ORIGIN ≡ 0

# Bayesian Analysis using MCMC
# Analog to the One Sample t-Test

**Markov Chain Monte Carlo (MCMC) methods have become an increasingly popular Bayesian supplement or alternative to standard (frequentist) statistical methods in assessing complex linear models (LM) and generalized linear models (GLM). Bayesian methods focus on the relationship between data collected in a particular study (D) and parameters of a model (Θ) chosen to interpret the data. According to Bayesian reasoning, all parameters of the model are assigned "prior" probabilities P(θ). Likelihood of the data given specific parameter values P(D|θ) are determined, and from this, "posterior" probabilities P(Θ|D) are calculated using Bayes Rule. The object of this form of Bayesian analysis is to find values of the parameters (Θ) that maximize posterior probability. Because Bayesian analysis incorporates prior knowledge, or guesswork, about the probability of parameter values P(Θ), the posterior probability P(Θ|D) makes use of this information for better or worse. If prior information is strong and data weak, then Bayes rule will assign posterior probability greatly influenced by the prior. This can be a good thing. In many if not most cases however, prior information is weak. Standard practice is then to set "uninformative" (or uninformed) prior probabilities thus allowing the data to dominate posterior probability. Under these circumstances, it doesn't really matter what prior probabilities P(Θ) are initially used, as long as they are uninformed; P(Θ|D) remains stable.**

**Some models applied to data are simple enough that Bayes rule can be applied "analytically" (i.e., through exact formula). In more complex models, however, "marginal probabilities" involving sums or integrals in the denominator of Bayes rule resist analytic solution. In addition, a complex model involving many parameters involves a "space" of possible parameter combinations that is usually too large for exhaustive enumeration of posterior probability for each and every parameter value combination. Therefore, a heuristic search "random walk" algorithm is employed that starts somewhere in the combination of parameter values, and seeks a posterior probability maximum. The form of random walk employed is a first order Markov chain - meaning that each new step taken in the walk is dependent only on current position in the parameter space. Step direction is determined by a form of the Metropolis algorithm (named after Nicholas Metropolis) in which stepping trends toward the posterior probability maximum, but also samples lesser values at frequencies related to their posterior probabilities. Typically multiple Markov chains are run, usually with differing starting points, to verify the existence of a single posterior probability maximum. When the Markov chain random walks stabilize, the Metropolis algorithm produces a frequency distribution of parameter estimates directly related to posterior probabilities around that maximum. Specific, and generally more efficient, variants of the Metropolis algorithm include the Gibbs sampler (in JAGS) and Hamiltonian Monte Carlo methods (in Stan), among others.**

**Shown below is a comparison example of MCMC analysis involving a model of data typical for the one-sample t-test. The model involves the assumption that a collected sample of data is derived from a Normal distribution governed by two parameters: Θ = (μ for mean, and σ for standard deviation). We wish to estimate these parameters. Although the model is simple, implementation of MCMC analysis in R involves a major increase in programming sophistication. Fortunately, J.K. Kruschke has come to our rescue with his wonderfully informative book *Doing Bayesian Data Analysis - A Tutorial with R, JAGS, and Stan* (i.e., 2nd edition) that includes complete scripts intended as "scaffolds" for doing many different kinds of MCMC analysis using R. The complete set of scaffolds are available at https://sites.google.com/site/doingbayesiandataanalysis/. Here I swipe the scaffold for the one-sample model, and modify it slightly for our use. For an introduction to Bayesian methods, Kruschke's text is highly recommended.**

## Set up a dedicated folder:

```
getwd()
setwd("C:/010 MCMC")
getwd()
# Example for Jags-Ymet-Xnom1grp-Mnormal.R
#------------------------------------------------------------------------
# Optional generic preliminaries:
graphics.off() # This closes all of R's graphics windows.
rm(list=ls())  # Careful! This clears all of R's memory!
#------------------------------------------------------------------------
```

**Since Kruschke's scaffold involves multiple files both read as scripts in R and written as interim files and analysis results, it is probably wise to set up a dedicated folder for each analysis to keep everything together. I copied his relevant scaffold files to a new folder labeled "010 MCMC". I'll modify these, and maintain his pristine originals elsewhere.**

