Die Another Bayes

Michael Betancourt

April 2024

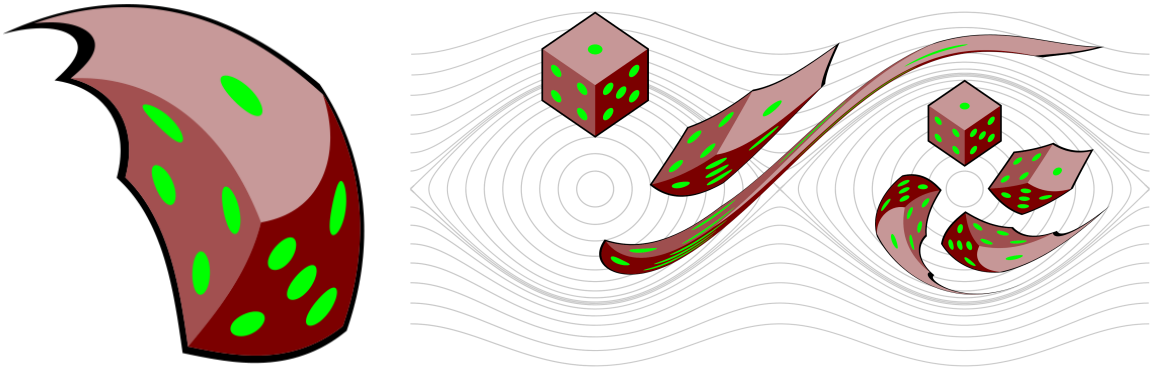
Table of contents

As a freelancer branding is a necessary evil. The logo for my own company (Figure 1a) is derived from applying a symplectomorphic transformation to the image of a six-sided die (Figure 1b), emulating how Hamiltonian Monte Carlo transforms probabilistic objects. A few years after establishing my company I came across the Dice Lab and their Skew Dice™. While the shape of these dice doesn't exactly match my logo, they were close enough for effective branding and I was able to place a bulk order for Skew Dice™ in my logo colors (Figure 2).

At a workshop in January 2023 I was giving away some dice to participants when one Will Pierce has the *audacity* to question the integrity of the Skew Dice™. Once I had calmed down I realized that this challenge raised an interesting question: how exactly can we learn how fair a die is in practice?

The Dice Lab website provides a geometric argument for why the Skew Dice™ should roll fairly:

> Skew Dice™ are based on the trigonal trapezohedron. (A cube is a special case of a trigonal trapezohedron.) We distorted the faces in such a way that they remained flat and are congruent (each one has the same size and shape). In addition, the distorted polyhedron is an isohedron, meaning that the symmetry group of the polyhedron is transitive on the faces. Despite their bizarre appearance, this means that Skew Dice are just as fair as regular dice.



(a) (b)

Figure 1: (a) The logo for my consulting and training company, Symplectomorphic, is a warped die (b) that was generated by transforming an imagine of a square die with a symplectomorphic transformation.

Figure 2: Because we live in the future one can actually exchange money for warped six-sided dice in custom colors. Although the shape of these dice doesn't exactly match the warping of a symplectomorphism it's about as close as one can get without contracting a factory directly.

This argument, however, assumes that each die is manufactured precisely and with uniform material density. Without being able to test these mechanical properties directly all we can do is roll the dice over and over again and investigate how often each side lands facing upwards.

In this case study I present a series of Bayesian analyses that attempt to infer just how fair the Skew Dice™ that I ordered might be using hundreds of physical die rolls performed by a small group of volunteers. Along the way I'll present various techniques for working with categorical data and simplex model configurations, including some approaches for incorporation heterogeneity.

1 Data Exploration

To ensure as rich of a data set as possible I sent Skew Dice™ to Adriano Yoshino, EM Wolkovich, Joe Wozny, Karim Naguib, Ian Costley, and John Flournoy, six of my generous Patreon supporters who volunteered their own, and in some cases their friends' and family's, time. Each volunteer then collected data from repeated rolls, recording which die was thrown, who threw it, and what the outcome was. Once all of the data had been shared I collected it into a single `R` data frame with unique indices for each die and player.

```
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, 5))
```

```
df <- read.csv("data/rolls.csv")


N <- nrow(df)
N_players <- length(unique(df$player_idxs))
N_dice <- length(unique(df$die_idxs))

data <- list("N" = N,
             "outcome" = df$outcome,
             "N_players" = N_players,
             "player_idxs" = df$player_idxs,
             "N_dice" = N_dice,
             "die_idxs" = df$die_idxs)
```

Altogether 10 unique individuals used a combination of 12 unique dice to give a 1682 total rolls.

```
print(sprintf("%s total rolls", N))
```

```
[1] "1682 total rolls"
```

```
print(sprintf("%s distinct players", N_players))
```

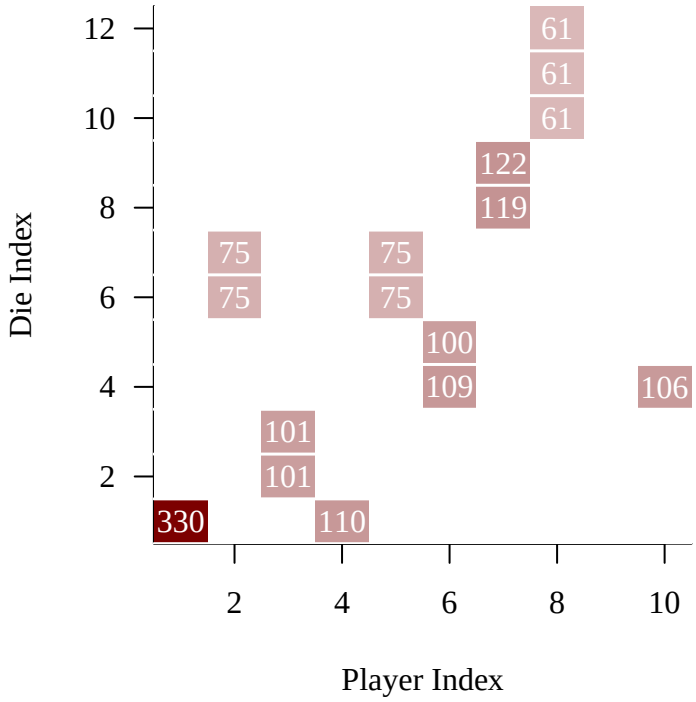
```
[1] "10 distinct players"
```

```
print(sprintf("%s distinct dice", N_dice))
```

```
[1] "12 distinct dice"
```

Because each individual had access to only a few dice, however, the data include only a few player-dice pairings.

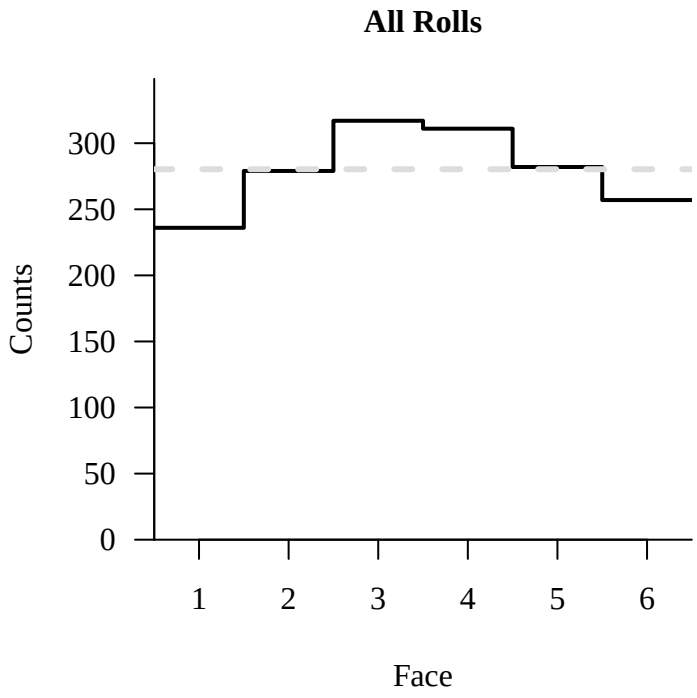
```
par(mfrow=c(1, 1), mar=c(5, 5, 1, 1))

library(colormap)
disc_colors <- c("#FFFFFF", "#DCBCBC", "#C79999", "#B97C7C",
                 "#A25050", "#8F2727", "#7C0000")
cont_colors <- colormap(colormap=disc_colors, nshades=100)

zs <- table(data$player_idxs, data$die_idxs)

image(1:N_players, 1:N_dice, zs,
      col=rev(cont_colors),
      xlab="Player Index", ylab="Die Index")

for (d in 1:N_dice) abline(h=d - 0.5, lwd=1.5, col="white")
for (p in 1:N_players) for (d in 1:N_dice) text(p, d, zs[p, d], col="white")
```



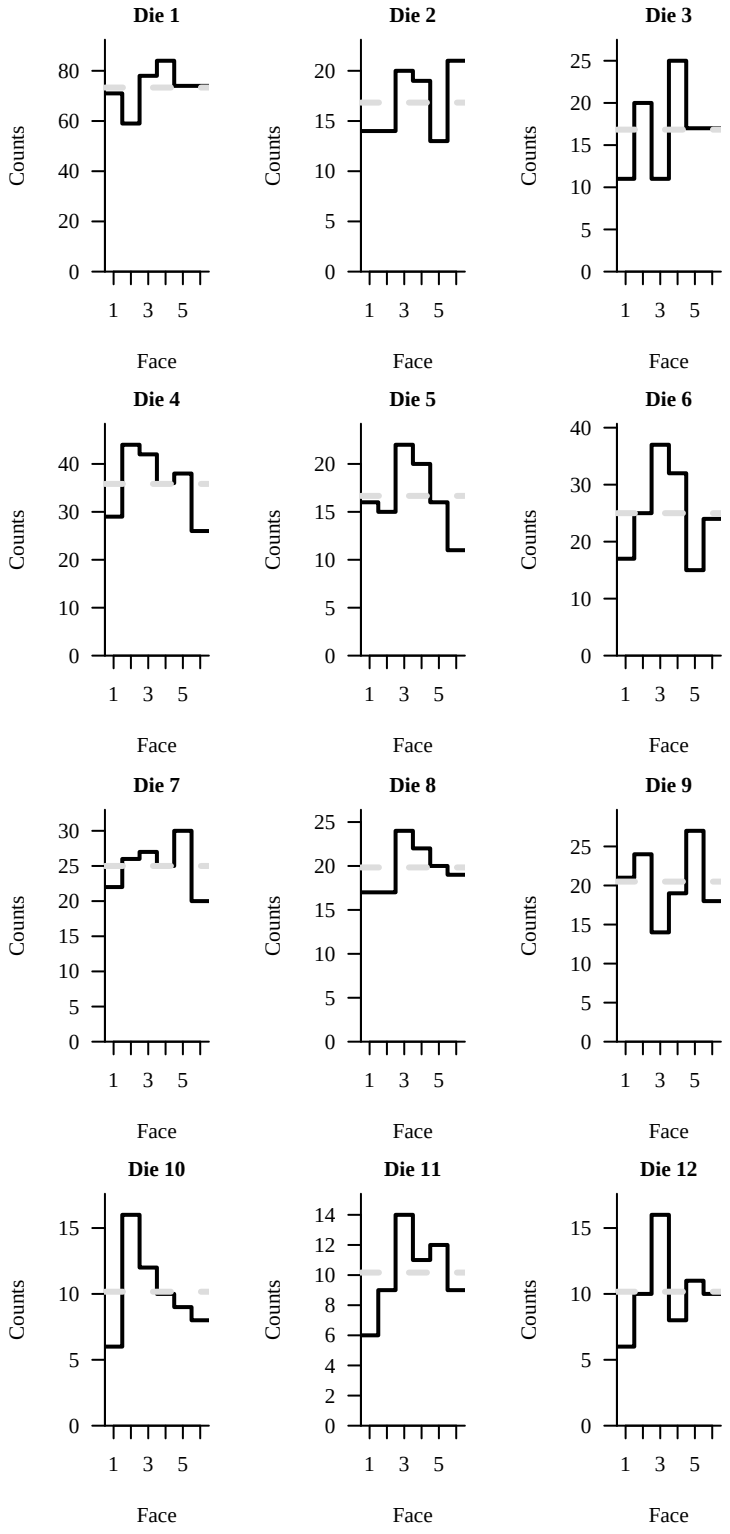
The rolls themselves appear to be reasonably uniform across the six faces, both in aggregate and individually for each die.

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

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

util$plot_line_hist(data$outcome, 0.5, 6.5, 1,
                    xlab="Face", main="All Rolls")
abline(h=data$N / 6, lty=2, lwd=3, col="#DDDDDD")
```




```
for (d in 1:data$N_dice) {
  if (d %% 6 == 1) par(mfrow=c(2, 3), mar=c(4, 5, 2, 1))

  util$plot_line_hist(data$outcome[data$die_idxs == d],
                      0.5, 6.5, 1, xlab="Face", main=paste("Die", d))
  abline(h=sum(data$die_idxs == d) / 6, lty=2, lwd=3, col="#DDDDDD")
}
```


Die 1
Die 2
Die 3
Die 4
Die 5
Die 6
Die 7
Die 8
Die 9
Die 10
Die 11
Die 12
Counts
Face
1
3
5

That said the recorded outcomes are not *exactly* uniform. Our statistical challenge is to determine just how large the deviations from exact uniformity need to be before we need to be concerned about unfair dice.

2 Uniform Simplex Model

If we assume that the outcome of rolling a die is unpredictable then we won't expect exactly uniform outcomes, even when a die is fair. In order to quantify these deviations we need to build a predictive model.

2.1 The Observational Model

Each roll of a die on a flat surface results in one, and only one, face. If we label each face with one of six integers then the observational space becomes

$$Y = \{1, 2, 3, 4, 5, 6\}.$$

In general we can model the relative frequencies of each these six outcomes with a finite probability distribution specified by six atomic probabilities,

$$\begin{aligned} \pi(\{1\}) &= q_1 \\ &\ldots \\ \pi(\{k\}) &= q_k \\ &\ldots \\ \pi(\{6\}) &= q_6 \end{aligned}$$

satisfying the constraints

$$0 \leq q_k \leq 1$$

and

$$\sum_{k=1}^{6} q_k = 1.$$

The observational model for a single outcome is then given by categorical distribution,

$$\begin{aligned} p(y \mid q) &= \text{categorical}(y \mid q_1, \ldots, q_k, \ldots, q_6) \\ &\equiv q_{k=y}. \end{aligned}$$

The space of all such probability distributions over six elements is known as the 5-**simplex** and is denoted Δ^5. Note that we refer to this space as 5-simplex and not a 6-simplex because the

normalization constraint effectively remove one degree of freedom. I will refer to any individual point in this simplex,

$$q = (q_1, \ldots, q_k, \ldots, q_6) \in \Delta^5,$$

as a simplex configuration.

If the properties of a particular die do not change appreciably between rolls then we should be able to model the outcome of multiple rolls using the same simplex configuration. In this case the joint observational model for multiple rolls reduces to a multinomial distribution,

$$\begin{aligned} p(y_1, \ldots, y_N \mid q_1, \ldots, q_6) &= \prod_{n=1}^{N} \text{categorical}(y_n \mid q_1, \ldots, q_6) \\ &= \prod_{n=1}^{N} q_{k=y_n} \\ &= \prod_{k=1}^{6} q_k^{\sum_{n=1}^{N} \delta_{y_n, k}} \\ &= \prod_{k=1}^{6} q_k^{n_k}, \end{aligned}$$

where $\delta_{y_n,k}$ is the Kronecker delta,

$$\delta_{y_n,k} = \begin{cases} 1, & y_n = k \\ 0, & y_n \neq k \end{cases},$$

and

$$n_k = \sum_{n=1}^{N} \delta_{y_n,k}$$

is the total number of rolls resulting in the kth face.

Finally if we assume that all of the dice behave identically then we can model all of their outcomes with a multinomial distribution and a single simplex configuration.

The context of this model provides a new way of interpreting "fairness". Instead of requiring that a fair die produce uniform outcomes we can require that a fair die be modeled by the uniform simplex configuration

$$\upsilon = \left(\frac{1}{6}, \frac{1}{6}, \frac{1}{6}, \frac{1}{6}, \frac{1}{6}, \frac{1}{6}, \right) \in \Delta^5.$$

Interestingly this strict modeling assumption leaves us with no degrees of freedom to infer from the data. Instead we can immediately move on to the simulation of predictive data.

2.2 Summary Statistics For Retrodictive Checks

The observational space for all $N = 1682$ rolls in our data set,

$$y_1, \ldots, y_n, \ldots, y_N,$$

is 1682 dimensional. Unfortunately there are just a few too many dimensions to be able to visualize a predictive distribution directly.

In order to construct productive retrodictive checks to critique this model we'll need summary statistics that can reduce the 1682-dimensional observational space into something a bit more manageable without sacrificing interpretability.

For example a histogram of the observed outcomes counts,

$$n_1, \ldots, n_6$$

is particularly natural for categorical data like this.

Moreover we can summarize the uniformity of these histograms using an empirical entropy function over the individual counts

$$\begin{aligned} \hat{H}(n_1, \ldots, n_6) &= -\sum_{k=1}^{6} \hat{q}_k \log \hat{q}_k \\ &= -\sum_{k=1}^{6} \left(\frac{n_k}{N}\right) \log \left(\frac{n_k}{N}\right). \end{aligned}$$

The empirical entropy function is maximized when the observed counts are exactly uniform,

$$n_1 = \ldots = n_6 = \frac{N}{6},$$

and minimized when we observe one and only one outcome k',

$$n_{k'} = N, n_{k \neq k'} = 0.$$

In a retrodictive check the empirical entropy allows us to investigate how well our model captures the uniformity, or lack thereof, exhibited by the observed rolls.

2.3 Implementation

Because there are no free parameters we can simulate predictive data directly in the `generated quantities` block of a Stan program.

The `Fixed_param` algorithm in `RStan` then allows us to repeatedly evaluate this `generated quantities` block to generate an ensemble of predictive summary statistic outputs.

Stan
Program 1 `uniform\_simplex.stan`

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

transformed data {
  // Uniform simplex configuration
  simplex[6] upsilon = rep_vector(1.0 / 6, 6);
}

generated quantities {
  // Compute predictions
  array[6] int pred_counts = multinomial_rng(upsilon, N);
  real pred_entropy = 0;
  for (k in 1:6) {
    real q_hat = pred_counts[k] * 1.0 / N;
    pred_entropy += - lmultiply(q_hat, q_hat);
  }
}
```

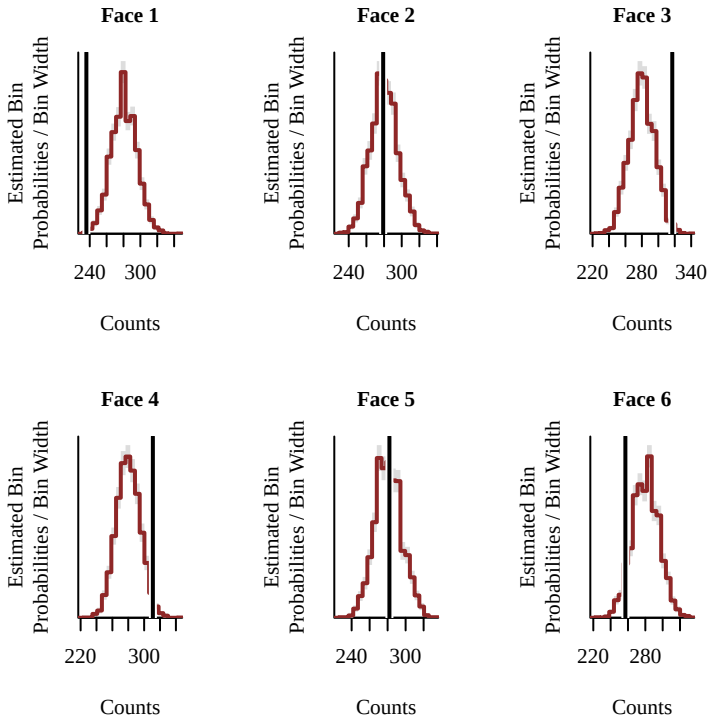
```
library(rstan)
rstan_options(auto_write = TRUE)            # Cache compiled Stan programs
options(mc.cores = parallel::detectCores()) # Parallelize chains
parallel:::setDefaultClusterOptions(setup_strategy = "sequential")

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

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

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

As we're not running Markov chain Monte Carlo here there are no computational diagnostics to consider and we can instead jump straight into the retrodictive checks.

Let's first consider the retrodictive behavior within each histogram bin, comparing each observed count to the corresponding predictive count.

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

obs_counts <- table(data$outcome)

for (k in 1:6) {
  name <- paste0('pred_counts[', k, ']')
  util$plot_expectand_pushforward(samples[[name]], 20,
                                  display_name="Counts",
                                  main=paste0("Face ", k),
                                  baseline=obs_counts[k])
}
```

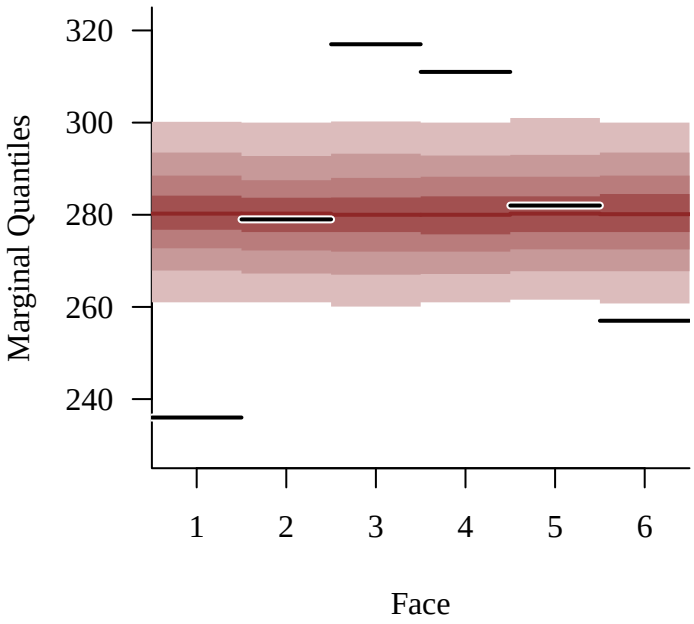


Overall there don't appear to be any serious retrodictive disagreements. That said there are a few outcomes for which the observed counts fall into the tails of the corresponding predictive distribution.

In order to investigate for any patterns in these mild retrodictive tensions it will help to visualizing the observed and predictive behaviors for all outcomes at the same time. One particularly convenient way to do this is with nested quantile intervals.

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

```
obs_counts <- table(data$outcome)

pred_names <- sapply(1:6, function(k) paste0('pred_counts[', k, ']'))
util$plot_disc_pushforward_quantiles(samples, pred_names,
                                     baseline_values=obs_counts,
                                     xlab="Face", display_ylim=c(225, 325))
```

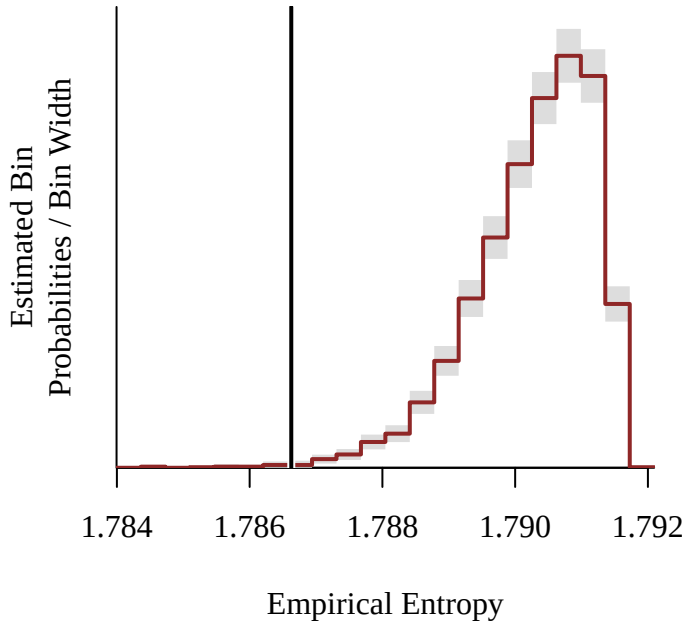


Immediately we see that the observed counts aren't completely contained within the predictive quantile intervals. That, however, is not necessarily surprising. For example the largest intervals shown here span only the 10% to 90% predictive quantiles; 10% predictive probability continues below the lightest band and 10% continues above it. At the same time because the marginal quantiles are not sensitive to correlations in the counts they tend to underestimate the joint variation.

The empirical entropy retrodictive check, which directly quantifies the deviation of the observed counts from exact uniformity, exhibits the same mild retrodictive tension seen in the retrodictive checks for each outcome. This at least qualitatively supports the adequacy of uniform simplex model, and the fairness of the dice.

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

obs_entropy <- sum(sapply(obs_counts,
                          function(n) -(n / data$N) * log(n / data$N)))

util$plot_expectand_pushforward(samples[["pred_entropy"]], 20,
```

```
                                    display_name="Empirical Entropy",
                                    baseline=obs_entropy)
```



2.4 Interpreting the Possible Retrodictive Tension

The shape of the mild retrodictive tension in the histogram summary statistic is useful for motivating possible model inadequacies, and hence productive model expansions. Here the tension suggests that the uniform simplex model might be underestimating the occurrence of faces 3 and 4 while overestimating the occurrence of faces 1 and 6.

One advantage of knowing the provenance of the data is that we can compare this behavior to our domain expertise. In particular we can consider whether or not this deviation from uniformity is physically reasonable.

The faces on each Skew Dice™ are arranged so that opposite faces always sum to 7: 1 is opposite 6, 2 is opposite 5, and 3 is opposite 4. A symmetric deficit of 1 and 6 rolls could be explained by less material on these opposite faces relative to faces 2, 3, 4, and 5. Similarly a symmetric excess of 3 and 4 rolls could be explained by an excess of material on those opposite faces.

That said not every source of non-uniformity is consistent with these patterns. For example biases caused by the asymmetric filling of a die mold from one side to the other would manifest in excesses and deficits that are similar between not *opposite* faces but rather *neighboring* faces. Without knowing more details about the manufacturing process, however, we can really only speculate.

3 General Simplex Model

One way to clarify the situation is to consider a more elaborate model and see if the added flexibility resolves any of the mild retrodictive tension exhibited by the simpler model. If we can't find any beneficial expansions then we can always return to the simpler model.

In this section we'll consider a model that takes advantage of the entire simplex Δ^5. We will continue to assume that the dice behave identically so that all of the data can be modeled with a single multinomial distribution.

3.1 Inferences

Conveniently modeling general simplex configurations is straightforward in Stan due to the `simplex` variable type which automatically enforces the simplex constraints. With only limited information about the manufacturing of the dice we'll use a uniform prior density function over the entire simplex.

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

The diagnostics don't suggest that Stan's Hamiltonian Monte Carlo sampler is encountering any problems in its efforts to quantifying the posterior distribution.

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

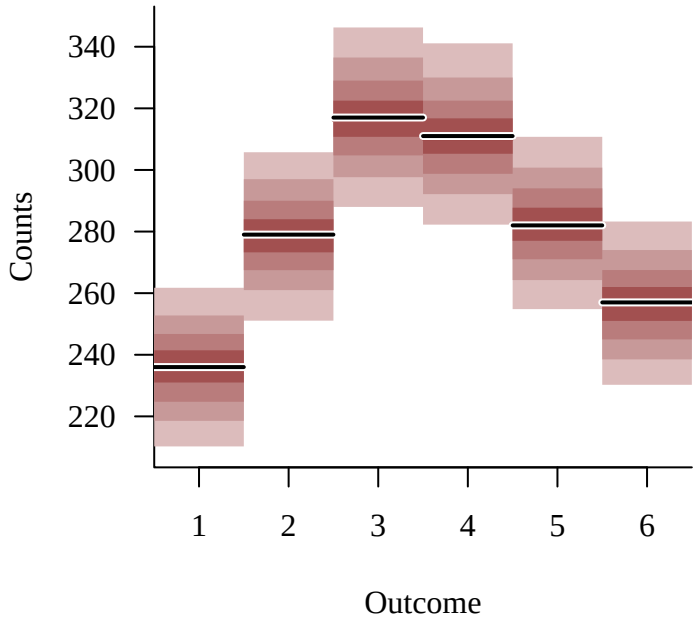
```
  All Hamiltonian Monte Carlo diagnostics are consistent with reliable
Markov chain Monte Carlo.
```

```
samples <- util$extract_expectands(fit)
base_samples <- util$filter_expectands(samples, 'q', check_arrays=TRUE)
util$check_all_expectand_diagnostics(base_samples)
```

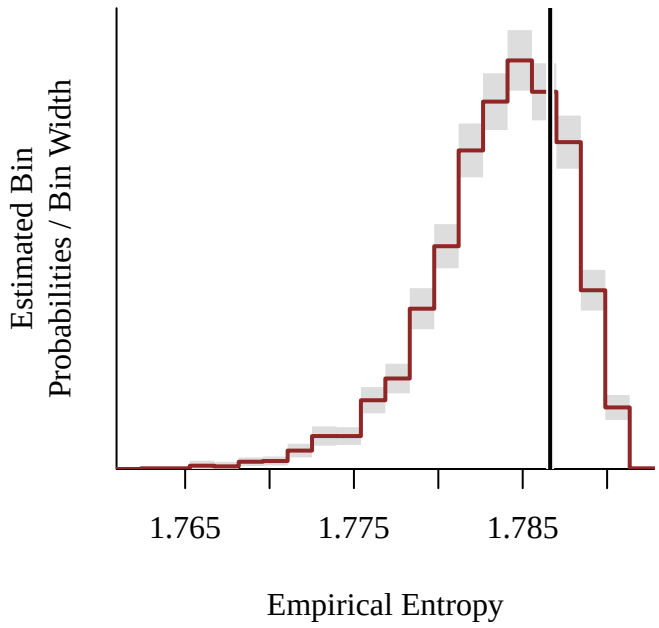
```
All expectands checked appear to be behaving well enough for reliable
Markov chain Monte Carlo estimation.
```

With an accurate posterior quantification we can faithfully compare the posterior predictive distribution to the observed data. The general simplex model appears to have more than enough flexibility to reproduce all of the behaviors in the observed data that manifest in the total outcome frequencies and empirical entropy.

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

pred_names <- sapply(1:6, function(k) paste0('pred_counts[', k, ']'))
util$plot_disc_pushforward_quantiles(samples, pred_names,
                                     baseline_values=obs_counts,
                                     xlab="Outcome", ylab="Counts")
```

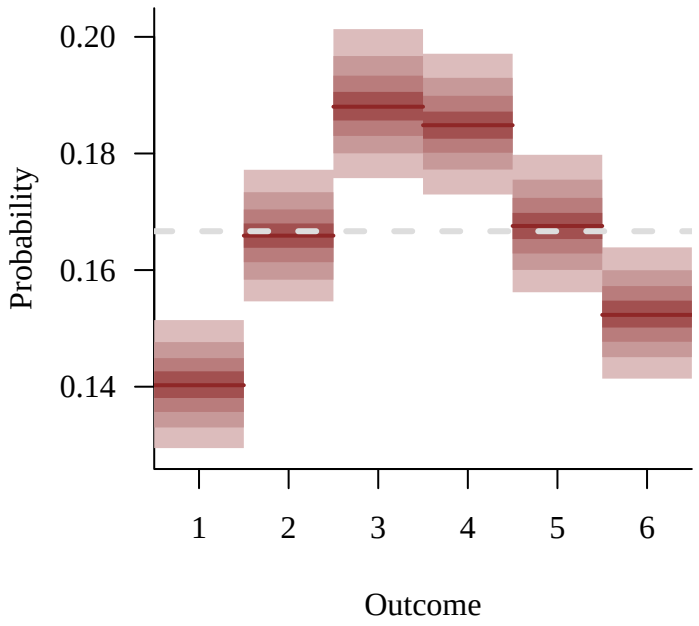



```
par(mfrow=c(1, 1))

obs_entropy <- sum(sapply(obs_counts,
                          function(n) -(n / data$N) * log(n / data$N)))

util$plot_expectand_pushforward(samples[["pred_entropy"]], 20,
                                display_name="Empirical Entropy",
                                baseline=obs_entropy)
```



Because there is no visible posterior retrodictive tension the posterior inferences from this model should capture meaningful insights. In this case the posterior distribution follows the pattern seen in the observed counts, with an excess of probability for faces 3 and 4 and a deficit for faces 1 and 6. That said the posterior distribution is not concentrating all that far from the uniform simplex configuration.

```
names <- sapply(1:6, function(k) paste0('q[', k, ']'))
util$plot_disc_pushforward_quantiles(samples, names,
                                     xlab="Outcome", ylab="Probability")
abline(h=1/6, col="#DDDDDD", lwd=3, lty=2)
```



3.2 Decisions

Inferences from the general simplex model provide multiple ways to characterize the fairness of the dice within the scope of the modeling assumptions. In particular we can use our inferences to inform explicit decisions about the fairness of the dice for each characterization. In this section we'll see how we can use Bayesian decision theory to construct systematic decisions.

3.2.1 A Primer on Bayesian Decision Theory

Formal decision making is defined by a space of actions, $a \in A$, and a utility function that numerically quantifies the overall benefits of taking each action,

$$\begin{aligned} U : A &\to \mathbb{R} \\ a &\mapsto U(a). \end{aligned}$$

Once we have defined a space of actions and a utility function then the optimal decision is simply to take the action with the highest utility,

$$a_{\text{optimal}} = \underset{a \in A}{\text{argmax}} \, \overline{U}(a).$$

Note that to simplify this discussion I am ignoring the possibility of ties, but they are not difficult to accommodate by first identifying the set of equally-optimal actions and then introducing a procedure to arbitrarily choose from those actions.

For many problems the utility of each action will depend on unknown behaviors. In these cases the best we can do is construct a model configuration space of possible behaviors Θ and then a model-based utility function,

$$\begin{aligned} U : A \times \Theta &\to \mathbb{R} \\ a, \theta &\mapsto U(a, \theta). \end{aligned}$$

Because the ordering of the actions by their assigned utilities will in general vary with each model configuration θ,

$$\underset{a \in A}{\text{argmax}} \, \overline{U}(a, \theta) \neq \underset{a \in A}{\text{argmax}} \, \overline{U}(a, \theta'),$$

there is no longer an unambiguous best action.

Bayesian decision theory uses posterior inferences to inform decision making under exactly this sort of uncertainty. A posterior distribution $p(\theta \mid \tilde{y})$ informing the unknown behaviors defines posterior expected utilities for each action,

$$\overline{U}(a \mid \tilde{y}) = \int \mathrm{d}\theta \, p(\theta \mid \tilde{y}), U(a, \theta).$$

Given the observation $\tilde{y}$ the posterior expected utilities depend on only the actions. Consequently we can use them to unambiguously rank the actions and identify a single best action,

$$a_{\text{optimal}} = \underset{a \in A}{\text{argmax}} \, \overline{U}(a \mid \tilde{y}).$$

Again for simplicity I am ignoring the possibility of ties here.

3.2.2 Fairness Actions

The space of actions that we consider in a decision making problem is easy to take for granted. This is unfortunate because the worst decisions often arise not from an inappropriate utility function but rather from the restriction to an unnecessarily small space of actions, all of which might be burdened with poor consequences.

When questioning the fairness of these dice there appear to be two immediate actions that we can take:

- a_1: declare that the dice are fair,
- a_2: declare that the dice are unfair.

In some applications a more refined set of actions might be more useful. For example we could incorporate additional actions that avoid making any declaration at all or even trigger followup data collection.

Even if these two actions are sufficient we still cannot move forwards without defining exactly what makes a die "fair" or not. In the context of our simplex models we can formalize

fairness as uniformity of the simplex configuration. For example we might say that any simplex configuration that is not *exactly* uniform is unfair. With this definition of fairness our actions become

- a_1: declare that the true simplex configuration is uniform,
- a_2: declare that the true simplex configuration is not uniform.

This comparison between a single model configuration and all other model configurations is common in frequentist null hypothesis significance testing.

If we think of the uniform simplex configuration as an atomic subset then can we reinterpret these actions as

- a_1: declare that the true simplex configuration is in $\{\upsilon\}$,
- a_2: declare that the true simplex configuration is in $\{\upsilon\}^c$.

This suggests an immediate generalization to this formalization of fairness where we define a fair die by not $\{\upsilon\}$ but rather a subset of the simplex that contains the uniform simplex configuration, $\upsilon \in \mathsf{u}$,

- a_1: declare that the true simplex configuration is in u,
- a_2: declare that the true simplex configuration is in u^c.

In other words this generalization requires that a fair die be only sufficiently close to uniform. In hindsight this perspective is typically more relevant to actual practice; a die that deviates from exact uniformity by only an infinitesimal amount may not technically be fair, but we would never be able to distinguish it from a truly fair die within our finite lifetimes. Consequently the more relevant decision isn't actually whether these dice are *perfectly* uniform but rather whether or not these dice are *close enough* to being uniform that we wouldn't be able to tell the difference in a particular application.

Exactly how close to uniform is close enough to be fair, and how small the subset $\mathsf{u} \subset \Delta^5$ needs to be, will depend on the details of a given application. Any choice of simplex subset, however, motivates the same class of utility functions.

3.2.3 Choosing Between Model Configuration Subsets

The utility of claiming that the true simplex configuration falls within or without a subset of nearly uniform model configurations depends upon what the true simplex configuration is. If the true simplex configuration fall into u then taking action a_1 should result in some positive benefit u_{11}. On the other hand if the true simplex configuration falls into u^c then taking a_1 should result in lower, typically negative, utility u_{12}. Mathematically we can encode these assumptions into the utility assignment

$$U(a_1, q) = \begin{cases} u_{11}, & q \in \mathsf{u} \\ u_{12}, & q \notin \mathsf{u} \end{cases}$$

Similarly the utility assigned to the second action should take the form

$$U(a_2, q) = \begin{cases} u_{21}, & q \in \mathsf{u} \\ u_{22}, & q \notin \mathsf{u} \end{cases}.$$

Note that by allowing for asymmetric utilities we allow for the possibility that some correct decisions are more beneficial than others, while some incorrect statements are more costly than others.

Conveniently we can simplify both of these utility assignments into indicator functions,

$$\begin{aligned} U(a_1, q) &= u_{11}\, I_{\mathsf{u}}(q) + u_{12}\, (1 - I_{\mathsf{u}}(q)) \\ &= (u_{11} - u_{12})\, I_{\mathsf{u}}(q) + u_{12} \\ U(a_2, q) &= u_{21}\, I_{\mathsf{u}}(q) + u_{22}\, (1 - I_{\mathsf{u}}(q)) \\ &= (u_{21} - u_{22})\, I_{\mathsf{u}}(q) + u_{21}. \end{aligned}$$

So long the subset of sufficiently uniform simplex configurations u is measurable the posterior expected utilities for each action become

$$\begin{aligned} \overline{U}(a_1 \mid \tilde{y}) &= \int \mathrm{d}q\, p(q \mid \tilde{y})\, U(a_1, q) \\ &= \int \mathrm{d}q\, p(q \mid \tilde{y})\, ((u_{11} - u_{12})\, I_{\mathsf{u}}(q) + u_{12}) \\ &= (u_{11} - u_{12})\, \pi(\mathsf{u} \mid \tilde{y}) + u_{12} \end{aligned}$$

and

$$\begin{aligned} \overline{U}(a_2 \mid \tilde{y}) &= \int \mathrm{d}q\, p(q \mid \tilde{y})\, U(a_2, q) \\ &= \int \mathrm{d}q\, p(q \mid \tilde{y})\, ((u_{21} - u_{22})\, I_{\mathsf{u}}(q) + u_{21}) \\ &= (u_{21} - u_{22})\, \pi(\mathsf{u} \mid \tilde{y}) + u_{21}. \end{aligned}$$

The optimal Bayesian decision is to choose a_1 over a_2 if and only if

$$\begin{gathered} \overline{U}(a_1 \mid \tilde{y}) > \overline{U}(a_2 \mid \tilde{y}) \\ (u_{11} - u_{12})\, \pi(\mathsf{u} \mid \tilde{y}) + u_{12} > (u_{21} - u_{22})\, \pi(\mathsf{u} \mid \tilde{y}) + u_{21}, \end{gathered}$$

or, equivalently,

$$\pi(\mathsf{u} \mid \tilde{y}) > \frac{u_{21} - u_{12}}{u_{11} - u_{12} - u_{21} + u_{22}} \equiv \tau.$$

In words we declare that the dice are practically fair only when the posterior probability allocated to u is sufficiently large.

