WinBUGS Jump Interface: User Manual

Dave Lunn
MRC Biostatistics Unit
david.lunn@mrc-bsu.cam.ac.uk

October 15th, 2008

Please use Lunn et al. (2008) and Lunn et al. (2006) when citing this interface. Please also refer to these for further clarification of our approach and for additional examples.

Introduction

The *WinBUGS Jump Interface* has been designed to enable *Reversible Jump* MCMC computation on graphical models with 'trans-dimensional' sub-graphs. At present the interface will handle various different types of "curve fitting" and "variable selection" problems, as described in the following sections. If you have any suggestions for future developments then please feel free to email me at the above address.

Models & Examples

As indicated above, there are two main classes of model that the "Jump" interface can currently handle – "curve fitting" and "variable selection". This manual is supported by three examples that illustrate the use of these modelling techniques. There are two examples on "variable selection" and one on "curve fitting" – these have self-explanatory titles and can be found in the `Jump/Examples/` directory. All three models are based on the same abstract graphical model, which is depicted in Figure 1. Note that it is an important feature of the interface/software that such sub-models can be incorporated into arbitrary other graphs. For example, there is no reason why the full model cannot include several trans-dimensional components.

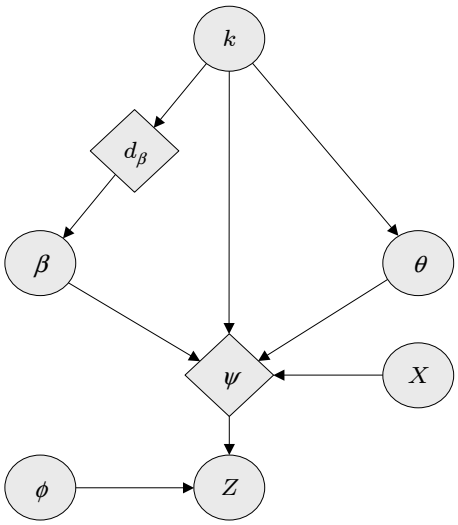

Figure 1: Graphical sub-model for reversible jump WinBUGS components.

The notation in Figure 1 is described as follows. Z represents the response variable of interest – typically this will comprise n observed values, $Z_1, Z_2, ..., Z_n$, say, although there is no requirement for them to be

observed or even *observable* (i.e. Z may represent a set of unknown parameters). ψ typically represents the location of the assumed distribution for Z: ψ is a deterministic function of fixed dimension (the same as that of Z) but it is defined in terms of an unknown number of 'entities of interest', for example, an unknown number of covariates in "variable selection" settings. This unknown number of 'entities of interest' is referred to as the dimension of the reversible jump problem and is denoted by k. Clearly k is a parent of ψ (and hence Z, since ψ is logical). In addition, ψ has three other parents: (i) θ represents the current 'configuration' of the model, i.e. in variable selection the currently selected variables/covariates – the θ vector always has dimension k; (ii) β is a d_β-dimensional vector of coefficients – in the case of variable selection $d_\beta = k + 1$, i.e. there is one coefficient for each included covariate and one intercept term; and (iii) X denotes any additional input parameters/data required to fully define ψ, such as the matrix of all covariates in variable selection. The remaining distributional parameters associated with Z are denoted by ϕ. Thus ϕ will typically comprise the (common) precision of the Z_is.

