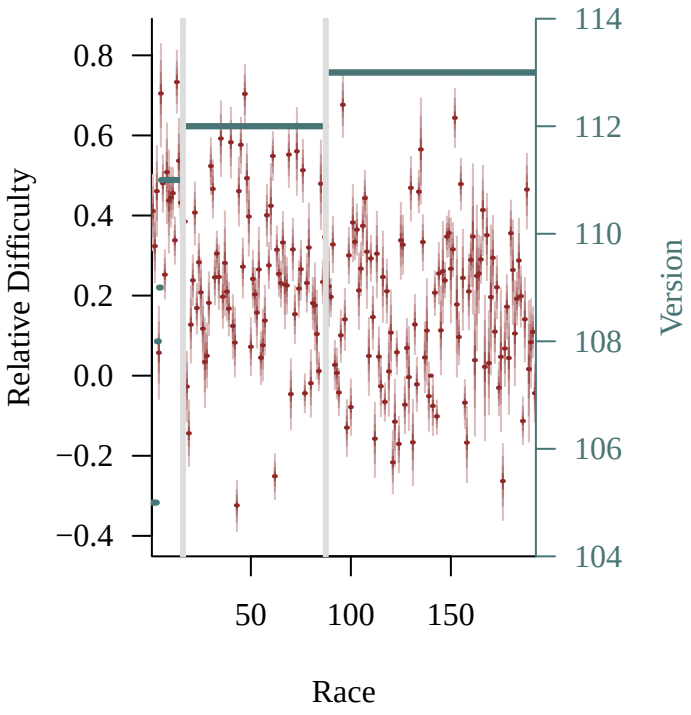
Let's go back to the relative difficulties and consider why we might see a pattern like that. Because the seeds for each race are ordered by the time at which the race occurred the pattern we see here may be due to a systematic change in seed difficulty over time.

One possibility is that the randomized maps are actually getting easier to complete. Another possibility is that our inferences for the seed difficulties are actually compensating for other time-dependent behaviors in these races that the model cannot otherwise accommodate. For example if the entire racing community was gradually getting better at the game then the entrant skills would improve with time. Because our model assumes static skills, however, the model could contort itself and absorb this time dependence into decreasing seed difficulties.

In order to distinguish between these possible hypotheses let's dive into this inferential behavior a bit deeper. If the MapRando code were static then it would be natural to assume that the seed difficulties scatter around some constant baseline. The MapRando code, however, is not static and has in fact undergone consistent develop throughout 2024. Fortunately the code version of each seed is included in our data, and we can visualize the MapRando development by overlaying the difficulties with the version numbers.

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

names <- sapply(1:data$N_races,
                function(r) paste0('rel_difficulties[', r,']'))
util$plot_disc_pushforward_quantiles(samples2, names,
                                     xlab="Race",
```

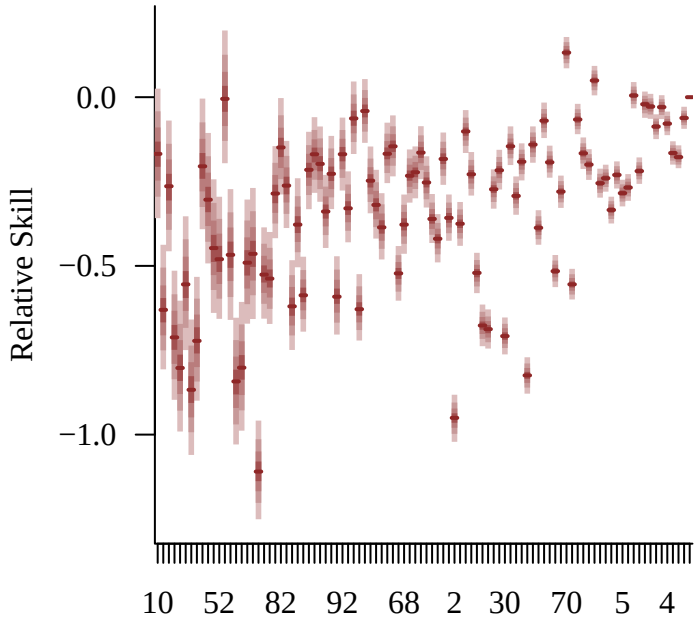
```
                                    ylab="Relative Difficulty")

text_versions <- c("105", "108", "109", "111",
                   "112 \\(DEV\\)", "112", "113 \\(DEV\\)", "113")
num_versions <- c(105, 108, 109, 111, 111.5, 112, 112.5, 113)
versions <- race_info$versions
for (n in seq_along(text_versions)) {
  versions <- gsub(text_versions[n], num_versions[n], versions)
}
versions <- as.numeric(versions)

par(new=TRUE)
plot(0, type='n', axes=FALSE, bty = "n",
     xlab = "", xlim=c(1, data$N_races),
     ylab = "", ylim=c(104, 114))

plot_xs <- sapply(1:data$N_races, function(r) c(r - 0.5, r + 0.5))
dim(plot_xs) <- c(1, 2 * data$N_races)

for (r in 1:data$N_races) {
  idx1 <- 2 * r - 1
  idx2 <- 2 * r
  lines(plot_xs[1, idx1:idx2], rep(versions[r], 2),
        col=util$c_mid_teal, lwd=3)
}

mtext("Version", side=4, col=util$c_mid_teal, line=3, las=0)
axis(4, ylim=c(104, 114), las=1,
     col=util$c_mid_teal, col.axis=util$c_mid_teal)

abline(v=16.5, col="#DDDDDD", lwd=3)
abline(v=87.5, col="#DDDDDD", lwd=3)
```



All of the prominent changes in seed difficulty over time seem to neatly line up with the transitions from one version to another. In hindsight this is completely reasonable: each version improves the randomization logic to be more consistent and easier for players to manage, especially in the earlier versions.

What about the hypothesis of improving entrant skills? If entrant skills were improving then it would be reasonable to expect systematic patterns between entrant skill and their overall experience with MapRando. While we do not have access to any exact quantification of experience we can consider proxies, such as the total number of race entrances. In particular while entrants might play MapRando, and gain experience, outside of official races that play time is likely to at least somewhat scale with the number of race entrances.

```
total_entrances <- table(data$race_entrant_f_idxs)
sorted_entrances <- as.data.frame(sort(total_entrances))

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

names <- sapply(sorted_entrances$Var1,
                function(n) paste0('rel_skills[', n, ']'))
util$plot_disc_pushforward_quantiles(samples2, names,
                                     xlab="Entrants Ordered By Total Entrances",
                                     xticklabs=sorted_entrances$Var1,
                                     ylab="Relative Skill")
```



The most striking pattern that we see is that the *uncertainty* in the entrant skill inferences decreases with increasing participation, which is just a consequence of having more data from which to learn. Beyond the decreasing uncertainty there might also be a mild increase in skill for the most experienced players.

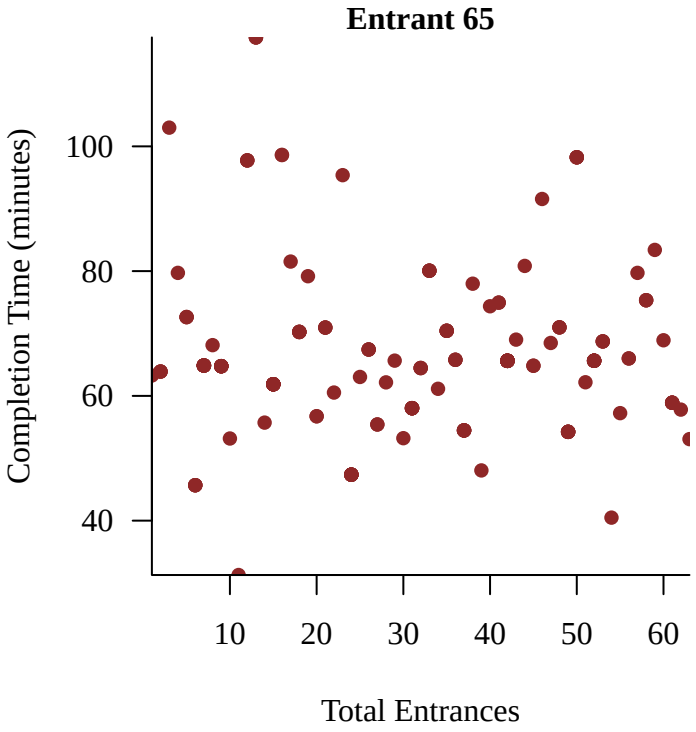
That said this increase is not necessarily tied to increased experience. For example entrant skills might be fixed with more skilled players just enjoying the MapRando races more and hence playing more.

In order to distinguish between these possibilities we need to start investigating how the behavior for a single entrant changes with increasing experience. If entrant skills increased enough, for instance, then we would see the finish times for a particular entrant systematically decrease with an increasing number of entrances.

Here let's look at entrant 65.

```
e <- 65
cum_completed_races <- c()
completion_times <- c()

for (r in 1:data$N_races) {
  N_previous_races <- length(cum_completed_races)

  entrant_idxs <- data$race_f_start_idxs[r]:data$race_f_end_idxs[r]
```

```
  if (e %in% data$race_entrant_f_idxs[entrant_idxs]) {
    entrant_idx <- which(data$race_entrant_f_idxs[entrant_idxs] == e)
    time <- data$race_entrant_f_times[data$race_f_start_idxs[r] + entrant_idx - 1]

    if (N_previous_races == 0) {
      cum_completed_races <- c(1)
    } else {
      cum_completed_races <- c(cum_completed_races,
                               cum_completed_races[N_previous_races] + 1)
    }
    completion_times <- c(completion_times, time)
  } else {
    if (N_previous_races > 0) {
      cum_completed_races <- c(cum_completed_races,
                               cum_completed_races[N_previous_races])
      completion_times <- c(completion_times,
                            completion_times[N_previous_races])
    }
  }
}

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

plot(cum_completed_races, completion_times / 60,
     pch=16, cex=1.0, col=util$c_dark,
     xlab="Total Entrances",
     ylab="Completion Time (minutes)",
     main=paste("Entrant", e))
```



While there might be a small reduction in the *variation* of finish times there doesn't seem to be any systematic increase or decrease in the mean. That's not to say that skills don't improve, just that they're not improving strongly enough to manifest in this particular visualization.

Overall the development of the MapRando code offers a satisfying explanation for the patterns we see in the inferred seed difficulties. That said it's always helpful to keep the other hypotheses in mind, especially if we are able to collect more data in the future.

4.3 Model 3

For our next model iteration let's consider forfeits. The danger with ignoring forfeits is that if the forfeit probability is coupled with entrant skill then inferences from the finish times alone will give us a biased view of those skills.

One possible assumption is that forfeits are completely random. For example entrants could forfeit mostly due to unexpected events that arise during each race that have nothing to do with their performance. In this case we could still extract information from the forfeit times

because we can lower bound what the finish time would have been,

$$\begin{aligned} p(t_{\text{forfeit}} \mid \mu_{se}, \psi) &= \pi(\, [t_{\text{forfeit}}, \infty) \, \mid \mu_{se}, \psi) \\ &= \int_0^{t_{\text{forfeit}}} \mathrm{d}t \, \text{inv-gamma}(t \mid \mu_{se}, \psi) \\ &= 1 - \Pi_{\text{inv-gamma}}(t_{\text{forfeit}} \mid \mu_{se}, \psi). \end{aligned}$$

Unfortunately while forfeit times are recorded they are difficult to programmatically access from `https://racetime.gg/smr`.

Forfeiting, however, is unlikely to be completely random. Entrants are more likely to forfeit when they're frustrated by the overall difficulty, for example when they get lost in a complex map layout or die at an inopportune point and lose too much progress. This suggests that $p(t_{\text{forfeit}})$ should depend on the contrast between seed difficulty and entrant skill,

$$p(t_{\text{forfeit}} \mid \lambda_{\text{difficulty},s}, \lambda_{\text{skill},e}) = f(\lambda_{\text{difficulty},s} - \lambda_{\text{skill},e}).$$

To start let's assume a logistic model,

$$p(t_{\text{forfeit}} \mid \lambda_{\text{difficulty},s}, \lambda_{\text{skill},e}, \kappa_e, \beta_e) = \text{logistic}(\beta_e \cdot ((\lambda_{\text{difficulty},s} - \lambda_{\text{skill},e}) - \kappa_e)),$$

where κ_e quantifies the threshold contrast where an entrant achieves a forfeit probability of $\frac{1}{2}$ and β_e quantifies how sensitive the forfeit probability is to the difference around this threshold. In order to ensure that a larger contrast always results in a higher forfeit we'll need to assume that β_e is limited to only positive values.

Note that this forfeit model is *another* pairwise comparison model! This time the contrasting item qualities are coupled to the forfeit probability with a discrimination parameter β_e in addition to a baseline parameter κ_e.

Beyond this functional form it's not straightforward to elicit domain expertise about reasonable values for κ_e and β_e. Here we'll take a more heuristic prior model that just constrains κ_e and β_e below five in order to avoid saturating the outputs of the logistic function too quickly.

Lastly once we explicitly model forfeits we are in a position to predict forfeits. This in turn provides new opportunities for retrodictive check summary statistics. In particular here we will consider the total number of forfeits in each race.

```
fit <- stan(file="stan_programs/model3.stan",
            data=data, seed=8438338,
            warmup=1000, iter=2024, refresh=0)
```

A few new warning have arisen suggesting that some of the κ_e exhibit heavier tails.

```
diagnostics3 <- util$extract_hmc_diagnostics(fit)
util$check_all_hmc_diagnostics(diagnostics3)
```

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

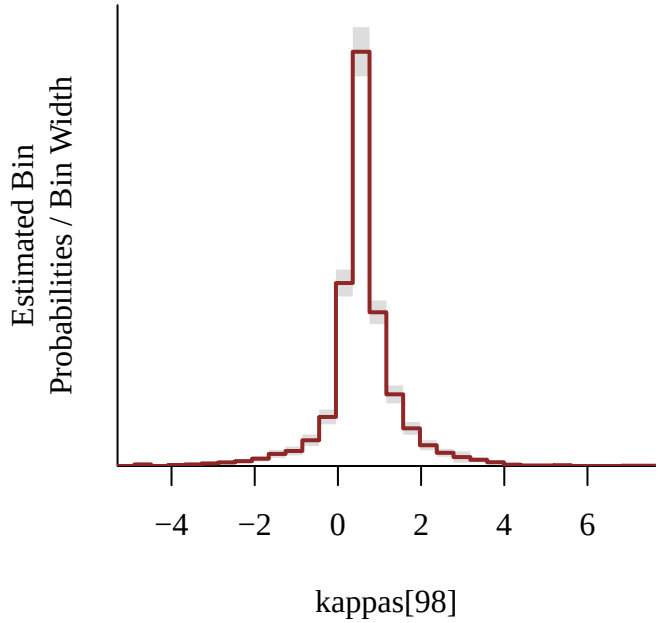
```
samples3 <- util$extract_expectand_vals(fit)
base_samples <- util$filter_expectands(samples3,
                                       c('eta',
                                         'rel_difficulties_free',
                                         'rel_skills_free',
                                         'psi',
                                         'kappas', 'betas'),
                                       check_arrays=TRUE)
util$check_all_expectand_diagnostics(base_samples)
```

```
eta:
  Chain 2: hat{ESS} (44.243) is smaller than desired (100).
  Chain 3: hat{ESS} (42.969) is smaller than desired (100).

rel_difficulties_free[77]:
  Chain 2: hat{ESS} (86.897) is smaller than desired (100).
  Chain 3: hat{ESS} (89.644) is smaller than desired (100).

rel_difficulties_free[102]:
  Chain 2: hat{ESS} (90.656) is smaller than desired (100).

rel_difficulties_free[117]:
  Chain 2: hat{ESS} (97.224) is smaller than desired (100).
  Chain 3: hat{ESS} (99.676) is smaller than desired (100).

rel_difficulties_free[134]:
  Chain 2: hat{ESS} (90.349) is smaller than desired (100).

rel_difficulties_free[142]:
  Chain 3: hat{ESS} (98.933) is smaller than desired (100).

kappas[6]:
  Chain 3: Right tail hat{xi} (0.251) exceeds 0.25.

kappas[7]:
```

```
  Chain 1: Left tail hat{xi} (0.305) exceeds 0.25.

kappas[44]:
  Chain 2: Right tail hat{xi} (0.307) exceeds 0.25.

kappas[61]:
  Chain 2: Left tail hat{xi} (0.283) exceeds 0.25.

kappas[94]:
  Chain 1: Right tail hat{xi} (0.274) exceeds 0.25.

kappas[98]:
  Chain 1: Both left and right tail hat{xi}s (0.354, 0.293) exceed 0.25.
  Chain 2: Left tail hat{xi} (0.349) exceeds 0.25.
  Chain 3: Left tail hat{xi} (0.343) exceeds 0.25.
  Chain 4: Both left and right tail hat{xi}s (0.446, 0.323) exceed 0.25.


Large tail hat{xi}s suggest that the expectand might not be
sufficiently integrable.

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

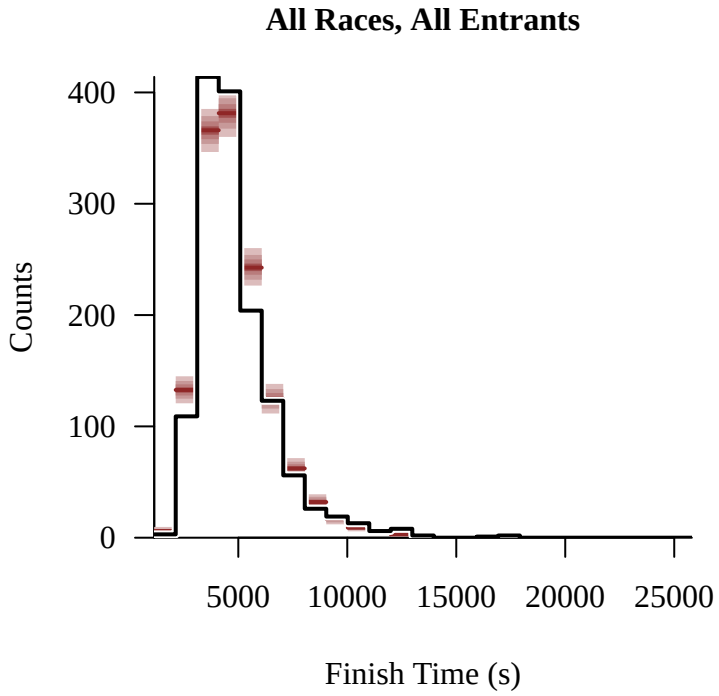
We can confirm this by examining the corresponding marginal posterior behaviors. That said these heavy tail warnings are of concern only if we attempt to estimate the expectation of the corresponding parameter functions, which will not in this analysis.

