

ggmcmc: Analysis of MCMC Samples and Bayesian Inference

Xavier Fernández-i-Marín
ESADE Business School

Abstract

ggmcmc is an R package for analyzing Markov chain Monte Carlo simulations from Bayesian inference. By using a well known example of hierarchical/multilevel modeling, the article reviews the potential uses and options of the package, ranging from classical convergence tests to caterpillar plots or posterior predictive checks. *This R vignette is based on the article published at the Journal of Statistical Software (?).*

Keywords: **ggmcmc**, MCMC, Bayesian inference.

1. Introduction

This article presents **ggmcmc**, an R package with graphical tools for analyzing Markov chain Monte Carlo (MCMC) simulations from Bayesian inference.

The article is organized as follows: Section 2 presents the motives for developing the package and the basic logic of its design and implementation. In order to have some data to plot, a dataset and sampled values from posterior distributions of a varying intercepts/varying slopes model are presented in Section 3. The `ggs()` function used to prepare data for **ggmcmc** is presented in Section 4. The `ggmcmc()` wrapper that writes all plots into a PDF (Portable Document Format) or HTML (HyperText Markup Language) file is reviewed in Section 5. Section 6 explains how to use the individual functions and what to expect from them in terms of convergence. Brief explanations of the potential uses of the `family` argument for selecting groups of parameters and the `par_label` argument for changing parameter names are performed in Section 7. The capacities of the caterpillar plots facilities in **ggmcmc** are reviewed in Section 8. Section 9 presents functions to perform model check and posterior predictive comparisons. Section 10 shows several examples of the potential extensions of the package, either alone or in combination with other packages and Section 11 concludes.

ggmcmc is available from the Comprehensive R Archive Network at <https://CRAN.R-project.org/package=ggmcmc> while the latest version and the discussions about its future can be followed at <http://github.com/xfim/ggmcmc>.

2. Why ggmcmc?

ggplot2, based on the grammar of graphics (?), empowers R users by allowing them to flexibly create graphics (?). Based on this idea, **ggmcmc** brings the design and implementation of

ggplot2 to MCMC diagnostics, allowing users of Bayesian inference to have better and more flexible visual diagnostic tools.

Markov chain Monte Carlo methods are used to generate samples from a probability distribution. They are widely used nowadays in many aspects of optimization and numerical integration, and are specially suitable for being used in sampling from posterior distributions in Bayesian inference. A key issue in MCMC is whether the chain has converged and is actually sampling from the target distribution. There is not a single infallible test of convergence, but many formal and informal ways to assess non-convergence.

There are currently two general-purpose packages for convergence analysis and diagnostics in R: **coda** (?) and **boa** (?). **boa** was last updated in 2008 and is less complex than **coda**, which is still maintained and considered to be the reference package. **coda** relies on the class **mcmc** to produce its results, which is the *de facto* standard class for MCMC objects in R.

ggmcmc combines elements of those general-purpose MCMC analysis packages with **ggplot2**. **ggplot2** is based on the idea that the input of any graphic is a data frame mapped to aesthetic attributes (color, size) of geometric objects (points, lines). Therefore, in order to create a graphic the three components must be supplied: a data frame, at least one aesthetic attribute and at least one geometric object. The flexibility comes from the fact that it is very easy to extend basic graphics by including more aesthetic elements and geometric objects, and even faceting the figure to generate the same plot for different subsets of the dataset (?, p. 3).

The implementation of **ggmcmc** follows this scheme and is based on a function (**ggs()**) that transforms the original input (time series of sampled values for different parameters and chains) into a data frame that is used for all graphing functions. The plotting functions do any necessary transformation to the samples and return a **ggplot** object, which can be plotted directly into the working device or simply stored as an object, as any other **ggplot** object. Getting **ggplot** objects as the output of the plotting functions has a positive side effect: while the defaults in **ggmcmc** have been carefully chosen, the user later can tweak any graph by adding other geometric figures, layers of data, contextual information (titles, axes) or applying themes.

To sum up, the implementation is driven by the following steps:

1. Convert any input into a data frame using **ggs()**, producing a tidy object (?).
2. Make all plotting functions (**ggs_*()**) work with the **ggs** object and produce a **ggplot** object.
3. Let the user post-process the resulting default **ggplot** object.

Compared to **coda**, **ggmcmc** does not use a specific class for its objects, but instead relies on a general-purpose object, the data frame, as the basis for all the operations. This approach also allows **ggmcmc** to work well with other tools designed to work with data frames, such as **dplyr** and **tidyr** (?) – which, in fact, are **ggmcmc** dependencies. Another advantage of **ggmcmc** is that it is relatively easy to convert any output of a sampling software into a data frame, which gives **ggmcmc** a lot of flexibility to be able to import samples from different software.

In addition to the need of good tools for convergence check, which is already covered by other packages in R, MCMC is usually used in conjunction with hierarchical (multilevel) models

in Bayesian inference, and there is not a versatile tool to process such samples in an easy and flexible way. **ggmcmc** aims also to provide functions to inspect batches of parameters in hierarchical models, to do posterior predictive checks and other model fit figures.

3. Get samples from Gelman & Hill radon example

In order to show the potential of **ggmcmc** with real data and a well-known dataset, this section briefly presents Gelman's example of radon and floor measurements for 919 houses from 85 counties in Minnesota. The model is a varying intercepts and slopes specification introduced in Section 13.1 (pages 280–283) and presented in BUGS/JAGS notation in Section 17.1 (page 401). Equation 1 presents the equation with the model definition.

$$\begin{aligned}
 y_i &\sim \mathcal{N}(\hat{y}_i, \sigma) \\
 \hat{y}_i &= \alpha_j + X\beta_j \\
 \alpha_j &\sim \mathcal{N}(\theta_\alpha, \sigma_\alpha) \\
 \beta_j &\sim \mathcal{N}(\theta_\beta, \sigma_\beta) \\
 \theta_\alpha &\sim \mathcal{N}(0, 1000) \\
 \theta_\beta &\sim \mathcal{N}(0, 1000) \\
 \sigma_\alpha &\sim \mathcal{U}(0, 100) \\
 \sigma_\beta &\sim \mathcal{U}(0, 100)
 \end{aligned} \tag{1}$$

The radon measurement (y_i) is explained by varying intercepts (α_j) and varying slopes (β_j) associated with a single covariate (soil uranium).

ggmcmc contains a list (`radon`) with several objects from this model sampled using JAGS: `s.radon` is an `mcmc` object containing samples of 2 chains of length 1000 thinned by 50 (resulting in only 20 samples) for all 175 parameters (batches of α and β , and θ_α and also θ_β , σ_α , σ_β and σ_y). `s.radon.short` contains only 4 parameters for pedagogical purposes: $\alpha[1 : 2]$ and $\beta[1 : 2]$, but the chains have not been thinned. This is the object that will be employed in the first part of the article.

```

R> library("ggmcmc")
R> data("radon")
R> s.radon.short <- radon$s.radon.short

```

4. Importing MCMC samples into ggmcmc using ggs()

The `s.radon.short` object is right now a list of arrays of an `mcmc` class. Each element in the list is a chain, and each matrix is defined by the number of iterations (rows) and the number of parameters (columns). In order to work with **ggplot2** and to follow the rules of the grammar of graphics, data must be converted into a data frame. This is achieved with `ggs()`:

```
R> S <- ggs(s.radon.short)
```

`ggs()` produces a data frame object with four variables, namely:

- **Iteration:** Number of iteration.
- **Chain:** Number of the chain.
- **Parameter:** Name of the parameter.
- **value:** value sampled.

