universität wien

MASTERARBEIT / MASTER'S THESIS

Titel der Masterarbeit / Title of the Master's Thesis

"Bayesian Workflow for Cognitive Scientists - An Accessible Tutorial"

verfasst von / submitted by

Jakob Weickmann, MA BSc

angestrebter akademischer Grad / in partial fulfillment of the requirements for the degree of

Master of Science (MSc)

Wien, 2021 / Vienna 2021

Studienkennzahl lt. Studienblatt / degree programme code as it appears on the student record sheet:	A 066 840
Studienrichtung lt. Studienblatt / degree programme as it appears on the student record sheet:	Masterstudium Psychologie / Master's degree programme Psychology
Betreut von / Supervisor:	Prof. Dr. Claus Lamm
Mitbetreut von / Co-Supervisor:	Dr. Lei Zhang

Acknowledgements

We are indebted to the communities behind the multiple open-source software packages on which we depend. I want to express my gratitude to all those, who contribute to free and open software.

I am grateful to both my supervisor, Claus Lamm, and my co-supervisor, Lei Zhang, who have supported me during this research project and given constructive feedback where needed. I also want to thank Damian Bednarz, who has spent many hours together with me, tinkering on the code and helping me make sense of it all.

I am more than grateful for my parents, who have always supported my studies and been there for me, no matter what. Thank you! And finally, I want to thank my dear wife, Yumeho, for her unconditional support and love.

Contents

Abstract

Going from an observable effect back to its cause, this was the dream of reverend Thomas Bayes, father of the Bayes' theorem. The Bayesian approach is a statistical framework that allows for powerful yet flexibel modelling processes and analyses. In Bayesian models, existing domain expertise is integrated via the choice of prior probability distributions, making Bayesian models robust in their parameter estimation, even when the data samples are small. This makes Bayesian statistics especially attractive in the cognitive sciences, where prior knowledge is relatively abundant, yet sample sizes are usually small (Lee, 2011; van de Schoot et al., 2021). Therefore it is not surprising, that Bayesian statistics have become a popular and compelling choice for cognitive scientists, especially when it comes to hierarchical computational modelling (Nicenboim & Vasishth, 2016; van de Schoot, Winter, Ryan, Zondervan-Zwijnenburg, & Depaoli, 2017). However, although the popularity of Bayesian statistics is has been growing alongside the increasing accessibility of high-performing computing hardware and software, there is still a lack of accessible and comprehensive tutorials for cognitive scientists that demonstrate the implementation and application of a full Bayesian workflow for statistical and cognitive modelling. The first steps can be intimidating, as Bayesian modelling is sensitive to the choice of prior and the formulation of the generative statistical model. In this tutorial, I demonstrate a full Bayesian workflow with two examples - a linear regression model as well as a more complex hierarchical reinforcement learning model - using *R* and the probabilistic programming language *Stan*. This tutorial should be particularly useful for all those who want to begin making their own computational models within the Bayesian framework.

Zusammenfassung

Von einem beobachtbaren Effekt auf dessen Ursache zu schließen, das war der Traum des Pastoren Thomas Bayes, Begründer des Bayestheorems. Der Bayesianische Ansatz ist ein statistisches Framework, welches mächtige und doch flexible Modellierungsprozesse und Analysen erlaubt. Dieses Framework erlaubt die Integrierung von Vorwissen und fachlicher Expertise in das Modell in Form von a-priori Wahrscheinlichkeitsverteilungen. Dies trägt dazu bei, dass Bayesianische Modelle selbst bei kleinen Stichprobengrößen besonders robust in der Parameterschätzung sind (Lee, 2011; van de Schoot et al., 2021). Daher ist es nicht verwunderlich, dass Bayesianische Statistik eine beliebte und verlockende Wahl für Kognitionswissenschaftler*innen geworden ist, insbesondere auch für hierarchische Computermodelle (Nicenboim & Vasishth, 2016; van de Schoot, Winter, Ryan, Zondervan-Zwijnenburg, & Depaoli, 2017). Auch wenn die Beliebtheit Bayesianischer Statistik gemeinsam mit der Entwicklung leistungsfähiger Rechenhardware und -software zugenommen haben mag, besteht noch immer einen Mangel an umfassenden und zugänglichen Tutorials, die die Implementierung and Anwendung eines vollständigen Bayesianischen Workflows für statistische und kognitive Modelle vermitteln. Die ersten Schritten können durchaus einschüchternd sein, da das Bayesianische Modellieren anfällig für die Wahl der a-priori-Verteilungen sowie für die Formulierung des passenden generativen statistischen Modells ist. In diesem Tutorial demonstriere ich einen vollständigen Bayesianischen Workflow in *R* und der probabilistischen Programmiersprache *Stan* anhand zweier Beispiele: Einer linearen Regression sowie eines komplexeren hierarchischen Modells von bestärkendem Lernen (*reinforcement learning*). Dieses Tutorial sollte besonders hilfreich für all jene sein, die anfangen möchten, eigene Bayesianische Rechenmodelle zu erstellen.

1. Introduction

1.1 Structure

This tutorial provides an introduction to Bayesian statistics, illustrating the workflow with two example models, a simple linear regression, and a more complex hierarchical reinforcement learning model. After a brief introduction to the history of the Bayes' theorem and the principles of Bayesian statistics, we begin with the creation of the linear regression model and the process of eliciting appropriate prior distributions. Then we will see how the model is programmed in *Stan*, a Bayesian programming language that interfaces with the popular statistics software *R*. We will see how the model can be evaluated, how the results can be reported, and how prior predictive, as well as posterior predictive checks, are performed. Subsequently, we will see how more complex models, like a hierarchical Rescorla-Wagner reinforcement learning model can be fitted in *Stan*.

1.2 A brief history of the Bayes' theorem

Reverend Thomas Bayes was born in 1701 to the Presbyterian minister Joshua Bayes, who oversaw a chapel in London. As Presbyterians were considered non-conformist at that time, Thomas Bayes was not able to attend university in England. This is how he wound up going to the University of Edinburgh in Scotland where he studied theology. After completing his studies he returned to London and worked as an assistant in his father's ministry in London.

In 1734, he moved to Tunbridge Wells, a wealthy spa resort town, at the age of 33 and became minister of the local Mount Sion chapel. There he searched for a mathematical way to go from an effect back to its cause. For this purpose, he designed the following thought experiment, where he attempts to guess the position of a ball on a table, to which he has his back turned. A friend first throws one ball onto the table. Then he throws

a second ball and reports whether the second ball had landed to the left or the right of the first ball. Thomas Bayes made the important assumption that all positions of the table have an equal probability. When the second ball landed to the *right* of the first ball, Bayes guessed that the first ball was more likely to be on the left half of the table. In the opposite case, when the second ball was reported to be to the *left* of the first ball, then Bayes concluded that it was more likely for the first ball to be on the right half of the table. By throwing a third ball, fourth ball and so on, repeating the process, one would receive a likelihood of where the first ball landed. This is only true if all positions on the table are equally likely. Just like with increasing sample size, the confidence about the true position of the first ball increases with the number of throws. In this way, Bayes found a mathematical way to get from an effect (the result of the throws) to a probable cause (the position of the ball on the table) (Lambert, 2018; McGrayne, 2012).

Even though his ideas were discussed by members of the Royal Society, he never published his work. It was not until after his death, that his young friend Richard Price, who was also a Presbyterian minister, was invited by Bayes' family to examine his mathematical papers and rediscovered this idea. Religiously more fervent than Bayes, Price saw his ideas as a possible way to counter Hume's attack on causation, which posed a threat to the core of the Christian philosophy. He worked two years on compiling the papers and then sent them to the Royal Society with a cover letter of religious content. It is noteworthy, that Bayes never mentioned religion in the context of this thought experiment. The Royal Society then published this work under the secular title 'An Essay towards solving a Problem in the Doctrine of Chances.'

While Bayes can be credited with the initial conceptualization of the Bayes' rule, he was not the one to develop the modern mathematical notation that we use today. He also did not intend this rule as proof of god, unlike his friend Price. Finally, it can be assumed that he was not convinced of the importance of his work as he never published it.

At the same time in France, Pierre Simon Laplace, born in 1749, studied theology at the University of Caen. There, his mathematical aptness was recognized and he began to focus on mathematics instead of theology. Laplace is well known for his important work in many different fields of mathematics, including analysis, differential equations, planetary orbits, and potential theory. Independently of Bayes, he had begun to work on a probabilistic way to go from an effect back to its cause.

In 1774 he postulated: "If an event can be produced by a number n of different causes,

then the probabilities of these causes given the event are to each other as the probabilities of the event given the causes, and the probability of the existence of each of these is equal to the probability of the event given that cause, divided by the sum of all the probabilities of the event given each of these causes" (Laplace, 1986).

At the end of his paper, he notes that it is important to note whether we assume that the initial probabilities are equal for the outcomes. "For example, if at the game of heads and tails B wagers with A that head will not occur on either of two tosses, the probability that B will win is composite, since it is the probability that a head will not occur on the first toss, and that it will not occur on the second toss, multiplied together. Now, in this case, the probability for B from the ordinary theory is $\frac{1}{4}$, while if instead we suppose heads and tails unequally possible, this probability is larger than $\frac{1}{4}$."

This difference in prior possibilities is recognized by Laplace as essential for the application of probability theory to the "different objects of civil life." As in modern Bayesian analysis, the assumed probabilities for each of the outcomes can be modified with the setting of the prior and allows us to adjust our analysis to the more complex probability structures that are presented by our everyday lives.

When Price visits Paris in 1781 he tells the Secretary of the French Royal Academy of Sciences, Marquis of Condorcet, about Bayes' discovery. Once this information reaches Laplaces he is reinforced in his pursuit of a probabilistic way to go from an effect back to its cause. However, as anyone familiar with Bayesian analysis knows, this approach requires a large amount of calculation. Laplace was therefore looking for a real-life application for his method, that was simple enough to be calculated. He eventually settled on the number of male and female babies born in Paris between 1745 and 1770. This is a large enough sample with around 500,000 entries of binary data. After running his computation he concluded with regards to the birth rate that the probability of boys exceeding girls was "*as certain as any moral truth* with an extremely tiny margin of being wrong" (McGrayne, 2012).

In 1820 Laplace then published a modern mathematical rule that looks similar to our modern notation of Bayes' rule (eq. 1.1).

$$P = \frac{Hp}{S.Hp} \tag{1.1}$$

Here, P on the left-hand side denotes the posterior probability. On the right-hand

side, H stands for the likelihood and p stands for the prior. Finally, in the denominator, $S.$ is an old notation for the sum $\sum$ of all possible causes, i.e. $p(data)$, referred to as the marginal probability of the data or simply as the *evidence*.

A more modern notation of the Bayes' theorem is shown in equation 1.2.

$$p(\theta|data) = \frac{p(data|\theta)p(\theta)}{p(data)} \tag{1.2}$$

Table 1.1 provides a short overview of the central elements of the Bayes' theorem.

The marginal probability $p(data)$ describes the probability of observing the $data$ without any given conditions, most importantly without θ. Therefore, $data$ and θ must be different events. In order to calculate the marginal probability, we can apply the *law of total probability*. It is a theorem for discrete data space and it allows us to calculate the marginal probability (eq. 1.3) as follows.

$$p(data) = \sum\nolimits_n p(A|B_n) \tag{1.3}$$

In this way, we can convert and obtain the Bayes' theorem for discrete variables (eq. 1.4).

$$p(\theta|data) = \frac{p(data|\theta)p(\theta)}{\sum\nolimits_{\theta^*} p(data|\theta^*)p(\theta^*)} \tag{1.4}$$

In the above notation of the Bayes' theorem (eq. 1.4) the divisor consists of a sum. It is therefore only valid for discrete variables.

Of course, there is also a notation for continuous variables, where the sum is replaced by an integral (eq. 1.5).

$$p(\theta|data) = \frac{p(data|\theta)p(\theta)}{\int\nolimits_{\theta^*} p(data|\theta^*)p(\theta^*)} \tag{1.5}$$

Looking back at the historical background of the inception of the Bayes' rule, I believe it is fair to say that history has not sufficiently recognized the efforts of both Price and Laplace. Without them, Bayesian statistics would surely not be where they are today. I would like to give credit to both of them by mentioning them in this thesis as their contribution to Bayesian statistics has not found the widespread appreciation that they deserved.

Table 1.1: Glossary of Terms.

Term	Notation	Meaning
Prior (distribution)	$p(\theta)$	Beliefs held by researchers about the parameters in a statistical model before seeing the data, expressed as probability distributions.
Posterior (distribution)	$p(\theta\|data)$	A way to summarize one's updated knowledge, balancing prior knowledge with observed data.
Likelihood	$p(data\|\theta)$	The conditional probability distribution of the given parameters of the data.
Marginal probability (evidence)	$p(data)$	Probability of observing the data without any given conditions

1.3 The relevance of Bayesian statistics for cognitive scientists

Making Bayesian methods part of one's statistical toolkit carries many benefits due to the numerous advantages of this framework. These include easier interpretation of results relative to research hypotheses, and flexible model specification (Nicenboim & Vasishth, 2016). Therefore it is not surprising, that Bayesian statistics are gaining increasing popularity among cognitive scientists (van de Schoot, Winter, Ryan, Zondervan-Zwijnenburg, & Depaoli, 2017). The Bayesian approach allows for the integration of existing domain expertise via a prior distribution to obtain the posterior distribution. Bayesian models are robust in their parameter estimation and produce more valid predictions, in particular when the data samples are small. This makes Bayesian statistics especially attractive in the cognitive sciences, where existing knowledge is relatively abundant but sample sizes tend to be rather small (Lee, 2011; van de Schoot et al., 2021; van de Schoot, Winter, Ryan, Zondervan-Zwijnenburg, & Depaoli, 2017).

At the core of Bayesian statistics is the model, that is used to update the posterior with the combination of prior information and the data. This model must be at least

partly generative, which means, that it has to contain (conceptual) information about the generative process underlying the data. The creation of this model is an essential part of the Bayesian workflow and requires the researcher to not only have an understanding of the data structure but also to have a theory about how this data came about. Once this model has been created its predictive ability can be compared with other models. As it has to be a generative model, it also allows the Bayesian researcher to simulate data, making predictions about unobserved data points. Guest & Martin (2021) argue about computational modelling that "constraining our inference processes through models enables us to build explanatory and predictive theories." Specifying models is a crucial step in the direction of open science, improves transparency and helps facilitate the discourse between researchers.

Lee (2011) categorizes three different ways that cognitive scientists can make use of this framework. One way is the conduction of data analyses, as part of a "statistician's" toolbox, a second way is to apply concepts of Bayesian inference directly to cognitive processes as a model for how the mind works and how it makes inference. The third way to use Bayesian statistics as a cognitive scientist is "to use them to relate models of psychological processes to data" (Lee, 2011). Here, the goal of the inference is not on the measured variable directly, but on some latent variable of cognitive models, like the *learning rate*, as we will see in the second example of this tutorial. One of the great strengths of the Bayesian framework is the ability to manage even complex hierarchical models and fit distributions with multiple latent parameters for a group of subjects. Hierarchical models describe statistical models, where information is distributed on several layers and sub-layers, like a group-level and a subject-level. This allows for the integration of uncertainty on more than one level of the model. Hierarchical modelling comes with many advantages, among them a greatly reduced risk of overfitting (Gelman, 2006; Lee, 2011; Nicenboim & Vasishth, 2016; Papaspiliopoulos, Roberts, & Sköld, 2007). Not only do hierarchical Bayesian methods allow the model development to happen at multiple levels of theoretical abstraction, but they also permit the same psychological variables to explain behaviour over groups of related tasks. Especially for cognitive scientists, Bayesian statistics provide a flexible and powerful framework for implementing hierarchical models that represent cognitive information processing mechanisms. Within the Bayesian framework, it is also possible to understand data from a single task as coming from a mixture of qualitatively and quantitatively different sources (Lee, 2011). These are just some of

the benefits that come with using the Bayesian framework. The rest of this tutorial will go into more detail about the Bayesian workflow with two example models, and hopefully, provide the reader with a place to start when trying it out on their own data sets.

1.4 Required software

This tutorial and the associated materials use the following open-source research software: *R* (R Core Team, 2021), *Stan* (Stan Development Team, 2020), `rstan` (Stan Development Team, 2020), `rstudioapi` (Ushey, Allaire, Wickham, & Ritchie, 2020), `extraDistr` (Wolodzko, 2020), `bayesplot` (Gabry & Mahr, 2021), `bayestestR` (Makowski, Ben-Shachar, & Ludecke, 2019), `ggpubr` (Kassambara, 2020), and `ggplot2` (Wickham, 2016). The process makes heavy use of random-number generation. To reproduce the exact results as demonstrated in this paper, it is necessary to set the random-number seed (lst. 1.1) at the top of the script and ensure that you are using *R* version 4.0 or higher:

Listing 1.1 Setting the seed at the beginning of the script and load necessary packages

```
# setting the seed makes sure that the script returns the same results each time
set.seed(1450154627)

# Load necessary packages
library(rstan)
library(ggplot2)
library(ggpubr)
library(bayesplot)
library(rstudioapi)
library(extraDistr)
library(bayestestR)
```

Changing the seed or using an earlier version of *R* should still produce valid results, although they might differ from the results demonstrated in this article.

1.5 Disclosure

The code to reproduce the analyses reported in this article is included in the attachments. Additionally, it is publicly available on *Github* and can be accessed at `https://github.com/jweickm/ppc_tutorial`. The author declares that he has no conflicts of interest.

2. Methods

This section provides an overview of the methods used in this tutorial. I aim to illustrate a Bayesian workflow in the form of an easy-to-follow tutorial. For demonstrative purposes, I employ a linear regression model. For software, this tutorial makes use of the Stan, made accessible through `rstan`, an interface for the popular statistical coding language *R*.

First, I will introduce the basic workflow of Bayesian inference. Second, I will show the different code blocks of both the *R* and the *Stan* scripts, what their respective function is and how they work together. Third, I will demonstrate the workflow including prior predictive and posterior predictive checks with two examples, a linear regression model of actual height and weight data from the *!Khung San* people, and a Rescorla-Wagner reinforcement learning model with simulated data from a two-armed bandit task. Using these examples I will demonstrate the principles for a good workflow and how to implement them.

2.1 Bayesian workflow overview

Bayesian statistics represents a framework for analysis and parameter estimation which is based on Bayes' theorem (eq. 1.2).

In the standard research cycle, the researcher searches for background knowledge, which is then used to define a problem, from which a research question and even more specific hypotheses are generated. From there, the analytic strategy needs to be specified, an appropriate model has to be selected and the data collection strategy designed (van de Schoot et al., 2021). Notably, there is refining going back-and-forth between the individual steps, hence the term "research *cycle*." The importance of this standard research cycle becomes especially evident in Bayesian statistics, where previous knowledge is explicitly included via the prior distribution. To this end, the researcher must make sure to have a firm understanding not only of the desired analytic strategy and the model type but also of the required background knowledge and of the confidence of that knowledge. This is referred to as domain expertise (Schad, Betancourt, & Vasishth, 2020; van de Schoot et al., 2021), which is relied on by the researcher to decide, which background knowledge is elicited as a prior and how those priors are expressed as probabilistic statements in the model. While it is not necessary to include the entirety of the prevalent domain expertise

into the model, the model should not be in outright conflict with that knowledge.

