runjags: An R Package Providing Interface Utilities, Model Templates, Parallel Computing Methods and Additional Distributions for MCMC Models in JAGS

Matthew J. Denwood
University of Copenhagen

Abstract

The **runjags** package provides a set of interface functions to facilitate running Markov chain Monte Carlo models in **JAGS** from within R. Automated calculation of appropriate convergence and sample length diagnostics, user-friendly access to commonly used graphical outputs and summary statistics, and parallelized methods of running **JAGS** are provided. Template model specifications can be generated using a standard **lme4**-style formula interface to assist users less familiar with the BUGS syntax. Automated simulation study functions are implemented to facilitate model performance assessment, as well as drop-k type cross-validation studies, using high performance computing clusters such as those provided by **parallel**. A module extension for **JAGS** is also included within **runjags**, providing the Pareto family of distributions and a series of minimally-informative priors including the DuMouchel and half-Cauchy priors. This vignette is taken from the publication for the **runjags** package (Denwood 2016). It outlines the primary functions of this package, and gives an illustration of a simulation study to assess the performance of equivalent model specifications.

Keywords: MCMC, Bayesian, graphical models, interface utilities, **JAGS**, BUGS, R.

1. Introduction

Over the last two decades, the increased availability of computing power has led to a substantial increase in the availability and use of Markov chain Monte Carlo (MCMC) methods for Bayesian estimation (Gilks, Richardson, and Spiegelhalter 1998). However, such methods have potential drawbacks if used inappropriately, including difficulties in identifying convergence (Toft, Innocent, Gettinby, and Reid 2007; Brooks and Roberts 1998) and the potential for auto-correlation to decrease the effective sample size of the numerical integration process (Kass, Carlin, Gelman, and Neal 1998). Although writing MCMC sampling algorithms such as the Metropolis-Hastings algorithm (Hastings 1970) is relatively straightforward, many users employ software such as the Bayesian analysis Using Gibbs Sampling (BUGS) software variants **WinBUGS** and **OpenBUGS** (Lunn, Thomas, Best, and Spiegelhalter 2000). Just Another Gibbs Sampler (**JAGS**; Plummer 2003) is a cross-platform alternative with a direct interface to R using **rjags** (Plummer 2016), which can be easily extended with user-specified modules supporting additional distributions and random number generators (Wabersich and

Vandekerckhove 2014). Each of these uses the BUGS syntax to allow the user to define arbitrary models more easily, which is attractive and attainable for researchers who are more familiar with traditional modeling techniques. However, many of these less experienced users may not be aware of the potential issues with MCMC analysis, hence the prominent warning that "MCMC sampling can be dangerous" in the **WinBUGS** user manual (Lunn *et al.* 2000). Some of this potential risk for inexperienced users can be reduced using a wrapper for the model-fitting software that analyzes the model output for common problems, such as failure to converge, parameter auto-correlation and effective sample size, which may otherwise be overlooked by the end user.

Bayesian statistical methods, such as those used by BUGS and **JAGS**, also require prior belief to be incorporated into the model. There are a number of different recommendations for an appropriate choice of prior distribution under various different circumstances, for example the half-Cauchy distribution has been recommended as a reasonable choice for standard deviation parameters within hierarchical models (Gelman 2006; Polson and Scott 2011), and DuMouchel (1994) gives an argument for the use of $\pi(\tau) = \frac{s_0}{(s_0+\tau)^2}$ as a prior for a variance parameter τ in meta-analysis models. However, these are not available as built-in distributions in BUGS or **JAGS**.

This paper describes the **runjags** package (**?**) for R (R Core Team 2015) which can be used to automate MCMC fitting and summarizing procedures for **JAGS** models and is available from the Comprehensive R Archive Network (CRAN) at `https://cran.r-project.org/package=runjags`. The functions are designed to be user-friendly (particularly for those less experienced with MCMC analysis), and provide a number of features to make the recommended convergence and sample size checks more obvious to the end user. The **runjags** package also provides additional distributions to extend the core functionality of **JAGS**, including the half-Cauchy and DuMouchel distributions, as well as functions implementing different types of simulation studies to assess the performance of **JAGS** models. Section 3 gives a worked example of usage to assess the sensitivity of an over-dispersed count observation model to various minimally-informative prior distributions. Some prior familiarity with the BUGS programming language and the underlying MCMC algorithms is assumed. All code shown below is also included in an R file in the supplementary material.

2. Package functions

2.1. Preparation

The core functionality of the **runjags** package allows a model specified by the user to be run in **JAGS**, using the `run.jags` function. The help file for this function gives an overview of the core functionality of the **runjags** package and provides links to other relevant functions. All functions require installation of **JAGS**, which is an open source software package available from `https://mcmc-jags.sourceforge.io/`.

Before running a model for the first time, it is advisable to check the installation of **JAGS** and set any desired global settings such as installation locations and warning message preferences using the `runjags.options` function. For example, the following will first test the **JAGS** installation, and then set function feedback from **runjags** and simulation updates from **JAGS** to be suppressed for future model runs in this R session:

```
R> testjags()

You are using R version 3.3.0 (2016-05-03) on a unix machine, with the
X11 GUI
JAGS version 4.2.0 found successfully using the command
'/usr/local/bin/jags'
The rjags package is installed

R> runjags.options(silent.runjags = TRUE, silent.jags = TRUE)
```

The help file for the `runjags.options` function gives a list of other possible global options, and instructions on how to set these in the R profile file for permanent use.

2.2. Basic usage

The `run.jags` function requires a valid model definition to the `model` argument and a character string of monitored variables to the `monitor` argument before a model can be run. The model can be specified in an external text file, or as a character string within R. The former is likely to be preferable for more complex model formulations, but the latter eliminates the need for multiple text files. Data will be necessary for most models, and it is highly recommended to provide over-dispersed starting values for multiple chains; the default settings give a warning if no initial values are provided.

There are a number of ways to provide data and initial values, depending on the preferences of the user. It is possible for the text file containing the model to also contain data and initial value "blocks", in which case these will be automatically imported with the model by `run.jags` and the number of chains is inferred from the number of initial value lists found. This is also compatible with standard **WinBUGS** or **OpenBUGS** text files, although the addition of curly brackets is necessary to demarcate the data and initial value blocks in the same way as for the model block. It is also necessary to convert any BUGS arrays from row-major order to column-major order, which is done automatically if the variables are specified inside a list (as is the case for BUGS, but not for R). To over-ride this setting within a specific data or initial value block, the user can include `#BUGSdata#` to ensure all arrays are converted from row- to column-major order, `#Rdata#` to ensure none of the arrays are converted, and `#modeldata#` to pass the data block directly to **JAGS** for data transformation (see Section 7.0.4 of the **JAGS** user manual).

As a basic example, we can use the Salmonella example from Chapter 6.5.2 of the BUGS book (with thanks to Lunn, Jackson, Best, Thomas, and Spiegelhalter 2012, for permission to reproduce their model). Simulation-specific options can be provided to the `run.jags` function, which may include the required burn-in period, sampling length and thinning interval. A basic model run with a fixed burn-in period (default 4,000 iterations after 1,000 adaptive iterations) and sampling period (default 10,000 iterations) can be obtained as follows:

```
R> filestring <- "
+  The BUGS Book example Chapter 6.5.2
+  The following example has been modified only to include
+  curly brackets around the Data and Inits specifications
+
```

```
+  Poisson model...
+
+  model {
+    for (i in 1:6) {
+      for (j in 1:3) {
+        y[i,j] ~ dpois(mu[i])
+      }
+      log(mu[i]) <- alpha + beta * log(x[i] + 10) + gamma * x[i]
+    }
+    for (i in 1:6) {
+      y.pred[i] ~ dpois(mu[i])
+    }
+    alpha ~ dnorm(0, 0.0001)
+    beta ~ dnorm(0, 0.0001)
+    gamma ~ dnorm(0, 0.0001)
+  }
+
+  Data {
+    list(y = structure(.Data = c(15, 21, 29, 16, 18, 21, 16, 26, 33,
+      27, 41, 60, 33, 38, 41, 20, 27, 42), .Dim = c(6, 3)),
+      x = c(0, 10, 33, 100, 333, 1000))
+  }
+
+  Inits {
+    list(alpha = 0, beta = 0, gamma = 0)
+  }
+  "
R> results <- run.jags(filestring, monitor = c("alpha", "beta", "gamma"))

Warning message:
Convergence cannot be assessed with only 1 chain
```