More specifically, `ggs()` produces a `tbl_df`, which is a data frame wrapper in **dplyr** that improves printing data frames. In this case, calling the object produces a compact view of its contents.

```
R> S
```

```
## Source: local data frame [16,000 x 4]
##
##   Iteration Chain Parameter      value
##   <int> <int>   <fctr>    <dbl>
## 1         1     1 alpha[1]  1.9675268
## 2         2     1 alpha[1]  1.0572830
## 3         3     1 alpha[1]  1.1732732
## 4         4     1 alpha[1]  1.2637594
## 5         5     1 alpha[1]  1.0365003
## 6         6     1 alpha[1]  1.7117239
## 7         7     1 alpha[1]  1.0868363
## 8         8     1 alpha[1]  0.8044958
## 9         9     1 alpha[1]  0.8232430
## 10        10     1 alpha[1]  1.1274792
## ..      ...     ...           ...     ...
```

A `ggs` object is generally around twice the size of a `mcmc` object (list of matrices), because the iteration number, the number of the chain and the name of the parameter are stored in the resulting object in a less compact way than in `mcmc`. But this duplication of information gives much more flexibility to the package, in the sense that the transformation is done only once and then the resulting object is suitable and ready for the rest of the plotting functions. The resulting object is tidy (?).

In addition to producing a data frame, `ggs()` also stores some attributes in the data frame, which will be later used by the rest of the functions. This speeds up the package performing expensive operations once and caching the results.

```
R> str(S)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame': 16000 obs. of  4 variables:
## $ Iteration: int  1 2 3 4 5 6 7 8 9 10 ...
## $ Chain      : int  1 1 1 1 1 1 1 1 1 1 ...
## $ Parameter: Factor w/ 4 levels "alpha[1]","alpha[2]",...: 1 1 1 1 1...
## $ value      : num  1.97 1.06 1.17 1.26 1.04 ...
## - attr(*, "nChains")= int 2
## - attr(*, "nParameters")= int 4
## - attr(*, "nIterations")= int 2000
## - attr(*, "nBurnin")= num 12000
## - attr(*, "nThin")= num 10
## - attr(*, "description")= chr "s.radon.short"
```

In addition to JAGS output mentioned above, `ggs()` can incorporate MCMC outputs from several sources:

- JAGS which produces objects of class `mcmc.list` (?).
- **MCMCpack** which produces objects of class `mcmc` (?).
- **rstan**, **rstanarm** and **brms** which produce, respectively, objects of classes `stanfit` (?), `stanreg` (?) and `brmsfit` (?).
- Stan running alone from the command which produces text files in comma-separated value (CSV) format (?).

5. Using `ggmcmc()`

`ggmcmc()` is a wrapper to several plotting functions that allows to create very easily a report of the diagnostics in a single PDF or HTML file. This output can then be used to inspect the results more comfortably than using the plots that appear in the screen.

```
R> ggmcmc(S)
```

By default, `ggmcmc()` produces a file called `ggmcmc-output.pdf` with 5 parameters in each page, although those default values can be easily changed.

```
R> ggmcmc(S, file = "model_simple-diag.pdf", param_page = 2)
```

It is also possible to specify `NULL` as a filename, and this allows the user to control the device. It is possible to combine other plots by first opening a PDF device (`pdf(file = "new.pdf")`), sending other plots, running and the `ggmcmc(S, file = NULL)` call, and finally closing the device (`dev.off()`).

It is also possible to ask `ggmcmc()` to dump only one or some of the plots, using their names as in the functions, without `ggs_`. This option can also be used to dump only one type of plots and get the advantage of having multiple pages in the output.

```
R> ggmcmc(S, plot = c("density", "running", "caterpillar"))
```

The usual procedure with `coda` nowadays is that once the model has been running, the user interactively does some convergence tests, mostly concentrating in the traceplots and density plots in a sequential way. The idea of having a single file (PDF or HTML) with all the graphics is that, in addition to running the MCMC simulations and waiting for the result, the user can also run `ggmcmc` to produce all the convergence plots for review at a later stage, going back and forward in the pages, comparing the behavior of traceplots with the autocorrelations or crosscorrelations. All the material is in a single file and there is no need to go back and forth interactively.

6. Using individual functions

We can also use the functions interactively. The parameter to be plotted is the data frame that results from the application of the `ggs()` function to the original source object that JAGS or other software produces.

6.1. Histograms, density plots and comparison of windows of the chain

Histograms with the posterior distributions can be obtained with `ggs_histogram()` (Figure 1, left). The figure combines the values of all the chains. Although it is not specifically a convergence plot, it is useful for providing a quick look on the distribution of the values and the shape of the posterior.

```
R> ggs_histogram(S)
```

Similar to the histogram, the density plot (Figure 1, middle) also allows for a quick inspection of the distribution of the posterior. However, `ggs_density()` produces overlapped density plots with different colors by chain, which allows comparing the target distribution by chains and whether each chain has converged in a similar space.

```
R> ggs_density(S)
```

Based on the idea of overlapping densities, `ggs_compare_partial` produces overlapped density plots that compare the last part of the chain (by default, the last 10% of the values, in green) with the whole chain (black) (Figure 1, right). Ideally, the initial and final parts of the chain have to be sampling in the same target distribution, so the overlapped densities should be similar. Notice that the overlapping densities belong to the same chain, so each column of the plot refers to one chain.

```
R> ggs_compare_partial(S)
```

6.2. Traceplots, running means and autocorrelations

A classical traceplot with the time series of the chain is obtained by `ggs_traceplot()` (Figure 2, left). A traceplot is an essential plot for assessing convergence and diagnosing chain problems. It shows the time series of the sampling process and the expected outcome is to produce “white noise”. In other words, one of the signals of lack of convergence is to find tendencies in the time series, so in this case the objective is to get a traceplot that looks purely random. Besides being a good tool to assess within-chain convergence, the fact that different colors are employed for each of the chains facilitates the comparison between chains.

```
R> ggs_traceplot(S)
```

In addition to the traceplots, a plot with the running mean of the chains is very useful to find within-chain convergence issues. A time series of the running mean of the chain (Figure 2, middle) is obtained by `ggs_running()`, and allows to check whether the chain is slowly or quickly approaching its target distribution. A horizontal line with the mean of the chain

