BDA A2

Piero Birello

March 2025

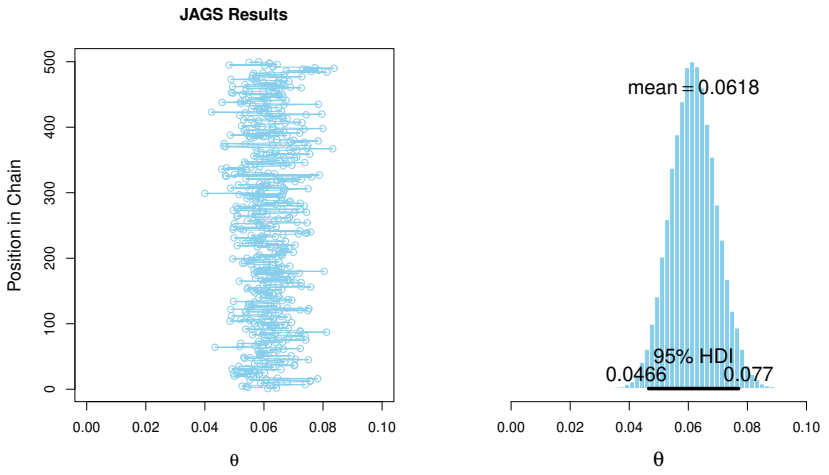
Exercise A

Task 1

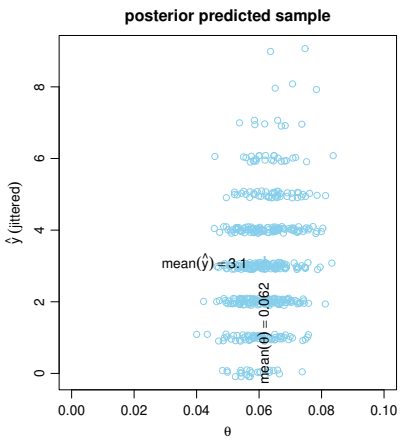
The modified code is as follows:

```
rm(list = ls())
graphics.off()
source("openGraphSaveGraph.R")
save_path = "Figures/"

require(rjags)          # Kruschke, J. K. (2011). Doing Bayesian Data Analysis:
# A Tutorial with R and BUGS. Academic Press / Elsevier.
#------------------------------------------------------------------------------
# THE MODEL.

# Specify the model in JAGS language, but save it as a string in R:
modelString = "
# JAGS model specification begins ...
model {
# Likelihood:
for ( i in 1:nStores ) {
y[i] ~ dbin( theta,50 )
}
# Prior distribution:
theta ~ dnorm( 0.05,1/(0.01)^2 )T( 0.0,0.1 )
# theta ~ dnorm( 0.1,1/(0.02)^2 )T( 0.05,0.15 )
}
# ... JAGS model specification ends.
" # close quote to end modelString

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

#------------------------------------------------------------------------------
# THE DATA.

# Specify the data in R, using a list format compatible with JAGS:
dataList = list(
  nStores = 5 ,
  y = c( 7,5,4,4,2 )
)

#------------------------------------------------------------------------------
# INTIALIZE THE CHAIN.

# Can be done automatically in jags.model() by commenting out inits argument.
```

```
# Otherwise could be established as:
# initsList = list( theta = sum(dataList$y)/length(dataList$y) )

#------------------------------------------------------------------------
# RUN THE CHAINS.

parameters = c( "theta" )     # The parameter(s) to be monitored.
adaptSteps = 500              # Number of steps to "tune" the samplers.
burnInSteps = 1000            # Number of steps to "burn-in" the samplers.
nChains = 3                   # Number of chains to run.
numSavedSteps=50000           # Total number of steps in chains to save.
thinSteps=1                   # Number of steps to "thin" (1=keep every step).
nIter = ceiling( ( numSavedSteps * thinSteps ) / nChains ) # Steps per chain.
# Create, initialize, and adapt the model:
jagsModel = jags.model( "model.txt" , data=dataList , # inits=initsList ,
                        n.chains=nChains , n.adapt=adaptSteps )
# Burn-in:
cat( "Burning in the MCMC chain...\n" )
update( jagsModel , n.iter=burnInSteps )
# The saved MCMC chain:
cat( "Sampling final MCMC chain...\n" )
codaSamples = coda.samples( jagsModel , variable.names=parameters ,
                            n.iter=nIter , thin=thinSteps )
# resulting codaSamples object has these indices:
#   codaSamples[[ chainIdx ]][ stepIdx , paramIdx ]


#------------------------------------------------------------------------
# EXAMINE THE RESULTS.

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

thetaSample = mcmcChain

# Make a graph using R commands:
trunc_low = 0.0
trunc_high = 0.1
openGraph(width=10,height=6)
layout( matrix( c(1,2) , nrow=1 ) )
plot( thetaSample[1:500] , 1:length(thetaSample[1:500]) , type="o" ,
      xlim=c(trunc_low,trunc_high) , xlab=bquote(theta) , ylab="Position in Chain" ,
      cex.lab=1.25 , main="JAGS Results" , col="skyblue" )
source("plotPost.R")
histInfo = plotPost( thetaSample , xlim=c(trunc_low,trunc_high) , xlab=bquote(theta) )
saveGraph( file=paste0(save_path,"BernBetaJagsFullPost") , type="eps" )

# Posterior prediction:
# For each step in the chain, use posterior theta to flip a coin:
chainLength = length( thetaSample )
yPred = rep( NULL , chainLength )  # define placeholder for flip results
for ( stepIdx in 1:chainLength ) {
  pHead = thetaSample[stepIdx]
  #yPred[stepIdx] = sample( x=c(0,1), prob=c(1-pHead,pHead), size=1 )
  yPred[stepIdx] = rbinom( n=1,size=50,prob=pHead )
}
# Jitter the y values for plotting purposes:
yPredJittered = yPred + runif( length(yPred) , -.1 , +.1 )
# Now plot the jittered values:
openGraph(width=5,height=5.5)
```



