Pairwise Comparison Modeling

Michael Betancourt

November 2024

Table of contents

A diversity of applications all consider data arising from the comparison of two objects to each other. For example we might be interesting in the outcome of a sporting event between two teams, how well a student can answer a battery of test questions, or even consumer preferences between two products.

In this chapter we'll discuss general strategies for modeling **pairwise comparisons**, the inferential degeneracies that are inherent to these models, and productive strategies for managing those degeneracies. Per tradition these concepts will all be demonstrated in a series of instructional analyses.

1 Modeling Pairwise Comparisons

Given the rich variety of pairwise comparisons that arise in practical applications we will need a reasonably generic vocabulary to describe the common modeling techniques.

To that end consider a collection of I **items**, any pair of which can be compared to each other according to some **criterion**. Items can refer to anything from individual living creatures to individual inanimate objects, collections of those individuals, and more. Comparative criterion include the personal judgement of an individual and the scoring of a direct competition.

In many applications every pair of items defines a valid comparison. Some applications, however, partition the items into two groups and allow comparisons between only an item from one group with an item from the other group; specifically we cannot compare items within each group to each other. For example in a sporting event we might model the points scored by one team as a comparison between that team's offense and their opponent's defense, but no other outcomes directly compare the opposing teams' offensives or defenses to each other. I will refer to these restricted pairings as **bipartite comparisons**.

A comparison between two abstract items will not always result in a quantitative outcome. Just think of all of the poets and songwriters who have struggled to express their feelings about contrasting concepts! To facilitate mathematical modeling we have to restrict consideration to comparative criteria that are quantifiable. In this chapter I will assume a convention where quantitative comparisons yield larger numerical values if the first item in the pair i_1 is superior to the second item i_2 according to the stated criterion.

More formally we will consider only pairwise comparisons that yield a point in some ordered space, with the superiority of item i_1 over item i_2 manifesting as larger values in that space. For example comparisons might be quantified by binary values, integers, or even real numbers. Critically these outputs depend on the arrangement of the items; swapping the roles of item i_1 and item i_2 will in general result in outcomes on the opposite side of the space. For example if i_1 is superior to i_2, and comparisons return larger values, then i_2 will be inferior to i_1 and comparisons will return smaller values.

Our general strategy for consistently modeling pairwise comparisons with ordered outputs will proceed in two stages.

First we equip each item with a real-valued **quality** parameter, $\alpha_i \in \mathbb{R}$, that quantifies how much each item contributes to a particular comparison. A useful convention when working with bipartite comparisons is to denote the quality parameters of each group with a separate variable name, for example α_{i_i} and β_{i_2}. To even further distinguish the two types of items we can also use separate indices, for example α_i and β_j or α_j and β_i.

Next we need to ensure that when comparing item i_1 to item i_2 the larger α_{i_1} is relative to α_{i_2} the more the comparison outcome $y_{i_1 i_2}$ will tend towards larger values. Equivalently the larger α_{i_2} is relative to α_{i_1} the more $y_{i_1 i_2}$ should tend towards smaller values.

One way to guarantee this behavior is to model the comparison outcome $y_{i_1 i_2}$ with an appropriate location family of probability density functions,

$$p(y_{i_1 i_2} \mid \mu_{i_1 i_2}, \phi),$$

and then replace the location parameter $\mu_{i_1 i_2}$ with the output of a monotonically-increasing function of the difference in quality parameters,

$$\mu_{i_1 i_2} = f(\alpha_{i_1} - \alpha_{i_2}).$$

The larger α_{i_1} is relative to α_{i_2} the larger the difference between them will be, the larger the output of the function f will be, and the more the comparison outcomes will concentrate on larger values as desired.

2 Coupling Outcome Location and Item Qualities

The general construction of pairwise comparison models presented in Section 1 allows for a variety of sophisticated behaviors. Perhaps the most important of these is the functional relationship that couples the location of the outcome model to the difference of item qualities,

$$\mu_{i_1 i_2} = f(\alpha_{i_1} - \alpha_{i_2}).$$

Often it is more productive to build up the function f as a composition of multiple component functions,

$$\mu_{i_1 i_2} = f_J \circ \cdots \circ f_1(\alpha_{i_1} - \alpha_{i_2}),$$

with each moderating the coupling in different ways.

In this section we'll discuss a few functional behaviors that are particularly useful for many pairwise comparison applications.

2.1 Constraining Functions

By construction item qualities are real-valued, and so too are their differences. If the location configuration of the outcome model is not real-valued then we cannot directly relate it to those real-valued differences. In particular if the location configuration is constrained then it will not be immediately compatible to the item differences. Instead we need to introduce an intermediate function to corral any inconsistent values.

First, however, consider a pairwise comparison that results in a real-valued outcome. In this case we might consider a normal outcome model

$$\text{normal}(y_{i_1 i_2} \mid \mu_{i_1 i_2}, \sigma).$$

Because the location $\mu_{i_1 i_2}$ is unconstrained we can directly relate it to the item differences with an identify function,

$$\mu_{i_1 i_2} = \text{Id}(\alpha_{i_1} - \alpha_{i_2}) = \alpha_{i_1} - \alpha_{i_2}.$$

Here the quality of the first item α_{i_1} directly translates the location of the outcome model upwards while the quality of the second item α_{i_2} translates the location downwards.

Now consider a pairwise comparison that yields non-negative, integer-valued outcomes $y_{i_1 i_2} \in \mathbb{N}$. Here we might reach for a Poisson outcome model,

$$\text{Poisson}(y_{i_1 i_2} \mid \lambda_{i_1 i_2}),$$

with a positive location configuration $\lambda_{i_1 i_2}$. In order to relate the unconstrained item quality differences to a valid location configuration we need a monotonic function that squeezes an entire real line into a positive real line,

$$f : \mathbb{R} \rightarrow \mathbb{R}^{+}.$$

One possibility is the exponential function,

$$\begin{aligned} \lambda_{i_1 i_2} &= \exp(\alpha_{i_1} - \alpha_{i_2}) \\ &= \exp(\alpha_{i_1}) \, \exp(-\alpha_{i_2}) \\ &= \exp(\alpha_{i_1}) \frac{1}{\exp(\alpha_{i_2})}. \end{aligned}$$

In this case the item quality differences don't translate the location configuration but rather *scale* it multiplicatively.

Similarly when working with binary outcomes $y_{i_1 i_2} \in \{0, 1\}$ we can always assume a Bernoulli model

$$\text{Bernoulli}(y_{i_1 i_2} \mid p_{i_1 i_2}).$$

We can then use a logistic function

$$\text{logistic} : \mathbb{R} \rightarrow [0, 1]$$

to map item quality differences into unit-interval-valued probability configurations,

$$p_{i_1 i_2} = \text{logistic}(\alpha_{i_1} - \alpha_{i_2}).$$

That said the logistic function is not the only possibility here. For example we could use the error function or really the cumulative distribution function for any probability distribution defined on a real line.

2.2 Baseline-Inducing Functions

In many applications it is useful to have item qualities modify a common *baseline* behavior. One way to achieve this is with a translation function that centers differences in item qualities around a baseline value,

$$t_{\eta}(\alpha_{i_1} - \alpha_{i_2}) = \eta + \alpha_{i_1} - \alpha_{i_2}.$$

For example we can model real-valued comparison outcomes with the same normal model that we considered above

$$\text{normal}(y_{i_1 i_2} \mid \mu_{i_1 i_2}, \sigma)$$

but with the location configuration

$$\begin{aligned} \mu_{i_1 i_2} &= t_{\eta}(\alpha_{i_1} - \alpha_{i_2}) \\ &= \eta + \alpha_{i_1} - \alpha_{i_2}. \end{aligned}$$

Here η models the baseline location which can be translated up or down by the qualities of the items being compared.

2.3 Discrimination-Inducing Functions

Another useful feature of a pairwise comparison model is the ability to moderate exactly how much the location configuration is coupled to the item quality differences. One way to accomplish this is with a scaling function,

$$s_{\gamma}(\alpha_{i_1} - \alpha_{i_2}) = \gamma \cdot (\alpha_{i_1} - \alpha_{i_2}).$$

Large, positive values of γ enhance the coupling while smaller values suppress it. As γ approaches zero the location configuration becomes completely uncoupled to the item qualities. Because of this behavior γ is known as a **discrimination** variable.

Negative values of γ invert the relationship between the item quality differences and the location configuration so that, for example, $\alpha_{i_1} > \alpha_{i_2}$ results in smaller outcomes that suggest the superiority of item i_2 over item i_1. This behavior can be useful when the nominal comparison criterion might actually be reversed.

That said negative values of γ also allow for some redundancy in the resulting pairwise comparison model. Flipping the sign of the discrimination and item qualities at the same time results in the same output,

$$\gamma \cdot (\alpha_{i_1} - \alpha_{i_2}) = (-\gamma) \cdot ((-\alpha_{i_1}) - (-\alpha_{i_2})),$$

and hence the same model for the comparison outcomes. This redundancy then manifests as degenerate likelihood functions.

Beyond cases where the nominal comparison criterion is suspect the best practice is to constrain the sign of a discrimination parameter to positive values. This eliminates the sign redundancy and any resulting inferential degeneracies.

All of this said, a homogeneous discrimination model is not all that useful in practice. Scaling the differences in item qualities is most useful when the discrimination varies across different circumstances, with outcomes more strongly informed by the item qualities for some observations but more weakly informed by the item qualities for others. Exactly how γ should vary will depend on the details of a particular application; we'll see some examples in Section 6.4 and Section 6.5.

2.4 Combining Functional Behaviors

As useful as these individual functional behaviors are on their own they can become especially useful when combined together.

To demonstrate let's reconsider pairwise comparisons that yield non-negative, integer-valued outcomes. Once again we'll appeal to a Poisson model for the outcomes

$$\text{Poisson}(y_{i_1 i_2} \mid \lambda_{i_1 i_2}),$$

but now we'll allow for more sophisticated functional relationships between the location configuration $\lambda_{i_1 i_2}$ and the item difficulties.

For example if we compose a translation function with an exponential function then the latter will define a baseline while the latter will ensure positive location configurations,

$$\begin{aligned} \lambda_{i_1 i_2} &= \exp \circ t_{\eta}(\alpha_{i_1} - \alpha_{i_2}) \\ &= \exp(\eta + \alpha_{i_1} - \alpha_{i_2}) \\ &= \exp(\eta) \cdot \exp(\alpha_{i_1}) \cdot \exp(-\alpha_{i_2}) \\ &= \exp(\eta) \cdot \exp(\alpha_{i_1}) \cdot \frac{1}{\exp(\alpha_{i_2})}. \end{aligned}$$

In this case $\exp(\eta)$ defines a baseline location that is multiplicatively perturbed by the quality parameters of the items being compared.

We might even compose a scaling function, a translation function, and an exponential function together at the same time,

$$\begin{aligned} \lambda_{i_1 i_2} &= \exp \circ t_{\eta} \circ s_{\gamma}(\alpha_{i_1} - \alpha_{i_2}) \\ &= \exp \circ t_{\eta}(\gamma \cdot (\alpha_{i_1} - \alpha_{i_2})) \\ &= \exp\left(\eta + \gamma \cdot (\alpha_{i_1} - \alpha_{i_2})\right) \\ &= \exp\left(\eta\right)\, \exp\left(\gamma \cdot (\alpha_{i_1} - \alpha_{i_2})\right). \end{aligned}$$

As before $\exp(\eta)$ defines a baseline location, but now the multiplicative perturbations defined by the item quality parameters are moderated by γ. In particular as $\gamma \to 0$ the output of any pairwise comparison shrinks towards the baseline behavior regardless of the qualities of the items being compared.

3 Notable Pairwise Comparisons Models

While pairwise comparison modeling is broadly applicable there are two applications that dominate the statistics literature, both of which consider only the special case of binary outcomes. In this case section we'll review these applications in the context of the more general techniques.

3.1 Bradley-Terry Model

One common pairwise comparison arises in competitions, with a game between two players or teams resulting in an unambiguous winner and loser. We can model these competitions as pairwise comparisons with the binary outcome $y_{i_1 i_2} = 1$ denoting a win for the i_1th competitor and $y_{i_1 i_2} = 0$ a win for the i_2th competitor. The items then model the competitors while the item qualities model their skills.

Wrapping the difference in item qualities in a logistic functions gives the pairwise comparison model

$$\text{Bernoulli}(y_{i_1 i_2} \mid p_{i_1 i_2})$$

with

$$\begin{aligned}
p_{i_1 i_2} &= \text{logistic}(\alpha_{i_1} - \alpha_{i_2}) \\
&= \frac{1}{1 + \exp(-(\alpha_{i_1} - \alpha_{i_2}))} \\
&= \frac{1}{1 + \exp(-\alpha_{i_1}) \, \exp(\alpha_{i_2})} \\
&= \frac{\exp(\alpha_{i_1})}{\exp(-\alpha_{i_1}) + \exp(\alpha_{i_2})}.
\end{aligned}$$

This model is commonly known as the **Bradley-Terry model** (Bradley and Terry (1952)), although historically it has been independently derived multiple times from multiple different perspectives (Hamilton, Tawn, and Firth (2023)).

One nice feature of this model is the symmetry between the competitors. If we swap the order of the competitors then the probability that the i_2th competitor wins is

$$\begin{aligned}
p_{i_2 i_1} &= \text{logistic}(\alpha_{i_2} - \alpha_{i_1}) \\
&= \frac{1}{1 + \exp(-(\alpha_{i_2} - \alpha_{i_1}))} \\
&= \frac{\exp(-(\alpha_{i_1} - \alpha_{i_2})}{1 + \exp(-(\alpha_{i_1} - \alpha_{i_2}))} \\
&= \frac{1 + \exp(-(\alpha_{i_1} - \alpha_{i_2}) - 1}{1 + \exp(-(\alpha_{i_1} - \alpha_{i_2}))} \\
&= \frac{1 + \exp(-(\alpha_{i_1} - \alpha_{i_2})}{1 + \exp(-(\alpha_{i_1} - \alpha_{i_2}))} - \frac{1}{1 + \exp(-(\alpha_{i_1} - \alpha_{i_2}))} \\
&= 1 - p_{i_1 i_2}.
\end{aligned}$$

Consequently modeling a win for either team will always yield the same inferences for their relative skills.

This model is also the basis for the Elo rating system (Elo (1967)) first developed to rank competitive chess players but since applied to all kinds of repeated competitions. The original Elo rating system used this pairwise comparison model to derive iterative updates to the competitor skills based on the initial skills and the outcome of a competition.

One limitation of considering only the winner of a competition is that the resulting analysis is largely insensitive to the details of competition itself. In particular any analysis using this model will be able to predict the winner of future competitions only so well.

In practice we can often use more general pairwise comparison models to capture these dynamics and inform more versatile predictions. For example in some competitions we can model the scores of two competitors, with item qualities capturing for their distinct offensive and defensive skills. In many cases we can even model the outcome of individual plays and then integrate these components models together into a detailed model of how a competition progresses.

When analyzing a baseball game, for instance, we could model only the winner or we could model the scores of each team. We could even go deeper and model the constituent competitions between individual batters and pitchers, balls in play and team defenses, and so on.

3.2 Item Response Theory

Another common pairwise comparison arises in standardized testing, also known as **item response theory** (Baker (2001)). These applications consider bipartite comparisons, with one item in each comparison representing a question on a test and the other representing a student attempting to answer that question. The item qualities then model the difficulty of the question and the test-taking ability of the student, respectively, while the binary outcome quantifies whether a not the student correctly answers the question.

The key difference between item response theory models and Bradley-Terry models is the asymmetry in the items. A Bradley-Terry model allows for comparisons between any two competitors, but an item response theory model considers only comparisons between one question and one student.

The simplest item response theory model takes the same form as the Bradley-Terry model with

$$\text{Bernoulli}(y_{ji} \mid p_{ji})$$

and

$$p_{ji} = \text{logistic}(\alpha_j - \beta_i),$$

where i indexes each question and j indexes each student. Historically this model is also known as the Rasch model (Rasch (1960)). In more contemporary literature it is often referred to as a 1PL model, with the "L" referring to the logistic mapping between item quality differences and outcome probabilities and the "1P" referring to the one parameter used to model the behavior of each question.

Adding a discrimination parameter for each question,

$$p_{ji} = \text{logistic}(\gamma_i \cdot (\alpha_j - \beta_i)),$$

allows the model to account for poorly designed questions that might be less sensitive to student abilities. This is also known as a 2PL model given the two parameters needed to model each question.

Lastly a 3PL model for each question introduces lower bounds to the correct answer probabilities,

$$p_{ji} = \lambda_i + (1 - \lambda_i) \cdot \text{logistic}(\gamma_i \cdot (\alpha_j - \beta_i)).$$

As we saw in the mixture modeling chapter this mixture of Bernoulli probability configurations can also be interpreted as a mixture of separate Bernoulli models,

$$\begin{aligned} p(y_{ji}) = & \quad\quad \lambda_i \, \text{Bernoulli}(y_{ji} \mid 1) \\ & + (1 - \lambda_i) \, \text{Bernoulli}(y_{ji} \mid \text{logistic}(\gamma_j \cdot (\alpha_j - \beta_i))). \end{aligned}$$

In other words the 3PL model implies that students have some fixed probability of guessing each answer correctly, independent of their test-taking abilities.

4 Managing Inferential Degeneracies

Unfortunately pairwise comparison models aren't as well-posed as we might initially have hoped. By construction observed outcomes inform only the differences between item qualities and not their definite values. Because of this the item qualities cannot be inferred independently of each other and, in a very real sense, they cannot be meaningfully interpreted independently of each other.

In other words pairwise comparison models are inherently redundant, and this redundancy complicates practical implementations.

4.1 The Fundamental Redundancy of Item Qualities

Pairwise comparison models are inherently non-identified, with any common translation to the item qualities yielding the same location configurations,

$$\begin{aligned} \mu_{i_1 i_2}(\alpha_{i_1} + \zeta, \alpha_{i_2} + \zeta) &= f((\alpha_{i_1} + \zeta) - (\alpha_{i_2} + \zeta)) \\ &= f(\alpha_{i_1} - \alpha_{i_2} + \zeta - \zeta) \\ &= f(\alpha_{i_1} - \alpha_{i_2}) \\ &= \mu_{i_1 i_2}(\alpha_{i_1}, \alpha_{i_2}). \end{aligned}$$

Consequently every likelihood function realized from one of these models will be invariant to translations of the item qualities. This invariance in turn traces out non-compact inferential degeneracies that frustrate computation (Figure 1b).

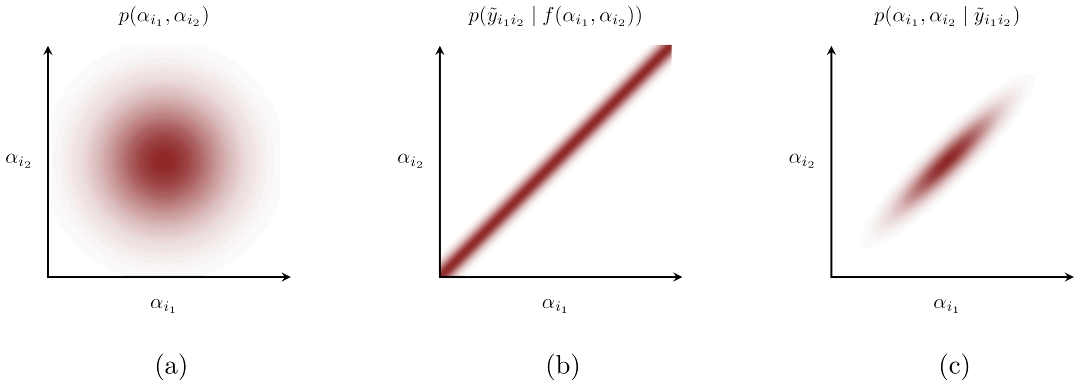


Figure 1: The item qualities in a naive pairwise comparison model are non-identified, resulting in (b) likelihood functions with non-compact degeneracies. (a) An informative prior model suppresses extreme qualities, resulting in a (c) much better behaved posterior distribution. Note, however, that a residual degeneracy remains within the containment of the prior model.

4.2 Informative Prior Modeling

Within a Bayesian analysis we can use an informative prior model to at least tame these non-compact degeneracies and contain the corresponding posterior distributions away from infinity (Figure 1). While inferential degeneracies can still persist within the breadth of a prior model and hinder computational efficiency this containment at least makes accurate computational possible.

Because we cannot interpret the item qualities independently of each other any prior model over the α_i will be somewhat arbitrary. What really determines a whether a prior model is useful or not is how consistent the *induced* behavior for the differences in item qualities, and ultimately the corresponding outcome behaviors, is with the available domain expertise.

For example the independent component prior model over the item qualities,

$$p(\alpha_1, \ldots, \alpha_I) = \prod_{i=1}^{I} \text{normal}(\alpha_i \mid 0, s_i),$$

implies a pushforward prior model for any item quality difference,

$$p(\alpha_{i_1} - \alpha_{i_2}) = \text{normal}\left(\alpha_{i_1} - \alpha_{i_2} \mid 0, \sqrt{s_{i_1}^2 + s_{i_2}^2}\right).$$

If this pushforward behavior is consistent with our domain expertise then the initial prior model will be satisfactory.

That said many distinct prior models will in general share the same pushforward behavior. In particular any independent component prior model of the form

$$p(\alpha_1, \dots, \alpha_I) = \prod_{i=1}^{I} \text{normal}(\alpha_i \mid m, s_i)$$

implies the same pushforward behavior,

$$p(\alpha_{i_1} - \alpha_{i_2}) = \text{normal}\left(\alpha_{i_1} - \alpha_{i_2} \mid 0, \sqrt{s_{i_1}^2 + s_{i_2}^2}\right).$$

Consequently any value of m defines an equally viable prior model.

4.3 Anchoring Item Qualities

A more effective approach than prior containment is to remove a degree of freedom from the pairwise comparison model and eliminate the underlying redundancy entirely.

4.3.1 Single Anchor

For example consider picking an arbitrary item and then subtracting the distinguished item quality from all of the item qualities,

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

After this transformation the item quality differences are unchanged,

$$\begin{aligned} \mu_{i_1 i_2}(\delta_{i_1}, \delta_{i_2}) &= \mu_{i_1 i_2}(\alpha_{i_1} - \alpha_{i'}, \alpha_{i_2} - \alpha_{i'}) \\ &= f((\alpha_{i_1} - \alpha_{i'}) - (\alpha_{i_2} - \alpha_{i'})) \\ &= f(\alpha_{i_1} - \alpha_{i_2}) \\ &= \mu_{i_1 i_2}(\alpha_{i_1}, \alpha_{i_2}), \end{aligned}$$

but the distinguished item quality is now **anchored** to zero,

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

Because $\delta_{i'}$ is now fixed we have to infer only the $I - 1$ remaining quality parameters.

The δ_i can be interpreted as the item qualities *relative* to the quality of the anchored item, with $\delta_i > 0$ implying that the ith item is superior to the anchored item and $\delta_i < 0$ implying that the ith item is inferior to the anchored item. This interpretation also facilitates prior modeling, allowing us to reason about the relative item qualities directly instead of having to analyze indirect pushforward behavior.

4.3.2 Multiple Anchors

All of this said, a single anchor isn't always enough. Recall that once we fix the quality of the i'th item to zero the residual quality parameters are interpreted relative to the quality of that anchored item. This immediately identifies the δ_i for any item that is directly compared to the i'th item. These identified parameters in turn identify any remaining items that are directly compared to them, and so on. Any items which are not at least indirectly compared to the anchored item, however, remain invariant to translations!

We can formalize the connectivity of the comparisons, and hence how many item qualities we need to anchor, with a little bit of graph theory (Clark and Holton (1991)). Any collection of observed pairwise comparisons defines a graph, with each item defining a node and the number of pairwise comparisons between any two items defining a weighted, undirected edge between the corresponding nodes (Figure 2).

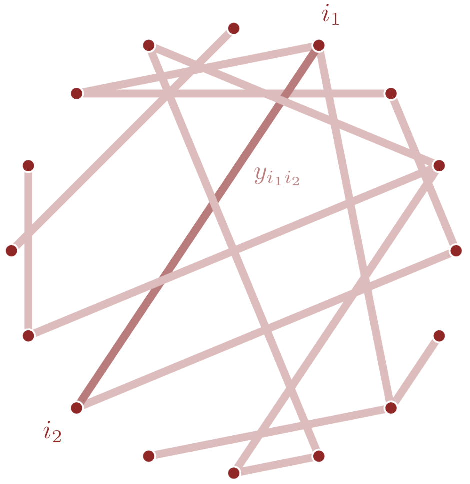


Figure 2: Any collection of pairwise comparisons defines an undirected graph, with the items being compared defining nodes and the comparisons themselves defining edges between the nodes.

Two items have been directly compared if there is an edge between their nodes; two items have been indirectly compared if their nodes can be connected by a *path* of edges. More formally these interacting items form the connected subgraphs, or **components**, of the full graph (Figure 3).

From an inferential perspective the behavior of items in one connected component are independent of the behavior of items in any other connected component. In other words the item

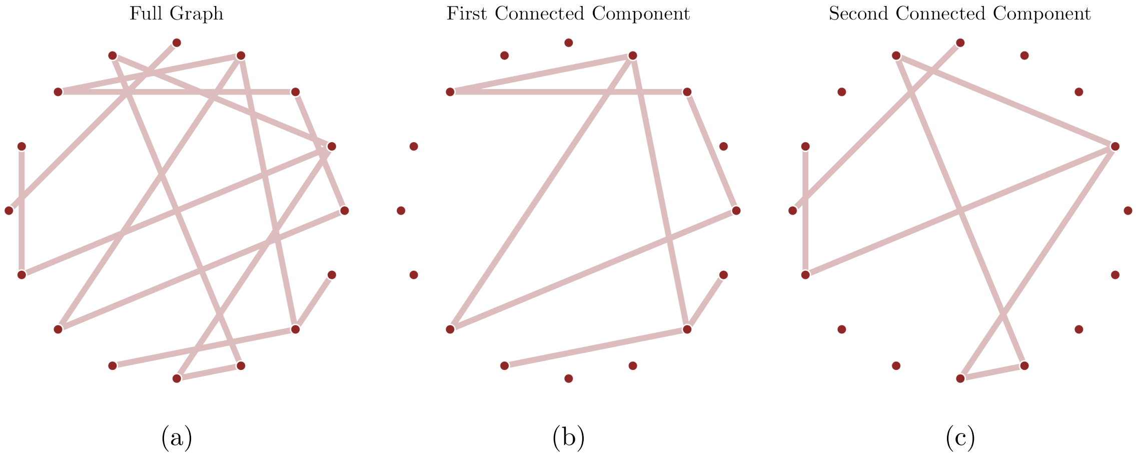


Figure 3: When some nodes are isolated from the others (a) a graph decomposes into (b, c) multiple connected components. In a graph defined by pairwise comparisons the item qualities in any one component are independent of the item qualities in any other component.

qualities are only *locally* interpretable within the encompassing connected component. We cannot learn the relationship between two items that have no connection to each other!

For example while we can learn the relative test-taking abilities of students who taken the same test we cannot learn the relative test-taking abilities of students who have taken completely different tests. Similarly when analyzing historical competition data we can learn how skilled more modern competitors are to older competitors only if we have enough intermediate competitions to bridge them together.

Mathematically anchoring the quality parameter of an item eliminates the redundancy only of the item qualities within the same component. The redundancy of item qualities in other components remains. To completely eliminate the redundancy of a pairwise component model and avoid the subsequent inferential degeneracies we have to anchor one item quality *in each connected component*.

Fortunately the translation of a collection of items into a graph and computation of the resulting connected components is relatively straightforward to implement in practice (Cormen et al. (2022)). We'll review a demonstration in Section 6.2.

That said we have to be careful with what inferences we construct from a pairwise comparison model with more than one anchored item quality. For example if we anchor the quality of items i' and i'' in two different connected components then we will have both

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

and

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

and consequently

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

Naively this might be interpreted as the quality of the two anchored items being identical, even thought those relative quantities are actually *incomparable* with each other.

In theory we could obstruct ill-defined comparisons by anchoring the items in the second connected component to a different value,

$$\begin{aligned}\delta_{i'} &= \alpha_{i'} - \alpha_{i'} = 0 \\ \delta_{i''} &= \alpha_{i''} - (\alpha_{i''} - \phi) = \phi.\end{aligned}$$

Unfortunately this just reintroduces the initial degeneracy: ϕ is completely uninformed by the observed data and we would have to rely on whatever domain expertise has been encoded in the prior model to inform its posterior behavior.

Perhaps the safest approach is to simply analyze the pairwise comparisons in each connected components separately from each other. When analyzing a single connected component at a time we will only ever need to anchor one item. Moreover inferences and predictions for all possible pairwise comparisons within each analysis will always be well-defined. The limitation with this approach, however, is that the full data set will no longer able to inform any shared behaviors, such as common baselines or discriminations.

Regardless of the approach we take *we* are ultimately responsible for avoiding ill-defined inferences or predictions for any comparison of items in different connected components.

4.4 Coupling Function Redundancies

Every function that couples the item quality differences to the outcome location will suffer the fundamental redundancy of the item qualities. Some coupling functions, however, can introduce new redundancies and inferential degeneracies with them.

Consider, for example, the translation function

$$t_{\eta}(\alpha_{i_1} - \alpha_{i_2}) = \eta + \alpha_{i_1} - \alpha_{i_2}.$$

Mathematically the baseline suffers from an additive redundancy with the item quality differences,

$$\begin{aligned}t_{\eta}(\alpha_{i_1} - \alpha_{i_2}) &= \eta + \alpha_{i_1} - \alpha_{i_2} \\ &= \eta + 0 + \alpha_{i_1} - \alpha_{i_2} \\ &= \eta + \zeta - \zeta + \alpha_{i_1} - \alpha_{i_2} \\ &= (\eta + \zeta) + (\alpha_{i_1} - \alpha_{i_2} - \zeta) \\ &= t_{\eta+\zeta}(\alpha_{i_1} - \alpha_{i_2} - \zeta).\end{aligned}$$

When items appear are both side of the comparisons, however, there's no way to factor that offset ζ into the individual item qualities. In this case the baseline η is identified by the observed behavior common to all comparisons.

With bi-partite comparisons, however, we can factor any offset ζ into either of the contrasting item qualities

$$\begin{aligned} t_{\eta}(\alpha_i - \beta_j) &= t_{\eta+\zeta}(\alpha_i - \beta_j - \zeta) \\ &= t_{\eta+\zeta}((\alpha_i - \zeta) - \beta_j) \\ &= t_{\eta+\zeta}(\alpha_i - (\beta_j + \zeta)). \end{aligned}$$

Because we can absorb the offset into either group of item qualities we cannot eliminate this redundancy by anchoring one type of item alone. Instead we have to anchor both types of items.

When the observed comparisons define a single connected component we need to set a distinguished $\alpha_{i'}$ and a distinguished $\beta_{j'}$ to zero at the same time. More generally we need to fix one of each type of item quality within each connected component.

For example if we're modeling points scored in the games of a sports league as

$$p(y_{ij}) = \text{Poisson}(y_{ij} \mid \exp(\eta + \alpha_i^{\text{offense}} - \alpha_j^{\text{defense}}))$$

then we would need to anchor one team's offensive skill to zero and one team's defensive skill to zero so that the remaining skills are defined relative to the anchored skills. In general we can anchor the offensive and defensive skills of the same team or even different teams. Either way we can then interpret the baseline η as modeling the score in a, possibly hypothetical, game between the anchored offense and the anchored defense.

Similarly a scaling function

$$s_{\gamma}(\alpha_{i_1} - \alpha_{i_2}) = \gamma \cdot (\alpha_{i_1} - \alpha_{i_2}).$$

suffers from a multiplicative redundancy,

$$\begin{aligned} s_{\gamma}(\alpha_{i_1} - \alpha_{i_2}) &= \gamma \cdot (\alpha_{i_1} - \alpha_{i_2}) \\ &= \gamma \cdot 1 \cdot (\alpha_{i_1} - \alpha_{i_2}) \\ &= \gamma \cdot \frac{\rho}{\rho} \cdot (\alpha_{i_1} - \alpha_{i_2}) \\ &= (\rho \cdot \gamma) \cdot \left(\frac{\alpha_{i_1} - \alpha_{i_2}}{\rho} \right) \\ &= s_{\rho \cdot \gamma} \left(\frac{\alpha_{i_1} - \alpha_{i_2}}{\rho} \right). \end{aligned}$$

Because scaling the item qualities does not obstruct translations,

$$\frac{\alpha_{i_1} - \alpha_{i_2}}{\rho} = \frac{(\alpha_{i_1} + \zeta) - (\alpha_{i_2} - \zeta)}{\rho},$$

anchoring the item qualities will not eliminate this scale redundancy. *Any* proportional increase in the discrimination parameter γ can be compensated by a proportional decrease in the item qualities and vice versa.

An informative prior model on the item qualities and discrimination parameters does limit the scope of this redundancy, and hence the resulting inferential degeneracies. That said constraints on the qualities and discriminations can interact in subtle ways that complicate principled prior modeling. By far the most effective strategy is to eliminate the redundancy entirely.

If the discrimination is homogeneous across all comparisons then we can always fix γ to one without any loss of generality. On the other hand if the discrimination is heterogeneous, with a separate γ_k for each distinct context, then fixing one of the $\gamma_{k'}$ will eliminate the redundancy in the pairwise comparison model. Specifically if we set

$$\gamma_{k'} = 1$$

then the remaining discrimination parameters end up quantifying scaling relative to the anchored context,

$$\kappa_k = \frac{\gamma_k}{\gamma_{k'}}.$$

4.5 Trouble With Dominating Items

Depending on the structure of the observational space pairwise comparison models can also suffer from some more circumstantial inferential degeneracies.

Consider for example comparison outcomes that are bounded above,

$$y_{i_1 i_2} \le y_{\max}.$$

If the observed comparisons to one particular item i' always saturate that maximum value,

$$y_{i' i_2} = y_{\max},$$

then the observations will not be able to determine just how much better that dominating item is from the others.

The saturated observations will be able to lower bound the definite or relative quality of the dominating item, but they will be indifferent to the remaining, arbitrarily large values. This then results in a non-compact degeneracy extending out towards positively-infinite values.

Similarly if the observed comparisons are bounded below,

$$y_{i_1 i_2} \geq y_{\min},$$

and the observed comparisons to one dominating item always saturate that minimum value,

$$y_{i' i_2} = y_{\min},$$

then the data will not be able to differentiate between arbitrarily large but negative values of the definite quality $\alpha_{i'}$ or even the relative quality $\delta_{i'}$. In this case the likelihood function will exhibit a non-compact degeneracy that extends out towards negatively-infinite values.

For both cases a reasonably informative prior model will be able contain these inferential degeneracies and prevent them from propagating to the posterior distribution. An even better strategy is to design more informative comparisons that can inform just how much better every item is from its peers.

For example in the context of sporting events analyzing score differentials is always more informative than analyzing winners. Similarly in the context of standardized testing one might be able to learn more from how long it takes students to reach the right answer than one can from whether or not the students report the right answer eventually.

5 Ranking Items

Because item qualities can be meaningfully interpreted only relative to one other they inform only a limited range of behaviors. One particularly useful behavior that they can inform are *rankings* of the items being compared.

Formally any configuration of item qualities,

$$(\alpha_1, \ldots, \alpha_i, \ldots, \alpha_I) \in A^I,$$

implies one of $I!$ possible rankings,

$$(r_1, \ldots, r_i, \ldots, r_I) \in R,$$

where the items are ordered by their relative qualities,

$$\alpha_{r_1} < \ldots < \alpha_{r_i} < \ldots < \alpha_{r_I}.$$

In fact this relationship defines a bijective function from the space of item qualities to the space of the possible orderings of the I items,

$$o : A^I \to R.$$

Theoretically pushing forward a posterior distribution over the item qualities π along this function will define a posterior distribution $o_{*}\pi$ that quantifies the consistent item rankings. The challenge in practice, however, is that the space of ranking R is difficult to navigate. In particular it is not immediately clear how we might be able to construct summaries of $o_{*}\pi$ that are straightforward to communicate.

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

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

Unfortunately even if a unique mode exists actually finding it is typically intractable. In general we will not be able to compute the atomic allocations $o_{*}\pi(\{r\})$ in closed form and estimation of all of these probabilities quickly becomes expensive. Consequently exhaustively searching through all $I!$ atomic allocations is almost always be too expensive. Moreover without gradient information we have no way to guide a more efficient search of the ranking space.

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

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

Conveniently there are a variety of useful distance functions on R that we could use to inform such point summaries, and in theory we can estimate the necessary expectation values with Markov chain Monte Carlo. The minimization over all possible candidate rankings $r \in R$, however, suffers from the same problems as above.

Because of these computational issues we usually need to appeal to more heuristic summaries in practice. For instance we can always rank the items by the posterior expectation values of their individual qualities,

$$\mathbb{E}_{\pi}[\alpha_{r_{1}}] < ... < \mathbb{E}_{\pi}[\alpha_{r_{i}}] < ... < \mathbb{E}_{\pi}[\alpha_{r_{I}}].$$

When considering only two items at a time we can can also consult the posterior probability that one skill surpasses the other,

$$\pi(\{\alpha_{i_{1}} > \alpha_{i_{2}}\}).$$

One nice feature of this pairwise inferential comparison is that it accounts for any coupled uncertainties between the two item qualities.

These binary inferential comparisons can also be used to construct another heuristic ranking. The posterior probability that the quality of each item is larger than the quality of all other items,

$$p_{i'} = \pi(\{\alpha_{i'} > \alpha_{1} \, \text{and} \, ... \, \text{and} \, \alpha_{i'} > \alpha_{i} \, \text{and} \, ... \, \text{and} \, \alpha_{i'} > \alpha_{I}\}),$$

can be used to assign a top rank,

$$r_I = \underset{i \in I}{\text{argmax}} \, p_i.$$

At this point we can compute the posterior probability that the quality of each remaining item is larger than the quality of all of the other remaining items and assign the second-highest rank based on the largest of these probabilities. We can then fill out all of the remaining rankings by iterating this procedure until only one item is left to occupy the lowest rank.

The main limitations with this approach is that it requires estimating so many probability allocations that we can exhaust the available computational resources, especially if there are a lot of items. The computation is even more demanding if we want to ensure robust rankings that are not corrupted by computational artifacts. This requires that the Markov chain Monte Carlo error for each posterior probability estimate is smaller than any of the differences between it and the other posterior probability estimates. Implementing robust rankings in practice requires not only carefully monitor the error of all of the posterior probability estimates but also potentially running additional or longer Markov chains to reduce any errors as needed.

6 Demonstrations

With the foundations laid let's investigate the implementation of pairwise comparison models and their inferential behaviors with a series of increasingly more sophisticated example analyses.

6.1 Setup

First and foremost we have to set up or local `R` environment.

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

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

library(colormap)
```

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

6.2 Graph Analysis Utilities

Next we'll need some utilities for analyzing the graphs defined by a collection of pairwise comparisons. There are a variety of dedicated graph analysis packages available in `R`, but here I'll code up basic implementations of the specific tools that we'll need.

Firstly we'll need a function that builds up a weighted **adjacency matrix** from a list of pairwise comparisons. The (i_1, i_2)-th element of a weighted adjacency matrix is the number of comparisons between item i_1 and item i_2. In particular if the (i_1, i_2)-th element is zero then the two items have not been directly compared.

```
build_adj_matrix <- function(N_items, N_comparisons, idx1, idx2) {
  if (min(idx1) < 1 | max(idx1) > N_items) {
    stop("Out of bounds idx1 values.")
  }
  if (min(idx2) < 1 | max(idx2) > N_items) {
    stop("Out of bounds idx2 values.")
  }

  adj <- matrix(0, nrow=N_items, ncol=N_items)

  # Compute directed edges
  for (n in 1:N_comparisons) {
    adj[idx1[n], idx2[n]] <- adj[idx1[n], idx2[n]] + 1
  }

  # Compute symmetric, undirected edges
  for (p1 in 1:(N_items - 1)) {
    for (p2 in (p1 + 1):N_items) {
      N1 <- adj[p1, p2]
      N2 <- adj[p2, p1]
      adj[p1, p2] <- N1 + N2
      adj[p2, p1] <- N1 + N2
    }
  }

  (adj)
}
```

Adjacency matrices are extremely useful for implementing many graph theory operations in practice. For example we can use a weighed adjacency matrix to visualize the comparison graph, with circles representing each node and lines representing the edges. Here the color of the lines visualizes the weight of each edge, equivalently how many comparisons have been made between any two items.