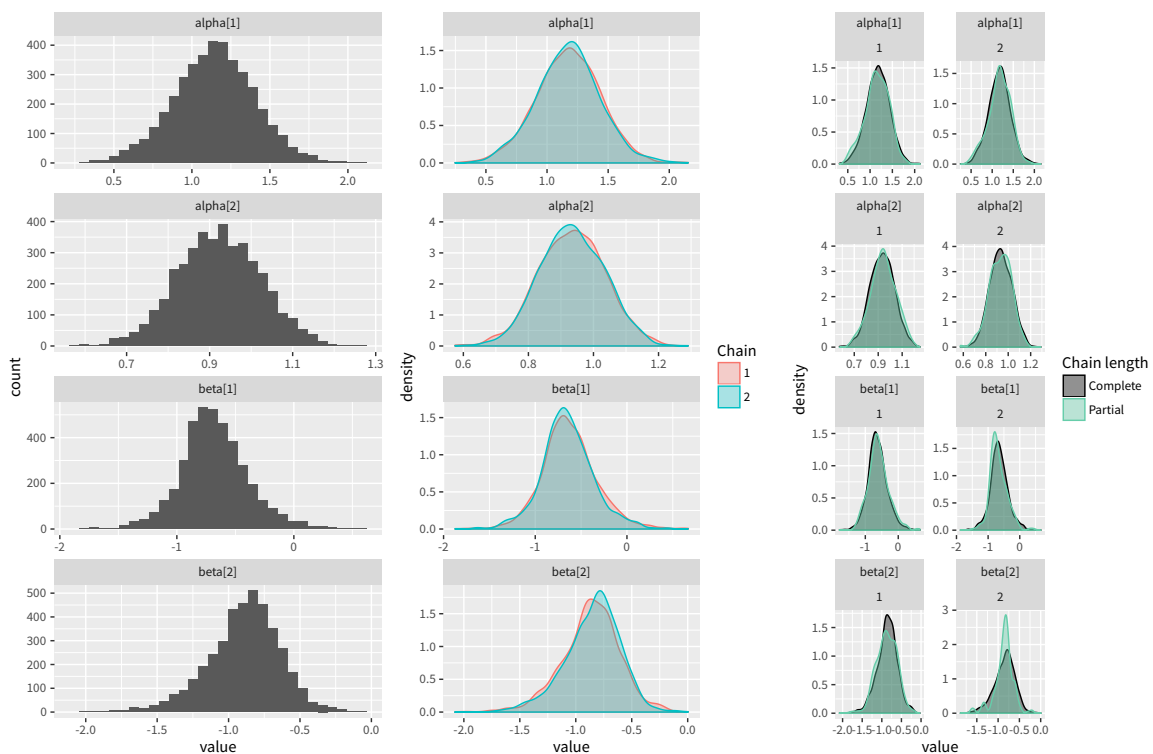


Figure 1: left) Histogram with the distribution of the posterior values combining all chains by parameter; middle) Density plots.; right) Density plots comparing the whole chain (black) with only the last part (green).

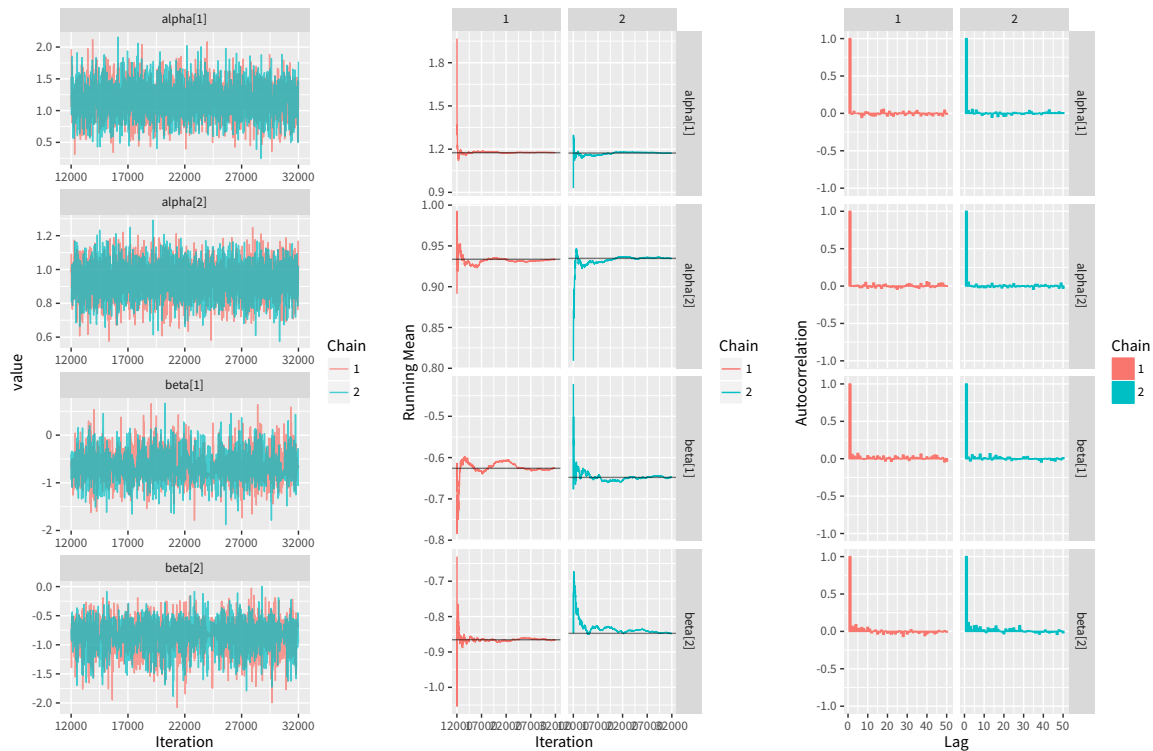


Figure 2: left) Traceplots with the time series of the chains; middle) Running means; right) Autocorrelation plots.

facilitates the comparison. Using the same scale in the vertical axis also allows comparing convergence between chains. The expected output is a line that quickly approaches the overall mean, and all chains should have the same mean (easily assessed through the comparison of the three horizontal lines).

```
R> ggs_running(S)
```

A more formal assessment of the quality of the chain can be obtained by inspecting the autocorrelation plots of the chains (Figure 2, right) using `ggs_autocorrelation()`. The expected outcome is a bar at one in the first lag, but no autocorrelation beyond the first lag. While autocorrelation is not *per se* a signal of lack of convergence, it may indicate some misbehavior in several chains or parameters, or indicate that a chain needs more time to converge. The easiest way to solve issues with autocorrelation is by thinning the chain. The thinning interval can be very easily extracted from the autocorrelation plot. By default, the autocorrelation axis is bounded between -1 and 1 , so all subplots are comparable. The argument `nLags` allows to specify the number of lags to plot, which defaults to 50.

```
R> ggs_autocorrelation(S)
```

6.3. Crosscorrelation plot

In order to diagnose potential problems of convergence due to highly correlated parameters,

