R package rjmcmc: The calculation of posterior model probabilities from MCMC output

Nicholas Gelling, Matthew R. Schofield, Richard J. Barker

November 2, 2017

1 Introduction

Discriminating among models can be a difficult problem. There are several options for models fitted using Bayesian inference, including use of Bayes factors, or equivalently posterior model probabilities (Kass and Raftery 1995), information criteria such as WAIC (Watanabe 2010) and cross validation (Arlot et al. 2010). All of these approaches have practical challenges: Bayes factors and posterior model probabilities require either the evaluation of a complex high dimensional integral or specification of a transdimensional algorithm such as reversible jump Markov chain Monte Carlo (RJMCMC); information criteria require an estimate of the effective number of parameters; cross-validation requires burdensome computational effort. Our focus is on the first two of these approaches. We have developed an **R** package that allows posthoc calculation of Bayes factors and posterior model probabilities using Markov chain Monte Carlo (MCMC) output, simplifying a frequently daunting problem.

The Bayes factor of Jeffreys (1935) is central to Bayesian model comparison and features in nearly every textbook on Bayesian inference (e.g. Gelman et al. 2013, Gill 2014). The Bayes factor B_{ij} compares the marginal likelihood for two competing models indexed i and j,

$$B_{ij} = \frac{p(y|M_i)}{p(y|M_j)} = \frac{\int p(y|\theta_i, M_i)p(\theta_i|M_i)d\theta_i}{\int p(y|\theta_j, M_j)p(\theta_j|M_j)d\theta_j},$$

where $p(y|\theta_k, M_k)$ is the likelihood function under model k and $p(\theta_k|M_k)$ is the prior distribution under model k. The random variable M is a model indicator with $M \in \{1, \ldots, K\}$ where K is the number of models considered – for ease of notation, we let M_k refer to the event $M = k$. It is straightforward to compute Bayes factors from posterior model probabilities and vice versa provided the prior model weights are known (Kass and Raftery 1995). This facilitates Bayesian model averaging (Hoeting et al. 1999) allowing for model uncertainty to be accounted for in estimation.

A major limitation in the implementation of Bayes factors and corresponding posterior model probabilities is the difficulty of calculating the marginal likelihood. Approximation is frequently used; for example, the Bayesian Information

Criterion (Schwarz et al. 1978) is an asymptotic approximation to the Bayes factor (Raftery 1986).

MCMC approaches are also available. Carlin and Chib (1995) proposed an MCMC sampler that uses 'pseudo-priors' to facilitate jumping between models while RJMCMC (Green 1995) augmented the model space in order to move between models using bijections. Generating sensible pseudo-priors or augmenting variables for these algorithms is challenging. Gill (2014) noted that reversible jump methodology continues to be an active research area. The **R** package demonstrated here is the first reversible jump package to be released on the Comprehensive **R** Archive Network (CRAN), and offers a general framework for the calculation of Bayes factors and posterior model probabilities from model output.

In Section 2, RJMCMC is discussed further. A Gibbs sampling approach to RJMCMC is also described which allows for post-processing, separating the model fitting and model comparison steps. In Section 3, we introduce the **R** package **rjmcmc**, which implements the Gibbs algorithm, with examples in Section 4. We conclude with a discussion in Section 5.

2 Transdimensional algorithms

Suppose we have data y, a set of models indexed $1, \ldots, K$, and a model-specific parameter vector θ_k for each model, $k = 1, \ldots, K$. If we also assign prior model probabilities $p(M_k)$, $k = 1, \ldots, K$, we can find the posterior model probabilities

$$\frac{p(M_i|y)}{p(M_j|y)} = B_{ij} \times \frac{p(M_i)}{p(M_j)}.$$

RJMCMC (Green 1995) is an approach to avoiding the integral required in finding the posterior model probabilities. A bijection (i.e., an invertible one-to-one mapping) is specified between the parameter spaces of each pair of models; a total of $\binom{K}{2}$ bijections are required. To match dimensions between models, augmenting variables u_k are specified so that $\dim(\theta_k, u_k) = \dim(\theta_j, u_j)$ for $j, k \in \{1, \ldots, K\}$. The augmenting variables do not change the posterior distribution but do affect computational efficiency. Figure 1 gives a stylised visual representation of the sets and bijections involved in RJMCMC.

The RJMCMC algorithm proceeds as follows. At iteration i of the Markov chain, a model M_h^* is proposed with the current value denoted $M_j^{(i-1)}$. Proposed parameter values for model M_h^* are found using the bijection $f_{jh}(\cdot)$

$$(\theta_h^*, u_h^*) = f_{jh}(\theta_j^{(i-1)}, u_j^{(i-1)}).$$

The joint proposal is then accepted using a Metropolis step (Green 1995). In defining a bijection, we can incorporate any known relationships between the parameters of two models and potentially simplify the relationship between the augmenting variables. Reasonable bijections can be hard to find if it is unclear how the parameters in each model correspond to one another. If our bijections

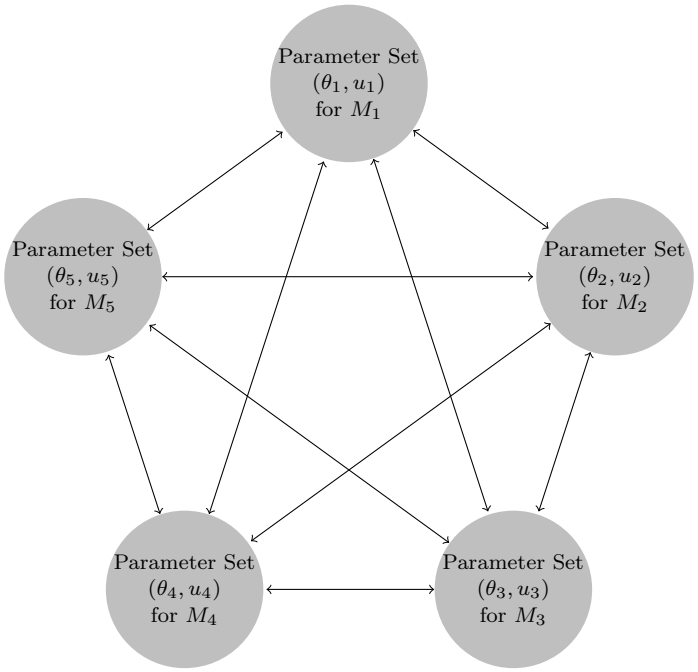

Figure 1: The ten reversible jump bijections required for a five-model set. Arrows represent bijections between parameter sets. Each parameter set contains the model-specific parameters θ_k and augmenting variables u_k.

are inefficient, we often only find out once the algorithm has run and failed to converge; at this point we must repeat the process with new bijections.

The RJMCMC framework is general and powerful, but has significant mathematical complexity and can be challenging to implement. Barker and Link (2013) suggested a restricted version of Green's RJMCMC algorithm that can be implemented via Gibbs sampling. The approach is based on the introduction of a universal parameter denoted by ψ, a vector of dimension greater than or equal to

$$\max\{\dim(\theta_k)\},\ k = 1, \ldots, K.$$

From ψ, the model-specific parameters θ_k, along with augmenting variables u_k, can be calculated using the bijection $g_k(\psi) = (\theta_k', u_k')'$ with $\psi = g^{-1}((\theta_k', u_k')')$. In practice this means that in order to find new parameters θ_h from θ_k we must first find the universal parameter ψ (Figure 2). If we have K models in our set, Barker & Link's approach requires the specification of K bijections where Green's approach requires $\binom{K}{2}$ bijections. Link and Barker (2009) referred to this method as a 'hybrid' between RJMCMC and the approach by Carlin and Chib (1995).

The joint distribution can be expressed as

$$p(y, \psi, M_k) = p(y|\psi, M_k)p(\psi|M_k)p(M_k),$$

where $p(y|\psi, M_k) = p(y|\theta_k, M_k)$ is the data model for model k, $p(\psi|M_k)$ is the prior for ψ for model k and $p(M_k)$ is the prior model probability for model k.

