Introduction to JAGS

Bayesian Psychometric Models, Lecture 5

JAGS stands for Just Another Gibbs Sampler and is a program that can build a large number of Bayesian models.

In order to following along during this lecture, please download and install JAGS from https://sourceforge.net/projects/mcmc-jags/. This lecture is based on version 4.3.0 of JAGS.

The JAGS user manual, available from https://sourceforge.net/projects/mcmc-jags/files/Manuals/4.x/jags_user_manual.pdf/download, is a good to read as it provides many details on how to use JAGS.

We will initially use the `rjags` package, which is described in detail in the JAGS user manual. You can install and/or load it from the following syntax chunk:

```r
if (!require(rjags)) install.packages("rjags")
```

```
## Loading required package: rjags
```

```
## Loading required package: coda
```

```
## Linked to JAGS 4.3.0
```

```
## Loaded modules: basemod,bugs
```

```r
library(rjags)
```

Additionally, we will use the `R2jags` package which can be installed using the following chunk of syntax. The R2jags package does a lot of the labor intensive work for us, making it much easier (and faster) to run JAGS.

```r
if (!require(R2jags)) install.packages("R2jags")
```

```
## Loading required package: R2jags
```

```
##
## Attaching package: 'R2jags'
```

```
## The following object is masked from 'package:coda':
##
##     traceplot
```

```r
library(R2jags)
```

Finally, we will use the `coda` package to examine the JAGS model output for convergence and to summarize the posterior distribution. You can install and load coda from the chunk below:

```r
if (!require(coda)) install.packages("coda")
library(coda)
```

To demonstrate JAGS in our introduction, we will use the syntax provided on p. 39 of Levy and Mislevy (2016) that estimates the probability parameter θ for a 7-success in 10-trail set of data, using a binomial distribution. The text file `model01.jags`, which is in the current directory of the repo, has a shortened version of this syntax. Note, you can avoid having to go outside of RStudio by enclosing the model in quotes and saving it as a variable. To pass it to the `jags.model()` function, put the name of the variable in the `textConnection` function.

To run JAGS several steps must be undertaken: 1. The "data" (i.e., everything JAGS depends on and won't sample) must be listed 2. The model must be compiled 3. The model must be "adapted" (tuned) 4. The samples must be drawn from the posterior distribution

```
# parts of data needing to be passed to JAGS:
J = 10
y = 7
alpha = 1
beta = 1

model01.text = "

model{
  # PRIOR DISTRIBUTION #####################################################
  theta ~ dbeta(alpha, beta)

  # P(X|THETA) - COND. DIST. OF DATA (AKA LIKELIHOOD) #######################
  y ~ dbin(theta, J)

}
"

# compile the JAGS model and run first part of adaptation phase (you may have to change the path to the
model01 = jags.model(file = textConnection(model01.text),
                     data = list(J = J, y = y, alpha = alpha, beta = beta),
                     inits = list(theta=runif(1)),
                     n.chains = 5,
                     n.adapt = 1000)
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 1
##    Unobserved stochastic nodes: 1
##    Total graph size: 5
##
## Initializing model
```

```
# not needed but included for didactic purposes (shows the algorithm that will be used)
list.samplers(model01)
```