A typical Bayesian modelling workflow contains three main steps. First, the available knowledge about a given model parameter is included via the prior distribution. Secondly, the *likelihood function*, also called the likelihood is determined using the information about the parameters available in the observed data. Finally, both the prior distribution and the likelihood are combined using Bayes' theorem (eq. 1.2), which results in the posterior distribution. The posterior distribution is a probability distribution that reflects the updated knowledge about the model parameters, balancing prior knowledge and observed data. It can then be used to conduct inferences (Gelman et al., 2020; Lambert, 2018). According to van de Schoot et al. (2021) the unique property of Bayesian statistics can be summarized as follows: "All observed and unobserved parameters in a statistical model are given a joint probability distribution, termed the prior and data distributions."

We need to keep in mind that in Bayesian inference, we wish to determine a posterior belief in each set of parameter values. This means that instead of fixing the parameter values and using our model to generate data, we keep the data constant and vary the parameter values. Accordingly, the posterior distribution no longer sums - for discrete distributions - or integrates - for continuous distributions - to 1. This causes our probability model to operate no longer as a valid probability distribution. To recognize this distinction in Bayesian statistics, the term *likelihood* is generally used instead of *probability distribution* (Lambert, 2018).

When we want to begin the sampling process, we first need to elicit the parameters prior distributions, from which we draw the initial parameters. In choosing appropriate prior distributions, we do not only choose the type of prior for each parameter but also the degree of informativeness of the prior.

Full Bayesian analysis requires a generative model. Unlike non-generative models that can be considered as simple data summaries, and "classical" statistical models that are characterized only by probability distributions for the data y given a parameter θ, but without a probability distribution for θ, generative models are models which result in a joint probability distribution for all data and parameters (Gelman et al., 2020; van de Schoot et al., 2021). A generative statistical model contains all the information necessary for the generation of observations. It can therefore be used for predictive simulation when the parameter estimates are known. Other advantages of generative models are, that it is described formally, that is mathematically in a precise fashion. However, Bayesian inference only requires the likelihood, which could also be the same for different generative models (Gelman et al., 2020). It is also common to use models that are not fully generative, as we will see in our example using linear regression. There we will only model an outcome y given the predictor x without a generative model for x itself. These types of models are still considered generative, although they include additional unmodeled data x.

But how does one come up with an appropriate generative model? Lambert (2018)

argues there are three main steps in creating a Bayesian model:

"First, we recognize a set of variables that we wish to understand. Some of these variables are observable. We call these data. Others are unobservable things like rates and averages. We call these parameters. [Second,] for each variable, we define it either in terms of the other variables or in terms of a probability distribution. These definitions make it possible to learn about associations between variables. [Third,] the combination of variables and their probability distributions defines a joint generative model that can be used both to simulate hypothetical observations as well as analyze real ones" (Lambert, 2018).

The production of this generative statistical model is inherently iterative, which is why Gelman et al. (2020) argue that the Bayesian workflow as a whole should be approached as an iterative process (Gelman et al., 2020). As there is an infinite number of possible distributions, and ideally, we would like to consider each possible distribution and its posterior plausibility, this is a task that cannot be accomplished completely. The geometry of the posterior space is usually too complex to be analytically derived, especially for continuous variables. Instead, we draw samples to approximate its shape. The sampling of the distribution is done using the Markov chain Monte-Carlo (MCMC) algorithm. This algorithm draws samples from different states of the distribution, which can then be used as an estimation of the shape of the distribution after a given number of jumps (van de Schoot et al., 2021)]. The recent advances in computing technology have allowed Bayesians to compute even large hierarchical models, which would have been unfeasible in the times of Bayes, Price and Laplace.

One of the most powerful sampling algorithms available to us today is the Hamiltonian Monte-Carlo (HMC) sampler (Betancourt, 2018). It is fast and as the number of simulations increases the theoretical error approaches zero, given that the chains have mixed (Gelman et al., 2020; Monnahan, Thorson, & Branch, 2017; Schad, Betancourt, & Vasishth, 2020). The very efficient and fast *No-U-Turn* sampler accessible via the programming language *Stan* is a dynamic implementation of HMC, which uses the so-called *No-U-Turn* termination criterion (see Betancourt, 2018; Carpenter et al., 2017). One other advantage of using *Stan* is, that it interfaces with many popular statistical coding languages, among which there is *R* that we will be using in this tutorial. This makes *Stan* ideal for psychologists and researchers in the cognitive sciences who are already familiar with *R*. While other algorithms are also available, they are not the scope of this thesis (for a comparison see van de Schoot et al., 2021).

Once the posterior distribution has been estimated with the MCMC algorithm, we can use it for model evaluation, model comparison or to conduct inference. One way to evaluate the model is by conducting prior and posterior predictive checks. These checks make use of the data-generating capabilities of the model to simulate data points based on draws from the prior or the posterior distributions. For inference, we can answer

our research question, about how likely a certain parameter is given our data, directly from the posterior, since it is a valid probability density distribution. Instead of simply providing a maximum likelihood parameter estimation, we have information about the entire posterior distribution, although it is common to report a 95 interval.

2.2 Choosing appropriate prior distributions

Aside from requiring the researcher to explicitly formulate a generative model, one of the biggest advantages of Bayesian statistics over frequentist statistics is the inclusion of prior knowledge into the model. This is especially useful when the sample size is small and when a lot of prior knowledge exists that can be expressed as probabilistic statements. For this, let us take another look at the Bayes' theorem eq. 2.1.

$$\overbrace{p(\theta|data)}^{\text{posterior}} = \frac{\overbrace{p(data|\theta)}^{\text{likelihood}} \quad \overbrace{p(\theta)}^{\text{prior}}}{\underbrace{\sum_{\theta^*} p(data|\theta^*)p(\theta^*)}_{\text{evidence}}}$$
$$p(\theta|data) \propto p(data|\theta) \quad p(\theta) \tag{2.1}$$

$\propto$... "proportional to"

The prior $p(\theta)$ is located in the numerator on the right-hand side and denotes the probability of the observed effect occurring without any external influences, i.e. the effect's *base probability*. This can be understood as akin to the baseline probability of the effect. The posterior distribution is updated based on the prior and the evidence in combination with the likelihood and becomes then the prior distribution for the next iteration.

Priors take shape in various distributions, such as normal, uniform, Poisson or Cauchy distributions, among others (van de Schoot et al., 2021). Depending on their definition, priors have different levels of *informativeness*. The prior's degree of *informativeness* depends on the parameters that are chosen to generate the prior distribution. This *informativeness* can be anywhere on a continuum from absolute uncertainty, i.e. uninformative, to relative certainty, i.e. highly informative. In practice, mainly three classifications of the degree of *informativeness* are used: *informative*, *weakly informative*, and *diffuse/uninformative*. Which of these verbal classifications is used depends on the researcher's expertise in the field and their idiosyncratic judgement. For example, while a numeric variance of $\sigma = 1,000$ may be considered informative in one research area, it might be rather uninformative in another, depending on the parameter scaling (McElreath, 2019; Sarma & Kay, 2020; van de Schoot et al., 2021; Zundert, Somer, & Miocevic, 2020).

2.3 The influence of the prior on the posterior

To illustrate the influence that different priors and different sample sizes have on the posterior, let us consider the following example. Let us assume that we have a coin for which we want to make an educated guess about its probability to show *Heads* (H) when tossed. As there are only two possible outcomes - let us ignore the off-chance that the coin lands on its side - the outcome *Heads* will follow the Bernoulli distribution, with θ as the parameter for the probability to land on *Heads* (eq. 2.2).

$$H \sim Be(\theta) \tag{2.2}$$

The result of many throws of this coin, each result following the Bernoulli distribution, can be modelled with a binomial distribution, that takes n as the number of throws in addition to θ as parameters (eq. 2.3).

$$H \sim Bin(n, \theta) \tag{2.3}$$

For our given coin we do not know its *true* probability to show *Heads*. If we had no previous information about coins we might assume a flat, i.e. uniform, prior as illustrated fig. 2.1.a. From $N = 5$ throws $H = 2$ throws show heads, which leads us to believe that the likelihood for the probability of the coin to show *Heads* (θ)is slightly lower than the likelihood for its inverse probability. Yet, due to the small sample size, the posterior is still very flat overall. After $N = 20$ throws we have observed $ H = 12$ times *Heads*. The posterior probability distribution has become narrower as we have collected more data. It has also shifted slightly to the right as the probability that the true value for $\theta(H)$ is larger than $\theta = \frac{1}{2}$. As we observe $H = 49$ times *Heads* out of $N = 80$ throws, our belief about the true parameter value manifests itself around $\theta = 0.6$, although other values for θ are still quite possible.

However, in reality, we feel that we are quite familiar with the tossing of coins. Our mental model of coins states that a coin usually has a probability of $\theta = \frac{1}{2}$ to show *Heads*. Therefore, we also approach this coin with the weak prior *assumption*, here weak prior A, that the coin is most likely to have a parameter value of $\theta = \frac{1}{2}$ (fig. 2.1.b). This weak prior in the shape of a beta distribution ($\sim Beta(5, 5)$) leads us to update the posterior in light of the data we collect more quickly and develop a narrower posterior compared to the flat prior. After $N = 80$ throws the maximum of the posterior is similar to the flat prior, albeit with a slightly lower θ and a higher density at the maximum.

Figure 2.1.c illustrates what the posterior would look like with a different weak prior, B, that assumes a lower value for θ. The difference to weak prior A is especially pronounced for smaller sample sizes, although there is still a visible difference between the two posterior distributions, even after $N = 80$ throws.

Finally, figure 2.1.d shows the update of the posterior if we had chosen the initial prior with greater confidence that the true value is $\theta = \frac{1}{2}$. This strong prior results in the most narrow posterior out of the four.

From this example, we see that the influence of the prior is greatest when the sample size is small. Herein lies also one of the great strengths of Bayesian statistics, especially when we have reliable prior information to include in our model. The combination of prior information and data allows us to make the most of the data at hand. However, choosing very informative priors may also create a bias that distorts every result towards the expected outcome. Therefore, choosing weakly informative priors is generally considered to be good practice in Bayesian statistics (Gelman et al., 2020; Kennedy, Simpson, & Gelman, 2019; Sarma & Kay, 2020; van de Schoot et al., 2021; van de Schoot, Winter, Ryan, Zondervan-Zwijnenburg, & Depaoli, 2017).

2.4 Subjectivity when choosing priors

The subjectivity of the analyst in deciding on the statistical model and the appropriate prior is often used as an argument against Bayesian statistics, as it depends on the analyst specifying their pre-experimental beliefs via prior distributions. However, advocates of Bayesian statistics argue that all kinds of analyses involve subjectivity to some extent, either stated explicitly or assumed implicitly (Lambert, 2018). For example, even in frequentist analyses, a model has to be selected for the analysis, which is based on certain (implicit) assumptions, which may, or may not, be true. Compared to this, Bayesian models are more often constructed from scratch, making the researcher more aware of the assumptions inherent in their approach. Furthermore, in applied research, data is often selected to fit the analysis. It is pre-processed by removing outliers or filtering for missing data points. This is unlike the Bayesian approach, where the observed data is considered to be the "ground truth." As opposed to significance tests there is no need for cut-off values in Bayesian statistics, as the output is reported as probability distributions. Neither Bayesian nor frequentist approaches to the pursuit of knowledge can exist without subjectivity on the researcher's side. However, there is a strong case to be made for Bayesian statistics as they explicitly acknowledge the subjective nature of the statistical process, and require the subjective assumptions to be made explicit by the analyst (Lambert, 2018; McElreath, 2019).

As the selection of the prior is the most frequently criticized aspect of Bayesian statistics, it is considered best practice to make use of so-called prior and posterior predictive checks for model evaluation, in order to make an informed decision about the prior (van de Schoot et al., 2021). Prior predictive checks provide insights into what the data might look like based solely on the prior distribution. This allows, for example, for the exclusion of unreasonable prior ranges. With posterior predictive checks, on the other hand, we

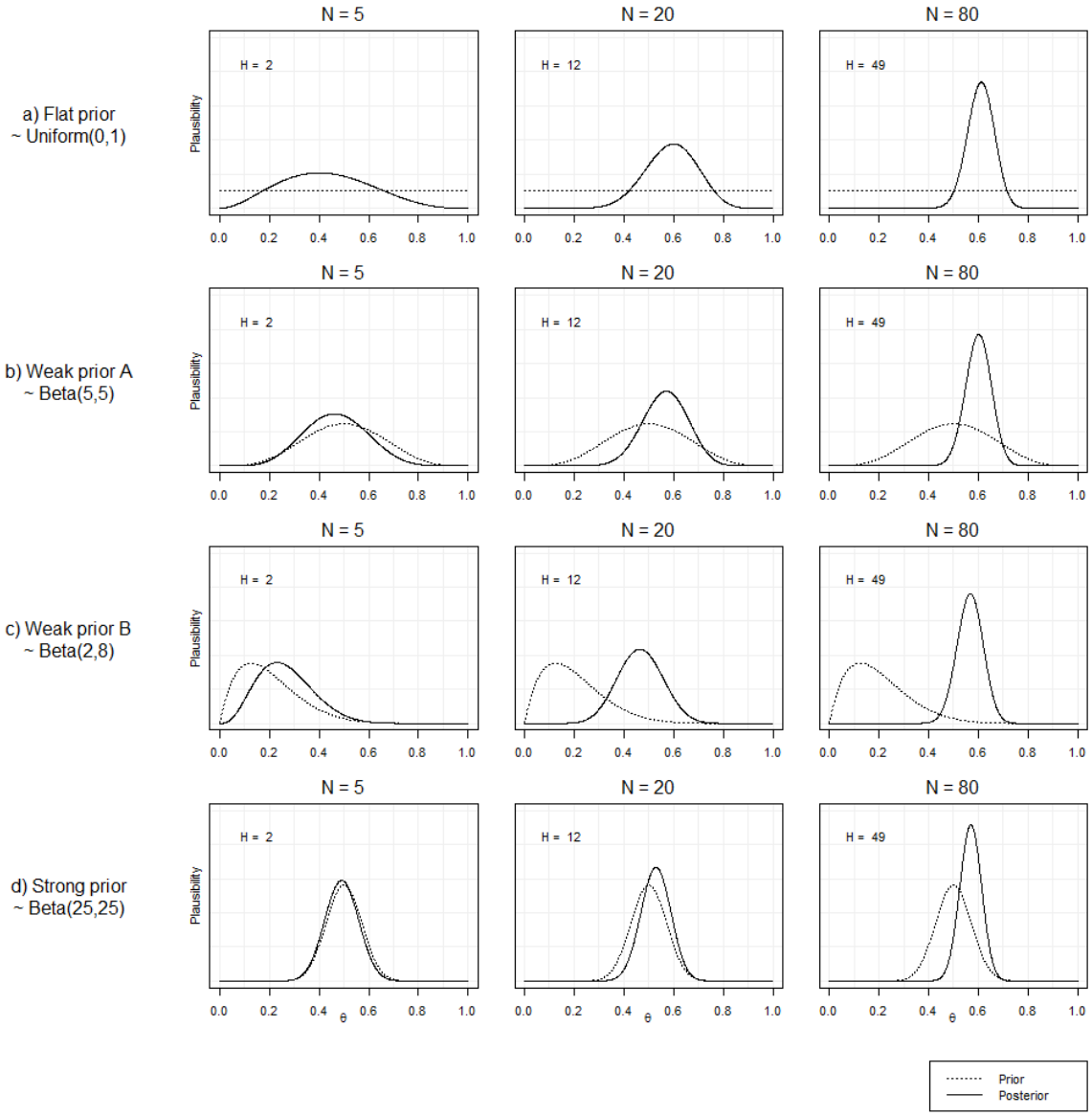

Figure 2.1: The effect of different priors and sample sizes (5, 20, 80) on the posterior. **a.** Flat prior. **b.** and **c.** Weakly informative priors. **d.** Strongly informative prior.

can illustrate the model performance by comparing the simulated model output to the observed data. However, while both prior and posterior predictive checks are essential for the Bayesian workflow, there is still a lack of comprehensive tutorials for their application in both statistical and cognitive models. Therefore, we will take a further look into the application of these checks in the following section.

3. Example 1: Fitting a linear regression model

The data set that is used for this example is taken from Howell, referenced in McElreath (2019). It is partial census data[1] about the *!Kung* San of the Dobe area, who most likely the most famous foraging population of the 20th century. This data was compiled from interviews conducted by Nancy Howell in the late 1960s (Howell, 2000). For those wondering about the pronunciation of the "!" in front of the word, it indicates an alveolar click that then releases into a voiceless uvular fricative encoded here by the "K."

3.1 Selecting an appropriate model

Bayesian statistics requires the researcher to create a generative model of the data, forcing them to critically examine their expectations about the data and the underlying generative processes. By having to write and revise their model the researcher must make the assumptions about the data and their interaction explicit. Creating the model might be likened to the drafting of a scientific paper. The first draft is just the first step to an iterative process that is improved upon by the subsequent editing and revision steps. Undergoing this process can be understood as a learning process, during which the researcher gains a deeper statistical understanding of the topic (McElreath, 2019).

For this example, we are interested in modelling the relationship between the weight and height of adult members of the *!Khung* San population. To select an appropriate generative model we first need to define the problem and specify the respective research question and hypothesis. In our case, we are curious about the relationship between weight and height of the *!Khung* San population, and how well we can model that relationship with a linear regression model. We, therefore, need estimates for the linear regression model parameters so that we can describe that relationship in mathematical terms. Our hypothesis is, that a linear regression model can describe the relationship between weight and height with sufficient precision.

The proposed linear regression model can now be expressed as follows in mathematical terms (eq. 3.1).

$$\text{height}_i = \alpha + \beta * \text{weight}_i + \epsilon_i \tag{3.1}$$

[1]The census data from Nancy Howell is available here: https://tspace.library.utoronto.ca/handle/1807/10395

As in other similar linear regression models, α stands for the *intercept* of the regression line with the ordinate, the point where $\text{weight}_i = 0$. α can be understood as a sort of 'initial value' for *height*. β represents, how many units *height* increases with each *weight* increase of 1. This is called the *slope* of the regression line. Finally, ϵ is the data point specific error term, which captures all other factors influencing the dependent variable *height* beside the regressor *weight*.

Now, one might wonder why we would choose a linear regression model for this specific interaction. It may be true that a person's height is zero when their weight is zero, their relationship does not necessarily have to be linear. However, even though it might be tempting to make the model as complex as possible to account for all the details in the data, there are also some profound advantages to a simpler statistical model. First, we reduce the risk of *overfitting* the model to our data. As our data only represents a small sample of the whole truth, fitting a model too close to the data strongly reduces its generalizability. This is a common issue in machine learning applications. Therefore, we want to avoid fitting the model too close to our data. Such a model might describe our data perfectly, but it would not be very useful when it comes to new data or when making predictions from data points that we have not observed. Instead, we usually want to create a model in such a way that it describes the relationship between the variables of interest in a more general fashion.