Note that the final decision doesn't depend on any of the individual benefits and costs but rather only the particular combination

$$t = \frac{u_{21} - u_{12}}{u_{11} - u_{12} - u_{21} + u_{22}}.$$

For example translating or scaling all of the benefits and costs by the same amount would result in the same t, and hence the same decisions.

To implement this Bayesian decision making process in practice we need to define the subset u and the probability threshold t.

The possibilities for t are, by construction, endless. In general the larger t is the more certain we have to be in the effective uniformity of the die behavior to declare fair dice.

Similarly we have infinite options for u. Here we'll consider three strategies for defining u: using exact uniformity, using direct features, and using pushforward features.

3.2.4 Exact Uniformity

Our initial, strict definition of fairness corresponds to taking

$$\mathsf{u} = \{\upsilon\}.$$

Unfortunately for our modeling assumptions, in particular our Dirichlet prior model, the posterior probability allocated to any atomic subset will always be zero,

$$\pi(\mathsf{u} \mid \tilde{y}) = \pi(\{\upsilon\} \mid \tilde{y}) = 0!$$

Consequently we will *never* choose a_1 so long as $t > 0$. On the other hand if we take $t = 0$ then we will *always* choose a_1 regardless of the realized data and resulting posterior distribution.

Intuitively the problem here is that the lone simplex configuration in $\mathsf{u} = \{\upsilon\}$ is overwhelmed by the infinitely many simplex configurations just outside of u. With only a finitely many rolls our posterior inferences will never be able to distinguish between the two.

There are only two circumstances in which we can have $t > 0$ but still might choose a_1. Firstly in the asymptotic limit of infinite rolls the posterior distribution will converge to a Dirac distribution with

$$\lim_{N \to \infty} \pi(\{\upsilon\} \mid \tilde{y}_1, \ldots, \tilde{y}_N) = 1$$

if the dice are exactly uniform and

$$\lim_{N \to \infty} \pi(\{\upsilon\} \mid \tilde{y}_1, \ldots, \tilde{y}_N) = 0$$

if the dice are not exactly uniform. In this ideal scenario we would choose a_1 when the dice are exactly uniform and a_2 when they are not for any value of $0 < t < 1$, as desired.

Secondly we can use an inflated prior model with the probability allocations

$$\pi(\mathsf{t}) = \alpha\, \delta_{\upsilon}(\mathsf{t}) + (1 - \alpha)\, \rho(\mathsf{t}).$$

In this case

$$\pi(\{\upsilon\}) = \alpha\, \delta_{\upsilon}(\upsilon) + (1 - \alpha)\, \rho(\upsilon).$$

The immediate issue here is that for any finite observation the posterior probability allocation will be completely determined by the strength of the inflation,

$$\begin{aligned} \pi(\{\upsilon\}) &= \alpha\, \delta_{\upsilon}(\upsilon) + (1 - \alpha)\, \rho(\upsilon). \\ &= \alpha\, 1 + (1 - \alpha)\, 0 \\ &= \alpha. \end{aligned}$$

Consequently the propensity to choose a_1 will be independent of the actual data we observe!

In general selecting between non-atomic subsets will be much better behaved than attempting to resolve any particular atomic subset from the infinitely many alternatives.

3.2.5 Effective Uniformity From Direct Features

One way to construct a subset of simplex configurations that contains the uniform simplex configuration is to impose a constraint on the component probabilities. For example we could require that all of the component probabilities are within some multiplicative tolerance of $1/6$,

$$\mathsf{u} = \left\{ (q_1, \ldots, q_6) \in \Delta^5 \mid \frac{1}{1 + t} \frac{1}{6} < q_k < (1 + t) \frac{1}{6} \right\}.$$

The smaller the tolerance t is the smaller the subset u will be.

Another useful strategy is motivated by the construction of open subsets in metric spaces. Recall that in Chapter Two we saw how any distance function

$$\begin{aligned} d : X \times X &\to \mathbb{R}^+ \\ x_1, x_2 &\mapsto d(x_1, x_2) \end{aligned}$$

implicitly defines open balls,

$$\mathsf{b}_{x,R} = \{x' \in X \mid d(x, x') < R\}.$$

By construction these subsets define neighborhoods of points around the center $x \in X$ whose size is determined by the radius $R \in \mathbb{R}^+$. Moreover if the distance function is measurable then these open balls will also be measurable.

Consequently any pair of a measurable distance function on the simplex Δ^5 and radius $R \in \mathbb{R}^+$ will defines a measurable subset of nearly uniform simplex configurations,

$$\mathsf{u} = \{q \in \Delta^5 \mid d(\upsilon, q) < R\}.$$

The smaller the radius is the stricter the notion of fairness these subsets will encode.

This construction can also be generalized beyond distance functions. Any measurable binary function

$$\begin{aligned} f : X \times X &\to \mathbb{R}^{+} \\ x_1, x_2 &\mapsto d(x_1, x_2) \end{aligned}$$

that vanishes when the two inputs are equal,

$$f(x, x) = 0,$$

can be used to construct two different ball-like, measurable subsets around a central point $x \in X$,

$$\mathsf{b}_{x,R} = \{x' \in X \mid f(x, x') < R\}$$

and

$$\mathsf{b}^{*}_{x,R} = \{x' \in X \mid f(x', x) < R\}.$$

I will refer to the any subset constructed in this way as a **radial subset**.

Distance functions on a simplex immediately satisfy this criterion, with the symmetry of their arguments implying that the two subsets we can define around the uniform simplex configuration are actually redundant,

$$\mathsf{b}_{\upsilon,R} = \mathsf{b}^{*}_{\upsilon,R}.$$

Because every point in a simplex defines a probability distribution we an also appeal to statistical divergences, such as the Kullback-Leibler divergence. Statistical divergences are not symmetric in their inputs so they define two distinct subsets around the uniform simplex configuration.

$$\mathsf{b}_{\upsilon,R} \neq \mathsf{b}^{*}_{\upsilon,R}.$$

Ultimately there are a wealth of distance and divergences functions on the simplex that we could use to define "sufficiently uniform". I review some of the possibilities and their properties in the Appendix A.

In a practical analysis we would need to use our domain expertise to choose a subset of sufficiently uniform simplex configurations whose shape and size are appropriate for a given application. That is not to say, however, that identifying the most appropriate shape and size is at all straightforward.

For the analysis here let's consider one subset defined by component-wise constraints and another defined by the geodesic distance function introduced in the Appendix A.

```
K <- 6
u <- 1 / 6
upsilon <- rep(u, K)

# Component-wise subset indicator function
componentwise_indicator <- function(q, t) {
  component_inclusion <- sapply(1:K, function(k) u / (1 + t) < q[k] &
                                                 q[k] < u * (1 + t) )
  Reduce("&", component_inclusion)
}

# Radial subset indicator function
geodesic_distance <- function(q) {
  2 * acos( sum(sqrt( q * upsilon )) )
}

radial_indicator <- function(q, R) {
  geodesic_distance(q) < R
}
```

To make these subsets somewhat comparable we can tune their size so that they are both allocated the same prior probability. Here let's require that each subset is allocated a prior probability of 0.001, enforcing a relatively strict notion of fairness.

```
# Prior configuration
alpha <- rep(1, 6)

# Subset configurations
t <- 0.412
R <- 0.21

# Monte Carlo estimates of prior probability allocations
S <- 100000

I_c_samples <- rep(NA, S)
I_r_samples <- rep(NA, S)

for (s in 1:S) {
  m <- rgamma(K, alpha, 1)
  q <- m / sum(m)

  I_c_samples[s] <- componentwise_indicator(q, t)
```

```
  I_r_samples[s] <- radial_indicator(q, R)
}
```

```
pi_c <- mean(I_c_samples)
delta_c <- 2 * sqrt(var(I_c_samples) / S)

cat(paste0("Prior probability allocated to component-wise subset = ",
           sprintf("%.5f +/- %.5f", pi_c, delta_c)))
```

```
Prior probability allocated to component-wise subset = 0.00115 +/- 0.00021
```

```
pi_r <- mean(I_r_samples)
delta_r <- 2 * sqrt(var(I_r_samples) / S)

cat(paste0("Prior probability allocated to radial subset = ",
           sprintf("%.5f +/- %.5f", pi_r, delta_r)))
```

```
Prior probability allocated to radial subset = 0.00107 +/- 0.00021
```

The posterior probability allocated to these subsets can then be estimated with Markov chain Monte Carlo. In practice we can evaluate these indicator functions directly in `R`, as we have done here, or in the `generated quantities` block of a new Stan program. Here we'll stay within `R`.

```
C <- 4
S <- 1024

I_c_samples <- matrix(NA, nrow=C, ncol=S)
I_r_samples <- matrix(NA, nrow=C, ncol=S)

for (c in 1:C) {
  for (s in 1:S) {
    q <- sapply(1:6, function(k)
                     samples[[paste0("q[", k, "]")]][c, s])

    I_c_samples[c, s] <- componentwise_indicator(q, t)
    I_r_samples[c, s] <- radial_indicator(q, R)
  }
}
```

```
est <- util$ensemble_mcmc_est(I_c_samples)
pi_c <- est[1]
delta_c <- 2 * est[2]

cat(paste0("Posterior probability allocated to component-wise subset = ",
           sprintf("%.5f +/- %.5f", pi_c, delta_c)))
```

```
Posterior probability allocated to component-wise subset = 0.99779 +/- 0.00146
```

```
est <- util$ensemble_mcmc_est(I_r_samples)
pi_r <- est[1]
delta_r <- 2 * est[2]

cat(paste0("Posterior probability allocated to radial subset = ",
           sprintf("%.5f +/- %.5f", pi_r, delta_r)))
```

```
Posterior probability allocated to radial subset = 1.00000 +/- 0.00000
```

We cannot make a final Bayesian decision, however, without specifying a probability threshold t. In a real application we might use Bayesian calibration to tune t to achieve desired false positive and true positive rates. That said the posterior probabilities here are so large that we would declare that the die are fair for all but the largest thresholds!

3.2.6 Effective Uniformity From Pushforward Features

One of the reasons why choosing an appropriate subset of effectively uniform simplex configurations, not to mention a corresponding probability threshold, can be so awkward is that most of us don't have experience reasoning about the high-dimensional 5-simplex. Typically lower-dimensional consequences are easier compare to our domain expertise. The practical challenge, however, is identifying which of these consequences are relevant in a given application.

Consider, for example, one of the many games of chance that rely not on all six outcomes from a single six-sided die roll but rather particular outcomes from multiple six-sided die rolls. For these games the more relevant notion of fairness is not now close the 5-simplex model is to the uniform simplex configuration but rather how close these particular consequences are to the those expected from a uniform die.

Apocalypse World (Baker and Baker 2016) is a table top role playing game that uses the sum of two six-sided die rolls to determine whether player actions succeed or fail. Since its introduction this straightforward mechanic has also been adopted by a variety of other role playing table top games.

The sum of two six-sided die rolls can take on eleven possible values (Table 1). Consequently the summation outcomes can be modeled with a 10-simplex space. Moreover we can use the rules for pushforward probability mass functions that we learned in Chapter 7 to map any point in the initial 5-simplex into a corresponding point in this 10-simplex.

Table 1: Many games of chance are based on the sum of two six-sided die rolls. The probabilities for each possible sum are given by pushing a product of two six-sided outcome probability distributions along the binary summation function, $f : (y_1, y_2) \mapsto s = y_1 + y_2$.

s	$f^{-1}(s)$	$q_s = f_* p(s)$
2	$(1,1)$	q_1^2
3	$(1,2), (2,1)$	$2\, q_1 q_2$
4	$(1,3), (2,2), (3,1)$	$2\, q_1\, q_3 + q_2^2$
5	$(1,4), (2,3), (3,2), (4,1)$	$2\, q_1\, q_4 + 2\, q_2\, q_3$
6	$(1,5), (2,4), (3,3), (4,2), (5,1)$	$2\, q_1\, q_5 + 2\, q_2\, q_4 + q_3^2$
7	$(1,6), (2,5), (3,4), (4,3), (5,2), (6,1)$	$2\, q_1\, q_6 + 2\, q_2\, q_5 + 2\, q_3, q_4$
8	$(2,6), (3,5), (4,4), (5,3), (6,2)$	$2\, q_2\, q_6 + 2\, q_3\, q_5 + q_4^2$
9	$(3,6), (4,5), (5,4), (6,3)$	$2\, q_3\, q_6 + 2\, q_4\, q_5$
10	$(4,6), (5,5), (6,4)$	$2\, q_4\, q_6 + q_5^2$
11	$(5,6), (6,5)$	$2\, q_5 q_6$
12	$(6,6)$	q_6^2

We can use this mathematical result to immediately derive the consequences from not only an idealized, exactly uniform die but also the real Skew Dice™.

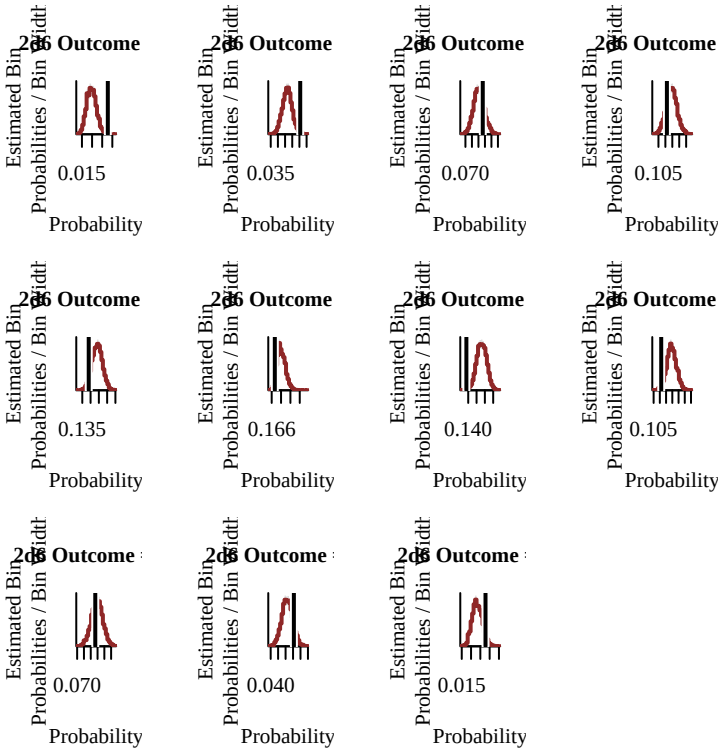
```
pushforward_simplex <- function(q) {
  c(    q[1] * q[1],
    2 * q[1] * q[2],
    2 * q[1] * q[3] +     q[2] * q[2],
    2 * q[1] * q[4] + 2 * q[2] * q[3],
    2 * q[1] * q[5] + 2 * q[2] * q[4] +     q[3] * q[3],
    2 * q[1] * q[6] + 2 * q[2] * q[5] + 2 * q[3] * q[4],
    2 * q[2] * q[6] + 2 * q[3] * q[5] +     q[4] * q[4],
    2 * q[3] * q[6] + 2 * q[4] * q[5],
    2 * q[4] * q[6] +     q[5] * q[5],
    2 * q[5] * q[6],
        q[6] * q[6])
}
```

```
unif_2d6_simplex <- pushforward_simplex(upsilon)
```

```
inferred_2d6_simplex <- lapply(1:11,
                               function(k) matrix(NA, nrow=C, ncol=S))
names(inferred_2d6_simplex) <- sapply(1:11,
                                      function(k) paste0('q[', k + 1, ']'))

C <- 4
S <- 1024

for (c in 1:C) {
  for (s in 1:S) {
    q <- sapply(1:6, function(k)
                     samples[[paste0("q[", k, "]")]][c, s])
    qs <- pushforward_simplex(q)
    for (k in 1:11) {
      inferred_2d6_simplex[[paste0('q[', k + 1, ']')]][c, s] <- qs[k]
    }
  }
}
```

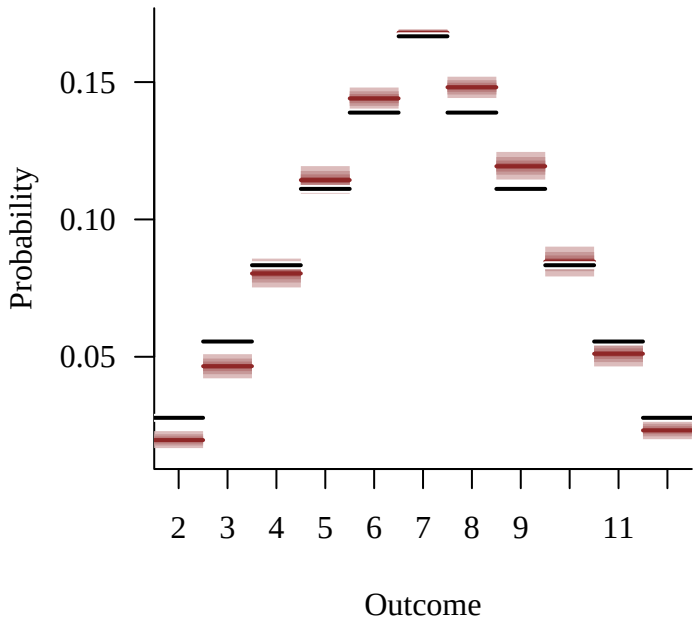
Overall the inferred behavior of rolling a Skew Dice™ twice is reasonably consistent with what we would expect from rolling an exactly uniform die twice.

```
par(mfrow=c(3, 4), mar=c(5, 5, 3, 1))

obs_counts <- table(data$outcome)

for (k in 1:11) {
  name <- paste0('q[', k + 1, ']')
  util$plot_expectand_pushforward(inferred_2d6_simplex[[name]], 25,
                                  display_name="Probability",
                                  baseline=unif_2d6_simplex[k])
  title(paste0("2d6 Outcome = ", k + 1))
}
```



Once again the nested quantile intervals are particularly useful for investigating for any systematic inconsistencies.

```
par(mfrow=c(1, 1))
util$plot_disc_pushforward_quantiles(inferred_2d6_simplex,
                                     names(inferred_2d6_simplex),
                                     baseline_values=unif_2d6_simplex,
                                     xlab="Outcome", xticklabs=2:12,
                                     ylab="Probability")
```



Here we see a *slight* reduction in the inferred probability of $s = 2$ and $s = 3$ relative to the exactly uniform baseline and a slight increase in the inferred probability of $s = 8$ and $s = 9$. That said the magnitude of this tension is weak at best.

Now the Apocalypse World game mechanics don't actually map all eleven possible sums to distinct game outcomes. Instead the individual sums are mapped into different degrees of success and failure. In particular to "do something under fire" one adds the outcomes from two rolls to their "cool" ability score; if

- $10 \leq s$ the player succeeds,
- $7 \leq s \leq 9$ the player flinches, hesitates, or stalls,
- $2 \leq s \leq 8$ the players fails.

For example if a player has a "cool" ability score of 2 then the sums

$$\mathsf{s}_{\text{success}} = \{8, 9, 10, 11, 12\}$$

result in a success, the sums

$$\mathsf{s}_{\text{flinch}} = \{5, 6, 7\}$$

result in a flinch, and the sums

$$\mathsf{s}_{\text{failure}} = \{2, 3, 4\}$$

result in a failure. The pushforward probability of each game outcome is then given by the

probability allocated to the corresponding level sets,

$$q_{\text{success}} = \pi(\mathsf{s}_{\text{success}}) = \sum_{s=8}^{12} q_s$$

$$q_{\text{flinch}} = \pi(\mathsf{s}_{\text{flinch}}) = \sum_{s=5}^{7} q_s$$

$$q_{\text{failure}} = \pi(\mathsf{s}_{\text{failure}}) = \sum_{s=2}^{4} q_s.$$

Note that these game outcome probabilities can also be interpreted as points on a lower-dimensonal 2-simplex.

Critically the relevant notion of fairness when playing Apocalypse World with this particular "cool" ability score isn't uniformity on the full 10-simplex but rather uniformity on the 2-simplex defined by the variables

$$(q_{\text{success}}, q_{\text{flinch}}, q_{\text{failure}}).$$

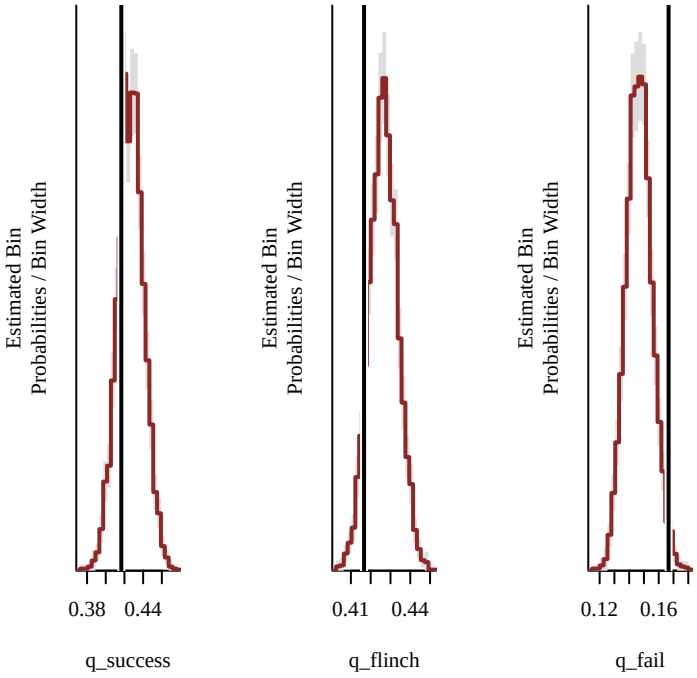
We can reduce this even further by considering how close just q_{success} is to what we would expect from an exactly uniform die.

```
q_success_unif <- sum(unif_2d6_simplex[7:11])
q_flinch_unif <- sum(unif_2d6_simplex[4:6])
q_fail_unif <- sum(unif_2d6_simplex[1:3])
```

```
inferred_outcomes <- lapply(1:3, function(n) matrix(NA, nrow=C, ncol=S))
names(inferred_outcomes) <- c('q_success', 'q_flinch', 'q_fail')

for (c in 1:C) {
  for (s in 1:S) {
    q <- sapply(1:6, function(k)
      samples[[paste0("q[", k, "]")]][c, s])
    q_2d6 <- pushforward_simplex(q)

    inferred_outcomes[['q_success']][c, s] <- sum(q_2d6[7:11])
    inferred_outcomes[['q_flinch']][c, s] <- sum(q_2d6[4:6])
    inferred_outcomes[['q_fail']][c, s] <- sum(q_2d6[1:3])
  }
}
```

Here the probabilities of these game outcomes that we have inferred from rolling the Skew Dice™ is consistent with what we would expect from rolling exactly uniform dice.

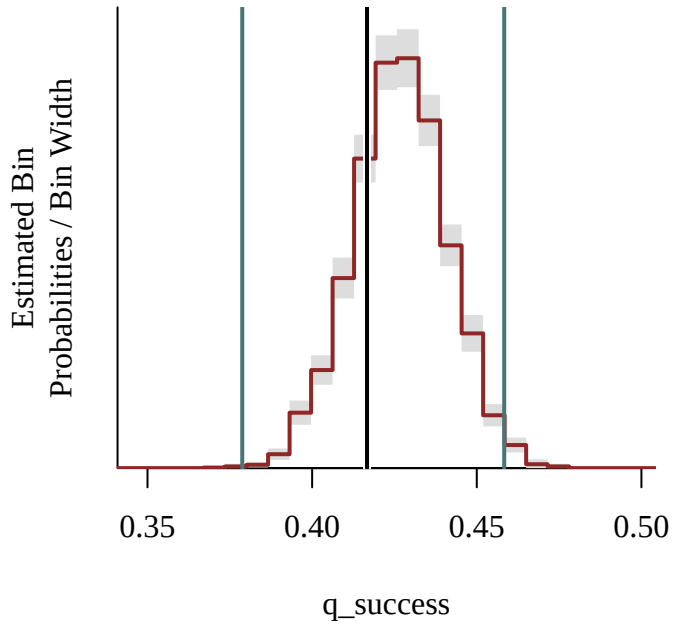
```
par(mfrow=c(1, 3), mar = c(5, 5, 3, 1))

util$plot_expectand_pushforward(inferred_outcomes[['q_success']], 25,
                                display_name="q_success",
                                baseline=q_success_unif)

util$plot_expectand_pushforward(inferred_outcomes[['q_flinch']], 25,
                                display_name="q_flinch",
                                baseline=q_flinch_unif)

util$plot_expectand_pushforward(inferred_outcomes[['q_fail']], 25,
                                display_name="q_fail",
                                baseline=q_fail_unif)
```



We can then use these inferences to inform explicit decisions about fairness. If we take the most relevant notion fairness in Apocalypse World to be whether or not the q_{success} of the Skew Dice™ falls into an interval around the value expected from exactly uniform dice then our decisions will be driven by the posterior probability allocated to this interval. Note that this is a special case of the decision making process that we considered in Section 3.2.3 only using a subset defined by this particular, application-motivated pushforward constraint.

For example we might consider a multiplicative interval with tolerance $t = 0.1$

```
t <- 0.1

par(mfrow=c(1, 1))

flim <- c(0.9 * q_success_unif / (1 + t),
          1.1 * q_success_unif * (1 + t))
util$plot_expectand_pushforward(inferred_outcomes[['q_success']], 25,
                                display_name="q_success", flim=flim,
                                baseline=q_success_unif)

c_mid_teal <- c("#487575")
abline(v=q_success_unif / (1 + t), lwd=2, col=c_mid_teal)
abline(v=q_success_unif * (1 + t), lwd=2, col=c_mid_teal)
```




```
C <- 4
S <- 1024

I_samples <- matrix(NA, nrow=C, ncol=S)

for (c in 1:C) {
  for (s in 1:S) {
    q <- inferred_outcomes[['q_success']][c, s]
    I_samples[c, s] <- q_success_unif / (1 + t) < q &
                       q < q_success_unif * (1 + t)
```

```
  }
}
```

```
est <- util$ensemble_mcmc_est(I_samples)

cat(paste0("Posterior probability allocated to component-wise subset = ",
           sprintf("%.5f +/- %.5f", est[1], 2 * est[2])))
```

```
Posterior probability allocated to component-wise subset = 0.98695 +/- 0.00358
```

The only remaining step is to choose a probability threshold t for how large this posterior probability needs to be before we claim that the dice are fair for this particular move in the game. This might for example depend on how many games we intend to play or just how careful the person running the game is. In this case, however, only the strictest criteria will reject the extremely large posterior probability.

4 Heterogeneous Simplices

Perhaps the strongest assumption that we have made so far is that all of the Skew Dice™ sent out to volunteers behave identically to each other and, consequently, can be modeled by a single simplex configuration. If any deviations from uniformity are actually heterogeneous across the dice then this assumption can obscure the true behavior of the dice.

4.1 Heterogeneous Summary Statistics

Before expanding our initial model to incorporate heterogeneous simplex configurations, however, let's first engineer some summary statistics that are sensitive to the consequences of that heterogeneity. The ideal summary statistics will be sensitive to *systematic* differences in the roll outcomes that cannot be attributed to differences in the number of rolls. We can then rerun our initial, homogeneous model while evaluating these new summary statistics and construct new retrodictive checks to see if we can discern any inadequacies with the assumption of a common die behavior.

One immediate strategy for designing summary statistics sensitive to heterogeneity in the die behavior is to evaluate our initial summary statistics on only the data from a single die at a time. For example we can construct histograms of the observed face counts for each die,

$$\frac{n_{d,1}}{\sum_{k=1}^{6} n_{d,k}}, \ldots, \frac{n_{d,6}}{\sum_{k=1}^{6} n_{d,k}},$$

with d indexing the individual die.

The only issue with these individual histograms is that differences in the die behavior will manifests in differences in the *shape* of the histograms but not their *area*. Instead differences in their area are completely determined by the varying number of rolls. That said we can readily isolate the shape by looking at *normalized* histograms which quantify the relative frequency of each outcome,

$$\hat{q}_{d,k} = \frac{n_{d,k}}{N_d}$$

with

$$N_d = \sum_{k'=1}^{6} n_{d,k'}.$$

Conveniently the empirical entropy statistic is already constructed from these relative frequencies,

$$\begin{aligned} \hat{H}_d &= \hat{H}(n_{d,1}, \ldots, n_{d,6}) \\ &= -\sum_{k=1}^{6} \frac{n_{d,k}}{N_d} \log \left(\frac{n_{d,k}}{N_d} \right). \end{aligned}$$

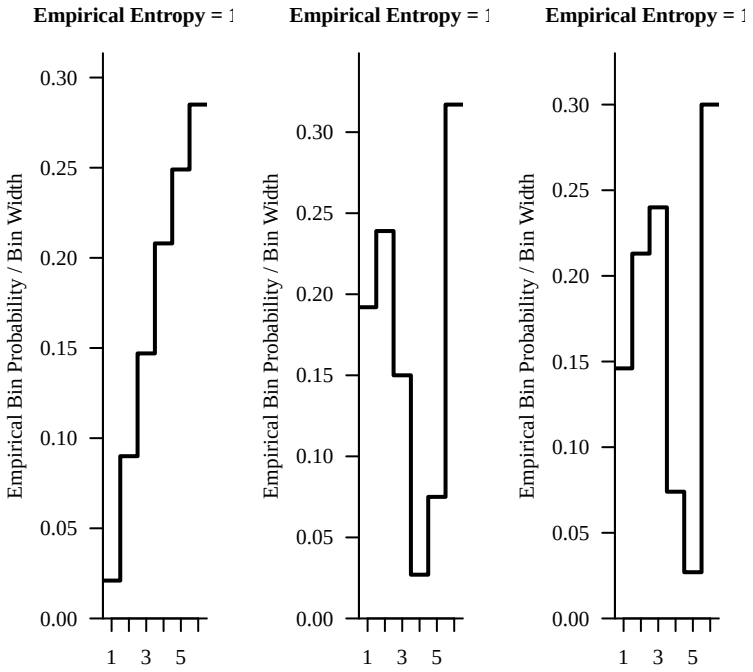
Consequently differences in the empirical entropies for each die will largely be insensitive to the varying total number of rolls.

On the other hand the empirical entropy function is invariant to *permutations* of the faces: it always gives the same value no matter how the face counts are permuted. Consequently any retrodictive check based on the empirical entropy statistic will not be able to differentiate between many die behaviors.

To see this ignorance let's sample from three data generating processes, each defined by a simplex configuration that is a permutation of the others. While the shapes of the normalized histograms clearly differentiate the three processes from each other, the empirical entropies are nearly identical.

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

compute_empirical_entropy <- function(outcomes) {
  counts <- table(outcomes)
  N <- sum(counts)
  sum(sapply(counts, function(n) -(n / N) * log(n / N)))
}

N <- 1000
q1 <- c(0.025, 0.075, 0.15, 0.2, 0.25, 0.3)
outcomes1 <- sample(1:6, N, replace=TRUE, prob=q1)
```

```
obs_entropy1 <- compute_empirical_entropy(outcomes1)
util$plot_line_hist(outcomes1, 0.5, 6.5, 1, prob=TRUE,
                    main=sprintf("Empirical Entropy = %.3f", obs_entropy1))

q2 <- sample(q1)
outcomes2 <- sample(1:6, N, replace=TRUE, prob=q2)
obs_entropy2 <- compute_empirical_entropy(outcomes2)
util$plot_line_hist(outcomes2, 0.5, 6.5, 1, prob=TRUE,
                    main=sprintf("Empirical Entropy = %.3f", obs_entropy2))

q3 <- sample(q1)
outcomes3 <- sample(1:6, N, replace=TRUE, prob=q3)
obs_entropy3 <- compute_empirical_entropy(outcomes3)
util$plot_line_hist(outcomes3, 0.5, 6.5, 1, prob=TRUE,
                    main=sprintf("Empirical Entropy = %.3f", obs_entropy3))
```



Visual comparisons between summary statistics evaluated on the data from individual die are typically the most informative, but at the same time they can quickly become overwhelming as the number of dice increase. In practice it can be useful to complement these visual comparisons with more quantitative comparisons. In particular composing any one-dimensional summary statistic with an empirical measure of variation, such as the empirical variance or mean absolute deviation, quantifies the heterogeneity of that statistic. The resulting retrodictive

checks can then be used to critique how well the model captures that particular manifestation of heterogeneity.

4.2 Critiquing the Homogeneous Simplex Model

Conveniently all of these summary statistics are straightforward to implement in Stan.

Unsurprisingly the additional summaries don't affect the actual posterior quantification.

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

```
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.
```

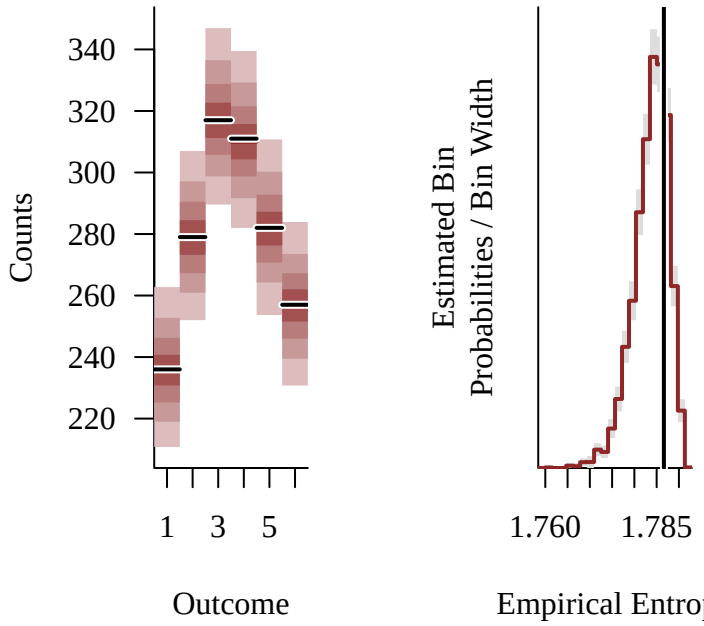
```
samples <- util$extract_expectands(fit)
base_samples <- util$filter_expectands(samples, 'q', check_arrays=TRUE)
util$check_all_expectand_diagnostics(base_samples)
```

```
All expectands checked appear to be behaving well enough for reliable
Markov chain Monte Carlo estimation.
```

While our focus is now on critiquing the homogeneity assumption the best practice is to run through our existing retrodictive checks first. This allows us to catch any unexpected consequences when we start modifying our model. Because we're rerunning our previous model here, however, the retrodictive performance is the same.

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

obs_counts <- table(data$outcome)

pred_names <- sapply(1:6, function(k) paste0('pred_counts[', k, ']'))
util$plot_disc_pushforward_quantiles(samples, pred_names,
                                     baseline_values=obs_counts,
                                     xlab="Outcome", ylab="Counts")
```

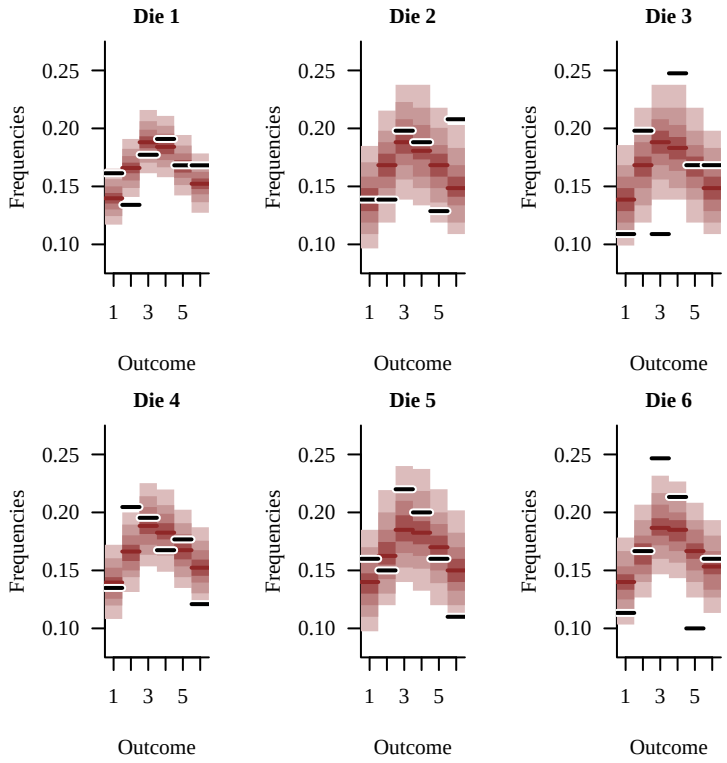
```
obs_entropy <- sum(sapply(obs_counts,
                          function(n) -(n / data$N) * log(n / data$N)))

util$plot_expectand_pushforward(samples[["pred_entropy"]], 20,
                                display_name="Empirical Entropy",
                                baseline=obs_entropy)
```

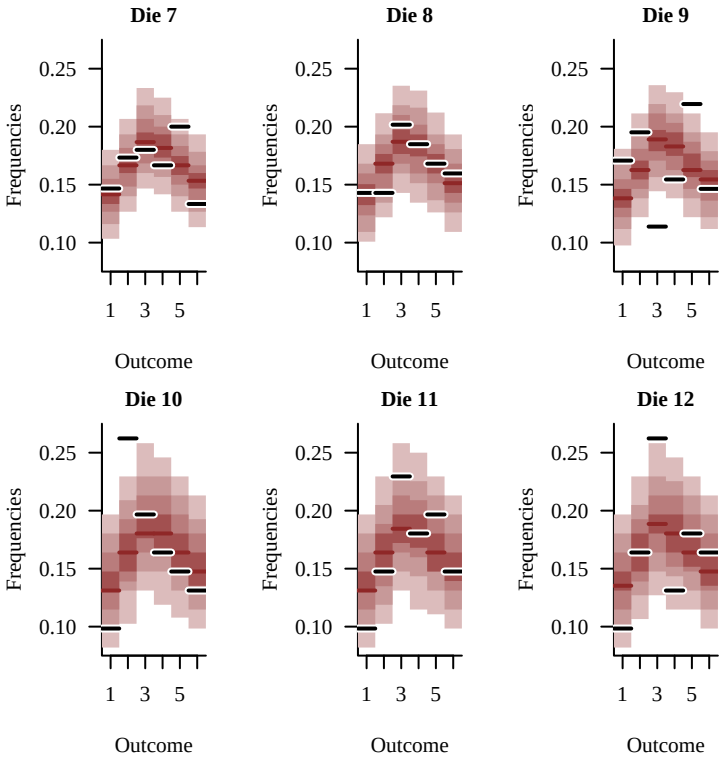


Now we can start hunting for signs of heterogeneity. Interestingly the outcome frequencies exhibit different patterns not only across the individual dice but also relative to the frequencies derived from all of the dice at once. That said of these patterns are inconsistent given the uncertainty in the posterior predictive distributions. Note also that the variation in how often each die was rolled manifests in heterogeneous posterior predictive uncertainties from die to die.

```
for (d in 1:data$N_dice) {
  if (d %% 6 == 1) par(mfrow=c(2, 3), mar=c(4, 5, 2, 1))

  obs_counts <- table(data$outcome[data$die_idxs == d])
  obs_freq <- obs_counts / sum(obs_counts)

  pred_names <- sapply(1:6,
                       function(k) paste0('pred_freq_die[', d, ',', k, ']'))
  util$plot_disc_pushforward_quantiles(samples, pred_names,
                                       baseline_values=obs_freq,
```