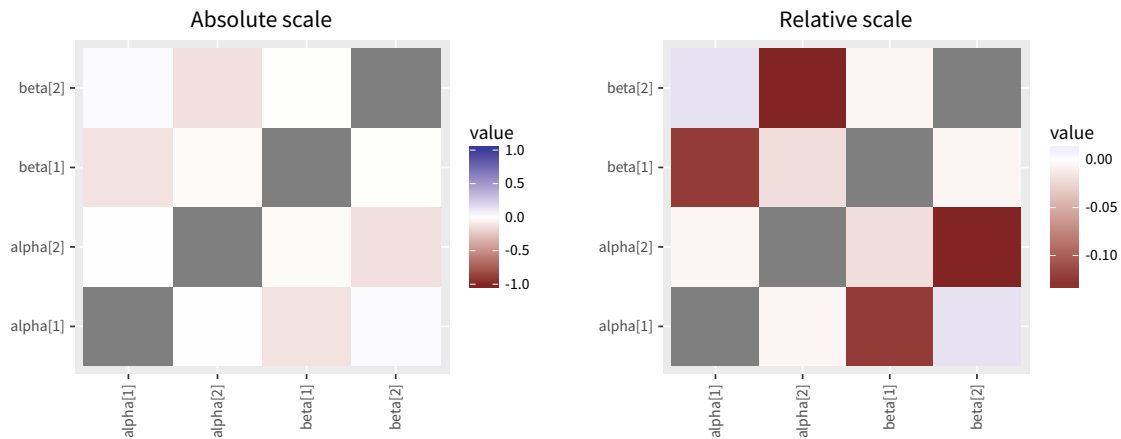


Figure 3: Tile plots with the crosscorrelations of the parameters in a relative scale (left) or absolute scale (right).

`ggs_crosscorrelation()` produces a tile plot (Figure 3, left) with the correlations between all pairs of parameters.

```
R> ggs_crosscorrelation(S)
```

The argument `absolute_scale` allows to specify whether the scale should be between -1 and $+1$ or the range of the parameter. The default is to use an absolute scale, which shows the crosscorrelation problems in overall perspective. But with cases where there is not a severe problem of crosscorrelation between parameters it may help to use relative scales in order to see the most problematic parameters.

```
R> ggs_crosscorrelation(S, absolute_scale = FALSE)
```

6.4. Formal diagnostics: Potential scale reduction factor (\hat{R}) and Geweke

Amongst the formal diagnostics of chain convergence, `ggmcmc` provides the Potential scale reduction factor and the Geweke diagnostic.

The Potential scale reduction factor (\hat{R} , ?, p. 296–297) relies on different chains for the same parameter, by comparing the between-chain variation with the within-chain variation. It is expected to be close to 1. A dotplot with its values (Figure 4, left) is produced with the `ggs_Rhat()` function.

```
R> ggs_Rhat(S)
```

By default, the function scales the axis so that there is an upper limit at 1.5 (or higher, if higher values are present), to contextualize the results. The argument `scaling` allows to change or eliminate this default (by setting it to `NA`).

By contrast the Geweke z-score diagnostic (?) focuses on the comparison of the first part of the chain with its last part. It is in fact a frequentist comparison of means, and the expected

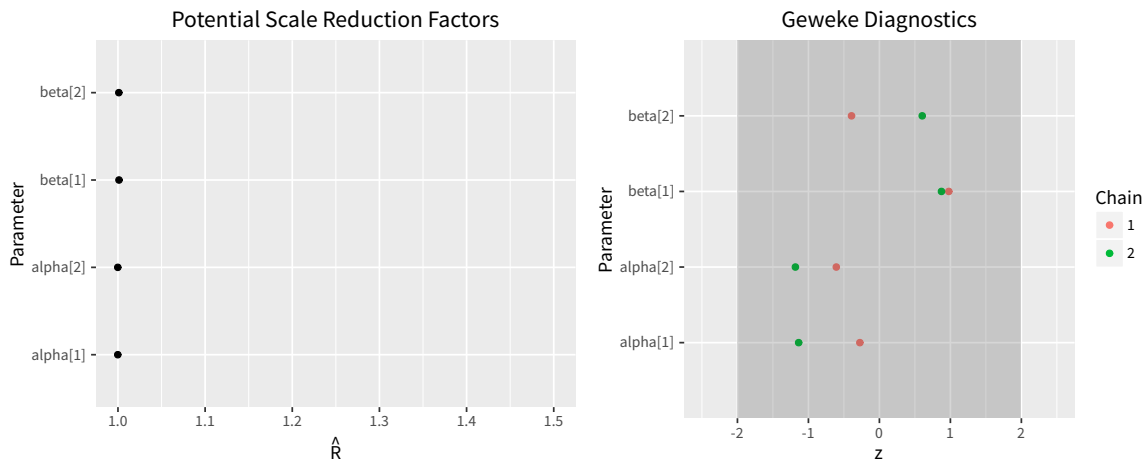


Figure 4: left) Dotplot with the Potential Scale Reduction Factor; right) Dotplot with the Geweke z-scores. For this model there is no evidence of non-convergence, as \hat{R} are very close to 1 and z-scores are between -2 and 2.

outcome is to have 95 percent of the values between -2 and 2 (Figure 4, right) It can be obtained with the `ggs_geweke()` function. By default, the area between -2 and 2 is shadowed for a quicker inspection of problematic chains.

```
R> ggs_geweke(S)
```

The overall idea is to provide multiple tools, so that a combination of them gives a precise indication of lack of convergence and, more importantly, provides hints on how to fix it.

7. Beyond basic options

In order to illustrate more advanced features of `ggmcmc`, the object with the full posterior densities of α and β will be used, instead of the short version used so far.

```
R> S.full <- ggs(radon$s.radon)
```

7.1. Select a family of parameters

Suppose that the object with the results of a model contains several families of parameters (say, `beta`, `theta`, `sigma`) and that only one of the families you want to use the previous functions but only with one of the families. The argument `family` can be used to tell `ggmcmc` to use only certain parameters. Figure 5 (left) shows a density plot similar to Figure 1 (middle), but with only the parameters that belong to the `sigma` family:

```
R> ggs_density(S.full, family = "sigma")
```

The character string provided to `family` can be any R regular expression. In cases of having multidimensional arrays of parameters (say, `theta[1,1]` or `theta[10, 5]`), the elements in

