Chapter 9.7a Using JAGS

Jim Albert and Monika Hu

Chapter 9 Simulation by Markov Chain Monte Carlo

Introduction

- There has been an effort to develop general-purpose Bayesian computing software.
- One of the earliest Bayesian simulation-based computing software was BUGS (for Bayesian inference Using Gibbs Sampling) and we illustrate a similar package JAGS (for Just Another Gibbs Sampler).
- In using JAGS, one defines a Bayesian model by writing a short script. One then inputs this script together with data and prior parameter values in a single R function from the `runjags` package. This function simulates from the MCMC algorithm for a specified number of samples.

Normal sampling model

- Consider the problem of estimating the mean Buffalo snowfall assuming Normal sampling with both the mean and standard deviation unknown and independent priors.
- Express the parameters of the Normal distribution as μ and the precisionϕ.
- Write this Bayesian model as
- Sampling, for $i = 1, \cdots, n$:

$$Y_i \overset{i.i.d.}{\sim} \mathrm{Normal}(\mu, \sqrt{1/\phi}).$$

- Independent priors for μ and ϕ:

$$\mu \sim \mathrm{Normal}(\mu_0, \sqrt{1/\phi_0}), \phi \sim \mathrm{Gamma}(a, b).$$

Describe the model by a script

- Write the following script defining this model.

```
modelString = "
model{
## sampling
for (i in 1:N) {
   y[i] ~ dnorm(mu, phi)
}
## priors
mu ~ dnorm(mu0, phi0)
phi ~ dgamma(a, b)
sigma <- sqrt(pow(phi, -1))
}
```

Comments

- Note that this script resembles the statement of the model.
- In the sampling part, the loop structure `for (i in 1:N)` is used to assign the distribution of each value in the data vector `y` the same Normal distribution, represented by `dnorm`.
- In the prior part , use `dnorm` and `dgamma` to specify the Normal prior and Gamma prior for `mu` and `phi`.

Define the data and prior parameters

- In the script below, a list `the_data` is used to collect the observations `y`, the number of observations `N`, and values of the Normal prior parameters `mu0`, `phi0`, and of the Gamma prior parameters `a` and `b`.

```
buffalo <- read.csv("../data/buffalo_snowfall.csv")
data <- buffalo[59:78, c("SEASON", "JAN")]
y <- data$JAN
N <- length(y)
the_data <- list("y" = y, "N" = N,
                 "mu0"=10, "phi0"=1/3^2,
                 "a"=1,"b"=1)
```

Define initial values

- One supplies initial values in the MCMC simulation for all of the parameters.
- Alternatively, one can specify the initial values by means of a function – this will be implemented when multiple chains are discussed.
- If no initial values are specified, then JAGS will select initial values – these are usually a "typical" value such as a mean or median from the prior distribution.

Generate samples from the posterior

- The `run.jags()` from the `runjags` package does the sampling.
- The input `n.chains = 1` indicates that one stream of simulated values will be generated.
- `adapt = 1000` says that 1000 simulated iterations are used in “adapt period” to prepare for MCMC
- `burnin = 1000` indicates 5000 simulated iterations are used in a “burn-in” period
- `sample = 5000` arguments indicates that 5000 iterations of the algorithm will be collected.
- The `monitor` arguments says that we are collecting simulated values of the mean `mu` and the standard deviation `sigma`.

run.jags() function

```
posterior <- run.jags(modelString,
                      n.chains = 1,
                      data = the_data,
                      monitor = c("mu", "sigma"),
                      adapt = 1000,
                      burnin = 5000,
                      sample = 5000,
                      inits = initsfunction)
```

MCMC diagnostics and summarization

- Before summarizing the simulated sample, some graphical diagnostics methods should be implemented to judge if the sample appears to move well across the space of likely values of the parameters.
- The `plot()` function in the `runjags` package constructs a collection of four graphs for a parameter of interest.
- By running `plot()` for `mu` and `sigma`, we obtain the graphs displayed in the next slide.

```
plot(posterior, vars = "mu")
plot(posterior, vars = "sigma")
```

