MCMC using JAGS

Applied Bayesian Statistics
Winter term 2018

Susumu Shikano
GSDS

Introduction

This session

- Using JAGS to set up MCMC
- . . . with beta-binomial and bivariate regression models

JAGS

Just Another Gibbs Sampler

- The name is a bit confusing since it utilizes not only Gibbs sampling, but also MH, slice sampling etc.
- Free available under:
  `http://mcmc-jags.sourceforge.net/`
- Or: simply google with “JAGS”
  - Download the latest version and install it.
  - currently
    `Download JAGS-4.3.0.exe (about 30 MB)`
- You can use `rjags` to use JAGS from R.

JAGS

Why JAGS?

- Easy, intuitive and flexible in programming.
- Main Code = Likelihood + Prior
- One of the most popular tools (at least in Social Sciences).

JAGS

Why JAGS?

- Easy, intuitive and flexible in programming.
- Main Code = Likelihood + Prior
- One of the most popular tools (at least in Social Sciences).

Some drawbacks

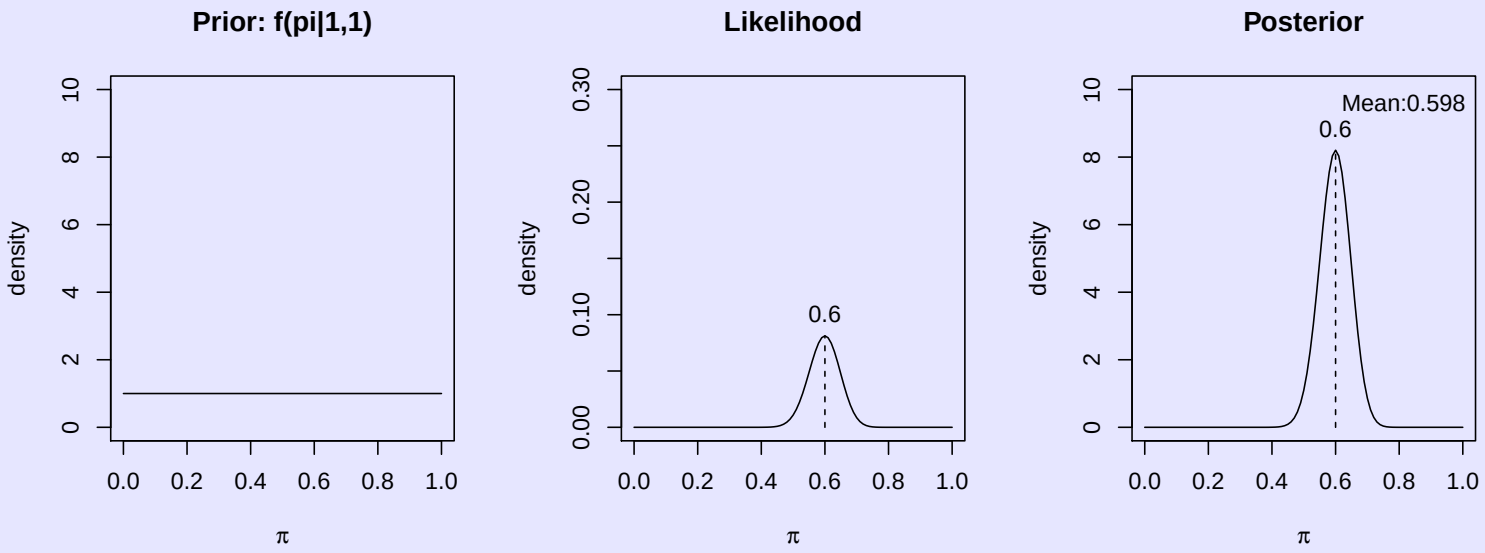
- Slow if the posteriors are correlated.
- Alternative: `stan` via `rstan` using the hybrid Monte Carlo algorithm (or Hamilton MC).
- . . . Installation a bit tricky (see the preparation document of this course).