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

util$plot_expectand_pushforward(samples3[['kappas[98]']], 30,
                                display_name="kappas[98]")
```



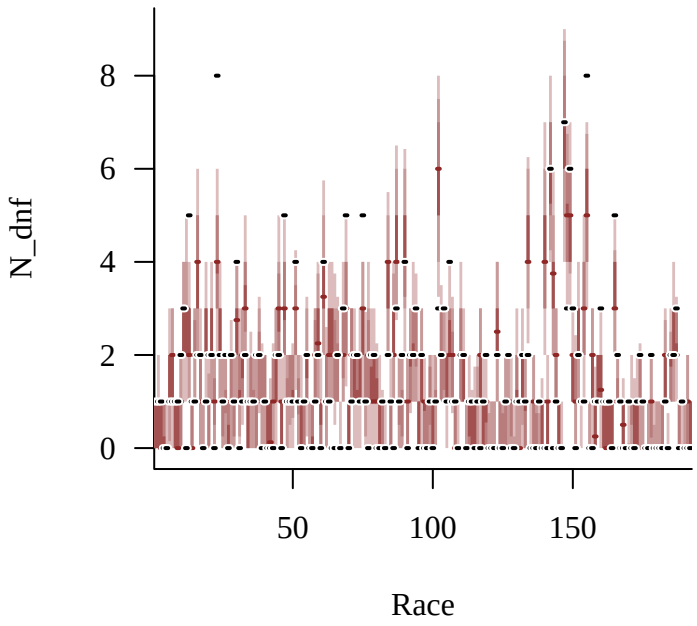
The retrodictive agreement between the observed and posterior predictive finish time histograms continues.

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

util$plot_hist_quantiles(samples3, 'race_entrant_f_times_pred',
                         baseline_values=data$race_entrant_f_times,
                         xlab="Finish Time (s)",
                         main="All Races, All Entrants")
```



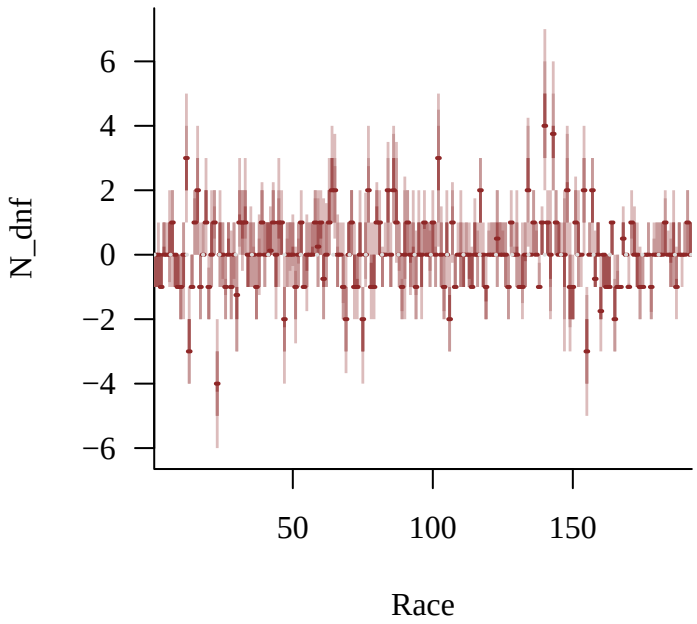
Now we can also consider the number of forfeits in each race. Fortunately the behavior of this statistic is also reasonably consistent.

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

names <- sapply(1:data$N_races,
                function(r) paste0('race_N_entrants_dnf_pred[', r,']'))
util$plot_disc_pushforward_quantiles(samples3, names,
                                     baseline_values=data$race_N_entrants_dnf,
                                     xlab="Race",
                                     ylab="N_dnf")
```



To make the comparison more clear we can always visualize the residuals and then compare to zero.

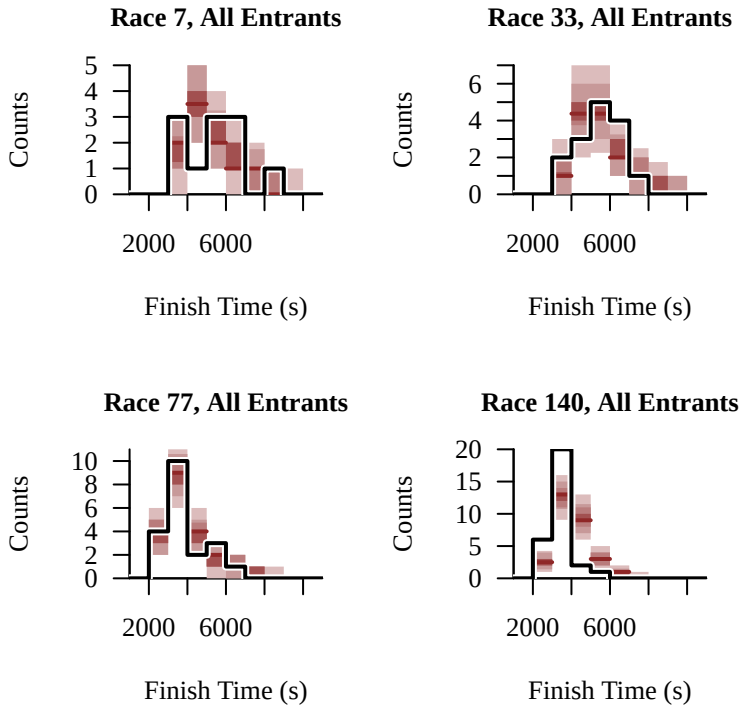
```
par(mfrow=c(1, 1), mar=c(5, 5, 3, 1))

names <- sapply(1:data$N_races,
                function(r) paste0('race_N_entrants_dnf_pred[', r,']'))
util$plot_disc_pushforward_quantiles(samples3, names,
                                     baseline_values=data$race_N_entrants_dnf,
                                     residual=TRUE,
                                     xlab="Race",
                                     ylab="N_dnf")
```



Finally the finish time histograms separated by selected races and entrants also show no signs of retrodictive tension.

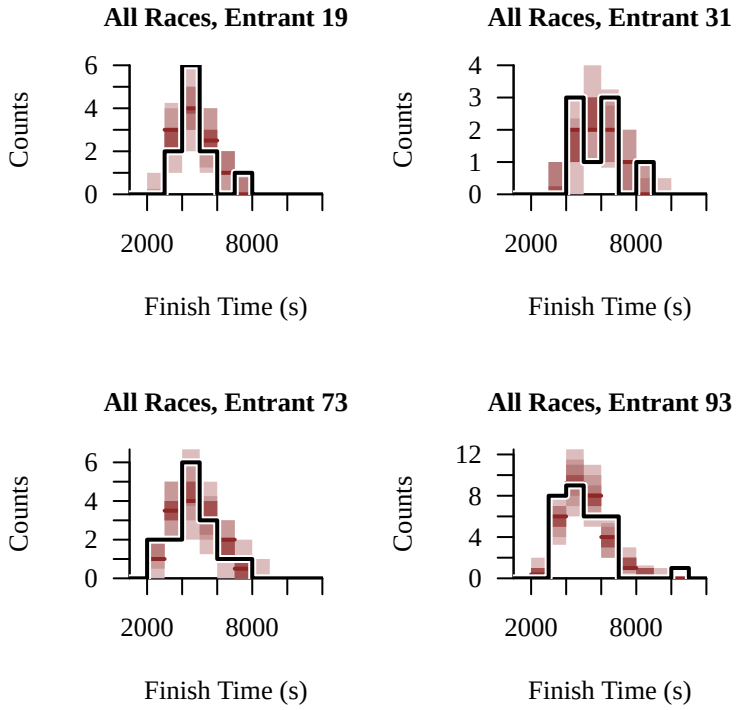
```
par(mfrow=c(2, 2), mar=c(5, 5, 3, 1))

for (r in c(7, 33, 77, 140)) {
  idxs <- data$race_f_start_idxs[r]:data$race_f_end_idxs[r]
  names <- sapply(idxs,
                  function(n) paste0('race_entrant_f_times_pred[', n, ']'))
  filtered_samples <- util$filter_expectands(samples3, names)
  util$plot_hist_quantiles(filtered_samples, 'race_entrant_f_times_pred',
                           1000, 11000, 1000,
                           baseline_values=data$race_entrant_f_times[idxs],
                           xlab="Finish Time (s)",
                           main=paste0("Race ", r, ", All Entrants"))
}
```

```
Warning in check_bin_containment(bin_min, bin_max, collapsed_values,
"predictive value"): 85 predictive values (0.2%) fell above the binning.

Warning in check_bin_containment(bin_min, bin_max, collapsed_values,
"predictive value"): 160 predictive values (0.3%) 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.
```



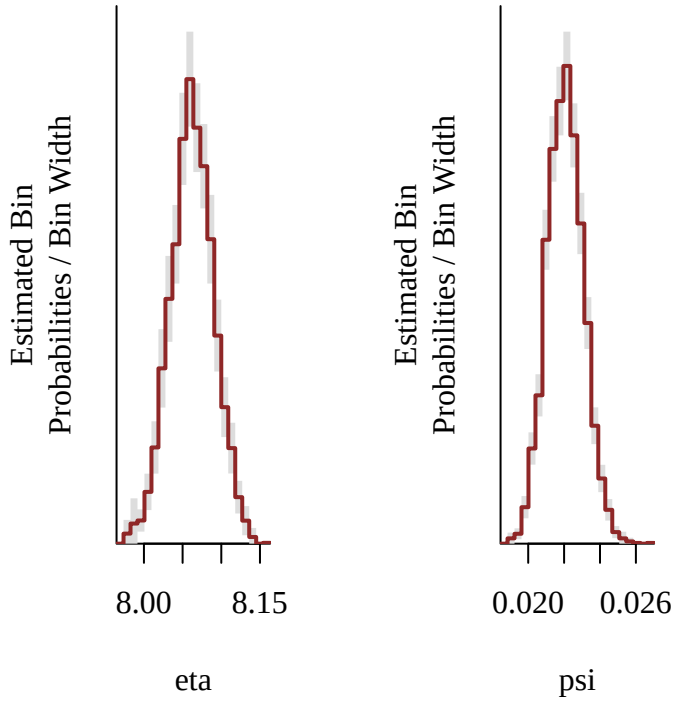

```
par(mfrow=c(2, 2), mar=c(5, 5, 3, 1))

for (e in c(19, 31, 73, 93)) {
  idxs <- which(data$race_entrant_f_idxs == e)
  names <- sapply(idxs,
                  function(n) paste0('race_entrant_f_times_pred[', n, ']'))
  filtered_samples <- util$filter_expectands(samples3, names)
  util$plot_hist_quantiles(filtered_samples, 'race_entrant_f_times_pred',
                           1000, 12000, 1000,
                           baseline_values=data$race_entrant_f_times[idxs],
                           xlab="Finish Time (s)",
                           main=paste0("All Races, Entrant ", e))
}
```

```
Warning in check_bin_containment(bin_min, bin_max, collapsed_values,
"predictive value"): 8 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"): 19 predictive values (0.0%) fell above the binning.
```

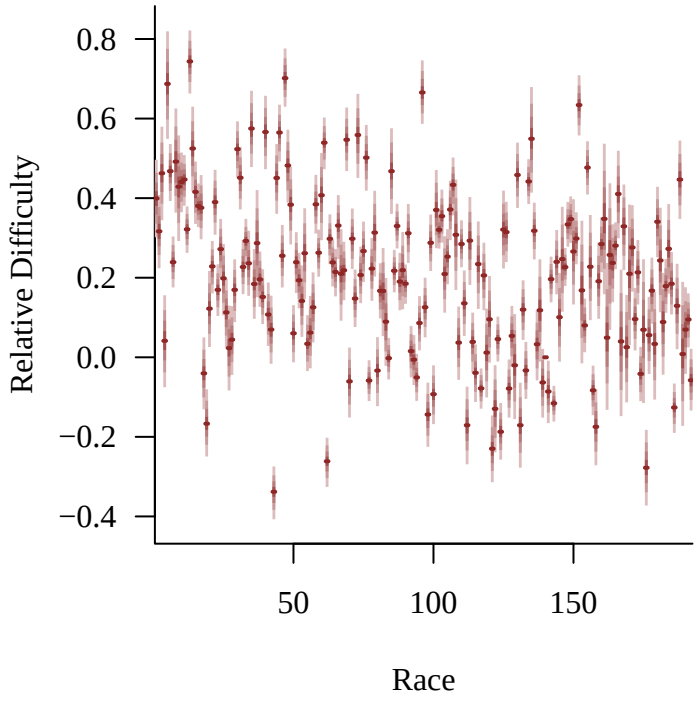


Without any concerns about our modeling assumptions we can move on to examining the resulting posterior inferences. Inferences for the existing parameters are at least superficially similar to those from the second model; we'll make a more direct comparison in Section 4.5.

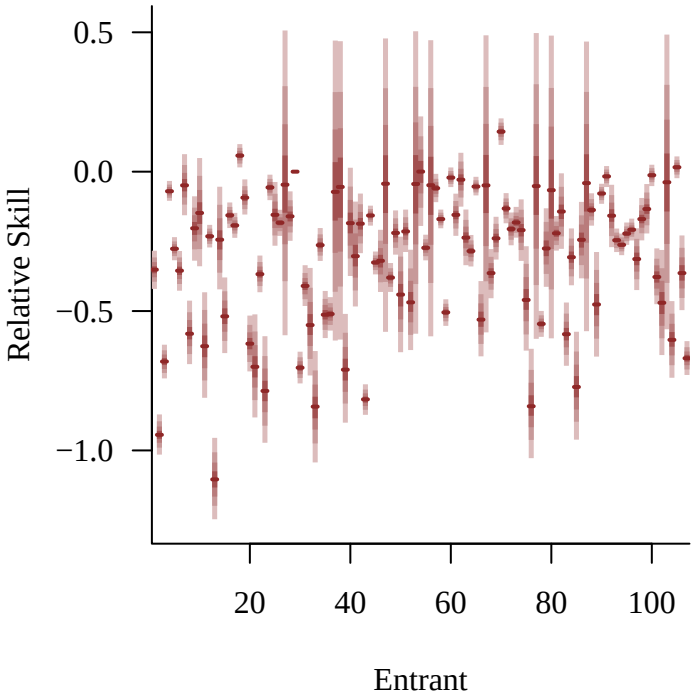
```
par(mfrow=c(1, 2), mar=c(5, 5, 1, 1))

util$plot_expectand_pushforward(samples3[['eta']], 20,
                                display_name="eta")

util$plot_expectand_pushforward(samples3[['psi']], 20,
                                display_name="psi")
```




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

names <- sapply(1:data$N_races,
                function(r) paste0('rel_difficulties[', r,']'))
util$plot_disc_pushforward_quantiles(samples3, names,
                                     xlab="Race",
                                     ylab="Relative Difficulty")
```

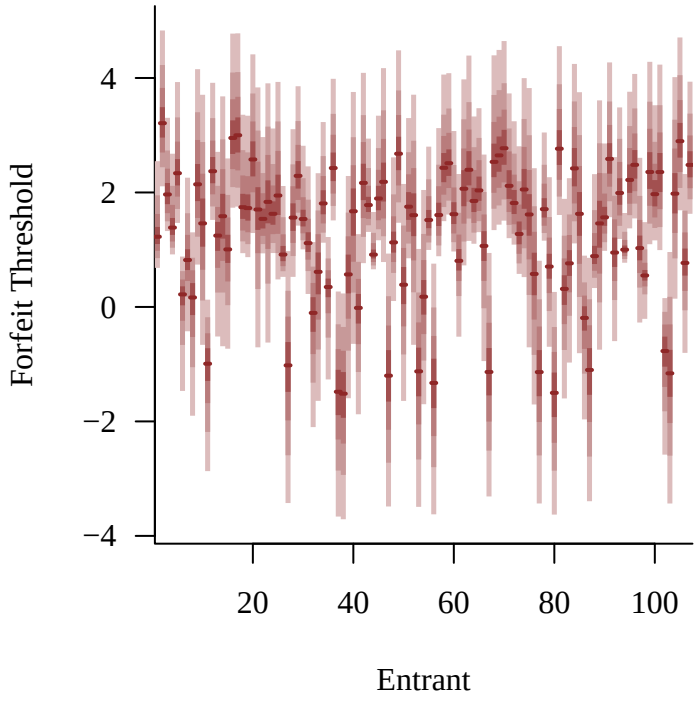



```
par(mfrow=c(1, 1), mar=c(5, 5, 1, 1))

names <- sapply(1:data$N_entrants,
                function(n) paste0('rel_skills[', n,']'))
util$plot_disc_pushforward_quantiles(samples3, names,
                                     xlab="Entrant",
                                     ylab="Relative Skill")
```

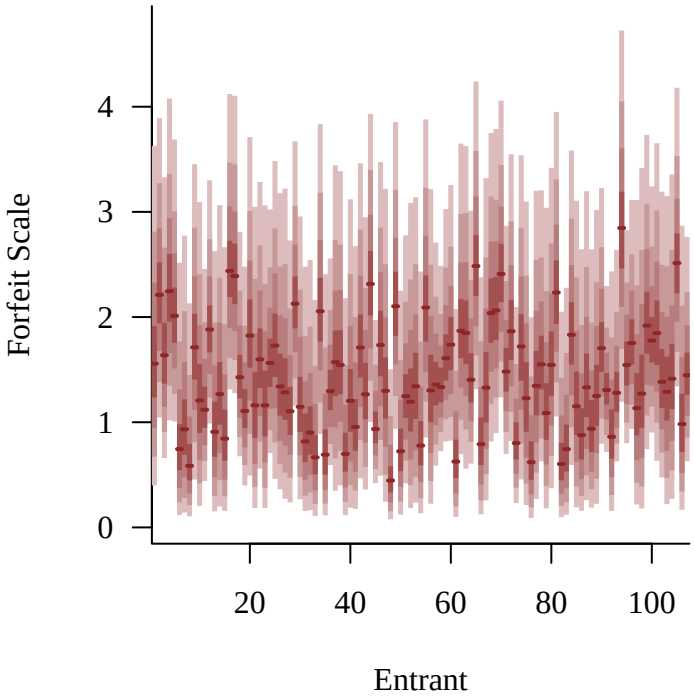


More relevant for this latest model iteration are the posterior inferences for the new, forfeit-related parameters. Overall the uncertainties are relatively large but we can pick out a few exceptional behaviors

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

names <- sapply(1:data$N_entrants,
                function(n) paste0('kappas[', n,']'))
util$plot_disc_pushforward_quantiles(samples3, names,
                                     xlab="Entrant",
                                     ylab="Forfeit Threshold")
```

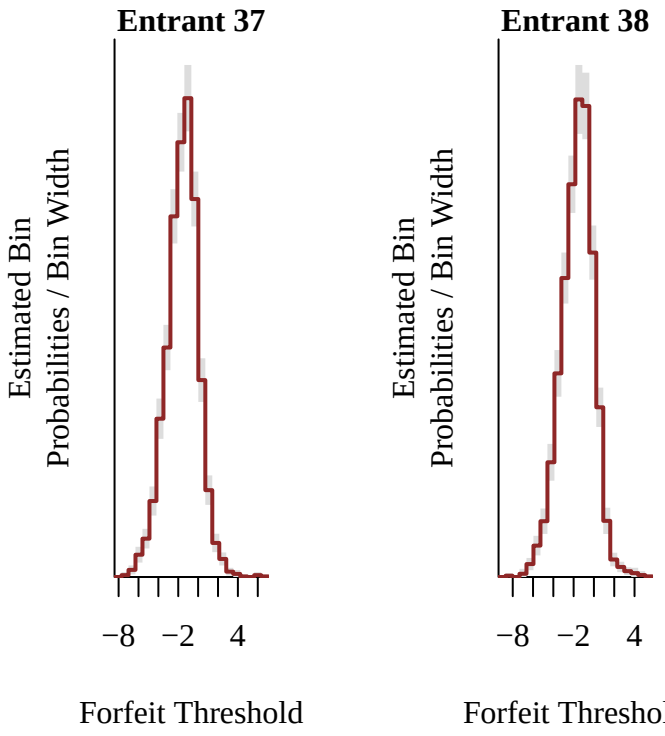



