Troubleshooting Bayesian cognitive models

Beth Baribault & Anne G.E. Collins
University of California, Berkeley

Abstract

Using Bayesian methods to apply computational models of cognitive processes, or *Bayesian cognitive modeling*, is an important new trend in psychological research. The rise of Bayesian cognitive modeling has been accelerated by the introduction of software that efficiently automates the Markov chain Monte Carlo sampling used for Bayesian model fitting — including the popular Stan and PyMC packages, which automate the dynamic Hamiltonian Monte Carlo and No-U-Turn Sampler (HMC/NUTS) algorithms that we spotlight here. Unfortunately, Bayesian cognitive models can struggle to pass the growing number of diagnostic checks required of Bayesian models. If any failures are left undetected, inferences about cognition based on the model's output may be biased or incorrect. As such, Bayesian cognitive models almost always require *troubleshooting* before being used for inference. Here, we present a deep treatment of the diagnostic checks and procedures that are critical for effective troubleshooting, but are often left underspecified by tutorial papers. After a conceptual introduction to Bayesian cognitive modeling and HMC/NUTS sampling, we outline the diagnostic metrics, procedures, and plots necessary to detect problems in model output with an emphasis on how these requirements have recently been changed and extended. Throughout, we explain how uncovering the exact nature of the problem is often the key to identifying solutions. We also demonstrate the troubleshooting process for an example hierarchical Bayesian model of reinforcement learning, including supplementary code. With this comprehensive guide to techniques for detecting, identifying, and overcoming problems in fitting Bayesian cognitive models, psychologists across subfields can more confidently build and use Bayesian cognitive models in their research.

Keywords: cognitive modeling, Bayesian methods, computational models, Hamiltonian Monte Carlo

The Bayesian revolution of the last few decades (S. P. Brooks, 2003) has enabled a much larger pool of psychologists than ever before to apply Bayesian methods in their work (Andrews & Baguley, 2013; van de Schoot et al., 2017). Thanks to tutorial books and papers targeted at psychologists (e.g., Rouder et al., 2009; Kruschke, 2014), it is no longer rare to see Bayesian hypothesis tests and Bayesian linear models reported in psychological research. However, Bayesian data analysis is not the only approach to using Bayesian methods in psychological research: In this paper, we will discuss a different approach, *Bayesian cognitive*

modeling, in which Bayesian methods are used to implement cognitive process models (Lee & Wagenmakers, 2013; not to be confused with Bayesian models of mind[1]). Process models are increasingly being used (Jarecki et al., 2020) to provide formal, testable accounts of the possible psychological mechanisms underlying observed behavior (Navarro, 2021). Using hierarchical Bayesian methods for cognitive modeling confers many benefits, such as the ability to quantify uncertainty in parameter estimates while simultaneously accounting for individual differences and other meaningful structures directly in a model (Lee, 2011). Bayesian cognitive modeling is a principled and coherent approach to quantitative evaluation of psychological theory.

While using Bayesian methods for cognitive modeling had long been the province of mathematical psychologists, as it required comfort with mathematical statistics and statistical programming (Gilks et al., 1995; Gelman et al., 2013; for an example, see Rouder & Lu, 2005), this has changed with the maturation of software that automates the Markov chain Monte Carlo methods (MCMC; S. Brooks et al., 2011) most popularly used for Bayesian model fitting[2] (such as JAGS: Plummer, 2003; and Stan: Carpenter et al., 2017) and software that likewise automates Bayesian model specification (for linear models: Bürkner, 2017; and for select cognitive models: Ahn et al., 2017). These developments have made Bayesian cognitive modeling accessible to psychologists across subfields, including cognitive psychologists (e.g., Donkin et al., 2016; Navarro et al., 2016; Westfall & Lee, 2021), cognitive neuroscientists (e.g., Frank et al., 2015; Nunez et al., 2019; Peters & D'Esposito, 2020), clinical psychologists (e.g., Haines et al., 2020; Brown et al., 2021; Lasagna et al., 2022), and social psychologists (e.g., Pleskac et al., 2018; Golubickis et al., 2018; Schaper et al., 2019).

However, while linear statistical models (e.g., multilevel regression models) can be relatively easily implemented in a Bayesian framework, Bayesian implementations of cogni-

[1]It is important to note that Bayesian cognitive modeling is also distinct from a Bayesian theory of mind approach (which is sometimes termed "Bayes in the head"; e.g., Griffiths et al., 2008). Bayesian models of mind view Bayes' theorem as a cognitive mechanism in and of itself, that is capable of capturing how one might rationally update their beliefs about the world in light of their experiences. In contrast, Bayesian cognitive models are used to express a wide variety of other candidate cognitive mechanisms and processes (which are not required to be rational — and as such, models may even be explicitly designed to capture non-optimal behavioral patterns; e.g., Busemeyer et al., 2011); Bayesian methods are only used as the technique for parameter estimation.

[2]Although other state-of-the-art methods including variational Bayes (Blei et al., 2017; Galdo et al., 2020) and Sequential Monte Carlo (Dai et al., 2022; Gunawan et al., 2020) are also used, MCMC methods remain the most widely used family of algorithms for Bayesian cognitive model fitting.

We would like to thank Michael Lee for his generous comments, which helped to substantively improve the manuscript. We also wish to thank our reviewers for their perspective and comments, as well as Aspen Yoo, Amy Zou, Milena Rmus, Gaia Molinaro, and Soobin Hong for their comments on an earlier draft. This work was supported by NIH grant #R01MH119383.

The code has been made publicly available as part of the `matstanlib` library and can be accessed at https://github.com/baribault/matstanlib.

Beth Baribault https://orcid.org/0000-0001-7370-2183

Correspondence concerning this article should be addressed to Beth Baribault, Helen Wills Neuroscience Institute, Department of Psychology, University of California, Berkeley, 2121 Berkeley Way, Berkeley, CA 94720. E-mail: baribault@berkeley.edu

tive models tend to require more careful testing and tweaking before they may be confidently applied to data. This is because most Bayesian cognitive models have characteristics which are known to pose challenges for Bayesian model fitting, even for the dynamic Hamiltonian Monte Carlo algorithms (e.g., Hoffman & Gelman, 2014) that we restrict our attention to here. In order to quantitatively express cognitive mechanisms, Bayesian cognitive models often require complicated, nonlinear likelihood functions, and to incorporate relevant domain knowledge, non-conjugate priors over restricted domains are often used. In some families of cognitive models, correlations among model parameters are closer to the rule than the exception (Turner et al., 2013; Krefeld-Schwalb et al., 2022). Hierarchical model structures are common, if not universally encouraged (Boehm et al., 2018), as they allow for the simultaneous account of group- and individual-level effects (among other meaningful dependencies; Lee, 2011; Scheibehenne & Pachur, 2015). These features all tend to produce posterior geometries that are challenging for MCMC algorithms to navigate, and therefore heighten the risk of computational failures. If active steps are not taken to conduct *computational checks* for such failures, as well as *consistency checks* of other key assumptions about model behavior, then any inference based on model output risks being fundamentally flawed.

As such, the ability to detect, diagnose, and remedy problems — via procedures which we collectively call *troubleshooting* — is essential for practitioners of Bayesian cognitive modeling. Unfortunately, troubleshooting seems to be a blind spot in the didactic literature on Bayesian methods aimed at cognitive scientists. Of these past tutorial papers and books, those that introduce the core concepts of Bayesian data analysis often do not cover Bayesian cognitive modeling (e.g., Etz & Vandekerckhove, 2018; Kruschke, 2014). Those that focus on Bayesian cognitive model implementation, design, and development (e.g., Rouder & Lu, 2005; Lee, 2008; Shiffrin et al., 2008; Vanpaemel, 2010; Lee & Wagenmakers, 2013; Heathcote et al., 2015; Annis & Palmeri, 2018; Lee, 2018; Heathcote et al., 2019; Schad et al., 2021; Greene & Rhodes, 2022) tend to underspecify the model-checking steps required before a model may be used for inference.[3] This is complicated by the fact that model-checking techniques have evolved and improved over time, such that failure modes that were not previously able to be detected may now be reliably exposed.

Specifically, recent advances in Bayesian statistical practice have amended and broadened the suite of diagnostic checks of Bayesian model output that are deemed necessary. Consider for a moment that the most familiar convergence diagnostic, $\hat{R} \leq 1.1$, has been considered standard since the 1990s (Gelman & Rubin, 1992; Gelman et al., 1995). In just the past couple of years, the computation of $\hat{R}$ has been made markedly more sensitive, such that $\hat{R}$ values must now meet the far more stringent criterion of being ≤ 1.01 (Vehtari et al., 2021; Gelman et al., 2020). As we will discuss in detail here, recent developments have mandated other significant changes, the collective effect of which is that some previously sufficient model output will now fail convergence checks. Other changes are a result of the collective shift away from previously preferred MCMC methods, including Gibbs sampling (via JAGS: Plummer, 2003), toward a newer, more efficient, and more robust method, Hamiltonian Monte Carlo (via Stan: Carpenter et al., 2017; and PyMC: Salvatier et al.,

[3]Some of these sources also use linear models as their guiding example, rather than cognitive process models. It is important to distinguish between the two, as some techniques for the testing and development of linear models are of limited use or importance for cognitive models, and vice versa.

2016). The increasingly popular class of Hamiltonian Monte Carlo (HMC) sampling algorithms (Duane et al., 1987; Neal, 2011), including the No-U-Turn Sampler (NUTS; Hoffman & Gelman, 2014) and the advanced dynamic HMC sampler implemented in Stan (Stan Development Team, 2022) — which we collectively call HMC/NUTS, and focus on exclusively here — requires that multiple additional diagnostic quantities are checked as a matter of course (e.g., BFMI; Betancourt, 2016). Despite the sharp rise in use of Bayesian cognitive modeling and HMC/NUTS, these notable changes to the core practices of Bayesian model fitting are still somewhat unfamiliar in the Bayesian cognitive modeling literature.

As such, the primary purpose of this tutorial is to present a current, thorough treatment of the computational and model consistency checks required for proper use of Bayesian cognitive models fit via HMC/NUTS sampling (see Figure 1), including clear guidance on what to do when model output fails one or more checks. Because some of the topics we cover here can seem arcane to psychological researchers who are not also Bayesian statistical researchers, we have taken care to provide conceptual explanations of all procedures, and to make connections to principles from cognitive science, if not actively demonstrating by example, where possible. To this end, we begin with an overview of the Bayesian cognitive modeling approach so as to build the conceptual groundwork that is prerequisite for successful troubleshooting. Then, we explain how to check for computational and other problems using computational diagnostics, consistency checks, and diagnostic plots, while offering remedies for simpler issues along the way. Next, we explain how thornier issues related to parameterization and posterior geometry can be elucidated through the use of additional plots and other techniques. Throughout, we offer guidance on how to better utilize and triage among the many techniques for identifying the exact nature of the problem, and how characterization this can naturally lead to solutions. Finally, we review more application-dependent methods, including posterior predictive checks, that check how capable and useful a model is (or is not) for a given research context.

Troubleshooting can be an exceptionally challenging stage within a larger Bayesian cognitive modeling workflow, as it is often an iterative, looping process through a sequence of steps that are rarely done in sequence. As the selection of techniques is part of troubleshooting, we make explicit note of many useful pathways among the various steps of troubleshooting. We also make explicit note what computations, plots, and procedures are expected to be manually programmed versus automated by publicly-available code libraries that support Bayesian model evaluation. Using these *support libraries*, as we will call them, is now both expected and encouraged (Gabry et al., 2019). Currently, the most widely-used support libraries are the `bayesplot` package in R (Gabry & Mahr, 2021), the `ArviZ` package in Python (Kumar et al., 2019), and the `matstanlib` library in MATLAB (which includes the scripts and Stan files for our example models; Baribault, 2021). In the Appendix, we offer a brief overview of how to use these libraries, and include Table A1 for quick reference of the commands needed in each library to automate many of the troubleshooting techniques recommended in this tutorial.

Above all, it is the conceptual principles and processes of troubleshooting that we seek to emphasize here. It is our intent that this paper will serve as a general reference for how to detect, diagnose, and correct many of the problems that are most frequently encountered in the construction and development of Bayesian cognitive models, particularly when they are implemented with state-of-the-art HMC/NUTS sampling methods.

Bayesian cognitive modeling

We begin with a review of the Bayesian cognitive modeling approach, with an emphasis on the specific Bayesian techniques and methods that are currently most commonly used in Bayesian cognitive model fitting. While we assume a general familiarity with Bayesian principles (see Etz et al., 2018 for a first introduction, or Gelman et al., 2013 for a deeper treatment) and computational cognitive modeling (e.g., Wilson & Collins, 2019; Farrell & Lewandowsky, 2018), we include this overview to establish conceptual ideas and terminology that we rely on throughout the tutorial.

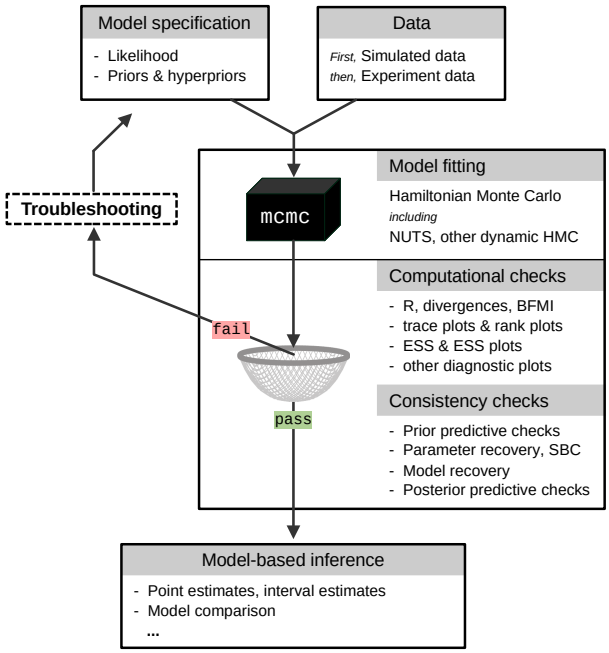

Figure 1

An abbreviated representation of the Bayesian cognitive modeling workflow that emphasizes the subset of steps most relevant to troubleshooting. Model output that does not pass through the filter (representing the requisite computational and consistency checks) must be rejected. A troubleshooting process should be used to improve the model specification such that the output might ultimately pass through all checks. Then, and only then, may the Bayesian cognitive model output be used as the basis for inference. (Note that the model-checking techniques listed in the figure need not be performed in the order in which they appear; prior predictive checks, for example, would ideally be performed before model fitting; see Gelman et al., 2020 for an exhaustive, ordered list.)

The Bayesian framework

All Bayesian analysis derives from Bayes' theorem:

$$p(\theta|x) = \frac{p(x|\theta)\, p(\theta)}{\int p(x|\theta')\, p(\theta')\, d\theta'}$$

and Bayesian cognitive modeling, of course, is no exception. Bayes' theorem tells us how the prior probability, $p(\theta)$, of an unobserved parameter or set of parameters, θ, and the likelihood, $p(x|\theta)$, of the observed data, x, may be used to derive the posterior probability, $p(\theta|x)$, of the parameters in light of the data.

In a Bayesian cognitive model, the likelihood is specifically used to express the cognitive process or mechanism theorized to have produced the behavioral data. Cognitive model likelihoods are typically nonlinear and may be rather complex, as in an evidence accumulator model of response times (Ratcliff & McKoon, 2008; Vandekerckhove et al., 2008), a cumulative prospect theory-based model of decisions (Nilsson et al., 2011), or a reinforcement learning model of action selection (Sutton & Barto, 2018; Dearden et al., 1998). This assumed distribution of the data is conditional on the unobserved parameters, which in a cognitive model will have meaningful psychological interpretations as they are intended to capture one aspect or shape one dynamic of the cognitive process expressed in the model. For example, in the respective aforementioned models, we interpret ν as the speed of evidence accumulation, λ as the relative weighting of losses and gains, and α as the learning rate. Bayesian analysis requires that each parameter has an associated prior distribution, which should be defined over all conceivably possible values. In a cognitive model, these priors are often seen as an opportunity to incorporate domain knowledge relevant to each of the parameterized cognitive dynamics. In mature subfields, this knowledge can be considerable, especially when a particular model has been long been used successfully (Tran et al., 2021). (For a worked example of how to use domain expertise to support prior elicitation for cognitive models, see Vanpaemel, 2010.)

The various dependencies among the parameters and data is called the *structure* of the model. It is common for Bayesian cognitive models to incorporate *hierarchical* structure in order to instantiate theoretically meaningful dependencies among the parameters and/or data (Lee, 2011; Scheibehenne & Pachur, 2015). For example, hierarchical structure may be used to simultaneously account for data from multiple participants, groups, conditions, and so on, at advanced levels of abstraction (Lee, 2011). A hierarchical extension of a Bayesian cognitive model over participants might specify that each participant is allowed a unique set of parameter values (e.g., in a reinforcement learning model, their own learning rate, α), but all *instances* of each parameter (e.g., all participants' α parameters) are assumed to be drawn from a common group-level *hyperprior* distribution. (In other words, Bayesian cognitive models can be multilevel models.) This example hierarchical extension confers dual benefits of sharing informative power, which can be helpful in small-data situations, and regularizing parameter estimates across participants, which engenders more reliable estimates (Gelman & Hill, 2006; Scheibehenne & Pachur, 2015; Katahira, 2016). Hierarchy is also commonly used to enable Bayesian cognitive models to capture how multiple cognitive processes simultaneously contribute to behavior, or how the same cognitive process is responsible for performance in multiple tasks. In the context of troubleshooting, it is important to keep the structure of the model in mind — especially hierarchical prior structure — as it is often necessary context for diagnostic interpretation.

Taken together, the likelihood and all priors compose a Bayesian *model specification*. Because deriving the exact posterior analytically is feasible only for the simplest of models, virtually all Bayesian cognitive model fitting relies on methods to *approximate* the joint posterior distribution, including MCMC sampling algorithms. MCMC algorithms allow for

samples to be drawn from the joint posterior in such a way that each possible value of a given parameter should be drawn with a probability *proportional to* its posterior density:

$$p(\theta|x) \propto p(x|\theta)p(\theta)$$

If an infinite number of samples were to be collected, one would be guaranteed to recover the true posterior (among other mathematical guarantees; Gilks et al., 1995; S. Brooks et al., 2011; Gelman et al., 2013). The finite number of posterior samples collected in practice serve to approximate the true posterior in much the same way that one might use a histogram of collected scores as an approximation of the true distribution of scores in the population.

