Bayesian Estimation of Decision Models

Ferdinand M. Vieider

2024-01-05

Table of contents

Preface

In these pages, I try and provide an introduction on how to obtain Bayesian estimations of decision models using R and Stan. The text posted at this point is preliminary, and comes with absolutely no guarantees. If you spot mistakes or have questions, do not hesitate to get in touch with me.

How to use these notes

The notes are not meant to be complete or exhaustive in any way. They also come with no guarantees. In particular, Stan can be a little tricky to install and use, and may not work on some platforms in the exact way which I present here. I am afraid that I cannot help with such issues—your best bet will be to either try an alternative installation (e.g., using Rstan instead of CommandStanR), or to seek help on the Stan forum. That being said, I am interested in any mistakes or shortcomings pertaining to the text and demonstration, including but not limited to ambiguities in the exposition.

You should of course feel free to use these notes as you see fit. That being said, I recommend taking a close look at the conjugate Bayesian analysis in chapter I before moving on to the estimation parts. Unless you have already a sound understanding of this material—try testing yourself by writing down the equations for a conjugate normal estimate of a Bayesian model with endogenous mean and variance—you will miss out in terms of interpretation and understanding if you start directly from the estimations shown in chapters II and III. Questions about the relevance of the prior keep coming up, and having a sound understanding of these issues from the conjugate part will help you in dealing with these issues in a principled way later on, when models become much more complex.

Since we learn best when we make mistakes and then confront these mistakes with reality, I would also highly recommend playing around with the code, and where possible, applying it to a problem that interests you, or at least changing the parameters or applying it to different data and seeiing what that yields. While a simple reading may give you an idea where to find the information if you need it, the passive exposure it provides is typically not sufficient to master the materials. This is why I provide all the code and data with the materials (this is currently still in progress, but most of the data are publicly available at this point). Feel free to play around with them, and do not hesitate to apply the insights contained here to other data and to modify the code in a way to suit your own needs!

Citation

If you use these notes for your work, or find the text useful in developing new code, please cite these notes as follows:

Vieider, Ferdinand M. (2024). Bayesian Estimation of Decision-Making Models. URL: https://fvieider.quarto.pub/bstats/

1 Introduction

Why Bayesian statistics? The relative merits of Bayesian and frequentist statistics have been debated for a long time, without any clear outcome. In these notes, I will thus focus on practical advantages of conducting Bayesian data analysis. When it comes to the practical advantages of Bayesian analysis, four elements stand out: 1) ease of interpretation; 2) the possibility to accept the null hypothesis; 3) estimation and manipulation of complex models; and 4) systematic accumulation of knowledge. I will provide a brief discussion of each in in this introduction. Together, I believe that these practical advantages constitute a powerful argument for the use of Bayesian statistics as the researcher's standard tool for data analysis.

Given these advantages, one may ask why Bayesian statistics are not used more commonly. The answer appears to be: path dependency. Bayesian models—especially the hierarchical models with multiple levels where Bayesian statistics has its main strengths—require intense computations that in the past would have been very costly. This meant that Bayesian analysis has not been commonly taught as a "standard tool" of analysis. Advances in computing power, however, have made general Bayesian approaches a viable alternative to classical statistics. The software package Stan provides a fast and flexible platform that allows us to implement such models.

1.1 Probabilistic interpretation of parameters

In classical statistics, model parameters are treated as certain quantities that are unknown to the statistician and need to be discovered. The data, on the other hand, are conceived of as a random sample from a much larger—and presumably infinitely large—population. This allows to build up a fiction that, as long as we can sample enough data, estimators of the fixed but unknown parameters will be unbiased. A p-value of 0.05 is then interpreted following the fiction that, in repeated data samples from the same population, only 1 out of 20 samples is expected to show to such an extreme value by chance. That is, if one were to sample from the population 20 times, on average the data should show such an extreme value 1 time *if the null hypothesis holds true*.

Bayesian analysis turns this interpretation on its head. The data at hand are considered *given* (i.e., they are a *datum*, a given), and may well be unique. The parameters, on the other hand, are both unknown to the statistician and inherently uncertain. This allows us to directly reason in terms of uncertainty surrounding the parameter estimates. In Bayesian statistics, a

statement like *there is a 95% chance that the parameter falls below 1* is actually correct, while such simple statements cannot be made in standard statistics. Intuitively, we may thus all be Bayesians.

1.2 Accepting the null

Hypothesis testing in classical statistics invariably takes the form of establishing and testing a *point hypothesis* (e.g., the effect is different from 0). This may be fine in most circumstances, even though it creates issues with large samples, since any point hypothesis will be rejected by definition as long as the sample is large enough. In some circumstances, however, it may be necessary to be able to *accept* the null hypothesis. For instance, when we test two models making different predictions about a parameter (the parameter is equal to vs. the parameter is smaller than a certain value) against each other, taking the point prediction as the null hypothesis will invariably bias the analysis against the model making that prediction.

Bayesian analysis can be coupled with the establishment of so-called *regions of practical equivalence* around a null hypothesis (I will simply refer to these regions as *acceptance regions*). This means that instead of establishing a point null hypothesis (e.g., the parameter is 1), we establish a region of equivalence to this value. For instance, we could say that we reject the null hypothesis if at least 95% of the posterior probability mass falls outside the region $[0.97, 1.03]$. We accept the null hypothesis if at least 95% of the probability density falls within that region (we could of course also pick another, equally arbitrary probability value, such as 97.5%). As we will see, such a setup can have many practical advantages, including when deciding between nested model, or discriminating between several sub-cases of one more general model.

This aspect of Bayesian statistics also has substantive implications for best practices in experimental studies, and for potential pre-registrations of experimental designs. Notably, if one establishes an acceptance region for one's hypotheses, sticking to a fixed number of observations in data collection is no longer meaningful. That is because, as long as the data do not give a clear verdict one way or another, the optimal course of action will always be to continue data collection until such a clear verdict is reached. This is clearly legitimate since the method is no longer inherently biased against the null hypothesis—the null can now be accepted as well as rejected. All that being said, transparent procedures would then require the pre-registration of the acceptance regions, since there may otherwise be a risk of ex post adjustment of such regions. Nevertheless, this risk is mitigated in most practical cases by scale constraints to these regions emerging from the scale of the data—an issue that is however beyond the scope of these technical notes.

1.3 Estimating complex models

Perhaps the greatest advantage of Bayesian analysis is the ease with which complex models with thousands of parameters can be estimated. In principle, we can even estimate models with more (nominal) parameters than data points. The parameters can furthermore be manipulated and combined post estimation, since the simulated posterior estimate allows to carry along all the information about the confidence we have in the inferences even while multiplying, dividing, subtracting, and otherwise manipulating parameters. The greatest advantage of the ease in handling complexity, however, is the possibility to estimate hierarchical or multilevel models.

Many types of data present natural structure, which is often ignored in standard analysis. Students participating in experiments are often recruited into sessions, and treatments are allocated at the session level. In field settings, we usually follow stratified sampling procedures, e.g. we select regions, from which we select provinces, from which we select households, and this is often done to maximize representativeness along dimensions relevant for the study. In meta-analysis, study-level effects are collected together with their standard errors. Such studies may stem from a handful of research groups, implement different measurement methods, etc. Neglecting such structure can yield severe biases in the estimation results. It also throws away valuable information, e.g. on the correlation structure of observations within any given group. Hierarchical models try to explicitly capture these structures in the statistical model, painting a much richer picture of the results.

One advantage of hierarchical modelling is that even fairly complex models with multiple parameters can be estimated starting from sparse data. The reason for this is that aggregate parameter estimates can serve as endogenous priors for lower-level estimates. Also, by pooling information across similar units of analysis Bayesian models can extract the most information from thin data by exploiting similarities, thus yielding an optimal compromise between aggregate and individual-level estimation that is endogenously determined by the relative confidence we have in these quantities. We will thus spend some time on building up the foundations of such models, and illustrating their power based on a number of examples.

1.4 Accumulation of knowledge

Finally, Bayesian statistics provide a principled way of accumulating knowledge across studies. Indeed, it seems natural to aggregate past results into probability distributions that summarize them; and to then use these distributions as priors for future estimations. This also means that Bayesian analysis is extremely suitable for contexts in which we may want to implement sequential learning, e.g. taking one data point at the time and updating our parameter estimates with it.

1.5 Further resources

This is not a general purpose tutorial of Bayesian statistics. An excellent introduction to Bayesian statistics light on technicalities is provided by McElreath (2016). A complete formal treatment can be found in Gelman et al. (2014) .

2 Bayesian Foundations

This chapter contains a discussion of the foundations of Bayesian statistics. I focus on simple conjugate analysis of the most common distributional families. This serves to build theoretical underpinnings that will prove invaluable in the interpretation of more complex models in later chapters. It will furthermore help in developing a thorough understanding of the relative role of the prior and the data in determining the inferences we draw. This is an extremely important exercise, since the effect of priors is something that often confuses non-Bayesians upon first impact with Bayesian techniques. It is, however, important to keep in mind that Bayesian analysis is much more about taking uncertainty seriously than it is about priors. Exploring these issues using simulated data and sequential updating will allow us to paint a precise picture of how exactly the influence of the prior vanes as subsequent data points are added. Beyond the examples provided, it is always a good idea to look at the underlying code, and to tweak any parameters of interest to see the effect that may have on the posterior inferences!

2.1 Bayes' equation

Bayes' equation is likely the single most important equation in all of statistics, if not life and nature in general. The fundamental idea is simple. Say we want to make an inference on the likelihood of a hypothesis, designated by h. We have some data that might be informative on this issue, call them d. What we want to find out is the probability that the hypothesis is true or false, given the data (and yes, the data are truly *given* in this context). This is a hard problem to answer directly. Instead, we may want to think about an easier problem, such as what is the likelihood of the data, given the hypothesis, as done in classical statistics. This is an easier problem because in an idealized experiment we can use the hypothesis to generate a large number of simulated datasets and see how often such a newly drawn dataset will resemble the one at hand. We can then use this to answer our original query in the following way:

$$p(h \mid d) = \frac{p(d \mid h)\, p(h)}{p(d)}$$

where $p(d)$ is the total probability of the data, i.e. $p(d) = \sum_i p(d \mid h_i)\, p(h_i)$, or $\int p(d \mid h)\, p(h)\, dh$ for a continuous hypothesis space. This is called a *marginal* distribution in Bayesian statistics,

and summing over all possible cases to eliminate the conditionality statements is thus called *marginalization*.

The trouble is that the denominator is difficult to calculate precisely in most situations, and may not allow a closed form solution at all. We thus often see the equation written in the following way:

$$p(h \mid d) \propto p(d \mid h)\, p(h),$$

where $\propto$ indicates proportionality and is read as *is proportional to*. That is, we can usually simulate the posterior distribution by the above expression, without calculating $p(d)$ precisely. This is a legitimate strategy, inasmuch as the marginal density $p(d)$ does not contain the hypothesis, so that it can be considered a constant. This also means, however, that no analytic solutions will generally be possible—we will have to rely on simulations only. This is indeed why Bayesian analysis has been a minority pursuit for a long time. However, with the ever-increasing availability of fast computing and the consequent development of dedicated Bayesian statistics packages, Bayesian statistics is quickly emerging as a viable alternative to the classical, frequentist type.

Luckily, there is a way of circumventing the thorny issue of calculating the normalization constant all together: we can use conjugate priors. A conjugate prior is a prior such that the posterior distribution obtained by combining likelihood and prior will belong to the same distributional family as the prior itself. This allows us to derive analytic solutions for the posterior, and is thus extremely useful in illustrating the workings underlying Bayesian analysis. This first chapter is dedicated to developing the underlying formalism, and to showing a number of practical examples. Two distributions will prove particularly useful—distributions from the Bernoulli-Binomial-Beta family, and distributions from the Gaussian family. We will start from the former.

2.2 Beliefs about proportions

We start from the standard example of a series of data points $x = \{x_1, ..., x_n\}$, which can take the value of 1 (success) or 0 (failure). A fitting example may be that we want to estimate the proportion of trustworthy individuals in a population, and we want to obtain a probability of a random individual from the population being trustworthy. Here and in sequence, I will assume the data points to be *exchangeable* (equivalent to an i.i.d. assumption), unless stated otherwise.

Let us assume that you observe a single realization from the data vector, call it x_i. We want to find the probability that $x_i = 1$, call it $p(x_i = 1 \mid \theta) = \theta$ (the notation emphasizes that this designates the probability of the data, *conditional* on the parameter; the same letter is used for the variable and its concrete value in this case, since no ambiguity arises from this). Since

$0 \leq \theta \leq 1$, we know that $p(x_i = 0 \,|\, \theta) = 1 - \theta$. The probability distribution will thus take the form of a Bernoulli function:

$$p(x_i \,|\, \theta) = \theta^{x_i} \, (1 - \theta)^{1 - x_i},$$

with a mean of θ and a variance of $\theta \, (1 - \theta)$. If the observations are exchangeable as assumed here, one can see directly that the probability of the overall data takes the following form

$$p(x \,|\, \theta) = \prod_{i=1}^{N} p(x_i \,|\, \theta)$$

A frequentist approach to the data would consist in maximizing the likelihood of the function above, although for computational reasons in practice it is usual to maximize the logarithm of the likelihood, which has the following form:

$$\begin{aligned} ln \;\; p(x \,|\, \theta) &= \sum_{i=1}^{N} ln \;\; p(x_i \,|\, \theta) \\ &= \sum_{i=1}^{N} [\, x_i \, ln(\theta) + (1 - x_i) \, ln(1 - \theta) \,]. \end{aligned}$$

Taking the derivative with respect to θ and setting it to zero, we obtain the following maximum likelihood estimate:

$$\theta_{ML} = \frac{1}{N} \sum_{i=1}^{N} x_i,$$

which indicates that the maximum-likelihood estimate of the probability is simply equal to the sample mean of the observations.

Although this approach will work well with large samples, it can lead to paradoxical conclusions based on small samples. Suppose that we flip a coin 4 times, and that all of the coins come up heads, encoded as 1. The maximum likelihood estimate then predicts that all future coin flips should come up heads. This seems an extreme prediction, illustrating the overfitting maximum likelihood techniques are prone to when using small samples. Before you protest that such a small sample is absurd, stop and think about the number of observations we often have when trying to assess the preference parameters of single individuals, where the models we try to fit are typically much more complex than in the example above.

We can also get at the solution described above more directly. Let m designate the number of successes out of N trials. We can then write the likelihood of this number of successes directly according to the Binomial model:

$$
\begin{aligned}
p(m \mid \theta) &= Bin(m \mid N, \theta) \\
&= \binom{N}{m} \theta^m \, (1-\theta)^{N-m}.
\end{aligned}
$$

In classical statistics, we would simply calculate this statistic from the sample, and possibly determine whether the conditions, such as the size of the sample, are such as to guarantee an unbiased estimate (though this is often neglected in practice). In the Bayesian setup we are interested in, however, the parameter θ is an uncertain quantity, while the data are considered given. We could then interpret the θ estimate obtained based on the above likelihood as *a random draw from a distribution of possible estimates*. As hinted at above, this will be especially advantageous when we are working with small datasets, potentially just one observation at the time.

We can thus specify a prior distribution that contains all kinds of knowledge we may have about the distribution of the parameter θ, simply written as $p(\theta)$. While in theory we could use any functional form, in practice it is often convenient to work with so-called *conjugate distributions*, i.e. functional forms which combined with the likelihood will yield a posterior that exhibits the same functional form as the prior. To do this, we can rewrite the likelihood above purely as a function of θ (we can do this because the remainder of the equation purely serves to normalize the density to 1, which we can also do ex post):

$$
p(x \mid \theta) \propto \theta^m \, (1-\theta)^{N-m}.
$$

We now want a prior density that has the same form, which so happens to be the Beta distribution:

$$
p(\theta) \propto \theta^{\alpha-1}(1-\theta)^{\beta-1},
$$

where I have again omitted the normalization constant since it is not important for our purposes.

To obtain the posterior distribution, we will need to multiply the likelihood with the prior. The posterior we obtain will again follow the same parametric form. Combining the likelihood and prior above, we can easily get the posterior distribution:

$$
\begin{aligned}
p(\theta \mid x) &\propto p(x \mid \theta) \, p(\theta) \\
&\propto \theta^m \, (1-\theta)^{N-m} \, \theta^{\alpha-1} \, (1-\theta)^{\beta-1} \\
&\propto Beta(\theta \mid \alpha + m \, , \, \beta + N - m).
\end{aligned}
$$

From this, we may obtain the posterior mean and variance as follows:

$$
\begin{aligned}
E(\theta \,|\, x) &= \frac{\alpha + m}{\alpha + \beta + N} \\
var(\theta|x) &= \frac{(\alpha + m)(\beta + N - m)}{(\alpha + \beta + N)^2(\alpha + \beta + N + 1)} \\
&= \frac{E(\theta|x)\,[1 - E(\theta \,|\, x)]}{\alpha + \beta + N + 1}.
\end{aligned}
$$

These equations serve to illustrate some fundamental properties of Bayesian analysis. For instance, as m and N become large for fixed parameters of the prior, the posterior expectation will converge to the sampling expectation, $\lim_{N\to\infty} E(\theta \,|\, x, \alpha, \beta) = \frac{m}{N}$. Equivalently, the variance will approach zero as N gets large, so that $var(\theta \,|\, x) \approx \frac{1}{N}\frac{m}{N}\,(1 - \frac{m}{N})$. That is, in the limit the prior has no influence over the posterior distribution. We can also easily see how new observations of successes or failures simply go to augment the parameters of the Beta distribution. This is especially useful in sequential updating, which I discuss next.

2.2.1 Sequential updating

In many applications, it will be interesting to update the prior sequentially as new information comes along. The posterior distribution derived from updating the prior with a new observation will then itself become the prior for the subsequent observation. The updating equations of the two parameters will simply take the following form:

$$
\begin{aligned}
\alpha_{i+1} &= \alpha_i + x_i \\
\beta_{i+1} &= \beta_i + (1 - x_i),
\end{aligned}
$$

where α_i and β_i are the parameters of the prior for the observation in period/round i, and the updated parameters indexed by $i + 1$ are the posterior parameters from the perspective of period i, as well as the *prior* parameters from the perspective of period $i + 1$, i.e. for the subsequent update with x_{i+1}. (The result obtains trivially by multiplying a Bernoulli likelihood with a Beta prior).

We can use this function to sequentially update our Beta prior. It will be helpful to illustrate this with a dataset of binary choices. Let us assume that we present a subject with choices between different sure amounts c and a lottery paying y with probability 0.5, or else 0. We know that the sure amounts c are equally spaced between y and 0, and that there is an equal number of sure amounts below and above the expected value py. The simplest way of discovering the preferences of the decision maker will then be to count the number of choices for the sure amount, indicated by $x_i = 1$.

```r
obs <- rbernoulli(n=50,p=0.71)
x <- append(obs,0,0)

alpha <- 2
beta <- 2

for(i in 2:51){ alpha[i] <- alpha[i-1] + x[i]
                beta[i] <- beta[i-1] + (1 - x[i]) }

ggplot(data = data.frame(x = c(0, 1)), aes(x=x)) +
  stat_function(fun = dbeta, n = 101, args = list(shape1 = alpha[1], shape2 = beta[1]), linew
  stat_function(fun = dbeta, n = 101, args = list(shape1 = alpha[2], shape2 = beta[2]), aes(
  stat_function(fun = dbeta, n = 101, args = list(shape1 = alpha[11], shape2 = beta[11]), aes
  stat_function(fun = dbeta, n = 101, args = list(shape1 = alpha[51], shape2 = beta[51]), aes
  geom_vline(xintercept=mean(obs), aes(colour="red"), linetype="dashed") + ylab("") + scale_y
  xlab("proportion of safe choices") +
  scale_colour_manual("Legend", values = c("black","darkgoldenrod", "chocolate4", "darkolive
  theme(legend.position = c(0.2, 0.75))
```

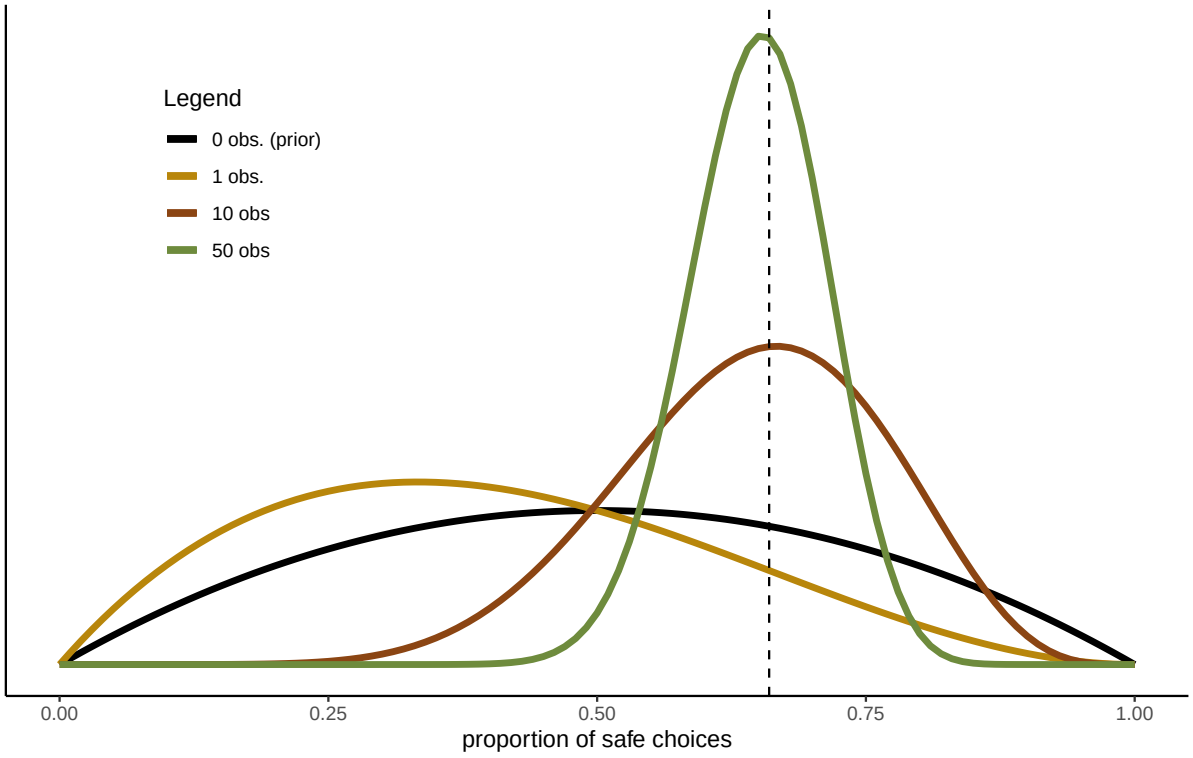


Figure 2.1: Sequentional Updating of Beta distribution.

Figure 2.1 shows the effect of sequentially updating a Beta prior with subsequent Bernoulli trials, simulated with a probability of 0.71 (and with a true sample average of 0.66 in the simulated data). The prior distribution $\mathcal{B}e(2,2)$ is fairly *diffuse*, assigning nonzero probability mass to the whole probability space. It is also *noninformative*, being centered on an ignorance value of 0.5. That is, we do not make any strong assumptions a priori, centering the distribution on expected value maximization, and only making it mildly informative, in the sense that we slightly discount extreme values towards 0 or 1. Clearly, the initial updating can go in the wrong direction, depending on the luck of the draw, as is indeed the case in the example in the figure. Nevertheless, after 10 draws we are closing in on the true mean, and after 50 draws we identify the true mean with a fair degree of confidence. In this case, the result indicates a pronounced degree of risk aversion.

Of course, we may choose to use stronger priors, for instance because we know from many previous observations that subjects in similar tasks tend to be risk averse. Such a strategy could, for instance, be useful if we want to test risk aversion in several small groups drawn from the same population, and we want to avoid over-testing (we will discuss a more principled and better way of doing this in chapter III in the context of hierarchical analysis). For illustration purposes, however, let us assume that we expect the decision maker to be risk *seeking*, and that we are quite confident in this prior expectation to boot. Let us illustrate the effects of such an assumption with a Beta prior given by $\mathcal{B}e(2,11)$, which indicates an expected outcome of $\frac{2}{2+11} \approx 0.15$, and a fair degree of confidence in this estimate (the confidence in an estimate can be approximately indicated by the concentration of a Beta distribution, given by $\alpha+\beta = 13$).

```
set.seed(345678)
obs <- rbernoulli(n=50,p=0.71)
x <- append(obs,0,0)

alpha <- 2
beta <- 11

for(i in 2:51){ alpha[i] <- alpha[i-1] + x[i]
                beta[i] <- beta[i-1] + (1 - x[i]) }

ggplot(data = data.frame(x = c(0, 1)), aes(x=x)) + stat_function(fun = dbeta, n = 101, args =
  stat_function(fun = dbeta, n = 101, args = list(shape1 = alpha[2], shape2 = beta[2]), aes(
  stat_function(fun = dbeta, n = 101, args = list(shape1 = alpha[11], shape2 = beta[11]), aes
  stat_function(fun = dbeta, n = 101, args = list(shape1 = alpha[51], shape2 = beta[51]), aes
  geom_vline(xintercept=mean(obs), aes(colour="red"), linetype="dashed") +
  ylab("") + scale_y_continuous(breaks = NULL) +
  scale_colour_manual("Legend", values = c("black","darkgoldenrod", "chocolate4", "darkolive
  theme(legend.position = c(0.8, 0.6))
```

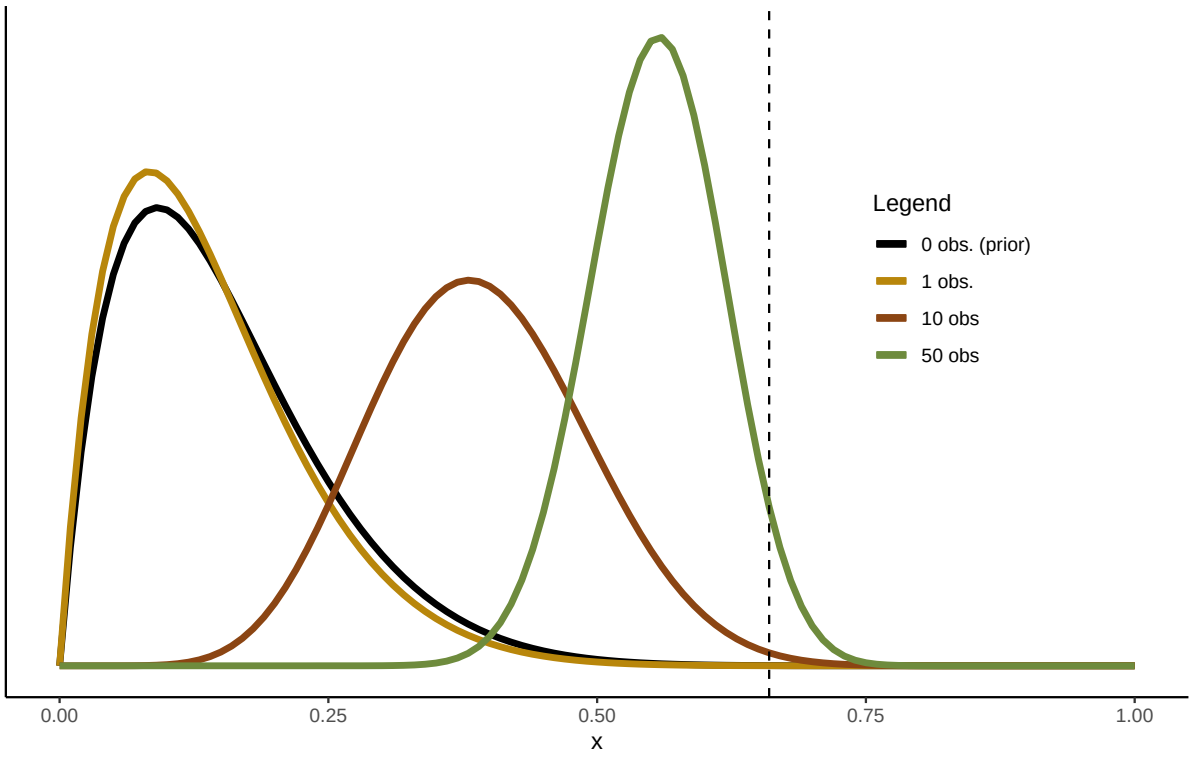


Figure 2.2: Sequentional Updating of Beta distribution, strong prior.

Using the same simulations as used in Figure 2.1 above, we see from the distributions in Figure 2.2 that 50 observations are now no longer enough to converge to the sample mean—the pull of the prior is too strong. In general, the posterior will always lie between the prior and the maximum likelihood estimate in a Bayesian setup, although the Bayesian and ML estimates will converge as $N \to \infty$. We can get a feel of this if we rewrite the updating equation with N observations as follows

$$\begin{aligned} E(\theta|x) &= \frac{\alpha + m}{\alpha + \beta + N} \\ &= \frac{\alpha + m}{K + N} \\ &= \frac{K}{K+N}\frac{\alpha}{K} + \frac{N}{K+N}\frac{m}{N} \\ &= (1-\gamma)\frac{\alpha}{K} + \gamma\frac{m}{N}, \end{aligned}$$

where $K \triangleq \alpha + \beta$ is the concentration. K then measures the "data points" contributed by the prior, while N measures the actual observations in the dataset as before. The parameter γ can then be taken as a measure of the shrinkage of the ML estimate $\frac{m}{N}$ to the prior mean $\frac{\alpha}{K}$.

While priors can be a strength of the Bayesian approach, one ought to choose them carefully, since they may just as well distort one's inferences from the data. In practice, it is thus always advisable to check the sensitivity of one's results to a more diffuse prior, unless one has truly good reasons to adopt a particular prior distribution. We will discuss such cases in some detail in later parts of these notes.

2.2.2 Predictive distribution

Very often, we will want to *predict* new data based on past data. Note that the predictive distribution (also variously referred to as *prior predictive* or *posterior predictive*, depending on the use one makes of it and the last updating step) is generally not the same as the estimated posterior distribution. To predict future outcomes based on past data, we will want to integrate over the possible posterior parameter values, such that

$$\begin{aligned} p(x_{N+1}|x) &= \int p(x_{N+1}|\theta)p(\theta|x)d\theta \\ &= \frac{\alpha + m}{\alpha + \beta + N}. \end{aligned}$$

That is, in the case of binary variables seen here, the predictive distribution is exceedingly simple, and coincides with the mean of the posterior. Note how the Bayesian approach can thus easily solve the issue arising from a series of four observations that all come up heads, as discussed above (aka *black swan paradox*). To solve this issue, simply assume a Beta prior with parameters $(1, 1)$. The posterior predictive will be:

$$p(x_{N+1}|x) = \frac{m+1}{N+2}.$$

This differs from 1 (and is also known as *Laplace's rule of succession*, which has been used as a solution to the paradox in non-Bayesian approaches). Notice, however, that as we accumulate more and more observations pointing in the same direction that prediction will become quite extreme, albeit without every going to 0 or 1 entirely. (And of course in a Bayesian modl we could use values of α and β different from 1, though we should be careful about what these parameters do and whether we are justified in choosing them).

2.3 Normal distribution

One of the most commonly used distributions in statistics is the normal, not least because of its computational tractability (but also because nature *loves* the normal distribution). Conveniently, the conjugate prior to the normal mean is itself normal, and their combination results

in a normal posterior. In this section, I will first present the general underlying theory, and then develop some examples.

2.3.1 Likelihood and Prior

Take a dataset $x = (x_1, ..., x_n)$. We will ultimately want to infer the probability of some hypothesis or parameter, conditional on these data. To start, however, we can depict the probability of the data, given the parameters of the distribution, otherwise known as the *likelihood*:

$$p(x|\mu, \sigma^2) = \frac{1}{\sqrt{2\pi}\sigma} exp\left(-\frac{1}{2\sigma^2}\sum_{i=1}^{n}(x_i - \mu)^2\right).$$

In Bayesian analysis, we will supplement this likelihood with a prior. It is important to note that, in Bayesian analysis, the data are always *given*, whereas it is the parameters that are uncertain. We thus want to use the known and given data to make inferences about the unknown and inherently uncertain parameters. Notice how this approach differs markedly from the assumption in classical or frequentist statistics, whereby parameters are given and certain but unknown to the statistician, and where we thus want to use random data samples to make inferences about these given but unknown parameters. In the latter, we could indeed use the sample quantities $\overline{x} \triangleq \frac{1}{n}\sum_i x_i$ and $s^2 \triangleq \frac{1}{n}\sum_i (x_i - \overline{x})^2$ to make inferences about the quantities μ and σ. Since theoretical occurrences from which the data are sampled are assumed to be infinite, our inferences should then be correct on average as long as we sample enough data (technically, in the limit as $N \to \infty$).

Since in the Bayesian setup the parameters μ and σ are themselves uncertain, we can model them as being drawn from a distribution. One way to think about this is that, although we estimate them from the data at hand, *they may still differ from the true underlying parameters due to sampling noise*. If that is the case, we ought to be able to stabilize the estimates by bringing to bear any prior knowledge we may have about these parameters. In the presence of sampling noise, that is indeed a normative way of reining in the influence of noisy outliers. As we will see shortly, combination of empirical ML estimates with a prior will serve to draw the parameter estimates towards the prior in a precise and principled way. This is the origin of one of the great strengths of Bayesian statistics—its applicability to small samples.

Let us thus assume that the mean is drawn from the following prior distribution (we assume the sampling variance to be given for the time being):

$$p(\mu|\mu_0, \sigma_0^2) = \frac{1}{\sqrt{2\pi}\sigma_0} exp\left(-\frac{1}{2\sigma_0^2}(\mu - \mu_0)^2\right),$$

where I have written $p(\mu|\mu_0, \sigma_0^2)$ to indicate the prior instead of $p(\mu)$ to emphasize the dependency on the parameters governing the normal distribution.

We can think of this prior distribution as incorporating anything we know about the mean of similar data. This could for instance incorporate any theoretical insights we may have about the quantities, such that they will always be constrained to a certain range (unless, of course, our aim is to test that theory). Alternatively, one could think about the prior as being built over time from estimates of μ obtained from similar datasets. Of course, we may then want to give a prior to the prior mean μ_0 itself, which is commonly referred to as a *hyperprior*. We will see in due time that meta-analysis may constitute a particularly principled way of building such a prior.

2.3.2 Bayesian updating with one datapoint

Let us assume for the moment that we only have one single data point, designated by x for simplicity. Throughout this section, we will also assume that the sample variance of this data point, σ^2, is known. This may seem like an artificial assumption, but it will help us developing the framework step by step. Such sequential approaches are indeed important in practice, especially when it comes to machine learning and artificial intelligence (but also in computational neuroscience). Under these assumptions, it is now straightforward to derive the posterior expectation of the unknown and uncertain parameter μ (the mean of the likelihood), conditional on the parameters of the prior:

$$\begin{aligned}\mathbb{E}[\,\mu \,|\, x,\, \sigma,\, \mu_0,\, \sigma_0\,] &= \frac{\sigma_0^2}{\sigma_0^2 + \sigma^2} \times x + \frac{\sigma^2}{\sigma_0^2 + \sigma^2} \times \mu_0 \\ &= \mu_0 + \frac{\sigma_0^2}{\sigma_0^2 + \sigma^2}(x - \mu_0) \\ &= x + \frac{\sigma^2}{\sigma_0^2 + \sigma^2}(\mu_0 - x).\end{aligned}$$

The three ways of writing the posterior expectation above highlight different intuitions about it. The first expression shows it as a convex combination between data point and prior mean; the second shows how the prior adjusts to the datapoint; and the third shows how the datapoint is drawn to the prior mean. This phenomenon in general is known as *shrinkage* or *pooling*, and it is a foundational concept in Bayesian statistics. For now, note that the degree of shrinkage is governed precisely by the ratio of variances: Shrinkage towards the mean of the prior is governed by the proportion of the total variance that is ascribed to sampling variation, σ^2. *Ceteris paribus*, the larger the sampling noise, the more shrinkage to the prior mean. *Mutatis mutandis*, the adjustment of the prior mean to the data point will be larger the larger the uncertainty surrounding the prior mean itself. In the limit, as $\sigma_0 \to \infty$, we can indeed see that the posterior mean converges to the maximum likelihood estimate. This is a useful result, since it implies that we can always recover the MLE estimate from Bayesian algorithms by imposing an (improper) prior with infinite variance.

Derivation of the normal updating equation

If you have never seen the equation above, you may wonder where it comes from. The result can be derived by multiplying the Gaussians describing the likelihood and prior to obtain the posterior, $p(\mu|x) = p(x|\mu) \times p(\mu)$. To start let us write out the distributions (we drop the normalization constants, since these can easily be reinstated at the end):

$$p(x|\mu) \propto exp\left(-\frac{1}{2}\frac{(x-\mu)^2}{\sigma^2}\right)$$
$$p(\mu) \propto exp\left(-\frac{1}{2}\frac{(\mu-\mu_0)^2}{\sigma_0^2}\right)$$

We next take logs to get rid of the exponentials, multiply out the squares, and group:

$$\begin{aligned} ln[p(\mu|x)] &\propto ln[p(x|\mu)] + ln[p(\mu)] \\ &\propto -\frac{1}{2}\frac{(x-\mu)^2}{\sigma^2} - \frac{1}{2}\frac{(\mu-\mu_0)^2}{\sigma_0^2} \\ &\propto -\frac{1}{2}\left(\frac{1}{\sigma^2}+\frac{1}{\sigma_0^2}\right)\mu^2 + \mu\left(\frac{x}{\sigma^2}+\frac{\mu_0}{\sigma_0^2}\right) - \left(\frac{x^2}{2\sigma^2}+\frac{\mu_0^2}{2\sigma_0^2}\right) \end{aligned}$$

The last step will be to "complete the square". That is, we know that our log-posterior function will again be a Gaussian, so that it will take the following form:

$$ln[p(\mu|x)] = -\left(\frac{\mu^2}{2\sigma_p^2} - \frac{2\mu\mu_p}{2\sigma_p^2} + \frac{\mu_p}{2\sigma_p^2}\right),$$

where $\mu_p = \mathbb{E}[\,\mu\,|\,x,\,\sigma,\,\mu_0,\,\sigma_0\,]$ is the mean of the posterior distribution, and σ_p designates the variance of the posterior distribution. We can now match the first expression in the last equation above to the first expression in the last line of the previous equation:

$$-\frac{\mu^2}{2\sigma_p^2} = -\frac{1}{2}\left(\frac{1}{\sigma^2}+\frac{1}{\sigma_0^2}\right)\mu^2,$$

which we solve for the variance of the posterior:

$$\sigma_p^2 = \frac{\sigma^2\sigma_o^2}{\sigma^2+\sigma_o^2}.$$

We then procced to matching the second term of the two equations:

$$-\frac{2\mu\mu_p}{2\sigma_p^2} = \mu\left(\frac{x}{\sigma^2}+\frac{\mu_0}{\sigma_0^2}\right).$$

Solving for the mean of the poserior, we get:

$$\mu_p = \frac{\sigma_0^2}{\sigma_0^2+\sigma^2}\,x + \frac{\sigma^2}{\sigma_0^2+\sigma^2}\,\mu_0.$$

While the calculations are tedious, there is clearly no magic involved.

Having obtained the posterior mean, we now also need to assess the variance surrounding the posterior mean. This will provide us with a direct measure of the uncertainty surrounding the estimate (NOTE: the standard deviation of the posterior in Bayesian analysis coincides with what is called a *standard error* in classical analysis). The variance of the posterior mean, conditional on the single data point being observed, will thus be:

$$\mathbb{V}(\mu \,|\, x,\, \sigma,\, \sigma_0) = \frac{\sigma^2\sigma_0^2}{\sigma^2 + \sigma_0^2} = \frac{1}{\frac{1}{\sigma^2} + \frac{1}{\sigma_0^2}} \triangleq (\lambda + \lambda_0)^{-1},$$

where $\lambda \triangleq \sigma^{-2}$ is the *precision* of the likelihood, and $\lambda_0 \triangleq \sigma_0^{-2}$ the precision of the prior. This notation is particularly intuitive, and it shows that the uncertainty surrounding the posterior will be inversely proportional to the sum of the precisions of the likelihood and the prior.

Let us discuss an example. Assume we have data made up of a collection of certainty equivalents (*CEs*), $c = (c_1, ..., c_n)$. Let us transform the CEs to indicate relative risk premia, $r_i \triangleq p_i - \frac{c_i - y_i}{x_i - y_i}$. We want to find the average relative risk premium in the sample, $\bar{r}$. We can simulate some data by drawing 50 observations from a distribution of relative risk premia, with a mean of 0.5 and a variance of about 0.1. Let us assume that we have a prior indicating that people are risk neutral on average, albeit with a fair degree of uncertainty $p(\mu \,|\, \mu_0, \sigma_0^2) = \mathcal{N}(0\,,\, 0.1)$. We can now take one data point at the time, and update this prior using the equation derived above. Note that the posterior resulting from the updating process with one data point can simply be taken to be the prior of the next iteration, thus nicely illustrating how information accumulates over time. Below, I do this starting from the first data point and following the sequence.

```
rrp <- rnorm(50,0.5,0.316)

lambda <- 1/var(rrp)
lambda0 <- 1/0.1
mu0 <- 0
x <- append(rrp,0,0)

for(i in 2:51){
lambda0[i] <- lambda0[i-1] + lambda
mu0[i] <- mu0[i-1] + (lambda)/(lambda + lambda0[i])*(x[i] - mu0[i-1])
}

ggplot(data = data.frame(x = c(-0.8, 0.8)), aes(x)) +
  stat_function(fun = dnorm, n = 101, args = list(mean = mu0[1], sd = sqrt(1/lambda0[1])), si
  stat_function(fun = dnorm, n = 101, args = list(mean = mu0[2], sd = sqrt(1/lambda0[2])) ,si
  stat_function(fun = dnorm, n = 101, args = list(mean = mu0[6], sd = sqrt(1/lambda0[6])), si
  stat_function(fun = dnorm, n = 101, args = list(mean = mu0[11], sd = sqrt(1/lambda0[11])),
```

```
  stat_function(fun = dnorm, n = 101, args = list(mean = mu0[31], sd = sqrt(1/lambda0[31]))),
  geom_vline(xintercept=mean(rrp),linetype="dashed",colour="red", size=1.2) +
  ylab("")  + scale_y_continuous(breaks = NULL) +
  scale_colour_manual("Legend", values = c("black","darkgoldenrod", "chocolate4", "darkolive
```

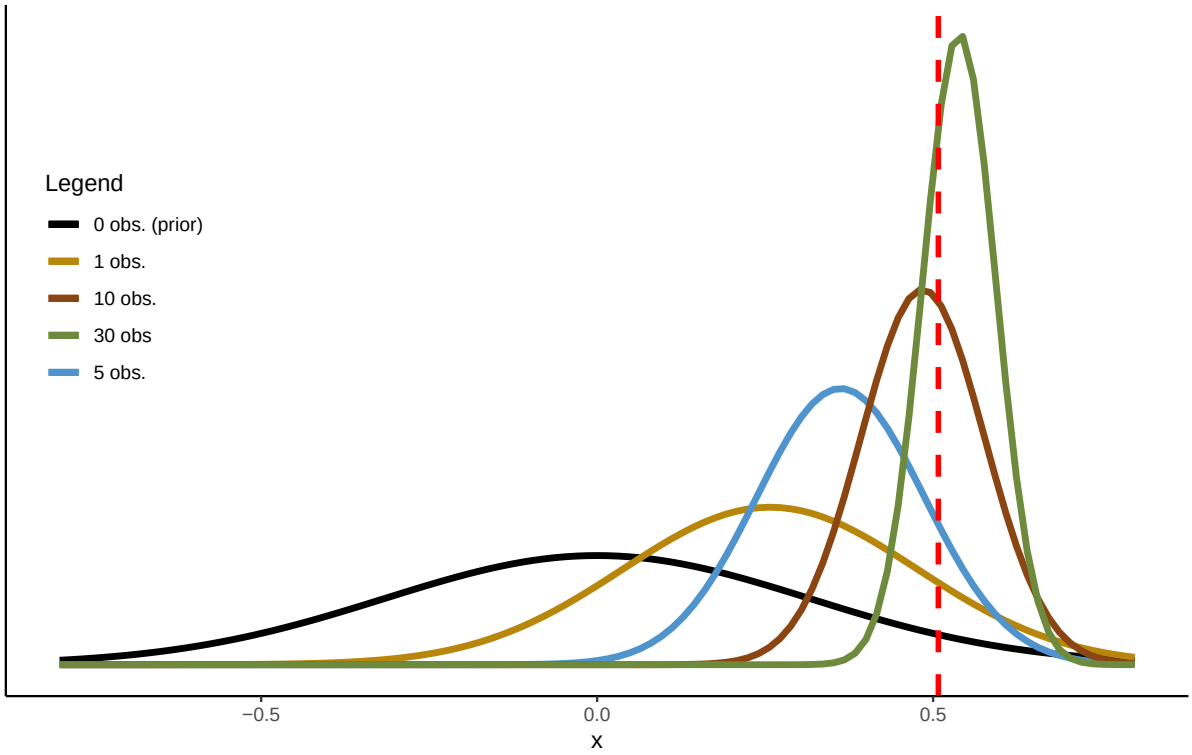


Figure 2.3: Sequentional Updating of Normal distribution.