```
par(mfrow=c(1, 1), mar=c(5, 5, 1, 1))

names <- sapply(1:data$N_entrants,
                function(n) paste0('betas[', n,']'))
util$plot_disc_pushforward_quantiles(samples3, names,
                                     xlab="Entrant",
                                     ylab="Forfeit Scale")
```



For example posterior inferences of the forfeit thresholds for entrants 37 and 38 both concentrate on negative values.

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

e <- 37
name <- paste0('kappas[', e, ']')
util$plot_expectand_pushforward(samples3[[name]], 20,
                                display_name="Forfeit Threshold",
                                main=paste('Entrant', e))

e <- 38
name <- paste0('kappas[', e, ']')
util$plot_expectand_pushforward(samples3[[name]], 20,
                                display_name="Forfeit Threshold",
                                main=paste('Entrant', e))
```



Both of these entrants forfeited every race they entered.

```
summarize_entrant <- function(e) {
  N <- N_entrant_f_races[e] + N_entrant_dnf_races[e]
  Nf <- N_entrant_f_races[e]
  Ndnf <- N_entrant_dnf_races[e]

  cat(sprintf("Entrant %i\n", e))
  if (N > 1)
    cat(sprintf("  %i total entrances\n", N))
  else
    cat(sprintf("  %i total entrance\n", N))

  if (Nf > 1)
    cat(sprintf("  %i finishes (%.1f%%)\n", Nf, 100 * Nf / N))
  else if (Nf == 1)
    cat(sprintf("  %i finish (%.1f%%)\n", Nf, 100 * Nf / N))

  if (Ndnf > 1)
    cat(sprintf("  %i forfeits (%.1f%%)\n", Ndnf, 100 * Ndnf / N))
  else if (Ndnf == 1)
    cat(sprintf("  %i forfeit (%.1f%%)\n", Ndnf, 100 * Ndnf / N))
```

```
}
```

```
summarize_entrant(37)
```

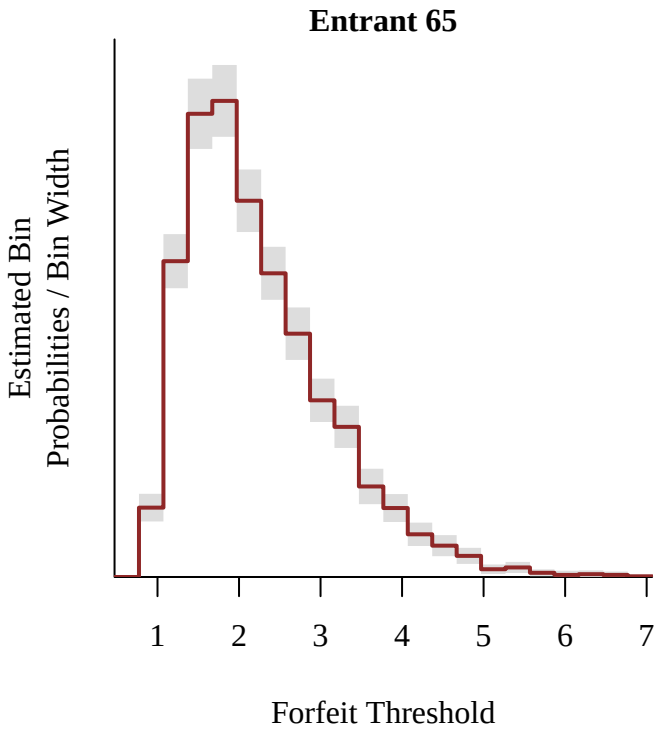
```
Entrant 37
  2 total entrances
  2 forfeits (100.0%)
```

```
summarize_entrant(38)
```

```
Entrant 38
  2 total entrances
  2 forfeits (100.0%)
```

On the other hand posterior inferences of the forfeit threshold for entrant 65 concentrates on positive values.

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

e <- 65
name <- paste0('kappas[', e, ']')
util$plot_expectand_pushforward(samples3[[name]], 20,
                                display_name="Forfeit Threshold",
                                main=paste('Entrant', e))
```



This entrant forfeited only once out of 64 total entrances.

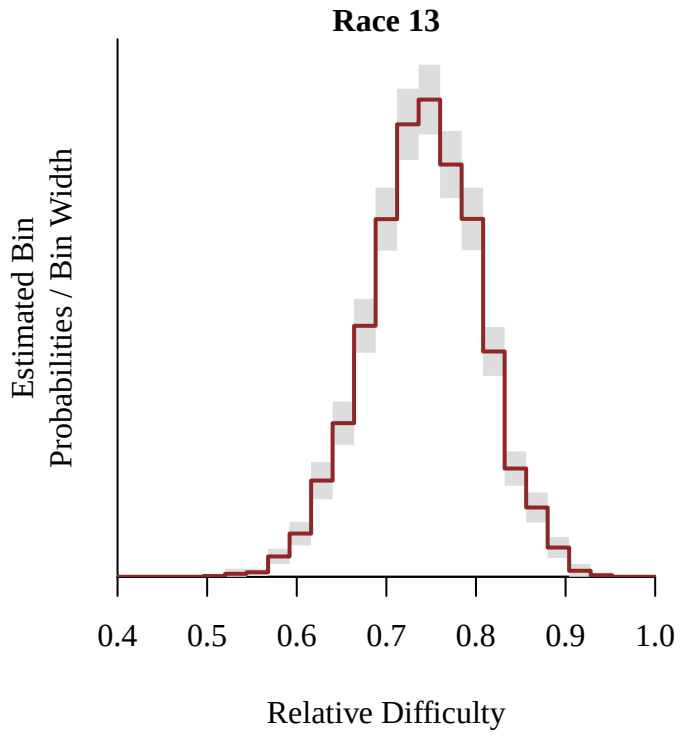
```
summarize_entrant(e)
```

```
Entrant 65
  64 total entrances
  63 finishes (98.4%)
  1 forfeit (1.6%)
```

Moreover that forfeit occurred for a particularly difficult seed, pushing the consistent forfeit threshold behaviors to larger values.

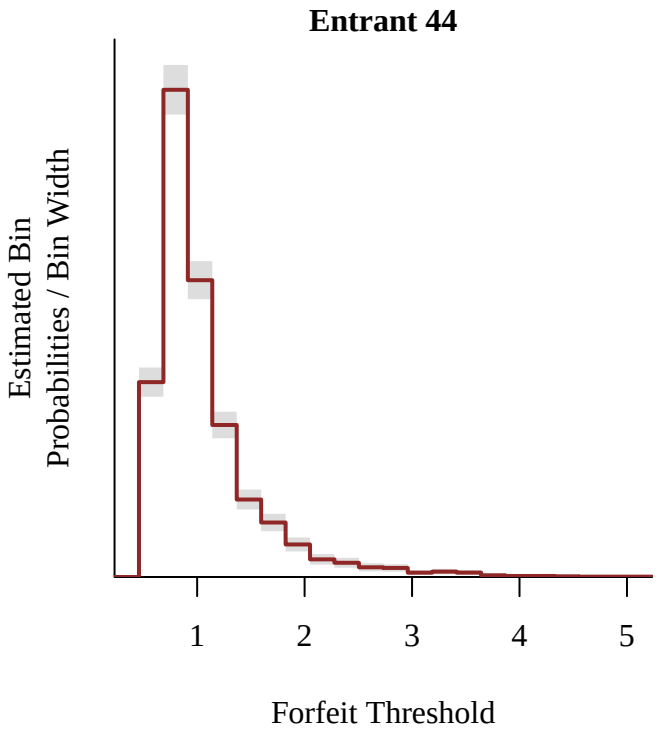
```
dnf_races <- c()
for (r in 1:data$N_races) {
  if (data$race_N_entrants_dnf[r] == 0) next
  idxs <- data$race_dnf_start_idxs[r]:data$race_dnf_end_idxs[r]
  if (e %in% data$race_entrant_dnf_idxs[idxs])
    dnf_races <- c(dnf_races, r)
}

name <- paste0('rel_difficulties[', dnf_races[1], ']')
```

```
util$plot_expectand_pushforward(samples3[[name]], 25, flim=c(0.4, 1),
                                display_name="Relative Difficulty",
                                main=paste('Race', dnf_races[1]))
```



Finally the posterior inferences of the forfeit threshold for entrant 44 mostly concentrate on values between 0 and 1.

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

e <- 44
name <- paste0('kappas[', e, ']')
util$plot_expectand_pushforward(samples3[[name]], 20,
                                display_name="Forfeit Threshold",
                                main=paste('Entrant', e))
```



While entrant 44 finishes most of their entrances forfeits are not uncommon.

```
summarize_entrant(e)
```

```
Entrant 44
  71 total entrances
  56 finishes (78.9%)
  15 forfeits (21.1%)
```

This higher propensity to forfeit suppresses larger values of the forfeit threshold.

Overall our posterior inferences for the forfeit behavior are reasonable, but the relative scarcity of forfeits prevents us from resolving that behavior with too much precision.

4.4 Model 4

A natural extension of the last model is to couple the behavior across individual race seeds and entrants. This would allow data to be shared across entrances and potentially reducing inferential uncertainties, especially for races and entrants with few entrances to inform them directly. In particular if our domain expertise about these behaviors is exchangeable then we can couple them together with hierarchical models. As a side benefit we can also use

the inferred hierarchical population behavior to make inferences and predictions about new, hypothetical seeds and entrants.

Because the MapRando version distinguishes some seeds from each other not all of the seed difficulties are exchangeable. That said we don't have any information to discriminate between the seeds *within* a version, suggesting conditional exchangeabilty. In other words we can couple the seed difficulties within each MapRando version together into separate hierarchical models.

For programmatic convenience we'll just need to convert the version numbers into sequential indices.

```
uniq_versions <- unique(race_info$versions)
data$N_versions <- length(uniq_versions)
data$version_idxs <- as.numeric(factor(race_info$versions,
                                       levels=uniq_versions,
                                       labels=1:data$N_versions))
```

On the other hand we don't have any prior information capable of discriminating between the entrants, at least not without doing additional research into their experience with Super Metroid® in general and MapRando in particular. Consequently all of the entrant behaviors are exchangeable with each other and can be captured with a single hierarchy. For simplicity I will couple only the entrant skills together, leaving the heterogeneous entrant forfeit behaviors independent of each other.

Because the relative seed difficulties and entrant skills are modeled with one-dimensional, and unconstrained, real values we can reach for the standard normal hierarchical model to start. All we need then to implement the model is a parameterization of the individual parameters in each hierarchy. Here let's start with a monolithic non-centered parameterizations for all of the hierarchies, hoping that the large number of seeds and entrants will result in strong enough regularization to suppress any problematic degeneracies. In the worst case our computational diagnostics will indicate if we need to consider more sophisticated parameterizations.

```
fit <- stan(file="stan_programs/model4a.stan",
            data=data, seed=8438338,
            warmup=1000, iter=2024, refresh=0)
```

We don't see any of the tell-tale signs of problematic hierarchical geometries, such as divergences and E-FMI warnings, but the persistent empirical effective sample size warnings across across the non-centered entrant skill parameters suggest that a centered parameterization of the entrant skill hierarchy might perform better.

```
diagnostics4 <- util$extract_hmc_diagnostics(fit)
util$check_all_hmc_diagnostics(diagnostics4)
```

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

```
samples4 <- util$extract_expectand_vals(fit)
base_samples <- util$filter_expectands(samples4,
                                       c('eta',
                                         'rel_difficulties_free_tilde',
                                         'mu_rel_difficulties',
                                         'tau_rel_difficulties',
                                         'rel_skills_free_tilde',
                                         'mu_rel_skills',
                                         'tau_rel_skills',
                                         'psi',
                                         'kappas', 'betas'),
                                       check_arrays=TRUE)
util$check_all_expectand_diagnostics(base_samples)
```

```
rel_skills_free_tilde[4]:
  Chain 3: hat{ESS} (78.199) is smaller than desired (100).
  Chain 4: hat{ESS} (98.670) is smaller than desired (100).

rel_skills_free_tilde[5]:
  Chain 3: hat{ESS} (70.555) is smaller than desired (100).

rel_skills_free_tilde[12]:
  Chain 3: hat{ESS} (82.799) is smaller than desired (100).

rel_skills_free_tilde[16]:
  Chain 3: hat{ESS} (83.127) is smaller than desired (100).

rel_skills_free_tilde[17]:
  Chain 3: hat{ESS} (80.187) is smaller than desired (100).

rel_skills_free_tilde[26]:
  Chain 3: hat{ESS} (76.101) is smaller than desired (100).

rel_skills_free_tilde[43]:
  Chain 3: hat{ESS} (86.035) is smaller than desired (100).
```

```
  Chain 4: hat{ESS} (91.771) is smaller than desired (100).

rel_skills_free_tilde[44]:
  Chain 3: hat{ESS} (76.720) is smaller than desired (100).

rel_skills_free_tilde[54]:
  Chain 3: hat{ESS} (58.138) is smaller than desired (100).

rel_skills_free_tilde[56]:
  Chain 4: hat{ESS} (96.016) is smaller than desired (100).

rel_skills_free_tilde[57]:
  Chain 3: hat{ESS} (64.781) is smaller than desired (100).
  Chain 4: hat{ESS} (86.086) is smaller than desired (100).

rel_skills_free_tilde[59]:
  Chain 3: hat{ESS} (72.204) is smaller than desired (100).
  Chain 4: hat{ESS} (93.851) is smaller than desired (100).

rel_skills_free_tilde[64]:
  Chain 3: hat{ESS} (58.798) is smaller than desired (100).
  Chain 4: hat{ESS} (93.856) is smaller than desired (100).

rel_skills_free_tilde[89]:
  Chain 3: hat{ESS} (71.703) is smaller than desired (100).
  Chain 4: hat{ESS} (91.745) is smaller than desired (100).

rel_skills_free_tilde[90]:
  Chain 3: hat{ESS} (83.120) is smaller than desired (100).

rel_skills_free_tilde[92]:
  Chain 3: hat{ESS} (86.610) is smaller than desired (100).

rel_skills_free_tilde[93]:
  Chain 3: hat{ESS} (77.070) is smaller than desired (100).

rel_skills_free_tilde[94]:
  Chain 3: hat{ESS} (75.684) is smaller than desired (100).

rel_skills_free_tilde[95]:
  Chain 3: hat{ESS} (55.985) is smaller than desired (100).
  Chain 4: hat{ESS} (98.756) is smaller than desired (100).
```

```
rel_skills_free_tilde[99]:
  Chain 3: hat{ESS} (88.862) is smaller than desired (100).

rel_skills_free_tilde[104]:
  Chain 3: hat{ESS} (81.761) is smaller than desired (100).

mu_rel_skills:
  Chain 3: hat{ESS} (86.111) is smaller than desired (100).
  Chain 4: hat{ESS} (97.419) is smaller than desired (100).

kappas[31]:
  Chain 3: Left tail hat{xi} (0.269) exceeds 0.25.
  Chain 4: Left tail hat{xi} (0.333) exceeds 0.25.

kappas[44]:
  Chain 1: Right tail hat{xi} (0.276) exceeds 0.25.

kappas[48]:
  Chain 2: Left tail hat{xi} (0.270) exceeds 0.25.
  Chain 3: Left tail hat{xi} (0.292) exceeds 0.25.

kappas[94]:
  Chain 3: Right tail hat{xi} (0.284) exceeds 0.25.
  Chain 1: hat{ESS} (61.972) is smaller than desired (100).

kappas[98]:
  Chain 1: Both left and right tail hat{xi}s (0.260, 0.256) exceed 0.25.
  Chain 2: Left tail hat{xi} (0.337) exceeds 0.25.
  Chain 3: Both left and right tail hat{xi}s (0.373, 0.310) exceed 0.25.
  Chain 4: Both left and right tail hat{xi}s (0.327, 0.268) exceed 0.25.


Large tail hat{xi}s suggest that the expectand might not be
sufficiently integrable.

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

Fortunately this looks to be exactly the case.

```
fit <- stan(file="stan_programs/model4b.stan",
            data=data, seed=8438338,
            warmup=1000, iter=2024, refresh=0)
```

```
diagnostics4 <- util$extract_hmc_diagnostics(fit)
util$check_all_hmc_diagnostics(diagnostics4)
```

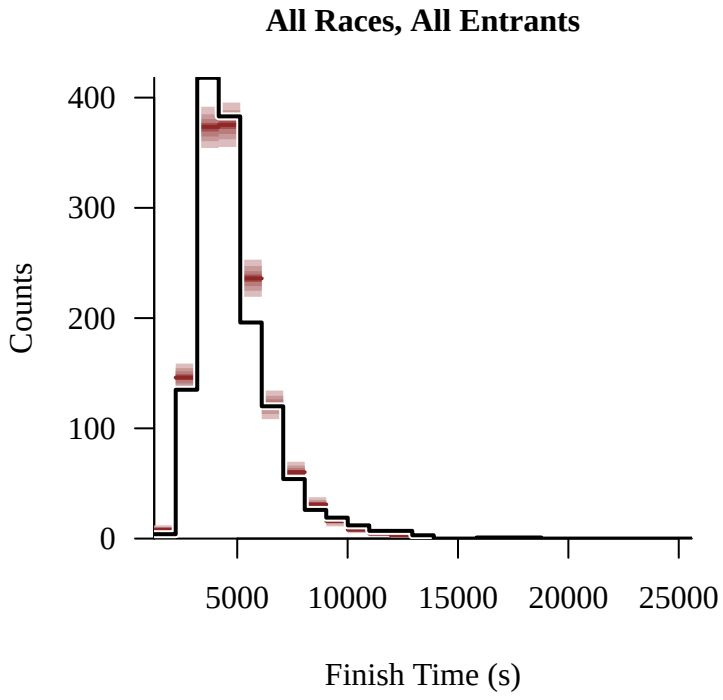
```
  All Hamiltonian Monte Carlo diagnostics are consistent with reliable
Markov chain Monte Carlo.
```

```
samples4 <- util$extract_expectand_vals(fit)
base_samples <- util$filter_expectands(samples4,
                                       c('eta',
                                         'rel_difficulties_free_tilde',
                                         'mu_rel_difficulties',
                                         'tau_rel_difficulties',
                                         'rel_skills_free',
                                         'mu_rel_skills',
                                         'tau_rel_skills',
                                         'psi',
                                         'kappas', 'betas'),
                                       check_arrays=TRUE)
util$check_all_expectand_diagnostics(base_samples)
```

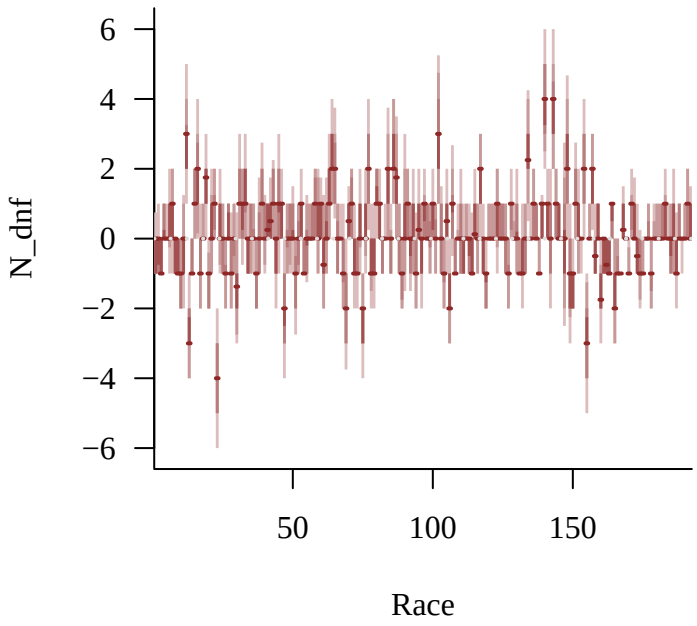
```
kappas[31]:
  Chain 4: Left tail hat{xi} (0.325) exceeds 0.25.

kappas[44]:
  Chain 1: Right tail hat{xi} (0.271) exceeds 0.25.
  Chain 2: Right tail hat{xi} (0.253) exceeds 0.25.

kappas[94]:
  Chain 2: Right tail hat{xi} (0.374) exceeds 0.25.

kappas[98]:
  Chain 1: Both left and right tail hat{xi}s (0.354, 0.337) exceed 0.25.
  Chain 3: Left tail hat{xi} (0.373) exceeds 0.25.
  Chain 4: Left tail hat{xi} (0.295) exceeds 0.25.