In general we do not have priors in the form $p(\psi|M_k)$ but $p(\theta_k|M_k)$. To find $p(\psi|M_k)$ we note that

$$p(\psi|M_k) = p(g_k(\psi)|M_k)\left|\frac{\partial g_k(\psi)}{\partial \psi}\right|$$

where $p(g_k(\psi)|M_k) = p(\theta_k, u_k|M_k)$. If we assume prior independence between θ_k and u_k this reduces to

$$p(\theta_k, u_k|M_k) = p(\theta_k|M_k)p(u_k|M_k).$$

The term $\left|\frac{\partial g_k(\psi)}{\partial \psi}\right|$ is the determinant of the Jacobian for the bijection g_k which we hereafter denote as $|J_k|$. Once we know $|J_k|$, we can find the prior $p(\psi|M_k)$ and in turn the joint distribution $p(y, \psi, M)$.

The algorithm proceeds by defining a Gibbs sampler that alternates between updating M and ψ. The full-conditional distribution for M is categorical with probabilities

$$p(M_k|\cdot) = \frac{p(y, \psi, M_k)}{\sum_j p(y, \psi, M_j)}.$$

To draw from the full-conditional for ψ, we sample θ_k from its posterior $p(\theta_k|M_k, y)$ and u_k from its prior $p(u_k|M_k)$ and determine $\psi = g_k^{-1}((\theta_k', u_k')')$. Posterior model probabilities are not estimated empirically based on the sampling frequencies for each model – rather, the results come from an eigendecomposition of the transition matrix for M.

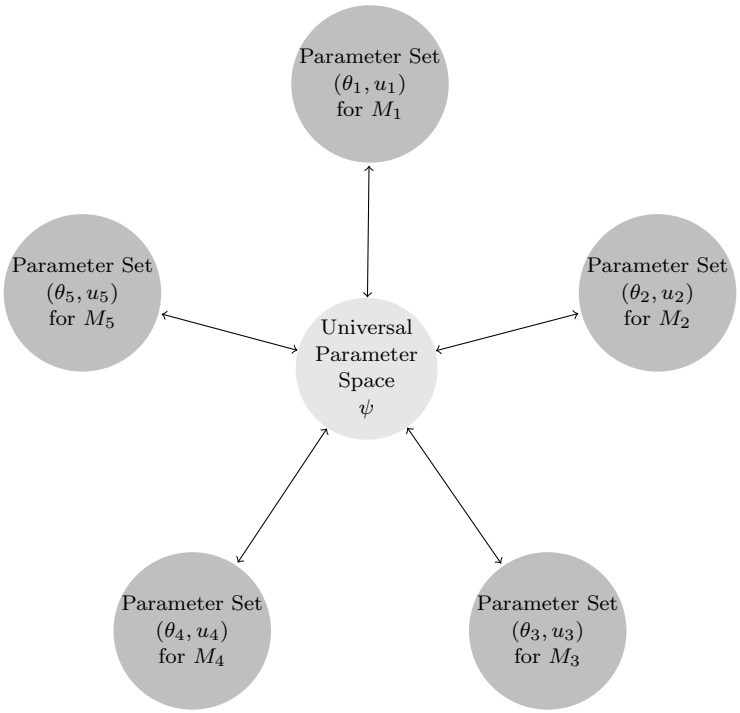

Figure 2: In Barker & Link's reversible jump approach, five bijections are required for a five-model set. Each transformation is evaluated via the universal parameter ψ.

The dimension of J_k is $\dim(\psi) \times \dim(\psi)$, for each of the K models under consideration. If we consider several models with several parameters each, finding $J_1, \ldots, J_K$ could involve hundreds of partial derivative calculations. We describe the automatic calculation of $|J_k|$ in the next section. This makes Barker & Link's formulation of RJMCMC more elegant and user-friendly.

3 Implementation in R package rjmcmc

Available from CRAN, the **rjmcmc** package utilises the work of Barker and Link (2013) to perform RJMCMC post-processing.

3.1 Automatic differentiation and madness

Automatic differentiation (AD; Griewank and Walther 2008), also called algorithmic differentiation, numerically evaluates the derivative of a function for a given input in a mechanical way. The process involves breaking a program into a series of elementary arithmetic operations ($+$, $\times$) and elementary function calls (log, exp, etc.). The chain rule is then propogated along these operations to give derivatives. The resulting derivatives are usually more numerically accurate than those from finite differencing and many other numerical methods (Carpenter et al. 2015). AD tends to be more versatile than symbolic differentiation as it works on any computer program, including those with loops and conditional statements (Carpenter et al. 2015).

Automatic differentiation has two variants – forward-mode and reverse-mode. We focus on forward-mode as this is the variant used by our software. Suppose we have a composition such that the chain rule can be written as $dy/dx = \frac{\partial y}{\partial w_1} \frac{\partial w_1}{\partial w_2} \frac{\partial w_2}{\partial x}$, where w_1, w_2 are variables representing intermediate chain rule sub-expressions. Then forward-mode AD traverses the chain rule from the inside to the outside. We compute $\partial w_2/\partial x$ first and work backwards to get to dy/dx. This amounts to fixing the independent variable x. In a multivariate situation where both $\mathbf{x}$ and $\mathbf{y}$ are vectors, we consider each independent variable x_i one at a time, differentiating the entire vector $\mathbf{y}$ with respect to x_i.

The **madness** package (Pav 2016) performs forward-mode automatic differentiation from within **R** using the S4 class `madness`. The package is not reliant on any external AD software. The primary drawback to **madness** is that it only calculates derivatives of specific **R** functions. Fortunately, the list of supported functions is extensive and is given in Pav (2016).

The function `adiff` from the **rjmcmc** package is essentially a wrapper to the primary functionality of **madness** as used in this application. The usage is

```
adiff(func, x, ...).
```

The object `x` is converted into a `madness` object, and the function `func` is applied to it. Generally, `func` will be a user-defined function of some sort. The '`...`' represents any further arguments to be passed to `func`.

The `adiff` function returns the result of computing `func(x, ...)` and, more importantly, the Jacobian matrix of the transformation `func`. This is accessed as the `gradient` attribute of the result. For a basic example, consider the function `x3`, which returns the cube of an object `x`. Suppose we pass $x_1 = 5, x_2 = 6$.

```
x3 = function(x){
 return(x^3)
}
y = rjmcmc::adiff(x3, c(5,6))
attr(y, "gradient")

##      [,1] [,2]
## [1,]   75    0
## [2,]    0  108
```

Entry (i, j) in the Jacobian is the result of differentiating `func` with respect to x_i and evaluating the derivative at x_j.

3.2 Posterior draws

The **rjmcmc** package does not itself fit the models of interest – it uses draws from posterior distributions that have already been obtained. In this way it can be considered a post-processing step, once model fitting is completed. There are many ways to obtain posterior draws by MCMC – popular software packages include **Stan** and the **WinBUGS/JAGS** packages, or we can write our own MCMC samplers. In some cases it is also possible to find the posterior analytically. The aforementioned MCMC packages return matrix-like coda output – each row of the coda is treated as a draw from the posterior distribution of the parameter vector.

3.3 The `rjmcmcpost` function

The core function of the **rjmcmc** package is `rjmcmcpost`, which automates much of the reversible jump MCMC process. An `rjmcmcpost` function call is of the form:

`rjmcmcpost(post.draw, g, ginv, likelihood, param.prior, model.prior)`.

For a model set of size K, the user must provide:

- `post.draw`: A list of K functions. The kth function randomly draws from the posterior distribution $p(\theta_k|y, M_k)$, $k = 1, \ldots, K$. Typically these functions will sample from the coda output of a model fitted using MCMC. Functions that draw from the posterior in known form are also allowed.
- `g`: A list of K functions specifying the transformations from ψ to (θ_k, u_k) for every k.

