1 Reference manual for BUGS language syntax

2 Introduction

The BUGS language facilitates a declarative, textual description of the probability model whereby the relationship between each node and its parents is stated explicitly; the software uses this to construct an internal, graphical representation of the model. Note that the declarative nature of the language means that the order in which the various relationships are specified is irrelevant. There are two possible types of relation:

- `~` means 'is distributed as' and denotes a *stochastic* relation;
- `<-` means 'is to be replaced by' and denotes a *logical* relation.

Nodes on the left of a `~` are stochastic nodes and those on the left of a `<-` are logical (or deterministic) nodes. In addition, a model may include *constant* nodes, which are fixed by the design of the study and have no parents – their values must be specified as part of the data set (see § **??**). Note that generally speaking, each logical or stochastic node should appear once and only once on the left-hand-side of a statement (although see § 8 for exceptions to this rule). This chapter provides an overview of the syntax available for fully specifying each parent-child relationship.

3 Distributions

3.1 Standard distributions

Lists of continuous and discrete distributions available in BUGS , along with their full definitions and examples of their usage, are given in Table 1 and Table 2, respectively. Note that the parameters of distributions, *i.e.* quantities occurring on the right-hand-side of a `~` relation, must be named-nodes or numerical values rather than expressions. For example, we cannot specify `x ~ dnorm(a + b*z, 1)`, say; instead, we must use something like:

```
x   ~ dnorm(mu, 1)
mu <- a + b*z
```

3.2 Censoring and truncation

Suppose a stochastic quantity x has been observed to lie in the interval (a, b). We can specify such *interval censoring* by following the distributional expression for x with `I(a, b)`. That is,

```
x ~ ddist(psi)I(a, b)
```

where `psi` ($\boldsymbol{\psi}$) denotes a generic set of distributional parameters. Note that censoring bounds must be named-nodes or numerical values rather than expressions, and that leaving either censoring bound blank simply corresponds to no limit, *e.g.* `I(a, )` designates a quantity that has been observed to be greater than a, but not less than any specific amount.

It is vital to understand that use of the `I(.,.)` construct is *not* the same as specifying a truncated distribution. The sole effect of the `I(.,.)` expression is to ensure that all values sampled for x outside the specified interval are rejected. This may seem like an appropriate strategy for sampling from truncated distributions, but the full conditional distribution derived for $\boldsymbol{\psi}$ is inconsistent with truncation. To see this, note that the density of a truncated distribution, when considered as a function of its parameters ($\boldsymbol{\psi}$, as opposed to x), is *not* simply proportional to the untruncated density. Instead, it is normalized by the integral of the untruncated density over the truncation interval, which is a complex function of $\boldsymbol{\psi}$ (and a and b). BUGS does not take account of this normalizing constant when deriving the full conditional for $\boldsymbol{\psi}$ as it is inappropriate to do so in the case of interval censoring – the correct likelihood contribution is the untruncated density.

Of course when $\boldsymbol{\psi}$ is known, for example when $p(x|\boldsymbol{\psi})$ is a hyperprior, there is no need to sample from $p(\boldsymbol{\psi}|.)$ and the effect of using `I(.,.)` *is* equivalent to specifying a truncated distribution for x. However, in general, it is the user's responsibility to check that appropriate sampling will result whenever the `I(.,.)` construct is stretched beyond its intended purpose. If truncated distributions are required in cases where $\boldsymbol{\psi}$ is unknown, then they might be handled by working out an algebraic form for the likelihood and using the techniques for implementing non-standard distributions discussed in the following subsection.

Note that the `I(.,.)` notation is never appropriate when x is observed (the bounds will simply be ignored in this case). Note also that even when `I(.,.)` is used for censoring, if x, $\boldsymbol{\psi}$, a and b are all unknown, then a and b must not be functions of $\boldsymbol{\psi}$.

3.3 Non-standard distributions

Distributions other than those given in Table 1 and Table 2 can still be implemented by making use of either the 'ones trick' or the 'zeros trick' (see § **??** for details). Alternatively, new distributions can be 'hard-wired' into WinBUGS using the WinBUGS Development Interface (WBDev; lunn:isba:2003; http://www.winbugs-development.org.uk). This provides template modules of source code that can be straightforwardly modified and then fully integrated back into the WinBUGS framework. A package containing a range of new distributions written by WinBUGS users can be downloaded from the WBDev website.