Ultimately, the goal is to use these posterior samples as the basis for inference about the nature of the cognitive process or processes the model was designed to capture. A wide variety of posterior estimates may be computed, but most frequently, the mean or median of the posterior samples for a given parameter will serve as its *point estimate*, and a *credible interval*, such as a 90% highest-density interval, will be used to express the posterior *uncertainty* in that estimate. The exact estimates, analyses, and tests that are of greatest interest will depend on the capabilities of your model as well as your research goals.

An example Bayesian cognitive model

As an example of a Bayesian cognitive model, we consider a hierarchical Bayesian implementation of a reinforcement learning (RL) model (that we will reference throughout the paper as we explain various troubleshooting techniques). The model will be applied to simulated data, assuming an experimental design in which each of 30 "participants" completes four blocks of a probabilistic three-armed bandit task. In each block, the simulated participant sees the bandit stimuli 20 times and must learn from the results of their choices, over time, which arm is most likely to give a point reward (see Figure 2a). Across the trials within each block, the participant should more and more often select the bandit with the highest reward probability. If this choice of bandit is considered *correct*, then we will expect accuracy to start at chance ($\frac{1}{3}$), rise, and asymptote, in a classic learning curve (as seen in Figure 2b).

Our reinforcement learning model (Sutton & Barto, 2018) captures participants' learning process by allowing for each stimulus to be assigned a separate value. At the start of each block, we assume that every participant begins by assigning each stimulus the same starting value, such as $Q_0 = [0, 0, 0]$. Then, on every trial, t, the values, Q, are scaled by an inverse temperature parameter, β, and run through a softmax function to determine the probability of selecting each stimulus, π. The participant then makes a choice, c, according to those probabilities:

$$\pi_{t\,i} = \frac{e^{\beta Q_{t\,i}}}{\sum_{j=1}^{3} e^{\beta Q_{t\,j}}} \quad \forall i$$

$$c_t \sim \text{Categorical}(\pi_t)$$

The difference between the reward resulting from that choice, r, and the current value of the chosen stimulus constitutes a prediction error, δ. The prediction error is used to update

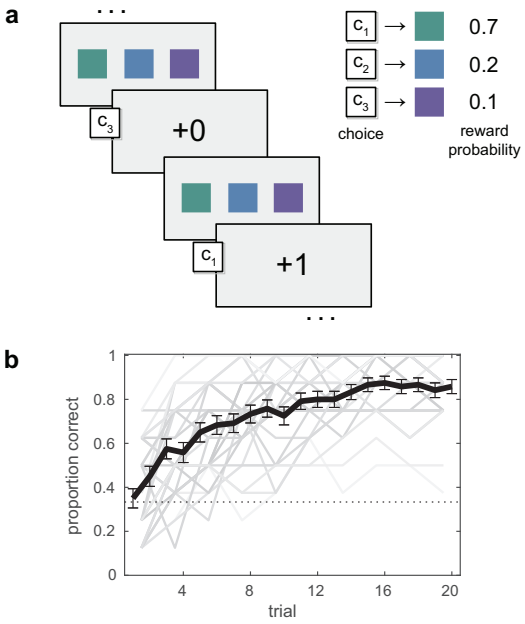

Figure 2

Many figures throughout this paper present output from our hierarchical Bayesian implementation of a classic delta-rule reinforcement learning model (detailed in text). When behavioral data for a probabilistic 3-armed bandit task (a) was simulated according to the model specification, characteristic learning curves (b) are seen in the simulated data at the group level (thick black line) and for individual subjects (lighter gray lines), which are all above chance (dotted line) by the final trials.

the chosen stimulus' value according to a learning rate, α:

$$\delta_t = r_t - Q_t(c_t)$$
$$Q_t(c_t) = Q_t(c_t) + \alpha\delta_t$$

Finally, to capture how participants may forget over time, all Q-values are subject to decay with rate ϕ before the next trial begins:

$$Q_{t+1} = Q_t + \phi(Q_0 - Q_t)$$

Together, the action selection, value updating, and forgetting mechanisms describe the *cognitive process* of learning over time. As the exact dynamics of the process will be unique to each individual, each participant p is allowed to have a different β, α, and ϕ. However, we also assume that all participants come from a group[4] that shares common cognitive processes. To express this knowledge in the model, we incorporate a hierarchy over participants,

[4]Humans.

and we set participant-level priors for each parameter:

$$\beta_p \sim \text{Normal}(\mu^\beta, \sigma^\beta)_{\mathcal{T}[0,\ ]}$$
$$\alpha_p \sim \text{Normal}(\mu^\alpha, \sigma^\alpha)_{\mathcal{T}[0,1]}$$
$$\phi_p \sim \text{Normal}(\mu^\phi, \sigma^\phi)_{\mathcal{T}[0,1]}$$

that are dependent on group-level hyperparameters, with associated hyperpriors:

$$\mu^\beta \sim \text{Normal}(10, 5)_{\mathcal{T}[0,\ ]}$$
$$\sigma^\beta \sim \text{Normal}(0, 5)_{\mathcal{T}[0,\ ]}$$
$$\mu^\alpha \sim \text{Uniform}(0, 1)$$
$$\sigma^\alpha \sim \text{Normal}(0, 0.5)_{\mathcal{T}[0,1]}$$
$$\mu^\phi \sim \text{Uniform}(0, 1)$$
$$\sigma^\phi \sim \text{Normal}(0, 0.5)_{\mathcal{T}[0,1]}$$

(The subscript $\mathcal{T}$ indicates a *truncation* of the distribution to between the specified bounds or bound.)

We selected this simple delta-rule learning model because it has many of the features common to Bayesian cognitive models that are known to pose problems for MCMC algorithms, such as the aforementioned nonlinear and complicated likelihood and the hierarchical model structure. In addition, its parameters require restricted ranges ($\alpha, \phi \in [0, 1]$, $\beta \in \mathbb{R}^+$, exclusive of 0), are decidedly not normally-distributed (e.g., the empirical distribution for β is positively skewed), and are well-known to be correlated (in some reinforcement learning experiments and models, though certainly not all). As such, this popular cognitive model presents good opportunities for demonstrating the principles of troubleshooting.

In fact, while the model specification above may appear sufficient at first glance, it will reliably fail most of the required computational and consistency checks (as described in the next section). As a supplement to this tutorial, we include a MATLAB script, `example_RL.m`, available in the `examples` folder of the `matstanlib` library, that will specify, apply, and troubleshoot this model. The script will run both this initial, flawed version of the model and a final, improved version of the model. For readers who are not MATLAB users but wish to follow along, we also include standalone files containing Stan code for each version of the model (as `RL_broken.stan` and `RL_fixed.stan`, respectively) — but the model specification detailed above is sufficient to implement the model in PyMC or any other HMC/NUTS sampling package.

Running the RL model script and another example script, `example_funnel.m`, will collectively reproduce many of the figure panels in the remainder of this paper (all of which present real Bayesian cognitive model output).

A brief introduction to sampling algorithms

By *fitting* or *running* a model, we specifically mean using MCMC sampling to estimate the joint posterior of the model (see Van Ravenzwaaij et al., 2018, for an accessible introduction to MCMC sampling that emphasizes many core principles). In practice, this entails submitting the model specification and the data — whether experimentally collected

or simulated — to software designed to automate MCMC sampling, via an interface specific to the programming language used.

For the remainder of this tutorial, our discussion of MCMC sampling will implicitly assume the use of HMC/NUTS sampling, as currently HMC/NUTS algorithms represent the state-of-the-art in MCMC methods (Gelman et al., 2020), and are immensely popular due to their automation in the well-supported Stan and PyMC software packages. However, we wish to note that much of this conceptual introduction — and a majority of the troubleshooting procedures we discuss beginning in the next section — will also be applicable to other MCMC sampling algorithms (for overviews, see Robert & Casella, 2011; Van Ravenzwaaij et al., 2018). It is important to build intuition for how posterior samples are generated in order to understand the computational diagnostics, as failed diagnostic checks will motivate many of the troubleshooting techniques we recommend.

To begin, the sampler is initialized at a random point in the parameter space as defined by the model. This place and every subsequent place the sampler visits in the posterior parameter space is recorded as a *sample* from the joint posterior. From the random initial position, the sampling algorithm is used to compute a trajectory within the parameter space from the current position to the next, and again from there to another position, and so on, until a pre-specified number of joint posterior samples have been collected. These samples, in order, are called a *chain*. Because the first few samples or *iterations* in the chain will usually be more representative of the initializing value than of the true posterior (called the *target distribution*), the first handful or more of iterations in a chain are discarded. In modern sampling software, this *warmup* period (or *burn-in*, in older sources) is also used for adaptation of the sampler itself. For example, roughly how big of a step in the joint parameter space is taken with each successive iteration is a tuning parameter that is adjusted during warmup (for a review of HMC/NUTS sampler dynamics, see Betancourt, 2018).

In practice, multiple chains are run simultaneously, because without multiple chains we cannot perform some of the computational checks required to assess the quality of the sampling (Gelman & Rubin, 1991) — in addition to saving precious time. A good expectation for Bayesian cognitive model applications is to collect at least 2000 total iterations per each of four chains, with at least the first 500 apportioned for warmup, and the remaining 1500 kept and used for inference.[5] Ultimately, the warmup period should be long enough that the chains have *converged* (or agreed) on a *stationary* distribution, and the subsequent period of collecting kept iterations should be sufficient both to pass the computational checks and to support all planned uses of the marginal posteriors for inference. However, when one is first beginning to work with a model, we recommend starting by collecting only 50 warmup and 50 kept iterations, just to ensure the model runs. Then, we recommend observing whether shorter runs of the model, such as 150 warmup and 500 kept iterations,

[5]If you are familiar with Gibbs sampling, then you may notice that the recommended numbers of kept (and burned) iterations are far lower than than the number of iterations recommended for Gibbs sampling. With HMC/NUTS, fewer posterior samples are required due to the the much higher efficiency per iteration, especially with respect to the ability of HMC/NUTS to move throughout the kinds of correlated and high-dimensional parameter spaces that are common in Bayesian cognitive modeling (Turner et al., 2013), which greatly hinder Gibbs sampling (but not HMC/NUTS; Neal, 2011; Hoffman & Gelman, 2014). As a result, where Gibbs sampling via JAGS might require 10,000–100,000 samples, HMC/NUTS sampling via Stan or PyMC might require only 1000–2000 samples. You may also notice that we do not mention thinning samples. Thinning samples is no longer recommended (Link & Eaton, 2012).

might reveal problems. These abbreviated runs will allow one to begin the iterative process of troubleshooting (which requires repeated model runs) without spending as much time waiting for failures (Gelman et al., 2020). A high-level view of the Bayesian cognitive modeling approach that emphases the iterative nature of model testing and troubleshooting is presented in Figure 1.

Sampler output

The output of a Bayesian cognitive model is a collection of samples from the marginal posterior distribution for every parameter of the model, as well as various diagnostic quantities from the computation the sampler used to generate each iteration in each chain. One should be mindful that which chain a sample was collected in, and the order in which the samples were collected within each chain is crucially important information. As such, one should never engage a `'permute'` option during sample extraction: Scrambling the iteration order and chain identity will prevent the use of multiple required model-checking diagnostics, and as such, will make your output unusable.

In the Appendix, we explain in more detail how the libraries mentioned earlier — including `bayesplot` in R, `ArviZ` in Python, and `matstanlib` in MATLAB — are not only useful for extracting samples, but are critical to facilitate many of the troubleshooting procedures we now describe. We recommend using Table A1 to check the command needed to facilitate each troubleshooting procedure we discuss as you work through the rest of the tutorial.

Detecting problems

After the Bayesian cognitive model finishes running and posterior samples and sampler diagnostics have been extracted, we first must check whether we can detect any problems in the output, and if so, initiate a troubleshooting process to identify a potential underlying cause. For a Bayesian cognitive model, these checks will likely be performed many times. At an absolute minimum, they should be performed twice: first, after applying the model to simulated data, and again after applying the model to experimentally-collected data. The model will tend to be run many more times as the troubleshooting process is *iterative*: Each time a check fails, one should investigate the output, tweak the model setup, and run the model again.

In the *simulation study*, data should be simulated according to the data distribution in the model specification, using known or *true* parameter values. While these values may be hand-selected, it is better to randomly generate the true values directly from the prior distributions, and to let new values be drawn each time the simulation study script is called. The simulated data should also mimic the experimental design as closely as possible (e.g., a similar number of participants, similar stimulus sequences, etc.). Once the troubleshooting process is complete for the simulation study, we can progress to troubleshooting the model's performance with experimentally-collected data (as needed).

In both applications, problems can be detected using the recommended suite of computational checks and consistency checks to probe, respectively, whether the HMC/NUTS sampling and the model itself have functioned as intended. The *computational checks* are

primarily concerned with evaluating the computational diagnostics for HMC/NUTS sampling, including $\hat{R}$, divergences, BFMI, and ESS (as discussed in detail below). Other quantities, such as R^* (a multivariate convergence metric for the *entire* model; Lambert & Vehtari, 2022) and $\hat{k}$ (which is computed as part of Bayesian cross-validation; Vehtari et al., 2017), are diagnostically useful, but as they are not currently recommended default diagnostics, and some require a deeper technical understanding, we consider them to be beyond the scope of the present paper. The assessments that we call *consistency checks* are a necessary complement to the computational checks, as they are designed to evaluate key assumptions and expectations about model behavior, including whether the model might be misspecified given the research context. These computational and consistency checks (or collectively, *diagnostic checks*) all have associated visualizations, which we call *diagnostic plots*.

Each diagnostic check presents a different opportunity for troubleshooting of a Bayesian cognitive model, as each is geared toward the detection of different types of problems. It is intuitive to see these diagnostic checks in terms of the questions they are most helpful in answering. These questions are:

1. Is there any evidence that the chains disagree about any of the marginal posteriors?
2. Is there any evidence that the posterior distribution was not fully explored?
3. Is there any evidence that sampling was not efficient enough to support good posterior estimates?
4. Is the model failing to generate coherent parameter estimates?
5. Is the data the model expects to encounter unreasonable or otherwise inconsistent with my domain expertise?

If the diagnostics suggest that the answer to any of these questions is "yes", then there is a problem with the model setup that absolutely must be corrected. By understanding what each diagnostic metric, plot, and procedure is designed to measure or assess, one can begin to identify the problem and, accordingly, a solution.

Computational checks

Convergence and divergence

The most familiar MCMC diagnostic is $\hat{R}$ (which is sometimes called the "Gelman-Rubin statistic" in older sources; Gelman & Rubin, 1992). If all of the chains have converged on the target distribution, then the chains should agree so strongly that they are functionally identical, in which case $\hat{R}$ will be close to 1. For each parameter, the chains should specifically agree with respect to both the location and spread of the marginal posterior distribution. Furthermore, there should be no remaining influence of any chain's starting value and, over the full range of the kept iterations, all chains should appear stationary. In this ideal case, $\hat{R}$ will be exactly 1; a value of $\hat{R}$ that is meaningfully greater than 1 suggests that the chains have failed to converge (Gelman et al., 2013; Vehtari et al., 2021).

Understanding how $\hat{R}$ is computed can build intuition for what kinds of problems can be detected by high $\hat{R}$ (as we review below). After splitting the chains (such that the

first and second half might temporarily be considered as separate chains), the base $\hat{R}$ computation essentially compares the between-chain variance to the within-chain variance. $\hat{R}$ will be high if the chains are not *mixing* (meaning failing to sample from similar ranges of values), or if any of the chains are not *stationary* (meaning that a notably different range of values is sampled over time), as in both cases the between-chain variance will be disproportionately high (Gelman et al., 2013). In the most recent reformulation of $\hat{R}$ (Vehtari et al., 2021), the samples are converted to ranks and (approximately) inverse normal transformed before the base $\hat{R}$ formula is applied. The combined effect of this transformation and a few other adjustments is that the new and improved $\hat{R}$ is simultaneously more robust (to monotone transformations) and more sensitive (to some instances of poor mixing that were not able to be detected by previous formulations; Vehtari et al., 2021). (In other words, the Type I and Type II error rates are *both* lower for convergence checks with the new formulation of $\hat{R}$; when viewed in this light, it is less surprising that the criterion has been tightened from 1.1 to 1.01.)

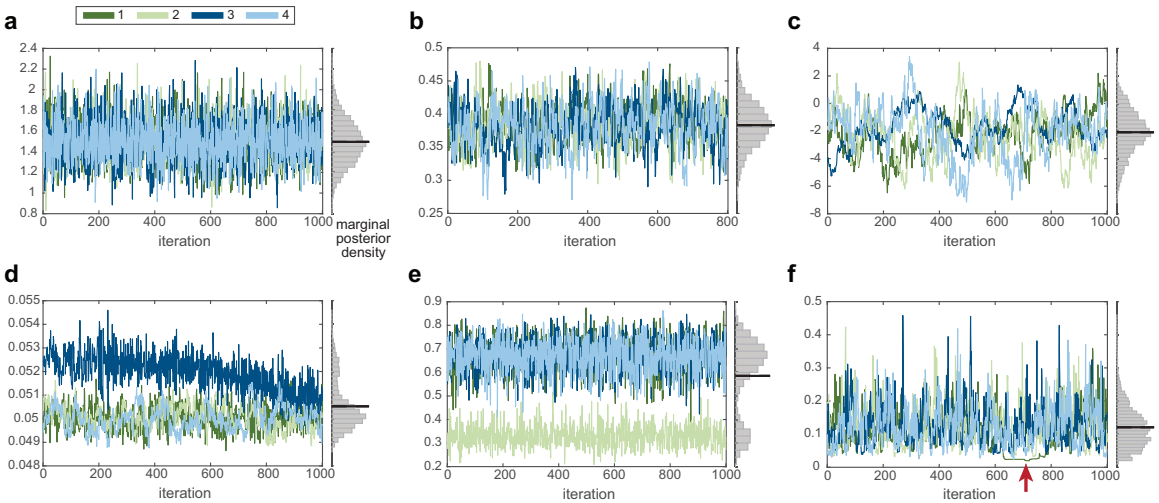

Figure 3

Trace plots can be used to visualize chain (dis)agreement, and support troubleshooting of convergence issues signaled by high $\hat{R}$. Only the chain traces in (a) and (b) are acceptable; the traces in (c–f) each have a common yet serious problem. Extreme autocorrelation (c), drift (d), label-switching (e), and sticking (f), all tend to cause high $\hat{R}$, and all are unacceptable. Troubleshooting techniques are necessary to identify the root of these problems.

