John Miyamoto

Test Run to Check the Installation of OpenBUGS, JAGS, BRugs, R2OpenBUGS, Rjags & R2jags

The following annotated code is extracted from John Kruschke's R scripts, "E:\btut\r\BernBetaBugsFull.R" and "E:\btut\r\BernBetaJagsFull.R" that were downloaded from his website (http://indiana.edu/~kruschke/DoingBayesianDataAnalysis/). I have made some changes to this code.

The purpose of this annotated R code is simply to check that your installation of OpenBUGS, JAGS, BRugs, R2OpenBUGS, Rjags & R2jags are all running and talking to each other in the way that they should. *It is assumed that all of these programs have been installed* (see, e.g., "Installing BUGS and the R to BUGS Interface"; the installing.bugs.jags.pdf is available on the Psych 548 website). The R-code for this document is shown as text at the end of this file, and also as an ascii file, test.bugs.install.txt. If you want to run the code as you are reading this document, I recommend that *you load the text file, test.bugs.install.txt, into RStudio or your favorite programming editor*. It will be easier to transfer the code to R from a programming editor than from this pdf file.

Contents (Cntrl-left click on a link to jump to the corresponding section)

1. The binomial inference problem that will be used in this document TOC

All of the computations in this document make use of the same Bayesian inference problem.

DATA	You have observed 14 independent flips of a coin which may or may not be biased. You observe 11 heads and 3 tails.
PARAMETER	The probability of a heads is an unknown parameter called **`theta`**.
PRIOR PROBABILITY DISTRIBUTION	Prior to observing the coin flips, you believe that all values of **`theta`** are equally likely, so the prior probability distribution of **`theta`** is a uniform distribution over the [0, 1] interval.
BAYESIAN INFERENCE PROBLEM	What is the posterior probability distribution over **`theta`** after observing 11 heads and 3 tails?

One reason that this inference problem is used so often to teach Bayesian statistics is that it can be solved by many different methods. Studying how these different methods apply to the same problem helps students understand some of the more esoteric methods. Figure 1 shows the analytical solution to this problem. Given a uniform prior distribution over **theta**, the posterior distribution is a beta distribution with parameters 12 and 4 (I don't know of any widely accepted names for these parameters; often they are denoted as α and β). In Figure 1, the dotted line represents the prior probability distribution and the solid curve represents the posterior probability distribution.

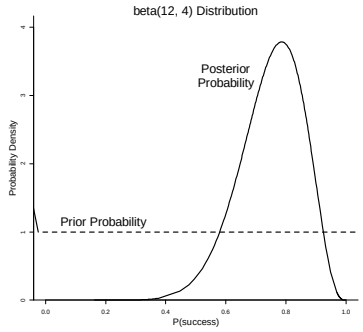

Figure 1

The remainder of this document solves this same problem using MCMC simulation software.

2. Checking the connection between R and OpenBUGS via BRugs. TOC

- Note: The **BRugs** documentation states that you may have problems running **BRugs** with the 32-bit version of R. Nevertheless, I had no problems running the following code with the 64-bit version of R.

Table 1. Test the **BRugs** interface between R and OpenBUGS. The code in this table is a subset for the code in Kruschke's script file, BernBetaBugsFull.R.

# R Code	Explanation
`library(BRugs)`	Attaches **BRugs** to the search path.

`modelString = "`
`BUGS model specification begins ...`
`model {`
`  # Likelihood:`
`  for ( i in 1:nFlips ) {`
`  y[i] ~ dbern( theta )`
`  }`
`  # Prior distribution:`
`  theta ~ dbeta( priorA , priorB )`
`  priorA <- 1`
`  priorB <- 1`
`}`
`... BUGS model specification ends.`
`" # close quote to end modelString`
	THE MODEL.
	
	Specify the model in BUGS language, but save it as a string in R:

`writeLines(modelString,con="model.txt")`	Write the modelString to a file, using R commands:
`modelCheck( "model.txt" )`	Use BRugs to send the model.txt file to BUGS, which checks the model syntax:

`dataList = list(`
`  nFlips = 14 ,`
`  y = c( 1,1,1,1,1,1,1,1,1,1,0,0,0 )`
`)`
	`# THE DATA.`
	Specify the data in R, using a list format compatible with BUGS: [This is not the only way to specify the data for BUGS. JM will show you other ways.]