```
plot_undir_graph <- function(adj) {
  if (!is.matrix(adj)) {
    stop("Input adj is not a matrix.")
  }
  if (nrow(adj) != ncol(adj)) {
    stop("Input adj is not a square matrix.")
  }

  nom_colors <- c("#DCBCBC", "#C79999", "#B97C7C", "#A25050")
  edge_colors <- colormap(colormap=nom_colors, nshades=100)

  I <- nrow(adj)
  delta <- 2 * pi / I
  thetas <- (0:(I - 1)) * delta + pi / 2
  R <- 1

  xs <- R * cos(thetas)
  ys <- R * sin(thetas)

  plot(0, type='n', axes=FALSE, ann=FALSE,
       xlim=c(-1.1 * R, +1.1 * R), ylim=c(-1.1 * R, +1.1 * R))

  cthetas <- seq(0, 2 * pi + 0.05, 0.05)
  cxs <- R * cos(cthetas)
  cys <- R * sin(cthetas)
  lines(cxs, cys, lwd=3, col="#DDDDDD")

  max_adj <- max(adj)
  for (i1 in 1:(I - 1)) {
    for (i2 in (i1 + 1):I) {
      cidx <- floor(100 * (adj[i1, i2] / max_adj))
      lines(xs[c(i1, i2)], ys[c(i1, i2)], lwd=3, col=edge_colors[cidx])
    }
  }

  points(xs, ys, pch=16, cex=1.75, col="white")
  points(xs, ys, pch=16, cex=1.25, col=util$c_dark)
}
```

We can also use an adjacency graph to decompose an initial graph into connected components.

Given an initial node we first find all of the nodes that are at least indirectly related by pairwise comparisons with a recursive, breadth-first search.

```r
breadth_first_search <- function(n, adj, local_nodes=c()) {
  local_nodes <- c(local_nodes, n)
  neighbors <- which(adj[n,] > 0)
  for (nn in neighbors) {
    if (!(nn %in% local_nodes))
      local_nodes <- union(local_nodes,
                           Recall(nn, adj, local_nodes))
  }
  (local_nodes)
}
```

Repeating this search until we've exhausted all of the nodes identifies the connected components.

```r
compute_connected_components <- function(adj) {
  if (!is.matrix(adj)) {
    stop("Input adj is not a matrix.")
  }
  if (nrow(adj) != ncol(adj)) {
    stop("Input adj is not a square matrix.")
  }

  all_nodes <- 1:nrow(adj)
  remaining_nodes <- all_nodes
  components <- list()

  for (n in all_nodes) {
    if (n %in% remaining_nodes) {
      subgraph <- breadth_first_search(n, adj)
      remaining_nodes <- setdiff(remaining_nodes, subgraph)
      components[[length(components) + 1]] <- subgraph
    }
  }

  (components)
}
```

For example let's say that we have a graph defined by 8 items and 8 comparisons.

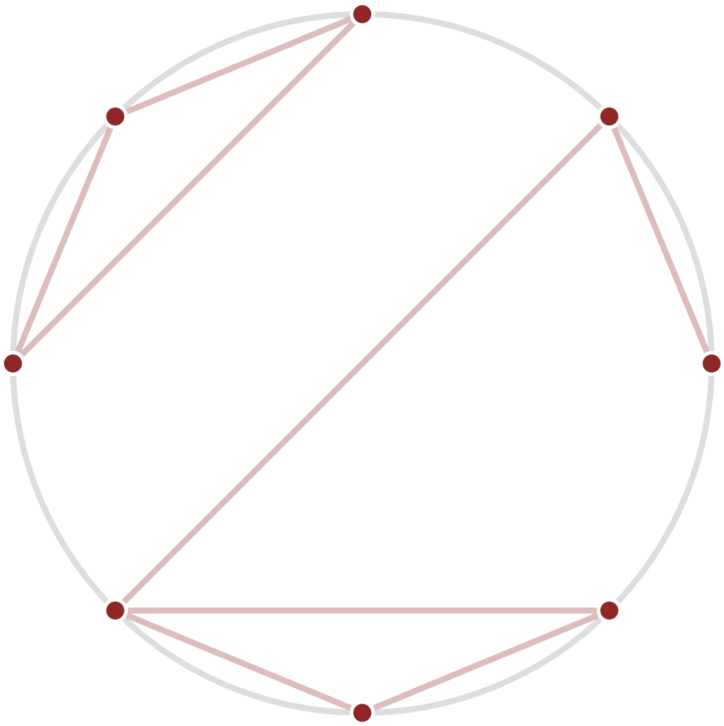
```
K <- 8
C <- 8

item1 <- c(1, 2, 3, 4, 4, 4, 5, 7)
item2 <- c(2, 3, 1, 5, 6, 8, 6, 8)
```

These comparisons define a weighted adjacency matrix.

```
adj <- build_adj_matrix(K, C, item1, item2)
```

The weighted adjacency matrix then informs a visualization of the graph.

```
par(mfrow=c(1, 1), mar=c(0, 0, 0, 0))

plot_undir_graph(adj)
```



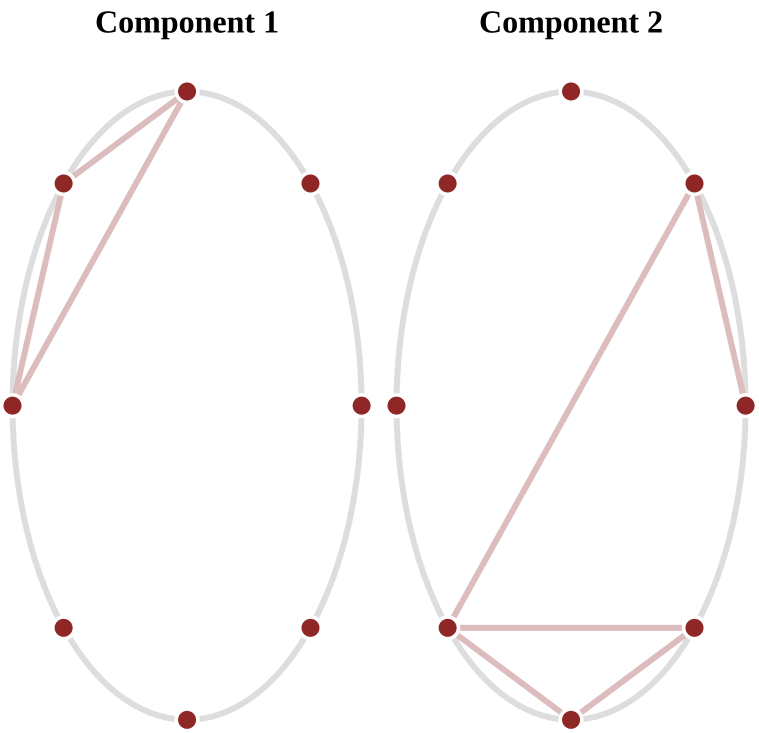
In this case the graph decomposes into two components.

```
components <- compute_connected_components(adj)
components
```

```
[[1]]
[1] 1 2 3

[[2]]
[1] 4 5 6 8 7
```

One way to visualize this decomposition is to visualize each connected component individually.

```
par(mfrow=c(1, 2), mar=c(0, 0, 2, 0))
for (c in seq_along(components)) {
  component_adj <- adj
  for (k in 1:K) {
    if (!(k %in% components[[c]]))
      component_adj[k,] <- rep(0, K)
  }
  plot_undir_graph(component_adj)
  title(paste("Component", c))
}
```



6.3 Modeling Binary Comparisons

With toolbox prepared let's move onto our first example. Here we'll emphasize inferential degeneracies with a model of binary comparisons. To provide some basic context let's say that these binary comparisons are the outcomes of games between two players and we're interesting in inferring the relative skills of each player.

6.3.1 Simulate data

To start we'll need to some data to analyze, including which players are matched up in each game and the eventual winner. For this example we'll simulate the games in two steps, using `player1_probs` to sample the first player and `pair_probs` to conditionally sample their opponent. Note the zeroes in `pair_probs` prevent some player matchups from occurring at all.

```
N_players <- 7
N_games <- 4000

player1_probs <- c(0.2, 0.16, 0.19, 0.11, 0.12, 0.13, 0.09)

pair_probs <- c(0.0, 0.4, 0.0, 0.0, 0.5, 0.0, 0.0,
                0.5, 0.0, 0.0, 0.0, 0.2, 0.6, 0.0,
                0.0, 0.0, 0.0, 0.6, 0.0, 0.0, 0.5,
                0.0, 0.0, 0.7, 0.0, 0.0, 0.0, 0.5,
                0.5, 0.3, 0.0, 0.0, 0.0, 0.4, 0.0,
                0.0, 0.3, 0.0, 0.0, 0.3, 0.0, 0.0,
                0.0, 0.0, 0.3, 0.4, 0.0, 0.0, 0.0)
pair_probs <-  matrix(pair_probs, nrow=N_players)
```

```
simu <- stan(file="stan_programs/simu_bradley_terry.stan",
             algorithm="Fixed_param", seed=8438338,
             data=list("N_players" = N_players,
                       "N_games" = N_games,
                       "player1_probs" = player1_probs,
                       "pair_probs" = pair_probs),
             warmup=0, iter=1, chains=1, refresh=0)
```

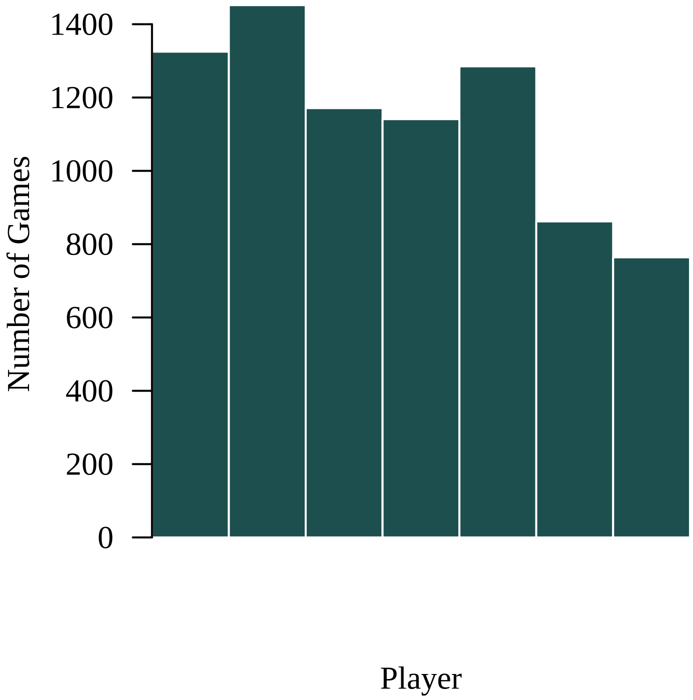
```
data <- list("N_players" = N_players,
             "N_games" = N_games,
             "player1_idx" = extract(simu)$player1_idx[1,],
             "player2_idx" = extract(simu)$player2_idx[1,],
                "y" = extract(simu)$y[1,])

alpha_true <- extract(simu)$alpha[1,]
```

6.3.2 Explore Data

Now we can explore the simulated data.

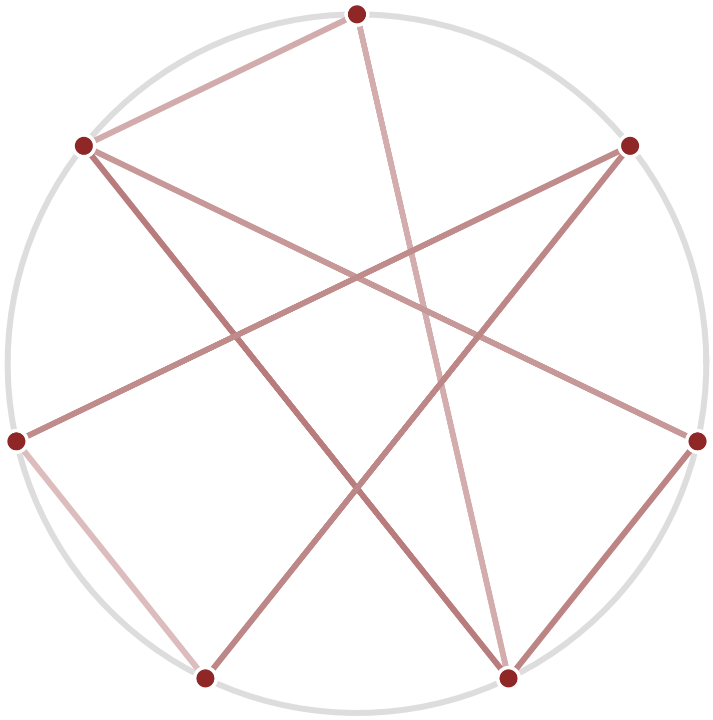
Counting the number of games that include each player we can see that not every player participates in the same number of games. Consequently we should be able to more precisely learn the relative skills of the more active players.

```
game_counts <- sapply(1:data$N_players,
                      function(i) sum(data$player1_idx == i |
                                      data$player2_idx == i   ) )

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

barplot(game_counts,
        space=0, col=util$c_dark_teal, border="white",
        xlab="Player", ylab="Number of Games")
```



To investigate the interaction between the players we can visualize the matchups as a graph.

```
adj <- build_adj_matrix(data$N_players, data$N_games,
                        data$player1_idx, data$player2_idx)

par(mfrow=c(1, 1), mar=c(0, 0, 0, 0))

plot_undir_graph(adj)
```

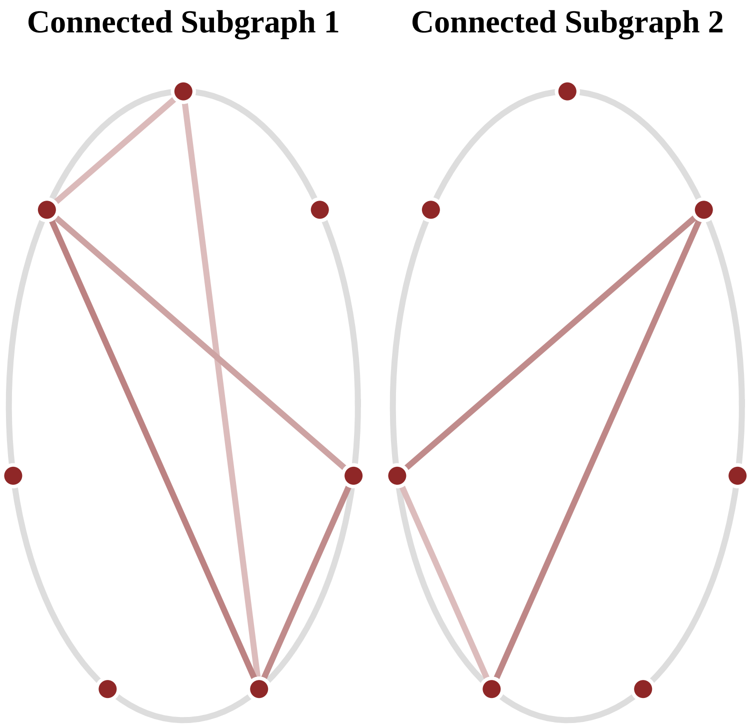


Interestingly the players decompose into two isolated groups, with no games matching up players across the groups.

```
components <- compute_connected_components(adj)
components
```

```
[[1]]
[1] 1 2 5 6

[[2]]
[1] 3 4 7
```

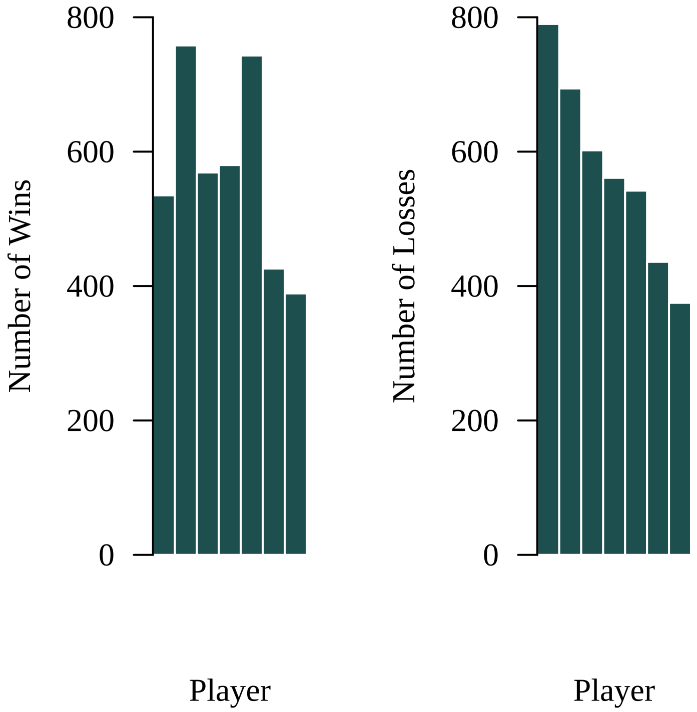
```
par(mfrow=c(1, 2), mar=c(0, 0, 2, 0))

for (c in seq_along(components)) {
  component_adj <- adj
  for (i in 1:data$N_players) {
    if (!(i %in% components[[c]]))
      component_adj[i,] <- rep(0, data$N_players)
  }
  plot_undir_graph(component_adj)
  title(paste("Connected Subgraph", c))
}
```



Given the variation in the number of games played it's no surprise that there is also heterogeneity in the total number of wins and losses for each player.

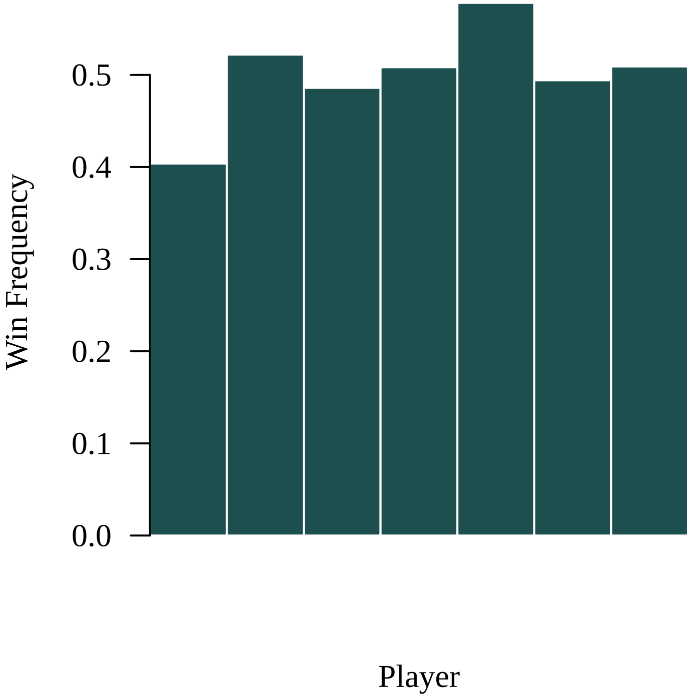
```
win_counts <- sapply(1:data$N_players,
                     function(i) sum(( data$player1_idx == i &
                                       data$y == 1              ) |
                                     ( data$player2_idx == i &
                                       data$y == 0              )  ) )
```

```
loss_counts <- sapply(1:data$N_players,
                      function(i) sum(( data$player1_idx == i &
                                        data$y == 0              ) |
                                      ( data$player2_idx == i &
                                        data$y == 1              )  ) )

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

barplot(win_counts,
        space=0, col=util$c_dark_teal, border="white",
        xlab="Player", ylab="Number of Wins", ylim=c(0, 800))

barplot(loss_counts,
        space=0, col=util$c_dark_teal, border="white",
        xlab="Player", ylab="Number of Losses", ylim=c(0, 800))
```



The proportion of games won should be particularly sensitive to the individual player skills.

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

barplot(win_counts / game_counts,
        space=0, col=util$c_dark_teal, border="white",
        xlab="Player", ylab="Win Frequency")
```



Indeed these empirical win frequencies make for a useful summary statistic.

6.3.3 Attempt 1

Given the binary outcomes we'll follow the Bradley-Terry construction, with a Bernoulli model,

$$\text{Bernoulli}(y_{i_1 i_2} \mid p_{i_1 i_2}),$$

and logistic relationship between the probability that player i_1 defeats player i_2 and the corresponding difference in skills,

$$p_{i_1 i_2} = \text{logistic}(\alpha_{i_1} - \alpha_{i_2}).$$

To investigate any inferential degeneracies as directly as possible let's start with an improper, flat prior density function so that our Markov chains are effectively exploring the contours of the realized likelihood function.

```
fit <- stan(file="stan_programs/bradley_terry1.stan",
            data=data, seed=8438338,
            warmup=1000, iter=2024, refresh=0)
```

Unfortunately the diagnostics are very unhappy.

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

```
  Chain 1: 485 of 1024 transitions (47.36328125%) saturated the maximum treedepth of 10.

  Chain 2: 439 of 1024 transitions (42.87109375%) saturated the maximum treedepth of 10.

  Chain 3: 562 of 1024 transitions (54.8828125%) saturated the maximum treedepth of 10.

  Chain 4: 497 of 1024 transitions (48.53515625%) saturated the maximum treedepth of 10.

  Numerical trajectories that saturate the maximum treedepth have
terminated prematurely.  Increasing max_depth above 10 should result in
more expensive, but more efficient, Hamiltonian transitions.
```

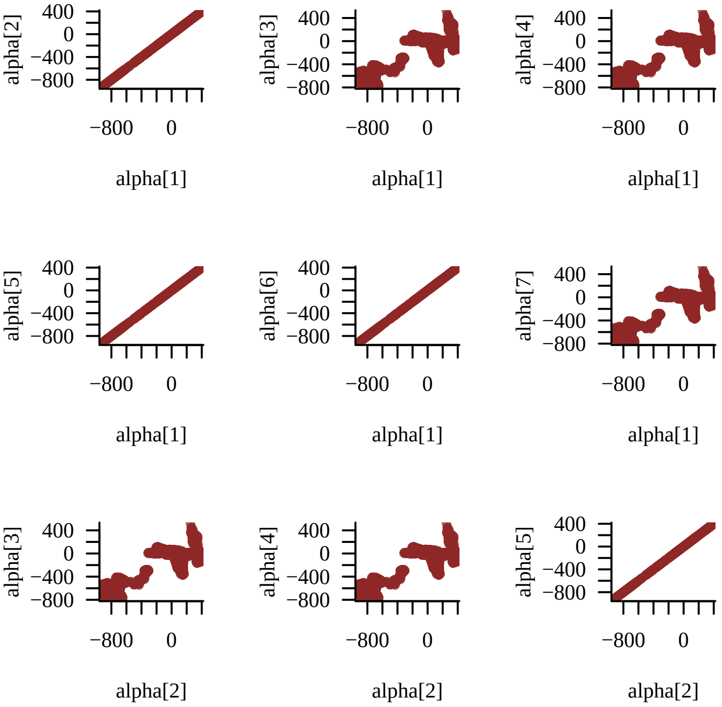
```
samples <- util$extract_expectand_vals(fit)
base_samples <- util$filter_expectands(samples,
                                       c('alpha'),
                                       check_arrays=TRUE)
util$check_all_expectand_diagnostics(base_samples)
```

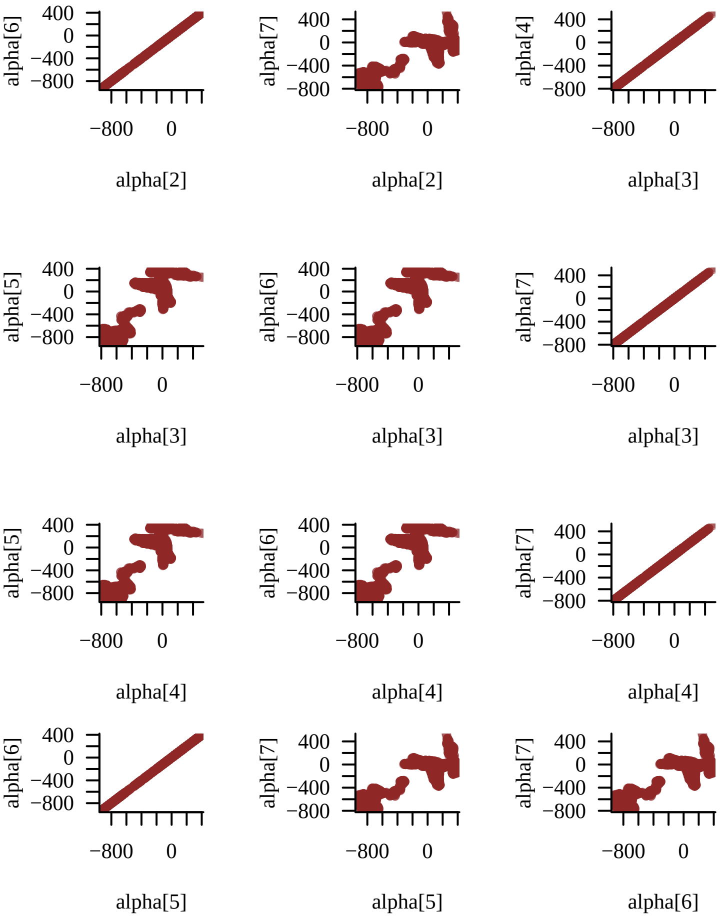
```
alpha[1]:
  Split hat{R} (4.600) exceeds 1.1.
  Chain 1: hat{ESS} (8.697) is smaller than desired (100).
  Chain 2: hat{ESS} (5.632) is smaller than desired (100).
  Chain 3: hat{ESS} (8.079) is smaller than desired (100).
  Chain 4: hat{ESS} (4.521) is smaller than desired (100).

alpha[2]:
  Split hat{R} (4.600) exceeds 1.1.
  Chain 1: hat{ESS} (8.697) is smaller than desired (100).
  Chain 2: hat{ESS} (5.632) is smaller than desired (100).
  Chain 3: hat{ESS} (8.079) is smaller than desired (100).
  Chain 4: hat{ESS} (4.521) is smaller than desired (100).
```

```
alpha[3]:
  Split hat{R} (3.996) exceeds 1.1.
  Chain 1: hat{ESS} (5.616) is smaller than desired (100).
  Chain 2: hat{ESS} (6.316) is smaller than desired (100).
  Chain 3: hat{ESS} (10.696) is smaller than desired (100).
  Chain 4: hat{ESS} (13.759) is smaller than desired (100).

alpha[4]:
  Split hat{R} (3.996) exceeds 1.1.
  Chain 1: hat{ESS} (5.616) is smaller than desired (100).
  Chain 2: hat{ESS} (6.316) is smaller than desired (100).
  Chain 3: hat{ESS} (10.696) is smaller than desired (100).
  Chain 4: hat{ESS} (13.759) is smaller than desired (100).

alpha[5]:
  Split hat{R} (4.600) exceeds 1.1.
  Chain 1: hat{ESS} (8.697) is smaller than desired (100).
  Chain 2: hat{ESS} (5.632) is smaller than desired (100).
  Chain 3: hat{ESS} (8.078) is smaller than desired (100).
  Chain 4: hat{ESS} (4.521) is smaller than desired (100).

alpha[6]:
  Split hat{R} (4.600) exceeds 1.1.
  Chain 1: hat{ESS} (8.694) is smaller than desired (100).
  Chain 2: hat{ESS} (5.632) is smaller than desired (100).
  Chain 3: hat{ESS} (8.078) is smaller than desired (100).
  Chain 4: hat{ESS} (4.520) is smaller than desired (100).

alpha[7]:
  Split hat{R} (3.996) exceeds 1.1.
  Chain 1: hat{ESS} (5.616) is smaller than desired (100).
  Chain 2: hat{ESS} (6.316) is smaller than desired (100).
  Chain 3: hat{ESS} (10.695) is smaller than desired (100).
  Chain 4: hat{ESS} (13.758) is smaller than desired (100).


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

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

Together the treedepth saturation and low $E\hat{S}S$s suggest that the likelihood function is concentrating into a long, narrow geometry. Indeed when we look through pairs plots of the player skills we see some of them exhibiting translation invariance.

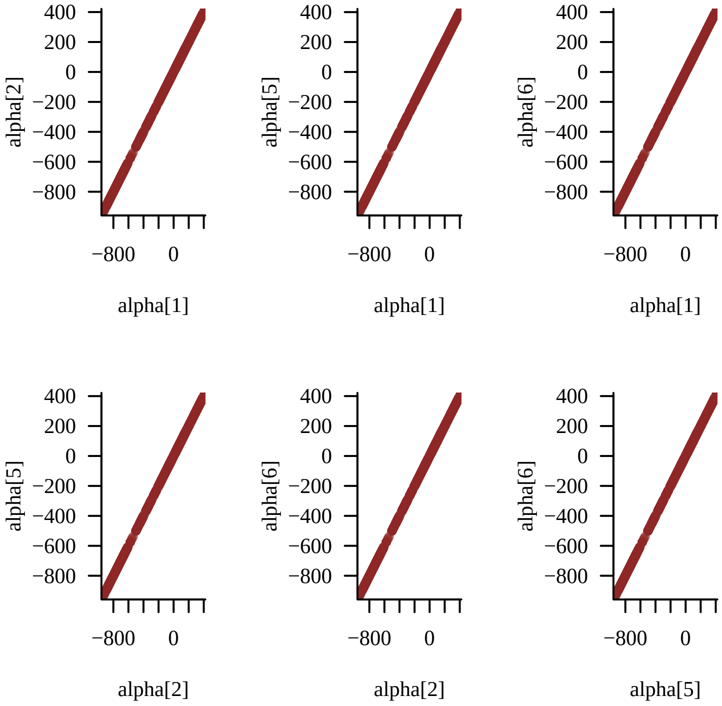
```
names <- sapply(1:data$N_players, function(i) paste0('alpha[', i, ']'))
util$plot_div_pairs(names, names, samples, diagnostics)
```



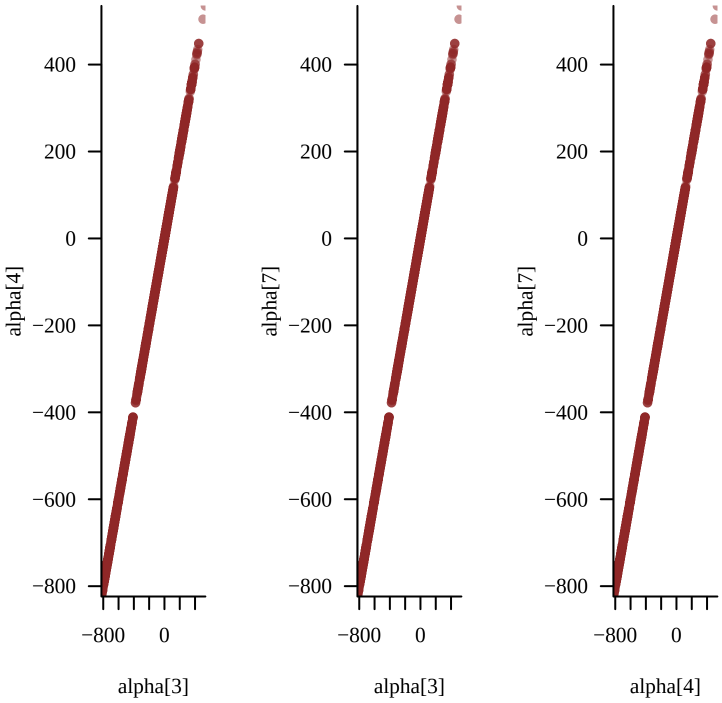


The behaviors become more consistent if we compare the skills of players only within the same connected components. All of these pairs plots show the same degenerate geometry.

```
names <- sapply(components[[1]], function(i) paste0('alpha[', i, ']'))
util$plot_div_pairs(names, names, samples, diagnostics)
```

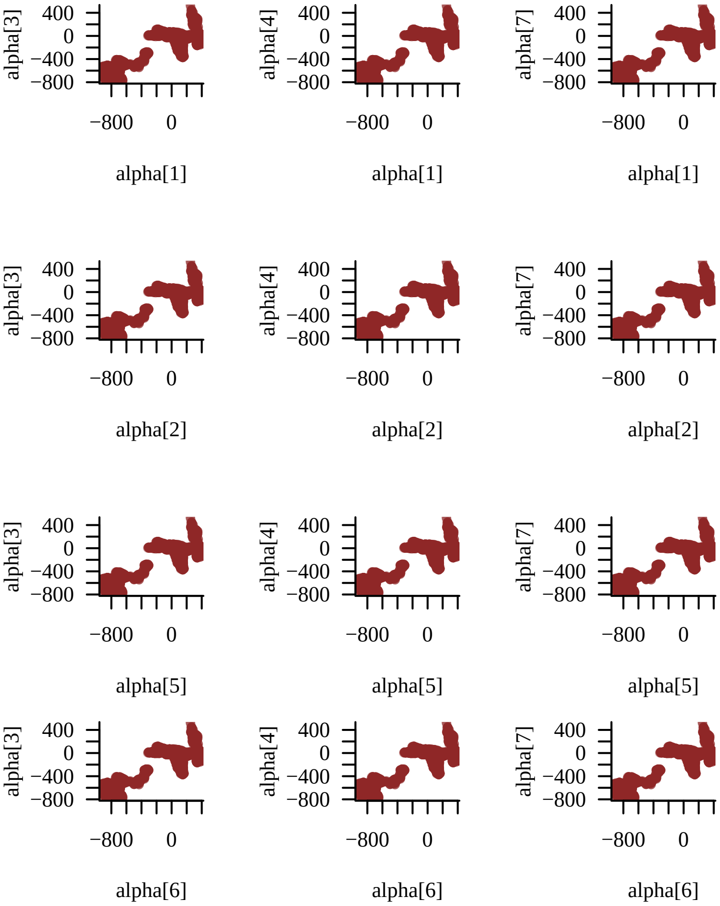


```
names <- sapply(components[[2]], function(i) paste0('alpha[', i, ']'))
util$plot_div_pairs(names, names, samples, diagnostics)
```



Interestingly the skills of players between the two connected subgroups appear to be largely uncorrelated. Instead the players skills incoherently meandering around the consistent values.

```r
names1 <- sapply(components[[1]], function(i) paste0('alpha[', i, ']'))
names2 <- sapply(components[[2]], function(i) paste0('alpha[', i, ']'))
util$plot_div_pairs(names1, names2, samples, diagnostics)
```

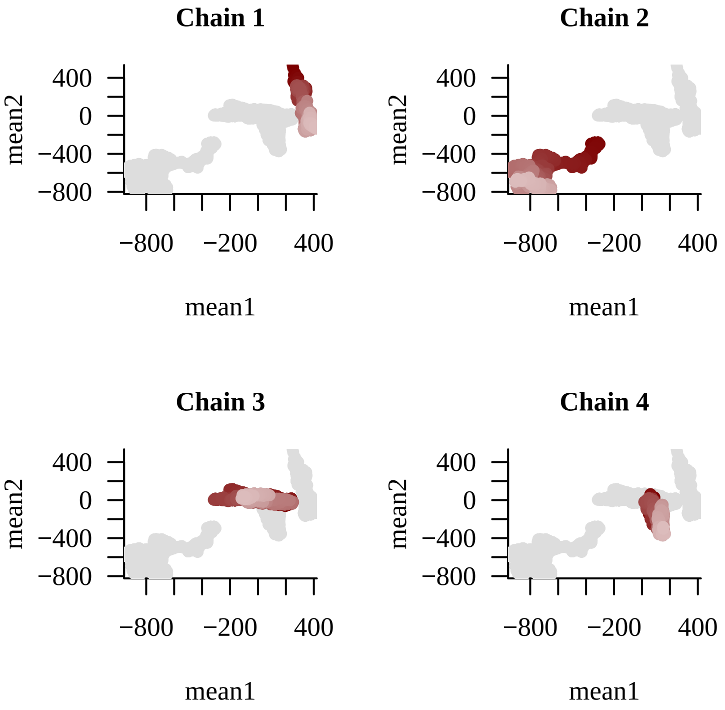


The fact that *all* of the cross-component pairs plots exhibit identical meandering patterns is somewhat disconcerting. Something has to be coupling all of these behaviors together.

Recall that the observed game outcomes inform each player's skill relative to their historical opponents. Nothing, however, informs how the two connected components should behave relative to each other. A player who has won all of their matches but played only poor opponents isn't likely to fare well against new competition. Similarly a player who has lost all of their matches against elite opponents can perform surprisingly well against weaker competition.

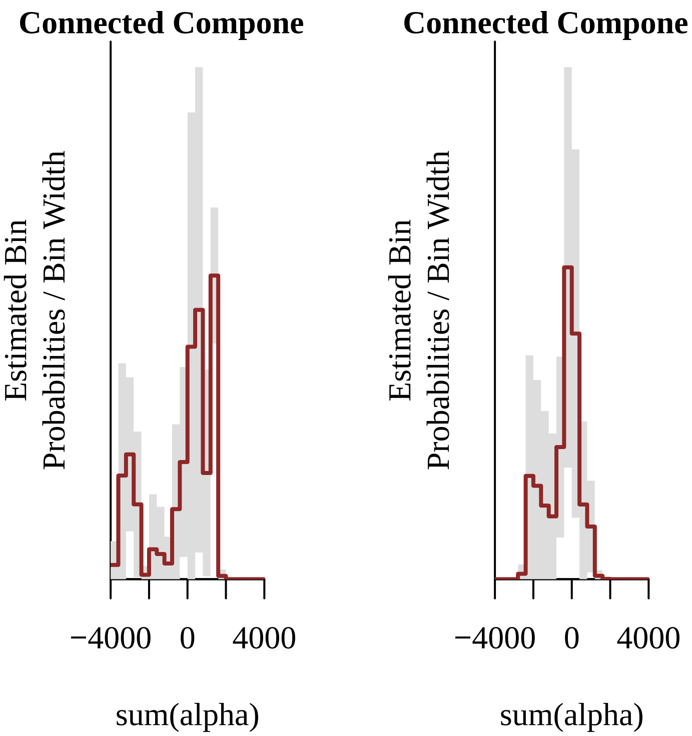
Consequently the average skills of the two connected components effectively random walk around each other.

