JAGS: Just Another Gibbs Sampler

Martyn Plummer

International Agency for Research on Cancer
Lyon, France

AppliBUGS 8 November 2007
Outline

1. Why JAGS?
2. Extending JAGS
3. Differences between JAGS and BUGS
4. rjags
5. Conclusions
BUGS

Bayesian inference Using Gibbs Sampling

Bayesian inference Using Gibbs Sampling

Bayesian inference Using Gibbs Sampling


- A **language** for defining Bayesian hierarchical models
- A **library** of sampling routines
- A **user interface** for running the sampler
- An **output processor** to interpret the results
JAGS: Just Another Gibbs Sampler
JAGS

Just Another Gibbs Sampler
JAGS is an open-source engine for the BUGS language written in C++. It is not a port of WinBUGS, but has a completely independent code base.

Just Another Gibbs Sampler
Motivations for JAGS

1. To have an alternative BUGS language engine that
   - is extensible,
   - runs on Unix/Linux,
   - can interface to R.

2. To create a platform for exploring ideas in Bayesian modelling
Why C++?

C++ is an object oriented language, which easily allows creation of a virtual graphical model: an internal representation of the BUGS model in computer memory.

In contrast, OpenBUGS is written in component pascal and is built on the black box component framework.
Advantages of C++

- Interface to existing software written in C, C++, or Fortran
  - Linear algebra routines for matrix operations
    - BLAS, LAPACK, mkl, acml, ...
  - R math library for statistical functions
    - dnorm, pnorm, qnorm, rnorm
  - Random number generators
  - Sampling methods
- Portability: every platform has a C++ compiler and standard library.
- Popularity: there are more experienced developers
A shared library containing
- A compiler for turning a BUGS-language description of a model into an internal graph.
- Abstract base classes for elements of the BUGS language (functions, distributions), and objects that act on the graph (samplers, RNGs).

2 Dynamically loadable modules containing concrete classes for the above.

3 User interfaces
- Command-line interface
- Experimental R package (rjags).
JAGS Modules

Modules can be dynamically loaded at runtime to extend the functionality of JAGS. A module can define five kinds of objects:

1. Function
2. Distribution
3. SamplerFactory
4. RNGFactory
5. MonitorFactory
Functions and Distributions

These are the building blocks of the BUGS language

\[ y \leftarrow \exp(x) \] #Deterministic relation
\[ x \sim \text{dnorm}(\mu, \tau) \] #Stochastic relation

Modules may define novel functions and distributions

\[ Y \leftarrow \text{mexp}(X) \] #Matrix exponential
\[ z \sim \text{dnormmix}(\mu, \tau, p) \] #Normal mixture

Novel distributions may require novel samplers.
Example: Enzyme data

- The figure shows the results of a biomarker experiment designed to distinguish “fast” and “slow” acetylator phenotype.
- Acetylator phenotype is related to polymorphisms in the NAT2 gene.
- This example was used by Richardson and Green (1997)
Finite mixture models

The enzyme data can be represented by a finite mixture model:

\[
P(y_i | x_i, \mu, \sigma) = \phi \left( \frac{y_i - \mu x_i}{\sigma x_i} \right)
\]

\[
P(X_i = x | \pi) = \pi_x
\]

where

- \( X_i \) is a latent indicator variable for the group to which individual \( i \) belongs
- \( \phi(.) \) is the standard normal density
Alternative representation of finite mixture models

It is computationally simpler to marginalize over the latent variables $X_1 \ldots X_n$.

$$p(y_i \mid \pi, \mu, \sigma) = \sum_{g=1}^{G} \pi_g \phi \left( \frac{Y_i - \mu_g}{\sigma_g} \right)$$

The `mix` module of JAGS defines the novel distribution `dnormmix`

```r
for (i in 1:N) {
  y[i] ~ dnormmix(pi, mu, tau)
}
sigma <- sqrt(1/tau)
```
Analyzing mixture models with MCMC

- Mixture models are notoriously hard to analyze by MCMC.
- We use *tempered transitions* (Neal, 1996) to jump between nodes of the multimodal posterior.
- This technique is implemented in the mix module of JAGS.
A 3-component mixture model

- Jumps in the traceplot represent label switching, in which \((\mu_1, \mu_2, \mu_3)\) are permuted.
- The marginal posterior distributions of \(\mu_1, \mu_2, \mu_3\) should be the same in the long run.
Incompatible differences between JAGS and BUGS

I have deliberately chosen to make JAGS incompatible with BUGS in four areas.

- Scripting language
- Data format
- Censoring
- Data transformations
The JAGS scripting language

Syntax borrowed from Stata.

```plaintext
model in blocker.bug
data in blocker-data.R
compile
parameters in blocker-ninit.R
initialize
update 3000
monitor set d, thin(10)
monitor set delta.new, thin(10)
monitor set sigma, thin(10)
update 30000
coda *
exit
```
JAGS uses the R `dump()` format for data and initial values.