The updating sequence in Figure 2.3 has been obtained by using the true sampling variance, and assuming it to be known. One can see that after 30 observations, the mean is learned almost perfectly and with a high degree of confidence. Note that while I start from prior parameters μ_0 and λ_0, these are quickly updated to incorporate the information provided by the data, with the resulting posterior serving as the prior for the next observation.

2.3.3 Integrating several data points at once

Typically, we will want to combine several data points at once with the prior. Notice, however, that there is no difference between updating the prior with several data points at once or doing it sequentially. On the one hand, this makes it straightforward to derive the general case from the case seen above. On the other, this means that one could employ sequential

methods if desirable, which may be interesting in learning algorithms and machine learning. In this section, I will show the equations needed to update the prior with several data points at once. Once again, it is assumed that the sampling variance is known.

Assuming that the observations in x are i.i.d., we know that the sample mean will follow the distribution $p(\,\overline{x}\,|\,\mu\,) = \mathcal{N}(\,\mu\,,\,\frac{\sigma^2}{n}\,)$. We can thus directly derive the posterior distribution as follows:

$$p(\mu\,|\,x) \sim \mathcal{N}\left(\frac{n\sigma_0^2}{n\sigma_0^2+\sigma^2} \times \overline{x} + \frac{\sigma^2}{n\sigma_0^2+\sigma^2}\mu_0 \;\;,\;\; (\lambda_0 + n\lambda)^{-1}\right).$$

Once again, we find shrinkage towards the mean of the prior, or a convex combination of sample mean and prior mean. Writing the updating equation for the mean as $E[\mu|x] = \mu + \frac{\sigma_0^2}{\sigma_0^2+\frac{\sigma^2}{n}}(\overline{x}-\mu)$ allows us to derive a number of interesting results. In the limit as $n \to 0$ we simply revert to the mean of the prior. As $n \to \infty$ the weight on the prior mean goes to zero. This shows that with enough data, the prior will be immaterial. It is for small datasets that the prior will be really important.

It is also instructive to look at the variance of the posterior in terms of precisions. This takes the following form:

$$\mathbb{V}(\mu\,|\,x) = \frac{1}{\sigma_0^2} + \frac{n}{\sigma^2} = \lambda_0 + n\lambda.$$

Unsurprisingly, this is the same result we obtained above, when we added new observations one observation at a time, every time adding $\frac{1}{\sigma^2}$ to the prior precision. It is interesting to see how the two precisions enter into the equation symmetrically. That is, for large n the posterior will depend mostly on σ^2. For $\sigma = \sigma_0$, as used in the illustration above, the prior will thus have the same weight as one extra observation with mean μ_0. This explains the quick convergence to the sample mean we observed in the simulated example above.

```
ggplot(data = data.frame(x = c(-1, 1)), aes(x)) +
  stat_function(fun = dnorm, n = 101, args = list(mean = 0, sd = 0.316), size=1.5) +
  stat_function(fun = dnorm, n = 101, args = list(mean = mu0[11], sd = sqrt(1/lambda0[11]) )
  stat_function(fun = dnorm, n = 101, args = list(mean = mean(head(rrp,n=10)), sd = (1/sqrt(
  ylab("")  + scale_y_continuous(breaks = NULL) +
  annotate(geom="text", x=-0.1, y=1.5, label="prior") +
  annotate(geom="text", x=0.3, y=4.5, label="N=10") +
  annotate(geom="text", x=0.7, y=1.3, label="likelihood")
```

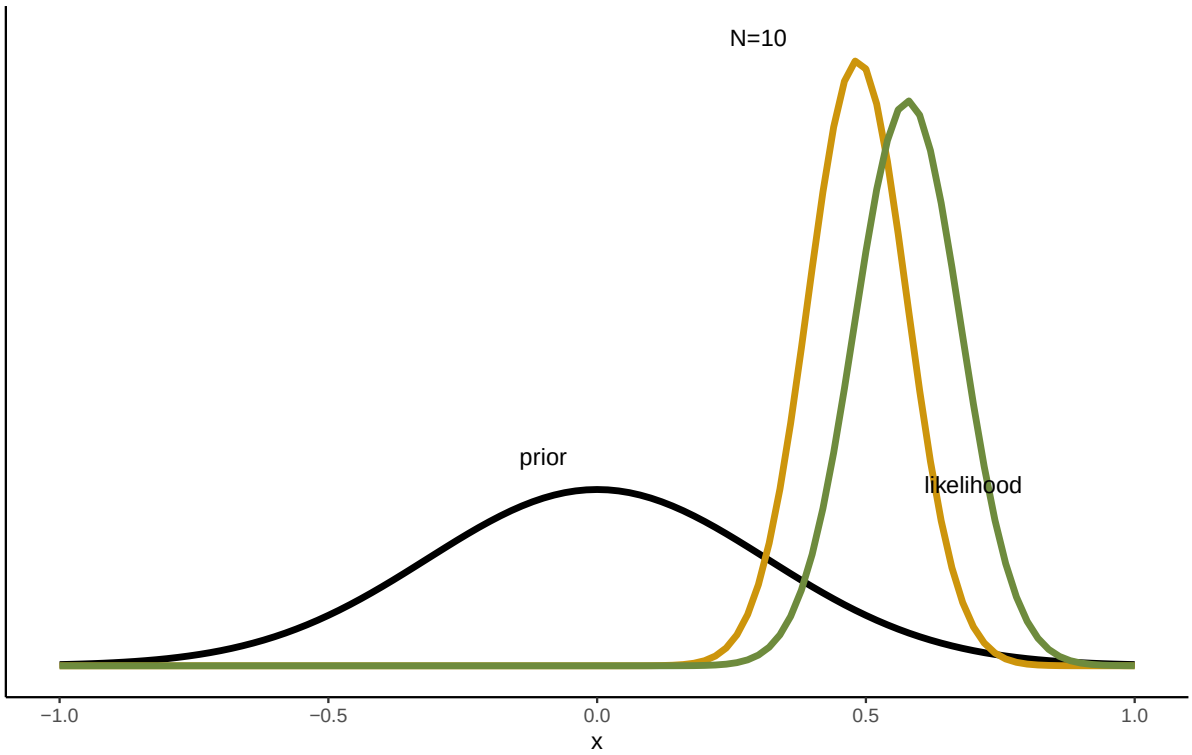


Figure 2.4: Sequentional Updating of Normal distribution with 10 draws.

It may be interesting to take one more look at the effect of the prior with these insights in mind. We can use the same simulated data as above and take, say, the first 10 observations. We start by reproducing the graph above, but this time comparing the posterior distribution obtained after 10 draws to both the prior *and* to the maximum likelihood estimate, obtained based o the data alone. Figure 2.4 shows the result. We can see that the posterior, even after only 10 observations, is very close to the maximum likelihood estimate. In other words, the prior has fairly little influence in this example—one tenth of the evidence provided by the data to be exact.

```
lambda <- 1/var(rrp)
lambda0 <- 100
mu0 <- 0
x <- append(rrp,0,0)


for(i in 2:51){
lambda0[i] <- lambda0[i-1] + lambda
mu0[i] <- mu0[i-1] + (lambda)/(lambda + lambda0[i])*(x[i] - mu0[i-1])
}
```

```
ggplot(data = data.frame(x = c(-1, 1)), aes(x)) +
  stat_function(fun = dnorm, n = 101, args = list(mean = 0, sd = sqrt(1/lambda0[1])), size=1
  stat_function(fun = dnorm, n = 101, args = list(mean = mu0[11], sd = sqrt(1/lambda0[11]))
  stat_function(fun = dnorm, n = 101, args = list(mean = mean(head(rrp,n=10)), sd = sqrt(1/1
  ylab("")  + scale_y_continuous(breaks = NULL) +
  annotate(geom="text", x=-0.23, y=1.5, label="prior") +
  annotate(geom="text", x=0.12, y=4.5, label="N=10") +
  annotate(geom="text", x=0.7, y=1.3, label="likelihood")
```

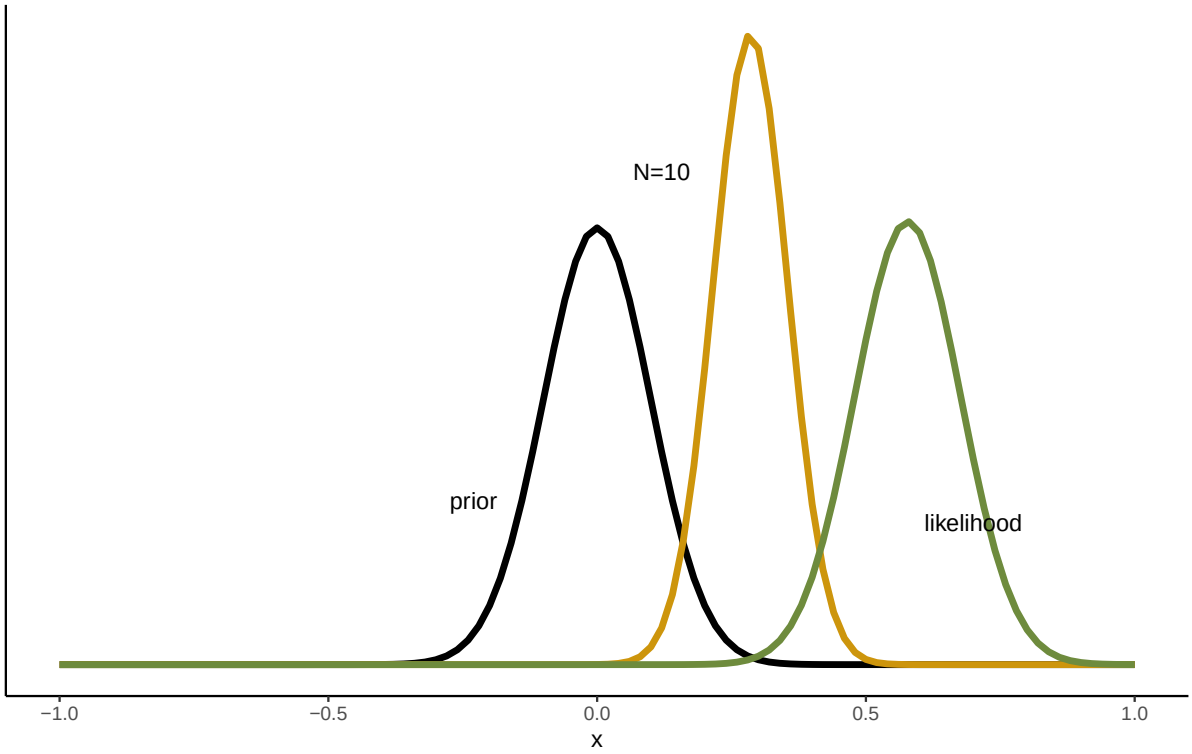


Figure 2.5: Sequentional Updating of Normal distribution, strong prior.

Next, we can repeat this exercise, but imposing a much narrower prior reflected by a large precision. The result is shown in Figure 2.5. We can see how the more precise prior exercises a stronger pull on the posterior estimate. Whereas the diffuse prior has little influence on the posterior even after 10 draws, resulting in a posterior distribution very close to the maximum likelihood estimate, the precise prior used just above means that the posterior mean estimated lands in the middle between the prior and the MLE.

Sequential updating is very natural in the Bayesian context, and can be put to use in contexts where it might be useful to learn gradually about the posterior. To see this more clearly, we

can write the posterior as follows:

$$p(\mu|x) \propto \left[p(\mu) \prod_{i=1}^{n-1} p(x_i|\mu) \right] p(x_n|\mu),$$

where the term in square brackets represents the posterior after $n-1$ observations (which can be seen as a prior for observation n), and the last term to the right presents the contribution of the n^{th} observation. Notice also that we could now take the posterior obtained from the last sequential updating procedure above, and further update it with observations 11-50. I do not do this here for parsimony. Suffice it to say that, given the strength of the prior now imposed, even 50 observations are no longer enough for convergence to the ML estimate, as shown in Figure 2.6.

```
ggplot(data = data.frame(x = c(-1, 1)), aes(x)) +
  stat_function(fun = dnorm, n = 101, args = list(mean = 0, sd = sqrt(1/lambda0[1])), size=1
  stat_function(fun = dnorm, n = 101, args = list(mean = mu0[51], sd = sqrt(1/lambda0[51]))
  stat_function(fun = dnorm, n = 101, args = list(mean = mean(head(rrp,n=10)), sd = sqrt(1/1
  ylab("")  + scale_y_continuous(breaks = NULL) +
  annotate(geom="text", x=-0.23, y=1.5, label="prior") +
  annotate(geom="text", x=0.5, y=7, label="N=50") +
  annotate(geom="text", x=0.7, y=1.3, label="likelihood")
```

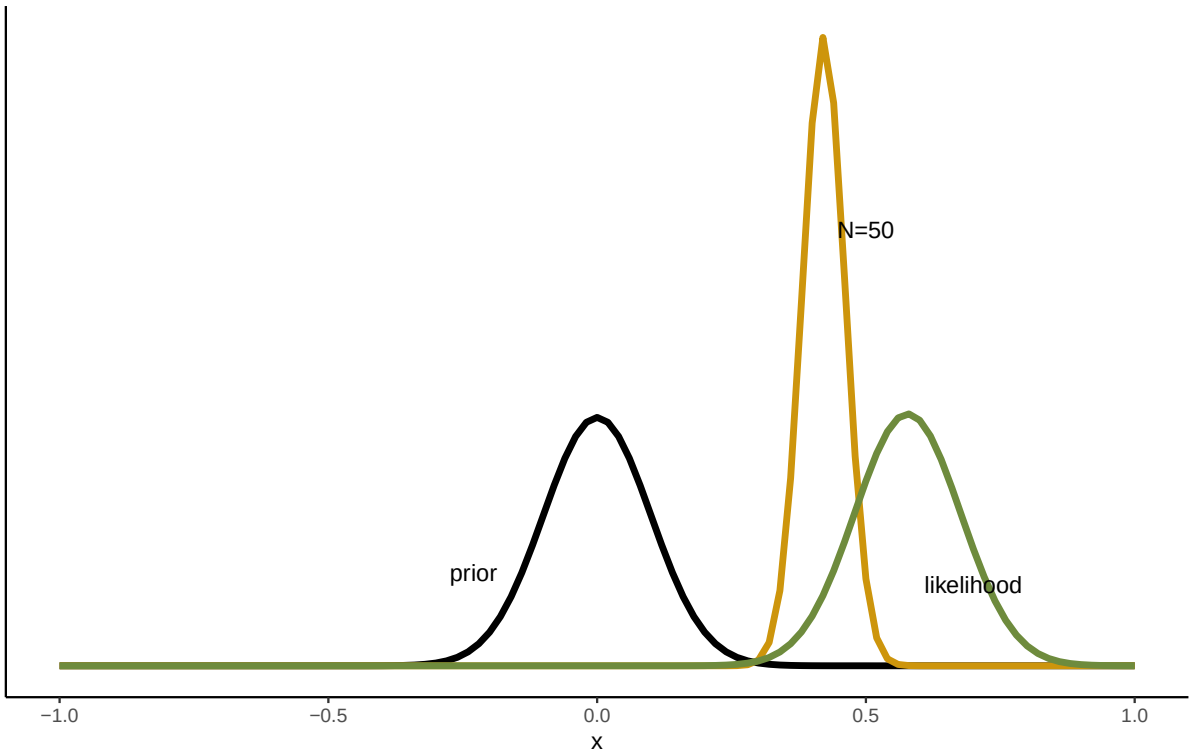


Figure 2.6: Sequentional Updating of Normal distribution with 50 draws, strong prior.

2.3.4 The posterior predictive distribution

We will often want to use the parameters obtained from past data to predict new data. For this, we need the *posterior predictive distribution*. Formally, we will want to predict observation(s) $\tilde{x}$ from past observation(s) x, so as to obtain $p(\tilde{x}|x)$. For this, we will need to consider the likelihood of the new data, based on the parameters estimated on the old data

$$p(\tilde{x} \mid x) \propto \int p(\tilde{x} \mid \mu)\, p(\mu \mid x)\, d\mu.$$

Note that the first distribution within the integral is just the likelihood, applied to the new data $\tilde{x}$ we want to predict. The second distribution is the posterior distribution of μ, based on data x. We do not need to condition the predicted quantity directly on past data, since the information they contain is fully reflected in this posterior. We need to integrate over all values of μ, because the posterior estimate of μ is itself an uncertain quantity (i.e., it has a distribution). We can rewrite this equation as:

$$p(\tilde{x}|x) \propto \int exp\left(-\frac{1}{2\sigma^2}(\tilde{x}-\mu)^2\right) exp\left(\frac{1}{2\sigma_p^2}(\mu-\mu_p)^2\right) d\mu,$$

where $\mu_p \triangleq \mu_0 + \frac{\sigma_0^2}{\sigma_0^2+\sigma^2}(x-\mu_0)$ and $\sigma_p^2 \triangleq \frac{\sigma^2\sigma_0^2}{\sigma^2+\sigma_0^2}$ are the mean and variance of the *posterior*, respectively. It can then be shown that

$$\begin{aligned}\mathbb{E}[\tilde{x} \mid x] &= \mu_p \\ \mathbb{V}[\tilde{x} \mid x] &= \sigma^2 + \sigma_p^2,\end{aligned}$$

that is, the posterior predictive distribution will have a mean equal to the posterior mean based on x, with a variance equal to the sum of the sampling variance and the posterior variance of the mean, which derives from uncertainty in the estimation of μ_p.

2.3.5 Conjugate priors for the variance

We have now seen how to infer the value of the mean when the variance is known. However, the variance will not be known in general, and we will thus need to make inferences about the variance of the sampling distribution. To simplify things, we will assume for the time being that the mean is known, and that we are trying to infer only the variance, making this the dual situation of the one seen above.

While different parametrizations are used in the literature, I will work with a prior for the precision, $\lambda \triangleq \frac{1}{\sigma^2}$. The likelihood function then takes the following form:

$$\begin{aligned}p(x \mid \lambda) &= \prod_{i=1}^{n} \mathcal{N}(x_i|\mu, \lambda^{-1}) \\ &\propto \lambda^{\frac{1}{2}} exp\left[-\frac{\lambda}{2}\sum_{i=1}^{n}(x_i-\mu)^2\right].\end{aligned}$$

Given the above formulation in terms of precision, the conjugate prior will take the form of a *Gamma* distribution, defined as follows:

$$Gam(\lambda|\alpha,\beta) = \frac{1}{\Gamma(\alpha)}\beta^\alpha \lambda^{\alpha-1} exp(-\beta\lambda),$$

where $\Gamma(\alpha)$ is a Gamma function which serves to normalize the distribution.

Assume now a prior distribution $Gam(\lambda|\alpha_0, \beta_0)$. Multiplying this with the likelihood function above we obtain a posterior distribution, which, conveniently, again follows a Gamma:

$$p(\lambda|D) \propto \lambda^{\alpha_0 - 1} \lambda^{\frac{N}{2}} exp\left[-\beta_0 \lambda - \frac{\lambda}{2} \sum_{i=1}^{n} (x_i - \mu)^2\right],$$

which we can designate as $Gam(\lambda|\alpha_n, \beta_n)$. We can thus write the updating equations as follows:

$$\begin{aligned} \alpha_n &= \alpha_0 + \frac{n}{2} \\ \beta_n &= \beta_0 + \frac{1}{2} \sum_{i=1}^{n} (x_i - \mu)^2 \\ &= \beta_0 + \frac{n}{2} s^2, \end{aligned}$$

where s^2 is the maximum likelihood estimate of the variance in the data. Note that the normalization factor $\Gamma(\alpha)$ can be found easily in the end, so that there is no need to keep track of it during the updating process. We can easily see that n data points increase the coefficient α by $\frac{n}{2}$. It follows immediately that one data point increases it by $\frac{1}{2}$. The parameter β, on the other hand, is increased proportionally to the squared deviation from the mean. Since the mean of a Gamma distribution is $\mathbb{E}[\lambda] = \frac{\alpha}{\beta}$, the crucial element for the evolution of the precision will be which of the two parameters increases more quickly. The variance of the precision is defined as $\mathbb{V}[\lambda] = \frac{\alpha}{\beta^2}$.

```
rrp <- rnorm(50,0.5,0.316)
lambda <- 1/var(rrp)
alpha <- 1
beta <- 1
x <- append(rrp,0,0)

for(i in 2:51){
alpha[i] <- alpha[i-1] + 0.5
beta[i] <- beta[i-1] + 0.5*(x[i] - mean(rrp))^2
}

ggplot(data = data.frame(x = c(0, 20)), aes(x)) +
  stat_function(fun = dgamma, args = list(shape = alpha[1], rate=beta[1]), size=1.5,aes(colou
  stat_function(fun = dgamma, args = list(shape = alpha[2], rate=beta[2]), size=1.5, aes(colo
  stat_function(fun = dgamma, args = list(shape = alpha[6], rate=beta[6]), size=1.5, aes(colo
  stat_function(fun = dgamma, args = list(shape = alpha[11], rate=beta[11]), size=1.5, aes(co
  stat_function(fun = dgamma, args = list(shape = alpha[51], rate=beta[51]), size=1.5, aes(co
  geom_vline(xintercept=1/var(rrp),colour="red",linetype="dashed") +
  ylab("") + xlab("precision")  + scale_y_continuous(breaks = NULL) +
  scale_colour_manual("Legend", values = c("black","darkgoldenrod", "chocolate4", "darkolive
```

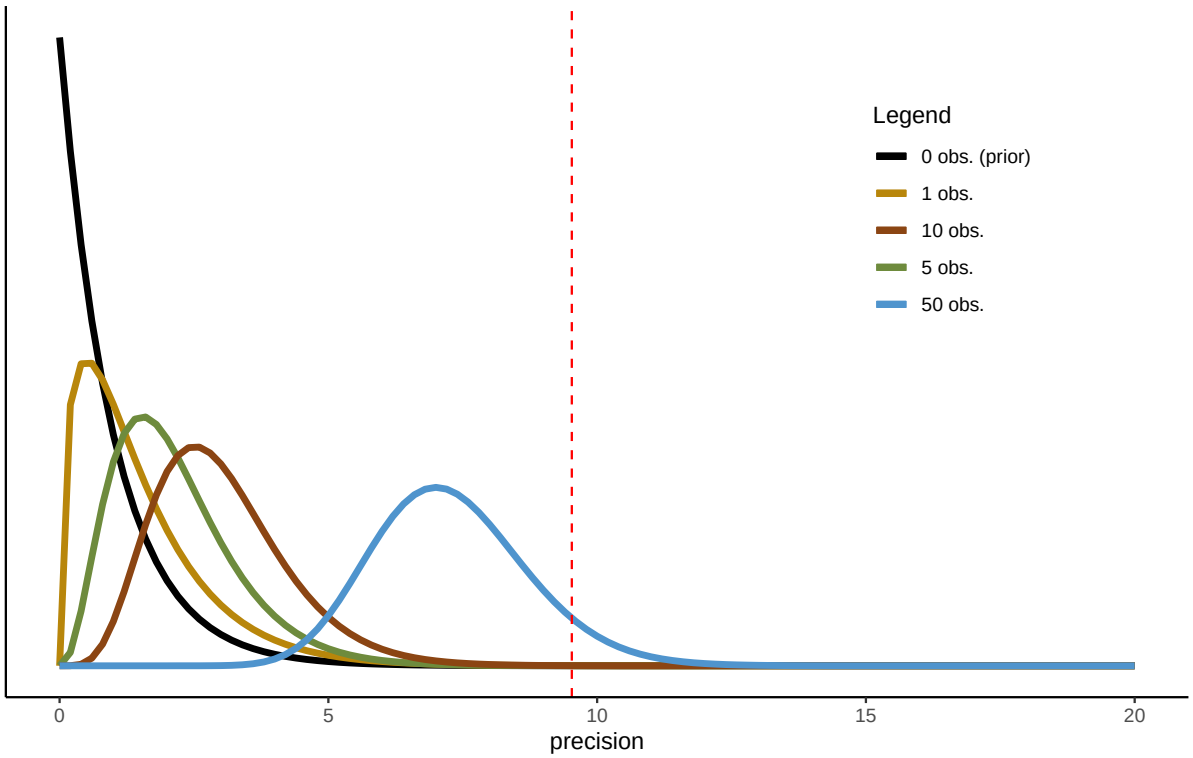


Figure 2.7: Sequential updating of precision.

This updating process can thus be implemented in a sequential fashion. Let us again consider an example. We start from a prior with parameters $\alpha_0 = 1$ and $\beta_0 = 1$, and we can use the same data of relative risk premia generated above. The result of the updating process is depicted in Figure 2.7. Convergence to the true mean now seems very slow, and even after 50 observations the posterior estimate is still far from the sample mean, indicated by the vertical dashed line. What has happened? A quick examination of the parameters used above will tell you that this is due to the high probability mass assigned to very low precision levels in the prior we have used. In particular, the prior distribution used has a mean of 1 (compared to a sampling mean of the precision of 12.55), as well as a variance of 1, indicating high confidence in the (unrealistic) prior mean of the precision.

```
alpha <- 1
beta <- 0.1
x <- append(rrp,0,0)

for(i in 2:51){
alpha[i] <- alpha[i-1] + 0.5
beta[i] <- beta[i-1] + 0.5*(rrp[i-1] - mean(rrp))^2
}
```

```
ggplot(data = data.frame(x = c(0, 20)), aes(x)) +
  stat_function(fun = dgamma, args = list(shape = alpha[1], rate=beta[1]), size=1.5,aes(colou
  stat_function(fun = dgamma, args = list(shape = alpha[2], rate=beta[2]), size=1.5, aes(colo
  stat_function(fun = dgamma, args = list(shape = alpha[6], rate=beta[6]), size=1.5, aes(colo
  stat_function(fun = dgamma, args = list(shape = alpha[11], rate=beta[11]), size=1.5, aes(co
  stat_function(fun = dgamma, args = list(shape = alpha[51], rate=beta[51]), size=1.5, aes(co
  geom_vline(xintercept=1/var(rrp),colour="red",linetype="dashed") +
  ylab("") + xlab("precision")  + scale_y_continuous(breaks = NULL) +
  scale_colour_manual("Legend", values = c("black","darkgoldenrod", "chocolate4", "darkoliveg
  theme(legend.position = c(0.85, 0.8))
```

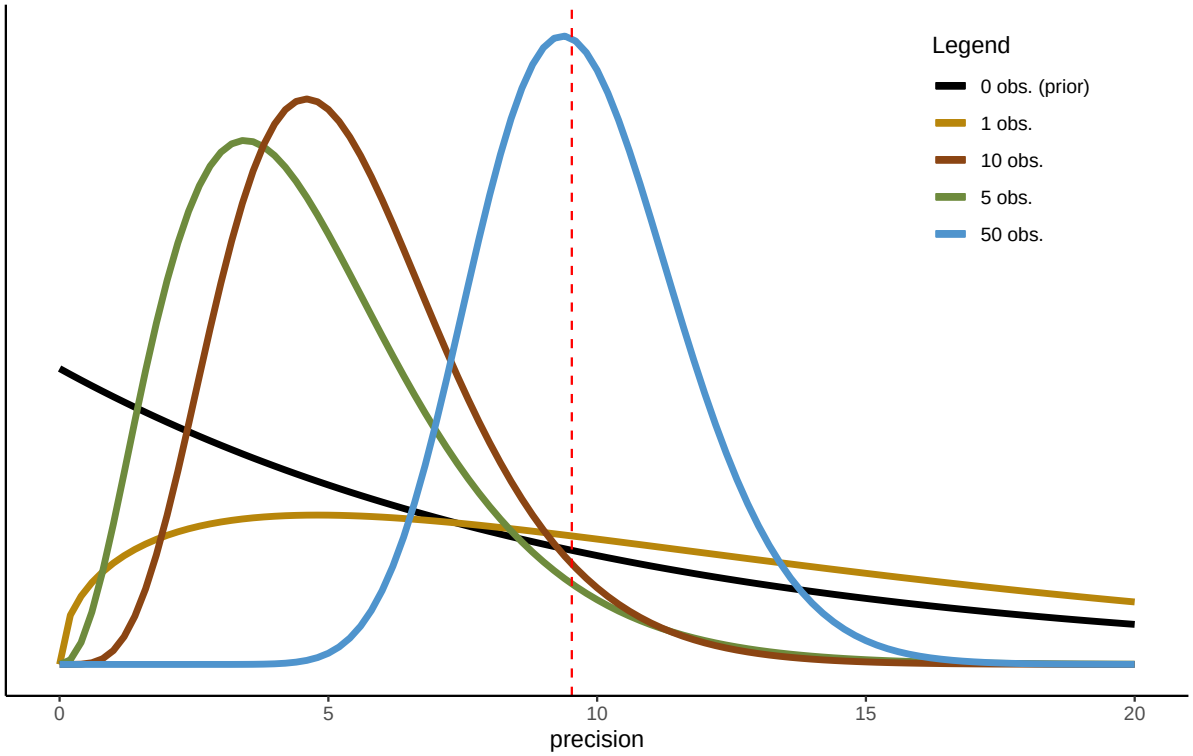


Figure 2.8: Sequential updating of precision, weak prior.

To convince yourself of this, try using a more diffuse prior. The distributions shown in Figure 2.8 do just this, by using a shape parameter of $\alpha_0 = 1$ for the prior as before, but a much lower rate parameter $\beta_0 = 0.1$. Note that this will have two effects. The mean of the prior, defined as $\frac{\alpha_0}{\beta_0} = 10$ is now much larger. At the same time, the variance, defined as $\frac{\alpha_0}{\beta_0^2} = 100$, makes the prior much more diffuse. Convergence to the sample variance is now much quicker, and the estimated mean (of the variance) coincides with the sample mean after

50 iterations/data points (though the mode falls slightly to the left of it—keep in mind that the distribution is not symmetric).

```
alpha <- 0.01
beta <- 0.01

for(i in 2:51){
alpha[i] <- alpha[i-1] + 0.5
beta[i] <- beta[i-1] + 0.5*(rrp[i-1] - mean(rrp))^2
}

ggplot(data = data.frame(x = c(0, 20)), aes(x)) +
  stat_function(fun = dgamma, args = list(shape = alpha[1], rate=beta[1]), size=1.5,aes(colou
  stat_function(fun = dgamma, args = list(shape = alpha[2], rate=beta[2]), size=1.5, aes(colo
  stat_function(fun = dgamma, args = list(shape = alpha[6], rate=beta[6]), size=1.5, aes(colo
  stat_function(fun = dgamma, args = list(shape = alpha[11], rate=beta[11]), size=1.5, aes(co
  stat_function(fun = dgamma, args = list(shape = alpha[51], rate=beta[51]), size=1.5, aes(co
  geom_vline(xintercept=1/var(rrp),colour="red",linetype="dashed") +
  ylab("") + xlab("precision")  + scale_y_continuous(breaks = NULL) +
  scale_colour_manual("Legend", values = c("black","darkgoldenrod", "chocolate4", "darkoliveg
  theme(legend.position = c(0.85, 0.8))
```

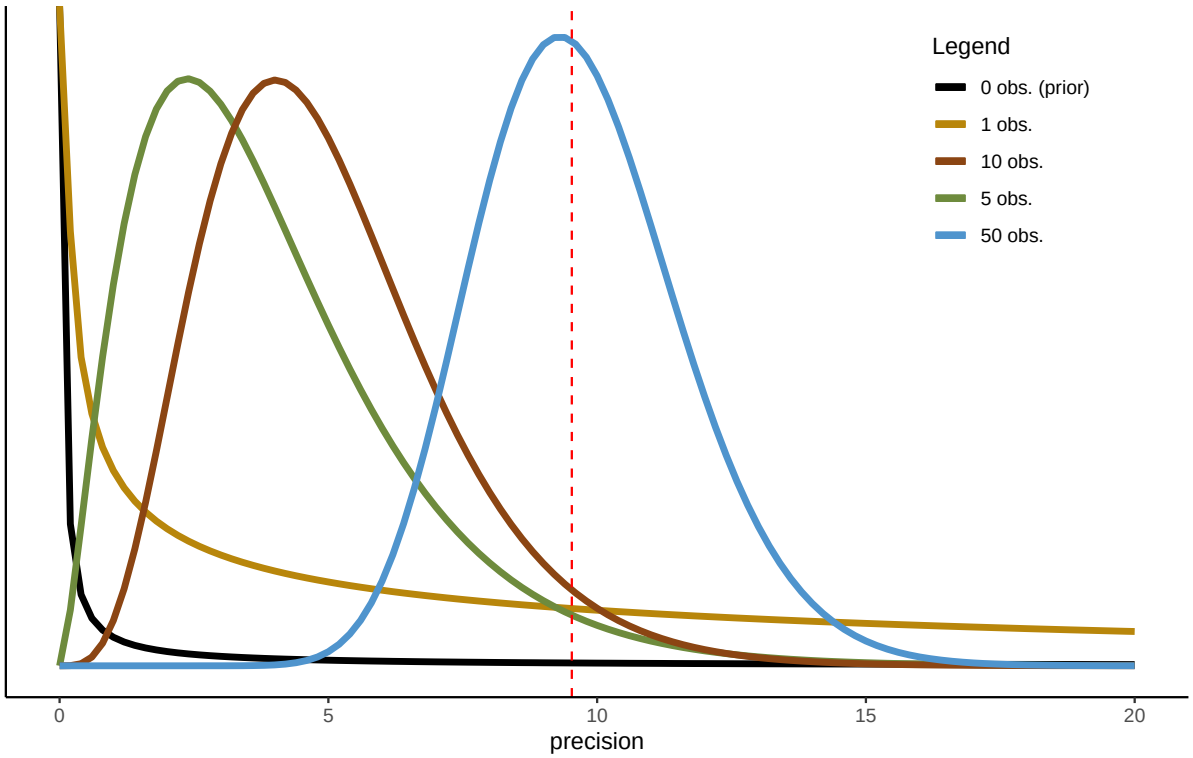


Figure 2.9: Sequential updating of precision, ceteris paribus.

You may well consider this example as cheating. After all, we have not only increased the variance to 100, making the prior more diffuse, but we have also adjusted the mean to 10, bringing it very close to the sample mean, and thus making it informative. Alternatively, one could thus try and keep the mean constant at a low level, while only allowing the variance to increase. Figure 2.9 shows an example doing exactly that. I have now set both the shape and the rate parameters to 0.1. This implies a mean of the precision equal to 1, as in the first example above, but a variance equal to 100, like in the second example above. Convergence to the sample mean is once again complete after 50 data points, thus showing that the result was driven mostly by the diffuseness of the prior (the larger variance), and not by the higher mean.

2.3.6 Conjugate priors: learning mean and variance together

We have now seen how to update the mean of the normal distribution with new information, as well as how to update the variance. An intuitive conclusion would seem that we can simply put the two steps together to update mean and variance at once, and to thus learn the entire distribution. This intuition, however, is only partially correct. The additional issue arising when updating both parameters at once is that, conditional on the data, they are not

independent from each other (to convince yourself of this, simply take a look at the functional form of the normal distribution). Ignoring this dependence would imply that the prior is not conjugate after all.

The solution turns out to be a prior that follows a so-called *normal-gamma distribution*, and indicated as $\mathcal{NG}(\mu_0, \kappa_0, \alpha_0, \beta_0)$. Assume as usual that the probability of the data conditional on the parameters follows a normal distribution, so that $p(x_i|\mu, \lambda) = \mathcal{N}(\mu, \lambda^{-1})$. Note that I have now conditioned the probability of the data explicitly on both the mean and the variance (actually: the precision), since both are now assumed unknown. We now want to identify a joint prior for the two unknown model parameters. To do this, we can exploit the fact that $p(\mu, \lambda) = p(\mu|\lambda)\, p(\lambda)$. As it turns out, $p(\lambda)$ will again take the form of a Gamma distribution, whereas $p(\mu|\lambda)$ will take the form of a normal. The dependency between the parameters is then captured by making the precision of the conditional distribution of the mean a linear function of λ:

$$p(\mu, \lambda) = \mathcal{N}(\mu|\mu_0, (\kappa_0\lambda)^{-1})\, Gam(\lambda|\alpha_0, \beta_0),$$

where β_0 is again the *rate* parameter of the Gamma distribution. The scaling parameter κ_0 drives home the dependency between the different parameters, and expresses the precision of the prior relative to the precision of the data. To better understand this, it is instructive to take a look at the updating equations. I here start from an update with n observations. Let us start from the equation for the mean conditional on data $x = (x_1, ..., x_n)$ and on the precision:

$$\mathbb{E}[\mu|x, \lambda] = \frac{n\lambda}{n\lambda + \kappa_0\lambda}\overline{x} + \frac{\kappa_0\lambda}{n\lambda + \kappa_0\lambda}\mu_0 \tag{2.1}$$

$$= \mu_0 + \frac{n\lambda}{n\lambda + \kappa_0\lambda}(\overline{x} - \mu_0) \tag{2.2}$$

$$= \mu_0 + \frac{n}{n + \kappa_0}(\overline{x} - \mu_0), \tag{2.3}$$

where $\overline{x} \triangleq \frac{1}{n}\sum_i^n x_i$. Notice that this equation is equivalent to the one used previously, with two differences. For one, I have expressed the updating weights in terms of precisions instead of in terms of variances. To see the equivalence, let $\kappa_0\lambda = \frac{1}{\sigma_0^2}$ and $n\lambda = \frac{1}{\sigma^2}$. Then $\frac{n\lambda}{n\lambda + n_0\lambda} = \frac{\frac{1}{\sigma^2}}{\frac{1}{\sigma^2} + \frac{1}{\sigma_0^2}} = \frac{\sigma_0^2}{\sigma_0^2 + \sigma^2}$. The second and more substantive change consists precisely in the reparameterization of the prior precision as $\kappa_0\lambda$. The scaling parameter κ_0 serves to fix the relative strength of the prior. The updating equation for the variance of the mean will then simply be as follows:

$$\mathbb{V}(\mu|x) = (n\lambda + \kappa_0\lambda)^{-1}.$$

The equations just described are conditional not only on the data x, but also on the precision λ. We thus still need to consider the probability of λ, $p(\lambda|x)$. We already know that this equation

will take the form of a Gamma. To obtain the posterior estimates of the parameters of this distribution, in the case of an n-dimensional data vector we can use the following updating equations:

$$\begin{aligned}
\alpha_n &= \alpha_0 + \frac{n}{2} \\
\beta_n &= \beta_0 + \frac{1}{2}\sum_{i=1}^{n}(x_i - \mu)^2 \\
&= \beta_0 + \frac{1}{2}\sum_{i=1}^{n}(x_i - \overline{x})^2 + \frac{1}{2}\frac{n\kappa_0}{n+\kappa_0}(\overline{x} - \mu_0)^2) \\
\kappa_n &= \kappa_0 + n,
\end{aligned}$$

where the last expression to the right in the middle equation obtains from taking into account the expression for μ above. That is, since both μ and λ need to be learned from the data and the dependence of both on x creates inter-dependencies between the variables, only an equation system explicitly taking account of these inter-dependencies will yield the desired results. In terms of interpretation, it is easy to see that the β value of the posterior will depend both on the variance of the data sample, $\sum_{i=1}^{n}(x_i - \overline{x})^2$, and on the deviation of the sampling mean from the prior mean, $(\overline{x} - \mu_0)^2$. The weight of the latter, in turn, will depend on the supposed 'number of observations' in each.

Simulations will serve to shed some light on the working mechanism of the equations. Some issues need to be kept in mind. First of all, while we have started from the probability of the mean conditional on the precision, we now need to work in the opposite direction. Finally, one needs to take good care not to get confused between the different variables—both mean and variance will have a posterior mean and a posterior variance of their own!

```
rrp <- rnorm(50,0.5,0.316)

n <- length(rrp)
mu0 <- 0
alpha <- 0.01
beta <- 0.01
k0 <- 1

alpha_n <- alpha + n/2
beta_n <- beta + 0.5*sum((rrp - mean(rrp))^2) + (n*k0)/(2*(n + k0)) * (mean(rrp) - mu0)^2
mu <- mu0 + (n)/(n  + k0) * (mean(rrp) - mu0)
k_n = k0 + n
lambda <- alpha_n/beta_n
lambda_n <- k_n*lambda
```

```
# learned distribution compared to prior and ML estimate:
ggplot(data = data.frame(x = c(-1.2, 1.2),y=c(0,0.2)), aes(x)) +
  stat_function(fun = dnorm, n = 101, args = list(mean = mu0, sd = (k0*(alpha/beta))^(-0.5) )
  stat_function(fun = dnorm, n = 101, args = list(mean = mu, sd = sqrt(lambda^(-1))) ,size=1
  stat_function(fun = dnorm, n = 101, args = list(mean = mean(rrp), sd = (sd(rrp))) , size=1
  scale_colour_manual("Legend", values = c("darkgoldenrod", "chocolate4","black", "darkolive
 scale_y_continuous(breaks = NULL) +
theme(legend.position = c(0.15, 0.85)) +
 labs(y="density",x="x")
```

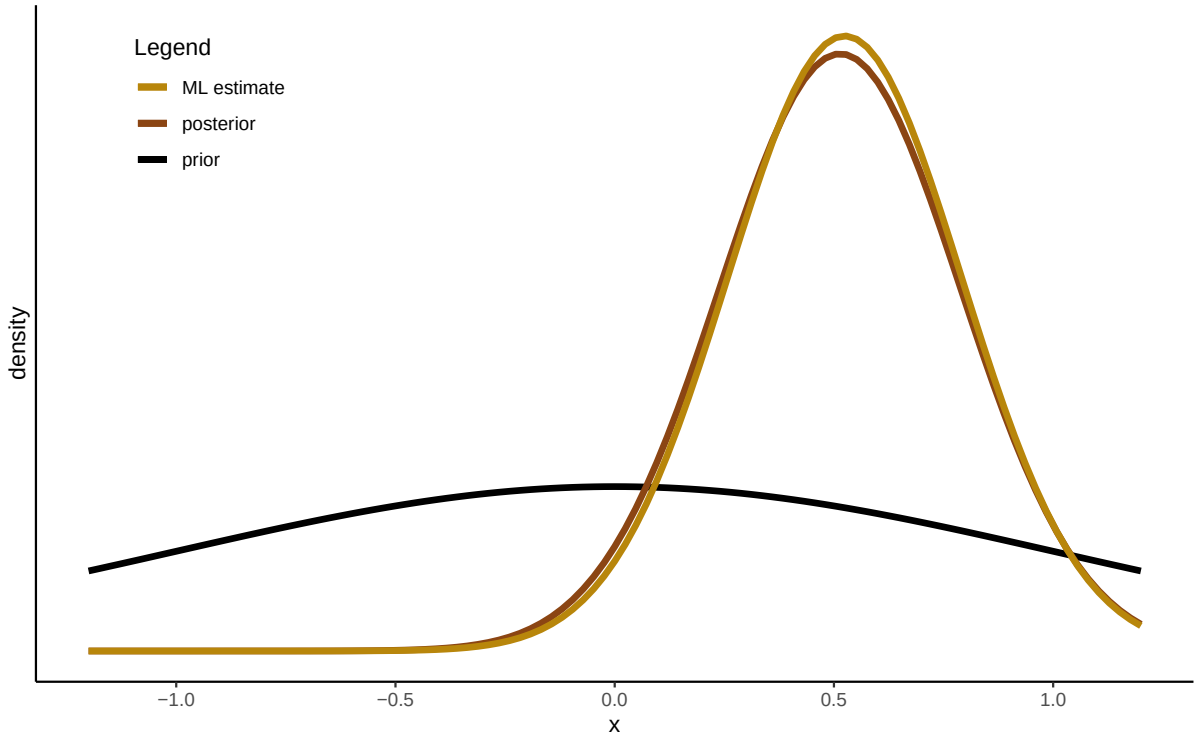


Figure 2.10: Batch updating of normal mean and variance.

