RATS

Rats: hierarchical linear regression

This example is taken from the openBUGS example library, originally from section 6 of Gelfand et al (1990), and concerns 30 young rats whose weights were measured weekly for five weeks.

The goal of this example is to give everyone a look at what fitting a model in JAGS looks like, (the nuts and bolts) but also a little bit of hierarchical modelling. We'll start with a single regression model, and then add one level of hierarchy: a population intercept and a population slope.

Let's look at the data, which we first have to load:

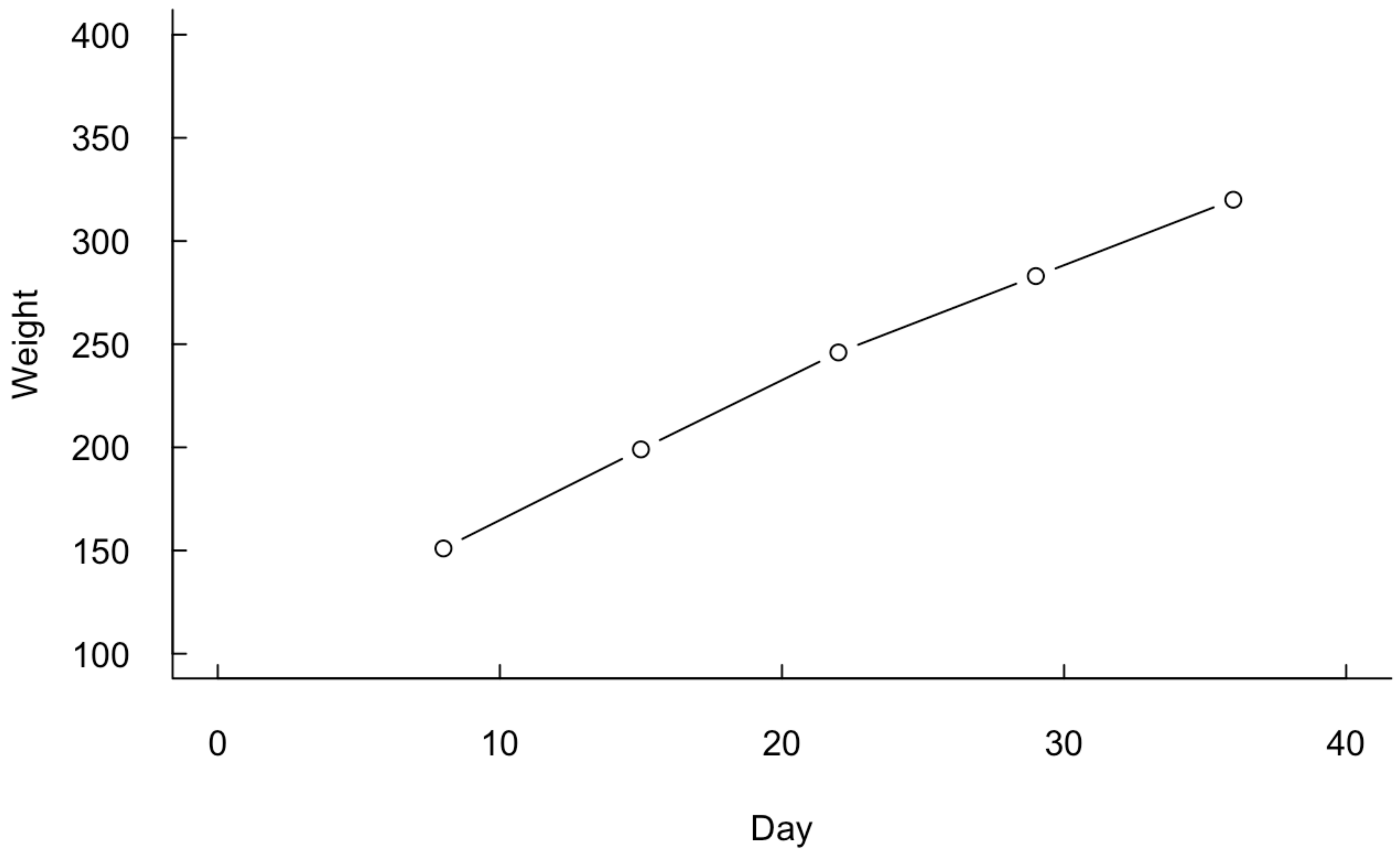
```
Weights <- read.csv("GiantRats.csv",header=T)
head(Weights)
```

```
##   week1 week2 week3 week4 week5
## 1   151   199   246   283   320
## 2   145   199   249   293   354
## 3   147   214   263   312   328
## 4   155   200   237   272   297
## 5   135   188   230   280   323
## 6   159   210   252   298   331
```

