Solutions to Exercises in Doing Bayesian Data Analysis Second Edition

Solutions manual by John K. Kruschke

 This version: 2/16/2015 4:21 PM

This manual provides solutions to exercises in the book *Doing Bayesian data analysis, 2nd Edition: A tutorial with R, JAGS, and Stan*, by John K. Kruschke (2015), published by Academic Press / Elsevier, ISBN-13: 978-0124058880.

This document and additional information can be found at https://sites.google.com/site/doingbayesiandataanalysis/

Contents

Chapter 1.

Chapter 1 has no exercises, because it describes the organization of the book and expresses acknowledgments.

Chapter 2.

Exercise 2.1.

Exercise 2.1. [Purpose: To get you actively manipulating mathematical models of probabilities.] Suppose we have a four-sided die from a board game. On a tetrahedral die, each face is an equilateral triangle. When you roll the die, it lands with one face down and the other three faces visible as a three-sided pyramid. The faces are numbered 1-4, with the value of the bottom face printed (as clustered dots) at the bottom edges of all three visible faces. Denote the value of the bottom face as x. Consider the following three mathematical descriptions of the probabilities of x. Model A: $p(x) = 1/4$. Model B: $p(x) = x/10$. Model C: $p(x) = 12/(25x)$. For each model, determine the value of $p(x)$ for each value of x. Describe in words what kind of bias (or lack of bias) is expressed by each model.

Model A:
p(x=1) = 1/4,
p(x=2) = 1/4,
p(x=3) = 1/4,
p(x=4) = 1/4.
(Notice that the probabilities sum to 1.0.) This model has no bias in the sense that all values of x are equally likely.

Model B:
p(x=1) = 1/10,
p(x=2) = 2/10,
p(x=3) = 3/10,
p(x=4) = 4/10.
(Notice that the probabilities sum to 1.0.) This model is biased toward higher values of x.

Model C:
p(x=1) = 12/25 = 144/300,
p(x=2) = 12/50 = 72/300,
p(x=3) = 12/75 = 48/300,
p(x=4) = 12/100 = 36/300.
(Notice that the probabilities sum to 1.0.) This model is biased toward lower values of x.

Exercise 2.2.

Exercise 2.2. [Purpose: To get you actively thinking about how data cause credibilities to shift.] Suppose we have the tetrahedral die introduced in the previous exercise, along with the three candidate models of the die's probabilities. Suppose that initially, we are not sure what to believe about the die. On the one hand, the die might be fair, with each face landing with the same probability. On the other hand, the die might be biased, with the faces that have more dots landing down more often (because the dots are created by embedding heavy jewels in the die, so that the sides with more dots are more likely to land on the bottom). On yet another hand, the die might be biased such that more dots on a face make it less likely to land down (because maybe the dots are bouncy rubber or protrude from the surface). So, initially, our beliefs about the three models can be described as $p(A) = p(B) = p(C) = 1/3$. Now we roll the die 100 times and find these results: #1's = 25, #2's = 25, #3's = 25, #4's = 25. Do these data change our beliefs about the models? Which model now seems most likely? Suppose when we rolled the die 100 times we found these results: #1's = 48, #2's = 24, #3's = 16, #4's = 12. Now which model seems most likely?

When the data show 25 of each face, then we change our beliefs such that Model A becomes more credible, because the data are most consistent with Model A.

When the data show 48 1's, 24 2's, etc., then we change our beliefs such that Model C becomes more credible, because the data are most consistent with Model C. Moreover, Model B becomes least credible, because its predictions are least consistent with the data.

Chapter 3.

Exercise 3.1.

This exercise asks you to install R on your computer. Hopefully you succeeded, because all the subsequent exercises rely on it.

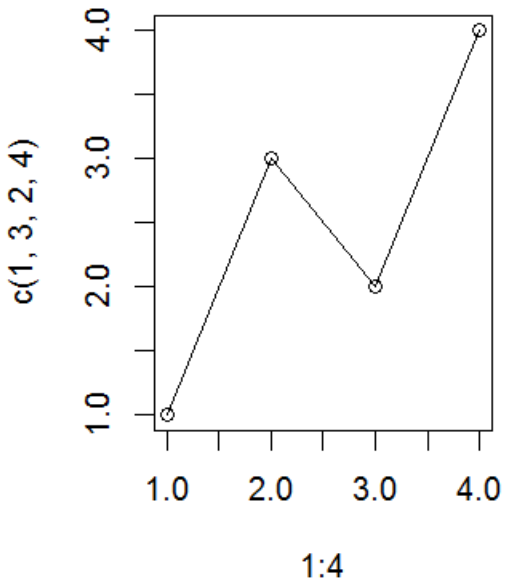
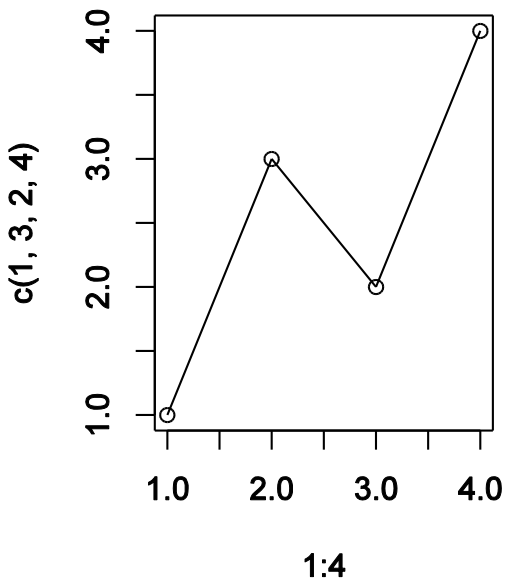
Exercise 3.2.

Exercise 3.2. [Purpose: Being able to record and communicate the results of your analyses.] Open the program `ExampleOfR.R`. At the end, notice the section that produces a simple graph. Your job is to save the graph so you can incorporate it into documents in the future, as you would for reporting actual data analyses. Save the graph in a format that is compatible with your word processing software. Import the saved file into your document and explain, in text, what you did. (Notice that for some word processing systems you could merely copy and paste directly from R's graphic window to the document window. But the problem with this approach is that you have no other record of the graph produced by the analysis. We want the graph to be saved separately so that it can be incorporated into various reports at a future time.)

In the R code below, the graph is saved twice, in different graphics formats:

```
# Exercise 3.2
source("DBDA2E-utilities.R")             # read defn. of openGraph, saveGraph
openGraph( width=3 , height=4 )          # open a graphics window
plot( x=1:4 , y=c(1,3,2,4) , type="o" )  # make a plot in the screen window
# Save the graph with specified file name and graphics type:
saveGraph( file="Exercise.03.2" , type="png" )
# Save the graph with specified file name and graphics type:
saveGraph( file="Exercise.03.2" , type="eps" )
```

To put the saved graph in a Microsoft Word document, click "Insert" then "Picture" and browse to the file Exercise.03.2.png or the file Exercise.03.2.eps. Insert the picture and then arrange as desired. I have inserted the figures into a table, below:

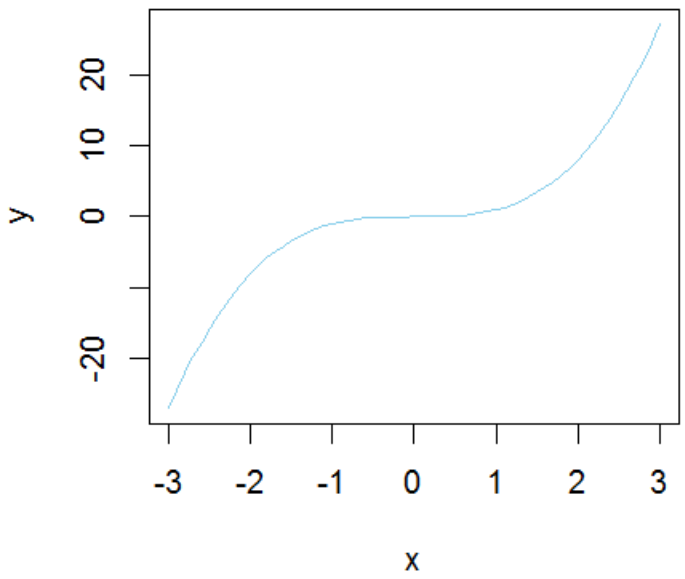
	
This is the inserted PNG file. PNG is a pixelated format. Its best resolution is limited by its original number of pixels.	This is the inserted EPS file. It is a vector format. The vectors are rendered as pixels in whatever device is being used to view it, such as a computer monitor or printer. When a Word file is converted to PDF, EPS figures are sometimes rendered badly, especially if they have dashed or dotted lines. But EPS files are often preferred by users of LaTeX.

Exercise 3.3.

Exercise 3.3. [Purpose: Getting experience with the details of the command syntax within R.] Adapt the program `SimpleGraph.R` so that it plots a cubic function ($y = x^3$) over the interval $x \in [-3, +3]$. Save the graph in a file format of your choice. Include a properly commented listing of your code, along with the resulting graph.

```
# Exercise 3.3
source("DBDA2E-utilities.R")             # read defn. of openGraph, saveGraph
openGraph( width=4 , height=4 )          # open a graphics window
x = seq( from = -3 , to = 3 , by = 0.1 )   # Specify vector of x values.
y = x^3                                    # Specify corresponding y values.
plot( x , y , col="skyblue" , type="l" )   # Plot the x,y points as a blue line.
saveGraph( file="Exercise.03.3" , type="png")
```

The resulting graph is inserted below:



For Exercise 3.3.

Chapter 4.

Exercise 4.1.

Exercise 4.1. [Purpose: To gain experience with the `apply` function in R, while dealing with a concrete example of computing conditional probabilities.] The eye-color hair-color data from Table 4.1 are built into R as the array named HairEyeColor. The array is frequencies of eye and hair color for males and females. Run the following code in R:

```
show( HairEyeColor ) # Show data
EyeHairFreq = apply( HairEyeColor, c("Eye","Hair"), sum ) # Sum across sex
EyeHairProp = EyeHairFreq / sum( EyeHairFreq )  # joint proportions, Table 4.1
show( round( EyeHairProp , 2 ) )
HairFreq = apply( HairEyeColor , c("Hair") , sum ) # Sum across sex and eye
HairProp = HairFreq / sum( HairFreq ) # marginal proportions, Table 4.1
show( round( HairProp , 2 ) )
EyeFreq = apply( HairEyeColor , c("Eye") , sum ) # Sum across sex and eye
EyeProp = EyeFreq / sum( EyeFreq ) # marginal proportions, Table 4.1
show( round( EyeProp , 2 ) )
EyeHairProp["Blue",] / EyeProp["Blue"] # conditional prob, Table 4.2
```

In your write-up, include each line above and its results. *Explain* what each line does (in a bit more detail than the inline comments). Extend the above commands by also computing the probabilities of the hair colors given Brown eyes, and the probabilities of the eye colors given Brown hair.

```
> show( HairEyeColor ) # Show data
, , Sex = Male

       Eye
Hair    Brown Blue Hazel Green
  Black    32   11    10     3
  Brown    53   50    25    15
  Red      10   10     7     7
  Blond     3   30     5     8

, , Sex = Female

       Eye
Hair    Brown Blue Hazel Green
  Black    36    9     5     2
  Brown    66   34    29    14
  Red      16    7     7     7
  Blond     4   64     5     8
```

Note regarding output above: The data, HairEyeColor, are in a 3-dimensional array, with two layers for Male and Female.

```
> EyeHairFreq = apply( HairEyeColor, c("Eye","Hair"), sum ) # Sum across sex
> EyeHairProp = EyeHairFreq / sum( EyeHairFreq ) # joint proportions, Table 4.1
> show( round( EyeHairProp , 2 ) )
       Hair
Eye     Black Brown  Red Blond
  Brown  0.11  0.20 0.04  0.01
  Blue   0.03  0.14 0.03  0.16
  Hazel  0.03  0.09 0.02  0.02
  Green  0.01  0.05 0.02  0.03
```

Note regarding output above: apply() applies the sum function to the HairEyeColor array, across all dimensions except Eye and Hair. The next line divides by the total N, to compute the proportion of the sample in each cell.

```
> HairFreq = apply( HairEyeColor , c("Hair") , sum ) # Sum across sex and eye
> HairProp = HairFreq / sum( HairFreq ) # marginal proportions, Table 4.1
> show( round( HairProp , 2 ) )
Black Brown   Red Blond
 0.18  0.48  0.12  0.21
```

Note regarding output above: See previous note. Result is marginal proportions of hair colors.

```
> EyeFreq = apply( HairEyeColor , c("Eye") , sum ) # Sum across sex and eye
> EyeProp = EyeFreq / sum( EyeFreq ) # marginal proportions, Table 4.1
> show( round( EyeProp , 2 ) )
Brown  Blue Hazel Green
 0.37  0.36  0.16  0.11
```

Note regarding output above: See previous note. Result is marginal proportions of eye colors.

```
> EyeHairProp["Blue",] / EyeProp["Blue"] # conditional prob, Table 4.2
     Black      Brown        Red      Blond
0.09302326 0.39069767 0.07906977 0.43720930
```

Note regarding output above: This is the "Blue" row of EyeHairProp divided by the "Blue" cell of the marginal, EyeProp. The result is the conditional probability of hair color given blue eyes.

```
> # Hair colors given brown eyes:
> # Notice location of comma!
> EyeHairProp["Brown",] / EyeProp["Brown"]

     Black      Brown        Red      Blond
0.30909091 0.54090909 0.11818182 0.03181818

> # Eye colors given brown hair:
> # Notice location of comma!
> EyeHairProp[,"Brown"] / HairProp["Brown"]

    Brown      Blue     Hazel     Green
0.4160839 0.2937063 0.1888112 0.1013986
```

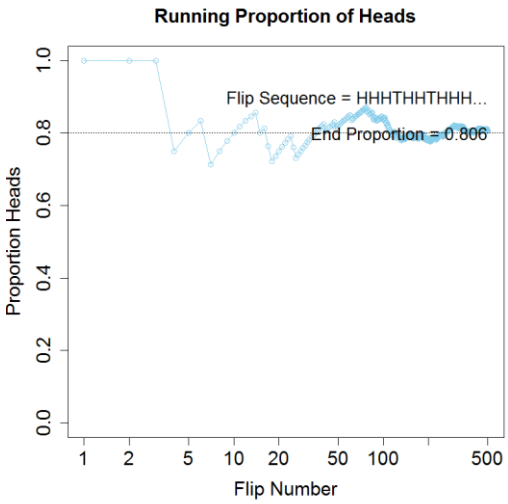
Exercise 4.2.

Exercise 4.2. **[Purpose: To give you some experience with random number generation in R.]** Modify the coin flipping program in Section 4.5 `RunningProportion.R` to simulate a biased coin that has $p(H) = 0.8$. Change the height of the reference line in the plot to match $p(H)$. Comment your code. *Hint:* Read the help for the `sample` command.

```
# For Exercise 04.2
source("DBDA2E-utilities.R")  # read defn. of openGraph, saveGraph
N = 500 # Specify the total number of flips, denoted N.
# CHANGED NEXT LINE TO 0.8:
pHeads = 0.8 # Specify underlying probability of heads.
# Flip a coin N times and compute the running proportion of heads at each flip.
# Generate a random sample of N flips (heads=1, tails=0):
flipSequence = sample( x=c(0,1), prob=c(1-pHeads,pHeads), size=N, replace=TRUE )
# NO NEED TO CHANGE ARGUMENT IN sample() ABOVE BECAUSE IT'S ALREADY SPECIFIED IN
# TERMS OF pHeads
# Compute the running proportion of heads:
r = cumsum( flipSequence ) # Cumulative sum: Number of heads at each step.
n = 1:N                    # Number of flips at each step.
```

```
runProp = r / n            # Component by component division.
# Graph the running proportion:
openGraph()
plot( n , runProp , type="o" , log="x" , col="skyblue" ,
      xlim=c(1,N) , ylim=c(0.0,1.0) , cex.axis=1.5 ,
      xlab="Flip Number" , ylab="Proportion Heads" , cex.lab=1.5 ,
      main="Running Proportion of Heads" , cex.main=1.5 )
# Plot a dotted horizontal reference line:
abline( h=pHeads , lty="dotted" )
# NO NEED TO CHANGE ARGUMENT IN abline() ABOVE BECAUSE IT'S ALREADY SPECIFIED IN
# TERMS OF pHeads
# Display the beginning of the flip sequence:
flipLetters = paste( c("T","H")[flipSequence[1:10]+1] , collapse="" )
displayString = paste0( "Flip Sequence = " , flipLetters , "..." )
text( N , .9 , displayString , adj=c(1,0.5) , cex=1.3 )
# Display the relative frequency at the end of the sequence.
text( N , .8 , paste("End Proportion =",runProp[N]) , adj=c(1,0.5) , cex=1.3 )
```

Resulting graph is below. Your sample will differ because of randomness.



Exercise 4.3.

Exercise 4.3. [Purpose: To have you work through an example of the logic presented in Section 4.2.1.2.] Determine the exact probability of drawing a 10 from a shuffled pinochle deck. (In a pinochle deck, there are 48 cards. There are six values: 9, 10, Jack, Queen, King, Ace. There are two copies of each value in each of the standard four suits: hearts, diamonds, clubs, spades.)

(A) What is the probability of getting a 10?

(B) What is the probability of getting a *10 or Jack*?

(A) There are 8 cards of value 10. Hence the probability of getting a 10 is 8/48 = 0.1666…

(B) The values 10 and Jack are mutually exclusive, so we add their probabilities to get the probability of the event that contains either 10 or Jack. Each has probability 8/48, so the probability of 10 or Jack is 8/48 + 8/48 = 0.333…

Exercise 4.4.

Exercise 4.4. [Purpose: To give you hands-on experience with a simple probability density function, in R and in calculus, and to reemphasize that density functions can have values larger than 1.] Consider a spinner with a [0,1] scale on its circumference. Suppose that the spinner is slanted or magnetized or bent in some way such that it is biased, and its probability density function is $p(x) = 6x(1-x)$ over the interval $x \in [0, 1]$.

(A) Adapt the program `IntegralOfDensity.R` to plot this density function and approximate its integral. Comment your code. Be careful to consider values of x only in the interval $[0, 1]$. *Hint:* You can omit the first couple of lines regarding `meanval` and `sdval`, because those parameter values pertain only to the normal distribution. Then set `xlow=0` and `xhigh=1`, and set `dx` to some small value.

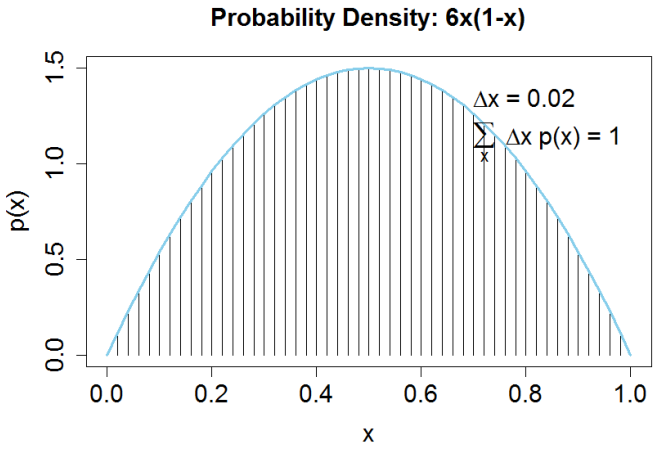
(B) Derive the exact integral using calculus. *Hint:* See the example, Equation 4.7.

(C) Does this function satisfy Equation 4.3?

(D) From inspecting the graph, what is the maximal value of $p(x)$?

(A)

```
source("DBDA2E-utilities.R")
# Graph of normal probability density function, with comb of intervals.
xlow  = 0 # Specify low end of x-axis.
xhigh = 1 # Specify high end of x-axis.
dx = 0.02 # Specify interval width on x-axis
# Specify comb of points along the x axis:
x = seq( from = xlow , to = xhigh , by = dx )
# Compute y values, i.e., probability density at each value of x:
y = 6 * x * ( 1 - x )
# Plot the function. "plot" draws the intervals. "lines" draws the bell curve.
openGraph(width=7,height=5)
plot( x , y , type="h" , lwd=1 , cex.axis=1.5
       , xlab="x" , ylab="p(x)" , cex.lab=1.5 ,
       , main="Probability Density: 6x(1-x)" , cex.main=1.5 )
lines( x , y , lwd=3 ,  col="skyblue" )
# Approximate the integral as the sum of width * height for each interval.
area = sum( dx * y )
# Display info in the graph.
text( 0.7 , .9*max(y) , bquote( paste(Delta , "x = " ,.(dx)) )
      , adj=c(0,.5) , cex=1.5 )
text( 0.7 , .75*max(y) ,
      bquote(
        paste( sum(,x,) , " " , Delta , "x p(x) = " , .(signif(area,3)) )
      ) , adj=c(0,.5) , cex=1.5 )
```



The resulting graph:

(B) $\int_0^1 6x(1-x)dx = 6\int_0^1 (x-x^2)dx = 6\left(\left[\frac{1}{2}x^2\right]_0^1 - \left[\frac{1}{3}x^3\right]_0^1\right) = 6\left(\frac{1}{2}-\frac{1}{3}\right) = 1.$

(C) Yes, the integral is 1.0, so the function satisfies Eqn 4.3.

(D) From the graph, it appears that the maximum density is 1.5 when x-0.5. Indeed, 6*0.5*(1-0.5) = 1.5.

Exercise 4.5.

Exercise 4.5. [Purpose: To have you use a normal curve to describe beliefs. It's also handy to know the area under the normal curve between μ and σ.]

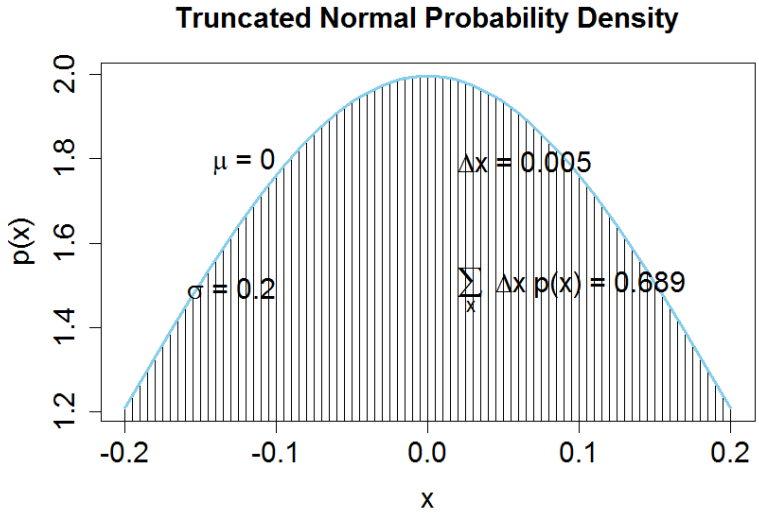
(A) Adapt the code from `IntegralOfDensity.R` to determine (approximately) the probability mass under the normal curve from $x = \mu - \sigma$ to $x = \mu + \sigma$. Comment your code. *Hint:* Just change `xlow` and `xhigh` appropriately, and change the `text` location so that the `area` still appears within the plot.

(B) Now use the normal curve to describe the following belief. Suppose you believe that women's heights follow a bell-shaped distribution, centered at 162 cm with about two-thirds of all women having heights between 147 and 177 cm. What should be the μ and σ parameter values?

(A)

```
source("DBDA2E-utilities.R")
# Graph of normal probability density function, with comb of intervals.
meanval = 0.0               # Specify mean of distribution.
sdval = 0.2                 # Specify standard deviation of distribution.
xlow  = meanval - 1.0*sdval # Specify low end of x-axis.
xhigh = meanval + 1.0*sdval # Specify high end of x-axis.
dx = sdval/40               # Specify interval width on x-axis
# Specify comb of points along the x axis:
x = seq( from = xlow , to = xhigh , by = dx )
# Compute y values, i.e., probability density at each value of x:
y = ( 1/(sdval*sqrt(2*pi)) ) * exp( -.5 * ((x-meanval)/sdval)^2 )
# Plot the function. "plot" draws the intervals. "lines" draws the bell curve.
openGraph(width=7,height=5)
plot( x , y , type="h" , lwd=1 , cex.axis=1.5
      , xlab="x" , ylab="p(x)" , cex.lab=1.5 ,
      , main="Truncated Normal Probability Density" , cex.main=1.5 )
lines( x , y , lwd=3 ,  col="skyblue" )
# Approximate the integral as the sum of width * height for each interval.
area = sum( dx * y )
# Display info in the graph.
text( meanval-0.5*sdval , .9*max(y) , bquote( paste(mu ," = " ,.(meanval)) )
      , adj=c(1,.5) , cex=1.5 )
text( meanval-0.5*sdval , .75*max(y) , bquote( paste(sigma ," = " ,.(sdval)) )
      , adj=c(1,.5) , cex=1.5 )
text( meanval+0.1*sdval , .9*max(y) , bquote( paste(Delta , "x = " ,.(dx)) )
      , adj=c(0,.5) , cex=1.5 )
text( meanval+0.1*sdval , .75*max(y) ,
      bquote(
        paste( sum(,x,) , " " , Delta , "x p(x) = " , .(signif(area,3)) )
      ) , adj=c(0,.5) , cex=1.5 )
```

Resulting graph (at right) shows that the area from $-\sigma$ to $+\sigma$ is about 0.689. In fact, the correct answer is closer to 0.683. This is very nearly 2/3 of the distribution.



(B) The description of beliefs indicates that the mean should be μ=162. Because 2/3 falls between 147 and 177, that implies that

$\mu - \sigma = 147$ and $\mu + \sigma = 177$,

hence $\sigma = 15$.

Exercise 4.6.

Exercise 4.6. [Purpose: Recognize and work with the fact that Equation 4.9 can be solved for the joint probability, which will be crucial for developing Bayes' theorem.] School children were surveyed regarding their favorite foods. Of the total sample, 20% were 1st graders, 20% were 6th graders, and 60% were 11th graders. For each grade, the following table shows the proportion of respondents that chose each of three foods as their favorite:

	Ice cream	Fruit	French fries
1st graders	0.3	0.6	0.1
6th graders	0.6	0.3	0.1
11th graders	0.3	0.1	0.6

From that information, construct a table of joint probabilities of grade and favorite food. Also, say whether grade and favorite food are independent or not, and how you ascertained the answer. *Hint:* You are given p(grade) and p(food|grade). You need to determine p(grade,food).

As the hint indicates, the table specifies p(food|grade), and the text provides p(grade). Next, remember that

p(food|grade) = p(food,grade)/p(grade),

which implies that

p(food,grade) = p(food|grade) * p(grade).

Hence, the table of joint probabilities is

	Ice cream	Fruit	French fries
1st grade	0.3*0.2=0.06	0.6*0.2=0.12	0.1*0.2=0.02
6th grade	0.6*0.2=0.12	0.3*0.2=0.06	0.1*0.2=0.02
11th grade	0.3*0.6=0.18	0.1*0.6=0.06	0.6*0.6=0.36

Notice that the sum of the joint probabilities across the nine cells is 1.0, as it should be.

Grade and food are not independent. This can be proven by exhibiting any cell for which p(food,grade) does not equal p(food) * p(grade). Consider Ice cream in 1st grade: p(Ice cream, 1st grade) is 0.06, while p(Ice cream) * p(1st grade) = 0.36 * 0.20 = 0.072.

Chapter 5.

Exercise 5.1.

Exercise 5.1. [Purpose: Iterative application of Bayes' rule, and seeing how posterior probabilities change with inclusion of more data.] This exercise extends the ideas of Table 5.4, so at this time, please review Table 5.4 and its discussion in the text. Suppose that the same randomly selected person as in Table 5.4 gets re-tested after the first test result was positive, and on the re-test, the result is negative. When taking into account the results of both tests, what is the probability that the person has the disease? *Hint:* For the prior probability of the re-test, use the posterior computed from the Table 5.4. Retain as many decimal places as possible, as rounding can have a surprisingly big effect on the results. One way to avoid unnecessary rounding is to do the calculations in R.

```
> # Specify hit rate of test:
> pPositiveGivenDisease = 0.99
> # Specify false alarm rate of test:
> pPositiveGivenNoDisease = 0.05
>
> # Specify the original prior:
> pDisease = 0.001
>
> # Bayes rule for first, positive test (this is the calculation on p. 104):
> pDiseaseGivenPositive = ( pPositiveGivenDisease * pDisease /
+                             ( pPositiveGivenDisease * pDisease
+                               + pPositiveGivenNoDisease * (1.0-pDisease) ) )
> show( pDiseaseGivenPositive )
[1] 0.01943463
>
> # Set the prior to the new probability of having the disease:
> pDisease = pDiseaseGivenPositive
>
> # Bayes rule for second, negative test:
> pDiseaseGivenNegative = ( (1.0-pPositiveGivenDisease) * pDisease /
+                             ( (1.0-pPositiveGivenDisease) * pDisease
+                               + (1.0-pPositiveGivenNoDisease) * (1.0-pDisease) ) )
> show( pDiseaseGivenNegative )
[1] 0.0002085862
```

Exercise 5.2.

Exercise 5.2. [Purpose: Getting an intuition for the previous results by using "natural frequency" and "Markov" representations]

(A) Suppose that the population consists of 100,000 people. Compute how many people would be expected to fall into each cell of Table 5.4. To compute the expected frequency of people in a cell, just multiply the cell probability by the size of the population. To get you started, a few of the cells of the frequency table are filled in here:

	$\theta = \ddot{\frown}$	$\theta = \ddot{\smile}$	
$D = +$	$\text{freq}(D=+, \theta=\ddot{\frown}) = p(D=+, \theta=\ddot{\frown})N = p(D=+\mid\theta=\ddot{\frown})p(\theta=\ddot{\frown})N = 99$	$\text{freq}(D=+, \theta=\ddot{\smile}) = p(D=+, \theta=\ddot{\smile})N = p(D=+\mid\theta=\ddot{\smile})p(\theta=\ddot{\smile})N =$	$\text{freq}(D=+) = p(D=+)N =$
$D = -$	$\text{freq}(D=-, \theta=\ddot{\frown}) = p(D=-, \theta=\ddot{\frown})N = p(D=-\mid\theta=\ddot{\frown})p(\theta=\ddot{\frown})N = 1$	$\text{freq}(D=-, \theta=\ddot{\smile}) = p(D=-, \theta=\ddot{\smile})N = p(D=-\mid\theta=\ddot{\smile})p(\theta=\ddot{\smile})N =$	$\text{freq}(D=-) = p(D=-)N =$
	$\text{freq}(\theta=\ddot{\frown}) = p(\theta=\ddot{\frown})N = 100$	$\text{freq}(\theta=\ddot{\smile}) = p(\theta=\ddot{\smile})N = 99,900$	$N = 100,000$

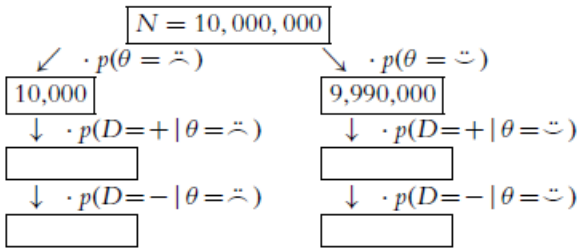
Notice the frequencies on the lower margin of the table. They indicate that out of 100,000 people, only 100 have the disease, while 99,900 do not have the disease. These marginal frequencies instantiate the prior probability that $p(\theta = \ddot{\frown}) = 0.001$. Notice also the cell frequencies in the column $\theta = \ddot{\frown}$, which indicate that of 100 people with the disease, 99 have a positive test result and 1 has a negative test result. These cell frequencies instantiate the hit rate of 0.99. Your job for this part of the exercise is to fill in the frequencies of the remaining cells of the table.

	θ = ☹	θ = ☺	
D = +	0.99 * 0.001 * 100,000 = 99	0.05 * (1.0-0.001) * 100,000 = 4,995	5,094
D = -	(1.0-0.99) * 0.001 * 100,000 = 1	(1.0-0.05) * (1.0-0.001) * 100,000 = 94,905	94,906
	0.001 * 100,000 = 100	(1.0-0.001) * 100,000 = 99,900	100,000

(B) Take a good look at the frequencies in the table you just computed for the previous part. These are the so-called "natural frequencies" of the events, as opposed to the somewhat unintuitive expression in terms of conditional probabilities (Gigerenzer & Hoffrage, 1995). From the cell frequencies alone, determine the proportion of people who have the disease, given that their test result is positive. Before computing the exact answer arithmetically, first give a rough intuitive answer merely by looking at the relative frequencies in the row $D = +$. Does your intuitive answer match the intuitive answer you provided when originally reading about Table 5.4? Probably not. Your intuitive answer here is probably much closer to the correct answer. Now compute the exact answer arithmetically. It should match the result from applying Bayes' rule in Table 5.4.

From the row D=+, we see 99 of 5,094 people have the disease. For a rough answer, that's about 200 of 10,000 = 2%, which is far from the hit rate of 99%. Calculating more exactly, 99/5,094 = 0.01943463, which matches what Bayes' rule provided.

(C) Now we'll consider a related representation of the probabilities in terms of natural frequencies, which is especially useful when we accumulate more data. This type of representation is called a "Markov" representation by Krauss, Martignon, and Hoffrage (1999). Suppose now we start with a population of $N = 10,000,000$ people. We expect 99.9% of them (i.e., 9,990,000) not to have the disease, and just 0.1% (i.e., 10,000) to have the disease. Now consider how many people we expect to test positive. Of the 10,000 people who have the disease, 99%, (i.e., 9,900) will be expected to test positive. Of the 9,990,000 people who do not have the disease, 5% (i.e., 499,500) will be expected to test positive. Now consider re-testing everyone who has tested positive on the first test. How many of them are expected to show a negative result on the retest? Use this diagram to compute your answer:



When computing the frequencies for the empty boxes above, be careful to use the proper conditional probabilities!

Down the left branch of the tree:	Down the right branch of the tree:
10,000 * 0.99 = 9,900	9,990,000 * 0.05 = 499,500
9,900 * (1.0-0.99) = 99	499,500 * (1.0-0.05) = 474,525

(D) Use the diagram in the previous part to answer this: What proportion of people, who test positive at first and then negative on retest, actually have the disease? In other words, of the total number of people at the bottom of the diagram in the previous part (those are the people who tested positive then negative), what proportion of them are in the left branch of the tree? *How does the result compare with your answer to Exercise 5.1?*

The proportion in the left branch of the tree is 99/(99+474,525) = 0.0002085862, which matches the result of Exercise 5.1.

Exercise 5.3.

Exercise 5.3. [Purpose: To see a hands-on example of data-order invariance.] Consider again the disease and diagnostic test of the previous two exercises.

(A) Suppose that a person selected at random from the population gets the test and it comes back negative. Compute the probability that the person has the disease.

(B) The person then gets re-tested, and on the second test the result is positive. Compute the probability that the person has the disease. *How does the result compare with your answer to Exercise 5.1?*

```
> # Specify hit rate of test:
> pPositiveGivenDisease = 0.99
> # Specify false alarm rate of test:
> pPositiveGivenNoDisease = 0.05
>
> # Specify the original prior:
> pDisease = 0.001
>
> # Bayes rule for first, NEGATIVE test:
> pDiseaseGivenNegative = ( (1.0-pPositiveGivenDisease) * pDisease /
+                             ( (1.0-pPositiveGivenDisease) * pDisease
+                               + (1.0-pPositiveGivenNoDisease) * (1.0-pDisease) ) )
> show( pDiseaseGivenNegative )
[1] 1.053674e-05
>
> # Set the prior to the new probability of having the disease:
> pDisease = pDiseaseGivenNegative
>
> # Bayes rule for second, POSITIVE test:
> pDiseaseGivenPositive = ( pPositiveGivenDisease * pDisease /
+                             ( pPositiveGivenDisease * pDisease
+                               + pPositiveGivenNoDisease * (1.0-pDisease) ) )
> show( pDiseaseGivenPositive )
[1] 0.0002085862
```

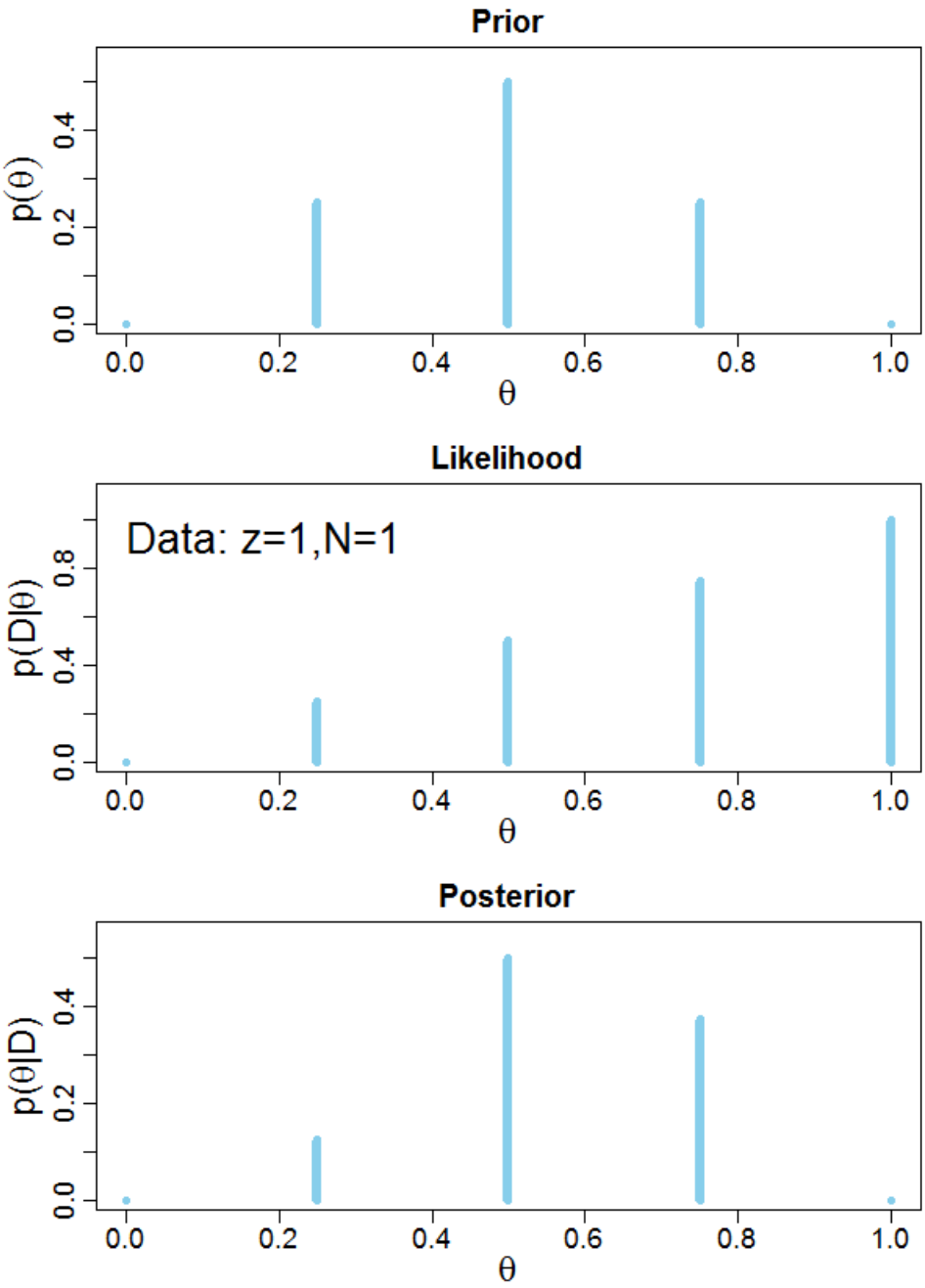
Notice that the final probability of having the disease is that same as in Exercise 5.1.

Exercise 5.4.

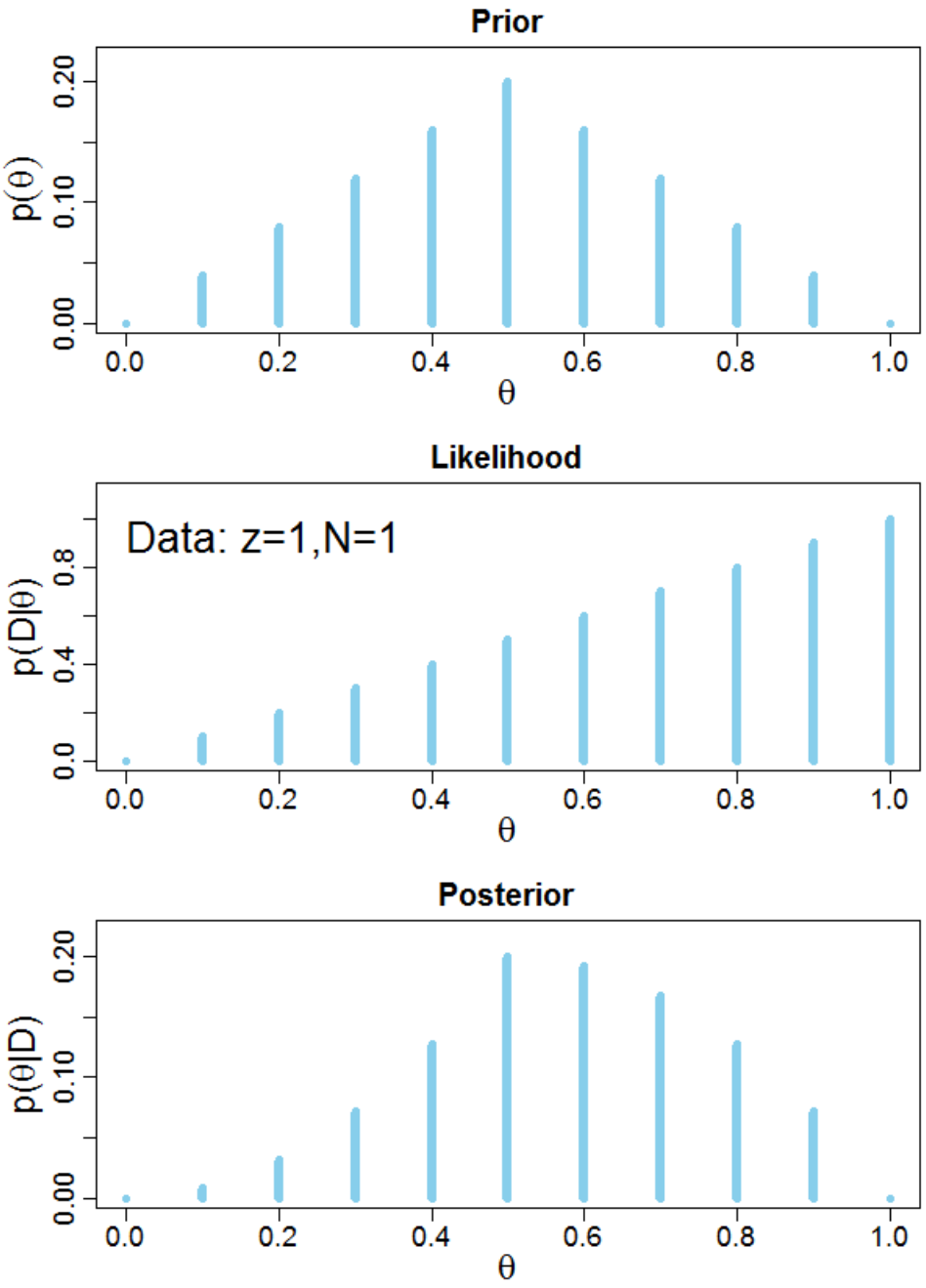
Exercise 5.4. [Purpose: To gain intuition about Bayesian updating by using `BernGrid`.] Open the program `BernGridExample.R`. You will notice there are several examples of using the function `BernGrid`. Run the script. For each example, include the R code and the resulting graphic and explain what idea the example illustrates. Hints: Look back at Figures 5.2 and 5.3, and look ahead to Figure 6.5. Two of the examples involve a single flip, with the only difference between the examples being whether the prior is uniform or contains only two extreme options. The point of those two examples is to show that a single datum implies little when the prior is vague, but a single datum can have strong implications when the prior allows only two very different possibilities.

The script BernGridExample.R is included below, along with comments that describe each of the examples.

