

Package: R2OpenBUGS (via r-universe)

October 24, 2024

Title Running OpenBUGS from R

Date 2017-2-20

Version 3.2-3.2.1

Author originally written as R2WinBUGS by Andrew Gelman

<gelman@stat.columbia.edu>; changes and packaged by Sibylle Sturtz <sturtz@statistik.tu-dortmund.de> and Uwe Ligges <ligges@statistik.tu-dortmund.de>. With considerable contributions by Gregor Gorjanc <gregor.gorjanc@bfro.uni-lj.si> and Jouni Kerman <kerman@stat.columbia.edu>. Adapted to R2OpenBUGS from R2WinBUGS by Neal Thomas.

Description Using this package, it is possible to call a BUGS model, summarize inferences and convergence in a table and graph, and save the simulations in arrays for easy access in R.

Depends R (>= 2.13.0)

Imports coda (>= 0.11-0), boot

SystemRequirements OpenBUGS (>= 3.2.2)

Maintainer Neal Thomas <snthomas99@gmail.com>

License GPL-2

NeedsCompilation no

Date/Publication 2020-04-02 17:31:15 UTC

Repository <https://snthomas99.r-universe.dev>

RemoteUrl <https://github.com/cran/R2OpenBUGS>

RemoteRef HEAD

RemoteSha 21402f0d531767b04db24ba94021f3d5d0507e6a

Contents

R2OpenBUGS-package	2
as.bugs.array	3
attach.all	4

bugs	5
bugs.data	11
bugs.inits	11
bugs.log	12
plot.bugs	13
print.bugs	13
read.bugs	14
schools	14
validateInstallOpenBUGS	15
write.model	16
Index	18

R2OpenBUGS-package *Running OpenBUGS from R*

Description

R2OpenBUGS Call a **BUGS** model, summarize inferences and convergence in a table and graph, and save the simulations in arrays for easy access in **R**. The main command is [bugs](#).

Details

The following are sources of information on **R2OpenBUGS** package:

DESCRIPTION file	<code>library(help="R2openBUGS")</code>
This file	<code>package?R2openBUGS</code>
Vignette	<code>vignette("R2openBUGS")</code>
Some help files	bugs write.model print.bugs plot.bugs
News	<code>file.show(system.file("NEWS", package="R2openBUGS"))</code>

as.bugs.array *Convert to bugs object*

Description

Function converting results from Markov chain simulations, that might not be from BUGS, to bugs object. Used mainly to display results with [plot.bugs](#).

Usage

```
as.bugs.array(sims.array, model.file=NULL, program=NULL,  
             DIC=FALSE, DICOutput=NULL, n.iter=NULL, n.burnin=0, n.thin=1)
```

Arguments

sims.array	3-way array of simulation output, with dimensions n.keep, n.chains, and length of combined parameter vector.
model.file	file containing the model written in OpenBUGS code
program	the program used
DIC	logical; whether DIC should be calculated, see also argument DICOutput and details
DICOutput	DIC value
n.iter	number of total iterations per chain used for generating sims.array
n.burnin	length of burn in, i.e. number of iterations to discarded at the beginning for generating sims.array
n.thin	thinning rate, a positive integer, used for generating sims.array

Details

This function takes a 3-way array of simulations and makes it into a [bugs](#) object that can be conveniently displayed using `print` and `plot` and accessed using `attach.bugs`. If the third dimension of `sims()` has names, the resulting bugs object will respect that naming convention. For example, if the parameter names are “alpha[1]”, “alpha[2]”, ..., “alpha[8]”, “mu”, “tau”, then `as.bugs.array` will know that alpha is a vector of length 8, and mu and tau are scalar parameters. These will all be plotted appropriately by `plot` and attached appropriately by `attach.bugs`.

If `DIC=TRUE` then DIC can be either already passed to argument `DICOutput` or calculated from deviance values in `sims.array`.

Value

A [bugs](#) object is returned

Author(s)

Jouni Kerman, <kerman@stat.columbia.edu> with modification by Andrew Gelman, <gelman@stat.columbia.edu>, packaged by Uwe Ligges, <ligges@statistik.tu-dortmund.de>.

