The `asp21bridge` Package

1 The `asp21bridge` Package

This chapter introduces various facettes of the `asp21bridge` package. Section 1.1 explains how to use and combine the functions that are contained in the package most efficiently. Section 1.2 focuses on the implementation of the Markov Chain Monte Carlo Sampler with Ridge Penalty, links the source code to the mathematical model from chapter **??** and explains, how the main function `mcmc_ridge()` as well as the helper functions implementing the Metropolis-Hastings algorithm are structured. Finally, some additional components of the package development process and ideas that the package is based upon are discussed in section 1.3.

1.1 User Guide

This section aims to provide help 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 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, that 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 1.2.1.
- *Graphical Analysis:* The `trace_plot()`, `density_plot()` and `acf_plot()` functions provide the three most common visualization of a *single* chain's development over time, their 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 will be used more often than the separate building blocks. 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 corresponding density plots are contained in the lower panel.
- *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 the context of Markov Chains and in particular for extracting more meaningful features of the samples from the posterior distributions.

1.1.1 Sampling with the `mcmc_ridge()` function

As explained in section 1.2.1 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 simulation `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 **??**), 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 introductory tutorials, documentation and unit tests. It consists of a column `y` representing a vector of observed values and the explanatory variables `x1`, `x2`, `z1` and `z2`. The data is simulated according to the correctly specified location-scale regression model from chapter **??**, 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 number of simulations which we increase to 10000.