`modelData( bugsData( dataList ) )`	Use **BRugs** commands to put the data into a file and ship the file to BUGS:
`modelCompile()` `modelGenInits()`	`# INTIALIZE THE CHAIN.` BRugs command tells BUGS to compile the model. BRugs command tells BUGS to randomly initialize a chain.

`samplesSet( "theta" )`	**`# RUN THE CHAINS.`** BRugs tells BUGS to keep a record of the sampled "theta" values:
`chainLength = 50000`	R command defines a new variable that specifies an arbitrary chain length:
`modelUpdate( chainLength )`	BRugs tells BUGS to generate a MCMC chain:
`thetaSample = samplesSample( "theta" )`	**`# EXAMINE THE RESULTS.`** BRugs asks BUGS for the sample values.

`mode(thetaSample)`
`length(thetaSample)`
`thetaSample[1:20]`
	`thetaSample` should exist in your R data set. It should be a vector of length 1,000 (the simulation produced 1,000 estimates of theta. If this is in fact true, then you have OpenBUGS sucessfully installed and the **`BRugs`** interface is working properly with R.

`thetaSummary = samplesStats( "theta" )` `thetaSummary`	BRugs asks BUGS for summary statistics. Again, if you can see these statistics, then the **`BRugs`** interface is working properly with R.

`res.param(thetaSample)`
 	
`mtext(`
`"Estimated Posterior Distribution of ThetaSample",`
`side = 3, cex = 2, line = 2)`
`mtext(`
`"Estimates Computed by OpenBUGS Through the BRugs Interface",`
`side = 3, cex = 1.25, line = 0)`
	The **`res.param`** function is in the **`jmfuns.rda`** file[1]. You will have to attach this file to the search path if you want to run this function. **`res.param`** makes a plot of the posterior distribution of a parameter.

`beta.exact = draw.beta(12, 4, plot.dist = FALSE)`
`x = beta.exact[ , 1] - 0.005`
`y = beta.exact[ , 2]`
 	
`lines(x, y, col = "red")`
	I have added the true beta(12, 4) distribution to this plot, except that I displaced the curve by 0.005 to the left in order to be able to see both the true distribution and the distribution that was estimated by MCMC.

The text output shows a statistical summary:

```
     mean     median       mode    bnd.low    bnd.hi n.samples
 7.51e-01   7.62e-01   7.90e-01   5.45e-01   9.37e-01   5.00e+04
```

It will be interesting to compare these results to other results computed below.

3. Checking the connection between R and OpenBUGS via `R2OpenBUGS` TOC

Table 2. Test the **`R2OpenBUGS`** interface between R and OpenBUGS.

`# R Code`	Explanation
`if ("package:R2WinBUGS" %in% search())` `detach("package:R2WinBUGS")`	This command detaches the **`R2WinBUGS`** package from the search path in the case where it is present on the search path. The reason for doing this is that the **`bugs`** function in **`R2WinBUGS`** package can interfere with the use of the **`bugs`** function in the **`R2OpenBUGs`** package.

[1] If you don't have the **`jmfuns.rda`** function, it can be downloaded from http://faculty.washington.edu/jmiyamot/downloads.htm or https://faculty.washington.edu/jmiyamot/p548/p548-set.htm

`# The following cells (yellow background) are`
`# repeated from Table 1. They need to be`
`# repeated only if modelString and dataList`
`# have been removed from .GlobalEnv, e.g.,`
`# if you are starting a new R session.`	

`modelString = "`
`BUGS model specification begins ...`
`model {`
`  # Likelihood:`
`  for ( i in 1:nFlips ) {`
`  y[i] ~ dbern( theta )`
`  }`
`  # Prior distribution:`
`  theta ~ dbeta( priorA , priorB )`
`  priorA <- 1`
`  priorB <- 1`
`}`
`... BUGS model specification ends.`
`" # close quote to end modelString`
	THE MODEL.
	
	Specify the model in BUGS language, but save it as a string in R:

`writeLines(modelString,con="model.txt")`	Write the modelString to a file, using R commands:

`dataList = list(`
`  nFlips = 14 ,`
`  y = c( 1,1,1,1,1,1,1,1,1,1,0,0,0 )`
`)`
	`# THE DATA.`
	Specify the data in R, using a list format compatible with BUGS: [This is not the only way to specify the data for BUGS. JM will show you other ways.]

`library(R2OpenBUGS)`	Attach **R2OpenBUGS** to the search path.
	WARNING: When you run the next command, your R program will appear to freeze while it transfers the computation to OpenBUGS. R will regain control after OpenBUGS has finished the MCMC computation. On my i5 laptop, the OpenBUGS computation takes about 5 seconds, but it could be longer on a slower computer. If your computer freezes for more than 30 seconds, I would assume that it is officially frozen, and you will have to force the closure of R and OpenBUGS.

