Bayesian Ridge Regression - Final Report

Dominik Strache, Nicolai Bäuerle & Joel Beck

Contents

Introduction

1 Mathematical Model

This chapter is dedicated to the underlying theoretical model of Bayesian Ridge Regression in the context of the Gaussian Location-Scale Regression model that the `lmls` package is based on. First, the distributional assumptions for the *location* parameter $\boldsymbol{\beta}$, the *scale* parameter $\boldsymbol{\gamma}$ and the prior variances τ^2 and ξ^2, which are specific to Bayesian models, are clearly stated. Based on these prior distributions, the full conditional distributions of each parameter given all of the remaining model components are derived. Throughout this rather theoretical exposition we will build connections from the derived equations to practical consequences that have to be kept in mind for the code implementation discussed in sections 2.2.1 and 2.2.2.

1.1 Prior Distributions

Assuming conditional independence among the regression coefficients $\boldsymbol{\beta}$ and $\boldsymbol{\gamma}$ as well as flat priors for the intercept parameters

$$f(\beta_0) \propto const \qquad \text{and} \qquad f(\gamma_0) \propto const,$$

first note that

$$\begin{aligned} f(\boldsymbol{\beta} \mid \tau^2) &= f(\beta_0) f(\tilde{\boldsymbol{\beta}} \mid \tau^2) \propto f(\tilde{\boldsymbol{\beta}} \mid \tau^2) \quad \text{and} \\ f(\boldsymbol{\gamma} \mid \xi^2) &= f(\gamma_0) f(\tilde{\boldsymbol{\gamma}} \mid \xi^2) \propto f(\tilde{\boldsymbol{\gamma}} \mid \xi^2), \end{aligned}$$

where the notation $\boldsymbol{\beta} = (\beta_0, ..., \beta_K) \in \mathbb{R}^{K+1}$, $\tilde{\boldsymbol{\beta}} = (\beta_1, ..., \beta_K) \in \mathbb{R}^K$ and, analogously, $\boldsymbol{\gamma} = (\gamma_0, ..., \gamma_J) \in \mathbb{R}^{J+1}$, $\tilde{\boldsymbol{\gamma}} = (\gamma_1, ..., \gamma_J) \in \mathbb{R}^J$ is used.

Thus, the Prior distributions of the parameters in the Bayesian Ridge Regression model, which are responsible for the regularizing effect compared to models without penalty, are given by

- $\tilde{\boldsymbol{\beta}} \mid \tau^2 \sim \mathcal{N}\left(\mathbf{0},\, \tau^2 \cdot \mathbf{I}_K\right)$,
- $\tilde{\boldsymbol{\gamma}} \mid \xi^2 \sim \mathcal{N}\left(\mathbf{0},\, \xi^2 \cdot \mathbf{I}_J\right)$,
- $\tau^2 \sim IG(a_\tau,\, b_\tau)$, with fixed hyperparameters a_τ and b_τ,
- $\xi^2 \sim IG(a_\xi,\, b_\xi)$, with fixed hyperparameters a_ξ and b_ξ.

1.2 Full Posterior Distribution

The starting point for deriving the Full Conditional distributions, which will majorly impact the implementation of the Metropolis-Hastings sampling process, is always the Full Posterior distribution. As usual in the Bayesian literature, we write the Full Posterior $f(\boldsymbol{\beta}, \boldsymbol{\gamma}, \tau^2, \xi^2 \mid \mathbf{y})$ in terms of the Likelihood function / observation model $f(\mathbf{y} \mid \boldsymbol{\beta}, \boldsymbol{\gamma}, \tau^2, \xi^2)$ and the *joint* Prior distribution $f(\boldsymbol{\beta}, \boldsymbol{\gamma}, \tau^2, \xi^2)$, i.e.

$$f(\boldsymbol{\beta}, \boldsymbol{\gamma}, \tau^2, \xi^2 \mid \mathbf{y}) \propto f(\mathbf{y} \mid \boldsymbol{\beta}, \boldsymbol{\gamma}, \tau^2, \xi^2) \cdot f(\boldsymbol{\beta}, \boldsymbol{\gamma}, \tau^2, \xi^2).$$

The specific form of the Likelihood function is given by the Location-Scale Regression model that the `lmls` package is built upon:

$$y_i \mid \boldsymbol{\beta}, \boldsymbol{\gamma}, \tau^2, \xi^2 = y_i \mid \boldsymbol{\beta}, \boldsymbol{\gamma} \sim \mathcal{N}\left(\mathbf{x}_i^T \boldsymbol{\beta},\, \exp\left(\mathbf{z}_i^T \boldsymbol{\gamma}\right)^2\right).$$

Taking the independence structure into account, the joint Prior distribution can be written as

$$\begin{aligned} f(\boldsymbol{\beta}, \boldsymbol{\gamma}, \tau^2, \xi^2) &= f(\boldsymbol{\beta} \mid \boldsymbol{\gamma}, \tau^2, \xi^2) \cdot f(\boldsymbol{\gamma}, \tau^2, \xi^2) \\ &= f(\boldsymbol{\beta} \mid \boldsymbol{\gamma}, \tau^2, \xi^2) \cdot f(\boldsymbol{\gamma} \mid \tau^2, \xi^2) \cdot f(\tau^2, \xi^2) \\ &= f(\boldsymbol{\beta} \mid \tau^2) \cdot f(\boldsymbol{\gamma} \mid \xi^2) \cdot f(\tau^2) \cdot f(\xi^2). \end{aligned}$$

Combining these results yields for the Full Posterior distribution the general form

$$\begin{aligned} f(\boldsymbol{\beta}, \boldsymbol{\gamma}, \tau^2, \xi^2 \mid \mathbf{y}) &\propto f(\mathbf{y} \mid \boldsymbol{\beta}, \boldsymbol{\gamma}, \tau^2, \xi^2) \cdot f(\boldsymbol{\beta} \mid \tau^2) \cdot f(\boldsymbol{\gamma} \mid \xi^2) \cdot f(\tau^2) \cdot f(\xi^2) \\ &\propto f(\mathbf{y} \mid \boldsymbol{\beta}, \boldsymbol{\gamma}) \cdot f(\tilde{\boldsymbol{\beta}} \mid \tau^2) \cdot f(\tilde{\boldsymbol{\gamma}} \mid \xi^2) \cdot f(\tau^2) \cdot f(\xi^2), \end{aligned}$$

in which the corresponding densities for the observation model and the individual prior distributions can be inserted. These are given by

- $f(\mathbf{y} \mid \boldsymbol{\beta}, \boldsymbol{\gamma}) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi \exp\left(\mathbf{z}_i^T \boldsymbol{\gamma}\right)^2}} \cdot \exp\left(-\frac{1}{2\exp\left(\mathbf{z}_i^T\boldsymbol{\gamma}\right)^2} \cdot \left(y_i - \mathbf{x}_i^T\boldsymbol{\beta}\right)^2\right)$,
- $f(\boldsymbol{\beta} \mid \tau^2) \propto \prod_{k=1}^{K} \frac{1}{\sqrt{2\pi\tau^2}} \cdot \exp\left(-\frac{1}{2\tau^2} \cdot \beta_k^2\right) = (2\pi)^{-\frac{K}{2}}\, \tau^{-K} \exp\left(-\frac{1}{2\tau^2} \cdot \tilde{\boldsymbol{\beta}}^T \tilde{\boldsymbol{\beta}}\right)$,
- $f(\boldsymbol{\gamma} \mid \xi^2) \propto \prod_{j=1}^{J} \frac{1}{\sqrt{2\pi\xi^2}} \cdot \exp\left(-\frac{1}{2\xi^2} \cdot \gamma_j^2\right) = (2\pi)^{-\frac{J}{2}}\, \xi^{-J} \exp\left(-\frac{1}{2\xi^2} \cdot \tilde{\boldsymbol{\gamma}}^T \tilde{\boldsymbol{\gamma}}\right)$,
- $f(\tau^2) = \frac{b_\tau}{\Gamma(a_\tau)} \left(\frac{1}{\tau^2}\right)^{a_\tau + 1} \exp\left(-\frac{b_\tau}{\tau^2}\right)$,
- $f(\xi^2) = \frac{b_\xi}{\Gamma(a_\xi)} \left(\frac{1}{\xi^2}\right)^{a_\xi + 1} \exp\left(-\frac{b_\xi}{\xi^2}\right)$,

leading to the Full Posterior

$$\begin{aligned} f(\boldsymbol{\beta}, \boldsymbol{\gamma}, \tau^2, \xi^2 \mid \mathbf{y}) \propto & \prod_{i=1}^{n} \frac{1}{\exp\left(\mathbf{z}_i^T\boldsymbol{\gamma}\right)} \cdot \exp\left(-\frac{1}{2\exp\left(\mathbf{z}_i^T\boldsymbol{\gamma}\right)^2} \cdot \left(y_i - \mathbf{x}_i^T\boldsymbol{\beta}\right)^2\right) \\ & \cdot \tau^{-K} \exp\left(-\frac{1}{2\tau^2}\tilde{\boldsymbol{\beta}}^T\tilde{\boldsymbol{\beta}}\right) \cdot \xi^{-J} \exp\left(-\frac{1}{2\xi^2}\tilde{\boldsymbol{\gamma}}^T\tilde{\boldsymbol{\gamma}}\right) \\ & \cdot \left(\frac{1}{\tau^2}\right)^{a_\tau+1} \exp\left(-\frac{b_\tau}{\tau^2}\right) \cdot \left(\frac{1}{\xi^2}\right)^{a_\xi+1} \exp\left(-\frac{b_\xi}{\xi^2}\right). \end{aligned}$$

1.3 Full Conditional Distributions

The Full Posterior distribution contains complete information about the statistical model. For our purposes, we are mostly interested in the Full Conditional Distribution of each model parameter. These can be obtained by simply neglecting all factors of the Full Posterior that do not depend on the parameter in consideration.

The Full Conditional distribution can then be recovered by the resulting density kernel, either by recognizing a known distribution or by adding a normalization constant (which, however, is not needed for Markov Chain Monte Carlo sampling).

1.3.1 Full Conditional of τ^2:

$$\begin{aligned} f(\tau^2 \mid \cdot) &\propto \tau^{-K} \cdot \exp\left(-\frac{1}{2\tau^2} \cdot \tilde{\boldsymbol{\beta}}^T\tilde{\boldsymbol{\beta}}\right) \cdot \left(\frac{1}{\tau^2}\right)^{a_\tau+1} \exp\left(-\frac{b_\tau}{\tau^2}\right) \\ &\propto \left(\frac{1}{\tau^2}\right)^{a_\tau + \frac{K}{2} + 1} \cdot \exp\left(-\frac{1}{\tau^2}\left(b_\tau + \frac{1}{2}\tilde{\boldsymbol{\beta}}^T\tilde{\boldsymbol{\beta}}\right)\right). \end{aligned}$$

This is the kernel of an Inverse-Gamma distribution parameterized by

$$\tau^2 \mid \cdot \sim IG(a_\tau + \frac{K}{2},\, b_\tau + \frac{1}{2}\tilde{\boldsymbol{\beta}}^T\tilde{\boldsymbol{\beta}}).$$

1.3.2 Full Conditional of ξ^2:

$$\begin{aligned} f(\xi^2 \mid \cdot) &\propto \xi^{-J} \cdot \exp\left(-\frac{1}{2\xi^2} \cdot \tilde{\boldsymbol{\gamma}}^T\tilde{\boldsymbol{\gamma}}\right) \cdot \left(\frac{1}{\xi^2}\right)^{a_\xi+1} \exp\left(-\frac{b_\xi}{\xi^2}\right) \\ &\propto \left(\frac{1}{\xi^2}\right)^{a_\xi + \frac{J}{2} + 1} \cdot \exp\left(-\frac{1}{\xi^2}\left(b_\xi + \frac{1}{2}\tilde{\boldsymbol{\gamma}}^T\tilde{\boldsymbol{\gamma}}\right)\right). \end{aligned}$$

Thus, the Full Conditional of ξ^2 follows an Inverse-Gamma distribution as well:

$$\xi^2 \mid \cdot \sim IG(a_\xi + \frac{J}{2},\, b_\xi + \frac{1}{2}\tilde{\boldsymbol{\gamma}}^T\tilde{\boldsymbol{\gamma}}).$$

1.3.3 Full Conditional of $\boldsymbol{\beta}$:

Here, the derivation is more involved. In order to keep the calculations structured, we introduce the following notation:

$$\mathbf{w}_i := \frac{\mathbf{x}_i}{\exp\left(\mathbf{z}_i^T\boldsymbol{\gamma}\right)} \in \mathbb{R}^{K+1}, \qquad \mathbf{W} := \begin{pmatrix} \mathbf{w}_1^T \\ \vdots \\ \mathbf{w}_n^T \end{pmatrix} \in \mathbb{R}^{n\,\times\,(K+1)}$$

and

$$u_i := \frac{y_i}{\exp\left(\mathbf{z}_i^T\boldsymbol{\gamma}\right)} \in \mathbb{R}, \qquad \mathbf{u} := \begin{pmatrix} u_1 \\ \vdots \\ u_n \end{pmatrix} \in \mathbb{R}^n,$$

yielding $\sum_{i=1}^n u_i\mathbf{w}_i = \mathbf{W}^T\mathbf{u}$ and $\sum_{i=1}^n \mathbf{w}_i\mathbf{w}_i^T = \mathbf{W}^T\mathbf{W}$.

Therefore the Full Conditional distribution of $\boldsymbol{\beta}$ can be written as

$$\begin{aligned}
f(\boldsymbol{\beta} \mid \cdot) &\propto \exp\left(-\frac{1}{2}\cdot\left[\frac{1}{\tau^2}\tilde{\boldsymbol{\beta}}^T\tilde{\boldsymbol{\beta}} + \sum_{i=1}^n \frac{1}{\exp\left(\mathbf{z}_i^T\boldsymbol{\gamma}\right)^2}\left(y_i - \mathbf{x}_i^T\boldsymbol{\beta}\right)^2\right]\right) \\
&= \exp\left(-\frac{1}{2}\cdot\left[\frac{1}{\tau^2}\tilde{\boldsymbol{\beta}}^T\tilde{\boldsymbol{\beta}} + \sum_{i=1}^n\left(\frac{y_i^2}{\exp\left(\mathbf{z}_i^T\boldsymbol{\gamma}\right)^2} - \frac{2y_i\mathbf{x}_i^T}{\exp\left(\mathbf{z}_i^T\boldsymbol{\gamma}\right)^2}\boldsymbol{\beta} + \boldsymbol{\beta}^T\frac{\mathbf{x}_i}{\exp\left(\mathbf{z}_i^T\boldsymbol{\gamma}\right)}\frac{\mathbf{x}_i^T}{\exp\left(\mathbf{z}_i^T\boldsymbol{\gamma}\right)}\boldsymbol{\beta}\right)\right]\right) \\
&\propto \exp\left(-\frac{1}{2}\cdot\left[\frac{1}{\tau^2}\tilde{\boldsymbol{\beta}}^T\tilde{\boldsymbol{\beta}} - 2\cdot\sum_{i=1}^n u_i\mathbf{w}_i^T\boldsymbol{\beta} + \sum_{i=1}^n \boldsymbol{\beta}^T\mathbf{w}_i\mathbf{w}_i^T\boldsymbol{\beta}\right]\right) \\
&= \exp\left(-\frac{1}{2}\cdot\left[\boldsymbol{\beta}^T\left(\sum_{i=1}^n \mathbf{w}_i\mathbf{w}_i^T + \frac{1}{\tau^2}\begin{pmatrix} 0 & 0 \\ 0 & \mathbf{I}_K\end{pmatrix}\right)\boldsymbol{\beta} - 2\cdot\sum_{i=1}^n\boldsymbol{\beta}^T u_i\mathbf{w}_i\right]\right) \\
&= \exp\left(-\frac{1}{2}\cdot\left[\boldsymbol{\beta}^T\left(\mathbf{W}^T\mathbf{W} + \frac{1}{\tau^2}\begin{pmatrix} 0 & 0 \\ 0 & \mathbf{I}_K\end{pmatrix}\right)\boldsymbol{\beta} - 2\cdot\boldsymbol{\beta}^T\mathbf{W}^T\mathbf{u}\right]\right).
\end{aligned}$$