Large tail hat{xi}s suggest that the expectand might not be
sufficiently integrable.
```

A review of our visual retrodictive checks doesn't show any indications that the introduction of the hierarchical coupling compromised the adequacy of our modeling assumptions.

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

util$plot_hist_quantiles(samples4, 'race_entrant_f_times_pred',
                         baseline_values=data$race_entrant_f_times,
                         xlab="Finish Time (s)",
                         main="All Races, All Entrants")
```

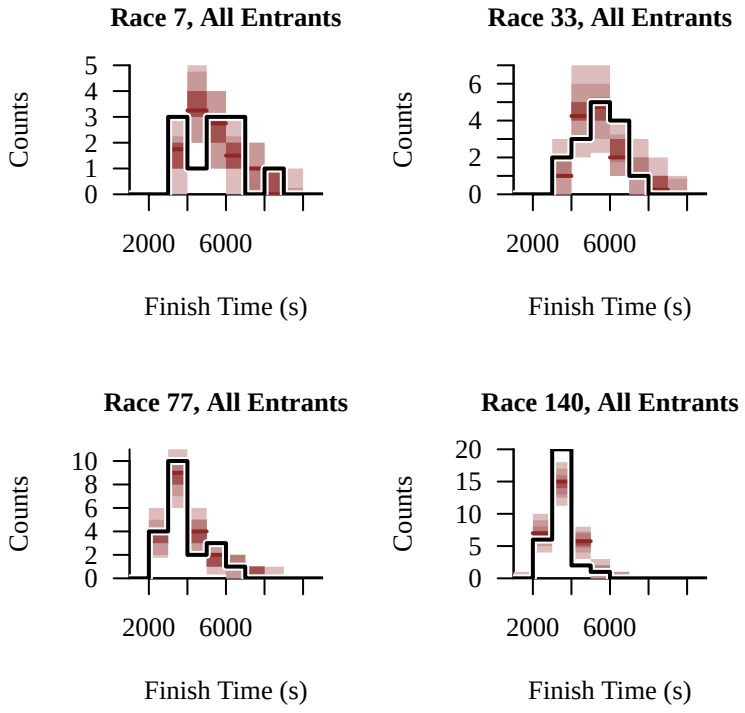



```
par(mfrow=c(1, 1), mar=c(5, 5, 3, 1))

names <- sapply(1:data$N_races,
                function(r) paste0('race_N_entrants_dnf_pred[', r,']'))
util$plot_disc_pushforward_quantiles(samples4, names,
                                     baseline_values=data$race_N_entrants_dnf,
                                     residual=TRUE,
                                     xlab="Race",
                                     ylab="N_dnf")
```




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

for (r in c(7, 33, 77, 140)) {
  idxs <- data$race_f_start_idxs[r]:data$race_f_end_idxs[r]
  names <- sapply(idxs,
                  function(n)
                  paste0('race_entrant_f_times_pred[', n, ']'))
  filtered_samples <- util$filter_expectands(samples4, names)
  util$plot_hist_quantiles(filtered_samples, 'race_entrant_f_times_pred',
                           1000, 11000, 1000,
                           baseline_values=data$race_entrant_f_times[idxs],
                           xlab="Finish Time (s)",
                           main=paste0("Race ", r, ", All Entrants"))
}
```

```
Warning in check_bin_containment(bin_min, bin_max, collapsed_values,
"predictive value"): 104 predictive values (0.2%) fell above the binning.

Warning in check_bin_containment(bin_min, bin_max, collapsed_values,
"predictive value"): 109 predictive values (0.2%) 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.
```

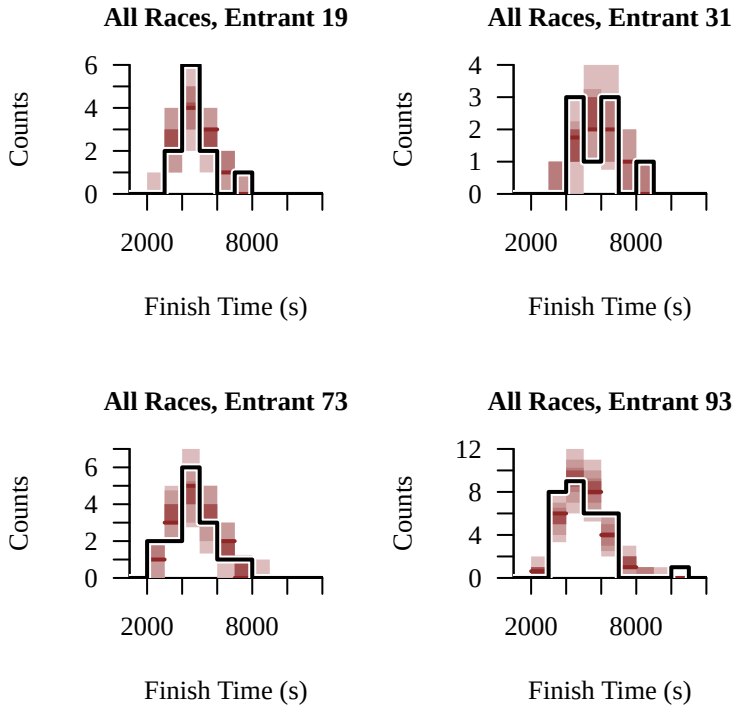



```
par(mfrow=c(2, 2), mar=c(5, 5, 3, 1))

for (e in c(19, 31, 73, 93)) {
  idxs <- which(data$race_entrant_f_idxs == e)
  names <- sapply(idxs,
                  function(n)
                  paste0('race_entrant_f_times_pred[', n, ']'))
  filtered_samples <- util$filter_expectands(samples4, names)
  util$plot_hist_quantiles(filtered_samples, 'race_entrant_f_times_pred',
                           1000, 12000, 1000,
                           baseline_values=data$race_entrant_f_times[idxs],
                           xlab="Finish Time (s)",
                           main=paste0("All Races, Entrant ", e))
}
```

```
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"): 14 predictive values (0.0%) fell above the binning.
```

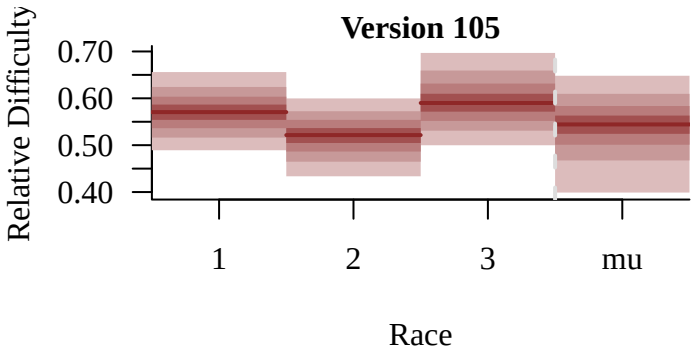


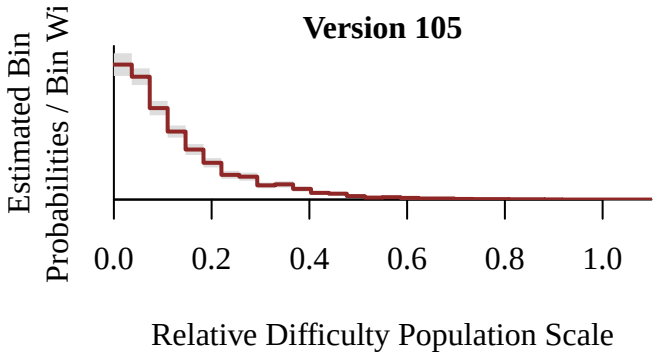
Now we can explore the posterior inferences for not just the individual behaviors but also the hierarchical populations from which those behaviors are, at least mathematically, drawn.

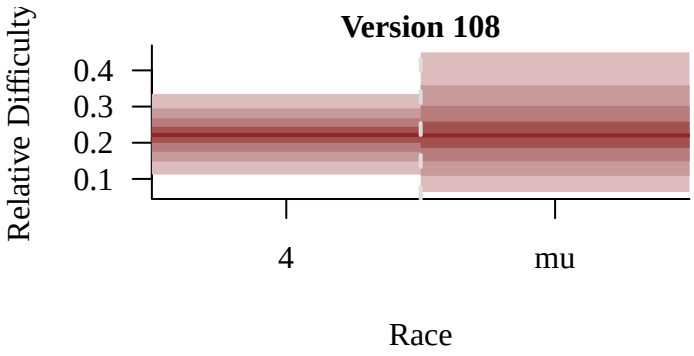
Each MapRando version defines a separate hierarchical population and, unsurprisingly, the inferred population behavior is most precise for the later versions that have been played the most.

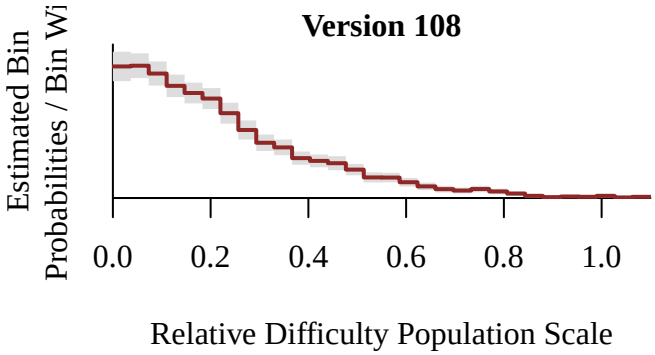
```
for (v in 1:data$N_versions) {
  par(mfrow=c(2, 1), mar=c(5, 5, 1, 1))

  races <- which(data$version_idxs == v)
  names <- sapply(races, function(r) paste0('rel_difficulties[', r,']'))
  names <- c(names, paste0('mu_rel_difficulties[', v, ']'))

  version_name <- paste("Version", uniq_versions[v])

  util$plot_disc_pushforward_quantiles(samples4, names,
                                       xlab="Race",
                                       xticklabs=c(races, 'mu'),
```

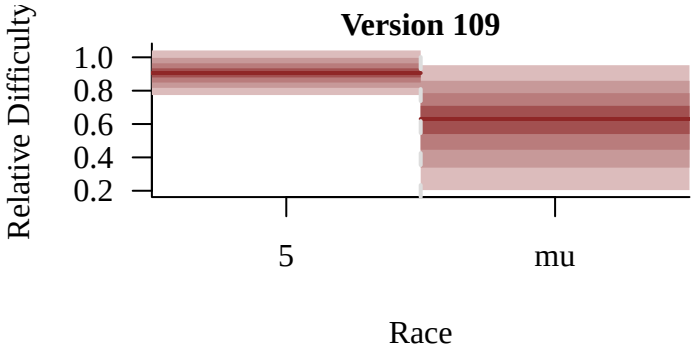
```
                                        ylab="Relative Difficulty",
                                        main=version_name)
  abline(v=length(races) + 0.5, lwd=2, lty=2, col="#DDDDDD")

  name <- paste0('tau_rel_difficulties[', v, ']')
  display_name <- "Relative Difficulty Population Scale"
  util$plot_expectand_pushforward(samples4[[name]], 30, flim=c(0, 1.1),
                                  display_name=display_name,
                                  main=version_name)
}
```

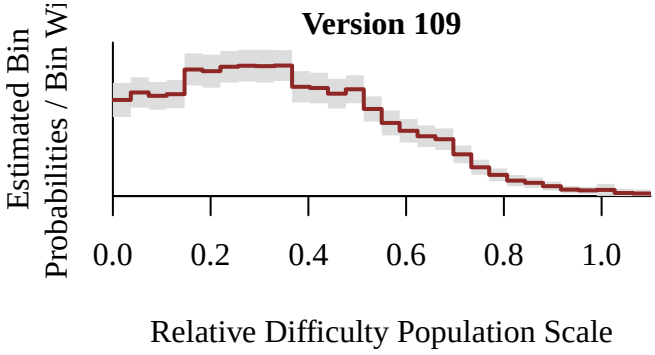


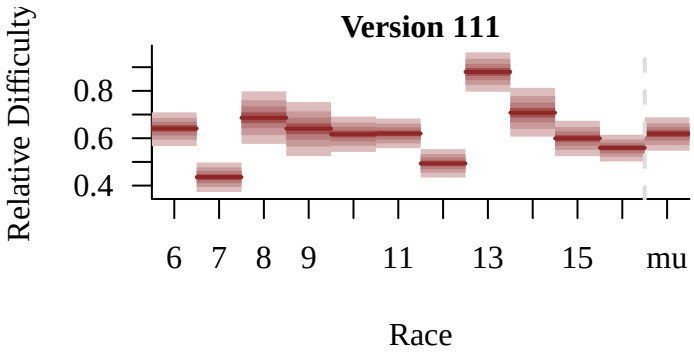


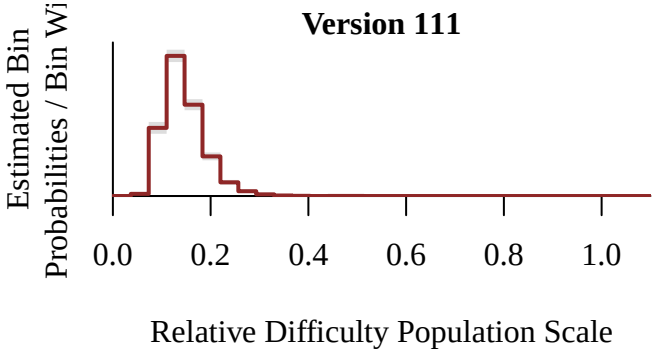

Version 108
Relative Difficulty
0.4
0.3
0.2
0.1
4
mu
Race

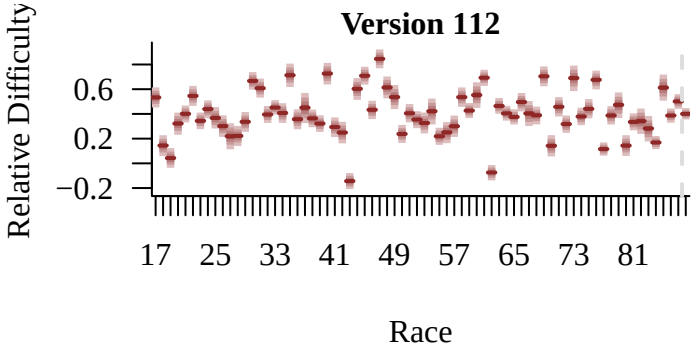

Version 108
Estimated Bin Probabilities / Bin Wi
0.0
0.2
0.4
0.6
0.8
1.0
Relative Difficulty Population Scale

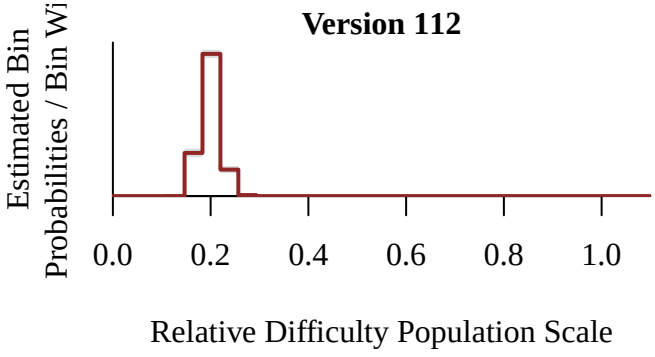

Version 109
Relative Difficulty
1.0
0.8
0.6
0.4
0.2
5
mu
Race

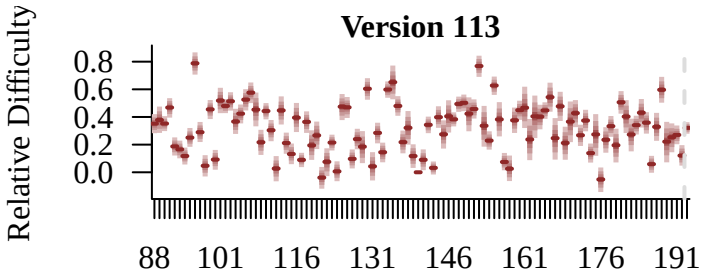

Version 109
Estimated Bin Probabilities / Bin Wi
0.0
0.2
0.4
0.6
0.8
1.0
Relative Difficulty Population Scale

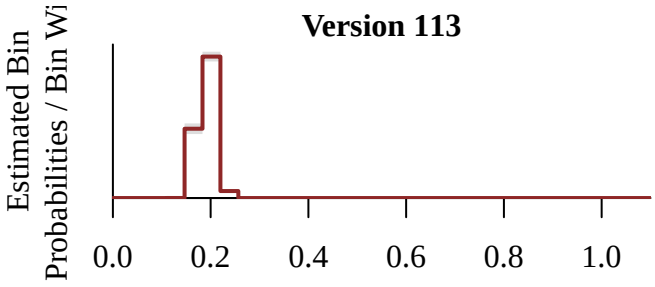

Version 111
Relative Difficulty
0.8
0.6
0.4
6
7
8
9
11
13
15
mu
Race


Version 111
Estimated Bin Probabilities / Bin Wi
0.0
0.2
0.4
0.6
0.8
1.0
Relative Difficulty Population Scale


Version 112
Relative Difficulty
0.6
0.2
−0.2
17
25
33
41
49
57
65
73
81
Race


Version 112
Estimated Bin Probabilities / Bin Wi
0.0
0.2
0.4
0.6
0.8
1.0
Relative Difficulty Population Scale





Subject to the posterior uncertainties all of the version population behaviors are consistent with each other. For example both versions 112 and 113 strongly suppress relative seed difficulty magnitudes above

$$2\tau_{\text{difficulty}} \approx 0.4,$$

implying range of proportional changes to the baseline finish time between

$$\exp(-0.4) \approx 0.67$$

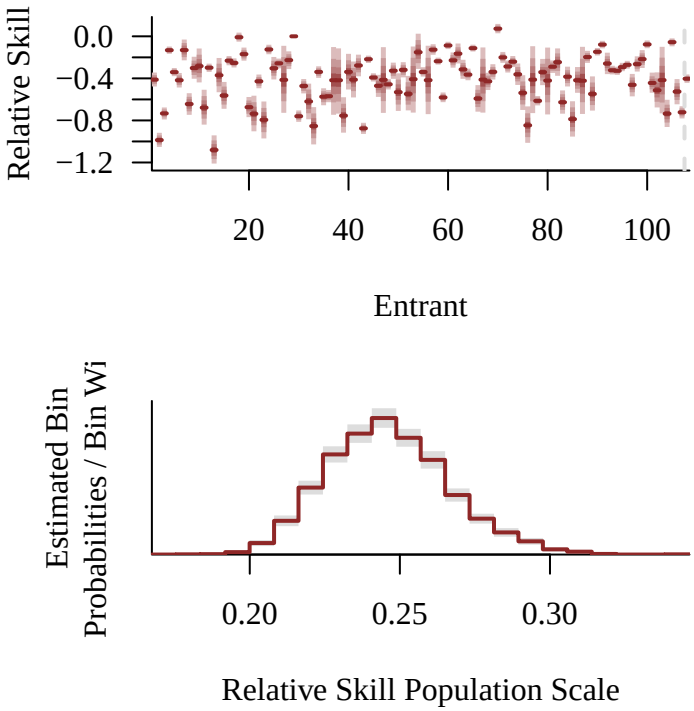
and

$$\exp(+0.4) \approx 1.49.$$

Interestingly the entrant skills exhibit similar regularization, with the population scale concentrating just under 0.3.

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

names <- sapply(1:data$N_entrants,
                function(n) paste0('rel_skills[', n,']'))
names <- c(names, 'mu_rel_skills')
util$plot_disc_pushforward_quantiles(samples4, names,
                                     xlab="Entrant",
                                     ylab="Relative Skill")
```

```
abline(v=data$N_entrants + 0.5, lwd=2, lty=2, col="#DDDDDD")

display_name <- "Relative Skill Population Scale"
util$plot_expectand_pushforward(samples4[['tau_rel_skills']], 20,
                                display_name=display_name)
```



4.5 Inferential Comparison

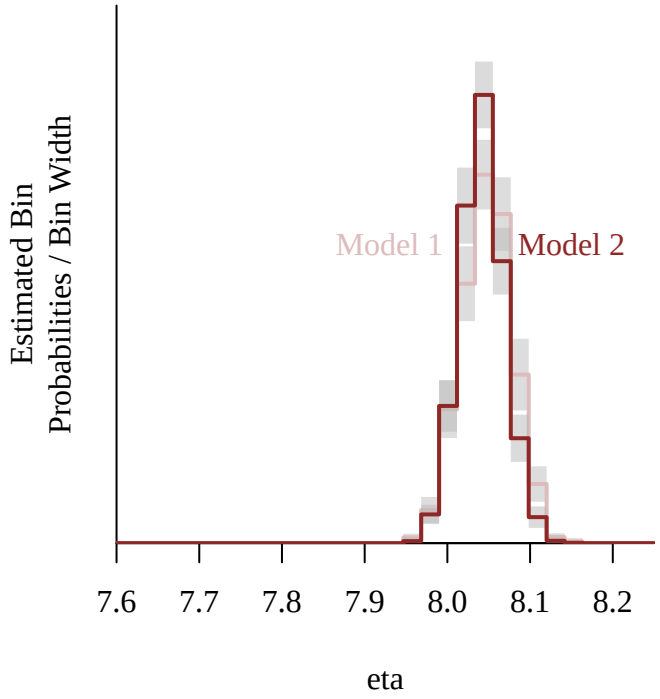
Before applying our posterior inferences to make useful statements about the entrants and their behavior in future races let's pause and examine the impact our model development has had on our posterior inferences.