```
var_repl <- list('alpha'=array(sapply(components[[1]],
                                      function(i)
                                      paste0("alpha[", i, "]"))))
mean_vals1 <- util$eval_expectand_pushforward(samples,
                                              function(alpha)
                                                mean(alpha),
                                              var_repl)

var_repl <- list('alpha'=array(sapply(components[[2]],
                                      function(i)
                                      paste0("alpha[", i, "]"))))
mean_vals2 <- util$eval_expectand_pushforward(samples,
                                              function(alpha)
                                                mean(alpha),
                                              var_repl)

util$plot_pairs_by_chain(mean_vals1, 'mean1',
                         mean_vals2, 'mean2')
```



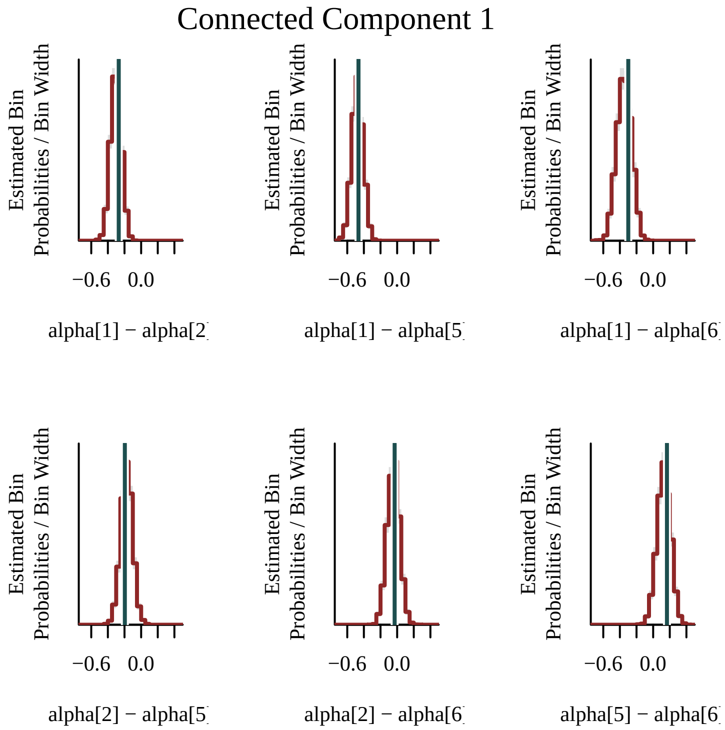
Another way to understand these inferential degeneracies is that the observed game outcomes inform the differences in the player skills but not the sums. For this initial model the explored skill sums are limited only by the finite length of the Markov chains.

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

var_repl <- list('alpha'=array(sapply(components[[1]],
                                      function(i)
                                      paste0("alpha[", i, "]"))))
sum_vals <- util$eval_expectand_pushforward(samples,
                                            function(alpha) sum(alpha),
                                            var_repl)
util$plot_expectand_pushforward(sum_vals, 20, flim=c(-4000, 4000),
                                display_name="sum(alpha)",
                                main="Connected Component 1")

var_repl <- list('alpha'=array(sapply(components[[2]],
                                      function(i)
                                      paste0("alpha[", i, "]"))))
sum_vals <- util$eval_expectand_pushforward(samples,
                                            function(alpha) sum(alpha),
                                            var_repl)
util$plot_expectand_pushforward(sum_vals, 20, flim=c(-4000, 4000),
                                display_name="sum(alpha)",
                                main="Connected Component 2")
```

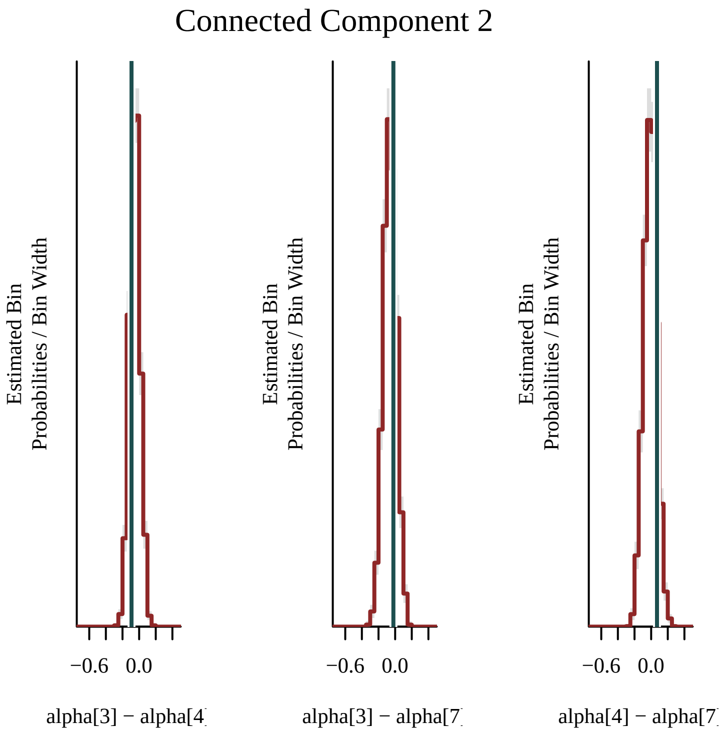


On the other hand the skill differences within each connected component are consistent with the behavior of the simulation data generating process.

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

N <- length(components[[1]])
for (i1 in 1:(N - 1)) {
  for (i2 in (i1 + 1):N) {
    idx1 <- components[[1]][i1]
    idx2 <- components[[1]][i2]
    name1 <- paste0('alpha[', idx1, ']')
    name2 <- paste0('alpha[', idx2, ']')
    diff_vals <- util$eval_expectand_pushforward(samples,
                                                 function(a1, a2) a1 - a2,
                                                 list('a1'=name1,
                                                      'a2'=name2))
    util$plot_expectand_pushforward(diff_vals,
                                    25, flim=c(-0.75, 0.5),
                                    baseline=alpha_true[idx1] -
                                             alpha_true[idx2],
                                    baseline_col=util$c_dark_teal,
                                    display_name=paste(name1, '-',
                                                     name2))
  }
}

mtext("Connected Component 1", line=-2, outer=TRUE)
```



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

N <- length(components[[2]])
for (i1 in 1:(N - 1)) {
  for (i2 in (i1 + 1):N) {
    idx1 <- components[[2]][i1]
    idx2 <- components[[2]][i2]
    name1 <- paste0('alpha[', idx1, ']')
    name2 <- paste0('alpha[', idx2, ']')
    diff_vals <- util$eval_expectand_pushforward(samples,
                                               function(a1, a2)
                                               a1 - a2,
                                               list('a1'=name1,
                                                    'a2'=name2))
    util$plot_expectand_pushforward(diff_vals,
                                            25, flim=c(-0.75, 0.5),
                                            baseline=alpha_true[idx1] -
                                                     alpha_true[idx2],
                                            baseline_col=util$c_dark_teal,
                                            display_name=paste(name1, '-',
                                                               name2))
  }
}

mtext("Connected Component 2", line=-2, outer=TRUE)
```



6.3.4 Attempt 2

We should be able to tame the degenerate geometry of the realized likelihood function with a proper prior model. Let's start by considering a diffuse, but well-defined, prior model that contains the magnitude of the player skills below 10. Keep in mind that inputs of ±10 to a logistic function correspond to pretty extreme output probabilities!

```
fit <- stan(file="stan_programs/bradley_terry2.stan",
            data=data, seed=8438338,
            warmup=1000, iter=2024, refresh=0)
```

Even with this conservative prior model the diagnostics are much better behaved than they were before.

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

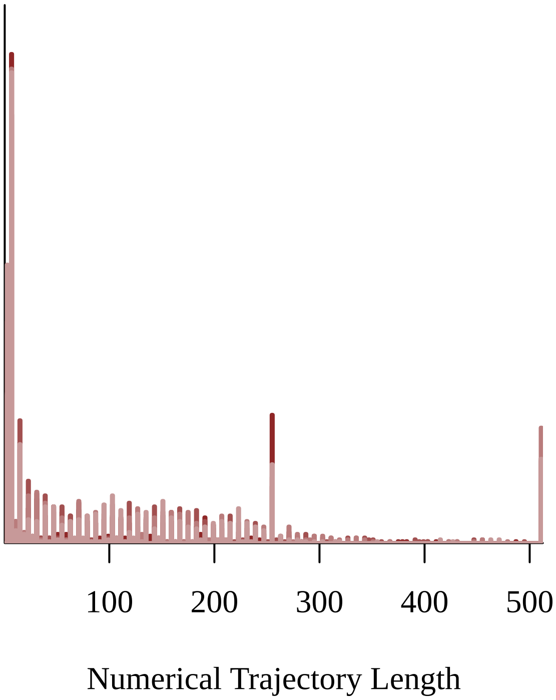
```
  All Hamiltonian Monte Carlo diagnostics are consistent with reliable
Markov chain Monte Carlo.
```

```
samples <- util$extract_expectand_vals(fit)
base_samples <- util$filter_expectands(samples,
                                       c('alpha'),
                                       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.
```

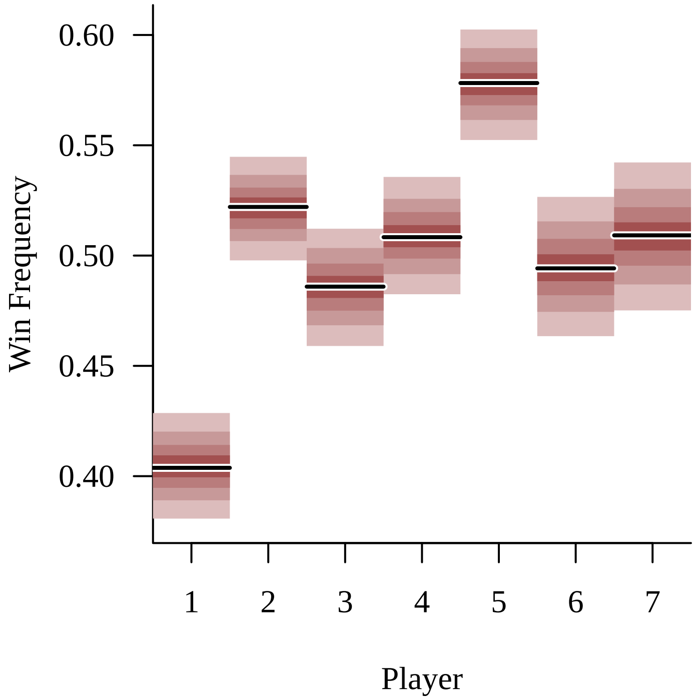
That said the posterior exploration requires long, and consequently expensive, Hamiltonian trajectories.

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

util$plot_num_leapfrogs(diagnostics)
```



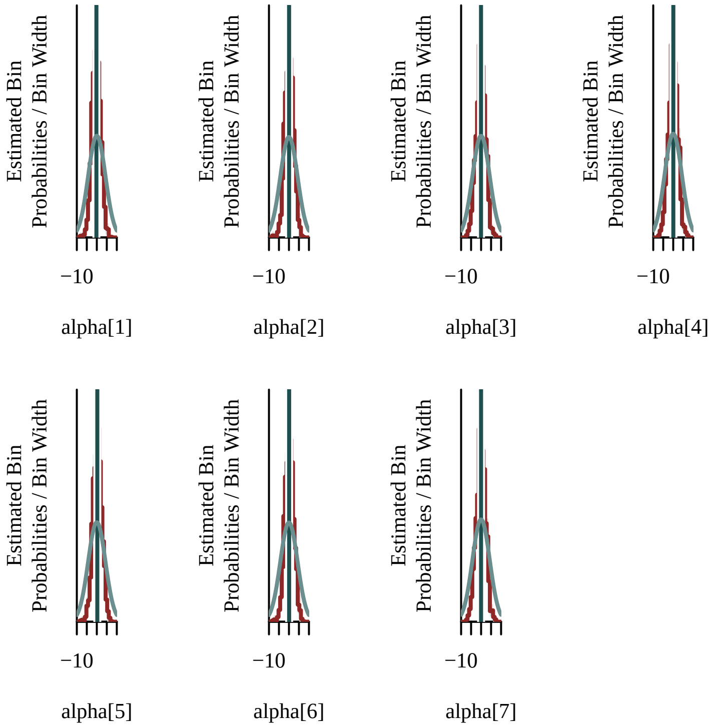
The costly exploration, however, does yield good retrodictive performance. In particular the behavior of the posterior predictive and observed empirical win frequencies are consistent with each other.

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

names <- sapply(1:data$N_players,
                function(i) paste0('win_freq_pred[', i, ']'))
util$plot_disc_pushforward_quantiles(samples, names,
                                     baseline_values=win_counts /
                                                     game_counts,
                                     xlab="Player",
                                     ylab="Win Frequency")
```



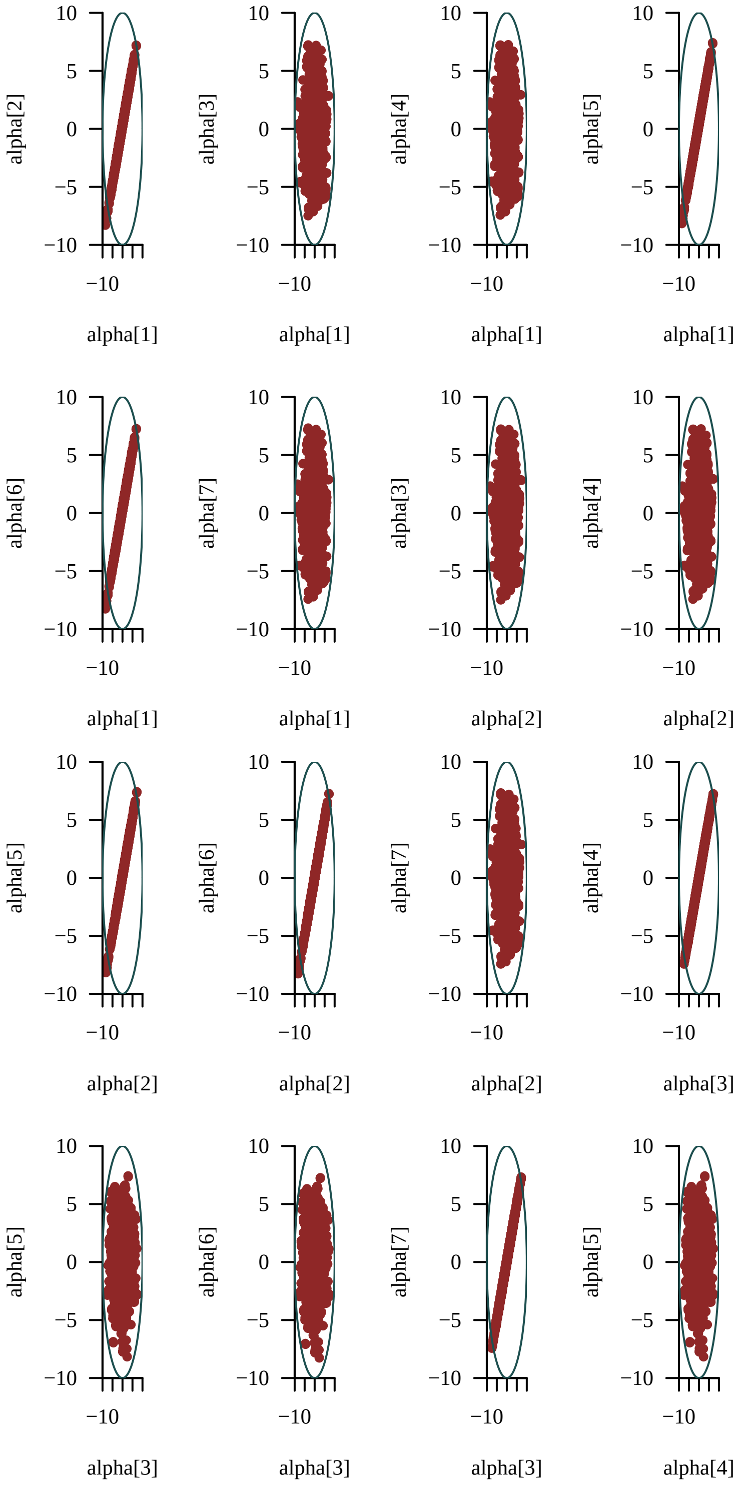
Marginally the posterior inferences for each player skill aren't that much more informative than the prior model.

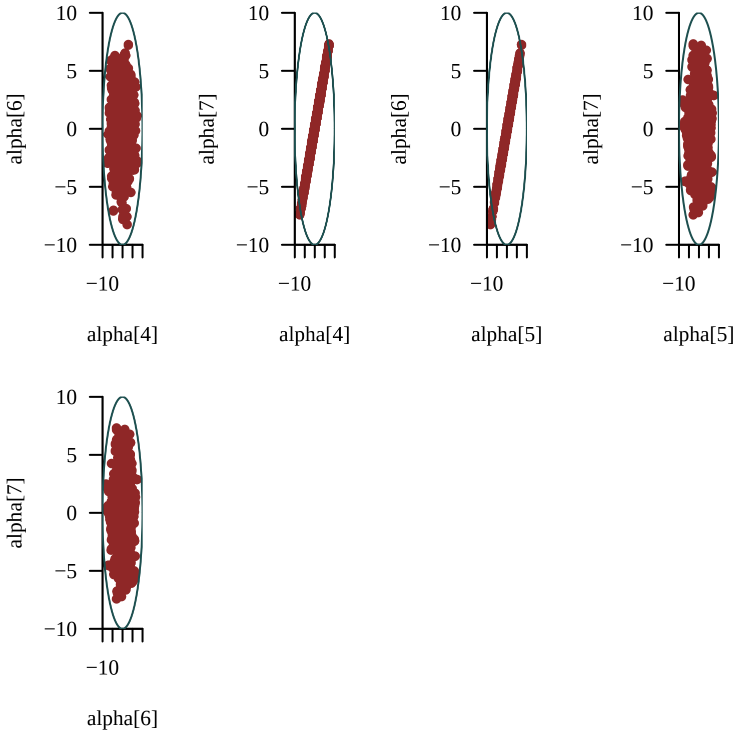
```
par(mfrow=c(2, 4), mar=c(5, 5, 1, 1))

for (i in 1:data$N_players) {
  name <- paste0('alpha[', i, ']')
  util$plot_expectand_pushforward(samples[[name]],
                                  25, flim=c(-10, 10),
                                  baseline=alpha_true[i],
                                  baseline_col=util$c_dark_teal,
                                  display_name=name)
  xs <- seq(-10, 10, 0.25)
  ys <- dnorm(xs, 0, 10 / 2.32)
  lines(xs, ys, lwd=2, col=util$c_light_teal)
}
```



It’s not that we’re not learning anything; the game outcomes just don’t pin down the absolute value of the player skills.

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

for (i in 1:(data$N_players - 1)) {
  for (j in (i + 1):data$N_players) {
    name1 <- paste0('alpha[', i, ']')
    name2 <- paste0('alpha[', j, ']')
    plot(c(samples[[name1]]), c(samples[[name2]]),
         cex=1, pch=16, col=util$c_dark,
         xlab=name1, xlim=c(-10, 10),
         ylab=name2, ylim=c(-10, 10))
    lines(10 * cos(seq(0, 2 * pi, 2 * pi / 250)),
          10 * sin(seq(0, 2 * pi, 2 * pi / 250)),
          col=util$c_dark_teal)
  }
}
```

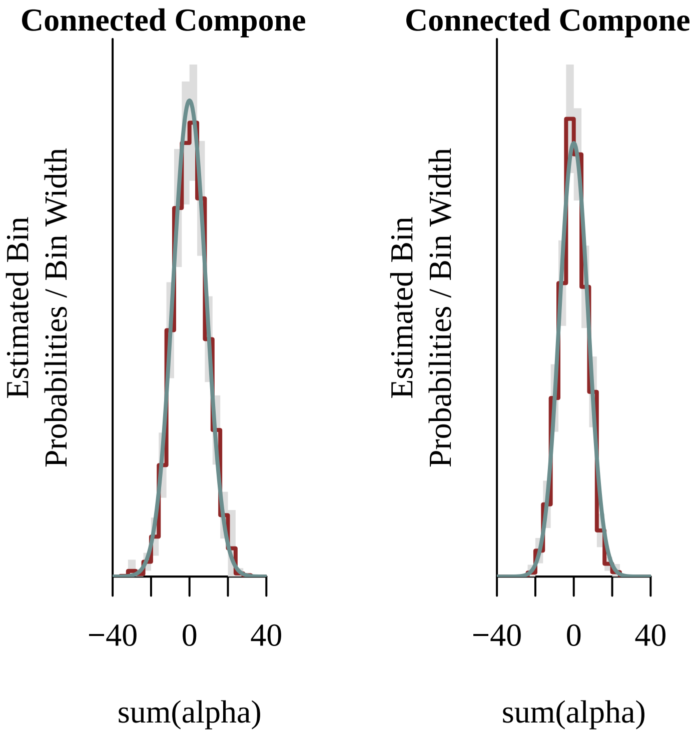

10
5
0
−5
−10
−10
alpha[2]
alpha[1]
alpha[3]
alpha[1]
alpha[4]
alpha[1]
alpha[5]
alpha[1]
alpha[6]
alpha[1]
alpha[7]
alpha[1]
alpha[3]
alpha[2]
alpha[4]
alpha[2]
alpha[5]
alpha[2]
alpha[6]
alpha[2]
alpha[7]
alpha[2]
alpha[4]
alpha[3]
alpha[5]
alpha[3]
alpha[6]
alpha[3]
alpha[7]
alpha[3]
alpha[5]
alpha[4]



In particular the sum of the player skills within each connected component are informed entirely by the prior model.

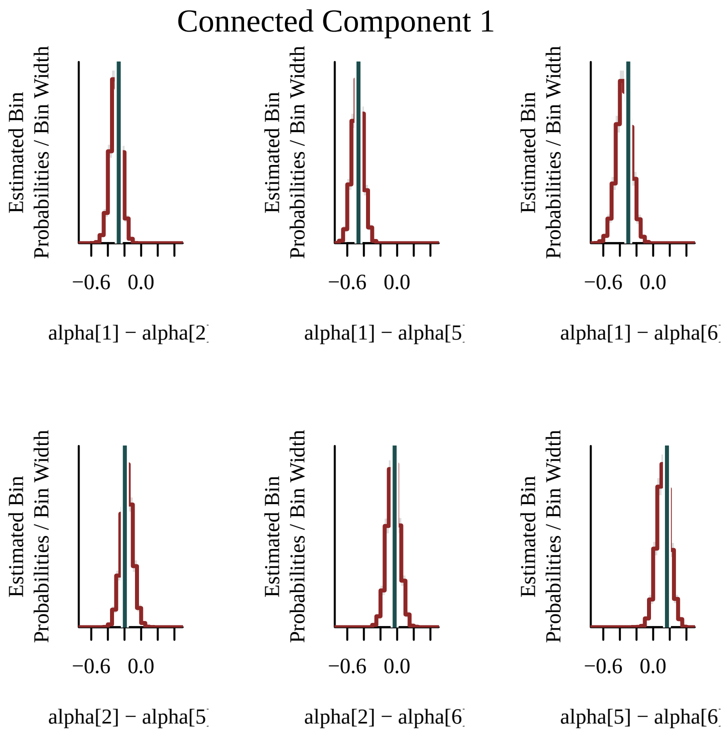
```
par(mfrow=c(1, 2), mar=c(5, 5, 1, 1))

var_repl <- list('alpha'=array(sapply(components[[1]],
                                      function(i)
                                      paste0("alpha[", i, "]"))))
sum_vals <- util$eval_expectand_pushforward(samples,
                                            function(alpha) sum(alpha),
                                            var_repl)
util$plot_expectand_pushforward(sum_vals, 20, flim=c(-40, 40),
                                display_name="sum(alpha)",
                                main="Connected Component 1")

xs <- seq(-40, 40, 0.5)
ys <- dnorm(xs, 0, sqrt(length(components[[1]])) * 10 / 2.32)
lines(xs, ys, lwd=2, col=util$c_light_teal)

var_repl <- list('alpha'=array(sapply(components[[2]],
                                      function(i)
                                      paste0("alpha[", i, "]"))))
```

```
sum_vals <- util$eval_expectand_pushforward(samples,
                                            function(alpha) sum(alpha),
                                            var_repl)
util$plot_expectand_pushforward(sum_vals, 20, flim=c(-40, 40),
                                display_name="sum(alpha)",
                                main="Connected Component 2")

xs <- seq(-40, 40, 0.5)
ys <- dnorm(xs, 0, sqrt(length(components[[2]])) * 10 / 2.32)
lines(xs, ys, lwd=2, col=util$c_light_teal)
```

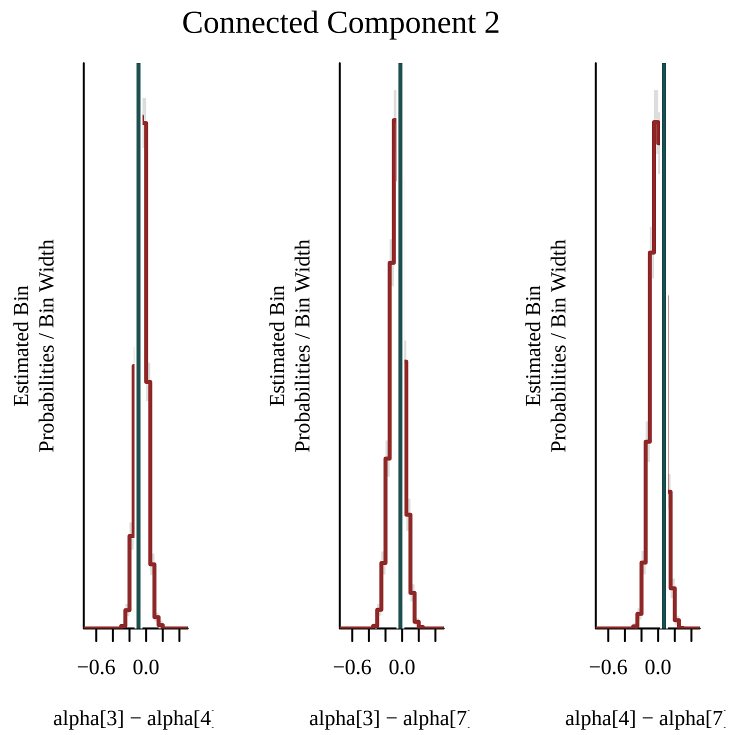


The observations instead inform only relative performance, and hence the differences in the player skills.

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

N <- length(components[[1]])
for (i1 in 1:(N - 1)) {
  for (i2 in (i1 + 1):N) {
    idx1 <- components[[1]][i1]
    idx2 <- components[[1]][i2]
    name1 <- paste0('alpha[', idx1, ']')
    name2 <- paste0('alpha[', idx2, ']')
    diff_vals <- util$eval_expectand_pushforward(samples,
                                                 function(a1, a2)
                                                 a1 - a2,
                                                 list('a1'=name1,
                                                      'a2'=name2))
    util$plot_expectand_pushforward(diff_vals,
                                    25, flim=c(-0.75, 0.5),
                                    baseline=alpha_true[idx1] -
                                             alpha_true[idx2],
                                    baseline_col=util$c_dark_teal,
                                    display_name=paste(name1, '-',
                                                       name2))
  }
}

mtext("Connected Component 1", line=-2, outer=TRUE)
```



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

N <- length(components[[2]])
for (i1 in 1:(N - 1)) {
  for (i2 in (i1 + 1):N) {
    idx1 <- components[[2]][i1]
    idx2 <- components[[2]][i2]
    name1 <- paste0('alpha[', idx1, ']')
    name2 <- paste0('alpha[', idx2, ']')
    diff_vals <- util$eval_expectand_pushforward(samples,
                                                 function(a1, a2)
                                                 a1 - a2,
                                                 list('a1'=name1,
                                                      'a2'=name2))
    util$plot_expectand_pushforward(diff_vals,
                                    25, flim=c(-0.75, 0.5),
                                    baseline=alpha_true[idx1] -
                                             alpha_true[idx2],
                                    baseline_col=util$c_dark_teal,
                                    display_name=paste(name1, '-',
                                                       name2))
  }
}

mtext("Connected Component 2", line=-2, outer=TRUE)
```



6.3.5 Attempt 3

Although the previous model was by no means a disaster we should be able to achieve faster computation with a more informative prior model. We don't even have to be all that precise with our domain expertise elicitation.

For example let's say that our domain expertise is inconsistent with win probabilities below 1/3 and above 2/3,

$$\frac{1}{3} \lessapprox p_{i_1 i_2} \lessapprox \frac{2}{3}.$$

In the context of our model this implies that

$$\begin{aligned}
\text{logit}\left(\frac{1}{3}\right) &\lessapprox \text{logit}(p_{i_1 i_2}) &&\lessapprox \text{logit}\left(\frac{2}{3}\right) \\
\log\left(\frac{\frac{1}{3}}{1-\frac{1}{3}}\right) &\lessapprox \alpha_{i1} - \alpha_{i2} &&\lessapprox \log\left(\frac{\frac{2}{3}}{1-\frac{2}{3}}\right) \\
\log\left(\frac{\frac{1}{3}}{\frac{2}{3}}\right) &\lessapprox \alpha_{i1} - \alpha_{i2} &&\lessapprox \log\left(\frac{\frac{2}{3}}{\frac{1}{3}}\right) \\
\log\left(\frac{1}{2}\right) &\lessapprox \alpha_{i1} - \alpha_{i2} &&\lessapprox \log\left(2\right) \\
-\log 2 &\lessapprox \alpha_{i1} - \alpha_{i2} &&\lessapprox +\log 2.
\end{aligned}$$

We can achieve this containment in the differences by containing the individual player skills to

$$
\begin{aligned}
- \log 2 \; &\lessapprox \; 2 \, \alpha_i \; \lessapprox \quad + \log 2 \\
- \frac{1}{2} \log 2 \; &\lessapprox \quad \alpha_i \; \lessapprox + \frac{1}{2} \log 2.
\end{aligned}
$$

For context the threshold $\frac{1}{2} \log 2$ is about 0.35, almost two orders of magnitude smaller than the more conservative threshold that we used in the previous model!

```
fit <- stan(file="stan_programs/bradley_terry3.stan",
            data=data, seed=8438338,
            warmup=1000, iter=2024, refresh=0)
```

A stronger prior containment should only make the posterior geometry more well-behaved, and unsurprisingly the computational diagnostics remain clean.

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

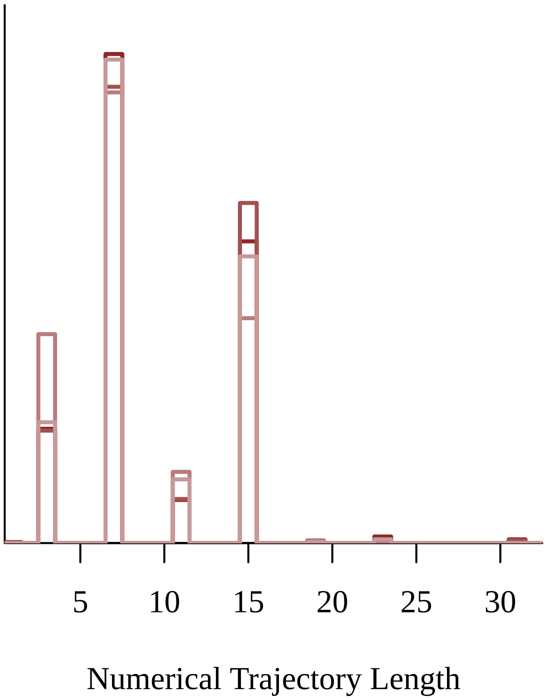
```
  All Hamiltonian Monte Carlo diagnostics are consistent with reliable
Markov chain Monte Carlo.
```

```
samples <- util$extract_expectand_vals(fit)
base_samples <- util$filter_expectands(samples,
                                       c('alpha'),
                                       check_arrays=TRUE)
util$summarize_expectand_diagnostics(base_samples)
```

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

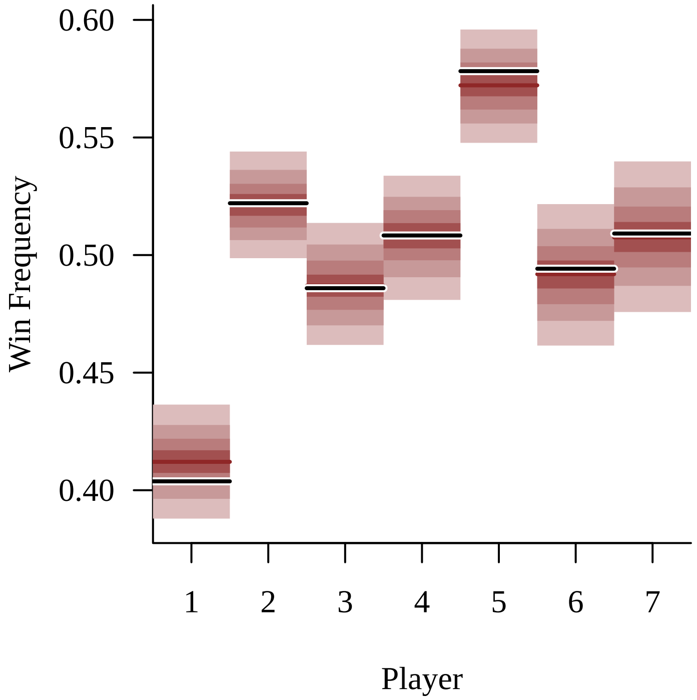
Moreover the improved geometry requires almost an order of magnitude fewer model evaluations to inform each Markov transition.

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

util$plot_num_leapfrogs(diagnostics)
```



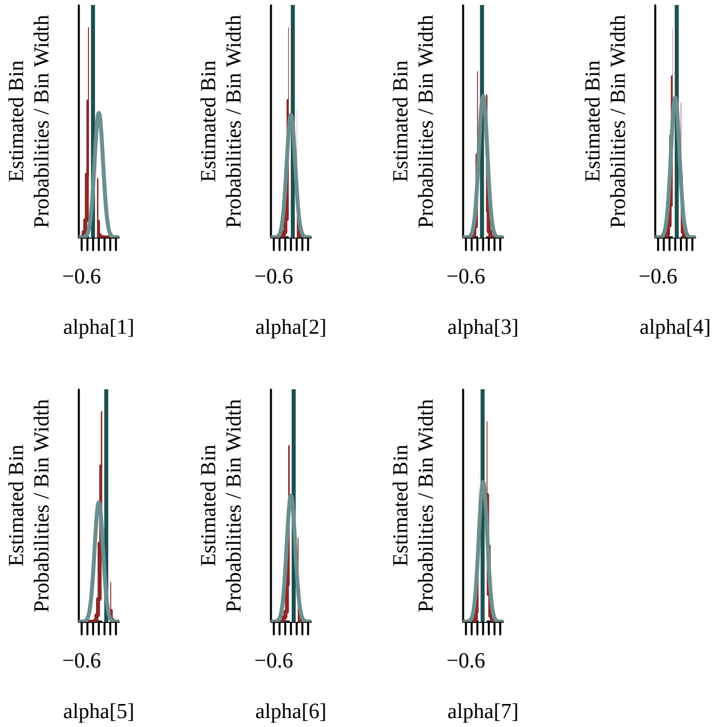
Similarly no problems have snuck into the posterior retrodictive checks.

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

names <- sapply(1:data$N_players,
                function(i) paste0('win_freq_pred[', i, ']'))
util$plot_disc_pushforward_quantiles(samples, names,
                                     baseline_values=win_counts /
                                                     game_counts,
                                     xlab="Player",
                                     ylab="Win Frequency")
```

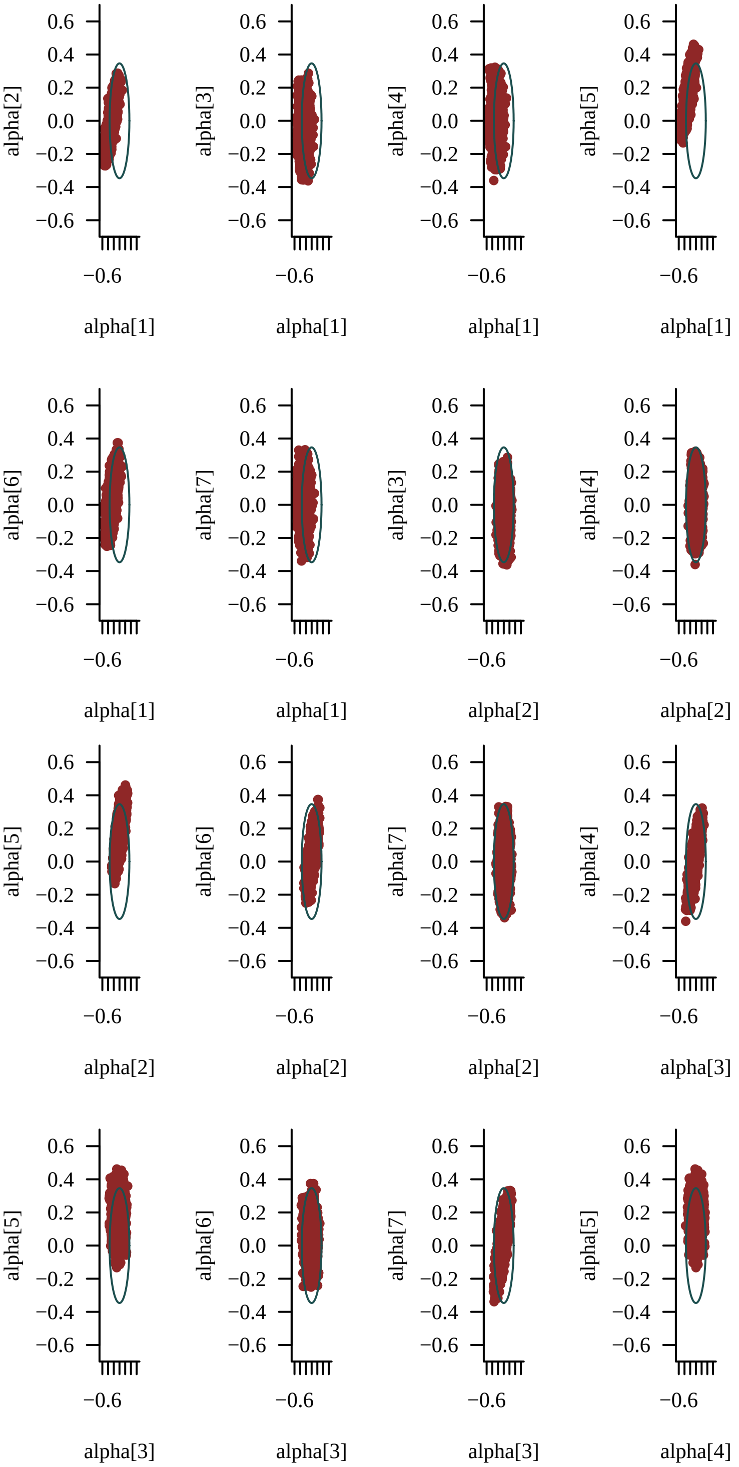


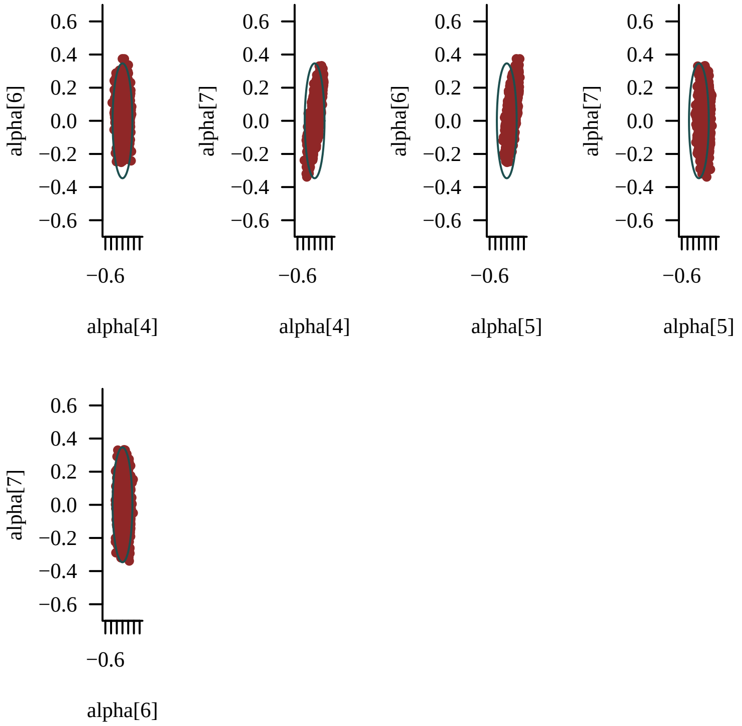
While the individual player skills are still poorly informed by the observed data, the stronger prior model corrals them into a smaller neighborhood.

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

for (i in 1:data$N_players) {
  name <- paste0('alpha[', i, ']')
  util$plot_expectand_pushforward(samples[[name]],
                                  25, flim=c(-0.7, 0.7),
                                  baseline=alpha_true[i],
                                  baseline_col=util$c_dark_teal,
                                  display_name=name)
  xs <- seq(-1.25, 1.25, 0.01)
  ys <- dnorm(xs, 0, log(sqrt(2)) / 2.32)
  lines(xs, ys, lwd=2, col=util$c_light_teal)
}
```



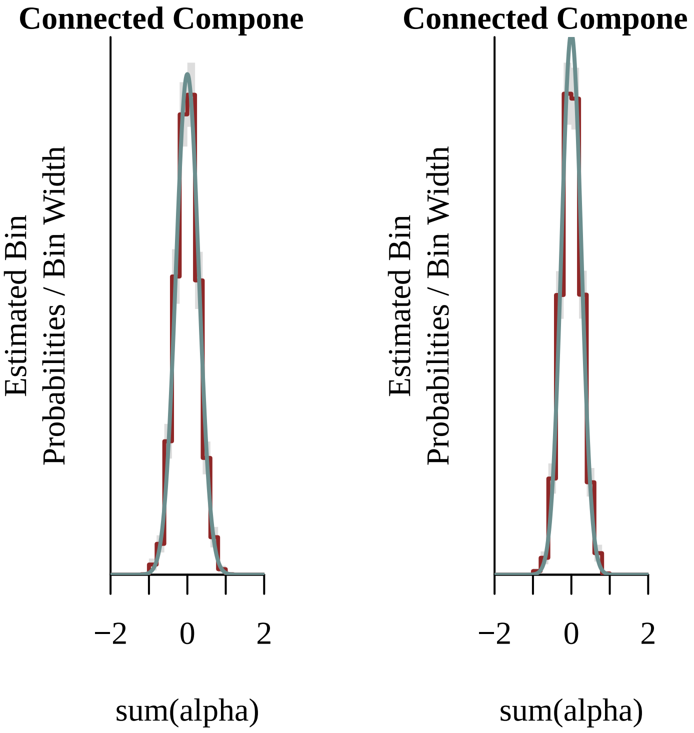
```r
par(mfrow=c(2, 4), mar=c(5, 5, 1, 1))

for (i in 1:(data$N_players - 1)) {
  for (j in (i + 1):data$N_players) {
    name1 <- paste0('alpha[', i, ']')
    name2 <- paste0('alpha[', j, ']')
    plot(c(samples[[name1]]), c(samples[[name2]]),
         cex=1, pch=16, col=util$c_dark,
         xlab=name1, xlim=c(-0.7, 0.7),
         ylab=name2, ylim=c(-0.7, 0.7))
    lines(log(sqrt(2)) * cos(seq(0, 2 * pi, 2 * pi / 250)),
          log(sqrt(2)) * sin(seq(0, 2 * pi, 2 * pi / 250)),
          col=util$c_dark_teal)
  }
}
```


0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.6
alpha[2]
alpha[1]
alpha[3]
alpha[1]
alpha[4]
alpha[1]
alpha[5]
alpha[1]
alpha[6]
alpha[1]
alpha[7]
alpha[1]
alpha[3]
alpha[2]
alpha[4]
alpha[2]
alpha[5]
alpha[2]
alpha[6]
alpha[2]
alpha[7]
alpha[2]
alpha[4]
alpha[3]
alpha[5]
alpha[3]
alpha[6]
alpha[3]
alpha[7]
alpha[3]
alpha[5]
alpha[4]



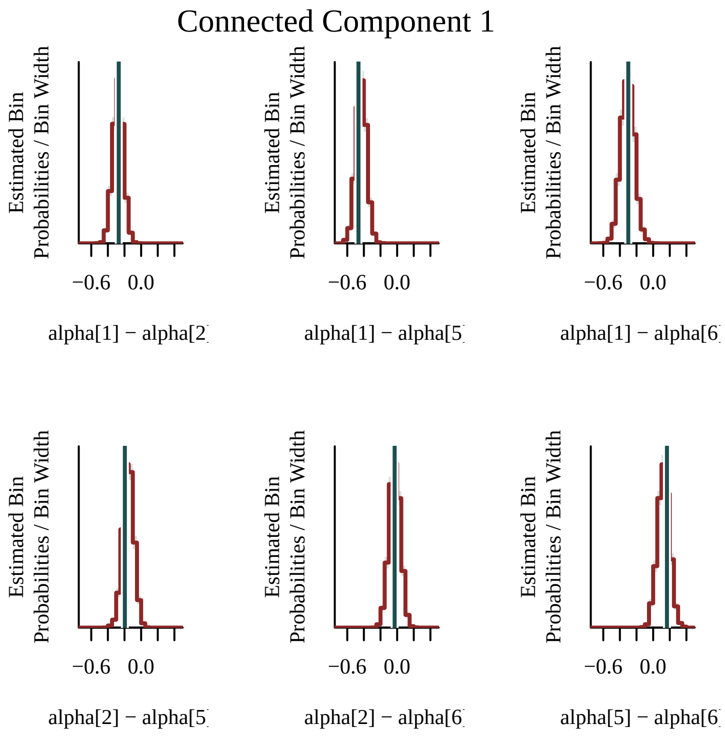
Because the inferential degeneracy persists within the containment of the prior model the observed data does not provide any additional information about the sum of the player skills within each cohort.