Comparing this representation with the kernel of a multivariate normal distribution leads to the conclusion

$$\boldsymbol{\beta} \mid \cdot \sim \mathcal{N}_{K+1}\left(\boldsymbol{\mu}_\beta, \boldsymbol{\Sigma}_\beta\right)$$

with the parameters

$$\boldsymbol{\Sigma}_\beta = \left(\mathbf{W}^T\mathbf{W} + \frac{1}{\tau^2}\begin{pmatrix} 0 & 0 \\ 0 & \mathbf{I}_K\end{pmatrix}\right)^{-1} \qquad \text{and} \qquad \boldsymbol{\mu}_\beta = \boldsymbol{\Sigma}_\beta\mathbf{W}^T\mathbf{u}.$$

1.3.4 Full Conditional of $\boldsymbol{\gamma}$:

Using the notation

$$\mathbf{z}_i \in \mathbb{R}^{J+1}, \qquad \mathbf{Z} = \begin{pmatrix} \mathbf{z}_1^T \\ \vdots \\ \mathbf{z}_n^T \end{pmatrix} \in \mathbb{R}^{n\,\times\,(J+1)} \qquad \text{and} \qquad \boldsymbol{\gamma} \in \mathbb{R}^{J+1},$$

the Full Conditional distribution of $\boldsymbol{\gamma}$ is given by

$$\begin{aligned}
f(\boldsymbol{\gamma} \mid \cdot) &\propto \exp\left(-\frac{1}{2}\cdot\left[\frac{1}{\xi^2}\tilde{\boldsymbol{\gamma}}^T\tilde{\boldsymbol{\gamma}} + \sum_{i=1}^n\left(\frac{1}{\exp\left(\mathbf{z}_i^T\boldsymbol{\gamma}\right)^2}\left(y_i - \mathbf{x}_i^T\boldsymbol{\beta}\right)^2 + 2\cdot\mathbf{z}_i^T\boldsymbol{\gamma}\right)\right]\right) \\
&= \exp\left(-\frac{1}{2}\cdot\left[\frac{1}{\xi^2}\tilde{\boldsymbol{\gamma}}^T\tilde{\boldsymbol{\gamma}} + 2\cdot\mathbf{1}_n^T\mathbf{Z}\boldsymbol{\gamma} + \sum_{i=1}^n\left(\frac{y_i - \mathbf{x}_i^T\boldsymbol{\beta}}{\exp\left(\mathbf{z}_i^T\boldsymbol{\gamma}\right)}\right)^2\right]\right).
\end{aligned}$$

In contrast to the Full Conditionals for $\boldsymbol{\beta}$, τ^2 and ξ^2, this kernel cannot be assigned to a known distribution. Thus, for sampling from the Full Posterior distribution, it is not feasible to use a Gibbs Sampler in its purest form. More specifically, we will include a Metropolis Hastings step for sampling the $\boldsymbol{\gamma}$ parameter vector.

Although this 'inconvenience' is not required for $\boldsymbol{\beta}$ (since we can use independent samples from a multivariate normal distribution), we have explored and analyzed the statistical properties of sampling both $\boldsymbol{\gamma}$ and $\boldsymbol{\beta}$ via the Metropolis-Hastings procedure, which will be briefly discussed in the following chapters.

2 The `asp21bridge` Package

This chapter introduces numerous facets of the `asp21bridge` package.

Section 2.1 explains how to use and combine the functions that are contained in the package most efficiently.

Section 2.2 focuses on the implementation of the Markov Chain Monte Carlo Sampler with Ridge Penalty, links the source code to the mathematical model from chapter 1 and explains how the main function `mcmc_ridge()` as well as internal functions implementing e.g. the Metropolis-Hastings algorithm are structured.

Finally, some additional components of the package development process and ideas, that the package is built upon, are discussed in section 2.3.

2.1 User Guide

This section aims to provide guidance for new users of the `asp21bridge` package. Although all exported functions are fully documented such that function arguments and brief examples can be looked up at the corresponding help page, the following tutorial far extends the documentation by illustrating a typical workflow of simulation via the penalized MCMC Sampler, extracting meaningful statistical quantities from the samples and visually analyzing the results.

The `asp21bridge` package inherits all functions from the `lmls` package and exports 9 additional functions, which can be grouped into three categories:

- **Sampling:**
  The whole sampling process is covered by the very flexible and robust `mcmc_ridge()` function that is explained in detail in section 2.2.1.
- **Graphical Analysis:**
  The `trace_plot()`, `density_plot()` and `acf_plot()` functions provide the three most common visualizations of a *single* chain's development over time, its distribution and the autocorrelation between the samples. Since all of these are *diagnostic* tools, they are combined in the high-level `diagnostic_plots()` function, which simply collects all three plots in a grid and is recommended to use for most applications.

  For visualizing *multiple* chains together, the `mult_plot()` function can be used. Several arguments for customizing the graphical output exist. In the most basic form, trace plots of all selected chains are displayed in the upper panel while the lower panel contains the corresponding density plots.
- **Statistical Analysis:**
  Here, the `summary_complete()` function represents the main tool and provides a more thorough statistical summary than the generic `summary()` function. In addition, the `burnin()` and `thinning()` functions are convenient in case the chains are highly autocorrelated.

2.1.1 Sampling with the `mcmc_ridge()` function

As explained in section 2.2.1, there are two valid input types for simulating with the `mcmc_ridge()` function. Most commonly, the sampling procedure will be built upon an existing model that was initialized by the `lmls()` function. In this case, only the name of the model object is required, although many further arguments can be specified such as the number of simulations `nsim`, which is set to 1000 by default.

The `mcmc_ridge()` function can, however, be used completely independent from the underlying `lmls` package. Therefore, the outcome vector $\mathbf{y}$ as well as the design matrices $\mathbf{X}$ and $\mathbf{Z}$, corresponding to the $\boldsymbol{\beta}$ and $\boldsymbol{\gamma}$ coefficient vectors, respectively (as explained in chapter 1), must be manually specified.

In order to illustrate both options, we construct two separate models:

- The first model uses the built-in `toy_data`, a data set which is specifically designed for tutorials, documentation and unit tests. It consists of a column `y` representing a vector of observed values and explanatory variables `x1`, `x2`, `z1` and `z2`.

The data is simulated according to the correctly specified location-scale regression model from chapter 1, where all explanatory variables predict the mean of **y** and only the latter two model the variance. This example will be used for model evaluation, since the true data generating values $\boldsymbol{\beta} = \begin{pmatrix} 0 & -2 & -1 & 1 & 2 \end{pmatrix}^T$ and $\boldsymbol{\gamma} = \begin{pmatrix} 0 & -1 & 1 \end{pmatrix}^T$ are known.

For the simulations, we use the model object that is created by the `lmls()` function as input and use most of the `mcmc_ridge()` default settings, except for the number of simulations, which we increase to 10.000.

- The second model uses the more realistic `abdom` data set from the `lmls` package.

  In this case, we have to provide the data input as well as starting values for $\boldsymbol{\beta}$ and $\boldsymbol{\gamma}$ explicitly. Note that the dimension of `beta_start` and `gamma_start` have to match the number of columns in `X` and `Z` in the model *input*. If necessary, the `mcmc_ridge()` functions adds intercept columns to both design matrices, such that the dimension of the coefficient vectors might have increased (as in the case illustrated here) in the model *output*.

  This example differs from the first one with regards to the conclusions that can be drawn: Since the true data generating values of $\boldsymbol{\beta} = \begin{pmatrix} \beta_0 & \beta_1 \end{pmatrix}^T$ and $\boldsymbol{\gamma} = \begin{pmatrix} \gamma_0 & \gamma_1 \end{pmatrix}^T$ are *not* known, we have to rely more heavily on the model estimates. The number of simulations is again increased to 10.000, such that statistically valid estimates of posterior characteristics are possible, even if preceding `burnin()` and `thinning()` steps might be required.

  Note, that we first standardize the covariates in this case, which is strongly recommended when working with penalized methods.

```r
library(asp21bridge)
set.seed(1234)

toy_fit <- lmls(
  location = y ~ x1 + x2 + z1 + z2, scale = ~ z1 + z2,
  data = toy_data, light = FALSE
) %>%
  mcmc_ridge(num_sim = 10000)

standardize <- function(x) (x - mean(x)) / sd(x)
y <- abdom$y
X <- as.matrix(standardize(abdom$x))
Z <- X

abdom_fit <- mcmc_ridge(
  y = y, X = X, Z = Z, beta_start = 1, gamma_start = 1, num_sim = 10000
)
```

The different types of data input cause different structures in the resulting model objects: `toy_fit` inherits the model structure as well as the S3 Class `lmls` from the `lmls()` function and adds a list entry `mcmc_ridge` containing the sampling results (refer to section 2.2.1 for details).

In contrast, `abdom_fit` only contains matrices filled with the simulations and the acceptance probability of the Metropolis-Hastings step for sampling $\boldsymbol{\gamma}$:

```r
str(abdom_fit, max.level = 1)
## List of 2
##  $ sampling_matrices:List of 4
##  $ acceptance_rate  : num 0.305
```

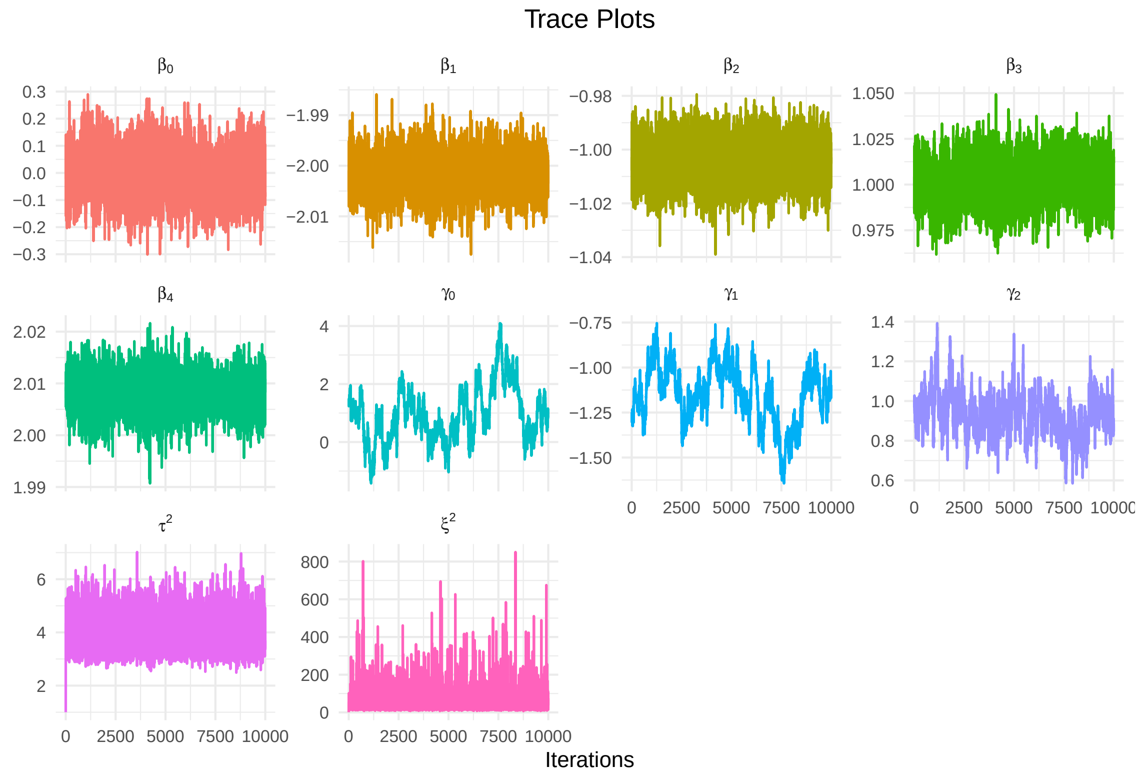

Figure 1: Trace Plots for all Coefficients, data: toy_data

2.1.2 Graphical Analysis

Most statistical analyses start with an exploratory phase. To obtain a graphical overview of the simulations, the `mult_plot()` function is convenient to display trace plots and/or density plots for all model coefficients.

The `free_scale` argument is often useful to obtain a meaningful graphical output, if the parameters are on different numerical scales. Setting `latex = TRUE` transforms the coefficient names to their corresponding greek symbols, which, although being a purely aesthetic feature, required a surprisingly nontrivial implementation.

```r
mult_plot(samples = toy_fit, type = "trace", free_scale = TRUE, latex = TRUE)
```

Figure 1 shows trace plots for all coefficients of the `toy_fit` model. Due to the high number of simulations, the plot facets appear a bit overloaded. Considering the y-axis scales, the samples for the $\boldsymbol{\beta}$ vector show a small variance and fast convergence, whereas the samples for γ_0 and γ_1 indicate a significant autocorrelation and no sign of convergence.

The `log` argument can be useful for displaying variance parameters such as τ^2 and ξ^2, which are strictly positive. Figure 2 illustrates the effect of this option on the y-axis of the trace plots as well as the x-axis of the density plots.

```r
variance_samples <- cbind(
  abdom_fit$sampling_matrices$tau_samples,
  abdom_fit$sampling_matrices$xi_samples
)

mult_plot(
  samples = variance_samples, type = "both", free_scale = TRUE,
  log = TRUE, latex = TRUE
)
```

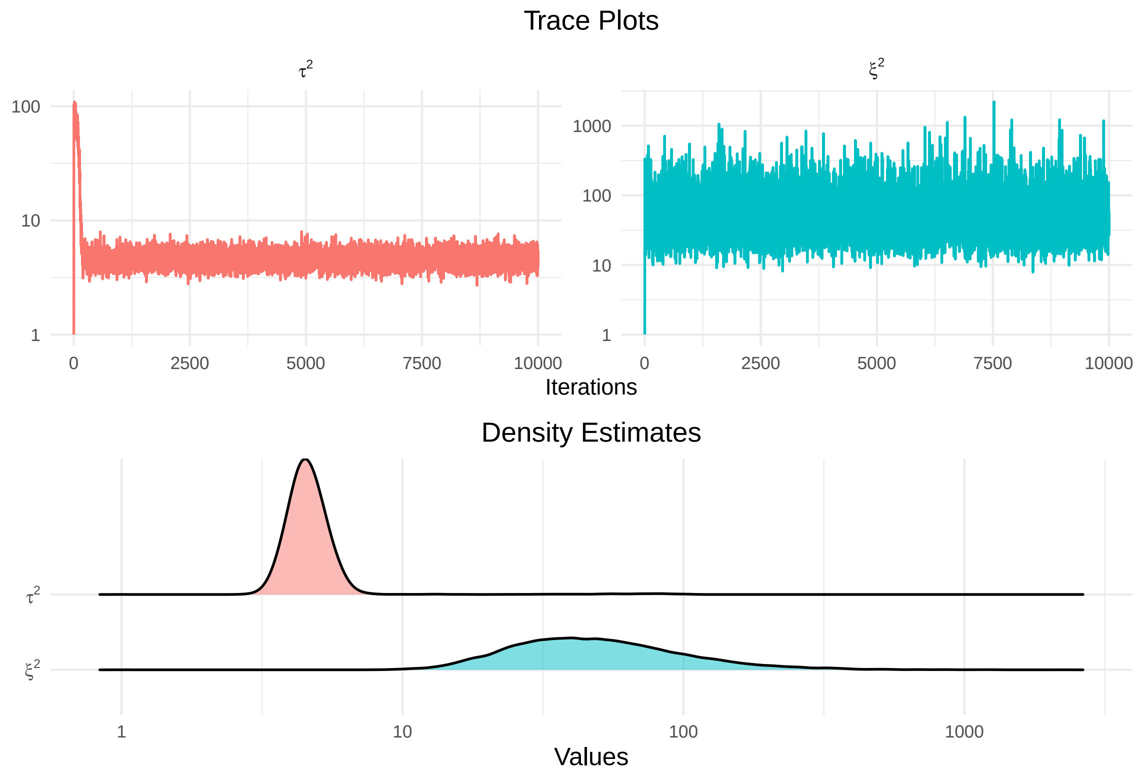

Figure 2: Trace and Density Plots for the Prior Variances, data: abdom

Note that the `mult_plot()` function accepts various kinds of input data, such as a complete `lmls` model in the first case or simply one or multiple sampling matrices in the latter case, and correctly extracts the required samples.