**Kruschke's scaffold for each MCMC analysis always involves three files, in this case labeled:**

```
Jags-Ymet—Xnom1grp—Mnormal-Example.R
Jags—Ymet—Xnom1grp—Mnormal.R
DBDA2E—utilities.R
```

**These files, destined to be run in JAGS via R package rjags, exist in a hierarchy. The first,** `Jags—Ymet—Xnom1grp—Mnorma1-Example.R`**, is a top-level "driver". It organizes all activities and calls functions residing in the other files. The second file,** `Jags—Ymet—Xnom1grp-Mnormal.R`**, is a 2nd level file consisting of model specifcations and calls to a 3rd level file,** `DBDA2E-utilities.R`**, containing a general set of functions, used by multiple models.**

**Since, I've modified the driver here, I will re-label it, but leave the other files unchanged. So, the hierarchy of files in my folder is:**

```
010 JAGS DRIVER.R
Jags-Ymet—Xnomlgrp-Mnormal.R
DBDA2E-utilities.R
```

## Read Data and Do a t-Test:

**The data is from Kruschke's simulated data of IQ in** `TwoGroupIQ.csv`**, a comma-delimited file, easily written by MS Excel. We use only the "Smart Drug" group for analysis in what follows.**

**From:** `Jags.OneSampleT-Driver.R`

```
# Load The data file
myDataFrame = read.csv( file="TwoGroupIQ.csv" )
myData = myDataFrame$Score[myDataFrame$Group=="Smart Drug"]
myData
```

```
> myData
 [1] 102 107  92 101 110  68 119 106  99 103  90  93  79  89
[15] 137 119 126 110  71 114 100  95  91  99  97 106 106 129
[29] 115 124 137  73  69  95 102 116 111 134 102 110 139 112
[43] 122  84 129 112 127 106 113 109 208 114 107  50 169 133
[57]  50  97 139  72 100 144 112
```

```
t.test(myData,alternative="two.sided",mu=100,conf.level = 0.95)
```

```
One Sample t-test

data:  myData
t = 2.446, df = 62, p-value = 0.0173
alternative hypothesis: true mean is not equal to 100
95 percent confidence interval:
 101.4330 114.2496
sample estimates:
mean of x
 107.8413
```

```
sd(myData)
```

```
> sd(myData)
[1] 25.4452
```

**^ Mean of sample lies outside the 95% confidence interval, with P value = 0.0173**
**Point estimate of $\mu$ = 107.8413; point estimate of $\sigma$ = 25.4452.**

## MCMC Analysis:

**Remainder of driver file:**

```
# Optional: Specify filename root and graphical format for saving output.
# Otherwise specify as NULL or leave saveName and saveType arguments
# out of function calls.
fileNameRoot = "OneGroupIQnormal-"
graphFileType = "eps"
#------------------------------------------------------------------------------
# Load the relevant model into R's working memory:
source("Jags-Ymet-Xnom1grp-Mnormal.R")
#------------------------------------------------------------------------------
# Generate the MCMC chain:
mcmcCoda = genMCMC( data=myData , numSavedSteps=20000 , saveName=fileNameRoot )
#------------------------------------------------------------------------------
# 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 ,
            compValMu=100.0 , ropeMu=c(99.0,101.0) ,
            compValSigma=15.0 , ropeSigma=c(14,16) ,
            compValEff=0.0 , ropeEff=c(-0.1,0.1) ,
            saveName=fileNameRoot )
show(summaryInfo)
# Display posterior information:
plotMCMC( mcmcCoda , data=myData ,
      compValMu=100.0 , ropeMu=c(99.0,101.0) ,
      compValSigma=15.0 , ropeSigma=c(14,16) ,
      compValEff=0.0 , ropeEff=c(-0.1,0.1) ,
      pairsPlot=TRUE , showCurve=FALSE ,
      saveName=fileNameRoot , saveType=graphFileType )
#------------------------------------------------------------------------------
```