- 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 model 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 10000 such that stable and statistically valid estimates of posterior characteristics are possible, even if preceding 'burnin' and 'thinning' steps might be required.

```
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)

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

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

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

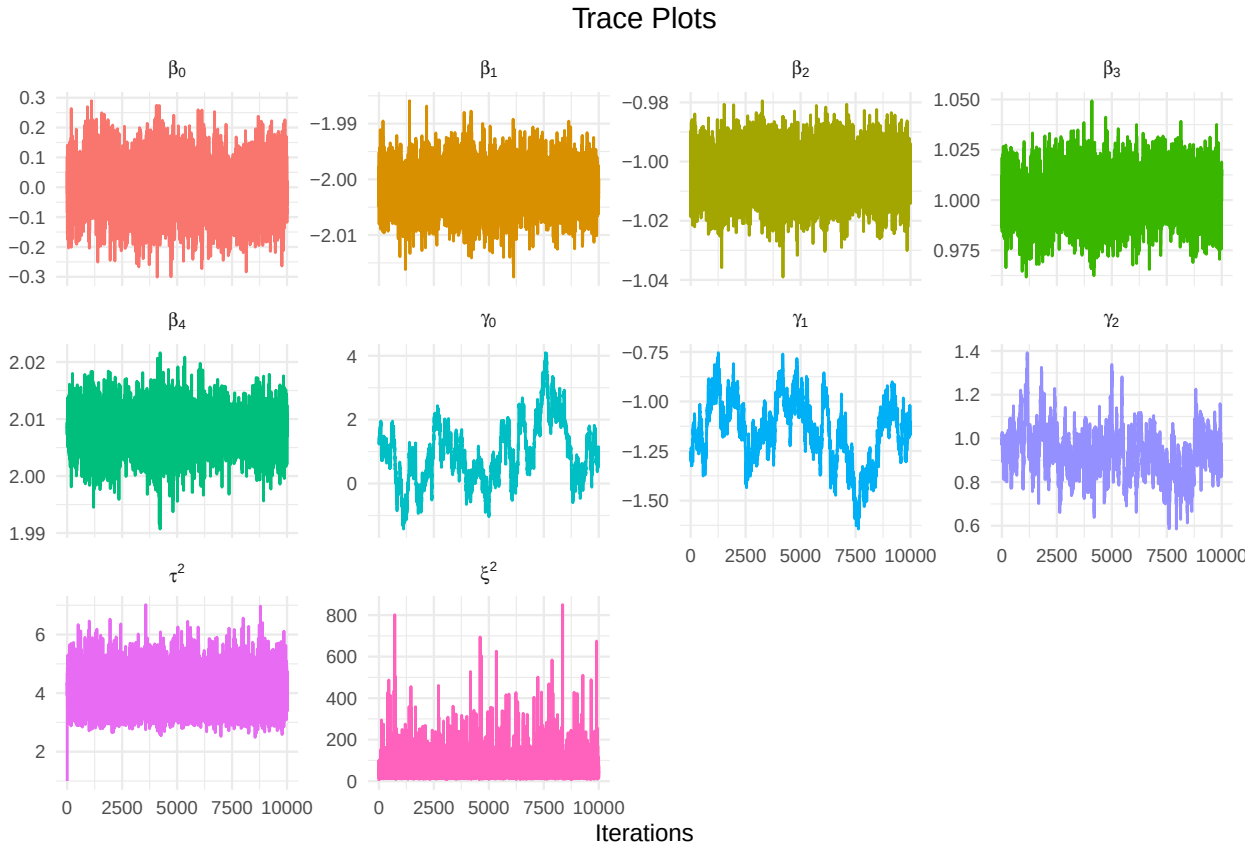
```
str(abdom_fit, max.level = 1)
## List of 2
##  $ sampling_matrices:List of 4
##  $ acceptance_rate  : num 0.431
```

1.1.2 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 of the sampling matrices to their corresponding greek symbols, which, although being a purely aesthetic feature, required a surprisingly nontrivial implementation:

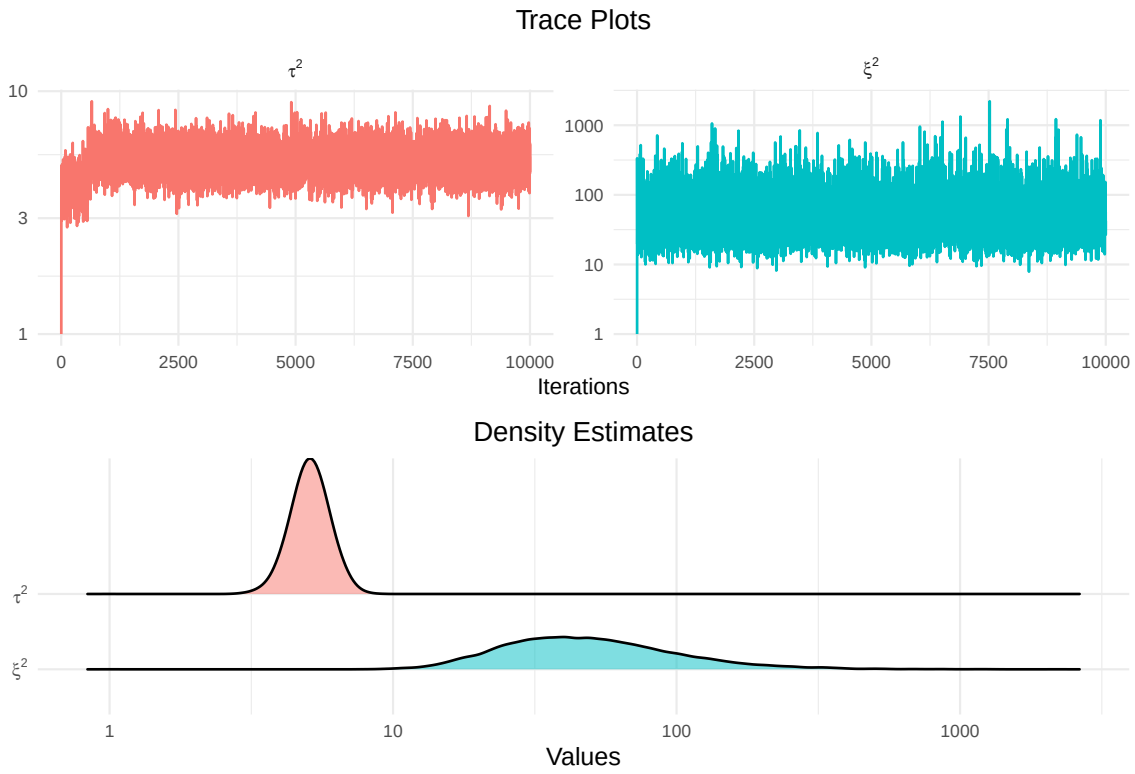
```
mult_plot(samples = toy_fit, type = "trace", free_scale = TRUE, latex = TRUE)
```



Due to the high number of simulations, the trace plots appear a bit overloaded. Considering the y-axis scales, the samples for the β 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:

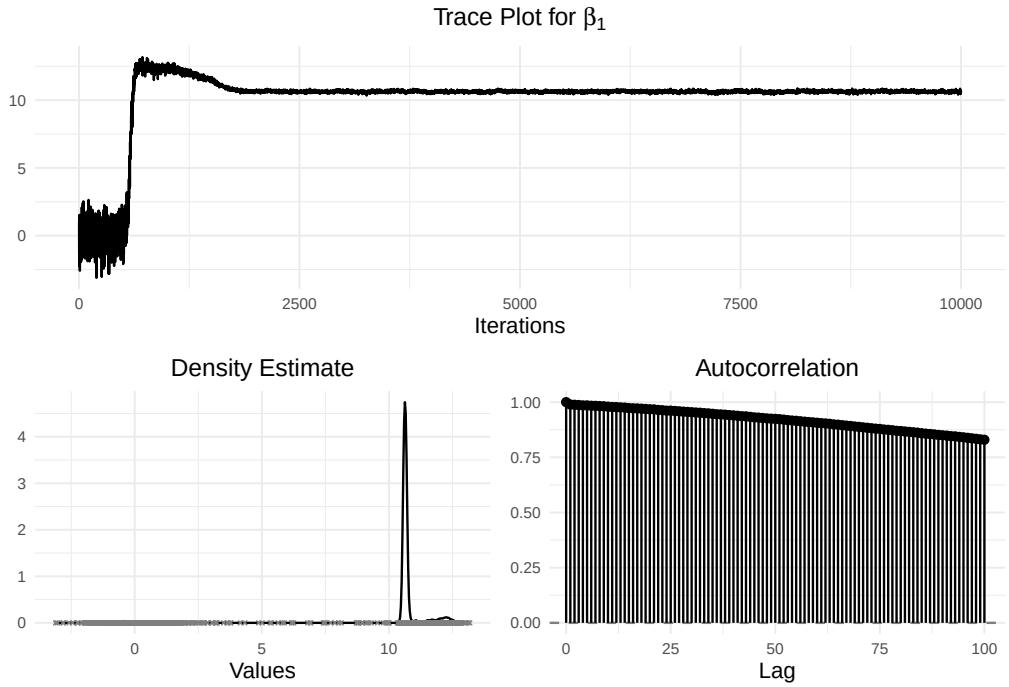
```
variance_samples <- cbind(
  abdom_fit$sampling_matrices$tau_samples,
  abdom_fit$sampling_matrices$xi_samples
)

