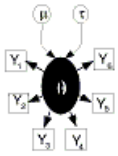


OpenBUGS User Manual

Version 3.2.3 March 2014

David Spiegelhalter [1] Andrew Thomas [2] Nicky Best [3] Dave Lunn [1]

[1] MRC Biostatistics Unit, Institute of Public Health, Robinson Way, Cambridge CB2 2SR, UK

[2] Dept of Mathematics & Statistics, University of St Andrews, St Andrews, Scotland

[3] Department of Epidemiology & Public Health, Imperial College School of Medicine, Norfolk Place, London W2 1PG, UK

e-mail:
bugs@mrc-bsu.cam.ac.uk [general]
helsinkiant@gmail.com [technical]

internet: http://www.mrc-bsu.cam.ac.uk/bugs

This manual describes how to use the OpenBUGS software. It starts with general sections on setting up statistical models in OpenBUGS and using Markov Chain Monte Carlo sampling for statistical inference. Then the idea of compound documents and the many menu options of the Windows interface to OpenBUGS are described in detail.

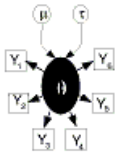
The OpenBUGS software is Open Source

please click here to read the legal bit

Potential users are reminded to be extremely careful if using this program for serious statistical analysis. We have tested the program on quite a wide set of examples, but be particularly careful with types of model that are currently not featured. If there is a problem, *OpenBUGS* might just crash, which is not very good, but it might well carry on and produce answers that are wrong, which is even worse. Please let us know of any successes or failures.

Beware: MCMC sampling can be dangerous!

Contents

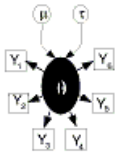


OpenBUGS User Manual

Contents

Introduction
Tutorial
Compound Documents
Model Specification
Scripts and Batch-mode
DoodleBUGS:The Doodle Editor
The File Menu
The Edit Menu
The Attributes Menu
The Tools Menu
The Text Menu
The Info Menu
The Model Menu
The Inference Menu
The Examples Menu
The Manuals Menu
The Help Menu
OpenBUGS Graphics

Tips and Troubleshooting
Advanced use of the BUGS Language
References



Introduction

Contents

Introduction to OpenBUGS
Advice for new users
MCMC methods

Introduction to OpenBUGS [top]

This manual describes the *OpenBUGS* software - a program for Bayesian analysis of complex statistical models using Markov chain Monte Carlo (MCMC) techniques. *OpenBUGS* allows models to be described using the *BUGS* language, or as *Doodles* (graphical representations of models) which can, if desired, be translated to a text-based description. The *BUGS* language is more flexible than the Doodles graphical representation.

Users are advised that this manual only concerns the syntax and functionality of *OpenBUGS* , and does not deal with issues of Bayesian reasoning, prior distributions, statistical modelling, monitoring convergence, and so on. If you are new to MCMC, you are strongly advised to use this software in conjunction with a course in which the strengths and weaknesses of this procedure are described. Please note the disclaimer at the beginning of this manual.

There is a large literature on Bayesian analysis and MCMC methods. For further reading, see, for example, Carlin and Louis (1996), Gelman et al (1995), Gilks, Richardson and Spiegelhalter (1996): Brooks (1998) provides an excellent introduction to MCMC. Chapter 9 of the *Classic BUGS* manual, 'Topics in Modelling', discusses 'non-informative' priors, model criticism, ranking, measurement error, conditional likelihoods, parameterisation, spatial models and so on, while the *CODA* documentation considers convergence diagnostics. Congdon (2001) shows how to analyse a very wide range of models using *OpenBUGS* . The *BUGS* website provides additional links to sites of interest, some of which provide extensive examples and tutorial material.

Advice for new users [top]

If you are using *OpenBUGS* for the first time, the following stages might be reasonable:

1. Step through the simple worked example in the tutorial.

2. Try other examples provided with this release (see Examples Volume 1 , Examples Volume 2 and Examples Volume 3)

3. Edit the *BUGS* language of an example model to fit an example of your own.

If you are interested in using Doodles:

4. Try editing an existing Doodle (e.g. from Examples Volume 1), perhaps to fit a problem of your own.

5. Try constructing a Doodle from scratch.

Note that there are many features in the *BUGS* language that cannot be expressed with Doodles. If you wish to proceed to serious, non-educational use, you may want to dispense with *DoodleBUGS* entirely, or just use it for initially setting up a simplified model that can be elaborated later using the *BUGS* language. Unfortunately we do not have a program to back-translate from a text-based model description to a Doodle!

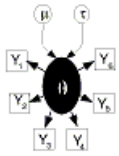
MCMC methods [top]

Users should already be aware of the background to Bayesian Markov Chain Monte Carlo (MCMC) methods: see for example Gilks *et al* (1996). Having specified the model as a full joint distribution on all quantities, whether parameters or observables, we wish to sample values of the unknown parameters from their conditional (posterior) distribution given those stochastic nodes (see model specification section) that have been observed. OpenBUGS uses three families of MCMC algorithms: Gibbs, Metropolis Hasting and slice sampling.

The basic idea behind the Gibbs sampling algorithm is to successively sample from the conditional distribution of each node given all the others in the graph (these are known as full conditional distributions). Gibbs sampling is a special case of the Metropolis Hastings algorithm. The Metropolis Hastings sampling algorithm is appropriate for difficult full conditional distributions . It is also the basis of many block updating algorithms. Note that it does not necessarily generate a new value at each iteration. Slice sampling is a general purpose algorithm for single site updating that always produces a new value at each iteration.

It can be shown that under broad conditions this process eventually provides samples from the joint posterior distribution of the unknown quantities. Empirical summary statistics can be formed from these samples and used to draw inferences about their true values.

OpenBUGS tries to block update groups of nodes that are likely to correlated based on the structure of the model. If *OpenBUGS* is unable to set up block updaters for some nodes in the model, it simulates the remaining nodes using single site updating. This can make convergence very slow and the program very inefficient for models with strongly related parameters.



BUGS Tutorial

Contents

Introduction
Specifying a model in the BUGS language
Running a model in WinBUGS
Monitoring parameter values
Checking convergence
How many iterations after convergence?
Obtaining summaries of the posterior distribution

Introduction [top]

This tutorial is designed to provide new users with a step-by-step guide to running an analysis in OpenBUGS. It is not intended to be prescriptive, but rather to introduce you to the main tools needed to run an MCMC simulation in OpenBUGS, and give some guidance on appropriate usage of the software.

The Seeds example from Volume I of the OpenBUGS examples will be used throughout this tutorial. This example is taken from Table 3 of Crowder (1978) and concerns the proportion of seeds that germinated on each of 21 plates arranged according to a 2 by 2 factorial layout by seed and type of root extract. The data are shown below, where r_i and n_i are the number of germinated and the total number of seeds on the i^{th} plate, i = 1, ..., N. These data are also analysed by, for example, Breslow and Clayton (1993).

seed O. aegyptiaco 75			*seed O. aegyptiaco 73*								
Bean			Cucumber			Bean			Cucumber		
r	n	r/n	r	n	r/n	r	n	r/n	r	n	r/n
10	39	0.26	5	6	0.83	8	16	0.50	3	12	0.25
23	62	0.37	53	74	0.72	10	30	0.33	22	41	0.54
23	81	0.28	55	72	0.76	8	28	0.29	15	30	0.50
26	51	0.51	32	51	0.63	23	45	0.51	32	51	0.63
17	39	0.44	46	79	0.58	0	4	0.00	3	7	0.43
	10	13	0.77								

The model is essentially a random effects logistic regression, allowing for over-dispersion. If p_i is the probability of germination on the i^{th} plate, we assume

$$r_i \sim \text{Binomial}(p_i, n_i)$$

$$\text{logit}(p_i) = a_0 + a_1 x_{1i} + a_2 x_{2i} + a_{12} x_{1i} x_{2i} + b_i$$

$$b_i \sim \text{Normal}(0, \tau)$$

where x_{1i} and x_{2i} are the seed type and root extract of the i^{th} plate, and an interaction term $a_{12} x_{1i} x_{2i}$ is included.

Specifying a model in the BUGS language [top]

The BUGS language allows a concise expression of the model, using the 'twiddles' symbol ~ to denote stochastic (probabilistic) relationships, and the left arrow ('<' sign followed by '-' sign) to denote deterministic (logical) relationships. The stochastic parameters a_0, a_1, a_2, a_{12}, and τ are given proper but minimally informative prior distributions, while the logical expression for sigma allows the standard deviation (of the random effects distribution) to be estimated.

```
model
{
  for (i in 1:N) {
```

```
    r[i] ~ dbin(p[i], n[i])
    b[i] ~ dnorm(0, tau)
    logit(p[i]) <- alpha0 + alpha1 * x1[i] + alpha2 * x2[i]
            + alpha12 * x1[i] * x2[i] + b[i]
  }
  alpha0 ~ dnorm(0, 1.0E-6)
  alpha1 ~ dnorm(0, 1.0E-6)
  alpha2 ~ dnorm(0, 1.0E-6)
  alpha12 ~ dnorm(0, 1.0E-6)
  tau ~ dgamma(0.001, 0.001)
  sigma <- 1 / sqrt(tau)
}
```

More detailed descriptions of the BUGS language along with lists of the available logical functions and stochastic distributions can be found in Model Specification , Appendix I Distributions and Appendix II Functions and functionals . See also the on-line examples:Volume I Volume II and Volume III . An alternative way of specifying a model in OpenBUGS is to use the graphical interface known as DoodleBUGS.

Running a model in OpenBUGS [top]

The OpenBUGS software uses compound documents , which comprise various different types of information (formatted text, tables, formulae, plots, graphs, etc.) displayed in a single window and stored in a single file. This means that it is possible to run the model for the "seeds" example directly from this tutorial document, since the model code can be made 'live' just by highlighting it. However, it is more usual when creating your own models to have the model code and data etc. in separate files. We have therefore created a separate file with the model code in it for this tutorial - you will find the file in Examples/Seedsmodel (or here).

Step 1
Open the Seedsmodel file as follows:

* Point to F ile on the tool bar and click once with the left mouse button (LMB).
* Highlight the O pen... option and click once with LMB.
* Select the Examples directory and double-click on the Seeds file to open.

Step 2
To run the model, we first need to check that the model description does fully define a probability model:

* Point to M odel on the tool bar and click once with LMB.
* Highlight the S pecification... option and click once with LMB.
* Focus the window containing the model code by clicking the LMB once anywhere in the window - the top panel of the window should then become highlighted in blue (usually) to indicate that the window is currently in focus.
* Highlight the word model at the beginning of the code by double clicking on the word with the LMB.
* Check the model syntax by clicking once with LMB on the check model button in the Specification Tool window. A message saying "model is syntactically correct" should appear in the bottom left of the OpenBUGS program window.

Step 3
We next need to load in the data. The data can be represented using S-Plus object notation (file Example/Seedsdata), or as a combination of an S-Plus object and a rectangular array with labels at the head of each column (file Examples/SeedsMixData).

* Open one o f the data files now.

* To load the data in file Examples/Seedsdata:
- Highlight the word list at the beginning of the data file.
- Click once with the LMB on the load data button in the Specification Tool window. A message saying "data loaded" should appear in the bottom left of the OpenBUGS program window.

* To load the data in file Examples/SeedsMixData:
- Highlight the word list at the beginning of the data file.
- Click once with the LMB on the load data button in the Specification Tool window. A message saying "data loaded" should appear in the bottom left of the OpenBUGS program window.
- Next highlight the whole of the header line (i.e. column labels) of the rectangular array data. - Click once with the LMB on the load data button in the Specification Tool window. A message saying "data loaded" should appear in the bottom left of the OpenBUGS program window.

Step 4
Now we need to select the number of chains (i.e. sets of samples to simulate). The default is 1, but we will use 2 chains for this tutorial, since running multiple chains is one way to check the convergence of your MCMC simulations.

* Type the number 2 in the white box labelled num of chains in the Specification Tool window. In practice, if you have a fairly complex model, you may wish to do a pilot run using a single chain to check that the model compiles and runs and obtain an estimate of the time taken per iteration. Once you are happy with the model, re-run it using multiple chains (say 2-5 chains) to obtain a final set of posterior estimates.

Step 5
Next compile the model by clicking once with the LMB on the compile button in the Specification Tool window. A message saying "model compiled" should appear in the bottom left of the OpenBUGS program window. This sets up the internal data structures and chooses the specific MCMC updating algorithms to be used by OpenBUGS for your particular model.

Step 6
Finally the MCMC sampler must be given some initial values for each stochastic node. These can be arbitrary values, although in practice, convergence can be poor if wildly inappropriate values are chosen. You will need a different set of initial values for each chain, i.e. two sets are needed for this tutorial since we have specified two chains - these are stored in files Examples/Seedsinits1 and Examples/Seedsinits2

* Open thess files now.
* To load the initial values:
- Highlight the word list at the beginning of the set of initial values.
- Click once with the LMB on the load inits button in the Specification Tool window. A message saying "initial values loaded: this or another chain contains uninitialized nodes" should appear in the bottom left of the OpenBUGS program window.- Repeat this process for the second initial values file. A message saying "initial values loaded: model initialized" should now appear in the bottom left of the OpenBUGS program window.

Note that you do not need to provide a list of initial values for every parameter in your model. You can get OpenBUGS to generate initial values for any stochastic parameter not already initialized by clicking with the LMB on the gen inits button in the Specification Tool window. OpenBUGS generates initial values by forward sampling from the prior distribution for each parameter. Therefore, you are advised to **provide your own initial values for parameters with vague prior distributions** to avoid wildly inappropriate values.

Step 7
Close the Specification Tool window. You are now ready to start running the simulation. However, before doing so, you will probably want to set some monitors to store the sampled values for selected parameters. For the seeds example, set monitors for the parameters alpha0 , alpha1 , alpha2 , alpha12 and sigma - see here for details on how to do this.

To run the simulation:

* Select the Update... option from the Model menu.
* Type the number of updates (iterations of the simulation) you require in the appropriate white box (labelled updates) - the default value is 1000.
* Click once on the update button: the program will now start simulating values for each parameter in the model. This may take a few seconds - the box marked iteration will tell you how many updates have currently been completed. The number of times this value is revised depends on the value you have set for the refresh option in the white box above the iteration box. The default is every 100 iterations, but you can ask the program to report more frequently by changing refresh to, say, 10 or 1. A sensible choice will depend on how quickly the program runs. For the seeds example, experiment with changing the refresh option from 100 to 10 and then 1.
* When the updates are finished, the message "updates took *** s" will appear in the bottom left of the OpenBUGS program window (where *** is the number of seconds taken to complete the simulation).
* If you previously set monitors for any parameters you can now check convergence and view graphical and numerical summaries of the samples. Do this now for the parameters you monitored in the seeds example - see Checking convergence for tips on how to do this (for the seeds example, you should find that at least 2000 iterations are required for convergence).
* Once you're happy that your simulation has converged, you will need to run some further updates to obtain a sample from the posterior distribution. The section How many iterations after convergence? provides tips on deciding how many more updates you should run. For the seeds example, try running a further 10000 updates.
* Once you have run enough updates to obtain an appropriate sample from the posterior distribution, you may summarise these samples numerically and graphically (see section Obtaining summaries of the posterior distribution for details on how to do this). For the seeds example, summary statistics for the monitored parameters are shown below:

	mean	sd	MC_error	val2.5pc	median	val97.5pc	start	sample
alpha0	-0.5473	0.1935	0.003929	-0.9384	-0.5448	-0.1678	4001	14000
alpha1	0.06959	0.3203	0.006121	-0.5839	0.08103	0.667	4001	14000
alpha12	-0.8165	0.4377	0.008459	-1.69	-0.8186	0.05156	4001	14000
alpha2	1.352	0.2718	0.005731	0.8208	1.349	1.906	4001	14000

sigma 0.2943 0.1435 0.005438 0.05322 0.2827 0.6112 4001 14000

* To save any files created during your OpenBUGS run, focus the window containing the information you want to save, and select the Save As... option from the File menu.
* To quit OpenBUGS, select the Exit option from the File menu.

Monitoring parameter values [top]

In order to check convergence and obtain posterior summaries of the model parameters, you first need to set **monitors** for each parameter of interest. This tells OpenBUGS to store the values sampled for those parameters; otherwise, OpenBUGS automatically discards the simulated values.

There are two types of monitor in OpenBUGS:

Samples monitors: Setting a samples monitor tells OpenBUGS to store *every* value it simulates for that parameter. You will need to set a samples monitor if you want to view trace plots of the samples to check convergence (see Checking convergence) or if you want to obtain 'exact' posterior quantiles, for example, the posterior 95% Bayesian credible interval for that parameter. (Note that you can also obtain *approximate* 2.5%, 50% and 97.5% quantiles of the posterior distribution for each parameter using summary monitors .)

To set a samples monitor:

* Select Samples... from the Inference menu;
* Type the name of the parameter to be monitored in the white box marked node ;
* Click once with the LMB on the button marked set ;
* Repeat for each parameter to be monitored.

Summary monitors: Setting a summary monitor tells OpenBUGS to store the running mean and standard deviation for the parameter, plus approximate running quantiles (2.5%, 50% and 97.5%). The values saved contain less information than saving each individual sample in the simulation, but require much less storage. This is an important consideration when running long simulations (e.g. 1000's of iterations) and storing values for many variables.

We recommend setting summary monitors on long vectors of parameters such as random effects in order to store posterior summaries, and then also setting full samples monitors on a small subset of the random effects, plus other relevant parameters (e.g. means and variances), to check convergence.

To set a summary monitor:

* Select Summary... from the Inference menu;
* Type the name of the parameter to be monitored in the white box marked node ;
* Click once with the LMB on the button marked set ;
* Repeat for each parameter to be monitored.

Note: you should not set a summary monitor until you are happy that convergence has been reached (see Checking convergence), since it is not possible to discard any of the pre-convergence ('burn-in') values from the summary once it is set, ot her than to clear the monitor and re-set it.

Checking convergence [top]

Checking convergence requires considerable care. **It is very difficult to say conclusively that a chain (simulation) has converged, only to diagnose when it definitely hasn't!**

The following are practical guidelines for assessing convergence:

* For models with many parameters, it is impractical to check convergence for every parameter, so just choose a selection of relevant parameters to monitor. For example, rather than checking convergence for every element of a vector of random effects, just choose a random subset (say, the first 5 or 10).
* Examine trace plots of the sample values versus iteration to look for evidence of when the simulation appears to have stabilised:

To obtain 'live' trace plots for a parameter:

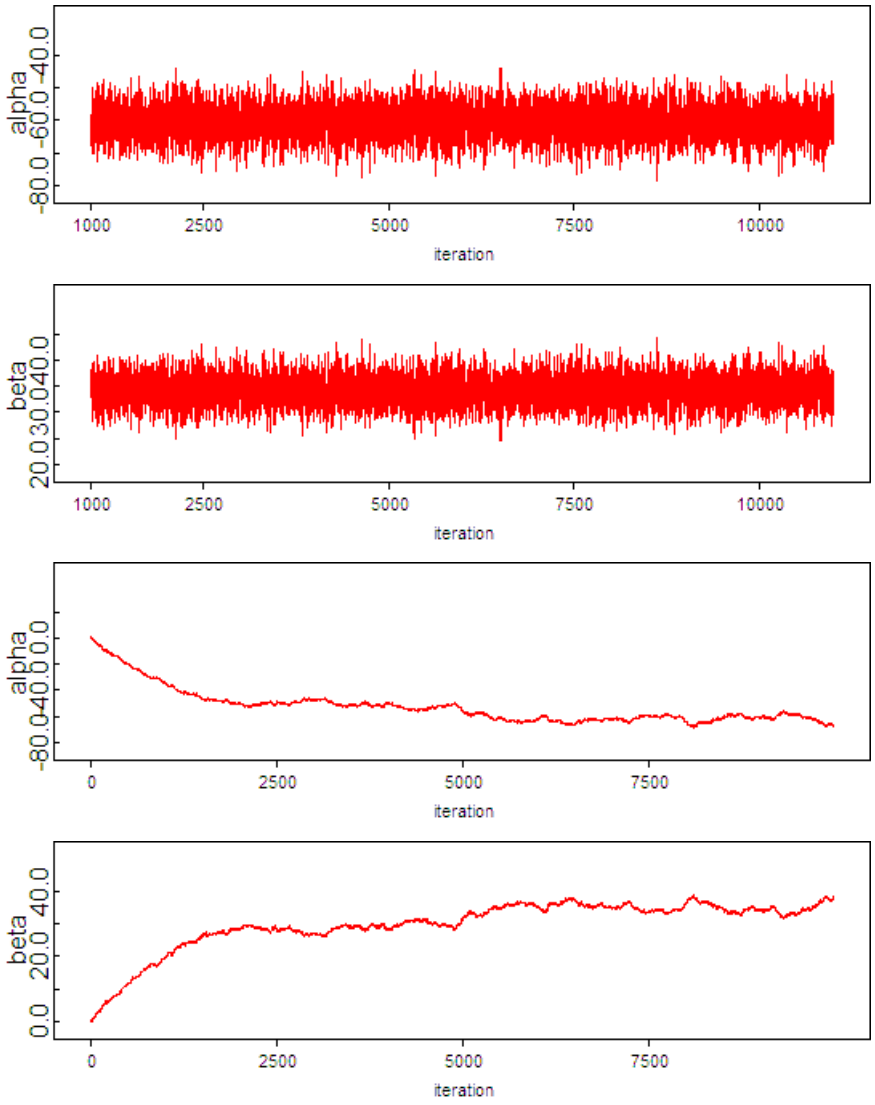
- Select Samples... from the Inference menu.
- Type the name of the parameter in the white box marked node .
- Click once with the LMB on the button marked trace : an empty graphics window will appear on screen.

- Repeat for each parameter of interest.
- Once you start running the simulations (using the Update... tool from the Model menu), trace plots for these parameters will appear 'live' in the graphics windows.

To obtain a trace plot showing the full history of the samples *for any parameter for which you have previously set a samples monitor and carried out some updates* :

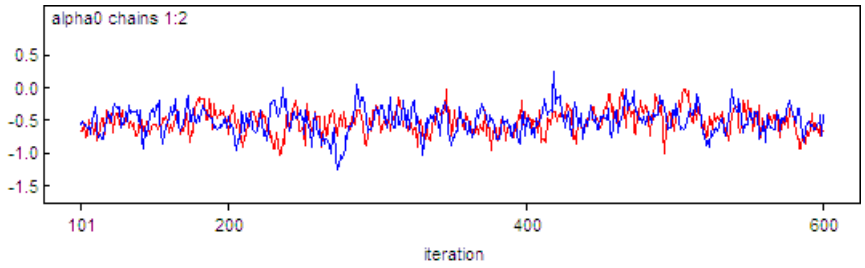
- Select Samples... from the Inference menu.
- Type the name of the parameter in the white box marked node (or select the name from the pull down list of currently monitored nodes - click once with the LMB on the downward-facing arrowhead to the immediate right of the node field).
- Click once with the LMB on the button marked history : a graphics window showing the sample trace will appear.
- Repeat for each parameter of interest.

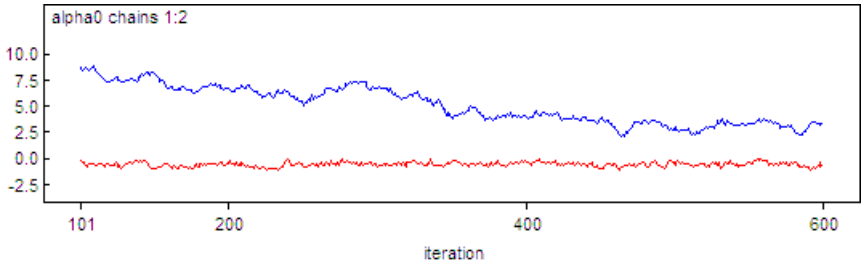
The following plots are examples of: (i) chains for which convergence (in the pragmatic sense) looks reasonable (top two plots); and (ii) chains which have clearly not reached convergence (bottom two plots).



If you are running more than one chain simultaneously, the trace and history plots will show each chain in a different colour. In this case, we can be reasonably confident that convergence has been achieved if all the chains appear to be overlapping one another.

The following plots are examples of: (i) multiple chains for which convergence looks reasonable (top); and (ii) multiple chains which have clearly not reached convergence (bottom).





For a more formal approach to convergence diagnosis the software also provides an implementation (see here) of the techniques described in Brooks & Gelman (1998) , and a facility for outputting monitored samples in a format that is compatible with the CODA software - see here.

How many iterations after convergence? [top]

Once you are happy that convergence has been achieved, you will need to run the simulation for a further number of iterations to obtain samples that can be used for posterior inference. The more samples you save, the more accurate will be your posterior estimates.