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

var_repl <- list('alpha'=array(sapply(components[[1]],
                                      function(i)
                                      paste0("alpha[", i, "]"))))
sum_vals <- util$eval_expectand_pushforward(samples,
                                            function(alpha) sum(alpha),
                                            var_repl)
util$plot_expectand_pushforward(sum_vals, 20, flim=c(-2, 2),
                                display_name="sum(alpha)",
                                main="Connected Component 1")

xs <- seq(-2, 2, 0.01)
ys <- dnorm(xs, 0, sqrt(length(components[[1]])) * log(sqrt(2)) / 2.32)
lines(xs, ys, lwd=2, col=util$c_light_teal)

var_repl <- list('alpha'=array(sapply(components[[2]],
                                      function(i)
                                      paste0("alpha[", i, "]"))))
sum_vals <- util$eval_expectand_pushforward(samples,
                                            function(alpha) sum(alpha),
                                            var_repl)
util$plot_expectand_pushforward(sum_vals, 20, flim=c(-2, 2),
                                display_name="sum(alpha)",
                                main="Connected Component 2")

xs <- seq(-2, 2, 0.01)
ys <- dnorm(xs, 0, sqrt(length(components[[2]])) * log(sqrt(2)) / 2.32)
lines(xs, ys, lwd=2, col=util$c_light_teal)
```

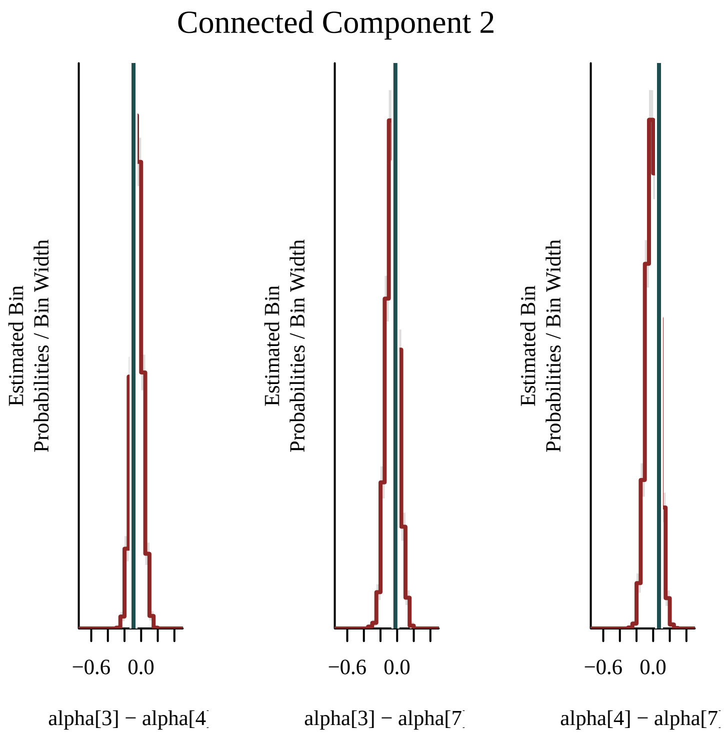


Instead the observations inform only the differences between the player skills in each cohort.

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

N <- length(components[[1]])
for (i1 in 1:(N - 1)) {
  for (i2 in (i1 + 1):N) {
    idx1 <- components[[1]][i1]
    idx2 <- components[[1]][i2]
    name1 <- paste0('alpha[', idx1, ']')
    name2 <- paste0('alpha[', idx2, ']')
    diff_vals <- util$eval_expectand_pushforward(samples,
                                                 function(a1, a2)
                                                 a1 - a2,
                                                 list('a1'=name1,
                                                      'a2'=name2))
    util$plot_expectand_pushforward(diff_vals,
                                    25, flim=c(-0.75, 0.5),
                                    baseline=alpha_true[idx1] -
                                             alpha_true[idx2],
                                    baseline_col=util$c_dark_teal,
                                    display_name=paste(name1, '-',
                                                       name2))
  }
}

mtext("Connected Component 1", line=-2, outer=TRUE)
```



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

N <- length(components[[2]])
for (i1 in 1:(N - 1)) {
  for (i2 in (i1 + 1):N) {
    idx1 <- components[[2]][i1]
    idx2 <- components[[2]][i2]
    name1 <- paste0('alpha[', idx1, ']')
    name2 <- paste0('alpha[', idx2, ']')
    diff_vals <- util$eval_expectand_pushforward(samples,
                                                 function(a1, a2)
                                                 a1 - a2,
                                                 list('a1'=name1,
                                                      'a2'=name2))
    util$plot_expectand_pushforward(diff_vals,
                                    25, flim=c(-0.75, 0.5),
                                    baseline=alpha_true[idx1] -
                                             alpha_true[idx2],
                                    baseline_col=util$c_dark_teal,
                                    display_name=paste(name1, '-',
                                                       name2))
  }
}

mtext("Connected Component 2", line=-2, outer=TRUE)
```



6.3.6 Attempt 4

Because we took the time to thoroughly investigate the inferential degeneracies of pairwise comparison models in Section 4 we have more effective tools at our disposal than just the prior model.

Let's go back to the more diffuse prior model that we introduced in Section 6.3.4 but anchor one of the player skills to zero to eliminate some of the redundancy in the model. Note that we have to be careful to update the prior model to accommodate the now relative skill parameters.

```
fit <- stan(file="stan_programs/bradley_terry4.stan",
            data=data, seed=8438338,
            warmup=1000, iter=2024, refresh=0)
```

None of the diagnostics indicate incomplete posterior exploration.

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

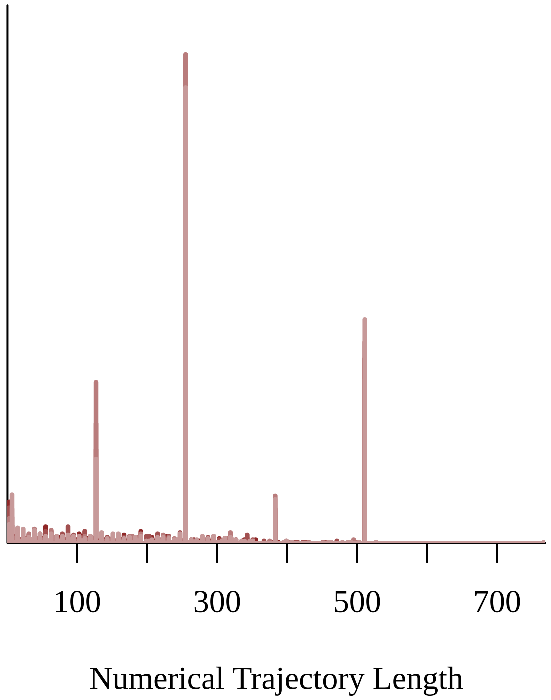
```
  All Hamiltonian Monte Carlo diagnostics are consistent with reliable
Markov chain Monte Carlo.
```

```
samples <- util$extract_expectand_vals(fit)
base_samples <- util$filter_expectands(samples,
                                       c('delta_free'),
                                       check_arrays=TRUE)
util$summarize_expectand_diagnostics(base_samples)
```

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

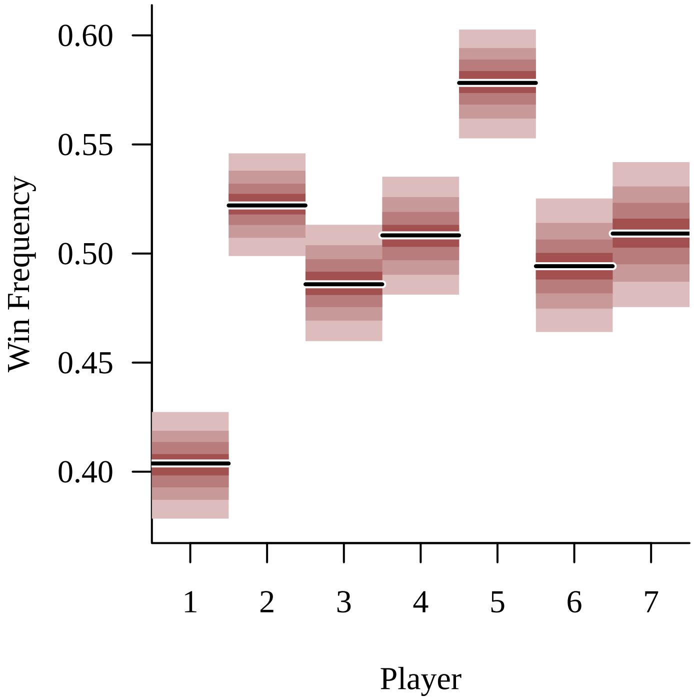
That said we're back to needing hundreds of model evaluations to inform each Markov transition.

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

util$plot_num_leapfrogs(diagnostics)
```



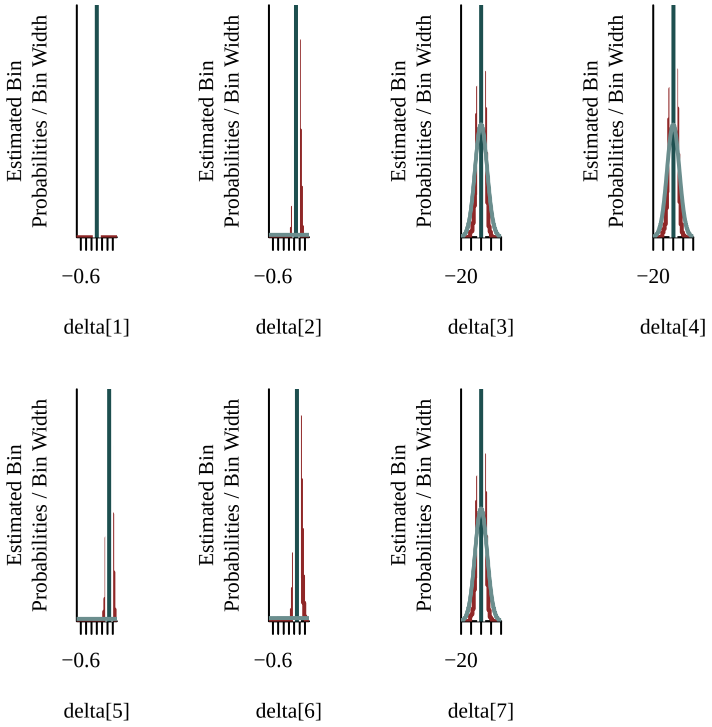
The posterior retrodictive performance remains high.

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

names <- sapply(1:data$N_players,
                function(i) paste0('win_freq_pred[', i, ']'))
util$plot_disc_pushforward_quantiles(samples, names,
                                     baseline_values=win_counts /
                                                     game_counts,
                                     xlab="Player",
                                     ylab="Win Frequency")
```

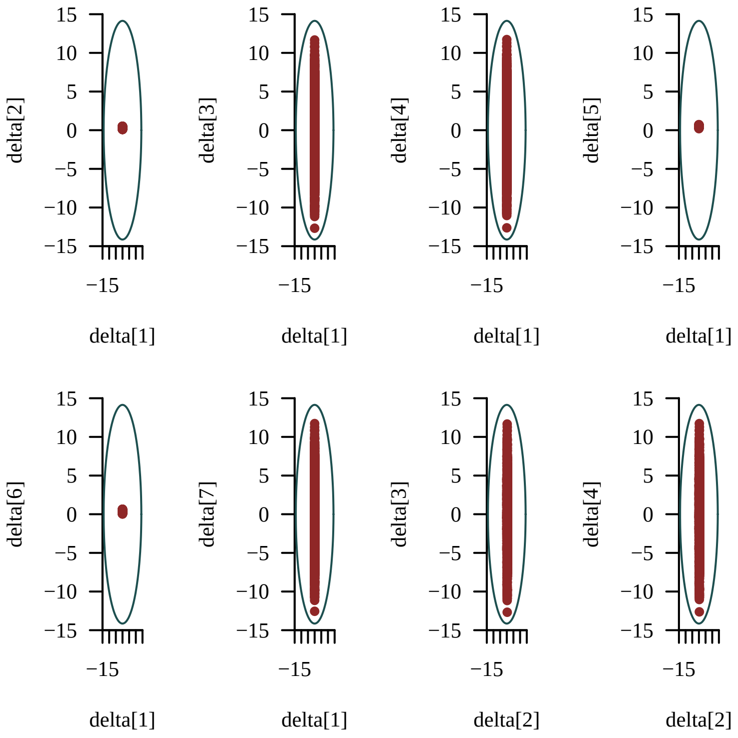


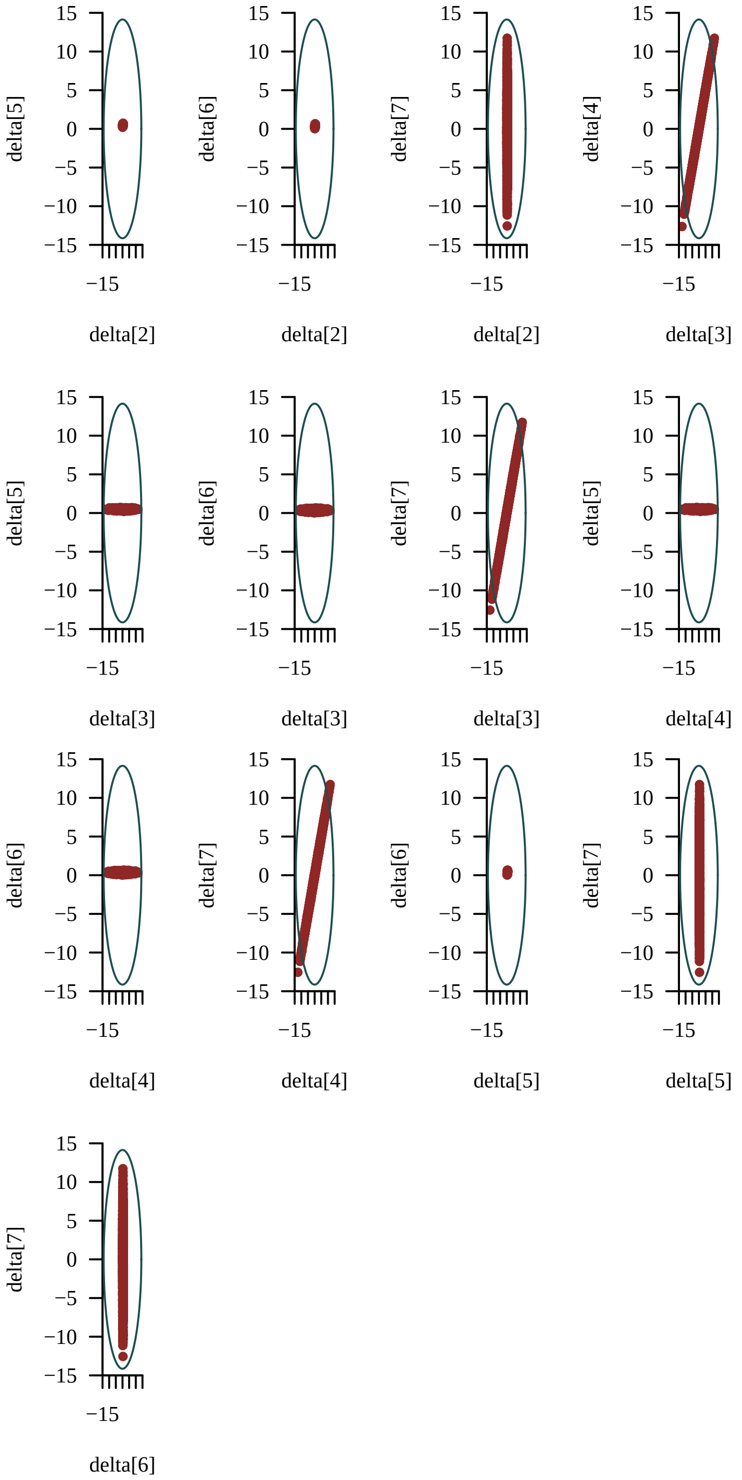
With the anchor in place posterior inferences for some of the relative player skills are now being informed by the observed data! That said some of the relative player skills are still being informed mostly by the prior model.

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

plot_lim <- rep(0, data$N_players)
plot_lim[components[[1]]] <- 0.75
plot_lim[components[[2]]] <- 20

for (i in 1:data$N_players) {
  name <- paste0('delta[', i, ']')
  util$plot_expectand_pushforward(samples[[name]],
                                  35,
                                  flim=c(-plot_lim[i], +plot_lim[i]),
                                  baseline=alpha_true[i] -
                                           alpha_true[1],
                                  baseline_col=util$c_dark_teal,
                                  display_name=name)
  if (i > 1) {
    xs <- seq(-plot_lim[i], +plot_lim[i], 0.01)
    ys <- dnorm(xs, 0, sqrt(2) * 10 / 2.32)
    lines(xs, ys, lwd=2, col=util$c_light_teal)
  }
}
```



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

for (i in 1:(data$N_players - 1)) {
  for (j in (i + 1):data$N_players) {
    name1 <- paste0('delta[', i, ']')
    name2 <- paste0('delta[', j, ']')
    plot(c(samples[[name1]]), c(samples[[name2]]),
         cex=1, pch=16, col=util$c_dark,
         xlab=name1, xlim=c(-15, 15),
         ylab=name2, ylim=c(-15, 15))
    lines(sqrt(2) * 10 * cos(seq(0, 2 * pi, 2 * pi / 250)),
          sqrt(2) * 10 * sin(seq(0, 2 * pi, 2 * pi / 250)),
          col=util$c_dark_teal)
  }
}
```

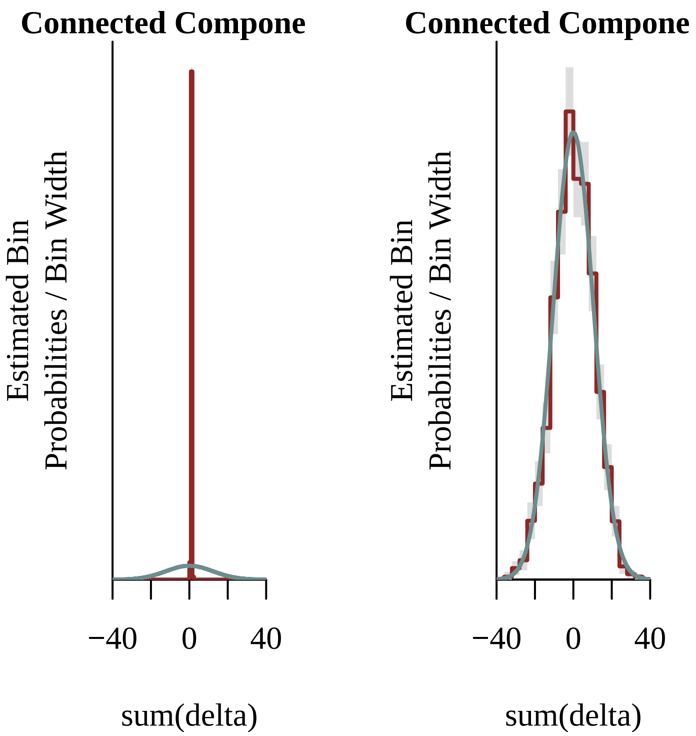




What distinguishes the relative player skills that are now being informed by the observed data and those that aren't? The former are in the same connected component as the anchor player while the latter are not. Setting a player skill to zero identifies the relative behavior of players only in that same connected component!

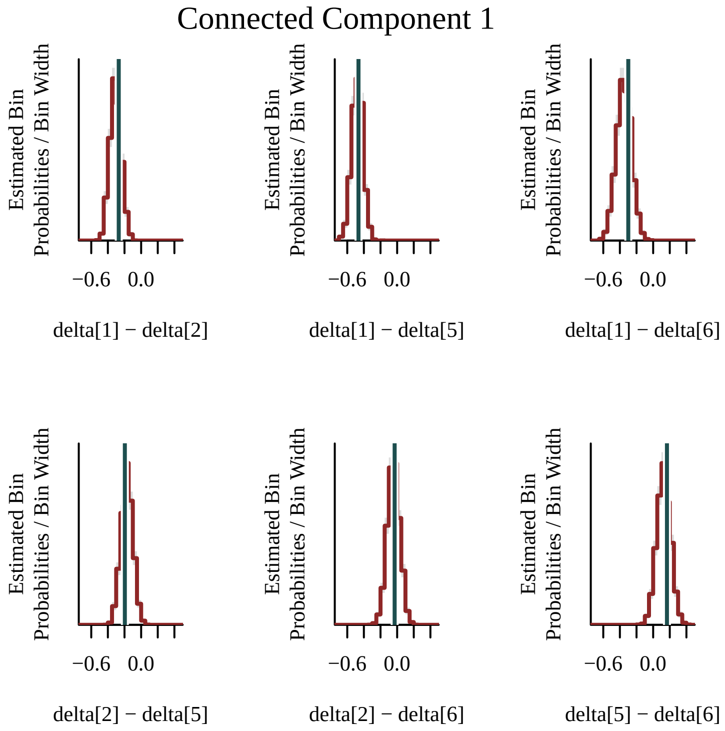
We can verify this by looking at the sum of relative player skills in the two components. Because the first component includes the anchor player the sum of the relative player skills ends up being informed by the observed data. On the other hand the sum of relative player skills in the other component is informed mostly by the prior model.

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

var_repl <- list('delta'=array(sapply(components[[1]],
                                      function(i)
                                      paste0("delta[", i, "]"))))
sum_vals <- util$eval_expectand_pushforward(samples,
                                            function(delta) sum(delta),
                                            var_repl)
util$plot_expectand_pushforward(sum_vals, 100, flim=c(-40, 40),
                                display_name="sum(delta)",
                                main="Connected Component 1")

xs <- seq(-40, 40, 0.01)
ys <- dnorm(xs, 0, sqrt(length(components[[1]])) * sqrt(2) * 10 / 2.32)
lines(xs, ys, lwd=2, col=util$c_light_teal)

var_repl <- list('delta'=array(sapply(components[[2]],
                                      function(i)
                                      paste0("delta[", i, "]"))))
sum_vals <- util$eval_expectand_pushforward(samples,
                                            function(delta) sum(delta),
                                            var_repl)
util$plot_expectand_pushforward(sum_vals, 20, flim=c(-40, 40),
                                display_name="sum(delta)",
                                main="Connected Component 2")

xs <- seq(-40, 40, 0.01)
ys <- dnorm(xs, 0, sqrt(length(components[[2]])) * sqrt(2) * 10 / 2.32)
lines(xs, ys, lwd=2, col=util$c_light_teal)
```

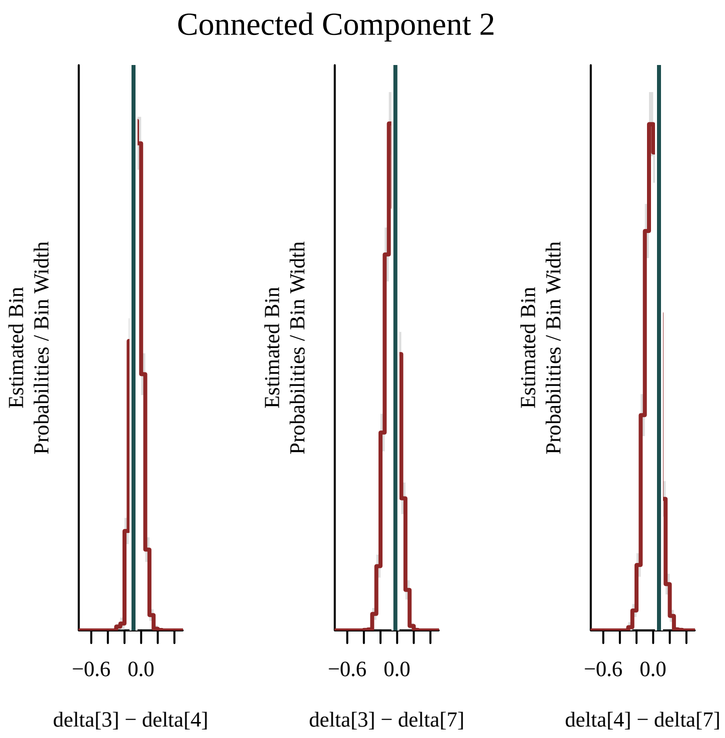


Regardless the differences in player skills within each component remain well-informed by the observed data.

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

N <- length(components[[1]])
for (i1 in 1:(N - 1)) {
  for (i2 in (i1 + 1):N) {
    idx1 <- components[[1]][i1]
    idx2 <- components[[1]][i2]
    name1 <- paste0('delta[', idx1, ']')
    name2 <- paste0('delta[', idx2, ']')
    diff_vals <- util$eval_expectand_pushforward(samples,
                                                 function(a1, a2)
                                                 a1 - a2,
                                                 list('a1'=name1,
                                                      'a2'=name2))
    util$plot_expectand_pushforward(diff_vals,
                                    25, flim=c(-0.75, 0.5),
                                    baseline=alpha_true[idx1] -
                                             alpha_true[idx2],
                                    baseline_col=util$c_dark_teal,
                                    display_name=paste(name1, '-',
                                                       name2))
  }
}

mtext("Connected Component 1", line=-2, outer=TRUE)
```



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

N <- length(components[[2]])
for (i1 in 1:(N - 1)) {
  for (i2 in (i1 + 1):N) {
    idx1 <- components[[2]][i1]
    idx2 <- components[[2]][i2]
    name1 <- paste0('delta[', idx1, ']')
    name2 <- paste0('delta[', idx2, ']')
    diff_vals <- util$eval_expectand_pushforward(samples,
                                                 function(a1, a2)
                                                 a1 - a2,
                                                 list('a1'=name1,
                                                      'a2'=name2))
    util$plot_expectand_pushforward(diff_vals,
                                    25, flim=c(-0.75, 0.5),
                                    baseline=alpha_true[idx1] -
                                             alpha_true[idx2],
                                    baseline_col=util$c_dark_teal,
                                    display_name=paste(name1, '-',
                                                       name2))
  }
}

mtext("Connected Component 2", line=-2, outer=TRUE)
```



6.3.7 Attempt 5

We could avoid this residual degeneracy by splitting the data for each connected component and then analyzing each subset separately. Alternatively we can eliminate it by anchoring the skill of any one player in each component.

```
data$N_anchors <- 2
data$anchor_idx <- c(components[[1]][1], components[[2]][1])
```

Zipping together the anchored player skills and the free player skills requires a little bit of care but otherwise the Stan program is largely unchanged.

```
fit <- stan(file="stan_programs/bradley_terry5.stan",
            data=data, seed=8438338,
            warmup=1000, iter=2024, refresh=0)
```

No computational diagnostic warnings have arisen.

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

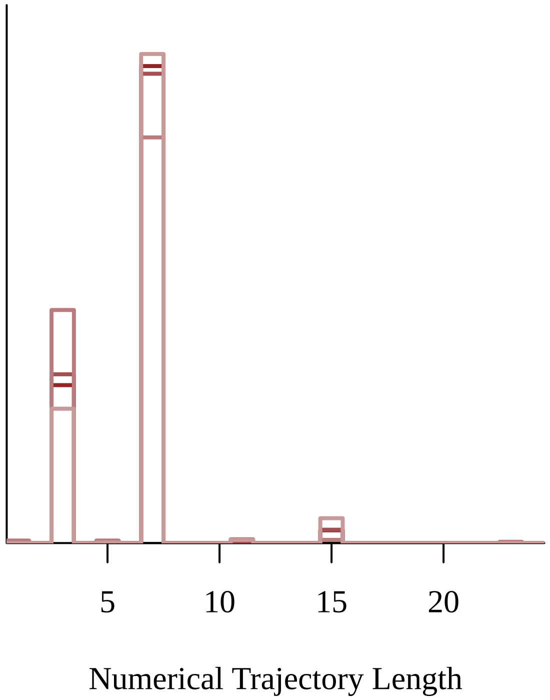
```
  All Hamiltonian Monte Carlo diagnostics are consistent with reliable
Markov chain Monte Carlo.
```

```
samples <- util$extract_expectand_vals(fit)
base_samples <- util$filter_expectands(samples,
                                       c('delta_free'),
                                       check_arrays=TRUE)
util$summarize_expectand_diagnostics(base_samples)
```

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

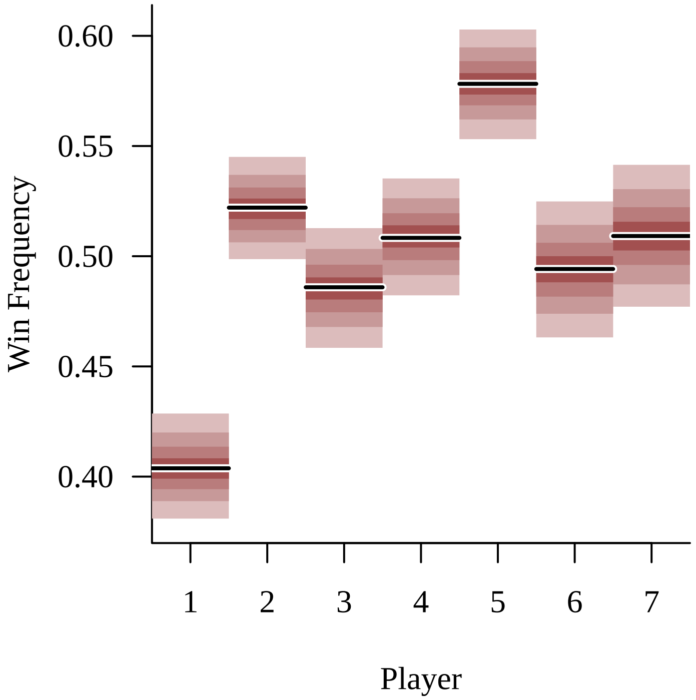
With a second anchor the computational costs have decreased even beyond what was needed for the informative prior model.

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

util$plot_num_leapfrogs(diagnostics)
```



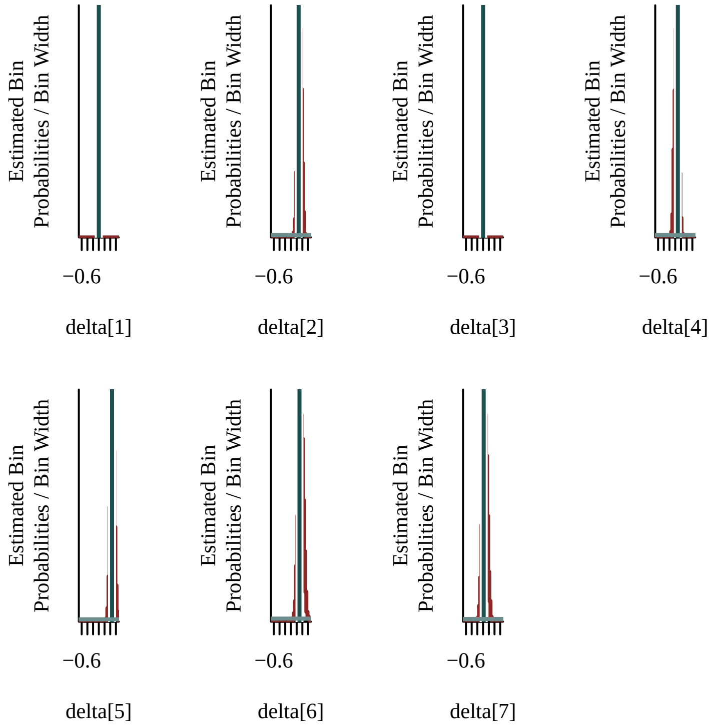
There are no compromises, however, to the retrodictive performance.

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

names <- sapply(1:data$N_players,
                function(i) paste0('win_freq_pred[', i, ']'))
util$plot_disc_pushforward_quantiles(samples, names,
                                     baseline_values=win_counts /
                                                     game_counts,
                                     xlab="Player",
                                     ylab="Win Frequency")
```

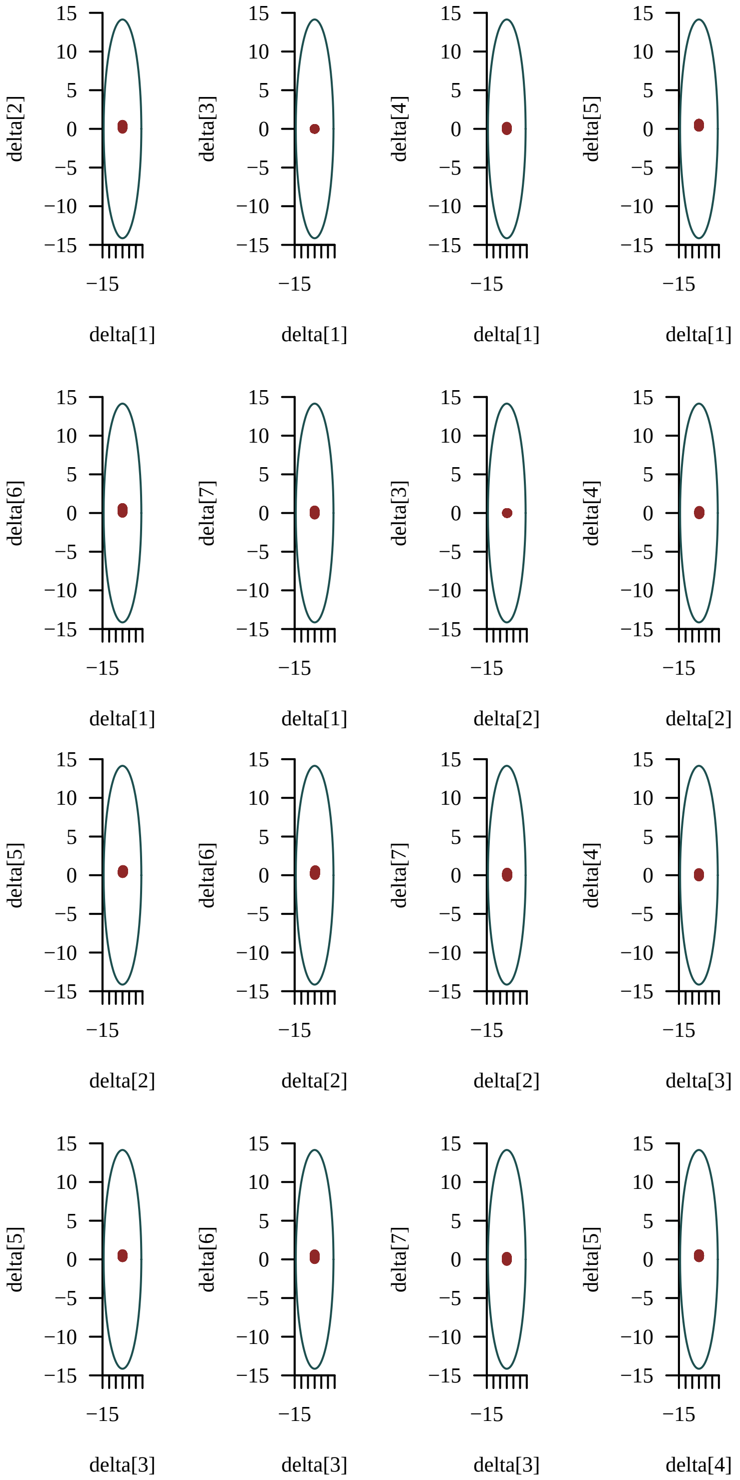


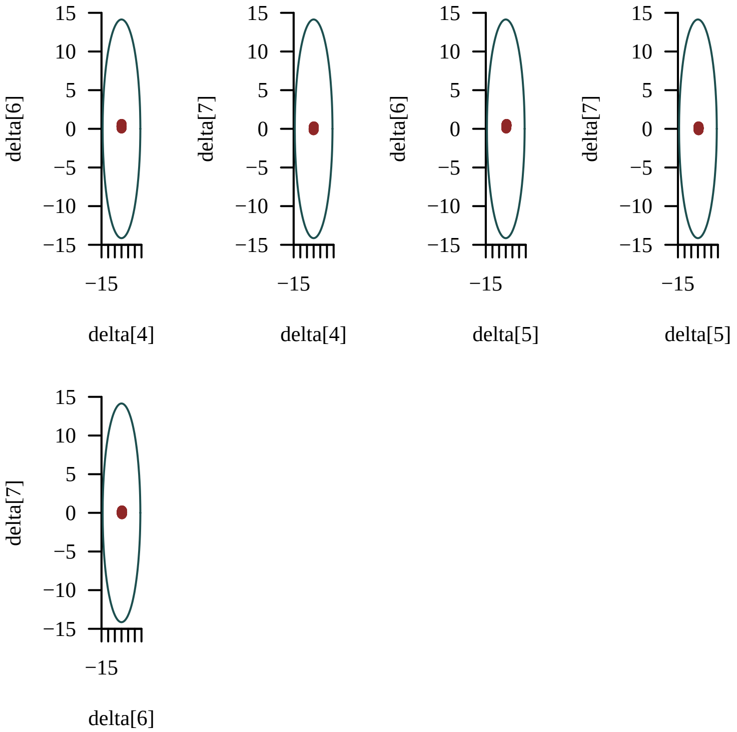
Now the posterior inferences for all of the relative player skills are being dominated by the observed data. Note that in order to compare the inferred player skills to the simulation player skills we need to account for both anchors.

```
delta_true <- rep(NA, data$N_players)
for (i in 1:data$N_players) {
  delta_true[i] <- ifelse(i %in% components[[1]],
                          alpha_true[i] -
                          alpha_true[data$anchor_idx[1]],
                          alpha_true[i] -
                          alpha_true[data$anchor_idx[2]])
}

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

for (i in 1:data$N_players) {
  name <- paste0('delta[', i, ']')
  util$plot_expectand_pushforward(samples[[name]],
                                  35, flim=c(-0.7, +0.7),
                                  baseline=delta_true[i],
                                  baseline_col=util$c_dark_teal,
                                  display_name=name)
  if (!(i %in% data$anchor_idx)) {
    xs <- seq(-0.7, 0.7, 0.01)
    ys <- dnorm(xs, 0, sqrt(2) * 10 / 2.32)
    lines(xs, ys, lwd=2, col=util$c_light_teal)
  }
}
```



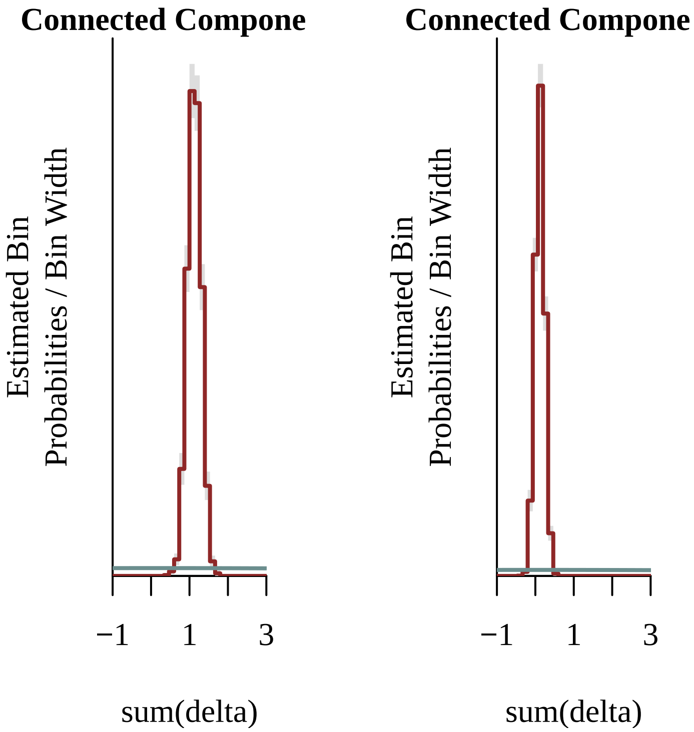
```
par(mfrow=c(2, 4), mar=c(5, 5, 1, 1))

for (i in 1:(data$N_players - 1)) {
  for (j in (i + 1):data$N_players) {
    name1 <- paste0('delta[', i, ']')
    name2 <- paste0('delta[', j, ']')
    plot(c(samples[[name1]]), c(samples[[name2]]),
         cex=1, pch=16, col=util$c_dark,
         xlab=name1, xlim=c(-15, 15),
         ylab=name2, ylim=c(-15, 15))
    lines(sqrt(2) * 10 * cos(seq(0, 2 * pi, 2 * pi / 250)),
          sqrt(2) * 10 * sin(seq(0, 2 * pi, 2 * pi / 250)),
          col=util$c_dark_teal)
  }
}
```


15
10
5
0
−5
−10
−15
−15
delta[2]
delta[1]
delta[3]
delta[1]
delta[4]
delta[1]
delta[5]
delta[1]
delta[6]
delta[1]
delta[7]
delta[1]
delta[3]
delta[2]
delta[4]
delta[2]
delta[5]
delta[2]
delta[6]
delta[2]
delta[7]
delta[2]
delta[4]
delta[3]
delta[5]
delta[3]
delta[6]
delta[3]
delta[7]
delta[3]
delta[5]
delta[4]



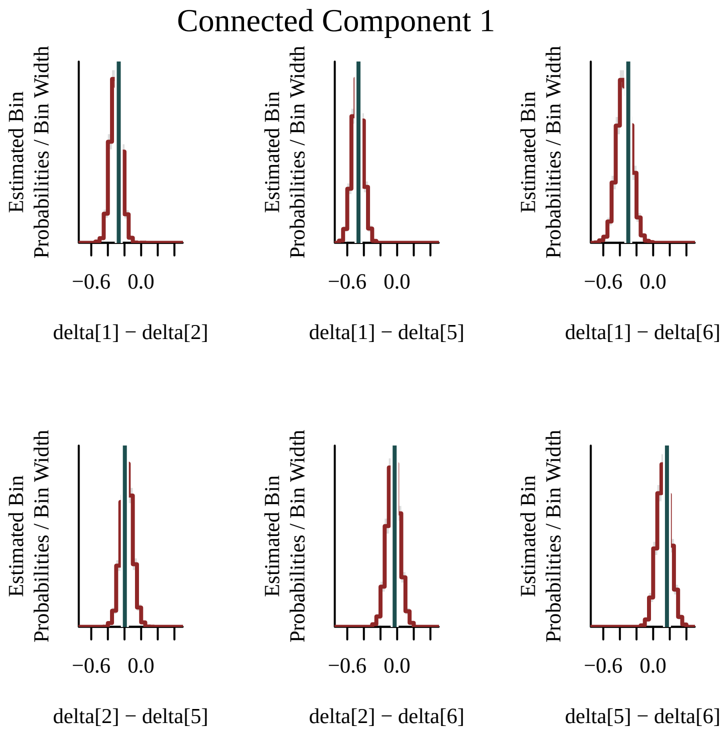
This includes the sum of the relative player skills in each component as well.

