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
```

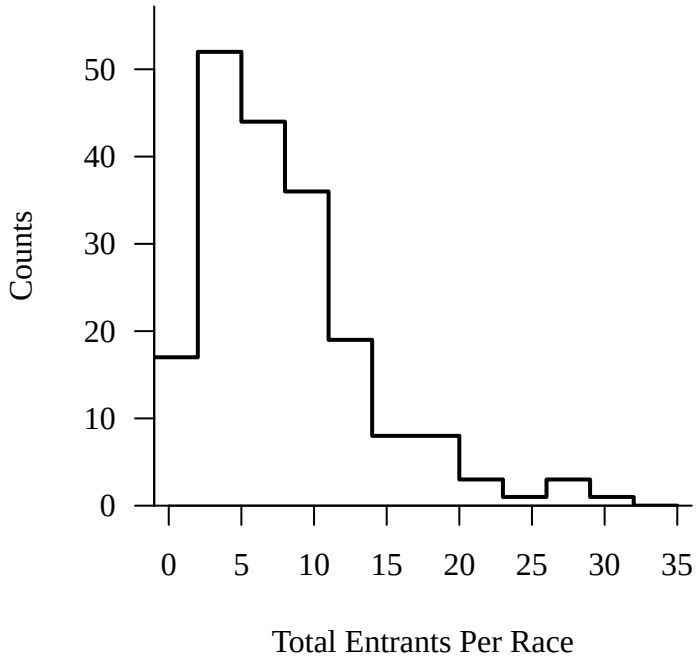
Those races included 107 distinct entrants.

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

```
107 total entrants
```

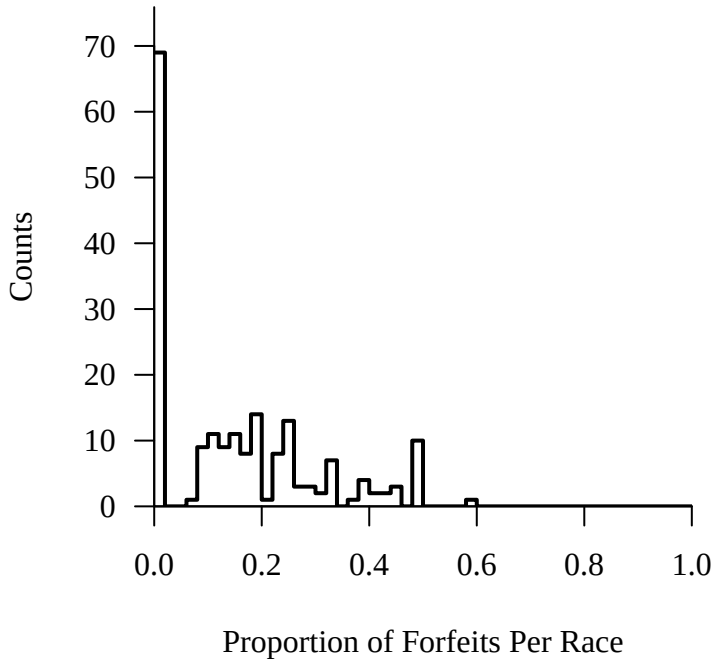
While the majority of races include at least five entrants some include over thirty.

```
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")
```



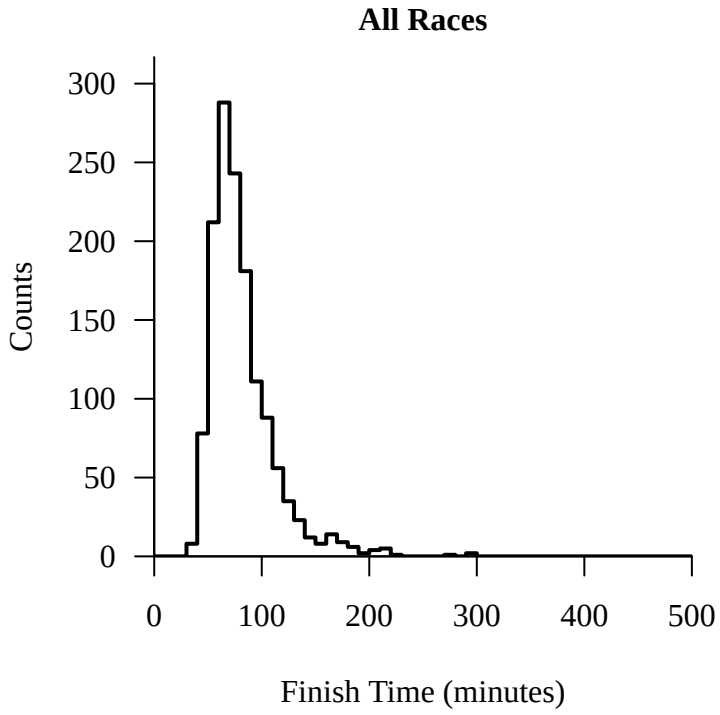
Similarly most races see most entrants finishing but some races, presumable using more difficult seeds, can see over half of the entrants forfeit.

```
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")
```



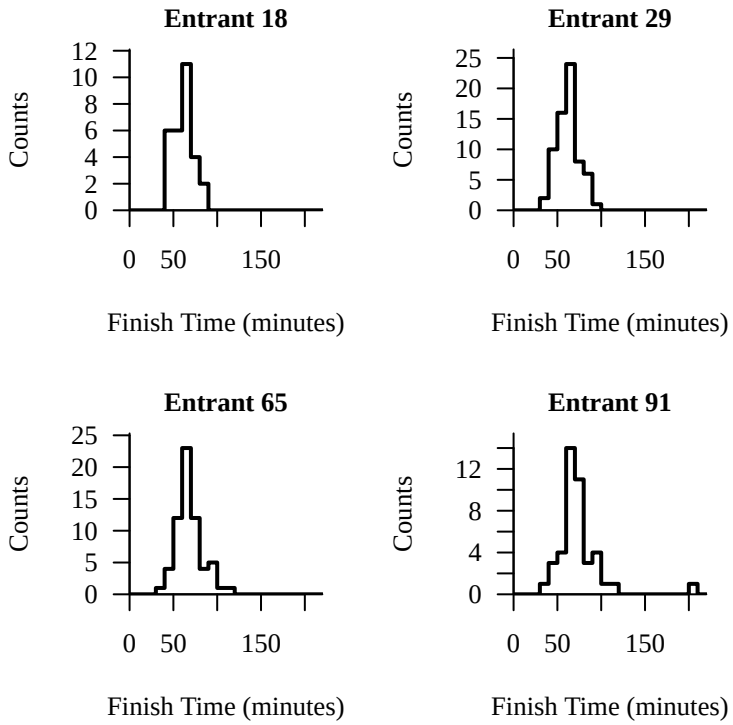
The finish times across races vary substantially, peaking near an hour but stretching from half an hour all the way to multiple hours. This suggests that player skill, seed difficulty, or some combination of the two, is highly variable from race to race.

```
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.

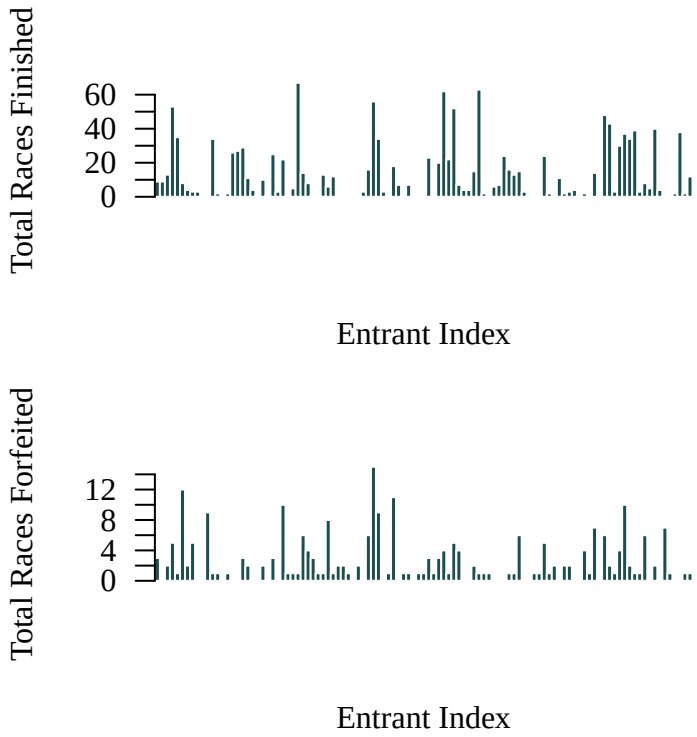
```
par(mfrow=c(2, 2), mar=c(5, 5, 2, 1))

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.

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,

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

Not all races, however, are the same.

For example some seeds result in map layouts that require traveling longer distances than others and some require spending 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,

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

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,

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

Note that this model is an example of an **adversarial model** where the result of a competition between two agents concentrates around some central value μ that increases when the ability

of the first agent, λ_1, is larger than the second, λ_2, and decreases when the ability of the second agent is larger than the first,

$$\mu = f(\lambda_1 - \lambda_2).$$

Comparing this model to other adversarial models, such as Bradley-Terry models and item response theory models, can be a productive way to motivate useful model expansions. For example if the MapRando algorithm ever suffered from weird bugs that sometimes resulted in seeds insensitive to player skill then we could introduce a discrimination parameter for each race, similar to some popular item response theory models,

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

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

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 parameterizations 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 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 seed-entrant baseline μ_{se} for each entrant in a particular race.

Finally in order 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

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

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

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

Note that this prior model doesn't suppress finish times below 30 minutes and above 90 minutes, just the central baseline. 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,

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

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

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

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 warnings indicate strong auto-correlations in γ, the difficulty parameters for some of the seeds, and the skill parameters for some of the entrants.

```
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$extract_expectands(fit)
base_samples <- util$filter_expectands(samples1,
                                       c('gamma', 'difficulties',
                                         'skills', 'psi'),
                                       check_arrays=TRUE)
util$summarize_expectand_diagnostics(base_samples)
```

```
The expectands gamma, difficulties[140], difficulties[143], skills[1],
skills[2], skills[3], skills[4], skills[5], skills[6], skills[12],
skills[16], skills[17], skills[18], skills[19], skills[22], skills[24],
skills[26], skills[29], skills[30], skills[31], skills[34], skills[35],
skills[36], skills[43], skills[44], skills[45], skills[48], skills[49],
skills[51], skills[55], skills[57], skills[58], skills[59], skills[60],
skills[61], skills[62], skills[64], skills[65], skills[68], skills[69],
skills[70], skills[71], skills[72], skills[73], skills[78], skills[81],
skills[83], skills[84], skills[88], skills[90], skills[91], skills[93],
skills[94], skills[95], skills[96], skills[98], skills[99],
skills[100], skills[105], skills[107] triggered diagnostic warnings.

The expectands gamma, difficulties[140], difficulties[143], skills[1],
skills[2], skills[3], skills[4], skills[5], skills[6], skills[12],
skills[16], skills[17], skills[18], skills[19], skills[22], skills[24],
skills[26], skills[29], skills[30], skills[31], skills[34], skills[35],
skills[36], skills[43], skills[44], skills[45], skills[48], skills[49],
skills[51], skills[55], skills[57], skills[58], skills[59], skills[60],
skills[61], skills[62], skills[64], skills[65], skills[68], skills[69],
skills[70], skills[71], skills[72], skills[73], skills[78], skills[81],
```

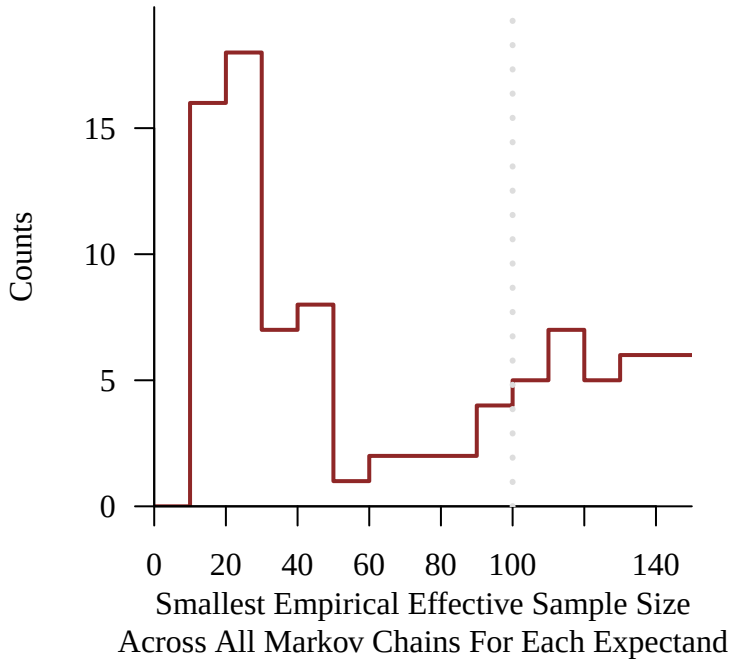
```
skills[83], skills[84], skills[88], skills[90], skills[91], skills[93],
skills[94], skills[95], skills[96], skills[98], skills[99],
skills[100], skills[105], skills[107] triggered hat{ESS} warnings.

Small empirical effective sample sizes indicate strong empirical
autocorrelations in the realized Markov chains. If the empirical
effective sample size is too small then Markov chain Monte Carlo
estimation may be unreliable even when a central limit theorem holds.
```

Fortunately the auto-correlations are not large enough to compromise the accuracy of our Markov chain Monte Carlo estimators, just limit their precision. It's only when the empirical effective sample sizes dip below ten or so that we really need to be worried.

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

min_eesss <- util$compute_min_eesss(base_samples)
util$plot_line_hist(min_eesss, 0, 150, 10, col=util$c_dark,
                    xlab=paste0("Smallest Empirical Effective Sample Size\n",
                                "Across All Markov Chains For Each Expectand"))
```

```
Warning in check_bin_containment(bin_min, bin_max, values): 212 values (70.4%)
fell below the binning.
```

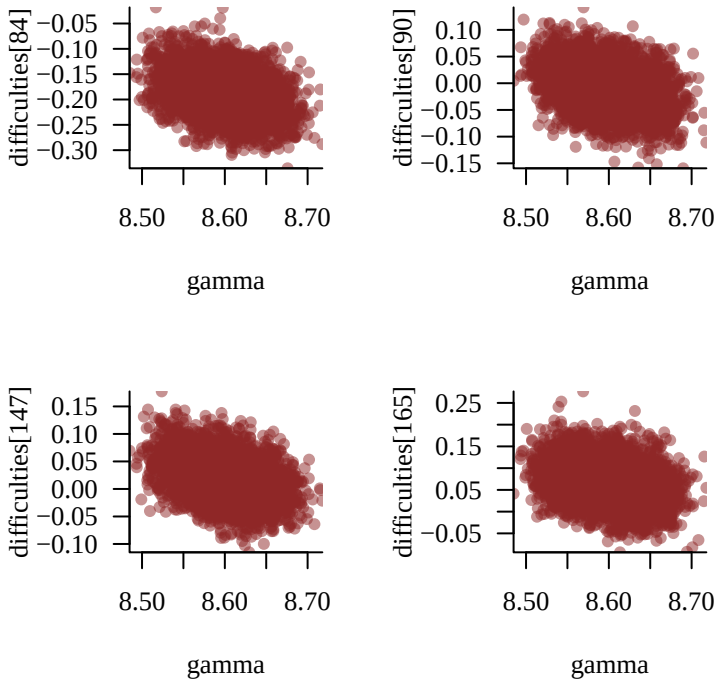
```
abline(v=100, col="#DDDDDD", lty=3, lwd=3)
```



This behavior is unfortunately not uncommon with adversarial models. Without enough data the model is vulnerable to degeneracies where some of the additive terms vary without changing their sum, tracing out a narrow plane of consistent model configurations that can be difficult to explore efficiently.

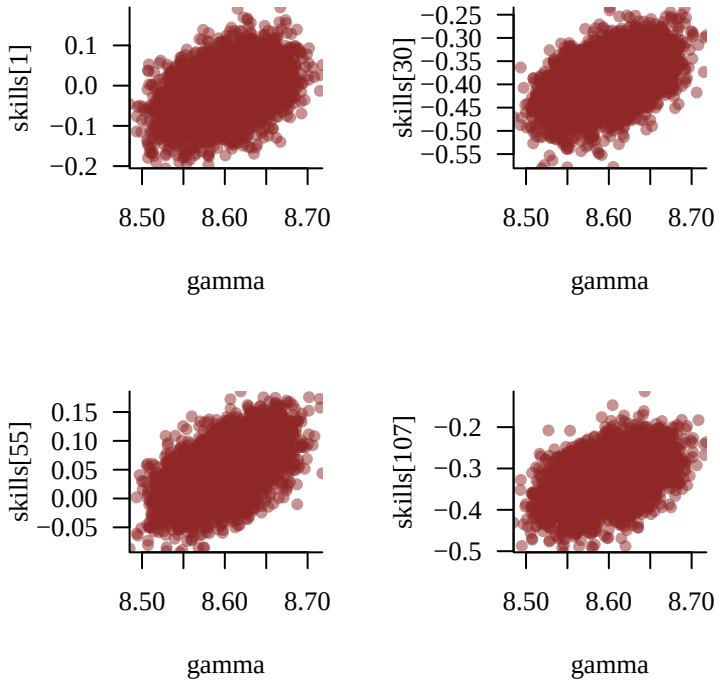
Indeed we see that the consistent values of γ are negatively correlated, albeit weakly, with the seed difficulties: γ can increase without changing the baseline finish times so long as all of the seed difficulties decrease at the same time.

```
util$plot_div_pairs(c('gamma'),
                    c('difficulties[84]', 'difficulties[90]',
                      'difficulties[147]', 'difficulties[165]'),
                    samples1, diagnostics1)
```



On the other hand γ is positively correlated with the individual skill parameters. In this case increases to γ and all of the entrant skills cancel to give the same baseline finish times.

