R Package rjmcmc: The Calculation of Posterior Model Probabilities from MCMC Output

Nicholas Gelling
University of Otago

Matthew R. Schofield
University of Otago

Richard J. Barker
University of Otago

Abstract

Reversible jump Markov chain Monte Carlo is a Bayesian multimodel inference method that involves 'jumping' between several candidate models. The method is powerful, but can be challenging to implement. Presented is an R package **rjmcmc** which automates much of the reversible jump process, in particular the post-processing algorithms of Barker and Link (2013). Previously-estimated posterior distributions (in the form of coda files) are used to estimate posterior model probabilities and Bayes factors. Automatic differentiation is used for the partial derivative calculations required in finding Jacobian determinants.

Keywords: Reversible jump, Bayesian multimodel inference, R, post-processing, Bayes factors, automatic differentiation.

1. Introduction

Discriminating between models is a difficult problem. There are several options for models fitted using Bayesian inference, including Bayes factors and posterior model probabilities (Kass and Raftery 1995), information criteria such as DIC and WAIC (Spiegelhalter, Best, Carlin, and Van Der Linde 2002, Watanabe 2010) and cross validation (Arlot, Celisse *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 trans-dimensional 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 posthoc calculates Bayes factors and posterior model probabilities using MCMC output, simplifying a frequently daunting problem.

The Bayes factor was developed by Jeffreys (1935). It is considered by many to be the default method of Bayesian model comparison and features in nearly every textbook on Bayesian inference (e.g. Gelman, Carlin, and Stern 2014, 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 M is a categorical variable representing model choice, $p(y|\theta_k, M=k)$ is the likelihood function under model k, and $p(\theta_k|M=k)$ is the prior distribution under model k.

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, Madigan, Raftery, and Volinsky 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 integral. Approximation is frequently used; for instance, the Bayesian Information Criterion (BIC) is derived by Schwarz *et al.* (1978) as an approximation to the Bayes factor.

Markov chain Monte Carlo (MCMC) approaches are also available for calculating the posterior model probabilities. Carlin and Chib (1995) propose an MCMC sampler that uses 'pseudo-priors' to facilitate jumping between models while RJMCMC (Green 1995) augments 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) notes that reversible jump methodology continues to be an active research area. The R package demonstrated is the first reversible jump package to be released on CRAN, and offers an accessible framework for the calculation of Bayes factors and posterior model probabilities.

In Section 2, RJMCMC is discussed further and a Gibbs sampling approach to RJMCMC is described. In Section 3, we introduce the R package **rjmcmc** which implements the Gibbs algorithm with examples. We conclude with a discussion in Section 4.

2. Transdimensional algorithms

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

$$\frac{p(M = i|y)}{p(M = j|y)} = B_{ij} \times \frac{p(M = i)}{p(M = j)}.$$

Hereafter, we use $p(\cdot|M_k)$ as shorthand notation for $p(\cdot|M = k)$ and $p(M_k|\cdot)$ as shorthand notation for $p(M = k|\cdot)$.

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 $\binom{N}{2}$ bijections are required. To match dimensions between models, augmenting variables u_k are required so that $\dim(\theta_k, u_k) = \dim(\theta_j, u_j)$ for $j, k \in \{1, \ldots, N\}$. 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 $M^{(i-1)} = j$. Proposed parameter values for model M^* 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

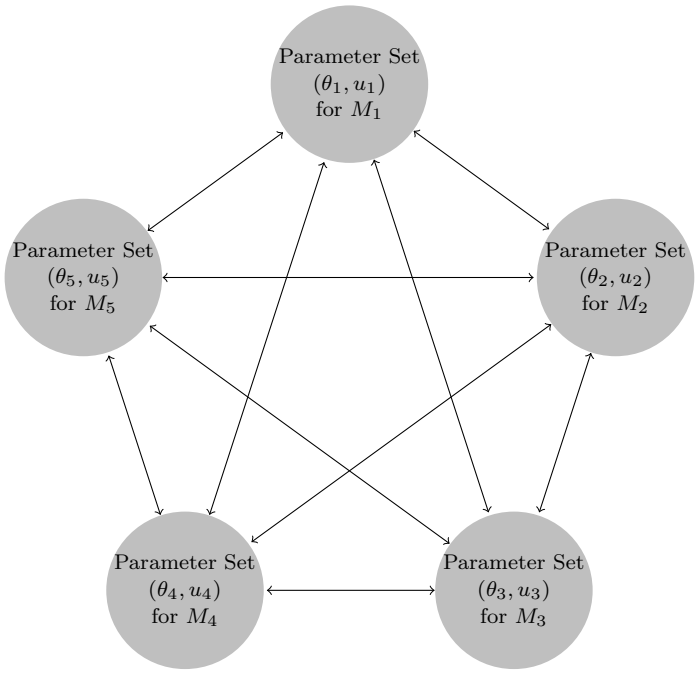

Figure 1: The ten reversible jump bijections required for a five-model set. Arrows represent bijections between parameter sets. Each parameter set contains the model-specific parameters θ_k and augmenting variables u_k.