(a) Sampled θ along MCMC chain, and corresponding histogram (sampled posterior distribution).



(b) Posterior predicted sample.

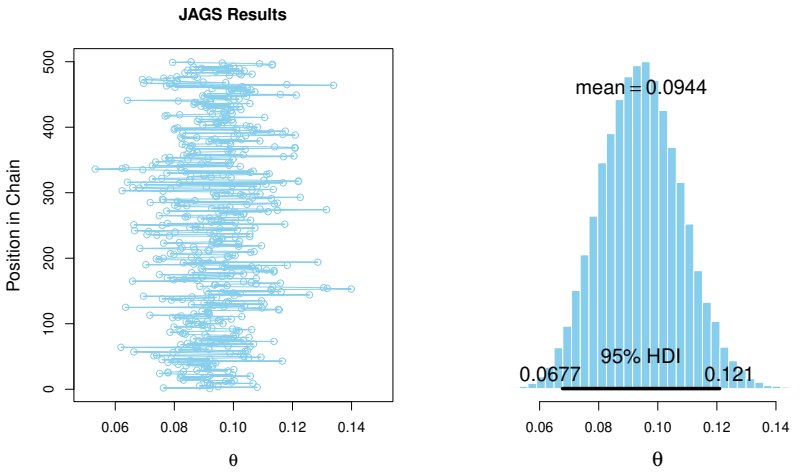
```
par( mar=c(3.5,3.5,2.5,1) , mgp=c(2,0.7,0) )
plot( thetaSample[1:500] , yPredJittered[1:500] , xlim=c(trunc_low,trunc_high) ,
      main="posterior predicted sample" ,
      xlab=expression(theta) , ylab=expression(hat(y)*" "*(jittered)) ,
      col="skyblue" )
points( mean(thetaSample) , mean(yPred) , pch="+" , cex=2 , col="skyblue" )
text( mean(thetaSample) , mean(yPred) ,
      bquote( mean(hat(y)) == .(signif(mean(yPred),2)) ) ,
      adj=c(1.2,.5) )
text( mean(thetaSample) , mean(yPred) , srt=90 ,
      bquote( mean(theta) == .(signif(mean(thetaSample),2)) ) ,
      adj=c(1.2,.5) )
#abline( trunc_low , trunc_high , lty="dashed" , lwd=2 )
saveGraph( file=paste0(save_path,"BernBetaJagsFullPred") , type="eps" )
```