Mathematically, the form of $\psi = (\psi_1, \psi_2, ..., \psi_n)'$ for variable selection and curve fitting, respectively, is given by Equations 1 and 2.

$$\psi = \begin{pmatrix} \psi_1 \\ \psi_2 \\ \vdots \\ \psi_n \end{pmatrix} = \begin{pmatrix} 1 & x_{1\theta_1} & \cdots & x_{1\theta_k} \\ 1 & x_{2\theta_1} & \cdots & x_{2\theta_k} \\ \vdots & \vdots & & \vdots \\ 1 & x_{n\theta_1} & \cdots & x_{n\theta_k} \end{pmatrix} \begin{pmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_{k+1} \end{pmatrix} = W\beta \tag{1}$$

where the $x_{..}$s denote elements of the full covariate matrix X, and the k elements of θ each index a specific column of X.

$$\psi_i = \begin{cases} 0 & x_i < l \\ \beta_1 & x_i = l \\ \beta_1 + \sum_{j=1}^{ord} \beta_{j+1}(x_i - l)^j + \sum_{j=1}^{k} \sum_{m=cont}^{ord} \beta_{[j\times(ord-cont+1)+m+1]}(x_i - \theta_j)_+^m & x_i > l \end{cases} \tag{2}$$

where $x_i, i = 1, ..., n$, represent instances of the independent variable, i.e. the variable on which $Z_i, i = 1, ..., n$, is to be regressed, and $(x_i - \theta_j)_+ = (x_i - \theta_j) \times I(x_i - \theta_j > 0)$. The $x_i > l$ term in Eqn. 2 above is referred to as a 'piecewise polynomial' with 'knots' (or 'knot-points') $\theta_j, j = 1, ..., k$ – a 'knot' is a point on the x-axis where the nature of the function changes. *ord* and *cont* denote the 'order' and 'continuity' of the piecewise polynomial, respectively, and l is the 'left-hand boundary', beyond which (i.e. $x < l$) the function equals zero. In general, we must have $ord \geq 0$ and $0 \leq cont \leq ord$ but often we simply set $cont = ord$, which minimizes the number of terms such that the distance between the evaluation point (x_i) and each knot (θ_j) is always raised to the same power, i.e. ord. In what follows we briefly discuss new elements incorporated into the BUGS language that represent models either given by or based on Equations 1 and 2.

New BUGS language elements

All of the following new BUGS language elements are vector-valued functions (hence the syntax `v <- jump...`). This means that they define the entire ψ vector, i.e. in BUGS syntax we write something like `psi[1:n] <- jump...`

- `v <- jump.lin.pred(v,s,s)`
  This specifies the linear predictor exactly as given by Equation 1. Both of the "variable selection" examples in the `Jump/Examples/` directory make use of this new component. The arguments are described as follows. The first argument is the full covariate matrix, with covariates arranged in columns and rows corresponding to the various instances of the response variable, $Z_i, i = 1, ..., n$. The second argument (a scalar) is k, the node used to represent the dimension of the reversible jump problem, i.e. the number of currently selected covariates. The third argument is the prior precision for the $k+1$ regression coefficients contained in β – the user can specify a fixed value or, alternatively, arbitrary sub-models for this precision (the prior mean for each coefficient is assumed to be zero). It is important to note that the model specifications provided in the examples do not

contain any references to the β or θ variables. These variable-dimension vectors are not visible to the user (see Lunn et al., 2008, for details) – their existence is implied by the specification of any of the various new BUGS language elements that comprise the 'Jump' interface, and they *do* form part of the graphical model, but you will not be able to access them directly – see later for details of how to make inferences about β and θ. One obvious implication of β and θ being hidden is the lack of control that the user has over the prior specification for these parameters. In the case of β, this is less of an issue since the default prior, $\beta \sim \text{MVN}_{k+1}(\mathbf{0}, \tau_\beta \mathbf{I}_{k+1})$ (where $\tau_\beta =$ `beta.prec` in the examples), is such a natural choice (except see later for a version in which a separate prior for the intercept is allowed). In the case of θ the default prior specification is that for a given value of k (i.e. for a given number of selected covariates) all possible values of theta are equally likely. Therefore,

$$p(\theta|k) = \left(\begin{array}{c} Q \\ k \end{array} \right)^{-1} = \frac{k!(Q-k)!}{Q!}$$

where Q is the total number of variables (i.e. the number of columns in X). This is because there are $Q!/k!(Q-k)!$ ways of selecting k covariates from Q covariates. If this is not the desired prior then do bear in mind that you have complete control over the prior for k. For example, one can specify that *all* models are equally likely via

$$p(k) \sim \text{Binomial}(Q, 0.5) \Rightarrow p(\theta, k) = p(\theta|k)p(k) = \left(\begin{array}{c} Q \\ k \end{array} \right)^{-1} \times \left(\begin{array}{c} Q \\ k \end{array} \right) \left(\frac{1}{2}\right)^Q = \frac{1}{2^Q}$$

This is the approach used in the two variable selection examples.