- `ginv`: A list of K functions specifying the transformations from (θ_k, u_k) to ψ for every k. These are the inverse transformations g^{-1}.
- `likelihood`: A list of K functions specifying the log-likelihood functions $\log p(y|\theta_k, M_k)$ for the data under each model.
- `param.prior`: A list of K functions specifying the log-priors $\log p(\theta_k|M_k)$ for each model-specific parameter vector θ_k.
- `model.prior`: A K-vector of prior model probabilities $p(M_1), \ldots, p(M_k)$.

The output from the `rjmcmcpost` function is an object of class `rj`. An `rj` object contains several elements which can be extracted using the `$` operator:

1. `result` – contains point estimates of:
   - The $K \times K$ transition matrix corresponding to the Markov chain for M. The (i, j)th entry is the probability of moving from M_i to M_j at a given iteration. The diagonal entries correspond to retaining a model, while the off-diagonal entries correspond to switching models. Each row sums to one. The algorithm mixes best when each entry is $\approx 1/K$.
   - The posterior model probabilities. The ith entry in this vector is the estimate of $p(M_i|y)$.
   - The Bayes factors, found using
   $$\mathrm{BF}_{ij} = \frac{p(y|M_i)}{p(y|M_j)} = \frac{p(M_i|y)}{p(M_j|y)} \frac{p(M_j)}{p(M_i)}.$$
   The Bayes factors from `rjmcmcpost` compare each model to the first model – i.e., they are BF_{i1}, $i = 1, \ldots, K$.
   - The second eigenvalue of the transition matrix λ_2, used to estimate a bound on the rate of convergence by Cheeger's Inequality (Liu 2008, page 261). Its range is $0 \leq \lambda_2 \leq 1$ where λ_2 close to zero implies fast convergence.

   Using the `print` method on an `rj` object will print `result`.
2. `densities` – matrices containing the log-likelihood, log-prior density, and log-posterior density for each model at every iteration of the algorithm. They can be used to assess problems with model specification etc. Using the `summary` method on an `rj` object returns quantiles of these densities, along with textttresult.
3. `psidraws` – a matrix containing the universal parameter vector ψ sampled at every iteration of the algorithm.

4. `progress` – contains the transition matrices and posterior model probabilities as they were calculated while the Markov chain progressed. This can be used to assess efficiency in reaching the values in `result`. Using the `plot` method on an `rj` object uses the `progress` element to illustrate how the posterior probability estimates changed as the algorithm progressed.

5. `meta` – Information about the `rjmcmcpost` call.

Note: As a memory-saving measure, both `densities` and `psidraws` are only saved and returned if `rjmcmcpost` is called with argument `save.all = TRUE`.

As this implementation is a post-processing algorithm, we can easily consider several sets of bijections g. Given that we have posterior information about each of the K models under consideration, it is quick to call `rjmcmcpost` several times to find the bijections which are most efficient. By contrast, in order to modify the bijections in standard RJMCMC, the entire algorithm must be executed again.

3.4 The `defaultpost` function

Often, defining efficient bijections between models is difficult. To aid in the package's usability, we have included a sister function to `rjmcmcpost` named `defaultpost` which does not require user-defined bijections. The `defaultpost` function uses a pseudo-prior approach similar to that of Carlin and Chib (1995) based on a normal approximation of the posterior distribution. While often inefficient, this function might be useful for preliminary analyses.

The function behaves very similarly to `rjmcmcpost` – the differences are laid out below:

`defaultpost(coda, likelihood, param.prior, model.prior)`.

- The first argument is a list of the codas themselves, rather than a list of functions that draw from the codas.
- The arguments `g` and `ginv` are removed – they are replaced by normal pseudo-priors determined within the function.

4 Examples

4.1 Fish growth – Gompertz vs. von Bertalanffy

Individual growth models represent how individual organisms increase in size over time. Two popular individual growth models are the Gompertz function (Gompertz 1825)

$$\mu_i = A\exp(-b\mathrm{e}^{-ct_i}) \qquad A > 0,\ b > 0,\ c > 0$$

and the von Bertalanffy growth equation (Von Bertalanffy 1938)

$$\mu_i = L(1 - \exp(-k(t_i + t_0)) \qquad L > 0,\ k > 0,\ t_0 > 0.$$

In particular, these curves are often used in the literature to model the length of fish over time; see, for example, Katsanevakis (2006) for a multi-model comparison across several datasets based on AIC. Here, we analyse the `Croaker2` dataset from the **R** package **FSAdata** (Ogle 2016) which records the growth of Atlantic croaker fish. We consider only the male fish. The goal is to assess model uncertainty of male croaker growth using the **rjmcmc** package.

Selected realisations of these curves can be found in Figure 3. Under our parameterisation, each model has three parameters. The Gompertz curve is parameterised by A, b and c. The value A is the mean length of a fish of infinite age, i.e. the value that the curve approaches asymptotically. The displacement along the x-axis is controlled by b, and c is the growth rate.

The von Bertalanffy curve has parameters L, t_0, and k. The mean length at infinity is denoted as L and corresponds with A in the Gompertz model. The value k is a growth rate coefficient, while t_0 is the theoretical time between size 0 and birth.

In order to define likelihoods for the purposes of RJMCMC, we treat the observations y_{ij} for fish i at time j as normally-distributed. The mean for each model is equal to the value of the respective growth curve at time j with the same variance for all fish.

$$\text{Model 1: } y_{ij} \sim \text{Normal}(A\exp(-be^{-ct_j}), \sigma^2)$$

$$\text{Model 2: } y_{ij} \sim \text{Normal}(L(1-\exp(-k(t_j+t_0)), \sigma^2)$$

In order to represent this in **R**, we define simple functions `dgomp` and `dbert` which calculate the height of the respective growth curves for supplied parameter values.

```
dgomp = function(t, A, b, c){ A*exp(-b*exp(-c*t)) }
dbert = function(t, L, t0, k){ L*(1-exp(-k*(t+t0))) }
```

Suppose that our data (the lengths of fish) are in an **R** vector `y`, and our parameter values are an **R** vector `theta` corresponding to $(A, b, c, \tau)'$ for Model 1 and $(L, t_0, k, \tau)'$ for Model 2, where the precision $\tau = 1/\sigma^2$. Then we can define the log-likelihoods for these models in **R**:

```
L1 = function(theta){ sum(dnorm(y, dgomp(t, theta[1], theta[2], theta[3]),
                                1/sqrt(theta[4]), log=TRUE)) }
L2 = function(theta){ sum(dnorm(y, dbert(t, theta[1], theta[2], theta[3]),
                                1/sqrt(theta[4]), log=TRUE)) }
```

Next, we define the bijections between the ψ space and the parameter set for each model. Recall that, under Barker and Link's algorithm, $\dim(\psi) = \dim(\theta^{(k)}, u^{(k)})$ for all k, and specifically $\dim(\psi) = \max\{\dim(\theta^{(k)})\}$. In this example, $\dim(\theta^{(1)}) = \dim(\theta^{(2)}) = 4$, so $\dim(\psi) = 4$ and we do not require any augmenting variables.

