JAGS Version 2.2.0 user manual

Martyn Plummer

November 7, 2010

Chapter 6

Functions

Functions allow deterministic nodes to be defined using the `<-` ("gets") operator. Most of the functions in JAGS are scalar functions taking scalar arguments. However, JAGS also allows arbitrary vector- and array-valued functions, such as the matrix multiplication operator `%*%` and the transpose function `t()` defined in the `bugs` module, and the matrix exponential function `mexp()` defined in the `msm` module. JAGS also uses an enriched dialect of the BUGS language with a number of operators that are used in the S language.

Scalar functions taking scalar arguments are automatically vectorized. They can also be called when the arguments are arrays with conforming dimensions, or scalars. So, for example, the scalar c can be added to the matrix A using

```
B <- A + c
```

instead of the more verbose form

```
D <- dim(A)
for (i in 1:D[1])
   for (j in 1:D[2]) {
      B[i,j] <- A[i,j] + c
   }
}
```

6.1 Base functions

The functions defined by the `base` module all appear as infix or prefix operators. The syntax of these operators is built into the JAGS parser. They are therefore considered part of the modelling language. Table 6.1 lists them in reverse order of precedence.

Logical operators convert numerical arguments to logical values: zero arguments are converted to FALSE and non-zero arguments to TRUE. Logical and comparison operators return the value 1 if the result is TRUE and 0 if the result is FALSE. Comparison operators are non-associative: the expression `x < y < z`, for example, is syntactically incorrect.

The `%special%` function is an exception in table 6.1. It is not a function defined by the `base` module, but is a place-holder for any function with a name starting and ending with the character "`%`" Such functions are automatically recognized as infix operators by the JAGS model parser, with precedence defined by table 6.1.

Type	Usage	Description
Logical operators	`x || y`	Or
	`x && y`	And
	`!x`	Not
Comparison operators	`x > y`	Greater than
	`x >= y`	Greater than or equal to
	`x < y`	Less than
	`x <= y`	Less than or equal to
	`x == y`	Equal
Arithmetic operators	`x + y`	Addition
	`x - y`	Subtraction
	`x * y`	Multiplication
	`x / y`	Division
	`x %special% y`	User-defined operators
	`-x`	Unary minus
Power function	`x^y`	

Table 6.1: Base functions listed in reverse order of precedence

6.2 Functions in the bugs module

6.2.1 Scalar functions

Table 6.2 lists the scalar-valued functions in the `bugs` module that also have scalar arguments. These functions are automatically vectorized when they are given vector, matrix, or array arguments with conforming dimensions.

Table 6.4 lists the link functions in the `bugs` module. These are smooth scalar-valued functions that may be specified using an S-style replacement function notation. So, for example, the log link

```
log(y) <- x
```

is equivalent to the more direct use of its inverse, the exponential function:

```
y <- exp(x)
```

This usage comes from the use of link functions in generalized linear models.

Table 6.3 shows functions to calculate the probability density, probability function, and quantiles of some of the distributions provided by the `bugs` module. These functions are parameterized in the same way as the corresponding distribution. For example, if x has a normal distribution with mean μ and precision τ

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

Then the usage of the corresponding density, probability, and quantile functions is:

```
density.x  <- dnorm(x, mu, tau)     # Density of normal distribution at x
prob.x     <- pnorm(x, mu, tau)     # P(X <= x)
quantile90.x <- qnorm(0.9, mu, tau) # 90th percentile
```

For details of the parameterization of the other distributions, see tables 7.1 and 7.2.

