Practical Markov chain Monte Carlo

Simon Jackman

Stanford University
http://jackman.stanford.edu/BASS

February 3, 2012

BUGS and JAGS

- BUGS (Spiegelhalter, Thomas and Best 2000) and OpenBUGS: Bayesian Inference Using Gibbs Sampling
- JAGS (Plummer 2010a): Just Another Gibbs Sampler, a look-alike to BUGS, with some subtle and welcome enhancements, runs natively on Mac/PC/Linux (hence my preferred software).
- I call JAGS from R via the rjags library (Plummer 2010b); I never "run" JAGS as a separate program. See §6.3.
- JAGS programs are not "procedural"; they are "declarative", defining a model.

R/JAGS Workflow

1. develop JAGS model declaration syntax in a file (extension is `.bug`)
2. in R, prepare data, pass data and model syntax to JAGS via the `jags.model` command
3. JAGS parses the data and model syntax, deduces the form of the implied DAG, creates a Gibbs sampler for the DAG
4. JAGS exploits conjugacy when it can recognize it; will use slice sampling, Metropolis etc when a conditional distribution is not recognizable in closed form etc.
5. `coda.samples` or `update` command actually runs the Gibbs sampler for a specified number of iterations
6. inspect results in R; `plot`; `summary`

JAGS Example Program for Regression

JAGS code

```
model{
        ## loop over the data
        for(i in 1:n){
              ## form the conditional mean of y
              mu[i] <- beta[1] + beta[2]*x[i]

              ## tau is inverse variance (precision)
              y[i] ~ dnorm(mu[i],tau)
        }

        ## priors
        beta[1] ~ dnorm(0,.0001)
        beta[2] ~ dnorm(0,.0001)

        ## contrast
        ## beta[1:2] ~ dmnorm(b0[1:2],B0[1:2,1:2])
        ## put a (nonconjugate) uniform prior
        ## on error std deviation, sigma
        sigma ~ dunif(0,100)
        ## convert std dev to a precision
        tau <- pow(sigma,-2)
}
```

- R like syntax; looping, subscripting, comments, assignment, etc.
- `dnorm` for a normal sampling model; many other densities available
- note non-conjugate prior on the standard deviation `sigma`
- a few commands for transformations of quantities: e.g., `pow`, `inverse` (for matrices)

Data Preparation in R

R code

```
library(pscl)
data(absentee)
attach(absentee)

## create variables for regression analysis
y <- (absdem - absrep)/(absdem + absrep)*100
x <- (machdem - machrep)/(machdem + machrep)*100

## environment for passing to JAGS
forJags <- list(y=y,
                x=x,
                n=length(y))

## initial values, one chain
inits <- list(list(beta=c(0,0),sigma=5,
                   .RNG.seed=1234,
                   .RNG.name="base::Mersenne-Twister"))
```

- this examples uses data from my R packages, `pscl`
- the list `forJags` will be passed to JAGS, contains `y` and `x`
- we create a list `inits` that will be used to initialize the Gibbs sampler
- we set the seed and specify the PRNG to use

Passing to JAGS

R code

```
library(rjags)
foo <- jags.model(file="regression.bug",
                  data=forJags,
                  inits=inits)
out <- coda.samples(model=foo,
                    variable.names=c("beta","sigma"),
                    n.iter=50000,thin=5)
```

- `jags.model` passes to JAGS
  1. syntax in `file`
  2. data in `forJags` to JAGS
  3. initial values for the Markov chain in `inits`
- JAGS:
  1. parses model declaration and data
  2. deduces DAG: what is fixed data, what are random/unobserved quantities; conditional independence relations among nodes of the DAG.
  3. compiles Gibbs sampling code
  4. will run a default of 1,000 iterations to let any adaptive sampling algorithms get calibrated.