One way to assess the accuracy of the posterior estimates is by calculating the Monte Carlo error for each parameter. This is an estimate of the difference between the mean of the sampled values (which we are using as our estimate of the posterior mean for each parameter) and the true posterior mean.

As a rule of thumb, the simulation should be run until the Monte Carlo error for each parameter of interest is less than about 5% of the sample standard deviation. The Monte Carlo error (MC error) and sample standard deviation (SD) are reported in the summary statistics table (see the next section) .

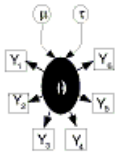
Obtaining summaries of the posterior distribution [top]

The posterior samples may be summarised either graphically, e.g. by kernel density plots, or numerically, by calculating summary statistics such as the mean, variance and quantiles of the sample.

To obtain summaries of the monitored samples, to be used for posterior inference:

* Select Samples... from the Inference menu.
* Type the name of the parameter in the white box marked node (or select the name from the pull down list, or ty pe "*" (star) to select all monitored parameters). * Type the iteration number that you want to start your summary from in the white box marked beg : this allows the pre-convergence 'burn-in' samples to be discarded.
* Click once with the LMB on the button marked stats : a table reporting various summary statistics based on the sampled values of the selected parameter will appear.
* Click once with the LMB on the button marked density : a window showing kernel density plots based on the sampled values of the selected parameter will appear.

Please see the manual entry Samples... for a detailed description of the Sample Monitor Tool dialog box (i.e. that which appears when Samples... is selected from the Inference menu).



Compound Documents

Contents

- What is a compound document?
- Working with compound documents
- Editing compound documents
- Compound documents and e-mail
- Printing compound documents
- Reading in plain text files
- Compound documents and Linux

What is a compound document? [top]

A compound document contains various types of information (formatted text, tables, formulae, plots, graphs etc) displayed in a single window and stored in a single file. The tools needed to create and manipulate these information types are always available, so there is no need to continuously move between different programs. The *OpenBUGS* software has been designed so that it produces output directly to a compound document and can get its input directly from a compound document. To see an example of a compound document click here. *OpenBUGS* is written in Component Pascal using the BlackBox development framework: see http://www.oberon.ch.

In *OpenBUGS* a document can be a description of a statistical analysis, the user interface to the software, and the resulting output.

Compound documents are stored in binary files with the .odc extension.

Working with compound documents [top]

A compound document is like a word-processor document that contains special rectangular embedded regions or elements, each of which can be manipulated by standard word-processing tools -- each rectangle behaves like a single large character, and can be focused, selected, moved, copied, deleted etc. If an element is "focused" the tools to manipulate its interior become available.

The *OpenBUGS* software works with many different types of elements, the most interesting of which are Doodles, which allow statistical models to be described in terms of graphs. *DoodleBUGS* is a specialised graphics editor and is described fully in
DoodleBUGS: The Doodle Editor . Other elements are rather simpler and are used to display plots of an analysis.

Editing compound documents [top]

OpenBUGS contains a built-in word processor, which can be used to manipulate any output produced by the software. If a more powerful editing tool is needed *OpenBUGS* documents or parts of them can be pasted into a standard OLE enabled word processor.

Each open document either displays a caret or a selection of highlighted text. New keyboard input is inserted at the caret position. Text is selected by holding down the left mouse button while dragging the mouse over a region of text. A single word of text can be selected by double clicking on it. Warning: if selection of text is highlighted and a key pressed the selection will be replaced by the character typed. (This can be un-done by selecting Undo from the Edit menu.) . The text can be unselected by pressing the "Esc" key or clicking the mouse outside the highlighted selection.

A single embedded element can be selected by single clicking into it with the left mouse button. A selected element is distinguished by a thin bounding rectangle. If this bounding rectangle contains small solid squares at the corners and mid sides it can be resized by dragging these with the mouse. An embedded element can be "focused" by double clicking into it with the left mouse button. A focused element is distinguished by a hairy grey bounding rectangle.

A selection can be moved to a new position by dragging it with the mouse. To copy the selection hold down the "control" key down while releasing the mouse button.

A selection can be cut to the Windows clip board by typing Cntrl + X or copied to the clip board by typing Cntrl + C. The contents of the clip board can be inserted at the current caret position by typing Cntl + V. These three operations can also be performed using options in the Edit menu.

These operations work across windows and across applications, and so the problem specification and the output can both be gathered into a single document. This can then be copied into another word-processor or presentation package if desired.

The style, size, font and colour of selected text can be changed using the Attributes menu. The vertical offset of the selection can be changed using the Text menu.

The formatting of text can be altered by embedding special elements. The most common format control is the ruler: pick the Show Marks option in the Text menu to see what rulers look like. The small black up-pointing triangles are tab stops, which can be moved by dragging them with the mouse and removed by dragging them outside the left or right borders of the ruler. The icons above the scale control, for example, centering and page breaks.

Compound documents and e-mail [top]

OpenBUGS compound documents contain non-ascii characters, but the Encode Document command in the Tools menu produces an ascii representation of the focus document. The original document can be recovered from this encoded form by using the Decode command of the Tools menu. This allows, for example, Doodles to be sent by e-mail.

Printing compound documents [top]

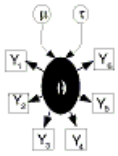
These can be printed directly from the File Print... menu. We find it useful to have printing set up to produce pdf files. CutePDF is a freely available Windows PDF printer that does this.

Reading in plain text files [top]

Open these using the File Open... menu option and pick 'txt' as the file type in the dialog box. The contents of text files can be copied into documents, or text files can be converted into odc documents by using the File Save As... menu option and picking 'odc' as the file type in the dialog box.

Compound documents and Linux [top]

The OpenBUGS implementation for Intel Linux machines accepts only ASCII file input for all input types, and all output is in ASCII format. Graphic output is not available in the Linux version of the program.



The BUGS Model Specification Language

Contents

- Graphical models
- Graphs as a formal language
- Stochastic nodes
- Logical nodes
- Arrays and indexing
- Repeated structures
- Data transformations
- Nested indexing and mixtures
- Formatting of data
- Appendix I Distributions
- Appendix II Functions and Functionals

Graphical models [top]

We strongly recommend that the first step in any analysis should be the construction of a *directed graphical model* . Briefly, this represents all quantities as nodes in a directed graph, in which arrows run to nodes from their direct influences (parents). The model represents the assumption that, given its parent nodes pa[*v*], each node *v* is independent of all other nodes in the graph except descendants of *v* , where descendant has the obvious definition.

Nodes in the graph are of three types.

1. *Constants* are fixed by the design of the study: they are always founder nodes (*i.e.* do not have parents), and are denoted as rectangles in the graph.

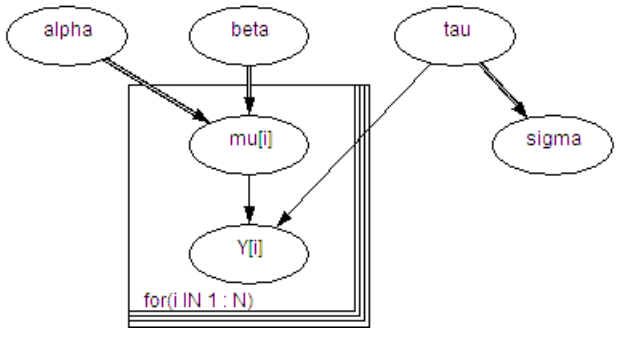
2. *Stochastic node* s are variables that are given a distribution, and are denoted as ellipses in the graph; they may be parents or children (or both). Stochastic nodes may be observed in which case they are *data* , or may be unobserved and hence be *parameters* , which may be unknown quantities underlying a model, observations on an individual case that are unobserved say due to censoring, or simply missing data.

3. *Deterministic nodes* are logical functions of other nodes.

Quantities are specified to be data by giving them values in a data declaration. Values for constants can also specified as data.

Directed links may be of two types: a thin solid arrow indicates a stochastic dependence while a thick hollow arrow indicates a logical function. An undirected dashed link may also be drawn to represent an upper or lower bound for a stochastic node.

Repeated parts of the graph can be represented using a 'plate', as shown below for the range `(i in 1:N)`.



A simple graphical model, where `Y[i]` depends on `mu[i]` and `tau` , with `mu[i]` being a logical function of `alpha` and `beta` .

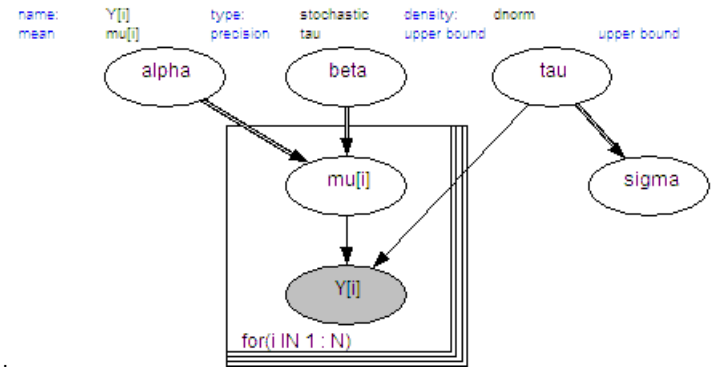
The conditional independence assumptions represented by the graph mean that the full joint distribution of all quantities *V* has a simple factorisation in terms of the conditional distribution p(*v* | parents[*v*]) of each node given its parents, so that

$$p(V) = \prod p(v \mid \text{parents}[v]) \qquad v \text{ in } V$$

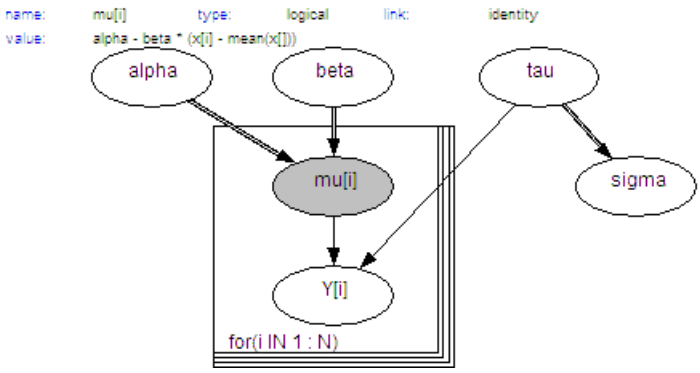
The crucial idea is that we need only provide the parent-child distributions in order to fully specify the model, and *OpenBUGS* then sorts out the necessary sampling methods directly from the expressed graphical structure.

Graphs as a formal language [top]

A special drawing tool DoodleBUGS has been developed for specifying graphical models, which uses a hyper-diagram approach to add extra information to the graph to give a complete model specification. Each stochastic and logical node in the graph must be given a name using the conventions explained in Creating a node .



The shaded node `Y[i]` is normally distributed with mean `mu[i]` and precision `tau` .



The shaded node `mu[i]` is a logical function of `alpha` , `beta` , and the constants `x` . (`x` is not required to be shown in the graph).

The value function of a logical node contains all the necessary information to define the logical node.

As an alternative to the Doodle representation, the model can be specified using the text-based *BUGS* language, headed by the model statement:

```
model {
 text-based description of graph in BUGS language
}
```

The BUGS language: stochastic nodes [top]

In the text-based model description, stochastic nodes are represented by the node name followed by a tilda symbol followed by the distribution name followed by a comma-separated list of parents enclosed in brackets, *e.g.*

```
x ~ dnorm(mu, tau)
```

The distributions that can be used in *OpenBUGS* are described in Appendix I Distributions . The parameters of a distribution must be explicit nodes in the graph (scalar parameters can also be numerical constants) and so may not be expressions.

Multivariate nodes must form contiguous elements in an array. Since the final element in an array changes fastest, such nodes must be defined as the final part of any array. For example, to define a set containing I multivariate normal variables of dimensional K as a single multidimensional array `x[i, j]` , we could write:

```
for (i in 1 : I) {
   x[i, 1 : K] ~ dmnorm(mu[], tau[ , ])
}
```

Data defined by a multivariate distribution must not contain missing (unobserved) values. The only exception to this rule is the multivariate normal distribution. We realise this is an unfortunate restriction and we hope to relax it in the future. For multinomial data, it may be possible to get round this problem by re-expressing the multivariate likelihood as a sequence of conditional univariate binomial distributions or as Poisson distributions.

Censoring is denoted using the notation `C(lower, upper)` *e.g.*

```
x ~ dnorm(mu, tau)C(lower, upper)
```

would denote a quantity x from the normal distribution with parameters mu, tau, which had been observed to lie between lower and upper. Leaving either lower or upper blank corresponds to no limit, e.g. C(lower,) corresponds to an observation known to lie above lower. Whenever censoring is specified the censored node contributes a term to the full conditional distribution of its parents. This structure is only of use if x has not been observed (if x is observed then the constraints will be ignored). In general multivariate nodes can not be censored, the multivariate normal distribution is exempted from this restriction.

Truncation is denoted by using the notation `T(lower, upper)`, *eg* .

```
x ~ dnorm(mu, tau)T(lower, upper)
```

would denote a quantity x from the modified normal distribution normalized by dividing by the integral of the distribution between limits lower and upper, which lies between lower and upper. Leaving either lower or upper blank corresponds to no limit, e.g. T(lower,) corresponds to an observation known to lie above lower. x can either be observed or unobserved.

It is also important to note that if x, mu, tau , lower and upper are all unobserved, then lower and upper must not be functions of mu and tau.

Nodes with a discrete distribution must in general have integer values. Observed variables having a binomial or Poisson distribution are exempt from this restriction.

Certain parameters of distributions must be constants, that is they can not be learnt. These include both parameters of the Wishart distributions, the order (N) of the multinomial distribution and the threshold (mu) of the generalized Pareto distribution.

The precision matrices for multivariate normals must be positive definite. If a Wishart prior is not used for the precision matrix, then the elements of the precision matrix are updated univariately without any check of positive-definiteness. This will result in a crash unless the precision matrix is parameterised appropriately. **This is the user's responsibility!** Forming the precision matrix from the product of cholesky factors will ensure positive definiteness.

The BUGS language: logical nodes [top]

Logical nodes are represented by the node name followed by a left pointing arrow followed by a logical expression of its parent nodes e.g.

```
mu[i] <- beta0 + beta1 * z1[i] + beta2 * z2[i] + b[i]
```

Logical expressions can be built using the following operators: plus (`A + B`), multiplication (`A * B`), minus (`A - B`), division (`A / B`) and unitary minus (`-A`). The scalar valued functions in

Appendix II Functions and Functionals can also be used in logical expressions. A vector-valued logical function can only be used as the sole term on the right hand side of a vector-valued logical relation.

A link function can also be specified acting on the left hand side of a logical node *e.g.*

```
logit(mu[i]) <- beta0 + beta1 * z1[i] + beta2 * z2[i] + b[i]
```

The following functions can be used on the left hand side of logical nodes as link functions: `log` , `logit` , `cloglog` , and `probit` (where `probit(x) <- y` is equivalent to `x <- phi(y)`).

A special logical node called "deviance" is created automatically by *OpenBUGS* : It calculates -2 * log(likelihood), where 'likelihood' is the conditional probability of all data nodes given their stochastic parent nodes. This node can be monitored, and is used in the DIC tool - see DIC...

Arrays and indexing [top]

Arrays are indexed by terms within square brackets. The four basic operators +, -, *, and / along with appropriate bracketing are allowed to calculate an integer function as an index, for example:

```
Y[(i + j) * k, l]
```

On the left-hand-side of a relation, an expression that always evaluates to a fixed value is allowed for an index, whether it is a constant or a function of data. On the right-hand-side the index can be a fixed value or a named node, which allows a straightforward formulation for mixture models in which the appropriate element of an array is 'picked' according to a random quantity (see Nested indexing and mixtures). However, functions of unobserved nodes are not permitted to appear directly as an index term (intermediate deterministic nodes may be introduced if such functions are required).

The conventions broadly follow those of S-Plus:

`n : m` represents n , n + 1, ..., m .
`x[ ]` represents all values of a vector x .
`y[, 3]` indicates all values of the third column of a two-dimensional array y .

Multidimensional arrays are handled as one-dimensional arrays with a constructed index. Thus functions defined on arrays must be over equally spaced nodes within an array: for example `sum(y[i, 1:4, k])` .

When dealing with unbalanced or hierarchical data a number of different approaches are possible - see Handling unbalanced datasets. The ideas discussed in Nested indexing and mixtures may also be helpful in this respect.

Repeated structures [top]

Repeated structures are specified using a "for - loop". The syntax for this is:

```
for (i in a : b) {
```
list of statements to be repeated for increasing values of loop-variable i
```
}
```

Note that a and b must either be explicit (such as `for (i in 1:100)`) or supplied as data. Neither a nor b may be logical nodes (such as `b <- 100`) or stochastic nodes - see here for a possible way to get round this.

Data transformations [top]

Although transformations of data can always be carried out before using *OpenBUGS* , it is convenient to be able to try various transformations of dependent variables within a model description. For example, we may wish to try both `y` and `sqrt(y)` as dependent variables without creating a separate variable `z = sqrt(y)` in the data file.

The BUGS language therefore permits the following type of structure to occur:

```
for (i in 1:N) {
   z[i] <- sqrt(y[i])
   z[i] ~ dnorm(mu, tau)
```

```
}
```

Strictly speaking, this goes against the declarative structure of the model specification, with the accompanying exhortation to construct a directed graph and then to make sure that each node appears once and only once on the left-hand side of a statement. However, a check has been built in so that, when finding a logical node which also features as a stochastic node (such as z above), a stochastic node is created with the calculated values as fixed data.

We emphasise that this construction is only possible when transforming observed data (not a function of data and parameters) with no missing values.

This construction is particularly useful in Cox modelling and other circumstances where fairly complex functions of data need to be used. It is preferable for clarity to place the transformation statements in a section at the beginning of the model specification, so that the essential model description can be examined separately. See the `Leuk` and `Endo` examples.

Nested indexing and mixtures [top]

Nested indexing can be very effective. For example, suppose `N` individuals can each be in one of `I` groups, and `g[1:N]` is a vector which contains the group membership. Then "group" coefficients `beta[i]` can be fitted using `beta[g[j]]` in a regression equation.

In the *BUGS* language, nested indexing can be used for the parameters of distributions: for example, the `Eyes` example concerns a normal mixture in which the i^{th} case is in an unknown group T_i which determines the mean λ_{Ti} of the measurement y_i. Hence the model is

$T_i \sim$ Categorical(P)
$y_i \sim$ Normal(λ_{Ti}, τ)

which may be written in the BUGS language as

```
for (i in 1:N) {
   T[i] ~ dcat(P[])
   y[i] ~ dnorm(lambda[T[i]], tau)
}
```

The mixture construct can also be applied to vector value parameters eg

```
for (i in 1 : ns){
   nbiops[i] <- sum(biopsies[i, ])
   true[i] ~ dcat(p[])
   biopsies[i, 1 : 4] ~ dmulti(error[true[i], ], nbiops[i])
}
```

Multiple (up to four) variable indices are allowed in setting up mixture models. eg

```
dyspnoea ~ dcat(p.dyspnoea[either,bronchitis,1:2])
either <- max(tuberculosis,lung.cancer)
bronchitis ~ dcat(p.bronchitis[smoking,1:2])
```

Formatting of data [top]

Data can be in an R/S-Plus format (used by R and S-Plus and most of the examples in OpenBUGS) or, for data in arrays, in rectangular format.

Missing values are represented as `NA` .

The whole array must be specified in the file - it is not possible just to specify selected components. Any parts of the array you do not want to specify must be filled with NAs.

All variables in a data file must be defined in a model, even if just left unattached to the rest of the model.

R/S-Plus format: This allows scalars and arrays to be named and given values in a single structure headed by key-word list. There must be no space after list.

For example, in the Rats example, we need to specify a scalar `xbar` , dimensions `N` and `T` , a vector `x` and a two-dimensional array `Y` with 30 rows and 5 columns. This is achieved using the following format:

```
list(
      xbar=22, N=30, T=5,
      x=c(8.0, 15.0, 22.0, 29.0, 36.0),
      Y=structure(
               .Data=c(
                     151, 199, 246, 283, 320,
                     145, 199, 249, 293, 354,
                     ...
                     ...
                     137, 180, 219, 258, 291,
                     153, 200, 244, 286, 324),
               .Dim=c(30, 5)
            )
      )
```

OpenBUGS reads data into an array by filling the right-most index first, whereas the R/S-Plus program fills the left-most index first. Hence *OpenBUGS* reads the string of numbers `c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)` into a 2 * 5 dimensional matrix in the order

```
[i, j]th element of matrix      value
 [1, 1]                                  1
 [1, 2]                                  2
 [1, 3]                                  3
 ...                                     ...
 [1, 5]                                  5
 [2, 1]                                  6
 ...                                     ...
 [2, 5]                                  10
```

whereas R/S-Plus read the same string of numbers in the order

```
[i, j]th element of matrix      value
 [1, 1]                                  1
 [2, 1]                                  2
 [1, 2]                                  3
 ...                                     ...
 [1, 3]                                  5
 [2, 3]                                  6
 ...                                     ...
 [2, 5]                                  10
```

Hence the ordering of the array dimensions must be reversed before using the R/S-Plus `dput` function to create a data file for input into *OpenBUGS* .

For example, consider the 2 * 5 dimensional matrix

```
1   2   3   4   5
 6   7   8   9   10
```

This must be stored in R/S-Plus as a 5 * 2 dimensional matrix:

```
> M
[,1] [,2]
[1,] 1 6
[2,] 2 7
[3,] 3 8
[4,] 4 9
[5,] 5 10
```

The R/S-Plus command

```
> dput(list(M=M), file="matrix.dat")
```

will then produce the following data file

```
list(M=structure(.Data=c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10),
                         .Dim=c(5,2))
```

Edit the `.Dim` statement in this file from `.Dim=c(5,2)` to `.Dim=c(2,5)` . The file is now in the correct format to input the required 2 * 5 dimensional matrix into *OpenBUGS* .

Now consider a 3 * 2 * 4 dimensional array

```
1   2   3   4
5   6   7   8

9   10   11   12
13   14   15   16

17   18   19   20
21   22   23   24
```

This must be stored in R/S-Plus as the 4 * 2 * 3 dimensional array:

```
> A
, , 1
[,1] [,2]
[1,] 1 5
[2,] 2 6
[3,] 3 7
[4,] 4 8

, , 2
[,1] [,2]
[1,] 9 13
[2,] 10 14
[3,] 11 15
[4,] 12 16

, , 3
[,1] [,2]
[1,] 17 21
[2,] 18 22
[3,] 19 23
[4,] 20 24
```

The command

```
> dput(list(A=A), file="array.dat")
```

will then produce the following data file

```
list(A=structure(.Data=c( 1, 2, 3, 4, 5, 6, 7, 8,
                                    9, 10, 11, 12, 13, 14, 15, 16,
                                    17, 18, 19, 20, 21, 22, 23, 24),
                         .Dim=c(4,2,3))
```

Edit the `.Dim` statement in this file from `.Dim=c(4,2,3)` to `.Dim=c(3,2,4)` . The file is now in the correct format to input the required 3 * 2 * 4 dimensional array into *OpenBUGS* in the order

```
[i, j, k]th element of matrix      value
[1, 1, 1]                            1
[1, 1, 2]                            2
...                                    ...
[1, 1, 4]                            4
[1, 2, 1]                            5
[1, 2, 2]                            6
...                                    ...
[2, 1, 3]                            11
[2, 1, 4]                            12
```

```
[2, 2, 1]                                   13
[2, 2, 2]                                   14
...
[3, 2, 3]                                   23
[3, 2, 4]                                   24
```

Rectangular format: The columns for data in rectangular format need to be headed by the array name. The arrays need to be of equal size, and the array names must have explicit brackets: for example:

```
age[]   sex[]
26   0
52   1
...
34   0
END
```

Note that the file must end with an 'END' keyword, as shown above and below, and this must be followed by at least one blank line.

Multi-dimensional arrays can be specified by explicit indexing: for example, the Ratsy file begins

```
Y[,1]   Y[,2]   Y[,3]   Y[,4]   Y[,5]
151   199   246   283   320
145   199   249   293   354
147   214   263   312   328
...
153   200   244   286   324
END
```

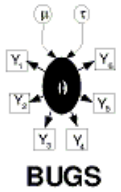
The first index position for any array must always be empty.

It is possible to load a mixture of rectangular and R/S-Plus format data files for the same model. For example, if data arrays are provided in a rectangular file, constants can be defined in a separate list statement (see also the Rats example with data files Ratsx and Ratsy).

(See here for details of how to handle unbalanced data.)

