bridgesampling: An R Package for Estimating Normalizing Constants

Quentin F. Gronau
University of Amsterdam

Henrik Singmann
University of Zurich

Eric-Jan Wagenmakers
University of Amsterdam

Abstract

Statistical procedures such as Bayes factor model selection and Bayesian model averaging require the computation of normalizing constants (e.g., marginal likelihoods). These normalizing constants are notoriously difficult to obtain, as they usually involve high-dimensional integrals that cannot be solved analytically. Here we introduce an R package that uses bridge sampling (Meng and Wong 1996; Meng and Schilling 2002) to estimate normalizing constants in a generic and easy-to-use fashion. For models implemented in Stan, the estimation procedure is automatic. We illustrate the functionality of the package with three examples.

Keywords: bridge sampling, marginal likelihood, model selection, Bayes factor, Warp-III.

1. Introduction

In many statistical applications, it is essential to obtain normalizing constants of the form

$$Z = \int_{\boldsymbol{\Theta}} q(\boldsymbol{\theta}) \, \mathrm{d}\boldsymbol{\theta}, \tag{1}$$

where $p(\boldsymbol{\theta}) = q(\boldsymbol{\theta})/Z$ denotes a probability density function (pdf) defined on the domain $\boldsymbol{\Theta} \subseteq \mathbb{R}^p$. For instance, the estimation of normalizing constants plays a crucial role in free energy estimation in physics, missing data analyses in likelihood-based approaches, Bayes factor model comparisons, and Bayesian model averaging (e.g., Gelman and Meng 1998). In this article, we focus on the role of the normalizing constant in Bayesian inference; however, the **bridgesampling** package can be used in any context where one desires to estimate a normalizing constant.

In Bayesian inference, the normalizing constant of the joint posterior distribution is involved in (1) parameter estimation, where the normalizing constant ensures that the posterior integrates to one; (2) Bayes factor model comparison, where the ratio of normalizing constants quantifies the data-induced change in beliefs concerning the relative plausibility of two competing models (e.g., Kass and Raftery 1995); (3) Bayesian model averaging, where the normalizing constant is required to obtain posterior model probabilities (BMA; Hoeting *et al.* 1999).

For Bayesian parameter estimation, the need to compute the normalizing constant can usually be circumvented by the use of sampling approaches such as Markov chain Monte Carlo (MCMC; e.g., Gamerman and Lopes 2006). However, for Bayes factor model comparison and BMA, the normalizing constant of the joint posterior distribution – in this context usually called *marginal likelihood* – remains of essential importance. This is evident from the fact that the posterior model probability of model $\mathcal{M}_i$, $i \in \{1, 2, \ldots, m\}$, given data $\boldsymbol{y}$ is obtained as

$$\underbrace{p(\mathcal{M}_i \mid \boldsymbol{y})}_{\text{posterior model probability}} = \underbrace{\frac{p(\boldsymbol{y} \mid \mathcal{M}_i)}{\sum_{j=1}^{m} p(\boldsymbol{y} \mid \mathcal{M}_j)\, p(\mathcal{M}_j)}}_{\text{updating factor}} \times \underbrace{p(\mathcal{M}_i)}_{\text{prior model probability}}, \tag{2}$$

where $p(\boldsymbol{y} \mid \mathcal{M}_i)$ denotes the *marginal likelihood* of model $\mathcal{M}_i$.

If the model comparison involves only two models, $\mathcal{M}_1$ and $\mathcal{M}_2$, it is convenient to consider the odds of one model over the other. Bayes' rule yields:

$$\underbrace{\frac{p(\mathcal{M}_1 \mid \boldsymbol{y})}{p(\mathcal{M}_2 \mid \boldsymbol{y})}}_{\text{posterior odds}} = \underbrace{\frac{p(\boldsymbol{y} \mid \mathcal{M}_1)}{p(\boldsymbol{y} \mid \mathcal{M}_2)}}_{\text{Bayes factor BF}_{12}} \times \underbrace{\frac{p(\mathcal{M}_1)}{p(\mathcal{M}_2)}}_{\text{prior odds}}. \tag{3}$$

The change in odds brought about by the data is given by the ratio of the marginal likelihoods of the models and is known as the *Bayes factor* (Jeffreys 1961; Kass and Raftery 1995; Etz and Wagenmakers 2017). Equation 2 and Equation 3 highlight that the normalizing constant of the joint posterior distribution, that is, the marginal likelihood, is required for computing both posterior model probabilities and Bayes factors.

The marginal likelihood is obtained by integrating out the model parameters with respect to their prior distribution:

$$p(\boldsymbol{y} \mid \mathcal{M}_i) = \int_{\boldsymbol{\Theta}} p(\boldsymbol{y} \mid \boldsymbol{\theta}, \mathcal{M}_i)\, p(\boldsymbol{\theta} \mid \mathcal{M}_i)\, \mathrm{d}\boldsymbol{\theta}. \tag{4}$$

The marginal likelihood implements the principle of parsimony also known as *Occam's razor* (e.g., Jefferys and Berger 1992; Myung and Pitt 1997; Vandekerckhove *et al.* 2015). Unfortunately, the marginal likelihood can be computed analytically for only a limited number of models. For more complicated models (e.g., hierarchical models), the marginal likelihood is a high-dimensional integral that usually cannot be solved analytically. This computational hurdle has complicated the application of Bayesian model comparisons for decades.

To overcome this hurdle, a range of different methods have been developed that vary in accuracy, speed, and complexity of implementation: naive Monte Carlo estimation, importance sampling, the generalized harmonic mean estimator, Reversible Jump MCMC (Green 1995), the product-space method (Carlin and Chib 1995; Lodewyckx *et al.* 2011), Chib's method (Chib 1995), thermodynamic integration (e.g., Lartillot and Philippe 2006), path sampling (Gelman and Meng 1998), and others. The ideal method is fast, accurate, easy to implement, general, and unsupervised, allowing non-expert users to treat it as a "black box".

In our experience, one of the most promising methods for estimating normalizing constants is bridge sampling (Meng and Wong 1996; Meng and Schilling 2002). Bridge sampling is a general procedure that performs accurately even in high-dimensional parameter spaces such as those that are regularly encountered in hierarchical models. In fact, simpler estimators such as the naive Monte Carlo estimator, the generalized harmonic mean estimator, and importance sampling are special sub-optimal cases of the bridge identity described in more detail below (e.g., Frühwirth–Schnatter 2004; Gronau *et al.* 2017b).

In this article, we introduce **bridgesampling**, an R (R Core Team 2016) package that enables the straightforward and user-friendly estimation of the marginal likelihood (and of normalizing constants more generally) via bridge sampling techniques. In general, the user needs to provide to the `bridge_sampler` function four quantities that are readily available:

1. an object with posterior samples (argument `samples`);
2. a function that computes the log of the unnormalized posterior density for a set of model parameters (argument `log_posterior`);
3. a data object that contains the data and potentially other relevant quantities for evaluating `log_posterior` (argument `data`);
4. lower and upper bounds for the parameters (arguments `lb` and `ub`, respectively).

Given these inputs, the **bridgesampling** package provides an estimate of the log marginal likelihood.

Figure 1 displays the steps that a user may take when using the **bridgesampling** package. Starting from the top, the user provides the basic required arguments to the `bridge_sampler` function which then produces an estimate of the log marginal likelihood. With this estimate in hand – usually for at least two different models – the user can compute posterior model probabilities using the `post_prob` function, Bayes factors using the `bf` function, and approximate estimation errors using the `error_measures` function. A schematic call of the `bridge_sampler` function looks as follows (detailed examples are provided in the next sections):

```
R> bridge_sampler(samples = samples, log_posterior = log_posterior,
+                 data = data, lb = lb, ub = ub)
```

The `bridge_sampler` function is an S3 generic which currently has methods for objects of class `mcmc.list` (Plummer *et al.* 2006), `stanfit` (Stan Development Team 2016a), `matrix`, `rjags` (Plummer 2016; Su and Yajima 2015), `runjags` (Denwood 2016), and `stanreg` (Stan Development Team 2016b).[1] This allows the user to obtain posterior samples in a convenient and efficient way, for instance, via JAGS (Plummer 2003) or a highly customized sampler. Hence, bridge sampling does not require users to program their own MCMC routines to obtain posterior samples; this convenience is usually missing for methods such as Reversible Jump MCMC (but see Gelling *et al.* 2017).

When the model is specified in Stan (Carpenter *et al.* 2017; Stan Development Team 2016a) – in a way that retains the constants, as described below – obtaining the marginal likelihood is even simpler: the user only needs to pass the `stanfit` object to the `bridge_sampler` function. The combination of Stan and the **bridgesampling** package therefore produces an unsupervised, black box computation of the marginal likelihood.

This article is structured as follows: First we describe the implementation details of the algorithm from **bridgesampling**; second, we illustrate the functionality of the package using a simple Bayesian t-test example where posterior samples are obtained via JAGS. In this section, we also explain a heuristic to obtain the function that computes the log of the unnormalized posterior density in JAGS; third, we describe in more detail the interface to Stan which enables an even more automatized computation of the marginal likelihood. Fourth, we illustrate use of the Stan interface with two well-known examples from the Bayesian model selection literature.

[1]We thank Ben Goodrich for adding the `stanreg` method to our package.

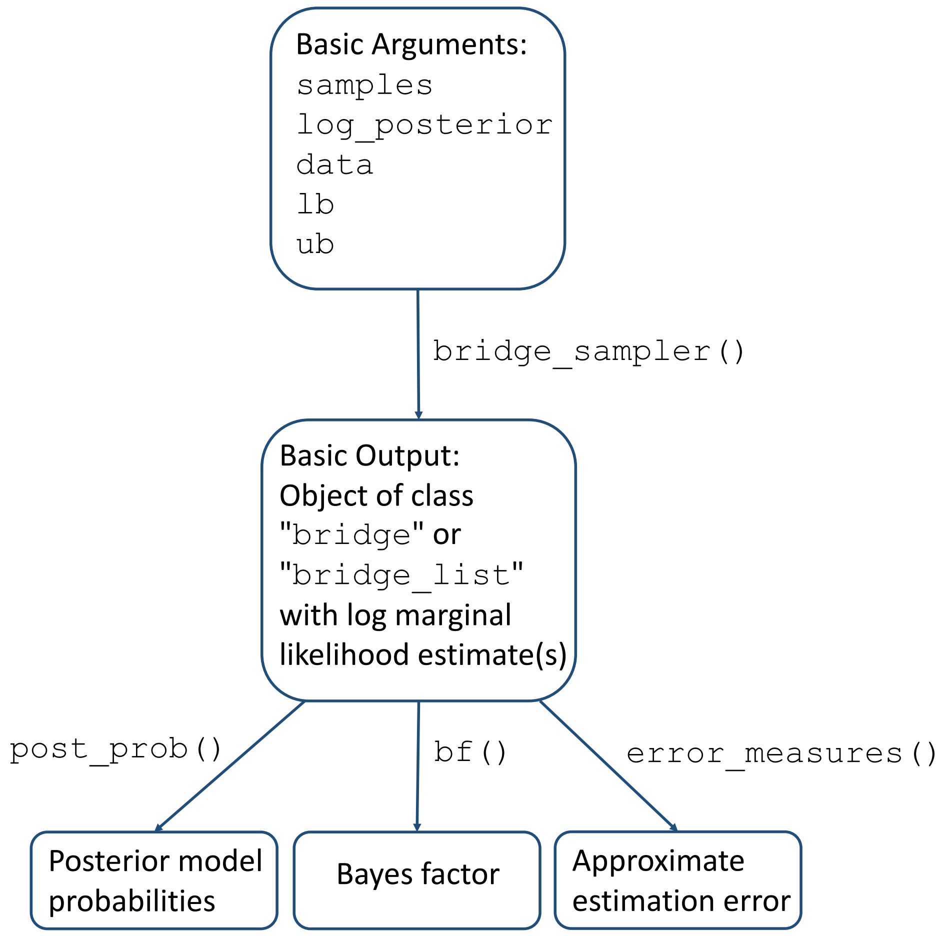
Basic Arguments:
samples
log_posterior
data
lb
ub
bridge_sampler()
Basic Output:
Object of class "bridge" or "bridge_list" with log marginal likelihood estimate(s)
post_prob()
bf()
error_measures()
Posterior model probabilities
Bayes factor
Approximate estimation error

Figure 1: Flow chart of the steps that a user may take when using the **bridgesampling** package. In general, the user needs to provide a posterior samples object (`samples`), a function that computes the log of the unnormalized posterior density (`log_posterior`), the data (`data`), and parameter bounds (`lb` and `ub`). The `bridge_sampler` function then produces an estimate of the log marginal likelihood. This is usually repeated for at least two different models. The user can then compute posterior model probabilities (using the `post_prob` function), Bayes factors (using the `bf` function), and approximate estimation errors (using the `error_measures` function). Note that the summary method for bridge objects automatically invokes the `error_measures` function. Figure available at `https://tinyurl.com/ybf4jxka` under CC license `https://creativecommons.org/licenses/by/2.0/`.