As famous statistician George Box eloquently put, "since all models are wrong the scientist must be alert to what is importantly wrong. It is inappropriate to be concerned about mice when there are tigers abroad" (Box, 1976). A model, especially a mathematical model, always represents a strong simplification of the actual phenomenon, just like a map of an area is not the actual area itself. They represent a selection of factors, that allow us to construct a model for our problem at hand. In order to be considered useful, this model must then enable us to infer the desired information from the model or make predictions based on information that we already have.

Another point to consider is, that we will only include the data from the adult population in our model. This excludes very small values that are closer to zero, thereby heavily limiting the expected data range. Considering this, a simple linear regression model might provide the best fit while still keeping the model simple to minimize the risk of *overfitting*. Therefore, we will stick with the linear regression model for this example.

When it comes to the height and weight of humans, we are lucky to be able to draw on our rich everyday experiences. As we have seen the height and weight of many different people, we have a pretty good understanding of the range of expected values for both variables. This knowledge comes in handy when eliciting the prior distributions for our model.

3.2 Prior distributions for the weight ~ height linear regression model

Coming back to our example of a linear regression model, it is generally a good idea to first take a look at the raw data before we begin with any data analysis step.

Table 3.1: Summary statistics of the *!Khung* data set.

	height (cm)	weight (kg)	age (y)	f (0); m (1)
mean	154.6	45.0	41	0.47
sd	7.7	6.5	16	0.50
median	154.3	44.8	39	0.50
min	136.5	31.1	18	0
max	179.1	63.0	88	1
n	352	352	352	352

This data set has already been filtered to include only adults, that is, persons with an age greater than 17 years. As we can see in tbl. 3.1, the mean *age* is 41 years ($\sigma = 16$). In total the sample includes $n = 532$ persons, of which 53 percent are female. Importantly, the mean *height* is 154.6 cm ($\sigma_{height} = 7.7$) and the mean *weight* is 45 kg ($\sigma_{weight} = 6.5$).

As we can see from scatterplot fig. 3.1.a, a linear regression model ought to fit our data. To recall, our linear regression model contains three model parameters: α, β and ϵ. In a linear regression model, α denotes the intercept, i.e. the value where the regression line intersects with the ordinate axis. For our model, this means that α denotes the expected height of a person with zero weight. However, this is quite unintuitive as we would normally expect a person with a weight of zero to also have a height of zero. As this will not be the case for our linear regression, the intercept value becomes arbitrary and more difficult to elicit a good prior for. If instead, α denoted the mean height, it would be more intuitive to find a good prior distribution for α as it would correspond more closely to the distribution of height in our data (Papaspiliopoulos, Roberts, & Sköld, 2007). Therefore, I propose a *mean-adjusted* linear regression model (eq. 3.2).

$$\text{height}_i = \alpha + \beta * (\text{weight}_i - \overline{\text{weight}}) + \epsilon_i \tag{3.2}$$

When we choose a prior for α we do not simply decide on a single arbitrary value. Rather, we elicit a prior distribution that we define by choosing sensible distribution parameters. One of the great advantages of adjusting the linear regression model in the above fashion is that the mean of the prior distribution for α now corresponds to the

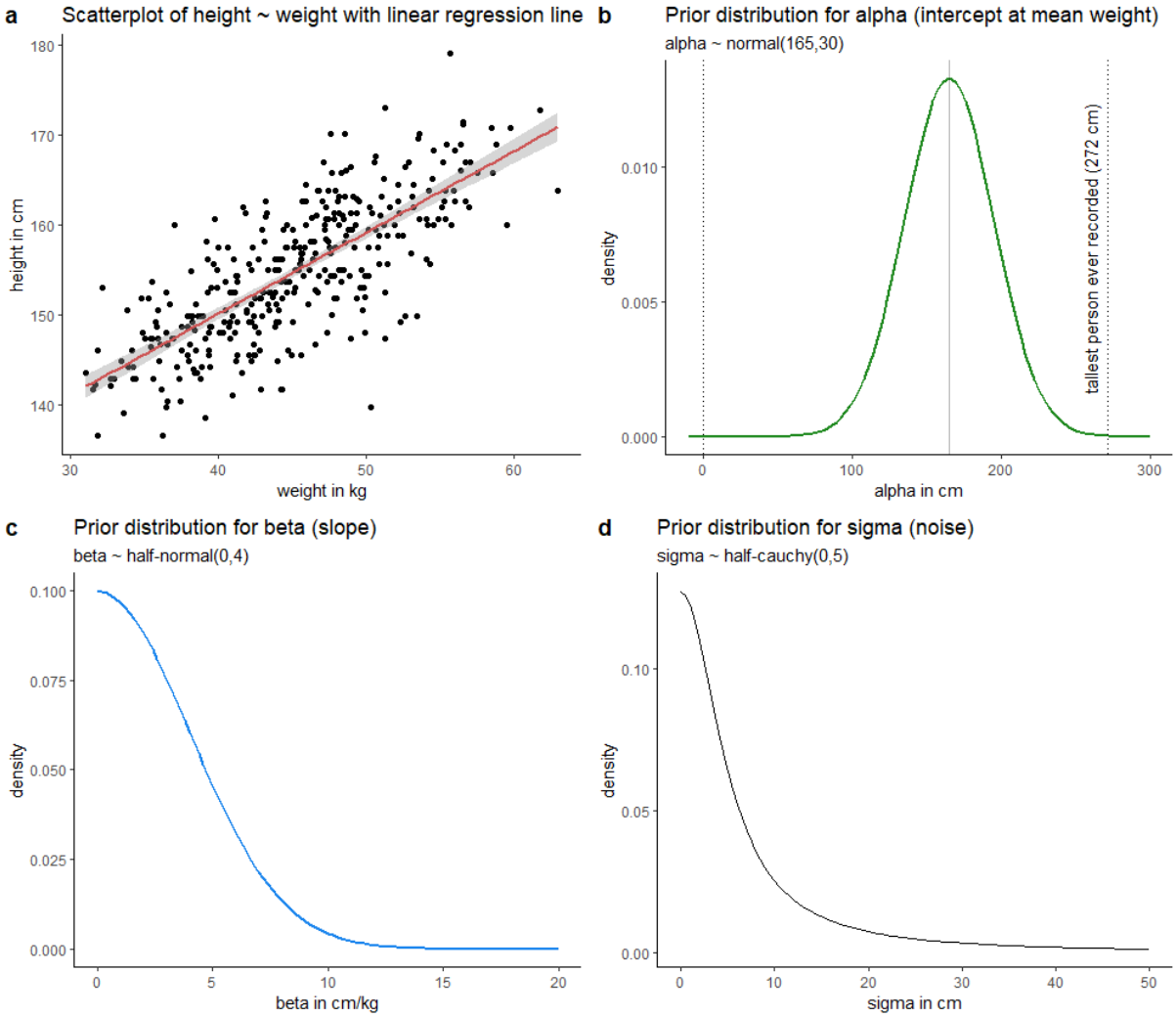

Figure 3.1: **a.** Scatterplot of raw data with linear regression line. **b.** Prior distribution of alpha. **c.** Prior distribution of beta. **d.** Prior distribution of epsilon.

value for height where the weight is equal to the mean weight. This corresponds to the centroid of the data point cloud.

Based on the extensive data that already exists about human height, we choose an informative prior with a mean of 165 cm for α and a slightly more robust standard deviation of 30 cm (fig. 3.1.b) as we are including data for both adult males and females (Roser, Appel, & Ritchie, 2013). Furthermore, with a sample size of $n = 352$, we do not need to worry that our prior might be too flat.

For the slope β we can assume, that it should be a positive value as height is expected to increase with weight, hence the linear regression model. Therefore, a half-normal prior distribution can be used, which is bounded at zero and thus only produces positive values for β (Gelman et al., 2020; Lambert, 2018; Zundert, Somer, & Miocevic, 2020). Like the normal distribution, the half-normal function takes two parameters, the mean and the standard deviation. A half-normal distribution with a mean of 0 and a standard deviation of 4 yields a distribution that has the highest density on 0 but still realistically

allows for the parameter to reach values of around 20, thereby incorporating the prior information that we have about β (see fig. 3.1.c).

Lastly, we also need a prior distribution for the standard deviation of the noise term ϵ, i.e. σ. It is common to use a very flat distribution for σ. McElreath (2019) recommends a uniform prior, however, it is usually not recommended to use uniform priors as they do not necessarily constitute valid probability distributions. This is the case when they are not bounded on both ends, as they do not integrate to 1. Furthermore, due to their extreme cutoff at the end of their range, they are prone to yielding inconsistent results, making their use not generally advisable (Lambert, 2018; Stone, 1976). Usually, a better option is the Cauchy distribution, which is similar to the normal distribution, however, with heavier tails. This makes the distribution overall more "flat" and allows for more frequent sampling of extreme values compared to the normal distribution. Yet, this can result in an unusually magnitude of the posterior samples for Cauchy priors, as well as heavy-tailed posteriors (Gelman, 2006; Ghosh, Li, & Mitra, 2017). As standard deviations must be positive, a half-Cauchy distribution that contains only positive values is ideal for our application. For this tutorial, we choose a half-Cauchy distribution with a standard deviation of 10 and a lower bound of 0 (fig. 3.1.d). To define the half-Cauchy distribution in *Stan* we have to choose a Cauchy distribution and set the lower bound to 0 in the `parameter` block.

Table 3.2 summarizes the chosen prior distributions for the linear regression. The left-hand side is the mathematical notation, while the right-hand side shows how the given distribution is expressed in Stan code.

Table 3.2: Prior distribution in both mathematical notation and as Stan code.

Mathematical notation	Stan code
$\alpha \sim \text{Normal}(165, 30)$	`real alpha` `alpha ~ normal(165, 30)`
$\beta \sim \text{Half-Normal}(0, 1)$	`real<lower=0> beta` `beta ~ normal(0, 1)`
$\sigma \sim \text{Half-Cauchy}(0, 10)$	`real<lower=0> sigma` `sigma ~ cauchy(0, 10)`

3.3 Writing the model in Stan

Once we have created a generative model in mathematical terms, we can translate it into a *Stan* model. The *Stan* model can then be called directly from within *R*. The basic structure of a *Stan* model is shown in lst. 3.1.

Listing 3.1 Basic structure of a Stan model

```
data { }

parameters { }

model { }

generated quantities { }
```

"The `data` block is for the declaration of variables that are read in as data. With the current model executable, each Markov chain of draws will be executed in a different process, and each such process will read the data exactly once. Data variables are not transformed in any way. […] The variables declared in the `parameter` program block correspond directly to the variables being sampled by Stan's samplers (HMC and NUTS). From a user's perspective, the parameters in the program block are the parameters being sampled by Stan. […] The `model` program block consists of optional variable declarations followed by statements. The variables in the model block are local and not written as part of the output. […] The `generated quantities` program block is rather different [from] the other blocks. Nothing in the `generated quantities` block affects the sampled parameter values. The block is executed only after a sample has been generated. Within the `generated quantities` block, the values of all other variables declared in earlier program blocks (other than local variables) are available for use in the `generated quantities` block. All variables declared as generated quantities are printed as part of the output" (Stan Development Team, 2021).

3.3.1 The `data` block

Listing 3.2 shows the data block of our *Stan* model. Here we declare the variables that we load via the call of `rstan` into our model. In *Stan* comments are lines that begin with `//`, and every line must end with `;`. The variables `N`, `weight`, `mean_weight`, and `height` should be straightforward. We declare `N` as an integer as it is a natural number. `mean_weight` is a continuous value, so it is declared as a variable of the `real` type. The variables `weight` and `height` have one data point for each subject, so we declare them as vectors. `analysis_step` is a logical switch that allows us to use the same script for both the prior predictive checks as well as the posterior predictive checks. Finally, we declare the hyperparameters of the three prior distributions in the next step. It should be noted here that it is necessary to set lower boundaries for `prior_beta` and `prior_sigma` as their distributions only contain positive values (Akaike, 1980; Gelman, 2006).

Listing 3.2 Declaring the imported data in the data block

```
data {
  int<lower=0> N;
  vector<lower=0>[N] weight;
  real<lower=0> mean_weight;
  vector<lower=0>[N] height;
  int<lower=0, upper=1> analysis_step;

  real prior_alpha[2];
  real<lower=0> prior_beta[2];
  real<lower=0, upper=20> prior_sigma[2];
}
```

3.3.2 The `parameter` block

After we have declared the data and the prior distributions we declare the model parameters that we want to sample with *Stan*'s sampler in the `parameter` block (lst. 3.3). Again `alpha`, `beta`, and `sigma` are declared as `real` variables, that is continuous values with `sigma` being bounded by both a lower and an upper bound.

Listing 3.3 Declaring the model parameters in the parameter block

```
parameters {
  real alpha; // declaring the intercept
  real<lower=0> beta; // declaring the slope (must be positive)
  real<lower=0> sigma; // declaring the standard deviation of the error term (cannot
   ↪  be negative)
}
```

3.3.3 The `model` block

In the `model` block we define the distribution types of the priors and declare the likelihood function (lst. 3.4). `alpha ~ normal(prior_alpha[1], prior_alpha[2]);` means that our previously declared parameter `alpha` is normally distributed with the in the `data` block imported hyperparameters `prior_alpha`. The draws for `beta` and `sigma` follow the same logic.

The conditional statement `if (analysis_step == 1) {...}` makes use of the boolean variable that we also imported in the beginning. When we call the *Stan* script from *R* through `rstan` we can set the `analysis_step` to `1` to include the likelihood function when we also want to update the posterior distribution. The likelihood function links the data, `weight` and `height` with our prior. It answers the question of how likely the data is for a given parameter. Here, the likelihood for `height` is normally distributed

with `alpha + beta * (weight - mean_weight)` as the mean and our noise parameter `sigma` as the standard deviation.

Listing 3.4 Declaring the model and likelihood in the model block

```
model {
  // priors
  alpha ~ normal(prior_alpha[1], prior_alpha[2]);
  beta  ~ normal(prior_beta[1], prior_beta[2]);
  sigma ~ cauchy(prior_sigma[1], prior_sigma[2]);

  if (analysis_step == 1) {
    height ~ normal(alpha + beta * (weight - mean_weight), sigma); // likelihood function
```

3.3.4 The `generated quantities` block

The final block of the example *Stan* model script is the `generated quantities` block (lst. 3.5). The `generated quantities` block is executed once for each iteration of the Markov chain Monte Carlo and generates the output distributions for our model. This block updates the model parameters from the model block for each iteration. The final iteration then yields the model parameters from the point where the chains (ideally) converged. Using the posterior distributions for our model parameters `alpha`, `beta` and `sigma` we simulate the dependent modelled height variable `height_bar`. This step is important for the **prior predictive check** as well as the **posterior predictive check**.

Listing 3.5 Defining the output in the generated quantities block

```
generated quantities {
  vector[N] height_bar; // the modelled height
   // here we loop over all N to draw a value for the simulated height from our distribution
   for (n in 1:N) {
      height_bar[n] = normal_rng(alpha + beta * (weight[n] - mean_weight), sigma); // calculates posterior distribution
  }
```

3.4 Harnessing the power of Hamilton Monte-Carlo with `rstan`

`rstan` is an *R*-package that allows us to harness the power of *Stan* directly through *R*. *Stan* uses one of the most efficient and powerful HMC samplers (Kelter, 2020; Monnahan, Thorson, & Branch, 2017) for the estimation of the posterior probability distribution.

One of the advantages of using *Stan* is its ability to be called from within an existing *R* environment via the package `rstan` (Stan Development Team, 2020). Following, I will briefly show how to call our *Stan* script from *R*. After loading the required packages (lst. 1.1) we tell *Stan* to automatically save a serialized version of the compiled model to the source directory and to detect the number of cores available and run that many chains in parallel for maximum performance (lst. 3.6).

Listing 3.6 Initial Stan options

```
rstan_options(auto_write = TRUE)
options(mc.cores = parallel::detectCores())
```

Following, we read in the *!Khung* data, filter it by age to include only the adults, and calculate the mean weight based on that group (lst. 3.7). The output of the `custom.summary` function can be seen in table 3.1. We also save the mean weight in the `mean_weight` variable so that it is available for the *Stan* script.

Listing 3.7 Load the data and do initial filtering

```
d <- read.csv(
    "https://raw.githubusercontent.com/rmcelreath/rethinking/master/data/Howell1.csv",
    sep = ";")
df <- d[ d$age >=  18, ] # filter only for the adults

custom.summary <- function(x, ...){
  c(mean   = mean(x, ...),
    sd     = sd(x, ...),
    median = median(x, ...),
    min    = min(x, ...),
    max    = max(x, ...),
    n      = as.integer(length(x, ...)))
}

df_summary <- apply(df, 2, custom.summary)
mean_weight <- df_summary["mean", "weight"]
```

In the `data` block of the, *Stan* script (lst. 3.2) we did not actually define the hyperparameters of the prior distributions yet. This step happens in the *R* script, although it could also be done within the *Stan* script. How to define the hyperparameters as well as the boolean variables for switching between prior and posterior predictive checks in *R* is shown in lst. 3.8. Apart from the logical switch, the definition of the prior hyperparameters can also be accomplished directly in the `model` block of the *Stan* script as shown in lst. 3.9.

Listing 3.8 Defining the hyperparameters in R

```r
# Define the states for the `analysis_step` variable in Stan
priorCheck <- 0
posteriorCheck <- 1

prior_alpha <- c(165, 30) # sets the mean and standard deviation of the mean-centred
 ↪  normal prior distribution for the intercept (alpha in the linear model), in cm
prior_beta  <- c(0, 4) # sets the mean and standard deviation of the normal prior
 ↪  distribution for the slope (beta), in cm/kg
prior_sigma <- c(0, 5) # sets the location and shape parameter of the Cauchy prior
 ↪  distribution from which the standard deviation for the noise term is drawn (sigma)
```

Listing 3.9 Defining the hyperparameters in Stan

```
model {
  alpha ~ normal(165, 30);
  beta ~ normal(0, 4);
  sigma ~ cauchy(0, 5);
}
```

After these model-specific parameters have been set, we need to provide a few more details about the sampling process before we can run our *Stan* model (see lst. 3.10). The model call includes information about the model name (`model_name`), the data (`data`), the number of chains (`chains`) that are run in total, the number of iterations that each chain runs for (`iter`), the number of warmup iterations (`warmup`) that are allocated for the learning of the Hamiltonian Monte Carlo parameters, the period of saving samples (`thin`), the specifications of the initial values for all or some of the Hamiltonian Monte Carlo parameters (`init`), and the seed (`seed`). The `dataList` contains all the variables that we want to have available for our *Stan* model, including the hyperparameters of the prior distributions and the actual data, of course. In our case, it also contains the value of the switch variable so that we can skip the likelihood update when we are conducting prior predicting checks.