Usage	Description	Value	Restrictions on arguments
`abs(x)`	Absolute value	Real	
`acos(x)`	Arc-cosine	Real	$-1 < x < 1$
`acosh(x)`	Hyperbolic arc-cosine	Real	$1 < x$
`asin(x)`	Arc-sine	Real	$-1 < x < 1$
`asinh(x)`	Hyperbolic arc-sine	Real	
`atan(x)`	Arc-tangent	Real	
`atanh(x)`	Hyperbolic arc-tangent	Real	$-1 < x < 1$
`cos(x)`	Cosine	Real	
`cosh(x)`	Hyperbolic Cosine	Real	
`cloglog(x)`	Complementary log log	Real	$0 < x < 1$
`equals(x,y)`	Test for equality	Logical	
`exp(x)`	Exponential	Real	
`icloglog(x)`	Inverse complementary log log function	Real	
`ilogit(x)`	Inverse logit	Real	
`log(x)`	Log function	Real	$x > 0$
`logfact(x)`	Log factorial	Real	$x > -1$
`loggam(x)`	Log gamma	Real	$x > 0$
`logit(x)`	Logit	Real	$0 < x < 1$
`phi(x)`	Standard normal cdf	Real	
`pow(x,z)`	Power function	Real	If $x < 0$ then z is integer
`probit(x)`	Probit	Real	$0 < x < 1$
`round(x)`	Round to integer away from zero	Integer	
`sin(x)`	Sine	Real	
`sinh(x)`	Hyperbolic Sine	Real	
`sqrt(x)`	Square-root	Real	$x >= 0$
`step(x)`	Test for $x \geq 0$	Logical	
`tan(x)`	Tangent	Real	
`tanh(x)`	Hyperbolic Tangent	Real	
`trunc(x)`	Round to integer towards zero	Integer	

Table 6.2: Scalar functions in the `bugs` module

Distribution	Density	Distribution	Quantile
Bernoulli	dbern	pbern	qbern
Beta	dbeta	pbeta	qbeta
Binomial	dbin	pbin	qbin
Chi-square	dchisqr	pchisqr	qchisqr
Double exponential	ddexp	pdexp	qdexp
Exponential	dexp	pexp	qexp
F	df	pf	qf
Gamma	dgamma	pgamma	qgamma
Generalized gamma	dgengamma	pgengamma	qgengamma
Hypergeometric	dhyper	phyper	qhyper
Log-normal	dlnorm	plnorm	qlnorm
Negative binomial	dnegbin	pnegbin	qnegbin
Normal	dnorm	pnorm	qnorm
Pareto	dpar	ppar	qpar
Poisson	dpois	ppois	qpois
Student t	dt	pt	qt
Weibull	dweib	pweib	qweib

Table 6.3: Wrappers for the d-p-q functions from the `Rmath` library

Link function	Description	Range	Inverse
`cloglog(y) <- x`	Complementary log log	$0 < y < 1$	`y <- icloglog(x)`
`log(y) <- x`	Log	$0 < y$	`y <- exp(x)`
`logit(y) <- x`	Logit	$0 < y < 1$	`y <- ilogit(x)`
`probit(y) <- x`	Probit	$0 < y < 1$	`y <- phi(x)`

Table 6.4: Link functions in the `bugs` module

Function	Description	Restrictions
`inprod(x1,x2)`	Inner product	Dimensions of a, b conform
`interp.lin(e,v1,v2)`	Linear Interpolation	e scalar, $v1, v2$ conforming vectors
`logdet(a)`	Log determinant	a is a square matrix
`max(x1,x2,...)`	Maximum element among all arguments	
`mean(x)`	Mean of elements of a	
`min(x1,x2,...)`	Minimum element among all arguments	
`prod(x)`	Product of elements of a	
`sum(a)`	Sum of elements of a	
`sd(a)`	Standard deviation of elements of a	

Table 6.5: Scalar-valued functions with general arguments in the `bugs` module

Usage	Description	Restrictions
`inverse(a)`	Matrix inverse	a is a symmetric positive definite matrix
`mexp(a)`	Matrix exponential	a is a square matrix
`rank(v)`	Ranks of elements of v	v is a vector
`sort(v)`	Elements of v in order	v is a vector
`t(a)`	Transpose	a is a matrix
`a %*% b`	Matrix multiplication	a, b conforming vector or matrices

Table 6.6: Vector- or matrix-valued functions in the `bugs` module

6.3 Scalar-valued functions with vector arguments

Table 6.5 lists the scalar-valued functions in the `bugs` module that take general arguments. Unless otherwise stated in table 6.5, the arguments to these functions may be scalar, vector, or higher-dimensional arrays.

