R2WinBUGS: A Package for Running WinBUGS from R

Sibylle Sturtz*  
Fachbereich Statistik  
Universität Dortmund  
Germany

Uwe Ligges†  
Fachbereich Statistik  
Universität Dortmund  
Germany

Andrew Gelman‡  
Department of Statistics  
Columbia University  
USA

Abstract

The R2WinBUGS package provides convenient functions to call WinBUGS from R. It automatically writes the data and scripts in a format readable by WinBUGS for processing in batch mode, which is possible since version 1.4. After the WinBUGS process has finished, it is possible either to read the resulting data into R by the package itself—which gives a compact graphical summary of inference and convergence diagnostics—or to use the facilities of the coda package for further analyses of the output. Examples are given to demonstrate the usage of this package.

Keywords: R, WinBUGS, interface, MCMC.


1. Introduction

The usage of Markov chain Monte Carlo (MCMC) methods became very popular within the last decade. WinBUGS (Bayesian inference Using Gibbs Sampling, Spiegelhalter, Thomas, Best, and Lunn 2003) is a popular software for analyzing complex statistical models using MCMC methods. This software uses Gibbs sampling (Geman and Geman 1984; Gelfand and Smith 1990; Casella and George 1992) and the Metropolis algorithm (Metropolis, Rosenbluth, Rosenbluth, Teller, and Teller 1953) to generate a Markov chain by sampling from full conditional distributions. The WinBUGS software is available for free at http://www.mrc-bsu.cam.ac.uk/bugs/. An introduction to MCMC methods is given in Gilks, Richardson, and Spiegelhalter (1996).

Using WinBUGS, the user must specify the model to run, and to load data and initial values for a specified number of Markov chains. Then it is possible to run the Markov chain(s) and to save the results for the parameters the user is interested in. Summary statistics of these data, convergence diagnostics, kernel estimates etc. are available as well. Nevertheless, some users of this software might be interested in saving the output and reading it into R (R Development Core Team 2004) for further analyses. WinBUGS 1.4 comes with the ability to run the software in batch mode using scripts.

The R2WinBUGS package makes use of this feature and provides the tools to call WinBUGS directly after data manipulation in R. Furthermore, it is possible to work with the results after importing them back into R again, for example to create posterior predictive simulations or, more generally, graphical displays of data and posterior simulations (Gelman 2004). Embedding in R can also be useful for frequently changed data or processing a bunch of data sets, because it is much more convenient to use some R functions (possibly within a loop) rather than using “copy & paste” to update data in WinBUGS each time; however difficulties have been encountered in this

*sturtz@statistik.uni-dortmund.de
†ligges@statistik.uni-dortmund.de
‡gelman@stat.columbia.edu
area because both R and WinBUGS can lock up RAM in the Windows operating system. R is a “language for data analysis and graphics” and an open source and freely available statistical software package implementing that language, see http://www.R-project.org/. Historically, R is an implementation of the award-winning S language and system (Becker and Chambers 1984; Becker, Chambers, and Wilks 1988; Chambers and Hastie 1992; Chambers 1998). R and R2WinBUGS are available from CRAN (Comprehensive R Archive Network), i.e., http://CRAN.R-project.org or one of its mirrors. R2WinBUGS could be ported to the commercial S implementation S-Plus. Minor adaptions would be needed since S-PLUS lacks some of R’s functions and capabilities. If an internet connection is available, R2WinBUGS can be installed by typing `install.packages("R2WinBUGS")` at the R command prompt. Do not forget to load the package with `library("R2WinBUGS")`. The package `coda` by Plummer, Best, Cowles, and Vines (2004) is very useful for the analysis of WinBUGS’ output, the reader might want to install this package as well. The CRAN package `boa` (Bayesian Output Analysis Program) by Smith (2004) has similar aims. JAGS (Just Another Gibbs Sampler) by Plummer (2003) is a program for analysis of Bayesian hierarchical models using Gibbs sampling that aims for the same functionality as classic BUGS. JAGS is developed to work closely together with R and the `coda` package.

