Project in STAT 547

University of British Colombia

Compute Graph Gibbs Sampling for GLMs in R

Author: Mathias Lerbech Jeppesen, Julie Køhler Munkvad

We share equal responsibility for all parts of the project, including the code.

Handed in: December 11, 2024

Compute Graph Gibbs Sampling for GLMs in R

Mathias Lerbech Jeppesen, Julie Køhler Munkvad

December 10, 2024

ABSTRACT

Bayesian statistics offers a paradigm shift from traditional frequentist approaches by treating parameters as having probability distributions rather than fixed values. This shift necessitates efficient methods for approximating the posterior distributions of parameters, often achieved through Markov Chain Monte Carlo (MCMC) sampling. The computational demands of MCMC methods constantly drive innovation of novel methods to increase efficiency.

This paper introduces the R package **mcmcglm**, which implements the Compute Graph Gibbs (CGGibbs) sampler, a method proposed by Luu et al. (2024) [10]. The CGGibbs sampler improves computational efficiency by leveraging incremental updates to the linear predictor during Gibbs sampling for generalized linear models, reducing complexity to linear time with respect to the number of parameters. The package extends the familiar interface of the `glm` function to the Bayesian setting with flexibility to specify any response family and prior distribution, leveraging slice sampling within Gibbs to not rely on conjugacy of priors.

This paper provides some context for the package in form of an introduction to Bayesian inference and the MCMC method of Gibbs sampling, specifically combined with slice sampling and the CGGibbs algorithm. The paper then demonstrate the usage of **mcmcglm**, showcasing the flexibility of the interface and linear runtime of the algorithm, though overall computational efficiency needs to be improved. The paper ends with a discussion around the package's advantages and disadvantages, potential applications and future implementations.

Contents

1 Introduction

Bayesian statistics is a huge field within statistics where the "traditional" idea of frequentist statistics, where we seek to infer the "true" value of a deterministic parameter, is exchanged with the idea that a target parameter has a "true" distribution rather than a "true" value. Thus, many methods seek to draw samples from a posterior target parameter's distribution to obtain an empirical distribution as an approximation. One way of doing this is using Markov Chain Monte Carlo (MCMC) methods which create a Markov chain with stationary distribution equal to the "target" distribution. Potentially, many samples are needed to "reach" the stationary distribution and draw enough samples from the stationary distribution to get a good approximation from the empirical distribution.

To improve computational efficiency and speed, researchers continuously develop new MCMC methods. Recently, the *Compute Graph Gibbs* (CGGibbs) sampler was introduced in Luu et. al (2024) [10]. The method is linear in computational speed as a function of the number of parameters in a generalized linear model (GLM), reducing computational complexity compared to alternative methods. As stated in the article, *"Generalized linear models (GLMs) are among the most commonly used tools in contemporary Bayesian statistics."* and methods for optimising computational speed of bayesian GLMs would be useful to a wide field of applications.

There exists other R packages such as the **arm** package [7] with functions like the `bayesglm` function that use expectation maximisation (EM) to update parameter values. However, we were unable to find any existing R package with an implementation of the CGGibbs sampler which motivated an implementation available to R programmers.

In this paper, we introduce an R package **mcmcglm** with an implementation of the CGGibbs sampler. The package implements this method using an interface that will be known to users of the "frequentist" `glm` function with the same arguments but with an addition of specification of a prior distribution of the parameters. The implementation supports any family specification, similar to the standard `glm` function. Additionally, to avoid relying on the conjugacy of priors, the package employs slice sampling. This approach allows users to specify any prior distribution, with the default slice sampling procedure described in Neal (2003) [13]. However, other slice samplers are also available.

The paper will first give a bit of introduction to Bayesian inference and MCMC sampling to provide some knowledge and context around the method and will then dive into more detail around the CGGibbs method and it's implementation in the package. The reader is then given an introduction to usage of the package, showcasing the specification options available and showing outputs created by the package. The paper ends with discussing advantages of the package but focusing on shortcomings and resulting scope of future development.

2 Bayesian Inference

In this section we aim to introduce a bit of terminology and context needed to understand the methods that are utilised by the package we will later describe in this paper. The section is based on [4], [2] and [9].

Note that we in the following use the terms distribution and density interchangeably and that p is used without subscript to denote the density of different random variables, letting the inputs provide context as to what distribution is being referenced.

Frequentist approach to statistical inference often seeks to maximise a likelihood function $p(y|\theta)$ by defining probability distributions $\mathbb{P}_\theta$ for all possible parameter values $\theta \in \Theta$ in some parameter space Θ. In this situation, the philosophy of maximising the likelihood function (as a function of θ, given data y) is to estimate our parameter θ by taking the parameter value that has the highest likelihood of producing the realisation of the data y we have. In bayesian statistics, however, we view the parameter θ as a stochastic entity and examine its distribution rather than creating separate probability distributions for ("potentially true") deterministic values of θ. This leads to the notion of probability intervals in the context of inference of θ rather than the confidence intervals of frequentist statistics.

Thus, put in a simplified manner, our goal is to find the distribution of the random vector θ given data y, that is $p(\theta|y)$. As is the case with most problems, this distribution is not known. In order to model this, we formulate a probability model for the joint distribution of y and θ, $p(y, \theta)$. From Bayes' theorem, we then have our "target" distribution as

$$p(\theta|y) = \frac{p(y, \theta)}{p(y)} \tag{2.1}$$

where we with another use of Bayes' theorem can write this as

$$p(\theta|y) = \frac{p(y|\theta)p(\theta)}{p(y)} \propto p(y|\theta)p(\theta) \tag{2.2}$$

Above we have the "heart" of Bayesian inference, denoting by $p(\theta|y)$ the *posterior distribution*, by $p(y|\theta)$ the *likelihood*, by $p(\theta)$ the *prior*, and $p(y)$ is often just referred to as the *normalizing constant*. In this way, we model the posterior distribution (of interest) using a likelihood that is specified by a statistical model and prior beliefs/expectations/evidence of the parameter θ.

In practice, Bayesian inference is often performed by iteratively sampling from the posterior distribution. Common are MCMC methods where a Markov chain is constructed in a way that the stationary distribution is the posterior distribution. One way of doing this is with gibbs sampling.

3 Gibbs Sampling

Gibbs sampling is a Markov Chain Monte Carlo (MCMC) method used to generate samples from a joint probability distribution $p(\theta|y)$. Instead of sampling directly from $p(\theta|y)$, Gibbs sampling sequentially samples from the conditional distributions

$$\theta_j \sim p(\theta_j|\theta_{-j}, y) \tag{3.1}$$

where $\theta_{-j} = (\theta_1, ..., \theta_{j-1}, \theta_{j+1}, ..., \theta_d)$ denotes the vector of all entries in θ despite θ_j.