The `max()` and `min()` functions work like the corresponding R functions. They take a variable number of arguments and return the maximum/minimum element over all supplied arguments. This usage is compatible with WinBUGS, although more general.

6.4 Vector- and array-valued functions

Table 6.6 lists vector- or matrix-valued functions in the `bugs` module.

The `sort` and `rank` functions behaves like their R namesakes: `sort` accepts a vector and returns the same values sorted in ascending order; `rank` returns a vector of ranks. This is distinct from WinBUGS, which has two scalar-valued functions `rank` and `ranked`.

Chapter 7

Distributions

Distributions are used to define stochastic nodes using the ~ operator. The distributions defined in the bugs module are listed in table 7.1 (real-valued distributions), 7.2 (discrete-valued distributions), and 7.3 (multivariate distributions).

Some distributions have restrictions on the valid parameter values, and these are indicated in the tables. If a Distribution is given invalid parameter values when evaluating the log-likelihood, it returns $-\infty$. When a model is initialized, all stochastic nodes are checked to ensure that the initial parameter values are valid for their distribution.

Name	Usage	Density	Lower	Upper
Beta	`dbeta(a,b)`	$\frac{x^{a-1}(1-x)^{b-1}}{\beta(a,b)}$	0	1
	$a>0$, $b>0$			
Chi-square	`dchisqr(k)`	$\frac{x^{\frac{k}{2}-1}\exp(-x/2)}{2^{\frac{k}{2}}\Gamma(\frac{k}{2})}$	0	
	$k>0$			
Double exponential	`ddexp(mu,tau)`	$\tau\exp(-\tau\lvert x-\mu\rvert)/2$		
	$\tau>0$			
Exponential	`dexp(lambda)`	$\lambda\exp(-\lambda x)$	0	
	$\lambda>0$			
F	`df(n,m)`	$\frac{\Gamma(\frac{n+m}{2})}{\Gamma(\frac{n}{2})\Gamma(\frac{m}{2})}\left(\frac{n}{m}\right)^{\frac{n}{2}}x^{\frac{n}{2}-1}\left\{1+\frac{nx}{m}\right\}^{-\frac{(n+m)}{2}}$	0	
	$n>0$, $m>0$			
Gamma	`dgamma(r, mu)`	$\frac{\mu^r x^{r-1}\exp(-\mu x)}{\Gamma(r)}$	0	
	$\mu>0$, $r>0$			
Generalized gamma	`dgen.gamma(r,mu,beta)`	$\beta\mu^{\beta r}x^{\beta r-1}\exp\{-(\mu x)^{\beta}\}$	0	
	$\mu>0$, $\beta>0$, $r>0$			
Log-normal	`dlnorm(mu,tau)`	$\tau^{\frac{1}{2}}x^{-1}\exp\left\{-\tau(\log(x)-\mu)^2/2\right\}$	0	
	$\tau>0$			
Normal	`dnorm(mu,tau)`	$\left(\frac{\tau}{2\pi}\right)^{\frac{1}{2}}\exp\{-(x-\mu)^2\tau\}$		
	$\tau>0$			
Pareto	`dpar(alpha, c)`	$\alpha c^{\alpha}x^{-(\alpha+1)}$	c	
	$\alpha>0$, $c>0$			
Student t	`dt(mu,tau,k)`	$\frac{\Gamma(\frac{k+1}{2})}{\Gamma(\frac{k}{2})}\left(\frac{\tau}{k\pi}\right)^{\frac{1}{2}}\left\{1+\frac{\tau(x-\mu)^2}{k}\right\}^{-\frac{(k+1)}{2}}$		
	$\tau>0$, $k>0$			
Uniform	`dunif(a,b)`	$\frac{1}{b-a}$	a	b
	$a<b$			
Weibull	`dweib(v, lambda)`	$v\lambda x^{v-1}\exp(-\lambda x^{v})$	0	
	$v>0$, $\lambda>0$			