4 Deterministic functions

4.1 Standard functions

Logical expressions can be built using the operators `+`, `-`, `*` and `/`, and the standard functions listed in Table 3. Bracketing can be used to any depth. Except for the 'index' parameter (`i`) in both `rank(.,.)` and `ranked(.,.)`, all scalar-valued parameters appearing on the right-hand-side of a `<-` relation in Table 3 can be expressions as well as named-nodes or

numerical values. Note that the functions `cloglog(.)`, `log(.)`, `logit(.)` and `probit(.)` can be used on the left-hand-side of a logical relation, as indicated in the 'Usage' column. Also note that logical nodes cannot be given data or initial values (except when using the data transformation facility described in §8).

4.2 Special functions

4.2.1 The 'cut' function

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 for whom we have observed partial data that we do not wish to use for parameter estimation; (b) when we want to use data to learn about some parameters but not others; or (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. In practical terms, the syntax `y <- cut(x)` produces a logical node `y` that is an exact copy of node `x` in the sense that it always has the same value as `x` (`x` may be stochastic or logical). However, any descendants of `y` that are introduced into the graph cannot then influence inference on `x`. The reader is referred to §**??** for examples of appropriate usage of `cut(.)`.

4.2.2 Deviance

BUGS automatically creates a logical node called `deviance` for the specified model. This evaluates, using the current state of the model, minus twice the logarithm of the conditional likelihood. By conditional likelihood we mean the joint probability distribution of all observed and censored nodes, conditional on their stochastic parents. The `deviance` node can be monitored and contributes to the calculation of DIC (see §**??**).

4.3 Other functions

There are several add-on interfaces to BUGS that contain libraries of specialized functions, *e.g.* PKBugs [**?**], Jump [**?**]. In addition, WBDev [**?**] can be used to 'hard-wire' one's own specialized functions into the WinBUGS framework. This can offer massive (orders of magnitude) gains in efficiency over building the relevant expression (where possible) using standard BUGS syntax. WBDev also provides much more scope for specifying highly complex functions than does the BUGS language. And furthermore, WBDev functions are *packaged* into 'black box' modelling components; this helps maintain the 'readability' of the BUGS model, which might lead to fewer errors. A range of 'shared' WBDev functions written by WinBUGS users can be downloaded from the WBDev website: http://www.winbugs-development.org.uk.

5 Repetition

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 any depth of nesting of for-loops is permitted (so long as a different index/loop-variable is used for each loop). Note also that `a` and `b` must both be integer-valued observed data or integer-valued logical expressions involving only standard operators (`+`, `-`, `*`, `/`), numerical values, observed data, and/or other for-loop indices (`j`, say). In particular, `a` and `b` must not be unobserved-stochastic or named-logical nodes (although see §**??** for a possible way of getting around this).

6 Multivariate quantities

We define a multivariate quantity as any collection of nodes, either all stochastic or all logical, that are defined simultaneously, *e.g.* `x[] ~ dmnorm(mu[], T[,])`, `y[,] <- inverse(z[,])`. All multivariate quantities must form contiguous elements of the array of nodes to which they belong. As we traverse contiguous elements of a given array, the fastest changing index is the final index, and so all multivariate quantities should be defined using the latter indices. For example, the following code specifies $\boldsymbol{y}_i \sim \text{MVN}_d(\boldsymbol{\mu}, \mathbf{T}^{-1})$ for a collection of vectors $\boldsymbol{y}_i$, $i = 1, ..., K$, which form the rows of a $K \times d$ matrix $\mathbf{Y}$:

```
for (i in 1:K) {
  Y[i, 1:d] ~ dmnorm(mu[1:d], T[1:d, 1:d])
}
```

Now suppose that each $\boldsymbol{y}_i$ has a distinct precision matrix $\mathbf{T}_i$, as opposed to a common precision $\mathbf{T}$, and that all of these matrices are to be stored in a 3-dimensional array `P[,,]`. We specify a Wishart$(\mathbf{R}, k)$ prior for each matrix via:

```
for (i in 1:K) {
  P[i, 1:d, 1:d] ~ dwish(R[1:d, 1:d], k)
}
```

7 Indexing

7.1 Functions as indices

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 (given the values of any loop-variables) is allowed for an index. On the right-hand-side the index can be a fixed-value expression 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 §7.3 below). However, functions of unobserved nodes are not permitted to appear directly as index terms (named logical nodes may be introduced if such functions are required).