By denoting $\theta^{(k)}$ as the value of θ after the k-th iteration, the sequence $\{\theta^{(k)}\}_{k=1}^{K}$ forms a Markov chain with the posterior $p(\theta|y)$ as its stationary distribution. After sufficient iterations, the samples $\theta^{(k)}$ can be treated as approximate draws from the posterior. Thus, this iterative process guarantees that the samples will asymptotically converge to $p(\theta|y)$ [17].

3.1 Compute Graph Gibbs (CGGibbs)

The Compute Graph Gibbs (CGGibbs) sampler, as introduced by Luu et al. (2024) [10], is an optimized approach to Gibbs sampling that focuses on efficiently updating the linear predictor in GLMs. When a closed-form expression of the distribution in (3.1) is unavailable, the linear predictor must be calculated to determine the log-potential, which is then used by various methods to sample from the posterior distribution. Instead of recalculating the linear predictor entirely for each Gibbs update, CGGibbs incrementally adjusts the predictor for the parameter being updated. This significantly reduces computational overhead while maintaining accuracy, especially noticeable for high-dimensional models.

In GLMs, the linear predictor η determines the mean of the response variable through a link function $g(\cdot)$, typically

$$\mathbb{E}[Y|X, \beta] = \mu = g^{-1}(\eta) = g^{-1}(X\beta).$$

In the k-th iteration of Gibbs sampling, after sampling the j-th coordinate of the d-dimensional parameter vector β, the updated vector is

$$\beta^{(k)(j)} = (\beta_1^{(k)}, \beta_2^{(k)}, \ldots, \beta_j^{(k)}, \beta_{j+1}^{(k-1)}, \ldots, \beta_d^{(k-1)}),$$

and so we denote the linear predictor after the j-th coordinate update in the k-th iteration by

$$\eta^{(k)(j)} = X\beta^{(k)(j)}$$

and by $\eta_i^{(k)(j)}$ the linear predictor for the i'th observation in the data, Y_i.

The CGGibbs approach calculates the linear predictor for observation i after the j-th coordinate update in the k-th iteration as

$$\eta_i^{(k)(j)} = \eta_i^{(k)(j-1)} + x_{ij}\left(\beta_j^{(k)} - \beta_j^{(k-1)}\right), \tag{3.2}$$

where x_{ij} is the i-th observation for the j-th parameter. This adjustment avoids a full recomputation of η_i, leveraging the fact that only one component of β changes per iteration.

In contrast, the *naive approach* recalculates the entire predictor from scratch:

$$\eta_i^{(k)(j)} = \sum_{l=1}^{d} x_{il}\beta_l^{(k-1)}. \tag{3.3}$$

This naive computation requires $O(dn)$ operations for each coordinate update of β, where n is the number of data points and d is the number of parameters, resulting in a computational complexity for each "full" update of β of $O(d^2 n)$. CGGibbs reduces this to $O(dn)$, providing a significant performance advantage, especially for high-dimensional models.

3.2 Slice Sampling within Gibbs Sampling

Slice sampling, integrated within the Gibbs sampling framework, is used for updating coordinates for β in cases where direct sampling from the conditional distributions in (3.1) is computationally challenging, for example, when closed-form distributions are not available. Slice sampling avoids the need for normalization constants, making it particularly useful in complex models.

The linear predictor η_i, updated using CGGibbs, is central to computing the *log-potential* given as

$$\text{log-potential} = \log p(y|\beta) + \log p(\beta)$$

Since slice sampling only requires the posterior up to a normalizing constant, it avoids the need for computationally expensive numerical integration. Techniques such as slice sampling with *stepping-out* and *shrinkage* as introduced in Neal (2003) [13] enhance efficiency by adaptively searching the sampling region.

Combining slice sampling with CGGibbs allows for flexible modeling in Bayesian GLMs, as it removes the reliance on conjugacy assumptions and supports complex prior and likelihood combinations. This combination of CGGibbs and slice sampling illustrates a powerful framework for high-dimensional Bayesian inference, particularly in GLMs. By reducing computational complexity and enabling efficient sampling, these techniques make scalable Bayesian modeling more practical.

4 mcmcglm Package

The `mcmcglm` package implements the CGGibbs sampler which, as stated in Section 3.1, has linear run time as a function of the number of parameters in a GLM model. The package is implemented in such a way that the user can specify any response family and any distribution for the prior of the β parameter to provide total flexibility.

Installation is available from GitHub with:

```r
devtools::install_github("mathiaslj/mcmcglm")
```

Documentation of the package is available in your IDE when having installed the package, but also a pkgdown is published to github pages.

In this Section, we demonstrate how to use the package by introducing its basic interface and providing examples of how to specify different families and priors. We also show how users can easily extend the package to include custom families that are not natively implemented. Finally, we discuss the package's performance by comparing the CGGibbs method to the "naive" calculation of the linear predictor and verifying whether the implementation achieves linear runtime with respect to the number of model parameters.[1]

4.0.1 Example Data

To show that the package provides samples from the target posterior distribution, we simulate data for each scenario with known β vector in order to demonstrate that the empirical distribution of each coefficient has mean equal to the value we simulated data from. This is apparent from the trace plots that for most examples are located in the Appendix and are referenced in each example.

We start by generating a linear predictor $\eta = X\beta$ from known values of β and sampled values that make up the design matrix X. Then, in each example, we can apply the relevant inverse link function g^{-1} to obtain the modeled mean $\mu = g^{-1}(X\beta)$ in the GLM model. We can then simulate the response as samples from the distribution corresponding to the family using the modeled mean μ as the relevant parameter of the distribution.

For all examples in this paper (expect when investigating the linear runtime as function of the number of parameters in the model - see Section 4.5), we simulate data from the model

$$\begin{aligned} \mathbb{E}[Y|X,\beta] &= g^{-1}(X\beta) \\ &= g^{-1}([X_0 \;\; X_1 \;\; X_2]\beta) \\ X_0 &\overset{iid}{\sim} \mathbf{1} \\ X_1 &\overset{iid}{\sim} N(0,1) \\ X_2 &\overset{iid}{\sim} Bernoulli(0.5) \\ \beta &= (1, 1.5, 2)^\top \end{aligned} \tag{4.1}$$

where g is the link function and $X = [X_0 \;\; X_1 \;\; X_2]$ is a 1000×3 design matrix.

4.1 Basic Usage of the Package

The use of the function `mcmcglm` is similar in interface to the `glm` function but with an added mandatory specification of:

- The prior distribution of the parameter β
  - Specified using the **distributional** [15] package; see more in Section 4.3
- Tuning parameter(s) for the slice sampling procedure

[1]The code demonstrating the usage for all examples can be found in the Appendix or in the articles on GitHub.

  - The default procedure by Neal (2003) [13] requires specifying a slice width, `w`. However, alternative slice sampling procedures are also available; see Section 4.4 for more details.