A single chain was used for this model because only one set of initial values was found in the example file, resulting in the warning message regarding convergence assessment. The results of the simulation can be examined using the default print method as follows:

```
R> results

JAGS model summary statistics from 10000 samples (adapt+burnin = 5000):

         Lower95      Median     Upper95       Mean         SD      Mode
alpha     1.8018      2.1899      2.6135     2.1913    0.20439    2.1939
beta     0.20684     0.31538     0.41891    0.31414   0.053648    0.3157
gamma -0.0014678 -0.00099063 -0.00053861 -0.0009922 0.00023808 -0.001008

            MCerr MC%ofSD SSeff   AC.10 psrf
alpha    0.019523     9.6   110 0.78631   --
```

```
beta    0.0052936     9.9   103 0.81478   --
gamma 0.000019396     8.1   151 0.58568   --

Total time taken: 0.5 seconds
```

The results show similar inference to that provided by Lunn *et al.* (2012), although with additional information regarding the effective sample size (`SSeff`), auto-correlation at a lag of 10 (`AC.10`), and the potential scale reduction factor (`psrf`) of the Gelman-Rubin statistic (Gelman and Rubin 1992) for models with multiple chains (the latter is sometimes referred to as 'Rhat'). In this case, an insufficient number of samples has been taken for this highly auto-correlated model (although it is important to note that the auto-correlation is markedly reduced if the '`glm`' module is loaded in **JAGS**). Displaying the effective sample size with the summary information will alert the user to the fact that additional steps should be taken before sensible inference can be made.

The data can also be specified to `run.jags` using the `data` argument, in which case it should take the format of a named list, data frame, character string as produced by `dump.format`, or a function (with no arguments) returning one of these. Similarly, the initial values can be specified using the `inits` argument as a list with length equal to the number of chains, with each element specifying a named list, data frame or character string for the initial values for that chain. The initial values may also be specified as a function taking either no arguments (as for the `data` argument) or one argument (specifying the chain number), in which case an additional `n.chains` argument will be required by `run.jags` to determine the number of chains required.

2.3. Alternative usage

To facilitate a more streamlined function call within R, an alternative method of specifying data and initial values is provided. The model formulation may contain special inline comments including: `#data#`, which indicates that the comma separated variable names to the right of the statement are to be included in the simulation as data, and `#inits#`, which indicates variables for which initial values are to be provided. Any variables specified by `#data#` and `#inits#` will be automatically retrieved from a named list, data frame or environment passed to the `data` and `inits` argument (or function returning one of these), or from the global environment. Any variable names specified in this way may also match a function returning an appropriate vector, and in the case of initial values, this function may accept a single argument indicating the chain for which the initial values are to be used. Note that any variables specified by `#data#` or `#inits#` will be ignored if a character string is provided to the `data` or `inits` arguments, which may be useful for temporarily over-riding the values specified in the model file. See the `dump.format` function for a way to generate these. In addition to `#data#` and `#inits#`, a number of optional inline comments are supported as follows:

- `#monitors#` – a comma-separated list of monitored variables to use, which may include the special variables "DIC" (Spiegelhalter, Best, Carlin, and van der Linde 2002) and "PED" (Plummer 2008), which can be used to assess model fit;
- `#modules#` – a comma-separated list of any **JAGS** extension modules required, optionally also specifying the status (e.g., `#modules# glm on, dic on`);

- `#factories#` – a comma-separated list of any **JAGS** factories and types required, optionally also specifying the status (e.g., `#factories# mix::TemperedMix sampler on`);
- `#response#` – a single variable name specifying the response variable;
- `#residual#` – a single variable name specifying a variable that represents the residuals;
- `#fitted#` – a single variable name specifying a variable that represents the fitted value.

Each of these options can also be supplied directly to the relevant function call in R. An example of running a model using this style of model specification is as follows:

```
R> model <- "model {
+    for (i in 1:N) { #data# N
+      Y[i] ~ dnorm(true.y[i], precision) #data# Y
+      true.y[i] <- coef * X[i] + int #data# X
+    }
+    coef ~ dunif(-1000, 1000)
+    int ~ dunif(-1000, 1000)
+    precision ~ dexp(1)
+    #inits# coef, int, precision, .RNG.seed, .RNG.name
+    #monitor# coef, int, precision
+  }"
```

Simulate the data:

```
R> set.seed(1)
R> N <- 100
R> X <- seq(1, N, by = 1)
R> Y <- rnorm(N, 2 * X + 10, 1)
```

The following code specifies functions that return initial values (including RNG seeds) for each chain. The use of `switch` within these functions allows different initial values to be chosen for chains one and two, ensuring that initial values are over-dispersed.

```
R> coef <- function(chain)
+    return(switch(chain, "1" = -10, "2" = 10))
R> int <- function(chain)
+    return(switch(chain, "1" = -10, "2" = 10))
R> precision <- function(chain)
+    return(switch(chain, "1" = 0.01, "2" = 100))
R> .RNG.seed <- function(chain)
+    return(switch(chain, "1" = 1, "2" = 2))
R> .RNG.name <- function(chain)
+    return(switch(chain, "1" = "base::Super-Duper",
+      "2" = "base::Wichmann-Hill"))
```

It is then possible to run the simulation specifying only the model and the number of chains to use (the monitored variables, data and initial values are specified in the model file and will be retrieved form our R working environment):

```
R> results <- run.jags(model, n.chains = 2)
```

2.4. Extending models

The `autorun.jags` function can be used in the same way as `run.jags`, but the burn-in period and sample length are calculated automatically rather than being directly controlled by the user. The `autorun.jags` function will continually extend a simulation until convergence – as assessed by the Gelman-Rubin statistic (Gelman and Rubin 1992) – has been achieved for all monitored variables, and will then extend the simulation further to compensate for any observed auto-correlation. The automated assessment of convergence should be verified graphically before making inference from models fit to real data, but a fully automated analysis is useful for simulated data and for reinforcing the importance of convergence assessment for novice users. The following code will run the same model as above, extending the model as necessary up to a maximum total elapsed time of one hour:

```
R> results <- autorun.jags(model, n.chains = 2, max.time = "1hr")
```

Alternatively, an existing model may be extended by the user in order to increase the sample size of the MCMC chains using either the `extend.jags` or `autoextend.jags` function. For these functions, the arguments `add.monitor`, `drop.monitor` and `drop.chain` are provided in order to change the monitored variables and number of chains being run. The `combine` argument controls whether the old MCMC chains should be discarded, or combined with the new chains. For example, the following code will manually extend the existing simulation by 5000 iterations, and then extend the simulation again with automatic control of convergence and sample size diagnostics:

```
R> results <- extend.jags(results, sample = 5000)
R> results <- autoextend.jags(results)
```

In the second function call, the automated diagnostics run by `autoextend.jags` determine that the simulation has converged and already has an adequate sample size, so no additional samples are taken. For more details on these functions including detailed descriptions of the other arguments and additional examples, consult the help pages for `run.jags` and `autorun.jags`.

Once a valid '`runjags`' class object has been obtained, the full representation of the model, data and current status of the random number generators can be saved to a file using the `write.jagsfile` function. This allows a model to be run from the last sampled values using the `run.jags` function at a later time point, and it may also be instructive to use this function to examine the format of a syntactically-valid and complete model file that can be read directly using the `run.jags` function. It is also possible to specify a value of 0 for the `sample` argument in the original `run.jags` function call, and then subsequently use `write.jagsfile` to produce a model file with the initial values specified.