Obtained results are shown in Figure 1a,Figure 1b. Based on the histogram in Figure 1a, we may say that, qualitatively, we suspend belief about the statement that the panels are produced by company A. The posterior distribution for θ (the probability of failure) qualitatively matches a truncated normal distribution with mean 0.05 and standard deviation 0.01, and the expected value of the mean of θ is contained within the HDI. However, the HDI would not be contained in a ROPE around the expected value of θ.

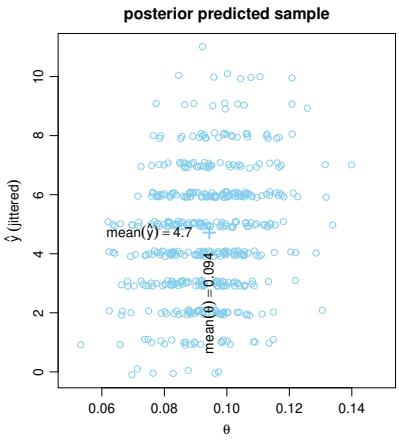
In fact, a more precise assessment would require to evaluate the posterior distributions for the parameters of the *dnorm* themselves, and check whether the corresponding HDI intervals contain a ROPE defined around the declared values. Otherwise, we could adopt a model comparison approach and compare the results under the current model with those under an alternative one.

Task 2

We now run the model under a different prior, namely $\theta \sim dnorm(0.1, 1/(0.02)^2)T(0.05, 0.15)$. Results are shown in Figure 2a, Figure 2b. Again, we may say that the obtained posterior distribution matches qualitatively the prior. Qualitatively, we suspect that this model is more accurate than the previous one. Figure 2b also shows a posterior predicted sample that is more similar to the one at disposal.



(a) Sampled θ along MCMC chain, and corresponding histogram (sampled posterior distribution).



(b) Posterior predicted sample.

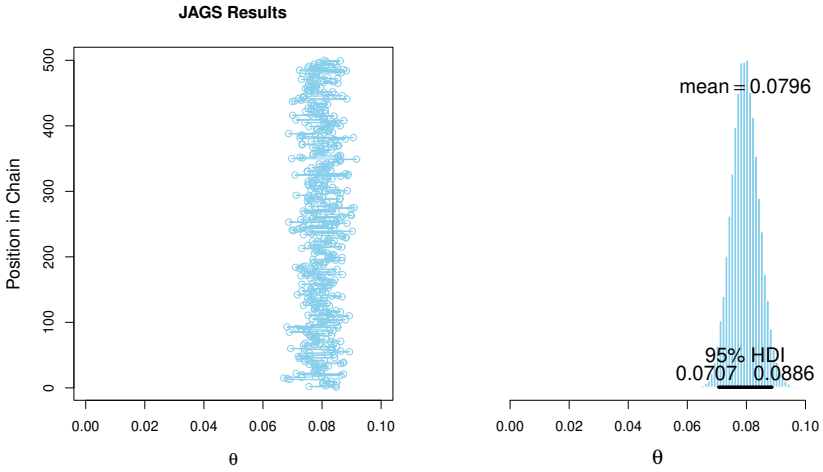
Task 3

Considering a ten times larger sample reporting the same fractions of failures, we have more evidence in support of factory B than factory A, see Figure 3a, Figure 3b, Figure 4a, Figure 4b. The HDI for θ given prior A does not contain the expected value for θ, while the HDI given prior B does -it ends right there. Still, a more accurate analysis would require to evaluate the mean and standard deviation parameters of the *dnorm*, or perform model comparison.