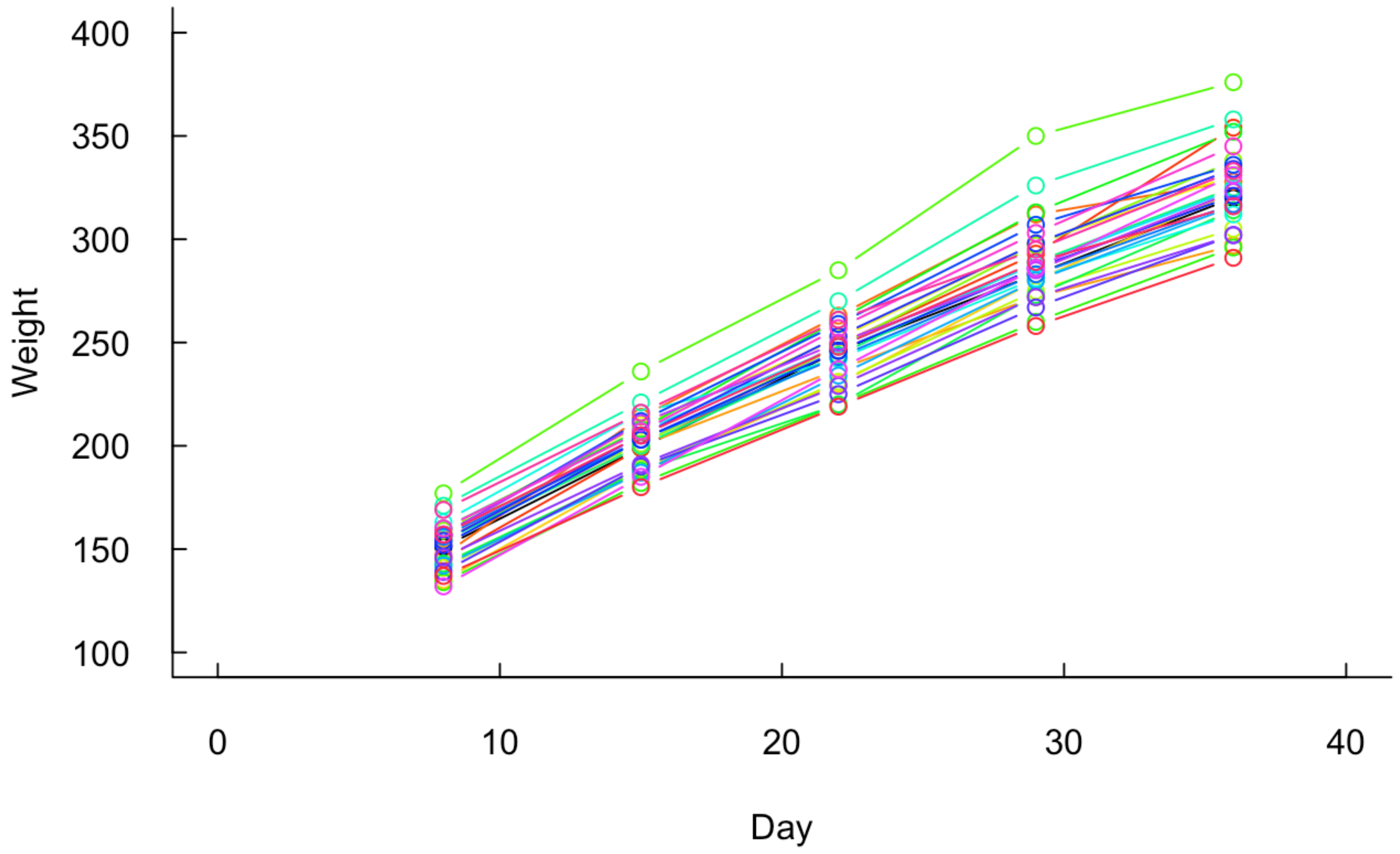
```
data <- list(x = c(8.0, 15.0, 22.0, 29.0, 36.0), xbar = 22, N = 30, T = 5,
             Y = Weights)
```

Here, x is time (days), N is the number of rats, T is the number of time points (5), and Y are the rat weights.

This is a single rat's weight over the 5 week period.



You can see there is some variation for the other rats, but a similar shape



OK, so let's look at the model

We would like to quantify the mean rate of growth of the rats per week, that is, we want to quantify the slope. So, we will fit a simple linear regression.

First is the likelihood and mean model:

none of this model code is runnable here! !

It's located in a separate file "RATS_model.R"

```
model{ #all of this is wrapped in a "model" loop
  # specify likelihood of each data component by looping through all
  # elements of the data (weights, Y), which is read-in as a matrix,
  # so need to loop through each row (rat) and each column (timepoints):
  for(i in 1:N){ #loop across rats
    for(j in 1:T){ #loop across time points (weeks)
      Y[i,j] ~ dnorm(mu[i,j], tau) # this is the likelihood (distribution of observat
ions)
      #we choose a normal liklihood bc assume data are normally distributed.
      mu[i,j] <- alpha + beta*(x[j] - xbar) # this is the regression ("mean model")
    }
  }
  ....
}
```

Next, we have to assign prior distributions to the stochastic parameters

```
....
# Conjugate, relatively non-informative priors to root
  # nodes (population parameters)
  alpha ~ dnorm(0,0.00001)
  beta ~ XXX #what goes here?
  tau ~ dgamm(0.01,0.01) #this is a precision (simply the inverse of the variance)

  sigma <- 1/sqrt(tau) # convert the precision to a standard deviation
}
```

beta ~ ??? --> Please take a minute to think about what this should be.