7.2 Implicit indexing

The conventions broadly follow those of S-Plus/R:

- `n:m` represents $n, n+1, ..., m$;
- `x[]` represents all values of vector `x`;
- `Y[,3]` indicates all values in the third column of matrix `Y`.

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

7.3 Nested indexing

Nested indexing can be very effective. For example, suppose `N` individuals can each be in one of `J` groups, and `g[1:N]` is a vector containing the group memberships for each individual. Then group coefficients `beta[j]` (`j` = 1, ..., `J`) can be fitted using `beta[g[i]]` in a regression equation in which individuals are indexed by `i`.

In the BUGS language, nested indexing can be used for the parameters of distributions, *e.g.*

```
for (i in 1:N) {
  c[i] ~ dcat(theta[1:J])
  y[i] ~ dnorm(mu[c[i]], tau)
}
```

Here `y[i]`, `i` = 1, ..., `N`, are realisations from a normal mixture distribution with `J` components, each with a distinct mean `mu[j]`, `j` = 1, ..., `J`, and a common precision `tau`. The `c[1:N]` vector contains the indices of the components to which each observation belongs.

8 Data transformations

Although transformations of data can always be carried out before using BUGS , 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 that also features as a stochastic node (such as `z` above), a stochastic node is created with the calculated values as fixed data. Note that this construction is only possible when transforming observed data (not a function of data and parameters) with no missing values.

Table 1: Continuous distributions available in BUGS .

Distribution	**Usage**	**Definition**	**Constraints**
beta	`x ~ dbeta(a,b)`	$p(x\|a,b) = x^{a-1}(1-x)^{b-1}\frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)}$; $x \in [0,1]$	$a>0,\ b>0$
chi-squared	`x ~ dchisqr(k)`	$p(x\|k) = 2^{-\frac{k}{2}}x^{\frac{k}{2}-1}e^{-\frac{x}{2}}/\Gamma(k/2)$; $x>0$	$k>0$
Dirichlet	`x[] ~ ddirch(theta[])`	$p(\boldsymbol{x}\|\boldsymbol{\theta}) = \frac{\Gamma(\sum_i \theta_i)}{\prod_i \Gamma(\theta_i)}\prod_i x_i^{\theta_i-1}$; $x_i \in [0,1],\ \sum_i x_i = 1$	$\theta_i > 0$
double exponential	`x ~ ddexp(mu,tau)`	$p(x\|\mu,\tau) = \frac{\tau}{2}e^{-\tau\|x-\mu\|}$; $-\infty < x < \infty$	$\tau > 0$
exponential	`x ~ dexp(theta)`	$p(x\|\theta) = \theta e^{-\theta x}$; $x>0$	$\theta > 0$
gamma	`x ~ dgamma(a,b)`	$p(x\|a,b) = b^a x^{a-1}e^{-bx}/\Gamma(a)$; $x>0$	$a>0,\ b>0$
generalized gamma	`x ~ dgen.gamma(a,b,c)`	$p(x\|a,b,c) = c\,b^{ca}x^{ca-1}e^{-(bx)^c}/\Gamma(a)$; $x>0$	$a>0,\ b>0,\ c>0$
log-normal	`x ~ dlnorm(mu,tau)`	$p(x\|\mu,\tau) = \sqrt{\frac{\tau}{2\pi}}\frac{1}{x}e^{-\frac{\tau}{2}(\log x-\mu)^2}$; $x>0$	$\tau > 0$
logistic	`x ~ dlogis(mu,tau)`	$p(x\|\mu,\tau) = \tau e^{\tau(x-\mu)}/(1+e^{\tau(x-\mu)})^2$; $-\infty < x < \infty$	$\tau > 0$
multivariate normal	`x[] ~ dmnorm(mu[],T[,])`	$p(\boldsymbol{x}\|\boldsymbol{\mu},\mathbf{T}) = (2\pi)^{-\frac{d}{2}}\|\mathbf{T}\|^{\frac{1}{2}}e^{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})'\mathbf{T}(\boldsymbol{x}-\boldsymbol{\mu})}$; $-\infty < x_i < \infty$	$\mathbf{T}$ symmetric & positive definite
multivariate Student-t	`x[] ~ dmt(mu[],T[,],k)`	$p(\boldsymbol{x}\|\boldsymbol{\mu},\mathbf{T},k) = \Gamma\left((k+d)/2\right)(k\pi)^{-\frac{d}{2}}\|\mathbf{T}\|^{\frac{1}{2}}$ $\times\{1+(\boldsymbol{x}-\boldsymbol{\mu})'\mathbf{T}(\boldsymbol{x}-\boldsymbol{\mu})/k\}^{-(k+d)/2}/\Gamma(k/2)$; $-\infty < x_i < \infty$	$\mathbf{T}$ symmetric & positive definite, $k \geq 2$
normal	`x ~ dnorm(mu,tau)`	$p(x\|\mu,\tau) = \sqrt{\frac{\tau}{2\pi}}e^{-\frac{\tau}{2}(x-\mu)^2}$; $-\infty < x < \infty$	$\tau > 0$

