Bayesian model in JAGS

Tutorial 3 for transition

Zhe Zheng (Gigi)

2023-01-16

```r
knitr::opts_chunk$set(echo = TRUE)
require(rjags)
#"Note that the rjags package does not include a copy of
# the JAGS you must install this separately.
# For instructions on downloading JAGS,
# see the home page at https://mcmc-jags.sourceforge.io."
require(mcmcplots) # to plot the trace plots
require(coda) # for gelman diagnostic
```

Outline

In this tutorial, we will learn how to use JAGS (Just Another Gibbs Sampler) in R to estimate unknown parameters with Markov Chain Monte Carlo (MCMC) methods in Bayesian framework. If you are interested in learning more about the theoretical knowledge of MCMC and Bayesian modeling, please check out courses “Probability and Statistics” (taught by Prof. Joseph Chang) and “Bayesian Statistics” (taught by Prof. Josh Warren).

In the first section, we will introduce a simple jags model to help understand the syntax for rjags package.

In the second section, we will learn how to build up complicated models step by step (censor data and spatial autocorrelation). Before reading this section, you will need to familiar yourself with the RSV_timing_explain tutorials and a research article on the relative RSV timing

Example 1: a simple jags model to estimate the unknown parameters

First: Simulate a dataset for analysis

In this example, I simulate my own data in R in order to compare the MCMC results with our input parameters. I create a continuous outcome variable y as a function of one predictor x and an error term ϵ . I simulate 100 observations. The linear regression looks like

$$y = \beta_0 + \beta_1 x + \epsilon$$

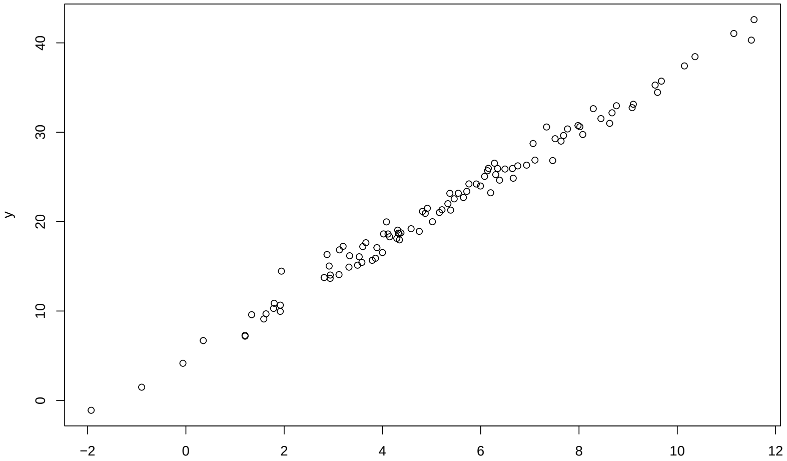
Our goal is to estimate the unknown parameters β_0 and β_1 given the observations and the linear relationship

```r
n.sim=100; set.seed(123)
x=rnorm(n.sim, mean = 5, sd = 3)
epsilon=rnorm(n.sim, mean = 0, sd = 1)
beta0=5
beta1=3.2
y=beta0 + beta1 * x + epsilon
```

Visualize the observations

Question: what is the intercept and slope for this linear regression?

```
plot(x,y,type="p")
```



rjags mamual

The following is quoted from rjags document, authored by Martyn Plummer:

“JAGS is a clone of BUGS (Bayesian analysis Using Gibbs Sampling). See Lunn et al (2009) for a history of the BUGS project. Note that the rjags package does not include a copy of the JAGS library: you must install this separately. For instructions on downloading JAGS, see the home page at https://mcmc-jags.sourceforge.io.

To fully understand how JAGS works, you need to read the JAGS User Manual. The manual explains the basics of modelling with JAGS and shows the functions and distributions available in the dialect of the BUGS language used by JAGS. It also describes the command line interface. The rjags package does not use the command line interface but provides equivalent functionality using R functions.”

rjags mamual (continued)

“Analysis using the rjags package proceeds in steps: 1. Define the model using the BUGS language in a separate file.

2. Read in the model file using the jags.model function. This creates an object of class “jags”.

3. Update the model using the update method for “jags” objects. This constitutes a ‘burn-in’ period.

4. Extract samples from the model object using the coda.samples function. This creates an object of class “mcmc.list” which can be used to summarize the posterior distribution. The coda package also provides convergence diagnostics to check that the output is valid for analysis (see Plummer et al 2006).”