- No need to transpose matrices before dumping
  - JAGS uses column major ordering of arrays, like R and Fortran.
- JAGS can parse any numeric vector or array dumped from R.
  - Attributes can be parsed by JAGS but are ignored.
Censoring and truncation

- The I(,) construct is used in WinBUGS to represent censoring (*a posteriori* restriction).
- Confusingly, it is also used to represent truncation of top-level parameters (*a priori* restriction).
- It is invalid to use I(,) for truncation for any distribution with unobserved parameters.

JAGS does not have the I(,) construct

- JAGS uses the T(,) construct for truncation
- Censoring is represented by the novel distribution dinterval
Interval censoring in JAGS

An example of a right-censored failure time:

```r
is.censored[i] ~ dinterval(t[i], t.censor[i]);
t[i] ~ dweib(r,mu[i]);
```

where

- `t.censor[i]` is a censoring time
- `is.censored[i]` is a censoring indicator
- `t[i]` is a failure time
  - missing (NA) if censored.
Data transformations

Data transformations must be in a separate data block

**BUGS**

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

**JAGS**

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

The data block can also be used to simulate data from a model.
JAGS uses a dialect of the BUGS language that is slightly different from WinBUGS.

Most of these changes reflect my experience as a BUGS and R user.

The general trend is to make BUGS more “S-like” while still allowing WinBUGS code to run on JAGS.
JAGS allows you to query the size of vectors and arrays supplied in the data file.

**BUGS**
for (i in 1:N) {
  y[i] ~ dnorm(mu[i], tau)
}

**JAGS**
for (i in 1:length(y)) {
  y[i] ~ dnorm(mu[i], tau)
}

No need to supply N in the data file
- There is also a `dim` function for array dimensions
Initial values

- WinBUGS generates initial values from the prior distribution
  - But diffuse priors generate initial values incompatible with the data
- JAGS tries to provide sensible initial values:
  - fixed effects A deterministic typical value from the prior distribution (mean, median, mode ...)
  - random effects A random sample from the prior.
Scalar functions with scalar arguments are automatically vectorized

**BUGS**

```plaintext
for (i in 1:N) {
    for (j in 1:M) {
        D[i,j] <- a + b * C[i,j]
    }
}
```

**JAGS**

```plaintext
D <- a + b * C
```
Use vectorization with caution

R users may be tempted to vectorize everything. But be careful not to create bottlenecks in the model

Fast

```r
for (i in 1:N) {
  y[i] ~ dnorm(mu[i], tau)
  mu[i] <- lp[i] + e[i]
  e[i] ~ dnorm(0, tau.e)
}
```

Slow

```r
for (i in 1:N) {
  y[i] ~ dnorm(mu[i], tau)
  e[i] ~ dnorm(0, tau.e)
}
mu <- lp + e
```

The right hand side recalculates the whole of \( \mu \) every time \( e[i] \) is updated!
Vector-valued functions

JAGS has more vector- and array-valued functions

**BUGS**

\[
\text{for (i in 1:N) \{ }
  \text{for (j in 1:M) \{ }
    \text{ip <- a[i] * B[i,j] * c[j]}
  \text{\}}
\text{\}}
\]

**JAGS**

\[
ip <- t(a) \%\% B \%\% c
\]
Embedding expressions

Deterministic nodes can be embedded inside other expressions

**BUGS**
\[
\begin{align*}
y[i,j] & \sim \text{dnorm}(\mu[i,j], \tau) \\
\mu[i,j] & \leftarrow a[i] + b[j]
\end{align*}
\]

**JAGS**
\[
\begin{align*}
y[i,j] & \sim \text{dnorm}(a[i]+b[j], \tau)
\end{align*}
\]
JAGS has an experimental R interface "rjags". It will be released (uploaded to CRAN) after JAGS 1.0.0.

```r
> library(rjags)
> m <- jags.model("line.bug", data=line.data)
Compiling model graph
Resolving undeclared variables
Allocating nodes
Checking graph
Graph Size: 37
```
A `jags.model` is not a fitted model object.

It is an object that we can query to get (dependent) random samples for the parameters.

`In the long run`, these samples will be from the posterior distribution.
Drawing Samples

To get samples from the posterior distribution

```r
x <- model.samples(m, variable.names=c("alpha", "beta", "tau"),
                   n.iter=1000)
```

The return value `x` is a list containing sampled values for the requested variables.
A model can be updated without drawing samples

```
m$update(1000)
```

This changes the state of the object m, and makes it more likely to generate samples close to the posterior distribution.
A `jags.model` has an object-oriented interface.

- `m$ptr()`  A pointer to an external C++ object created by the JAGS library.
- `m$data()`  A copy of the model data.
- `m$model()`  A character vector defining the BUGS model.
- `m$state()`  The current parameter values.
- `m$update(n)`  Updates the sampler by $n$ iterations.
- `m$recompile()`  Recompiles the model.
Current status and future plans

- The JAGS library is complete and documented
  - JAGS 1.0.0 to be released before end 2007.
- Developer manual required to show other users how to write modules.
- R package rjags to be released after JAGS 1.0.0.
JAGS runs on Windows. But I am not a habitual Windows user and it needs better support.

- Windows users expect an installer (e.g. innosetup).
- Needs a better GUI than the DOS console.
Further information

- JAGS home page: http://mcmc-jags.sourceforge.net
- Source code repository: http://sourceforge.net/projects/mcmc-jags