jags: Mean and variance of Anorexia data

C..J. Andersion

9/18/2019

In this session, we’ll use rjags and runjags to estimate the mean and variance of a weight change of girls being treated for Anorexia.

## Set-up

Below are the packages that we’ll be using. coda will produce various diagnositics and the other packages are different front ends of jags.

library(coda)  
library(rjags)

## Linked to JAGS 4.3.0

## Loaded modules: basemod,bugs

library(runjags)  
library(jagsUI)

## Loading required package: lattice

##   
## Attaching package: 'jagsUI'

## The following object is masked from 'package:coda':  
##   
## traceplot

## The following object is masked from 'package:utils':  
##   
## View

As always, our first step is to set your working directory, read in the data, create a change in weight score, and take a look at the data:

setwd("C:/Users/cja/dropbox/edps 590BAY/Lectures/6 Gibbs and jags")  
ano <- read.table("anorexia\_data.txt",header=T)  
ano$change <- ano$weight2 - ano$weight1  
head(ano)

## rx girl weight1 weight2 change  
## 1 1 1 80.5 82.2 1.7  
## 2 1 2 84.9 85.6 0.7  
## 3 1 3 81.5 81.4 -0.1  
## 4 1 4 82.6 81.9 -0.7  
## 5 1 5 79.9 76.4 -3.5  
## 6 1 6 88.7 103.6 14.9

I would like to get the same results each time I run this code, so I’ll set a seed for the random number generaters.

set.seed(234590)

## Create data list

The data list will be the same for rjags, runjags and jagsUI. We list and define variables that will be used to create our data model (i.e., the likelihood). For our problem this is

dataList <- list(y=ano$change,   
 Ntotal=length(ano$change),  
 meanY = mean(ano$change),  
 sdY = sd(ano$change)  
 )

The dataLlist object now contains the following

dataList

## $y  
## [1] 1.7 0.7 -0.1 -0.7 -3.5 14.9 3.5 17.1 -7.6 1.6 11.7  
## [12] 6.1 1.1 -4.0 20.9 -9.1 2.1 -1.4 1.4 -0.3 -3.7 -0.8  
## [23] 2.4 12.6 1.9 3.9 0.1 15.4 -0.7 -0.5 -9.3 -5.4 12.3  
## [34] -2.0 -10.2 -12.2 11.6 -7.1 6.2 -0.2 -9.2 8.3 3.3 11.3  
## [45] 0.0 -1.0 -10.6 -4.6 -6.7 2.8 0.3 1.8 3.7 15.9 -10.2  
## [56] 11.4 11.0 5.5 9.4 13.6 -2.9 -0.1 7.4 21.5 -5.3 -3.8  
## [67] 13.4 13.1 9.0 3.9 5.7 10.7  
##   
## $Ntotal  
## [1] 72  
##   
## $meanY  
## [1] 2.763889  
##   
## $sdY  
## [1] 7.983598

## Create the data model

The model will define the likelihood and also give the priors for the parameters being estimated. Note that jags “dnorm” is different from base “dnorm”. For jags, the parameters for jags “dnorm” are the mean and preicsion, which is 1/variance of the data.

Model1 = "model {   
 for (i in 1:Ntotal){  
 y[i] ~ dnorm( mu, prescision )   
 }  
 mu ~ dnorm( meanY , 1/(100\*sdY^2) ) # mean (y) and precision N(2.763889,1.568927e-06)  
 prescision <- pow(sigma, -2) # tau is presicion = 1/var(y)  
 sigma ~ dunif( sdY/1000, sdY\*1000 ) # variance (y) and uniform(.00798,7983.598)   
 }  
 "

Note the quotes around this code. We will save the model to a text file that will be called by jags. The folowing command writes the model to a file that I’m calling “``”Model1.txt"

writeLines(Model1, con="Model1.txt")

Go to your working memory and look for “Model1.txt” and take a look at it.