To focus on a single Markov chain, the `diagnostic_plots()` function is useful:

```
diagnostic_plots(
  samples = abdom_fit$sampling_matrices$beta_samples[, "beta_1", drop = FALSE],
  lag_max = 100, latex = TRUE
)
```

Figure 3 clearly shows the need for a `burnin()` and `thinning()` step for β_1, since the chain varies heavily among the first iterations and the samples are strongly autocorrelated. The `lag_max` argument controls the number of lags that are included in the autocorrelation plot.

In order to confront these issues, we remove the first 500 samples and additionally only keep every 10th iteration:

```
abdom_fit$sampling_matrices$beta_samples[, "beta_1", drop = FALSE] %>%
  burnin(num_burn = 500) %>%
  thinning(freq = 10) %>%
  diagnostic_plots(lag_max = 100, latex = TRUE)
```

The diagnostic plots for β_1 in Figure 4 look much better. The chain has converged and there is little residual autocorrelation left after thinning out the samples. Additionally, the posterior density approximately follows a normal distribution.

Although removing posterior estimates in this way (and arguably losing valuable information) is not uncontroversial in the Bayesian community, statistical estimates from the remaining, well behaved chain are usually

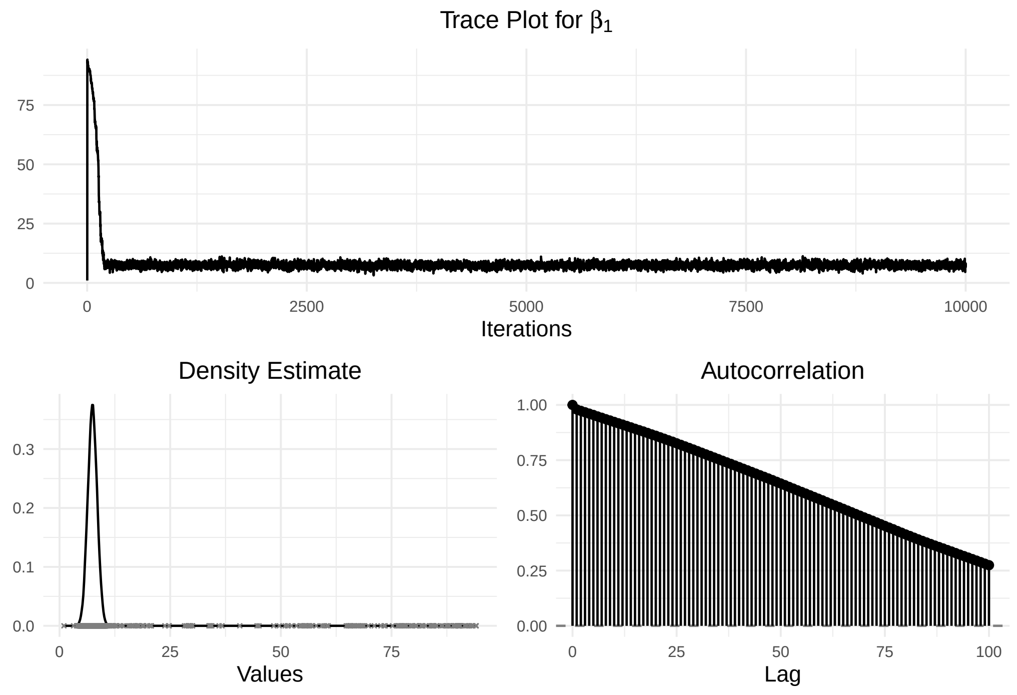

Figure 3: Diagnostic Plots for β_1 before thinning, data: abdom

more stable and reliable. In any case, the above procedure clearly emphasizes the usefulness of a graphical diagnosis of the sampling results as a preliminary step before drawing any far reaching conclusions.

2.1.3 Statistical Analysis

At the end of a Bayesian statistical analysis, summary statistics of the posterior distribution are of major interest. These include estimates for *centrality*, such as the Posterior Mean or the Posterior Median, estimates for the *spread*, e.g. the Posterior Variance, and quantile estimates in the tails of the posterior distribution as bounds for credible intervals.

The `asp21bridge` package offers two options to quickly access this information: If only a quick look at the numerical results without further investigation is desired, the `summary()` function can be used. The `lmls` package adapts this generic S3 method to the `lmls` class with an additional `type` argument. Specifying `mcmc_ridge` displays the estimates of the `mcmc_ridge()` function, in this case the Posterior Mean, the Posterior Median and the 0.025 and 0.975 quantiles of the Posterior distribution.

Note, that this option is only applicable for the `toy_fit` model, which inherits the `lmls` class:

```
summary(toy_fit, type = "mcmc_ridge")
##
## Call:
## lmls(location = y ~ x1 + x2 + z1 + z2, scale = ~z1 + z2, data = toy_data,
##     light = FALSE)
##
## Pearson residuals:
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -3.29800 -0.38660  0.11770 -0.01354  0.57410  2.54600
##
## Location coefficients (identity link function):
##               Mean       2.5%        50%  97.5%
## beta_0  0.0003161 -0.1488290 -0.0013337  0.156
```

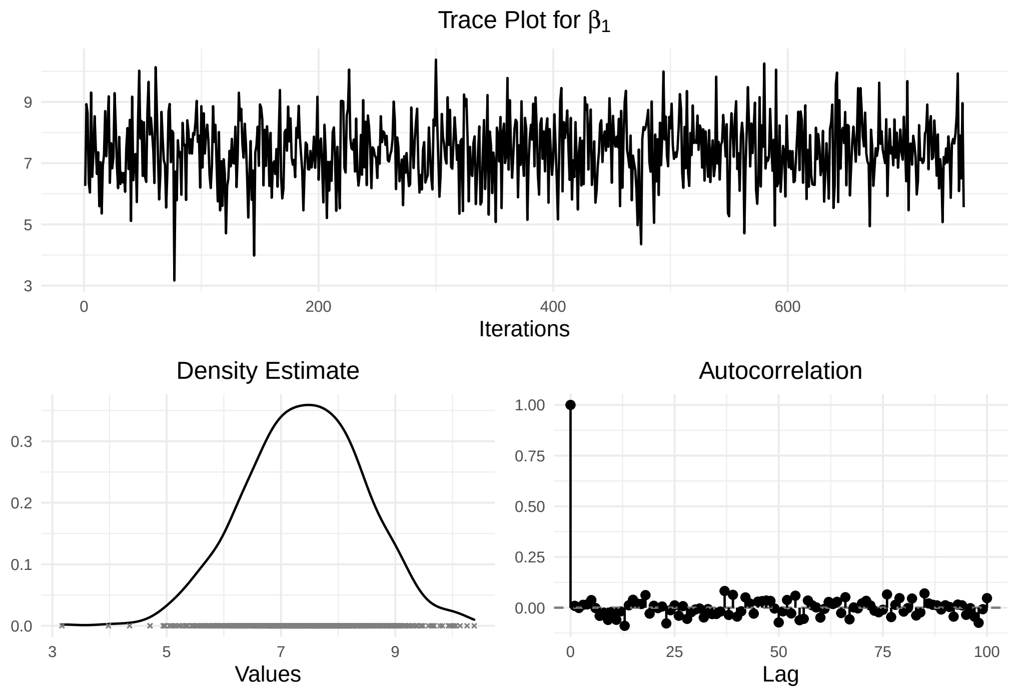

Figure 4: Diagnostic Plots for β_1 after thinning, data: abdom

```
## beta_1 -2.0015679 -2.0082464 -2.0015799 -1.995
## beta_2 -1.0044943 -1.0179012 -1.0044382 -0.991
## beta_3  1.0014607  0.9802516  1.0017075  1.022
## beta_4  2.0080638  2.0015876  2.0081264  2.014
##
## Scale coefficients (log link function):
##            Mean    2.5%     50%  97.5%
## gamma_0  0.9373 -0.7053  0.8277  3.074
## gamma_1 -1.1471 -1.5038 -1.1357 -0.862
## gamma_2  0.9325  0.7238  0.9305  1.176
##
## Residual degrees of freedom: 42
## Log-likelihood: 32.28
## AIC: -48.57
## BIC: -33.27
```

One downside of this approach is that the displayed values are not saved anywhere by default and, thus, cannot be immediately accessed. This issue is solved by the more informative `summary_complete()` function, which conveniently saves all relevant quantities in a data frame, the central object for data analysis in `R`.

In the following section we focus on the `toy_data` model, since there the true coefficient values are known, and use the popular `dplyr` package for further data frame manipulations. We are primarily interested in Posterior Mean estimates for all coefficients, such that we only select a subset of the output columns and add a new column containing the true data generating values:

```
library(dplyr)

true_beta <- c(0, -2, -1, 1, 2)
true_gamma <- c(0, -1, 1)
```

```
summary_complete(samples = toy_fit) %>%
  filter(stringr::str_detect(Parameter, pattern = "beta|gamma")) %>%
  mutate(Truth = c(true_beta, true_gamma)) %>%
  select(Parameter, `Posterior Mean`, Truth, `Standard Deviation`)
## # A tibble: 8 x 4
##   Parameter `Posterior Mean` Truth `Standard Deviation`
##   <chr>                <dbl> <dbl>                <dbl>
## 1 beta_0            0.000316     0              0.0769
## 2 beta_1           -2.00        -2              0.00339
## 3 beta_2           -1.00        -1              0.00676
## 4 beta_3            1.00         1              0.0104
## 5 beta_4            2.01         2              0.00315
## 6 gamma_0           0.937        0              0.967
## 7 gamma_1          -1.15        -1              0.163
## 8 gamma_2           0.933        1              0.109
```

Except for γ_0, all coefficients are estimated very accurately with a slightly larger deviation from the true values for the remaining $\boldsymbol{\gamma}$ vector, which is sampled by the Metropolis-Hastings algorithm, compared to the $\boldsymbol{\beta}$ vector, which is drawn directly from a multivariate normal distribution.

Similar to the plotting functions introduced in section 2.1.2, the `summary_complete()` function is very robust with respect to its data input: Besides the whole model object as in the example above, just the `toy_fit$mcmc_ridge` list entry or even simply the sampling matrices `toy_fit$mcmc_ridge$sampling_matrices` are valid inputs, all leading to the same output.

Further, the function has the optional `include_plot` argument. If set to `TRUE`, one additional `Plot` column is added to the data frame, which leverages the `diagnostic_plots()` function with some default settings and, thus, contains diagnostic plot objects for all coefficients. This option comes in handy in an interactive workflow: Particularly interesting findings from the `summary_complete()` output can be quickly extracted from the data frame without interrupting the current thought process by using the `diagnostic_plots()` function separately.

In the example above, the `gamma_0` row shows a large posterior variance in addition to the large deviation of the Posterior Mean estimate from the true value. In order to investigate if the chain suffers from a lack of convergence or a significant autocorrelation, we could create the desired (yet here omitted) output with the following simple command:

```
summary_complete(toy_fit, include_plot = TRUE) %>%
  filter(Parameter == "gamma_0") %>%
  pull(Plot)
```

This introductory tutorial is not intended to validate the sampler's performance nor to interpret the obtained coefficient estimates, but rather to provide an overview of the various applications, where the `asp21bridge` package turns out to be useful. A more in depth analysis of the `mcmc_ridge()` results, also in comparison to alternatives like the `lmls()` and `mcmc()` functions of the underlying `lmls` package, can be found in chapter 3 after explaining some of the internal implementations in section 2.2.

2.2 Implementation of the Markov Chain Monte Carlo Sampler

Of all components that the `asp21bridge` package is built upon, the implementation of the `mcmc_ridge()` function is of special interest from a statistical perspective. Thus, section 2.2.1 explains both the overall structure of the `mcmc_ridge()` function and the Gibbs Sampler that is responsible for inducing the Ridge penalty. Section 2.2.2 is dedicated to the integrated Metropolis-Hastings step, which is used for sampling $\boldsymbol{\gamma}$ and, if desired, $\boldsymbol{\beta}$ as well.

2.2.1 Iterative Parameter Sampling

The `mcmc_ridge()` function implements a Markov chain Monte Carlo (MCMC) sampler that iteratively simulates from the full conditional distribution of each parameter of interest. Thus, the resulting values can be interpreted as samples from the joint Posterior Distribution (as described in chapter 1), which in this case cannot be obtained analytically.

Overall Structure:

The function signature shows all mandatory as well as optional input parameters and their default values:

```
mcmc_ridge(m, X, Z, y, num_sim = 1000,
  beta_start = 1, gamma_start = 1, tau_start = 1, xi_start = 1,
  a_tau = 100, b_tau = 50, a_xi = 2, b_xi = 200,
  prop_var_scale = 3, mh_location = FALSE, prop_var_loc = 200
)
```

Many of the inputs have already been introduced in section 2.1.1. Explanations of each input can be found in the documentation, e.g. by calling `?mcmc_ridge()` after loading the `asp21bridge` package.

The `mcmc_ridge()` function is structured in a modular manner; it contains various internal helper functions collected in the `mcmc_ridge_helpers` file in order to maintain a concise and readable code structure.

Valid inputs can be provided by a `lmls` model object or separate design matrices $\mathbf{X}$ and $\mathbf{Z}$ and observation vector $\mathbf{y}$. In case both are given, the function prioritizes the additional manually specified inputs, such that for instance a single covariate can be added for the Bayesian sampler while using all remaining components from the fitted `lmls` model.

The `validate_input()` function handles these aspects and extracts all input parameters that are required for the sampling process. Further, it checks for missing input data and improper dimensions of the input parameters, in which case it returns an informative error message with one example given in section 2.3.2. Next, the `includes_intercept()` function checks both design matrices for intercept columns, while the `add_intercept()` function adds those for the sampling procedure, if necessary.

Now, that the raw input data is processed into a modified and convenient form, the Gibbs Sampler starts to iteratively simulate coefficient values. Since this part lies right at the heart of the `mcmc_ridge()` function, it is explained in more detail in the following subsection.

Whereas the simulation process itself is captured by a unified framework independent of the *input* type, the *output* type of the `mcmc_ridge()` function does depend on this distinction. If a `lmls` model is given as input, the sampling results are added to the existing model object in the new list entry `mcmc_ridge`. In this case, the sampling matrices for $\boldsymbol{\beta}$ and $\boldsymbol{\gamma}$ are assigned the names `location` and `scale`, respectively, which plays nicely into the existing code base of the `lmls` package and in particular for adjusting the `summary()` output to the `mcmc_ridge()` estimates.

In contrast, the output is structured in a list with the original names, if the inputs are specified manually without an existing model object. This conditional output type based upon the common sampling results is generated by the `create_output()` helper function.

Gibbs Sampler:

This paragraph explains the iterative Gibbs updates of the `mcmc_ridge()` function in a stepwise fashion. The specific form of the updates are based on the conditional distributions, derived in section 1.3. The following code snippets intend to merely emphasize the most important steps from a conceptual perspective and are therefore not syntactically exact. If the specific components of each step are of interest, they can of course be examined in the `mcmc_ridge()` source code.

1. Initialize matrices for collecting all simulations with the `init_sampling_matrix()` helper function. Start a counter `acc_count_scale` for calculating the acceptance rate of the Metropolis-Hastings step for $\boldsymbol{\gamma}$.

