

Package ‘R2WinBUGS’

August 29, 2013

Title Running WinBUGS and OpenBUGS from R / S-PLUS

Date 2013-04-07

Version 2.1-19

Author originally written 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>. Ported to S-PLUS by Insightful Corp.

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 / S-PLUS. In S-PLUS, the openbugs functionality and the windows emulation functionality is not yet available.

Depends R (>= 2.13.0), coda (>= 0.11-0), boot

Suggests BRugs (>= 0.3-2)

SystemRequirements WinBUGS 1.4

URL <http://www.stat.columbia.edu/~gelman/bugsR/>

Maintainer Uwe Ligges <ligges@statistik.tu-dortmund.de>

License GPL-2

Dialect R, S-PLUS

NeedsCompilation no

Repository CRAN

Date/Publication 2013-04-07 19:01:39

R topics documented:

R2WinBUGS-package	2
as.bugs.array	2
attach.all	4
bugs	5
bugs.log	10
openbugs	11
plot.bugs	13
print.bugs	14
read.bugs	15
schools	15
write.model	16
Index	18

R2WinBUGS-package	<i>Running WinBUGS and OpenBUGS from R / S-PLUS</i>
-------------------	---

Description

R2WinBUGS package provides possibility 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 / S-PLUS**. In **S-PLUS**, the **OpenBUGS** functionality and the windows emulation functionality is not yet available. The main command is [bugs](#).

Details

The following are sources of information on **R2WinBUGS** package:

DESCRIPTION file	library(help="R2WinBUGS")
This file	package?R2WinBUGS
Vignette	vignette("R2WinBUGS")
Some help files	bugs write.model print.bugs plot.bugs
News	file.show(system.file("NEWS", package="R2WinBUGS"))

as.bugs.array	<i>Convert to bugs object</i>
---------------	-------------------------------

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 WinBUGS 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 as it is done in [openbugs](#) 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	<i>Attach / detach elements of (bugs) objects to search path</i>
------------	--

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

x	An object, which must be of class bugs for attach.bugs.
overwrite	If TRUE, objects with identical names in the Workspace (.GlobalEnv) that are masking objects in the database to be attached will be deleted. If NA (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 overwrite=FALSE, i.e. nothing will be deleted.
name	The name of the environment where x 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="R2WinBUGS")
# 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:
## You may need to edit "bugs.directory",
## also you need write access in the working directory:
schools.sim <- bugs(data, inits, parameters, model.file,
  n.chains = 3, n.iter = 1000,
  bugs.directory = "c:/Program Files/WinBUGS14/",
  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 **WinBUGS** script, calls the model, and saves the simulations for easy access in **R** or **S-PLUS**.

Usage

```
bugs(data, inits, parameters.to.save, model.file="model.bug",
     n.chains=3, n.iter=2000, n.burnin=floor(n.iter/2),
     n.thin=max(1, floor(n.chains * (n.iter - n.burnin) / n.sims)),
     n.sims = 1000, bin=(n.iter - n.burnin) / n.thin,
     debug=FALSE, DIC=TRUE, digits=5, codaPkg=FALSE,
     bugs.directory="c:/Program Files/WinBUGS14/",
     program=c("WinBUGS", "OpenBUGS", "winbugs", "openbugs"),
     working.directory=NULL, clearWD=FALSE,
     useWINE=.Platform$OS.type != "windows", WINE=NULL,
     newWINE=TRUE, WINEPATH=NULL, bugs.seed=NULL, summary.only=FALSE,
     save.history=!summary.only, over.relax = FALSE)
```