- The number of iterations and burnin
  - Defaults are `n_samples = 500` and `burnin = 100` to make examples etc. run faster. In a scenario where the results will be used for analysis, the user will usually increase the values of these parameters

4.1.1 Simple example for Gaussian Family

We simulate data to demonstrate an example usage for a Gaussian family, based on the model in (4.1) with the link function g being the identity function and assuming a standard deviation of the response of $\sigma = 1$. See code used to generate data for this example in appendix A.2.1.

Then, we can simply use the `mcmcglm` function like so:

```
norm <- mcmcglm(formula = Y ~ .,
                family = "gaussian",
                data = dat_norm,
                beta_prior = distributional::dist_normal(0, 1),
                w = 0.5)
```

This creates an `mcmcglm` object which prints as:

```
norm
```

```
#> Object of class 'mcmcglm'
#>
#> Call:  mcmcglm(formula = Y ~ ., family = "gaussian", data = dat_norm,
#>     beta_prior = distributional::dist_normal(0, 1), w = 0.5)
#>
#> Average of parameter samples:
#>   (Intercept)        X1        X2
#> 1    1.011134 1.490459 2.026047
```

The print method summarises the call of the function with averages of non-burnin samples of each parameter in the GLM model.

Investigating results

The averages shown in the print method of the object can be retrieved with the generic `coef`:

```
coef(norm)
```

```
#>   (Intercept)        X1        X2
#> 1    1.011134 1.490459 2.026047
```

Quantiles of the samples (that are not marked as burnin) are available with the `coef` method, which as a default has `probs = c(0.025, 0.5, 0.975)`, i.e. it returns the 2.5%, 50% and 97.5% quantiles

```
quantile(norm)
```

```
#>             var     mean    q_0025     q_05   q_0975
#> 1 (Intercept) 1.014583 0.8349995 1.013909 1.099346
#> 2          X1 1.457432 0.6940698 1.497910 1.569321
#> 3          X2 1.996584 1.8807500 2.024372 2.177676
```

The full data set of samples can be accessed with the `samples` function:

```
head(samples(norm))
```

```
#>   (Intercept)           X1          X2 iteration burnin
#> 1   0.6173367 -0.004541141 -0.09125636         0   TRUE
#> 2   2.6508146  0.281295470  0.68343132         1   TRUE
#> 3   0.8240996  0.324627659  2.30889073         2   TRUE
#> 4   0.8170086  1.028326905  2.20351455         3   TRUE
#> 5   0.8777350  1.592074284  2.16115289         4   TRUE
#> 6   0.9092187  1.442872350  2.02913214         5   TRUE
```

A trace plot can be seen in Figure 1, which is created with the function `trace_plot`:

```
trace_plot(norm)
```

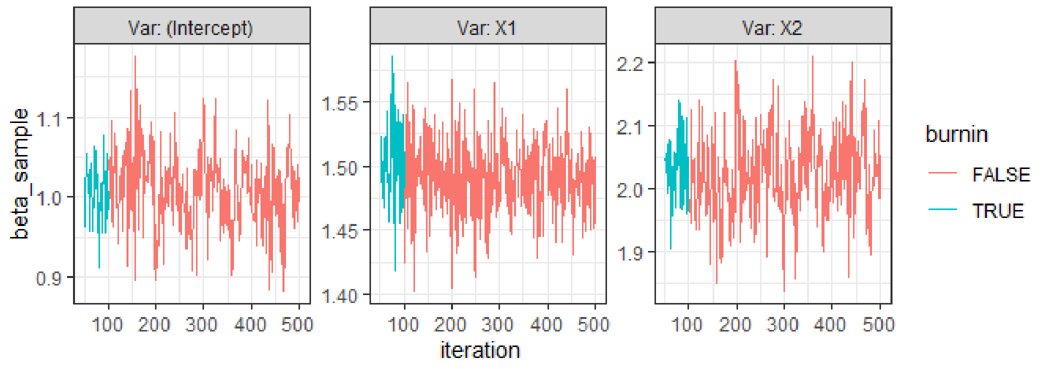

Figure 1: Trace plot with samples of components of $\beta = (\beta_0, \beta_1, \beta_2)^\top$ modelled with a Gaussian response family and standard normal prior distribution

4.2 Family of response

The family of the response can, in principle, be any family with an implemented `family` class in R. As of now, the implemented families in the package are:

- Gaussian (using function `stats::gaussian`)
- Binomial (using function `stats::binomial`)
- Poisson (using function `stats::poisson`)
- Negative binomial (using function `MASS::negative.binomial`)

Minimal effort is required from the user to enable the use of any family. The process of expanding available families is described in Section 4.2.2.

Due to the implementation's use of a `family` object, any compatible link function can be applied to a family, e.g. logit or probit for the binomial family. To further enhance user convenience, the package includes an unexported function, `check_family`, which ensures that the `family` argument in all package functions accepts values as either a `character`, a `family` function, or a `family` class object. Examples showcasing these different usages are provided throughout the examples.

4.2.1 Example for binomial family

For the binomial family, the most commonly used link function is the logit [6], given by:

$$\text{logit}(p) = \ln \frac{p}{1-p}$$

Other link functions are also available, including probit, cauchit, log, and cloglog. See more details in the documentation of the `family` class in the **stats** [16] package or other relevant documentation pages of R functions.

We'll show an example first for the binomial family with the logit link function.

Logit link function

The data generation code can be seen in Appendix A.2.2. Then the syntax for fitting an `mcmcglm` object for the binomial family with logit link function is as follows:

```r
logit <- mcmcglm(formula = Y ~ .,
                 data = dat_logit,
                 family = binomial(link = "logit"),
                 beta_prior = distributional::dist_normal(mean = 0, sd = 1),
                 w = 0.5)
```

A trace plot of the resulting object can be seen in the same Appendix as the data generation.

Probit link function

Although the logit link is the most widely used for the binomial family, the probit model, using link function equal to the inverse of the cumulative distribution function of the standard normal distribution, Φ, is also widely used [1].

Using the family classes in R, we can just specify link = "probit" as an argument to the binomial family function, and the syntax for fitting this model is then just:

```r
probit <- mcmcglm(formula = Y ~ .,
                  data = dat_probit,
                  family = binomial(link = "probit"),
                  beta_prior = distributional::dist_normal(mean = 0, sd = 1),
                  w = 0.5)
```

Data generation and a trace plot of the resulting object can be seen in Appendix A.2.3.

4.2.2 Implementation for Other Families

As mentioned at the beginning of this section, in addition to the Gaussian and binomial families demonstrated earlier, the package also supports the Poisson and negative binomial families for modeling count data. Code examples for these implementations can be found in Appendices A.2.4 and A.2.5, respectively.

Adding Methods for New Families

Extending the package to include a new family is straightforward. The only requirement needed is a method for calculating the log-density of the distribution. Since the calculation of the log-density for a family is implemented using S3 methods, the user simply needs to define a function called log_density.*family_name*, where *family_name* specifies the relevant family, in their global environment.