2.5. Visualization methods

The output of these functions is an object of class '`runjags`'. This class is associated with a number of S3 methods, as well as utility functions for combining multiple '`runjags`' objects (`combine.jags`), and for conversion to and from objects produced by the **rjags** package

(`as.runjags` and `as.jags`). Many of these allow a `vars` argument giving a subset of monitored nodes (using partial matching), as well as a `mutate` argument. This should specify a function (or a list with first element a function and remaining elements arguments to this function), and can be used to add new variables to the posterior chains that are derived from the directly monitored variables in **JAGS**. This allows the variables to be summarized or extracted as part of the MCMC objects as if they had been calculated in **JAGS**, but without the computational or storage overheads associated with calculating them directly in **JAGS**. One possible application for this is for pair-wise comparisons of different levels within fixed effects using the supplied `contrasts.mcmc` function.

The `print` method displays relevant overview information, including summary statistics for monitored variables calculated and stored by the `run.jags` function. The `summary` method returns a summary table for the monitored variables, which is taken from the stored values created by `run.jags` if available; otherwise it will be recalculated during the function call. Alternatively, summary statistics can be recalculated and stored in the '`runjags`' object using the `add.summary` function. There are a series of options available to these summary functions, including `vars` and `mutate` as outlined above, `confidence` which specifies a numeric vector of confidence intervals to calculate, and `custom` which allows one or more statistics calculated by a user-supplied function to be appended to the summary statistics. Note that summary options may also be passed to `run.jags` in order to control the summary statistics calculated and appended to the '`runjags`' object.

The `plot` method produces a series of relevant plots for the selected variables, including trace plots, empirical cumulative distribution function plots, histograms, and a cross-correlation plot, with additional options allowing auto-correlation plots and density plots if desired. Further plot parameters can be specified using the `col` and `separate.chains` arguments, as well as a named list for each plot type which will be passed to the underlying **lattice** functions (**?**). The primary intention with these plots is to provide rapid access to commonly used convergence diagnostics, and `plot` methods associated with '`mcmc`' or '`mcmc.list`' objects may be more flexible and intuitive for producing more specific graphical output from converged MCMC chains. The **coda** package (Plummer, Best, Cowles, and Vines 2006) provides such plotting methods, as well as many of the underlying functions that calculate the summaries given by **runjags**. A typical examination of a simulation output (the default print method, and a plot output for variable names partially matching the letter "c") could be obtained as follows:

```
R> results

JAGS model summary statistics from 30000 samples (chains = 2;
adapt+burnin = 5000):

          Lower95  Median Upper95    Mean        SD   Mode        MCerr MC%ofSD
coef       1.9922  1.9995   2.007  1.9996 0.0038115 1.9994 0.000073962     1.9
int        9.4388  9.8711  10.307  9.8692   0.22139 9.8768   0.0042942     1.9
precision 0.61225 0.82946  1.0685 0.83515   0.11731  0.826   0.0007013     0.6

          SSeff     AC.10   psrf
coef       2656   0.16954  1.002
int        2658   0.16942 1.0018
```

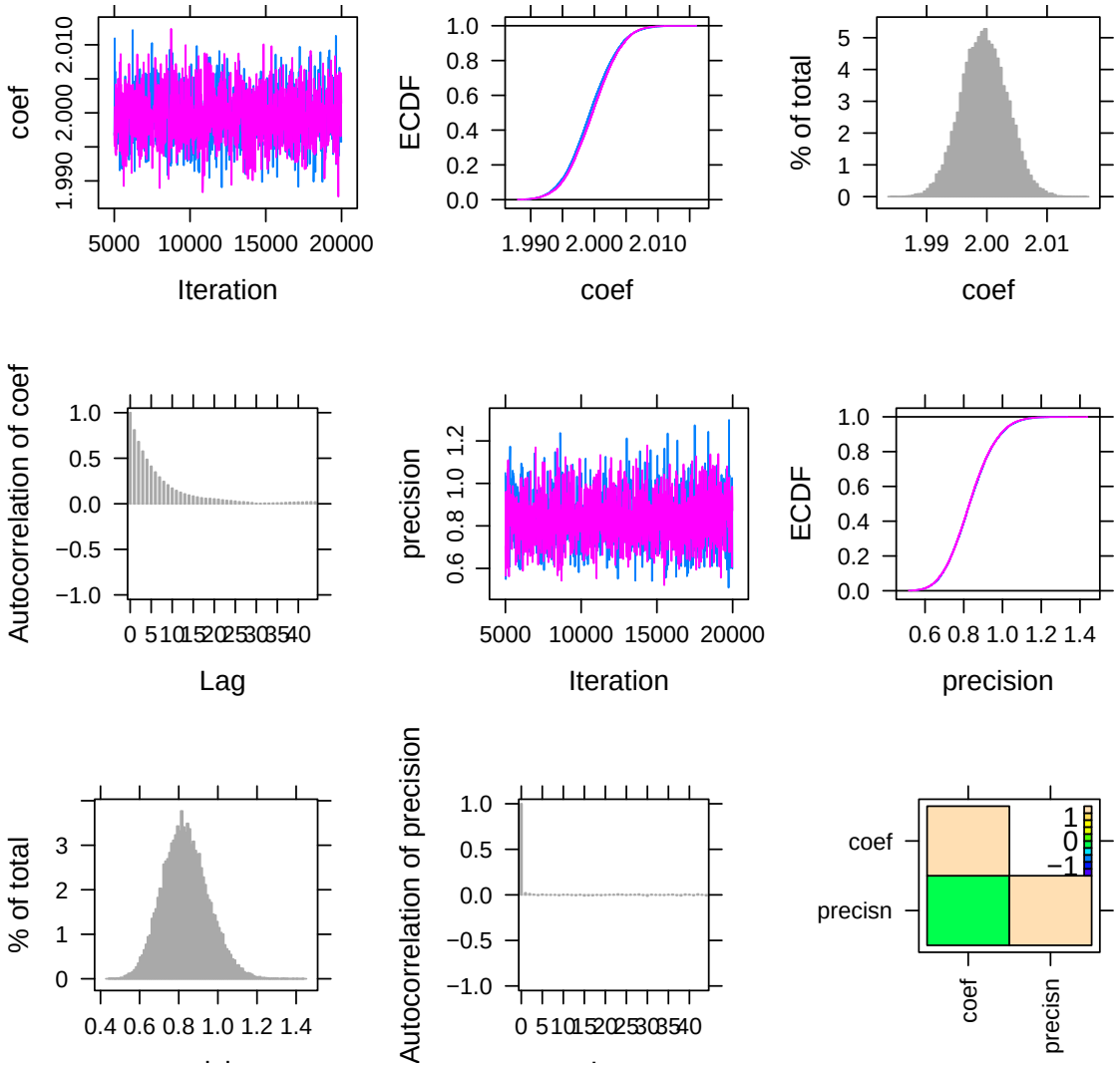


Figure 1: A series of plots displayed by the `plot` method for the '`runjags`' class, showing only parameters partially matched using the letter "c" with plots shown in a 3 × 3 layout. Multiple chains are shown using different colors as indicated by the key plot.

```
precision 27982 0.0020679      1

Total time taken: 4.3 seconds
```

```
R> plot(results, vars = "c", layout = c(3, 3))
```

The standard `plot` method presents the commonly required information in an easily readable format (including model fit statistics where available), but the same information can be returned in the form of a numeric matrix using the `summary` method. To extract additional information from the '`runjags`' object not covered by these summary statistics, see the `extract` method.