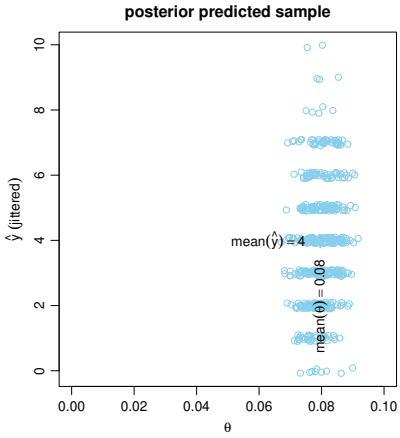
Task 4

We add the following piece of code:

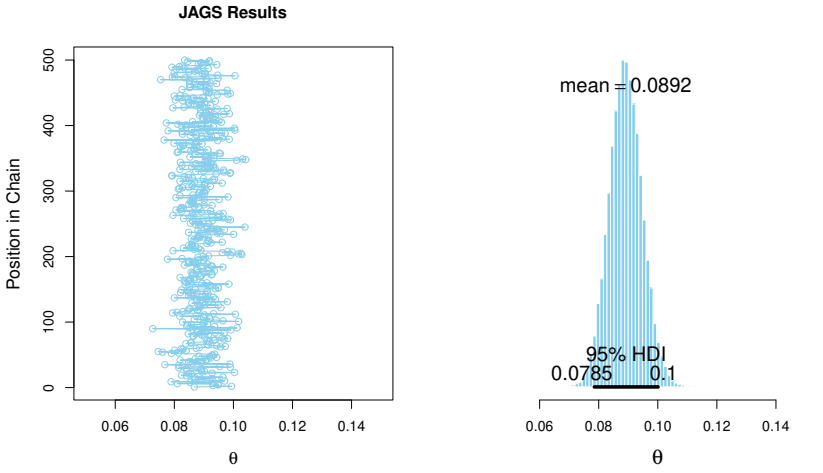
```
trunc_low = 0.05
trunc_high = 0.15
dataN = 50*10
mu = 0.1
sigma = 0.02

# COMPUTE PROBABILITY OF DATA

# Define truncated normal
truncNorm = function(x,mean,sd,a,b) {
  pdfValues = dnorm(x, mean = mean, sd = sd) /
  (pnorm(b, mean = mean, sd = sd) - pnorm(a, mean = mean, sd = sd))
  return(pdfValues)
  }

# Compute mean and standard deviation of MCMC values:
meanTheta = mean(thetaSample)
sdTheta = sd(thetaSample)

# Compute 1/p(D):
logOneOverPD = mean( log( truncNorm( thetaSample,meanTheta,sdTheta,trunc_low,trunc_high ) ) -
    # h(theta_i)
```



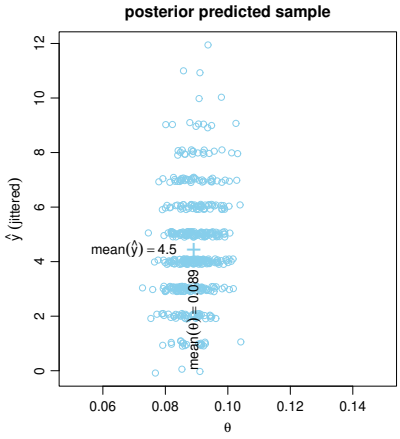
(a) Sampled θ along MCMC chain, and corresponding histogram (sampled posterior distribution).



(b) Posterior predicted sample.



(a) Sampled θ along MCMC chain, and corresponding histogram (sampled posterior distribution).



(b) Posterior predicted sample.

```
                    ( rowSums( dbinom( dataList$y,dataN,thetaSample, log=TRUE ) ) +          #
                       p(D|theta_i)
                  log( truncNorm( thetaSample,mu,sigma,trunc_low,trunc_high ) ) ) )      # p(
                     theta_i)
logPD = -(logOneOverPD)
show(logPD)
```

We run the script under the two prior distributions for θ, then compute the log Bayes Factor as:

$$log\, BF = log\, p(D|m = A) - log\, p(D|m = B) = -11.20936 - (-7.179099) = -4.030261 \quad (1)$$

Hence, as expected, model comparison lets us select the hypothesis that panels are produced by company B.