Below is an example showcasing the single line of code a user needs to write to enable the use of an inverse Gaussian response:

```r
log_density.inverse.gaussian <- function(family, mu, Y, ...) {
  statmod::dinvgauss(Y, mean = mu, ..., log = T)
}
```

where mu is the modeled mean $\mu = g^{-1}(X\beta)$ in the GLM model, and Y is the response variable.

Then, the user just need to specify the family using the stats::inverse.gaussian function and make sure to include any additional arguments required for the density function through the log_likelihood_extra_args argument, which in this case are shape and disperson parameters:

```r
invnorm <- mcmcglm(formula = Y ~ .,
                   family = inverse.gaussian(),
                   log_likelihood_extra_args = list(
                      shape = 1,
                      dispersion = 1),
                   data = dat_invnorm,
                   beta_prior = distributional::dist_gamma(1, 1),
                   w = 0.5)
```

Code for data generation and a trace plot is available in Appendix A.2.6.

4.3 Specifying the Prior Distribution

The `mcmcglm` package uses the **distributional** [15] package for specifying the prior distribution of the β parameter. The following examples illustrate different ways to define priors.

IID Priors

In all the previous examples, IID priors were specified by simply specifying a single univariate distribution in the `beta_prior` argument.

(Different) Independent Priors

In case the user wants to specify different priors (whether it be completely different distributions or just different moments) for the different parameters in the β vector, this can be done simply by specifying a list of priors. This is possible due to an S3 implementation of the unexported `log_prior_density` function in the backend of the package. Example code is available in Appendix A.3.1

Multivariate Prior

The **distributional** [15] package has a native implementation for the multivariate normal distribution, enabling a specification of this distribution directly using the `dist_multivariate_normal` function from the package. Example code is available in Appendix A.3.2.

4.4 Tuning (and Changing) the Sampling Procedure

The package uses the **qslice** [8] package for implementation of slice samplers. The task left to the user is to specify the tuning parameters of the chosen algorithm. The default slice sampler uses the `slice_stepping_out()` function from the **qslice** package, which is an implementation of Neal (2003) [13], which has a mandatory non-default argument `w` with the slice width of the algorithm.

All previous examples have displayed how to use the default slice sampler with a specification of the slice width in the call to `mcmcglm` and will not be repeated here.

4.4.1 Changing the default slice sampling method

As mentioned above, changing the slice sampling procedure corresponds to just providing a different function from the **qslice** package and specifying the tuning parameters. A full list of available slice samplers from the **qslice** package is available in the reference manual of the CRAN page of the package. Below is an example of using an elliptical slice sampler introduced in Murray et al. (2010) [12] with tuning parameters `mu` and `sigma`:

```
el <- mcmcglm(formula = Y ~ .,
              family = "gaussian",
              data = dat_norm,
              beta_prior = distributional::dist_normal(0, 1),
              qslice_fun = qslice::slice_elliptical,
              mu = 3,
              sigma = 2)
```

Data is the same as in Appendix A.2.1, and a trace plot is available in Appendix A.4.1.

4.4.2 Investigating effect of tuning parameters of slice sampler

The functions `mcmcglm_across_tuningparams` and `plot_mcmcglm_across_tuningparams` can be used to create trace plots from running the algorithm across different values of tuning parameters. Using the `tuning_parameter_name` argument, the user can specify for what tuning parameter to iterate across different values, meaning it can work for any tuning parameter for any slice sampling procedure.

Below is an example of varying the slide width for the default slice sampler, and in appendix A.4.2 is code syntax for investigating the effect of the `df` (degrees of freedom) for the generalized elliptical slice sampler introduced in Nishihara et al. (2014) [14].

Varying slice width for the default slice sampler

The syntax needed to perform this comparison is giving a vector (or list) with values together with a specification of the `tuning_parameter_name` argument:

```
w05_mcmcglms <- mcmcglm_across_tuningparams(
  seq(from = 0.5, by = 0.5, length.out = 4),
  tuning_parameter_name = "w",
  formula = Y ~ .,
  family = "gaussian",
  data = dat_norm,
  parallelise = FALSE)
```

The results are plotted in Figure 2 by running:

```
plot_mcmcglm_across_tuningparams(w05_mcmcglms)
```

In this example it is apparent that changing the value of the slice width does not seem to affect the empirical distribution of the parameters.

4.4.3 Normal-normal sample method

Mostly for testing purposes, the package includes an implementation for sampling from a closed-form conditional normal distribution when both the response and the prior follow a

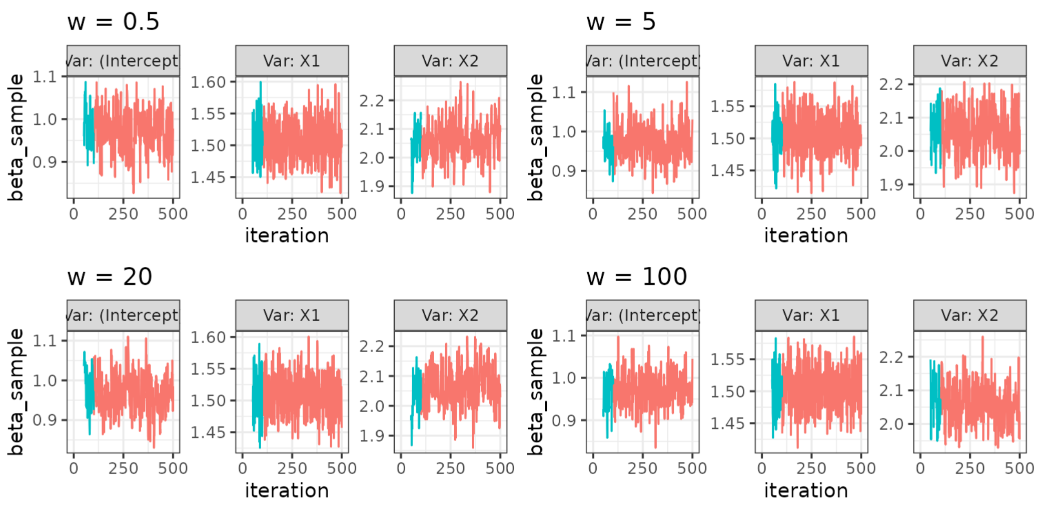

Figure 2: Trace plots of β samples using different slice widths in the default slice sample procedure from Neal (2003) [13]

normal distribution. This can be used by specifying `sample_method = "normal-normal"`. Example code syntax can be seen in appendix A.4.3

4.5 Performance of Compute Graph Gibbs vs. Naive Computation of Linear Predictor

We can use the functions `compare_eta_comptime_across_nvars` and `plot_eta_comptime` to investigate computation time of the algorithm for varying dimension of the parameter vector β. This can also be used to investigate computation time for different sample methods, different tuning parameters, or what else might be of interest to the user.