## Set Initial values (rjags)

Regardless of which implemenations you use for jags, they all require that intital values are a list object. In this case, I simpley used the same mean and variance.

thetaInit = mean(ano$change)  
sigmaInit = sd(ano$change)  
initsList = list(mu=thetaInit,sigma=sigmaInit)

## Run rjags

jagsModel1 <- jags.model(file="Model1.txt", # compiles and intializes model  
 data=dataList,  
 inits=initsList,  
 n.chains=4,  
 n.adapt=500) # adaptive phase for 500 iterations

## Compiling model graph  
## Resolving undeclared variables  
## Allocating nodes  
## Graph information:  
## Observed stochastic nodes: 72  
## Unobserved stochastic nodes: 2  
## Total graph size: 88  
##   
## Initializing model

Model model has successfully complieds; that is, it has be initialized and we can do some sampling.

update (jagsModel1, n.iter=500) # gets samples from posterior with a   
 # burn in of 500 iterations

And now for getting the samples for many iterations.

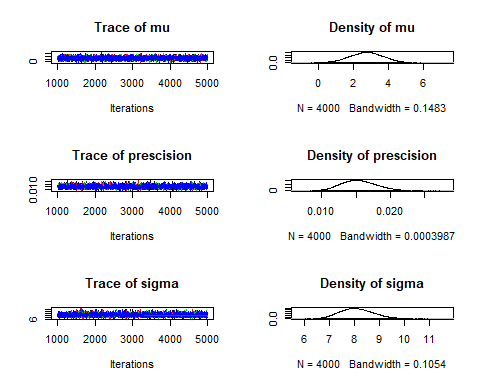
# contains samples from all chains with 500 iterations  
Samples <- coda.samples(jagsModel1, variable.names=c("mu"," prescision","sigma"), n.iter=4000)

## Did our model converge?

There are many diagnositics that we can look at including densities, traceplots, geweke statistics, Rhat/psrf/Gelman statistics, auto-correlations, effective sample sizes and more.

We will start with trace plots and densities for each of our parameters that we’ve estimated. Note that these look better (nicer formating) when I run them in R.

plot(Samples)



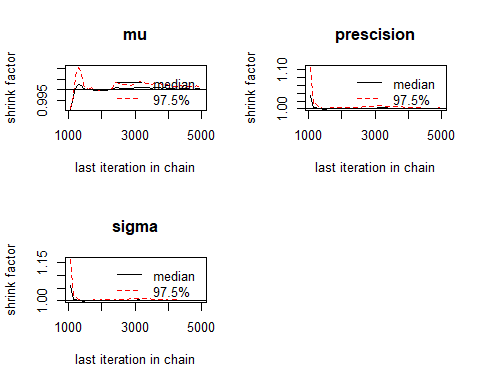
Although we can look at the trace plots for stationarity and nice mixing of the chains, to back what we think we see we can get the Gelman (proportional scale reduction factors or Rhats). These should be less than .01.

gelman.diag(Samples)

## Potential scale reduction factors:  
##   
## Point est. Upper C.I.  
## mu 1 1  
## prescision 1 1  
## sigma 1 1  
##   
## Multivariate psrf  
##   
## 1

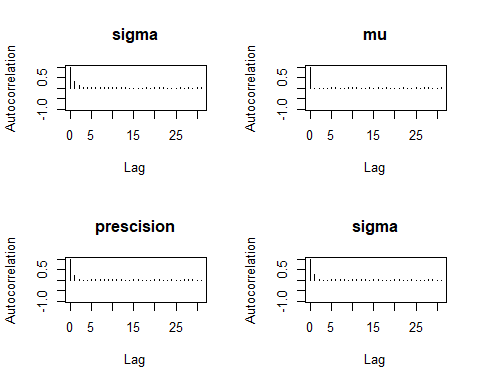
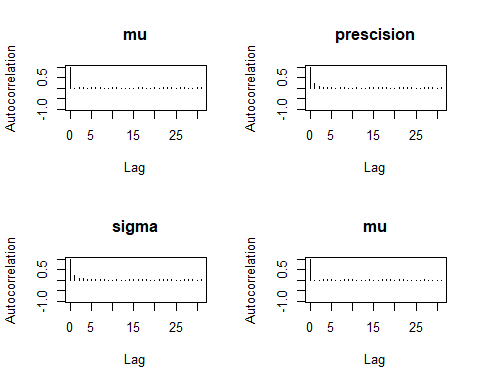
Can’t get much better than these numbers. We can also get a plot of them

gelman.plot(Samples)



Below are dome other diagnostics as can look at and these done for each chain for each parameter.