2. Bridge sampling: the algorithm

In its original formulation, bridge sampling was intended to estimate a ratio of *two* normalizing constants (Meng and Wong 1996). However, the accuracy of the estimator depends in part on the overlap between the two involved distributions; consequently, the accuracy can be increased by estimating one normalizing constant at a time, using as a second distribution a convenient normalized proposal distribution that closely matches the distribution of interest (e.g., Gronau *et al.* 2017b; Overstall and Forster 2010). The bridge sampling estimator of the marginal likelihood is then given by:[2]

$$p(\boldsymbol{y}) = \frac{\mathbb{E}_{g(\boldsymbol{\theta})}\left[h(\boldsymbol{\theta})\, p(\boldsymbol{y} \mid \boldsymbol{\theta})\, p(\boldsymbol{\theta})\right]}{\mathbb{E}_{p(\boldsymbol{\theta}|\boldsymbol{y})}\left[h(\boldsymbol{\theta})\, g(\boldsymbol{\theta})\right]} \approx \frac{\frac{1}{n_2}\sum_{j=1}^{n_2} h(\tilde{\boldsymbol{\theta}}_j)\, p(\boldsymbol{y} \mid \tilde{\boldsymbol{\theta}}_j)\, p(\tilde{\boldsymbol{\theta}}_j)}{\frac{1}{n_1}\sum_{i=1}^{n_1} h(\boldsymbol{\theta}_i^*)\, g(\boldsymbol{\theta}_i^*)}, \tag{5}$$

where $h(\boldsymbol{\theta})$ is called the *bridge function* and $g(\boldsymbol{\theta})$ denotes the *proposal distribution.* $\{\boldsymbol{\theta}_1^*, \boldsymbol{\theta}_2^*, \ldots, \boldsymbol{\theta}_{n_1}^*\}$ denote n_1 samples from the posterior distribution $p(\boldsymbol{\theta} \,|\, \boldsymbol{y})$ and $\{\tilde{\boldsymbol{\theta}}_1, \tilde{\boldsymbol{\theta}}_2, \ldots, \tilde{\boldsymbol{\theta}}_{n_2}\}$ denote n_2 samples from the proposal distribution $g(\boldsymbol{\theta})$.

A remaining question is how to choose the bridge function $h(\boldsymbol{\theta})$ and the proposal distribution $g(\boldsymbol{\theta})$. Meng and Wong (1996) showed that the optimal bridge function (in the sense that it minimizes the relative mean-squared error of the estimator) is given by

$$h(\boldsymbol{\theta}) \propto \frac{1}{s_1\, p(\boldsymbol{y} \mid \boldsymbol{\theta})\, p(\boldsymbol{\theta}) + s_2\, p(\boldsymbol{y})\, g(\boldsymbol{\theta})}, \tag{6}$$

where $s_i = \frac{n_i}{n_1+n_2}, i \in \{1, 2\}$. This choice is optimal only when the samples are independent and identically distributed (*i.i.d.*) which is not the case when the posterior samples are obtained using MCMC procedures. Therefore, by default, the **bridgesampling** package replaces n_1 in defining the weights s_1 and s_2 by the effective posterior sample size which is computed using the **coda** package (Plummer *et al.* 2006).[3] When the user sets the argument `use_neff = FALSE`, the actual obtained sample size is used instead of the effective sample size. Another method for getting closer to the *i.i.d.* assumption is to thin the MCMC chains. As detailed below, the proposal distribution is chosen in such a way that it yields *i.i.d.* samples.

Note that the expression for the optimal bridge function contains $p(\boldsymbol{y})$, that is, the marginal likelihood – the very quantity that we attempt to estimate. Therefore, Meng and Wong (1996) suggested an iterative updating scheme where an initial guess of the marginal likelihood $\hat{p}(\boldsymbol{y})^{(0)}$ is updated until convergence. The estimate at iteration $t+1$ is obtained as follows:

$$\hat{p}(\boldsymbol{y})^{(t+1)} = \frac{\frac{1}{n_2}\sum\limits_{j=1}^{n_2} \frac{l_{2,j}}{s_1\, l_{2,j} + s_2\, \hat{p}(\boldsymbol{y})^{(t)}}}{\frac{1}{n_1}\sum\limits_{i=1}^{n_1} \frac{1}{s_1\, l_{1,i} + s_2\, \hat{p}(\boldsymbol{y})^{(t)}}}, \tag{7}$$

where $l_{1,i} = \frac{p(\boldsymbol{y}|\boldsymbol{\theta}_i^*)\, p(\boldsymbol{\theta}_i^*)}{g(\boldsymbol{\theta}_i^*)}$, and $l_{2,j} = \frac{p(\boldsymbol{y}|\tilde{\boldsymbol{\theta}}_j)\, p(\tilde{\boldsymbol{\theta}}_j)}{g(\tilde{\boldsymbol{\theta}}_j)}$. In practice, a more numerically stable version of Equation 7 is implemented (e.g., Gronau *et al.* 2017b, Appendix B). Furthermore, the `Brobdingnag` R package (Hankin 2007) is used to avoid numerical underflow.

The iterative scheme usually converges within a few iterations. Note that, crucially, $l_{1,i}$ and $l_{2,j}$ need only be computed once before the iterative updating scheme is started. In practice, evaluating $l_{1,i}$ and $l_{2,j}$ takes up most of the computational time. Luckily, $l_{1,i}$ and $l_{2,j}$ can be computed completely in parallel for each $i \in \{1, 2, \ldots, n_1\}$ and each $j \in \{1, 2, \ldots, n_2\}$, respectively. That is, in contrast to MCMC procedures, the evaluation of, for instance,

[2]We omit conditioning on the model for enhanced legibility. It should be kept in mind, however, that this yields the estimate of the marginal likelihood for a particular model $\mathcal{M}_i$, that is, $p(\boldsymbol{y} \mid \mathcal{M}_i)$.

[3]Specifically, the median effective sample size across all parameters is used.

$l_{1,i+1}$ does *not* require one to evaluate $l_{1,i}$ first (since the posterior samples and proposal samples are already available). The **bridgesampling** package enables the user to compute $l_{1,i}$ and $l_{2,j}$ in parallel by setting the argument `cores` to an integer larger than one. On Unix/macOS machines, this parallelization is implemented using the **parallel** package. On Windows machines this is achieved using the **snowfall** package (Knaus 2015).[4]

After having specified the bridge function, one needs to choose the proposal distribution $g(\boldsymbol{\theta})$. The **bridgesampling** package implements two different choices: (1) a multivariate normal proposal distribution and (2) a standard multivariate normal distribution combined with a *warped* posterior distribution. Both choices increase the efficiency of the estimator by making the proposal and the posterior distribution as similar as possible. Note that under the optimal bridge function, the bridge sampling estimator is robust to the relative tail behavior of the posterior and the proposal distribution. This stands in sharp contrast to the importance and the generalized harmonic mean estimator for which unwanted tail behavior produces estimators with very large or even infinite variances (e.g., Owen and Zhou 2000; Frühwirth–Schnatter 2004; Gronau *et al.* 2017b).

2.1. Option I: the multivariate normal proposal distribution

The first choice for the proposal distribution that is implemented in the **bridgesampling** package is a multivariate normal distribution with mean vector and covariance matrix that match the respective posterior samples quantities. This choice (henceforth "the normal method") generalizes to high dimensions and accounts for potential correlations in the joint posterior distribution. This proposal distribution is obtained by setting the argument `method = "normal"` in the `bridge_sampler` function; this is the default setting. This choice assumes that all parameters are allowed to range across the entire real line. In practice, this assumption may not be fulfilled for all components of the parameter vector, however, it is usually possible to transform the parameters so that this requirement is met. This is achieved by transforming the original p-dimensional parameter vector $\boldsymbol{\theta}$ (which may contain components that range only across a subset of $\mathbb{R}$) to a new parameter vector $\boldsymbol{\xi}$ (where all components are allowed to range across the entire real line) using a diffeomorphic vector-valued function f so that $\boldsymbol{\xi} = f(\boldsymbol{\theta})$. By the change-of-variable rule, the posterior density with respect to the new parameter vector $\boldsymbol{\xi}$ is given by:

$$p(\boldsymbol{\xi} \mid \boldsymbol{y}) = p_{\boldsymbol{\theta}}(f^{-1}(\boldsymbol{\xi}) \mid \boldsymbol{y}) \left|\det\left[J_{f^{-1}}(\boldsymbol{\xi})\right]\right|, \tag{8}$$

where $p_{\boldsymbol{\theta}}(f^{-1}(\boldsymbol{\xi}) \mid \boldsymbol{y})$ refers to the untransformed posterior density with respect to $\boldsymbol{\theta}$ evaluated for $f^{-1}(\boldsymbol{\xi}) = \boldsymbol{\theta}$. $J_{f^{-1}}(\boldsymbol{\xi})$ denotes the Jacobian matrix with the element in the i-th row and j-th column given by $\frac{\partial\theta_i}{\partial\xi_j}$. Crucially, the posterior density with respect to $\boldsymbol{\xi}$ retains the normalizing constant of the posterior density with respect to $\boldsymbol{\theta}$; hence, one can select a convenient transformation without changing the normalizing constant. Note that in order to apply a transformation no new samples are required; instead the original samples can simply be transformed using the function f.

In principle, users can select transformations themselves. Nevertheless, the **bridgesampling** package comes with a set of built-in transformations (see Table 1), allowing the user to work with the model in a familiar parameterization. When the user then supplies a named vector

[4]Due to technical limitations specific to Windows, this parallelization is not available for the `stanfit` and `stanreg` methods.

Table 1: Overview of built-in transformations in the **bridgesampling** package. l denotes a parameter lower bound and u denotes an upper bound. $\Phi(\cdot)$ denotes the cumulative distribution function (cdf) and $\phi(\cdot)$ the probability density function (pdf) of the normal distribution.