**< Multiple files are written in the working directory. Graphics files are in *.eps format.**

**< 2nd level file placed in memory here.**

**< diagnostic function at 2nd level called here, generating graphic output.**

**< summary information at 2nd level called here, with results reported to R's console and to a *.csv file.**

**< Estimate distribution function at 2nd level called here. Results plotted in *.eps files and repored to R's console graphics.**

```
# THE MODEL.
 modelString = "
 model {
  for ( i in 1:Ntotal ) {
   y[i] ~ dnorm( mu , 1/sigma^2 )
  }
  mu ~ dnorm( meanY , 1/(100*sdY)^2 )
  sigma ~ dunif( sdY/1000 , sdY*1000 )
 }
 " # close quote for modelString
```

**Inside the 2nd level file:** `Jags-Ymet-Xnom1grp-Mnormal.R` **function gen MCMC() contains specification of the model as well as the driver for running the MCMC. This includes setting up uninformative prior probabilities, setting initial values for four chains, running a "burn-in" set of steps (hopefully finding a maximum in posterior probability), and then sampling the distribution of parameters near the maximum.**

## MCMC Run & Sample Results:

```
Compiling model graph
   Resolving undeclared variables
   Allocating nodes
Graph information:
   Observed stochastic nodes: 63
   Unobserved stochastic nodes: 2
   Total graph size: 206

Initializing model

  |++++++++++++++++++++++++++++++++++++++++++++++++++| 100%
Burning in the MCMC chain...
  |**************************************************| 100%
Sampling final MCMC chain...
  |**************************************************| 100%
```

**> show(summaryInfo)**

```
              Mean      Median        Mode     ESS HDImass       HDIlow
mu    107.8483785 107.8475830 107.6582644 20000.0    0.95 101.22564584
sigma  25.9843709  25.8224529  25.5828597 11418.5    0.95  21.53742063
effSz   0.3045989   0.3044004   0.2967111 19487.3    0.95   0.04509976
          HDIhigh CompVal PcntGtCompVal ROPElow ROPEhigh PcntLtROPE PcntInROPE
mu    114.2027194     100        99.085    99.0    101.0      0.420       1.53
sigma  30.8633374      15       100.000    14.0     16.0      0.000       0.00
effSz   0.5522863       0        99.085    -0.1      0.1      0.085       5.59
      PcntGtROPE
mu        98.050
sigma    100.000
effSz     94.325
```

**^ Summary results from smryMCMC() in the 2nd level file are reported to R's console. These include means, medians, modes for the posterior distribution of parameters μ and σ. A derived quantity for Effect Size is also calculated.**

**Effect Size (EFFSz) is defined by: EFFSz = (μ-100)/σ**

## MCMC Run Diagnostics:

**Behavior of the Markov chains are assessed graphically. For each parameter:**

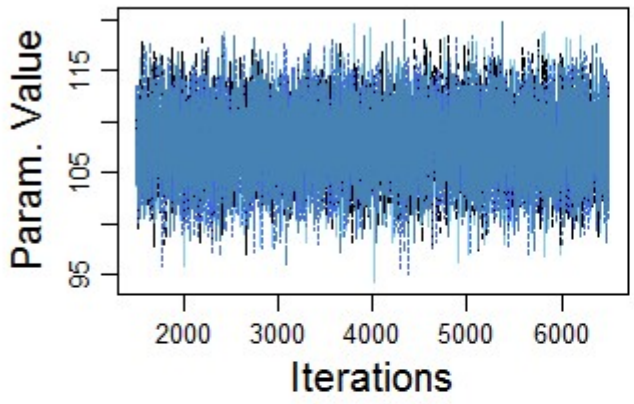
**Top-left graph indicates "mixing" of the chains. One hopes to see that the values sampled are more-or-less equivalent across chains, indicating that none of the chains have become stuck at a unique place in the sampled parameter space.**