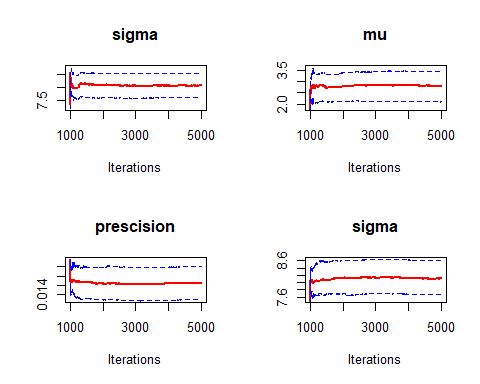
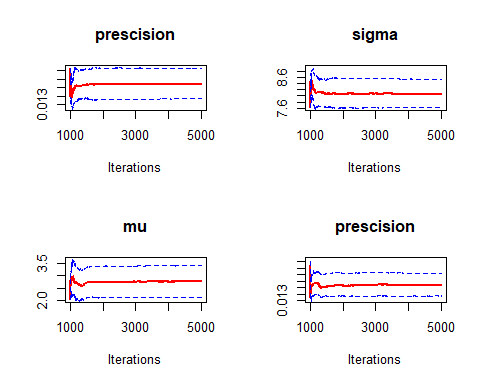
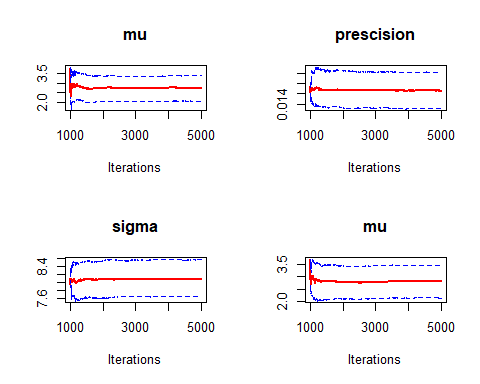
autocorr.plot(Samples,auto.layout=TRUE)



geweke.diag(Samples,frac1=0.1,frac2=0.5)

## [[1]]  
##   
## Fraction in 1st window = 0.1  
## Fraction in 2nd window = 0.5   
##   
## mu prescision sigma   
## -0.1309 0.9966 -1.1390   
##   
##   
## [[2]]  
##   
## Fraction in 1st window = 0.1  
## Fraction in 2nd window = 0.5   
##   
## mu prescision sigma   
## -0.9688 -0.5949 0.4945   
##   
##   
## [[3]]  
##   
## Fraction in 1st window = 0.1  
## Fraction in 2nd window = 0.5   
##   
## mu prescision sigma   
## -2.57351 -0.15325 -0.03616   
##   
##   
## [[4]]  
##   
## Fraction in 1st window = 0.1  
## Fraction in 2nd window = 0.5   
##   
## mu prescision sigma   
## 1.0452 0.1595 -0.4104

cumuplot(Samples,probs=c(.25,.50,.75),lwd=c(1,2),lty=c(2,1),col=c("blue","red"))



Lastley, we might want to know what the effective sample size is for each of our parameters in the posterior distribution.

effectiveSize(Samples)

## mu prescision sigma   
## 16000.000 10188.271 9233.592

## Summary Statistic from our Posterior

Dropping the burn-ins or warm-ups, we can compute various sample statistics that describe our posterior distributions. Note that these are computed over all chains.

summary(Samples)

##   
## Iterations = 1001:5000  
## Thinning interval = 1   
## Number of chains = 4   
## Sample size per chain = 4000   
##   
## 1. Empirical mean and standard deviation for each variable,  
## plus standard error of the mean:  
##   
## Mean SD Naive SE Time-series SE  
## mu 2.76632 0.969424 7.664e-03 7.664e-03  
## prescision 0.01548 0.002621 2.072e-05 2.603e-05  
## sigma 8.12546 0.697490 5.514e-03 7.273e-03  
##   
## 2. Quantiles for each variable:  
##   
## 2.5% 25% 50% 75% 97.5%  
## mu 0.83422 2.10945 2.76537 3.41081 4.67311  
## prescision 0.01078 0.01363 0.01533 0.01713 0.02103  
## sigma 6.89616 7.64111 8.07706 8.56441 9.63193