Type	Transformation	Inv.-Transformation	Jacobian Contribution
unbounded	$\xi_i = \theta_i$	$\theta_i = \xi_i$	$\left\lvert \frac{\partial \theta_i}{\partial \xi_i} \right\rvert = 1$
lower-bounded	$\xi_i = \log\left(\theta_i - l\right)$	$\theta_i = \exp\left(\xi_i\right) + l$	$\left\lvert \frac{\partial \theta_i}{\partial \xi_i} \right\rvert = \exp\left(\xi_i\right)$
upper-bounded	$\xi_i = \log\left(u - \theta_i\right)$	$\theta_i = u - \exp\left(\xi_i\right)$	$\left\lvert \frac{\partial \theta_i}{\partial \xi_i} \right\rvert = \exp\left(\xi_i\right)$
double-bounded	$\xi_i = \Phi^{-1}\left(\frac{\theta_i - l}{u - l}\right)$	$\theta_i = \left(u - l\right)\Phi\left(\xi_i\right) + l$	$\left\lvert \frac{\partial \theta_i}{\partial \xi_i} \right\rvert = \left(u - l\right)\phi\left(\xi_i\right)$

with lower and upper bounds for the parameters (arguments `lb` and `ub`, respectively), the package internally transforms the relevant parameters and adjusts the expressions by the Jacobian term. Furthermore, as will be elaborated upon below, when the model is fitted in `Stan`, the **bridgesampling** package takes advantage of the rich class of `Stan` transformations.

The transformations built into the **bridgesampling** package are useful whenever each component of the parameter vector can be transformed separately. In this scenario, there are four possible cases per parameter: (1) the parameter is unbounded; (2) the parameter has a lower bound (e.g., variance parameters); (3) the parameter has an upper bound; and (4) the parameter has a lower and an upper bound (e.g., rate parameters). As shown in Table 1, in case (1) the identity (i.e., no) transformation is applied. In case (2) and (3), logarithmic transformations are applied to transform the parameter to the real line. In case (4) a probit transformation is applied. Note that internally, the posterior density is automatically adjusted by the relevant Jacobian term. Since each component is transformed separately, the resulting Jacobian matrix will be diagonal. This is convenient since it implies that the absolute value of the determinant is the product of the absolute values of the diagonal entries of the Jacobian matrix:

$$\left\lvert \det\left[J_{f^{-1}}(\boldsymbol{\xi})\right]\right\rvert = \prod_{i=1}^{p} \left\lvert \frac{\partial \theta_i}{\partial \xi_i} \right\rvert. \tag{9}$$

Once all posterior samples have been transformed to the real line, a multivariate normal distribution is fitted using method-of-moments. On a side note, bridge sampling may underestimate the marginal likelihood when the same posterior samples are used both for fitting the proposal distribution and for the iterative updating scheme (i.e., Equation 7). Hence, as recommended by Overstall and Forster (2010), the **bridgesampling** package divides each MCMC chain into two halves, using the first half for fitting the proposal distribution and the second half for the iterative updating scheme.

2.2. Option II: warping the posterior distribution

The second choice for the proposal distribution that is implemented in the **bridgesampling** package is a standard multivariate normal distribution in combination with a *warped* posterior distribution. The goal is still to match the posterior and the proposal distribution as closely as possible. However, instead of manipulating the proposal distribution, it is fixed to a standard multivariate normal distribution, and the posterior distribution is manipulated (i.e., warped). Crucially, the warped posterior density retains the normalizing constant of the original posterior density. The general methodology is referred to as Warp bridge sampling (Meng and Schilling 2002).

There exist several variants of Warp bridge sampling; in the **bridgesampling** package, we implemented Warp-III bridge sampling (Meng and Schilling 2002; Overstall 2010; Gronau *et al.* 2017c) which can be used by setting `method = "warp3"`. This version matches the first three moments of the posterior and the proposal distribution. That is, in contrast to the simpler normal method described above, Warp-III not only matches the mean vector and the covariance matrix of the two distributions, but also the skewness. Consequently, when the posterior distribution is skewed, Warp-III may result in an estimator that is less variable. When the posterior distribution is symmetric, both Warp-III and the normal method should yield estimators that are about equally efficient. Hence, in principle, Warp-III should always provide estimates that are at least as accurate as the normal method. However, the Warp-III method also takes about twice as much time to execute as the normal method; the reason for this is that Warp-III sampling results in a mixture density (for details, see Overstall 2010; Gronau *et al.* 2017c) which requires that the unnormalized posterior density is evaluated twice as often as in the normal method.

Warp-III bridge sampling starts with posterior samples that can range across the entire real line (i.e., $\boldsymbol{\xi}$) and is based on the following stochastic transformation:

$$\boldsymbol{\eta} = \underbrace{b}_{\text{symmetry}} \times \underbrace{\boldsymbol{R}^{-1}}_{\text{covariance } \boldsymbol{I}} \times \underbrace{(\boldsymbol{\xi} - \boldsymbol{\mu})}_{\text{mean } \mathbf{0}}, \tag{10}$$

where $b \sim \text{Bernoulli}(0.5)$ on $\{-1, 1\}$ and $\boldsymbol{\mu}$ corresponds to the expected value of $\boldsymbol{\xi}$ (i.e., the mean vector). The matrix $\boldsymbol{R}$ is obtained via the Cholesky decomposition of the covariance matrix of $\boldsymbol{\xi}$, denoted as $\boldsymbol{\Sigma}$, hence, $\boldsymbol{\Sigma} = \boldsymbol{R}\boldsymbol{R}^\top$. In practical applications, $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ are unknown and are estimated using the first half of the posterior samples. The iterative scheme in Equation 7 is then applied with

$$l_{1,i} = \frac{\frac{|\boldsymbol{R}|}{2}\left[\tilde{p}_{\boldsymbol{\xi}}(2\boldsymbol{\mu} - \boldsymbol{\xi}_i^* \mid \boldsymbol{y}) + \tilde{p}_{\boldsymbol{\xi}}(\boldsymbol{\xi}_i^* \mid \boldsymbol{y})\right]}{g\left(\boldsymbol{R}^{-1}\left(\boldsymbol{\xi}_i^* - \boldsymbol{\mu}\right)\right)}, \tag{11}$$

and

$$l_{2,j} = \frac{\frac{|\boldsymbol{R}|}{2}\left[\tilde{p}_{\boldsymbol{\xi}}(\boldsymbol{\mu} - \boldsymbol{R}\tilde{\boldsymbol{\eta}}_j \mid \boldsymbol{y}) + \tilde{p}_{\boldsymbol{\xi}}(\boldsymbol{\mu} + \boldsymbol{R}\tilde{\boldsymbol{\eta}}_j \mid \boldsymbol{y})\right]}{g(\tilde{\boldsymbol{\eta}}_j)}, \tag{12}$$

where $\{\boldsymbol{\xi}_1^*, \boldsymbol{\xi}_2^*, \ldots, \boldsymbol{\xi}_{n_1}^*\}$ are n_1 draws from $p(\boldsymbol{\xi} \mid \boldsymbol{y})$, and $\{\tilde{\boldsymbol{\eta}}_1, \tilde{\boldsymbol{\eta}}_2, \ldots, \tilde{\boldsymbol{\eta}}_{n_2}\}$ are n_2 draws from the standard normal proposal distribution $g(\boldsymbol{\eta})$. Furthermore, $\tilde{p}_{\boldsymbol{\xi}}(\boldsymbol{\xi} \mid \boldsymbol{y})$ denotes the unnormalized posterior density of the unbounded (but not warped) posterior samples given by:

$$\tilde{p}_{\boldsymbol{\xi}}(\boldsymbol{\xi} \mid \boldsymbol{y}) = p(\boldsymbol{y} \mid \boldsymbol{\xi})\, p(\boldsymbol{\xi}) = \left|\det\left[J_{f^{-1}}(\boldsymbol{\xi})\right]\right| p_{\boldsymbol{\theta}}(\boldsymbol{y} \mid f^{-1}(\boldsymbol{\xi}))\, p_{\boldsymbol{\theta}}(f^{-1}(\boldsymbol{\xi})). \tag{13}$$

2.3. Estimation error

Once the marginal likelihood has been estimated, the user can obtain an estimate of the estimation error in a number of different ways. One method is to use the `error_measures` function which is an `S3` generic. Note that the `summary` method for objects returned by `bridge_sampler` internally calls the `error_measures` function and thus provides a convenient summary of the estimated log marginal likelihood and the estimation uncertainty. For marginal likelihoods estimated with the `"normal"` method and `repetitions = 1`, the `error_measures` function provides an approximate relative mean-squared error of the marginal likelihood estimate, an approximate coefficient of variation, and an approximate percentage error. The relative mean-squared error of the marginal likelihood estimate is given by:

$$\text{RE}^2 = \frac{\mathbb{E}\left[\big(\hat{p}(\boldsymbol{y}) - p(\boldsymbol{y})\big)^2\right]}{p(\boldsymbol{y})^2}. \tag{14}$$

Frühwirth–Schnatter (2004) provided a derivation of an approximate relative mean-squared error of the marginal likelihood which takes into account that the samples from the proposal distribution are independent, whereas the samples from the posterior distribution may be autocorrelated (e.g., when using MCMC sampling procedures). The approximate relative mean-squared error is given by:

$$\widehat{\text{RE}}^2 = \frac{1}{n_2} \frac{\mathbb{V}_{g(\boldsymbol{\xi})}\big(f_1(\boldsymbol{\xi})\big)}{\mathbb{E}^2_{g(\boldsymbol{\xi})}\big(f_1(\boldsymbol{\xi})\big)} + \frac{\rho_{f_2}(0)}{n_1} \frac{\mathbb{V}_{p(\boldsymbol{\xi}|\boldsymbol{y})}\big(f_2(\boldsymbol{\xi})\big)}{\mathbb{E}^2_{p(\boldsymbol{\xi}|\boldsymbol{y})}\big(f_2(\boldsymbol{\xi})\big)}, \tag{15}$$

where $f_1(\boldsymbol{\xi}) = \frac{p(\boldsymbol{\xi}|\boldsymbol{y})}{s_1 p(\boldsymbol{\xi}|\boldsymbol{y}) + s_2 g(\boldsymbol{\xi})}$, $f_2(\boldsymbol{\xi}) = \frac{g(\boldsymbol{\xi})}{s_1 p(\boldsymbol{\xi}|\boldsymbol{y}) + s_2 g(\boldsymbol{\xi})}$, and $\mathbb{V}_{g(\boldsymbol{\xi})}\big(f_1(\boldsymbol{\xi})\big) = \int_{\boldsymbol{\Xi}} \left(f_1(\boldsymbol{\xi}) - \mathbb{E}\left[f_1(\boldsymbol{\xi})\right]\right)^2 g(\boldsymbol{\xi})\, \mathrm{d}\boldsymbol{\xi}$ denotes the variance of $f_1(\boldsymbol{\xi})$ with respect to the proposal distribution $g(\boldsymbol{\xi})$ (the variance $\mathbb{V}_{p(\boldsymbol{\xi}|\boldsymbol{y})}\big(f_2(\boldsymbol{\xi})\big)$ is defined analogously); $\rho_{f_2}(0)$ corresponds to the normalized spectral density of the autocorrelated process $f_2(\boldsymbol{\xi})$ at frequency zero.

The unknown variances and expected values are estimated using sample variances and means. For instance, to obtain the variance and expected value with respect to $g(\boldsymbol{\xi})$, we use the n_2 samples for $\tilde{\boldsymbol{\xi}}_j$ from the proposal distribution. Analogously, to evaluate the variance and expected value with respect to the posterior distribution of the unbounded parameters (i.e., $\boldsymbol{\xi}$), we use the second batch of the n_1 to the real line transformed samples $\boldsymbol{\xi}_i^*$ from the posterior distribution which are also used in the iterative updating scheme (i.e., Equation 7). The second term in Equation 15 is adjusted by the normalized spectral density to account for potential autocorrelation in the posterior samples. The spectral density at frequency zero is estimated by fitting an autoregressive model using the `spectrum0.ar` function from the **coda** R package (Plummer *et al.* 2006). Furthermore, the bridge sampling estimate is used as a normalizing constant to compute the normalized posterior density for the unbounded parameter vector $\boldsymbol{\xi}$ which appears in the numerator of $f_1(\boldsymbol{\xi})$ and the denominator of both $f_1(\boldsymbol{\xi})$ and $f_2(\boldsymbol{\xi})$.