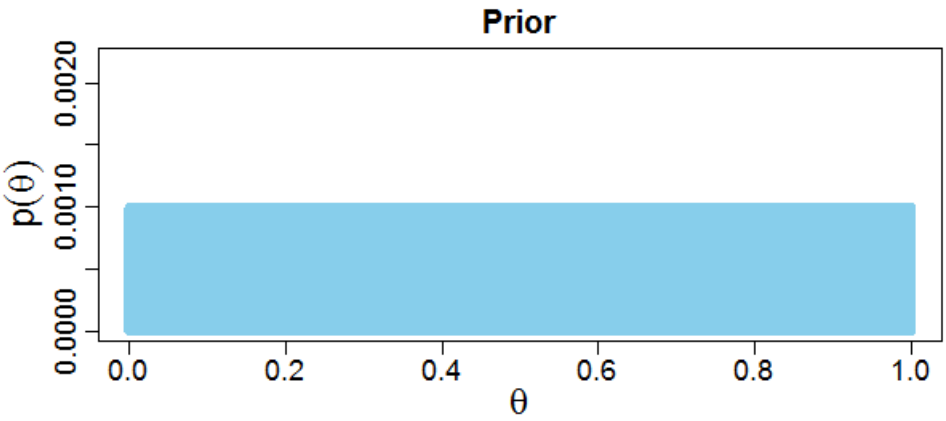
```
source("DBDA2E-utilities.R")
source("BernGrid.R")
```

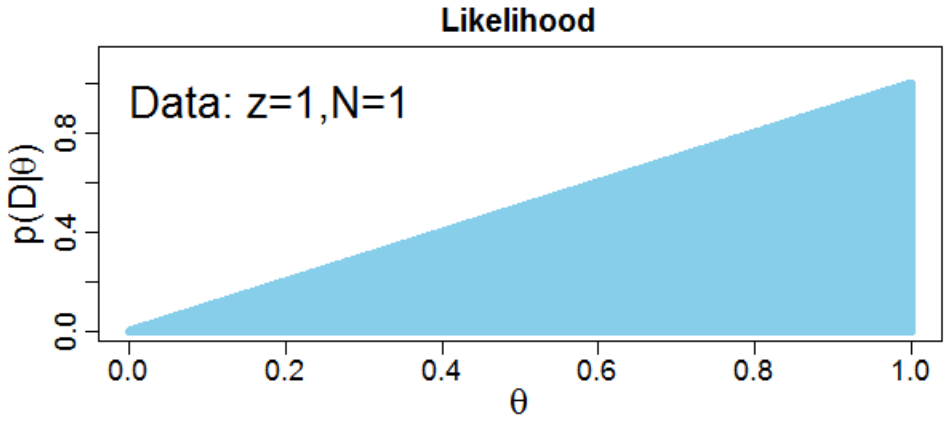


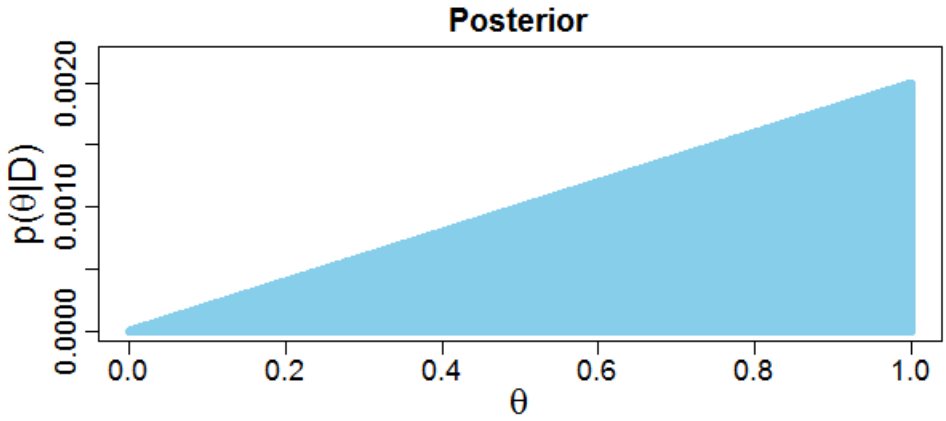

```
Theta = seq( 0 , 1 , length=5 )  # Sparse teeth for Theta.
pTheta = pmin( Theta , 1-Theta ) # Triangular shape for pTheta.
pTheta = pTheta/sum(pTheta)      # Make pTheta sum to 1.0
Data = c(rep(0,0),rep(1,1))      # Single flip with 1 head
openGraph(width=5,height=7)
posterior = BernGrid( Theta, pTheta , Data , plotType="Bars" ,
                      showCentTend="None" , showHDI=FALSE , showpD=FALSE )

# This just shows reallocation of credibility for a small number of candidate
parameter values and a single datum.
#------------------------------------------------------------------------------
```



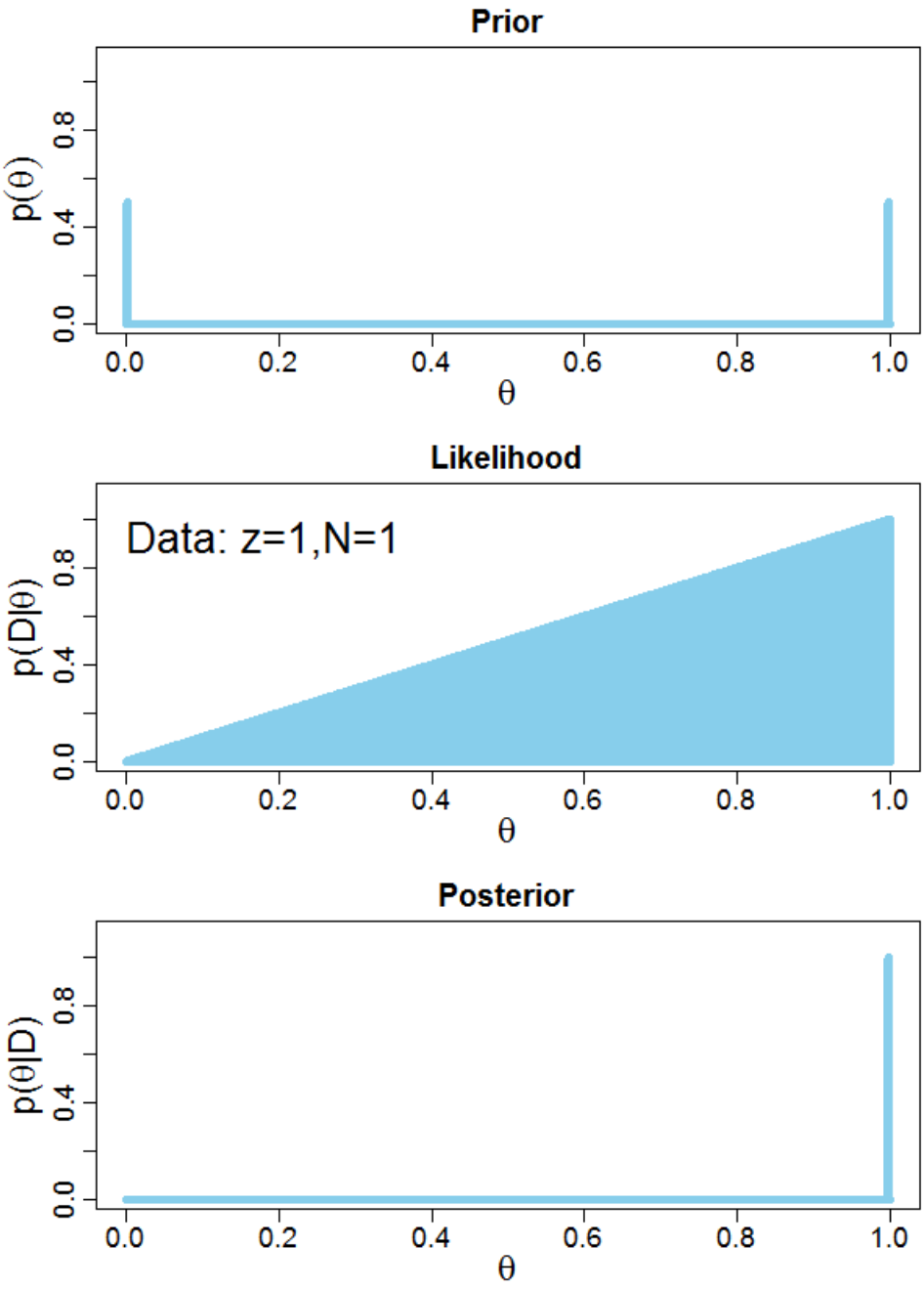

```
Theta = seq( 0 , 1 , length=11 )  # Sparse teeth for Theta.
pTheta = pmin( Theta , 1-Theta ) # Triangular shape for pTheta.
pTheta = pTheta/sum(pTheta)      # Make pTheta sum to 1.0
Data = c(rep(0,0),rep(1,1))      # Single flip with 1 head
openGraph(width=5,height=7)
posterior = BernGrid( Theta, pTheta , Data , plotType="Bars" ,
                      showCentTend="None" , showHDI=FALSE , showpD=FALSE )

# Ditto previous example.
#-------------------------------------------------------------------------------
```



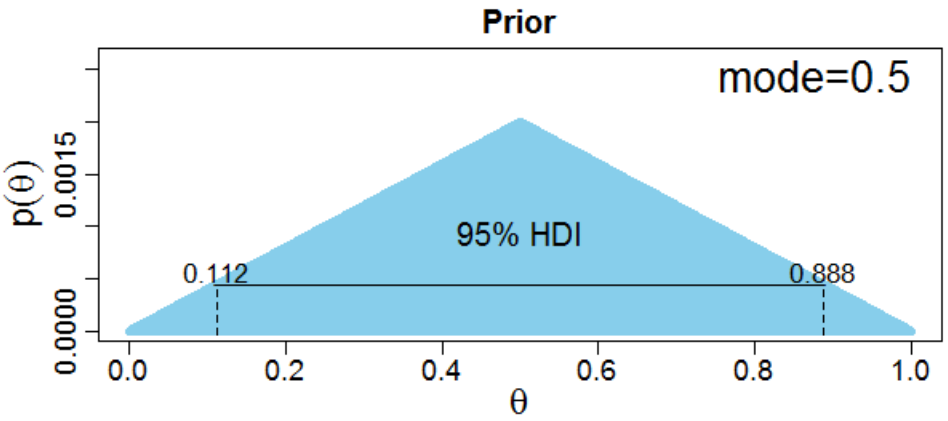


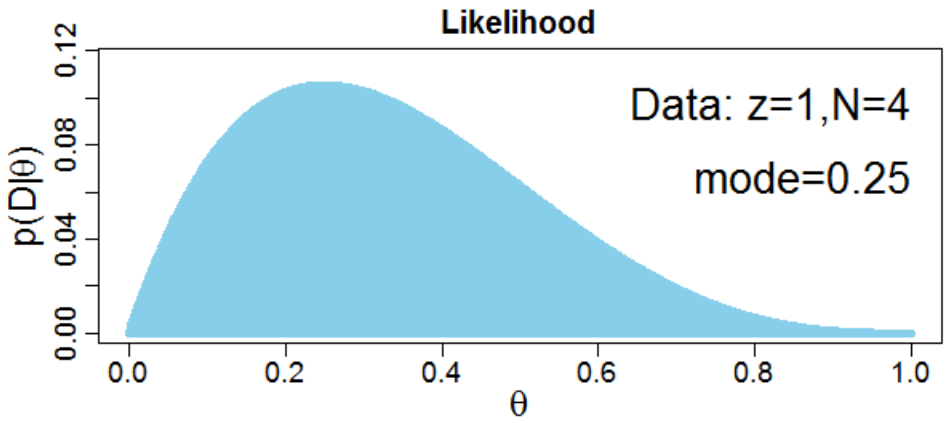


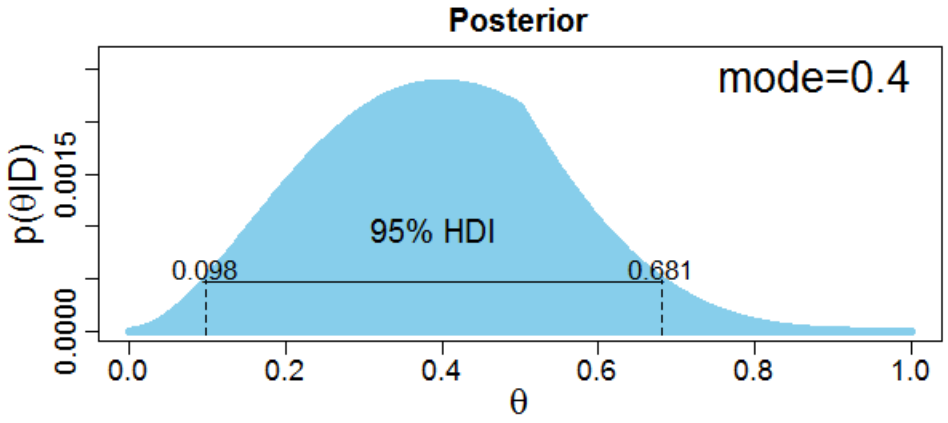

```
Theta = seq( 0 , 1 , length=1001 ) # Fine teeth for Theta.
pTheta = rep(1,length(Theta))      # Uniform (horizontal) shape for pTheta.
pTheta = pTheta/sum(pTheta)        # Make pTheta sum to 1.0
Data = c(rep(0,0),rep(1,1))        # Single flip with 1 head
openGraph(width=5,height=7)
posterior = BernGrid( Theta, pTheta , Data , plotType="Bars" ,
                      showCentTend="None" , showHDI=FALSE , showpD=FALSE )

# Posterior looks like likelihood when prior is uniform; also builds intuition about
what the Bernoulli likelihood function looks like.
#------------------------------------------------------------------------------
```



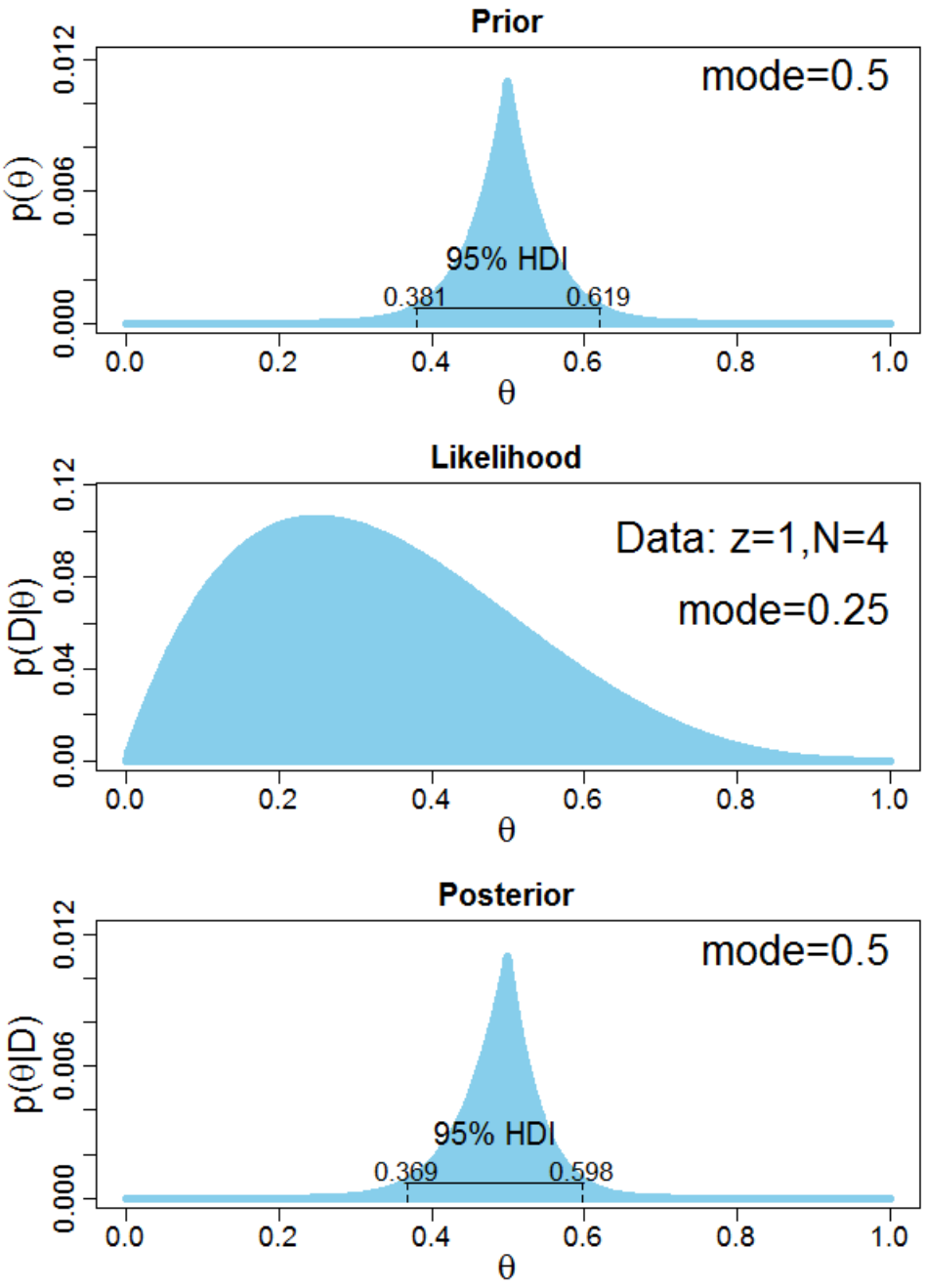

```
Theta = seq( 0 , 1 , length=1001 ) # Fine teeth for Theta.
pTheta = rep(0,length(Theta))      # Only extremes are possible!
pTheta[2] = 1                      # Only extremes are possible!
pTheta[length(pTheta)-1] = 1
pTheta = pTheta/sum(pTheta)        # Make pTheta sum to 1.0
Data = c(rep(0,0),rep(1,1))        # Single flip with 1 head
openGraph(width=5,height=7)
posterior = BernGrid( Theta, pTheta , Data , plotType="Bars" ,
                      showCentTend="None" , showHDI=FALSE , showpD=FALSE )

# This example is interesting because the prior allows only two possible parameter
values. (The floor between extremes has zero height!) Then a single flip of the coin
shifts all probability to one candidate parameter value.
#------------------------------------------------------------------------------
```



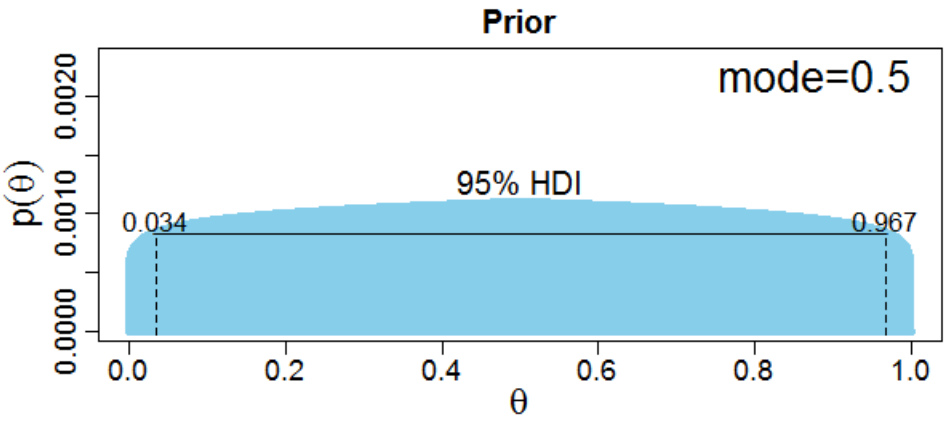


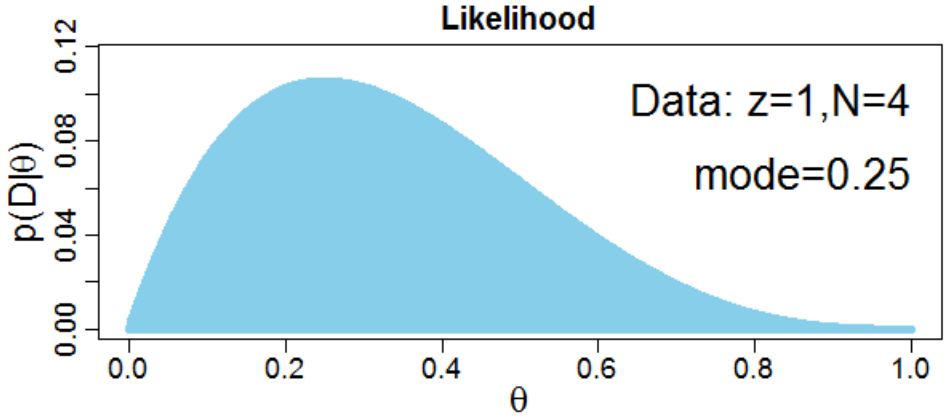


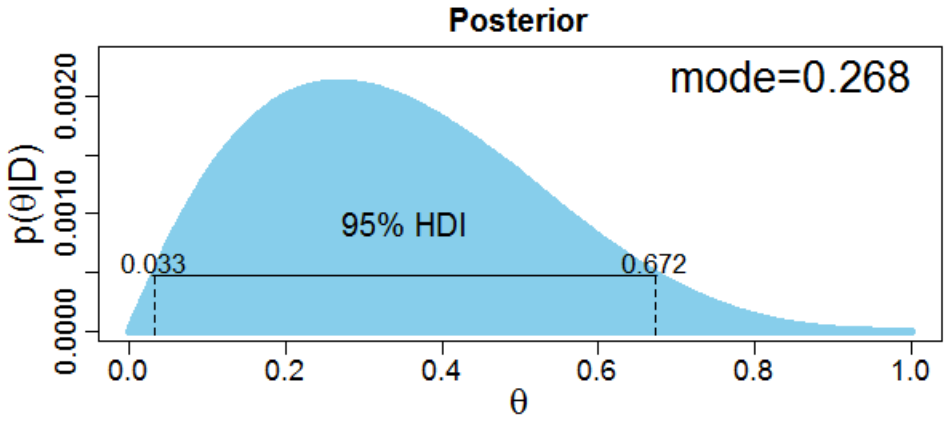

```
Theta = seq( 0 , 1 , length=1001 )  # Fine teeth for Theta.
pTheta = pmin( Theta , 1-Theta ) # Triangular shape for pTheta.
pTheta = pTheta/sum(pTheta)      # Make pTheta sum to 1.0
Data = c(rep(0,3),rep(1,1))      # 25% heads, N=4
openGraph(width=5,height=7)
posterior = BernGrid( Theta, pTheta , Data , plotType="Bars" ,
                      showCentTend="Mode" , showHDI=TRUE , showpD=FALSE )

# Shows that a small data set only shifts beliefs a little; the prior retains
noticeable influence on the posterior.
#------------------------------------------------------------------------------
```



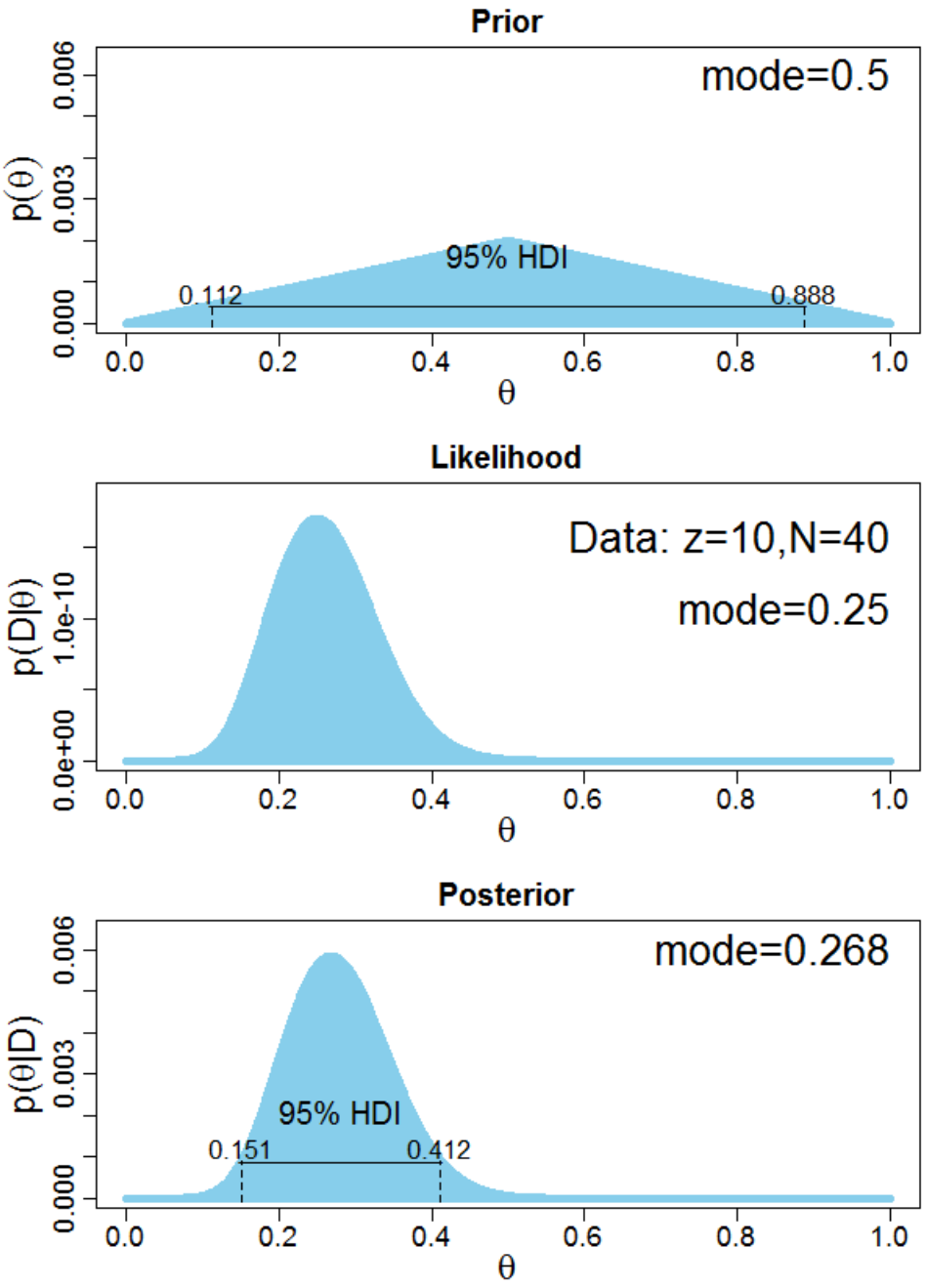

```
Theta = seq( 0 , 1 , length=1001 )  # Fine teeth for Theta.
pTheta = pmin( Theta , 1-Theta ) # Triangular shape for pTheta.
pTheta = pTheta/sum(pTheta)      # Make pTheta sum to 1.0
pTheta = pTheta^10               # Sharpen pTheta !
pTheta = pTheta/sum(pTheta)      # Make pTheta sum to 1.0
Data = c(rep(0,3),rep(1,1))      # 25% heads, N=4
openGraph(width=5,height=7)
posterior = BernGrid( Theta, pTheta , Data , plotType="Bars" ,
                      showCentTend="Mode" , showHDI=TRUE , showpD=FALSE )

# A small data set with a strong prior: The prior dominates the posterior.
#-------------------------------------------------------------------------------
```



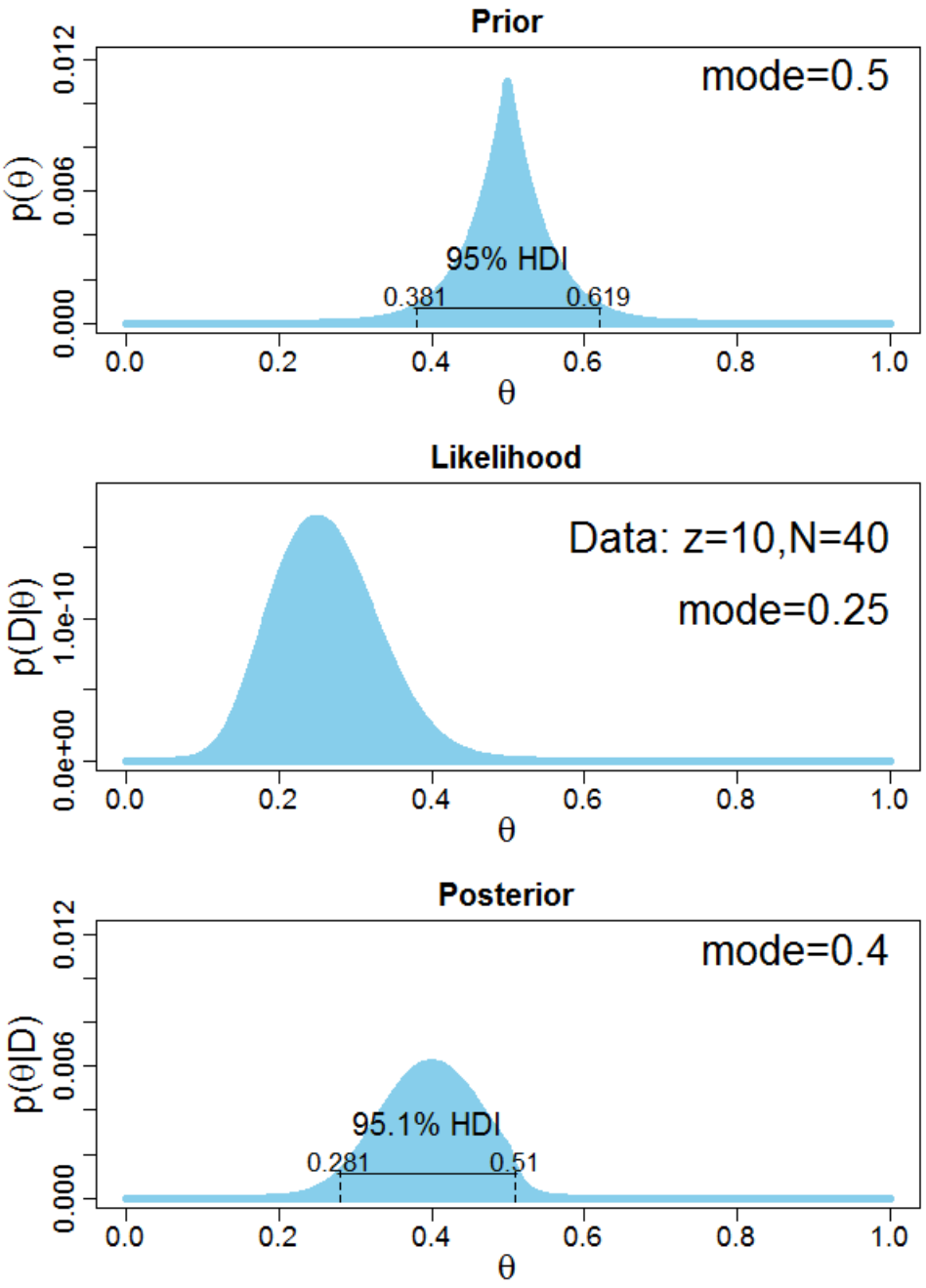




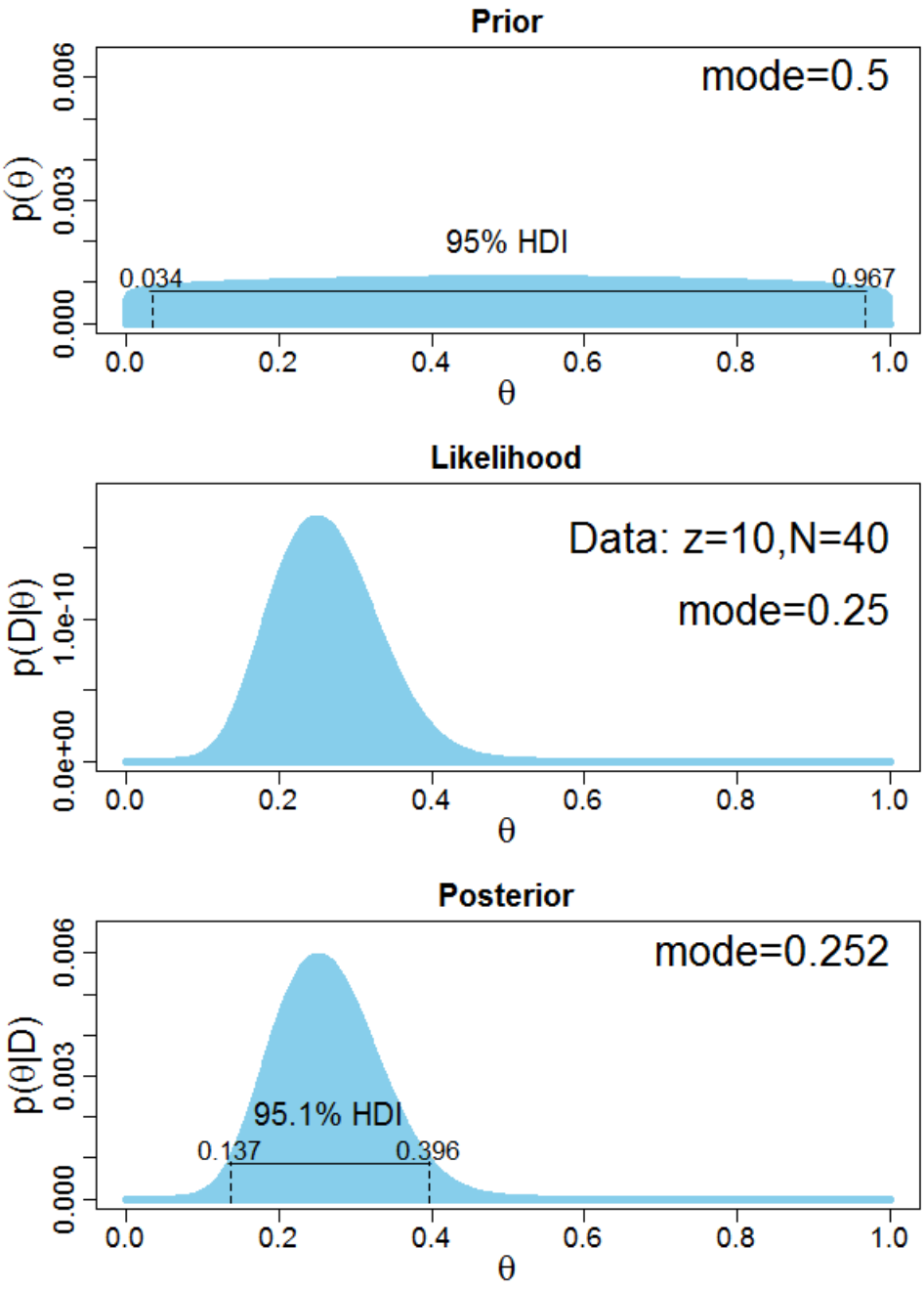

```
Theta = seq( 0 , 1 , length=1001 )  # Fine teeth for Theta.
pTheta = pmin( Theta , 1-Theta ) # Triangular shape for pTheta.
pTheta = pTheta/sum(pTheta)      # Make pTheta sum to 1.0
pTheta = pTheta^0.1              # Flatten pTheta !
pTheta = pTheta/sum(pTheta)      # Make pTheta sum to 1.0
Data = c(rep(0,3),rep(1,1))      # 25% heads, N=4
openGraph(width=5,height=7)
posterior = BernGrid( Theta, pTheta , Data , plotType="Bars" ,
                      showCentTend="Mode" , showHDI=TRUE , showpD=FALSE )

# A small data set with a vague prior: The data dominate the posterior.
#------------------------------------------------------------------------------
```



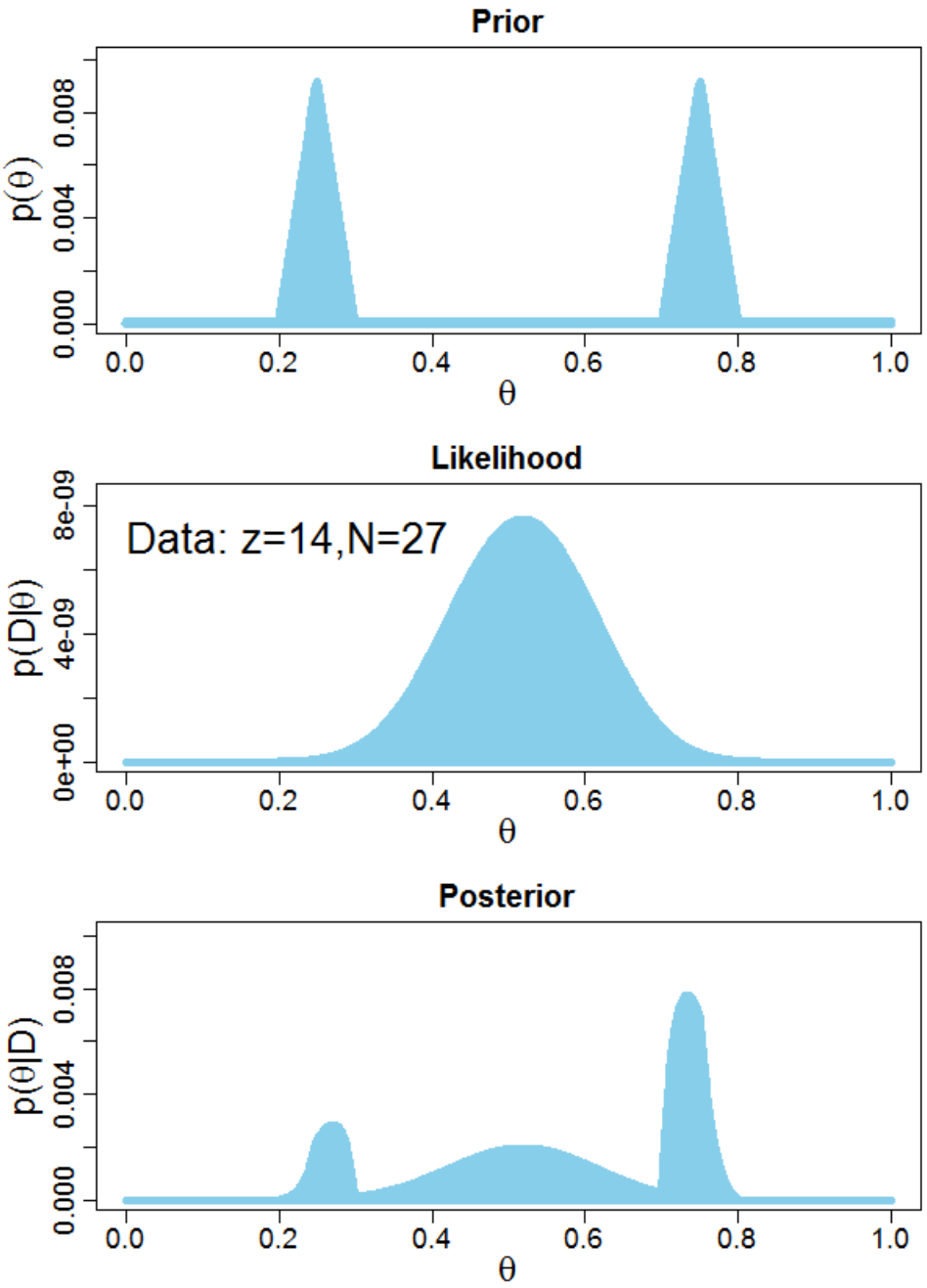

```
Theta = seq( 0 , 1 , length=1001 )  # Fine teeth for Theta.
pTheta = pmin( Theta , 1-Theta ) # Triangular shape for pTheta.
pTheta = pTheta/sum(pTheta)      # Make pTheta sum to 1.0
Data = c(rep(0,30),rep(1,10))    # 25% heads, N=40

openGraph(width=5,height=7)
posterior = BernGrid( Theta, pTheta , Data , plotType="Bars" ,
                      showCentTend="Mode" , showHDI=TRUE , showpD=FALSE )

# A modest amount of data with a moderately informed prior: The amount of data is
enough to mostly overwhelm the prior.
#------------------------------------------------------------------------------
```




```
Theta = seq( 0 , 1 , length=1001 )  # Fine teeth for Theta.
pTheta = pmin( Theta , 1-Theta ) # Triangular shape for pTheta.
pTheta = pTheta/sum(pTheta)      # Make pTheta sum to 1.0
pTheta = pTheta^10               # Sharpen pTheta !
pTheta = pTheta/sum(pTheta)      # Make pTheta sum to 1.0
Data = c(rep(0,30),rep(1,10))    # 25% heads, N=40
openGraph(width=5,height=7)
posterior = BernGrid( Theta, pTheta , Data , plotType="Bars" ,
                      showCentTend="Mode" , showHDI=TRUE , showpD=FALSE )

# A modest amount of data is not enough to overwhelm a strong prior. The posterior is
a compromise between prior and data. This is appropriate, because the prior must have
been informed by previous data/knowledge, so should not be abandoned easily.
#------------------------------------------------------------------------------
```




```
Theta = seq( 0 , 1 , length=1001 )  # Fine teeth for Theta.
pTheta = pmin( Theta , 1-Theta ) # Triangular shape for pTheta.
pTheta = pTheta/sum(pTheta)      # Make pTheta sum to 1.0
pTheta = pTheta^0.1              # Flatten pTheta !
pTheta = pTheta/sum(pTheta)      # Make pTheta sum to 1.0
Data = c(rep(0,30),rep(1,10))    # 25% heads, N=40
openGraph(width=5,height=7)
posterior = BernGrid( Theta, pTheta , Data , plotType="Bars" ,
                      showCentTend="Mode" , showHDI=TRUE , showpD=FALSE )

# Vague prior has little influence on posterior.
#------------------------------------------------------------------------------
```




```
Theta = seq( 0 , 1 , length=1000 )  # Fine teeth for Theta.
# Two triangular peaks on a small non-zero floor:
pTheta = c( rep(1,200),seq(1,100,length=50),seq(100,1,length=50),rep(1,200) ,
            rep(1,200),seq(1,100,length=50),seq(100,1,length=50),rep(1,200) )
pTheta = pTheta/sum(pTheta)      # Make pTheta sum to 1.0
Data = c(rep(0,13),rep(1,14))
openGraph(width=5,height=7)
posterior = BernGrid( Theta, pTheta , Data , plotType="Bars" ,
                      showCentTend="None" , showHDI=FALSE , showpD=FALSE )

# Unusual prior can only be expressed by grid approximation. Notice floor is non-zero.
The posterior is compromise between prior and likelihood because of modest amount of
data.
#------------------------------------------------------------------------------
```

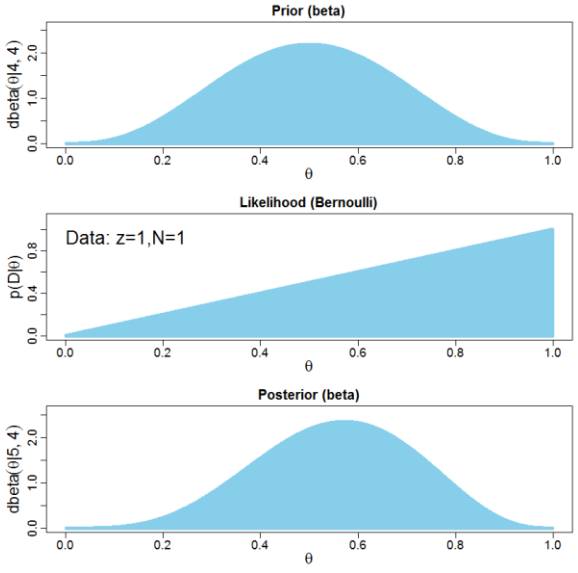
Chapter 6

Exercise 6.1

Exercise 6.1. [Purpose: For you to see the influence of the prior in each successive flip, and for you to see another demonstration that the posterior is invariant under re-orderings of the data.] For this exercise, use the R function explained in Section 6.6 (`BernBeta.R`). (Don't forget to `source` the function before calling it.) Notice that the function returns the posterior beta values each time it is called, so you can use the returned values as the prior values for the next function call.

(A) Start with a prior distribution that expresses some uncertainty that a coin is fair: beta(θ|4, 4). Flip the coin once; suppose we get a head. What is the posterior distribution?

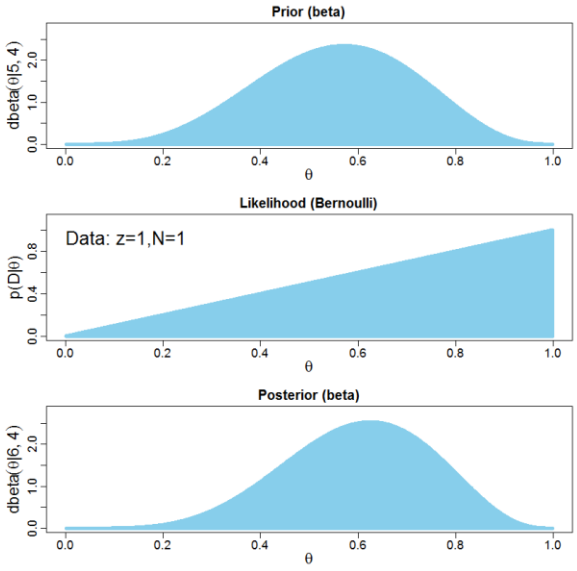
```
source("DBDA2E-utilities.R")  # Load definitions of graphics functions etc.
source("BernBeta.R")          # Load the definition of the BernBeta function
openGraph()

# Exercise 6.1A:
post = BernBeta( priorBetaAB=c(4,4) , Data=c(1) )
```



The posterior is dbeta(theta|5,4); see the label on the y-axis of the posterior distribution or type show(post) or just use the simple formula for updating a beta distribution.

(B) Use the posterior from the previous flip as the prior for the next flip. Suppose we flip again and get a head. Now what is the new posterior? (*Hint:* If you type `post = BernBeta( c(4,4) , c(1) )` for the first part, then you can type `post = BernBeta( post , c(1) )` for the next part.)

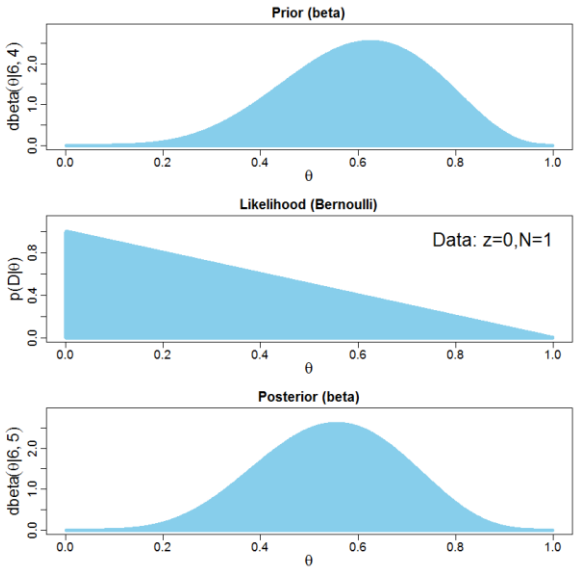
```
# Exercise 6.1B:
post = BernBeta( priorBetaAB=post , Data=c(1) )
```



The posterior is dbeta(theta|6,4).

(C) Using that posterior as the prior for the next flip, flip a third time and get a tail. Now what is the new posterior? (*Hint:* Type `post = BernBeta( post , c(0) )`.)

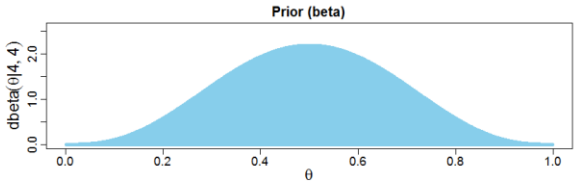
```
# Exercise 6.1C:
post = BernBeta( priorBetaAB=post , Data=c(0) )
```

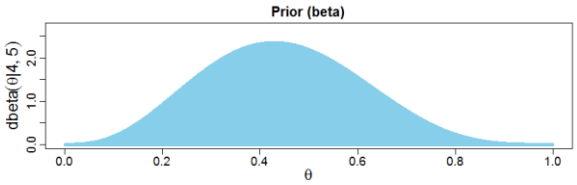


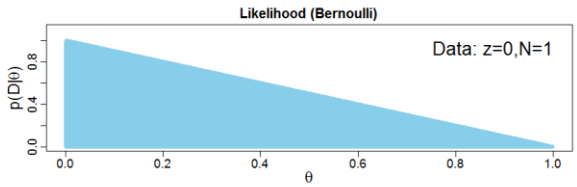
The posterior is dbeta(theta|6,5).

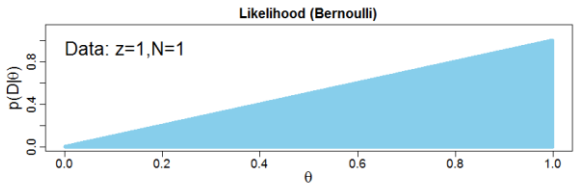
(D) Do the same three updates but in the order T, H, H instead of H, H, T. Is the final posterior distribution the same for both orderings of the flip results?

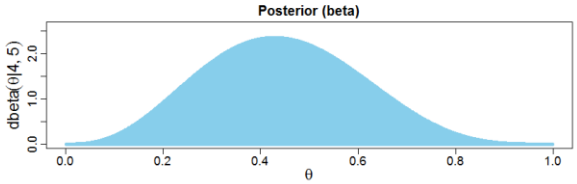
```
# Exercise 6.1D:
post = BernBeta( priorBetaAB=c(4,4) , Data=c(0) )
post = BernBeta( priorBetaAB=post , Data=c(1) )
post = BernBeta( priorBetaAB=post , Data=c(1) )
```

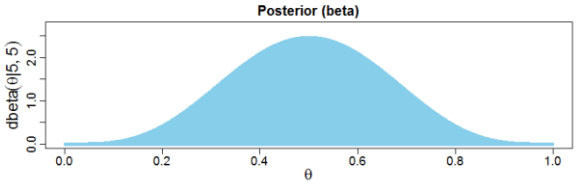


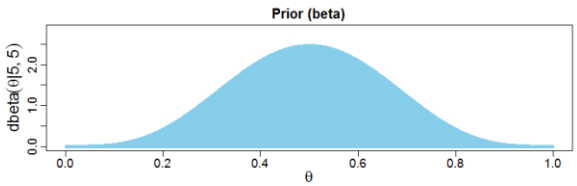


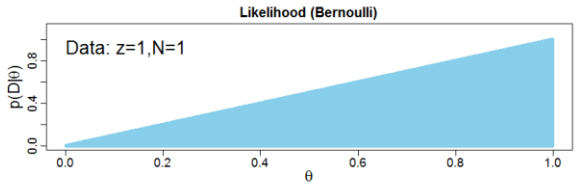


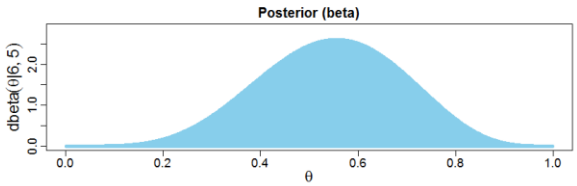












The final posterior is again dbeta(theta|6,5).

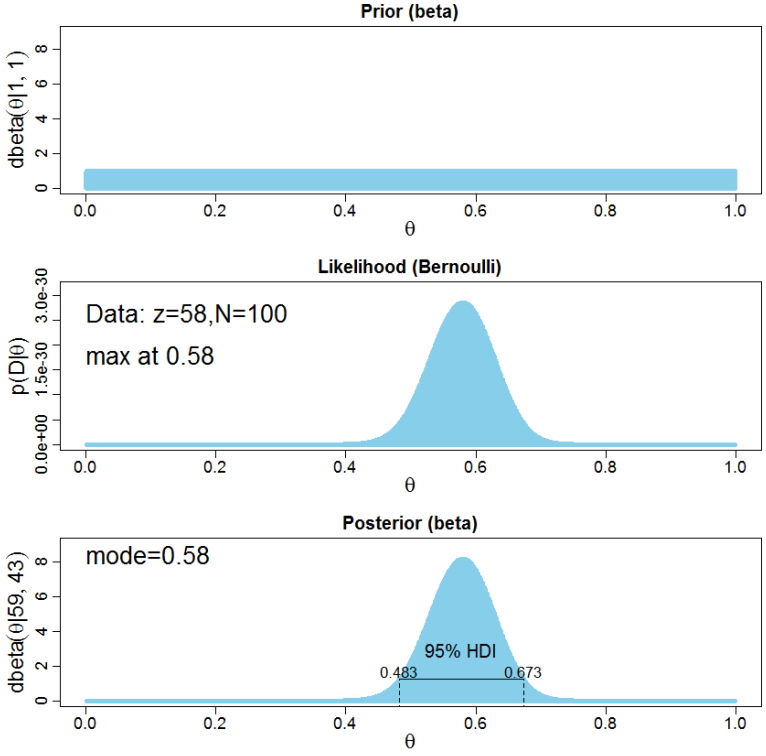
Exercise 6.2

Exercise 6.2. [Purpose: Connecting HDIs to the real world, with iterative data collection.] Suppose an election is approaching, and you are interested in knowing whether the general population prefers candidate A or candidate B. There is a just-published poll in the newspaper, which states that of 100 randomly sampled people, 58 preferred candidate A and the remainder preferred candidate B.

(A) Suppose that before the newspaper poll, your prior belief was a uniform distribution. What is the 95% HDI on your beliefs after learning of the newspaper poll results?

The R code below assumes candidate A is a "1" and candidate B is a "0".