`bugs.out = bugs(`
`  data = dataList,`
`  inits = NULL,`
`  parameters.to.save = "theta",`
`  model.file= "model.txt",`
`  n.chains= 1,`
`  n.iter= 50000,`
`  n.burnin= 0,`
`  n.thin= 1`
`  )`
	bugs sends the computation to OpenBUGS. When OpenBUGS has finished, it returns the results to R. The results are stored in **bugs.out** which is a list structure.
	
	In future classes, we will thoroughly discuss the structure of the **bugs** function and its output. For now, suffice it to say that this function requests that OpenBUGS compute 1,000 parameter estimates from the posterior distribution of the **theta** parameter, and stores these estimates in the **bugs.out** list.

`names(bugs.out)`	We see that **bugs.out** has many components.
`names(bugs.out$sims.list)`	We see that the estimates of **theta** are stored in **bugs.out$sims.list**.

`est.theta = bugs.out$sims.list$theta`
`mode(est.theta)`
`length(est.theta)`
`est.theta[1:20]`
	We store the estimates of **theta** in **est.theta**. **est.theta** is a numeric vector of length 1,000, representing the 1,000 estimates of **theta** which have been sampled via MCMC simulation from the posterior distribution of **theta**.

`res.param(est.theta)`
 	
`mtext(`
`"Estimated Posterior Distribution of est.theta",`
`  side = 3, cex = 2, line = 2)`
`mtext(`
`"Estimates Computed by OpenBUGS Through the R2OpenBUGS Interface",`
`  side = 3, cex = 1.25, line = 0)`
	The **res.param** function is in the **jmfuns.rda** file[2]. You will have to attach this file to the search path if you want to run this function. **res.param** makes a plot of the posterior distribution of a parameter.

`beta.exact = draw.beta(12, 4, plot.dist = FALSE)`
`x = beta.exact[ , 1] - 0.005`
`y = beta.exact[ , 2]`
 	
`lines(x, y, col = "red")`
	I have added the true beta(12, 4) distribution to this plot, except that I displaced the curve by 0.005 to the left in order to be able to see both the true distribution and the distribution that was estimated by MCMC.

This analysis finds:

```
     mean    median      mode   bnd.low    bnd.hi n.samples
 7.51e-01  7.62e-01  7.90e-01  5.45e-01  9.37e-01  5.00e+04
```

Not surprisingly, these results are exactly identical to the results produced by **BRugs** in Section 2. In both cases, the actual computation is performed by OpenBUGS. Next we will see check the performance of JAGS.

4. Checking the connection between R and JAGS via rjags TOC

Table 3. Test the **rjags** interface between R and JAGS.

# R Code	Explanation
`rm(list = ls())`	Delete (remove) all objects from **.GlobalEnv**, to clear the work space.
`graphics.off()`	Turn off all graphics windows, again to clear the work space.

`if ( .Platform$OS.type != "windows" ) {`
`  windows <- function( ... ) X11( ... )`
`}`
	Sets the meaning of the "windows" function so that it is compatible with either Microsoft Windows or with a Mac or Linux environment.

`require(rjags)`	

[2] If you don't have the **jmfuns.rda** function, it can be downloaded from http://faculty.washington.edu/jmiyamot/downloads.htm or https://faculty.washington.edu/jmiyamot/p548/p548-set.htm

`modelString = "`
`# JAGS model specification begins ...`
`model {`
`  # Likelihood:`
`  for ( i in 1:nFlips ) {`
`  y[i] ~ dbern( theta )`
`  }`
`  # Prior distribution:`
`  theta ~ dbeta( priorA , priorB )`
`  priorA <- 1`
`  priorB <- 1`
`}`
`# ... JAGS model specification ends.`
`" # close quote to end modelString`
	`# THE MODEL.`
	
	Specify the model in JAGS language, but save it as a string in R:
	
	It is interesting to note that the model language for OpenBUGS is identical to the model language (for the same model) in JAGS (see Table 1).