- `v <- jump.lin.pred.trans(v,s,s)`
  This specifies exactly the same linear predictor as the previous BUGS-language element (`jump.lin.pred(.)`) except that the covariate matrix X is now transposed, so that covariates are arranged in rows and columns correspond to instances of the response variable.

- `v <- jump.lin.pred.pred(v,v)`
  With the variable-dimension vectors β and θ hidden, one might wonder how to perform prediction based on ψ, i.e. how to calculate ψ for new values of the covariates. This new BUGS-language element allows such prediction by accessing the β and θ vectors via the linear predictor specified in the non-predictive part of the model (i.e. that part used for inferring the posterior distributions of β and θ). The two arguments are described as follows: first, the vector of linear predictor nodes (either `jump.lin.pred(.)` or `jump.lin.pred.trans(.)`) used in the non-predictive part of the model are given – these allow the predictive nodes access to β and θ; the second argument is the full covariate matrix for which predicted values are required. Please see the example "`variable_selection_2.odc`", where predictive nodes are incorporated into the graphical model in order to make inferences about the coefficients in β, for further clarification.

- `v <- jump.lin.pred.cat(v,v,s,s)`
  This new element can be useful for variable selection problems in which some or all of the covariates are categorical. In such cases, it may be desirable to construct several binary covariates, say, from each categorical covariate and set the model up so that if the categorical covariate is to be selected then all of its binary components should be included in the evaluation of ψ, with a separate coefficient (from β) used to represent each one. In short, `jump.lin.pred.cat(.)` is a generalization of `jump.lin.pred(.)` in which each 'variable' may correspond to an arbitrary number of (consecutive) columns in X. The four arguments are described as follows: (i) the full (expanded) covariate matrix with covariates arranged in columns; (ii) an "offsets" vector that gives the 'starting column' (in X) of each 'variable' – for example, if we have six three-level categorical variables in total, each of which we wish to split into two indicator (or dummy) variables, then our X matrix would contain 12 columns and our 'offsets' vector would be $(1, 3, 5, 7, 9, 11)$; (iii) k, the dimension, i.e. the number of currently selected variables ($k \in \{0, 1, 2, 3, 4, 5, 6\}$ for the example just mentioned); and (iv) the prior precision for the regression coefficients. Note that `jump.lin.pred.cat(.)` is exactly equivalent to `jump.lin.pred(.)` in cases where only one column of X is specified for each 'variable' (although `jump.lin.pred.cat(.)` takes slightly longer to evaluate due to its additional generality).

- `v <- jump.lin.pred.cat.trans(v,v,s,s)`
  This is exactly the same as `jump.lin.pred.cat(.)` except that the (expanded) covariate matrix is transposed, i.e. with covariates arranged in rows.

- `v <- jump.lin.pred.cat.pred(v,v)`
  This new BUGS-language element allows prediction based on ψ for linear predictors specified via `jump.lin.pred.cat(.)` or `jump.lin.pred.cat.trans(.)`. It works in exactly the same way as the `jump.lin.pred.pred(.)` element described above.