The basic usage of the function is to investigate the computational gain for high dimensions of the CGGibbs sampler with updates of the linear predictor as described in equation (3.2) compared to the naive approach of doing d computations for each coordinate update as seen in equation (3.3). To investigate the computation time of a single sweep across all coordinates, the following code is used:

```r
res <- compare_eta_comptime_across_nvars(
  n_vars = c(2, seq(from = 50, to = 500, by = 50)),
  n_samples = 1,
  burnin = 0,
  w = 0.5)
```

The results can be plotted (see Figure 3) by simply calling

```r
plot_eta_comptime(res)
```

The plot confirms that the algorithm achieves linear runtime, with a speedup compared to

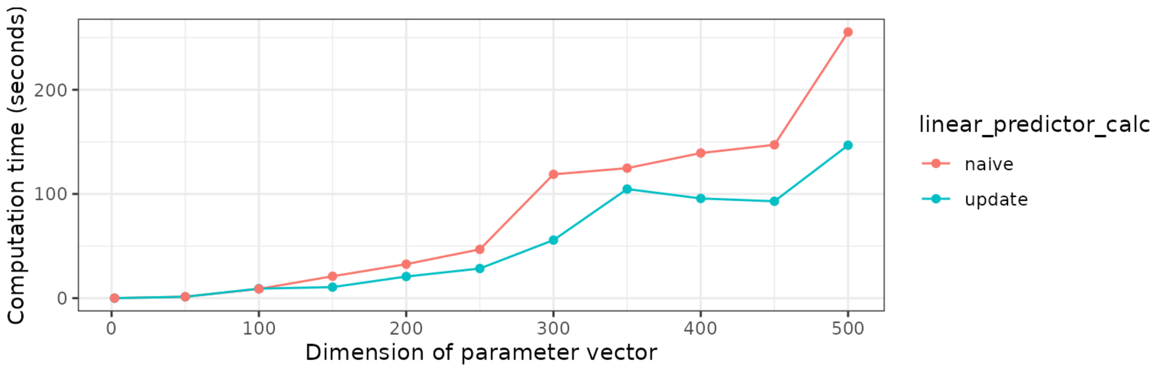

Figure 3: Plot of computation time of CGGibbs compared to naive computation of linear predictor in each step of the sampling algorithm as a function of the number of parameters in the GLM

the naive approach for high-dimensional models. This outcome aligns with the expected goal of the implementation.

However, despite the speedup, the computation time remains quite high. Running long or multiple Markov chains, such as those used in the exploratory analyses in this and the previous section, can still take several minutes or even hours.

5 Discussion

The R package **mcmcglm** introduces a user-friendly yet flexible interface for Bayesian inference, targeting GLMs through the innovative CGGibbs sampler. By extending the familiar interface of `glm`, the package facilitates adoption among users familiar with frequentist GLMs, offering the flexibility to specify arbitrary response families and prior distributions. Additionally, it enables users to explore a range of slice sampling within-Gibbs procedures, providing tools to evaluate the impact of tuning parameters on results. These features make **mcmcglm** a valuable addition to the ecosystem of R packages for Bayesian analysis.

However, overall computational efficiency remains a key limitation. Current runtimes, while faster than a corresponding naive implementation, can still be prohibitive for obtaining good approximations of posterior distributions, especially for high-dimensional parameter spaces. Addressing this challenge represents a priority for future development.

To enhance performance, we plan to explore parallel tempering methods, leveraging recent innovations described in Syed et al. (2021) [20], Surjanovic et al. (2022) [19], and Biron-Lattes et al. (2024) [3]. These methods, implemented in the Julia package **Pigeons**, offer promising avenues for sampling from complex probability distributions. A corresponding implementation in **mcmcglm** could significantly benefit R users working with challenging posterior distributions. Comparative evaluations against **rstan** [18], a robust and widely used R interface to the Stan probabilistic programming language [5], are also planned to assess **mcmcglm**'s efficiency and applicability relative to state-of-the-art Bayesian tools.

In addition to improving computational performance, we aim to expand the package's functionality. Planned features include tools for measuring uncertainty, as outlined in [4][p. 126], and effective sample size (ESS) calculations [11], which will offer users better diagnostics for evaluating their models. Automatic detection of conjugate priors, allowing the use of closed-form sampling where possible, could further improve both usability and efficiency. Moreover, introducing a formal testing framework will ensure stability and reliability as the package evolves.

Once the **mcmcglm** package reaches greater maturity, hope is that the package can be applied by researchers and practitioners to perform Bayesian analyses.

References

[1] Agresti, A. *Foundations of linear and generalized linear models*, 1st ed. ed. Wiley series in probability and statistics. John Wiley Sons Inc., Hoboken, New Jersey, 2015 - 2015.

[2] Andrew Gelman, John Carlin, H. S. D. D. A. V., and Rubin, D. *Bayesian Data Analysis, Third edition.* 2021.

[3] Biron-Lattes, M., Surjanovic, N., Syed, S., Campbell, T., and Bouchard-Cote, A. autoMALA: Locally adaptive Metropolis-adjusted Langevin algorithm. In *Proceedings of The 27th International Conference on Artificial Intelligence and Statistics* (02–04 May 2024), S. Dasgupta, S. Mandt, and Y. Li, Eds., vol. 238 of *Proceedings of Machine Learning Research*, PMLR, pp. 4600–4608.

[4] Bouchard-Côté, A. Probability, illustrated. `https://www.stat.ubc.ca/~bouchard/pub/probability-illustrated.pdf`. Accessed: 11-8-2024.

[5] Carpenter, B., Gelman, A., Hoffman, M. D., Lee, D., Goodrich, B., Betancourt, M., Brubaker, M., Guo, J., Li, P., and Riddell, A. Stan: A probabilistic programming language. *Journal of Statistical Software 76*, 1 (2017), 1–32.

[6] Cramer, J. The early origins of the logit model. *Studies in History and Philosophy of Science Part C: Studies in History and Philosophy of Biological and Biomedical Sciences 35*, 4 (2004), 613–626.

[7] Gelman, A., and Su, Y.-S. *arm: Data Analysis Using Regression and Multilevel/Hierarchical Models*, 2024. R package version 1.14-4.

[8] Heiner, M., Dahl, D. B., and Johnson, S. *qslice: Implementations of Various Slice Samplers*, 2024. R package version 0.3.1.

[9] Lambert, B. *A Student's Guide to Bayesian Statistics.* 2018.

[10] Luu, S., Xu, Z., Surjanovic, N., Biron-Lattes, M., Campbell, T., and Bouchard-Côté, A. Is gibbs sampling faster than hamiltonian monte carlo on glms?, 2024.

[11] Morita, S., Thall, P., and Müller, P. Determining the effective sample size of a parametric prior. *Biometrics 64* (07 2008), 595–602.