Table 7.1: Univariate real-valued distributions in the `bugs` module

Name	Usage	Density	Lower	Upper
Bernoulli	`dbern(p)`	$p^x(1-p)^{1-x}$	0	1
	$0<p<1$			
Binomial	`dbin(p,n)`	$\binom{n}{x}p^x(1-p)^{n-x}$	0	n
	$0<p<1$, $n\in\mathbb{N}^*$			
Categorical	`dcat(p)`	$\frac{p_x}{\sum_i p_i}$	1	N
	$p\in(\mathbb{R}^+)^N$			
Hypergeometric	`dhyper(n1,n2,m1,psi)`	$\binom{n_1}{x}\binom{n_2}{m_1-x}\psi^x$	$\max(0,n_+-m_1)$	$\min(n_1,m_1)$
	$0\le n_i$, $0<m_1\le n_+$			
Negative binomial	`dnegbin(p, r)`	$\binom{x+r-1}{x}p^r(1-p)^x$	0	
	$0<p<1$, $r\in\mathbb{N}^+$			
Poisson	`dpois(lambda)`	$\frac{\exp(-\lambda)\lambda^x}{x!}$	0	
	$\lambda>0$			

Table 7.2: Discrete univariate distributions in the `bugs` module

Name	Usage	Density
Dirichlet	`p ~ ddirch(alpha)` $\alpha_j \geq 0$	$\Gamma(\sum_i \alpha_i) \prod_j \frac{p_j^{\alpha_j - 1}}{\Gamma(\alpha_j)}$
Multivariate normal	`x ~ dmnorm(mu,Omega)` Ω positive definite	$\left(\frac{\lvert\Omega\rvert}{2\pi}\right)^{\frac{1}{2}} exp\{-(x-\mu)^T\Omega(x-\mu)/2\}$
Wishart	`Omega ~ dwish(R,k)` R pos. def.	$\frac{\lvert\Omega\rvert^{(k-p-1)/2}\lvert R\rvert^{k/2}\exp\{-\mathrm{Tr}(R\Omega/2)\}}{2^{pk/2}\Gamma_p(k/2)}$
Multivariate Student t	`x ~ dmt(mu,Omega,k)` Ω pos. def.	$\frac{\Gamma\{(k+p)/2\}}{\Gamma(k/2)(n\pi)^{p/2}}\lvert\Omega\rvert^{1/2}\left\{1+\frac{1}{k}(x-\mu)^T\Omega(x-\mu)\right\}^{-\frac{(k+p)}{2}}$
Multinomial	`x ~ dmulti(p, n)` $\sum_i x_i = n$	$n!\prod_j \frac{p_j^{x_j}}{x_j!}$

Table 7.3: Multivariate distributions in the `bugs` module

Chapter 8

Differences between JAGS and WinBUGS

Although JAGS aims for the same functionality as WinBUGS, there are a number of important differences.

8.0.1 Data format

There is no need to transpose matrices and arrays when transferring data between R and JAGS, since JAGS stores the values of an array in "column major" order, like R and FORTRAN (*i.e.* filling the left-hand index first).

If you have an S-style data file for WinBUGS and you wish to convert it for JAGS, then use the command `bugs2jags`, which is supplied with the `coda` package.

8.0.2 Distributions

Structural zeros are allowed in the Dirichlet distribution. If

```
p ~ ddirch(alpha)
```

and some of the elements of alpha are zero, then the corresponding elements of p will be fixed to zero.