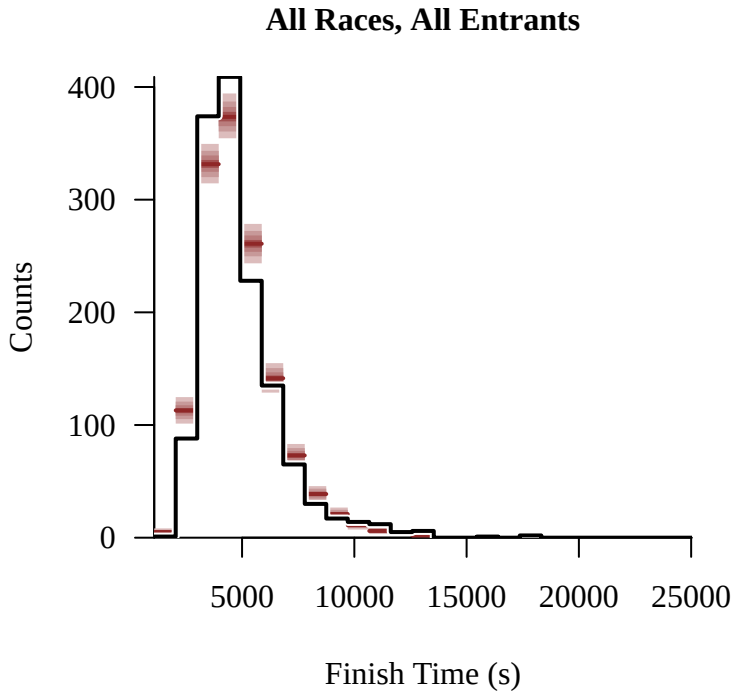
```
util$plot_div_pairs(c('gamma'),
                    c('skills[1]', 'skills[30]',
                      'skills[55]', 'skills[107]'),
                    samples1, diagnostics1)
```



If we wanted to be careful then we could run longer Markov chains to compensate for the large auto-correlations and ensure more well-behaved Markov chain Monte Carlo estimation. As this is our first model, however, let's just push ahead to the posterior retrodictive checks.

There are a lot of summary statistics that we might consider for our visual posterior retrodictive checks. 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")
```



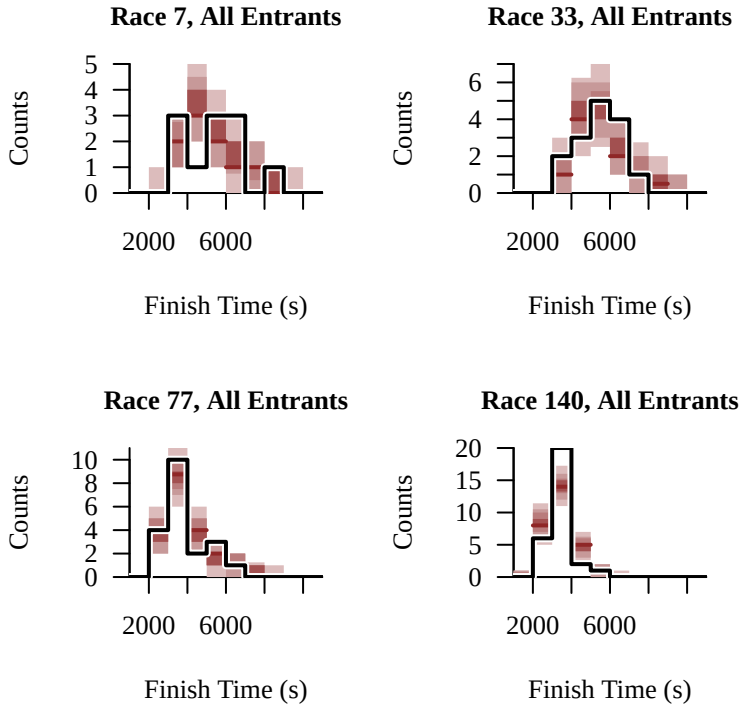
This tension could be due to inadequacy of the gamma observational model, but it could also be a consequence of poorly modeling the heterogeneity in seed difficulties and entrant skills. One way to explore these possibilities is to separate the histogram summary statistic by race and entrant.

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"): 49 predictive values (0.1%) fell below the binning.

Warning in check_bin_containment(bin_min, bin_max, collapsed_values,
"predictive value"): 108 predictive values (0.2%) fell below the binning.
```



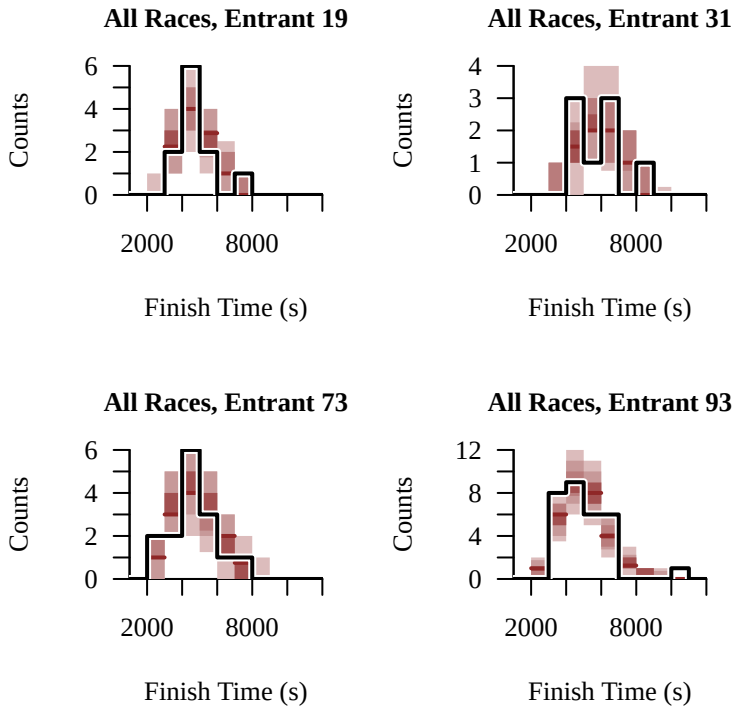
Similarly the finish time behaviors for a few spot-checked entrants are consistent between the observed data and our posterior predictions. One might argue that the observed behavior for entrant 93 is slightly heavier-tailed than the posterior predictions but the disagreement is relatively weak.

```
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"): 1 predictive value (0.0%) fell below the binning.
```

```
Warning in check_bin_containment(bin_min, bin_max, collapsed_values,
"predictive value"): 11 predictive values (0.0%) fell below 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. At the same time 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 computational diagnostics continue to show strong auto-correlation warnings but nothing else.

```
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$extract_expectands(fit)
base_samples <- util$filter_expectands(samples2,
                                       c('gamma', 'difficulties',
                                         'skills', 'psi'),
                                       check_arrays=TRUE)
util$summarize_expectand_diagnostics(base_samples)
```

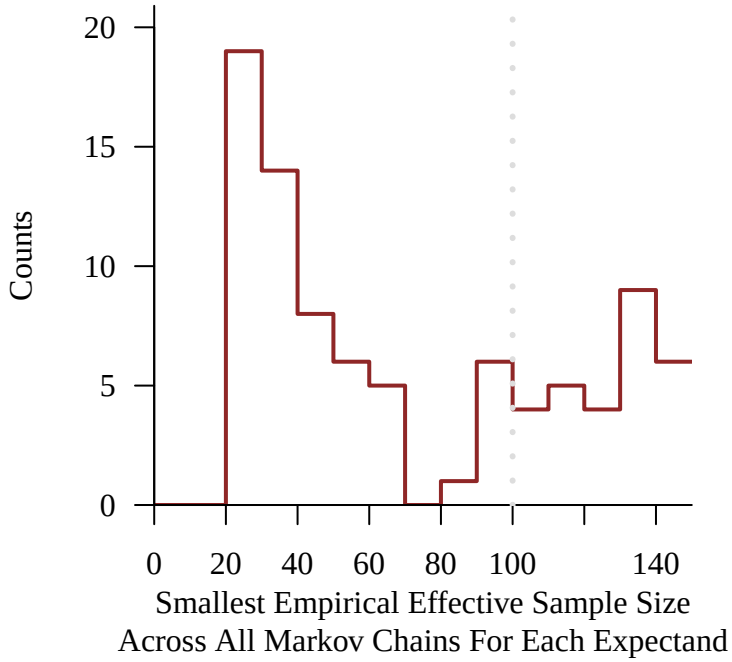
```
The expectands gamma, difficulties[140], difficulties[147], skills[1],
skills[2], skills[3], skills[4], skills[5], skills[6], skills[12],
skills[16], skills[17], skills[18], skills[19], skills[22], skills[24],
skills[26], skills[28], skills[29], skills[30], skills[31], skills[34],
skills[35], skills[36], skills[43], skills[44], skills[45], skills[48],
skills[49], skills[51], skills[55], skills[57], skills[58], skills[59],
skills[60], skills[61], skills[62], skills[64], skills[65], skills[68],
skills[69], skills[70], skills[71], skills[72], skills[73], skills[78],
skills[81], skills[88], skills[90], skills[91], skills[93], skills[94],
skills[95], skills[96], skills[98], skills[99], skills[100],
skills[105], skills[107] triggered diagnostic warnings.

The expectands gamma, difficulties[140], difficulties[147], skills[1],
skills[2], skills[3], skills[4], skills[5], skills[6], skills[12],
skills[16], skills[17], skills[18], skills[19], skills[22], skills[24],
skills[26], skills[28], skills[29], skills[30], skills[31], skills[34],
skills[35], skills[36], skills[43], skills[44], skills[45], skills[48],
skills[49], skills[51], skills[55], skills[57], skills[58], skills[59],
skills[60], skills[61], skills[62], skills[64], skills[65], skills[68],
skills[69], skills[70], skills[71], skills[72], skills[73], skills[78],
skills[81], skills[88], skills[90], skills[91], skills[93], skills[94],
skills[95], skills[96], skills[98], skills[99], skills[100],
skills[105], skills[107] triggered hat{ESS} warnings.

Small empirical effective sample sizes indicate strong empirical
autocorrelations in the realized Markov chains. If the empirical
effective sample size is too small then Markov chain Monte Carlo
estimation may be unreliable even when a central limit theorem holds.
```

Again the auto-correlations are not strong enough to undermine our Markov chain Monte Carlo estimators entirely.

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

min_eesss <- util$compute_min_eesss(base_samples)
util$plot_line_hist(min_eesss, 0, 150, 10, col=util$c_dark,
                    xlab=paste0("Smallest Empirical Effective Sample Size\n",
                                "Across All Markov Chains For Each Expectand"))
```

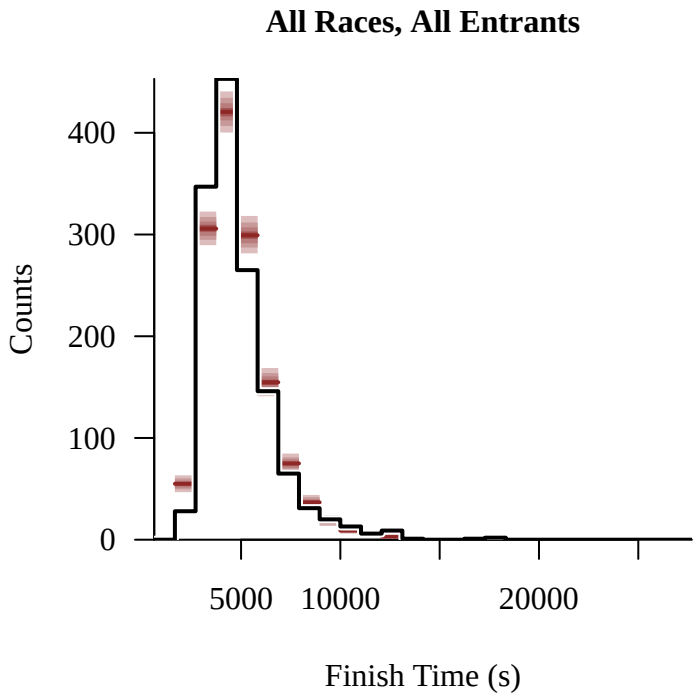
```
Warning in check_bin_containment(bin_min, bin_max, values): 214 values (71.1%)
fell below the binning.
```

```
abline(v=100, col="#DDDDDD", lty=3, lwd=3)
```



It looks like this 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")
```



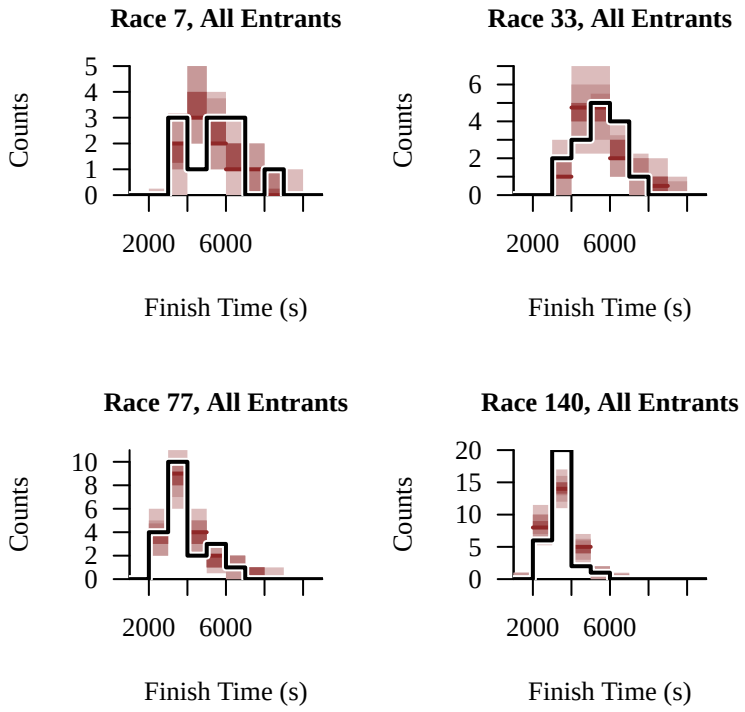
The retrodictive agreement in the individual race finish time histograms is similar to what we saw above. In particular no new retrodictive tensions have arisen.

```
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"): 71 predictive values (0.2%) fell below the binning.
```

```
Warning in check_bin_containment(bin_min, bin_max, collapsed_values,
"predictive value"): 133 predictive values (0.2%) fell below the binning.
```

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



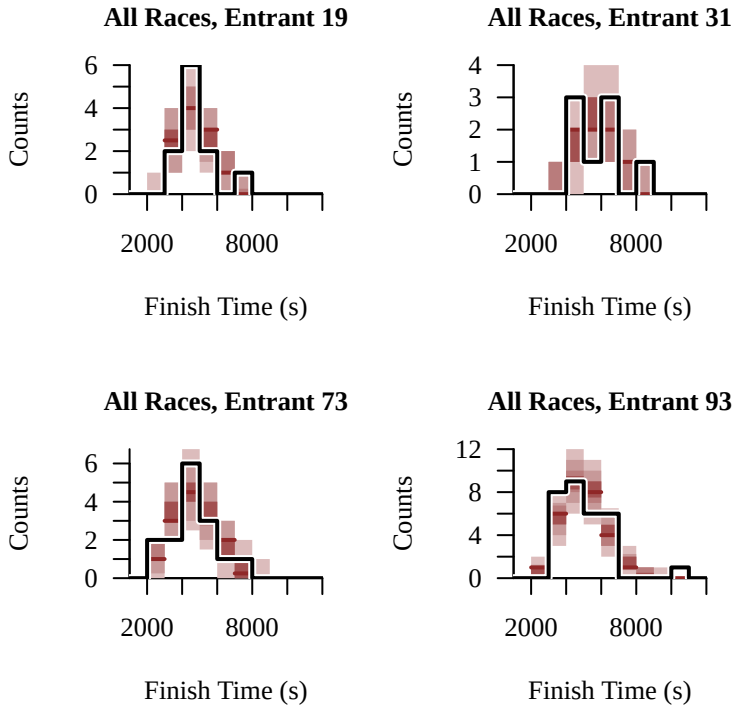
Interestingly the heavier tail of the inverse gamma family appears to allow the posterior predictive behavior for entrant 93 to spread out further and better match the observed behavior.

```
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"): 9 predictive values (0.0%) fell below the binning.
```

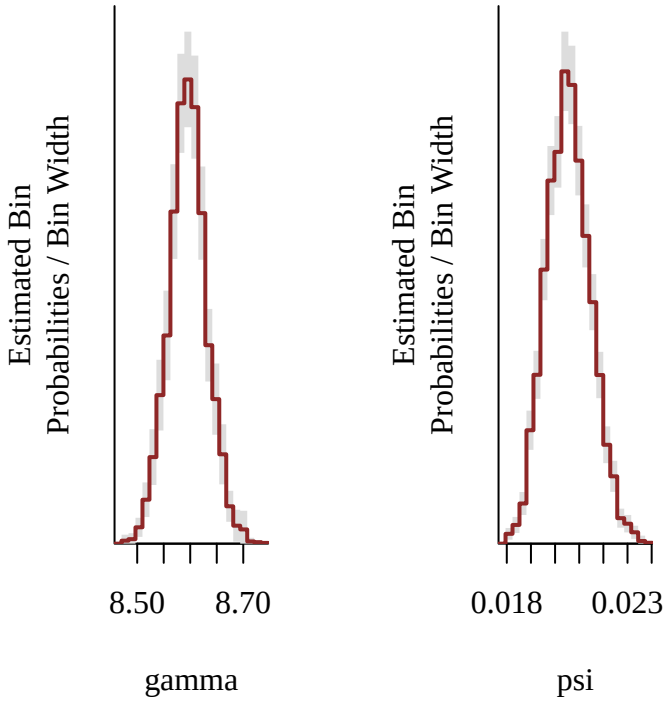
```
Warning in check_bin_containment(bin_min, bin_max, collapsed_values,
"predictive value"): 2 predictive values (0.0%) fell below the binning.
```