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

var_repl <- list('delta'=array(sapply(components[[1]],
                                      function(i)
                                      paste0("delta[", i, "]"))))
sum_vals <- util$eval_expectand_pushforward(samples,
                                            function(delta) sum(delta),
                                            var_repl)
util$plot_expectand_pushforward(sum_vals, 30, flim=c(-1, 3),
                                display_name="sum(delta)",
                                main="Connected Component 1")

xs <- seq(-3, 3, 0.01)
ys <- dnorm(xs, 0, sqrt(length(components[[1]])) * sqrt(2) * 10 / 2.32)
lines(xs, ys, lwd=2, col=util$c_light_teal)

var_repl <- list('delta'=array(sapply(components[[2]],
                                      function(i)
                                      paste0("delta[", i, "]"))))
sum_vals <- util$eval_expectand_pushforward(samples,
                                                function(delta) sum(delta),
                                                var_repl)
util$plot_expectand_pushforward(sum_vals, 30, flim=c(-1, 3),
                                display_name="sum(delta)",
                                main="Connected Component 2")

xs <- seq(-3, 3, 0.01)
ys <- dnorm(xs, 0, sqrt(length(components[[2]])) * sqrt(2) * 10 / 2.32)
lines(xs, ys, lwd=2, col=util$c_light_teal)
```

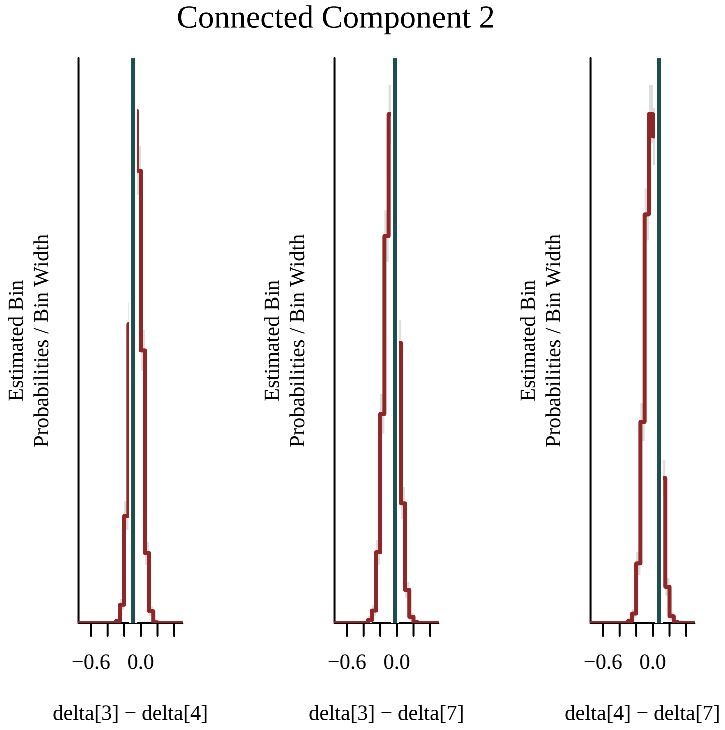


At the same time the constraints on the difference in player skills *within each connected component* remains the same. Now that we've anchored the skills in more than one connected component we have to take care to not mistakenly compare players from different components to each other.

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

N <- length(components[[1]])
for (i1 in 1:(N - 1)) {
  for (i2 in (i1 + 1):N) {
    idx1 <- components[[1]][i1]
    idx2 <- components[[1]][i2]
    name1 <- paste0('delta[', idx1, ']')
    name2 <- paste0('delta[', idx2, ']')
    diff_vals <- util$eval_expectand_pushforward(samples,
                                                 function(a1, a2)
                                                 a1 - a2,
                                                 list('a1'=name1,
                                                      'a2'=name2))

    util$plot_expectand_pushforward(diff_vals,
                                    25, flim=c(-0.75, 0.5),
                                    baseline=alpha_true[idx1] -
                                             alpha_true[idx2],
                                    baseline_col=util$c_dark_teal,
                                    display_name=paste(name1, '-',
                                                       name2))
  }
}

mtext("Connected Component 1", line=-2, outer=TRUE)
```



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

N <- length(components[[2]])
for (i1 in 1:(N - 1)) {
  for (i2 in (i1 + 1):N) {
    idx1 <- components[[2]][i1]
    idx2 <- components[[2]][i2]
    name1 <- paste0('delta[', idx1, ']')
    name2 <- paste0('delta[', idx2, ']')
    diff_vals <- util$eval_expectand_pushforward(samples,
                                                 function(a1, a2)
                                                 a1 - a2,
                                                 list('a1'=name1,
                                                      'a2'=name2))
    util$plot_expectand_pushforward(diff_vals,
                                    25, flim=c(-0.75, 0.5),
                                    baseline=alpha_true[idx1] -
                                             alpha_true[idx2],
                                    baseline_col=util$c_dark_teal,
                                    display_name=paste(name1, '-',
                                                       name2))
  }
}

mtext("Connected Component 2", line=-2, outer=TRUE)
```



Ultimately all of these attempts but the first yielded solid inferences for the differences in player skills, which inform not only retrodictive game outcomes but also predictions for new game outcomes. The benefit a stronger prior model or anchored player skills is a substantial reduction in computational cost. Between the two approaches the advantage of anchoring is that is requires only some careful programming, which we can always implement, and not abundant domain expertise, which is not always available, especially when trying to compare items from disconnected groups.

6.4 Modeling Item Responses

Having built up some experience with the modeling of binary pairwise comparisons let's now consider an item response theory application. In particular we will attempt to analyze data collected from two cohorts of 250 students who have each taken standardized tests consisting of 100 questions.

Critically these two tests were not identical. Only 50 of the questions were shared between the two tests, with the 50 remaining questions in each test unique to that test. In other words the two tests spanned 150 total questions: 50 specific to the first test, 50 specific to the second test, and 50 shared between the two.

6.4.1 Explore Data

The testing data consists of responses for each student/question pair.

```
data <- read_rdump('data/irt.data.R')
```

The 50 questions that are overlap in the two tests are answered more than the 100 questions that are unique to one of the tests.

```
q_n_answers <- sapply(1:data$N_questions,
                      function(i)
                      length(data$y[data$question_idx == i]))
table(q_n_answers)
```

```
q_n_answers
250 500
100  50
```

All of the students, however, answer the same number of questions.

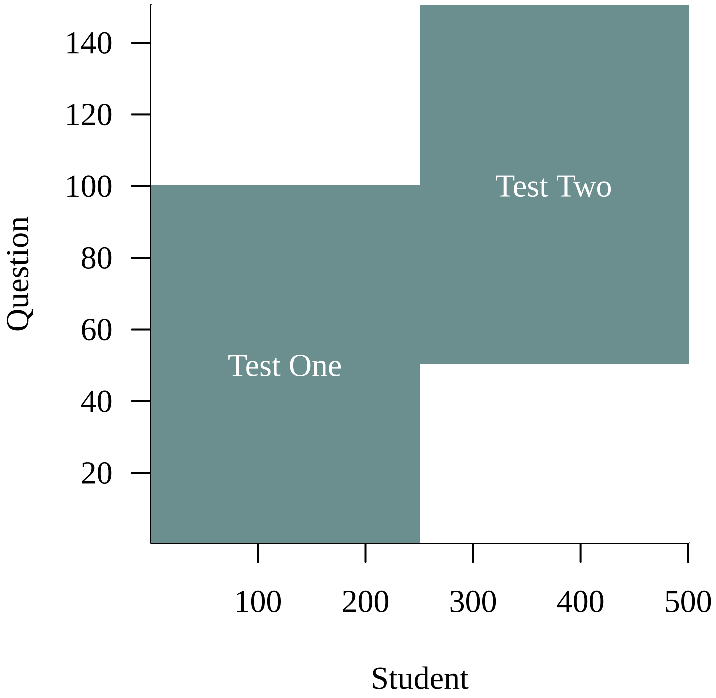
```
s_n_answers <- sapply(1:data$N_students,
                      function(j)
                      length(data$y[data$student_idx == j]))
table(s_n_answers)
```

```
s_n_answers
100
500
```

We can visualize the entire test configuration by plotting the observed student/question pairs. This clearly demonstrates the overlapping questions between the tests, and hence the two student cohorts.

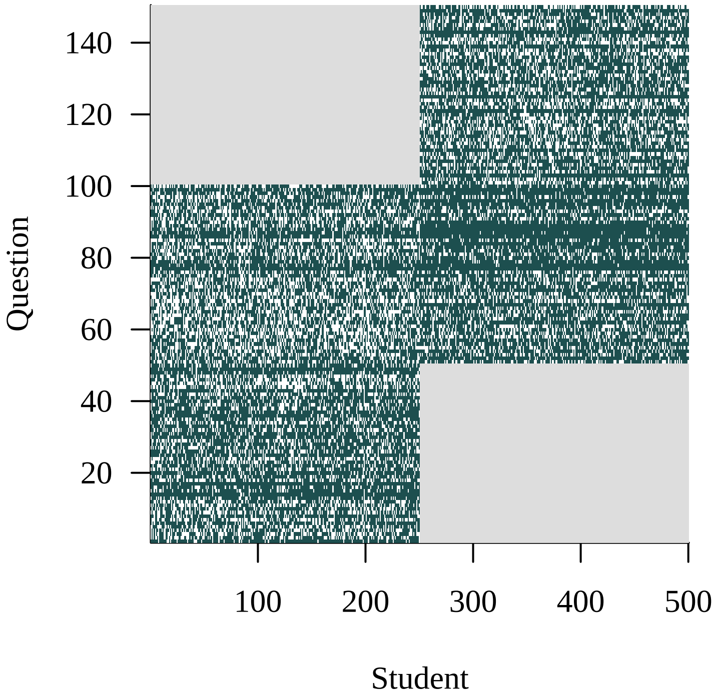
```
xs <- seq(1, data$N_students, 1)
ys <- seq(1, data$N_questions, 1)
zs <- matrix(0, nrow=data$N_students, ncol=data$N_questions)

for (a in 1:data$N_answers) {
  zs[data$student_idx[a], data$question_idx[a]] <- 1
}
```

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

image(xs, ys, zs, col=c("white", util$c_light_teal),
      xlab="Student", ylab="Question")

text(0.25 * 500, 50,  "Test One", col="white")
text(0.75 * 500, 100, "Test Two", col="white")
```



We can also visualize the answers for each of those pairs. Here grey indicates an unobserved pair, white an observed incorrect answer, and teal an observed correct answer.

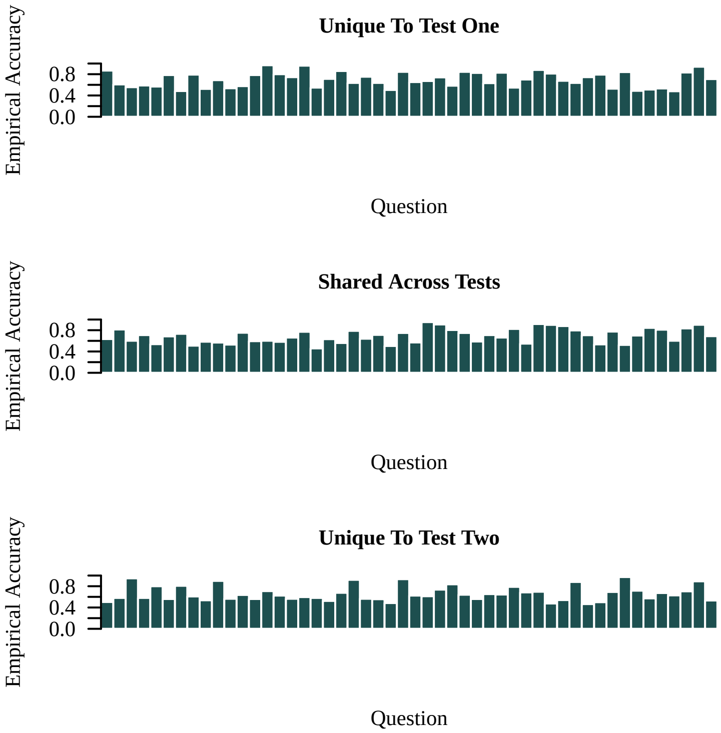
```
zs <- matrix(-1, nrow=data$N_students, ncol=data$N_questions)

for (a in 1:data$N_answers) {
  zs[data$student_idx[a], data$question_idx[a]] <- data$y[a]
}

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

image(xs, ys, zs, col=c("#DDDDDD", "white", util$c_dark_teal),
      xlab="Student", ylab="Question")
```



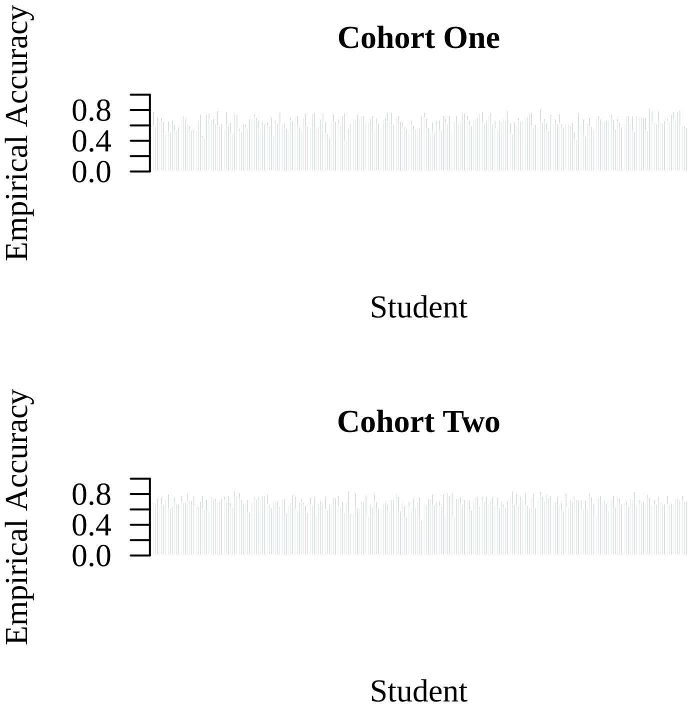
The empirical frequency of correct answers, equivalently the empirical accuracy, varies pretty strongly across both questions and students. This suggests that some questions are more difficult than others while some students are better at taking tests than others.

```
mean_q <- sapply(1:data$N_questions,
                 function(i) mean(data$y[data$question_idx == i]))

mean_s <- sapply(1:data$N_student,
                 function(j) mean(data$y[data$student_idx == j]))
```

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

barplot(mean_q[1:50],    space=0, col=util$c_dark_teal, border="white",
        main="Unique To Test One",
        xlab="Question", ylab="Empirical Accuracy", ylim=c(0, 1))

barplot(mean_q[51:100],  space=0, col=util$c_dark_teal, border="white",
        main="Shared Across Tests",
        xlab="Question", ylab="Empirical Accuracy", ylim=c(0, 1))

barplot(mean_q[101:150], space=0, col=util$c_dark_teal, border="white",
        main="Unique To Test Two",
        xlab="Question", ylab="Empirical Accuracy", ylim=c(0, 1))
```

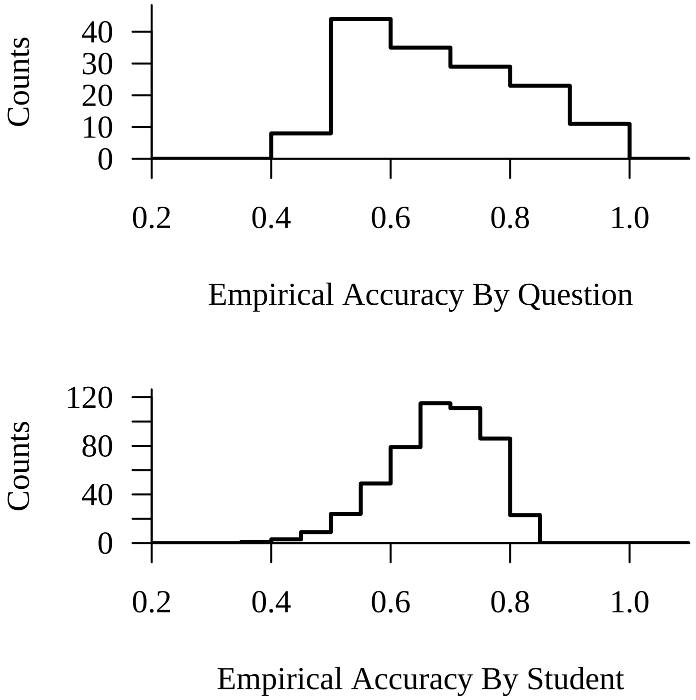


```
par(mfrow=c(2, 1), mar=c(5, 5, 3, 1))

barplot(mean_s[1:250],   space=0, col=util$c_dark_teal, border="white",
        main="Cohort One",
        xlab="Student", ylab="Empirical Accuracy", ylim=c(0, 1))

barplot(mean_s[251:500], space=0, col=util$c_dark_teal, border="white",
        main="Cohort Two",
        xlab="Student", ylab="Empirical Accuracy", ylim=c(0, 1))
```



Because there are so many questions and students a histogram of these stratified empirical accuracies can offer a more digestible summary.

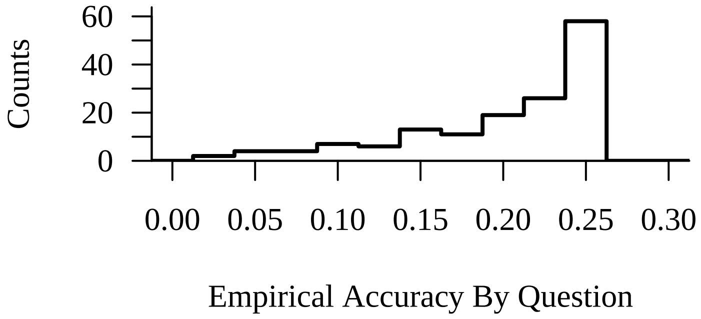
```
par(mfrow=c(2, 1), mar=c(5, 5, 1, 1))

util$plot_line_hist(mean_q, 0.2, 1.1, 0.1,
                    xlab="Empirical Accuracy By Question")

util$plot_line_hist(mean_s, 0.2, 1.1, 0.05,
                    xlab="Empirical Accuracy By Student")
```

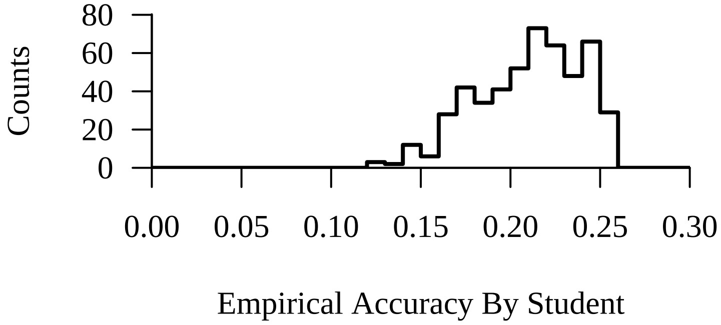


The variance of the responses also varies across questions and students. Note that we can see the discretization of the empirical variances due to the binary values of the answers.

```r
var_q <- sapply(1:data$N_questions,
                 function(i) var(data$y[data$question_idx == i]))

var_s <- sapply(1:data$N_student,
                 function(j) var(data$y[data$student_idx == j]))
```

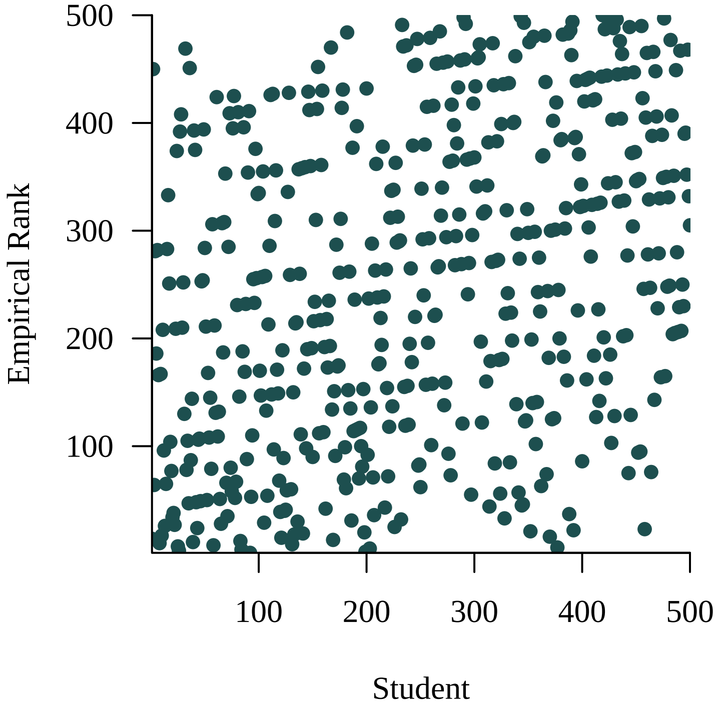
```r
par(mfrow=c(2, 1), mar=c(5, 5, 1, 1))

util$plot_line_hist(var_q,  0, 0.3, 0.025,
                    xlab="Empirical Accuracy By Question")

util$plot_line_hist(var_s,  0, 0.3, 0.01,
                    xlab="Empirical Accuracy By Student")
```





If all of these questions were of the same difficulty then we would be able to at least approximately rank the test-taking ability of the students based on how many questions they answered correctly.

```
empirical_ordering <- sort(mean_s, index.return=TRUE)$ix
empirical_ranks <- sapply(1:data$N_students,
                          function(j) which(empirical_ordering == j))

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

plot(1:data$N_students, empirical_ranks,
     pch=16, col=util$c_dark_teal,
     xlab="Student", ylab="Empirical Rank")
```



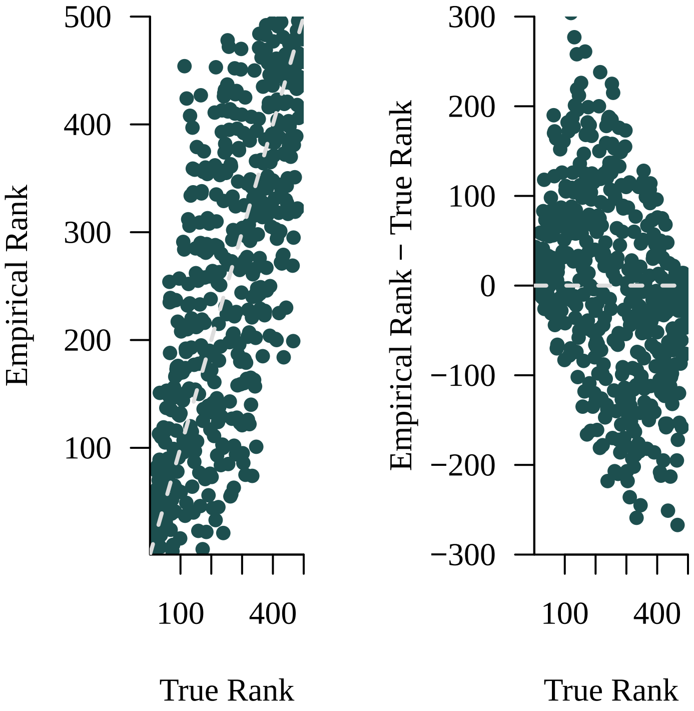
When the question difficulties are not uniform, however, these empirical rankings can be extremely misleading.

Here let's define a true ranking from the true test-taking aptitude of each student and compare it to the empirical ranking.

```
truth <- read_rdump('data/irt.truth.R')

true_ordering <- sort(truth$alpha_true, index.return=TRUE)$ix
true_ranks <- sapply(1:data$N_students,
                     function(j) which(true_ordering == j))
```

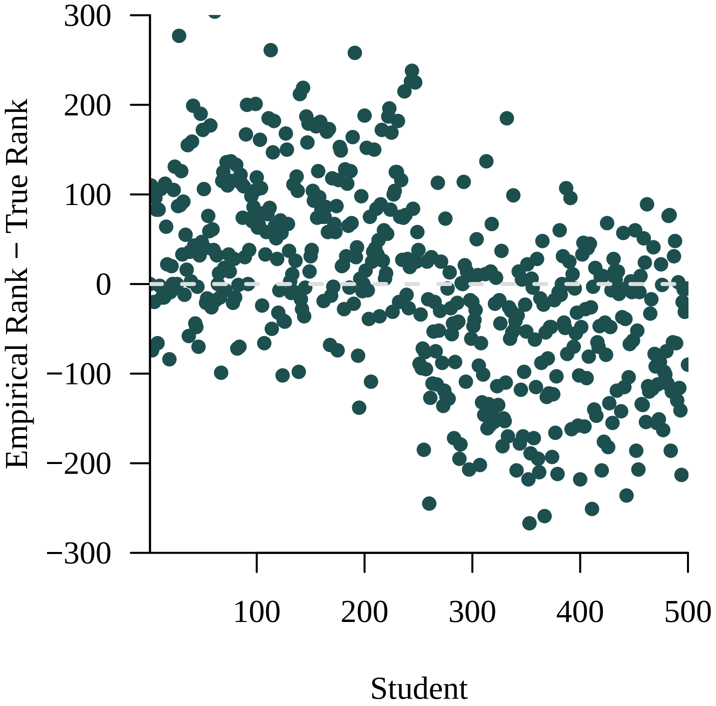
There is some weak correlation between the empirical ranking and the true ranking, but overall the empirical ranking poorly estimates the true ranking.

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

plot(true_ranks, empirical_ranks, pch=16, col=util$c_dark_teal,
     xlab="True Rank", ylab="Empirical Rank")
lines(c(0, 500), c(0, 500), lwd=2, lty=2, col="#DDDDDD")

plot(true_ranks, empirical_ranks - true_ranks,
     pch=16, col=util$c_dark_teal,
     xlab="True Rank", ylab="Empirical Rank - True Rank",
     ylim=c(-300, 300))
abline(h=0, lwd=2, lty=2, col="#DDDDDD")
```



The limitation of the empirical ranking becomes especially clear if we look at the rank differences across the two student cohorts. The empirical accuracies appear to be substantially overstated the test-taking ability of the students in the first cohort while underestimate those in the second cohort.

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

plot(1:data$N_students, empirical_ranks - true_ranks,
     pch=16, col=util$c_dark_teal,
     xlab="Student", ylab="Empirical Rank - True Rank",
     ylim=c(-300, 300))
abline(h=0, lwd=2, lty=2, col="#DDDDDD")
```



In order to inform a more faithful ranking of the students' test-taking abilities we will need to model not only the student behavior but also the question behavior.

6.4.2 Attempt 1

Let's begin with a 1PL/Rasch model.

To minimize the chances of any inferential degeneracies we'll anchor the first student ability to zero. In this case the remaining α_j can be interpreted as the student abilities relative to the first student, and the question difficulties β_i can be interpreted as the negative logit probability of that first student answering each question correctly.

```
fit <- stan(file="stan_programs/irt1.stan",
            data=data, seed=8438338,
            warmup=1000, iter=2024, refresh=0)
```

A few stray $\hat{ESS}$ warnings.

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

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

```
samples <- util$extract_expectand_vals(fit)
base_samples <- util$filter_expectands(samples,
                                       c('beta', 'alpha_free'),
                                       check_arrays=TRUE)
util$summarize_expectand_diagnostics(base_samples)
```

```
The expectands beta[56], beta[58] triggered diagnostic warnings.

The expectands beta[56], beta[58] triggered hat{ESS} warnings.

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

That said the empirical effective sample sizes of concern are right below the warning threshold. Across all four Markov chains we should have collected enough information to accurately estimate posterior expectation values.

```
base_samples <- util$filter_expectands(samples,
                                       c('beta[56]', 'beta[58]'))
util$check_all_expectand_diagnostics(base_samples)
```

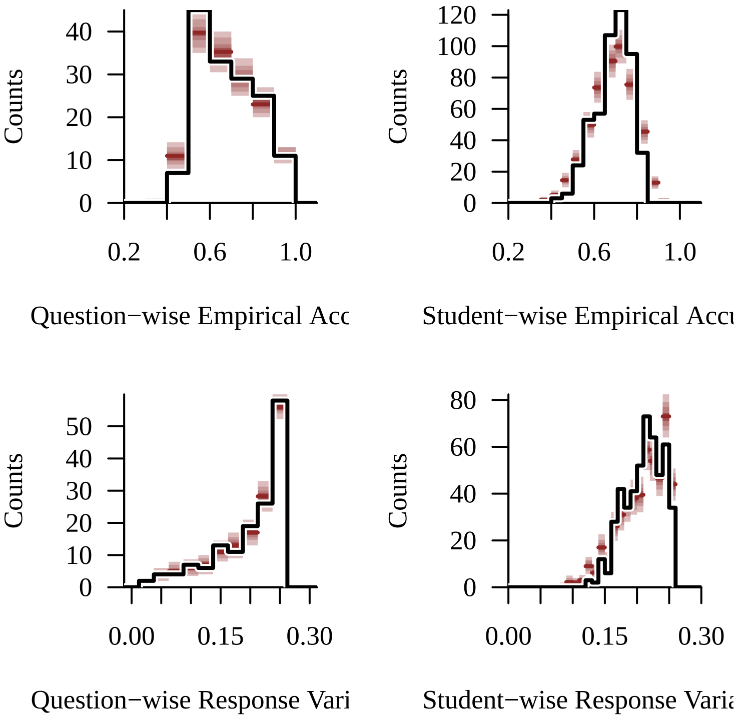
```
beta[56]:
  Chain 1: hat{ESS} (90.610) is smaller than desired (100).

beta[58]:
  Chain 1: hat{ESS} (88.748) is smaller than desired (100).


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

To probe the adequacy of our model let's consult the histograms of stratified empirical means and variances that we considered in Section 6.4.1. The retrodictive behavior is consistent for the question-wise behaviors, but not for the student-wise behaviors. In particular the posterior predictive student-wise behaviors exhibit more variation than the observed behaviors.

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

util$plot_hist_quantiles(samples, 'mean_q_pred', 0.2, 1.1, 0.1,
                         baseline_values=mean_q,
                         xlab="Question-wise Empirical Accuracy")

util$plot_hist_quantiles(samples, 'mean_s_pred', 0.2, 1.1, 0.05,
                         baseline_values=mean_s,
                         xlab="Student-wise Empirical Accuracy")
```

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

```
util$plot_hist_quantiles(samples, 'var_q_pred', 0, 0.3, 0.025,
                         baseline_values=var_q,
                         xlab="Question-wise Response Variances")

util$plot_hist_quantiles(samples, 'var_s_pred', 0, 0.3, 0.01,
                         baseline_values=var_s,
                         xlab="Student-wise Response Variances")
```



6.4.3 Attempt 2

Our first model's ability to recover the question-wise behaviors but not the student-wise behaviors suggests that something is moderating the student-wise behaviors. Specifically the over-dispersion exhibited by the posterior predictive distribution indicates that the correct answer probabilities for each student should be closer to 0.5.

One phenomenon that can cause this exact behavior are questions that don't probe the student abilities as much as we might have hoped. In this case student behavior would resemble random guessing more often, pulling the student-wise probabilities closer to 0.5.

For example we might consider expanding our model into a 2PL model with

$$\begin{aligned} \operatorname{logit} p_{ji} &= \gamma_i \cdot (\alpha_j - \beta_i) \\ &= \gamma_i \cdot \alpha_j - \gamma_i \cdot \beta_i. \end{aligned}$$

The question difficulties in the 1PL model can contort themselves to mimic the second term,

$$\beta_i^{\text{1PL}} = \gamma_i \cdot \beta_i^{\text{2PL}},$$

but the student abilities in the 1PL cannot match the question-wise moderation of the first term.

To prevent as many inferential degeneracies as possible we will fix the discrimination of the first question to 1 so that $\gamma_i > 0$ implies a question that is more sensitive to student abilities than the first question while $\gamma_i < 0$ implies a question that is less sensitive to student abilities than the first question.

```
fit <- stan(file="stan_programs/irt2.stan",
            data=data, seed=8438338,
            warmup=1000, iter=2024, refresh=0)
```

Pleasantly there are diagnostic warnings.

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

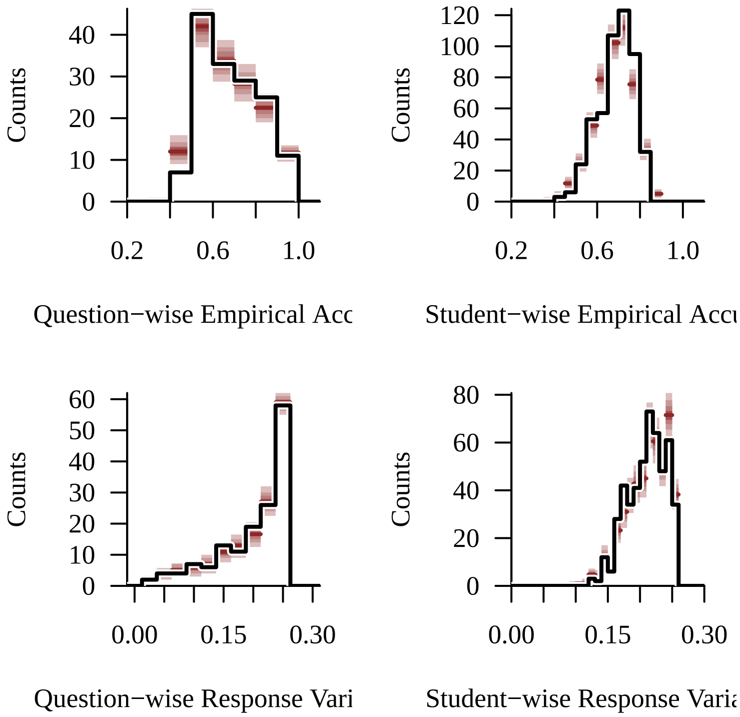
```
  All Hamiltonian Monte Carlo diagnostics are consistent with reliable
Markov chain Monte Carlo.
```

```
samples <- util$extract_expectand_vals(fit)
base_samples <- util$filter_expectands(samples,
                                       c('gamma_free',
                                         'beta',
                                         'alpha_free'),
                                       check_arrays=TRUE)
util$summarize_expectand_diagnostics(base_samples)
```

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

Moreover the retrodictive behavior is now consistent for the both the question-wise behaviors and the student-wise behaviors.