A new and completely revised version of WinBUGS called OpenBUGS (Spiegelhalter, Thomas, Best, and Lunn 2004) was lately published under the terms of the GPL. OpenBUGS is also expected to run under Linux. It provides a much more flexible API on which “BRugs” is based including a dynamic link library, incorporating a component loader that allows R to make use of OpenBUGS components. OpenBUGS is still in development and suffers frequent crashes. As OpenBUGS becomes more reliable, it is planned to merge “BRugs” and R2WinBUGS into one R package.

For other packages and projects on spatial statistics related to R, follow the link to “R spatial projects” at CRAN.

In this paper, we give two examples, involving educational testing experiments in schools (cf. Section 2.1), and incidence of childhood leukaemia depending on benzene emissions (cf. Section 2.2). Details on the functions of R2WinBUGS are given in Section 3. These functions automatically write the data and a script in a format readable by WinBUGS for processing in batch mode, and call WinBUGS from R. After the WinBUGS process has finished, it is possible either to read the resulting data into R by the package itself or to use the facilities of the `coda` package for further analyses of the output. In Section 4, we demonstrate how to apply the functions provided by R2WinBUGS on the examples’ data, and how to analyze the output both with package `coda` and with R2WinBUGS’s methods to `plot()` and `print()` the output.

### 2. Examples

In this Section, we introduce two examples which will be continued in Section 4.

#### 2.1. Schools data

The Scholastic Aptitude Test (SAT) measures the aptitude of high-schoolers in order to help colleges to make admissions decisions. It is divided into two parts, verbal (SAT-V) and mathematical (SAT-M). Our data comes from the SAT-V (Scholastic Aptitude Test-Verbal) on eight different high schools, from an experiment conducted in the late 1970s. SAT-V is a standard multiple choice test administered by the Educational Testing Service. This Service was interested in the effects of coaching programs for each of the selected schools. The study included coached and uncoached pupils, about sixty in each of the eight different schools; see Rubin (1981). All of them had already taken the PSAT (Preliminary SAT) which results were used as covariates. For each school, the estimated treatment effect and the standard error of the effect estimate are given. These are calculated by an analysis of covariance adjustment appropriate for a completely randomized experiment (Rubin 1981). This example was analyzed using a hierarchical normal model in Rubin (1981) and Gelman, Carlin, Stern, and Rubin (2003, Section 5.5).
2.2. Leukaemia registration data

Spatial data usually arises on different, non-nesting spatial scales. One example is childhood leukaemia registration data analyzed by Best, Cockings, Bennett, Wakefield, and Elliott (2001) using ecologic regression. Data are given for Greater London bounded by the M25 orbital motorway. The data are not available as an example in R2WinBUGS but we use the example here to illustrate alternative calls to the `bugs()` function and output analysis using the `coda` package.

The observed number of leukaemia cases among children under 15 years old is given at ward level. Census wards are administrative areas containing approximately 5000 to 10 000 people. Central London is divided into 873 wards. The number of incident cases of leukaemia in children is available from 1985 until 1996 from the Office of National Statistics and the Thames Cancer Registry. A plot of these numbers is given in Figure 1.

Additionally, the number of expected cases (cf. Fig. 2) is calculated on the same resolution using population numbers for different age-sex-strata and the national leukaemia rate for the corresponding strata, for details see Best et al. (2001).

It is assumed that benzene emissions have an effect on the incidence rate of leukaemia. Benzene emission rates are available in tonnes per year from an atmospheric emissions inventory for London.
Figure 3: Benzene emissions in tonnes per year

(Buckingham, Clewley, Hutchinson, Sadler, and Shah 1997) produced by the London Research Centre. They are provided at 1km × 1km grid cells, giving 2132 grid cells in total. Their spatial distribution is shown in Figure 3.

For further details on the data see Best et al. (2001).