Under the assumption that the bridge sampling estimator $\hat{p}(\boldsymbol{y})$ is unbiased, the square root of the expected relative mean-squared error (Equation 14) can be interpreted as the coefficient of variation (i.e., the ratio of the standard deviation and the mean). To facilitate interpretation, the **bridgesampling** package also provides a percentage error which is obtained by simply converting the coefficient of variation to a percentage.

Note that the `error_measures` function can currently not be used to obtain approximate errors for the `"warp3"` method with `repetitions = 1`. The reason is that, in our experience, the approximate errors appear to be unreliable in this case. A reason might be that the warping transformation in Equation 10 adds a minus sign with probability 0.5 to the posterior samples which may alter the autocorrelation structure.

There are two further methods for assessing the uncertainty of the marginal likelihood estimate. These methods are computationally more costly than computing approximate errors, but are available for both the `"normal"` method and the `"warp3"` method. The first option is to set the `repetitions` argument of the `bridge_sampler` function to an integer larger than one. This allows the user to obtain an empirical estimate of the variability across repeated applications of the method. Applying the `error_measures` function to the output of the `bridge_sampler` function that has been obtained with `repetitions` set to an integer large than one provides the user with the minimum/maximum log marginal likelihood estimate across repetitions and the interquartile range of the log marginal likelihood estimates. Note that this procedure assesses the uncertainty of the estimate conditional on the posterior samples, that is, in each repetition new samples are drawn from the proposal distribution, but the posterior samples are fixed across repetitions.

In case the user is able to easily draw new samples from the posterior distribution, the second option is to repeatedly call the `bridge_sampler` function, each time with new posterior samples. This way, the user obtains an empirical assessment of the variability of the estimate which takes into account both uncertainty with respect to the samples from the proposal and also from the posterior distribution.

After having outlined the underlying bridge sampling algorithm, we next demonstrate the capabilities of the **bridgesampling** package using three examples. Additional examples are available as vignettes at: `https://cran.r-project.org/package=bridgesampling`

3. Toy example: Bayesian t-test

We start with a simple statistical example: a Bayesian paired-samples t-test (Jeffreys 1961; Rouder *et al.* 2009; Ly *et al.* 2016; Gronau *et al.* 2017a). We use R's `sleep` data set (Cushny and Peebles 1905) which contains measurements for the effect of two soporific drugs on ten patients. Two different drugs where administered to the same ten patients and the dependent measure was the average number of hours of sleep gained compared to a control night in which no drug was administered. Figure 2 shows the increase in sleep (in hours) of the ten patients for each of the two drugs. To test whether the two drugs differ in effectiveness, we can conduct a Bayesian paired-samples t-test.

The null hypothesis $\mathcal{H}_0$ states that the n difference scores d_i, $i = 1, 2, \ldots, n$, where $n = 10$, follow a normal distribution with mean zero and variance σ^2, that is, $d_i \sim \mathcal{N}(0, \sigma^2)$. The alternative hypothesis $\mathcal{H}_1$ states that the difference scores follow a normal distribution with mean $\mu = \sigma\delta$, where δ denotes the standardized effect size, and variance σ^2, that is, $d_i \sim \mathcal{N}(\sigma\delta, \sigma^2)$. Jeffreys's prior is assigned to the variance σ^2 so that $p(\sigma^2) \propto 1/\sigma^2$ and a zero-centered Cauchy prior with scale parameter $r = 1/\sqrt{2}$ is assigned to the standardized effect size δ (for details, see Rouder *et al.* 2009; Ly *et al.* 2016; Morey and Rouder 2015).

In this example, we are interested in computing the Bayes factor BF_{10} which quantifies how much more likely the data are under $\mathcal{H}_1$ (i.e., there is a difference between the two drugs) than under $\mathcal{H}_0$ (i.e., there is no difference between the two drugs) by using the **bridgesampling** package. For this example, the Bayes factor can also be easily computed using the **BayesFactor** package (Morey and Rouder 2015), allowing us to compare the results from the **bridgesampling** package to the correct answer.

The first step is to obtain posterior samples for both $\mathcal{H}_1$ and $\mathcal{H}_0$. In this example, we use JAGS

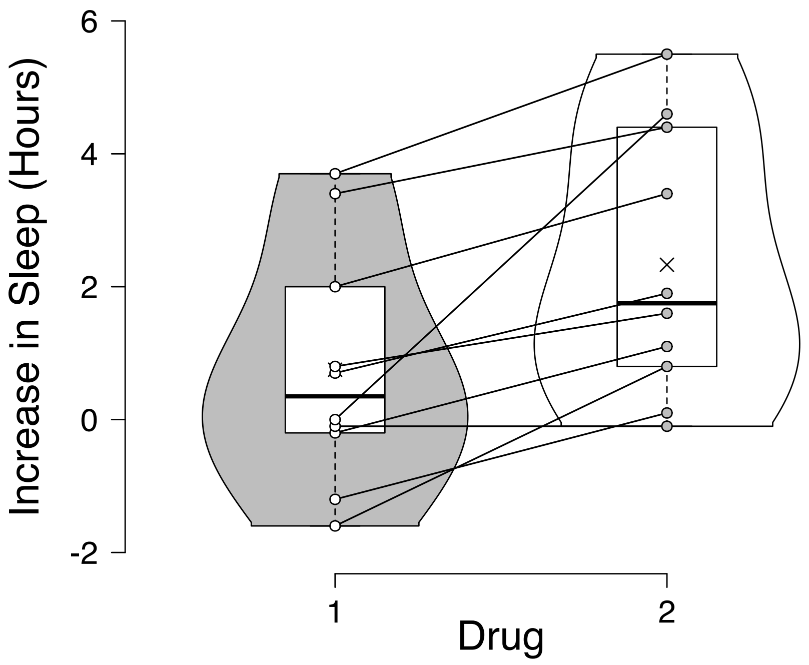

Figure 2: The sleep data set (Cushny and Peebles 1905). The left violin plot displays the distribution of the increase in sleep (in hours) of the ten patients for the first drug, the right violin plot displays the distribution of the increase in sleep (in hours) of the ten patients for the second drug. Boxplots and the individual observations are superimposed. Observations for the same participant are connected by a line. Figure available at `https://tinyurl.com/yalskr23` under CC license `https://creativecommons.org/licenses/by/2.0/`.

in order to sample from the models. The easiest way to implement Jeffreys's prior on the variance is to approximate it by a gamma prior on the precision (i.e., the inverse of σ^2) with shape and rate parameter close to zero. Furthermore, a Cauchy prior on the standardized effect size is implemented as a t distribution with one degree of freedom. The JAGS models[5] can be implemented as character strings as follows:

```
R> code_H1 <-
+ "model {
+   delta ~ dt(0, 1 / r ^ 2, 1)              # prior
+   inv_sigma2 ~ dgamma(0.0001, 0.0001)      # prior
+   sigma <- 1 / sqrt(inv_sigma2)            # convert precision to sigma
+   for (i in 1:n) {
+     d[i] ~ dnorm(sigma * delta, inv_sigma2) # likelihood
+   }
```

[5]Note that in JAGS, normal distributions are parameterized with respect to the precision, that is, one over the variance.

```
+  }"
R> code_H0 <-
+ "model {
+   inv_sigma2 ~ dgamma(0.0001, 0.0001)        # prior
+   for (i in 1:n) {
+     d[i] ~ dnorm(0, inv_sigma2)              # likelihood
+   }
+ }"
```

Posterior samples can then be obtained using the **R2jags** package (Su and Yajima 2015) as follows:[6]

```
R> library("R2jags")
R> data("sleep")
R> y <- sleep$extra[sleep$group == 1]
R> x <- sleep$extra[sleep$group == 2]
R> d <- x - y  # compute difference scores
R> n <- length(d)
R> set.seed(1)
R> jags_H1 <- jags(data = list(d = d, n = n, r = 1 / sqrt(2)),
+                  parameters.to.save = c("delta", "inv_sigma2"),
+                  model.file = textConnection(code_H1), n.chains = 3,
+                  n.iter = 16000, n.burnin = 1000, n.thin = 1)
R> jags_H0 <- jags(data = list(d = d, n = n),
+                  parameters.to.save = "inv_sigma2",
+                  model.file = textConnection(code_H0), n.chains = 3,
+                  n.iter = 16000, n.burnin = 1000, n.thin = 1)
```

Note the relatively large number of posterior samples; reliable estimates for the quantities of interest in testing usually necessitate many more posterior samples than are required for estimation. As a rule of thumb, we suggest that testing requires about an order of magnitude more posterior samples than estimation.

Next, we need to specify functions that take as input a named vector with parameter values and a data object, and return the log of the unnormalized posterior density (i.e., the log of the integrand in Equation 4). These functions are easily specified by inspecting the JAGS model. As a heuristic, one only needs to consider the model code where a "$\sim$" sign appears. The log of the densities on the right-hand side of these "$\sim$" symbols needs to be evaluated for the relevant quantities and then these log density values are summed.[7] Using this heuristic, we obtain the following unnormalized log posterior density functions:

```
R> log_posterior_H1 <- function(pars, data) {
+   delta <- pars["delta"]             # extract parameter
+   inv_sigma2 <- pars["inv_sigma2"]   # extract parameter
+   sigma <- 1 / sqrt(inv_sigma2)      # convert precision to sigma
```

[6]The complete code can be also found on the Open Science Framework: https://osf.io/3yc8q/.

[7]This heuristic assumes that the model does not include other random quantities that are generated during sampling, such as posterior predictives.

```
+   out <-
+     dcauchy(delta, scale = data$r, log = TRUE) +         # prior
+     dgamma(inv_sigma2, 0.0001, 0.0001, log = TRUE) +     # prior
+     sum(dnorm(data$d, sigma * delta, sigma, log = TRUE)) # likelihood
+   return(out)
+ }
R> log_posterior_H0 <- function(pars, data) {
+   inv_sigma2 <- pars["inv_sigma2"]  # extract parameter
+   sigma <- 1 / sqrt(inv_sigma2)     # convert precision to sigma
+   out <-
+     dgamma(inv_sigma2, 0.0001, 0.0001, log = TRUE) +  # prior
+     sum(dnorm(data$d, 0, sigma, log = TRUE))          # likelihood
+   return(out)
+ }
```

Recall that R parameterizes the normal distribution in terms of the standard deviation and provides a density function for the Cauchy distribution so that we do not need to use the t distribution with one degree of freedom as in JAGS.

The final step before we can compute the log marginal likelihoods is to specify named vectors with the parameter bounds:

```
R> lb_H1 <- rep(-Inf, 2)
R> ub_H1 <- rep(Inf, 2)
R> names(lb_H1) <- names(ub_H1) <- c("delta", "inv_sigma2")
R> lb_H1[["inv_sigma2"]] <- 0
R> lb_H0 <- 0
R> ub_H0 <- Inf
R> names(lb_H0) <- names(ub_H0) <- "inv_sigma2"
```

The log marginal likelihoods can then be obtained by calling the `bridge_sampler` function as follows:

```
R> library("bridgesampling")
R> set.seed(12345)
R> bridge_H1 <- bridge_sampler(samples = jags_H1,
+                              log_posterior = log_posterior_H1,
+                              data = list(d = d, n = n, r = 1 / sqrt(2)),
+                              lb = lb_H1, ub = ub_H1)
R> bridge_H0 <- bridge_sampler(samples = jags_H0,
+                              log_posterior = log_posterior_H0,
+                              data = list(d = d, n = n),
+                              lb = lb_H0, ub = ub_H0)
```

We obtain:

```
R> print(bridge_H1)

Bridge sampling estimate of the log marginal likelihood: -27.17103
Estimate obtained in 5 iteration(s) via method "normal".
```

```
R> print(bridge_H0)
```

```
Bridge sampling estimate of the log marginal likelihood: -30.01942
Estimate obtained in 4 iteration(s) via method "normal".
```

Note that by default, the `"normal"` bridge sampling method is used.

