BDA A3

Piero Birello

April 2025

Exercise A

Part 1

Let y_i be the number of tulips grown in row i (equivalently, from bag i). Let n_i be the number of bulbs contained in bag i and b_i the bag (bulb) type. The model can be written as:

$$y_i \sim Binomial(n_i, p_{b_i}) \tag{1}$$
$$n_i = round(\hat{n}_i) \tag{2}$$
$$\hat{n}_i \sim Normal(\mu = 30, \sigma = 1.5) \tag{3}$$
$$p_{b_i=1} \sim Beta(40, 10) \tag{4}$$
$$p_{b_i=2} \sim Beta(1, 1) \tag{5}$$
$$b_i \sim Bernoulli(1 - \theta) + 1 \tag{6}$$
$$\theta \sim Beta(1, 1) \tag{7}$$

In Jags, this translates to:

```
modelString = "
# JAGS model specification
model {
# Likelyhood
for (i in 1:nBags) {
y[i] ~ dbin( p[b[i]], n[i] )
n[i] <- round(nContinuous[i])
nContinuous[i] ~ dnorm(normMu,normTau)
b[i] <- bBernoulli[i] + 1
bBernoulli[i] ~ dbern(1-catTheta) # equivalent to dcat
}
# Priors
p[1] ~ dbeta(betaA,betaB)  # when bBernoulli=0
p[2] ~ dbeta(1,1)          # when bBernoulli=1
catTheta ~ dbeta(1,1)
# Constants
normMu <- 30
normTau <- 1/(normSigma*normSigma) #Convert to precision
normSigma <- 1.5
betaA <- 40
betaB <- 10
}
# ... JAGS model specification ends.
" # close quote to end modelString
```

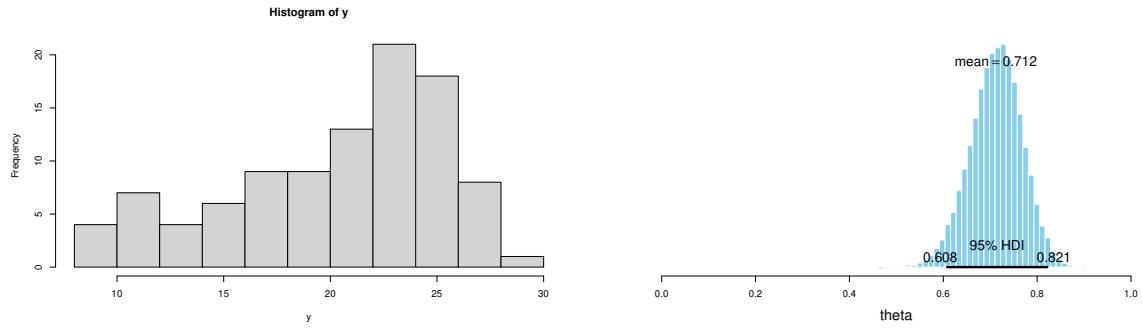

Figure 1: Histogram of y and posterior estimate of parameter θ.

A histogram for the occurrence of the growth of a number of tulips y is shown in Figure 1. Parameter estimates are shown in Figure 1 and Figure 2. Results indicate that around 70% of the bags are of type A (mean=0.712, 95% HDI=$[0.608, 0.821]$). Type A bulbs actually have a fertility $p_{b_i=1}$ close to 0.8 (mean=0.785, 95% HDI= $[0.761, 0.809]$), while type B bulbs have lower fertility (mean=0.468, 95% HDI= $[0.417, 0.514]$). We may exclude the hypothesis that the two bulb types have the same fertility, since a ROPE centered in 0 of half-length 0.1 contains no samples of the difference $p_{b_i=1} - p_{b_i=2}$. If we assume prior knowledge that the value of θ is close to 0.5, i.e., $\theta \sim Beta(15, 15)$, we get the results reported in Figure 3 and Figure 4. While estimates for $p_{b=1}$ and $p_{b=2}$ are similar to the previous ones, posterior estimates for θ are pulled towards 0.5, resulting in mean=0.652, 95% HDI= $[0.556, 0.743]$ (previous was mean=0.712, 95% HDI=$[0.608, 0.821]$). We can perform a proper model comparison to select the most plausible prior distribution. We can implement this in JAGS by introducing a dummy variable for the model index, distributed according to a categorical distribution with prior probability of 0.5 on each of the indices. This reads:

```
# JAGS MODEL

# Specify the model in JAGS language, save it as a string in R:
modelString = "
# JAGS model specification
model {
# Likelyhood
for (i in 1:nBags) {
y[i] ~ dbin( p[b[i]], n[i] )
n[i] <- round(nContinuous[i])
nContinuous[i] ~ dnorm(normMu,normTau)
b[i] <- bBernoulli[i] + 1
bBernoulli[i] ~ dbern(1-catTheta) # equivalent to dcat(pCat[]) // pCat[1] = catTheta
}
# Priors
p[1] ~ dbeta(p1BetaA,p1BetaB)  # when b=0
p[2] ~ dbeta(1,1)          # when b=1
catTheta ~ dbeta(thetaBetaA[m],thetaBetaB[m])
# Constants
normMu <- 30
normTau <- 1/(normSigma*normSigma) #Convert to precision
normSigma <- 1.5
p1BetaA <- 40
p1BetaB <- 10
thetaBetaA[1] <- 1
thetaBetaB[1] <- 1
thetaBetaA[2] <- 15
thetaBetaB[2] <- 15
```

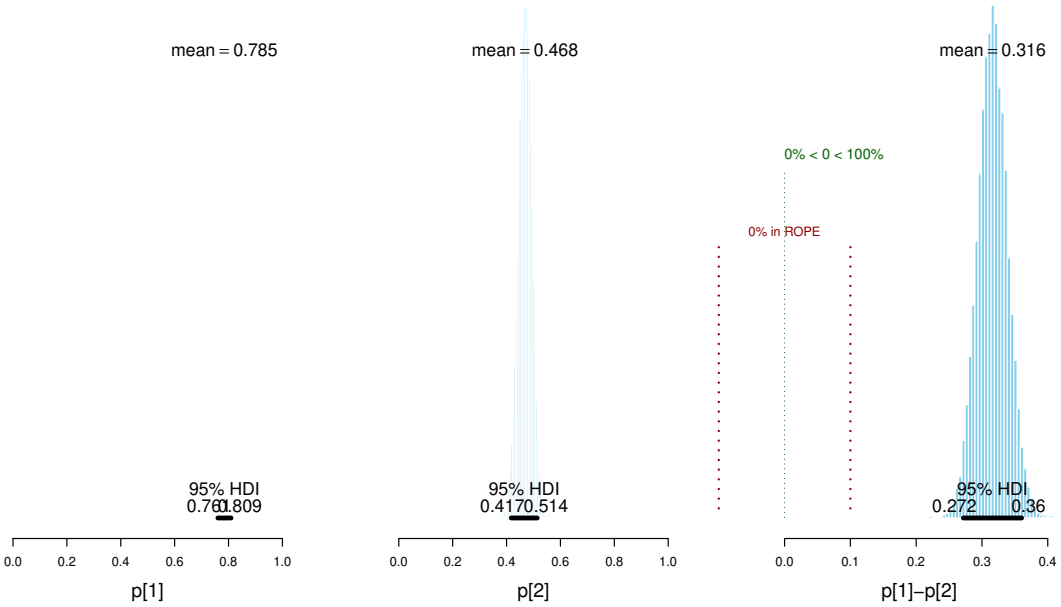

Figure 2: Posterior estimates for the fertility of the two bulb types $p_{b=1}$ and $p_{b=2}$.