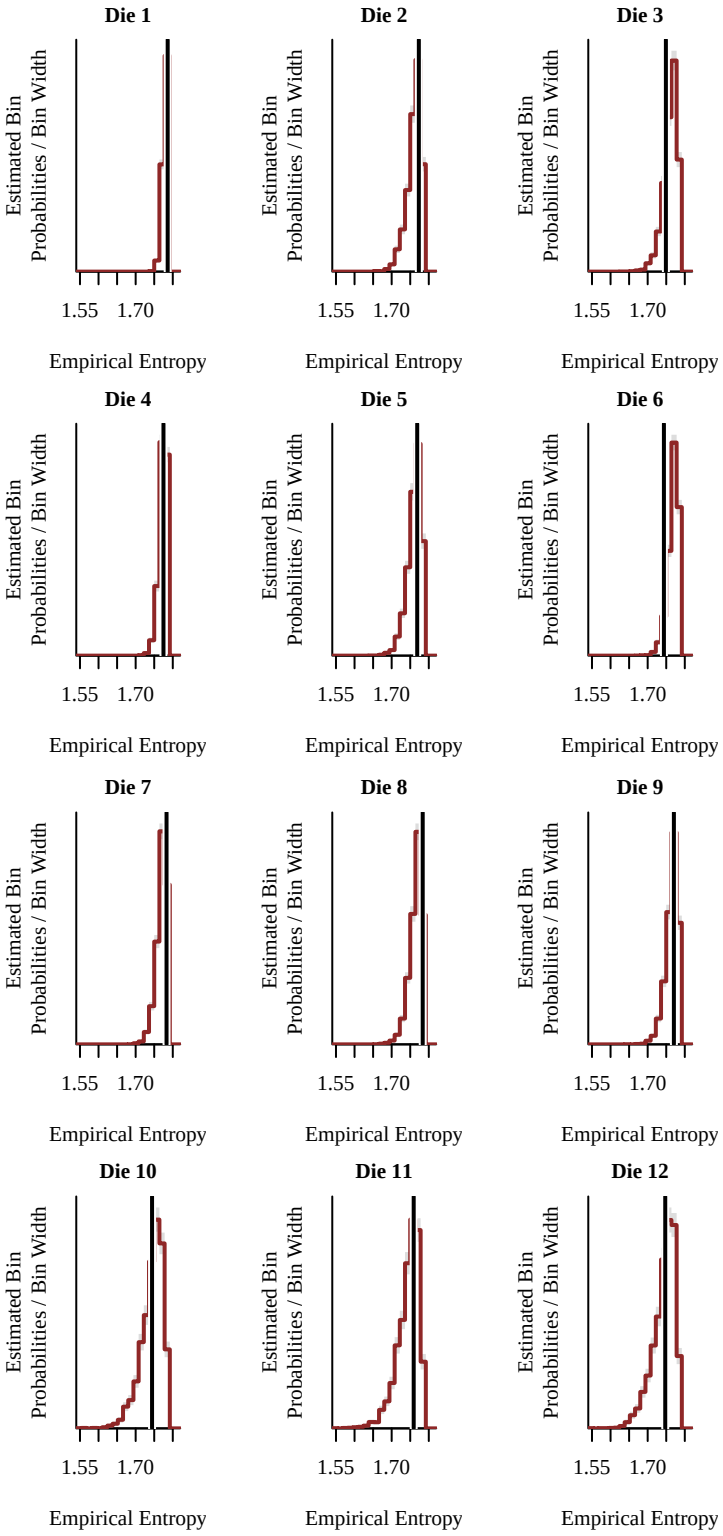
```
                                    xlab="Outcome", ylab="Frequencies",
                                    display_ylim=c(0.075, 0.275),
                                    main=paste("Die", d))
}
```





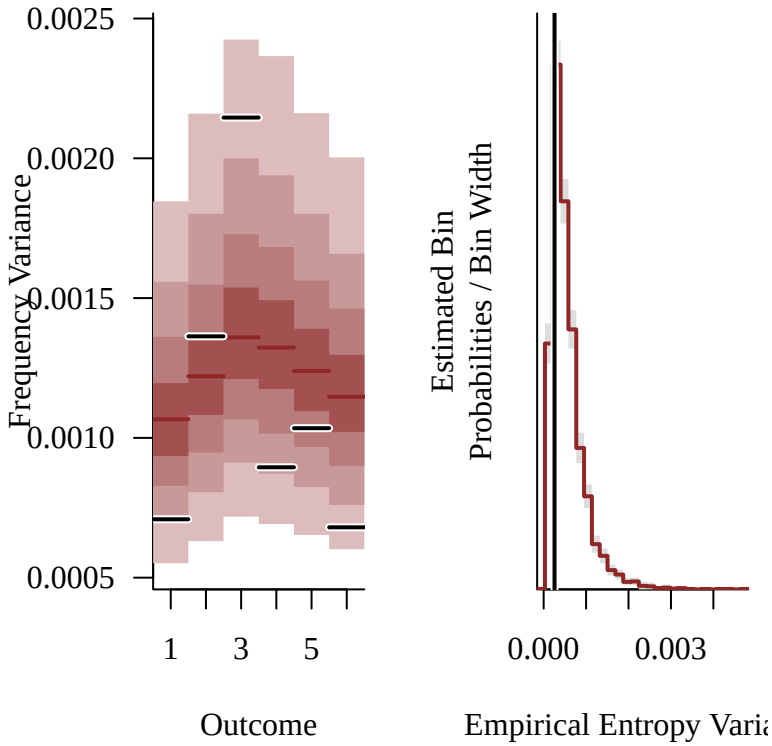
Likewise the small variation in the empirical entropies across the dice is consistent with the posterior predictive uncertainty.

```
for (d in 1:data$N_dice) {
  if (d %% 6 == 1) par(mfrow=c(2, 3), mar=c(4, 5, 2, 1))

  obs_entropy <- compute_empirical_entropy(data$outcome[data$die_idxs == d])

  name <- paste0('pred_entropy_die[', d, ']')
  util$plot_expectand_pushforward(samples[[name]], 20, flim=c(1.54, 1.82),
                                  display_name="Empirical Entropy",
                                  baseline=obs_entropy, main=paste("Die", d))
}
```


Die 1
Die 2
Die 3
Die 4
Die 5
Die 6
Die 7
Die 8
Die 9
Die 10
Die 11
Die 12
Estimated Bin Probabilities / Bin Width
1.55
1.70
Empirical Entropy

The empirical variance of the individual outcome frequencies and empirical entropies reinforces the adequacy of the homogeneous model.

```
par(mfrow=c(1, 2), mar=c(4, 4, 1, 0.5))

compute_freq <- function(k, d) {
  obs_counts <- table(data$outcome[data$die_idxs == d])
  obs_counts[k] / sum(obs_counts)
}

obs_var_freq <- sapply(1:6, function(k)
                           var(sapply(1:data$N_dice,
                                      function(d) compute_freq(k, d))))

pred_names <- sapply(1:6, function(k) paste0('pred_var_freq[', k, ']'))
util$plot_disc_pushforward_quantiles(samples, pred_names,
                                     baseline_values=obs_var_freq,
                                     xlab="Outcome",
                                     ylab="Frequency Variance")

die_data <- function(d) {
 data$outcome[data$die_idxs == d]
}

obs_var_entropy <- var(sapply(1:data$N_dice,
                              function(d) compute_empirical_entropy(die_data(d))))

util$plot_expectand_pushforward(samples[['pred_var_entropy']], 25,
                                display_name="Empirical Entropy Variance",
                                baseline=obs_var_entropy)
```



4.3 Inferring Heterogeneity

At this point there doesn't seem to be any heterogeneous die behavior that we can resolve with this particular collection of data. If we really want to be rigorous, however, then we can consider expanding the model to allow for heterogeneity anyways. In particular this allows us to turn the deduction question of adequacy into an inductive question about inferred uniformity, just as it did when we expanded the uniform simplex model into a general simplex model.

Fortunately this expansion is straightforward to implement; all we have to do is separate the outcome counts by die and then replicate the simplex parameters for each die.

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

This more sophisticated model hasn't introduced any computational complications.

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

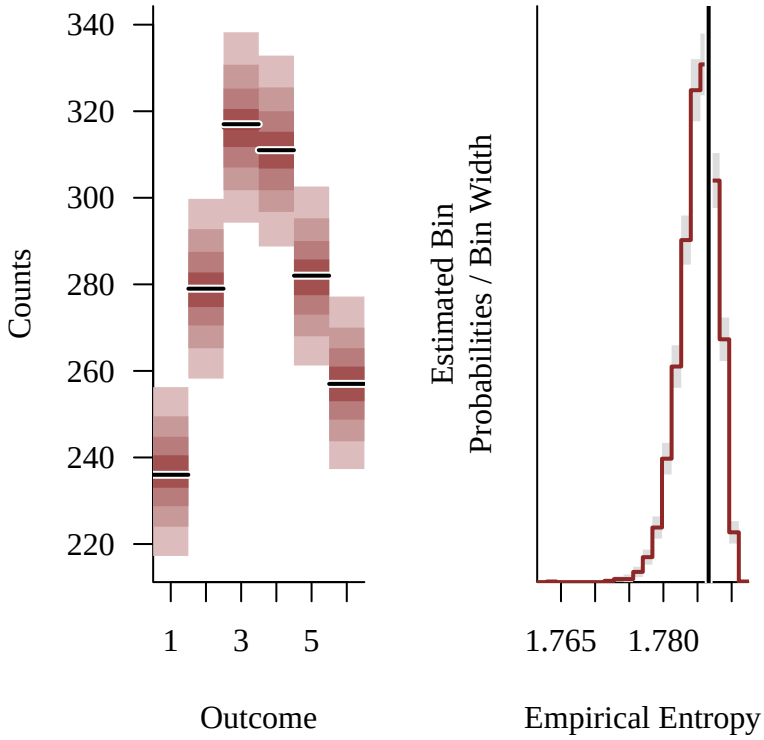
```
  All Hamiltonian Monte Carlo diagnostics are consistent with reliable
Markov chain Monte Carlo.
```

```
samples <- util$extract_expectands(fit)
base_samples <- util$filter_expectands(samples, 'q', check_arrays=TRUE)
util$check_all_expectand_diagnostics(base_samples)
```

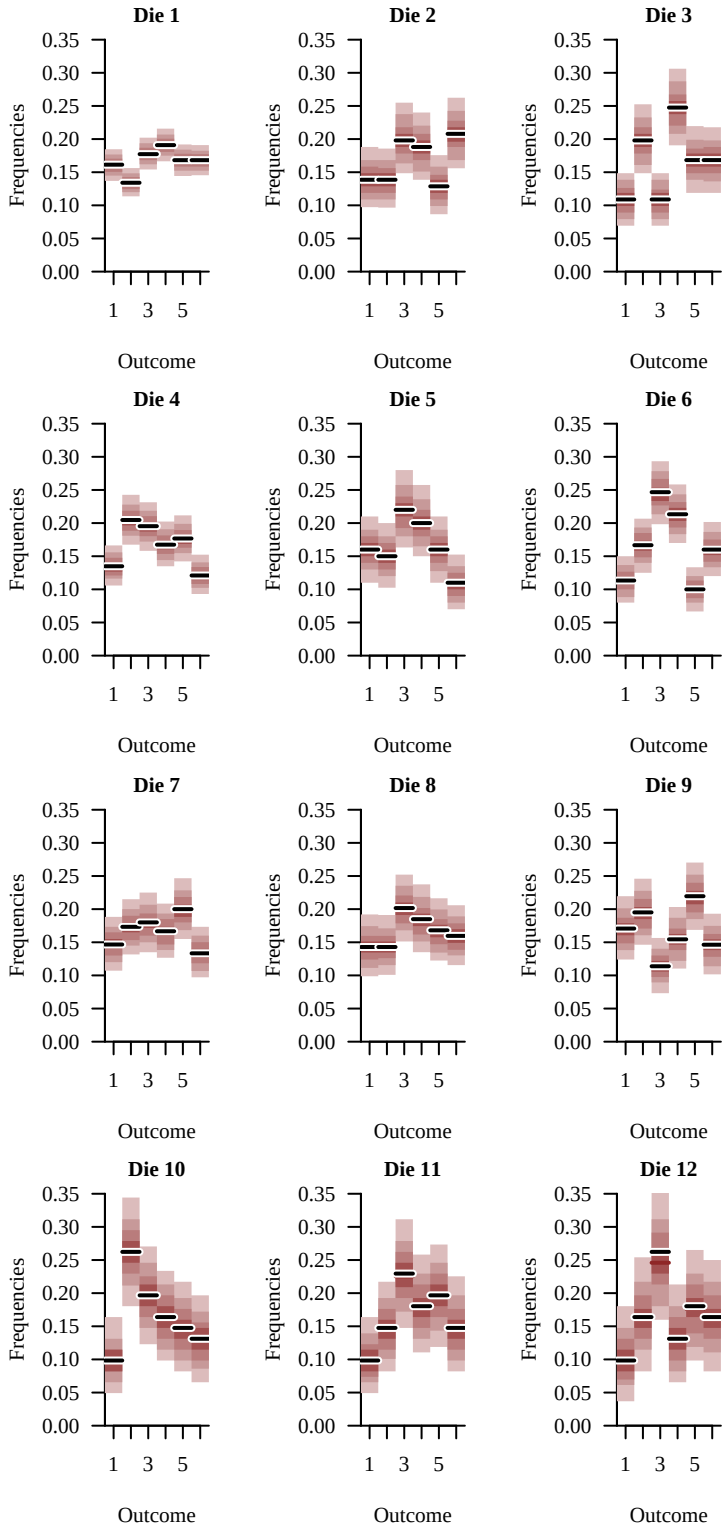
```
All expectands checked appear to be behaving well enough for reliable
Markov chain Monte Carlo estimation.
```

Unsurprisingly adding more flexibility to the model doesn't compromise the retrodictive performance for neither the homogeneous summary statistics nor the heterogeneous summary statistics.

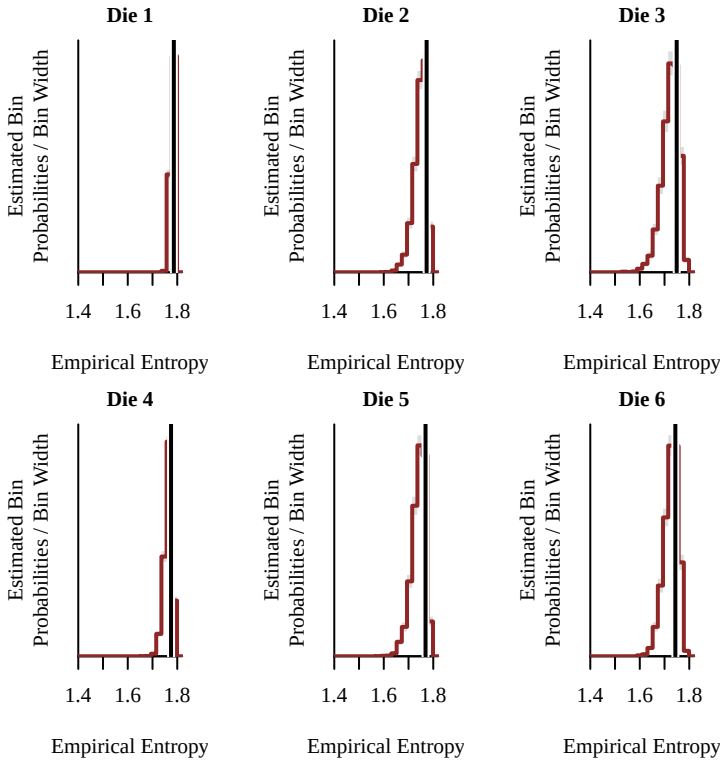
```
par(mfrow=c(1, 2))

obs_counts <- table(data$outcome)

pred_names <- sapply(1:6, function(k) paste0('pred_counts[', k, ']'))
util$plot_disc_pushforward_quantiles(samples, pred_names,
                                     baseline_values=obs_counts,
                                     xlab="Outcome", ylab="Counts")

obs_entropy <- sum(sapply(obs_counts,
                          function(n) -(n / data$N) * log(n / data$N)))

util$plot_expectand_pushforward(samples[["pred_entropy"]], 20,
                                display_name="Empirical Entropy",
                                baseline=obs_entropy)
```




```
for (d in 1:data$N_dice) {
  if (d %% 6 == 1) par(mfrow=c(2, 3), mar=c(4, 5, 2, 1))

  obs_counts <- table(data$outcome[data$die_idxs == d])
  obs_freq <- obs_counts / sum(obs_counts)

  pred_names <- sapply(1:6,
                       function(k) paste0('pred_freq_die[', d, ',', k, ']'))
  util$plot_disc_pushforward_quantiles(samples, pred_names,
                                       baseline_values=obs_freq,
                                       xlab="Outcome", ylab="Frequencies",
                                       display_ylim=c(0, 0.35),
                                       main=paste("Die", d))
}
```

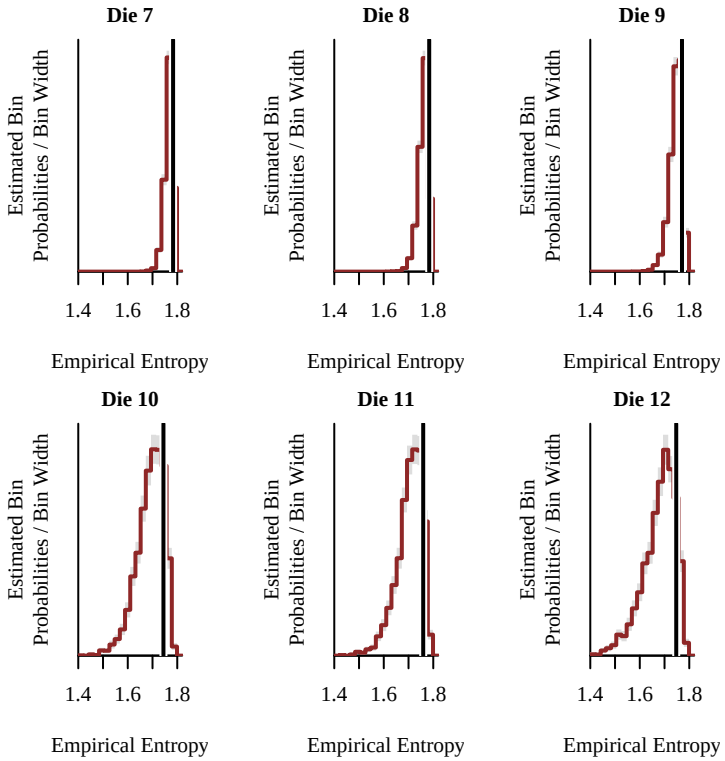

Die 1
Die 2
Die 3
Die 4
Die 5
Die 6
Die 7
Die 8
Die 9
Die 10
Die 11
Die 12
Frequencies
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
1
3
5
Outcome

```
for (d in 1:data$N_dice) {
  if (d %% 6 == 1) par(mfrow=c(2, 3), mar=c(4, 5, 2, 1))

  obs_entropy <- compute_empirical_entropy(data$outcome[data$die_idxs == d])

  name <- paste0('pred_entropy_die[', d, ']')
  util$plot_expectand_pushforward(samples[[name]], 20, flim=c(1.4, 1.82),
                                  display_name="Empirical Entropy",
                                  baseline=obs_entropy, main=paste("Die", d))
}
```




```
Warning in util$plot_expectand_pushforward(samples[[name]], 20, flim = c(1.4, :
2 samples (0.0%) fell below the histogram binning.
```

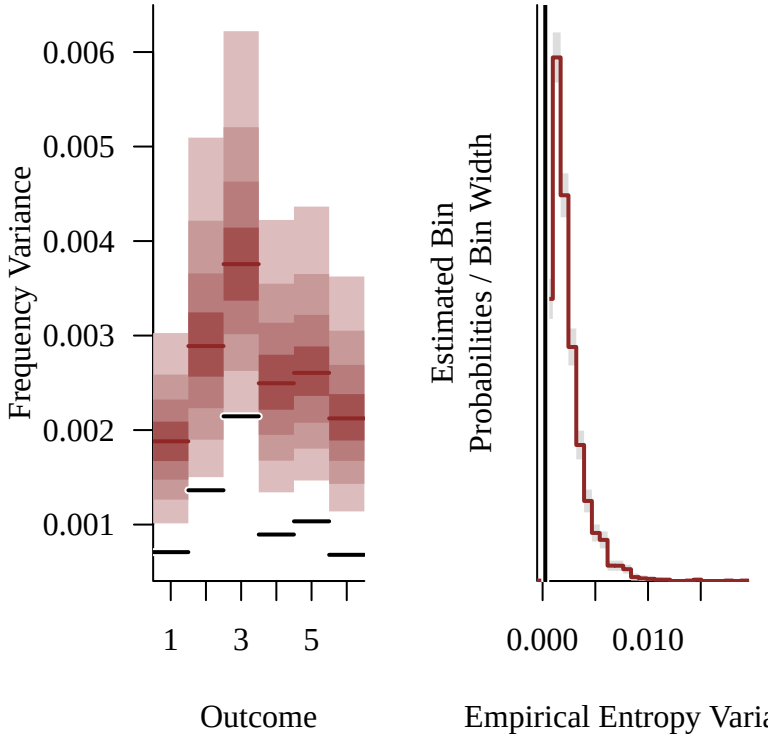
```
Warning in util$plot_expectand_pushforward(samples[[name]], 20, flim = c(1.4, :
7 samples (0.2%) fell below the histogram binning.
```



We have to be careful with interpreting the retrodictive checks based on variance summary statistics. The added flexibility of the heterogeneous model results inflates the variation in the posterior predictive behaviors, pulling the variance predictions to larger and larger values. While the observed values are no longer in the center of the posterior predictive values they are still very much consistent.

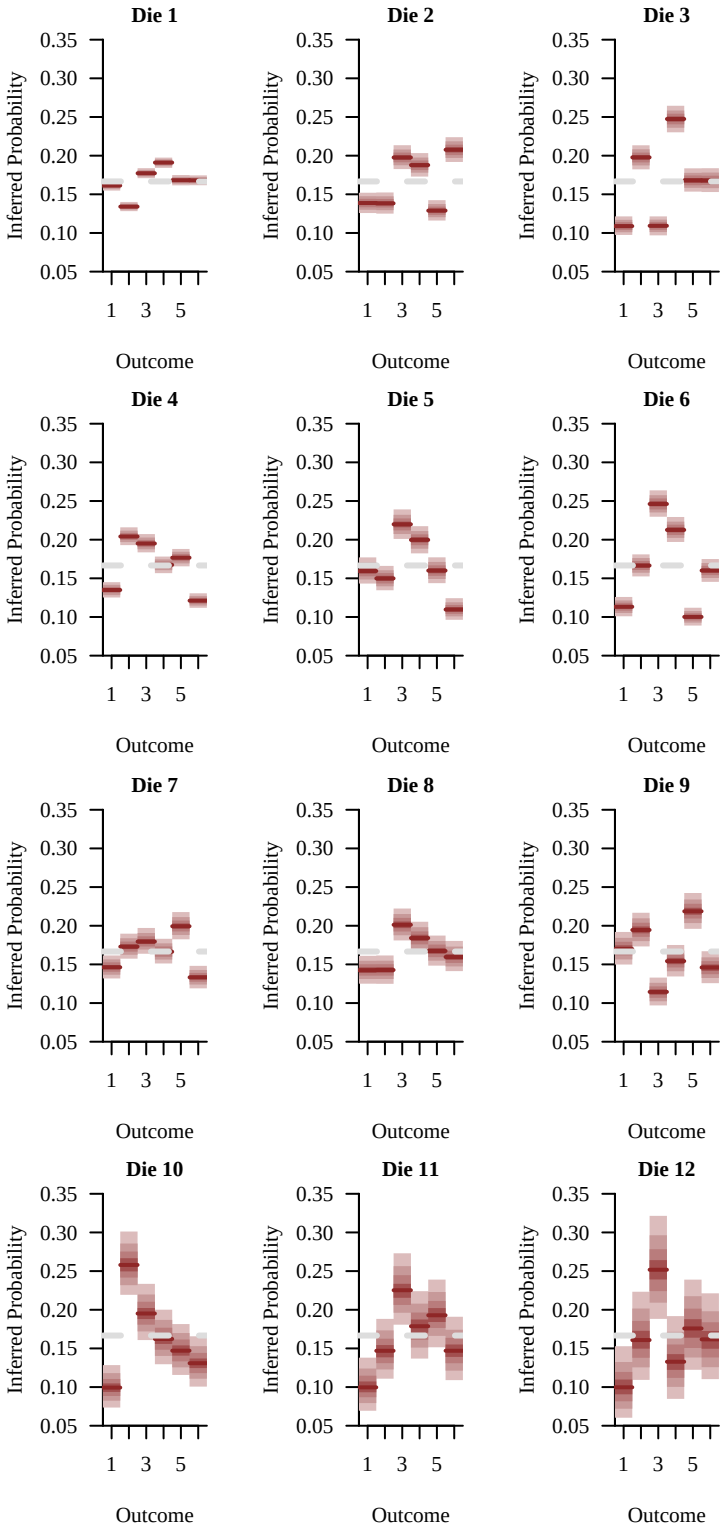
```
par(mfrow=c(1, 2), mar=c(4, 4, 1, 0.5))

obs_var_freq <- sapply(1:6, function(k)
                           var(sapply(1:data$N_dice,
                                      function(d) compute_freq(k, d))))

pred_names <- sapply(1:6, function(k) paste0('pred_var_freq[', k, ']'))
util$plot_disc_pushforward_quantiles(samples, pred_names,
                                     baseline_values=obs_var_freq,
                                     xlab="Outcome",
                                     ylab="Frequency Variance")

obs_var_entropy <- var(sapply(1:data$N_dice,
                              function(d) compute_empirical_entropy(die_data(d))))
```

```
util$plot_expectand_pushforward(samples[['pred_var_entropy']], 25,
                                display_name="Empirical Entropy Variance",
                                baseline=obs_var_entropy)
```



This completely heterogeneous model results in independent inferences for each die.

```
for (d in 1:data$N_dice) {
  if (d %% 6 == 1) par(mfrow=c(2, 3), mar=c(4, 5, 2, 1))

  names <- sapply(1:6, function(k) paste0('q[', d, ',', k, ']'))
  util$plot_disc_pushforward_quantiles(samples, names,
                                       xlab="Outcome",
                                       ylab="Inferred Probability",
                                       display_ylim=c(0.05, 0.35),
                                       main=paste("Die", d))
  abline(h=1/6, col="#DDDDDD", lwd=3, lty=2)
}
```


Die 1
Die 2
Die 3
Die 4
Die 5
Die 6
Die 7
Die 8
Die 9
Die 10
Die 11
Die 12
Inferred Probability
0.35
0.30
0.25
0.20
0.15
0.10
0.05
1
3
5
Outcome

These individual inferences are clearly drawn to the pattern of relative frequencies in each die, although none of them completely exclude the uniform simplex configuration. Interestingly these inferred simplex behaviors differ from what we inferred when treating the die as equivalent. This suggests that these very weak non-uniformities may just be artifacts of the unpredictable outcomes and not a manifestation of any systematic property of the dice.

5 Hierarchical Simplices

The precision of the independent simplex model inferences are limited by the data collected for each die. We may, however, be able to reduce these inferential uncertainties with a model that couples the die behaviors together without assuming that they are exactly the same.

Theoretically we might be able to distinguish individual dice from each other by comparing precise measurements of physical properties, such as overall mass, center of mass, face areas, and the like. Moreover if the dice were manufactured in different batches or in different factories then that provenance could be used to differentiate the dice. In this case, however, we don't have access to any of that information. We have no domain expertise that can discriminate one die sent to volunteers from another, let alone to other dice from the full order, or even hypothetical dice that could be manufactured in the future.

In other words our domain expertise is infinitely exchangeable, and hierarchical modeling is a natural way to couple the die behaviors together. Indeed drawing objects from a bag of objects is about as ideal of an exchangeable system as we are likely to encounter in practice! As an added bonus a hierarchical model would allow us to make predictions about new Skew Dice™ not included in our measurements.

To implement a hierarchical model over the individual die behaviors we will need to engineer a population model

$$p(q_{d,1}, \ldots, q_{d,6} \mid \phi)$$

that is not only multi-dimensional but also respects the simplex constraints. The ubiquitous, one-dimensional normal model or its multivariate generalization just won't do here.

Surprisingly there are a variety of possible hierarchical population modeling strategies at our disposal. In this section we'll review just a few of them.

5.1 Hyper Population Model

One strategy for developing hierarchical population models is to start with an appropriate family of probability density functions and then elevate some of the configuration variables to model parameters. Because these new parameters are sometimes referred to as "hyperparameters" I will refer to this as the "hyper population model" approach for lack of a better term.

5.1.1 Dirichlet Population Model

An immediate candidate for a population model over a simplex is the Dirichlet family of probability density functions,

$$\text{dirichlet}(q_{d,1}, \ldots, q_{d,6} \mid \alpha_1, \ldots, \alpha_6) = \frac{\Gamma\left(\sum_{k=1}^{K} \alpha_k\right)}{\prod_{k=1}^{K} \Gamma\left(\alpha_k\right)} \prod_{k=1}^{K} q_{d,k}^{\alpha_k - 1}.$$

All we have to do is elevate the variables

$$\alpha_1, \ldots, \alpha_6$$

to model parameters and then introduce a population prior model

$$p(\alpha_1, \ldots, \alpha_6)$$

compatible with our domain expertise.

The construction of a principled population prior model, however, is limited by our ability to interpret how the $\alpha_1, \ldots, \alpha_6$ moderate simplex behaviors. Conveniently we can enhance the interpretability of the Dirichlet population model by factoring the parameters $\alpha_1, \ldots, \alpha_K$ into a *baseline simplex* q_{baseline} and a *dispersion* parameter_ ω,

$$(\alpha_1, \ldots, \alpha_K) = \frac{1}{\omega} \cdot (q_{\text{baseline},1}, \ldots, q_{\text{baseline},K}) + (1, \ldots, 1),$$

with $\omega > 0$ and

$$0 \leq q_{\text{baseline},k} \leq 1$$
$$\sum_{k=1}^{K} q_{\text{baseline},k} = 1$$

a simplex configuration.

As ω decreases towards zero the population model concentrates more and more strongly around q_{baseline}, pulling the individual simplex configurations together with it. On the other hand when ω increases towards infinity the population model becomes more and more uniform, eventually decoupling the individual simplex configuration from each other entirely. In other words ω directly quantifies the strength of the population *heterogeneity*.

```
lmult <- function(x, a) {
  if (a == 0) {
    (0)
  } else {
    (a * log(x))
```

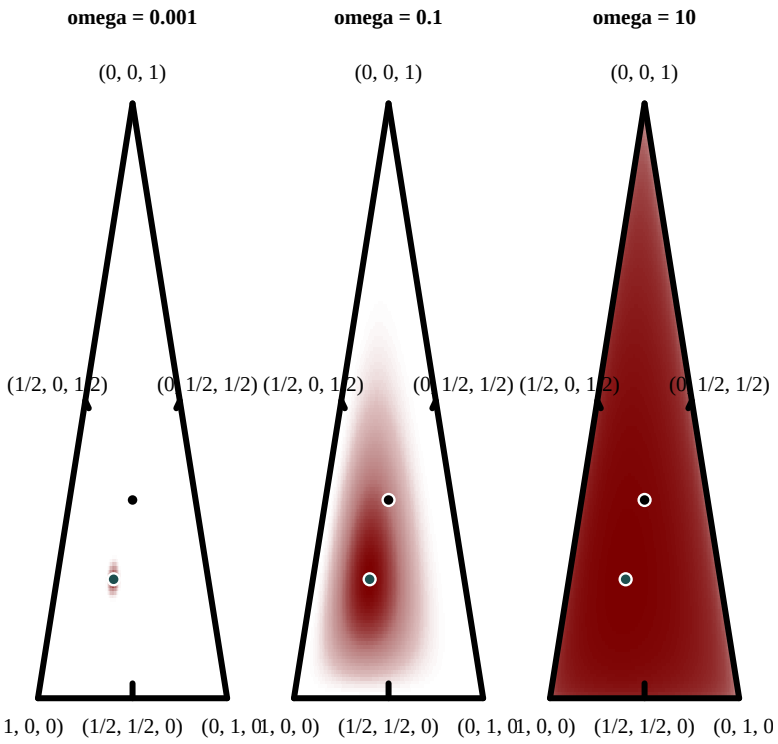
```
  }
}

dirichlet_pdf <- function(q, alpha) {
  exp(  lmult(q[1], alpha[1] - 1)
        + lmult(q[2], alpha[2] - 1)
        + lmult(q[3], alpha[3] - 1)
        + lgamma(alpha[1] + alpha[2] + alpha[3])
        - lgamma(alpha[1]) - lgamma(alpha[2]) - lgamma(alpha[3]) )
}
```

```
to_simplex_config <- function(x, y, C) {
  c((1 - x / C - y) / 2, (1 + x / C - y) / 2, y)
}

to_plot_coordinates <- function(q, C) {
  c(C * (q[2] - q[1]), q[3])
}

valid_simplex_config <- function(q) {
  q[1] >= 0 & q[2] >= 0 & q[3] >= 0 & q[1] + q[2] + q[3] <= 1 + 1e-8
}

plot_simplex_border <- function(label_cex, C, center_label=TRUE) {
  lines( c(-C, 0), c(0, 1), lwd=3)
  lines( c(+C, 0), c(0, 1), lwd=3)
  lines( c(-C, +C), c(0, 0), lwd=3)

  text_delta <- 0.05
  text( 0, 1 + text_delta, "(0, 0, 1)", cex=label_cex)
  text(-C - text_delta, -text_delta, "(1, 0, 0)", cex=label_cex)
  text(+C + text_delta, -text_delta, "(0, 1, 0)", cex=label_cex)

  tick_delta <- 0.025
  lines( c(0, 0), c(0, tick_delta), lwd=3)
  text(0, 0 - text_delta, "(1/2, 1/2, 0)", cex=label_cex)

  lines( c(+C * 0.5, +C * 0.5 - tick_delta * 0.5 * sqrt(3)),
         c(0.5, 0.5 - tick_delta * 0.5), lwd=3)
  text(C * 0.5 + text_delta * 0.5 * sqrt(3) + 2.5 * text_delta,
       0.5 + text_delta * 0.5, "(0, 1/2, 1/2)", cex=label_cex)
```

```
  lines( c(-C * 0.5, -C * 0.5 + tick_delta * 0.5 * sqrt(3)),
         c(0.5, 0.5 - tick_delta * 0.5), lwd=3)
  text(-C * 0.5 - text_delta * 0.5 * sqrt(3) - 2.5 * text_delta,
       0.5 + text_delta * 0.5, "(1/2, 0, 1/2)", cex=label_cex)

  points(0, 1/3, col="white", pch=16, cex=1.5)
  points(0, 1/3, col="black", pch=16, cex=1)
  if (center_label)
    text(0, 1/3 - 1.5 * text_delta, "(1/3, 1/3, 1/3)", cex=label_cex)
}
```

```
plot_dirichlet <- function(omega, q_baseline, label_cex=1, main="") {
  N <- 200
  C <- 1 / sqrt(3)
  xs <- seq(-C - 0.2, C + 0.2, (2 * C + 0.4) / (N - 1))
  ys <- seq(0 - 0.1, 1 + 0.1, 1.2 / (N - 1))
  zs <- matrix(0, nrow=N, ncol=N)

  for (n in 1:N) {
    for (m in 1:N) {
      q <- to_simplex_config(xs[n], ys[m], C)

      if (valid_simplex_config(q)) {
        zs[n, m] <- dirichlet_pdf(q, q_baseline / omega + c(1, 1, 1))
      } else {
        zs[n, m] <- NA
      }
    }
  }

  par(mar = c(0, 0, 2, 0))
  image(xs, ys, zs, col=rev(cont_colors), axes=FALSE, ann=FALSE)
  title(main)

  plot_simplex_border(label_cex, C, FALSE)

  xy <- to_plot_coordinates(q_baseline, C)
  points(xy[1], xy[2], col="white", pch=16, cex=1.5)
  points(xy[1], xy[2], col=util$c_dark_teal, pch=16, cex=1)
}
```

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

q_baseline <- c(0.5, 0.3, 0.2)

plot_dirichlet(1e-3, q_baseline, main="omega = 0.001")
plot_dirichlet(1e-1, q_baseline, main="omega = 0.1")
plot_dirichlet(1e1,  q_baseline, main="omega = 10")
```



5.1.2 Implementation

The Dirichlet population model is straightforward to implement in Stan. Indeed we have to make only a few changes from the independent heterogeneity model.

That said I want to introduce one additional modification here. The independent heterogeneity analyses used a uniform Dirichlet prior model for all of the dice. In these hierarchical analyses, however, we'll use a more informative prior model to demonstrate how we might develop a population prior model in practice.