See Also[bugs](#)

`attach.all`*Attach / detach elements of (bugs) objects to search path*

Description

The database is attached/detached to the search path. See [attach](#) for details.

Usage

```
attach.all(x, overwrite = NA, name = "attach.all")
attach.bugs(x, overwrite = NA)
detach.all(name = "attach.all")
detach.bugs()
```

Arguments

<code>x</code>	An object, which must be of class <code>bugs</code> for <code>attach.bugs</code> .
<code>overwrite</code>	If <code>TRUE</code> , objects with identical names in the Workspace (<code>.GlobalEnv</code>) that are masking objects in the database to be attached will be deleted. If <code>NA</code> (the default) and an interactive session is running, a dialog box asks the user whether masking objects should be deleted. In non-interactive mode, behaviour is identical to <code>overwrite=FALSE</code> , i.e. nothing will be deleted.
<code>name</code>	The name of the environment where <code>x</code> will be attached / which will be detached.

Details

While `attach.all` attaches all elements of an object `x` to a database called `name`, `attach.bugs` attaches all elements of `x$sims.list` to the database `bugs.sims` itself making use of `attach.all`.

`detach.all` and `detach.bugs` are removing the databases mentioned above.

`attach.all` also attaches `n.sims` (the number of simulations saved from the MCMC runs) to the database.

Each scalar parameter in the model is attached as vectors of length `n.sims`, each vector is attached as a 2-way array (with first dimension equal to `n.sims`), each matrix is attached as a 3-way array, and so forth.

Value

`attach.all` and `attach.bugs` invisibly return the [environment](#)(s).

`detach.all` and `detach.bugs` detach the environment(s) named `name` created by `attach.all`.

Note

Without detaching, do not use `attach.all` or `attach.bugs` on another (bugs) object, because instead of the given name, an object called `name` is attached. Therefore strange things may happen ...

See Also

[bugs](#), [attach](#), [detach](#)

Examples

```
# An example model file is given in:
model.file <- system.file("model", "schools.txt", package="R2OpenBUGS")
# Some example data (see ?schools for details):
data(schools)
J <- nrow(schools)
y <- schools$estimate
sigma.y <- schools$sd
data <- list ("J", "y", "sigma.y")
inits <- function(){
  list(theta = rnorm(J, 0, 100), mu.theta = rnorm(1, 0, 100),
        sigma.theta = runif(1, 0, 100))
}
parameters <- c("theta", "mu.theta", "sigma.theta")
## Not run:
## See ?bugs if the following fails:
schools.sim <- bugs(data, inits, parameters, model.file,
  n.chains = 3, n.iter = 1000,
  working.directory = NULL)

# Do some inferential summaries
attach.bugs(schools.sim)
# posterior probability that the coaching program in school A
# is better than in school C:
print(mean(theta[,1] > theta[,3]))
# 50
# and school C's program:
print(quantile(theta[,1] - theta[,3], c(.25, .75)))
plot(theta[,1], theta[,3])
detach.bugs()

## End(Not run)
```

Description

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

Usage

```
bugs(data, inits, parameters.to.save, n.iter, model.file="model.txt",
     n.chains=3, n.burnin=floor(n.iter / 2), n.thin=1,
     saveExec=FALSE, restart=FALSE,
     debug=FALSE, DIC=TRUE, digits=5, codaPkg=FALSE,
     OpenBUGS.pgm=NULL,
     working.directory=NULL,
     clearWD=FALSE, useWINE=FALSE, WINE=NULL,
     newWINE=TRUE, WINEPATH=NULL, bugs.seed=1, summary.only=FALSE,
     save.history=(.Platform$OS.type == "windows" | useWINE==TRUE),
     over.relax = FALSE)
```