Jags model structure (Step 1)

```
basic_mod <-  "model{
 #model
  for(i in 1:n.sim){
   y[i] ~ dnorm(mu[i], tau)
   mu[i] = beta0 + beta1 * x[i]
  }

 #priors
 beta0 ~ dnorm(0, 0.01)
 beta1 ~ dnorm(0, 0.01)
 tau ~ dgamma(0.01,0.01)
 }
 "
```

Initialize model with known data and initial guess of unknown parameters (Step 2)

```
######## Define known data ###########
datalist=list("y"=y,"x"=x,"n.sim"=n.sim)

######## Set initial values for parameter estimate ########
inits=function(){list("beta0"=rnorm(1), "beta1"=rnorm(1))}
```

Read in the model file using the jags.model function (Step 3)

```
### Read in the model file using the jags.model function ##
jags.basic.mod = jags.model(textConnection(basic_mod),
                            data = datalist,
                            inits = inits,
                            n.chains = 4)
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 100
##    Unobserved stochastic nodes: 3
##    Total graph size: 406
##
## Initializing model
```

Discard burn-in period and sample for posterior distribution (Step 4)

```
############ burn-in period ############
update(jags.basic.mod,5000) # n.burnin = 1000

##### sample for posterior distribution #######
model_sample <- coda.samples(jags.basic.mod,
                             c("beta0","beta1"),
                             100000,thin=5)
#You can also use jags.samples(). The results will be the same
```

Summarize the posterior distribution

```
summary(model_sample)
```

```
##
## Iterations = 5005:105000
## Thinning interval = 5
## Number of chains = 4
## Sample size per chain = 20000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##        Mean      SD  Naive SE Time-series SE
## beta0 4.982 0.21473 0.0007592      0.0010566
## beta1 3.183 0.03621 0.0001280      0.0001764
##
## 2. Quantiles for each variable:
##
##        2.5%   25%   50%   75% 97.5%
## beta0 4.558 4.839 4.982 5.126 5.404
## beta1 3.111 3.158 3.183 3.207 3.254
```

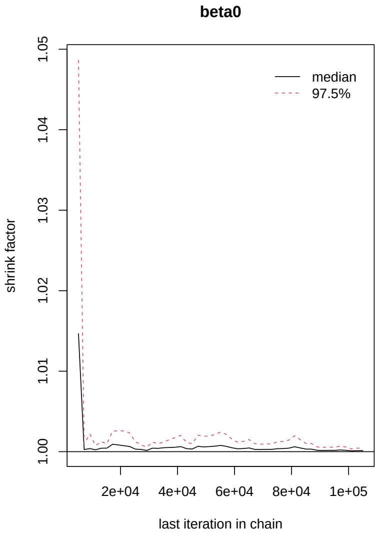
Check for model convergence (gelman diagnotics)

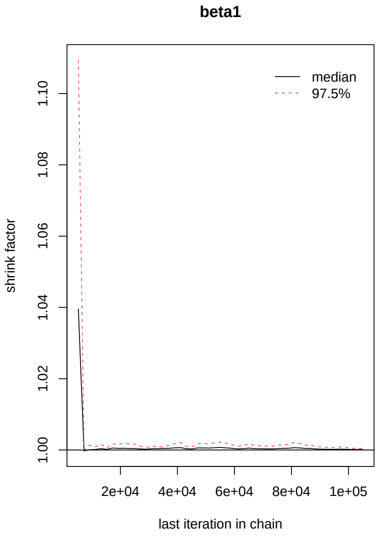
```
gelman.diag(model_sample) ## <=1.1 means well converge
```

```
## Potential scale reduction factors:
##
##       Point est. Upper C.I.
## beta0          1          1
## beta1          1          1
##
## Multivariate psrf
##
## 1
```

Check for model convergence (gelman plot)

