STAT 431 — Applied Bayesian Analysis — Course Notes

Introduction to Computational Methods Part 2

Spring 2019

Gibbs Sampling Software

Software is available that automates the Gibbs sampling process and extends it to handle non-conjugacy.

Cowles illustrates use of **OpenBUGS** (`http://www.openbugs.net`).

We will instead use **JAGS** (`http://mcmc-jags.sourceforge.net/`).

JAGS and OpenBUGS are similar, but have some important differences.

OpenBUGS:

- Point-and-click interface (Windows)
- Extensive online tutorial with many classic examples

JAGS:

- Developed actively
- Cross-platform (easily)
- Web site more stable

Both can interface with R through appropriate packages.

JAGS has its own interface — a simple command line — but we will instead run it from within R using package `rjags`.

(Other JAGS-related R packages exist — see JAGS manual.)

To run JAGS (or OpenBUGS) you need:

- the data
- a specification of the model (including priors)
- the initial values (for all chains)

R/JAGS Example 8.5:

Population Proportion

BEWARE: JAGS and OpenBUGS are not fully compatible — OpenBUGS code (in Cowles and on web sites) may need modifications to be used as JAGS code.

Download and use the current JAGS user manual to avoid problems.

JAGS Model Specification

Every model is specified in a `model` block:

```
model {
   ... statements ...
}
```

Many statements have the form

```
variablename ~ distribution(params)
```

Example:

$$
\begin{aligned}
Y \mid \pi &\sim \text{binomial}(n = 10,\ \pi) \\
\pi &\sim \text{beta}(\alpha = 2,\ \beta = 1)
\end{aligned}
$$

```
model {
  y ~ dbin(pi, 10)
  pi ~ dbeta(2, 1)
}
```

- the order of the statements *does not* matter
- the order of the parameters *does* matter
- variables do not need to be "declared"

There is a **graph** for this model:

[Draw model graph ...]

The graph says that:

- π and y are both specified as random variables
- π is specified marginally (unconditionally)
- y is specified conditionally on π

Variables may be specified together as a **vector** or as an **array**.

Eg: If `y` is a vector, `y[i]` is its `i`th element.

Vectors can be handled easily using a `for` statement, e.g. if `y` has `n` elements,

```
for (i in 1:n) {

  y[i] ~ distribution(params)

}
```

Important: `n` must be a constant (non-random).

Example:

$$
\begin{aligned}
Y_1, \ldots Y_n \mid \mu, \tau^2 &\sim \text{ i.i.d. } \mathrm{N}(\mu,\, 1/\tau^2) \\
\left.\begin{aligned} \mu &\sim \mathrm{N}(0, 100) \\ \tau^2 &\sim \mathrm{gamma}(0.01, 0.01) \end{aligned}\right\} &\text{ independent}
\end{aligned}
$$

```
model {
  for (i in 1:n) {
    y[i] ~ dnorm(mu, tausq)
  }
  mu ~ dnorm(0, 0.01)
  tausq ~ dgamma(0.01, 0.01)
}
```

The model graph:

[Draw model graph ...]

A **plate** indicates that the variables on it are repeated as vector/array elements.

The length of a *data* vector `y` can be referenced as `length(y)`:

```
model {
  for (i in 1:length(y)) {
    y[i] ~ dnorm(mu, tausq)
  }
  ...
}
```

This avoids the need to specify the length as a separate data value.

Improper priors, such as

$$\mu \sim 1\, d\mu$$

are NOT allowed in JAGS.

You can instead use a "vague" proper prior, such as

```
mu ~ dnorm(0, 0.000001)
```

though whether this is "vague" enough depends on the context.

What if we want inference about σ^2 (not just τ^2)?

We could use

```
model {
  for (i in 1:length(y)) {
    y[i] ~ dnorm(mu, tausq)
  }
  mu ~ dnorm(0, 0.000001)
  tausq ~ dgamma(0.01, 0.01)

  sigmasq <- 1/tausq
}
```

A statement

```
variablename <- expression
```

represents a **deterministic** relationship: the variable is computed according to the expression.

The expression may use the four usual arithmetic symbols, the unary minus, the power symbol `^`, and certain scalar functions — see JAGS documentation.

Deterministic relationships are *hollow* lines in the model graph:

[Draw model graph ...]