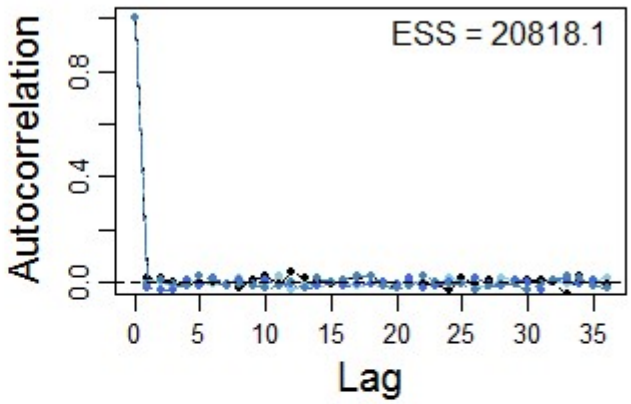
**Top-right graph shows autocorrelation between successive steps for each Markov chain. One hopes to see the chains settle down during the run, observed left to right, to essentially zero autocorrelation. ESS is effective sample size, which is a ratio of steps over ACF (autocorrelation factor) for all autocorrelation lags. For a stable & accurate estimates, ESS equal to or greater than 10,000 is preferred. See Kruschke p. 184.**

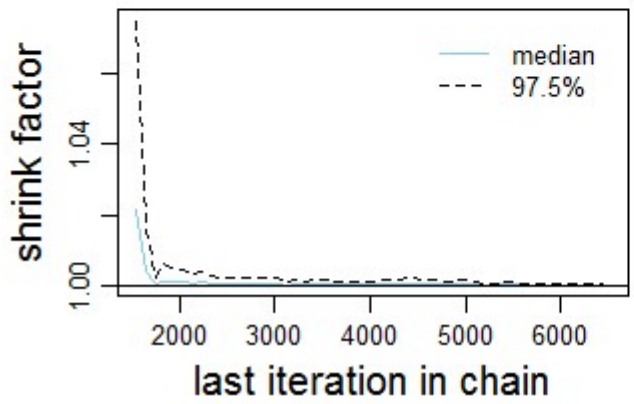
**Bottom-left graph shows the "shrink factor" (Gelman-Rubin convergence statistic) along the chains. This statistic measures a ratio of variance between chains versus within. Kruschke suggests that levels below 1.1 indicate satisfactory convergence of the chains toward a single maximum.**

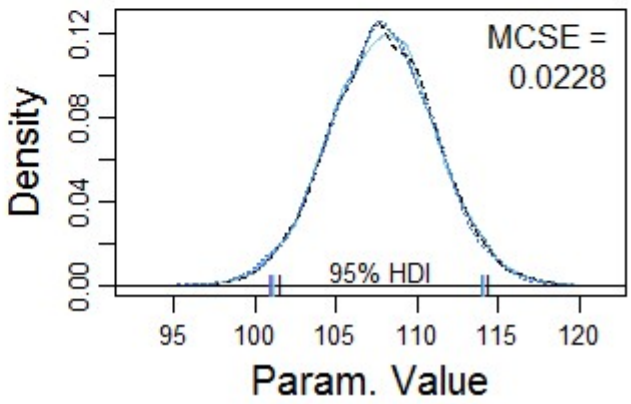
**Bottom-right shows the distribution of the parameter estimates for each Markov chain. One hopes to see close agreement among the curves. MCSE = standard deviation of sample/sqrt(ESS). Low values indicate a stable estimate of the parameter mean - a good thing.**