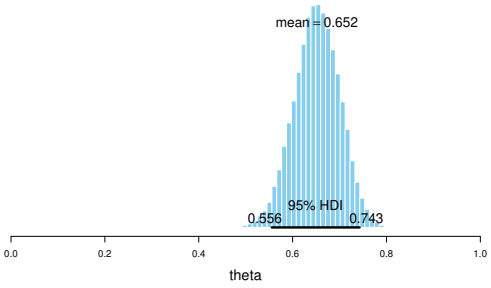

Figure 3: Posterior estimate for θ when the prior is $Beta(15, 15)$.

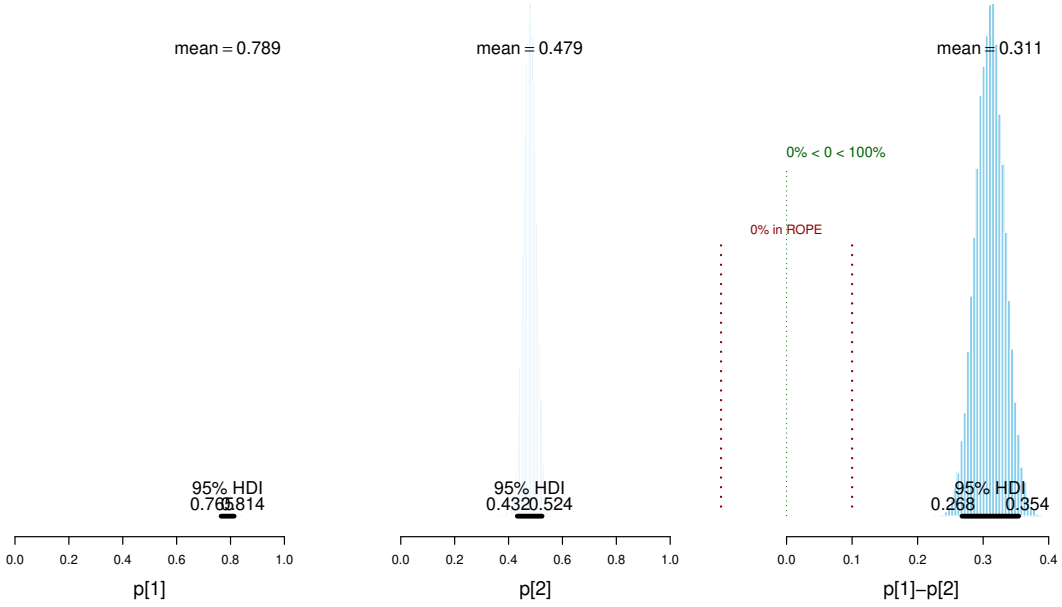

Figure 4: Posterior estimates for $p_{b=1}$ and $p_{b=2}$ when the prior for θ is $Beta(15, 15)$.

```
# Model Dummy Variable
m ~ dcat(mPriorProb)
mPriorProb[1] <- 0.5
mPriorProb[2] <- 0.5
}
# ... JAGS model specification ends.
" # close quote to end modelString

# Write the modelString to a file, using R commands:
writeLines(modelString,con="modelExercise1c.txt")

#------------------------------------------------------------------------------
# INTIALIZE THE CHAIN
initsList = list( p=c(0.8,0.8), catTheta=0.5 , m=1 )

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

# Assign parameters
parameters = c( "catTheta","p[1]","p[2]","m" )   # 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( "modelExercise1c.txt" , data=dataList , inits=initsList ,
                        n.chains=nChains , n.adapt=adaptSteps )
```

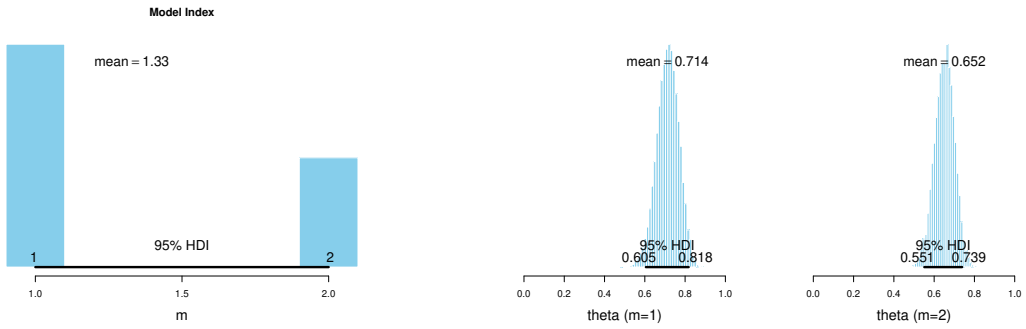

Figure 5: Posterior estimates for the model index m and parameter θ under the two models.

```
# 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 sample from JAGS:
catThetaSample = mcmcChain[,"catTheta"]
p1Sample = mcmcChain[,"p[1]"]
p2Sample = mcmcChain[,"p[2]"]
pDiffSample = p1Sample-p2Sample
mSample = mcmcChain[,"m"]

# Compute the proportion of m at each index value:
pM1 = sum( mSample == 1 ) / length( mSample )
pM2 = 1 - pM1
print(paste0('model 1 prob:', pM1))

# Extract param values for each model index:
catThetaSampleM1 = catThetaSample[ mSample == 1 ]
catThetaSampleM2 = catThetaSample[ mSample == 2 ]
p1SampleM1 = p1Sample[ mSample == 1 ]
p1SampleM2 = p1Sample[ mSample == 2 ]
p2SampleM1 = p2Sample[ mSample == 1 ]
p2SampleM2 = p2Sample[ mSample == 2 ]
pDiffSampleM1 = p1SampleM1-p2SampleM1
pDiffSampleM2 = p1SampleM2-p2SampleM2
```

Results for posterior estimates of the model index m, parameter θ and the fertilities $p_{b=1}$ and $p_{b=2}$ under the two models are shown in Figure 5, Figure 6 and Figure 7. Again, we see that the two models result in equivalent estimates for the fertilities. However, model 1 estimates θ to be larger than model 2, whose prior pulls the estimate towards the value $\theta = 0.5$. Model 1 is also the most likely. Precisely, the posterior probability for model 1 is 0.6708. Equivalently, posterior odds are $p(m = 1|D)/p(m = 2|D) = 2.0377$, i.e., model 1 is two times as likely as model 2.