Arguments

<code>data</code>	either a named list (names corresponding to variable names in the <code>model.file</code>) of the data for the WinBUGS model, <i>or</i> (which is not recommended and unsafe) a vector or list of the names of the data objects used by the model. If <code>data</code> is a one element character vector (such as <code>"data.txt"</code>), it is assumed that data have already been written to the working directory into that file, e.g. by the function bugs.data .
<code>inits</code>	a list with <code>n.chains</code> elements; each element of the list is itself a list of starting values for the WinBUGS model, <i>or</i> a function creating (possibly random) initial values. Alternatively, if <code>inits=NULL</code> , initial values are generated by WinBUGS . 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 bugs.inits .
<code>parameters.to.save</code>	character vector of the names of the parameters to save which should be monitored
<code>model.file</code>	file containing the model written in WinBUGS code. The extension can be either <code>' .bug'</code> or <code>' .txt'</code> . If the extension is <code>' .bug'</code> and <code>program=="WinBUGS"</code> , a copy of the file with extension <code>' .txt'</code> will be created in the <code>bugs()</code> call and removed afterwards. Note that similarly named <code>' .txt'</code> files will be overwritten. Alternatively, <code>model.file</code> can be an R function that contains a BUGS model that is written to a temporary model file (see tempfile) using write.model .
<code>n.chains</code>	number of Markov chains (default: 3)
<code>n.iter</code>	number of total iterations per chain (including burn in; default: 2000)
<code>n.burnin</code>	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. Set <code>n.thin > 1</code> to save memory and computation time if <code>n.iter</code> is large. Default is <code>max(1, floor(n.chains * (n.iter-n.burnin) / 1000))</code> which will only thin if there are at least 2000 simulations.
n.sims	The approximate number of simulations to keep after thinning.
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. This is done in WinBUGS directly using the rule $pD = \bar{D} - \hat{D}$. If there are less iterations than required for the adaptive phase, the rule $pD = \text{var}(\text{deviance}) / 2$ is used.
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 through function read.bugs (not used if <code>program="OpenBUGS"</code>). A bugs object can be converted to an <code>mcmc.list</code> object as used by the coda package with the method as.mcmc.list (for which a method is provided by R2WinBUGS).
bugs.directory	directory that contains the WinBUGS executable. If the global option <code>R2WinBUGS.bugs.directory</code> is not NULL, it will be used as the default.
program	the program to use, either <code>winbugs/WinBUGS</code> or <code>openbugs/OpenBUGS</code> , the latter makes use of function openbugs and requires the CRAN package BRugs . The <code>openbugs/OpenBUGS</code> choice is not available in S-PLUS.
working.directory	sets working directory during execution of this function; WinBUGS ' in- 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:nchains].txt' should be removed after WinBUGS 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 WinBUGS , defaults to FALSE on Windows, and TRUE otherwise. Not available in S-PLUS.
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 WinBUGS (default is no seed)
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 WinBUGS.

Details

To run:

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

BUGS version support:

- **WinBUGS** 1.4.*default
- **OpenBUGS** 2.*via argument `program="OpenBUGS"`

Operation system support:

- **MS Windows**no problem
- **Linux, Mac OS X** and **Unix** in generalpossible with Wine emulation via `useWINE=TRUE`, but only for **WinBUGS** 1.4.*

If `useWINE=TRUE` is used, all paths (such as `working.directory` and `model.file`, must be given in native (Unix) style, but `bugs.directory` can be given in Windows path style (e.g. “c:/Program Files/WinBUGS14/”) or native (Unix) style (e.g. “/path/to/wine/folder/dosdevices/c:/Program Files/WinBUGS14/”). This is done to achieve greatest portability with default argument value for `bugs.directory`.

Value

If `codaPkg=TRUE` the returned values are the names of coda output files written by **WinBUGS** 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 <code>n.chains*n.keep</code> rows and one column for each element of each saved parameter (for convenience, the <code>n.keep*n.chains</code> 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 to save 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 WinBUGS for further iterations
<code>pD</code>	an estimate of the effective number of parameters, for calculations see the section “Arguments”.
<code>DIC</code>	<code>mean(deviance) + pD</code>

Author(s)

Andrew Gelman, <gelman@stat.columbia.edu>, <http://www.stat.columbia.edu/~gelman/bugsR/>; modifications and packaged by Sibylle Sturtz, <sturtz@statistik.tu-dortmund.de>, and Uwe Ligges.

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

```
# An example model file is given in:
model.file <- system.file(package="R2WinBUGS", "model", "schools.txt")
# Let's take a look:
file.show(model.file)

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


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 WinBUGS .

Author(s)

Jouni Kerman

See Also

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

openbugs

Wrapper to run OpenBUGS

Description

The openbugs function takes data and starting values as input. It automatically calls the package **BRugs** and runs something similar to [BRugsFit](#). Not available in S-PLUS.

Usage

```
openbugs(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) / n.sims)),
  n.sims = 1000, DIC = TRUE,
  bugs.directory = "c:/Program Files/OpenBUGS/",
  working.directory = NULL, digits = 5, over.relax = FALSE, seed=NULL)
```

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. 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 bugs.data .
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. Alternatively, if inits are missing or inits = NULL, initial values are generated by OpenBUGS .
parameters.to.save	character vector of the names of the parameters to save which should be monitored

<code>model.file</code>	file containing the model written in OpenBUGS code. The extension can be either <code>‘.bug’</code> or <code>‘.txt’</code> . If <code>‘.bug’</code> , a copy of the file with extension <code>‘.txt’</code> will be created in the <code>bugs()</code> call and removed afterwards. Note that similarly named <code>‘.txt’</code> files will be overwritten.
<code>n.chains</code>	number of Markov chains (default: 3)
<code>n.iter</code>	number of total iterations per chain (including burn in; default: 2000)
<code>n.burnin</code>	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.
<code>n.thin</code>	thinning rate. Must be a positive integer. Set $n.thin > 1$ to save memory and computation time if <code>n.iter</code> is large. Default is $\max(1, \text{floor}(n.chains * (n.iter - n.burnin) / 1000))$ which will only thin if there are at least 2000 simulations.
<code>n.sims</code>	The approximate number of simulations to keep after thinning.
<code>DIC</code>	logical; if TRUE (default), compute deviance, pD, and DIC. This is done in BRugs directly.
<code>digits</code>	number of significant digits used for OpenBUGS input, see formatC
<code>bugs.directory</code>	directory that contains the OpenBUGS executable - currently unused
<code>working.directory</code>	sets working directory during execution of this function; WinBUGS in- and output will be stored in this directory; if NULL, a temporary working directory via tempdir is used.
<code>over.relax</code>	If TRUE, over-relaxed form of MCMC is used if available from OpenBUGS.
<code>seed</code>	random seed (default is no seed)

Value

A [bugs](#) object.

Note

By default, **BRugs** (and hence `openbugs()`) is quite verbose. This can be controlled for the whole **BRugs** package by the option `‘BRugsVerbose’` (see [options](#)) which is set to TRUE by default.

Author(s)

Andrew Gelman, <gelman@stat.columbia.edu>, <http://www.stat.columbia.edu/~gelman/bugsR/>; modifications and packaged by Sibylle Sturtz, <sturtz@statistik.tu-dortmund.de>, and Uwe Ligges.

See Also

[bugs](#) and the **BRugs** package

Examples