[12] Murray, I., Adams, R., and MacKay, D. Elliptical slice sampling. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics* (Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010), Y. W. Teh and M. Titterington, Eds., vol. 9 of *Proceedings of Machine Learning Research*, PMLR, pp. 541–548.

[13] Neal, R. M. Slice sampling. *The Annals of Statistics 31*, 3 (2003), 705 – 767.

[14] Nishihara, R., Murray, I., and Adams, R. P. Parallel mcmc with generalized elliptical slice sampling. *Journal of Machine Learning Research 15*, 61 (2014), 2087–2112.

[15] O'Hara-Wild, M., Kay, M., Hayes, A., and Hyndman, R. *distributional: Vectorised Probability Distributions*, 2024. R package version 0.5.0, https://github.com/mitchelloharawild/distributional.

[16] R Core Team. *R: A Language and Environment for Statistical Computing.* R Foundation for Statistical Computing, Vienna, Austria, 2024.

[17] Robert, C. P. *The Bayesian Choice: From Decision-Theoretic Foundations to Computational Implementation*, second ed. Springer Texts in Statistics. Springer, 2001.

[18] Stan Development Team. RStan: the R interface to Stan, 2024. R package version 2.32.6.

[19] Surjanovic, N., Syed, S., Bouchard-Côté, A., and Campbell, T. Parallel tempering with a variational reference. In *Advances in Neural Information Processing Systems* (2022), S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, Eds., vol. 35, Curran Associates, Inc., pp. 565–577.

[20] Syed, S., Bouchard-Côté, A., Deligiannidis, G., and Doucet, A. Non-reversible parallel tempering: A scalable highly parallel mcmc scheme. *Journal of the Royal Statistical Society Series B: Statistical Methodology 84*, 2 (12 2021), 321–350.

A Appendix

A.1 Data Simulation

The code used in the examples to generate the linear predictor η is:

```r
n <- 1000
x1 <- rnorm(n)
x2 <- rbinom(n, 1, .5)
b0 <- 1
b1 <- 1.5
b2 <- 2
lin_pred <- b0 + b1 * x1 + b2 * x2
```

A.2 Examples for Different Response Families

A.2.1 Example for Gaussian Family

The data for the Gaussian family is defined by

```r
y_norm <- rnorm(n, mean = lin_pred, sd = 1)
dat_norm <- data.frame(Y = y_norm, X1 = x1, X2 = x2)
```

A.2.2 Example for Binomial Family with Logit Link Function

Data is generated by:

```r
logit_binom <- binomial(link = "logit")

mu_logit <- logit_binom$linkinv(lin_pred)
y_logit <- rbinom(n, size = 1, prob = mu_logit)
dat_logit <- data.frame(Y = y_logit, X1 = x1, X2 = x2)
```

A trace plot of the results looks like:

```r
trace_plot(logit)
```

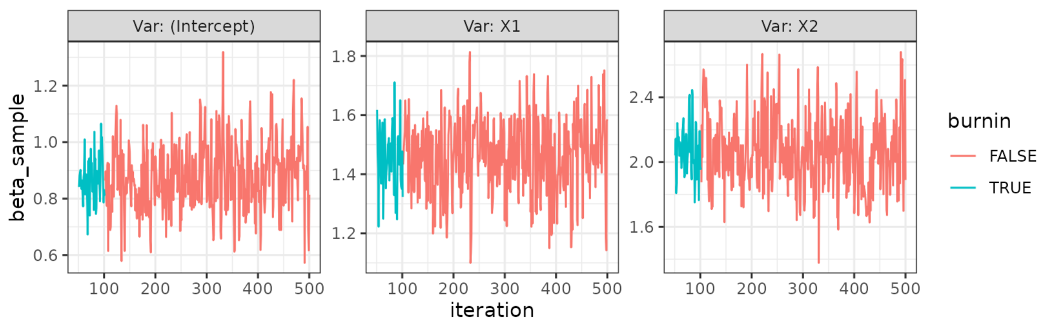

Figure 4: Trace plots of β samples with binomial response and logit link

A.2.3 Example for Binomial Family with Probit Link Function

Data is generated by:

```
probit_binom <- binomial(link = "probit")

mu_probit <- probit_binom$linkinv(lin_pred)
y_probit <- rbinom(n, size = 1, prob = mu_probit)
dat_probit <- data.frame(Y = y_probit, X1 = x1, X2 = x2)
```

A trace plot of the results looks like:

```
trace_plot(probit)
```

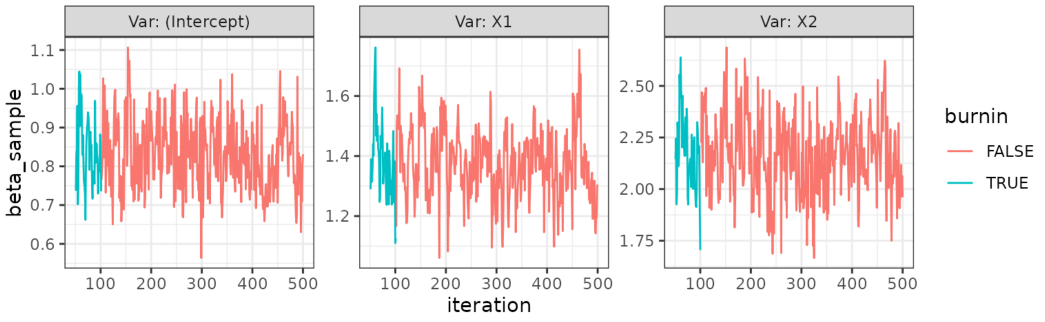

Figure 5: Trace plots of β samples with binomial response and probit link

A.2.4 Example for Poisson Family

For the Poisson family, the most commonly used link function is the log link, which we will showcase here. For other link functions, refer to the `help("family")` documentation.

The data for the Poisson family is defined as follows:

```
mu_log <- exp(lin_pred)
y_pois <- rpois(n, lambda = mu_log)
dat_pois <- data.frame(Y = y_pois, X1 = x1, X2 = x2)
```

We fit the `mcmcglm` model and inspect the trace plot as shown:

```
pois <- mcmcglm(formula = Y ~ .,
                data = dat_pois,
                family = "poisson",
                beta_prior = distributional::dist_normal(mean = 0, sd = 1),
                w = 0.5)
trace_plot(pois)
```

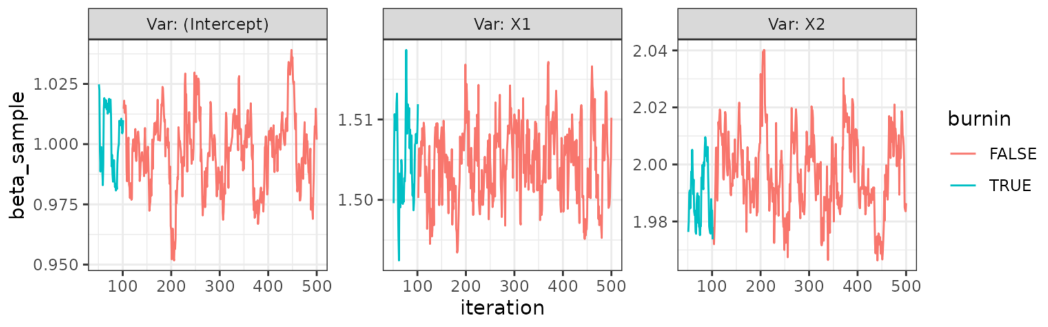

Figure 6: Trace plots of β samples with poisson response and log link

A.2.5 Example for Negative Binomial Family

The negative binomial family is useful for modeling count data when the variance exceeds the mean, making it a more flexible alternative to the Poisson family.