Next, we can use the `error_measures` function to obtain approximate percentage errors of the estimates:

```
R> error_measures(bridge_H1)$percentage
```

```
[1] "0.087%"
```

```
R> error_measures(bridge_H0)$percentage
```

```
[1] "0.06%"
```

The small approximate percentage errors indicate that the marginal likelihoods have been estimated reliably. As mentioned before, we can use the `summary` method to obtain a convenient summary of the bridge sampling estimate and the estimation error. For instance, for the bridge sampling estimate for $\mathcal{H}_1$, we obtain:

```
R> summary(bridge_H1)
```

```
Bridge sampling log marginal likelihood estimate
(method = "normal", repetitions = 1):

 -27.17103

Error Measures:

 Relative Mean-Squared Error: 7.564225e-07
 Coefficient of Variation: 0.0008697255
 Percentage Error: 0.087%

Note:
All error measures are approximate.
```

We can compute the Bayes factor for $\mathcal{H}_1$ over $\mathcal{H}_0$ using the `bf` function:

```
R> bf(bridge_H1, bridge_H0)
```

```
The estimated Bayes factor in favor of x1 over x2 is equal to: 17.26001
```

Hence, the observed data are about 17 times more likely under $\mathcal{H}_1$ (which assigns the standardized effect size δ a zero-centered Cauchy prior with scale $r = 1/\sqrt{2}$) than under $\mathcal{H}_0$ (which fixes δ to zero). This is strong evidence for a difference in effectiveness between the two drugs (Jeffreys 1939, Appendix I). Finally, we can confirm that the Bayes factor based on bridge sampling (i.e., $\text{BF}_{10} = 17.260$) is almost identical to the one obtained from the **BayesFactor** package (i.e., $\text{BF}_{10} = 17.259$):

```
R> library("BayesFactor")
R> extractBF(ttestBF(x = x, y = y, paired = TRUE),
+             onlybf = TRUE, log = FALSE)

[1] 17.25888
```

4. A "black box" Stan interface

The previous section demonstrated how the **bridgesampling** package can be used to estimate the marginal likelihood for models coded in JAGS. For custom samplers, the steps needed to compute the marginal likelihood are the same. What is required is (1) an object with posterior samples; (2) a function that computes the log of the unnormalized posterior density; (3) the data; and (4) parameter bounds. A crucial step is the specification of the unnormalized log posterior density function. For applied researchers, this step may be challenging and error-prone, whereas for experienced statisticians it might be tedious and cumbersome, especially for complex models with a hierarchical structure.

In order to facilitate the computation of the marginal likelihood even further, the **bridgesampling** package contains an interface to the generic sampling software Stan (Carpenter *et al.* 2017). Assisted by the **rstan** package (Stan Development Team 2016a), this interface allows users to skip steps 2-4 above. Specifically, users who fit their models in Stan (in a way that retains the constants, as is detailed below) can obtain an estimate of the marginal likelihood by simply passing the `stanfit` object to the `bridge_sampler` function.

The implementation of this "black box" functionality profited from the fact that, just as the **bridgesampling** package, Stan's No-U-Turn sampler internally operates on unconstrained parameters (Hoffman and Gelman 2014; Stan Development Team 2017). The **rstan** package provides access to these unconstrained parameters and the corresponding log of the unnormalized posterior density. This means that users can fit models with parameter types that have more complicated constraints than those currently built into **bridgesampling** (e.g., probability vectors and covariance/correlation matrices) without having to hand-code the appropriate transformations.

As mentioned above, in order to use the **bridgesampling** package in combination with Stan the models need to be implemented in a way that retains the constants. This can be achieved relatively easily: instead of writing, for instance, `y ~ normal(mu, sigma)` or `y ~ bernoulli(theta)`, one needs to write

```
target += normal_lpdf(y | mu, sigma);
```

and

```
target += bernoulli_lpmf(y | theta);
```

That is, one starts with the fixed expression `target +=` which is then followed by the name of the distribution (e.g., `normal`). The name of the distribution is followed by `_lpdf` for continuous distributions and `_lpmf` for discrete distributions. Finally, in parentheses, there is the variable that was to the left of the "`~`" sign (here, `y`), then a "`|`" sign, and finally the arguments of the distribution. This achieves that the user specifies the log target density (in this case, the log of the unnormalized posterior density) in a way that retains the constants of the involved distributions.

Note that in case the distributions are truncated, the user needs to code the correct renormalization. For instance, a normal distribution with upper truncation at `upper` is implemented as follows

```
target += normal_lpdf(y | mu, sigma) - normal_lcdf(upper | mu, sigma);
```

where the function `normal_lcdf` yields the log of the cumulative distribution function (cdf) of the normal distribution. Likewise, a normal distribution with lower truncation at `lower` is obtained as

```
target += normal_lpdf(y | mu, sigma) - normal_lccdf(lower | mu, sigma);
```

where `normal_lccdf` yields the log of the complementary cumulative distribution function (ccdf) of the normal distribution (i.e., the log of one minus the cumulative distribution function of the normal distribution). A normal distribution with lower truncation point `lower` and upper truncation point `upper` can be implemented as follows:

```
target += normal_lpdf(y | mu, sigma) -
          log_diff_exp(normal_lcdf(upper | mu, sigma),
                       normal_lcdf(lower | mu, sigma));
```

where `log_diff_exp(a, b)` is a numerically more stable version of the operation $\log(\exp(a) - \exp(b))$. Note that when implementing a truncated distribution, it is of course also important to give the variable of interest the correct bounds. For instance, for the last example where `y` has a lower truncation at `lower` and an upper truncation at `upper` the variable `y` should be declared as

```
real<lower = lower, upper = upper> y;
```

For more details about how to implement truncated distributions in Stan we refer the user to the Stan manual (Stan Development Team 2017, section 5.3, "Truncated Distributions").

In sum, the **bridgesampling** package enables users to obtain an estimate of the marginal likelihood for any Stan model (programmed to retain the constants) simply by passing the `stanfit` object to the `bridge_sampler` function. Next we demonstrate this functionality using two prototypical examples in Bayesian model selection.

4.1. Stan example 1: Bayesian GLMM

The first example features a generalized linear mixed model (GLMM) applied to the turtles data set (Janzen *et al.* 2000).[8] This data set is included in the **bridgesampling** package and contains information about 244 newborn turtles from 31 different clutches. For each turtle, the data set includes information about survival status (0 = died, 1 = survived), birth weight

[8]Data were obtained from Overstall and Forster (2010) and made available in the **bridgesampling** package with permission from the original authors.

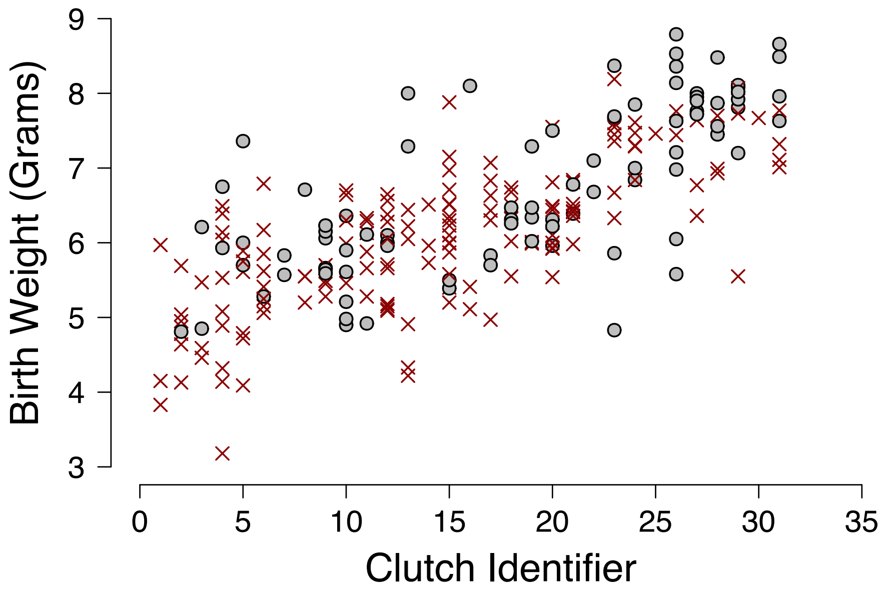

Figure 3: Data for 244 newborn turtles (Janzen *et al.* 2000). Birth weight is plotted against clutch membership. The clutches have been ordered according to their mean birth weight. Dots indicate turtles who survived and red crosses indicate turtles who died. Figure inspired by Sinharay and Stern (2005). Figure available at `https://tinyurl.com/yagfxrbw` under CC license `https://creativecommons.org/licenses/by/2.0/`.

in grams, and clutch (family) membership (indicated by a number between one and 31). Figure 3 displays a scatterplot of clutch membership and birth weight. The clutches have been ordered according to mean birth weight. Dots indicate turtles who survived and red crosses indicate turtles who died. This data set has been analyzed in the context of Bayesian model selection before, allowing us to compare the results from the **bridgesampling** package to the results reported in the literature (e.g., Sinharay and Stern 2005; Overstall and Forster 2010).

Here we focus on the model comparison that was conducted in Sinharay and Stern (2005). The data set was analyzed using a probit regression model of the form:

$$\begin{aligned} y_i &\sim \text{Bernoulli}(\Phi(\alpha_0 + \alpha_1 x_i + b_{\text{clutch}_i})), & i &= 1, 2, \ldots, N \\ b_j &\sim \mathcal{N}(0, \sigma^2), & j &= 1, 2, \ldots, C, \end{aligned} \tag{16}$$

where y_i denotes the survival status of the i-th turtle (i.e., $0 =$ died, $1 =$ survived), x_i denotes the birth weight (in grams) of the i-th turtle, $\text{clutch}_i \in \{1, 2, \ldots, C\}$, $i = 1, 2, \ldots, N$, indicates the clutch to which the i-th turtle belongs, C denotes the number of clutches, and b_{clutch_i} denotes the random effect for the clutch to which the i-th turtle belongs. Sinharay and Stern (2005) investigated the question whether there is an effect of clutch membership, that is, they tested the null hypothesis $\mathcal{H}_0 : \sigma^2 = 0$. The following priors where assigned to the model parameters:

$$\begin{aligned}\alpha_0 &\sim \mathcal{N}(0, 10),\\ \alpha_1 &\sim \mathcal{N}(0, 10),\\ p(\sigma^2) &= \left(1+\sigma^2\right)^{-2}.\end{aligned} \tag{17}$$

Sinharay and Stern (2005) computed the Bayes factor in favor of the null hypothesis $\mathcal{H}_0 : \sigma^2 = 0$ versus the alternative hypothesis $\mathcal{H}_1 : p(\sigma^2) = \left(1+\sigma^2\right)^{-2}$ using different methods and they reported a "true" Bayes factor of $\text{BF}_{01} = 1.273$ (based on extensive numerical integration). Here we implement the above model in Stan and examine the extent to which we can reproduce the Bayes factor using the **bridgesampling** package.