Passing to JAGS

R code

```
library(rjags)
foo <- jags.model(file="regression.bug",
                  data=forJags,
                  inits=inits)
out <- coda.samples(model=foo,
                    variable.names=c("beta","sigma"),
                    n.iter=50000,thin=5)
```

- `coda.samples`:
  1. takes model stored in `foo`, the output of the `jags.model` command
  2. will run `n.iter` iterations of the Gibbs sampler
  3. will `thin` the output by 5; i.e., saving only every 5-th iteration to memory
  4. results come back to `R` in the object `out`
  5. `coda.samples` produces objects of class `coda`; see the `coda` package (Plummer et al. 2008).

Convergence and Run-Length Diagnostics, §6.2

- how long should we run a MCMC algorithm?
- no deterministic stopping rules of the sort we have for optimization algorithms (e.g., when a normed gradient vector is within some ε of zero).
- and more fundamentally: are we sure that the sampler has reached the equilibrium/stationary distribution p of the Markov chain? i.e., is t big enough such that $p_t \equiv p_{t-1}$ in

$$p_t = \int_{\boldsymbol{\Theta}} K(\boldsymbol{\theta}^{(t-1)}, \cdot) p_{t-1} d\boldsymbol{\theta}^{(t-1)}$$

- tests of stationarity from time-series statistics

Convergence and Run-Length Diagnostics, §6.2

- Geweke
- Raftery-Lewis
- Multiple-chains; Gelman-Rubin

Geweke (1992) mean stationarity test

- MCMC output: $\{g(\boldsymbol{\theta}^{(t)})\}$
- spectral density of $\{g(\boldsymbol{\theta}^{(t)})\}$ can be used to estimate the variance of an ergodic average $\bar{g} = T^{-1} \sum_{t=1} g^{(t)}$.
- compare two $\bar{g}$, once based on a set of T_A "early" iterations from the Markov chain output, the other based on a set of T_B "late" iterations
- difference of means divided by the asymptotic standard error of the difference $\rightarrow N(0, 1)$ as $T \rightarrow \infty$.
- defaults are $T_A = .1$ (1st 10%) of iterations, $T_B = .5$ (last 50%).
- implemented as `geweke.diag` in the `coda` package in `R`

Raftery and Lewis (1996) run-length assessment

- how long must the MCMC algorithm be run in order to be get an accurate estimate of an extreme quantile?
- implemented as `raftery.diag` in the `coda` package in `R`.
- defaults are to assess the accuracy of the MCMC-based estimate of the $q = .025$ quantile with a 95% bound on the estimate no greater than .005 (in quantile terms).
- procedure forms the binary sequence $\{Z^{(t)}\}$, with $Z_t = \mathcal{I}(\theta^{(t)} \leq u_q)$, where $\mathcal{I}$ is a binary indicator function and u_q is the q-th quantile of $\{\theta^{(t)}\}$.
- estimates how big T has to be such that $\{Z^{(t)}\}$ looks like a Markov chain (which it can't be, but...)
- with defaults, requires a minimum of 3,746 samples to be implemented
- can lead to surprisingly long estimates of desired run-length

Multiple chains (Gelman and Rubin 1992)

- start multiple MCMC samplers from dispersed starting points
- do they "mix"?
- formal test: assess magnitude of cross-chain variance to total variance. This should be small.
- Test-statistic:

$$\sqrt{\hat{R}} = \sqrt{\frac{\widehat{\text{var}}^{+}(\psi|y)}{W}}.$$

  where $\widehat{\text{var}}^{+}(\psi|y) = \frac{T\text{-}1}{T}W + \frac{1}{T}B$, W is the (average) within-chain variance and B is the between-chain variance for some scalar estimand ψ (usually an element of $\boldsymbol{\theta}$).
- $\sqrt{\hat{R}} \to 1$ as $T \to \infty$; $\sqrt{\hat{R}} < 1.2$ are "acceptable"
- `gelman.diag` in `coda`

Example: one-way ANOVA for presidential elections data

$$y_{ij} \sim N(\mu + \alpha_j, \sigma^2) \qquad \alpha_j \sim N(0, \omega^2) \qquad \mu \sim N(0, 10^2)$$
$$\sigma \sim \text{Unif}(0, 10) \qquad \omega \sim \text{Unif}(0, 10)$$