What is beta?

let's look at the whole model

```
model{
  #likelihood and mean model
  for(i in 1:N){
    for(j in 1:T){
      Y[i,j] ~ dnorm(mu[i,j], tau)
      mu[i,j] <- alpha + beta*(x[j] - xbar)
    }
  }
  #prior distributions
  alpha ~ dnorm(0,0.00001)
  beta ~ dnorm(0,0.00001)
  tau ~ dgamma(0.01, 0.01) #this is called a "confugate prior". Allows gibbs sampling
.
  sigma <- 1/sqrt(tau)
}
```

Before we run the model, let's add a level of hierarchy

Want to quantify growth rates and intercepts for each rat, as well as overall means. Random-intercepts, random-slopes model.

```
model{
  for(i in 1:N){
    for(j in 1:T){
      Y[i,j] ~ dnorm(mu[i,j], tau)
      mu[i,j] <- alpha[i] + beta[i]*(x[j] - xbar) #notice, we now have one intercept
and slope for each rat, i.
    }
  }
  ...
}
```

So what do priors look like now?

```
...
  #prior distributions#
    #hierarchical priors
  for(i in 1:N){
    alpha[i] ~ dnorm(mu.alpha, tau.alpha)
    beta[i] ~ ???
  }
    #root, or global priors
  mu.alpha ~ dnorm(0,0.001)
  mu.beta ~ ???
    #precisions
  tau ~ dgamma(0.01,0.01)
  tau.alpha ~ ???
  tau.beta ~ ???

  # Calculate standard deviations associated with each
  # precition:
  sigma <- 1/sqrt(tau)
  sigma.alpha <- ???
  sigma.beta <- ???

}
```

Everyone take a minute or two to think about how to fill in the blanks (???)

Any questions so far?

This model (above) will be saved in a .R file called "RATS_Hmodel.R". There is also a "RATS_model.R" which is non-hierarchical, the first version we looked at. We can call either model in as part of the JAGS model run.

Final step before we run the model

Need to provide intial values to the stochastic parameters (anything with a prior distribution). #Which parameters need initial values? Specify initials for root nodes, for 3 MCMC chains as a list of lists:

```
inits = list(
  list(mu.alpha = 1, mu.beta = 0, tau=1,tau.alpha=1,tau.beta=1),
  list(mu.alpha = xx, mu.beta = xx, tau=0.5,tau.alpha=2,tau.beta=0.5),
  list(mu.alpha = xx, mu.beta = xx, tau=2,tau.alpha=0.5,tau.beta=2))
```

```
inits = list(
  list(mu.alpha = 1, mu.beta = 0, tau=1,tau.alpha=1,tau.beta=1),
  list(mu.alpha = 5, mu.beta = 0, tau=0.5,tau.alpha=2,tau.beta=0.5),
  list(mu.alpha = 0.5, mu.beta = 0, tau=2,tau.alpha=0.5,tau.beta=2))
```

Run the hierarchical model

Need to provide the function with data, initial values, number of chains (more on this shortly), and adaptive iterations. #Let's look at how that works first The model will run three parallel MCMC chains. MCMC stands for Markov chain Monte Carlo, ie, the chains proceed at each iteration by random sampling from distributions, followed by acceptance or rejection of proposed samples, and adjustment of the sampling distributions. This is akin to maximum likelihood estimation, except that the sampling algorithms "under the hood" vary according to the model selected. Different programs openBUGS, JAGS, Stan, have different available sampling algorithms. These are, for the most part, not something the user interacts with in any way, that is, this is automated.

Here, we will just define the model and sample it for 10 iterations (Note the n.adapt here is way too short, but this is useful for this example)

```
library(rjags) #this package interfaces between R and JAGS
```

```
## Loading required package: coda
```