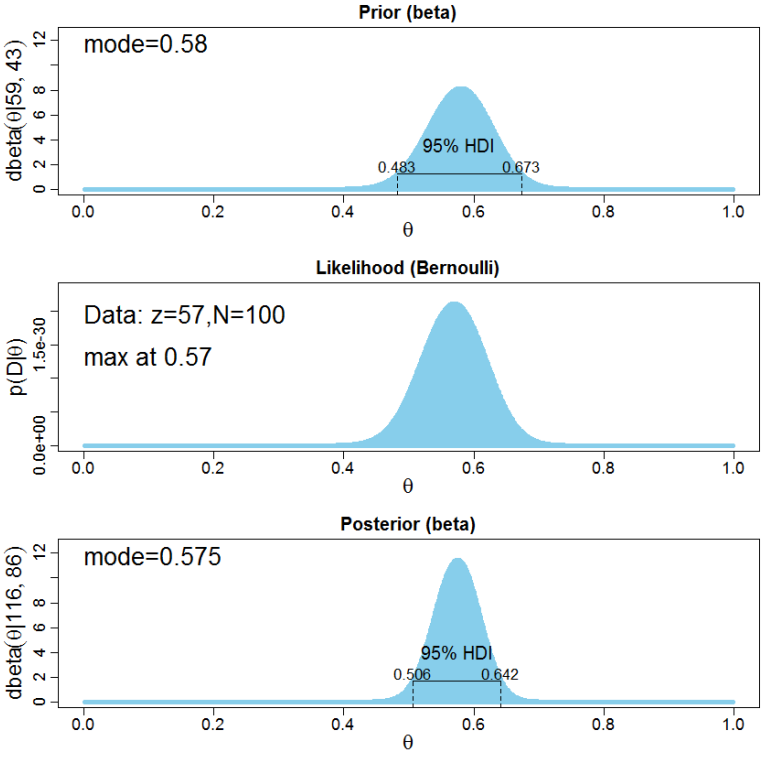
```
source("DBDA2E-utilities.R")  # Load definitions of graphics functions etc.
source("BernBeta.R")          # Load the definition of the BernBeta function
openGraph()

# Exercise 6.2A:
post = BernBeta( priorBetaAB=c(1,1) , Data=c(rep(1,58),rep(0,100-58)) ,
                 showHDI=TRUE , showCentTend="Mode" )
```



The 95% HDI is 0.483 to 0.673 for probability of candidate A.

(B) You want to conduct a follow-up poll to narrow down your estimate of the population's preference. In your follow-up poll, you randomly sample 100 other people and find that 57 prefer candidate A and the remainder prefer candidate B. Assuming that peoples' opinions have not changed between polls, what is the 95% HDI on the posterior?

```
# Exercise 6.2B:
post = BernBeta( priorBetaAB=post , Data=c(rep(1,57),rep(0,100-57)) ,
                 showHDI=TRUE , showCentTend="Mode" )
```

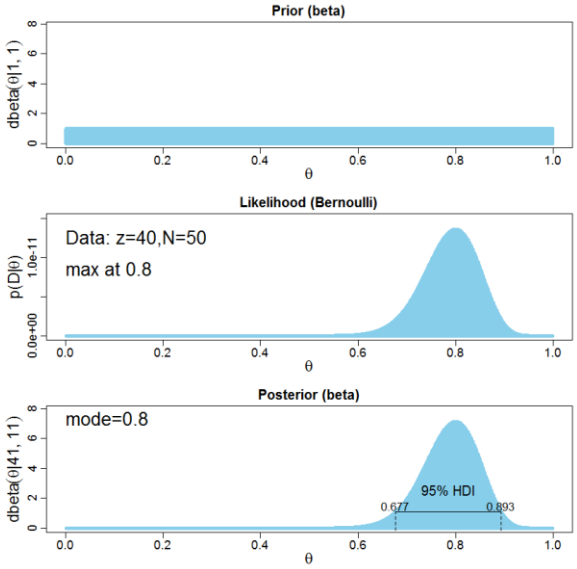


The 95% HDI is now narrower, from 0.506 to 0.642, for probability of candidate A.

Exercise 6.3

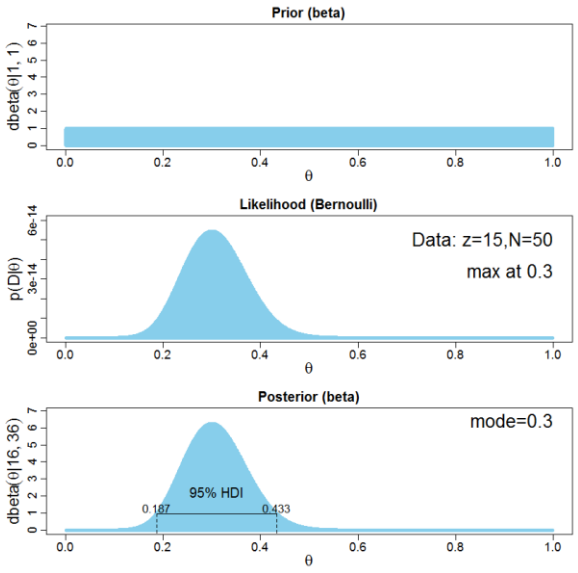
Exercise 6.3. [Purpose: Apply the Bayesian method to real data analysis. These data are representative of real data (Kruschke, 2009).] Suppose you train people in a simple learning experiment, as follows. When people see the two words, "radio" and "ocean," on the computer screen, they should press the F key on the computer keyboard. They see several repetitions and learn the response well. Then you introduce another correspondence for them to learn: Whenever the words "radio" and "mountain" appear, they should press the J key on the computer keyboard. You keep training them until they know both correspondences well. Now you probe what they've learned by asking them about two novel test items. For the first test, you show them the word "radio" by itself and instruct them to make the best response (F or J) based on what they learned before. For the second test, you show them the two words "ocean" and "mountain" and ask them to make the best response. You do this procedure with 50 people. Your data show that for "radio" by itself, 40 people chose F and 10 chose J. For the word combination "ocean" and "mountain," 15 chose F and 35 chose J. Are people biased toward F or toward J for either of the two probe types? To answer this question, assume a uniform prior, and use a 95% HDI to decide which biases can be declared to be credible. (Consult Chapter 12 for how to declare a parameter value to be not credible.)

```
source("DBDA2E-utilities.R")  # Load definitions of graphics functions etc.
source("BernBeta.R")          # Load the definition of the BernBeta function
openGraph()

# Response F is y=1, response J is y=0.
# "radio":
post = BernBeta( priorBetaAB=c(1,1) , Data=c(rep(1,40),rep(0,10)) ,
                 showHDI=TRUE , showCentTend="Mode" )
```



The posterior 95% HDI goes from 0.677 to 0.893, which excludes any reasonable ROPE around theta=0.5. In other words, we reject the null value of theta=0.5 because the 95% most credible values are all not practically equivalent to theta=0.5.

```
# "ocean":
post = BernBeta( priorBetaAB=c(1,1) , Data=c(rep(1,15),rep(0,35)) ,
                 showHDI=TRUE , showCentTend="Mode" )
```



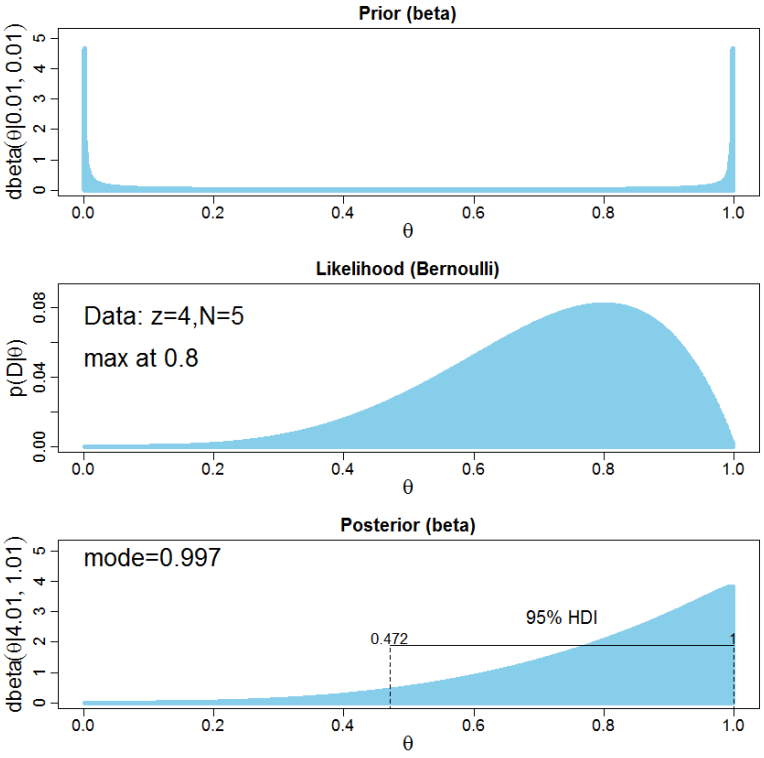
The posterior 95% HDI goes from 0.187 to 0.433, which excludes any reasonable ROPE around theta=0.5. In other words, we reject the null value of theta=0.5 because the 95% most credible values are all not practically equivalent to theta=0.5.

Exercise 6.4

Exercise 6.4. [Purpose: To explore an unusual prior and learn about the beta distribution in the process.] Suppose we have a coin that we know comes from a magic-trick store, and therefore we believe that the coin is strongly biased either usually to come up heads or usually to come up tails, but we don't know which. Express this belief as a beta prior. (*Hint:* See Figure 6.1, upper-left panel.) Now we flip the coin 5 times and it comes up heads in 4 of the 5 flips. What is the posterior distribution? (Use the R function of Section 6.6 (`BernBeta.R`) to see graphs of the prior and posterior.)

```
source("DBDA2E-utilities.R")  # Load definitions of graphics functions etc.
source("BernBeta.R")          # Load the definition of the BernBeta function
openGraph()

# Exercise 6.4:
post = BernBeta( priorBetaAB=c(1,1)/100 , Data=c(rep(1,4),rep(0,1)) ,
                 showHDI=TRUE , showCentTend="Mode" )
```

Notice the code above uses a prior of dbeta(theta|0.01,0.01), coded as c(1,1)/100. This makes the graph look better than c(1,1)/10, but the result is qualitatively similar.



The posterior distribution is dbeta(theta|4.01,1.01) which, as shown above, has its mode essentially at 1.0, not at 0.8.

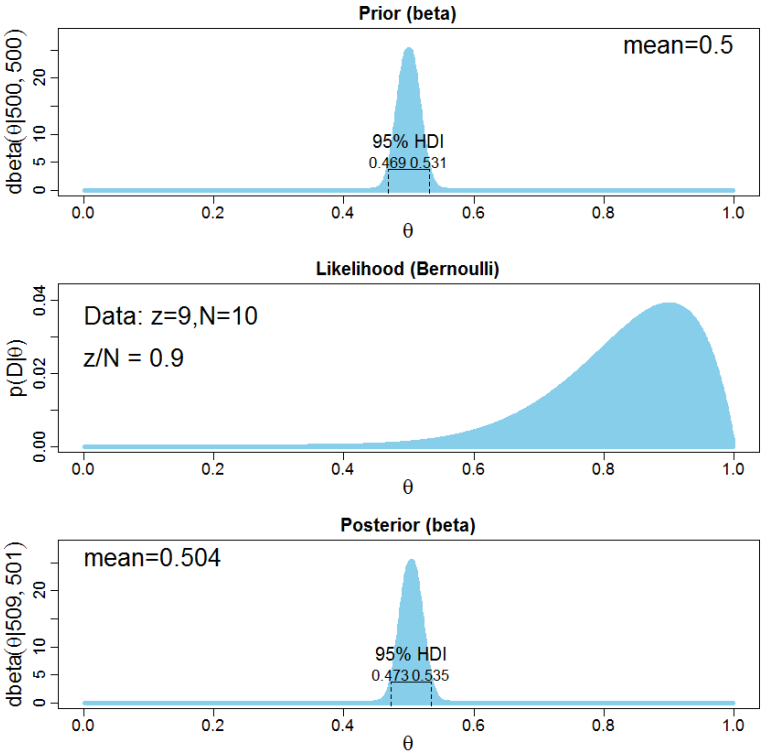
Exercise 6.5

Exercise 6.5. [Purpose: To get hands on experience with the goal of predicting the next datum, and to see how the prior influences that prediction.]

(A) Suppose you have a coin that you know is minted by the government and has not been tampered with. Therefore you have a strong prior belief that the coin is fair. You flip the coin 10 times and get 9 heads. What is your predicted probability of heads for the 11th flip? Explain your answer carefully; justify your choice of prior.

For the strong prior belief, let's suppose it's equivalent to previously seeing 1,000 flips of the coin and observing 50% (i.e., 500) heads. So the prior is dbeta(theta|500,500). You could use something different, but it should be sharply peaked at theta=0.5.

```
source("DBDA2E-utilities.R")  # Load definitions of graphics functions etc.
source("BernBeta.R")          # Load the definition of the BernBeta function
openGraph()
post = BernBeta( priorBetaAB=c(1,1)*500 , Data=c(rep(1,9),rep(0,1)) ,
                 showHDI=TRUE , showCentTend="Mean" )
```

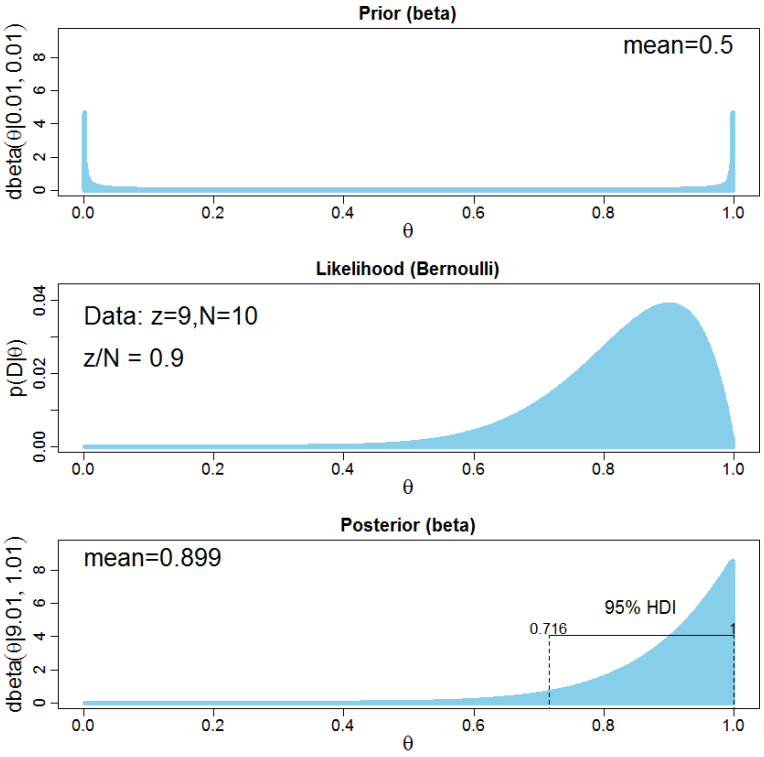


The predicted probability of heads on the next flip is the mean of the posterior distribution, which in this case is 0.504 (as displayed in the graph). Thus, even though the new data showed 90% heads, the strong prior keeps the posterior prediction very nearly 50%.

(B) Now you have a different coin, this one made of some strange material and marked (in fine print) "Patent Pending, International Magic, Inc." You flip the coin 10 times and get 9 heads. What is your predicted probability of heads for the 11th flip? Explain your answer carefully; justify your choice of prior. *Hint:* Use the prior from Exercise 6.4.

I used the dbeta(theta|0.01,0.01) prior from Exercise 6.4, to express a strong prior belief that the coin is strongly heads biased or strongly tails biased.

```
post = BernBeta( priorBetaAB=c(1,1)/100 , Data=c(rep(1,9),rep(0,1)) ,
                 showHDI=TRUE , showCentTend="Mean" )
```



The predicted probability of heads on the next flip is the mean of the posterior distribution, which is 0.899 (as displayed in the graph).

Chapter 7

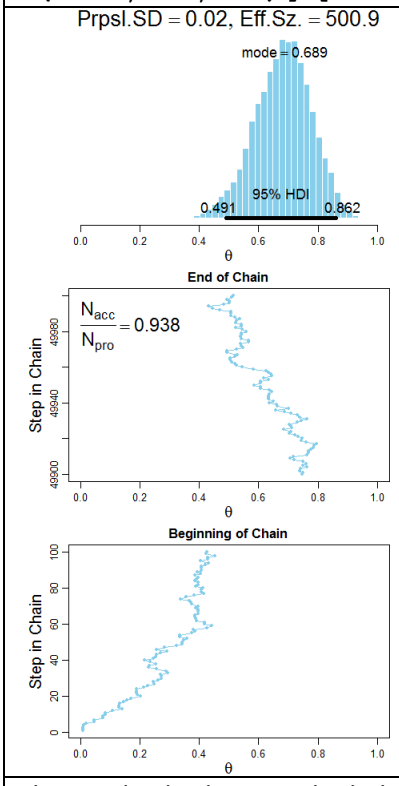
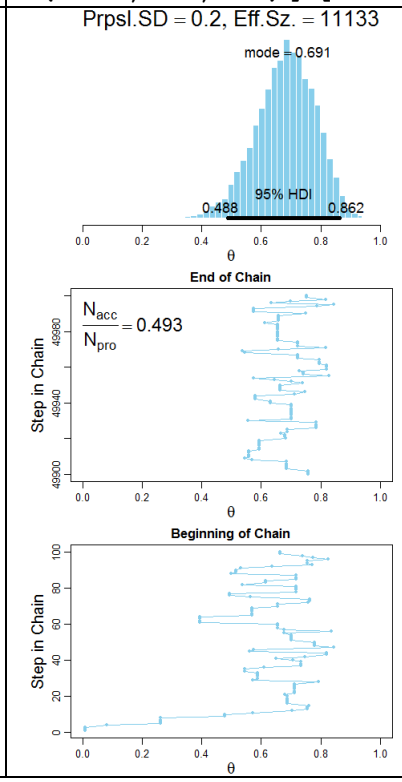
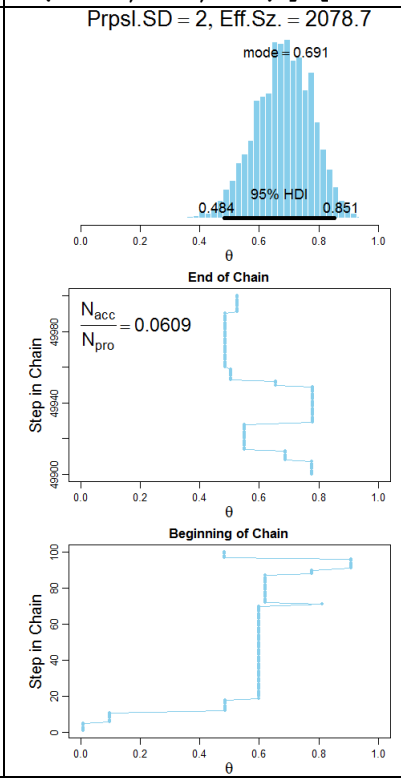
Exercise 7.1

Exercise 7.1. [Purpose: Experiment with the Metropolis algorithm as displayed in Figure 7.4.] Open the program named `BernMetrop.R` from the files that accompany this book. The script implements a Metropolis algorithm for Figure 7.4 Midway through the script, you will find a line that specifies the SD of the proposal distribution:

```
proposalSD = c(0.02,0.2,2.0)[2]
```

The line may look strange but it's merely a vector of constants with an index at the end to specify which component should be used. Thus, it's a simple way of specifying three options and then selecting one option. Run the script three times, once with each option (i.e., once with `[1]`, once with `[2]`, and once with `[3]`). *There is also a line that specifies the seed for the random number generator; comment it out so that you get a different trajectory than the ones shown in Figure 7.4.* Notice at the end of the script that you can specify the format of the graphic file for saving the resulting graph. Include the graphs in your write-up and describe whether they show similar behavior as the corresponding trajectories in Figure 7.4. Be sure to discuss the ESS.

Comment out set.seed(): **`# set.seed(47405) # Hence your graphs will differ.`**

Set the proposal SD to 0.02: **`proposalSD = c(0.02,0.2,2.0)[1]`**	Set the proposal SD to 0.2: **`proposalSD = c(0.02,0.2,2.0)[2]`**	Set the proposal SD to 2.0: **`proposalSD = c(0.02,0.2,2.0)[3]`**
		

The graphs do show similar behavior to those in Figure 7.4: The acceptance ratio is very high for proposal SD=0.02, moderate for SD=0.2, and very low for SD=2.0. The ESS is best for proposal SD=0.2, but poor for SD=0.02 and for SD=2.0.

Exercise 7.2

Exercise 7.2. **[Purpose: To explore the autocorrelation function in Figure 7.12.]** At the end of the script BernMetrop.R, add these lines:

```
openGraph(height=7,width=3.5)
layout(matrix(1:2,nrow=2))
acf( acceptedTraj , lag.max=30 , col="skyblue" , lwd=3 )
Len = length( acceptedTraj )
Lag = 10
trajHead = acceptedTraj[  1      : (Len-Lag) ]
trajTail = acceptedTraj[ (1+Lag) :  Len      ]
plot( trajHead , trajTail , pch="." , col="skyblue" ,
      main=bquote( list( "Prpsl.SD" == .(proposalSD) ,
                         lag == .(Lag) ,
                         cor == .(round(cor(trajHead,trajTail),3)))) )
```

(A) Before each line, add a comment that explains what the line does. Include the commented code in your write-up.

(B) Repeat the previous exercise, with the lines above appended to the script. Include the resulting new graphs in your write-up. For each run, verify that the height of the ACF bar at the specified lag matches the correlation in the scatterplot.

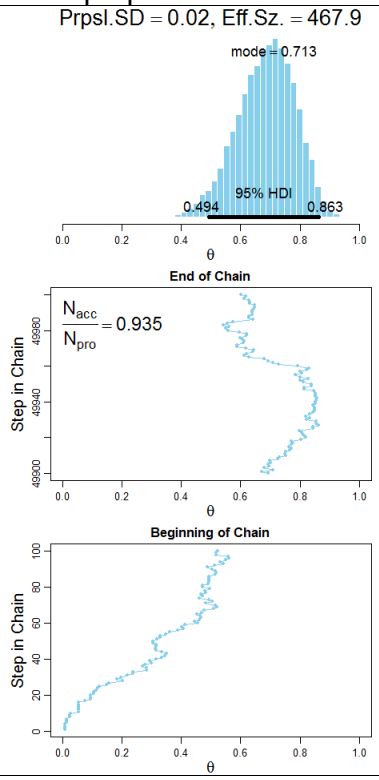
(C) When the proposal distribution has SD=2, why does the scatter plot have a dense line of points on the diagonal? (*Hint:* Look at the trajectory.)

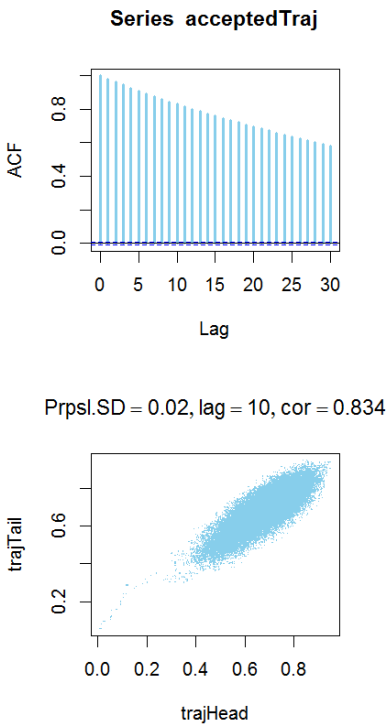
(A) Commented new code at end of script:

```
openGraph(height=7,width=3.5) # Open a tall graphics window.
layout(matrix(1:2,nrow=2))    # Graphics window has two panels.
# Compute autocorrelation function of accepted trjectory, with maximum lag of
# 30, and plot the function using skyblue color and linewidth of 3:
acf( acceptedTraj , lag.max=30 , col="skyblue" , lwd=3 )
# Now make a scatter plot of the MCMC trjectory plotted against the its values
# 10 steps later. Call the initial values trajHead, and call the lagged values
# trajTail.
Len = length( acceptedTraj ) # Store length of trajectory for convenience.
Lag = 10                     # Specify the lag.
trajHead = acceptedTraj[ 1 : (Len-Lag) ] # Extract all but the last Lag steps.
trajTail = acceptedTraj[ (1+Lag) : Len ] # Extract all but the 1st Lag steps.
# Make a scatter plot of original values on x axis against values Lag steps
# later on y axis. Also display the correlation in the main title of the plot.
plot( trajHead , trajTail , pch="." , col="skyblue" ,
      main=bquote( list( "Prpsl.SD" == .(proposalSD) ,
                         lag == .(Lag) ,
                         cor == .(round(cor(trajHead,trajTail),3)))) )
```

(B) Graphical results at each proposal SD:

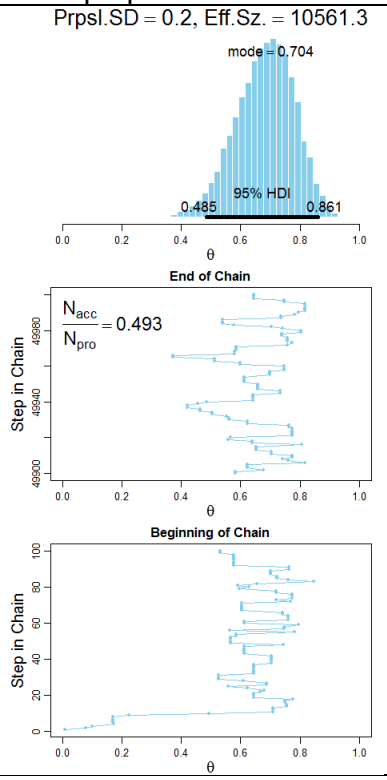
When proposal SD = 0.02:

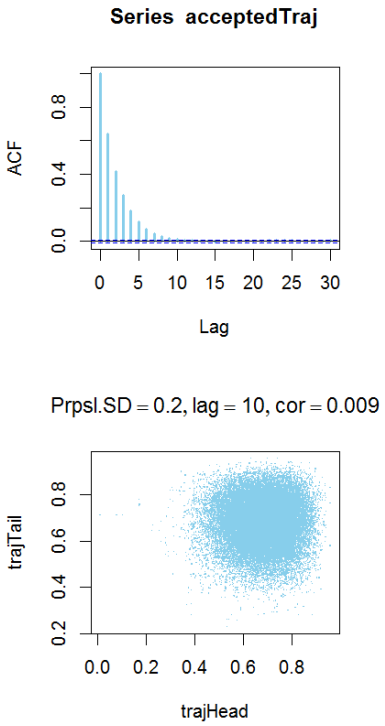




From the trajectory on the left, we can see that each step does not move very far from its previous position, hence the autocorrelation should be high. This is verified in the ACF plotted on the right. In the upper-right plot, the bar at Lag=10 has a height between .8 and .9, which is matched by cor=0.834 in the title of the lower right scatter plot.

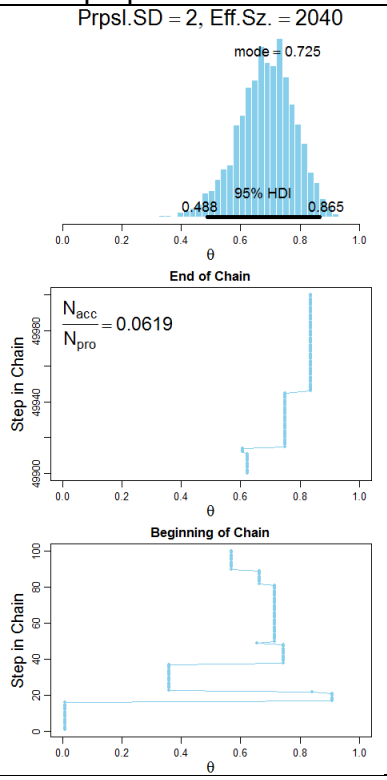
When proposal SD = 0.2:

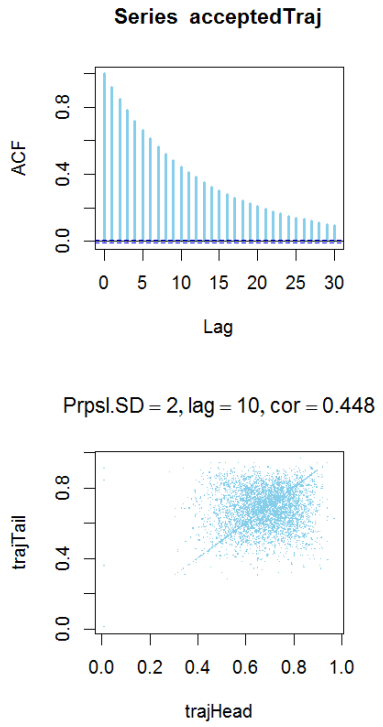




From the trajectory on the left, we can see that each step moves a fair distance from its previous position, hence the autocorrelation should be moderate. This is verified in the ACF plotted on the right. In the upper-right plot, the bar at Lag=10 has a height of nearly 0, which is matched by cor=0.009 in the title of the lower right scatter plot.

When proposal SD = 2.0:





From the trajectory on the left, we can see that each step does not move very far from its previous position most of the time, but has occasional big jumps, hence the autocorrelation should be fairly high. This is verified in the ACF plotted on the right. In the upper-right plot, the bar at Lag=10 has a height between .4 and .5, which is matched by cor=0.448 in the title of the lower right scatter plot.

(C) *When the proposal SD is 2.0*, the scatter plot has a dense line of points on the diagonal (see lower right panel of previous part) because of all the steps in the trajectory for which the value at step i is the same as the value at step i+10. For most steps, the value of the parameter does not change, and there are many steps for which the values has not changed even after 10 proposals. Thus, acceptedTraj[i] = acceptedTraj[i+10], and the plotted point falls on the main diagonal.

Exercise 7.3

Exercise 7.3. [Purpose: Using a multimodal prior with the Metropolis algorithm, and seeing how chains can transition across modes or get stuck within them.] In this exercise, you will see that the Metropolis algorithm operates with multimodal distributions.

(A) Consider a prior distribution on coin bias that puts most credibility at 0.0, 0.5, and 1.0, which we will formulate as $p(\theta) = (\cos(4\pi\theta) + 1)^2/1.5$.

(B) Make a plot of the prior. *Hint:* `theta = seq(0,1,length=501); plot (theta , (cos(4*pi*theta)+1)^2/1.5 )`

(C) In the script `BernMetrop.R`, find the function definition that specifies the prior distribution. Inside that function definition, comment out the line that assigns a beta density to `pTheta`, and instead put in a trimodal prior like this:

```
#pTheta = dbeta( theta , 1 , 1 )
pTheta = (cos(4*pi*theta)+1)^2/1.5
```

To have the Metropolis algorithm explore the prior, we give it empty data. Find the line in the script that specifies the data and set `myData = c()`. Run the script, using a proposal SD=0.2. Include the graphical output in your write-up. Does the histogram of the trajectory look like the graph of the previous part of the exercise?

(D) Repeat the previous part but now with `myData = c(0,1,1)`. Include the graphical output in your write-up. Does the posterior distribution make sense? Explain why.

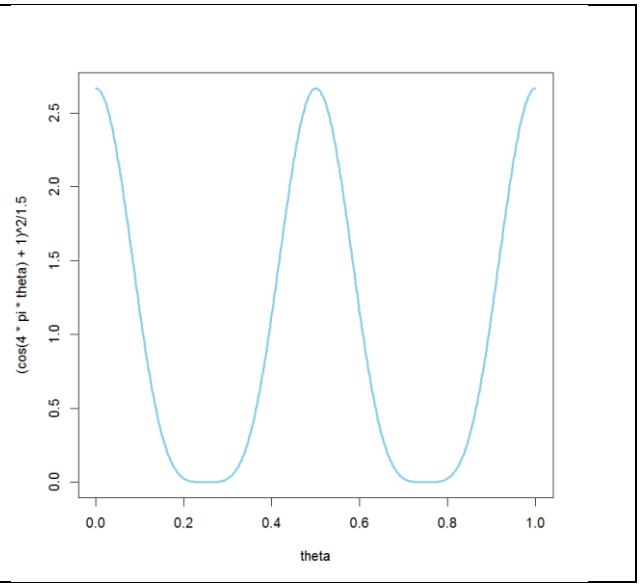
(E) Repeat the previous part but now with proposal SD=0.02. Include the graphical output in your write-up. Does the posterior distribution make sense? Explain why *not*; what has gone wrong? If we did not know from the previous part that this output was unrepresentative of the true posterior, how could we try to check? *Hint:* See next part.

(F) Repeat the previous part but now with the initial position at 0.99: `trajectory[1] = 0.99`. In conjunction with the previous part, what does this result tell us?

(A) Okay, I'm considering… (There's nothing tangible to produce for this part.)

(B) The graph shows that the distribution really is tri-modal:

```
source("DBDA2E-utilities.R")
openGraph()
theta = seq(0,1,length=501)
plot( theta , (cos(4*pi*theta)+1)^2/1.5 ,
      type="l" , lwd=3 , col="skyblue" )
```



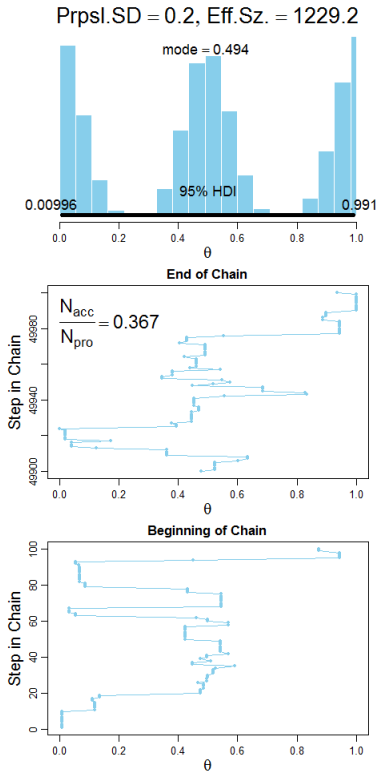
(C) The prior specification:

```
# Define the prior density function.
prior = function( theta ) {
  # pTheta = dbeta( theta , 1 , 1 )
  pTheta = (cos(4*pi*theta)+1)^2/1.5
  # The theta values passed into this function are generated at random,
  # and therefore might be inadvertently greater than 1 or less than 0.
  # The prior for theta > 1 or for theta < 0 is zero:
  pTheta[ theta > 1 | theta < 0 ] = 0
  return( pTheta )
}
```

The empty data specification:

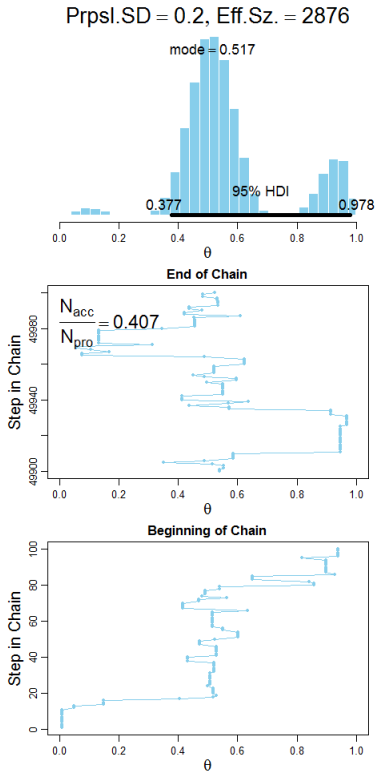
```
# myData = c(rep(0,6),rep(1,14))
myData = c()
```

The resulting trajectory for SD=0.2:



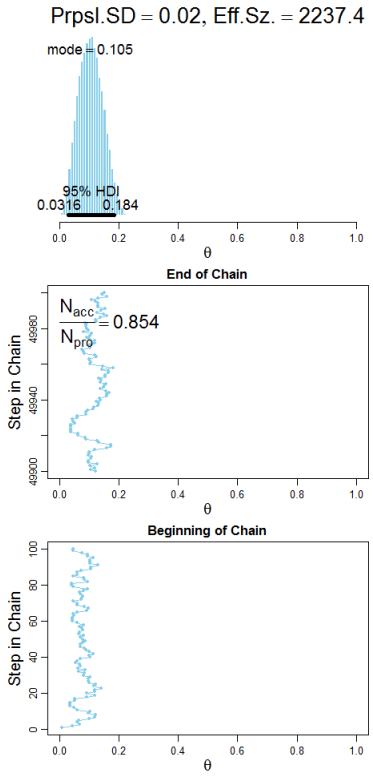
The histogram of the trajectory does indeed look like the trimodal prior distribution graphed in the previous part. Thus, the metropolis algorithm is able to sample representatively from multimodal distributions, at least in this case. (The HDI as marked is irrelevant and potentially misleading because its algorithm assumes a unimodal distribution; see Ch. 25.)

(D) Using myData=c(0,1,1) and proposal SD=0.2:



The posterior distribution makes sense because it is a compromise between the trimodal prior and the gently peaked likelihood function of the small data set. The likelihood function peaks at 0.667 (i.e., 2/3), but is very broad, gently descending to 0 height at theta=0 and theta=1. Therefore the prior peaks at theta=0.5 and near theta=1.0 are "allowed" by the data, but the prior peak near theta=0.0 is relatively unlikely given the data. (The HDI as marked is irrelevant and potentially misleading because its algorithm assumes a unimodal distribution; see Ch. 25.)

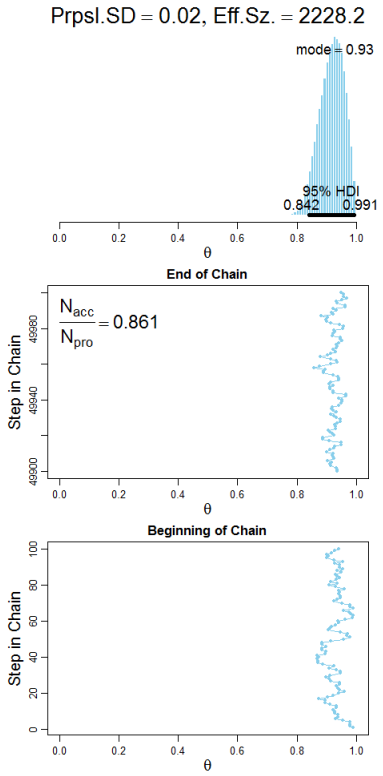
(E) With proposal SD=0.02:



The MCMC posterior shows a single peak near theta=0, which is strange because the data have 2/3 1's, and the prior allows theta values near 0.5 and near 1.0. Why does this anomalous result occur? Because the arbitrary initial value of the chain is very near theta=0, and the very small proposal SD has a difficult time squeezing under the very low ceiling between modes of the prior. With a long enough wait, there might be a series of proposals that just happen to make it under the low ceiling, but this would take a very very long time!

If we did not know from other considerations that this was an anomalous result, we could start several chains at different starting positions and check that they all converged to the same region of parameter space. (See the next part of the exercise for an example of starting at a different point in parameter space.)

(F) Using trajectory[1]=0.99:



When the trajectory arbitrarily starts at theta=0.99, and the proposal SD is 0.02, the MCMC chain gets stuck in the prior mode near theta=1. If we started chains at several different values of theta, we would notice that they do not converge to the same region of parameter space, which would be a sign that the results are not fully representative of the posterior, and we would need to adjust the MCMC sampler.

Chapter 8

Exercise 8.1

Exercise 8.1. [Purpose: Run the high level scripts with other data to see how easy they are.] Consider the high-level script, Jags-Ydich-XnomSsubj-MbernBeta-Example.R. For this exercise, you will use that script with a new data file, and notice that you only need to change a single line, namely the one that loads the data file.

In RStudio, open a new blank file by selecting the menu items File → New → Text file. Manually type in new fictional data in the same format as the data shown in Section 8.4, p. 208, but with three subjects instead of two. Use whatever names you fancy, and as many trials for each subject as you fancy. Perhaps put a preponderance of 0's for one subject and a preponderance of 1's for another subject. Use a lot of trials of one subject, and relatively few trials for another. Save the file with a filename that ends with ".csv." Then, in the script, use that file name in the `read.csv` command. In your report, include the data file and the graphical output of the analysis. Are the estimates reasonable? What is the effect of different sample sizes for the estimates of different subjects?

Here is an example of how some new data can be set up:

```
# Here is one way to set up simulated data in R and save it in a file. You can
# use a text editor or Excel or whatever. For subject names, I've used "A", "B",
# and "C", merely for simplicity.
y = c( rep(1,9),rep(0,3) , rep(1,45),rep(0,15) , rep(1,3),rep(0,9)  )
s = c( rep("A",12)       , rep("B",60)         , rep("C",12)        )
write.csv( data.frame(y=y,s=s) , file="Exercise.08.1.csv" , row.names=FALSE )
```

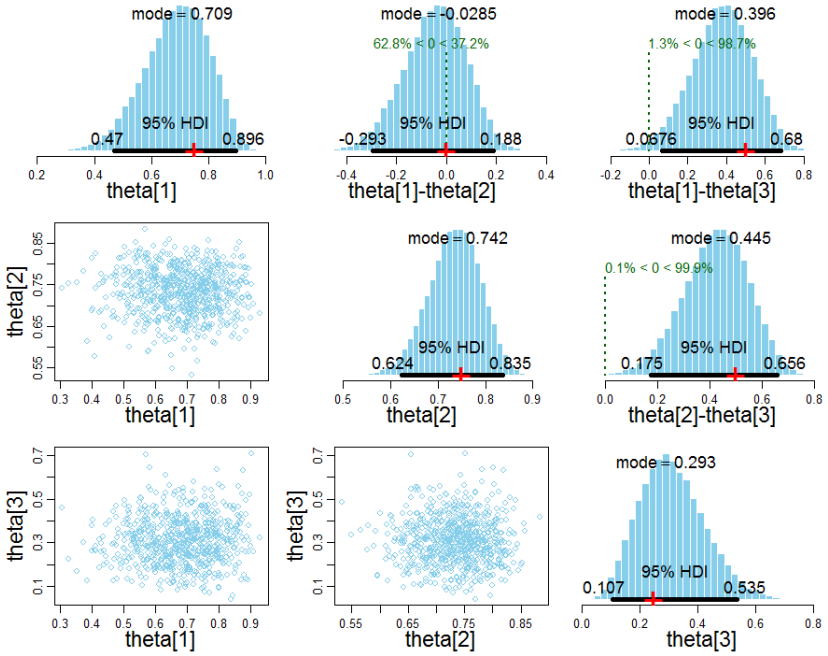
Notice above that the data are written to a file named Exercise.08.1.csv. The resulting data file has 84 rows, and its beginning looks like this:

	y	s
1	1	A
2	1	A
3	1	A
4	1	A
5	1	A
6	1	A
7	1	A
8	1	A
9	1	A
10	0	A
11	0	A
12	0	A
13	1	B
14	1	B

We then run the Bayesian analysis exactly as in the example script, with a single change to the line that reads in the data file, highlighted in yellow below:

```
# Below is just the essential lines of Jags-Ydich-XnomSsubj-MbernBeta-Example.R
# with the data file changed:
graphics.off()
rm(list=ls(all=TRUE))
fileNameRoot="Exercise.08.1" # for output filenames
source("DBDA2E-utilities.R")
# Load The data from the file:
myData = read.csv("Exercise.08.1.csv")
# Load the relevant model into R's working memory:
source("Jags-Ydich-XnomSsubj-MbernBeta.R")
# Generate the MCMC chain:
mcmcCoda = genMCMC( data=myData , numSavedSteps=50000 , saveName=fileNameRoot )
parameterNames = varnames(mcmcCoda) # get all parameter names
for ( parName in parameterNames ) {
  diagMCMC( codaObject=mcmcCoda , parName=parName )
}
# Get summary statistics of chain:
summaryInfo = smryMCMC( mcmcCoda , compVal=NULL , #rope=c(0.45,0.55) ,
                        compValDiff=0.0 , #ropeDiff = c(-0.05,0.05) ,
                        saveName=fileNameRoot )
# Display posterior information:
plotMCMC( mcmcCoda , data=myData , compVal=NULL , #rope=c(0.45,0.55) ,
          compValDiff=0.0 , #ropeDiff = c(-0.05,0.05)
          )
```

MCMC diagnostic graphs are produced, not shown here. Below is the posterior distribution:



Looking at the diagonal panels, we see that the HDI for theta[1] (subject "A") is wider than the HDI for theta[2] (subject "B"), despite the fact that they had identical proportions of 1's in their data (75%), because the data for theta[1] are far fewer than the data for theta[1]. The mode for theta[3] is noticeably shrunken toward the higher proportions.

Exercise 8.2

Exercise 8.2. [Purpose: Pay attention to the output of `smryMCMC`.] The graphical plots from `plotMCMC` are useful for understanding, but they lack some numerical details. Run the high-level script, Jags-Ydich-XnomSsubj-MbernBeta-Example.R, and explain the details in the output from `smryMCMC`. Explain what the `rope` and `ropeDiff` arguments do.

The script has some of the arguments of smryMCMC commented out; for purposes of illustration I have used all the arguments in this answer, like this:

```
summaryInfo = smryMCMC( mcmcCoda , compVal=0.5 , rope=c(0.45,0.55) ,
                        compValDiff=0.0 , ropeDiff = c(-0.05,0.05) )
```

The output of smryMCMC looks like the following:

```
                       Mean    Median      Mode     ESS HDImass       HDIlow   HDIhigh
theta[1]           0.6665165 0.6757450 0.6774373 50000.0    0.95  0.41323339 0.9063190
theta[2]           0.3629850 0.3539830 0.3170888 50971.5    0.95  0.10821283 0.6344008
theta[1]-theta[2] 0.3035315 0.3111328 0.3010779 50000.0    0.95 -0.06657935 0.6693374

                  CompVal PcntGtCompVal ROPElow ROPEhigh PcntLtROPE PcntInROPE
theta[1]              0.5        88.624    0.45     0.55      6.072     13.326
theta[2]              0.5        17.002    0.45     0.55     73.556     16.366
theta[1]-theta[2]     0.0        93.732   -0.05     0.05      3.840      6.024

                  PcntGtROPE
theta[1]              80.602
theta[2]              10.078
theta[1]-theta[2]     90.136
```

Your output will differ in the numerical details because of randomness in the MCMC chain. Although the output includes many decimal places, most are not significant because of the sampling randomness in the MCMC chain; only the first few digits are stable, depending on the ESS.

Each row corresponds to the parameter or parameter difference indicated in the left-most column. The columns labelled Mean, Median, and Mode are the corresponding values of the MCMC chain for that parameter. The ESS is the effective sample size, which is the chain length divided by the autocorrelation, as defined in the book. The next three columns indicate the probability mass of the HDI (which here is the default of 95%), the lower limit of the HDI, and the upper limit. Next is the comparison value (CompVal) for single-parameter decisions, which was specified in the argument as 0.5. The next column indicates the percentage of the posterior that is greater than the comparison value (PcntGtCompVal). Next are the columns for the ROPE, which repeat the specifications in the arguments. The final three columns indicate the percentage of the posterior distribution that is less than the ROPE lower limit, within the ROPE limits, and greater than the ROPE upper limit.

Exercise 8.3

Exercise 8.3. [Purpose: Notice what gets saved by the high-level scripts.] Run the high-level script, Jags-Ydich-XnomSsubj-MbernBeta-Example.R, and notice what files are created (i.e., saved) in your computer's working directory. Explain what these files are, and why they might be useful for future reference. Hint: In the file Jags-Ydich-XnomSsubj-MbernBeta.R, search for the word "save."

Here is the script, with relevant phrases highlighted in yellow, and explanatory comments:

```
fileNameRoot = "Jags-Ydich-XnomSsubj-MbernBeta-"
graphFileType = "eps"

The first line above specifies the beginning of the filenames for saved information,
and the second line above specifies the graphics format for saved graphs.

# Generate the MCMC chain:
mcmcCoda = genMCMC( data=myData , numSavedSteps=50000 , saveName=fileNameRoot )

The MCMC chain is saved in a file named
Jags-Ydich-XnomSsubj-MbernBeta-Mcmc.Rdata
Notice the name is the fileNameRoot with Mcmc appended.
It is in compressed Rdata format.

# Display diagnostics of chain, for specified parameters:
parameterNames = varnames(mcmcCoda) # get all parameter names
for ( parName in parameterNames ) {
  diagMCMC( codaObject=mcmcCoda , parName=parName ,
                saveName=fileNameRoot , saveType=graphFileType )
}

The diagnostic graphs are saved in files named
Jags-Ydich-XnomSsubj-MbernBeta-Diagtheta[1].eps and
Jags-Ydich-XnomSsubj-MbernBeta-Diagtheta[2].eps
Notice the names are the fileNameRoot with Diag<parameter> appended.

# Get summary statistics of chain:
summaryInfo = smryMCMC( mcmcCoda , compVal=0.5 , rope=c(0.45,0.55) ,
                        compValDiff=0.0 , ropeDiff = c(-0.05,0.05) ,
                        saveName=fileNameRoot )

The summary information is saved in a file named
Jags-Ydich-XnomSsubj-MbernBeta-SummaryInfo.csv
which is in comma-separated-value format.
Notice the name is the fileNameRoot with SummaryInfo appended.

# Display posterior information:
plotMCMC( mcmcCoda , data=myData , compVal=NULL , #rope=c(0.45,0.55) ,
          compValDiff=0.0 , #ropeDiff = c(-0.05,0.05) ,
          saveName=fileNameRoot , saveType=graphFileType )

The graph of the posterior distribution is saved in a file named
Jags-Ydich-XnomSsubj-MbernBeta-Post.eps
Notice the name is the fileNameRoot with Post appended
```

Exercise 8.4

Exercise 8.4. [Purpose: Explore the prior on a difference of parameters implied from the priors on the individual parameters.]