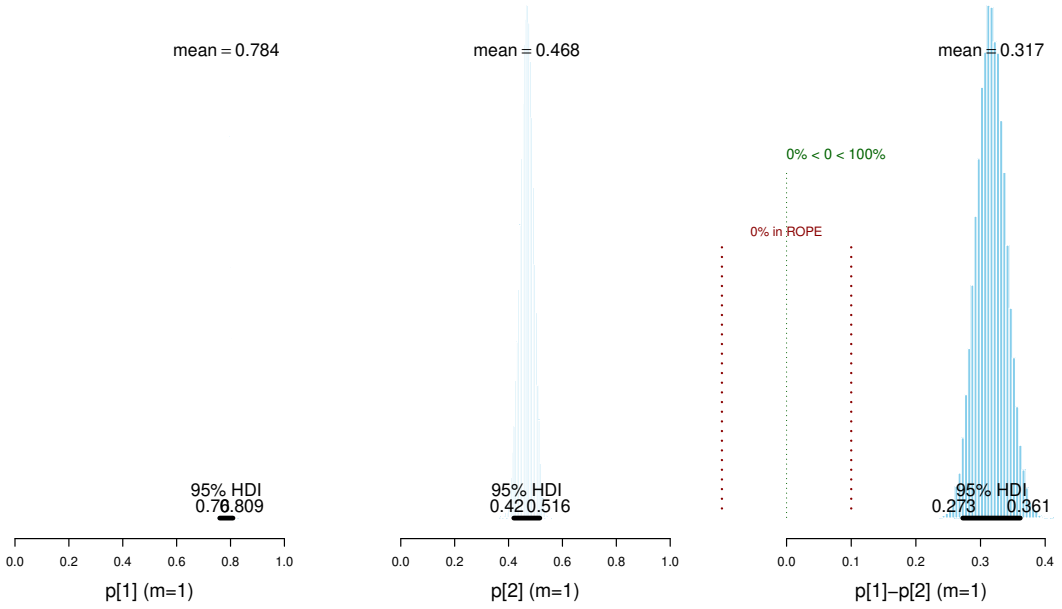

Figure 6: Posterior estimates for $p_{b=1}$ and $p_{b=2}$ under model 1.

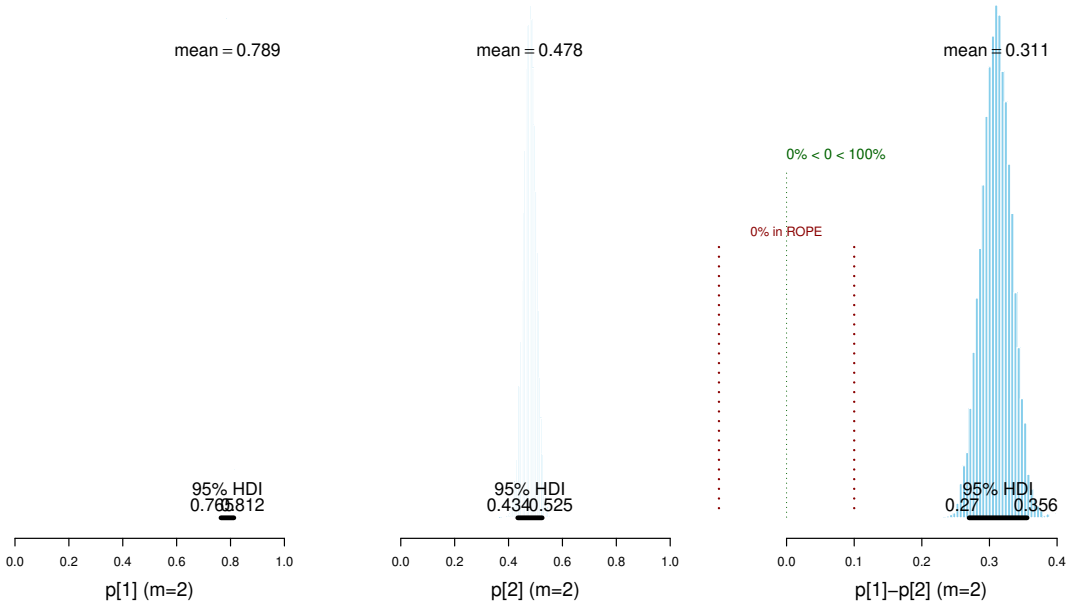

Figure 7: Posterior estimates for $p_{b=1}$ and $p_{b=2}$ under model 2.

Part 2

We now perform a power analysis to estimate the number of bags we will need to test in order for HDI's of the two fertility parameters not to overlap, given hypothesized hyper-parameters $p_{b_i=1} = 0.8, p_{b_i=2} = 0.5, \theta = 0.3$. We generate representative samples of n_i and b_i given the hypothesized hyper-parameters. From the representative parameters, we generate a random sample of the observable data y_i. We run Bayesian inference according to the model used so far (using $\theta \sim Beta(1, 1)$). We then check whether the goal of non overlapping HDI between $p_{b_i=1}$ and $p_{b_i=2}$ is achieved. We repeat for various sample sizes, i.e., number of bags, starting from 4 and increasing until the power crosses 0.8. For each sample size, we repeat the sampling procedure 10 times. The developed code is the following:

```
# Jags-Ybin-Nnorm-Bbern-Pbeta-Power.R
graphics.off() # This closes all of R's graphics windows.
rm(list=ls())  # Careful! This clears all of R's memory!
fileNameRoot = "Jags-Ydich-Xnom1subj-MbernBeta-Power-" # for future use

# Load the functions genMCMC, smryMCMC, and plotMCMC:
# (This also sources DBDA2E-utilities.R)
source("Jags-Ybin-Nnorm-Bbern-Pbeta.R")

#------------------------------------------------------------------------------
# Define function that assesses goal achievement for a single set of data:
goalAchievedForSample = function( data ) {
  # Generate the MCMC chain:
  mcmcCoda = genMCMC( data=data , numSavedSteps=10000 , saveName=NULL )
  # Check goal achievement. First, compute the HDI:
  p1HDI = HDIofMCMC( as.matrix(mcmcCoda[,"p[1]"]) )
  p2HDI = HDIofMCMC( as.matrix(mcmcCoda[,"p[2]"]) )
  # Define list for recording results:
  goalAchieved = list()
  # Goal: No overlap between HDIs
  goalAchieved = c(goalAchieved,
                   "noOverlap"=(( p1HDI[2] < p2HDI[1]) || (p2HDI[2] < p1HDI[1])) )
  # More goals can be inserted here if wanted...
  # Return list of goal results:
  return(goalAchieved)
}

#------------------------------------------------------------------------------
# Specify hypothetical parameters:
p = c(0.8,0.5)
catTheta=0.3
normMu = 30
normSigma = 1.5

# Define proportion success list
proportionSuccessList = c()

# Iterate over sample size (number of bags):
for (sampleN in 4:100) {
  # Run a bunch of simulated experiments:
  nSimulatedDataSets = 10
  # Track number of times goal is achieved
  goalCount=0

  for ( simIdx in 1:nSimulatedDataSets ) {
    # Generate random value from hypothesized parameter distribution:
    nContinuous = rnorm(n=sampleN,mean=normMu,sd=normSigma)
    n = round(nContinuous)
    bBernoulli = rbinom(n=sampleN,size=1,prob=1-catTheta)
```

```
    b = bBernoulli + 1
    # Generate random data based on parameter value:
    sampleY = sapply(1:length(n), function(i) {
      rbinom(n = 1, size = n[i], prob = p[b[i]])
    })
    # Do Bayesian analysis on simulated data:
    goalAchieved = goalAchievedForSample( sampleY )
    # 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="Jags-Ydich-Xnom1subj-MbernBeta-Power-goalTally.Rdata" )
    if (goalAchieved[["noOverlap"]] == TRUE) {
      goalCount = goalCount + 1
    }
  }

  # Calculate proportion of successes:
  proportionSuccess = goalCount / nSimulatedDataSets
  cat("SampleN =", sampleN, " | Success rate:", round(proportionSuccess * 100, 2), "%\n")
  proportionSuccessList[as.character(sampleN)] = proportionSuccess
  # Check stopping condition:
  if (proportionSuccess > 0.8) {
    print(proportionSuccessList)
    cat("Stopping early: goal achieved in over 80% of simulations for sample size", sampleN,
        "\n")
    break
  }

}
```

Note that the sourced script is the one developed for the previous part of the exercise, with the only difference that the relevant section has been wrapped in the function `genMCMC`. We obtain that a power of 0.8 or more can be obtained already by testing 7 bags. More precisely, obtained power estimates are 0.6, 0.7, 0.6, 0.9 for 4, 5, 6, 7 bags respectively. Note that a larger number of trials per each sample size would guarantee a more accurate estimate.