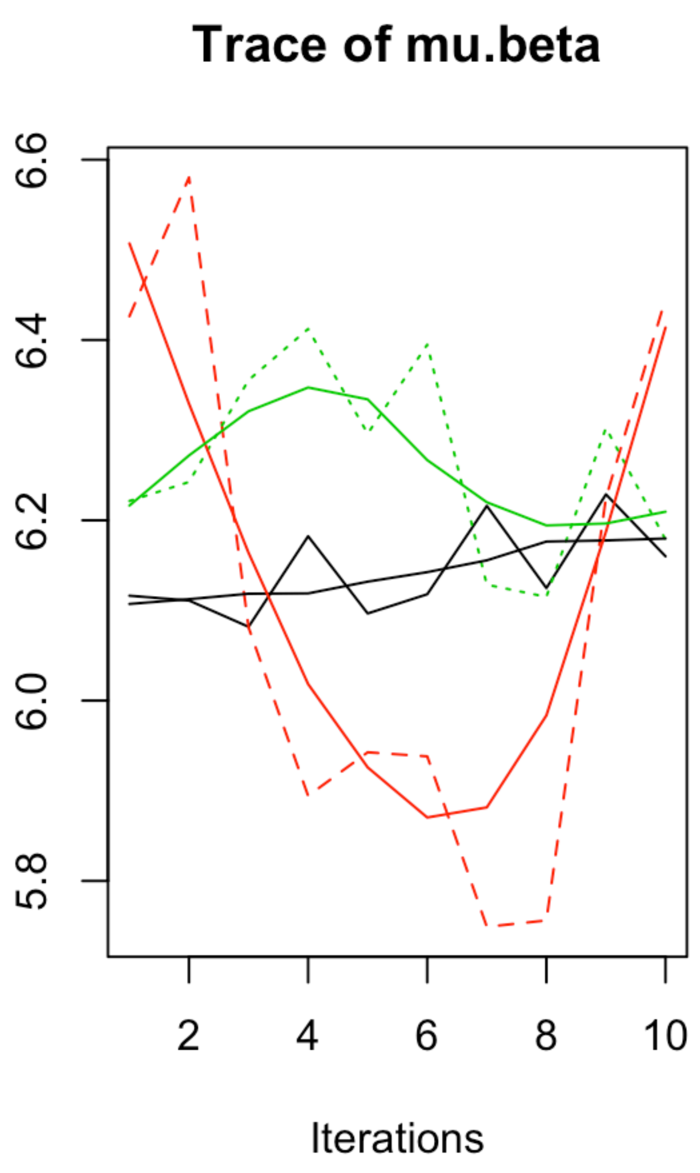
```
## Linked to JAGS 4.2.0
```

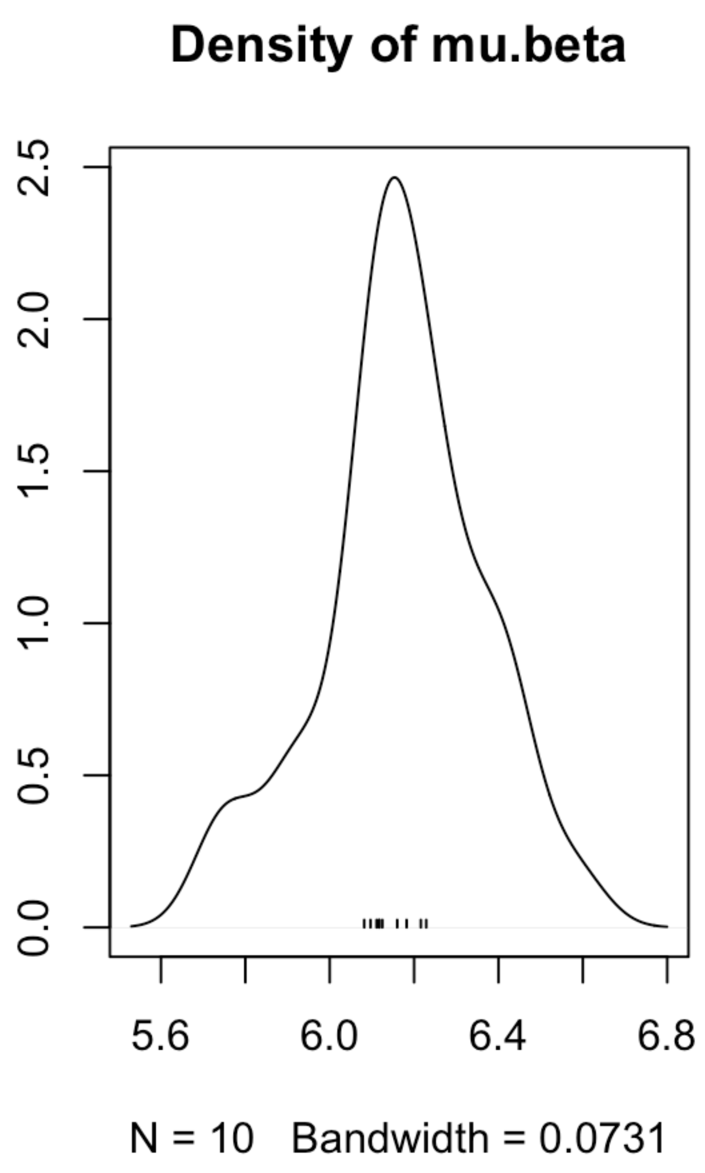
```
## Loaded modules: basemod,bugs
```

```
Rats_model <- jags.model("RATS_Hmodel.R", data=data, inits=inits, n.chains = 3, n.ada
pt=1)
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 150
##    Unobserved stochastic nodes: 65
##    Total graph size: 550
##
## Initializing model
```

```
burnin_samples <- coda.samples(Rats_model,variable.names = c("mu.beta"),n.iter=10)
plot(burnin_samples)
```





#This model has not converged, yet. What you see on the left is the value of the "mu.beta" parameter for each of three chains (black, red, green) at each iteration up to 10 along the x-axis. This is a trace plot, or history plot, and shows how the chains are moving through parameter space.

On the right is the density plot, a histogram which shows the distribution of samples for mu.beta.

Two things show the model is not yet converged: 1) Depending on which iterations along the x-axis you look at, the mean of all three is different.This is shown by a wobbly density plot. 2) Different chains are doing different things. They don't agree. This is why we start the three chains in different places.