(A) Reproduce Figure 8.7 in Section 8.5. Explain how you did it.

(B) Change the priors on the individual θ's to beta($\theta|1, 1$) and produce the figure anew. Describe its panels and explain.

(C) Change the priors on the individual θ's to beta($\theta|0.5, 0.5$) and produce the figure again. Describe its panels and explain.

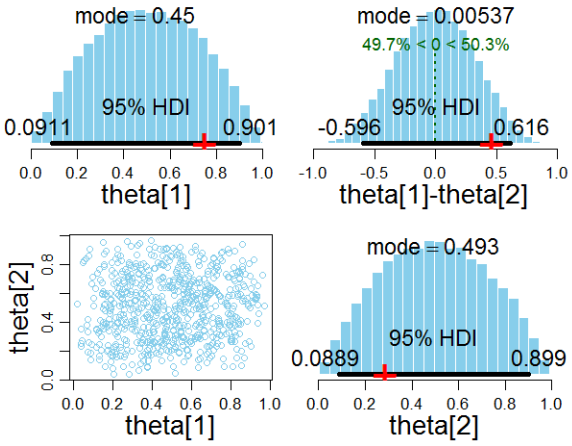
(A) To generate Figure 8.7, do the following:

1. In the file Jags-Ydich-XnomSsubj-MbernBeta.R, comment out the line that specifies y in the dataList, like this:

```
dataList = list(
  # y = y ,
  s = s ,
  Ntotal = Ntotal ,
  Nsubj = Nsubj
)
```

2. Save the file.

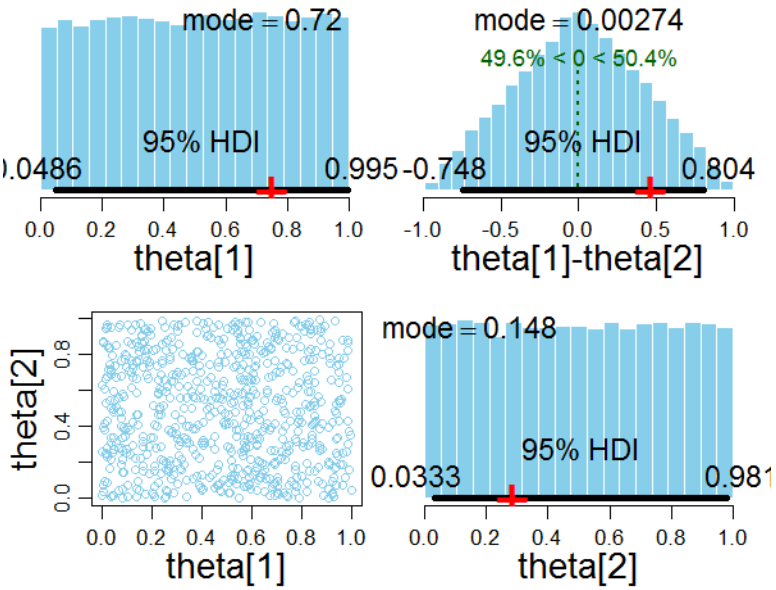
3. Run the script Jags-Ydich-XnomSsubj-MbernBeta-Example.R. The graph of the "posterior" looks like this (which is Figure 8.7):



(B) In the file Jags-Ydich-XnomSsubj-MbernBeta.R, change the specification of the prior to dbeta(1,1), as highlighted below:

```
model {
  for ( i in 1:Ntotal ) {
    y[i] ~ dbern( theta[s[i]] )
  }
  for ( s in 1:Nsubj ) {
    theta[s] ~ dbeta( 1 , 1 )
  }
}
```

Save the file. Then run the script Jags-Ydich-XnomSsubj-MbernBeta-Example.R. The graph of the prior looks like this:



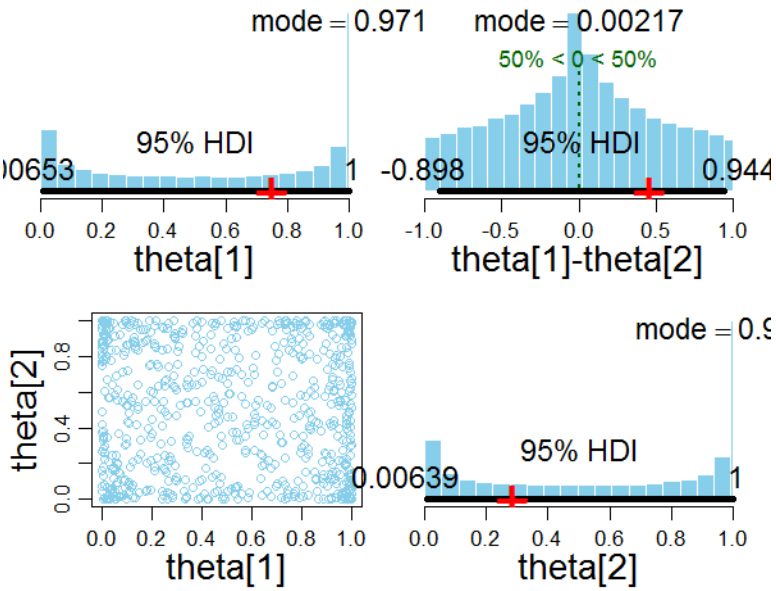
Notice that the distributions on theta[1] and theta[2] look uniform, as they should, because that is a dbeta(1,1) distribution. (The modes of the distributions are spurious because they are merely indicating what tiny random bump in the MCMC sample happens to highest.)

Interestingly, the prior distribution on theta[1]-theta[2] is *not* uniform, but is instead triangular. This makes sense if you consider the lower left panel, which shows the uniform distribution on theta[1] x theta[2] space. If you collapse that square space along the theta[1]=theta[2] diagonal, you see that there are a lot of points along the diagonal, but the number of points drops off linearly toward the corners. In any case, the moral is that uniform priors on theta[1] and theta[2] do not imply a uniform prior on the difference of parameters.

(C) The model statement in Jags-Ydich-XnomSsubj-MbernBeta.R is changed to the following

```
model {
  for ( i in 1:Ntotal ) {
    y[i] ~ dbern( theta[s[i]] )
  }
  for ( s in 1:Nsubj ) {
    theta[s] ~ dbeta( 0.5 , 0.5 )
  }
}
```

and the file is saved, and then the script is run. The resulting prior distribution on the parameters looks like this:



The individual parameters can be seen to have dbeta(0.5,0.5) marginal distributions, but the resulting distribution on theta[1]-theta[2] is curious. By looking at the joint distribution in the lower left panel, some insight can be gleaned; the joint distribution has higher density toward the corners. Again, the main point is that a prior on individual parameters may have unforeseen implications for the prior on the difference of parameters.

When you are done with this exercise, you might want to return the program Jags-Ydich-XnomSsubj-MbernBeta.R to its original condition!

Chapter 9

Exercise 9.1

Exercise 9.1. [Purpose: Try different priors on κ to explore the role of κ in shrinkage.] Consider the analysis of the therapeutic touch data in Figure 9.10, p. 243. The analysis used a generic gamma distributed prior on κ that had a *mean* of 1.0 and a standard deviation of 10.0. We assumed that the prior had minimal influence on the results; here, we examine the robustness of the posterior when we change the prior to other reasonably vague and noncommittal distributions. In particular, we will examine a gamma distributed prior on κ that had a *mode* of 1.0 and a standard deviation of 10.0.

(A) What are the shape and rate parameters for a gamma distribution that has mean of 1.0 and standard deviation of 10.0? What are the shape and rate parameters for a gamma distribution that has mode of 1.0 and standard deviation of 10.0? *Hint:* use the utility functions `gammaShRaFromMeanSD` and `gammaShRaFrom ModeSD`.

```
> source("DBDA2E-utilities.R")

> gammaShRaFromMeanSD( mean=1.0 , sd=10.0 )
$shape
[1] 0.01
$rate
[1] 0.01

> gammaShRaFromModeSD( mode=1.0 , sd=10.0 )
$shape
[1] 1.105125
$rate
[1] 0.1051249
```

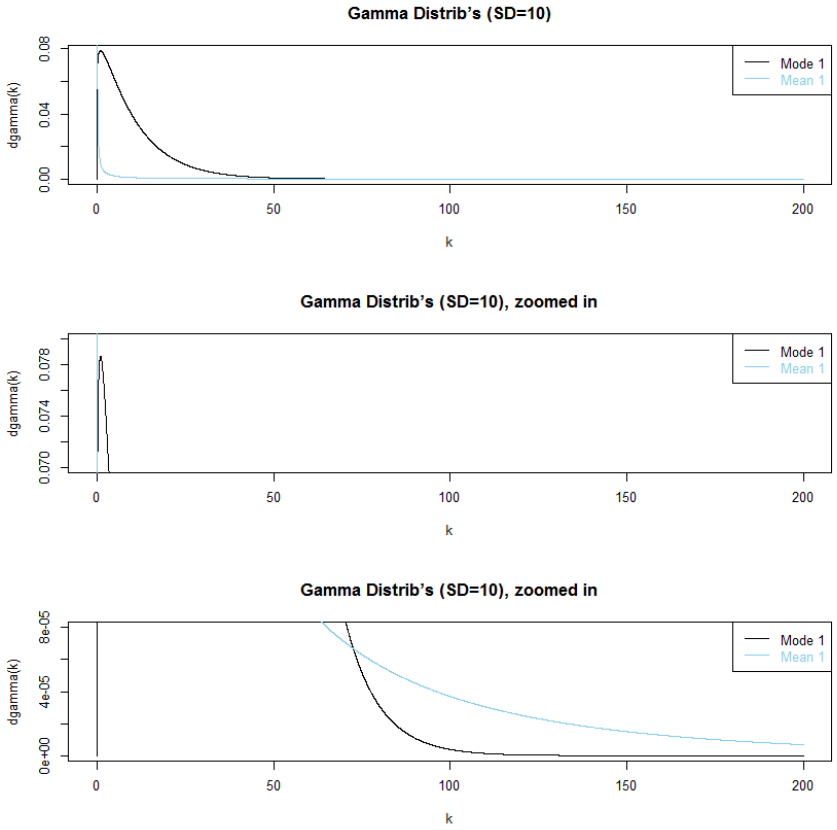
(B) Plot the two gamma distributions, superimposed, to see which values of κ they emphasize. If you like, make the graphs with this R code:

```
openGraph(height=7,width=7)
layout(matrix(1:3,ncol=1))
k=seq(0,200,length=10001)
plot( k , dgamma(k,1.105125,0.105125) , ylab="dgamma(k)" ,
      type="l" , main="Gamma Distrib's (SD=10)" )
lines( k , dgamma(k,0.01,0.01) , col="skyblue" )
legend( "topright" , c("Mode 1","Mean 1") ,
        lty=c(1,1) , col=c("black","skyblue") , text.col=c("black", "skyblue") )
plot( k , dgamma(k,1.105125,0.105125) , ylab="dgamma(k)" ,
      type="l" , ylim=c(.07,.08) , main="Gamma Distrib's (SD=10), zoomed in" )
lines( k , dgamma(k,0.01,0.01) , col="skyblue" )
legend( "topright" , c("Mode 1","Mean 1") ,
        lty=c(1,1) , col=c("black","skyblue") , text.col=c("black", "skyblue") )
plot( k , dgamma(k,1.105125,0.105125) , ylab="dgamma(k)" ,
      type="l" , ylim=c(0,8.0e-5) , main="Gamma Distrib's (SD=10), zoomed in" )
lines( k , dgamma(k,0.01,0.01) , col="skyblue" )
legend( "topright" , c("Mode 1","Mean 1") ,
        lty=c(1,1) , col=c("black","skyblue") , text.col=c("black", "skyblue") )
```

The result is shown in Figure 9.18. Relative to each other, which gamma distribution favors values of κ between about 0.1 and 75? Which gamma distribution favors values of κ that are tiny or greater than 75?

```
openGraph(height=7,width=7)
layout(matrix(1:3,ncol=1))
k=seq(0,200,length=10001)
plot( k , dgamma(k,1.105125,0.105125) , ylab="dgamma(k)" ,
      type="l" , main="Gamma Distrib's (SD=10)" )
lines( k , dgamma(k,0.01,0.01) , col="skyblue" )
legend( "topright" , c("Mode 1","Mean 1") ,
        lty=c(1,1) , col=c("black","skyblue") , text.col=c("black", "skyblue") )
plot( k , dgamma(k,1.105125,0.105125) , ylab="dgamma(k)" ,
      type="l" , ylim=c(.07,.08) , main="Gamma Distrib's (SD=10), zoomed in" )
lines( k , dgamma(k,0.01,0.01) , col="skyblue" )
legend( "topright" , c("Mode 1","Mean 1") ,
        lty=c(1,1) , col=c("black","skyblue") , text.col=c("black", "skyblue") )
plot( k , dgamma(k,1.105125,0.105125) , ylab="dgamma(k)" ,
      type="l" , ylim=c(0,8.0e-5) , main="Gamma Distrib's (SD=10), zoomed in" )
lines( k , dgamma(k,0.01,0.01) , col="skyblue" )
legend( "topright" , c("Mode 1","Mean 1") ,
        lty=c(1,1) , col=c("black","skyblue") , text.col=c("black", "skyblue") )
```

yields this graph, which looks exactly like Figure 9.18:



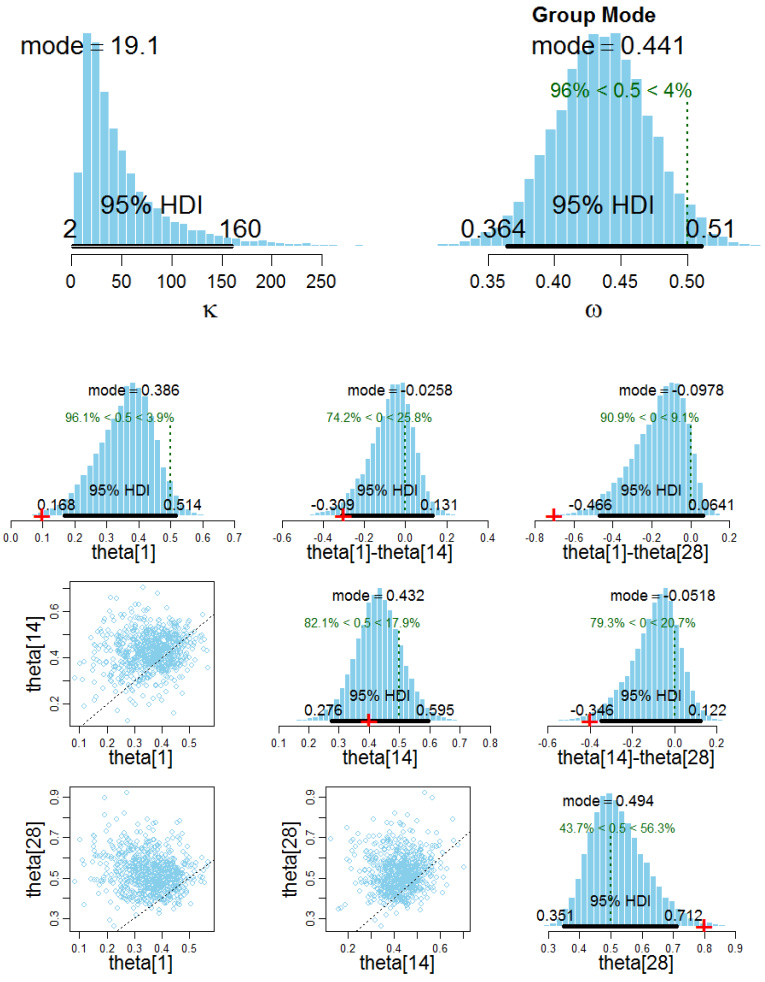
It can be seen from the graphs that the Mode 1 gamma has higher density for k approximately in the range 0.1 to 75. On the other hand, the Mean 1 gamma has higher density for k greater than about 75 or less than 0.1.

(C) In the program Jags-Ydich-XnomSsubj-MbinomBetaOmegaKappa.R, find the line in the model specification for the prior on `kappaMinusTwo`. Run the program once using a gamma with mean of 1.0, and run the program a second time using a gamma with a mode of 1.0. Show the graphs of the posterior distribution. *Hints:* in the model specification, just comment out one or the other of the lines:

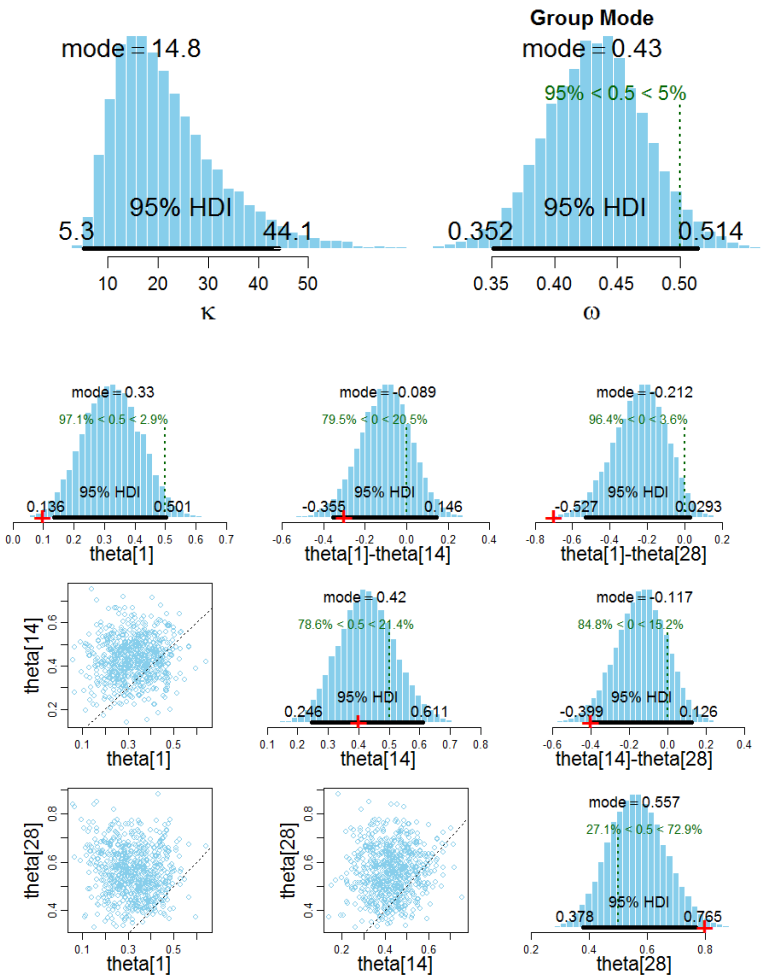
```
#kappaMinusTwo ~ dgamma( 0.01 , 0.01 )  # mean=1 , sd=10 (generic vague)
kappaMinusTwo ~ dgamma( 1.105125 , 0.1051249 )  # mode=1 , sd=10
```

Be sure to save the program before calling it from the script! In the script, you might want to change the file name root that is used for saved graph files.

kappaMinusTwo ~ dgamma(0.01 , 0.01) # mean=1 , sd=10
yields:



kappaMinusTwo ~ dgamma(1.105125 , 0.1051249) # mode=1 , sd=10
yields



(D) Does the posterior distribution change much when the prior is changed? In particular, for which prior does the marginal posterior distribution on κ have a bigger large-value tail? When κ is larger, what effect does that have on shrinkage of the θ_s values?

When using the mean=1 prior on kappa-2, the posterior on kappa has a larger tail than when using the mode=1 prior. This makes sense because the mean=1 prior emphases larger kappa values (>75) than the mode=1 prior. When kappa is larger, the individual theta values must be shrunken more, as can be seen in the posterior distributions of theta[1] and theta[28]: There is more shrinkage when using the mean=1 prior than when using the mode=1 prior.

(E) Which prior do you think is more appropriate? To properly answer this question, you should do the next exercise!

Presumably, we want the prior on kappa-2 that puts uniform priors on the individual theta's and puts as broad a prior as possible on the differences of theta's. The next exercise shows that this is the mean=1 prior on kappa-2, not the mode=1 prior.

Exercise 9.2

Exercise 9.2. [Purpose: Examine the prior on θ_s implied by the prior constants at higher levels.] To sample from the prior in JAGS, we just comment out the data, as was explained in Section 8.5. In the program Jags-Ydich-XnomSsubj-MbinomBetaOmegaKappa.R, just comment out the line that specifies z, like this:

```
  dataList = list(
#   z = z ,
    N = N ,
    Nsubj = Nsubj
  )
```

Save the program, and run it with the two priors on κ discussed in the previous exercise. You may want to change the file name root for the saved graphics files. For both priors, include the graphs of the prior distributions on θ_s and the differences of θ_s's such as `theta[1]-theta[28]`. See Figure 9.19.

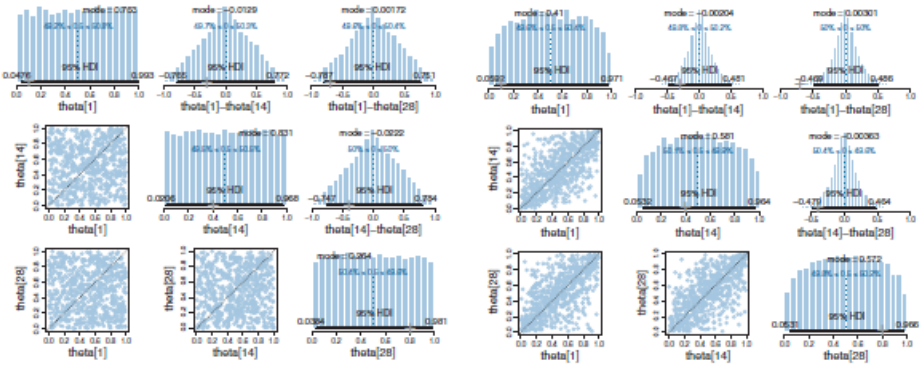
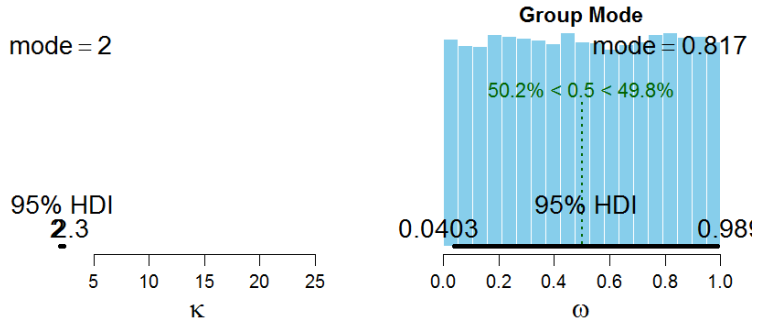


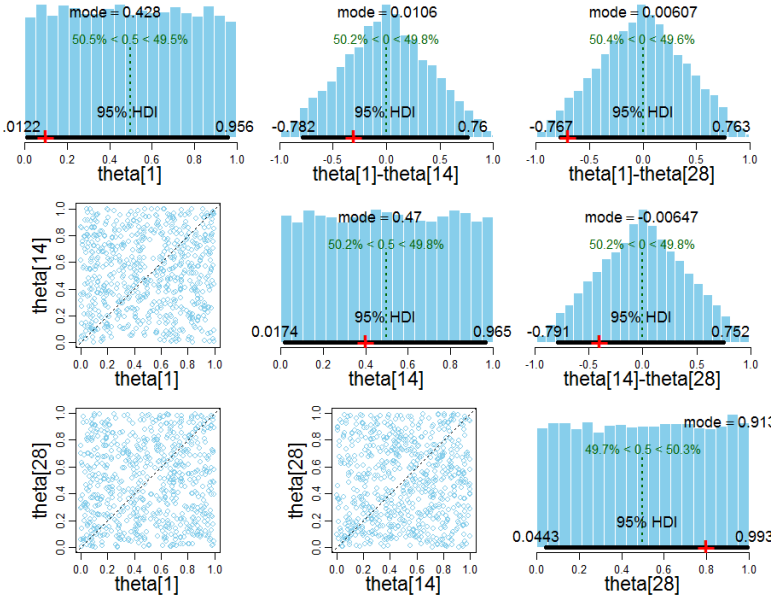
Figure 9.19 Priors on θ_s implied by different gamma distributions on κ. For use with Exercise 9.2.

Correction: In the dataList, comment out the line that specifies y (there is no line that specifies z).

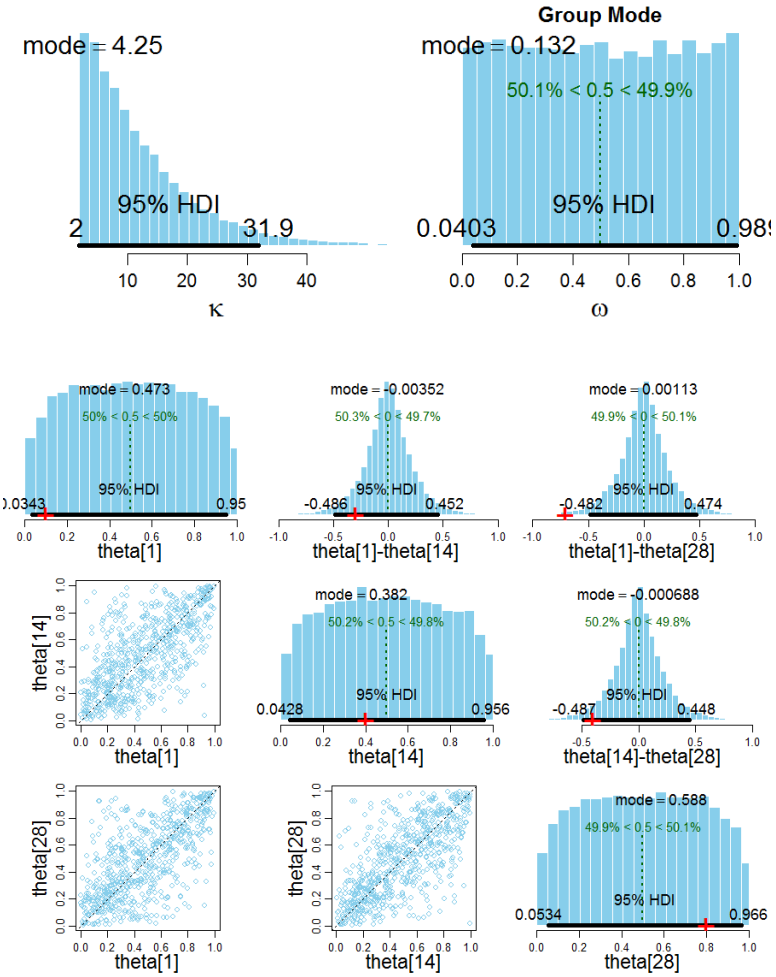
When kappaMinusTwo ~ dgamma(0.01 , 0.01) # mean=1 , sd=10:



The prior on kappa, above, plots badly because it is strongly skewed. The displayed mode is spurious because of the approximation algorithm that computes it.



When using kappaMinusTwo ~ dgamma(1.105125 , 0.1051249) # mode=1 , sd=10:



(A) Explain why the implied prior distribution on individual θ_s has rounded shoulders (instead of being essentially uniform) when using a prior on κ that has a mode of 1 (instead of a mean of 1).

When the prior has mode=1, kappa does not get extremely small. (But when the prior has mean=1, kappa has a fairly high probability of being extremely small.) It is only when kappa is extremely small that two different theta's can have opposite extremes. For example, randomly generating theta[1]=0.01 and theta[28]=0.99 from the prior can reasonably happen only when kappa is extremely small, which is when the prior mean=1.

(B) Which prior do you think is more appropriate?

Presumably, we want the prior on kappa-2 that puts uniform priors on the individual theta's and puts as broad a prior as possible on the differences of theta's. This implies the mean=1 prior on kappa-2, not the mode=1 prior.

Exercise 9.3

Exercise 9.3. [Purpose: Compare Bayesian shrinkage with MLE shrinkage] Construct a data set like the data in Figure 9.12 and do a Bayesian analysis like that done for the therapeutic touch data. Compare the Bayesian parameter estimates with the MLE estimates (gleaned from Figure 9.12). What does the Bayesian analysis provide that is not provided by the MLE?

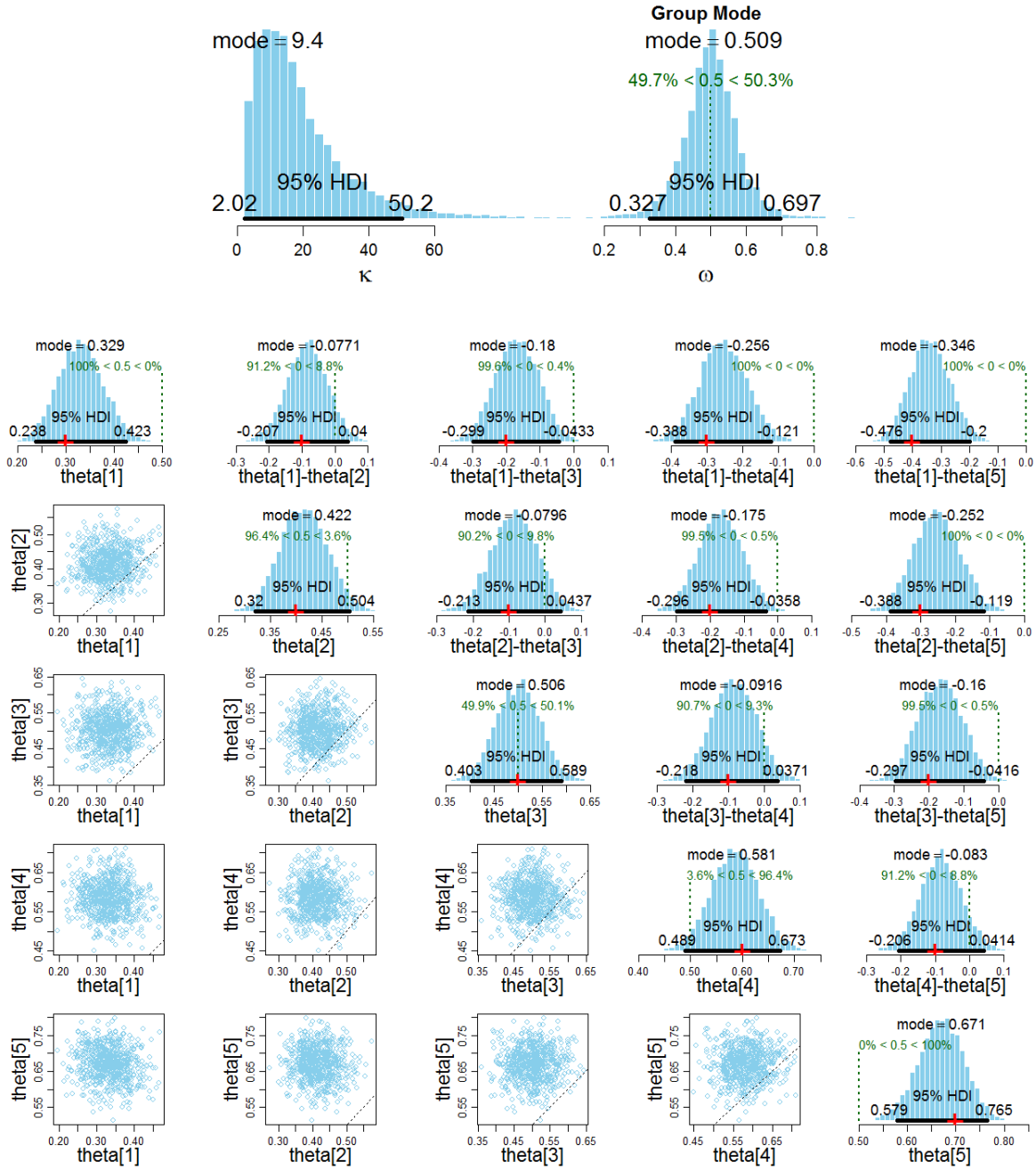
Here is an adaptation of the script that generates the data and runs the analysis:

```
# Generate the data frame:
# N.B.: The functions below expect the data to be a data frame,
# with one component being a vector of integer 0,1 values,
# and one component being a factor of subject identifiers.
headsTails = c( rep(1,30),rep(0,100-30),
                rep(1,40),rep(0,100-40),
                rep(1,50),rep(0,100-50),
                rep(1,60),rep(0,100-60),
                rep(1,70),rep(0,100-70) )
subjID = factor( c( rep("A",100),
                    rep("B",100),
                    rep("C",100),
                    rep("D",100),
                    rep("E",100) ) )
myData = data.frame( y=headsTails , s=subjID )
#------------------------------------------------------------------------------
# Load the relevant model into R's working memory:
source("Jags-Ydich-XnomSsubj-MbernBetaOmegaKappa.R")
fileNameRoot = "Exercise.09.3-"
graphFileType = "eps"
# Generate the MCMC chain:
mcmcCoda = genMCMC( data=myData , sName="s" , yName="y" ,
                    numSavedSteps=10000 , saveName=fileNameRoot , thinSteps=10 )
# Display diagnostics of chain, for specified parameters:
parameterNames = varnames(mcmcCoda) # get all parameter names for reference
for ( parName in parameterNames[c(1:3,length(parameterNames))] ) {
  diagMCMC( codaObject=mcmcCoda , parName=parName ,
            saveName=fileNameRoot , saveType=graphFileType )
}
# Get summary statistics of chain:
summaryInfo = smryMCMC( mcmcCoda , compVal=0.5 ,
                        diffIdVec=c(1,2,3,4,5), compValDiff=0.0,
                        saveName=fileNameRoot )
# Display posterior information:
plotMCMC( mcmcCoda , data=myData , sName="s" , yName="y" ,
          compVal=0.5 , #rope=c(0.45,0.55) ,
          diffIdVec=c(1,2,3,4,5), compValDiff=0.0, #ropeDiff = c(-0.05,0.05) ,
          saveName=fileNameRoot , saveType=graphFileType )
```

When using the mean=1 prior on kappa, the posterior shows little shrinkage at all, because the prior emphasizes small kappa values. Instead, let's use a *uniform* prior on kappa, like this:

```
#kappaMinusTwo ~ dgamma( 0.01 , 0.01 )  # mean=1 , sd=10 (generic vague)
#kappaMinusTwo ~ dgamma( 1.105125 , 0.1051249 )  # mode=1 , sd=10
kappaMinusTwo ~ dunif( 0.0001 , 100.0 )
```

Then there is noticeable shrinkage, qualitatively similar to the MLE solution described in the book, as shown below. In either case, what the Bayesian posterior distribution provides that the MLE does not provide is an explicit description of the uncertainty of the estimate, for all the parameters.



Exercise 9.4

Exercise 9.4. [Purpose: Explore duration of processing by JAGS] Consider the therapeutic touch data of Figure 9.9, as analyzed by the program `Jags-Ydich-XnomSsubj-MbinomBetaOmegaKappa.R`. Open that program in RStudio and find the section that calls runjags or rjags.

(A) Set the program (if it is not already) to use three parallel chains with runjags. Be sure to save the program after any changes. Then run the high-level script, `Jags-Ydich-XnomSsubj-MbinomBetaOmegaKappa-Example.R`. Be sure it has `proc.time()` before and after `genMCMC` so that you can monitor how long it takes to generate the MCMC chain. Report the elapsed time of the chain generation, and also include the chain-diagnostic graph of omega, which includes its ESS.

In the script Jags-Ydich-XnomSsubj-MbinomBetaOmegaKappa-Example.R, be sure the proc.time() commands are like this:

```
# Generate the MCMC chain:
startTime = proc.time()
mcmcCoda = genMCMC( data=myData , sName="s" , yName="y" ,
                    numSavedSteps=20000 , saveName=fileNameRoot ,
                    thinSteps=10 )
stopTime = proc.time()
show( stopTime-startTime )
```

In the program Jags-Ydich-XnomSsubj-MbinomBetaOmegaKappa.R, first set runjags to parallel like this:

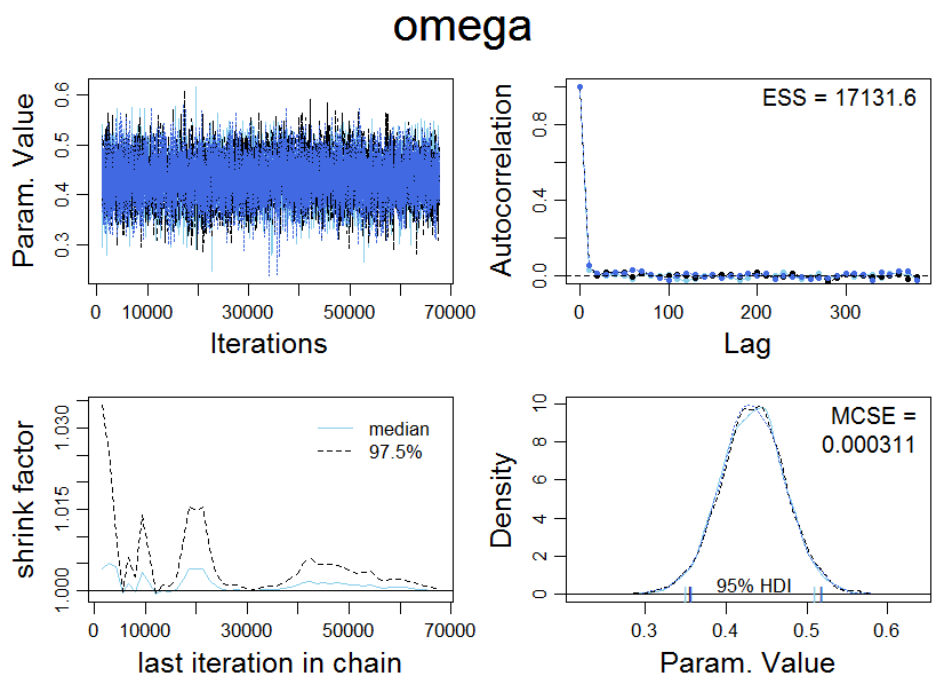
```
  nChains = 3
  useRunjags = TRUE
  if ( useRunjags ) {
    runJagsOut <- run.jags( method=c("rjags","parallel")[2] ,
                            model="TEMPmodel.txt" ,
                            monitor=parameters ,
                            data=dataList ,
                            inits=initsList ,
                            n.chains=nChains ,
                            adapt=adaptSteps ,
                            burnin=burnInSteps ,
                            sample=ceiling(numSavedSteps/nChains) ,
                            thin=thinSteps ,
                            summarise=FALSE ,
                            plots=FALSE )
    codaSamples = as.mcmc.list( runJagsOut )
  } else { …
```

On my computer, the resulting time was this:

```
   user  system elapsed
   2.18    0.50   34.77
```

which means that 34.77 seconds elapsed to run the chains in parallel.

The diagnostic plots on omega show ESS=17,132 and look like this:



(B) Set the program `Jags-Ydich-XnomSsubj-MbinomBetaOmegaKappa.R` to use rjags (with a single long chain). Be sure to save the program after any changes. Then run the high-level script, `Jags-Ydich-XnomSsubj-Mbinom BetaOmega Kappa-Example.R`. Be sure it has `proc.time()` before and after `genMCMC` so that you can monitor how long it takes to generate the MCMC chain. Report the elapsed time of the chain generation, and also include the chain-diagnostic graph of omega, which includes its ESS.

To run the chains serially, we set the runjags method to "rjags" like this:

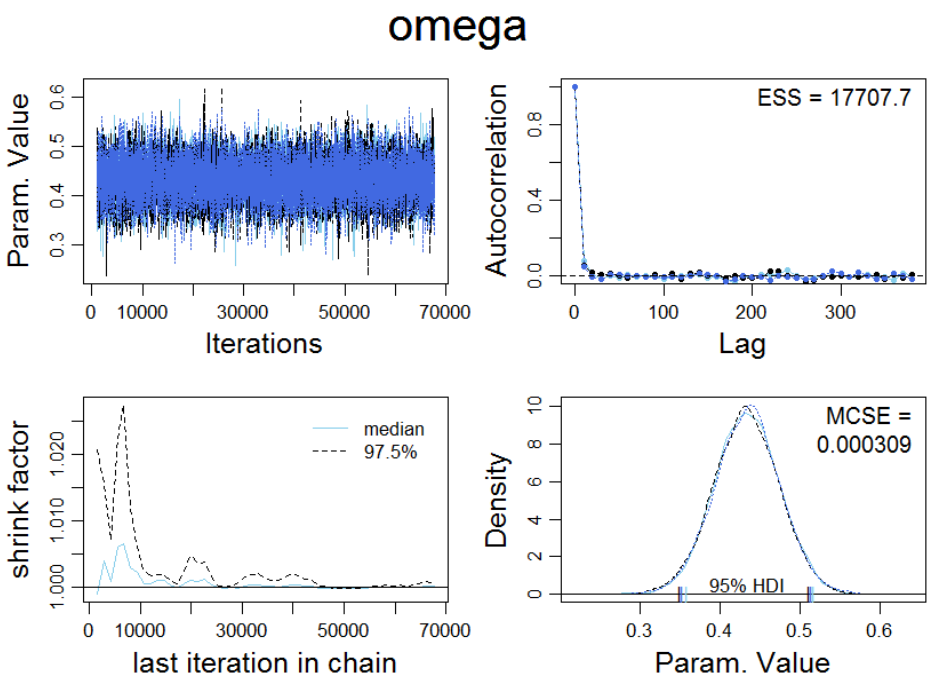
```
runJagsOut <- run.jags( method=c("rjags","parallel")[1] ,
```

The resulting duration is shown as:

```
   user  system elapsed
  76.86    0.06   77.61
```

which means that 77.61 seconds elapsed.

The diagnostic plot for omega shows ESS=17,708, which is essentially the same as parallel chains, and looks like this:



(C) Compare the two runs. What is the difference in run durations? Are the ESSs about the same?

77.61 seconds versus 34.77 seconds means that 3 parallel chains take less than half the time of 3 serial chains. Thus, running chains in parallel can save a lot of time while producing the same quality of result.

Chapter 10

Exercise 10.1

Exercise 10.1. [Purpose: To illustrate the fact that models with more distinctive predictions can be more easily discriminated.] Consider the scenario of Section 10.2.1, in which there were two coin factories, one of which was tail-biased and the other head-biased. Suppose we flip a coin that we know is from one of the two factories but we do not know which factory, and the prior probabilities of the factories are 50/50. The results show $z = 7$ heads in $N = 10$ flips.

(A) If $\omega_1 = 0.25$, $\omega_2 = 0.75$, and $\kappa = 6$, what are the posterior probabilities of the factories?

(B) If $\omega_1 = 0.25$, $\omega_2 = 0.75$, and $\kappa = 202$, what are the posterior probabilities of the factories?

(C) Why are the posterior probabilities so different in parts A and B, even though the modes of the factories are the same?

(A) R code and result:

```
# Define function for computing the marginal likelihood, as on p. 270 of book:
pD = function(z,N,a,b) { exp( lbeta(z+a,N-z+b) - lbeta(a,b) ) }
# Specify parameter values of the factories:
omega1 = 0.25
omega2 = 0.75
kappa = 6 # 6 for 10.1A, 202 for 10.1B
# Compute corresponding a,b values:
a1 = omega1*(kappa-2) + 1
b1 = (1-omega1)*(kappa-2) + 1
a2 = omega2*(kappa-2) + 1
b2 = (1-omega2)*(kappa-2) + 1
# Specify the data:
z = 7
N = 10
# Compute the marginal likelihoods:
pDg1 = pD(z,N,a1,b1)
pDg2 = pD(z,N,a2,b2)
# Compute the Bayes factor:
BF12 = pDg1/pDg2
# Specify prior probabilities:
p1 = 0.5
p2 = 1-p1
# Compute posterior probabilities as on p. 271 of book:
BF12xPriorOdds = (pDg1/pDg2)*(p1/p2)
p1gD = BF12xPriorOdds/(1.0+BF12xPriorOdds)
p2gD = 1.0-p1gD
# Display results:
show(BF12)
show(p1gD)
show(p2gD)
```

```
> show(BF12)
[1] 0.3333333
> show(p1gD)
[1] 0.25
> show(p2gD)
[1] 0.75
```

(B) R code is same as for part A, but with kappa=202. Result:

```
> show(BF12)
[1] 0.01621596
> show(p1gD)
[1] 0.0159572
> show(p2gD)
[1] 0.9840428
```

(C) In both cases, the head-biased factory (model 2) is favored because the data showed more heads than tails. But the data are more decisive in the second part (when kappa=202) because the factories make more consistently distinctive coins. In Part B (kappa=202), coins from the first factory have biases always near 0.25 and coins from the second factory have biases always near 0.75. But in Part A (kappa=6), coins from the first factory have biases that could be far from 0.25, even head biased, and coins from the second factory have biases that could be far from 0.75, even tail biased. In the scenario of Part A, data such as z=7,N=10 could have come from either factory with reasonably high probability, but in the scenario of Part B, it's relatively unlikely that the data could have come from the tail-biased factory.

Exercise 10.2

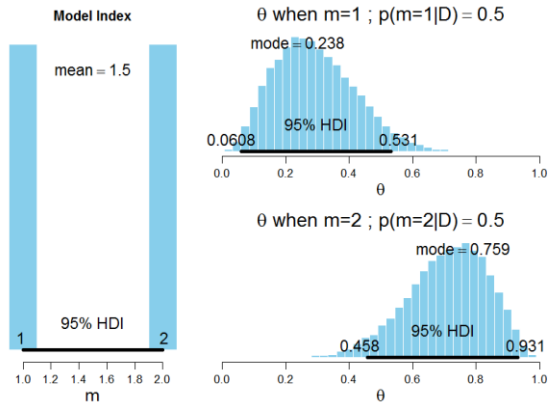
Exercise 10.2. [Purpose: To be sure you really understand the JAGS program for Figure 10.4.]

(A) Use the script Jags-Ydich-Xnom1subj-MbernBetaModelComp.R to reproduce Figure 10.4, including both the prior and the posterior. Explain how you generated the MCMC sample from the prior. Include the graphical output in your answer, which will be slightly different than Figure 10.4 because of randomness in the MCMC chain.

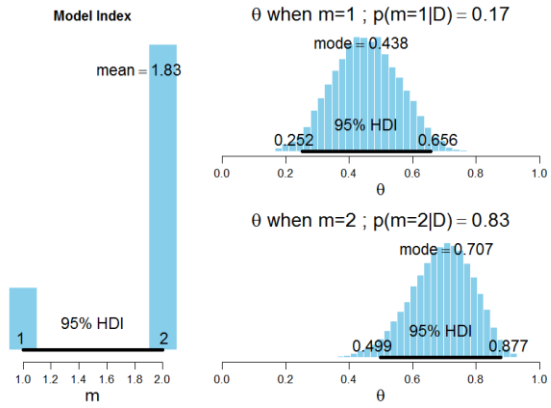
(B) Make a histogram of the θ values collapsed across both models. It should look like the bottom-middle panel of Figure 10.3. Explain why.

(C) Use the script to reproduce the previous exercise. That is, change the data to $z = 7$ heads in $N = 10$, and run the script once with $\kappa = 6$ and once with $\kappa = 202$. Do the MCMC results match the analytical results?

(A) Running the script *with the line y=y commented out of the dataList* yields the following graph for the prior distribution, which matches Figure 10.4 (except for MCMC random variation):



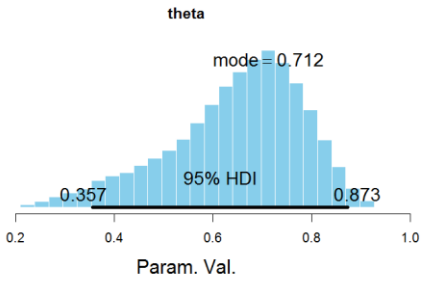
Running the script with the data present (not commented out) produces the following graph, which is the same as Figure 10.4 (except for MCMC random variation):



(B) (For this part, be sure that the original data are being used, not commented out.) After running the script, at the end type these lines into R:

```
> openGraph(height=4,width=7)
> plotPost( theta , main="theta" )
```

The resulting graph looks like this:



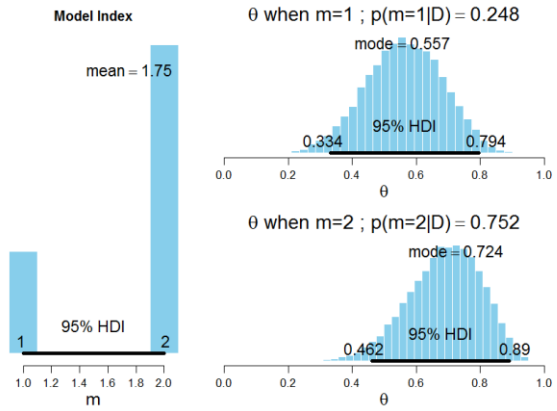
Notice it has the left-tail skew just as in the bottom-middle panel of Figure 10.3.