As such, $\hat{R}$ may be interpreted as the degree to which the chains disagree, and a value of $\hat{R} \leq 1.01$ is required for every instance of every parameter in the model (Vehtari et al., 2021; Stan Development Team, 2022). While high $\hat{R}$ values do not suggest a remedy in and of themselves, with the assistance of *trace plots*, we can begin to understand why the chain disagreement flagged by $\hat{R}$ might be occurring. Trace plots visualize chain behavior by plotting the sequence of parameter values sampled at each iteration in each chain, in order, as a line (called the chain *trace*). In some support libraries, it is possible to include a histogram of the samples across all chains alongside the chain traces; this summary representation of the marginal posterior is often helpful to interpret the trace plot. Some

classic examples of ideal, acceptable, and unacceptable chain behavior are presented in Figure 3.

In an ideal situation where $\hat{R}$ is close to 1 and the chains are stationary and mixing well, the chain traces will tend to appear in a trace plot as they do in Figure 3a; this appearance has been said to resemble a fuzzy caterpillar or the flower of a bottlebrush plant. More commonly in Bayesian cognitive models, there will be some degree of *autocorrelation* within each chain, meaning a tendency for similar values to be sampled in successive iterations versus more distant iterations (as seen in Figure 3b). Unless $\hat{R}$ or other diagnostics have failed their check (or effective sample size, as discussed below, is undesirably low), a mild amount of autocorrelation is not of concern, and the model may simply be run for more kept iterations (Link & Eaton, 2012). Extreme autocorrelation, on the other hand (as in Figure 3c), should be investigated further. This behavior can indicate that some feature of the parameter space defined by the model is — directly or indirectly — making it difficult for the chains to move efficiently. Effective sample size plots (discussed in the next subsection) will likely be of help in searching for the specific underlying issue.

Another unacceptable chain behavior that may be recognized in a trace plot is chain *drift* (Figure 3d). This may occur if the starting points of one or more chains are still exerting an influence on the sampled values. Alternatively, this may occur if the drifting chain was in a local posterior maximum, and is (somewhat slowly) transitioning to a higher-density region of the posterior space. Regardless of cause, $\hat{R}$ will be high in cases of drift because one (or more) of the chains is not stationary. The first remedy to try in this situation is to run the model again with a longer warmup period (e.g., twice the number of warmup iterations) to give the chains more time to find a stationary distribution.

A more challenging pattern to resolve is when each individual chain is stationary and moving well, but collectively the chains fail to mix, as they disagree on the location of the posterior distribution (as in Figure 3e). This kind of confident disagreement is common to see when a cognitive model is insufficiently identified. For example, in a latent mixture model, behavior may be modeled as a weighted combination of two or more cognitive processes. If these component processes predict similar behavior, then the mixture parameter may only be weakly identified, and different chains may settle on different values of the mixture proportion (Jasra et al., 2005). This phenomenon, known as *label-switching*, can also occur in models where two parameters are directly multiplied, but the priors and data are insufficient to identify more than the parameters' product. In both cases, a first remedy is to make the priors more informative. However, if domain knowledge is not available or appropriate to incorporate, and the component processes or parameters cannot be distinguished in another way, then the experiment in which the behavioral data was collected may simply not be sufficient to distinguish the processes intended to be captured by model.

The last pattern that may be signaled by high $\hat{R}$ is when a chain will sample the same value over and over for an extended period of time (as in Figure 3f). When this *sticking* behavior occurs, it is often a result of the sampler trying and failing to reach a nearby area in the joint parameter space. A common scenario in which this might be observed is in a hierarchical model, where a chain for the hyper-level dispersion parameter gets stuck near(-ish) to 0 (due to a hierarchical funnel, as we will fully explain in the *Posterior geometry* subsection; Betancourt & Girolami, 2015). Most of the time, when sticking behavior is seen in a trace plot for one parameter, it will be seen for others as well. Unfortunately, this

can sometimes make it challenging to identify the true culprit, but if divergences (which we discuss in a moment) occur whenever the chain sticks (as they often do), the divergences will likely be more useful to investigate.

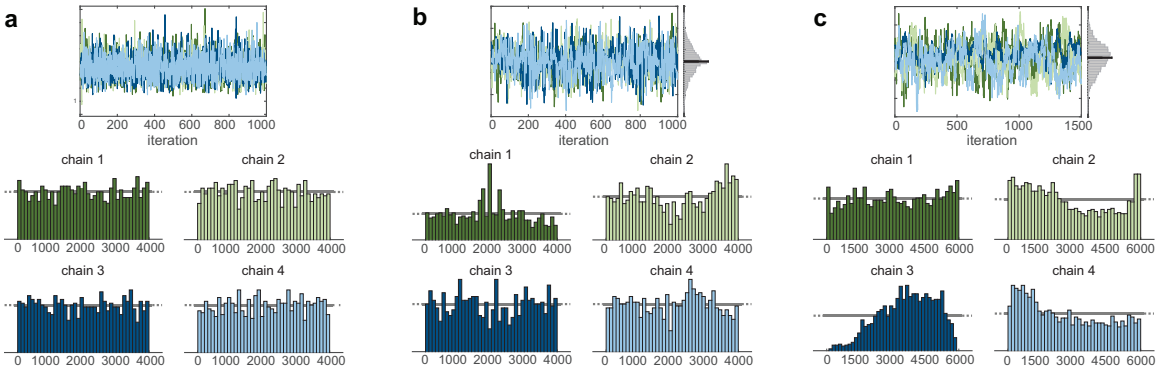

Figure 4

Rank plots are a new way to visualize chain (dis)agreement, and support troubleshooting of convergence issues signaled by high $\hat{R}$. In some cases, rank plots can expose problems that were not visible in a classic trace plot. While all three trace plots look acceptable, the corresponding rank plots of the same chains reveal that this impression is only genuine for (a), where the ranks for all chains appear roughly uniformly distributed. In (b), the sticking behavior is hidden under the bulk of the chain traces, but is readily apparent from the peak in chain 1's rank plot. Similarly, the lower variance of one chain in (c) is not discernible in trace plot, but the skewed rank plot for chain 3 clearly suggests that this chain is sampling a restricted range of values relative to the others.

In some situations, trace plots can be exceptionally difficult to interpret. For example, when a very high number of samples have been collected, cramming the long traces into a standard-sized plot can hide some problematic chain behaviors, and the traces will spuriously appear good (see Figure 4). Trace plots can also be difficult to judge when distributions are highly skewed and/or fat-tailed, in which case the stereotypical bottlebrush pattern (Figure 3a) would not be expected even when the chains have converged and are mixing well (for examples of how these chain traces can look, see Vehtari et al., 2021). For these reasons, it is now recommended to use *rank plots* in addition to, if not in place of, trace plots, so that any differences in the values being sampled by each chain can be more reliably recognized (Vehtari et al., 2021). Rank plots are a new diagnostic plot that is generated by ranking the samples pooled across all chains, then presenting a histogram of the ranks originating from each chain separately. If the chains have perfectly converged, then the distribution of ranks for each chain should approximate a uniform distribution (as in Figure 4a). Deviations from uniformity can indicate a wider variety of convergence issues. Two examples of problems that are more easily detected in rank plots are presented in Figure 4b and 4c (for additional examples, see Vehtari et al., 2021).

A relatively recent addition to the suite of computational checks is *divergences* (or *divergent transitions*), which are specific to the HMC family of algorithms. For each posterior sample generated through HMC/NUTS sampling, whether the numerical trajectory

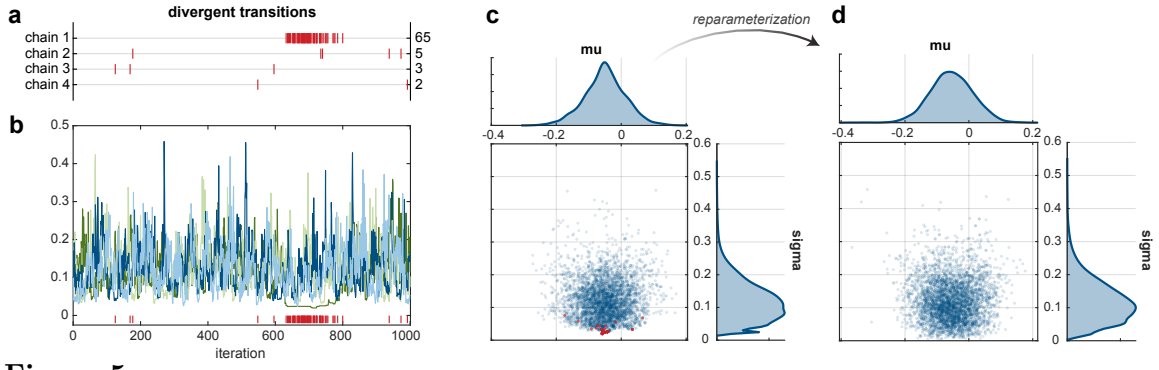

Figure 5

Diagnostic plots for troubleshooting divergences. (a) In these rug plots, a red tick marks each iteration within a given chain where a divergence occurred. (b) It is useful to include a similar plot of divergences (collapsed across chains) at the bottom of a trace plot. In this case, the divergences correlate with the sticking behavior (seen for chain 1, dark green, over iterations 650–800); the sampler is likely struggling to sample values near 0 for this parameter. (c) If univariate plots are insufficient to localize the issue, a bivariate density plot can demonstrate whether divergences (in red) are randomly distributed or are concentrated in one area. Here, the divergences concentrate at the tip of this funnel, where lower values of `sigma`*, a standard deviation hyperparameter, increasingly constrain the values for* `mu`*, a mean hyperparameter. (d) This common problem for hierarchical models is overcome by reparameterizing the model (see Reparameterization subsection for details).*

diverged is recorded as an indicator (1 or 0). The divergent iterations occur when a chain has attempted to travel to a point in the joint posterior, but failed to do so as it was unable to navigate the *high curvature* in that region (Livingstone et al., 2019; Betancourt, 2018). Divergences are a critically important diagnostic because they signal that part of the posterior distribution could not be explored, and as such, the available posterior samples are known to be biased (as is demonstrated in Figure 5; Betancourt & Girolami, 2015; Betancourt, 2018; Monnahan et al., 2017).

As such, when using an HMC/NUTS sampler, it is required to check that no divergences occurred (Betancourt, 2018; Stan Development Team, 2022). If there were any divergences, the samples cannot be trusted (Gelman et al., 2020) and should not be used for parameter estimation, model comparison, or any other type of inference (Betancourt, 2018; Stan Development Team, 2022). Instead, the output should be investigated in order to determine what parameter or part of the model specification might be inducing the unnavigable posterior geometry.[6]

[6]Veteran practitioners of Bayesian cognitive modeling who update their modeling pipelines from JAGS to Stan, for example, may experience that model specifications that previously passed convergence checks may suddenly fail due to the detection of divergences. While it is tempting to immediately conclude that Gibbs sampling is more capable of estimating such a model, this is unlikely to be the case, as HMC/NUTS is a more efficient sampler that circumvents many limitations of Gibbs sampling (such as being challenged by correlated parameters, high dimensionality, etc.; Neal, 2011; Hoffman & Gelman, 2014). Rather, it is more likely that the Gibbs sampler was silently failing to explore the posterior distribution fully, and these

Admittedly, in very rare cases, the sampler may record a divergence when the trajectory did not in fact diverge. While some sources note that divergences may be disregarded in special cases (Gabry et al., 2019; Schad et al., 2021), we find it important to note that these sources are universally written in the context of statistical linear models. In the context of Bayesian cognitive modeling, we do not recommend ever disregarding divergences. Unlike linear models, which have been used for more than a century and are exceptionally well-understood, cognitive models are fundamentally bespoke things: They are continually being customized, tweaked, and extended, and entirely novel models are regularly designed. In our experience, even established cognitive models can suddenly fail when applied to a new dataset (as in a case where a participant is not performing the task, and produces a series of nonsensical responses that "break" the model). For these reasons, we recommend that practitioners of Bayesian cognitive modeling always work from the assumption that divergences are genuine, and consider their diagnostic potential.[7]

The simplest strategy to use in investigating the cause of divergences is to generate trace plots with divergence indicators included. In each support library, there is an optional input that will tell the trace plot command to include a rug plot of divergences at the bottom of the trace plot (wherein each iteration for which a divergence was observed for *any* chain is marked by a red tick). Often if the aforementioned sticking behavior is seen in a trace plot, this sticking will correspond directly with the occurrence of divergences for each chain (as seen in Figure 5a and 5b). However, if a chain appears to stick in one parameter's trace, this often constrains sampling for other parameters such that the same chain may appear to stick over the same iterations in their traces as well.

To be sure of which parameter is driving the divergences, one should also visualize *bivariate* marginal densities with indicators for divergent samples overlaid. For example, in Figure 5c, the divergences are concentrated at the bottom of the joint distribution, where the `sigma` parameter takes lower values. After using reparameterization to fix this model, it is apparent that a considerable portion of the joint distribution (nearby to the divergences) was previously inaccessible, but is now able to be sampled. Reparameterization is often a successful approach to overcome the problems flagged by divergences, by enabling the sampler to more easily navigate the posterior geometry (without otherwise changing the model, as we discuss in the *Reparameterization* subsection later on). As such, the goal of investigating divergences is simply to identify which parameters or part of the model specification is the best candidate for this reworking. We will return to problems indicated by divergent transitions in the next section, *Identifying the root issue*.

Another recently introduced diagnostic that is specific to HMC/NUTS sampling is the estimated *Bayesian fraction of missing information* (BFMI or E-BFMI; Betancourt,

---

failures only became detectable with the advanced diagnostics of HMC/NUTS. Some cognitive models may have defining features that are ultimately unable to be implemented with HMC/NUTS samplers, but this determination should be made *after* troubleshooting and other inquiries.

[7] We offer supplementary code to demonstrate exactly this issue. The `example_funnel.m` script (included in the `examples` folder of `matstanlib`; Baribault, 2021) implements a toy model (adapted from Betancourt & Girolami, 2015) that was written to demonstrate a serious structural problem common in hierarchical models that is sometimes signaled only by divergences. We encourage you to run this script a few times: You may notice that on some runs, only a small number, or even 0 divergences occur. Consider whether a couple of divergences can be disregarded, given that even this model which is designed to fail may not reliably throw divergences.

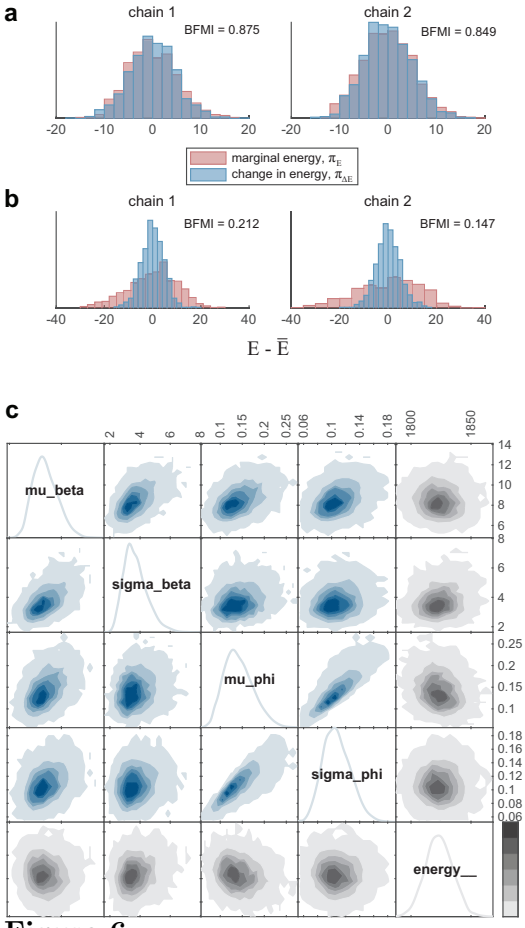

Figure 6

Diagnostic plots for troubleshooting low BFMI. (a) In an energy diagnostic plot, every chain's marginal and transitional energy distributions should overlap. (b) If they do not, the discrepancy suggests that the chain has likely failed to efficiently explore the posterior distribution. (c) Including the energy diagnostic in a grid of bivariate densities may be used to identify which parameters are the most likely contributors to this inefficiency as their samples will tend to correlate with the energy history. (None of the parameters in this plot are suspicious.)

2016). BFMI is computed from the energy diagnostic that is recorded during the generation of each iteration within each chain. It is a metric of the HMC/NUTS algorithm's accuracy at a much deeper computational level than the other diagnostics we discuss here. Nonetheless, the interpretation of BFMI is clear: When the BFMI value for a given chain is extremely low, it indicates that the chain was unable to fully explore the posterior distribution, and as such, the samples we do have are insufficient and likely biased (Betancourt, 2016).

It is currently required that BFMI is ≥ 0.2 for all chains (Betancourt, 2016; Stan Development Team, 2022). If BFMI is less than 0.2 for one or more chains, the output should not be used as the basis for inference (Stan Development Team, 2022) as the posterior estimates will be biased (Betancourt, 2016). Most often, when the BFMI check is failed, other computational checks will be failed also; in this case, the other diagnostics are more targeted, and so should be investigated first. However, sometimes only the BFMI computational check will fail. In this more difficult scenario, an energy diagnostic plot (Figure 6a,b) should be used to visualize the energy distribution differences for each chain, and a plot of multiple bivariate densities that includes the energy diagnostic (as in Figure 6c) should be inspected. If one or more parameters' marginal posterior samples appear to correlate with energy, then tweaking the places in the model specification that most directly involve those parameters is most likely to help.