- 50,000 iterations
- Geweke diagnostic:

R output

```
[[1]]

Fraction in 1st window = 0.1
Fraction in 2nd window = 0.5

      mu    omega    sigma
 0.83725  0.01474 -0.15899
```

Example: one-way ANOVA for presidential elections data

$$y_{ij} \sim N(\mu + \alpha_j, \sigma^2) \qquad \alpha_j \sim N(0, \omega^2) \qquad \mu \sim N(0, 10^2)$$
$$\sigma \sim \text{Unif}(0, 10) \qquad \omega \sim \text{Unif}(0, 10)$$

- 50,000 iterations
- Raftery-Lewis diagnostic:

R output

```
[[1]]

Quantile (q) = 0.025
Accuracy (r) = +/- 0.005
Probability (s) = 0.95

       Burn-in  Total Lower bound  Dependence
       (M)      (N)   (Nmin)       factor (I)
 mu    24       27380 3746         7.31
 omega 5        5830  3746         1.56
 sigma 5        5691  3746         1.52


JAGS model:

model{
        ## loop over data frame
        for(i in 1:N){
```

Tricks of the trade

- thinning §6.4.1
- blocking §6.4.2
- re-parameterization §6.4.3
- over-parameterization §6.5

Thinning §6.4.1

- Not a statistical issue; a computing and storage issue.
- Slow mixing MCMC takes many iterations to generate a good random tour of the posterior density
- Too much MCMC output to store
- Retain only every 10th iteration (or some appropriate *thinning* interval)
- We are throwing away data; but successive draws look so much like previous draws, so efficiency loss not huge.

Thinning §6.4.1

Thinning Interval	Nominal n	Mean	SD	2.5%	AR(1)
1	1,000,000	0.01	1.00	-1.94	0.98
5	200,000	0.01	1.00	-1.94	0.90
10	100,000	0.01	1.00	-1.94	0.81
25	40,000	0.01	0.99	-1.93	0.60
50	20,000	0.01	0.99	-1.93	0.36
100	10,000	0.01	1.00	-1.93	0.12
250	4,000	-0.0002	0.99	-1.90	0.02
500	2,000	0.02	0.98	-1.90	-0.02
1,000	1,000	0.04	0.98	-1.86	-0.02
2,500	400	-0.03	0.97	-1.85	0.03
5,000	200	0.05	0.97	-1.83	-0.13
10,000	100	0.14	0.99	-1.82	0.06

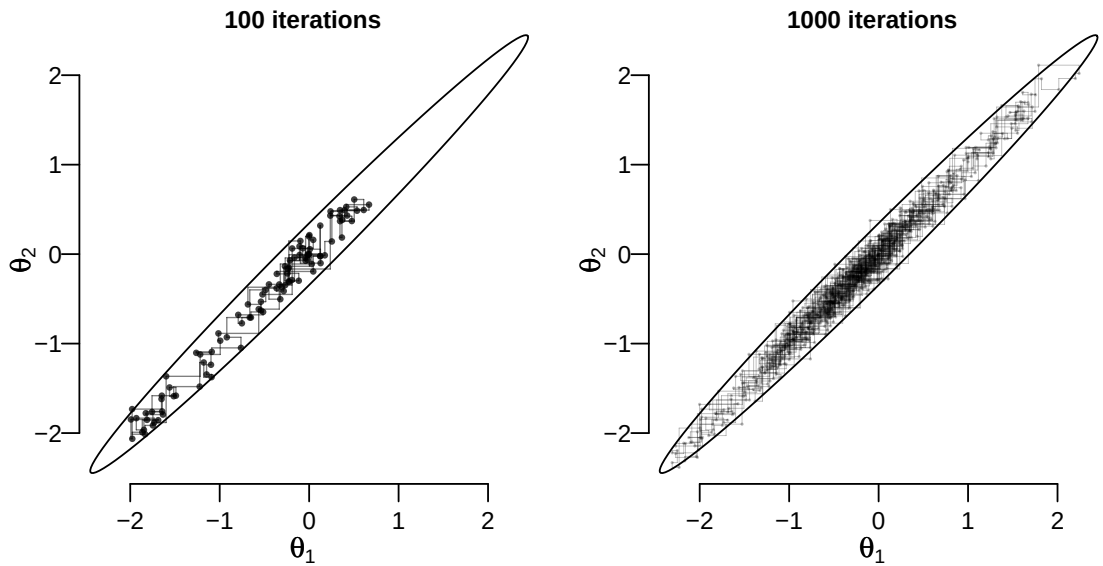
Effect of Various Thinning Intervals on Inferences for $\theta \sim N(0, 1)$. One million samples are drawn from the $N(0, 1)$ density for θ, but (by design) are highly autocorrelated ($\rho = .98$). Entries in the column labelled AR(1)