```
Warning in check_bin_containment(bin_min, bin_max, collapsed_values,
"predictive value"): 11 predictive values (0.0%) fell below the binning.
```



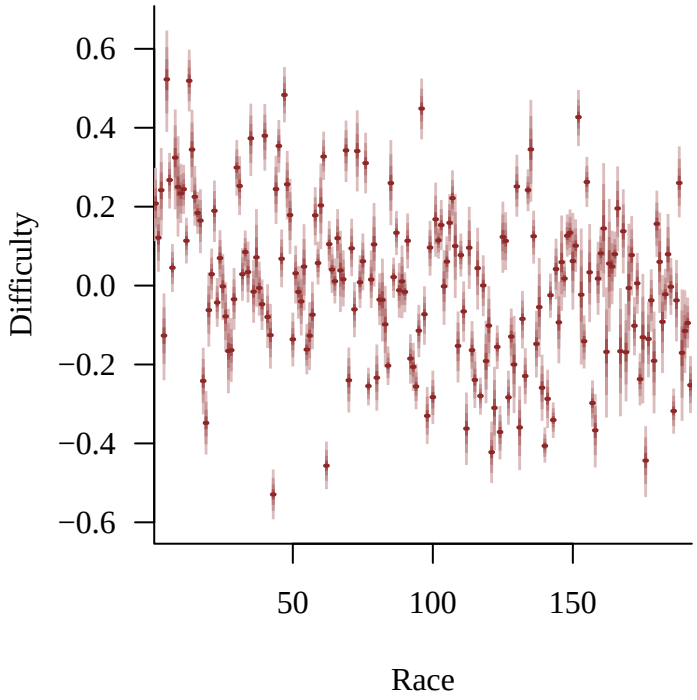
With no immediate reason to doubt our modeling assumptions we can finally move on to investigating our posterior inferences. The marginal posterior distributions for γ and ψ look reasonable, with both strongly contracting within the prior model.

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

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

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



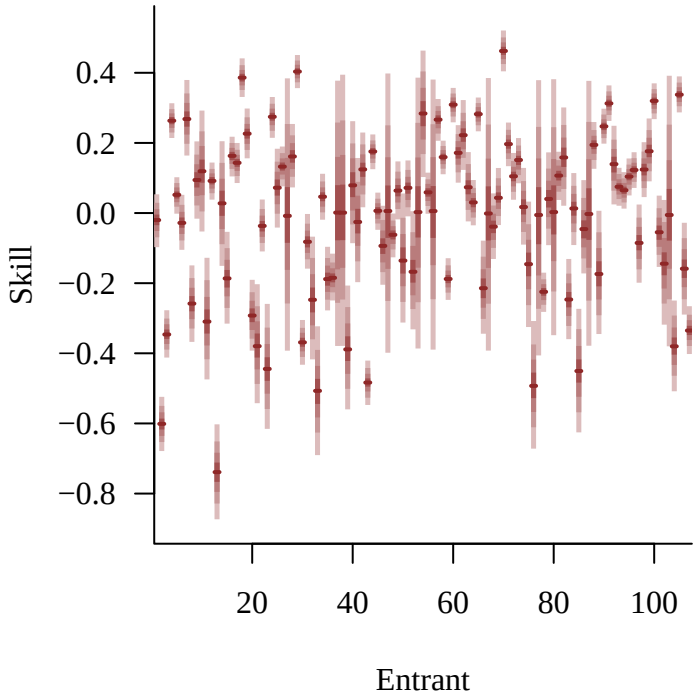
While the values of the individual seed difficulties all seem reasonable there does appear to be an unexpected pattern across the races. Initially the difficulties systematically decay before flatting out.

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

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



On the other hand the 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$N_entrants,
                function(n) paste0('skills[', n,']'))
util$plot_disc_pushforward_quantiles(samples2, names,
                                     xlab="Entrant",
                                     ylab="Skill")
```



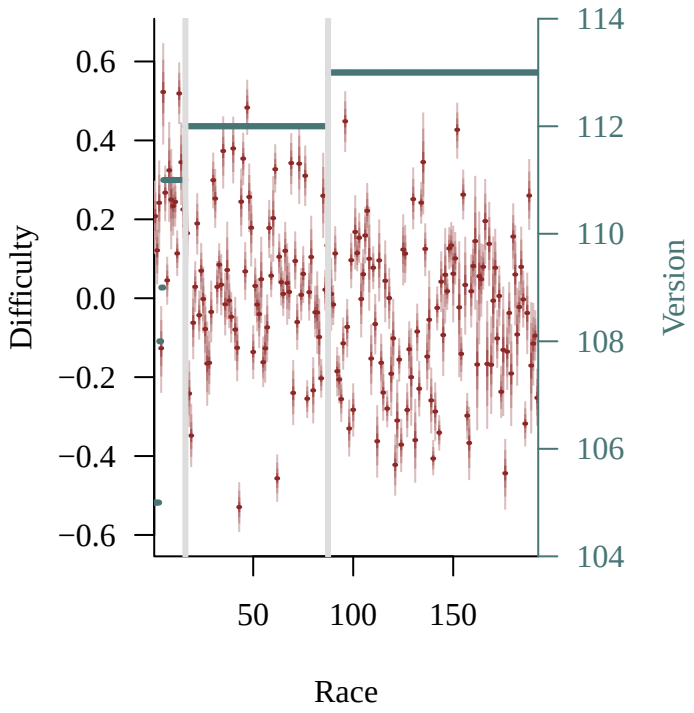
Let's go back to the difficulties and consider why we might see a pattern like that. Notice that the seeds for each race are ordered by the time at which the race occurred. Consequently the pattern we see is likely a relationship between seed difficulty and time.

One possibility is that the seed difficulties are actually getting easier. 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, this improvement could manifest only as 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$N_races,
                function(r) paste0('difficulties[', r,']'))
util$plot_disc_pushforward_quantiles(samples2, names,
                                     xlab="Race",
```

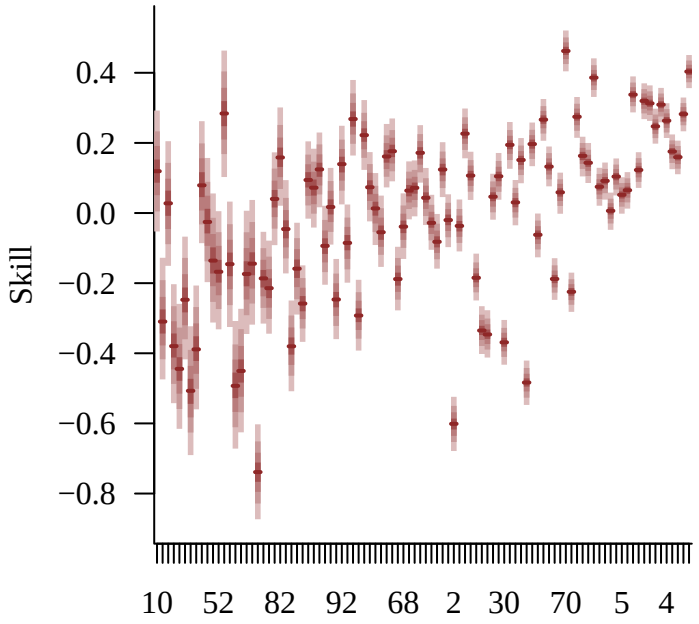
```
                                    ylab="Difficulty")

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)
```



Indeed many of the prominent patterns in seed difficulty over time perfectly line up with the transition from one version to another. In hindsight this is completely reasonable as each version improves the randomization logic to be more consistent and easier for experienced 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$Var1,
                function(n) paste0('skills[', n, ']'))
util$plot_disc_pushforward_quantiles(samples2, names,
                                     xlab="Entrants Ordered By Total Entrances",
                                     xticklabs=sorted_entrances$Var1,
                                     ylab="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.

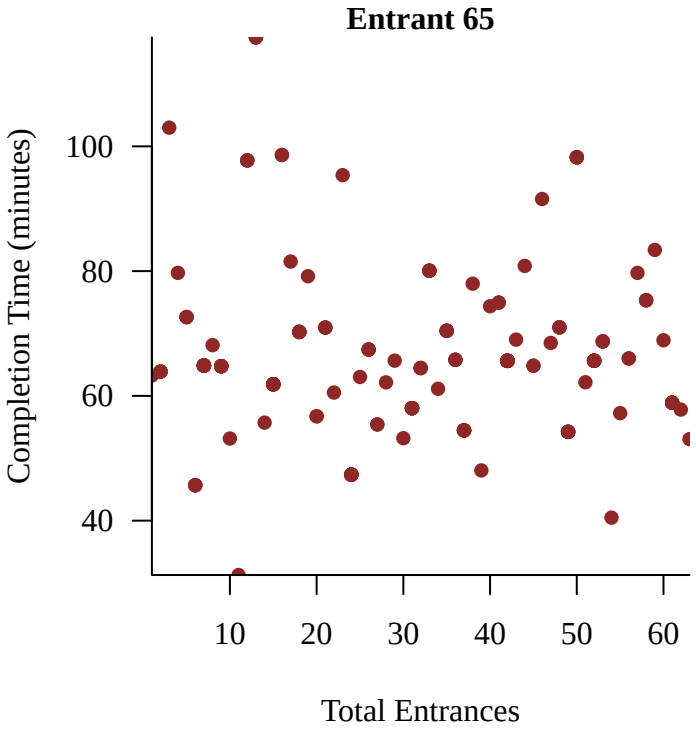
That said this increase is not necessarily tied to increased experience. For example entrant skills might be fixed with more skilled players just enjoying the MapRando races more and hence playing more.

In order to distinguish between these possibilities we would 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 the in 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 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 quantifies the threshold contrast where an entrant achieves a forfeit probability of $\frac{1}{2}$ and β 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 is limited to only positive values.

Beyond this functional form it's not straightforward to elicit domain expertise about reasonable values for κ and β. Here let's just take a prior model that constraint κ and β 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)
```

The diagnostics continue to complain about strong auto-correlations but no new problems have arisen.

```
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.
```

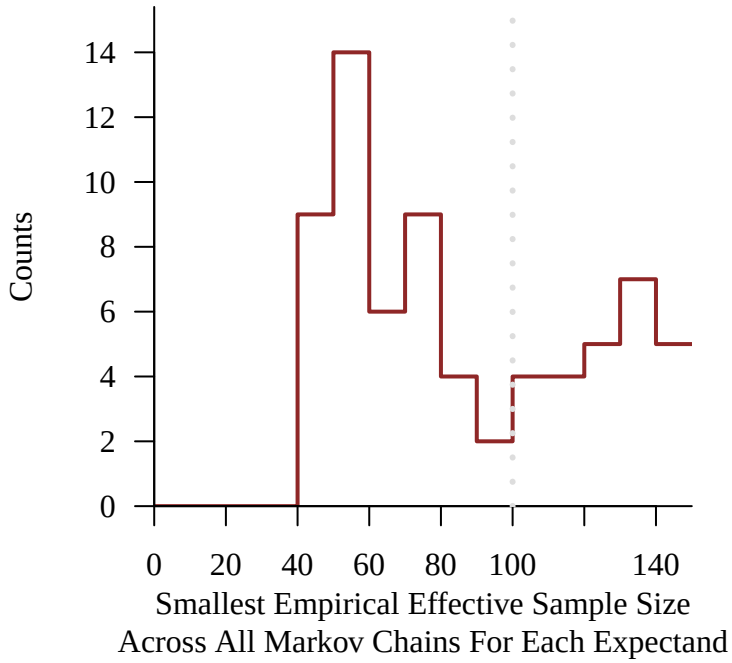
```
samples3 <- util$extract_expectands(fit)
base_samples <- util$filter_expectands(samples3,
                                       c('gamma', 'difficulties',
                                         'skills', 'kappas',
                                         'betas', 'psi'),
                                       check_arrays=TRUE)
util$summarize_expectand_diagnostics(base_samples)
```

```
The expectands gamma, skills[1], skills[3], skills[4], skills[5],
skills[12], skills[16], skills[17], skills[18], skills[22], skills[24],
skills[26], skills[29], skills[30], skills[34], skills[36], skills[43],
skills[44], skills[45], skills[48], skills[55], skills[57], skills[58],
skills[59], skills[60], skills[64], skills[65], skills[70], skills[71],
skills[72], skills[73], skills[78], skills[81], skills[88], skills[90],
skills[91], skills[93], skills[94], skills[95], skills[96],
skills[100], skills[105], skills[107], kappas[44], kappas[94],
kappas[98] triggered diagnostic warnings.

The expectands gamma, skills[1], skills[3], skills[4], skills[5],
skills[12], skills[16], skills[17], skills[18], skills[22], skills[24],
skills[26], skills[29], skills[30], skills[34], skills[36], skills[43],
skills[44], skills[45], skills[48], skills[55], skills[57], skills[58],
skills[59], skills[60], skills[64], skills[65], skills[70], skills[71],
skills[72], skills[73], skills[78], skills[81], skills[88], skills[90],
skills[91], skills[93], skills[94], skills[95], skills[96],
skills[100], skills[105], skills[107], kappas[44] triggered hat{ESS}
warnings.

Small empirical effective sample sizes indicate strong empirical
autocorrelations in the realized Markov chains. If the empirical
effective sample size is too small then Markov chain Monte Carlo
estimation may be unreliable even when a central limit theorem holds.
```

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

min_eesss <- util$compute_min_eesss(base_samples)
util$plot_line_hist(min_eesss, 0, 150, 10, col=util$c_dark,
                    xlab=paste0("Smallest Empirical Effective Sample Size\n",
                                "Across All Markov Chains For Each Expectand"))
```

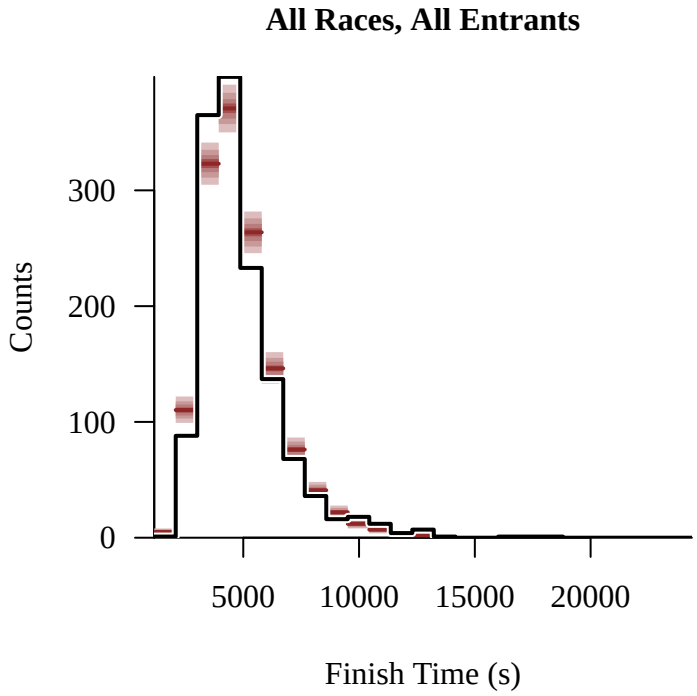
```
Warning in check_bin_containment(bin_min, bin_max, values): 446 values (86.6%)
fell below the binning.
```

```
abline(v=100, col="#DDDDDD", lty=3, lwd=3)
```



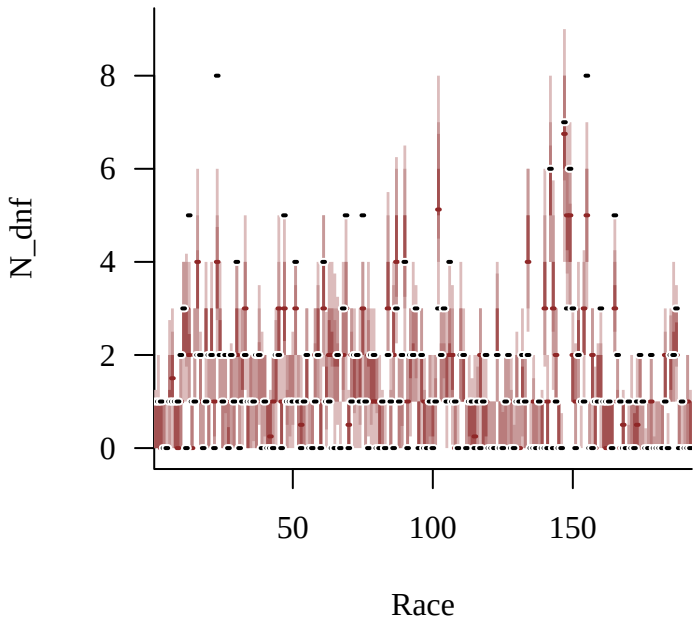
The retrodictive agreement between the observed and posterior predictive finish time histograms continues.

```
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")
```



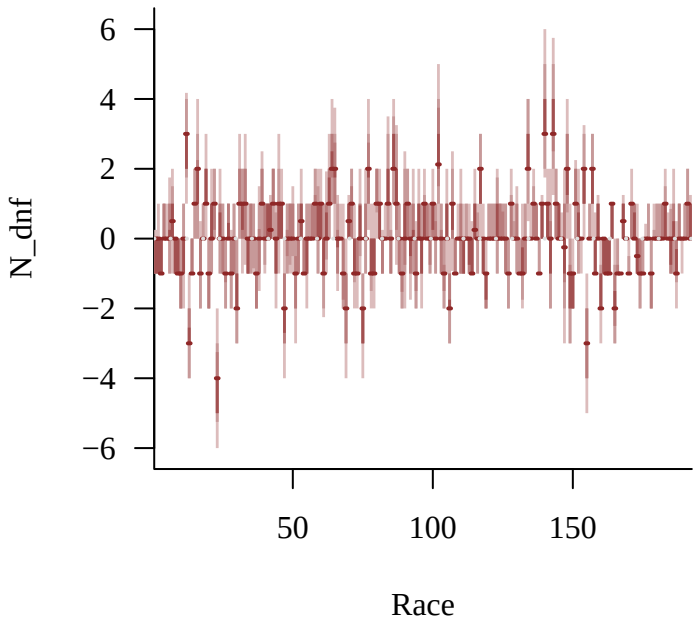
Now we can also consider the number of forfeits in each race. Fortunately the behavior of this statistic is also reasonably consistent.

```
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.

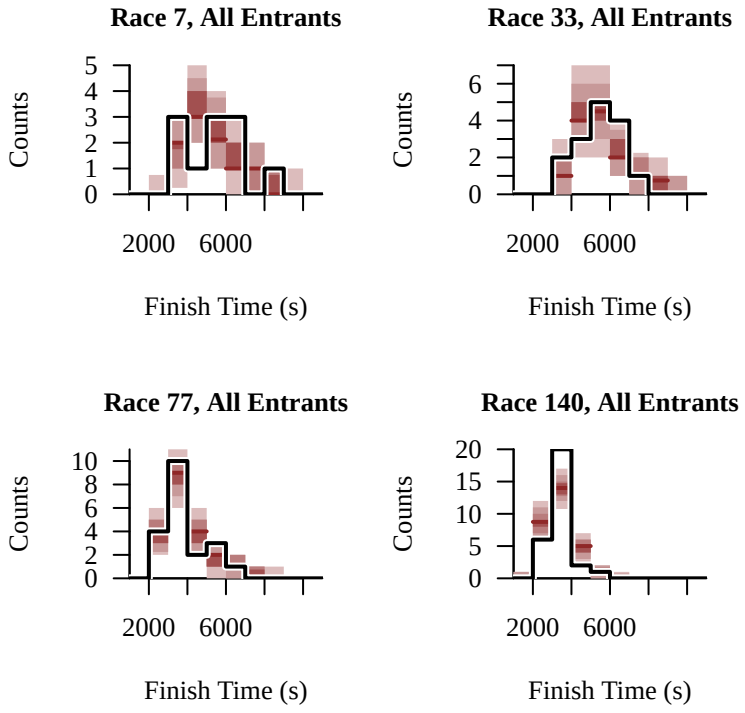
```
par(mfrow=c(1, 1), mar=c(5, 5, 3, 1))

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.

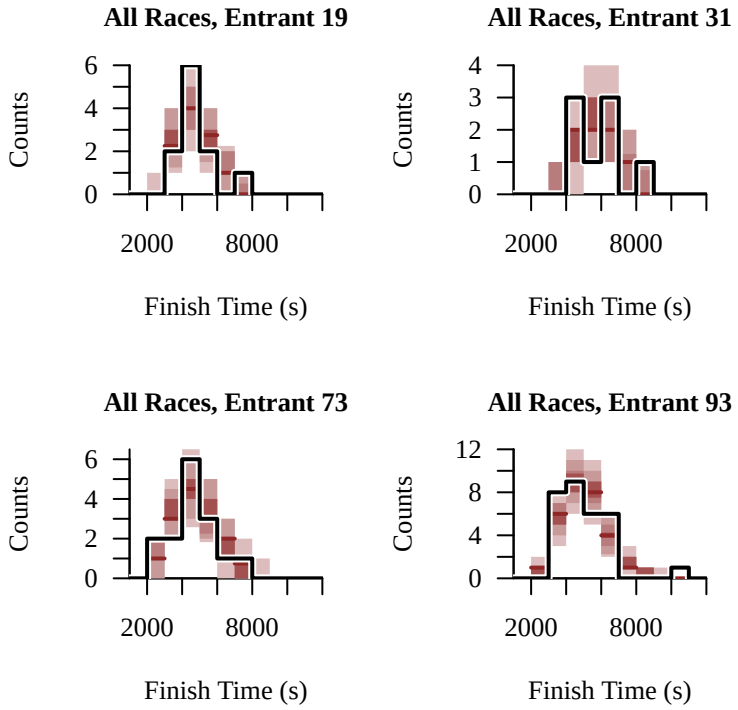
```
par(mfrow=c(2, 2), mar=c(5, 5, 3, 1))

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"): 99 predictive values (0.2%) fell below the binning.