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

util$plot_hist_quantiles(samples, 'mean_q_pred', 0.2, 1.1, 0.1,
                         baseline_values=mean_q,
                         xlab="Question-wise Empirical Accuracy")

util$plot_hist_quantiles(samples, 'mean_s_pred', 0.2, 1.1, 0.05,
                         baseline_values=mean_s,
                         xlab="Student-wise Empirical Accuracy")

util$plot_hist_quantiles(samples, 'var_q_pred', 0, 0.3, 0.025,
                         baseline_values=var_q,
                         xlab="Question-wise Response Variances")

util$plot_hist_quantiles(samples, 'var_s_pred', 0, 0.3, 0.01,
                         baseline_values=var_s,
                         xlab="Student-wise Response Variances")
```

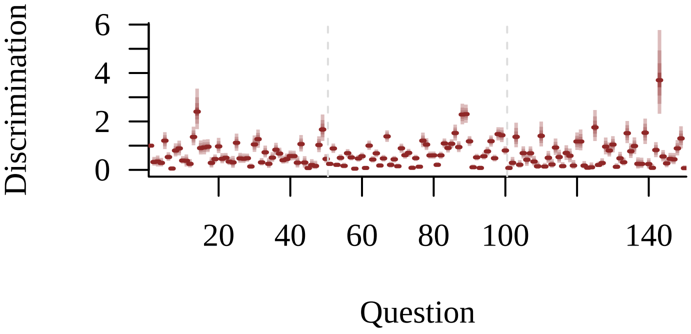


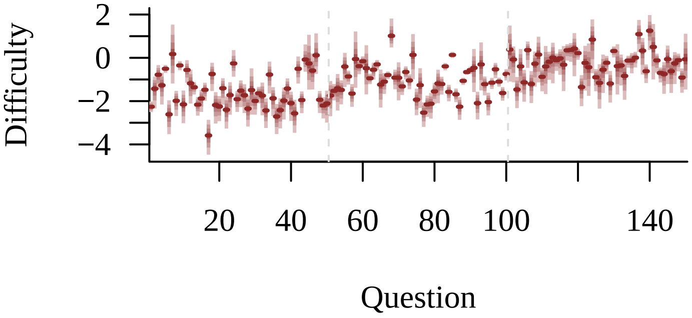
Comfortable in the adequacy of our model we can progress to the analysis of our posterior inferences.

The inferred question discriminations are certainly heterogeneous, but there is no obvious pattern to that variation. On the other hand the inferred question difficulties shift systematically across the tests. In particular the questions unique to the first test tend to be the least difficult while those unique to the second test tend to be the most difficult.

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

names <- sapply(1:data$N_questions,
                function(i) paste0('gamma[', i, ']'))
util$plot_disc_pushforward_quantiles(samples, names,
                                     xlab="Question",
                                     ylab="Discrimination")
abline(v=50.5, lty=2, col="#DDDDDD")
abline(v=100.5, lty=2, col="#DDDDDD")

names <- sapply(1:data$N_questions,
                function(i) paste0('beta[', i, ']'))
util$plot_disc_pushforward_quantiles(samples, names,
                                     xlab="Question",
                                     ylab="Difficulty")
```

```
abline(v=50.5, lty=2, col="#DDDDDD")
abline(v=100.5, lty=2, col="#DDDDDD")
```

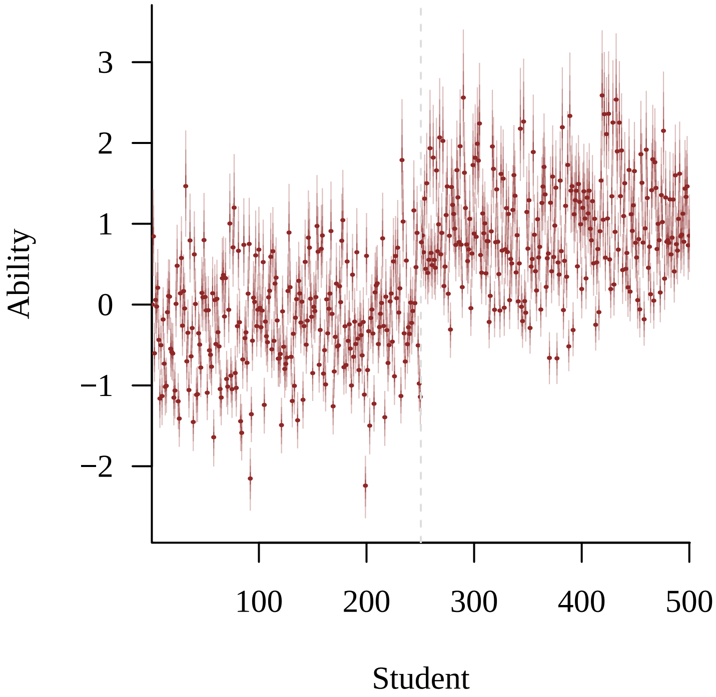




Without more domain expertise we can only speculate about what could cause this behavior. For example if the two tests were given at different times then the questions unique to the first test might have been discarded as too easy but then replaced with questions that were too hard for the second test.

We also see a systematic shift in the student test-taking abilities across the two cohorts. The students who took the second test appear to be on average better at taking tests than the students who took the first test.

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

names <- sapply(1:data$N_students, function(j) paste0('alpha[', j, ']'))
util$plot_disc_pushforward_quantiles(samples, names,
                                     xlab="Student", ylab="Ability")
abline(v=250.5, lty=2, col="#DDDDDD")
```



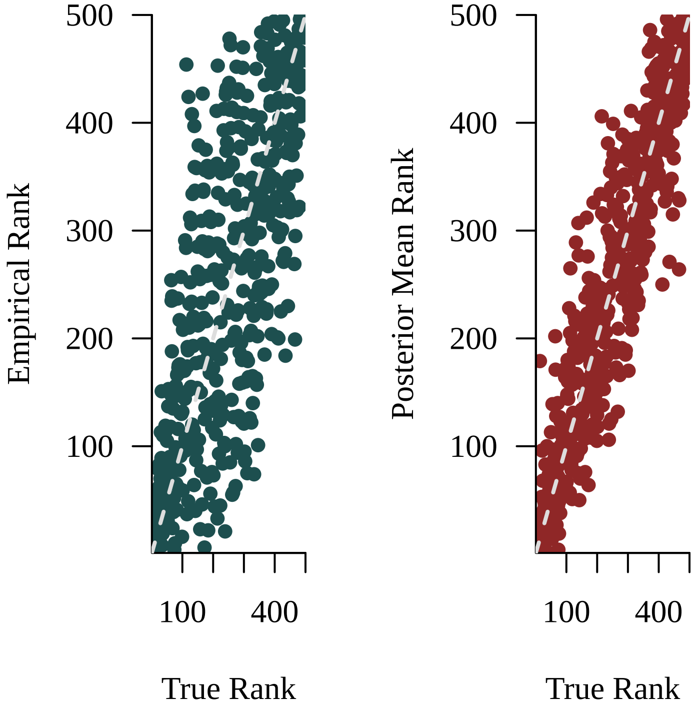
These inferences help to explain why the empirical accuracies are so misleading. Although the students taking the first test were less skilled they were given easier questions. On the other hand the students taking the second test were more skilled but had to contend with more difficult questions. The varying skills and difficulties largely compensate for each other, resulting in relatively uniform empirical performance across both both student cohorts.

Consequently a student ranking informed by the posterior distribution should be closer to the true ranking than the one informed by the empirical accuracies that we considered above. Here we'll consider a ranking informed by the marginal posterior mean of each student skill.

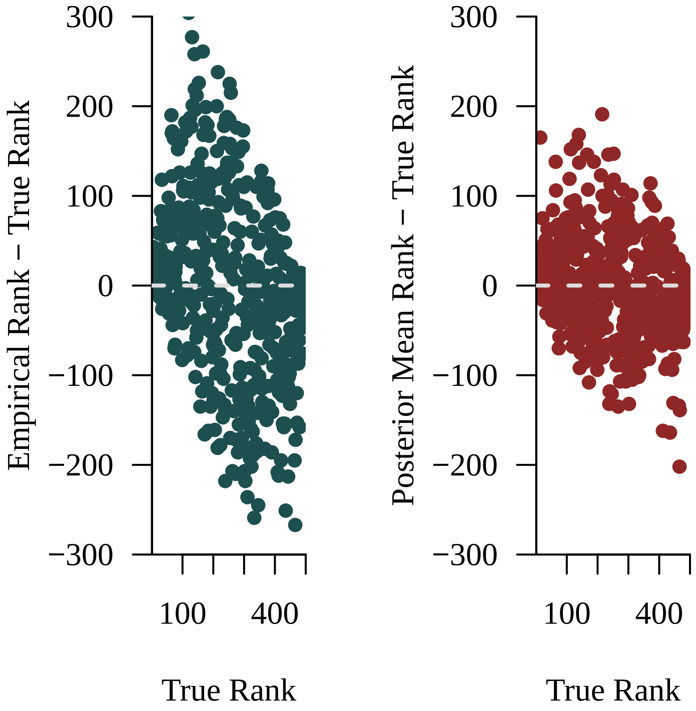
```r
expected_ability <- function(j) {
  util$ensemble_mcmc_est(samples[[paste0('alpha[', j, ']')]])[1]
}

expected_abilities <- sapply(1:data$N_students,
                             function(j) expected_ability(j))

post_mean_ordering <- sort(expected_abilities, index.return=TRUE)$ix
post_mean_ranks <- sapply(1:data$N_students,
                          function(j) which(post_mean_ordering == j))
```

Indeed we see much stronger correlation between the true rank and a posterior mean rank than between the true rank and the naive empirical rank.

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

plot(true_ranks, empirical_ranks, pch=16, col=util$c_dark_teal,
     xlab="True Rank", ylab="Empirical Rank")
lines(c(0, 500), c(0, 500), lwd=2, lty=2, col="#DDDDDD")

plot(true_ranks, post_mean_ranks, pch=16, col=util$c_dark,
     xlab="True Rank", ylab="Posterior Mean Rank")
lines(c(0, 500), c(0, 500), lwd=2, lty=2, col="#DDDDDD")
```

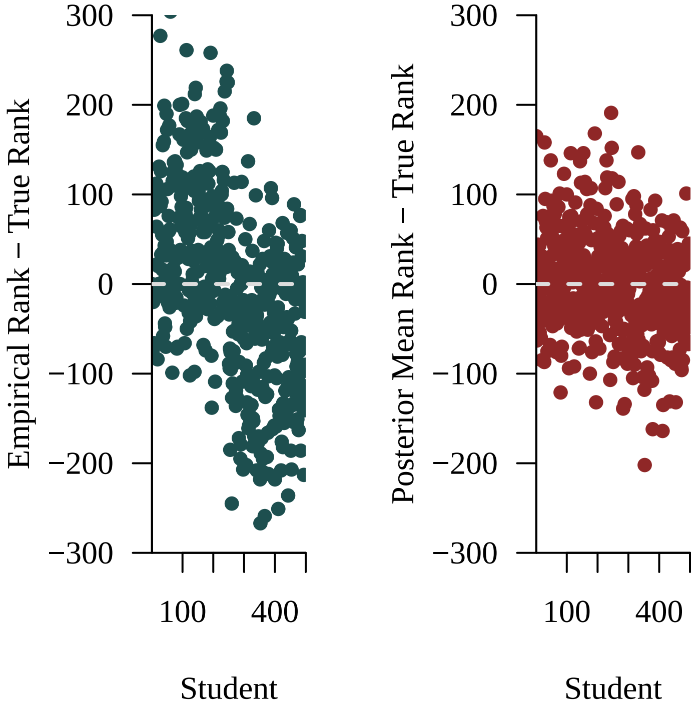


```
par(mfrow=c(1, 2), mar=c(5, 5, 1, 1))

plot(true_ranks, empirical_ranks - true_ranks,
     pch=16, col=util$c_dark_teal,
     xlab="True Rank", ylab="Empirical Rank - True Rank",
     ylim=c(-300, 300))
abline(h=0, lwd=2, lty=2, col="#DDDDDD")

plot(true_ranks, post_mean_ranks - true_ranks,
     pch=16, col=util$c_dark,
     xlab="True Rank", ylab="Posterior Mean Rank - True Rank",
     ylim=c(-300, 300))
abline(h=0, lwd=2, lty=2, col="#DDDDDD")
```



Unlike the empirical ranking the posterior mean ranking behaves reasonably well across the two student cohorts!

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

plot(1:data$N_students, empirical_ranks - true_ranks,
     pch=16, col=util$c_dark_teal,
     xlab="Student", ylab="Empirical Rank - True Rank",
     ylim=c(-300, 300))
abline(h=0, lwd=2, lty=2, col="#DDDDDD")

plot(1:data$N_students, post_mean_ranks - true_ranks,
     pch=16, col=util$c_dark,
     xlab="Student", ylab="Posterior Mean Rank - True Rank",
     ylim=c(-300, 300))
abline(h=0, lwd=2, lty=2, col="#DDDDDD")
```



6.4.4 Attempt 3

The ability to infer systematic shifts in student ability under heterogeneous testing circumstances is particularly useful for learning about the impact of pedagogical interventions. For example let's say that the two student cohorts in our data set are actually part of a designed experiment, with the first cohort receiving standard instruction and the second receiving some modified instruction that might influence their ability to answer test questions correctly.

If we don't have any additional information about the students within each cohort then we can model their abilities hierarchically and then quantify the impact of the modified instruction by the difference in those population behaviors.

In this case we can effectively identify the student abilities by fixing the location of the location configuration of the first cohort's population to zero instead of anchoring any single ability. We will still, however, need to anchor one of the question discrimination parameters.

To ensure as nice a posterior geometry as possible we'll start with a monolithic non-centered parameterization of the student abilities within each cohort.

```
fit <- stan(file="stan_programs/irt3.stan",
            data=data, seed=8438338,
            warmup=1000, iter=2024, refresh=0)
```

Beyond warning about some heavy-tailed behavior the diagnostics indicate some sporadic auto-correlations issues but nothing too problematic.

```r
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.
```

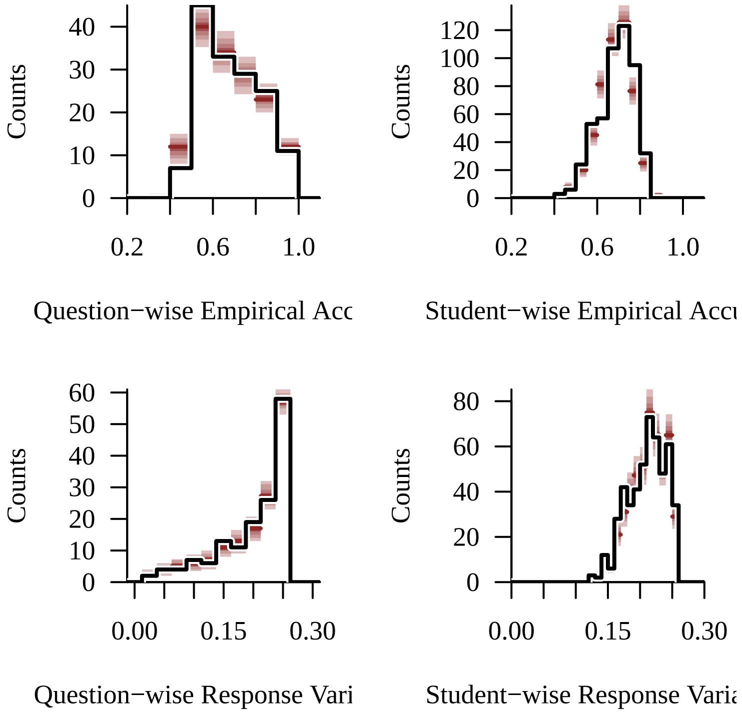
```r
samples <- util$extract_expectand_vals(fit)
base_samples <- util$filter_expectands(samples,
                                       c('gamma_free',
                                         'beta',
                                         'alpha_tilde',
                                         'delta_mu',
                                         'tau1', 'tau2'),
                                       check_arrays=TRUE)
util$check_all_expectand_diagnostics(base_samples)
```

```
beta[3]:
  Chain 4: Left tail hat{xi} (0.254) exceeds 0.25.

beta[11]:
  Chain 1: Left tail hat{xi} (0.495) exceeds 0.25.
  Chain 2: Left tail hat{xi} (0.535) exceeds 0.25.
  Chain 3: Both left and right tail hat{xi}s (0.333, 0.314) exceed 0.25.
  Chain 4: Left tail hat{xi} (0.443) exceeds 0.25.

beta[24]:
  Chain 1: Left tail hat{xi} (0.398) exceeds 0.25.
  Chain 2: Both left and right tail hat{xi}s (0.291, 0.553) exceed 0.25.
  Chain 3: Right tail hat{xi} (0.389) exceeds 0.25.
  Chain 4: Both left and right tail hat{xi}s (0.423, 0.369) exceed 0.25.
  Chain 3: hat{ESS} (78.135) is smaller than desired (100).

beta[42]:
  Chain 1: Left tail hat{xi} (0.325) exceeds 0.25.
  Chain 2: Left tail hat{xi} (0.381) exceeds 0.25.
  Chain 3: Left tail hat{xi} (0.320) exceeds 0.25.
  Chain 4: Left tail hat{xi} (0.358) exceeds 0.25.
```

```
beta[44]:
  Chain 1: Right tail hat{xi} (0.605) exceeds 0.25.
  Chain 2: Both left and right tail hat{xi}s (0.514, 0.288) exceed 0.25.
  Chain 3: Right tail hat{xi} (0.473) exceeds 0.25.
  Chain 4: Both left and right tail hat{xi}s (0.428, 0.389) exceed 0.25.
  Chain 1: hat{ESS} (92.255) is smaller than desired (100).

beta[47]:
  Chain 1: Left tail hat{xi} (0.302) exceeds 0.25.
  Chain 2: Left tail hat{xi} (0.287) exceeds 0.25.
  Chain 4: Left tail hat{xi} (0.255) exceeds 0.25.

beta[106]:
  Chain 2: Left tail hat{xi} (0.481) exceeds 0.25.
  Chain 4: Left tail hat{xi} (0.472) exceeds 0.25.

beta[112]:
  Chain 1: Left tail hat{xi} (0.389) exceeds 0.25.
  Chain 2: Left tail hat{xi} (0.350) exceeds 0.25.
  Chain 4: Left tail hat{xi} (0.303) exceeds 0.25.

beta[115]:
  Chain 3: Left tail hat{xi} (0.507) exceeds 0.25.
  Chain 3: hat{ESS} (49.282) is smaller than desired (100).

beta[118]:
  Chain 2: Left tail hat{xi} (0.348) exceeds 0.25.
  Chain 3: Left tail hat{xi} (0.355) exceeds 0.25.
  Chain 4: Left tail hat{xi} (0.524) exceeds 0.25.
  Chain 4: hat{ESS} (94.273) is smaller than desired (100).

beta[124]:
  Chain 1: Left tail hat{xi} (0.267) exceeds 0.25.
  Chain 2: Left tail hat{xi} (0.275) exceeds 0.25.
  Chain 4: Left tail hat{xi} (0.360) exceeds 0.25.

beta[125]:
  Chain 2: Left tail hat{xi} (0.255) exceeds 0.25.

beta[132]:
  Chain 2: Left tail hat{xi} (0.251) exceeds 0.25.

beta[134]:
  Chain 1: Left tail hat{xi} (0.257) exceeds 0.25.

beta[135]:
  Chain 3: Left tail hat{xi} (0.275) exceeds 0.25.

beta[137]:
  Chain 1: Left tail hat{xi} (0.423) exceeds 0.25.
  Chain 2: Left tail hat{xi} (0.397) exceeds 0.25.
  Chain 3: Left tail hat{xi} (0.399) exceeds 0.25.
  Chain 4: Left tail hat{xi} (0.424) exceeds 0.25.

beta[139]:
  Chain 2: Left tail hat{xi} (0.282) exceeds 0.25.

beta[140]:
  Chain 2: Left tail hat{xi} (0.288) exceeds 0.25.
  Chain 3: Left tail hat{xi} (0.285) exceeds 0.25.
  Chain 4: Left tail hat{xi} (0.299) exceeds 0.25.

beta[148]:
  Chain 4: Left tail hat{xi} (0.362) exceeds 0.25.
  Chain 4: hat{ESS} (89.583) is smaller than desired (100).

delta_mu:
  Chain 2: hat{ESS} (73.522) is smaller than desired (100).

tau1:
  Chain 2: hat{ESS} (75.766) is smaller than desired (100).


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

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

Overall the addition of the cohort hierarchies does not seem to have compromised the retrodictive performance.

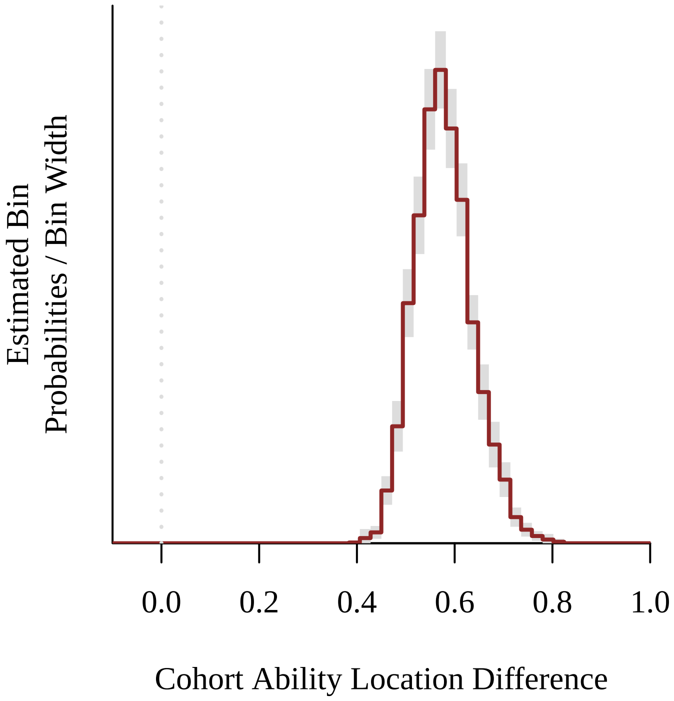
```r
par(mfrow=c(2, 2), mar=c(5, 5, 1, 1))

util$plot_hist_quantiles(samples, 'mean_q_pred', 0.2, 1.1, 0.1,
                         baseline_values=mean_q,
                         xlab="Question-wise Empirical Accuracy")

util$plot_hist_quantiles(samples, 'mean_s_pred', 0.2, 1.1, 0.05,
                         baseline_values=mean_s,
                         xlab="Student-wise Empirical Accuracy")

util$plot_hist_quantiles(samples, 'var_q_pred', 0, 0.3, 0.025,
                         baseline_values=var_q,
                         xlab="Question-wise Response Variances")

util$plot_hist_quantiles(samples, 'var_s_pred', 0, 0.3, 0.01,
                         baseline_values=var_s,
                         xlab="Student-wise Response Variances")
```



With an adequate model we can see examine our posterior inferences to see just how impactful the modified instruction is. The posterior behavior for `delta_mu` concentrates on large values suggesting quite a strong effect.

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

title="Cohort Ability Location Difference"
```

```
util$plot_expectand_pushforward(samples[['delta_mu']],
                                50, flim=c(-0.1, 1),
                                display_name=title)
abline(v=0, lwd=2, lty=3, col="#DDDDDD")
```



In fact there is vanishingly little posterior probability over any negative effects at all.

```
p_est <- util$implicit_subset_prob(samples,
                                   function(x) x > 0,
                                   list('x' = 'delta_mu'))

format_string <- paste0("Posterior probability that delta_mu > 0 ",
                        "= %.3f +/- %.3f.")
cat(sprintf(format_string, p_est[1], 2 * p_est[2]))
```

```
Posterior probability that delta_mu > 0 = 1.000 +/- 0.000.
```

6.5 Modeling a Gaming League

For our final example we'll analyze the outcome of games in an organized league.

Each game consists of two teams competing to score points, with the winner of each game determined by the team with the most points. If the two teams have the same score at the end of the game then the game ends in a tie. The games are not played at neutral locations; instead an *away* team travels to the venue of the opposing *home* team.

The league itself is comprised of ten teams. During the season each team plays each other in a repeated round robin format, with every possible matchup taking place exactly 8 times. In total the season consists of 360 total games, with 5 games played per week across 72 total weeks.

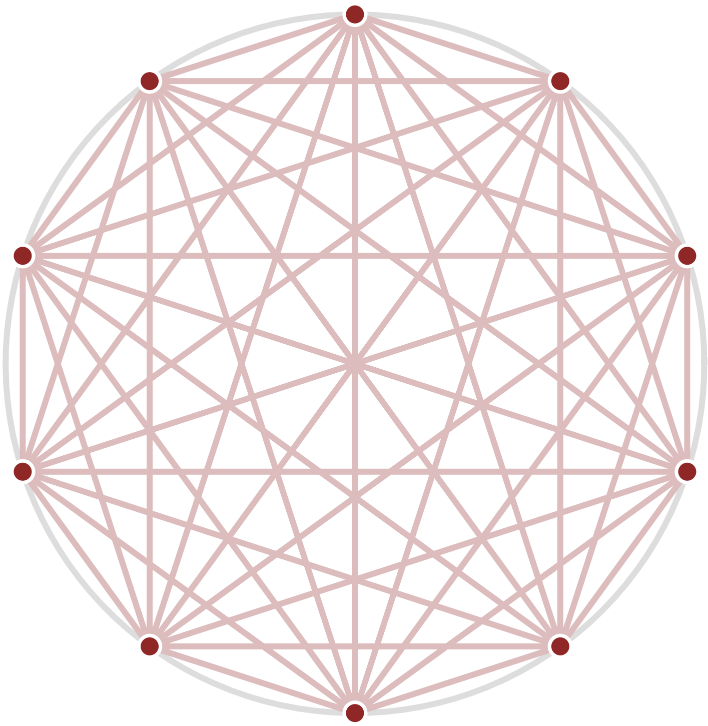
6.5.1 Explore data

Before attempting our first model let's take a look at the available data.

```
data <- read_rdump('data/season.data.R')
```

The repeated round robin format ensures a balanced schedule, with every team playing each other team the same number of times.

```
par(mfrow=c(1, 1), mar=c(0, 0, 0, 0))

adj <- build_adj_matrix(data$N_teams, data$N_games,
                        data$home_idx, data$away_idx)
plot_undir_graph(adj)
```



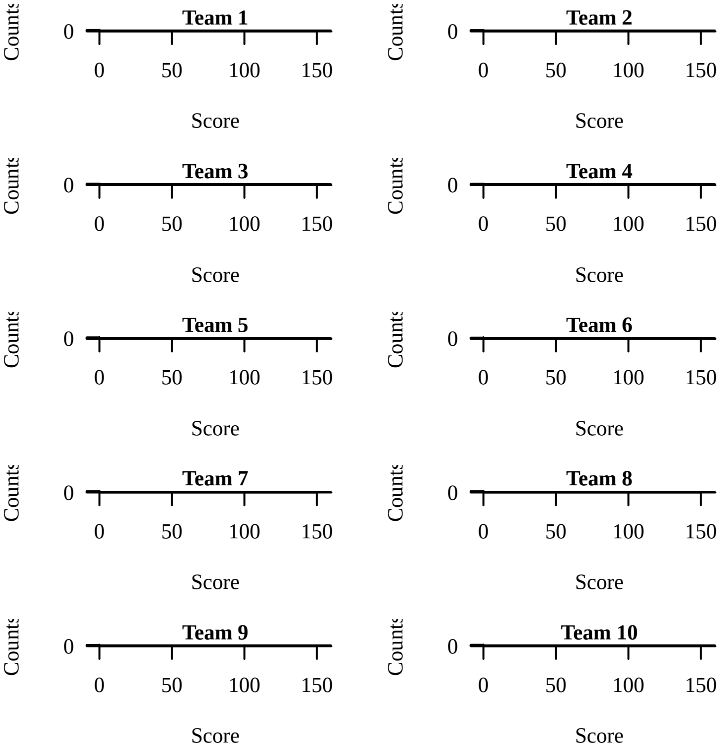
In particular there are no disconnected components in the resulting connectivity graph.

```
compute_connected_components(adj)
```

```
[[1]]
 [1]  1  2  3  4  5  6  7  8  9 10
```

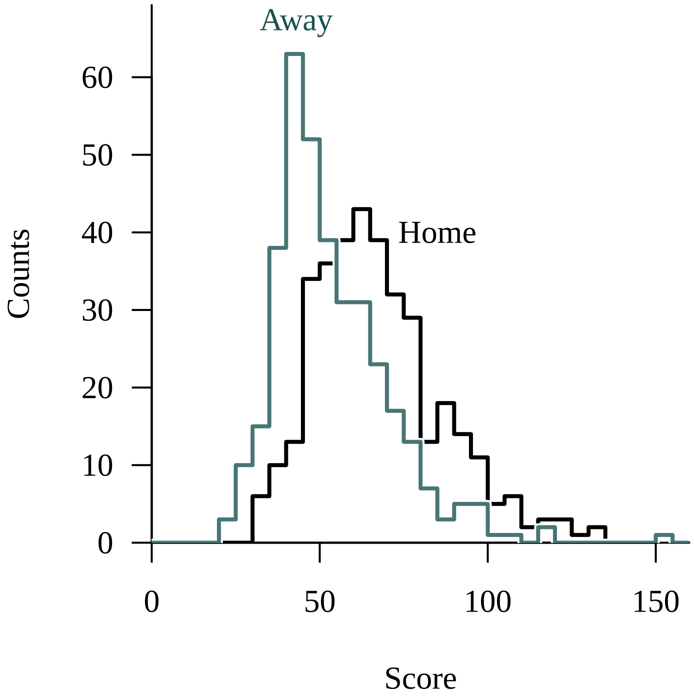
While the schedule is uniform the team performance is not. Some teams consistently score more points than others, and some teams exhibit more variation in the scoring than others.

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

for (t in 1:data$N_teams) {
  y_team <- c(data$y_home[data$home_idx == t],
              data$y_away[data$away_idx == t])
  util$plot_line_hist(y_team, 0, 160, 10,
                      xlab="Score", main=paste("Team", t))
}
```



Overall the home teams appear to score more points than the away teams.

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

util$plot_line_hists(data$y_home, data$y_away,
                     0, 160, 5, xlab="Score")
text(85, 40, "Home", col="black")
text(43, 67, "Away", col=util$c_dark_teal)
```

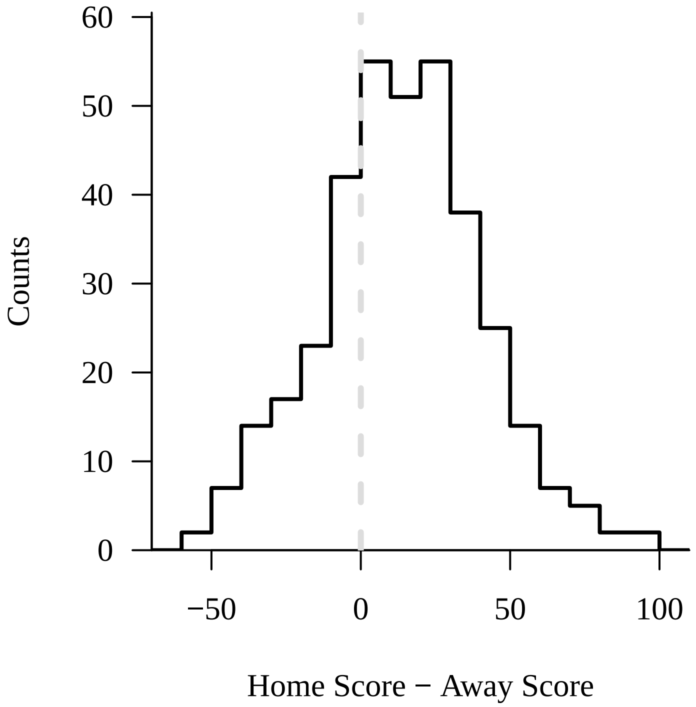


This results in an empirical distribution of score differentials that is slightly biased to positive values. The score differential is a particularly useful summary of a game because it immediately communicates the winner – positive values indicate a home team win, negative values an away team win, and zero values a tie.

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

util$plot_line_hist(data$y_home - data$y_away,
                    -70, 110, 10,
                    xlab="Home Score - Away Score")
```

```
Warning in check_bin_containment(bin_min, bin_max, values): 1 value (0.3%) fell
below the binning.
```

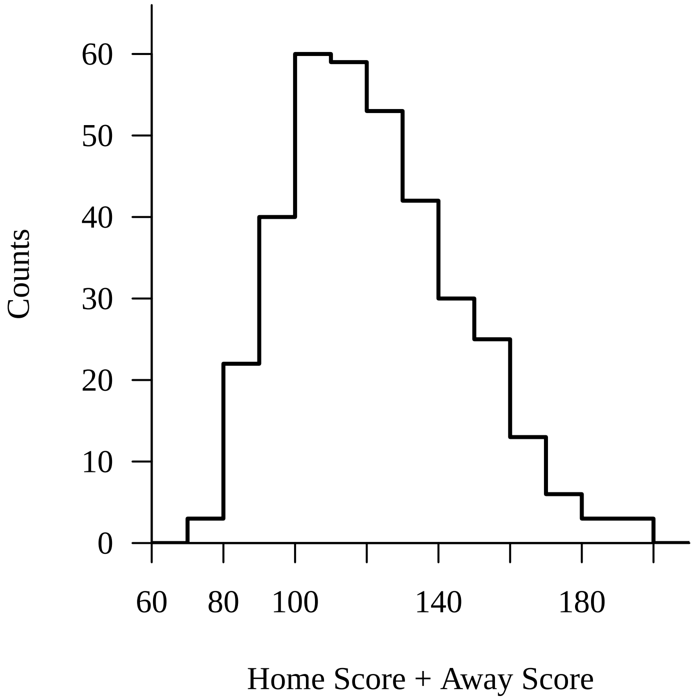
```
abline(v=0, lty=2, lwd=3, col="#DDDDDD")
```



Another useful summary is the total score which captures for example whether offensive or defensive play dictated each game.

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

util$plot_line_hist(data$y_home + data$y_away,
                    60, 210, 10,
                    xlab="Home Score + Away Score")
```

```
Warning in check_bin_containment(bin_min, bin_max, values): 1 value (0.3%) fell
above the binning.
```



6.5.2 Attempt 1

At this point is should come as no surprise that we're going to reach for a pairwise comparison modeling techniques to model these games. The key question, however, is exactly what comparison we want to model.

We could model the outcome of each game, but that approach complicated by the presence of ties and ignores a most of the available data. We can build a richer picture of each game by instead modeling the score of each team, treating them as the outcome of a pairwise comparison between that team's offensive skill and their opponent's defensive skill.

Because the scores are non-negative integers a natural place to start is a Poisson model,

$$\text{Poisson}(y_{n,i_1 i_2} \mid \lambda_{i_1 i_2}).$$

To start let's couple $\lambda_{i_1 i_2}$ to the difference in offensive and defensive skills with an exponential baseline function,

$$\begin{aligned}\lambda_{i_1 i_2} &= \exp\left(\alpha + \beta^{\text{off}}_{i_1} - \beta^{\text{def}}_{i_2}\right)\\ &= \exp\left(\alpha\right)\,\exp\left(+\beta^{\text{off}}_{i_1}\right)\,\exp\left(-\beta^{\text{def}}_{i_2}\right).\end{aligned}$$

Note that this is a bipartite pairwise comparison model – we can compare any team's offense to any team's defense, but we cannot compare offensives or defences to each other, In order to ensure that each parameter can be informed by the observed game outcomes we need to anchor one of the offensive skills and one of the defensive skills to zero,

$$\begin{aligned} \delta_i^{\text{off}} &= \beta_i^{\text{off}} - \beta_{i'}^{\text{off}} \\ \delta_i^{\text{def}} &= \beta_i^{\text{def}} - \beta_{i'}^{\text{def}}, \end{aligned}$$

yielding the model

$$\lambda_{i_1 i_2} = \exp\left(\alpha\right) \, \exp\left(+ \delta_{i_1}^{\text{off}} \right) \, \exp\left(- \delta_{i_2}^{\text{def}} \right).$$

Here we will anchor the offensive and defensive skills of the first team to zero.

We can then interpret the baseline α as the average score if the anchored offensive played the anchored defense. Because we anchored the offensive and defensive skills of the same team this is a purely hypothetical matchup, but we can exclude unreasonable behaviors. For demonstration purposes let's say that our domain expertise is consistent with

$$25 \lessapprox \exp\left(\alpha\right) \lessapprox 75$$

or equivalently

$$3.77 - 0.55 \lessapprox \alpha \lessapprox 3.77 + 0.55.$$

The relative skill parameters δ_i^{off} and δ_i^{off} then determine if the other teams' offense and defense are better than those of the anchor team. Here we'll take a pretty conservative prior threshold

$$0.1 \lessapprox \exp\left(\delta_i^{\text{off/def}}\right) \lessapprox 7.4$$

or

$$-2 \lessapprox \delta_i^{\text{off/def}} \lessapprox 2.$$

```r
fit <- stan(file="stan_programs/season1.stan",
            data=data, seed=8438338,
            warmup=1000, iter=2024, refresh=0)
```

One of the Markov chains exhibits large auto-correlations in the baseline α, but the lack of auto-correlation in the other Markov chains, let alone any other diagnostics warnings, suggests that we should have be able to inform accurate Markov chain Monte Carlo estimators.

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

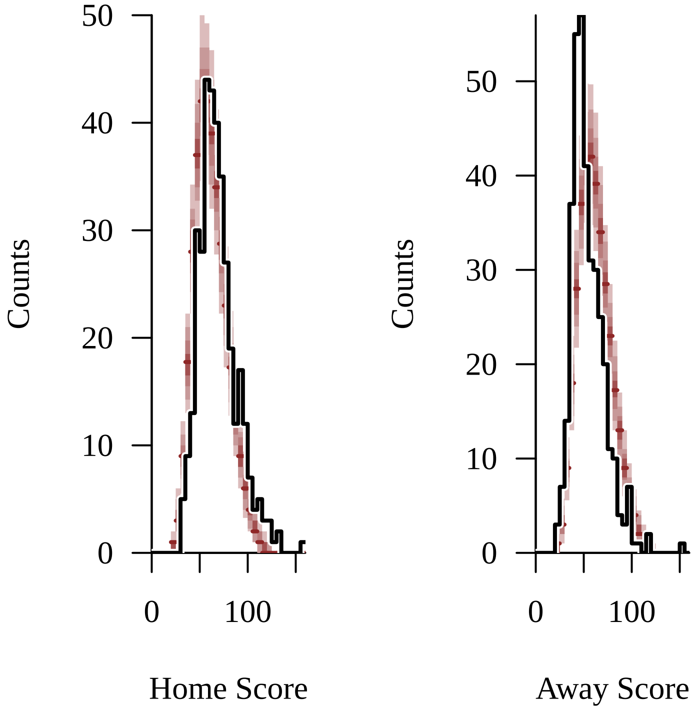
```
  All Hamiltonian Monte Carlo diagnostics are consistent with reliable
Markov chain Monte Carlo.
```

```
samples <- util$extract_expectand_vals(fit)
base_samples <- util$filter_expectands(samples,
                                       c('alpha',
                                         'delta_off_free',
                                         'delta_def_free'),
                                       check_arrays=TRUE)
util$check_all_expectand_diagnostics(base_samples)
```

```
alpha:
  Chain 4: hat{ESS} (88.393) is smaller than desired (100).


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

Unfortunately the posterior retrodictive performance suggests that this model is not sophisticated enough to capture the details of the observed data. Specifically both the observed home scores and away scores exhibit a skewness, towards larger values in the first case and smaller values in the second case, that the posterior predictive distribution cannot reproduce.