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. We can only determine if our bijections are inefficient 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) suggest 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

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

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 the parameters θ_j from θ_k we must first find the universal parameter ψ (Figure 2). If we have N models in our set, Barker & Link's approach requires the specification of N bijections where Green's approach requires $\binom{N}{2}$ bijections. Link and Barker (2009) refer 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.

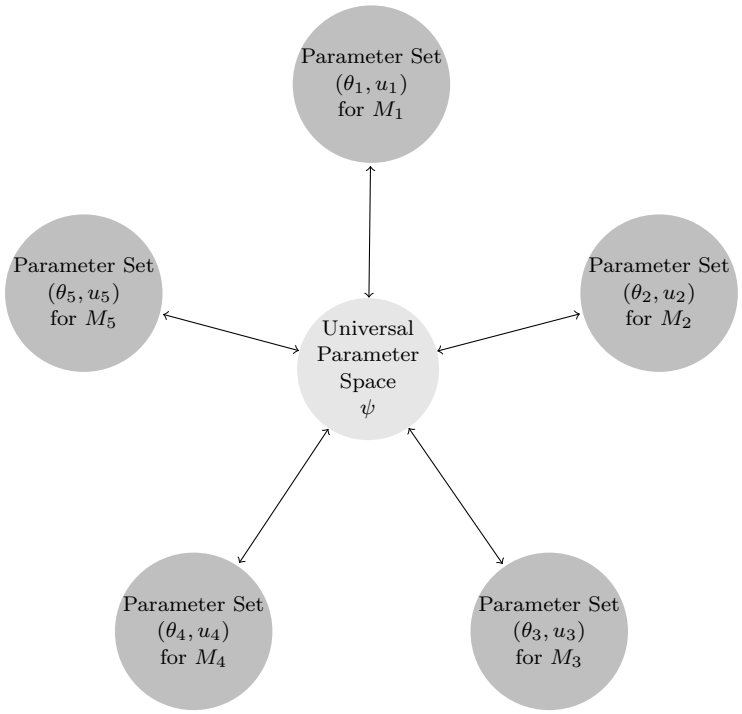

Figure 2: In Barker & Link's reversible jump approach, five bijections are required for a five-model set. Each transformation is evaluated via the universal parameter ψ.

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 $\psi = g_k^{-1}((\theta_k', u_k')')$.

Note that the dimension of J_k is $\dim(\psi) \times \dim(\psi)$, for each of the N models under consideration. If we consider several models with several parameters each, finding $J_1, \ldots, J_N$ 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 the Comprehensive R Archive Network (CRAN), the **rjmcmc** package utilises the work of Barker and Link (2013) and the **madness** package 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, Hoffman, Brubaker, Lee, Li, and Betancourt 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 $\frac{dy}{dx} = \frac{\partial y}{\partial w_1}\frac{\partial w_1}{\partial w_2}\frac{\partial w_2}{\partial x}$, 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 $\frac{\partial w_2}{\partial x}$ first and work backwards to get to $\frac{dy}{dx}$. 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.

Recently published, 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. See the package documentation for further detail about this function.

3.2. 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, chainlength)`.

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

- `post.draw`: N functions that randomly draw from the posterior distribution $p(\theta_k|y, M_k)$ for every k. Generally these functions sample from the coda output of a model fitted using MCMC. Functions that draw from the posterior in known form are also allowed.
- `g`: N functions specifying the transformations from ψ to (θ_k, u_k) for every k.
- `ginv`: N functions specifying the transformations from (θ_k, u_k) to ψ for every k. These are the inverse transformations g^{-1}.
- `likelihood`: N functions specifying the log-likelihood functions $\log p(y|\theta_k, M_k)$ for the data under each model.
- `param.prior`: N functions specifying the log-priors $\log p(\theta_k|M_k)$ for each model-specific parameter vector θ_k.
- `model.prior`: A vector of the prior model probabilities $p(M_k)$.

There are three outputs from the `rjmcmcpost` function:

1. The transition matrix `$TM`, which describes how the Markov chain for M moves over time. The (i, j)th entry in this matrix is the probability of moving from model M_i to model M_j at a given time step. The diagonal entries correspond to retaining a model, while the off-diagonal entries correspond to switching models.
2. The posterior model probabilities `$prb`. The ith entry in this vector is $p(M_i|y)$.

3. The Bayes factors `$BF`, 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}, $i = 1, \ldots, N$. The first Bayes factor printed will always equal 1.

Crucially, this implementation is a post-processing algorithm. We use the coda to sample from the model-specific posteriors $p(\theta_k|y, M_k)$. Once we have fitted each of the N models under consideration we are able to quickly post-process for several sets of bijections g to optimize efficiency. By contrast, standard RJMCMC requires the entire algorithm to be repeated in order to modify the bijections.