# mu

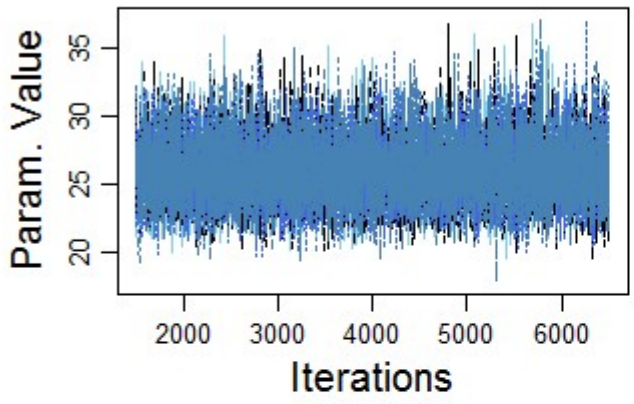

Param. Value
95
105
115
2000
3000
4000
5000
6000
Iterations

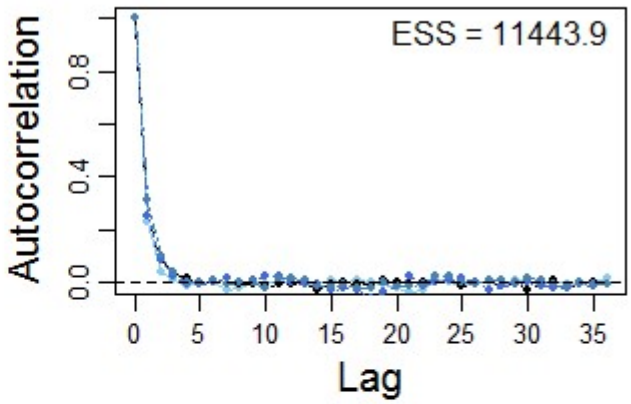

Autocorrelation
0.0
0.4
0.8
ESS = 20818.1
0
5
10
15
20
25
30
35
Lag

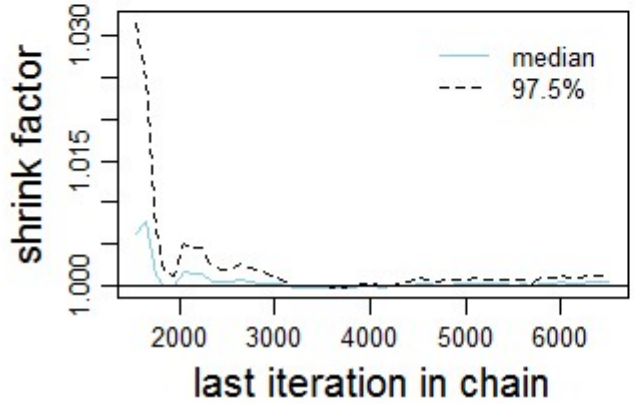

shrink factor
1.00
1.04
median
97.5%
2000
3000
4000
5000
6000
last iteration in chain

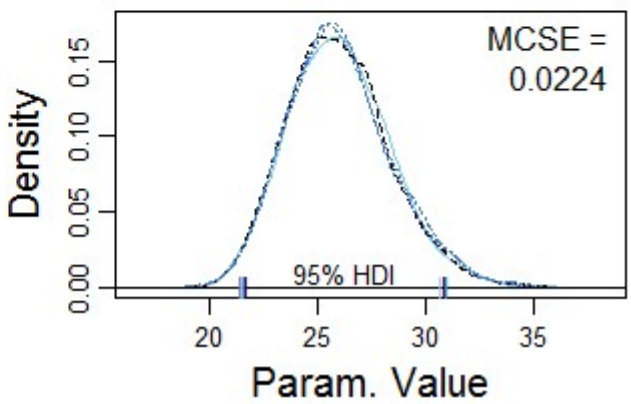

Density
0.00
0.04
0.08
0.12
MCSE =
0.0228
95% HDI
95
100
105
110
115
120
Param. Value