A third diagnostic that is specific to HMC/NUTS sampling is the treedepth of the trajectory computation used to generate each sample. If the *maximum treedepth* was reached, it does not signal either failure or bias as the other diagnostics do. Rather, it may be interpreted as an indicator that the sampler is taking too long to compute each sample, and so maximum treedepth warnings are generally not of as much concern (Livingstone et al., 2019; Stan Development Team, 2022). If the maximum treedepth is being reached for a considerable proportion of samples, there might be a problem — but as this is very infrequently encountered in Bayesian cognitive modeling, we do not discuss treedepth or other advanced topics related to sampler tuning dynamics here (but see Betancourt, 2018 or Hoffman & Gelman, 2014 for a introduction).

Sampling efficiency

The final computational check that model output must pass is related to sampling efficiency. Before the model output is used for inference, one should check that the samples offer enough information to support the specific sample-based estimates one intends to use as the basis for inference (Gelman et al., 2013; Vehtari et al., 2021). (While this concept is superficially similar to the idea of statistical power, whereas power can is assessed a priori, sampling efficiency is assessed post hoc. This is because the sampling efficiency check depends not just on the number of samples one set out to collect, but on the sequence of samples that was actually collected.) Sometimes low sampling efficiency is obvious, as in cases of significant autocorrelation (see Figure 3c), where it appears that *effectively* fewer places were visited in the posterior space than we would expect considering the actual number of samples we collected.[8]

Estimates of *effective sample size* (ESS; previously called the number of effective samples, N_{eff}) get at exactly this issue, by quantifying sampling efficiency in a reliable way.

[8] While this is technically an oversimplification, it is the correct intuition.

When sampling efficiency is poor, ESS will be much lower than the actual number of samples collected. For all Bayesian models, it is now required that the ESS is at least 100 $\times$ the number of chains (i.e., ESS $\geq$ 400, assuming four chains) for all parameters (Vehtari et al., 2021). By default, the current implementation of ESS quantifies the sampling efficiency in both the bulk and tails of the posterior distribution, but ESS estimates can also be computed for other applications, such as for specific quantiles and small intervals of quantiles, as well as for the posterior mean, median, standard deviation, and mean absolute deviation (Vehtari et al., 2021). Before reporting credible intervals for Bayesian cognitive model parameters, one should ensure that the ESS estimates for the relevant quantiles are likewise above the criterion.

When ESS estimates are low, trace and rank plots may again be used to probe whether this could be the result of mild autocorrelation. If no other computational problems were detected and the chain traces appear autocorrelated, then one may simply run the model again with an increased number of kept iterations in hopes of a proportional increase in the relevant ESS estimates. (While past sources may recommend thinning the samples by a factor of n — meaning discarding every nth iteration — to reduce autocorrelation posthoc, thinning is no longer recommended, except in cases of severe computer memory constraints, as it degrades the precision of posterior estimates; Link & Eaton, 2012.) In many cases, however, the poor sampling efficiency indicated by low ESS estimates is signaling a deeper problem with the model. In particular, low ESS may suggest that one or another factor is making it difficult for the sampler to move through the posterior space.

Diagnostic ESS plots (introduced very recently in Vehtari et al., 2021) may be used to clarify whether low ESS is indicative of any systematic sampling inefficiencies and biases. The first such plot visualizes the efficiency over subsets of iterations (Figure 7a). This plot is particularly useful as some sampling inefficiencies may only become evident when sufficiently long chains are run. Ideally, the sampling efficiency should be such that ESS estimates grow linearly with the number of samples. One should be wary if ESS estimates level off or decrease, as this suggests those periods of sampling were relatively less efficient; this metric should be stable over time.

The other diagnostic ESS plots both help to visualize whether different values for a given parameter are being more or less efficiently estimated. Visualizing whether ESS estimates are notably lower for some quantiles (Figure 7b) or regions of quantiles (Figure 7c), especially if those quantiles seem to correlate with divergences or hitting maximum treedepth, can help to identify what areas of the marginal posterior are driving the low ESS. For example, while ESS might be somewhat lower for extreme quantiles, if it is so markedly lower in one region that it is below the ESS criterion, it may suggest that that area of the parameter space was unable to be efficiently explored. If this is the case, it might help to explain other diagnostics, such as divergences, by isolating which part of the posterior is problematic.

In this way, diagnostic ESS plots can be helpful to distinguish cases of too few samples due to an acceptable amount of autocorrelation (in which case the model may simply be run again to collect a greater number of samples) from deeper, more fundamental problems with a model (in which case the model specification should be improved). While the latter is more challenging to correct, we will discuss techniques to target and address these issues also in the next section, *Identifying the root issue*.

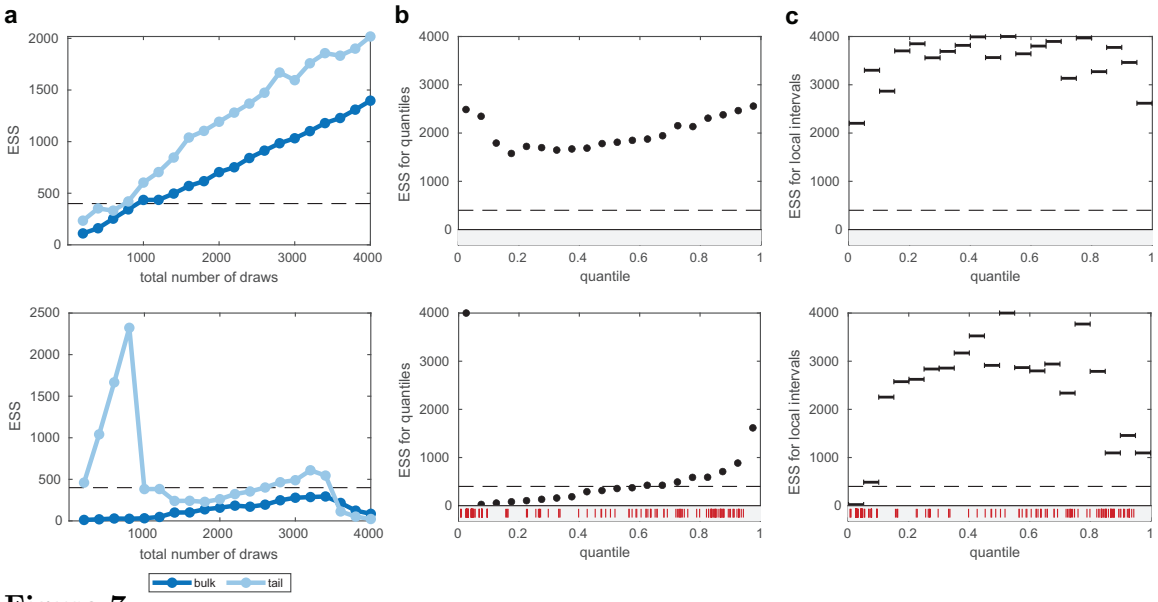

Figure 7

Diagnostic plots for troubleshooting low ESS. Ideally, ESS will grow linearly with the total number of samples (pooled across chains) in the efficiency per iteration plot (a), and all ESS estimates will be above the dashed line representing the minimum ESS in the efficiency of quantile estimates plot (b) and the local efficiency of small-interval estimates plot (c). While these patterns are seen for the well-behaved model (top), they do not hold for the problematic model (bottom), where tail ESS crashes as more samples are collected and other ESS measures are often below the criterion. Troubleshooting techniques are necessary to identify the root of these problems.

Consistency checks

Because the computational checks described in the previous section were designed only to assess the quality of the HMC/NUTS sampling, additional checks are necessary to assess whether the model itself is behaving in a way that is consistent with our intentions and assumptions. For example, whether the model — *as it is currently implemented* — is able to capture the kind of behavior that we expect to observe is important to check, as this directly bears on whether the model is appropriate for the research context.

The two *consistency checks* that we discuss here are essential techniques for detecting these sorts of problems with Bayesian cognitive model behavior, and should be seen as of equal importance to the computational checks.

Prior predictives

Even before the simulation study is performed, *prior predictive checks* should be used to check whether the patterns of behavior predicted by the model specification are sensible, given the research context (Box, 1980; Gelman et al., 2013; Lee & Vanpaemel, 2018). The first step is to generate the *prior predictive distribution* by sampling a large

number of datasets from the model specification. To sample a dataset, hyperparameter values are drawn from the hyperpriors, which are subsequently used to draw parameter values from the priors, which are used to simulate data from the model's data distribution, given the experimental design (i.e., in the exact same way that a dataset is generated for a simulation study). Next, we simply visualize the distribution of this data or, more usefully, of a meaningful *summary statistic* of the data (Lee & Vanpaemel, 2018).

This is called a *prior predictive check*, as it allows one to observe the distribution of behavioral data that is implied by the model a priori (i.e., before any data is seen by the model), and evaluate whether it is reasonable and consistent with domain knowledge, theory, the experimental design, and so on (Lee, 2018; Lee & Vanpaemel, 2018; Kennedy et al., 2019). If the prior predictive is inconsistent with these expectations, it can reveal both subtle and deep problems with a model specification. For example, even if each individual prior seemed reasonable, a failed prior predictive check indicates that the joint behavior of all priors in the context of the likelihood is unreasonable (for a discussion, see Gelman et al., 2017).

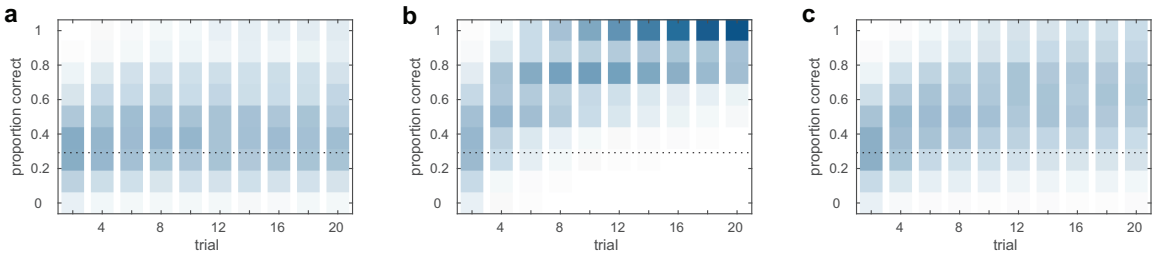

Figure 8

Prior predictive checks assess whether the model specification is consistent with one's expectations about behavior. Here, for three versions of the example RL model, prior predictive learning curves are plotted as a probability density over the proportion correct relative to to each pair of trials (where darker colors indicate higher density). The prior predictive should not place excessive weight on unlikely patterns of behavior (a), nor should it place too little weight on patterns of behavior that might reasonably be observed (b). The ideal prior predictive for our RL model example (c) is consistent with the range of behaviors that is reasonably expected, but is diffuse enough to include all possible behavioral patterns. In practice, one should use multiple prior predictive checks to evaluate a model, each of which visualizes a different quantity that is meaningful within the research context.

The key to a useful series of prior predictive checks is the careful selection of quantities to visualize. While a simple histogram of the prior predictive data can be a good check for some linear models, for a Bayesian cognitive model, it is far more common (and informative) to select a variety of performance metrics and patterns of behavior, each of which is meaningful within the specific research context. For example, in the context of a reinforcement learning task, prior predictive checks of the learning curves and asymptotic means would be essential. In the context of decision-making task, prior predictive checks might include the participant-level overall accuracy, their rates of a sub-optimal behavior, or specific error type distributions.

In Figure 8, we present a prior predictive check for three different specifications of our RL model of the bandit task. By visualizing the prior predictive accuracy distribution across trials, we can understand what range of learning curves (as in Figure 2b) are more or less likely under each RL model specification — and evaluate whether the predictions of each model are reasonable and consistent with our expectations.

If the prior predictive check reveals that too much probability has been placed on grossly unrealistic data, it strongly suggests a problem with the model as specified (Lee & Vanpaemel, 2018). For example, if an inappropriately wide swath of behavioral patterns are predicted (e.g., if model of response times predicted that, for an easy 2AFC task, response times of 5 ms, 500 ms, and 5 s were equally likely), it may suggest that the parameters of the model are too loosely constrained; in this case, more informative priors may help. Unfortunately, another example of this issue is evident in the prior predictive for the RL model specification that we outlined earlier in the tutorial. In Figure 8a, the prior predictive check has revealed that the model considers it most likely for participants to have accuracy near chance across all trials. As this is seemingly nonsensical for a model of learning to predict, and fundamentally inconsistent with the gradual learning behavior we expect to observe, this version of the RL model specification has failed the prior predictive check, and is unacceptable as a model of the bandit task.

It is also a problem if a severely restricted range of behavior is predicted, as in Figure 8b, where a significantly smaller range of learning curves is implied by the model than we might observe in the lab. Even though the most weight is given to the more commonly observed patterns of behavior in the bandit task, this second version of the RL model specification has also failed the prior predictive check because it is too tightly constrained, most likely as a result of priors that are excessively informative. Altering the priors and/or structure of the model specification in a way that alleviates these sorts of imbalances may not only lead to a more suitable prior predictive, but sometimes may be sufficient to resolve some computational and recovery failures by changing the posterior geometry.

Ideally, a prior predictive distribution will encompass a sufficiently broad range of possible behavior such that any possible pattern of behavior that we could potentially observe should be given a nonzero amount of probability by the prior predictive, and the typical range of behavior we are expecting to observe will be given just moderately more weight. After developing a third version of the RL model specification, the posterior predictive matches this ideal description (Figure 8c).

In general, whether a prior predictive check might be characterized as wholly unrealistic, overly broad, excessively constrained, or reasonable, will be intensely dependent on the particular behavioral pattern being predicted, the cognitive model, and the research context, etc. However, a number of illustrative examples are available in the Bayesian cognitive modeling literature, such as the effect of vague versus informed priors on a psychophysical model in Lee (2018; see also the examples in Kennedy et al., 2019; Lee & Vanpaemel, 2018). Even when problems are not revealed, performing a series of prior predictive checks is an excellent way to better understand the behavior and capabilities of a model.

Parameter recovery

Another way to assess the behavior of a Bayesian cognitive model is to perform a *parameter recovery check*. After running a simulation study, it can be informative to assess the correspondence between the posterior estimates and the true values, especially with assistance from *recovery plots* (see Figure 9). Before we outline three ways in which we recommend using parameter recovery checks to facilitate troubleshooting, we will first review how parameter recovery checks are most commonly used in Bayesian cognitive modeling.

In the Bayesian cognitive modeling literature, the quality of parameter recovery is often rather loosely defined as: (1) whether each true parameter value is contained within the corresponding 95% credible interval for ≈95% of the parameters in the model (Rubin, 1984; or a generalized version of this criterion if multiple simulation studies are considered, as in Heathcote et al., 2019), and (2) whether the recovery plots, which visualize the correspondence between the true parameter values and the model-derived estimates for each parameter, appear satisfactory. When recovery is very good, the point estimates will all be sprinkled closely along the diagonal unity line (representing perfect recovery), the credible intervals will be small (suggesting the model is certain in its estimates), and no bias will be evident across the estimates (i.e., the estimates will not appear to be consistently concentrated either above or below the unity line). With this description in mind, we might classify the quality of the recovery in Figure 9a as *strong*, and the recovery in Figure 9b as relatively *weak*.

However, one must be careful in interpreting the results of these recovery checks. While parameter recovery is frequently presented as a means to evaluate the accuracy and reliability of posterior estimates (e.g., Heathcote et al., 2015), this would be an overzealous interpretation of a single simulation study. Rather, each parameter recovery simulation is simply a snapshot of a model's performance in the context of a single dataset. There is no guarantee that the current pattern of parameter recovery will generalize to other possible datasets (Talts et al., 2020), and the expectation to recover true values is different than the expectation of coherent inference (for a concise discussion, see Lee, 2018). Parameter recovery checks are ultimately heuristic, *qualitative* assessments based on brief looks at model behavior.

Performing a formal, *quantitative* assessment of the internal consistency of parameter estimates for a Bayesian model requires simulation-based calibration (SBC; Talts et al., 2020; Cook et al., 2006). As we will discuss in detail later on (in an eponymous subsection), SBC is the correct way to test, on a parameter-by-parameter basis, whether the posterior estimates are systematically biased or overly wide or narrow across the entire prior predictive distribution of data. Unfortunately, the large number of small simulations required makes the procedure rather time-intensive, and sometimes impractical. As such, while SBC is an ideal capstone to the troubleshooting process, we still recommend parameter recovery checks — carefully interpreted — as a quick, heuristic check of Bayesian cognitive model behavior at the end of every simulation study (as the regular exploration of estimates is an important support to iterative model evaluation processes, Box, 1980).