3.3. Example 1: Poisson vs. negative binomial

This example, originally from Green and Hastie (2009), uses the goals scored over three seasons of English Premier League football as count data. There were 380 games in each season, so N=1140. Each observation is the total number of goals scored in a single game.

We are interested in determining whether the counts are overdispersed relative to a Poisson distribution. To this end we consider two models. Under Model 1, the number of goals y_i in game i ($i = 1, \ldots, N$) is assumed to follow a Poisson distribution with constant mean $\lambda > 0$.

$$M_1 : y_i \sim \mathrm{Pois}(\lambda), \qquad L(y|\lambda) = \prod_{i=1}^{N} \frac{\lambda^{y_i}}{y_i!} \exp(-\lambda).$$

Under Model 2, the number of goals is instead assumed to follow a negative binomial distribution, with parameters $\lambda > 0$ and $\kappa > 0$.

$$M_2 : y_i \sim \mathrm{NegBin}(\lambda, \kappa), \qquad L(y|\lambda, \kappa) = \prod_{i=1}^{N} \frac{\lambda^{y_i}}{y_i!} \frac{\Gamma(\frac{1}{\kappa} + y_i)}{\Gamma(\frac{1}{\kappa})(\frac{1}{\kappa+\lambda})^{y_i}} (1 + \kappa\lambda)^{-1/\kappa}$$

We follow Green and Hastie (2009) in using a Gamma$(25, 10)$ prior for λ (indicating a mean of 2.5 goals per match) and a Gamma$(1, 10)$ prior for κ. The mean negative binomial prior has approximately 25% more variation than the mean Poisson prior.

From Figure 3, we observe that the data are right-skewed and truncated at zero. It appears feasible that the data might fit either a Poisson or a negative binomial distribution. The variance of the data ($\sigma^2 = 2.62$) is slightly higher than its mean ($\mu = 2.52$), which may indicate minor overdispersion relative to the Poisson distribution.

First 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, $\max\{\dim(\theta^{(k)})\} = \dim(\theta^{(2)}) = 2$ so $\dim(\psi) = 2$, and we will require an augmenting variable for M_1.

We choose to associate the first element of the universal parameter ψ_1 with λ in both models and ψ_2 with κ in model M_2. Under model M_1 there is no κ parameter or equivalent, so we follow Green and Hastie (2009) in using an independent augmenting variable that follows a

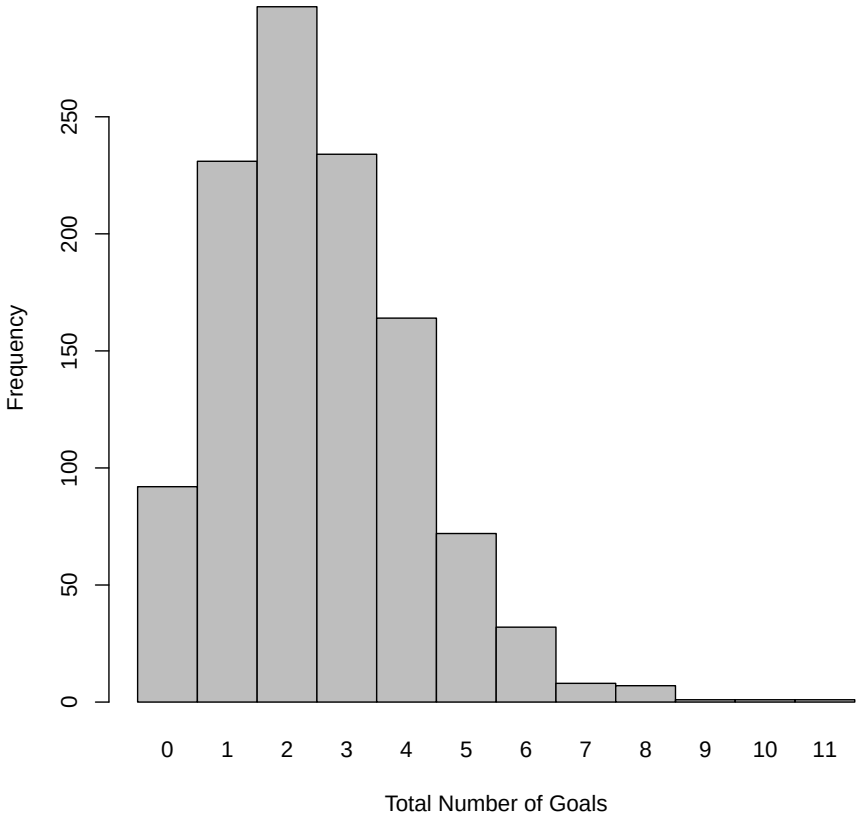

Figure 3: The empirical distribution of the goals scored in each match over three seasons of English Premier League football.