2.6. GLMM templates

There are many available frameworks for fitting standard generalized linear mixed models (GLMMs) in R, but new users to MCMC may find that running relatively simple models in **JAGS** and comparing the results to those obtained through other software packages allows them to better understand the flexibility and syntax of BUGS models. To this end, the **runjags** package provides a `template.jags` function which generates model specification files based on a formula syntax similar to that employed by the well-known **lme4** package (Bates, Maechler, Bolker, and Walker 2014; **?**). After generating the template model, the user is encouraged to examine the model file and make whatever changes are necessary before running the model using `run.jags`. For example, a basic generalized linear model (from the help file for `glm`) can be compared to the output of **JAGS** as follows:

```
R> counts <- c(18, 17, 15, 20, 10, 20, 25, 13, 12)
R> outcome <- gl(3, 1, 9)
R> treatment <- gl(3, 3)
R> d.AD <- data.frame(treatment, outcome, counts)
R> glm.D93 <- glm(counts ~ outcome + treatment, family = poisson())
R> template.jags(counts ~ outcome + treatment, data = d.AD,
+    family = "poisson")

Your model template was created at "JAGSmodel.txt" - it is highly
advisable to examine the model syntax to be sure it is as intended
You can then run the model using run.jags("JAGSmodel.txt")

R> jags.D93 <- run.jags("JAGSmodel.txt")
```

The results of these comparisons are not displayed here, but show how the same inference is presented slightly differently in a Bayesian framework. The `template.jags` function supports Gaussian, (zero-inflated) binomial, (zero-inflated) Poisson and (zero-inflated) negative binomial distributions, as well as linear and fixed effects, 2-way interactions and random intercept terms specified using the same syntax as used in **lme4**. Additional distributions and link functions can be introduced by manually editing the template model file. All necessary data, initial values, monitored variables and modules are saved to the model file using the previously described comment syntax, and the template function also saves information about the response variable, fitted estimates and residuals to the model file, allowing `residuals` and `fitted` methods to be used with the objects returned by `run.jags`.

2.7. JAGS module

In addition to the R code used to facilitate running **JAGS** models and summarizing results, the **runjags** package also provides a modular extension to the **JAGS** language, providing additional distributions. The module can be loaded using the following command:

```
R> load.runjagsmodule()

module runjags loaded
```

This makes the module available to any **JAGS** model, including those run using the **rjags** package. The available distributions extend the Pareto Type I distribution provided within **JAGS** to Pareto Types II, III and IV, as well as providing the generalized Pareto distribution, the Lomax distribution (a special case of the Pareto Type II distribution with $\mu = 0$), and two distributions advocated for use as "minimally-informative" priors for variance parameters: the DuMouchel distribution (DuMouchel 1994), and the half-Cauchy distribution (Gelman 2006). The usage, probability density function (PDF) and lower bound for the support of each of the distributions provided by the module are shown in Table 1, and an example of how to use the distributions in this module is given in Section 3.

2.8. Method options

There are a number of different methods for calling **JAGS** from within R using **runjags**, which can be controlled using the `method` argument or by changing the global option using the `runjags.options` function. The main difference between these is that some allow multiple chains to be run in parallel using separate **JAGS** models, with automatic pseudo-random number generation handled by **runjags** where necessary. The `"interruptible"`, `"rjags"`, `"parallel"` or `"bgparallel"` methods are recommended for most situations, but all possible methods and their advantages and disadvantages are summarized in Table 2. Note that a pre-existing cluster created using the **parallel** package can be used by specifying a `cl` argument, and a maximum number of parallel simulations for these methods can optionally be specified using a `n.sims` argument to the main function call (the default will use a separate simulation per chain, but it is possible to specify fewer simulations than chains). The model fit statistics are not available with parallel methods because multiple chains within the same model are required for calculation of DIC and PED, but these can be obtained using the `extract` method which will extend the simulation using a single simulation. The adaptation phase is always explicitly controlled to allow MCMC simulations with the same pseudo-random number seed to be reproducible regardless of the method used to call **JAGS**.

The two background methods do not return a completed simulation, but instead create a folder in the working environment where the simulation results will be written once the **JAGS** process has completed. For example, the following code will allow a **JAGS** simulation to be run in the background using two processors in parallel, and saving the results in a folder called '`mysimulation`' in the current working directory:

```
R> info <- run.jags(model, n.chains = 2, method = "bgparallel",
+    keep.jags.files = "mysimulation", n.sims = 2)

Starting the simulations in the background...
The JAGS processes are now running in the background
```

This returns the control of the terminal to the user, who can then carry on working in R while waiting for the simulation to complete. The default behavior on completion of the simulations is to alert the user by emitting a beep from the speakers, but configuration using `runjags.options` allows a shell script file to be executed instead. The `info` variable in this code contains the name and directory of the simulation, which is given to the user if the object is printed. The results can be retrieved using either the folder name or the variable returned by the function that started the simulation:

Name	Usage in **JAGS**	Density	Lower
Pareto I[1]	`dpar1(alpha, sigma)` $\alpha > 0$, $\sigma > 0$	$\alpha\,\sigma^{\alpha}\,x^{-(\alpha+1)}$	σ
Pareto II	`dpar2(alpha, sigma, mu)` $\alpha > 0$, $\sigma > 0$	$\frac{\alpha}{\sigma}\left(\frac{\sigma + x - \mu}{\sigma}\right)^{-(\alpha+1)}$	μ
Pareto III	`dpar3(sigma, mu, gamma)` $\sigma > 0$, $\gamma > 0$	$\frac{\left(\frac{x-\mu}{\sigma}\right)^{\frac{1}{\gamma}-1}\left(\frac{x-\mu}{\sigma}^{\frac{1}{\gamma}} + 1\right)^{-2}}{\gamma\,\sigma}$	μ
Pareto IV	`dpar4(alpha, sigma, mu, gamma)` $\alpha > 0$, $\sigma > 0$, $\gamma > 0$	$\frac{\alpha\left(\frac{x-\mu}{\sigma}\right)^{\frac{1}{\gamma}-1}\left(\frac{x-\mu}{\sigma}^{\frac{1}{\gamma}} + 1\right)^{-(\alpha+1)}}{\gamma\,\sigma}$	μ
Lomax[2]	`dlomax(alpha, sigma)` $\alpha > 0$, $\sigma > 0$	$\frac{\alpha}{\sigma}\left(1 + \frac{x}{\sigma}\right)^{-(\alpha+1)}$	0
Gen. Par.	`dgenpar(sigma, mu, xi)` $\sigma > 0$	$\frac{1}{\sigma}\left(1 + \xi\frac{x-\mu}{\sigma}\right)^{-\left(\frac{1}{\xi}+1\right)}$ For $\xi = 0$: $\frac{1}{\sigma}\,e^{\frac{-(x-\mu)}{\sigma}}$	μ[3]
DuMouchel	`dmouch(sigma)` $\sigma > 0$	$\frac{\sigma}{(x+\sigma)^2}$	0
Half-Cauchy	`dhalfcauchy(sigma)` $\sigma > 0$	$\frac{2\sigma}{\pi\,(x^2+\sigma^2)}$	0

Table 1: Distributions provided by the **JAGS** module included with the **runjags** package. The name, **JAGS** code with parameterization, PDF and lower bound of the distributions are shown. All distributions have an upper bound of ∞ unless otherwise stated.
[1]This is equivalent to the `dpar(alpha, c)` distribution and provided for naming consistency.
[2]This is referred to as the "2nd kind Pareto" distribution by Van Hauwermeiren and Vose (2009); an alternative form for the PDF of this distribution is given by: $\frac{\alpha\,\sigma^{\alpha}}{(x+\sigma)^{\alpha+1}}$.
[3]The Generalized Pareto distribution also has an upper bound of $x \leq \mu - \frac{\sigma}{\xi}$ for $\xi < 0$.

```
R> background.results <- results.jags("mysimulation")
```

If the simulation has not yet completed, the `results.jags` function will display the **JAGS** output so that the user can gauge how much longer the simulation will take. Further op-

Method name	Description	Method options
`"interruptible"`[1]	**JAGS** called using a shell, with output monitored and displayed within R.	–
`"rjags"`[1,2]	**JAGS** called using the **rjags** package.	`by` and `progress.bar`: as for the **rjags** package.
`"background"`[1] *	**JAGS** called as a background process, with the R prompt returned to the user.	–
`"simple"`[1]	**JAGS** called directly using a shell.	–
`"parallel"`[3]	Multiple **JAGS** instances called using separate shells to allow chain parallelization.	`n.sims`: the number of parallel simulations.
`"bgparallel"`[3] *	Multiple **JAGS** instances called using separate background processes to allow chain parallelization.	`n.sims`: the number of parallel simulations.
`"rjparallel"`[3,4]	Multiple **rjags** models run within R using a **parallel** cluster.	`cl`: a pre-created cluster to be used, and `n.sims`: the number of parallel simulations.
`"snow"`[1]	Multiple **JAGS** instances called using separate shells set up using a **parallel** cluster.	`cl` and `n.sims`: as above, and `remote.jags`: the **JAGS** path on the cluster nodes.