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

util$plot_hist_quantiles(samples, 'y_home_pred', 0, 160, 5,
                         baseline_values=data$y_home,
                         xlab="Home Score")

util$plot_hist_quantiles(samples, 'y_away_pred', 0, 160, 5,
                         baseline_values=data$y_away,
                         xlab="Away Score")
```

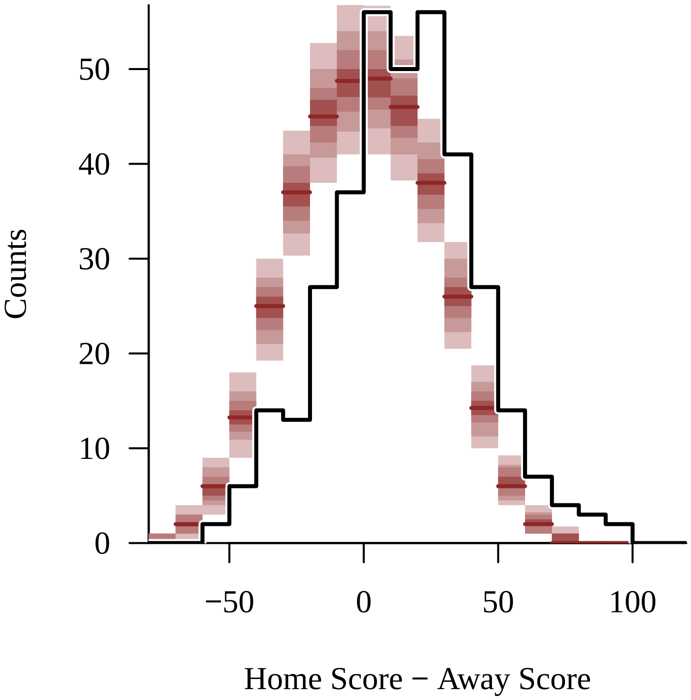


Even more strikingly the observed score differentials are biased towards much larger values relative that the posterior predictive score differentials.

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

util$plot_hist_quantiles(samples, 'y_diff_pred', -80, 120, 10,
                         baseline_values=data$y_home - data$y_away,
                         xlab="Home Score - Away Score")
```

```
Warning in check_bin_containment(bin_min, bin_max, collapsed_values,
"predictive value"): 283 predictive values (0.0%) fell below the binning.
```

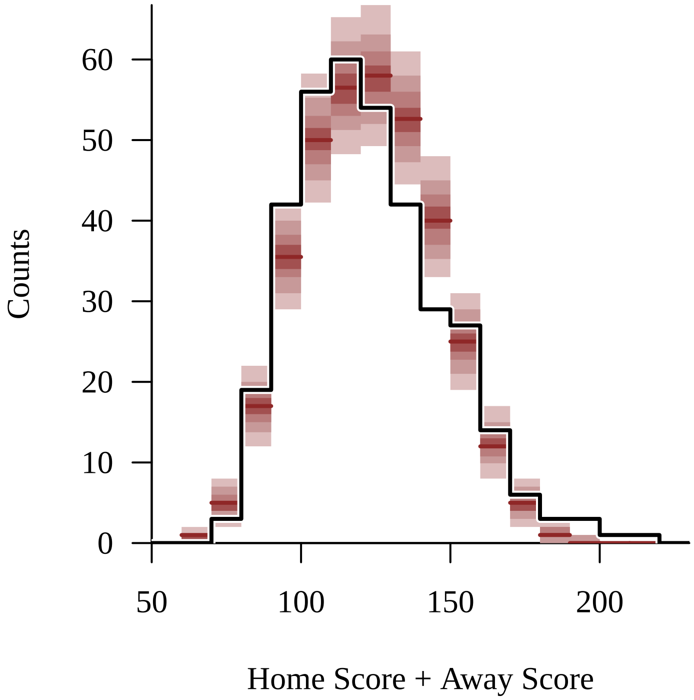
```
Warning in check_bin_containment(bin_min, bin_max, baseline_values, "observed
value"): 1 observed value (0.3%) fell below the binning.
```



That said the observed and posterior predictive total scores are consistent with each other.

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

util$plot_hist_quantiles(samples, 'y_sum_pred', 50, 230, 10,
                         baseline_values=data$y_home + data$y_away,
                         xlab="Home Score + Away Score")
```

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



Overall it looks like the data exhibits a home-field advantage that our model cannot accommodate.

6.5.3 Attempt 2

Fortunately a home-field advantage is straightforward to incorporate into our model by offsetting the skill differences. Specifically for we expand our model for the home team score to

$$\lambda_{i_1 i_2} = \exp\left(\alpha + \eta^{\text{off}}\right) \exp\left(+ \delta^{\text{off}}_{i_1}\right) \exp\left(- \delta^{\text{def}}_{i_2}\right)$$

and for the away team score to

$$\lambda_{i_2 i_1} = \exp\left(\alpha - \eta^{\text{def}}\right) \exp\left(+ \delta^{\text{off}}_{i_2}\right) \exp\left(- \delta^{\text{def}}_{i_1}\right).$$

The negative sign on η^{def} is somewhat arbitrary, but it ensures that for both scores a positive η corresponds to a home-field advantage.

Let's say that our available domain expertise constrains the home-field advantage to relatively small proportional changes,

$$0.78 \lessapprox \exp\left(\eta^{\text{off/def}}\right) \lessapprox 1.28$$

or

$$-0.25 \lessapprox \eta^{\text{off/def}} \lessapprox +0.25.$$

```
fit <- stan(file="stan_programs/season2.stan",
            data=data, seed=8438338,
            warmup=1000, iter=2024, refresh=0)
```

Conveniently the diagnostics are now completely 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.
```

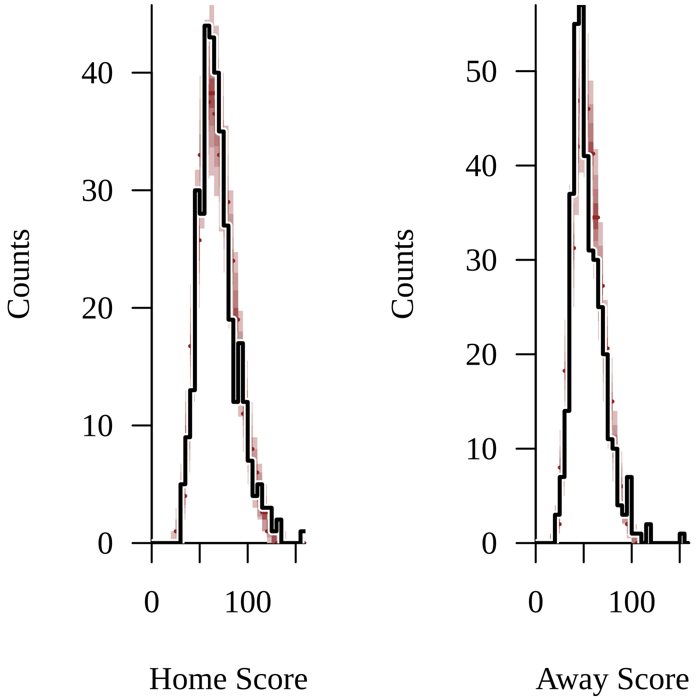
```
samples <- util$extract_expectand_vals(fit)
base_samples <- util$filter_expectands(samples,
                                       c('alpha',
                                         'eta_off',
                                         'eta_def',
                                         'delta_off_free',
                                         'delta_def_free'),
                                       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.
```

Overall the posterior predictive performance has improved, but there's still a noticeable disagreement in the shape of the summary statistics.

For example while the observed and posterior predictive home scores now agree the separated away scores still exhibit a skew towards smaller values that the posterior predictive distribution does not match.

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

util$plot_hist_quantiles(samples, 'y_home_pred', 0, 160, 5,
                         baseline_values=data$y_home,
                         xlab="Home Score")

util$plot_hist_quantiles(samples, 'y_away_pred', 0, 160, 5,
                         baseline_values=data$y_away,
                         xlab="Away Score")
```

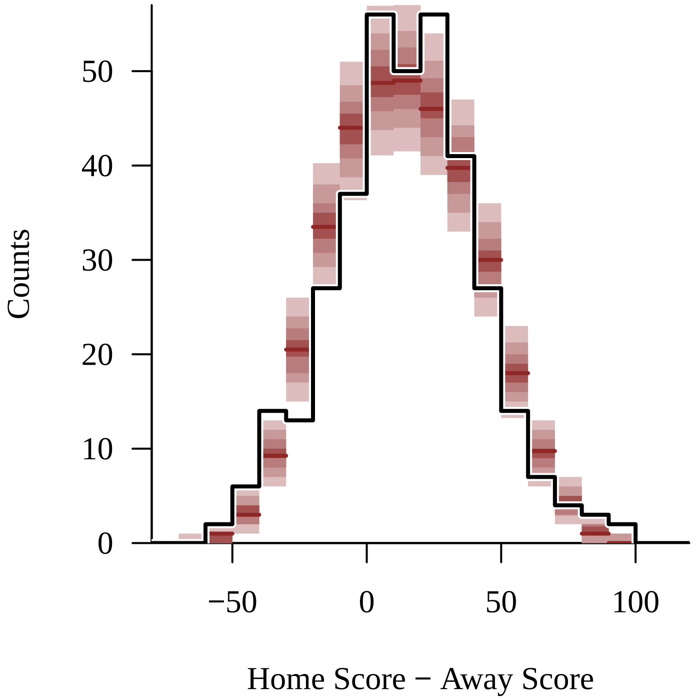


The observed and posterior predictive score differentials now align, but the observed behaviors are more over-dispersed than the posterior predictive behaviors.

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

util$plot_hist_quantiles(samples, 'y_diff_pred', -80, 120, 10,
                         baseline_values=data$y_home - data$y_away,
                         xlab="Home Score - Away Score")
```

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

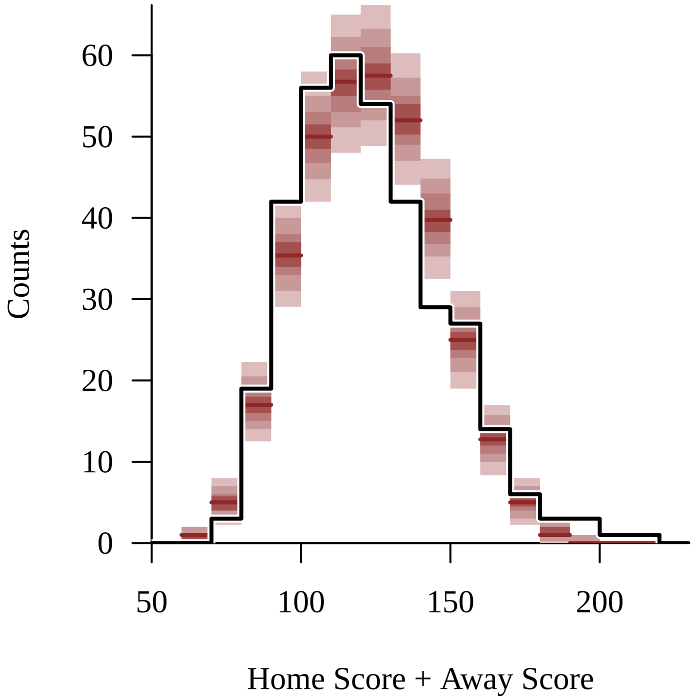
```
Warning in check_bin_containment(bin_min, bin_max, baseline_values, "observed
value"): 1 observed value (0.3%) fell below the binning.
```



We see a similar observed over-dispersion in the total scores, with a skew towards smaller values in the peak and a heavier tail towards larger values than what we see in the posterior predictive behavior.

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

util$plot_hist_quantiles(samples, 'y_sum_pred', 50, 230, 10,
                         baseline_values=data$y_home + data$y_away,
                         xlab="Home Score + Away Score")
```

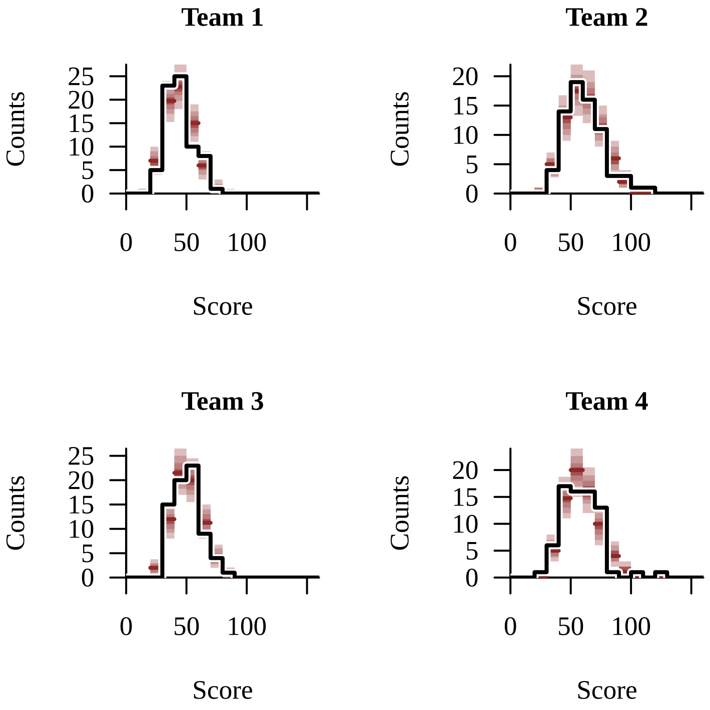
```
Warning in check_bin_containment(bin_min, bin_max, collapsed_values,
"predictive value"): 5 predictive values (0.0%) fell below the binning.
```

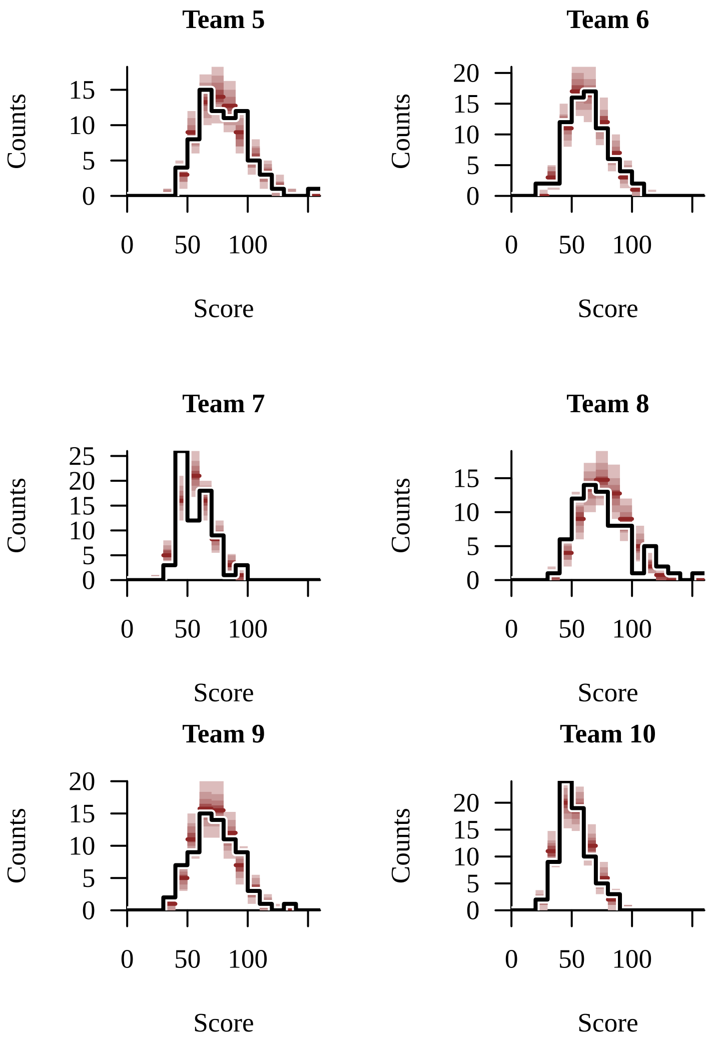


Given these patterns of disagreement we might be tempted to consider a more flexible observational model, in particular expanding the Poisson model into a negative binomial model. If we can identify and then model a more mechanistic source of the excess variation, however, then we would be rewarded with more precise and generalizeable inferences. To that end let's see if we can find heterogeneity that manifests in the observed behaviors but not the posterior predictive behaviors.

For example there might be variation in the team performances beyond what we have already incorporated into the current model. Examining the observed and posterior predictive scores for each team, however, doesn't reveal any obvious discrepancy.

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

for (t in 1:data$N_teams) {
  team_scores_obs <- c(data$y_home[data$home_idx == t],
                       data$y_away[data$away_idx == t])

  team_scores <- c(lapply(which(data$home_idx == t),
                          function(n)
                          samples[[paste0('y_home_pred[', n, ']')]]),
                   lapply(which(data$away_idx == t),
                          function(n)
                          samples[[paste0('y_away_pred[', n, ']')]]))
  names(team_scores) <- sapply(1:data$N_weeks,
                               function(n) paste0('y[', n, ']'))

  util$plot_hist_quantiles(team_scores, 'y', 0, 160, 10,
                           baseline_values=team_scores_obs,
                           xlab="Score", main=paste("Team", t))
}
```





Another possible source of heterogeneity is systematic trends across time. To investigate that we can survey the behavior of the score differential summary statistic early in the season, in the middle of the season, and towards the end of season.

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

names <- lapply(1:120, function(n) paste0('y_diff_pred[', n, ']'))
filt_samples <- lapply(names, function(n) samples[[n]])
names(filt_samples) <- names

util$plot_hist_quantiles(filt_samples, 'y_diff_pred', -80, 120, 10,
                         baseline_values=data$y_home[1:120] -
                                         data$y_away[1:120],
                         xlab="Home Score - Away Score",
                         main="First Third of Season")
```

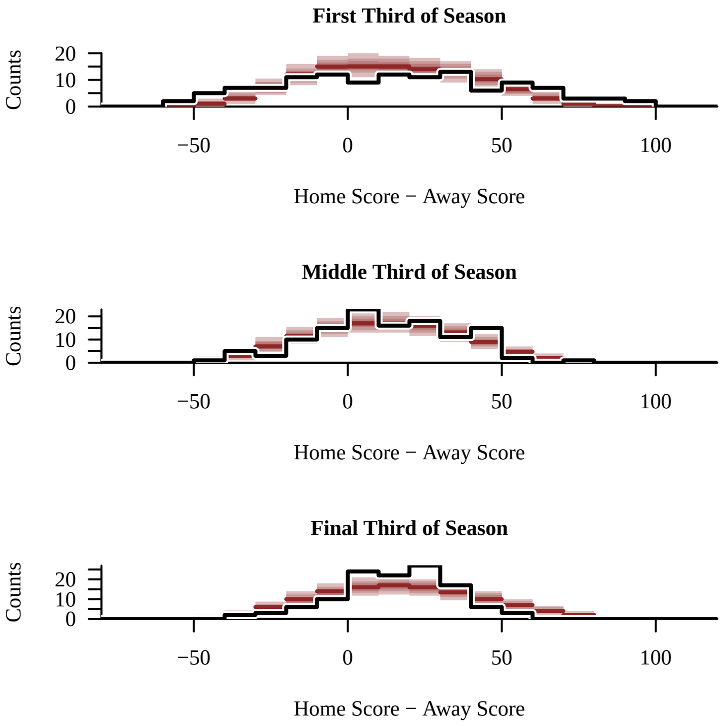
```
Warning in check_bin_containment(bin_min, bin_max, collapsed_values,
"predictive value"): 1 predictive value (0.0%) fell below the binning.
```

```
Warning in check_bin_containment(bin_min, bin_max, baseline_values, "observed
value"): 1 observed value (0.8%) fell below the binning.
```

```
names <- lapply(121:240, function(n) paste0('y_diff_pred[', n, ']'))
filt_samples <- lapply(names, function(n) samples[[n]])
names(filt_samples) <- names

util$plot_hist_quantiles(filt_samples, 'y_diff_pred', -80, 120, 10,
                         baseline_values=data$y_home[121:240] -
                                         data$y_away[121:240],
                         xlab="Home Score - Away Score",
                         main="Middle Third of Season")
```

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

```
names <- lapply(241:360, function(n) paste0('y_diff_pred[', n, ']'))
filt_samples <- lapply(names, function(n) samples[[n]])
names(filt_samples) <- names

util$plot_hist_quantiles(filt_samples, 'y_diff_pred', -80, 120, 10,
                         baseline_values=data$y_home[241:360] -
                                         data$y_away[241:360],
                         xlab="Home Score - Away Score",
                         main="Final Third of Season")
```



Here we see a clear discrepancy between the observed and posterior predictive behaviors. Because our model is static across time the posterior predictive behavior is the same for all three time intervals, but the observed behavior gradually becomes less and less variable as the season progresses.

6.5.4 Attempt 3

The main source of systematic game score variation in our model are the heterogeneous team skills. One reason why we might see decreasing variation in game scores across time is a weakening influence of those skills. For example after a long season of persistent games the players on each time might become more and more fatigued, reducing their impact on the later games.

We can incorporate this hypothesis into our model by adding a discrimination variable that evolves with time,

$$\begin{aligned}\lambda_{i_1 i_2} &= \exp\left(\alpha + \eta^{\text{off}} + \gamma(t) \cdot (\delta^{\text{off}}_{i_1} - \delta^{\text{def}}_{i_2})\right)\\ \lambda_{i_2 i_1} &= \exp\left(\alpha - \eta^{\text{def}} + \gamma(t) \cdot (\delta^{\text{off}}_{i_2} - \delta^{\text{def}}_{i_1})\right).\end{aligned}$$

Because we expect fatigue to only increase as the season progresses the outputs $\gamma(t)$ should decay with time. Here let's assume the functional form

$$\gamma(t) = \gamma_0^{t/1\,\text{week}}.$$

Finally we'll use a prior model that constrains γ_0 to

$$0.8 \lessapprox \gamma_0 \lessapprox 1.0$$

so that by the effective discrimination by the end of season is contained to

$$10^{-7} \lessapprox \gamma_0^{72} \lessapprox 1.$$

```
fit <- stan(file="stan_programs/season3.stan",
            data=data, seed=8438338,
            warmup=1000, iter=2024, refresh=0)
```

A few of the marginal auto-correlation are a bit low, but nothing that should compromise our posterior computation.

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

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

```
samples <- util$extract_expectand_vals(fit)
base_samples <- util$filter_expectands(samples,
                                       c('alpha',
                                         'eta_off',
                                         'eta_def',
                                         'gamma0',
                                         'delta_off_free',
                                         'delta_def_free'),
                                       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.
```

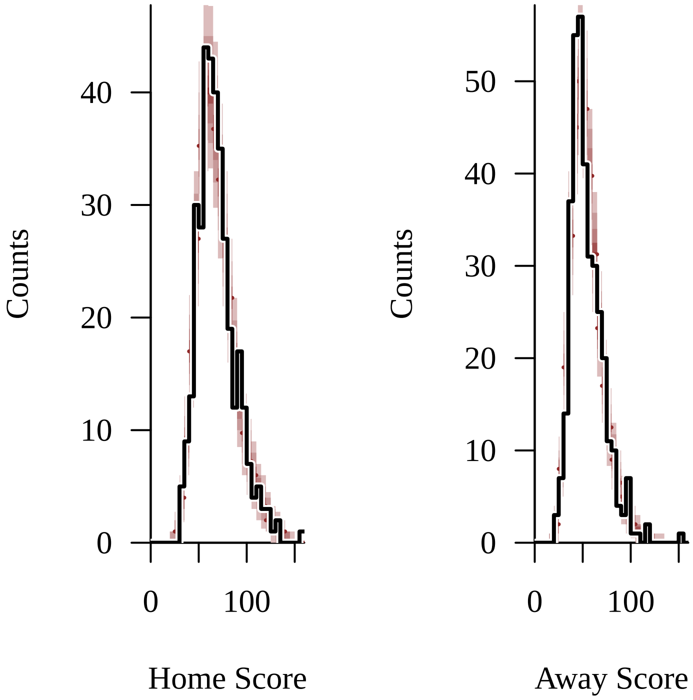
It looks like this model expansion has done the trick. Specifically the posterior predictive behaviors now exhibit the same skewness as the observed behaviors.

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

util$plot_hist_quantiles(samples, 'y_home_pred', 0, 160, 5,
                         baseline_values=data$y_home,
                         xlab="Home Score")
```

```
Warning in check_bin_containment(bin_min, bin_max, collapsed_values,
"predictive value"): 63 predictive values (0.0%) fell above the binning.
```

```
util$plot_hist_quantiles(samples, 'y_away_pred', 0, 160, 5,
                         baseline_values=data$y_away,
                         xlab="Away Score")
```

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



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

util$plot_hist_quantiles(samples, 'y_diff_pred', -80, 120, 10,
                         baseline_values=data$y_home - data$y_away,
                         xlab="Home Score - Away Score")
```

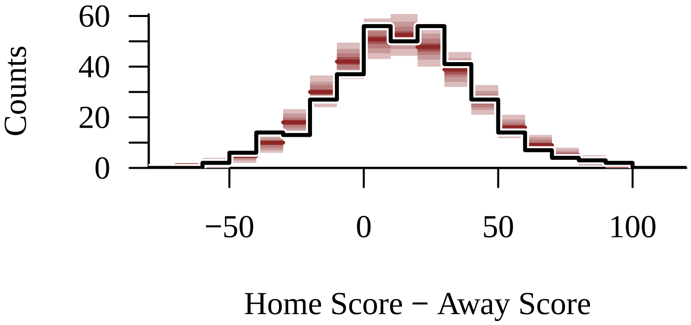
```
Warning in check_bin_containment(bin_min, bin_max, collapsed_values,
"predictive value"): 432 predictive values (0.0%) fell below the binning.
```

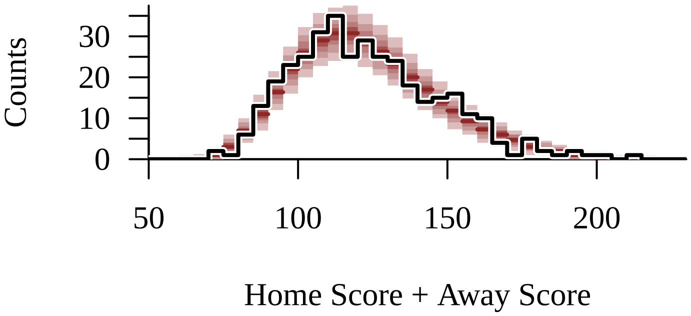
```
Warning in check_bin_containment(bin_min, bin_max, collapsed_values,
"predictive value"): 37 predictive values (0.0%) fell above the binning.
```

```
Warning in check_bin_containment(bin_min, bin_max, baseline_values, "observed
value"): 1 observed value (0.3%) fell below the binning.
```

```
util$plot_hist_quantiles(samples, 'y_sum_pred', 50, 230, 5,
                         baseline_values=data$y_home + data$y_away,
                         xlab="Home Score + Away Score")
```

```
Warning in check_bin_containment(bin_min, bin_max, collapsed_values,
"predictive value"): 115 predictive values (0.0%) fell above the binning.
```





The retrodictive agreement also persists across the season.

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

names <- sapply(1:120, function(n) paste0('y_diff_pred[', n, ']'))
filt_samples <- lapply(names, function(n) samples[[n]])
names(filt_samples) <- names
util$plot_hist_quantiles(filt_samples, 'y_diff_pred', -80, 120, 10,
                         baseline_values=data$y_home[1:120] -
                                         data$y_away[1:120],
                         xlab="Home Score - Away Score",
                         main="First Third of Season")
```

```
Warning in check_bin_containment(bin_min, bin_max, collapsed_values,
"predictive value"): 431 predictive values (0.1%) fell below the binning.
```

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

```
Warning in check_bin_containment(bin_min, bin_max, baseline_values, "observed
value"): 1 observed value (0.8%) fell below the binning.
```

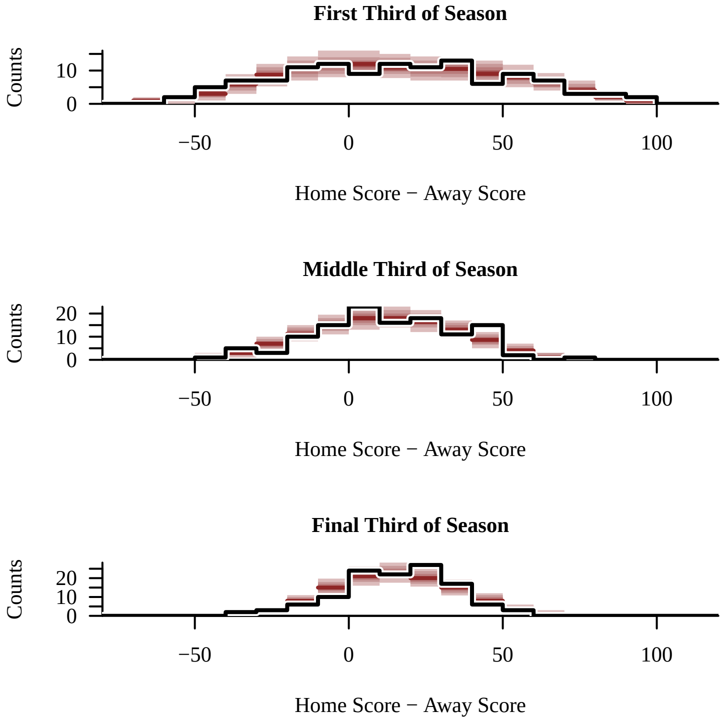
```r
names <- lapply(121:240, function(n) paste0('y_diff_pred[', n, ']'))
filt_samples <- lapply(names, function(n) samples[[n]])
names(filt_samples) <- names

util$plot_hist_quantiles(filt_samples, 'y_diff_pred', -80, 120, 10,
                         baseline_values=data$y_home[121:240] -
                                         data$y_away[121:240],
                         xlab="Home Score - Away Score",
                         main="Middle Third of Season")
```

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

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

```r
names <- lapply(241:360, function(n) paste0('y_diff_pred[', n, ']'))
filt_samples <- lapply(names, function(n) samples[[n]])
names(filt_samples) <- names
```

```
util$plot_hist_quantiles(filt_samples, 'y_diff_pred', -80, 120, 10,
                         baseline_values=data$y_home[241:360] -
                                         data$y_away[241:360],
                         xlab="Home Score - Away Score",
                         main="Final Third of Season")
```


First Third of Season

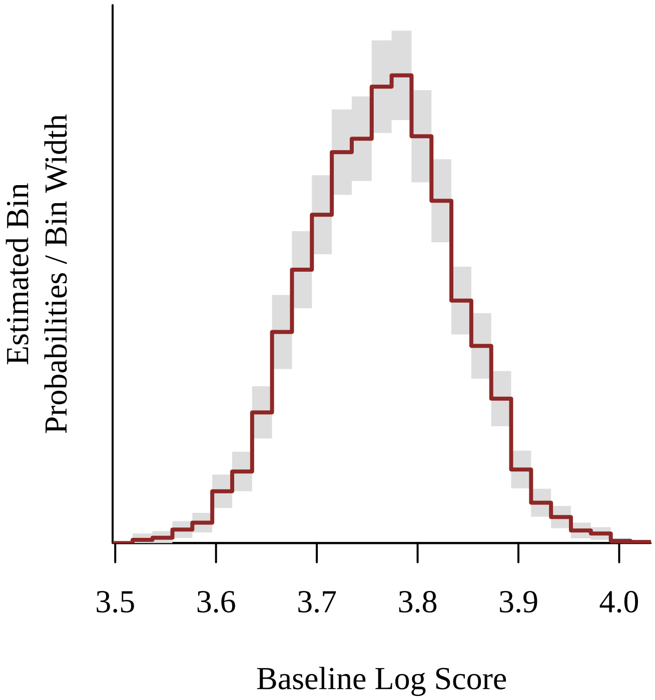
Middle Third of Season

Final Third of Season

Because our model appears to be adequate, at least in the context of these particular retrodictive checks, we can examine the resulting posterior inferences to see what we've learned about the teams.

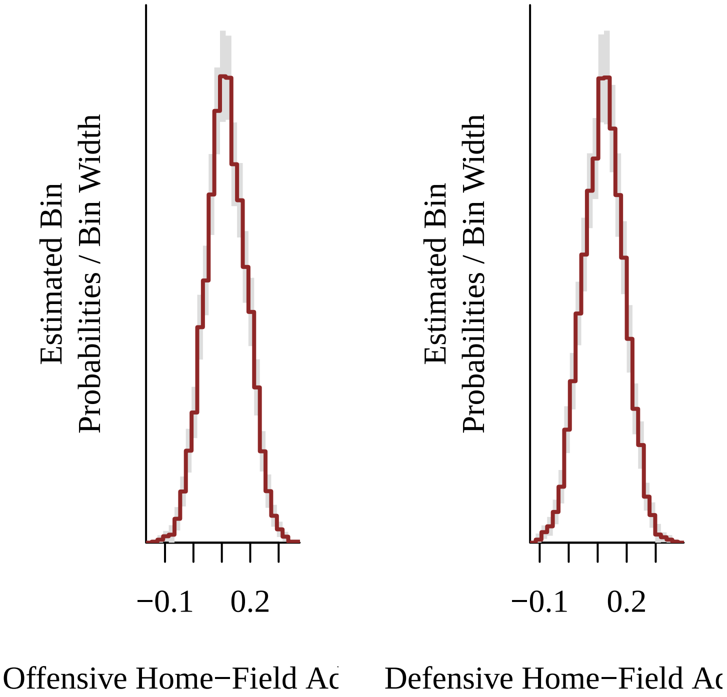
Firstly we have the baseline behavior which appears to be reasonably well-informed by the observed games.

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

util$plot_expectand_pushforward(samples[['alpha']], 25,
                                baseline_col=util$c_dark_teal,
                                display_name="Baseline Log Score")
```



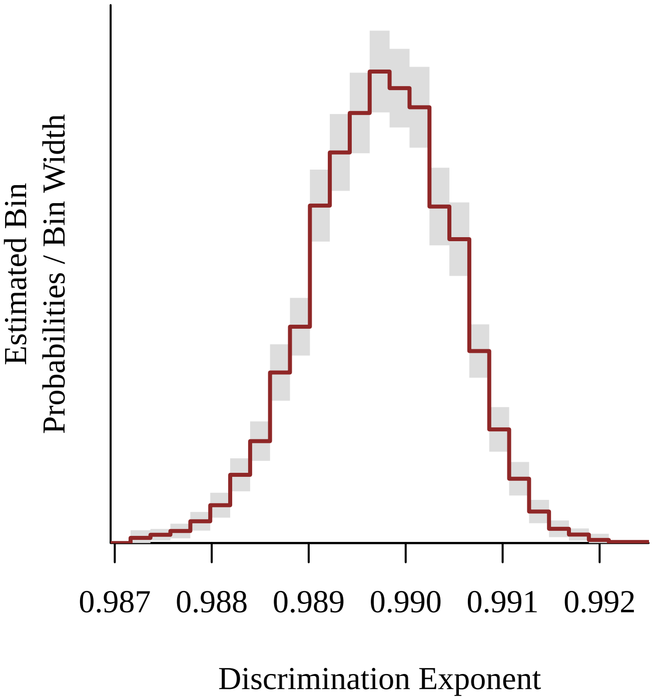
Our inferences preclude negative home-field advantages in favor of moderately positive values.

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

title <- "Offensive Home-Field Advantage"
util$plot_expectand_pushforward(samples[['eta_off']], 25,
                                baseline_col=util$c_dark_teal,
                                display_name=title)

title <- "Defensive Home-Field Advantage"
util$plot_expectand_pushforward(samples[['eta_def']], 25,
                                baseline_col=util$c_dark_teal,
                                display_name=title)
```



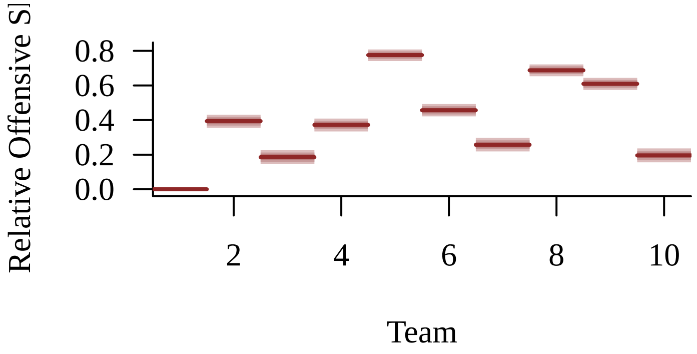
The baseline discrimination is close to zero, but small enough that by the end of the season the scores are substantially less sensitive to the team skills.

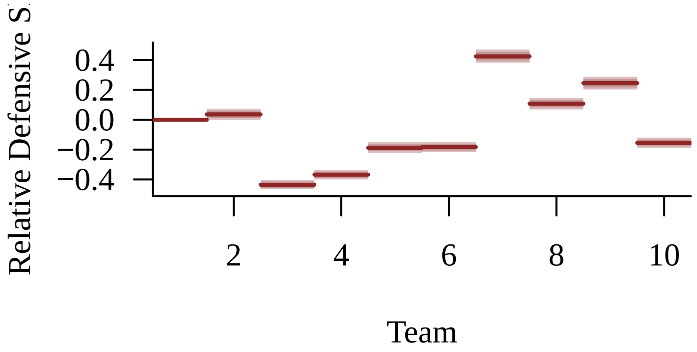
```
par(mfrow=c(1, 1), mar=c(5, 5, 1, 1))

util$plot_expectand_pushforward(samples[['gamma0']], 25,
                                baseline_col=util$c_dark_teal,
                                display_name="Discrimination Exponent")
```



Speaking of team skills, the inferred offensive and defensive skills allow us to compare the performance of each team to each other.

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

names <- sapply(1:data$N_teams,
                function(i) paste0('delta_off[', i, ']'))
util$plot_disc_pushforward_quantiles(samples, names,
                                     xlab="Team",
                                     ylab="Relative Offensive Skill")

names <- sapply(1:data$N_teams,
                function(i) paste0('delta_def[', i, ']'))
util$plot_disc_pushforward_quantiles(samples, names,
                                     xlab="Team",
                                     ylab="Relative Defensive Skill")
```





For example we can see that Team 5 features a strong offense but a mediocre defense. On the other hand Team 7 enjoys the strongest defense but a subpar offensive. The offensive skill of the anchor team is worse than that of all of the other teams, but their defensive skill is in the middle of the pack.

6.5.5 Informing Betting Decisions

Another use of these posterior inferences is to predict the outcomes of games that have not yet been played. Even more we can use those predictions to inform decisions about those future games, such as how to best gamble on them.

Let's say that immediately after the season ends a playoff beings with the four teams at the top of the standings playing each other in the first round. Specifically the fourth place team travels to play the first place team, and the third place team travels to play the second place team.

```
wins <- rep(0, data$N_teams)
ties <- rep(0, data$N_teams)
losses <- rep(0, data$N_teams)

for (n in 1:data$N_games) {
  if (data$y_home[n] > data$y_away[n]) {
    wins[data$home_idx[n]] <- wins[data$home_idx[n]] + 1
    losses[data$away_idx[n]] <- losses[data$away_idx[n]] + 1
  } else if (data$y_home[n] < data$y_away[n]) {
    losses[data$home_idx[n]] <- losses[data$home_idx[n]] + 1
    wins[data$away_idx[n]] <- wins[data$away_idx[n]] + 1
  } else if (data$y_home[n] == data$y_away[n]) {
    ties[data$home_idx[n]] <- ties[data$home_idx[n]] + 1
    ties[data$away_idx[n]] <- ties[data$away_idx[n]] + 1
  }
}

standings <- data.frame(1:data$N_teams, wins, losses, ties)
names(standings) <- c("Team", "Wins", "Losses", "Ties")

print(standings[rev(order(wins, ties)),], row.names=FALSE)
```

```
 Team Wins Losses Ties
    8   56     14    2
    9   56     15    1
    7   51     19    2
    5   51     20    1
    2   38     34    0
    6   34     36    2
    4   20     50    2
   10   19     53    0
    1   16     54    2
    3   13     59    0
```

At the top of the season standings we have

- First Place: Team 8,
- Second Place: Team 9,
- Third Place: Team 7,
- Fourth Place: Team 5.

Consequently the first round of the playoffs would consist of the two games

- Team 8 (Home) vs Team 5 (Away),
- Team 9 (Home) vs Team 7 (Away).

Many sporting events feature a variety of outcomes on which individuals can gamble, but here we'll focus on the score differentials.

For example we might have an opportunity to bet on the outcome of the first game with Team 8 giving 21 points to Team 5. This means that we're not betting on whether or not Team 8 defeats Team 5 but rather whether or not they beat Team 5 by more than 21 points.

More formally we might have an available gamble defined such that we can bet 110 units of currency and then win back our bet plus 100 units if

$$y_{8,5} > y_{5,8} + 21$$

but lose the 110 units if

$$y_{8,5} \leq y_{5,8} + 21.$$

Similarly we might be able to bet 110 units of currency and then win back our bet plus 100 units if

$$y_{8,5} < y_{5,8} + 21$$

but again lose the 110 unit bet if

$$y_{8,5} \leq y_{5,8} + 21.$$

These available then define a decision making problem. We have three actions that we can take – bet that the score differential of the first game will be larger than 21, bet that it will be less than 21, and not bet at all – with associated utilities that depend on the unknown game outcome. Reasoning about all of these possibilities is a bit easier if we organize all of this information into a table (Table 1).

Table 1: Bets on the winner of a game define a decision making problem. Inferences about the skills of the competing teams can be used to inform the best possible decisions.

Outcome	$y_{8,5} > y_{5,8} + 21$	$y_{8,5} < y_{5,8} + 21$	$y_{8,5} = y_{5,8} + 21$
Action 1	+100	-110	-110
Action 2	-110	+100	-110
Action 3	0	0	0

Notice the asymmetry in what we have to bet verses what we might be able to win in each bet. Moreover note that ties are excluded from either winning condition. The house, as they say, always wins.

Once we have established the viable actions and their potential utilities we can use the evaluate posterior predictive expected utilities,

$$\begin{aligned}
U_1 &= 100\,\pi(\{y_{8,5} > y_{5,8} + 21\}) - 110\,\pi(\{y_{8,5} \leq y_{5,8} + 21\}) \\
&= 100\,\pi(\{y_{8,5} > y_{5,8} + 21\}) - 110\,(1 - \pi(\{y_{8,5} > y_{5,8} + 21\})) \\
U_2 &= -110\,\pi(\{y_{8,5} \geq y_{5,8} + 21\}) + 100\,\pi(\{y_{8,5} < y_{5,8} + 21\}) \\
&= 100\,\pi(\{y_{8,5} < y_{5,8} + 21\}) - 110\,(1 - \pi(\{y_{8,5} < y_{5,8} + 21\})) \\
U_3 &= 0.
\end{aligned}$$

For the second game the available bets give 11 points to Team 5 (Table 2).

Table 2: The point spread for the second game define another decision making problem.

Outcome	$y_{9,7} > y_{7,9} + 11$	$y_{9,7} < y_{7,9} + 11$	$y_{9,7} = y_{7,9} + 11$
Action 1	+100	-110	-110
Action 2	-110	+100	-110
Action 3	0	0	0

All we have to do now is estimate the posterior predictive probability allocated to the various game outcomes. Fortunately this is straightforward with Markov chain Monte Carlo.