2. Repeat steps 3 - 5 for $i = 2, \dots,$ `num_sim`:
3. Update $\boldsymbol{\beta}^{(i)}$ by sampling from a closed form multivariate normal distribution (see section 1.3.3), using values from the previous iteration for all remaining coefficients:

   ```
   beta_samples[i, ] <- rmvnorm(1, mu = beta_mean, chol_sig_inv = chol(beta_var_inv))
   ```

   This step can also be performed via an additional Metropolis-Hastings step analogous to sampling $\boldsymbol{\gamma}$ below.
4. Sample $\boldsymbol{\gamma}^{(i)}$ via an integrated Metropolis-Hastings step discussed in section 2.2.2 with the inputs $\boldsymbol{\beta}^{(i)}$, $\boldsymbol{\gamma}^{(i-1)}$, $(\tau^2)^{(i-1)}$ and $(\xi^2)^{(i-1)}$. Update the value of `acc_count_scale` if the proposed value for $\boldsymbol{\gamma}$ was accepted:

   ```
   gamma_list <- mh_gamma(
     beta_samples[i, ], gamma_samples[i - 1, ], xi_samples[i - 1, ], ...
   )
   gamma_samples[i, ] <- gamma_list$gamma
   acc_count_scale <- acc_count_scale + gamma_list$accepted
   ```

5. Update the values for τ^2 and ξ^2 with samples from Inverse-Gamma distributions (refer to sections 1.3.1 and 1.3.2), using values $\boldsymbol{\beta}^{(i)}$ and $\boldsymbol{\gamma}^{(i)}$ from the current iteration:

   ```
   tau_samples[i, ] <- invgamma::rinvgamma(...)
   xi_samples[i, ] <- invgamma::rinvgamma(...)
   ```

2.2.2 Metropolis Hastings Step

As the Full Conditional of $\boldsymbol{\gamma}$ cannot be assigned to a known closed form distribution (see section 1.3.4), the MCMC update for $\boldsymbol{\gamma}$ cannot be sampled directly, but requires an Metropolis Hastings step.
This step is implemented by the `mh_gamma()` function.

Recall from chapter 1 that the Full Conditional of $\boldsymbol{\gamma}$ is given by

$$f(\boldsymbol{\gamma} \mid \cdot) \propto \exp\left(-\frac{1}{2} \cdot \left[\frac{1}{\xi^2}\tilde{\boldsymbol{\gamma}}^T\tilde{\boldsymbol{\gamma}} + 2 \cdot \mathbf{1}_n^T\mathbf{Z}\boldsymbol{\gamma} + \sum_{i=1}^{n}\left(\frac{y_i - \mathbf{x}_i^T\boldsymbol{\beta}}{\exp\left(\mathbf{z}_i^T\boldsymbol{\gamma}\right)}\right)^2\right]\right).$$

For numerical reasons, we used the Log - Full Conditionals for the Metropolis Hastings implementation. Since some parts of the expression above are used for sampling $\boldsymbol{\beta}$ as well, we computed the relevant terms only once in the `mcmc_ridge()` function and handed them as input to `mh_gamma()`. Similarly, the current state of the Markov Chains for $\boldsymbol{\beta}$ and $\boldsymbol{\gamma}$, required for evaluating the Full Conditionals, are provided by the `mcmc_ridge()` function in each iteration.

The implementation of the Metropolis-Hastings algorithm can be summarized by the following steps:

1. Compute the covariance matrix for the proposal distribution

   ```
   Sigma <- diag(1 / colMeans(Z^2)) * prop_var_scale / (n * length(gamma))
   ```

   and sample a proposed value for $\boldsymbol{\gamma}$ as

   $$\boldsymbol{\gamma}_{prop} \sim \mathcal{N}(\boldsymbol{\gamma}_{curr}, \boldsymbol{\Sigma})$$

   with $\boldsymbol{\gamma}_{curr}$ being the current state of $\boldsymbol{\gamma}$ in the Markov chain.

   Note, that the term `Z^2` squares all elements of `Z` elementwise and `n` denotes the length of the observation vector $\boldsymbol{y}$. Furthermore, the `colMeans()` function returns a numeric vector with the column means of a matrix.
2. Compute the Log - Full Conditional distribution of $\boldsymbol{\gamma}$ at the states $\boldsymbol{\gamma}_{curr}$ and $\boldsymbol{\gamma}_{prop}$.

3. Since we used a multivariate normal random walk proposal, the acceptance probability α for the proposed value can be computed by evaluation of the Full Conditionals only. Thus, we obtain

   ```
   log_acceptance_prob <- min(0, log_full_cond_proposal - log_full_cond_curr)
   ```

   The log - acceptance probability is restricted to be smaller or equal to zero to ensure that the acceptance probability does not exceed the value of 1. In order to compare the acceptance probability α with a standard uniform random variable $U \sim \mathcal{U}(0,1)$, the exponential function is used for transforming α to the original scale:

   ```
   acceptance_prob <- exp(log_acceptance_prob)
   ```

4. Finally, the magnitude of the acceptance probability α relative to the realized value of U controls the new state of the chain. In addition, we kept track of the overall acceptance *rate* across all proposals by introducing boolean variables `accepted` in each iteration.

   ```
   if (stats::runif(1) < acceptance_prob) {
     new_gamma <- gamma_proposal
     accepted <- TRUE
   } else {
     new_gamma <- gamma
     accepted <- FALSE
   }
   ```

The covariance matrix $\mathbf{\Sigma}$ of the proposal distribution in step 1 was chosen by simulating various combinations of the remaining model parameters to keep the acceptance rates in recommended ranges for a random walk proposal. The parameter `prop_var_scale` as a scaling factor of $\mathbf{\Sigma}$ has major impact on the acceptance rates and is included as an input parameter in the `mcmc_ridge()` function to provide manual control for the user.

The default value `prop_var_scale` $= 3$ was obtained from the same simulations and serves as a solid benchmark that works well in many scenarios. Larger values lead to larger jumps from the current to the proposed state, such that the acceptance probability decreases and the Markov Chain for $\boldsymbol{\gamma}$ might stay at the same point for a long time. By similar logic, smaller values of `prop_var_scale` lead to larger acceptance probabilities. Since the resulting steps will be small, the chain might converge to the Posterior Distribution very slowly and not explore the Posterior space sufficiently. Therefore both extremes will result in strongly correlated samples.

The Metropolis Hastings algorithm for $\boldsymbol{\beta}$, implemented by the `mh_beta()` function, works in a similar way but is of minor importance, since the $\boldsymbol{\beta}$ updates can also be sampled directly from a multivariate normal distribution.

2.3 Package Development

This section is dedicated to more niche aspects that come along with the package development process. Part 2.3.1 focuses on more formal components of the package, whereas some of the ideas behind the `asp21bridge` functions are discussed in section 2.3.2.

2.3.1 Code Coverage

Arguably the most important component of a package for new users is the **documentation**. All exported functions of the `asp21bridge` package as well as the `toy_data` data set are fully documented according to common standards for `R` packages.

In particular, we put effort into informative, yet not excessively long descriptions of the overall function and their parameters with highlighted default settings. The help pages are designed in a consistent manner across all functions. The `examples` section usually starts with the most basic applications signaling the function's *intent* and proceeds with more complex use cases that cover many of the function's optional arguments.

A more elaborate, publicly accessible tutorial similar to section 2.1 of this report can be found in the `README` file or at the front page of the `asp21bridge` GitLab website.

A second major aspect, that contributes to the quality of a package, are **unit tests**. Up to this point, a total of 251 unit tests have been written for the `lmls` and the `asp21bridge` packages combined. The number of implemented tests alone is, of course, not a meaningful metric, since the tests might be highly redundant and still capture only a small fraction of the entire package functionality. Thus, we prioritized both *width* and *depth* of the test coverage.

More specifically, unit tests are written for every single exported function with a larger focus on the more complex and more central ones such as the main `mcmc_ridge()` function.

Further, not only simple input checks (which are important nonetheless!), but also more esoteric edge cases and especially very common use cases of combining inputs and outputs of multiple `asp21bridge` functions are covered. Whenever discovering and fixing an unexpected error during the development phase, we implemented corresponding unit tests, such that this specific error will not reoccur in the future. Tests are continuously written and updated to ensure a stable and enjoyable user experience.

At a larger scope, the `R CMD CHECK`, or equivalently the `devtools::check()` command, produces 0 errors, 0 warnings and 0 messages. Besides working unit tests, this includes correctly specified `DESCRIPTION` and `NAMESPACE` files for all imported and exported functions as well as functioning examples in the documentation. The master branch of the `asp21bridge` project will maintain this standard in the foreseeable future; possible new features that could involve breaking changes will first be implemented on separate Git branches and at a later stage merged into the master branch after a thorough testing procedure.

2.3.2 Design Choices

Throughout the development process, we tried to adhere to some 'best practices' for general software development. These include several different aspects:

Default Settings:

For many `asp21bridge` functions, the user has to specify very few inputs manually since many input options are set to sensible default arguments. These inputs are chosen in a *neutral* way, such that the output aligns as best as possible with the user's expectation.

As an example, the default starting values for $\boldsymbol{\beta}$ and $\boldsymbol{\gamma}$ are the Maximum Likelihood estimates from the `lmls()` function, if a model object is provided as input to the `mcmc_ridge()` function. This serves as guidance for users with little prior knowledge about their model. It is worth noting, however, that there are multiple input parameters which *can* be set manually if desired, enabling fine control over the sampling procedure.

Moreover, the default values for the hyperparameters a_τ, b_τ, a_ξ and b_ξ are set according to the results of our conducted simulation studies in sections 3.5 and 3.6, which correspond to a conservative penalty effect and, thus, a moderate induced bias.

Input Flexibility / Defensive Programming:

As already mentioned in section 2.1, all exported functions are designed to be as flexible as possible with respect to their input arguments. Thus, many functions accept multiple data structures like (nested) lists, data frames, matrices or even simple vectors as data input and internally transforms the input into the desired format.

These transformations, however, are only performed as long as the user's *intention* is clear, e.g. providing the samples as input to a plotting functions regardless of the exact data structure, where the samples are stored. Whenever there is ambiguity or the input is simply not valid / incomplete, we have put effort into writing informative error messages.

To illustrate this last point, assume that we had forgotten to provide the second design matrix `Z` as input to our second application example from section 2.1 using the `abdom` data set. This might happen quite frequently, especially if the user is not particularly familiar with the structure of location-scale regression models:

```
y <- abdom$y
X <- as.matrix(abdom$x)

abdom_fit <- mcmc_ridge(
  y = y, X = X, beta_start = 1, gamma_start = 1, num_sim = 10000
)
## Error in validate_input(m = m, X = X, Z = Z, y = y, beta_start = beta_start, :
## At least either all model matrices (X, Z, y) and coefficients
## (beta_start, gamma_start) or a model object (m) must be given.
```

From the error message, it is immediately obvious which part of the input is missing.

One further quite neat feature is implemented for the generic `summary()` function. Since the available options for the `type` argument are specific to the `lmls` class and therefore a potential reason for confusion, the error message provides some guidance in case of spelling mistakes, suggesting the closest valid input option:

```
summary(toy_fit, type = "mcmc-ridge")
## Error: `type` must be one of "ml", "boot", "mcmc", or "mcmc_ridge".
## Did you mean "mcmc_ridge"?
```

Modularity:

The `lmls` package served as a fantastic example how to design and compose functions in a modular way, leading to independent applications of single components in various contexts, easier debugging and arguably better code transparency and readability.

The `asp21bridge` package similarly splits large functions into multiple pieces according to the well known premise, that functions should be doing one thing only, but one thing well. Therefore the `mcmc_ridge_helpers` file contains various helper functions for the main `mcmc_ridge()` function, which perform tasks like input validation or inclusion of the Metropolis-Hastings step for the γ vector as explained in section 2.2.1.

However, there is certainly a trade off to keep in mind: Since naming functions and variables is notoriously challenging, excessive modularity with poor naming choices can hinder code comprehension by inducing a wrong mental model of the function's task. Moreover, it can make sense to keep related functions close together in a physical sense and avoid spreading them across multiple files across the project.

Coding Style:

Before starting our work on different parts of the code base, we agreed to a consistent coding style. This includes using only lowercase letters and underscores in function and variable names and a limit of 80 characters per line. Here, we mostly conformed to the recommendations from the tidyverse style guide by Hadley Wickham. Moreover, we leveraged the styler package for aesthetically pleasing code formatting.

To extend consistent naming schemes even further, we chose the same argument names for all exported functions whenever possible. For instance, the data input for all functions that expect the MCMC simulations is called `samples` and all shared arguments of the 5 plotting functions are assigned the same name and the same functionality. Hence, it often suffices to be familiar with one function of a certain family, since that knowledge can be easily transferred to all related functions.

Finally, there are two design choices, that are very specific to the `asp21bridge` context:

Operation Chaining:

First, we aimed to allow for frictionless function composition via the popular `%>%` operator. As illustrated in section 2.1, this comes in particularly handy when including `burnin()` and `thinning()` operations in a larger pipeline without the need to define dummy variables for temporary objects:

```r
lmls(
  location = y ~ x1 + x2 + z1 + z2, scale = ~ z1 + z2,
  data = toy_data, light = FALSE
) %>%
  mcmc_ridge(num_sim = 1000) %>%
  purrr::pluck("mcmc_ridge", "sampling_matrices") %>%
  burnin(num_burn = 100) %>%
  thinning(freq = 5) %>%
  mult_plot(type = "both", free_scale = TRUE, latex = TRUE)
```

This workflow proved to be so useful, that we provided access to the pipe operator directly by loading the `asp21bridge` package, i.e. we reexported `%>%` from the `magrittr` package. Alternatively, one could have used the native `|>` pipe introduced in `R 4.1`. However, since the release of this `R` version was so recent, it might induce a stronger dependency than just relying on the `%>%` operator, which is ubiquitous at least in the `R` community that is related to the data science field.

There are two conditions that functions must fulfill to chain multiple operations:

- The first argument must be chosen uniformly for all (except the first) involved commands. In case of the `asp21bridge` package this corresponds to the `samples` argument at the first position.
- Functions that serve as intermediary steps inside a chain should return the same object type as their input. If a `lmls` model object is provided, the `mcmc_ridge()` function returns a modified (copy of the same) `lmls` model object. Even more flexible are the `burnin()` and `thinning()` functions: They can take a list of matrices, a single matrix or a numeric vector as input and always return the same provided input data structure.

Object-Oriented Programming:

Secondly, we thought about initializing our own S3 class when calling the `mcmc_ridge()` function, but finally decided against it for two reasons: We always considered the `asp21bridge` package as a strict extension to the `lmls` package. Thus, we did not want to make the underlying `lmls()` results less accessible after using `mcmc_ridge()`, i.e. generic functions like `coef()`, `plot()` or `print()` should in our mind create the same output before and after extending the original model.

One exception is the `summary()` function, as we have already mentioned. Here, we added one more option to the `type` argument, which has to be explicitly specified to access information from the `boot()` and `mcmc()` function of the `lmls` package as well. Apart from that, we could not see much additional value in implementing wrapper functions, which adapt e.g. the `print()` output to the penalized MCMC results, beyond the already existing tools that were introduced in section 2.1.

3 Simulation Studies

This chapter investigates the effect of manipulating some of the `mcmc_ridge()` inputs (and thereby sampling parameters) in a controlled environment, such that changes in the estimation outcome can be directly linked to respective changes in the model inputs. The simulation studies discussed in this chapter can be categorized into three groups:

1. Evaluating the **quality** and **robustness** of the sampler.

   Section 3.1 induces correlation into the underlying data, which can potentially be challenging for estimation. Just as in sections 3.3 and 3.4, the variation in the input parameters is therefore focused on the function argument `m` or alternatively the combination of `X`, `Z` and `y`. Section 3.2 is dedicated to manipulating the sample size through the input parameter `n`. Here, we are looking for a possible stabilization process with increasing size of the input data hinting at asymptotic/convergence properties.

2. Investigating the **penalty** effect.

   Sections 3.3 and 3.4 replicate real world estimation scenarios, where the `mcmc_ridge()` sampler is primarily chosen for the desired regularization effect. To emphasize the shrinkage effect on the coefficient estimates, the `mcmc_ridge()` model is compared to the `lmls()` and the `mcmc()` models from the `lmls` package, which do not leverage a penalty.