Table 2: Methods provided by the **runjags** package to run simulations in **JAGS**. Availability of **JAGS** modules is as follows:
[1]Installed in **JAGS**.
[2]Loadable in the R session.
[3]Installed in **JAGS** (except DIC).
[4]Loadable in R code run remotely on the cluster nodes (except DIC).
*These methods are not compatible with `autorun.jags` and `autoextend.jags`.

tions for the `results.jags` function include `recover.chains` which allows the results of successful simulations to be read even if other parallel simulations did not produce output, and `read.monitor` which allows only a chosen subset of the monitored variables to be read from the MCMC output. For all methods except `"rjags"` and `"rjparallel"`, any calls to `run.jags` where the `keep.jags.files` argument is specified will result in a folder being created in the working directory that can be reimported using `results.jags`. Any failed simulations created are also kept using the same mechanism, and a message is displayed detailing how the user can attempt to recover these simulations. These failed simulation folders are automatically cleaned up when the R session is terminated. The `failed.jags` function returns any output captured from **JAGS** in such cases, and is helpful to debug model code.

2.9. Simulation studies

One of the principle motivations behind the development of the **runjags** package is to automate the analysis of simulated data sets for the purposes of model validation. A common motivation for this type of analysis is a drop-k validation study, also known as a leave-one-out cross-validation where $k = 1$. This procedure re-fits the same model to a single data set multiple times, with one or more of the observed data points removed from each re-fit of the model. This can either be a randomly selected group of a fixed number "k" of data points, or each individual data point in turn. The goal is to evaluate the ability of the model to predict each observation from the explanatory variables, so that any unusual observations can be identified. While it is possible to repeatedly use the `autorun.jags` function to analyze multiple data sets, the higher level `run.jags.study` and `drop.k` functions are provided to automate much of this process. Large simulation studies are likely to be computationally intensive, but are ideal candidates for parallelization. For this reason, parallel computation is built directly into these functions using the **parallel** package. This can be used to parallelize the simulation locally, or to run the simulation on any cluster set up using the **snow** package (Tierney, Rossini, Li, and Sevcikova 2013). This allows for the maximization of the available computing power without requiring the end user to write any additional code, and includes an initial check to ensure that the model compiles and runs locally before beginning the parallelized study.

A drop-k study is implemented in **runjags** using the `drop.k` function as follows. The '`runjags`' class object on which the drop-k analysis will be performed must first be obtained using the `run.jags` function. Here, we will use the simple linear regression model obtained in Section 2.3, with the result of `run.jags` contained in the variable `results`. The `drop.k` function takes arguments `drop` (indicating the data variables to remove between simulations), and `k` (indicating the number of data points to drop for each simulation). In this case, a drop-1 study is run with the number of simulations equal to the number of data points. All individual simulations are run using the underlying `autorun.jags` function; additional arguments for `autorun.jags` can be passed through `drop.k` as required. The initial values for each simulation are taken from the parent simulation, including the observed values of the removed data points to ensure that the model will compile. The drop-1 study is run and the results displayed using the following syntax (limited to the first five data-points for brevity):

```
R> assessment <- drop.k(results, drop = "Y[1:5]", k = 1)
R> assessment

Values obtained from a drop-k study with a total of 5 simulations:

     Target Median   Mean Lower95%CI Upper95%CI Range95%CI Within95%CI
Y[1] 11.544 11.885 11.879     9.6268     14.055     4.4282           1
Y[2] 13.051 13.914 13.906     11.549     15.994     4.4445           1
Y[3] 15.828 15.888 15.878     13.559     18.059     4.5009           1
Y[4] 18.784 17.832  17.83     15.663      20.12     4.4572           1
Y[5] 21.398 19.821 19.821     17.657     22.086     4.4285           1

     AutoCorr(Lag10)
Y[1]        0.010115
Y[2]        0.014556
```

```
Y[3]       0.0015873
Y[4]       0.0047826
Y[5]       0.0064177

Average time taken:  2.6 seconds (range: 2.5 seconds - 2.7 seconds)
Average adapt+burnin required:  5000 (range: 5000 - 5000)
Average samples required:  10506 (range: 10000 - 11292)
```

The results show the 95% confidence interval (CI) for each data point obtained from the corresponding simulation where this data point was removed, which in this case indicates that the first five data-points were predicted reasonably well. For drop-k cross-validation with "k" greater than 1, the indicated number of data points will be randomly removed from each simulation and the average values for the corresponding summary statistics from each data point will be shown. In this case, the argument `simulations` must also be provided. Additional arguments to `autorun.jags` can also be provided to the `drop.k` function. For example, the following syntax will run 100 simulations with a random selection of 2 of the 5 first five data-points removed from each:

```
R> assessment <- drop.k(results, drop = "Y[1:5]", k = 2, simulations = 100,
+    method = "simple", psrf.target = 1.1)
R> assessment

Average values obtained from a drop-k study with a total of 100 simulations:

     Target Av.Median Av.Mean Av.Lower95%CI Av.Upper95%CI Av.Range95%CI
Y[1] 11.544    11.879  11.874        9.6118        14.079        4.4667
Y[2] 13.051    13.894  13.899        11.667         16.13        4.4629
Y[3] 15.828    15.868  15.871        13.636        18.088        4.4516
Y[4] 18.784    17.847  17.853        15.627        20.036        4.4082
Y[5] 21.398    19.822  19.827        17.656        22.071        4.4152

     Prop.Within95%CI Av.AutoCorr(Lag10) Simulations
Y[1]                1           0.010223          41
Y[2]                1          0.0066466          39
Y[3]                1          0.0032953          42
Y[4]                1           0.012031          38
Y[5]                1          0.0076121          40

Average time taken:  6.2 seconds (range: 3.4 seconds - 7.4 seconds)
Average adapt+burnin required:  5000 (range: 5000 - 5000)
Average samples required:  10645 (range: 10000 - 12182)
```

In the latter case, inference was made on each data point in several different data sets, so the results present the mean values of each summary statistic obtained from the multiple simulations.

The `drop.k` function is a wrapper for the `run.jags.study` function, which can be used to perform various different types of simulation studies. This function takes the following

arguments: the number of data sets to analyze, the model to use, a function to produce data that will be provided to each simulation, and a named list of "target" variables with true values representing parameters to be monitored and used to summarize the output of the simulation. Inline `#monitor#` statements can be used as with `run.jags`, and any target variables are also automatically monitored. Any variables specified using the inline `#data#` statement will be retrieved from the working environment as usual and will be common to all simulations – data which is intended to change between simulations must therefore be provided using the `datafunction` argument instead. Initial variables can be specified using `#inits#` in the model file, but it is also necessary to pass a character string of all variable names required to the `export.cluster` argument to ensure these variables are visible on the cluster nodes. It may be preferable to specify initial values as a function, to which the data will be made available by `run.jags` at run time (this may be required in cases where the choice of appropriate initial values depends on the values in the data). An illustration of the `run.jags.study` function is provided in Section 3.

3. Illustration of usage with a simulation study

Here we will consider a worked example of a simulation study analysis using **runjags**, in order to assess the performance of two equivalent model formulations with two different "minimally-informative" priors. The application is an over-dispersed count model, the use of which is widespread in many biological fields (Bolker, Brooks, Clark, Geange, Poulsen, Stevens, and White 2009), including parasitology (Wilson, Grenfell, and Shaw 1996; Wilson and Grenfell 1997; Shaw, Grenfell, and Dobson 1998), where Bayesian methods of analysis have been shown to provide more robust inference than traditional methods (Denwood, Stear, Matthews, Reid, Toft, and Innocent 2008; Denwood, Reid, Love, Nielsen, Matthews, McKendrick, and Innocent 2010).