Specifically, we recommend using recovery checks in three informal, yet informative, ways to support troubleshooting for Bayesian cognitive models. The first way is as a means to detect problems by highlighting major points of failure. Any extreme and obvious

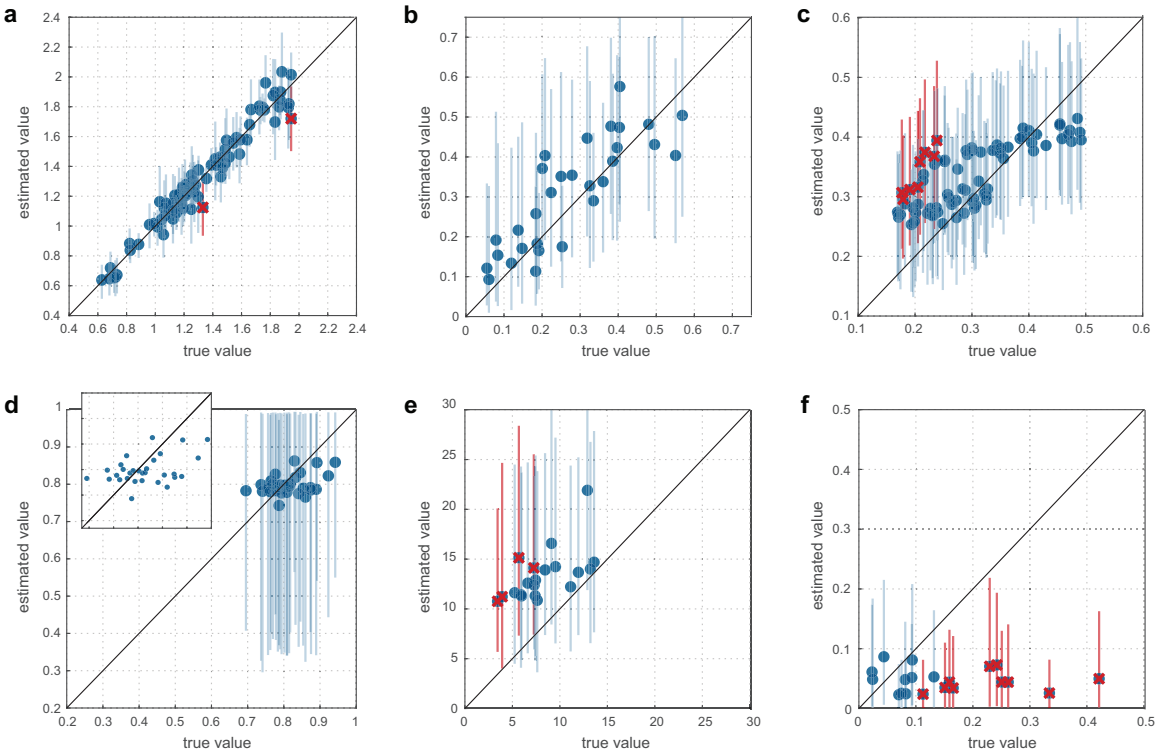

Figure 9

Parameter recovery plots. When recovery is strong (a), the 95% credible intervals (vertical lines) for around 95% of all parameters will include (i.e., recover) the true value, and the point estimates (markers) will cluster nicely around the unity or "perfect recovery" line (diagonal); few parameters fail to recover the true value (red x's). The quality of recovery in (a–c) is all potentially acceptable: Depending on the context, the weaker recovery in (b) and mild flattening of point estimates in (c) that is characteristic of over-shrinkage in hierarchical models, may or may not be sufficient. The quality of recovery in (d–f) is generally unacceptable. The extreme uncertainty in (d, main panel), which would fail to be detected if the credible intervals are omitted (inset), could signal that this parameter is insufficiently identified. The consistent overestimation bias in (e) and abject failure to recover in (f) may indicate more severe problems with the model. (Note that recovery plots should be square in order to support the consistent recognition of such patterns of recovery.)

problems are useful to investigate, especially in the earliest stages of troubleshooting. For example, if a parameter is being estimated in a way that demonstrates no relationship to the true values whatsoever (as with the collapsed estimates in Figure 9f), or if data do not seem to have distinguished the posterior in any way from the prior (as the similar estimates in Figure 9d could suggest, depending on the prior), these major failures can be important clues to the root of computational and other problems. Other times, these most extreme modes of recovery failure actually have a simpler solution: They may be the result of a garden-variety implementation error in the model specification. (Figure 9f was actually the result of inadvertently commenting out a line in the data simulation code, such that the parameter in question had no influence on the likelihood.) Other times, such an abject lack of ability for the model to capture one or more dynamics of the proposed cognitive process will suggest a deeper problem with the model, which should be investigated further.

Later in troubleshooting, recovery checks may also be used to better understand important dynamics of model behavior through the descriptive characterization of patterns in recovery plots. For example, when the hierarchical prior structure is unduly constraining lower-level parameter estimates, this can sometimes be evident from a recovery plot. While the regularizing influence of the hyperprior on lower-level estimates is a decidedly a feature of hierarchical models (Lee, 2011; Scheibehenne & Pachur, 2015), it can become a bug if this influence is pulling or *shrinking* (Efron & Morris, 1977) the estimates toward the hyper-level mean to the extent that genuine individual differences are suppressed. If the true and estimated values appear correlated, but also flattened (as in Figure 9c), such that higher true values are underestimated while lower true values are over-estimated, then it could suggest that such an excessive degree of shrinkage (or *over-shrinkage*, as in Rouder & Lu, 2005) is occurring. While a mild squashing of posterior estimates may be accepted in some cases, extreme over-shrinkage, seen as nearly flat estimates, indicates that the parameter estimates are being excessively constrained by the prior. This can occur if the hyperpriors are too strongly informative, or if the data is not sufficient to meaningfully inform the individual parameters (i.e., the realized likelihood is too flat).

It is important to note that recovery plots can easily be mischaracterized if credible intervals are omitted, as the degree of uncertainty can substantively change the interpretation of a recovery plot. The omission of credible intervals can be particularly detrimental for the detection of weakly identified and unidentified parameters. For example, Figure 9d includes two versions of the same recovery plot for the mixture proportion in a hierarchical latent mixture model. When the credible intervals are included, it is readily apparent that the most pressing issue is the extreme uncertainty, which should certainly be investigated. However, this important pattern would have been impossible to glean from the point estimates alone. Only the complete plot could lead us to correctly characterize this mixture parameter as being *weakly informed*, which suggests that either not enough data is available to update this parameter's value, or the data that is available is not informative enough. The omission of uncertainty can also make it challenging to recognize some hallmarks of structurally *unidentified* parameters. For example, the posteriors for an unidentified parameter might appear quite certain about estimates that fail to meaningfully correspond to the true values; recognizing this unusual pattern of confidently incorrect recovery relies on the availability of information about the posterior certainty (for examples, see Spektor & Kellen, 2018).

Another important pattern that recovery plots may reveal is bias, such that a parameter is being consistently being over- or under-estimated (as in Figure 9e). This can occur if the prior places most of its weight on values that are far from the values it would otherwise infer; depending on the model, this can bias all estimates in the same direction. *Prior simulation* (introduced later on in the *Parameterization* subsection) is a good approach to investigate this possibility, especially in hierarchical models when hyperpriors can be challenging to specify. This kind of consistent bias may also occur when two or more parameters are *trading off*, in which case a pathological inverse coupling is observable each time the model is fit. While this behavior is sometimes just a structural fact of some cognitive models (e.g., Turner et al., 2013; Krefeld-Schwalb et al., 2022), in other cases, it can be ameliorated by making the priors for the relevant parameters more informative. That these two rather different issues can cause the same pattern in a recovery plot raises an important point: While Figure 9 showcases some classic recovery patterns, these are not uniquely mapped to a singular underlying problem (e.g., while shrinkage often causes estimates to appear squashed as in Figure 9c, the converse is certainly not always true).

Finally, recovery checks can be an excellent tool to explore how different design choices might change these patterns of parameter recovery (Apgar et al., 2010; Lee, 2018). For example, if the decay rate parameter in our example RL model seemed to be weakly-informed, we could perform additional simulation studies to explore how increasing the number of participants, the number of bandit problems per participant, and the number trials per bandit problem each might affect the strength of recovery differently. These sorts of informal simulation-based investigations can be particularly useful towards the end of the troubleshooting process, at which point the model specification may be in good shape but whether the planned experimental design is sufficiently informative for the parameters of interest may yet be unclear. Additional simulation studies can shed much-needed light on whether the final experimental design will strike a good balance between informativeness and efficiency — which is particularly important in research contexts where every data point is at a premium (Gluth & Jarecki, 2019). While this is currently a less common application of parameter recovery checks (but for a recent example, see Danwitz et al., 2022), it is likely to be one that will increase in prevalence and importance as the use of Bayesian cognitive models continue to broaden.

Identifying the root issue

While some problems that have been detected by the computational and consistency checks will be easier to identify, other problems will require a longer and more investigative troubleshooting process before the nature of the problem — and therefore a candidate solution — can be identified. In many cases, visualizing the posterior samples for multiple parameters simultaneously, with key diagnostics included, will be a fruitful approach to understand the root cause of the detected problems.

Posterior geometry

These visualizations are especially important tools when attempting to identify the cause of issues related to posterior geometry, such as the *regions of extreme curvature* that HMC/NUTS samplers will loudly struggle to traverse. As mentioned earlier, whenever

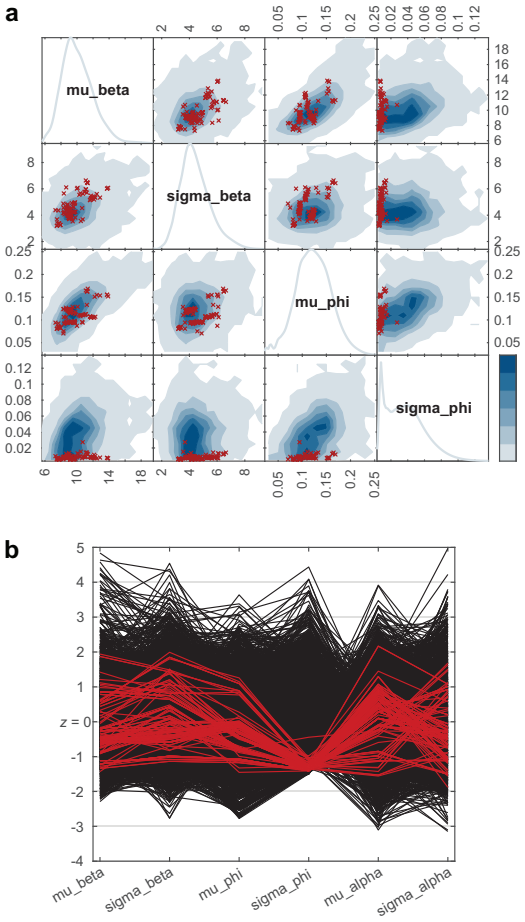

Figure 10

Visualizing the posterior samples for multiple parameters simultaneously using (a) grids of bivariate marginal densities with diagnostic overlays and (b) parallel coordinate plots are both are useful to search for problems related to posterior geometry. In the grid of densities, one should look for parameters where divergences (red x's) are not randomly distributed, but rather are clustered together. In the (z-scored) parallel coordinate plot, where each line represents a joint posterior sample, one should look for where the red lines representing divergent samples seem to "pull together." Both of these plots loudly suggest that root of the issue is an unnavigable region of high posterior curvature at the lower bound for `sigma_phi`*.*

the sampler reports divergences, it indicates that an unnavigable region of high curvature was encountered. The goal is to uncover where exactly in the joint posterior that region is located, and which parameters are most directly implicated in creating the high curvature, so that the relevant section(s) of the model specification might be reparameterized or otherwise improved.

Grids of many bivariate marginal posterior densities (as in Figure 10) are often the most useful diagnostic visualizations in this pursuit as they allow one to simultaneously search for concentrations of divergences as well as posterior dependencies among pairs of parameters that signal specific problems. This style of plot (which is available in all support libraries; see Table A1) makes it readily apparent when parameters are correlated, as their joint density will appear as an oblong shape. While parameter correlations are well-known to harm the quality of Metropolis and Gibbs sampling (e.g., Turner et al., 2013), parameter correlations are rarely a cause for concern for HMC/NUTS sampling, due to HMC algorithms' avoidance of random walk behavior (Neal, 2011). However, extreme correlations that seem to approach collinearity, or other strange shapes like bananas, will require investigation as these features can suggests nonidentifiability with respect to that pair of parameters, or other degenerate model configurations, may be the root of the issue.

One should also be suspicious of bivariate densities with a *funnel* shape, as is commonly observed in models with hierarchical prior structure. For example, when setting priors directly on mean and standard deviation hyperparameters, progressively smaller values of the standard deviation hyperparameter will increasingly constrain the range of the lower-level parameter values, which in turn constrain the value of the mean hyperparameter. This can induce the progressively narrow funnel shape in the bivariate density. If the "tip" of the funnel takes on a much higher curvature than the rest of the posterior, then the sampler will struggle to access this region, leading to a concentration of divergences close by (as is seen at the bottom of the joint distribution in Figure 5c). If this pattern is recognized, the issue is often easy to correct by converting to a non-centered parameterization (as demonstrated at the conclusion of this section), which enables the funnel to be fully explored by breaking the dependency between the relevant parameters (Betancourt & Girolami, 2015).

Other potentially difficult to navigate posterior regions can occur at parameter boundaries, especially when the boundary was introduced by *truncation*. For example, a Normal prior that has been truncated such that its lower boundary of -1 should not pose a problem *except* if the highest density in the target distribution is very close to -1. For computational reasons that are beyond our scope, this can make it challenging for the sampler to both enter and exit this region, leading to a variety of problems. If it is at all possible for the boundary to be adjusted while still respecting the relevant domain knowledge or original theoretical justification, moving the truncation further out should help to resolve these issues.

Unfortunately, when a very high proportion of the posterior samples are the result of divergences, it can be difficult to use the bivariate density plots for troubleshooting as the densities can seem to all be thoroughly covered in the red divergence indicators. In these cases, a bivariate density plot can still be useful as a quick way to view many univariate marginal densities (as shown along the diagonal of the Figure 10a). If bumps or multimodality are observed in the univariate distribution, this may suggest a place where

the divergences are more highly concentrated, even when it is not otherwise apparent.

We can also search for concentrations of divergences with *parallel coordinate plots*. In this diagnostic plot, each joint posterior sample for a given subset of model parameters is presented as a separate line, with divergent samples shown in red (as in Figure 10b). If the divergent sample lines appear to "pull together" for one of the parameters (while appearing randomly distributed across the values of other parameters), it indicates that this parameter may be responsible for the divergences, and naturally highlights the range of values that might be inaccessible. How this issue should be resolved will depend on that parameter's role in the model specification.

A challenge in using either of these plots as diagnostic tools is that there will almost always be too many parameters to include on the plot at once. A strategy that we have found useful is to begin with the parameters at the highest levels of hierarchy in the model, investigating all possible pairs, then working downward. At lower hierarchical levels where there are many parameter instances, including one or two instances of any parameter at most is also generally more useful than visualizing many instances of the same parameter. However, in some cases, neither the parallel sample plots nor the bivariate density plots, or any other previously discussed diagnostic techniques will clearly implicate any particular part of the model specification. If the nature of the issue is still unclear after a thorough troubleshooting process, critical review of the model specification, combined with exploratory changes, may prove more worthwhile.

Parameterization

When the troubleshooting process has led to the identification of a parameter or segment of the model specification that is problematic, one or more strategies to adjust this portion of the model specification may be used to attempt to resolve the model's issues. Here, we review a number of useful techniques to *change the parameterization* of the model, which may have small or large consequences for domain knowledge expressed in the priors and the theoretical implications of the model overall. If the questionable part of the model specification cannot be altered without severe undesirable consequences for the interpretation of a key parameter, or the model's ability to express a specific cognitive process, one should instead try to *reparameterize* the model, as we discuss in the next subsection.

In Bayesian cognitive models, it is not uncommon to use non-conjugate, non-normal priors, nor is it uncommon to include hierarchical structure in the model. Unfortunately the conjunction of these two design decisions can make it exceptionally difficult to specify good hyperpriors. For example, while one may have an intuition for what a good participant-level Gamma prior distribution would be, one may feel at a loss in determining suitable group-level hyperpriors for the shape and rate hyperparameters of that Gamma prior. In this scenario, using *prior simulation* to visualize the distribution of priors implied by different hyperpriors, and their collective effect on the distribution of prior probability across the domain of a given parameter, can offer the support needed to specify reasonable hyperpriors.

In Figure 11, we demonstrate how *prior simulation* can be used to both identify problems and investigate solutions. Given the specified hyperprior distributions (top left), random samples from each distribution (or function of them; top right) are used to define a random selection of priors (bottom). If prior simulation reveals that extremely undesirable

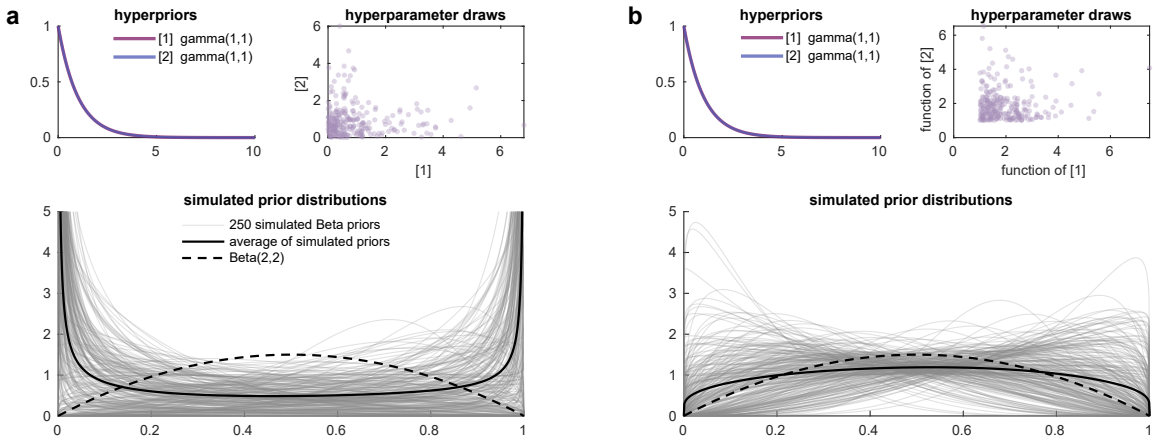

Figure 11

Prior simulation may be used to check whether the priors and hyperpriors imply an allocation of prior density are consistent with domain knowledge and other expectations. In this example, a Gamma hyperprior is specified for each hyperparameter of a Beta prior. (a) The original hyperpriors lead to undesirably high prior weight at the extreme values of the parameter of interest. (b) Enforcing a minimum value of 1 on both hyperparameters (by specifying $\sim Beta(1 + a, 1 + b)$ *instead of* $\sim Beta(a, b)$*) prevents the selection of U-shaped priors, allowing for a more appropriate distribution over priors that, on average, allows for a more even spread of prior weight across the whole range of the parameter value, excluding the bounds.*

priors are too often being sampled, prior simulation should continue to be used to explore alternative hyperpriors. In these subsequent simulations, it can sometimes be useful to constrain or transform the hyperparameters in such a way that the unsuitable priors are no longer possible.

For example, in our efforts to improve the specification of our example RL model, we used prior simulation to help specify a different prior and hyperprior for the learning rate parameter, α. In place of the truncated Normal prior (that was likely unsuitable in light of the failed prior predictive check), we decided to use a Beta prior, as it is naturally defined over the same $(0, 1)$ interval as the learning rate. Our goal was to set hyperpriors that would imply a distribution over Beta priors that would not be unduly biased toward any part of the parameter space (i.e., that is lightly informative, given the research context). In Figure 11a, our initial choice of Gamma(1,1) hyperpriors was revealed to lead to the overselection of priors that placed infinite weight over 0 and/or 1 (approximately 40% of priors!), which is inconsistent with our intentions (represented by the Beta(2,2) distribution) and unreasonable for a learning rate (a prior that suggests no learning and perfect learning are both more likely than gradual learning is silly). After we enforced a minimum value of 1 for each hyperparmeter, as in Figure 11b, horseshoe priors were no longer possible, and a more reasonable distribution over priors was achieved. In our final, corrected specification of the example RL model, we used exactly this specification for the prior and hyperpriors for the learning rate, and a similar setup for the decay rate, ϕ (but with slightly different

hyperpriors, so that most prior weight allocated to lower values).