Continued on next page...

Table 1 – Continued

Distribution	Usage	Definition	Constraints
Pareto	`x ~ dpar(a,b)`	$p(x\|a,b) = a\,b^a x^{-(a+1)};\quad x > b$	$a > 0,\ b > 0$
Student-t	`x ~ dt(mu,tau,k)`	$p(x\|\mu,\tau,k) = \Gamma\left((k+1)/2\right)(k\pi)^{-\frac{1}{2}}\tau^{\frac{1}{2}}$ $\times\{1+\tau(x-\mu)^2/k\}^{-(k+1)/2}/\Gamma(k/2);$ $-\infty < x < \infty$	$\tau > 0,\ k \geq 2$
uniform	`x ~ dunif(a,b)`	$p(x\|a,b) = 1/(b-a);\quad x \in [a,b]$	$b > a$
Weibull	`x ~ dweib(a,b)`	$p(x\|a,b) = a\,b\,x^{a-1}e^{-bx^a};\quad x > 0$	$a > 0,\ b > 0$
Wishart	`x[,] ~ dwish(R[,],k)`	$p(\mathbf{x}\|\mathbf{R},k) = \|\mathbf{R}\|^{\frac{k}{2}}\|\mathbf{x}\|^{\frac{k-d-1}{2}}e^{-\frac{1}{2}\mathrm{tr}(\mathbf{Rx})};$ $\mathbf{x}$ symmetric & positive definite	$\mathbf{R}$ symmetric & positive definite, $k \geq d$

Table 2: Discrete distributions available in BUGS .

Distribution	Usage	Definition	Constraints
Bernoulli	`x ~ dbern(theta)`	$p(x\|\theta) = \theta^x(1-\theta)^{1-x};\quad x = 0,1$	$\theta \in [0,1]$
binomial	`x ~ dbin(theta,n)`	$p(x\|\theta,n) = \frac{n!}{x!(n-x)!}\theta^x(1-\theta)^{n-x};$ $x = 0,...,n$	$\theta \in [0,1],\ n \in \mathbb{Z}^{+1}$
categorical	`x ~ dcat(theta[])`	$p(x\|\boldsymbol{\theta}) = \theta_x;\quad x = 1,2,...,\dim(\boldsymbol{\theta})$	$\theta_i \in [0,1],\ \sum_i \theta_i = 1$
multinomial	`x[] ~ dmulti(theta[],n)`	$p(\boldsymbol{x}\|\boldsymbol{\theta},n) = \frac{n!}{\prod_i x_i!}\prod_i \theta_i^{x_i};$ $\sum_i x_i = n$	$\theta_i \in [0,1],\ \sum_i \theta_i = 1,$ $n \in \mathbb{Z}^+$
negative	`x ~ dnegbin(theta,n)`	$p(x\|\theta,n) = \frac{(x+n-1)!}{x!(n-1)!}\theta^n(1-\theta)^x;$	$\theta \in [0,1],\ n \in \mathbb{Z}^+$

Continued on next page...