Exercise B

The model can be written as:

$$y_j \sim Binomial(p_c^{machines_j}) \quad (2)$$
$$p_c^i \sim trunc.Normal(\mu, \sigma, a, b) \quad (3)$$
$$\mu \sim Beta(1.1, 9.9) \quad (4)$$
$$\sigma \sim Gamma(2, 28) \quad (5)$$
$$a = 0.0 \quad (6)$$
$$b = 0.5 \quad (7)$$

It can be implemented in JAGS as:

```
# THE MODEL.

# Specify the model in JAGS language, but save it as a string in R:
modelString = "
# JAGS model specification
model {
# Likelyhood
for (j in 1:nItemsTotal) {
y[j] ~ dbin( theta[machines[j]], 40 )
}
# Prior
for (i in 1:nMachines){
theta[i] ~ dnorm( normMu, 1/normSigma^2 )T( 0.0,0.5 )
}
# Hyperpriors
normMu ~ dbeta( 1.1, 9.9 )
normSigma ~ dgamma( 2, 28 )
}
# ... JAGS model specification ends.
"
# close quote to end modelString

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


# THE DATA.


dataList = list(
```

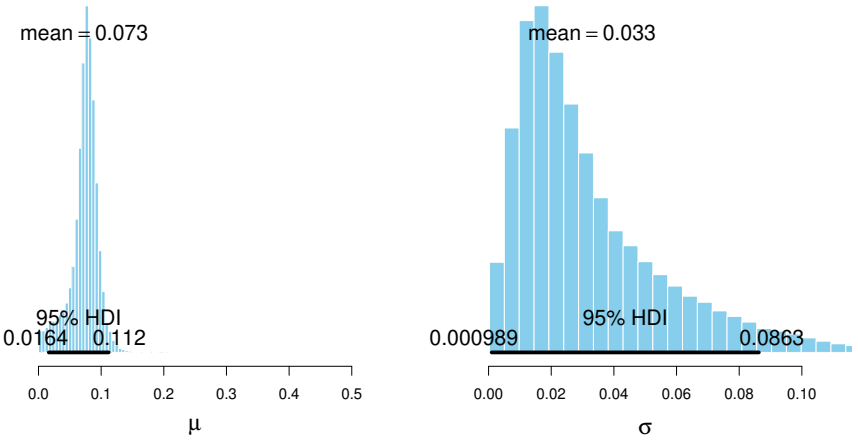


Figure 5: Sampled posterior distributions for hyperparameters of the truncated normal distribution.

```
y=c(1, 5, 5, 5, 2, 4, 2, 1, 3, 3, 3, 5, 3, 4, 6, 4, 2, 4, 5, 1, 1, 3, 2, 4, 1),
machines = rep(c(1,2,3,4,5),each=5),
nMachines = 5,
#nItems=rep(40, 25),
nItemsTotal = 25
)
```

The sampled posterior distributions for hyperparameters are reported in Figure 5.

Exercise C

The problem can be formulated as follows. The number of successes under the three treatments are $z = (8, 11, 14)$ with $N = (30, 30, 30)$ trials. The model can be written as:

$$z_c \sim Binomial(\theta_c, N_c) \tag{8}$$
$$\theta_c \sim Beta(\omega(\kappa - 2) + 1, (1 - \omega)(\kappa - 2) + 1) \tag{9}$$
$$\omega \sim Beta(\omega_0(\kappa_0 - 2) + 1, (1 - \omega_0)(\kappa_0 - 2) + 1) \tag{10}$$
$$\kappa \sim Gamma(0.01, 0.01) \tag{11}$$
$$\omega_0 = 0.3 \tag{12}$$
$$\kappa_0 = 10 \tag{13}$$

Values for hyperpriors of ω where set based on previous knowledge on recovery rate under no treatment. Values for hyperpriors of κ are set such that κ is vague.