- `v <- jump.lin.pred.alt(v,s,s)`
  This component is exactly the same as `jump.lin.pred(.)` described above except that the internal β vector does not include an intercept term – hence it has dimension k. This is so that the intercept can be modelled separately in the fixed-dimension part of the model, for example it may then be assigned a different prior to the coefficients in β. There is a potential problem with this approach, however, in that all model-moves proposed by the reversible jump sampler will be conditional on the current value of the intercept (since the coefficients and intercept are no longer updated together) – hence it may be harder to accept new model states. The `jump.lin.pred.int(.)` component described below circumvents this issue.

- `v <- jump.lin.pred.alt.trans(v,s,s)`
  This is exactly the same as `jump.lin.pred.alt(.)` except that the covariate matrix is transposed, i.e. with covariates arranged in rows.

- `v <- jump.lin.pred.alt.pred(v,v)`
  This component allows prediction based on ψ for linear predictors specified via `jump.lin.pred.alt(.)` or `jump.lin.pred.alt.trans(.)`. It works in exactly the same way as the `jump.lin.pred.pred(.)` element described above.

- `v <- jump.lin.pred.int(v,s,s,s,s)`
  is exactly the same as `jump.lin.pred(.)` except that it allows a separate prior mean and precision to be specified for the intercept term. The first three arguments are the same as for `jump.lin.pred(.)` whereas the fourth and fifth arguments represent the prior mean and precision, respectively, for the intercept term. The intercept and gradient parameters can be updated together as they both appear in the internal β vector (cf. `jump.lin.pred.alt(.)`).

- `v <- jump.lin.pred.int.trans(v,s,s,s,s)`
  This is exactly the same as `jump.lin.pred.int(.)` except that the covariate matrix is transposed, i.e. with covariates arranged in rows.

- `v <- jump.lin.pred.int.pred(v,v)`
  This component allows prediction based on ψ for linear predictors specified via `jump.lin.pred.int(.)` or `jump.lin.pred.int.trans(.)`. It works in exactly the same way as the `jump.lin.pred.pred(.)` element described above.

- `v <- jump.pw.poly.c.lin(v,s,s,s,s)`
  This specifies the piecewise polynomial given by Eqn. 2 with $cont = ord = 1$. The first three arguments are described as follows. First, we have X, the vector of evaluation points, i.e. the set of x-values at which the piecewise polynomial is to be evaluated. The second argument is k, the unknown number of 'knot-points' constituting θ. And the third argument is the prior precision for the regression coefficients in β. The "`c`" in the `jump.pw.poly.c.lin(.)` name stands for 'continuous knots': for all piecewise polynomials with names beginning "`jump.pw.poly.c`", each knot may take any real value in the interval (l, r) (where l is the 'left-hand boundary' defined earlier) – the prior distribution over this interval is assumed to be uniform. The fourth and fifth arguments to `jump.pw.poly.c.lin(.)` are l and r respectively – these must be constants rather than stochastic parameters or logical functions.

- `v <- jump.pw.poly.c.quad(v,s,s,s,s)`
  Exactly the same as `jump.pw.poly.c.lin(.)` but with $cont = ord = 2$

- `v <- jump.pw.poly.c.cub(v,s,s,s,s)`
  Exactly the same as `jump.pw.poly.c.lin(.)` but with $cont = ord = 3$

- `v <- jump.pw.poly.c.gen(v,s,s,s,s,s,s)`
  This is a generalized form of `jump.pw.poly.c....(.)` with two additional arguments for specifying the values of *cont* and *ord*. Arguments 1–5 are the same as for other piecewise polynomials (with continuous knots). The sixth and seventh arguments are *ord* and *cont* respectively. Note that both of these must be constants rather than logical or stochastic nodes and we must also have that $ord \geq 0$ and $0 \leq cont \leq ord$.