# sigma


Param. Value
20
25
30
35
2000
3000
4000
5000
6000
Iterations

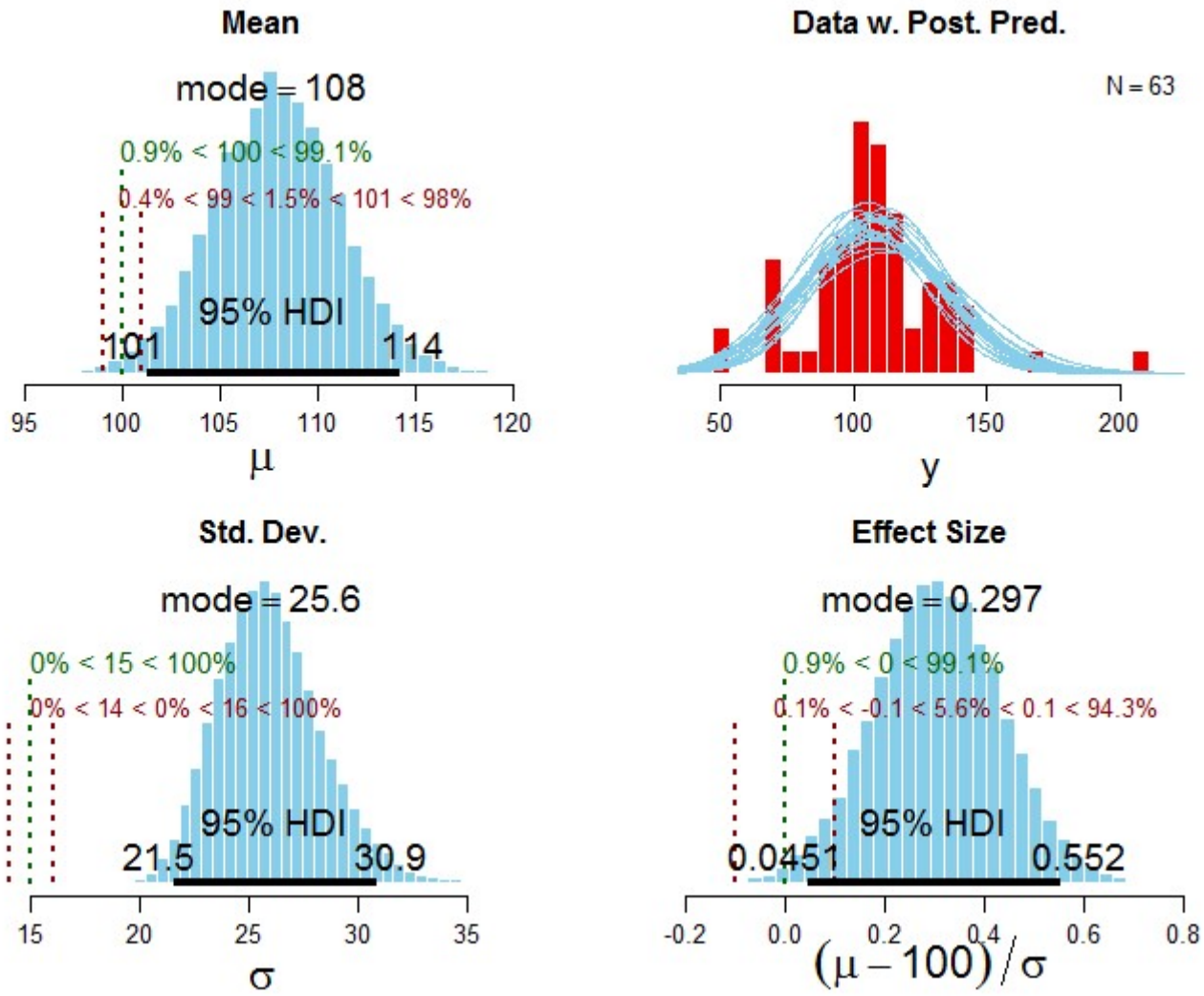

Autocorrelation
0.0
0.4
0.8
ESS = 11443.9
0
5
10
15
20
25
30
35
Lag


shrink factor
1.000
1.015
1.030
median
97.5%
2000
3000
4000
5000
6000
last iteration in chain


Density
0.00
0.05
0.10
0.15
MCSE =
0.0224
95% HDI
20
25
30
35
Param. Value

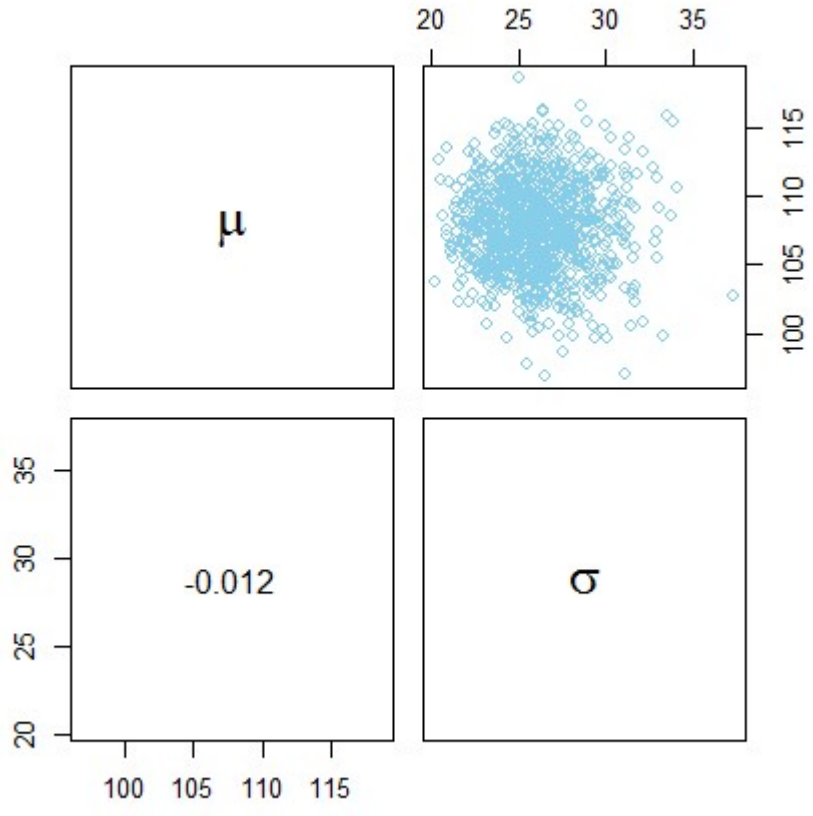
## MCMC Parameter Estimates & HDI:

**The blue histograms on this page display posterior distributions found for μ, σ, and Effect Size already summarized above. By default, as specified in the header of function plotPost() in'3rd level file `DBDA2E-utilities.R`, the mode of each distribution is given numeric value in the graph (median and mean are other options). Thus, a mode of μ = 109 is observed for the posterior probability distribution of μ collected by the Gibbs sampler, with 95% of the posterior sample ranging between 102 and 114. The interval, termed "95% Highest Density Interval" (HDI), functions in a way somewhat similar to Confidence Intervals (CI) in regular statistics. A "Comparison Value" (100 in this instance shown in green and passed to functions smryMCMC() and plotMCMC() in the 2nd level file by the firs level driver), indicates a value to compare the HDI against more-or-less similar to conducting a t-test using CIs. However, as pointed out, CIs are formulated by considering the concept of repeated samples in regular statistics, whereas HDIs indicate a posterior probability distribution, so the two are not equivalent. In this case a Comparison Value of 100 resides below 99% of the sampled posterior distribution, so is very unlikely to characterize μ. The other boundaries (indicated in red on the histograms) specify a "Region of Practical Equivalence" (ROPE) for the Comparison Value (100), allowing for graded as opposed to all-or-none decisions of "in" versus "out"of the HDI. The ROPE is specified in the same manner as the Comparison Value in the driver file. Histograms for σ and Effect Size work similarly, with Comparison Values of (15,0) for (σ, Effect Size) respectively.**




**The top-right graph, termed "Post Prediction Plot" compares frequencies of the original data in red (N=62) versus a sample of Normal distributions determined by values of (μ, σ) that have near the highest joint posterior probability. The number of curves plotted is specified in function plotMCMC() within the 2nd level file `Jags-Ymet-Xnomlgrp-Mnormal.R`.**

## Pairwise Plots




**The stable sample of parameters derived from successive steps in the Markov chains are simultaneous and joint estimates, in this case of μ and σ. Pairwise plots are used to identify whether parameter estimates are correlated. If so, then the histograms above, as marginal estimates of each parameter separately, cannot tell the whole story! Here, correlation between μ and σ is low.**

## So, how do the t-test and MCMC results compare?

| Parameter: | t-Test: | 95% CI | MCMC using JAGS: | 95% HDI |
|---|---|---|---|---|
| μ: | sample mean = 107.8413 | [101.4 - 114. 2] | posterior distribution mode = 109 | [102 - 114] |
| σ' | sample sd = 25.4452 | | posterior distribution mode = 25.7 | [21.6 - 30.8] |

**Although derived from very different perspectives, application of a Normal distribution model to the data yields roughly comparable results. As gleefully pointed out by Bayesians, MCMC provides more detail, and this extra information may turn out to be useful. Notably, the Post Prediction Plot above suggests that the Normal distribution model applied in both Bayesian and non-Bayesian studies may not be adequate. The central hump of the data exceeds that predicted by "best-fit" Normal distribution curves because data outliers pull the Normal distribution outward (i.e.. increase σ) to account for such divergent observations. So, as suggested by Kruschke, a different model distribution might better fit the data.**