```
fit <- stan(file="stan_programs/hierarchical_simplex_prior_1.stan",
            algorithm="Fixed_param", seed=8438338,
```

```
            warmup=0, iter=1024, refresh=0)

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

A principled population prior model requires that we elicit domain expertise about both the baseline simplex configuration and the heterogeneity across the individual simplex configurations. For this demonstration let's say that we've elicited the crude thresholds

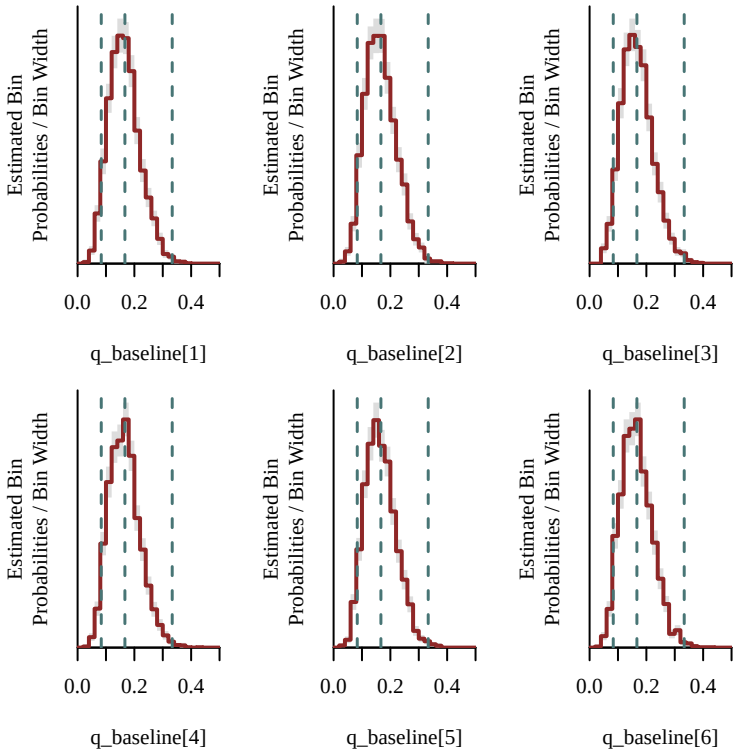
$$\left(\frac{1}{1+1}\right)\frac{1}{6} \lessapprox q_{\text{baseline},k} \lessapprox \left(1+1\right)\frac{1}{6}$$

and

$$\left(\frac{1}{1+2.5}\right)\frac{1}{6} \lessapprox q_{d,k} \lessapprox \left(1+2.5\right)\frac{1}{6}$$

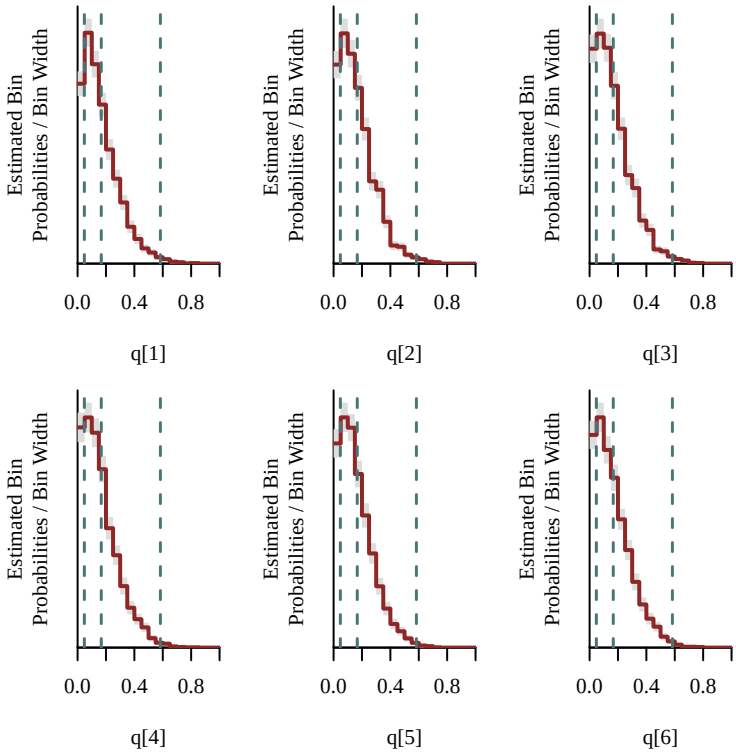
Any domain expertise about the baseline simplex configuration, the point around which the individual simplex behaviors concentrate, informs a prior model for q_{baseline}.

```
par(mfrow=c(2, 3), mar=c(4, 4, 1, 0.5))

for (k in 1:K) {
  name <- paste0("q_baseline[", k, "]")
  util$plot_expectand_pushforward(samples[[name]], 25,
                                  display_name=name,
                                  flim=c(0, 0.5))
  abline(v=1/6 / (1 + 1), col=util$c_mid_teal, lwd=1.5, lty=2)
  abline(v=1/6,           col=util$c_mid_teal, lwd=1.5, lty=2)
  abline(v=1/6 * (1 + 1), col=util$c_mid_teal, lwd=1.5, lty=2)
}
```



The excess variation in the reasonable individual simplex configurations then implicitly informs a prior model for ω.

```
par(mfrow=c(2, 3), mar=c(4, 4, 1, 0.5))

for (k in 1:K) {
  name <- paste0("q[", k, "]")
  util$plot_expectand_pushforward(samples[[name]], 20,
                                  display_name=name,
                                  flim=c(0, 1))
  abline(v=1/6 / (1 + 2.5), col=util$c_mid_teal, lwd=1.5, lty=2)
  abline(v=1/6,             col=util$c_mid_teal, lwd=1.5, lty=2)
  abline(v=1/6 * (1 + 2.5), col=util$c_mid_teal, lwd=1.5, lty=2)
}
```



With an informative prior model established we can incorporate the observed data and estimate posterior inferences.

```
fit <- stan(file="stan_programs/hierarchical_simplex_1.stan",
            data=data, seed=8438338,
            warmup=1000, iter=2024, refresh=0)
```

There are no signs of inaccurate posterior quantification.

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

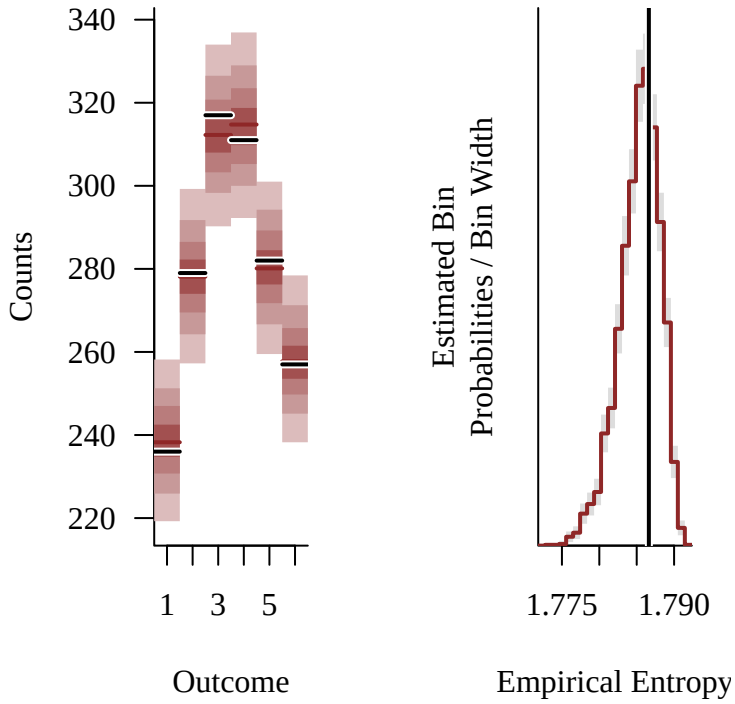
```
  All Hamiltonian Monte Carlo diagnostics are consistent with reliable
Markov chain Monte Carlo.
```

```
samples1 <- util$extract_expectands(fit)
base_samples <- util$filter_expectands(samples1,
                                       c('q', 'q_baseline', 'omega'),
                                       check_arrays=TRUE)
util$check_all_expectand_diagnostics(base_samples)
```

```
All expectands checked appear to be behaving well enough for reliable
Markov chain Monte Carlo estimation.
```

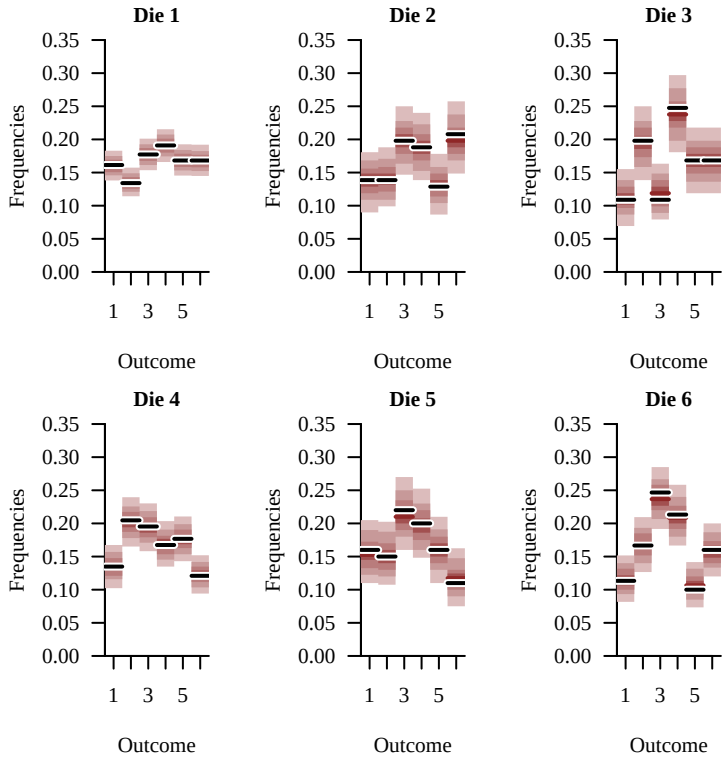
At this point we have a lot of retrodictive checks to consider. We start with those sensitive to homogeneous behavior in the observed data, all of which look clean.

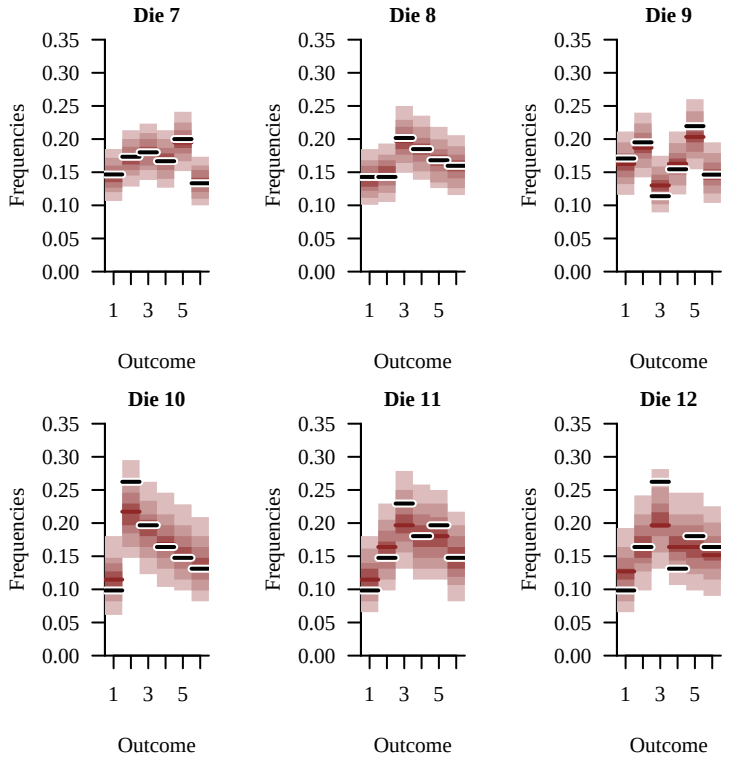
```
par(mfrow=c(1, 2), mar=c(4, 5, 2, 1))

obs_counts <- table(data$outcome)

pred_names <- sapply(1:6, function(k) paste0('pred_counts[', k, ']'))
util$plot_disc_pushforward_quantiles(samples1, pred_names,
                                     baseline_values=obs_counts,
                                     xlab="Outcome", ylab="Counts")

obs_entropy <- sum(sapply(obs_counts,
                          function(n) -(n / data$N) * log(n / data$N)))

util$plot_expectand_pushforward(samples1[["pred_entropy"]], 20,
                                display_name="Empirical Entropy",
                                baseline=obs_entropy)
```



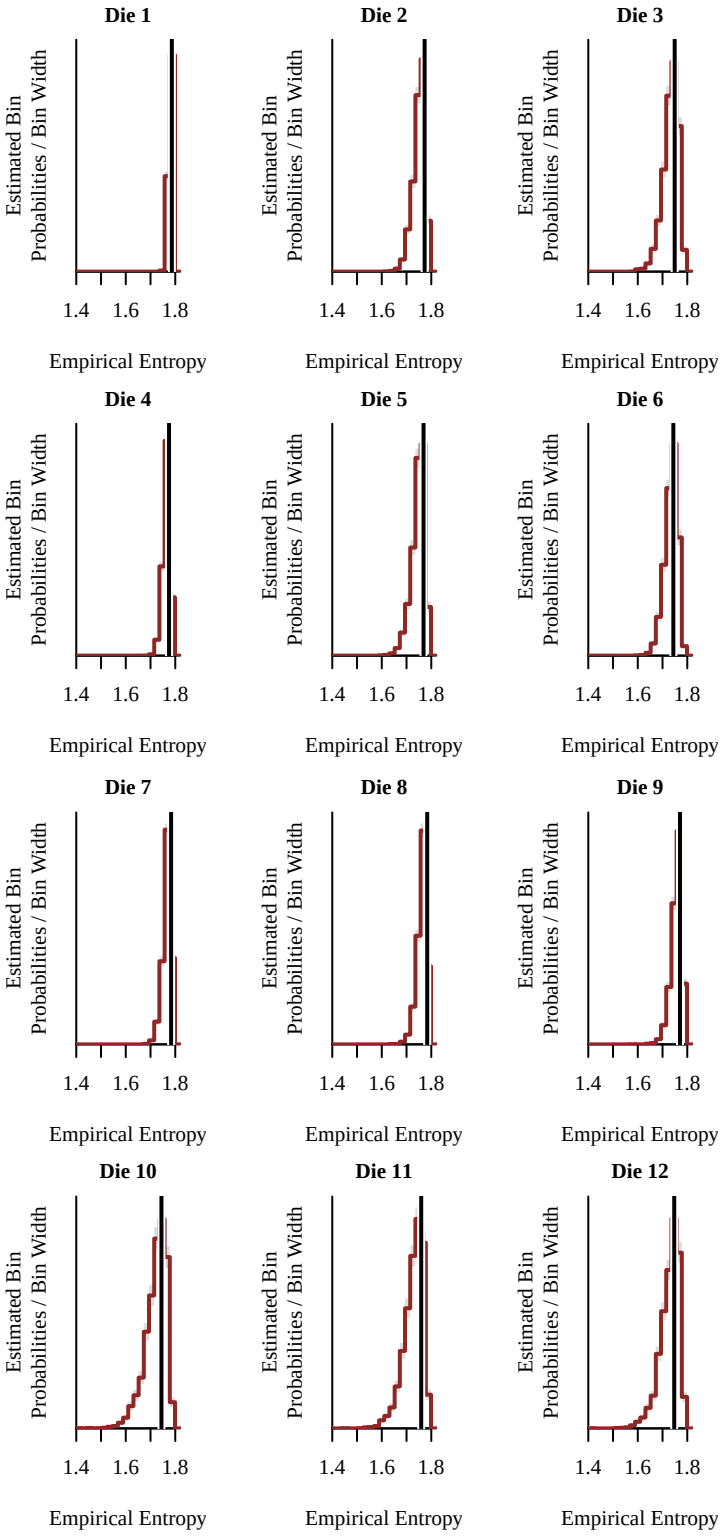
Next we investigate the retrodictive checks that isolate heterogeneous behavior in the observed data. Once again everything look reasonable.

```
for (d in 1:data$N_dice) {
  if (d %% 6 == 1) par(mfrow=c(2, 3), mar=c(4, 5, 2, 1))

  obs_counts <- table(data$outcome[data$die_idxs == d])
  obs_freq <- obs_counts / sum(obs_counts)

  pred_names <- sapply(1:6,
                       function(k) paste0('pred_freq_die[', d, ',', k, ']'))
  util$plot_disc_pushforward_quantiles(samples1, pred_names,
                                       baseline_values=obs_freq,
                                       xlab="Outcome", ylab="Frequencies",
                                       display_ylim=c(0, 0.35),
                                       main=paste("Die", d))
}
```

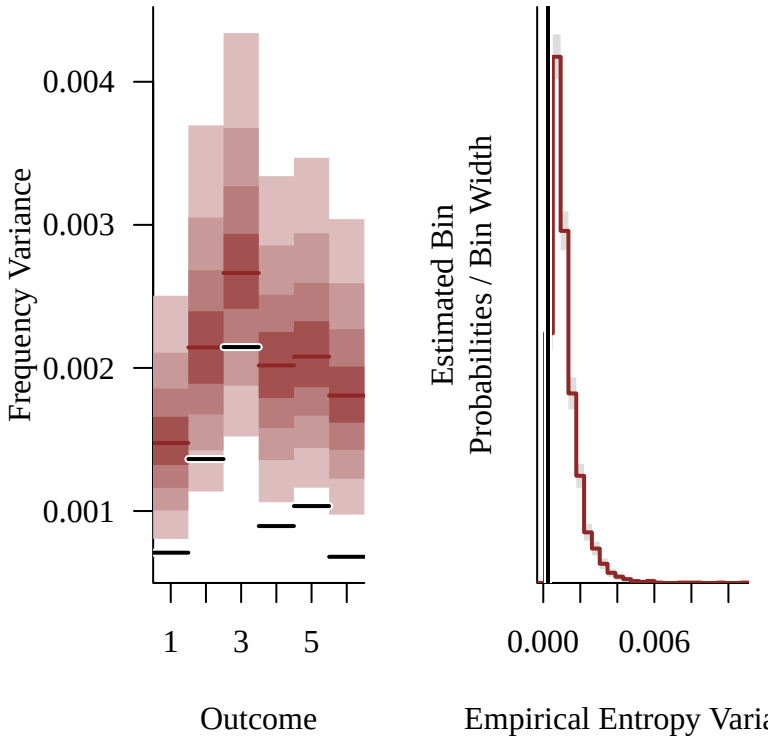





```
for (d in 1:data$N_dice) {
  if (d %% 6 == 1) par(mfrow=c(2, 3), mar=c(4, 5, 2, 1))

  obs_entropy <- compute_empirical_entropy(data$outcome[data$die_idxs == d])

  name <- paste0('pred_entropy_die[', d, ']')
  util$plot_expectand_pushforward(samples1[[name]], 20, flim=c(1.4, 1.82),
                                  display_name="Empirical Entropy",
                                  baseline=obs_entropy, main=paste("Die", d))
}
```


Die 1
Die 2
Die 3
Die 4
Die 5
Die 6
Die 7
Die 8
Die 9
Die 10
Die 11
Die 12
Estimated Bin Probabilities / Bin Width
1.4
1.6
1.8
Empirical Entropy

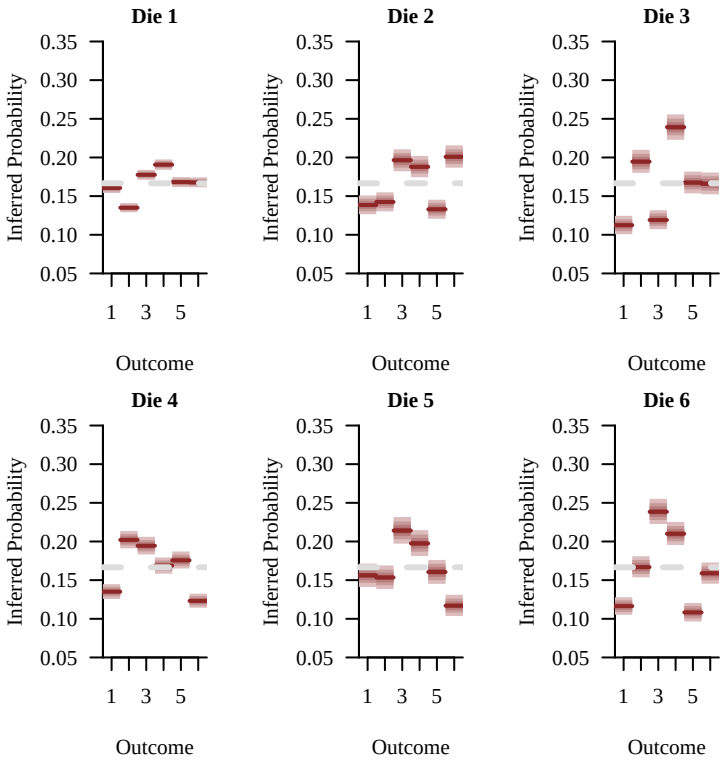
```
par(mfrow=c(1, 2), mar=c(4, 4, 1, 0.5))

obs_var_freq <- sapply(1:6, function(k)
  var(sapply(1:data$N_dice,
             function(d) compute_freq(k, d))))

pred_names <- sapply(1:6, function(k) paste0('pred_var_freq[', k, ']'))
util$plot_disc_pushforward_quantiles(samples1, pred_names,
                                     baseline_values=obs_var_freq,
                                     xlab="Outcome",
                                     ylab="Frequency Variance")

obs_var_entropy <- var(sapply(1:data$N_dice,
                              function(d) compute_empirical_entropy(die_data(d))))

util$plot_expectand_pushforward(samples1[['pred_var_entropy']], 25,
                                display_name="Empirical Entropy Variance",
                                baseline=obs_var_entropy)
```

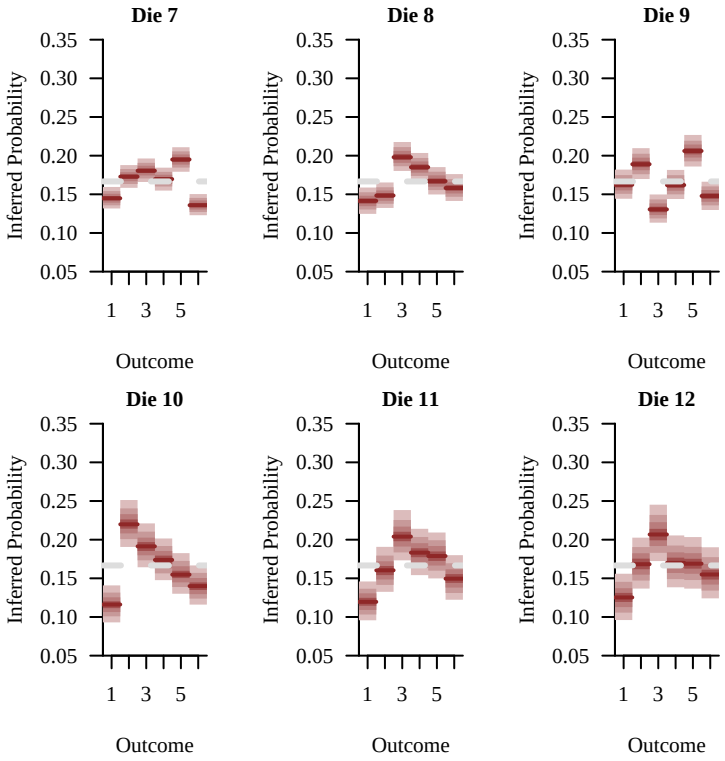


Without any indication that our modeling assumptions are inadequate we can move on to analyzing posterior inferences. Each die exhibits its own ideosyncratic behavior, but none are

too far off from uniformity.

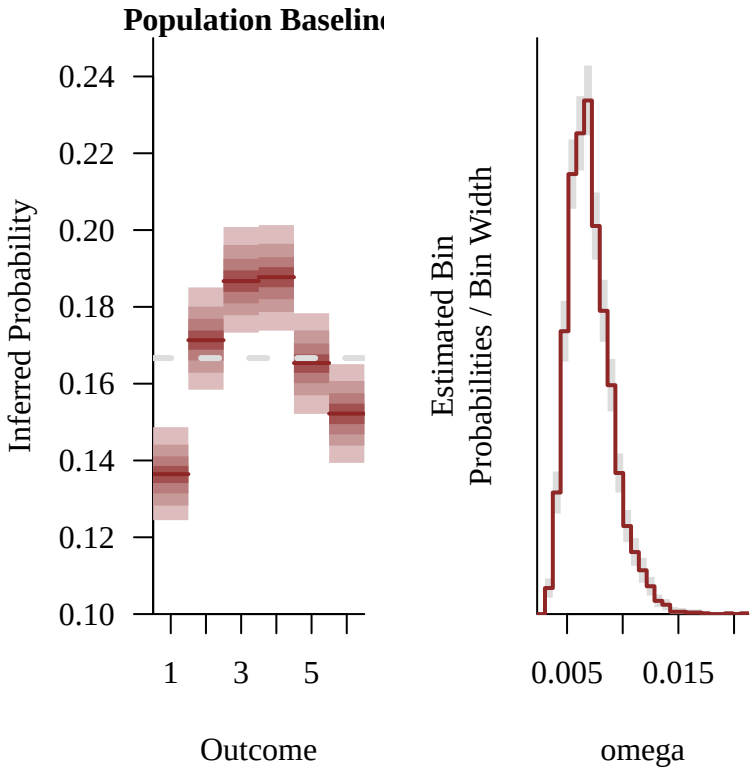
```
for (d in 1:data$N_dice) {
  if (d %% 6 == 1) par(mfrow=c(2, 3), mar=c(4, 5, 2, 1))

  names <- sapply(1:6, function(k) paste0('q[', d, ',', k, ']'))
  util$plot_disc_pushforward_quantiles(samples1, names,
                                       xlab="Outcome",
                                       ylab="Inferred Probability",
                                       display_ylim=c(0.05, 0.35),
                                       main=paste("Die", d))
  abline(h=1/6, col="#DDDDDD", lwd=3, lty=2)
}
```





The heterogeneity in the die behavior is really characterized in the inferred population behavior.

```
par(mfrow=c(1, 2), mar=c(4, 4, 1, 0.5))

names <- sapply(1:6, function(k) paste0('q_baseline[', k, ']'))
util$plot_disc_pushforward_quantiles(samples1, names,
                                     xlab="Outcome",
                                     ylab="Inferred Probability",
                                     display_ylim=c(0.10, 0.25),
                                     main="Population Baseline")
abline(h=1/6, col="#DDDDDD", lwd=3, lty=2)

util$plot_expectand_pushforward(samples1[['omega']], 25,
                                display_name="omega")
```

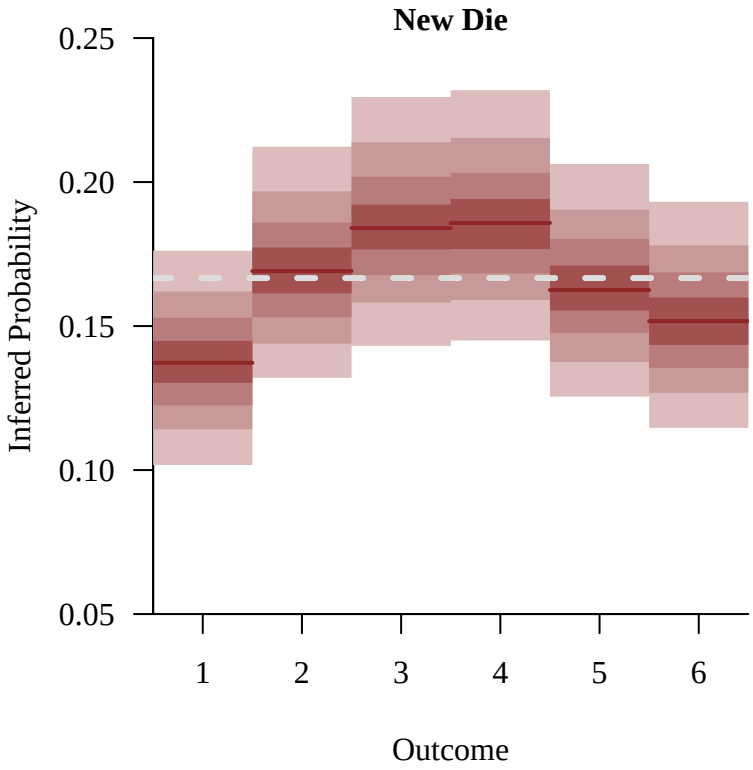


For example here we see that the consistent baseline simplex behaviors are very similar to what we observed with the homogeneous model. Interestingly our inferences are inconsistent with exact homogeneity, with the posterior distribution suppressing values of ω near zero. That said the posterior distribution doesn't stray too far away from $\omega = 0$, concentrating on very small values of ω.

In other words while our inferences disfavor exact homogeneity between the dice the heterogeneities they allow are exceedingly weak. They are likely to be weak enough to not be relevant in practice, but to confirm that we would have to implement an appropriate decision theory analysis.

Finally we can use our population model to simulate the behavior of new, hypothetical dice.

```
par(mfrow=c(1, 1), mar=c(4, 4, 1, 0.5))

names <- sapply(1:6, function(k) paste0('q_new[', k, ']'))
util$plot_disc_pushforward_quantiles(samples1, names,
                                     xlab="Outcome",
                                     ylab="Inferred Probability",
                                     display_ylim=c(0.05, 0.25),
                                     main="New Die")
abline(h=1/6, col="#DDDDDD", lwd=3, lty=2)
```



For example we could use this to model the dice that weren't included in the data collection but might be used by others in the future. We could even incorporate these inferences into our previous decision analyses to study how other dice might fare in particular games.

5.2 Algebraic Population Model

Another strategy for developing hierarchical population models takes advantage of algebraic operations on the model configuration space. Given a family of operations that transforms one individual behavior into another we can construct a hierarchical population model by specifying a baseline behavior and a probability distribution over those transformations.

For example when modeling real-valued behaviors we can take advantage of the addition operation. Given a baseline behavior $\mu \in \mathbb{R}$, individual deviations $\delta_d \in \mathbb{R}$, and a population model $p(\delta_d)$ we can derive the individual behaviors

$$\theta_d = \mu + \delta_d.$$

Unfortunately standard addition is not a valid algebraic operator on a simplex because of the constraints. There are a variety of candidate algebraic operations that do respect the simplex constraints, but all of them carry some form of conceptual if not mathematical baggage. Life in a constrained space is always a bit of a challenge.

5.2.1 Convex Simplex Addition

In general adding the components of two points on a simplex does not yield another simplex because there's no guarantee that the output components will satisfy the simplex constraints. For example adding the components of the two simplex configurations

$$\begin{aligned} q_1 &= (0.1, 0.1, 0.5, 0.1, 0.1, 0.1) \\ q_2 &= (0.05, 0.1, 0.6, 0.1, 0.1, 0.05) \end{aligned}$$

together gives

$$x = q_1 + q_2 = (0.15, 0.2, 1.1, 0.2, 0.2, 0.15).$$

The third output component violates the unit interval constraint,

$$x_3 = q_{1,3} + q_{2,3} = 1.1 > 1,$$

and the summed outputs violate the normalization constraint,

$$\sum k = 1^K x_k = \sum_{k=1}^{K} q_{1,k} + q_{2,k} = 2.$$

One way to maintain the simplex constraints when adding input components together is to carefully weight them. **Convex addition** weights the inputs with interval-valued coefficients that sum to one,

$$q_3 = (1 - \lambda)\, q_1 + \lambda\, q_2$$

with

$$0 \leq \lambda \leq 1.$$

If q_1 and q_2 are both simplices then

$$\begin{aligned} q_{3,k} &= (1 - \lambda)\, q_{1,k} + \lambda\, q_{2,k} \\ &\geq (1 - \lambda)\, 0 + \lambda\, 0 \\ &\geq 0, \end{aligned}$$

$$\begin{aligned} q_{3,k} &= (1 - \lambda)\, q_{1,k} + \lambda\, q_{2,k} \\ &\leq (1 - \lambda)\, 1 + \lambda\, 1 \\ &\leq 1 - \lambda + 1 \\ &\leq 1, \end{aligned}$$

and

$$\begin{aligned}
\sum_{k=1}^{K} q_{3,k} &= \sum_{k=1}^{K} (1 - \lambda)\, q_{1,k} + \lambda\, q_{2,k} \\
&= (1 - \lambda) \sum_{k=1}^{K} q_{1,k} + \lambda \sum_{k=1}^{K} q_{2,k} \\
&= (1 - \lambda)\, 1 + \lambda\, 1 \\
&= (1 - \lambda) + \lambda \\
&= 1.
\end{aligned}$$

In other words q_3 will also be a simplex.

Consequently we can build up consistent consistent hierarchical population model over a simplex space by convex adding deviations to a baseline simplex,

$$\begin{aligned}
q_d &= (1 - \lambda)\, q_{\text{baseline}} + \lambda\, \delta_d \\
p(\delta_d) &= \text{dirichlet}(\delta_d \mid 1, \ldots, 1).
\end{aligned}$$

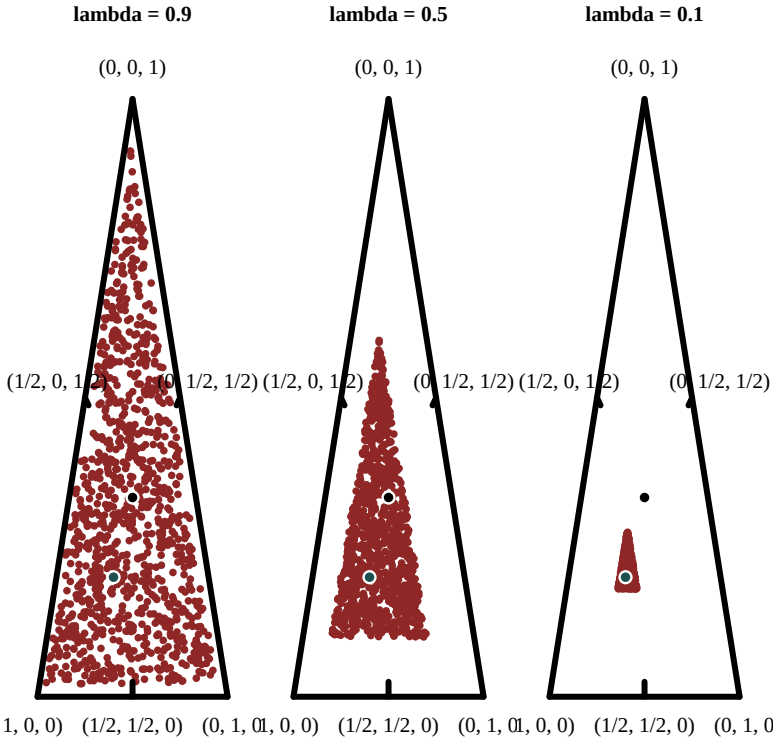
The larger λ is the closer each q_d will be to the uniformly distributed deviation δ_d, effectively decoupling the individual behaviors from each other. On the other hand when λ approaches zero all of the q_d will collapse towards q_{baseline}, coupling the individual behaviors together. This suggests that λ quantifies the strength of the population heterogeneity, albeit in a different way than ω does for the Dirichlet population model.

We can see these behaviors directly if we simulate individual simplex behaviors from the convex addition population model for increasing values of λ.

```
plot_individual_samples <- function(q_baseline, lambda, label_cex=1, main="") {
  C <- 1 / sqrt(3)

  par(mar = c(0, 0, 2, 0))
  plot(0, type='n', main=main, axes=FALSE,
       xlim=c(-C - 0.2, C + 0.2),
       ylim=c(0 - 0.1, 1 + 0.1))

  S <- 1000
  for (s in 1:S) {
    delta <- rgamma(3, 1, 1)
    delta <- delta / sum(delta)

    q <- (1 - lambda) * q_baseline + lambda * delta
    if (valid_simplex_config(q)) {
```

```
      xy <- to_plot_coordinates(q, C)
      points(xy[1], xy[2], col=util$c_dark, pch=16, cex=0.8)
    }
  }

  plot_simplex_border(label_cex, C, FALSE)

  xy <- to_plot_coordinates(q_baseline, C)
  points(xy[1], xy[2], col="white", pch=16, cex=1.5)
  points(xy[1], xy[2], col=util$c_dark_teal, pch=16, cex=1)
}
```

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

q_baseline <- c(0.5, 0.3, 0.2)

plot_individual_samples(q_baseline, 0.9, main="lambda = 0.9")
plot_individual_samples(q_baseline, 0.5, main="lambda = 0.5")
plot_individual_samples(q_baseline, 0.1, main="lambda = 0.1")
```



Notice that the concentration around q_{baseline} isn't continuous. Instead increasing λ removes

the closer and closer simplex configurations entirely.

We can make comparison to the Dirichlet population model a bit direct by working with not λ itself but rather the ratio

$$\zeta = \frac{\lambda}{1 - \lambda}.$$

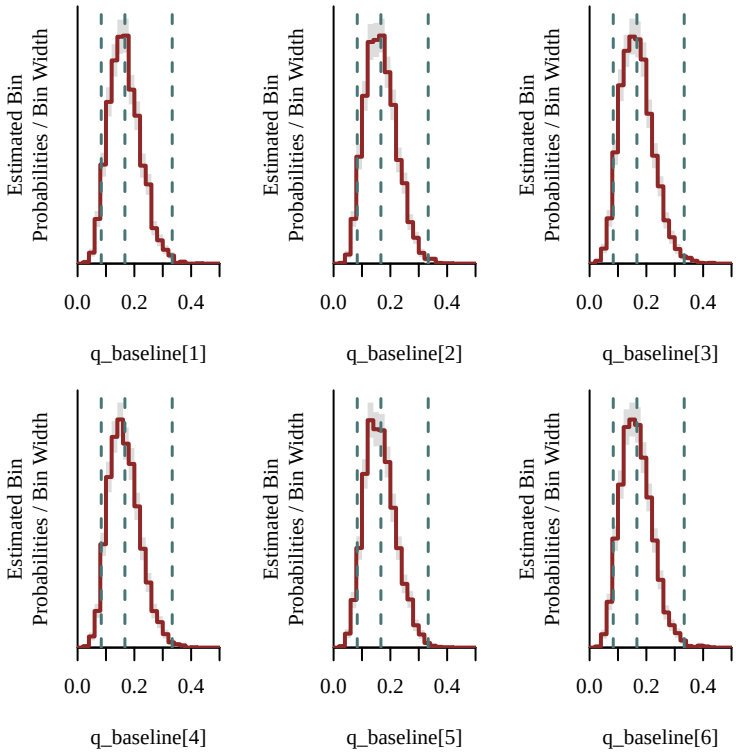
In this case the homogeneous model is given by $\zeta = 0$ while the independent heterogeneous model is given by $\zeta = \infty$, similar to the influence of ω in the Dirichlet population model. That said while the limiting behavior is the same we cannot, in general, expect that equal intermediate values of ω and ζ will correspond to exactly the same kind of heterogeneity.

5.2.2 Implementation

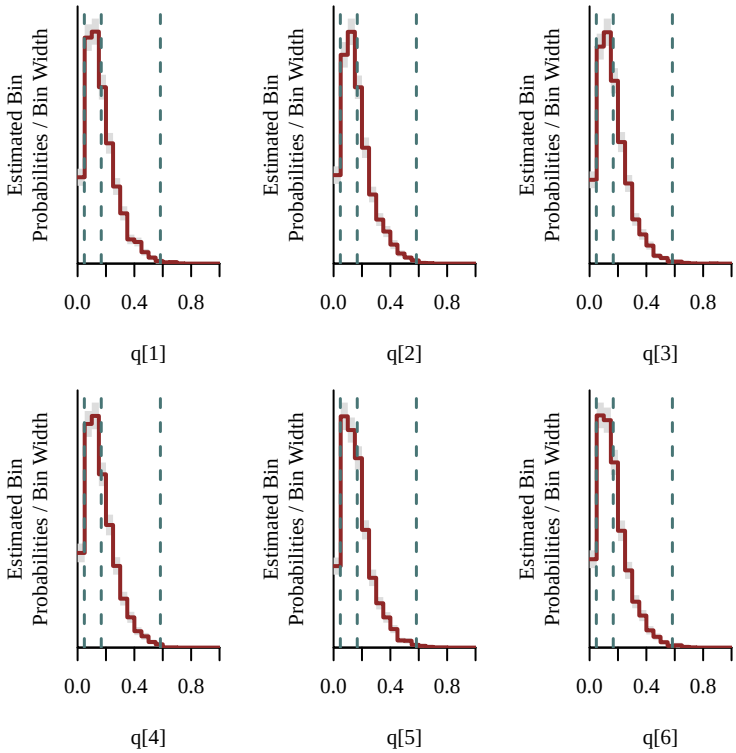
Before considering the data we need to make sure that our population prior model is at least roughly consistent with the hypothetically-elicited thresholds for the baseline and individual simplex component behaviors.

```
fit <- stan(file="stan_programs/hierarchical_simplex_prior_2.stan",
            algorithm="Fixed_param", seed=8438338,
            warmup=0, iter=1024, refresh=0)

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

```
par(mfrow=c(2, 3), mar=c(4, 4, 1, 0.5))

for (k in 1:K) {
  name <- paste0("q_baseline[", k, "]")
  util$plot_expectand_pushforward(samples[[name]], 25,
                                  display_name=name,
                                  flim=c(0, 0.5))
  abline(v=1/6 / (1 + 1), col=util$c_mid_teal, lwd=1.5, lty=2)
  abline(v=1/6,           col=util$c_mid_teal, lwd=1.5, lty=2)
  abline(v=1/6 * (1 + 1), col=util$c_mid_teal, lwd=1.5, lty=2)
}
```




```
par(mfrow=c(2, 3), mar=c(4, 4, 1, 0.5))

for (k in 1:K) {
  name <- paste0("q[", k, "]")
  util$plot_expectand_pushforward(samples[[name]], 20,
                                  display_name=name,
                                  flim=c(0, 1))
  abline(v=1/6 / (1 + 2.5), col=util$c_mid_teal, lwd=1.5, lty=2)
  abline(v=1/6,             col=util$c_mid_teal, lwd=1.5, lty=2)
  abline(v=1/6 * (1 + 2.5), col=util$c_mid_teal, lwd=1.5, lty=2)
}
```



We can now build our posterior inferences on top of this principled prior foundation.

```
fit <- stan(file="stan_programs/hierarchical_simplex_2.stan",
            data=data, seed=8438338,
            warmup=1000, iter=2024, refresh=0)
```

The computational diagnostics are clean.