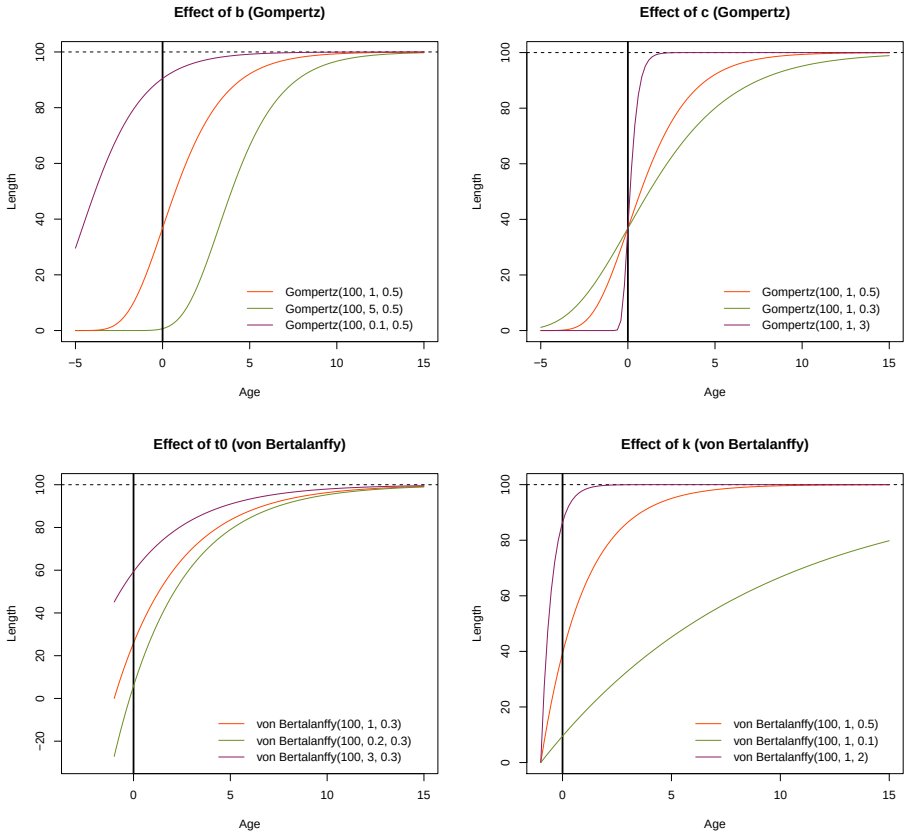

Figure 3: Some possible curves under the Gompertz and von Bertalanffy models. A and L are fixed at 100 for their respective models. In each plot, we also fix the value of a second parameter to ascertain the effect of the final parameter. For example, on the top left we fix $c = 0.5$ to examine the effect of varying b.

We consider three sets of bijections for this problem. Under the first scheme, we choose to associate $(\psi_1, \psi_2, \psi_3, \psi_4)'$ with the parameter vector $(L, t_0, k, \tau)'$ under Model 2 (von Bertalanffy). The bijection g_2 is simply the identity transformation and we can easily define an **R** function to represent each direction of the bijection, as follows.

```
g2 = function(psi){ return(theta=psi) }
ginv2 = function(theta){ return(psi=theta) }
```

The parameter A in the Gompertz model is exactly equivalent to L in the von Bertalanffy model so we also associate A with ψ_1 directly. Likewise, we directly relate the precision τ in both models. We relate the other parameters so that the resulting growth curves are as similar as possible. We do this by having the curves intersect at two points: $t = 0$ and $t = t^*$. The choice of t^* has no effect on the posterior distribution but does influence MCMC efficiency and can be thought of as a tuning parameter. In practice, t^* should be chosen where there is high data concentration. Under this bijection, we can calculate $\theta^{(1)}$ by taking:

$$\theta = \begin{bmatrix} A \\ b \\ c \\ \tau \end{bmatrix} = g_1\left(\begin{bmatrix} \psi_1 \\ \psi_2 \\ \psi_3 \\ \psi_4 \end{bmatrix}\right) = \begin{bmatrix} \psi_1 \\ -\log(1-\exp(-\psi_2\psi_3)) \\ -\log\left[\frac{\log(1-\exp(-\psi_3[\psi_2+t^*]))}{\log(1-\exp(-\psi_2\psi_3))}\right]/t^* \\ \psi_4 \end{bmatrix}$$

and solving for ψ gives the inverse $g_1^{-1}(\theta)$:

$$\psi = g_1^{-1}\left(\begin{bmatrix} A \\ b \\ c \\ \tau \end{bmatrix}\right) = g_1^{-1}\left(\begin{bmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \\ \theta_4 \end{bmatrix}\right) = \begin{bmatrix} \theta_1 \\ \log(1-\exp(-\theta_2))t^*/\log\left[\frac{\exp(-\theta_2\exp(-\theta_3 t^*))-1}{\exp(-\theta_2-1)}\right] \\ -\log\left[\frac{\exp(-\theta_2\exp(-\theta_3 t^*))-1}{\exp(-\theta_2-1)}\right]/t^* \\ \theta_4 \end{bmatrix}$$

```
g1 = function(psi){
  temp = exp(-psi[2]*psi[3])
  c(psi[1],
    -log(1-temp),
    -log((log(1-temp*exp(-psi[3]*tstar))) / (log(1-temp)))/tstar,
    psi[4])
}
ginv1 = function(theta){
  temp = -log((exp(-theta[2]*exp(-theta[3]*tstar))-1)
              / (exp(-theta[2])-1))/tstar
  c(theta[1],
    -log(1-exp(-theta[2]))/temp,
    temp,
```

```
    theta[4])
}
```

These bijections are efficient at the expense of simplicity. The other two sets of bijections that we consider are much simpler and avoid the manipulation required above. Under the second scheme, we use the `defaultpost` function to invoke the default method using normal posterior approximations. Since, in theory, almost any choice of bijection should give correct results if run for long enough, our third scheme is simply an identity map for both g_1 and g_2.

Next we define the prior distributions. We have used weakly informative prior distributions (Gelman et al. 2006) so that the overall variability of the prior predictive distribution was similar between the two models. We used the following independent half-normal prior distributions:

$$A, L \sim \text{Half-Normal}(0, 10^6), \qquad b, t_0 \sim \text{Half-Normal}(0, 20), \qquad c, k \sim \text{Half-Normal}(0, 1).$$

Finally, for both models we use a conjugate gamma prior for the precision τ:

$$\tau \sim \text{Gamma}(0.01, 0.01).$$

Since $\psi = g^{-1}(\theta_k)$, we can find $p(\psi|M_k)$ by applying the change of variables theorem to the prior for the parameters $p(\theta_k|M_k)$. So the prior for ψ is

$$\begin{aligned} p(\psi|M_1) &= p(A|M_1)p(b|M_1)p(c|M_1)p(\tau|M_1) \times |J_1| \\ &= \text{N}(A; 0, 10^6) \times \text{N}(b; 0, 20) \times \text{N}(c; 0, 1) \times \text{Gamma}(0.01, 0.01) \times |J_1|. \end{aligned}$$

The **R** function we define to represent this must be $\log p(\psi|M_1)$, since the `rjmcmcpost` function uses log-priors along with log-likelihoods. Note that, although the determinant of the Jacobian $|J_1|$ is required for this transformation, the algorithm will automatically calculate and multiply by $|J_1|$ so we need not include it.

```
p.prior = function(theta){
  sum(dnorm(theta[1:3], 0, 1/sqrt(c(1e-6, 0.05, 1)), log=T)) +
  dgamma(theta[4], 0.01, 0.01, log=T)
}
```

Ordinarily, we would define one prior function per model. Since our priors are the same for both models, we can just use the same function twice.

Finally, we need a function defined for each model which randomly draws from the posterior. Given the MCMC output from an analysis of the model, this function should select an iteration at random and return the parameter vector θ at that iteration. The **rjmcmc** package includes a function `getsampler` which may be of use here. It takes an object `modelfit` which may be coerced to an `mcmc` object – for example, a matrix with one column per variable or an `rjags` object – and defines a sampling function of the correct form. The function usage is:

```
getsampler(modelfit, sampler.name="post.draw", order="default").
```

The parameters can be sorted using the `order` argument before they are returned. This argument is unrelated to the `order` function from base **R**.

If the posterior is in known form, a function can instead be defined by the user which randomly generates values from the known distribution directly.