Let us start from a prior centered at μ_0, and with $\alpha_0 = \beta_0 = 0.1$, yielding a diffuse uninformative Gamma prior for the precision. We set $\kappa_0 = 1$, given the precision prior a low weight corresponding to just 1 observation. The convergence to the sample distribution, shown in Figure 2.10, is almost complete (notice how I have plotted the learned posterior distribution, instead of the posterior of the mean, which has precision λ_n). In terms of the variance, this is due to the very diffuse albeit uninformative prior. For the mean, however, the low value of κ_0 is crucial. We can think of this as virtual draws from our prior, or in other words, of the confidence we have in the prior itself. It is easy to show that increasing this parameter

will result in a more precise prior, and consequently in increased shrinkage towards the prior mean. Figure 2.11 does just that. By increasing the weight on the prior to $\kappa_0 = 10$, the prior becomes much more precise, and convergence to the likelihood after integrating 50 data points is thus much less complete.

```
rrp <- rnorm(50,0.5,0.316)

n <- length(rrp)
mu0 <- 0
alpha <- 0.01
beta <- 0.01
k0 <- 10

alpha_n <- alpha + n/2
beta_n <- beta + 0.5*sum((rrp - mean(rrp))^2) + (n*k0)/(2*(n + k0)) * (mean(rrp) - mu0)^2
mu <- mu0 + (n)/(n  + k0) * (mean(rrp) - mu0)
k_n = k0 + n
lambda <- alpha_n/beta_n
lambda_n <- k_n*lambda

# learned distribution compared to prior and ML estimate:
ggplot(data = data.frame(x = c(-1.2, 1.2),y=c(0,0.2)), aes(x)) +
  stat_function(fun = dnorm, n = 101, args = list(mean = mu0, sd = (k0*(alpha/beta))^(-0.5) )
  stat_function(fun = dnorm, n = 101, args = list(mean = mu, sd = sqrt(lambda^(-1))) ,
  colour="darkgoldenrod3",size=1.5) +
  stat_function(fun = dnorm, n = 101, args = list(mean = mean(rrp), sd = (sd(rrp))) , colour=
  ylab("")  + scale_y_continuous(breaks = NULL) +
  annotate(geom="text", x=-0.23, y=1.25, label="prior") +
  annotate(geom="text", x=0.47, y=1, label="N=50") +
  annotate(geom="text", x=0.75, y=1.3, label="likelihood")
```

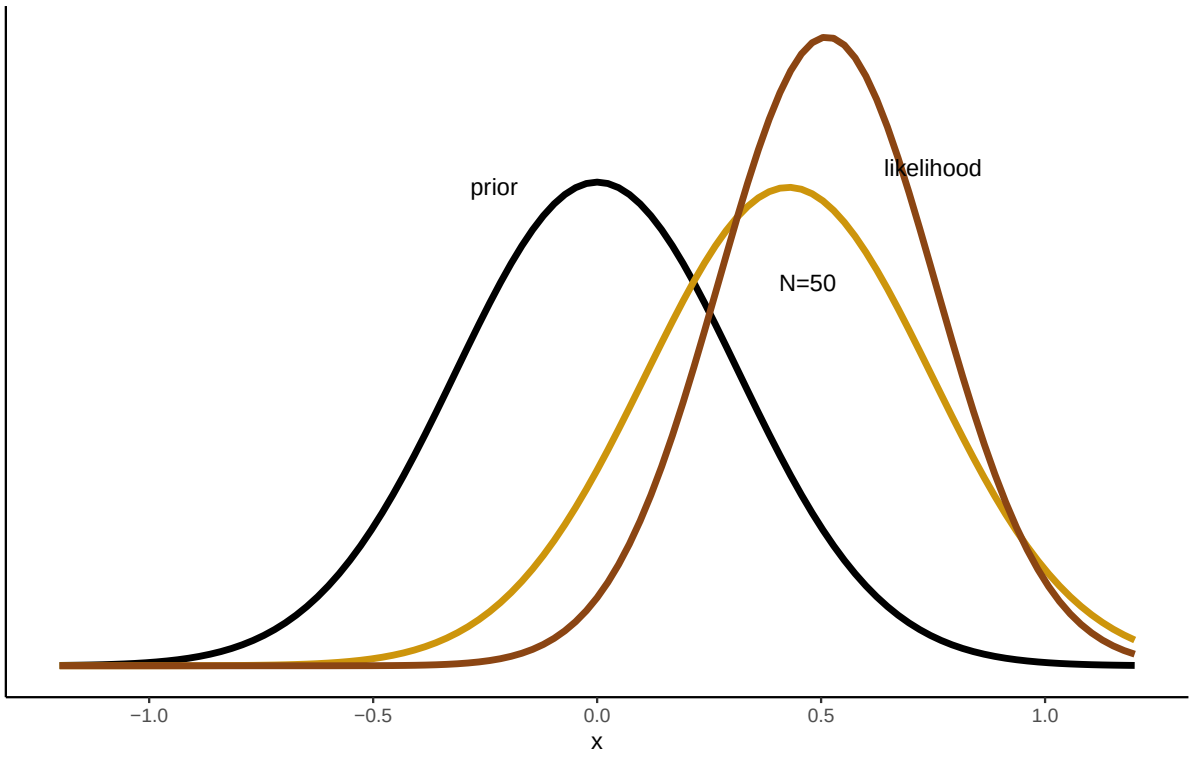


Figure 2.11: Batch updating of normal mean and variance.

```
rrp <- rnorm(50,0.5,0.316)

n <- length(rrp)
mu0 <- 0
alpha <- 3
beta <- 3
k0 <- 1

alpha_n <- alpha + n/2
beta_n <- beta + 0.5*sum((rrp - mean(rrp))^2) + (n*k0)/(2*(n + k0)) * (mean(rrp) - mu0)^2
mu <- mu0 + (n)/(n  + k0) * (mean(rrp) - mu0)
k_n = k0 + n
lambda <- alpha_n/beta_n
lambda_n <- k_n*lambda

# learned distribution compared to prior and ML estimate:
ggplot(data = data.frame(x = c(-1.2, 1.2),y=c(0,0.2)), aes(x)) +
  stat_function(fun = dnorm, n = 101, args = list(mean = mu0, sd = (k0*(alpha/beta))^(-0.5) )
  stat_function(fun = dnorm, n = 101, args = list(mean = mu, sd = sqrt(lambda^(-1))) ,
```

```
  colour="darkgoldenrod3",size=1.5) +
  stat_function(fun = dnorm, n = 101, args = list(mean = mean(rrp), sd = (sd(rrp))) , colour=
  ylab("")  + scale_y_continuous(breaks = NULL) +
  annotate(geom="text", x=-0.8, y=0.4, label="prior") +
  annotate(geom="text", x=0.47, y=1, label="N=50") +
  annotate(geom="text", x=0.75, y=1.3, label="likelihood")
```

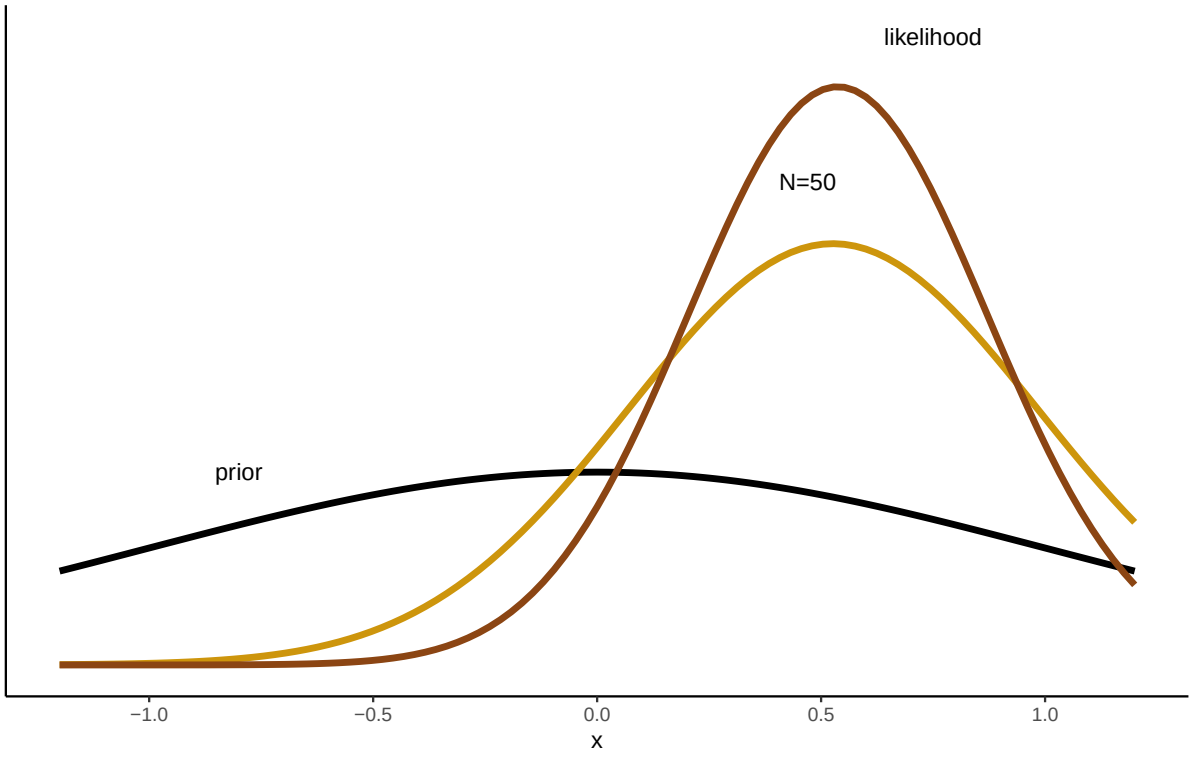


Figure 2.12: Batch updating of normal mean and variance.

It is also interesting to see what happens if one plays around with the parameters of the precision prior. In Figure 2.12 I have set the weight on the prior back to 1, i.e. $\kappa_0 = 1$. I have, however, increased both the shape and rate parameters of the Gamma prior, to $\alpha_0 = \beta_0 = 3$. Note that the mean of the prior precision remains unaffected at $\frac{\alpha_0}{\beta_0} = 3$. What changes, however, is the variance of the precision, which becomes very small at $\frac{\alpha_0}{\beta_0^2} = \frac{1}{3}$, making the prior of the precision, well, more *precise*. We can now see that the posterior precision is shrunk towards the variance of the prior. The effect, shown in the graph, is that the mean still converges to the mean of the data, but we have now much less confidence in this estimate. This is the effect of the strong prior put on the precision, as explored in detail in the previous section, when displayed in the form of the posterior inference on the normal distribution.

2.3.7 Sequential updating

An interesting special case of the equations above obtains in the case of sequential updating with 1 observation at the time, i.e. updating subsequently with every arrival of a new x_i. Notice how only the squared deviation from the prior mean is of importance now, since $x_i = \overline{x}$. The sample variance will thus be made up by a single squared deviation from the prior mean in each case. As the sample mean is updated and learned from observation to observation, we will furthermore need to adjust the scaling parameter, which I will now refer to as κ_i. The updating equations for the prior will thus become:

$$
\begin{aligned}
\alpha_i &= \alpha_{i-1} + \frac{1}{2} \\
\beta_i &= \beta_{i-1} + \frac{1}{2}\frac{\kappa_{i-1}}{\kappa_{i-1}+1}(x_i - \mu_{i-1})^2 \\
\mu_i &= \mu_{i-1} + \frac{1}{\kappa_{i-1}+1}(x_i - \mu_{i-1}) \\
\kappa_i &= \kappa_{i-1} + 1 \\
\tau_i &= \lambda_i + \kappa_i \lambda_i.
\end{aligned}
$$

```
rrp <- rnorm(50,0.5,0.316)
x <- append(rrp,0,0)

alpha <- 0.01
k <- 1
beta <- 0.01
mu <- 0
lambda <- alpha/beta
lambda_m <- alpha/beta

for(i in 2:51){
alpha[i] <- alpha[i-1] + 1/2
beta[i] <- beta[i-1] + (k[i-1])/(2*(k[i-1] + 1)) * (x[i] - mu[i-1])^2
mu[i] <- mu[i-1] + (1)/(1  + k[i-1]) * (x[i] - mu[i-1])
k[i] <- k[i-1] + 1
lambda[i] <- alpha[i]/beta[i]
lambda_m[i] <- lambda[i-1] + k[i-1] * lambda[i-1]
}


# learned distribution compared to prior and ML estimate:
ggplot(data = data.frame(x = c(-1, 1.2)), aes(x)) +
```

```
  stat_function(fun = dnorm, n = 101, args = list(mean = mu[1], sd = sqrt(1/lambda[1])), size
  stat_function(fun = dnorm, n = 101, args = list(mean = mu[2], sd = sqrt(1/lambda[2])) ,size
  stat_function(fun = dnorm, n = 101, args = list(mean = mu[6], sd = sqrt(1/lambda[6])), size
  stat_function(fun = dnorm, n = 101, args = list(mean = mu[11], sd = sqrt(1/lambda[11])), si
  stat_function(fun = dnorm, n = 101, args = list(mean = mu[31], sd = sqrt(1/lambda[31])), si
  geom_vline(xintercept=mean(rrp),linetype="dashed",colour="red", size=1.2) +
  ylab("")  + scale_y_continuous(breaks = NULL) +
  scale_colour_manual("Legend", values = c("black","darkgoldenrod", "chocolate4", "darkolive
```

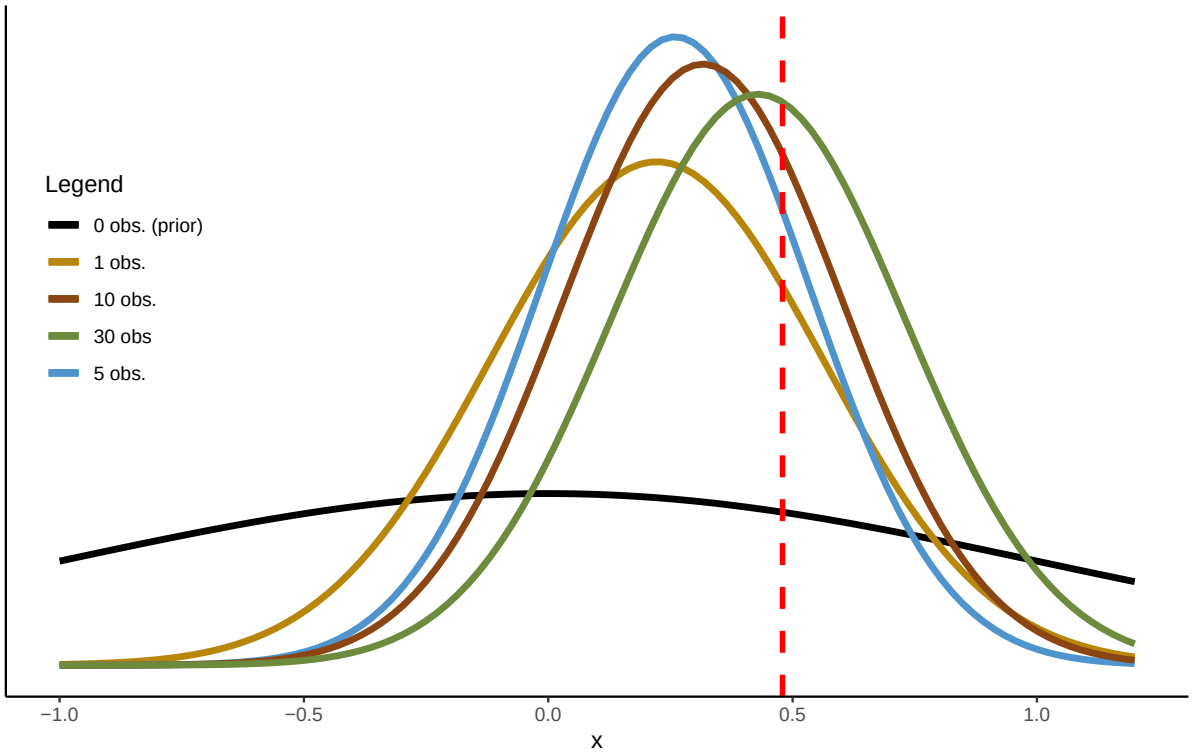


Figure 2.13: Sequential updating of normal mean and variance.

Figure 2.13 shows the sequential updating starting from a diffuse noninformative prior. After about 30 observations, both the mean and variance are (almost) perfectly learned under this parametrization. Alternatively, we can take a look at the posterior for the mean, i.e. incorporating the precision of the mean instead of the sampling distribution, shown in Figure 2.14. The mean estimate converges to the sampling estimate fairly quickly, just like shown above. The major difference now is that the precision surrounding the mean becomes ever larger, and the distribution hence more concentrated. That is indeed what should happen—as we add observations, our confidence in the mean estimate will grow progressively larger.

```r
ggplot(data = data.frame(x = c(-0.8, 0.8)), aes(x)) +
  stat_function(fun = dnorm, n = 101, args = list(mean = mu[1], sd = sqrt(1/lambda_m[1])), si
  stat_function(fun = dnorm, n = 101, args = list(mean = mu[2], sd = sqrt(1/lambda_m[2])) ,si
  stat_function(fun = dnorm, n = 101, args = list(mean = mu[6], sd = sqrt(1/lambda_m[6])), si
  stat_function(fun = dnorm, n = 101, args = list(mean = mu[11], sd = sqrt(1/lambda_m[11])),
  stat_function(fun = dnorm, n = 101, args = list(mean = mu[31], sd = sqrt(1/lambda_m[31])),
  geom_vline(xintercept=mean(rrp),linetype="dashed",colour="red", size=1.2) +
  ylab("")  + scale_y_continuous(breaks = NULL) +
  scale_colour_manual("Legend", values = c("black","darkgoldenrod", "chocolate4", "darkolive
```

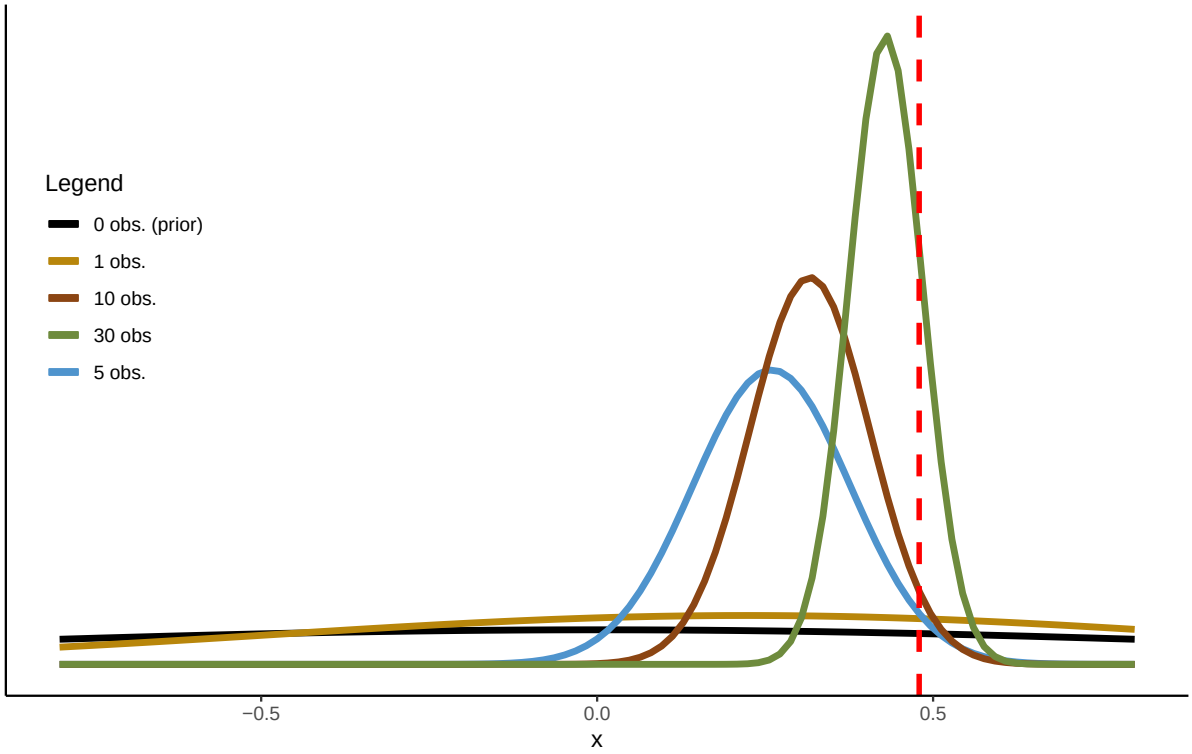


Figure 2.14: Sequential updating of normal mean and precision.

2.3.8 An alternative parametrization

We have now seen how to obtain the mean and variance by modelling the mean conditional on the precision. In some cases, however, it may be convenient to work with the variance directly. This could be done using an inverse-Gamma distribution. A more intuitive parametrization, however, can be obtained by means of a scaled inverse-Chi squared, which I will write χ^{-2}. Once again, we will factorize the joint prior distribution, $p(\mu, \sigma^2) = p(\sigma^2)p(\mu|\sigma^2)$. The two distributions take the following form:

$$p(\mu|\mu_0, \sigma^2/\kappa_0) = \mathcal{N}\left(\mu_0, \frac{\sigma^2}{\kappa_0}\right)$$
$$p(\sigma^2/\nu_0, \sigma_0) = \chi^{-2}(\nu_0, \sigma_0^2).$$

The basic setup is thus similar to the one seen above. The main difference consists in the use of the Chi-2, which is parameterized using degrees of freedom ν_0 and the variance of the prior σ_0^2.

Assume you observe n data points $x = (x_1, ..., x_n)$. Then the posterior distribution will again take the form of a normal-Chi-square distribution, $N\chi^{-2}(\mu, \sigma^2|\mu_n, \sigma_n^2/\kappa_n, \nu_n, \sigma_n^2)$. The updating equations will look as follows:

$$\begin{aligned}
\mu_n &= \mu_0 + \frac{n}{\kappa_0 + n}(\overline{x} - \mu_0) \\
\kappa_n &= \kappa_0 + n \\
\nu_n &= \nu_0 + n \\
\sigma_n^2 &= \frac{1}{\nu_n}\left(\nu_0\sigma_0^2 + \sum_i (x_i - \overline{x})^2 + \frac{\kappa_0 n}{\kappa_0 + n}(\overline{x} - \mu_0)^2\right), \\
\tau_i &= \frac{\nu_0}{\sigma_0^2} + \frac{\kappa_0}{\sigma_0^2}
\end{aligned}$$

where the posterior degrees of freedom are thus given by the prior degrees of freedom plus the sample size. It is left as an exercise to show that this parametrization produces results identical to those obtained above based on the normal-gamma prior. Results based on sequential updating are shown in Figure 2.15.

```
rrp <- rnorm(50,0.5,0.316)

k0 <- 1
nu0 <- 1
mu0 <- 0
sigma0 <- 0.5

n <- length(rrp)
sm <- mean(rrp)
sq <- sum((rrp - sm)^2)

mu <- mu0 + (n/(n+k0))*(sm - mu0)
k <- k0 + n
nu <- nu0 + n
sigma <- 1/nu * (nu0*sigma0 + sq + (k0*n/(k0+n)) * (sm - mu0)^2 )
```

```
tau <- nu0/sigma0 + k0/sigma0


ggplot(data = data.frame(x = c(-1.2, 1.2),y=c(0,0.2)), aes(x)) +
  stat_function(fun = dnorm, n = 101, args = list(mean = mu0, sd = (sigma0/k0) ), size=1.5) +
  stat_function(fun = dnorm, n = 101, args = list(mean = mu, sd = sqrt(sigma)) ,
  colour="darkgoldenrod3",size=1.5) +
  stat_function(fun = dnorm, n = 101, args = list(mean = mean(rrp), sd = (sd(rrp))) , colour=
  ylab("")  + scale_y_continuous(breaks = NULL) +
  annotate(geom="text", x=-0.8, y=0.4, label="prior") +
  annotate(geom="text", x=0.47, y=1, label="N=50") +
  annotate(geom="text", x=0.75, y=1.3, label="likelihood")
```

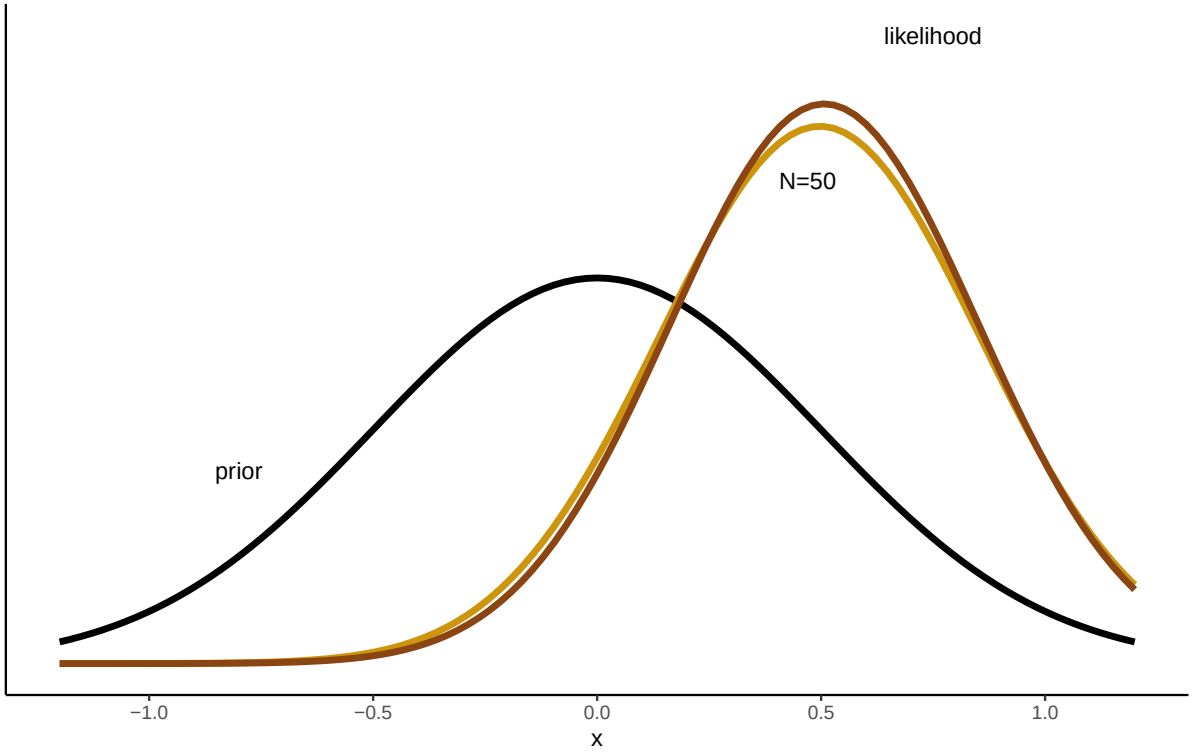


Figure 2.15: Sequential updating of normal mean and precision.

2.4 Conclusion

We have now examined some simple cases of conjugate updating. While more general analysis will require the use of simulation algorithms, these simple cases nevertheless help developing

intuition for some of the fundamental processes at work. They are also very useful if one wants to use back-of-the-envelop calculations to obtain quick but exact estimates of the posterior under different assumptions about the prior, albeit at the cost of having to assume specific distributional forms.

3 Aggregate estimation in Stan

In this chapter, we will cover the basics of estimation in Stan. In particular, this chapter will start from a basic linear regression in Stan, and then quickly move to more specilized topics in nonlinear estimation. While in practice you are unlikely to want to use Stan for the type of aggregate- or individual-level estimation discussed in this chapter, it constitutes an important stepping stone in familiarizing yourself with Stan and Bayesian estimation more generally. Aggregate models also constitute an important point of reference to which more complex hierarchical models—covered in chapter III—can be compared. Indeed, it is always a good idea to start from a simple model, and to then increase its complexity step-by-step towards your preferred specification. Although it may sound paradoxical, this can save you a lot of time in the end, especially when highly nonlinear hierarchical models on largish datasets may run for hours or days on end.

3.1 Basics of Stan coding

3.1.1 About Stan

Stan is a very versatile programme for Bayesian analysis written in C++. It can be launched from a number of different platforms, including *R* and the command prompt. Analysis files are written in text editors, and the programmes need to be compiled before they can be run. Always keep the Stan user manual close by to look up the best way of implementing a programme https://mc-stan.org/users/documentation/). Also consult the online resources on implemented functions https://mc-stan.org/docs/2_21/functions-reference/), and if need be, the user community website https://discourse.mc-stan.org/).

At the time of writing, there are two ways of launching Stan from R. Rstan works like a standard R package. CmdStanR works as an R interface to communicate with the terminal. There are pros and cons to each method. Rstan will look more familiar to R users, especially when it comes to post-processing of estimation results. It also makes it easier to set and control starting values, and the experience is smoother, in that Rstan tends to throw fewer (and more meaningful) errors. CmdStanR, however, has the advantage that it is quite a bit faster. It is also much lighter in terms of memory use—a decisive quality if you want to estimate large models, which in Rstan frequently lead to crashes. While at the time of writing there are other differences between the two modes of operations, these are subject to rapid change, and are thus not described here. In this tutorial, I will use CmdStanR

throughout. You can find detailed instructions on the installation and a guide to get you started at https://mc-stan.org/cmdstanr/articles/cmdstanr.html

3.1.2 The structure of a Stan model

The basic structure of a Stan model looks as follows:

```
functions {
  // ... self-defined functions (optional) ...
}
data {
  // ... all data need to be specified ...
}
transformed data {
   // ... data can be transformed here (optional) ...
}
parameters {
   // ... declare all parameters to be used ...
}
transformed parameters {
   // ... transform parameters (optional) ...
}
model {
   // ... define priors and write model ...
}
generated quantities {
   // ... post-estimation commands (optional) ...
}
```

The `data` block, the `parameters` block, and the `model` block are compulsory. The other blocks are optional, but will often turn out to be useful. We will explore the use of different statements within the blocks progressively as we move through these notes. The different blocks present some overlaps. For instance, one can declare variables in the `parameters` block, the `transformed parameters` block, in the `model block`, or even the `generated quantities` block. However, variables declared in the model block are only available locally in that block, and variables declared in `generated quantities` are not available in any earlier blocks. Variables declared in the `parameters` or in the `transformed parameters` block, on the other hand, are available globally. E.g., they can be used directly in the model block without the need to declare them again.

3.2 Linear regression

3.2.1 A simple regression

In this section, we will examine a simple linear regression model, and learn how to programme it in Stan. Take the following equation:

$$y_n = \alpha + \beta x_n + \epsilon_n,$$

where n indicates a single observation, and $\epsilon_n \sim \mathcal{N}(0, \sigma)$ (NOTE: I follow the convention in Stan to write the *standard deviation* in the distribution, instead of the variance). This equation can be rewritten as follows:

$$y_n - (\alpha + \beta x_n) \sim \mathcal{N}(0, \sigma),$$

or indeed as

$$y_n \sim \mathcal{N}(\alpha + \beta x_n, \sigma).$$

The latter form is typically used to code regression models in Stan.

To make the problem more interesting, let $y = ce$ be a certainty equivalent, and $x = p$ the probability of winning provided by the wager. The following implements a Stan model regressing the certainty equivalent on the probability of winning the prize:

```
data{
  int<lower=0> N;
  vector[N] ce;
  vector[N] p;
}
parameters{
  real alpha;
  real beta;
  real<lower=0> sigma;
}
model{
  ce ~ normal(alpha + beta * p , sigma) ;
}
```

A few comments are in place. The statement `<lower = 0>` indicates that a variable is limited below at 0. This is optional for data, where it serves mostly as a check on the data one imports. It is essential for some variables, such as σ in the example above, since the programme may otherwise attempt to sample negative values. Certain data need to be coded as integers, or else they will be rejected by Stan. For instance, this always applies to the sample size N, and we will see other examples in due time. The data are imported as vectors, which—where possible—has computational advantages and speeds up the sampling. Alternatively, the model line could be rewritten using a loop:

```
model{
for (n in 1:N)
  ce[n] ~ normal(alpha + beta * p[n] , sigma) ;
}
```

The latter way of writing the model is slightly slower. However, it provides additional flexibility in contexts where matrix notation cannot be used, e.g. because of the use of powers, or because of difficulties in matching the dimensions of parameter and data matrices in hierarchical models. Finally, you should notice that I have not specified any priors. This means that Stan will automatically set the prior for the parameters to a uniform distribution over the whole support of the parameters. We willll return to priors later.

Let us now go ahead and estimate the model. We can use the data of Bouchouicha and Vieider (2017), experiment 1. We will start by creating a new dataset to send to Stan. This helps because Stan hates missing values, even in variables it does not use. It is also a chance to rename and transform variables to get them into the appropriate format. In this instance, it is convenient to normalize the CE by dividing it by the prize of the wager (try running it without the normalization to convince yourself of this):

```
bv <- read.csv(file="data/data_BV_exp1.csv")

# creates data to send to Stan
stanD <- list(N = nrow(bv),
              ce = bv$ce/bv$high,
              p = bv$prob )

# compile the Stan programme:
sr <- cmdstan_model("stancode/agg_normal.stan")

# run the programme:
nm <- sr$sample(
  data = stanD,
  seed = 123,
  chains = 4,
  parallel_chains = 4,
  refresh=0,
  init=0,
  show_messages = TRUE,
  diagnostics = c("divergences", "treedepth", "ebfmi")
)
```

```
print(nm, c("alpha","beta","sigma"),digits = 3)
nm$save_object(file="stanoutput/agg_normal.Rds")
```

The coefficient $\beta = 0.65$ can be interpreted as the slope of a neoadditive weighting function, $w(p) = \alpha + \beta p$, capturing likelihood-insensitivity; the intercept can proxy for optimism (see Abdellaoui et al. (2011) for more appropriate indices, such as $1 - \beta - 2 * \alpha$). The `sd` is the standard deviation of the posterior simulations, which is equivalent to a standard *error* in frequentist statistics, except that we can directly conclude that β has a 90% probability of falling between 0.627 and 0.681.

It may be interesting to determine the proportion of the overall variance in CEs explained by this simple model. One complication, however, is that the output contains the standard deviation of the error instead of the R^2. This is easily solved using the definition of R^2:

$$R^2 = 1 - \frac{\sigma_1^2}{\sigma_0^2} = 1 - \frac{0.163^2}{0.0769} = 0.655,$$

where σ_0^2 is the variance of the model empty of covariates. In this case, I have obtained this measure simply from the data by using `var(bv$ce_norm)`, and σ_1^2 is the variance of the model estimated above. Technically, one could obtain σ_0 from estimating a model empty of co-variates, and then derive an R^2 measure that is itself an uncertain quantity. While properly Bayesian, this is rarely done in practice since it produces informational overload.

One can now use the posterior simulations obtained from the model in a variety of ways:

```
nm <- readRDS(file="stanoutput/agg_normal.Rds")
p <- as_draws_df(nm)

ggplot() +
  geom_line(aes(x = p$beta), stat="density",colour="darkgoldenrod", size=1.2) +
  geom_line(aes(x=c(quantile(p$beta,0.025),quantile(p$beta,0.975)),y=c(2,2)),colour="blue", s
  labs(x = "likelihood-sensitivity") +
  annotate("text", x=0.65, y=2.8, label= "95% credibility interval")
```

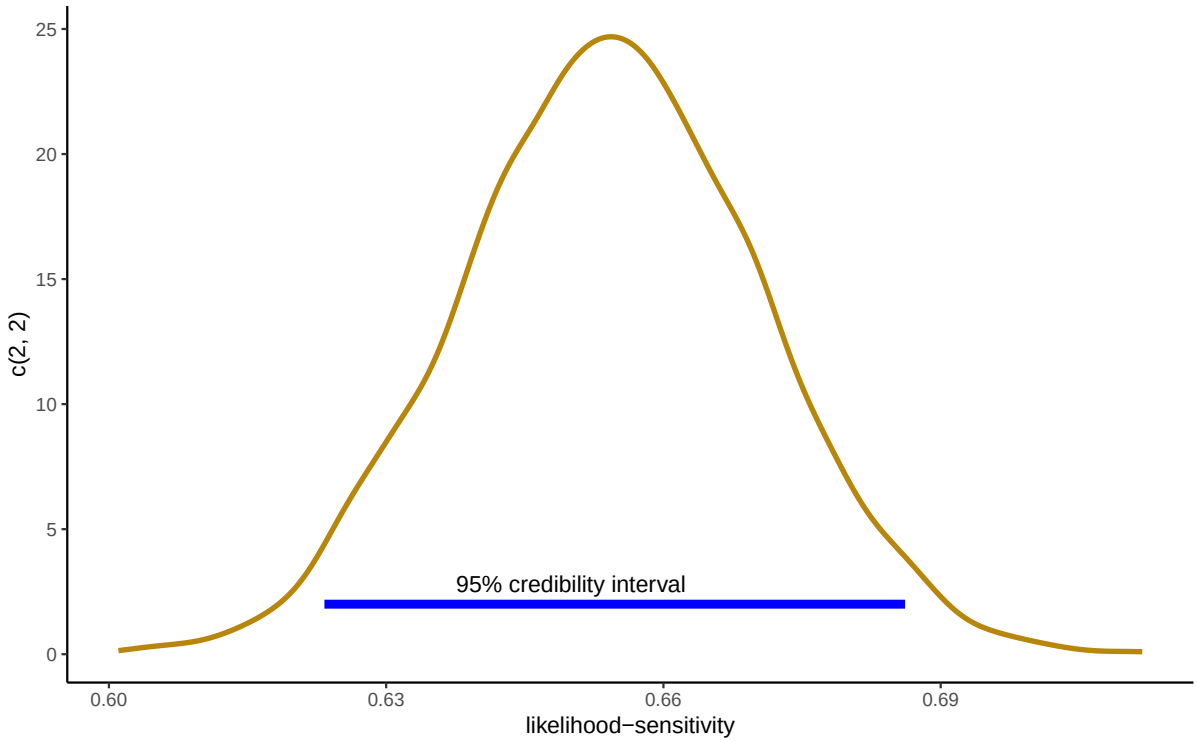


Figure 3.1: Slope parameter of linear regression of ce/x on p

The distribution in Figure 3.1 represents all the posterior draws of the parameter β, which I refer to as *likelihood-sensitivity*. Note that in the Bayesian interpretation, it is indeed the *parameter* that is considered uncertain, while the data are treated as given. The blue line gives the 95% credibility interval (which in the symmetric case coincides with the 95% highest density interval), which gives the probability that the true parameter falls within the indicated range.

3.2.2 Multiple regression

We have so far regressed our CEs only on one single variable. It is, however, straightforward to extend this setup to multiple variables, e.g. by using the following code:

```
model{
ce ~ normal(alpha + beta_p * p + beta_h * high , sigma) ;
}
```

We now have expanded the code to include two independent variables (don't forget to import the new variable in the `data` block, and to add the new parameter in the `parameters` block).

However, one can see how this sort of coding can get old quickly—for models with many independent variables, we will need to code a new model for every single specification we intend to use! A better way to go about it will thus be to code the regression coefficients as a vector. That is easily achieved with the following code:

```
data{
int<lower=0> N;
int<lower=0> k;
vector[N] ce;
matrix[N,k] x;
}
parameters{
vector[k] beta;
real<lower=0> sigma;
}
model{
ce ~ normal(x * beta , sigma) ;
}
```

I am now declaring an additional data point k, which gives me the columns of the design matrix x. The design matrix includes a column of 1s, so that I have furthermore dropped the intercept parameter α, with the first coefficient in β taking its place. Given that the model is written in matrix notation, β needs to be post-multiplied with the design matrix.

```
x <- model.matrix(~prob + high, data = bv)

# creates data to send to Stan
stanD <- list(N = nrow(bv),
              k = ncol(x),
              ce = bv$ce/bv$high,
              x = x)

# compile the Stan programme:
sr <- cmdstan_model("stancode/agg_normal_multi.stan")

# run programme:
nm <- sr$sample(
  data = stanD,
  seed = 123,
  chains = 4,
  parallel_chains = 4,
  refresh=0,
```

```
  init=0,
  show_messages = TRUE,
  diagnostics = c("divergences", "treedepth", "ebfmi")
)

print(nm, c("beta","sigma"),digits = 3)
nm$save_object(file="stanoutput/agg_normal_multi.Rds")
```

The coefficients obtained for probability and prize are difficult to compare directly, given the different scale of the prize. However, an interesting question may be how much more variance is explained once we add the prize over the regression on probabilities alone. The answer is *not much*. The total R^2 of the model is now 0.6754313, which is not much higher than the one of the simple model above. This is the sense in which Bouchouicha and Vieider (2017) conclude that likelihood-distortions are much more important than outcome-distortions to account for behaviour!

3.2.3 A linearized dual-EU model

We have started making some inferences on the goodness of fit of different models using linear regression. We can, however, also estimate linearized versions of nonlinear models using linear regression techniques. For $u(x) = x^r$, we can rewrite the EU model $u(c) = pu(x)$ as $c = u^{-1}(p)x$, which gives us a dual-EU model. We know that risk attitudes over probabilities tend to change rather radically as one moves from small to large probabilities, certainly when using CEs. A function that has often been used to capture this pattern is as follows:

$$\pi(p) = \frac{\delta p^\gamma}{\delta p^\gamma + (1-p)^\gamma}$$

The function is highly non-linear. It can, however, be reformulated in terms of log-odds as follows:

$$ln\left(\frac{\pi(p)}{1-\pi(p)}\right) = ln(\delta) + \gamma\, ln\left(\frac{p}{1-p}\right)$$

Note also that under dual-EU $\pi(p) \triangleq \frac{ce-y}{x-y}$. This means that the left-hand side is fully defined in terms of the data. The right hand side has two parameters. Since these happen to be simply a slope and an intercept, we can estimate them using a simple linear regression.

```
data{
int<lower=0> N;
vector[N] ce;
vector[N] p;
vector[N] x;
}
```

```
transformed data{
pi = ce/x;
dv = log(pi/(1-pi));
lo = log(p/(1-p));
}
parameters{
real alpha;
real<lower 0> gamma;
real<lower=0> sigma;
}
model{
dv ~ normal(alpha + gamma * lo , sigma) ;
}
generated quantities{
real delta;
delta = exp(alpha)
}
```

In the code above, I use the `transformed data` block to obtain the measures of interest (alternatively, I could have created them in `R` and declared them as data). I also use the `generated quantities` block to recover δ—the true quantity of interest, giving us optimism—–from the estimated intercept $\alpha = ln(\delta)$, so that $\delta = exp(\alpha)$. We can easily recover this parameter from the posterior including all the parameter uncertainty (note that this could also be done post-processing in R if need be). This is indeed a strength of the Bayesian approach: we can manipulate parameters in the posterior as we want, all the while carrying along all and any parameter uncertainty. Executing the programme and recovering and manipulating these quantities is left for your as an exercise.

3.3 Refinement of regression models

3.3.1 Robust regression

We have so far used the normal density in regressions, but nothing is special about the normal density. That is, we can simply replace that density by an alternative density of our choice. A fitting choice may be the Student-t density, which due to its fat tails is often used to implement *robust* regression. That is, the fat tails allow the distribution to discount outliers, thus making the results more robust to outlying observations. The following code implements such a regression with 2 degrees of freedom for the Student-t distribution:

```
data{
int<lower=0> N;
int<lower=0> k;
vector[N] ce;
matrix[N,k] x;
}
parameters{
vector[k] beta;
real<lower=0> sigma;
}
model{
ce ~ student_t(2 , x * beta , sigma) ;
}
```

And running the code in Stan:

```
x <- model.matrix(~ prob + high, data = bv)

# creates data to send to Stan
stanD <- list(N = nrow(bv),
              k = ncol(x),
              ce = bv$ce/bv$high,
              x = x)

# compile the Stan programme:
sr <- cmdstan_model("stancode/agg_student.stan")

# run programme:
nm <- sr$sample(
  data = stanD,
  seed = 123,
  chains = 4,
  parallel_chains = 4,
  refresh=0,
  init=0,
  show_messages = TRUE,
  diagnostics = c("divergences", "treedepth", "ebfmi")
)

print(nm, c("beta","sigma"),digits = 3)
nm$save_object(file="stanoutput/agg_student.Rds")
```

The parameters have changed rather significantly, with a lower intercept and larger slope/sensitivity. We can, indeed, test for this difference:

```
nm <- readRDS("stanoutput/agg_normal_multi.Rds")
st <- readRDS("stanoutput/agg_student.Rds")
n <- as_draws_df(nm)
s <- as_draws_df(st)

diff <- s$`beta[2]` - n$`beta[2]`

ggplot() +
  geom_line(aes(x = diff), stat="density",colour="darkgoldenrod", size=1.2) +
  geom_vline(xintercept=0,colour="red") +
  labs(x = "likelihood-sensitivity difference")
```

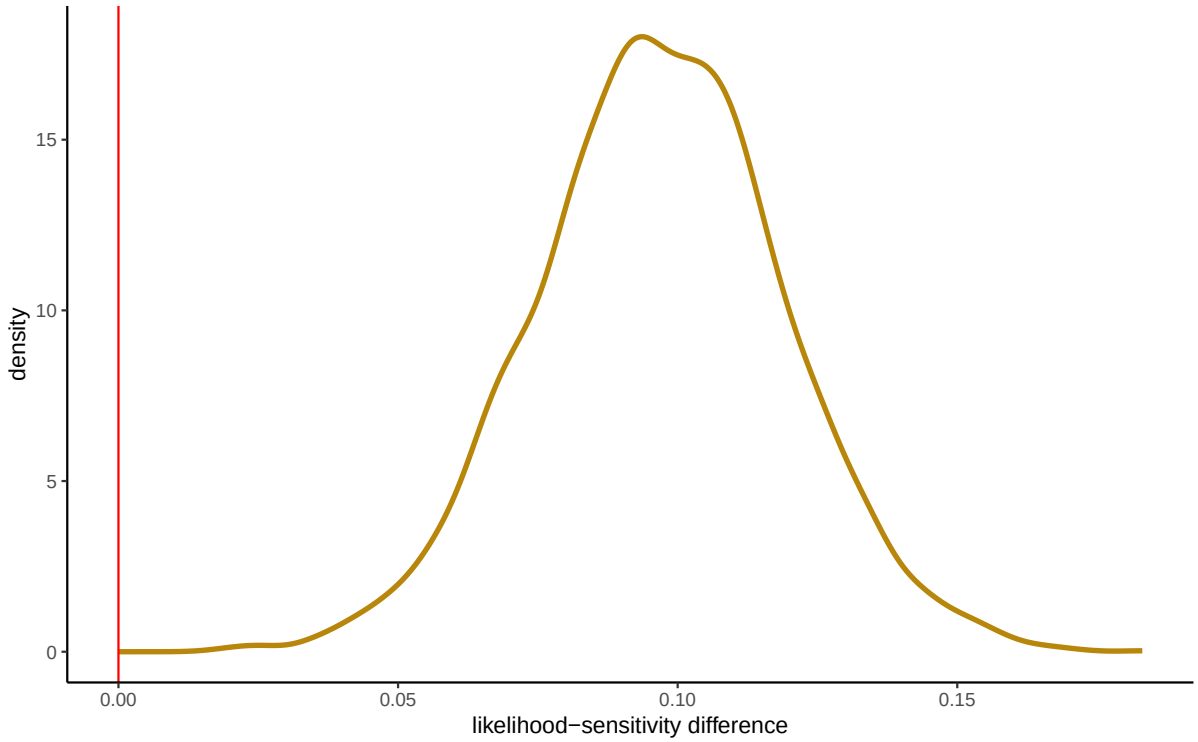


Figure 3.2: Slope difference from normal verus student-t distribution

The test for the difference is technically carried out by taking the difference of all the observations in the posterior simulations. One can then compare this difference to a benchmark of 0 (or some interval around 0 that can be justified as an acceptability region), and quantify how much of the probability mass lines up with the hypothesis. This is shown in Figure 3.2. In this

case, virtually all the probability mass falls above 0, showing a significant difference between the two estimates. To test this, once could simply look at the probability mass falling above 0 (or above 0.03 or 0.05, depending on how one wants to conceive of the null hypothesis).