Blocking §6.4.2

- The Gibbs sampler will take a long time to explore a posterior density with a high degree of covariance between "blocks" of $\boldsymbol{\theta}$
- Example 6.2:

$$\boldsymbol{\theta} = \begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix} \sim N(\boldsymbol{\mu}, \boldsymbol{\Sigma}), \; \boldsymbol{\mu} = \begin{bmatrix} \mu_1 \\ \mu_2 \end{bmatrix}, \; \boldsymbol{\Sigma} = \begin{bmatrix} \sigma_{11} & \sigma_{12} \\ \sigma_{12} & \sigma_{22} \end{bmatrix},$$

- Gibbs sampler: $\theta_1^{(t)} \; g_1(\theta_1 | \theta_2^{(t\text{-}1)})$; $\theta_2^{(t)} \; g_2(\theta_2 | \theta_1^{(t)})$.



Blocking §6.4.2

- sample from joint distributions whenever possible
- Example 6.3: a regression model in JAGS, β is of length 2.

Compare priors for `beta` written as:

JAGS code

```
        ## priors
        beta[1] ~ dnorm(0,.0001)
        beta[2] ~ dnorm(0,.0001)
```

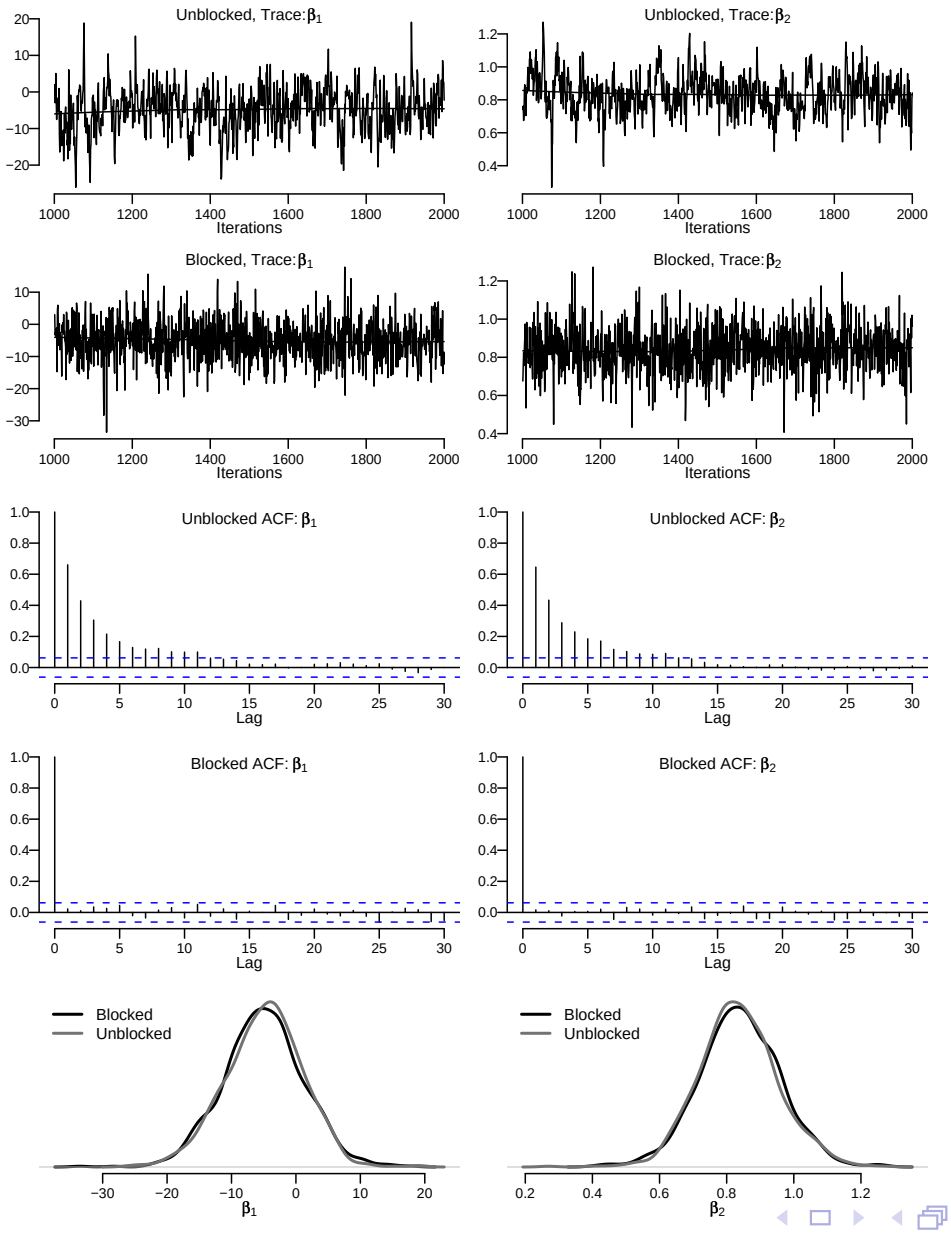
versus "blocked version" via `dmnorm`

JAGS code

```
model{
  for(i in 1:n){   ## regression model for y
    mu[i] <- beta[1] + beta[2]*x[i]
    y[i] ~ dnorm(mu[i],tau)
  }
  ## priors
  beta[1:2] ~ dmnorm(b0[1:2],B0[1:2,1:2])
  sigma ~ dunif(0,100)
  tau <- pow(sigma,-2)