Warning in check_bin_containment(bin_min, bin_max, collapsed_values,
"predictive value"): 152 predictive values (0.2%) fell below the binning.

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



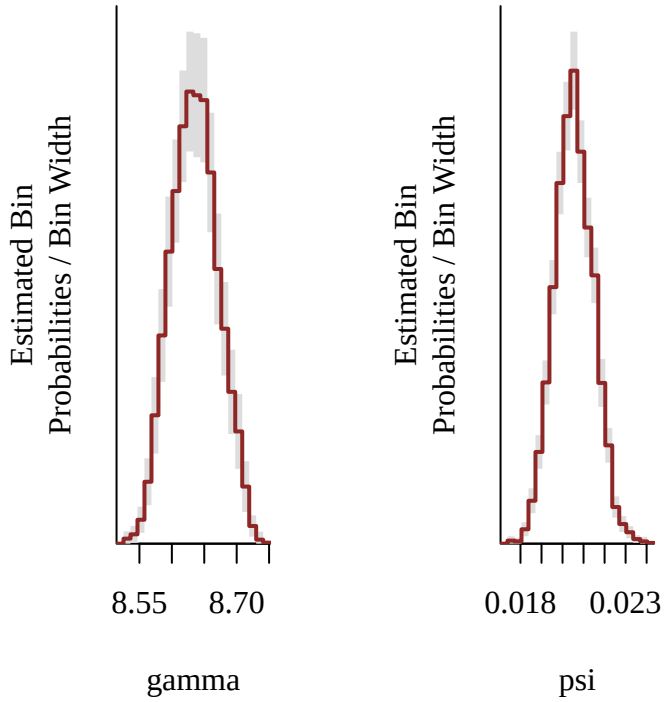

```
par(mfrow=c(2, 2), mar=c(5, 5, 3, 1))

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"): 6 predictive values (0.0%) fell below the binning.

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

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

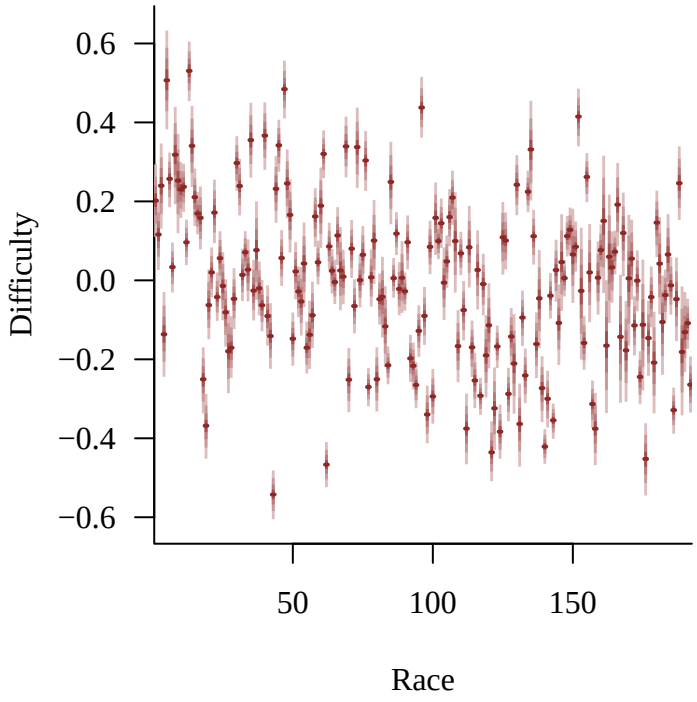


Without any concerns about our modeling assumptions we can move on to examining the resulting posterior inferences. Inferences for the existing parameters are at least superficially similar to those from the second model; we'll make a more direct comparison in Section 4.5.

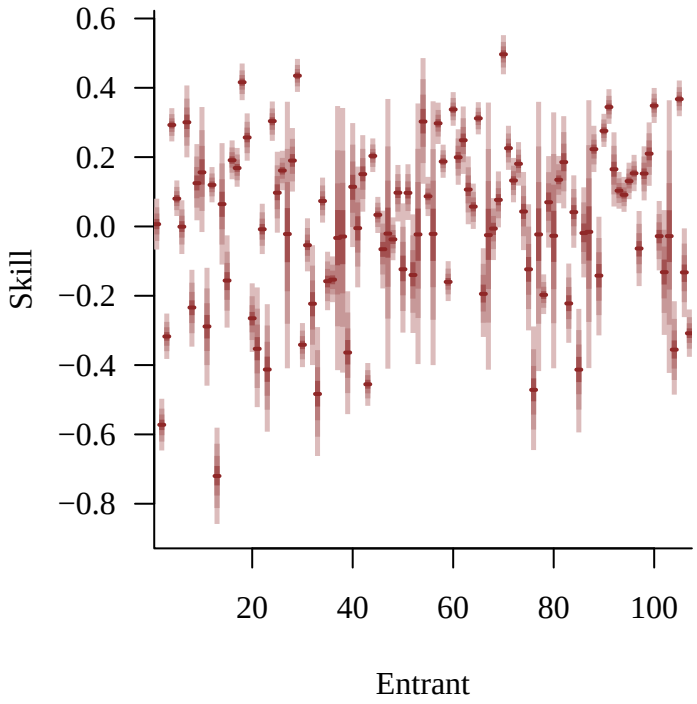
```
par(mfrow=c(1, 2), mar=c(5, 5, 1, 1))

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

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$N_races,
                function(r) paste0('difficulties[', r,']'))
util$plot_disc_pushforward_quantiles(samples3, names,
                                     xlab="Race",
                                     ylab="Difficulty")
```

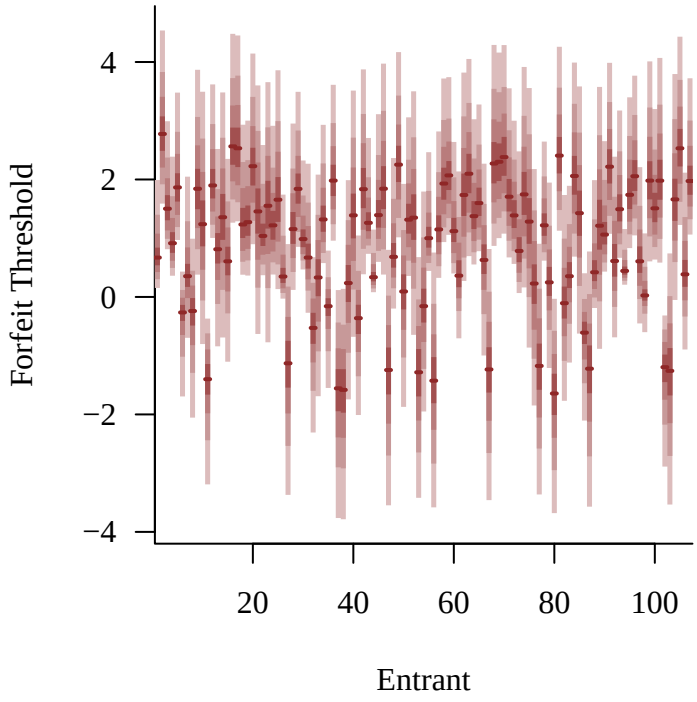



```
par(mfrow=c(1, 1), mar=c(5, 5, 1, 1))

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

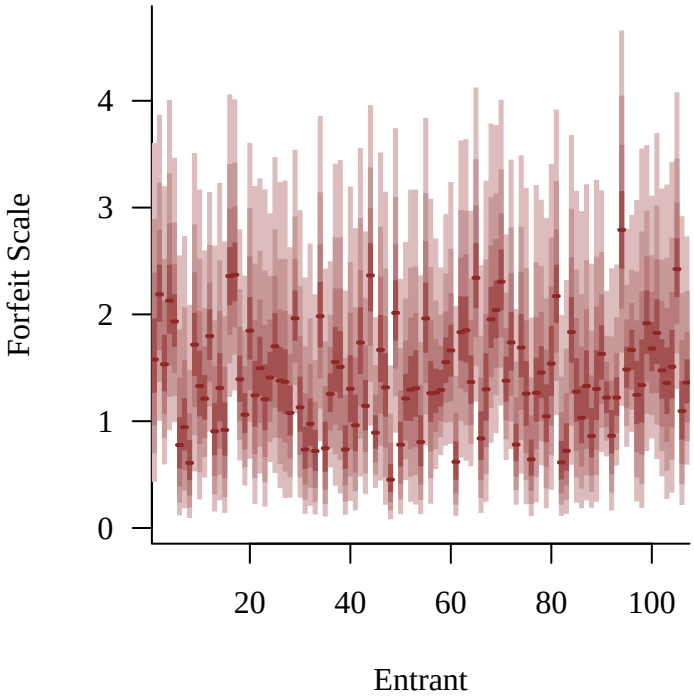


More interesting here are the posterior inferences for the new, forfeit-related parameters. Overall the uncertainties are relatively large but we can pick out a few exceptional behaviors

```
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")
```

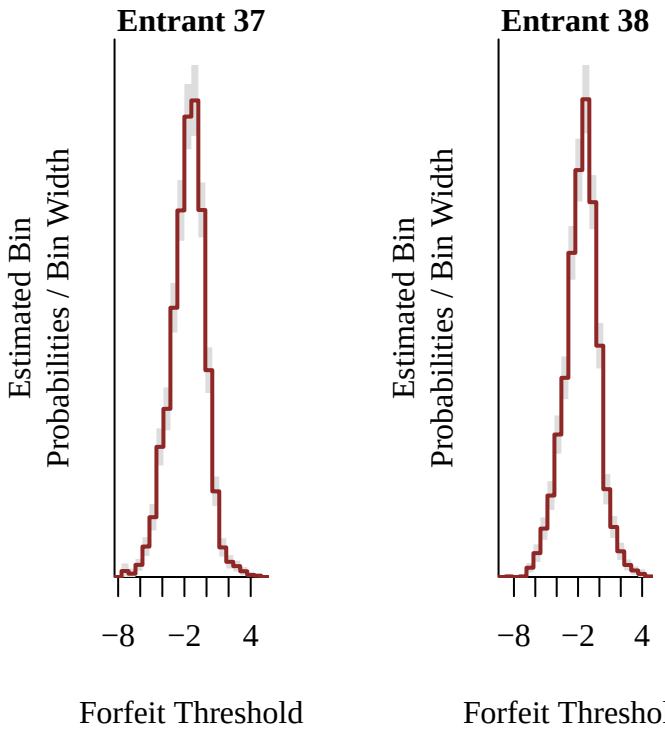



```
par(mfrow=c(1, 1), mar=c(5, 5, 1, 1))

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)
```

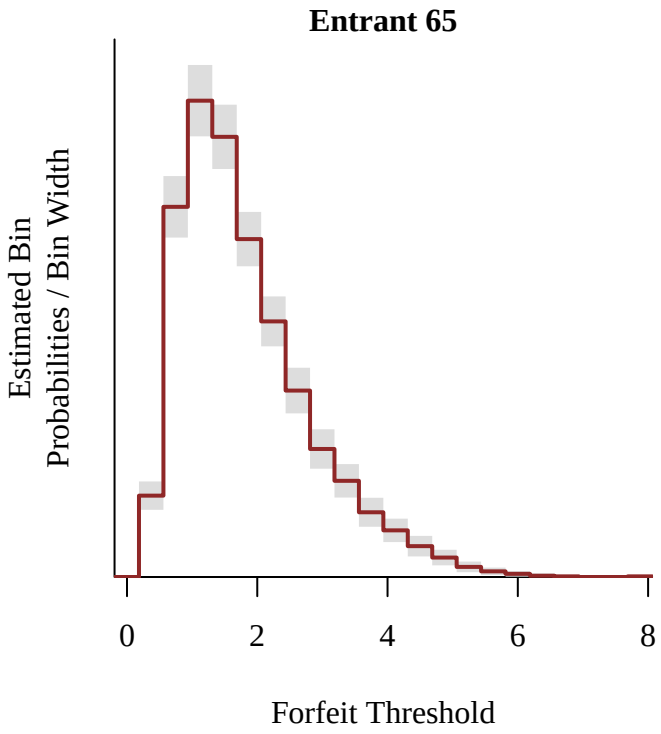
```
Entrant 37
  2 total entrances
  2 forfeits (100.0%)
```

```
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.

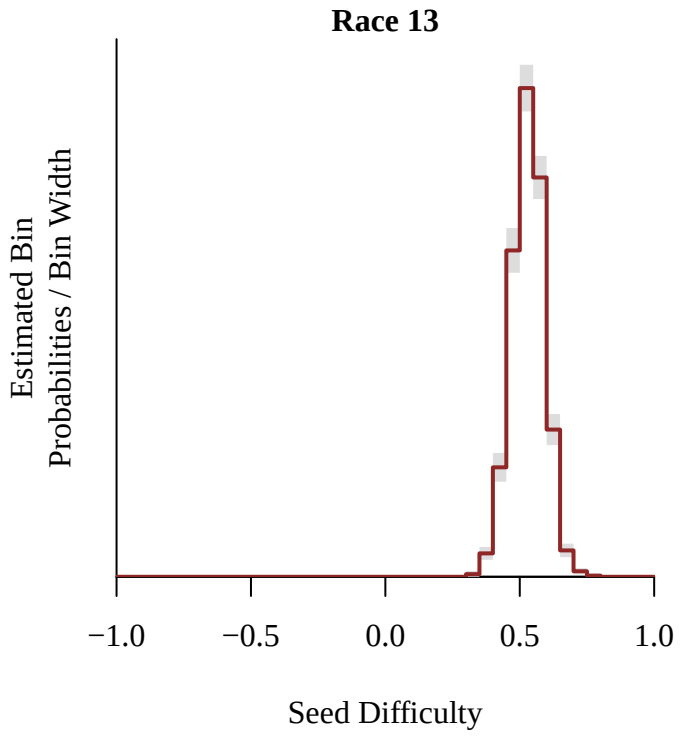
```
summarize_entrant(e)
```

```
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.

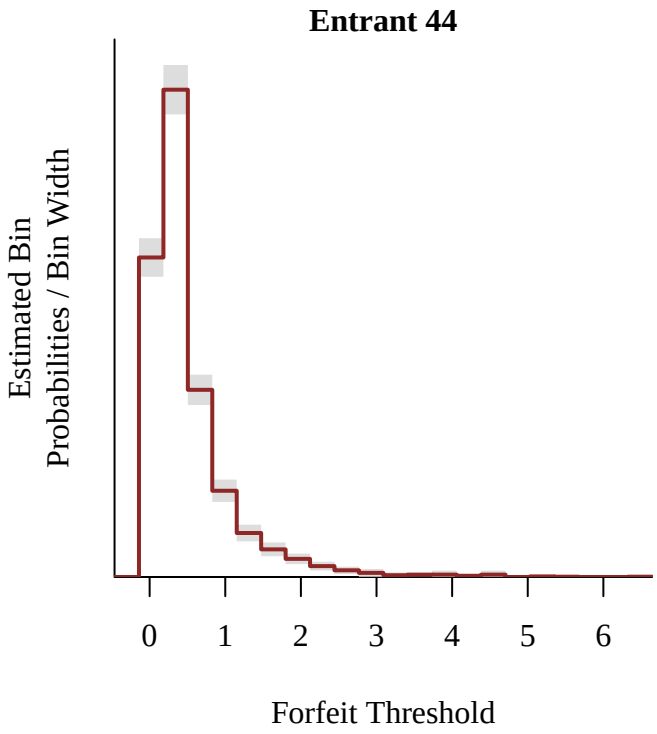
```
dnf_races <- c()
for (r in 1:data$N_races) {
  if (data$race_N_entrants_dnf[r] == 0) next
  idxs <- data$race_dnf_start_idxs[r]:data$race_dnf_end_idxs[r]
  if (e %in% data$race_entrant_dnf_idxs[idxs])
    dnf_races <- c(dnf_races, r)
}

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

```
util$plot_expectand_pushforward(samples3[[name]], 40, flim=c(-1, 1),
                                display_name="Seed Difficulty",
                                main=paste('Race', dnf_races[1]))
```



Finally the posterior inferences of the forfeit threshold for entrant 44 mostly concentrates 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 current model is to couple the behavior across seeds and entrants, allowing data to be shared and 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, however, 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 a 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 within 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 seed difficulties and entrant skills are modeled with one-dimensional, and unconstrained, real values we can reach for the standard normal hierarchical model. The last step we then need in order to fully define the model is a parameterization of the individual parameters in each hierarchy. Here I will use a monolithic non-centered parameterization for all of the hierarchies and hope that the large number of seeds and entrants results 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/model4.stan",
            data=data, seed=8438338,
            warmup=1000, iter=2024, refresh=0)
```

Fortunately we don't see any of the tell-tale signs of problematic hierarchical geometries, such as divergences and E-FMI warnings.