(C) Modify the script as follows:

```
N=10
z=7
y = c( rep(0,N-z) , rep(1,z) )
dataList = list(
  y = y ,
  N = N
)

#------------------------------------------------------------------------------
# THE MODEL.

modelString = "
model {
  for ( i in 1:N ) {
```

```
    y[i] ~ dbern( theta )
  }
  theta ~ dbeta( omega[m]*(kappa-2)+1 , (1-omega[m])*(kappa-2)+1 )
  omega[1] <- .25
  omega[2] <- .75
  kappa <- 6
  ...
```

Which produces the following graphical output:



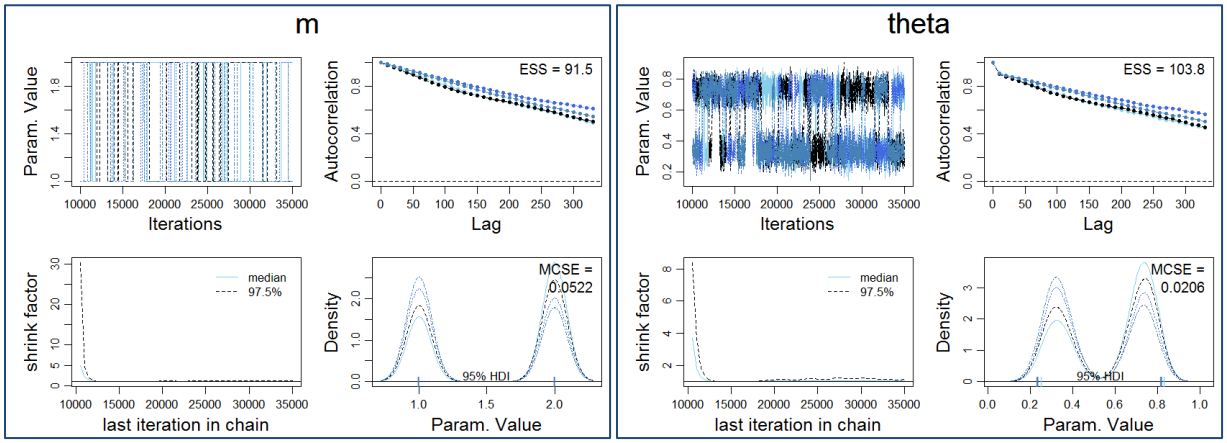
Notice that the posterior on the model index is p(m=2|D) = 0.75, which matches the result of the previous exercise.

Re-running with kappa <- 202 yields the following: The MCMC chain gets stuck in one model or the other. It is very difficult for the MCMC process to jump to the other model because the theta value from one model doesn't work well in the other model. We can coerce the chain to jump a bit by (i) explicitly initializing the chains so they start in both models, (ii) setting the prior probabilities of the models to compensate for the dominance of one model, (iii) using a kappa value that is not as extreme as 202, and (iv) thinning the chains:

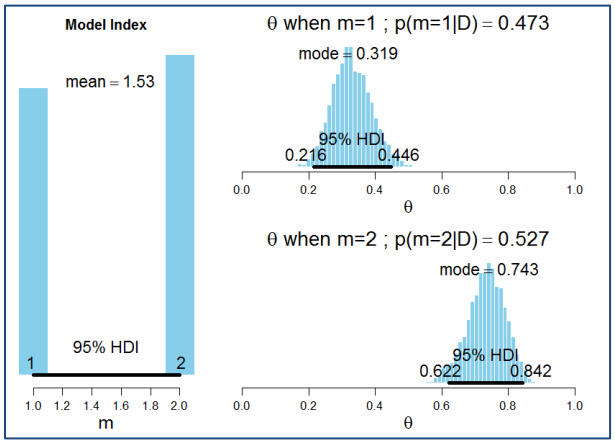
```
modelString = "
model {
  for ( i in 1:N ) {
    y[i] ~ dbern( theta )
  }
  theta ~ dbeta( omega[m]*(kappa-2)+1 , (1-omega[m])*(kappa-2)+1 )
  omega[1] <- .25
  omega[2] <- .75
  kappa <- 52
  m ~ dcat( mPriorProb[] )
  mPriorProb[1] <- .95
  mPriorProb[2] <- .05
}
" # close quote for modelString
writeLines( modelString , con="TEMPmodel.txt" )
#------------------------------------------------------------------------------
# INTIALIZE THE CHAINS.
initsList = list( list(theta=0.5,m=1) ,
                  list(theta=0.5,m=2) ,
                  list(theta=0.5,m=1) ,
                  list(theta=0.5,m=2) )
#------------------------------------------------------------------------------
# RUN THE CHAINS.
```

```
parameters = c("theta","m")
adaptSteps = 10000              # Number of steps to "tune" the samplers.
burnInSteps = 10000            # Number of steps to "burn-in" the samplers.
nChains = 4                    # Number of chains to run.
numSavedSteps=10000           # Total number of steps in chains to save.
thinSteps=10                    # Number of steps to "thin" (1=keep every step).
nPerChain = ceiling( ( numSavedSteps * thinSteps ) / nChains ) # Steps per chain.
# Create, initialize, and adapt the model:
jagsModel = jags.model( "TEMPmodel.txt" , data=dataList , inits=initsList ,
                        n.chains=nChains , n.adapt=adaptSteps )
```

Even with those modifications, the result is still a badly autocorrelated chain:



Notice in the trace plots of theta that the chain stays in one model or the other for long periods.



In the final graph, notice that the posterior probabilities of the model index is based on the priors of 0.95 and 0.05! You can transform the result to 50/50 priors as described near the bottom of p. 287 of the book.

A more effective approach to this problem would be using pseudo-priors, which is the topic of the next exercise.

Exercise 10.3

Exercise 10.3. [Purpose: To get some hands-on experience with pseudo-priors.]

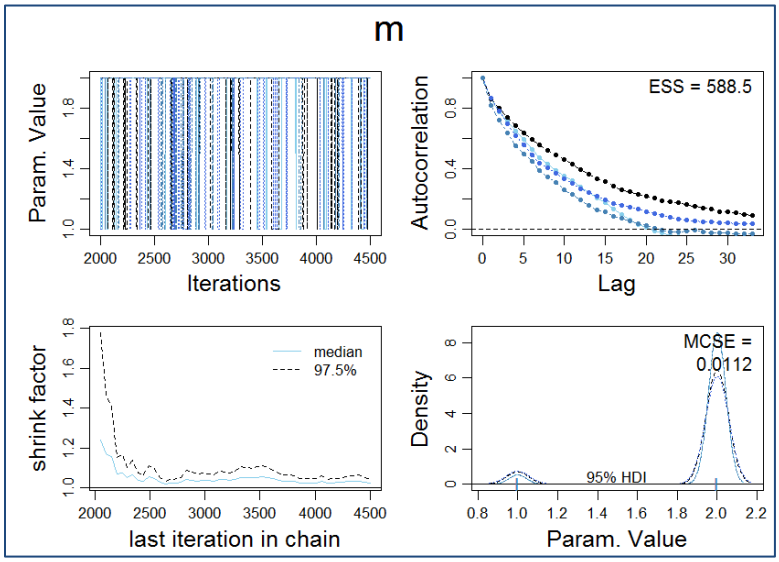
(A) Use the script Jags-Ydich-Xnom1subj-MbernBetaModelCompPseudoPrior.R to reproduce Figures 10.5 and 10.6. That is, run the script once with the pseudo-prior set to the true prior, and then again with the pseudo-prior set to the values shown in the text. Include the graphical output of the chain diagnostics on the model index. Your results will differ slightly from Figures 10.5 and 10.6 because of randomness in the MCMC chain.

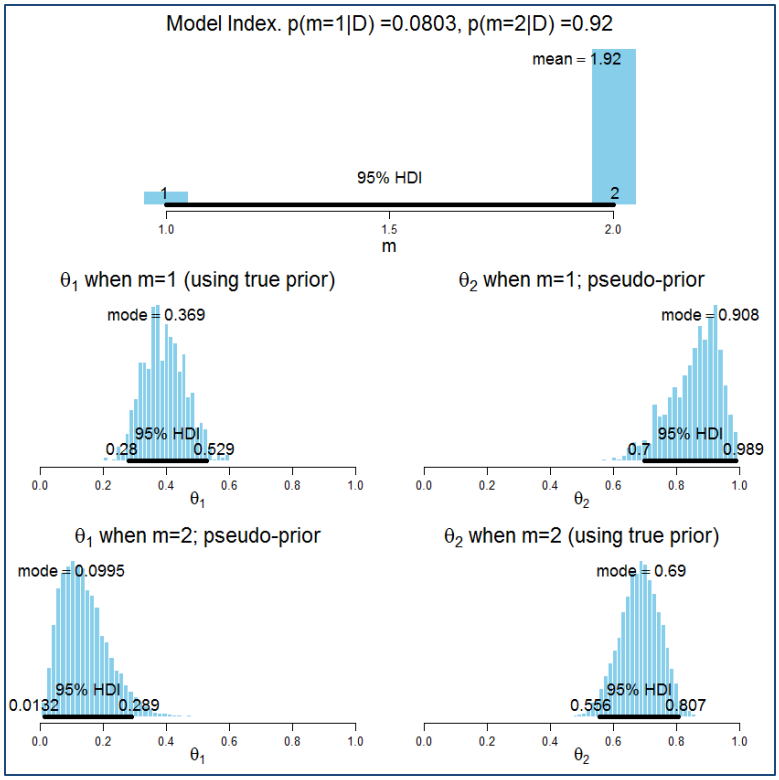
(B) Change the pseudo-prior to broad distributions, with `omega1[2]-omega2[1]-0.5` and `kappa1[2]-kappa2[1]-2.1`. Run the script and report the results, including the chain diagnostic on the model index. Discuss.

(A) With the pseudo priors set at the true priors, as highlighted below,

```
model {
  for ( i in 1:N ) {
    y[i] ~ dbern( theta )
  }
  theta <- equals(m,1)*theta1 + equals(m,2)*theta2
  theta1 ~ dbeta( omega1[m]*(kappa1[m]-2)+1 , (1-omega1[m])*(kappa1[m]-2)+1 )
  omega1[1] <- .10 # true prior value
  omega1[2] <- .10 # pseudo prior value
  kappa1[1] <- 20 # true prior value
  kappa1[2] <- 20 # pseudo prior value
  theta2 ~ dbeta( omega2[m]*(kappa2[m]-2)+1 , (1-omega2[m])*(kappa2[m]-2)+1 )
  omega2[1] <- .90 # pseudo prior value
  omega2[2] <- .90 # true prior value
  kappa2[1] <- 20 # pseudo prior value
  kappa2[2] <- 20 # true prior value
  m ~ dcat( mPriorProb[] )
  mPriorProb[1] <- .5
  mPriorProb[2] <- .5
}
```

the result closely resembles Figure 10.5:

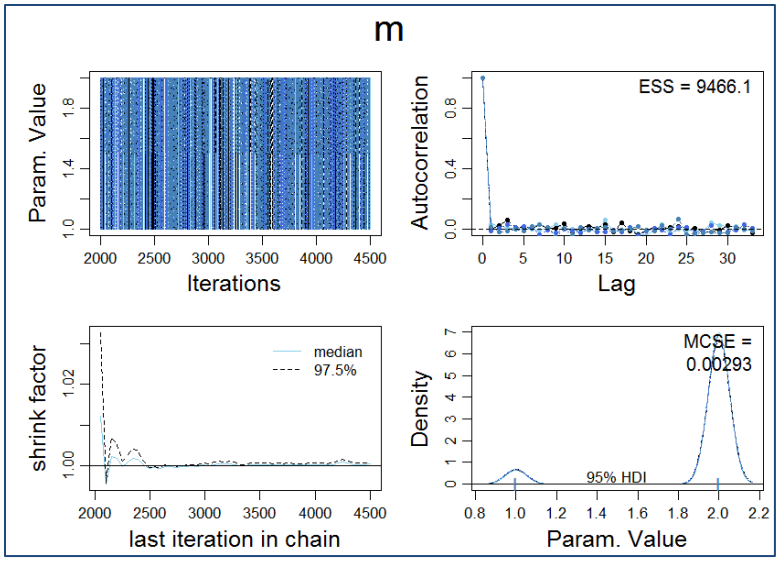


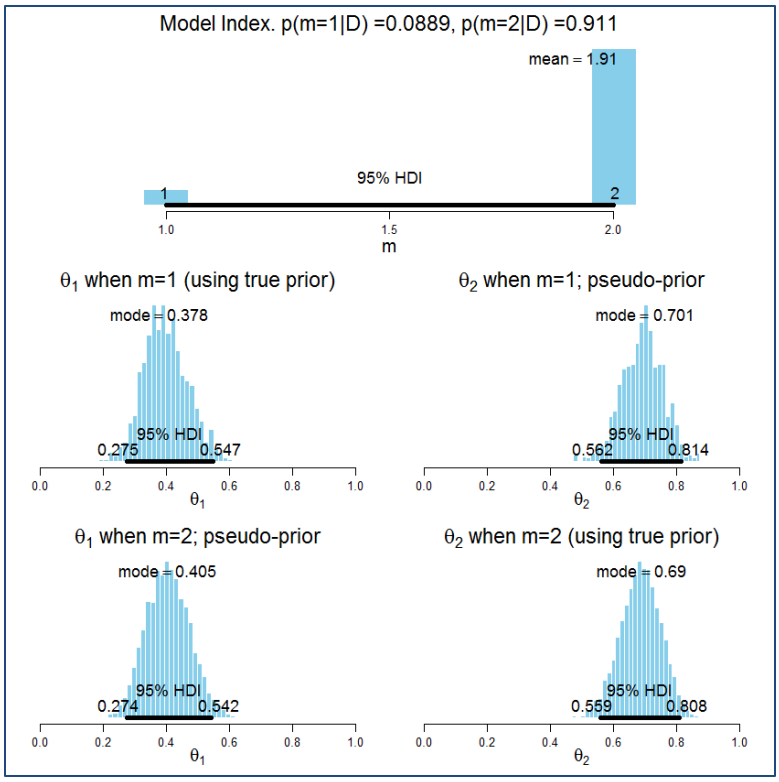


With the pseudo priors set to mimic the posterior, as highlighted below,

```
model {
  for ( i in 1:N ) {
    y[i] ~ dbern( theta )
  }
  theta <- equals(m,1)*theta1 + equals(m,2)*theta2
  theta1 ~ dbeta( omega1[m]*(kappa1[m]-2)+1 , (1-omega1[m])*(kappa1[m]-2)+1 )
  omega1[1] <- .10 # true prior value
  omega1[2] <- .40 # pseudo prior value
  kappa1[1] <- 20 # true prior value
  kappa1[2] <- 50 # pseudo prior value
  theta2 ~ dbeta( omega2[m]*(kappa2[m]-2)+1 , (1-omega2[m])*(kappa2[m]-2)+1 )
  omega2[1] <- .70 # pseudo prior value
  omega2[2] <- .90 # true prior value
  kappa2[1] <- 50 # pseudo prior value
  kappa2[2] <- 20 # true prior value
  m ~ dcat( mPriorProb[] )
  mPriorProb[1] <- .5
  mPriorProb[2] <- .5
}
```

the result closely resembles Figure 10.6:



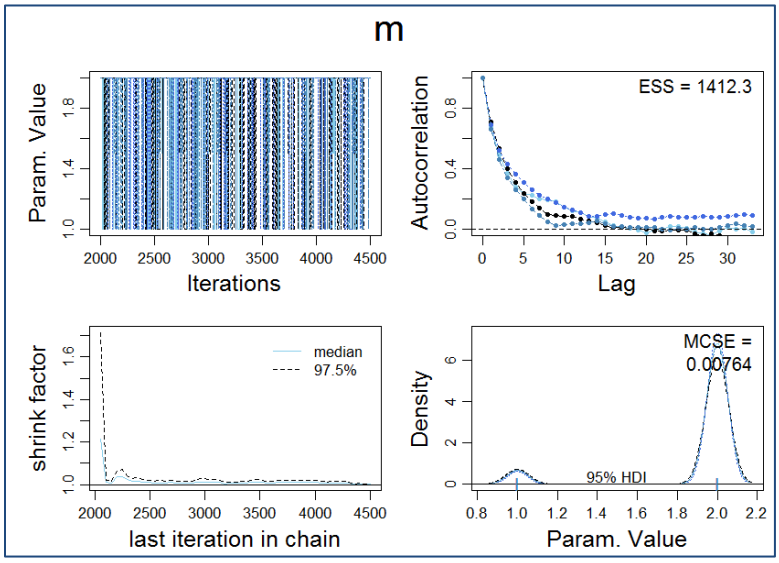


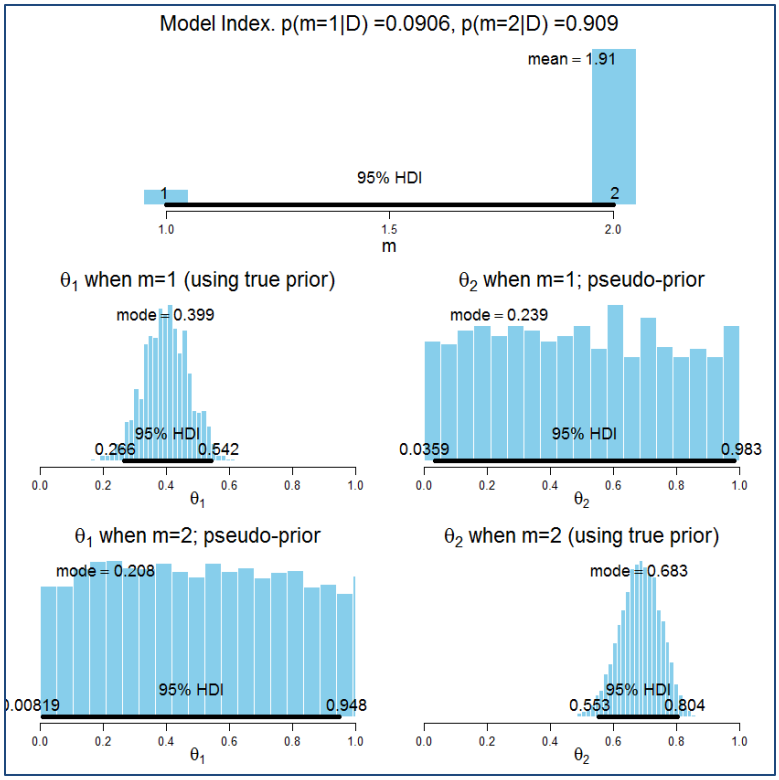
As emphasized in the text, when the pseudo priors resemble the posterior, there is far less autocorrelation and higher ESS.

(B) With broad pseudo priors, set as highlighted below,

```
model {
  for ( i in 1:N ) {
    y[i] ~ dbern( theta )
  }
  theta <- equals(m,1)*theta1 + equals(m,2)*theta2
  theta1 ~ dbeta( omega1[m]*(kappa1[m]-2)+1 , (1-omega1[m])*(kappa1[m]-2)+1 )
  omega1[1] <- .10 # true prior value
  omega1[2] <- .50 # pseudo prior value
  kappa1[1] <- 20 # true prior value
  kappa1[2] <- 2.1 # pseudo prior value
  theta2 ~ dbeta( omega2[m]*(kappa2[m]-2)+1 , (1-omega2[m])*(kappa2[m]-2)+1 )
  omega2[1] <- .50 # pseudo prior value
  omega2[2] <- .90 # true prior value
  kappa2[1] <- 2.1 # pseudo prior value
  kappa2[2] <- 20 # true prior value
  m ~ dcat( mPriorProb[] )
  mPriorProb[1] <- .5
  mPriorProb[2] <- .5
}
```

the result is





With the broad pseudo prior, we see that the autocorrelation and ESS is intermediate between the poor result from the true prior and the good result from the posterior-mimicking pseudo-prior. This makes sense because the broad pseudo-prior has sub-regions that cover both the true prior and the posterior. Because of the sub-regions that cover the posterior, the broad pseudo-prior has better opportunity to jump between model indices than the true prior.

Chapter 11

Exercise 11.1

Exercise 11.1. [Purpose: To compute p values for stopping at fixed N and fixed z.] We have a six-sided die, and we want to know whether the probability that the six-dotted face comes up is fair. Thus, we are considering two possible outcomes: six-dots or not six-dots. If the die is fair, the probability of the six-dotted face is 1/6.

(A) Suppose we roll the die $N = 45$ times, intending to stop at that number of rolls. Suppose we get 3 six-dot rolls. What is the two-tailed p value?

Hints: Use Equation 11.5 (p. 303) to compute the tail probability of the binomial sampling distribution in R. R has various relevant functions built in, such as `factorial`, `choose`, and even `dbinom`.[9] To maintain correspondence with Equation 11.5, I will not use `dbinom`. Try this script:

```
N = 45 ; z = 3 ; theta = 1/6
lowTailZ = 0:z
sum( choose(N,lowTailZ) * theta^lowTailZ * (1-theta)^(N-lowTailZ) )
```

Explain carefully what each line of the script does. Why does it consider the low tail and not the high tail? Explain the meaning of the final result.

```
> # Data:
> N = 45 ; z = 3
> # p(y=1):
> theta = 1/6
>
> # Consider the low tail because z/N = 3/45 is less than expected p=1/6:
> lowTailZ = 0:z
> # Cumulative low tail probability:
> lowTailP = sum( choose(N,lowTailZ) * theta^lowTailZ * (1-theta)^(N-lowTailZ) )
> # Two-tail probability:
> TwoTailP = 2 * lowTailP
> show( TwoTailP )
[1] 0.08920334
```

The final result (above) is the two-tailed p value, which is greater than .05, meaning that we would *not* reject the hypothesis that theta=1/6.

(B) Suppose that instead of stopping at fixed N, we stop when we get 3 six-dot outcomes. It takes 45 rolls. (Notice this is the same result as the previous part.) What is the two-tailed p value?

Hint: Use Equation 11.6 (p. 306). Try this:

```
sum( (lowTailZ/N) * choose(N,lowTailZ) * theta^lowTailZ * (1-theta) ^(N-lowTailZ) )
```

Explain carefully what that code does and what its result means.

The Hint provides the form of the negative binomial function, but we need to be careful about how we sum over the distribution. See Footnote 2, p. 307.

```
> # Cumulative low tail probability is now given by negative binomial, but now the
> # tail is over n>=N, which is computed as 1-p(n<N), as explained in Footnote 2,
> # p. 307.
> complN = z:(N-1)
> complP = sum( (z/complN) *
+                 choose(complN,z) * theta^z * (1-theta)^(complN-z) )
> lowTailP = 1-complP
```

```
> # Two-tail probability:
> TwoTailP = 2 * lowTailP
> show( TwoTailP )
[1] 0.03126273
```

The final result above is the two-tailed p value, which is less than .05, which means we *do* reject the hypothesis that theta=1/6. Notice that this is a different conclusion than part (A), because we used a different stopping intention to establish the sampling distribution.

Exercise 11.2

Exercise 11.2. [Purpose: To determine NHST CIs, and notice that they depend on the experimenter's intention.] We continue with the scenario of the previous exercise: A dichotomous outcome, with $N = 45$ and $z = 3$.

(A) If the intention is to stop when $N = 45$, what is the 95% CI?

Hints: Try this continuation of the R script from the previous exercise:

```
for ( theta in seq( 0.170 , 0.190 , 0.001) ) {
  show( c(
    theta ,
    2*sum( choose(N,lowTailZ) * theta^lowTailZ * (1-theta)^(N-lowTailZ) )
  ))
}

highTailZ = z:N
for ( theta in seq( 0.005 , 0.020 , 0.001) ) {
  show( c(
    theta ,
    2*sum( choose(N,highTailZ) * theta^highTailZ * (1-theta)^(N-highTailZ) )
  ))
}
```

Explain carefully what the code does and what it means!

The confidence interval is that range of parameter values not rejected (i.e., two-tail p>.05 or one-tail p>.025). The R code increments through candidate theta values and computes the two-tail p value for the appropriate tail.

```
> N = 45 ; z = 3
> theta = 1/6
# z/N = 3/45 = 0.06666…
# For candidate theta values from 0.170 to 0.190, which are greater than z/N observed,
# compute the left-tail p value:
> lowTailZ = 0:z
> for ( theta in seq( 0.170 , 0.190 , 0.001) ) {
+   show( c(
+     theta ,
+     2*sum( choose(N,lowTailZ) * theta^lowTailZ * (1-theta)^(N-lowTailZ) )
+   ))
+ }
# The columns are theta and p value:
[1] 0.170000 0.079301
[1] 0.17100000 0.07652818
[1] 0.17200000 0.07384245
[1] 0.17300000 0.07124149
[1] 0.17400000 0.06872304
[1] 0.17500000 0.06628487
[1] 0.17600000 0.06392479
[1] 0.17700000 0.06164067
[1] 0.17800000 0.05943039
[1] 0.17900000 0.05729192
[1] 0.18000000 0.05522324
```

```
[1] 0.18100000 0.05322237
[1] 0.1820000 0.0512874 # This is the highest theta that is not rejected.
[1] 0.18300000 0.04941643
[1] 0.18400000 0.04760762
[1] 0.18500000 0.04585918
[1] 0.18600000 0.04416934
[1] 0.18700000 0.04253637
[1] 0.18800000 0.04095861
[1] 0.1890000 0.0394344
[1] 0.19000000 0.03796215
# For candidate theta values from 0.005 to 0.020, which are less than z/N observed,
# compute the right-tail p value:
> highTailZ = z:N
> for ( theta in seq( 0.005 , 0.020 , 0.001) ) {
+   show( c(
+     theta ,
+     2*sum( choose(N,highTailZ) * theta^highTailZ * (1-theta)^(N-highTailZ) )
+   ))
+ }
# The columns are theta and p value:
[1] 0.005000000 0.003032142
[1] 0.006000000 0.005078266
[1] 0.007000000 0.007816233
[1] 0.00800000 0.01130928
[1] 0.00900000 0.01560897
[1] 0.01000000 0.02075627
[1] 0.01100000 0.02678246
[1] 0.01200000 0.03371014
[1] 0.01300000 0.04155399
[1] 0.0140000 0.0503216 # This is the lowest theta that is not rejected
[1] 0.01500000 0.06001419
[1] 0.01600000 0.07062725
[1] 0.01700000 0.08215124
[1] 0.01800000 0.09457208
[1] 0.0190000 0.1078717
[1] 0.0200000 0.1220287
```

Therefore the confidence interval, which is the range of theta values not rejected, goes from 0.014 to 0.182, when the stopping intention is fixed N.

(B) If the intention is to stop when $z = 3$, what is the 95% CI? Is the CI the same as for stopping when $N = 45$?

Hint: Modify the R script of the previous part for use with stopping at z, like the second part of the previous exercise.

```
> # For candidate theta values GREATER than z/N observed, compute the LEFT-tail p
> # value:
> complN = z:(N-1)
> for ( theta in seq( 0.150 , 0.160 , 0.001) ) {
+   show( c(
+     theta ,
+     2*(1-sum( (z/complN) * choose(complN,z) * theta^z * (1-theta)^(complN-z) ) )
+   ) )
+ }
[1] 0.15000000 0.05995277
[1] 0.15100000 0.05770908
[1] 0.15200000 0.05554267
[1] 0.15300000 0.05345117
[1] 0.15400000 0.05143228 # This is the highest theta not rejected
[1] 0.15500000 0.04948376
[1] 0.15600000 0.04760341
[1] 0.15700000 0.04578911
[1] 0.15800000 0.04403879
[1] 0.15900000 0.04235041
[1] 0.16000000 0.04072202
>
```

```
> # For candidate theta values LESS than z/N observed, compute the RIGHT-tail p
> # value:
> highTailN = z:N # Notice N not N-1
> for ( theta in seq( 0.005 , 0.020 , 0.001) ) {
+   show( c(
+     theta ,
+     2*sum(
+       (z/highTailN) * choose(highTailN,z) * theta^z * (1-theta)^(highTailN-z)
+       )
+   ) )
+ }
[1] 0.005000000 0.003032142
[1] 0.006000000 0.005078266
[1] 0.007000000 0.007816233
[1] 0.00800000 0.01130928
[1] 0.00900000 0.01560897
[1] 0.01000000 0.02075627
[1] 0.01100000 0.02678246
[1] 0.01200000 0.03371014
[1] 0.01300000 0.04155399
[1] 0.0140000 0.0503216  # This is the lowest theta not rejected.
[1] 0.01500000 0.06001419
[1] 0.01600000 0.07062725
[1] 0.01700000 0.08215124
[1] 0.01800000 0.09457208
[1] 0.0190000 0.1078717
[1] 0.0200000 0.1220287
```

Thus, when the stopping at fixed z, the confidence interval goes from 0.014 to 0.154. This is quite different than when stopping at fixed N.

Notice that the hypothetical value of theta=1/6=0.1666… falls outside the fixed-z confidence interval, but falls inside the fixed-N confidence interval.

Exercise 11.3

Exercise 11.3. [Purpose: To determine the p value when data collection stops at a fixed duration.] (For another example of NHST for fixed-duration samples, see Kruschke, 2010.) We continue with the scenario of the previous exercises: A dichotomous outcome, with $N = 45$ and $z = 3$. Suppose that the die-roller of the previous exercises stopped rolling because time expired at 6 min. For simplicity, suppose that during a 6-min interval, the roller could have rolled $N = 40$, or $N = 41$, or $N = 42$, through $N = 50$, with equal probability. What is the p value for the observed outcome? Is it the same p value as when assuming fixed N or fixed z?

Hints: We need to compute the p value for each possible N, and then average them according to the probability they would happen. For each N, the low tail consists of outcomes that are a proportion less than or equal to the observed $z/N = 3/45$. Examine the follow R script. Explain exactly what it does and interpret its output.

```
N = 45 ; z = 3 ; theta = 1/6
# Specify possible N values:
Nposs = 40:50
# Specify probability of each N (here all equal):
Nprob = rep(1,length(Nposs)) ; Nprob = Nprob/sum(Nprob)
# For each possible N, compute p value, and compute the weighted total p:
totalP = 0
for ( i in 1:length(Nposs) ) {
  thisN = Nposs[i]
  # For this N, determine the max z that is in the low tail:
  thisZ =  max( (0:thisN)[ (0:thisN)/thisN <= z/N ] )
  lowTailZ = 0:thisZ
  thisP = 2*sum( choose(thisN,lowTailZ) * theta^lowTailZ * (1-theta) ^(thisN-lowTailZ) )
  totalP = totalP + Nprob[i] * thisP
  show( c( thisN , thisP ) )
}
show( totalP )
```

The comments in the code explain the procedure:

```
> # Data:
> N = 45 ; z = 3
> # Hypothetical value of parameter:
> theta = 1/6
> # Specify possible N values:
> Nposs = 40:50
> # Specify probability of each N (here all equal):
> Nprob = rep(1,length(Nposs)) # All Nposs get relative probability of 1.
> Nprob = Nprob/sum(Nprob)     # Normalize to get actual probability mass.
> # Initialize total tail probability to zero.
> totalP = 0
> # For each N, compute its p value, and accumulate the weighted total p:
> for ( i in 1:length(Nposs) ) {
+   # For convenience, rename the N that is presently being considered:
+   thisN = Nposs[i]
+   # For this N, determine the max z that is in the low tail.
+   # It must satisfy thisZ/thisN <= z/N.
+   thisZ = max( (0:thisN)[ (0:thisN)/thisN <= z/N ] )
+   # Now compute tail probability, i.e., sum of binomial probabilities from z/N
+   # down to zero.
+   lowTailZ = 0:thisZ
+   thisP = 2*sum(
+     choose(thisN,lowTailZ) * theta^lowTailZ * (1-theta)^(thisN-lowTailZ) )
+   # Accumulate to totalP the value of thisP weight by the probability of thisN:
+   totalP = totalP + Nprob[i] * thisP
+   # Display progress through the loop:
+   show( c( thisN , thisP ) )
+ }
```

```
# The first column below is thisN, the second column is 2 times the left tail
# probability:
[1] 40.00000000  0.05470238
[1] 41.00000000  0.04762645
[1] 42.00000000  0.04142745
[1] 43.00000000  0.03600333
[1] 44.00000000  0.03126273
[1] 45.00000000  0.08920334 # Notice this matches the result of Exercise 11.1(A)
[1] 46.00000000  0.07885681
[1] 47.00000000  0.06963315
[1] 48.00000000  0.0614227
[1] 49.00000000  0.05412451
[1] 50.00000000  0.04764606
> # Display the final result:
> show( totalP )
[1] 0.05562808
```

The final result, above, shows the weighted average of the column of tail probabilities, with each row weighted by the probability of thisN (which in this case is 1/11 for every value of thisN). Notice that the fixed-duration p=0.05562808 is *not* equal to the fixed-N p=0.08920334.

Chapter 12

Exercise 12.1

Exercise 12.1. [Purpose: To make sure you understand the Bayes' factors regarding a single coin in Figure 12.3 and Equation 12.4, including the Savage-Dickey method.] Find the file BernBeta.R in the programs that accompany this book. Open RStudio with the folder of that file as R's working directory. Source the file so that R knows about the function `BernBeta`:

```
source("BernBeta.R")
```

Now, suppose we have a coin that is flipped 24 times and shows 7 heads. Enter these data into R, like this:

```
z=7 ; N=24
```

(A) According to the spike null hypothesis, for which the only credible value of θ is 0.5, what is the probability of the data? *Hint:* It is $\theta^z(1-\theta)^{(N-z)}$. Compute the value.

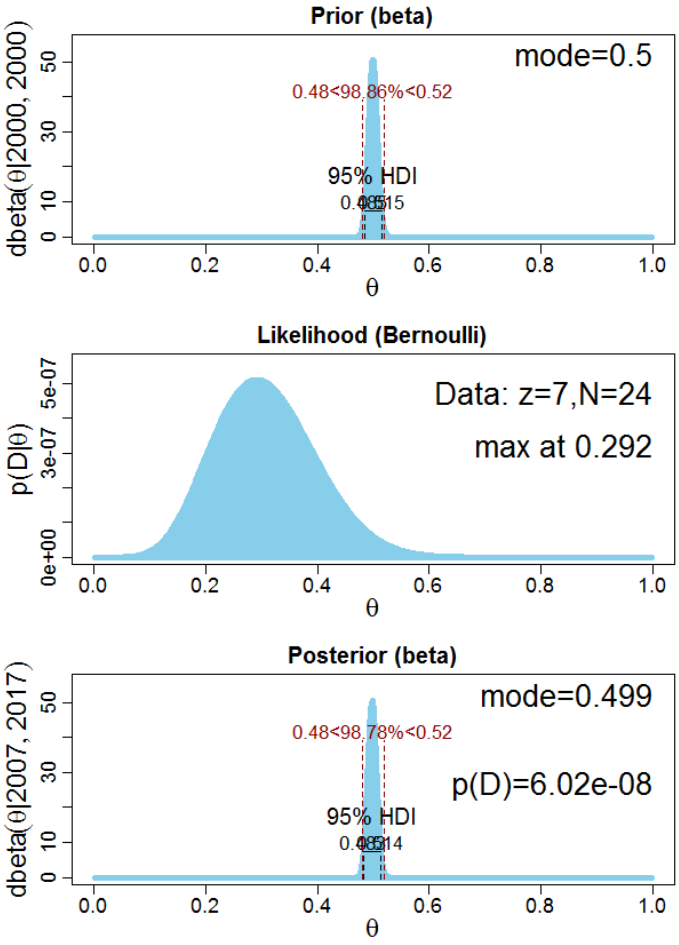
```
source("BernBeta.R")
z = 7 ; N = 24
theta = 0.5
pDgTheta = theta^z * (1-theta)^(N-z)
print( pDgTheta )
# Result is 5.96e-08
```

(B) Verify the result of the previous part by approximating a spike prior with a narrow beta distribution. Use the `BernBeta` function with a beta(θ|2000, 2000) prior, like this:

```
a=2000 ; b=2000
openGraph(width=5,height=7)
BernBeta( c(a,b) , c(rep(0,N-z),rep(1,z)) ,  ROPE=c(0.48,0.52) ,
          plotType="Bars" , showCentTend="Mode" , showHDI=TRUE , showpD=TRUE )
```

Include the resulting graph in your report. What is the value of $p(D)$ for this prior? Is it very close to the value computed for the exact spike prior in the previous part of this exercise? (It should be.) Explain why they are not exactly equal.

```
a = 2000 ; b = 2000
openGraph(width=5,height=7)
BernBeta( c(a,b) , c(rep(0,N-z),rep(1,z)) , ROPE=c(0.48,0.52) ,
          plotType="Bars" , showCentTend="Mode" , showHDI=TRUE , showpD=TRUE )
# Result is p(D)=6.02e-08, displayed in graph
```

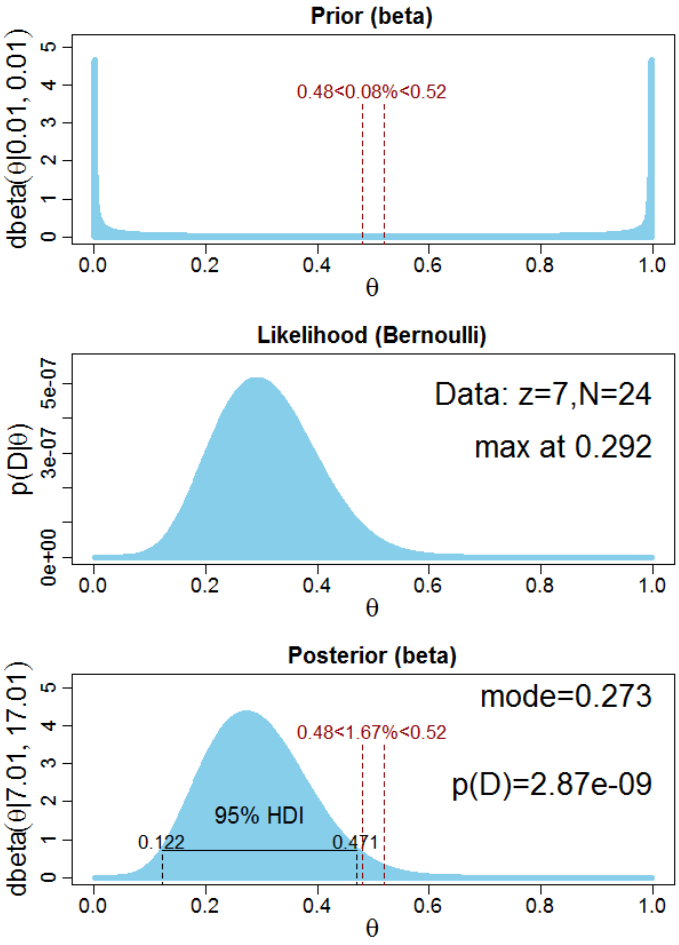


(C) Show the result when using a nearly Haldane prior, like this:

```
a=0.01 ; b=0.01
openGraph(width=5,height=7)
BernBeta( c(a,b) , c(rep(0,N-z),rep(1,z)) ,  ROPE=c(0.48,0.52) ,
          plotType="Bars" , showCentTend="Mode" , showHDI=TRUE ,showpD=TRUE )
```

Include the resulting graph in your report. What is the value of $p(D)$ for this prior? Compute and report the Bayes' factor of this prior relative to the spike (null) prior, using the formula $p(D|\text{Haldane})/p(D|\text{null})$.

```
a = 0.01 ; b = 0.01
openGraph(width=5,height=7)
BernBeta( c(a,b) , c(rep(0,N-z),rep(1,z)) , ROPE=c(0.48,0.52) ,
          plotType="Bars" , showCentTend="Mode" , showHDI=TRUE , showpD=TRUE )
# Result is p(D)=2.87e-09, displayed in graph
# Bayes factor, Haldane/null:
print( 2.87e-09 / 5.96e-08 )
# Result is 0.048
```



(D) Continuing with the Haldane prior from the previous part, compute the approximate Bayes' factor using the Savage-Dickey method. That is, compute and report the ratio of percentage of prior within the ROPE over percentage of posterior with the ROPE.

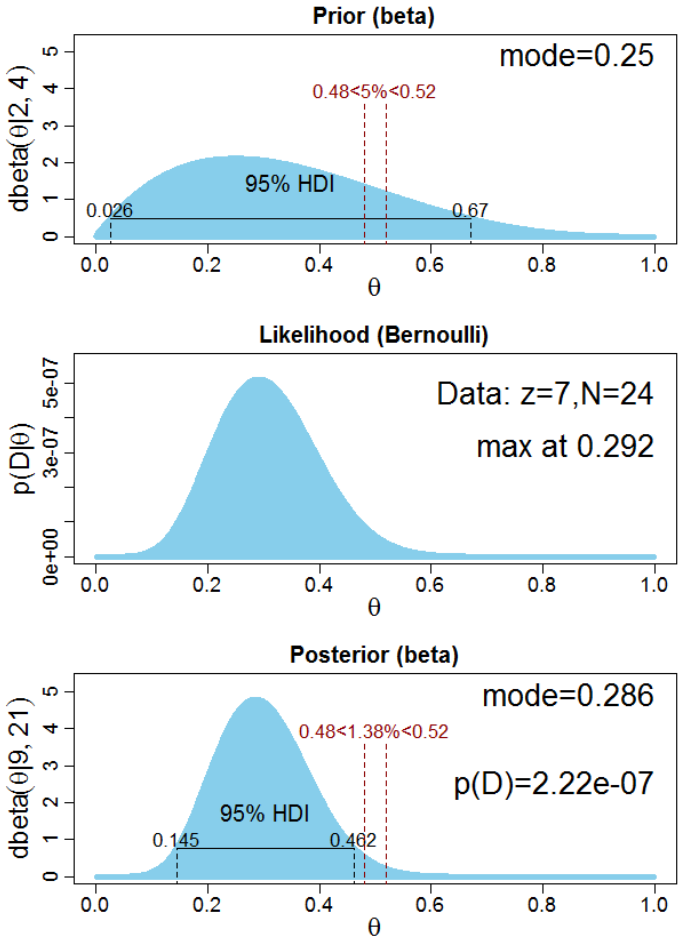
```
# From the graph using the Haldane prior, prior p in ROPE is 0.08%, and
# posterior p in ROPE is 1.67%. The ratio is
print( 0.08 / 1.67 )
# Result is 0.048. Notice this is very close to the previous answer.
```

(E) Suppose we have previous knowledge that in this application there tend to be more tails than heads. Show the result when using a mildly informed prior, like this:

```
a=2 ; b=4
openGraph(width=5,height=7)
BernBeta( c(a,b) , c(rep(0,N-z),rep(1,z)) ,  ROPE=c(0.48,0.52) ,
          plotType="Bars" , showCentTend="Mode" , showHDI=TRUE , showpD=TRUE )
```

Include the resulting graph in your report. What is the value of $p(D)$ for this prior? Compute and report the Bayes' factor of this prior relative to the spike (null) prior, using the formula $p(D|\text{informed})/p(D|\text{null})$.

```
a = 2 ; b = 4
openGraph(width=5,height=7)
BernBeta( c(a,b) , c(rep(0,N-z),rep(1,z)) , ROPE=c(0.48,0.52) ,
          plotType="Bars" , showCentTend="Mode" , showHDI=TRUE , showpD=TRUE )
# Result is p(D)=2.22e-07, displayed in graph
# Bayes factor, informed/null:
print( 2.22e-07 / 5.96e-08 )
# Result is 3.72
```



(F) Continuing with the mildly informed prior from the previous part, compute the approximate Bayes' factor using the Savage-Dickey method. That is, compute and report the ratio of percentage of prior within the ROPE over percentage of posterior with the ROPE.

```
# From the graph using the informed prior, prior p in ROPE is 5.0%, and
# posterior p in ROPE is 1.38%. The ratio is
print( 5.0 / 1.38 )
# Result is 3.62. Notice this is very close to the previous answer.
```

(G) Report the 95% HDIs when starting with the Haldane prior and the mildly informed prior. Are the HDIs very different? Were the Bayes' factors very different?

```
# From the graph using the Haldance prior, the posterior HDI is 0.122 to 0.471.
# From the graph using the informed prior, the posterior HDI is 0.145 to 0.462.
# The posterior HDI's are not very different and both exclude the ROPE.
# The BF's, on the other hand, are very different and make opposite decisions.
```

(H) Which approach, model comparison or estimation, seems most informative? Why? Within the model-comparison approach, which prior, uninformed Haldane or mildly informed, seems most meaningful? Why?

```
# Parameter estimation seems more informative than model comparison because
# estimation gives explicit information about the magnitude of the parameter
# that describes the data, while the BF only indicates which prior distribution
# (model index) gets more credibility without saying anything about the
# parameter magnitude. Within the model-comparison approach, the mildly-informed
# prior is more meaningful than the Haldane prior because the Haldane, despite
# being a mathematically motivated default prior, does not express a
# theoretically meaningful hypothesis in this case.
```

Exercise 12.2

Exercise 12.2. [Purpose: Model comparison for different partitions of group modes, using the script of Section 12.2.2.1.] Open the script `OneOddGroupModelComp2E.R`, making sure that R's working directory includes the various utility programs used with this book.

(A) For this part of the exercise, the goal is to reproduce the findings presented in Section 12.2.2.1. First, be sure that the prior probabilities on the models are set to 50/50:

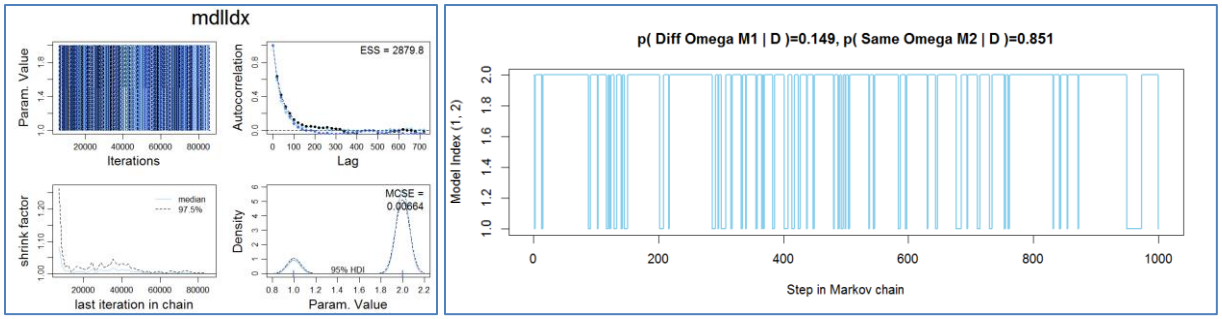
```
modelProb[1] <- 0.5
modelProb[2] <- 0.5
```

Run the script and report the results, including the graphs for the model index and the modes and differences of modes. State what the two models are, and state which model is preferred and by how much. (*Hint:* Model 2 is the single-mode model, and it is preferred.)

The shape parameters of the beta distribution are determined by the model index, as this code snippet shows:

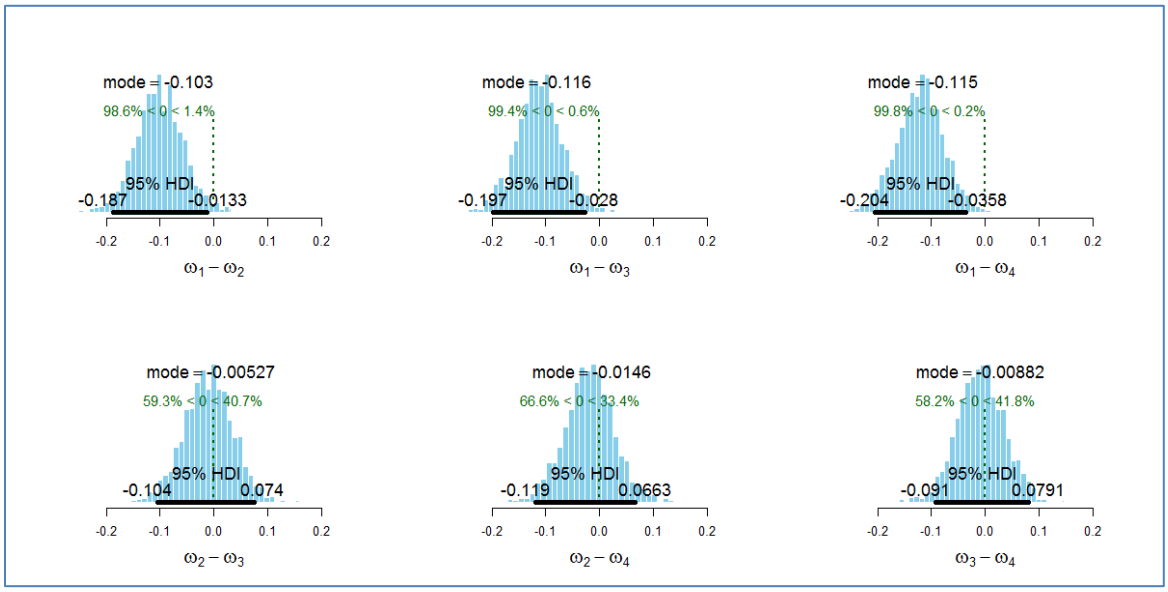
```
# Use omega[j] for model index 1, omega0 for model index 2:
aBeta[j] <-       ( equals(mdlIdx,1)*omega[j]
                  + equals(mdlIdx,2)*omega0  )   * (kappa[j]-2)+1
bBeta[j] <- ( 1 - ( equals(mdlIdx,1)*omega[j]
                  + equals(mdlIdx,2)*omega0  ) ) * (kappa[j]-2)+1
```

When mdlIdx is 1, then omega[j] is used, and when mdlIdx is 2, then omega0 is used.