3. Exploring the effect of the **hyperparameters**.

   Sections 3.5 and 3.6 analyze the connection between the input parameters `a_tau`, `b_tau`, `a_xi` and `b_xi` of the Inverse-Gamma prior distributions of τ^2 and ξ^2 to the simulation results. The effect of hyperparameters in a hierarchical Bayesian model can be difficult to predict based on pure logical reasoning. Therefore simulations are a useful tool to either confirm prior assumptions or discover unexpected behaviour.

Since the resulting simulation studies serve such diverse purposes (e.g. diagnostic vs. exploratory), they demand for different approaches in the simulation settings, the implementation as well as the analysis and presentation of the results. For that reason, we decided against forcing all of the following sections into one common rigid framework. Instead, each section individually motivates, explains and interprets the methods chosen for its particular use case.

In order to keep the analysis in this chapter succinct, there will be almost no code included. It it worth noting though that all `R Scripts` used for the simulations are contained in the `simulation-studies` folder of the `asp21bridge` package. Thus, each figure as well as all numerical results are fully reproducible and can be repeated and extended by the reader.

3.1 Correlated Predictor Variables

Up to this point, we have often illustrated the usage and results of the `mcmc_ridge()` sampler with simulated data from the built-in `toy_data` set. As stated in part 2.1, each regressor variable is independently sampled from a normal distribution and the outcome variable is simulated based on the correctly specified location-scale regression model introduced in chapter 1. All these conditions lead to an excellent performance of the `mcmc_ridge()` sampler, but might arguably not represent the most challenging task.

Hence, many of the simulation studies contained in this chapter intend to imitate data configurations that might be closer to data found in the real world. We start by inducing correlation among the predictor variables. Further, the `mcmc_ridge()` performance is compared to the Maximum Likelihood based `lmls()` estimates and the unpenalized Bayesian `mcmc()` sampler.

Simulation Setting

Briefly stated, we simulate covariate data from a three dimensional multivariate normal distribution with three different values for the pairwise correlation ρ. Then, we fit the three models `mcmc_ridge()`, `mcmc()` and `lmls()`, where the number of simulations was set to 10.000 for the Bayesian Samplers.

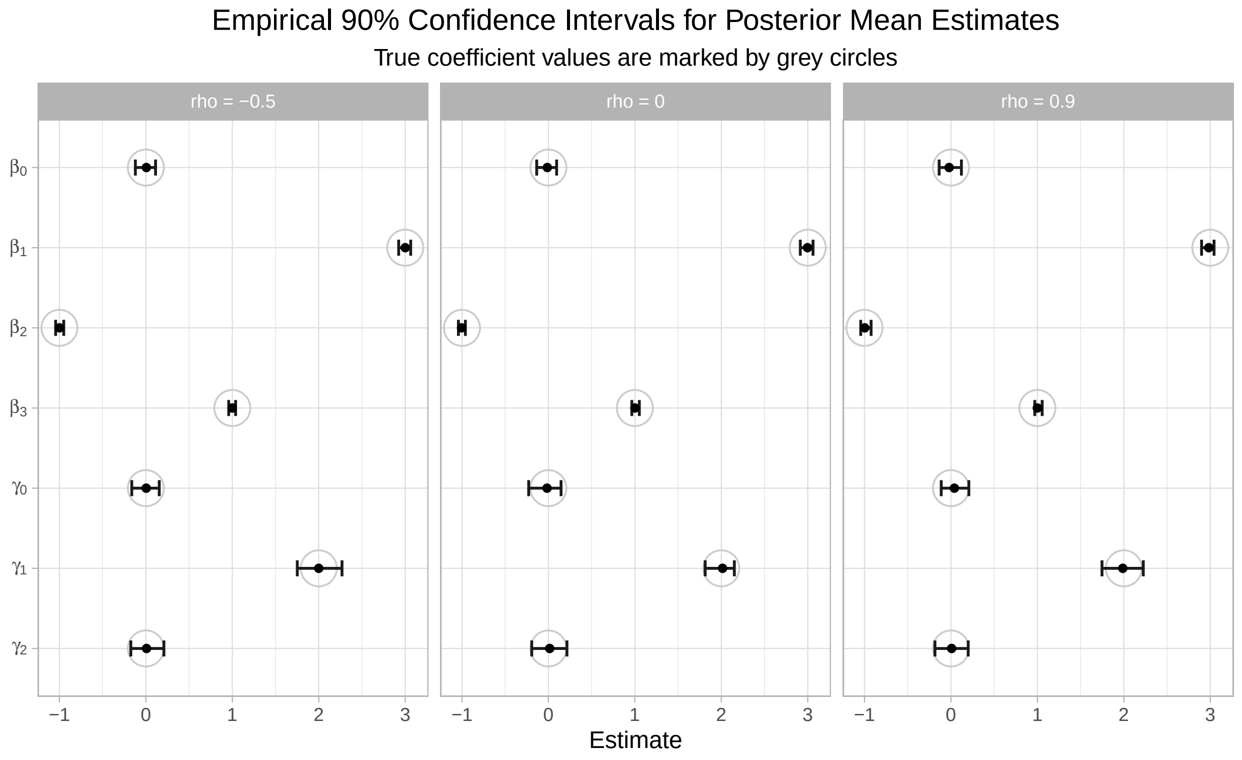

Figure 5: Comparison of Correlation Structures - 50 Simulation Cycles

This simulation study is intended to evaluate the *accuracy* of the three models. Therefore, the same (standardized) values of the covariates are used for both the generation of the outcome variable $\mathbf{y}$ and the model fit. This entails that the true coefficient values are known and can be compared to the resulting estimates of all three and in particular the `mcmc_ridge()` model. The exact simulation setting is described in full detail in the Appendix.

Moreover, we compared the performance of the classical `mcmc_ridge()` implementation, which draws $\boldsymbol{\beta}$ from the closed form full conditional (multivariate normal) distribution, with an alternative sampling process that uses a Metropolis-Hastings approach for the location parameter $\boldsymbol{\beta}$ as well as the scale parameter $\boldsymbol{\gamma}$.

Simulation Results

All three models perform equally well for each correlation structure: The estimates for $\boldsymbol{\beta}$ are almost exactly equal to the true values, whereas there are slight deviations for the $\boldsymbol{\gamma}$ vector. These errors, however, are consistent in all three sampling scenarios and, thus, do not depend on the strength nor the direction of the correlation between the covariates. Due to this similarity between the three models, the remaining analysis of this simulation study is dedicated to the `mcmc_ridge()` model only.

There is one additional aspect to be noted: The performance of the `mcmc_ridge()` function is considerably worse, when $\boldsymbol{\beta}$ is sampled by means of the Metropolis-Hastings algorithm. Although the acceptance rates of the proposed values are in acceptable ranges for a random walk proposal (between 35% and 40%), the chains are strongly correlated and did not fully converge even after 10.000 iterations. For that reason, we limit the Metropolis-Hastings sampling process for $\boldsymbol{\beta}$ to this one example and will focus on the classical `mcmc_ridge()` implementation in the remaining parts of chapter 3.

The corresponding graphical display for the comparison between the three models and between the two sampling procedures for $\boldsymbol{\beta}$ is omitted here for brevity reasons, since the findings are not that surprising. The code as well as all plots can be found and fully reproduced in the `regressor-correlation.R` file located inside of the `simulation-studies` folder of the project directory.

In order to make any conclusions about bias and variance, the above procedure is repeated 50 times with 1000 simulations each. The black points in Figure 5 represent the mean of these 50 Posterior Mean estimates from the penalized Ridge sampler. For a better visual comparison, the true values for each coefficient are indicated by grey circles. Since we cannot rely on distributional theory for the standard errors, the variability of the estimates is displayed by nonparametric 'confidence' intervals, which are simply given by the range from the empirical 0.05 quantile to the 0.95 quantile of the 50 estimated values.

These intervals are very narrow and centered around the true value in all cases with slightly larger variability for the $\boldsymbol{\gamma}$ vector, which is sampled via the Metropolis-Hastings algorithm. Overall, these results emphasize the sampler's stability in presence of correlated covariates. The bias towards zero that is typical for the Ridge penalty cannot be observed in this case; all coefficients are estimated with both high accuracy and high precision.

It is worth noting that the results are partially affected by the standardization of the covariates before fitting the models: The standardization generally leads to a stabilized fit for all three of them. Compared to the performance for unstandardized data (as used for the second report) the variability of the Posterior Mean estimates of β_0 in particular is significantly reduced.

3.2 Sample Size

This simulation study analyzes the effect of the sample size `n` on the Posterior Means for the coefficient vectors $\boldsymbol{\beta}$ and $\boldsymbol{\gamma}$. There are two main goals of this simulation study: On the one hand, we want to investigate whether the posterior means of large samples are closer to the true values than the posterior means of small samples. On the other hand, we want to analyze whether the `mcmc_ridge()` penalty affects the location of the posterior means.

Simulation Setting

For the analysis purposes of this simulation study, the sample size `n` varies between 30 and 500. There are in total six different scenarios.

In the first step, the n-dimensional covariates $\mathbf{x}_1$, $\mathbf{x}_2$, $\mathbf{z}_1$ and $\mathbf{z}_2$ are all sampled from a normal distribution with variance 1. After that, all of these four vectors are standardized to have zero mean and unit variance. Then the n-dimensional vector $\mathbf{y}$ is sampled based on a linear model with an error term drawn from a standard normal distribution. The `mcmc_ridge()` function is applied afterwards and provides estimates for the Posterior Means of $\boldsymbol{\beta}$ and $\boldsymbol{\gamma}$. The underlying true coefficients as well as all details of the simulation setting can be found in the appendix.

To make the results more reliable, above procedure is repeated 100 times for each of the six different sample sizes. The performance of the estimates is then measured for each of the $\boldsymbol{\beta}$ and $\boldsymbol{\gamma}$ coefficients by the mean values of the Posterior Means and the *MAE* over the 100 repetitions.

Simulation Results

The means of the Posterior Mean estimates are displayed in the left plot of Figure 6. None of the parameters seems to be biased even for a sample size of `n` $= 30$.

After getting an impression about empirical biases of the coefficients, we now focus on the variability of the posterior means of the coefficients, which are measured by the *MAE* based on the results of the 100 repetitions.

The right plot of Figure 6 points out that the posterior means of all six coefficients have mean absolute errors of quite the same magnitude. In addition, for increasing sample sizes, the *MAE* of the Posterior Means converges to zero for all coefficients.

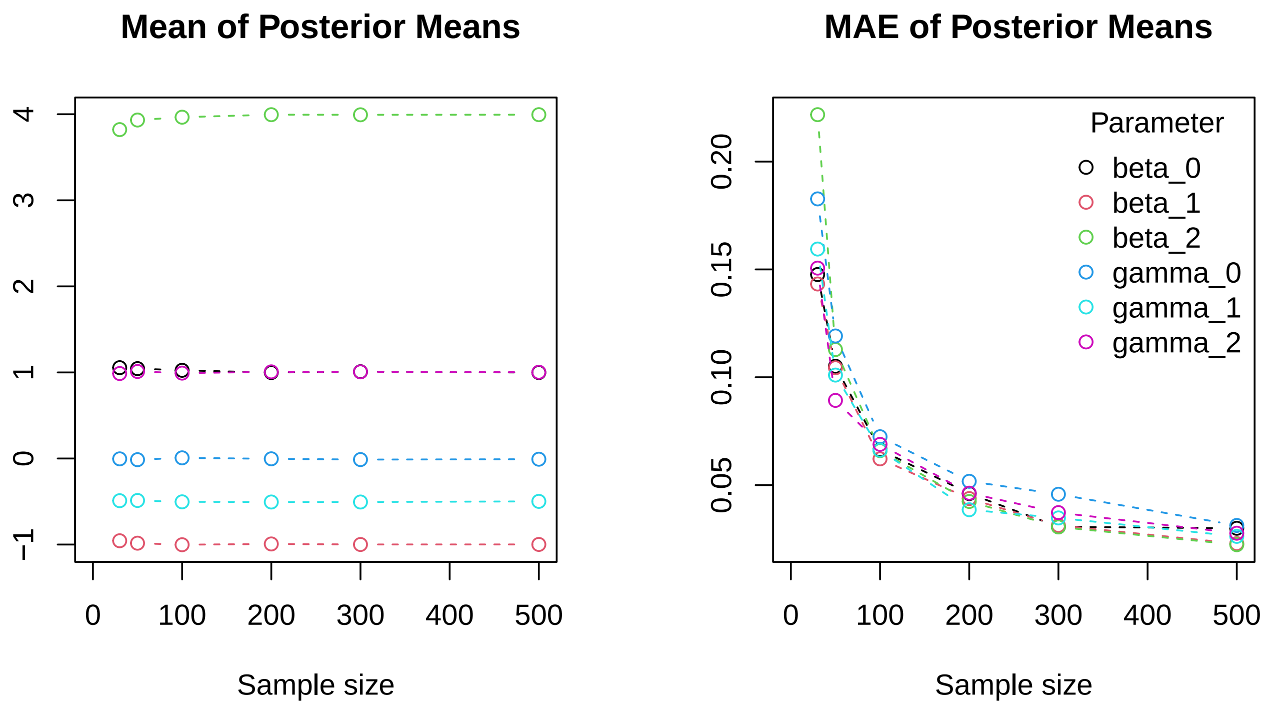

Figure 6: Mean and MAE of 100 Posterior Mean Estimates

3.3 Redundant Covariates

Similar to the simulation study from section 3.1, this part investigates the sampler's behaviour in presence of strong *pairwise* correlation among the covariates. However, in contrast to all other simulation studies, we add additional *redundant* regressors to the model, that were not used for generating the outcome variable $\mathbf{y}$.

The statistical theory of (Ridge) penalization suggests, that the coefficient estimates from the `mcmc_ridge()` function are affected by the shrinkage effect induced by the coefficient's prior distributions in presence of a high number of correlated covariates. Thus, the magnitude of the estimated $\boldsymbol{\beta}$ vector should be smaller compared to models without a regularization component.

Simulation Setting

The model contains 10 pairs of covariates drawn from a multivariate normal distribution with correlation $\rho = 0.9$. For each pair, the first covariate contributes to generating the outcome variable $\mathbf{y}$ with a true coefficient value of 1. The second covariate has no impact on $\mathbf{y}$, but is included nonetheless in the model fitting stage.

Since the purpose of this study is to demonstrate the Ridge regularization and not primarily the accuracy of the models, we used the covariates on their original scale for generating $\mathbf{y}$ and standardized them afterwards before fitting the models. Thus, the true coefficient values cannot be immediately compared to the estimates without further transformations, such that conclusions about the model *quality* are not valid. This scenario resembles a real world application, where the data generating process is latent and estimates are obtained based on purely observational data. The exact design of the simulation study can again be found in the Appendix.