Differences in the estimates are not truly what we are interested in here. One could now go ahead and directly test for *model performance* (we will discuss how to do this at a later point). Another possibility may be to construct a model where the degrees of freedom are estimated endogenously. If outliers are indeed a problem, the degrees of freedom ought to be estimated to some low number. If this is not the case, we will get large degrees of freedom, and the distribution will approach the normal distribution (at infinity in theory, but starting already from about 10 for all practical purposes; see Kruschke (2013) for details). Here is the code implementing such a model:

```
data{
int<lower=0> N;
int<lower=0> k;
vector[N] ce;
matrix[N,k] x;
}
parameters{
vector[k] beta;
real<lower=0> sigma;
real<lower=0> df;
}
model{
ce ~ student_t(df , x * beta , sigma) ;
}
```

And the code used to estimate it:

```
x <- model.matrix(~prob + high, data = bv)

# creates data to send to Stan
stanD <- list(N = nrow(bv),
              k = ncol(x),
              ce = bv$ce/bv$high,
              x = x)

# compile the Stan programme:
sr <- cmdstan_model("stancode/agg_student_end.stan")

# run the programme:
nm <- sr$sample(
```

```
  data = stanD,
  seed = 123,
  chains = 4,
  parallel_chains = 4,
  refresh=0,
  init=0,
  show_messages = TRUE,
  diagnostics = c("divergences", "treedepth", "ebfmi")
)

print(nm, c("df","beta","sigma"),digits = 3)
saveRDS(nm,file="stanoutput/agg_student_end.Rds")
```

The degrees of freedom are estimated at 5, thus falling into a range where the student-t distribution differs substantially from the normal distribution. This vindicates the use of the student-t, and suggests that it would indeed perform better than a normal distribution in out-of-sample prediction. We will leave a test of this statement for a later chapter. Notice, however, that it is relatively low cost to move to a student-t distribution. This only costs one more parameter, which for most datasets is not an issue at the aggregate level. If the underlying distribution is truly normal, the student-t will reflect this automatically by estimating large degrees of freedom. If it is not, we have built a model that is resistant to outliers. Win-win.

3.3.2 Reparameterizing the model

Although mathematically equivalent, different ways of parameterizing a model can have a large impact on the convergence of a model in Stan. The details of this depend on how the Hamiltonian Monte Carlo simulation algorithm explores the parameter space. This is an issue to which we will have to return repeatedly in these notes. Here, I specifically discuss reparameterizing the regression matrix and coefficients. Although in the simple model above we did not encounter any problems, issues will quickly arise when we move to more complex models. This is especially true when several regression variables are (partially) colinear, which creates difficulties for the algorithm in exploring the parameter space. The solution will thus involve transforming the independent variables in a way as to be orthogonal.

For regression analysis, we will want to decompose the design matrix as follows: $x = QR$, where Q is an orthogonal matrix, and R an upper triangular matrix, referred to, rather fittingly, as *Q-R decomposition*. Major commercial software typically performs this decomposition *under the hood*. Luckily, these days Stan allows us to directly implement this within the programme, without the need to first do the decomposition by hand in R, and later reconvert the parameters to the original scale. Most of the action happens within the transformed data block, with the model part integrating the regression on the orthogonal matrix. The generated quantities part is then used to transform the regression parameters back to the original scale:

```
data {
int<lower=0> N;
int<lower=0> k;
matrix[N, k] x;
vector[N] ce;
}
transformed data {
matrix[N, k] Q_ast;
matrix[k, k] R_ast;
matrix[k, k] R_ast_inverse;
// thin and scale the QR decomposition:
Q_ast = qr_thin_Q(x) * sqrt(N - 1);
R_ast = qr_thin_R(x) / sqrt(N - 1);
R_ast_inverse = inverse(R_ast);
}
parameters {
vector[k] gamma; // coefficients on Q_ast
real<lower=0> sigma; // error scale
}
model {
ce ~ normal(Q_ast * gamma , sigma); // likelihood
}
generated quantities {
vector[k] beta;
beta = R_ast_inverse * gamma; // coefficients on x
}
```

You can convince yourself that the vector `beta` contains estimates identical to those obtained previously. Although in the present setup this does not (yet) make a difference, it is good practice to code any regression using the Q-R decomposition from the start. Otherwise, you risk to get stuck at some point without knowing why. Your estimations may also run much more slowly, which for large and complex models can become a real problem.

3.3.3 Introducing priors

Let us again use the example above. We can formulate priors as follows:

```
data{
int<lower=0> N;
int<lower=0> k;
vector[N] ce;
```

```
matrix[N,k] x;
}
parameters{
vector[k] beta;
real<lower=0> sigma;
}
model{
// priors:
beta[1] ~ normal(0,0.5);
beta[2] ~ normal(0.7,0.25);
sigma ~ normal(0,1);
//likelihood:
ce ~ normal(x * beta , sigma) ;
}
```

The parameter `beta[1]` represents the intercept in our regression model. The prior mean at 0 is chosen to be uninformative; the SD is chosen so as to give the parameter a wide berth—given what we know about the quantities in the model, we would typically not expect it to be much above 0.3 or below -0.3, yet we have attributed substantial probability mass to the range from -1 to 1. Such priors are called *mildly regularizing*, inasmuch as they help the estimation algorithm to search in roughly the right range without imposing any strong prior opinions of where we would expect the estimates to fall.

The prior for `beta[2]` has been chosen to be somewhat more informative. We know from a large literature on likelihood-insensitivity that the parameter will typically fall below 1, and typical values may be between 0.6 and 0.8 (given a meta-analysis, one could use the posterior from the analysis as the prior for new estimations). At the same time, the SD is chosen to be somewhat more restrictive, while still providing enough wiggle room not to influence the final estimates too much. (Of course, you should always play around with the prior to make sure this is the case).

A more interesting question concerns what may happen if we choose a prior that is wildly inappropriate. Imagine we take a prior with mean 2 and SD 0.1, giving a negligible probability to the maximum likelihood estimate. This results in an estimate of 0.736 instead of 0.686. A distortion to be sure, but not a huge one, considering how far away and how overconfident the prior is. The reason is simply that the amount of data is easily enough to largely overwrite even a relatively strong and wildly inappropriate prior (i.e., the data will typically produce estimates with enough precision to counteract the narrow variance of the prior). As a general rule, the less data one has, the more impact the prior will have—for good or for bad. Priors should thus be chosen wisely, and one should examine the stability of the model to several assumptions about the prior to make sure that it does not unduly distort the results.

3.3.4 Informative priors on sparse stimuli

It may be interesting to see what happens when there are relatively little data to work with. It is in these cases that priors will unfold their true power. This may well be a bad thing, as happens when priors unduly influence the posterior estimates one obtains, possibly even without the analyst being aware of it. To see what might happen, we can run the same model as above on a much reduced dataset. For instance, let us select only one subject at random:

```
bv5 <-  bv %>% filter(subject==5)

x <- model.matrix(~ prob + high, data = bv)

# creates data to send to Stan
stanD <- list(N = nrow(bv5),
              ce = bv5$ce/bv5$high,
              p = bv5$prob)

# compile the Stan programme:
sr <- cmdstan_model("stancode/agg_normal.stan")

# run programme:
nm <- sr$sample(
  data = stanD,
  seed = 123,
  chains = 4,
  parallel_chains = 4,
  refresh=0,
  init=0,
  show_messages = TRUE,
  diagnostics = c("divergences", "treedepth", "ebfmi")
)

print(nm, c("alpha","beta","sigma"),digits = 3)
saveRDS(nm,file="stanoutput/agg_normal_s5.Rds")

# run again with strong prior
sr <- cmdstan_model("stancode/agg_normal_dp.stan")

nm <- sr$sample(
  data = stanD,
  seed = 123,
  chains = 4,
  parallel_chains = 4,
```

```
  refresh=0,
  init=0,
  show_messages = TRUE,
  diagnostics = c("divergences", "treedepth", "ebfmi")
)

print(nm, c("alpha","beta","sigma"),digits = 3)
saveRDS(nm,file="stanoutput/agg_normal_s5dp.Rds")
```

The slope parameter for this particular subject is 0.75—quite typical (estimated based on a regularizing prior with mean 0.7, and a SD of 0.5). The standard error associated with the parameter, however, is quite large at 0.078, indicating that the parameter could fall anywhere between 0.59612 and 0.90188. We can see the consequence of this if we again impose the outlandish prior with mean 2 and SD 0.1. The slope is now estimated at 1.892—a far cry from the original estimate. Figure 3.3 shows what is happening.

```
s5 <- readRDS("stanoutput/agg_normal_s5.Rda")
s5dp <- readRDS("stanoutput/agg_normal_s5dp.Rda")
n <- as_draws_df(s5)
dp <- as_draws_df(s5dp)
beta_n <- n$beta
prior_n <- rnorm(4000, mean=0.7, sd=1)
beta_dp <- dp$beta
prior_dp <- rnorm(4000, mean=2, sd=0.1)

ggplot() +
  geom_line(aes(x=beta_n,color="slope, improper prior"), stat="density", size=1.2) +
  geom_line(aes(x=beta_dp,color="slope, strong prior"), stat="density", size=1.2) +
  geom_line(aes(x=prior_dp,color="prior"), stat="density", size=1.2) +
  labs(x="slope parameter (lkelihood sensitivity)") +
  scale_colour_manual("Legend", values = c("black","darkgoldenrod","steelblue3","#999999")) +
  theme(legend.position = c(0.5, 0.8))
```

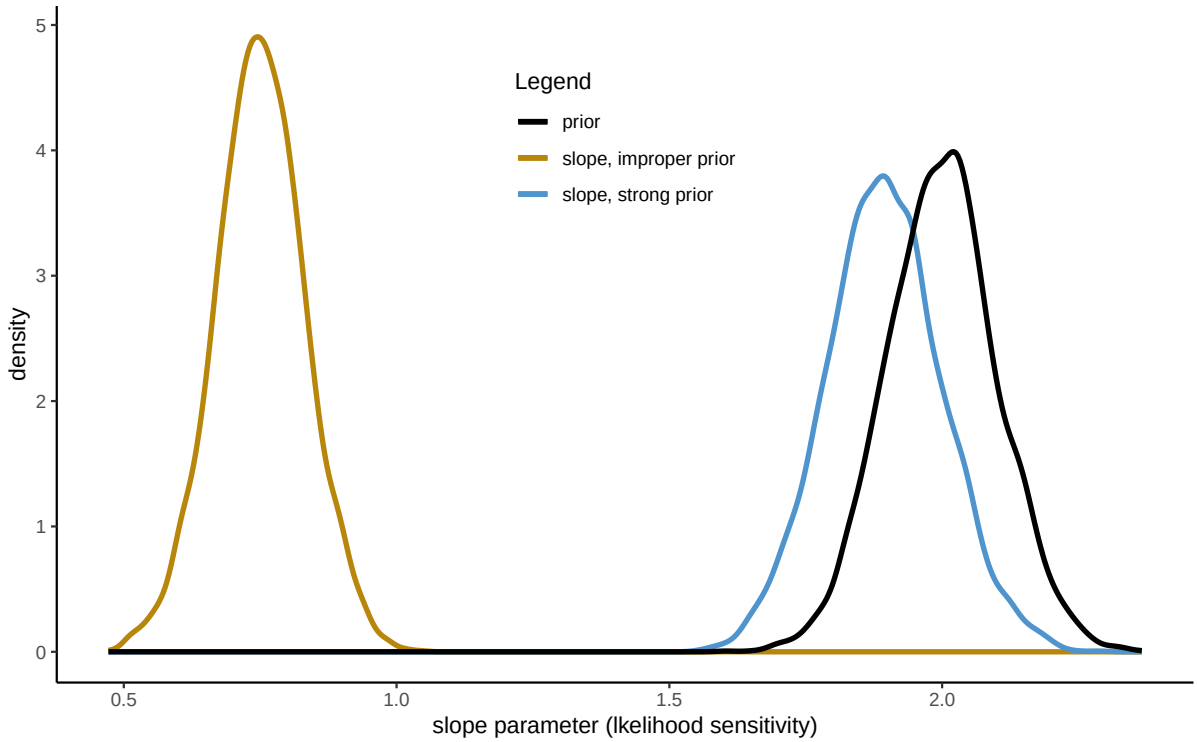


Figure 3.3: Slope parameter estimated with weak annd strong prior

3.3.5 Probit and logit regression

Let us assume that we have discrete choice data instead of a continuous variable, such as a certainty equivalent (or a present equivalent for inter-temporal choice). The main change to our setup will be that we now need to use a link function to map our model to a choice probability. Stan of course allows for the implementation of either a Probit model (using the standard normal CDF `Phi` or a faster approximation `Phi_approx`) or a Logit model. A standard logit regression will then look like this:

```
data{
int<lower=0> N;
array[N] int choice_risky;
vector[N] ev_diff;
}
parameters{
real alpha;
real beta;
}
model{
```

```
choice_risky ~ bernoulli_logit(alpha + beta * ev_diff);
}
```

where we examine the choice probability as a function of the difference in expected value between the prospect and the sure amount. Let us use the data of choice under risk from Abdellaoui et al. (2011) to illustrate the estimation:

```
rd <- read_csv("data/data_rich_domain_risk.csv")
rd <- rd %>% filter(low==0) %>%
             mutate(ev_diff = p*high - sure)

# creates data to send to Stan
stanD <- list(N = nrow(rd),
              ev_diff = rd$ev_diff,
              choice_risky = as.integer(1 - rd$choice_sure) )


# compile the Stan programme:
sr <- cmdstan_model("stancode/logit_ev.stan")

# run the model:
nm <- sr$sample(
  data = stanD,
  seed = 123,
  chains = 4,
  parallel_chains = 4,
  refresh=0,
  init=0,
  show_messages = TRUE,
  diagnostics = c("divergences", "treedepth", "ebfmi")
)

print(nm, c("alpha","beta"),digits = 3)
saveRDS(nm,file="stanoutput/logit_ev.Rds")
```

In discrete choice models of decision making, the model we want to implement will typically look different from the standard model shown above. For one, we may want to look at the difference between transformed quantities, i.e. the difference between the expected *utility* of the wager and the utility of the sure amount. Furthermore, it is customary to add an explicit noise term to the implementation, normally reflecting a random utility setup, and which takes the form of a Fechner error in the Probit model, or of a precision term in the logistic regression. We will examine such more realistic models in due time.

3.4 Aggregate non-linear models

In this section, I will develop some nonlinear models at the aggregate level. This will go fairly quickly—there is nothing that special about these types of models. This setup will, however, lay the foundations for the more interesting models to be considered in the next chapter.

3.4.1 A simple dual-EU model

Take indifferences of the type $c \sim (x, p)$, ie certainty equivalents for simple wagers with only one nonzero outcome. In a EU framework, we could model them as $c^\rho = p \times x^\rho$. We know, of course, that a perfectly equivalent way of modelling them is $c = p^\gamma \times x$, where $\gamma \triangleq \frac{1}{\rho}$. I will work with the latter for simply because it allows for a more natural transition to the multi-parameter models we will see below. The function, in either form, could of course be easily linearized and estimated with standard regression tools (try this as an exercise), but that would miss the point of this section.

For a change, we will use the data from Vieider et al. (2018), containing CEs for some 500 subjects from rural Ethiopia. For illustrative purposes, let us first explore the data in a nonparametric way. To this purpose, let $\pi(p)$ we a decision weight associated to the probability p, so that $c = \pi(p)x$, from which we get $\pi(p) = \frac{c}{x}$. To keep it simple, let us restrict our attention to $p = 0.5$. We can define an index of relative risk aversion as $rrp = p - \pi(p) = 0.5 - \pi(p)$. Risk averse subjects will have $\pi(p) < p$, which corresponds to the typical risk aversion for moderate probabilities documented in hosts of studies. Let us enshrine this knowledge in a prior $p(\mu) = \mathcal{N}(0.3, 200^{-1})$, which I have chosen to be fairly narrow, given the large evidence base. (I am not suggesting that this is appropriate; it rather serves illustration purposes).

The following code is devised to extract CEs from the data. Assume now you are conducting experiments in Ethiopia for the first time. You will observe the CEs sequentially, and you can thus update your prior with one observation at a time. To keep the illustration as simple as possible, we can cheat a little and use the true sample variance from the whole dataset, so that we only have to update the mean of the prior and the connected variance using the procedures developed in the first part. Note that I will treat the data as fully exchangeable, thereby ignoring the actual data collection structure (several CEs per subject, which is solved by using only the 50-50 prospect; subjects sampled from different villages; different villages sampled from different districts (Woredas), Woredas sampled from regions). The sequential updating with 50 randomly selected observations is illustrated in Figure 3.4.

```
eth <- read_csv("data/data_ETH.csv")
ece <- eth %>% filter(probability==0.5) %>%
               sample_n(50) %>%
               mutate(dw = equivalent/high)
```

```r
rrp <- 0.5 - ece$dw
sigma2 <- var(rrp)
tau2 <- 200^(-1)
mu <- 0.3

for(i in 2:51){
mu[i] <- mu[i-1] + (tau2[i-1])/(tau2[i-1] + sigma2)*(rrp[i-1] - mu[i-1])
tau2[i] <- (tau2[i-1]*sigma2)/(tau2[i-1] + sigma2)
}

ggplot(data = data.frame(x = c(-0.2, 0.6)), aes(x)) +
  stat_function(aes(color = "prior"),fun = dnorm, n = 101, args = list(mean = mu[1], sd = sq
  geom_vline(xintercept=0,colour="black",linetype="dashed",size=1.1) +
  geom_vline(xintercept=mean(rrp),colour="cornflowerblue",linetype="dashed",size=1.5) +
  stat_function(aes(color="after 5 obs."),fun = dnorm, n = 101, args = list(mean = mu[6], sd
  stat_function(aes(color="after 10 obs."),fun = dnorm, n = 101, args = list(mean = mu[11], 
  stat_function(aes(color="after 50 obs."),fun = dnorm, n = 101, args = list(mean = mu[51], 
  ylab("")  + scale_y_continuous(breaks = NULL)  + xlab("relative risk premium (posterior mea
  scale_colour_manual("Legend", values = c("darkgoldenrod","chocolate4","darkolivegreen4","bl
  theme(legend.position = c(0.85, 0.85))
```

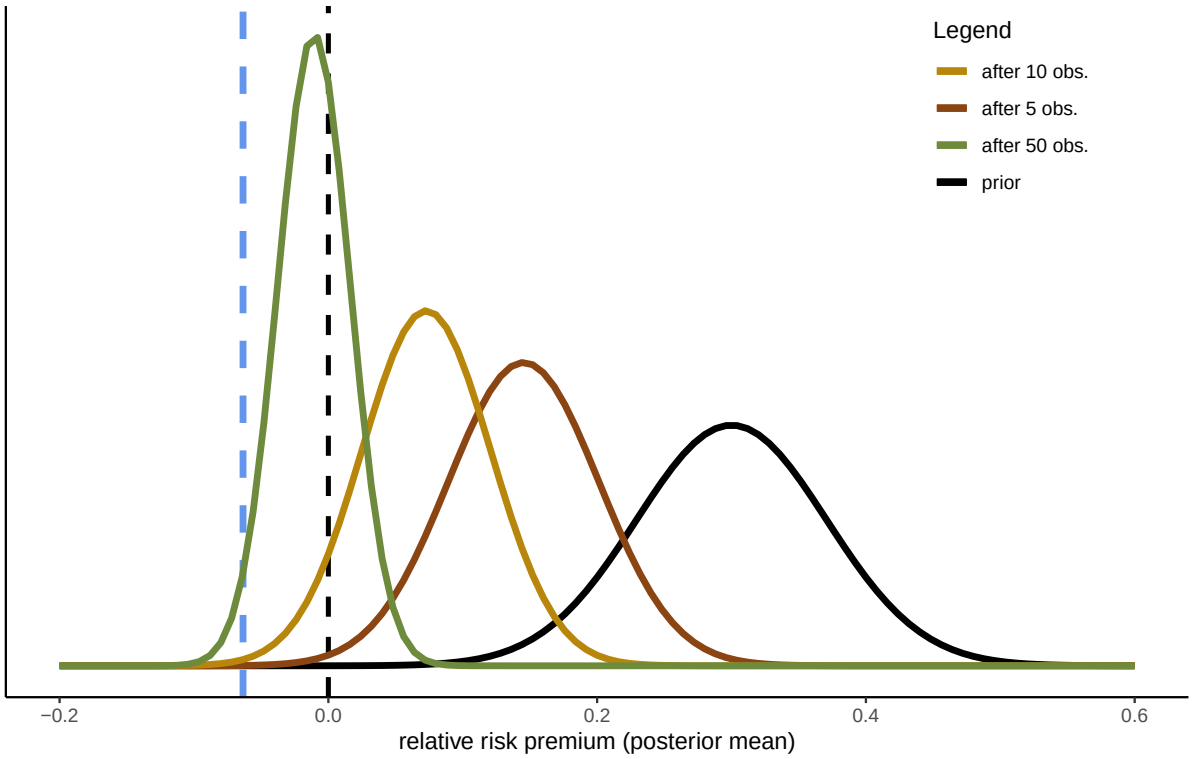


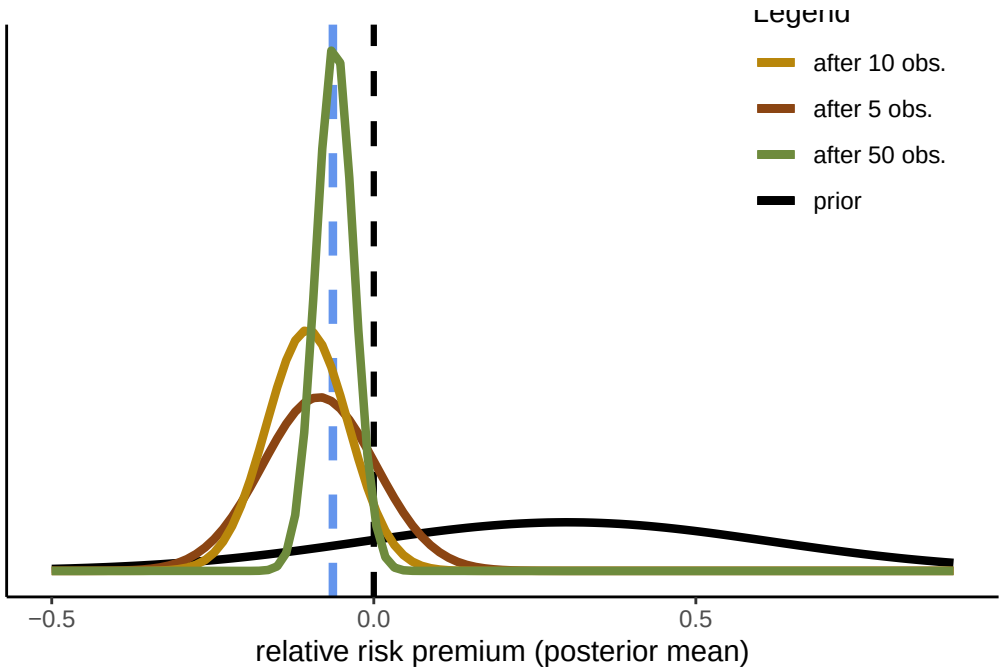
Figure 3.4: Sequential updating of decision weights

We can see that the model gradually converges to the true sample mean, indicated by the dashed blue line. How close this is to the true mean of the overall data, and how quickly the mean converges to the true mean, will of course depend on the random sample extracted (just run the simulation including the random extraction repeatedly changing the seed to convince yourself of that). One way to speed up the process is, of course, to reduce the precision of your prior. That is, if you are not convinced that everybody will necessarily decide like select samples from a handful fo WEIRD Western countries, you may choose your prior to have a wider variance, even while sticking with the same mean:

```
sigma2 <- var(rrp)
tau2 <- 10^(-1)
mu <- 0.3

for(i in 2:51){
mu[i] <- mu[i-1] + (tau2[i-1])/(tau2[i-1] + sigma2)*(rrp[i-1] - mu[i-1])
tau2[i] <- (tau2[i-1]*sigma2)/(tau2[i-1] + sigma2)
}
```

```
ggplot(data = data.frame(x = c(-0.5, 0.9)), aes(x)) +
  stat_function(aes(color = "prior"),fun = dnorm, n = 101, args = list(mean = mu[1], sd = sq
  geom_vline(xintercept=0,colour="black",linetype="dashed",size=1.1) +
  geom_vline(xintercept=mean(rrp),colour="cornflowerblue",linetype="dashed",size=1.5) +
  stat_function(aes(color="after 5 obs."),fun = dnorm, n = 101, args = list(mean = mu[6], sd
  stat_function(aes(color="after 10 obs."),fun = dnorm, n = 101, args = list(mean = mu[11], 
  stat_function(aes(color="after 50 obs."),fun = dnorm, n = 101, args = list(mean = mu[51], 
labs(x="relative risk premium (posterior mean)",y="") +
  scale_y_continuous(breaks = NULL) +
  scale_colour_manual("Legend", values = c("darkgoldenrod","chocolate4","darkolivegreen4","bl
  theme(legend.position = c(0.85, 0.85))
```



As you would expect, convergence to the true mean is both quicker and more complete.

3.4.2 Dual-EU model: Aggregate estimation

We can now proceed to aggregate estimation. I will here use all stimuli, instead of restricting attention only to 50-50 wagers as above. To this end, we can use the model below. A few comments are in order. Other than above, I have imposed mildly regularizing priors on the parameters. While it may be legitimate to impose stronger priors based on past evidence on risk aversion over much of the probability interval, there are two major counterarguments (even without taking the evidence we have seen above into account): 1) risk seeking has been

found to coexist with risk aversion; 2) the novel context makes strong priors taken from other contexts questionable.

Another change from the models seen to far is that I now define several auxiliarity variables in the model block (which are thus available only locally), which I then enter into subsequent computations. This is arguably not necessary in a simple model such as this one, but it will become essential as the model complexity increases. Also, I am using a loop instead of vectorizing, because a vector taken to a power is not defined (NOTE: The Stan development team has been working on a function allowing to take powers of individual vector elements, which would overcome this issue).

```
data{
    int<lower=1> N;
    array[N] real ce;
    array[N] real high;
    array[N] real p;
}
parameters{
    real<lower=0> gamma;
    real<lower=0> sigma;
}
model{
//auxiliary variables (available locally in model block):
    vector[N] wp;
    vector[N] pv;
// diffuse priors:
    sigma ~ normal( 0.2 , 0.5 );
    gamma ~ normal( 1 , 0.5 );
//model:
    for ( i in 1:N ) {
        wp[i] = p[i]^gamma ;
        pv[i] = wp[i] * high[i];
        ce[i] ~ normal( pv[i] , sigma * high[i] );
    }
}
```

We are now ready to estimate the model:

```
eth <- read.csv("data/data_ETH.csv")

# creates data to send to Stan
stanD <- list(N = nrow(eth),
              ce = eth$equivalent,
```

```
              high = eth$high,
              p = eth$probability)


# compile the Stan programme:
sr <- cmdstan_model("stancode/agg_DEU.stan")

# run the model:
nm <- sr$sample(
  data = stanD,
  seed = 123,
  chains = 4,
  parallel_chains = 4,
  refresh=0,
  init=0,
  show_messages = TRUE,
  diagnostics = c("divergences", "treedepth", "ebfmi")
)

nm$print(c("gamma", "sigma"), digits = 3,
    "mean", SE = "sd", "rhat" ,
    ~quantile(., probs = c(0.025, 0.975)))
nm$save_object(file="stanoutput/agg_DEU_ETH.Rds")
```

We can now compare the fitted data to the nonparametric data points to check the validity of our model. This is straightforward in a dual-EU setting, where the nonparametric decision weights are simply $\frac{c}{x}$. A rigorous analysis might add confidence intervals around the fitted function. This is usually done with simulations. Bayesian analysis is particularly convenient here, since we can simply sample from the posterior. Creating a graph including the uncertainty intervals is left for an exercise.

```
deu <- readRDS("stanoutput/agg_DEU_ETH.Rds")
de <- as_draws_df(deu)
m <- mean(de$gamma)

eth <- eth %>% mutate(dw = equivalent/high) %>%
               group_by(probability) %>%
               mutate(mean_dw = mean(dw)) %>%
               ungroup()

pw <- function(x) (x^m)
ggplot(data=eth, aes(x=probability,y=mean_dw)) +
```

```
  stat_function(fun = pw,color="chartreuse4",size=1) + xlim(0,1) +
  geom_point(size=3, color="blue",shape=10) +
  geom_abline(intercept=0, slope=1,linetype="dashed",color="gray") + labs(x = "probability",
```

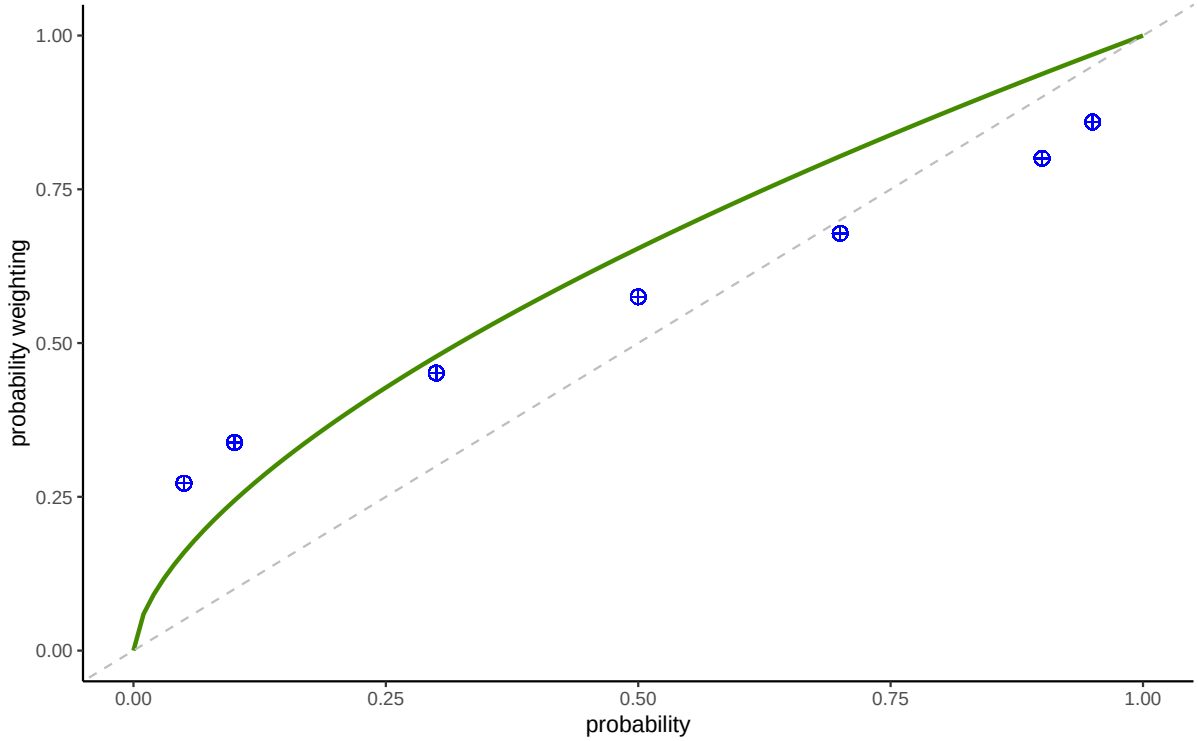


Figure 3.5: Dual-EU model with power weighting function

The aggregate data shown in Figure 3.5 indicate a concave pattern, whereas risk aversion requires a convex pattern. After looking at the nonparametric data patterns above, this is no surprise. Clearly, the model might benefit from a second parameter, allowing it to capture the coexistence of risk seeing and risk aversion for different probability levels (or simply a different functional form, achieving the same with just one parameter).

3.4.3 Multiple parameters

There is nothing special about having a model with more than one parameter (technically, the model above already had two including the variance). The following code implements a (nonlinear version of the) Goldstein-Einhorn weighting function:

```
data{
    int<lower=1> N;
    array[N] real ce;
    array[N] real high;
    array[N] real p;
}
parameters{
    real<lower=0> gamma;
    real<lower=0> delta;
    real<lower=0> sigma;
}
model{
\\auxiliary variables:
    vector[N] wp;
    vector[N] pv;
\\priors:
    sigma ~ normal( 0.2 , 0.5 );
    delta ~ normal( 1 , 0.5 );
    gamma ~ normal( 1 , 0.5 );
\\model:
    for ( i in 1:N ) {
        wp[i] = (delta*p[i]^gamma)/(delta*p[i]^gamma + (1-p[i])^gamma);
        pv[i] = wp[i] * high[i];
        ce[i] ~ normal( pv[i] , sigma * high[i] );
    }
}
```

```
eth <- read_csv("data/data_ETH.csv")

# creates data to send to Stan
stanD <- list(N = nrow(eth),
              ce = eth$equivalent,
              high = eth$high,
              p = eth$probability)


# compile the Stan programme:
sr <- cmdstan_model("stancode/agg_DEU_GE.stan")

# run the model:
nm <- sr$sample(
  data = stanD,
```

```
  seed = 123,
  chains = 4,
  parallel_chains = 4,
  refresh=0,
  init=0,
  show_messages = TRUE,
  diagnostics = c("divergences", "treedepth", "ebfmi")
)

nm$print(c("delta","gamma", "sigma"), digits = 3,
    "mean", SE = "sd", "rhat" ,
    ~quantile(., probs = c(0.025, 0.975)))
nm$save_object(file="stanoutput/agg_DEU_GE.Rds")
```

```
deu <- readRDS("stanoutput/agg_DEU_GE.Rda")
de <- as_draws_df(deu)
m <- mean(de$gamma)
l <- mean(de$delta)

eth <- eth %>% mutate(dw = equivalent/high) %>%
               group_by(probability) %>%
               mutate(mean_dw = mean(dw)) %>%
               ungroup()

pw <- function(x) (l*x^m/(l*x^m + (1-x)^m))
ggplot(data=eth, aes(x=probability,y=mean_dw)) +
  stat_function(fun = pw,color="chartreuse4",size=1) + xlim(0,1) +
  geom_point(size=3, color="blue",shape=10) +
  geom_abline(intercept=0, slope=1,linetype="dashed",color="gray") + labs(x = "probability",
```

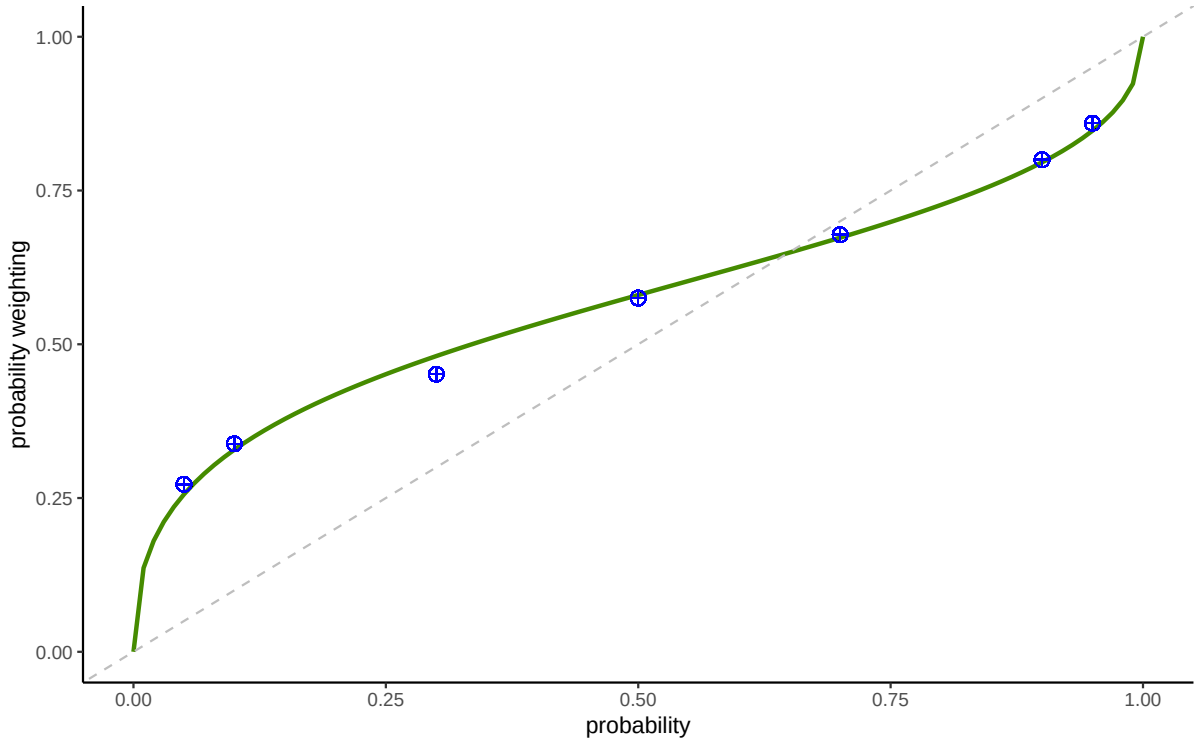


Figure 3.6: Dual-EU model with power weighting function

We can now compare the fitted function to the nonparametric data points to check the validity of our model. This is shown in Figure 3.6. The fit to the aggregate data is no almost perfect.

3.4.4 A discrete choice DU model

To illustrate the estimation of a multiparameter model in a discrete choice setup, I will use some discreet choice data of tradeoffs between sooner-smaller against larger-later options obtainned in an incentivized classroom experiment. Before launching into the estimations, however, it is always useful to obtain an idea of the main patterns present in the data using nonparametric methods. This can then serve as a check for the patterns one observes in the structural model, and it is here left for an exercise.

The data consist of a large number of binary choices between a smaller-sooner amount and a larger later amount, which is kept fixed at 50 Euros. Say we are interested in quasi-hyperbolic discounting, i.e. in whether the choice patterns exhibit present-bias. This can be captured by the discounting function $D(t) = \beta \times exp(-rt)$, where $\beta < 1$ indicates present-bias. For $t = 0$, the function will be $D(0) = 1$. When both outcomes are pushed to the future, β drops out of the model and discounting reduces to an exponential model. (In practice, present bias will be confounded by subadditivity in the data, which the function cannot handle; however, I am

interested in the technicalities of the estimation process, rather than in the accuracy of the inferences drawn).

We can also use this example to illustrate another Bayesian feature—so-called regions of practical equivalence, or *acceptance regions*. These correspond to parameter intervals for which we will *accept* the null hypothesis. Other than in frequentist statistics, which invariably tests point hypotheses, Bayesian statistics allows both for rejecting and for accepting the null hypothesis. Let us assume an acceptance region of > 0.97 (of of > 0.97 and < 1.03, if we think there might be a chance of finding the opposite of present bias).

Now we are ready to run the structural model. Other than in the simple regressions seen above, we will want to add an explicit error term. Let the discounted utility of the smaller-sooner option be $D(s)y$, with the discounted utility of the larger-later option described by $D(\ell)x$. The discount function $D(s)$ will be the exponential function, except when the sooner time is 0, in which case there will be a β in $D(\ell)$ to account for present bias. We can then implement a logistic model by means of the following link function, capturing the probability with which the larger-later option is chosen:

$$Pr[(x, \ell) \succ (y, s)] = \frac{e^{\lambda D(\ell)x}}{e^{\lambda D(\ell)x} + e^{\lambda D(s)y}},$$

where λ is a inverse temperature parameter, so that $\lambda = \infty$ corresponds to a noiseless process. We can code the model as follows:

```
data{
    int<lower=1> N;
    array[N] int id;
    array[N] real x;
    array[N] real y;
    array[N] real s;
    array[N] real l;
    array[N] int cl;
    array[N] int s0;
}
parameters{
  real<lower=0> rho;
  real<lower=0> beta;
  real<lower=0> lambda;
}
model{
    vector[N] dus;
    vector[N] dul;
    vector[N] prl;
```

```
    rho ~ normal(0.1, 0.2);
    beta ~ normal(1, 0.5);
    lambda ~ normal(10, 20);

  for ( i in 1:N ) {
        dus[i] = exp(-rho * l[i] ) * y[i];
        dul[i] = beta^s0[i] * exp(-rho * l[i] ) * x[i];
        prl[i] = exp(lambda * dul[i])/( exp(lambda * dul[i]) + exp(lambda * dus[i]) );
        cl[i] ~ bernoulli( prl[i] );
    }
}
```

```
d <- read_csv("data/data_NCT.csv")
da <- d %>% mutate(immediate = ifelse(ts==0 , 1 , 0)) %>%
            filter( progress==100)

#create data to send to Stan
stanD <- list(N = nrow(da),
              id = da$id,
              x = abs(da$large),
              y = abs(da$small),
              s = da$ts,
              l = da$tl,
              cl = da$choice_later,
              s0 = da$immediate)

# compile the Stan programme:
sr <- cmdstan_model("stancode/agg_QH_logit.stan")

# run the model:
nm <- sr$sample(
  data = stanD,
  seed = 123,
  chains = 4,
  parallel_chains = 4,
  refresh=200,
  init=0,
  show_messages = TRUE,
  diagnostics = c("divergences", "treedepth", "ebfmi")
)

nm$print(c("rho","beta", "lambda"), digits = 3,
```

```
    "mean", SE = "sd", "rhat" ,
    ~quantile(., probs = c(0.025, 0.975)))
nm$save_object(file="stanoutput/agg_qh_logit.Rds")
```

```
nm <- readRDS("stanoutput/agg_qh_logit.Rds")

nm$print(c("rho","beta", "lambda"), digits = 3,
    "mean", SE = "sd", "rhat" ,
    ~quantile(., probs = c(0.025, 0.975)))
```

```
 variable  mean    SE  rhat  2.5% 97.5%
   rho    0.039 0.002 1.002 0.035 0.044
   beta   0.820 0.002 1.003 0.815 0.824
   lambda 0.294 0.011 1.002 0.273 0.316
```

We find substantial present bias. It is clear from the distribution of the parameter that iit falls very far from the region of accepance, so that we can reject the null hypothesis.