Diagnostic Plots for μ

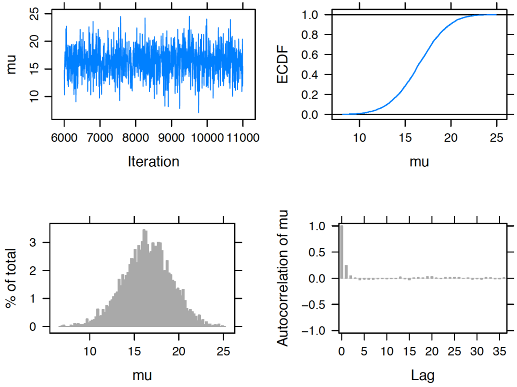

Figure 1: Diagnostic plots of simulated draws of mean using the JAGS software with the runjags package.

Diagnostic Plots for σ

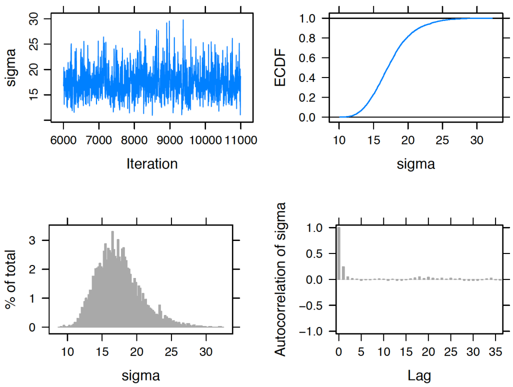

Figure 2: Diagnostic plots of simulated draws of standard deviation using the JAGS software with the runjags package.

Diagnostic plots

- Trace and autocorrelation plots are helpful for seeing how the sampler moves across the posterior distribution.
- Here the trace plots show little autocorrelation and both simulated samples of μ and σ appear to mix well.
- In the autocorrelation plots, the value of the autocorrelation drops sharply to zero as a function of the lag which confirms modest autocorrelation.

$=$ The remaining graphs are a histogram of the simulated draws and an estimate at the cumulative distribution function.

Posterior Summaries

- We are encouraged by these diagnostic graphs so we obtain summaries of the simulated samples of μ and σ
- Use `print()` function on our MCMC object.
- The posterior mean of μ is 16.5.
- The standard error of this simulation estimate is the "MCerr" value of 0.0486 – this standard error takes in account the correlated nature of these simulated draws.

```
print(posterior, digits = 3)
      Lower95 Median Upper95 Mean   SD Mode  MCerr
mu       10.8   16.5    21.4 16.5 2.68   -- 0.0486
sigma    11.8   17.1      24 17.4 3.18   -- 0.0576
```

Multiple chains

- Can try different starting values and running several MCMC chains.
- This is facilitated by arguments in the `run.jags()` function. Suppose one considers the very different pairs of starting values, $(\mu, \phi) = (2, 1/4)$ and $(\mu, \phi) = (30, 1/900)$.
- Define a value `InitialValues`, a list containing two lists, each list containing a starting value.

```
InitialValues <- list(
  list(mu = 2, phi = 1 / 4),
  list(mu = 30, phi = 1 / 900)
)
```

Multiple Chains

- The `run.jags()` function is run with two modifications
- One chooses `n.chains = 2` and the initial values are input through the `inits = InitialValues` option.

```
posterior <- run.jags(modelString,
                      n.chains = 2,
                      data = the_data,
                      monitor = c("mu", "sigma"),
                      adapt = 1000,
                      burnin = 5000,
                      sample = 5000,
                      inits = InitialValues)
```

Output

- The output variable `posterior` contains simulated draws from both chains.
- One compares posterior quantiles from the two chains. – they are close in value indicating that the MCMC run is insensitive to the choice of starting value.

```
summary(posterior$mcmc[[1]], digits = 3)
       2.5%   25%   50%   75% 97.5%
mu    10.99 14.64 16.49 18.35 21.62
sigma 12.26 15.15 17.03 19.31 25.07

summary(posterior$mcmc[[2]], digits = 3)
       2.5%   25%   50%   75% 97.5%
mu    10.97 14.59 16.55 18.33 21.54
sigma 12.21 15.08 16.96 19.18 24.99
```