We repeat the experiment considering the case in which $p_{b_i=2} = 0.5$. Unfortunately here the sampler repeatedly fails to initialize when testing large sample sizes, despite efforts to improve initialization and catchment of errors. The code developed to -unsuccessfully- try solve the issue is reported in section .

Exercise B

We perform a linear regression of the form:

$$y_i \sim Normal(\mu_i, \sigma) \tag{8}$$
$$\mu_i = \beta_0 + \beta_1 x_i \tag{9}$$
$$\beta_0 \sim Normal(0, 10) \tag{10}$$
$$\beta_1 \sim Normal(0, 10) \tag{11}$$
$$\sigma = 1/\sqrt{\tau} \tag{12}$$
$$\tau \sim Gamma(0.01, 0.01) \tag{13}$$

where variables x and y have been standardized. Estimates for the coefficients are shown in Table 1. Posterior estimates are shown in Figure 8 insetad. The interesting parameter to observe in order to understand whether

Parameter	Mean	Median	Mode	HDI Low	HDI High
β_0	684.00	684.14	685.15	660.50	707.83
β_1	-1.55	-1.56	-1.58	-2.75	-0.34
σ	18.36	18.32	18.32	16.74	19.99

Table 1: Posterior summary statistics with 95% highest density intervals (HDI)

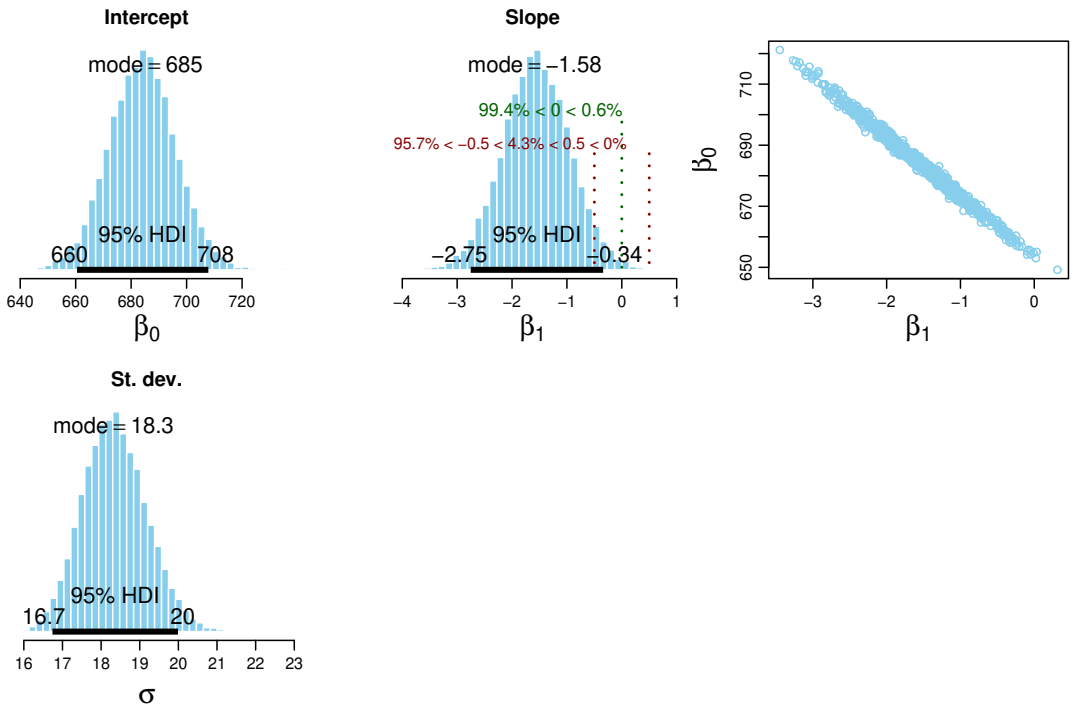

Figure 8: Posterior estimates of linear regression parameters.

the students to teacher ratio influences test scores is the slope β_1. Posterior estimates suggests that it is very

likely that higher student to teacher ratios correspond to lower test scores. The HDI does not stand completely out of the ROPE centered in 0 with half-length 0.5, however only the 4.255% of the HDI falls inside the ROPE, and only the 0.6% of sampled values is higher than zero. In Figure 9 we show a posterior predictive check of the linear regression model, with plausible regression lines and sampled noise distributions. The model seems to correctly fit the data, with few outliers with respect to the noise distribution. A fit using a t-distribution could improve robustness and include these outliers better. Hence, although model improvement would be recommended (also testing a hierarchical model including group variables), we may preliminary conclude that we would suggest investing in assuming more teachers. As an example, we show the posterior predicted distribution for test scores when the students to teacher ratio is 20 in Figure 10 The code used in this Exercise B is composed by a source script and an example script. The source script:

```
# Jags-Ymet-Xmet-Mrobust.R
source("DBDA2E-utilities.R")

#===============================================================================

genMCMC = function( data , xName="x" , yName="y" ,
                    numSavedSteps=50000 , saveName=NULL ) {
  require(rjags)
  #-------------------------------------------------------------------------------
  # THE DATA.
  y = data[,yName]
  x = data[,xName]
  # Do some checking that data make sense:
  if ( any( !is.finite(y) ) ) { stop("All y values must be finite.") }
  if ( any( !is.finite(x) ) ) { stop("All x values must be finite.") }
  #Ntotal = length(y)
  # Specify the data in a list, for later shipment to JAGS:
  dataList = list(
    x = x ,
    y = y
  )
  #-------------------------------------------------------------------------------
  # THE MODEL.
  modelString = "
  # 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
    }
  }
  # Specify the model for standardized data:
  model {
    for ( i in 1:Ntotal ) {
      zy[i] ~ dnorm( zmu[i] , ztau )
      zmu[i] <- zbeta0 + zbeta1 * zx[i]
    }
    # Priors vague on standardized scale:
    zbeta0 ~ dnorm( 0 , 1/(10)^2 )
    zbeta1 ~ dnorm( 0 , 1/(10)^2 )
    ztau ~ dgamma( 1.0E-2 , 1.0E-2 ) # corresponds to mean=1, var=100
    # Transform to original scale:
    beta1 <- zbeta1 * ysd / xsd
    beta0 <- zbeta0 * ysd  + ym - zbeta1 * xm * ysd / xsd
```

