Bayesian modelling in R with JAGS

Martyn Plummer[1]

[1] International Agency for Research on Cancer
Lyon, France

UseR! 2006

Outline

Design goals of JAGS

JAGS Modules

R interface

Outline

Design goals of JAGS

JAGS Modules

R interface

BUGS (Bayesian Inference Using Gibbs Sampling)

- A *declarative language* for defining Bayesian hierarchical models.
  - see Thomas, A, *R News*, Vol 6/1, 17–21.
- An application for analyzing such models by Markov Chain Monte Carlo.
  - `http://www.mathstat.helskinki.fi/openbugs`.

A Linear regression model in BUGS

```
model {
    for (i in 1:N) {
          mu[i] <- alpha + beta*(x[i] - x.bar);
          Y[i]    ~ dnorm(mu[i],tau);
    }
    x.bar   <- mean(x[]);
    alpha    ~ dnorm(0.0,1.0E-4);
    beta     ~ dnorm(0.0,1.0E-4);
    tau      ~ dgamma(1.0E-3,1.0E-3);
    sigma   <- 1.0/sqrt(tau);
}
```

Motivations for JAGS (Just Another Gibbs Sampler)

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

Most of these goals are now obsolete.

Current structure of JAGS

1. **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
   - Basic R package (rjags).

Outline

Design goals of JAGS

JAGS Modules

R interface

Modules

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

1. Function
2. Distribution
3. SamplerFactory
4. RNGFactory

The JAGS library is agnostic about *how* modules are dynamically loaded. The two user interfaces use different methods:

- `ltdl` for the CLI.
- `dyn.load()` for rjags

Functions and Distributions

These are the building blocks of the BUGS language

```
y <- exp(x) #Deterministic relation
x ~ dnorm(mu, tau) #Stochastic relation
```

Modules may define novel functions and distributions, which are added to a static table in the jags library.

```
Y <- mexp(X) #Matrix exponential
z ~ dnormmix(mu, tau, p) #Normal mixture
```

Novel distributions may require novel samplers.

SamplerFactories

- A `SamplerFactory` object recognizes a suitable set of Nodes in the graph to sample, based purely on the *graphical structure* of the model.
- It generates a new Sampler object specifically for those nodes.
- JAGS works through the list of SamplerFactories until there are no more Nodes in the graph left to sample.
- Precedence is determined by load order.

RNGFactories

- Each parallel chain has its own RNG.
- RNGFactories must generate independent RNGs for parallel chains.
- The `baserng` module uses code borrowed from R and generates an RNG with a *different generator for each chain*:
  1. Wichmann-Hill
  2. Marsaglia-Multicarry
  3. Super-Duper
  4. Mersenne-Twister
- We could also create wrappers for the GNU scientific library, or L'Ecuyer RNGStreams.

Initializing the RNG

Initial state of the RNG is set from the date stamp. You can also supply an initial seed

```
".RNG.name" <- "base::Wichmann-Hill"
".RNG.init" <- 71113
```

or use a state saved from a previous session

```
".RNG.name" <- "base::Wichmann-Hill"
".RNG.state" <- c(19900, 14957, 25769)
```

Outline

Design goals of JAGS

JAGS Modules

R interface

Package rjags

The rjags package loads the default JAGS modules.

```
> library(rjags)
Loading required package: coda
Loading required package: lattice
loading JAGS modules
basefunctions
baserngs
basesamplers
bugs
```

Defining a JAGS model

A JAGS model is defined by:

1. A model description (in a file)
2. The data (a list of vectors/matrices/arrays)
3. A set of initial values for each chain (optional)

```
> m <- jags.model("line.bug", data=line.data)
Compiling model graph
Resolving undeclared variables
Allocating nodes
Checking graph
Graph Size: 37
```

JAGS model objects

- 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

```
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.

Burn-in

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.

The Console class

The C++ class `Console` provides a “safe” interface to the JAGS library.

- Catches exceptions
- Prints informative information and/or error messages to output streams

In R, a jags.model object contains an external pointer to a JAGS Console object.

- `http://www.stat.uiowa.edu/~luke/simpleref.html`

A `jags.model` has an object-oriented interface (*q.v.* the scoping demo)

`m$ptr()`	The external C++ pointer
`m$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

You can never go home

- A `jags.model` object can persist between R sessions, but the external pointer does not.
  - Interface to external pointers takes care of this.
- A `jags.model` stores sufficient data to allow it to be recompiled.
- But the exact state of the model can never be restored!
  - Samplers can have an arbitrary internal state

Help Wanted!

- Compiling on Windows, and other platforms.
- R class for simulated output.

The JAGS home page:
`http://www-ice.iarc.fr/~martyn/software/jags`