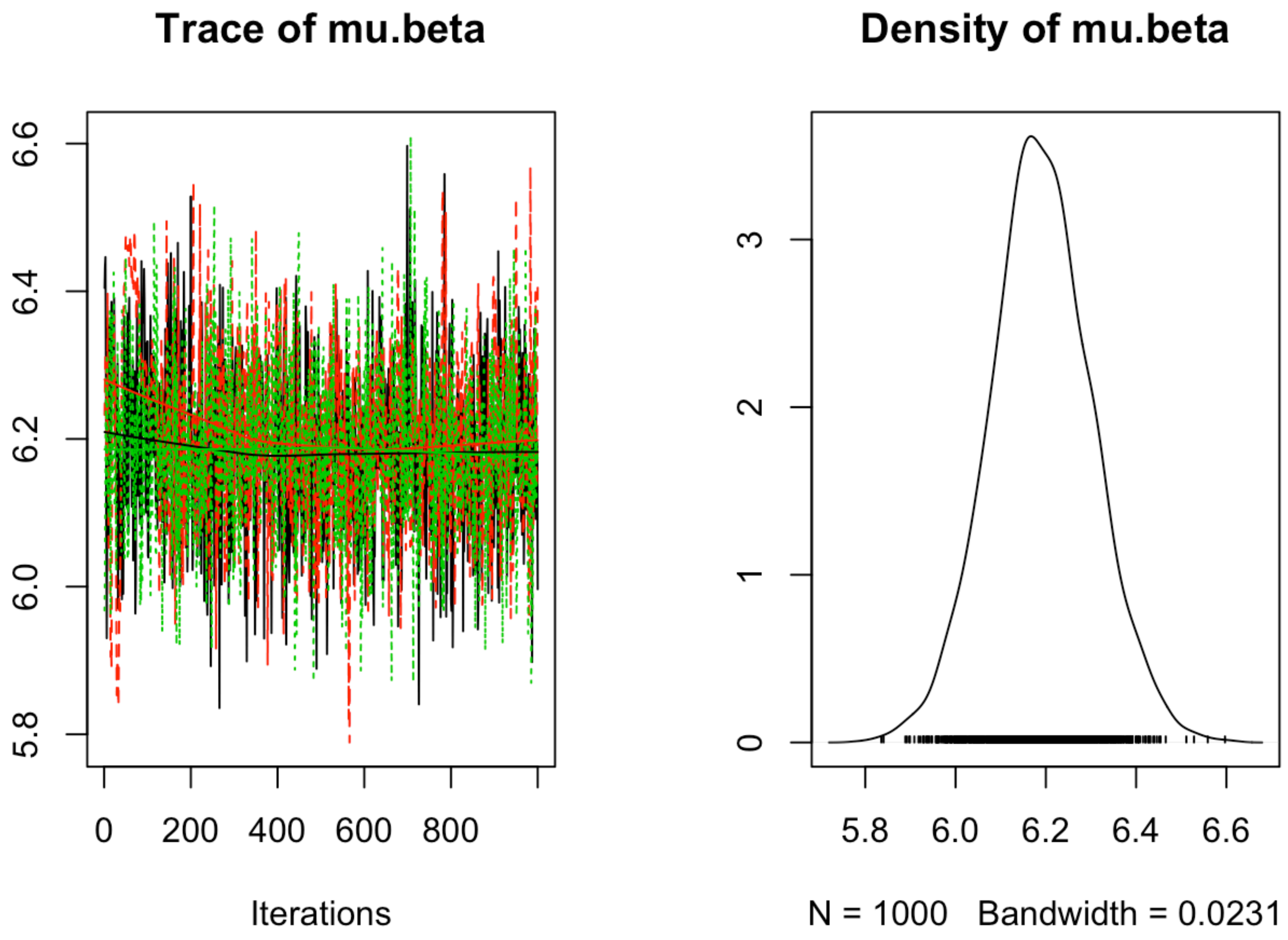
Let's run the model longer:

(Note the much more realistic n.adapt=1000)

```
Rats_model <- jags.model("RATS_Hmodel.R", data=data, inits=inits, n.chains = 3, n.ada
pt=500)
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 150
##    Unobserved stochastic nodes: 65
##    Total graph size: 550
##
## Initializing model
```

```
converged_samples <- coda.samples(Rats_model,variable.names = c("mu.beta"),n.iter=100
0)
plot(converged_samples) #base R method to plot output. Flexible but less user-friendl
y.
```



#this model is now converged. We ran many more samples, but you'll notice that the mean is stable across iterations, and the chains agree on the mean (and uncertainty).