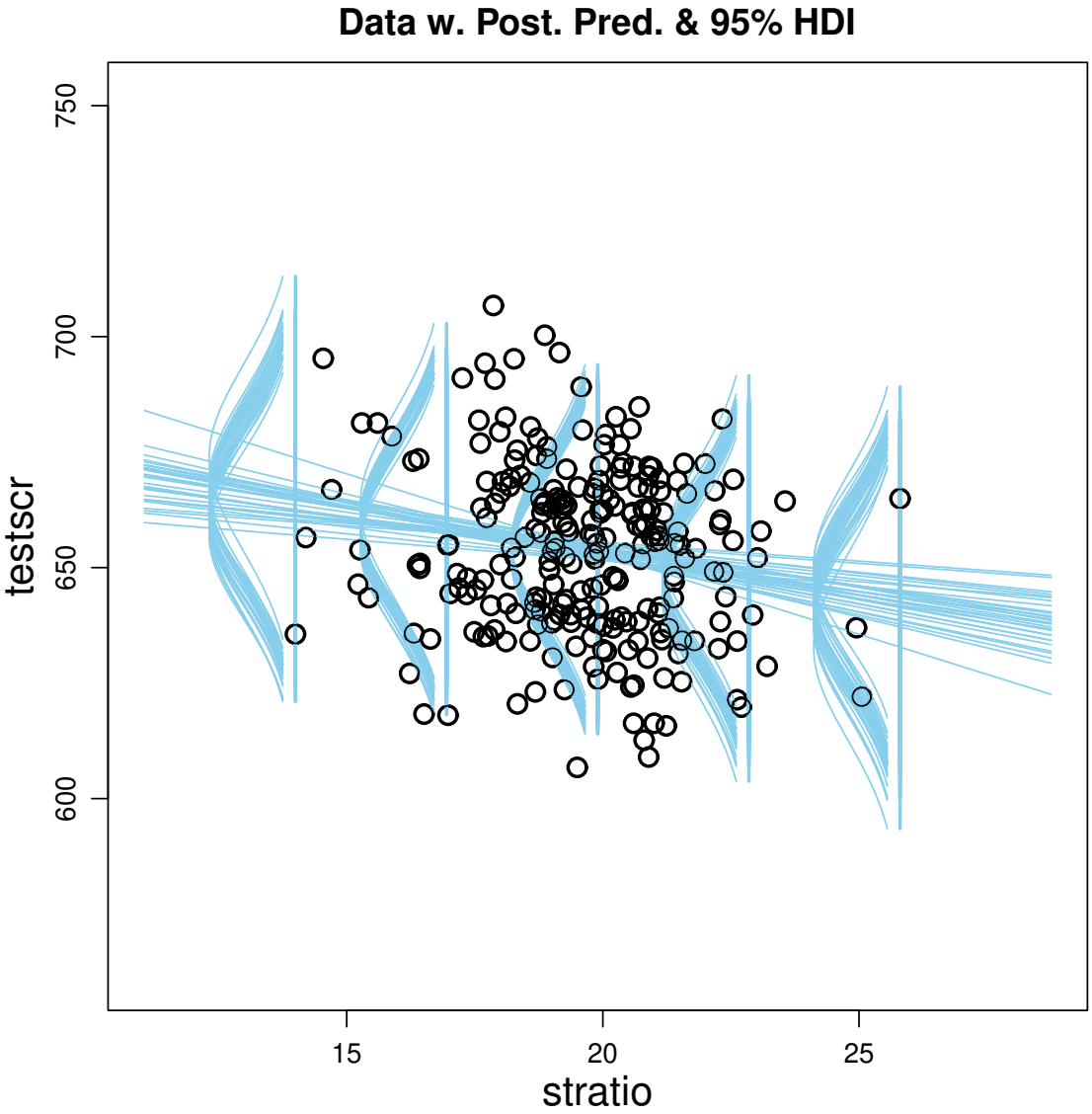

Figure 9: Posterior predictive check

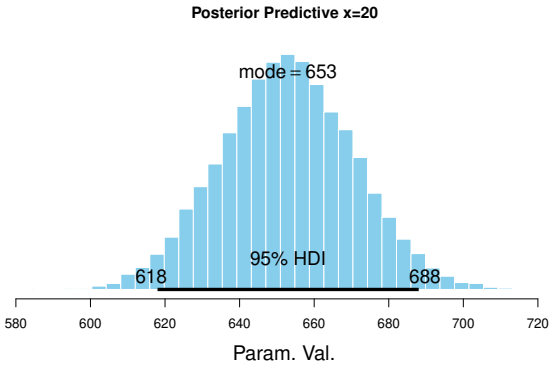

Figure 10: Posterior predicted distribution when x=20.

```
    tau <- ztau / (ysd^2)
    sigma <- 1/sqrt(tau)
  }
  " # close quote for modelString
  # Write out modelString to a text file
  writeLines( modelString , con="TEMPmodel.txt" )
  #------------------------------------------------------------------------------
  # INTIALIZE THE CHAINS.
  # Let JAGS do it...
  #------------------------------------------------------------------------------
  # RUN THE CHAINS
  parameters = c( "beta0" ,  "beta1" ,  "tau", "sigma",
                  "zbeta0" , "zbeta1" , "ztau" )
  adaptSteps = 500  # Number of steps to "tune" the samplers
  burnInSteps = 1000
  nChains = 4
  thinSteps = 1
  nIter = ceiling( ( numSavedSteps * thinSteps ) / nChains )
  # Create, initialize, and adapt the model:
  jagsModel = jags.model( "TEMPmodel.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 ]
  if ( !is.null(saveName) ) {
    save( codaSamples , file=paste(saveName,"Mcmc.Rdata",sep="") )
  }
  return( codaSamples )
} # end function

#===============================================================================

smryMCMC = function(  codaSamples ,
                      compValBeta0=NULL , ropeBeta0=NULL ,
                      compValBeta1=NULL , ropeBeta1=NULL ,
```

```
                      compValSigma=NULL , ropeSigma=NULL ,
                      saveName=NULL ) {
  summaryInfo = NULL
  mcmcMat = as.matrix(codaSamples,chains=TRUE)
  summaryInfo = rbind( summaryInfo ,
                       "beta0" = summarizePost( mcmcMat[,"beta0"] ,
                                                compVal=compValBeta0 ,
                                                ROPE=ropeBeta0 ) )
  summaryInfo = rbind( summaryInfo ,
                       "beta1" = summarizePost( mcmcMat[,"beta1"] ,
                                                compVal=compValBeta1 ,
                                                ROPE=ropeBeta1 ) )
  summaryInfo = rbind( summaryInfo ,
                       "sigma" = summarizePost( mcmcMat[,"sigma"] ,
                                                compVal=compValSigma ,
                                                ROPE=ropeSigma ) )

  if ( !is.null(saveName) ) {
    write.csv( summaryInfo , file=paste(saveName,"SummaryInfo.csv",sep="") )
  }
  return( summaryInfo )
}

#===============================================================================

plotMCMC = function( codaSamples , data , xName="x" , yName="y" ,
                     compValBeta0=NULL , ropeBeta0=NULL ,
                     compValBeta1=NULL , ropeBeta1=NULL ,
                     compValSigma=NULL , ropeSigma=NULL ,
                     showCurve=FALSE ,  pairsPlot=FALSE ,
                     saveName=NULL , saveType="jpg" ) {
  # showCurve is TRUE or FALSE and indicates whether the posterior should
  #   be displayed as a histogram (by default) or by an approximate curve.
  # pairsPlot is TRUE or FALSE and indicates whether scatterplots of pairs
  #   of parameters should be displayed.
  #-----------------------------------------------------------------------------
  y = data[,yName]
  x = data[,xName]
  mcmcMat = as.matrix(codaSamples,chains=TRUE)
  chainLength = NROW( mcmcMat )
  zbeta0 = mcmcMat[,"zbeta0"]
  zbeta1 = mcmcMat[,"zbeta1"]
  ztau = mcmcMat[,"ztau"]
  beta0 = mcmcMat[,"beta0"]
  beta1 = mcmcMat[,"beta1"]
  tau = mcmcMat[,"tau"]
  sigma = mcmcMat[,"sigma"]
  #-----------------------------------------------------------------------------
  if ( pairsPlot ) {
    # Plot the parameters pairwise, to see correlations:
    openGraph()
    nPtToPlot = 1000
    plotIdx = floor(seq(1,chainLength,by=chainLength/nPtToPlot))
    panel.cor = function(x, y, digits=2, prefix="", cex.cor, ...) {
      usr = par("usr"); on.exit(par(usr))
      par(usr = c(0, 1, 0, 1))
      r = (cor(x, y))
      txt = format(c(r, 0.123456789), digits=digits)[1]
      txt = paste(prefix, txt, sep="")
      if(missing(cex.cor)) cex.cor <- 0.8/strwidth(txt)
      text(0.5, 0.5, txt, cex=1.25 ) # was cex=cex.cor*r
      on.exit(par(usr=usr))
```

```
    }
    pairs( cbind( beta0 , beta1 , tau )[plotIdx,] ,
           labels=c( expression(beta[0]) , expression(beta[1]) ,
                     expression(tau) ) ,
           lower.panel=panel.cor , col="skyblue" )
    if ( !is.null(saveName) ) {
      saveGraph( file=paste(saveName,"PostPairs",sep=""), type=saveType)
    }
  }
  #-------------------------------------------------------------------------------
  # Marginal histograms:
  # Set up window and layout:
  nPtToPlot = 1000
  plotIdx = floor(seq(1,chainLength,by=chainLength/nPtToPlot))
  openGraph(width=8,height=5)
  layout( matrix( 1:6 , nrow=2, byrow=TRUE ) )
  par( mar=c(4,4,2.5,0.5) , mgp=c(2.5,0.7,0) )
  histInfo = plotPost( beta0 , cex.lab = 1.75 , showCurve=showCurve ,
                       compVal=compValBeta0 , ROPE=ropeBeta0 ,
                       xlab=bquote(beta[0]) , main=paste("Intercept") )
  histInfo = plotPost( beta1 , cex.lab = 1.75 , showCurve=showCurve ,
                       compVal=compValBeta1 , ROPE=ropeBeta1 ,
                       xlab=bquote(beta[1]) , main=paste("Slope") )
  plot( beta1[plotIdx] , beta0[plotIdx] ,
        xlab=bquote(beta[1]) , ylab=bquote(beta[0]) ,
        col="skyblue" , cex.lab = 1.75 )