While we have coded the inverse-logit link function by hand using the ratio of exponentials. Aas a rule, this is not needed in Stan, which offers (more efficient) pre-rpogrammed funcion. n particular, the `bernulli_logit` function reunites the link function and Bernouli ransformation in one neat function. Here, I will illustrate how to code a Probit model using Stan's pre-existing functions.

```
data{
    int<lower=1> N
    array[N] int id;
    array[N] real x;
    array[N] real y;
    array[N] real s;
    array[N] real l;
    array[N] int cl;
    array[N] int s0;
}
parameters{
  real<lower=0> rho;
  real<lower=0> beta;
  real<lower=0> sigma;
}
model{
    vector[N] disc;
```

```
    vector[N] udiff;

    rho ~ normal(0.1, 0.2);
    beta ~ normal(1, 0.5);
    sigma ~ normal(0.2, 0.5);

  for ( i in 1:N ) {
        dus[i] = exp(-rho * l[i] ) * y[i];
        dul[i] = beta^s0[i] * exp(-rho * l[i] ) * x[i];
        udiff[i] = (dul[i] - dus[i])/(sqrt(2) * sigma);
        cl[i] ~ bernoulli( Phi( udiff[i] ) );
    }
}
```

```
d <- read_csv("data/data_NCT.csv")
da <- d %>% mutate(immediate = ifelse(ts==0 , 1 , 0)) %>%
            filter( progress==100)

#create data to send to Stan
stanD <- list(N = nrow(da),
              id = da$id,
              x = abs(da$large),
              y = abs(da$small),
              s = da$ts,
              l = da$tl,
              cl = da$choice_later,
              s0 = da$immediate)

# compile the Stan programme:
sr <- cmdstan_model("stancode/agg_QH.stan")

# run the model:
nm <- sr$sample(
  data = stanD,
  seed = 123,
  chains = 4,
  parallel_chains = 4,
  refresh=200,
  init=0,
  show_messages = TRUE,
  diagnostics = c("divergences", "treedepth", "ebfmi")
)
```

```
nm$print(c("rho","beta", "sigma"), digits = 3,
    "mean", SE = "sd", "rhat" ,
    ~quantile(., probs = c(0.025, 0.975)))
nm$save_object(file="stanoutput/agg_qh.Rds")
```

```
nm <- readRDS("stanoutput/agg_qh.Rds")

nm$print(c("rho","beta", "sigma"), digits = 3,
    "mean", SE = "sd", "rhat" ,
    ~quantile(., probs = c(0.025, 0.975)))
```

```
 variable  mean    SE  rhat  2.5% 97.5%
    rho   0.012 0.000 1.000 0.012 0.012
    beta  0.950 0.003 1.000 0.943 0.956
    sigma 0.152 0.002 1.002 0.148 0.155
```

The quasi-hyperbolicity parameter we obtain is virtually identical to the one we obtained from the logistic model above. The discount rate, however, is slightly larger. That is of course not surprising, given that we have estimated a different model. Statisticallly speaking, however, the two discount rates aare not clearly distinguishable, showcasing the similarity between the two models.

3.5 Individual-level estimation

While we have so far focused on aggregate estimations, it is often of interest to present individual-level estimations. There is double good news in this respect. Not only can the aggregate-level techniques we have just seen be applied to individual-level estimations, but they are typically also both easier to obtain and more stable than using maximum likelihood techniques. This section starts by illustrating some of the problems in traditional ML estimation of individual-level estimations, and how to overcome them in Bayesian estimations.

3.5.1 The trouble with individual-level estimates

Individual-level estimations tend to be quite delicate due to the reduced number of data points that are typically available. Overfitting sparse noisy data thus becomes a major concern. This is particularly true in multi-parameter models such as prospect theory, where multiple unconstrained parameters compete to pick up similar behavioural motives. In the presence of noise, the different dimensions may not be well separated, giving rise to likelihoods with

local maxima, or to flat regions presenting similar likelihoods. It is thus doubly important to check the results against the data. Estimates may be sensitive to starting values of the parameters when data are noisy, which constitutes itself a type of identifiability problem. The standard solution in ML estimation is to run repeated estimations for each individual using different starting values (a so-called *grid search procedure*), which serves to detect whether the estimations may converge to different maxima of the likelihood function, some of which may be local. The Bayesian routines we use here do this automatically to some extent, if every chain is started at randomly generated values (which is done automatically if no initial values are specfied). It is thus important to check whether the chains "mix" properly, i.e. the different chains should explore the same parameter values.

We illustrate this issue by estimating the individual-level parameters of subject 65 in the Zurich 2006 data of Bruhin, Fehr-Duda, and Epper (2010) using the aggregate routine used previously, which imposes improper priors and is thus equivalent to a ML estimator, i.e. an "emprical Bayes" estimation (the data are not included inn the repository, but can be downloaded from https://www.econometricsociety.org/publications/econometrica/2010/07/01/risk-and-rationality-uncovering-heterogeneity-probability/supp/7139_data%20and%20programs_0.zip). The diagnostic summary indicates that, while three chains had no problems, chain 4 had a fairly large share of so-called "divergent iterations" (as indicated by the warning: *Warning: 362 of 4000 (9.0%) transitions ended with a divergence*; NOTE: the exact value may differ depending on the seed you set and the resulting random starting values). While such divergent iterations may at times occur in nonlinear estimation, they should always be checked to be sure that they do not signal fundamental problems with the estimation (and 9% is a lot, raising all kinds of red flags).

```
d <- read_csv("data/data_RR_ch06.csv")
dg <- d %>% filter(z1 > 0) %>%
            mutate(high = z1) %>%
            mutate(low = z2) %>%
            mutate(prob = p1) %>%
            group_by(id) %>%
            mutate(id = cur_group_id()) %>%
            ungroup()

dg65 <- dg %>% filter(id==65)

pt <- cmdstan_model("stancode/agg_RDU_improper.stan")


# ind 65
stanD <- list(N = nrow(dg65),
              high = dg65$high,
              low = dg65$low,
```

```
              p = dg65$prob,
              ce = dg65$ce)

s65 <- pt$sample(
  data = stanD,
  seed = 123,
  chains = 4,
  parallel_chains = 4,
  refresh=0,
  show_messages = TRUE,
  diagnostics = c("divergences", "treedepth", "ebfmi")
)

s65$save_object(file="stanoutput/ind_RR_s65.Rds")
```

```
s65 <- readRDS("stanoutput/ind_RR_s65.Rds")

s65$diagnostic_summary()
```

```
$num_divergent
[1] 273   0   0   0

$num_max_treedepth
[1] 725   0   0   0

$ebfmi
[1] 1.0474750 0.8806989 0.8291272 0.7814946
```

```
p65 <- s65$draws(format = "array")

mcmc_trace(p65, pars = c("rho", "sigma"))
```

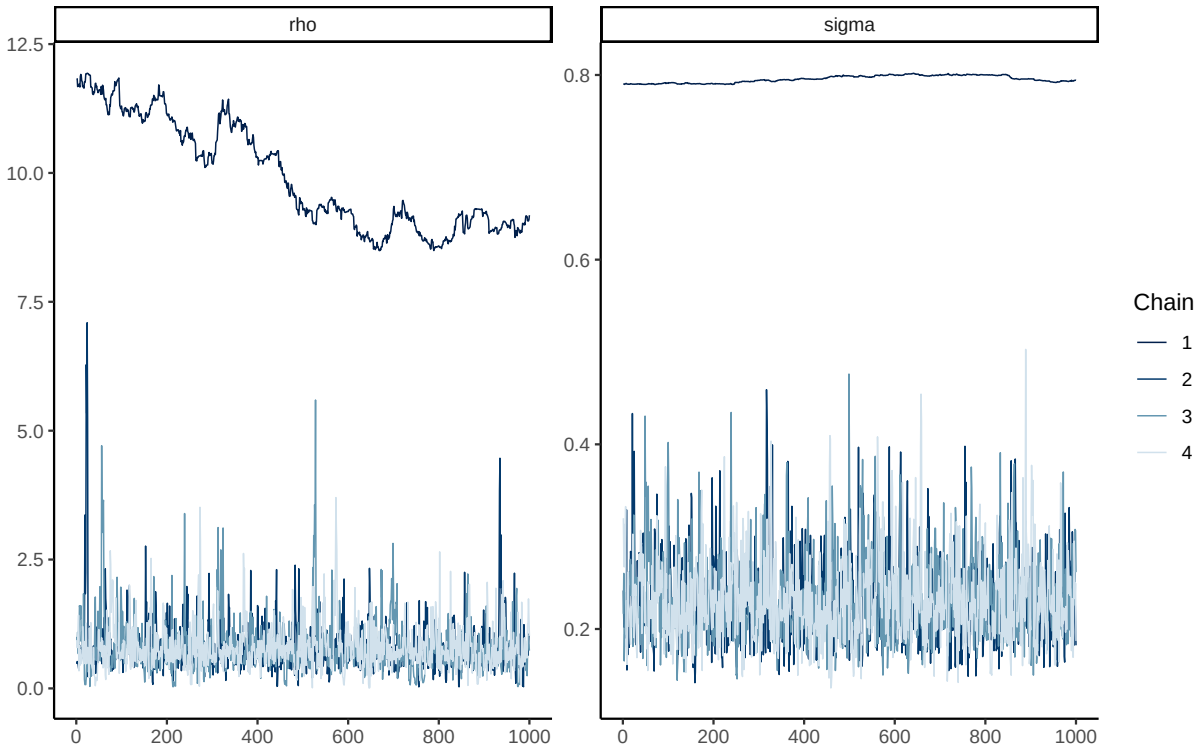


Figure 3.7: Trace plot for subject 65, Emprical Bayes estimation

Figure 3.7 shows the trace plot, i.e. the parameter points visited by the simulation algorithm in each chain, for parameters ρ and σ. Three chains can be seen to have mixed properly (although some extreme spikes are indicative of large posterior uncertainty). One chain, however, produces very different values for both parameters, as well as displaying high degrees of autocorrelation. We should thus not accept the estimates, since these patterns indicate major problems in the estimation.

The results just seen clearly indicate trouble in the estimation routine. Notice that, by choosing appropriate starting values by hand, and possibly even with a given seed, you may be able to sweep these issues under the rug. That would, however, be a terrible idea. What these results indicate is very simply that the estimates we obtained cannot be trusted. It is thus not enough to simply force the divergent chain towards the others by brute force, and we will need to come up with an alternative procedure. Luckily, that is easily done.

3.5.2 How to overcome issues in individual estimates

One way of overcoming this issue is to impose so-called *regularizing priors*. Such priors are typically determined to be both noninformative and diffuse. Below, we reproduce a code snippet of the model block, where priors are now given for the 4 model parameters. Note

that the mean of the prior is fixed at 1, which for the model parameters ρ, γ, and δ coincide with the values entailing expected value maximization. This is the sense in which the values are noninformative, since we do not actually incorporate any prior knowledge, e.g. on the near-universality of likelihood-insensitivity (see below). Furthermore, by choosing a standard deviation that is suitably large, we make sure that the data can easily overpower the prior. In the case illustrated below, 95% of the probability mass in the prior is attributed to parameter values ranging between $[-1, 3]$ (even though we cut off the distribution at 0, since values below 0 are not admissible for most of the parameters). This gives a large enough berth to accommodate any plausible parameter values. Even though this prior does not restrict the estimations in any substantive way, it may nevertheless be sufficient to point the algorithm in the right direction, thus avoiding the issues described above.

```
model {
  vector[N] pw;
  rho ~ normal( 1 , 1 ); //prior for rho
  gamma ~ normal( 1 , 1 ); //prior for gamma
  delta ~ normal( 1 , 1 ); //prior for delta
  sigma ~ normal( 1 , 1 ); //prior for sigma
for (i in 1:N){
    pw[i] = ( delta * p[i]^gamma ) / ( delta * p[i]^gamma + ( 1 - p[i] )^gamma );
    ce[i] ~ normal( ( pw[i] * high[i]^rho + (1 - pw[i]) * low[i]^rho )^(1/rho) ,
                    sigma * (high[i] - low[i]) );
    }
}
```

Below, I repeat the estimation for subject 65 with the non-informative, diffuse priors described above. The divergent iterations disappear, and the chains now appear to mix properly, as can be seen from Figure 3.8. This shows the power of priors. Even though the priors do not restrain the parameter estimates in any substantive way, such mildly regularizing priors are sufficient to discipline the estimates and avoid convergence problems in this case. Note, however, that in general one ought to be careful when choosing priors for sparse data, as we have done here. Imposing specific values and making the priors too narrow may well sway the estimate. It is thus always necessary to check the fit to the data *in addition* to executing convergence checks, such as illustrated above. It is also a good idea to vary the prior and to make sure that the particular results obtained are not sensitive to different priors falling in a reasonable range.

```
d <- read_csv("data/data_RR_ch06.csv")
dg <- d %>% filter(z1 > 0) %>%
            mutate(high = z1) %>%
            mutate(low = z2) %>%
            mutate(prob = p1) %>%
            group_by(id) %>%
```

```
            mutate(id = cur_group_id()) %>%
            ungroup()

dg65 <- dg %>% filter(id==65)
dg37 <- dg %>% filter(id==37)

pt <- cmdstan_model("stancode/agg_PT_priors.stan")

# ind 65
stanD <- list(N = nrow(dg65),
              high = dg65$high,
              low = dg65$low,
              p = dg65$prob,
              ce = dg65$ce)

s65 <- pt$sample(
  data = stanD,
  seed = 123,
  chains = 4,
  parallel_chains = 4,
  refresh=0,
  show_messages = FALSE,
  diagnostics = c("divergences", "treedepth", "ebfmi")
)

s65$save_object(file="stanoutput/ind_RR_s65_clean.Rds")

# ind 37
stanD <- list(N = nrow(dg37),
              high = dg37$high,
              low = dg37$low,
              p = dg37$prob,
              ce = dg37$ce)

s37 <- pt$sample(
  data = stanD,
  seed = 123,
  chains = 4,
  parallel_chains = 4,
  refresh=0,
  show_messages = FALSE,
  diagnostics = c("divergences", "treedepth", "ebfmi")
```

```
)

s37$save_object(file="stanoutput/ind_RR_s37.Rds")
```

```
s65 <- readRDS("stanoutput/ind_RR_s65_clean.Rds")
p65 <- s65$draws(format = "array")

mcmc_trace(p65, pars = c("rho", "sigma"))
```

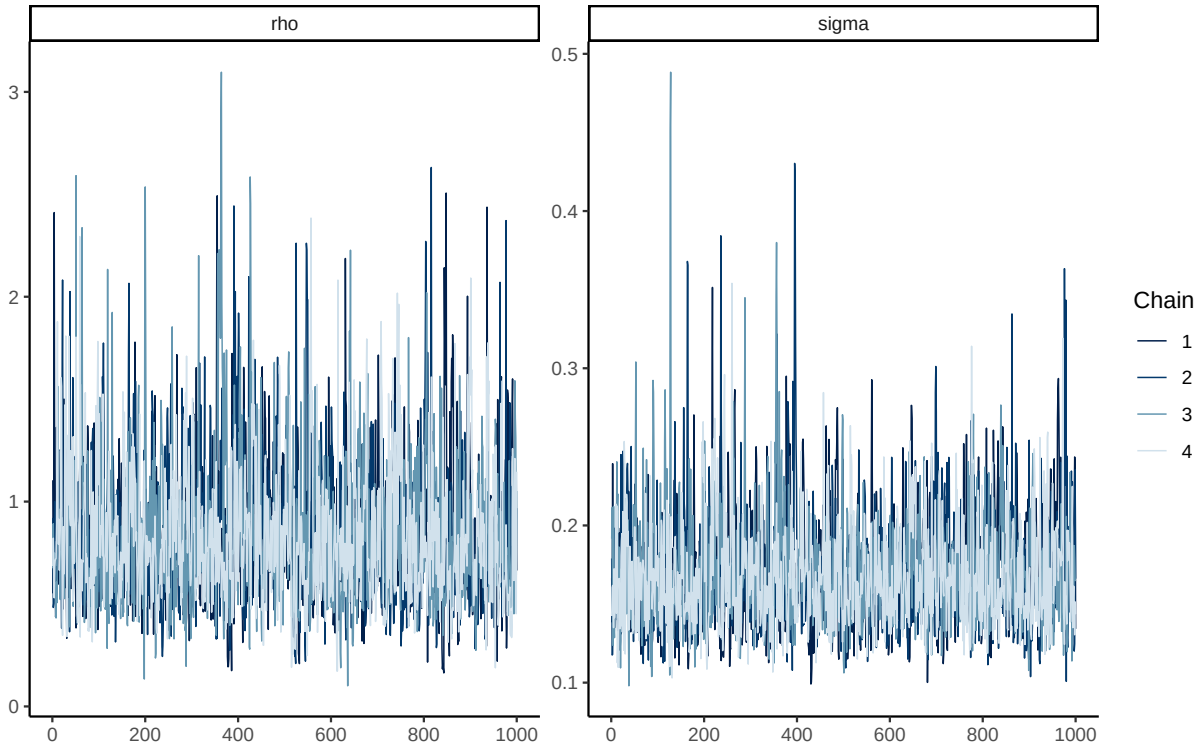


Figure 3.8: Trace plot for subject 65, with regularizing priors

```
d <- read_csv("data/data_RR_ch06.csv")
dg <- d %>% filter(z1 > 0) %>%
            mutate(high = z1) %>%
            mutate(low = z2) %>%
            mutate(prob = p1) %>%
            group_by(id) %>%
            mutate(id = cur_group_id()) %>%
            ungroup()
```

```
dg65 <- dg %>% filter(id==65)
dg37 <- dg %>% filter(id==37)

s37 <- readRDS("stanoutput/ind_RR_s37.Rds")
s65 <- readRDS("stanoutput/ind_RR_s65_clean.Rds")

#
color_scheme_set("red")
p37 <- s37$draws(format="array")
g37 <- mcmc_intervals(p37, pars = c("rho","gamma","delta","sigma")) +
       geom_label(label="Subject 37", x=0.15 , y=4.2)

color_scheme_set("brightblue")
p65 <- s65$draws(format="array")
g65 <- mcmc_intervals(p37, pars = c("rho","gamma","delta","sigma")) +
       geom_label(label="Subject 37", x=0.15 , y=4.2)

#
d37 <- as_draws_df(s37)
d65 <- as_draws_df(s65)

rho37 <- d37$rho
rho65 <- d65$rho
gamma37 <- d37$gamma
gamma65 <- d65$gamma
delta37 <- d37$delta
delta65 <- d65$delta
sigma37 <- d37$sigma
sigma65 <- d65$sigma


# graph of aggregate functions with uncertainty

dg37 <- dg37 %>% mutate( dw = (ce - low)/(high - low) ) %>%
                 group_by(prob,high,low) %>%
                 mutate(mean_dw = mean(dw)) %>%
                 mutate(median_dw = median(dw)) %>%
                 ungroup()

dg65 <- dg65 %>% mutate( dw = (ce - low)/(high - low) ) %>%
                 group_by(prob,high,low) %>%
                 mutate(mean_dw = mean(dw)) %>%
```

```
                mutate(median_dw = median(dw)) %>%
                ungroup()

# subject 37
r37 <- d37 %>% sample_n(size=100) %>%
                mutate(row = row_number()) %>%
                select(delta,gamma,row)

pw37 <- function(x) (mean(delta37) * x^mean(gamma37) /
                     (mean(delta37) * x^mean(gamma37) + (1-x)^mean(gamma37)))
wf37 <- ggplot() +
  purrr::pmap(r37, function(delta,gamma, row) {
    stat_function( fun = function(x) ( delta*x^gamma )/( delta*x^gamma + (1-x)^gamma) ,
                   color="grey" , linewidth = 0.25)
    }) +
  stat_function(fun = pw37,color="chartreuse4",size=1) + xlim(0,1) +
  geom_point(aes(x=dg37$prob,y=dg37$mean_dw),size=3, color="chartreuse4",shape=10) +
  geom_abline(intercept=0, slope=1,linetype="dashed",color="gray") +
    xlab("probability") + ylab("decision weights")


# subject 65 with credibility bands
r65 <- d65 %>% sample_n(size=100) %>%
                mutate(row = row_number()) %>%
                select(delta,gamma,row)

pw65 <- function(x) (mean(delta65) * x^mean(gamma65) /
                     (mean(delta65) * x^mean(gamma65) + (1-x)^mean(gamma65)))

wf65 <- ggplot(dg65) +
   purrr::pmap(r65, function(delta,gamma, row) {
    stat_function( fun = function(x) ( delta*x^gamma )/( delta*x^gamma + (1-x)^gamma) ,
                   color="grey" , linewidth = 0.25)
    }) +
  stat_function(fun = pw65,color="blue",size=1) + xlim(0,1) +
  geom_point(aes(x=prob,y=mean_dw),size=3, color="cornflowerblue",shape=10) +
  geom_abline(intercept=0, slope=1,linetype="dashed",color="gray") +
    xlab("probability") + ylab("decision weights")


 g <- ggarrange(g37,g65,wf37,wf65,
            ncol = 2, nrow=2)
```

```
g
```

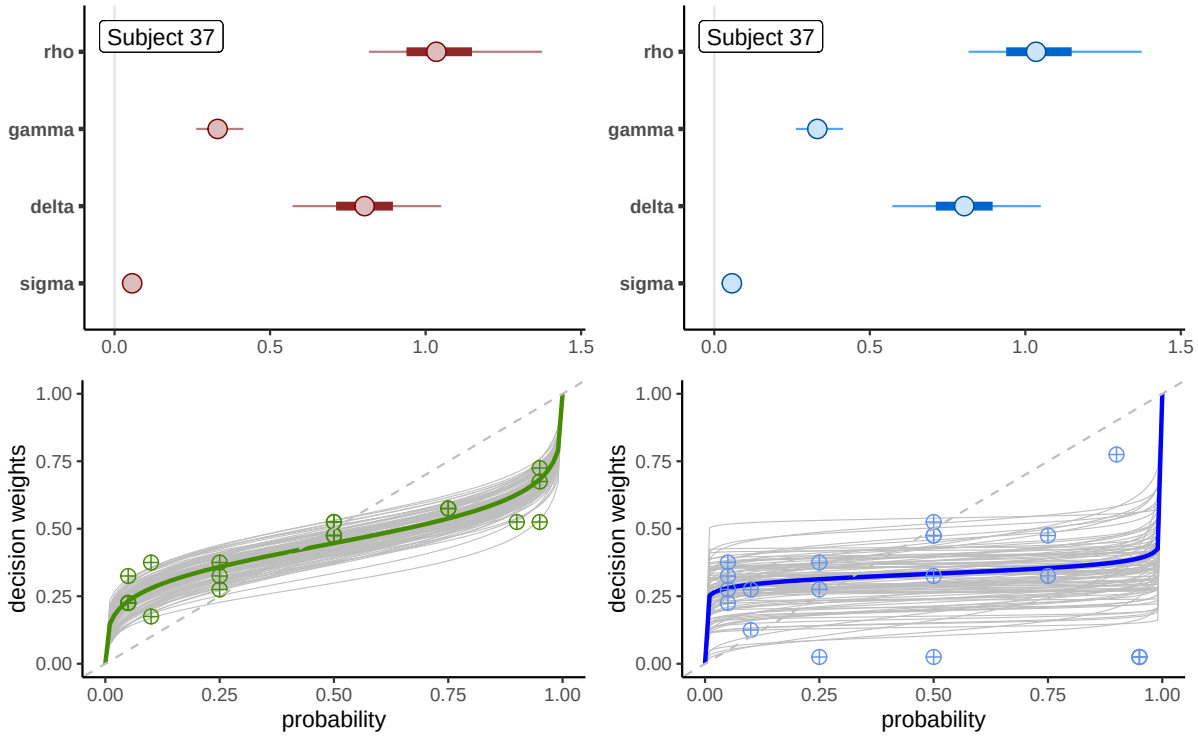


Figure 3.9: Parameters and functions for subjects 37 and 65

Figure 3.9 shows the parameter estimates for two individuals (top panels): the already-discussed subject 65, and subject 27 from the same dataset, jointly with the posterior uncertainty intervals. The estimated parameter values are fairly similar, although subject 65 displays lower likelihood-sensitivity. In addition, subject 65 has larger noise levels, as reflected by larger values of the noise standard deviation σ. Notice furthermore that this larger noise level is not only reflected in the larger value of σ, but also seeps through to all the other parameters, which are estimated with less precision (that is, they have larger uncertainty intervals). This is particularly evident for the utility curvature parameter ρ and the optimism parameter δ. This is indeed a common problem in PT estimations, since optimism and utility curvature capture similar motives, and can be difficult to tease apart in noisy data.

The bottom panels of Figure 3.9 plot the fit of the probability weighting functions to the non-parameteric data points for the same two subjects. The thin lighter lines constitute randomly selected draws from the Bayesian posterior, and represent the overall uncertainty surrounding the mean estimate of the probability weighting function. The draws for subject 37 stay relatively close to the mean estimate, but are still dispersed enough to encompass almost all the

nonparametric data points. The lines surrounding the mean estimate for subject 65, however, are very dispersed. This indicates a very low degree of confidence in the mean parameters, and is a symptom of the uncertainty attached to the single parameters, as already discussed above. The fact that these lines move up or down while staying almost parallel is an indication that the uncertainty affects mostly the optimism parameter δ, whereas likelihood-sensitivity γ is estimated with a much higher degree of confidence.

3.5.3 Batch estimation of individual parameters

One possibility to obtain individual-level data is to use the aggregate model from above, and to apply it individual-by-individual in a loop. That approach, however, is rather tedious. Here, we will take a different approach. We will carefully leverage priors to nudge individual-level estimations in the right direction, as illustrated above in the individual-level estimations. In addition, we will use a Stan model that yields fixed-effects estimates of all individual-level parameters in one go.

Below, we show the code of the Stan programme. The main trick is that we now estimate parameter vectors, the elements of which represent estimates of different subjects. Note that the identifier needs to start at 1 and comprise consecutive numbers for the code to work. If this is not the case, odd results may appear due to inconsistent mapping between IDs and parameter vectors (typically, the code will attempt to obtain estimates for a number of individuals equal to the maximum ID, with the values for missing individuals sampled from the priors). The priors as usual nudge the estimation into a region where we would expect the parameters to lie, without imposing any strong prior knowledge. [NOTE: here, as above, we limit ρ at 0 from below; a more general procedure would be to write conditional statements whereby $u(x) = x^{\rho}$ if $\rho > 0$, $u(x) = ln(x)$ if $\rho \equiv 0$, and $u(x) = -x^{\rho}$ if $\rho < 0$].

```
data {
  int<lower=1> N;
  int<lower=1> Nid;
  array[N] int id;
  array[N] real ce;
  array[N] real high;
  array[N] real low;
  array[N] real p;
}
parameters {
  vector<lower=0>[Nid] rho;
  vector<lower=0>[Nid] gamma;
  vector<lower=0>[Nid] delta;
  vector<lower=0>[Nid] sigma;
}
```

```
model {
  vector[N] pw;
  vector[N] pv;
  rho ~ normal( 1 , 1 );
  gamma ~ normal( 1 , 1 );
  delta ~ normal( 1 , 1 );
  sigma ~ normal( 1 , 1 );
for (i in 1:N){
    pw[i] = ( delta,* p[i]^gamma[id[i]] ) / ( delta[id[i]] * p[i]^gamma[id[i]] + ( 1 - p[i] )
    pv[i] = ( pw[i] * high[i]^rho[id[i]] + (1 - pw[i]) * low[i]^rho[id[i]] );
    ce[i] ~ normal( pv[i]^( 1/rho[id[i]] ) , sigma[id[i]] * (high[i] - low[i]) );
    }
}
```

```
d <- read_csv("data/data_RR_ch06.csv")
dg <- d %>% filter(z1 > 0) %>%
            mutate(high = z1) %>%
            mutate(low = z2) %>%
            mutate(prob = p1) %>%
            group_by(id) %>%
            mutate(id = cur_group_id()) %>%
            ungroup()

pt <- cmdstan_model("stancode/ind_PT_fixed.stan")

stanD <- list(N = nrow(dg),
              Nid = max(dg$id),
              id = dg$id,
              high = dg$high,
              low = dg$low,
              p = dg$prob,
              ce = dg$ce)

fe <- pt$sample(
  data = stanD,
  seed = 123,
  chains = 4,
  parallel_chains = 4,
  refresh=0,
  show_messages = TRUE,
  diagnostics = c("divergences", "treedepth", "ebfmi")
)
```

```
fe$save_object(file="stanoutput/ind_ZCH06_fixed.RDS")
```

```
fe <- readRDS("stanoutput/ind_ZCH06_fixed.RDS")

zz <- fe$summary(variables = c("rho","gamma","delta","sigma"),
                 "mean" , "sd")
z <- zz %>% separate(variable,c("[","]")) %>%
            rename( var = `[`) %>% rename(id = `]`) %>%
            pivot_wider(names_from = var , values_from = c(mean,sd))


ggplot(data = z,aes(x=mean_sigma,y=mean_gamma)) +
 geom_point(shape=1,color="blue") +
 geom_hline(yintercept=1, linewidth=1, color="red",lty="dashed") +
  geom_label_repel(aes(label = id),
                   box.padding   = 0.35,
                   point.padding = 0.5,
                   size          = 1.5,
                   segment.color = 'grey50') +
  xlab("noise SD") + ylab("likelihood-sensitivity")
```

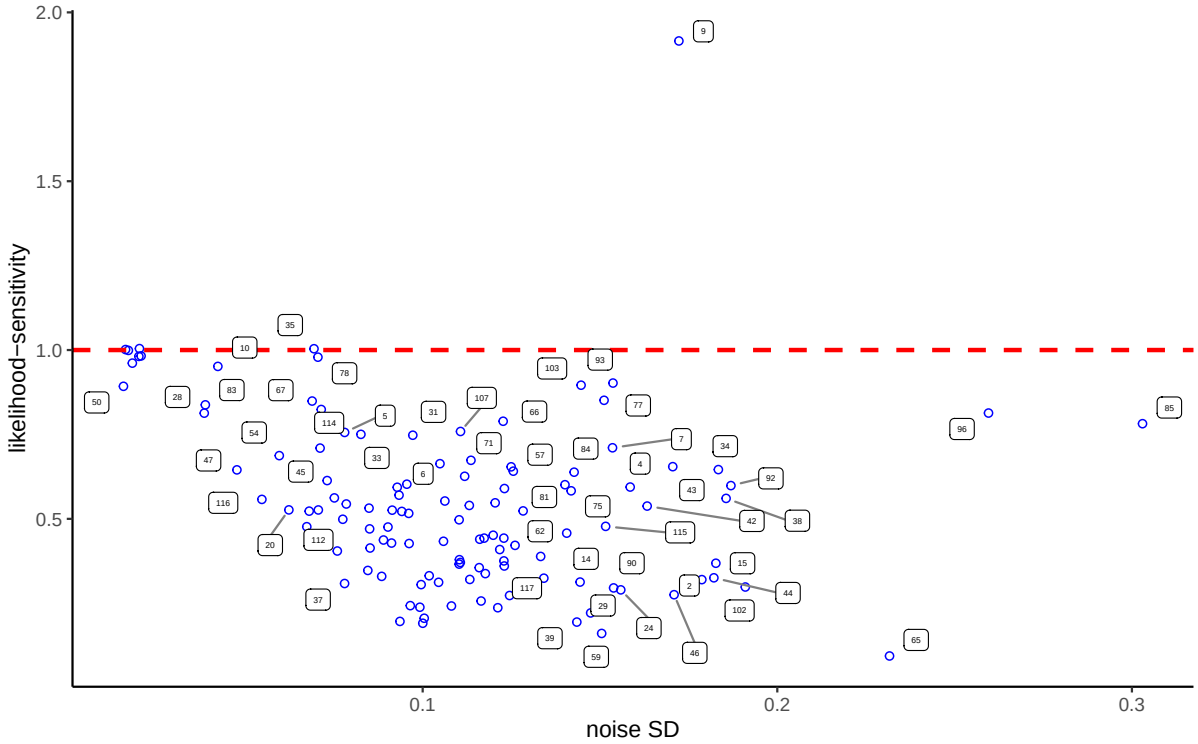


Figure 3.10: Scatter plot of noise annd likelihood-sennsitivity, fixed effects

Figure 3.10 shows a scatter plot of the noise SD against likelihood-sensitivity in the individual-level fixed effects estimates. Some patterns stand out in the data. A handful of individuals have sensitivity arbitrarily close to 1, combined with very low noise levels. These are the expected value maximizers described in the mixture model of Bruhin, Fehr-Duda, and Epper (2010). Overall, likelihood-sensitivity and noise appear to show a negative correlation. That being said, individuals 37 and 65, for whom we have examined the individual patterns above, both show low levels of likelihood-sensitivity, yet have very different levels of noise. We will re-examine some of these patterns shortly, in the context of hierarchical estimation. For now, note also subject 9—a subject with fairly large noise, and the only subject displaying likelihood-oversensitivity. There is no reasonn to think that the estimatte has not converged properly. Nevertheless, we should be careful before accepting such a large outlier (and possibly enntering this parameter into a regression). We will dscuss how to deal with this inn a principled way in the next chapter.

3.5.4 Regression of parameters

It may be tempting to build parameter regressions directly into an aggregate model. While technically possible, this would ignore that fact that we have a multiplicity of observations

for each individual, and would thus yield biased results. We can, however, easily estimate a regression using individual-level parameters such as mentioned above. A very simple approach would be to obtain the individual-level estimates, and then to use the means of the parameter estimates (or the medians for that matter) in a second-stage regression. The main problem with this sort of approach is that we ignore uncertainty surrounding the parameters. There are two ways around this. We can either estimate the regression in the same model estimating the parameters themselves, thus taking a long the entire posterior uncertainty about the parameters; or we can build an measurement error model that makes use of the second-order information summarizing the distribution of the parameters around their mean.

We will again use the Zurich 2006 data used above. The main model will be identical to the one used above. Assume now we want to regress only likelihood-senstivity γ on some background characteristics—say the gender and high income status of the subject. The trick will be to add a second loop after the first one. This loop is now defined at the level of the *individual*, thus eliminating the dimentionality problem mentioned above. Running the regression within the same model further ensures that all the parameter uncertainty is taken into account in the estimation. Note also that we need to define the design matrix at the level of the *inndividual*, so that we will first need to obtain data with only one observation per subject to create that matrix.

```
data {
  int<lower=1> N;
  int<lower=1> Nid;
  int<lower=1> k;
  array[N] int id;
  array[N] real ce;
  array[N] real high;
  array[N] real low;
  array[N] real p;
  matrix[Nid,k] x;

}
parameters {
  vector<lower=0>[Nid] rho;
  vector<lower=0>[Nid] gamma;
  vector<lower=0>[Nid] delta;
  vector<lower=0>[Nid] sigma;
  vector[k] beta;
  real<lower=0> tau;
}
model {
  vector[N] pw;
  vector[N] pv;
```

```
  rho ~ normal( 1 , 1 );
  gamma ~ normal( 1 , 1 );
  delta ~ normal( 1 , 1 );
  sigma ~ normal( 1 , 1 );
for (i in 1:N){
    pw[i] = ( delta[id[i]] * p[i]^gamma[id[i]] ) /
            ( delta[id[i]] * p[i]^gamma[id[i]] + ( 1 - p[i] )^gamma[id[i]] );
    pv[i] = ( pw[i] * high[i]^rho[id[i]] + (1 - pw[i]) * low[i]^rho[id[i]] );
    ce[i] ~ normal( pv[i]^( 1/rho[id[i]] ) , sigma[id[i]] * (high[i] - low[i]) );
    }
// regression:
for (n in 1:Nid){
 gamma[n] ~ student_t(3 , x[n] * beta , tau );
 }
}
```

```
d <- read_csv("data/data_RR_ch06.csv")
dg <- d %>% filter(z1 > 0) %>%
            mutate(high = z1) %>%
            mutate(low = z2) %>%
            mutate(prob = p1) %>%
            group_by(id) %>%
            mutate(id = cur_group_id()) %>%
            ungroup()

di <- dg %>% group_by(id) %>%
             filter(row_number()==1) %>%
             ungroup()

x <- model.matrix(~ female + highincome,di)


pt <- cmdstan_model("stancode/ind_PT_fixed_reg.stan")

stanD <- list(N = nrow(dg),
              Nid = max(dg$id),
              k = ncol(x),
              id = dg$id,
              high = dg$high,
              low = dg$low,
              p = dg$prob,
              ce = dg$ce,
```

```
              x = x)

fe <- pt$sample(
  data = stanD,
  seed = 123,
  chains = 4,
  parallel_chains = 4,
  refresh=0,
  show_messages = TRUE,
  diagnostics = c("divergences", "treedepth", "ebfmi")
)

fe$print(c("beta", "tau"), digits = 3,
    "mean", SE = "sd", "rhat" ,
    ~quantile(., probs = c(0.025, 0.975)))
fe$save_object(file="stanoutput/ind_ZCH06_fixed_reg.RDS")
```

```
fe <- readRDS("stanoutput/ind_ZCH06_fixed_reg.RDS")

fe$print(c("beta", "tau"), digits = 3,
    "mean", SE = "sd", "rhat" ,
    ~quantile(., probs = c(0.025, 0.975)))
```

```
 variable   mean    SE  rhat   2.5%  97.5%
  beta[1]  0.538 0.029 1.000  0.483  0.595
  beta[2] -0.098 0.040 1.001 -0.177 -0.017
  beta[3]  0.055 0.107 1.001 -0.149  0.268
  tau      0.158 0.017 0.999  0.127  0.194
```

The regression results show that women have lower likelihood-sensitvity thann men. There is no clear effect of high income status (which is not surprising if you look at he data—there are only very few subjects for which the dummy takes the value 1).

In general, you will typically run the regression for all parameters at once. One way of doing this is using an equation for each parameter inside the loop. A more efficient way of coding this may be to collect the parameters into a vector, and to estimate a matrix of parameters. This is left as an exercise. Let me conclude this discussion by noting that the code used here is very flexible. For instance, you could just as well use the parameters on the right-hand side of the equation. We will return to regression analysis when discussing hierarchical models in the next chapter.

3.5.5 Should one do individual-level estimation?

When estimating a data set like this, should one pool the data, implicitly assuming that all the choices come from one single agent? Or should one recur to individual-level estimations, thus treating every individual as completely distinct? A principled way of going about this question could be to compare *within-individual* variation in responses to *between-individual* variation. If between-individual variation is small relative to within-individual variation, then one could pool the data; if between-individual variation is large relative to within-individual variation, then one could estimate the parameters subject-by-subject. However, why not do both at once and let the endogenously-estimated variance components decide how much weight to put on individual versus aggregate estimates?

Exactly this is achieved in (Bayesian) hierarchical models. The model endogenously estimates between-subject variation, as well as within-subject variation for each subject. Individual-level parameters are then "shrunk" toward the aggregate mean in function of their noisiness (more noisy data are shrunk more) and of their distance to the aggregate mean (outliers are shrunk more than data close to the mean). Such models can indeed be shown to maximize out-of-sample predictive power by discounting noisy outliers. Alas, hierarchical models are typically still only mentioned in passing in econometrics textbooks aimed at economists. They do, however, constitute the workhorse statistical tool in other disciplines, such as educational sciences, psychology, epidemiology, and medicine. Although hierarchical models can be estimated by maximum likelihood routines using so-called *empirical Bayes* algorithms, they are inherently Bayesian. The aggregate-level estimate indeed serves as an endogenously-estimated prior for lower-level estimates. Economists ignore such models at their own risk, since neglecting hierarchical structure where it exists can lead to severe bias in the estimation results.

3.6 Stan workflow

It is important to build up a proper workflow in Stan. This involves many aspects. Two I would like to stress are gradual model building, and simulations.

The models we have seen thus far are exceedingly simple and do not need much particular attention. As we move to more complex (and more interesting!) models, however, it will be important to carefully build up the models step by step. The general recommendation is to build up the models gradually, starting from simpler models, and then only gradually adding additional parameters and more structure. This avoids thorny issues with models not converging for no apparent reason, or parameters that are not identified (this can be difficult to spot in Stan, since a completely unidentified parameter may simply converge to its prior; i.e. the parameter will be 'simulated' rather than being estimated, which can be easy to miss if priors are informative).

Another important topic is model convergence. This is covered in the manual and in many blog posts, and I will not disscuss this in depth. In general, hints of problems are 1) large `R_hat`

values (usually, anything above 1.05 should be considered as suspicious, and values above 1.1 should ring alarm bells); and especially 2) divergent iterations (these manifest in warnings post-estimation). While some divergent iterations will often occur in complex and high-dimensional non-linear problems, as a general rule these should always trigger a convergence check. Finally, low numbers of effective samples are also problematic and often indicate difficulties of the algorithm exploring the parameter space. Reparametrization of the model is often the solution (we will see some examples later; also consult the manual).

Stan uses a default of 1000 warmup iterations and 1000 samples per chain, and this will be plenty for most purposes if the model is well-specified. If the model is not efficient, just cranking up the samples will be of limited use. An exception when you may want to increase the number of samples is when examining a precise hypothesis concerning the tails of the distribution (e.g., whether the evidence in favour of a hypothesis is more or less than 95%). In cases that fall close to that value, this may well make a difference, since for typical distributions such as the normal there will be relatively few observations in the tails. In general, such precise cutoffs are not very relevant from a Bayesian point of view, but we all know that the average referee may well have a different take of this issue.

Simulations are an important part of the Stan workflow. An interesting feature in Stan is that it allows you to use the Stan code itself to generate simulations. In particular, data are then simulated starting from the priors you have specified, so that you may want to specify informative priors. Note, however, that simulating data from Stan does not necessarily replace traditional simulation in `R`. Simulations directly from Stan can be useful to check your model. However, recoverability of parameters is best tested using simulations from R, since writing two independent codes to simulate and recover the parameters reduces the likelihood of mistakes.

4 Hierarchical Bayesian Models

Hierarchical models are seldom discussed in much detail in econometrics textbooks used in economics. They are, however, essential tools in epidemiology, psychology, and education research. A large part of the reason for that is historical and down to path dependence. In panel data analysis, dummies at the lowest level of analysis have long been used to implement "fixed effects", thus allowing for clean identification and avoiding that estimated effects be contaminated by selection effects. This is an important goal, albeit one that can be easily obtained using alternative implementations providing more flexibility.

Hierarchies are extremely common. And while ignoring such hierarchies where they are relevant may result in biased estimators, imposing hierarchical structure where it is not relevant is (almost) costless: the estimator will simply collapse to its OLS equivalent. Economists thus ignore hierarchies at their own risk. Below, I will give a number of examples of cases that are best thought of as hierarchies. Classical cases are, for instance, measurement error models or meta-analysis, or stratified experimental or survey designs. Far from being a marginal or specialist topic, hierarchical modelling can be used to tackle a large variety of practical problems, such as for instance concerns about multiple testing that are often voiced in classical statistics.

Interestingly, the opposite concern of insufficient power in single studies can be addressed by exactly the same means, showing how both share a common origin (see discussion of the canonical 8 schools example). We will also see how hierarchical analysis is essential when trying to capture a complex sampling setup. I will provide an extended example based on panel data analysis. I will also show how stratified sampling setups ought to be reflected in the data analysis, and how explicitly modelling the data collection procedures can actually affect our conclusions (or rather, how ignoring such structure may lead to loss of information and biased inferences).

To see the value of hierarchical modelling in nonlinear estimation, imagine the following situation. You have collected choice data from a large set of individuals, each of whom has made a number of choices e.g. between lotteries. How should you analyze these data? Obvious responses that have been given to this question in the literature are: 1) we can assume that all choices come from a representative agent, and run an aggregate estimation; or 2) We treat each subject as a separate unit, and estimate the model for each of these units. We have seen both approaches in the last chapter. The problem is that aggregate estimation may ignore heterogeneity between agents, and individual-level estimation runs the risk of overfitting sparse noisy data.

A more principled approach may thus be to first assess the variance in responses for each agent, possibly after conditioning on a model (i.e., obtain the residual variance after fitting a given model). This corresponds to an individual-level estimate. Then compare the within subject variation to the variation in parameters across subjects. Little variation across subjects may then indicate the legitimacy of aggregate estimation, or vice versa. However, why not let the relative variances determine which weight each observation gets? That is exactly what hierarchical analysis does. It thus constitutes a principled way of compromising between aggregate and individual data patterns, which are taken into account in a way that is proportional to their relative precision (the inverse of the variance).

Yet a different intuition can be had if we think of the aggregate estimation as the prior. We have discussed in the previous chapter that defining a prior can be beneficial for individual-level estimation. One issue, however, concerned what prior we could reasonably adopt without "unduly" influencing the results. One answer may be to obtain an aggregate estimate across all individuals, and to then use this as a prior for the individual-level estimates. As we will see shortly, the hierarchical model does something very close to that, albeit taking into account the relative noisiness of different observations.