For this example, we fitted our models using **JAGS** (Plummer et al. 2003). We obtained the coda objects `C1` and `C2` for our respective models (see Appendix for the code used). We then used `getsampler` to define functions `draw1` and `draw2`. Note that the rows of `C2` are ordered alphabetically as (`k,L,t_0,tau`) but we require the parameter vector as (`L,t_0,k,tau`). We use the `order` argument to arrange the rows into the desired order. The parameters in `C1`, (`A,b,c,tau`), are already arranged correctly so using `order` is not required.

```
library("rjmcmc")
getsampler(C1, "draw1")
getsampler(C2, "draw2", order=c(2, 3, 1, 4))
```

We are now ready to read in the data and call `rjmcmcpost`. The algorithm is most efficient when the posterior model probabilities are each $\approx 1/K$. To facilitate this, we have specified 'informative' prior model probabilities which result in $p(M_1|y)$ and $p(M_2|y)$ each being approximately $1/2$. This choice does not affect our Bayes factor estimates. We choose $t^* = 6$ because of the high data concentration around $t = 6$. The output from the call corresponding to our first set of bijections is presented below.

```
data("Croaker2", package="FSAdata")
CroakerM = Croaker2[which(Croaker2$sex=="M"),]
y = CroakerM$tl; t = CroakerM$age
tstar = 6

growth = rjmcmcpost(post.draw = list(draw1,draw2), g = list(g1,g2),
               ginv = list(ginv1,ginv2), likelihood = list(L1,L2),
               param.prior = list(p.prior,p.prior),
               model.prior = c(0.7,0.3), chainlength = 5000,
               progress = FALSE)
growth$result

## $`Transition Matrix`
##           [,1]      [,2]
## [1,] 0.6870376 0.3129624
## [2,] 0.2986703 0.7013297
##
## $`Posterior Model Probabilities`
## [1] 0.4883164 0.5116836
##
## $`Bayes Factors`
## [1] 1.000000 2.444989
```

```
##
## $`Second Eigenvalue`
## [1] 0.3883673
```

The prior odds are equal to 0.3/0.7, so $\text{BF}_{21} = \frac{0.511}{0.489} \times \frac{0.3}{0.7} = 2.43$, despite the fitted models being barely distinguishable by eye (Figure 4). Repeating the function call with equal model priors gives posterior model probabilities of 0.287 and 0.713 for Models 1 and 2.

These results indicate that Model 2, the von Bertalanffy curve, may fit Atlantic croaker growth better than Model 1, the Gompertz function. This is perhaps unsurprising since Ogle (2016) used the male croaker data to demonstrate the suitability of the von Bertalanffy function for modelling fish growth. We also note that both fitted models seem to approximate exponential growth; there is no evidence of a sigmoid shape in the Gompertz model. This may be due to the lack of information we have on young fish, with only a single observation for $t < 2$.

The algorithm presented is fairly computationally efficient. The call above, with 10^5 iterations and two models, took 9.4 seconds on the lead author's desktop machine.

For our second scheme, we use the `defaultpost` function. The only difference in the function call is that `g` and `ginv` are not required, and the `post.draw` functions are replaced with the codas themselves.

```
growthDef = defaultpost(coda = list(C1, C2[,c(2,3,1,4)]),
                        likelihood = list(L1, L2),
                        param.prior = list(p.prior, p.prior),
                        model.prior = c(0.5,0.5),
                        chainlength = 5000, progress = FALSE)
growthDef$result

## $`Transition Matrix`
##            [,1]      [,2]
## [1,] 0.82079228 0.1792077
## [2,] 0.06797657 0.9320234
##
## $`Posterior Model Probabilities`
## [1] 0.2750036 0.7249964
##
## $`Bayes Factors`
## [1] 1.000000 2.636316
##
## $`Second Eigenvalue`
## [1] 0.7528157
```

This method appears to give a transition matrix with more weight on the diagonal, indicating poorer mixing between models. As we might expect, `defaultpost`

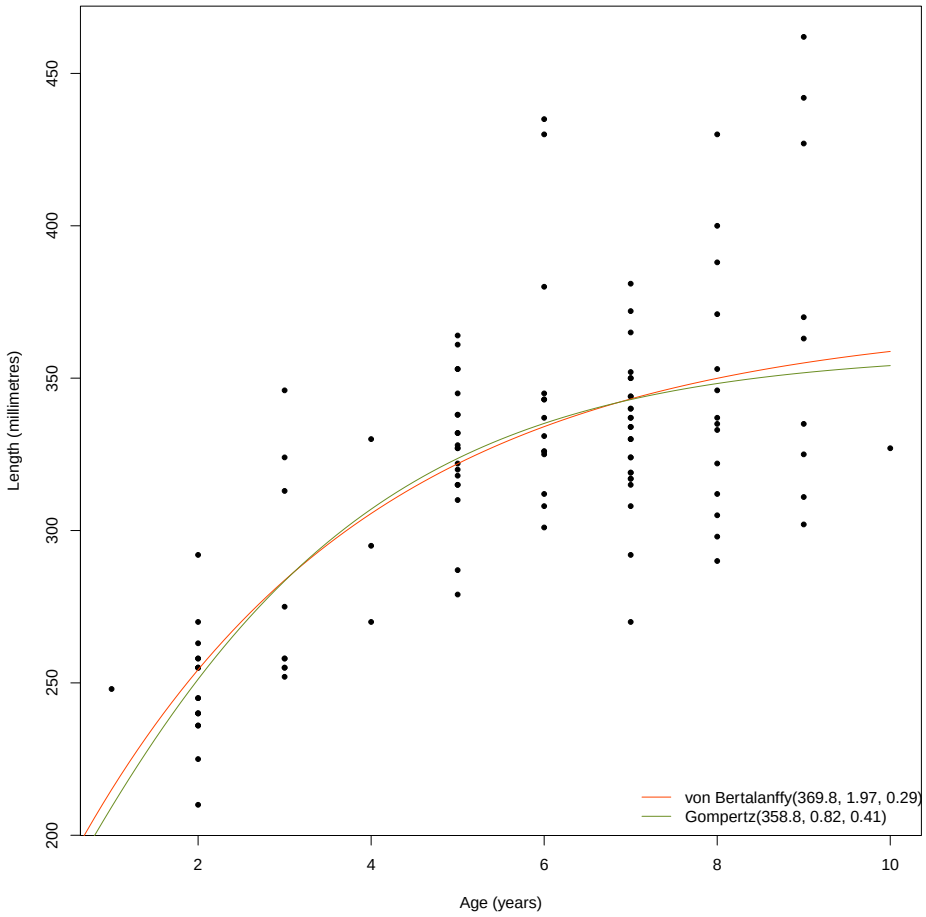

Figure 4: Fitted growth curves for male Atlantic croakers, obtained using median posterior estimates from **JAGS** output. The von Bertalanffy curve is preferred by RJMCMC with probability 0.713.

appears to require longer to converge to the correct Bayes factor. The second eigenvalue of the transition matrix is further from 0, implying a slower rate of convergence. The scheme using identity maps produced an estimated Bayes factor of 3.151 after 10^5 iterations, an even less accurate result than was given by the `defaultpost` approach. The value of λ_2 was 0.908, indicating much slower convergence.

4.2 Seed germination – logistic regression

Originally appearing in the distribution of WinBUGS (Lunn et al. 2000), this example considers a 2×2 factorial experiment of seed germination rates with two types of seeds and two root extracts. The data are modelled as exchangeable binomial random variables

$$X_i \sim \text{Bin}(N_i, p_i), \qquad \eta_i = \text{logit}(p_i),$$

where p_i are the germination rates. The linear predictor η_i is modelled in three different ways:
Model 1: $\eta_i = \gamma_{f_i}$,
Model 2: $\eta_i = \gamma_{f_i} + \epsilon_i$, $\epsilon_i \sim \text{N}(0, \sigma_\epsilon^2)$,
Model 3: $\eta_i \in \mathbb{R}$,
where the factor level of an observation i is $f(i) = 2E_i + S_i + 1$ with indicators E_i for extract 1 and S_i for seed type 'bean'. The data are presented in Table 1.

i	X_i	N_i	Extract	Seed	$f(i)$	i	X_i	N_i	Extract	Seed	$f(i)$
1	10	39	0	B	2	12	8	16	1	B	4
2	23	62	0	B	2	13	10	30	1	B	4
3	23	81	0	B	2	14	8	28	1	B	4
4	26	51	0	B	2	15	23	45	1	B	4
5	17	39	0	B	2	16	0	4	1	B	4
6	5	6	0	C	1	17	3	12	1	C	3
7	53	74	0	C	1	18	22	41	1	C	3
8	55	72	0	C	1	19	15	30	1	C	3
9	32	51	0	C	1	20	32	51	1	C	3
10	46	79	0	C	1	21	3	7	1	C	3
11	10	13	0	C	1						

Table 1: Number of germinated seeds X_i out of total seeds N_i. Seeds were either beans (B) or cucumbers (C), and were in the presence of one of two root extracts.