Arguments

data	either a named list (names corresponding to variable names in the <code>model.file</code>) of the data for the OpenBUGS model, <i>or</i> a vector or list of the names of the data objects used by the model. If data is a one element character vector (such as "data.txt"), it is assumed that data have already been written to the working directory into that file, e.g. by the function <code>bugs.data</code> .
inits	a list with <code>n.chains</code> elements; each element of the list is itself a list of starting values for the OpenBUGS model, <i>or</i> a function creating (possibly random) initial values. Alternatively, if <code>inits=NULL</code> , initial values are generated by OpenBUGS . If <code>inits</code> is a character vector with <code>n.chains</code> elements, it is assumed that <code>inits</code> have already been written to the working directory into those files, e.g. by the function <code>bugs.inits</code> .
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 OpenBUGS code. The extension must be <code>'.txt'</code> . The default location is given by <code>working.directory</code> . The old convention allowing <code>model.file</code> to be named <code>'.bug'</code> has been eliminated because the new OpenBUGS feature that allows the program image to be saved and later restarted uses the <code>.bug</code> extension for the saved images. Alternatively, <code>model.file</code> can be an R function that contains a BUGS model that is written to a temporary model file (see <code>tempfile</code>) using <code>write.model</code> .
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 <code>n.iter/2</code> , that is, discarding the first half of the simulations.
n.thin	Thinning rate. Must be a positive integer. The default is <code>n.thin = 1</code> . The thinning is implemented in the OpenBUGS update phase, so thinned samples are never stored, and they are not counted in <code>n.burnin</code> or <code>n.iter</code> . Setting <code>n.thin=2</code> , doubles the number of iterations OpenBUGS performs, but does not change <code>n.iter</code> or <code>n.burnin</code> . Thinning implemented in this manner is not captured in summaries created by packages such as coda .

saveExec	If TRUE, a re-startable image of the OpenBUGS execution is saved with basename (model, file) and extension .bug in the working directory, which must be specified. The .bug files can be large, so users should monitor them carefully and remove them when not needed.
restart	If TRUE, execution resumes with the final status from the previous execution stored in the .bug file in the working directory. If $n.burnin=0$, additional iterations are performed and all iterations since the previous burnin are used (including those from past executions). If $n.burnin>0$, a new burnin is performed, and the previous iterations are discarded, but execution continues from the status at the end of the previous execution. When <code>restart=TRUE</code> , only <code>n.burnin</code> , <code>n.iter</code> , and <code>saveExec</code> inputs should be changed from the call creating the .bug file, otherwise failed or erratic results may be produced. Note the default has $n.burnin>0$.
debug	if FALSE (default), OpenBUGS is closed automatically when the script has finished running, otherwise OpenBUGS remains open for further investigation. The debug option is not available for linux execution.
DIC	logical; if TRUE (default), compute deviance, pD, and DIC. The results are extracted directly from the OpenBUGS log, which uses the rule $pD = \bar{D} - \hat{D}$. If extraction fails or if there are less iterations than required for the adaptive phase, the rule $pD = \text{var}(\text{deviance}) / 2$ is computed in R. See bugs.log for more information on extracting results from the log file.
digits	number of significant digits used for OpenBUGS input, see formatC
codaPkg	logical; if FALSE (default) a bugs object is returned, if TRUE file names of OpenBUGS output are returned for easy access by the coda package through function read.bugs . A bugs object can be converted to an <code>mcmc.list</code> object as used by the coda package with the method <code>as.mcmc.list</code> (for which a method is provided by R2OpenBUGS).
OpenBUGS.pgm	For Windows or WINE execution, the full path to the OpenBUGS executable. For linux execution, the full path to the OpenBUGS executable or shell script (the path to the shell script is not required if the OpenBUGS shell script is in the user's PATH variable). If NULL (unset) and the environment variable <code>OpenBUGS_PATH</code> is set the latter will be used as the default. If NULL (unset), the environment variable <code>OpenBUGS_PATH</code> is unset and the global option <code>R2OpenBUGS.pgm</code> is not NULL the latter will be used as the default. If none of the former are set and OS is Windows, the most recent OpenBUGS version registered in the Windows registry will be used as the default. For other operating systems, the location is guessed by <code>Sys.which("OpenBUGS")</code> .
working.directory	sets working directory during execution of this function; OpenBUGS ' input and output will be stored in this directory; if NULL, a temporary working directory via tempdir is used.
clearWD	logical; indicating whether the files 'data.txt', 'inits[1:n.chains].txt', 'log.odc', 'codaIndex.txt', and 'coda[1:n.chains].txt' should be removed after OpenBUGS has finished. If set to TRUE, this argument is only respected if <code>codaPkg=FALSE</code> .