3.5 Prior predictive checks

Conducting prior predictive checks before fitting the model allows us to make sure that the model makes sense with basic expectations. This means, that our model does not produce "impossible" values and does not produce distributions that go against the theory, on which we based the model. For example, if we wanted to include the prior information that the dependent variable can only have positive values then we should comfirm that our model mostly if not exclusivley produces positive values, depending on the certainty we have regarding that restriction of the parameter space. In these

Listing 3.10 Initializing and running Stan

```
modelFile <- "regression_height_combined.stan"
nIter     <- 2000
nChains   <- 4
nWarmup   <- floor(nIter/2)
nThin     <- 1

N <- length(df$height)

dataList <- list(N = N, weight = df$weight, mean_weight = mean_weight, height =
 ↪  df$height, prior_alpha = prior_alpha, prior_beta = prior_beta, prior_sigma =
 ↪  prior_sigma, analysis_step = -1)
Seed = 1450154627

# Let Stan know whether we want to update the posterior with the likelihood (required
 ↪  for a posterior predictive check)
dataList$analysis_step <- priorCheck

cat("Estimating", modelFile, "model... \n")
startTime = Sys.time(); print(startTime)
cat("Calling", nChains, "simulations in Stan... \n")

fit_reg_pripc <- stan(modelFile,
                  model_name = "Linear Regression",
                  data    = dataList,
                  chains  = nChains,
                  iter    = nIter,
                  warmup  = nWarmup,
                  thin    = nThin,
                  init    = "random",
                  seed    = Seed)

cat("Finishing", modelFile, "model simulation ... \n")
endTime = Sys.time(); print(endTime)
cat("It took", as.character.Date(endTime - startTime), "\n")
```

checks, we examine the predictions about our dependent data based on the parameter distributions from the prior. Conducting prior predictive checks works in three steps. First, we draw the parameter values from the priors. Second, we simulate multiple draws of the dependent variable based on the model. Third, we summarize the response visually or by reporting the mean (Gabry, Simpson, Vehtari, Betancourt, & Gelman, 2019; Levy, 2011; Zundert, Somer, & Miocevic, 2020). The prior predictive check is useful to identify unrealistic prior distributions. If a lot of "impossible" values for the dependent variable are generated, this could signify that the chosen prior specifications may be unrealistic (Zundert, Somer, & Miocevic, 2020). In our example linear regression model, this would be the case if we found generated height values below 0 cm or significantly higher than the tallest person ever recorded (207 cm).

In the `generated quantities` block of the *Stan* script, we simulated modelled height data (`height_bar`). We can use this simulated data now for a prior predictive check.

With the `ppc_dens_overlay` function of the *R*-package `bayesplot`, we can easily plot the density kernels of draws from our predicted data in comparison to the actual data. For model parameter draws from the prior distribution, this is illustrated in figure 3.2. In this plot, the dark-blue line is the distribution of the observed outcomes y and each of the 50 lighter lines (y_{rep}) is the kernel density estimate of one of the replications of y from the posterior predictive distribution. With regards to our linear regression model, this means that y denotes the actual height distribution from the data, and y_{pred} stands for the modelled height based on a model with parameters drawn from the prior distribution without update via the data. The overall shape of the posterior kernel density appears similar to the actual data, with no data points at "impossible" values. This shows us that the chosen prior distributions make sense for our model. Depending on the random draws, we might still get more extreme values in this plot as the chosen prior distribution for *sigma*, the Cauchy distribution, has very wide tails and therefore usually allows for some extreme values in the posterior.

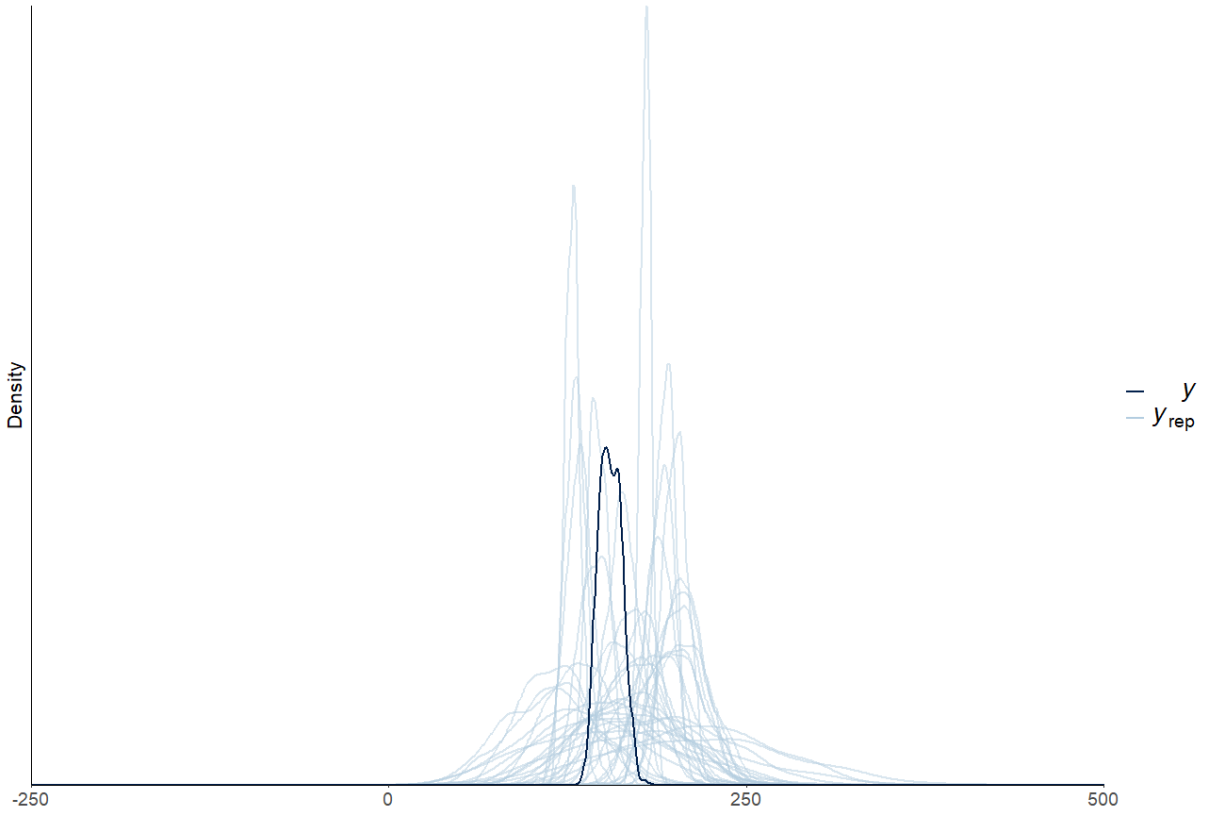

Figure 3.2: Prior kernel density plot

3.6 Posterior predictive checks

The posterior predictive check is in essence very similar to the prior predictive check, however, instead of the prior distribution, we use the posterior distribution of the model

parameters. With these model parameters, we simulate new datasets and check whether their distributions match the distribution of the original data, in order to validate the model and confirm that it is adequate for the phenomenon that we wish to model. To generate the dataset necessary for the posterior predictive check using our script, we need to set the switch variable `analysis_step` to `posteriorCheck`, which we previously defined to be `1`. This causes the inclusion of the posterior update via the likelihood (see 3.4).

The posterior distributions for the model parameters *alpha*, *beta*, and *sigma* after the chains have converged are shown in figures 3.3.a-c. Figure 3.3.d shows the resulting plot for the kernel density of the predicted data from the posterior distribution.

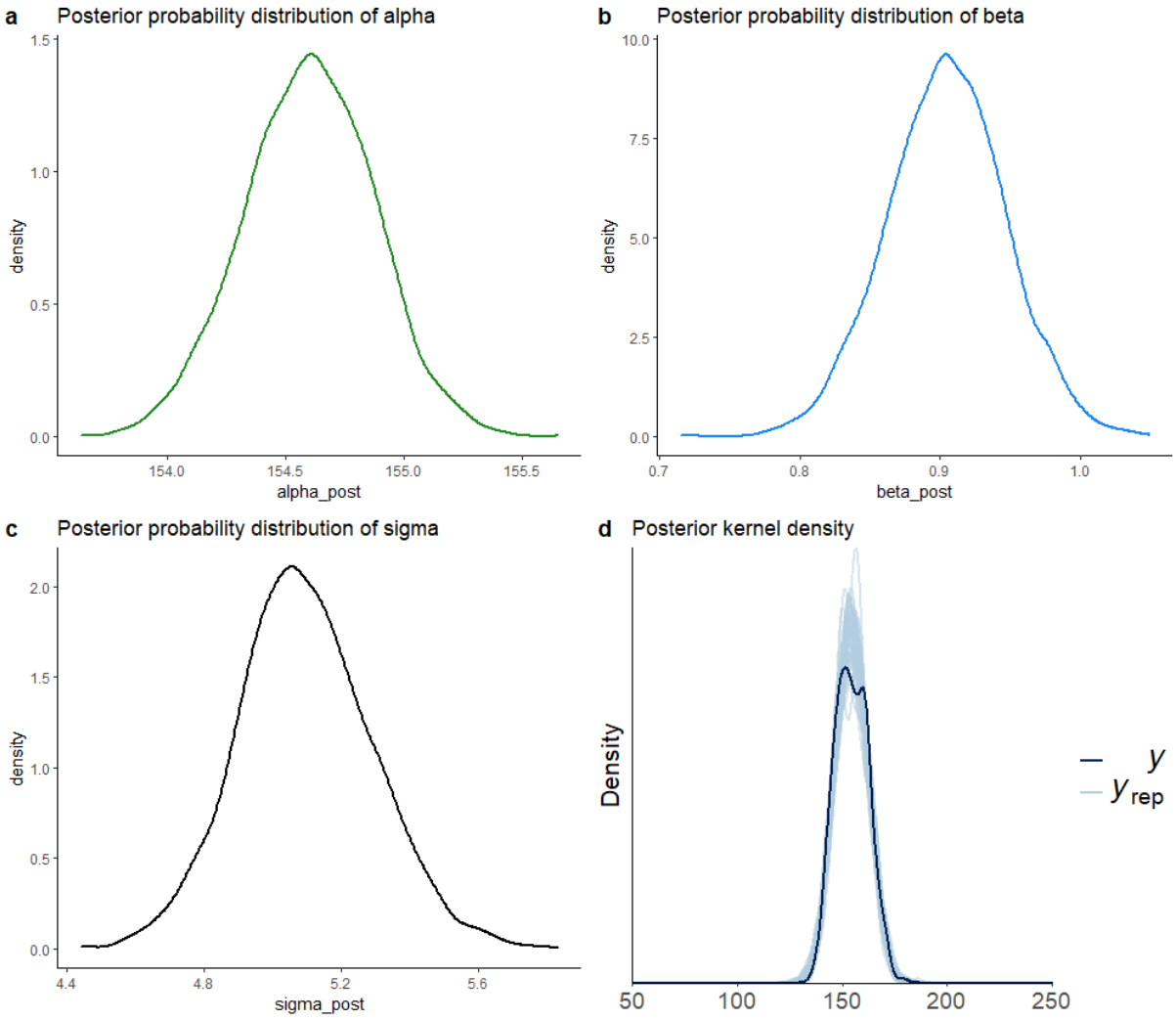

Figure 3.3: Posterior Distributions - **a**. Alpha, **b.** Beta, **c.** Sigma, **d.** Posterior kernel density of predicted data

When compared to the prior predictive kernel density plot, we can see that the simulation based on the updated model results in a much better fit to the original data. The distribution of the predicted values also closely follows the data, indicating that the model with the posterior parameter probability distributions does function as a generative model

for our data.

3.7 Reporting the results of Bayesian inference

The output of Bayesian inference, the posterior, is again a probability distribution. This is very useful for further computation, but sometimes one wants to report a summarized result about the parameter. One way to report the results of Bayesian inference is via the *credible interval*, in which the unobserveable parameter value lies with reasonable certainty. Unlike the confidence interval from frequentist inference, which is a property of the statistical process, the credible interval can be directly interpreted with the data (Kawasaki & Miyaoka, 2010; McElreath, 2019; Nicenboim & Vasishth, 2016; Schad, Betancourt, & Vasishth, 2020; van de Schoot et al., 2021). The Bayesian way of thinking about inference is already quite intuitive for most of us, and we might even already have incorporated it into our frequentist analyses. When we, for example, report a p-value of .06 we tend to resort to expressions of graded differentiation, conveying that the results might not have been significant, but that they were "on the verge of significance" (Nicenboim & Vasishth, 2016). While different types of credible intervals exist, the two most commonly used types are the *highest density interval* (*HDI*), and the *equal-tailed interval* (*ETI*). The *HDI* is the narrowest interval that contains the specified probability mass, usually 95 as with confidence intervals. For the *HDI* it must be true, that all points within the interval have a higher probability density than points outside the interval (Nicenboim & Vasishth, 2016). The *ETI* assigns equal probability mass to each tail, usually resulting in each tail accounting for %2.5%. While both ways of reporting the credible interval are the same for symmetric distributions, they can differ greatly for skewed distributions. Therefore, it is mandatory to also report the type of credible interval when reporting the results of Bayesian inference.

In *R*, we can use the package `bayestestR` (Makowski, Ben-Shachar, & Ludecke, 2019) to calculate the *HDI* and *ETI* for our regression parameters (lst. 3.11). The output is shown in tbl. 3.3.

Table 3.3: Credible intervals for the regression parameters.

95%	HDI	ETI
α	[154.06, 155.14]	[154.05, 155.14]
β	[0.82, 0.98]	[0.82, 0.99]
σ	[4.72, 5.49]	[4.72, 5.49]

This concludes the tutorial on implementing Bayesian inference with *R* and *Stan* for our example linear regression model. The full *Stan* model and the *R* code can be

Listing 3.11 Calculating the credible interval using bayestestR

```r
# extract samples
alpha_post <- rstan::extract(fit_reg_postpc, pars = "alpha", permuted = TRUE)$alpha
beta_post <- rstan::extract(fit_reg_postpc, pars = "beta", permuted = TRUE)$beta
sigma_post <- rstan::extract(fit_reg_postpc, pars = "sigma", permuted = TRUE)$sigma

# Calculate credible intervals for parameters
ci_alpha_eti <- bayestestR::ci(alpha_post, method = "ETI")
ci_alpha_hdi <- bayestestR::ci(alpha_post, method = "HDI")

ci_beta_eti <- bayestestR::ci(beta_post, method = "ETI")
ci_beta_hdi <- bayestestR::ci(beta_post, method = "HDI")

ci_sigma_eti <- bayestestR::ci(sigma_post, method = "ETI")
ci_sigma_hdi <- bayestestR::ci(sigma_post, method = "HDI")
```

found in sec. A.2 and sec. A.3. Of course, this workflow can also be expanded for the application to more complex hierarchical models. As an outlook, I briefly demonstrate the implementation of a hierarchical Rescorla-Wagner reinforcement learning model in *Stan* in the next part.

4. Example 2: Fitting a hierarchical Rescorla-Wagner reinforcement learning model

The Rescorla-Wagner model is a popular cognitive model for studying learning and decision-making. It describes reinforcement learning processes with a relatively simple probabilistic equation. The associated value of an action is continuously updated through the feedback that the subject receives in the form of either reward or punishment. For the increased association of a given action with positive feedback, i.e. reward, the Rescorla-Wagner model postulates an increased probability of this action to be executed again. The higher the rate of reward the higher the probability of repeating this behaviour. The Rescorla-Wagner model is based on the idea of error-driven learning (Zhang, Lengersdorff, Mikus, Gläscher, & Lamm, 2020).

Equation 4.1 describes the Rescorla-Wagner model mathematically (Zhang, Lengersdorff, Mikus, Gläscher, & Lamm, 2020).

$$\begin{aligned} \text{Value update: } V_t &= V_{t-1} + \alpha * PE_{t-1} \\ \text{Prediction error: } PE_{t-1} &= R_{t-1} - V_{t-1} \end{aligned} \tag{4.1}$$

The expected value for the next trial V_t is equal to the expectation for the current trial V_{t-1} plus the product of the learning rate α and the prediction error PE_{t-1}. The prediction error PE_{t-1} is calculated as the difference between the reward of the current trial R_{t-1} and the expected reward for the current trial V_{t-1}.

Combining these two equations gives us eq. 4.2.

$$V_t = V_{t-1} + \alpha * (R_{t-1} - V_{t-1}) \tag{4.2}$$

4.1 The advantages of hierarchical modelling

We have seen that the ability to include prior information into our inference models, and the potential to directly answer the question of interest from the posterior - that is, how plausible our hypothesis is given the data - are one of the biggest advantages of Bayesian inference. Still, there is another great advantage to Bayesian statistics. And that is the ease, with which we can flexibly define hierarchical models. In a hierarchical model, we include population-level distributions from which we then draw the parameters for

the subject-level parameter distributions. This allows us to include individual differences directly in the model, greatly reduces the risk of overfitting and helps us to avoid the need for averaging, which prevents loss of valuable data (Gelman, 2006; Lee, 2011; Nicenboim & Vasishth, 2016; Papaspiliopoulos, Roberts, & Sköld, 2007).

For example, the aforementioned Rescorla-Wagner model of prediction error-based learning includes the learning rate α as a subject-level parameter. For this parameter estimate, we could simply calculate a posterior probability distribution as we did in the linear regression model. When we decide to model it hierarchically, however, we assume that there is another underlying distribution of the learning rate α in the population, from which the subject-level parameter is then drawn.

4.2 Modelling a two-armed bandit task

As a second example, let us consider a simple two-armed bandit task, where participants are presented with two choices for each trial. Each choice is associated with a win probability that is not revealed directly to the participant, however. In this example, option A has a win probability of $p_1(\text{win}) = .25$ and option B has a win probability of $p_2(\text{win}) = .75$. Therefore, choosing option B over option A is beneficial and the subjects are expected to learn this throughout the 100 trials. The probabilities for A and B are directly dependent on each other, that is, a loss when choosing option B means that option A would have resulted in a win on that trial. The Rescorla-Wagner model proposes that the subject attaches an expected value to each of the two options, which increases when the selection of this option is rewarded and decreases when this option is "punished," i.e. was not rewarded. The rate at which the expected value of each option is updated in light of new data is encoded in the learning rate α. A learning rate of $\alpha = 1$ would mean that the expected value only depends on the result of the most recent trial, whereas a learning rate of $\alpha = 0$ would mean that the expected value is never updated. Going from this expected value to the decision for either of the options requires another function. Here, we will make use of the Softmax function, which is a normalized exponential function that transforms an N-dimensional original vector with arbitrary real values into an N-dimensional probability vector with real values in the range $[0, 1]$ that add up to 1 (Gao & Pavel, 2018; He, Zhang, Ao, & Huang, 2018; Zhang, Lengersdorff, Mikus, Gläscher, & Lamm, 2020). This allows us to translate the value difference $(V(A) - V(B))$ into an action probability, i.e. a probability to choose one of the two options over the other, e.g. A over B (see eq. 4.3).

$$\begin{aligned} p_t(A) &= \frac{e^{\tau * V_t(A)}}{e^{\tau * V_t(A)} + e^{\tau * V_t(B)}} \\ &= \frac{1}{1 + e^{-\tau * (V_t(A) - V_t(B))}} \end{aligned} \tag{4.3}$$

The Softmax function uses the *inverse temperature parameter* ($\tau > 0$), which turns the Softmax function into a logistic curve, where the input is the value difference between $V(A)$ and $V(B)$, and the output is the probability of choosing A (eq. 4.3). a small value of τ results in a shallow curve, representing more random choices, i.e. less consistent preference for the higher-value option, while a larger value of τ results in a steeper curve and more consistent choices (Zhang, Lengersdorff, Mikus, Gläscher, & Lamm, 2020).