}
```

i.e., force JAGS to treat `beta` *en bloc*

Example 6.3; Figure 6.3



References

Gelman, Andrew and Donald B. Rubin. 1992. "Inference from Iterative Simulation Using Multiple Sequences." *Statistical Sciences* 7:457--511.

Geweke, J. 1992. "Evaluating the accuracy of sampling-based approaches to the calculation of posterior moments (with discussion)." In *Bayesian Statistics 4*, ed. J. M. Bernardo, J. O. Berger, A. P. Dawid and A. F. M. Smith. Oxford: Oxford University Press pp. 169--193.

Plummer, Martyn. 2010a. *JAGS Version 2.00 manual*.
URL: *http://mcmc-jags.sourceforge.net*

Plummer, Martyn. 2010b. *rjags: Bayesian graphical models using MCMC*. R package version 2.1.0-2.
URL: *http://mcmc-jags.sourceforge.net*

Plummer, Martyn, Nicky Best, Kate Cowles and Karen Vines. 2008. *coda: Output analysis and diagnostics for MCMC*. R package version 0.13-2.

Raftery, Adrian E. and Steven M. Lewis. 1996. "Implementing MCMC." In *Markov Chain Monte Carlo in Practice*, ed. W. R. Gilks, S. Richardson and D. J. Spiegelhalter. London: Chapman and Hall pp. 115--130.

Spiegelhalter, David J., Andrew Thomas and Nicky Best. 2000. *WinBUGS Version 1.3*. Cambridge, UK: MRC Biostatistics Unit.