



JAGS: Just Another Gibbs Sampler

Martyn Plummer¹

¹International Agency for Research on Cancer
Lyon, France

IceBUGS 2006



Outline

Design goals of JAGS

Minor differences from BUGS

Major differences from BUGS

- Data Transformations

- Censoring, truncation and ordering

- Observable Functions

Modules



BUGS can mean several things

A **language** for defining Bayesian hierarchical models

A **library** of sampling routines

An **application** for running the sampler

An **output processor** to interpret the results



Motivations for JAGS

1. To have an alternative BUGS language engine that
 - is open source
 - runs on Unix/Linux.
 - can be extended by the user
 - can be called from R
2. To create a platform for exploring ideas in Bayesian modelling



Code Base

The JAGS code is a completely independent of OpenBUGS

- Written in C++
- Draws together existing tools and libraries:
 - `flex`, `yacc` for building the compiler.
 - `libRmath` for distribution functions.
 - BLAS, LAPACK for linear algebra (needed by multivariate functions and distributions)



Data format

JAGS stores arrays in column-major order, like R, not row-major order, like BUGS.

$$\begin{pmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{pmatrix} \neq \begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix}$$

To avoid confusion, JAGS uses a different format for data and initial values.

Data is created in R and written to file with the `dump()` function.



Initialization

Variables without an explicit starting value are initialized to a *deterministic value*.

- a “typical value” drawn from the prior
- The exact meaning is specific to each distribution, e.g.
 - The median
 - The mode
 - The mean

The intention is to give reasonable starting values: even diffuse priors encode information about what we think the value should be

$X \sim \text{dnorm}(0, 1.0\text{E-}4)$

$Y \sim \text{dnorm}(58, 1.0\text{E-}4)$



Data Transformations

Data transformations in JAGS must be in a separate data block.

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



Data Simulation

The data block can include stochastic nodes. New values are generated by sampling from the prior.

```
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)
}
```



Data Simulation

- A data block may define a complete model, just like a `model` block.
- Unobserved values are filled in by a single sweep of forward sampling.
- You can generate data in one model, and analyze it in a completely different one.



Censoring, Truncation and Ordering in BUGS

The interval censoring construct $I(,)$ is used for three quite distinct purposes in OpenBUGS

Censoring *a posteriori* restriction.

Truncation *a priori* restriction.

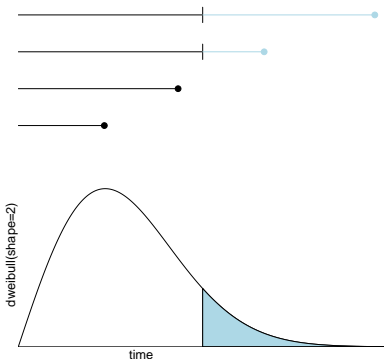
Ordering defines a potential function on a block of nodes.

JAGS tries to separate these concepts.



Example of Right Censoring

Right censoring arises in survival (time-to-event) studies when the observation period ends and some individuals have “survived” (e.g. disease-free).





BUGS: right censoring

```
for (i in 1:N) {  
  t[i] ~ dweib(mu[i], r[i]) I(t.cen[i],);  
}
```

where

- If subject i fails, $t[i]$ is observed and $t.cen[i]$ is 0.
- if subject i does not fail, $t[i]$ is missing and $t.cen[i]$ is the censoring time.

but you cannot simulate new interval censored data from this model



JAGS: Posterior Restriction

Use a novel distribution `dinterval`

$Y \sim \text{dinterval}(t, b);$

where

- t is a scalar
- b is a vector of breakpoints (length m)

$$Y = \begin{cases} 0 & \text{if } t \leq b_1 \\ i & \text{if } b_i < t \leq b_{i+1} \\ m & \text{if } t > b_m \end{cases}$$

This can be used

- in a `data` block to generate new interval censored data
- in a `model` block for inference on t .



The MICE Example

```
for(i in 1:N) {  
  is.censored[i] ~ dinterval(t[i], last.t[i]);  
  t[i] ~ dweib(r,mu[i]);  
}
```

To simulate new censored data you need an extra step:

```
T[i] <- ifelse(is.censored[i], NA, t[i])
```



BUGS: Prior Restriction

$I(,)$ can also be used to represent prior restriction on top-level parameters

```
theta ~ dnorm(0, 3) I(0,)
```

but **not** for variables with unobserved parameters

```
theta ~ dnorm(mu, tau) I(0,)
```



JAGS: Prior Restriction

JAGS uses the $T(,)$ notation for prior restriction.

$Y \sim \text{dfoo}(\text{theta}) \text{ T}(L,U);$

means

$$\begin{aligned} p(y \mid \theta, l, u) &= p(y \mid \theta, l \leq Y \leq u) \\ &= \frac{p(y \mid \theta)}{P(l \leq Y \leq u \mid \theta)} \end{aligned}$$

JAGS can calculate this using the *density* (dfoo) and *distribution* (pfoo) functions provided by `libRmath`.



Observable Functions

- Deterministic nodes are one of the most useful innovations of the BUGS language.
- Formally speaking, they are just a syntactic shorthand for describing relations between variables (stochastic nodes).
- They cannot be observed, which can be a nuisance.