Markov-Chain-Monte-Carlo (MCMC) using JAGS

- Installation of JAGS
- Basic procedure
  - Specifying a model
  - Reading data
  - Giving initial values
  - Running MCMC
  - Describing the posterior
- Example 1: beta binomial model
- Example 2: bivariate regression model

Beta-Binomial Model

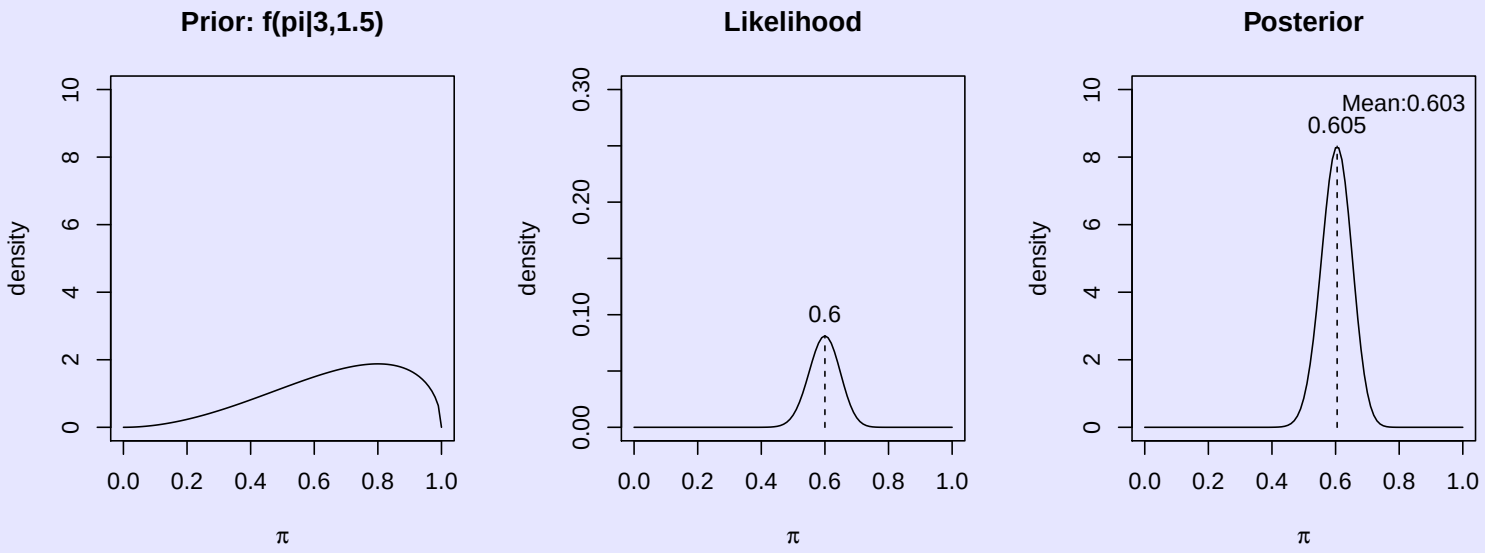
Hopefully you still remember...



- The more certain $p(\theta)$ is,

Beta-Binomial Model

Hopefully you still remember...