The plots of the model index, above, indicate that the ESS of model-index chain is about 3,000, which is not too bad for our purposes. Model index 2 is preferred, with a posterior probability of about 85%. In other words, the single-mode model is preferred to the four-mode model.

(B) Continuing with the previous part, consider the graphs of differences of modes. What do they imply about differences between groups? Does this conclusion agree or disagree with the conclusion from the model comparison? How do you reconcile the conclusions? (*Hint:* The model index and the groups modes are all parameters being simultaneously estimated, so there is no contradiction. The parameters answer different questions; which questions?)



Despite the single-mode model being preferred, some differences of modes (e.g., ω_1- ω_4) appear to be noticeably different, as shown by the posterior distribution of the difference, above. This seems to disagree with the conclusion from the model comparison. But there is no real contradiction because the model index and the group modes are being simultaneously estimated. The group modes are close enough together that the single-mode model can account for the data better than the four-mode model when taking into account the prior being diluted over a larger parameter space. The posterior on the model-index addresses the question of which prior better accounts for the data; the posterior on the four mode parameters addresses the question of what are credible estimates of the group modes.

(C) For this part of the exercise, the goal is to compare the single-mode model against a different partitioning of the group modes. Instead of letting each group have its own distinct mode, we will allow a distinct mode for the first group, but restrict groups 2 through 4 to use a single mode. One way to accomplish this is to change this part of the model specification:

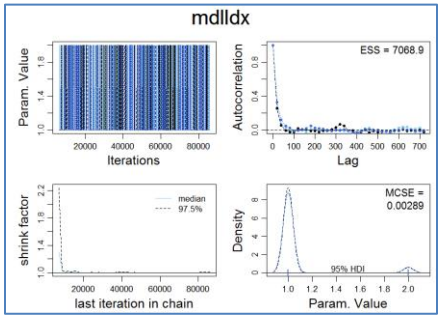
```
for ( j in 1:nCond ) {
  # Use omega[j] for model index 1, omega0 for model index 2:
  aBeta[j] <-       ( equals(mdlIdx,1)*omega[j]
                    + equals(mdlIdx,2)*omega0  )   * (kappa[j]-2)+1
  bBeta[j] <- ( 1 - ( equals(mdlIdx,1)*omega[j]
                    + equals(mdlIdx,2)*omega0  ) ) * (kappa[j]-2)+1
  omega[j] ~ dbeta( a[j,mdlIdx] , b[j,mdlIdx] )
}
```

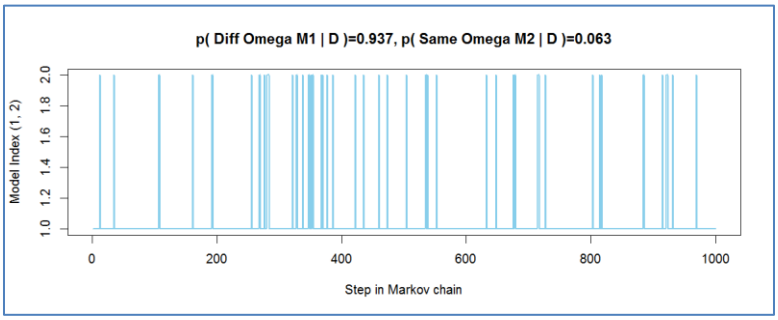
to this:

```
for ( j in 1:nCond ) {
  # Use omega[j] for model index 1, omega0 for model index 2:
  aBeta[j] <-       ( equals(mdlIdx,1)*omega[j]
                    + equals(mdlIdx,2)*omega0  )   * (kappa[j]-2)+1
  bBeta[j] <- ( 1 - ( equals(mdlIdx,1)*omega[j]
                    + equals(mdlIdx,2)*omega0  ) ) * (kappa[j]-2)+1
}
for ( j in 1:2 ) {
  omega[j] ~ dbeta( a[j,mdlIdx] , b[j,mdlIdx] )
}
omega[3] <- omega[2]
omega[4] <- omega[2]
```

In your report, *carefully explain what the change does*. Make the change, and run the script (with the prior probabilities on the models set to 50/50). Report the results, including the graphs for the model index and the modes and differences of modes. State what the two models are, and state which model is preferred and by how much. (*Hint:* Model 2 is the single-mode model, and it is *not* preferred.)

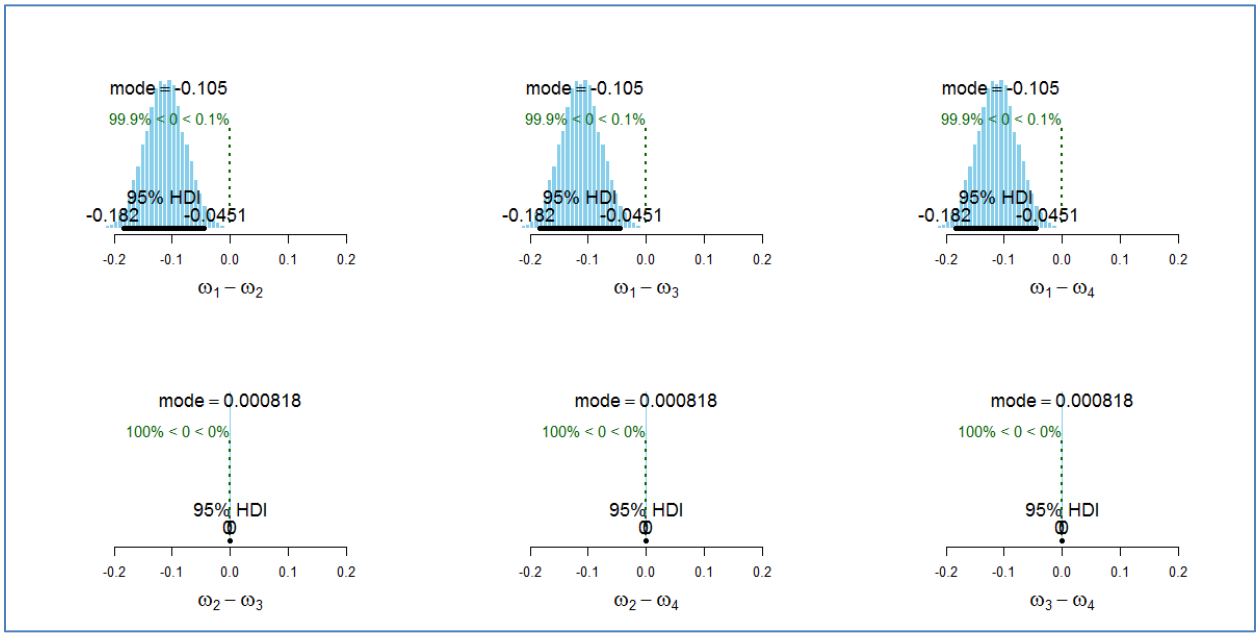
The modified code changes the loop for omega[j]~… Instead of every omega[j] having its own dbeta(a[j,mdlIdx],b[j,mdlIdx) prior, only the first two groups have their own priors, and groups 3 and 4 are fixed to match group 2. In other words, groups 2, 3, and 4 are modeled as having the same mode, distinct from group 1. Model index 1 is the two-mode model, and model index 2 is the single-mode model. Here are the results:





We see above that model index 1 (the two-mode model) is strongly preferred, with a posterior probability of about 94%. Thus, although the single-mode model is preferred to the four-mode model, this particular two-mode model is preferred to the single-mode model.

(D) Continuing with the previous part, consider the graphs of differences of modes. What do they imply about differences between groups? Does this conclusion agree or disagree with the conclusion from the model comparison? Even though Model 1 is preferred, is it really meaningful?



The posterior distribution on ω_1- ω_2 shows that the estimated magnitude of difference clearly excludes zero. But since $\omega_2 = \omega_3 = \omega_4$, the differences of the latter three group modes are exactly zero. Is this really meaningful? Even though this model is preferred to the single-mode model, do we really think that $\omega_2 = \omega_3 = \omega_4$ *exactly*? This seems like an implausible model in the first place, and it should have a small prior probability. On the other hand, so should the single-mode model. It seems we are comparing one implausible model against another implausible model.

(E) Considering the results of the previous parts, what seems to be the most meaningful approach to analyzing differences of groups? (I'm hoping you'll say parameter estimation, not model comparison. But you might give arguments to the contrary.) Can you think of other applications in which model comparison might be more useful?

If it is not plausible that the group modes are truly and exactly identical, then the most meaningful approach is to estimate their magnitudes and make decisions based on the posterior distribution of the modes. N.B.: Model comparison can be useful when both models are plausible and equivalently informed by previous data.

Chapter 13

Exercise 13.1

If you create an answer to this exercise, please notify Prof. Kruschke.

Exercise 13.2

Exercise 13.2. [Purpose: Understanding power for flipping a single coin, in Tables 13.1 and 13.2.] For this exercise, consider flipping a single coin and inferring its bias.

(A) Table 13.2 indicates that when the data-generating distribution is vague, with $\kappa = 10$ and $\omega = 0.80$, then 87 flips are needed for an 80% chance of getting the 95% HDI width to be less than 0.2. What is the minimal N needed if the data-generating distribution is very certain, with $\kappa = 2000$? Show the command you used, and report the exact power for the smallest N that has power greater than 0.8. *Hint:* Change the appropriate argument(s) in `minNforHDIpower(genPriorMode=0.80, genPriorN=10, HDImaxwid=0.2, nullVal=NULL, ROPE=c(0.48,0.52), desiredPower=0.8, audPriorMode=0.5, audPriorN=2, HDImass=0.95, initSampSize=5, verbose=TRUE)`. Don't forget to `source` the function first.

```
source("minNforHDIpower.R")

# For reference (not asked for in exercise), confirm that 87 flips are needed
# for 80% prob that 95%HDI has width less than 0.2 when generating prior has
# kappa=10 and omega=0.80:
sampSize = minNforHDIpower( genPriorMode=0.80, genPriorN=10,
                            HDImaxwid=0.20, nullVal=NULL, ROPE=NULL,
                            desiredPower=0.8,
                            audPriorMode=0.5, audPriorN=2,
                            HDImass=0.95, initSampSize=80, verbose=TRUE )
# Result:
# For sample size = 80, power = 0.6925385
# For sample size = 81, power = 0.7112756
# For sample size = 82, power = 0.7290568
# For sample size = 83, power = 0.7459331
# For sample size = 84, power = 0.7619551
# For sample size = 85, power = 0.7771725
# For sample size = 86, power = 0.7916341
# For sample size = 87, power = 0.8053871

# Now when generating prior has kappa=2000
sampSize = minNforHDIpower( genPriorMode=0.80, genPriorN=2000,
                            HDImaxwid=0.20, nullVal=NULL, ROPE=NULL,
                            desiredPower=0.8,
                            audPriorMode=0.5, audPriorN=2,
                            HDImass=0.95, initSampSize=60, verbose=TRUE )
# Result:
# For sample size = 60, power = 0.5738026
# For sample size = 61, power = 0.5485766
# For sample size = 62, power = 0.6431908
# For sample size = 63, power = 0.619178
# For sample size = 64, power = 0.7055195
# For sample size = 65, power = 0.6833197
```

```
# For sample size = 66, power = 0.760158
# For sample size = 67, power = 0.8239177
```

The result above indicates that the minimal sample size required is 67.

(B) Regarding the previous part, why might a researcher pursue a goal of precision if the data-generating hypothesis is already very precise? *Hint:* The audience prior may be different than the data-generating hypothesis. Discuss briefly, perhaps with an example.

As the hint suggests, the researcher might be privy to prior information that is not cogent for a general audience of the data analysis. Perhaps that researcher has done extensive private or proprietary previous research that gives the researcher a high-certainty prior for generating simulated data, but that previous research has not been published and won't or can't be. Therefore the simulated data can come from the high-certainty generating distribution but the data analysis must use a prior that is appropriate for a skeptical audience that does not have the previous research.

(C) Table 13.1 indicates that when the data-generating distribution is highly certain, with $\kappa = 2000$ and $\omega = 0.80$, then 19 flips are needed for an 80% chance of getting the 95% HDI to exclude a small ROPE around $\theta = 0.5$. What is the minimal N needed if the data-generating distribution is vague, with $\kappa = 2$? Show the command you used, and report the exact power for the smallest N that has power greater than 0.8.

```
# First, verify the table entry:
sampSize = minNforHDIpower( genPriorMode=0.80, genPriorN=2000,
                            HDImaxwid=NULL, nullVal=0.50, ROPE=c(0.48,0.52),
                            desiredPower=0.8,
                            audPriorMode=0.5, audPriorN=2,
                            HDImass=0.95, initSampSize=15, verbose=TRUE )
# Result:
# For sample size = 15, power = 0.6468554
# For sample size = 16, power = 0.796604
# For sample size = 17, power = 0.7565303
# For sample size = 18, power = 0.7146664
# For sample size = 19, power = 0.8350836

# When the generating distribution is vague:
sampSize = minNforHDIpower( genPriorMode=0.80, genPriorN=2,
                            HDImaxwid=NULL, nullVal=0.50, ROPE=c(0.48,0.52),
                            desiredPower=0.8,
                            audPriorMode=0.5, audPriorN=2,
                            HDImass=0.95, initSampSize=130, verbose=TRUE )
# Result:
# For sample size = 130, power = 0.7938931
# For sample size = 131, power = 0.7878788
# For sample size = 132, power = 0.7969925
# For sample size = 133, power = 0.7910448
# For sample size = 134, power = 0.8
```

(D) For the previous part, the goal was for the HDI to exclude the null value (i.e., 0.5). Notice that the goal can be satisfied if the HDI is above the null value *or* if the HDI is below the null value. (i) When the data-generating prior is a beta distribution with $\mu = 0.8$ and $\kappa = 2$, as in the previous part, what proportion of the data-generating biases are greater than the null value? (ii) If the goal is for the HDI to fall entirely *above* the null value, what sample size is needed to achieve a power of 0.8? *Hint:* Use `minNforHDIpower.R` with the argument `ROPE=c(0,0.5)`. Watch the sample size increase indefinitely, with the power creeping toward an asymptote. Why does the power never exceed the proportion you computed for (i)?

(i) When the data-generating distribution has omega=0.8 (not mu, which is a typo) and kappa=2, what proportion of the data-generating distribution falls above the null value of 0.50? Hopefully the answer is intuitively clear, because when kappa=2, the distribution is uniform, and therefore we know that half the distribution falls about 0.5. We can verify in R::

```
> omega=0.8
> kappa=2
> pbeta( 1-0.50 , shape1=(omega)*(kappa-2)+1 , shape2=(1-omega)*(kappa-2)+1 )
[1] 0.5
> pbeta( 1-0.52 , shape1=(omega)*(kappa-2)+1 , shape2=(1-omega)*(kappa-2)+1 )
[1] 0.48
```

The above also shows the result for 0.52, for use in the next subpart.

(ii) Here I use a ROPE of c(0,0.52) instead of c(0,0.50) for consistency with part (C).

```
# With goal of HDI all above 0.5:
sampSize = minNforHDIpower( genPriorMode=0.80, genPriorN=2,
                            HDImaxwid=NULL, nullVal=0.50, ROPE=c(0.0,0.52),
                            desiredPower=0.8,
                            audPriorMode=0.5, audPriorN=2,
                            HDImass=0.95, initSampSize=130, verbose=TRUE )
# Result:
# For sample size = 130, power = 0.3969466
# For sample size = 131, power = 0.3939394
# For sample size = 132, power = 0.3984962
# For sample size = 133, power = 0.3955224
# For sample size = 134, power = 0.4 # Notice this is exactly half of power before
# ...
# For sample size = 996, power = 0.449348
# For sample size = 997, power = 0.4488978
# For sample size = 998, power = 0.4494494
# For sample size = 999, power = 0.449
# For sample size = 1000, power = 0.4495504
# ... # Approaches asymptote of 0.48, never getting close to desired power. The
asymptote is 0.48 because the data generating distribution only has 48% above 0.52. If
you use a ROPE of c(0,0.50), the asymptotic power is 0.50 because the data generating
distribution has only 50% above 0.50.
```

Exercise 13.3

Exercise 13.3. [Purpose: Hands on experience with Monte Carlo power simulation in Section 13.2.5.] The script `Jags-Ydich-XnomSsubj-MbinomBetaOmegaKappa-Power.R` in Section 13.2.5 was used to estimate power for the therapeutic-touch experiment of Figure 9.9.

(A) Run the script using `nSimulatedDataSets` of 50. Show which line of code you changed to accomplish this. Report the final power estimates. How do your results compare with the results shown in Section 13.2.5? *Hint:* The power estimates should be about the same, but because you used a smaller number of simulated data sets, the bounds on your power estimate should be wider (less certain).

I changed only 1 line in the script:

```
# Specify the number of simulated experiments:
nSimulatedDataSets = 50 # min(500,NROW(mcmcMat)) # An arbitrary large number.
```

Here are the results:

```
==================== Simulation 50 of 50 ====================

Calling the simulation using the parallel method...
Following the progress of chain 1 (the program will wait for all chains to fi
nish before continuing):
Welcome to JAGS 3.4.0 on Sat Dec 06 10:48:39 2014
JAGS is free software and comes with ABSOLUTELY NO WARRANTY
Loading module: basemod: ok
Loading module: bugs: ok
. . Reading data file data.txt
. Compiling model graph
   Resolving undeclared variables
   Allocating nodes
   Graph Size: 56
. Reading parameter file inits1.txt
. Initializing model
. Adapting 500
-------------------------------------------------| 500
+++++++++++++++++++++++++++++++++++++++++++++++++ 100%
Adaptation successful
. Updating 500
-------------------------------------------------| 500
************************************************* 100%
. . . . Updating 3334
-------------------------------------------------| 3300
************************************************* 99%
* 100%
. . . Updating 0
. Deleting model
.
All chains have finished
Simulation complete.  Reading coda files...
Coda files loaded successfully
Finished running the simulation
       kappa    omega theta[1] theta[2] theta[3]
[1,]  8.9640 0.592768 0.555104 0.515620 0.492694
[2,] 59.4837 0.723772 0.771680 0.730234 0.713290
...
```

The last two lines above were output by the line **`show(HDImat[,1:5])`** in the function goalAchievedForSample. Each column shows the limits of the 95%HDI of the corresponding parameter for the current batch of simulated data.

The estimated power from the 50 simulations is as follows. Your results will differ because of randomness in data generation and MCMC chains.

```
[1] “omegaAboveROPE: Est.Power=1; Low Bound=0.943; High Bound=1"
[1] "omegaNarrowHDI: Est.Power=1; Low Bound=0.943; High Bound=1"
[1] "thetasAboveROPE: Est.Power=1; Low Bound=0.943; High Bound=1"
[1] "thetasNarrowHDI: Est.Power=0; Low Bound=0; High Bound=0.057"
```

The results above were for 14 simulated subjects and 47 trials per subject. They are similar to the results reported on p. 382 of the book, in the sense that the first three goals have high power and the last goal has low power. The quantitative details are different because of the random data generation and using only 50 simulations instead of 500.

(B) Now you will run the power simulation starting with idealized data that mimic the actual data. Refer to the posterior distribution from the analysis of the actual data in Figure 9.10, p. 243. Notice the central tendency and HDI on the group-level mode. We will use those characteristics for the idealized data generating hypothesis. Specifically, near the beginning of the script, set `idealGroupMean = 0.44`, `idealGroupSD = 0.04`, `idealNsubj = 28`, and `idealNtrlPerSubj = 10`. Explain what each of those settings does and explain why those values were chosen.

```
# Specify idealized hypothesis:
idealGroupMean = 0.44 # Matches the actual data group mean
idealGroupSD = 0.04   # Matches the actual data SD
idealNsubj = 28       # Matches the actual data number of subjects
idealNtrlPerSubj = 10 # Matches the actual data trials per subject
```

(C) Because the idealized data have central tendency near chance performance, we cannot have high hopes for rejecting the null, and therefore our goal might be high precision. In the function `goalAchievedForSample`, set the `HDImaxwid` to 0.1. Also, for high precision, we will need more data than was obtained in the original experiment, so try setting `NSubj` to 40 and `NtrlPerSubj` to 100. Because this is an exercise, not real research, change the number of simulated data sets to only 20. Report the lines of code you changed (and any you deleted or commented out). Now run the simulation and report the final estimated power for each of the goals. Why does the goal `omegaNarrowHDI` have high power but the goal `thetaNarrowHDI` have low power?

In goalAchievedForSample:

```
  # Specify criteria for goals:
  nullROPE = c(0.48,0.52)
  HDImaxWid = 0.1
```

Before the simulation loop:

```
# Specify sample size for each simulated data set:
Nsubj = 40 ; NtrlPerSubj = 100
#Nsubj = 2*7 ; NtrlPerSubj = 47  # 658 flips total
```

```
nSimulatedDataSets = 20 # min(500,NROW(mcmcMat)) # An arbitrary large number.
```

Results:

```
…
==================== Simulation 20 of 20 ====================

Calling the simulation using the parallel method...
Following the progress of chain 1 (the program will wait for all chains to finish before
continuing):
Welcome to JAGS 3.4.0 on Sat Dec 06 11:23:05 2014
JAGS is free software and comes with ABSOLUTELY NO WARRANTY
Loading module: basemod: ok
Loading module: bugs: ok
. . Reading data file data.txt
. Compiling model graph
   Resolving undeclared variables
   Allocating nodes
   Graph Size: 134
. Reading parameter file inits1.txt
. Initializing model
. Adapting 500
-------------------------------------------------| 500
+++++++++++++++++++++++++++++++++++++++++++++++++ 100%
Adaptation successful
. Updating 500
-------------------------------------------------| 500
************************************************* 100%
. . . . Updating 3334
-------------------------------------------------| 3300
************************************************* 99%
* 100%
. . . Updating 0
. Deleting model
.
All chains have finished
Simulation complete.  Reading coda files...
Coda files loaded successfully
Finished running the simulation
       kappa    omega theta[1] theta[2] theta[3]
[1,] 14.8656 0.376964 0.413457 0.412082 0.241574
[2,] 45.0655 0.444261 0.586941 0.581204 0.401381
> …

[1] "omegaAboveROPE: Est.Power=0; Low Bound=0; High Bound=0.133"
[1] "omegaNarrowHDI: Est.Power=1; Low Bound=0.867; High Bound=1"
[1] "thetasAboveROPE: Est.Power=0; Low Bound=0; High Bound=0.133"
[1] "thetasNarrowHDI: Est.Power=0; Low Bound=0; High Bound=0.133"
```

omegaNarrowHDI has high power because there are 40 subjects, which is enough to consistently get a group-level estimate of omega that is modestly certain (HDI width less than 0.1), but thetasNarrowHDI has low power because 100 trials per subject usually yields HDIs on individual theta's such that at least one of them has width greater than 0.1.

(D) For those who want a simple programming exercise in R, try this: Instead of using idealized data to create hypothetical data-generating parameter values, use the actual data from the original experiment. In the first part of the script, just comment out or delete the lines that create idealized data. Instead, use the actual data in the `genMCMC` function. Then repeat the previous part. Are the power estimates about the same?

Right after the idealized data are generated, simply read in the actual data, which overwrites the idealized data:

```
idealDatFrm = data.frame(dataMat)
idealDatFrm = read.csv("TherapeuticTouchData.csv") # overwrites previous idealDatFrm
# Run Bayesian analysis on idealized data:
mcmcCoda = genMCMC( data=idealDatFrm , saveName=NULL ,
                    numSavedSteps=2000 , thinSteps=20 )
```

The results from running the whole script:

```
[1] "omegaAboveROPE: Est.Power=0; Low Bound=0; High Bound=0.133"
[1] "omegaNarrowHDI: Est.Power=0.7; Low Bound=0.491; High Bound=0.864"
[1] "thetasAboveROPE: Est.Power=0; Low Bound=0; High Bound=0.133"
[1] "thetasNarrowHDI: Est.Power=0; Low Bound=0; High Bound=0.133"
```

The resulting power estimates are similar to the previous part, in the sense that there is high-ish power for omegaNarrowHDI and low power for thetasNarrowHDI. Remember that there is a lot of uncertainty in these estimates because they are based on only 20 simulated data sets.

Exercise 13.4

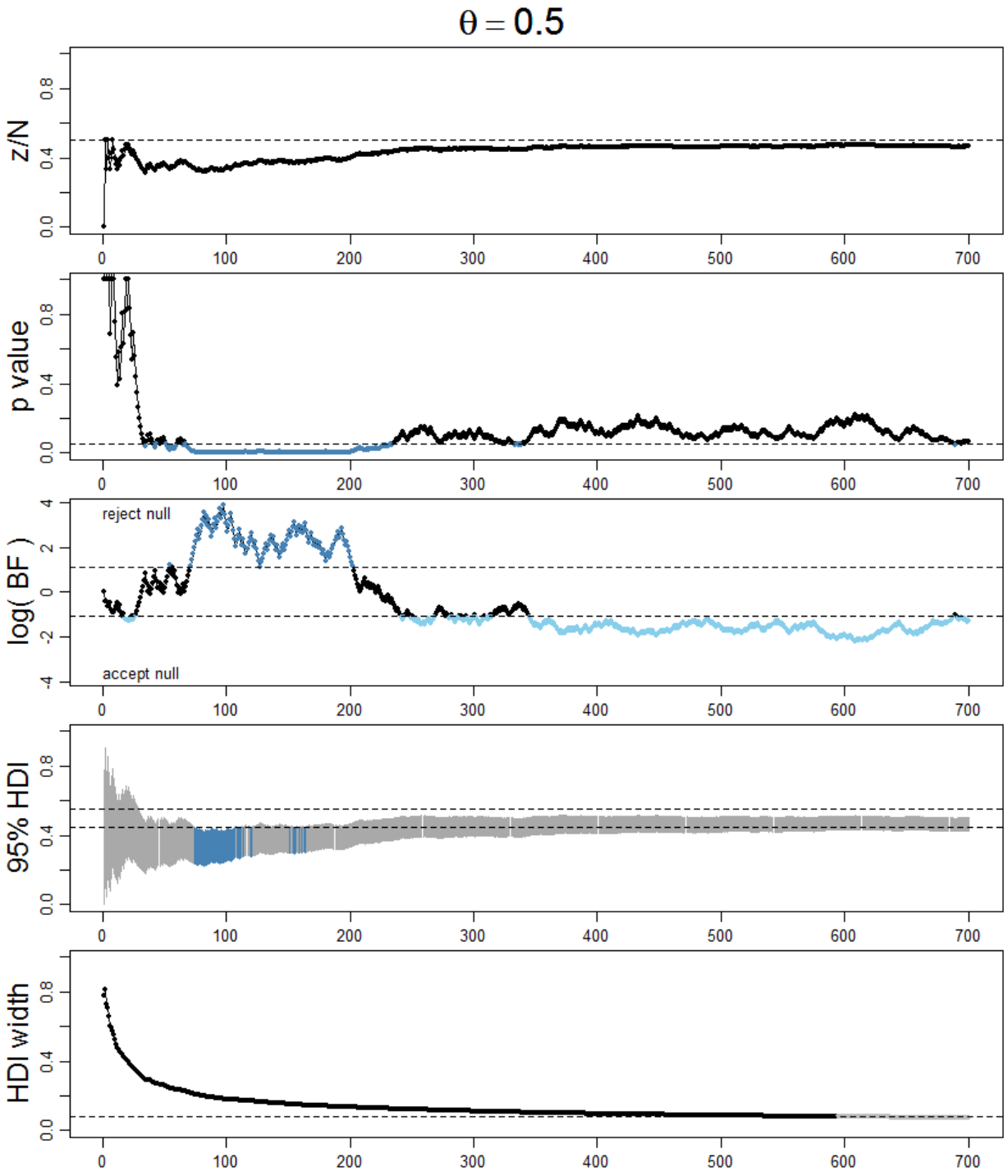
Exercise 13.4. [Purpose: To explore sequential testing of coin flips.] For this exercise, your job is to create your own versions of the graphs in Figure 13.4. Ready? Go! *Hints:* (continued on next page)

- For a reminder of how to create and plot z/N, see Figure 4.1, p. 75, and its accompanying description in the text.
- To compute a p value for a proportion, assuming the intention was to stop at N, you could use R's function `binom.test`. For example: `z=9 ; N=10 ; theta=0.5 ; binom.test( x=z , n=N , p=theta , alternative="two.sided" )$p.value` returns the p value. Or you could compute it "from scratch" by using the definitions of the binomial distribution.
- The BF can be computed from Equation 12.3, p. 344. Be careful to use the beta *function* and not the beta *distribution*.
- The HDI can be computed with the function `HDIofICDF` that has been used in some of the power scripts and functions. It is defined in the utilities functions that come with this book, and therefore that file must be sourced before the function can be used. For this application, its basic format is `HDIofICDF( qbeta , shape1=1+z , shape2=1+N-z )`.

The R code and the resulting graph are on the next pages:

```
graphics.off()
rm(list=ls(all=TRUE))
source("DBDA2E-utilities.R")

# Specify some properties of the flipped coin:
rs=21         # random number seed
theta=0.5     # true theta of coin
maxN=700      # maximum number of flips
nullTheta=0.5 # null hypothesis theta

# Specify some colors for graphs:
noDecCol = "darkgrey"
rejectCol = "steelblue"
acceptCol = "skyblue"

set.seed(rs) # set random seed

# Generate random sequence of flips:
y = sample( c(0,1) , size=maxN , replace=TRUE , prob=c(1-theta,theta) )

# Compute proportion of heads at each step in sequence:
z = cumsum(y)
N = 1:maxN
prop = z / N

# Compute p value at each step, assuming fixed N at that step:
pval = rep(NA,maxN)
for ( i in 1:maxN ) {
  plow = pbinom( q=z[i] , size=N[i] , prob=nullTheta )
  phi = 1.0-pbinom( q=z[i]-1 , size=N[i] , prob=nullTheta )
  pval[i] = 2*min(c(plow,phi))
}

# Compute Bayes factor at each step:
aPrior=1
bPrior=1
bf = ( exp( lbeta(aPrior+z,bPrior+N-z) - lbeta(aPrior,bPrior) )
       / ( nullTheta^z * (1.0-nullTheta)^(N-z) ) )

# Compute HDI at each step:
hdi = matrix(NA,nrow=2,ncol=maxN)
for ( i in 1:maxN ) {
  hdi[,i] = HDIofICDF( qbeta , shape1=aPrior+z[i] , shape2=bPrior+N[i]-z[i] )
}
# HDI width:
hdiwd = hdi[2,] - hdi[1,]

# Open graphics window, with room at top for overall title:
openGraph(width=7,height=8)
layout(matrix(1:5,nrow=5))
par( mar=c(1.5,3.5,0.5,1) , mgp=c(2.0,0.7,0) ,  oma=0.1+c(0,0,2,0) ,
     cex.lab=1.75 , cex.main=1.5 ,  pch=20 )
```

```
# Plot proportion of heads:
plot( N , prop , xlab="" , ylab="z/N" , ylim=c(0,1) , type="o" )
abline(h=theta,lty="dashed")

# Plot p values:
plot( N , pval , xlab="" , ylab="p value" , ylim=c(0,1) , type="o" )
points( N[pval<.05] , pval[pval<.05] , col=rejectCol )
abline(h=0.05,lty="dashed")

# Plot Bayes factor on log scale:
ymax = max(abs(log(bf)))
plot( N , log(bf) , xlab="" , ylab="log( BF )" , type="o" ,
      ylim=c(-ymax,ymax) )
points( N[bf>3] , log(bf[bf>3]) , col=rejectCol )
points( N[bf<1/3] , log(bf[bf<1/3]) , col=acceptCol )
abline(h=log(3),lty="dashed")
abline(h=log(1/3),lty="dashed")
text( 0,-ymax,"accept null",adj=c(-0.0,-0.0))
text( 0,ymax,"reject null",adj=c(-0.0,1.0))

# Plot HDI as segments:
plot( N , hdi[1,] , xlab="" , ylab="95% HDI" , ylim=c(0,1) , type="n" )
segments( x0=N , y0=hdi[1,] , x1=N , y1=hdi[2,] , col=noDecCol )
exidx = N[ hdi[1,] > 0.55 | hdi[2,] < 0.45 ]
inidx = N[ hdi[1,] > 0.45 & hdi[2,] < 0.55 ]
segments( x0=N[exidx] , y0=hdi[1,exidx] , x1=N[exidx] , y1=hdi[2,exidx] ,
col=rejectCol )
segments( x0=N[inidx] , y0=hdi[1,inidx] , x1=N[inidx] , y1=hdi[2,inidx] ,
col=acceptCol )
abline(h=0.55,lty="dashed")
abline(h=0.45,lty="dashed")

# Plot HDI widths:
plot( N , hdiwd , xlab="" , ylab="HDI width" , ylim=c(0,1) , type="o" )
points( N[hdiwd<.08] , hdiwd[hdiwd<.08] , col="grey" )
abline(h=0.08,lty="dashed")

# Title at top of all panels:
mtext( text=bquote(list(theta==.(theta))) , outer=TRUE ,
       adj=c(0.5,0.5) , cex=1.5 )
```


θ = 0.5
z/N
p value
log(BF)
reject null
accept null
95% HDI
HDI width
0.0
0.4
0.8
-4
-2
0
2
4
0
100
200
300
400
500
600
700

Chapter 14

Exercise 14.1

Exercise 14.1. [Purpose: Transformed parameters in Stan, and comparison with JAGS.] For this exercise, we analyze in Stan the therapeutic-touch data from Section 9.2.4, p. 240. The model structure is depicted in Figure 9.7, p. 236. The relevant Stan scripts are included in the online suite of programs that accompany the book; see `Stan-Ydich-XnomSsubj-MbernBetaOmegaKappa-Example.R`. Compare it with the JAGS version, which is the same file name but starting with Jags-.

(A) Consider the model block of the Stan code called by the script. What does the `transformed parameters` block do?

Here is an excerpt from the Stan model definition:

```
transformed parameters {
  real<lower=0> kappa ;
  kappa <- kappaMinusTwo + 2 ;
}
model {
  omega ~ beta( 1 , 1 ) ;
  kappaMinusTwo ~ gamma( 0.01 , 0.01 ) ; // mean=1 , sd=10 (generic vague)
  theta ~ beta( omega*(kappa-2)+1 , (1-omega)*(kappa-2)+1 ) ; // vectorized
  for ( i in 1:Ntotal ) {
    y[i] ~ bernoulli( theta[s[i]] ) ;
  }
}
```

Notice that theta is defined in terms of kappa, while there is a prior on kappaMinusTwo. The `transformed parameters` block transforms parameter kappaMinusTwo into parameter kappa. In JAGS, this sort of transformation would be done in the model block, but in Stan it is done in a separate block.

(B) Consider the model block of the Stan code called by the script. Why is `theta` vectorized but not y?

Notice in the model block above that theta is treated as a vector:

```
theta ~ beta( omega*(kappa-2)+1 , (1-omega)*(kappa-2)+1 ) ; // vectorized
```

instead of putting theta[j] in a for (j in 1:Nsubj) {theta[j] ~ … } loop. Stan (and most languages) runs faster when operations are vectorized. It would be nice to vectorize the line that specifies y[i] ~ bernoulli(theta[s[i]]), but unfortunately it cannot be because of the need for nested indices in theta[s[i]]. In other words, if we drop the i index, theta[s] does not work.

(C) Run the Stan program, and note how long it takes. Run the Jags version and note how long it takes. Do they produce the same posterior distribution with the same effective sample size (ESS)? Which is faster? You might find that Stan is no faster, or even much slower. Thus, there is no guarantee that Stan will work better. Please note that to make comparisons between Stan and Jags, the ESS of the resulting chains must be matched, not the number of steps. The summaries and graphs created by the `coda`-based functions are convenient for displaying the ESS computed with the same ESS algorithm for the outputs of both Stan and JAGS. For a fair comparison of Stan and JAGS, they also must be matched for their required burn-in phases, as they might need different amounts of burn-in to reach converged chains.

To time the runs, we can use code like the following:

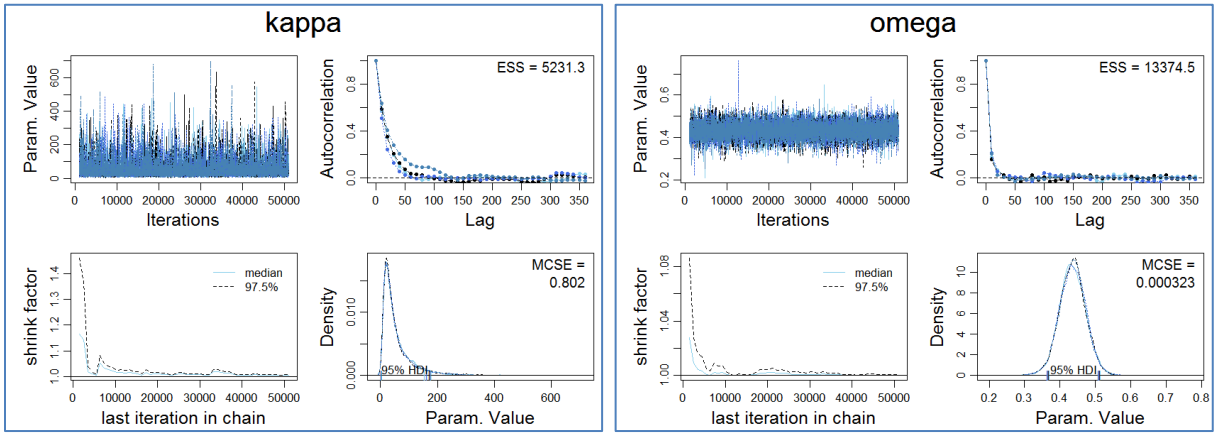
```
startTime = proc.time()
mcmcCoda = genMCMC( data=myData , sName="s" , yName="y" ,
                    numSavedSteps=20000 , saveName=fileNameRoot , thinSteps=10 )
stopTime = proc.time()
duration = stopTime - startTime
show(duration)
```

I used 20000 steps (as shown above) for JAGS and 12000 steps for Stan.

For JAGS (*not* parallelized with runjags)**:**

```
> show(duration)
   user  system elapsed
  59.56    0.06   59.78
```

In other words, the elapsed time was essentially 1 minute.

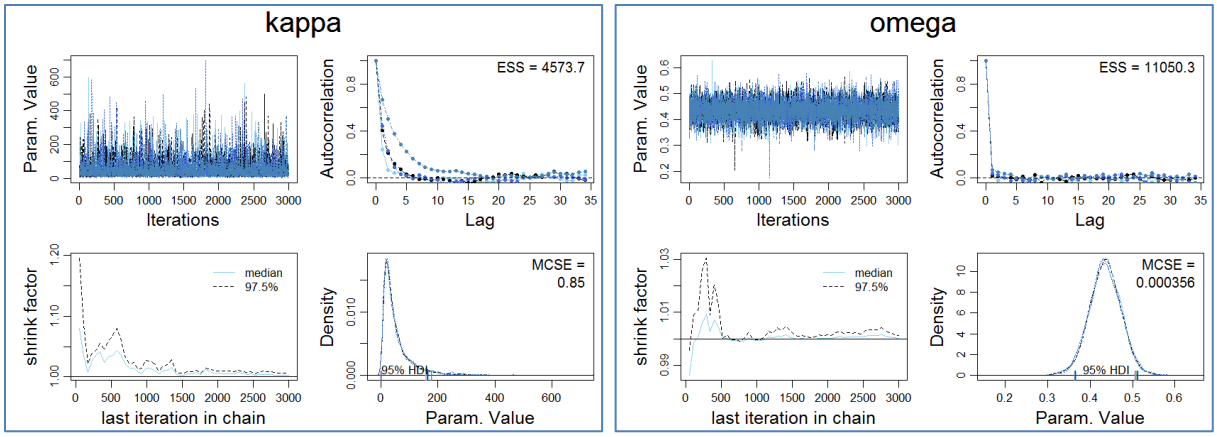


The ESS of kappa was about 5,231; of omega was about 13,375; and of theta[j] (not shown above) about 13,000.

For Stan:

```
> show(duration)
   user  system elapsed
 211.35    0.31  282.39
```

In other words, the elapsed time was about 4.7 minutes (4.7 times longer than JAGS).



The ESS of kappa was about 4,574; of omega was about 11,050; and of theta's (not shown) about 10,100. It is suprizing (to me) that the ESS of kappa is not closer to the ESS of the other parameters, because Stan tends to be better at sampling "difficult" parameters than JAGS.

Exercise 14.2

Exercise 14.2. [Purpose: Power analysis in Stan.] It was briefly mentioned in Section 14.3.1 that the DSO created by the Stan command `stan_model` could be reused in multiple calls of the Stan `sampling` command. For this exercise, your goal is to implement in Stan the Monte Carlo approximation of power described in Section 13.2.4, p. 372. It is straight forward to implement it in Stan using the exactly analogous function definitions, but this approach, while logically correct, is inefficient because it will recompile the Stan model every time a new simulated data set is analyzed. The real exercise of this exercise is to modify the function that does the Stan analysis so that it can either compile the Stan model afresh or reuse an existing DSO. In your modified version, *compile the Stan model outside the loop that runs the analyses on simulated data, and pass the compiled DSO as an argument into the function that runs an analysis on a simulated data set.* Show your code, with explanatory comments.

The following R script started with the script **`Jags-Ydich-Xnom1subj-MbernBeta-Power.R`** and made minimal modifications to convert it to Stan, using the corresponding Stan model in **`Stan-Ydich-Xnom1subj-MbernBeta.R`**. Here is the complete Stan script, with a few lines highlighted to call attention to the main variations needed for Stan:

```
# Stan-Ydich-Xnom1subj-MbernBeta-Power.R
graphics.off() # This closes all of R's graphics windows.
rm(list=ls())  # Careful! This clears all of R's memory!
source("DBDA2E-utilities.R")
require(rstan)
fileNameRoot = "Stan-Ydich-Xnom1subj-MbernBeta-Power-" # for future use

# Create the Stan DSO:
modelString = "
  data {
    int<lower=0> Ntotal ;
    int<lower=0,upper=1> y[Ntotal] ;
  }
  parameters {
    real<lower=0,upper=1> theta ;
  }
```

```
  model {
    theta ~ beta(1,1) ;
    y ~ bernoulli(theta) ; // implicitly vectorized
  }
" # close quote for modelString
# Translate to C++ and compile to DSO:
stanDso <- stan_model( model_code=modelString )

# Define specialized genMCMC that uses pre-made Stan DSO
genMCMC = function( stanDso=stanDso , data , numSavedSteps=50000 ,
                    saveName=NULL ) {
  #-----------------------------------------------------------------------------
  # THE DATA.
  if ( class(data)=="data.frame" ) {  # If data is a data.frame
    y = myData$y                      # then pull out the column named y
  } else {                            # else
    y = data                          # rename the data as y.
  }
  # Do some checking that data make sense:
  if ( any( y!=0 & y!=1 ) ) { stop("All y values must be 0 or 1.") }
  Ntotal = length(y)
  # Specify the data in a list, for later shipment to JAGS:
  dataList = list(
    y = y ,
    Ntotal = Ntotal
  )
  #-----------------------------------------------------------------------------
  # INTIALIZE THE CHAINS.
  initsList = function() {
    resampledY = sample( y , replace=TRUE )
    thetaInit = sum(resampledY)/length(resampledY)
    thetaInit = 0.001+0.998*thetaInit # keep away from 0,1
    return( list( theta=thetaInit ) )
  }
  #-----------------------------------------------------------------------------
  # RUN THE CHAINS
  parameters = c( "theta")     # The parameters to be monitored
  burnInSteps = 500            # Stan defaults to iter/2 for overdispersed inits
  nChains = 4                  # nChains should be 2 or more for diagnostics
  thinSteps = 4                # In Stan there is autocorrelation, so thin
  stanFit <- sampling( object=stanDso ,
                       data = dataList ,
                       pars = parameters , # optional
                       chains = nChains ,
                       iter = ( ceiling(numSavedSteps/nChains)*thinSteps
                                +burnInSteps ) ,
                       warmup = burnInSteps ,
                       thin = thinSteps ,
                       init = initsList ) # optional
  # For consistency with JAGS-oriented functions in DBDA2E collection,
  # convert stan format to coda format:
  codaSamples = mcmc.list(lapply(1:ncol(stanFit) ,
                                 function(x){mcmc(as.array(stanFit)[,x,])}) )
  # resulting codaSamples object has these indices:
  #   codaSamples[[ chainIdx ]][ stepIdx , paramIdx ]
  if ( !is.null(saveName) ) {
    save( codaSamples , file=paste(saveName,"Mcmc.Rdata",sep="") )
    save( stanFit , file=paste(saveName,"StanFit.Rdata",sep="") )
    save( stanDso , file=paste(saveName,"StanDso.Rdata",sep="") )
  }
  return( codaSamples )
} # end function
```

```
# Define function that assesses goal achievement for a single set of data:
goalAchievedForSample = function( data ) {
  # Generate the MCMC chain:
  mcmcCoda = genMCMC( stanDso=stanDso , data=data , numSavedSteps=10000 ,
                      saveName=NULL )
  # Check goal achievement. First, compute the HDI:
  thetaHDI = HDIofMCMC( as.matrix(mcmcCoda[,"theta"]) )
  # Define list for recording results:
  goalAchieved = list()
  # Goal: Exclude ROPE around null value:
  thetaROPE = c(0.48,0.52)
  goalAchieved = c( goalAchieved ,
                    "ExcludeROPE"=( thetaHDI[1] > thetaROPE[2]
                                  | thetaHDI[2] < thetaROPE[1] ) )
  # Goal: HDI less than max width:
  thetaHDImaxWid = 0.2
  goalAchieved = c( goalAchieved ,
                    "NarrowHDI"=( thetaHDI[2]-thetaHDI[1] < thetaHDImaxWid ) )
  # More goals can be inserted here if wanted...
  # Return list of goal results:
  return(goalAchieved)
}

# Specify mode and concentration of hypothetical parameter distribution:
omega = 0.70
kappa = 2000
# Specify sample size for each simulated data set:
sampleN = 74
# Run a bunch of simulated experiments:
nSimulatedDataSets = 100 # An arbitrary large number.
for ( simIdx in 1:nSimulatedDataSets ) {
  # Generate random value from hypothesized parameter distribution:
  genTheta = rbeta( 1 , omega*(kappa-2)+1 , (1-omega)*(kappa-2)+1 )
  # Generate random data based on parameter value:
  sampleZ = rbinom( 1 , size=sampleN , prob=genTheta )
  # Convert to vector of 0's and 1's for delivery to Stan function:
  simulatedData = c(rep(1,sampleZ),rep(0,sampleN-sampleZ))
  # Do Bayesian analysis on simulated data:
  goalAchieved = goalAchievedForSample( simulatedData )
  # Tally the results:
  if (!exists("goalTally")) { # if goalTally does not exist, create it
    goalTally=matrix( nrow=0 , ncol=length(goalAchieved) )
  }
  goalTally = rbind( goalTally , goalAchieved )
  # save( goalTally ,
  #       file="Stan-Ydich-Xnom1subj-MbernBeta-Power-goalTally.Rdata" )
}

# For each goal...
for ( goalIdx in 1:NCOL(goalTally) ) {
  # Extract the goal name for subsequent display:
  goalName = colnames(goalTally)[goalIdx]
  # Compute number of successes:
  goalHits = sum(unlist(goalTally[,goalIdx]))
  # Compute number of attempts:
  goalAttempts = NROW(goalTally)
  # Compute proportion of successes:
  goalEst = goalHits/goalAttempts
  # Compute HDI around proportion:
  goalEstHDI = HDIofICDF( qbeta ,
                          shape1=1+goalHits ,
                          shape2=1+goalAttempts-goalHits )
  # Display the result:
```

```
  show( paste0( goalName,
                ": Est.Power=" , round(goalEst,3) ,
                "; Low Bound=" , round(goalEstHDI[1],3) ,
                "; High Bound=" , round(goalEstHDI[2],3) ) )
}
```

When the script is run, it spews a bunch of info to the console for each simulated data set. At the end, it displays the summary information:

```
[1] "ExcludeROPE: Est.Power=0.91; Low Bound=0.844; High Bound=0.956"
[1] "NarrowHDI: Est.Power=0.38; Low Bound=0.289; High Bound=0.477"
```

This (above) is based on 100 simulated data sets.

As a check, we can compare with the output of the corresponding JAGS script, which produces this:

```
[1] "ExcludeROPE: Est.Power=0.88; Low Bound=0.807; High Bound=0.934"
[1] "NarrowHDI: Est.Power=0.38; Low Bound=0.289; High Bound=0.477"
```

Chapters 15

Table 15.3 is referred to in the exercises of this chapter, and therefore the Table is reproduced here for convenience:

Table 15.3 Book chapters that discuss combinations of scale types for predicted and predictor variables of Tables 15.1 and 15.2

	Scale type of predictor *x*					
			Metric		Nominal	
Scale type of predicted *y*	Single group	Two groups	Single predictor	Multiple predictors	Single factor	Multiple factors
Metric	Chapter 16		Chapter 17	Chapter 18	Chapter 19	Chapter 20
Dichotomous	Chapters 6–9			Chapter 21		
Nominal				Chapter 22		
Ordinal				Chapter 23		
Count				Chapter 24		

Exercise 15.1

Exercise 15.1. [Purpose: For real-world examples of research, identify which statistical model is relevant.] For each of the examples below, identify the predicted variable and its scale type, identify the predictor variable(s) and its scale type, and identify the cell of Table 15.3 to which the example belongs.