Dinterval is an Observable Function

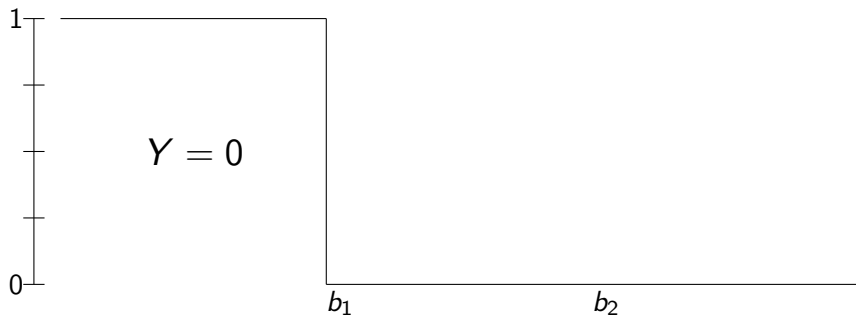
`dinterval` has features of both a distribution and a function.

- Formally a distribution, it defines a **likelihood function** for the parameters.
- “Random” sampling from `dinterval` always gives you the same answer. The value is a **deterministic** function of the parameters.

We can say `dinterval` is a distribution with zero degrees of freedom.

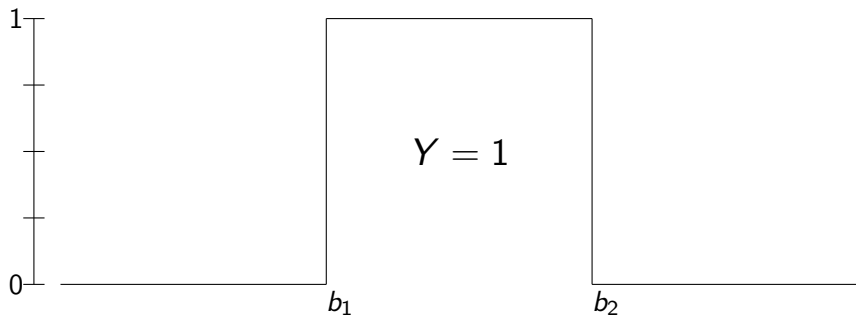


Example likelihood function from dinterval



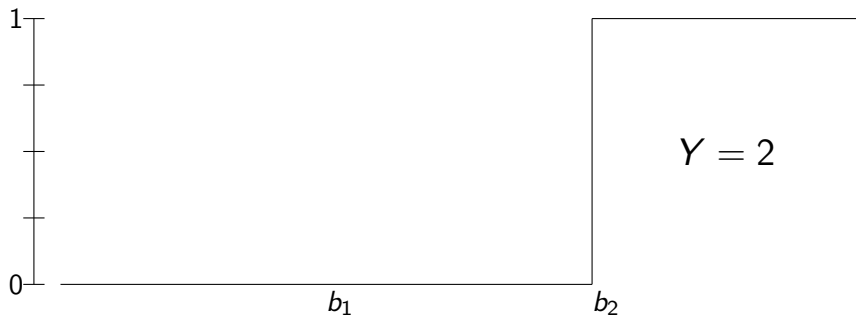


Example likelihood function from dinterval





Example likelihood function from dinterval





Distribution `dsum`

Distribution `dsum` also defines an observable function: the sum of two discrete random variables:

	$Y = 0$	$Y = 1$	
$X = 0$	Y_0^t		N_0^t
$X = 1$	Y_1^t		N_1^t
	Y^t	$N^t - Y^t$	N^t

This problem arises in ecological inference, when we observe the margins of a series of 2×2 tables:



Wakefield's hierarchical model for partially observed 2 x 2 tables

```
model {
  for (t in 1:n) {
    y0[t] ~ dbin(p[t,1], N0[t])
    y1[t] ~ dbin(p[t,2], N1[t])
    y[t] ~ dsum(y0[t], y1[t])
  }
  for (i in 1:2) {
    for (t in 1:n) {
      logit(p[t,i]) <- theta[t,i]
      theta[t,i] ~ dnorm(mu[i], tau[i])
    }
    ...
  }
}
```



Ad hoc Bayesian modelling

What happens when you step outside the class of problems handled by BUGS?

- Many papers on the application of Bayesian graphical modelling are **demonstration projects** using custom software.
- They show that the Bayesian approach **can be applied** to a particular problem, without making it easier for the reader to follow in their footsteps.
- This struck me as a bit of a waste.



JAGS modules

The development version of JAGS has a modular architecture.
Modules can be dynamically loaded to add new

- Functions
- Distributions
- Samplers

The aim is to have an extensible platform for MCMC.



Current modules

core Core functions: inline operators whose existence, and behaviour, is hard-coded into the compiler.

generic Generic samplers that use only abstract properties of a distribution

- Real slice sampler
- Discrete slice sampler
- Finite sampler
- *Random-walk Metropolis*

bugs BUGS functions, distributions and conjugate samplers.



Example: Continuous-time multi-state Markov models

JAGS has a novel distribution, `dmstate` for modelling interval-censored transitions in continuous time Markov chains

$S[j] \sim \text{dmstate}(S[j-1], \text{deltat}[j], \text{Lambda}[,])$

where

- $S[]$ is a vector of observed states ($1, \dots, m$)
- $\text{deltat}[]$ is a vector of elapsed times
- $\text{Lambda}[,]$ is an $m \times m$ transition intensity matrix



Example: normal mixture models

Normal mixture models can be defined in the BUGS language using data augmentation and nested indexing

```
Y[i] ~ dnorm(mu[G[i]], tau[G[i]])  
G[i] ~ dcat(p[])
```

But mixing can be poor and the deviance is focused on $G[]$.
Novel distribution `dnormmix` integrates out G .

```
Y[i] ~ dnormmix(mu[], tau[], p[])
```

We may also wish to use special sampling techniques for multi-modal posterior e.g. simulated tempering.



Future plans

- More modules (msm, glm, mix)
- R interface (awaiting a stable library API)
- Running parallel chains in separate threads
- Allowing chain graphs (paves the way for spatial processes)
- Vectorized version of the BUGS language.