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

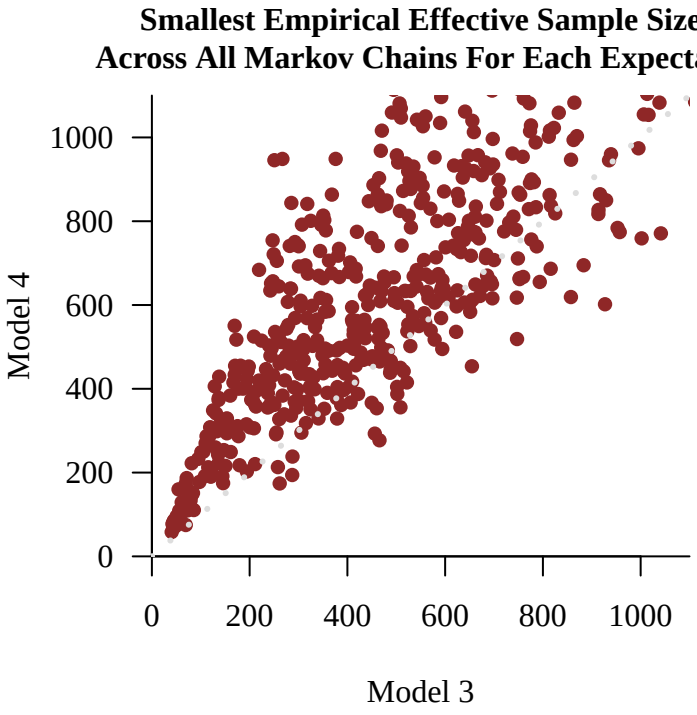
```
  All Hamiltonian Monte Carlo diagnostics are consistent with reliable
Markov chain Monte Carlo.
```

```
samples4 <- util$extract_expectands(fit)
base_samples <- util$filter_expectands(samples4,
                                       c('gamma',
                                         'eta_difficulties',
                                         'tau_difficulties',
                                         'eta_skills',
                                         'tau_skills',
                                         'kappas', 'betas',
                                         'psi'),
                                       check_arrays=TRUE)
util$summarize_expectand_diagnostics(base_samples)
```

```
The expectands gamma, tau_difficulties[6], eta_skills[4],
eta_skills[5], eta_skills[12], eta_skills[16], eta_skills[17],
eta_skills[18], eta_skills[24], eta_skills[29], eta_skills[44],
eta_skills[45], eta_skills[58], eta_skills[60], eta_skills[65],
eta_skills[70], eta_skills[90], eta_skills[91], eta_skills[93],
eta_skills[94], eta_skills[95], eta_skills[96], eta_skills[100],
eta_skills[105], kappas[44], kappas[86], kappas[94], kappas[98]
triggered diagnostic warnings.

The expectands gamma, tau_difficulties[6], eta_skills[4],
eta_skills[5], eta_skills[12], eta_skills[16], eta_skills[17],
eta_skills[18], eta_skills[24], eta_skills[29], eta_skills[44],
eta_skills[45], eta_skills[58], eta_skills[60], eta_skills[65],
eta_skills[70], eta_skills[90], eta_skills[91], eta_skills[93],
eta_skills[94], eta_skills[95], eta_skills[96], eta_skills[100],
eta_skills[105], kappas[44] triggered hat{ESS} warnings.

Small empirical effective sample sizes indicate strong empirical
autocorrelations in the realized Markov chains. If the empirical
effective sample size is too small then Markov chain Monte Carlo
estimation may be unreliable even when a central limit theorem holds.
```

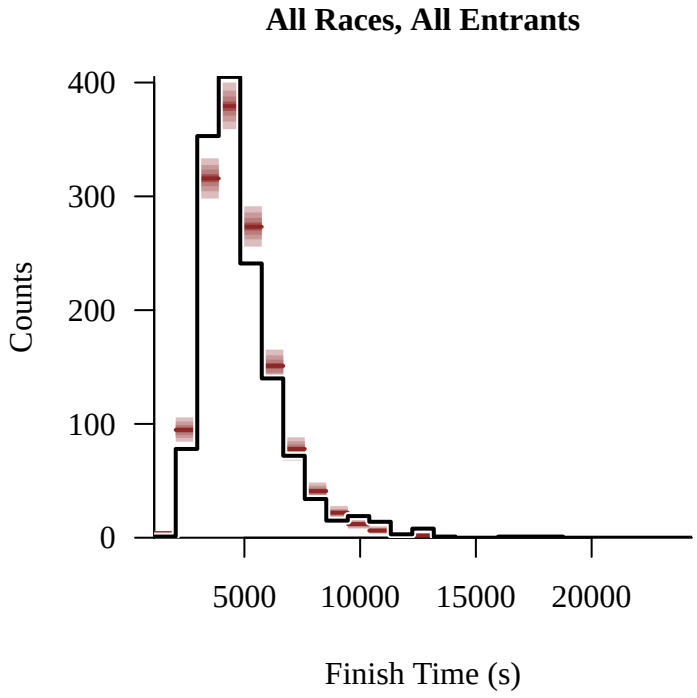
In fact the empirical effective sample sizes are consistently larger, and hence the autocorrelations consistently weaker, than before! This suggests that the hierarchical coupling is indeed reducing the posterior uncertainties and improving the overall posterior geometry.

```
base_samples <- util$filter_expectands(samples3,
                                       c('gamma',
                                         'difficulties', 'skills',
                                         'kappas', 'betas',
                                         'psi'),
                                       check_arrays=TRUE)
min_eesss3 <- util$compute_min_eesss(base_samples)

base_samples <- util$filter_expectands(samples4,
                                       c('gamma',
                                         'difficulties', 'skills',
                                         'kappas', 'betas',
                                         'psi'),
                                       check_arrays=TRUE)
min_eesss4 <- util$compute_min_eesss(base_samples)

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

plot(min_eesss3, min_eesss4, col=util$c_dark, pch=16,
     main=paste0("Smallest Empirical Effective Sample Size\n",
                 "Across All Markov Chains For Each Expectand"),
     xlab="Model 3", xlim=c(0, 1100),
     ylab="Model 4", ylim=c(0, 1100))
abline(a=0, b=1, col="#DDDDDD", lty=3, lwd=3)
```

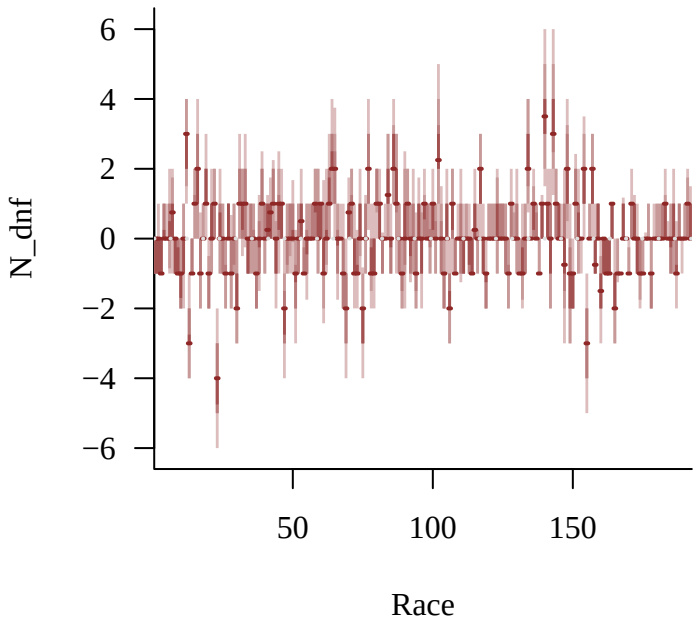


A review of our visual retrodictive checks doesn't show any indications that the introduction of the hierarchical coupling compromised the adequacy of our modeling assumptions.

```
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")
```

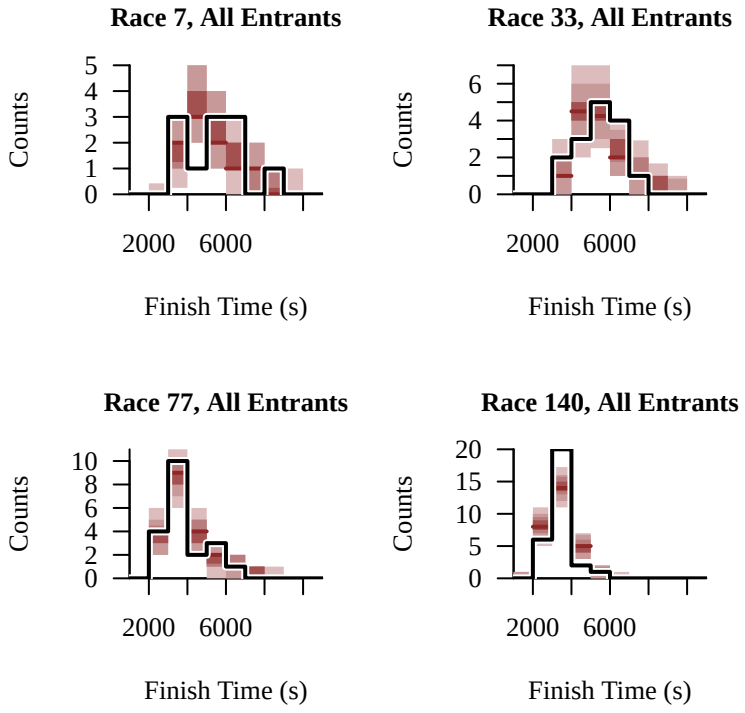



```
par(mfrow=c(1, 1), mar=c(5, 5, 3, 1))

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")
```



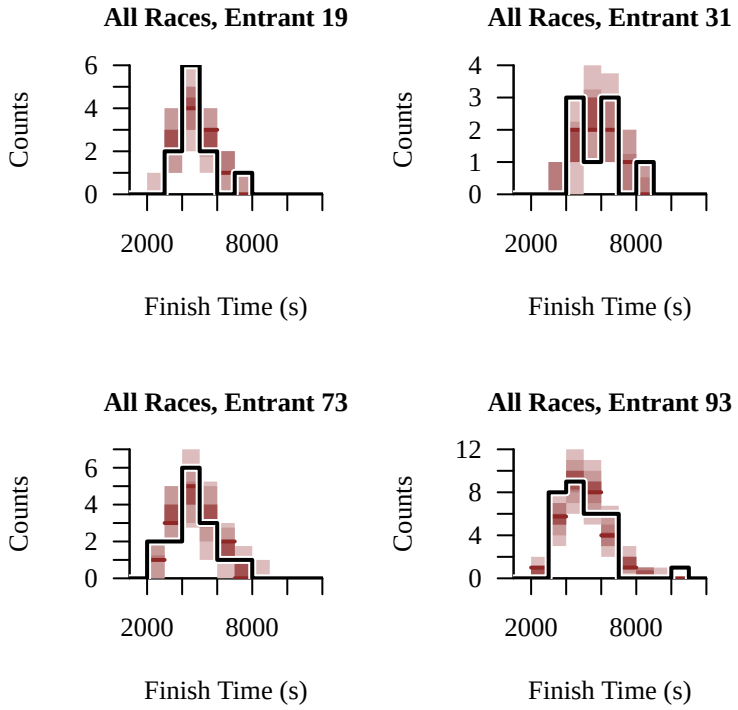

```
par(mfrow=c(2, 2), mar=c(5, 5, 3, 1))

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"): 79 predictive values (0.2%) fell below the binning.

Warning in check_bin_containment(bin_min, bin_max, collapsed_values,
"predictive value"): 87 predictive values (0.1%) fell below the binning.

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

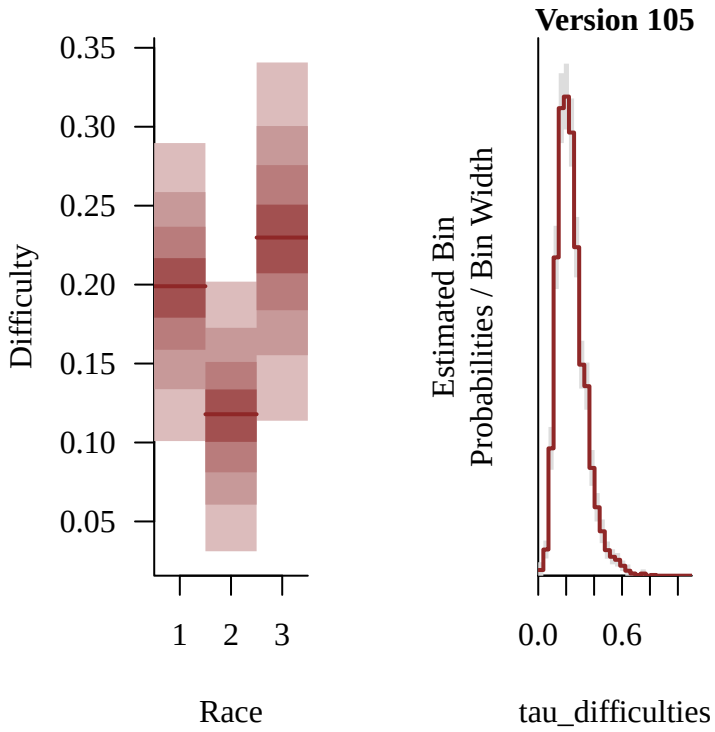
```
Warning in check_bin_containment(bin_min, bin_max, collapsed_values,
"predictive value"): 5 predictive values (0.0%) fell below the binning.

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

Warning in check_bin_containment(bin_min, bin_max, collapsed_values,
"predictive value"): 9 predictive values (0.0%) fell below the binning.
```

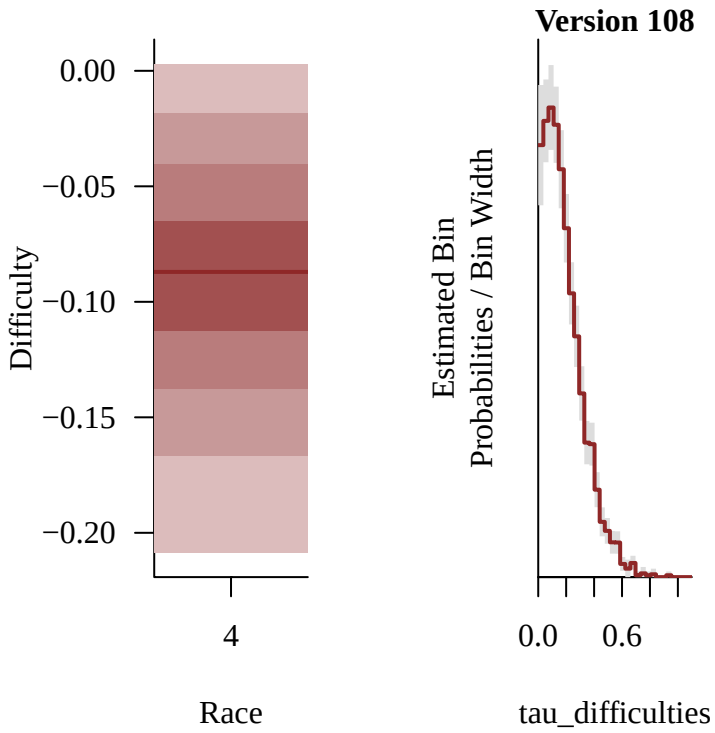


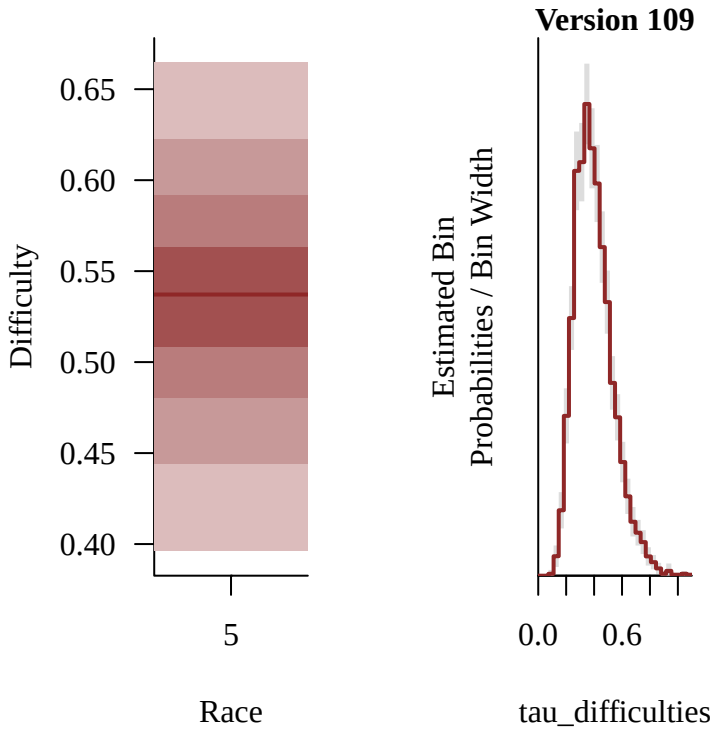
Now we can explore the posterior inferences for not just the individual behaviors but also the hierarchical populations from which those behaviors are, at least mathematically, drawn.

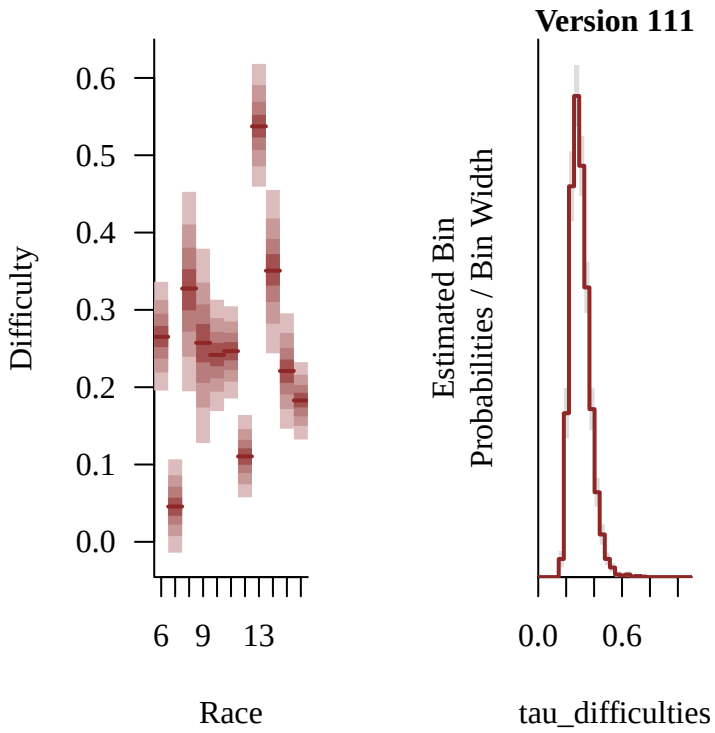
Each MapRando version defines a separate hierarchical population and, unsurprisingly, the inferred population behavior is most precise for the later versions that have been played the most.

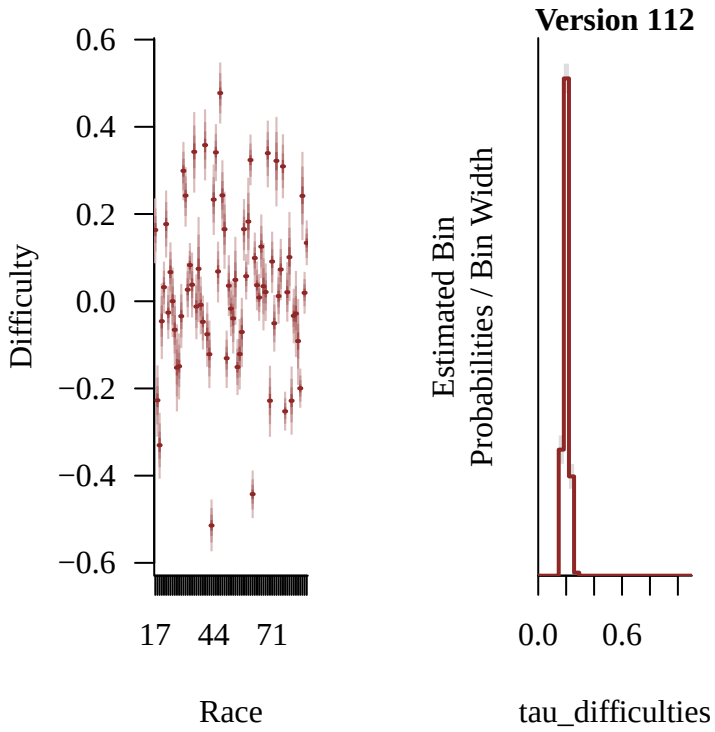
```
for (v in 1:data$N_versions) {
  par(mfrow=c(1, 2), mar=c(5, 5, 1, 1))