4.1 Hierarchical Models and Linear Regression

I will start by illustrating the workings of the random effects models using a simple linear regression. Here is the code for an aggregate regression model as a reminder:

```
data{
int<lower=0> N;
vector[N] ce;
vector[N] p;
}
parameters{
real alpha;
real beta;
real<lower=0> sigma;
}
model{
ce ~ normal(alpha + beta * p , sigma) ;
}
```

The code below implements a linear regression identical to the one seen in chapter 2 on my 30 country risk preference dataset (you can download the data from here). For simplicity, we will use only gains and lotteries with 0 lower outcomes. Let us regress the normalized certainty equivalent on the probability of winning the prize.

```
d30 <- read.csv(file="data/data_30countries.csv")
d30 <- d30 %>% filter(risk==1 & gain==1 & low==0) %>%
               mutate(ce_norm = equivalent/high)

# creates data to send to Stan
stanD <- list(N = nrow(d30),
              ce = d30$ce_norm,
              p = d30$probability )

# compile the Stan programme:
sr <- cmdstan_model("stancode/agg_normal.stan")

# run the programme:
nm <- sr$sample(
  data = stanD,
  seed = 123,
  chains = 4,
  parallel_chains = 4,
  refresh=0,
  init=0,
  show_messages = TRUE,
  diagnostics = c("divergences", "treedepth", "ebfmi")
)

nm$save_object(file="stanoutput/agg_30c.Rds")
```

We have now obtained the intercept as well as the slope (the coefficient of probability) and the noise term. The slope has an immediate interpretation as probabilistic sensitivity (or we could use $1-\beta$ as a measure of *insensitivity*). The intercept, on the other hand, has no natural interpretation. The good news is that we can easily use the two parameters to obtain other indices. In fact, in the Bayesian context we can work directly with the posterior to carry along all the parameter uncertainty, so that we do not lose any information, as we would in traditional analysis. Below, I show this for the insensitivity parameter, and for a parameter capturing pessimism, which is defined as $m = 1 - \beta - 2\alpha$.

```
nm <- readRDS(file="stanoutput/agg_30c.Rda")
print(nm, c("alpha","beta","sigma"),digits = 3)
```

```
Inference for Stan model: agg_normal.
4 chains, each with iter=2000; warmup=1000; thin=1;
post-warmup draws per chain=1000, total post-warmup draws=4000.
```

```
       mean se_mean    sd  2.5%  25%   50%   75% 97.5% n_eff  Rhat
alpha 0.142       0 0.003 0.135 0.14 0.142 0.144 0.148  1673 1.000
beta  0.704       0 0.006 0.692 0.70 0.704 0.708 0.716  1658 1.000
sigma 0.210       0 0.001 0.208 0.21 0.210 0.211 0.212  1948 1.002

Samples were drawn using NUTS(diag_e) at Mon Jun 13 20:59:31 2022.
For each parameter, n_eff is a crude measure of effective sample size,
and Rhat is the potential scale reduction factor on split chains (at
convergence, Rhat=1).
```

Treating all 3000 subjects across 30 countries as identical (or at least: drawn from one common population) may appear a stretch, yet this is the standard assumption made in aggregate models. Of course, one could go to the opposite extreme, and estimate the same regression subject-by-subject. The risk, in that case, is that we may over-fit noisy data given the relatively few observation points at the individual level, and the typically high levels of noise inherent in data of tradeoffs under risk.

4.1.1 Country-specific intercepts

To start, however, let us consider a rather obvious consideration given the context of the data—the subjects in the 30 countries are clearly drawn from different subject pools. Once we accept that logic, we should aim to capture it in our econometric model. One way of doing that is to allow the data to follow a hierarchical structure. That is, we can model the parameters as country-specific, while also being drawn from one over-reaching population.

To do that, let us start from a model in which only the intercept can vary country by country, so that we will write α_c, where the subscript reminds us that the intercept is now country-specific. At the same time, we will model the intercept as being randomly drawn from a larger sample, i.e. the sample of all countries, so that $\alpha_c \sim \mathcal{N}(\hat{\alpha}, \tau)$, where $\hat{\alpha}$ is the aggregate intercept.The Stan model used to estimate this looks as follows:

```
data{
int<lower=0> N; //size of dataset
int<lower=0> Nc; //number of countries
vector[N] ce; // normalized CE, ce/x
vector[N] p; // probability of winning
array[N] int ctry; //unique and sequential integer identifying country
}
parameters{
real alpha_hat;
vector[Nc] alpha;
```

```
real beta;
real<lower=0> sigma;
real<lower=0> tau;
}
model{
// no prior for aggregate parameters (empirical Bayes)

//prior with endogenous aggregate parameters
alpha ~ normal(alpha_hat, tau);

//regression model with country-level intercepts
ce ~ normal( alpha[ctry] + beta * p , sigma  );
}
```

In this particular instance, I have programmed the aggregate parameter estimate as a prior (while not specifying any hyperpriors on the aggregate-level parameters for the time being). This way of writing the code is not a coincidence, and drives home the point that the aggregate-level estimates in hierarchical models *serve as an endogenously-estimated prior for the lower-level estimates*. That is, we can exploit the power of the global dataset to obtain the best possible estimate of a prior, and then let this aggregate prior inform the individual-level estimates, which will often be based on much less information (we will see an example of this shortly).

```
# creates data to send to Stan
stanD <- list(N = nrow(d30),
              ce = d30$ce_norm,
              p = d30$probability,
              Nc = length(unique(d30$country)),
              ctry = d30$country)


# compile the Stan programme:
sr <- cmdstan_model("stancode/ri_30countries.stan")

# run the programme:
nm <- sr$sample(
  data = stanD,
  seed = 123,
  chains = 4,
  parallel_chains = 4,
  refresh=200,
  init=0,
```

```
  show_messages = TRUE,
  diagnostics = c("divergences", "treedepth", "ebfmi")
)

nm$save_object(file="stanoutput/ri_30c.Rds")
```

```
ri <- readRDS("stanoutput/ri_30c.Rds")
print(ri, c("alpha_hat","tau","sigma","alpha"),digits = 3)
```

```
  variable  mean median    sd   mad    q5   q95  rhat ess_bulk ess_tail
 alpha_hat 0.143  0.143 0.007 0.007 0.131 0.155 1.000     3200     3264
 tau       0.036  0.036 0.005 0.005 0.029 0.046 1.001     6875     2628
 sigma     0.207  0.207 0.001 0.001 0.206 0.209 1.001     6618     2856
 alpha[1]  0.108  0.107 0.008 0.009 0.094 0.121 1.000     4521     2707
 alpha[2]  0.128  0.128 0.007 0.007 0.116 0.140 1.000     4007     2874
 alpha[3]  0.158  0.158 0.008 0.008 0.145 0.171 1.001     4446     2518
 alpha[4]  0.145  0.145 0.008 0.008 0.133 0.158 1.000     4794     3195
 alpha[5]  0.106  0.106 0.007 0.007 0.094 0.118 1.000     3942     2847
 alpha[6]  0.129  0.129 0.005 0.005 0.120 0.138 1.001     2987     3288
 alpha[7]  0.130  0.130 0.006 0.006 0.120 0.141 1.000     3899     3212

 # showing 10 of 33 rows (change via 'max_rows' argument or 'cmdstanr_max_rows' option)
```

```
p <- as_draws_df(ri)
zz <- ri$summary(variables = c("alpha"), "mean" , "sd")
z <- zz %>% separate(variable,c("[","]")) %>%
            rename( var = `[`) %>% rename(nid = `]`) %>%
            pivot_wider(names_from = var , values_from = c(mean,sd))

ggplot() +
  geom_abline(intercept = z$mean_alpha , slope = mean(p$beta), color="cornflowerblue") +
  geom_abline(intercept = mean(p$alpha_hat) , slope = mean(p$beta), color="navy" , linewidth=
  geom_abline(intercept = 0 , slope = 1 , color="red") +
  xlim(0,1) + ylim(0,1) + labs(x="probability",y="probability weighting")
```

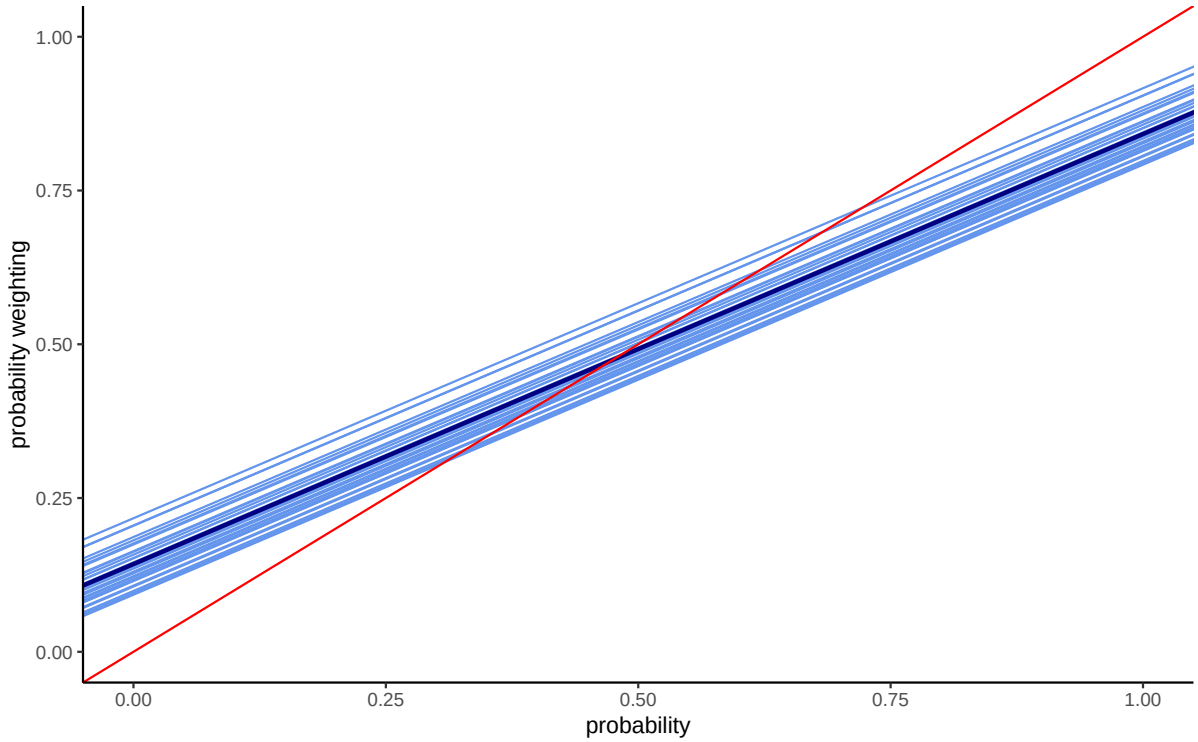


Figure 4.1: Coountry-specific hierarchical intercepts

The mean intercept turns out to be the same as before, although this is not necessarily the case, given that the weight for different observations can shift due to the hierarchical structure. The variance is smaller, since we now capture more of the heterogeneity. And indeed, we find significant variation in intercepts between countries, as shown by the value of τ. The model thus results in a series of parallel lines with different elevations, as shown in Figure 4.1. In terms of interpretation, we thus find very different levels of optimism across the 30 countries.

4.1.2 Random intercepts, and random slopes

Once we have allowed for intercepts to differ between countries, however, there is really no reason to force the slope to be the same. We can thus modify the model to allow for a hierarchical slope parameter (sometimes called, *random intercepts, random slopes* model) in addition to the random intercepts (or hierarchical intercepts—the term "random effects" is best avoided if one wants to publish results in economics, since it triggers memories of panel data models that "one should never use" in economics). Further adding some regularizing hyperpriors to help speeding the estimation along, the Stan code will look as follows:

```
data{
  int<lower=0> N;
  int<lower=0> Nc;
  vector[N] ce;
  vector[N] p;
  array[N] int ctry;
}
parameters{
  real alpha_hat;
  real beta_hat;
  vector[Nc] alpha;
  vector[Nc] beta;
  real<lower=0> sigma;
  real<lower=0> tau_a;
  real<lower=0> tau_b;
}
model{
// hyperpriors:
alpha_hat ~ std_normal();
beta_hat ~ normal(1,1);
tau_a ~ normal(0,2);
tau_b ~ normal(0,2);

// hierarchical intercepts
  alpha ~ normal(alpha_hat, tau_a);

// hierarchical slopes
  beta ~ normal(beta_hat, tau_b);

// likelihood
for ( i in 1:N )
  ce[i] ~ normal( alpha[ctry[i]] + beta[ctry[i]] * p[i] , sigma  ) ;
}
```

The code looks much like the one before, with one hierarchical parameter added. However, I now loop through the observations using the `for ( i in 1:N )` loop. This avoids issues of dimensionality when multiplying the parameter vector `beta` (30x1) with the data vector p (Nx1). We can now estimate the model like before. As a note of caution, this model may run for a little while, depending on the clock speed of your processors. It may thus be desirable to first examine whether we can speed up the model in any way. One possibility may be to parallelize the model to speed it up, but I will not do this here for the sake of clarity of the code. However, a much easier way to speed it up is available by rewriting the loop through the

likelihood in vector notation. given the different dimensionality of `beta`and `p`, this will require using a "dot-product", but it is straightforward to implement by replacing the loop with the following line:

```
ce ~ normal(alpha[ctry] + beta[ctry] .* p, sigma);
```

This simple change to vectorizing the model means that it now runs in about 40% of the time required for the model with the loop.

```
# creates data to send to Stan
stanD <- list(N = nrow(d30),
              ce = d30$ce_norm,
              p = d30$probability,
              Nc = length(unique(d30$country)),
              ctry = d30$country)

# compile the Stan programme:
sr <- cmdstan_model("stancode/rirs_30countries_vec.stan")

# run the programme:
nm <- sr$sample(
  data = stanD,
  seed = 123,
  chains = 4,
  parallel_chains = 4,
  refresh=200,
  init=0,
  show_messages = TRUE,
  diagnostics = c("divergences", "treedepth", "ebfmi")
)

print(nm, c("alpha_hat","beta_hat","tau_a","tau_b","sigma","alpha","beta"),digits = 3)
nm$save_object(file="stanoutput/rirs_30c.Rds")
```

```
rirs <- readRDS("stanoutput/rirs_30c.Rds")
print(rirs, c("alpha_hat","beta_hat","tau_a","tau_b","sigma","alpha","beta"),digits = 3)
```

```
  variable  mean median    sd   mad    q5   q95  rhat ess_bulk ess_tail
 alpha_hat 0.134  0.134 0.015 0.015 0.109 0.159 1.000     6808     2995
 beta_hat  0.718  0.718 0.024 0.025 0.678 0.758 1.001     6521     2939
 tau_a     0.081  0.080 0.012 0.011 0.064 0.102 1.001     7768     3153
```

```
 tau_b     0.128  0.127 0.019 0.018 0.102 0.163 1.001     7043     3126
 sigma     0.205  0.205 0.001 0.001 0.203 0.206 1.003     5920     2485
 alpha[1]  0.076  0.076 0.020 0.021 0.043 0.110 1.000     7736     3209
 alpha[2]  0.085  0.085 0.017 0.016 0.057 0.112 1.005     7977     2554
 alpha[3]  0.120  0.120 0.018 0.018 0.091 0.148 1.004     7742     2915
 alpha[4]  0.187  0.187 0.018 0.018 0.158 0.216 1.001     7542     2900
 alpha[5]  0.124  0.124 0.016 0.016 0.097 0.150 1.001     8002     3205

 # showing 10 of 65 rows (change via 'max_rows' argument or 'cmdstanr_max_rows' option)
```

```
p <- as_draws_df(rirs)
zz <- rirs$summary(variables = c("alpha","beta"), "mean" , "sd")
z <- zz %>% separate(variable,c("[","]")) %>%
            rename( var = `[`) %>% rename(nid = `]`) %>%
            pivot_wider(names_from = var , values_from = c(mean,sd))
ggplot() +
  geom_abline(intercept = z$mean_alpha , slope = z$mean_beta, color="cornflowerblue") +
  geom_abline(intercept = mean(p$alpha_hat) , slope = mean(p$beta_hat), color="navy" , linew
  geom_abline(intercept = 0 , slope = 1 , color="red") +
  xlim(0,1) + ylim(0,1) + labs(x="probability",y="probability weighting")
```

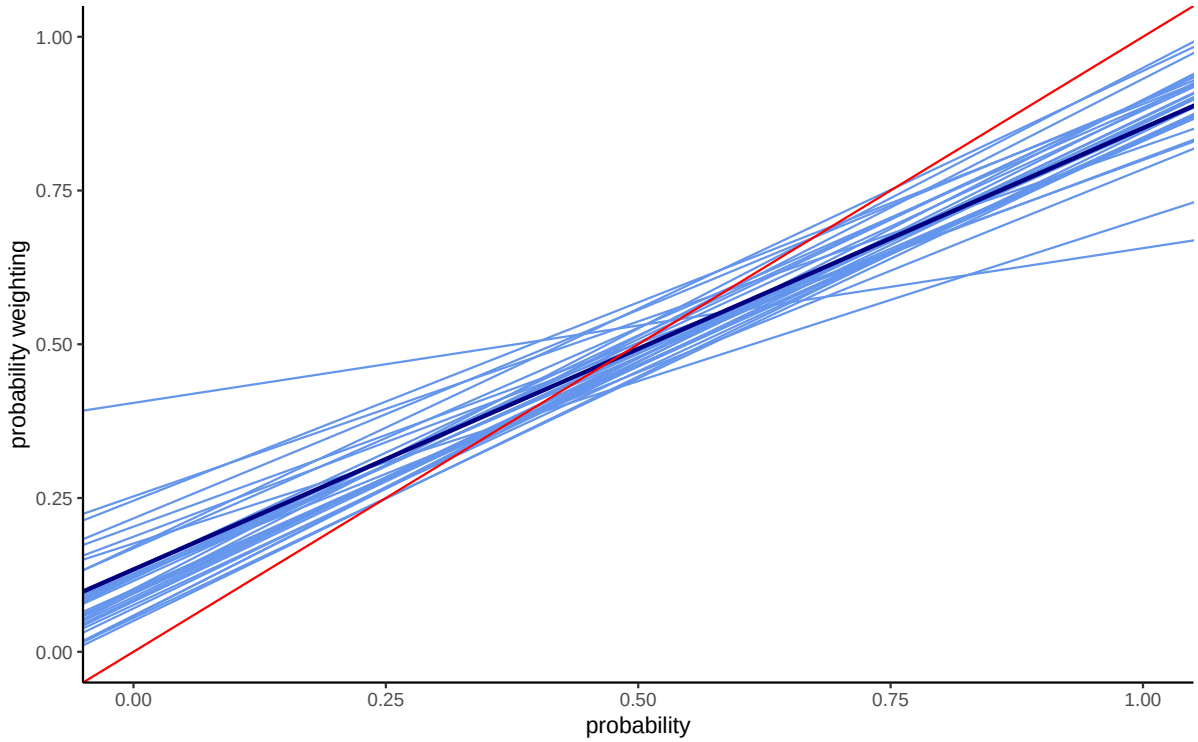


Figure 4.2: Country-specific hierarchical intercepts and slopes

Several things are noteworthy in the results shown in Figure 4.2. We now capture considerable heterogeneity in both the intercept and the slope. The lines thus no longer run parallel to each other. Indeed, providing more flexibility for the slope has *increased* the variation in intercepts. At the same time, the average intercept has now changed, too, and is somewhat lower than before.

It seems natural to ask whether any difference exists between estimating the global hierarchical model, or estimating the model country by country, or indeed using a fixed effects estimator, which amounts to the same thing (can you see how to implement such a model starting from the code above?). Below, I compare the country-level estimates obtained from the two approaches.

```
fe <- readRDS("stanoutput/fifs_30c.Rds")
re <- readRDS("stanoutput/rirs_30c.Rds")

p <- as_draws_df(re)

zz <- re$summary(variables = c("alpha","beta"), "mean" , "sd")
zr <- zz %>% separate(variable,c("[","]")) %>%
```

```
            rename( var = `[`) %>% rename(nid = `]`) %>%
            pivot_wider(names_from = var , values_from = c(mean,sd))

zz <- fe$summary(variables = c("alpha","beta"), "mean" , "sd")
zf <- zz %>% separate(variable,c("[","]")) %>%
            rename( var = `[`) %>% rename(nid = `]`) %>%
            pivot_wider(names_from = var , values_from = c(mean,sd))

ggplot() +
  geom_point(aes(x=zf$mean_alpha,y=zf$mean_beta,color="Fixed effects estimates"),shape=1,size
   geom_point(aes(x=zr$mean_alpha,y=zr$mean_beta,color="Bayesian hierarchical estimates"),sha
   scale_colour_manual("Legend", values = c("darkgoldenrod","steelblue3","#999999")) +
    geom_vline(xintercept=mean(p$alpha_hat),color="gray",lty="dashed",lwd=1) +
     geom_hline(yintercept=mean(p$beta_hat),color="gray",lty="dashed",lwd=1) +
  labs(x="intercept",y="sensitivity") +
   theme(legend.position = c(0.75, 0.85))
```

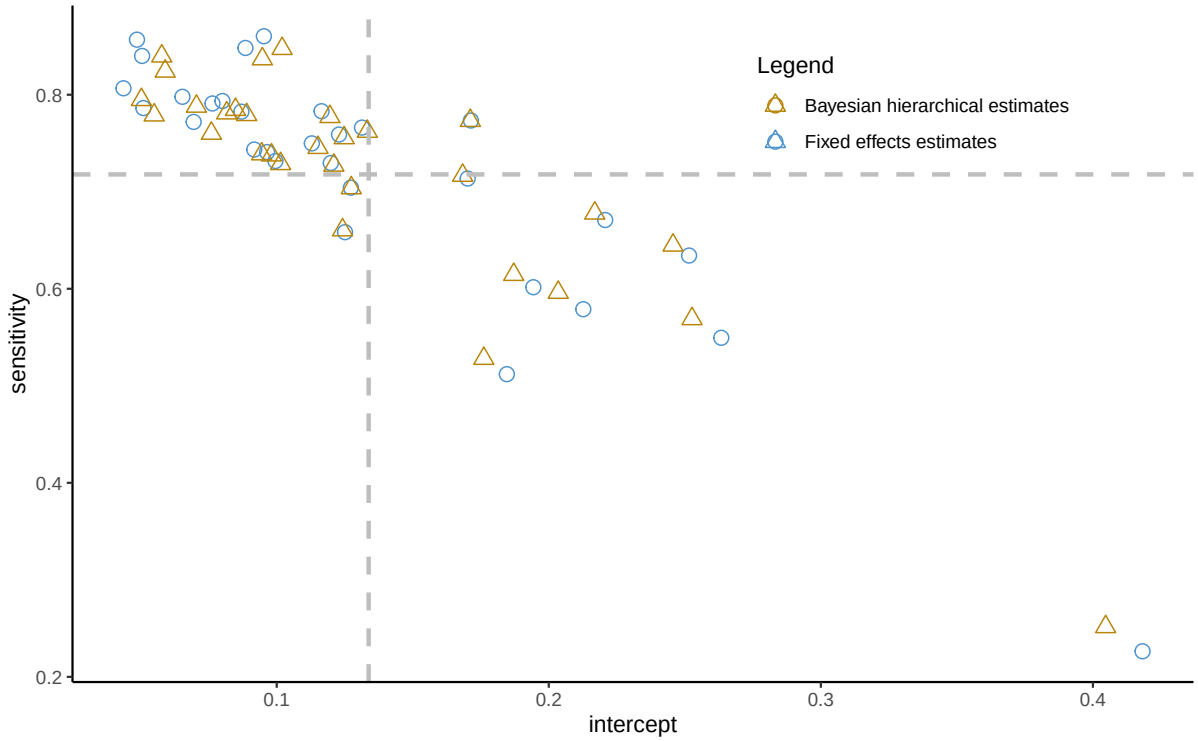


Figure 4.3: Fixed effects versus hierarchical estimates

Some striking patterns become apparent in Figure 4.3. For one, the intercept and slope

parameters show a strong, negative correlation—a point to which we will return shortly. More interestingly for the question at hand, the estimates from the hierarchical model are not the same as the ones from the fixed effects model. The difference is indeed systematic, with hierarchical estimate being less dispersed about the mean values, indicated by the vertical and horizontal lines. This is shrinkage at work. That is, estimates that fall far from the mean will be shrunk or pooled towards their mean estimate, with the shrinkage increasing in the noisiness of the data. This process follows the exact same equation as the regression to mean observed previously, and once again illustrates the influence of the prior, except that the latter is now endogenously estimated from the aggregate data.

4.2 Hierarchies of Parameters

We are now ready to apply the elements seen above to the estimation of nonlinear models. In principle, this application constitutes a straightforward extension of what we have already seen. In practice, however, the estimation of multiple parameters at once—which are often highly constrained to boot—may pose problems for Stan to explore. Luckily, the code can be written using transformed parameters in such a way as to make it much more efficient.

4.2.1 A simple hierarchical model

Let us start from a simple EU model, in which only the utility curvature parameter is hierarchical.

```
data {
 int<lower=1> N;
 int<lower=1> Nid;
 array[N] int id;
 array[N] real ce;
 array[N] real high;
 array[N] real low;
 array[N] real p;
}
parameters {
 vector[Nid] rho;
 real rho_hat;
 real<lower=0> tau;
 real<lower=0> sigma;
}
model {
 vector[N] uh;
```

```
 vector[N] ul;
 vector[N] pv;

 rho_hat ~ normal(1,2); // hyperprior for the mean of utility curvature
 tau ~ cauchy(0,5); // hyperprior of variance of rho
 sigma ~ cauchy(0,5); // hyperprior for (aggregate) residual variance

for (n in 1:Nid)
 rho[n] ~ normal(rho_hat , tau); // hierarchical model

for (i in 1:N){
 uh[i] = high[i]^rho[id[i]];
 ul[i] = low[i]^rho[id[i]];
 pv[i] = p[i] * uh[i] + (1-p[i]) * ul[i];
 ce[i] ~ normal( pv[i]^(1/rho[id[i]]) , sigma * high[i] );
 }
}
```

I use the data from the 2006 Zurich experiment in Bruhin, Fehr-Duda, and Epper (2010) to illustrate the model:

```
d <- read_csv("data/data_RR_ch06.csv")
dg <- d %>% filter(z1 > 0) %>%
            mutate(high = z1) %>%
            mutate(low = z2) %>%
            mutate(prob = p1) %>%
            group_by(id) %>%
            mutate(id = cur_group_id()) %>%
            ungroup()

pt <- cmdstan_model("stancode/hm_EU.stan")

stanD <- list(N = nrow(dg),
              Nid = max(dg$id),
              id = dg$id,
              high = dg$high,
              low = dg$low,
              p = dg$prob,
              ce = dg$ce)

hm <- pt$sample(
  data = stanD,
```

```
  seed = 123,
  chains = 4,
  parallel_chains = 4,
  refresh=200,
  show_messages = FALSE,
  diagnostics = c("divergences", "treedepth", "ebfmi")
)

hm$save_object(file="stanoutput/rr_ch06.Rds")
```

```
hm <- readRDS("stanoutput/rr_ch06.Rds")

# print aggregate parameters
hm$print(c("rho_hat","tau","sigma"), digits = 3,
          "mean", SE = "sd" , "rhat",
         ~quantile(., probs = c(0.025, 0.975)),max_rows=20)
```

```
 variable  mean    SE  rhat  2.5% 97.5%
  rho_hat 1.014 0.039 1.000 0.943 1.094
  tau     0.358 0.033 1.022 0.299 0.429
  sigma   0.134 0.002 1.004 0.130 0.137
```

```
# recover individual-level estimates
zz <- hm$summary(variables = c("rho"), "mean" , "sd" )
z <- zz %>% separate(variable,c("[","]")) %>%
            rename( var = `[`) %>% rename(id = `]`) %>%
            pivot_wider(names_from = var , values_from = c(mean,sd))

# plot distribution
ggplot(z,aes(x=mean_rho)) +
  geom_line(stat="density",color="steelblue3",lwd=1) +
  geom_vline(xintercept = 1, color="red", lty="dashed") +
  xlab("power utility parameter across subjects")
```

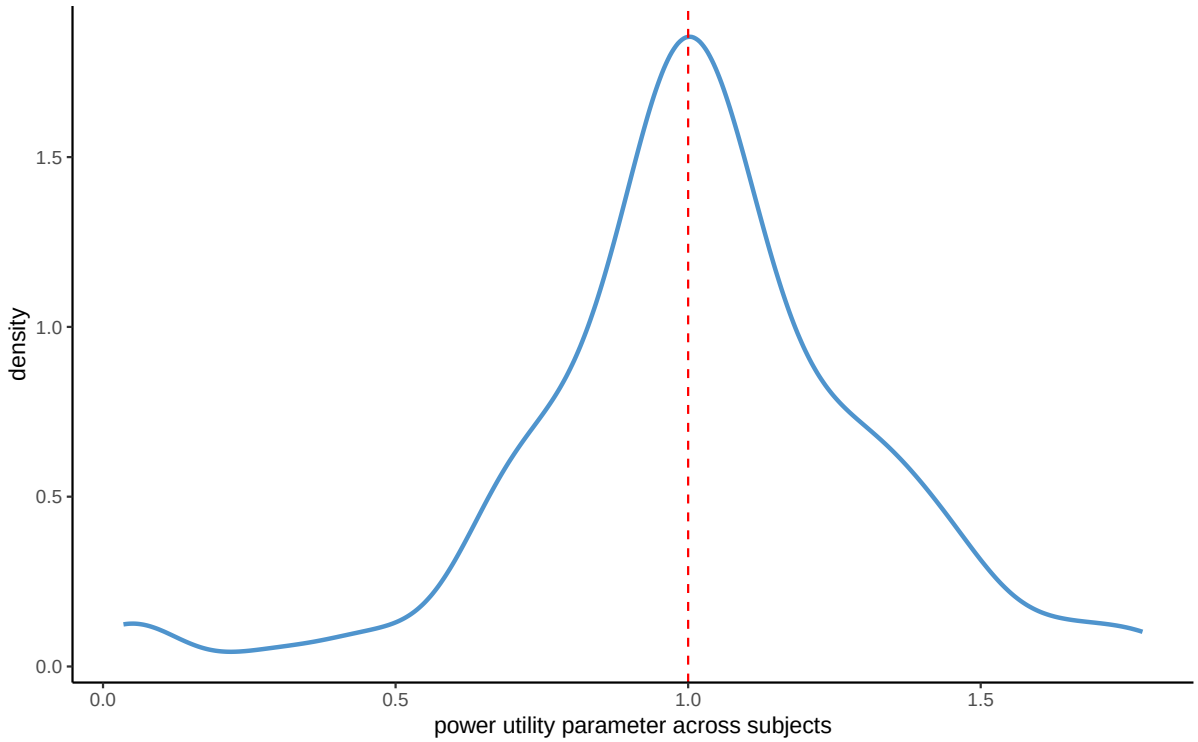


Figure 4.4: Distribution of Power utility parameter

The model above converges readily (it takes about 30 seconds on my laptop). No divergences are produced, and the R-hat statistics look reasonable. It also seems like we are getting a reasonable distribution of parameters, as shown in Figure 4.4. While in practice you will want to look at the standard deviations of the parameters to make sure that they are sufficiently tightly estimated, we will skip this here. Instead, we will consider an alternative way of programming the same model.

4.2.2 Rescaling parameters

The issue is that the model above does not always scale well to multiple parameters. Chances are that even at 2 parameters it will already become problematic. Once you have 3 or more hierarchical parameters, a simple generalization of the model used above will rarely fare well. The model may also behave less well when the parameter to be estimated is highly constrained and many observations fall close to the constraint. The solution in such cases is to reparameterize the model to help Stan explore the posterior distribution. Here, I show how to use a parameter rescaled to follow a standard-normal distribution for the one-parameter model above, before moving on to more complex models.

```
data {
 int<lower=1> N;
 int<lower=1> Nid;
 array[N] int id;
 array[N] real ce;
 array[N] real high;
 array[N] real low;
 array[N] real p;
}
parameters {
 real rho_hat;
 real<lower=0> tau;
 real<lower=0> sigma;
 array[Nid] real rho_z; //parameter on standard-normal scale
}
transformed parameters{
 vector[Nid] rho;

for (n in 1:Nid){
 rho[n] = rho_hat + tau * rho_z[n]; // mean plus var times rescaled par.
 }
}
model {
 vector[N] uh;
 vector[N] ul;
 vector[N] pv;

 rho_hat ~ normal(1,2);
 tau ~ cauchy(0,5);
 sigma ~ cauchy(0,5);

for (n in 1:Nid)
 rho_z[n] ~ std_normal( ); //prior for rescaled parameters

for (i in 1:N){
 uh[i] = high[i]^rho[id[i]];
 ul[i] = low[i]^rho[id[i]];
 pv[i] = p[i] * uh[i] + (1-p[i]) * ul[i];
 ce[i] ~ normal( pv[i]^(1/rho[id[i]]) , sigma * high[i] );
 }
}
```

You can see the re-parameterized model above. The data block has remained unchanged. The parameter block now contains a new variable `rho_z` instead of `rho`, which `rho` instead being declared in the `transformed parameters` block. Notice also that I now specify `rho_z` as an array instead of a vector. This is not hugely important here, but will make the code more efficient down the road. A second—and indeed crucial—point is that, other than `rho` above, `rho_z` is now rescaled: relative to the parameter `rho`, it is demeaned and multiplied by the standard deviation of the parameter, which ensures that it will follow a standard-normal distribution (which is indeed used as a prior in the `model` block). Given this new writing of the model, the aggregate parameters are now specified in the `transformed parameters` block. While this way of writing the model may seem backwards at first, it can yield substantial improvements in estimation speed, and for more complex models, may well make the difference between a model that converges and one that does not.

```
d <- read_csv("data/data_RR_ch06.csv")

dg <- d %>% filter(z1 > 0) %>%
            mutate(high = z1) %>%
            mutate(low = z2) %>%
            mutate(prob = p1) %>%
            group_by(id) %>%
            mutate(id = cur_group_id()) %>%
            ungroup()

pt <- cmdstan_model("stancode/hm_EU_resc.stan")

stanD <- list(N = nrow(dg),
              Nid = max(dg$id),
              id = dg$id,
              high = dg$high,
              low = dg$low,
              p = dg$prob,
              ce = dg$ce)

hm <- pt$sample(
  data = stanD,
  seed = 123,
  chains = 4,
  parallel_chains = 4,
  refresh=500,
  show_messages = FALSE,
  diagnostics = c("divergences", "treedepth", "ebfmi")
)
```

```
hm$save_object(file="stanoutput/rr_ch06_resc.Rds")
```

```
hm <- readRDS("stanoutput/rr_ch06.Rds")

# print aggregate parameters
hm$print(c("rho_hat","tau","sigma"), digits = 3,
          "mean", SE = "sd" , "rhat",
         ~quantile(., probs = c(0.025, 0.975)),max_rows=20)
```

```
 variable  mean    SE  rhat  2.5% 97.5%
  rho_hat 1.014 0.039 1.000 0.943 1.094
  tau     0.358 0.033 1.022 0.299 0.429
  sigma   0.134 0.002 1.004 0.130 0.137
```

```
# recover individual-level estimates
zz <- hm$summary(variables = c("rho"), "mean" , "sd" )
z <- zz %>% separate(variable,c("[","]")) %>%
            rename( var = `[`) %>% rename(id = `]`) %>%
            pivot_wider(names_from = var , values_from = c(mean,sd))

# plot distribution
ggplot(z,aes(x=mean_rho)) +
  geom_line(stat="density",color="steelblue3",lwd=1) +
  geom_vline(xintercept = 1, color="red", lty="dashed") +
  xlab("power utility parameter across subjects")
```

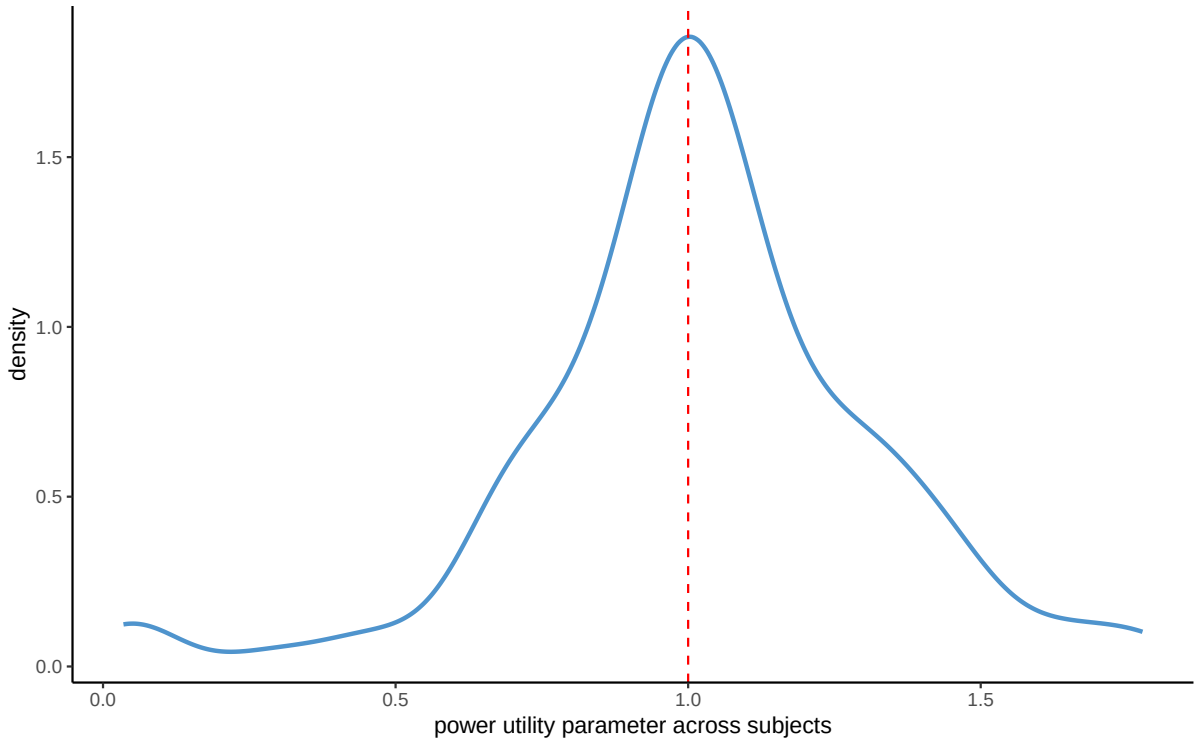


Figure 4.5: Distribution of Power utility parameter

Let us go ahead and estimate the model now. Again, there are no divergences, and the R-hats on the aggregate level parameters look good. The distribution of the individual-level means looks much like before. The model now converges in some 20 seconds on my laptop, instead of the 34 seconds of the previous model. While this improvement may not look like much, it amounts to close to a 40% speed gain. Once we are dealing with models that take hours or days to converge, that can make a real difference. The distribution of individual-level parameters in Figure 4.5 looks *almost* exactly the same as before.

4.2.3 Generalization to multiple parameters

With the decentralized model above in hand, it is now straightforward to build a model with multiple hierarchical parameters. Let us again use the same data, and let us build a PT model, restricted to gains only for simplicity (adding additional parameters for losses is straightforward). First, we will want to change the model to include a probability weighting function. Making the error term hierarchical, too, yields 4 hierarchical parameters.

Next, we get to work on the transformed parameters block. Instead of using an array `rho`, I introduce an element `pars` made up of an array of vectors. This will allow us to loop through vectors of the 4 parameters in the subsequent code. Do not forget to actually create the

vectors of parameters. In place of `rho_hat`, we now use a 4-dimensional vector called `means`, since the dimensionality of the different elements in the loop needs to line up for the code to work. I now furthermore restrict all parameters to be positive (although for `rho` this is ot strictly necessary) by defining them as the exponential of the corresponding element in `pars`. Finally, don't forget to give hyperpriors to the aggregate means, keeping in mind that they are expressed on a transformed scale.

Since we have 4 parameters, `tau` now becomes a vector of parameter variances. The `diag_matrix` function maps this vector into a matrix, i.e. it creates a matrix $\tau \times$ I, where I is an indentity matrix with the same number of rows and columns as hierarchical parameters in the model. What used to be `rho_z` becomes an array of vectors, which we simply call `Z`. Note that it needs to be indexed by the subject identifier `n` in the loop, which extracts one vector at the time, thus ensuring again that all expressions have the same dimension. We then obtain the original parameters of interest as usual, by defining them based on the transformed parameter line, and applying the appropriate constraints. Note that we could of course also define every parameter using a separate line, following the code used for utility curvature above. Using matrix notation like done here, however, makes the code more compact and prepares the ground for the estimation of covariance matrices of the parameters, which we will develop shortly.

```
data {
 int<lower=1> N;
 int<lower=1> Nid;
 array[N] int id;
 array[N] real ce;
 array[N] real high;
 array[N] real low;
 array[N] real p;
}
parameters {
 vector[4] means;
 vector<lower=0>[4] tau;
 array[Nid] vector[4] Z;
}
transformed parameters{
 array[Nid] vector[4] pars;
 vector<lower=0>[Nid] rho;
 vector<lower=0>[Nid] gamma;
 vector<lower=0>[Nid] delta;
 vector<lower=0>[Nid] sigma;
for (n in 1:Nid){
 pars[n] = means + diag_matrix(tau) * Z[n];
 rho[n] = exp( pars[n,1] );
```

```
 gamma[n] = exp( pars[n,2] );
 delta[n] = exp( pars[n,3] );
 sigma[n] = exp( pars[n,4] );
 }
}
model {
 vector[N] uh;
 vector[N] ul;
 vector[N] pw;
 vector[N] pv;

 means[1] ~ normal(0,1);
 means[2] ~ normal(0,1);
 means[3] ~ normal(0,1);
 means[4] ~ normal(0,1);
 tau ~ exponential(5);

for (n in 1:Nid)
 Z[n] ~ std_normal( );

for (i in 1:N){
 uh[i] = high[i]^rho[id[i]];
 ul[i] = low[i]^rho[id[i]];
 pw[i] = (delta[id[i]] * p[i]^gamma[id[i]])/
         (delta[id[i]] * p[i]^gamma[id[i]] + (1 - p[i])^gamma[id[i]]);
 pv[i] = pw[i] * uh[i] + (1-pw[i]) * ul[i];
 ce[i] ~ normal( pv[i]^(1/rho[id[i]]) , sigma[id[i]] * (high[i] - low[i]) );
 }
}
```

```
d <- read_csv("data/data_RR_ch06.csv")

dg <- d %>% filter(z1 > 0) %>%
            mutate(high = z1) %>%
            mutate(low = z2) %>%
            mutate(prob = p1) %>%
            group_by(id) %>%
            mutate(id = cur_group_id()) %>%
            ungroup()

pt <- cmdstan_model("stancode/hm_RDU_var.stan")
```

```
stanD <- list(N = nrow(dg),
              Nid = max(dg$id),
              id = dg$id,
              high = dg$high,
              low = dg$low,
              p = dg$prob,
              ce = dg$ce)

hm <- pt$sample(
  data = stanD,
  seed = 123,
  chains = 4,
  parallel_chains = 4,
  refresh=200,
  adapt_delta = 0.99,
  max_treedepth = 15,
  show_messages = FALSE,
  diagnostics = c("divergences", "treedepth", "ebfmi")
)
#257.0 seconds

hm$save_object(file="stanoutput/RR_ch6_RDU_var.Rds")
```

We are now ready to estimate the model. To do this, it will be wise to lower the stepsize to small values using the `adapt_delta` option—a value of 0.99 is typically recommended. At the same time, let us allow for a larger value of `max_treedepth` (the default is 10), which means that the algorithm may explore for longer. Both these tunings are made necessary by the highly nonlinear nature of the model, which creates parameter regions that are tricky to explore. Even so, such models may on occasion produce a small number of divergences. This is not unusual in highly nonlinear models such as this one. You would nevertheless be well advised to check the convergence statistics in depth, especially for new models that you have never used before.

```
hm <- readRDS("stanoutput/RR_ch6_RDU_var.Rds")

# print aggregate parameters
hm$print(c("means","tau"), digits = 3,
          "mean", SE = "sd" , "rhat",
         ~quantile(., probs = c(0.025, 0.975)),max_rows=20)
```

```
 variable   mean    SE  rhat   2.5%  97.5%
```

```
means[1] -0.068 0.020 1.002 -0.109 -0.031
means[2] -0.773 0.041 1.002 -0.852 -0.694
means[3] -0.171 0.032 1.001 -0.235 -0.108
means[4] -2.364 0.050 1.003 -2.462 -2.264
tau[1]    0.083 0.035 1.012  0.011  0.149
tau[2]    0.418 0.034 1.001  0.358  0.489
tau[3]    0.290 0.025 1.001  0.243  0.341
tau[4]    0.523 0.040 1.002  0.451  0.608
```

```r
# recover individual-level estimates
zz <- hm$summary(variables = c("rho","gamma","delta","sigma"), "mean" , "sd" )
z <- zz %>% separate(variable,c("[","]")) %>%
            rename( var = `[`) %>% rename(id = `]`) %>%
            pivot_wider(names_from = var , values_from = c(mean,sd))

# plot distribution
ggplot(z,aes(x=mean_gamma)) +
  geom_line(stat="density",color="steelblue3",lwd=1) +
  geom_vline(xintercept = 1, color="red", lty="dashed") +
  xlab("likelihood-insensitivity across subjects")
```

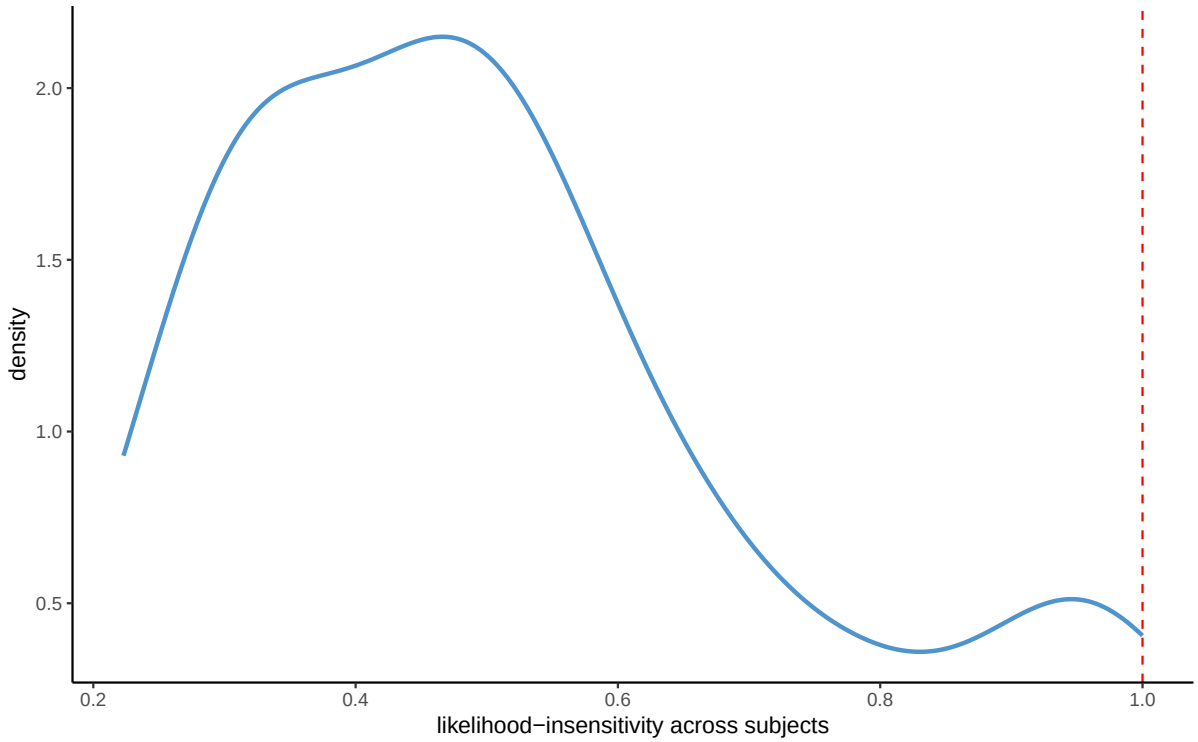


Figure 4.6: Distribution of Power utility parameter

Figure 4.6 shows the distribution of the means of individual-level likelihood-sensitivity parameters. The means are somewhat difficult to interpret, since they are on a transformed scale (and in any case, we may want to examine the means or medians of the individual-level parameters instead). The standard deviations of the individual-level estimates seem to be suitably small, indicating a fairly good identification of the parameters, although there is also quite some heterogeneity between subjects.