4.5.1 Log Baseline

To start let's look at the parameter η which, once exponentiated, sets the baseline finish time.

Interestingly changing the observational model doesn't seem to have strongly impacted the inferences for η, at least within the resolution of our Markov chain Monte Carlo estimators.

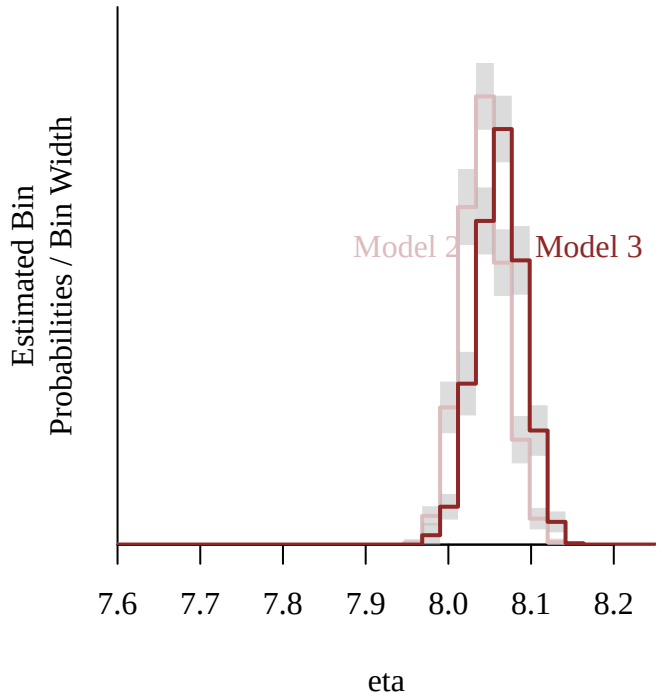
```
par(mfrow=c(1, 1), mar=c(5, 5, 1, 1))

util$plot_expectand_pushforward(samples1[['eta']], 30,
                                flim=c(7.6, 8.25),
                                ylim=c(0, 18),
                                display_name="eta",
                                col=util$c_light)
text(7.93, 10, "Model 1", col=util$c_light)

util$plot_expectand_pushforward(samples2[['eta']], 30,
                                flim=c(7.6, 8.25),
                                border="#BBBBBB88", add=TRUE)
text(8.15, 10, "Model 2", col=util$c_dark)
```



On the other hand incorporating forfeits results in a slight shift of the entire marginal posterior distribution towards larger values. This direction makes sense because without accounting for forfeits the observed finish times are biased towards more optimistic outcomes.

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

util$plot_expectand_pushforward(samples2[['eta']], 30,
                                flim=c(7.6, 8.25),
```

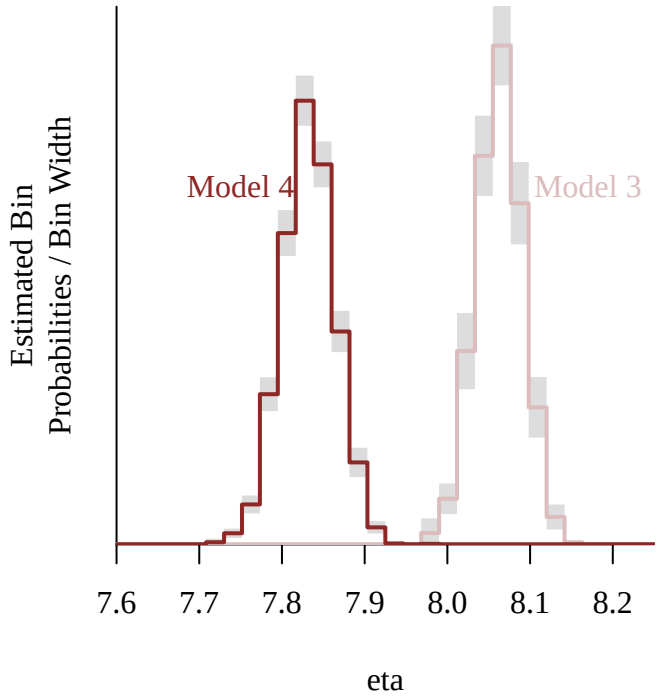
```
                                  ylim=c(0, 18),
                                  display_name="eta",
                                  col=util$c_light)
text(7.95, 10, "Model 2", col=util$c_light)

util$plot_expectand_pushforward(samples3[['eta']], 30,
                                  flim=c(7.6, 8.25),
                                  border="#BBBBBB88", add=TRUE)
text(8.17, 10, "Model 3", col=util$c_dark)
```



The introduction of the seed difficulty and entrant skill hierarchies substantially shifts the posterior inferences for η down towards smaller values.

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

util$plot_expectand_pushforward(samples3[['eta']], 30,
                                  flim=c(7.6, 8.25),
                                  ylim=c(0, 15),
                                  display_name="eta",
                                  col=util$c_light)
text(8.17, 10, "Model 3", col=util$c_light)
```

```
util$plot_expectand_pushforward(samples4[['eta']], 30,
                                flim=c(7.6, 8.25),
                                border="#BBBBBB88", add=TRUE)
text(7.75, 10, "Model 4", col=util$c_dark)
```



4.5.2 Entrant 65 Relative Skill

Now let's dig into posterior inferences for some entrants with particularly extreme observed behaviors that will hopefully emphasize the impact of our model improvements.

For example the record of entrant 65 features lots of entrances and only a single forfeit. Consequently we might naively expect the introduction of forfeits and the entrant skill hierarchy to have less impact on inferences for the skill parameter of entrant 65 relative to the anchor entrant 29.

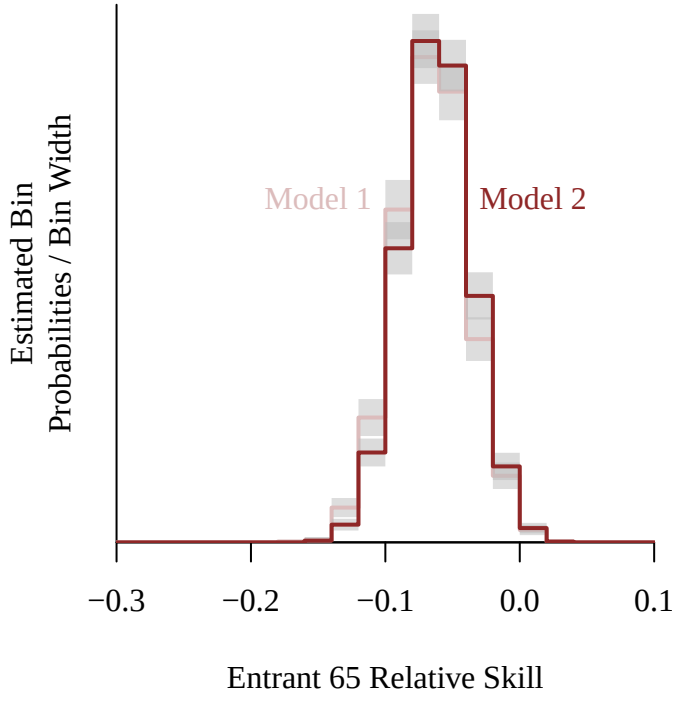
```
e <- 65
summarize_entrant(e)
```

```
Entrant 65
  64 total entrances
  63 finishes (98.4%)
  1 forfeit (1.6%)
```

```
name <- paste0('rel_skills[', e,']')
xname <- paste0('Entrant ', e, ' Relative Skill')
```

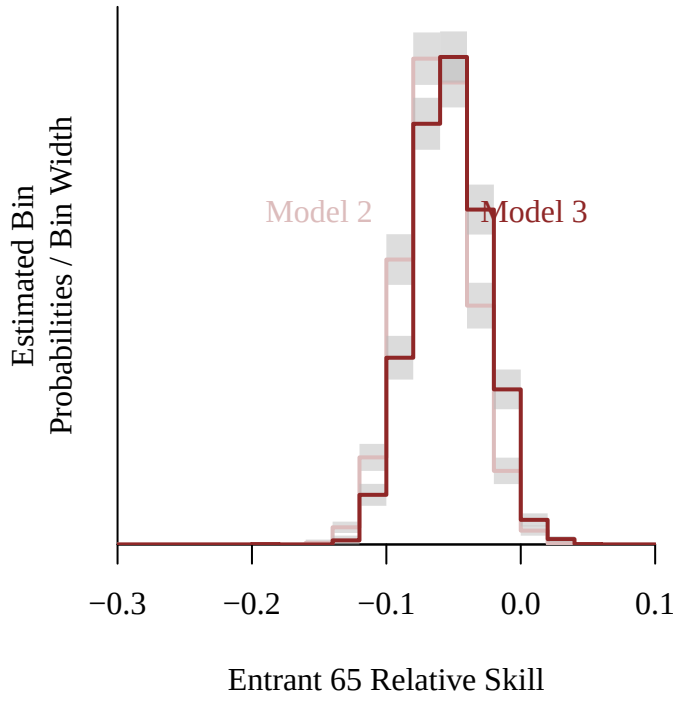
If anything transitioning from a gamma to inverse gamma observational model yields a very slight shift of the marginal skill posterior distribution to larger values. On the other hand because this shift is largely enveloped by the Markov chain Monte Carlo errors it could also just be a computational artifact.

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

util$plot_expectand_pushforward(samples1[[name]], 20,
                                flim=c(-0.3, 0.1),
                                display_name=xname,
                                col=util$c_light)
text(-0.15, 10, "Model 1", col=util$c_light)

util$plot_expectand_pushforward(samples2[[name]], 20,
                                flim=c(-0.3, 0.1),
                                border="#BBBBBB88", add=TRUE)
text(0.01, 10, "Model 2", col=util$c_dark)
```



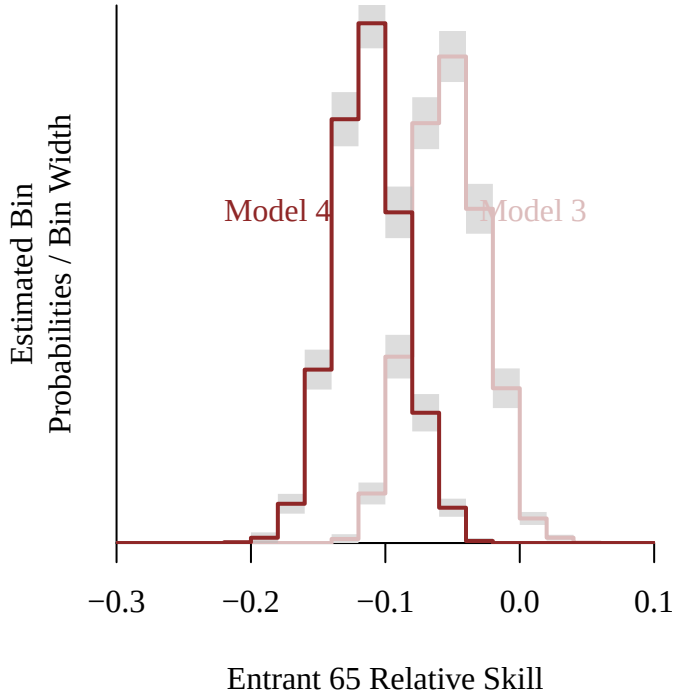
The introduction of forfeits into the model has a slightly stronger impact on the marginal skill posterior distribution, shifting it up to further larger values. Even though entrant 65 rarely forfeited the ignorance of forfeits can allow data from other entrants to bias inferences for common parameters like η, which then biases inferences for all entrant skills.

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

util$plot_expectand_pushforward(samples2[[name]], 20,
                                flim=c(-0.3, 0.1),
                                display_name=xname,
                                col=util$c_light)
text(-0.15, 10, "Model 2", col=util$c_light)

util$plot_expectand_pushforward(samples3[[name]], 20,
                                flim=c(-0.3, 0.1),
                                border="#BBBBBB88", add=TRUE)
text(0.01, 10, "Model 3", col=util$c_dark)
```



The introduction of the skill hierarchy has an even stronger effect, only this time in the opposite direction with the posterior inferences regularized towards more negative values.

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

util$plot_expectand_pushforward(samples3[[name]], 20,
                                flim=c(-0.3, 0.1),
                                display_name=xname,
                                col=util$c_light)
text(0.01, 10, "Model 3", col=util$c_light)

util$plot_expectand_pushforward(samples4[[name]], 20,
                                flim=c(-0.3, 0.1),
                                border="#BBBBBB88", add=TRUE)
text(-0.18, 10, "Model 4", col=util$c_dark)
```



4.5.3 Entrant 44 Relative Skill

Let's contrast these changes with those for the relative skill parameter of entrant 44, who also entered into many races but forfeited at a much higher rate than entrant 65.

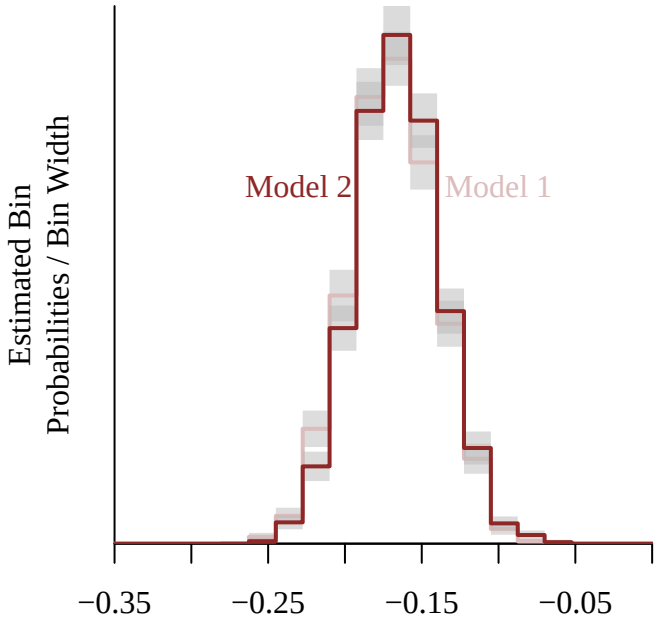
```
e <- 44
summarize_entrant(e)
```

```
Entrant 44
  71 total entrances
  56 finishes (78.9%)
  15 forfeits (21.1%)
```

```
name <- paste0('rel_skills[', e,']')
xname <- paste0('Entrant ', e, ' Relative Skill')
```

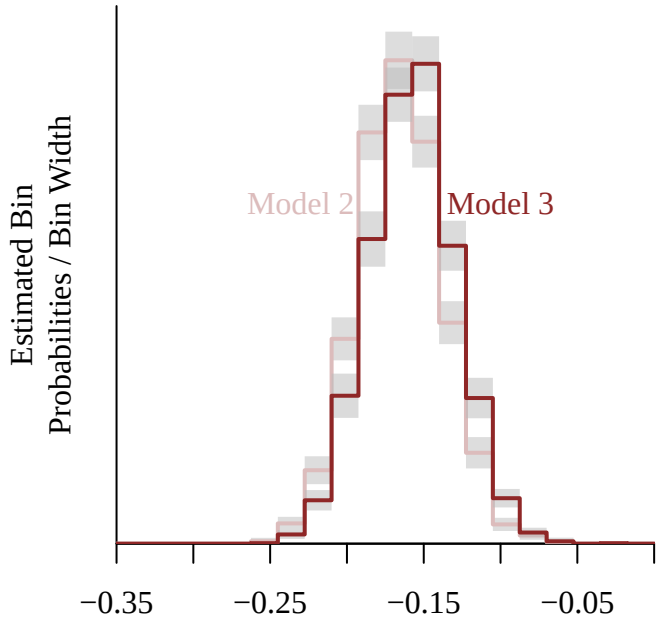
Again the tweak of the observational model has a negligible impact on the marginal posterior inferences.

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

util$plot_expectand_pushforward(samples1[[name]], 20,
                                flim=c(-0.35, 0),
                                display_name=xname,
                                col=util$c_light)
text(-0.1, 10, "Model 1", col=util$c_light)

util$plot_expectand_pushforward(samples2[[name]], 20,
                                flim=c(-0.35, 0),
                                border="#BBBBBB88", add=TRUE)
text(-0.23, 10, "Model 2", col=util$c_dark)
```



Somewhat surprisingly incorporating forfeits shifts the entrant 44 relative skill parameter to larger values!