- `v <- jump.pw.poly.d.lin(v,s,s)`
  This is the same as `jump.pw.poly.c.lin(.)` but with discrete knots instead of continuous knots (hence the "`d`" in the name instead of "`c`"). Here any of the x_is (elements of X) except x_1 and x_n can be chosen as a knot-point and the 'left-hand boundary' l is given by x_1. The three arguments are the same as the first three arguments for the 'continuous knot' polynomials defined above. There is no need to specify a bounding interval for the knots in this case and so the fourth and fifth arguments from above are no longer required. The prior distribution for θ given k, $p(\theta|k)$, is such that all possible combinations of k knots (from a total of $n-2$ possible knots) are equally likely, i.e. $p(\theta|k) = k!(n-2-k)!/(n-2)!$

- `v <- jump.pw.poly.d.quad(v,s,s)`
  Discrete knots version of `jump.pw.poly.c.quad(.)`

- `v <- jump.pw.poly.d.cub(v,s,s)`
  Discrete knots version of `jump.pw.poly.c.cub(.)`

- `v <- jump.pw.poly.d.gen(v,s,s,s,s)`
  Discrete knots version of `jump.pw.poly.c.gen(.)`. We list the arguments as follows, for maximum clarity: (i) the X vector; (ii) the number of knots k; (iii) the prior precision for the regression coefficients $\tau_\beta =$ `beta.prec`; (iv) the order of the polynomial *ord*; and (v) the continuity *cont*.

- `v <- jump.pw.poly.df.lin(v,s,s)`
  The X vector needn't be ordered for the previous 8 piecewise polynomials. However, in the case of discrete knots we can speed up evaluation of the polynomial by assuming that X *is* ordered. `jump.pw.poly.df.lin(.)` is exactly the same as `jump.pw.poly.d.lin(.)` except that it is 'faster' because it assumes that X is ordered – note that "`df`" stands for 'discrete knots' + 'fast'.

- `v <- jump.pw.poly.df.quad(v,s,s)`
  Exactly the same as `jump.pw.poly.d.quad(.)` except that it assumes the X vector to be ordered to save computation time.

- `v <- jump.pw.poly.df.cub(v,s,s)`
  Exactly the same as `jump.pw.poly.d.cub(.)` except that it assumes the X vector to be ordered to save computation time.

- `v <- jump.pw.poly.df.gen(v,s,s,s,s)`
  Exactly the same as `jump.pw.poly.d.gen(.)` except that it assumes the X vector to be ordered to save computation time.

- `v <- jump.pw.poly.pred(v,v)`
  This new BUGS-language element allows prediction based on ψ for any of the above piecewise polynomials. It works in exactly the same way as `jump.lin.pred.pred(.)`, etc., by providing indirect access to the hidden θ and β vectors. The first argument is the vector of piecewise polynomial nodes specified in the non-predictive part of the model. The second argument is the vector of x-values at which predicted values from the polynomial are required. Please see the `variable_selection_2.odc` example for further insight into how such predictive nodes work.

- `v <- jump.model.id(v)`
  For cases in which the hidden vector θ is discrete (i.e. for all variable selection problems and for all piecewise polynomials labelled with a "`d`" or a "`df`"), the `jump.model.id(.)` node-type can be used to make inferences about the model structure, which is defined by the joint quantity $\{\theta, k\}$. This simply provides a way of *efficiently* storing the information contained in $\{\theta, k\}$. You can look at the values of `jump.model.id(.)` nodes directly if you wish but they are not directly informative. For this reason I have provided a separate interface for 'deciphering' them – select

`Summarize...` from the new `Jump` menu. In the model specification, one `jump.model.id(.)` node is required for each block of 20 possible entities of interest. For example, suppose we have a variable selection problem with a total of 27 covariates (as in the `variable_selection_2.odc` example): then we need two `jump.model.id(.)` nodes to store the information contained in $\{\theta, k\}$. These must form contiguous elements of a vector or matrix and their single argument should be the relevant ψ-vector – please see the examples. The dialogue box that appears when you select `Summarize...` from the `Jump` menu is quite straightforward: enter the name of your `jump.model.id(.)` node (or vector) in the "`id node`" field and click on "`history`" and/or "`table`" (the "`mixing`" button doesn't work properly yet!). The `history` button produces a history plot of the model configuration over MCMC iterations – the grey-levels are proportional to the posterior probabilities associated with the relevant models. The `table` button generates a table of posterior model probabilities along with a table of marginal probabilities associated with each possible variable or knot-location. The "`razor`" field pertains to "Occam's Razor", which discards models with low posterior probability. The default value of `razor` is 0.99, which means that the table of posterior model probabilities will only contain those models that make up the first 99% of the posterior – you may set `razor` to any value $\in [0, 1]$.