`writeLines(modelString,con="model.txt")`	Write the modelString to a file, using R commands:

`dataList = list(`
`  nFlips = 14 ,`
`  y = c( 1,1,1,1,1,1,1,1,1,1,0,0,0 )`
`)`
	`# THE DATA.`
	
	Specify the data in R, using a list format compatible with JAGS:

`# initsList = list( theta = sum(dataList$y) /` `  length(dataList$y)`	**`# INTIALIZE THE CHAIN.`** Can be done automatically in jags.model() by commenting out inits argument. Otherwise could be established as:
	`# RUN THE CHAINS.`
`parameters = c( "theta" )`	The parameter(s) to be monitored.
`adaptSteps = 500`	Number of steps to "tune" the samplers.
`burnInSteps = 1000`	Number of steps to "burn-in" the samplers.
`nChains = 3`	Number of chains to run.
`numSavedSteps=50000`	Total number of steps in chains to save.
`thinSteps=1`	Number of steps to "thin" (1=keep every step).
`nIter = ceiling((numSavedSteps*thinSteps) / nChains)`	Steps per chain.

`jagsModel = jags.model(`
`  "model.txt" , data=dataList , # inits=initsList ,`
`  n.chains=nChains , n.adapt=adaptSteps )`
	Create, initialize, and adapt the model:

`cat( "Burning in the MCMC chain...\n" )`	**`# Burn-in:`**
`update( jagsModel , n.iter=burnInSteps )`	

`cat( "Sampling final MCMC chain...\n" )`
 	
`codaSamples = coda.samples( jagsModel ,`
`  variable.names=parameters, n.iter=nIter,`
`  thin=thinSteps )`
	The saved MCMC chain:
	
	Tesulting codaSamples object has these indices:
	
	codaSamples[[chainIdx]][stepIdx , paramIdx]

`mcmcChain = as.matrix( codaSamples )`
 	
`thetaSample = mcmcChain`
	`# EXAMINE THE RESULTS.`
	
	Convert coda-object **`codaSamples`** to matrix object for easier handling. But note that this concatenates the different chains into one long chain. Result is
	`mcmcChain[ stepIdx , paramIdx ]`

`mode(thetaSample)`
`length(thetaSample)`
`thetaSample[1:20, ]`
	`thetaSample` should exist in your R data set. It should be a matrix with 50,001 rows (the simulation produced 50,001 estimates of theta) and 1 column. If this is in fact true, then you have JAGS sucessfully installed and the **`Rjags`** interface is working properly with R.

`summary(thetaSample)`	BRugs asks BUGS for summary statistics. Again, if you can see these statistics, then the **`BRugs`** interface is working properly with R.

`res.param(thetaSample)`
 	
`mtext(`
`"Estimated Posterior Distribution of ThetaSample",`
`  side = 3, cex = 2, line = 2)`
`mtext(`
`"Estimates Computed by JAGS Through the Rjags Interface",`
`  side = 3, cex = 1.25, line = 0)`
	The **`res.param`** function is in the **`jmfuns.rda`** file[3]. You will have to attach this file to the search path if you want to run this function. **`res.param`** makes a plot of the posterior distribution of a parameter.

`beta.exact = draw.beta(12, 4, plot.dist = FALSE)`
`x = beta.exact[ , 1] - 0.005`
`y = beta.exact[ , 2]`
 	
`lines(x, y, col = "red")`
	I have added the true beta(12, 4) distribution to this plot, except that I displaced the curve by 0.005 to the left in order to be able to see both the true distribution and the distribution that was estimated by MCMC.

Summary of Results from the Rjags Run:

```
     mean    median      mode   bnd.low    bnd.hi n.samples
    0.750     0.760     0.778     0.543     0.936 50001.000
```

Comparing these results to the results from **`BRugs`** and **`R2OpenBUGS`**, we can see that they are virtually identical.

5. Checking the connection between R and JAGS via `R2jags` TOC

Table 4. Test the **`R2jags`** interface between R and JAGS.

# R Code	Explanation
`rm(list = ls())`	Delete (remove) all objects from **`.GlobalEnv`**, to clear the work space.
`graphics.off()`	Turn off all graphics windows, again to clear the work space.

`if ( .Platform$OS.type != "windows" ) {`
`  windows <- function( ... ) X11( ... )`
`}`
	Sets the meaning of the "windows" function so that it is compatible with either Microsoft Windows or with a Mac or Linux environment.

`library(R2jags)` `search()`	Notice that the **`R2jags`** package requires that **`Rjags`** and **`R2WinBUGS`** be on the search path.