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

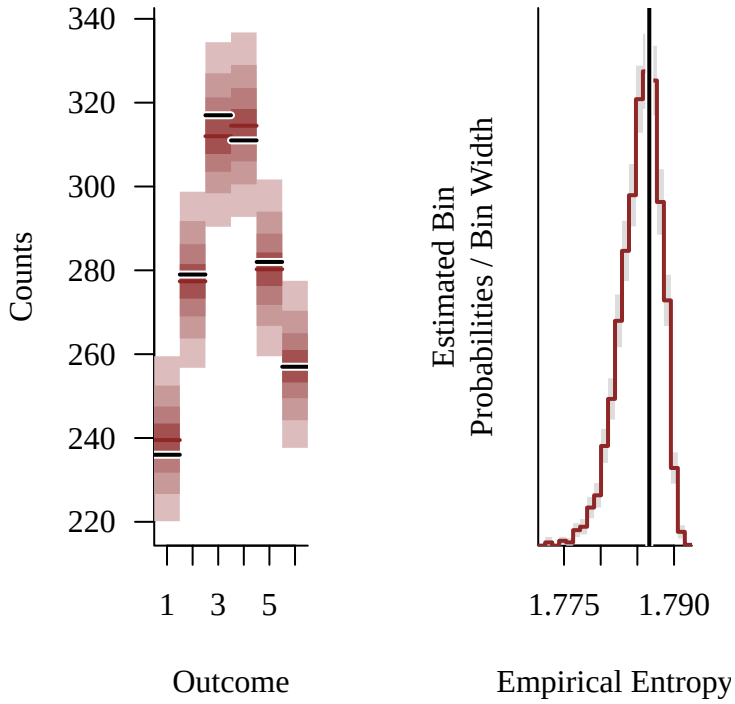
```
  All Hamiltonian Monte Carlo diagnostics are consistent with reliable
Markov chain Monte Carlo.
```

```
samples2 <- util$extract_expectands(fit)
base_samples <- util$filter_expectands(samples2,
                                       c('q_baseline', 'delta', 'zeta'),
                                       check_arrays=TRUE)
util$check_all_expectand_diagnostics(base_samples)
```

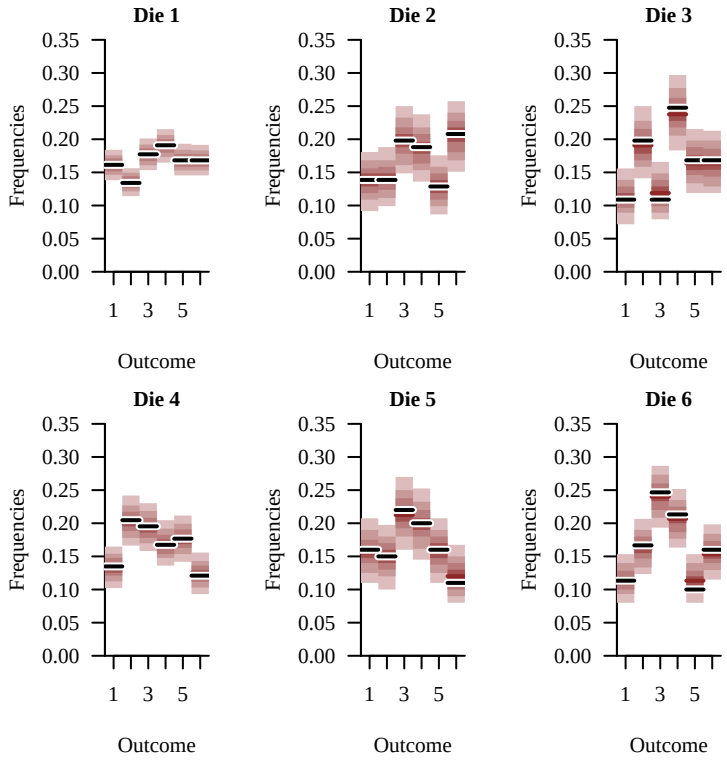
```
All expectands checked appear to be behaving well enough for reliable
Markov chain Monte Carlo estimation.
```

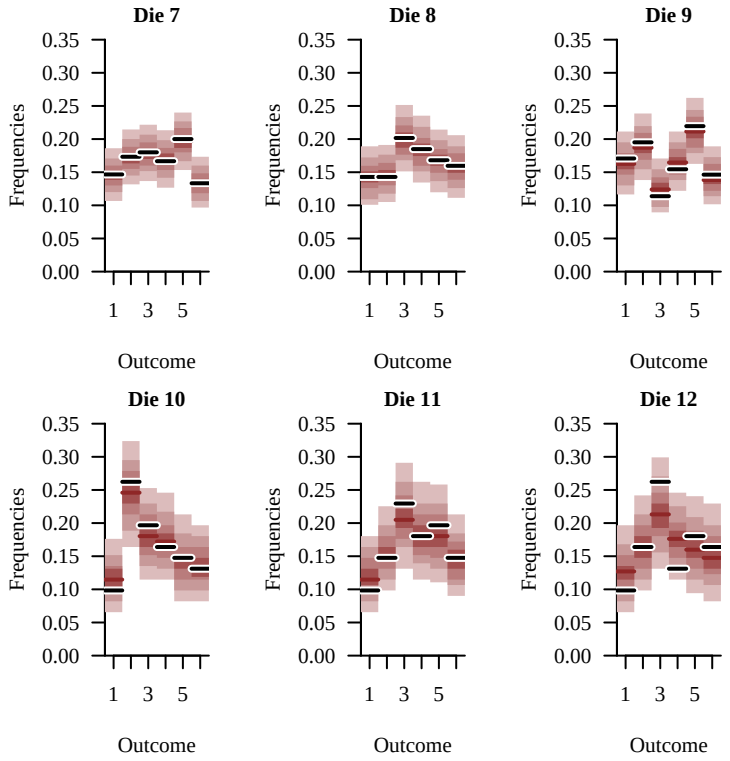
With trustworthy computation we move on to the extensive retrodictive checks, all of which look reasonable.

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

obs_counts <- table(data$outcome)

pred_names <- sapply(1:6, function(k) paste0('pred_counts[', k, ']'))
util$plot_disc_pushforward_quantiles(samples2, pred_names,
                                     baseline_values=obs_counts,
                                     xlab="Outcome", ylab="Counts")

obs_entropy <- sum(sapply(obs_counts,
                          function(n) -(n / data$N) * log(n / data$N)))

util$plot_expectand_pushforward(samples2[["pred_entropy"]], 20,
                                display_name="Empirical Entropy",
                                baseline=obs_entropy)
```

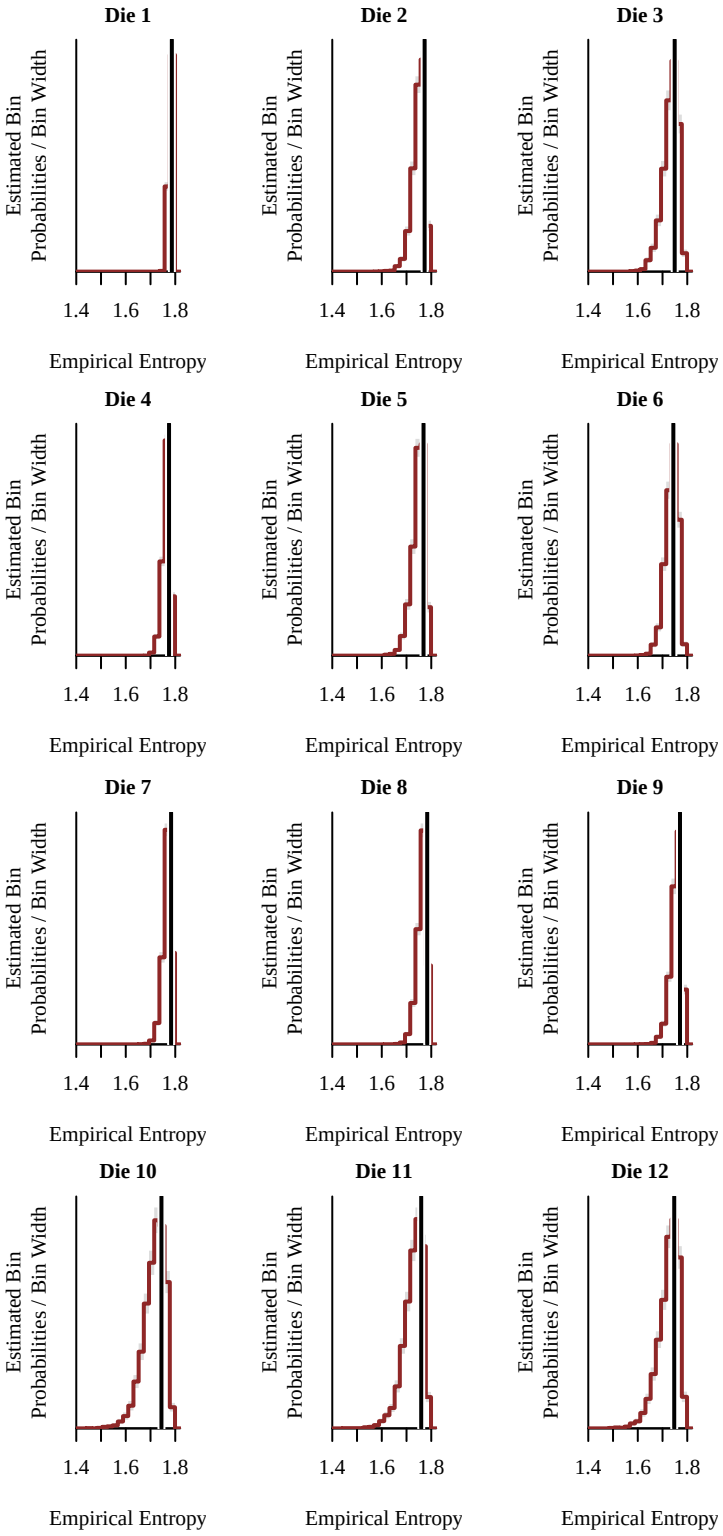


```
for (d in 1:data$N_dice) {
  if (d %% 6 == 1) par(mfrow=c(2, 3), mar=c(4, 5, 2, 1))

  obs_counts <- table(data$outcome[data$die_idxs == d])
  obs_freq <- obs_counts / sum(obs_counts)

  pred_names <- sapply(1:6,
                       function(k) paste0('pred_freq_die[', d, ',', k, ']'))
  util$plot_disc_pushforward_quantiles(samples2, pred_names,
                                       baseline_values=obs_freq,
                                       xlab="Outcome", ylab="Frequencies",
                                       display_ylim=c(0, 0.35),
                                       main=paste("Die", d))
}
```

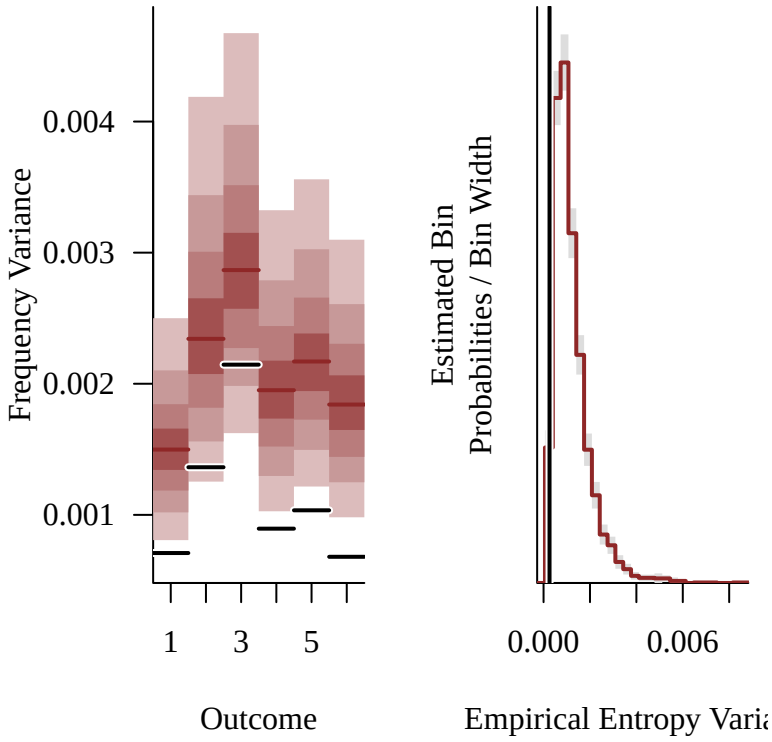





```
for (d in 1:data$N_dice) {
  if (d %% 6 == 1) par(mfrow=c(2, 3), mar=c(4, 5, 2, 1))

  obs_entropy <- compute_empirical_entropy(data$outcome[data$die_idxs == d])

  name <- paste0('pred_entropy_die[', d, ']')
  util$plot_expectand_pushforward(samples2[[name]], 20, flim=c(1.4, 1.82),
                                  display_name="Empirical Entropy",
                                  baseline=obs_entropy, main=paste("Die", d))
}
```


Die 1
Die 2
Die 3
Die 4
Die 5
Die 6
Die 7
Die 8
Die 9
Die 10
Die 11
Die 12
Estimated Bin Probabilities / Bin Width
1.4
1.6
1.8
Empirical Entropy

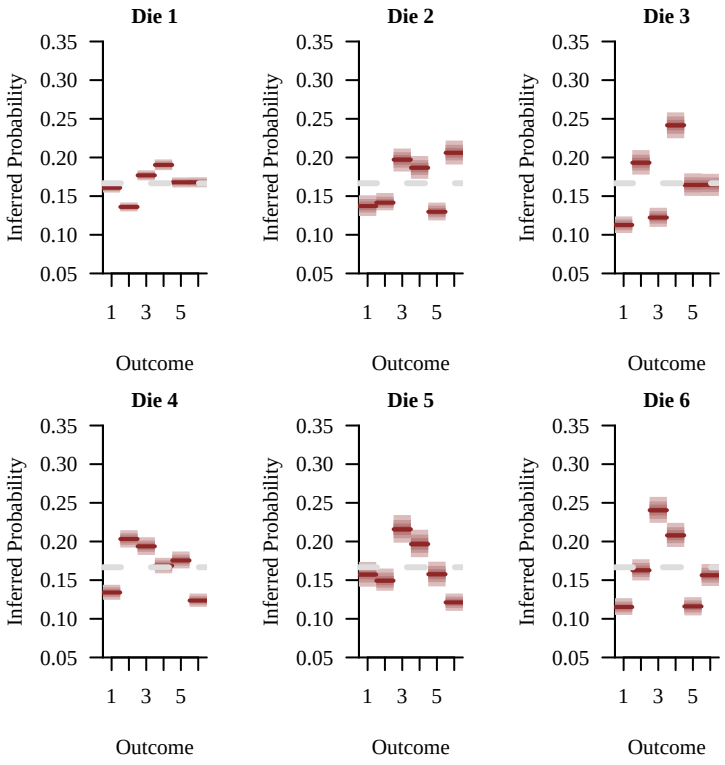
```
par(mfrow=c(1, 2), mar=c(4, 4, 1, 0.5))

obs_var_freq <- sapply(1:6, function(k)
  var(sapply(1:data$N_dice,
             function(d) compute_freq(k, d))))

pred_names <- sapply(1:6, function(k) paste0('pred_var_freq[', k, ']'))
util$plot_disc_pushforward_quantiles(samples2, pred_names,
                                     baseline_values=obs_var_freq,
                                     xlab="Outcome",
                                     ylab="Frequency Variance")

obs_var_entropy <- var(sapply(1:data$N_dice,
                              function(d) compute_empirical_entropy(die_data(d))))

util$plot_expectand_pushforward(samples2[['pred_var_entropy']], 25,
                                display_name="Empirical Entropy Variance",
                                baseline=obs_var_entropy)
```

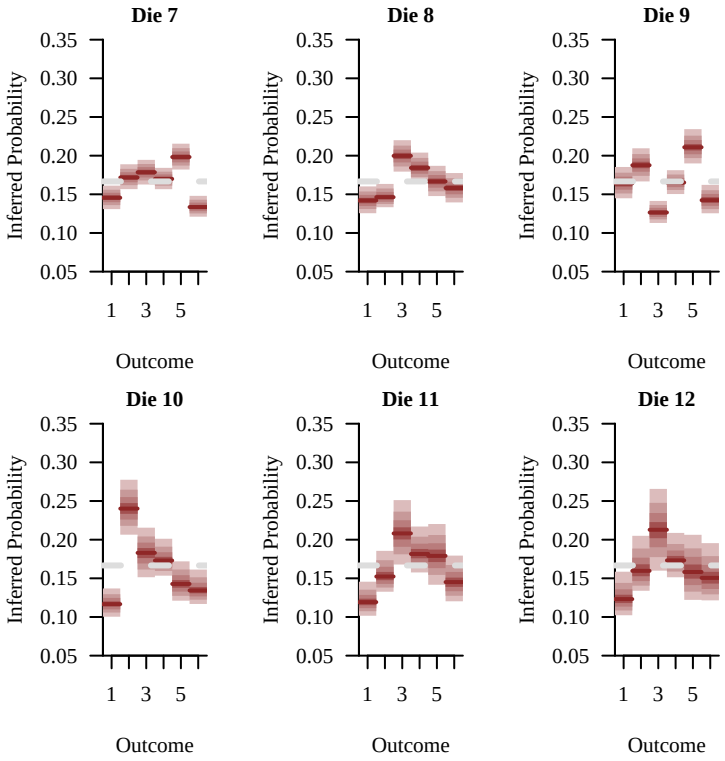


Because our retrodictive checks don't suggest any model inadequacies we now finally ready to investigate our posterior inferences. Each die once again exhibits its own unique, but relatively

weak, deviation from uniformity.

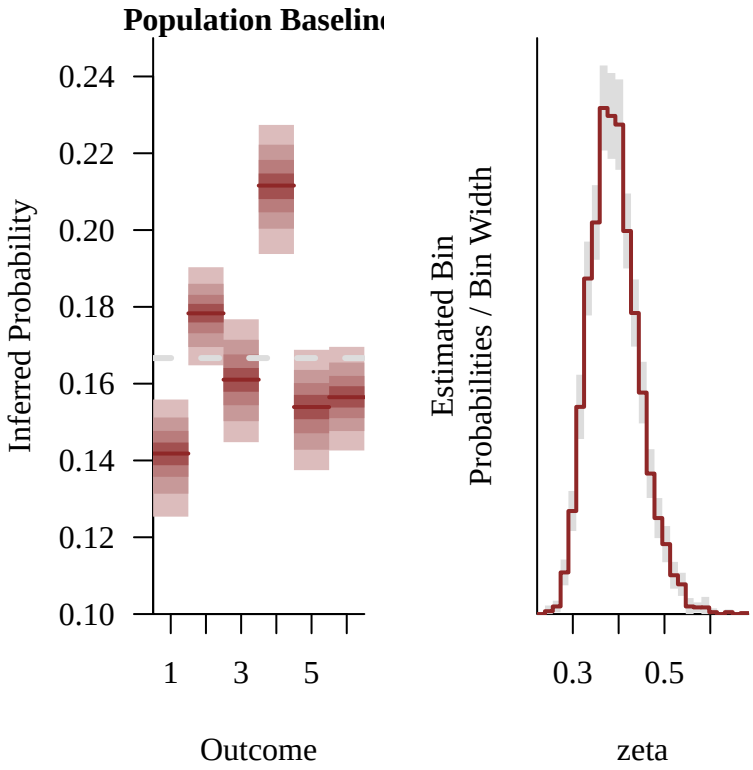
```
for (d in 1:data$N_dice) {
  if (d %% 6 == 1) par(mfrow=c(2, 3), mar=c(4, 5, 2, 1))

  names <- sapply(1:6, function(k) paste0('q[', d, ',', k, ']'))
  util$plot_disc_pushforward_quantiles(samples2, names,
                                       xlab="Outcome",
                                       ylab="Inferred Probability",
                                       display_ylim=c(0.05, 0.35),
                                       main=paste("Die", d))
  abline(h=1/6, col="#DDDDDD", lwd=3, lty=2)
}
```





The inferred population behavior also quantifies these heterogeneities.

```
par(mfrow=c(1, 2), mar=c(4, 4, 1, 0.5))

names <- sapply(1:6, function(k) paste0('q_baseline[', k, ']'))
util$plot_disc_pushforward_quantiles(samples2, names,
                                     xlab="Outcome",
                                     ylab="Inferred Probability",
                                     display_ylim=c(0.10, 0.25),
                                     main="Population Baseline")
abline(h=1/6, col="#DDDDDD", lwd=3, lty=2)

util$plot_expectand_pushforward(samples2[['zeta']], 25,
                                display_name="zeta")
```

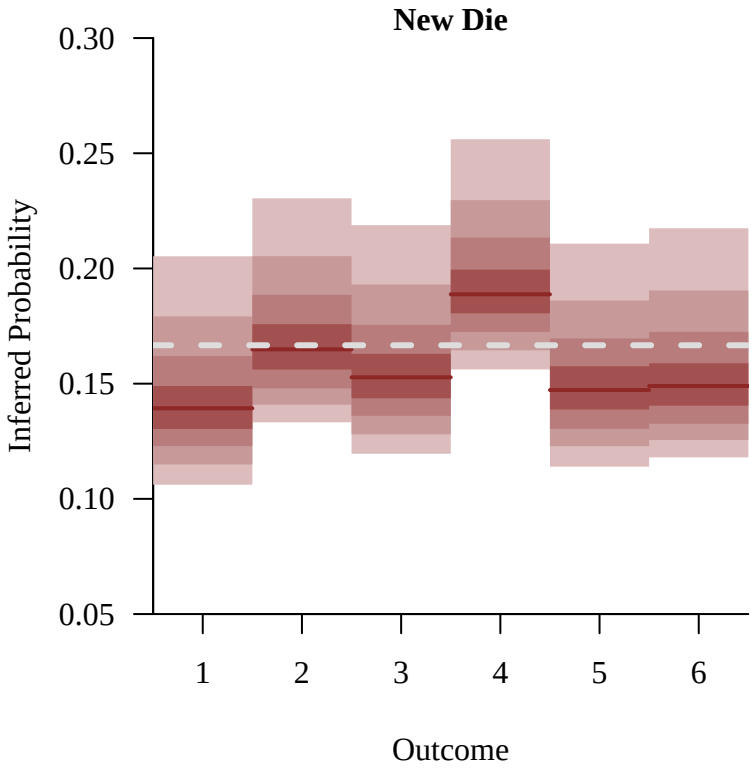


Interestingly the inferences for q_{baseline} in this population model are noticeable different from those in the Dirichlet population model. This is not necessarily surprising, however, given that the baseline simplex configuration isn't used in exactly the same way within each model.

The inferences for ζ are qualitatively similar to the inferences for ω in the Dirichlet population model: homogeneous behaviors with $\zeta = 0$ and stronger heterogeneous behaviors with $\zeta > 1$ are excluded. We cannot directly compare the quantitative inferences for ζ and ω, however, because they encode different notions of heterogeneity.

In order to model the behavior of new, hypothetical dice we need to simulate new simplex configurations from the population model.

```
par(mfrow=c(1, 1), mar=c(4, 4, 1, 0.5))

names <- sapply(1:6, function(k) paste0('q_new[', k, ']'))
util$plot_disc_pushforward_quantiles(samples2, names,
                                     xlab="Outcome",
                                     ylab="Inferred Probability",
                                     display_ylim=c(0.05, 0.3),
                                     main="New Die")
abline(h=1/6, col="#DDDDDD", lwd=3, lty=2)
```



These inferences are much closer to what we saw with the Dirichlet population model. While the hierarchical models were constructed in different ways they result in equivalent inferences for the individual behaviors of both observed and unobserved dice.

5.3 Transformed Population Model

The last strategy that we will consider here is to remove the simplex constraint by transforming the $(K-1)$-simplex to a $(K-1)$-dimensional real space and then model heterogeneity on that unconstrained real space with a multivariate normal population model. There are many possible ways to transform ourselves out of the simplex constraints but, as we will see, not all are as useful for probabilistic modeling as others.

5.3.1 Unconstraining the Simplex

Before attempting to engineer a transformation from a simplex to an unconstrained real space let's first consider moving in the opposite direction, transforming an unconstrained real space to a simplex.

The **softmax function** maps K real numbers to a point on a simplex,

$$\begin{aligned} \sigma : \mathbb{R}^K &\to \Delta^{K-1} \\ (x_1, \ldots, x_K) &\mapsto (q_1, \ldots, q_K) \end{aligned}$$

with

$$q_k = \frac{\exp(x_k)}{\sum_{k'=1}^{K} \exp(x_{k'})}.$$

Unfortunately the softmax function is not injective – any inputs that are translations of each other always map to the same output simplex configuration,

$$\begin{aligned} \sigma(x_1 + \delta, \ldots, x_K + \delta) &= \frac{\exp(x_k + \delta)}{\sum_{k'=1}^{K} \exp(x_{k'} + \delta)} \\ &= \frac{\exp(x_k)\exp(\delta)}{\sum_{k'=1}^{K} \exp(x_{k'})\exp(\delta)} \\ &= \frac{\exp(x_k)}{\sum_{k'=1}^{K} \exp(x_{k'})} \\ &= \sigma(x_1, \ldots, x_K). \end{aligned}$$

Consequently we cannot invert the softmax function to map simplex configurations to unique real-valued inputs.

In hindsight, however, this shouldn't be surprising. Because a $(K-1)$-simplex is a $(K-1)$-dimensional space it can be isomorphic to only other $(K-1)$-dimensional spaces. Specifically we cannot construct bijections between Δ^{K-1} and $\mathbb{R}^K$, but we can construct them between Δ^{K-1} and $\mathbb{R}^{K-1}$

In order to construct an invertible mapping between an unconstrained real space and a simplex we need to constrain the inputs of the softmax function to eliminate a degree of freedom and reduce the dimensionality by one. Formally we can achieve this with a function

$$\begin{aligned} T : \mathbb{R}^{K-1} &\to X \subset \mathbb{R}^K \\ (y_1, \ldots, y_{K-1}) &\mapsto (x_1, \ldots, x_K) \end{aligned}$$

that maps an unconstrained $(K-1)$-dimensional real space to a constrained, $(K-1)$-dimensional subset of a K-dimensional real space. If the constraint is linear then T will be a linear function which we can also implement as a $K \times (K-1)$ matrix with its action given by matrix multiplication,

$$T(y) = \mathbf{T} \cdot \mathbf{y}$$

or in components

$$(\mathbf{T} \cdot \mathbf{y})_k = \sum_{k'=1}^{K-1} T_{kk'} y_{k'}.$$

Composing this transformation with the softmax function

$$\sigma \circ T : \mathbb{R}^{K-1} \to X \subset \mathbb{R}^{K} \to \Delta^{K-1}$$

finally gives a transformation from $K-1$ real numbers to a valid simplex configuration,

$$\begin{aligned} \sigma \circ T : \mathbb{R}^{K-1} &\to \Delta^{K-1} \\ (y_1, \ldots, y_{K-1}) &\mapsto (q_1, \ldots, q_K) \end{aligned}$$

with

$$q_k = \frac{\exp\left(\sum_{k''=1}^{K-1} T_{kk''} y_{k''} \right)}{\sum_{k'=1}^{K} \exp\left(\sum_{k''=1}^{K-1} T_{k'k''} y_{k''} \right)}.$$

Interestingly this construction relates to a statistical technique known as compositional data analysis Appendix B.

Now we can consider building up a transformation that takes a simplex configuration to an unconstrained real space. Given a well-defined constraint we can invert the softmax function to map simplex configurations to constrained real numbers,

$$\begin{aligned} \sigma^{-1} : \Delta^{K-1} &\to X \subset \mathbb{R}^{K} \\ (q_1, \ldots, q_K) &\mapsto (x_1, \ldots, x_K). \end{aligned}$$

To take these constrained real numbers to unconstrained real numbers we can always use the transpose of T, which defines a mapping

$$\begin{aligned} T^{*} : X \subset \mathbb{R}^{K} &\to \mathbb{R}^{K-1} \\ (x_1, \ldots, x_K) &\mapsto (y_1, \ldots, y_{K-1}) \end{aligned}$$

with the action

$$T^{*}(x) = \mathbf{T}^{T} \cdot \mathbf{x}$$

defined by the components

$$(\mathbf{T}^{T} \cdot \mathbf{x})_k = \sum_{k'=1}^{K} T_{k'k} x_{k'}.$$

Our only problem is that the resulting composition

$$T^{*} \circ \sigma^{-1} : \Delta^{K-1} \to X \subset \mathbb{R}^{K} \to \mathbb{R}^{K-1}$$

will not, in general, be an inversion of

$$\sigma \circ T : \mathbb{R}^{K-1} \to X \subset \mathbb{R}^{K} \to \Delta^{K-1}.$$

If we were to use these two inconsistent transformations then information would propagate differently from the simplex to the unconstrained space and back. Consequently we would

have to use different reasoning to translate our domain expertise from the simplex to the unconstrained space than we would to translate modeling assumptions on the unconstrained space to the simplex. This isn't an absolute disaster, but it is really inconvenient and can frustrate principled prior modeling.

The two transformations $T^* \circ \sigma^{-1}$ and $\sigma \circ T$ will be consistent with each other only if the transformation is semi-orthogonal,

$$T^* \circ T = I$$

or, in matrix form,

$$\mathbf{T}^T \cdot \mathbf{T} = \mathbf{I}_{K-1, K-1}.$$

5.3.2 Preserving Probabilistic Structure

Our next challenge is that there are many linear constraints that reduce $\mathbb{R}^K$ to a $(K-1)$-dimensional subspace. Moreover, for a given constraint will there be many different maps T between the constrained subspace and $\mathbb{R}^{K-1}$. Not all of these choices, however, are equally useful.

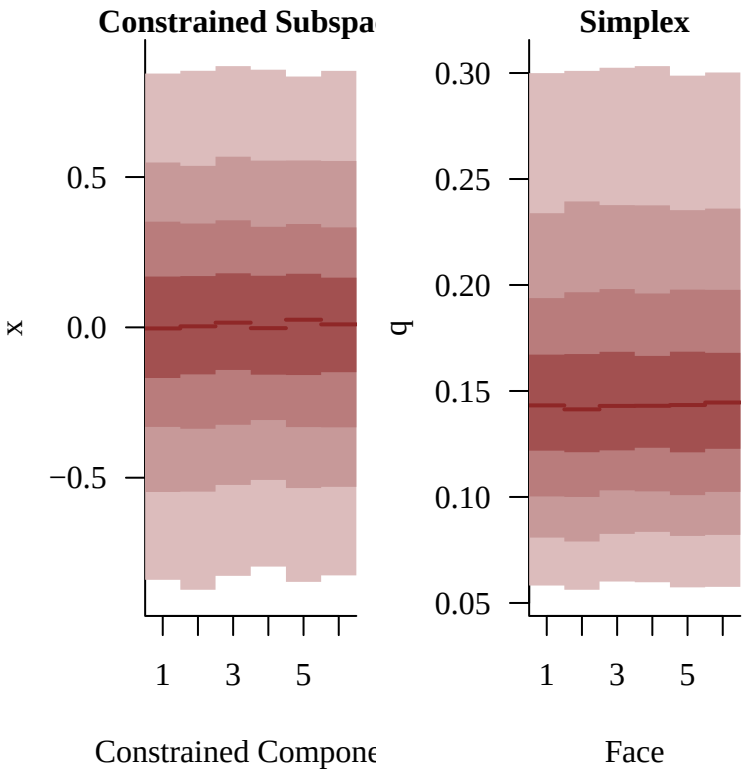
Remember that our goal here is to translate our domain expertise about simplex behaviors to behaviors in an unconstrained space so that we can inform an appropriate multivariate normal hierarchical model. If a latent hierarchical model isn't consistent with our domain expertise then it will push forward to unreasonable behaviors on the simplex. Consequently the most useful transformations are those that facilitate this translation. That, in turn, depends on the nature of the available domain expertise.

Without any knowledge about the manufacturing process we don't have any information to distinguish the behavior of one die face from another. In other words our domain expertise about any particular die is exchangeable with respect to the faces. Any consistent hierarchical model needs to be exchangeable not with respect to not only the individual dice but also the simplex components that model the faces on each of those dice. Unfortunately this is easier said than done.

By itself the softmax function respects exchangeability: permuting the inputs doesn't change the form of the outputs. Because of this a K-dimensional multivariate normal hierarchical model that treats all K components symmetrically pushes forward to an exchangeable probability distribution on the simplex.

```
fit <- stan(file="stan_programs/simu_softmax_1.stan",
            algorithm="Fixed_param", seed=8438338,
            warmup=0, iter=1024, refresh=0)

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

```
par(mfrow=c(1, 2), mar=c(4, 4, 1, 0.5))

names <- sapply(1:6, function(k) paste0('x[', k, ']'))
util$plot_disc_pushforward_quantiles(samples, names,
                                     xlab="Constrained Component", ylab="x",
                                     main="Constrained Subspace")

names <- sapply(1:6, function(k) paste0('q[', k, ']'))
util$plot_disc_pushforward_quantiles(samples, names,
                                     xlab="Face", ylab="q",
                                     main="Simplex")
```



In order for the full mapping from $\mathbb{R}^{K-1}$ to Δ^{K-1} to respect exchangeability we just need to ensure that the function T also respects exchangeability, if not exactly then at least well enough that a symmetric $(K-1)$-dimensional multivariate normal hierarchical model pushes forward to an probability distribution that is exchangeable with respect to the simplex components. If it doesn't then we would need to engineer an asymmetric hierarchical model to induce a symmetric behavior on the simplex and vice versa.

That said I do not know what conditions on T are sufficient to ensure this behavior. I don't even know if straightforward sufficient conditions even exist!

To avoid getting lost in mathematical minutia let's consider instead a more heuristic perspective. A multivariate normal density function on the constrained space X is exchangeable if and only if its mean vector is constant and its covariance matrix can be written as linear combination of the identity matrix $\mathbf{I}_{K,K}$ and square ones matrix $\mathbf{1}_{K,K}$,

$$\Sigma = c_1 \, \mathbf{I}_{K,K} + c_2 \, \mathbf{1}_{K,K}.$$

At the same time a multivariate normal density function on the unconstrained space with a symmetric and diagonal covariance matrix

$$\Omega = c_3 \, \mathbf{I}_{K-1,K-1}$$

pushes forward along T to a multivariate normal density function with covariance matrix

$$\begin{aligned} \Sigma &= \mathbf{T} \cdot \Omega \cdot \mathbf{T}^T \\ &= \mathbf{T} \cdot (c_3 \, \mathbf{I}_{K-1,K-1}) \cdot \mathbf{T}^T \\ &= c_3 \mathbf{T} \cdot \mathbf{I}_{K-1,K-1} \cdot \mathbf{T}^T \\ &= c_3 \mathbf{T} \cdot \mathbf{T}^T. \end{aligned}$$

Consequently the symmetric-diagonal multivariate normal density function on the unconstrained space pushes forward to an exchangeable multivariate normal density function on the constrained space only if

$$\begin{aligned} c_3 \mathbf{T} \cdot \mathbf{T}^T &= c_1 \, \mathbf{I}_{K,K} + c_2 \, \mathbf{1}_{K,K} \\ \mathbf{T} \cdot \mathbf{T}^T &= \frac{c_1}{c_3} \left(\mathbf{I}_{K,K} + \frac{c_2}{c_1} \, \mathbf{1}_{K,K} \right) \\ \mathbf{T} \cdot \mathbf{T}^T &\propto \mathbf{I}_{K,K} + c_4 \, \mathbf{1}_{K,K}. \end{aligned}$$

When building a latent multivariate normal hierarchical model the configuration of the covariance matrix is not fixed but is constrained by population prior model. If the population prior model is centered around a symmetric-diagonal covariance matrix configuration, however, then we might conjecture that any function $T : \mathbb{R}^{K-1} \to X \subset \mathbb{R}^K$ satisfying this condition will push latent behaviors forward to exchangeable behaviors on $X \subset \mathbb{R}^K$. While I cannot offer a formal proof of this conjecture we will see below that it appears to be true empirically.

5.3.3 Anchor Component Constraint

One of the more common techniques for constructing a well-behaved map from a $(K-1)$-simplex to an unconstrained, $(K-1)$-dimensional real space and back is by fixing, or *anchoring*, one of the intermediate components to zero. In general any component can be anchored, but here I will assume that the Kth component is anchored so that the transformations are a bit

easier to write out. The intermediate space X then becomes the subspace of $\mathbb{R}^K$ where the last component is always zero,

$$X = \{(x_1, \ldots, x_K) \in \mathbb{R}^K \mid x_K = 0\}.$$

An immediate mapping from $\mathbb{R}^{K-1}$ to X simply appends the fixed anchor point,

$$\begin{aligned} x_1 &= y_1 \\ &\ldots \\ x_{K-1} &= y_{K-1} \\ x_K &= 0, \end{aligned}$$

or in matrix form,

$$\mathbf{x} = \mathbf{T}_1 \cdot \mathbf{y} = \begin{pmatrix} \mathbf{I}_{K-1,K-1} \\ \mathbf{0}_{1,K-1} \end{pmatrix} \cdot \mathbf{y} = \begin{pmatrix} \mathbf{I}_{K-1,K-1} \cdot \mathbf{y} \\ \mathbf{0}_{1,K-1} \cdot \mathbf{y} \end{pmatrix} = \begin{pmatrix} \mathbf{y} \\ 0 \end{pmatrix},$$

where $\mathbf{I}_{M,M}$ is the M-dimensional identity matrix and $\mathbf{0}_{M,N}$ is the $M \times N$ matrix with all zero entries.

Convolving this transformation with the softmax function gives the simplex components

$$q_k = \frac{\exp(x_k)}{\sum_{k'=1}^{K} \exp(x_{k'})} = \begin{cases} \frac{\exp(y_k)}{1 + \sum_{k'=1}^{K-1} \exp(y_{k'})}, & k < K \\ \frac{1}{1 + \sum_{k'=1}^{K-1} \exp(y_{k'})}, & k = K \end{cases}.$$

Conveniently T_1 is semi-orthogonal

$$\begin{aligned} \mathbf{T}_1^T \cdot \mathbf{T}_1 &= \begin{pmatrix} \mathbf{I}_{K-1,K-1} & \mathbf{0}_{K-1,1} \end{pmatrix} \cdot \begin{pmatrix} \mathbf{I}_{K-1,K-1} \\ \mathbf{0}_{1,K-1} \end{pmatrix} \\ &= \mathbf{I}_{K-1,K-1} \cdot \mathbf{I}_{K-1,K-1} + \mathbf{0}_{K-1,1} \cdot \mathbf{0}_{1,K-1} \\ &= \mathbf{I}_{K-1,K-1} + \mathbf{0}_{K-1,K-1} \\ &= \mathbf{I}_{K-1,K-1}, \end{aligned}$$

allowing us to invert the full map from the unconstrained space to the simplex.

In fact we can invert $\sigma \circ T_1$ all at once by taking advantage of the fact that q_K is equal to the normalizing constant in all of the other simplex components,

$$\begin{aligned} q_k &= \frac{\exp(y_k)}{1 + \sum_{k'=1}^{K-1} \exp(y_{k'})} \\ q_k &= \exp(y_k) \frac{1}{1 + \sum_{k'=1}^{K-1} \exp(y_{k'})} \\ q_k &= \exp(y_k)\, q_K, \end{aligned}$$

or

$$\begin{aligned} \exp(y_k) &= \frac{q_k}{q_K} \\ y_k &= \log \frac{q_k}{q_K}. \end{aligned}$$

This inverse function $T_1^* \circ \sigma^{-1}$ is known as the **multilogit function**.

The multilogit transformation is particularly well-suited to accommodate domain expertise about *proportional* relationships between the simplex components. In particular any information about how strongly q_k can vary proportional to the anchored component probability q_K directly informs the variability of the unconstrained component y_k.