Let's sample the rest of the parameters, and let's monitor the deviance.

```
load.module("dic") #loads the DIC module
```

```
## module dic loaded
```

```
full_coda <- coda.samples(Rats_model, variable.names = c("deviance", "mu.alpha", "mu.
beta","alpha","beta","sig","sig.alpha","sig.beta"), n.iter=1000)
library(mcmcplots) #another way to plot output, more user-friendly, but inflexible. A
nnoying for large models.
mcmcplot(full_coda)
```

```
##

Preparing plots for deviance.  2% complete.
```

```
##

Preparing plots for mu.alpha.  3% complete.
```

```
##

Preparing plots for mu.beta.  5% complete.
```

```
##

Preparing plots for sig.  6% complete.
```

```
##

Preparing plots for sig.alpha.  8% complete.
```

```
##

Preparing plots for sig.beta.  9% complete.
```

```
##

Preparing plots for alpha.  11% complete.
```

```
##

Preparing plots for alpha.  12% complete.
```

```
##

Preparing plots for alpha.  14% complete.
```

```
##

Preparing plots for alpha.  15% complete.
```

```
##

Preparing plots for alpha.  17% complete.
```

```
##

Preparing plots for alpha.  18% complete.
```

```
##

Preparing plots for alpha.  20% complete.
```

```
##

Preparing plots for alpha.  21% complete.
```

```
##

Preparing plots for alpha.  23% complete.
```

```
##

Preparing plots for alpha.  24% complete.
```

```
##

Preparing plots for alpha.  26% complete.
```

```
##

Preparing plots for alpha.  27% complete.
```

```
##

Preparing plots for alpha.  29% complete.
```

```
##

Preparing plots for alpha.  30% complete.
```

```
##

Preparing plots for alpha.  32% complete.
```

```
##

Preparing plots for alpha.  33% complete.
```

```
##

Preparing plots for alpha.  35% complete.
```

```
##

Preparing plots for alpha.  36% complete.
```

```
##

Preparing plots for alpha.  38% complete.
```

```
##

Preparing plots for alpha.  39% complete.
```

```
##

Preparing plots for alpha.  41% complete.
```

```
##

Preparing plots for alpha.  42% complete.
```

```
##

Preparing plots for alpha.  44% complete.
```

```
##

Preparing plots for alpha.  45% complete.
```

```
##

Preparing plots for alpha.  47% complete.
```

```
##

Preparing plots for alpha.  48% complete.
```

```
##

Preparing plots for alpha.  50% complete.
```

```
##

Preparing plots for alpha.  52% complete.
```

```
##

Preparing plots for alpha.  53% complete.
```

```
##

Preparing plots for alpha.  55% complete.
```

```
##

Preparing plots for beta.  56% complete.
```

```
##

Preparing plots for beta.  58% complete.
```

```
##

Preparing plots for beta.  59% complete.
```

```
##

Preparing plots for beta.  61% complete.
```

```
##

Preparing plots for beta.  62% complete.
```

```
##

Preparing plots for beta.  64% complete.
```

```
##

Preparing plots for beta.  65% complete.
```

```
##

Preparing plots for beta.  67% complete.
```

```
##

Preparing plots for beta.  68% complete.
```

```
##

Preparing plots for beta.  70% complete.
```

```
##

Preparing plots for beta.  71% complete.
```

```
##

Preparing plots for beta.  73% complete.
```

```
##

Preparing plots for beta.  74% complete.
```

```
##

Preparing plots for beta.  76% complete.
```

```
##

Preparing plots for beta.  77% complete.
```

```
##

Preparing plots for beta.  79% complete.
```

```
##

Preparing plots for beta.  80% complete.
```

```
##

Preparing plots for beta.  82% complete.
```

```
##

Preparing plots for beta.  83% complete.
```

```
##

Preparing plots for beta.  85% complete.
```

```
##

Preparing plots for beta.  86% complete.
```

```
##

Preparing plots for beta.  88% complete.
```

```
##

Preparing plots for beta.  89% complete.
```

```
##

Preparing plots for beta.  91% complete.
```

```
##

Preparing plots for beta.  92% complete.
```

```
##

Preparing plots for beta.  94% complete.
```

```
##

Preparing plots for beta.  95% complete.
```

```
##

Preparing plots for beta.  97% complete.
```

```
##

Preparing plots for beta.  98% complete.
```

```
##

Preparing plots for beta.  100% complete.
```

This is the bgr (brooks-gelman-rubin) convergence diagnostic

Want to be less than 1.2. But, doesn't work so well with large, complex models. Need to visually evaluate!!

```
gelman.diag(full_coda)
```

```
## Potential scale reduction factors:
##
```

```
##           Point est. Upper C.I.
## alpha[1]           1       1.00
## alpha[2]           1       1.00
## alpha[3]           1       1.00
## alpha[4]           1       1.00
## alpha[5]           1       1.00
## alpha[6]           1       1.00
## alpha[7]           1       1.00
## alpha[8]           1       1.00
## alpha[9]           1       1.00
## alpha[10]          1       1.00
## alpha[11]          1       1.00
## alpha[12]          1       1.01
## alpha[13]          1       1.00
## alpha[14]          1       1.00
## alpha[15]          1       1.00
## alpha[16]          1       1.00
## alpha[17]          1       1.01
## alpha[18]          1       1.00
## alpha[19]          1       1.00
## alpha[20]          1       1.00
## alpha[21]          1       1.00
## alpha[22]          1       1.00
## alpha[23]          1       1.00
## alpha[24]          1       1.00
## alpha[25]          1       1.00
## alpha[26]          1       1.00
## alpha[27]          1       1.00
## alpha[28]          1       1.00
## alpha[29]          1       1.01
## alpha[30]          1       1.00
## beta[1]            1       1.01
## beta[2]            1       1.00
## beta[3]            1       1.00
## beta[4]            1       1.00
## beta[5]            1       1.00
## beta[6]            1       1.01
## beta[7]            1       1.00
## beta[8]            1       1.00
## beta[9]            1       1.00
## beta[10]           1       1.00
## beta[11]           1       1.01
## beta[12]           1       1.01
## beta[13]           1       1.00
## beta[14]           1       1.00
## beta[15]           1       1.00
## beta[16]           1       1.01
## beta[17]           1       1.00
## beta[18]           1       1.00
## beta[19]           1       1.00
## beta[20]           1        1.01
## beta[21]           1        1.00
## beta[22]           1        1.00
## beta[23]           1        1.00
## beta[24]           1        1.00
## beta[25]           1        1.00
## beta[26]           1        1.01
## beta[27]           1        1.00
## beta[28]           1        1.00
## beta[29]           1        1.01
## beta[30]           1        1.00
## deviance           1        1.00
## mu.alpha           1        1.01
## mu.beta            1        1.00
## sig                1        1.00
## sig.alpha          1        1.00
## sig.beta           1        1.00
##
## Multivariate psrf
##
## 1.02
```

Summarize our model output and get posterior stats (means, sds, 95% CI)

Note that this only summarizes parameters that you monitored (with the coda.samples() function above). If you forgot to monitor something.... need to start over with jags.model() step.