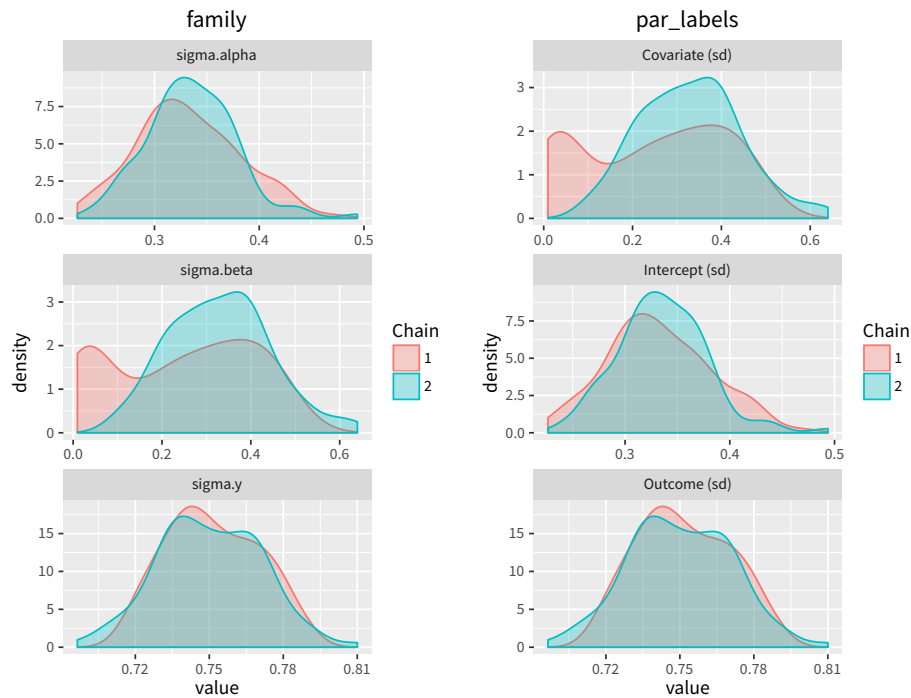


Figure 5: left) Density plot with a restricted number of parameters controlled by the argument `family`; right) Density plot with the parameters having different labels using the argument `par_labels`. Notice that the first two parameters in each column show signals of non-convergence.

the 4th row can be plotted using `family = "theta\\[4,\\.\\]"`. Regular expressions are a very powerful set of tools to manage the names of the parameters. Covering the full range of options provided by regular expressions is out of the scope of this article. For most of the cases in the context that `ggmcmc` is used the only issues to remind are the double-backslash in front of the the square bracket that indicates the dimension of a family of parameters, and the use of the *hat* ($\hat{\ }$) to distinguish between $\hat{\theta}$ and `sigma.theta`.

7.2. Change parameter labels

By default, `ggs` objects use the parameter names provided by the MCMC software. But it is possible to change it when treating the samples with the `ggs()` function using the argument `par_labels`. `par_labels` requires a data frame with at least two columns. One named *Parameter* with the corresponding names of the parameters that are to be changed and another named *Label* with the new parameter labels (Figure 5, right).

```
R> P <- data.frame(
+   Parameter = c("sigma.alpha", "sigma.beta", "sigma.y"),
+   Label = c("Intercept (sd)", "Covariate (sd)", "Outcome (sd)"))
R> ggs_density(ggs(radon$s.radon, par_labels = P, family = "sigma"))
```

The combination of the arguments `family` and `par_labels` gives high control over the final

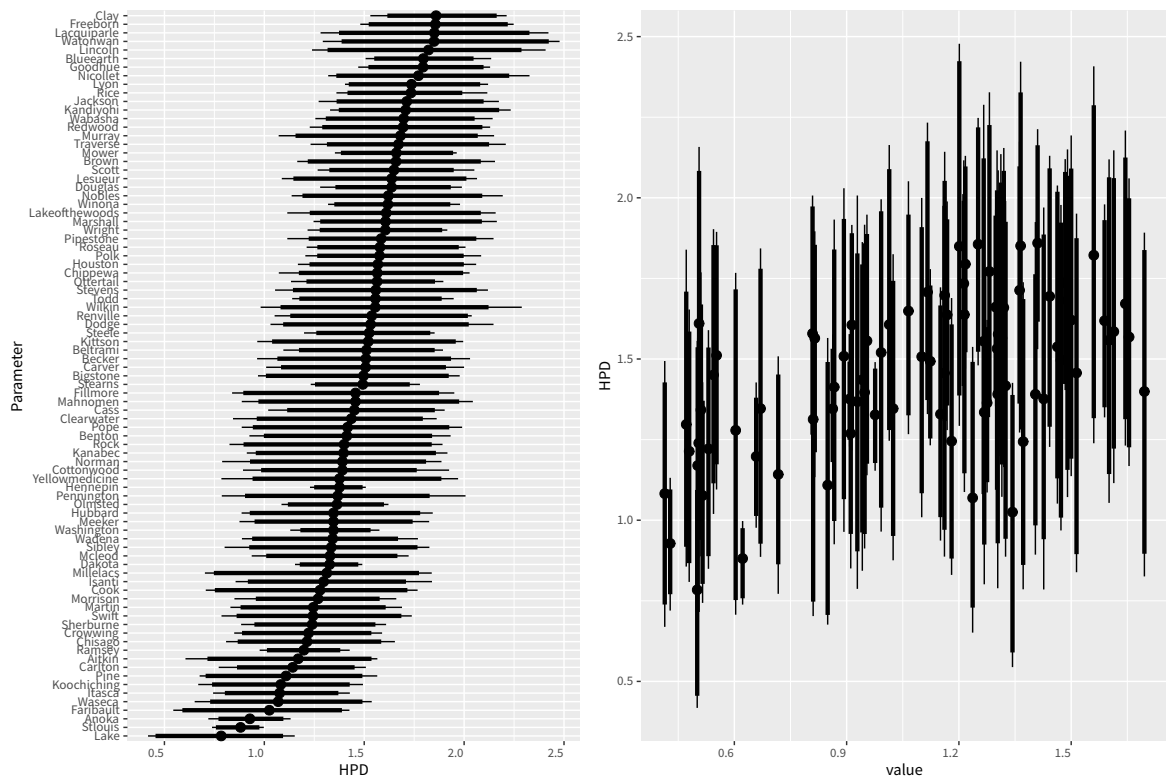


Figure 6: Caterpillar plot for the varying intercepts. Dots represent medians, and thick and thin lines represent 90 and the 95% of the Highest Posterior Density regions, respectively. left) Plain caterpillar plot with labels of the parameters, showing Clay and Lake as the counties with highest and lowest α intercept, respectively; right) Caterpillar plot against continuous values (uranium levels) using the argument `par_labels`, suggesting a tendency of higher α intercepts with increasing uranium levels.

plots, and substantially aid interpretation. However, it must be noticed that selecting a family of parameters can be done either when converting the output of the MCMC software using the `ggs()` function, or inside the `ggs_*()` individual functions. But using the `par_labels` argument is only done through the `ggs()` call.

8. Caterpillar plots

Caterpillar plots provide a sense of the distribution of the parameters, using their name as a label. `ggs_caterpillar()` produces caterpillar plots of the highest posterior densities (HPD) of the parameters produces. By default thick lines represent 90% of the HPD and thin lines represent 95% of the HPD.

The following code creates a data set with the matches between the parameter names for the intercepts (`alpha[1]:alpha[85]`) and their substantial meaning (labels for counties) and then it passes this data frame as an argument to the `ggs()` function that converts the original samples in JAGS into a proper object ready for being used by all `ggs_*()` functions.

```
R> L.radon.intercepts <- data.frame(
+   Parameter = paste("alpha[", radon$counties$id.county, "]", sep = ""),
+   Label = radon$counties$County)
R> head(L.radon.intercepts)

##   Parameter   Label
## 1  alpha[1]   Aitkin
## 2  alpha[2]   Anoka
## 3  alpha[3]   Becker
## 4  alpha[4]  Beltrami
## 5  alpha[5]   Benton
## 6  alpha[6]  Bigstone

R> S.full <- ggs(radon$s.radon,
+   par_labels = L.radon.intercepts, family = "^alpha")
```

Figure 6 (left) shows `ggs_caterpillar()` with the previously created `ggs()` object containing the varying intercepts and labeled with parameter names, and is produced by:

```
R> ggs_caterpillar(S.full)
```

`ggs_caterpillar()` can also be used to plot continuous variables. This feature is very useful when plotting the varying slopes or intercepts of several groups in multilevel modeling against a continuous feature of such groups.