Note that programs exist for conversion of data from other packages: please see the OpenBUGS resources at

http://www.openbugs.info



Appendix I Distributions

Contents [top]

Discrete Univariate
- Bernoulli
- Binomial
- Categorical
- Negative Binomial
- Poisson
- Geometric
- Geometric (alternative)
- Non-central Hypergeometric

Continuous Univariate
- Beta
- Chi-squared

- Double Exponential
- Exponential
- Flat
- Gamma
- Generalized Extreme Value
- Generalized F
- Generalized Gamma
- Generalized Pareto
- Generic LogLikelihood Distribution
- Log-normal
- Logistic
- Normal
- Pareto
- Student-t
- Uniform
- Weibull

Discrete Multivariate
- Multinomial

Continuous Multivariate
- Dirichlet
- Multivariate Normal
- Multivariate Student-t
- Wishart

Discrete Univariate [top] [top appendix i]

Bernoulli

`r ~ dbern(p)` $p^r(1-p)^{1-r}; \quad r = 0, 1$

Binomial

`r ~ dbin(p, n)` $\frac{n!}{r!(n-r)!}p^r(1-p)^{n-r}; \quad r = 0, \ldots, n$

Categorical

`r ~ dcat(p[])` $p[r]; \quad r = 1, 2, \ldots, \dim(p); \quad \sum_i p[i] = 1$

Negative Binomial

`x ~ dnegbin(p, r)` $\frac{(x+r-1)!}{x!(r-1)!}p^r(1-p)^x; \quad x = 0, 1, 2, \ldots$

Poisson

`r ~ dpois(lambda)` $e^{-\lambda}\frac{\lambda^r}{r!}; \quad r = 0, 1, \ldots$

Geometric

`r ~ dgeom(p)` $(1-p)^{r-1}p; \qquad r = 1, 2, \ldots$

Geometric (alternative form)

`r ~ dgeom0(p)` $(1-p)^r p; \qquad r = 0, 1, \ldots$

Non-central Hypergeometric

`x ~ dhyper(n, m, N, psi)`

$$\frac{\binom{n}{x}\binom{N-n}{m-x}\psi^x}{\sum_{u=u_0}^{u_1}\binom{n}{u}\binom{N-n}{m-u}\psi^u}$$

$$u_0 = max(0, m-N+n);$$

$$u_1 = min(n, m);\ u_0 <= x <= u_1$$

Continuous Univariate [top] [top appendix i]

Beta

`p ~ dbeta(a, b)` $p^{a-1}(1-p)^{b-1}\frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)}; \quad 0 < p < 1$

Chi-squared

`x ~ dchisqr(k)` $\frac{2^{-k/2}x^{k/2-1}e^{-x/2}}{\Gamma(\frac{k}{2})}; \quad x > 0$

Double Exponential

`x ~ ddexp(mu, tau)` $\frac{\tau}{2}\exp(-\tau\,|x-\mu|); \quad -\infty < x < \infty$

Exponential

`x ~ dexp(lambda)` $\lambda e^{-\lambda x}; \quad x > 0$

Flat

`x ~ dflat()` constant value for all x; not a proper distribution

Gamma

`x ~ dgamma(r, mu)` $\frac{\mu^r x^{r-1}e^{-\mu x}}{\Gamma(r)}; \quad x > 0$

Generalized Extreme Value

`x ~ dgev(mu,sigma,eta)` $\frac{1}{\sigma}\left(1+\frac{\eta}{\sigma}(x-\mu)\right)^{-(1+\frac{1}{\eta})}\exp\left\{-\left(1+\frac{\eta}{\sigma}(x-\mu)\right)^{-\frac{1}{\eta}}\right\};$ $\frac{\eta}{\sigma}(x-\mu) \geq -1$

Generalized F

`x ~ df(n,m,mu,tau)`

$$\frac{\Gamma(\frac{n+m}{2})}{\Gamma(\frac{n}{2})\Gamma(\frac{m}{2})}\left(\frac{n}{m}\right)^{\frac{n}{2}}(\sqrt{\tau}(x-\mu))^{\frac{n}{2}-1}\sqrt{\tau}\left\{1+\frac{n\sqrt{\tau}(x-\mu)}{m}\right\}^{-\frac{(n+m)}{2}}$$

$x > 0 \qquad n, m > 0$

Reduces to the standard F for mu=0, tau=1.

Generalized Gamma

`x ~ dggamma(r, mu, beta)` $\frac{\beta}{\Gamma(r)}\mu^{\beta r}x^{\beta r-1}\exp\left[-(\mu x)^{\beta}\right]; \quad x > 0$

Generalized Pareto

`x ~ dgpar(mu,sigma,eta)` $\frac{1}{\sigma}\left(1+\frac{\eta}{\sigma}(x-\mu)\right)^{-(1+\frac{1}{\eta})};$ $\frac{\eta}{\sigma}(x-\mu) \geq -1; \quad x \geq \mu$

Generic LogLikelihood distribution

`x ~dloglik(lambda)` exp(lambda); NB does not depend on x. See Generic sampling distributions.

Log-normal

`x ~ dlnorm(mu, tau)` $\sqrt{\frac{\tau}{2\pi}}\frac{1}{x}\exp\left(-\frac{\tau}{2}(\log x-\mu)^2\right); \quad x > 0$

Logistic

`x ~ dlogis(mu, tau)` $\frac{\tau \exp(\tau(x-\mu))}{(1+\exp(\tau(x-\mu)))^2}; \quad -\infty < x < \infty$

Normal

`x ~ dnorm(mu, tau)` $\sqrt{\frac{\tau}{2\pi}} \exp\left(-\frac{\tau}{2}(x-\mu)^2\right); \quad -\infty < x < \infty$

Pareto

`x ~ dpar(alpha, c)` $\alpha c^{\alpha} x^{-(\alpha+1)}; \quad x > c$

Student-t

`x ~ dt(mu, tau, k)` $\frac{\Gamma\left(\frac{k+1}{2}\right)}{\Gamma\left(\frac{k}{2}\right)} \sqrt{\frac{\tau}{k\pi}} (1+\frac{\tau}{k}(x-\mu)^2)^{-(k+1)/2}; \quad -\infty < x < \infty; \quad k \geq 1$

Uniform

`x ~ dunif(a, b)` $\frac{1}{b-a}; \quad a < x < b$

Weibull

`x ~ dweib(v, lambda)` $v\lambda x^{v-1} \exp(-\lambda x^v); \quad x > 0$

Discrete Multivariate [top] [top appendix i]

Multinomial

`x[] ~ dmulti(p[], N)` $\frac{(\sum_i x_i)!}{\prod_i x_i!} \prod_i p_i^{x_i}; \quad \sum_i x_i = N; \quad 0 < p_i < 1; \quad \sum_i p_i = 1$

Continuous Multivariate [top] [top appendix i]

Dirichlet

`p[] ~ ddirich(alpha[])` $\frac{\Gamma(\sum_i \alpha_i)}{\prod_i \Gamma(\alpha_i)} \prod_i p_i^{\alpha_i - 1}; \quad 0 < p_i < 1; \quad \sum_i p_i = 1$

May also be spelt `ddirch` as in WinBUGS.

Multivariate Normal

`x[] ~ dmnorm(mu[], T[,])`

Multivariate Student-t

`x[] ~ dmt(mu[], T[,], k)`

Wishart

`x[,] ~ dwish(R[,], k)`

Appendix II Functions and Functionals

Function arguments represented by **e** can be expressions , those by **s** must be scalar-valued nodes in the graph and those represented by **v** must be vector-valued nodes in a graph. Some function arguments must be stochastic nodes. Functionals

are described using a similar notation to functions, the special notation `F(x)` is used to describe the function on which the functional acts. See example Functionals for details. Systems of ordinary differential equations and their solution can be described in the BUGS language by using the special `D(x[1:n], t)` notation. See example ode for details.

Scalar functions [top]

`abs(e)` absolute value of e, |e|

`arccos(e)` inverse cosine of e

`arccosh(e)` inverse hyperbolic cosine of e

`arcsin(e)` inverse sine of e

`arcsinh(e)` inverse hyperbolic sine of e

`arctan(e)` inverse tangent of e

`arctanh(e)` inverse hyperbolic tangent of e

`cloglog(e)` complementary log log of e, ln(-ln(1 - *e*))

`cos(e)` cosine of e

`cosh(e)` hyperbolic cosine of e

`cumulative(s1, s2)` tail area of distribution of s1 up to the value of s2, s1 must be stochastic, s1 and s2 can be the same

`cut(e)` cuts edges in the graph - see Use of the "cut" function

`density(s1, s2)` density of distribution of s1 at value of s2, s1 must be a stochastic node supplied as data, s1 and s2 can be the same.

`deviance(s1, s2)` deviance of distribution of s1 at value of s2, s1 must be a stochastic node supplied as data, s1 and s2 can be the same.

`equals(e1, e2)` 1 if *value of e* 1 equals value of *e* 2; 0 otherwise

`exp(e)` exp(*e*)

`gammap(s1, s2)` partial (incomplete) gamma function, value of standard gamma density with parameter s1 integrated up to s2

`ilogit(e)` exp(*e)* / (1 + exp(*e*))

`icloglog(e)` 1 - exp(- exp(*e*))

`integral(F(s), s1, s2, s3)` definite integral of function F(s) between s = s1 and s = s2 to accuracy s3

`log(e)` natural logarithm of e

`logfact(e)` ln(*e* !)

`loggam(e)` logarithm of gamma function of e

`logit(e)` ln(*e* / (1 - *e*))

`max(e1, e2)` e1 if e1 > e2; e2 otherwise

`min(e1, e2)` e1 if e1 < e2; e2 otherwise

`phi(e)` standard normal cdf

`post.p.value(s)` s must be a stochastic node, returns one if a sample from the prior is less than the value of s.

`pow(e1, e2)` $e1^{e2}$

`prior.p.value(s)` s must be a stochastic node, returns one if a sample from the prior after resampling its stochastic parents is less than value of s.

`probit(e)` inverse of `phi(e)`

`replicate.post(s)` replicate from distribution of s, s must be stochastic node

`replicate.prior(s)` replicate from distribution of s after replicating from it parents if they are stochastic, s must be stochastic node

`round(e)` nearest integer to e

`sin(e)` sine of e

`sinh(e)` hyperbolic sine of e

`solution(F(s), s1, s2, s3)`
a solution of equation F(s) = 0 lying between s = s1 and s = s2 to accuracy s3, s1 and s2 must bracket a solution

`sqrt(e)` $e^{1/2}$

`step(e)` 1 if *e* >= 0; 0 otherwise

`tan(e)` tangent of e

`tanh(e)` hyperbolic tangent of e

`trunc(e)` greatest integer less than or equal to e

Vector functions [top]

`inprod(v1, v2)` inner product of v1 and v2, $\sum_i v1_i v2_i$

`interp.lin(e, v1, v2)`
$v2_p + (v2_{p+1} - v2_p) * (e - v1_p) / (v1_{p+1} - v1_p)$
where the elements of $v1$ are in ascending order and p is such that $v1_p < e < v1_{p+1}$.

Given function values in the vector v2 evaluated at the points in v1, this estimates the function value at a new point e by simple linear interpolation using the closest bounding pair of points. For example, given the population in 1991,2001 and 2011, we might want to estimate the population in 2004.

`inverse(v)` inverse of symmetric positive-definite matrix v

`logdet(v)` log of determinant of v for symmetric positive-definite *v*

`mean(v)` $\sum_i v_i / n$ n = dim(v)

`eigen.vals(v)` eigenvalues of matrix v

`ode(v1, v2, D(v3, s1), s2, s3)`

solution of system of ordinary differential equations at grid of points v2 given initial values v1 at time s2 solved to accuracy s3. v3 is a vector of components of the system of ode and s1 is the time variable. See the PDF files in the `Diff/Docu` directory of the OpenBUGS installation for further details.

`prod(v)` $\prod_i v_i$

`p.valueM(v)` v must be a multivariate stochastic node, returns a vector of ones and zeros depending on if a sample from the prior is less than value of the corresponding component of v

`rank(v, s)` number of components of vless than or equal to s

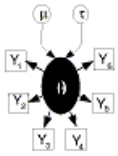
`ranked(v, s)` the s^{th} smallest component of v

`replicate.postM(v)` replicate from multivariate distribution of v, v must be stochastic and multivariate

`sd(v)` standard deviation of components of *v* (n - 1 in denominator)

`sort(v)` vector v sorted in ascending order

`sum(v)` $\sum_i v_i$



Scripts and Batch-mode

Scripting

As an alternative to the menu / dialog box interface, scripting commands has been created that can 1) automate routine analyses, 2) create reproducible analyses, and 3) produce batch execution for simulation studies, etc. The scripting commands work in effect by writing values into fields and clicking on buttons in relevant dialog boxes. It is possible to combine use of scripting with the menu / dialog box interface.

The scripting commands are very similar to those in the R interface to OpenBUGS called BRugs. Their syntax differs from the scripting commands in WinBUGS 1.4.x, although there are corresponding WinBUGS/OpenBUGS script commands for most tasks.

To make use of the scripting commands, a minimum of three files is required: a file containing the script commands, a file containing the BUGS language representation of the model; and a file (or several) containing the data. If initial values are supplied (recommended) rather than generated by OpenBUGS, an additional file is required for each chain.

On all operating systems, directory names are separated by a forward slash (/). A backslash (\) is also allowed with Windows. Quoted input can be created using either single (') or double (") quotation marks, but the quotation mark types must match. The modelSetWD and modelGetWD script commands (Script Commands) can be used so that full path names are not required for all file names. The linux shell commands (Linux execution) offer an alternative approach for path names in linux.

Executing script commands from the GUI

A set of script commands can be executed from the GUI by selecting the window that contains them, and then selecting Script from the Model menu. All commands in the window are executed, unless some commands are highlighted, in which case only the highlighted commands are executed (a partially highlighted command will produce an error). The contents of a window with script commands does not have to be saved before it is executed. Input files may be in either native *OpenBUGS* format (.odc) or text format, in which case it must have a .txt extension.

After execution of script commands, GUI menu selections can be used to view changed settings. If a dialog box was already open when a script was executed, it may need to be closed and re-opened to ensure that its contents are refreshed. Execution of script and GUI commands can be mixed, allowing automation of tasks such as summary setting for models with numerous parameters.

Batch-mode in Windows

OpenBUGS can execute a file of script commands in Batch-mode from a programming shell using the following syntax:

"FULLPATH/OpenBUGS.exe" /PAR "FULLPATH/ScriptName.txt" /HEADLESS

It is important to add modelQuit('yes') at the end of the script file so OpenBUGS will stop execution and return control to the initiating shell when the script commands are complete. It is also important to include a modelDisplay('log') at the beginning of the script file, and modelSaveLog('filename.odc') before modelQuit as this is the only way to capture tabular and graphical output in batch-mode. (If there is no graphical output, the log file can be of type '.txt'.).

The /HEADLESS option causes OpenBUGS to run without displaying any output. Removing /HEADLESS will cause the usual GUI windows to appear during execution. If /HEADLESS is removed and the modelQuit command is omitted, the user can interact with the OpenBUGS program before returning control to the programming shell. Using modelQuit() without 'yes' will also cause OpenBUGS to display a dialog box for user input before exiting.

Input files (including the script) may be in either native *OpenBUGS* format (.odc) or text format. Text format requires a .txt

extension.

Batch-mode in Linux

OpenBUGS can be executed from the Linux OS on Intel-based machines in a native format that yields results that exactly agree with those from the Windows OS on the same hardware. The GUI, and graphical output (e.g., history plots) depend on Windows-specific code, however, and are not available with Linux execution.

A bash shell script called OpenBUGS is included in the standard Linux distribution in the /bin sub-directory. Once this directory is added to a user's path, the OpenBUGS command behaves like a standard Linux program. Typing OpenBUGS without inputs starts the program in interactive mode with screen I/O. To exit the program type "modelQuit()". To execute a script file, script.txt, in batch mode with output to file log.txt:

OpenBUGS script.txt > log.txt &
or
OpenBUGS <script.txt >log.txt &

The paths for input and output files within a script file, such as the model code or coda output, is assumed to be the current working directory unless the paths are fully specified. The script command **modelSetWD("/path/to/dir/")** can be specified to reset the working directory to /path/to/dir **.**

Additional options can be found by typing OpenBUGS --help.

Input files must be in ASCII format. The log file (ASCII format) is always written to standard output with Linux execution, so the modelSaveLog command is disabled during Linux execution.

The scripting commands

Script commands and a brief synopsis of their menu/dialog box equivalent are supplied below. The script command and corresponding menu name are in bold. The menu names are followed by their associated menu options or dialog box and potential user inputs. You can find detailed descriptions of the menu/dialog box options in the corresponding Manual entries for the Model menu, Inference menu , Info menu and File menu .

Script commands have two input types: character strings and integers. If a string argument contains non-alpha-numeric characters (including space but excluding the period) the string must be enclosed in quotes. Arguments that are required to be supplied by the user are shown in *blue italics* ; optional arguments are displayed in plain text. Optional arguments will be set to default values if they are omitted.

If the menu/dialog box equivalent of the specified script command would normally be grayed out because of inappropriate timing, for example, then the script command will not execute and an error message will be produced instead.

Commands that are preceded by an asterisk (*) , are not available with Linux execution.

The translations between commands in the script language and the underlying Component Pascal procedures implementing them in OpenBUGS are in the file Bugs/Mod/Script.odc in the OpenBUGS program directory. Explanations of the commands and their inputs are in the documentation for the menu items.

Script command description **Menu / dialog box equivalent**

modelGetWD() **None**
Returns the default path for file name input

modelSetWD(*string* **)** **None**
- *string* = Full default path for file name input

modelCheck(*string* **)** **Model** > Specification... >
- *string* = (full path to) Model file

modelData(*string* **)** **Model** > Specification... >
- *string* = (full path to) Data file

modelCompile(*int* **)** **Model** > Specification... >
- *int* = number of chains

modelInits(*string* , int **)** **Model** > Specification... >
- *string* = (full path to) Inits file
- int = for chain (default=1)
The initial values for each chain must be in separate files.

modelGenInits() **Model** > Specification... >

---

modelUpdate(*int0* , int1, int2, string**)** **Model** > Update...>
- *int0* = updates
- int1 = thin. Thinning in this way will discard the samples (c.f. the samplesThin() command)
- int2 = frequency of refreshing the display in the Windows interface
- string = "T" or "F" for over relax

---

modelSaveState(string **)** **Model** > Save State
- string = fileStem
Note that the string must be specified for Linux execution.

---

modelGetRN() **Model** > RN generator...
Returns the starting (preset) state of the random number generator

modelSetRN(int **)** **Model** > RN generator...
Set the starting (preset) state of the random number generator to 'int'. This must be an integer from 1 to 14 inclusive.

---

modelDisplay(*string* **)** **Model** > Input/Output options...
- *string* = 'window' or 'log'

modelPrecision(*int* **)** **Model** > Input/Output options...
- *int* = number of significant digits for output

---

modelDisable(*string* **)** **Model** > Updater options...
- *string* = name of updater to disable

modelEnable(*string* **)** **Model** > Updater options...
- *string* = name of updater to enable

modelSetAP(*string* , *int* **)** **Model>** Updater options...
- *string* = name of sampler
- *int* = number of iterations in the adaptive phase

modelSetIts(*string* , *int* **)** **Model** > Updater options...
- *string* = name of sampler
- *int* = number of iterations

modelSetOR(*string* , *int* **)** **Model** > Updater options...
- *string* = name of sampler
- *int* = number of samples to generate for over-relaxed MCMC

---

modelExternalize(*string* **)** **Model** > Externalize
- *string* = file name, including the extension. The extension is conventionally ".bug" for externalized OpenBUGS models.

modelInternalize(*string* **)** **Model** > Internalize
- *string* = file name, including extension.

modelQuit(string **)** **File** > Exit
- string = 'y' or 'yes' to quick without a dialog box opening before exiting
If string is omitted, a dialog box appears before exit.

* **modelSaveLog(** *string* **)** **File** > Save As... >
- *string* = file name
If the file ends with ".txt" the log window is saved to a text file (with all graphics, fonts, etc., stripped out).

---

samplesSet(*string* **)** **Inference** > Samples... >
- *string* = node to set

samplesClear(*string* **)** **Inference** > Samples... >
- *string* = node to clear

samplesBeg(*int* **)** **Inference** > Samples...
- *int* = beginning update to define subset of the stored sample for analysis

samplesEnd(*int* **)** **Inference** > Samples...
- *int* = ending update to define subset of the stored sample for analysis

samplesFirstChain(*int* **)** **Inference** > Samples...
- *int* = select beginning of range of chains which contribute to statistics being calculated

samplesLastChain(*int* **)** **Inference** > Samples...
- *int* = select end of range of chains which contribute to statistics being calculated

samplesThin(*int* **)** **Inference** > Samples...
- *int* = thin. Every *int* th sample is used for inference. Does not impact storage requirements.

samplesStats(*string* **)** **Inference** > Samples... >
- *string* = node for which to calculate sample statistics. Can use '*' for all set nodes.

* **samplesDensity(** *string* **)** **Inference** > Samples... >
- *string* = node for which to generate density plot. Can use '*' for all set nodes.

* **samplesAutoC(** *string* **)** **Inference** > Samples... >
- *string* = node for which to generate autocorrelation plot. Can use '*' for all set nodes.

* **samplesTrace(** *string* **)** **Inference** > Samples... >
- *string* = node for which to generate trace plot. Can use '*' for all set nodes.

* **samplesHistory(** *string* **)** **Inference** > Samples... >
- *string* = node for which to generate history plot. Can use '*' for all set nodes.

* **samplesQuantiles(** *string* **)** **Inference** > Samples... >
- *string* = node for which to generate quantiles plot. Can use '*' for all set nodes.

* **samplesBgr(** *string* **)** **Inference** > Samples... >
- *string* = node for which to generate Brooks-Gelman-Rubin diagnostics. Can use '*' for all set nodes.

samplesCoda(*string0* , string1 **)** **Inference** > Samples... >
- *string0* = node for which to save all monitored values in plain text files, formatted for use in the `coda` package for R. Can use '*' for all set nodes.
- string1 = 'fileStem'

There will be one file for each parallel chain, named as `fileStemCODAchain1.txt`, `fileStemCODAchain2.txt` and so on, and a file `fileStemCODAindex.txt` indexing the row numbers corresponding to each node.

---

summarySet(*string* **)** **Inference** > Summary... >
- *string* = node for which to calculate running mean, standard deviation and quantiles

summaryStats(*string* **)** **Inference** > Summary... >
- *string* = node for which to display approximate running summary statistics

summaryMean(*string* **)** **Inference** > Summary... >
- *string* = node for which to display running mean in comma delimited form

summaryClear(*string* **)** **Inference** > Summary... >
- *string* = node to remove the running summary statistics

---

ranksSet(*string* **)** **Inference** > Rank... >
- *string* = node to be ranked (must be an array)

ranksStats(*string* **)** **Inference** > Rank... >
- *string* = node for which to summarize the simulated ranks of the components

* **ranksHistogram(** *string* **)** **Inference** > Rank... >
- *string* = node for which to display a histogram of the simulated ranks of the components

ranksClear(*string* **)** **Inference** > Rank... >
- *string* = node for which to clear the running summary

---

dicSet() **Inference** > DIC... >
Start calculating DIC and related statistics

dicClear() **Inference** > DIC... >
Clear DIC calculations from memory

dicStats() **Inference** >DIC... >
Display DIC, Dbar, Dhat, and pD

---

infoNodeValues(*string* **)** **Info** > Node info... >
- *string* = node for which to display the current value(s)

infoNodeMethods(*string* **)** **Info** > Node info... >
- *string* = node for which you want to see the type of updater used to sample from

infoNodeTypes(*string* **)** **Info** > Node info... >
- *string* = node for which you want to see the node type

infoMemory() **Info** > Memory
Show the amount of memory allocated

---

* **infoUnitializedUpdaters()** **Info** > Uninitialized Nodes
Shows nodes in the compiled model that have not been initialized yet.

infoUpdatersbyName() **Info** > Updaters(by name)
List the nodes with their associated updater algorithm in alphabetical order .

infoUpdatersbyDepth() **Info** > Updaters(by depth)
List the nodes with their associated updater algorithm in the reverse topological order to which they occur in the graphical model.

---

infoModules() **Info** > Modules
Displays all the modules (dynamic link libraries) in use.