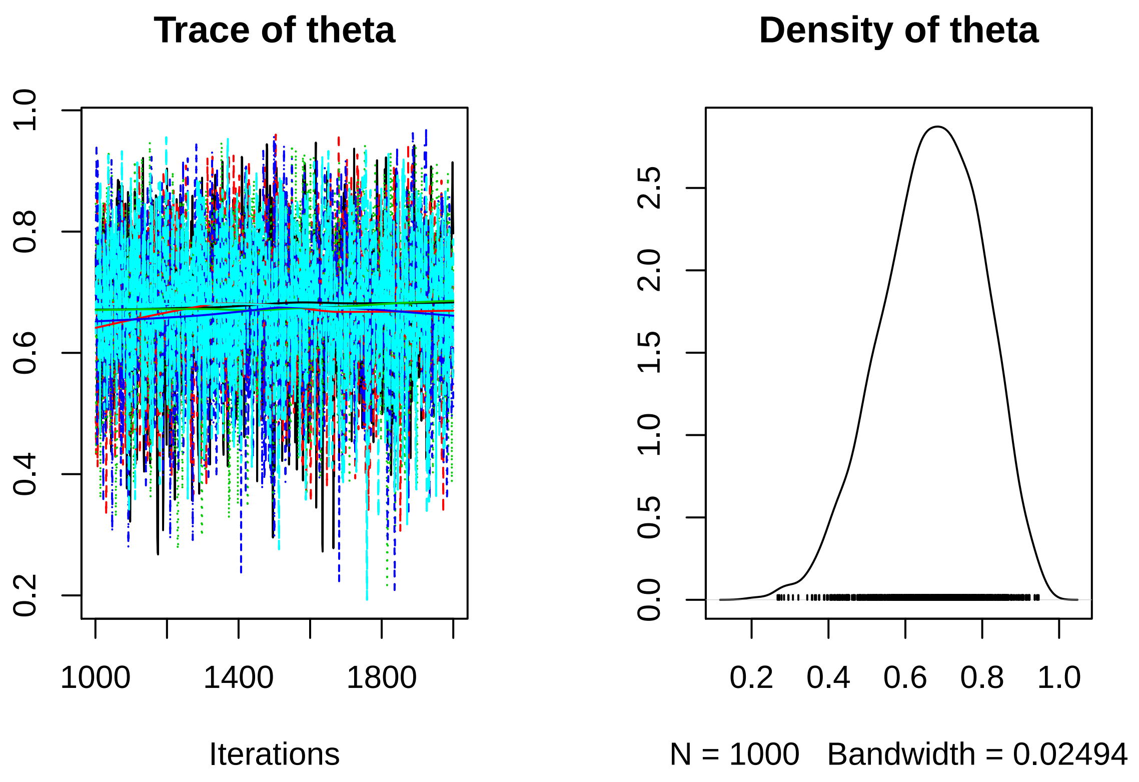
```
## $`bugs::BinomSlicer`
## [1] "theta"
```

```
# draw samples from the posterior of the model
model01.samples = coda.samples(model = model01,
                               variable.names = "theta",
                               n.iter = 1000)

# samples are drawn as a coda mcmc.list object, so you can use some generic function with them:
summary(model01.samples)
```

```
##
## Iterations = 1001:2000
## Thinning interval = 1
## Number of chains = 5
## Sample size per chain = 1000
##
```

```
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##           Mean             SD       Naive SE Time-series SE
##       0.668803       0.129235       0.001828       0.002309
##
## 2. Quantiles for each variable:
##
##   2.5%    25%    50%    75%  97.5%
## 0.4041 0.5826 0.6765 0.7656 0.8931
```

```
# plotting the chain:
plot(model01.samples)
```



```
# looking at convergence (discussed in a later class)
gelman.diag(model01.samples)
```

```
## Potential scale reduction factors:
##
##       Point est. Upper C.I.
## theta          1          1
```

Next, `R2jags` can be used to run the same model. Here, the model is put into an R function (without the "model{}" open/closing brackets). This method

```
# parts of data needing to be passed to JAGS:
J = 10
y = 7
alpha = 1
beta = 1
```