```
# An example model file is given in:
model.file <- system.file(package = "R2WinBUGS", "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")

## Not run:
## both write access in the working directory and package BRugs required:
schools.sim <- bugs(data, inits, parameters, model.file,
  n.chains = 3, n.iter = 5000,
  program = "openbugs", working.directory = NULL)
print(schools.sim)
plot(schools.sim)

## End(Not run)
```

plot.bugs

Plotting a bugs object

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 **WinBUGS** and returns an object of class `mcmc.list` for further output analysis using the **coda** package.

Usage

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

Arguments

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

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.

write.model

Creating a WinBUGS model file

Description

Convert R / S-PLUS function to a **WinBUGS** model file

Usage

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

Arguments

model	R / S-PLUS 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 WinBUGS 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 and S-PLUS. To overcome this incompatibility, use dummy operator `%_%` before `I(...)`: `dnorm(...)` `%_%` `I(...)`. The dummy operator `%_%` will be removed before the BUGS code is saved.

In S-PLUS, a warning is generated when the model function is defined if the last statement in the model is an assignment. To avoid this warning, add the line `"invisible()"` to the end of the model definition. This line 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 "schoolsmodel" that is used in the examples in ?bugs:
schoolsmodel <- 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)
}

if (is.R()){ # for R
  ## some temporary filename:
  filename <- file.path(tempdir(), "schoolsmodel.bug")
} else{ # for S-PLUS
  ## put the file in the working directory:
  filename <- "schoolsmodel.bug"
}

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

Index

- *Topic **IO**
 - bugs.log, [10](#)
 - read.bugs, [15](#)
 - write.model, [16](#)
- *Topic **datasets**
 - schools, [15](#)
- *Topic **data**
 - attach.all, [4](#)
- *Topic **file**
 - bugs.log, [10](#)
 - read.bugs, [15](#)
 - write.model, [16](#)
- *Topic **hplot**
 - plot.bugs, [13](#)
- *Topic **interface**
 - as.bugs.array, [2](#)
 - bugs, [5](#)
 - openbugs, [11](#)
- *Topic **manip**
 - as.bugs.array, [2](#)
- *Topic **models**
 - bugs, [5](#)
 - openbugs, [11](#)
- *Topic **package**
 - R2WinBUGS-package, [2](#)
- *Topic **print**
 - print.bugs, [14](#)

as.bugs.array, [2](#)
as.mcmc.list, [7](#)
attach, [4](#), [5](#)
attach.all, [4](#)
attach.bugs (attach.all), [4](#)

BRugsFit, [11](#)
bugs, [2](#), [3](#), [5](#), [5](#), [11](#), [12](#), [14](#), [15](#), [17](#)
bugs.data, [6](#), [11](#)
bugs.inits, [6](#)
bugs.log, [10](#)

detach, [5](#)
detach.all (attach.all), [4](#)
detach.bugs (attach.all), [4](#)

environment, [4](#)

formatC, [7](#), [12](#), [16](#)

mcmc.list, [15](#)

openbugs, [3](#), [7](#), [11](#)
options, [12](#)

plot, [14](#)
plot.bugs, [2](#), [3](#), [9](#), [13](#)
print, [14](#)
print.bugs, [2](#), [9](#), [14](#)

R2WinBUGS (R2WinBUGS-package), [2](#)
R2WinBUGS-package, [2](#)
read.bugs, [7](#), [8](#), [15](#)
read.coda, [15](#)

schools, [15](#)

tempdir, [7](#), [12](#)
tempfile, [6](#)

write.model, [2](#), [6](#), [8](#), [16](#)
writeLines, [16](#)