`# All of the following code (shaded yellow) is the`
`# same as the code for the Rjags example.  It is`
`# reproduced here because it was deleted when we`
`# cleared the work space.`	

[3] If you don't have the **`jmfuns.rda`** function, it can be downloaded from http://faculty.washington.edu/jmiyamot/downloads.htm or https://faculty.washington.edu/jmiyamot/p548/p548-set.htm

`modelString = "`
`# JAGS model specification begins ...`
`model {`
`  # Likelihood:`
`  for ( i in 1:nFlips ) {`
`  y[i] ~ dbern( theta )`
`  }`
`  # Prior distribution:`
`  theta ~ dbeta( priorA , priorB )`
`  priorA <- 1`
`  priorB <- 1`
`}`
`# ... JAGS model specification ends.`
`" # close quote to end modelString`
	`# THE MODEL.`
	
	Specify the model in JAGS language, but save it as a string in R:
	
	It is interesting to note that the model language for OpenBUGS is identical to the model language (for the same model) in JAGS (see Table 1).

`writeLines(modelString,con="model.txt")`	Write the modelString to a file, using R commands:

`dataList = list(`
`  nFlips = 14 ,`
`  y = c( 1,1,1,1,1,1,1,1,1,1,0,0,0 )`
`)`
	`# THE DATA.`
	
	Specify the data in R, using a list format compatible with JAGS:

`# initsList = list( theta = sum(dataList$y) /` `  length(dataList$y)`	**`# INTIALIZE THE CHAIN.`** Can be done automatically in jags.model() by commenting out inits argument. Otherwise could be established as:
	`# RUN THE CHAINS.`
`parameters = c( "theta" )`	The parameter(s) to be monitored.
`adaptSteps = 500`	Number of steps to "tune" the samplers.
`burnInSteps = 1000`	Number of steps to "burn-in" the samplers.
`nChains = 3`	Number of chains to run.
`numSavedSteps=50000`	Total number of steps in chains to save.
`thinSteps=1`	Number of steps to "thin" (1=keep every step).
`nIter = ceiling((numSavedSteps*thinSteps) / nChains)`	Steps per chain.
`# The following R2jags command has the same` `# structure as the preceding Rjags computation.`	
`jags.out = jags(`	The output of the **`jags`** analysis will be saved in **`jags.out`**.
`  data = dataList,`	Specify the data.
`  inits = NULL,`	We could specify the initial values (start values) for the 3 chains of sampled estimates. By setting **`inits = NULL`**, we let **`JAGS`** choose the initial values at random.
`  parameters.to.save = parameters,`	The parameter(s) to be monitored (returned to R for further analysis).
`  n.iter= nIter + 1000,`	The number of iterations (samples of estimates) to be drawn in each chain.
`  model.file= "model.txt",`	The designation of a model file.
`  n.chains= nChains,`	The number of chains of estimates.
`  n.burnin= burnInSteps,`	The number of "burn-in" samples that will be discarded.

`    n.thin= thinSteps )`	The rate at which samples will be saved from a chain, e.g., **`n.thin = 5`** would mean that every 5th sample is saved.

`# Next we need to find the component of the output`
`# that contains the samples from the posterior`
`# distribution of theta.`	

`names(jags.out)`	Names of the components of the list, **`jags.out`**.
`names(jags.out$BUGSoutput)`	Names of the component, **`jags.out$BUGSoutput`**.
`names(jags.out$BUGSoutput$sims.list)`	Names of the component, **`jags.out$BUGSoutput$sims.list`**. If you are wondering how I knew that the estimates of **`theta`** would be stored in the **`sims.list`** component of **`BUGSoutput`**, I based this guess on the fact that the **`bugs`** function in **`R2OpenBUGS`** also stores the parameter estimates in a component called **`sims.list`**. (Also, I know that **`R2WinBUGS`**, **`R2OpenBUGS`** and **`R2jags`** were all written by researchers in the same lab, so we can expect parallels in how the functions in these packages will work.)
`est.theta = jags.out$BUGSoutput$sims.list$theta`	

`mode(est.theta)`
`length(est.theta)`
`est.theta[1:20]`	

`res.param(est.theta)`
 	
`mtext(`
`"Estimated Posterior Distribution of est.theta",`
`   side = 3, cex = 2, line = 2)`
`mtext(`
`"Estimates Computed by JAGS Through the R2jags Interface",`
`   side = 3, cex = 1.25, line = 0)`
	The **`res.param`** function is in the **`jmfuns.rda`** file[4]. You will have to attach this file to the search path if you want to run this function. **`res.param`** makes a plot of the posterior distribution of a parameter.