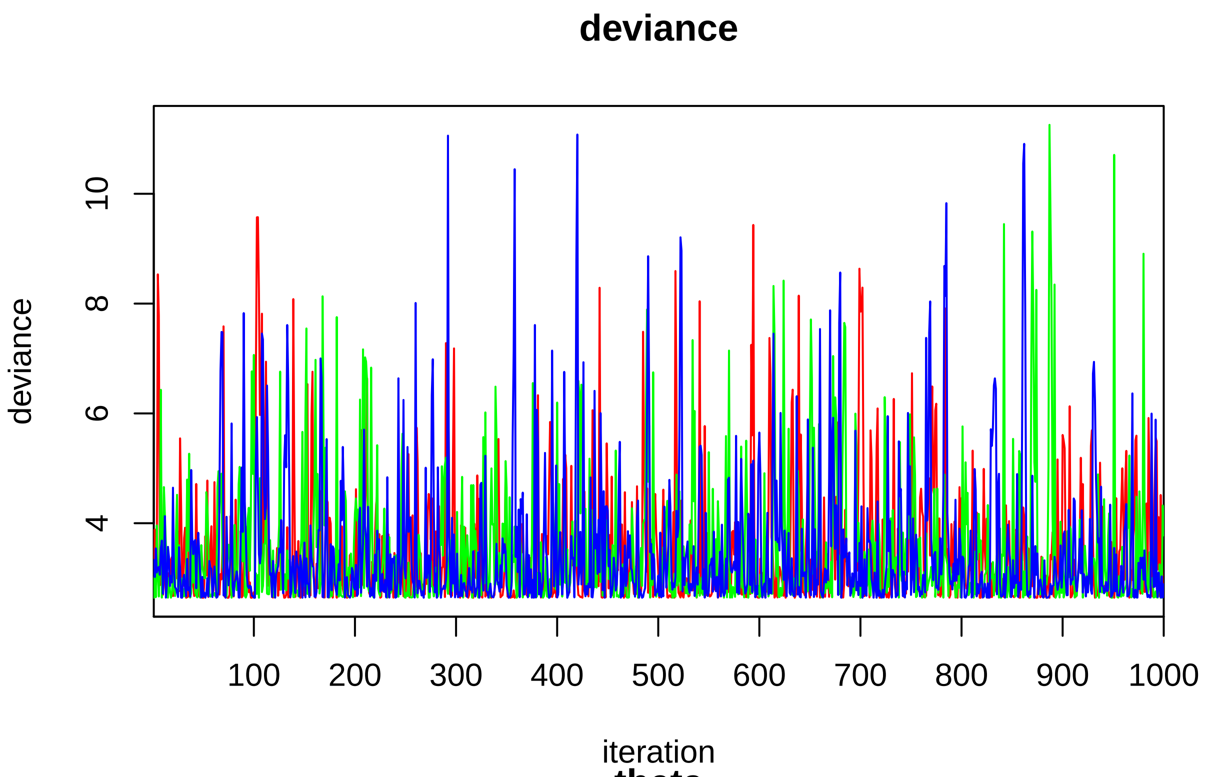
```r
binomialModel01 = function(){

  # PRIOR DISTRIBUTION #######################################################
  theta ~ dbeta(alpha, beta)

  # P(X|THETA) - COND. DIST. OF DATA (AKA LIKELIHOOD) #######################
  y ~ dbin(theta, J)

}

# initial values of parameters:
jags.inits <- function(){
    list("theta"=runif(1))
}


model01.R2jags = jags(data = list(J = J, y = y, alpha = alpha, beta = beta),
                      inits = jags.inits,
                      parameters.to.save = "theta",
                      model.file = binomialModel01,
                      working.directory = getwd())
```

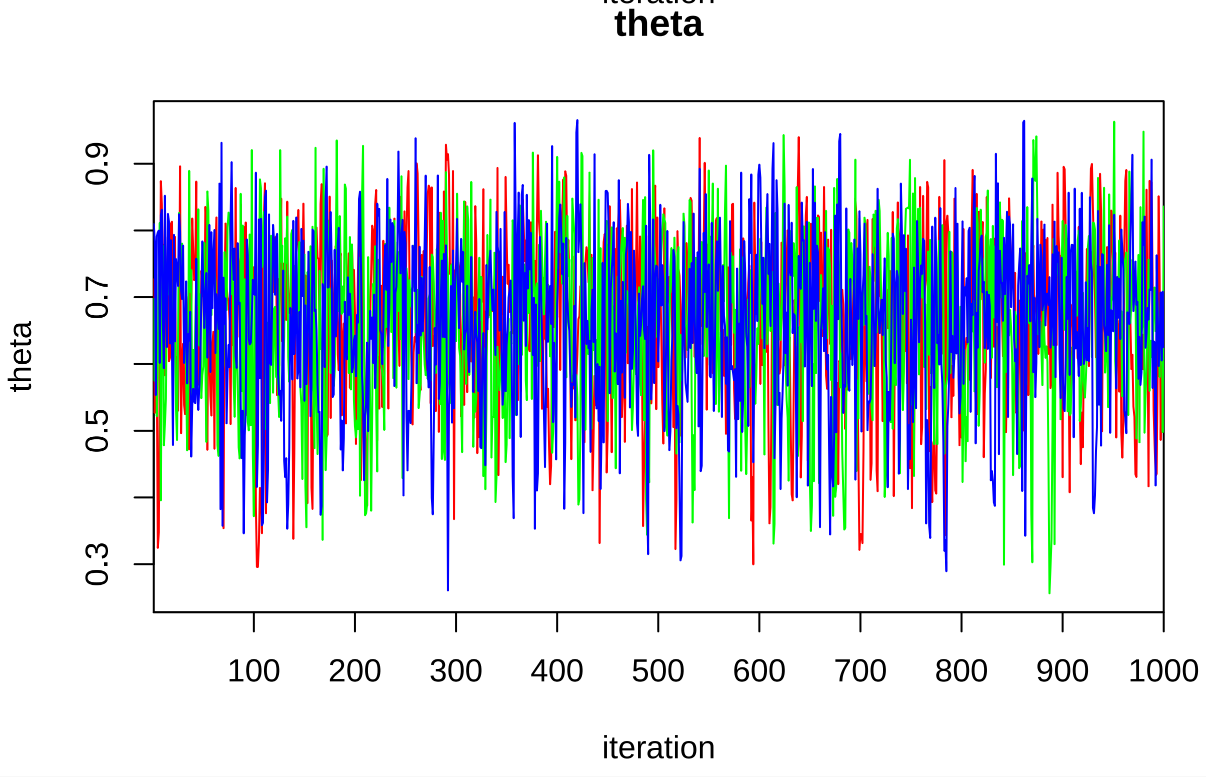
```
## module glm loaded

## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 1
##    Unobserved stochastic nodes: 1
##    Total graph size: 5
##
## Initializing model
```

```r
# now to see the output use the print function
print(model01.R2jags)
```