We model these data by Poisson-Gamma models introduced by Best, Ickstadt, and Wolpert (2000) using WinBUGS. A linking matrix containing information which grid cell belongs to which ward and to which amount is required. This matrix is calculated using R. Unfortunately, WinBUGS does not support a list format such as directly produced by R. Therefore, the data must be provided as a matrix with 2132 rows and 873 columns (or vice versa). Most of the entries of this matrix are zeroes, but using dump() to export it from R yields in a file size of 14.2 MB. Unfortunately, opening a file of such size really slows WinBUGS down, and it was not even possible on some of our PCs. Importing data written by our R2WinBUGS package does not make any problems using the batch mode, probably due to memory management issues in WinBUGS.

3. Implementation

The implementation of the R2WinBUGS package is straightforward. The “main” function bugs() is intended to be called by the user. In principle, it is a wrapper for several other functions called therein step by step as follows:

1. bugs.data.inits() writes the data files ‘data.txt’, and ‘inits1.txt’, ‘inits2.txt’, ... into the working directory. These files will be used by WinBUGS during batch processing.

   In particular, input for WinBUGS must not exceed a certain number of digits. Moreover, it needs an E instead of an e in scientific notation. Scientific notation is particularly desirable because of the “number of digits” limitation. The default (digits = 5) is to, e.g., reformat the number 123456.789 to 1.23457E+05.

2. bugs.script() writes the file ‘script.txt’ that is used by WinBUGS for batch processing.

3. bugs.run() updates the lengths of the adaptive phases in the WinBUGS registry (using a function bugs.update.settings()), calls WinBUGS, and runs it in batch mode with ‘script.txt’.

4. bugs.sims() is only called if the argument codaPkg has been set to FALSE (the default). Otherwise bugs() returns the filenames of stored data. These can, for example, be imported
by package *coda* (see the example in Section 4.2, page 10), which provides functions for
convergence diagnostics, calculation of Monte Carlo estimates, trace plots, and so forth.

The function *bugs.sims()* reads simulations from *WinBUGS* into R (not necessarily called
by *bugs()* itself), formats them, monitors convergence, performs convergence checks, and
computes medians and quantiles. It also prepares the output for *bugs()* itself.

These functions are not intended to be called by the user directly. Arguments are passed from
*bugs()* to the other functions, if appropriate. A shortened help file of *bugs()* listing all arguments
is given in Appendix A; for the full version type *?bugs* in R after having installed and loaded the
package *R2WinBUGS* (see Section 1).

As known from *WinBUGS*, one must specify the data in form of a list, with list names equal to
the names of data in the corresponding *WinBUGS* model. Alternatively, it is possible to specify
a vector or list of names (of mode character). In that case objects of that names are looked
for in the environment in which *bugs()* has been called (usually that is the user’s Workspace,
*.GlobalEnv*). If data have already been written in a file called ‘data.txt’ to the working directory,
it is possible to specify *data = "data.txt"*. One will usually want to supply initial values. This
can be done either in the form of a function *inits()* that creates these values, so that different
chains can be automatically initialized at different points (see Section 4.1), or by specifying them
directly (see Section 4.2). If *inits()* is not specified, *bugs()* just uses the starting values created
by *WinBUGS*; but in practice *WinBUGS* can crash when reasonable initial values are not specified,
and so we recommend constructing a simple *inits()* function to simulate reasonable starting
points (Gelman et al. 2003, Section C.2). It is also necessary to specify which parameters should
be saved for monitoring by specifying *parameters.to.save*.

The user might also want to change the defaults for the length of the burn-in (*n.burnin*, which
defaults to half the length of the chain) period for every MCMC run and the number of iterations
(*n.iter*, default value 3) that are used to calculate Monte Carlo estimates. The specification of a
thinning parameter (*n.thin*) is possible as well; this is useful when the number of parameters is
large, to keep the saved output to a reasonably-sized R object. In the default setting, the chains
are thinned enough so that approximately 1000 simulation draws are saved.