Taken together, the learning rate α and the Softmax *inverse temperature parameter* form the basis of our simple Rescorla-Wagner modelling parameters. To demonstrate the modelling process for the Rescorla-Wagner model, data from a simulation study with 10 participants and 100 trials each has been generated using the code in sec. A.6. The true parameters used for the data generation are shown in lst. 4.1.

Listing 4.1 True parameters for the RW data generation

```
# alpha
lr <- rnorm(10, mean = 0.6, sd = 0.12)

# tau
tau <- rnorm(10, mean = 1.5, sd = 0.2)
```

The entire code for the *Stan* model for the two-armed bandit task is shown in lst. A.4.

4.3 Defining population-level priors for the Rescorla-Wagner model

A crucial difference to the linear regression model is the inclusion of population-level parameters. In the `model` block in addition to subject-level priors for the learning rate and the inverse temperature parameter we also have to include distributions for the mean and standard deviation of the population-level distributions. Here, the subject-level distribution for the learning rate α is described by two parameters μ_α and σ_α, that in themselves follow distributions on the population-level (eq. 4.4). The same is true for the distributions of the inverse temperature parameter τ (eq. 4.5). The *Stan* code to define these distributions is shown in lst. 4.3.

$$\begin{aligned} \alpha &\sim \text{Normal}(\mu_\alpha, \sigma_\alpha) \\ \mu_\alpha &\sim \text{Normal}(0, 1) \\ \sigma_\alpha &\sim \text{Normal}(0, 1) \end{aligned} \tag{4.4}$$

$$\begin{aligned} \tau &\sim \text{Normal}(\mu_\tau, \sigma_\tau) \\ \mu_\tau &\sim \text{Normal}(0, 1) \\ \sigma_\tau &\sim \text{Normal}(0, 1) \end{aligned} \tag{4.5}$$

For the mean μ of the learning rate α and the inverse temperature parameter τ we choose a weakly informative prior with a normal distribution $N(0, 1)$ for the mean. While it has been recommended in previous studies to use a Cauchy distribution, e.g. $Cauchy(0, 3)$, as the prior for standard deviations (He, Zhang, Ao, & Huang, 2018; Zhang, Lengersdorff, Mikus, Gläscher, & Lamm, 2020), the transformation of the distribution to the `[0,1]` parameter space due to the `Phi_approx` function in combination with the heavy tails of the Cauchy distribution causes a lot of accumulation of probability mass at the boundaries. To avoid this side-effect, we choose a $Normal(0, 2)$ for the standard deviation instead. The difference between these two prior distributions on the subject level is illustrated in fig. 4.1. The prior distributions for the learning rate and the inverse temperature parameter that use a $Normal(0, 2)$ distribution (a and c) for the standard deviation are shown on the left-hand side, those that use the $Cauchy(0, 3)$ distribution (b and d) are shown on the right-hand side.

While α is bounded between `[0,1]`, the inverse temperature parameter τ has no natural upper bound. However, as *Stan* is more efficient when handling parametrized distributions (Carpenter et al., 2017; Gosselin et al., 2017; Papaspiliopoulos, Roberts, & Sköld, 2007), we define τ here with normal distributions and then scale the distribution on the subject level in the *transformed parameters* block of the *Stan* code (lst. 4.2). Via the `Phi_approx` function, the raw mean, the raw standard deviation and the raw individual learning rate are combined to generate one distribution of the individual learning rate per subject and constrain it between `0.0` and `1.0`. Similarly, the inverse temperature parameter τ is mapped to the `[0, 50]` parameter space as we multiply the resulting function with `50` (lst. 4.2, l. `7`). According to (Gershman, 2016), this parameter space for τ covers the necessary possible values, although in practice, also smaller parameter spaces, like `[0, 3]` are common (Ghosh, Li, & Mitra, 2017; He, Zhang, Ao, & Huang, 2018; Zhang, Lengersdorff, Mikus, Gläscher, & Lamm, 2020).

In fig. 4.2 we see the posterior density distributions for the learning rate (fig. 4.2.a), and for the inverse temperature parameter (fig. 4.2.b) for the population and for each participant individually. In order to improve computation, the distributions are kept in a standardized form, e.g. $Normal(0, 1)$, for as long as possible.

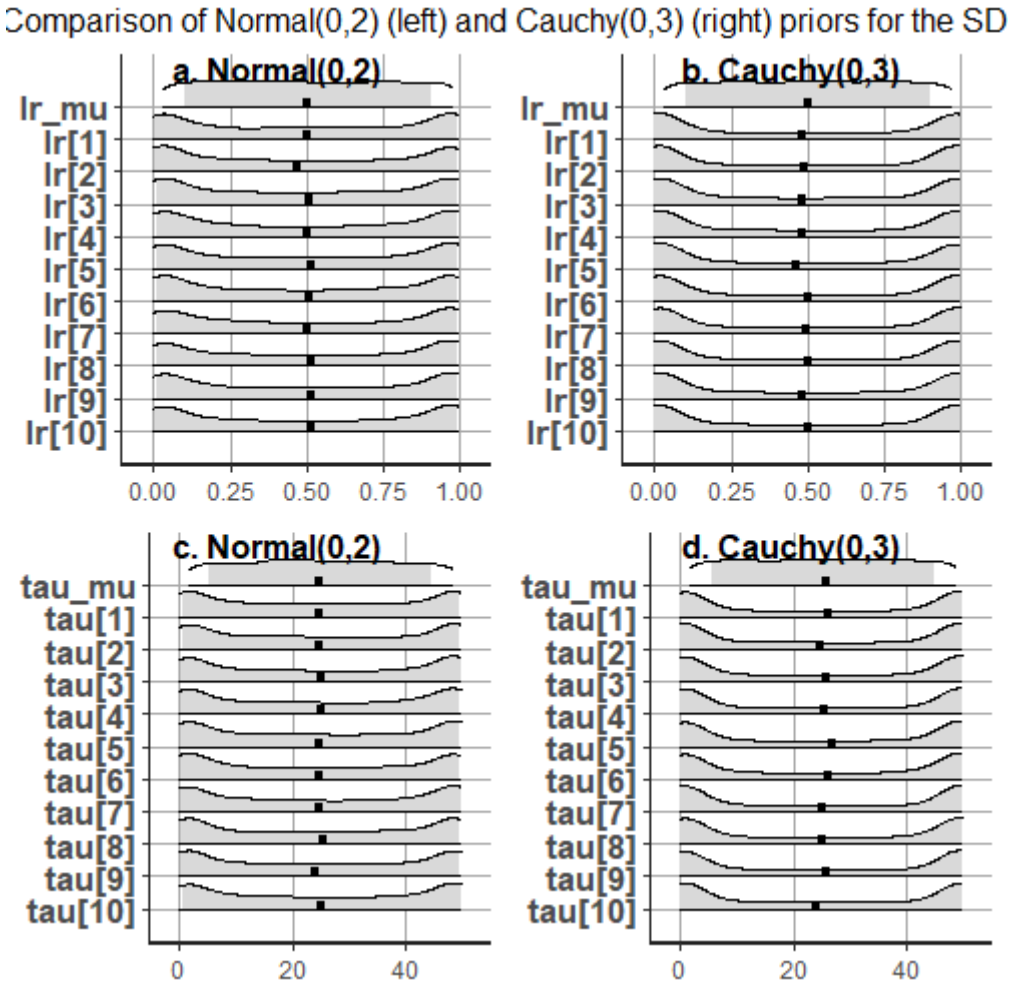

Figure 4.1: Effect of different prior distributions for the standard deviation in the Rescorla-Wagner model - **a.** and **b.** for the learning rate, **c.** and **d.** for the inverse temperature parameter

4.4 The Rescorla-Wagner `transformed parameters` block

"The transformed parameters program block consists of optional variable declarations followed by statements. After the statements are executed, the constraints on the transformed parameters are validated. Any variable declared as a transformed parameter is part of the output produced for draws. Any variable that is defined wholly in terms of data or transformed data should be declared and defined in the transformed data block. Defining such quantities in the transformed parameters block is legal, but much less efficient than defining them as transformed data" (Stan Development Team, 2021).

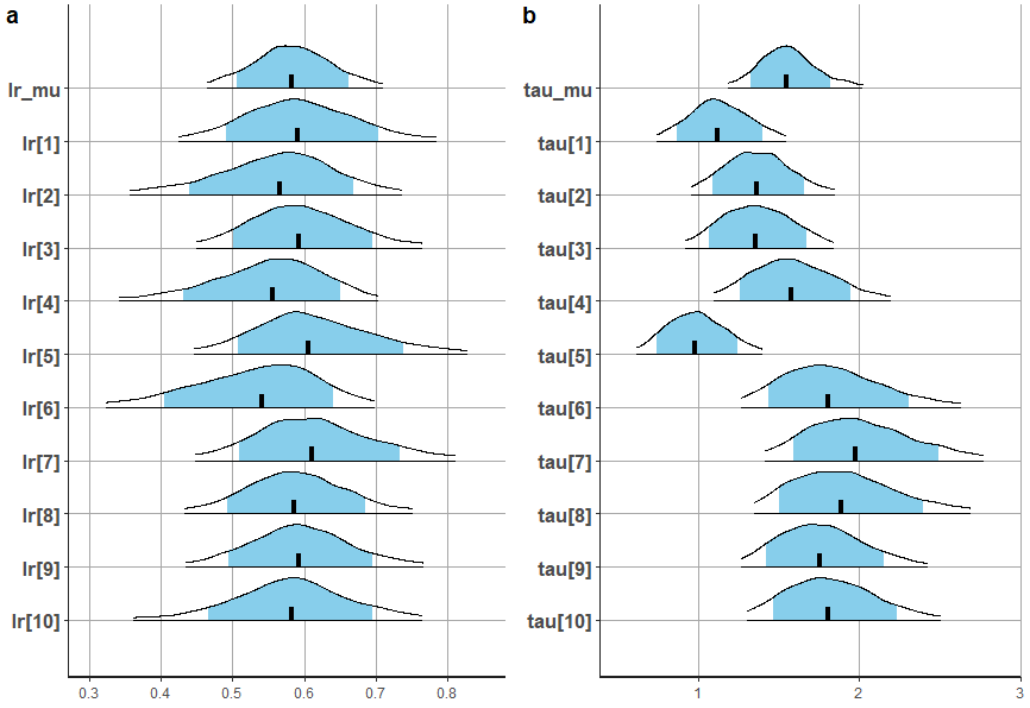

Figure 4.2: Posterior density distributions for the Rescorla-Wagner model for **a.** the learning rate and **b.** the inverse temperature parameter

Listing 4.2 The transformed parameters block for the Rescorla-Wagner model

```
1 transformed parameters {
2   vector<lower=0,upper=1>[nSubjects] lr;
3   vector<lower=0,upper=50>[nSubjects] tau;
4
5   for (s in 1:nSubjects) {
6     lr[s]  = Phi_approx( lr_mu_raw  + lr_sd_raw * lr_raw[s] );
7     tau[s] = Phi_approx( tau_mu_raw + tau_sd_raw * tau_raw[s] ) * 50;
8   }
9 }
```

4.5 The Rescorla-Wagner `model` block

Listing 4.3 shows how the prior distributions can also be set directly from within *Stan*. Between l. `11` and `22` the likelihood function of the model is defined. As we have more than one subject and for each subject a time series of trial data, we first loop over each trial for every subject. In each trial loop (ll. `16-20`) we first model a choice using the Softmax function and the inverse temperature parameter τ (`tau`) and the value (`v`) associated with the two choices A and B (l. `12`). For the first trial, the value is `0` for either of them so the choice is completely random. Next, we calculate the prediction error (l. `13`) by subtracting the expected value from the received reward. Finally, we update the expected value for each choice by adding the product of the prediction error (`pe`) with the learning rate of that subject (`lr[s]`). To recall, the higher the learning rate, the faster

new information is used to update the value expectations.

Listing 4.3 Setting the prior distributions in the model block of the Rescorla-Wagner model

```
model {
  lr_mu_raw  ~ normal(0,1);
  tau_mu_raw ~ normal(0,1);
  lr_sd_raw  ~ normal(0,2);
  tau_sd_raw ~ normal(0,2);

  lr_raw  ~ normal(0,1);
  tau_raw ~ normal(0,1);

  if (analysis_step == 1) {
    for (s in 1:nSubjects) {
      vector[2] v;
      real pe;
      v = initV;

      for (t in 1:nTrials) {
        choice[s,t] ~ categorical_logit( tau[s] * v );
        pe = reward[s,t] - v[choice[s,t]];
        v[choice[s,t]] = v[choice[s,t]] + lr[s] * pe;
      }
    }
  }
}
```

4.6 The Rescorla-Wagner `generated quantities` block

Listing 4.4 shows the final model block, where we simulate the choice (`y_pred`) for each subject in each trial based on random draws from the learning rate and inverse temperature parameter distributions for that subject. This generated data can then be used to conduct the prior and posterior predictive checks.

4.7 Calculating the posterior

Running the Rescorla-Wagner *Stan* model from *R* using `rstan` works exactly like the call of the linear regression model. We feed our data into the data list so that it becomes available within *Stan* and can be declared in the `data` block. We decide on the number of chains, the number of iterations, the warmup duration whether to thin the data, how the initial values are to be set, and whether we want to set a random number generation seed for reproducibility. As this is a hierarchical model, however, when we extract the posterior

Listing 4.4 Setting the prior distributions in the model block of the Rescorla-Wagner model

```
generated quantities {
  real<lower=0,upper=1> lr_mu; // declaring the mean of the learning rate
  real<lower=0,upper=50> tau_mu; // declaring the mean of tau

  int y_pred[nSubjects, nTrials]; // declaring the modelled choice (predicted)

  lr_mu  = Phi_approx(lr_mu_raw);
  tau_mu = Phi_approx(tau_mu_raw) * 50;
  y_pred = rep_array(-999,nSubjects ,nTrials);

  { // local block
    for (s in 1:nSubjects) {
      vector[2] v;
      real pe;
      v = initV;
      for (t in 1:nTrials) {
        y_pred[s,t] = categorical_logit_rng( tau[s] * v ); // draws one tau value from
 ↪  the tau distribution for this iteration of the MCMC
        pe = reward[s,t] - v[choice[s,t]]; // prediction error according to the reward
 ↪  and choice from the data
        v[choice[s,t]] = v[choice[s,t]] + lr[s] * pe; // choice update according to
 ↪  the data
      }
    }
  }
}
```

distributions, we will receive both population-level and subject-level posterior probability distributions. Figure 4.2 shows a visual representation of the posterior distributions.

4.8 Prior predictive check

In order to confirm that the chosen priors yield sensible results, we conduct a prior predictive check. Unlike the data from the linear regression example, the data from the two-armed bandit task is not only hierarchical but also longitudinal, as we are interested in the development of the per-trial performance over time. We have previously obtained the generated data `y_pred` based on the posterior distributions in the `generated_quantities` block. The prior predictive check is illustrated in fig. 4.3.a. The mean of the "correct" choices, that is choosing option B with the higher win probability, is displayed for each trial. The same is done for the simulated data with the 95 *highest density interval* for that trial. We see that, overall, subjects seem to perform relatively well after a few trial. The modelled data based on the prior information allows for a relatively wide range. This is to be expected as we chose a weakly informative prior. Still, the model does not appear to produce any invalid data. Therefore, we can say that this check confirms that the data

generation based on parameter draws from the prior distribution produces sensible results. The code for this check was adapted from Zhang (2021).

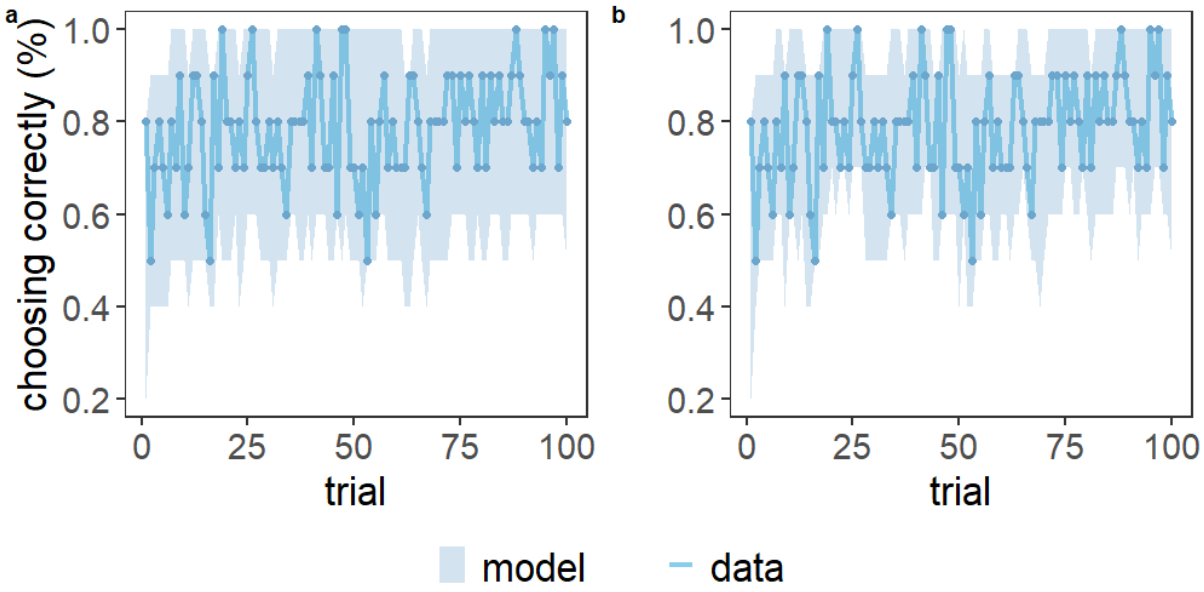

Figure 4.3: **a.** Prior predictive check (left). **b.** posterior predictive check (right) for the two-armed bandit task

4.9 Posterior predictive check

The posterior predictive check is conducted in the same way as the prior predictive check. The result is displayed in fig. 4.3.b. We see that the predicted data fit well with the actual data, especially when we consider that we only used a sample size of ten (simulated)aparticipants. Compared with the prior predictive check, the posterior predictive check reveals a narrower *HDI*, which is to be expected as the posterior has been updated with the likelihood, resulting in a narrower posterior distribution.

4.10 Reporting the result as parameter estimates

Again, to summarize the inference we can use the method of the highest density interval *HDI* to calculate the 95 credible interval for the population-level learning rate `lr_mu` and the inverse temperature parameter `tau_mu`. The results are shown in tbl. 4.1. The true mean of *alpha* used for data generation was 0.6, and 1.5 for τ. Here we see that the "true" value is indeed "right in the middle" of our credible interval, despite the relatively small sample size.