multiplicatively-centred log-normal distribution. We give u a $N(0, \sigma)$ prior, then take $\psi_2 = \mu \times \exp(u)$ for some reasonable σ and small μ. If we select hyperparameters in a way that generates feasible values for κ under M_2, the efficiency of our algorithm is increased. Since the value of u does not effect inference, neither do our choices of μ and σ. We calculate ψ by taking:

$$\psi = g_1^{-1}\left(\begin{bmatrix} \lambda \\ u \end{bmatrix}\right) = g_1^{-1}\left(\begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix}\right) = \begin{bmatrix} \theta_1 \\ \mu \exp(\theta_2) \end{bmatrix}$$

and solving for θ gives the inverse $g_1(\psi)$:

$$\theta = \begin{bmatrix} \lambda \\ u \end{bmatrix} = g_1\left(\begin{bmatrix} \psi_1 \\ \psi_2 \end{bmatrix}\right) = \begin{bmatrix} \psi_1 \\ \log\left(\frac{\psi_2}{\mu}\right) \end{bmatrix}$$

We assume that our model-specific parameter values are in an R vector `theta` of length two:

$$M_1 : \theta = (\lambda, u); \qquad M_2 : \theta = (\lambda, \kappa).$$

Then we can define an R function to represent each direction of the bijection, as follows.

```
g1 = function(psi){ c(psi[1], log(psi[2]/mu)) }
ginv1 = function(theta){ c(theta[1], mu*exp(theta[2])) }
```

This setup allows the bijection under model M_2 to be the identity such that $(\lambda, \kappa)' = g_2(\psi) = \psi$ and $\psi = g_2^{-1}(\theta) = \theta$. This is straightforward to represent in R:

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

Next we give the log-likelihoods for these models in R. Assuming that our data are in a vector `y` of length N, we define:

```
L1 = function(theta){ sum(dpois(y, theta[1], log=T)) }
L2 = function(theta){ sum(dnbinom(y, 1/theta[2], mu=theta[1], log=T)) }
```

Now we calculate the prior distributions for ψ. 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 $p(\theta_k|M_k)$. The determinant of the Jacobian $|J_k|$ is required for this transformation. Under Model 1, ψ_1 is associated with λ and ψ_2 is associated with u. So the prior for ψ is

$$p(\psi|M_1) = p(\theta_1|M_1) \times |J_1| = p(\lambda|M_1) \times p(u|M_1) \times |J_1| = \text{Gamma}(25, 10) \times \text{Normal}(0, \sigma) \times |J_1|.$$

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 we are not required to multiply by $|J_1|$ manually, as the algorithm will automatically calculate $|J_1|$ and complete this step for us.

```
p.prior1 = function(theta){dgamma(theta[1], lamprior[1], lamprior[2], log=T)
                           + dnorm(theta[2], 0, sigma, log=T)}
```

Under Model 2, ψ_1 is associated with λ and ψ_2 is associated with κ, so

$$p(\psi|M_2) = p(\theta_2|M_2) \times |J_2| = p(\lambda|M_2) \times p(\kappa|M_2) \times |J_2| = \text{Gamma}(25, 10) \times \text{Gamma}(1, 10) \times |J_2|.$$

Again, we define the R function as the logarithm of this.

```
p.prior2 = function(theta){dgamma(theta[1], lamprior[1], lamprior[2], log=T)+
                           dgamma(theta[2], kapprior[1], kapprior[2], log=T)}
```

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 a timestep at random and return the parameter vector θ at that timestep. The **rjmcmc** package includes a function `getsampler` which may be of use here. It takes a matrix-like object `modelfit` with one column per variable and defines a sampling function of the correct form. The full function usage is:

```
getsampler(modelfit, sampler.name="sampler", order="default", envir=.GlobalEnv).
```

The parameters can be sorted using the `order` argument before they are returned. By default, they are in alphabetical order. The coda-sampling function is defined in the global environment by default, but this can be altered using the `envir` argument.

If the posterior is in known form, no MCMC computation is required and `getsampler` is of no use. Instead, a function should be defined by the user which randomly generates values from the known distribution directly.

For this example, we fit our models using JAGS (Plummer *et al.* 2003) and defined functions `draw1` and `draw2` (see Appendix for the code used).