Here is the code we implemented:

```
rm(list = ls())
graphics.off()
source("openGraphSaveGraph.R")
require(rjags)
```

```
savePath = "Figures/ExerciseC/"

#------------------------------------------------------------------------------
# THE MODEL.
modelString = "
  model {
    for ( cIdx in 1:Ncat ) {
      z[cIdx] ~ dbin( theta[cIdx] , N[cIdx] )
      theta[cIdx] ~ dbeta( omega*(kappa-2)+1 ,
                           (1-omega)*(kappa-2)+1 )
    }
    omega ~ dbeta( omega0*(kappa0-2)+1, omega0*(kappa0-2)+1 )
    kappa <- kappaMinusTwo + 2
    kappaMinusTwo ~ dgamma( 0.01 , 0.01 )  # mean=1 , sd=10 (generic vague)
    omega0 <- 0.3
    kappa0 <- 10
  }
  " # close quote for modelString
writeLines( modelString , con="Cmodel.txt" )

#------------------------------------------------------------------------------
# THE DATA.


z = c(8,11,14)
N = c(30,30,30)
Ncat =  length(unique(z))
# Specify the data in a list, for later shipment to JAGS:
dataList = list(
  z = z ,
  N = N ,
  Ncat = Ncat
)

#------------------------------------------------------------------------------
# INTIALIZE THE CHAIN.

# Initial values of MCMC chains based on data:
initsList = function() {
  thetaInit = rep(NA,Ncat)
  for ( cIdx in 1:Ncat ) { # for each subject
    resampledZ = rbinom(1, size=N[cIdx] , prob=z[cIdx]/N[cIdx] )
    thetaInit[cIdx] = resampledZ/N[cIdx]
  }
  thetaInit = 0.001+0.998*thetaInit # keep away from 0,1
  kappaInit = 100 # lazy, start high and let burn-in find better value
  return( list( theta=thetaInit ,
                omega0=mean(thetaInit) ,
                kappaMinusTwo0=kappaInit-2 ) )
}

#------------------------------------------------------------------------------
# RUN THE CHAINS.

parameters = c( "theta" )   # The parameter(s) to be monitored.
adaptSteps = 1000              # Number of steps to "tune" the samplers.
burnInSteps = 5000            # Number of steps to "burn-in" the samplers.
nChains = 3                   # Number of chains to run.
numSavedSteps=50000           # Total number of steps in chains to save.
thinSteps=1                   # Number of steps to "thin" (1=keep every step).
nIter = ceiling( ( numSavedSteps * thinSteps ) / nChains ) # Steps per chain.
# Create, initialize, and adapt the model:
```

```
jagsModel = jags.model( "Cmodel.txt" , data=dataList , inits=initsList ,
                        n.chains=nChains , n.adapt=adaptSteps )
# Burn-in:
cat( "Burning in the MCMC chain...\n" )
update( jagsModel , n.iter=burnInSteps )
# The saved MCMC chain:
cat( "Sampling final MCMC chain...\n" )
codaSamples = coda.samples( jagsModel , variable.names=parameters ,
                            n.iter=nIter , thin=thinSteps )


#-------------------------------------------------------------------------------
# EXAMINE THE RESULTS.
mcmcChain = as.matrix( codaSamples )

# Extract the posterior samples
theta1Sample = mcmcChain[, "theta[1]"]
theta2Sample = mcmcChain[, "theta[2]"]
theta3Sample = mcmcChain[, "theta[3]"]

# Compute differences:
thetaDiff12 = theta1Sample - theta2Sample
thetaDiff13 = theta1Sample - theta3Sample
thetaDiff23 = theta2Sample - theta3Sample

# Load plotting function
source("plotPost.R")

# Open a plotting window
openGraph(width=12, height=8)
par(mfrow=c(2,3)) # 2 rows, 3 columns for all 6 plots

# Plot posteriors for theta1, theta2, theta3
plotPost(theta1Sample, xlab=expression(theta[1]), main="Posterior of theta1")
plotPost(theta2Sample, xlab=expression(theta[2]), main="Posterior of theta2")
plotPost(theta3Sample, xlab=expression(theta[3]), main="Posterior of theta3")

# Plot posteriors for differences
plotPost(thetaDiff12, xlab=expression(theta[1] - theta[2]), main="theta1 - theta2",
         compVal=0.0,ROPE=c(-0.05,0.05))
plotPost(thetaDiff13, xlab=expression(theta[1] - theta[3]), main="theta1 - theta3",
         compVal=0.0,ROPE=c(-0.05,0.05))
plotPost(thetaDiff23, xlab=expression(theta[2] - theta[3]), main="theta2 - theta3",
         compVal=0.0,ROPE=c(-0.05,0.05))

# Save the graph
saveGraph(file = paste0(savePath, "ThetaPosteriors"), type="eps")
```