We wish to use RJMCMC to investigate the strength of evidence for each of these models. Respectively, our parameter sets are of dimensions 4, 25 and 21, so the universal parameter ψ is of dimension 25. Twenty-one augmenting variables are required for Model 1 and four are required for Model 3, as shown in Table 2.

ψ	$\Theta^{(1)}$	$\Theta^{(2)}$	$\Theta^{(3)}$
ψ_1	γ_1	γ_1	u_1
ψ_2	γ_2	γ_2	u_2
ψ_3	γ_3	γ_3	u_3
ψ_4	γ_4	γ_4	u_4
ψ_5	u_5	ϵ_1	η_1
ψ_6	u_6	ϵ_2	η_2
ψ_7	u_7	ϵ_3	η_3
$\vdots$	$\vdots$	$\vdots$	$\vdots$
ψ_{25}	u_{25}	ϵ_{21}	η_{21}

Table 2: The universal parameter vector ψ and the corresponding model-specific parameters.

The log-likelihoods for these models can be written as follows:

```
L1 = function(theta){
  pr = plogis(theta[f])
  sum(dbinom(X, N, pr, log=T))
}

L2 = function(theta){
  pr = plogis(theta[f] + theta[5:25])
  sum(dbinom(X, N, pr, log=T))
}

L3 = function(theta){
  pr = plogis(theta[5:25])
  sum(dbinom(X, N, pr, log=T))
}
```

Next, we specify priors on our parameters. We define a variance hyperparameter V used to control how much the parameters can vary. Then (using V) we assign the following priors:
Model 1: $\gamma_i \sim \text{N}(0, V^{-1}), \quad i = 1, \ldots, 4$
Model 2: $\gamma_i \sim \text{N}(0, (2V)^{-1}), \quad i = 1, \ldots, 4, \quad \epsilon_i \sim \text{N}(0, (2V)^{-1}), \quad i = 5, \ldots, 25$
Model 3: $\eta_i \sim \text{N}(0, V^{-1}), \quad i = 1, \ldots, 21$
Link and Barker (2009) showed that assigning $1/V$ a Gamma$(3.29, 7.80)$ prior gives the desirable property that p is approximately Uniform$(0, 1)$ distributed. Finally, we need priors on the augmenting variables. Since their posterior distributions are not informed by data, the posterior is equal to the prior – for this reason, a sensible choice is to choose the priors for $u_1, \ldots, u_4$ (Model 3) as normal approximations to the posteriors for $\gamma_1, \ldots, \gamma_4$ (Model 2). Similarly, we

choose the priors for $u_5, \ldots, u_{25}$ (Model 1) to be independent normal approximations of the posteriors for $\epsilon_1, \ldots, \epsilon_{21}$ (Model 2).

```
prior1 = function(theta){
  sum(dnorm(theta[1:4], 0, 1/sqrt(theta[26]), log=T)) +
  sum(dnorm(theta[5:25], mu[2,5:25], sig[2,5:25], log=T))
}

prior2 = function(theta){
  sum(dnorm(theta[1:25], 0, 1/sqrt(2*theta[26]), log=T))
}

prior3 = function(theta){
  sum(dnorm(theta[1:4], mu[2,1:4], sig[2,1:4], log=T)) +
  sum(dnorm(theta[5:25], 0, 1/sqrt(theta[26]), log=T))
}
```

Note that `mu` and `sig` above are 3×25 matrices of posterior means and standard deviations. Each row corresponds to one of the three models, and each column corresponds to a parameter.

Finally, we define bijections from the ψ space to each model-specific parameter set. We choose to use an identity transformation for Model 2, meaning that $g_2(\psi) = \theta$. From here, we can use the value of γ_i under Model 2 to 'predict' the value of γ_i under Model 1 using the regression equation to get

$$g_1(\psi_i) = \mu_1(\gamma_i) + \frac{\sigma_1(\gamma_i)}{\sigma_2(\gamma_i)}(\psi_i - \mu_2(\gamma_i))$$

for $i = 1, \ldots, 4$, and where $\mu_k(\cdot)$ and $\sigma_k(\cdot)$ denote the posterior means and standard deviations under Model k. We can do the same thing to relate ϵ_i under Model 2 with η_i under Model 3:

$$g_3(\psi_{i+4}) = \mu_3(\eta_i) + \frac{\sigma_3(\eta_i)}{\sigma_2(\epsilon_i)}(\psi_{i+4} - \mu_2(\epsilon_i))$$

for $i = 1, \ldots, 21$. The bijections for the augmenting variables can simply be the identity map.

```
g1 = function(psi){
  theta = c(mu[1,1:4] + sig[1,1:4]/sig[2,1:4] *(psi[1:4] - mu[2,1:4]),
            psi[5:26])  # u5,...,u25 and V follow identity map
}
ginv1 = function(theta){
  psi = c(mu[2,1:4] + sig[2,1:4]/sig[1,1:4] *(theta[1:4] - mu[1,1:4]),
          theta[5:26])
}
```

```
g2 = function(psi){ psi }
ginv2 = function(theta){ theta }

g3 = function(psi){
  theta = c(psi[1:4],
            mu[3,5:25] + sig[3,5:25]/sig[2,5:25]*(psi[5:25] - mu[2,5:25]),
            psi[26])
}
ginv3 = function(theta){
  psi = c(theta[1:4],
          mu[2,5:25] + sig[2,5:25]/sig[3,5:25]*(theta[5:25] - mu[3,5:25]),
          theta[26])
}
```

We again use **JAGS** and `getsampler` to define functions `draw1` and `draw2` which sample from coda output; the code used can be found in the Appendix. Finally, we read in the data and complete the function call. We again assign prior model probabilities that lead to roughly equal sampling frequencies.

```
seeds = rjmcmcpost(post.draw = list(draw1, draw2, draw3),
                   likelihood = list(L1, L2, L3),
                   g = list(g1, g2, g3),
                   ginv = list(ginv1, ginv2, ginv3),
                   param.prior = list(prior1, prior2, prior3),
                   model.prior = c(0.011, 0.028, 0.961),
                   chainlength = 5000, progress = FALSE)
seeds$result

## $`Transition Matrix`
##            [,1]       [,2]       [,3]
## [1,] 0.66029103 0.02624504 0.31346393
## [2,] 0.02923802 0.91362804 0.05713394
## [3,] 0.30947725 0.05354552 0.63697723
##
## $`Posterior Model Probabilities`
## [1] 0.3401671 0.3163207 0.3435122
##
## $`Bayes Factors`
## [1] 1.00000000 0.36531710 0.01155897
##
## $`Second Eigenvalue`
## [1] 0.8746866
```

The results indicate that Model 1, the simplest model, is preferred. The Bayes factors in favour of this Model are $BF_{12} = 2.71$ and $BF_{13} = 86.78$ (by convention we work with Bayes factors greater than one, so we have inverted the