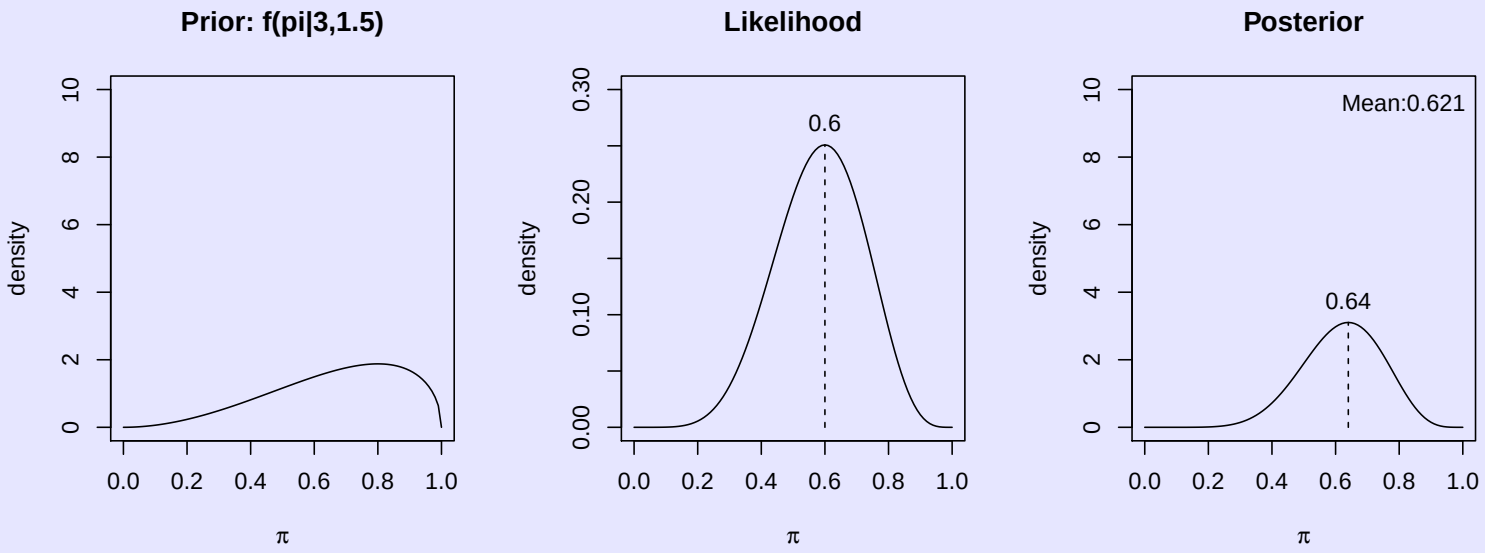


- The more certain $p(\theta)$ is,

...the more the prior distribution influences the posterior.
$\rightarrow$ A more different result from ML.

Beta-Binomial Model

Hopefully you still remember...



- The more certain $p(\theta)$ is,
- The smaller n is,*

...the more the prior distribution influences the posterior.
$\rightarrow$ A more different result from ML.

* Here, 6 out of 10 are correct judgement.

Beta-Binomial Model

JAGS-Code

```
library(rjags)

# JAGS Modell
beta.binom.model <- "model{
   y ~ dbin(p,N)   # Likelihood
   p ~ dbeta(1,1) # Prior
}"

write(beta.binom.model, "Bayes_Beta_Binom.bug")

# Data
jags.data <- list(y = 60,N=100)
```

Beta-Binomial Model

JAGS-Code

```
# Running JAGS
jags.binom <- jags.model(file="Bayes_Beta_Binom.bug",
          data=jags.data, n.chains=3)

update(jags.binom, 1000)
parameters <- c("p")
jags.out <- coda.samples(jags.binom,
                variable.name= parameters,
                n.iter=1000, thin=1)
```

Beta-Binomial Model

JAGS-Code

```
# Simple description of posterior
summary(jags.out)
plot(jags.out)

# Which percentage of posterior p>0.5 ?
p <- unlist(jags.out)

table(p>0.5)

hist(p)
plot(density(p))
```

How to set up a model

An example: Regression model

$$(\mathbf{y}|\beta_0, \beta_1, \sigma^2, X) \quad \sim \quad N(\beta_0 + \beta_1 X, \sigma^2) \tag{1}$$

. . . is equivalent to. . .

$$\begin{aligned} y_i &\sim N(\mu_i, \sigma^2) \\ \mu_i &= \beta_0 + x_i \beta_1 \\ &\quad \text{for all } i = 1, \ldots, n \end{aligned}$$

How to set up a model

An example: Regression model

$$(\mathbf{y}|\beta_0, \beta_1, \sigma^2, X) \quad \sim \quad N(\beta_0 + \beta_1 X, \sigma^2) \tag{1}$$

. . . is equivalent to. . .

$$\begin{aligned} y_i &\sim N(\mu_i, \sigma^2) \\ \mu_i &= \beta_0 + x_i\beta_1 \\ &\quad \text{for all } i = 1, \ldots, n \\ \beta_0 &\sim \text{prior distribution} \\ \beta_1 &\sim \text{prior distribution} \\ \sigma^2 &\sim \text{prior distribution} \end{aligned}$$