## runjags run

Different front ends onto jags are a bit different in terms of default output and input. All these use the same datalist and model but differ in terms of initial or starting values and getting the samples. First we’ll look at runjag and then jagsUI. The latter is overly verbous.

So to we set up out starting values. I’ll use the sample mean and standard deviation of change values for one of the chains and for the others use different means and variances. For the other 3 chains, I sampled from normal distributions with different means and standard deviations.

(meanY = mean(ano$change))

## [1] 2.763889

(sdY = sd(ano$change))

## [1] 7.983598

initsList = list(list("mu"=meanY, "sigma"=sdY),  
 list("mu"=rnorm(1,2,4), "sigma"=1 ),  
 list("mu"=rnorm(1,4,1), "sigma"=8 ),  
 list("mu"=rnorm(1,-4,2),"sigma"=.5 )  
 )

Go ahead and take a look at this.

The next comand gets the samples (i.e., runs jags)

out.runjags <- run.jags(model=Model1,   
 monitor=c("mu","sigma","prescision","dic"),  
 data=dataList,   
 n.chains=4,   
 inits=initsList)

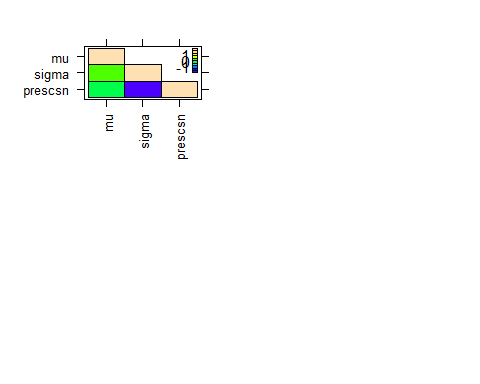
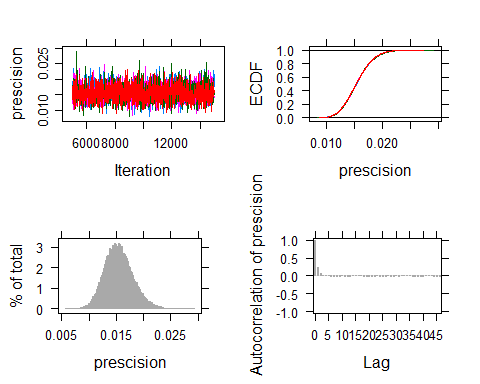
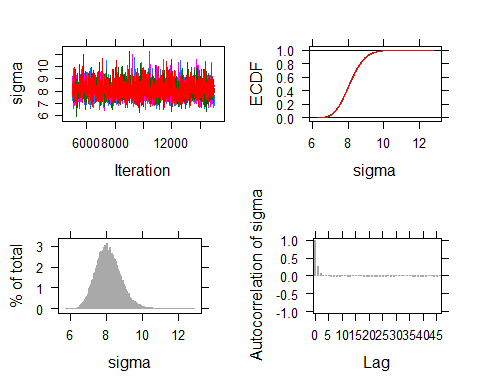
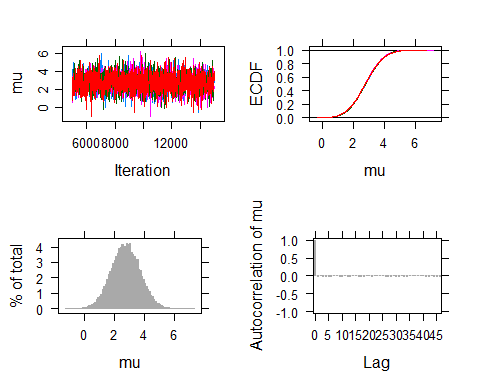
## module dic loaded

## Compiling rjags model...  
## Calling the simulation using the rjags method...  
## Adapting the model for 1000 iterations...  
## Burning in the model for 4000 iterations...  
## Running the model for 10000 iterations...  
## Simulation complete  
## Calculating summary statistics...  
## Calculating the Gelman-Rubin statistic for 3 variables....  
## Finished running the simulation

runjag will easly product diagnostic plots, for example

plot(out.runjags)

## Generating plots...

 You can also use coda to produce the same graphics as I did with rjags.