useWINE	logical; attempt to use the Wine emulator to run OpenBUGS . Default is FALSE. If WINE is used, the arguments <code>OpenBUGS.pgm</code> and <code>working.directory</code> must be given in form of Linux paths rather than Windows paths (if not NULL).
WINE	Character, path to ‘wine’ binary file, it is tried hard (by a guess and the utilities <code>which</code> and <code>locate</code>) to get the information automatically if not given.
newWINE	Use new versions of Wine that have ‘winepath’ utility
WINEPATH	Character, path to ‘winepath’ binary file, it is tried hard (by a guess and the utilities <code>which</code> and <code>locate</code>) to get the information automatically if not given.
bugs.seed	Random seed for OpenBUGS . Must be an integer between 1-14. Seed specification changed between WinBUGS and OpenBUGS; see the OpenBUGS documentation for details.
summary.only	If TRUE, only a parameter summary for very quick analyses is given, temporary created files are not removed in that case.
save.history	If TRUE (the default), trace plots are generated at the end.
over.relax	If TRUE, over-relaxed form of MCMC is used if available from OpenBUGS.

Details

To run:

1. Write a **BUGS** model in an ASCII file (hint: use `write.model`).
2. Go into R.
3. Prepare the inputs for the bugs function and run it (see Example section).
4. An **OpenBUGS** window will pop up and R will freeze up. The model will now run in **OpenBUGS**. It might take awhile. You will see things happening in the Log window within **OpenBUGS**. When **OpenBUGS** is done, its window will close and R will work again.
5. If an error message appears, re-run with `debug=TRUE`.

BUGS version support:

- **OpenBUGS** $\geq 3.2.1$

Operation system support:

- **MS Windows** no problem
- **Linux, intel processors** GUI display and graphics not available.
- **Mac OS X** and **Unix** in general possible with Wine emulation via `useWINE=TRUE`

If `useWINE=TRUE` is used, all paths (such as `working.directory` and `model.file`, must be given in native (Unix) style, but `OpenBUGS.pgm` can be given in Windows path style (e.g. “`c:/Program Files/OpenBUGS/`”) or native (Unix) style (e.g. “`/path/to/wine/folder/dosdevices/c:/Program Files/OpenBUGS/OpenBUGS321/OpenBUGS.exe`”).

Value

If `codaPkg=TRUE` the returned values are the names of coda output files written by **OpenBUGS** containing the Markov Chain Monte Carlo output in the CODA format. This is useful for direct access with `read.bugs`.

If `codaPkg=FALSE`, the following values are returned:

<code>n.chains</code>	see Section ‘Arguments’
<code>n.iter</code>	see Section ‘Arguments’
<code>n.burnin</code>	see Section ‘Arguments’
<code>n.thin</code>	see Section ‘Arguments’
<code>n.keep</code>	number of iterations kept per chain (equal to $(n.iter - n.burnin) / n.thin$)
<code>n.sims</code>	number of posterior simulations (equal to $n.chains * n.keep$)
<code>sims.array</code>	3-way array of simulation output, with dimensions <code>n.keep</code> , <code>n.chains</code> , and length of combined parameter vector
<code>sims.list</code>	list of simulated parameters: for each scalar parameter, a vector of length <code>n.sims</code> for each vector parameter, a 2-way array of simulations, for each matrix parameter, a 3-way array of simulations, etc. (for convenience, the $n.keep * n.chains$ simulations in <code>sims.matrix</code> and <code>sims.list</code> (but NOT <code>sims.array</code>) have been randomly permuted)
<code>sims.matrix</code>	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 <code>sims.matrix</code> and <code>sims.list</code> (but NOT <code>sims.array</code>) have been randomly permuted)
<code>summary</code>	summary statistics and convergence information for each saved parameter.
<code>mean</code>	a list of the estimated parameter means
<code>sd</code>	a list of the estimated parameter standard deviations
<code>median</code>	a list of the estimated parameter medians
<code>root.short</code>	names of argument parameters. <code>to.save</code> and “deviance”
<code>long.short</code>	indexes; programming stuff
<code>dimension.short</code>	dimension of <code>indexes.short</code>
<code>indexes.short</code>	indexes of <code>root.short</code>
<code>last.values</code>	list of simulations from the most recent iteration; they can be used as starting points if you wish to run OpenBUGS for further iterations
<code>pD</code>	an estimate of the effective number of parameters, for calculations see the section “Arguments”.
<code>DIC</code>	$mean(deviance) + pD$