These sorts of bounding tricks are especially useful when there is a need to exert control over the prior near a boundary to avoid model misspecification (as for parameters whose values cannot conceivably be 0 *and* sufficient domain knowledge is available to further specify what parameter values should qualify as "near 0" or "practically equivalent to 0"). In our example RL model, an inverse temperature of $\beta = 0$ breaks the model (i.e., leads to a degenerate model configuration), as the learned Q values will have no bearing on action selection (and so α and ϕ will be unidentified). As such, a prior for β that apportions most of the prior probability to 0 and values near 0 is effectively a model misspecification, as the most prior weight is given to not just the least likely values, but values that are so inappropriate that one would conclude the model is malfunctioning rather than accept the estimates. One approach to cope with exactly this scenario is to use a *boundary-avoiding prior* (Gelman et al., 2013; Chung et al., 2013). These parameterizations should approached with caution for dispersion parameters in hierarchical models, where reparameterization may be preferable (as discussed at the end of this section), and in Bayesian linear statistical models, where recommendations may be entirely different (for a recent account, see Röver et al., 2021). However, for parameters in Bayesian cognitive models, boundary-avoiding priors are sometimes not only permissible, but more appropriate than alternative prior specifications. In our final, corrected specification of the example RL model, we use a Gamma prior for the inverse temperature parameter where the shape hyperparameter is required to be be greater than 1; this creates a *zero-avoiding* prior by ensuring the Gamma distribution allocates zero probability to a value of 0. In our earlier prior simulation example, in which we set lower bounds on both hyperparameters of a Beta distribution, you may notice that the revised prior simultaneously avoids both the lower bound and the upper bound (0 and 1; see Figure 11).

Prior simulation is often a critical support to prior predictive checks, and vice versa. While prior predictive checks may reveal implications of the model that are inconsistent with one's expectations, they rarely also reveal the exact cause of that inconsistency: In these cases, prior simulation can be used to understand *why* a prior predictive check failed, and therefore help to isolate the root of the problem. However, good prior simulation results alone are likewise insufficient: Prior predictive checks must be used to demonstrate that all priors collectively make sense in the context of the likelihood (from which they cannot be divorced; Gelman et al., 2017).

If one is still struggling to simulate a sensible apportionment of probability across the distribution of priors, then changing the distributional form of the prior can open up additional opportunities for model improvement. In particular, changing the form of a prior may help in cases where one or more features of the model specification are known to induce posterior geometries that are challenging for HMC/NUTS algorithms to navigate. For example, prior distributions with fat tails, such as the Cauchy and Student's t distributions, can lead to divergences and an overwhelming number of maximum treedepth warnings when sampling in the tails. If this occurs, using a lighter-tailed alternative, such as a normal distribution, is a good alternative (although reparameterizing is also an option).

Changing the form of the prior is especially likely to help in cases where the cognitive model demands that a parameter be defined over just a subset of the reals and truncation is currently being used to effect the domain constraint. In some cases, selecting an alternative

prior form that is naturally defined on the desired domain may help to resolve a variety of issues (as in our revised specification of the example RL model). For example, rather than truncating a distribution to the positive reals (when it is naturally defined over the entire real line), one might use a distribution that is already defined only on the positive reals, such as an Exponential or Lognormal distribution. In a similar fashion, rather than doubly truncating a distribution when both a lower and upper bound is needed, the generalized Beta distribution (meaning, a Beta distribution that has been scaled and/or shifted such that it is defined over a domain other than $[0, 1]$) is likewise a handy alternative.[9]

If such a change of prior leads to less interpretable hyperparameters, one can often use known formulae to derive more useful quantities (e.g., samples for a Gamma prior's shape and rate hyperparameters may be used to compute, sample by sample, the hyper-level mean and standard deviation). These kinds of posthoc reparameterizations are straightforward to implement (and may be automated; for example, the MATLAB support library `matstanlib` can apply select commonly used transformations, see Table A1). Some priors with less-easily-interpreted parameters also have established alternative parameterizations that are more intuitive to work with, and may facilitate the setting of hyperpriors in hierarchical models. Not all parameterizations are useful: For example, the Gamma distribution may be parameterized by a shape parameter, α and a *rate* parameter, β (as in Stan), or in terms of a shape and *scale* parameter, where the scale is simply the inverse of the rate (as in MATLAB), but a more interesting alternative parameterization of the Gamma distribution is in terms of its *mean*. The mean of a Gamma distribution is defined as the ratio of its hyperparameters, $\mu = \frac{\alpha}{\beta}$. A simple variable substitution permits the reparameterization $\text{Gamma}(\alpha, \frac{\alpha}{\mu})$. Other distributions have known mean-based reparameterizations; we have found these to be helpful to strike an easier balance between sampler-friendly geometry and parameter interpretabilty in a variety of contexts.

Reparameterization

While each technique discussed in the previous subsection required a making a change to the model, in some cases one may need to strictly preserve the model to protect the psychological interpretation of one or more model parameters, or the theoretical implications of the model overall. In this scenario, the preferred approach would be *reparameterization*, which allows for a part of the model specification to be converted to a form that is more computationally efficient, but ultimately equivalent (Gelman, 2004; Gelman & Hill, 2006). The goal of many reparameterization techniques is to permit easier sampling and faster convergence by reducing correlations or other dependencies in the joint posterior (Gelman et al., 2008).

Some reparameterization techniques are quite tailored to the particular Bayesian cognitive model at hand, such as those that serve to better distinguish the role of key parameters within the model (for a recent example, see Park et al., 2021). Other reparameterization techniques are more general-purpose, and might apply to any Bayesian cognitive model where a certain structure is incorporated. For example, order constraints may be reparameterized such that the same idea is re-expressed as a multiplicative scaling between

[9] We also caution against truncations generally, even when they work well in a model, because they can make extending the model later on extremely difficult.

independent parameters (Knapp & Batchelder, 2004); while this technique is very commonly used with multinomial processing tree models (Batchelder & Riefer, 1999; Klauer et al., 2015), it might be more broadly applied to other models.

There are a number of other general-purpose reparameterization techniques (Gelman, 2004), including many that are of particular use for hierarchical models (for an in-depth discussion with specific techniques and examples, see Gelman & Hill, 2006). Many such approaches rely on the inclusion of *redundant parameters* which, while not identified, often are effective in facilitating more reliable posterior exploration and sampling for the parameters of interest. These additive or multiplicative *parameter expansion* techniques can improve the quality sampling and recovery (Gelman et al., 2008; Browne et al., 2009), and have been successfully applied in Bayesian cognitive models (for an example, see Matzke et al., 2015).

Another widely-applicable repararmeterization technique for hierarchical models is *non-centered parameterization* (Betancourt & Girolami, 2015, previously called the "Matt trick" in some older sources), which we have obliquely referenced throughout this tutorial. A pathological funnel-shaped posterior geometry (as shown earlier in Figure 5c) can be induced when a *centered* parameterization was used:

$$\begin{aligned} \mu &\sim \text{Normal}(0, \sqrt{10}) \\ \sigma &\sim \text{Gamma}(2, 1) \\ \theta_n &\sim \text{Normal}(\mu, \sigma) \end{aligned}$$

so called because the prior is *centered* on the mean parameter. This section of the model may be rewritten to use a *non-centered* parameterization:

$$\begin{aligned} \mu &\sim \text{Normal}(0, \sqrt{10}) \\ \sigma &\sim \text{Gamma}(2, 1) \\ \eta_n &\sim \text{Normal}(0, 1) \\ \theta_n &= \sigma \cdot \eta_n + \mu \end{aligned}$$

which is mathematically equivalent. Even though the same hyperpriors are used, this expansion allows the entirety of the funnel to be explored efficiently, by introducing an auxilliary sampled variable η that is independent of μ, and then rescaling it by the sampled standard deviation σ (Figure 5d). While non-centered parameterizations are most frequently applied in the context of Normal prior distributions, they are also applicable to any prior distribution that is parameterized in terms of a location and dispersion hyperparameters. However, it is important to note that the non-centered parameterization is not always superior: If a wealth of informative data is available, the centered parameterization may offer better performance, while the non-centered parameterization would cause issues (Betancourt & Girolami, 2015). The `example_funnel.m` script included in `matstanlib` demonstrates both the centered and non-centered parameterizations for a toy model, exactly as specified above.

Of course, parameterization and reparameterization are such broad terms that we cannot hope to cover even most of the most popular methods, techniques, and tricks in this level of depth. We encourage you to explore the referenced sources and recent work on similar models to further investigate techniques that might be of greatest use for your specific Bayesian cognitive model.

From troubleshooting to model development

At this point in the tutorial, we have described in great detail how diagnostic checks and plots may be used to detect and identify the most commonly encountered problems in Bayesian cognitive modeling. We have suggested a handful of remedies along the way, and highlighted specific changes in parameterization and reparameterizations that often constitute good solutions for Bayesian cognitive models. Each time you apply a Bayesian cognitive model, it is always necessary to perform the model-checking steps and any subsequently needed troubleshooting as outlined here to ensure that the output from your model is (1) computationally sufficient and (2) consistent with your intentions. Even if you are using an established Bayesian cognitive model, diagnostic checks and plots can suddenly reveal problems when the model is applied to a new dataset, or is fit with a different sampling algorithm.

In this section, we discuss a few final techniques that may vary in relevance depending on your analysis plan, but all test additional important expectations and assumptions about model behavior. While some techniques are more often seen in terms of their role in a wider Bayesian cognitive modeling workflow, all should also be considered as possible steps in the troubleshooting process, as each can support the identification of shortcomings that can compromise the validity of model-based inference.

Depending on how one is planning to apply a given Bayesian cognitive model (or models), some or all of these final checks may be needed to ensure your model capable of doing what you will ask of it. Most of these techniques may need to be customized to your model and research context to even be implemented at all.

Simulation-based calibration

As we discussed earlier, while parameter recovery studies have been the most commonly used method to explore the quality of Bayesian cognitive model estimates, they are far more useful as a qualitative troubleshooting tool. If one does intend to assess the accuracy or reliability of posterior estimates, particularly in the sense of whether a Bayesian cognitive model's estimates are internally consistent, then the correct Bayesian approach is *simulation-based model calibration* (SBC; Talts et al., 2020; Cook et al., 2006). Whereas parameter recovery is a heuristic assessment of model behavior for a single simulated dataset, SBC offers a more principled, comprehensive approach to quantify the coherence of posterior estimates over the entire prior predictive distribution of data and, ultimately, to formally validate a Bayesian model as it is currently implemented. Unlike a recovery check, SBC allows for a quantitative assessment of whether the posteriors tend to be overly wide, overly narrow, or otherwise biased on a parameter-by-parameter basis. This is exceptionally useful information for troubleshooting as these biases and other miscalibrations may be targeted and ultimately corrected through many of the investigative and model-adjusting techniques previously discussed.

The SBC procedure entails running N_{reps} replications of a recovery scoring routine. For each replication, a small simulation study is run: true parameter values $\tilde{\theta}$ drawn from the priors are used to simulate a dataset $\tilde{y}$, which is then submitted to the Bayesian cognitive model to collect a relatively small number L of post-warmup iterations. The result of each replication is each true parameter value's *rank* within the corresponding posterior samples

for all parameters in the model (Talts et al., 2020; vs. the earlier approach of Cook et al., 2006 which uses *quantiles*). If the model is correctly implemented, then the ranks across the N_{rep} replications will be uniformly distributed for every parameter in the model. A calibration failure in the posterior estimates is detected when deviations from uniformity are evident in the rank histograms or other SBC diagnostic plots for a given parameter (Talts et al., 2020). Because chain autocorrelation violates an assumption of SBC, adjustments to this base SBC procedure are required when the ESS estimate is notably lower than L. Namely, the samples should be thinned (*during the SBC procedure only*) by a factor of $\frac{\text{ESS}}{L}$ to overcome the influence of autocorrelation on the SBC plots (Talts et al., 2020). This amended and extended version of the base SBC procedure will often need to be used to properly validate Bayesian cognitive model scripts.

There has been a recent push to consider SBC as less of an option and more of a requirement in Bayesian workflows, yet SBC has only rarely been applied in the Bayesian cognitive modeling literature to date (but for an example, see Hartmann & Klauer, 2020). While we encourage psychologists to consider incorporating SBC toward the end of their troubleshooting workflow, especially for Bayesian cognitive models that are novel or relatively untested, there are some limitations on SBC's usability in practice. The first caveat is a practical limitation: Depending on the computational demands of the model (and on N_{rep} and L), performing SBC may take a considerable amount of time and/or computational resources. Using back-of-the-envelope calculations to estimate SBC runtime (from the time of a single simulation study with L iterations, multiplying by the highest thinning factor and N_{rep}, and dividing by one's ability to parallelize) is helpful to gauge feasibility. Another practical consideration is whether automation of the SBC routine is available (as via the `SBC` package in R) or unavailable (no dice for MATLAB users) in your preferred programming language, as one may also need to account for time to code the SBC routine and diagnostic plots.

Second, it is important to note that SBC is an active area of statistical research where methods and recommendations are still evolving. For example, whether subtler problems will be detected by SBC can depend on the values selected for N_{rep} and L, but there is relatively little published guidance on how to select appropriate values. Similarly, there are many ways to tweak the rank histogram and empirical cumulative distribution function plots used to draw conclusions from SBC results; as such, the effectiveness of SBC in assessing posterior calibration can depend to some extent on one's ability to perform these hands-on exploratory adjustments. Because these finer points of SBC procedure are still being actively tested and refined by Bayesian statistical researchers, we urge psychologists to keep up to date with the SBC literature if they intend to apply SBC to Bayesian cognitive models.

While SBC is currently unfamiliar in the cognitive modeling literature, we expect that SBC will begin to take the place of larger recovery simulations, and ultimately emerge as another key tool in the Bayesian cognitive modeling toolbox.

Model recovery

In research that involves the comparison of multiple models applied to the same experimentally-collected dataset, we recommend assessing in some way whether the planned model comparison will be capable of distinguishing among the candidate models. While

it would be ideal to rigorously test the accuracy and reliability of comparisons among Bayesian cognitive models, it is currently more common in Bayesian cognitive modeling to assess whether one can sufficiently recover the identity of the model that generated the data, given the set of candidate models (for a discussion of this distinction, see Lee, Gluck, et al., 2019). Whether you find either of the techniques we discuss here to be useful for troubleshooting will depend on your choice of model comparison metric, the goal of your planned comparison, and the exact expectation about model behavior that you intend to evaluate.

By far, the most prevalent approach used in Bayesian cognitive modeling research is the *model recovery study* (e.g., Pitt et al., 2003). The model recovery procedure is simple to explain: First, N datasets are to be simulated from each of M models. Then, for each dataset, all M models are applied and compared using a fully-Bayesian model comparison metric (e.g., WAIC or LOO; Vehtari et al., 2017). The output of the study is typically a contingency table summarizing the frequency with which each data-generating model was judged to be the best-fitting model (i.e., a confusion matrix, although other summaries and metrics may be used). If the models are sufficiently distinguishable, then the model comparisons should identify the true generating model for the majority of the $M \cdot N$ datasets. Even though model recovery is not a means to evaluate the quality of model-based inference (Lee, Gluck, et al., 2019; Schad et al., 2022), it may still be informative when the intent is only to perform a consistency check of the expectation for a model comparison to regularly identify the true generating model: If confusion matrix is confused, then this expectation of model inversion is not supported. In the context of troubleshooting, model recovery may also be quite useful as a means to explore the influence of experimental design on model comparison outcomes (for a recent example, see Evans & Brown, 2018).

If the planned method of model comparison is a Bayes factor (as is now feasible for Bayesian cognitive models via methods such as bridge sampling; Gronau et al., 2017), then a new, alternative approach is to perform SBC specifically for the Bayes factor comparison (Schad et al., 2022). Similar to how SBC is a principled way to quantify the internal consistency and accuracy of posterior estimates, *SBC for Bayes factors* is a principled way to quantify the expected accuracy for a planned comparison via Bayes factor, and uncover the degree of variability in that Bayes factor that one might reasonably expect to encounter.

Unfortunately, running either of these two procedures can require a considerable investment of time. Still, both techniques offer opportunities for directly troubleshooting comparisons among Bayesian cognitive models, unlike any other techniques in this tutorial. We expect that SBC for Bayes factors will find good use within the Bayesian cognitive modeling community, particularly in applied research where there may be a stronger need to plan for precision in the comparison of Bayesian cognitive models.

Posterior predictives

Another way to better understand the behavior of a Bayesian cognitive model is to perform a series of *posterior predictive checks* (Rubin, 1984; Gelman et al., 2013). As the name might suggest, posterior predictive checks (which are generated *after* the model has been applied to a particular behavioral dataset) are similar in many ways to prior predictive checks (which are generated *before* the model was exposed to any data). After the *posterior predictive distribution* has been generated by using each joint posterior sample in turn to

simulate a new dataset,[10] performing each posterior predictive check is simply a matter of comparing a summary statistic of the observed data to the same summary statistic across the posterior predictive distribution of data. For a Bayesian cognitive model, these summary statistics are expected to be both nontrivial and meaningful within the research context. As such, the same carefully chosen set of behavioral patterns and performance metrics can and ideally would be visualized in both the prior *and* posterior predictive checks (Berkhof et al., 2000).

In the context of a simulation study, the posterior predictive checks together are the final assessment of the internal consistency of a Bayesian cognitive model. If the model is behaving as intended, then the data used to estimate the parameters of the model should easily fall within the spread of the posterior predictive distribution of data based on those same estimates. If this is not the case, then further troubleshooting is needed to investigate in case this is signaling a serious problem in the model specification (or a serious error in the code).