How to set up a model

JAGS-Code

```
for (i in 1:N){
    y[i]} ~ dnorm(mu[i],tau)
    mu[i] <- beta0 + beta1 * x[i]
}

beta0 ~ dnorm(0,0.0001)
beta1 ~ dnorm(0,0.0001)
tau ~ dgamma(0.001,0.001)
sigma <- 1/sqrt(tau)
```

How to set up a model

JAGS-Code

```
for (i in 1:N){
    y[i]} ~ dnorm(mu[i],tau)
    mu[i] <- beta0 + beta1 * x[i]
}

beta0 ~ dnorm(0,0.0001)
beta1 ~ dnorm(0,0.0001)
tau ~ dgamma(0.001,0.001)
sigma <- 1/sqrt(tau)
```

2nd parameter of normal distribution

How to set up a model

JAGS-Code

```
for (i in 1:N){
    y[i]} ~ dnorm(mu[i],tau)
    mu[i] <- beta0 + beta1 * x[i]
}

beta0 ~ dnorm(0,0.0001)
beta1 ~ dnorm(0,0.0001)
tau ~ dgamma(0.001,0.001)
sigma <- 1/sqrt(tau)
```

2nd parameter of normal distribution

- Precision
- = Inverse of variance

Example data

`Bayes_Student_Survey.RData`

A reduced dataset of Student Survey during the Lecture in Introduction to Political Methodology Winter term 2016/2017

- poleff: Political Efficacy (Likert Score based on 7 items)
  - A larger value $\rightarrow$ A higher level of efficacy
- friend: Number of alteri in friendship network
- poldisc: Number of alteri in political discussion network
- lr.self: Ideological orientation (left right self-placement)
  - 1: Left <- -> 11: Right
- lr.self.2: Ideological orientation (left right self-placement, 2nd measurement with the same scale above)
- univ.election: Vote intention at the next university election
  - 1: Yes; 0: other (No and DK)
- polint: interest at university politics
  - 1: not interested at all <- -> 5 strongly interested

Example data

`Bayes_Student_Survey.RData`

tuition: opinion on the general tuition fee for German universities

- 1: support; 2: reject; 3: indifferent

acceptable: acceptable level of the tuition fee (in Euro per Semester)

- (Only those who support the tuition fee or indifferent)

protest1 - protest6: willingness to participate a protest action against the general tuition fee

- 1: yes; 0: no

1. demonstration in Konstanz
2. demonstration in Stuttgart
3. giving signature at petitions
4. strike
5. occupation of university buildings
6. legal dispute at courts

Basic Procedure

Step by Step

1. Specify and save your model
2. Read your data
3. Select/generate initial values for unknown parameters
4. Compile your model with data
5. Determine the parameter of interest
6. Run Gibbs Sampling
7. Observe posterior distribution to check the convergence
8. Repeat Step 6-7 if needed
9. Describe posterior and save the results

Basic Procedure in rjags: Step by step

Specify your model

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

  beta0 ~ dnorm(0,0.0001)
  beta1 ~ dnorm(0,0.0001)

  tau ~ dgamma(0.001,0.001)
  sigma <- 1/sqrt(tau)
}"

write(reg.model,
      "Bayes_Bivariate_Reg_Student_Survey.bug")
```

Basic Procedure in rjags: Step by step

Prepare the Sampler

```
y <- dat$poleff
x <- log(dat$friend+1)
N <- length(y)
jags.data <- list(y=y,x=x,N=N)

jags.inits <- function(){list(sigma=runif(1,0,100))}

jags.reg <- jags.model(file=
     "Bayes_Bivariate_Reg_Student_Survey.bug")
                        inits=jags.inits,
                        data=jags.data, n.chains=3)
```

Basic Procedure in rjags: Step by step

Run Gibbs Sampler

```
# setting the parameters
parameters <- c("beta0","beta1","sigma")

# running Gibbs sampler (for burin-in)
update(jags.reg, 2000)

# random draw from the posterior
jags.reg.out <- coda.samples(jags.reg,
                             variable.names=parameters,
                             n.iter=2000, thin=1)
```

Basic Procedure in rjags: Step by step

Describe the result

```
# Summary Statistics
summary(jags.reg.out)