Example

The script code below shows the steps you might use to reproduce the Rats example. To execute, highlight the entire script (or one line at a time) and select **Model** > Script.

```
# Check model syntax
modelCheck('C:/Program Files/OpenBUGS/Examples/Ratsmodel.txt')

# Load data
modelData('C:/Program Files/OpenBUGS/Examples/Ratsdata.txt')

# Compile with one chain
modelCompile(1)
```

```
# Load inital values for first chain
modelInits('C:/Program Files/OpenBUGS/Examples/Ratsinits.txt',1)

# Start with 1000 update burn-in
modelUpdate(1000)

# Set nodes of interest
samplesSet('alpha0')
samplesSet('beta.c')
samplesSet('sigma')

# Follow by a further 10,000 updates
modelUpdate(10000)

# Look at sample statistics
samplesStats('*')
```

DoodleBUGS: The Doodle Editor

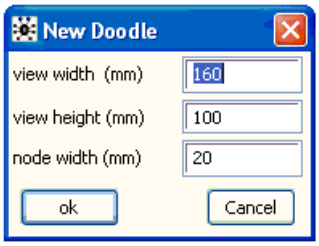
Contents

- General properties
- Creating a node
- Selecting a node
- Deleting a node
- Moving a node
- Creating a plate
- Selecting a plate
- Deleting a plate
- Moving a plate
- Resizing a plate
- Creating an edge
- Deleting an edge
- Moving a Doodle
- Resizing a Doodle

General properties [top]

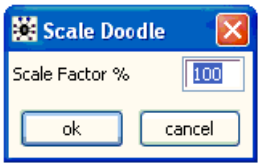
Doodles consist of three elements: *nodes* , *plates* and *edges* . The graph is built up out of these elements using the mouse and keyboard. The *Doodle* menu contains options that apply to the whole *Doodle* . These menu options are described below.

New... Dialog box for creating a new *Doodle* . Allows a choice of size of the Doodle graphic and the size of the nodes in the Doodle.



No grid, grid 1mm, grid 2mm, grid 5mm: options for the snap grid the snap grid to which the centre of each node and each corner of each plate is constrained to lie.

Scale Model: shrinks the Doodle so that the size of each node and plate plus the separation between them is reduced by a constant factor. The Doodle will move towards the top left corner of the window. This command is useful if you run out of space while drawing the Doodle. Note that the Doodle will still be constrained by the snap grid, and if the snap is coarse then the Doodle could be badly distorted.



Remove Selection: removes the highlighting from the selected node or plate of the Doodle if any.

Creating a node [top]

Point the mouse cursor to an empty region of the *Doodle* window and click. A new grey elipse will appear centered at the mouse cursor position. A flashing caret appears next to the blue word *name* . Typed characters will appear both at this caret

and within the outline of the elipse.

When first created the node will be of type *stochastic* and have associated density *dnorm.*

The type of the node can be changed by clicking on the blue word *type* at the top of the doodle. A menu will drop down giving a choice of *stochastic* , *logical* and *constant* for the node type. Constant nodes are shown as rectangular boxes.

The name of a node starts with a letter and can also contain digits and the period character "." The name must not contain two successive periods and must not end with a period. Vectors are denoted using a square bracket notation, with indices separated by commas. A colon-separated pair of integers is used to denote an index range of a multivariate node or plate.

Stochastic: Associated with stochastic nodes is a density. Click on the blue word *density* to see the choice of densities available (this will not necessarily include all those available in the *BUGS* language and described in Appendix II Functions and functionals). For each density the appropriate name(s) of the parameters are displayed in blue. For some densities default values of the parameters will be displayed in black next to the parameter name. When edges are created pointing into a stochastic node these edges are associated with the parameters in a left to right order. To change the association of edges with parameters click on one of the blue parameter names, a menu will drop down from which the required edge can be selected. This drop down menu will also give the option of editing the parameter's default value.

Logical: Associated with logical nodes is a link, which can be selected by clicking on the blue word *link* . Logical nodes also require a *value* to be evaluated (modified by the chosen link) each time the value of the node is required. To type input in the value field click the mouse on the blue word *value* . The value field of a logical node corresponds to the right hand side of a logical relation in the *BUGS* language and can contain the same functions. The value field must be completed for all logical nodes.

We emphasise that the *value* determines the value of the node and the logical links in the Doodle are for cosmetic purposes only.

It is possible to define two nodes in the same Doodle with the same name - one as logical and one as stochastic - in order to use the data transformation facility described in Data transformations

Constants: Constant nodes can be given a name or a numerical value.

Selecting a node [top]

Point the mouse cursor inside the node and left-mouse-click.

Deleting a node [top]

Select a node and press *ctrl* + *delete* key combination.

Moving a node [top]

Select a node and then point the mouse into selected node. Hold mouse button down and drag node. The cursor keys may also be used to move the centre of the node to the next snap grid point.

Creating a plate [top]

Point the mouse cursor to an empty region of the *Doodle* window and click while holding the *Ctrl* key down.

Selecting a plate [top]

Point the mouse into the lower or right hand border of the plate and click.

Deleting a plate [top]

Select a plate and press *ctrl* + *delete* key combination.

Moving a plate [top]

Select a plate and then point the mouse into the lower or right hand border of the selected plate. Hold the mouse button down and drag the plate. The cursor keys may also be used to move the cornor of the node to the next snap grid point. If the *Ctrl* key is held down while moving a plate any nodes within the plate will move with the plate.

Resizing a plate [top]

Select a plate and then point the mouse into the small region at the lower right where the two borders intercept. Hold the mouse button down and drag to resize the plate.

Creating an edge [top]

Select node into which the edge should point and then click into its parent while holding down the *ctrl* key.

Deleting an edge [top]

Select node whose incoming edge is to be deleted and then click into its parent while holding down the *ctrl* key.

Moving a Doodle [top]

After constructing a Doodle, it can be moved into a document that may also contain data, initial values, and other text and graphics. This can be done by choosing *Select Document* from the *Edit* menu, and then either copying and pasting, or dragging, the Doodle.

Resizing a Doodle [top]

To change the size of Doodle which is already in a document containing text, click once into the Doodle with the left mouse button. A narrow border with small solid squares at the corners and mid-sides will appear. Drag one of these squares with the mouse until the Doodle is of the required size.

The File Menu

Contents

- General properties
- New
- Open...
- Open Stationary...
- Save...
- Save As...
- Save Copy As...
- Close
- Page Setup...
- Print...
- Exit

General properties [top]

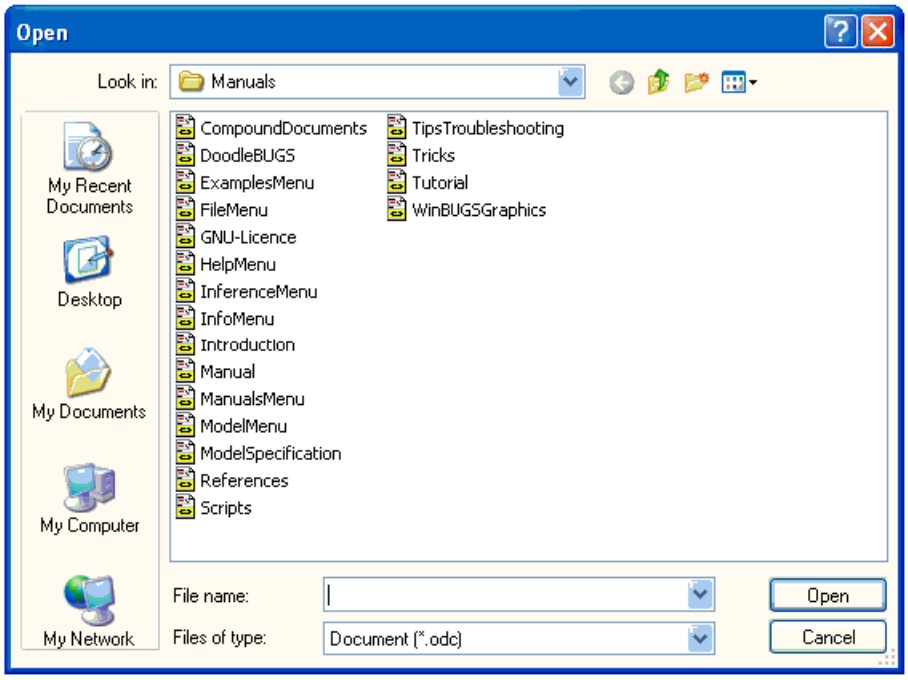
The commands in this menu apply to documents. There are options for creating, opening and saving documents

New [top]

Creates a new document and opens it in a window

Open... [top]

Opens an existing document that has been saved to a file. A modal dialog showing a list of files pops up.



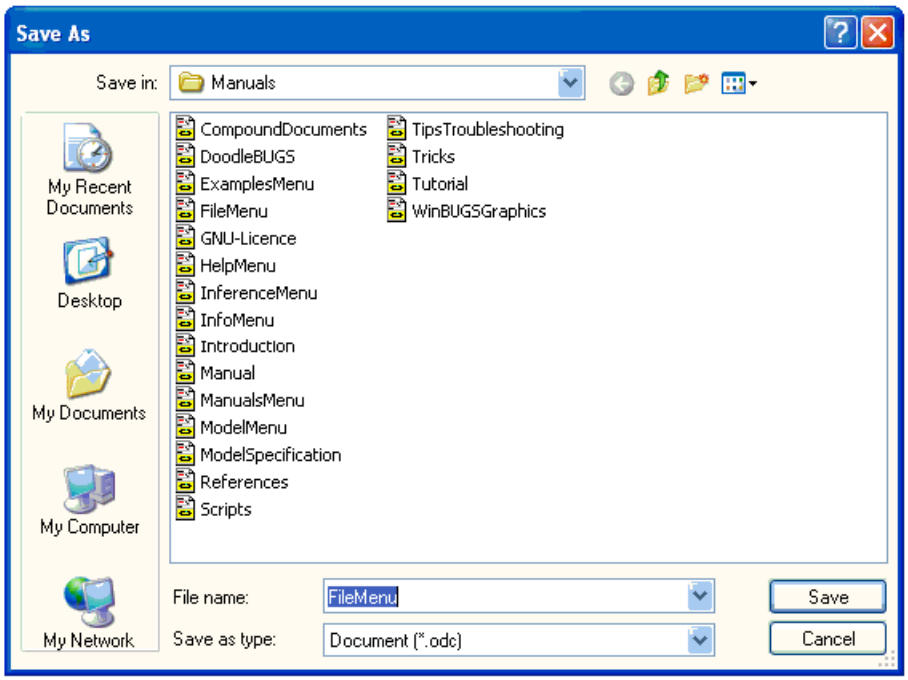
The types of files show in the list can be changed by using the 'Files of type' pull down list. By default the files list are ones storing documents. The location of the list of files displayed can be changed with the 'Look in' pull down list.

Save... [top]

Saves a document to a file. If the document has already been saved to a file it is saved to this file. If the document has never been saved to a file 'Save' behaves as 'Save As*, see the next section.

Save As... [top]

Save a document to a different file than the one it was originally stored in.



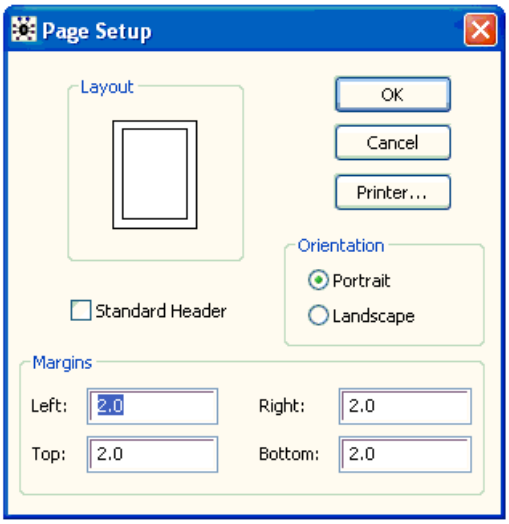
The name of the file to which the document is to be saved can be typed in the 'File name' field or a existing file can be picked from the main dialog. The location where the file is to be stored can be changed using the 'Save in' pull down list. Once the document has been saved its title bar will change to show where it has been saved. By default the type of the document does not change on saving. This behavior can be altered by using the 'save as type' pull down list.

Close [top]

Closes the top window. If the document in the top window has changed since it was last saved to file a dialog box will pop up asking the user if they want to save the document before closing the window.

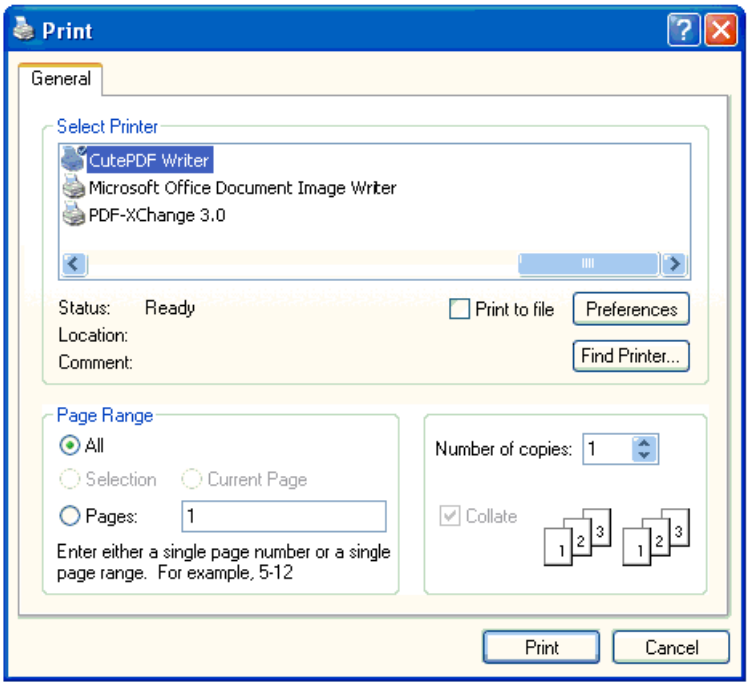
Page Setup... [top]

Options for changing the appearance of documents.



Print... [top]

Prints the document in the top window. A modal dialog box pops up allowing the user to choose which printer to use etc. We have found CutePDF Writer very useful, this 'printer' produces PDF documents that can be saved to file and then printed to paper later on.



Send Document... [top]

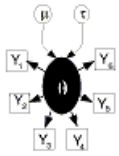
No action is taken.

Send Note... [top]

No action is taken.

Exit [top]

Closes the OpenBUGS program. The user will be prompted to save any documents that have changed. The state of the MCMC simulation will be lost. OpenBUGS can also be closed by clicking on the red button at the extreme right of the menu bar. This can easily be done by accident, so to prevent the unintended loss of the MCMC simulation a modal dialog box will pop up asking the user if they really want to quit OpenBUGS. The internalize/externalize commands can be used to preserve the MCMC simulation for later use: Externalize .



The Edit Menu

Contents

- General properties
- Undo
- Redo
- Cut
- Copy
- Paste
- Delete
- Paste Object
- Paste Special...
- Paste to Window
- Insert Object...
- Object Properties...
- Bitmap Image Object
- Select Document
- Select All
- Select Next Object
- Preferences...

General properties [top]

The commands in this menu apply to the editing of documents. OpenBUGS is build using a component framework. One component of this framework is a powerful document editor.

Undo [top]

This command un-does the last editing action carried out by the user. The full name of this command changes depending on what type of editing action is to be un-done

Redo [top]

This command restores the last editing action carried out by the user if it has been undone. The full name of this command changes depending on what type of editing action is to be re-done.

Cut [top]

This command cuts the highlighted section of the document to the Windows clip board.

Copy [top]

This command copies the highlighted section of the document to the Windows clip board.

Paste [top]

This command copies the contents of the Windows clip board to the caret position in the document.

Delete [top]

This command deletes the highlighted section from the document.

Paste Object [top]

This command copies the contents of the Windows clip board to the caret position in the document.

Paste Special... [top]

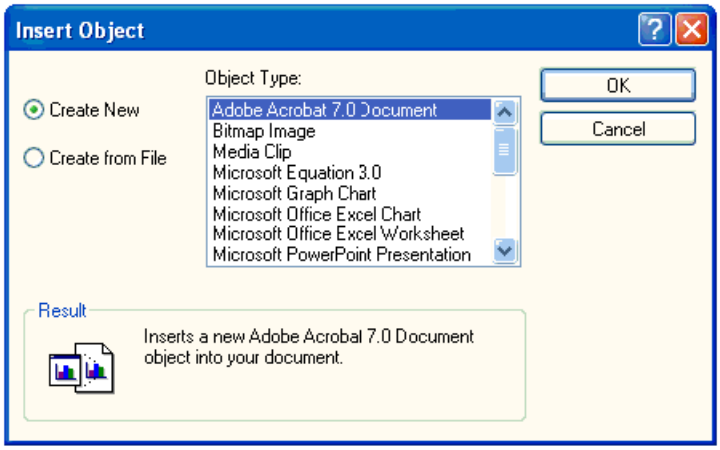
This command copies the contents of the Windows clip board to the caret position in the document. But in addition it allows the user to choose the format in which the object is copied into the document.

Paste to Window [top]

This command copies the contents of the Windows clip board to a new window.

Insert Object... [top]

This command inserts an OLE object at the current caret position. A modal dialog box pops up giving a choice of OLE objects to insert



Object Properties... [top]

This command allows the user to change the font and font style, size, color and effects.

Bitmap Image Object [top]

This command allows the user to open or edit an in-line bitmap image.

Select Document [top]

This command selects the document in the window.

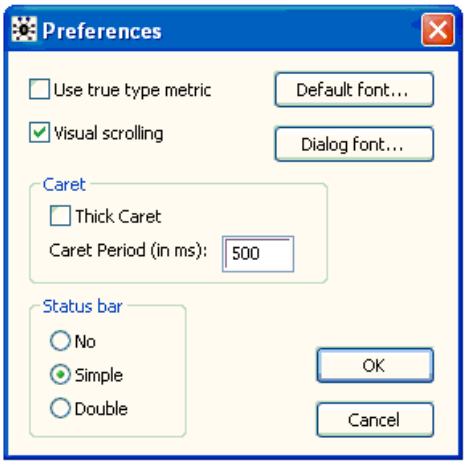
Select All [top]

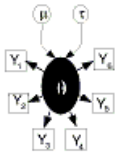
This command selects the contents of the entire document.

Select Next Object [top]

Preferences... [top]

This command opens a dialog that allows the user to set certain preferences for documents. In addition there is an option that affects the status bar, the line of information at the bottom of the main OpenBUGS window. By default (option Simple) this line just shows error messages, but it can be hidden (option No) or made to show how much memory OpenBUGS is using (option Double). Ticking the thick caret box can make the caret easier to see.





The Attributes Menu

Contents

- General properties
- Weight
- Fixed Size...
- Size...
- Fixed Color
- Color...
- Default Font
- Font...
- Typeface...

General properties [top]

The commands change the appearance of text in documents. They either act on the highlighted text or affect the appearance of subsequently entered text.

Weight [top]

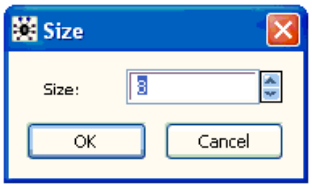
The commands in this group set the weight of the highlighted text. The weight can either be Regular, Bold, Italic or Underline. If the entire highlighted text is of one single weight a tick mark will appear against the relevant weight.

Fixed Size [top]

The commands in this group change the size of the highlighted text to a predefined font size. Sizes available are 8, 9, 10, 12, 16, 20 and 24pts. If the entire highlighted text is of one single font size a tick mark will appear against the relevant size.

Size... [top]

This command opens a non modal dialog box which allows the user to choose the font size for the highlighted text.



Fixed Color [top]

The commands in this group change the color of the highlighted text to a predefined color. Colors available are Default Color, Black, Red, Green, and Blue. If the entire highlighted text is of one single color a tick mark will appear against this color.

Color... [top]

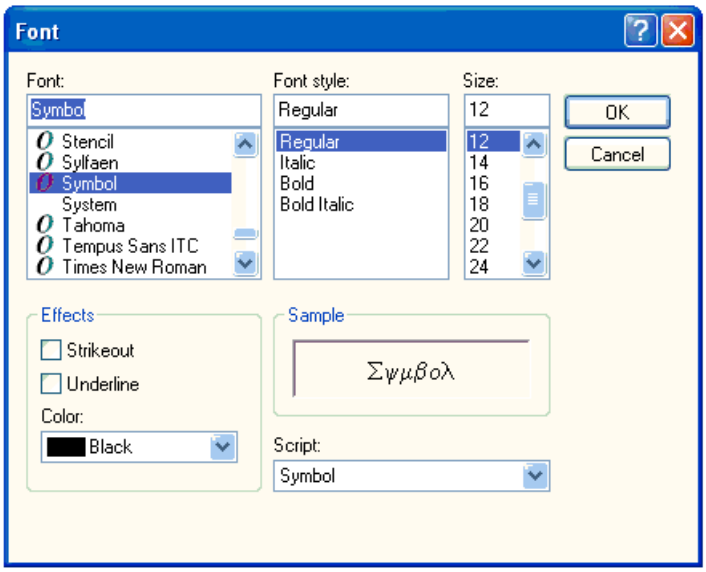
This command opens a modal dialog box which allows the user to choose the color for the highlighted text.

Default Font [top]

This command changes the highlighted text to be in the default font. If the entire highlighted text is in the default font a tick mark will appear against this option.

Font... [top]

This command opens a modal dialog box which allows the user to choose the font for the highlighted text. The color of the text can also be chosen from a restricted range of options. This command is useful for entering Greek characters into a document.



Typeface... [top]

This command is a simplified version of the Font... command without the ability to change the color of the text.

The Tools Menu

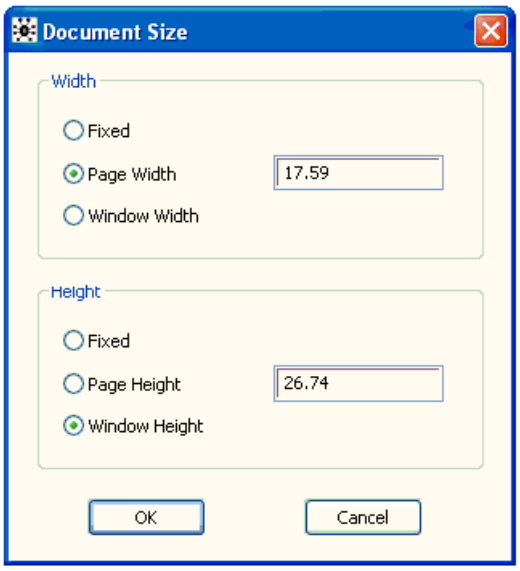
Contents

- General properties
- Document Size...
- Insert OLE Object...
- Insert Header
- Create Link
- Create Target
- Create Fold
- Expand All
- Collapse All
- Fold...
- Encode Document
- Encode Selection
- Encode File...
- Encode File List
- Decode
- About Encoded Material

General properties [top]

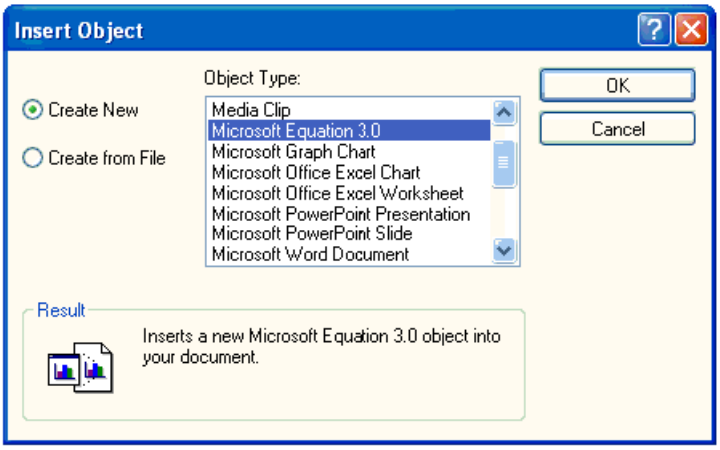
The commands in this menu provide various miscelaneous tools.

Document Size... [top]



Insert OLE Object [top]

This command opens a modal dialog box allowing an OLE object to be inserted into a document at the caret position.



Insert Header [top]

This command inserts a header into the document which becomes visible when the document is printed.

Create Link [top]

This command turns the highlighted text into a hyper text link if possible. The highlighted text must have a form like <Dialog.Bleep> Bleep <> which gives the following link Bleep .

Typical links are

<StdLinks.ShowTarget('Contents')> top <>

<StdCmds.OpenBrowser('Manuals/ToolsMenu', 'Tools Menu')> Tools Menu <>

If the 'Show Marks' command from the Edit menu is choosen links show up as angled hollow arrows.To return the link into text form hold the control key down and click into one of these arrows.