  races <- which(data$version_idxs == v)
  names <- sapply(races, function(r) paste0('difficulties[', r,']'))
  util$plot_disc_pushforward_quantiles(samples4, names,
                                       xlab="Race",
                                       xticklabs=races,
                                       ylab="Difficulty")

  name <- paste0('tau_difficulties[', v, ']')
  util$plot_expectand_pushforward(samples4[[name]], 30, flim=c(0, 1.1),
                                  display_name="tau_difficulties",
                                  main=paste("Version", uniq_versions[v]))
}
```

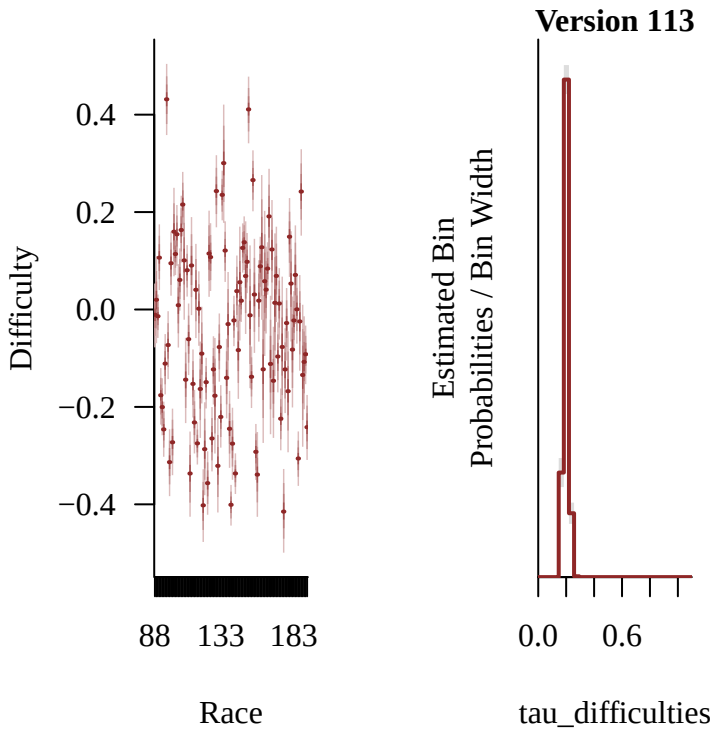

Version 105
Difficulty
0.35
0.30
0.25
0.20
0.15
0.10
0.05
1
2
3
Race
Estimated Bin Probabilities / Bin Width
0.0
0.6
tau_difficulties


Version 108
Difficulty
0.00
−0.05
−0.10
−0.15
−0.20
4
Race
Estimated Bin Probabilities / Bin Width
0.0
0.6
tau_difficulties


Version 109
Difficulty
0.65
0.60
0.55
0.50
0.45
0.40
5
Race
Estimated Bin Probabilities / Bin Width
0.0
0.6
tau_difficulties


Version 111
Difficulty
0.6
0.5
0.4
0.3
0.2
0.1
0.0
6
9
13
Race
Estimated Bin Probabilities / Bin Width
0.0
0.6
tau_difficulties





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 seed difficulty magnitudes above

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

implying range of proportional changes to the baseline finish time between

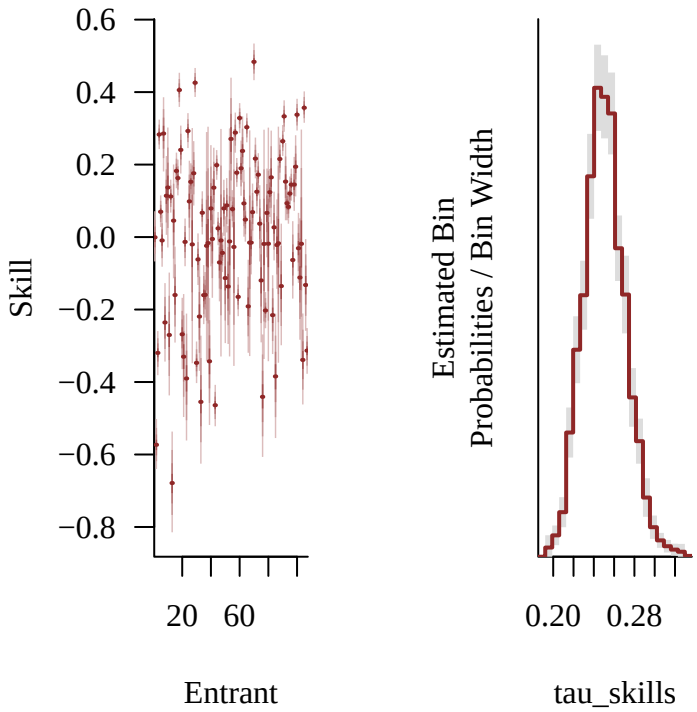
$$\exp(-0.4) \approx 0.67$$

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

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

util$plot_expectand_pushforward(samples4[['tau_skills']], 20,
                                display_name="tau_skills")
```



4.5 Inferential Comparison

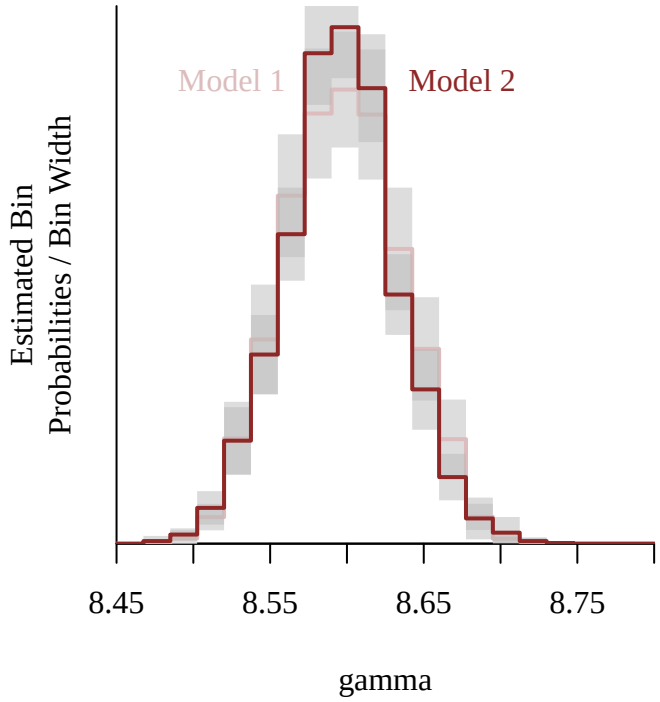
Before applying our posterior inferences to make useful statements about the entrants and their behavior in future races let's pause and examine the impact our model development has had on our posterior inferences.

4.5.1 Log Baseline

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

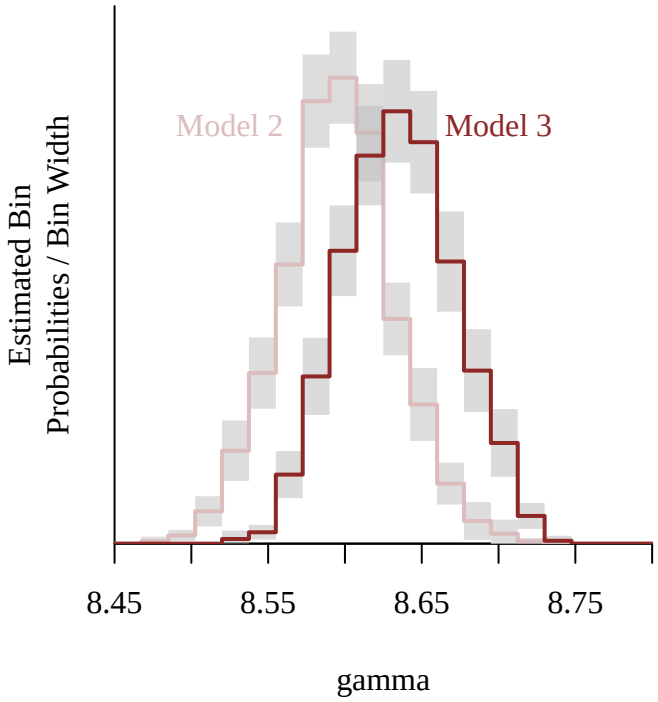
Interestingly changing the observational model doesn't seem to have strongly impacted γ, at least within the resolution of our Markov chain Monte Carlo estimators.

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

util$plot_expectand_pushforward(samples1[['gamma']], 20,
                                flim=c(8.45, 8.8),
                                display_name="gamma",
                                col=util$c_light)
text(8.525, 10, "Model 1", col=util$c_light)

util$plot_expectand_pushforward(samples2[['gamma']], 20,
                                flim=c(8.45, 8.8),
                                border="#BBBBBB88", add=TRUE)
text(8.675, 10, "Model 2", col=util$c_dark)
```



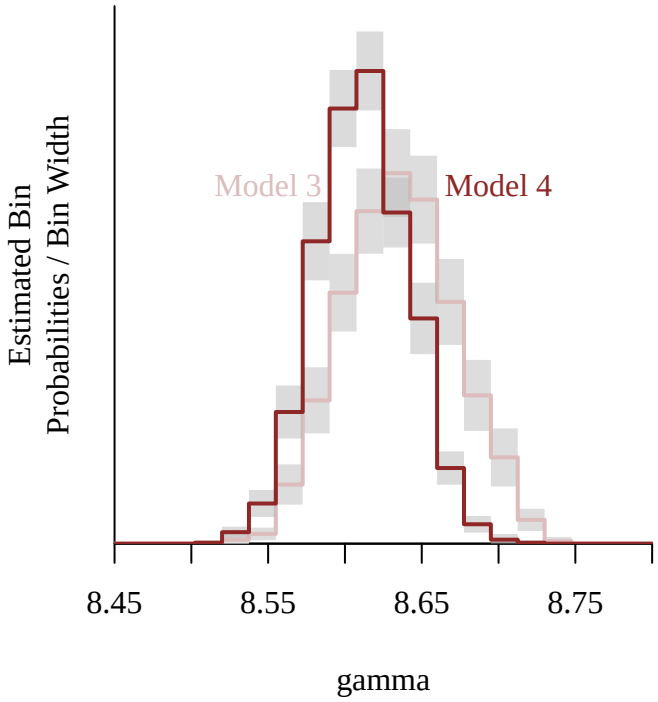
On the other hand incorporating forfeits results in a substantial shift of the entire marginal posterior distribution up to larger values, implying longer baseline finish times. This makes sense because without accounting for forfeits the observed finish times are biased towards more optimistic outcomes.

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

util$plot_expectand_pushforward(samples2[['gamma']], 20,
                                flim=c(8.45, 8.8),
                                display_name="gamma",
                                col=util$c_light)
text(8.525, 10, "Model 2", col=util$c_light)

util$plot_expectand_pushforward(samples3[['gamma']], 20,
                                flim=c(8.45, 8.8),
                                border="#BBBBBB88", add=TRUE)
text(8.7, 10, "Model 3", col=util$c_dark)
```



The introduction of the seed difficulty and entrant skill hierarchies has no impact on smaller values of γ but it does suppress larger values, giving a narrower marginal posterior distribution. This is just a manifestation of the smaller uncertainties that appropriately coupling the individuals behaviors together can give.

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

util$plot_expectand_pushforward(samples3[['gamma']], 20,
                                flim=c(8.45, 8.8),
                                ylim=c(0, 15),
                                display_name="gamma",
                                col=util$c_light)
text(8.55, 10, "Model 3", col=util$c_light)

util$plot_expectand_pushforward(samples4[['gamma']], 20,
                                flim=c(8.45, 8.8),
                                border="#BBBBBB88", add=TRUE)
text(8.7, 10, "Model 4", col=util$c_dark)
```



4.5.2 Entrant 29 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 29 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 29.

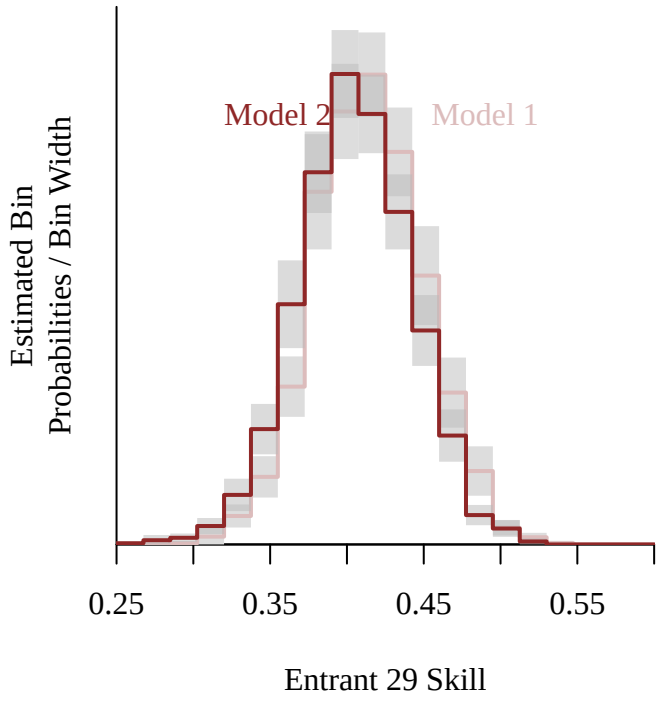
```
e <- 29
summarize_entrant(e)
```

```
Entrant 29
  68 total entrances
  67 finishes (98.5%)
  1 forfeit (1.5%)
```

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

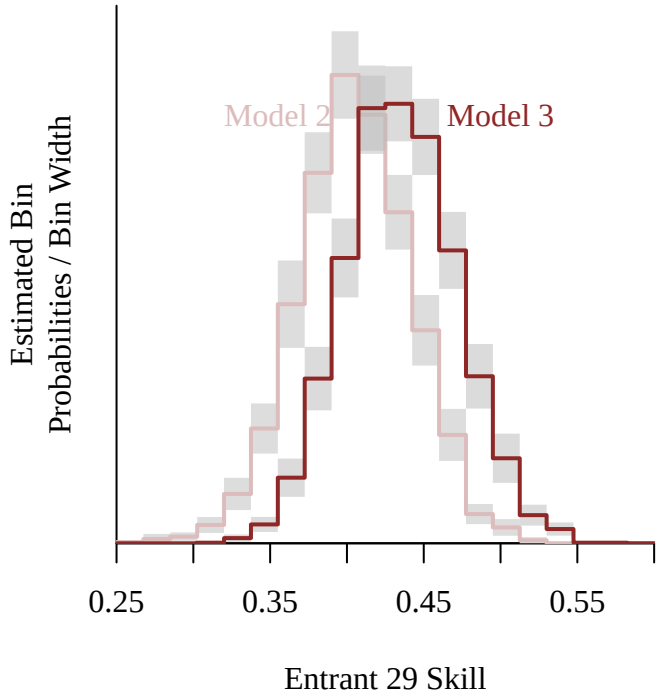
Transitioning from a gamma to inverse gamma observational model seems to yield a very slight shift of the marginal skill posterior distribution to smaller values. On the other hand because this shift is largely enveloped by the Markov chain Monte Carlo errors it could also just be a computational artifact.

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

util$plot_expectand_pushforward(samples1[[name]], 20,
                                flim=c(0.25, 0.6),
                                display_name=xname,
                                col=util$c_light)
text(0.49, 10, "Model 1", col=util$c_light)

util$plot_expectand_pushforward(samples2[[name]], 20,
                                flim=c(0.25, 0.6),
                                border="#BBBBBB88", add=TRUE)
text(0.355, 10, "Model 2", col=util$c_dark)
```



Interestingly the introduction of forfeits into the model has a much stronger impact on the marginal skill posterior distribution, shifting it up to larger values. Even though entrant 29 rarely forfeited the ignorance of forfeits can allow data from other entrants to bias inferences for common parameters like γ, which then bias inferences for all entrant skills.

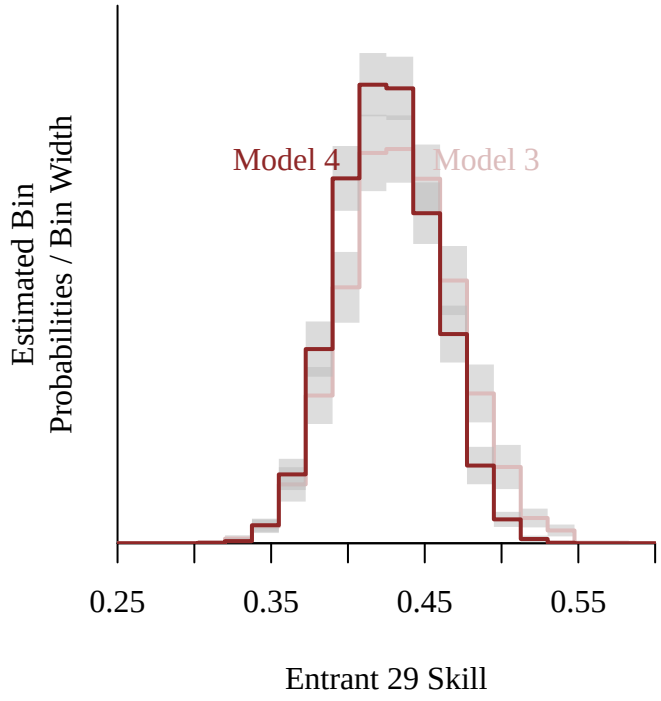
```
par(mfrow=c(1, 1), mar=c(5, 5, 1, 1))

util$plot_expectand_pushforward(samples2[[name]], 20,
                                flim=c(0.25, 0.6),
                                display_name=xname,
                                col=util$c_light)
text(0.355, 10, "Model 2", col=util$c_light)

util$plot_expectand_pushforward(samples3[[name]], 20,
                                flim=c(0.25, 0.6),
                                border="#BBBBBB88", add=TRUE)
text(0.5, 10, "Model 3", col=util$c_dark)
```



The introduction of the skill hierarchy has a similar, albeit weaker, influence on the entrant 29 skill parameter as it did on γ. Larger values are suppressed, narrowing the marginal posterior distribution.