Plot continuous variables by passing a data frame (`X`) with two columns. One column with the `Parameter` name and the other with its `value`. Notice that when used against a continuous variable it is more convenient to use vertical lines instead of the default horizontal lines (Figure 6, right).

```
R> Z <- data.frame(
+   Parameter = paste("alpha[", radon$counties$id.county, "]", sep = ""),
+   value = radon$counties$uranium)
R> ggs_caterpillar(ggs(radon$s.radon, family = "^alpha"),
+   X = Z, horizontal = FALSE)
```

Another feature of caterpillar plots is the possibility to plot two different models, and be able to easily compare between them. A list of two `ggs()` objects must be provided.

It is also worth mentioning that `ggs_caterpillar()` uses the function `ci()` to calculate the credible intervals (see a concrete example in Section 10.2).

9. Posterior predictive checks and model fit

`ggmcmc` includes several functions to check model fit and perform posterior predictive checks. As to version 0.6, only outcomes in uni-dimensional vectors are allowed. The outcomes can be continuous or binary, and different functions take care of them. The main input for the

functions is also a `ggs` object, but in this case it must only contain the predicted values. \hat{y}_i are the expected values in the context of the radon example. They can also be sampled and converted into a `ggs` object using the argument `family`.

In order to illustrate posterior predictive checks, samples from a faked dataset will be used. `ggmcmc` contains samples from a linear model with an intercept and a covariate (object `s`), where `s.y.rep` are posterior predictive samples from a dataset replicated from the original, but without the original outcome (y).

```
R> data("linear")
R> S.y.rep <- ggs(s.y.rep)
R> y.observed <- y
```

9.1. Continuous outcomes

For continuous outcomes, `ggs_ppmean()` (posterior predictive means) presents a histogram with the means of the posterior predictive values at each iteration, and a vertical line showing the location of the sample mean. This allows comparing the sample mean with the means of the posterior and check if there are substantial deviances (Figure 7, right).

```
R> ggs_ppmean(S.y.rep, outcome = y.observed)
```

`ggs_ppsd()` (posterior predictive standard deviations) is the same idea as `ggs_ppmean()` but with standard deviations (Figure 7, left).

```
R> ggs_ppsd(S.y.rep, outcome = y.observed)
```

9.2. Binary outcomes

In order to illustrate the functions suitable for binary outcomes a new dataset must be simulated, and samples of parameters from a simple logistic regression model must be obtained. `ggmcmc` also contains samples from such a model, in the dataset `data("binary")`. Again, arrange the samples as a `ggs` object

```
R> data("binary")
R> S.binary <- ggs(s.binary, family = "mu")
```

The ROC (receiver operating characteristic) curve is shown in Figure 8:

```
R> ggs_rocplot(S.binary, outcome = y.binary)
```

The `ggs_rocplot()` is not fully Bayesian by default. This means that the predicted probabilities by observation are reduced to their median values across iterations from the beginning. Only information relative to the chain is kept. If a fully Bayesian version is desired, the argument `fully_bayesian = TRUE` has to be set up. Use it with care, because it is very demanding in terms of CPU and memory.

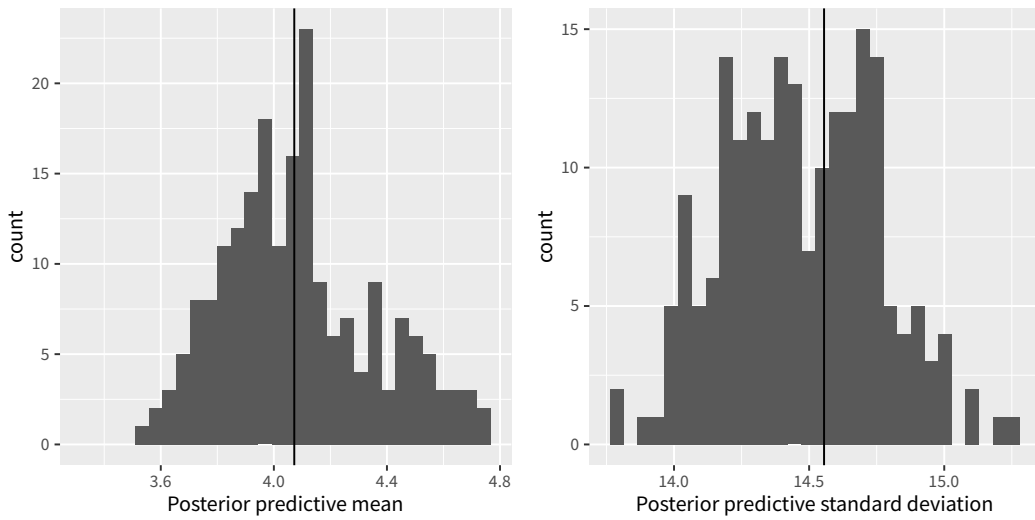


Figure 7: Histograms of the posterior predictive distributions of the mean (left) and standard deviation (right) of the replicated datasets, against the vertical lines showing the mean and standard deviations of the data, respectively.

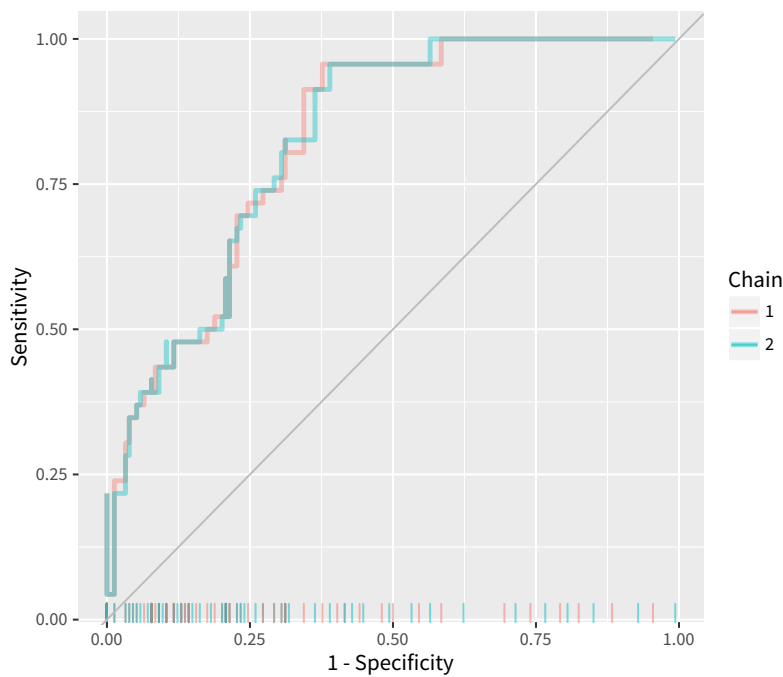


Figure 8: ROC (receiver operating characteristic) curve.