Create Target [top]

This command turns the highlighted text into a target that can be jumped to. The highlighted text should have a form like <ExpandAll>Expand All<>.

If the 'Show Marks' command from the Edit menu is choosen links show up as hollow circles.To return the target into text form hold the control key down and click into one of these circles.

Create Fold [top]

This command turns the highlighted text into a fold enclosed between hollow (open) arrows. Clicking on either of the arrows will close the fold (the arrows become solid black), the caret is then placed between the arrows and text can be typed into the closed fold. Clicking on the black arrow will open the fold again.

The text in the open fold

⇨This command turns the highlighted text into a fold enclosed between hollow arrows. Clicking on either of the arrows will close the fold (the arrows become solid black), the caret then is placed between the arrows and text can be typed into the closed fold. Clicking on the black arrow will open the fold again.⇦

The text in the closed fold

➔ this is a folded version of the above paragraph ➔

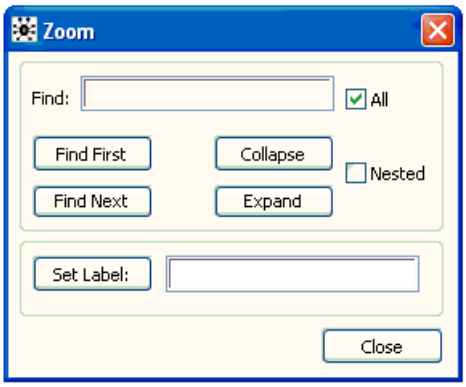
Expand All [top]

This command opens all the closed folds in the document.

Collapse All [top]

This command closes all the open folds in the document.

Fold... [top]



Encode Document [top]

This command encodes a document into ascii characters. The encoded version of the document is opened in a new window. The encoded document can be sent by email.

Encode Selection [top]

This command encodes a highlighted section of a document into ascii characters. The encoded section of the document is opened in a new window.

Encode File... [top]

This command encodes a file into ascii characters. The encoded version of the file is opened in a new window. A standard file open dialig pops up for choosing which file to encode

Encode File List [top]

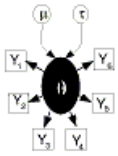
This command encodes a list of files into a document containg only ascii charcters. The document is opened in a new window. The list of files to be enco´ded must be highlighted in the document in the top window.

Decode [top]

This command decodes an encoded document, file or list of files.

About Encoded Material [top]

This command gives information about what is encoded.



The Text Menu

Contents

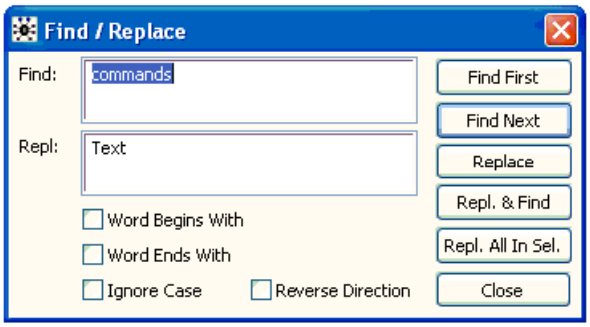
- General properties
- Find/Replace...
- Find Again
- Find Previous
- Find First
- Find Last
- Shift Left
- Shift Right
- Insert Paragraph
- Insert Ruler
- Insert-Soft-Hyphen
- Insert Non-Brk-Hyphen
- Insert Non-Brk-Space
- Insert Digit Space
- Show/Hide Marks
- Make Default Attributes
- Make Default Ruler

General properties [top]

The commands in this menu provide tools for searching for and replacing text and inserting special formatting options into the text.

Find/Replace... [top]

This command is used to find and replace text. It causes a non modal dialog box to pop up.



The 'Find' field contains the text to be searched for. If a stretch of text is highlighted before this dialog is opened this will appear in the 'Find' field. The 'Repl.' field is the text that will be substituted for the text in the 'Find' field if the 'Replace', 'Repl. & Find' or 'Repl. All In Sel.' buttons are clicked. 'Repl. & Find' will also find the next occurrence of the 'Find' text. 'Repl. All In Sel.' will replace all occurrences of the 'Find' text that are within the highlighted section of text in the window.

Find Again [top]

This command finds the next occurrence of a highlighted piece of text in the document.

Find Previous [top]

This command finds the previous occurrence of a highlighted piece of text in the document.

Find First [top]

This command finds the first occurence of a highlighted piece of text in the document.

Find Last [top]

This command finds the last occurence of a highlighted piece of text in the document.

Shift Left [top]

This command shifts the highlighted text one tab stop to the left.

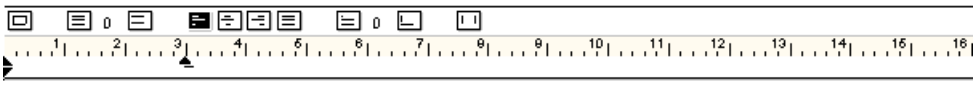
Shift Right [top]

This command shifts the highlighted text one tab stop to the right.

Insert Paragraph [top]

Insert Ruler [top]

This command inserts a ruler at the caret position of the text document.



The ruler controls the formatting of the text below the ruler until another ruler is encountered. Tab stops can be added to the ruler by clicking with the mouse in the white region below the grey shaded bar. The tab stop shows up as a black inverted triangle. The position of the tab stop can be changed by dragging it with the mouse. To remove a tab stop drag it all the way to the extreme left edge of the ruler.

The buttons above the grey shaded bar control the margins, line spacing, and justification. If you click the first button, a black triangle will appear on the right side of the ruler. Dragging this triangle will change the right margin. The second and third buttons decrease and increase the line spacing, respectively. The current line spacing is shown by the number between the two buttons. The justification of the text can be changed to left-justified, center-justified, right-justified, or fill-justified by clicking in the fourth, fifth, sixth or seventh button, respectively. The eighth and ninth buttons control the line spacing between paragraphs. Clicking the last button will cause the ruler to force a page break.

Insert-Soft-Hyphen [top]

Insert Non-Brk-Hyphen [top]

This command inserts a hyphen that will not split over a line break.

Insert Non-Brk-Space [top]

This command inserts a space that will not split over a line break. It is useful if you want to underline a stretch of text containing spaces.

Insert Digit Space [top]

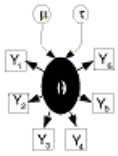
This command inserts a space that has the same width as a digit (0..9).

Show/Hide Marks [top]

This command toggles between making special formatting controls visible in the text or making them invisible. Seeing the formatting controls is particularly useful when creating targets and links. See the Tools menu for a description on how to do this.

Make Default Attributes [top]

Make Default Ruler [top]



The Model Menu

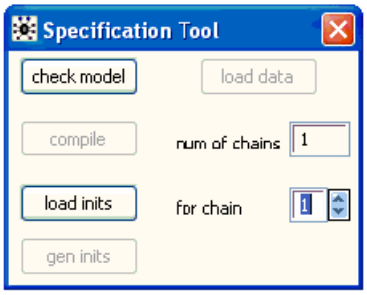
Contents

- General properties
- Specification...
- Update...
- Save State
- Random number generator...
- Script
- Pretty print
- Latex
- Input / Output options...
- Compile options...
- Updater options...
- Externalize
- Internalize

General properties [top]

The commands in this menu either apply to the whole statistical model or open dialog boxes.

Specification... [top]



This non-modal dialog box acts on the focus view (the window with its title bar highlighted).

check model : If the focus view contains text, *OpenBUGS* assumes the model is specified in the *BUG* S language. The *check model* button parses the *BUG* S language description of the statistical model. If a syntax error is detected the cursor is placed where the error was found and a description of the error is given on the status line (lower left corner of screen). If no syntax errors are found, the message "model is syntactically correct" should appear in the status line. If text is highlighted, parsing starts from the first character highlighted *(* i.e. highlight the word `model`), otherwise parsing starts at the top of the window even if the cursor is currently in a different location..

If the focus view is a *Doodle* or contains a selected *Doodle (i.e.* the Doodle has been selected and is surrounded by a hairy border *), OpenBUG* S assumes the model has been specified graphically. If a syntax error is detected the node where the error was found is highlighted and a description of the error is given on the status line.

load data: The *load dat* a button acts on the focus view; it will be grayed out unless the focus view contains text and a syntactically correct model has been checked.

Data can be identified in two ways:

1) if the data are in a separate document, the window containing that document needs to be the focus view when the *load dat* a command is used;
2) if the data are specified as part of a document, the first character of the data (either the word `list` if in S-Plus format, or

the first array name if in rectangular format) must be highlighted and the data will be read starting with the first highlighted character.

See Formatting of data for details of how the data should be formatted.

Any syntax errors or data inconsistencies are displayed in the status line. Corrections can be made to the data without returning to th *e check mode* l stage. When the data have been loaded successfully, "Data Loaded" should appear in the status line.

The *load dat* a button becomes active once a model has been successfully checked, and ceases to be active once the model has been successfully compiled.

num of chains: The number of chains to be simulated can be entered into the text entry field next to the caption *num of chains* . This field can be typed in after the model has been checked and before the model has been compiled. By default, one chain is simulated.

compile: The *compil* e button builds the data structures needed to carry out MCMC sampling. The model is checked for completeness and consistency with the data. Any inconsistencies or errors are displayed on the status line.

A node called 'deviance' is automatically created which calculates minus twice the log-likelihood at each iteration, up to a constant. This node can be used like any other node in the graphical model.

This command becomes active once the model has been successfully checked. When the model has been successfully compiled, the message ' model compiled ' should appear in the status line.

load inits: The *load init* s button acts on the focus view; it will be grayed out unless the focus view contains text. The initial values will be loaded for the chain indicated in the text entry field to the right of the caption *for chai* n. The value of this text field can be edited to load initial values for any of the chains.

Initial values are specified in exactly the same way as data files. If some of the elements in an array are known (say because they are constraints in a parameterisation), those elements should be specified as missing (NA) in the initial values file.

This command becomes active once the model has been successfully compiled, and checks that initial values are in the form of an S-Plus object or rectangular array and that they are consistent with the model and any previously loaded data. Any syntax errors or inconsistencies in the initial value are displayed on the status line.

If, after loading the initial values, the model is fully initialized this will be reported by displaying the message"initial values loaded: model initialized" . Otherwise the status line will show the message"initial values loaded but this or another chain contain uninitialized variables" . The second message can have several meanings:

a) If only one chain is simulated it means that the chain contains some nodes that have not been initialized yet.
b) If several chains are to be simulated it could mean that no initial values have been loaded for one of the chains.

In either case further initial values can be loaded, or the*gen init* s button can be pressed to generate initial values for*all* the uninitialized nodes in *all* the simulated chains. See the 'gen inits' section below for a description of how the values are generated.

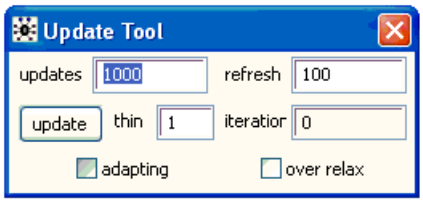
Generally it is recommended to load initial values for all fixed effect nodes (founder nodes with no parents) for all chains, initial values for random effects can be generated using the *gen init* s button.

The *load init* s button can still be executed once the MCMC sampling has been started. It will have the effect of starting the sampler out on a new trajectory. A modal warning message will appear if the command is used in this context.

gen inits: The *gen init* s button attempts to generate initial values by sampling either from the prior or from an approximation to the prior. In the case of discrete variables a check is made that a configuration of zero probability is not generated. This command can generate extreme values if any of the priors are very vague. If the command is successful the message "initial values generated: model initialized" is displayed otherwise the message "could not generate initial values" is displayed .

The *gen init* s button becomes active once the model has been successfully compiled, and will cease to be active once the model has been initialized.

Update... [top]



This menu will become active after the model has been compiled and initialized. It has the following fields that accept user input:

updates : The number of MCMC samples to save. Thus, *update* s * *thin* MCMC updates will be carried out.

refresh : The number of updates divided by *thin* between redrawing the screen.

thin : The samples from every k th iteration will be used for inference, where k is the value of *thin* . Setting *k* > 1 can help to reduce the autocorrelation in the sample, but there is no real advantage in thinning except to reduce storage requirements.

over relax : Click on this box (a tick will then appear) to select an over-relaxed form of MCMC (Neal, 1998) which will be executed where possible. This generates multiple samples at each iteration and then selects one that is negatively correlated with the current value. The time per iteration will be increased, but the within-chain correlations should be reduced and hence fewer iterations may be necessary. However, this method is not always effective and should be used with caution. The auto-correlation function may be used to check whether the mixing of the chain is improved.

Click on the ***update button*** to start updating the model. Clicking on the update button during sampling will pause the simulation after the current block of iterations, as defined by *refres* h, has been completed; the number of updates required can then be changed if needed. Clicking on the update button again will restart the simulation. This button becomes active when the model has been successfully compiled and given initial values.

The ***iteration*** field displays the total number of MCMC updates divided by *thin* that have been carried out.

The ***adapting*** box will be ticked while the Metropolis or slice-sampling MCMC algorithm is in its initial tuning phase where some optimization parameters are tuned. The Metropolis and slice-sampling algorithms have adaptive phases of 4000 and 500 iterations respectively which will be discarded from all statistics. For details on how to change these default settings please see the Updater Options section .

When there are multiple chains, a single update is generated for each chain and then the cycle is repeated for the requested number of updates. Random numbers are currently generated from a single random number sequence shared across the chains. When thin>1, thinned updates are drawn from the first chain, then thinned updates are generated from the second chain, and so on. However if thin=1, and the monitored samples are thinned via *Inference* > *Samples* , the output will be different, since we are then looping over chains within each iteration.

Save State [top]

The current state of all the stochastic variables in the model are displayed in S-Plus format with a separate window for each chain. These can be used as an initial value file for future runs.

Random number generator... . [top]

Opens a non-modal dialog box, which is available only after compilation is completed and before any updates have been performed. The state can be changed after initial values are generated but before updates have been performed, however, this is not recommended.

The internal state of the random number generator can be set to one of 14 predefined states using the up down arrows. Each predefined state is 10^{12} draws apart to avoid overlap in random number sequences.

Script [top]

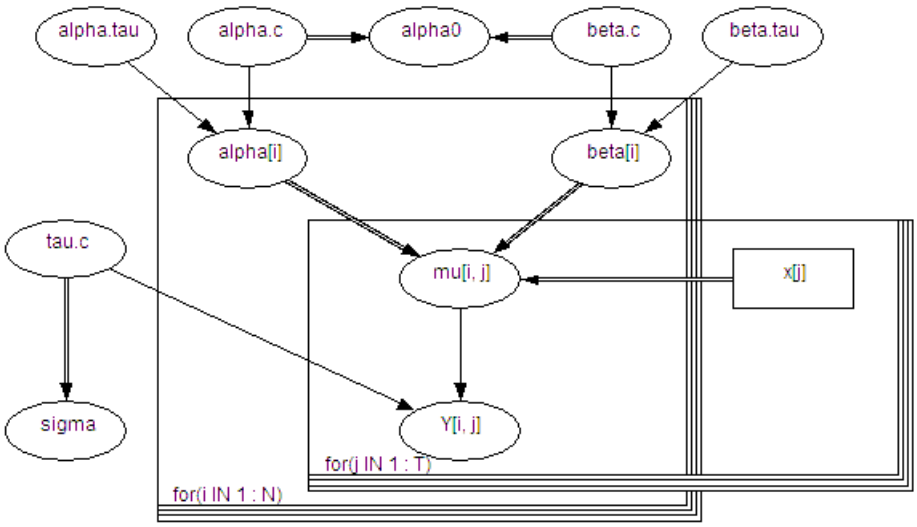
The Script menu item is used to execute "batch scripts" from the GUI-mode. If the focus-view contains a series of

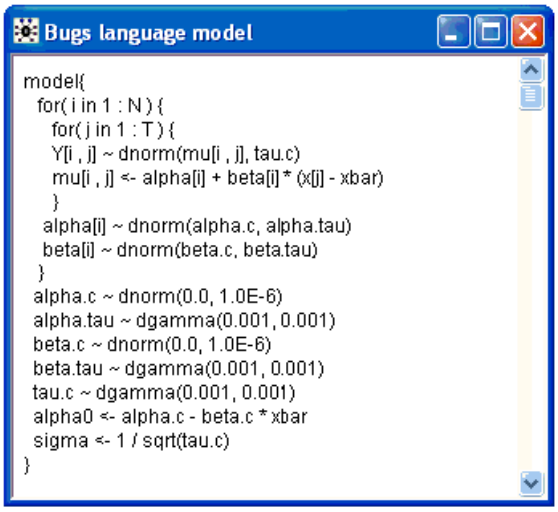
OpenBUGS batch-mode commands, then selecting the script menu item from the *Model* menu will cause the script to be executed. A subset of the script commands can be executed by highlighting them. Highlighting part of a batch command will likely produce an error. The script menu item allows batch mode execution to be mixed with GUI execution so batch scripts can be created incrementally and some repetitive tasks under the GUI (e.g., specifying variables to be monitored) can be automated during the model building/testing analysis phase. See Scripts and Batch-mode for more details.

Pretty print [top]

Opens a new window containing the *BUGS* language code describing the currently checked model. The model can be specified in the *BUGS* language or as a *Doodle* . Any comments in the original *BUGS* language description will be lost. The *BUGS* langguage code will be formatted in a standardized form.

For example if the model described by this Doodle is checked the pretty print option will open a windows containing the BUGS language code below





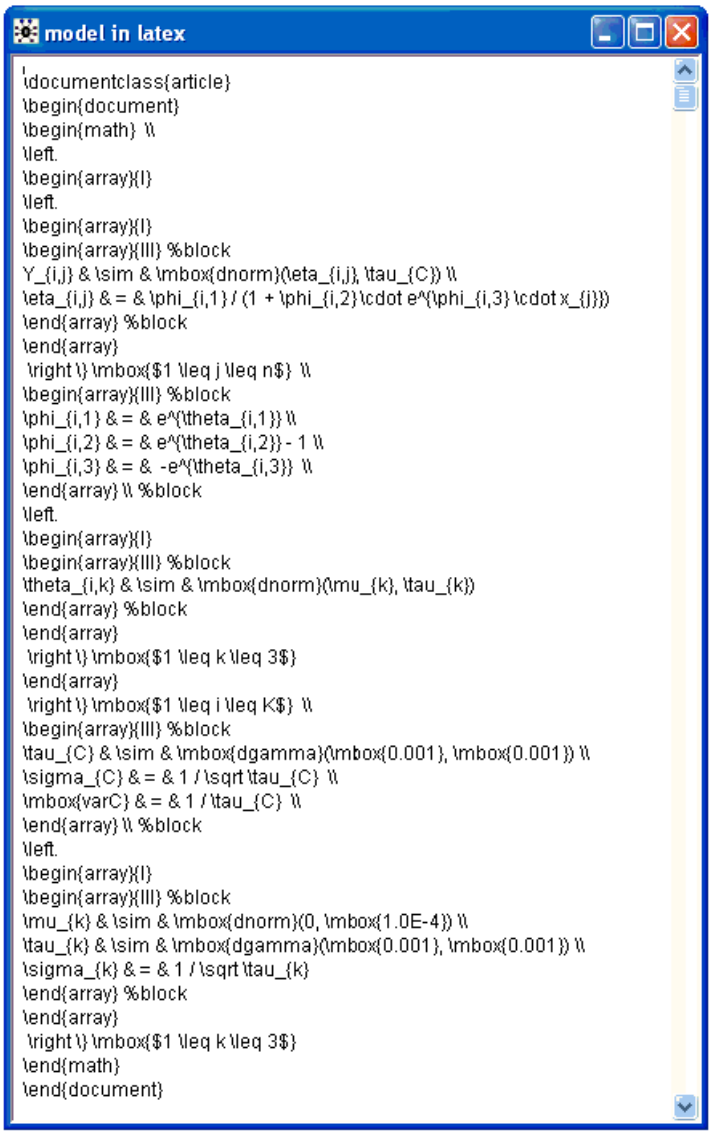
Latex [top]

Opens a new window containing Latex code describing the currently checked model. The model can be specified in the *BUGS* language or as a *Doodle* . Any comments in the original *BUGS* language description will be lost.

For example, if the following model is checked then the latex option will produce:

```
model {
  for (i in 1:K) {
    for (j in 1:n) {
      Y[i, j] ~ dnorm(eta[i, j], tau.C)
      eta[i, j] <- phi[i, 1] / (1 + phi[i, 2] * exp(phi[i, 3] * x[j]))
```

```
    }
    phi[i, 1] <- exp(theta[i, 1])
    phi[i, 2] <- exp(theta[i, 2]) - 1
    phi[i, 3] <- -exp(theta[i, 3])
    for (k in 1:3) {
      theta[i, k] ~ dnorm(mu[k], tau[k])
    }
  }
  tau.C ~ dgamma(1.0E-3, 1.0E-3)
  sigma.C <- 1 / sqrt(tau.C)
  varC <- 1 / tau.C
  for (k in 1:3) {
    mu[k] ~ dnorm(0, 1.0E-4)
    tau[k] ~ dgamma(1.0E-3, 1.0E-3)
    sigma[k] <- 1 / sqrt(tau[k])
  }
}
```



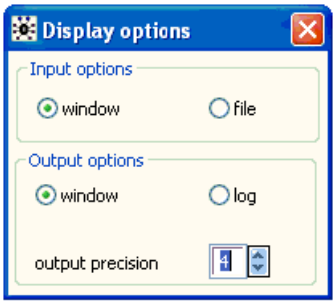
Processing the latex code gives

$$
\left.\begin{array}{rcl}
\left.\begin{array}{rcl} Y_{i,j} & \sim & \text{dnorm}(\eta_{i,j}, \tau_C) \\ \eta_{i,j} & = & \phi_{i,1}/(1+\phi_{i,2}\cdot e^{\phi_{i,3}\cdot x_j}) \end{array}\right\} 1 \le j \le n \\
\phi_{i,1} = e^{\theta_{i,1}} \\
\phi_{i,2} = e^{\theta_{i,2}} - 1 \\
\phi_{i,3} = -e^{\theta_{i,3}} \\
\left.\begin{array}{rcl} \theta_{i,k} & \sim & \text{dnorm}(\mu_k, \tau_k) \end{array}\right\} 1 \le k \le 3
\end{array}\right\} 1 \le i \le K
$$

$$
\begin{array}{rcl}
\tau_C & \sim & \text{dgamma}(0.001, 0.001) \\
\sigma_C & = & 1/\sqrt{\tau_C} \\
\text{varC} & = & 1/\tau_C
\end{array}
$$

$$
\left.\begin{array}{rcl}
\mu_k & \sim & \text{dnorm}(0, 1.0\text{E-}4) \\
\tau_k & \sim & \text{dgamma}(0.001, 0.001) \\
\sigma_k & = & 1/\sqrt{\tau_k}
\end{array}\right\} 1 \le k \le 3
$$

Input / Output options... [top]



The model specification tool takes input from the focus window when the Input options is set to window, or from a file menu prompt when the Input option is set to file.

OpenBUGS will either produce output in windows if the Output option is set to window, or produce output in the log window if the Output option is set to log.

The precision to which *OpenBUGS* displays output is controlled by the output precision field.

Compile options... [top]

compile logicals : *OpenBUGS* tries to write and compile new Component Pascal classes to represent logical nodes in the statistical model. This option, which can increase execution speed, is available only with GUI execution of OpenBUGS.exe, as it relies on run-time linking This option is not available with native linux execution or programs such as BRugs, which utilize OpenBUGS.so or OpenBUGS.dll.

updater by method : *OpenBUGS* chooses update algorithms for the model in a "by method" order if this box is checked, otherwise updaters are chosen by order of node name.

use chain graph : *OpenBUGS* tries to rewrite random effects in terms of a chain graph.

trap on error : Causes *OpenBUGS* to display a "trap" window when an unidentified error occurs, instead of displaying a short message "Sorry, something went wrong..." in the status bar. Trap windows contain information intended to help programmers fix the source code of OpenBUGS. Therefore some experts may find them useful for identifying difficult errors, but most users will find them more confusing than the default message.