The Multinomial (`dmulti`) and Categorical (`dcat`) distributions, which take a vector of probabilities as a parameter, may use unnormalized probabilities. The probability vector is normalized internally so that

$$p_i \to \frac{p_i}{\sum_j p_j}$$

8.0.3 Observable Functions

Logical nodes in the BUGS language are a convenient way of describing the relationships between observables (constant and stochastic nodes), but are not themselves observable. You cannot supply data values for a logical node.

This restriction can occasionally be inconvenient, as there are important cases where the data are a deterministic function of unobserved variables. Two important examples are

1. Censored data, which commonly occurs in survival analysis. In the most general case, we know that unobserved failure time T lies in the interval $(L, U]$.

2. Aggregate data when we observe the sum of two or more unobserved variables.

JAGS contains two novel distributions to handle these situations.

1. The `dinterval` distribution represents interval-censored data. It has two parameters: t the original continuous variable, and $c[]$, a vector of cut points of length M, say. If `X` $\sim$ `dinterval(t, c)` then
   $X = 0$ if $t \leq c[1]$
   $X = m$ if $c[m] < t \leq c[m+1]$ for $1 \leq m < M$
   $X = M$ if $c[M] < t$.

2. The `dsum` distribution represents the sum of two or more variables. It takes a variable number of parameters. If `Y` $\sim$ `dsum(x1,x2,x3)` then $Y = x1 + x2 + x3$.

These distributions exist to give a likelihood to data that is, in fact, a deterministic function of the parameters. The relation

```
Y ~ dsum(x1, x2)
```

is logically equivalent to

```
Y <- x1 + x2
```

But the latter form does not create a contribution to the likelihood, and does not allow you to define Y as data. The likelihood function is trivial: it is 1 if the parameters are consistent with the data and 0 otherwise. The `dsum` distribution also requires a special sampler, which can currently only handle the case where the parameters of `dsum` are unobserved stochastic nodes, and where the parameters are either all discrete-valued or all continuous-valued. A node cannot be subject to more than one `dsum` constraint.

8.0.4 Data transformations

JAGS allows data transformations, but the syntax is different from BUGS. BUGS allows you to put a stochastic node twice on the left hand side of a relation, as in this example taken from the manual

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

This is forbidden in JAGS. You must put data transformations in a separate block of relations preceded by the keyword `data`:

```
data {
   for (i in 1:N) {
      z[i] <- sqrt(y[i])
   }
```

```
}
model {
   for (i in 1:N) {
      z[i] ~ dnorm(mu, tau)
   }
   ...
}
```

This syntax preserves the declarative nature of the BUGS language. In effect, the data block defines a distinct model, which describes how the data is generated. Each node in this model is forward-sampled once, and then the node values are read back into the data table. The data block is not limited to logical relations, but may also include stochastic relations. You may therefore use it in simulations, generating data from a stochastic model that is different from the one used to analyse the data in the `model` statement.

This example shows a simple location-scale problem in which the "true" values of the parameters `mu` and `tau` are generated from a given prior in the `data` block, and the generated data is analyzed in the `model` block.

```
data {
   for (i in 1:N) {
      y[i] ~ dnorm(mu.true, tau.true)
   }
   mu.true ~ dnorm(0,1);
   tau.true ~ dgamma(1,3);
}
model {
   for (i in 1:N) {
      y[i] ~ dnorm(mu, tau)
   }
   mu ~ dnorm(0, 1.0E-3)
   tau ~ dgamma(1.0E-3, 1.0E-3)
}
```

Beware, however, that every node in the `data` statement will be considered as data in the subsequent `model` statement. This example, although superficially similar, has a quite different interpretation.

```
data {
   for (i in 1:N) {
      y[i] ~ dnorm(mu, tau)
   }
   mu ~ dnorm(0,1);
   tau ~ dgamma(1,3);
}
model {
   for (i in 1:N) {
      y[i] ~ dnorm(mu, tau)
   }
```