(A) Guber (1999) examined average performance by public high school students on the SAT as a function of how much money was spent per pupil by the state, and what percentage of eligible students actually took the exam.

Predicted variable: SAT score – metric.
Predictor variables: Money per pupil – metric; percentage of students taking exam – metric.
Cell of Table: Metric y scale, multiple metric x scale (**Chapter 18**).

(B) Hahn, Chater, and Richardson (2003) were interested in perceived similarity of simple geometric patterns. Human observers rated pairs of patterns for how similar the patterns appeared, by circling one of the digits 1–7 printed on the page, where 1 meant "very dissimilar" and 7 meant "very similar." The authors presented a theory of perceived similarity, in which patterns are perceived to be dissimilar to the extent that it takes more geometric transformations to produce one pattern from the other. The theory specified the exact number of transformations needed to get from one pattern to the other.

Predicted variable: Similarity rating – ordinal.
Predictor variables: Number of transformations – metric (not just ordinal because each transformation counts the same; and could be count scale but it acts the same as metric in this application).
Cell of Table: ordinal y scale, metric x scale (**Chapter 23**).

(C) R. L. Berger, Boos, and Guess (1988) were interested in the longevity of rats, measured in days, as a function of the rat's diet. One group of rats fed freely, another group of rats had a very low calorie diet.

Predicted variable: Longevity – metric.
Predictor variables: Diet type – nominal.
Cell of Table: metric y scale, nominal (two-group) x scale (**Chapter 16**).

(D) McIntyre (1994) was interested in predicting the tar content of a cigarette (measured in milligrams) from the weight of the cigarette.

Predicted variable: tar content – metric.
Predictor variables: weight – metric.
Cell of Table: metric y scale, (single) metric x scale (**Chapter 17**).

(E) You are interested in predicting the gender of a person, based on the person's height and weight.

Predicted variable: gender – dichotomous [or nominal for multiple gender categories].
Predictor variables: height – metric; weight – metric.
Cell of Table: dichotomous [or nominal] y scale, multiple metric x scales (**Chapter 21 [or 22]**).

(F) You are interested in predicting whether a respondent will agree or disagree with the statement, "The United States needs a federal health care plan with a public option," on the basis of the respondent's political party affiliation.

Predicted variable: agreement – dichotomous.
Predictor variables: party affiliation – nominal.
Cell of Table: dichotomous y scale, nominal x scale (**Chapter 22**).

Exercise 15.2

Exercise 15.2. [Purpose: Find *student-relevant* real-world examples of each type of situation in Table 15.3.] For each cell of Table 15.3, provide an example of research involving that cell's model structure. Do this by finding published articles that describe research with the corresponding structure. The articles do *not* need to have Bayesian data analysis; the articles *do* need to report research that involves the corresponding types of predictor and predicted variables. Because it might be overly time consuming to find published examples of all the cells, please find published articles of at least six cells spanning at least three different rows. For each example, specify the following:

- The full citation to the published article (see the references of this book for examples of how to cite articles),
- The predictor and predicted variables. Describe their meaning and their type of scale. Briefly describe the meaningful context for the variables, that is, the goal of the research.

An example of an answer:

Liddell, T. M. and Kruschke, J. K. (2014). Ostracism and fines in a public goods game with accidental contributions: The importance of punishment type. *Judgment and Decision Making*, **9**(6), 523-547.

Experiment 2 of that article had people choose what kind of punishment to apply to free riders in a public goods game. The punishment choice could be ostracism from the game, a "monetary" fine, or no punishment. The authors were interested in assessing how punishment choice depended on the extent to which the targeted player intentionally contributed, actually contributed (i.e., accidentally free loaded), and gained more than the punisher (measured as "indignation" of the punisher). Thus, the predicted variable was nominal punishment choice,

and the predictor variables were three metric values (actual contribution, intended contribution, and indignation). This corresponds to the cell of Table 15.3 for **Chapter 22**.

Chapters 16

Exercise 16.1

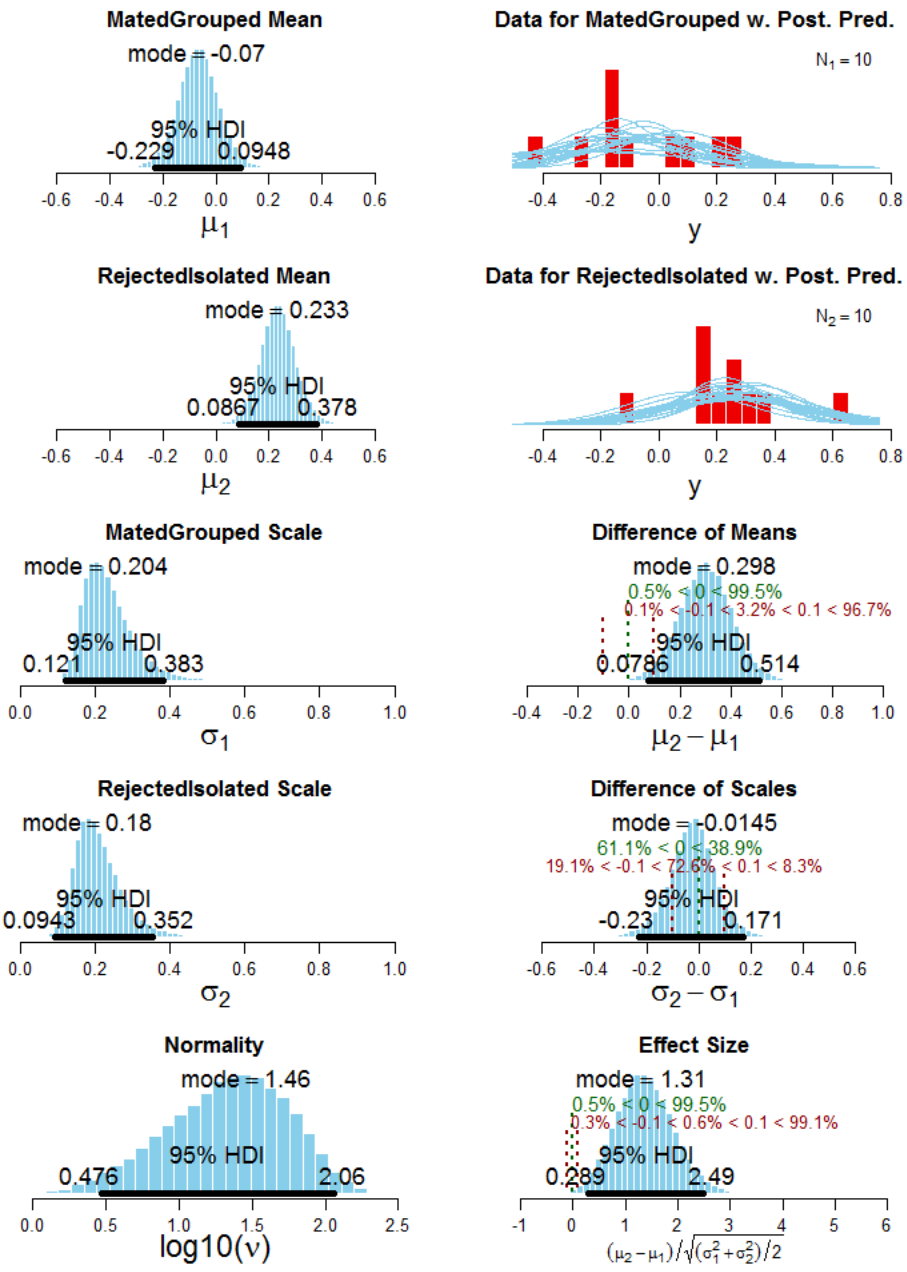
Exercise 16.1. [Purpose: Practice using different data files in the high-level script, with an interesting real example about alcohol preference of sexually frustrated males.] Shohat-Ophir et al. (2012) were interested in alcohol preferences of sexually deprived males. The procedure is illustrated in Figure 16.13, and was described as follows: "One cohort, rejected-isolated, was subjected to courtship conditioning; they experienced 1-h sessions of sexual rejection by mated females, three times a day, for 4 days. …Flies in the mated-grouped cohort experienced 6-h sessions of mating with multiple receptive virgin females (ratio 1:5) for 4 days. Flies from each cohort were then tested in a two-choice preference assay, in which they voluntarily choose to consume food with or without 15% ethanol supplementation. (Shohat-Ophir et al., 2012, p. 1351, citations and figure reference removed)" For each fly, the amount of each type of food consumed was converted to a *preference ratio*: the amount of ethanol-supplemented food minus the amount of regular food divided by the total of both. I constructed 3-day summary preference scores for each individual fruit fly by summing the consumption of ethanol and non-ethanol across days 6–8. The amounts of food consumed and the preference ratios are in the data file named ShohatOphirKAMH2012dataReduced.csv. My thanks to Dr. Galit Shohat-Ophir for providing the data.

(A) Run `Jags-Ymet-Xnom2grp-MrobustHet-Example.R` on the preference scores. Make sure that the ROPE on the means and standard deviation is scaled appropriately to the data. How big are differences between groups relative to the uncertainty of the estimate? What do you conclude? (If this result interests you, then you will also be intrigued by the results in Section 19.3.2, p. 561.)

Here are the relevant lines for reading in the data file and setting the ROPE:

```
myDataFrame = read.csv( file="ShohatOphirKAMH2012dataReduced.csv" )
xName="Group"
yName="PreferenceIndex"
fileNameRoot="ShohatOphirKAMH2012data-PI-"
RopeMuDiff=c(-0.1,0.1) ; RopeSdDiff=c(-0.1,0.1) ; RopeEff=c(-0.1,0.1)
```

The resulting posterior is summarized in the figure below. The difference of means has a modal value near 0.3. The 95% HDI clearly excludes zero, but not the arbitrary ROPE of -0.1 to +0.1. On the other hand, the effect size has a mode of about 1.3 (which is big) and a 95% HDI that clearly excludes a ROPE from -0.1 to 0.1, which is a typical limit (in psychological research) for a "small" effect size. In other words, the difference between groups is fairly large compared to the uncertainty of the estimate. (The difference between scales, on the other hand, is nearly zero.) In other words, the rejected males preferred the alcohol more than the mated males.

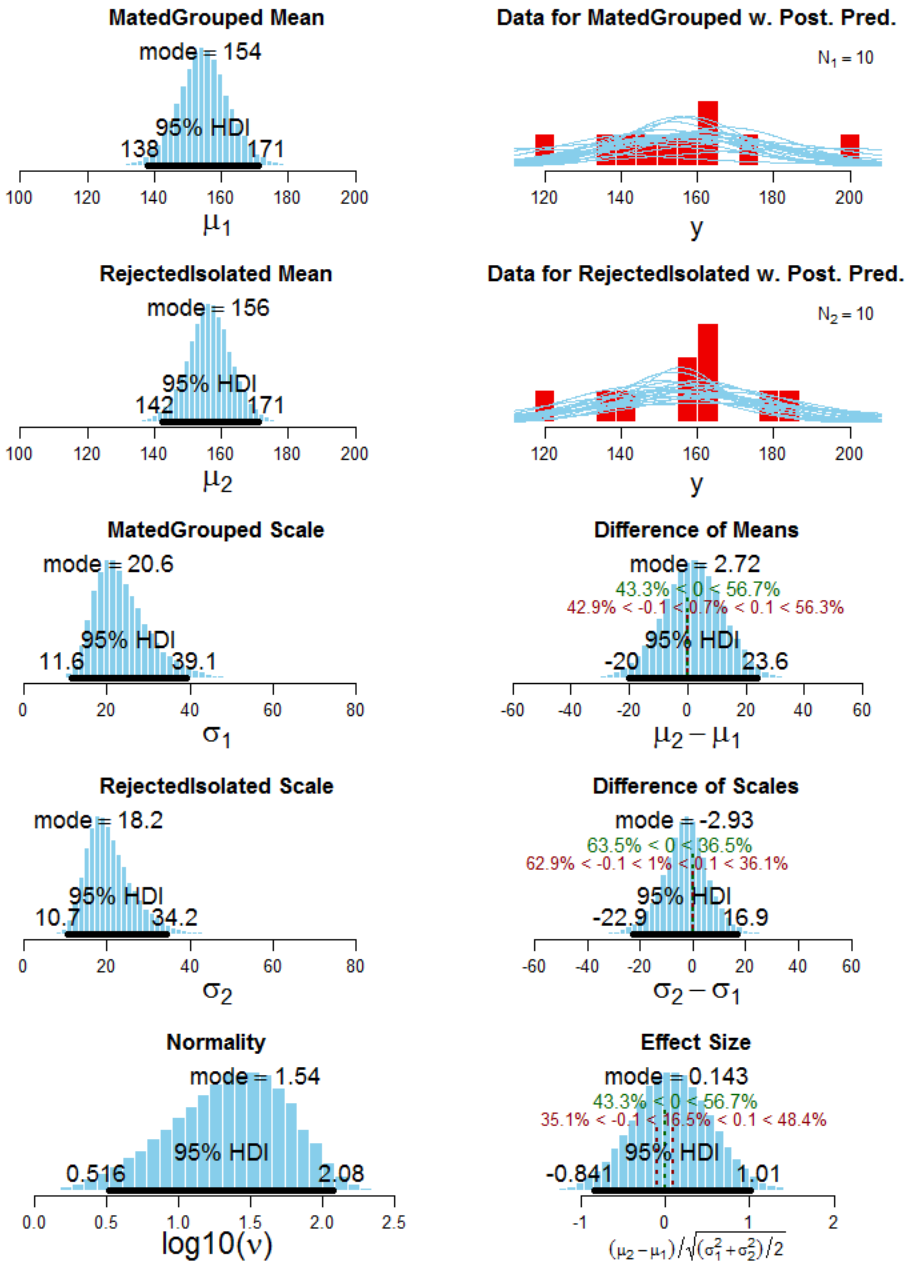


(B) Instead of focusing on the *relative* amounts of ethanol and regular food consumed, we might also be interested in the absolute total amount of food consumed. Run the analysis on the total consumption data, which has column name GrandTotal in the data file. What do you conclude? In particular, would you want to make an argument to accept the null hypothesis of no difference? (Review Section 12.1.1, beginning on p. 336.)

The R commands to read in the data are as follows. Notice that yName is now GrandTotal.

```
myDataFrame = read.csv( file="ShohatOphirKAMH2012dataReduced.csv" )
xName="Group"
yName="GrandTotal"
fileNameRoot="ShohatOphirKAMH2012data-GT-"
RopeMuDiff=c(-0.1,0.1) ; RopeSdDiff=c(-0.1,0.1) ; RopeEff=c(-0.1,0.1)
```

The result is shown below:



The effect size has a modal value of about 0.14, and a 95% HDI that extends from -0.84 to +1.0. Clearly there is no notable difference between the groups in terms of total consumption, and we would **not reject** a difference of zero. But also we would **not accept** a difference of zero because the estimate is very uncertain. These results do, however, demonstrate that we cannot attribute the difference in relative alcohol consumption (shown in the previous part) to a difference in overall consumption.

Exercise 16.2

Exercise 16.2. [Purpose: More practice using different data files in the high-level script, using a real example, with skewed data.] The typical lifespan of a laboratory rat that eats *ad lib* is approximately 700 days. When rats are placed on a restricted diet, their longevity can increase, but there is a lot of variability in lifespans across different individual rats. Restricting the diet might not only affect the typical lifespan, but restricting the diet might also affect the variance of the lifespan across rats. We consider data from R. L. Berger et al. (1988), as reported in Hand, Daly, Lunn, McConway, and Ostrowski (1994, data set #242), and which are available in the file named `RatLives.csv`.

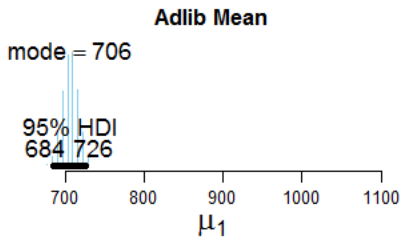
(A) Run the two-group analysis on the rat longevity data. Use JAGS or Stan as you prefer (report which one you used). Report the code you used to read in the data file, specify the column names for the data, and the ROPEs appropriate to the scale of the data. Do the groups appear to differ in their central tendencies and variances? Does the value of the normality parameter suggest that the data have outliers relative to a normal distribution?

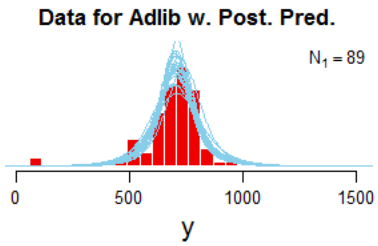
Using Jags-Ymet-Xnom2grp-MrobustHet-Example.R, the initial code for reading in the data and setting the ROPEs is

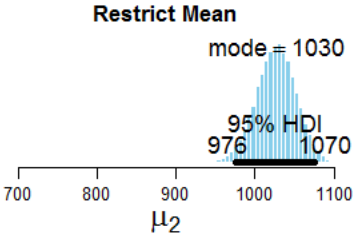
```
myDataFrame = read.csv( file="RatLives.csv" )
xName="Group"
yName="DaysLive"
fileNameRoot = "RatLives-"
RopeMuDiff=c(-10,10) ; RopeSdDiff=c(-10,10) ; RopeEff=c(-0.1,0.1)
```

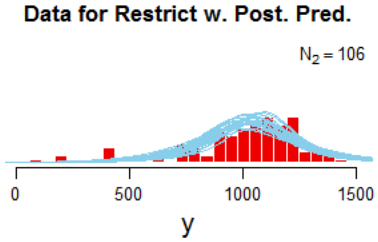
The ROPEs on mu and sigma were set to c(-10,+10) because a 10 day difference in longevity seems intuitively like a noticeable difference, for typical lifespan of 700 days. The ROPE on effect size was set to c(-0.1,+0.1) because 0.1 is conventionally a "small" effect size.

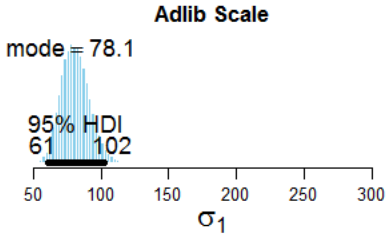
The result is shown on the next page. There is a very strong difference between groups both in means (mu) and in scales (sigma). The restricted diet group lives about 322 days longer on average, but also has a scale (sigma) about 117 days bigger. The normality parameter is very small, indicating that the data have large, non-normal tails. Visual inspection of the data suggests that the left tail is heavy.

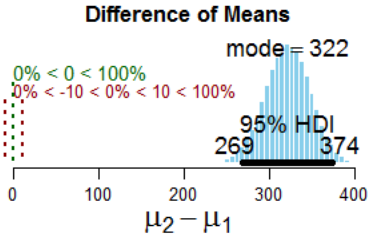

Adlib Mean
mode = 706
95% HDI
684 726
700 800 900 1000 1100
μ_1

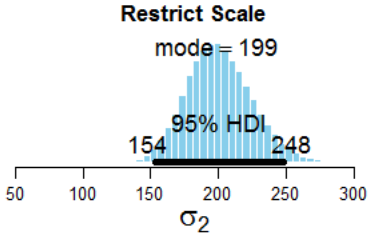

Data for Adlib w. Post. Pred.
$N_1 = 89$
0 500 1000 1500
y

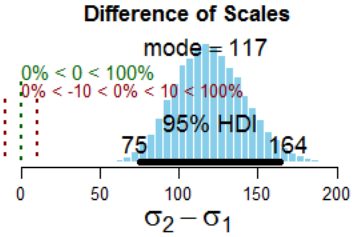

Restrict Mean
mode = 1030
95% HDI
976 1070
700 800 900 1000 1100
μ_2

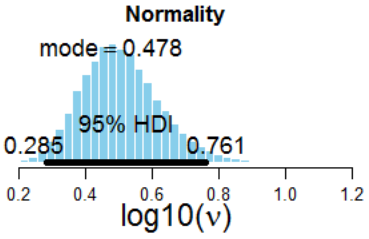

Data for Restrict w. Post. Pred.
$N_2 = 106$
0 500 1000 1500
y

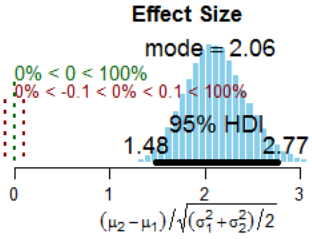

Adlib Scale
mode = 78.1
95% HDI
61 102
50 100 150 200 250 300
σ_1


Difference of Means
mode = 322
0% < 0 < 100%
0% < -10 < 0% < 10 < 100%
95% HDI
269 374
0 100 200 300 400
$\mu_2 - \mu_1$


Restrict Scale
mode = 199
95% HDI
154 248
50 100 150 200 250 300
σ_2


Difference of Scales
mode = 117
0% < 0 < 100%
0% < -10 < 0% < 10 < 100%
95% HDI
75 164
0 50 100 150 200
$\sigma_2 - \sigma_1$


Normality
mode = 0.478
95% HDI
0.285 0.761
0.2 0.4 0.6 0.8 1.0 1.2
log10(ν)


Effect Size
mode = 2.06
0% < 0 < 100%
0% < -0.1 < 0% < 0.1 < 100%
95% HDI
1.48 2.77
0 1 2 3
$(\mu_2 - \mu_1)/\sqrt{(\sigma_1^2 + \sigma_2^2)/2}$

(B) The data within each group appear to be skewed to the left. That is, within each group, there are many rats that died relatively young, but there are fewer outliers on the high end. We could try to implement a skewed noise distribution, or we could try to transform the data so they are approximately symmetric within each group. We will try the latter approach here. To get rid of leftward skew, we need a transformation that expands the rightward values. We will try squaring the data. Read in the data and append a transformed data column like this:

```
myDataFrame = read.csv( file="RatLives.csv" )
myDataFrame = cbind( myDataFrame , DaysLiveSq = myDataFrame$ DaysLive^2 )
yName="DaysLiveSq"
```

Change the specification of the ROPEs to be appropriate to the transformed data. Do the groups appear to differ in their central tendencies and variances on the days-squared scale? Does the value of the normality parameter suggest that the data have outliers relative to a normal distribution on the days-squared scale? Is the posterior effect size on the days-squared scale much different than the posterior effect size on the days scale from the previous part?

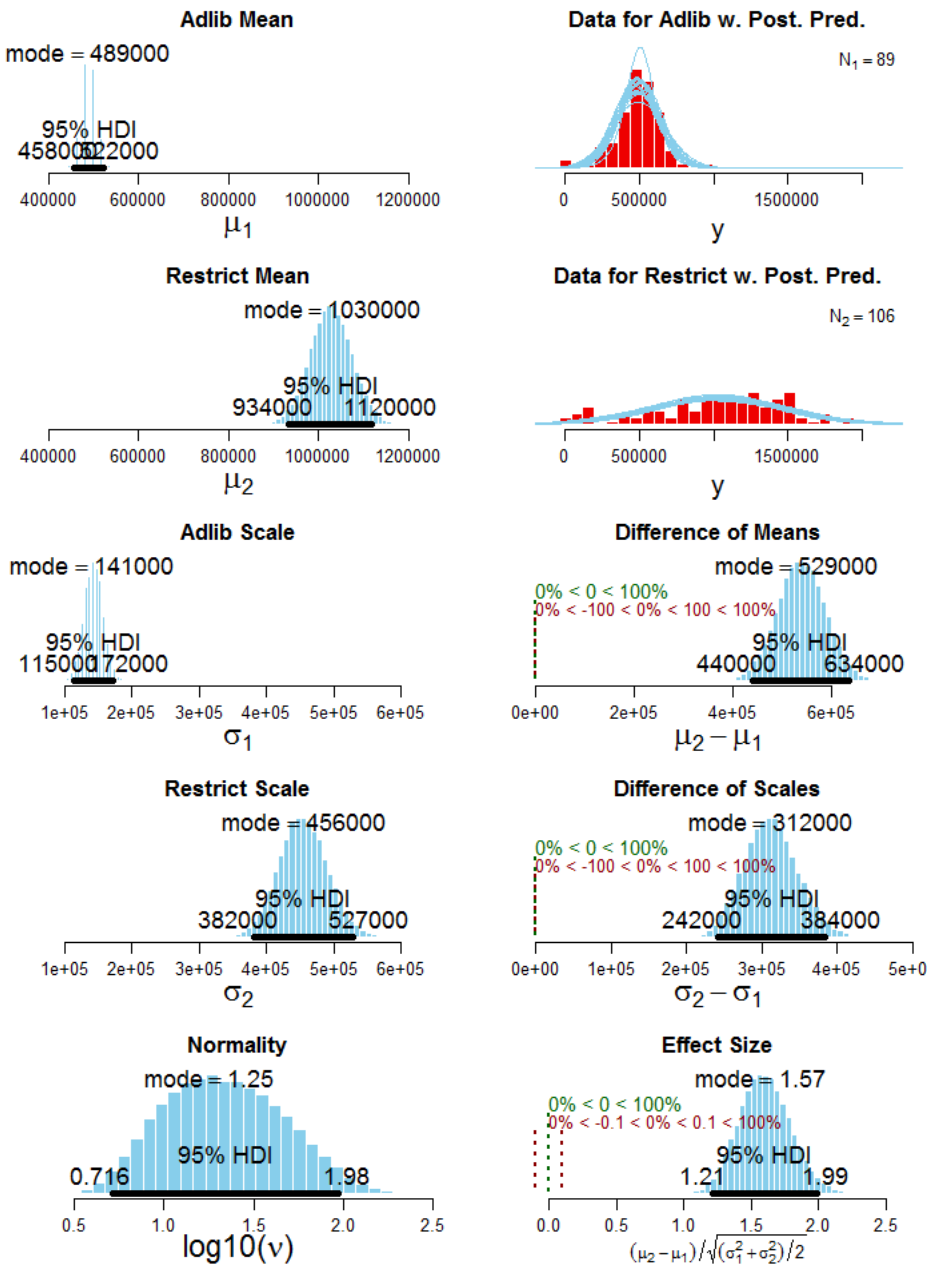
I used the following modified R code:

```
myDataFrame = read.csv( file="RatLives.csv" )
xName="Group"
myDataFrame = cbind( myDataFrame , DaysLiveSq = myDataFrame$DaysLive^2 )
yName="DaysLiveSq"
fileNameRoot = "RatLives-DaySq-"
RopeMuDiff=c(-100,100) ; RopeSdDiff=c(-100,100) ; RopeEff=c(-0.1,0.1)
```

The ROPEs on mu and sigma were set to c(-10^2,+10^2) to be consistent with the previous part of the exercise. That is, because the data were squared, the ROPE limits were squared. The ROPE on the effect size, however, stayed the same, because the effect size is already standardized (i.e., divided by the scale of the data).

Results are shown below. The differences of mu and sigma are clearly large and non-zero. The effect size is also quite large and non-zero, although on the squared-data the modal effect size is only about 1.6 while on the original data the modal effect size was about 2.1. The normality parameter is much larger than when using the original data, and the squared data appear to be reasonably normally distributed.

In this case, the parameters on the original scale are more directly interpretable, but the model predicts symmetric data distributions. When using the squared scale, the parameter values can only be interpreted on that scale, but the data are more nearly symmetric within groups.



Exercise 16.3

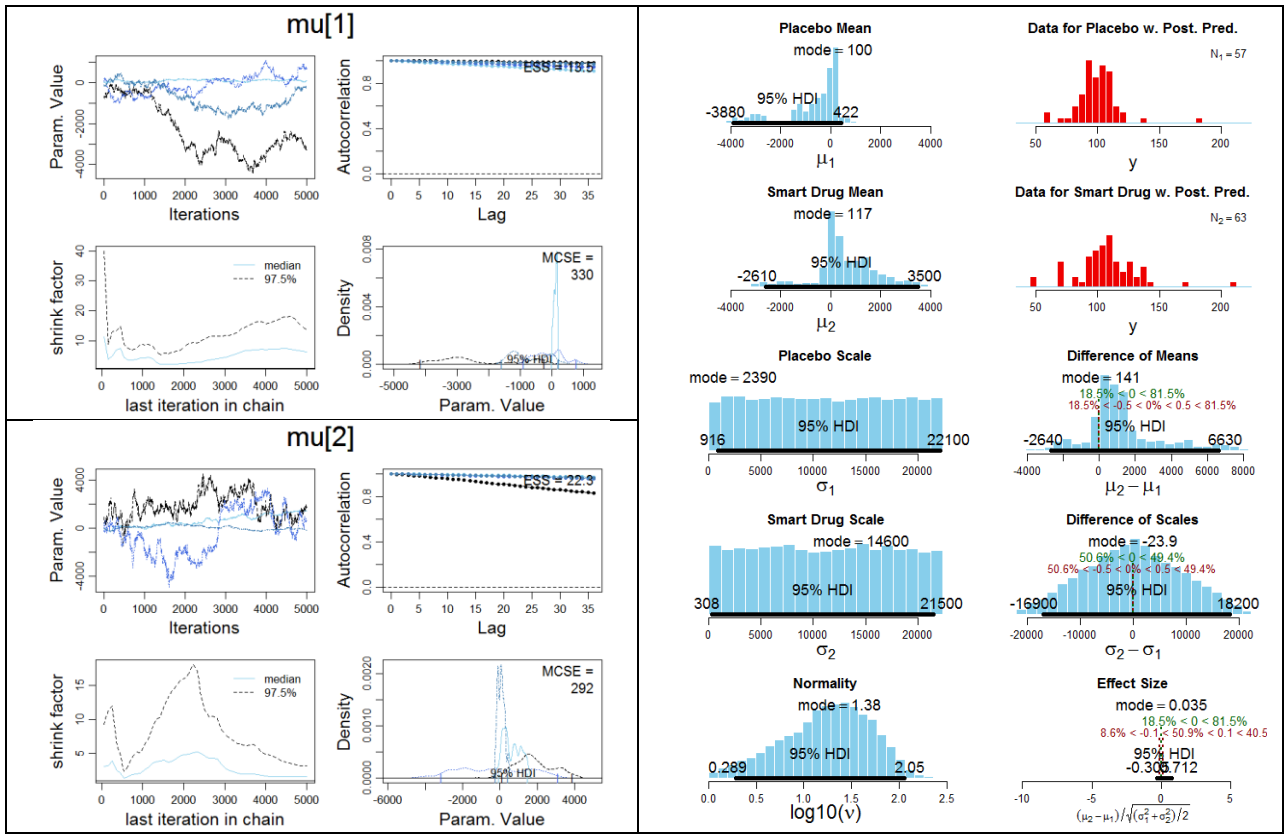
Exercise 16.3. [Purpose: For two groups, examine the implied prior on the effect size and on the differences of means and scales.]

(A) Modify the script `Stan-Ymet-Xnom2grp-MrobustHet-Example.R` and functions in `Stan-Ymet-Xnom2grp-MrobustHet.R` as needed to display the prior distribution. See Section 14.3.4, p. 412, for details. Explain what changes you made, and include the graphical output. Does Stan have convergence problems?

In the file Stan-Ymet-Xnom2grp-MrobustHet.R, in the model specification, comment out the lines the increment the log-likelihood of the data, as follows:

```
model {
  sigma ~ uniform( unifLo , unifHi ) ; // vectorized
  mu ~ normal( meanY , normalSigma ) ; // vectorized
  nuMinusOne ~ exponential( expLambda ) ;
//  for ( i in 1:Ntotal ) {
//    y[i] ~ student_t( nu , mu[x[i]] , sigma[x[i]] ) ;
//  }
}
```

Unfortunately Stan has convergence problems because the gradient on the prior is so flat. In particular, the chains for mu1 and mu2 are badly autocorrelated, as shown in the left below:



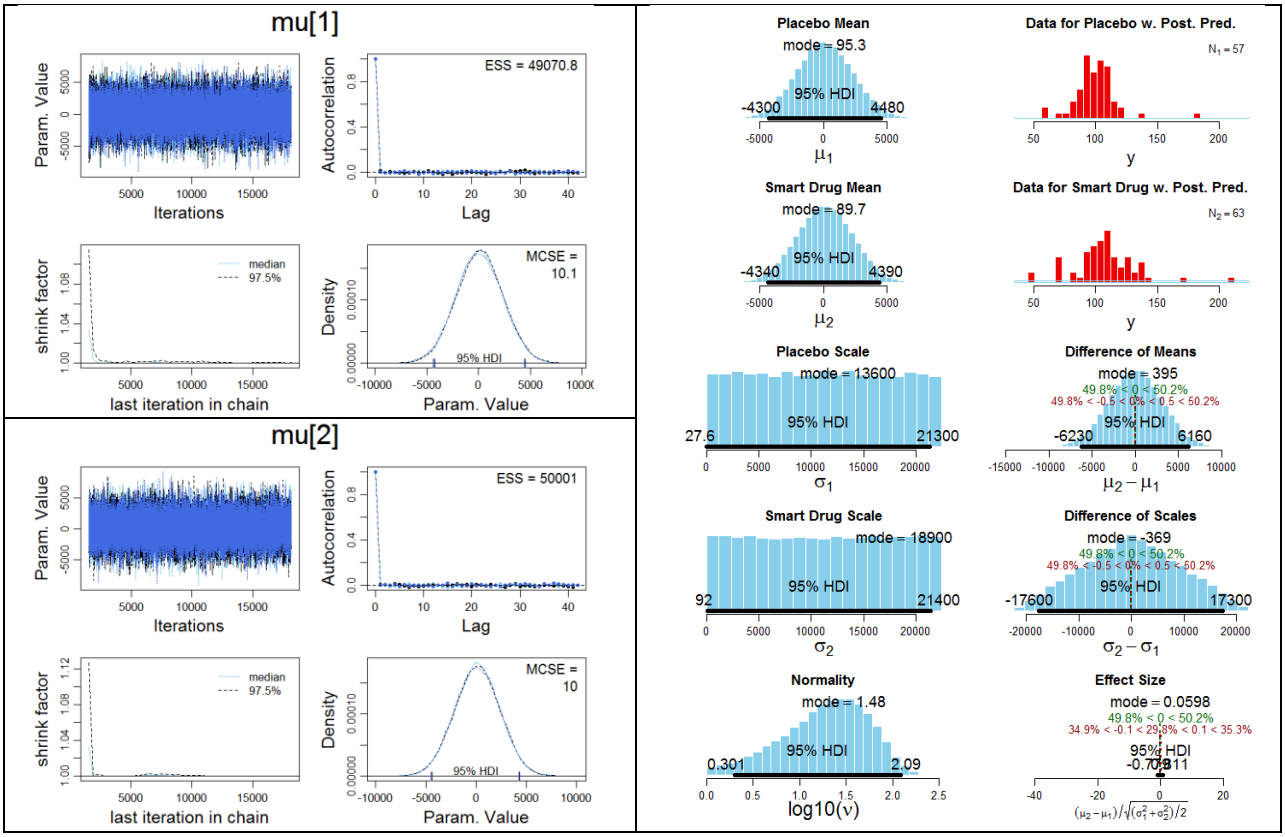
The sigma and nu parameters are also autocorrelated, but not as badly as the mu parameters. The histograms of prior differences and effect sizes are shown in the right above. Because of the autocorrelation, it's difficult to see whether the prior on the mu's or difference of mu's makes sense.

(B) Modify the JAGS version of the program to sample from the prior. Refer to Section 8.5, p. 211, for a reminder of how to sample from the prior in JAGS.

In the file Jags-Ymet-Xnom2grp-MrobustHet.R, change the data specification to comment out the y values, like this:

```
dataList = list(
#   y = y ,
  x = x ,
  Ntotal = Ntotal ,
  meanY = mean(y) ,
  sdY = sd(y)
)
```

The resulting MCMC chain mixes perfectly:



(C) From the previous two parts, is the prior on the effect size and differences suitably "noncommittal"? Briefly discuss.

The priors on the mu's are extremely broad (95% HDI from about -4,400 to +4,400) relative to the mean of the data (around 100), and the prior on the sigma's is very broad (about 0 to 25,000) relative to the scale of the data (around 15 or less). Notice that the shape of the prior is just as intended: normal on the mu's and uniform on the sigma's. Importantly, the implied prior on the difference of means is very broad around zero, so it imposes very little constriction on the posterior, and the same goes for the difference of scales.

Chapters 17

Exercise 17.1

Exercise 17.1. [Purpose: Apply linear model to quadratic data and do a posterior predictive check.]

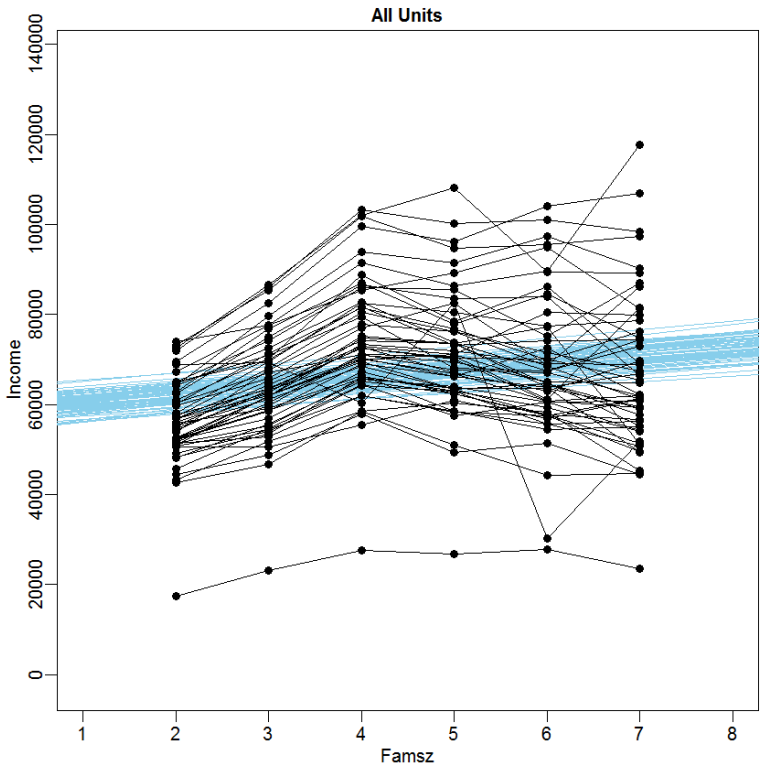
(A) Change the script `Jags-Ymet-XmetSsubj-MrobustHier-Example.R` so it uses the family income data. (Notice how simple it is to do that.) Are the data well described by the linear model? What exactly is the "systematic discrepancy" between model and data? Don't just say, "the data are curved." What exactly does that mean in terms of how the data deviate from the model and where?

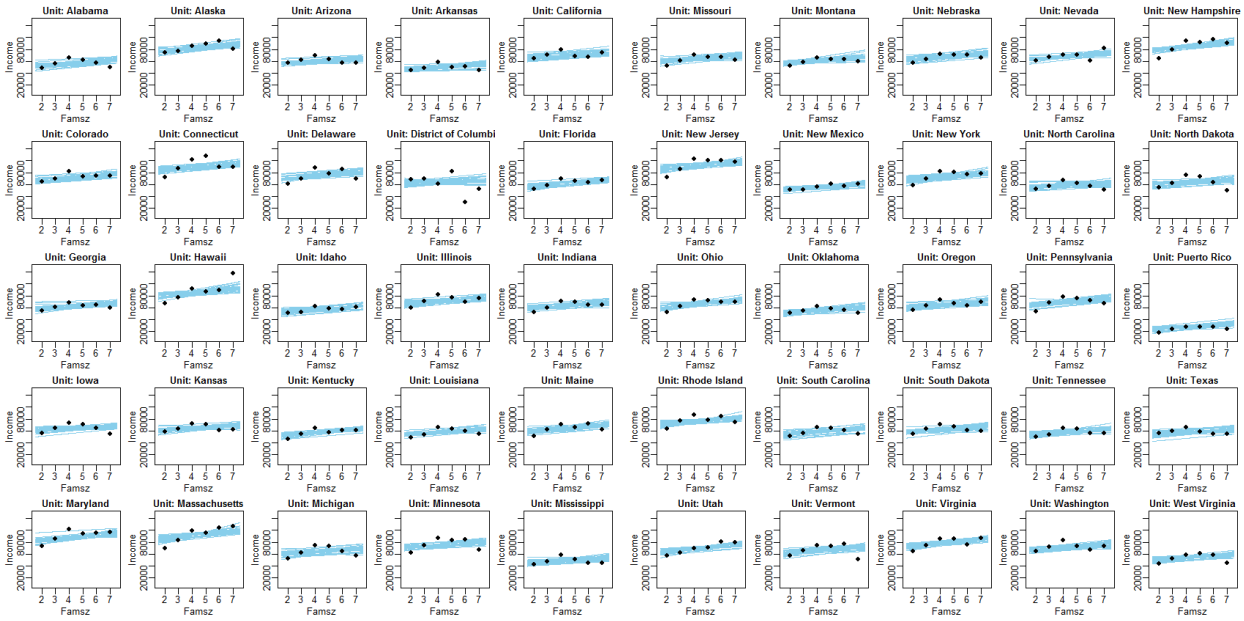
(B) Is there a way of rejecting the linear model without reference to any other model? In other words, how might you compute a "Bayesian p value" for this situation? Should you? *See Kruschke (2013b) for information and references.*

(A) At the top of the script, we simply change the file specification as follows:

```
myData = read.csv( file="IncomeFamszState.csv" )
xName = "Famsz" ; yName = "Income" ; sName="State"
```

The result looks like this:





The data are not well described by the linear model because of the systematic discrepancy between predicted trends and actual data, that might be described in detail this way: For x values at the low and high ends of the range, the data tend to fall below the predicted trend, but for x values in the middle of the range, the data tend to fall above the predicted trend.

(B) The article by Kruschke (2013b) discusses this sort of situation, namely, quadratic discrepancy from a linear trend model and how to assess that discrepancy in a Bayesian way. The title of the article is, "Posterior predictive checks can and should be Bayesian." The article argues against defining an arbitrary measure of discrepancy between predicted and actual data and then making a sampling distribution of that discrepancy to compute a *p* value. (See Chapter 11 of this book for reasons to eschew *p* values.) The article argues instead that the model should be expanded to incorporate the desired systematicity –in this case a quadratric trend— and then the parameter on that systematicity can be explicitly estimated in a Bayesian way.

Exercise 17.2

Exercise 17.2. [Purpose: Observe the autocorrelation in JAGS when data are not standardized.] Change the (simple, nonhierarchical) JAGS program `Jags-Ymet-Xmet-Mrobust.R` so that it uses the raw data instead of the standardized data. Be sure to rename it so that you don't destroy the original program. You will have to remove the standardization from the data block and remove the transformation to the original scale. Make sure that the prior is appropriate for the original scale (perhaps use the prior in the Stan version for guidance). Show the diagnostic graphs of the chains and discuss. Finally, run the chains a long time, with thinning if needed to conserve computer memory, and show that the chains eventually converge to the same posterior as when using standardized data.

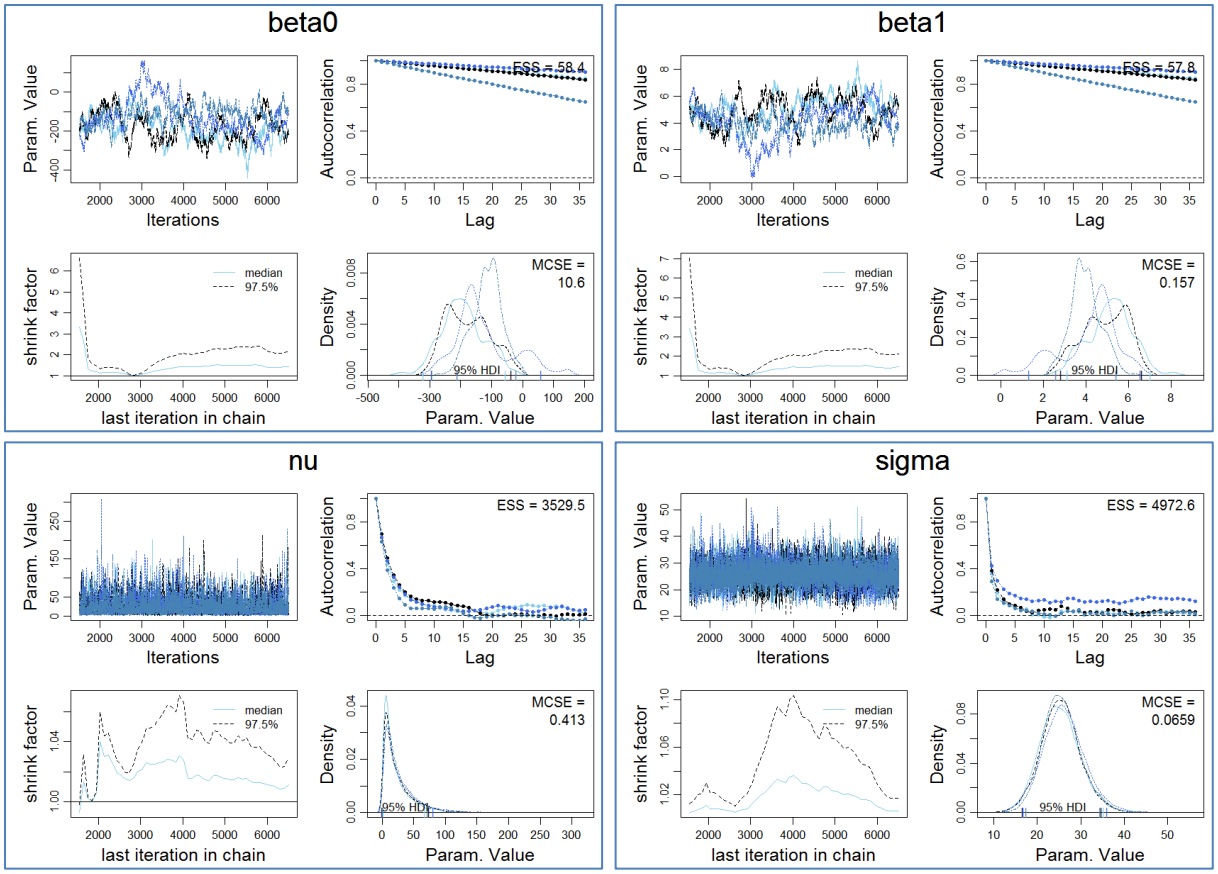
Here is one way to make changes in the scripts. First, Jags-Ymet-Xmet-Mrobust-Example.R is changed so it calls a version of the program prefixed with "Exercise.17.2-":

```
fileNameRoot = "Exercise.17.2-HtWtData30-Jags-"
source("Exercise.17.2-Jags-Ymet-Xmet-Mrobust.R")
```

Then the script Jags-Ymet-Xmet-Mrobust.R is saved as Exercise.17.2-Jags-Ymet-Xmet-Mrobust.R with these changes to its model specification:

```
  # Standardize the data:
  data {
    Ntotal <- length(y)
    xm <- mean(x)
    ym <- mean(y)
    xsd <- sd(x)
    ysd <- sd(y)
    for ( i in 1:length(y) ) {
      # zx[i] <- ( x[i] - xm ) / xsd
      # zy[i] <- ( y[i] - ym ) / ysd
      zx[i] <- x[i] # original scale
      zy[i] <- y[i] # original scale
    }
  }
  # Specify the model for standardized data:
  model {
    for ( i in 1:Ntotal ) {
      zy[i] ~ dt( zbeta0 + zbeta1 * zx[i] , 1/zsigma^2 , nu )
    }
    # Priors vague on standardized scale:
    # zbeta0 ~ dnorm( 0 , 1/(10)^2 )
    # zbeta1 ~ dnorm( 0 , 1/(10)^2 )
    # zsigma ~ dunif( 1.0E-3 , 1.0E+3 )
    # Priors vague on original scale:
    zbeta0 ~ dnorm( 0 , 1/(10*abs(xm*ysd/xsd))^2 ) # same as Stan version
    zbeta1 ~ dnorm( 0 , 1/(10*abs(ysd/xsd))^2 )    # same as Stan version
    zsigma ~ dunif( ysd/1000 , ysd*1000 )          # same as Stan version
    nu <- nuMinusOne+1
    nuMinusOne ~ dexp(1/29.0)
    # Transform to original scale:
    # beta1 <- zbeta1 * ysd / xsd
    # beta0 <- zbeta0 * ysd  + ym - zbeta1 * xm * ysd / xsd
    # sigma <- zsigma * ysd
    beta1 <- zbeta1 # original scale
    beta0 <- zbeta0 # original scale
    sigma <- zsigma # original scale
  }
```

Diagnostic plots of the chains look like the following:

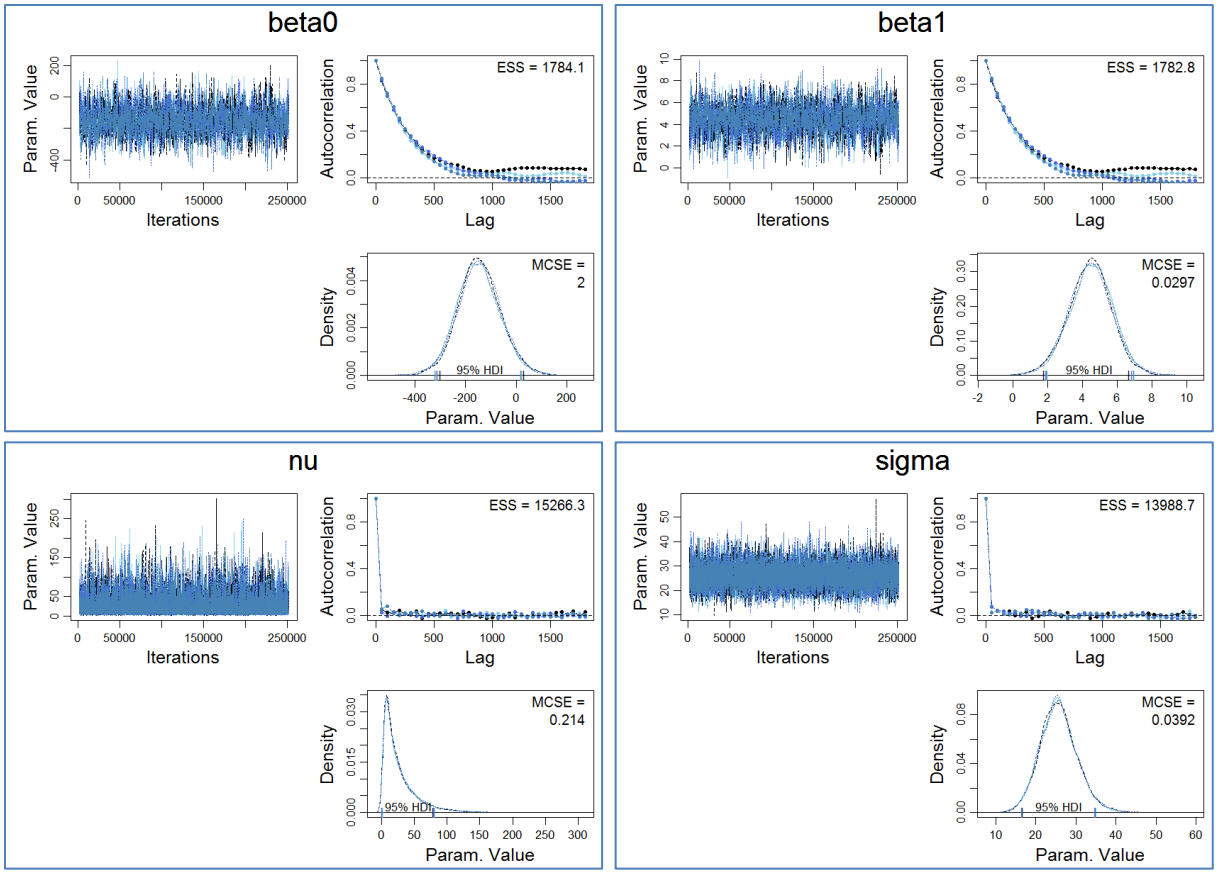


You can see above that the chains for beta0 and beta1 are badly autocorrelated. Even for 20,000 steps in the chain, the effective sample size is only about 58. This result dramatically illustrates the benefit of standardizing the data for MCMC sampling of the posterior distribution.