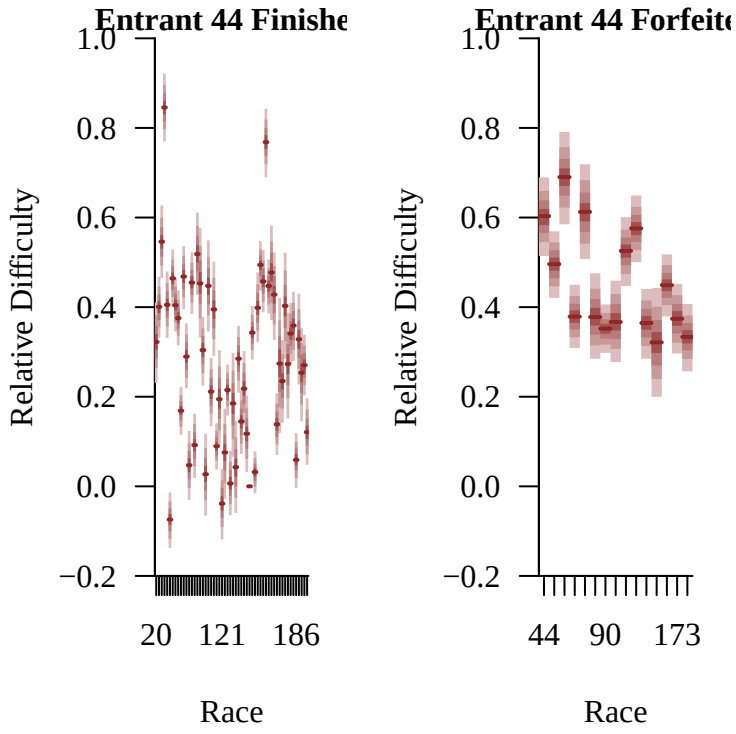
```
par(mfrow=c(1, 1), mar=c(5, 5, 1, 1))

util$plot_expectand_pushforward(samples2[[name]], 20,
                                flim=c(-0.35, 0),
                                display_name=xname,
                                col=util$c_light)
text(-0.23, 10, "Model 2", col=util$c_light)

util$plot_expectand_pushforward(samples3[[name]], 20,
                                flim=c(-0.35, 0),
                                border="#BBBBBB88", add=TRUE)
text(-0.1, 10, "Model 3", col=util$c_dark)
```



This suggests that entrant 44 forfeited for only particularly difficult races. Indeed examining the seed difficulty inferences it appears that entrant 44 has largely forfeited only when confronted with more difficult seeds.

```
f_races <- c()
for (r in 1:data$N_races) {
  idxs <- data$race_f_start_idxs[r]:data$race_f_end_idxs[r]
  if (e %in% data$race_entrant_f_idxs[idxs])
    f_races <- c(f_races, r)
}

dnf_races <- c()
for (r in 1:data$N_races) {
  if (data$race_N_entrants_dnf[r] == 0) next
  idxs <- data$race_dnf_start_idxs[r]:data$race_dnf_end_idxs[r]
  if (e %in% data$race_entrant_dnf_idxs[idxs])
    dnf_races <- c(dnf_races, r)
}

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

names <- sapply(f_races,
                function(r) paste0('rel_difficulties[', r,']'))
```

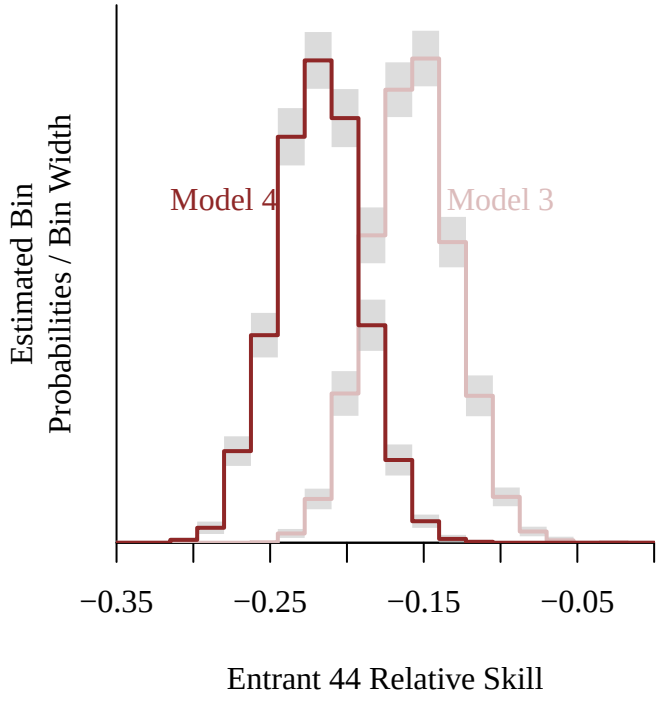
```
util$plot_disc_pushforward_quantiles(samples4, names,
                                     xlab="Race",
                                     xticklabs=f_races,
                                     ylab="Relative Difficulty",
                                     display_ylim=c(-0.2, 1.0),
                                     main="Entrant 44 Finished")

names <- sapply(dnf_races,
                function(r) paste0('rel_difficulties[', r,']'))
util$plot_disc_pushforward_quantiles(samples4, names,
                                     xlab="Race",
                                     xticklabs=dnf_races,
                                     ylab="Relative Difficulty",
                                     display_ylim=c(-0.2, 1.0),
                                     main="Entrant 44 Forfeited")
```



With the introduction of a skill hierarchy the posterior inferences for the relative skill parameter of entrant 44 are regularized towards more negative values.

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

```
util$plot_expectand_pushforward(samples3[[name]], 20,
                                flim=c(-0.35, 0),
                                display_name=xname,
                                col=util$c_light)
text(-0.1, 10, "Model 3", col=util$c_light)

util$plot_expectand_pushforward(samples4[[name]], 20,
                                flim=c(-0.35, 0),
                                border="#BBBBBB88", add=TRUE)
text(-0.28, 10, "Model 4", col=util$c_dark)
```



4.5.4 Entrant 83 Relative Skill

Lastly let's take a look at an entrant with only a few race entrances. In particular entrant 83 has only five entrances and almost half of them are forfeits.

```
e <- 83
summarize_entrant(e)
```

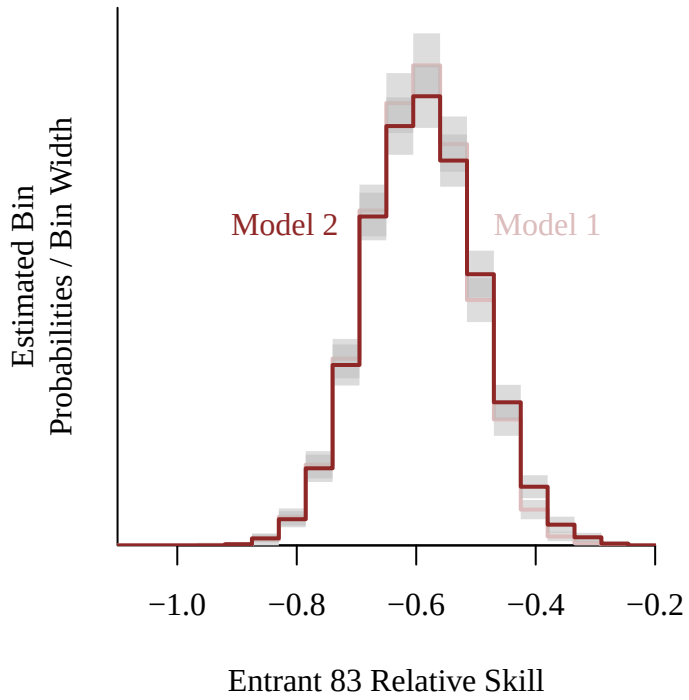
```
Entrant 83
```

```
  5 total entrances
  3 finishes (60.0%)
  2 forfeits (40.0%)
```

```
name <- paste0('rel_skills[', e,']')
xname <- paste0('Entrant ', e, ' Relative Skill')
```

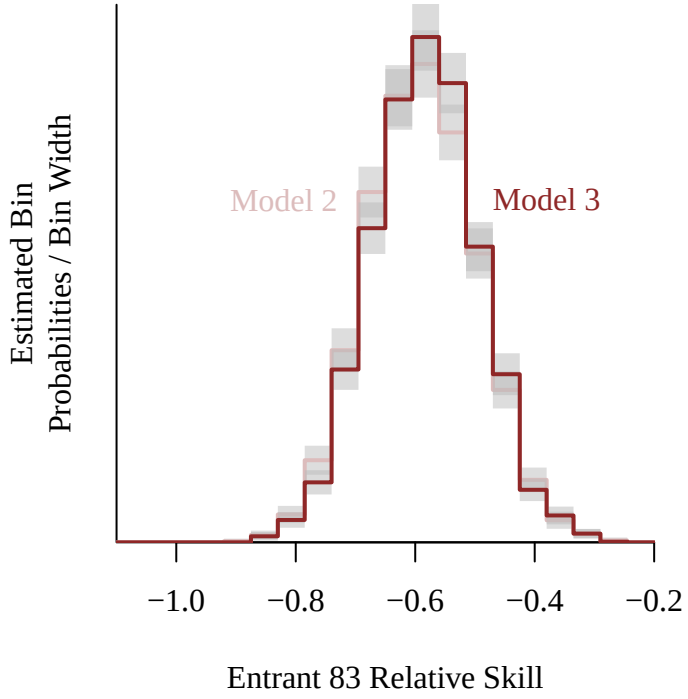
Once again the transition from gamma to inverse gamma observational models has little impact on the marginal skill inferences.

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

util$plot_expectand_pushforward(samples1[[name]], 20,
                                flim=c(-1.1, -0.2),
                                display_name=xname,
                                col=util$c_light)
text(-0.38, 3, "Model 1", col=util$c_light)

util$plot_expectand_pushforward(samples2[[name]], 20,
                                flim=c(-1.1, -0.2),
                                border="#BBBBBB88", add=TRUE)
text(-0.82, 3, "Model 2", col=util$c_dark)
```



Incorporating forfeits perhaps slightly pushes the skill marginal posterior distribution to larger values, but the differences are largely consistent with the Markov chain Monte Carlo estimator error.

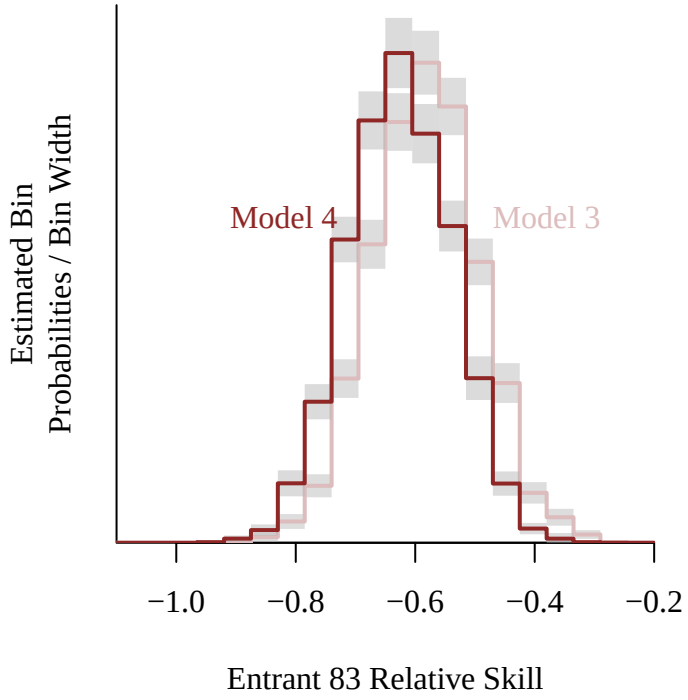
```
par(mfrow=c(1, 1), mar=c(5, 5, 1, 1))

util$plot_expectand_pushforward(samples2[[name]], 20,
                                flim=c(-1.1, -0.2),
                                display_name=xname,
                                col=util$c_light)
text(-0.82, 3, "Model 2", col=util$c_light)

util$plot_expectand_pushforward(samples3[[name]], 20,
                                flim=c(-1.1, -0.2),
                                border="#BBBBBB88", add=TRUE)
text(-0.38, 3, "Model 3", col=util$c_dark)
```



The hierarchical coupling weakly shifts the posterior inferences towards more negative skill values.

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

```
util$plot_expectand_pushforward(samples3[[name]], 20,
                                flim=c(-1.1, -0.2),
                                display_name=xname,
                                col=util$c_light)
text(-0.38, 3, "Model 3", col=util$c_light)

util$plot_expectand_pushforward(samples4[[name]], 20,
                                flim=c(-1.1, -0.2),
                                border="#BBBBBB88", add=TRUE)
text(-0.82, 3, "Model 4", col=util$c_dark)
```



4.6 Possible Model Expansions

At this point our model appears to adequately capture the observed summary statistics behaviors. That said because we spot checked the retrodictive behavior for only a few individual races and entrants there is plenty of room for subtle model inadequacies to hide. Moreover the available domain expertise suggests plenty of possible model improvements that we could investigate more carefully if we had the time, need, or both.

Attempting to implement any of these model expansions would be a useful exercise for any enterprising readers.

4.6.1 Idiosyncratic Entrants

Given that we have already argued that our domain expertise about the entrant behaviors is exchangeable there is nothing preventing us from hierarchically modeling the variation in not only entrant skill but also entrant forfeit behaviors.

If entrant skill, forfeit threshold, and forfeit scale all varied independently then implementing this model would be mostly straightforward, with some possible challenges in accommodating the positivity constraint on the forfeit scale. There's no immediate reason, however, why the heterogeneity in these parameters couldn't be coupled together. For example entrants with higher skills might also tend to have higher forfeit thresholds and vice versa. In this case we would need to consider a multivariate hierarchical population model.

This is the inevitable challenge with hierarchical modeling in practice. Once we identify which behaviors are heterogeneous and exchangeable we still need to determine *how* those behaviors could vary.

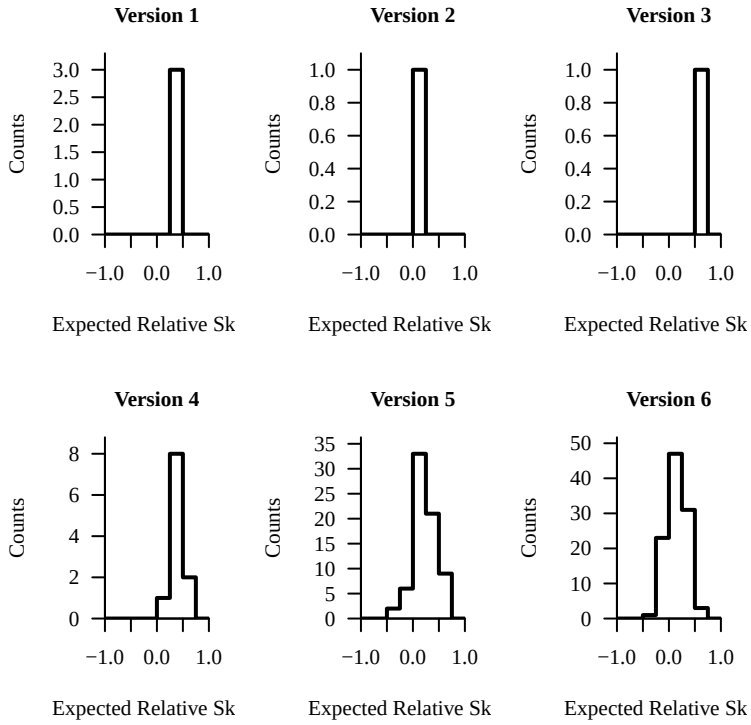
4.6.2 Non-Normal Population Models

Speaking of hierarchies, we are in a somewhat privileged position with the large number of entrants and seeds in our data set. This abundance of contexts might allow us to resolve subtle hierarchical population behaviors that would require non-normal population models. For example we might need to consider population models that accommodate sparsity, asymmetry, and more.

One way to motivate possible population model behaviors is to examine the distribution of expected relative difficulties and expected relative skills from our last non-hierarchical model. For example the expected relative difficulties within each version appear to be consistent with a normal population model.

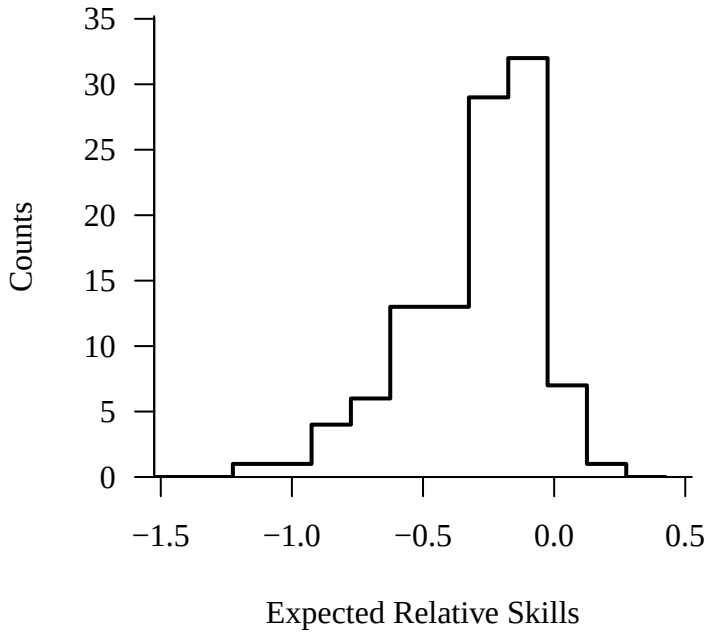
```
par(mfrow=c(2, 3), mar=c(5, 5, 3, 1))

expected_difficulty <- function(r){
  name <- paste0('rel_difficulties[', r, ']')
  util$ensemble_mcmc_est(samples3[[name]])[1]
}

for (v in 1:data$N_versions) {
  mean_rel_difficulties <- sapply(which(data$version_idxs == v),
                                  expected_difficulty)
  util$plot_line_hist(mean_rel_difficulties, -1, 1, 0.25,
                      xlab="Expected Relative Skills",
```

```
                    main=paste("Version", v))
}
```



On the other hand the expected relative skills appear to be much more asymmetric than a normal population model can accommodate.

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

expected_skill <- function(e) {
  util$ensemble_mcmc_est(samples3[[paste0('rel_skills[', e, ']')]])[1]
}

mean_rel_skills <- sapply(1:data$N_entrants, expected_skill)
util$plot_line_hist(mean_rel_skills, -1.5, 0.5, 0.15,
                    xlab="Expected Relative Skills")
```



In particular our use of a normal population model here might be over-regularizing posterior inferences for the more skilled entrants. An asymmetric population model for the relative skills, for instance a skew normal population model, might yield more accurate inferences and predictions.

4.6.3 Self-Improvement

When exploring the time-dependence of the seed difficulties we briefly considered time-dependent entrant skills before accepting the evolving MapRando version as the most likely explanation. That said there's no reason why we couldn't expand our model to allow for time-dependent entrant skills, if only to see if we could resolve any substantial dependencies with the data we have collected.

The main challenge with implementing time-dependent skills is determining how exactly to model how entrants improve and hence what kinds of time-dependencies we should prioritize. For example if improvement scales with the number of MapRando games played, and entrant interest in the game is not uniform in time, then modeling skill as a function of race date-time might not be the most best path. Instead it might be more productive to allow entrant skills to depend on cumulative participation if not something else entirely.

We still then have to determine the possible functional relationships between skill and the appropriate evolution metric. We could, for instance, simply assume a linear relationship for simplicity or consider more sophisticated relationships that allow for behaviors like saturation.

4.6.4 Variable Variability

Throughout our model development have assumed a common ψ across all races, even as the randomization seeds change. That said sometimes the MapRando randomization logic results in particularly ambiguous progression paths; entrants taking the right path first will tend to finish especially fast while those who explore the wrong paths will tend to finish later, resulting in especially large variability regardless of entrant skill. On the other hand some seeds result in progression paths that are easier to predict, narrowing the range of possible finish times.

One way to account for this heterogeneity is to allow ψ to vary across race seeds. We could even model the ψ parameters hierarchically to help regularize inferences for races with only a few entrants.