3.1. Model formulation and assessment

The gamma distribution is parameterized in **JAGS** and BUGS by the `shape` (α) and `rate` (β) parameters, with the expectation given by $\frac{\alpha}{\beta}$ and variance given by $\frac{\alpha}{\beta^2}$. This distribution can be used to describe underlying variability in a Poisson observation, representing an unknown amount of over-dispersion between observations. In this situation the extra-Poisson coefficient of variation cv is a useful measure of the variability of the underlying gamma distribution, and is a simple function of the `shape` parameter: $cv = \sqrt{\frac{1}{\alpha}}$

A candidate **JAGS** Model A (using inline data and monitor statements to be detected by **runjags**) is as follows:

```
R> modelA <- "model {
+    for (i in 1:N) {
+      Count[i] ~ dpois(lambda[i])
+      lambda[i] ~ dgamma(shape, rate)
+    }
+    shape ~ dmouch(1)
+    mean ~ dmouch(1)
+    rate <- shape / mean
```

```
+
+    #data# N
+    #modules# runjags
+    #monitor# mean, shape
+  }"
```

This model allows each observed `Count` to follow a Poisson distribution with `lambda` drawn from a gamma distribution with `shape` parameter to be estimated, and `rate` parameter calculated from the `shape` parameter and the `mean` of the distribution, which is also to be estimated. The prior distribution used for the `mean` and `shape` parameters is the DuMouchel prior distribution as shown in Table 1 – this distribution is provided by the **runjags** extension module which can be loaded using the `#modules#` tag. Here we use the same minimally-informative prior distribution for both `shape` and `mean` parameters. The `#data#` statement is used to include `N` as data that does not change between simulations. The `Count` variable is also observed, but will vary between simulations so it is not retrieved from R memory using `#data#`.

An alternative formulation of this same model could be provided using a negative binomial distribution rather than a gamma mixture of Poisson distributions, as represented in Model B:

```
R> ModelB <- "model {
+    for (i in 1:N) {
+      Count[i] ~ dnegbin(prob, shape)
+    }
+
+    shape ~ dmouch(1)
+    mean ~ dmouch(1)
+    prob <- shape / (shape + mean)
+
+    #data# N
+    #modules# runjags
+    #monitor# mean, shape
+  }"
```

In this model, the same priors are placed on the parameters `shape` and `mean`, but the negative binomial distribution is parameterized by a probability `p` in place of the parameter `mean`. However, the gamma-Poisson and negative binomial distributions are equivalent (see Appendix A), and these models share the same prior distributions for the two parameters of interest. The two might therefore be expected to give equivalent inference.

The posterior coverage and auto-correlation of these models can be assessed using simulation studies, with data generated from a distribution with a mean of 2, *cv* of 1.1, and sample size of 20. These values are chosen to exaggerate any model performance issues by providing a comparatively small data set with a large number of zero observations, and are similar to those typically found in veterinary parasitological data sets (Denwood 2010). The two parameters of interest are the `mean` parameter which is directly monitored in the model, and the *cv* parameter which is a function of the monitored `shape` parameter. Rather than calculate the *cv* parameter in **JAGS**, this can be calculated more efficiently in R using a mutate function:

```
R> getcv <- function(x)
+    return(list(cv = sqrt(1 / x[, "shape"])))
```

The model performance assessment can be automated using `run.jags.study` by creating a function to return a pre-generated simulated data set for each simulation:

```
R> N <- 20
R> S <- 1000
R> truemean <- 2
R> truecv <- 1.1
R> trueshape <- 1 / truecv^2
R> truerate <- trueshape / truemean
R> set.seed(1)
R> alldata <- lapply(1:S, function(x) {
+    return(rpois(N, rgamma(N, trueshape, rate = truerate)))
+  })
R> datafunction <- function(i) return(list(Count = alldata[[i]]))
```

In this case we specify the initial values as a function, illustrating the potential to make use of the stochastically-generated data while creating the initial values within the function:

```
R> initsfunction <- function(chain) {
+    stopifnot(data$N == 20)
+    stopifnot(chain %in% c(1, 2))
+    shape <- c(0.1, 10)[chain]
+    mean <- c(10, 0.1)[chain]
+    .RNG.seed <- c(1, 2)[chain]
+    .RNG.name <- c("base::Super-Duper", "base::Wichmann-Hill")[chain]
+    return(list(shape = shape, mean = mean, .RNG.seed = .RNG.seed,
+      .RNG.name = .RNG.name))
+  }
```

Finally, a **parallel** cluster with 10 nodes is set up on the local machine, before the two simulation studies are run on this cluster using the same data. The `run.jags.study` function will check each of the models locally using a single randomly chosen data set to ensure that the model is valid before it is passed to the cluster:

```
R> library("parallel")
R> cl <- makeCluster(10)
R> resultsA <- run.jags.study(S, ModelA, datafunction,
+    targets = list(mean = truemean, cv = truecv), cl = cl,
+    inits = initsfunction, n.chains = 2, mutate = getcv)
R> resultsB <- run.jags.study(S, ModelB, datafunction,
+    targets = list(mean = truemean, cv = truecv), cl = cl,
+    inits = initsfunction, n.chains = 2, mutate = getcv)
```

Each function call returns an object of class '`runjagsstudy`', with a default `print` method that summarizes the results as for `drop.k`:

```
R> resultsA

Average values obtained from a JAGS study with a total of 1000 simulations:

     Target Av.Median Av.Mean Av.Lower95%CI Av.Upper95%CI Av.Range95%CI
mean      2    1.9689  2.0756       0.99815         3.354        2.3559
cv      1.1    1.0627  1.0909       0.52457        1.6983        1.1737

     Prop.Within95%CI Av.AutoCorr(Lag10) Simulations
mean             0.93           0.043658        1000
cv              0.926            0.19093        1000

Average time taken:  5.2 seconds (range: 2.3 seconds - 12.4 seconds)
Average adapt+burnin required:  5550 (range: 5000 - 27000)
Average samples required:  10059 (range: 10000 - 21465)

R> resultsB

Average values obtained from a JAGS study with a total of 1000 simulations:

     Target Av.Median Av.Mean Av.Lower95%CI Av.Upper95%CI Av.Range95%CI
mean      2    1.9632   2.069        1.0007        3.3294        2.3287
cv      1.1    1.0625  1.0891       0.52306        1.6905        1.1675

     Prop.Within95%CI Av.AutoCorr(Lag10) Simulations
mean            0.925           0.015783        1000
cv              0.931           0.040415        1000

Average time taken:  4.7 seconds (range: 1.9 seconds - 10.1 seconds)
Average adapt+burnin required:  5099 (range: 5000 - 16000)
Average samples required:  10000 (range: 10000 - 10000)
```

The inference made from the two models indicates that they are generally similar, except that the auto-correlation for both parameters is reduced for Model B, meaning that on average fewer samples were required for this model. As would be expected, the 95% confidence intervals for both parameters identified the true value approximately 95% of the time.