Simulation Results

The following analysis is exclusively focused on the location parameter $\boldsymbol{\beta}$. As mentioned before, there are two aspects of interest we wish to investigate further:

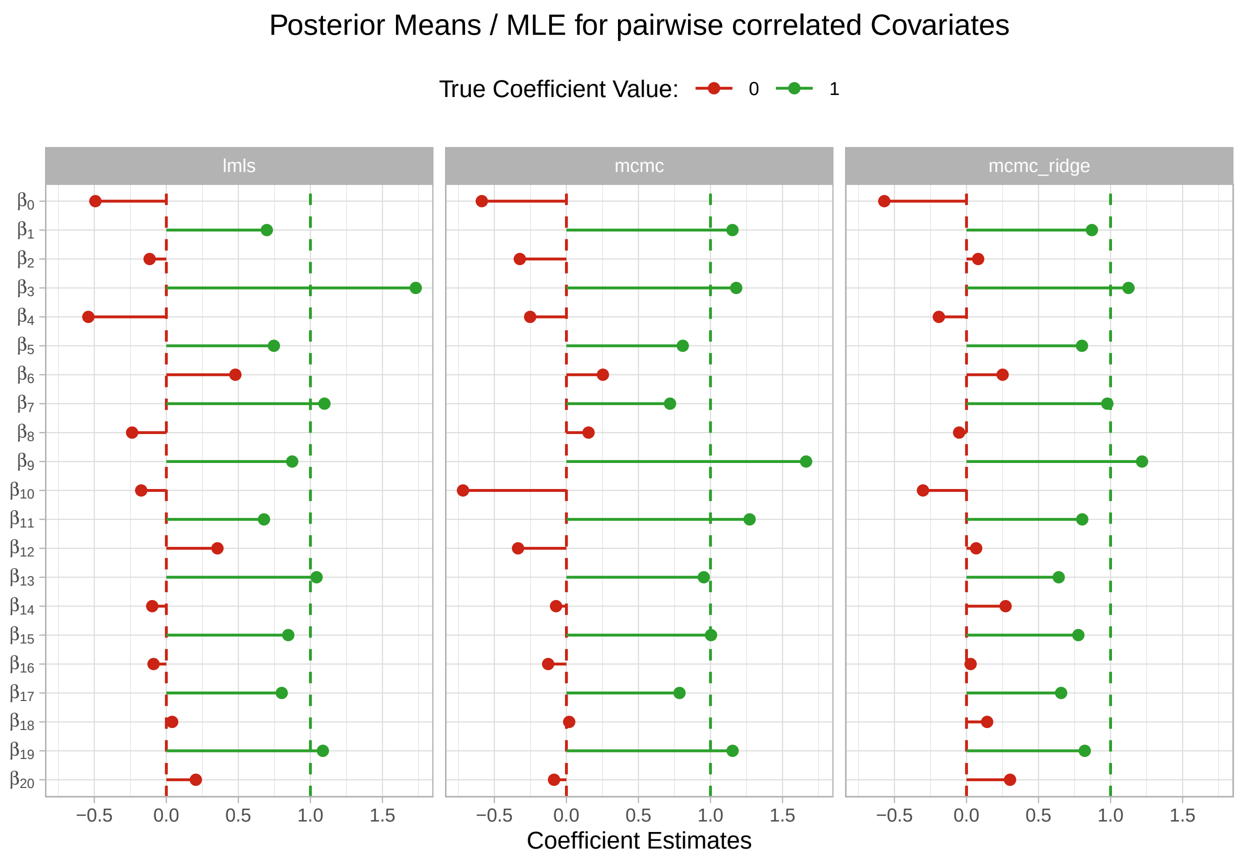

Figure 7: Shrinkage Effect of Ridge Penalty in Presence of Redundant Covariates

1. Can each model differentiate between the relevant and the redundant covariate for each pair of regressor and, thus, isolate the true effect on $\mathbf{y}$?

2. Due to the high number of 20 covariates compared to the low sample size of 50 observations and the presence of correlated and partially redundant regressors, the setup of this study is prone to overfitting. Can we therefore observe the hypothesized shrinkage effect on the coefficient estimates for the `mcmc_ridge()` model compared to the models without penalty?

Figure 7 illustrates the results obtained by fitting all three models.

We start with answering the first question: For each of the three models and each coefficient pair the absolute value of the coefficient corresponding to the relevant regressor is larger than the magnitude of the paired quantity. Thus all three models always contribute the major impact on the outcome variable to the correct covariate.

Further, the differences between estimations for those coefficients with true value 1 and those with true value 0 do not differ in an *obvious* manner across the models. Since the variance of the estimates seems to be largest for the `lmls()` model and smallest for the `mcmc_ridge()` model, the ratio (instead of the difference) tends to be largest for the regularized model.

The reduced variance of the `mcmc_ridge()` model can be explained by the Ridge penalty: Since all estimates are shrunken towards zero, fewer ‘outlier’ estimates that are far away from zero will be observed. This property is beneficial for the red points, since the estimates clutter around their true value in this case. However, the coefficients corresponding to the green points are penalized as well such that a bias towards zero is induced, which is not present for the two unpenalized models. Hence, the hypothesized shrinkage effect can indeed be observed.

Table 1: $\|\boldsymbol{\beta}\|^2$ excluding β_0

	True Value = 0	True Value = 1
lmls	1.05	10.09
mcmc	1.26	12.13
mcmc_ridge	0.72	7.87

It is insightful to compare the observed effect from the Ridge penalty to the expected effect of the LASSO penalty. The LASSO penalty encourages *sparse* solutions: each coefficient is assigned the same weight such that shrinkage of large estimates is not preferred over shrinkage of small estimates. Thus, small estimates are often set to exactly zero, while large estimates, although reduced, can still be significantly different from zero.

In contrast, the Ridge penalty generally does not induce sparsity. In fact, quite the opposite can be the case: Small coefficient estimates such as β_{14} in Figure 7 are sometimes *increased* in magnitude compared to the unpenalized models. Large estimates such as β_4 or β_{12}, however, are reduced in size quite heavily. The Ridge penalty tries to put all coefficients on the *same scale* and, thus, prioritizes shrinkage of large estimates compared to small ones! Note also, that the intercept term is typically excluded from the penalization, which is the reason why the estimates of β_0 are of approximately equal size for all three models.

In order to quantify the observed shrinkage effect from Figure 7 numerically, we calculate the sum of squared coefficient estimates, i.e. the squared Euclidean norm $\|\boldsymbol{\beta}\|^2$, for each model. Moreover, these magnitudes are compared separately for all coefficients corresponding to relevant regressors (those with an odd subscript) and those corresponding to redundant regressors (even subscript). Table 1 summarizes the results.

The most interesting and reassuring finding is obtained from the `mcmc_ridge()` vector norms: In agreement with the visual illustration, the Ridge regularization effect can be numerically detected for all coefficient estimates, independent of the true value. While this property induces a bias that is not desired for the second column (underestimating the true values on average), it does indeed prevent overfitting by shrinking the coefficients corresponding to the redundant covariates, which is indicated by the lowest value in the first column.

In that sense, the Ridge penalty increases the model *robustness* in presence of a high number of (correlated) regressors and often leads to an overall lower Mean Squared Error due to the reduction in variance.

3.4 Challenging the Model Assumptions

This simulation study is structured in a very similar way to the study considered in section 3.1. Instead of varying the correlation structure among the regressors in the underlying data set, both the regressors and the outcome variable $\mathbf{y}$ are sampled from distributions that are more challenging for estimation than the normal distribution.

Simulation Setting

More specifically, the covariates of the design matrices $\mathbf{X}$ and $\mathbf{Z}$ are generated by a mix of distributions, including the Normal, Exponential, Uniform, Bernoulli and t - distribution. Given these values, the outcome variable $\mathbf{y}$ is drawn from three different distributions (Normal, t and Uniform) corresponding to three separate scenarios, which are compared in the following analysis. As usual, the full simulation design is stated in the Appendix.

Note that the `lmls()`, `mcmc()` and `mcmc_ridge()` models are built upon the assumption of a Gaussian distribution for $\mathbf{y}$. Hence, all three estimation procedures are expected to perform well under the first outcome specification, which they were designed for. The remaining two cases present mild (in case of the t distribution) and moderately strong (Uniform distribution) violations of this model assumption.

It is worth noting that the results vary heavily across models and outcome distributions if the models are fit

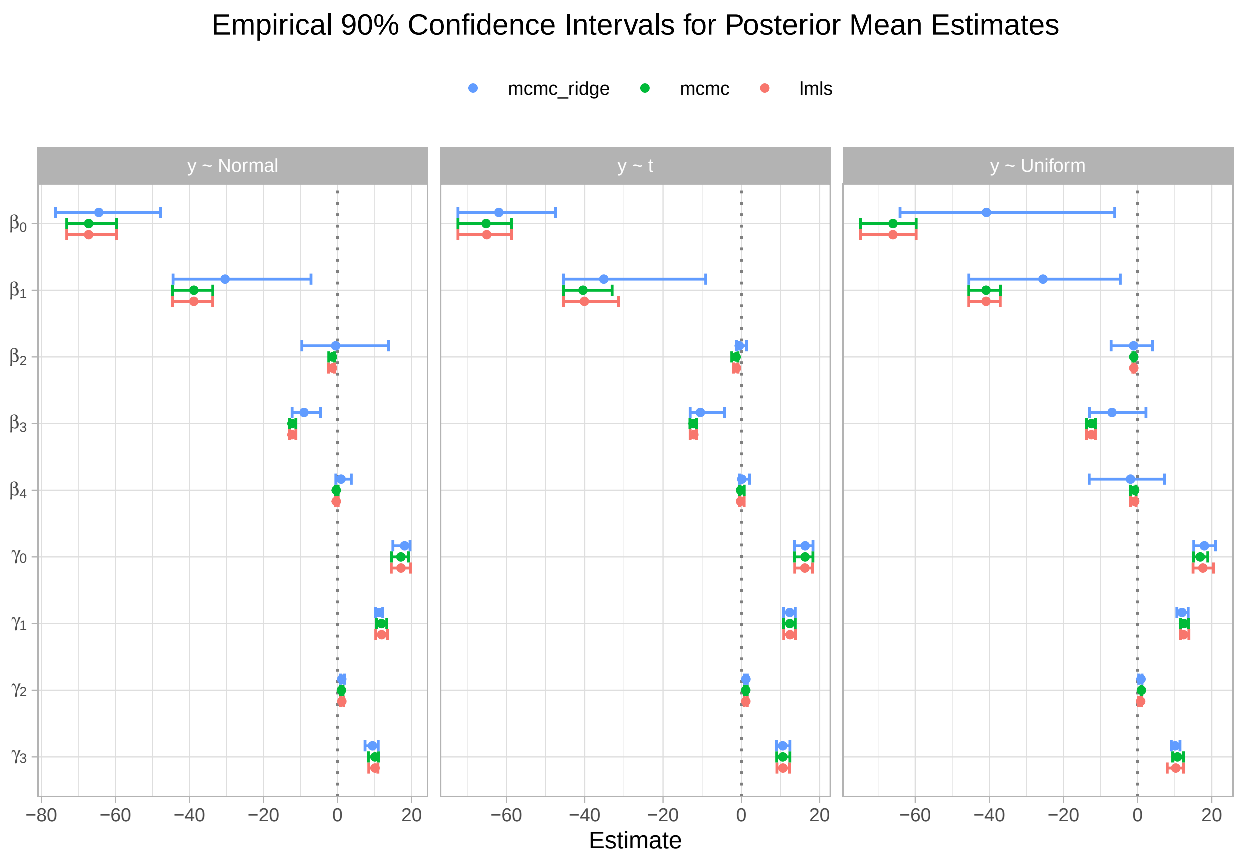

Figure 8: Comparison of different Outcome Distributions - 50 Simulation Cycles

with covariates on their original scale, as seen in our Second Report. For this simulation study standardization makes a huge difference, such that coefficient estimates are a lot more consistent for all 9 scenarios, when the models are fit with standardized regressor columns.

Quite surprisingly, the results also considerably depend on the data generation of the outcome $\mathbf{y}$: When covariates are already standardized before contributing to $\mathbf{y}$, MLE and Posterior Mean Estimates are almost exact with very low variability. When the features are standardized after the data generation, the estimates are still accurate (with respect to the scale transformed original true values), but indicate a larger variance. For that reason, we decided to focus this study on the penalization effect rather than precision metrics and therefore will not compare estimates to the (transformed) true coefficient values.

Simulation Results

Figure 8 shows the results for each of the 9 scenarios over 50 simulation cycles. The points indicate the means of the 50 MLE / Posterior Mean estimates, the 'confidence' bounds are given by the empirical 5% and 95% quantiles of those 50 values and, thus, not based on distributional assumptions. As a brief technical note, although the second facet is labeled by $y \sim t$, it is formally sampled from an affine transformation of a t-distributed random variable, which does not follow an exact t distribution.

Overall, the $\boldsymbol{\gamma}$ coefficients seem to be much less affected by the penalty for the given choice of hyperparameters that is used by the `mcmc_ridge()` model. The effect of the hyperparameter values on the Ridge Penalty is discussed in section 3.6, such that we do not investigate this observation further here. Instead we take a closer look at the $\boldsymbol{\beta}$ estimates.

We first examine the results for values close to the dotted zero line: Compared to the `lmls()` and `mcmc()` models without penalty, there is a slight shrinkage effect for β_3 estimates observable in each of the facets.

Table 2: $\|\boldsymbol{\beta}\|^2 + \|\boldsymbol{\gamma}\|^2$ excluding β_0 and γ_0

	Normal	t	Uniform
lmls	1906	2024	2088
mcmc	1903	2054	2100
mcmc_ridge	1222	1609	950

Further, the increased variability for some of the small values such as β_2 in the left plot or β_4 in the right plot is interesting.

One potential explanation was already introduced in section 3.3: Rather than shrinking small coefficients to zero exactly, the Ridge penalty tries to put all parameters (except the intercept) on equal scale, while simultaneously reducing the sum of squared coefficient estimates. Thus, taking β_2 as an example, the original small estimate of β_2 might increase in simulation cycles, where many of the remaining coefficients are large in absolute value in order to close the gap between them. In contrast, β_2 might stay approximately equal in those cases, when other estimates happen to be small as well for this iteration and all estimates shrink synchronously.

The penalty effect is most obvious for the largest estimate, in this case β_1. For all three outcome distributions, the estimates of the `mcmc_ridge()` function are consistently closest to zero compared to the unpenalized models. The size of the penalty in each simulation cycle can vary quite heavily, as indicated by the empirical confidence bounds, potentially for the reasons stated above.

In addition to the visual illustration, the penalty effect can be numerically established by computing the (squared) Euclidean Norm of the coefficient vectors $\boldsymbol{\beta}$ and $\boldsymbol{\gamma}$, since this is the quantity involved in Ridge penalization. Excluding the intercept estimates, the calculated values for this simulation study are shown in Table 2. Confirming the graphical impression, the values of the unpenalized models are almost identical, independent of the outcome distribution, whereas the magnitude of the `mcmc_ridge()` coefficients is significantly reduced in all cases.

Although there are some differences among the three plot facets in Figure 8, the observed patterns appear too random to draw meaningful conclusions about the connection between the strength of violating the distributional assumptions and the regularization. The reduced estimate of the intercept parameter in the right panel is not caused by the Ridge penalty, in which case the observed effect would have been much stronger in all plot facets, but rather by the generally increased estimation variance of the intercept terms in the `mcmc_ridge()` framework. The large overall magnitude of the intercept term is induced by mean-centering all covariates between data generation and model fitting.

Technical Aspects

As outlined in the previous paragraph, a total of $50 \cdot 9 = 450$ models were fitted to analyze the differences across models. In order to speed up the involved computations of this specific and some of the other simulation studies in this chapter, we used the *parallel computing* capabilities of `R`.

There are many options from various packages to choose from. We decided to use the furrr package, which is built on top of the `future` package specialized on parallel processing. As the name suggests, `furrr` provides a convenient way to use many functions from the popular `purrr` package, while using multiple cores at the same time. This *functional programming* based approach (similar to the `apply()` family in 'base `R`') is particularly well suited for simulation studies and provides some structural as well as minor performance advantages compared to the classical `for`-loop approach.

The following (slightly modified) code snippet provides a brief insight into the implementation:

```r
plan(multisession, workers = 8)
```

```r
full_results <- tibble(id = 1:50) %>%
  mutate(samples = future_map(
    .x = id,
    .f = ~ show_results(n = 50, num_sim = 1000),
    .options = furrr_options(seed = 1)
  ))
```

The `plan()` function borrowed from the `future` package initializes the parallel computing process and determines the number of cores/workers available for computation. The `show_results()` helper function fits all three models `mcmc_ridge()`, `mcmc()` and `lmls()` for each outcome distribution in a single simulation cycle.

This entire procedure is repeated 50 times in parallel using the `future_map()` function from the `furrr` package, where the results of all 450 models are saved in a well organized structure inside of a list column. This new column of the data frame contains complete information about all simulations, such that any required element for the further analysis can be easily extracted and post processed.

Finally, the `.options()` argument allows the specification of a random seed. Random number generation in the context of parallel computing is slightly more involved compared to the sequential approach. This additional complexity is automatically handled by the `future_map()` function, such that all results are sampled in a statistically valid and fully reproducible manner.