Results are shown in Figure 6. The three posteriors exhibit different means. In particular, θ_3 has higher mean that θ_2 and θ_2 has higher mean than θ_1. However, there is not sufficient evidence to defend the claim that any of the recovery rates under placebo is higher than the one under no treatment, nor that the expensive placebo is more effective than the cheap one. Actually, the ROPE $[-0.05, 0.05]$ is always included in the HDIs of θ differences.

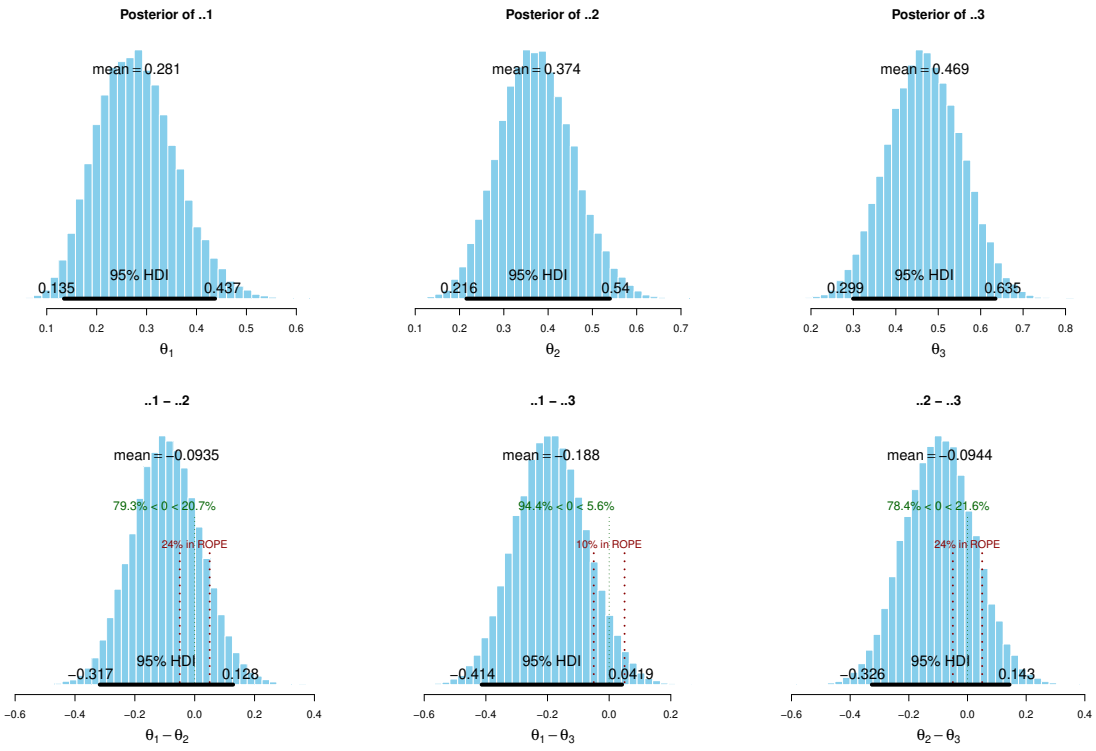


Figure 6: Posteriors and posterior differences for recovery rate θ under the three different treatments of exercise C.