  histInfo = plotPost( sigma , cex.lab = 1.75 , showCurve=showCurve ,
                       compVal=compValSigma , ROPE=ropeSigma ,
                       xlab=bquote(sigma) , main=paste("St. dev.") )
  if ( !is.null(saveName) ) {
    saveGraph( file=paste(saveName,"PostMarg",sep=""), type=saveType)
  }
  #-------------------------------------------------------------------------------
  # Data with superimposed regression lines and noise distributions:
  openGraph()
  par( mar=c(3,3,2,1)+0.5 , mgp=c(2.1,0.8,0) )
  # Plot data values:
  postPredHDImass = 0.95
  xRang = max(x)-min(x)
  yRang = max(y)-min(y)
  xLimMult = 0.25
  yLimMult = 0.45
  xLim= c( min(x)-xLimMult*xRang , max(x)+xLimMult*xRang )
  yLim= c( min(y)-yLimMult*yRang , max(y)+yLimMult*yRang )
  plot( x , y , cex=1.5 , lwd=2 , col="black" , xlim=xLim , ylim=yLim ,
        xlab=xName , ylab=yName , cex.lab=1.5 ,
        main=paste( "Data w. Post. Pred. & ",postPredHDImass*100,"% HDI" ,sep="") ,
        cex.main=1.33  )
  # Superimpose a smattering of believable regression lines:
  nPredCurves=30
  xComb = seq(xLim[1],xLim[2],length=501)
  for ( i in floor(seq(from=1,to=chainLength,length=nPredCurves)) ) {
    lines( xComb , beta0[i] + beta1[i]*xComb , col="skyblue" )
  }
  # Superimpose some vertical distributions to indicate spread:
  #source("HDIofICDF.R")
  nSlice = 5
  curveXpos = seq(min(x),max(x),length=nSlice)
  curveWidth = (max(x)-min(x))/(nSlice+2)
  for ( i in floor(seq(from=1,to=chainLength,length=nPredCurves)) ) {
    for ( j in 1:length(curveXpos) ) {
      sigma = 1/sqrt(tau[i])  # Convert precision to standard deviation
```

```
      yHDI = HDIofICDF( qnorm , credMass=postPredHDImass )  # Use normal distribution
      yComb = seq(yHDI[1],yHDI[2],length=75)
      xVals = dnorm( yComb )  # Use normal density
      xVals = curveWidth * xVals / dnorm(0)
      yPred = beta0[i] + beta1[i]*curveXpos[j]
      yComb = yComb*sigma + yPred
      lines( curveXpos[j] - xVals , yComb , col="skyblue" )
      lines( curveXpos[j] - 0*xVals , yComb , col="skyblue" , lwd=2 )
    }
  }
  # replot the data, in case they are obscured by lines:
  points( x , y , cex=1.5 )
  if ( !is.null(saveName) ) {
    saveGraph( file=paste(saveName,"PostPred",sep=""), type=saveType)
  }
  # if you want to show the y intercept, set this to TRUE:
  showIntercept=TRUE
  if ( showIntercept ) {
    openGraph()
    par( mar=c(3,3,2,1)+0.5 , mgp=c(2.1,0.8,0) )
    # Plot data values:
    xRang = max(x)-min(x)
    yRang = max(y)-min(y)
    xLimMult = 0.25
    yLimMult = 0.45
    xLim= c( min(x)-xLimMult*xRang , max(x)+xLimMult*xRang )
    xLim = c(0,max(xLim))
    yLim= c( min(y)-yLimMult*yRang , max(y)+yLimMult*yRang )
    nPredCurves=30
    pltIdx = floor(seq(from=1,to=chainLength,length=nPredCurves))
    intRange = range( beta0[pltIdx] )
    yLim = range( c(yLim,intRange) )
    postPredHDImass = 0.95
    plot( x , y , cex=1.5 , lwd=2 , col="black" , xlim=xLim , ylim=yLim ,
          xlab=xName , ylab=yName , cex.lab=1.5 ,
          main=paste( "Data w. Post. Pred. & ",postPredHDImass*100,"% HDI" ,sep="") ,
          cex.main=1.33  )
    abline(v=0,lty="dashed")
    # Superimpose a smattering of believable regression lines:
    xComb = seq(xLim[1],xLim[2],length=501)
    for ( i in pltIdx  ) {
      lines( xComb , beta0[i] + beta1[i]*xComb , col="skyblue" )
    }
    # Superimpose some vertical distributions to indicate spread:
    #source("HDIofICDF.R")
    nSlice = 5
    curveXpos = seq(min(x),max(x),length=nSlice)
    curveWidth = (max(x)-min(x))/(nSlice+2)
    for ( i in floor(seq(from=1,to=chainLength,length=nPredCurves)) ) {
      for ( j in 1:length(curveXpos) ) {
        sigma = 1/sqrt(tau[i])  # Convert precision to standard deviation
        yHDI = HDIofICDF( qnorm , credMass=postPredHDImass )  # Use normal distribution
        yComb = seq(yHDI[1],yHDI[2],length=75)
        xVals = dnorm( yComb )  # Use normal density
        xVals = curveWidth * xVals / dnorm(0)
        yPred = beta0[i] + beta1[i]*curveXpos[j]
        yComb = yComb*sigma + yPred
        lines( curveXpos[j] - xVals , yComb , col="skyblue" )
        lines( curveXpos[j] - 0*xVals , yComb , col="skyblue" , lwd=2 )
      }
    }
    # replot the data, in case they are obscured by lines:
```

```
    points( x , y , cex=1.5 )
    if ( !is.null(saveName) ) {
      saveGraph( file=paste(saveName,"PostPredYint",sep=""), type=saveType)
    }
  }
  showPosteriorPredictiveSample = TRUE
  if (showPosteriorPredictiveSample) {
    # Function to sample from posterior predictive distribution
    sample_posterior_predictive <- function(beta0,beta1,sigma, xNew) {

      # Calculate mean for each posterior sample
      muNew <- beta0 + beta1 * xNew

      # Generate predicted y values (one for each posterior sample)
      y_predicted <- rnorm(length(muNew), mean = muNew, sd = sigma)

      return(y_predicted)
    }
    # Sample from posterior predictive at x=20
    y_pred_x20 <- sample_posterior_predictive(beta0,beta1,sigma,xNew = 20)
    # Plot the posterior predictive distribution
    openGraph(width=8,height=5)
    histInfo = plotPost(y_pred_x20,main=paste("Posterior Predictive x=20"))
    if ( !is.null(saveName) ) {
      saveGraph( file=paste(saveName,"PostPredx20",sep=""), type=saveType)
    }
  }
}

#===============================================================================
```