4.2.4 Are covariance matrices worthwhile?

The model above works perfectly fine in estimating the desired PT parameters. It does, however, treat the various model parameters as independent from each other. From a Bayesian point of view—*if* there is some correlation between the parameters—then we should be able to improve our estimations by explicitly modelling the co-variation structure between the parameters. This is due as usual to the posterior uncertainty about the true parameters. To illustrate, we can use the fixed-effects (individual-level) estimates on the data of Bruhin et al. discussed in chapter 2. I display them again here below for easy access. A remarkable feature of the data is that likelihood-sensitivity and the residual error are strongly negatively correlated (with subject nr. 9 constituting a noteworthy outlier). Such correlations indeed also

occur between other model parameters—Vieider (2024) provides a theoretical account showing what may be behind this.

```
knitr::include_graphics('figures/scatter_RR.png')
```

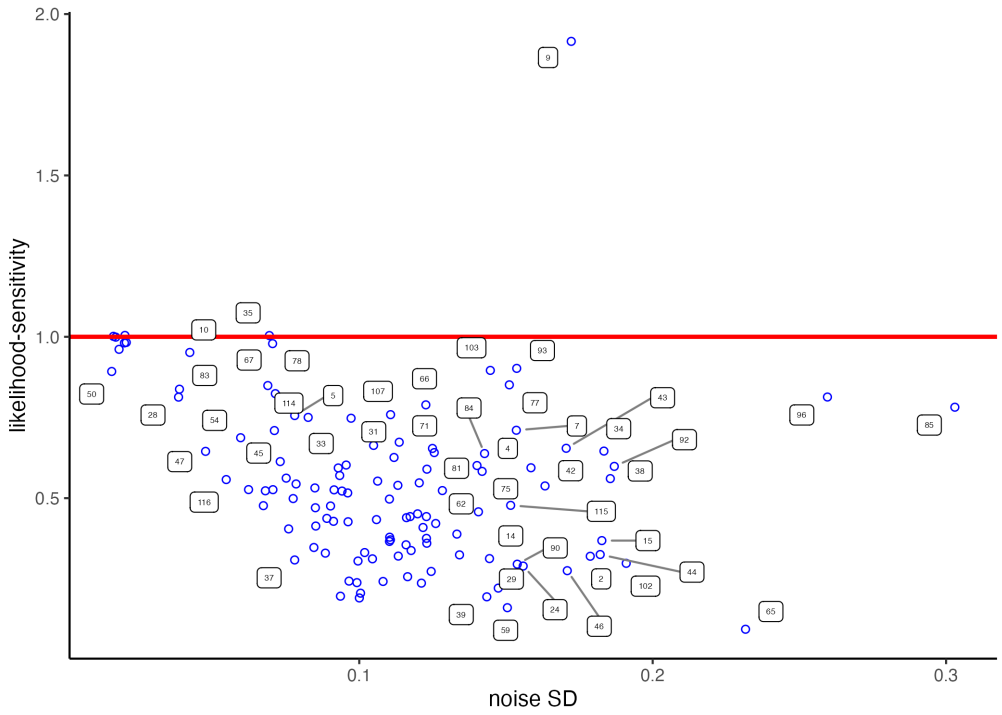


Figure 4.7: Aggregate utility function derived from probability equivalents

We will now try to leverage this additional information by estimating a model including a covariance matrix between all the different model parameters. The change to the model needed to achieve this—displayed below—is fairly trivial. We simply replace the `diag_matrix` containing the parameter variance vector, with the `diag_pre_multiply` matrix. The latter contains the parameter variance vector, and a Cholesky-decomposed covariance matrix. Of course, we give that matrix an appropriate prior, following best practices as recommended by the Stan community. For good measure we also define a transformed parameter `Rho`, which recovers the correlation parameters on the original scale (this could of course also be done post-estimation in R, or we could specify the same line of code in the `generated quantities` block, since it is not used in the estimations).

```
data {
 int<lower=1> N;
 int<lower=1> Nid;
```

```
 array[N] int id;
 array[N] real ce;
 array[N] real high;
 array[N] real low;
 array[N] real p;
}
parameters {
 vector[4] means;
 vector<lower=0>[4] tau;
 array[Nid] vector[4] Z;
 cholesky_factor_corr[4] L_omega;
}
transformed parameters{
 matrix[4,4] Rho = L_omega*transpose(L_omega);
 array[Nid] vector[4] pars;
 vector<lower=0>[Nid] rho;
 vector<lower=0>[Nid] gamma;
 vector<lower=0>[Nid] delta;
 vector<lower=0>[Nid] sigma;
for (n in 1:Nid){
 pars[n] = means + diag_pre_multiply(tau, L_omega) * Z[n];
 rho[n] = exp( pars[n,1] );
 gamma[n] = exp( pars[n,2] );
 delta[n] = exp( pars[n,3] );
 sigma[n] = exp( pars[n,4] );
 }
}
model {
 vector[N] uh;
 vector[N] ul;
 vector[N] pw;
 vector[N] pv;

 means[1] ~ normal(0,1);
 means[2] ~ normal(0,1);
 means[3] ~ normal(0,1);
 means[4] ~ normal(0,1);
 tau ~ exponential(5);
 L_omega ~ lkj_corr_cholesky(4);

for (n in 1:Nid)
 Z[n] ~ std_normal( );
```

```
for (i in 1:N){
 uh[i] = high[i]^rho[id[i]];
 ul[i] = low[i]^rho[id[i]];
 pw[i] = (delta[id[i]] * p[i]^gamma[id[i]])/
         (delta[id[i]] * p[i]^gamma[id[i]] + (1 - p[i])^gamma[id[i]]);
 pv[i] = pw[i] * uh[i] + (1-pw[i]) * ul[i];
 ce[i] ~ normal( pv[i]^(1/rho[id[i]]) , sigma[id[i]] * (high[i] - low[i]) );
 }
}
```

```
d <- read_csv("data/data_RR_ch06.csv")

dg <- d %>% filter(z1 > 0) %>%
            mutate(high = z1) %>%
            mutate(low = z2) %>%
            mutate(prob = p1) %>%
            group_by(id) %>%
            mutate(id = cur_group_id()) %>%
            ungroup()

pt <- cmdstan_model("stancode/hm_RDU.stan")

stanD <- list(N = nrow(dg),
              Nid = max(dg$id),
              id = dg$id,
              high = dg$high,
              low = dg$low,
              p = dg$prob,
              ce = dg$ce)

hm <- pt$sample(
  data = stanD,
  seed = 123,
  chains = 4,
  parallel_chains = 4,
  refresh=200,
  adapt_delta = 0.99,
  max_treedepth = 15,
  show_messages = FALSE,
  diagnostics = c("divergences", "treedepth", "ebfmi")
)
#841.9 seconds
```

```
hm$save_object(file="stanoutput/RR_ch6_RDU.Rds")
```

Figure 4.8 plots the likelihood-sensitivity parameter from the model with variances only against the one including the covariance matrix. Most of the parameters are virtually the same. This is reassuring, inasmuch as we would not expect the covariance matrix to fundamentally affected our inferences. However, especially the estimates falling into the fourth quartile of the residual variance—i.e. the observations with the largest noise—can be seen to be adjusted downwards. This happens because large error variance is associated with low likelihood-sensitivity, which is exactly the effect that most strongly emerges from the correlation matrix. Using this information, the model thus corrects observations indicating strong likelihood-sensitivity with little confidence downwards, to maximize predictive performance.

```
hc <- readRDS(file="stanoutput/RR_ch6_RDU.Rds")
hv <- readRDS(file="stanoutput/RR_ch6_RDU_var.Rds")

zz <- hc$summary(variables = c("rho","gamma","delta","sigma"), "mean" , "sd" )
zc <- zz %>% separate(variable,c("[","]")) %>%
            rename( var = `[`) %>% rename(id = `]`) %>%
            pivot_wider(names_from = var , values_from = c(mean,sd)) %>%
            rename(gamma_c = mean_gamma,rho_c = mean_rho,delta_c = mean_delta,sigma_c = mean_

zz <- hv$summary(variables = c("rho","gamma","delta","sigma"), "mean" , "sd" )
zv <- zz %>% separate(variable,c("[","]")) %>%
            rename( var = `[`) %>% rename(id = `]`) %>%
            pivot_wider(names_from = var , values_from = c(mean,sd)) %>%
            rename(gamma_v = mean_gamma,rho_v = mean_rho,delta_v = mean_delta,sigma_v = mean_

z <- left_join(zc,zv,by="id")
z <- z %>% mutate(qs = ntile(sigma_v,4))

ggplot(z,aes(x=gamma_v,y=gamma_c,color=as.factor(qs))) +
  geom_point() +
  geom_abline(intercept=0,slope=1,color="red") +
  #geom_vline(xintercept=mean(z$gamma_v)) +
  geom_label_repel(aes(label = id),
                   box.padding   = 0.35,
                   point.padding = 0.5,
                   size          = 1.5,
                   segment.color = 'grey50') +
  scale_colour_manual("Error quartile", values = c("black","darkgoldenrod","#999999","steelbl
  theme(legend.position = c(0.25, 0.8))
```

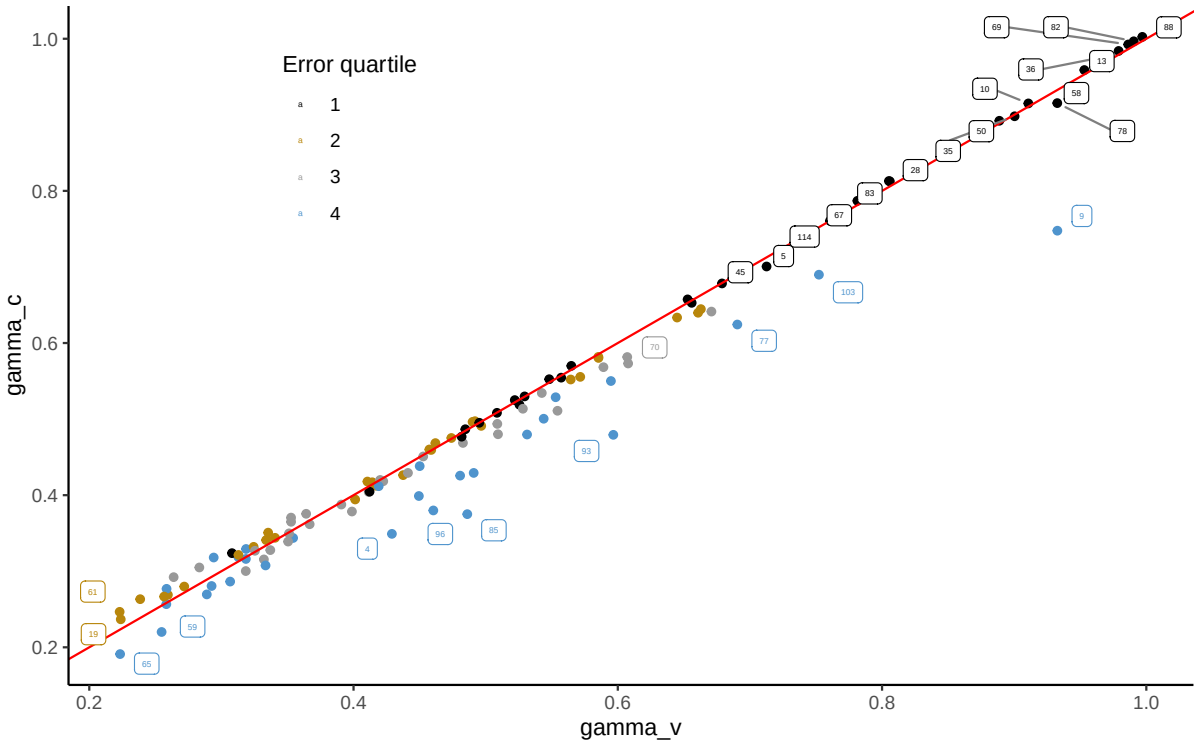


Figure 4.8: Likelihood-sensitivity with and without covariance matrix

A particularly strong adjustment can be seen for subject 9. Figure 4.9 shows three probability weighting functions fit to the datapoints for subject 9 in the Zurich 2006 data of Bruhin et al.—a function based on individual-level estimation (fixed effects); a function based on the hierarchical model with variance only; and a function based on the hierarchical model including the covariance matrix. The fixed effect model seems to provide the best fit to the nonparametric data. However, this fit seems only slightly better than the one of the hierarchical estimate with variance only, which changes our inference from the subject being oversensitive to probabilities to being slightly insensitive. Indeed, a closer look reveals that the hierarchical estimate—even though it is (slightly) inverse S-shaped instead of S-shaped—may not be statistically different from the fixed effect estimate, which carries very low confidence, as shown by the wide variation in the posterior draws, indicated by the light grey lines (made up of 200 randomly drawn rowns of the posterior).

```
# load fixed effects estimate
hf <- readRDS(file="stanoutput/ind_ZCH06_fixed.RDS")

zz <- hf$summary(variables = c("rho","gamma","delta","sigma"), "mean" , "sd")
zf <- zz %>% separate(variable,c("[","]")) %>%
            rename( var = `[`) %>% rename(id = `]`) %>%
```

```
            pivot_wider(names_from = var , values_from = c(mean,sd)) %>%
            rename(rho = mean_rho , gamma = mean_gamma , delta = mean_delta , sigma = mean_si

# load hierarchical estimate with covar
hc <- readRDS(file="stanoutput/RR_CH6_RDU.Rds")

zz <- hc$summary(variables = c("rho","gamma","delta","sigma"), "mean" , "sd")
zc <- zz %>% separate(variable,c("[","]")) %>%
            rename( var = `[`) %>% rename(id = `]`) %>%
            pivot_wider(names_from = var , values_from = c(mean,sd)) %>%
            rename(rho = mean_rho , gamma = mean_gamma , delta = mean_delta , sigma = mean_si


# load hierarchical estimate with var only
hv <- readRDS(file="stanoutput/RR_CH6_RDU_var.Rds")
zzv <- hv$summary(variables = c("rho","gamma","delta","sigma"), "mean" , "sd")
zv <- zzv %>% separate(variable,c("[","]")) %>%
            rename( var = `[`) %>% rename(id = `]`) %>%
            pivot_wider(names_from = var , values_from = c(mean,sd)) %>%
            rename(rho = mean_rho , gamma = mean_gamma , delta = mean_delta , sigma = mean_si


d <- read_csv("data/data_rr_ch06.csv")
dg9 <- d %>% filter(z1 > 0) %>%
            mutate(high = z1) %>%
            mutate(low = z2) %>%
            mutate(prob = p1) %>%
            group_by(id) %>%
            mutate(id = cur_group_id()) %>%
            ungroup() %>%
            filter(id==9) %>%
            mutate( dw = (ce - low)/(high - low) ) %>%
            group_by(prob,high,low) %>%
            mutate(mean_dw = mean(dw)) %>%
            mutate(median_dw = median(dw)) %>%
            ungroup()

###
d9 <- as_draws_df(hf)
g9 <- d9$`gamma[9]`
de9 <- d9$`delta[9]`
r9 <- as.data.frame(cbind(g9,de9))
```

```
r9 <- r9 %>% sample_n(size=200) %>%
               mutate(row = row_number())

fe9 <- function(x) (zf$delta[id=9] * x^zf$gamma[id=9] / (zf$delta[id=9] * x^zf$gamma[id=9] +
hm9 <- function(x) (zc$delta[id=9] * x^zc$gamma[id=9] / (zc$delta[id=9] * x^zc$gamma[id=9] +
hm9v <- function(x) (zv$delta[id=9] * x^zv$gamma[id=9] / (zv$delta[id=9] * x^zv$gamma[id=9]
ggplot() +
   purrr::pmap(r9, function(de9,g9, row) {
    stat_function( fun = function(x) ( de9*x^g9 )/( de9*x^g9 + (1-x)^g9) ,
                   color="grey" , linewidth = 0.25)
    }) +
  stat_function(fun = fe9,aes(color="fixed effect"),linewidth=1) + xlim(0,1) +
  stat_function(fun = hm9,aes(color="hierarchical covar."),linewidth=1) +
  stat_function(fun = hm9v,aes(color="hierarchical"),linewidth=1) +
  geom_point(aes(x=dg9$prob,y=dg9$mean_dw),size=3, color="chartreuse4",shape=10) +
  geom_abline(intercept=0, slope=1,linetype="dashed",color="gray") +
  scale_colour_manual("Legend", values = c("steelblue3","darkgoldenrod", "chocolate4")) +
    xlab("probability") + ylab("decision weights") + theme(legend.position = c(0.8, 0.2))
```

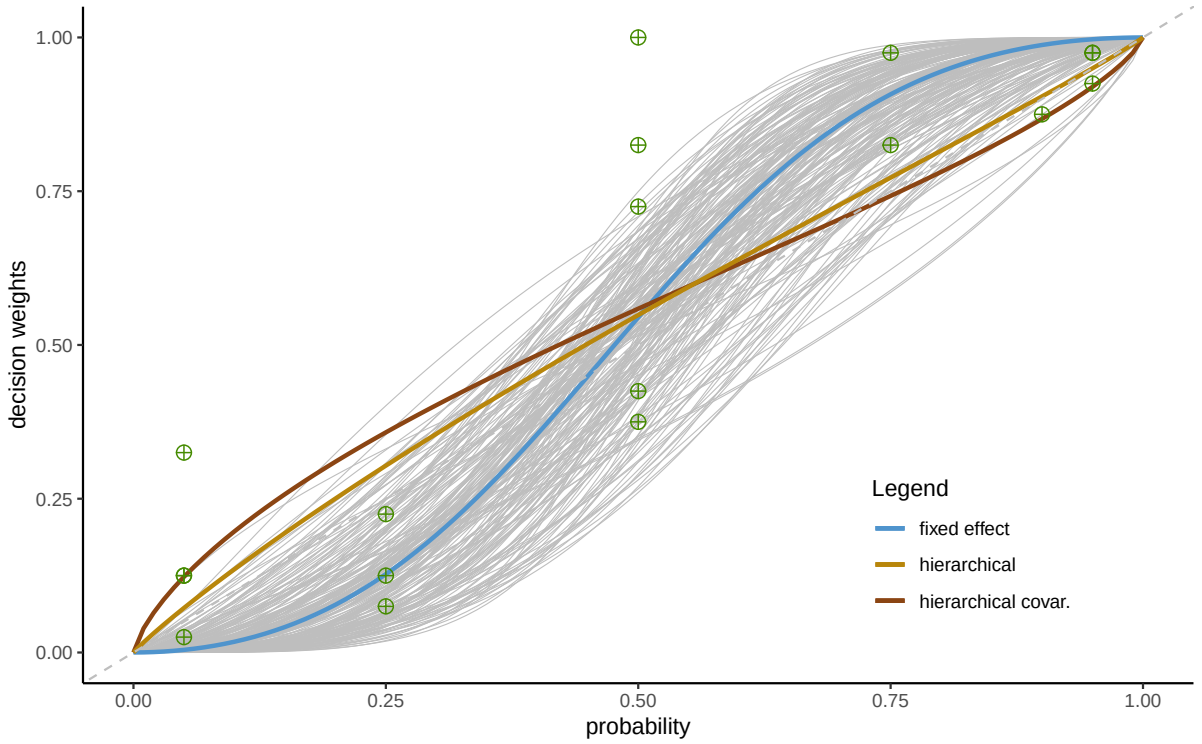


Figure 4.9: Distribution of Power utility parameter

Most importantly, however, the hierarchical function reflects the fact that this particular observation seems unlikely, *given the distribution of parameters in the population*. Since the estimate is furthermore very noisy, it is shrunk towards the population mean. This follows standard Bayesian procedures, whereby noisy outliers are discounted and drawn towards the prior, which in this case is endogenously estimated from the aggregate data (see BOX for a short discussion). Such shrinkage serves to optimize *predictive* performance, and is based on the probability distributions gathered from the environment.

> **An example of hierarchical aggregation**
>
> Take a parameter γ that is distributed $\gamma_i \sim \mathcal{N}(\hat{\gamma}_i, \sigma_i^2)$, where $\hat{\gamma}_i$ indicates the true parameter mean to be estimated for subject i, whereas γ_i is the maximum likelihood estimator (individual fixed effect). The two may not coincide, for instance because of large sampling variation in small samples. Assume further that the prior distribution of this estimator in the population is also normal, so that $\hat{\gamma} \sim \mathcal{N}(\mu, \tau^2)$. The Bayesian posterior will then itself follow a normal distribution, with mean $\hat{\gamma}_i = \frac{\sigma^2}{\sigma^2+\tau^2}\gamma_i + \frac{\tau^2}{\sigma^2+\tau^2}\mu$ and variance $\frac{\sigma_i^2\tau^2}{\sigma_i^2+\tau^2} = \frac{1}{\sigma_i^2} + \frac{1}{\tau^2}$. That is, the posterior mean will be a convex combination of the fixed effect estimator and the prior mean. The "shrinkage weight" $\frac{\sigma^2}{\sigma^2+\tau^2} = 1 - \frac{\tau^2}{\sigma^2+\tau^2}$ will be a function of the variance of the parameter itself, and of the prior variance. The larger the variance of the individual parameter, the lower the weight attributed to it, and the larger the weight attributed to the prior mean. The greater the population variance σ^2, the less weight is attributed to the prior mean, capturing the intuition that outliers from more homogeneous populations are discounted more. Notice that these equations are directly applicable to our calculations for subject 9. We have $\gamma_9 = 2.08$, $\sigma_9^2 = 0.36$, $\mu = 0.45$, and $\tau^2 = 0.16$. We thus obtain $\hat{\gamma}_9 = 0.9515385$. The value estimated from the hierarchical model is very similar at 0.934. Note, however, that including a covariance matrix further lowers this value to 0.73. This is due to the more complex structure implemented in the estimated model, which takes into account the strongly negative correlation between likelihood-sensitivity and noise.

This is particularly apparent from the function estimated from the hierarchical model including a covariance matrix. The latter indeed seems to fall even farther from the fixed effects estimator, and is now arguably statistically distinct from it at conventional levels taken to indicate significance. Nevertheless, it appears the most likely function in the context of the entirety of the data, incorporating not only the likelihood of any given parameter value based on the parameter distribution in the population, but also the correlation structure between the different parameters in the model. Note that these large differences emerge because the outlier is *noisy*—perfectly identified outliers will not be shrunk.

Covariance matrices can thus be a great way of improving the information content in hierarchical models. Nevertheless, some caution is called for. In particular, there can be circumstances where a covariance matrix does more harm than good. This will for instance occur in data with large, low-noise outliers. Such outliers may then distort the parametric correlations in

the matrix, in extreme cases even in the opposite direction of the nonparametric correlations observed between estimated parameters. It is thus always a good idea to check the correlations in both ways (i.e., to compare the parametric estimation in the matrix to rank-correlations of the posterior means of the individual-level parameters). While some quantitative differences are expected, wildly differing correlation coefficients that may even go in opposite directions are a sign of trouble afoot.

4.2.5 Regression analysis in hierarchical models

Often we will want to conduct regression analysis using the parameters of the decision model as dependent variables (or indeed as independent variables). It may be tempting to simply estimate the model, obtain the posterior means of the individual-level parameters, and then to conduct the regression analysis using these parameter means using standard regression tools. Such a procedure, however, has the distinct drawback of throwing away the information on the confidence we have in the different estimates. Given large variation in the reliability of estimates, that may severely bias our ultimate inferences. Here, I discuss how to conduct regression analysis in such a way as to consider the full posterior uncertainty about the parameter estimates. (In some cases, this method may not be applicable, e.g. because the original data are not available. In this case, one could approximate this procedure using a measurement error model—see the end of the following section for details).

Let us look at this based on the PT model with covariance matrix from above. Assume you want to regress the likelihood-sensitivity parameter, γ, on some demographics, say sex and age. Just like in linear regression models discussed in chapter 2, it will be convinient to add these demographics to a design matrix, `x`, together with a column of 1s to model the intercept. All that is then required is to add the following line below the loop estimating the model parameters (of course, you will also need to import the design matrix `x` in the `data` block and declare its dimensions; and to declare the vector of regression coefficients `beta` and the scale parameter of the regression `tau` in the `parameters` block): `gamma ~ normal( x * beta , tau);`.

We are now conducting a regression of the parameter, which is automatically treated as an uncertain quantity. That is, writing the regression equation like this will take the full posterior uncertainty about `gamma` into account. The code here below executes a regression of likelihood-sensitivity on sex (specifically, a female dummy) and on the semester of study (for no other reason than that there are not many other variables available in the data set).

```
d <- read_csv("data/data_RR_ch06.csv")

dg <- d %>% filter(z1 > 0) %>%
            mutate(high = z1) %>%
            mutate(low = z2) %>%
            mutate(prob = p1) %>%
```

```
            group_by(id) %>%
            mutate(id = cur_group_id()) %>%
            ungroup()

pt <- cmdstan_model("stancode/hm_RDU_reg.stan")

di <- dg %>% group_by(id) %>%
             filter(row_number()==1) %>%
             ungroup()

x <- model.matrix( ~ female + semester, di)

stanD <- list(N = nrow(dg),
              Nid = max(dg$id),
              id = dg$id,
              high = dg$high,
              low = dg$low,
              p = dg$prob,
              ce = dg$ce,
              K = ncol(x),
              x = x)

hm <- pt$sample(
  data = stanD,
  seed = 123,
  chains = 4,
  parallel_chains = 4,
  refresh=200,
  adapt_delta = 0.99,
  max_treedepth = 15,
  show_messages = TRUE,
  init = 0,
  diagnostics = c("divergences", "treedepth", "ebfmi")
)


hm$save_object(file="stanoutput/RR_ch6_RDU_reg.Rds")
```

```
hm <- readRDS("stanoutput/RR_ch6_RDU_reg.Rds")

hm$print(c("beta"), digits = 3,
          "mean", SE = "sd" , "rhat",
```

```
          ~quantile(., probs = c(0.025, 0.975)),max_rows=30)
```

```
 variable   mean    SE  rhat   2.5%  97.5%
  beta[1]  0.529 0.033 1.000  0.464  0.595
  beta[2] -0.106 0.034 1.000 -0.170 -0.039
  beta[3]  0.003 0.007 1.000 -0.010  0.016
```

The results indicate that females display less likelihood-sensitivity—a common finding in the literature, see e.g. L'Haridon and Vieider (2019). In most practical applications, however, you will want to regress *all* the parameters at once on the independent variables of interest. A very simple way of coding this would be to add a line of code for each separate parameter into the model (and of course to declare as many scale parameters and regression coefficient vectors). Alternatively, one could collect all the parameters into a matrix and run all the regressions at once. The principle is however the same as above, so that generalizing this model is left as an exercise. Of course, this is not the only way of incorporating regressions into the model. It is, however, particularly flexible, allowing for recombinations of (full posteriors of) parameters, and to include parameter e.g. as right-hand side controls, instead of as dependent variables as shown above.

4.3 Meta-analysis as hierarchical analysis

Meta-analysis is nothing but a hierarchical analysis where the lowest level is encoded from existing studies, i.e. we directly obtain the mean and standard deviation (standard error in frequentist terms) of the variable of interest. Beyond standard meta-analytic applications, the model is thus suitable in any situation in which one may wish to obtain aggregate estimates across several instances in which an effect is measured with error. This may for instance apply to the case where (almost) identical experiments are executed in a number of different locations. In such cases, the method will deliver several desirable outcomes at once—obtaining an aggregate estimate; correcting individual estimates for measurement error; disciplining multiple tests by explicitly modelling their inter-dependence; and correcting local inferences from the single experiments by making use of the global information provided by all the data. The usual caveat applies—this is not a textbook in meta-analysis, and such a textbook should be consulted for many important details. Here, I will rather focus on the technical details required to implement meta-analysis in Stan.

4.3.1 The power of meta-analysis

Let us start from an example based on the clinical trial data from table 5.4 in BDA3 to illustrate the approach. Technically, the data summarize the effect of beta-blockers on reducing mortality

after myocardial infarction. The data have been collected in 22 hospitals, and we have data on the log-odds and the associated standard errors, which can be assumed to be approximately normally distributed. Beyond these details, however, the data could be anything, and you might think of them as risk taking tendencies in 22 experiments, or indeed as the effect of a nudging intervention on some sort of behaviour.

```
bb <- data.frame(log_odds  = c("0.028","-0.741","-0.541","-0.246","0.069","-0.584",
                               "-0.512","-0.079","-0.424","0.335","-0.213","-0.039",
                               "-0.593","0.282","-0.321","-0.135","0.141","0.322","0.444",
                               "-0.218","-0.591","-0.608"),
                         sd = c("0.850","0.483","0.565","0.138","0.281","0.676","0.139","0.2
                                "0.274","0.117","0.195","0.229","0.425","0.205","0.298",
                                "0.261","0.364","0.553","0.717","0.260","0.257","0.272"))
bb <- as.data.frame(apply(bb, 2, as.numeric))
bb <- bb %>% mutate( lower = log_odds - 1.96*sd ) %>%
             mutate( upper = log_odds + 1.96*sd ) %>%
             mutate(study = row_number())
```

We can display these data in a graph as follows:

```
ggplot(data=bb, aes(y=study, x=log_odds, xmin=lower, xmax=upper)) +
  geom_point(size=2,color="cornflowerblue") +
  geom_errorbarh(height=.1,color="cornflowerblue") +
  geom_vline(xintercept=0,color="red",linetype="dashed") +
  scale_y_continuous(name = "", breaks=1:nrow(bb), labels=bb$study)
```

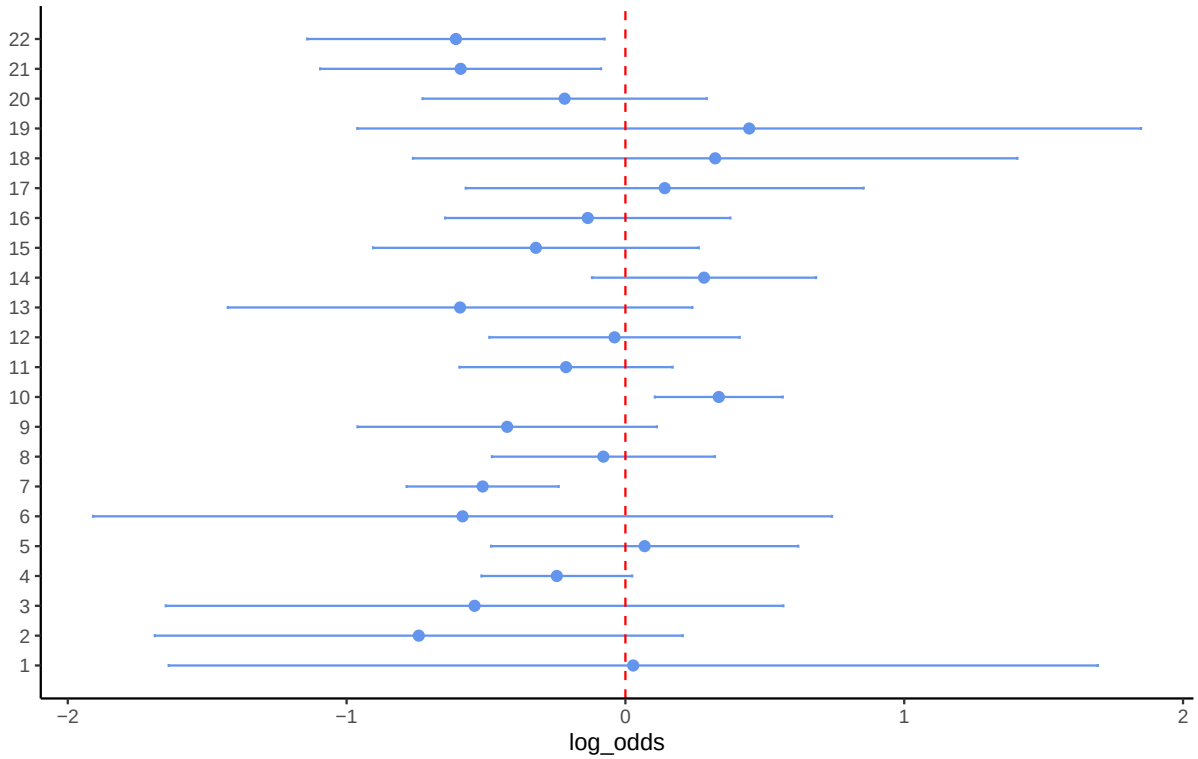


Figure 4.10: Effects of Beta-blockers across 22 experiments

The graph shows that most of the effects were not significant, i.e. Beta blockers do not seem to reduce the likelihood of myocardial infarction based on most of these studies. Three studies had a significantly negative effect, suggesting that the intervention worked. However, one study recorded a significantly *positive* effect, suggesting that the intervention made the disease worse. Given evidence like this, most traditional literature reviews would conclude that there is no effect. Even tallying up significant effects, as some scholars have done, would not yield any clear insights here. At the very least, a systematic Bayesian analysis should allow us to describe the evidence somewhat better.

First, however, we need to check whether the conditions are fulfilled under which we can apply Bayesian hierarchical analysis to the problem. Given that we know nothing about the 22 hospitals, we should consider them as *exchangeable*—an important condition for a hierarchical model (Note: if we were aware of some characteristics that might matter, say the size of the hospital or the prevalence of the disease in the city where the hospital is located, we could control for this in a regression, so that the observations would again become exchangeable *conditional on size and disease prevalence*). We should also make sure that the experiments were comparable, though in the absence of evidence to the contrary we may well choose to assume so. We are thus good to go.

The basic assumption underlying a meta-analysis (or for that matter, any measurement error

model) is that we might not be observing the true effect size in single experiments. The reasons for that can vary. Perhaps the most standard reason is that, according to sampling theory, we would approximate the true effect size only as the sample size tends to infinity. Any small subsample may thus be affected by sampling error. Sampling error can be increased by all sorts of other errors, including (unobserved) randomization failures, measurement or recording errors, unobserved differences between locations and implementation protocols and the like. We will thus model the observed log-odds, x, as being distributed normally, with as their mean the *true* (but unobserved) log odds, $\hat{x}$, and with the standard deviation given by the observed standard error:

$$x_i \sim \mathcal{N}(\hat{x}_i\,,\, se_i),$$

where I have subscriped the quantities by i to emphasize that they belong to the ith experiment. We will further model the true effect sizes as being drawn from a common mean μ, and following some distribution around that mean that captures the "true variation" in effect sizes that is not due to error. This is indeed the very reason to model the effect hierarchically—after all, if we were sure that all experiments were truly identical, we could simply pool the data. Each study will enter this equation with a weight that is proportional to the inverse of its variance (i.e. it will receive a weight $\frac{\sigma^2}{\sigma^2+se_i^2}$ in a normal-normal model as above). The Stan code below implements such a model using vector notation.

```
data{
    int<lower=1> N;
    vector[N] lo;
    vector[N] se;
}
parameters{
    vector<lower=0>[N] lo_hat;
    real<lower=0> mu;
    real<lower=0> sigma;
}
model{
//priors:
    sigma ~ cauchy( 0 , 5 );
    mu ~ normal( 0 , 5 );

//model:
    lo ~ normal( x , se );
    x ~ normal( mu , sigma );
}
```

```
ma <- cmdstan_model("stancode/meta_bb.stan")

stanD <- list(N = nrow(bb), #creates data to send to Stan
```

```
              lo = bb$log_odds,
              se = bb$sd)


hm <- ma$sample(
  data = stanD,
  seed = 123,
  chains = 4,
  parallel_chains = 4,
  refresh=200,
  adapt_delta = 0.99,
  max_treedepth = 15,
  show_messages = FALSE,
  diagnostics = c("divergences", "treedepth", "ebfmi")
)

hm$save_object(file="stanoutput/meta_bb.Rds")
```

```
 variable   mean    SE  rhat   2.5%  97.5%
    mu    -0.181 0.086 1.001 -0.355 -0.013
    sigma  0.261 0.079 1.002  0.131  0.444
    m[1]  -0.163 0.269 1.000 -0.696  0.370
    m[2]  -0.307 0.249 1.001 -0.832  0.168
    m[3]  -0.245 0.245 1.003 -0.744  0.224
    m[4]  -0.230 0.121 1.000 -0.466  0.010
    m[5]  -0.068 0.194 1.000 -0.442  0.320
    m[6]  -0.236 0.261 1.001 -0.777  0.261
    m[7]  -0.429 0.126 1.001 -0.684 -0.180
    m[8]  -0.121 0.163 1.001 -0.440  0.205
    m[9]  -0.287 0.198 1.000 -0.681  0.096
    m[10]  0.234 0.115 1.002  0.011  0.466
    m[11] -0.201 0.156 1.002 -0.517  0.100
    m[12] -0.104 0.174 1.002 -0.441  0.247
    m[13] -0.291 0.241 1.000 -0.784  0.163
    m[14]  0.092 0.175 1.001 -0.239  0.458
    m[15] -0.237 0.199 1.000 -0.652  0.146
    m[16] -0.161 0.189 1.002 -0.533  0.209
    m[17] -0.074 0.219 1.001 -0.501  0.364
    m[18] -0.092 0.256 1.002 -0.587  0.445
    m[19] -0.105 0.263 1.002 -0.612  0.441
    m[20] -0.197 0.186 1.000 -0.581  0.168
    m[21] -0.379 0.196 1.000 -0.782 -0.032
```

```
    m[22] -0.378 0.199 1.000 -0.798  0.002
```

The posterior indicates a mean effect size of −0.18, which corresponds to -0.8352702 in odds, or -45.51 in percentage change. This treatment effect is clearly significant, thus markedly changing the conclusion we would likely have reached based on a less systematic or less quantitative approach. At the same time, the substantial variance indicates that the experiments were not identical, thus justifying the hierarchical modelling.

```
est <- readRDS(file="stanoutput/meta_bb.Rds")

zz <- est$summary(variables = c("m"), "mean", pse = "sd")
z <- zz %>% separate(variable,c("[","]")) %>%
            rename(study = "]" , post = mean) %>%
            mutate(study = as.integer(study)) %>%
            mutate(pl = post - 1.96*pse) %>%
            mutate(pu = post + 1.96*pse)

dls <- left_join(bb,z,by="study")


ggplot(dls) +
  geom_point(aes(x=log_odds,y=study + 0.2),size=2.5,color="cornflowerblue",shape=19) +
  geom_point(aes(x=post,y=study - 0.2),size=2.5,color="navy",shape=17) +
  geom_errorbarh(aes(xmin = lower, xmax = upper , y=study + 0.2), height = 0.2,color="cornflo
  geom_errorbarh(aes(xmin = pl, xmax = pu , y=study - 0.2), height = 0.2,color="navy",size=1)
  geom_vline(xintercept=0,color="red") +
  geom_vline(xintercept=-0.184,lty=2,size=1.2,color="navy") +
  annotate('rect', xmin=-0.355, xmax=-0.013, ymin=0, ymax=23, alpha=.2, fill='navy') +
  scale_x_continuous(breaks=seq(-5,5,by=0.25), expand = c(0.01, 0.01)) +
  scale_y_continuous(breaks=seq(1,22,by=1), expand = c(0.02, 0)) +
  labs(x="log-odds (95% CI)" , y ="") +
  ggtitle("raw data versus posterior inferences") +
  theme(plot.title = element_text(hjust = 0.5, vjust = 0,size=10,face = "bold"),
        legend.position = c(0.1, 0.8))
```

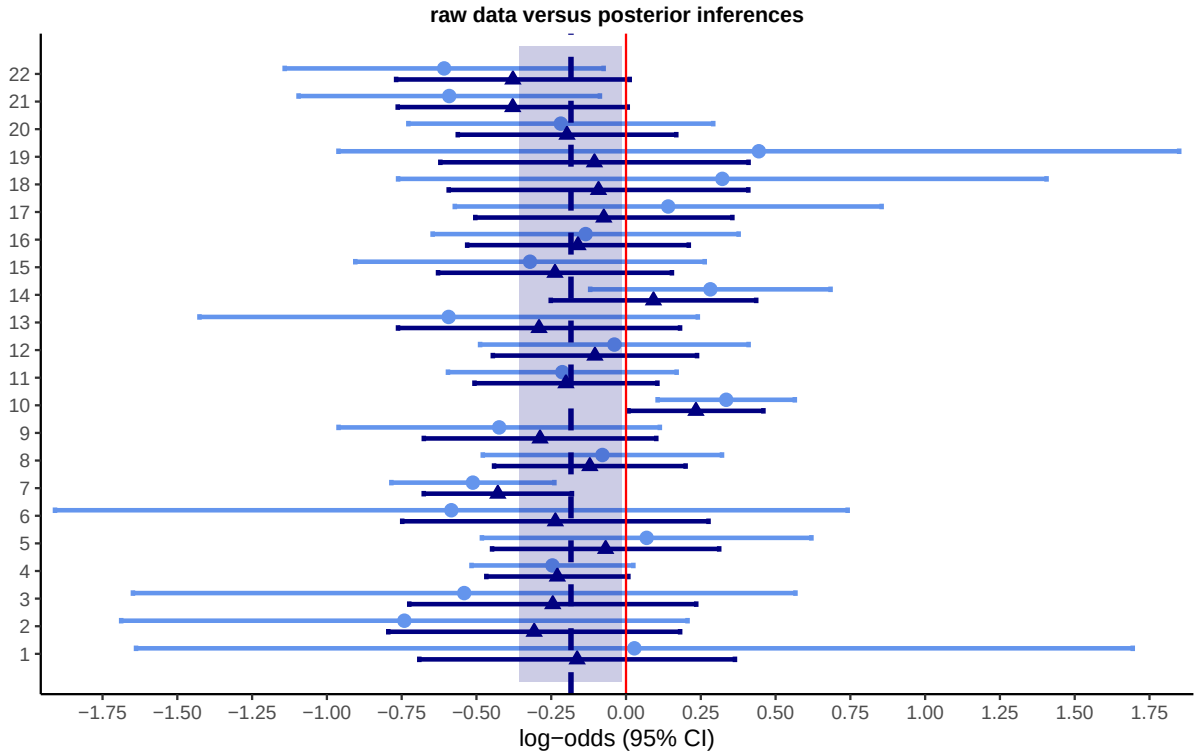


Figure 4.11: Forest plot data vs meta-analytic posterior

Figure 4.11 plots the data from the 22 experiments together with the posterior inferences on the true parameters (indicated by the dark blue triangles). The shaded region indicates the posterior mean and surrounding credible interval. A few other patterns stand out. The posterior means—indicated by the dark blue triangles—tend to fall closer to the meta-analytic mean than the original data means, indicated by the light blue circles. This is of course the now-familiar shrinkage to the mean of the endogenous prior at work. The posterior credible intervals are quite a bit narrower than the intervals in the data, which is also expected. Of course, meta-analysis is not always this simple. Below, we will see a case with a more complex structure.