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

util$plot_expectand_pushforward(samples3[[name]], 20,
                                flim=c(0.25, 0.6),
                                ylim=c(0, 14),
```

```
                                  display_name=xname,
                                  col=util$c_light)
text(0.49, 10, "Model 3", col=util$c_light)

util$plot_expectand_pushforward(samples4[[name]], 20,
                                  flim=c(0.25, 0.6),
                                  border="#BBBBBB88", add=TRUE)
text(0.36, 10, "Model 4", col=util$c_dark)
```



4.5.3 Entrant 44 Skill

Let's contrast these changes with those for the skill parameter of entrant 44, who also entered into many races but forfeited at a much higher rate than entrant 29.

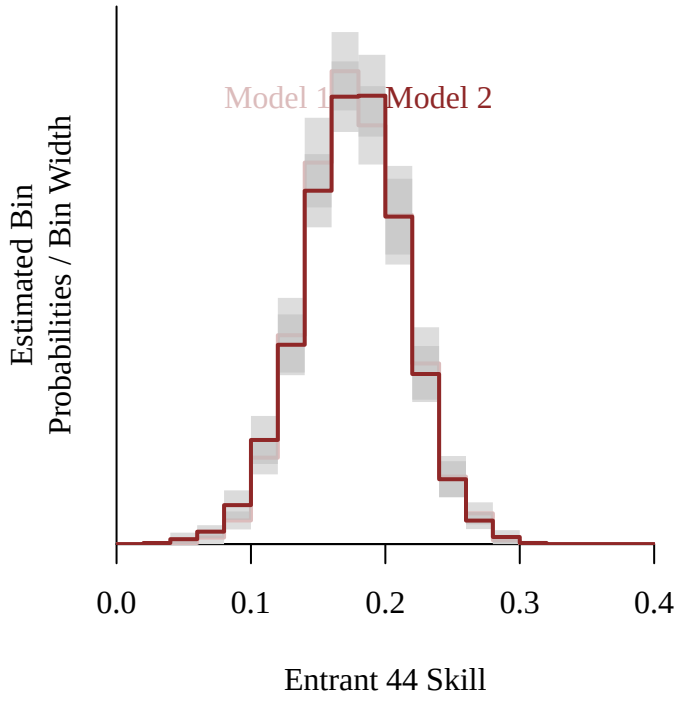
```
e <- 44
summarize_entrant(e)
```

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

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

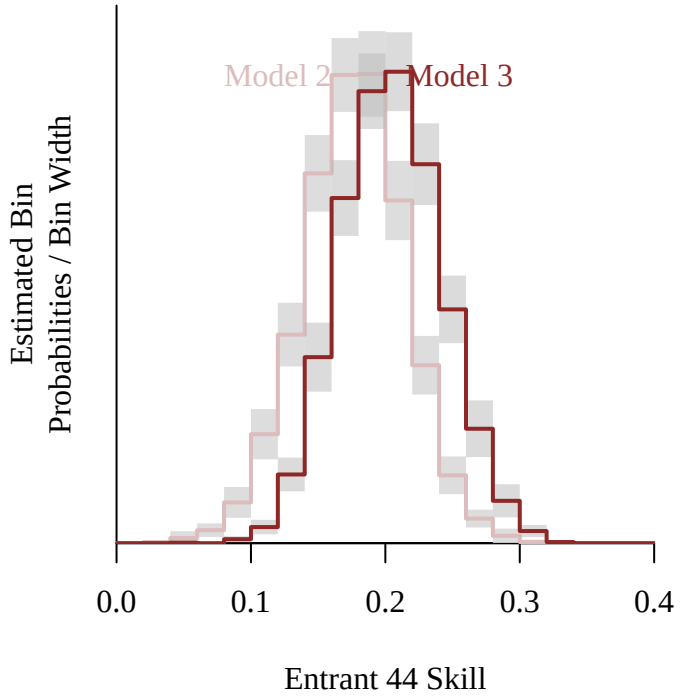
Again the tweak of the observational model has a negligible impact on the marginal posterior inferences.

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

util$plot_expectand_pushforward(samples1[[name]], 20,
                                flim=c(0, 0.4),
                                display_name=xname,
                                col=util$c_light)
text(0.12, 10, "Model 1", col=util$c_light)

util$plot_expectand_pushforward(samples2[[name]], 20,
                                flim=c(0, 0.4),
                                border="#BBBBBB88", add=TRUE)
text(0.24, 10, "Model 2", col=util$c_dark)
```



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

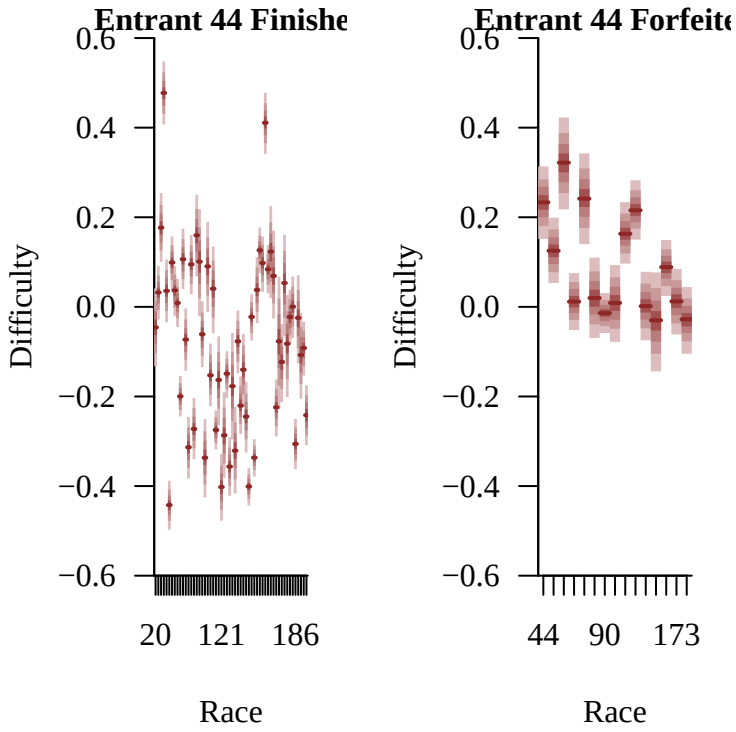
```
par(mfrow=c(1, 1), mar=c(5, 5, 1, 1))

util$plot_expectand_pushforward(samples2[[name]], 20,
                                flim=c(0, 0.4),
                                display_name=xname,
                                col=util$c_light)
text(0.12, 10, "Model 2", col=util$c_light)

util$plot_expectand_pushforward(samples3[[name]], 20,
                                flim=c(0, 0.4),
                                border="#BBBBBB88", add=TRUE)
text(0.255, 10, "Model 3", col=util$c_dark)
```



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

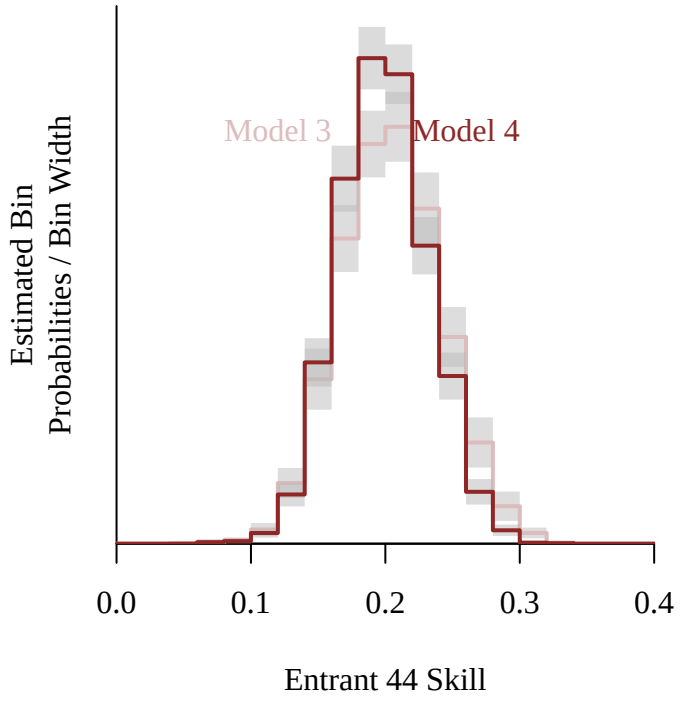
```
f_races <- c()
for (r in 1:data$N_races) {
  idxs <- data$race_f_start_idxs[r]:data$race_f_end_idxs[r]
  if (e %in% data$race_entrant_f_idxs[idxs])
    f_races <- c(f_races, r)
```

```
}

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('difficulties[', r,']'))
util$plot_disc_pushforward_quantiles(samples4, names,
                                     xlab="Race",
                                     xticklabs=f_races,
                                     ylab="Difficulty",
                                     display_ylim=c(-0.6, 0.6),
                                     main="Entrant 44 Finished")

names <- sapply(dnf_races, function(r) paste0('difficulties[', r,']'))
util$plot_disc_pushforward_quantiles(samples4, names,
                                     xlab="Race",
                                     xticklabs=dnf_races,
                                     ylab="Difficulty",
                                     display_ylim=c(-0.6, 0.6),
                                     main="Entrant 44 Forfeited")
```



Because the entrant 44 skill parameter concentrates on smaller values than the entrant 29 skill parameter the influence of the hierarchical coupling isn't as pronounced. Here only a small slice of larger values are suppressed and the marginal posterior distribution tightens only slightly.

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

util$plot_expectand_pushforward(samples3[[name]], 20,
                                flim=c(0, 0.4),
                                ylim=c(0, 13),
                                display_name=xname,
                                col=util$c_light)
text(0.12, 10, "Model 3", col=util$c_light)

util$plot_expectand_pushforward(samples4[[name]], 20,
                                flim=c(0, 0.4),
                                border="#BBBBBB88", add=TRUE)
text(0.26, 10, "Model 4", col=util$c_dark)
```



4.5.4 Entrant 83 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.

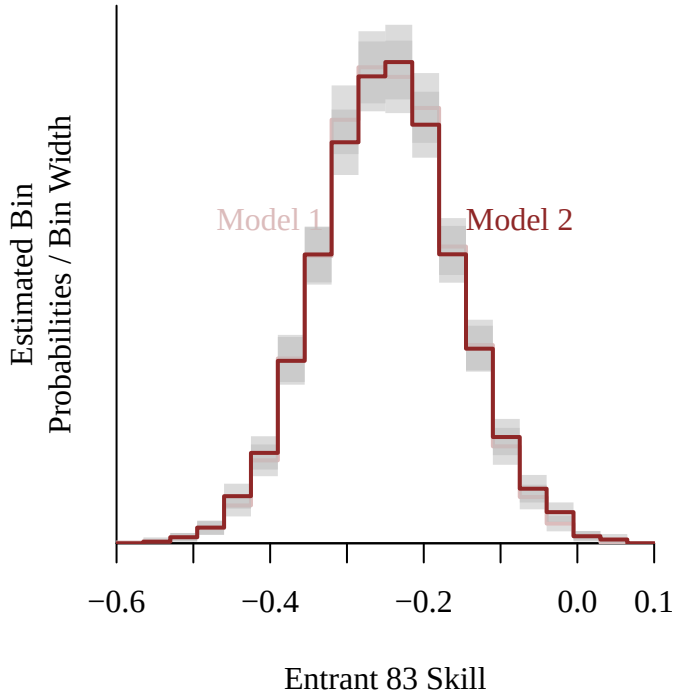
```
e <- 83
summarize_entrant(e)
```

```
Entrant 83
  5 total entrances
  3 finishes (60.0%)
  2 forfeits (40.0%)
```

```
name <- paste0('skills[', e,']')
xname <- paste0('Entrant ', e, ' 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$plot_expectand_pushforward(samples1[[name]], 20,
                                flim=c(-0.6, 0.1),
                                display_name=xname,
                                col=util$c_light)
text(-0.4, 3, "Model 1", col=util$c_light)

util$plot_expectand_pushforward(samples2[[name]], 20,
                                flim=c(-0.6, 0.1),
                                border="#BBBBBB88", add=TRUE)
text(-0.075, 3, "Model 2", col=util$c_dark)
```



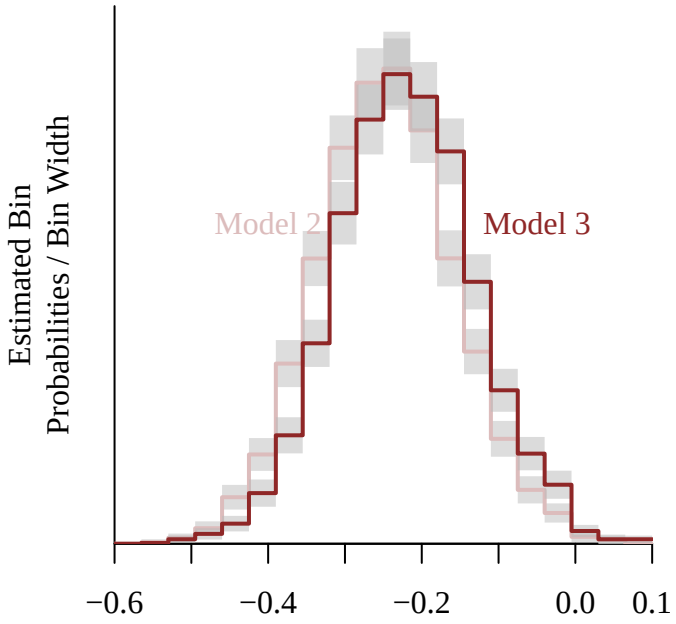
Incorporating forfeits pushes the skill marginal posterior distribution to larger values, but only slightly.

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

util$plot_expectand_pushforward(samples2[[name]], 20,
                                flim=c(-0.6, 0.1),
                                display_name=xname,
                                col=util$c_light)
```

```
text(-0.4, 3, "Model 2", col=util$c_light)

util$plot_expectand_pushforward(samples3[[name]], 20,
                                flim=c(-0.6, 0.1),
                                border="#BBBBBB88", add=TRUE)
```

```
Warning in util$plot_expectand_pushforward(samples3[[name]], 20, flim = c(-0.6,
: 1 samples (0.0%) fell above the histogram binning.
```

```
text(-0.05, 3, "Model 3", col=util$c_dark)
```



It's hard to say if the hierarchical coupling has any substantial impact. There is perhaps a very weak suppression of more negative skill values, but that trend is also within the span of the Markov chain Monte Carlo estimator errors.

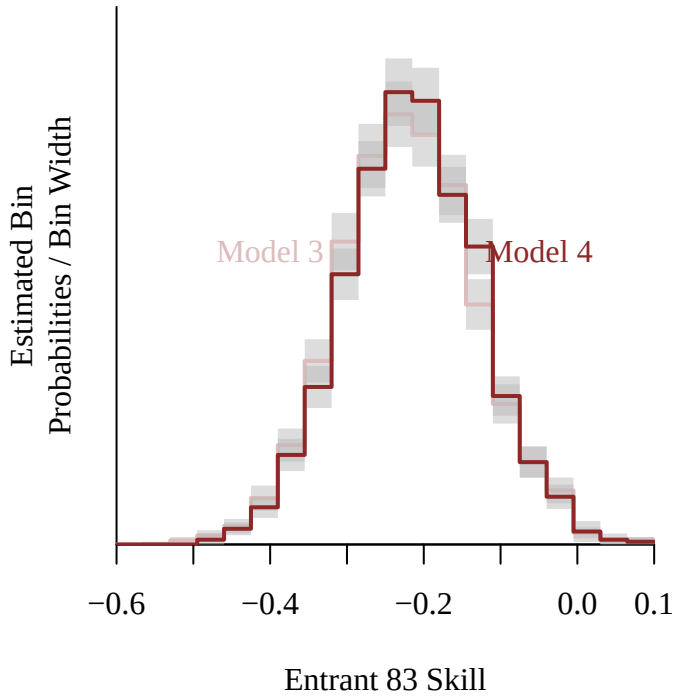
```
par(mfrow=c(1, 1), mar=c(5, 5, 1, 1))

util$plot_expectand_pushforward(samples3[[name]], 20,
                                flim=c(-0.6, 0.1),
                                ylim=c(0, 5.5),
                                display_name=xname,
                                col=util$c_light)
```

```
Warning in util$plot_expectand_pushforward(samples3[[name]], 20, flim = c(-0.6,
: 1 samples (0.0%) fell above the histogram binning.
```

```
text(-0.4, 3, "Model 3", col=util$c_light)

util$plot_expectand_pushforward(samples4[[name]], 20,
                                flim=c(-0.6, 0.1),
                                border="#BBBBBB88", add=TRUE)
```

```
Warning in util$plot_expectand_pushforward(samples4[[name]], 20, flim = c(-0.6,
: 2 samples (0.0%) fell above the histogram binning.
```

```
text(-0.05, 3, "Model 4", col=util$c_dark)
```



In hindsight we shouldn't expect any substantial impact from the introduction of the hierarchical modeling components given that the initial posterior inferences for the entrant 83 skill parameter already concentrated with the inferred span of the hierarchical population model.

4.6 Possible Model Expansions

At this point our model appears to be adequate, at least within the scope of the summary statistics that we considered. In particular because we only spot checked the retrodictive

behavior for 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 wouldn'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 Transcending 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 more sophisticated hierarchical population behavior beyond the normal population model that we have assumed. For example we could consider a Student's t population model where we have to infer not only the breadth of the population but also the precise shape of the population tails.

This more flexible hierarchical population model would allow our inferences to better accommodate sparsity, strongly regularizing most of the seed difficulties or entrant skills towards zero while allowing the more extreme behaviors to be more weakly regularized.

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 learning 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 or even 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 more complicated behaviors such as 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 correct path first will tend to finish especially fast while those who explore the incorrect paths will tend to finish later, resulting in especially large variability. This is especially true if a seed allows for unintended sequence breaks of which only the more skilled entrants can take advantage. On the other hand some seeds result in progression paths that are easier to predict which narrows the range of possible finish times.

One way to account for this heterogeneity is to allow ψ to vary across 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 can identify any consequences of this behavior in new posterior retrodictive checks. For example we could 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 and even 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 last model we don't want to forget 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 performance 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

Our posterior distribution can also be used to rank the entrants by their inferred skills.