```r
data$N_playoff_games <- 2
data$playoff_home_idx <- c(8, 5)
data$playoff_away_idx <- c(9, 7)
data$playoff_week <- c(data$N_weeks + 1, data$N_weeks + 1)
```

```r
fit <- stan(file="stan_programs/season3_playoff.stan",
            data=data, seed=8438338,
            warmup=1000, iter=2024, refresh=0)
```

Fortunately the diagnostics remain clean.

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

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

```
samples <- util$extract_expectand_vals(fit)
base_samples <- util$filter_expectands(samples,
                                       c('alpha',
                                         'eta_off',
                                         'eta_def',
                                         'gamma0',
                                         'delta_off_free',
                                         'delta_def_free'),
                                       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.
```

Let's start with the first game. First we estimate the posterior predictive probabilities.

```
g1_p_home_spread <-
  util$implicit_subset_prob(samples,
                            function(yh, ya) yh > ya + 21,
                            list('yh' = 'y_home_playoff_pred[1]',
                                 'ya' = 'y_away_playoff_pred[1]'))

g1_p_away_spread <-
  util$implicit_subset_prob(samples,
                            function(yh, ya) yh < ya + 21,
                            list('yh' = 'y_home_playoff_pred[1]',
                                 'ya' = 'y_away_playoff_pred[1]'))
```

Then we compute the expected utilities for each bet.

```
m3_g1_U1 <- g1_p_home_spread[1] * 100 +
            (1 - g1_p_home_spread[1]) * (-110)
m3_g1_U2 <- g1_p_away_spread[1] * 100 +
            (1 - g1_p_away_spread[1]) * (-110)
```

Finally we can do the same for the second game.

```
g2_p_home_spread <-
  util$implicit_subset_prob(samples,
                            function(yh, ya) yh > ya + 11,
                            list('yh' = 'y_home_playoff_pred[2]',
                                 'ya' = 'y_away_playoff_pred[2]'))

g2_p_away_spread <-
  util$implicit_subset_prob(samples,
                            function(yh, ya) yh < ya + 11,
                            list('yh' = 'y_home_playoff_pred[2]',
                                 'ya' = 'y_away_playoff_pred[2]'))
```

```
m3_g2_U1 <- g2_p_home_spread[1] * 100 +
            (1 - g2_p_home_spread[1]) * (-110)
m3_g2_U2 <- g2_p_away_spread[1] * 100 +
            (1 - g2_p_away_spread[1]) * (-110)
```

Putting everything together allows us to compare the potential winnings of the two bets, as well as the third option to not bet and just enjoy the game risk-free.

```
expected_winnings <- data.frame(c(paste("Team 8 (Home)      Defeats",
                                        "Team 5 (Away) + 21"),
                                  paste("Team 5 (Away) + 21 Defeats",
                                        "Team 8 (Home)    "),
                                  paste("Team 9 (Home)      Defeats",
                                        "Team 7 (Away) + 11"),
                                  paste("Team 7 (Away) + 11 Defeats",
                                        "Team 9 (Home)    ")),
                                c(m3_g1_U1, m3_g1_U2,
                                  m3_g2_U1, m3_g2_U2))
names(expected_winnings) <- c("Bet", "Expected Winnings")

print(expected_winnings, row.names=FALSE, right=FALSE)
```

```
 Bet                                           Expected Winnings
 Team 8 (Home)      Defeats Team 5 (Away) + 21 -77.329164
 Team 5 (Away) + 21 Defeats Team 8 (Home)       62.620648
 Team 9 (Home)      Defeats Team 7 (Away) + 11 -17.926448
 Team 7 (Away) + 11 Defeats Team 9 (Home)       -0.689294
```

Based on our inferences, which in turn are based on our modeling assumptions and the observed data, the only profitable bet here is to take Team 5 and the points in the first playoff game.

What if we had ignored retrodictive performance and stuck with our first model? Well we can just calculate how the expected utilities would have been informed by those unreliable inferences.

```
fit <- stan(file="stan_programs/season1_playoff.stan",
            data=data, seed=8438338,
            warmup=1000, iter=2024, refresh=0)
```

There are no computational concerns.

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

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

```
samples <- util$extract_expectand_vals(fit)
base_samples <- util$filter_expectands(samples,
                                       c('alpha',
                                         'delta_off_free',
                                         'delta_def_free'),
                                       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.
```

The calculation of the expected utilities proceeds the same as before.

```
g1_p_home_spread <-
  util$implicit_subset_prob(samples,
                            function(yh, ya) yh > ya + 21,
                            list('yh' = 'y_home_playoff_pred[1]',
                                 'ya' = 'y_away_playoff_pred[1]'))

g1_p_away_spread <-
  util$implicit_subset_prob(samples,
                            function(yh, ya) yh < ya + 21,
                            list('yh' = 'y_home_playoff_pred[1]',
                                 'ya' = 'y_away_playoff_pred[1]'))
```

```
m1_g1_U1 <- g1_p_home_spread[1] * 100 +
            (1 - g1_p_home_spread[1]) * (-110)
m1_g1_U2 <- g1_p_away_spread[1] * 100 +
            (1 - g1_p_away_spread[1]) * (-110)
```

```
g2_p_home_spread <-
  util$implicit_subset_prob(samples,
                            function(yh, ya) yh > ya + 11,
                            list('yh' = 'y_home_playoff_pred[2]',
                                 'ya' = 'y_away_playoff_pred[2]'))

g2_p_away_spread <-
  util$implicit_subset_prob(samples,
                            function(yh, ya) yh < ya + 11,
                            list('yh' = 'y_home_playoff_pred[2]',
                                 'ya' = 'y_away_playoff_pred[2]'))

m1_g2_U1 <- g2_p_home_spread[1] * 100 +
            (1 - g2_p_home_spread[1]) * (-110)
m1_g2_U2 <- g2_p_away_spread[1] * 100 +
            (1 - g2_p_away_spread[1]) * (-110)
```

Now we can directly compare the optimal decisions according to the two models.

```
expected_winnings <- data.frame(c(paste("Tm 8 (Home)      Dfts",
                                        "Tm 5 (Away) + 21"),
                                  paste("Tm 5 (Away) + 21 Dfts",
                                        "Tm 8 (Home)    "),
                                  paste("Tm 9 (Home)      Dfts",
                                        "Tm 7 (Away) + 11"),
                                  paste("Tm 7 (Away) + 11 Dfts",
                                        "Tm 9 (Home)    ")),
                                c(m1_g1_U1, m1_g1_U2,
                                  m1_g2_U1, m1_g2_U2),
                                c(m3_g1_U1, m3_g1_U2,
                                  m3_g2_U1, m3_g2_U2))
names(expected_winnings) <- c("Bet",
                              "Expd Wngs (M1)",
                              "Expd Wngs (M3)")

print(expected_winnings, row.names=FALSE, right=FALSE)
```

```
Bet                                    Expd Wngs (M1) Expd Wngs (M3)
Tm 8 (Home)      Dfts Tm 5 (Away) + 21 -105.90104     -77.329164
Tm 5 (Away) + 21 Dfts Tm 8 (Home)        94.92753      62.620648
Tm 9 (Home)      Dfts Tm 7 (Away) + 11  -94.88692     -17.926448
Tm 7 (Away) + 11 Dfts Tm 9 (Home)        81.64673      -0.689294
```

The magnitude of the expected utilities from the first model are quite a bit larger that the expected utilities from the third model. Moreover according to the first model both away bets would be pretty strongly profitable. If we used this model we might have been tempted to make a bad bet on the away team in game 2!

Bayesian decision theory is a powerful tool, but it is only as reliable as the posterior inferences and the reliability of the posterior inferences is limited by the adequacy of the modeling assumptions.

7 Conclusion

The ubiquity of pairwise comparisons in practical applications may not be immediately obvious, but once we start looking for them they become difficult to miss. Even if a data generating process doesn't fully manifest as a pairwise comparison, parts of it might.

With the general modeling techniques presented in this chapter, however, we are equipped to appropriately model pairwise comparisons whenever they might arise. That said these techniques are very much only a stable foundation, and they can be extended and combined with other techniques to develop even more sophisticated models.

For example instead of treating item qualities as individual parameters to infer we can often derive them from other parts of a model, in particular as the output of some function of item properties. In a sports application for instance we might be able to model the offensive skill of a team as a combination of individual player skills. Similarly we might model the ability of a student to correctly answer a test question as a function of their cumulative study efforts.

The basic pairwise comparison modeling strategy can also be generalized to model the comparison of multiple items at the same time. For example consider a comparison of multiple items that results in a singular best item. We can use a multinomial distribution to model this output,

$$p(y_{i_1,\ldots,i_K}) = \text{multinomial}(y_{i_1,\ldots,i_K} \mid p_{i_1}, \ldots, p_{i_K})$$

and then couple the multinomial probabilities to item qualities with a softmax function,

$$\begin{aligned}
(p_{i_1}, \ldots, p_{i_K}) &= \text{softmax}(\alpha_{i_1}, \ldots, \alpha_{i_K}) \\
&= \left(\frac{\exp(\alpha_{i_1})}{\sum_{k=1}^{K} \exp(\alpha_{i_k})}, \ldots, \frac{\exp(\alpha_{i_K})}{\sum_{k=1}^{K} \exp(\alpha_{i_k})} \right).
\end{aligned}$$

We can even iterate this approach to model observed rankings, using a multinomial distribution informed by the item qualities to model the top ranked item, another multinomial over the remaining items to model the second ranked item, and so on.

Acknowledgements

A very special thanks to everyone supporting me on Patreon: Adam Fleischhacker, Adriano Yoshino, Alejandro Navarro-Martínez, Alessandro Varacca, Alex D, Alexander Noll, Andrea Serafino, Andrew Mascioli, Andrew Rouillard, Andrew Vigotsky, Ara Winter, Austin Rochford, Avraham Adler, Ben Matthews, Ben Swallow, Benoit Essiambre, Bertrand Wilden, Bradley Kolb, Brendan Galdo, Bryan Chang, Brynjolfur Gauti Jónsson, Cameron Smith, Canaan Breiss, Cat Shark, CG, Charles Naylor, Chase Dwelle, Chris Jones, Christopher Mehrvarzi, Colin Carroll, Colin McAuliffe, Damien Mannion, dan mackinlay, Dan W Joyce, Dan Waxman, Dan Weitzenfeld, Daniel Edward Marthaler, Daniel Saunders, Darshan Pandit, Darthmaluus, David Galley, David Wurtz, Doug Rivers, Dr. Jobo, Dr. Omri Har Shemesh, Dylan Maher, Ed Cashin, Edgar Merkle, Eli Witus, Eric LaMotte, Ero Carrera, Eugene O'Friel, Felipe González, Fergus Chadwick, Finn Lindgren, Francesco Corona, Geoff Rollins, Håkan Johansson, Hamed Bastan-Hagh, haubur, Hector Munoz, Henri Wallen, hs, Hugo Botha, Ian, Ian Costley, idontgetoutmuch, Ignacio Vera, Ilaria Prosdocimi, Isaac Vock, Isidor Belic, 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, John Flournoy, Jonathan H. Morgan, Jonathon Vallejo, Joran Jongerling, Josh Knecht, JU, June, Justin Bois, Kádár András, Karim Naguib, Karim Osman, Kejia Shi, Kristian Gårdhus Wichmann, Lars Barquist, lizzie , LOU ODETTE, Luís F, Marcel Lüthi, Marek Kwiatkowski, Mariana Carmona, Mark Donoghoe, Markus P., Márton Vaitkus, Matthew, Matthew Kay, Matthew Mulvahill, Matthieu LEROY, Mattia Arsendi, Maurits van der Meer, Max, Michael Colaresi, Michael DeWitt, Michael Dillon, Michael Lerner, Mick Cooney, Mike Lawrence, MisterMentat , N Sanders, N.S. , Name, Nathaniel Burbank, Nic Fishman, Nicholas Clark, Nicholas Cowie, Nick S, Octavio Medina, Ole Rogeberg, Oliver Crook, Olivier Ma, Patrick Kelley, Patrick Boehnke, Pau Pereira Batlle, Peter Johnson, Pieter van den Berg, ptr, quasar, Ramiro Barrantes Reynolds, Raúl Peralta Lozada, Ravin Kumar, Rémi , Rex Ha, Riccardo Fusaroli, Richard Nerland, Robert Frost, Robert Goldman, Robert kohn, Robin Taylor, Ryan Gan, Ryan Grossman, Ryan Kelly, S Hong, Sean Wilson, Sergiy Protsiv, Seth Axen, shira, Simon Duane, Simon Lilburn, Simone Sebben, Spencer Carter, sssz, Stefan Lorenz, Stephen Lienhard, Steve Harris, Stew Watts, Stone Chen, Susan Holmes, Svilup, Tao Ye, Tate Tunstall, Tatsuo Okubo, Teresa Ortiz, Theodore Dasher, Thomas Siegert, Thomas Vladeck, Tobychev, Tony Wuersch, Tyler Burch, Virginia Fisher, Vladimir Markov, Wil Yegelwel, Will Farr, Will Lowe, Will Wen, woejozney, yolhaj, yureq, Zach A, Zad Rafi, and Zhengchen Cai.

References

Baker, Frank B. 2001. *The Basics of Item Response Theory*. Second. College Park, MD: ERIC Clearinghouse on Assessment; Evaluation.

Bradley, Ralph Allan, and Milton E. Terry. 1952. "Rank Analysis of Incomplete Block Designs: I. The Method of Paired Comparisons." *Biometrika* 39 (3/4): 324–45.

Clark, John, and Derek Allan Holton. 1991. *A First Look at Graph Theory*. World Scientific Publishing Co., Inc., Teaneck, NJ.

Cormen, Thomas H, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. 2022. *Introduction to Algorithms*. Fourth. Cambridge, MA: MIT Press.

Elo, Arpad E. 1967. "The Proposed USCF Rating System, Its Development, Theory, and Applications." *Chess Life* XXII (8): 242–47.

Hamilton, Ian, Nick Tawn, and David Firth. 2023. "The Many Routes to the Ubiquitous Bradley-Terry Model." *arXiv e-Prints*, no. 2312.13619 (December).

Rasch, Georg. 1960. *Probabilistic Models for Some Intelligence and Attainment Tests*. Danish Institute for Educational Research.

License

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

Original Computing Environment

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

```
CC=clang

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

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

```r
sessionInfo()
```

```
R version 4.3.2 (2023-10-31)
Platform: x86_64-apple-darwin20 (64-bit)
Running under: macOS Sonoma 14.4.1

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

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

time zone: America/New_York
tzcode source: internal

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

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

loaded via a namespace (and not attached):
 [1] gtable_0.3.4       jsonlite_1.8.8     compiler_4.3.2     Rcpp_1.0.11
 [5] stringr_1.5.1      parallel_4.3.2     gridExtra_2.3      scales_1.3.0
 [9] yaml_2.3.8         fastmap_1.1.1      ggplot2_3.4.4      R6_2.5.1
[13] curl_5.2.0         knitr_1.45         tibble_3.2.1       munsell_0.5.0
[17] pillar_1.9.0       rlang_1.1.2        utf8_1.2.4         V8_4.4.1
[21] stringi_1.8.3      inline_0.3.19      xfun_0.41          RcppParallel_5.1.7
[25] cli_3.6.2          magrittr_2.0.3     digest_0.6.33      grid_4.3.2
[29] lifecycle_1.0.4    vctrs_0.6.5        evaluate_0.23      glue_1.6.2
[33] QuickJSR_1.0.8     codetools_0.2-19   stats4_4.3.2       pkgbuild_1.4.3
[37] fansi_1.0.6        colorspace_2.1-0   rmarkdown_2.25     matrixStats_1.2.0
[41] tools_4.3.2        loo_2.6.0          pkgconfig_2.0.3    htmltools_0.5.7
```

Stan
Program 1 `simu\_bradley\_terry.stan`

```
data {
  int<lower=1> N_players; // Number of players
  int<lower=1> N_games;   // Number of games

  // Matchup probabilities
  simplex[N_players] player1_probs;
  array[N_players] simplex[N_players] pair_probs;
}

generated quantities {
  // Player skills
  array[N_players] real alpha
    = normal_rng(rep_vector(0, N_players), log(sqrt(2)) / 2.32);

  // Game outcomes
  array[N_games] int<lower=1, upper=N_players> player1_idx;
  array[N_games] int<lower=1, upper=N_players> player2_idx;
  array[N_games] int<lower=0, upper=1> y;

  for (n in 1:N_games) {
    // Simulate first player
    player1_idx[n] = categorical_rng(player1_probs);

    // Simulate second player
    player2_idx[n] = categorical_rng(pair_probs[player1_idx[n]]);

    // Simulate winner
    y[n] = bernoulli_logit_rng(  alpha[player1_idx[n]]
                               - alpha[player2_idx[n]]);
  }
}
```

Stan
Program 2 `bradley\_terry1.stan`

```
data {
  int<lower=1> N_players; // Number of players
  int<lower=1> N_games;   // Number of games

  // Game outcomes
  array[N_games] int<lower=1, upper=N_players> player1_idx;
  array[N_games] int<lower=1, upper=N_players> player2_idx;
  array[N_games] int<lower=0, upper=1> y;
}

parameters {
  vector[N_players] alpha; // Player skills
}

model {
  // Implicit uniform prior density function for the alpha components

  // Observational model
  y ~ bernoulli_logit(alpha[player1_idx] - alpha[player2_idx]);
}

generated quantities {
  array[N_games] int<lower=0, upper=1> y_pred;
  array[N_players] int<lower=0> win_counts_pred
    = rep_array(0, N_players);
  array[N_players] int<lower=0> loss_counts_pred
    = rep_array(0, N_players);
  array[N_players] real win_freq_pred;

  for (n in 1:N_games) {
    int idx1 = player1_idx[n];
    int idx2 = player2_idx[n];

    y_pred[n] = bernoulli_logit_rng(alpha[idx1] - alpha[idx2]);
    if (y_pred[n]) {
      win_counts_pred[idx1] += 1;
      loss_counts_pred[idx2] += 1;
    } else {
      win_counts_pred[idx2] += 1;
      loss_counts_pred[idx1] += 1;
    }
  }

  for (i in 1:N_players) {
    win_freq_pred[i] =   (1.0 * win_counts_pred[i])
                       / (win_counts_pred[i] + loss_counts_pred[i]);
  }
}
```

Stan
Program 3 `bradley\_terry2.stan`

```
data {
  int<lower=1> N_players; // Number of players
  int<lower=1> N_games;   // Number of games

  // Game outcomes
  array[N_games] int<lower=1, upper=N_players> player1_idx;
  array[N_games] int<lower=1, upper=N_players> player2_idx;
  array[N_games] int<lower=0, upper=1> y;
}

parameters {
  vector[N_players] alpha; // Player skills
}

model {
  // Prior model
  // -10 <~ alpha[i] <~ +10
  alpha ~ normal(0, 10 / 2.32);

  // Observational model
  y ~ bernoulli_logit(alpha[player1_idx] - alpha[player2_idx]);
}

generated quantities {
  array[N_games] int<lower=0, upper=1> y_pred;
  array[N_players] int<lower=0> win_counts_pred
    = rep_array(0, N_players);
  array[N_players] int<lower=0> loss_counts_pred
    = rep_array(0, N_players);
  array[N_players] real win_freq_pred;

  for (n in 1:N_games) {
    int idx1 = player1_idx[n];
    int idx2 = player2_idx[n];

    y_pred[n] = bernoulli_logit_rng(alpha[idx1] - alpha[idx2]);
    if (y_pred[n]) {
      win_counts_pred[idx1] += 1;
      loss_counts_pred[idx2] += 1;
    } else {
      win_counts_pred[idx2] += 1;
      loss_counts_pred[idx1] += 1;
    }
  }

  for (i in 1:N_players) {
    win_freq_pred[i] =   (1.0 * win_counts_pred[i])
                       / (win_counts_pred[i] + loss_counts_pred[i]);
  }
}
```

Stan
Program 4 bradley_terry3.stan

```
data {
  int<lower=1> N_players; // Number of players
  int<lower=1> N_games;   // Number of games

  // Game outcomes
  array[N_games] int<lower=1, upper=N_players> player1_idx;
  array[N_games] int<lower=1, upper=N_players> player2_idx;
  array[N_games] int<lower=0, upper=1> y;
}

parameters {
  vector[N_players] alpha; // Player skills
}

model {
  // Prior model
  // -log sqrt(2) <~ alpha[i] <~ +log sqrt(2)
  alpha ~ normal(0, log(sqrt(2)) / 2.32);

  // Observational model
  y ~ bernoulli_logit(alpha[player1_idx] - alpha[player2_idx]);
}

generated quantities {
  array[N_games] int<lower=0, upper=1> y_pred;
  array[N_players] int<lower=0> win_counts_pred
    = rep_array(0, N_players);
  array[N_players] int<lower=0> loss_counts_pred
    = rep_array(0, N_players);
  array[N_players] real win_freq_pred;

  for (n in 1:N_games) {
    int idx1 = player1_idx[n];
    int idx2 = player2_idx[n];

    y_pred[n] = bernoulli_logit_rng(alpha[idx1] - alpha[idx2]);
    if (y_pred[n]) {
      win_counts_pred[idx1] += 1;
      loss_counts_pred[idx2] += 1;
    } else {
      win_counts_pred[idx2] += 1;
      loss_counts_pred[idx1] += 1;
    }
  }

  for (i in 1:N_players) {
    win_freq_pred[i] =   (1.0 * win_counts_pred[i])
                       / (win_counts_pred[i] + loss_counts_pred[i]);
  }
}
```

Stan
Program 5 `bradley\_terry4.stan`

```
data {
  int<lower=1> N_players; // Number of players
  int<lower=1> N_games;   // Number of games

  // Game outcomes
  array[N_games] int<lower=1, upper=N_players> player1_idx;
  array[N_games] int<lower=1, upper=N_players> player2_idx;
  array[N_games] int<lower=0, upper=1> y;
}

parameters {
  vector[N_players - 1] delta_free; // Relative player skills
}

transformed parameters {
  // Relative player skills with anchor skill
  vector[N_players] delta = append_row([0]', delta_free);
}

model {
  // Prior model
  // -sqrt(2) * 10 <~ beta_free[i] <~ + sqrt(2) * 10
  delta_free ~ normal(0, sqrt(2) * 10 / 2.32);

  // Observational model
  y ~ bernoulli_logit(delta[player1_idx] - delta[player2_idx]);
}

generated quantities {
  array[N_games] int<lower=0, upper=1> y_pred;
  array[N_players] int<lower=0> win_counts_pred
    = rep_array(0, N_players);
  array[N_players] int<lower=0> loss_counts_pred
    = rep_array(0, N_players);
  array[N_players] real win_freq_pred;

  for (n in 1:N_games) {
    int idx1 = player1_idx[n];
    int idx2 = player2_idx[n];

    y_pred[n] = bernoulli_logit_rng(delta[idx1] - delta[idx2]);
    if (y_pred[n]) {
      win_counts_pred[idx1] += 1;
      loss_counts_pred[idx2] += 1;
    } else {
      win_counts_pred[idx2] += 1;
      loss_counts_pred[idx1] += 1;
    }
  }
```

Stan
Program 6 bradley_terry5.stan

```
data {
  int<lower=1> N_players; // Number of players
  int<lower=1> N_games;   // Number of games

  // Game outcomes
  array[N_games] int<lower=1, upper=N_players> player1_idx;
  array[N_games] int<lower=1, upper=N_players> player2_idx;
  array[N_games] int<lower=0, upper=1> y;

  // Anchor configuration
  int<lower=0, upper=N_players - 1> N_anchors;
  array[N_anchors] int<lower=1, upper=N_players> anchor_idx;
}

parameters {
  // Relative player skills
  vector[N_players - N_anchors] delta_free;
}

transformed parameters {
  // Relative player skills with anchor skills
  vector[N_players] delta = rep_vector(0, N_players);

  {
    array[N_anchors + 2] int boundary_idx
      = append_array(append_array({0}, anchor_idx), {N_players + 1});

    for (n in 1:(N_anchors + 1)) {
      int idx1 = boundary_idx[n]     + 1;
      int idx2 = boundary_idx[n + 1] - 1;
      if (idx1 <= idx2) {
        delta[idx1:idx2] = delta_free[(idx1 - n + 1):(idx2 - n + 1)];
      }
    }
  }
}

model {
  // Prior model
  // -sqrt(2) * 10 <~ beta_free[i] <~ + sqrt(2) * 10
  delta_free ~ normal(0, sqrt(2) * 10 / 2.32);

  // Observational model
  y ~ bernoulli_logit(delta[player1_idx] - delta[player2_idx]);
}

generated quantities {
  array[N_games] int<lower=0, upper=1> y_pred;
  array[N_players] int<lower=0> win_counts_pred
    = rep_array(0, N_players);
```

Stan
Program 7 irt1.stan

```
data {
  int<lower=1> N_questions; // Number of questions
  int<lower=1> N_students;  // Number of students
  int<lower=1> N_answers;   // Number of answers

  // Testing configuration and outcomes
  array[N_answers] int<lower=1, upper=N_questions> question_idx;
  array[N_answers] int<lower=1, upper=N_students>  student_idx;
  array[N_answers] int<lower=0, upper=1> y;
}

parameters {
  vector[N_questions]    beta;       // Question difficulties
  vector[N_students - 1] alpha_free; // Relative student abilities
}

transformed parameters {
  vector[N_students] alpha = append_row([0]', alpha_free);
}

model {
  // Prior model
  beta       ~ normal(0, 3 / 2.32); // -3 <~  beta[i] <~ +3
  alpha_free ~ normal(0, 3 / 2.32); // -3 <~ alpha[j] <~ +3

  // Observational model
  y ~ bernoulli_logit(alpha[student_idx] - beta[question_idx]);
}

generated quantities {
  array[N_questions] real mean_q_pred = rep_array(0, N_questions);
  array[N_questions] real var_q_pred  = rep_array(0, N_questions);
  array[N_students]  real mean_s_pred = rep_array(0, N_students);
  array[N_students]  real var_s_pred  = rep_array(0, N_students);

  {
    array[N_questions] real N_answers_q = rep_array(0, N_questions);
    array[N_students]  real N_answers_s = rep_array(0, N_students);

    for (n in 1:N_answers) {
      int i = question_idx[n];
      int j = student_idx[n];
      real delta = 0;

      int y_pred = bernoulli_logit_rng(alpha[j] - beta[i]);

      N_answers_q[i] += 1;
      delta = y_pred - mean_q_pred[i];
      mean_q_pred[i] += delta / N_answers_q[i];
      var_q_pred[i] += delta * (y_pred - mean_q_pred[i]);
```

Stan
Program 8 irt2.stan

```
data {
  int<lower=1> N_questions; // Number of questions
  int<lower=1> N_students;  // Number of students
  int<lower=1> N_answers;   // Number of answers

  // Testing configuration and outcomes
  array[N_answers] int<lower=1, upper=N_questions> question_idx;
  array[N_answers] int<lower=1, upper=N_students>  student_idx;
  array[N_answers] int<lower=0, upper=1> y;
}

parameters {
  // Relative question discriminations
  vector<lower=0>[N_questions - 1] gamma_free;

  vector[N_questions]    beta;       // Question difficulties
  vector[N_students - 1] alpha_free; // Relative student abilities
}

transformed parameters {
  vector[N_questions] gamma = append_row([1]', gamma_free);
  vector[N_students] alpha = append_row([0]', alpha_free);
}

model {
  // Prior model
  gamma_free ~ normal(0, 10 / 2.57); // 0 <~ gamma[i] <~ +10
  beta       ~ normal(0, 3 / 2.32);  // -3 <~  beta[i] <~ +3
  alpha_free ~ normal(0, 3 / 2.32);  // -3 <~ alpha[j] <~ +3

  // Observational model
  y ~ bernoulli_logit(   gamma[question_idx]
                      .* (alpha[student_idx] - beta[question_idx]) );
}

generated quantities {
  array[N_questions] real mean_q_pred = rep_array(0, N_questions);
  array[N_questions] real var_q_pred  = rep_array(0, N_questions);
  array[N_students]  real mean_s_pred = rep_array(0, N_students);
  array[N_students]  real var_s_pred  = rep_array(0, N_students);

  {
    array[N_questions] real N_answers_q = rep_array(0, N_questions);
    array[N_students]  real N_answers_s = rep_array(0, N_students);

    for (n in 1:N_answers) {
      int i = question_idx[n];
      int j = student_idx[n];
      real delta = 0;
```

Stan
Program 9 irt3.stan

```
data {
  int<lower=1> N_questions; // Number of questions
  int<lower=1> N_students;  // Number of students
  int<lower=1> N_answers;   // Number of answers

  // Testing configuration and outcomes
  array[N_answers] int<lower=1, upper=N_questions> question_idx;
  array[N_answers] int<lower=1, upper=N_students>  student_idx;
  array[N_answers] int<lower=0, upper=1> y;
}

parameters {
  // Relative question discriminations
  vector<lower=0>[N_questions - 1] gamma_free;

  vector[N_questions] beta;  // Question difficulties

  // Non-centered, relative student abilities
  vector[N_students] alpha_tilde;

  real delta_mu;      // Difference in student cohort ability locations
  real<lower=0> tau1; // First student cohort ability variation
  real<lower=0> tau2; // Second student cohort ability variation
}

transformed parameters {
  vector[N_questions] gamma = append_row([1]', gamma_free);

  vector[N_students] alpha;
  alpha[1:(N_students / 2)]
    = tau1 * alpha_tilde[1:(N_students / 2)];
  alpha[(N_students / 2):N_students]
    = delta_mu + tau2 * alpha_tilde[(N_students / 2):N_students];
}

model {
  // Prior model
  gamma_free ~ normal(0, 10 / 2.57);
  beta ~ normal(0, 3 / 2.32);

  alpha_tilde ~ normal(0, 1);

  delta_mu ~ normal(0, 3 / 2.32);
  tau1 ~ normal(0, 3 / 2.57);
  tau2 ~ normal(0, 3 / 2.57);

  // Observational model
  y ~ bernoulli_logit(   gamma[question_idx]
                      .* (alpha[student_idx] - beta[question_idx]) );
}
```

Stan
Program 10 season1.stan

```
data {
  int<lower=1> N_games; // Number of games

  // Matchups
  int<lower=1> N_teams;
  array[N_games] int<lower=1, upper=N_teams> home_idx;
  array[N_games] int<lower=1, upper=N_teams> away_idx;

  int<lower=1> N_weeks; // Number of weeks
  array[N_games] int<lower=1, upper=N_weeks> week;

  // Game scores
  array[N_games] int<lower=0> y_home;
  array[N_games] int<lower=0> y_away;
}

parameters {
  real alpha;                          // Baseline log score
  vector[N_teams - 1] delta_off_free; // Relative team offensive skills
  vector[N_teams - 1] delta_def_free; // Relative team defensive skills
}

transformed parameters {
  // Relative skills for all teams
  vector[N_teams] delta_off = append_row([0]', delta_off_free);
  vector[N_teams] delta_def = append_row([0]', delta_def_free);
}

model {
  // Prior model
  alpha ~ normal(3.77, 0.55 / 2.32); // 25 <~  exp(alpha) <~ 75

  // -2 <~ delta_off/delta_def <~ +2
  delta_off_free ~ normal(0, 2 / 2.32);
  delta_def_free ~ normal(0, 2 / 2.32);

  // Observational model
  y_home ~ poisson_log(  alpha
                       + delta_off[home_idx] - delta_def[away_idx]);
  y_away ~ poisson_log(  alpha
                       + delta_off[away_idx] - delta_def[home_idx]);
}

generated quantities {
  array[N_games] int<lower=0> y_home_pred;
  array[N_games] int<lower=0> y_away_pred;
  array[N_games] int y_diff_pred;
  array[N_games] int<lower=0> y_sum_pred;

  for (n in 1:N_games) {
```

Stan
Program 11 season2.stan

```
data {
  int<lower=1> N_games; // Number of games

  // Matchups
  int<lower=1> N_teams;
  array[N_games] int<lower=1, upper=N_teams> home_idx;
  array[N_games] int<lower=1, upper=N_teams> away_idx;

  int<lower=1> N_weeks; // Number of weeks
  array[N_games] int<lower=1, upper=N_weeks> week;

  // Game scores
  array[N_games] int<lower=0> y_home;
  array[N_games] int<lower=0> y_away;
}

parameters {
  real alpha;                           // Baseline log score
  real eta_off;                         // Offensive home-field advantage
  real eta_def;                         // Defensive home-field advantage
  vector[N_teams - 1] delta_off_free; // Relative team offensive skills
  vector[N_teams - 1] delta_def_free; // Relative team defensive skills
}

transformed parameters {
  // Relative skills for all teams
  vector[N_teams] delta_off = append_row([0]', delta_off_free);
  vector[N_teams] delta_def = append_row([0]', delta_def_free);
}

model {
  // Prior model
  alpha ~ normal(3.77, 0.55 / 2.32); // 25 <~  exp(alpha) <~ 75

  // 0.78 <~ exp(eta_off/eta_def) <~ 1.28
  eta_off ~ normal(0, 0.25 / 2.32);
  eta_def ~ normal(0, 0.25 / 2.32);

  // -2 <~ delta_off/delta_def <~ +2
  delta_off_free ~ normal(0, 2 / 2.32);
  delta_def_free ~ normal(0, 2 / 2.32);

  // Observational model
  y_home ~ poisson_log(  alpha + eta_off
                       + delta_off[home_idx] - delta_def[away_idx]);
  y_away ~ poisson_log(  alpha - eta_def
                       + delta_off[away_idx] - delta_def[home_idx]);
}

generated quantities {
```

Stan Program 12 season3.stan

```
data {
  int<lower=1> N_games; // Number of games

  // Matchups
  int<lower=1> N_teams;
  array[N_games] int<lower=1, upper=N_teams> home_idx;
  array[N_games] int<lower=1, upper=N_teams> away_idx;

  int<lower=1> N_weeks; // Number of weeks
  array[N_games] int<lower=1, upper=N_weeks> week;

  // Game scores
  array[N_games] int<lower=0> y_home;
  array[N_games] int<lower=0> y_away;
}

parameters {
  real alpha;                         // Baseline log score
  real eta_off;                       // Offensive home-field advantage
  real eta_def;                       // Defensive home-field advantage
  real<lower=0> gamma0;               // Discrimination mantissa
  vector[N_teams - 1] delta_off_free; // Relative team offensive skills
  vector[N_teams - 1] delta_def_free; // Relative team defensive skills
}

transformed parameters {
  // Relative skills for all teams
  vector[N_teams] delta_off = append_row([0]', delta_off_free);
  vector[N_teams] delta_def = append_row([0]', delta_def_free);
}

model {
  // Prior model
  alpha ~ normal(3.77, 0.55 / 2.32); // 25 <~  exp(alpha) <~ 75

  // 0.78 <~ exp(eta_off/eta_def) <~ 1.28
  eta_off ~ normal(0, 0.25 / 2.32);
  eta_def ~ normal(0, 0.25 / 2.32);

  gamma0 ~ normal(0.9, 0.1 / 2.57); // 0.8  <~ gamma0 <~ 1.0

  // -2 <~ delta_off/delta_def <~ +2
  delta_off_free ~ normal(0, 2 / 2.32);
  delta_def_free ~ normal(0, 2 / 2.32);

  // Observational model
  y_home ~ poisson_log(  alpha + eta_off
                       + to_vector(pow(gamma0, week))
                         .* (  delta_off[home_idx]
                             - delta_def[away_idx] ) );
```

Stan
Program 13 season3_playoff.stan

```
data {
  int<lower=1> N_games; // Number of games

  // Matchups
  int<lower=1> N_teams;
  array[N_games] int<lower=1, upper=N_teams> home_idx;
  array[N_games] int<lower=1, upper=N_teams> away_idx;

  int<lower=1> N_weeks; // Number of weeks
  array[N_games] int<lower=1, upper=N_weeks> week;

  // Game scores
  array[N_games] int<lower=0> y_home;
  array[N_games] int<lower=0> y_away;

  // Playoff games
  int<lower=1> N_playoff_games; // Number of playoff games
  array[N_playoff_games] int<lower=1, upper=N_teams> playoff_home_idx;
  array[N_playoff_games] int<lower=1, upper=N_teams> playoff_away_idx;
  array[N_playoff_games] int playoff_week;
}

parameters {
  real alpha;                          // Baseline log score
  real eta_off;                        // Offensive home-field advantage
  real eta_def;                        // Defensive home-field advantage
  real<lower=0> gamma0;                // Discrimination mantissa
  vector[N_teams - 1] delta_off_free; // Relative team offensive skills
  vector[N_teams - 1] delta_def_free; // Relative team defensive skills
}

transformed parameters {
  // Relative skills for all teams
  vector[N_teams] delta_off = append_row([0]', delta_off_free);
  vector[N_teams] delta_def = append_row([0]', delta_def_free);
}

model {
  // Prior model
  alpha ~ normal(3.77, 0.55 / 2.32); // 25 <~  exp(alpha) <~ 75

  // 0.78 <~ exp(eta_off/eta_def) <~ 1.28
  eta_off ~ normal(0, 0.25 / 2.32);
  eta_def ~ normal(0, 0.25 / 2.32);

  gamma0 ~ normal(0.9, 0.1 / 2.57); // 0.8  <~ gamma0 <~ 1.0

  // -2 <~ delta_off/delta_def <~ +2
  delta_off_free ~ normal(0, 2 / 2.32);
  delta_def_free ~ normal(0, 2 / 2.32);
```

Stan
Program 14 season1_playoff.stan

```
data {
  int<lower=1> N_games; // Number of games

  // Matchups
  int<lower=1> N_teams;
  array[N_games] int<lower=1, upper=N_teams> home_idx;
  array[N_games] int<lower=1, upper=N_teams> away_idx;

  int<lower=1> N_weeks; // Number of weeks
  array[N_games] int<lower=1, upper=N_weeks> week;

  // Game scores
  array[N_games] int<lower=0> y_home;
  array[N_games] int<lower=0> y_away;

  // Playoff games
  int<lower=1> N_playoff_games; // Number of playoff games
  array[N_playoff_games] int<lower=1, upper=N_teams> playoff_home_idx;
  array[N_playoff_games] int<lower=1, upper=N_teams> playoff_away_idx;
  array[N_playoff_games] int playoff_week;
}

parameters {
  real alpha;                         // Baseline log score
  vector[N_teams - 1] delta_off_free; // Relative team offensive skills
  vector[N_teams - 1] delta_def_free; // Relative team defensive skills
}

transformed parameters {
  // Relative skills for all teams
  vector[N_teams] delta_off = append_row([0]', delta_off_free);
  vector[N_teams] delta_def = append_row([0]', delta_def_free);
}

model {
  // Prior model
  alpha ~ normal(3.77, 0.55 / 2.32); // 25 <~  exp(alpha) <~ 75

  // -2 <~ delta_off/delta_def <~ +2
  delta_off_free ~ normal(0, 2 / 2.32);
  delta_def_free ~ normal(0, 2 / 2.32);

  // Observational model
  y_home ~ poisson_log(  alpha
                       + delta_off[home_idx] - delta_def[away_idx]);
  y_away ~ poisson_log(  alpha
                       + delta_off[away_idx] - delta_def[home_idx]);
}

generated quantities {
```