4.3.2 Imputation of missing data

A frequent problem in meta-analysis is that some of the statistical information needed to conduct the analysis cannot be recovered from the original papers. This mostly (but not necessarily only) concerns statistical information that would allow one to infer the standard errors surrounding the estimates of the effects of interest. Dropping such estimates risks biasing the analysis, or at the very least leads to the loss of valuable information. The good news is that one does not have to drop such observations. If one can reasonably assume that the

information is missing at random—i.e. that it is not studies with particular results that would omit this information—then we can impute the missing information within the statistical code used to conduct the analysis.

I will illustrate how one can approach that problem using a set of estimates of prospect theory parameters. In the data I use below, I will focus on some complete observations of parameters for gains, including a utility curvature parameter, a likelihood-sensitivity parameter, and a pessimism parameter. This data does of course NOT encompass the universe of all estimates, so that one ought to be careful to interpret too much into the results of the analysis. It will, however, do fine as an example to develop some more advanced code for meta-analysis.

Let us assume we want to analyze the likelihood-sensitivity parameter, called `sens` in the data. Let us set aside for now the fact that this parameter has been obtained from different functional forms and measurements, which questions the exchangeability principle. We could now simply use the code from above to analyze this model, perhaps substituting `sens` for `lo` in the data, and designating its mean by `sens_hat`. The problem is that, out of the 102 observations, 16 are missing a standard error, which was not reported in the original paper from which the estimate has been taken. Throwing out these observations is clearly undesirable. Instead, we can impute the standard errors, which will allow us to keep the observations in the meta-analysis. This can be done using the following model:

```
data{
    int<lower=1> N;
    vector[N] sens;

//  additional elements for imputation
    int<lower=0> N_obs; // number of observed SEs
    int<lower=0> N_mis; // number of missing SEs

//indices for imputation, and observed standard errors:
    array[N_obs] int<lower = 1,upper=N> ii_obs; // index of observed values
    array[N_mis] int<lower = 1,upper=N> ii_mis; // index of missing values
    array[N_obs] real<lower=0,upper=N> se_obs; // observed SEs entered as data

//design matrix used for imputation
    int<lower=0> K; //dimension of the design matrix
    matrix[N,K] x; // design matrix to impute SEs
}
parameters{
    vector<lower=0>[N] sens_hat;
    real<lower=0> mu;
    real<lower=0> sigma;
//
```

```
    array[N_mis] real<lower=0> se_mis;
    real<lower=0> sd_se;
    vector[K] beta;
}
// compose array of SEs
transformed parameters{
    array[N] real<lower=0> se;
    se[ii_obs] = se_obs;
    se[ii_mis] = se_mis;
}
model{
//priors:
    sigma ~ cauchy( 0 , 5 );
    mu ~ normal( 1 , 5 );
    sd_se ~ normal(0 , 5);
    beta ~ normal( 0 , 5);

//imputation model
    se ~ normal( x * beta, sd_se);

//hierarchical model:
    sens ~ normal( sens_hat , se );
    sens_hat ~ normal( mu , sigma );
}
```

The model is best examined from the bottom up. The last two lines implement a standard meta-analysis model like seen above. The imputation model is new: it comprises a vector of standard errors, `se`, that is modelled as being normally distributed, with a standard deviation `sd_se` to be estimated, and a regression `x * beta`. The key to understanding the model is to examine the vector `se`. This vector is composed in the newly added `transformed parameters` block. This reveals how this vector of size `N` is composed of the actual data points in `se_obs`, which are imported as data, and a vector of variables `se_mis`, created in the `parameters` block. This way of proceeding thus reveals that the vector `se` is made up of some data points and some variables. The indices `ii_obs`and `ii_mis` thereby ensure the data points and variables are entered into the correct positions, i.e. corresponding to the existing and missing data on SEs, respectively. The new line in the `model` block thus achieves 2 things at once: 1) it uses the existing observations of the SEs to obtain the regression coefficients in `beta`; and 2) it simultaneously uses these coefficients and the data in the design matrix `x` to *simulate* data points for the missing observations. These are our imputed SEs. NOTE: this will of course only work as long as the data in `x` are available for ALL observations.

The model is thus ready to run. Care, however, needs to be taken in creating the indices and data vectors to be sent to the programme. We start by examining which variables are

predictive of the SEs. It turns out that the encoded sensitivity parameter itself does a pretty good job of that by itself, capturing 27% of the variance. We then organize the values and create a unique index. Now we are ready to run the programme:

```
d <- read_csv("data/data_example_meta.csv",show_col_types = FALSE)
d %>% summarize(missing = sum(is.na(sens_se)))

# explore predictors of SEs
summary( lm(sens_se ~ sens,data=d) )

# create indices for missing values
d <- d %>%   mutate(obs_na = ifelse(is.na(sens_se) , 1 , 0)) %>%
             arrange(desc(obs_na)) %>%
             select(sens,sens_se,obs_na) %>%
             mutate(index = row_number())

# create data subsets for observed a missing SEs
do <- d %>% filter(!is.na(sens_se))
dm <- d %>% filter(is.na(sens_se))

# create design matrix for all data
x <- model.matrix(data = d , ~ sens)

# data to send to Stan
stanD <- list(N = nrow(d),
              K = ncol(x),
              N_obs = nrow(do),
              N_mis = nrow(dm),
              ii_obs = do$index,
              ii_mis = dm$index,
              sens = d$sens,
              se_obs = do$sens_se,
              x = x)

# execute model
ma <- cmdstan_model("stancode/meta_imp.stan")

# launch estimation
est <- ma$sample(
  data = stanD,
  seed = 123,
  chains = 4,
  parallel_chains = 4,
```

```
  refresh=200,
  init=0,
  adapt_delta = 0.999,
  max_treedepth = 15,
  iter_warmup = 1000,
  iter_sampling = 1000,
  show_messages = TRUE,
  diagnostics = c("divergences", "treedepth", "ebfmi")
)

est$print(c("mu","sigma", "beta","se_mis"), digits = 3,
          "mean", SE = "sd", "rhat", ~quantile(., probs = c(0.025, 0.975)) ,
           max_rows=20)
```

Even though we have forced convergence statistics, using a rather extreme `adapt_delta = 0.999` and thereby slowing down convergence, we get about 2% of divergent iterations. We should thus not trust the results. Instead, we can try and improve the performance of the model by decentralizing some or all of its parameters. Let us start by decentralizing the estimated latent parameters `sens_hat`. To do that, we create a new vector of parameters `sens_hat_z` and add it in the parameter block. We then use the following `transformed parameters` and `model` blocks:

```
transformed parameters {
    array[N] real<lower=0> se;
    vector[N] sens_hat;

    // Compose array of SEs
    se[ii_obs] = se_obs;
    se[ii_mis] = se_mis;

  // Recenter sens_hat using decentralized parameters
    sens_hat = mu + sigma * sens_hat_z;
}
model {
    // Priors
    sigma ~ cauchy(0, 5);
    mu ~ normal(1, 5);
    sd_se ~ normal(0, 5);
    beta ~ normal(0, 5);

    //prior for sens_hat_raw
    sens_hat_z ~ std_normal( );
```

```
    // Imputation model
    se ~ normal(x * beta, sd_se);

    // measurement error model
    sens ~ normal(sens_hat, se);
}
```

Let us look at the bottom line of the model first, where we now include the measurement error part of the hierarchical model only. The part pertaining to the distribution of the latent parameters is shifted to the end of the `transformed parameters` block. Here, we now define the latent variables `sens_hat` as being made up of the aggregate mean, `mu`, plus the rescaled parameters `sens_hat_z`, multiplied by the standard deviation of the aggregate distribution. This means that `sens_hat_z` takes the form of z-scores of the `sens_hat` parameters, which is reflected in the standard normal distribution we give to the prior, which holds by definition. The following code estimates the model:

```
# execute model
ma <- cmdstan_model("stancode/meta_imp_dec.stan")

# launch estimation
est <- ma$sample(
  data = stanD,
  seed = 123,
  chains = 4,
  parallel_chains = 4,
  refresh=200,
  init=0,
  adapt_delta = 0.999,
  max_treedepth = 15,
  iter_warmup = 1000,
  iter_sampling = 1000,
  show_messages = TRUE,
  diagnostics = c("divergences", "treedepth", "ebfmi")
)


est$save_object(file="stanoutput/meta_imp_dec.Rds")
```

The model now converges almost twice as quickly, and the divergences almost completely disappear. Arguably, a model decentralizing the imputation model will work even better. However, this is good enough for now, and I thus leave that even more efficient model as

an exercise. The figure below shows the imputed standard errors next to the observed ones, and plotted against the observed likelihood-sensitivity parameter. The imputed means of the standard errors follow the trend in the observed variables, as we should expect based on the imputation model. Importantly, however, while the observed SEs are treated as sure quantities, the imputed SEs have some uncertainty surrounding them, indicated by the plotted 95% credible intervals. The model will take this additional uncertainty into account, so that observed and imputed data enter with different weights.

```
# import data
d <- read_csv("data/data_example_meta.csv",show_col_types = FALSE)

# create indices for missing values
d <- d %>%   mutate(obs_na = ifelse(is.na(sens_se) , 1 , 0)) %>%
             arrange(desc(obs_na)) %>%
             select(sens,sens_se,obs_na) %>%
             mutate(index = row_number())

# create data subsets for observed a missing SEs
do <- d %>% filter(!is.na(sens_se))
dm <- d %>% filter(is.na(sens_se))

# import estimates
est <- readRDS(file="stanoutput/meta_imp_dec.Rds")

zz <- est$summary(variables = c("se_mis"), "mean", "sd")
z <- zz %>% separate(variable,c("[","]")) %>%
            rename(index = "]" , se = mean) %>%
            mutate(index = as.integer(index))

ggplot() +
  geom_point(aes(x=do$sens,y=do$sens_se), color="blue") +
  geom_point(aes(x=dm$sens,y=z$se), color="orange") +
  geom_errorbar(aes(x = dm$sens, ymin = z$se - z$sd, ymax = z$se + z$sd), color = "orange", w
  labs(x="likelihood sensitivity (observed)", y= "standard error, observed (blue) and imputed
```

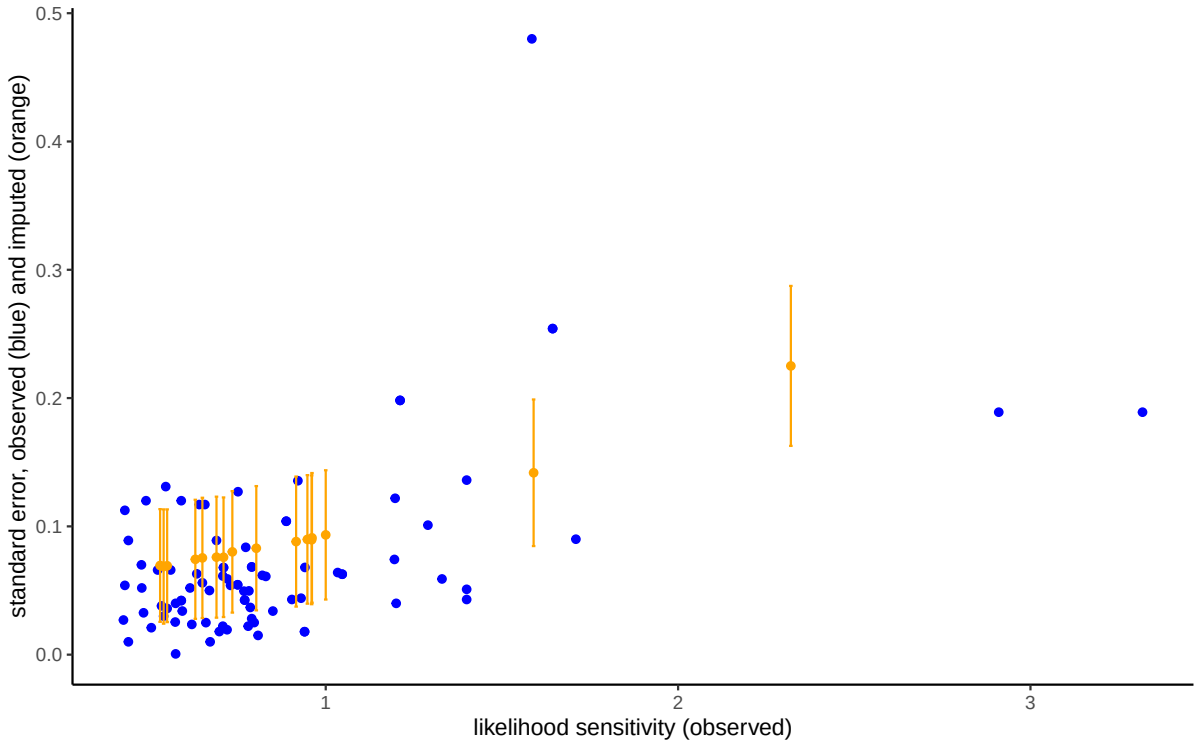


Figure 4.12: Observed versus imputed standard errors

4.3.3 Meta-regression

One of the goals of meta-analysis is usually to explore the correlates of the differences in effects between studies, at least in cases where those correlates are observed or can be encoded, and where there is reason to believe that such differences exist. This will not only allow one to uncover correlates, but also to discuss the degree to which they explain the between-study variance. Even if this is not the goal of your analysis, a meta-analysis such as in the examples above is of questionable solidity because it violates the exchangeability principle: at the very least, we ought to control for the functional form of the weighting function, and for the type of measurement (aggregate vs individual mean vs individual median).

Whether the model is written in a centralized or decentralized way, such a meta-regression is easy to integrate. They ke is simply to replace the mean, `mu`, by a regression, `z * gamma`, where the design matrix `z` used for the meta-regression will generally be different from the matrix `x`, used for error imputation. In this specification, `z` will contain a first column of 1s that implement the constant. In fact, if `z` only contains that column, than `gamma` will be a scalar equal to `mu` in the previous model.

```
data {
    int<lower=1> N;
    vector[N] sens;

    // Additional elements for imputation
    int<lower=0> N_obs; // number of observed SEs
    int<lower=0> N_mis; // number of missing SEs

    // Indices for imputation, and observed standard errors:
    array[N_obs] int<lower=1,upper=N> ii_obs; // index of observed values
    array[N_mis] int<lower=1,upper=N> ii_mis; // index of missing values
    array[N_obs] real<lower=0> se_obs; // observed SEs entered as data

    // Design matrix used for imputation
    int<lower=0> K; // dimension of the design matrix
    matrix[N,K] x; // design matrix to impute SEs

    // Design matrix used for imputation
    int<lower=0> M; // dimension of the design matrix
    matrix[N,M] z; // design matrix to impute SEs
}
parameters {
    vector[N] sens_hat_raw; // Decentralized version of sens_hat
    real<lower=0> sigma;

    // Additional parameters for imputation
    array[N_mis] real<lower=0> se_mis;
    real<lower=0> sd_se;
    vector[K] beta;
    vector[M] gamma;
}
transformed parameters {
    array[N] real<lower=0> se;
    vector[N] sens_hat; // Recentered sens_hat

    // Compose array of SEs
    se[ii_obs] = se_obs;
    se[ii_mis] = se_mis;

    // Recenter sens_hat using decentralized parameters
    sens_hat = z * gamma + sigma * sens_hat_raw;
}
```

```
model {
    // Priors
    sigma ~ cauchy(0, 5);
    sd_se ~ normal(0, 5);
    beta ~ normal(0, 5);
    gamma ~ normal(0, 5);
    sens_hat_raw ~ std_normal( ); // Standard normal prior for decentralized parameter

    // Imputation model
    se ~ normal(x * beta, sd_se);

    // Hierarchical model
    sens ~ normal(sens_hat, se);
}
```

The following code executes the meta-regression:

```
d <- read_csv("data/data_example_meta.csv",show_col_types = FALSE)
d %>% summarize(missing = sum(is.na(sens_se)))

# explore predictors of SEs
summary( lm(sens_se ~ sens,data=d) )

# create indices for missing values
d <- d %>%   mutate(obs_na = ifelse(is.na(sens_se) , 1 , 0)) %>%
             arrange(desc(obs_na)) %>%
             mutate(index = row_number()) %>%
             mutate(prelec2 = ifelse(pwf_form=="LLO",0,1))

# create data subsets for observed a missing SEs
do <- d %>% filter(!is.na(sens_se))
dm <- d %>% filter(is.na(sens_se))

# create design matrix for imputation
x <- model.matrix(data = d , ~ sens)

# design matrix for meta-regression
z <- model.matrix(data = d , ~ prelec2)

# data to send to Stan
stanD <- list(N = nrow(d),
              K = ncol(x),
```

```
              M = ncol(z),
              N_obs = nrow(do),
              N_mis = nrow(dm),
              ii_obs = do$index,
              ii_mis = dm$index,
              sens = d$sens,
              se_obs = do$sens_se,
              x = x,
              z = z)

# execute model
ma <- cmdstan_model("stancode/meta_imp_reg.stan")

# launch estimation
est <- ma$sample(
  data = stanD,
  seed = 123,
  chains = 4,
  parallel_chains = 4,
  refresh=200,
  init=0,
  adapt_delta = 0.999,
  max_treedepth = 15,
  iter_warmup = 1000,
  iter_sampling = 1000,
  show_messages = TRUE,
  diagnostics = c("divergences", "treedepth", "ebfmi")
)

est$print(c("gamma","sigma", "beta","se_mis"), digits = 3,
          "mean", SE = "sd", "rhat", ~quantile(., probs = c(0.025, 0.975)) ,
           max_rows=20)
```

4.3.4 Concluding words on meta-analysis

The code above provides the fundamental building blocks for meta-analysis. That being said, how to build the model exactly will depend on the actual questions of interest. For instance, while at the lowest level we are forced to assume a normal distribution, a different type of distribution may well be called for at the aggregate level (e.g., a lognormal form usually fits better for the types of constrained parameters considered above). One may further want to explicitly model dependency between different studies stemming from the same paper, either

by additional hierarchical levels, or by cross-classification. And in the specific case above, one may want to analyze the three parameters jointly, rather than in isolation. All of this is fairly straightforward to include in Stan, and you should now have the tools to do so yourself.

4.3.5 2-stage regression as a measurement error model

Above, we have seen a way to integrate regressions directly into the models estimating the parameters. While this is a principled way of conducting regressions, it may at times not be very practical, e.g. because your estimation takes a long time to complete, and you need to run many regressions. In such cases, one can use a two-stage procedure: by explicitly treating the parameters as quantities measured with error, one can then nevertheless avoid the fallacy of standard two-stage procedures using the parameter mean, which problematically treat estimated parameters as objectively observed quantities and throw away the information we have on our confidence in the value we estimate (conditional on the model we assume). The price to pay is that we will need to assume the posterior of the parameters to follow a normal distribution. This assumption is often good enough, and of course it can be tested if you have access to the full posteriors.

Given the tools we have explored above, building such a regression is now quite trivial. We first model the parameter we want to regress—let us call it generically `theta`—as a quantity measured with uncertainty, so that `theta ~ normal(theta_hat, se);`, just like done above for meta-analysis. Except that now, `theta`is an estimated parameter and `se` its associated standard deviation of the posterior. The rest proceeds like the meta-regression discussed above, which I here implement using a robust Student-t distribution with 3 degrees of freedom:

```
data {
    int<lower=1> N;
    vector[N] theta;
    vector[N] se;

   // Design matrix used for regression
    int<lower=0> K; // dimension of the design matrix
    matrix[N,K] x; // design matrix to impute SEs
}
parameters {
    vector[N] theta_hat;
    real<lower=0> sigma;
    vector[K] beta;
}
model {
    // Priors
    sigma ~ normal(0, 2);
```

```
    beta ~ normal(0, 2);

    // Measurement error model
    theta ~ normal(theta_hat, se);

    // Regression model:
    theta_hat ~ student_t(3 , x * beta  , sigma );
}
```

For simplicity, let us use again the estimates from Bruhin, Fehr-Duda, and Epper (2010) that we have examined above, and look at the same regression for the likelihood-sensitivity parameter.

```
hc <- readRDS(file="stanoutput/RR_CH6_RDU.Rds")

zz <- hc$summary(variables = c("rho","gamma","delta","sigma"), "mean" , "sd")
zc <- zz %>% separate(variable,c("[","]")) %>%
             rename( var = `[`) %>% rename(id = `]`) %>%
             pivot_wider(names_from = var , values_from = c(mean,sd)) %>%
             rename(rho = mean_rho , gamma = mean_gamma , delta = mean_delta , sigma = mean_s
             mutate(id = as.numeric(id))

#
d <- read_csv("data/data_RR_ch06.csv")

dg <- d %>% filter(z1 > 0) %>%
             mutate(high = z1) %>%
             mutate(low = z2) %>%
             mutate(prob = p1) %>%
             group_by(id) %>%
             mutate(id = cur_group_id()) %>%
             ungroup()

di <- dg %>% group_by(id) %>%
              filter(row_number()==1) %>%
              ungroup()

z <- left_join(di,zc,by="id")

# design matrix
x <- model.matrix( ~ 1 + female + semester,z)
```

```r
# data to send to Stan
stanD <- list(N = nrow(z),
              theta = z$gamma,
              se = z$sd_gamma,
              x = x,
              K = ncol(x))

# execute model
ma <- cmdstan_model("stancode/reg2s_mem.stan")

# launch estimation
est <- ma$sample(
  data = stanD,
  seed = 123,
  chains = 4,
  parallel_chains = 4,
  refresh=200,
  init=0,
  adapt_delta = 0.85,
  max_treedepth = 12,
  iter_warmup = 1000,
  iter_sampling = 1000,
  show_messages = TRUE,
  diagnostics = c("divergences", "treedepth", "ebfmi")
)

est$save_object(file="stanoutput/reg2s_mem.Rds")
```

```r
est <- readRDS("stanoutput/reg2s_mem.Rds")


est$print(c("beta","sigma"), digits = 3,
          "mean", SE = "sd", "rhat", ~quantile(., probs = c(0.025, 0.975)) ,
           max_rows=20)
```

```
 variable   mean    SE  rhat   2.5%  97.5%
  beta[1]  0.529 0.035 1.002  0.461  0.597
  beta[2] -0.111 0.036 1.003 -0.182 -0.040
  beta[3]  0.003 0.007 1.001 -0.011  0.016
  sigma    0.185 0.013 1.003  0.161  0.213
```

We again find a negative effect of the gender dummy. The quantitative effect is indeed arbitrarily close to the one we documented above, while taking the full parameter uncertainty into account and estimating the regression jointly with the model parameters. In general, however, this approach will only work well as long as errors are at least approximatley normally distributed around the estimated parameters—something that is up to you to check when you are estimating the parameters.

4.4 Multiple hierarchical levels

We have so far seen mostly models with 2 hierarchical levels—subjects drawn from a common group may be the classical example. In many cases, however, it will be desirable to construct more complex hierarchies than the ones we have examined so far. For instance, in the 30 country model a meaningful model would treat subjects as being drawn from a national distribution, and countries as being drawn from a global distribution, thus resulting in 3 hierarchical levels. In other cases, some observations may belong to two different categories along different dimensions. Stan is very flexible in allowing for multiple hierarchical levels and cross-classification, whereby observations may at the same time belong to different categories. Such more complex models are essential when e.g. modelling stratified sampling designs. Here, I discuss how to implement such complex hierarchies for the nonlinear models we have examined.

4.4.1 Multiple hierarchies for simple models

Let us start from the simple random intercept model we discussed above. I will now use the Ethiopian data of Vieider et al. (2018), since the smaller size of the data will ensure that the example code converges quickly. As a reminder, the simple, 2-level model with random intercepts takes the following form, where I now use `n` for the individual-level intercepts, and `s` for the slope measuring likelihood-dependence, which is assumed to be the same for everybody. The line `n ~ normal(mu, tau);` uses vector notation to model the individual-level intercepts as drawn from an aggregate distribution with mean `mu` and standard deviation `tau`.

```
data{
int<lower=0> N;
int<lower=0> Nid;
vector[N] ce;
vector[N] p;
array[N] int id;
}
parameters{
real mu;
real s;
```

```
real<lower=0> sigma;
real<lower=0> tau;
vector[Nid] n;
}
model{
// hyperpriors for aggregate parameters
mu ~ std_normal( );
s ~ normal( 1 , 1 );
sigma ~ normal( 0.2 , 2 );
tau ~ normal( 0.2 , 2 );

//distribution of intercepts
n ~ normal(mu, tau);

//regression model with individual-level intercept
ce ~ normal( n[id] + s * p , sigma  ) ;
}
```

The following code runs the model and estimates the parameters:

```
d <- read_csv("data/data_ETH.csv")
d <- d %>% group_by(subject) %>%
           mutate(id = cur_group_id()) %>%
           ungroup()

# creates data to send to Stan
stanD <- list(N = nrow(d),
              ce = d$equivalent/d$high,
              p = d$probability,
              Nid = max(d$id),
              id = d$id)


# compile the Stan programme:
sr <- cmdstan_model("stancode/ri_ETH.stan")

# run the programme:
nm <- sr$sample(
  data = stanD,
  seed = 123,
  chains = 4,
  parallel_chains = 4,
```

```
  refresh=200,
  init=0,
  show_messages = TRUE,
  diagnostics = c("divergences", "treedepth", "ebfmi")
)

print(nm, c("mu","s","sigma","tau","n"),digits = 3)
```

Assume we are interested in risk attitudes across the Ethiopian rural highlands (the study population underlying the data). We might be tempted to use (some indices based on) the mean `mu` and the aggregate slope `s` to characterize such preferences. However, the model used above would introduce bias into our estimation results, simply because we are ignoring the study design. Indeed the study used a stratified design, whereby Woredas (Ethiopian administrative districts) were randomly selected from the study region. Subjects were subsequently randomly selected from these Woredas. To be able to infer the attitudes of a representative agent, we will thus need to explicitly model this sampling design (cfr. BDM3 for details).

The simplest way of introducing the additional hierarchical level is to proceed exactly like above. In particular, we will now want the individual-level intercepts `n` to have a Woreda-level mean `w`, which values corresponding to the means in the 21 Woredas in the data; and we will then distribute the Woreda-level means as being drawn from an aggregate distribution. To make sure that the dimensions agree at the different levels, it is further convenient to introduce two loops to the code:

```
data {
  int<lower=0> N;
  int<lower=0> Nid;
  int<lower=0> Nw;
  vector[N] ce;
  vector[N] p;
  array[N] int id;
  array[N] int wid;
}
parameters {
  real mu;
  real s;
  real<lower=0> sigma;
  real<lower=0> xi;
  real<lower=0> tau;
  vector[Nid] n;
  vector[Nw] w;
}
```

```
model {
  // hyperpriors for aggregate parameters
  mu ~ std_normal();
  s ~ normal(1, 1);
  sigma ~ normal(0.2, 2);
  tau ~ normal(0.2, 2);
  xi ~ normal(0.2, 2);

  // distribution of Woreda-level intercepts
  w ~ normal(mu, tau);

  // distribution of intercepts at the Woreda level
  for (n in 1:Nid)
    n[n] ~ normal(w[wid[id[n]]], xi);

  // regression model with individual-level intercept
  for (i in 1:N)
    ce[i] ~ normal(n[id[i]] + s * p[i], sigma);
}
```

```
d <- read_csv("data/data_ETH.csv")
d <- d %>% group_by(subject) %>%
           mutate(id = cur_group_id()) %>%
           ungroup() %>%
           group_by(Woreda) %>%
           mutate(wid = cur_group_id()) %>%
           ungroup()

# creates data to send to Stan
stanD <- list(N = nrow(d),
              ce = d$equivalent/d$high,
              p = d$probability,
              Nid = max(d$id),
              id = d$id,
              Nw = max(d$wid),
              wid = d$wid)


# compile the Stan programme:
sr <- cmdstan_model("stancode/ri_ETH2.stan")

# run the programme:
```

```
nm <- sr$sample(
  data = stanD,
  seed = 123,
  chains = 4,
  parallel_chains = 4,
  refresh=200,
  init=0,
  adapt_delta = 0.99,
  max_treedepth = 15,
  show_messages = TRUE,
  diagnostics = c("divergences", "treedepth", "ebfmi")
)

print(nm, c("mu","s","sigma","tau","n"),digits = 3)
```

This model works. However, it has become quite slow. I further had to crank up the `adapt_delta` metric to .99 to get reasonably convergence, and even then, a number of divergence remains—too many to look the other way. The key, as so often in Stan, is to decentralize the parameters. This is done below in the line `n = mu_w + w[wid[id[1:Nid]]] + xi * n_z;`, which moves the estimation of the individual-level estimates to the `transformed data` block, and defines them in function of the stndardized parameters in `n_z`, which follow a standard normal distribution as usual. There are two additional elements I would like to point out. The first is the indexing in `w[wid[id[1:Nid]]]`: This tells Stan that a particular `id` is associated to a particular `wid`, and that the `id` ranges from 1:Nid (Stan would otherwise attempt to include all observations, resulting in paradoxical results in the estimates `w` parameters). The second thing to note is that I have reverted from the loop in the previous version to the vectorized version of the likelihood, `ce ~ normal( n[id] + s * p , sigma ) ;`. Both will work, but this simple change brings the running time from 10 to 3 minutes on my laptop. Finally, one could, of course, also decentralize the Woreda-level intercepts in `w`. In this particular case at least, that however makes things worse rather than further improving the model.

```
data {
  int<lower=0> N;           // Number of observations
  int<lower=0> Nid;         // Number of individual-level units
  int<lower=0> Nw;          // Number of Woreda-level units
  vector[N] ce;             // Outcome variable
  vector[N] p;              // Predictor variable
  array[N] int id;          // Individual-level indices
  array[N] int wid;         // Woreda-level indices
}
parameters {
```

```
  real mu_w;                // Global mean for Woreda-level intercepts
  real<lower=0> tau_w;      // Between-Woreda variance
  real<lower=0> sigma;      // Observation-level noise
  real<lower=0> xi;         // Between-individual variance within Woredas
  vector[Nw] w;             // Woreda-level intercepts
  vector[Nid] n_z;        // Individual-level deviations (standard normal)
  real s;                   // Slope coefficient for the predictor
}
transformed parameters {
  vector[Nid] n;            // Individual-level intercepts
  n = mu_w + w[wid[id[1:Nid]]] + xi * n_z;
}
model {
  // Priors
  mu_w ~ std_normal();                // Prior on global mean
  tau_w ~ normal(0, 1);               // Prior on Woreda-level variance
  xi ~ normal(0, 1);                  // Prior on individual-level variance
  sigma ~ normal(0.2, 2);             // Prior on noise
  s ~ normal(1, 1);                   // Prior on slope

  // Woreda-level effects
  w ~ normal(0, tau_w);               // Woreda-level intercepts

  // Individual-level deviations
  n_z ~ std_normal();               // Raw individual deviations

  // Likelihood
  ce ~ normal( n[id] + s * p , sigma  ) ;
}
```

```
d <- read_csv("data/data_ETH.csv")
d <- d %>% group_by(subject) %>%
           mutate(id = cur_group_id()) %>%
           ungroup() %>%
           group_by(Woreda) %>%
           mutate(wid = cur_group_id()) %>%
           ungroup()

# creates data to send to Stan
stanD <- list(N = nrow(d),
              ce = d$equivalent/d$high,
              p = d$probability,
```

```
               Nid = max(d$id),
               id = d$id,
               Nw = max(d$wid),
               wid = d$wid)


# compile the Stan programme:
sr <- cmdstan_model("stancode/ri_ETH2_dec.stan")

# run the programme:
nm2 <- sr$sample(
  data = stanD,
  seed = 123,
  chains = 4,
  parallel_chains = 4,
  refresh=200,
  init=0,
  adapt_delta = 0.95,
  max_treedepth = 18,
  show_messages = TRUE,
  diagnostics = c("divergences", "treedepth", "ebfmi")
)

print(nm2, c("n","s","sigma"),digits = 3)
```

Of course, nothing stops us from expanding the model to including a random slope as well. The likelihood then becomes `ce ~ normal( n[id] + s[id] .* p , sigma[id] ) ;`, where `.*` implements the dot product. This ensures compatibility between the different dimensions. In addition to that, all that is required is to add a second line `s = mu_s + w_s[wid[id[1:Nid]]] + xi_s * s_raw;` in the transformed parameters block (and of course to create the new variables and to give them priors). Nevertheless, it may be desirable to collect these two lines using matrix notation, since that will allow us to add a covariance matrix. The `transformed parameters` block then becomes as follows, where the loop is being used mainly for coding transparency:

```
"'{stan, output.var="model_test.stan", eval = FALSE, tidy = FALSE} transformed parameters {   array[Nid] vector[2] pars;   vector[Nid] n;   vector[Nid] s;  for (j in 1:Nid){    pars[j] = means + W[wid[id[j]]] + diag_pre_multiply(xi, L_omega) * Z[j];    n[j] =  pars[j,1] ;     s[j] =  pars[j,2] ;   } }
```

The following code runs the model:

```r
d <- read_csv("data/data_ETH.csv")
d <- d %>% group_by(subject) %>%
           mutate(id = cur_group_id()) %>%
           ungroup() %>%
           group_by(Woreda) %>%
           mutate(wid = cur_group_id()) %>%
           ungroup()

# creates data to send to Stan
stanD <- list(N = nrow(d),
              ce = d$equivalent/d$high,
              p = d$probability,
              Nid = max(d$id),
              id = d$id,
              Nw = max(d$wid),
              wid = d$wid)


# compile the Stan programme:
sr <- cmdstan_model("stancode/rirs_ETH2_covar.stan")

# run the programme:
nm2 <- sr$sample(
  data = stanD,
  seed = 123,
  chains = 4,
  parallel_chains = 4,
  refresh=200,
  init=0,
  adapt_delta = 0.9,
  max_treedepth = 15,
  show_messages = TRUE,
  diagnostics = c("divergences", "treedepth", "ebfmi")
)

print(nm2, c("W","s","n"),digits = 3)

nm2$save_object(file="stanoutput/rirs_ETH2_dec.Rds")
```

4.4.2 Looping through all observations

Note that there is also a different possibility to efficiently introduce the additional level. It involves defining the intercept at the lowest level, call it `n_obs`, which will have N dimensions. Notice that we now have a different intercept for each data point or certainty equivalent in this case. This variable can then be defined in the `transformed parameters` block while looping through all observations. We then use `n_z` to obtain decentralized individual-level intercepts. Since we are looping through all observations, it is further straightforward to add Woreda-level effects (or indeed regression coefficients, or effects at any other level that may interest you). If you are interested in the individual-level intercepts, you may want to recover them by adding an individual-level loop in the `generated quantities` block. Notice, however, that this is not quite the same model: you are *de facto* defining a "random preference model", where preference parameters are allowed to change for each single choice, though they are constrained by having an individual-level mean. Indeed, the parameters in the likelihood, `ce ~ normal(n_obs + s_obs * p, sigma);`, are now defined at the level of the observation.

```
data {
  int<lower=0> N;           // Number of observations
  int<lower=0> Nid;         // Number of individual-level units
  int<lower=0> Nw;          // Number of Woreda-level units
  vector[N] ce;             // Outcome variable
  vector[N] p;              // Predictor variable
  array[N] int id;          // Individual-level indices
  array[N] int wid;         // Woreda-level indices
}
parameters {
  real mu;                // Global mean for Woreda-level intercepts
  real<lower=0> tau;      // Between-Woreda variance
  real<lower=0> sigma;      // Observation-level noise
  real<lower=0> xi;         // Between-individual variance within Woredas
  vector[Nw] w;             // Woreda-level intercepts
  vector[Nid] n_z;        // Individual-level deviations (standard normal)
  real s;                   // Slope coefficient for the predictor
}
transformed parameters {
  vector[N] n_obs;              // Individual-level intercepts for each observation
  for (i in 1:N)
    n_obs[i] = mu + w[wid[i]] + xi * n_z[id[i]];
}
model {
  // Priors
  mu_w ~ std_normal();               // Prior on global mean
  tau_w ~ normal(0, 1);              // Prior on Woreda-level variance
```

```
  xi ~ normal(0, 1);                    // Prior on individual-level variance
  sigma ~ normal(0.2, 2);               // Prior on noise
  s ~ normal(1, 1);                     // Prior on slope

  // Woreda-level effects
  w ~ normal(0, tau);                 // Woreda-level intercepts

  // Individual-level deviations
  n_z ~ std_normal();                 // Raw individual deviations

  // Likelihood
  ce ~ normal(n_obs + s_obs * p, sigma);      // Regression model
}
```

```
d <- read_csv("data/data_ETH.csv")
d <- d %>% group_by(subject) %>%
           mutate(id = cur_group_id()) %>%
           ungroup() %>%
           group_by(Woreda) %>%
           mutate(wid = cur_group_id()) %>%
           ungroup()

# creates data to send to Stan
stanD <- list(N = nrow(d),
              ce = d$equivalent/d$high,
              p = d$probability,
              Nid = max(d$id),
              id = d$id,
              Nw = max(d$wid),
              wid = d$wid)


# compile the Stan programme:
sr <- cmdstan_model("stancode/rirs_ETH2_obs.stan")

# run the programme:
nm3 <- sr$sample(
  data = stanD,
  seed = 123,
  chains = 4,
  parallel_chains = 4,
  refresh=200,
```

```
  init=0,
  adapt_delta = 0.9,
  max_treedepth = 15,
  show_messages = TRUE,
  diagnostics = c("divergences", "treedepth", "ebfmi")
)

print(nm2, c("W","s","n"),digits = 3)

nm3$save_object(file="stanoutput/rirs_ETH2_obs.Rds")
```

The model is more than 3 times faster to estimate than the one with the individual-level parameters only, which may seem paradoxical, given that it comprises many more parameters. This may well be a sign that it fits better, i.e. that the posterior is easier to explore. A potentially interesting question concerns the inferences we draw on the individual-level parameters. The figures below show correlations of individual-level parameters estimated in the two models. They are indeed very similar, though slightly dispersed in the observation-level model around the value in the individual-level model. The slight shift in parameters seems to stem from increased uncertainty in some parameters, which yields stronger shrinkage to the Woreda-level means. Which model you prefer depends on your substantive questions, and is for you to decide.

```
nm2 <- readRDS("stanoutput/rirs_ETH2_dec.Rds")
nm3 <- readRDS("stanoutput/rirs_ETH2_obs.Rds")


zz <- nm2$summary(variables = c("n","s"), "mean", "sd")
z2 <- zz %>% separate(variable,c("[","]")) %>%
             rename(index = "]",var = "[") %>%
             pivot_wider(names_from = var , values_from = c(mean,sd)) %>%
             mutate(index = as.integer(index)) %>%
             rename(n = mean_n,s = mean_s)

zz <- nm3$summary(variables = c("n","s"), "mean", "sd")
z3 <- zz %>% separate(variable,c("[","]")) %>%
             rename(index = "]",var = "[") %>%
             pivot_wider(names_from = var , values_from = c(mean,sd)) %>%
             mutate(index = as.integer(index)) %>%
             rename(no = mean_n,so = mean_s)

z <- left_join(z2,z3,by="index")
```

```
ggplot(z,aes(x=n,y=no)) +
  geom_point( color="blue") +
  geom_abline(intercept=0,slope=1) +
  labs(x="intercept, individual model", y= "intercept, observational model")
```

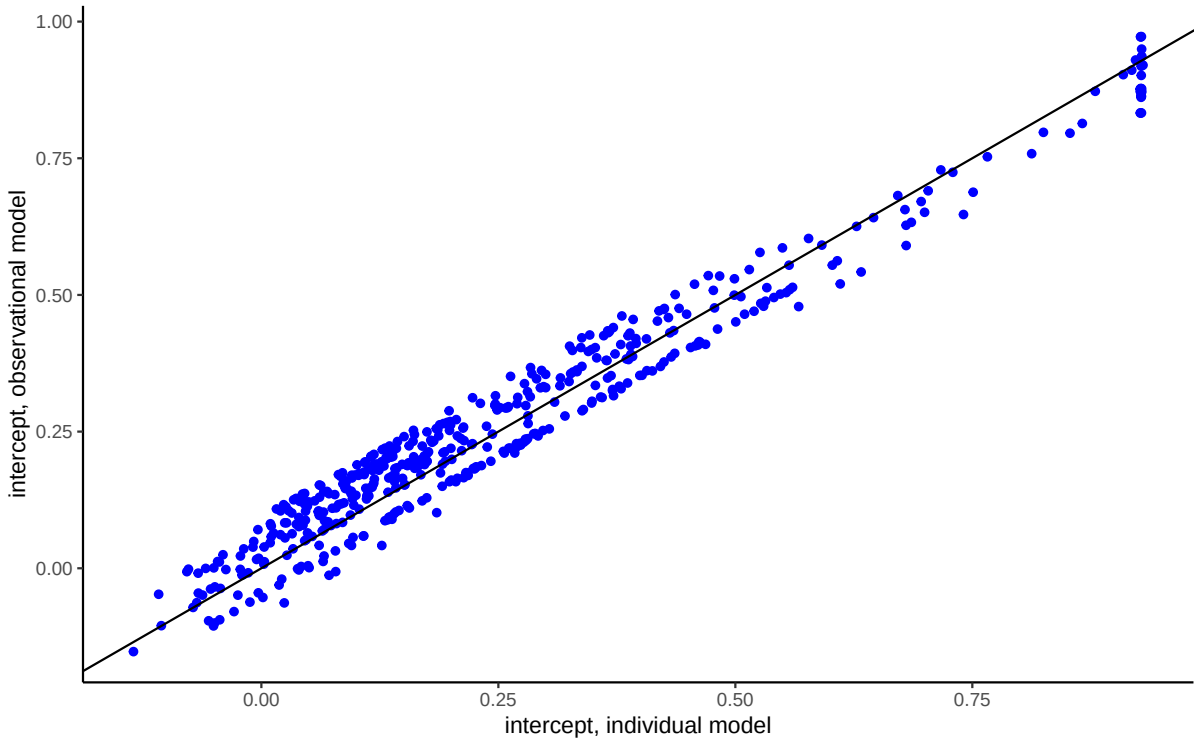


Figure 4.13: Individual intercept parameters in the two mutli-hierarchy models

4.5 Conclusion

This is the end as of 03 September 2024. Stand by for additional sections to be added soon.

References

Abdellaoui, Mohammed, Aurélien Baillon, Lætitia Placido, and Peter P. Wakker. 2011. "The Rich Domain of Uncertainty: Source Functions and Their Experimental Implementation." *American Economic Review* 101: 695–723.

Bouchouicha, Ranoua, and Ferdinand M. Vieider. 2017. "Accommodating Stake Effects Under Prospect Theory." *Journal of Risk and Uncertainty* 55 (1): 1–28.

Bruhin, Adrian, Helga Fehr-Duda, and Thomas Epper. 2010. "Risk and Rationality: Uncovering Heterogeneity in Probability Distortion." *Econometrica* 78 (4): 1375–1412.

Gelman, Andrew, John B Carlin, Hal S Stern, David B Dunson, Aki Vehtari, and Donald B Rubin. 2014. *Bayesian Data Analysis 3*. CRC press Boca Raton, FL.

Kruschke, John K. 2013. "Bayesian Estimation Supersedes the t-Test." *Journal of Experimental Psychology: General* 142 (2): 573.

L'Haridon, Olivier, and Ferdinand M. Vieider. 2019. "All over the Map: A Worldwide Comparison of Risk Preferences." *Quantitative Economics* 10: 185–215.

McElreath, Richard. 2016. *Statistical Rethinking: A Bayesian Course with Examples in r and Stan*. Academic Press.

Vieider, Ferdinand M. 2024. "Decisions Under Uncertainty as Bayesian Inference on Choice Options." *Management Science, Forthcoming*.

Vieider, Ferdinand M., Abebe Beyene, Randall A. Bluffstone, Sahan Dissanayake, Zenebe Gebreegziabher, Peter Martinsson, and Alemu Mekonnen. 2018. "Measuring Risk Preferences in Rural Ethiopia." *Economic Development and Cultural Change* 66 (3): 417–46.