3.2. Sensitivity to prior distributions

The ability to incorporate prior information is an advantage of Bayesian methods, but there is often a variety of potential distributions that could be equally justifiable in a given situation. The choice between these possibilities is known to affect the shape of the posterior in some situations (Lele and Dennis 2009), particularly when the information in the data is relatively sparse. In particular, there are various different minimally-informative priors advocated for use with variance parameters in hierarchical models, including the `Gamma(0.001, 0.001)` distribution which is characterized by a mean of one and a very large variance. The sensitivity of a model to the choice of priors between this gamma prior and the DuMouchel prior can

Parameter	Priors: mean | shape	Mean	CI Range	Within CI	AC10	Simulations
mean	dmouch | dmouch	2.069	2.329	0.925	0.016	1000
mean	dmouch | dgamma	2.072	2.340	0.916	0.021	1000
mean	dgamma | dmouch	2.150	2.527	0.934	0.032	1000
mean	dgamma | dgamma	2.156	2.556	0.922	0.041	1000
cv	dmouch | dmouch	1.089	1.167	0.931	0.040	1000
cv	dmouch | dgamma	1.068	1.270	0.882	0.089	1000
cv	dgamma | dmouch	1.093	1.173	0.933	0.041	1000
cv	dgamma | dgamma	1.073	1.277	0.881	0.090	1000

Table 3: Average values for the inference on the mean parameter (true value 2) and *cv* parameter (true value 1.1) obtained from a negative binomial MCMC model formulation using DuMouchel and gamma priors for the mean and shape parameters.

be evaluated using the `run.jags.study` function, with a total of four candidate sets of priors (using each combination of DuMouchel and gamma distributions for the `mean` and `shape` parameters). These were applied to the same 1,000 simulated data sets using Model B and very similar R code to that given above. The results of these four simulation studies are shown in Table 3. There are small but noticeable differences between the inference made for both parameters using these prior distributions. The bias and auto-correlation are both approximately doubled for the `mean` parameter between DuMouchel and gamma priors, and more substantial changes in bias and auto-correlation are seen between priors for the *cv* parameter. In addition, the 95% confidence intervals for the *cv* parameter have less than 90% coverage when using the gamma prior, despite a slightly larger average range of these confidence intervals relative to the DuMouchel prior.

3.3. Discussion

The results presented here demonstrate the utility of simulation studies facilitated by the **runjags** package to evaluate the relative performance of alternative model formulations and the effect of prior distribution choices. In this case, the DuMouchel prior out-performed the more standard gamma prior, and it also possesses properties that are theoretically desirable for a minimally-informative distribution, such as invariance to inverse transformation, infinite variance and a mode of zero. DuMouchel (1994) proposed this prior for use with variance parameters in hierarchical models, but it has also been used in situations outside the meta-analysis application for which it was originally devised (see for example Phillips, Tam, Conti, Rodrigues, Brown, Iturriza-Gomara, Gray, and Lopman 2010; Conti, Presanis, van Veen, Xiridou, Donoghoe, Rinder Stengaard, and De Angelis 2011; Yin, Conti, Desai, Stafford, Slater, Gill, and Simms 2013). Christiansen and Morris (1997) also used the same distribution as a prior for a hierarchical regression model, and Daniels (1999) uses a uniform shrinkage prior which is equivalent to the DuMouchel distribution. Although this connection is not stated directly by DuMouchel (1994), the distribution is equivalent to a Lomax distribution with $\tau = x$, $s_0 = \sigma$ and $\alpha = 1$, and therefore to a Pareto type II distribution with $\tau = x$, $s_0 = \sigma$, $\alpha = 1$ and $\mu = 0$ (Table 1). The choice of σ dictates the median – a value of 1 is advocated since this also ensures invariance to the inverse transformation of τ, so this prior is equivalent in terms of variance and precision. The half-Cauchy distribution has a similar form

to the DuMouchel distribution, and has also been suggested for use as a prior for variance parameters (Gelman 2006; Polson and Scott 2011).

Although it is also possible to extend other variants of BUGS, **JAGS** is fully open source and written in C++, making extension modules such as the one provided by **runjags** much easier to implement. A very useful tutorial on writing and installing a standalone **JAGS** module is provided by Wabersich and Vandekerckhove (2014), but it is arguably more straightforward to implement a shared **JAGS** library inside an R package. The configure script provided inside the **runjags** package sets up the necessary environmental variables for compilation on any platform, and can be used as a template for creating additional extension modules within R packages.

4. Summary

There are several advantages to using MCMC, but also some potential disadvantages associated with failure to identify poor convergence and high Monte Carlo error. The **runjags** package attempts to partially safeguard against some of these difficulties by calculating and automatically reporting convergence and sample length diagnostics every time a **JAGS** model is run, and provides a more user-friendly way to access commonly used visual convergence diagnostics and summary statistics. Implementations of common GLMMs are provided using a standard formula-style interface, in order to encourage new users to explore the potential of MCMC inference without having to generate the full code for the model themselves. A further application of the **runjags** package is in implementing simulation studies so that model formulations and prior specifications can be validated using techniques such as drop-k cross-validation studies. Given that the inference made using **JAGS** and BUGS can be sensitive to subtly different model specifications and prior distributions, a user-friendly mechanism to perform these types of analyses is potentially very useful.

Acknowledgments

The author is grateful to the anonymous referees for their very useful comments and suggestions, to Stefano Conti for useful discussions regarding the Pareto family of distributions, to Vaetta Editing for proofreading this manuscript, and to the authors of "The BUGS Book" (Lunn *et al.* 2012) for kind permission to use the Salmonella example.

References

Bates D, Maechler M, Bolker B, Walker S (2014). ***lme4****: Linear mixed-effects models using Eigen and S4*. R package version 1.1-7, URL https://cran.r-project.org/package=lme4.

Bolker BM, Brooks ME, Clark CJ, Geange SW, Poulsen JR, Stevens MHH, White JSS (2009). "Generalized Linear Mixed Models: a Practical Guide for Ecology and Evolution." *Trends in Ecology & Evolution*, **24**(3), 127–35. ISSN 0169-5347. URL https://dx.doi.org/10.1016/j.tree.2008.10.008.

Brooks SP, Roberts GO (1998). "Assessing Convergence of Markov Chain Monte Carlo Algorithms." *Statistics and Computing*, **8**, 319–333.

Christiansen CL, Morris CN (1997). "Hierarchical Poisson Regression Modeling." *Journal of the American Statistical Association*, **92**, 618–632. ISSN 0162-1459. doi:10.1080/01621459.1997.10474013. URL https://www.tandfonline.com/doi/abs/10.1080/01621459.1997.10474013.

Conti S, Presanis AM, van Veen MG, Xiridou M, Donoghoe MC, Rinder Stengaard A, De Angelis D (2011). "Modeling of the HIV Infection Epidemic in the Netherlands: a Multi-Parameter Evidence Synthesis Approach." *The Annals of Applied Statistics*, **5**(4), 2359–2384. ISSN 1932-6157. URL https://dx.doi.org/10.1214/11-AOAS488.

Daniels M (1999). "A prior for the variance in hierarchical models." *The Canadian Journal of Statistics*, **27**, 567–578. ISSN 03195724. doi:doi:10.2307/3316112. URL https://onlinelibrary.wiley.com/doi/10.2307/3316112/abstract.

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

Denwood MJ (2010). *A Quantitative Approach to Improving the Analysis of Faecal Worm Egg Count Data*. Doctoral thesis, University of Glasgow. URL https://theses.gla.ac.uk/1837/.

Denwood MJ, Reid SWJ, Love S, Nielsen MK, Matthews L, McKendrick IJ, Innocent GT (2010). "Comparison of Three Alternative Methods for Analysis of Equine Faecal egg Count Reduction Test Data." *Preventive Veterinary Medicine*, **93**(4), 316–23. ISSN 1873-1716. URL https://dx.doi.org/10.1016/j.prevetmed.2009.11.009.

Denwood MJ, Stear MJ, Matthews L, Reid SWJ, Toft N, Innocent GT (2008). "The Distribution of the Pathogenic Nematode *Nematodirus battus* in Lambs is Zero-Inflated." *Parasitology*, **135**(10), 1225–1235. ISSN 1469-8161 (Electronic). URL https://dx.doi.org/10.1017/S0031182008004708.

DuMouchel W (1994). "Hierarchical Bayes Linear Models for Meta-Analysis." *Technical Report 27*, National Institute of Statistical Sciences. URL https://www.niss.org/sites/default/files/technicalreports/tr27.pdf.

Gelman A (2006). "Prior Distributions for Variance Parameters in Hierarchical Models." *Bayesian Analysis*, **1**(3), 515–533.

Gelman A, Rubin DB (1992). "Inference from Iterative Simulation Using Multiple Sequences." *Statistical Science*, **7**(4), 457–472. URL https://www.jstor.org/stable/2246093.