Unfortunately this construction is less accommodating of exchangeable domain expertise,

$$\begin{aligned} \mathbf{T}_1 \cdot \mathbf{T}_1^T &= \begin{pmatrix} \mathbf{I}_{K-1,K-1} \\ \mathbf{0}_{1,K-1} \end{pmatrix} \cdot \begin{pmatrix} \mathbf{I}_{K-1,K-1} & \mathbf{0}_{K-1,1} \end{pmatrix} \\ &= \begin{pmatrix} \mathbf{I}_{K-1,K-1} \cdot \mathbf{I}_{K-1,K-1} & \mathbf{I}_{K-1,K-1} \cdot \mathbf{0}_{K-1,1} \\ \mathbf{0}_{1,K-1} \cdot \mathbf{I}_{K-1,K-1} & \mathbf{0}_{1,K-1} \cdot \mathbf{0}_{1,K-1} \end{pmatrix} \\ &= \begin{pmatrix} \mathbf{I}_{K-1,K-1} & \mathbf{0}_{K-1,1} \\ \mathbf{0}_{1,K-1} & 0 \end{pmatrix} \end{aligned}$$

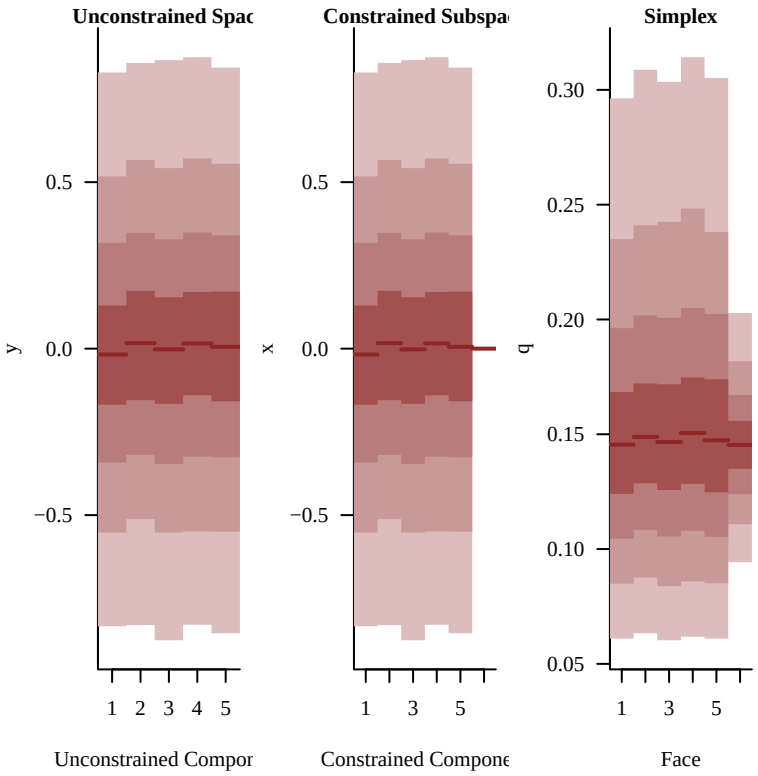
which cannot be written as a linear combination of the identity and ones matrices. Anchoring a particular component distinguishes it from the others and obstructs exchangeability. Specifically the variation of x_K will always be zero, resulting in *less* variation in q_K relative to the other simplex components.

We can confirm this by estimating the pushforward of a latent multivariate normal hierarchical model first to X and then to Δ^{K-1} with Monte Carlo.

```
fit <- stan(file="stan_programs/simu_softmax_2.stan",
            algorithm="Fixed_param", seed=8438338,
            warmup=0, iter=1024, refresh=0)

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

```
par(mfrow=c(1, 3), mar=c(4, 4, 1, 0.5))

names <- sapply(1:5, function(k) paste0('y[', k, ']'))
util$plot_disc_pushforward_quantiles(samples, names,
                                     xlab="Unconstrained Component", ylab="y",
                                     main="Unconstrained Space")

names <- sapply(1:6, function(k) paste0('x[', k, ']'))
```

```
util$plot_disc_pushforward_quantiles(samples, names,
                                     xlab="Constrained Component", ylab="x",
                                     main="Constrained Subspace")

names <- sapply(1:6, function(k) paste0('q[', k, ']'))
util$plot_disc_pushforward_quantiles(samples, names,
                                     xlab="Face", ylab="q",
                                     main="Simplex")
```



5.3.4 Naive Zero Sum Constraint

A more symmetric way to reduce the dimensionality of $\mathbb{R}^K$ is to introduce a constraint on the sum of component values. Without loss of generality we can always take the constrained sum to be one so that the constrained subspace becomes

$$X = \left\{ (x_1, \ldots, x_K) \in \mathbb{R}^K \,\middle|\, \sum_{k=1}^{K} x_k = 0 \right\}.$$

One way to map $K-1$ unconstrained real values to a point in X is to introduce a Kth component that balances the initial values,

$$\begin{aligned} x_1 &= y_1 \\ &\ldots \\ x_{K-1} &= y_{K-1} \\ x_K &= -\sum_{k=1}^{K-1} y_k. \end{aligned}$$

In matrix form this becomes

$$\mathbf{x} = \mathbf{T}_2 \cdot \mathbf{y} = \begin{pmatrix} \mathbf{I}_{K-1,K-1} \\ -\mathbf{1}_{1,K-1} \end{pmatrix} \cdot \mathbf{y} = \begin{pmatrix} \mathbf{I}_{K-1,K-1} \cdot \mathbf{y} \\ -\mathbf{1}_{1,K-1} \cdot \mathbf{y} \end{pmatrix} = \begin{pmatrix} \mathbf{y} \\ -\sum_{k=1}^{K-1} y_k \end{pmatrix},$$

where $\mathbf{1}_{M,N}$ is the $M \times N$ matrix with all one entries.

Applying the softmax transformation then gives the simplex components

$$q_k = \frac{\exp(x_k)}{\sum_{k'=1}^{K} \exp(x_{k'})} = \frac{\exp((\mathbf{T}_2 \cdot \mathbf{y})_k)}{\sum_{k'=1}^{K} \exp((\mathbf{T}_2 \cdot \mathbf{y})_{k'})}$$

We can invert the softmax function by leveraging the zero sum constraint. For any two simplex components we have

$$\begin{aligned} \frac{q_k}{q_{k'}} &= \frac{\exp(x_k)}{\sum_{k''=1}^{K} \exp(x_{k''})} \frac{\sum_{k''=1}^{K} \exp(x_{k''})}{\exp(x_{k'})} \\ \frac{q_k}{q_{k'}} &= \frac{\exp(x_k)}{\exp(x_{k'})} \\ \frac{q_k}{q_{k'}} &= \exp(x_k - x_{k'}) \\ \log \frac{q_k}{q_{k'}} &= x_k - x_{k'} \\ \log q_k - \log q_{k'} &= x_k - x_{k'}. \end{aligned}$$

Summing over the k' index then gives

$$\begin{aligned} \sum_{k'=1}^{K} (\log q_k - \log q_{k'}) &= \sum_{k'=1}^{K} (x_k - x_{k'}) \\ \sum_{k'=1}^{K} \log q_k - \sum_{k'=1}^{K} \log q_{k'} &= \sum_{k'=1}^{K} x_k - \sum_{k'=1}^{K} x_{k'} \\ K \log q_k - \sum_{k'=1}^{K} \log q_{k'} &= K\, x_k - 0 \end{aligned}$$

or

$$x_k = \log q_k - \frac{1}{K}\sum_{k'=1}^{K} \log q_{k'}.$$

Once we've constructed the components of $\mathbf{x}$ we can then map to $\mathbb{R}^{K-1}$ by applying the transpose operation,

$$T_2^*(x) = \mathbf{T}_2^T \cdot \mathbf{x}.$$

The only problem is that this isn't an inverse of $\sigma \circ T_2$; indeed T_2 is not semi-orthogonal,

$$\begin{aligned}
\mathbf{T}_2^T \cdot \mathbf{T}_2 &= \begin{pmatrix} \mathbf{I}_{K-1,K-1} & -\mathbf{1}_{K-1,1} \end{pmatrix} \cdot \begin{pmatrix} \mathbf{I}_{K-1,K-1} \\ -\mathbf{1}_{1,K-1} \end{pmatrix} \\
&= \mathbf{I}_{K-1,K-1} \cdot \mathbf{I}_{K-1,K-1} + \mathbf{1}_{K-1,1} \cdot \mathbf{1}_{1,K-1} \\
&= \mathbf{I}_{K-1,K-1} + \mathbf{1}_{K-1,K-1} \\
&\neq \mathbf{I}_{K-1,K-1}.
\end{aligned}$$

Applying $T_2^* \circ \sigma^{-1}$ and then $\sigma \circ T_2$ verifies the inconsistency in the two maps.

```
K <- 6

T2 <- matrix(0, K, K - 1)
for (k in 1:(K - 1)) {
  T2[k, k] <- 1
  T2[K, k] <- -1
}

q_in <- c(0.1, 0.25, 0.2, 0.35, 0.075, 0.025)
x_in <- sapply(1:K, function(k) log(q_in[k]) - mean(log(q_in)) )
y <- t(T2) %*% x_in
x_out <- T2 %*% y
q_out <- sapply(1:K, function(k) exp(x_out[k]) / sum(exp(x_out)) )

data.frame(q_in, q_out)
```

```
   q_in        q_out
1 0.100 1.025639e-01
2 0.250 2.564098e-01
3 0.200 2.051278e-01
4 0.350 3.589737e-01
5 0.075 7.692293e-02
6 0.025 1.907811e-06
```

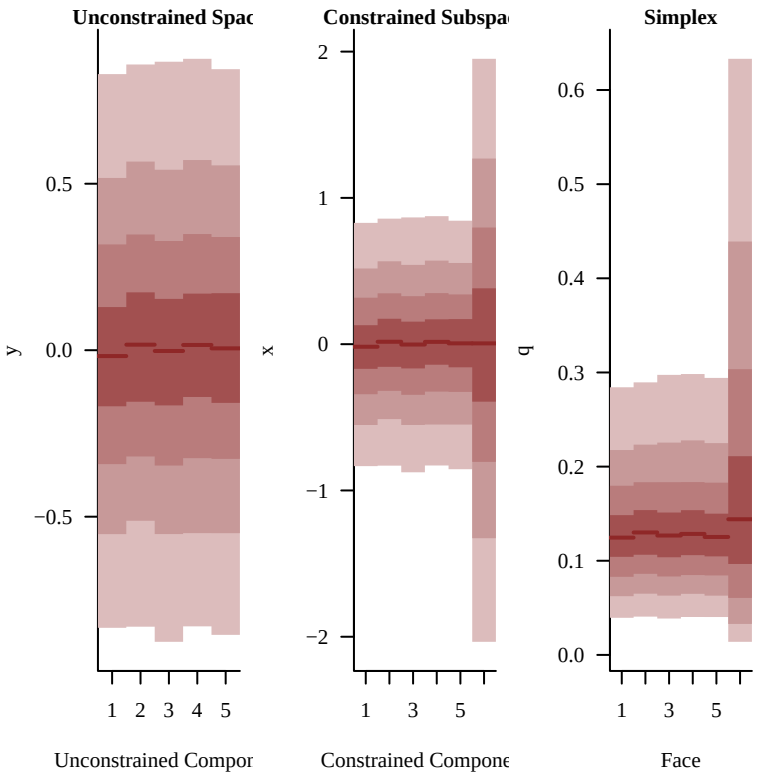
Even worse this T_2 does not satisfy our heuristic exchangeability condition,

$$\begin{aligned}
\mathbf{T}_2 \cdot \mathbf{T}_2^T &= \begin{pmatrix} \mathbf{I}_{K-1,K-1} \\ -\mathbf{1}_{1,K-1} \end{pmatrix} \cdot \begin{pmatrix} \mathbf{I}_{K-1,K-1} & -\mathbf{1}_{K-1,1} \end{pmatrix} \\
&= \begin{pmatrix} \mathbf{I}_{K-1,K-1} \cdot \mathbf{I}_{K-1,K-1} & -\mathbf{I}_{K-1,K-1} \cdot \mathbf{1}_{K-1,1} \\ -\mathbf{1}_{1,K-1} \cdot \mathbf{I}_{K-1,K-1} & \mathbf{1}_{1,K-1} \cdot \mathbf{1}_{K-1,1} \end{pmatrix} \\
&= \begin{pmatrix} \mathbf{I}_{K-1,K-1} & -\mathbf{1}_{K-1,1} \\ -\mathbf{1}_{1,K-1} & K \end{pmatrix}
\end{aligned}$$

The problem here is that T_2 distinguishes the Kth component of $\mathbf{x}$, which depends on all of the components of $\mathbf{y}$, from the others, which each depend on only a single component of $\mathbf{y}$. Probabilistically x_K will tend to exhibit *more* variation than $x_1, \ldots, x_{K-1}$, inflating the variation of q_K relative to the other simplex components.

```
fit <- stan(file="stan_programs/simu_softmax_3.stan",
            algorithm="Fixed_param", seed=8438338,
            warmup=0, iter=1024, refresh=0)

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

```
par(mfrow=c(1, 3), mar=c(4, 4, 1, 0.5))

names <- sapply(1:5, function(k) paste0('y[', k, ']'))
util$plot_disc_pushforward_quantiles(samples, names,
                                     xlab="Unconstrained Component", ylab="y",
                                     main="Unconstrained Space")

names <- sapply(1:6, function(k) paste0('x[', k, ']'))
util$plot_disc_pushforward_quantiles(samples, names,
                                     xlab="Constrained Component", ylab="x",
                                     main="Constrained Subspace")

names <- sapply(1:6, function(k) paste0('q[', k, ']'))
util$plot_disc_pushforward_quantiles(samples, names,
                                     xlab="Face", ylab="q",
                                     main="Simplex")
```



5.3.5 Balanced Zero Sum Constraints

Our initial attempt to encode a zero sum constraint left much to be desired but how do we do any better?

Let's go back to the heuristic exchangeability condition,

$$\mathbf{T} \cdot \mathbf{T}^T = \frac{c_1}{c_3}\left(\mathbf{I}_{K,K} + \frac{c_2}{c_1}\,\mathbf{1}_{K,K}\right).$$

Notice that the outer product on the left-hand side results in $K \times K$ matrix of only rank $K-1$. On the right-hand side we also have $K \times K$ matrix, but the exact rank of that matrix will depend on the coefficients.

Fortunately we can tune these coefficients to engineer consistent behavior on both sides of the equation. For example taking $c_1/c_3 = 1$ and $c_2/c_1 = -1/K$ gives the rank $K-1$ matrix

$$\mathbf{P} = \mathbf{I}_{K,K} - \frac{1}{K}\,\mathbf{1}_{K,K}.$$

Interestingly applying $\mathbf{P}$ twice is the same as applying $\mathbf{P}$ once,

$$\begin{aligned}
\mathbf{P} \cdot \mathbf{P} &= \left(\mathbf{I}_{K,K} - \frac{1}{K} \mathbf{1}_{K,K} \right) \cdot \left(\mathbf{I}_{K,K} - \frac{1}{K} \mathbf{1}_{K,K} \right) \\
\mathbf{P} \cdot \mathbf{P} &= \mathbf{I}_{K,K} \cdot \mathbf{I}_{K,K} - \frac{1}{K} \mathbf{1}_{K,K} \cdot \mathbf{I}_{K,K} - \mathbf{I}_{K,K} \cdot \frac{1}{K} \mathbf{1}_{K,K} + \frac{1}{K} \mathbf{1}_{K,K} \cdot \frac{1}{K} \mathbf{1}_{K,K} \\
\mathbf{P} \cdot \mathbf{P} &= \mathbf{I}_{K,K} - \frac{1}{K} \mathbf{1}_{K,K} - \frac{1}{K} \mathbf{1}_{K,K} + \frac{1}{K^2} (K \, \mathbf{1}_{K,K}) \\
\mathbf{P} \cdot \mathbf{P} &= \mathbf{I}_{K,K} - \frac{1}{K} \mathbf{1}_{K,K} - \frac{1}{K} \mathbf{1}_{K,K} + \frac{1}{K} \mathbf{1}_{K,K} \\
\mathbf{P} \cdot \mathbf{P} &= \mathbf{I}_{K,K} - \frac{1}{K} \mathbf{1}_{K,K} \\
\mathbf{P} \cdot \mathbf{P} &= \mathbf{P}.
\end{aligned}$$

In other words $\mathbf{P}$ is a **projection matrix** (Trefethen and Bau 1997), specifically one that projects $\mathbb{R}^K$ to the $(K-1)$-dimensional subspace $X \subset \mathbb{R}^K$ of zero sum vectors,

$$\begin{aligned}
\mathbf{x}' &= \mathbf{P} \cdot \mathbf{x} \\
&= \left(\mathbf{I}_{K,K} - \frac{1}{K} \mathbf{1}_{K,K} \right) \cdot \mathbf{x} \\
&= \mathbf{I}_{K,K} \cdot \mathbf{x} - \frac{1}{K} \mathbf{1}_{K,K} \cdot \mathbf{x} \\
&= \mathbf{x} - \left(\frac{1}{K} \sum_{k=1}^{K} x_k \right) \mathbf{1}_{K,1}
\end{aligned}$$

with

$$\begin{aligned}
\sum_{k=1}^{K} x'_k &= \sum_{k=1}^{K} \left(x_k - \frac{1}{K} \sum_{k'=1}^{K} x_{k'} \right) \\
&= \sum_{k=1}^{K} x_k - \frac{1}{K} \sum_{k=1}^{K} \left(\sum_{k'=1}^{K} x_{k'} \right) \\
&= \sum_{k=1}^{K} x_k - \frac{1}{K} \, K \sum_{k'=1}^{K} x_{k'} \\
&= \sum_{k=1}^{K} x_k - \sum_{k'=1}^{K} x_{k'} \\
&= 0.
\end{aligned}$$

Even more $\mathbf{P}$ is equal to its own transpose,

$$\begin{aligned}
\mathbf{P}^T &= \left(\mathbf{I}_{K,K} - \frac{1}{K} \mathbf{1}_{K,K} \right)^T \\
\mathbf{P}^T &= \mathbf{I}_{K,K}^T - \frac{1}{K} \mathbf{1}_{K,K}^T \\
\mathbf{P}^T &= \mathbf{I}_{K,K} - \frac{1}{K} \mathbf{1}_{K,K} \\
\mathbf{P}^T &= \mathbf{P}.
\end{aligned}$$

% Because of this $\mathbf{P}$ is an **orthogonal projection matrix** (Trefethen and Bau 1997). Orthogonal projection matrices are special because they can always be decomposed into the outer product of a $K \times (K-1)$ matrix $\mathbf{Q}$ and its transpose,

$$\mathbf{Q} \cdot \mathbf{Q}^T = \mathbf{P},$$

so long as the columns of $\mathbf{Q}$ form an orthonormal basis for $X \subset \mathbb{R}^K$,

$$\mathbf{Q}^T \cdot \mathbf{Q} = \mathbf{I}_{K-1,K-1}.$$

Intuitively this decomposition allows us to interpret the rank $K-1$ projection function

$$P : \mathbb{R}^K \to X \subset \mathbb{R}^K$$

as a composite function that first maps the K-dimensional input space to a $(K-1)$-dimensional intermediate space,

$$Q : \mathbb{R}^K \to \mathbb{R}^{K-1},$$

and then maps that intermediate space to the projected subspace,

$$Q^* : \mathbb{R}^{K-1} \to X \subset \mathbb{R}^K.$$

This result implies that *any* orthonormal basis of $X \subset \mathbb{R}^K$ defines a transformation Q from $\mathbb{R}^{K-1}$ to the zero sum subspace that is both semi-orthogonal,

$$\mathbf{Q}^T \cdot \mathbf{Q} = \mathbf{I}_{K-1,K-1},$$

and satisfies our heuristic exchangeability condition,

$$\mathbf{Q} \cdot \mathbf{Q}^T = \mathbf{P} = \mathbf{I}_{K,K} - \frac{1}{K} \mathbf{1}_{K,K}!$$

In other words any orthonormal basis of the zero sum subspace defines a transformation that satisfies all of our criteria.

Conveniently there are a variety of ways of ways to construct an orthonormal basis of $X \subset \mathbb{R}^K$, and hence a candidate transformation matrix $\mathbf{Q}$. For example the left singular vectors of

the thin/reduced singular value decomposition of $\mathbf{P}$ always define an orthonormal basis of $X \subset \mathbb{R}^K$.

Alternatively we can always use a thin/reduced QR decomposition to construct an orthonormal basis from an initial transformation

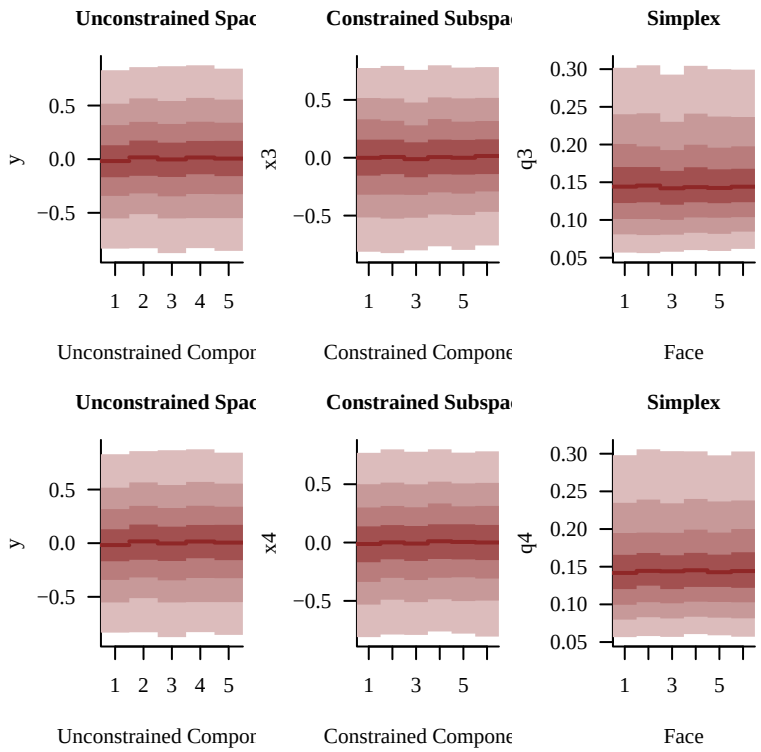
$$T : \mathbb{R}^{K-1} \to X \subset \mathbb{R}^K$$

that is not already semi-orthogonal. In particular the thin/reduced $\mathbf{Q}$ matrix of $\mathbf{T}$ is always a $K \times (K-1)$ matrix whose columns define an orthonormal basis for the output space $X \subset \mathbb{R}^K$, and hence satisfies all of our criteria. For example we could use the $\mathbf{Q}$ matrix from the thin/reduced QR decomposition of the previous map $\mathbf{T}_2$.

```
fit <- stan(file="stan_programs/simu_softmax_4.stan",
            algorithm="Fixed_param", seed=8438338,
            warmup=0, iter=1024, refresh=0)

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

```
par(mfrow=c(2, 3), mar=c(4, 4, 3, 0.5))

names <- sapply(1:5, function(k) paste0('y[', k, ']'))
util$plot_disc_pushforward_quantiles(samples, names,
                                     xlab="Unconstrained Component", ylab="y",
                                     main="Unconstrained Space")

names <- sapply(1:6, function(k) paste0('x3[', k, ']'))
util$plot_disc_pushforward_quantiles(samples, names,
                                     xlab="Constrained Component", ylab="x3",
                                     main="Constrained Subspace")

names <- sapply(1:6, function(k) paste0('q3[', k, ']'))
util$plot_disc_pushforward_quantiles(samples, names,
                                     xlab="Face", ylab="q3",
                                     main="Simplex")

names <- sapply(1:5, function(k) paste0('y[', k, ']'))
util$plot_disc_pushforward_quantiles(samples, names,
                                     xlab="Unconstrained Component", ylab="y",
                                     main="Unconstrained Space")

names <- sapply(1:6, function(k) paste0('x4[', k, ']'))
util$plot_disc_pushforward_quantiles(samples, names,
```

```
                                    xlab="Constrained Component", ylab="x4",
                                    main="Constrained Subspace")

names <- sapply(1:6, function(k) paste0('q4[', k, ']'))
util$plot_disc_pushforward_quantiles(samples, names,
                                    xlab="Face", ylab="q4",
                                    main="Simplex")
```

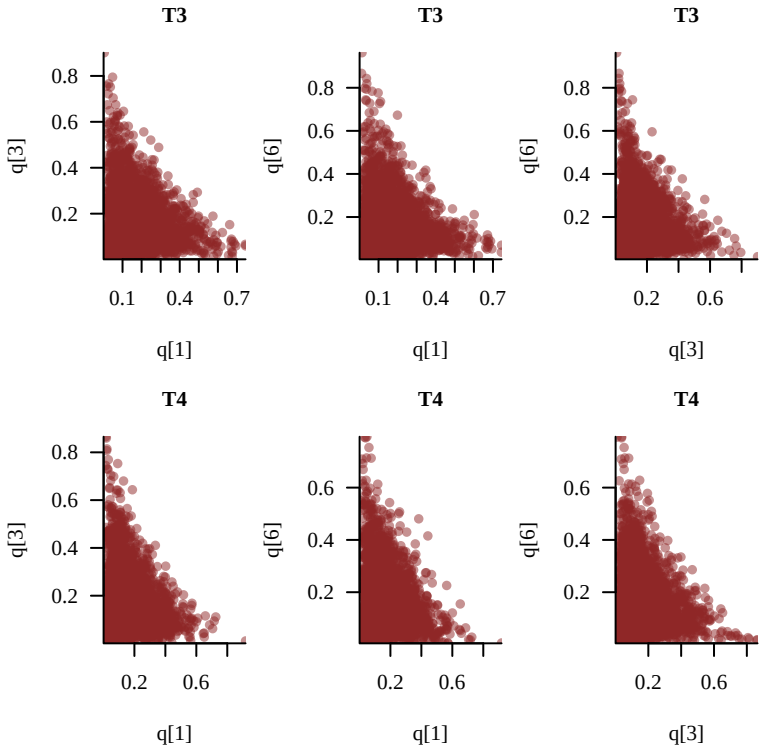


The only downside to this approach is that $y = T^{*} \circ \sigma^{-1}(q)$ will not generally give a clean relationship between simplex behaviors and latent behaviors that we can use to inform an appropriate prior model. That said the preservation of exchangeability allows us to reduce a latent multivariate normal prior model to a few scales which we can then tune with prior pushforward checks.

Note, however, that different transformations will, in general, induce different couplings between simplex components.

```
c_dark_trans <- c("#8F272780")

par(mfrow=c(2, 3), mar=c(4, 4, 3, 0.5))
```

```
plot(samples[['q3[1]']], samples[['q3[3]']], type="p",
     main="T3", col=c_dark_trans, pch=16, cex=1.0,
     xlab="q[1]", ylab="q[3]")
plot(samples[['q3[1]']], samples[['q3[6]']], type="p",
     main="T3", col=c_dark_trans, pch=16, cex=1.0,
     xlab="q[1]", ylab="q[6]")
plot(samples[['q3[3]']], samples[['q3[6]']], type="p",
     main="T3", col=c_dark_trans, pch=16, cex=1.0,
     xlab="q[3]", ylab="q[6]")

plot(samples[['q4[1]']], samples[['q4[3]']], type="p",
     main="T4", col=c_dark_trans, pch=16, cex=1.0,
     xlab="q[1]", ylab="q[3]")
plot(samples[['q4[1]']], samples[['q4[6]']], type="p",
     main="T4", col=c_dark_trans, pch=16, cex=1.0,
     xlab="q[1]", ylab="q[6]")
plot(samples[['q4[3]']], samples[['q4[6]']], type="p",
     main="T4", col=c_dark_trans, pch=16, cex=1.0,
     xlab="q[3]", ylab="q[6]")
```



If we don't have sufficiently informative data then we may have to take care to choose a

transformation that results in reasonable couplings between the individual simplex components. Prior pushforward checks are extremely useful here, especially if we can engineer summary statistics that isolate interpretable coupling behaviors.

5.3.6 Implementation

For this demonstration let's use the linear transformation T_3 so that our full hierarchical model becomes

$$\begin{aligned} p(\mathbf{y} \mid \mu, \tau, \mathbf{L}) &= \text{multinormal}(\mathbf{y}_d \mid \mu, \text{diag}(\tau) \cdot \mathbf{L} \cdot \mathbf{L}^T \cdot \text{diag}(\tau)) \\ \mathbf{x}_d &= \mathbf{T}_3 \cdot \mathbf{y}_d \\ q_d &= \text{softmax}(\mathbf{x}_d). \end{aligned}$$

Note that the latent locations μ define a baseline simplex configurations around which the individual die simplex configurations concentrate,

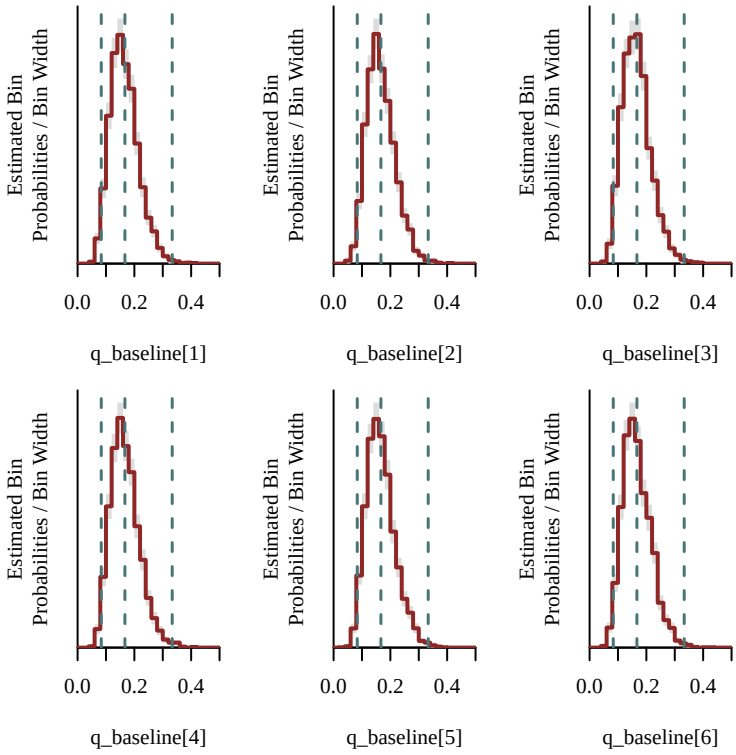
$$q_{\text{baseline}} = \text{softmax}(\mathbf{T}_3 \cdot \mu).$$

Similarly the latent scales τ quantify the strength of the heterogeneity in the hierarchy: when all of the components of τ decrease towards zero the population of individual simplex behaviors collapses around q_{baseline} and when they all increase towards infinity the population decouples into an independent heterogeneity model.

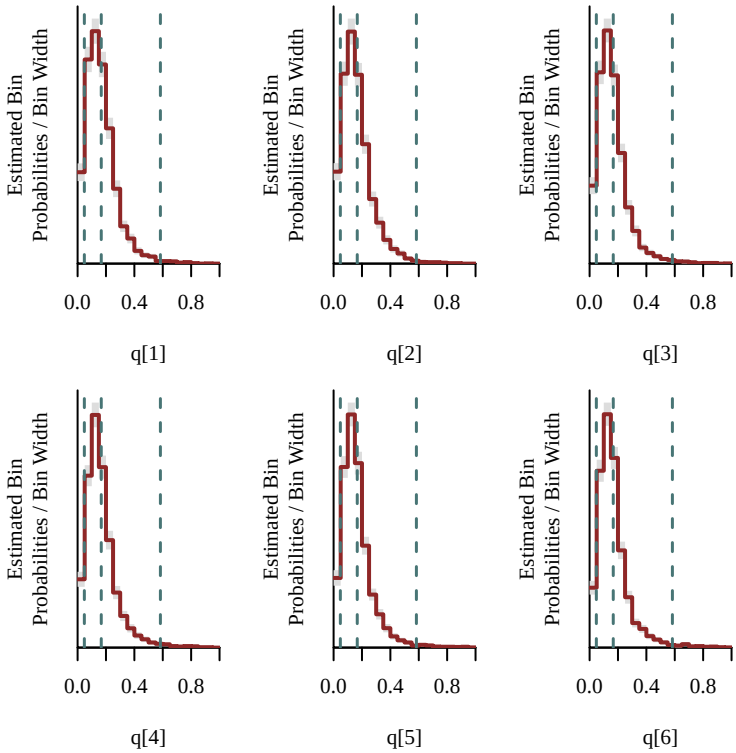
Our hypothetical domain expertise about q_{baseline} and the q_d then allows us to constrain the population prior model for the latent locations, μ, and scales, τ.

```
fit <- stan(file="stan_programs/hierarchical_simplex_prior_3.stan",
            algorithm="Fixed_param", seed=8438338,
            warmup=0, iter=1024, refresh=0)

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

```
par(mfrow=c(2, 3), mar=c(4, 4, 1, 0.5))

for (k in 1:K) {
  name <- paste0("q_baseline[", k, "]")
  util$plot_expectand_pushforward(samples[[name]], 25,
                                  display_name=name,
                                  flim=c(0, 0.5))
  abline(v=1/6 / (1 + 1), col=util$c_mid_teal, lwd=1.5, lty=2)
  abline(v=1/6,           col=util$c_mid_teal, lwd=1.5, lty=2)
  abline(v=1/6 * (1 + 1), col=util$c_mid_teal, lwd=1.5, lty=2)
}
```




```
par(mfrow=c(2, 3), mar=c(4, 4, 1, 0.5))

for (k in 1:K) {
  name <- paste0("q[", k, "]")
  util$plot_expectand_pushforward(samples[[name]], 20,
                                  display_name=name,
                                  flim=c(0, 1))
  abline(v=1/6 / (1 + 2.5), col=util$c_mid_teal, lwd=1.5, lty=2)
  abline(v=1/6,             col=util$c_mid_teal, lwd=1.5, lty=2)
  abline(v=1/6 * (1 + 2.5), col=util$c_mid_teal, lwd=1.5, lty=2)
}
```



Next is the posterior computation. Here we'll use a non-centered parameterization of the latent multivariate normal population model.

```
fit <- stan(file="stan_programs/hierarchical_simplex_3.stan",
            data=data, seed=8438338,
            warmup=1000, iter=2024, refresh=0)
```

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

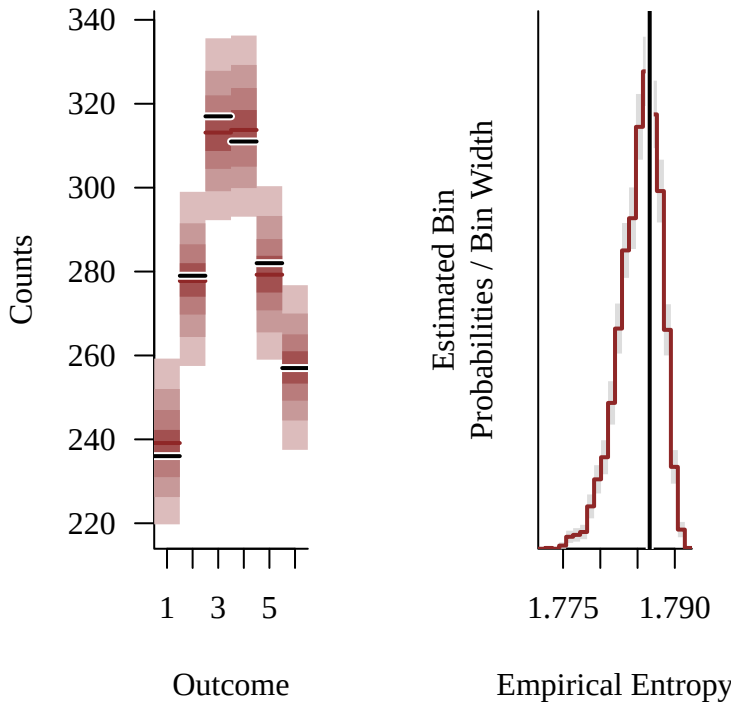
```
  All Hamiltonian Monte Carlo diagnostics are consistent with reliable
Markov chain Monte Carlo.
```

```
samples3 <- util$extract_expectands(fit)
base_samples <- util$filter_expectands(samples3,
                                       c('eta', 'mu', 'tau', 'L'),
                                       check_arrays=TRUE)
util$check_all_expectand_diagnostics(base_samples, exclude_zvar=TRUE)
```

```
All expectands checked appear to be behaving well enough for reliable
Markov chain Monte Carlo estimation.
```

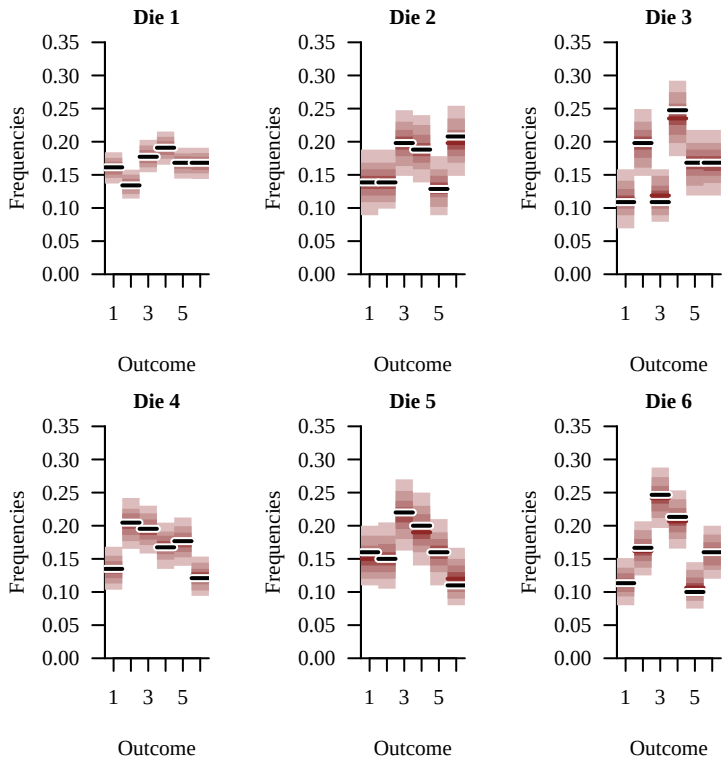
Posterior retrodictive checks follow.