Important: Variables having data values are *not allowed* on the left-hand side of a deterministic relationship (<-).

Eg: Suppose data `y[i]` needs to be log-transformed to normality.

NOT allowed:

```
y[i] <- exp(z[i])
z[i] ~ dnorm(mu, tausq)
```

Also NOT allowed (in the `model` block):

```
z[i] <- log(y[i])
z[i] ~ dnorm(mu, tausq)
```

Use a preceding `data` block to transform data:

```
data {
  for (i in 1:length(y)) {
    z[i] <- log(y[i])
  }
}
model {
  for (i in 1:length(z)) {
    z[i] ~ dnorm(mu, tausq)
  }
  ...
}
```

The `data` block is executed (once) before the model is run.

Example: Wikipedia Article Modifications

For $n = 10$ random English Wikipedia articles, time (days) since last modification is recorded:

$$Y_1, \ldots Y_{10} \sim \text{i.i.d.} \quad \text{(given parameters)}$$

We want to know about the *median* time, and perhaps the variation.

Preliminary analysis (Q-Q plot) suggests approximate normality on the *log* scale:

$$\ln(Y_1), \ldots \ln(Y_{10}) \mid \mu, \sigma^2 \sim \text{i.i.d. } \mathrm{N}(\mu, \sigma^2)$$

Consider prior

$$\left.\begin{aligned} \mu &\sim \mathrm{N}(0, \sigma_0^2) \\ \sigma^2 &\sim \mathrm{IG}(\alpha, \beta) \end{aligned}\right\} \text{ independent}$$

We will try

$$\sigma_0^2 = 10000 \qquad \alpha = 0.0001 \qquad \beta = 0.0001$$

because they produce a "vague" prior.

We might want inference for μ, σ^2, and

$$e^{\mu} \quad = \quad \text{(conditional) median of } Y$$

(Why is that the median?)

Remember, JAGS uses

$$\tau^2 \quad = \quad 1/\sigma^2$$

Graph:

[Draw model graph ...]

Note: Use a square (instead of a circle) for values that are "constants," i.e. not on the left-hand side of ~ or <-.

R/JAGS Example 8.6:

(Transformed) Normal Sample

Convergence Assessment

- Initial Values

  ... should be **overdispersed** — some should be chosen far away from the values you expect to see for the parameters

- Trace (History) Plots

  check for convergence time and speed of "mixing"

  Eg: showing transient (initialization) effects

  [Draw trace plot ...]

Eg: showing lack of convergence

[Draw trace plot ...]

Eg: showing slow mixing (which gives high MC error)

[Draw trace plot ...]

- Autocorrelation Plots

  check for serial dependence

  Eg: showing high correlations (which gives slow mixing)

  [Draw autocorrelation plot ...]

- Gelman-Rubin Statistic and Plots

The **Gelman-Rubin statistic**, or "potential scale reduction factor," helps to monitor convergence when using multiple chains with overdispersed starting points.

For a specific monitored quantity, it is essentially

$$R = \sqrt{\frac{\hat{V}}{W}}$$

where

$\hat{V}$ = an estimated variance of the quantity based on all chains ("pooled")

W = average of the estimated variances from each chain individually

Idea:

[Draw trace plots ...]

Recommend $R < 1.05$ to declare convergence.

The `coda` function `gelman.plot` plots R versus iterations used, which makes it easier to read how many iterations should be burned.

[Draw example plot ...]

Posterior Predictive Distributions

JAGS offers two ways to sample from a posterior predictive distribution:

- add a new variable to your model (e.g. "`ynew`")
- add a "missing" value in your data (using "`NA`")

Example: Wikipedia Article Modifications (continued)

Want:

- a range containing the days since modification for 95% of all articles
- the estimated percentage of articles modified in the last 30 days

We will add a new variable called `ynew` and another variable indicating whether `ynew` is less than or equal to 30.

R/JAGS Example 8.7:

(Transformed) Normal Sample with Posterior Prediction

Notice: Certain transformations (including `log`) are allowed on the left hand side of a deterministic relationship:

```
log(ynew) <- znew
```

Q: Why didn't we use the "missing" value method?

A: Because the transformation of `y[i]` in the data block (on the right hand side of a `<-` statement) requires all non-missing values.