Gilks WR, Richardson S, Spiegelhalter DJ (1998). *Markov Chain Monte Carlo in Practice*. Chapman and Hall, Boca Raton, Fla. ISBN 0412055511. URL https://search.catalog.loc.gov/search?option=lccn&query=98033429.

Hastings WK (1970). "Monte Carlo Sampling Methods Using Markov Chains and Their Applications." *Biometrika*, **57**(1), 97–109. URL https://dx.doi.org/10.1093/biomet/57.1.97.

Kass RE, Carlin BP, Gelman A, Neal RM (1998). "Markov Chain Monte Carlo in Practice: a Roundtable Discussion." *The American Statistician*, **52**(2), 93–100.

Lele SR, Dennis B (2009). "Bayesian Methods for Hierarchical Models: are Ecologists Making a Faustian Bargain?" *Ecological Applications : a Publication of the Ecological Society of America*, **19**(3), 581–4. ISSN 1051-0761. doi:doi:10.1890/08-0549.1.

Lunn D, Jackson C, Best N, Thomas A, Spiegelhalter D (2012). *The BUGS book: A practical introduction to Bayesian analysis*. CRC press.

Lunn DJ, Thomas A, Best N, Spiegelhalter D (2000). "**WinBUGS** - a Bayesian Modelling Framework: Concepts, Structure, and Extensibility." *Statistics and Computing*, **10**(4), 325–337. ISSN 0960-3174. URL https://dx.doi.org/10.1023/A:1008929526011.

Phillips G, Tam CC, Conti S, Rodrigues LC, Brown D, Iturriza-Gomara M, Gray J, Lopman B (2010). "Community Incidence of Norovirus-Associated Infectious Intestinal Disease in England: Improved Estimates Using Viral Load for Norovirus Diagnosis." *American Journal of Epidemiology*, **171**(9), 1014–22. ISSN 1476-6256. URL https://dx.doi.org/10.1093/aje/kwq021.

Plummer M (2003). "JAGS : A Program for Analysis of Bayesian Graphical Models Using Gibbs Sampling." In *Proceedings of the 3rd International Workshop on Distributed Statistical Computing (DSC 2003)*, pp. March 20–22,Vienna, Austria. ISSN 1609–395X. ISSN 1609395X.

Plummer M (2008). "Penalized loss functions for Bayesian model comparison." *Biostatistics*, **9**, 523–539. ISSN 14654644. doi:10.1093/biostatistics/kxm049.

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

Plummer M, Best N, Cowles K, Vines K (2006). "**CODA**: Convergence Diagnosis and Output Analysis for MCMC." *R News*, **6**(1), 7–11. URL https://cran.r-project.org/package=coda.

Polson NG, Scott JG (2011). "On the Half-Cauchy Prior for a Global Scale Parameter." *Cornell University Library: arXiv.org*. URL https://arxiv.org/abs/1104.4937.

R Core Team (2015). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL https://www.r-project.org.

Shaw DJ, Grenfell BT, Dobson AP (1998). "Patterns of Macroparasite Aggregation in Wildlife Host Populations." *Parasitology*, **117**, 597–610. ISSN 0031-1820.

Spiegelhalter DJ, Best NG, Carlin BP, van der Linde A (2002). "Bayesian Measures of Model Complexity and Fit." *Journal of the Royal Statistical Society B*, **64**(4), 583–639. ISSN 13697412. URL https://www.jstor.org/stable/3088806.

Tierney L, Rossini AJ, Li N, Sevcikova H (2013). ***snow**: Simple Network of Workstations*. R package version 0.3-13, URL https://cran.r-project.org/package=snow.

Toft N, Innocent GT, Gettinby G, Reid SWJ (2007). "Assessing the Convergence of Markov Chain Monte Carlo Methods: an Example from Evaluation of Diagnostic Tests in Absence of a Gold Standard." *Preventive Veterinary Medicine*, **79**(2-4), 244–256. ISSN 0167-5877. URL https://dx.doi.org/10.1016/j.prevetmed.2007.01.003.

Van Hauwermeiren M, Vose D (2009). *A Compendium of Distributions*. Vose Software, Ghent, Belgium.

Wabersich D, Vandekerckhove J (2014). "Extending JAGS: a tutorial on adding custom distributions to JAGS (with a diffusion model example)." *Behavior research methods*, **46**(1), 15–28. ISSN 1554-3528. doi:doi:10.3758/s13428-013-0369-3.

Wilson K, Grenfell BT (1997). "Generalized Linear Modelling for Parasitologists." *Parasitology Today*, **13**(1), 33–38. ISSN 0169-4758. URL https://dx.doi.org/10.1016/S0169-4758(96)40009-6.

Wilson K, Grenfell BT, Shaw DJ (1996). "Analysis of Aggregated Parasite Distributions: a Comparison of Methods." *Functional Ecology*, **10**, 592–601.

Yin Z, Conti S, Desai S, Stafford M, Slater W, Gill ON, Simms I (2013). "The Geographic Relationship Between Sexual Health Deprivation and the Index of Multiple Deprivation 2010: a Comparison of two Indices." *Sexual Health*, **10**(2), 102–11. ISSN 1448-5028. URL https://dx.doi.org/10.1071/SH12057.

A. Formulation of the negative binomial as a gamma-Poisson

The compound probability mass function of a Poisson distribution (with mean λ) integrated over a gamma distribution (with shape and scale parameters α and β respectively) is given in Equation 1.

$$f(x;\alpha,\beta) = \int_0^\infty \frac{\lambda^x}{x!}e^{-\lambda} \,.\, \beta^\alpha \frac{1}{\Gamma(\alpha)}\lambda^{\alpha-1}e^{-\beta\lambda}\,\mathrm{d}\lambda \tag{1}$$

Substituting $\alpha = r$ and $\beta = \frac{1-p}{p}$ into Equation 1 gives Equation 2, which can be re-written and simplified to Equation 4.

$$f(x;r,p) = \int_0^\infty \frac{\lambda^x}{x!}e^{-\lambda} \,.\, \left(\frac{1-p}{p}\right)^r \frac{1}{\Gamma(r)}\lambda^{r-1}e^{-\left(\frac{1-p}{p}\right)\lambda}\,\mathrm{d}\lambda \tag{2}$$

$$= \frac{(1-p)^r}{x!\; p^r\; \Gamma(r)} \int_0^\infty \lambda^{x+r-1}e^{-\lambda}e^{-\frac{(1-p)\lambda}{p}}\,\mathrm{d}\lambda \tag{3}$$

$$= \frac{(1-p)^r}{x!\; p^r\; \Gamma(r)} \int_0^\infty \lambda^{x+r-1}e^{-\frac{\lambda}{p}}\,\mathrm{d}\lambda \tag{4}$$

Substituting the gamma function $\frac{\Gamma(b+1)}{a^{b+1}} = \int_0^\infty t^b e^{-at}\mathrm{d}t$ for $a = \frac{1}{p}$, $b = x + r - 1$ and $t = \lambda$ into Equation 4 gives Equation 5.

$$f(x;r,p) = \frac{(1-p)^r}{x!\; p^r\; \Gamma(r)} \frac{\Gamma(x+r-1+1)}{\left(\frac{1}{p}\right)^{x+r-1+1}} \tag{5}$$

$$= \frac{(1-p)^r}{x!\; p^r\; \Gamma(r)}\, \Gamma(x+r)\, p^{x+r} \tag{6}$$

$$= \frac{\Gamma(x+r)}{x!\; \Gamma(r)}\, p^x (1-p)^r \tag{7}$$

Equation 7 is the probability mass function of the negative binomial distribution defining the number of successes x before r failures with success probability p, which is therefore exactly equivalent to a gamma-Poisson compound distribution with mean $\frac{\alpha}{\beta} = \frac{pr}{1-p}$ and shape $\alpha = r$.

Affiliation:

Matthew J. Denwood
Department of Large Animal Sciences
Section for Animal Welfare and Disease Control
Faculty of Health and Medical Sciences
University of Copenhagen
Denmark
E-mail: md@sund.ku.dk
URL: https://researchprofiles.ku.dk/en/persons/matt-denwood/