Before expanding the model, however, we would first want to see if we could identify any consequences of this behavior in new posterior retrodictive checks. For example we might look at the finish time histograms for more races to see if the observed behavior is wider or narrow than the posterior predictive behavior. We could also try to engineer summary statistics that are directly sensitive to the variability, such as the ratio of the empirical variance to the squared empirical mean within each race or statistics that are sensitive to heterogeneity in those individual race statistics.

At the same time entrants who are more experienced with MapRando games, especially the underlying logic of the map randomization, can often identify the correct progression paths quickly and avoid wasting time exploring dead ends. This suggests that ψ could also vary across entrants. The study of this heterogeneity would proceed similarly to the above study of seed heterogeneity, only separating the summary statistics by individual entrants instead of individual races.

5 Actionable Insights

Although it's easy to become distracted by all of the directions we can take our final model we shouldn't dismiss all of the powerful things that we can already do with it. In this section we'll apply our posterior inferences to a few applications that might arise in actual practice.

5.1 Ranking Entrants

A common objective of races is to construct leader boards where entrants are ranked in order of their performance. For example `https://racetime.gg/smr` uses a heuristic, iterative system to assign points to entrants based on their placement in each race and then uses those points to determine a dynamic leader board. The top nine entrants as of August 3rd, 2024 are shown in Table 1.

Table 1: The website `https://racetime.gg/smr` ranks entrants based on points earned during each race.

Rank	1	2	3	4	5	6	7	8	9
Entrant Index	70	105	29	100	18	91	60	65	4

As discussed in Section 5 of the pairwise comparison modeling chapter we can rank the entrants by their inferred skills.

For instance we can always rank the entrants by the posterior expectation values of their individual relative skills.

```
expected_skill <- function(e) {
  util$ensemble_mcmc_est(samples4[[paste0('rel_skills[', e, ']')]])[1]
}

expected_skills <- sapply(1:data$N_entrants,
                          function(e) expected_skill(e))

ranked_entrants <- sort(expected_skills, index.return=TRUE)$ix

for (r in 1:9) {
  cat(sprintf("Rank %i: Entrant %i\n",
              r, ranked_entrants[data$N_entrants + 1 - r]))
  if (r == 9) cat("...")
}
```

```
Rank 1: Entrant 70
Rank 2: Entrant 29
Rank 3: Entrant 18
Rank 4: Entrant 105
Rank 5: Entrant 100
Rank 6: Entrant 91
Rank 7: Entrant 60
Rank 8: Entrant 65
Rank 9: Entrant 24
...
```

This heuristic ranking is similar to the official `https://racetime.gg/smr` leader boards, but not quite the same. In both cases the first eight ranks are held by the same eight entrants, although the position of the ranks is slightly permuted.

```
rank <- 1:9
post_mean_ranking <- rev(tail(ranked_entrants, 9))
racetime_ranking <- c(70, 105, 29, 100, 18, 91, 60, 65, 4)

df <- data.frame(rank, post_mean_ranking, racetime_ranking)
names(df) <- c("Rank", "Posterior Mean Ranking", "racetime.gg Ranking")

print(df, row.names=FALSE)
```

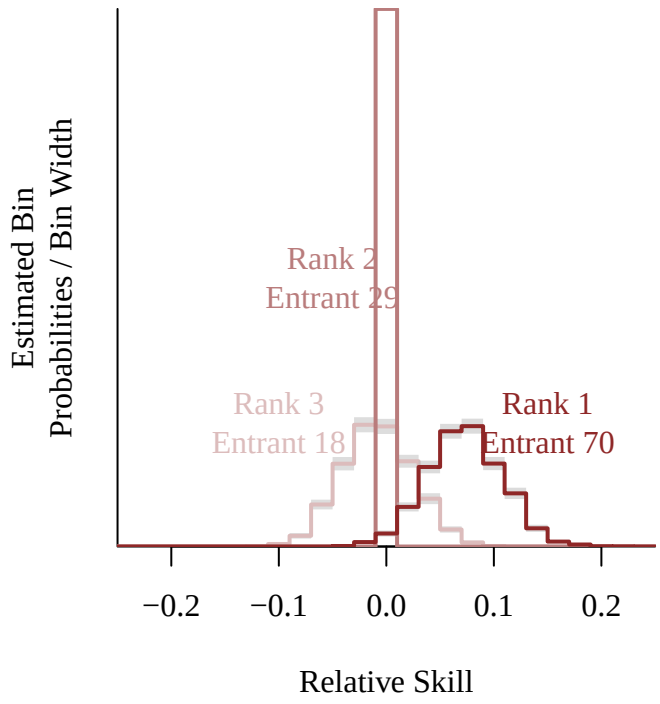
```
 Rank Posterior Mean Ranking racetime.gg Ranking
    1                     70                   70
    2                     29                  105
    3                     18                   29
    4                    105                  100
    5                    100                   18
    6                     91                   91
    7                     60                   60
    8                     65                   65
    9                     24                    4
```

Inconsistent rankings are not at all surprising given the subtle differences in their construction; the expected rankings are based on race finish times while the `https://racetime.gg/smr` rankings are based on race placement. Moreover the uncertainty in our inferences allow for multiple rankings to be consistent with the observed data.

For example even though entrant 70 exhibits a higher expected skill than entrant 29 our inferential uncertainties are not inconsistent with the exact skill of entrant 29 actually surpassing the exact skill of entrant 70. Recall that entrant 29 is the anchor so their relative skill is exactly zero.

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

e <- ranked_entrants[data$N_entrants + 1 - 3]
name <- paste0('rel_skills[', e,']')

util$plot_expectand_pushforward(samples4[[name]], 25,
                                flim=c(-0.25, 0.25),
                                ylim=c(0, 50),
                                display_name="Relative Skill",
                                col=util$c_light)
text(-0.1, 11.5, paste("Rank 3\nEntrant", e), col=util$c_light)
```

```
e <- ranked_entrants[data$N_entrants + 1 - 2]
name <- paste0('rel_skills[', e,']')

util$plot_expectand_pushforward(samples4[[name]], 25,
                                flim=c(-0.25, 0.25),
                                col=util$c_mid,
                                border="#BBBBBB88",
                                add=TRUE)
text(-0.05, 25, paste("Rank 2\nEntrant", e), col=util$c_mid)

e <- ranked_entrants[data$N_entrants + 1 - 1]
name <- paste0('rel_skills[', e,']')

util$plot_expectand_pushforward(samples4[[name]], 25,
                                flim=c(-0.25, 0.25),
                                col=util$c_dark,
                                border="#BBBBBB88",
                                add=TRUE)
text(0.15, 11.5, paste("Rank 1\nEntrant", e), col=util$c_dark)
```



If we want to compare only two entrants at a time then instead of comparing their expected skills we can compute the posterior probability that one skill surpasses the other. Unlike dif-

ferences in expected skills this latter comparison accounts for any inferential coupling between the two skill parameters.

```
e1 <- ranked_entrants[data$N_entrants + 1 - 1]
e2 <- ranked_entrants[data$N_entrants + 1 - 2]
var_repl <- list('s1' = paste0('rel_skills[', e1,']'),
                 's2' = paste0('rel_skills[', e2,']'))

p_est <-
  util$implicit_subset_prob(samples4,
                            function(s1, s2) s1 > s2,
                            var_repl)

format_string <- paste0("Posterior probability that entrant %i skill ",
                        "> entrant %i skill = %.3f +/- %.3f.")
cat(sprintf(format_string, e1, e2, p_est[1], 2 * p_est[2]))
```

```
Posterior probability that entrant 70 skill > entrant 29 skill = 0.982 +/- 0.005.
```

These relative comparisons can also be used to construct another heuristic ranking. For example we could compute the probability that the skill of each entrant is larger than all other entrants and assign first place based on the highest probability. Then we could compute the probability that the skill of each remaining entrant is larger than all of the other remaining entrants and assign second place based on the highest of these probabilities. Finally we could fill out all of the rankings by iterating this procedure until only one entrant is left to occupy last place.

```
skill_comp <- function(active_skill, other_skills) {
  pairwise_comps <- sapply(other_skills,
                           function(other_skill)
                           active_skill > other_skill)
  as.numeric(Reduce("&", pairwise_comps))
}

best_entrant <- function(entrant_idxs) {
  probs <- c()

  for (e in entrant_idxs) {
    other_entrant_idxs <- entrant_idxs[-which(entrant_idxs == e)]
    var_repl <- list('active_skill' = paste0('rel_skills[', e,']'),
                     'other_skills' = array(sapply(other_entrant_idxs,
                                                   function(oe)
```

```
                                                  paste0('rel_skills[', oe,']'))))

    prob_est <- util$implicit_subset_prob(samples4,
                                          skill_comp,
                                          var_repl)
    probs <- c(probs, prob_est[1])
  }

  entrant_idxs[which(probs == max(probs))]
}
```

```
entrant_idxs <- 1:data$N_entrants
e_first <- best_entrant(entrant_idxs)
cat(sprintf("First Place: Entrant %i", e_first))
```

```
First Place: Entrant 70
```

```
entrant_idxs <- entrant_idxs[-which(entrant_idxs == e_first)]
e_second <- best_entrant(entrant_idxs)
cat(sprintf("Second Place: Entrant %i", e_second))
```

```
Second Place: Entrant 29
```

```
entrant_idxs <- entrant_idxs[-which(entrant_idxs == e_second)]
e_third <- best_entrant(entrant_idxs)
cat(sprintf("Third Place: Entrant %i", e_third))
```

```
Third Place: Entrant 18
```

Interestingly this gives the same top three as the ranking of entrants by their posterior expected skills, which is not generally the case. This suggests that the differences between the expected skill rankings and the `https://racetime.gg/smr` rankings may have more to do with the difference between comparing finish times to finish placements.

The practical limitation of this second ranking approach is that it requires estimating *a lot* of expectation values. Moreover if we really wanted to be careful then we would need to ensure that the Markov chain Monte Carlo error for each probability estimate is smaller than any of the differences between the probability estimates so that the resulting ranks are not corrupted by computational artifacts. In practice this might require running more Markov chains than usual, longer Markov chains than usual, or both.

5.2 Predicting Race Outcomes

Another common application is to make predictions about the outcomes of future races, or even hypothetical races that might never occur. We can use our posterior inferences for the observed seed and entrants to immediately inform predictions about how existing entrants would fare if they were able to play previous seeds again. In order to make predictions about the performance of entirely new entrants or new seeds we will need to take advantage of the hierarchical population models.

Let's rerun our last, hierarchical model only with a new `generated quantities` block where we simulate posterior predictive finish statuses and finish times for all entrants in a hypothetical race using a new seed in the latest MapRando version.

```
fit <- stan(file="stan_programs/model5.stan",
            data=data, seed=8438339,
            warmup=1000, iter=2024, refresh=0)
```

Because the `generated quantities` block of Model 5 will consume pseudo-random number generate state differently than that of Model 4 there is a chance that the realized Markov chains will encounter different pathologies. Consequently we'll need to double check the computational diagnostics. Fortunately no new warnings have arisen.

```
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 <- util$extract_expectand_vals(fit)
base_samples <- util$filter_expectands(samples,
                                       c('eta',
                                         'rel_difficulties_free_tilde',
                                         'mu_rel_difficulties',
                                         'tau_rel_difficulties',
                                         'rel_skills_free',
                                         'mu_rel_skills',
                                         'tau_rel_skills',
                                         'psi',
                                         'kappas', 'betas'),
                                       check_arrays=TRUE)
util$check_all_expectand_diagnostics(base_samples)
```

```
kappas[44]:
  Chain 1: Right tail hat{xi} (0.265) exceeds 0.25.
  Chain 2: Right tail hat{xi} (0.289) exceeds 0.25.
  Chain 4: Right tail hat{xi} (0.324) exceeds 0.25.

kappas[48]:
  Chain 4: Left tail hat{xi} (0.314) exceeds 0.25.

kappas[94]:
  Chain 1: Right tail hat{xi} (0.310) exceeds 0.25.

kappas[98]:
  Chain 1: Both left and right tail hat{xi}s (0.303, 0.319) exceed 0.25.
  Chain 2: Left tail hat{xi} (0.296) exceeds 0.25.
  Chain 3: Left tail hat{xi} (0.371) exceeds 0.25.
  Chain 4: Left tail hat{xi} (0.465) exceeds 0.25.


Large tail hat{xi}s suggest that the expectand might not be
sufficiently integrable.
```

The uses of these predictions are endless.

5.2.1 Single Entrant Predictions

For example some entrants not only live-stream their race entrances to their communities but also allow viewers to make non-monetary, over/under bets on their finish times. If we wanted to take these casual activities a bit too far then we could use our predictions to set a betting line where both outcomes are equally probable.

Without forfeits the balanced betting line t_{gamble} would be implicitly defined by the posterior predictive probability

$$\pi(\, [0, t_{\text{gamble}}) \,) = 0.5.$$

In other words t_{gamble} would be given by the median of the posterior predictive finish time distribution for the hosting entrant, which we can estimate with Markov chain Monte Carlo.

To be accurate, however, we need to account for the fact that the hosting entrant might forfeit. Consequently the relevant condition is actually

$$\begin{aligned} \pi(\, [0, t_{\text{gamble}}) \,, \text{forfeit} = 0) &= 0.5 \\ \pi(\, [0, t_{\text{gamble}}) \;\mid \text{forfeit} = 0) \, \pi(\text{forfeit} = 0) &= 0.5 \\ \pi(\, [0, t_{\text{gamble}}) \;\mid \text{forfeit} = 0) \; (1 - \pi(\text{forfeit} = 1)) &= 0.5, \end{aligned}$$

or

$$\pi(\,[0, t_{\text{gamble}})\ \mid \text{forfeit} = 0) = \frac{0.5}{1 - \pi(\text{forfeit} = 1)}.$$

Fortunately we can readily compute all of these ingredients using our `Stan` output.

To demonstrate let's look at entrant 65. First we can compute the probability of forfeit.

```
e <- 65
name <- paste0('entrant_statuses_pred[', e, ']')
p_dnf <- util$ensemble_mcmc_est(samples[[name]])[1]
cat(sprintf("Probability entrant %i forfeits = %.3f.", e, p_dnf))
```

```
Probability entrant 65 forfeits = 0.017.
```

Because the forfeit probability `p_dnf` is so small the balanced probability allocation needed to define t_{gamble} is very close to $\frac{1}{2}$.

```
p_balanced <- 0.5 / (1 - p_dnf)
```

Averaging the empirical quantiles within each Markov chain provides a consistent estimate of the exact posterior quantile.

```
name <- paste0('entrant_f_times_pred[', e, ']')
t_gamble <- util$ensemble_mcmc_quantile_est(samples[[name]],
                                            c(p_balanced))

cat(sprintf("t_gamble = %.3f minutes", t_gamble / 60))
```

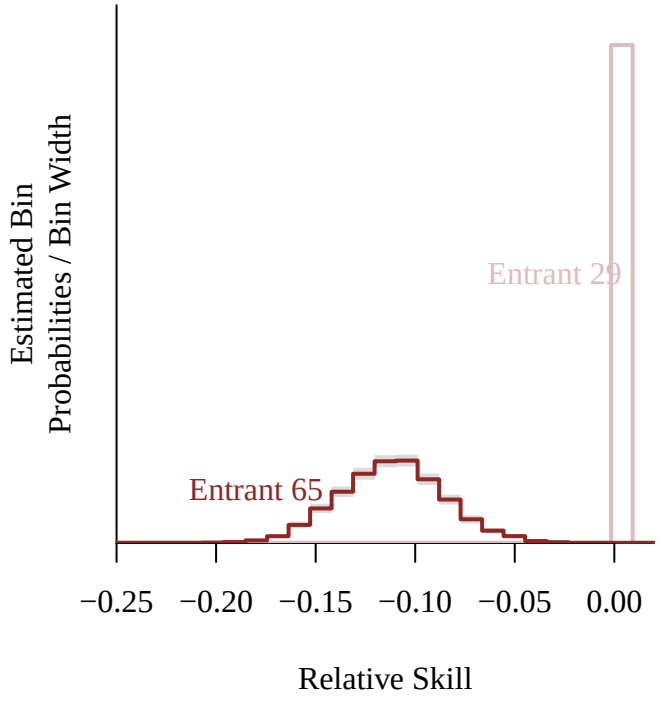
```
t_gamble = 64.146 minutes
```

5.2.2 Head-to-Head Predictions

We can also predict how two entrants will perform relative to each other in our hypothetical race. Here we'll consider entrant 29 racing against entrant 65.

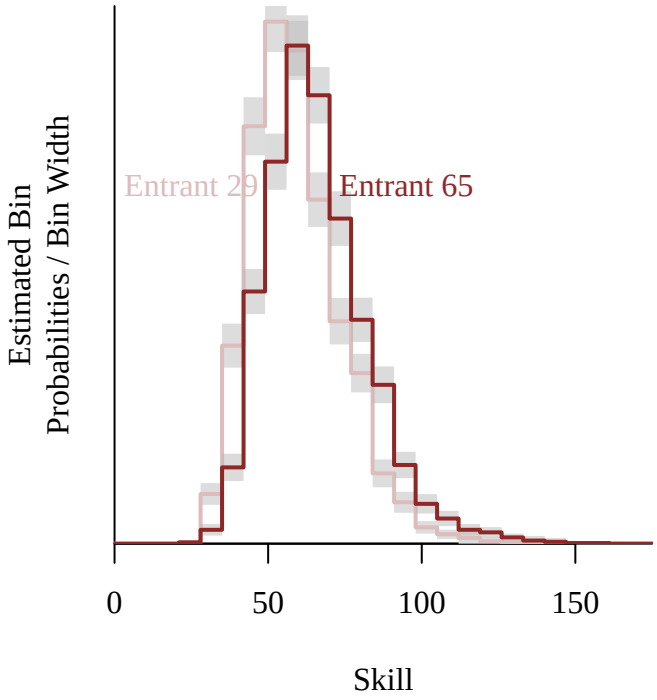
The marginal posterior distribution for the two entrants' relative skill parameters overlap quite a bit, but the relative skill of entrant 29 does favor larger values than that of entrant 65.

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

e <- 29
name <- paste0('rel_skills[', e,']')

util$plot_expectand_pushforward(samples[[name]], 25,
                                flim=c(-0.25, 0.02),
                                ylim=c(0, 100),
                                display_name="Relative Skill",
                                col=util$c_light)
text(0.-0.03, 50, paste("Entrant", e), col=util$c_light)

e <- 65
name <- paste0('rel_skills[', e,']')

util$plot_expectand_pushforward(samples[[name]], 25,
                                flim=c(-0.25, 0.02),
                                col=util$c_dark,
                                border="#BBBBBB88",
                                add=TRUE)
text(-0.18, 10, paste("Entrant", e), col=util$c_dark)
```



What really matters for predictive race outcomes, however, are not the latent skills but rather the predicted finish times. The marginal posterior predictive distributions for the predicted finish times overlap even more, indicating a much closer race than we might expect from the skills alone.