`beta.exact = draw.beta(12, 4, plot.dist = FALSE)`
`x = beta.exact[ , 1] - 0.005`
`y = beta.exact[ , 2]`
 	
`lines(x, y, col = "red")`
	I have added the true beta(12, 4) distribution to this plot, except that I displaced the curve by 0.005 to the left in order to be able to see both the true distribution and the distribution that was estimated by MCMC.

Summary of Results from the R2jags Run:

```
     mean     median       mode    bnd.low    bnd.hi n.samples
    0.750      0.760      0.781      0.542      0.934 50001.000
```

Obviously, the results are similar from all four methods. The plots of the estimated posterior distributions are shown on the next page.

[4] If you don't have the **`jmfuns.rda`** function, it can be downloaded from http://faculty.washington.edu/jmiyamot/downloads.htm or https://faculty.washington.edu/jmiyamot/p548/p548-set.htm

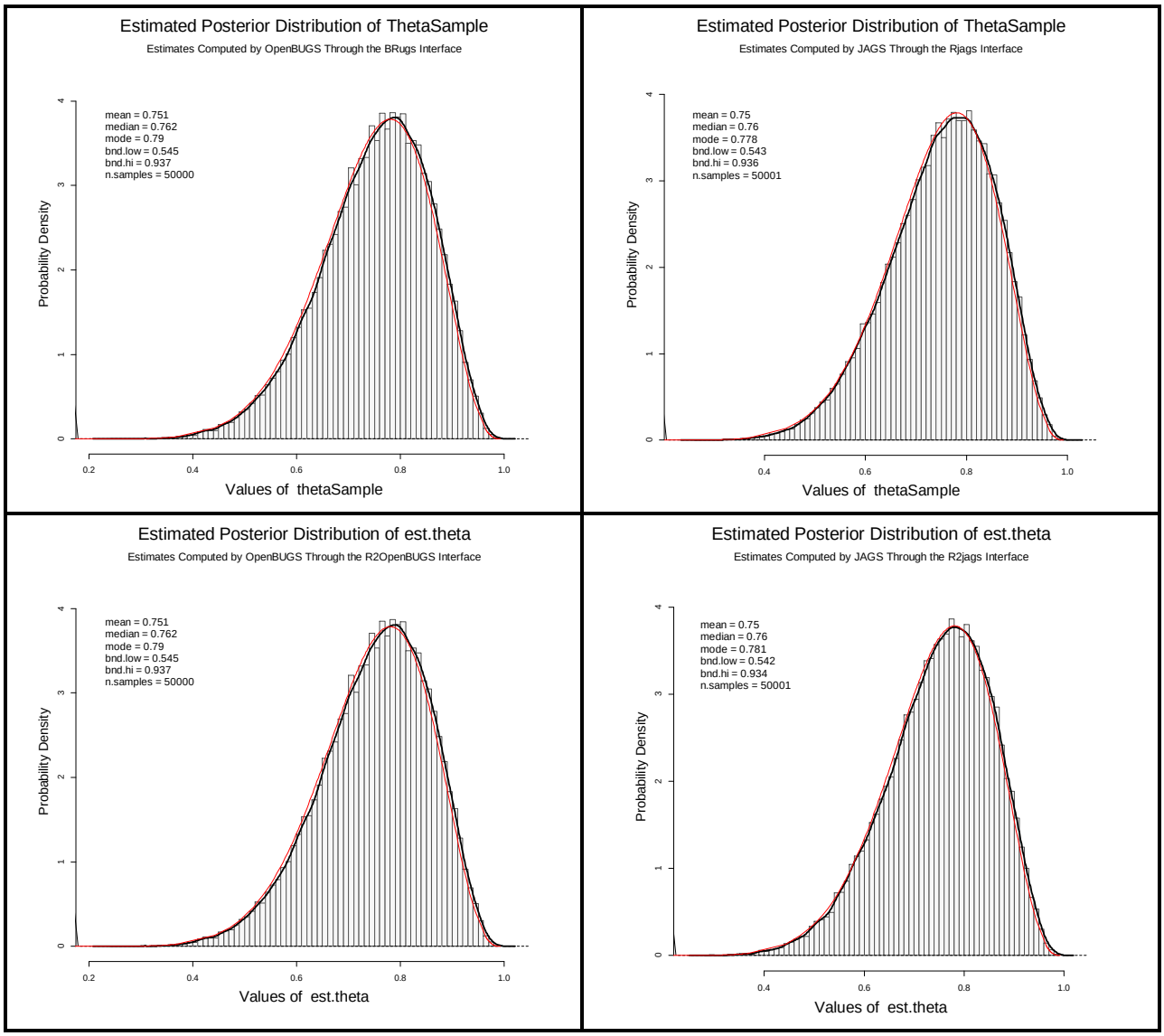

Figure 2. Posterior distributions of **`theta`** estimated by four different methods. The thin red line shows the beta(12, 4) distribution (exact posterior distribution); it has been displaced to the left by 0.005 in order to make it more visible.