Table 4.1: Credible intervals for the Rescorla-Wagner model parameters.

95%	HDI	ETI
`lr_mu` (α)	[0.46, 0.70]	[0.47, 0.72]
`tau_mu` (τ)	[1.22, 1.82]	[1.23, 1.84]

This concludes the second example, building on the workflow of the linear regression and demonstrating the implementation of a more complex hierarchical Rescorla-Wagner reinforcement learning model with *Stan*. The full *R* and *Stan* code for the Rescorla-Wagner model can be found in lst. A.4.

5. Closing considerations

While other similar workflow tutorials already exist (Gabry, Simpson, Vehtari, Betancourt, & Gelman, 2019; Gelman et al., 2020; Schad, Betancourt, & Vasishth, 2020; e.g. Wilson & Collins, 2019), there is still a necessity for accessible tutorials that introduce a full and repeatable Bayesian workflow that use tools already available to the researchers, especially for those that could benefit from including computational modelling in their toolbox. We have demonstrated the potential and flexibility of this workflow, by applying it to two repeatable example models of differing complexity. This highlights the versatility of this method and shows how easy it is to apply this workflow to different types of (cognitive) models. We have shown step by step the relevant Bayesian workflow elements and how to perform them from within the widespread statistical language *R* using the package `rstan`.

The workflow demonstrated in this thesis should prove especially useful for cognitive scientists who want to get started with computational and hierarchical modelling. As we have seen, hierarchical modelling is a natural extension of the Bayesian workflow and advanced MCMC sampling algorithms, like HMC, allow for the estimation of posterior distributions even in complex models where hundreds of integrals need to be calculated. Additionally, while cognitive scientists often deal with relatively small sample sizes, they can usually rely on rather abundant domain expertise. This Bayesian workflow allows for the seamless integration of domain expertise via the definition of prior distributions. This allows for significantly more efficient inference as we do not have to start from scratch every time we model data. Instead, we can build on previous findings. And once a generative model has been created, it can then be used again and be updated with data from new studies. Another benefit to Bayesian inference is, that it allows us to directly answer our question of interest with the data at hand, as the posterior density distribution shows us directly the probability of all parameter values given the data at hand. With increasing amounts of data, the estimation of the "true" parameter value gets more precise - as long as the model is appropriate -, but this process is a continuous one. There are no arbitrary cut-off values or statistical parameters that define whether a test result was significant. More data leads to a more precise posterior, that is the gist of it.

With the inclusion of prior information, however, we can improve the amount of information that we extract from the data at hand. For this, the formulation of the prior distribution, especially for small sample sizes can have a large impact on the posterior.

In this tutorial, we therefore also demonstrated predictive checks, that help evaluate the generative abilities of the model. Prior predictive checks generate data from the model based on parameter values that were drawn from the prior distributions. Visualizing this predicted data helps identify whether the prior allows for "impossible" values. If that should be the case, the prior distributions can be modified to improve the model fitting process. Posterior predictive checks are similar to prior predictive checks as they also help to evaluate the model, however, after the fitting process. This allows us to evaluate the generative properties of the fitted model, and can even be used to compare different models for the same data, although this was not part of the scope of this thesis (for further information on this use of posterior predictive checking see Berkhof, van Mechelen, & Hoijtink, 2000; Ranganath & Blei, 2019).

While Bayesian inference has been gaining increasing popularity in psychology and related fields, many articles fail to discuss their choice of priors, which might be facilitated by a lack of deeper understanding of the exact analytical workflow, as new software makes Bayesian analysis more accessible (van de Schoot, Winter, Ryan, Zondervan-Zwijnenburg, & Depaoli, 2017). I believe that Bayesian inference holds great potential for a wide number of applications and should therefore be made accessible to researchers, but I also think that it is equally important to promote a mindset of transparent research techniques and to deepen the understanding of the modelling process beyond the simple application of a procedure to some data. This workflow forces the analyst to make their assumptions about the data as well as the underlying model explicit, which encourages a deeper understanding of the workflow steps, and ultimately improving transparency and replicability in the analytical process. In this light, the sharing of data and source code used for the analysis alongside the article or on a freely accessible repository hosting site should be further encouraged in the spirit of open science.

In this tutorial, I have introduced the main concepts and steps necessary to get started with Bayesian computation modelling. It aims to make Bayesian statistics more accessible for those interested in computation modelling and familiar with *R*. Following the code examples, you can develop your understanding, create computation models and perform Bayesian inference on your own data sets.

6. References

Akaike, H. (1980). The Interpretation of Improper Prior Distributions as Limits of Data Dependent Proper Prior Distributions. *Journal of the Royal Statistical Society. Series B (Methodological)*, *42*(1), 46–52. https://doi.org/gk4dgn

Berkhof, J., van Mechelen, I., & Hoijtink, H. (2000). Posterior predictive checks: Principles and discussion. *Computational Statistics*, *15*(3), 337–354. https://doi.org/cffrq3

Betancourt, M. (2018). A Conceptual Introduction to Hamiltonian Monte Carlo. *arXiv:1701.02434 [Stat]*. Retrieved from http://arxiv.org/abs/1701.02434

Box, G. E. P. (1976). Science and statistics. *Journal of the American Statistical Association*, *71*(356), 791–799. https://doi.org/gdm28w

Carpenter, B., Gelman, A., Hoffman, M. D., Lee, D., Goodrich, B., Betancourt, M., … Riddell, A. (2017). Stan: A Probabilistic Programming Language. *Journal of Statistical Software*, *76*(1). https://doi.org/b2pm

Gabry, J., & Mahr, T. (2021). *Bayesplot: Plotting for bayesian models.*

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. https://doi.org/gftqws

Gao, B., & Pavel, L. (2018). On the Properties of the Softmax Function with Application in Game Theory and Reinforcement Learning. *arXiv:1704.00805 [Cs, Math]*. Retrieved from http://arxiv.org/abs/1704.00805

Gelman, A. (2006). Prior distributions for variance parameters in hierarchical models. *Bayesian Analysis*, *1*(3), 515–534. https://doi.org/bh3qjf

Gelman, A., Aki Vehtari, Daniel Simpsons, Charles C. Margossian, Bob Carpenter, Yuling Yao, … Martin Modrák. (2020). *Bayesian Workflow* (p. 77).

Gershman, S. J. (2016). Empirical priors for reinforcement learning models. *Journal of Mathematical Psychology*, *71*, 1–6. https://doi.org/f8nb9c

Ghosh, J., Li, Y., & Mitra, R. (2017). On the Use of Cauchy Prior Distributions for Bayesian Logistic Regression. *arXiv:1507.07170 [Stat]*. Retrieved from http://arxiv.org/abs/1507.07170

Gosselin, J. M., Dosso, S. E., Cassidy, J. F., Quijano, J. E., Molnar, S., & Dettmer, J. (2017). A gradient-based model parametrization using Bernstein polynomials in Bayesian inversion of surface wave dispersion. *Geophysical Journal International*, *211*(1), 528–540. https://doi.org/gb27bm

Guest, O., & Martin, A. E. (2021). How Computational Modeling Can Force Theory Building in Psychological Science. *Perspectives on Psychological Science*, *16*(4), 789–802. https://doi.org/ghvcwr

He, Y.-L., Zhang, X.-L., Ao, W., & Huang, J. Z. (2018). Determining the optimal temperature parameter for Softmax function in reinforcement learning. *Applied Soft Computing*, *70*, 80–85. https://doi.org/gd9db7

Howell, N. (2000). *Demography of the dobe !Kung*. Aldine de Gruyter.

Kassambara, A. (2020). *Ggpubr: 'ggplot2' based publication ready plots* [Manual].

Kawasaki, Y., & Miyaoka, E. (2010). A Posterior Density for the Difference Between Two Binominal Proportions and the Highest Posterior Density Credible Interval. *JOURNAL OF THE JAPAN STATISTICAL SOCIETY*, *40*(2), 265–275. https://doi.org/10.14490/jjss.40.265

Kelter, R. (2020). Bayesian Survival Analysis in STAN for Improved Measuring of Uncertainty in Parameter Estimates. *Measurement: Interdisciplinary Research and Perspectives*, *18*(2), 101–109. https://doi.org/gjbsq4

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. https://doi.org/ghpdbj

Lambert, B. (2018). *A Student's Guide to Bayesian Statistics* (Vol. 1).

Laplace, P. S. (1986). Memoir on the Probability of the Causes of Events. *Statistical Science*, *1*(3), 364–378. https://doi.org/ck7d8k

Lee, M. D. (2011). How cognitive modeling can benefit from hierarchical Bayesian models. *Journal of Mathematical Psychology*, *55*(1), 1–7. https://doi.org/fpxz2v

Levy, R. (2011). Bayesian Data-Model Fit Assessment for Structural Equation Modeling. *Structural Equation Modeling: A Multidisciplinary Journal*, *18*(4), 663–685. https://doi.org/c4dsgc

Makowski, D., Ben-Shachar, M. S., & Ludecke, D. (2019). bayestestR: Describing effects and their uncertainty, existence and significance within the bayesian framework. *Journal of Open Source Software*, *4*(40), 1541. https://doi.org/gf9pds

McElreath, R. (2019). *Statistical Rethinking - A Bayesian Course with Examples in R and STAN* (Vol. 1).

McGrayne, S. B. (2012). *The Theory That Would Not Die: How Bayes' Rule Cracked the Enigma Code, Hunted Down Russian Submarines, and Emerged Triumphant from Two Centuries of Controversy*. Yale University Press.

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. https://doi.org/ggkqvm

Nicenboim, B., & Vasishth, S. (2016). Statistical methods for linguistic research: Foundational Ideas - Part II. *Language and Linguistics Compass*, *10*(11), 591–613. https:

//doi.org/gcpb49

Papaspiliopoulos, O., Roberts, G. O., & Sköld, M. (2007). A General Framework for the Parametrization of Hierarchical Models. *Statistical Science*, *22*(1). https://doi.org/b9h6bs

R Core Team. (2021). *R: A language and environment for statistical computing* [Manual]. Vienna, Austria: R Foundation for Statistical Computing.

Ranganath, R., & Blei, D. M. (2019). Population Predictive Checks. *arXiv:1908.00882 [Cs, Stat]*. Retrieved from http://arxiv.org/abs/1908.00882

Roser, M., Appel, C., & Ritchie, H. (2013). Human Height. *Our World in Data*.

Sarma, A., & Kay, M. (2020). Prior Setting in Practice: Strategies and Rationales Used in Choosing Prior Distributions for Bayesian Analysis. *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, 1–12. https://doi.org/ghf94v

Schad, D. J., Betancourt, M., & Vasishth, S. (2020). Toward a principled Bayesian workflow in cognitive science. *Psychological Methods*. https://doi.org/ghbtt6

Stan Development Team. (2020). *RStan: The R interface to Stan*.

Stan Development Team. (2021). *Stan Modeling Language Users Guide and Reference Manual, ver. 2.27*.

Stone, M. (1976). Strong inconsistency from uniform priors. *Journal of the American Statistical Association*, *71*(353), 114–116. https://doi.org/10.1080/01621459.1976.10481490

Ushey, K., Allaire, J., Wickham, H., & Ritchie, G. (2020). *Rstudioapi: Safely access the RStudio API* [Manual].

van de Schoot, R., Depaoli, S., King, R., Kramer, B., Märtens, K., Tadesse, M. G., … Yau, C. (2021). Bayesian statistics and modelling. *Nature Reviews Methods Primers*, *1*(1), 1–26. https://doi.org/ghs29x

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. https://doi.org/gbj7nw

Wickham, H. (2016). *Ggplot2: Elegant graphics for data analysis*. Springer-Verlag New York.

Wilson, R. C., & Collins, A. G. (2019). Ten simple rules for the computational modeling of behavioral data. *eLife*, *8*, e49547. https://doi.org/ggdtd2

Wolodzko, T. (2020). *extraDistr: Additional univariate and multivariate distributions* [Manual].

Zhang, L. (2021). *Lei-zhang/BayesCog_Wien*.

Zhang, L., Lengersdorff, L., Mikus, N., Gläscher, J., & Lamm, C. (2020). Using reinforcement learning models in social neuroscience: Frameworks, pitfalls and suggestions of best practices. *Social Cognitive and Affective Neuroscience*, *15*(6), 695–707. https://doi.org/gg9n5t

Zundert, C. van, Somer, E., & Miocevic, M. (2020). *Prior Predictive Checks for the Method of Covariances in Bayesian Mediation Analysis*. https://doi.org/10.31234/osf.io/j2h5z

List of Figures

List of Tables

List of Listings

A. Appendices

A.1 Session info

```
> sessionInfo()
R version 4.0.5 (2021-03-31)
Platform: x86_64-w64-mingw32/x64 (64-bit)
Running under: Windows >= 8 x64 (build 9200)

Matrix products: default

locale:
[1] LC_COLLATE=Japanese_Japan.932  LC_CTYPE=Japanese_Japan.932
 ↪  LC_MONETARY=Japanese_Japan.932 LC_NUMERIC=C
 ↪  LC_TIME=Japanese_Japan.932
system code page: 65001

attached base packages:
[1] stats     graphics  grDevices utils     datasets  methods   base

other attached packages:
[1] extraDistr_1.9.1     ggpubr_0.4.0         rstudioapi_0.13      bayestestR_0.10.0
 ↪  bayesplot_1.8.0       rstan_2.21.2         ggplot2_3.3.3
 ↪  StanHeaders_2.21.0-7
```

A.2 Commented *Stan* model for the linear regression

```
data {
  int<lower=0> N; // This is the number of data points, i.e. participants, in the case
   ↪  of prior predictive checks this encodes the number of simulations
  vector<lower=0>[N] weight; // In this regression, the weight is the independent
   ↪  variable. We are iterating on a model that allows us to predict the height from
   ↪  the weight in the data.
  real<lower=0> mean_weight;
  vector<lower=0>[N] height; // In the prior predictive check we are simulating the
   ↪  posterior distribution based only on the prior. The dependent variable `height`
   ↪  is only necessary when we include the likelihood function and conduct the
   ↪  posterior predictive check.
  int<lower=0, upper=1> analysis_step; // boolean variable for the switch that allows
   ↪  us to use the same script with little variation depending on whether we are
   ↪  running the posterior (1) or the prior (0) predictive checks

```

```
  real prior_alpha[2]; // mean and standard deviation of the prior distribution from
   ↪  which we draw alpha (the intercept)
  real<lower=0> prior_beta[2]; // mean and standard deviation of the half-normal prior
   ↪  distribution from which we draw beta (the slope)
  real<lower=0> prior_sigma[2]; // shape parameters of the half-cauchy prior
   ↪  distribution (with a lower bound at 0) from which we draw sigma (the noise term)
}

parameters {
  real alpha; // declaring the intercept
  real<lower=0> beta; // declaring the slope
  real<lower=0> sigma; // declaring the standard deviation of the error term (cannot
   ↪  be negative)
}

model {
  // priors
  alpha ~ normal(prior_alpha[1], prior_alpha[2]); // drawing the intercept
  beta  ~ normal(prior_beta[1], prior_beta[2]); // drawing the slope
  sigma ~ cauchy(prior_sigma[1], prior_sigma[2]); // drawing the noise term

  if (analysis_step == 1) {
    height ~ normal(alpha + beta * (weight - mean_weight), sigma); // this is the
 ↪  likelihood function that links the data (weight and height) with our priors. How
 ↪  likely is the data for a given parameter? P(y|Phi). Only active during the
 ↪  posterior predictive check and the only difference in the scripts function between
 ↪  prior predictive checks and posterior predictive checks.
  }
}

generated quantities {
  vector[N] height_bar; // this is the modelled height
   // here we loop over all N to draw a value for the simulated height from our
    ↪  distribution
   for (n in 1:N) {
      height_bar[n] = normal_rng(alpha + beta * (weight[n] - mean_weight), sigma); //
 ↪  calculate posterior distributions
  }
}
```