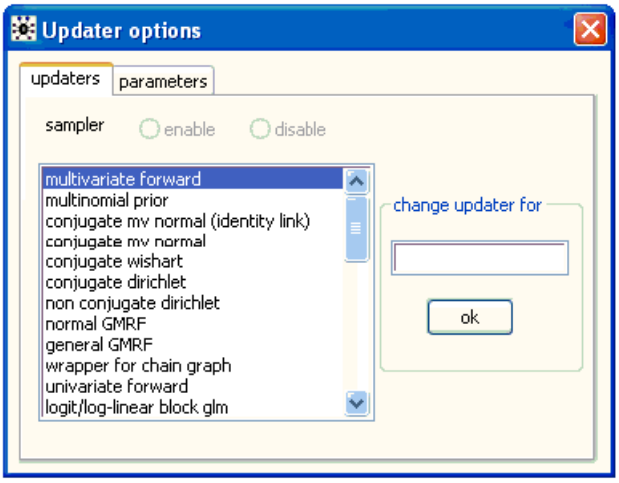
Updater options [top]

Tabbed dialog box for controlling how *OpenBUGS* updates the model.

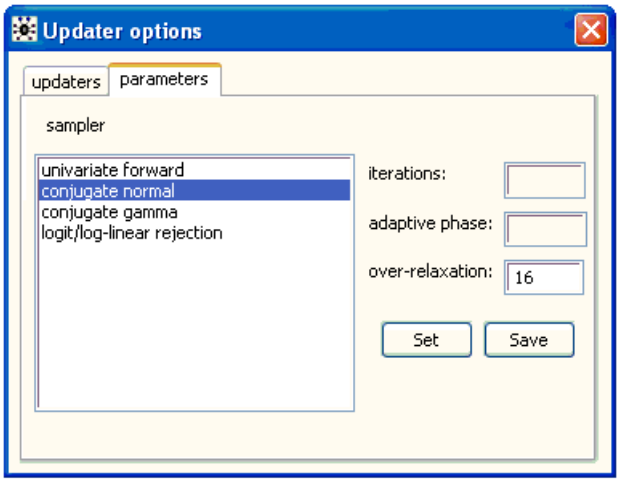
The *updaters* tab is used to influence which updater algorithms are used, while the *parameters* tab is used to adjust the default parameters of the chosen algorithms.

The updater (sampler) algorithms that OpenBUGS has available are displayed in the list box underneath "sampler". The radio button can be used to enable / disable the selected sampler. The radio buttons are only active before a model has been compiled. They control which algorithms *OpenBUGS* will consider using when the model is compiled.

Once the model has been compiled, it is possible to change the algorithm used for a node (or a block of nodes) by typing the name of the node (variable) in the text entry box underneath "change updater for", selecting a new updater algorithm from the list and clicking ok. You can see which updaters are being used for each node by clicking *Info* > *Updaters (by name)* from the menu bar. The info menu can also be checked after a change is requested to confirm the change was made. There is more information on the resulting list in the Info menu manual page. To change the updater for a block of nodes (from Info>Updaters (by name), enter the name of the first node in the block. A block updater for a block of nodes can only be changed to another block updater for the block of nodes.



The *parameters* tab of the dialog lists the actual updater algorithms used in sampling nodes in the compiled model. To change the default parameters of one of these updaters, select it from the list and then edit the appropriate fields. Clicking *set* will modify the sampling parameters for the duration of the current *OpenBUGS* session; the default parameters will be restored when *OpenBUGS* is restarted. Clicking *save* will make changes to the default parameters of the algorithm persistent across *OpenBUGS* sessions. Changes to the default parameters can only be made after a model has been compiled.



Externalize [top]

This option writes out all the internal data structures that OpenBUGS has created for a compiled model . It provides a way of "saving" the model for future use. The user is prompted to enter a filestem name in a dialog box and a file called 'filestem.bug' is created in a folder called "Restart" in the OpenBUGSxxx program folder. If there were no errors when saving the compiled model, the message "model externalized to file ok"is displayed in the status bar.

The output files can be very large; users should remove these files when they are no longer needed. Script commands (see Scripts and Batch-mode) are a better method for creating re-producible results, especially if these results need to be sent to other users.

Internalize [top]

This option reads in all the internal data structures that OpenBUGS needs to re-create an executing model from a file created by Externalize. It provides a way of "restarting" the model for further use. This option will open a window containing a BUGS language description of the restarted model and all the tool dialogs will be restored to the state when the model was last executed. If there were no errors when internalizing the saved model, the message "model internalized from file ok" is displayed in the status bar.

The Inference Menu

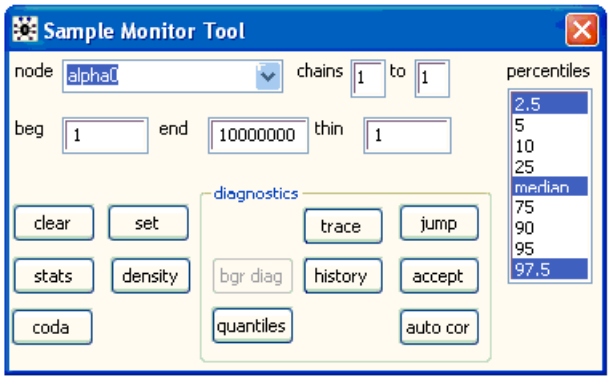
Contents

- General properties
- Samples...
- Compare...
- Correlations...
- Summary...
- Rank...
- DIC...

General properties [top]

These menu items open dialog boxes for making inferences about parameters of the model. The commands are divided into three sections: the first three commands concern an entire set of monitored values for a variable; the next two commands are space-saving short-cuts that monitor running statistics; and the final command, DIC..., concerns evaluation of the *Deviance Information Criterion* proposed by Spiegelhalter et al. (2002). **Users should ensure their simulation has converged before using Summary..., Rank... or DIC...** Note that if the MCMC simulation has an adaptive phase it will not be possible to make inference using values sampled before the end of this phase.

Samples... [top]



This command opens a non-modal dialog for analysing stored samples of variables produced by the MCMC simulation.

It is incorrect to make statistical inference about the model when the simulation is in an adaptive phase. For this reason some of the buttons in the samples dialog will be grayed out during any adaptive phase.

The dialog fields are:

node: The variable of interest must be typed in this text field. If the variable of interest is an array, slices of the array can be selected using the notation variable[lower0:upper0, lower1:upper1, ...]. The buttons at the bottom of the dialog act on this variable. A star '*' can be entered in the node text field as shorthand for all the stored samples. These buttons are arranged in two groups: the first group is associated with quantities of substantive interest while the second group gives information about how well the simulation is performing.

OpenBUGS automatically sets up a logical node to measure a quantity known as *deviance* ; this may be accessed, in the same way as any other variable of interest, by typing its name, i.e. "deviance", in the *node* field of the *Sample Monitor Tool* . The definition of deviance is -2 * log(likelihood): 'likelihood' is defined as p(*y* | *theta*), where *y* comprises all stochastic nodes given values (i.e. data), and *theta* comprises the *stochastic parents* of *y* - 'stochastic parents' are the stochastic nodes upon which the distribution of *y* depends, when collapsing over all logical relationships.

beg and ***end:*** numerical fields used to select a subset of the stored sample for analysis.

thin: numerical field used to select every k^{th} iteration of each chain to contribute to the statistics being calculated, where *k*

is the value of the field. Note the difference between this and the thinning facility on the Update Tool dialog box: when thinning via the *Update Tool* we are *permanently* discarding samples as the MCMC simulation runs, whereas here we have already generated (and stored) a suitable number of (posterior) samples and may wish to discard some of them only temporarily. Thus, setting $k > 1$ here will not have any impact on the storage (memory) requirements of processing long runs; if you wish to reduce the number of samples actually stored (to free-up memory) you should thin via the *Update Tool* .

chains . to .: can be used to select the chains which contribute to the statistics being calculated.

clear: removes the stored values of the variable from computer memory.

set: must be used to start recording a chain of values for the variable.

stats: produces summary statistics for the variable, pooling over the chains selected. The required percentiles can be selected using the *percentile* selection box. The quantity reported in the MC error column gives an estimate of $s / N^{1/2}$, the Monte Carlo standard error of the mean. The batch means method outlined by Roberts (1996; p.50) is used to estimate s.

density: plots a smoothed kernel density estimate for the variable if it is continuous or a histogram if it is discrete.

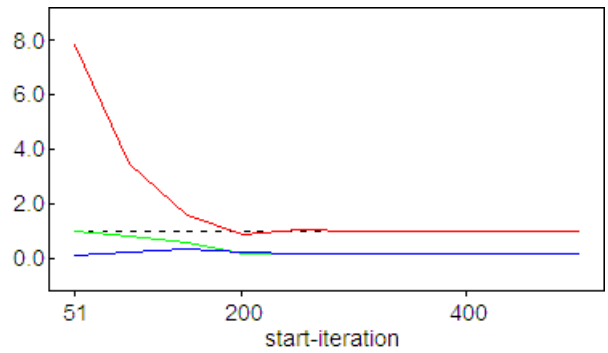
coda: produces an ascii representation of the monitored values suitable for use in the *CODA* R/Splus diagnostic package. A window for each chain is produced, corresponding to the .out files of *CODA* , showing the iteration number and value (to four significant figures). There is also a window containing a description of which lines of the .out file correspond to which variable - this corresponds to the *CODA* .ind file. These can be named accordingly and saved as text files for further use. (Care may be required to stop the Windows system adding a .txt extension when saving: enclosing the required file name in quotes should prevent this.)

trace: plots the variable value against iteration number. This trace is dynamic, being redrawn each time the screen is redrawn.

jump: plots the mean square jumping distance of nodes with updaters averaged over batches of 100 iterations. This is related to the lag one auto-correlation.

bgr diag: calculates the Gelman-Rubin statistic, as modified by Brooks and Gelman (1998). The basic idea is to generate multiple chains starting at over-dispersed initial values, and assess convergence by comparing within- and between-chain variability over the second half of those chains. We denote the number of chains generated by M and the length of each chain by 2T. We take as a measure of posterior variability the width of the $100(1 - \alpha)$% credible interval for the parameter of interest (in OpenBUGS, $\alpha = 0.2$). From the final T iterations we calculate the empirical credible interval for each chain. We then calculate the average width of the intervals across the M chains and denote this by W. Finally, we calculate the width B of the empirical credible interval based on all MT samples pooled together. The ratio R = B / W of pooled to average interval widths should be greater than 1 if the starting values are suitably overdispersed; it will also tend to 1 as convergence is approached, and so we might assume convergence for practical purposes if R < 1.05, say.

Rather than calculating a single value of R, we can examine the behaviour of R over iteration-time by performing the above procedure repeatedly for an increasingly large fraction of the total iteration range, ending with all of the final T iterations contributing to the calculation as described above. Suppose, for example, that we have run 1000 iterations (T = 500) and we wish to use the resulting sample to calculate 10 values of R over iteration-time, ending with the calculation involving iterations 501 – 1000. Calculating R over the final halves of iterations 1– 100, 1 – 200, 1 – 300, ..., 1 – 1000, say, will give a clear picture of the convergence of R to 1 (assuming the total number of iterations is sufficiently large). If we plot against the starting iteration of each range (51, 101, 151, ..., 501), then we can immediately read off the approximate point of convergence, e.g.



OpenBUGS automatically chooses the number of iterations between the ends of successive ranges: max(100, 2T / 100). It then plots R in red, B (pooled) in green and W (average) in blue. Note that B and W are normalised so that the maximum estimated interval width is one – this is simply so that they can be seen clearly on the same scale as R. Brooks and Gelman (1998) stress the importance of ensuring not only that R has converged to 1 but also that B and W have converged to stability. This strategy works because both the length of the chains used in the calculation and the start-iteration are always increasing. Hence we are guaranteed to eventually (with an increasing sample size) discard any burn-in iterations and

include a sufficient number of stationary samples to conclude convergence.

In the above plot convergence can be seen to occur at around iteration 250. Note that the values underlying the plot can be listed to a window by right-clicking on the plot, selecting Properties, and then clicking on Data (see OpenBUGS Graphics).

history: plots out a complete trace for the variable.

accept: plots the fraction of iterations a node with an updater changes its value over batches of 100 iterations.

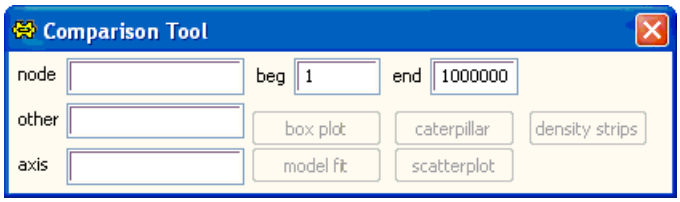
quantiles: plots out the running mean with running 95% confidence intervals against iteration number.

auto cor: plots the autocorrelation function of the variable out to lag 100.

See `OpenBUGS Graphics` for details of how to customize these plots.

Compare... [top]

Select *Compare...* from the *Inference* menu to open the *Comparison Tool* dialog box. This is designed to facilitate comparison of elements of a vector of nodes (with respect to their posterior distributions).



node: defines the vector of nodes to be compared with each other. As the comparisons are with respect to posterior distributions, *node* must be a monitored variable.

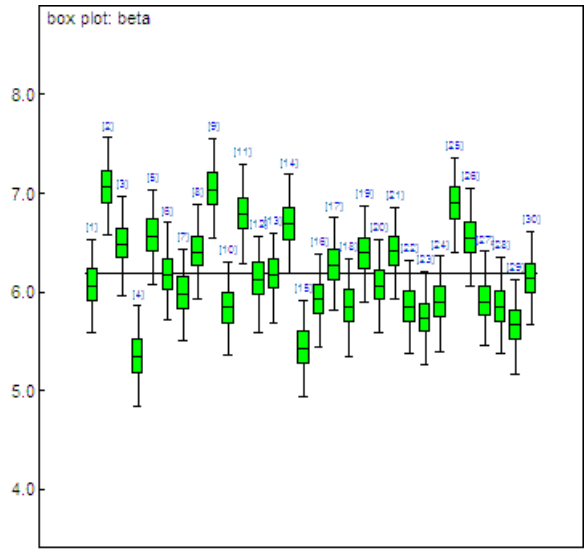
other: where appropriate, *other* defines a vector of reference points to be plotted alongside each element of *node* (on the same scale), for example, *other* may be the observed data in a *model fit* plot (see below). The elements of *other* may be either monitored variables, in which case the posterior mean is plotted, or they may be observed/known.

axis: where appropriate, *axis* defines a set of values against which the elements of *node* (and *other*) should be plotted. Each element of *axis* must be either known/observed or, alternatively, a monitored variable, in which case the posterior mean is used.

Note: *node* , *other* , and *axis* should all have the same number of elements!

beg and ***end:*** are used to select the subset of stored samples from which the desired plot should be derived.

box plot: this command button produces a single plot in which the posterior distributions of all elements of *node* are summarised side by side. For example,



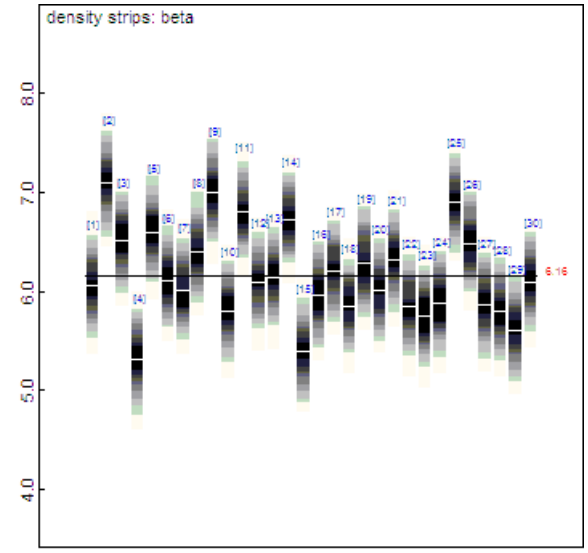
By default, the distributions are plotted in order of the corresponding variable's index in *node* and are also labeled with that index. Boxes represent inter-quartile ranges and the solid black line at the (approximate) centre of each box is the mean; the arms of each box extend to cover the central 95 per cent of the distribution - their ends correspond, therefore, to the 2.5% and 97.5% quantiles. (Note that this representation differs somewhat from the traditional.)

(The default value of the baseline shown on the plot is the global mean of the posterior means.)

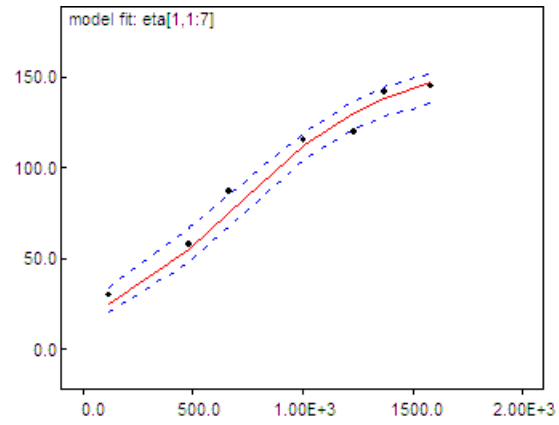
There is a special "property editor" available for box plots, as indeed there is for all graphics generated via the *Comparison Tool* . This can be used to interact with the plot and change the way in which it is displayed, for example, it is possible to rank the distributions by their means or medians and/or plot them on a logarithmic scale.

caterpillar: a "caterpillar" plot is conceptually very similar to a box plot. The only significant differences are that the inter-quartile ranges are not shown and the default scale axis is now the x-axis - each distribution is summarised by a horizontal line representing the 95% interval and a dot to show where the mean is. (Again, the default baseline - in red - is the global mean of the posterior means.) Due to their greater simplicity caterpillar plots are typically preferred over box plots when the number of distributions to be compared is large.

density strips: A density strip is similar to a box plot, but more informative. Instead of just summary statistics, it represents the entire posterior distribution through shading. A density estimate of the distribution is computed, as described in Density plot , and the darkness of the strip at each point is defined as proportional to the estimated density. For a discussion of this shading technique see Jackson (2008) .



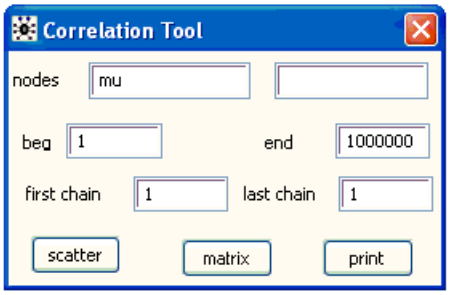
model fit: the elements of *node* (and *other* if specified) are treated as a time-series, defined by (increasing values of) the elements of *axis* . The posterior distribution of each element of *node* is summarised by the 2.5%, 50% and 97.5% quantiles. Each of these quantities is joined to its direct neighbours (as defined by *axis*) by straight lines (solid red in the case of the median and dashed blue for the 95% posterior interval) to form a piecewise linear curve - the 'model fit'. In cases where *other* is specified, its values are also plotted, using black dots, against the corresponding values of *axis* , e.g.



Where appropriate, either or both axes can be changed to a logarithmic scale via a property editor

scatterplot: by default, the posterior means of *node* are plotted (using blue dots) against the corresponding values of *axis* and an exponentially weighted smoother is fitted.

Correlations... [top]



This non-modal dialog box is used to plot out the relationship between the simulated values of selected variables, which must have been monitored.

nodes : scalars or arrays may be entered in each box, and all combinations of variables entered in the two boxes are selected. If a single array is given, all pairwise correlations will be plotted.

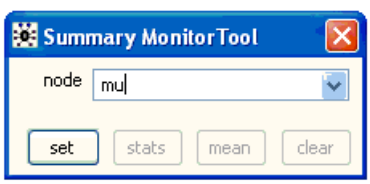
scatter: produces a scatter plot of the individual simulated values.

matrix: produces a matrix summary of the cross-correlations.

print : opens a new window containing the coefficients for all possible correlations among the selected variables.

The calculations may take some time.

Summary... [top]



This non modal dialog box is used to calculate running means, standard deviations and quantiles. Th e commands in this dialog are less powerful and general than those in the *Sample Monitor Too* l, but they also require much less storage (an important consideration when many variables and/or long runs are of interest).

node: The variable of interest must be typed in this text field.

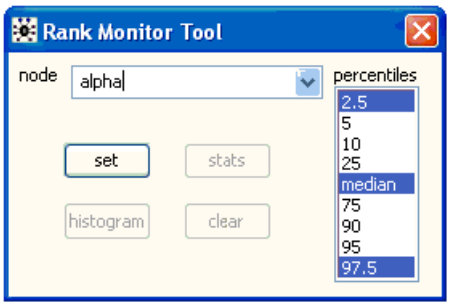
set : starts recording the running totals for *nod* e.

stats **:** displays the running means, standard deviations, and 2.5%, 50% (median) and 97.5% quantiles for *nod* e. **Note that these running quantiles are calculated via an approximate algorithm** (see here for details) **and should therefore be used with caution** .

means : displays the running means for *nod* e in a comma delimited form. This can be useful for passing the results to other statistical or display packages.

clear: removes the running totals for *nod* e.

Rank... [top]



This non-modal dialog box is used to store and display the ranks of the simulated values in an array.

node *:* the variable to be ranked must be typed in this text field (must be an array).

set : starts building running histograms to represent the rank of each component of *nod* e. An amount of storage proportional to the square of the number of components of *nod* e is allocated. Even when *nod* e has thousands of components this can require less storage than calculating the ranks explicitly in the model specification and storing their samples, and it is also much quicker.

stats **:** summarises the distribution of the ranks of each component of the variable *nod* e. The quantiles highlighted in the percentile selection box are displayed.

histogram **:** displays the empirical distribution of the simulated rank of each component of the variable *nod* e.

clear : removes the running histograms for *nod* e.

DIC... [top]

The *DIC Too* l dialog box is used to evaluate the *Deviance Information Criterio* n (DIC; Spiegelhalter et al., 2002) and related statistics - these can be used to assess model complexity and compare different models. Most of the examples packaged with *OpenBUG* S contain an example of their usage.

It is important to note that DIC assumes the posterior mean to be a good estimate of the stochastic parameters. If this is not so, say because of extreme skewness or even bimodality, then DIC may not be appropriate. There are also circumstances, such as with mixture models, in which OpenBUGS will not permit the calculation of DIC and so the menu option is grayed out. Please see the OpenBUGS web-page for current restrictions:

https://openbugs.info

set : starts calculating DIC and related statistics - the user should ensure that convergence has been achieved before pressing *se* t as all subsequent iterations will be used in the calculation.

clear : if a DIC calculation has been started (via *se* t) this will clear it from memory, so that it may be restarted later.

stats : displays the calculated statistics, as described below; see Spiegelhalter et al. (2002) for full details; the section Advanced Use of the BUGS Language also contains some comments on the use of DIC.

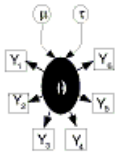
The stats button generates the following statistics:

Dbar : this is the posterior mean of the deviance, which is exactly the same as if the node 'deviance' had been monitored (see here). This deviance is defined as -2 * log(likelihood): 'likelihood' is defined as p(y |*thet* a), where y comprises all stochastic nodes given values (i.e. data), and *thet* a comprises the *stochastic parent* s of y - 'stochastic parents' are the stochastic nodes upon which the distribution of y depends, when collapsing over all logical relationships.

Dhat : this is a point estimate of the deviance (-2 * log(likelihood)) obtained by substituting in the posterior means *theta.ba* r of *thet* a: thus Dhat = -2 * log(p(y |*theta.ba* r)) .

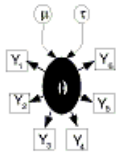
pD : this is 'the effective number of parameters', and is given by **pD = Dbar - Dha** t. Thus pD is the posterior mean of the deviance minus the deviance of the posterior means.

DIC : this is the 'Deviance Information Criterion', and is given by **DIC = Dbar + pD** or
DIC = Dhat + 2 * pD . The model with the smallest DIC is estimated to be the model that would best predict a replicate dataset of the same structure as that currently observed **.**



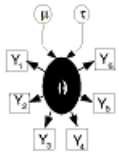
Examples Menu

This menu gives access to the various volumes of examples illustrating the use of the OpenBUGS software. The first group of options opens the contents pages of the example volumes as hyper-text documents. Clicking on the blue links will open further windows displaying the individual examples. The second group of options prints a copy of the examples. If a physical printer is the default printer the result is a paper. If the default printer is set to a file you will get a Postscript or a PDF file. When printing the volume of examples please make sure you do not have that volume open.



Manuals Menu

This menu gives access to the various manuals describing the use of the OpenBUGS software. The first group of options opens the contents pages of the manuals as hyper-text documents. Clicking on the blue links will open further windows displaying the contents of the manuals. The second group of options prints a copy of the manuals. If a physical printer is the default printer the result is a paper. If the default printer is set to a file you will get a Postscript or a PDF file. When printing the manual please make sure you do not have that manual open.



Help Menu

This menu gives short cuts to documentation about distributions and functions that can be used in OpenBUGS models. It also shows a copy of the licence agreement under which the OpenBUGS software is distributed and displays a message box giving a brief summary of the version and history of the software.