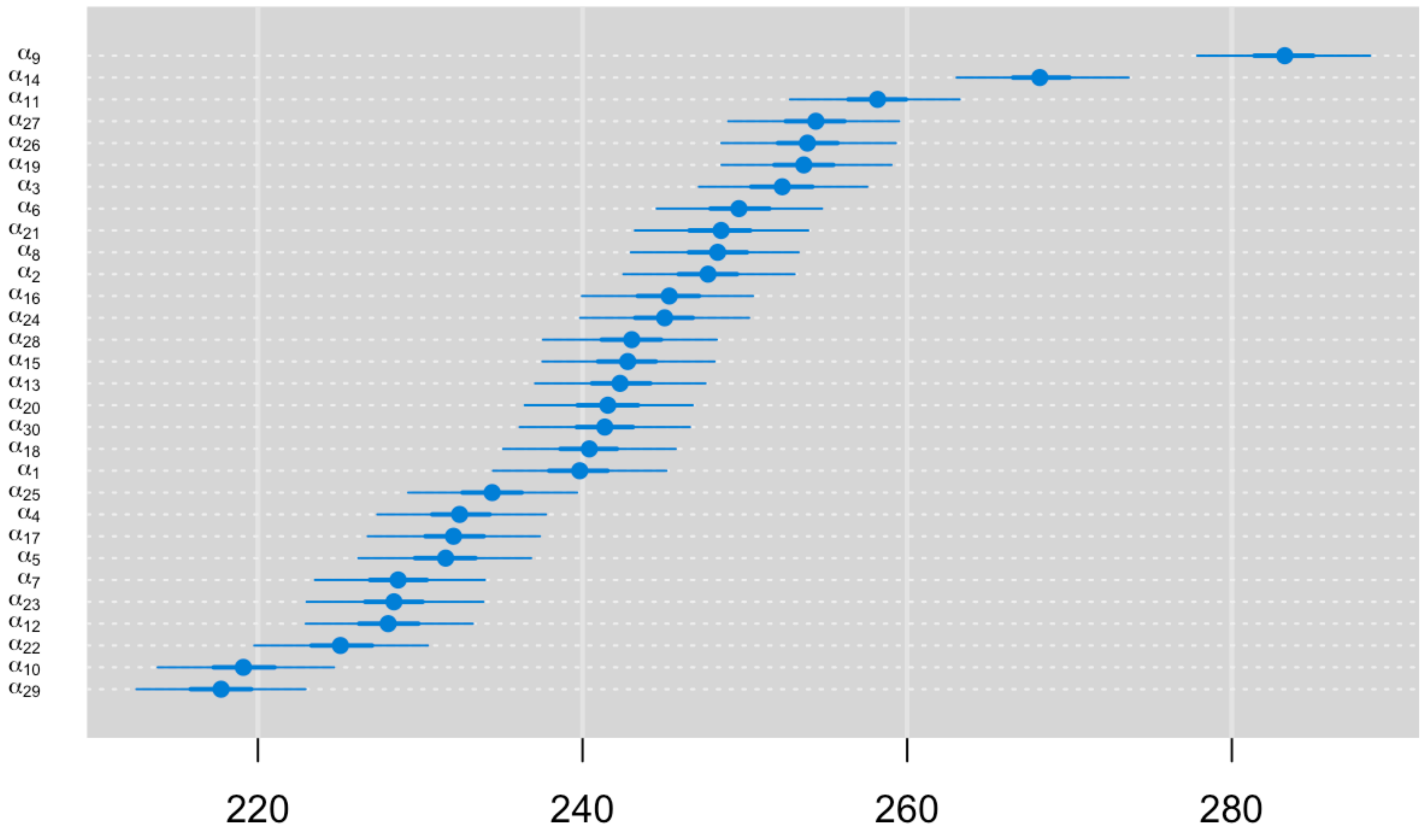
```
table1=summary(full_coda)$stat
table2=summary(full_coda)$quantiles
outstats<-cbind(table2[,1],table1[,2],table2[,1],table2[,5])
colnames(outstats)<-c("mean","sd","val2.5pc","val97.5pc")
outstats
```

```
##                  mean         sd    val2.5pc   val97.5pc
## alpha[1]  234.4840720  2.66669983 234.4840720 245.1701141
## alpha[2]  242.5141366  2.68975854 242.5141366 253.0666868
## alpha[3]  247.1678715  2.71015986 247.1678715 257.5411989
## alpha[4]  227.3559988  2.65769114 227.3559988 237.7408118
## alpha[5]  226.1804099  2.72307897 226.1804099 236.8294139
## alpha[6]  244.5626186  2.65540744 244.5626186 254.7679843
## alpha[7]  223.5125206  2.65446374 223.5125206 233.9869094
## alpha[8]  242.9720849  2.68150185 242.9720849 253.3134459
## alpha[9]  277.8630068  2.72410755 277.8630068 288.5243161
## alpha[10] 213.8270624  2.72629295 213.8270624 224.6972398
## alpha[11] 252.7864364  2.68800484 252.7864364 263.2308455
## alpha[12] 222.9461499  2.68456875 222.9461499 233.2413824
## alpha[13] 237.0759368  2.70783434 237.0759368 247.5810407
## alpha[14] 263.0450383  2.68789488 263.0450383 273.6464219
## alpha[15] 237.5260388  2.68665396 237.5260388 248.1323196
## alpha[16] 239.9607663  2.69953582 239.9607663 250.4987685
## alpha[17] 226.7645178  2.67472601 226.7645178 237.3784586
## alpha[18] 235.1066124  2.68877269 235.1066124 245.7390551
## alpha[19] 248.5603640  2.69478968 248.5603640 259.0287976
## alpha[20] 236.4270793  2.72090227 236.4270793 246.7932853
## alpha[21] 243.2229048  2.71906895 243.2229048 253.9111586
## alpha[22] 219.7718044  2.71235200 219.7718044 230.4694443
## alpha[23] 222.9908155  2.68687347 222.9908155 233.8935357
## alpha[24] 239.8480563  2.66814263 239.8480563 250.2584261
## alpha[25] 229.2587017  2.68173050 229.2587017 239.6689408
## alpha[26] 248.5540097  2.71094632 248.5540097 259.3022553
## alpha[27] 248.9791440  2.65439655 248.9791440 259.5061148
## alpha[28] 237.5616677  2.71067953 237.5616677 248.2675715
## alpha[29] 212.5171285  2.71307862 212.5171285 222.9343925
## alpha[30] 236.1216576  2.67122552 236.1216576 246.6180031
## beta[1]     5.6192845  0.23738358   5.6192845   6.5431156
## beta[2]     6.5544004  0.25084268   6.5544004   7.5413863
## beta[3]     6.0088845  0.24203954   6.0088845   6.9579860
## beta[4]     4.8340610  0.25440794   4.8340610   