We are now ready to call `rjmcmcpost`. In this case, we give the models equal prior probability and run for 10^4 iterations. The output from the call is presented below.

```
n = length(y)
mu=0.015; sigma=1.5
lamprior = c(25,10); kapprior = c(1,10)  # hyperparameters for lambda & kappa

goals_post = rjmcmcpost(post.draw = list(draw1, draw2), g = list(g1, g2),
               ginv = list(ginv1, ginv2), likelihood = list(L1, L2),
               param.prior = list(p.prior1, p.prior2),
               model.prior = c(0.5, 0.5), chainlength = 10000)
```

```
goals_post

## $TM
##           [,1]      [,2]
## [1,] 0.7407535 0.2592465
## [2,] 0.6349091 0.3650909
##
## $prb
## [1] 0.7100655 0.2899345
##
## $BF
## [1] 1.0000000 0.4083207
```

The prior odds are 1, since the two models had equal prior probabilities, so $\text{BF}_{21} = \frac{0.29}{0.71} = 0.408$. By convention we work with Bayes factors greater than one, so we might prefer to consider $\text{BF}_{12} = \frac{1}{0.408} = 2.449$ instead. This value being greater than one indicates that Model 1 has performed better than Model 2 for this data, though by the interpretation of Kass and Raftery (1995) it is not large enough to provide substantial evidence.

The estimated posterior distributions for each model are overlaid in Figure 4. The small posterior estimate of κ under Model 2 (median= 0.02) indicates little overdispersion (as $\kappa \to 0$, the distribution reduces to a Poisson).

The posterior model probabilities found by Green and Hastie (2009) using traditional RJMCMC were 0.708 and 0.292, agreeing with those found here. The algorithm presented is fairly computationally efficient. The call above, with ten thousand iterations and two models, took 26.1 seconds on the lead author's desktop machine.

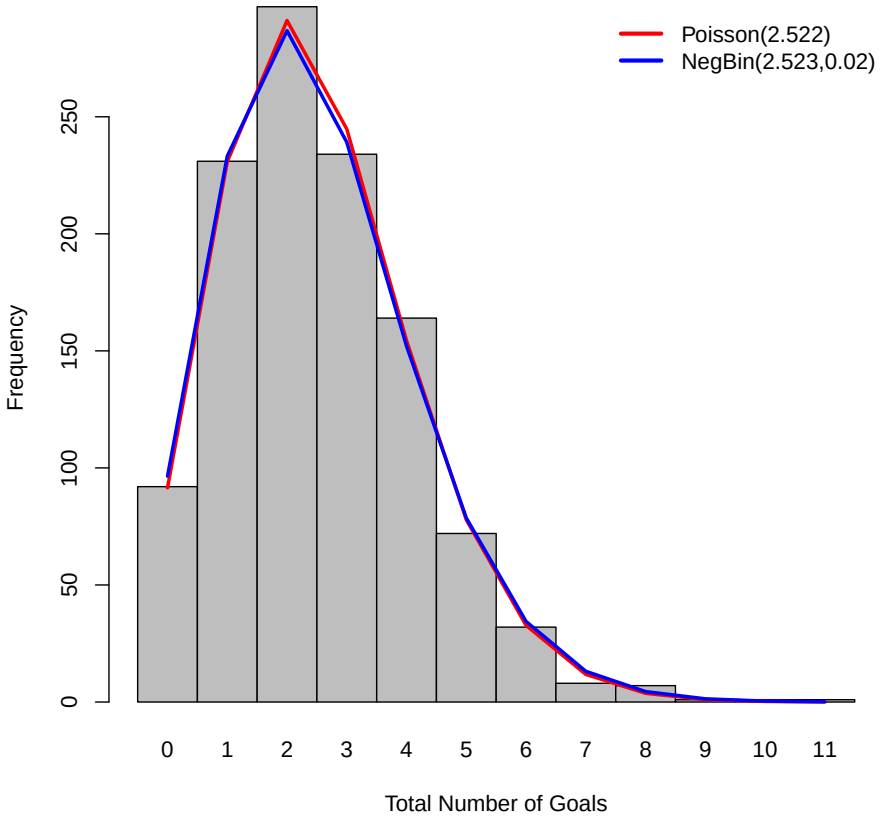

Figure 4: The fitted models for the English Premier League dataset, obtained using median posterior estimates from JAGS output. The Poisson distribution is preferred by RJMCMC with probability 0.71.