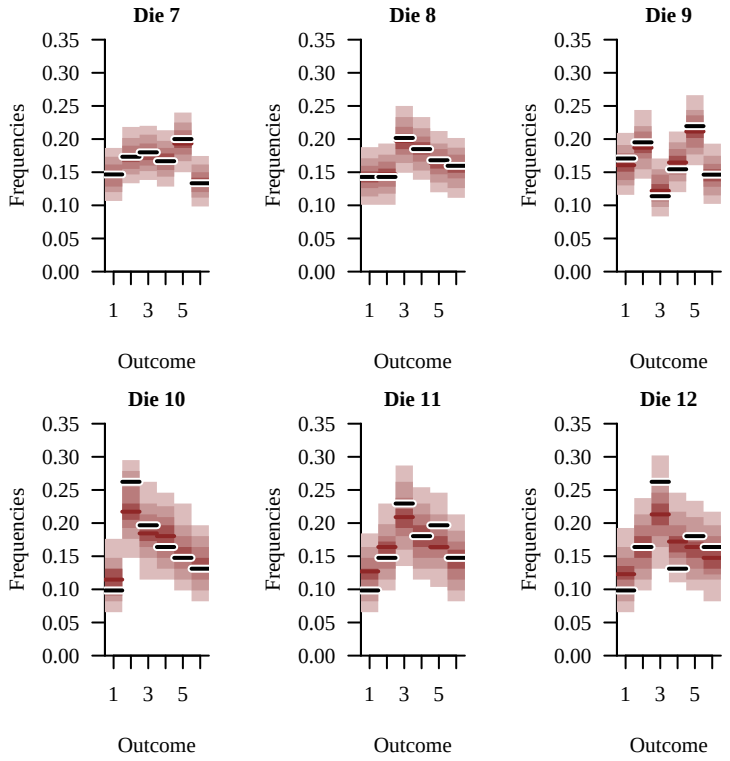
```
par(mfrow=c(1, 2), mar=c(4, 5, 2, 1))

obs_counts <- table(data$outcome)

pred_names <- sapply(1:6, function(k) paste0('pred_counts[', k, ']'))
util$plot_disc_pushforward_quantiles(samples3, pred_names,
                                     baseline_values=obs_counts,
                                     xlab="Outcome", ylab="Counts")

obs_entropy <- sum(sapply(obs_counts,
                          function(n) -(n / data$N) * log(n / data$N)))

util$plot_expectand_pushforward(samples3[["pred_entropy"]], 20,
                                display_name="Empirical Entropy",
                                baseline=obs_entropy)
```



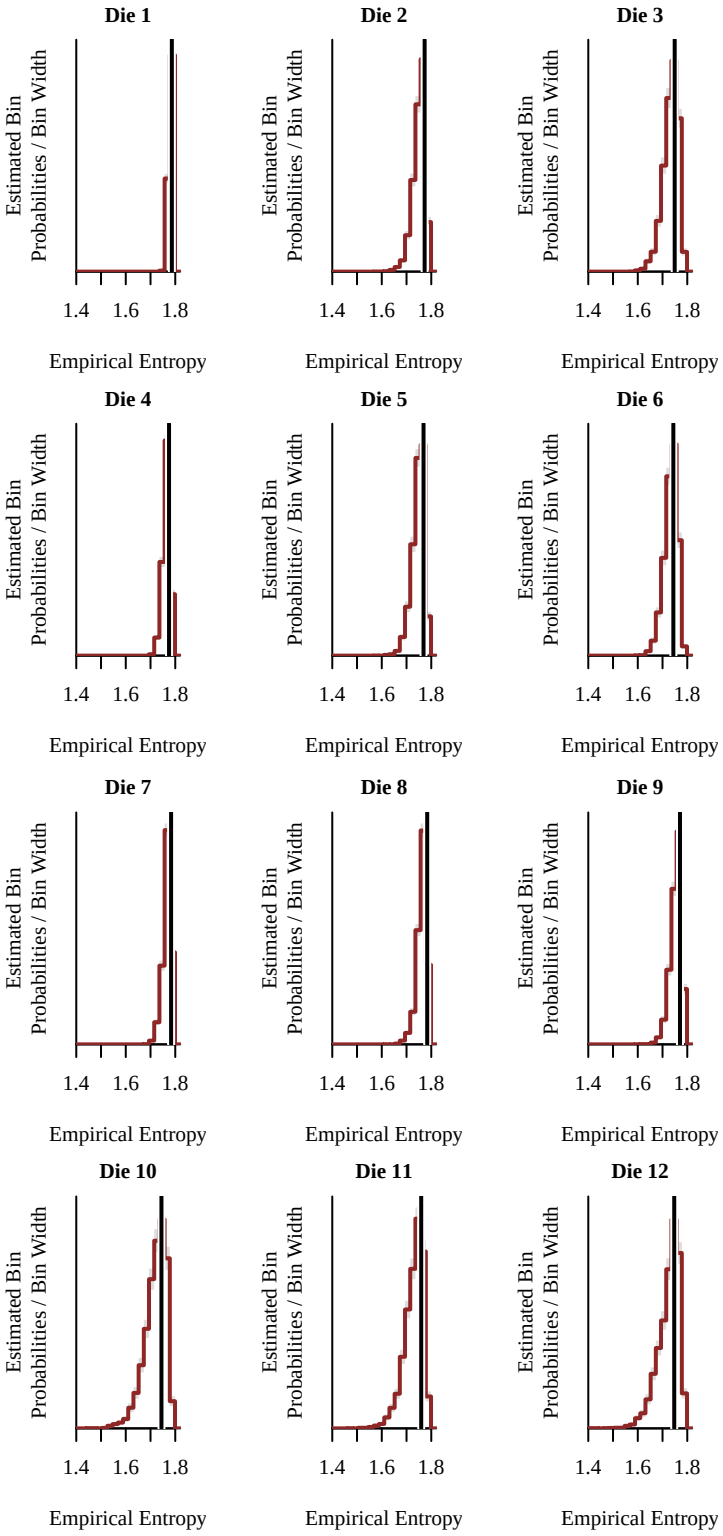

```
for (d in 1:data$N_dice) {
  if (d %% 6 == 1) par(mfrow=c(2, 3), mar=c(4, 5, 2, 1))

  obs_counts <- table(data$outcome[data$die_idxs == d])
```

```
  obs_freq <- obs_counts / sum(obs_counts)

  pred_names <- sapply(1:6,
                       function(k) paste0('pred_freq_die[', d, ',', k, ']'))
  util$plot_disc_pushforward_quantiles(samples3, pred_names,
                                       baseline_values=obs_freq,
                                       xlab="Outcome", ylab="Frequencies",
                                       display_ylim=c(0, 0.35),
                                       main=paste("Die", d))
}
```

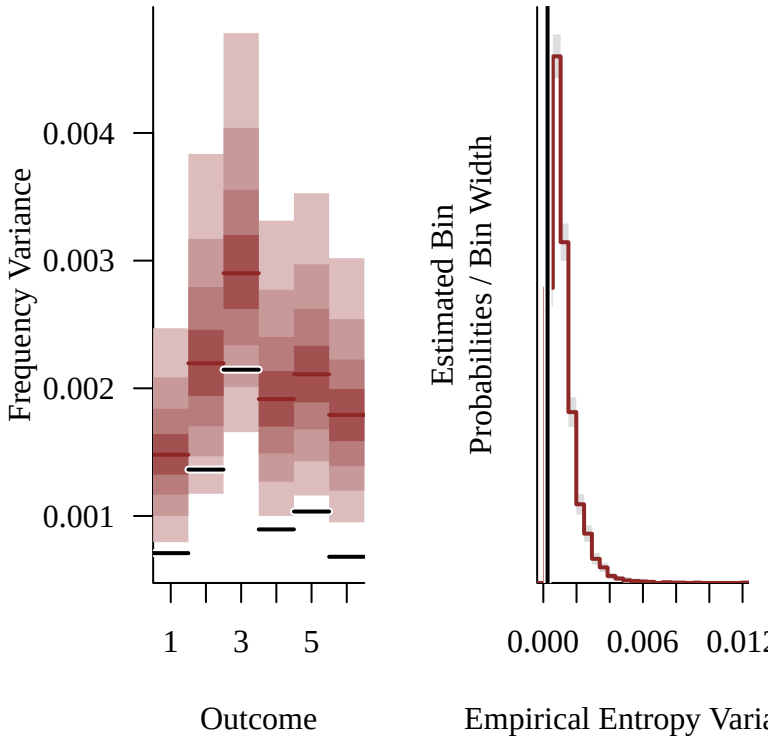





```
for (d in 1:data$N_dice) {
  if (d %% 6 == 1) par(mfrow=c(2, 3), mar=c(4, 5, 2, 1))

  obs_entropy <- compute_empirical_entropy(data$outcome[data$die_idxs == d])

  name <- paste0('pred_entropy_die[', d, ']')
  util$plot_expectand_pushforward(samples3[[name]], 20, flim=c(1.4, 1.82),
                                  display_name="Empirical Entropy",
                                  baseline=obs_entropy, main=paste("Die", d))
}
```

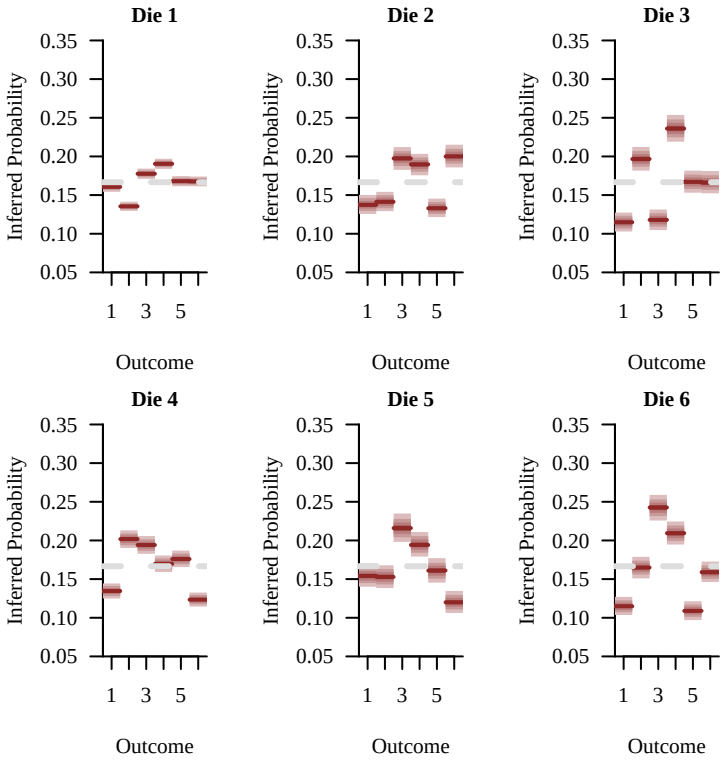

Die 1
Die 2
Die 3
Die 4
Die 5
Die 6
Die 7
Die 8
Die 9
Die 10
Die 11
Die 12
Estimated Bin Probabilities / Bin Width
1.4
1.6
1.8
Empirical Entropy

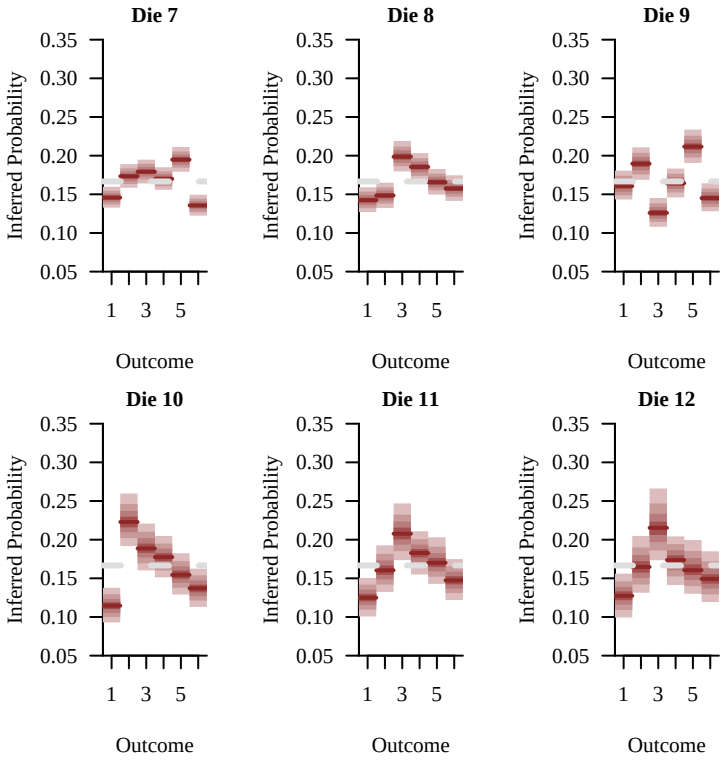
```
par(mfrow=c(1, 2), mar=c(4, 4, 1, 0.5))

obs_var_freq <- sapply(1:6, function(k)
  var(sapply(1:data$N_dice,
             function(d) compute_freq(k, d))))

pred_names <- sapply(1:6, function(k) paste0('pred_var_freq[', k, ']'))
util$plot_disc_pushforward_quantiles(samples3, pred_names,
                                     baseline_values=obs_var_freq,
                                     xlab="Outcome",
                                     ylab="Frequency Variance")

obs_var_entropy <- var(sapply(1:data$N_dice,
                              function(d) compute_empirical_entropy(die_data(d))))

util$plot_expectand_pushforward(samples3[['pred_var_entropy']], 25,
                                display_name="Empirical Entropy Variance",
                                baseline=obs_var_entropy)
```



Finally we can analyze our estimated posterior inferences, starting with the inferences for each individual die.

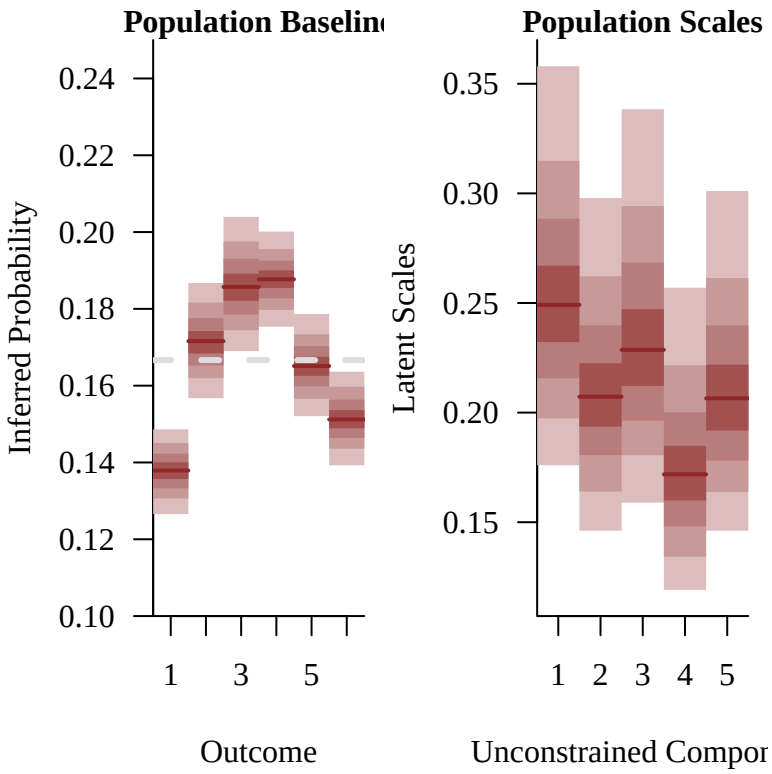
```
for (d in 1:data$N_dice) {
  if (d %% 6 == 1) par(mfrow=c(2, 3), mar=c(4, 5, 2, 1))

  names <- sapply(1:6, function(k) paste0('q[', d, ',', k, ']'))
  util$plot_disc_pushforward_quantiles(samples3, names,
                                       xlab="Outcome",
                                       ylab="Inferred Probability",
                                       display_ylim=c(0.05, 0.35),
                                       main=paste("Die", d))
  abline(h=1/6, col="#DDDDDD", lwd=3, lty=2)
}
```





More relevant to the discussion of heterogeneity are the inferences for the population model.

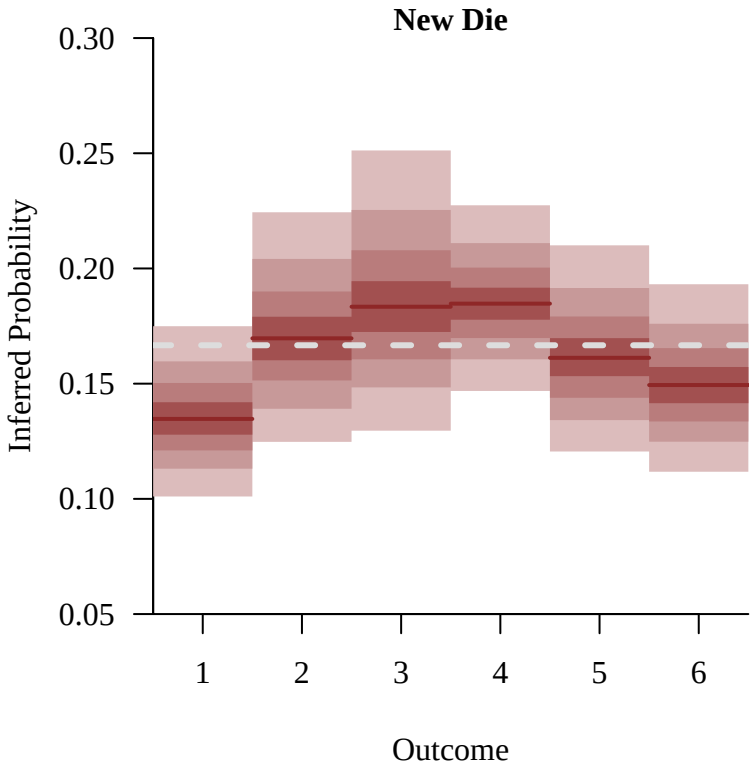
```
par(mfrow=c(1, 2), mar=c(4, 4, 1, 0.5))

names <- sapply(1:6, function(k) paste0('q_baseline[', k, ']'))
util$plot_disc_pushforward_quantiles(samples3, names,
                                     xlab="Outcome",
                                     ylab="Inferred Probability",
                                     display_ylim=c(0.10, 0.25),
                                     main="Population Baseline")
abline(h=1/6, col="#DDDDDD", lwd=3, lty=2)

names <- sapply(1:5, function(k) paste0('tau[', k, ']'))
util$plot_disc_pushforward_quantiles(samples3, names,
                                     xlab="Unconstrained Component",
                                     ylab="Latent Scales",
                                     main="Population Scales")
```



The qualitative shape of these inferences is very similar to those from the Dirichlet population model and not too dissimilar to those from the convex addition population model.

Lastly we have our inferences for the behavior of new, hypothetical dice we need to simulate new simplex configurations from the population model.

```
par(mfrow=c(1, 1), mar=c(4, 4, 1, 0.5))

names <- sapply(1:6, function(k) paste0('q_new[', k, ']'))
util$plot_disc_pushforward_quantiles(samples3, names,
                                     xlab="Outcome",
                                     ylab="Inferred Probability",
                                     display_ylim=c(0.05, 0.3),
                                     main="New Die")
abline(h=1/6, col="#DDDDDD", lwd=3, lty=2)
```



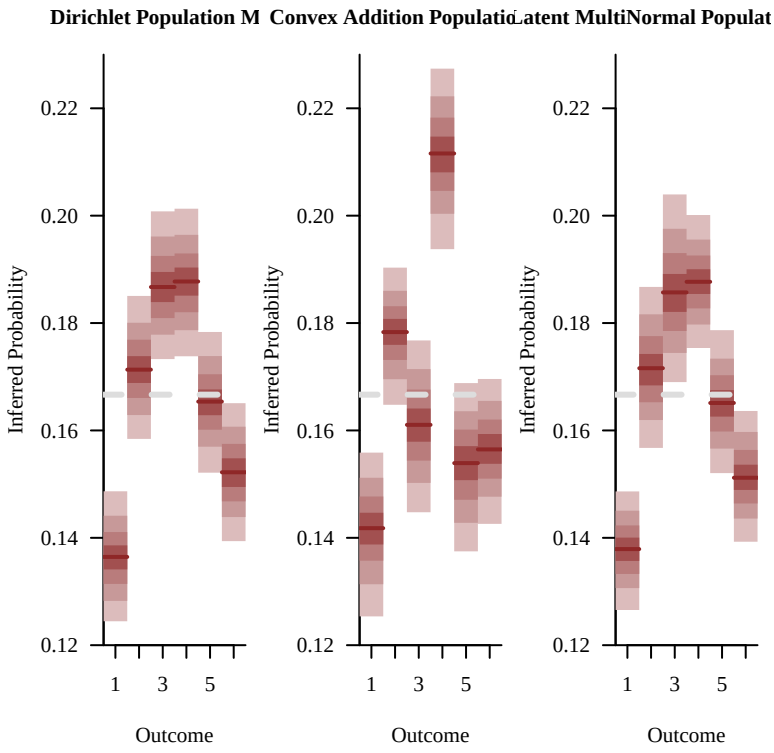
5.4 Inferential Comparison

All three of these simplical hierarchical models share common features, including a baseline simplex configuration and scale parameters that determine how far individual simplex behaviors can deviate from that baseline. The detailed implementation of these features, however, is different enough that we have to be careful when comparing them. In this section we'll directly compare the inferential output of these models.

Let's start by examining posterior inferences for the baseline simplex configurations. Inferences for the Dirichlet and latent multinormal population models are almost exactly the same, but the inferences for the convex addition population model exhibit some unique behaviors for faces 3, 4, and 5.

```
par(mfrow=c(1, 3), mar=c(4, 4, 3, 0.5))

names <- sapply(1:6, function(k) paste0('q_baseline[', k, ']'))
util$plot_disc_pushforward_quantiles(samples1, names,
                                     xlab="Outcome",
                                     ylab="Inferred Probability",
                                     display_ylim=c(0.12, 0.23),
```

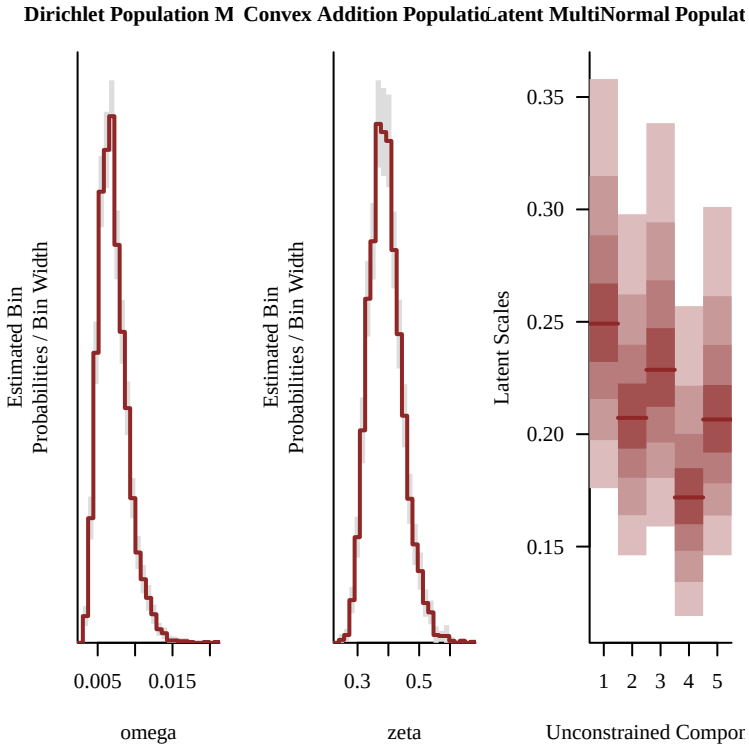
```
                                      main="Dirichlet Population Model")
abline(h=1/6, col="#DDDDDD", lwd=3, lty=2)

names <- sapply(1:6, function(k) paste0('q_baseline[', k, ']'))
util$plot_disc_pushforward_quantiles(samples2, names,
                                      xlab="Outcome",
                                      ylab="Inferred Probability",
                                      display_ylim=c(0.12, 0.23),
                                      main="Convex Addition Population Model")
abline(h=1/6, col="#DDDDDD", lwd=3, lty=2)

names <- sapply(1:6, function(k) paste0('q_baseline[', k, ']'))
util$plot_disc_pushforward_quantiles(samples3, names,
                                      xlab="Outcome",
                                      ylab="Inferred Probability",
                                      display_ylim=c(0.12, 0.23),
                                      main="Latent MultiNormal Population Model")
abline(h=1/6, col="#DDDDDD", lwd=3, lty=2)
```



The marginal posterior distributions for the scale parameters all concentrate on different numerical values but their qualitative behaviors are all similar. Inferences from each model are

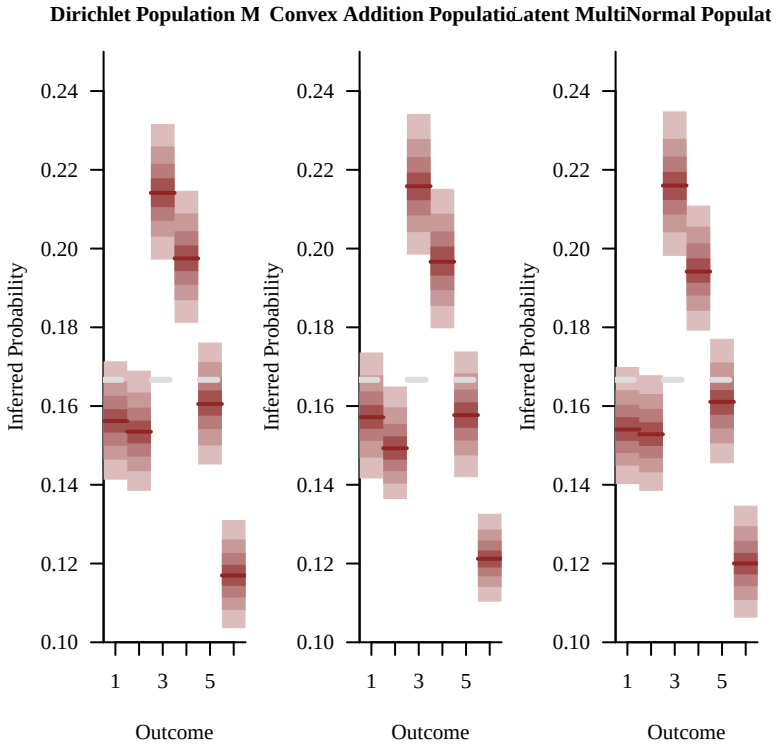
consistent with non-zero but relatively weak heterogeneities, suppressing configurations with both exact homogeneity and strong heterogeneity.

```
par(mfrow=c(1, 3), mar=c(4, 4, 3, 0.5))

util$plot_expectand_pushforward(samples1[['omega']], 25,
                                display_name="omega",
                                main="Dirichlet Population Model")

util$plot_expectand_pushforward(samples2[['zeta']], 25,
                                display_name="zeta",
                                main="Convex Addition Population Model")

names <- sapply(1:5, function(k) paste0('tau[', k, ']'))
util$plot_disc_pushforward_quantiles(samples3, names,
                                     xlab="Unconstrained Component",
                                     ylab="Latent Scales",
                                     main="Latent MultiNormal Population Model")
```



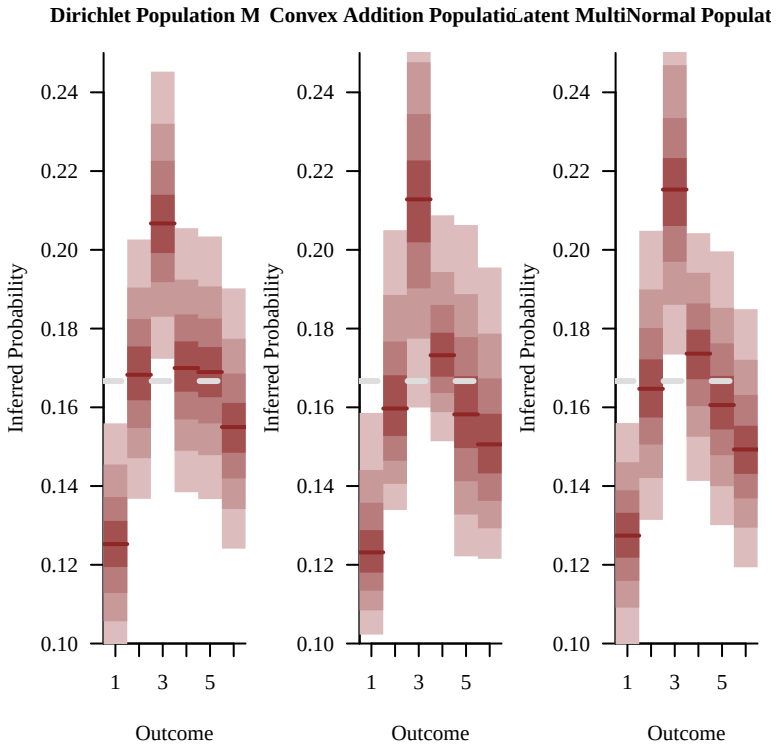
None of these population behaviors, however, are directly observable. When we look at the posterior inferences for the individual die behaviors we see nearly universal agreement.

```
par(mfrow=c(1, 3), mar=c(4, 4, 3, 0.5))

names <- sapply(1:6, function(k) paste0('q[5,', k, ']'))
util$plot_disc_pushforward_quantiles(samples1, names,
                                     xlab="Outcome",
                                     ylab="Inferred Probability",
                                     display_ylim=c(0.10, 0.25),
                                     main="Dirichlet Population Model")
abline(h=1/6, col="#DDDDDD", lwd=3, lty=2)

util$plot_disc_pushforward_quantiles(samples2, names,
                                     xlab="Outcome",
                                     ylab="Inferred Probability",
                                     display_ylim=c(0.10, 0.25),
                                     main="Convex Addition Population Model")
abline(h=1/6, col="#DDDDDD", lwd=3, lty=2)

util$plot_disc_pushforward_quantiles(samples3, names,
                                     xlab="Outcome",
                                     ylab="Inferred Probability",
                                     display_ylim=c(0.10, 0.25),
                                     main="Latent MultiNormal Population Model")
abline(h=1/6, col="#DDDDDD", lwd=3, lty=2)
```




```
par(mfrow=c(1, 3), mar=c(4, 4, 3, 0.5))

names <- sapply(1:6, function(k) paste0('q[12,', k, ']'))
util$plot_disc_pushforward_quantiles(samples1, names,
                                     xlab="Outcome",
                                     ylab="Inferred Probability",
                                     display_ylim=c(0.10, 0.25),
                                     main="Dirichlet Population Model")
abline(h=1/6, col="#DDDDDD", lwd=3, lty=2)

util$plot_disc_pushforward_quantiles(samples2, names,
                                     xlab="Outcome",
                                     ylab="Inferred Probability",
                                     display_ylim=c(0.10, 0.25),
                                     main="Convex Addition Population Model")
abline(h=1/6, col="#DDDDDD", lwd=3, lty=2)

util$plot_disc_pushforward_quantiles(samples3, names,
                                     xlab="Outcome",
                                     ylab="Inferred Probability",
                                     display_ylim=c(0.10, 0.25),
```

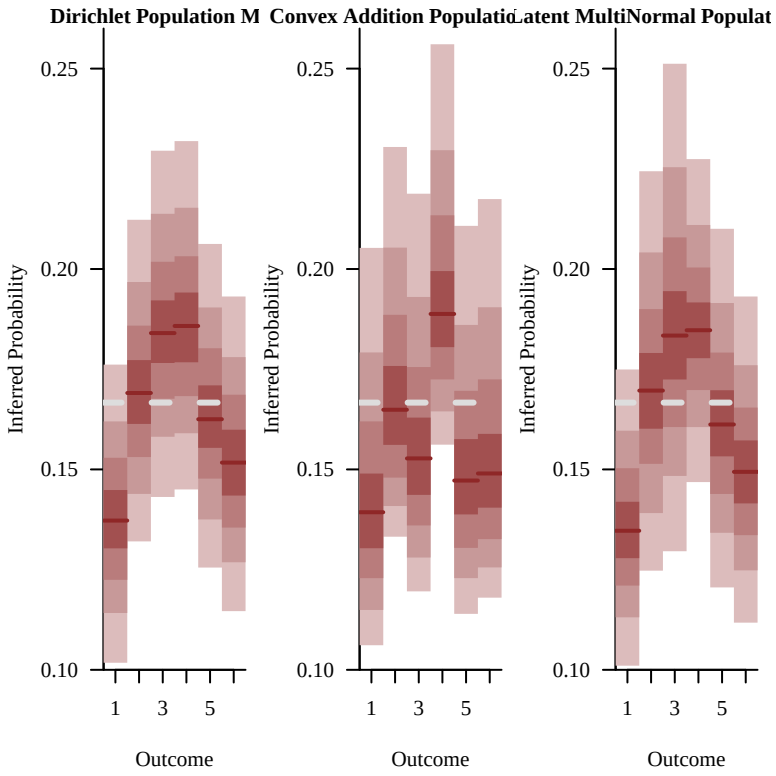
```
                                        main="Latent MultiNormal Population Model")
abline(h=1/6, col="#DDDDDD", lwd=3, lty=2)
```



The same is true for the inferences of new die behaviors, at least within the posterior uncertainties.

```
par(mfrow=c(1, 3), mar=c(4, 4, 1, 0.5))

names <- sapply(1:6, function(k) paste0('q_new[', k, ']'))
util$plot_disc_pushforward_quantiles(samples1, names,
                                     xlab="Outcome",
                                     ylab="Inferred Probability",
                                     display_ylim=c(0.1, 0.26),
                                     main="Dirichlet Population Model")
abline(h=1/6, col="#DDDDDD", lwd=3, lty=2)

names <- sapply(1:6, function(k) paste0('q_new[', k, ']'))
util$plot_disc_pushforward_quantiles(samples2, names,
                                     xlab="Outcome",
                                     ylab="Inferred Probability",
                                     display_ylim=c(0.1, 0.26),
```

```
                                      main="Convex Addition Population Model")
abline(h=1/6, col="#DDDDDD", lwd=3, lty=2)

names <- sapply(1:6, function(k) paste0('q_new[', k, ']'))
util$plot_disc_pushforward_quantiles(samples3, names,
                                      xlab="Outcome",
                                      ylab="Inferred Probability",
                                      display_ylim=c(0.1, 0.26),
                                      main="Latent MultiNormal Population Model")
abline(h=1/6, col="#DDDDDD", lwd=3, lty=2)
```



6 Next Steps

We have come a very far way from our initial questions about die fairness to explicit Bayesian models, inferences, and decisions. That said we are not even close to extracting all of the information in the data at hand.

First and foremost we have not yet taken into account the fact that the die were rolled by different players, each of which may be rolling not only with different techniques but also on

different surfaces. Our analysis of die heterogeneity implicitly assumes that these differences have no influence on the outcomes. We all know how dangerous implicit assumptions can be!

One immediate analysis that we might consider next is to repeat our heterogeneity study across individual players but assuming that each die behaves the same. This would be straightforward mechanically – we just have to replace die indices with player indices everywhere – although we would have to account for any changes to our domain expertise about the potential magnitude of the heterogeneity.

Taking into account heterogeneity across *both* the dice and players can become a bit more challenging.

The most straightforward approach is to build a single hierarchical model with respect to the die/player intersections in which case we can largely reuse the existing implementations. This approach would be particularly useful for this data set given that each player rolled at most only a few dice and most of the die/player intersections were not observed.

We might also consider a factor modeling approach where we expand the heterogeneity in a sequence of main contributions and interaction contributions. The difficulty with factor modeling, however, is that of the three population modeling strategies we discussed only the transformed strategy immediately allows for this additive expansion. For example convex addition of simplices is neither commutative nor associative,

$$(1 - \kappa) \left((1 - \lambda) \, q_1 + \lambda \, q_2 \right) + \kappa q_3 \neq (1 - \lambda) \, q_1 + \lambda \left((1 - \kappa) \, q_2 + \kappa \, q_3 \right).$$

Regardless of what potential sources of heterogeneity that we ultimately consider we have yet to incorporate heterogeneity into our decision making. For example we could repeat the two die roll analysis of Section 3.2.6 only using two different dice instead of rolling the same die twice. Moreover we could consider two different dice that we have observed or even two new dice that we have not yet observed.

7 Conclusion

Beyond actually studying the behavior of my custom Skew Dice™ this analysis serves a few purposes.

Firstly it provides an explicit context for demonstrating the practical implementation of important statistical techniques such as Bayesian decision theory and modeling the heterogeneity in constrained behaviors. Secondly it shows that even apparently simple questions often requires subtle and complex answers.

Bayesian inference is conceptually straightforward, if not outright elegant. The practical implementation of Bayesian inference, however, is frustrated by the requirement that we confront

our assumptions directly. Instead of looking up black box answers we have to address what it actually means for a die to be fair, both in theory and in application.

Once we come to terms with this reality, and anticipate the struggle from the beginning of an analysis, principled Bayesian inferences becomes much less of a burden and much more of an opportunity.

Appendix

In this appendix we'll go into more depth on two of the more tangential topics that arose in the main text.

Appendix A: Radial Subsets on a Simplex

In Section 3.2.5 we saw how any measurable, binary function

$$\begin{aligned} f : X \times X &\to \mathbb{R}^{+} \\ x_1, x_2 &\mapsto d(x_1, x_2) \end{aligned}$$

that vanishes when the two inputs are equal,

$$f(x, x) = 0,$$

can be used to construct measurable subsets around a central point $x \in X$,

$$\mathsf{b}_{x,R} = \{x' \in X \mid f(x, x') < R\}$$

and

$$\mathsf{b}^{*}_{x,R} = \{x' \in X \mid f(x', x) < R\}.$$

If f is symmetric in its inputs then these two subsets will be the same but in general they will be different.

One class of functions that immediate satisfy these properties are measurable distance functions. Conveniently simplex spaces are rich with distance functions, each of which we can use to construct neighborhoods of simplex configurations that are "close" to the uniform simplex configuration,

$$\begin{aligned} \mathsf{u}_{d,R} &= \{q \in \Delta^K \mid d(\upsilon, q) < R\} \\ &= \{q \in \Delta^K \mid d(q, \upsilon) < R\}. \end{aligned}$$

For example because each point in a simplex defines a discrete probability distribution we can use any distance function between probabilities distributions (Betancourt 2021). This includes the total variation distance,

$$d_{\text{TV}}(q, q') = \frac{1}{2} \sum_{k=1}^{K} |q_k - q'_k|,$$

and more generally the Wasserstein distances.

On the other hand information geometry (Amari and Nagaoka 2007) motivates a **geodesic distance** between simplex configurations defined by

$$\begin{aligned} d_g(q, q') &= 2 \arccos \left(\sum_{k=1}^{K} \sqrt{q_k} \sqrt{q'_k} \right) \\ &= 2 \arccos \left(\sum_{k=1}^{K} \sqrt{q_k \, q'_k} \right). \end{aligned}$$

Perhaps surprisingly this distance function admits a relatively straightforward geometric interpretation. The square-roots of the simplex component variables,

$$r_k = \sqrt{q_k},$$

satisfy the constraints

$$0 \leq r_k \leq 1, \quad \sum_{k=1}^{K} r_k^2 = 1,$$

which is exactly the equation for the positive orthant of a K-dimensional hypersphere (Betancourt 2012). A natural way to quantify the distance between points on a hypersphere is with angles, and the cosine of the angular distance between two points is given by the dot product of the corresponding unit vectors,

$$\cos \left(\frac{1}{2} \theta(r, r') \right) = r \cdot r' = \sum_{k=1}^{K} r_k \, r'_k.$$

Consequently the angular distance is given by

$$\theta(r, r') = 2 \arccos \left(\sum_{k=1}^{K} r_k \, r'_k \right).$$

Substituting $r_k = \sqrt{q_k}$ and $r'_k = \sqrt{q'_k}$ gives the geodesic distance function d_g.

Another class of functions that can be used to implicitly define subsets are **divergences** between measures (Csiszár and Shields 2004). In general divergences are binary functions that map two measures into a positive real number that vanish when the input measures are

identical and increases as the input measures deviate more and more strongly. Unlike distance functions, however, divergences are not necessarily symmetric in their arguments.

In contemporary data analysis the most popular divergence is the **Kullback-Leibler divergence**. For finite probability distributions the Kullback-Leibler divergence takes the form

$$\mathrm{KL}(q' \mid\mid q) = -\sum_{k=1}^{K} q'_k \log \frac{q_k}{q'_k}.$$

A common convention to to differentiate the two arguments is to say that $\mathrm{KL}(q' \mid\mid q)$ is the Kullback-Leibler divergence *from* q *to* q'. Because the Kullback-Leibler divergence is not symmetric in its arguments it can be used to define two unique subsets around the uniform simplex configuration $\upsilon \in \Delta^K$: the subset *from* υ

$$\mathsf{u}_{\mathrm{KL},R} = \{q \in \Delta^K \mid \mathrm{KL}(\upsilon \mid\mid q) < R\}$$

and the subset *to* υ,

$$\mathsf{u'}_{\mathrm{KL},R} = \{q \in \Delta^K \mid \mathrm{KL}(q \mid\mid \upsilon) < R\}.$$

The precise size and shape of any subset implicitly defined by a distance or divergence function will depend on the particular properties of that function. Unfortunately these geometric consequences are not always intuitive. To built some geometric intuition for the distance and divergence functions that we've discussed so far in this section let's explore their behavior in the 3-simplex Δ^2 which we can conveniently visualize in its entirety.