3.5 Hyperparameters: Impact on Estimation Accuracy

In the past, we have been sampling data with the `mcmc_ridge()` function without having a closer look on the effect of the hyperparameters and model inputs `a_tau`, `b_tau`, `a_xi` and `b_xi`. However, they affect the Full Conditional Distributions of τ^2 and ξ^2, as stated in sections 1.3.1 and 1.3.2 in chapter 1.

Moreover, the mean vector $\boldsymbol{\mu}_{beta}$ and covariance matrix $\boldsymbol{\Sigma}_{beta}$ of the $\boldsymbol{\beta}$ vector both depend on τ^2 and, thus, implicitly on the hyperparameters a_τ and b_τ (see section 1.3.3). Analogously, section 1.3.4 illustrates the dependence of the Full Conditional distribution of $\boldsymbol{\gamma}$ on ξ^2, which in turn depends on the hyperparameters a_ξ and b_ξ.

Finally, cross effects can be observed, since $f(\boldsymbol{\beta} \mid \cdot)$ depends on $\boldsymbol{\gamma}$ through the quantities $\mathbf{W}$ and $\mathbf{u}$ as defined in chapter 1 and $f(\boldsymbol{\gamma} \mid \cdot)$ directly depends on $\boldsymbol{\beta}$. These connections are reflected in the `mcmc_ridge()` sampler by the iterative sampling procedure which is discussed in great detail in the previous sections 2.2.1 and 2.2.2.

Thus, the hyperparameter choice of a_τ, b_τ, a_ξ and b_ξ inevitably impacts the result of *all* coefficient estimates contained in the model in a nontrivial way, such that pure analytical reasoning might be misleading. For this reason, the following paragraphs investigate these effects based on a simulation approach.

Section 3.5 is dedicated to the impact of a_τ, b_τ, a_ξ and b_ξ on the accuracy of the Posterior Mean estimates for $\boldsymbol{\beta}$ and $\boldsymbol{\gamma}$, whereas section 3.6 examines their effect on the Ridge Penalty that is induced by the Inverse-Gamma prior distributions.

Simulation Setting

The design matrices are generated by normal distributions. Moreover, $\boldsymbol{\beta}$ is sampled more reliably from a closed form multivariate normal distribution by the `mcmc_ridge()` function (rather than by an additional Metropolis-Hastings step) in order to distinguish effects from the hyperparameters and effects caused by general estimation variability. A more detailed description of the simulation setting is listed in the Appendix.

Simulation Results

The upper panel of Figure 9 displays the absolute deviations of the Posterior Mean estimates from the true parameters with the stated different values for a_τ and b_τ. For each estimate, the Posterior Mean averages over 1000 simulations of the `mcmc_ridge()` sampler. Note, that location and scale parameters are plotted

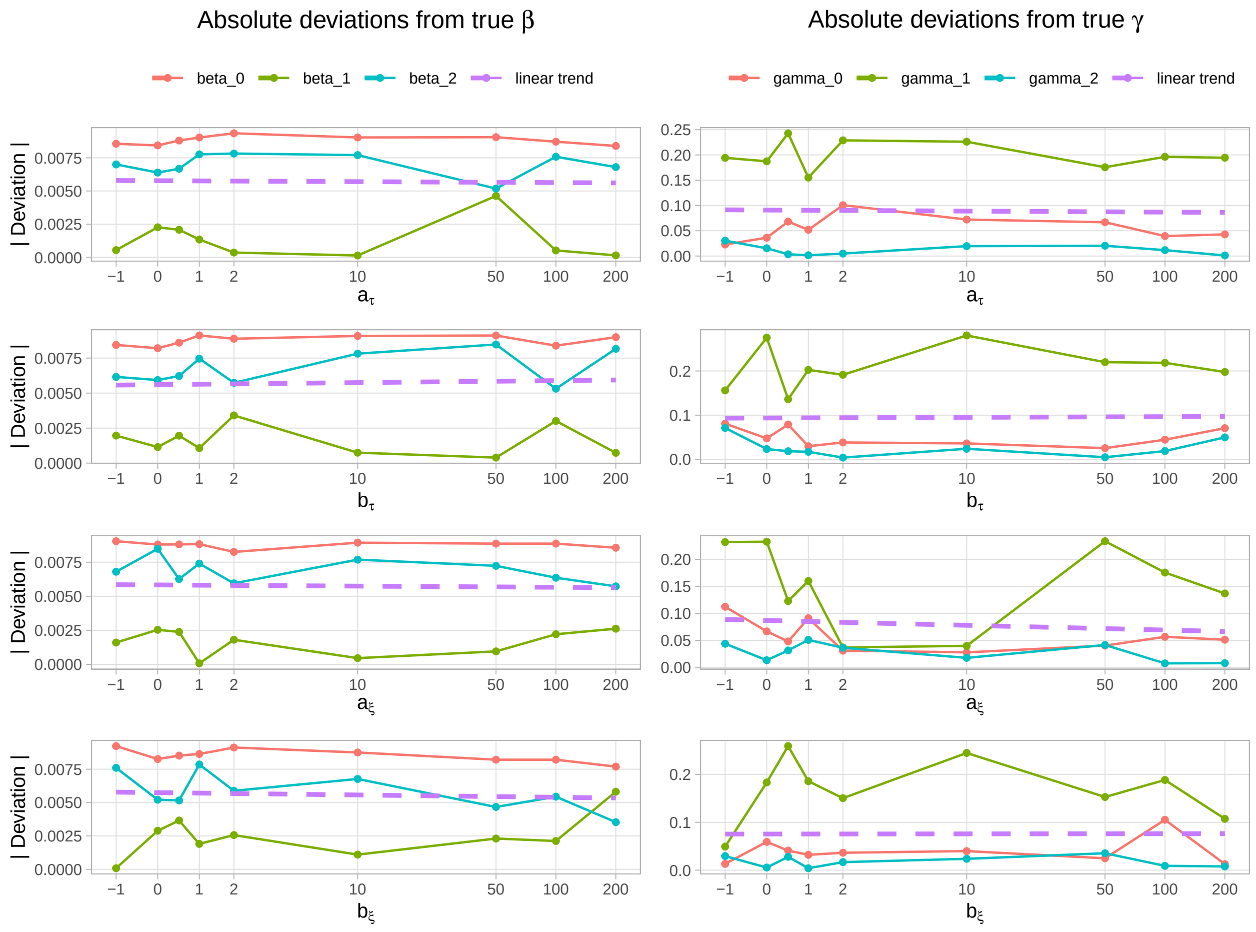

Figure 9: Absolute deviations from true $\boldsymbol{\beta}$ and $\boldsymbol{\gamma}$

separately, according to the relationship mentioned above. For a better overview, the dashed line displays the linear trend of all estimate deviations.

The x - axis is transformed by a pseudo logarithm in order to clearly visualize the deviations in the range of -1 to 10, which would not be possible on original scales. Since -1 and 0 are also part of the hyperparameter values, the `pseudo_log_trans()` function of the `scales` package is applied, log-transforming positive values only.

It can be observed, that the intercept estimates in each plot show the largest deviations from their true value for $\boldsymbol{\beta}$, whereas the largest deviations can be found for γ_1 among the $\boldsymbol{\gamma}$ estimates. In the left panel of Figure 9, however, the *overall* deviations of $\boldsymbol{\beta}$ estimates from their corresponding true value are small in absolute value. In contrast, deviations of the $\boldsymbol{\gamma}$ estimates in the right panel are fairly significant, especially for γ_1.

The functional chain that applies to the estimates of $\boldsymbol{\beta}$ can be described by the effect of the mean of the Inverse-Gamma distribution on τ^2: A larger value for b_τ leads to larger values of τ^2, which in turn affects the Full Conditional distribution of $\boldsymbol{\beta}$ and, thus, potentially magnifies the absolute deviation of the corresponding estimates from their true values. Increasing a_τ causes the opposite effect. This numerically observable effect, however, is hidden by the overall small deviation in the upper left plot of Figure 9.

It is remarkable, that the deviation of $\boldsymbol{\beta}$ estimates is smallest when $a_\tau, b_\tau \in \{100, 200\}$. For values of $a_\tau \leq 1$, one obtains wider variances of absolute deviations, since the Posterior Mean requires values larger than one.

In the upper right plots of Figure 9, there is no clear impact of τ^2 and its parameters. Rooted in no direct effect of τ^2 on $\boldsymbol{\gamma}$ according to our underlying mathematical model, one observes cross-effects through the sampling procedure of the `mcmc_ridge()` sampler, where the Full Posterior $f(\boldsymbol{\gamma} \mid \cdot)$ depends on $\boldsymbol{\beta}$. The lowest deviations of $\boldsymbol{\gamma}$ estimates are connected to values $a_\tau, b_\tau \in \{1, 50, 100\}$, where 1 likely results from randomness, since for $a_\tau \leq 1$ wide variations of absolute deviations are observable again.

The lower panel of Figure 9 is constructed analogously, but showing the impact of a_ξ and b_ξ on the location and scale parameters respectively. Again, the overall absolute deviations for the $\boldsymbol{\beta}$ estimates from their true values are small, whereas the deviations for the $\boldsymbol{\gamma}$ estimates are considerably larger.

Arguing with the mean of the Inverse Gamma distribution of ξ^2 in a similar way, one obtains larger mean values for b_ξ, while a_ξ lowers them. The impact of ξ^2 on $\boldsymbol{\gamma}$ is assumed to decrease $f(\boldsymbol{\gamma} \mid \cdot)$ according to our underlying theoretical model. This effect is indicated by the linear trend lines in the lower right part of Figure 9.

In general, one obtains no significant differences in deviations of $\boldsymbol{\gamma}$ along b_ξ, but lower deviations for larger values of a_ξ, where especially lots of randomness occurs in the deviations of γ_1. Therefore, the impact of a_ξ and b_ξ on the scale parameter is overshadowed by the randomness induced by the Metropolis-Hastings algorithm. The even wider variations exclusively for $a_\xi \leq 1$ can again be observed.

The effect of a_ξ and b_ξ on $\boldsymbol{\beta}$ can be explained through the cross-effects of the matrix $\mathbf{W}$ and the vector $\mathbf{u}$ introduced at the beginning of this section, both containing $\boldsymbol{\gamma}$. These diminish with increasing values of the $\boldsymbol{\gamma}$ entries. The smallest deviations of the location estimates can be detected for $a_\xi = 1$ and $b_\xi = 200$.

The matrix $\mathbf{W}$ affects the variance of the full conditional distribution of $\boldsymbol{\beta}$ negatively, while the mean is positively affected. Hence, larger values of a_ξ cause higher Posterior Means of the location parameters. The larger variability of deviations for $a_\xi \leq 1$ are once more observable here. In contrast, the randomness observable for scale estimates does not show up for location estimates anymore.

3.6 Hyperparameters: Impact on Ridge Penalty

This section relates the values of a_τ, b_τ, a_ξ and b_ξ to the observed Penalty on the coefficient estimates for $\boldsymbol{\beta}$ and $\boldsymbol{\gamma}$, where we exclude β_0 and γ_0 from the analysis since the intercept terms are not penalized in the Bayesian Ridge Regression model.

Simulation Setting

Many characteristics of the Inverse-Gamma distribution, such as the Mean, Median and Variance, depend in some way on the *ratio* of its two parameters. Thus, we construct grids of value pairs for a_τ and b_τ for the prior distribution of $\boldsymbol{\beta}$ and pairs of a_ξ and b_ξ for the prior distribution of $\boldsymbol{\gamma}$.

As noted before, the values of each pair do not only impact the immediate parameter one step above in the hierarchical model, but all other quantities as well due to cross effects in the Full Conditional distributions. For this reason, we evaluate the impact of each hyperparameter pair on both coefficient vectors $\boldsymbol{\beta}$ and $\boldsymbol{\gamma}$. As usual, more details about the simulation setting are provided in the Appendix.

Simulation Results

Figure 10 shows 4 Heatmaps. On the left, values of $\|\boldsymbol{\beta}\|^2$ are displayed depending on combinations of a_τ and b_τ (top panel) and a_ξ and b_ξ (bottom panel). Analogous observations for $\|\boldsymbol{\gamma}\|^2$ are shown on the right side.

In both cases, the euclidean norms are calculated without the intercept terms β_0 and γ_0 to isolate the Penalty Effect. Larger values (lower Penalty) are indicated by dark red, whereas the color blue is chosen for lower values (larger Penalty). For comparison, the unpenalized Euclidean norms of the true data generating coefficient values are given by $\|\boldsymbol{\beta}\|^2 = 68$ and $\|\boldsymbol{\gamma}\|^2 = 18$.

Starting with the top panel, there are almost no patterns observable. Interestingly, some combinations of a_τ and b_τ that are located in direct neighbourhood in the grid lead to large and small regularization effects at the same time.

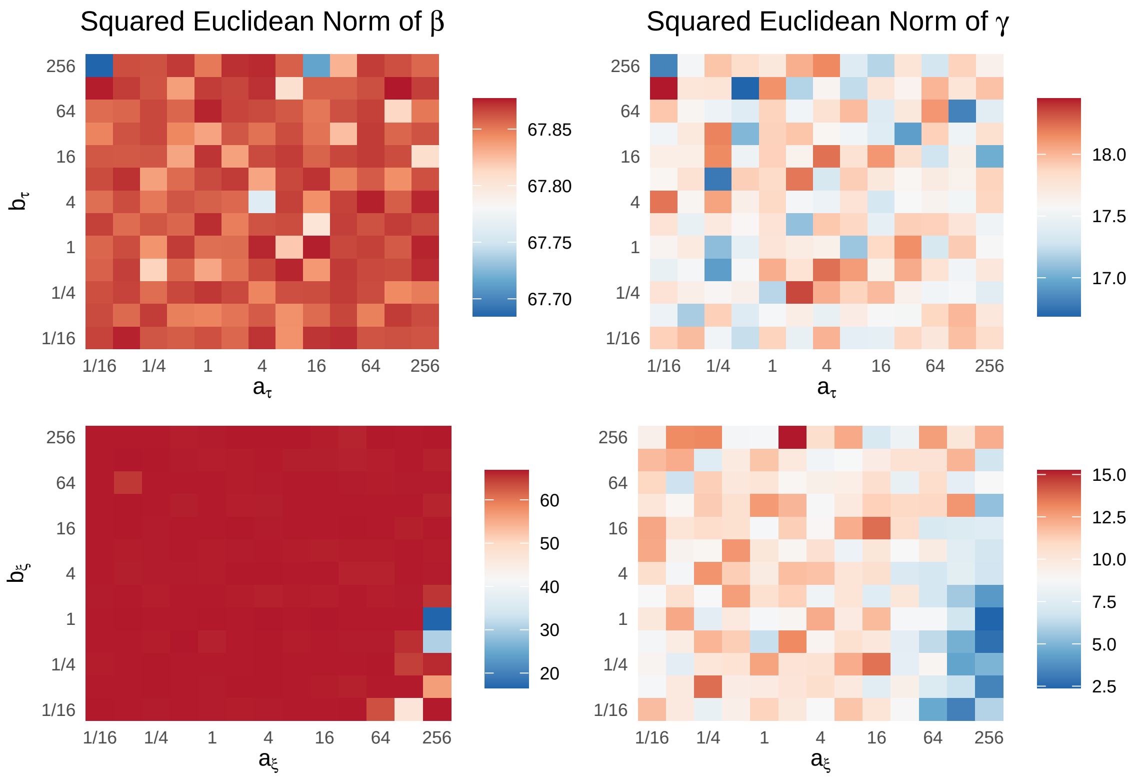

Figure 10: Heatmaps of $\|\boldsymbol{\beta}\|^2$ and $\|\boldsymbol{\gamma}\|^2$ (excluding Intercepts)

However, we have to keep the ranges of the color scales in mind. Drastically different colors are deceiving in this case, since all Euclidean norms are almost equal, indicating that the hyperparameters of τ^2 have very little impact overall on the Ridge Penalty, independent of their size. This finding is somewhat surprising, since e.g. the theoretical Posterior Mean of τ^2 differs wildly throughout the chosen combinations, suggesting a heavily varying prior variance of $\boldsymbol{\beta}$ and therefore different *potentials* for larger absolute values.