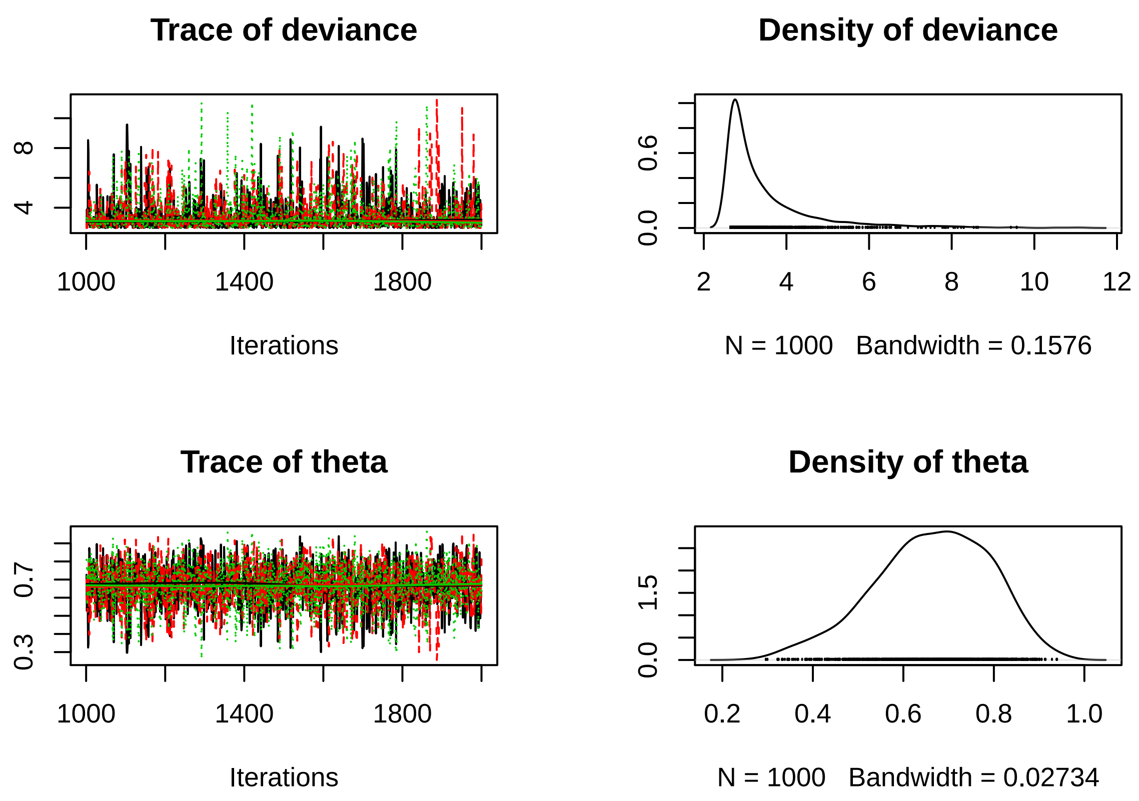
```
## Inference for Bugs model at "/var/folders/sl/n1qxb2m57pqfs1hcfqb8p4nx6fm6zn/T//Rtmp8o8gd3/modelbbdc2
##  3 chains, each with 2000 iterations (first 1000 discarded)
##  n.sims = 3000 iterations saved
##          mu.vect sd.vect  2.5%   25%   50%   75% 97.5%  Rhat n.eff
## theta      0.663   0.128 0.387 0.580 0.672 0.760 0.883 1.001  3000
## deviance   3.486   1.189 2.643 2.735 3.035 3.723 7.145 1.001  3000
##
## For each parameter, n.eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor (at convergence, Rhat=1).
##
## DIC info (using the rule, pD = var(deviance)/2)
## pD = 0.7 and DIC = 4.2
## DIC is an estimate of expected predictive error (lower deviance is better).
```

```r
# the plot function changes to traceplot
traceplot(model01.R2jags)
```