The table of summary statistics from runjag includes more than rjags gives.

print(out.runjags)

##   
## JAGS model summary statistics from 40000 samples (chains = 4; adapt+burnin = 5000):  
##   
## Lower95 Median Upper95 Mean SD Mode MCerr  
## mu 0.90864 2.7601 4.6631 2.7607 0.96131 -- 0.0048272  
## sigma 6.8095 8.0819 9.5136 8.1313 0.69491 -- 0.0045825  
## prescision 0.010511 0.01531 0.020685 0.015451 0.0025967 -- 0.000016522  
##   
## MC%ofSD SSeff AC.10 psrf  
## mu 0.5 39658 -0.00025417 1.0001  
## sigma 0.7 22996 0.0065709 1  
## prescision 0.6 24701 0.0064725 1  
##   
## Model fit assessment:  
## DIC = 506.579  
## [PED not available from the stored object]  
## Estimated effective number of parameters: pD = 2.06232  
##   
## Total time taken: 3.9 seconds

## jagsUI

If you only use jags once in a while and don’t remember what all this various statistics are, the jagsUI is for you. We use the same data list, model, and intial values list as we did for runjags, but a little different command to get the samples.

out.jagsUI <- jags(model.file="Model1.txt",  
 data=dataList,inits=initsList,   
 parameters.to.save=c("mu","sigma","prescision"),   
 n.iter=2000,  
 n.burnin=500,  
 n.chains=4)

##   
## Processing function input.......   
##   
## Done.   
##   
## Compiling model graph  
## Resolving undeclared variables  
## Allocating nodes  
## Graph information:  
## Observed stochastic nodes: 72  
## Unobserved stochastic nodes: 2  
## Total graph size: 88  
##   
## Initializing model  
##   
## Adaptive phase.....   
## Adaptive phase complete   
##   
##   
## Burn-in phase, 500 iterations x 4 chains   
##   
##   
## Sampling from joint posterior, 1500 iterations x 4 chains   
##   
##   
## Calculating statistics.......   
##   
## Done.

and a table of summary statistics

print(out.jagsUI)

## JAGS output for model 'Model1.txt', generated by jagsUI.  
## Estimates based on 4 chains of 2000 iterations,  
## adaptation = 100 iterations (sufficient),  
## burn-in = 500 iterations and thin rate = 1,  
## yielding 6000 total samples from the joint posterior.   
## MCMC ran for 0.007 minutes at time 2019-09-18 15:57:34.  
##   
## mean sd 2.5% 50% 97.5% overlap0 f Rhat  
## mu 2.743 0.962 0.859 2.745 4.615 FALSE 0.997 1.000  
## sigma 8.141 0.708 6.873 8.091 9.663 FALSE 1.000 1.001  
## prescision 0.015 0.003 0.011 0.015 0.021 FALSE 1.000 1.001  
## deviance 504.563 2.112 502.519 503.926 510.116 FALSE 1.000 1.003  
## n.eff  
## mu 3511  
## sigma 3982  
## prescision 6000  
## deviance 1949  
##   
## Successful convergence based on Rhat values (all < 1.1).   
## Rhat is the potential scale reduction factor (at convergence, Rhat=1).   
## For each parameter, n.eff is a crude measure of effective sample size.   
##   
## overlap0 checks if 0 falls in the parameter's 95% credible interval.  
## f is the proportion of the posterior with the same sign as the mean;  
## i.e., our confidence that the parameter is positive or negative.  
##   
## DIC info: (pD = var(deviance)/2)   
## pD = 2.2 and DIC = 506.791   
## DIC is an estimate of expected predictive error (lower is better).