When the chain is run a long time, with thinning to conserve computer memory, the chains do eventually converge to the same values that are obtained by standardizing the data. In Exercise.17.2-Jags-Ymet-Xmet-Mrobust.R, specify:

```
thinSteps = 50  # instead of thinSteps = 1
```

The results are shown below, where it can be seen that even with thinning of 50, there is still considerable autocorrelation, but at least it is clear that the chains have converged and mixed well, even if they don't have a big enough ESS for very stable estimates of HDI limits.



Exercise 17.3

Exercise 17.3. [Purpose: Examining chain convergence in JAGS and Stan.] As mentioned in Footnote 5 (p. 504), both JAGS and Stan show some difficulty converging when the quadratic-trend model is applied to the fictitious data of Figure 17.5. The JAGS programs are provided in files `Jags-Ymet-XmetSsubj-MrobustHierQuadWt-Example.R` and `Jags-Ymet-XmetSsubj-MrobustHierQuadWt.R`. The corresponding Stan programs are provided in files `Stan-Ymet-XmetSsubj-MrobustHierQuadWt-Example.R` and `Stan-Ymet-XmetSsubj-Mrobust HierQuadWt.R`. The Stan model specification is shown below so that you can study it and compare it to the JAGS version in Section 17.4 without having to be at your computer:

...

Review Section 14.4 (p. 414) for hints about programming Stan.

(A) Run Stan on the family-income data, so it achieves the same ESS as the JAGS program for the group-level trend coefficients. How long (in real time) do Stan and JAGS take? Does Stan more consistently converge than JAGS? Does Stan produce better chains for the normality and noise parameters?

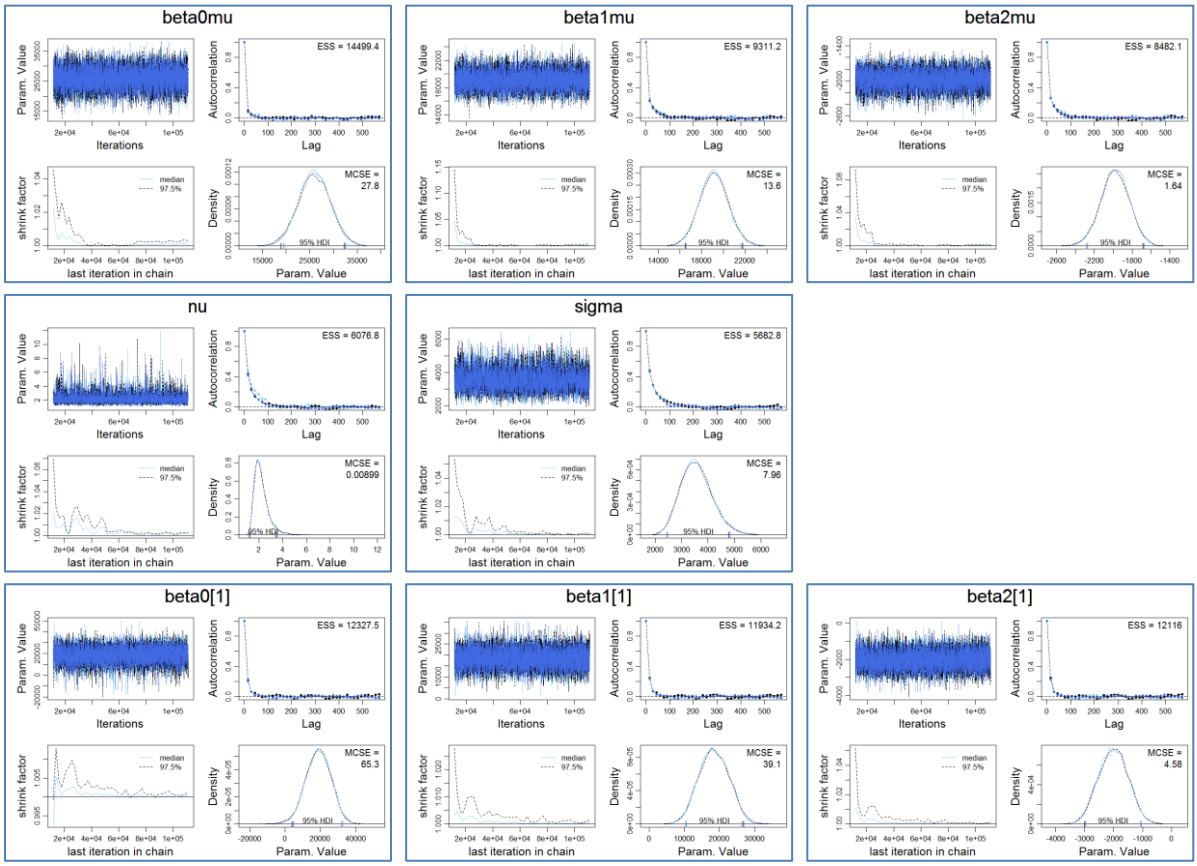
(B) Now repeat on the fictitious data of Figure 17.8, which typically gives JAGS and Stan troubles of differing sorts as described Footnote 5 (p. 504). Try to produce examples of these troubles and discuss. Which type of trouble is more tolerable, autocorrelation in the normality parameter (in JAGS) or "bumps" in the regression coefficients (in Stan)?

(A) First, here is the result of JAGS on the family-income data, using these commands:

```
source("Jags-Ymet-XmetSsubj-MrobustHierQuadWt.R")
#------------------------------------------------------------------------------
# Generate the MCMC chain:
startTime = proc.time()
mcmcCoda = genMCMC( data=myData ,
                    xName=xName , yName=yName , sName=sName , wName=wName ,
                    numSavedSteps=20000 , thinSteps=15 , saveName=fileNameRoot )
stopTime = proc.time()
duration = stopTime - startTime
show(duration)
   user  system elapsed
  23.68    1.97 1402.03
```

1402 seconds is 23.4 minutes. Importantly, this was running 3 chains in parallel using runjags.

The resulting diagnostics graphs, which also indicate ESS, are as follows:



You can see that even with 20,000 steps and thinning of 15 to conserve memory, the ESS for beta2mu is 8,482, for nu is 6,077, and for sigma is 5,683. But the ESS for the other parameters is generally well over 10,000. In all cases, the chains are well converged, even when ESS is not as high as we might want.

Now we run Stan on the family income data, using these R commands:

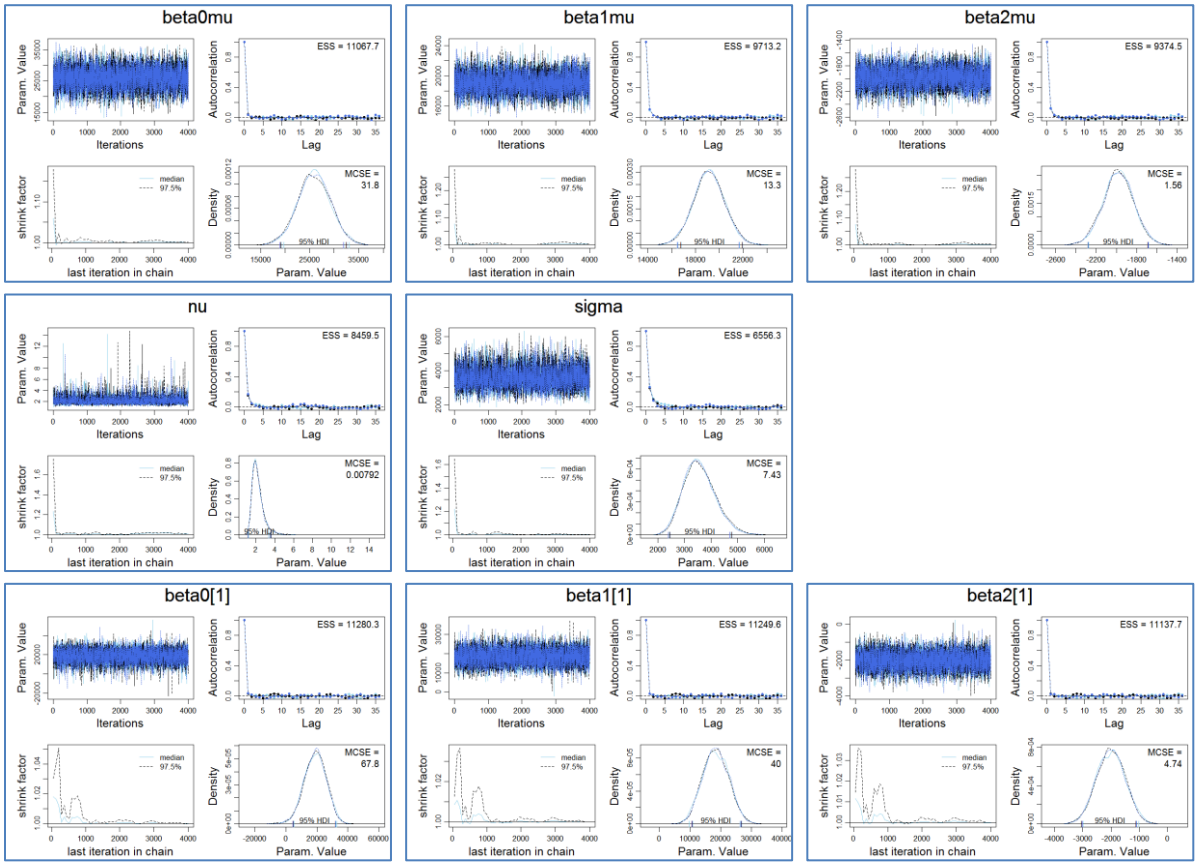
```
source("Stan-Ymet-XmetSsubj-MrobustHierQuadWt.R")
```

```
# Generate the MCMC chain:
startTime = proc.time()
mcmcCoda = genMCMC( data=myData ,
                    xName=xName , yName=yName , sName=sName , wName=wName ,
                    numSavedSteps=12000 , thinSteps=5 , saveName=fileNameRoot )
stopTime = proc.time()
duration = stopTime - startTime
show(duration)
```

```
   user  system elapsed
2110.80    9.72 2202.68
```

Notice that 12,000 steps are specified with a thinning of 5 to conserve computer memory. This is less than what was specified for JAGS. The elapsed time is about 36.7 minutes, but running the chains sequentially not in parallel.

Here are the resulting chain diagnostics:

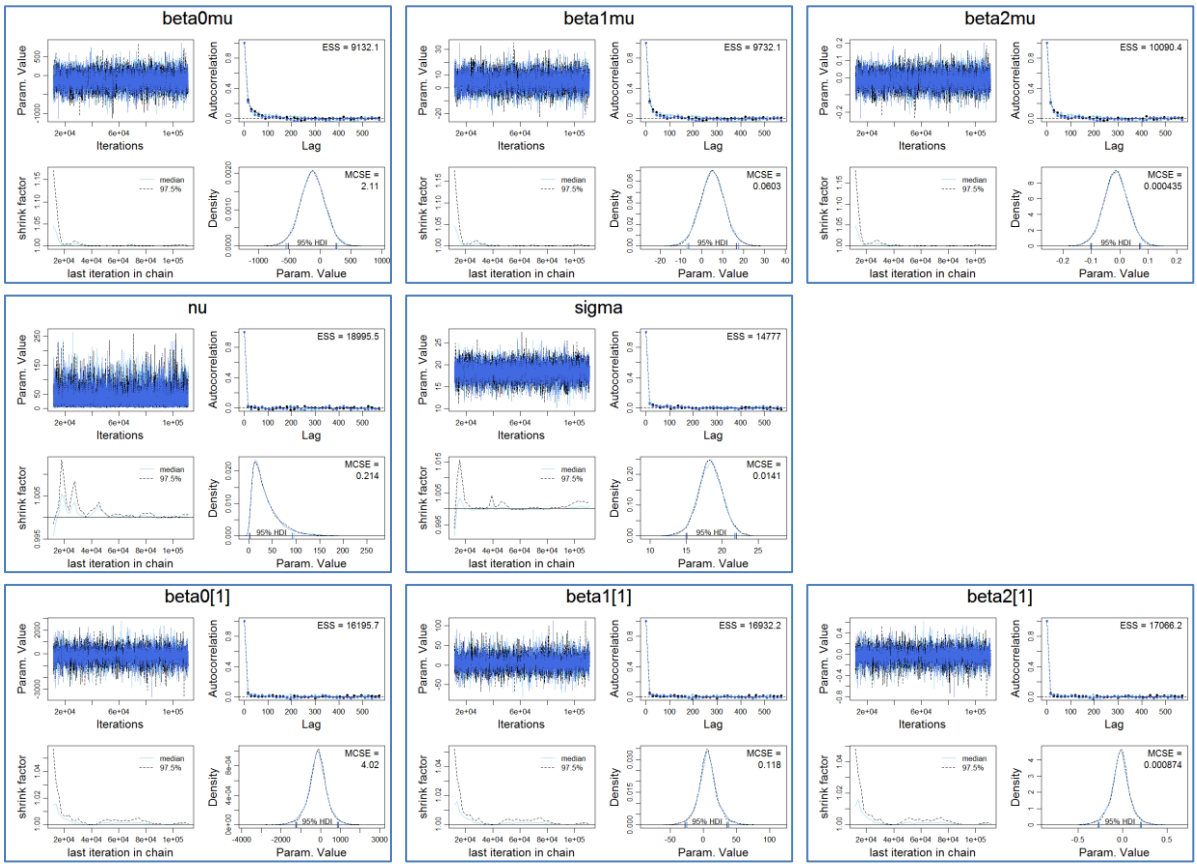


You can see that the ESS for these parameters is 10% to 40% better for Stan than for JAGS, even though Stan took fewer actual steps in the chain. Stan took almost 60% more real time to do it, however. Stan does relatively better on the nu and sigma parameters than JAGS, but Stan still does not achieve an ESS on nu and sigma as high as the other parameters. If JAGS were not run with three parallel chains, it would take longer than Stan.

(B) Now we repeat, but using HierLinRegressData.csv. First, with JAGS, using 3 chains in parallel:

```
   user  system elapsed
   9.80    0.59  381.64
```

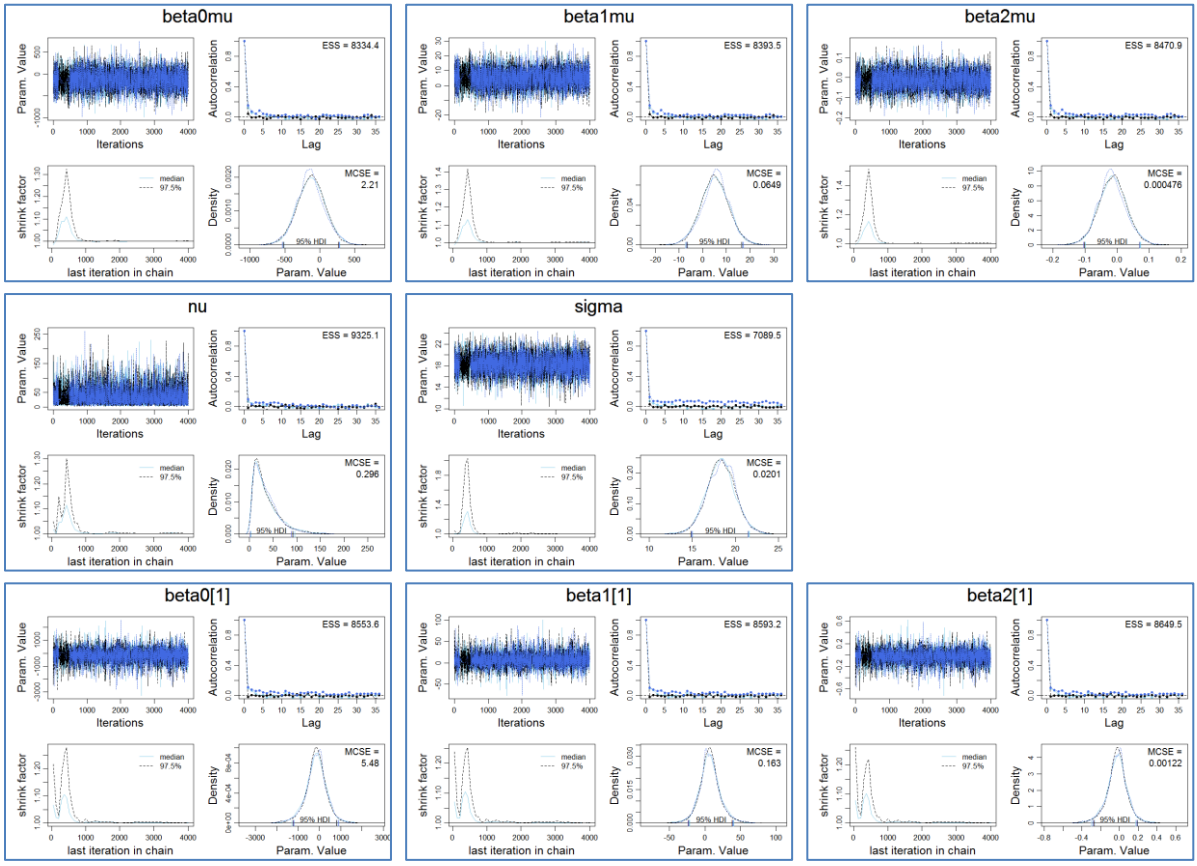
Contrary to footnote 5 on p. 504 of the book, all chain diagnostics were quite good for this particular run of JAGS, including the normality parameter:



The results for Stan:

```
   user  system elapsed
1442.59    0.43 1485.50
```

Stan takes much longer than JAGS, even accounting for the 3 parallel chains used by runjags. And, unfortunately in this case, Stan suffers from some "stuck" chains that are severely autocorrelated for portions of their runs, as can be seen below for about iterations 100-500. Unfortunately, the "stuck" portions of the chains introduce unsmooth spikes in the in the posterior sample.



Chapters 18

Exercise 18.1

Exercise 18.1. [Purpose: Understanding multiplicative interaction.] Consider the right panel of Figure 18.8. Use Equation 18.4 to compute the slope of the dark lines when $x_1 = 0$ and $x_1 = 10$. Show your work (algebraically). Confirm your answer by computing the slopes from the figure: Visually inspect how much each line rises as x_2 goes from 0 to 10, and compute the ratio of the rise over the run for each line.

In the right panel of Fig 18.8, the equation of the surface is $y = 10 + -1x_1 + (2 + 0.2\ x_1)\ x_2$. The slope on x_2 is therefore $(2 + 0.2\ x_1)$. When x_1=0, the slope is 2+0.2*0 = 2.0. When x_1=10, the slope is 2+0.2*10 = 4.0.

Visually, at the left edge of the graph, where x_1=0, the surface starts at y=10, and rises to y=30 as x_2 goes from 0 to 10. Hence the slope is (30-10)/10=2.0, which verifies the previous algebra. At the right edge of the graph, , where x_1=10, the surface starts at y=0, and rises to y=40 as x_2 goes from 0 to 10. Hence the slope is (40-0)/10=4.0, which verifies the previous algebra.

Exercise 18.2

Exercise 18.2. [Purpose: Understand the effect of including/excluding predictors, even when they are not correlated. This is also a prelude to analysis of covariance.] The fictitious data in Figure 18.1, p. 511, involves two predictors that are uncorrelated. The data are in the file MultLinRegrPlotUnif.csv.

(A) Run a multiple regression on the two predictors. What is the correlation of the two predictors? Are the estimates of the intercept, slopes, and standard deviation close to the values indicated in Figure 18.1?

(B) Run the regression of y on the single predictor x_1. What parameter estimates have noticeably changed? In particular, why is the estimate of σ so much bigger? Discuss with respect to the upper right panel of Figure 18.1.

(C) Repeat the previous two parts, but this time using only lines 101–150 of the data file (i.e., fewer data points). How does the interpretation of the regression coefficient on x_1 change when x_2 is included?

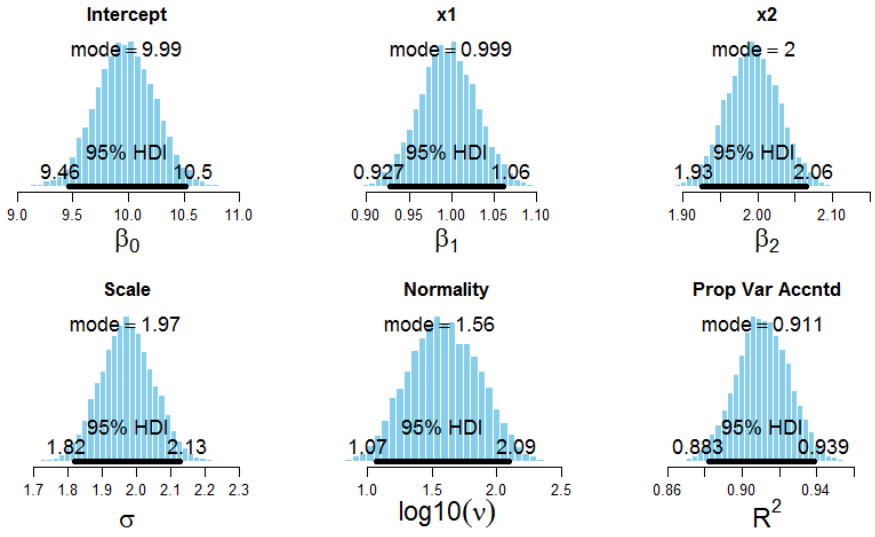
(A) Using the file Jags-Ymet-XmetMulti-Mrobust-Example.R, the data are read in with this R code:

```
myData = read.csv( file="MultLinRegrPlotUnif.csv" )
yName = "y" ; xName = c("x1","x2")
fileNameRoot = "MultLinRegrPlotUnif-"
numSavedSteps=11000 ; thinSteps=2
```

The console output shows the correlations of the predictors:

```
CORRELATION MATRIX OF PREDICTORS:
       x1    x2
x1 1.000 0.007
x2 0.007 1.000
```

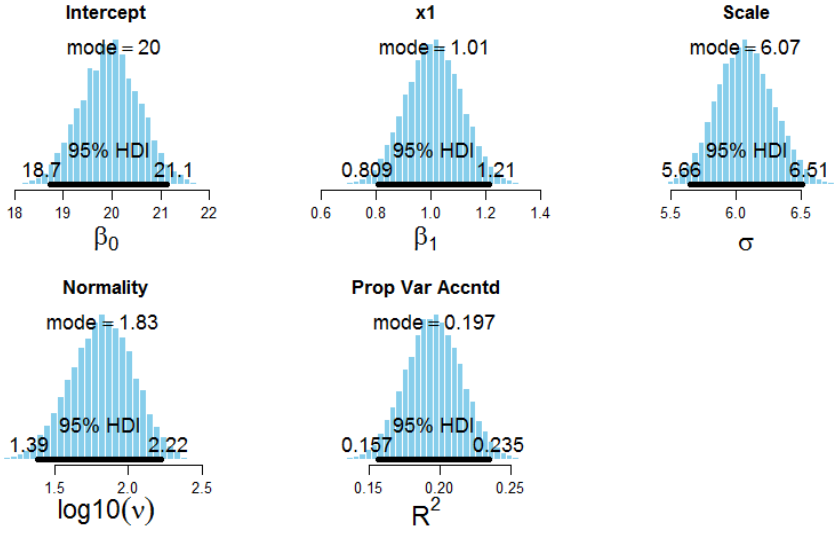
The estimates of the parameters are extremely close to the generating values shown in Figure 18.1:



(B) We run the analysis on only the first predictor by specifying

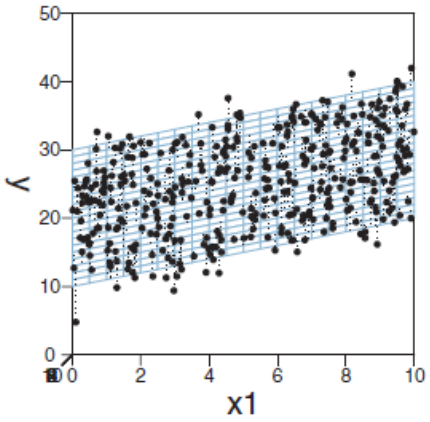
```
myData = read.csv( file="MultLinRegrPlotUnif.csv" )
yName = "y" ; xName = c("x1")
fileNameRoot = "MultLinRegrPlotUnif-"
numSavedSteps=11000 ; thinSteps=2
```

The resulting parameter estimate now looks like this:



In particular, the intercept has changed from 10 to 20 and the standard deviation (i.e., scale or sigma) has changed from 2 to 6. (The proportion of variance accounted for has also decreased

substantially, reflecting the increase in residual standard deviation.) The estimated parameter values nicely correspond with the scatterplot in the upper right of Figure 18.1, reproduced here:



We can see that the central spine of the scatter plot has intercept around 20 and a slope of about 1.0 (because y goes up about 10 when x1 goes up 10). And the spread of data seems to go about 12 units above and below the spine, which corresponds roughly to a standard deviation of about 6.

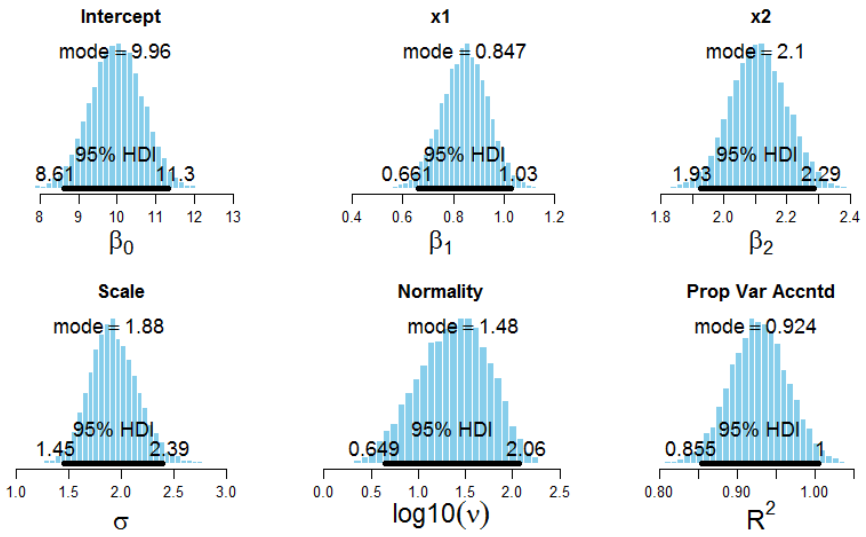
(C) Repeating the above with fewer data points:

```
myData = read.csv( file="MultLinRegrPlotUnif.csv" )
myData = myData[101:150,]
yName = "y" ; xName = c("x1","x2")
fileNameRoot = "MultLinRegrPlotUnif-"
numSavedSteps=11000 ; thinSteps=2
```

The console output shows the still tiny correlations of the predictors:

```
CORRELATION MATRIX OF PREDICTORS:
        x1     x2
x1  1.000 -0.075
x2 -0.075  1.000
```

The resulting parameter estimates are shown below:

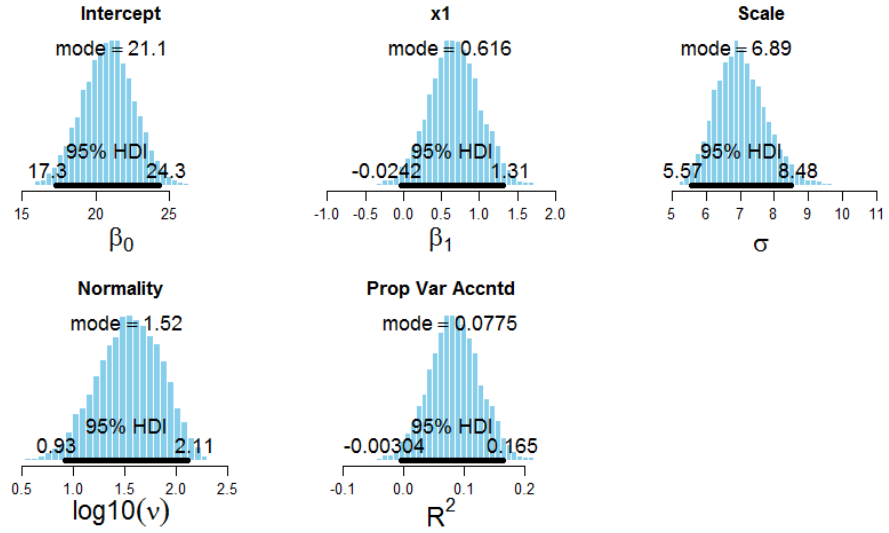


The parameter estimates are not so close to the generating values as when using the full data set, and there is much more uncertainty, that is, the HDI's are much wider than when using the full data set.

Now we repeat the single-predictor analysis with the reduced data set:

```
myData = read.csv( file="MultLinRegrPlotUnif.csv" )
myData = myData[101:150,]
yName = "y" ; xName = c("x1")
fileNameRoot = "MultLinRegrPlotUnif-"
numSavedSteps=11000 ; thinSteps=2
```

The resulting parameter estimate is shown below:



The reduced data set still shows a positive trend in the slope on x1, but it is very uncertain, with the 95% HDI including zero. ***Here's the point of the exercise: Notice that even***

when the predictors are uncorrelated, when you include the second predictor, you get a more precise estimate of the slope on the first predictor. Sometimes this can make the difference between deciding that a predictor has non-zero influence or not.

Exercise 18.3

Exercise 18.3. [Purpose: View the prior distribution.] Figure 18.7 (p. 522) showed the prior distribution for a multiple linear regression. Your job for this exercise is to produce the graph. To do so, read Footnote 2 (p. 523), about commenting out the specification of `zy[i]` in the JAGS data block.

The data are defined at the beginning of the script Jags-Ymet-XmetMulti-Mrobust-Example.R as follows:

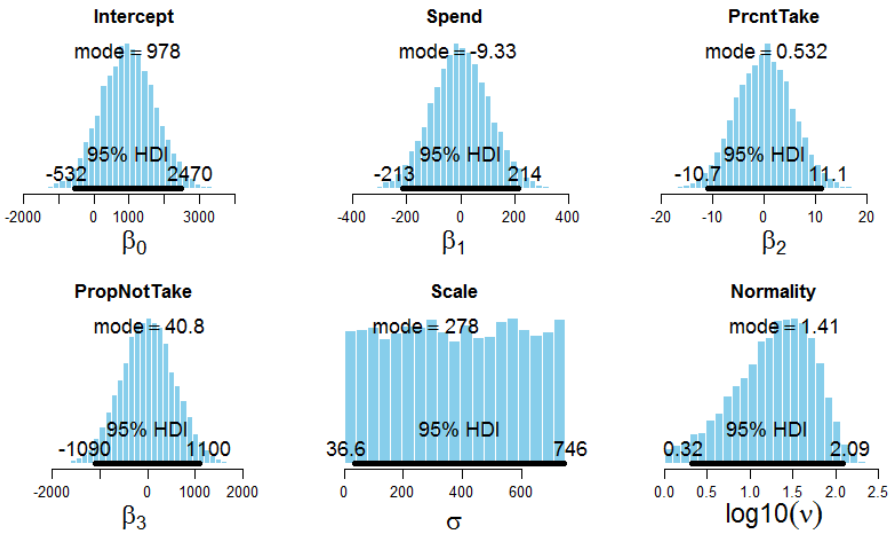
```
myData = read.csv( file="Guber1999data.csv" )
PropNotTake = (100-myData[,"PrcntTake"])/100
myData = cbind( myData , PropNotTake )
yName = "SATT" ; xName = c("Spend","PrcntTake","PropNotTake")
fileNameRoot = "Guber1999data-Jags-Redund-"
numSavedSteps=15000 ; thinSteps=15
```

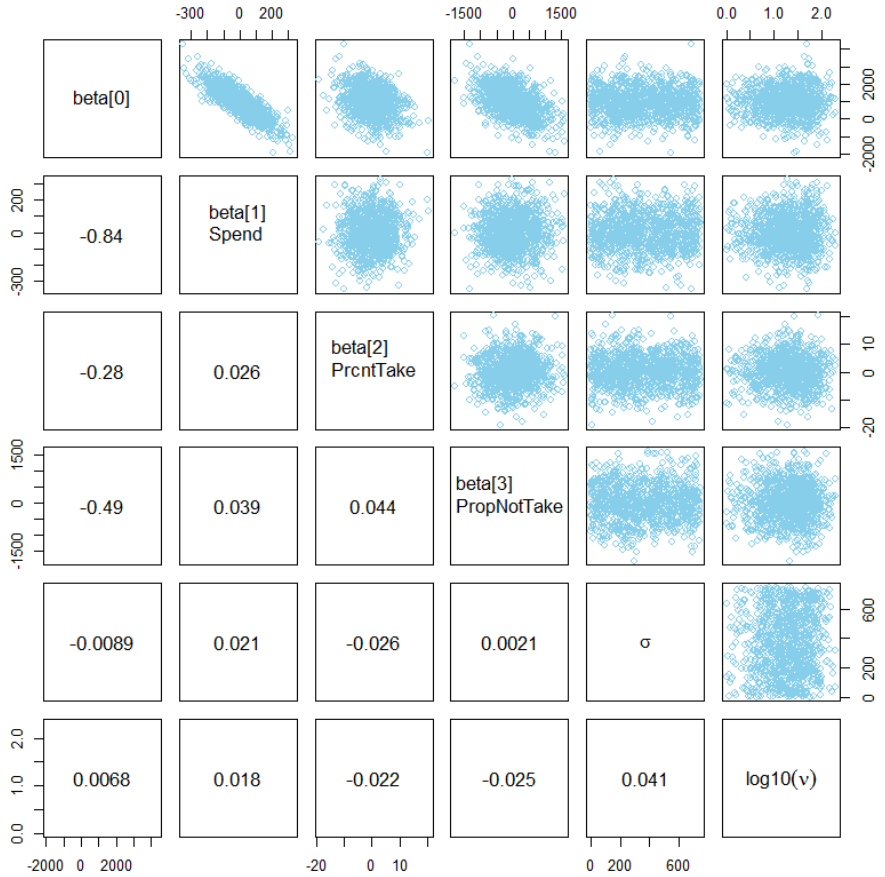
Then the model in Jags-Ymet-XmetMulti-Mrobust.R is modified according to Footnote 2, p. 523. Understanding this implementational detail is the main point of the exercise:

```
  data {
    ym <- mean(y)
    ysd <- sd(y)
  #  for ( i in 1:Ntotal ) {
  #    zy[i] <- ( y[i] - ym ) / ysd
  #  }
    for ( j in 1:Nx ) {
      xm[j]  <- mean(x[,j])
      xsd[j] <-   sd(x[,j])
      for ( i in 1:Ntotal ) {
        zx[i,j] <- ( x[i,j] - xm[j] ) / xsd[j]
      }
    }
  }
```

(The modified program file must be saved before it is called by the Example script. Don't forget to undo those changes before using the program for other applications.)

The result looks just like Figure 18.7, except of course for random differences in the MCMC sample:


Intercept
mode = 978
95% HDI
-532
2470
β0
Spend
mode = -9.33
95% HDI
-213
214
β1
PrcntTake
mode = 0.532
95% HDI
-10.7
11.1
β2
PropNotTake
mode = 40.8
95% HDI
-1090
1100
β3
Scale
mode = 278
95% HDI
36.6
746
σ
Normality
mode = 1.41
95% HDI
0.32
2.09
log10(ν)


beta[0]
-0.84
beta[1]
Spend
-0.28
0.026
beta[2]
PrcntTake
-0.49
0.039
0.044
beta[3]
PropNotTake
-0.0089
0.021
-0.026
0.0021
σ
0.0068
0.018
-0.022
-0.025
0.041
log10(ν)

Exercise 18.4

Exercise 18.4. [Purpose: Hands-on experience with variable selection and its sensitivity to priors.] Your goal in this exercise is to produce Figure 18.15 (p. 545) and explore some variations. The relevant programs are `Jags-Ymet-XmetMulti-MrobustVarSelect.R` and `Jags-Ymet-XmetMulti-MrobustVarSelect-Example.R`. For all the parts of this exercise, read the SAT data file with all four candidate predictors. At the top of `Jags-Ymet-XmetMulti-MrobustVarSelect-Example.R` uncomment and comment out lines so that you have the equivalent of

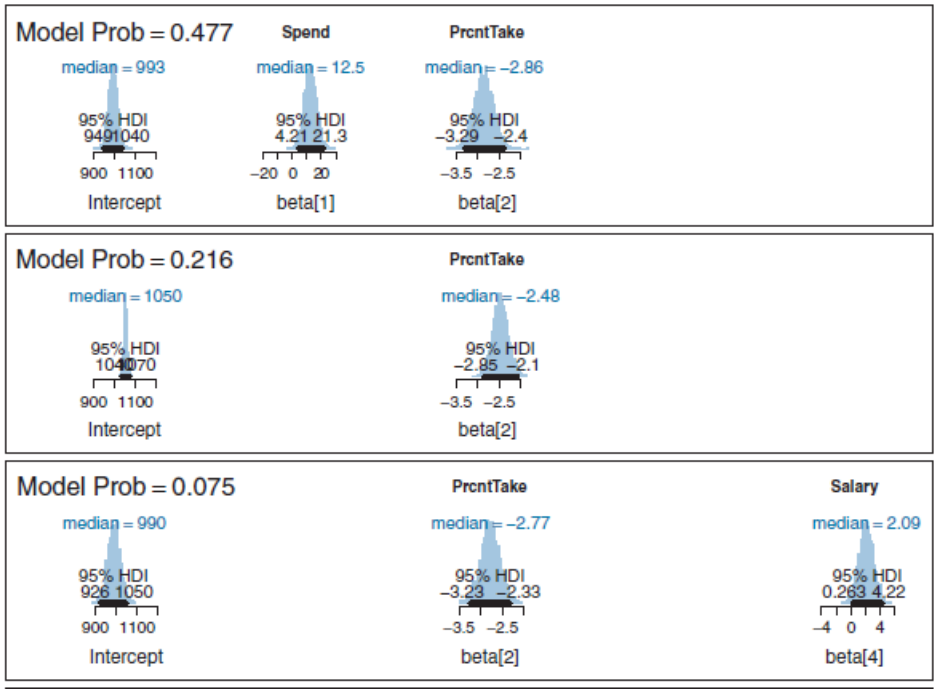
```
myData = read.csv( file="Guber1999data.csv" )
yName = "SATT"
xName = c("Spend","PrcntTake","StuTeaRat","Salary")
fileNameRoot = "Guber1999data-Jags-4X-VarSelect-" # change for distinct saved files
numSavedSteps=15000 ; thinSteps=20
```

(A) In the program `Jags-Ymet-XmetMulti-MrobustVarSelect.R`, be sure that the line

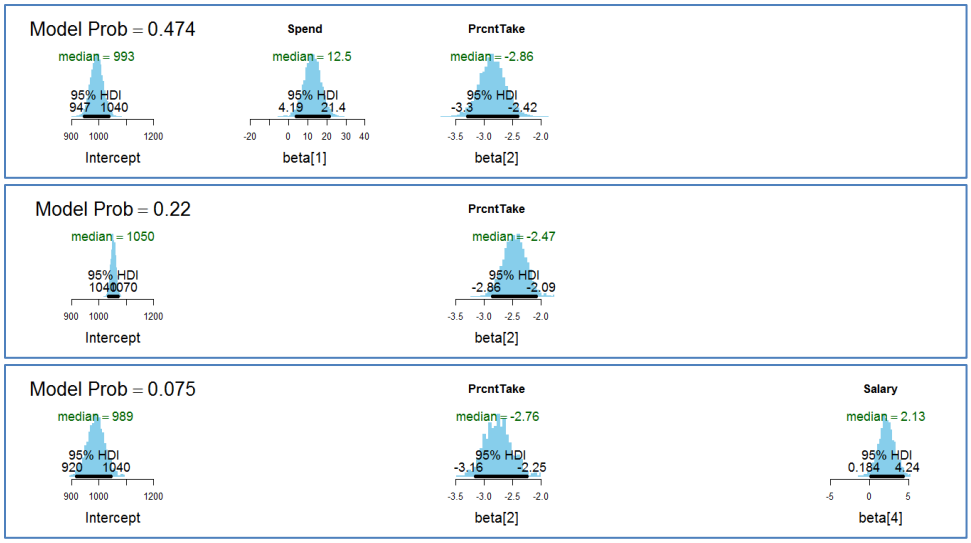
```
sigmaBeta ~ dgamma(1.1051,0.1051) # mode 1.0, sd 10.0
```

is being used (i.e., is the only line of its section not commented out). Run the high-level script. Does its output resemble Figure 18.15 (p. 545)? (It should.)

For convenient reference, here are the top three panels of Figure 18.15:



When running the program again as specified in the exercise, the corresponding output looks very similar except for random variation due to MCMC fluctuation:



Other panels are also similar. Thus, we have successfully replicated Figure 18.15 in the book.

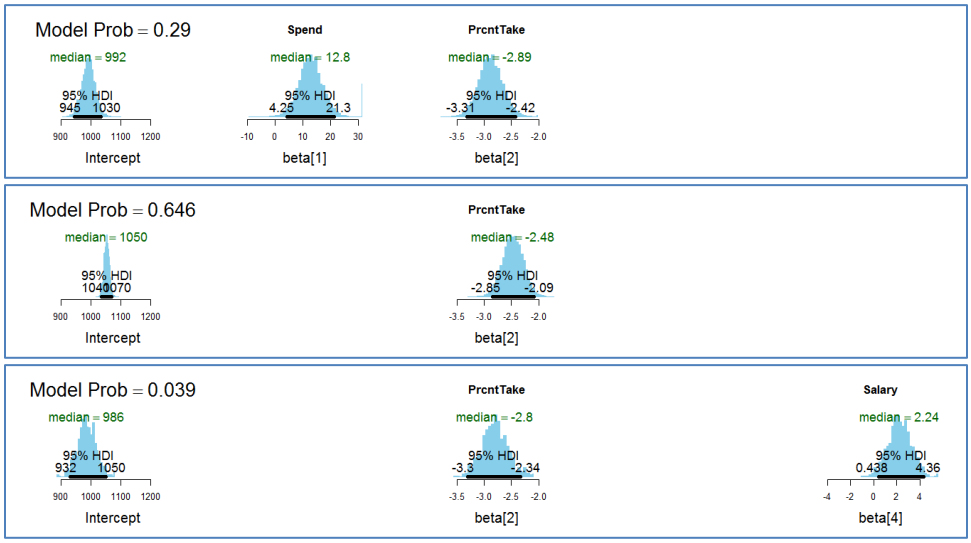
(B) In the program `Jags-Ymet-XmetMulti-MrobustVarSelect.R`, comment out the line for `sigmaBeta` that gives it a gamma prior, and instead use

```
sigmaBeta <- 10.0
```

Run the high-level script. In what ways is the posterior different than the previous part? Discuss model probabilities, inclusion probabilities, and HDIs of regression coefficients.

The distributed code might have a pre-commented line that has sigmaBeta <- 2.0; be sure you use sigmaBeta <- 10.0. (You need to save the changed file before calling it from the high-level script. Be sure to change it back before using the program for other purposes.)

With this fixed value of sigmaBeta, results are as shown below:



Notice in particular that the Model Probabilities are very different than in the previous part of the exercise. Now, the model that includes only PrcntTake (2nd panel above) has probability of

0.646, which is much greater than the probability of the model that includes both Spend and PrcntTake (0.290). Thus, a simple change in the prior on sigmaBeta has made a big change in the probabilities of including the predictors.

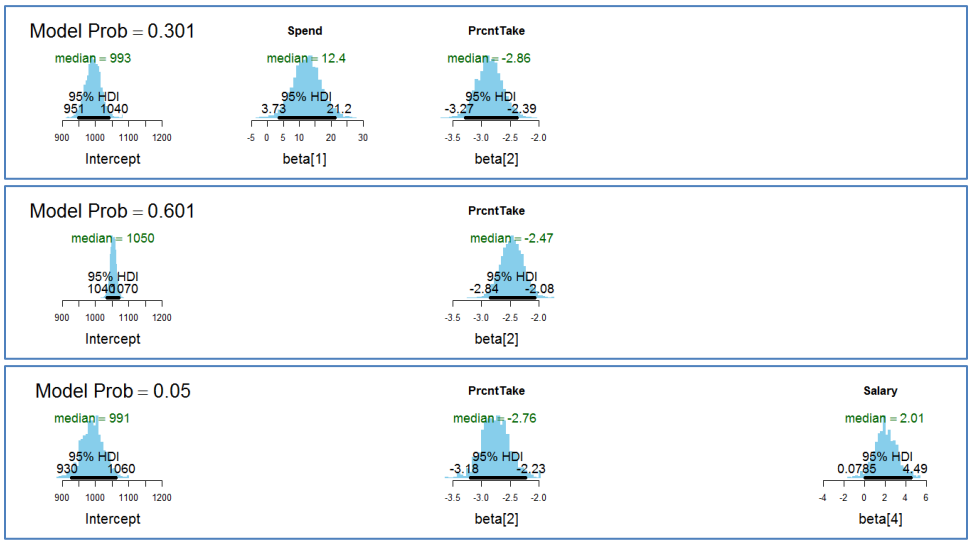
To understand why this happens, re-read section 18.4.1 of the book. Setting sigmaBeta to 10 is just like part of what's happening in the version at the bottom of p. 540, in which SD is set to 10. Because the prior on beta is so diffuse and diluted, the prior probability of any particular value of beta is relatively small, so there is a cost (expressed in the prior) of including beta in the model.

(C) Set the prior on `sigmaBeta` back to the gamma distribution of the first part. Now change the prior on the inclusion indices so that

```
delta[j] ~ dbern( 0.2 )
```

Run the high-level script. In what ways is the posterior different than the first part? Discuss model probabilities, inclusion probabilities, and HDIs of regression coefficients.

Results are shown here:



The probabilities of the models are different than part (A), because now the priors prefer models with fewer included predictors. In particular, the model with only PrcntTake is now preferred over the model that includes both PrcntTake and Spend. Within each model, however, the estimated regression coefficients are essentially the same (mode and HDI) as when using the 0.5 prior on delta[j].

Thus, even though we have re-set the prior on sigmaBeta so it does not overly dilute the prior on regression coefficients, the prior on the inclusion indicators explicitly prefers models with fewer included predictors, and a similar effect is produced on the models.

Solutions for exercises in Chapters 19 – 25 will be supplied in a future update of this manual.