[1] $\mathbb{Z}^+$ denotes the set of all positive integers.

Table 2 – Continued

Distribution	**Usage**	**Definition**	**Constraints**
binomial		$x = 0, 1, 2, ...$	
Poisson	`x ~ dpois(theta)`	$p(x\|\theta) = \frac{\theta^x}{x!}e^{-\theta}; \quad x = 0, 1, ...$	$\theta > 0$

Table 3: Deterministic functions available in BUGS .

Expression	**Function**	**Usage**	**Definition**
`abs`	absolute value	`y <- abs(x)`	$y = \|x\|$
`cloglog`	complementary log-log	`y <- cloglog(x)` `cloglog(z) <- a + b*x`	$y = \ln(-\ln(1-x)); \quad x \in (0,1)$ $\ln(-\ln(1-z)) = a + bx$
`cos`	cosine	`y <- cos(x)`	$y = \cos(x)$
`equals`	logical equals	`y <- equals(a,b)`	$y = 1$ if $a = b$, $y = 0$ otherwise
`exp`	exponential	`y <- exp(x)`	$y = e^x$
`inprod`	inner product	`y <- inprod(a[],b[])`	$y = \sum_i a_i b_i$
`inverse`	matrix inverse	`y[,] <- inverse(x[,])`	$\mathbf{y} = \mathbf{x}^{-1}$; $\mathbf{x}$ & $\mathbf{y}$ both $n \times n$
`log`	natural logarithm	`y <- log(x)` `log(z) = a + b*x`	$y = \ln(x); \quad x > 0$ $\ln(z) = a + bx$
`logdet`	log determinant	`y <- logdet(x[,])`	$y = \ln \|\mathbf{x}\|$; $\mathbf{x}$ symmetric & positive definite
`logfact`	log factorial	`y <- logfact(x)`	$y = \ln(x!); \quad x = 0, 1, 2, ...$
`loggam`	log gamma function	`y <- loggam(x)`	$y = \ln(\Gamma(x)); \quad x > 0$
`logit`	logistic transform	`y <- logit(x)` `logit(z) <- a + b*x`	$y = \ln(\frac{x}{1-x}); \quad x \in (0,1)$ $\ln(\frac{z}{1-z}) = a + bx$
`max`	maximum	`y <- max(a,b)`	$y = \max(a, b)$
`mean`	mean	`y <- mean(x[])`	$y = \frac{1}{n} \sum_i x_i$
`min`	minimum	`y <- min(a,b)`	$y = \min(a, b)$
`phi`	standard normal	`y <- phi(x)`	$y = \Phi(x) = \int_{-\infty}^{x} \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}t^2} \, dt$

Continued on next page...

Table 3 – Continued

Expression	**Function**	**Usage**	**Definition**
	distribution function		
`pow`	power	`y <- pow(a,b)`	$y = a^b$
`probit`	probit	`probit(z) <- a + b*x`	$\Phi^{-1}(z) = a + bx$
`sqrt`	square root	`y <- sqrt(x)`	$y = \sqrt{x};\ \ x > 0$
`rank`	rank of ith element in vector	`y <- rank(x[],i)`	$y = \sum_j I(x_j \leq x_i)$
`ranked`	ith smallest value in vector	`y <- ranked(x[],i)`	$y = i$th smallest value in $\boldsymbol{x}$
`round`	nearest integer	`y <- round(x)`	$y = \lfloor x + 0.5 \rfloor$
`sd`	standard deviation	`y <- sd(x[])`	$y = \sqrt{\frac{1}{n-1}\sum_i \left(x_i - \frac{1}{n}\sum_i x_i\right)^2}$
`step`	unit step	`y <- step(x)`	$y = 1$ if $x \geq 0,\ y = 0$ otherwise
`sum`	sum	`y <- sum(x[])`	$y = \sum_i x_i$
`trunc`	truncate towards 0	`y <- trunc(x)`	$y = \lfloor x + (1 - \varepsilon)I(x < 0) \rfloor$