A.3 Commented code to run the linear regression from *R*

```
# Linear regression model
# Running both the prior predictive and the posterior predictive checks using Stan

# Jakob Weickmann, 2021
# jakob.weickmann@posteo.de
# in cooperation with Damian Bednarz and Lei Zhang
# damian.bednarz@posteo.de, lei-zhang.net

# Adapted from Richard McElreath, 2016 and the data set "Howell1"
# This data set is based on the data set from Nancy Howell accessible under
```

```r
# https://tspace.library.utoronto.ca/handle/1807/18219

#### Construct Data ####
# Load necessary packages
library(rstan)
library(ggplot2)
library(bayesplot)
library(bayestestR)
library(rstudioapi)
library(ggpubr)
library(extraDistr)

# Set some options
# automatically saves a serialized version of the compiled model to the directory of the `.stan` file.
rstan_options(auto_write = TRUE)

# detects the number of cores available and runs that many chains in parallel. This allows maximum performance, but the user may want to manually set the number of cores if not all cores should be used.
options(mc.cores = parallel::detectCores())

ggplot2::theme_set(theme_classic())
# Loading the data
# Read in the Howell1 data set from the github repository of Richard McElreath
d <- read.csv(
  "https://raw.githubusercontent.com/jweickm/ppc_tutorial/main/data/Howell1.csv",
   sep = ";")
str(d)

df <- d[ d$age >=  18, ] # filter only for the adults
str(df)

custom.summary <- function(x, ...){
  c(mean   = mean(x, ...),
    sd     = sd(x, ...),
    median = median(x, ...),
    min    = min(x, ...),
    max    = max(x, ...),
    n      = as.integer(length(x, ...)))
}

df_summary <- apply(df, 2, custom.summary)
mean_weight <- df_summary["mean", "weight"]

# define the values for the switch variable. This allows us to use the same Stan script for both the prior predictive checks and the posterior predictive checks
priorCheck <- 0 # 0: Prior predictive check
posteriorCheck <- 1 # 1: Posterior predictive check

# The linear regression model looks as follows:
# height_predicted_i = alpha + beta * (weight_i - mean_weight) + epsilon_i

# Defining the prior distribution (for the regression parameters) based on the existing domain-specific knowledge
prior_alpha <- c(165, 30) # sets the mean and standard deviation of the normal prior distribution for the intercept, where xi = mean(x) (alpha in the linear model), in cm
```

```r
prior_beta  <- c(0, 4) # sets the mean and standard deviation of the half-normal prior distribution for the slope (beta), in cm/kg
prior_sigma <- c(0, 5) # sets the shape parameters for the half-Cauchy prior distribution of the noise term (sigma)

# INITIALIZING STAN
# Set the stan file and define the MCMC parameters
modelFile <- "regression_height_combined.stan"
nIter     <- 2000
nChains   <- 4
nWarmup   <- floor(nIter/2)
nThin     <- 1

N <- length(df$height)

dataList <- list(N = N,
                 weight = df$weight,
                 mean_weight = mean_weight,
                 height = df$height,
                 prior_alpha = prior_alpha,
                 prior_beta = prior_beta,
                 prior_sigma = prior_sigma,
                 analysis_step = -1)
Seed = 1450154627

#### Running Stan with prior predictive checks ####
# Let Stan know that we want to conduct a prior predictive check
dataList$analysis_step <- priorCheck

cat("Estimating", modelFile, "model... \n")
startTime = Sys.time(); print(startTime)
cat("Calling", nChains, "simulations in Stan... \n")

fit_reg_pripc <- stan(modelFile,
                      model_name = "Linear Regression Height Prior Predictive Check",
                      data    = dataList,
                      chains  = nChains,
                      iter    = nIter,
                      warmup  = nWarmup,
                      thin    = nThin,
                      init    = "random",
                      seed    = Seed)

cat("Finishing", modelFile, "model simulation ... \n")
endTime = Sys.time(); print(endTime)
cat("It took", as.character.Date(endTime - startTime), "\n")

#### Running Stan with posterior predictive checks ####
# Let Stan know that we want to include the likelihood to update the posterior
dataList$analysis_step <- posteriorCheck

cat("Estimating", modelFile, "model... \n")
startTime = Sys.time(); print(startTime)
cat("Calling", nChains, "simulations in Stan... \n")

fit_reg_postpc <- stan(modelFile,
                       model_name = "Linear Regression Height Posterior Predictive Check",
                       data    = dataList,
```

```
                  chains  = nChains,
                  iter    = nIter,
                  warmup  = nWarmup,
                  thin    = nThin,
                  init    = "random",
                  seed    = Seed)

cat("Finishing", modelFile, "model simulation ... \n")
endTime = Sys.time(); print(endTime)
cat("It took",as.character.Date(endTime - startTime), "\n")

#### Visualizing prior predictive checks ####
# extract samples from the generated stan model
height_bar <- rstan::extract(fit_reg_pripc, pars = "height_bar", permuted =
 ↪  TRUE)$height_bar

# Initial plot of the data
scatterplot <- ggplot2::ggplot(df, aes(weight, height)) +
  ggplot2::geom_point() +
  ggplot2::geom_smooth(method = 'lm', colour = "indianred3") +
  ggplot2::labs(title = "Scatterplot of height ~ weight with linear regression line",
       x = "weight in kg",
       y = "height in cm")

# Plot for alpha prior distribution
p_alpha <- ggplot2::ggplot() +
  xlim(-10, 300) +
  ggplot2::geom_function(fun = dnorm, args = list(mean = prior_alpha[1], sd =
 ↪  prior_alpha[2]), colour = "forestgreen", size=1) +
  ggplot2::geom_vline(aes(xintercept = 272), linetype = 'dotted') +
  ggplot2::annotate("text", x = 260, y = 0.007, label = "tallest person ever recorded
 ↪  (272 cm)", angle = 90) +
  ggplot2::geom_vline(aes(xintercept = 0), linetype = 'dotted') +
  ggplot2::geom_vline(aes(xintercept = prior_alpha[1]), alpha = 0.3) +
  ggplot2::labs(x = "alpha in cm",
       y = "density",
       title = "Prior distribution for alpha (intercept at mean weight)",
       subtitle = paste("alpha ~ normal(", as.character(prior_alpha[1]), ",",
        ↪  as.character(prior_alpha[2]), ")", sep = ''))

# Plot for beta prior distribution
p_beta <- ggplot2::ggplot() +
  xlim(0, 20)+
  ggplot2::geom_function(fun = dnorm, args = list(mean = prior_beta[1], sd =
 ↪  prior_beta[2]), colour = "dodgerblue2", size = 1) +
  ggplot2::labs(x = "beta in cm/kg",
       y = "density",
       title = "Prior distribution for beta (slope)",
       subtitle = paste("beta ~ half-normal(", as.character(prior_beta[1]), ",",
        ↪  as.character(prior_beta[2]), ")", sep = ''))

# Plot for sigma prior distribution
p_sigma <- ggplot2::ggplot() +
  xlim(0, 50) +
  ggplot2::geom_function(fun = extraDistr::dhcauchy, args = list(prior_sigma[2])) +
  ggplot2::labs(x = "sigma in cm",
       y = "density",
       title = "Prior distribution for sigma (noise)",
```

```r
       subtitle = paste("sigma ~ half-cauchy(", as.character(prior_sigma[1]), ",",
        ↪  as.character(prior_sigma[2]), ")", sep = ''))

initial_plots <- ggpubr::ggarrange(scatterplot, p_alpha, p_beta, p_sigma,
                           labels= c("a", "b", "c", "d"),
                           ncol = 2, nrow = 2)
initial_plots

# prior kernel dens
bayesplot::color_scheme_set("blue")
prior_kernel_dens <- bayesplot::ppc_dens_overlay(y = df$height,
                                                 yrep = rstan::extract(fit_reg_pripc,
                                                  ↪  pars =
                                                  ↪  "height_bar")$height_bar[1:40, ],
                                                  ↪
                                                 alpha = 0.5, size = .8) +
  xlim(-250, 500) +
  ggplot2::ylab("Density") +
  ggplot2::theme(axis.text=element_text(size=14),
        axis.title=element_text(size=16),
        legend.text=element_text(size=20))

prior_kernel_dens

#### Visualizing the posterior predictive check ####
# extract samples
height_bar <- rstan::extract(fit_reg_postpc, pars = "height_bar", permuted =
 ↪  TRUE)$height_bar
alpha_post <- rstan::extract(fit_reg_postpc, pars = "alpha", permuted = TRUE)$alpha
beta_post <- rstan::extract(fit_reg_postpc, pars = "beta", permuted = TRUE)$beta
sigma_post <- rstan::extract(fit_reg_postpc, pars = "sigma", permuted = TRUE)$sigma

# Calculate credible intervals for parameters
ci_alpha_eti <- bayestestR::ci(alpha_post, method = "ETI")
ci_alpha_hdi <- bayestestR::ci(alpha_post, method = "HDI")

ci_beta_eti <- bayestestR::ci(beta_post, method = "ETI")
ci_beta_hdi <- bayestestR::ci(beta_post, method = "HDI")

ci_sigma_eti <- bayestestR::ci(sigma_post, method = "ETI")
ci_sigma_hdi <- bayestestR::ci(sigma_post, method = "HDI")

# posterior kernel dens
bayesplot::color_scheme_set("blue")
posterior_kernel_dens <-
  bayesplot::ppc_dens_overlay(y = df$height,
                              yrep = rstan::extract(fit_reg_postpc,
                                                    pars =
                                                     ↪  "height_bar")$height_bar[1:40,
                                                     ↪  ],
                              alpha = 0.5, size = .8) +
  xlim(50, 250) +
  ggplot2::ylab("Density") +
  ggplot2::theme(axis.text=element_text(size=14),
        axis.title=element_text(size=16),
        legend.text=element_text(size=20)) +
  ggplot2::labs(title = "Posterior kernel density")

```

```r
posterior_kernel_dens

p_alpha_post_gg <- ggplot2::ggplot(as.data.frame(alpha_post), aes(alpha_post)) +
  ggplot2::geom_density(colour = "forestgreen", size = 1) +
  ggplot2::labs(title = "Posterior probability distribution of alpha",
       xlab = "alpha")

p_beta_post_gg <- ggplot2::ggplot(as.data.frame(beta_post), aes(beta_post)) +
  ggplot2::geom_density(colour = "dodgerblue2", size = 1) +
  ggplot2::labs(title = "Posterior probability distribution of beta",
       xlab = "beta")

p_sigma_post_gg <- ggplot2::ggplot(as.data.frame(sigma_post), aes(sigma_post)) +
  ggplot2::geom_density(size = 1) +
  ggplot2::labs(title = "Posterior probability distribution of sigma",
       xlab = "sigma")

ggpubr::ggarrange(p_alpha_post_gg, p_beta_post_gg, p_sigma_post_gg,
 ↪  posterior_kernel_dens,
                           labels= c("a", "b", "c", "d"),
                           ncol = 2, nrow = 2)
```

A.4 Commented *Stan* model for Rescorla-Wagner reinforcement learning

```
// Modelling reinforcement learning in a two-armed bandit task with a Rescorla-Wagner
 ↪  model
// Jakob Weickmann, 2021
// jakob.weickmann@posteo.de

// in cooperation with Damian Bednarz and Lei Zhang
// damian.bednarz@posteo.de, lei-zhang.net

data {
  int<lower=1> nSubjects; // This is the number of participants, in the case of prior
   ↪  predictive checks this encodes the number of simulations. There has to be at
   ↪  least 1 participant
  int<lower=1> nTrials; // This is the number of trials per participant, at least 1
   ↪  trial
  int<lower=1,upper=2> choice[nSubjects, nTrials]; // This is the choice that was made
   ↪  by a participant in a given trial, either 1 or 2
  real<lower=-1, upper=1> reward[nSubjects, nTrials]; // This variable encodes whether
   ↪  a reward was received in a given trial, 1 if a reward was received, and -1 if no
   ↪  reward was received
  int<lower=0, upper=1> analysis_step; // This is the boolean variable for the switch
   ↪  that allows us to use the same script with little variation depending on whether
   ↪  we are running the posterior (1) or the prior (0) predictive checks
}

transformed data {
  vector[2] initV;  // Declaring the variable initial values for V (the value attached
   ↪  to each choice)
  initV = rep_vector(0.0, 2); // V is a vector with two entries, one for each choice.
 ↪  V is initiated with the precursor variable `initV`, that is initialized with two
 ↪  zeros.
```

```
}

parameters {
  // group-level parameters
  real lr_mu_raw;  // the mean of the distribution from which we draw the learning rate (lr), assuming that the learning rates of the subjects are drawn from a common distribution
  real<lower=0> lr_sd_raw; // the standard deviation for the above distribution. Must be positive

  real tau_mu_raw; // same as above for tau, the 'temperature parameter'. The higher the value of tau, the higher the probability to choose the option with the higher value associated to it (especially for small value differences)
  real<lower=0> tau_sd_raw; // the standard deviation for the distribution, from which we draw tau

  // subject-level raw parameters
  vector[nSubjects] lr_raw; // declaring the subject-specific value for the individual learning rate
  vector[nSubjects] tau_raw; // declaring the subject-specific value for the 'temperature parameter' tau
}

transformed parameters {
  vector<lower=0,upper=1>[nSubjects] lr; // defining the constraints for the per subject learning rate parameter (lr)
  vector<lower=0,upper=50>[nSubjects] tau; // defining the constraints for the per subject temperature parameter (tau), 5 is chosen here as it appears to yield usable results in practice

  for (s in 1:nSubjects) {
    lr[s]  = Phi_approx( lr_mu_raw  + lr_sd_raw * lr_raw[s] ); // the function Phi_approx maps a distribution on the [0,1] parameter space. T
    tau[s] = Phi_approx( tau_mu_raw + tau_sd_raw * tau_raw[s] ) * 50; // the same is true for the temperature parameter tau. It is instead mapped to a [0,5] parameter space.
  }
}

model {
  lr_mu_raw  ~ normal(0,1);
  tau_mu_raw ~ normal(0,1);
  lr_sd_raw  ~ normal(0,2);
  tau_sd_raw ~ normal(0,2);

  lr_raw  ~ normal(0,1);
  tau_raw ~ normal(0,1);

  // Only active during the posterior predictive check and the only difference in the scripts function between prior predictive checks and posterior predictive checks.
  if (analysis_step == 1) {
    for (s in 1:nSubjects) { // loops over all subjects
      vector[2] v; // declare the variable value (v)
      real pe; // declare the variable prediction error (pe)
      v = initV; // set the value to the initial value (0,0) for that subject

      for (t in 1:nTrials) { // loops over all trials for that subject
```

```
61          choice[s,t] ~ categorical_logit( tau[s] * v ); // this is the likelihood
    ↪  function that links the data (choice and reward) with our priors. How likely is
    ↪  the data (choice) for a given parameter (tau, v)? The `categorical_logit` function
    ↪  applies the softmax function internally to convert a vector (tau[s] * v) to a
    ↪  simplex.
62          pe = reward[s,t] - v[choice[s,t]]; // updating the prediction error depending
    ↪  on the outcome of the trial and the expected reward
63          v[choice[s,t]] = v[choice[s,t]] + lr[s] * pe; // updating the value of each
    ↪  choice depending on the prediction error, the learning rate and the value
    ↪  previously attached to that choice
64          // the combination of the above statements is the Rescorla Wagner Model
             ↪  expressed in probability statements
65        }
66      }
67    }
68  }
69
70  generated quantities {
71    real<lower=0,upper=1> lr_mu; // declaring the mean of the learning rate
72    real<lower=0,upper=50> tau_mu; // declaring the mean of tau
73
74    int y_pred[nSubjects, nTrials]; // declaring the modelled choice (predicted)
75
76    lr_mu  = Phi_approx(lr_mu_raw);
77    tau_mu = Phi_approx(tau_mu_raw) * 50;
78    y_pred = rep_array(-999,nSubjects ,nTrials);
79
80    { // local block
81      for (s in 1:nSubjects) {
82        vector[2] v;
83        real pe;
84        v = initV;
85        for (t in 1:nTrials) {
86          y_pred[s,t] = categorical_logit_rng( tau[s] * v ); // draws one tau value from
    ↪  the tau distribution for this iteration of the MCMC
87          pe = reward[s,t] - v[choice[s,t]]; // prediction error according to the reward
    ↪  and choice from the data
88          v[choice[s,t]] = v[choice[s,t]] + lr[s] * pe; // choice update according to
    ↪  the data
89        }
90      }
91    }
92  }
```