OpenBUGS Graphics

Contents

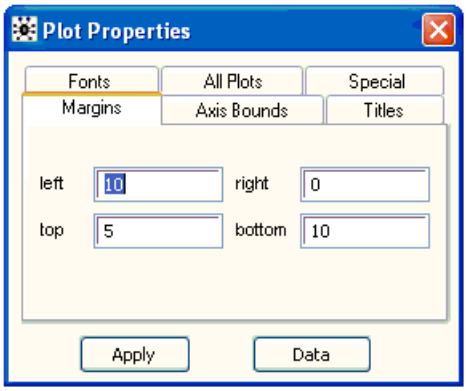
- General properties
- Margins
- Axis Bounds
- Titles
- All Plots
- Fonts
- Specific properties (via Special)
- Density plot
- Box plot
- Caterpillar plot
- Density strips
- Model fit plot
- Scatterplot

General properties [top]

All OpenBUGS graphics have a set of basic properties that can be modified using the *Plot Properties* dialog box. This is opened as follows: first, focus the relevant plot by left-clicking on it; then select *Object Properties...* from the *Edit* menu. Alternatively, right-clicking on a focused plot will reveal a pop-up menu from which *Properties...* may equivalently be selected. The *Plot Properties* dialogue box comprises a "tab-view" and two command buttons, namely *Apply* and *Data...* . The tab-view contains five tabs that allow the user to make different types of modification - these are discussed below. The *Apply* command button applies the properties displayed in the currently selected tab to the focused plot. And *Data...* opens a separate window with the raw data values used to plot the figure in the chosen plot. The other properties of the plot that user can edit are discussed below.

Margins [top]

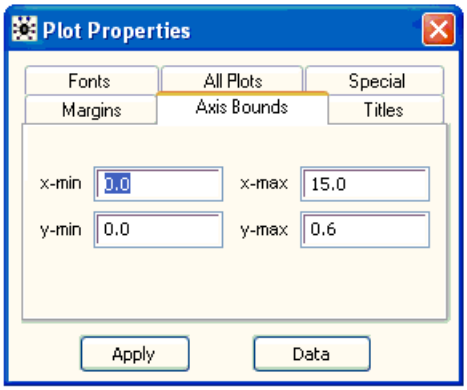
The *Margins* tab displays the plot's left, right, top and bottom margins in millimetres (mm). The left and bottom margins are used for drawing the y- and x-axes respectively. The top margin provides room for the plot's title and the right margin is typically used for plots that require a legend. (Note that top margins (and hence titles) are always inside the plotting rectangle, i.e. there is no gap between the plotting rectangle and the top edge of the graphic.)



In cases where it is appropriate to alter a plot's margins the user may enter his/her preferred values and click on *Apply* to effect the desired change. If the specified values are not appropriate, e.g. if left + right is greater than the width of the graphic (which would result in a plotting rectangle of negative width) or if any margin is negative, etc., then either nothing will happen and the *Margins* tab will reset itself or some form of compromise will be made.

Axis Bounds [top]

The user may specify new minimum and maximum values for either or both axes using the *Axis Bounds* tab (followed by the *Apply* button).

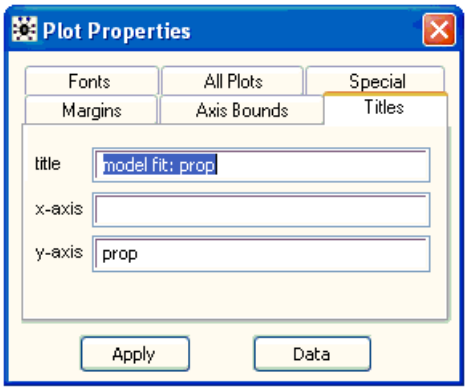


Note that the resulting minima and maxima may not be exactly the same as the values specified because OpenBUGS always tries to ensure that axes range between 'nice' numbers and also have a sufficient number of tick marks. Note also that if max < min is specified then OpenBUGS will ignore it and the *Axis Bounds* tab will reset itself, but there is no guard against specifying a range that does not coincide with the data being plotted, and so the contents of the plot may disappear!

(Some types of plot, such as trace plots, do not allow the user to change their axis bounds, because it would be inappropriate to do so.)

Titles [top]

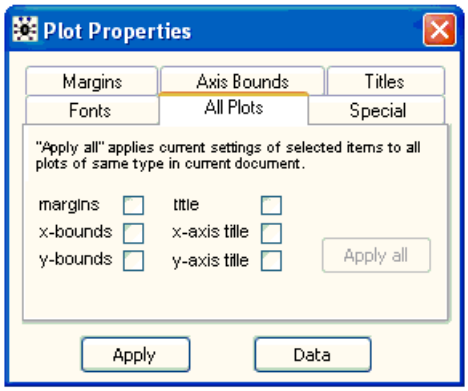
The *Titles* tab should be self-explanatory.



Note, however, that because OpenBUGS does not (yet) support vertical text, a substantial amount of space may be required in the left margin in order to write the y-axis title horizontally - if sufficient space is not available then the y-axis title may not appear at all (clearly, this can be rectified by increasing the left margin).

All Plots [top]

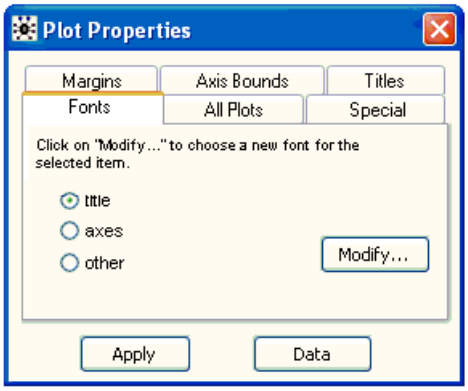
The idea behind the *All Plots* tab is to allow the user to apply some or all of the properties of the focused plot to all plots of the same type (as the focused plot) in the same window (as the focused plot).



The user should first configure the focused plot in the desired way and then decide which of the plot's various properties are to be applied to all plots (of the same type, in the same window). The user should then check all relevant check-boxes and click on *Apply all* . Be careful! It is easy to make mistakes here and there is no undo option - the best advice for users who go wrong is to reproduce the relevant graphics window via the *Sample Monitor Tool* or the *Comparison Tool.*

Fonts [top]

The *Fonts* tab allows the user to change the font of the plot's title, its axes, and any other text that may be present.



First select the font to be modified and click on *Modify...* . (Note that it will only be possible to select *other* if the focused plot has a third font, i.e. if text other than the title and axes is present on the plot, e.g. the labels on a box plot - see Compare...) The self-explanatory *Font* dialogue box should appear - select the required font and click on *OK* (or *Cancel*).

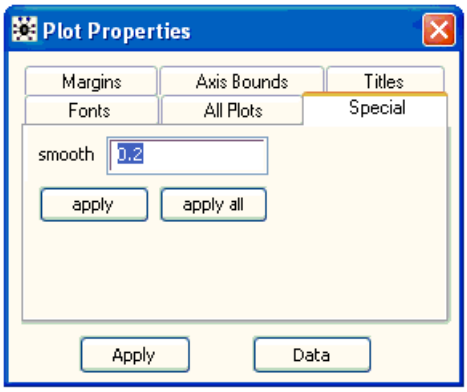
In order to apply the same font to an arbitrarily large group of plots (not necessarily of the same type), rather than using the *All Plots* tab, OpenBUGS uses a "drag-and-pick" facility. First focus a single plot and select which font is to be modified for the whole group: *title* , *axes* , or *other* (if available). Now highlight the group of plots using the mouse. Hold down the ALT key and then the left-hand mouse button and drag the mouse over an area of text with the desired font; then release the mouse button and the ALT key in turn, and the required changes should be made. As an alternative to dragging the mouse over a piece of text with the desired font, the user may instead drag over another plot (even one in the group to be modified) - the group will adopt that plot's properties for the selected font on the *Fonts* tab.

Specific properties (via Special) [top]

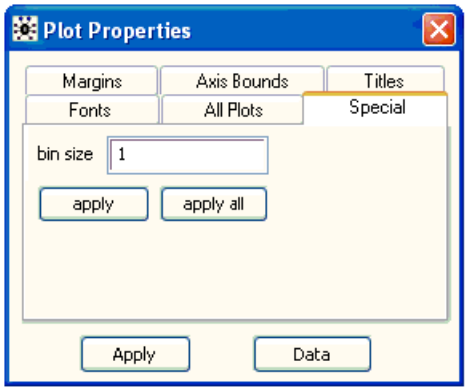
Below we describe the special property editors that are available for certain types of plot. These allow user-interaction beyond that afforded by the standard tabs of the *Plot Properties* dialogue box and are accessed via its *Special* tab.

Density plot [top]

When the *density* button on the *Sample Monitor Tool* is pressed, the output depends on whether the specified variable is discrete or continuous - if the variable is discrete then a histogram is produced whereas if it is continuous a kernel density estimate is produced instead. The specialized property editor that appears when *Special* tab on the *Plot Properties* dialogue box is selected also differs slightly depending on the nature of the specified variable. In both cases the editor comprises a numeric field and two command buttons (*apply* and *apply all*) but in the case of a histogram the numeric field corresponds to the "bin-size" whereas for kernel density estimates it is the smoothing parameter* (see below):



Property editor for histogram



Property editor for kernel density estimate

In either case, the *apply* button sets the bin-size or smoothing parameter of the focused plot to the value currently displayed in the numeric field. The *apply all* button, on the other hand, applies the value currently displayed in the numeric field to *all* plots of the same type (as the focused plot) in the same window (as the focused plot).

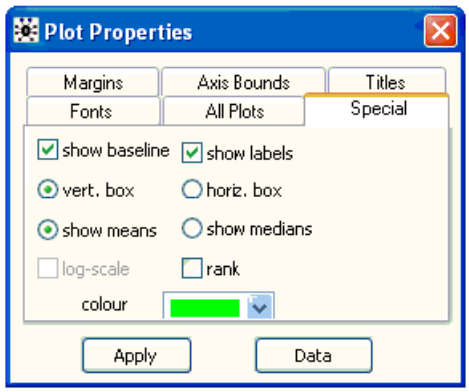
Note *: We define the smoothing parameter for kernel density estimates, s, via the definition of band-width. Suppose our posterior sample comprises m realisations of variable z - denote these by z_i , i = 1,..., m:

band-width = $V^{1/2} / m^{s}$; $V = m^{-1} \sum_i z_i^2 - (m^{-1} \sum_i z_i)^2$

where the summations are from i = 1 to i = m. The default setting for s is 0.2.

Box plot [top]

The *Special* tab of the box plot property editor, is described below:



show baseline: this check-box should be used to specify whether or not a baseline should be shown on the plot.

show labels: check-box that determines whether or not each distribution/box should be labeled with its index in *node* (that is *node* on the *Comparison Tool*). The default setting is that labels should be shown.

show means or ***show medians:*** these radio buttons specify whether the solid black line at the approximate centre of each box is to represent the posterior mean or the posterior median - mean is the default.

rank: use this check-box to specify whether the distributions should be ranked and plotted in order. The basis for ranking is either the posterior mean or the posterior median, depending on which is chosen to be displayed in the plot (via *show means* or *show medians*).

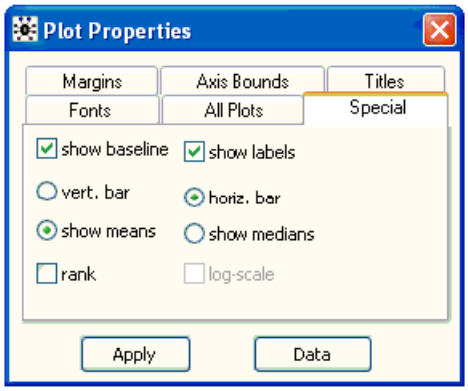
vert. box or ***horiz. box:*** these radio buttons determine the orientation of the plot. The default is "vertical boxes", which means that the scale axis (i.e. that which measures the 'width' of the distributions) is the y-axis.

log-scale: the scale axis can be given a logarithmic scale by checking this check-box.

Finally, at the bottom there is a colour field for selecting the fill-colour of the displayed boxes.

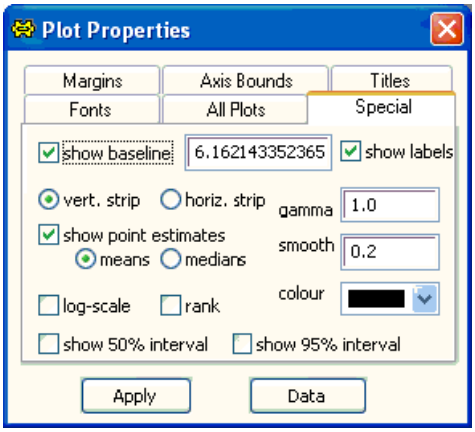
Caterpillar plot [top]

The *Special* tab of the caterpillar plot property editor is virtually identical to that of the box plot property editor except that there is no fill-colour field on the former:



Density strips [top]

The *Special tab* for density strips reveals several ways to customise these plots.



show baseline: specify whether or not a baseline should be shown on the plot. The numeric field to the immediate right of the check-box gives the value of that baseline. By default, a baseline equal to the global mean of the posterior means is shown.

show labels: check-box that determines whether or not each strip should be labelled with its index in *node* (that is *node* on the *Comparison Tool*). The default setting is that labels should be shown.

vert. strip or ***horiz. strip:*** plot vertically or horizontally-aligned strips.

show point estimates: determines whether a point estimate is shown by tick marks on each strip. If this is selected, the definition of the point estimate is defined by the choice of ***mean*** or ***median*** . The estimates are shown as white lines by

default, since the density is usually high, thus the strip is dark, at the posterior mean or median. But if the density is less than 0.2 of the maximum density, then the estimates are shown as black lines on a light strip background.

log scale: display the scale axis on a logarithmic scale.

rank: plot the distributions in the order of their posterior means or medians, depending on the selection in the ***mean*** or ***median*** radio button.

show 50% interval: select to display the 25% and 75% sample quantiles as tick marks on each strip.

show 95% interval: select to display the 2.5% and 97.5% sample quantiles as tick marks on each strip.

gamma: "gamma correction" γ for the variation in shading. γ is a real number greater than 0. The shading at a point x_i with density d_i is defined as a mixture of the colour at the maximum density (black by default) and white, in proportions $p = (d_i / \max_i(d_i))^{\gamma}$ and $1 - p$ respectively. By default, $\gamma = 1$, so that the perceived darkness is proportional to the density.

smooth: the smoothing parameter σ for the density estimate, as described under Density plot **.** This is a value between 0 and 1, with smaller values producing smoother estimates.

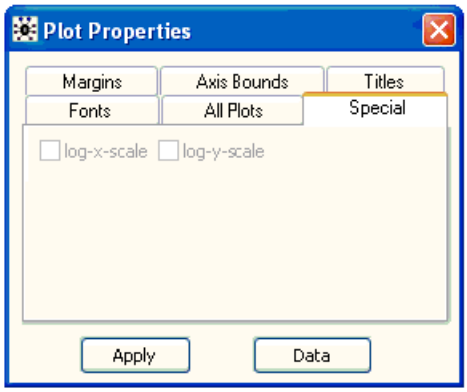
If the data are discrete, then the bin size for the histogram is defined by the closest integer to the band-width implied by the supplied value of σ, or 1 if this is greater. Smaller σ give greater bin sizes, up to a maximum bin size equal to the closest integer to the standard deviation of the sample. Values of σ closer to 1 give smaller bin sizes, up to a minimum bin size of 1, or of the closest integer to the standard deviation divided by the sample size if this is greater.

To update the plot after typing in a new smoothing parameter, click *Apply* . The plot does not update automatically, since recalculating the density may be slow for large samples.

Colour: In the right-hand side of the dialog there is a colour field for selecting the colour at the point of maximum density. The colours for the rest of the strip are defined by interpolating between this colour and white, which represents zero density, so that the darkness of the strip increases with the density.

Model fit plot [top]

In cases where an axis is defined on a strictly positive range, it may be given a logarithmic scale by checking the appropriate check-box:



Scatterplot [top]

With a scatterplot focused, select the *Special* tab on the *Plot Properties* dialogue box to obtain the following property editor:

show means or ***show medians:*** these radio buttons determine whether it is the posterior means or medians of *node* that are plotted/scattered against *axis* to form the plot (that is *node* and *axis* on the *Comparison Tool*) - the default is means.

Immediately beneath the ***show means*** or ***show medians*** radio buttons is a colour field for selecting the colour of the scattered points.

log-x-/log-y-scale: either or both axes may be given a logarithmic scale (assuming they are defined on strictly positive ranges) by checking the appropriate check-box(es).

show bars: 95 per cent posterior intervals (2.5% - 97.5%) for *node* will be shown on the plot (as vertical bars) if this box is checked.

show line: use this check-box to specify whether or not a reference line (either a straight line specified via its intercept and gradient or an exponentially weighted smoother - see below) is to be displayed.

To the immediate right of the ***show line*** check-box is a colour field that determines the reference line's colour (if displayed).

linear or ***smooth:*** these radio buttons are used to specify whether the reference line (if it is to be displayed) should be linear or whether an exponentially weighted smoother should be fitted instead. In the case where a linear line is required, its intercept and gradient should be entered in the two numeric fields to the right of the *linear* radio button (in that order). If a smoother is required instead, then the desired degree of smoothing* (see below) should be entered in the numeric field to the right of the *smooth* radio button. To save unnecessary redrawing of the plot as the various numeric parameters are changed, the *Redraw line* command button is used to inform OpenBUGS when all alterations to the parameters have been completed.

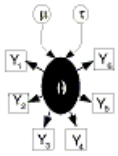
Note *: The exponentially weighted smoother that OpenBUGS uses on scatterplots is defined as:

$$s_i = {}_{S\,j}\, w_{ij}\, y_j \,/\, {}_{S\,j}\, w_{ij} \quad i = 1,..., n$$

where n is the number of scattered points and the summations are from j = 1 to j = n. The weights w_{ij} are given by

$$w_{ij} = \exp(-|x_i - x_j| / s)$$

where s is the smoothing parameter defined in the *Scatterplot properties* dialogue box (next to the *smooth* radio button). The default setting for s is (somewhat arbitrarily) {max(x) - min(x)} / 20.



Tips and Troubleshooting

Contents

- Restrictions when modelling
- Some error messages
- Trap windows
- The program hangs
- Speeding up sampling
- Improving convergence

Restrictions when modelling [top]

Restrictions have been stated throughout this manual. A summary list is as follows:

a) Each stochastic and logical node must appear once and only once on the left-hand-side of an expression. The only exception is when carrying out a data transformation (see Data transformations). This means, for example, that it is generally not possible to give a distribution to a quantity that is specified as a logical function of an unknown parameter.

b) Truncated sampling distributions cannot be handled using the C(.,.) construct - see BUGS language: stochastic nodes.

c) Multivariate distributions: Wishart distributions may only be used as conjugate priors and must have known parameters; multivariate normal and Student-t distributions can be used anywhere in the graph (i.e. as prior or likelihood) and there are no restrictions regarding their parameters. See BUGS language: stochastic nodes .

d) Logical nodes cannot be given data or initial values. This means, for example, that it is not possible to model observed data that is the sum of two random variables. (See Logical nodes .)

Some error messages [top]

In general OpenBUGS tries to give informative, easily understood error messages when there is a problem with the model specification, the data or initial values, or if problems occur with the sampling algorithms. However sometimes numerical problems occur with the sampling algorithms, when this happens a trap message is produced providing a detailed trace back to where the error occurs.

When an error occurs in the syntax of the model specification or in the data or initial values the position of the error will be marked by changing the nearest symbol in the relevant window to inverse video, you must make sure the title bar of this window is selected to be able locate the position of the error. Errors can still occur with a model even if it is syntactically correct. When this happens the error message contains the name of the variable for which the error occurs.

Some common error messages are:

a) **'expected variable name'** indicates an inappropriate variable name, for example a variable name in a data set is not a variable name in the model.

b) **'linear predictor in probit regression too large'** indicates numerical overflow. See possible solutions below for Trap 'undefined real result'.

c) **'logical expression too complex'** - a logical node is defined in terms of too many parameters/constants or too many operators: try introducing further logical nodes to represent parts of the overall calculation; for example, a1 + a2 + a3 + b1 + b2 + b3 could be written as A + B where A and B are the simpler logical expressions a1 + a2 + a3 and b1 + b2 + b3, respectively. Note that linear predictors with many terms should be formulated by 'vectorizing' parameters and covariates and by then using the inprod(.,.) function (see Logical nodes).

d) **'invalid or unexpected token scanned' -** the parser has encountered an unexpected symbol, for example if using a

Doodle check that the *value* field of a logical node has been completed.

e) **'unable to choose update method'** indicates that a restriction in the program has been violated - see Restrictions when modelling above.

f) **'undefined variable'** - undefined variables are allowed in the model so long as they are not used on the right hand side of a relation.

g) **'index out of range'** - usually indicates that a loop-index goes beyond the size of a vector (or matrix dimension).

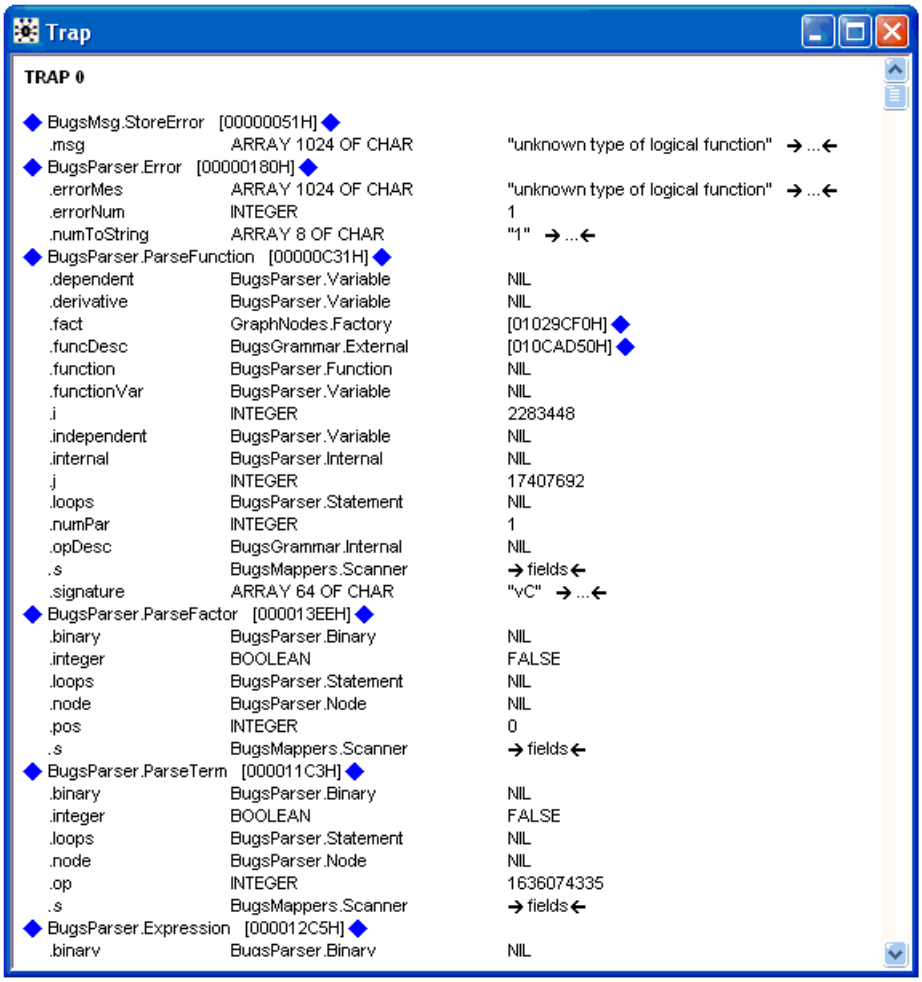
In rare cases *OpenBUGS* is unable to find a suitable sampling algorithm for a node in the model. The error message **'Unable to choose update method for node x'** will be displayed where x is the node for which *OpenBUGS* can not find an update method.

Trap windows [top]

Trap windows occur when a problem with the BUGS software is detected by the run time system. The run time system is a small piece of software that watches the main program (OpenBUGS) and reports if some illegal action occurs. The first line of the trap windows gives a description of the type of illegal action detected by the run time system. Note that a trap window is different form an error message. It can be thought of as an error in the use of the OpenBUGS software that should have produced an error message but for which the error handling code does not exist / work.