By setting the argument *debug = TRUE*, *WinBUGS* remains open after the run. This way it is
possible to find errors in the code or the data structure, or even to work with that software as in
a usual run.

It is possible to run one or more Markov chains. The number of chains (*n.chains*) must be specified
together with the chains’ initial values (*inits*). If more than one Markov chain is requested and
*codaPkg* is set to *FALSE*, the convergence diagnostic $\hat{R}$ (Brooks and Gelman 1998) is calculated by
*bugs.sims()* for each of the saved parameters.

Since the communication between *WinBUGS* and R is based on files, rather huge files will be saved
in the working directory by the *bugs()* call, either files to be read in by *bugs()* itself, or by
the *coda* package. The user might want to delete those files after the desired contents has been
imported into R, and save those objects, e.g., as compressed R data files.

The function *bugs()* returns a rather complex object of class *bugs*, if called with argument
*codaPkg = FALSE*. In order to look at the structure of such an object, type *str(objectname)*. For
convenience, *R2WinBUGS* provides methods corresponding to class *bugs* for the generic functions
*print()* and *plot()*.

So that user will not be overwhelmed with information; summaries of the output are provided by
the *print()* method. That is, some parameters of the *bugs()* call are summarized, and mean,
standard deviation, several quantiles of the parameters and convergence diagnostics based on
Gelman and Rubin (1992) are printed. See the example in Section 4.1, page 7, for a typical
output. As with Spiegelhalter, Best, Carlin, and van der Linde (2002), the DIC computed by
*bugs.sims()* is defined as the posterior mean of the deviance plus $p_D$, the estimated effective
number of parameters in the posterior distribution. We define $p_D$ as half the posterior variance
of the deviance and estimate it as half the average of the within-chain variances of the deviance.¹

¹In contrast, Spiegelhalter et al. (2002), and *WinBUGS*, define $p_D$ as the posterior mean of the deviance
The `plot()` function for objects of class `bugs` provides information condensed in some plots conveniently arranged within the same graphics device. For an example, see Figure 4 in Section 4.1. It is intended to adapt this function to work with MCMC output in general, even if obtained from software other than WinBUGS.

### 4. Examples continued

The Examples introduced in Section 4 are continued in this Section. We apply the functions provided by R2WinBUGS to the examples’ data and analyze the output.

#### 4.1. Schools data

Schools example data (see Section 2.1) are available with the R2WinBUGS package:

```r
> data(schools)
> schools
       school estimate   sd
   1      A    28.39 14.9
   2      B     7.94 10.2
   3      C   -2.75 16.3
   4      D    6.82 11.0
   5      E   -0.64  9.4
   6      F     0.63 11.4
   7      G    18.01 10.4
   8      H    12.16 17.6
```

For modeling these data, we use a hierarchical model as proposed by Gelman et al. (2003, Section 5.5). We assume a normal distribution for the observed estimate for each school with mean `theta` and inverse-variance `tau.y`. The inverse-variance is given as `1/sigma.y^2` and its prior distribution is uniform on `(0,1000)`. For the mean `theta`, we employ another normal distribution with mean `mu.theta` and inverse-variance `tau.theta`. For their prior distributions, see the following WinBUGS code:

```winbugs
model {
    for (j in 1:J)
    {
        y[j] ~ dnorm (theta[j], tau.y[j])
        theta[j] ~ dnorm (mu.theta, tau.theta)
        tau.y[j] <- pow(sigma.y[j], -2)
    }
    mu.theta ~ dnorm (0.0, 1.0E-6)
    tau.theta <- pow(sigma.theta, -2)
    sigma.theta ~ dunif (0, 1000)
}
```

This model must be stored in a separate file, e.g. ‘schools.bug’², in an appropriate directory, say `c:/schools/`. In R the user must prepare the data inputs the `bugs()` function needs. This can be a list containing the name of each data vector, e.g.