Mathematically any configuration of entrant skills

$$(\lambda_{\text{skill},1}, \ldots, \lambda_{\text{skill},e}, \ldots, \lambda_{\text{skill},N_{\text{entrants}}}) \in (\Lambda_{\text{skill}})^{N_{\text{entrants}}}$$

implies a unique ranking

$$(r_1, \ldots, r_e, \ldots, r_{N_{\text{entrants}}}) \in R$$

where entrants are ordered by their their individual skills,

$$\lambda_{\text{skill},r_1} > \ldots > \lambda_{\text{skill},r_e} > \ldots > \lambda_{\text{skill},r_{N_{\text{entrants}}}}.$$

This in fact defines a bijective function from the space of entrant skills to the space R of the $N_{\text{entrants}}!$ possible orderings of the N_{entrants} entrants,

$$o : (\Lambda_{\text{skill}})^{N_{\text{entrants}}} \to R.$$

Pushing forward our posterior distribution along this function gives a posterior distribution $o_*\pi$ that quantifies our uncertainty about the possible entrant rankings. The only challenge is that the space of ranking R is massive and difficult to navigate. In particular it's not immediately clear how we can construct practical point summaries of this rank posterior distribution for applications like leader boards.

For example intuitively we might be interested in point estimates that quantify the centrality of the rank posterior distribution in some way. Because the space of rankings is discrete each rank will in general be allocated non-zero probability and we might consider a modal ranking,

$$r^* = \underset{r \in R}{\text{argmax}}\, o_*\pi(\{r\}).$$

Unfortunately even if a unique mode exists actually finding it will typically be intractable. Not only can we not compute the atomic allocations $o_*\pi(\{r\})$ in closed form but also exhaustively searching through all $N_{\text{entrants}}!$ elements will almost always be too expensive and we have no gradient information to guide a more efficient search.

In analogy to a posterior mean we might consider a distance function $d : R \times R \to \mathbb{R}^+$ and then define a point summary that minimizes the expected distance,

$$\mu_R = \underset{r \in R}{\text{argmin}}\, \mathbb{E}_{o_*\pi}[d(\cdot, r)].$$

Conveniently there are a variety of useful distance functions on R that we could use here. and the expectation values can be readily estimated with Markov chain Monte Carlo. The minimization over all possible candidate rankings $r \in R$, however, suffers from the same problems as above.

Because of these computational issues we will usually need to appeal to more heuristic methods in practice. For instance we can always rank the entrants by the posterior expectation values of their individual skills.

```
expected_skill <- function(e) {
  util$ensemble_mcmc_est(samples4[[paste0('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
...
```

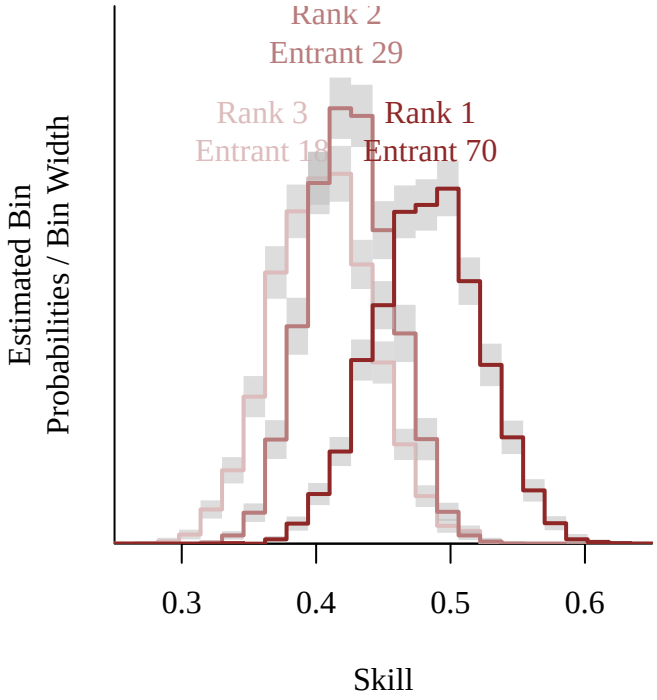
Interestingly the top eight entrants in this heuristic ranking are the same as the top eight entrants in the official `https://racetime.gg/smr` leader boards. That said the ordering of positions two through five are different.

```
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 is not at all surprising given our posterior uncertainties. 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.

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

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

util$plot_expectand_pushforward(samples4[[name]], 25,
                                flim=c(0.25, 0.65),
                                ylim=c(0, 15),
                                display_name="Skill",
                                col=util$c_light)
text(0.36, 11.5, paste("Rank 3\nEntrant", e), col=util$c_light)

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

util$plot_expectand_pushforward(samples4[[name]], 25,
                                flim=c(0.25, 0.65),
                                col=util$c_mid,
                                border="#BBBBBB88",
                                add=TRUE)
text(0.415, 14.25, paste("Rank 2\nEntrant", e), col=util$c_mid)

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

util$plot_expectand_pushforward(samples4[[name]], 25,
                                flim=c(0.25, 0.65),
                                col=util$c_dark,
                                border="#BBBBBB88",
                                add=TRUE)
text(0.485, 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. In particular this latter comparison accounts for inferential coupling between the two skill parameters.

```
C <- 4
S <- 1024

e1 <- ranked_entrants[data$N_entrants + 1 - 1]
e2 <- ranked_entrants[data$N_entrants + 1 - 2]
derived_samples <- matrix(NA, nrow=C, ncol=S)

for (c in 1:C) {
  for (s in 1:S) {
    derived_samples[c, s] <- samples4[[paste0('skills[', e1,']')]][c, s] >
                             samples4[[paste0('skills[', e2,']')]][c, s]
  }
}

p <- util$ensemble_mcmc_est(derived_samples)[1]

format_string <- paste0("Probability that entrant %i skill is ",
                        "larger than entrant %i skill = %.3f.")
cat(sprintf(format_string, e1, e2, p))
```

```
Probability that entrant 70 skill is larger than entrant 29 skill = 0.941.
```

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.

```
best_entrant <- function(entrant_idxs) {
  C <- 4
  S <- 1024

  E <- length(entrant_idxs)
  probs <- c()
  skills <- array(NA, dim = c(data$N_entrants, C, S))
  for (e in entrant_idxs)
    skills[e,,] <- samples4[[paste0('skills[', e,']')]]

  for (e in entrant_idxs) {
    other_entrant_idxs <- entrant_idxs[-which(entrant_idxs == e)]
    comps <- matrix(NA, nrow=C, ncol=S)
    for (c in 1:C) {
      for (s in 1:S) {
        pairwise_comps <- sapply(other_entrant_idxs,
                                 function(eo) skills[e] > skills[eo])
        comps[c, s] <- Reduce("&", pairwise_comps)
      }
    }
    probs <- c(probs, util$ensemble_mcmc_est(comps)[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 give the same top three as the ranking of entrants by their posterior expected skills. In general, however, this will not always be the case.

The practical limitations of this 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 existing entrants in a hypothetical race using a new seed in the latest MapRando version.

```
fit <- stan(file="stan_programs/model5.stan",
            data=data, seed=8438338,
            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.
```

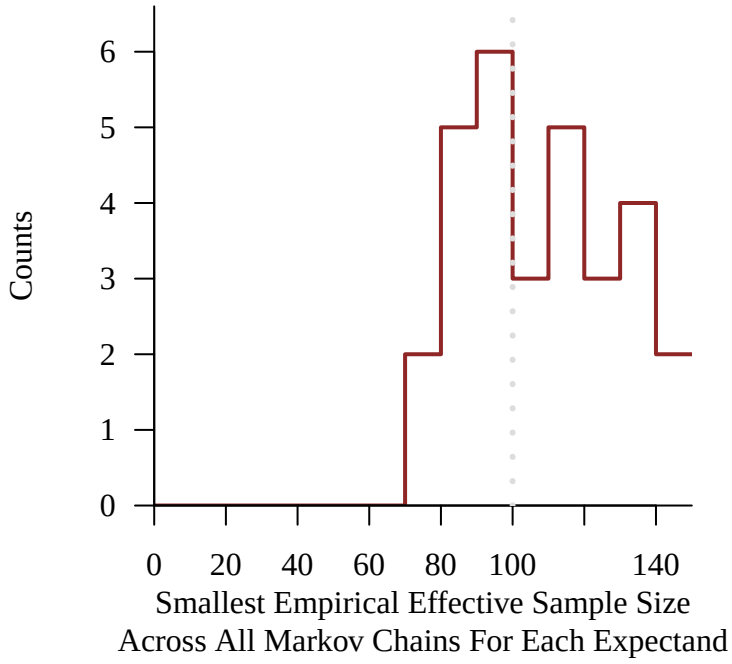
```
samples <- util$extract_expectands(fit)
base_samples <- util$filter_expectands(samples,
                                       c('gamma',
                                         'eta_difficulties',
                                         'tau_difficulties',
                                         'eta_skills',
                                         'tau_skills',
                                         'kappas', 'betas',
                                         'psi'),
                                       check_arrays=TRUE)
util$summarize_expectand_diagnostics(base_samples)
```

```
The expectands gamma, eta_skills[4], eta_skills[18], eta_skills[29],
eta_skills[44], eta_skills[58], eta_skills[60], eta_skills[65],
eta_skills[90], eta_skills[91], eta_skills[100], eta_skills[105],
kappas[6], kappas[44], kappas[94], kappas[98] triggered diagnostic
warnings.

The expectands gamma, eta_skills[4], eta_skills[18], eta_skills[29],
eta_skills[44], eta_skills[58], eta_skills[60], eta_skills[65],
eta_skills[90], eta_skills[91], eta_skills[100], eta_skills[105],
kappas[94] triggered hat{ESS} warnings.

Small empirical effective sample sizes indicate strong empirical
autocorrelations in the realized Markov chains. If the empirical
effective sample size is too small then Markov chain Monte Carlo
estimation may be unreliable even when a central limit theorem holds.
```

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

min_eesss <- util$compute_min_eesss(base_samples)
util$plot_line_hist(min_eesss, 0, 150, 10, col=util$c_dark,
                    xlab=paste0("Smallest Empirical Effective Sample Size\n",
                                "Across All Markov Chains For Each Expectand"))
```

```
Warning in check_bin_containment(bin_min, bin_max, values): 492 values (94.3%)
fell below the binning.
```

```
abline(v=100, col="#DDDDDD", lty=3, lwd=3)
```



The uses of these predictions are endless.

5.2.1 Single Entrant Predictions

For example some entrants not only live-stream their entrances to their communities but also allow viewers to make non-monetary over/under bets on their finish time. 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 condition

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

In other words t_{gamble} would just 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 completely fair, 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 = 0.018
```

Because the forfeit probability 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.

```
q <- 0

name <- paste0('entrant_f_times_pred[', e, ']')
for (c in 1:C) {
  q <- q + quantile(samples[[name]][c,], probs=c(p_balanced)) / C
}

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

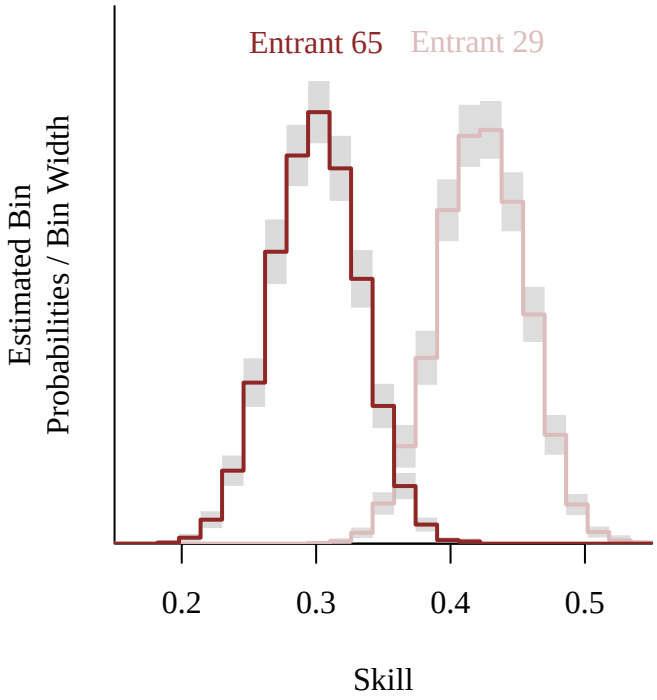
```
t_gamble = 66.923 minutes
```

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' skill parameters overlap quite a bit, but that of entrant 29 does favor larger values than that of entrant 65.

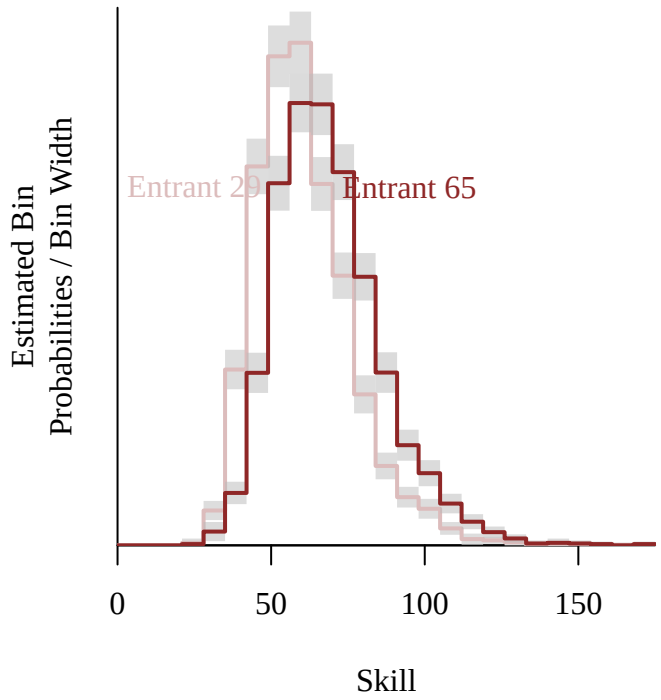
```
par(mfrow=c(1, 1), mar=c(5, 5, 1, 1))

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

util$plot_expectand_pushforward(samples[[name]], 25,
                                flim=c(0.15, 0.55),
                                ylim=c(0, 15),
                                display_name="Skill",
                                col=util$c_light)
text(0.42, 14, paste("Entrant", e), col=util$c_light)

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

util$plot_expectand_pushforward(samples[[name]], 25,
                                flim=c(0.15, 0.55),
                                col=util$c_dark,
                                border="#BBBBBB88",
                                add=TRUE)
text(0.3, 14, 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 taking into account 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, "]")

C <- 4
S <- 1024

derived_samples <- lapply(1:3, function(k) matrix(NA, nrow=C, ncol=S))
names(derived_samples) <- c('forfeit_win', 'no_forfeits', 'neg_time_diff')

for (c in 1:C) {
  for (s in 1:S) {
    status1 <- samples[[status_name1]][c, s]
    status2 <- samples[[status_name1]][c, s]

    derived_samples[['forfeit_win']][c, s] <- (status1 == 0 & status2 == 1)
    derived_samples[['no_forfeits']][c, s] <- (status1 == 0 & status2 == 0)

    derived_samples[['neg_time_diff']][c, s] <- samples[[time_name1]][c, s] <
                                                samples[[time_name2]][c, s]
  }
}

p_forfeit_win <- util$ensemble_mcmc_est(derived_samples[['forfeit_win']])[1]
p_no_forfeits <- util$ensemble_mcmc_est(derived_samples[['no_forfeits']])[1]
p_neg_time_diff <- util$ensemble_mcmc_est(derived_samples[['neg_time_diff']])[1]

p <- p_forfeit_win + p_neg_time_diff * p_no_forfeits

cat(sprintf("Probability that entrant %i beats entrant %i = %.3f",
            e1, e2, p))
```

```
Probability that entrant 29 beats entrant 65 = 0.730
```

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 obscure 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 celebrate the capabilities of open source projects along the communities they inspire along the way 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;
}

parameters {
  real gamma;                       // Log baseline finish time (log seconds)
  array[N_races] real difficulties; // Seed difficulties
  array[N_entrants] real skills;    // Entrant skills
  real<lower=0> psi;                // Gamma dispersion configuration
}

model {
  // Prior model
  gamma ~ normal(8.045, 0.237);    // log(1800 s) < gamma < log(5400 s)
  difficulties ~ normal(0, 0.299); // -log(2) <~ difficulties <~ +log(2)
  skills ~ normal(0, 0.299);       // -log(2) <~    skills    <~ +log(2)
  psi ~ normal(0, 0.389);          // 0 <~ psi <~ 1

  // Observational model
  for (r in 1:N_races) {
    // Extract details for entrants who finished
```

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;
}

parameters {
  real gamma;                       // Log baseline finish time (log seconds)
  array[N_races] real difficulties; // Seed difficulties
  array[N_entrants] real skills;    // Entrant skills
  real<lower=0> psi;                // Inverse gamma dispersion configuration
}

model {
  // Prior model
  gamma ~ normal(8.045, 0.237);    // log(1800 s) < gamma < log(5400 s)
  difficulties ~ normal(0, 0.299); // -log(2) <~ difficulties <~ +log(2)
  skills ~ normal(0, 0.299);       // -log(2) <~    skills    <~ +log(2)
  psi ~ normal(0, 0.389);          // 0 <~ psi <~ 1

  // Observational model
  for (r in 1:N_races) {
    // Extract details for entrants who finished
```

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;
}

parameters {
  real gamma;                       // Log baseline finish time (log seconds)
  array[N_races] real difficulties; // Seed difficulties
  array[N_entrants] real skills;    // Entrant skills
```

Stan
Program 4 `model4.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 did not finish
  array[N_races] int<lower=0, upper=N_entrants> race_N_entrants_dnf;

  // Indices for extracting did not finish 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;
  array[N_entrances_fs] int race_entrant_f_idxs;   // Entrant index
  array[N_entrances_fs] real race_entrant_f_times; // Entrant finish times

  // Total number of forfeits across all races
  int<lower=0> N_entrances_dnfs;
  array[N_entrances_dnfs] int race_entrant_dnf_idxs; // Entrant index

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



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

  vector[N_races] eta_difficulties; // Non-centered seed difficulties
  vector<lower=0>[N_versions] tau_difficulties; // Seed difficulty population scale
```

Stan
Program 5 `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 did not finish
  array[N_races] int<lower=0, upper=N_entrants> race_N_entrants_dnf;

  // Indices for extracting did not finish 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;
  array[N_entrances_fs] int race_entrant_f_idxs;   // Entrant index
  array[N_entrances_fs] real race_entrant_f_times; // Entrant finish times

  // Total number of forfeits across all races
  int<lower=0> N_entrances_dnfs;
  array[N_entrances_dnfs] int race_entrant_dnf_idxs; // Entrant index

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



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

  vector[N_races] eta_difficulties; // Non-centered seed difficulties
  vector<lower=0>[N_versions] tau_difficulties; // Seed difficulty population scale
```