Author(s)

Andrew Gelman, <gelman@stat.columbia.edu>; modifications and packaged by Sibylle Sturtz, <sturtz@statistik.tu-dortmund.de>, Uwe Ligges, and Neal Thomas

References

Gelman, A., Carlin, J.B., Stern, H.S., Rubin, D.B. (2003): *Bayesian Data Analysis*, 2nd edition, CRC Press.

Sturtz, S., Ligges, U., Gelman, A. (2005): R2WinBUGS: A Package for Running WinBUGS from R. *Journal of Statistical Software* 12(3), 1-16.

See Also

[print.bugs](#), [plot.bugs](#), as well as **coda** and **BRugs** packages

Examples

```
## Not run:
# An example model file is given in:
model.file <- system.file(package="R2OpenBUGS", "model", "schools.txt")
# Let's take a look:
#file.show(model.file)

# Some example data (see ?schools for details):
data(schools)
schools

J <- nrow(schools)
y <- schools$estimate
sigma.y <- schools$sd
data <- list("J", "y", "sigma.y")
inits <- function(){
  list(theta=rnorm(J, 0, 100), mu.theta=rnorm(1, 0, 100),
        sigma.theta=runif(1, 0, 100))
}
## or alternatively something like:
# inits <- list(
#   list(theta=rnorm(J, 0, 90), mu.theta=rnorm(1, 0, 90),
#         sigma.theta=runif(1, 0, 90)),
#   list(theta=rnorm(J, 0, 100), mu.theta=rnorm(1, 0, 100),
#         sigma.theta=runif(1, 0, 100)),
#   list(theta=rnorm(J, 0, 110), mu.theta=rnorm(1, 0, 110),
#         sigma.theta=runif(1, 0, 110)))

parameters <- c("theta", "mu.theta", "sigma.theta")

## You may need to specify "OpenBUGS.pgm"
## also you need write access in the working directory:
schools.sim <- bugs(data, inits, parameters, model.file,
  n.chains=3, n.iter=5000)
print(schools.sim)
plot(schools.sim)

## End(Not run)
```

bugs.data *Writing input for OpenBUGS*

Description

Write file for **OpenBUGS** to read.

Usage

```
bugs.data(data, dir = getwd(), digits = 5, data.file = "data.txt")
```

Arguments

data	either a named list (names corresponding to variable names in the model.file) of the data for the OpenBUGS model, <i>or</i> a vector or list of the names of the data objects used by the model
dir	the directory to write the file 'data.txt' to
digits	number of significant digits used for OpenBUGS input, see formatC
data.file	name for the file R writes the data into.

Value

The name of data.file is returned and as a side effect, the data file is written

See Also

The main function to be called by the user is [bugs](#).

bugs.inits *Writing input for OpenBUGS*

Description

Write files 'inits1.txt', 'inits2.txt', etc., in the working directory for **OpenBUGS** to read

Usage

```
bugs.inits(inits, n.chains, digits,
           inits.files = paste("inits", 1:n.chains, ".txt", sep = ""))
```

Arguments

inits	a list with n.chains elements; each element of the list is itself a list of starting values for the OpenBUGS model, <i>or</i> a function creating (possibly random) initial values
n.chains	number of Markov chains
digits	number of significant digits used for OpenBUGS input, see formatC
inits.files	name for the inits files R write the inits into.