We define the data as follows:

```
mu_log <- exp(lin_pred)
y_nbinom <- rnbinom(n, size = 1, mu = mu_log)
dat_nbinom <- data.frame(Y = y_nbinom, X1 = x1, X2 = x2)

theta <- 3
```

Next, we fit the `mcmcglm` model and generate a trace plot:

```
nbinom <- mcmcglm(formula = Y ~ .,
                  data = dat_nbinom,
                  family = MASS::negative.binomial(theta),
                  beta_prior = distributional::dist_normal(mean = 0, sd = 1),
                  w = 0.5)
trace_plot(nbinom)
```

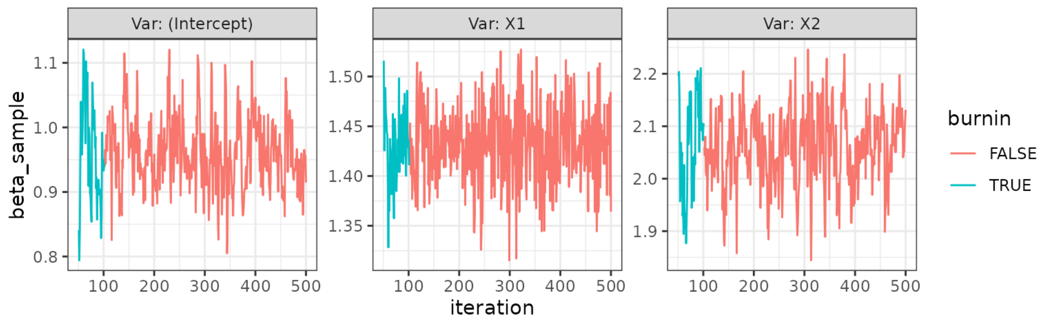

Figure 7: Trace plots of β samples with negative binomial response and log link

A.2.6 Example for Inverse Gaussian Family

Data is generated with an assumed shape and dispersion parameter of 1:

```
invgauss_fam <- inverse.gaussian()
y_invnorm <- statmod::rinvgauss(n,
                                mean =
                                  invgauss_fam$linkinv(lin_pred),
                                shape = 1, dispersion = 1)
dat_invnorm <- data.frame(Y = y_invnorm, X1 = x1, X2 = x2)
```

Trace plot:

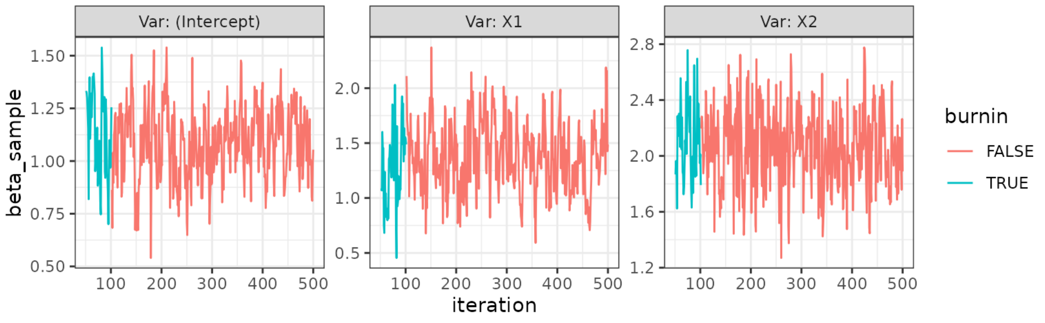

Figure 8: Trace plot of β samples with an inverse gaussian response

A.3 Examples of Specifying Priors

A.3.1 Example of (Different) Independent Priors

Here, we specify different priors for each β component:

```
indep <- mcmcglm(formula = Y ~ .,
                 family = "gaussian",
                 data = dat_norm,
                 beta_prior = list(
                   distributional::dist_normal(mean = 0, sd = 1),
                   distributional::dist_gamma(shape = 1, rate = 2),
                   distributional::dist_student_t(df = 8, mu = 3, sigma = 1)
                 ),
                 w = 0.5)
trace_plot(indep)
```

A.3.2 Example of Multivariate Normal Prior

We use the multivariate normal distribution as a prior:

$$\beta \sim \mathcal{N}_3\left(\begin{pmatrix}1\\2\\3\end{pmatrix}, \begin{bmatrix}1 & 0.5 & 0.2\\0.5 & 1 & 0.5\\0.2 & 0.5 & 1\end{bmatrix}\right)$$

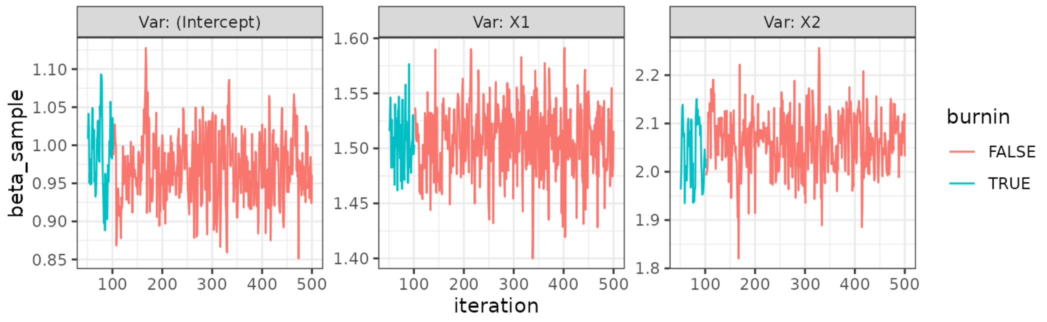

Figure 9: Trace plot of β samples with different prior distributions for each parameter

```r
mult <- mcmcglm(formula = Y ~ .,
                family = "gaussian",
                data = dat_norm,
                beta_prior = distributional::dist_multivariate_normal(
                  mu = list(1:3),
                  sigma = list(
                    matrix(
                      c(1, 0.5, 0.2, 0.5, 1, 0.5, 0.2, 0.5, 1),
                      ncol = 3,
                      byrow = TRUE))
                ),
                w = 0.5)
```