The example script:

```
# Bayesian Data Analysis 2025, Assignment 3, Exercise B
#-------------------------------------------------------------------------------
# Optional generic preliminaries:
graphics.off() # This closes all of R's graphics windows.
rm(list=ls())  # Careful! This clears all of R's memory!
#-------------------------------------------------------------------------------
# Preliminaries
source("openGraphSaveGraph.R")
source("plotPost.R")
require(rjags)
fileNameRoot = "Figures/Mnormal-"
graphFileType = "eps"
# Load the relevant model into R's working memory:
source("Jags-Ymet-Xmet-Mnormal.R")

#-------------------------------------------------------------------------------
# LOAD DATA
myData = read.table('../../caschools.csv', header = TRUE, sep=',')
xName = "stratio" ; yName = "testscr"

# Summary
str(myData)
summary(myData)
# Hist and scatterplot
openGraph(width=12,height=6)
layout( matrix( c(1,2,3) , nrow=1 ) )
hist(myData[,xName])
hist(myData[,yName])
plot(myData[,xName], myData[,yName],
```

```
     xlab = "Student-Teacher Ratio",
     ylab = "Test Score"
)
saveGraph(file=paste0(fileNameRoot,"preliminary"),type="eps")

#------------------------------------------------------------------------------
# Generate the MCMC chain:
#startTime = proc.time()
mcmcCoda = genMCMC( data=myData , xName=xName , yName=yName ,
                    numSavedSteps=20000 , saveName=fileNameRoot )
#stopTime = proc.time()
#duration = stopTime - startTime
#show(duration)

#------------------------------------------------------------------------------
# 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 )
}

#------------------------------------------------------------------------------
# Get summary statistics of chain:
summaryInfo = smryMCMC( mcmcCoda ,
                        compValBeta1=0.0 , ropeBeta1=c(-0.5,0.5) ,
                        saveName=fileNameRoot )
show(summaryInfo)
# Display posterior information:
plotMCMC( mcmcCoda , data=myData , xName=xName , yName=yName ,
          compValBeta1=0.0 , ropeBeta1=c(-0.5,0.5) ,
          pairsPlot=TRUE , showCurve=FALSE ,
          saveName=fileNameRoot , saveType=graphFileType )
#------------------------------------------------------------------------------
```

appendix

The source code:

```
# Bayesian Data Analysis 2025, Assignment 3, Exercise 1
#-------------------------------------------------------------------------------
source("DBDA2E-utilities.R")
#===============================================================================

# Improved initialization function
genMCMC = function(data, numSavedSteps=20000, saveName=NULL) {
  require(rjags)
  savePath = "Figures/Exercise1/2-"

  # LOAD DATA
  if (class(data)=="data.frame") {
    y = myData$y
  } else {
    y = data
  }

  # Do some checking that data make sense:
  if (any(y != floor(y))) { stop("All y values must be integers.") }

  # Compute length
  nBags = length(y)

  # Specify the data in a list
  dataList = list(y = y, nBags = nBags)

  # JAGS MODEL
  modelString = "
  model {
    # Likelihood
    for (i in 1:nBags) {
      y[i] ~ dbin(p[b[i]], n[i])
      n[i] <- round(nContinuous[i])
      nContinuous[i] ~ dnorm(normMu, normTau)
      b[i] <- bBernoulli[i] + 1
      bBernoulli[i] ~ dbern(1-catTheta)
    }
    # Priors
    p[1] ~ dbeta(betaA, betaB)  # when b=0
    p[2] ~ dbeta(1, 1)          # when b=1
    catTheta ~ dbeta(1, 1)
    # Constants
    normMu <- 30
    normTau <- 1/(normSigma*normSigma)
    normSigma <- 1.5
    betaA <- 40
    betaB <- 10
  }
  "

  writeLines(modelString, con="modelExercise1.txt")

  # IMPROVED INITIALIZATION
  # Create multiple initialization lists to increase chances of convergence
  initsList = list()
  for (i in 1:3) { # for each chain
    initsList[[i]] = list(
      p = c(runif(1, 0.7, 0.9), runif(1, 0.6, 0.8)),  # randomize within reasonable range
```

```
      catTheta = runif(1, 0.2, 0.4),                  # randomize within reasonable range
      bBernoulli = rbinom(prob=0.7, size=1, n=nBags),
      nContinuous = pmax(10, rnorm(nBags, mean=30, sd=1.5)) # ensure positive values
    )
  }

  # RUN THE CHAINS
  parameters = c("catTheta", "p[1]", "p[2]")
  adaptSteps = 2000              # Increased for better adaptation
  burnInSteps = 10000            # Increased for better convergence
  nChains = 3
  thinSteps = 1
  nIter = ceiling((numSavedSteps * thinSteps) / nChains)

  # Create, initialize, and adapt the model with robust error handling
  tryCatch({
    jagsModel = jags.model("modelExercise1.txt", data=dataList, inits=initsList,
                           n.chains=nChains, n.adapt=adaptSteps)

    cat("Burning in the MCMC chain...\n")
    update(jagsModel, n.iter=burnInSteps)

    cat("Sampling final MCMC chain...\n")
    codaSamples = coda.samples(jagsModel, variable.names=parameters,
                               n.iter=nIter, thin=thinSteps)

    if (!is.null(saveName)) {
      save(codaSamples, file=paste(saveName, "Mcmc.Rdata", sep=""))
    }
    return(codaSamples)
  }, error = function(e) {
    cat("JAGS Error:", e$message, "\n")
    cat("Attempting to recover with alternative initialization...\n")

    # Alternative initialization as fallback
    altInitsList = list()
    for (i in 1:3) {
      altInitsList[[i]] = list(
        p = c(0.75, 0.65),
        catTheta = 0.3,
        bBernoulli = rep(0, nBags), # simplified initialization
        nContinuous = rep(30, nBags) # simplified initialization
      )
    }

    # Try again with simpler initialization
    jagsModel = jags.model("modelExercise1.txt", data=dataList, inits=altInitsList,
                           n.chains=nChains, n.adapt=adaptSteps*2)
    update(jagsModel, n.iter=burnInSteps*2)
    codaSamples = coda.samples(jagsModel, variable.names=parameters,
                               n.iter=nIter, thin=thinSteps)

    if (!is.null(saveName)) {
      save(codaSamples, file=paste(saveName, "Mcmc.Rdata", sep=""))
    }
    return(codaSamples)
  })
}

#===============================================================================
smryMCMC = function(  codaSamples , compVal=NULL , rope=NULL , saveName=NULL ) {
  summaryInfo = NULL
```

```
  mcmcMat = as.matrix(codaSamples,chains=TRUE)
  summaryInfo = rbind( summaryInfo ,
                       "catTheta" = summarizePost( mcmcMat[,"catTheta"] ,
                                                   compVal=compVal , ROPE=rope ) )
  if ( !is.null(saveName) ) {
    write.csv( summaryInfo , file=paste(saveName,"SummaryInfo.csv",sep="") )
  }
  show( summaryInfo )
  return( summaryInfo )
}

#===============================================================================
plotMCMC = function(codaSamples ,
                    savePath = "Figures/Exercise1/c-",
                    graphFileType = "eps" ) {

  source("openGraphSaveGraph.R")
  source("plotPost.R")

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

  # Extract the posterior sample from JAGS:
  catThetaSample = mcmcChain[,"catTheta"]
  p1Sample = mcmcChain[,"p[1]"]
  p2Sample = mcmcChain[,"p[2]"]
  pDiffSample = p1Sample - p2Sample

  # Make a graph using R commands:
  openGraph(width=10,height=6)
  #layout( matrix( c(1,3) , nrow=1 ) )
  histInfo = plotPost( catThetaSample , xlim=c(0,1) , xlab=bquote("theta") )
  saveGraph(file = paste0(savePath,"catTheta"), type="eps")

  # Make a graph using R commands:
  openGraph(width=10,height=6)
  layout( matrix( c(1,2,3) , nrow=1 ) )
  histInfo = plotPost( p1Sample , xlim=c(0,1) , xlab=bquote("p[1]") )
  histInfo = plotPost( p2Sample , xlim=c(0,1) , xlab=bquote("p[2]") )
  histInfo = plotPost( pDiffSample , xlab=bquote("p[1]-p[2]") , compVal=0. , ROPE=c(-0.1,0.1)
      )
  saveGraph(file = paste0(savePath,"p[1]_p[2]"), type="eps")

} # Close plotMCMC
```