3.4. Example 2: 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.

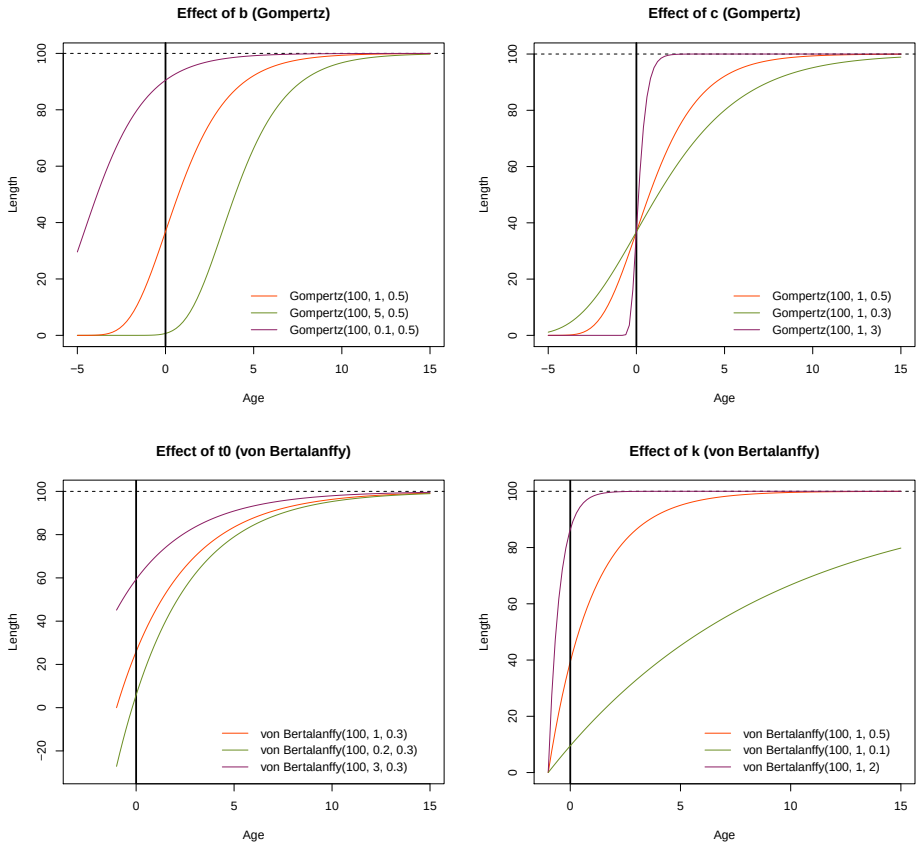

Figure 5: Some possible curves under the Gompertz and von Bertalanffy models. A and L are fixed at 100 for their respective models. In each plot, we also fix the value of a second parameter to ascertain the effect of the final parameter. For example, on the top left we fix $c = 0.5$ to examine the effect of varying b.

Selected realisations of these curves can be found in Figure 5. Under our parameterisations, 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. Also representing the mean length at infinity, L (sometimes L_∞ in other texts) 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 can 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 standard deviation 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))) }
```

Now we can easily define log-likelihoods. In each case, the parameter values are read in as a vector `theta` of length four, which includes the precision $\tau = \frac{1}{\sigma^2}$. Note that `theta` corresponds to $(A, b, c, \tau)'$ for Model 1 but $(L, t_0, k, \tau)'$ for Model 2.

```
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))}
```

Each model-specific parameter space has four dimensions (including τ). The universal parameter ψ will therefore also be of length 4, with no augmenting variables required. Suppose that, under Model 2 (von Bertalanffy) we associate $(\psi_1, \psi_2, \psi_3, \psi_4)'$ with the parameter vector $(L, t_0, k, \tau)'$. Then the bijection g_2 is simply the identity transformation.

```
g2 = function(psi){ psi }
ginv2 = function(theta){ 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. The bijection is

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

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

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

```
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])
}
```

Next we define the priors for all seven parameters. 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:

$$A, L \sim \text{Half-Normal}(0, 1000), \qquad b, t_0 \sim \text{Half-Normal}(0, \sqrt{20}), \qquad c, k \sim \text{Half-Normal}(0, 1).$$

Finally, we use a conjugate gamma prior for the precision $\tau = \frac{1}{\sigma^2}$:

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

Ordinarily, we would define one prior function per model. Since our priors are the same for both models, we can instead use the same function twice. Recall that the `rjmcmcpost` function accepts the logarithm of the prior for ψ, obtained by summing the log-priors on individual parameters.

```
p.prior1 = 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)
}
```

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 give the models equal prior probability once more. We choose $t^* = 6$ because of the high data concentration around $t = 6$.

```
library("FSAdata")
data("Croaker2")
CroakerM = Croaker2[which(Croaker2$sex=="M"),]
y = CroakerM$tl; t = CroakerM$age
n = length(y)
tstar = 6       # chosen for algorithmic efficiency

growth_post=rjmcmcpost(post.draw = list(draw1,draw2), g = list(g1,g2),
```

```
                ginv = list(ginv1,ginv2), likelihood = list(L1,L2),
                param.prior = list(p.prior1,p.prior1),
                model.prior = c(0.5,0.5), chainlength = 1e6)
```

```
## $TM
##           [,1]      [,2]
## [1,] 0.5431605 0.4568395
## [2,] 0.1884140 0.8115860
##
## $prb
## [1] 0.292 0.708
##
## $BF
## [1] 1.000000 2.424657
```

The results indicate that Model 2, the von Bertalanffy curve, may fit Atlantic croaker growth better than Model 1, the Gompertz function. The Bayes factor in favour of the von Bertalanffy curve is $\text{BF}_{21} = 2.425$, despite the fitted models being barely distinguishable by eye (Figure 6). It is perhaps unsurprising that the von Bertalanffy growth curve is preferred 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 $t < 2$.