The separation plot (?) is obtained by `ggs_separation()`. The separation plot (Figure 9) conveys information useful to assess goodness of fit of a model with binary outcomes (and also with ordered categories). The horizontal axis orders the values by increasing predicted probability. The observed successes (ones) have a darker color than observed failures (or zeros). Therefore, a perfect model would have the lighter colors in the right hand side,

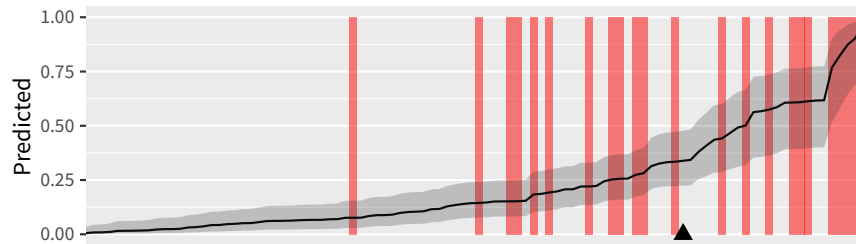



Figure 9: Separation plot.

separated from darker colors. The black horizontal line represents the predicted probabilities of success for each of the observations, which allows to easily evaluate whether there is a strong or a weak separation between successes and failures predicted by the model. Lastly, a triangle on the lower side marks the expected number of total successes (events) predicted by the model.

```
R> ggs_separation(S.binary, outcome = y.binary)
```

A minimalist version of the separation plot to be used inline () is also available with:

```
R> ggs_separation(S.binary, outcome = y.binary, minimal = TRUE)
```

10. Using ggplot2 with ggmcmc

10.1. Aesthetic variations

The combination of **ggmcmc** with the package **ggthemes** allows using pre-specified **ggplot2** themes, like **theme_tufte** (that is based on a minimal ink principle by ?), **theme_economist** (that replicates the aesthetic appearance of *The Economist* magazine), or **thema_stata** (that produces Stata graph schemes), amongst others.

```
R> library("gridExtra")
R> library("ggthemes")
R> f1 <- ggs_traceplot(ggs(s, family = "^beta\\[[1234]\\]")) +
+   theme_tufte()
R> f2 <- ggs_density(ggs(s, family = "^beta\\[[1234]\\]")) +
+   theme_solarized(light = FALSE)
R> grid.arrange(f1, f2, ncol = 2, nrow = 1)
```

In addition to **ggthemes**, **ggmcmc** can also work well with **gridExtra**, a package that allows combining several **ggplot** objects in a single layout.

10.2. Beyond default plots

Once the sampled values are stored in a tidy way in the `ggs()` object, it is trivial to reshape the object in order to extend the capabilities of the package. The following example shows how to extract the medians of the posteriors of intercepts and slopes so that a simple scatterplot to check their correlation can be produced using plain **ggplot2** functions. First, extract the credible intervals (including the median) of the parameters that start with `alpha` or `beta`, and using the pipe operator, keep only the variables `Parameter` and `median`.

```
R> ci.median <- ci(ggs(radon$s.radon, family = "^alpha|^beta")) %>%
+   dplyr::select(Parameter, median)
```

Then, generate a data frame that maps the parameter names with their labels and also specifies whether the parameter is an intercept or a slope.

```
R> L.radon <- data.frame(
+   Parameter = c(
+     paste("alpha[", radon$counties$id.county, "]", sep = ""),
+     paste("beta[", radon$counties$id.county, "]", sep = "")),
+   Label = rep(radon$counties$County, 2),
+   Uranium = rep(radon$counties$uranium, 2),
```

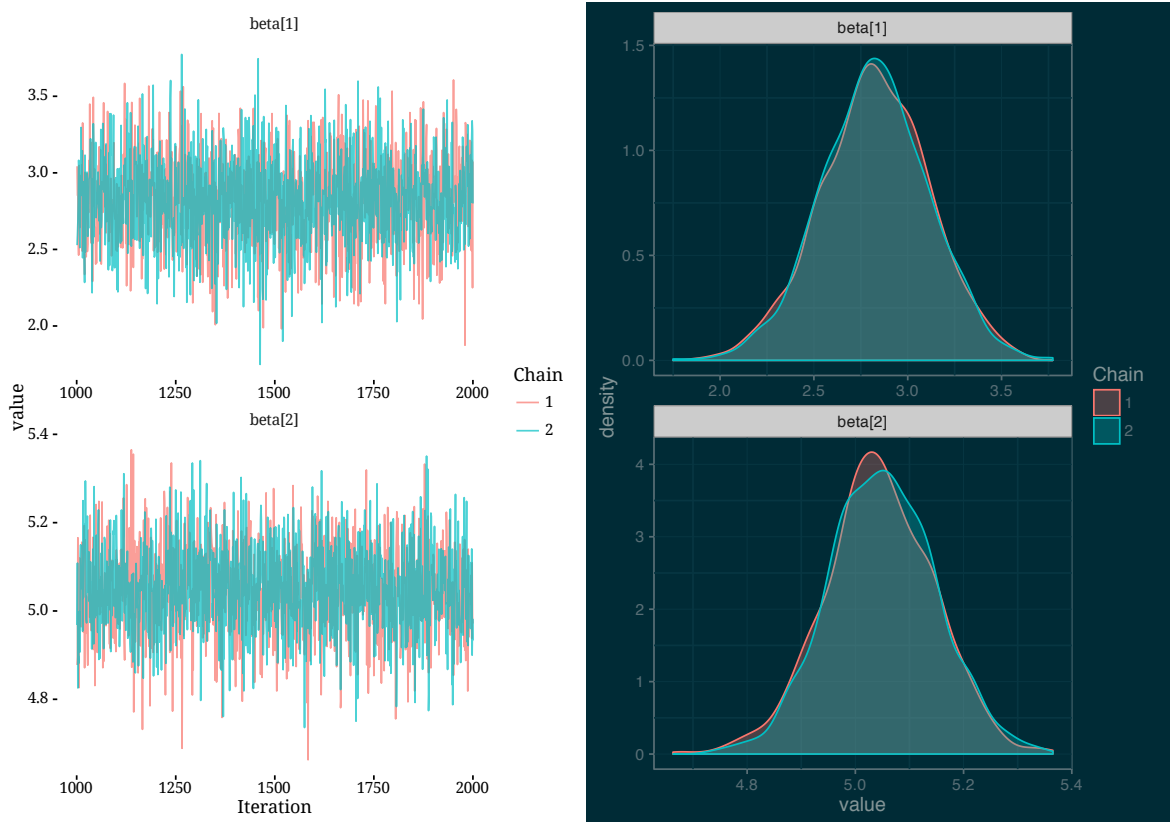


Figure 10: Combination of the aesthetically-driven options that complement **ggplot2**: **ggthemes** and **gridExtra**.

```
+   Location = rep(radon$counties$ns.location, 2),
+   Coefficient = gl(2, length(radon$counties$id.county),
+     labels = c("Intercept", "Slope"))))
```