When the model has been applied to an experimentally-collected dataset, posterior predictive checks should be used to evaluate the *descriptive adequacy* of the Bayesian cognitive model (Shiffrin et al., 2008). Each systematic discrepancy or *misfit* between the posterior predictive and the observed data can reveal a different way in which the model was unable to capture the true data-generating process (i.e., the cognitive process that participants actually used to perform the task). While formal discrepancy measures are regularly used in statistical linear modeling (e.g., posterior predictive p-values; Gelman et al., 1996), these posterior predictive checks are typically qualitative assessments only in Bayesian cognitive modeling. As such, how to characterize the severity and importance of any misfits will depend heavily on the research context. While misfits characterized as small may sometimes simply be acknowledged, numerous severe misfits may indicate that the model is neither useful nor valid as a model of the latent cognitive process (or, again, suggest a need for further troubleshooting).

At this late stage of troubleshooting, it can be frustrating to find that further loops of diagnostic investigation are required. While the troubleshooting process can be a long slog, a side-benefit is often that a rich understanding of the inner workings and behavior of the model is developed along the way. Through the wide variety of checks and assessments needed, you might have noticed trade-offs among key parameters; found conditions under which the model is unidentified; learned which estimates are especially sensitive or robust to their priors; discovered patterns of behavior that the model presently fails to capture; and so on. It can be fruitful to make note of these types of tendencies, strengths, and limitations, particularly if you might continue to use this particular Bayesian cognitive model in subsequent work.

After the current research project has come to a close, future projects might be both inspired and facilitated by this kind of knowledge. For example, one may begin a new

[10] It is important to note that this is different than using the collection of point parameter estimates for each parameter to simulate new data, which is an incorrect approach for Bayesian models. Using only the point estimates ignores the uncertainty associated with these estimates, and as such is unlikely to capture the full range of behavioral data seen as likely by the fitted model. (Also consider that the collection of marginal posterior point estimates is not necessarily a point in the joint parameter space that was visited during sampling, let alone guaranteed to be the most likely point in the joint parameter space.)

project later on to explore whether one or more theory-driven extensions to the model could account for patterns of behavior left unaccounted for by the original version of the model. In other words, the knowledge gained though troubleshooting process might guide a future *model development* process — but at this point, the lines between model checking, model adjustment, and model usage have become rather blurred. A number of the techniques in this tutorial, particularly in this last collection of troubleshooting procedures, are also important steps in a larger Bayesian cognitive modeling workflow that, like all Bayesian workflows (Gelman et al., 2020; Schad et al., 2021), may be used to work toward a wider variety of analytic and research purposes.

The troubleshooting process ends when no further problems of any kind are able to be detected, and as such, from both a computational perspective *and* a model consistency perspective, one is reasonably confident that any inferences one will make based on the final Bayesian cognitive model will be computationally sufficient, internally consistent, and reasonable for the task at hand.

Reporting results

When publishing results from research using Bayesian cognitive modeling, authors should explicitly mention that the required model checks were performed. It is not necessary to record and report exhaustively every detail of your troubleshooting and model development process (although this may be done as a "postregistration" of model-based work; Lee, Criss, et al., 2019). However, the final specification of the model that is being used and what diagnostic checks were performed should always be made clear (for an example, see Kruschke, 2021). All reports of results from Bayesian cognitive models should include the model specification (i.e., the likelihood and priors used), the sampling algorithm used (including any actively given sampler-specific inputs), the criteria used to evaluate the computational sufficiency of the model, and some reference to the checks of model consistency performed. An example of how this may be reported is:

> ... Finally, we performed prior predictive checks to demonstrate that each model specification was reasonable in the context of our experiment (see Figure X).
>
> With these model specifications and our behavioral data in hand, we used Stan (Carpenter et al., 2017) to estimate the joint posterior distribution of each model via dynamic Hamiltonian Monte Carlo sampling. For each model, we ran 4 chains of 500 warmup iterations and 1500 kept iterations each, then performed a series of diagnostic checks. We required an $\hat{R}$ value of ≤ 1.01 and an effective sample size of ≥ 400 for all parameters, a BFMI of ≥ 0.2 for all chains, and that no divergences were observed. When we report 90% credible intervals (equal-tailed), we also required effective sample size estimates of ≥ 400 for the 5% and 95% quantiles of those parameters. These checks were supported by a visual inspection of diagnostic and other plots. Only kept iterations from models that met these criteria were used for inference.

Conclusion

While Bayesian cognitive modeling can be a challenging method to use properly, it is also a rewarding approach to psychological research that is only increasing in popularity

(Jarecki et al., 2020; van de Schoot et al., 2017). In this tutorial, we have sought to make the troubleshooting process clear and accessible, especially for psychologists who may be new to Bayesian methods for cognitive modeling.

Of course, one may sometimes find it is justified or necessary to deviate from the exact recommendations outlined here. Practical concerns, including the time and computational power available, may shape the troubleshooting process in a number of ways. As touched on earlier, some Bayesian cognitive models can require a rather long time to run, and some more time-intensive troubleshooting procedures, such as SBC, may not always be feasible within a reasonable research timeline. Experienced practitioners of Bayesian cognitive modeling one may realize conditions under which it is reasonable to relax — or necessary to tighten — the criteria for certain diagnostic checks. The procedures recommended here are also not an exhaustive list of model-checking techniques: Further troubleshooting procedures including as prior sensitivity analyses, while beyond the scope of this tutorial, are important tools for certain applications.

While the exact sequence of troubleshooting steps needed will be different depending on one's choice of cognitive model, experimental design, and planned application, one should now have a firm enough grasp on the core tenets of Bayesian troubleshooting to investigate one's own models. One will not only now know the most essential steps — from the requisite automated computational checks, to more custom methods to ensure the model is functioning as intended, alongside the full investigative toolbox of diagnostic plots — but should also be able to judge the quality of the output at each step. Ultimately, it is our hope that this guide will not only encourage more vigilant and conscientious use of Bayesian cognitive models, but also might empower psychologists to build and apply Bayesian cognitive models in their own research with confidence in the quality of their work.

References

Ahn, W.-Y., Haines, N., & Zhang, L. (2017). Revealing neurocomputational mechanisms of reinforcement learning and decision-making with the hBayesDM package. *Computational Psychiatry*, *1*, 24–57.

Andrews, M., & Baguley, T. (2013). Prior approval: The growth of Bayesian methods in psychology. *British Journal of Mathematical and Statistical Psychology*, *66*(1), 1–7.

Annis, J., & Palmeri, T. J. (2018). Bayesian statistical approaches to evaluating cognitive models. *Wiley Interdisciplinary Reviews: Cognitive Science*, *9*(2), e1458.

Apgar, J. F., Witmer, D. K., White, F. M., & Tidor, B. (2010). Sloppy models, parameter uncertainty, and the role of experimental design. *Molecular BioSystems*, *6*(10), 1890–1900.

Baribault, B. (2021). *matstanlib: A library of MATLAB functions for visualization, processing, and analysis of output from Bayesian models* (Version 1.0) [MATLAB library]. https://github.com/baribault/matstanlib

Batchelder, W. H., & Riefer, D. M. (1999). Theoretical and empirical review of multinomial process tree modeling. *Psychonomic Bulletin & Review*, *6*(1), 57–86.

Berkhof, J., Van Mechelen, I., & Hoijtink, H. (2000). Posterior predictive checks: Principles and discussion. *Computational Statistics*, *15*(3), 337–354.

Betancourt, M. (2016). *Diagnosing suboptimal cotangent disintegrations in Hamiltonian Monte Carlo*. https://arxiv.org/abs/1604.00695

Betancourt, M. (2018). *A conceptual introduction to Hamiltonian Monte Carlo*. https://arxiv.org/abs/1701.02434

Betancourt, M., & Girolami, M. (2015). Hamiltonian Monte Carlo for hierarchical models. In S. Upadhyay, U. Singh, D. Dey, & A. Loganathan (Eds.), *Current trends in bayesian methodology with applications* (pp. 79–101). Chapman & Hall/CRC.

Blei, D. M., Kucukelbir, A., & McAuliffe, J. D. (2017). Variational inference: A review for statisticians. *Journal of the American Statistical Association*, *112*(518), 859–877.

Boehm, U., Marsman, M., Matzke, D., & Wagenmakers, E.-J. (2018). On the importance of avoiding shortcuts in applying cognitive models to hierarchical data. *Behavior Research Methods*, *50*(4), 1614–1631.

Box, G. E. (1980). Sampling and Bayes' inference in scientific modelling and robustness. *Journal of the Royal Statistical Society: Series A (General)*, *143*(4), 383–404.

Brooks, S. P. (2003). Bayesian computation: A statistical revolution. *Philosophical Transactions of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, *361*(1813), 2681–2697.

Brooks, S., Gelman, A., Jones, G., & Meng, X.-L. (2011). *Handbook of markov chain monte carlo*. Chapman & Hall/CRC.

Brown, V. M., Zhu, L., Solway, A., Wang, J. M., McCurry, K. L., King-Casas, B., & Chiu, P. H. (2021). Reinforcement learning disruptions in individuals with depression and sensitivity to symptom change following cognitive behavioral therapy. *JAMA Psychiatry*, *78*(10), 1113–1122.

Browne, W. J., Steele, F., Golalizadeh, M., & Green, M. J. (2009). The use of simple reparameterizations to improve the efficiency of Markov chain Monte Carlo estimation for multilevel models with applications to discrete time survival models. *Journal of the Royal Statistical Society: Series A (Statistics in Society)*, *172*(3), 579–598.

Bürkner, P.-C. (2017). *Advanced Bayesian multilevel modeling with the R package brms*. https://arxiv.org/abs/1705.11123

Busemeyer, J. R., Pothos, E. M., Franco, R., & Trueblood, J. S. (2011). A quantum theoretical explanation for probability judgment errors. *Psychological Review*, *118*(2), 193–218.

Carpenter, B., Gelman, A., Hoffman, M. D., Lee, D., Goodrich, B., Betancourt, M., Brubaker, M., Guo, J., Li, P., & Riddell, A. (2017). Stan: A probabilistic programming language. *Journal of Statistical Software*, *76*(1).

Chung, Y., Rabe-Hesketh, S., Dorie, V., Gelman, A., & Liu, J. (2013). A nondegenerate penalized likelihood estimator for variance parameters in multilevel models. *Psychometrika*, *78*(4), 685–709.

Cook, S. R., Gelman, A., & Rubin, D. B. (2006). Validation of software for Bayesian models using posterior quantiles. *Journal of Computational and Graphical Statistics*, *15*(3), 675–692.

Dai, C., Heng, J., Jacob, P. E., & Whiteley, N. (2022). An invitation to Sequential Monte Carlo samplers. *Journal of the American Statistical Association*, 1–38.

Danwitz, L., Mathar, D., Smith, E., Tuzsus, D., & Peters, J. (2022). Parameter and model recovery of reinforcement learning models for restless bandit problems. *Computational Brain & Behavior*.

Dearden, R., Friedman, N., & Andre, D. (1998). Bayesian Q-learning. *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, 761–768.

Donkin, C., Kary, A., Tahir, F., & Taylor, R. (2016). Resources masquerading as slots: Flexible allocation of visual working memory. *Cognitive Psychology*, *85*, 30–42.

Duane, S., Kennedy, A. D., Pendleton, B. J., & Roweth, D. (1987). Hybrid monte carlo. *Physics Letters B*, *195*(2), 216–222.

Efron, B., & Morris, C. (1977). Stein's paradox in statistics. *Scientific American*, *236*(5), 119–127.

Etz, A., Gronau, Q. F., Dablander, F., Edelsbrunner, P. A., & Baribault, B. (2018). How to become a Bayesian in eight easy steps: An annotated reading list. *Psychonomic Bulletin & Review*, *25*(1), 219–234.

Etz, A., & Vandekerckhove, J. (2018). Introduction to Bayesian inference for psychology. *Psychonomic Bulletin & Review*, *25*(1), 5–34.

Evans, N. J., & Brown, S. D. (2018). Bayes factors for the linear ballistic accumulator model of decision-making. *Behavior Research Methods*, *50*(2), 589–603.

Farrell, S., & Lewandowsky, S. (2018). *Computational modeling of cognition and behavior*. Cambridge University Press.

Frank, M. J., Gagne, C., Nyhus, E., Masters, S., Wiecki, T. V., Cavanagh, J. F., & Badre, D. (2015). fMRI and EEG predictors of dynamic decision parameters during human reinforcement learning. *Journal of Neuroscience*, *35*(2), 485–494.

Gabry, J., & Mahr, T. (2021). *bayesplot: Plotting for Bayesian models* (Version 1.8.0) [R package]. https://mc-stan.org/bayesplot/

Gabry, J., Simpson, D., Vehtari, A., Betancourt, M., & Gelman, A. (2019). Visualization in Bayesian workflow. *Journal of the Royal Statistical Society: Series A (Statistics in Society)*, *182*(2), 389–402.

Galdo, M., Bahg, G., & Turner, B. M. (2020). Variational Bayesian methods for cognitive science. *Psychological Methods*, *25*(5), 535–559.

Gelman, A. (2004). Parameterization and Bayesian modeling. *Journal of the American Statistical Association*, *99*(466), 537–545.

Gelman, A., Carlin, J. B., Stern, H. S., Dunson, D. B., Vehtari, A., & Rubin, D. B. (2013). *Bayesian data analysis* (3rd ed.). Chapman & Hall/CRC.

Gelman, A., Carlin, J. B., Stern, H. S., & Rubin, D. B. (1995). *Bayesian data analysis* (1st ed.). Chapman & Hall/CRC.

Gelman, A., & Hill, J. (2006). *Data analysis using regression and multilevel/hierarchical models*. Cambridge University Press.

Gelman, A., Meng, X.-L., & Stern, H. (1996). Posterior predictive assessment of model fitness via realized discrepancies. *Statistica Sinica*, 733–760.

Gelman, A., & Rubin, D. B. (1991). A single series from the Gibbs sampler provides a false sense of security. *Bayesian Statistics*, *4*, 625–631.

Gelman, A., & Rubin, D. B. (1992). Inference from iterative simulation using multiple sequences. *Statistical Science*, *7*(4), 457–472.

Gelman, A., Simpson, D., & Betancourt, M. (2017). The prior can often only be understood in the context of the likelihood. *Entropy*, *19*(10), 555.

Gelman, A., Van Dyk, D. A., Huang, Z., & Boscardin, J. W. (2008). Using redundant parameterizations to fit hierarchical models. *Journal of Computational and Graphical Statistics*, *17*(1), 95–122.

Gelman, A., Vehtari, A., Simpson, D., Margossian, C. C., Carpenter, B., Yao, Y., Kennedy, L., Gabry, J., Bürkner, P.-C., & Modrák, M. (2020). *Bayesian workflow*. https://arxiv.org/abs/2011.01808

Gilks, W. R., Richardson, S., & Spiegelhalter, D. (1995). *Markov chain Monte Carlo in practice*. Chapman & Hall/CRC.

Gluth, S., & Jarecki, J. B. (2019). On the importance of power analyses for cognitive modeling. *Computational Brain & Behavior*, *2*(3), 266–270.

Golubickis, M., Falben, J. K., Cunningham, W. A., & Macrae, C. N. (2018). Exploring the self-ownership effect: Separating stimulus and response biases. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, *44*(2), 295–306.

Greene, N. R., & Rhodes, S. (2022). A tutorial on cognitive modeling for cognitive aging research. *Psychology and Aging*, *37*(1), 30–42.

Griffiths, T. L., Kemp, C., & Tenenbaum, J. B. (2008). Bayesian models of cognition. In R. Sun (Ed.), *Cambridge handbook of computational psychology* (pp. 59–100). Cambridge University Press.

Gronau, Q. F., Sarafoglou, A., Matzke, D., Ly, A., Boehm, U., Marsman, M., Leslie, D. S., Forster, J. J., Wagenmakers, E.-J., & Steingroever, H. (2017). A tutorial on bridge sampling. *Journal of Mathematical Psychology*, *81*, 80–97.

Gunawan, D., Hawkins, G. E., Tran, M.-N., Kohn, R., & Brown, S. (2020). New estimation approaches for the hierarchical Linear Ballistic Accumulator model. *Journal of Mathematical Psychology*, *96*, 102368.

Haines, N., Beauchaine, T. P., Galdo, M., Rogers, A. H., Hahn, H., Pitt, M. A., Myung, J. I., Turner, B. M., & Ahn, W.-Y. (2020). Anxiety modulates preference for immediate rewards among trait-impulsive individuals: A hierarchical Bayesian analysis. *Clinical Psychological Science*, *8*(6), 1017–1036.

Hartmann, R., & Klauer, K. C. (2020). Extending RT-MPTs to enable equal process times. *Journal of Mathematical Psychology*, *96*, 102340.

Heathcote, A., Brown, S. D., & Wagenmakers, E.-J. (2015). An introduction to good practices in cognitive modeling. In B. U. Forstmann & E.-J. Wagenmakers (Eds.), *An introduction to model-based cognitive neuroscience* (pp. 25–48). Springer.

Heathcote, A., Lin, Y.-S., Reynolds, A., Strickland, L., Gretton, M., & Matzke, D. (2019). Dynamic models of choice. *Behavior Research Methods*, *51*(2), 961–985.

Hoffman, M. D., & Gelman, A. (2014). The No-U-Turn sampler: Adaptively setting path lengths in Hamiltonian Monte Carlo. *Journal of Machine Learning Research*, *15*(1), 1593–1623.

Jarecki, J. B., Tan, J. H., & Jenny, M. A. (2020). A framework for building cognitive process models. *Psychonomic Bulletin & Review*, *27*(1), 1218–1229.

Jasra, A., Holmes, C. C., & Stephens, D. A. (2005). Markov chain Monte Carlo methods and the label switching problem in Bayesian mixture modeling. *Statistical Science*, *20*(1), 50–67.

Katahira, K. (2016). How hierarchical models improve point estimates of model parameters at the individual level. *Journal of Mathematical Psychology*, *73*, 37–58.

Kennedy, L., Simpson, D., & Gelman, A. (2019). The experiment is just as important as the likelihood in understanding the prior: A cautionary note on robust cognitive modeling. *Computational Brain & Behavior*, *2*(3), 210–217.