The trap window contains detailed information to help locate the position of the problem in the source code plus information about what was happening at the time the error occurred. The information in the trap window is arranged in three columns, each column can contain blue diamonds (see the developer manual for more details on blue diamonds). The first column contains procedure names and underneath the procedure name local variables in that procedure. The second column contains information on the type of the local variable. The third column gives the value of the local variable. The first procedure in the left hand column is where the error causing the trap occurred, clicking on the blue diamond in the second column on a level with this procedure name will open a window showing the source code of the procedure and the exact position where the error occurred. The second procedure name in the first column is the procedure which called the procedure where the error occurred and so on. The blue diamonds on the left side of the first column give information about global variables in the module which encloses the procedure, clicking on these blue diamonds will open a new window showing the global variable. Blue diamonds to the right of the third column give information about pointer type variables, while the folds like fields give information about structured variables.

A section of a typical trap widow is shown below (in this case TRAP 0 is a special, deliberate -that is programmed - error).



Some common illegal actions detected by the run time system are:

a) **'undefined real result'** indicates numerical overflow. Possible reasons include:
- initial values generated from a 'vague' prior distribution may be numerically extreme - specify appropriate initial values;
- numerically impossible values such as log of a non-positive number - check, for example, that no zero expectations have been given when Poisson modelling;
- numerical difficulties in sampling. Possible solutions include:
- better initial values;
- more informative priors - uniform priors might still be used but with their range restricted to plausible values;
- better parameterisation to improve orthogonality;
- standardisation of covariates to have mean 0 and standard deviation 1.
- can happen if all initial values are equal.

b) **'index array out of range' -** possible reasons include:
- attempting to assign values beyond the declared length of an array;

c) **'stack overflow'** can occur if there is a recursive definition of a logical node.

d) **'NIL dereference (read)'** can occur at compilation in some circumstances when an inappropriate transformation is made, for example an array into a scalar.

e) **argument of sqrt must not be negative** the square root of a negative number has been taken, this occurs if the precision matrix of a multivariate normal is not positive definite.

The program hangs [top]

This could be due to:

a) a problem that seems to happen with NT - rebooting is a crude way out;
b) interference with other programs running - try to run WinBUGS on its own;
c) a particularly ill-posed model - try the approaches listed above under Trap messages.

Speeding up sampling [top]

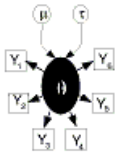
The key is to reduce function evaluations by expressing the model in as concise a form as possible. For example, take advantage of the functions provided and use nested indexing wherever possible. Look at the packaged examples and others provided on users' web sites.

Improving convergence [top]

Possible solutions include:

a) better parameterisation to improve orthogonality of joint posterior;
b) standardisation of covariates to have mean 0 and standard deviation 1;
c) use of ordered over-relaxation.

For other problems, try the FAQ pages of the web site (http://openbugs.info).



Advanced Use of the BUGS Language

Contents

- Generic sampling distribution
- Specifying a new prior distribution
- Using pD and DIC
- Mixtures of models of different complexity
- Where the size of a set is a random quantity
- Assessing sensitivity to prior assumptions
- Modelling unknown denominators
- Handling unbalanced datasets
- Use of the "cut" function

Generic sampling distribution [top]

Suppose we wish to use a sampling distribution that is not included in the standard distributions (see Appendix I Distributions), in which an observation x[i] contributes a likelihood term L[i] (a function of x[i]). We may use the 'loglik' distribution dloglik, for a dummy observed variable:

```
dummy[i] <- 0
dummy[i] ~dloglik(logLike[i])
```

where logLike[i] is the contribution to the log-likelihood for the ith observation. The dloglik function implements the 'zero poisson' method utilized in WinBUGS.

This is illustrated in the example below in which a normal likelihood is constructed and the results are compared to the standard formulation.

```
model {
  for (i in 1:7) {
    dummy[i] <- 0
    dummy[i] ~ dloglik(logLike[i])        # likelihood is exp(logLike[i])
      # log(likelihood)
    logLike[i] <- -log(sigma) - 0.5 * pow((x[i] – mu) / sigma, 2)
  }
  mu ~ dunif(-10, 10)
  sigma ~ dunif(0, 10)
}
```

or

```
model {
  #      check using normal distribution
  for (i in 1:7) {
    x[i] ~ dnorm(mu, prec)
  }
  prec <- 1 / (sigma * sigma)
  mu ~ dunif(-10, 10)
  sigma ~ dunif(0, 10)
}
```

Data:

```
list(x = c(-1, -0.3, 0.1, 0.2, 0.7, 1.2, 1.7))
```

Initial values:

```
list(sigma = 1, mu = 0)
```

Results:

	mean	sd	MC_error	val2.5pc	median	val97.5pc	start	sample
mu	0.3763	0.447	0.01203	-0.5436	0.3789	1.302	1001	9000
sigma	1.145	0.4368	0.01743	0.6113	1.047	2.263	1001	9000

or

	mean	sd	MC_error	val2.5pc	median	val97.5pc	start	sample
mu	0.3642	0.494	0.004657	-0.6213	0.3663	1.342	1001	9000
sigma	1.208	0.5572	0.01174	0.6299	1.083	2.486	1001	9000

Specifying a new prior distribution [top]

For a parameter *theta* , if we want to use a prior distribution not included in the standard distributions (see Appendix I Distributions) , then we can use the 'dloglik' distribution (see above) for the prior specification. A single dummy observation contributes the appropriate term to the likelihood for *theta* ; and when it is combined with a 'flat' prior for *theta,* the correct distribution results:

```
theta ~ dflat()
dummy ~dloglik(logLike)
logLike <- log( desired prior for theta )
```

This is illustrated by the below example in which a normal prior is constructed and the results are compared to the standard formulation. It is important to note that this method will cause the theta variable to be sampled using the metropolis algorithm and may lead to poor convergence and a high Monte Carlo error.

```
model {
   for (i in 1:7) {
     x[i] ~ dnorm(mu, prec)
   }
   dummy <- 0
   dummy ~ dloglik(phi)         # likelihood is exp(phi)
   phi <- -0.5 * pow(mu, 2)         # log(N(0, 1))
   mu ~ dflat()         # 'flat' prior
   prec <- 1 / (sigma * sigma)
   sigma ~ dunif(0, 10)
  }
```

or

```
model {
   for (i in 1:7) {
     x[i] ~ dnorm(mu, prec)
   }
      mu ~ dnorm(0, 1)         # 'known' normal prior
   prec <- 1 / (sigma * sigma)
   sigma ~ dunif(0, 10)
  }
```

Data:

```
list(x = c(-1, -0.3, 0.1, 0.2, 0.7, 1.2, 1.7))
```

Initial values:

```
list(sigma = 1, mu = 0)
```

Results:

	mean	sd	MC_error	val2.5pc	median	val97.5pc	start	sample
mu	0.2859	0.4108	0.008778	-0.5574	0.3002	1.08	1001	9000
sigma	1.162	0.4721	0.008885	0.6178	1.05	2.359	1001	9000

or

	mean	sd	MC_error	val2.5pc	median	val97.5pc	start	sample
mu	0.2973	0.4043	0.003525	-0.5499	0.3112	1.07	1	10000

sigma 1.166 0.4785 0.007654 0.6234 1.059 2.315 1 10000

Using pD and DIC [top]

Here we make a number of observations regarding the use of DIC and pD - for a full discussion see Spiegelhalter et al. (2002):

1) DIC is intended as a generalisation of Akaike's Information Criterion (AIC). For non-hierarchical models, pD should be approximately the true number of parameters.

2) Slightly different values of Dhat (and hence pD and DIC) can be obtained depending on the parameterisation used for the prior distribution. For example, consider the precision *tau* (1 / variance) of a normal distribution. The two priors

```
tau ~ dgamma(0.001, 0.001) and
log.tau ~ dunif(-10, 10); log(tau) <- log.tau
```

are essentially identical but will give slightly different results for Dhat. The first prior distribution has stochastic parent *tau* and hence the posterior mean of *tau* is substituted in Dhat, while in the second parameterisation the stochastic parent is *log.tau* and hence the posterior mean of log(*tau*) is substituted in Dhat.

3) For sampling distributions that are log-concave in their stochastic parents, pD is guaranteed to be positive (provided the simulation has converged). However, we have obtained negative pD's in the following situations:

i) with non-log-concave likelihoods (e.g. Student-t distributions) when there is substantial conflict between prior and data;
ii) when the posterior distribution for a parameter is symmetric and bimodal, and so the posterior mean is a very poor summary statistic and gives a very large deviance.

4) No MC error is available on the DIC. MC error on Dbar can be obtained by monitoring deviance and is generally quite small. The primary concern is to ensure convergence of Dbar - it is therefore worthwhile checking the stability of Dbar over a long chain.

5) The minimum DIC estimates the model that will make the best short-term predictions, in the same spirit as Akaike's criterion. However, if the difference in DIC is, say, less than 5, and the models make very different inferences, then it could be misleading just to report the model with the lowest DIC.

6) DICs are comparable only over models with exactly the same observed data, but there is no need for them to be nested.

7) DIC differs from Bayes factors and BIC in both form and aims.

8) Caution is advisable in the use of DIC until more experience has been gained. **It is important to note that the calculation of DIC will be disallowed for certain models.**

Mixtures of models of different complexity [top]

Suppose we assume that each observation, or group of observations, is from one of a set of distributions, where the members of the set have different complexity. For example, we may think data for each person's growth curve comes from either a linear or quadratic line. We might think we would require 'reversible jump' techniques, but this is not the case as we are really only considering a single mixture model as a sampling distribution. Thus standard methods for setting up mixture distributions can be adopted, but with components having different numbers of parameters.

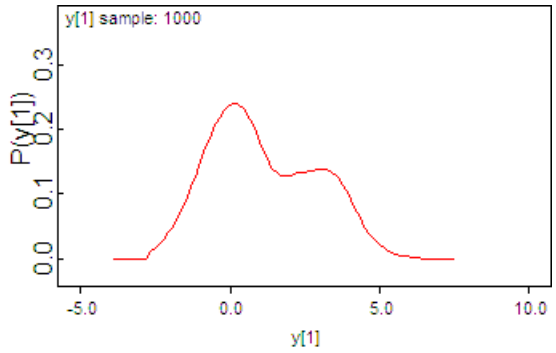
The below example illustrates how this is handled in *OpenBUGS* , using a set of simulated data.

Suppose that for each i: x[i] ~ N(mu, 1) with probability p; and x[i] ~ N(0, 1) otherwise (i.e. with probability 1 - p). We generate 100 observations with p = 0.4 and mu = 3 as follows. We forward sample *once* from the model below by compiling the code and then using the 'gen inits' facility. The simulated data can then be obtained by selecting *Save State* from the *Model* menu.

```
model {
  mu <- 3
  p <- 0.4
  m[1] <- 0
  m[2] <- mu
  for (i in 1 : 100) {
    group[i] ~ dbern(p)
    index[i] <- group[i] + 1
```

```
    y[i] ~ dnorm(m[index[i]], 1)
  }
}
```

We may observe the underlying mixture distribution by monitoring any one of the y[i]'s over a number of additional sampling cycles. For example, the following kernel density plot was obtained after monitoring y[1] for 1000 iterations:



To analyse the simulated data we use the following code:

```
model {
  mu ~ dunif(-5, 5)
  p ~ dunif(0, 1)
  m[1] <- 0
  m[2] <- mu
  for (i in 1:100) {
    group[i] ~ dbern(p)
    index[i] <- group[i] + 1
    y[i] ~ dnorm(m[index[i]], 1)
  }
}
```

After 101000 iterations (with a burn-in of 1000) we have good agreement with the 'true' values:

	mean	sd	MC_error	val2.5pc	median	val97.5pc	start	sample
mu	3.058	0.2071	0.001288	2.655	3.056	3.472	1001	100000
p	0.4243	0.05985	3.718E-4	0.3098	0.4232	0.5438	1001	100000

Initial values :

```
list(mu = 0, p = 0.5)
```

Simulated data:

```
list(
y = c(
2.401893189187883,0.2400077667887621,-0.1556589480882761,-0.8457182007723154,0.37008097263224,
3.586009960655263,1.598955590680827,3.826518138558907,-0.9630895329522409,0.6951468806412424,
3.129328672725175,0.01025316168135796,0.4887298480200992,0.3865632519840305,-0.2697534502300845,
-1.18891944058751,4.654771935717583,0.8063807170988319,-0.9867060769784521,0.9154433557950319,
-0.5419217214549653,3.358942981432516,3.33734145389337,3.3960739633218,2.038185222382867,
5.241414085016386,3.362823353717864,-0.6013483154102028,0.4414804913168 43,2.96228336554288,
-2.278054802181326,1.446861613005477,1.49864667127073,2.819410923921955,3.668112206659865,
0.8253991892717565,1.117718710956935,3.976040128045549,1.261678474198661,-0.03343173803015926,
-0.4908566523519207,0.3532664087054739,-1.679362022373703,-1.555760053262808,1.213022071911081,
3.421072023150202,2.523431239569371,-1.218844344999495,-0.270208787763775,-0.1217560919494103,
2.033835091596821,0.4654798734609423,-0.7231540561359688,2.146640407714382,4.286633169106659,
1.445348149793759,0.180718235361594,1.527791426438174,-1.010060680847808,1.969758236040937,
-0.3553936244225268,1.465488166547136,3.32669874753109,1.061348836020805,2.31746192198435,
0.9564080472865206,1.877477903911581,-0.6964242592539615,5.695159167887606,2.807268123194206,
-1.69815612298043,0.1881330355284009,0.04018232765300667,2.272096174325846,0.9694913345710192,
3.979152197443702,-0.7028546956095989,1.371423010966528,1.646618045628321,0.1919331499516834,
3.587928853903195,0.1219688057614579,2.570950727333546,1.888563572434331,1.10249950266553,
4.119103539377994,3.824513529111282,1.509248826260637,1.156700011660602,-0.0343697724869282,
3.816521442901518,0.0675520018218364,-0.002197033376085265,2.994147650487649,1.856351699226103,
```

3.539838516568062,0.5861211569557628,4.343058450523658,0.1760647082096519,3.678908111531086))

Naturally, the standard warnings about mixture distributions apply, in that convergence may be poor and careful parameterisation may be necessary to avoid some of the components becoming empty.

Where the size of a set is a random quantity [top]

Suppose the size of a set is a random quantity: this naturally occurs in 'changepoint' problems where observations up to an unknown changepoint K come from one model, and after K come from another. Note that we **cannot** use the construction

```
for (i in 1:K) {
    y[i] ~ model 1
}
for (i in (K + 1):N) {
    y[i] ~ model 2
}
```

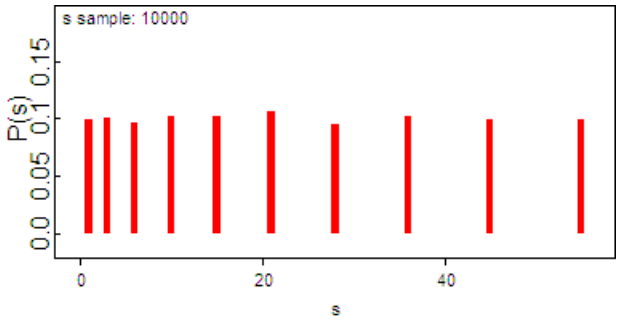
since the index for a loop cannot be a random quantity. Instead we can use the step function to set up an indicator as to which set each observation belongs to:

```
for (i in 1:N) {
    ind[i] <- 1 + step(i - K - 0.01)     # will be 1 for all i <= K, 2 otherwise
    y[i] ~ model ind[i]
}
```

This is illustrated by the problem of adding up terms in a series of unknown length.

Suppose we want to find the distribution of the sum of the first K integers, where K is a random quantity. We shall assume K has a uniform distribution on 1 to 10.

```
model
{
  for (i in 1:10) {
    p[i] <- 1 / 10        # set up prior for K
    x[i] <- i        # set up array of integers
  }
  K ~ dcat(p[])        # sample K from its prior
  for (i in 1:10) {
# determine which of the x[i]'s are to be summed
    xtosum[i] <- x[i] * step(K - i + 0.01)
  }
  s <- sum(xtosum[])
}
```



Assessing sensitivity to prior assumptions [top]

One way to do this is to repeat the analysis under different prior assumptions, but within the same simulation in order to aid direct comparison of results. Assuming the consequences of K prior distributions are to be compared:

a) replicate the dataset K times within the model code;
b) set up a loop to repeat the analysis for each prior, holding results in arrays;
c) compare results using the 'compare' facility.

The example prior-sensitivity explores six different suggestions for priors on the random-effects variance in a meta-analysis.

Modelling unknown denominators [top]

Suppose we have an unknown Binomial denominator for which we wish to express a prior distribution. It can be given a Poisson prior but this makes it difficult to express a reasonably uniform distribution. Alternatively a continuous distribution could be specified and then the 'round' function used. For example, suppose we are told that a fair coin has come up heads 10 times - how many times has it been tossed?

```
model {
    r <- 10
    p <- 0.5
    r ~ dbin(p, n)
    n.cont ~ dunif(1, 100)
    n <- round(n.cont)
}
```

node	mean	sd	MC error	2.5%	median	97.5%	start	sample
n	21.08	4.794	0.07906	13.0	21.0	32.0	1001	5000
n.cont	21.08	4.804	0.07932	13.31	20.6	32.0	1001	5000

Assuming a uniform prior for the number of tosses, we can be 95% sure that the coin has been tossed between 13 and 32 times. A discrete prior on the integers could also have been used in this context.

Handling unbalanced datasets [top]

Suppose we observe the following data on three individuals:

Person 1: 13.2
Person 2: 12.3 , 14.1
Person 3: 11.0, 9.7, 10.3, 9.6

There are three different ways of entering such 'ragged' data into WinBUGS:

1. Fill-to-rectangular: Here the data is 'padded out' by explicitly including the missing data, i.e.

```
y[,1]   y[,2]   y[,3]   y[,4]
13.2    NA      NA      NA
12.3    14.1    NA      NA
11.0    9.7     10.3    9.6
END
```

or list(y = structure(.Data = c(13.2, NA, NA, NA, 12.3, 14.1, NA, NA, 11.0, 9.7, 10.3, 9.6), .
Dim = c(3, 4)).

A model such as y[i, j] ~ dnorm(mu[i], 1) can then be fitted. This approach is inefficient unless one explicitly wishes to estimate the missing data.

2. Nested indexing: Here the data are stored in a single array and the associated person is recorded as a factor, i.e.

```
y[]   person[]
13.2   1
12.3   2
14.1   2
11.0   3
9.7    3
10.3   3
9.6    3
END
```

or list(y = c(13.2, 12.3, 14.1, 11.0, 9.7, 10.3, 9.6), person = c(1, 2, 2, 3, 3, 3, 3)).

A model such as y[k] ~ dnorm(mu[person[k]], 1) can then be fitted. This seems an efficient and clear way to handle the problem.

3. Offset: Here an 'offset' array holds the position in the data array at which each person's data starts. For example, the data might be

list(y = c(13.2, 12.3, 14.1, 11.0, 9.7, 10.3, 9.6), offset = c(1, 2, 4, 8))

and lead to a model containing the code

```
for (k in offset[i]:(offset[i + 1] - 1)) {
  y[k] ~ dnorm(mu[i], 1)
}
```

The danger with this method is that it relies on getting the offsets correct and they are difficult to check.

The three methods are illustrated in the small example below.

Fill-to-rectangular:

```
model {
  for (i in 1:3) {
    mu[i] ~ dunif(0, 100)
    for (j in 1:4) {
      y[i, j] ~ dnorm(mu[i], 1)
    }
  }
}
```

```
y[, 1]  y[, 2]  y[, 3]  y[, 4]
13.2  NA  NA  NA
12.3  14.1  NA  NA
11.0  9.7  10.3  9.6
END
```

```
list(y = structure(.Data = c(13.2, NA, NA, NA, 12.3, 14.1, NA, NA, 11.0, 9.7, 10.3, 9.6), .Dim = c(3, 4))
```

Nested indexing:

```
model
 {
  for (i in 1:3) {
    mu[i] ~ dunif(0, 100)
  }
  for (k in 1:7) {
    y[k] ~ dnorm(mu[person[k]], 1)
  }
}
```

```
y[]  person[]
13.2  1
12.3  2
14.1  2
11.0  3
9.7  3
10.3  3
9.6  3
END
```

```
list(y = c(13.2, 12.3, 14.1, 11.0, 9.7, 10.3, 9.6), person = c(1, 2, 2, 3, 3, 3, 3))
```

Offset:

```
model {
  for (i in 1:3) {
    mu[i] ~ dunif(0, 100)
    for (k in offset[i]:(offset[i + 1] - 1)) {
      y[k] ~ dnorm(mu[i], 1)
    }
```

```
  }
}

y[]
13.2
12.3
14.1
11.0
9.7
10.3
9.6
END

offset[]
1
2
4
8
END
```

```
list(y = c(13.2, 12.3, 14.1, 11.0, 9.7, 10.3, 9.6), offset = c(1, 2, 4, 8))
```

Use of the "cut" function [top]

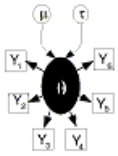
Suppose we observe some data that we do not wish to contribute to the parameter estimation and yet we wish to consider as part of the model. This might happen, for example:

a) when we wish to make predictions on some individuals on whom we have observed some partial data that we do not wish to use for parameter estimation;
b) when we want to use data to learn about some parameters and not others;
c) when we want evidence from one part of a model to form a prior distribution for a second part of the model, but we do not want 'feedback' from this second part.

The "cut" function forms a kind of 'valve' in the graph: prior information is allowed to flow 'downwards' through the cut, but likelihood information is prevented from flowing upwards.

For example, the following code leaves the distribution for theta unchanged by the observation y.

```
model
{
  y <- 2
  y ~ dnorm(theta.cut, 1)
  theta.cut <- cut(theta)
  theta ~ dnorm(0, 1)
}
```



BUGS

References

Best N G, Cowles M K and Vines S K (1997) *CODA: Convergence diagnosis and output analysis software for Gibbs sampling output, Version 0.4* . MRC Biostatistics Unit, Cambridge:
http://www.mrc-bsu.cam.ac.uk/bugs/classic/coda04/readme.shtml

Breslow N E and Clayton D G (1993) Approximate inference in generalized linear mixed models. *Journal of the American Statistical Association* . **88** , 9-25.

Brooks S P (1998) Markov chain Monte Carlo method and its application. *The Statistician* . **47** , 69-100.

Brooks S P and Gelman A (1998) Alternative methods for monitoring convergence of iterative simulations. *Journal of Computational and Graphical Statistics* . **7** , 434-455.

Carlin B P and Louis T A (1996) *Bayes and Empirical Bayes Methods for Data Analysis* . Chapman and Hall, London, UK.

Congdon P (2001) *Bayesian Statistical Modelling* . John Wiley & Sons, Chichester, UK.

Crowder M J (1978) Beta-binomial Anova for proportions. *Applied Statistics* . **27** , 34-37.

Gamerman D (1997) Sampling from the posterior distribution in generalized linear mixed models. *Statistics and Computing* . **7** , 57-68.

Gelman A, Carlin J C, Stern H and Rubin D B (1995) *Bayesian Data Analysis.* Chapman and Hall, New York.

Gilks W (1992) Derivative-free adaptive rejection sampling for Gibbs sampling. In *Bayesian Statistics 4* , (J M Bernardo, J O Berger, A P Dawid, and A F M Smith, eds), Oxford University Press, UK, pp. 641-665.

Gilks W R, Richardson S and Spiegelhalter D J (Eds.) (1996) *Markov chain Monte Carlo in Practice* . Chapman and Hall, London, UK.

Jackson C H (2008) Displaying uncertainty with shading. *The American Statistician.* **62** , 340-347.

Neal R (1997) Markov chain Monte Carlo methods based on 'slicing' the density function. *Technical Report 9722* , Department of Statistics, University of Toronto, Canada:
http://www.cs.utoronto.ca/~radford/publications.html

Neal R (1998) Suppressing random walks in Markov chain Monte Carlo using ordered over-relaxation. In *Learning in Graphical Models* , (M I Jordan, ed). Kluwer Academic Publishers, Dordrecht, pp. 205-230.
http://www.cs.utoronto.ca/~radford/publications.html

Roberts G O (1996). Markov chain concepts related to sampling algorithms. In W R Gilks, S Richardson and D J Spiegelhalter (Eds.) *Markov chain Monte Carlo in Practice* . Chapman and Hall, London, UK.

Spiegelhalter D J, Best N G, Carlin B P and van der Linde A (2002) Bayesian measures of model complexity and fit (with discussion). *J. Roy. Statist. Soc. B* . **64** , 583-640.

Tierney L (1983) A space-efficient recursive procedure for estimating a quantile of an unknown distribution. *SIAM J. Sci. Stat. Comput* . **4** , 706-711.