---

²Emacs Speaks Statistics (ESS) by Rossini, Heiberger, Sparapani, Mächler, and Hornik (2004), a package available with GNU Emacs (Stallmann 1999), recognizes and properly formats Bugs model files that have the .bug extension.
> J <- nrow(schools)
> y <- schools$estimate
> sigma.y <- schools$sd
> data <- list("J", "y", "sigma.y")

Using these data and the model file, we can run an MCMC simulation to get estimates for \( \theta \), \( \mu_\theta \) and \( \sigma_\theta \). Before running, the user must decide how many chains to be run (\( n\text{.chain} = 3 \)) for how many iterations (\( n\text{.iter} = 1000 \)). If the length of burn-in is not specified, \( n\text{.burnin} = \lceil n\text{.iter}/2 \rceil \) is used, that is, 500 in this example. Additionally, the user must specify initial values for the chains, for example by writing a function. This can be done by

```r
> inits <- function()
+   list(theta = rnorm(J, 0, 100), mu.theta = rnorm(1, 0, 100),
+         sigma.theta = runif(1, 0, 100))
+ }
```

Now, the user can start the MCMC simulation by typing

```r
> schools.sim <- bugs(data, inits, model.file = "c:/schools/schools.bug",
+                      parameters = c("theta", "mu.theta", "sigma.theta"),
+                      n.chains = 3, n.iter = 1000,
+                      bugs.directory = "c:/Program Files/WinBUGS14/")
```

in R. The argument `bugs.directory` must point to the directory where WinBUGS has been installed. For other available arguments, see Appendix A.

The results in objects `schools.sim` can conveniently be printed by `print(schools.sim)`. The generic function `print()` calls the print method for an object of class `bugs` provided by R2WinBUGS. For this example, you will get something like

```r
> print(schools.sim)
Inference for Bugs model at "c:/schools/schools.bug"
3 chains, each with 1000 iterations (first 500 discarded)
n.sims = 1500 iterations saved

mean sd 2.5%  25% 50% 75% 97.5% Rhat n.eff
theta[1] 11.1  9.1  -3.0  5.0 10.0 16.0 31.8 1.1  39
theta[2]  7.6  6.6  -4.7  3.3  7.8 11.6 21.1 1.1  42
theta[3]  5.7  8.4  -12.5  0.6  6.1 10.8 21.8 1.0 150
theta[4]  7.1  7.0  -6.6  2.7  7.2 11.5 21.0 1.1  42
theta[5]  5.1  6.8  -9.5  0.7  5.2  9.7 18.1 1.0  83
theta[6]  5.7  7.3  -9.7  1.0  6.2 10.2 20.0 1.0  56
theta[7] 10.4  7.3  -2.1  5.3  9.8 15.3 25.5 1.1  27
theta[8]  8.3  8.4  -6.6  2.8  8.1 12.7 26.2 1.0  64
mu.theta  7.6  5.9  -3.0  3.7  8.0 11.0 19.5 1.1  35
sigma.theta 6.7  5.6  0.3  2.8  5.1  9.2 21.2 1.1  46
deviance 60.8  2.5  57.0  59.1 60.2 62.1 66.6 1.0 170
pD = 3 and DIC = 63.8 (using the rule, pD = var(deviance)/2)

For each parameter, n.eff is a crude measure of effective sample size, and Rhat is the potential scale reduction factor (at convergence, Rhat=1). DIC is an estimate of expected predictive error (lower deviance is better).

Additionally, the user can generate a plot of the results by typing `plot(schools.sim)`. The resulting plot is given in Figure 4. In this plot, the left column shows a quick summary of inference and convergence (\( \hat{R} \) is close to 1.0 for all parameters, indicating good mixing of the three chains and thus approximate convergence); and the right column shows inferences for each set of parameters. As can be seen in the right column, R2WinBUGS uses the parameter names in WinBUGS to structure the output into scalar, vector, and arrays of parameters, in addition to storing the parameters as a long vector.
For the interpretation of these results see Gelman et al. (2003, Section 5.5).

### 4.2. Leukaemia registration data

The leukaemia registration data (see Section 2.2) are used to show data modeling and output reading into R using the `coda` package. A simple model for these data looks as follows:

```r
model{
  beta.0 ~ dgamma(a.0, tau.0)
  beta.benz ~ dgamma(a.benz, tau.benz)
  a.0 <- 0.575
  tau.0 <- a.0*2
  a.benz <- 0.575
  tau.benz <- a.benz*2
```
for (i in 1:I)
{
  count[i] ~ dpois(lambda[i])
  lambda[i] <- p[i]*expect[i]
  for (j in 1:J)
  {
    prop[j,i] <- gamma[j,i]*(benz[j] - benzbar)
  }
  p[i]<- beta.0 + beta.benz*sum(prop[,i])
}

Here count denotes the number of observed incidences of childhood leukaemia in ward i. These are assumed to be Poisson distributed with mean lambda depending on the number of expected cases expect in ward i and an area-specific risk rate p. For calculation of this area specific risk rate we use an intercept beta.0 and a term depending on the weighted sum of benzene emissions benz in each grid cell j. The weights are chosen proportional to the amount of area that ward i and grid cell j have in common.

In R we can define all these data and then initialize the model. The data needed for this example are

benzbar: arithmetic mean of all benzene values,
benz: a vector containing benzene emissions of all 2132 grid cells,
expect: expected number of cases of childhood leukaemia in each of the 873 wards,
count: observed number of childhood leukaemia in these wards,
gamma: a 2132 × 873 matrix containing the amount of area each grid cell and each ward have in common,
J: total number of grid cells, i.e. 2132, and
I: total number of ward cells, i.e. 873.

The parameters we want to store are regression coefficients beta.0 and beta.benz as well as p, the area specific relative risk compared to the reference rate. This reference rate was used to calculate the expected number of cases in each ward.

Since we want to use the coda package for reading the data into WinBUGS, we specify codaPkg = TRUE in the bugs() call:

```R
> data <- list(benzbar = mean(benz), benz = benz, expect = expect, count = count, gamma = gamma, J = J, I = I)
> parameters <- c("beta.0", "beta.benz", "p")
> inits1 <- list(beta.0 = 1, beta.benz = 1)
> inits2 <- list(beta.0 = 0.5, beta.benz = 0.5)
> inits <- list(inits1, inits2)
> model <- bugs(data, inits, parameters, model.file = "c:/model.bug", + n.chains = 2, n.iter = 8000, n.burnin = 5000, n.thin = 1, + codaPkg = TRUE, bugs.directory = "c:/Program Files/WinBUGS14/")
```
Starting with, e.g.,

```r
> library("coda")
> codaobject <- read.bugs(model)
> plot(codaobject)
```

it is now possible to use the `coda` package for output analyses.

Acknowledgments

The work of Uwe Ligges has been supported by the Deutsche Forschungsgemeinschaft, Sonderforschungsbereich 475. The work of Andrew Gelman has been supported by the U.S. National Science Foundation.

References


A. Help page for the function `bugs()`

This help page has been shortened.

```
bugs

Run WinBUGS from R

Description

The `bugs` function takes data and starting values as input. It automatically writes a WinBUGS script, calls the model, and saves the simulations for easy access in R.

Usage

```
bugs(data, inits, parameters.to.save, model.file = "model.txt",          
    n.chains = 3, n.iter = 2000, n.burnin = floor(n.iter/2),          
    n.thin = max(1, floor(n.chains * (n.iter - n.burnin)/1000)),       
    bin = (n.iter - n.burnin) / n.thin,                               
    debug = FALSE, DIC = TRUE, digits = 5, codaPkg = FALSE,           
    bugs.directory = "c:/Program Files/WinBUGS14/",                    
    working.directory = NULL)
```

Arguments

data: either a named list (names corresponding to variable names in the `model.file`) of the data for the WinBUGS model, or a vector or list of the names of the data objects used by the model. If `data = "data.txt"`, it is assumed that data have already been written to the working directory in a file called `data.txt`, e.g. by the function `bugs.data`.