4. Discussion

Bayes factors are often difficult to compute, impeding the practicality of Bayesian multimodel inference. The **rjmcmc** package presents a relatively simple 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.

The use of Bayes factors is controversial. There are well documentated problems with the Bayes factor when certain vague or improper priors are used (Berger and Pericchi 1998, Han and Carlin 2001). This is particularly relevant for cases where candidate models are nested (as in our first example) and care is needed. We think Bayes factors are best used when candidate models are non-nested (as in our second example) and the variability in the prior predictive distribution is similar between our 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 post-processing 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

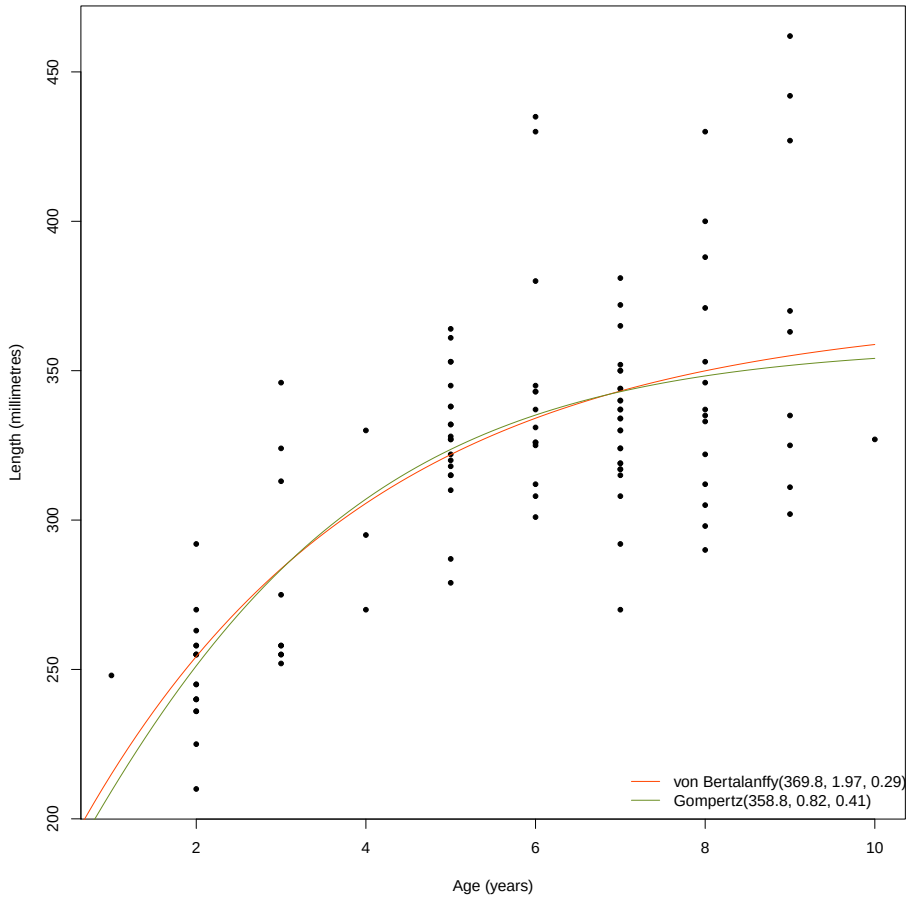

Figure 6: Fitted growth curves for male Atlantic croakers, obtained using median posterior estimates from JAGS output. The von Bertalanffy curve is preferred by RJMCMC with probability 0.708.

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 **rjmcmc** package is also not suited to variable selection contexts. 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. The burden of running each model is likely to be prohibitive.

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

Arlot S, Celisse A, *et al.* (2010). "A Survey of Cross-Validation Procedures for Model Selection." *Statistics Surveys*, **4**, 40–79.

Barker RJ, Link WA (2013). "Bayesian Multimodel Inference by RJMCMC: A Gibbs Sampling Approach." *The American Statistician*, **67**(3), 150–156.

Berger JO, Pericchi LR (1998). *On Criticisms and Comparisons of Default Bayes Factors for Model Selection and Hypothesis Testing*. Institute of Statistics and Decision Sciences, Duke University.

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

Carpenter B, Hoffman MD, Brubaker M, Lee D, Li P, Betancourt M (2015). "The Stan Math Library: Reverse-Mode Automatic Differentiation in C++." *arXiv Preprint arXiv:1509.07164*.

Gelman A, Carlin JB, Stern HS (2014). *Bayesian Data Analysis*, volume 2.

Gelman A, *et al.* (2006). "Prior Distributions for Variance Parameters in Hierarchical Models (comment on article by Browne and Draper)." *Bayesian Analysis*, **1**(3), 515–534.

Gill J (2014). *Bayesian Methods: A Social and Behavioral Sciences Approach*, volume 20. CRC Press.