# plotting posterior
plot(jags.reg.out)

# Gelman-Rubin-Statstics
gelman.plot(jags.reg.out)

# Deviance Information Criterion
jags.reg.dic.out <- dic.samples(jags.reg,
                                variable.names=parameters,
                                n.iter=2000, thin=1)
jags.reg.dic.out
```

Obtaining information

Two possibilities in `rjags` to sample from the posterior

```
jags.post <- coda.samples(jags.mcmc,
               variable.names=c("beta0","beta1","sigma"),
               n.iter=5000, thin=1)

jags.post <- jags.samples(jags.mcmc,
               variable.names=c("beta0","beta1","sigma"),
               n.iter=5000, thin=1)
```

They are same in sampling, but differ in the output form.

- `coda.samples`: `mcmc.list`-object
- `jags.samples`: `mcarray`-object

Below, only `coda.sample` will be discussed.

Obtaining information

Structure of the output (`mcmc.list`-object)

- It is a list of individual chains.
- It is quite easy to get a summary statistics:
  - `summary(jags.post)`

Obtaining information

To extract a single chain

- e.g. 1st chain: `jags.post[[1]]`
- . . . A matrix (row: iterations; column: parameters)

```
          beta0     beta1    sigma
[1,] 139.8247 -24.99645 15.70429
[2,] 142.4995 -26.94930 17.77505
[3,] 139.0966 -26.15395 18.35460
[4,] 138.6545 -24.32788 17.88755
[5,] 137.4641 -24.57537 15.34373
     ...
```

Obtaining information

To obtain information of a specific parameter

- Obtained posterior by e.g. `jags.post[,"beta1"]` is a list:

```
[[1]]
Markov Chain Monte Carlo (MCMC) output:
Start = 4001
End = 5000
Thinning interval = 1
   [1] -0.858293314  0.040098476 -0.698333929 -0.829177717
             ...
[[2]]
Markov Chain Monte Carlo (MCMC) output:
Start = 4001
End = 5000
Thinning interval = 1
   [1]  0.432260841  0.149573781  0.238826681  0.014094765
             ...
```

- The list can be transformed into vector by using e.g. `unlist(jags.post[,"beta1"])`

Obtaining information

To obtain information of multiple parameters

- E.g. `beta0` and `beta1`
- `jags.post[,c("beta0","beta1")]` can also work.
- The list can be transformed into a matrix:
  - by using e.g. `as.matrix(jags.post[,"beta0","beta1"])`.
  - Do not use here `unlist`. The matrix structure would be lost.

Obtaining information

Sometimes working with `mcmc.list`-object is annoying

- A better way: transform the object into a matrix:
  - `post.mat <- as.matrix(jags.post)`
  - with parameters in columns
  - with iterations in rows
- Some advantages:
  - Different calculation is easier.
  - Easily extract parameters with similar names:
  - `posterior.mat[,grep("beta",colnames(post.mat))]`
  - Extract all variables with a name including `beta`
  - `grep` returns which elements of an object include a character pattern is included.

Storing posterior

Some possibilities

- `save(jags.post, file=c:xxx.RData")`
  - By using `load(file=c:xxx.RData")` you can restore the same object.
  - Only for `R`
- `write.dta(xxx)` `write.csv(xxx)`.
  - In the formats readable in the other softwares.
  - You have to first convert the `mcmc.list`-object into a data frame.

Storing posterior

You have to care:

- Sometimes your posterior information requires a large disk space.
- You do not have to save all iterations.
  - The number of iterations and the MC-standard error is not in a linear relationship.
- You can thin the posterior information post-hoc.

```
post.mat <- as.matrix(jags.post)
thinned <- seq(1,nrow(post.mat),by=10)
post.mat.thinned <- post.mat[thinned,]
```

After all...

Cleaning up trash

- `rjags` produces `bug`-files in your working directory.
- `bug`-files: files for the JAGS-model
- They can be cleaned up by the following commands.

```
all.files <- dir()
             # list up the name of all files in WD
bug.files <- all.files[grep(".bug", all.files)]
             # choose only the file names with ".bug"
file.remove(bug.files)
             # delete the specified files
```