Value

Vector of names of inits.files; as a side effect, the inits files ‘inits*.txt’ are written

See Also

The main function to be called by the user is [bugs](#).

bugs.log	<i>Read data from OpenBUGS logfile</i>
----------	--

Description

Read data such as summary statistics and DIC information from the **OpenBUGS** logfile

Usage

```
bugs.log(file)
```

Arguments

file	Location of the OpenBUGS logfile
------	---

Details

Returns the OpenBUGS summary statistics and DIC extracted directly from the log file.

Value

A list with components:

stats	A matrix containing summary statistics for each saved parameter. Comparable to the information in the element summary of a bugs object as returned by bugs .
DIC	A matrix containing the DIC statistics as returned from OpenBUGS .

Author(s)

Jouni Kerman

See Also

The main function that generates the log file is [bugs](#).

plot.bugs	<i>Plotting a bugs object</i>
-----------	-------------------------------

Description

Plotting a bugs object

Usage

```
## S3 method for class 'bugs'
plot(x, display.parallel = FALSE, ...)
```

Arguments

x an object of class ‘bugs’, see [bugs](#) for details

display.parallel display parallel intervals in both halves of the summary plots; this is a convergence-monitoring tool and is not necessary once you have approximate convergence (default is FALSE)

... further arguments to [plot](#)

See Also

[bugs](#)

print.bugs	<i>Printing a bugs object</i>
------------	-------------------------------

Description

Printing a bugs object

Usage

```
## S3 method for class 'bugs'
print(x, digits.summary = 1, ...)
```

Arguments

x an object of class ‘bugs’, see [bugs](#) for details

digits.summary rounding for tabular output on the console (default is to round to 1 decimal place)

... further arguments to [print](#)

See Also[bugs](#)

read.bugs	<i>Read output files in CODA format</i>
-----------	---

Description

This function reads Markov Chain Monte Carlo output in the CODA format produced by **OpenBUGS** and returns an object of class `mcmc.list` for further output analysis using the `coda` package.

Usage

```
read.bugs(codafiles, ...)
```

Arguments

codafiles	character vector of filenames (e.g. returned from bugs in call such as <code>bugs(..., codaPkg=TRUE, ...)</code>). Each of the files contains coda output for one chain produced by OpenBUGS , the <i>directory</i> name of the corresponding file ‘CODAindex.txt’ is extracted from the first element of codafiles.
...	further arguments to be passed to read.coda

See Also[bugs](#), [read.coda](#), [mcmc.list](#)

schools	<i>8 schools analysis</i>
---------	---------------------------

Description

8 schools analysis

Usage

```
data(schools)
```

Format

A data frame with 8 observations on the following 3 variables.

school See Source.

estimate See Source.

sd See Source.

Source

Rubin, D.B. (1981): Estimation in Parallel Randomized Experiments. *Journal of Educational Statistics* 6(4), 377-400.

Section 5.5 of Gelman, A., Carlin, J.B., Stern, H.S., Rubin, D.B. (2003): *Bayesian Data Analysis*, 2nd edition, CRC Press.

```
validateInstallOpenBUGS
```

Compare OpenBUGS/R2OpenBUGS execution to results supplied with OpenBUGS

Description

A selected subset of the examples from the OpenBUGS manual is executed and compared to results supplied with the package to validate installation.

Usage

```
validateInstallOpenBUGS(  
  OpenBUGS.pgm=NULL,  
  useWINE=FALSE, WINE=NULL,  
  newWINE=TRUE, WINEPATH=NULL  
)
```

Arguments

OpenBUGS.pgm	See bugs .
useWINE	logical; attempt to use the Wine emulator to run OpenBUGS . Default is FALSE. If WINE is used, the arguments OpenBUGS.pgm and working.directory must be given in form of Linux paths rather than Windows paths (if not NULL).
WINE	Character, path to 'wine' binary file, it is tried hard (by a guess and the utilities which and locate) to get the information automatically if not given.
newWINE	Use new versions of Wine that have 'winepath' utility
WINEPATH	Character, path to 'winepath' binary file, it is tried hard (by a guess and the utilities which and locate) to get the information automatically if not given.