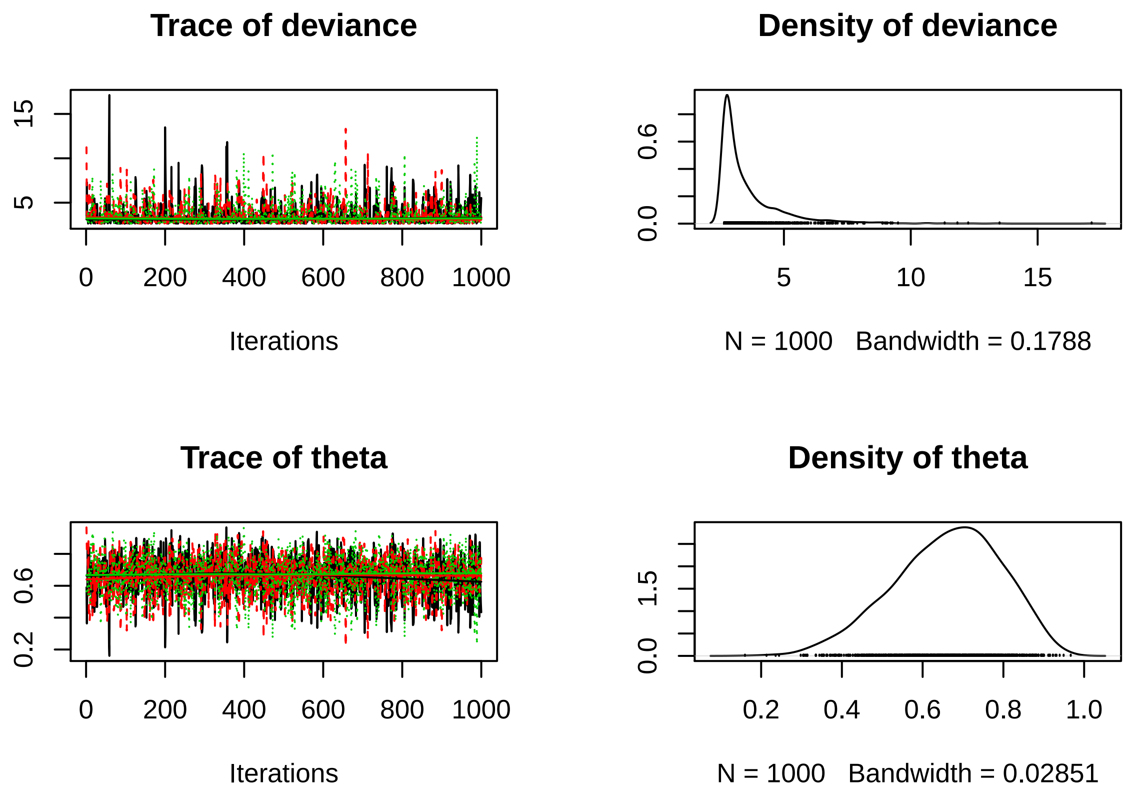
```r
# or you can use the coda plotting and summary functions
plot(as.mcmc(model01.R2jags))
```



```r
summary(as.mcmc(model01.R2jags))
```

```
##
## Iterations = 1001:2000
## Thinning interval = 1
## Number of chains = 3
## Sample size per chain = 1000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##            Mean     SD Naive SE Time-series SE
## deviance 3.4864 1.1892 0.021711       0.031175
## theta    0.6631 0.1279 0.002335       0.003027
##
## 2. Quantiles for each variable:
##
##           2.5%    25%    50%   75%  97.5%
## deviance 2.643 2.7348 3.0352 3.723 7.1446
## theta    0.387 0.5797 0.6723 0.760 0.8835
```

```r
# if the chain did not converge, you can use the autojags function to run and check it automatically
model01.auto = autojags(object = model01.R2jags)



plot(as.mcmc(model01.auto))
```



```r
summary(as.mcmc(model01.auto))
```

```
##
## Iterations = 1:1000
## Thinning interval = 1
## Number of chains = 3
## Sample size per chain = 1000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##            Mean     SD Naive SE Time-series SE
## deviance 3.5631 1.3050 0.023827       0.033171
## theta    0.6605 0.1334 0.002435       0.003182
##
## 2. Quantiles for each variable:
##
##            2.5%    25%    50%    75%  97.5%
## deviance 2.6432 2.7308 3.0587 3.8518 7.3096
## theta    0.3785 0.5727 0.6713 0.7578 0.8868
```

Now you can try playing around with the syntax by picking other distributions for the prior or changing the values of the prior's parameters.