Gompertz B (1825). "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.

Green PJ (1995). "Reversible Jump Markov Chain Monte Carlo Computation and Bayesian Model Determination." *Biometrika*, **82**(4), 711–732.

Green PJ, Hastie DI (2009). "Reversible Jump MCMC." *Genetics*, **155**(3), 1391–1403.

Griewank A, Walther A (2008). *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*. Siam.

Han C, Carlin BP (2001). "Markov Chain Monte Carlo Methods for Computing Bayes Factors: A Comparative Review." *Journal of the American Statistical Association*, **96**(455), 1122–1132.

Hoeting JA, Madigan D, Raftery AE, Volinsky CT (1999). "Bayesian Model Averaging: A Tutorial." *Statistical Science*, pp. 382–401.

Jeffreys H (1935). "Some Tests of Significance, Treated by the Theory of Probability." In *Proceedings of the Cambridge Philosophical Society*, volume 31, pp. 203–222.

Kass RE, Raftery AE (1995). "Bayes Factors." *Journal of the American Statistical Association*, **90**(430), 773–795.

Katsanevakis S (2006). "Modelling Fish Growth: Model Selection, Multi-model Inference and Model Selection Uncertainty." *Fisheries Research*, **81**(2), 229–235.

Link WA, Barker RJ (2009). *Bayesian Inference: With Ecological Applications*. Academic Press.

Ogle DH (2016). *FSAdata: Fisheries Stock Analysis, Datasets*. R package version 0.3.5.

Pav SE (2016). *madness: Automatic Differentiation of Multivariate Operations*. R package version 0.2.0, URL https://github.com/shabbychef/madness.

Plummer M, *et al.* (2003). "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, p. 125. Vienna.

Schwarz G, *et al.* (1978). "Estimating the Dimension of a Model." *The Annals of Statistics*, **6**(2), 461–464.

Spiegelhalter DJ, Best NG, Carlin BP, Van Der Linde A (2002). "Bayesian Measures of Model Complexity and Fit." *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, **64**(4), 583–639.

Von Bertalanffy L (1938). "A Quantitative Theory of Organic Growth (Inquiries on Growth Laws. II)." *Human Biology*, **10**(2), 181–213.

Watanabe S (2010). "Asymptotic Equivalence of Bayes Cross Validation and Widely Applicable Information Criterion in Singular Learning Theory." *Journal of Machine Learning Research*, **11**(Dec), 3571–3594.

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**, the models can be defined in an external text file, or in an R function using JAGS syntax. Here, we use text files `goalsPois.txt` and `goalsNB.txt`.

```
## Inside goalsPois.txt:
model{
  for(i in 1:n){
    y[i] ~ dpois(lambda)
  }
  lambda ~ dgamma(lamprior[1], lamprior[2])
  kappa ~ dnorm(0, 1/(sigma^2))  # precision
}

## Inside goalsNB.txt:
model{
  for(i in 1:n){
    y[i] ~ dnegbin(p, r)
  }
  p <- 1/(lambda*kappa + 1)
  r <- 1/kappa

  lambda ~ dgamma(lamprior[1],lamprior[2])
  kappa ~ dgamma(kapprior[1],kapprior[2])
}
```

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

```
library("R2jags")

inits = function(){list("lambda" = rgamma(1, 1, 0.1),
                        "kappa" = rgamma(1, 1, 0.1))}
params = c("lambda", "kappa")

jagsfit1 = jags(data = c('y', 'n', 'lamprior', 'sigma'), inits, params,
                n.iter=10000, model.file = "goalsPois.txt")

jagsfit2 = jags(data = c('y', 'n', 'lamprior', 'kapprior'), inits, params,
                n.iter=10000, model.file = "goalsNB.txt")
```

Finally, we define functions which randomly select a timestep and return the values of both parameters at that timestep. Note that JAGS includes an estimate of the deviance as a third parameter, which these functions exclude.

```
# Manually
fit1 = as.mcmc(jagsfit1); C1 = as.matrix(fit1)
draw1 = function(){rev(C1[sample(dim(C1)[1], 1, replace=T),
                          -which(colnames(C1) == "deviance")])}

# Using getsampler function
getsampler(jagsfit2, "draw2", order=c(3,2))   # alphabetically, lambda is 3rd parameter and kappa is 2nd
```

Defining coda-sampling functions for Example 2

```
## In fishGomp.txt:
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)
}

## In fishBert.txt:
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)
}
```

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

```
## 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")

## 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")

## Define samplers
getsampler(jagsfit1, "draw1")
getsampler(jagsfit2, "draw2", c(3,4,2,5))
```

Affiliation:

Nicholas Gelling
University of Otago
Department of Mathematics and Statistics
PO Box 56
Dunedin 9054
New Zealand
E-mail: ngelling@maths.otago.ac.nz