Klauer, K. C., Singmann, H., & Kellen, D. (2015). Parametric order constraints in multinomial processing tree models: An extension of Knapp and Batchelder (2004). *Journal of Mathematical Psychology*, *64*, 1–7.

Knapp, B. R., & Batchelder, W. H. (2004). Representing parametric order constraints in multi-trial applications of multinomial processing tree models. *Journal of Mathematical Psychology*, *48*(4), 215–229.

Krefeld-Schwalb, A., Pachur, T., & Scheibehenne, B. (2022). Structural parameter interdependencies in computational models of cognition. *Psychological Review*, *129*(2), 313–339.

Kruschke, J. K. (2014). *Doing Bayesian data analysis: A tutorial with R, JAGS, and Stan*. Academic Press.

Kruschke, J. K. (2021). Bayesian analysis reporting guidelines. *Nature Human Behaviour*, *5*(10), 1282–1291.

Kumar, R., Carroll, C., Hartikainen, A., & Martin, O. A. (2019). ArviZ a unified library for exploratory analysis of Bayesian models in Python. *Journal of Open Source Software*, *4*(33), 1143.

Lambert, B., & Vehtari, A. (2022). R*: A robust MCMC convergence diagnostic with uncertainty using decision tree classifiers. *Bayesian Analysis*, *17*(2), 353–379.

Lasagna, C. A., Pleskac, T. J., Burton, C. Z., McInnis, M. G., Taylor, S. F., & Tso, I. F. (2022). Mathematical modeling of risk-taking in bipolar disorder: Evidence of reduced behavioral consistency, with altered loss aversion specific to those with history of substance use disorder. *Computational Psychiatry*, *6*(1), 96–116.

Lee, M. D. (2008). Three case studies in the Bayesian analysis of cognitive models. *Psychonomic Bulletin & Review*, *15*(1), 1–15.

Lee, M. D. (2011). How cognitive modeling can benefit from hierarchical Bayesian models. *Journal of Mathematical Psychology*, *55*(1), 1–7.

Lee, M. D. (2018). Bayesian methods in cognitive modeling. In J. T. Wixted & E.-J. Wagenmakers (Eds.), *The stevens' handbook of experimental psychology and cognitive neuroscience* (pp. 37–84).

Lee, M. D., Criss, A. H., Devezer, B., Donkin, C., Etz, A., Leite, F. P., Matzke, D., Rouder, J. N., Trueblood, J. S., White, C. N., et al. (2019). Robust modeling in cognitive science. *Computational Brain & Behavior*, *2*(3), 141–153.

Lee, M. D., Gluck, K. A., & Walsh, M. M. (2019). Understanding the complexity of simple decisions: Modeling multiple behaviors and switching strategies. *Decision*, *6*(4), 335.

Lee, M. D., & Vanpaemel, W. (2018). Determining informative priors for cognitive models. *Psychonomic Bulletin & Review*, *25*(1), 114–127.

Lee, M. D., & Wagenmakers, E.-J. (2013). *Bayesian cognitive modeling: A practical course*. Cambridge University Press.

Link, W. A., & Eaton, M. J. (2012). On thinning of chains in MCMC. *Methods in Ecology and Evolution*, *3*(1), 112–115.

Livingstone, S., Betancourt, M., Byrne, S., & Girolami, M. (2019). On the geometric ergodicity of hamiltonian monte carlo. *Bernoulli*, *25*(4A), 3109–3138.

Matzke, D., Dolan, C. V., Batchelder, W. H., & Wagenmakers, E.-J. (2015). Bayesian estimation of multinomial processing tree models with heterogeneity in participants and items. *Psychometrika*, *80*(1), 205–235.

Monnahan, C. C., Thorson, J. T., & Branch, T. A. (2017). Faster estimation of bayesian models in ecology using hamiltonian monte carlo. *Methods in Ecology and Evolution*, *8*(3), 339–348.

Navarro, D. J. (2021). If mathematical psychology did not exist we might need to invent it: A comment on theory building in psychology. *Perspectives on Psychological Science*, *16*(4), 707–716.

Navarro, D. J., Newell, B. R., & Schulze, C. (2016). Learning and choosing in an uncertain world: An investigation of the explore–exploit dilemma in static and dynamic environments. *Cognitive Psychology*, *85*, 43–77.

Neal, R. M. (2011). MCMC using Hamiltonian dynamics. In S. Brooks, A. Gelman, G. Jones, & X.-L. Meng (Eds.), *Handbook of markov chain monte carlo* (pp. 113–160). Chapman & Hall/CRC.

Nilsson, H., Rieskamp, J., & Wagenmakers, E.-J. (2011). Hierarchical bayesian parameter estimation for cumulative prospect theory. *Journal of Mathematical Psychology*, *55*(1), 84–93.

Nunez, M. D., Gosai, A., Vandekerckhove, J., & Srinivasan, R. (2019). The latency of a visual evoked potential tracks the onset of decision making. *Neuroimage*, *197*, 93–108.

Park, H., Yang, J., Vassileva, J., & Ahn, W.-Y. (2021). Development of a novel computational model for the Balloon Analogue Risk Task: The exponential-weight mean–variance model. *Journal of Mathematical Psychology*, *102*, 102532.

Peters, J., & D'Esposito, M. (2020). The drift diffusion model as the choice rule in intertemporal and risky choice: A case study in medial orbitofrontal cortex lesion patients and controls. *PLoS Computational Biology*, *16*(4), e1007615.

Pitt, M. A., Kim, W., & Myung, I. J. (2003). Flexibility versus generalizability in model selection. *Psychonomic Bulletin & Review*, *10*(1), 29–44.

Pleskac, T. J., Cesario, J., & Johnson, D. J. (2018). How race affects evidence accumulation during the decision to shoot. *Psychonomic Bulletin & Review*, *25*(4), 1301–1330.

Plummer, M. (2003). Jags: A program for analysis of Bayesian graphical models using Gibbs sampling. In K. Hornik, F. Leisch, & A. Zeileis (Eds.), *Proceedings of the 3rd international workshop on distributed statistical computing* (pp. 1–10).

Ratcliff, R., & McKoon, G. (2008). The diffusion decision model: Theory and data for two-choice decision tasks. *Neural Computation*, *20*(4), 873–922.

Robert, C., & Casella, G. (2011). A short history of Markov chain Monte Carlo: Subjective recollections from incomplete data. In S. Brooks, A. Gelman, G. Jones, & X.-L. Meng (Eds.), *Handbook of markov chain monte carlo* (pp. 49–67). Chapman & Hall/CRC.

Rouder, J. N., & Lu, J. (2005). An introduction to Bayesian hierarchical models with an application in the theory of signal detection. *Psychonomic bulletin & review*, *12*(4), 573–604.

Rouder, J. N., Speckman, P. L., Sun, D., Morey, R. D., & Iverson, G. (2009). Bayesian t tests for accepting and rejecting the null hypothesis. *Psychonomic Bulletin & Review*, *16*(2), 225–237.

Röver, C., Bender, R., Dias, S., Schmid, C. H., Schmidli, H., Sturtz, S., Weber, S., & Friede, T. (2021). On weakly informative prior distributions for the heterogeneity parameter in bayesian random-effects meta-analysis. *Research Synthesis Methods*, *12*(4), 448–474.

Rubin, D. B. (1984). Bayesianly justifiable and relevant frequency calculations for the applied statistician. *The Annals of Statistics*, 1151–1172.

Salvatier, J., Wiecki, T. V., & Fonnesbeck, C. (2016). Probabilistic programming in Python using PyMC3. *PeerJ Computer Science*, *2*, e55.

Schad, D. J., Betancourt, M., & Vasishth, S. (2021). Toward a principled Bayesian workflow in cognitive science. *Psychological Methods*, *26*(1), 103–126.

Schad, D. J., Nicenboim, B., Bürkner, P.-C., Betancourt, M., & Vasishth, S. (2022). Workflow techniques for the robust use of Bayes factors. *Psychological Methods*.

Schaper, M. L., Mieth, L., & Bell, R. (2019). Adaptive memory: Source memory is positively associated with adaptive social decision making. *Cognition*, *186*, 7–14.

Scheibehenne, B., & Pachur, T. (2015). Using Bayesian hierarchical parameter estimation to assess the generalizability of cognitive models of choice. *Psychonomic Bulletin & Review*, *22*(2), 391–407.

Shiffrin, R. M., Lee, M. D., Kim, W., & Wagenmakers, E.-J. (2008). A survey of model evaluation approaches with a tutorial on hierarchical Bayesian methods. *Cognitive Science*, *32*(8), 1248–1284.

Spektor, M. S., & Kellen, D. (2018). The relative merit of empirical priors in non-identifiable and sloppy models: Applications to models of learning and decision-making. *Psychonomic Bulletin & Review*, *25*(6), 2047–2068.

Stan Development Team. (2022). *Stan modeling language users guide and reference manual* (Version 2.30). https://mc-stan.org

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT Press.

Talts, S., Betancourt, M., Simpson, D., Vehtari, A., & Gelman, A. (2020). *Validating Bayesian inference algorithms with simulation-based calibration*. https://arxiv.org/abs/1804.06788

Tran, N.-H., Van Maanen, L., Heathcote, A., & Matzke, D. (2021). Systematic parameter reviews in cognitive modeling: Towards a robust and cumulative characterization of psychological processes in the diffusion decision model. *Frontiers in Psychology*, *11*, 608287.

Turner, B. M., Sederberg, P. B., Brown, S. D., & Steyvers, M. (2013). A method for efficiently sampling from distributions with correlated dimensions. *Psychological Methods*, *18*(3), 368–384.

van de Schoot, R., Winter, S. D., Ryan, O., Zondervan-Zwijnenburg, M., & Depaoli, S. (2017). A systematic review of Bayesian articles in psychology: The last 25 years. *Psychological Methods*, *22*(2), 217–239.

Van Ravenzwaaij, D., Cassey, P., & Brown, S. D. (2018). A simple introduction to Markov Chain Monte-Carlo sampling. *Psychonomic Bulletin & Teview*, *25*(1), 143–154.

Vandekerckhove, J., Tuerlinckx, F., & Lee, M. (2008). A bayesian approach to diffusion process models of decision-making. *Proceedings of the 30th annual conference of the Cognitive Science Society*, 1429–1434.

Vanpaemel, W. (2010). Prior sensitivity in theory testing: An apologia for the Bayes factor. *Journal of Mathematical Psychology*, *54*(6), 491–498.

Vehtari, A., Gelman, A., & Gabry, J. (2017). Practical Bayesian model evaluation using leave-one-out cross-validation and WAIC. *Statistics and Computing*, *27*(5), 1413–1432.

Vehtari, A., Gelman, A., Simpson, D., Carpenter, B., & Bürkner, P.-C. (2021). Rank-normalization, folding, and localization: An improved $\hat{R}$ for assessing convergence of MCMC. *Bayesian Analysis*, *16*(2), 667–718.

Westfall, H. A., & Lee, M. D. (2021). A model-based analysis of the impairment of semantic memory. *Psychonomic Bulletin & Review*, *28*(5), 1484–1494.

Wilson, R. C., & Collins, A. G. (2019). Ten simple rules for the computational modeling of behavioral data. *eLife*, *8*, e49547.

Appendix
Code libraries to support troubleshooting in R, Python, and MATLAB

Libraries that facilitate many of the troubleshooting procedures recommended here are freely available in each of the three programming languages — R, Python, and MATLAB — that are most commonly used for Bayesian cognitive modeling. As of this writing (8/2022), the most comprehensive support libraries available are `bayesplot` in R (Gabry & Mahr, 2021), `ArviZ` in Python (and Julia; Kumar et al., 2019), and `matstanlib` in MATLAB (Baribault, 2021).

Below, we outline exactly how these support libraries may be used to automate the computations and plots described in the main text.

Using `matstanlib` for troubleshooting in MATLAB

While an abundance of tutorials are available that demonstrate the use of `bayesplot` and `ArviZ`, scarcely any demonstrate use of `matstanlib` as it has just recently been released. As such, we offer a brief walkthrough of how this new resource may be used to automate and/or support each the troubleshooting procedures we recommend. However, the sequence of steps is the same in R and Python (see Table A1 below for analogous commands in `bayesplot` and `ArviZ`).

After running the model, it is often necessary to convert the output to a format that is compatible with the support library. For MATLAB users, neither the recognized MATLAB interface to Stan, MatlabStan (https://github.com/brian-lau/MatlabStan), nor the alternative interface, Trinity (https://github.com/joachimvandekerckhove/trinity), returns samples in the format that is required by `matstanlib`. `matstanlib`'s `extractsamples.m` function automates the reformatting process for output from either interface:

```
[samples,diagnostics] = extractsamples('MatlabStan',fit);

[samples,diagnostics] = extractsamples('trinity',chains,info);
```

while ensuring that chain identity and the iteration order is faithfully maintained. Posterior samples and sampler diagnostics are returned in separate structures.

The required computational checks are fully automated by `matstanlib`. First, those computational diagnostics that are based on the posterior samples ($\hat{R}$ and ESS), along with some basic posterior summary statistics, are computed and collected in a table:

```
posteriorTable = mcmctable(samples);
```

Then, an automated assessment of all HMC/NUTS diagnostics may be printed at the command line:

```
interpretdiagnostics(diagnostics,posteriorTable)
```

This report will include a warning if any of the required computational checks for HMC/NUTS sampling ($\hat{R}$, ESS, divergences, BFMI) were not passed.

A subset of the required consistency checks are automated by `matstanlib`. While prior predictive checks must be programmed manually, as they require customization to the research context, parameter recovery checks are supported. After running a simulation study, recovery plots may be generated using the `matstanlib`'s `plotrecovery.m` function, as we used to create all six recovery plots in Figure 9.

If any problems are detected through these checks, a variety of diagnostic plots are needed to investigate further. Nearly all of the diagnostic plots mentioned in the tutorial are available in `matstanlib`.

`matstanlib` offers trace plots and rank plots to support troubleshooting high $\hat{R}$, and ESS plots to support troubleshooting low ESS. The trace plots in Figure 3, Figure 4 (top), and Figure 5b and the rank plots in Figure 4 (bottom) were generated by `matstanlib`'s `tracedensity.m` and `rankplots.m` functions, respectively. Each trio of ESS plots in Figure 7 was generated by `matstanlib`'s `plotess.m` function.

`matstanlib` also offers diagnostic plots for troubleshooting each diagnostics specific to HMC/NUTS, including BFMI and divergences. Low BFMI warnings can be investigated using energy plots; the energy plots in Figure 6a and 6b which were generated by `matstanlib`'s `plotenergy.m` function. The visualization of divergent transitions by chain in Figure 5a, generated by the `plotdivergences.m` function, and the parallel coordinate plot of samples in Figure 10b, generated by the `parallelsamples.m` function, are useful to recognize if divergences are more concentrated in the samples from a particular chain, or in a specific part of the joint parameter space, respectively.

A wide variety of other `matstanlib` functions support visual exploration of the joint posterior densities and estimates. As many of these functions can accept optional inputs to trigger diagnostic overlays, they may simultaneously support troubleshooting of HMC/NUTS diagnostics. To demonstrate the effect of reparameterization on a funnel-shaped density, we used the `jointdensity.m` function to generate the bivariate density plots, with indicators for divergent transitions (if any) overlaid, as seen in Figure 5c and 5d. The grids of bivariate densities for multiple conjunctions of parameters in Figure 6c and Figure 10a were generated by the `multidensity.m` function. Using optional inputs for `multidensity.m` to request that the energy diagnostic was included in Figure 6c and that divergence indicators were overlaid in Figure 10a these plots especially useful for troubleshooting low BFMI and divergent transitions, respectively.

Diagnostic overlays are also able to be included on a number of other `matstanlib` plots. For example, to add a rug plot of iterations with divergences or the maximum treedepth was reached to a trace plot, as we did for Figure 5b, the structure of diagnostic quantities must be given as an additional input:

```
tracedensity(samples,parameterNames,diagnostics)
```

To see the most recent documentation for any `matstanlib` function, including a full list of the optional inputs that are available, simply run the `help` command at the command line:

```
help plotess
```

Finally, `matstanlib` offers limited support for improving model specifications. To support the elicitation of new hyperpriors, prior simulation is automated through the `hyperpriortester.m` function, which was used to generate both panels in Figure 11. A small number of post hoc reparameterizations for hyperparameters of select distributions are automated by the `hypertransform.m` function.

All of the supplementary code mentioned in the main text (`example_RL.m`, `example_funnel.m`, `RL_broken.stan`, `RL_fixed.stan`) is available in the `examples` folder of `matstanlib`.

The `matstanlib` library is freely available from https://github.com/baribault/matstanlib.

Table A1

Essential commands for troubleshooting in various programming languages' support libraries.

	command name		
	MATLAB `matstanlib`	R `bayesplot`	Python `ArviZ`
Core functionality			
reformat samples and diagnostics	`extractsamples`	`as.array`, `nuts_params`	`from_pystan`, etc.
generate a table of posterior statistics and convergence diagnostics	`mcmctable`	`monitor`	`summary`
diagnostics report	`interpretdiagnostics`	`check_hmc_diagnostics`	—
Diagnostic plots			
trace plot	`tracedensity`	`mcmc_trace`, `mcmc_combo`	`plot_trace`
rank plots	`rankplots`	`mcmc_rank_hist`	`plot_rank`
divergences by chain	`plotdivergences`	—	—
energy plot	`plotenergy`	`mcmc_nuts_energy`	`plot_energy`
ESS diagnostic plots	`plotess`	—	`plot_ess`
bivariate density with marginals	`jointdensity`	`mcmc_scatter`	`plot_pair`
grid of bivariate densities	`multidensity`	`mcmc_pairs`	`plot_pair`
parallel coordinates plot	`parallelsamples`	`mcmc_parcoord`	`plot_parallel`
parameter recovery plot	`plotrecovery`	`mcmc_recover_scatter`	—
Other functionality			
prior simulation	`hyperpriortester`	—	—
posthoc application of select known reparameterizations	`hypertransform`	—	—

Function names in gray indicate the command is from the main interface package (i.e., Rstan, PyStan) as similar functionality is not included in the support package (i.e., is not available in `bayesplot` or `ArviZ`). If no function name is given, then as of this writing (8/2022), there is no counterpart in either the interface package or the specified support package.