Details

Operation system support:

- **MS Windows** Yes
- **Linux, intel processors** Yes, but GUI display and graphics not available.
- **Mac OS X and Unix** Wine emulation via useWINE=TRUE

If useWINE=TRUE is used, all paths (such as working.directory and model.file, must be given in native (Unix) style, but OpenBUGS.pgm can be given in Windows path style (e.g. "c:/Program Files/OpenBUGS/") or native (Unix) style (e.g. "/path/to/wine/folder/dosdevices/c:/Program Files/OpenBUGS/OpenBUGS321/OpenBUGS.exe").

Value

No data returned. Prints match/no match result to console for each example.

Author(s)

Neal Thomas based on BRugs examples created by Chris Jackaon.

References

Gelman, A., Carlin, J.B., Stern, H.S., Rubin, D.B. (2003): *Bayesian Data Analysis*, 2nd edition, CRC Press.

Sturtz, S., Ligges, U., Gelman, A. (2005): R2WinBUGS: A Package for Running WinBUGS from R. *Journal of Statistical Software* 12(3), 1-16.

See Also

[bugs](#)

write.model

Creating a OpenBUGS model file

Description

Convert R function to a **OpenBUGS** model file

Usage

```
write.model(model, con = "model.bug", digits = 5)
```

Arguments

model	R function containing the BUGS model in the BUGS model language, for minor differences see Section Details.
con	passed to writeLines which actually writes the model file
digits	number of significant digits used for OpenBUGS input, see formatC

Details

BUGS models follow closely S syntax. It is therefore possible to write most BUGS models as R functions.

As a difference, BUGS syntax allows truncation specification like this: `dnorm(...) I(...)` but this is illegal in R. To overcome this incompatibility, use dummy operator `%_%` before `I(...)`: `dnorm(...) %_% I(...)`. The dummy operator `%_%` will be removed before the BUGS code is saved.

Value

Nothing, but as a side effect, the model file is written

Author(s)

original idea by Jouni Kerman, modified by Uwe Ligges

See Also

[bugs](#)

Examples

```
## Same "schoolmodel" that is used in the examples in ?bugs:
schoolmodel <- function(){
  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)
}

## some temporary filename:
filename <- file.path(tempdir(), "schoolmodel.bug")

## write model file:
write.model(schoolmodel, filename)
## and let's take a look:
file.show(filename)
```

Index

- * **IO**
 - bugs.data, 11
 - bugs.inits, 11
 - bugs.log, 12
 - read.bugs, 14
 - write.model, 16
 - * **datasets**
 - schools, 14
 - * **data**
 - attach.all, 4
 - * **file**
 - bugs.data, 11
 - bugs.inits, 11
 - bugs.log, 12
 - read.bugs, 14
 - write.model, 16
 - * **hplot**
 - plot.bugs, 13
 - * **interface**
 - as.bugs.array, 3
 - bugs, 5
 - validateInstallOpenBUGS, 15
 - * **manip**
 - as.bugs.array, 3
 - * **models**
 - bugs, 5
 - validateInstallOpenBUGS, 15
 - * **model**
 - write.model, 16
 - * **package**
 - R2OpenBUGS-package, 2
 - * **print**
 - print.bugs, 13
-
- as.bugs.array, 3
 - as.mcmc.list, 7
 - attach, 4, 5
 - attach.all, 4
 - attach.bugs (attach.all), 4
 - bugs, 2-5, 5, 11-17
 - bugs.data, 6, 11
 - bugs.inits, 6, 11
 - bugs.log, 7, 12
 - detach, 5
 - detach.all (attach.all), 4
 - detach.bugs (attach.all), 4
 - environment, 4
 - formatC, 7, 11, 12, 16
 - mcmc.list, 14
 - plot, 13
 - plot.bugs, 2, 3, 10, 13
 - print, 13
 - print.bugs, 2, 10, 13
 - R2OpenBUGS (R2OpenBUGS-package), 2
 - R2OpenBUGS-package, 2
 - read.bugs, 7, 9, 14
 - read.coda, 14
 - schools, 14
 - tempdir, 7
 - tempfile, 6
 - validateInstallOpenBUGS, 15
 - write.model, 2, 6, 8, 16
 - writeln, 16