A.5 Commented *R* script for Rescorla-Wagner reinforcement learning

```
1  # Simple Reinforcement Learning Model
2  # Running both the prior predictive and the posterior predictive checks using Stan
3
4  # Jakob Weickmann, 2021
5  # jakob.weickmann@posteo.de
6  # in cooperation with Damian Bednarz and Lei Zhang
```

```r
# damian.bednarz@posteo.de, lei-zhang.net

#### Construct Data ####
library(rstan)
library(ggplot2)
library(dplyr)
library(tidyr)
library(bayestestR)

# Set some options
# automatically saves a serialized version of the compiled model to the directory of the `.stan` file.
rstan_options(auto_write = TRUE)

# detects the number of cores available and runs that many chains in parallel. This allows maximum performance, but the user may want to manually set the number of cores if not all cores should be used.
options(mc.cores = parallel::detectCores())

# Loading the data

# The data are randomly generated. True parameters for the data generation are:
# lr  = rnorm(10, mean=0.6, sd=0.12); tau = rnorm(10, mean=1.5, sd=0.2)

# Read in the generated data from Github file
load(url('https://github.com/jweickm/ppc_tutorial/blob/main/data/rw.RData?raw=true'))
load("_data/rw_sim_data.RData")
sz <- dim(rw_mat)
nSubjects <- sz[1]
nTrials   <- sz[2]

# define the switch variable for the stan script
priorCheck <- 0 # 0: Prior predictive check
posteriorCheck <- 1 # 1: Posterior predictive check

# INITIALIZING STAN
# Set the stan file and define the MCMC parameters
modelFile <- "rescorla_wagner.stan"
nIter     <- 2000
nChains   <- 4
nWarmup   <- floor(nIter/2)
nThin     <- 1

dataList <- list(nSubjects = nSubjects,
                 nTrials = nTrials,
                 choice = rw_mat[, , 1],
                 reward = rw_mat[, , 2],
                 analysis_step = -1)
Seed = 1450154627

#### Running Stan & prior predictive checks ####
dataList$analysis_step <- 0

cat("Estimating", modelFile, "model... \n")
startTime = Sys.time(); print(startTime)
cat("Calling", nChains, "simulations in Stan... \n")

fit_rw_pri <- stan(modelFile,
```

```
                data    = dataList,
                chains  = nChains,
                iter    = nIter,
                warmup  = nWarmup,
                thin    = nThin,
                init    = "random",
                seed    = Seed
)

cat("Finishing", modelFile, "model simulation ... \n")
endTime = Sys.time(); print(endTime)
cat("It took",as.character.Date(endTime - startTime), "\n")

#### Running Stan & posterior predictive checks ####
dataList$analysis_step <- 1

cat("Estimating", modelFile, "model... \n")
startTime = Sys.time(); print(startTime)
cat("Calling", nChains, "simulations in Stan... \n")

fit_rw_post <- stan(modelFile,
                data    = dataList,
                chains  = nChains,
                iter    = nIter,
                warmup  = nWarmup,
                thin    = nThin,
                init    = "random",
                seed    = Seed
)

cat("Finishing", modelFile, "model simulation ... \n")
endTime = Sys.time(); print(endTime)
cat("It took",as.character.Date(endTime - startTime), "\n")

#### Model Summary and Diagnostics ####
print(fit_rw_pri)
print(fit_rw_post)

plot_dens_lr_pri  <- stan_plot(fit_rw_pri, pars=c('lr_mu','lr'), show_density=T,
 ↪  fill_color = 'gray85')
plot_dens_tau_pri <- stan_plot(fit_rw_pri, pars=c('tau_mu','tau'), show_density=T,
 ↪  fill_color = 'gray85')
plot_dens_lr_post  <- stan_plot(fit_rw_post, pars=c('lr_mu','lr'), show_density=T,
 ↪  fill_color = 'skyblue')
plot_dens_tau_post <- stan_plot(fit_rw_post, pars=c('tau_mu','tau'), show_density=T,
 ↪  fill_color = 'skyblue')

ggpubr::ggarrange(plot_dens_lr_post, plot_dens_tau_post,
                  labels= c("a", "b"),
                           ncol = 2, nrow = 1)

prior_comp_p <- ggpubr::ggarrange(norm_plot_dens_lr_pri, plot_dens_lr_pri,
 ↪  norm_plot_dens_tau_pri, plot_dens_tau_pri,
                  labels = c("a. Normal(0,2)", "b. Cauchy(0,3)", "c. Normal(0,2)", "d.
                   ↪  Cauchy(0,3)"),
                  font.label = list(size = 12),
                  hjust = -0.8,
                  ncol = 2, nrow = 2)
```

```
ggpubr::annotate_figure(prior_comp_p,
                        top = "Comparison of Normal(0,2) (left) and Cauchy(0,3) (right) priors for the SD")


lr_post <- rstan::extract(fit_rw_post, pars = "lr_mu", permuted = TRUE)$lr_mu
tau_post <- rstan::extract(fit_rw_post, pars = "tau_mu", permuted = TRUE)$tau_mu

bayestestR::hdi(lr_post)
bayestestR::eti(lr_post)
bayestestR::hdi(tau_post)
bayestestR::eti(tau_post)

#### Visualizing prior predictive checks ####

# extract samples from the generated stan model
y_pred_pri  <- rstan::extract(fit_rw_pri, pars ="y_pred", permuted = TRUE)$y_pred
y_pred_post <- rstan::extract(fit_rw_post, pars ="y_pred", permuted = TRUE)$y_pred

data_summary <- function(x) {
    m <- mean(x)
    ymin <- m-sd(x)
    ymax <- m+sd(x)
    return(c(y=m,ymin=ymin,ymax=ymax))
}

#### Violin plot of posterior means ####
pars_value <- get_posterior_mean(fit_rw_post, pars=c('lr','tau'))[,5]
pars_name  <- as.factor(c(rep('lr',10),rep('tau',10)))
df <- data.frame(pars_value=pars_value, pars_name=pars_name)

myconfig <- theme_bw(base_size = 20) +
    theme(panel.grid.major = element_blank(),
          panel.grid.minor = element_blank(),
          panel.background = element_blank() )

data_summary <- function(x) {
    m <- mean(x)
    ymin <- m-sd(x)
    ymax <- m+sd(x)
    return(c(y=m,ymin=ymin,ymax=ymax))
}

g1 <- ggplot(df, aes(x=pars_name, y=pars_value, color = pars_name, fill=pars_name))
g1 <- g1 + geom_violin(trim=TRUE, size=2)
g1 <- g1 + stat_summary(fun.data=data_summary, geom="pointrange", color="black", size=1.5)
g1 <- g1 + scale_fill_manual(values=c("#2179b5", "#c60256"))
g1 <- g1 + scale_color_manual(values=c("#2179b5", "#c60256"))
g1 <- g1 + myconfig + theme(legend.position="none")
g1 <- g1 + labs(x = '', y = 'parameter value') + ylim(0.3,2.2)
print(g1)



# make predictive checks
# adapted from
# Zhang, L. (2021). Lei-zhang/BayesCog_Wien [R]. https://github.com/lei-zhang/BayesCog_Wien (Original work published 2019)
```

```
170
171 ######################################################################
172 dL <- list(nSubjects=nSubjects,
173                 nTrials=nTrials,
174                 choice= rw_mat[,,1],
175                 reward= rw_mat[,,2])
176
177 ######################################################################
178 # overall mean of choosing the second (better) option
179 mean(dL$choice[,] == 2 )
180 mean(dL$reward[,] == 1 ) # overall mean of getting rewarded
181
182 # trial-by-trial sequence of choosing the 2nd option
183 y_mean = colMeans(dL$choice == 2)
184
185 y_pred_pri  <- rstan::extract(fit_rw_pri, pars ="y_pred", permuted = TRUE)$y_pred
186 y_pred_post <- rstan::extract(fit_rw_post, pars ="y_pred", permuted = TRUE)$y_pred
187
188 dim(y_pred_pri)
189 dim(y_pred_post)
190
191 # for prior predictive checks
192
193 y_pred_pri_mean_mcmc = apply(y_pred_pri==2, c(1,3), mean)
194 dim(y_pred_pri_mean_mcmc)  # [4000, 100]
195 y_pred_pri_mean = colMeans(y_pred_pri_mean_mcmc)
196 y_pred_pri_mean_hdi = apply(y_pred_pri_mean_mcmc, 2, HDIofMCMC)
197
198 #plot(1:100, colmeans(y_pred_pri_mean),type='b')
199
200 # ================================================================================
201 #### make plots ####
202 # ================================================================================
203 myconfig <- theme_bw(base_size = 20) +
204   theme(panel.grid.major = element_blank(),
205         panel.grid.minor = element_blank(),
206         panel.background = element_blank() )
207
208 df = data.frame(trial = 1:100,
209                 data  = y_mean,
210                 model = y_pred_pri_mean,
211                 hdi_l = y_pred_pri_mean_hdi[1,],
212                 hdi_h = y_pred_pri_mean_hdi[2,])
213
214 ## time course of the choice
215 g2 = ggplot(df, aes(trial,data))
216 g2 = g2 + geom_line(size = 1.5, aes(color= 'data')) + geom_point(size = 2, shape = 21,
    ↪  fill='skyblue3',color= 'skyblue3')
217 #g2 = g2 + geom_ribbon(aes(ymin=hdi_l, ymax=hdi_h), linetype=2, alpha=0.3, fill =
    ↪  'skyblue3')
218 g2 = g2 + geom_ribbon(aes(ymin=hdi_l, ymax=hdi_h, fill='model'), linetype=2,
    ↪  alpha=0.3)
219 g2 = g2 + myconfig + scale_fill_manual(name = '',  values=c("model" = "skyblue3")) +
220   scale_color_manual(name = '',  values=c("data" = "skyblue"))  +
221   labs(y = 'choosing correct (%)')
222 g2 = g2 + theme(axis.text   = element_text(size=22),
223                 axis.title  = element_text(size=25),
224                 legend.text = element_text(size=25))
```

```
g2
ggsave(plot = g2, "_plots/choice_seq_ppc.png", width = 8, height = 4, type = "cairo-png", units = "in")


## overall choice: true data (vertical line) + model prediction (hist)
tt_y = mean(df$data)
df2 = data.frame(model = rowMeans(y_pred_pri_mean_mcmc)) # overall mean, 4000 mcmc samples
g3 = ggplot(data=df2, aes(model)) + geom_histogram(binwidth =.005, alpha=.5, fill = 'skyblue3')
g3 = g3 + geom_vline(xintercept=tt_y, color = 'skyblue3',size=1.5)
g3 = g3 + labs(x = 'choosing correct (%)', y = 'frequency')
g3 = g3 + myconfig# + scale_x_continuous(breaks=c(tt_y), labels=c("event1"))
g3 = g3 + theme(axis.text   = element_text(size=22),
                axis.title  = element_text(size=25),
                legend.text = element_text(size=25))
g3
ggsave(plot = g3, "_plots/choice_mean_ppc.png", width = 6, height = 4, type = "cairo-png", units = "in")



# for posterior predictive checks

y_pred_post_mean_mcmc = apply(y_pred_post==2, c(1,3), mean)
dim(y_pred_post_mean_mcmc)  # [4000, 100]
y_pred_post_mean = colMeans(y_pred_post_mean_mcmc)
y_pred_post_mean_hdi = apply(y_pred_post_mean_mcmc, 2, HDIofMCMC)

#plot(1:100, colmeans(y_pred_post_mean),type='b')

# ================================================================================
#### make plots ####
# ================================================================================
myconfig <- theme_bw(base_size = 20) +
  theme(panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        panel.background = element_blank() )

df = data.frame(trial = 1:100,
                data  = y_mean,
                model = y_pred_post_mean,
                hdi_l = y_pred_post_mean_hdi[1,],
                hdi_h = y_pred_post_mean_hdi[2,])

## time course of the choice
g4 = ggplot(df, aes(trial,data))
g4 = g4 + geom_line(size = 1.5, aes(color= 'data')) + geom_point(size = 2, shape = 21, fill='skyblue3',color= 'skyblue3')
#g4 = g4 + geom_ribbon(aes(ymin=hdi_l, ymax=hdi_h), linetype=2, alpha=0.3, fill = 'skyblue3')
g4 = g4 + geom_ribbon(aes(ymin=hdi_l, ymax=hdi_h, fill='model'), linetype=2, alpha=0.3)
g4 = g4 + myconfig + scale_fill_manual(name = '',  values=c("model" = "skyblue3")) +
  scale_color_manual(name = '',  values=c("data" = "skyblue"))  +
  labs(y = 'choosing correct (%)')
g4 = g4 + theme(axis.text   = element_text(size=22),
```

```
                axis.title  = element_text(size=25),
                legend.text = element_text(size=25))
g4
ggsave(plot = g4, "_plots/choice_seq_ppc.png", width = 8, height = 4, type =
 ↪  "cairo-png", units = "in")


## overall choice: true data (vertical line) + model prediction (hist)
tt_y = mean(df$data)
df2 = data.frame(model = rowMeans(y_pred_post_mean_mcmc)) # overall mean, 4000 mcmc
 ↪  samples
g5 = ggplot(data=df2, aes(model)) + geom_histogram(binwidth =.005, alpha=.5, fill =
 ↪  'skyblue3')
g5 = g5 + geom_vline(xintercept=tt_y, color = 'skyblue3',size=1.5)
g5 = g5 + labs(x = 'choosing correct (%)', y = 'frequency')
g5 = g5 + myconfig# + scale_x_continuous(breaks=c(tt_y), labels=c("event1"))
g5 = g5 + theme(axis.text   = element_text(size=22),
                axis.title  = element_text(size=25),
                legend.text = element_text(size=25))
g5
ggsave(plot = g5, "_plots/choice_mean_ppc.png", width = 6, height = 4, type =
 ↪  "cairo-png", units = "in")
```

A.6 Data generation for the two-armed bandit task

```
# Simulate Rescorla-Wagner prediction error learning data for a two-armed bandit task

# Jakob Weickmann, 2021
# jakob.weickmann@posteo.de

# Based on generative code by Nathaniel Phillips
# https://rpubs.com/YaRrr/ReinforcementLearningSimulation

# lr  = rnorm(10, mean=0.6, sd=0.12); tau = rnorm(10, mean=1.5, sd=0.2)
nTrials <- 100
# These functions define the main learning and choice processes. The learning function
 ↪  rw.fun() is the Rescorla-Wagner prediction error learning rule. The choice
 ↪  function softmax.fun() is softmax.
# Rescorla-Wagner prediction error updating function
rw.fun <- function(exp.prior = c(1, 1),      # A vector of prior expectations
                   new.inf = c(NA, NA),        # A vector of new information (NAs
                    ↪  except for selected option)
                   lr = rnorm(1, mean=0.6, sd=0.12)) {   # Updating rate

  # Save new expectations as prior
  exp.new <- exp.prior

  # Determine which option was selected
  selection <- which(is.finite(new.inf))

  # Update expectation of selected option
  exp.new[selection] <- exp.prior[selection] + lr * (new.inf[selection] -
 ↪  exp.prior[selection])

```

```
  return(exp.new)
}

# Softmax selection function
softmax.fun <- function(exp.current = c(1, 1),
                        tau = rnorm(1, mean=1.5, sd=0.2)
) {

  output <- exp(exp.current * tau) / sum(exp(exp.current * tau))

  return(output)
}

## The main simulation function is rl.sim.fun(). The function returns a dataframe
 ↪  containing the main results of the agent (and plots the agent's cumulative
 ↪  earnings when plot = TRUE)

# Create main simulation function
rl.sim.fun <- function(n.trials = nTrials,     # Trials in game
                       option.probability = c(0.25, 0.75), # Probability for each
                        ↪  option to produce a reward
                       n.options = 2,
                       prior.exp.start = rep(0, 2),
                       tau = rnorm(1, mean=1.5, sd=0.2),
                       lr = rnorm(1, mean=0.6, sd=0.12)) {

  # Get some game parameters from inputs
  n.options <- length(option.probability)

  # Create outcome matrix giving the outcome on each trial for each option
  outcome.mtx <- matrix(NA, nrow = n.trials, ncol = n.options)

  reward_seq <- rbinom(n.trials, 1, option.probability[2]) + 1

  for(option.i in 1:n.options) {
    outcome.mtx[,option.i] <- reward_seq == option.i
  }

  outcome.mtx[outcome.mtx == FALSE] <- -1
  outcome.mtx[outcome.mtx == TRUE] <- 1

  # Create exp.prior and exp.new matrices
  #  These will hold the agent's expectation (either prior or new)
  #   of each option on each trial

  exp.prior.mtx <- matrix(NA, nrow = n.trials, ncol = n.options)
  exp.prior.mtx[1,] <- prior.exp.start
  exp.new.mtx <- matrix(NA, nrow = n.trials, ncol = n.options)

  # Now create some matrices to store values

  selection.v <- rep(NA, n.trials)      # Actual selections
  outcome.v <- rep(NA, n.trials)        # Actual outcomes
  selprob.mtx <- matrix(NA,             # Selection probabilities
                        nrow = n.trials,
                        ncol = n.options)

  # RUN SIMULATION!
```

```r

  for(trial.i in 1:n.trials) {

    # STEP 0: Get prior expectations for current trial
    exp.prior.i <- exp.prior.mtx[trial.i,]

    # STEP 1: SELECT AN OPTION
    # Selection probabilities
    selprob.i <- softmax.fun(exp.current = exp.prior.i,
                             tau = tau)

    # Select an option
    selection.i <- sample(1:n.options, size = 1, prob = selprob.i)

    # Get outcome from selected option
    outcome.i <- outcome.mtx[trial.i, selection.i]

    # STEP 3: CREATE NEW EXPECTANCIES
    # Create a new.inf vector with NAs except for outcome of selected option
    new.inf <- rep(NA, n.options)
    new.inf[selection.i] <- outcome.i

    # Get new expectancies
    new.exp.i <- rw.fun(exp.prior = exp.prior.i,
                        new.inf = new.inf,
                        lr = lr)

    # assign new expectatations to exp.new.mtx[trial.i,]
    # and prior.expecation.mtx[trial.i + 1,]
    exp.new.mtx[trial.i,] <- new.exp.i

    if(trial.i < n.trials) {
      exp.prior.mtx[trial.i + 1,] <- new.exp.i
    }

    # Save some values
    selprob.mtx[trial.i,] <- selprob.i  # Selection probabilities
    selection.v[trial.i] <- selection.i # Actual selection
    outcome.v[trial.i] <- outcome.i     # Actual outcome
  }

  # Put main results in a single dataframe called sim.result.df
  sim.result.df <- data.frame("selection" = selection.v,
                              "outcome" = outcome.v,
                              "outcome.cum" = cumsum(outcome.v),
                              stringsAsFactors = FALSE)

  # Now return the main simulation dataframe
  return(sim.result.df)
}


# Simulate the data for 10 subjects
set.seed(100) # For replicability
nSubjects <- 10
rw_mat <- array(NA, c(nSubjects, nTrials, 3))
for (subject in 1:nSubjects){
  trial_mat <- rl.sim.fun()
```

```
  rw_mat[subject, , 1] <- trial_mat[, 1]
  rw_mat[subject, , 2] <- trial_mat[, 2]
  rw_mat[subject, , 3] <- trial_mat[, 3]
}

save(rw_mat, file = "_data/rw_sim_data.RData")
```

A.7 Influence of the prior and the sample size on the posterior

```
library(ggplot2)
library(dplyr)
library(tidyr)
library(latex2exp)
library(ggpubr)

ggplot2::theme_set(theme_classic2())

# this is a grid approximation with 1000 parts
theta <- seq(.0005, .9995, .001)
res <- 1/length(theta)

weak_prior <- dbeta(x = theta, shape1 = 5, shape2 = 5) * res
strong_prior <- dbeta(x = theta, shape1 = 25, shape2 = 25) * res
flat_prior <- dunif(x = theta, min = 0, max = 1) * res
weak_prior_b <- dbeta(x = theta, shape1 = 2, shape2 = 8) * res
# ----------------------------------------
# ----------------------------------------
# model 100 throws with a flat, 2 weak and a strong prior. posterior update for each throw
observations <- rbinom(80, 1, 0.6)
head_tails <- c("T", "H")
all_throws <- head_tails[observations+1]
prior_1 <- flat_prior
prior_2 <- weak_prior
prior_2b <- weak_prior_b
prior_3 <- strong_prior
par(mar = c(2.5, 2, 2.5, 0.5))
layout(matrix(c(1,2,3,4,17, 5,6,7,8,17, 9,10,11,12,17, 13,14,15,16,17), ncol=4), heights=c(2,2,2,2,1), widths = c(.8, 1, 1, 1))

plot.new()
text(0.5, 0.5, paste("a) Flat prior", "~ Uniform(0,1)", sep="\n"), cex=1.5, font=1)
plot.new()
text(0.5, 0.5, paste("b) Weak prior A", "~ Beta(5,5)", sep="\n"), cex=1.5, font=1)
plot.new()
text(0.5, 0.5, paste("c) Weak prior B", "~ Beta(2,8)", sep="\n"), cex=1.5, font=1)
plot.new()
text(0.5, 0.5, paste("d) Strong prior", "~ Beta(25,25)", sep="\n"), cex=1.5, font=1)

for (i in 1:length(observations)){
  likelihood <- dbinom(x = observations[i], size = 1, prob = theta)
  # flat prior
```

```r
  posterior_1 <- likelihood * prior_1
  posterior_1 <- posterior_1 / sum(posterior_1)
  # weak prior
  posterior_2 <- likelihood * prior_2
  posterior_2 <- posterior_2 / sum(posterior_2)
  # weak prior 2
  posterior_2b <- likelihood * prior_2b
  posterior_2b <- posterior_2b / sum(posterior_2b)
  # strong prior
  posterior_3 <- likelihood * prior_3
  posterior_3 <- posterior_3 / sum(posterior_3)

  # likelihood <- likelihood / sum(likelihood)
  throw <- head_tails[observations[i] + 1]

  if (is.element(i, c(5, 20, 80))){
    # flat prior
    plot(x = theta, y = posterior_1,
         type = "l",
         ylim=c(0, 0.01),
         yaxt='n',
         ylab="",
         xlab = "",
         panel.first = c(abline(h=seq(0, 0.01, 0.002), col="gray94"), abline(v=seq(0,
          ↪  1, 0.1), col="gray94")))
    lines(x = theta, y = flat_prior, type = "l", lty=3)
    mtext(paste("N =", i), side=3, line=.5, adj=0.5)
    text(0.15, 0.0085, paste("H = ", sum(observations[1:i])))

    if (i == 5){
      title(ylab = "Plausibility", line = .5)
    }

    # weak prior 1
    plot(x = theta, y = posterior_2,
         type = "l",
         ylim=c(0, 0.01),
         yaxt='n',
         ylab="",
         xlab = "",
         panel.first = c(abline(h=seq(0, 0.01, 0.002), col="gray94"), abline(v=seq(0,
          ↪  1, 0.1), col="gray94")))
    lines(x = theta, y = weak_prior, type = "l", lty=3)
    mtext(paste("N =", i), side=3, line=.5, adj=0.5)
    text(0.15, 0.0085, paste("H = ", sum(observations[1:i])))

    if (i == 5){
      title(ylab = "Plausibility", line =.5)
    }

    # weak prior 2
    plot(x = theta, y = posterior_2b,
         type = "l",
         ylim=c(0, 0.01),
         yaxt='n',
         ylab="",
         xlab = "",
         panel.first = c(abline(h=seq(0, 0.01, 0.002), col="gray94"), abline(v=seq(0,
          ↪  1, 0.1), col="gray94")))
```

```
    lines(x = theta, y = weak_prior_b, type = "l", lty=3)
    mtext(paste("N =", i), side=3, line=.5, adj=0.5)
    text(0.15, 0.0085, paste("H = ", sum(observations[1:i])))

    if (i == 5){
      title(ylab = "Plausibility", line =.5)
    }

    # strong prior
    plot(x = theta, y = posterior_3,
         type = "l",
         ylim=c(0, 0.01),
         yaxt='n',
         ylab="",
         xlab = "",
         panel.first = c(abline(h=seq(0, 0.01, 0.002), col="gray94"), abline(v=seq(0,
          ↪  1, 0.1), col="gray94")))
    lines(x = theta, y = strong_prior, type = "l", lty=3)
    mtext(paste("N =", i), side=3, line=.5, adj=0.5)
    text(0.15, 0.0085, paste("H = ", sum(observations[1:i])))
    title(xlab = TeX("$\\theta$"), line=1.5)
  }

    if (i == 5){
      title(ylab = "Plausibility", line =.5)
    }

  # lines(x = theta, y = likelihood, type = "l", lty=1, col = 'red')
  prior_1 <- posterior_1
  prior_2 <- posterior_2
  prior_2b <- posterior_2b
  prior_3 <- posterior_3
}
plot(0,0, type='l', bty='n', xaxt='n', yaxt='n')
legend("topright", legend=c("Prior", "Posterior"), lty=c(3,1))
```