inits: a list with `n.chains` elements; each element of the list is itself a list of starting values for the WinBUGS model, or a function creating (possibly random) initial values. Alternatively, if `inits = NULL`, initial values are generated by WinBUGS.

parameters.to.save: character vector of the names of the parameters to save which should be monitored.

model.file: file containing the model written in WinBUGS code. The extension can be either `.bug` or `.txt`. If `.bug`, a copy of the file with extension `.txt` will be created in the `bugs()` call and removed afterwards. Note that similarly named `.txt` files will be overwritten.

n.chains: number of Markov chains (default: 3)

n.iter: number of total iterations per chain (including burn in; default: 2000)

n.burnin: length of burn in, i.e. number of iterations to discard at the beginning. Default is `n.iter/2`, that is, discarding the first half of the simulations.

n.thin: thinning rate. Must be a positive integer. Set `n.thin > 1` to save memory and computation time if `n.iter` is large. Default is `max(1, floor(n.chains * (n.iter - n.burnin) / 1000))` which will only thin if there are at least 2000 simulations.

bin: number of iterations between saving of results (i.e. the coda files are saved after each bin iterations); default is to save only at the end.

debug: if `FALSE` (default), WinBUGS is closed automatically when the script has finished running, otherwise WinBUGS remains open for further investigation.

DIC: logical; if `TRUE` (default), compute deviance, pD, and DIC.

digits: number of significant digits used for WinBUGS input, see `formatC`.

codaPkg: logical; if `FALSE` (default) a `bugs` object is returned, if `TRUE` file names of WinBUGS output are returned for easy access by the `coda` package.
bugs.directory directory that contains the WinBUGS executable
working.directory sets working directory during execution of this function; WinBUGS’ in- and output will be stored in this directory; if NULL, the current working directory is chosen.

Value

If codaPkg = TRUE the returned values are the names (without file extension) of files written by WinBUGS containing the Markov Chain Monte Carlo output in the CODA format and corresponding index files. This is useful for direct access with `read.bugs` from package ‘coda’.

If codaPkg = FALSE, the following values are returned:

- `n.chains` see Section ‘Arguments’
- `n.iter` see Section ‘Arguments’
- `n.burnin` see Section ‘Arguments’
- `n.thin` see Section ‘Arguments’
- `n.keep` number of iterations kept per chain (equal to `(n.iter-n.burnin) / n.thin`)
- `n.sims` number of posterior simulations (equal to `n.chains * n.keep`)
- `sims.array` 3-way array of simulation output, with dimensions `n.keep`, `n.chains`, and length of combined parameter vector
- `sims.list` list of simulated parameters:
  - for each scalar parameter, a vector of length `n.sims`
  - for each vector parameter, a 2-way array of simulations,
  - for each matrix parameter, a 3-way array of simulations, etc.
- `sims.matrix` matrix of simulation output, with `n.chains * n.keep` rows and one column for each element of each saved parameter (for convenience, the `n.keep * n.chains` simulations in `sims.array` and `sims.list` have been randomly permuted)
- `summary` summary statistics and convergence information for each element of each saved parameter.
- `mean` a list of the estimated parameter means
- `sd` a list of the estimated parameter standard deviations
- `median` a list of the estimated parameter medians
- `root.short` names of argument `parameters.to.save` and “deviance”
- `long.short` indexes; programming stuff
- `dimension.short` dimension of `indexes.short`
- `indexes.short` indexes of `root.short`
- `last.values` list of simulations from the most recent iteration; they can be used as starting points if you wish to run WinBUGS for further iterations
- `pD` \( \text{var(deviance)}/2 \), an estimate of the effective number of parameters (the variance is computed as the average of the within-chain variances, which gives a more reasonable estimate when convergence has not been reached)
- `DIC` \( \text{mean(deviance)} + \text{pD} \)