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

e <- 29
name <- paste0('entrant_f_times_pred[', e,']')

util$plot_expectand_pushforward(samples[[name]] / 60, 25,
                                flim=c(0, 175),
                                ylim=c(0, 0.03),
                                display_name="Skill",
                                col=util$c_light)
text(25, 0.02, paste("Entrant", e), col=util$c_light)

e <- 65
name <- paste0('entrant_f_times_pred[', e,']')

util$plot_expectand_pushforward(samples[[name]] / 60, 25,
                                flim=c(0, 175),
                                col=util$c_dark,
                                border="#BBBBBB88",
                                add=TRUE)
text(95, 0.02, paste("Entrant", e), col=util$c_dark)
```



That said the predicted finish times still don't tell the entire story. To accurately predict a winner we also need to take into account the possibility that one, or possibly even both, of the entrants forfeits. Altogether there are five possible outcomes that are relevant to whether or not entrant 29 beats entrant 65:

- Entrant 29 forfeits and entrant 65 forfeits,
- Entrant 29 forfeits and entrant 65 finishes,
- Entrant 29 finishes and entrant 65 forfeits,
- Entrant 29 finishes and entrant finishes and $t_{29} < t_{65}$,
- Entrant 29 finishes and entrant finishes and $t_{29} > t_{65}$.

Of these entrant 29 decisively wins only in the third and fourth outcomes.

In order to evaluate the probability that entrant 29 wins we'll need to make careful use of

conditional probability theory to account for all of these outcomes,

$$
\begin{aligned}
&\pi(\text{Entrant 29 beats entrant 65 }) \\
&\quad = \quad \pi(\text{Entrant 29 beats entrant 65 } \mid \text{forfeit}_{29} = 1, \text{forfeit}_{65} = 1) \\
&\qquad \cdot \pi(\text{forfeit}_{29} = 1, \text{forfeit}_{65} = 1) \\
&\qquad + \ \pi(\text{Entrant 29 beats entrant 65 } \mid \text{forfeit}_{29} = 1, \text{forfeit}_{65} = 0) \\
&\qquad \cdot \pi(\text{forfeit}_{29} = 1, \text{forfeit}_{65} = 0) \\
&\qquad + \ \pi(\text{Entrant 29 beats entrant 65 } \mid \text{forfeit}_{29} = 0, \text{forfeit}_{65} = 1) \\
&\qquad \cdot \pi(\text{forfeit}_{29} = 0, \text{forfeit}_{65} = 1) \\
&\qquad + \ \pi(\text{Entrant 29 beats entrant 65 } \mid \text{forfeit}_{29} = 0, \text{forfeit}_{65} = 0, t_{29} < t_{65}) \\
&\qquad \cdot \pi(\text{forfeit}_{29} = 0, \text{forfeit}_{65} = 0, t_{29} < t_{65}) \\
&\qquad + \ \pi(\text{Entrant 29 beats entrant 65 } \mid \text{forfeit}_{29} = 0, \text{forfeit}_{65} = 0, t_{29} > t_{65}) \\
&\qquad \cdot \pi(\text{forfeit}_{29} = 0, \text{forfeit}_{65} = 0, t_{29} > t_{65}) \\
&\quad = \quad 0 \\
&\qquad \cdot \pi(\text{forfeit}_{29} = 1, \text{forfeit}_{65} = 1) \\
&\qquad + \ 0 \\
&\qquad \cdot \pi(\text{forfeit}_{29} = 1, \text{forfeit}_{65} = 0) \\
&\qquad + \ 1 \\
&\qquad \cdot \pi(\text{forfeit}_{29} = 0, \text{forfeit}_{65} = 1) \\
&\qquad + \ 1 \\
&\qquad \cdot \pi(\text{forfeit}_{29} = 0, \text{forfeit}_{65} = 0, t_{29} < t_{65}) \\
&\qquad + \ 0 \\
&\qquad \cdot \pi(\text{forfeit}_{29} = 0, \text{forfeit}_{65} = 0, t_{29} > t_{65}),
\end{aligned}
$$

or

$$
\begin{aligned}
&\pi(\text{Entrant 29 beats entrant 65 }) \\
&\quad = \quad \pi(\text{forfeit}_{29} = 0, \text{forfeit}_{65} = 1) \\
&\qquad + \pi(\text{forfeit}_{29} = 0, \text{forfeit}_{65} = 0, t_{29} < t_{65}) \\
&\quad = \quad \pi(\text{forfeit}_{29} = 0, \text{forfeit}_{65} = 1) \\
&\qquad + \ \pi(t_{29} < t_{65} \mid \text{forfeit}_{29} = 0, \text{forfeit}_{65} = 0) \\
&\qquad \cdot \pi(\text{forfeit}_{29} = 0, \text{forfeit}_{65} = 0) \\
&\quad = \quad p_{\text{forfeit win}} \\
&\qquad + \ \pi(t_{29} < t_{65} \mid \text{forfeit}_{29} = 0, \text{forfeit}_{65} = 0) \cdot p_{\text{no forfeits}}.
\end{aligned}
$$

At that is left is using Markov chain Monte Carlo to estimate the three posterior predictive probabilities on the right-hand side and then combine them together to give the left-hand side.

```
e1 <- 29
status_name1 <- paste0("entrant_statuses_pred[", e1, "]")
time_name1 <- paste0("entrant_f_times_pred[", e1, "]")

e2 <- 65
status_name2 <- paste0("entrant_statuses_pred[", e2, "]")
time_name2 <- paste0("entrant_f_times_pred[", e2, "]")

p_forfeit_win_est <-
  util$implicit_subset_prob(samples,
                            function(s1, s2) s1 == 0 & s2 == 1,
                            list('s1' = status_name1,
                                 's2' = status_name2))

p_no_forfeits_est <-
  util$implicit_subset_prob(samples,
                            function(s1, s2) s1 == 0 & s2 == 0,
                            list('s1' = status_name1,
                                 's2' = status_name2))

p_neg_time_diff_est <-
  util$implicit_subset_prob(samples,
                            function(t1, t2) t1 < t2,
                            list('t1' = time_name1,
                                 't2' = time_name2))


p <- p_forfeit_win_est[1] + p_neg_time_diff_est[1] * p_no_forfeits_est[1]
cat(sprintf("Probability that entrant %i beats entrant %i = %.3f.",
            e1, e2, p))
```

```
Probability that entrant 29 beats entrant 65 = 0.704.
```

Although entrant 29 is definitely favored the outcome is by no means certain!

6 Conclusion

Although the domain of this analysis might be a bit niche the best practices that it demonstrates are fundamental. By understanding the provenance of the data we can motivate an initial probabilistic model and then iteratively improve it until we can no longer resolve any

model inadequacies. The inferences from the final model not only provide a variety of insights about the source of the data but also allow inform all kinds of predictions that might be of practical relevance.

Being able to wax nostalgic about the glory days of the Super Nintendo Entertainment System® and along the way celebrate open source projects and the communities they inspire is just a pleasant bonus.

Acknowledgements

I thank jd for helpful comments.

A very special thanks to everyone supporting me on Patreon: Adam Fleischhacker, Adriano Yoshino, Alejandro Navarro-Martínez, Alessandro Varacca, Alex D, Alexander Noll, Alexander Rosteck, Andrea Serafino, Andrew Mascioli, Andrew Rouillard, Andrew Vigotsky, Ara Winter, Austin Rochford, Avraham Adler, Ben Matthews, Ben Swallow, Benoit Essiambre, Bertrand Wilden, Bradley Kolb, Brandon Liu, Brendan Galdo, Brynjolfur Gauti Jónsson, Cameron Smith, Canaan Breiss, Cat Shark, CG, Charles Naylor, Chase Dwelle, Chris Jones, Christopher Mehrvarzi, Colin Carroll, Colin McAuliffe, Damien Mannion, dan mackinlay, Dan W Joyce, Dan Waxman, Dan Weitzenfeld, Daniel Edward Marthaler, Daniel Saunders, Darshan Pandit, Darthmaluus , David Galley, David Wurtz, Doug Rivers, Dr. Jobo, Dr. Omri Har Shemesh, Dylan Maher, Ed Cashin, Edgar Merkle, Eric LaMotte, Ero Carrera, Eugene O'Friel, Felipe González, Fergus Chadwick, Finn Lindgren, Florian Wellmann, Geoff Rollins, Håkan Johansson, Hamed Bastan-Hagh, Hauke Burde, Hector Munoz, Henri Wallen, hs, Hugo Botha, Ian, Ian Costley, idontgetoutmuch, Ignacio Vera, Ilaria Prosdocimi, Isaac Vock, Isidor Belic, J, J Michael Burgess, jacob pine, Jair Andrade, James C, James Hodgson, James Wade, Janek Berger, Jason Martin, Jason Pekos, Jason Wong, jd, Jeff Burnett, Jeff Dotson, Jeff Helzner, Jeffrey Erlich, Jessica Graves, Joe Sloan, Joe Wagner, John Flournoy, Jonathan H. Morgan, Jonathon Vallejo, Joran Jongerling, JU, June, Justin Bois, Kádár András, Karim Naguib, Karim Osman, Kejia Shi, Kristian Gårdhus Wichmann, Lars Barquist, lizzie , Logan Sullivan, LOU ODETTE, Luís F, Marcel Lüthi, Marek Kwiatkowski, Mark Donoghoe, Markus P., Márton Vaitkus, Matt Moores, Matthew, Matthew Kay, Matthieu LEROY, Mattia Arsendi, Maurits van der Meer, Michael Colaresi, Michael DeWitt, Michael Dillon, Michael Lerner, Mick Cooney, Mike Lawrence, N Sanders, N.S. , Name, Nathaniel Burbank, Nic Fishman, Nicholas Clark, Nicholas Cowie, Nick S, Octavio Medina, Ole Rogeberg, Oliver Crook, Patrick Kelley, Patrick Boehnke, Pau Pereira Batlle, Peter Johnson, Pieter van den Berg , ptr, Ramiro Barrantes Reynolds, Raúl Peralta Lozada, Ravin Kumar, Rémi , Rex Ha, Riccardo Fusaroli, Richard Nerland, Robert Frost, Robert Goldman, Robert kohn, Robin Taylor, Ryan Grossman, Ryan Kelly, S Hong, Sean Wilson, Sergiy Protsiv, Seth Axen, shira, Simon Duane, Simon Lilburn, sssz, Stan_user, Stephen Lienhard, Stew Watts, Stone Chen, Susan Holmes, Svilup, Tao Ye, Tate Tunstall, Tatsuo Okubo, Teresa Ortiz, Theodore Dasher, Thomas Siegert,

Thomas Vladeck, Tobychev , Tomáš Frýda, Tony Wuersch, Virginia Fisher, Vladimir Markov, Wil Yegelwel, Will Farr, woejozney, yolhaj , yureq , Zach A, Zad Rafi, and Zhengchen Cai.

References

"Super Metroid Map Rando." n.d.
"Super Metroid Randomizer | Racetime.gg." n.d.

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 Sonoma 14.4.1

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] parallel_4.3.2     gridExtra_2.3      scales_1.3.0       yaml_2.3.8
 [9] fastmap_1.1.1      ggplot2_3.4.4      R6_2.5.1           curl_5.2.0
[13] knitr_1.45         tibble_3.2.1       munsell_0.5.0      pillar_1.9.0
[17] rlang_1.1.2        utf8_1.2.4         V8_4.4.1           inline_0.3.19
[21] xfun_0.41          RcppParallel_5.1.7 cli_3.6.2          magrittr_2.0.3
[25] digest_0.6.33      grid_4.3.2         lifecycle_1.0.4    vctrs_0.6.5
[29] evaluate_0.23      glue_1.6.2         QuickJSR_1.0.8     codetools_0.2-19
[33] stats4_4.3.2       pkgbuild_1.4.3     fansi_1.0.6        colorspace_2.1-0
[37] rmarkdown_2.25     matrixStats_1.2.0  tools_4.3.2        loo_2.6.0
[41] pkgconfig_2.0.3    htmltools_0.5.7
```

Stan
Program 1 `model1.stan`

```
functions {
  // Mean-dispersion parameterization of gamma family
  real gamma_md_lpdf(real x, real mu, real psi) {
    return gamma_lpdf(x | inv(psi), inv(mu * psi));
  }

  real gamma_md_rng(real mu, real psi) {
    return gamma_rng(inv(psi), inv(mu * psi));
  }
}

data {
  int<lower=1> N_races;    // Total number of races
  int<lower=1> N_entrants; // Total number of entrants
  // Each entrant is assigned a unique index in [1, N_entrants]

  // Number of entrants in each race who finished
  array[N_races] int<lower=1, upper=N_entrants> race_N_entrants_f;

  // Indices for extracting finished entrant information in each race
  array[N_races] int race_f_start_idxs;
  array[N_races] int race_f_end_idxs;

  // Total number of entrant finishes across all races
  int <lower=1> N_entrances_fs;

  // Finished entrant indices within each race
  array[N_entrances_fs] int race_entrant_f_idxs;

  // Entrant finish times within each race
  array[N_entrances_fs] real race_entrant_f_times;

  // Anchor configuration
  int<lower=1, upper=N_races> anchor_race_idx;
  int<lower=1, upper=N_entrants> anchor_entrant_idx;
}

parameters {
  real eta; // Log baseline finish time (log seconds)

  // Relative seed difficulties
  array[N_races] real rel_difficulties_free;

  // Relative entrant skills
  array[N_entrants] real rel_skills_free;

  real<lower=0> psi; // Gamma dispersion configuration
}

transformed parameters {
```

Stan
Program 2 `model2.stan`

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

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

data {
  int<lower=1> N_races;    // Total number of races
  int<lower=1> N_entrants; // Total number of entrants
  // Each entrant is assigned a unique index in [1, N_entrants]

  // Number of entrants in each race who finished
  array[N_races] int<lower=1, upper=N_entrants> race_N_entrants_f;

  // Indices for extracting finished entrant information in each race
  array[N_races] int race_f_start_idxs;
  array[N_races] int race_f_end_idxs;

  // Total number of entrant finishes across all races
  int <lower=1> N_entrances_fs;

  // Finished entrant indices within each race
  array[N_entrances_fs] int race_entrant_f_idxs;

  // Entrant finish times within each race
  array[N_entrances_fs] real race_entrant_f_times;

  // Anchor configuration
  int<lower=1, upper=N_races> anchor_race_idx;
  int<lower=1, upper=N_entrants> anchor_entrant_idx;
}

parameters {
  real eta; // Log baseline finish time (log seconds)

  // Relative seed difficulties
  array[N_races] real rel_difficulties_free;

  // Relative entrant skills
  array[N_entrants] real rel_skills_free;

  real<lower=0> psi; // Inverse gamma dispersion configuration
}

transformed parameters {
```

Stan
Program 3 `model3.stan`

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

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

data {
  int<lower=1> N_races;    // Total number of races
  int<lower=1> N_entrants; // Total number of entrants
  // Each entrant is assigned a unique index in [1, N_entrants]

  // Number of entrants in each race who finished
  array[N_races] int<lower=1, upper=N_entrants> race_N_entrants_f;

  // Indices for extracting finished entrant information in each race
  array[N_races] int race_f_start_idxs;
  array[N_races] int race_f_end_idxs;

  // Number of entrants in each race who forfeit and did not finish
  array[N_races] int<lower=0, upper=N_entrants> race_N_entrants_dnf;

  // Indices for extracting forfeited entrant information in each race
  array[N_races] int race_dnf_start_idxs;
  array[N_races] int race_dnf_end_idxs;

  // Total number of finishes across all races
  int <lower=1> N_entrances_fs;

  // Finished entrant indices within each race
  array[N_entrances_fs] int race_entrant_f_idxs;

  // Entrant finish times within each race
  array[N_entrances_fs] real race_entrant_f_times;

  // Total number of forfeits across all races
  int<lower=0> N_entrances_dnfs;

  // Forfeited entrant indices within each race
  array[N_entrances_dnfs] int race_entrant_dnf_idxs;

  // Anchor configuration
  int<lower=1, upper=N_races> anchor_race_idx;
  int<lower=1, upper=N_entrants> anchor_entrant_idx;
}
```

Stan
Program 4 `model4a.stan`

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

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

data {
  int<lower=1> N_races;    // Total number of races
  int<lower=1> N_entrants; // Total number of entrants
  // Each entrant is assigned a unique index in [1, N_entrants]

  // Number of entrants in each race who finished
  array[N_races] int<lower=1, upper=N_entrants> race_N_entrants_f;

  // Indices for extracting finished entrant information in each race
  array[N_races] int race_f_start_idxs;
  array[N_races] int race_f_end_idxs;

  // Number of entrants in each race who forfeit and did not finish
  array[N_races] int<lower=0, upper=N_entrants> race_N_entrants_dnf;

  // Indices for extracting forfeited entrant information in each race
  array[N_races] int race_dnf_start_idxs;
  array[N_races] int race_dnf_end_idxs;

  // Total number of finishes across all races
  int <lower=1> N_entrances_fs;

  // Finished entrant indices within each race
  array[N_entrances_fs] int race_entrant_f_idxs;

  // Entrant finish times within each race
  array[N_entrances_fs] real race_entrant_f_times;

  // Total number of forfeits across all races
  int<lower=0> N_entrances_dnfs;

  // Forfeited entrant indices within each race
  array[N_entrances_dnfs] int race_entrant_dnf_idxs;

  // MapRando versioning
  int<lower=1> N_versions;
  array[N_races] int<lower=1, upper=N_versions> version_idxs;

  // Anchor configuration
```

Stan
Program 5 `model4b.stan`

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

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

data {
  int<lower=1> N_races;    // Total number of races
  int<lower=1> N_entrants; // Total number of entrants
  // Each entrant is assigned a unique index in [1, N_entrants]

  // Number of entrants in each race who finished
  array[N_races] int<lower=1, upper=N_entrants> race_N_entrants_f;

  // Indices for extracting finished entrant information in each race
  array[N_races] int race_f_start_idxs;
  array[N_races] int race_f_end_idxs;

  // Number of entrants in each race who forfeit and did not finish
  array[N_races] int<lower=0, upper=N_entrants> race_N_entrants_dnf;

  // Indices for extracting forfeited entrant information in each race
  array[N_races] int race_dnf_start_idxs;
  array[N_races] int race_dnf_end_idxs;

  // Total number of finishes across all races
  int <lower=1> N_entrances_fs;

  // Finished entrant indices within each race
  array[N_entrances_fs] int race_entrant_f_idxs;

  // Entrant finish times within each race
  array[N_entrances_fs] real race_entrant_f_times;

  // Total number of forfeits across all races
  int<lower=0> N_entrances_dnfs;

  // Forfeited entrant indices within each race
  array[N_entrances_dnfs] int race_entrant_dnf_idxs;

  // MapRando versioning
  int<lower=1> N_versions;
  array[N_races] int<lower=1, upper=N_versions> version_idxs;

  // Anchor configuration
```

Stan
Program 6 `model5.stan`

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

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

data {
  int<lower=1> N_races;    // Total number of races
  int<lower=1> N_entrants; // Total number of entrants
  // Each entrant is assigned a unique index in [1, N_entrants]

  // Number of entrants in each race who finished
  array[N_races] int<lower=1, upper=N_entrants> race_N_entrants_f;

  // Indices for extracting finished entrant information in each race
  array[N_races] int race_f_start_idxs;
  array[N_races] int race_f_end_idxs;

  // Number of entrants in each race who forfeit and did not finish
  array[N_races] int<lower=0, upper=N_entrants> race_N_entrants_dnf;

  // Indices for extracting forfeited entrant information in each race
  array[N_races] int race_dnf_start_idxs;
  array[N_races] int race_dnf_end_idxs;

  // Total number of finishes across all races
  int <lower=1> N_entrances_fs;

  // Finished entrant indices within each race
  array[N_entrances_fs] int race_entrant_f_idxs;

  // Entrant finish times within each race
  array[N_entrances_fs] real race_entrant_f_times;

  // Total number of forfeits across all races
  int<lower=0> N_entrances_dnfs;

  // Forfeited entrant indices within each race
  array[N_entrances_dnfs] int race_entrant_dnf_idxs;

  // MapRando versioning
  int<lower=1> N_versions;
  array[N_races] int<lower=1, upper=N_versions> version_idxs;

  // Anchor configuration
```