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



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 corresponding samples.

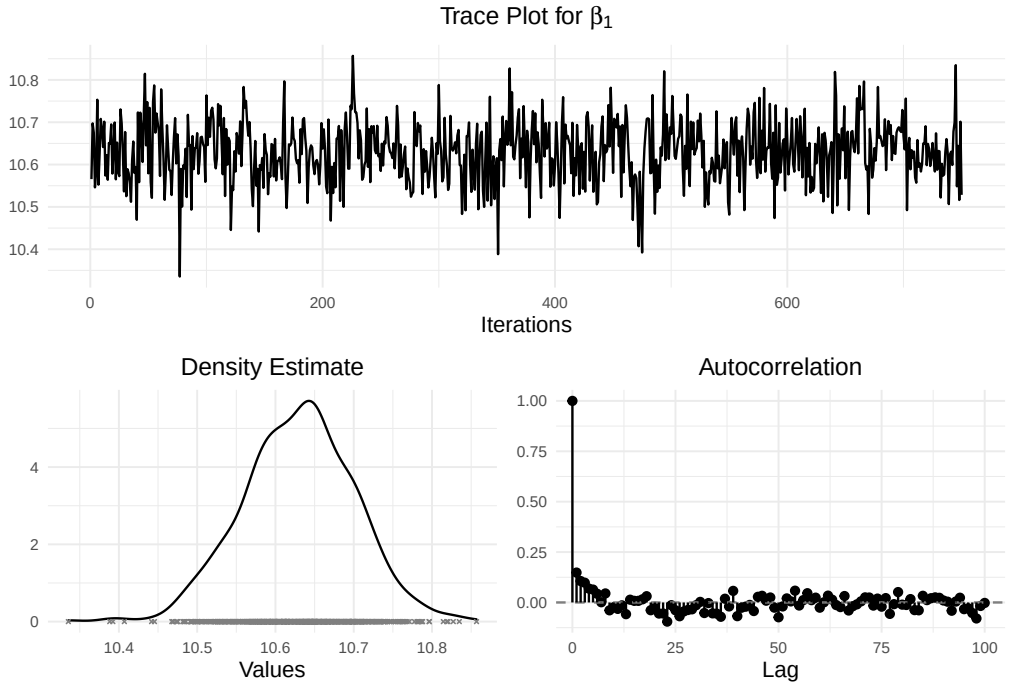
To focus on a single Markov chain, the `diagnostic_plots()` function is useful. The trace plot of β_1 for the `abdom_fit` model clearly shows the need for a burnin and thinning step, since the chain seem to converge after roughly 2500 iterations and the samples are strongly autocorrelated:

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



Thus, we remove the first 2500 samples and additionally only keep every 10th simulation.

```
abdom_fit$sampling_matrices$beta_samples[, "beta_1", drop = FALSE] %>%
  burnin(num_burn = 2500) %>%
  thinning(freq = 10) %>%
  diagnostic_plots(lag_max = 100, latex = TRUE)
```



These plots look much better: The chain has converged and there is little residual autocorrelation 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 uncon-

troversial in the Bayesian community, statistical estimates from the remaining, well behaved chain are usually more stable and reliable. In any case, the above procedure clearly emphasizes the usefulness of a graphical diagnosis of the sampling results as a first step before drawing any far reaching conclusions.

1.1.3 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. Note that this option is only applicable for the `toy_fit` model, which is based on the `lslm` model:

```
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
## 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 thorough `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 coefficient values:

```r
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 form the true values for the remaining $\boldsymbol{\gamma}$ vector, 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 1.1.2, 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 argument `include_plot` argument. When 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 output 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. If we are interested, if the chain suffers from a lack of convergence or a significant autocorrelation, we could create the desired output with the following simple command:

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

This introductory tutorial is not intended to underline the sampler's performance or validity, but rather provide an overview of various applications, where the `asp21bridge` package can be useful. A more in depth analysis of the `mcmc_ridge()` results, also in comparison to alternative models from the `lmls()` and `mcmc()` functions of the underlying `lmls` package, can be found in chapter **??** after explaining some of the internal implementations in section 1.2.

1.2 Implementation of the Markov Chain Monte Carlo Sampler

1.2.1 Iterative Parameter Sampling

1.2.2 Metropolis Hastings Step

1.3 Package Development

This section is dedicated to more niche aspects that come along with the package development process. Part 1.3.1 focuses on the more formal components of the package, whereas some of the ideas beyond the `asp21bridge` functions are discussed in section 1.3.2.

1.3.1 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. Specifically we put effort into precise and not excessively long descriptions of the overall function and their parameters with highlighted default settings. The structure of the help pages is kept uniformly across all functions. The `examples` section usually starts with the most basic applications signaling the function's *intent* and proceeding with more complex use cases that cover many of the function's optional arguments.

A more elaborate publicly accessible tutorial similar to section 1.1 of this report can be found in the `README` file or the front page of the `asp21bridge` GitLab website.

A second major aspect, that contributes to the quality of a package are **unit tests**. At the time of writing, 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 only capture a small fraction of the entire package's 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 functions such as the main `mcmc_ridge()` function. Further, not only simple input checks (which are important nonetheless!), but also more esoteric edge cases and in particular 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 joyful 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 for example 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 merged into the master branch at a later stage after a thorough testing process.

1.3.2 Design Choices

- pipe operator
- robustness to inputs and helpful error messages
- modularity
- style conventions (tidyverse style guide and styler package)
- S3 Class