The Stan models corresponding to $\mathcal{H}_0$ and $\mathcal{H}_1$ can be implemented as character strings as follows:

```
R> H0_code <-
+ "data {
+   int<lower = 1> N;
+   int<lower = 0, upper = 1> y[N];
+   real<lower = 0> x[N];
+ }
+ parameters {
+   real alpha0_raw;
+   real alpha1_raw;
+ }
+ transformed parameters {
+   real alpha0 = sqrt(10.0) * alpha0_raw;
+   real alpha1 = sqrt(10.0) * alpha1_raw;
+ }
+ model {
+   target += normal_lpdf(alpha0_raw | 0, 1);   // prior
+   target += normal_lpdf(alpha1_raw | 0, 1);   // prior
+   for (i in 1:N) {                            // likelihood
+     target += bernoulli_lpmf(y[i] | Phi(alpha0 + alpha1 * x[i]));
+   }
+ }"
R> H1_code <-
+ "data {
+   int<lower = 1> N;
+   int<lower = 0, upper = 1> y[N];
+   real<lower = 0> x[N];
+   int<lower = 1> C;
+   int<lower = 1, upper = C> clutch[N];
+ }
+ parameters {
+   real alpha0_raw;
+   real alpha1_raw;
+   vector[C] b_raw;
+   real<lower = 0> sigma2;
+ }
+ transformed parameters {
+   vector[C] b;
+   real<lower = 0> sigma = sqrt(sigma2);
+   real alpha0 = sqrt(10.0) * alpha0_raw;
+   real alpha1 = sqrt(10.0) * alpha1_raw;
+   b = sigma * b_raw;
+ }
+ model {
+   target += - 2 * log(1 + sigma2); // p(sigma2) = 1 / (1 + sigma2) ^ 2
+   target += normal_lpdf(alpha0_raw | 0, 1);   // prior
+   target += normal_lpdf(alpha1_raw | 0, 1);   // prior
+   target += normal_lpdf(b_raw | 0, 1);        // random effects
+   for (i in 1:N) {                            // likelihood
+     target += bernoulli_lpmf(y[i] | Phi(alpha0 + alpha1 * x[i] +
+                                         b[clutch[i]]));
+   }
+ }"
```

Note that the prior on σ^2 is a distribution that is not implemented in Stan. Furthermore, we use the non-centered parameterization for α_0, α_1, and b_j for better sampling performance which is the reason why the model code features the parameters `alpha0_raw`, `alpha1_raw`, and `b_raw`. This makes use of the fact that when a parameter such as α_0 is assigned a distribution that belongs to the location-scale family – here, the normal distribution $\mathcal{N}(\mu, \sigma^2)$, where $\mu = 0$ and $\sigma^2 = 10$ – it can be reparameterized as $\alpha_0 = \mu + \sigma \alpha_0^{\text{raw}}$, where α_0^{raw} is in case of a normal prior distributed as $\alpha_0^{\text{raw}} \sim \mathcal{N}(0, 1)$ (see also Stan Development Team 2017, section 27.6).

The next step is to run Stan and obtain the posterior samples:[9]

```
R> library("bridgesampling")
R> library("rstan")
R> data("turtles")
R> set.seed(1)
R> stanfit_H0 <- stan(model_code = H0_code,
+                    data = list(y = turtles$y,
+                    x = turtles$x, N = nrow(turtles)),
+                    iter = 15500, warmup = 500,
+                    chains = 4, seed = 1)
R> stanfit_H1 <- stan(model_code = H1_code,
+                    data = list(y = turtles$y,
+                    x = turtles$x, N = nrow(turtles),
+                    C = max(turtles$clutch),
+                    clutch = turtles$clutch),
```

[9]The complete R code can be found on the Open Science Framework (https://osf.io/3yc8q/) and is also available at `?turtles`. Note that the results are dependent on the compiler and the optimization settings. Thus, even with identical seeds results can differ slightly from the ones reported here.

```
+                       iter = 15500, warmup = 500,
+                       chains = 4, seed = 1)
```

With these Stan objects in hand, estimates of the log marginal likelihoods are obtained by simply passing the objects to the `bridge_sampler` function:

```
R> set.seed(1)
R> bridge_H0 <- bridge_sampler(stanfit_H0)
R> bridge_H1 <- bridge_sampler(stanfit_H1)
```

The Bayes factor in favor of $\mathcal{H}_0$ over $\mathcal{H}_1$ can then be obtained as follows:

```
R> bf(bridge_H0, bridge_H1)

The estimated Bayes factor in favor of x1 over x2 is equal to: 1.27438
```

This value is close to that of 1.273 reported in Sinharay and Stern (2005). The data are only slightly more likely under $\mathcal{H}_0$ than under $\mathcal{H}_1$, suggesting that the data do not warrant strong claims about whether or not clutch membership affects survival.

The precision of the estimates for the marginal likelihoods can be obtained as follows:

```
R> error_measures(bridge_H0)$percentage

 [1] "0.00972%"

R> error_measures(bridge_H1)$percentage

[1] "0.342%"
```

These error percentages indicate that both marginal likelihoods have been estimated accurately, but – as expected – the marginal likelihood for the more complicated model with random effects (i.e., $\mathcal{H}_1$) has the larger estimation error.

4.2. Stan example 2: Bayesian factor analysis

The second example concerns Bayesian factor analysis. In particular, we determine the number of relevant latent factors by implementing the Bayesian factor analysis model proposed by Lopes and West (2004). The model assumes that there are t, $t = 1, 2, \ldots, T$, observations on each of m variables. That is, each observation $\boldsymbol{y}_t$ is an m-dimensional vector. The k-factor model – where k denotes the number of factors – relates each of the T observations $\boldsymbol{y}_t$ to a latent k-dimensional vector $\boldsymbol{f}_t$ which contains for observation t the values on the latent factors, as follows:[10]

$$\begin{aligned} \boldsymbol{y}_t \mid \boldsymbol{f}_t &\sim \mathcal{N}_m\left(\boldsymbol{\beta} \boldsymbol{f}_t, \boldsymbol{\Sigma}\right) \\ \boldsymbol{f}_t &\sim \mathcal{N}_k\left(\boldsymbol{0}_k, \boldsymbol{I}_k\right), \end{aligned} \tag{18}$$

[10]Note that the model assumes that the observations are zero-centered.

where $\boldsymbol{\beta}$ denotes the $m \times k$ factor loadings matrix[11], $\boldsymbol{\Sigma} = \text{diag}\left(\sigma_1^2, \sigma_2^2, \ldots, \sigma_m^2\right)$ denotes the $m \times m$ diagonal matrix with residual variances, $\mathbf{0}_k$ denotes a k-dimensional vector with zeros, and $\boldsymbol{I}_k$ denotes the $k \times k$ identity matrix. Hence, conditional on the latent factors, the observations on the m variables are assumed to be uncorrelated with each other. Marginally, however, the observations are usually not uncorrelated and they are distributed as

$$\boldsymbol{y}_t \sim \mathcal{N}_m\left(\mathbf{0}_m, \boldsymbol{\Omega}\right), \tag{19}$$

where $\boldsymbol{\Omega} = \boldsymbol{\beta}\boldsymbol{\beta}^\top + \boldsymbol{\Sigma}$.

Here we reanalyze a data set that contains the changes in monthly international exchange rates for pounds sterling from January 1975 to December 1986 (West and Harrison 1997, pp. 612-615). Currencies tracked are US Dollar (US), Canadian Dollar (CAN), Japanese Yen (JAP), French Franc (FRA), Italian Lira (ITA), and the (West) German Mark (GER). Figure 4 displays the data.[12] Using different computational methods, including bridge sampling, Lopes and West (2004) estimated the marginal likelihoods and posterior model probabilities for a factor model with one, two, and three factors. As before, this allows us to compare the results from the **bridgesampling** package to the results reported in the literature. To identify the model, the factor loading matrix $\boldsymbol{\beta}$ is constrained to be lower-triangular (Lopes and West 2004). The diagonal elements of $\boldsymbol{\beta}$ are constrained to be positive by assigning them standard half-normal priors with lower truncation point zero: $\beta_{jj} \sim \mathcal{N}(0,1)_{T(0,)}$, $j = 1, 2, \ldots, k$, and the lower-diagonal elements are assigned standard normal priors. The residual variances are assigned inverse-gamma priors of the form $\sigma_i^2 \sim \text{Inverse-Gamma}(\nu/2, \nu s^2/2)$, $i = 1, 2, \ldots, m$, where $\nu = 2.2$ and $\nu s^2 = 0.1$ (for details, see Lopes and West 2004).

The first step in our reanalysis is to specify the Stan model as a character string:

```
R> model_code <-
+ "data {
+   int<lower = 1> T;   // number of observations
+   int<lower = 1> m;   // number of variables
+   int<lower = 1> k;   // number of factors
+   matrix[T,m] Y;      // data matrix
+ }
+ transformed data {
+   int<lower = 1> r;
+   vector[m] zeros;
+   r = m * k - k * (k - 1) / 2;     // number of non-zero factor loadings
+   zeros = rep_vector(0.0, m);
+ }
+ parameters {
+   real beta_lower[r - k];          // lower-diagonal elements of beta
+   real<lower = 0> beta_diag [k];  // diagonal elements of beta
+   vector<lower = 0>[m] sigma2;     // residual variances
+ }
```

[11]We use the original notation by Lopes and West (2004) who denoted the factor loadings matrix with a lower-case letter. In the remainder of the article, matrices are denoted by upper-case letters.

[12]Each series has been standardized with respect to its sample mean and standard deviation. These standardized data are included in the **bridgesampling** package.