5.8299706
## beta[5]     6.1004583  0.23922618   6.1004583   7.0273990
## beta[6]     5.6895226  0.24132877   5.6895226   6.6474454
## beta[7]     5.4757919  0.24970419   5.4757919   6.4626282
## beta[8]     5.9405147  0.24257065   5.9405147   6.8823685
## beta[9]     6.5334007  0.25977521   6.5334007   7.5655272
## beta[10]    5.3620812  0.23713470   5.3620812   6.3037455
## beta[11]    6.3075677  0.24942549   6.3075677   7.2951849
## beta[12]    5.6492791  0.24789974   5.6492791   6.6097200
## beta[13]    5.6705691  0.24533918   5.6705691   6.6394358
## beta[14]    6.2125064  0.24456379   6.2125064   7.1750957
## beta[15]    4.9203773  0.25255891   4.9203773   5.9152577
## beta[16]    5.4418199  0.23612650   5.4418199   6.3903376
## beta[17]    5.8078086  0.24275844   5.8078086   6.7544273
## beta[18]    5.3516707  0.24656402   5.3516707   6.3227056
## beta[19]    5.9417635  0.23998518   5.9417635   6.8833683
## beta[20]    5.5636222  0.24395252   5.5636222   6.5375372
## beta[21]    5.9252588  0.23856103   5.9252588   6.8661789
## beta[22]    5.3684131  0.24859277   5.3684131   6.3683297
## beta[23]    5.2443845  0.24918437   5.2443845   6.2423029
## beta[24]    5.4016633  0.24702075   5.4016633   6.3704200
## beta[25]    6.4396816  0.24636765   6.4396816   7.4049458
## beta[26]    6.0680786  0.24858305   6.0680786   7.0316027
## beta[27]    5.4148977  0.23879148   5.4148977   6.3602492
## beta[28]    5.3782609  0.24135177   5.3782609   6.3196762
## beta[29]    5.1958257  0.24327252   5.1958257   6.1525283
## beta[30]    5.6509616  0.24378369   5.6509616   6.6181485
## deviance  941.6814157 14.69105443 941.6814157 998.6431775
## mu.alpha  234.9584366  2.81352570 234.9584366 246.1969752
## mu.beta      5.9859087  0.10609261   5.9859087   6.3969180
## sig          5.2577023  0.46755566   5.2577023   7.0774646
## sig.alpha   11.1944887  2.09419421  11.1944887  19.3329990
## sig.beta     0.3636014  0.08985325   0.3636014   0.7086419
```

Compare parameter values

Let's look at mean values for the intercept (mean weight)

```
library(mcmcplots)
caterplot(full_coda, parms="alpha", quantiles=list(outer=c(0.025,0.975),inner=c(.25,.
75)), greek=T)
```


α_9
α_{14}
α_{11}
α_{27}
α_{26}
α_{19}
α_3
α_6
α_{21}
α_8
α_2
α_{16}
α_{24}
α_{28}
α_{15}
α_{13}
α_{20}
α_{30}
α_{18}
α_1
α_{25}
α_4
α_{17}
α_5
α_7
α_{23}
α_{12}
α_{22}
α_{10}
α_{29}