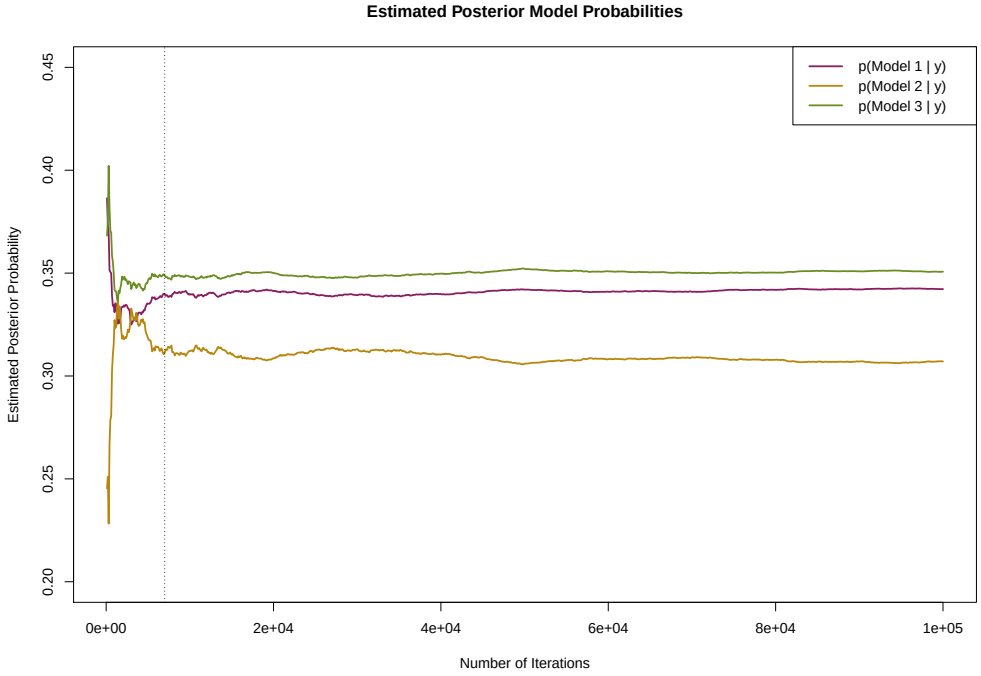

Figure 5: Posterior probabilities in favour of the three models as estimated while the RJMCMC algorithm progressed.

Bayes factors from the output above). With equal model weights, the posterior model probabilities are 0.74, 0.251, and 0.009.

Calling `probplot(seeds)` gives us a visualisation of how the estimated probabilities change as the algorithm progresses, shown in Figure 5. The probabilities are stable after roughly 7000 iterations.

5 Discussion

Bayes factors are often difficult to compute, impeding the practicality of Bayesian multimodel inference. The **rjmcmc** package presents a framework for accurately estimating Bayes factors and posterior model probabilities for a set of specified models. Further, the user is not required to find any derivatives due to the integrated automatic differentiation software. We believe this package makes reversible jump MCMC more accessible to practitioners.

Other **R** packages exist which use Bayes factors and marginal likelihoods for model selection. For instance, **BayesFactor** (Morey et al. 2015) is a package for calculating Bayes factors for simple designs including ANOVA and linear

regression models. Similarly, **BMS** (Zeugner 2015) is powerful for working with linear models, particularly in variable selection and model averaging problems, and allows efficient computation of posterior model probabilities for these models. The **MitISEM** package (Basturk et al. 2017) can calculate the marginal likelihood of a function using Importance Sampling, given that the function can be well-approximated by a mixture of t-distributions. The **MCMCpack** package (Martin et al. 2017) can use post-processing to calculate Bayes factors for eighteen common statistical models, as long as these models were also fitted using **MCMCpack**. The value of **rjmcmc** lies in its generality. Users are not restricted to common classes of models – custom probability models can also be compared. For instance, none of the above packages are able to recreate our analysis in Example 1 due to the models' unusual forms. Another clear benefit comes when we already have output prior to undertaking model comparison – **rjmcmc** does not require the models to be refit.

The use of Bayes factors is controversial. There are well documentated issues with the Bayes factor when certain vague or improper priors are used (Berger and Pericchi 1998, Han and Carlin 2001). We stress that we do not advocate the unconditional use of Bayes factors over other multimodel methods – we simply provide a new way to calculate them. In particular, care should be taken when candidate models are nested, as in variable selection contexts. We think Bayes factors are best used when candidate models are non-nested (as in our first example) and the variability in the prior predictive distribution is similar between models. In variable selection problems, we follow Gelman et al. (2013, page 84) in advocating for continuous model expansion in place of Bayes factors.

The **rjmcmc** package is not suited to variable selection problems with many candidate models. For example, consider a regression problem with k predictor variables where we wish to compare all possible models. Then, even excluding interactions, we must fit 2^k models and calculate each posterior distribution by MCMC. The burden of running every model is likely to be prohibitive. In contrast, the **BMS** package uses a birth-death sampler for large values of k, avoiding the need to explicitly sample from all possible models.

As the algorithm uses coda output, most of the intensive computation is completed prior to the function call. The model fitting and model comparison steps are effectively separate. The nature of the algorithm means that we can, for instance, adjust our choice of bijections without recalculating posteriors. For models of very high dimensionality, storing codas may become an issue. Because the algorithm requires a posterior distribution for every parameter, our coda files could occupy considerable memory. If full conditional distributions are known for any of the parameters, we may be able to mitigate this problem by computing posterior draws as required, instead of storing them in a coda.

The gradients calculated from reverse-mode automatic differentiation should theoretically be more efficient for statistical purposes than the directional derivatives obtained from forward-mode, since we usually have fewer outputs than we have parameters. If **madness** was swapped out for a reverse-mode AD engine, one might expect an increase in performance for models with many parameters.

However, as mentioned earlier, **madness** appears a more accessible option for **R** users than any current reverse-mode implementation.

References

Sylvain Arlot, Alain Celisse, et al. A survey of cross-validation procedures for model selection. *Statistics Surveys*, 4:40–79, 2010.

Richard J Barker and William A Link. Bayesian multimodel inference by rjmcmc: A gibbs sampling approach. *The American Statistician*, 67(3):150–156, 2013.

N. Basturk, L.F. Hoogerheide, A. Opschoor, and H.K. van Dijk. *Mixture of Student t Distributions using Importance Sampling and Expectation Maximization*, 2017. URL `https://CRAN.R-project.org/package=MitISEM`. R package version 1.2.

James O Berger and Luis Raúl Pericchi. *On Criticisms and Comparisons of Default Bayes Factors for Model Selection and Hypothesis Testing*. Institute of Statistics and Decision Sciences, Duke University, 1998.

Bradley P Carlin and Siddhartha Chib. Bayesian model choice via markov chain monte carlo methods. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 473–484, 1995.

Bob Carpenter, Matthew D Hoffman, Marcus Brubaker, Daniel Lee, Peter Li, and Michael Betancourt. The stan math library: Reverse-mode automatic differentiation in c++. *arXiv Preprint arXiv:1509.07164*, 2015.

Andrew Gelman, John B Carlin, Hal S Stern, David B Dunson, Aki Vehtari, and Donald B Rubin. *Bayesian Data Analysis*. CRC Press, 2013.

Andrew Gelman et al. Prior distributions for variance parameters in hierarchical models (comment on article by browne and draper). *Bayesian Analysis*, 1(3): 515–534, 2006.

Jeff Gill. *Bayesian Methods: A Social and Behavioral Sciences Approach*. CRC Press, 2014.

Benjamin Gompertz. On the nature of the function expressive of the law of human mortality, and on a new mode of determining the value of life contingencies. *Philosophical Transactions of the Royal Society of London*, 115: 513–583, 1825.

Peter J Green. Reversible jump markov chain monte carlo computation and bayesian model determination. *Biometrika*, 82(4):711–732, 1995.

Andreas Griewank and Andrea Walther. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*. Siam, 2008.

Cong Han and Bradley P Carlin. Markov chain monte carlo methods for computing bayes factors: A comparative review. *Journal of the American Statistical Association*, 96(455):1122–1132, 2001.

Jennifer A Hoeting, David Madigan, Adrian E Raftery, and Chris T Volinsky. Bayesian model averaging: A tutorial. *Statistical Science*, pages 382–401, 1999.

Harold Jeffreys. Some tests of significance, treated by the theory of probability. In *Proceedings of the Cambridge Philosophical Society*, volume 31, pages 203–222, 1935.