6. Text version of the R-code in Tables 1 - 4.

```
# The code shown below is also available in a separate text file called 'test.bugs.install.txt'.
# You may want to copy this code from the current document and paste it into a script file
# within RStudio.  Then you can run it in RStudio.

# File:   test.bugs.install
# Title:  Test Run to Check the Installation of
# OpenBUGS, JAGS, BRugs, R2OpenBUGS, Rjags & R2jags
#------------------------------------------------------------

# Table 1.  Test the BRugs interface between R and OpenBUGS.
```

```
# The code in this table is a subset for the code in
# Kruschke's script file, BernBetaBugsFull.R.
library(BRugs)
modelString = "
BUGS model specification begins ...
model {
    # Likelihood:
    for ( i in 1:nFlips ) {
        y[i] ~ dbern( theta )
    }
    # Prior distribution:
    theta ~ dbeta( priorA , priorB )
    priorA <- 1
    priorB <- 1
}
... BUGS model specification ends.
" # close quote to end modelString
writeLines(modelString,con="model.txt")
modelCheck( "model.txt" )
dataList = list(
    nFlips = 14 ,
    y = c( 1,1,1,1,1,1,1,1,1,1,0,0,0 )
)
modelData( bugsData( dataList ) )
modelCompile()
modelGenInits()
samplesSet( "theta" )
chainLength = 50000
modelUpdate( chainLength )
thetaSample = samplesSample( "theta" )
mode(thetaSample)
length(thetaSample)
thetaSample[1:20]
thetaSummary = samplesStats( "theta" )
thetaSummary
res.param(thetaSample)

mtext(
"Estimated Posterior Distribution of ThetaSample",
   side = 3, cex = 2, line = 2)
mtext(
"Estimates Computed by OpenBUGS Through the BRugs Interface",
   side = 3, cex = 1.25, line = 0)
beta.exact = draw.beta(12, 4, plot.dist = FALSE)
x = beta.exact[ , 1] - 0.005
y = beta.exact[ , 2]

lines(x, y, col = "red")
#-------------------------------------------------------------

# Table 2.  Test the R2OpenBUGS interface between R and
# OpenBUGS.
if ("package:R2WinBUGS" %in% search())
   detach("package:R2WinBUGS")
```

```
# The following cells (yellow background) are
# repeated from Table 1.  They need to be
# repeated only if modelString and dataList
# have been removed from .GlobalEnv, e.g.,
# if you are starting a new R session.
modelString = "
BUGS model specification begins ...
model {
    # Likelihood:
    for ( i in 1:nFlips ) {
        y[i] ~ dbern( theta )
    }
    # Prior distribution:
    theta ~ dbeta( priorA , priorB )
    priorA <- 1
    priorB <- 1
}
... BUGS model specification ends.
" # close quote to end modelString
writeLines(modelString,con="model.txt")
dataList = list(
    nFlips = 14 ,
    y = c( 1,1,1,1,1,1,1,1,1,1,0,0,0 )
)
library(R2OpenBUGS)

bugs.out = bugs(
   data = dataList,
   inits = NULL,
   parameters.to.save = "theta",
   model.file= "model.txt",
   n.chains= 1,
   n.iter= 50000,
   n.burnin= 0,
   n.thin= 1
   )
names(bugs.out)
names(bugs.out$sims.list)
est.theta = bugs.out$sims.list$theta
mode(est.theta)
length(est.theta)
est.theta[1:20]
res.param(est.theta)

mtext(
"Estimated Posterior Distribution of est.theta",
   side = 3, cex = 2, line = 2)
mtext(
"Estimates Computed by OpenBUGS Through the R2OpenBUGS Interface",
   side = 3, cex = 1.25, line = 0)
beta.exact = draw.beta(12, 4, plot.dist = FALSE)
x = beta.exact[ , 1] - 0.005
y = beta.exact[ , 2]
```

```
lines(x, y, col = "red")
#------------------------------------------------------------

# Table 3.  Test the rjags interface between R and JAGS.
rm(list = ls())
graphics.off()
if ( .Platform$OS.type != "windows" ) {
  windows <- function( ... ) X11( ... )
}
require(rjags)
modelString = "
# JAGS model specification begins ...
model {