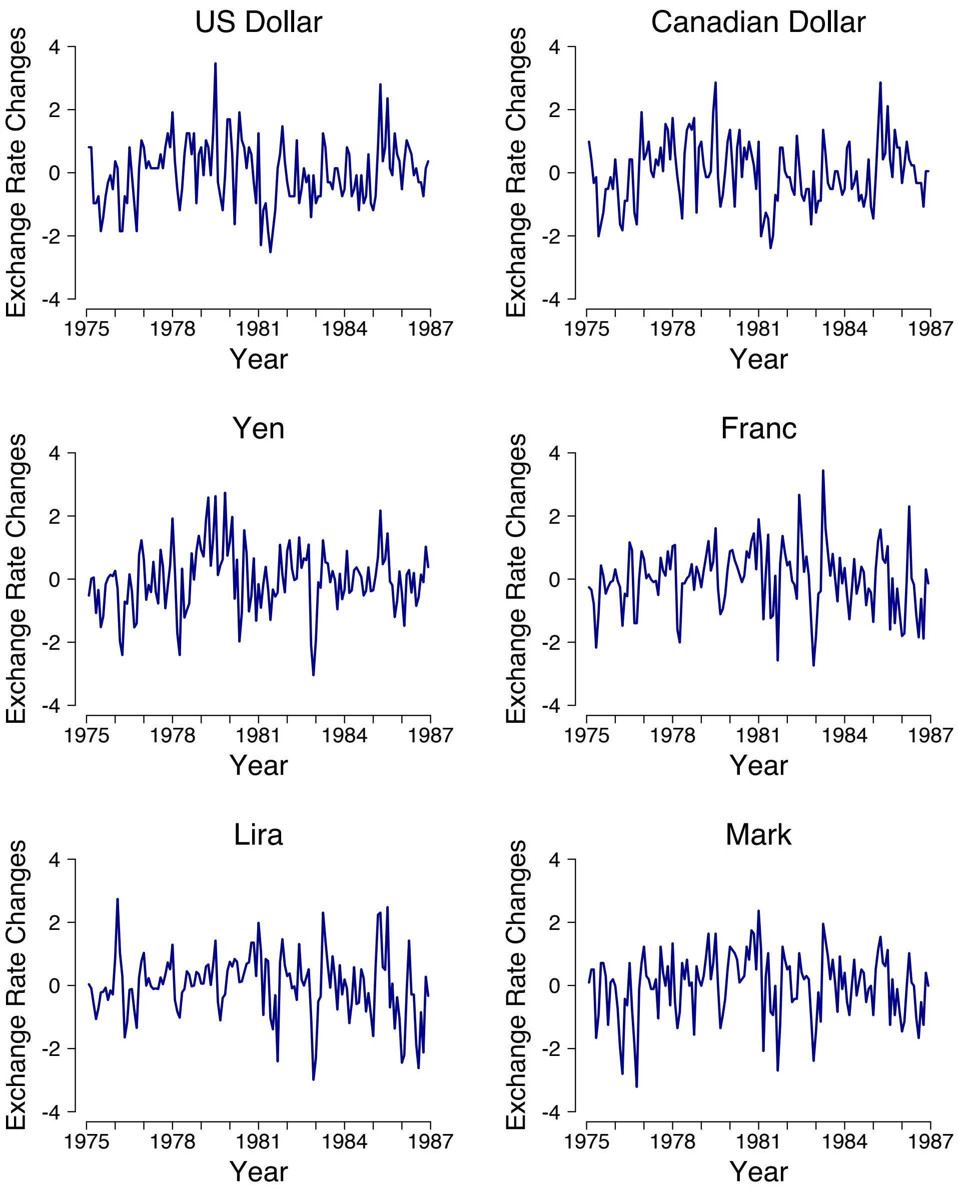

Figure 4: Changes in monthly international exchange rates for pounds sterling from January 1975 to December 1986 (West and Harrison 1997, pp. 612 – 615). Currencies tracked are US Dollar (US), Canadian Dollar (CAN), Japanese Yen (JAP), French Franc (FRA), Italian Lira (ITA), and the (West) German Mark (GER). Each series has been standardized with respect to its sample mean and standard deviation. Figure reproduced from Lopes and West (2004). Figure available at `https://tinyurl.com/ybtdddyv` under CC license `https://creativecommons.org/licenses/by/2.0/`.

```
+ transformed parameters {
+   matrix[m,k] beta;
+   cov_matrix[m] Omega;
+   {
+     int index_lower = 1;
+     for (j in 1:k) { // construct lower-triangular factor loadings matrix
+       for (i in 1:m) {
+         if (i == j) {
+           beta[j,j] = beta_diag[j];
+         } else if (i >= j) {
+           beta[i,j] = beta_lower[index_lower];
+           index_lower = index_lower + 1;
+         } else {
+           beta[i,j] = 0.0;
+         }
+       }
+     }
+   }
+   Omega = beta * beta' + diag_matrix(sigma2);
+ }
+ model {
+   target += normal_lpdf(beta_diag | 0, 1) - k * normal_lccdf(0 | 0, 1);
+   target += normal_lpdf(beta_lower | 0, 1);                // prior
+   target += inv_gamma_lpdf(sigma2 | 2.2 / 2.0, 0.1 / 2.0); // prior
+   for(t in 1:T) {
+     target += multi_normal_lpdf(Y[t] | zeros, Omega);      // likelihood
+   }
+ }"
```

Note that in order to implement the half-normal priors on the diagonal elements of $\boldsymbol{\beta}$ correctly, the prior densities need to be renormalized. Each of the k standard normal priors needs to be renormalized by the area larger than zero which is achieved by subtracting `k * normal_lccdf(0 | 0, 1)`. This expression was chosen to illustrate the general case (i.e., when the mean of the normal distribution may be different from zero). When the mean is equal to zero, as in this example, the same could be achieved by replacing `k * normal_lccdf(0 | 0, 1)` by `k * log(0.5)`. We can then fit the three models corresponding to $k = 1$, $k = 2$, and $k = 3$ latent factors and estimate the log marginal likelihoods using **bridgesampling** as follows:[13]

```
R> library("rstan")
R> library("bridgesampling")
R> data("ier")
R> cores <- 4
R> options(mc.cores = cores)                    # for parallel MCMC chains
R> model <- stan_model(model_code = model_code) # compile model
```

[13]Note that we specify initial values using a custom `init_fun` function. This function may need to be changed for different applications. Furthermore, it is strongly advised to check that the chains have indeed converged since we sometimes encountered convergence issues with this model. The complete R code can be found on the Open Science Framework (`https://osf.io/3yc8q/`) and is also available at `?ier`. Note that the results are dependent on the compiler and the optimization settings. Thus, even with identical seeds results can differ slightly from the ones reported here.

```
R> init_fun <- function(nchains, k, m) {        # generates starting values
+   r <- m * k - k * (k - 1) / 2
+   out <- vector("list", nchains)
+   for (i in seq_len(nchains)) {
+     beta_lower <- array(runif(r - k, 0.05, 1), dim = r - k)
+     beta_diag <- array(runif(k, .05, 1), dim = k)
+     sigma2 <- array(runif(m, .05, 1.5), dim = m)
+     out[[i]] <- list(beta_lower = beta_lower,
+                      beta_diag = beta_diag,
+                      sigma2 = sigma2)
+   }
+   return(out)
+ }
R> set.seed(1)
R> stanfit <- bridge <- vector("list", 3)
R> for (k in 1:3) {
+   stanfit[[k]] <- sampling(model,
+                            data = list(Y = ier, T = nrow(ier),
+                                        m = ncol(ier), k = k),
+                            iter = 11000, warmup = 1000, chains = 4,
+                            init = init_fun(nchains = 4, k = k,
+                                            m = ncol(ier)),
+                            cores = cores, seed = 1)
+   bridge[[k]] <- bridge_sampler(stanfit[[k]], method = "warp3",
+                                 repetitions = 10, cores = cores)
+ }
```

Note that in this example, we use the `"warp3"` method instead of the `"normal"` method. Furthermore, since the `error_measures` function cannot be used when the estimate has been obtained using `method = "warp3"` with `repetitions = 1`, we set `repetitions = 10` to obtain an empirical estimate of the estimation uncertainty (conditional on the posterior samples). We also select parallel computation by setting `cores = 4`. The `summary` method provides a convenient overview of the estimate and the estimation uncertainty. For instance, for the 2-factor model, we obtain as output:

```
R> summary(bridge[[2]])

Bridge sampling log marginal likelihood estimate
(method = "warp3", repetitions = 10):

 -903.4509

Error Measures:

 Min: -903.4562
 Max: -903.4472
 Interquartile Range: 0.002325984
```

```
Note:
All error measures are based on 10 estimates.
```

Table 2 displays for each of the three factor models (i.e., $k = 1$, $k = 2$, $k = 3$) the median log marginal likelihood (logml) across repetitions, the minimum/maximum log marginal likelihood across repetitions, and the log marginal likelihood value reported in Lopes and West (2004) based on bridge sampling. Note that the negative infinity reported by Lopes and West (2004) might be due to a numerical problem. For the 1-factor model and the 2-factor model, the log marginal likelihoods obtained via **bridgesampling** are very similar to the ones reported in Lopes and West (2004). Furthermore, the narrow range of the estimates indicates that the estimation uncertainty is small (conditional on the posterior samples, as described above).

To examine the support for the three different models (i.e., different numbers of latent factors), we can use the `post_prob` function to compute posterior model probabilities. By default, the function assumes that all models are equally likely a priori; this can be adjusted using the `prior_prob` argument. Furthermore, the `model_names` argument can optionally be used to provide names for the models. Here we use the default of equal prior model probabilities and we obtain:

```
R> post_prob(bridge[[1]], bridge[[2]], bridge[[3]],
+            model_names = c("k = 1", "k = 2", "k = 3"))

             k = 1     k = 2     k = 3
 [1,] 6.283088e-49 0.8469707 0.1530293
 [2,] 6.310609e-49 0.8515854 0.1484146
 [3,] 6.382763e-49 0.8593477 0.1406523
 [4,] 6.492560e-49 0.8731465 0.1268535
 [5,] 6.546073e-49 0.8772237 0.1227763
 [6,] 6.430437e-49 0.8673352 0.1326648
 [7,] 6.401000e-49 0.8660352 0.1339648
 [8,] 6.352043e-49 0.8565560 0.1434440
 [9,] 6.433428e-49 0.8659839 0.1340161
[10,] 6.447604e-49 0.8650544 0.1349456
```

Each row presents the posterior model probabilities based on one repetition of the bridge sampling procedure for all three models (i.e., each row sums to one). Hence, there are as many rows as `repetitions`.[14] The 2-factor model receives most support from the observed data. This is in line with Lopes and West (2004), who also preferred the 2-factor model[15]; based on the factor loadings, they proposed the presence of a North American factor and a European Union factor.

5. Discussion

This paper introduced **bridgesampling**, an R package for computing marginal likelihoods,

[14]Note that the output of the `post_prob` function can be directly passed to the `boxplot` function which allows one to visualize the estimation uncertainty in the posterior model probabilities across repetitions.

[15]Note that Lopes and West (2004) report a posterior model probability of 1 for the 2-factor model. However, this estimate may be inflated by the infinite log marginal likelihood value for the 3-factor model.

Table 2: Log marginal likelihood (logml) estimates for the $k = 1$, $k = 2$, and $k = 3$ factor model. The rightmost column displays the values based on bridge sampling reported in Lopes and West (2004).

Number of Factors	Median Logml	Min Logml	Max Logml	Lopes & West
$k = 1$	-1014.273	-1014.274	-1014.271	-1014.5
$k = 2$	-903.451	-903.456	-903.447	-903.7
$k = 3$	-905.312	-905.423	-905.163	$-\infty$

Bayes factors, posterior model probabilities, and normalizing constants in general. We have demonstrated how researchers can use **bridgesampling** to conduct Bayesian model comparisons in a generic, user-friendly way: researchers need only provide posterior samples, a function that computes the log of the unnormalized posterior density, the data, and lower and upper bounds for the parameters. Furthermore, we have described the Stan interface which makes it even easier to obtain the marginal likelihood: researchers need only provide a `stanfit` object and the **bridgesampling** package will automatically produce an estimate of the log marginal likelihood. In other words, the **bridgesampling** package makes it possible to obtain marginal likelihood estimates for any model that can be implemented in Stan (in a way that retains the constants). By combining the Stan state-of-the-art No-U-Turn sampler with **bridgesampling**, researchers are provided with a general purpose, easy-to-use computational solution to the challenging task of comparing complex Bayesian models.

Even though the **bridgesampling** package enables researchers to compute the marginal likelihoods in an almost black-box manner, this does not imply that the user can mindlessly exploit the package functionality to conduct Bayesian model comparisons. As is apparent from Equation 4, Bayesian model comparisons depend on the choice of the parameter prior distribution. Crucially, the prior distribution has a lasting influence on the results. Hence, meaningful Bayesian model comparisons require that researchers carefully consider their parameter prior distribution (e.g., Lee and Vanpaemel in press), engage in sensitivity analyses, or use default prior choices that have certain desirable properties (e.g., Jeffreys 1961; Bayarri *et al.* 2012; Ly *et al.* 2016).[16] Thus, the **bridgesampling** package removes the computational hurdle of obtaining the marginal likelihood, thereby allowing researchers to spend more time and effort on the specification of meaningful prior distributions.

Another note of caution is that the **bridgesampling** package does not check the validity of the posterior samples, but simply executes the algorithm. Hence, in case of MCMC sampling it is crucial that researchers confirm that the chains have converged to the joint posterior distribution. In addition, researchers need to make sure that the model does not contain any discrete parameters since those are currently not supported. This may sound more restrictive than it is. In practice the solution is to marginalize out the discrete parameters, something that is often possible. Note the similarity to Stan which also deals with discrete parameters by marginalizing them out (Stan Development Team 2017, section 15). Furthermore, as demonstrated in the examples, for conducting model comparisons based on bridge sampling,

[16]Note that in the first example (i.e., the Bayesian t-test) we have used prior distributions which lead to these desirable properties. However, in the second and third example, we simply used the prior distributions that have been used in the literature so that we could compare our results to the reported results.

the number of posterior samples often needs to be an order of magnitude larger than for estimation. This of course depends on a number of factors such as the complexity of the model of interest, the number of posterior samples that one usually uses for estimation, the posterior sampling algorithm used, and also the accuracy of the marginal likelihood estimate that one desires to achieve.