Finally, merge the medians and the parameter names, reshaping the resulting object into a wide format.

```
R> radon.median <- left_join(ci.median, L.radon) %>%
+   dplyr::select(Label, Coefficient, median) %>%
+   tidyr::spread(Coefficient, median)
R> head(radon.median)
```

```
## Source: local data frame [6 x 3]
##
##   Label Intercept      Slope
##   <fctr>      <dbl>      <dbl>
## 1  Aitkin  1.1695607 -0.6469183
## 2   Anoka  0.9277764 -0.7848758
## 3   Becker 1.5085158 -0.6660689
## 4 Beltrami 1.5113049 -0.6894470
## 5   Benton 1.4130455 -0.6382200
## 6 Bigstone 1.4965704 -0.7078009
```

The resulting object can be very easily passed to standard **ggplot2** functions. In the following case (Figure 11), a scatterplot with the medians of the posteriors of intercepts and slopes is produced. A data frame (`radon.median`) is provided to the main **ggplot2** function (`ggplot`) along with a description of the aesthetic elements (x and y axis), with the geometrical element desired (`points`) added later on (`+ geom_point()`).

```
R> ggplot(radon.median, aes(x = Intercept, y = Slope)) + geom_point()
```

10.3. Beyond default options

```
R> ggs_caterpillar(ggs(radon$s.radon, par_labels = L.radon, family = "^alpha")) +
+   facet_wrap(~ Location, scales = "free") +
+   aes(color = Uranium)
```

Another option is to extend `ggs_caterpillar()` using facets with variables passed through `par_labels`, or adding other variables to the aesthetics. Figure 12 shows a caterpillar plot of the intercepts, with facets on the North/South location and using a color scale to emphasize the uranium level by county.

11. Concluding remarks

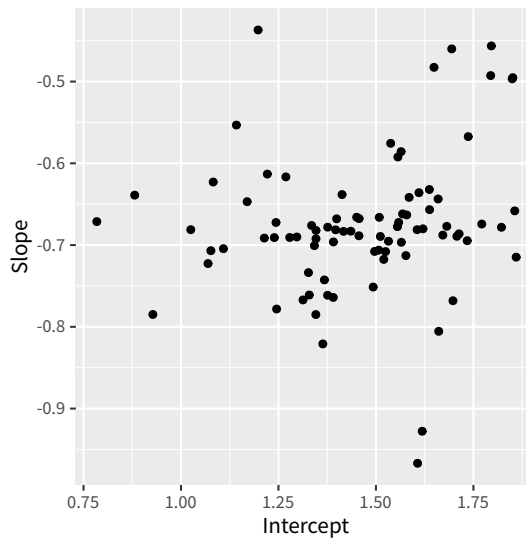


Figure 11: Scatterplot with the medians of the posteriors of intercepts and slopes for the radon data example.

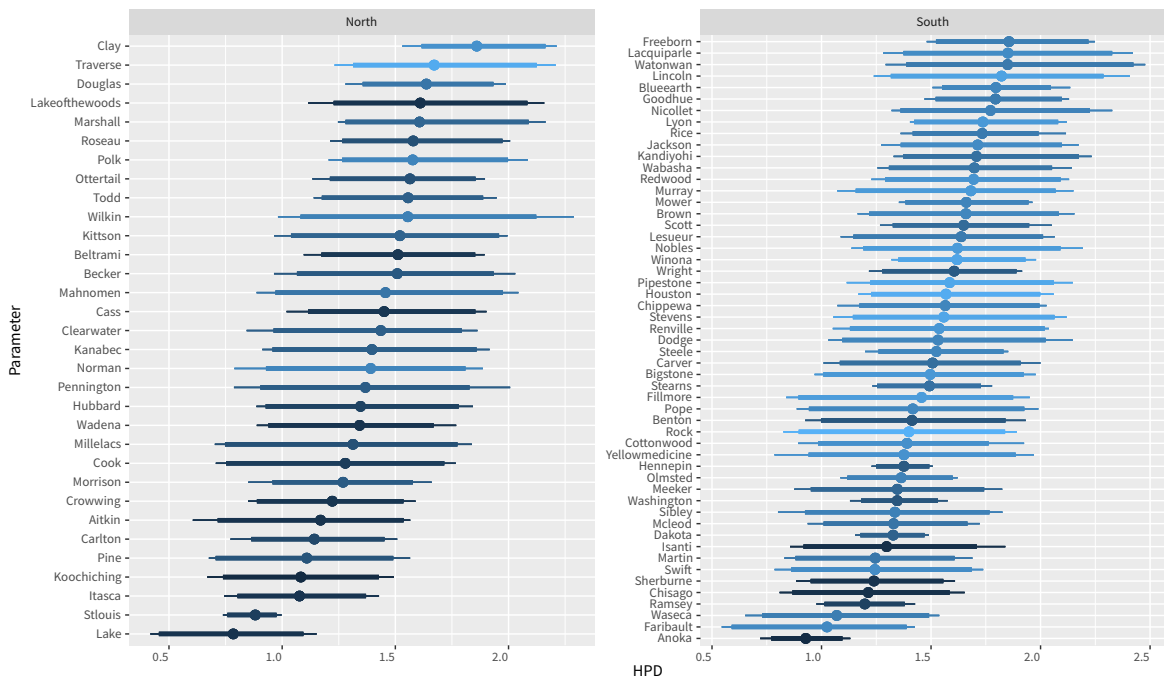


Figure 12: Caterpillar plot of the varying intercepts faceted by North/South location and using county’s uranium level as color indicator.

The article has provided an introduction to the potential uses of **ggmcmc** for analyzing MCMC samples from Bayesian inference. We have covered the basic principles of the package, based on the transformation of samples into a *tidy* version, a data frame. We have also shown the functions used for classical convergence tests, as well as the use of caterpillar plots for substantial interpretation of the results. Finally, we have briefly introduced potential extensions

to the package by combining it with other tools from the **ggplot2** ecosystem.

Acknowledgments

I would like to thank code contributions and ideas from Zachary M. Jones (<http://www.zmjones.com/>), Julien Cornebise (<http://www.cornebise.com/julien/>), and ideas from Dieter Menne (<http://www.menne-biomed.de/>), Bill Dixon, Christopher Gandrud (<http://christophergandrud.blogspot.com>), Maxwell B. Joseph (<http://mbjoseph.github.io/about/>), Barret Schloerke and GitHub users knokknok, akrawitz, Justina (Juste) and stefanherzog.

Affiliation:

Xavier Fernández-i-Marín
ESADE Business School
Universitat Ramon Llull
Av. Torre Blanca, 59
08172 Sant Cugat del Vallès, Spain
E-mail: xavier.fernandez3@esade.edu
URL: <http://xavier-fim.net>