The power estimate experiment:

```
# Jags-Ybin-Nnorm-Bbern-Pbeta-Power.R
graphics.off() # This closes all of R's graphics windows.
rm(list=ls())  # Careful! This clears all of R's memory!
fileNameRoot = "Jags-Ydich-Xnom1subj-MbernBeta-Power-" # for future use

# Load the functions genMCMC, smryMCMC, and plotMCMC:
# (This also sources DBDA2E-utilities.R)
source("Jags-Ybin-Nnorm-Bbern-Pbeta.R")

#-------------------------------------------------------------------------------
# Improved power analysis
goalAchievedForSample = function(data) {
  # Try to generate the MCMC chain with error handling
  tryCatch({
    mcmcCoda = genMCMC(data=data, numSavedSteps=10000, saveName=NULL)
```

```
16
17      # Check for convergence issues
18      gd = gelman.diag(mcmcCoda)
19      if (any(gd$psrf > 1.1)) {
20        warning("Potential convergence issues detected")
21      }
22
23      # Compute the HDI
24      p1HDI = HDIofMCMC(as.matrix(mcmcCoda[,"p[1]"]))
25      p2HDI = HDIofMCMC(as.matrix(mcmcCoda[,"p[2]"]))
26
27      # Define list for recording results
28      goalAchieved = list()
29
30      # Goal: No overlap between HDIs
31      goalAchieved = c(goalAchieved,
32                       "noOverlap"=((p1HDI[2] < p2HDI[1]) || (p2HDI[2] < p1HDI[1])))
33
34      return(goalAchieved)
35    }, error = function(e) {
36      cat("Error in MCMC analysis:", e$message, "\n")
37      # Return NA for this simulation
38      return(list("noOverlap"=NA))
39    })
40  }
41
42  # Main power analysis loop with better data generation
43  powerAnalysis = function() {
44    # Specify hypothetical parameters
45    p = c(0.8, 0.7)
46    catTheta = 0.3
47    normMu = 30
48    normSigma = 1.5
49
50    proportionSuccessList = c()
51
52    # Iterate over sample size (number of bags)
53    for (sampleN in seq(200, 500, by=100)) {
54      nSimulatedDataSets = 10
55      goalCount = 0
56      validSimulations = 0
57
58      for (simIdx in 1:nSimulatedDataSets) {
59        cat("Running simulation", simIdx, "of", nSimulatedDataSets, "for sample size", sampleN,
              "\n")
60
61        # Generate better data with appropriate checks
62        tryCatch({
63          # Generate random value from hypothesized parameter distribution
64          nContinuous = rnorm(n=sampleN, mean=normMu, sd=normSigma)
65          n = round(pmax(5, nContinuous))  # Ensure n is at least 5
66          bBernoulli = rbinom(n=sampleN, size=1, prob=1-catTheta)
67          b = bBernoulli + 1
68
69          # Generate random data based on parameter value
70          sampleY = sapply(1:length(n), function(i) {
71            rbinom(n=1, size=n[i], prob=p[b[i]])
72          })
73
74          # Verify data validity
75          if (all(is.finite(sampleY)) && all(sampleY >= 0) && all(sampleY <= n)) {
76            # Do Bayesian analysis on simulated data
```

```
          goalAchieved = goalAchievedForSample(sampleY)

          # Only count valid results
          if (!is.na(goalAchieved[["noOverlap"]])) {
            validSimulations = validSimulations + 1
            if (goalAchieved[["noOverlap"]] == TRUE) {
              goalCount = goalCount + 1
            }
          }
        } else {
          cat("Generated invalid data in simulation", simIdx, "\n")
        }
      }, error = function(e) {
        cat("Error in simulation", simIdx, "for sample size", sampleN, ":", e$message, "\n")
      })
    }

    # Calculate proportion of successes
    if (validSimulations > 0) {
      proportionSuccess = goalCount / validSimulations
      cat("\nSampleN =", sampleN, "| Success rate:", round(proportionSuccess * 100, 2),
          "% (", goalCount, "out of", validSimulations, "valid simulations)\n\n")

      proportionSuccessList[as.character(sampleN)] = proportionSuccess

      # Check stopping condition
      if (proportionSuccess > 0.8 && validSimulations >= nSimulatedDataSets*0.8) {
        print(proportionSuccessList)
        cat("Stopping early: goal achieved in over 80% of valid simulations for sample size",
            sampleN, "\n")
        break
      }
    } else {
      cat("No valid simulations completed for sample size", sampleN, "\n")
    }
  }

  return(proportionSuccessList)
}

# Run the power analysis
results = powerAnalysis()
print(results)
```