Robert E Kass and Adrian E Raftery. Bayes factors. *Journal of the American Statistical Association*, 90(430):773–795, 1995.

Stelios Katsanevakis. Modelling fish growth: Model selection, multi-model inference and model selection uncertainty. *Fisheries Research*, 81(2):229–235, 2006.

William A Link and Richard J Barker. *Bayesian Inference: With Ecological Applications*. Academic Press, 2009.

Jun S Liu. *Monte Carlo Strategies in Scientific Computing*. Springer Science & Business Media, 2008.

David J Lunn, Andrew Thomas, Nicky Best, and David Spiegelhalter. Winbugs-a bayesian modelling framework: concepts, structure, and extensibility. *Statistics and computing*, 10(4):325–337, 2000.

Andrew D. Martin, Kevin M. Quinn, Jong Hee Park, Ghislain Vieille-dent, Michael Malecki, and Matthew Blackwell. *Markov Chain Monte Carlo (MCMC) Package*, 2017. URL `https://CRAN.R-project.org/package=MCMCpack`. R package version 1.4-0.

Richard D. Morey, Jeffrey N. Rouder, and Tahira Jamil. *Computation of Bayes Factors for Common Designs*, 2015. URL `https://CRAN.R-project.org/package=BayesFactor`. R package version 0.9.12-2.

Derek H. Ogle. *FSAdata: Fisheries Stock Analysis, Datasets*, 2016. R package version 0.3.5.

Steven E. Pav. *madness: Automatic Differentiation of Multivariate Operations*, 2016. URL `https://github.com/shabbychef/madness`. R package version 0.2.0.

Martyn Plummer et al. Jags: A program for analysis of bayesian graphical models using gibbs sampling. In *Proceedings of the 3rd international workshop on distributed statistical computing*, volume 124, page 125. Vienna, 2003.

Adrian E Raftery. A note on bayes factors for log-linear contingency table models with vague prior information. *Journal of the Royal Statistical Society, Series B*, 48:249–250, 1986.

Gideon Schwarz et al. Estimating the dimension of a model. *The Annals of Statistics*, 6(2):461–464, 1978.

Ludwig Von Bertalanffy. A quantitative theory of organic growth (inquiries on growth laws. ii). *Human Biology*, 10(2):181–213, 1938.

Sumio Watanabe. Asymptotic equivalence of bayes cross validation and widely applicable information criterion in singular learning theory. *Journal of Machine Learning Research*, 11(Dec):3571–3594, 2010.

Stefan Zeugner. *Bayesian Model Averaging Library*, 2015. URL `https://CRAN.R-project.org/package=BMS`. R package version 0.3.4.

Appendix

Defining coda-sampling functions for Example 1

We obtain coda files using the program **JAGS** (Plummer et al. 2003), specifically the package **R2jags** which interfaces with **R**. A coda file contains the posterior distribution for the parameters, which we randomly sample from. First, we must define our models. Using **R2jags**, each model must be defined in an external text file. Here, we use the `cat` function to write text files `fishGomp.txt` and `fishBert.txt`.

```
cat("model{
    for(ti in 1:10){
      mu[ti] <- A*exp(-b*exp(-c*ti))
    }
    for(i in 1:n){
      y[i] ~ dnorm(mu[t[i]], tau)
    }
    A ~ dnorm(0, 0.00001)T(0,)  #
    b ~ dnorm(0, 0.05)T(0,)     # precision = 1/variance
    c ~ dnorm(0, 1)T(0,)        #
    tau ~ dgamma(0.01, 0.01)
  }", file="fishGomp.txt")

cat("model{
    for(ti in 1:10){
      mu[ti] <- L*(1-exp(-k*(ti+t0)))
    }
    for(i in 1:n){
      y[i] ~ dnorm(mu[t[i]], tau)
    }
    L ~ dnorm(0, 0.000001)T(0,)
    t0 ~ dnorm(0, 0.05)T(0,)
    k ~ dnorm(0, 1)T(0,)
    tau ~ dgamma(0.01, 0.01)
  }", file="fishBert.txt")
```

Next, we perform the MCMC sampling using **R2jags**.

```
## Gompertz model
inits = function(){list(A = abs(rnorm(1, 350, 200)), b = abs(rnorm(1, 2, 3)),
                        c = abs(rnorm(1, 1, 2)), tau = rgamma(1, 0.1, 0.1))}
params = c("A", "b", "c", "tau")
jagsfit1 = jags(data = c('y', 't', 'n'), inits, params, n.iter=1e5,
                n.thin=20, model.file = "fishGomp.txt")
C1 = as.matrix(as.mcmc(jagsfit1))[,-4]
```

```
## von Bertalanffy model
inits = function(){list(L = abs(rnorm(1, 350, 200)), t0 = abs(rnorm(1, 2, 3)),
                        k = abs(rnorm(1, 1, 2)), tau = rgamma(1, 0.1, 0.1))}
params = c("L", "t0", "k", "tau")
jagsfit2 = jags(data = c('y', 't', 'n'), inits, params, n.iter=1e5,
                n.thin=20, model.file = "fishBert.txt")
C2 = as.matrix(as.mcmc(jagsfit1))[,-1]
```

Finally, as shown in the manuscript, we define functions using `getsampler` which randomly select a timestep and return the values of both parameters at that timestep.

Defining coda-sampling functions for Example 2

```
cat("model{
    for(i in 1:21) {
      X[i] ~ dbin(p[i],N[i])
      logit(p[i]) <- beta[f[i]]
    }
    for(j in 1:4){
      beta[j] ~ dnorm(0, V)  # precision, not variance
    }
    V ~ dgamma(3.29, 7.80)
  }", file="model1.txt")
```

The JAGS files for models 2 and 3 are very similar.

We then run the sampler to estimate the posteriors and define the coda functions.

```
## Fit Model 1
jags1 = jags(data = c('N', 'X', 'f'), param=c('beta', 'V'), n.iter=n.i,
             n.burnin=n.burn, model.file = "model1.txt")
fit1 = as.mcmc(jags1); coda1 = as.matrix(fit1)[,-5]

## Fit Model 2
jags2 = jags(data = c('N', 'X', 'f'), param=c('beta', 'eps', 'V'), n.iter=n.i,
             n.burnin=n.burn, model.file = "model2.txt")
fit2 = as.mcmc(jags2); coda2 = as.matrix(fit2)[,-5]

## Fit Model 3
jags3 = jags(data = c('N', 'X'), param=c('eta', 'V'), n.iter=n.i,
             n.burnin=n.burn, model.file = "model3.txt")
fit3 = as.mcmc(jags3); coda3 = as.matrix(fit3)[,-1]
```

```
## Re-order codas to match order in manuscript
coda2 = coda2[,c(1:5, 16, 19:25, 6:15, 17:18, 26)]
coda3 = coda3[,c(1, 12, 15:21, 2:11, 13:14, 22)]

## Calculate posterior means and standard deviations
mu = sig = matrix(NA, 3, 25)
for(j in 1:25){
  mu[2,j] = mean(coda2[,j])
  sig[2,j] = sd(coda2[,j])
  if(j<=4){
    mu[1,j] = mean(coda1[,j])
    sig[1,j] = sd(coda1[,j])
  } else {
    mu[3,j] = mean(coda3[,j-4])
    sig[3,j] = sd(coda3[,j-4])
  }
}

## attach posteriors for augmenting variables to codas
lcoda = dim(coda1)[1]
u1_4 = matrix(rnorm(lcoda*4, mu[2,1:4], sig[2,1:4]), lcoda, 4, byrow=T)
u21_25 = matrix(rnorm(lcoda*21, mu[2,5:25], sig[2,5:25]), lcoda, 21, byrow=T)
coda1 = cbind(coda1[,1:4], u21_25, coda1[,5])
coda3 = cbind(u1_4, coda3)

## Define functions to randomly sample from posterior
getsampler(coda1, "draw1")
getsampler(coda2, "draw2")
getsampler(coda3, "draw3")
```