In contrast, the bottom panel shows a huge impact of a_ξ and b_ξ on the regularization. While the left plot does not look incredibly informative at first sight, there is a pattern observable that is common across both plots: If a_ξ is small and/or b_ξ is large, larger norms (and thus a lower penalty effect) of the coefficient vectors tend to occur. Only if a_ξ is large *and* b_ξ is small at the same time, there might be a significant Penalty Effect.

While the pattern of this reduction is not super clear in case of $\boldsymbol{\beta}$ and could potentially be due to randomness, the values of $\|\boldsymbol{\gamma}\|^2$ are consistently lower for $a_\xi \geq 64$ and $b_\xi \leq 4$. This combination leads to low Posterior Means for ξ^2, which again induces a lower / more informative prior variance τ^2.

The results from this simulation study are difficult to connect to their hypothesized immediate effect based on the underlying mathematical model and appear in some cases even counter-intuitive. This clearly illustrates that tweaking parameters at one end in a Bayesian model of moderate complexity might not affect the model as intended originally. Indirect effects through the cross connections in the full Conditional Distributions might dominate the simple relations through the Prior distributions. This again emphasizes the value of simulation studies to validate or possibly change the statistical methodology in practice.

3.7 Conclusion

Appendix

The following paragraphs contain additional information about the **simulation studies** that were covered in chapter 3.

Section 3.1: Correlated Predictor Variables

The simulation design was chosen in the following way:

- The design matrix $\mathbf{X} = \begin{pmatrix}\mathbf{x}_1 & \mathbf{x}_2 & \mathbf{x}_3\end{pmatrix}$ is simulated from a three dimensional normal distribution $\mathcal{N}_3(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ with mean vector $\boldsymbol{\mu} = \begin{pmatrix}-5 & 2 & 0\end{pmatrix}^T$ and covariance matrix $\begin{pmatrix}1 & \rho & \rho \\ \rho & 3 & \rho \\ \rho & \rho & 5\end{pmatrix}$. Hence, the dependence among the regressors is fully determined by the parameter ρ.
- The design matrix $\mathbf{Z} = \begin{pmatrix}\mathbf{z}_1 & \mathbf{z}_2\end{pmatrix}$ consists of linear combinations of the regressors $\mathbf{x}_1$ up to $\mathbf{x}_3$, more specifically $\mathbf{z}_1 = 0.8 \cdot \mathbf{x}_1 + 0.2 \cdot \mathbf{x}_2$ and $\mathbf{z}_2 = \mathbf{x}_2 - 0.5 \cdot \mathbf{x}_3$. In both design matrices intercept columns are added for estimation purposes.
- The true coefficient vectors are given by $\boldsymbol{\beta} = \begin{pmatrix}\beta_0 & \beta_1 & \beta_2 & \beta_3\end{pmatrix}^T = \begin{pmatrix}0 & 3 & -1 & 1\end{pmatrix}^T$ and $\boldsymbol{\gamma} = \begin{pmatrix}\gamma_0 & \gamma_1 & \gamma_2\end{pmatrix}^T = \begin{pmatrix}0 & 2 & 0\end{pmatrix}^T$.
- The outcome variable $\mathbf{y}$ is generated according to the correctly specified location-scale model $y_i \overset{iid}{\sim} \mathcal{N}\left(\mathbf{x}_i^T\boldsymbol{\beta}, \exp\left(\mathbf{z}_i^T\boldsymbol{\gamma}\right)^2\right)$ for $i = 1, \ldots, n$ with sample size $n = 50$. Before data generation all columns in $\mathbf{X}$ and $\mathbf{Z}$ are standardized, i.e. mean-centered around 0 and scaled to unit variance.
- Three different values were chosen for $\rho \in \{0, -0.5, 0.9\}$ to compare the 'nice' case of uncorrelated predictors with the performance for negative and positive dependence. For each covariance structure the three models `mcmc_ridge()`, `mcmc()` and `lmls()` are fitted to the standardized covariates, where each Posterior Mean estimate from both of the Markov Chain Monte Carlo samplers is based on 10.000 samples.

Section 3.2: Sample Size

The simulation study is based on the following conditions:

- The design matrix $\mathbf{X} = \begin{pmatrix}\mathbf{1}_n & \mathbf{x}_1 & \mathbf{x}_2\end{pmatrix}$ contains two independently sampled regressor variables plus one intercept column:
  - $\mathbf{x}_1 \overset{iid}{\sim} \mathcal{N}(1, 1)$,
  - $\mathbf{x}_2 \overset{iid}{\sim} \mathcal{N}(2, 1)$.
- The design matrix $\mathbf{Z} = \begin{pmatrix}\mathbf{1}_n & \mathbf{z}_1 & \mathbf{z}_2\end{pmatrix}$ is structured in the same way with the regressor variables:
  - $\mathbf{z}_1 \overset{iid}{\sim} \mathcal{N}(1, 1)$,
  - $\mathbf{z}_2 \overset{iid}{\sim} \mathcal{N}(2, 1)$.
- After sampling the matrices $\mathbf{X}$ and $\mathbf{Z}$, the columns of $\mathbf{x}_1, \mathbf{x}_2, \mathbf{z}_1$ and $\mathbf{z}_2$ are standardized.
- The true coefficient vectors are given by $\boldsymbol{\beta} = \begin{pmatrix}\beta_0 & \beta_1 & \beta_2\end{pmatrix}^T = \begin{pmatrix}1 & -1 & 4\end{pmatrix}^T$ and $\boldsymbol{\gamma} = \begin{pmatrix}\gamma_0 & \gamma_1 & \gamma_2\end{pmatrix}^T = \begin{pmatrix}0 & -0.5 & 1\end{pmatrix}^T$.
- The posterior means are analyzed with respect to 6 different sample sizes $n \in \{0, 50, 100, 200, 300, 500\}$.
- In the next step, the outcome vector $y \in \mathbb{R}^n$ is simulated and passed to the `mcmc_ridge()` function with `nsim` $= 500$ simulations.

- To make the results more stable, the above procedure is repeated 100 times. For each coefficient, the mean value of the Posterior Mean estimates of each coefficient is calculated as well as the Mean Absolute Error (MAE) with respect to the true values of $\boldsymbol{\beta}$ and $\boldsymbol{\gamma}$.

Section 3.3: Redundant Covariates

We again state the conditions that the simulation study is based on:

- The design matrix $\mathbf{X} = \begin{pmatrix}\mathbf{1}_n & \mathbf{x}_1 & \cdots & \mathbf{x}_{20}\end{pmatrix}$ consists of one intercept column plus 10 *pairs* of successive regressors, starting with the pair $(\mathbf{x}_1, \mathbf{x}_2)$. Each pair $(\mathbf{x}_i, \mathbf{x}_{i+1})$ for $i \in \{1, 3, \dots, 19\}$ is (independently from all remaining pairs) drawn from a bivariate normal distribution with mean vector $\boldsymbol{\mu} = \begin{pmatrix}0 & 0\end{pmatrix}^T$ and correlation matrix $\begin{pmatrix}1 & 0.9\\ 0.9 & 1\end{pmatrix}$.
- The design matrix $\mathbf{Z} = \begin{pmatrix}\mathbf{1}_n & \mathbf{x}_1 & \mathbf{x}_3\end{pmatrix}$ is of minor interest in this case and consists of an intercept column plus two uncorrelated columns chosen from $\mathbf{X}$.
- The true coefficients of $\boldsymbol{\beta}$ are determined by the pattern $\beta_i = 0$, if i is even and $\beta_i = 1$, if i is odd. Thus, all covariates with even subscripts are redundant, whereas those with odd subscripts contribute to $\mathbf{y}$. The true $\boldsymbol{\gamma}$, again of minor interest here, is given by $\boldsymbol{\gamma} = \begin{pmatrix}0 & 1 & 1\end{pmatrix}^T$.
- The outcome variable $\mathbf{y}$ is generated according to the correctly specified location-scale model $y_i \overset{iid}{\sim} \mathcal{N}\left(\mathbf{x}_i^T\boldsymbol{\beta}, \exp\left(\mathbf{z}_i^T\boldsymbol{\gamma}\right)^2\right)$ for $i = 1, \dots, n$, where the covariates are used on their original scale.
- The sample size $n = 50$ is deliberately chosen small compared to the number of regressors. Before fitting each of the three models, all columns of $\mathbf{X}$ except the intercept column are standardized to zero mean and unit variance. Both of the Bayesian models generate 10.000 simulated values for each coefficient.

Section 3.4: Challenging the Model Assumptions

The data for this simulation study is generated by the following conventions:

- The design matrix $\mathbf{X} = \begin{pmatrix}\mathbf{1}_n & \mathbf{x}_1 & \mathbf{x}_2 & \mathbf{x}_3 & \mathbf{x}_4\end{pmatrix}$ contains four independently sampled regressor variables plus one intercept column:
  - $\mathbf{x}_1 \overset{iid}{\sim} \mathcal{N}(5, 16)$,
  - $\mathbf{x}_2 \overset{iid}{\sim} \text{Exp}(5)$,
  - $\mathbf{x}_3 \overset{iid}{\sim} \mathcal{U}([-2,\ 12])$,
  - $\mathbf{x}_4 \overset{iid}{\sim} \text{Ber}(0.3)$.
- The design matrix $\mathbf{Z} = \begin{pmatrix}\mathbf{1}_n & \mathbf{x}_1 & \mathbf{x}_2 & \mathbf{z}_3\end{pmatrix}$ contains the additional regressor variable $\mathbf{z}_3 \overset{iid}{\sim} t_{10}$, which is independently sampled from all other columns.
- The true coefficient vectors are given by $\boldsymbol{\beta} = \begin{pmatrix}\beta_0 & \beta_1 & \beta_2 & \beta_3 & \beta_4\end{pmatrix}^T = \begin{pmatrix}0 & -10 & -5 & -3 & -1\end{pmatrix}^T$ and $\boldsymbol{\gamma} = \begin{pmatrix}\gamma_0 & \gamma_1 & \gamma_2 & \gamma_3\end{pmatrix}^T = \begin{pmatrix}0 & 3 & 5 & 10\end{pmatrix}^T$.
- Three different specifications for the outcome distribution were chosen:
  - $y_i \sim \mathcal{N}\left(\mu, \sigma^2\right)$,
  - $y_i \sim \mu + \left(\sigma \cdot \sqrt{\frac{3}{5}}\right) T$, where $T \sim t_5$,
  - $y_i \sim \mu + \sigma \cdot U$, where $U \sim \mathcal{U}([0,\ 1])$.

  In all cases, the outcome vectors are generated with the covariates on their original (unstandardized) scale.

- In order to isolate the impact of the different shapes of the three probability distributions from the effect of varying moment structures, the mean $\mu = \mathbf{x}_i^T \boldsymbol{\beta}$ and the variance $\sigma^2 = \exp\left(\mathbf{z}_i^T \boldsymbol{\gamma}\right)^2$ are held constant across the models.
- All three models `mcmc_ridge()`, `mcmc()` and `lmls()` are fitted with standardized covariates. The sample size is set to $n = 50$ and the result of both Bayesian samplers are based on 10.000 simulations.

Section 3.5: Hyperparameters - Impact on Estimation Accuracy

The simulation study of the impact of the Hyperparameters on the estimated coefficients is constructed as follows:

- The design matrix $\mathbf{X} = \begin{pmatrix}\mathbf{x}_1 & \mathbf{x}_2\end{pmatrix}$ is simulated from a two dimensional normal distribution $\mathcal{N}_2\left(\boldsymbol{\mu}, \boldsymbol{\Sigma}\right)$ with mean vector $\boldsymbol{\mu} = \begin{pmatrix}1 & 2\end{pmatrix}^T$ and identity covariance matrix $\boldsymbol{\Sigma} = \mathbf{I}_2$. The same holds true for the design matrix $\mathbf{Z} = \begin{pmatrix}\mathbf{z}_1 & \mathbf{z}_2\end{pmatrix}$ with mean vector $\boldsymbol{\mu} = \begin{pmatrix}5 & 3\end{pmatrix}^T$ and identity covariance matrix. However, after simulating $\mathbf{X}$ and $\mathbf{Z}$, both are standardized to zero mean and unit variance.
- In both design matrices intercept columns are added for estimation purposes. The true coefficient vectors are given by $\boldsymbol{\beta} = \begin{pmatrix}\beta_0 & \beta_1 & \beta_2\end{pmatrix}^T = \begin{pmatrix}0 & -1 & 4\end{pmatrix}^T$ and $\boldsymbol{\gamma} = \begin{pmatrix}\gamma_0 & \gamma_1 & \gamma_2\end{pmatrix}^T = \begin{pmatrix}0 & -2 & 1\end{pmatrix}^T$.
- The outcome variable $\mathbf{y}$ is generated according to the correctly specified location-scale model $y_i \overset{iid}{\sim} \mathcal{N}\left(\mathbf{x}_i^T \boldsymbol{\beta}, \exp\left(\mathbf{z}_i^T \boldsymbol{\gamma}\right)^2\right)$ for $i = 1, \ldots, n$ with sample size $n = 50$ as well as standardized data matrices $\mathbf{X}$ and $\mathbf{Z}$.
- For sampling the location parameter, the full conditional multivariate normal distribution of $\boldsymbol{\beta}$ is chosen, i.e. `mcmc_ridge(..., mh_location = FALSE)` is used. Therefore, the location estimate is directly affected by the hyperparameters.
- For simulating the influence of the hyperparameters, nine different values are chosen: $a_\tau, b_\tau, a_\xi, b_\xi \in \{-1, 0, 0.5, 1, 2, 10, 50, 100, 200\}$. Since for statistical properties like the mean of an Inverse Gamma distribution $\frac{b}{a-1}$ the condition $a > 1$ is required, particular attention is given to larger values. However, it is an aim to inspect the performance of the sampler for hyperparameter values smaller than 1 as well.

Section 3.6: Hyperparameters - Impact on Ridge Penalty

This simulation study is conducted in the following way:

- The column vectors of the design matrices $\mathbf{X} = \begin{pmatrix}\mathbf{x}_1 & \mathbf{x}_2\end{pmatrix}$ and $\mathbf{Z} = \begin{pmatrix}\mathbf{z}_1 & \mathbf{z}_2\end{pmatrix}$ are independently drawn from normal distributions with variance $\sigma^2 = 1$ and varying means $\mu \in (1, 2, 5, 3)$. Then both design matrices are standardized and intercept columns are added.
- The true coefficient vectors are given by $\boldsymbol{\beta} = \begin{pmatrix}\beta_0 & \beta_1 & \beta_2\end{pmatrix}^T = \begin{pmatrix}0 & 8 & 2\end{pmatrix}^T$ and $\boldsymbol{\gamma} = \begin{pmatrix}\gamma_0 & \gamma_1 & \gamma_2\end{pmatrix}^T = \begin{pmatrix}0 & 3 & 3\end{pmatrix}^T$.
- The outcome variable $\mathbf{y}$ is generated based on the standardized covariates according to the correctly specified location-scale model $y_i \overset{iid}{\sim} \mathcal{N}\left(\mathbf{x}_i^T \boldsymbol{\beta}, \exp\left(\mathbf{z}_i^T \boldsymbol{\gamma}\right)^2\right)$ for $i = 1, \ldots, n$ with sample size $n = 50$.
- The hyperparameter pairs (a_τ, b_τ) and (a_ξ, b_ξ) take values on a grid, which is constructed by all combinations of the sequence $\left(\frac{1}{16}, \frac{1}{8}, \frac{1}{4}, \frac{1}{2}, 1, 2, 4, 8, 16, 32, 64, 128, 256\right)$. While one pair is varied, the other pair is fixed on the values $(1, 1)$.
- The `mcmc_ridge()` function does not use the `lmls()` function as basis in this case, but rather works with the standardized data matrices $\mathbf{X}$ and $\mathbf{Z}$ directly. The number of simulations `num_sim` is chosen as 1000 and the starting values `beta_start` and `gamma_start` are set to $\begin{pmatrix}1 & 1\end{pmatrix}^T$, respectively.