Additional new components

Here we describe three new univariate distributions that can prove useful when analysing trans-dimensional models.

- `s` $\sim$ `dshifted.cat(v,s)`
  The prior distribution for the dimension parameter k is often naturally defined on the set $\{0, 1, 2, ..., Q\}$ where Q is some maximal value. The most flexible distribution for discrete variables defined on finite sets is the 'categorical' distribution. However, WinBUGS' standard categorical distribution, `dcat(.)`, has a minimum value of 1. The new `dshifted.cat(.)` distribution is exactly the same as `dcat(.)` except that the second argument specifies an arbitrary integer value to be used as the distribution's minimum (or first) value.

- `s` $\sim$ `dbern.aux(s)`
  If the response variable is binary, then it is natural to introduce a logit- or probit-link function into the model. However, this would mean that the full conditional distribution of the regression coefficients β is no longer available in closed form, which presents a serious problem for the current implementation of our approach. Fortunately, we can get around this, at least if we are happy to restrict ourselves to a probit link function, by introducing auxiliary variables into the model as follows. The following two specifications are exactly equivalent:

$$(i) \quad Z_i \sim \text{Bernoulli}(p_i); \quad \text{probit}(p_i) = \psi_i$$
$$(ii) \quad Z_i = I(a_i \geq 0); \quad a_i \sim \text{Normal}(\psi_i, 1)$$

  The second specification ensures that the full conditional for β is available in closed form (since a_i is *normally* distributed and ψ_i is a *linear* function of β). However, the Z_is are typically observed and in WinBUGS it is not possible for any node to be both observed and logical. Thus setting $Z_i = I(a_i \geq 0)$ is problematic. We get round this by equivalently stating $Z_i \sim \text{Bernoulli}(I(a_i \geq 0))$ (since nodes *can* be both observed and stochastic) but this is somewhat clumsy and so the new `dbern.aux(.)` distribution was conceived to 'tidy up' such specifications: $Z_i \sim$ `dbern.aux(`a_i`)` is equivalent to $Z_i = I(a_i \geq 0)$.

- `s` $\sim$ `dcat.aux(v, s, s)`
  The `dcat.aux(.)` distribution extends the above idea for binary data to ordered categorical data. The first argument should be a vector of 'cut-points' that define the ranges where the auxiliary variable corresponds to each category – the length of this cut-point vector should be equal to the number of categories minus one. The second argument is the auxiliary variable (which, again, should be assigned a normal distribution with mean ψ_i and variance 1), and the third argument is

an arbitrary integer constant to inform WinBUGS of what the minimum value of the categorical distribution is (e.g. zero).

These auxiliary variable distributions may of course be used outside the scope of trans-dimensional modelling, in which case there is no requirement for the full conditional of the regression coefficients to be available in closed form. Hence distributions other than normal distributions may be specified for the auxiliary variable, which gives rise to an equivalent link function other than probit. For example $a_i \sim \text{Logistic}(\psi_i, 1)$ corresponds to a logit link.

References

Lunn, D. J., Best, N. and Whittaker, J. C. (2008). Generic reversible jump MCMC using graphical models, *Statistics and Computing* **DOI: 10.1007/s11222-008-9100-0**.

Lunn, D. J., Whittaker, J. C. and Best, N. (2006). A Bayesian toolkit for genetic association studies, *Genet. Epidemiol.* **30**: 231–247.