Let's start by plotting the functions themselves.

```
# Total variation distance
tv_distance <- function(q1, q2) {
  sum( abs(q1 - q2) ) / 2
}

# Geodesic distance
geodesic_distance <- function(q1, q2) {
  2 * acos( sum(sqrt( q1 * q2 )) )
}

# Kullback-Leibler divergence
lmult <- function(x, y) {
  if (x == 0) 0 else x * log(y)
}

kl_divergence <- function(q1, q2) {
  -sum(sapply(1:3, function(k) lmult(q1[k], q2[k] / q1[k])))
}
```

```
to_simplex_config <- function(x, y, C) {
  c((1 - x / C - y) / 2, (1 + x / C - y) / 2, y)
}
```

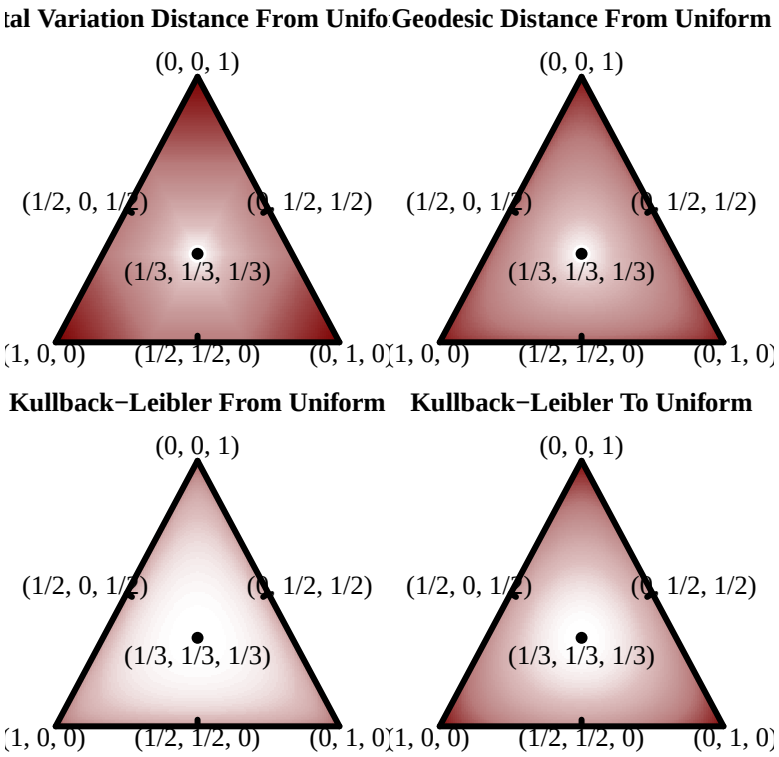
```
valid_simplex_config <- function(q) {
  q[1] >= 0 & q[2] >= 0 & q[3] >= 0 & q[1] + q[2] + q[3] <= 1 + 1e-8
}
```

```
plot_simplex_border <- function(label_cex, C, center_label=TRUE) {
  lines( c(-C, 0), c(0, 1), lwd=3)
  lines( c(+C, 0), c(0, 1), lwd=3)
  lines( c(-C, +C), c(0, 0), lwd=3)

  text_delta <- 0.05
  text( 0, 1 + text_delta, "(0, 0, 1)", cex=label_cex)
  text(-C - text_delta, -text_delta, "(1, 0, 0)", cex=label_cex)
  text(+C + text_delta, -text_delta, "(0, 1, 0)", cex=label_cex)

  tick_delta <- 0.025
  lines( c(0, 0), c(0, tick_delta), lwd=3)
  text(0, 0 - text_delta, "(1/2, 1/2, 0)", cex=label_cex)

  lines( c(+C * 0.5, +C * 0.5 - tick_delta * 0.5 * sqrt(3)),
         c(0.5, 0.5 - tick_delta * 0.5), lwd=3)
  text(C * 0.5 + text_delta * 0.5 * sqrt(3) + 2.5 * text_delta,
       0.5 + text_delta * 0.5, "(0, 1/2, 1/2)", cex=label_cex)

  lines( c(-C * 0.5, -C * 0.5 + tick_delta * 0.5 * sqrt(3)),
         c(0.5, 0.5 - tick_delta * 0.5), lwd=3)
  text(-C * 0.5 - text_delta * 0.5 * sqrt(3) - 2.5 * text_delta,
       0.5 + text_delta * 0.5, "(1/2, 0, 1/2)", cex=label_cex)

  points(0, 1/3, col="white", pch=16, cex=1.5)
  points(0, 1/3, col="black", pch=16, cex=1)
  if (center_label)
    text(0, 1/3 - 1.5 * text_delta, "(1/3, 1/3, 1/3)", cex=label_cex)
}
```

```
plot_function <- function(f, label_cex=1, main=NA) {
  N <- 200
  C <- 1 / sqrt(3)
  xs <- seq(-C - 0.2, C + 0.2, (2 * C + 0.4) / (N - 1))
```

```
  ys <- seq(0 - 0.1, 1 + 0.1, 1.2 / (N - 1))
  zs <- matrix(0, nrow=N, ncol=N)

  for (n in 1:N) {
    for (m in 1:N) {
      q <- to_simplex_config(xs[n], ys[m], C)

      if (valid_simplex_config(q)) {
        zs[n, m] <- f(q)
      } else {
        zs[n, m] <- NA
      }
    }
  }

  if (is.na(main)) {
    par(mar = c(0, 0, 0, 0))
    image(xs, ys, zs, col=rev(cont_colors), axes=FALSE, ann=FALSE)
  } else {
    par(mar = c(0, 0, 2, 0))
    image(xs, ys, zs, col=rev(cont_colors), axes=FALSE, ann=FALSE)
    title(main)
  }

  plot_simplex_border(label_cex, C)
}
```

```
upsilon <- rep(1/3, 3)

par(mfrow=c(2, 2))

plot_function(function(q) { tv_distance(q, upsilon) },
              main="Total Variation Distance From Uniform")
plot_function(function(q) { geodesic_distance(q, upsilon) },
              main="Geodesic Distance From Uniform")
plot_function(function(q) { kl_divergence(upsilon, q) },
              main="Kullback-Leibler From Uniform")
plot_function(function(q) { kl_divergence(q, upsilon) },
              main="Kullback-Leibler To Uniform")
```

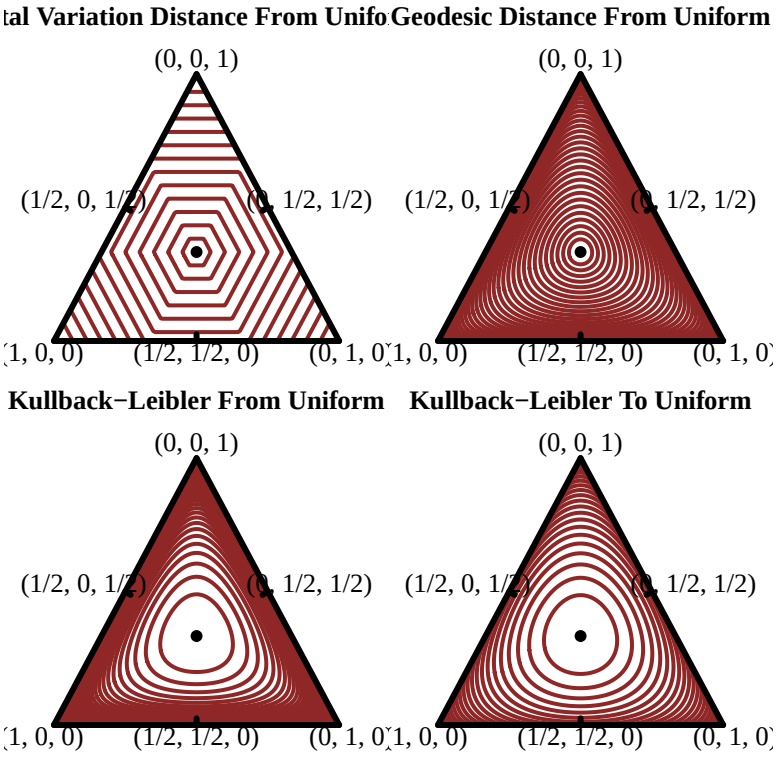


The potential subset geometries are easier to see if we plot level sets of these functions instead their full output.

```
c_dark <- c("#8F2727")

plot_contours <- function(f, label_cex=1, main=NA) {
  N <- 200
  C <- 1 / sqrt(3)
  xs <- seq(-C - 0.2, C + 0.2, (2 * C + 0.4) / (N - 1))
  ys <- seq(0 - 0.1, 1 + 0.1, 1.2 / (N - 1))
  zs <- matrix(0, nrow=N, ncol=N)

  for (n in 1:N) {
    for (m in 1:N) {
      q <- to_simplex_config(xs[n], ys[m], C)

      if (valid_simplex_config(q)) {
        zs[n, m] <- f(q)
      } else {
        zs[n, m] <- NA
      }
```

```
    }
  }

  if (is.na(main)) {
    par(mar = c(0, 0, 0, 0))
    contour(xs, ys, zs, levels=seq(0, 2, 0.05),
            axes=FALSE, ann=FALSE,
            drawlabels=FALSE, col=c_dark, lwd=2)
  } else {
    par(mar = c(0, 0, 2, 0))
    contour(xs, ys, zs, levels=seq(0, 2, 0.05),
            axes=FALSE, ann=FALSE,
            drawlabels=FALSE, col=c_dark, lwd=2)
    title(main)
  }

  plot_simplex_border(label_cex, C, FALSE)
}
```

```
plot_contours(function(q) { tv_distance(q, upsilon) },
              main="Total Variation Distance From Uniform")
plot_contours(function(q) { geodesic_distance(q, upsilon) },
              main="Geodesic Distance From Uniform")
plot_contours(function(q) { kl_divergence(upsilon, q) },
              main="Kullback-Leibler From Uniform")
plot_contours(function(q) { kl_divergence(q, upsilon) },
              main="Kullback-Leibler To Uniform")
```



The geometric consequences of these choices are pretty striking. The level sets of the total variation distance function exhibit a hexagonal symmetry with sharp corners while the other functions yield much smoother level sets. Moreover the level sets of the Kullback-Leibler divergence from the uniform simplex configuration tend to bow out towards the corners of the simplex while the level sets of the Kullback-Leibler divergence to the uniform simplex configuration tend to bow out towards the edges of the simplex. Those of the geodesic distance function are a bit more uniform in all directions.

Moreover the separation between the level sets is quite different for each function. In particular the level sets of the Kullback-Leibler divergence from the uniform simplex configuration become extremely dense as we move towards the simplex boundaries.

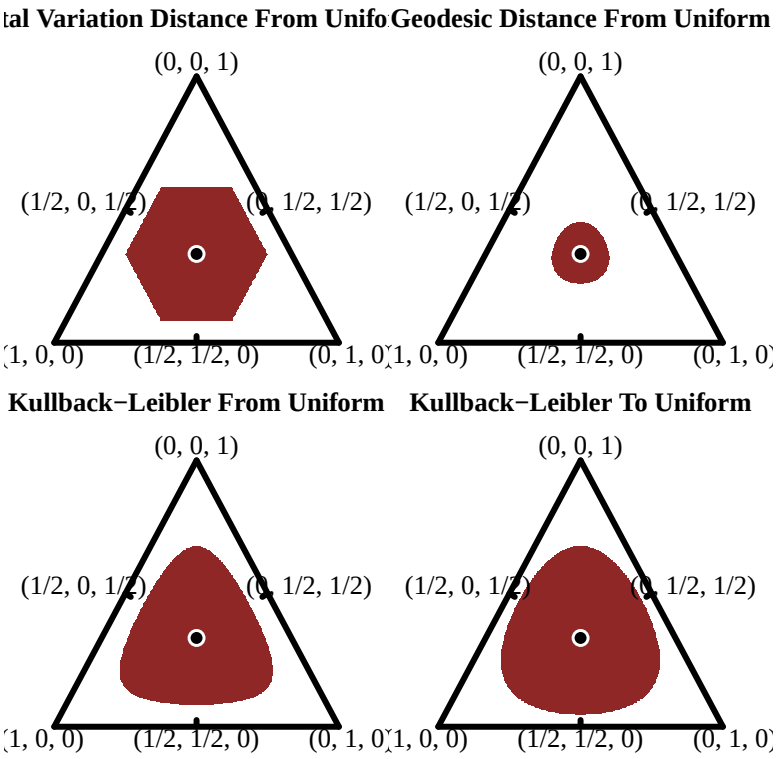
In addition to the shape the size of these subsets can also vary, even if the defining the radius R is fixed. To investigate this let's examine the particular subsets we get for the radius $R = 0.25$.

```
disc_colors <- c("#FFFFFF", "#8F2727")
cont_colors <- colormap(colormap=disc_colors, nshades=2)
```

```
plot_radial_subset <- function(f, R, label_cex=1, main=NA) {
  N <- 500
  C <- 1 / sqrt(3)
```

```
  xs <- seq(-C - 0.2, C + 0.2, (2 * C + 0.4) / (N - 1))
  ys <- seq(0 - 0.1, 1 + 0.1, 1.2 / (N - 1))
  zs <- matrix(0, nrow=N, ncol=N)

  for (n in 1:N) {
    for (m in 1:N) {
      q <- to_simplex_config(xs[n], ys[m], C)

      if (valid_simplex_config(q)) {
        r <- f(q)
        if (r <= R)
          zs[n, m] <- 1
        else
          zs[n, m] <- 0
      } else {
        zs[n, m] <- NA
      }
    }
  }

  if (is.na(main)) {
    par(mar = c(0, 0, 0, 0))
    image(xs, ys, zs, col=rev(cont_colors), axes=FALSE, ann=FALSE)
  } else {
    par(mar = c(0, 0, 2, 0))
    image(xs, ys, zs, col=rev(cont_colors), axes=FALSE, ann=FALSE)
    title(main)
  }

  plot_simplex_border(label_cex, C, FALSE)
}
```

```
plot_radial_subset(function(q) { tv_distance(q, upsilon) },
                   0.25, main="Total Variation Distance From Uniform")
plot_radial_subset(function(q) { geodesic_distance(q, upsilon) },
                   0.25, main="Geodesic Distance From Uniform")
plot_radial_subset(function(q) { kl_divergence(upsilon, q) },
                   0.25, main="Kullback-Leibler From Uniform")
plot_radial_subset(function(q) { kl_divergence(q, upsilon) },
                   0.25, main="Kullback-Leibler To Uniform")
```



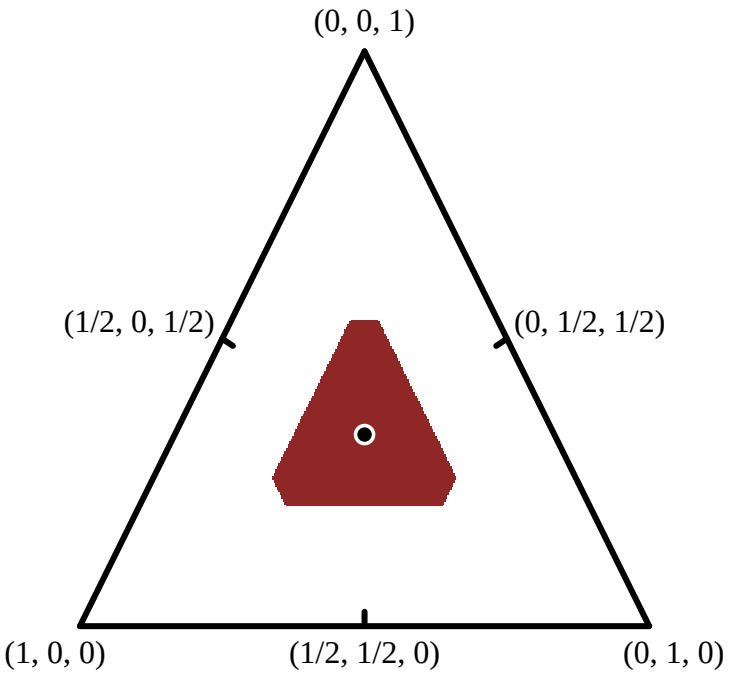
Here the size of the subset defined by the geodesic distance function is noticeably smaller than the others. Because the radius can have different geometric consequences for different functions there is no well-defined default radius on which we can rely in practice. Instead we have to separately tune the radius for each choice of function to ensure an appropriate subset, as we did in Section 3.2.5.

While we are visualizing implicitly-defined subsets let's consider a subset defined by interval constraints on the individual components.

```
plot_componentwise_subset <- function(u, t, label_cex=1, main=NA) {
  N <- 500
  C <- 1 / sqrt(3)
  xs <- seq(-C - 0.2, C + 0.2, (2 * C + 0.4) / (N - 1))
  ys <- seq(0 - 0.1, 1 + 0.1, 1.2 / (N - 1))
  zs <- matrix(0, nrow=N, ncol=N)

  for (n in 1:N) {
    for (m in 1:N) {
      q <- to_simplex_config(xs[n], ys[m], C)

      if (valid_simplex_config(q)) {
        comp_inclusion <- sapply(seq_along(q),
```

```
                                function(k) u[k] / (1 + t) < q[k] &
                                            q[k] < u[k] * (1 + t))
        total_inclusion <- Reduce("&", comp_inclusion)
        zs[n, m] <- total_inclusion
      } else {
        zs[n, m] <- NA
      }
    }
  }

  if (is.na(main)) {
    par(mar = c(0, 0, 0, 0))
    image(xs, ys, zs, col=rev(cont_colors), axes=FALSE, ann=FALSE)
  } else {
    par(mar = c(0, 0, 2, 0))
    image(xs, ys, zs, col=rev(cont_colors), axes=FALSE, ann=FALSE)
    title(main)
  }

  plot_simplex_border(label_cex, C, FALSE)
}
```

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

plot_componentwise_subset(upsilon, 0.6, main="Component-Constrained Subset")
```



This subset exhibits the similar corners as the the subset defined by the total variation distance function but not the same symmetry. Instead the component interval constraints result in different lengths along the six edges.

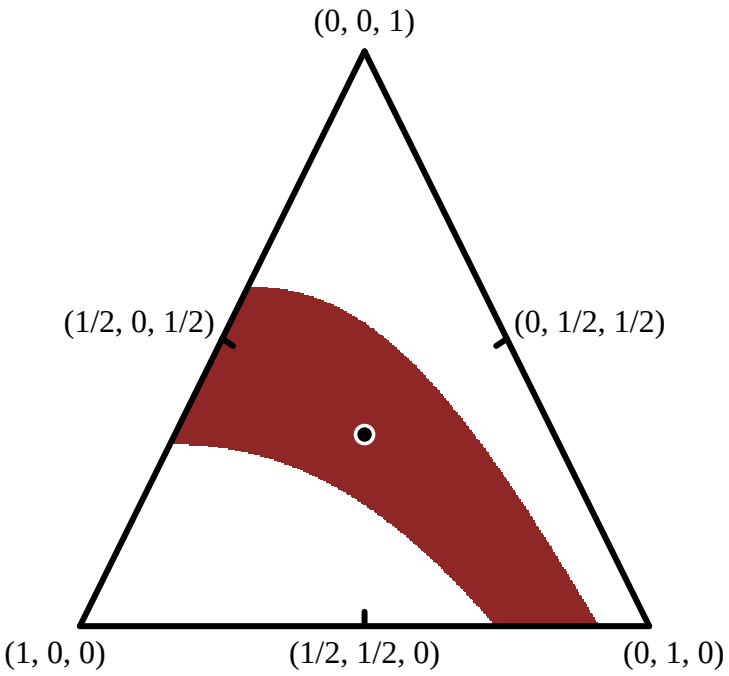
Finally let's see what kind of geometry we get if we implicitly define a subset using a pushforward criterion, emulating the criterion that we considered in Section 3.2.6.

```
pushforward_simplex <- function(q) {
  c(    q[1] * q[1],
    2 * q[1] * q[2],
    2 * q[1] * q[3] + q[2] * q[2],
    2 * q[2] * q[3],
        q[3] * q[3])
}
```

```
plot_pushforward_subset <- function(u, t, label_cex=1, main=NA) {
  N <- 500
  C <- 1 / sqrt(3)
  xs <- seq(-C - 0.2, C + 0.2, (2 * C + 0.4) / (N - 1))
  ys <- seq(0 - 0.1, 1 + 0.1, 1.2 / (N - 1))
  zs <- matrix(0, nrow=N, ncol=N)
```

```
  for (n in 1:N) {
    for (m in 1:N) {
      q <- to_simplex_config(xs[n], ys[m], C)

      if (valid_simplex_config(q)) {
        unif_2d3 <- pushforward_simplex(upsilon)
        unif_success <- sum(unif_2d3[3:5])

        q_2d3 <- pushforward_simplex(q)
        q_success <- sum(q_2d3[3:5])

        zs[n, m] <- unif_success / (1 + t) < q_success &
                    q_success < unif_success * (1 + t)
      } else {
        zs[n, m] <- NA
      }
    }
  }

  if (is.na(main)) {
    par(mar = c(0, 0, 0, 0))
    image(xs, ys, zs, col=rev(cont_colors), axes=FALSE, ann=FALSE)
  } else {
    par(mar = c(0, 0, 2, 0))
    image(xs, ys, zs, col=rev(cont_colors), axes=FALSE, ann=FALSE)
    title(main)
  }

  plot_simplex_border(label_cex, C, FALSE)
}
```

```
plot_pushforward_subset(upsilon, 0.25, main="Pushforward-Constrained Subset")
```



This subset exhibits a much more complicated shape corresponding to the fact that there are may ways for the simplex to contort itself in order to match the pushforward constraint.

Appendix B: Compositional Data Analysis

In Section 5.3 we engineered a family of transformations $\sigma \circ T$ from a $(K-1)$-dimensional real space to a $(K-1)$-simplex. These transformations can also be used to translate the algebraic structure of the input space, especially addition, to an algebraic structure on the output space.

Any two points $y_1, y_2 \in \mathbb{R}^{K-1}$ induce two simplex configurations $q_1, q_2 \in \Delta^{K-1}$ with the components

$$q_{1,k} = \frac{\exp\left(\sum_{k''=1}^{K-1} T_{kk''} y_{1,k''}\right)}{\sum_{k'=1}^{K} \exp\left(\sum_{k''=1}^{K-1} T_{k'k''} y_{1,k''}\right)}.$$

$$q_{2,k} = \frac{\exp\left(\sum_{k''=1}^{K-1} T_{kk''} y_{2,k''}\right)}{\sum_{k'=1}^{K} \exp\left(\sum_{k''=1}^{K-1} T_{kk''} y_{2,k''}\right)}.$$

The sum of the two unconstrained points also defines a third simplex configuration with the

components

$$
\begin{aligned}
q_{3,k} &= \frac{\exp\left(\sum_{k''=1}^{K-1} T_{kk''}(y_{1,k''} + y_{2,k''})\right)}{\sum_{k'=1}^{K} \exp\left(\sum_{k''=1}^{K-1} T_{k'k''}(y_{1,k''} + y_{2,k''})\right)} \\
&= \frac{\exp\left(\sum_{k''=1}^{K-1} T_{kk''} y_{1,k''}\right) \exp\left(\sum_{k''=1}^{K-1} T_{kk''} y_{2,k''}\right)}{\sum_{k'=1}^{K} \exp\left(\sum_{k''=1}^{K-1} T_{k'k''} y_{1,k''}\right) \exp\left(\sum_{k''=1}^{K-1} T_{k'k''} y_{2,k''}\right)} \\
&= \frac{\frac{\exp\left(\sum_{k''=1}^{K-1} T_{kk''} y_{1,k''}\right)}{\sum_{k'''=1}^{K} \exp\left(\sum_{k''=1}^{K-1} T_{k'''k''} y_{1,k''}\right)} \frac{\exp\left(\sum_{k''=1}^{K-1} T_{kk''} y_{2,k''}\right)}{\sum_{k'''=1}^{K} \exp\left(\sum_{k''=1}^{K-1} T_{k'''k''} y_{2,k''}\right)}}{\sum_{k'=1}^{K} \frac{\exp\left(\sum_{k''=1}^{K-1} T_{k'k''} y_{1,k''}\right)}{\sum_{k'''=1}^{K} \exp\left(\sum_{k''=1}^{K-1} T_{k'''k''} y_{1,k''}\right)} \frac{\exp\left(\sum_{k''=1}^{K-1} T_{k'k''} y_{2,k''}\right)}{\sum_{k'''=1}^{K} \exp\left(\sum_{k''=1}^{K-1} T_{k'''k''} y_{2,k''}\right)}} \\
&= \frac{q_{1,k}\, q_{2,k}}{\sum_{k'=1}^{K} q_{1,k'}\, q_{2,k'}}.
\end{aligned}
$$

Please excuse the gratuitous tick marks; this is only an appendix after all!

In words adding together two unconstrained inputs and then applying the transformation $\sigma \circ T$ is equivalent to multiplying the individual output components and then renormalizing. This simplex operation also defines a technique known as **compositional data analysis** (Aitchison 1982).

Acknowledgements

A very special thanks to everyone supporting me on Patreon: Adam Fleischhacker, Adriano Yoshino, Alessandro Varacca, 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, Charles Shaw, 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, Denis Vlašiček, Doug Rivers, Dr. Jobo, Dr. Omri Har Shemesh, Dylan Maher, Ed Cashin, Edgar Merkle, Eric LaMotte, Ero Carrera, Eugene O'Friel, Eunchong Kang, 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, 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, Justin Bois, Kádár András, Karim Naguib, Karim Osman, Kejia Shi, Kristian Gårdhus Wichmann, Lars Barquist, lizzie , LOU ODETTE, Luís

F, Marcel Lüthi, Marek Kwiatkowski, Mark Donoghoe, Markus P., Márton Vaitkus, Matt Moores, 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, Neel, 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 Ca.

References

License

Aitchison, J. 1982. "The Statistical Analysis of Compositional Data." *Journal of the Royal Statistical Society: Series B (Methodological)* 44 (2): 139–60.

Amari, Shun-ichi, and Hiroshi Nagaoka. 2007. *Methods of Information Geometry*. American Mathematical Soc.

Baker, D. Vincent, and Meguey Baker. 2016. *Apocalypse World*. 2nd ed. Lumpley Games.

Betancourt, Michael. 2012. "Cruising the Simplex: Hamiltonian Monte Carlo and the Dirichlet Distribution." In *Bayesian Inference and Maximum Entropy Methods in Science and Engineering: 31st International Workshop on Bayesian Inference and Maximum Entropy Methods in Science and Engineering*, 1443:157–64. AIP Publishing.

———. 2021. "A Short Review of Ergodicity and Convergence of Markov Chain Monte Carlo Estimators," October.

Csiszár, Imre, and Paul C Shields. 2004. "Information Theory and Statistics: A Tutorial." *Communications and Information Theory* 1 (4): 417–528.

Trefethen, Lloyd N., and David Bau III. 1997. *Numerical Linear Algebra*. Society for Industrial; Applied Mathematics (SIAM), Philadelphia, PA.

Stan
Program 2 homogeneous_simplex.stan

```
data {
  int<lower=1> N;                          // Number of rolls
  array[N] int<lower=1, upper=6> outcome; // Roll outcomes
}

transformed data {
  // Compute outcome totals
  array[6] int counts = rep_array(0, 6);
  for (n in 1:N) {
    counts[outcome[n]] += 1;
  }
}

parameters {
  // General simplex configuration
  simplex[6] q;
}

model {
  q ~ dirichlet(rep_vector(1, 6));
  counts ~ multinomial(q);
}

generated quantities {
  // Posterior predictions
  array[6] int pred_counts = multinomial_rng(q, N);
  real pred_entropy = 0;

  for (k in 1:6) {
    real q_hat = pred_counts[k] * 1.0 / N;
    pred_entropy += - lmultiply(q_hat, q_hat);
  }
}
```

Stan
Program 3 homogeneous_simplex_heterogeneous_retro.stan

```
data {
  int<lower=1> N;                          // Number of rolls
  array[N] int<lower=1, upper=6> outcome; // Roll outcomes

  int<lower=1> N_dice;                           // Number of dice
  array[N] int<lower=1, upper=N_dice> die_idxs; // Die rolled
}

transformed data {
  // Compute outcome totals
  array[6] int counts = rep_array(0, 6);
  array[N_dice] int N_per_die;

  for (n in 1:N) {
    counts[outcome[n]] += 1;
  }

  for (i in 1:N_dice) {
    N_per_die[i] = 0;

    for (n in 1:N) {
      if (die_idxs[n] == i) {
        N_per_die[i] += 1;
      }
    }
  }
}

parameters {
  simplex[6] q;
}

model {
  q ~ dirichlet(rep_vector(1, 6));
  counts ~ multinomial(q);
}

generated quantities {
  array[6] int pred_counts = multinomial_rng(q, N);
  real pred_entropy = 0;

  array[N_dice, 6] real pred_freq_die;
  array[N_dice] real pred_entropy_die;

  array[6] real pred_var_freq;
  real pred_var_entropy;

  for (k in 1:6) {
    real q_hat = pred_counts[k] * 1.0 / N;
    pred_entropy += - lmultiply(q_hat, q_hat);
```

Stan
Program 4 `heterogeneous\_simplex.stan`

```
data {
  int<lower=1> N;                          // Number of rolls
  array[N] int<lower=1, upper=6> outcome; // Roll outcomes

  int<lower=1> N_dice;                            // Number of dice
  array[N] int<lower=1, upper=N_dice> die_idxs; // Die rolled
}

transformed data {
  // Compute outcome totals
  array[6] int counts = rep_array(0, 6);
  array[N_dice, 6] int counts_per_die;
  array[N_dice] int N_per_die;

  for (n in 1:N) {
    counts[outcome[n]] += 1;
  }

  for (i in 1:N_dice) {
    counts_per_die[i] = rep_array(0, 6);
    for (n in 1:N) {
      counts_per_die[die_idxs[n], outcome[n]] += 1;
    }
    N_per_die[i] = sum(counts_per_die[i]);
  }
}

parameters {
  array[N_dice] simplex[6] q;
}

model {
  for (i in 1:N_dice) {
    q[i] ~ dirichlet(rep_vector(1, 6));
    counts_per_die[i] ~ multinomial(q[i]);
  }
}

generated quantities {
  array[6] int pred_counts = rep_array(0, 6);
  real pred_entropy = 0;

  array[N_dice, 6] real pred_freq_die;
  array[N_dice] real pred_entropy_die;

  array[6] real pred_var_freq;
  real pred_var_entropy;

  for (i in 1:N_dice) {
    array[6] int pred_counts_die = multinomial_rng(q[i], N_per_die[i]);
```

Stan
Program 5 `hierarchical\_simplex\_prior\_1.stan`

```
transformed data {
  vector[6] ones = rep_vector(1, 6);
}

generated quantities {
  real omega = abs(normal_rng(0, 1));
  vector[6] q_baseline = dirichlet_rng(7.5 * ones);
  vector[6] q = dirichlet_rng(q_baseline / omega + ones);
}
```

Stan
Program 6 `hierarchical\_simplex\_1.stan`

```
data {
  int<lower=1> N;                          // Number of rolls
  array[N] int<lower=1, upper=6> outcome; // Roll outcomes

  int<lower=1> N_dice;                            // Number of dice
  array[N] int<lower=1, upper=N_dice> die_idxs; // Die rolled
}

transformed data {
  // Compute outcome totals
  array[6] int counts = rep_array(0, 6);
  array[N_dice, 6] int counts_per_die;
  array[N_dice] int N_per_die;

  for (n in 1:N) {
    counts[outcome[n]] += 1;
  }

  for (i in 1:N_dice) {
    counts_per_die[i] = rep_array(0, 6);
    for (n in 1:N) {
      counts_per_die[die_idxs[n], outcome[n]] += 1;
    }
    N_per_die[i] = sum(counts_per_die[i]);
  }
}

parameters {
  array[N_dice] simplex[6] q;
  simplex[6] q_baseline;
  real<lower=0> omega;
}

model {
  vector[6] ones = rep_vector(1, 6);
  vector[6] alpha = q_baseline / omega + ones;

  q_baseline ~ dirichlet(7.5 * ones);
  omega ~ normal(0, 1);

  for (i in 1:N_dice) {
    q[i] ~ dirichlet(alpha);
    counts_per_die[i] ~ multinomial(q[i]);
  }
}

generated quantities {
  simplex[6] q_new = dirichlet_rng(q_baseline / omega + rep_vector(1, 6));

  array[6] int pred_counts = rep_array(0, 6);
```

Stan
Program 7 `hierarchical\_simplex\_prior\_2.stan`

```
transformed data {
  vector[6] ones = rep_vector(1, 6);
}

generated quantities {
  real zeta = abs(normal_rng(0, 5));
  vector[6] q_baseline = dirichlet_rng(7.5 * ones);
  vector[6] delta = dirichlet_rng(ones);
  vector[6] q =  (1    / (1 + zeta)) * q_baseline
              + (zeta / (1 + zeta)) * delta;
}
```

Stan
Program 8 `hierarchical\_simplex\_2.stan`

```
data {
  int<lower=1> N;                          // Number of rolls
  array[N] int<lower=1, upper=6> outcome; // Roll outcomes

  int<lower=1> N_dice;                            // Number of dice
  array[N] int<lower=1, upper=N_dice> die_idxs; // Die rolled
}

transformed data {
  // Compute outcome totals
  array[6] int counts = rep_array(0, 6);
  array[N_dice, 6] int counts_per_die;
  array[N_dice] int N_per_die;

  for (n in 1:N) {
    counts[outcome[n]] += 1;
  }

  for (i in 1:N_dice) {
    counts_per_die[i] = rep_array(0, 6);
    for (n in 1:N) {
      counts_per_die[die_idxs[n], outcome[n]] += 1;
    }
    N_per_die[i] = sum(counts_per_die[i]);
  }
}

parameters {
  array[N_dice] simplex[6] delta;
  simplex[6] q_baseline;
  real<lower=0, upper=1> zeta;
}

transformed parameters {
  array[N_dice] simplex[6] q;
  for (i in 1:N_dice) {
    q[i] = (1  / (1 + zeta)) * q_baseline + (zeta / (1 + zeta)) * delta[i];
  }
}

model {
  vector[6] ones = rep_vector(1, 6);
  q_baseline ~ dirichlet(7.5 * ones);
  zeta ~ normal(0, 5);

  for (i in 1:N_dice) {
    delta[i] ~ dirichlet(ones);
    counts_per_die[i] ~ multinomial(q[i]);
  }
}
```

Stan
Program 9 simu_softmax_1.stan

```
generated quantities {
  vector[6] x;
  vector[6] q;
  {
    vector[6] mu = to_vector(normal_rng(rep_array(0, 6), 0.5));
    vector[6] tau = abs(to_vector(normal_rng(rep_array(0, 6), 0.5)));
    matrix[6, 6] L = lkj_corr_cholesky_rng(6, 4.0);

    x = multi_normal_cholesky_rng(mu, diag_pre_multiply(tau, L));
    q = softmax(x);
  }
}
```

Stan
Program 10 simu_softmax_2.stan

```
transformed data {
  matrix[6, 5] T1 = rep_matrix(0, 6, 5);
  T1[1:5,1:5] = identity_matrix(5);
}

generated quantities {
  vector[5] y;
  vector[6] x;
  vector[6] q;
  {
    vector[5] mu = to_vector(normal_rng(rep_array(0, 5), 0.5));
    vector[5] tau = abs(to_vector(normal_rng(rep_array(0, 5), 0.5)));
    matrix[5, 5] L = lkj_corr_cholesky_rng(5, 4.0);

    y = multi_normal_cholesky_rng(mu, diag_pre_multiply(tau, L));
    x = T1 * y;
    q = softmax(x);
  }
}
```

Stan
Program 11 `simu\_softmax\_3.stan`

```
transformed data {
  matrix[6, 5] T2 = rep_matrix(0, 6, 5);
  T2[1:5,1:5] = identity_matrix(5);
  T2[6,1:5] = rep_row_vector(-1, 5);
}

generated quantities {
  vector[5] y;
  vector[6] x;
  vector[6] q;
  {
    vector[5] mu = to_vector(normal_rng(rep_array(0, 5), 0.5));
    vector[5] tau = abs(to_vector(normal_rng(rep_array(0, 5), 0.5)));
    matrix[5, 5] L = lkj_corr_cholesky_rng(5, 4.0);

    y = multi_normal_cholesky_rng(mu, diag_pre_multiply(tau, L));
    x = T2 * y;
    q = softmax(x);
  }
}
```

Stan
Program 12 simu_softmax_4.stan

```
transformed data {
  matrix[6, 6] P = identity_matrix(6) + (-1.0/6.0) * rep_matrix(1, 6, 6);
  matrix[6, 5] T3 = svd_U(P)[:,1:5];
  matrix[6, 5] T4 = rep_matrix(0, 6, 5);
  matrix[6, 5] T2 = rep_matrix(0, 6, 5);

  T2[1:5,1:5] = identity_matrix(5);
  T2[6,1:5] = rep_row_vector(-1, 5);
  T4 = qr_Q(T2)[:,1:5];
}

generated quantities {
  vector[5] y;
  vector[6] x3;
  vector[6] x4;
  vector[6] q3;
  vector[6] q4;
  {
    vector[5] mu = to_vector(normal_rng(rep_array(0, 5), 0.5));
    vector[5] tau = abs(to_vector(normal_rng(rep_array(0, 5), 0.5)));
    matrix[5, 5] L = lkj_corr_cholesky_rng(5, 4.0);

    y = multi_normal_cholesky_rng(mu, diag_pre_multiply(tau, L));

    x3 = T3 * y;
    q3 = softmax(x3);

    x4 = T4 * y;
    q4 = softmax(x4);
  }
}
```

Stan
Program 13 hierarchical_simplex_prior_3.stan

```
transformed data {
  matrix[6, 6] P = identity_matrix(6) + (-1.0/6.0) * rep_matrix(1, 6, 6);
  matrix[6, 5] T3 = svd_U(P)[:,1:5];
}

generated quantities {
  vector[6] q_baseline;
  vector[6] q;
  {
    vector[5] mu = to_vector(normal_rng(rep_array(0, 5), 0.33));
    vector[5] tau = abs(to_vector(normal_rng(rep_array(0, 5), 0.66)));
    matrix[5, 5] L = lkj_corr_cholesky_rng(5, 4.0);

    vector[5] y = multi_normal_cholesky_rng(mu, diag_pre_multiply(tau, L));

    q_baseline = softmax(T3 * mu);
    q = softmax(T3 * y);
  }
}
```

Stan
Program 14 `hierarchical\_simplex\_3.stan`

```
data {
  int<lower=1> N;                          // Number of rolls
  array[N] int<lower=1, upper=6> outcome; // Roll outcomes

  int<lower=1> N_dice;                            // Number of dice
  array[N] int<lower=1, upper=N_dice> die_idxs; // Die rolled
}

transformed data {
  matrix[6, 6] P = identity_matrix(6) + (-1.0/6.0) * rep_matrix(1, 6, 6);
  matrix[6, 5] T3 = svd_U(P)[:,1:5];

  // Compute outcome totals
  array[6] int counts = rep_array(0, 6);
  array[N_dice, 6] int counts_per_die;
  array[N_dice] int N_per_die;

  for (n in 1:N) {
    counts[outcome[n]] += 1;
  }

  for (i in 1:N_dice) {
    counts_per_die[i] = rep_array(0, 6);
    for (n in 1:N) {
      counts_per_die[die_idxs[n], outcome[n]] += 1;
    }
    N_per_die[i] = sum(counts_per_die[i]);
  }
}

parameters {
  array[N_dice] vector[5] eta; // Non-centered individual parameters
  vector[5] mu;                // Population locations
  vector<lower=0>[5] tau;      // Population scales
  cholesky_factor_corr[5] L;   // Population correlations
}

transformed parameters {
  array[N_dice] simplex[6] q;
  {
    matrix[5, 5] L_Sigma = diag_pre_multiply(tau, L);
    for (i in 1:N_dice) {
      vector[5] y = mu + L_Sigma * eta[i];
      q[i] = softmax(T3 * y);
    }
  }
}

model {
  mu ~ normal(0, 0.33);     // Prior model
```