    # Likelihood:
    for ( i in 1:nFlips ) {
        y[i] ~ dbern( theta )
    }
    # Prior distribution:
    theta ~ dbeta( priorA , priorB )
    priorA <- 1
    priorB <- 1
}
# ... JAGS model specification ends.
" # close quote to end modelString
writeLines(modelString,con="model.txt")
dataList = list(
    nFlips = 14 ,
    y = c( 1,1,1,1,1,1,1,1,1,1,1,0,0,0 )
)
# initsList = list( theta = sum(dataList$y) /        length(dataList$y)

parameters = c( "theta" )
adaptSteps = 500
burnInSteps = 1000
nChains = 3
numSavedSteps=50000
thinSteps=1
nIter = ceiling((numSavedSteps*thinSteps) / nChains)
jagsModel = jags.model(
   "model.txt" , data=dataList , # inits=initsList ,
   n.chains=nChains , n.adapt=adaptSteps )
cat( "Burning in the MCMC chain...\n" )
update( jagsModel , n.iter=burnInSteps )
cat( "Sampling final MCMC chain...\n" )

codaSamples = coda.samples( jagsModel ,
   variable.names=parameters, n.iter=nIter,
   thin=thinSteps )
mcmcChain = as.matrix( codaSamples )

thetaSample = mcmcChain
mode(thetaSample)
length(thetaSample)
thetaSample[1:20, ]
```

```
summary(thetaSample)
res.param(thetaSample)

mtext(
"Estimated Posterior Distribution of ThetaSample",
   side = 3, cex = 2, line = 2)
mtext(
"Estimates Computed by JAGS Through the Rjags Interface",
   side = 3, cex = 1.25, line = 0)
beta.exact = draw.beta(12, 4, plot.dist = FALSE)
x = beta.exact[ , 1] - 0.005
y = beta.exact[ , 2]

lines(x, y, col = "red")
#-------------------------------------------------------

# Table 4.  Test the R2jags interface between R and JAGS.
rm(list = ls())
graphics.off()
if ( .Platform$OS.type != "windows" ) {
  windows <- function( ... ) X11( ... )
}
library(R2jags)
search()
# All of the following code (shaded yellow) is the
# same as the code for the Rjags example.  It is
# reproduced here because it was deleted when we
# cleared the work space.
modelString = "
# JAGS model specification begins ...
model {
    # Likelihood:
    for ( i in 1:nFlips ) {
        y[i] ~ dbern( theta )
    }
    # Prior distribution:
    theta ~ dbeta( priorA , priorB )
    priorA <- 1
    priorB <- 1
}
# ... JAGS model specification ends.
" # close quote to end modelString
writeLines(modelString,con="model.txt")
dataList = list(
    nFlips = 14 ,
    y = c( 1,1,1,1,1,1,1,1,1,1,1,0,0,0 )
)
# initsList = list( theta = sum(dataList$y) /        length(dataList$y)

parameters = c( "theta" )
adaptSteps = 500
burnInSteps = 1000
nChains = 3
numSavedSteps=50000
```

```
thinSteps=1
nIter = ceiling((numSavedSteps*thinSteps) / nChains)
# The following R2jags command has the same
# structure as the preceding Rjags computation.

jags.out = jags(
    data = dataList,
    inits = NULL,
    parameters.to.save = parameters,
    n.iter= nIter + 1000,
    model.file= "model.txt",
    n.chains= nChains,
    n.burnin= burnInSteps,
    n.thin= thinSteps )
# Next we need to find the component of the output
# that contains the samples from the posterior
# distribution of theta.
names(jags.out)
names(jags.out$BUGSoutput)
names(jags.out$BUGSoutput$sims.list)
est.theta = jags.out$BUGSoutput$sims.list$theta
mode(est.theta)
length(est.theta)
est.theta[1:20]
res.param(est.theta)

mtext(
"Estimated Posterior Distribution of est.theta",
   side = 3, cex = 2, line = 2)
mtext(
"Estimates Computed by JAGS Through the R2jags Interface",
   side = 3, cex = 1.25, line = 0)
beta.exact = draw.beta(12, 4, plot.dist = FALSE)
x = beta.exact[ , 1] - 0.005
y = beta.exact[ , 2]

lines(x, y, col = "red")
#-------------------------------------------------------------
```