The accuracy of the estimate is governed not only be the number of samples, but also by the overlap between the posterior and the proposal distribution (e.g., Meng and Wong 1996; Meng and Schilling 2002). The **bridgesampling** package attempts to maximize this overlap by (1) focusing on one marginal likelihood at a time which allows one to use a convenient proposal distribution which closely resembles the posterior distribution, (2) using a proposal distribution which matches the mean vector and covariance matrix of the posterior samples (i.e., `method = "normal"`) or additionally also the skewness (i.e., `method = "warp3"`). Nevertheless, in case the posterior distribution exhibits multiple modes, the overlap of the two distributions may still be subject to improvement. The development of efficient bridge sampling variants for these cases is subject to ongoing research (e.g., Wang and Meng 2016; Frühwirth–Schnatter 2004).

It should also be kept in mind that there may be cases in which the bridge sampling procedure may not be the ideal choice for conducting Bayesian model comparisons. For instance, when the models are nested it might be faster and easier to use the Savage-Dickey density ratio (Dickey and Lientz 1970; Wagenmakers *et al.* 2010). Another example is when the comparison of interest concerns a very large model space, and a separate bridge sampling based computation of marginal likelihoods may take too much time. In this scenario, Reversible Jump MCMC (Green 1995) may be more appropriate. The downside of Reversible Jump MCMC is that it is usually problem-specific and cannot easily be applied in a generic fashion to different nested and non-nested model comparison problems (but see Gelling *et al.* 2017). The goal with the **bridgesampling** package, however, was exactly that: to provide users with a generic way of computing marginal likelihoods which can in principle be applied to any Bayesian model comparison problem.

In sum, the **bridgesampling** package provides a generic, accurate, easy-to-use, automatic, and fast way of computing marginal likelihoods and conducting Bayesian model comparisons. With the computational challenge all but overcome, researchers can spend more time and effort on addressing the conceptual challenge that comes with Bayesian model comparisons: specifying prior distributions that are either robust or meaningful.

6. Acknowledgements

This research was supported by a Netherlands Organisation for Scientific Research (NWO) grant to QFG (406.16.528), by an NWO Vici grant to EJW (016.Vici.170.083), and by the Berkeley Initiative for Transparency in the Social Sciences, a program of the Center for Effective Global Action (CEGA), with support from the Laura and John Arnold Foundation.

References

Bayarri MJ, Berger JO, Forte A, García-Donato G (2012). "Criteria for Bayesian Model

Choice With Application to Variable Selection." *The Annals of Statistics*, **40**, 1550–1577.

Carlin BP, Chib S (1995). "Bayesian Model Choice via Markov Chain Monte Carlo Methods." *Journal of the Royal Statistical Society B*, **57**, 473–484.

Carpenter B, Gelman A, Hoffman M, Lee D, Goodrich B, Betancourt M, Brubaker M, Guo J, Li P, Riddell A (2017). "Stan: A Probabilistic Programming Language." *Journal of Statistical Software*, **76**, 1–32.

Chib S (1995). "Marginal Likelihood from the Gibbs Output." *Journal of the American Statistical Association*, **90**, 1313–1321.

Cushny AR, Peebles AR (1905). "The Action of Optical Isomers: II Hyoscines." *The Journal of Physiology*, **32**, 501–510.

Denwood MJ (2016). "**runjags**: An R Package Providing Interface Utilities, Model Templates, Parallel Computing Methods and Additional Distributions for MCMC Models in JAGS." *Journal of Statistical Software*, **71**(9), 1–25. doi:10.18637/jss.v071.i09.

Dickey JM, Lientz BP (1970). "The Weighted Likelihood Ratio, Sharp Hypotheses about Chances, the Order of a Markov Chain." *The Annals of Mathematical Statistics*, **41**, 214–226.

Etz A, Wagenmakers EJ (2017). "J.B.S Haldane's Contribution to the Bayes Factor Hypothesis Test." *Statistical Science*, **32**, 313–329.

Frühwirth–Schnatter S (2004). "Estimating Marginal Likelihoods for Mixture and Markov Switching Models Using Bridge Sampling Techniques." *Econometrics Journal*, **7**, 143–167.

Gamerman D, Lopes HF (2006). *Markov Chain Monte Carlo: Stochastic Simulation for Bayesian Inference*. Chapman & Hall/CRC, Boca Raton, FL.

Gelling N, Schofield MR, Barker RJ (2017). ***rjmcmc**: Reversible-Jump MCMC Using Post-Processing*. R package version 0.3.2, URL https://CRAN.R-project.org/package=rjmcmc.

Gelman A, Meng XL (1998). "Simulating Normalizing Constants: From Importance Sampling to Bridge Sampling to Path Sampling." *Statistical Science*, **13**, 163–185.

Green PJ (1995). "Reversible Jump Markov Chain Monte Carlo Computation and Bayesian Model Determination." *Biometrika*, **82**, 711–732.

Gronau QF, Ly A, Wagenmakers EJ (2017a). "Informed Bayesian T-Tests." *Manuscript submitted for publication and uploaded to arXiv*. URL https://arxiv.org/abs/1704.02479.

Gronau QF, Sarafoglou A, Matzke D, Ly A, Boehm U, Marsman M, Leslie DS, Forster JJ, Wagenmakers EJ, Steingroever H (2017b). "A Tutorial on Bridge Sampling." *Journal of Mathematical Psychology*, **81**, 80–97. URL https://doi.org/10.1016/j.jmp.2017.09.005.

Gronau QF, Wagenmakers EJ, Heck DW, Matzke D (2017c). "A Simple Method for Comparing Complex Models: Bayesian Model Comparison for Hierarchical Multinomial Processing Tree Models Using Warp-III Bridge Sampling." *Manuscript submitted for publication and uploaded to PsyArXiv.* URL `https://psyarxiv.com/yxhfm`.

Hankin RKS (2007). "Very Large Numbers in R: Introducing Package **Brobdingnag**." *R News*, **7**.

Hoeting JA, Madigan D, Raftery AE, Volinsky CT (1999). "Bayesian Model Averaging: A Tutorial." *Statistical Science*, **14**, 382–417.

Hoffman MD, Gelman A (2014). "The No-U-Turn Sampler: Adaptively Setting Path Lengths in Hamiltonian Monte Carlo." *Journal of Machine Learning Research*, **15**, 1593–1623.

Janzen FJ, Tucker JK, Paukstis GL (2000). "Experimental Analysis of an Early Life-History Stage: Selection on Size of Hatchling Turtles." *Ecology*, **81**, 2290–2304.

Jefferys WH, Berger JO (1992). "Ockham's Razor and Bayesian Analysis." *American Scientist*, **80**, 64–72.

Jeffreys H (1939). *Theory of Probability.* 1st edition. Oxford University Press, Oxford, UK.

Jeffreys H (1961). *Theory of Probability.* 3rd edition. Oxford University Press, Oxford, UK.

Kass RE, Raftery AE (1995). "Bayes Factors." *Journal of the American Statistical Association*, **90**, 773–795.

Knaus J (2015). ***snowfall**: Easier Cluster Computing (Based on **snow**).* R package version 1.84-6.1, URL `https://CRAN.R-project.org/package=snowfall`.

Lartillot N, Philippe H (2006). "Computing Bayes Factors Using Thermodynamic Integration." *Systematic Biology*, **55**, 195–207.

Lee MD, Vanpaemel W (in press). "Determining Informative Priors for Cognitive Models." *Psychonomic Bulletin & Review.*

Lodewyckx T, Kim W, Lee MD, Tuerlinckx F, Kuppens P, Wagenmakers EJ (2011). "A Tutorial on Bayes Factor Estimation with the Product Space Method." *Journal of Mathematical Psychology*, **55**, 331–347.

Lopes HF, West M (2004). "Bayesian Model Assessment in Factor Analysis." *Statistica Sinica*, **14**, 41–67.

Ly A, Verhagen AJ, Wagenmakers EJ (2016). "Harold Jeffreys's Default Bayes Factor Hypothesis Tests: Explanation, Extension, and Application in Psychology." *Journal of Mathematical Psychology*, **72**, 19–32.

Meng XL, Schilling S (2002). "Warp Bridge Sampling." *Journal of Computational and Graphical Statistics*, **11**, 552–586.

Meng XL, Wong WH (1996). "Simulating Ratios of Normalizing Constants via a Simple Identity: A Theoretical Exploration." *Statistica Sinica*, **6**, 831–860.

Morey RD, Rouder JN (2015). "**BayesFactor** 0.9.11-1." Comprehensive R Archive Network. URL `http://cran.r-project.org/web/packages/BayesFactor/index.html`.

Myung IJ, Pitt MA (1997). "Applying Occam's Razor in Modeling Cognition: A Bayesian Approach." *Psychonomic Bulletin & Review*, **4**, 79–95.

Overstall AM (2010). *Default Bayesian Model Determination for Generalised Linear Mixed Models*. Ph.D. thesis, University of Southampton. URL `https://eprints.soton.ac.uk/170229/`.

Overstall AM, Forster JJ (2010). "Default Bayesian Model Determination Methods for Generalised Linear Mixed Models." *Computational Statistics & Data Analysis*, **54**, 3269–3288.

Owen A, Zhou Y (2000). "Safe and Effective Importance Sampling." *Journal of the American Statistical Association*, **95**, 135 – 143.

Plummer M (2003). "JAGS: A Program for Analysis of Bayesian Graphical Models Using Gibbs Sampling." In K Hornik, F Leisch, A Zeileis (eds.), *Proceedings of the 3rd International Workshop on Distributed Statistical Computing*. Vienna, Austria.

Plummer M (2016). ***rjags**: Bayesian Graphical Models Using MCMC*. R package version 4-6, URL `https://CRAN.R-project.org/package=rjags`.

Plummer M, Best N, Cowles K, Vines K (2006). "**coda**: Convergence Diagnosis and Output Analysis for MCMC." *R News*, **6**, 7–11.

R Core Team (2016). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL `https://www.R-project.org/`.

Stan Development Team (2016a). "**rstan**: The R interface to Stan." R package version 2.14.1, URL `http://mc-stan.org/`.

Stan Development Team (2016b). "**rstanarm**: Bayesian Applied Regression Modeling via Stan." R package version 2.13.1, URL `http://mc-stan.org/`.

Stan Development Team (2017). *Stan Modeling Language Users Guide and Reference Manual, Version 2.16.0*. URL `http://mc-stan.org`.

Rouder JN, Speckman PL, Sun D, Morey RD, Iverson G (2009). "Bayesian T Tests for Accepting and Rejecting the Null Hypothesis." *Psychonomic Bulletin & Review*, **16**, 225–237.

Sinharay S, Stern HS (2005). "An Empirical Comparison of Methods for Computing Bayes Factors in Generalized Linear Mixed Models." *Journal of Computational and Graphical Statistics*, **14**, 415–435.

Su YS, Yajima M (2015). ***R2jags**: Using R to Run JAGS*. R package version 0.5-7, URL `https://CRAN.R-project.org/package=R2jags`.

Vandekerckhove J, Matzke D, Wagenmakers EJ (2015). "Model Comparison and the Principle of Parsimony." In J Busemeyer, J Townsend, ZJ Wang, A Eidels (eds.), *Oxford Handbook of Computational and Mathematical Psychology*, pp. 300–319. Oxford University Press.

Wagenmakers EJ, Lodewyckx T, Kuriyal H, Grasman R (2010). "Bayesian Hypothesis Testing for Psychologists: A Tutorial on the Savage–Dickey Method." *Cognitive Psychology*, **60**, 158–189.

Wang L, Meng XL (2016). "Warp Bridge Sampling: The Next Generation." *arXiv preprint arXiv:1609.07690*.

West M, Harrison J (1997). *Bayesian Forecasting and Dynamic Models*. 2nd edition. Springer-Verlag, New York.

Affiliation:

Quentin F. Gronau
Department of Psychological Methods
University of Amsterdam
Nieuwe Achtergracht 129 B
1018 WT Amsterdam, The Netherlands
E-mail: Quentin.F.Gronau@gmail.com