A trace plot of the resulting fit is:

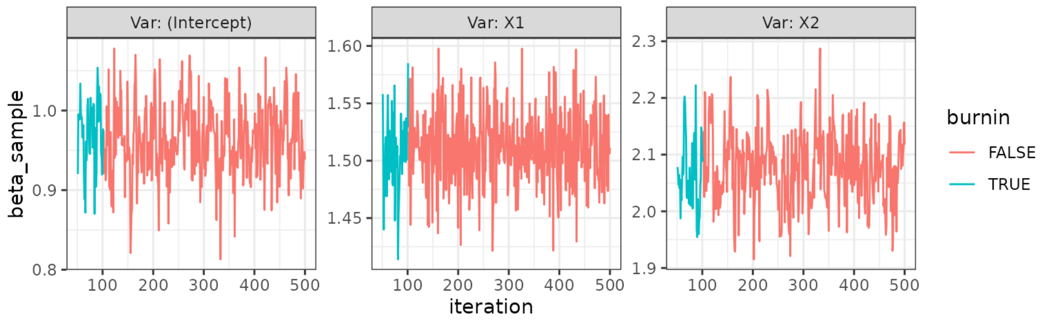

Figure 10: Trace plot of β samples with multivariate prior specification

A.4 Examples of Changing the Samping Procedure

A.4.1 Example of using Elliptical Slice Sampler

A figure is available in 11

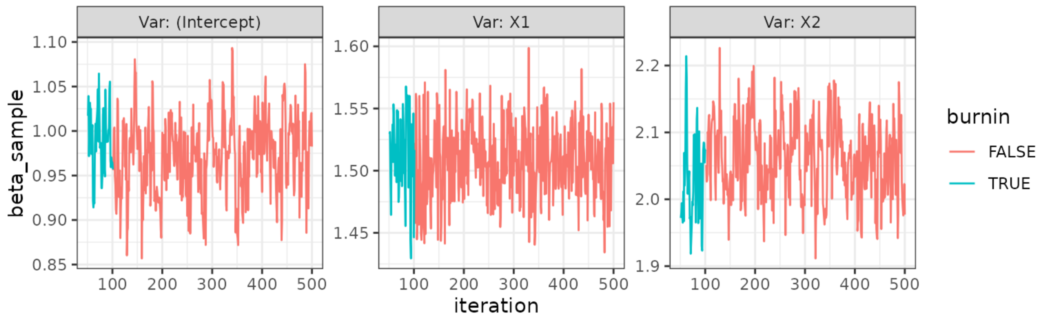

Figure 11: Trace plot of β samples using the elliptical slice sampler from Murray et. al (2010) [12]

A.4.2 Example of Varying degrees of Freedom in a Generalized Elliptical Slice Sampler

Below is an example of using the `qslice::slice_genelliptical` sampling procedure, varying the `df` parameter (using data from appendix A.2.1):

```
df1_mcmcglms <- mcmcglm_across_tuningparams(
  c(1, seq(from = 10, by = 10, length.out = 3)),
  tuning_parameter_name = "df",
  formula = Y ~ .,
  family = "gaussian",
  data = dat_norm,
  qslice_fun = qslice::slice_genelliptical,
  mu = 2,
  sigma = 1,
  parallelise = FALSE)
```

And trace plot:

```
plot_mcmcglm_across_tuningparams(df1_mcmcglms)
```

A.4.3 Example of Normal Conjugate Prior

```
norm_norm <- mcmcglm(formula = Y ~ .,
                     family = "gaussian",
                     data = dat_norm,
                     beta_prior = distributional::dist_normal(0, 1),
                     sample_method = "normal-normal")
trace_plot(norm_norm)
```

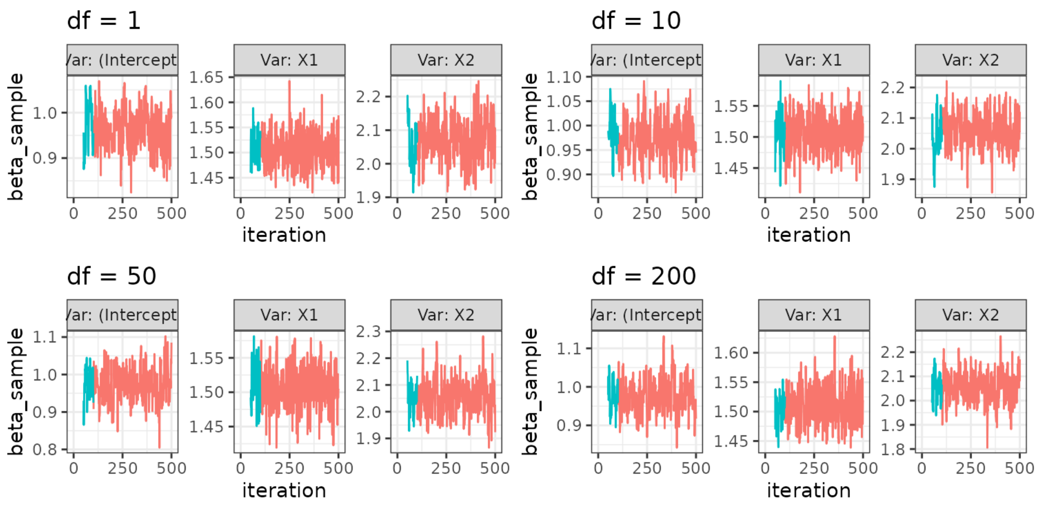

Figure 12: Trace plots of β samples using different degrees of freedom in the generalized elliptical slice sampler from Nishihara et al. (2014) [14]

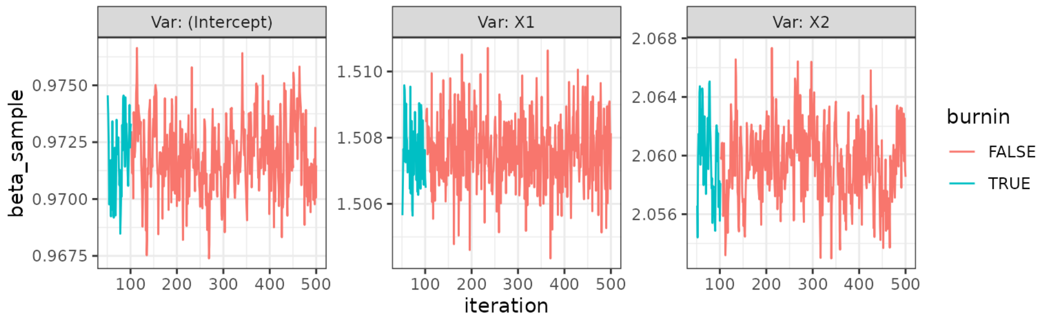

Figure 13: Trace plots of β samples using Gibbs sampling with a closed-form conditional normal distribution in case of normal-conjugacy with gaussian response and prior