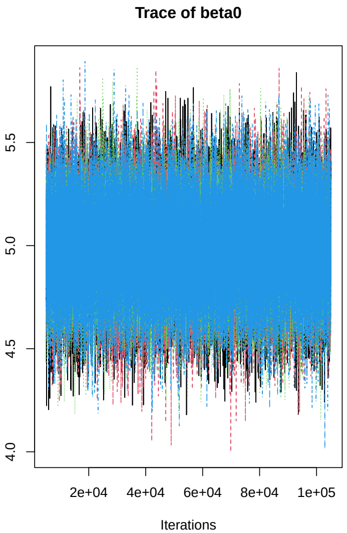
```
gelman.plot(model_sample)
```

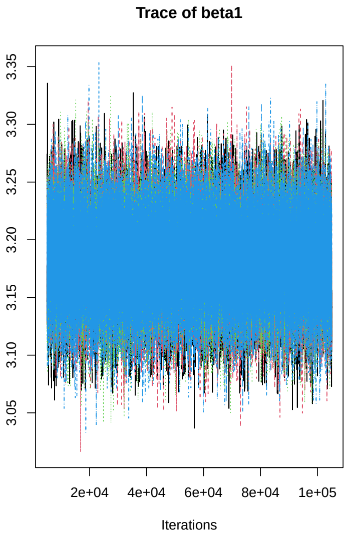




Plot the trace plot

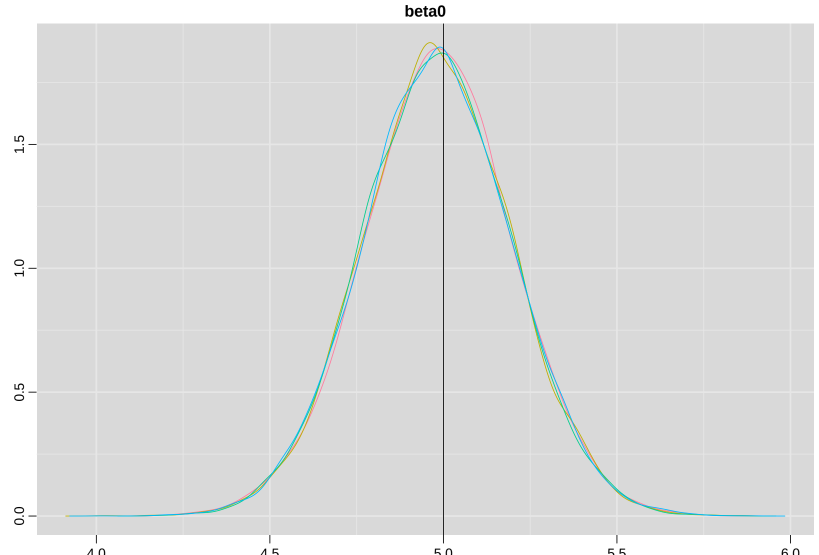
```
par(mfrow=c(1,2))
traceplot(model_sample)
```





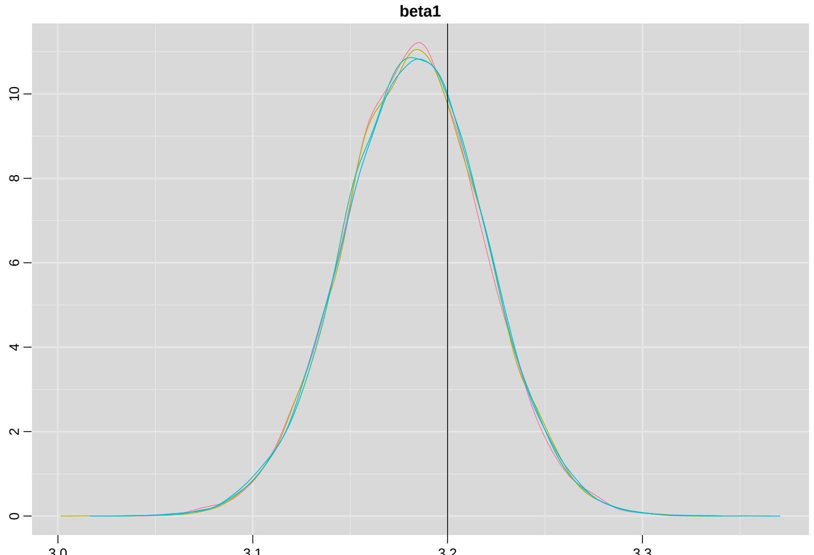
Compare the posterior density and real value for beta0

```
denplot(model_sample, parms = c("beta0"))
abline(v=5)  # True value
```



Compare the posterior density and real value for beta1

```
denplot(model_sample, parms = c("beta1"))
abline(v=3.2) # True value
```


beta1