Exam 3

Nate Hawkins

4/6/2021

This is a take-home exam. The exam is due on Tuesday, April 13, at 9:30 am. Please complete the exam using a Markdown file. Please change the **author** field in the .Rmd file from "Stat 451" to your name. Please email your completed exam to gwf@byu.edu and to dteuscher.37.12@gmail.com prior to the 9:30 am deadline. Please email both your Mardkown file (.Rmd) and your output file. You may use whatever format is most convenient for your output document.

Use the following convention to name your files (assuming you chose .html as your output format): 'lastname_exam3.Rmd' and 'lastname_exam3.html' where lastname is replaced with your last name. Make sure you show your code as well as your answers.

Take-home exams should be your own work. However, you are welcome to use class notes, class videos, and help documentation publicly available for all the programs we have used. You should not search the web for similar problems which someone else may have solved. You should not discuss the exam with any living person.
The data for the exam are in data files that I will email to you.

You will be given pretty explicit directions on number of iterations, burnin, seed, etc. so my answers will match yours. Please follow those directions carefully.

For the first set of problems we will be using the data file **dyes.dat**. These data concern variation in batches of material being dyed. The file contains rows and five columns. Each row is a different batch with five repeated measurements taken on that batch. Thus, there are six batches, with five measurements taken on each batch. You may reorder the data in the file if you find it useful.

1. What is the sample variance for all the data in the data file. That is, what is the overall sample variance for the 30 different measurements.

The sample variance is 3,972

2. Write code in Stan to get draws from the posterior distributions of the two variances: (1) the within batch variance, σ^2_{error} and (2) the batch to batch variance σ^2_{batch}. You may assume the likelihood for the data is normal. Use a normal prior with a mean of 1500 and a standard deviation of 1000 for the overall mean. Remember that Stan by default works with standard deviations. Use gamma(shape=2,rate=.05) as priors for both the standard deviations. What is the posterior mean of σ^2_{batch} (ie, the batch variance, not standard deviation). Use a seed of 1234, iter of 10500, warmup of 500, 4 chains, a thin of 2, and an adapt_delta of 0.99 so our output will match.

The posterior mean of s2batch is 3,481.9

```r
library(rstan)

model <- "

data {
  int <lower = 1> N;
  real meas[N];
  int q;
  int batch[N];
}

parameters {
  real <lower = 0> serror;
  real <lower = 0> sbatch;
  real <lower = 0> mu[q];
}


model {
  serror ~ gamma(2, 0.05);
  sbatch ~ gamma(2, 0.05);
  mu ~ normal(1500, 1000);
  for(i in 1:N){
    meas[i] ~ normal(mu, serror);
    meas[i] ~ normal(mu[batch[i]], sbatch);
  }
}

generated quantities {
  real s2error;
  real s2batch;
  s2error = serror*serror;
  s2batch = sbatch*sbatch;
}
"


writeLines(model, 'dyes1.stan')

meas = new.dyes$meas
N <- 30
q <- 6
batch <- new.dyes$batch
fit1.dat <- list(N=N, meas = meas, q=q, batch = batch)
fit1 <- stan(file = "dyes1.stan", data = fit1.dat, iter = 10500, seed = 1234,
             control = list(adapt_delta = 0.99),
            warmup = 500, chains = 4, thin = 2)
```

```
Inference for Stan model: dyes1.
4 chains, each with iter=10500; warmup=500; thin=2;
post-warmup draws per chain=5000, total post-warmup draws=20000.

           mean se_mean     sd    2.5%     25%     50%     75%   97.5% n_eff
serror    63.37    0.02   3.42   57.05   61.00   63.24   65.57   70.50 18817
sbatch    58.46    0.06   8.01   45.03   52.78   57.70   63.32   76.48 17423
mu[1]   1523.60    0.08  10.58 1503.39 1516.49 1523.64 1530.62 1544.40 18983
mu[2]   1527.51    0.08  10.59 1506.85 1520.24 1527.48 1534.64 1548.16 17969
mu[3]   1533.58    0.08  10.63 1512.79 1526.44 1533.54 1540.74 1554.34 19024
mu[4]   1522.54    0.08  10.60 1501.82 1515.36 1522.55 1529.62 1543.34 18397
mu[5]   1539.86    0.08  10.92 1518.31 1532.48 1539.91 1547.21 1561.31 18643
mu[6]   1517.71    0.08  10.95 1496.31 1510.36 1517.73 1525.04 1539.02 18875
s2error 4026.93    3.19 436.77 3254.31 3720.80 3999.91 4299.04 4969.71 18759
s2batch 3481.89    7.49 981.29 2028.12 2785.60 3328.91 4008.86 5848.75 17178
lp__    -920.06    0.02   2.04 -924.89 -921.19 -919.74 -918.57 -917.07 15492
        Rhat
serror     1
sbatch     1
mu[1]      1
mu[2]      1
mu[3]      1
mu[4]      1
mu[5]      1
mu[6]      1
s2error    1
s2batch    1
lp__       1

Samples were drawn using NUTS(diag_e) at Mon Apr 12 10:10:22 2021.
For each parameter, n_eff is a crude measure of effective sample size,
and Rhat is the potential scale reduction factor on split chains (at
convergence, Rhat=1).
```

3. In case you couldn't get your Stan code to work, write the R code you would use to check the Raftery-Louis diagnostic for the chains of your parameters. If you actually have chains, your code will produce the R-L diagnostic.

```r
library(coda)

sims <- as.matrix(fit1)
chains <- as.mcmc(sims)
chains <- as.matrix(chains)
raftery.diag(chains)
```

```
Quantile (q) = 0.025
Accuracy (r) = +/- 0.005
```

```
Probability (s) = 0.95

         Burn-in  Total Lower bound  Dependence
         (M)      (N)   (Nmin)       factor (I)
 serror  3        4012  3746         1.07
 sbatch  2        3818  3746         1.02
 mu[1]   3        4012  3746         1.07
 mu[2]   3        4112  3746         1.10
 mu[3]   3        4249  3746         1.13
 mu[4]   3        4095  3746         1.09
 mu[5]   3        4061  3746         1.08
 mu[6]   2        3945  3746         1.05
 s2error 3        4012  3746         1.07
 s2batch 2        3818  3746         1.02
 lp__    3        4061  3746         1.08
```

4. Write the R code to get the effective sample sizes for the posterior chains of the parameters. If you actually have the chains, your code will produce the effective sample sizes.

```
effectiveSize(chains)

   serror   sbatch    mu[1]    mu[2]    mu[3]    mu[4]    mu[5]    mu[6]
18769.24 17769.65 18951.58 19023.21 18991.19 18816.76 17244.43 18831.72
  s2error  s2batch     lp__
18709.50 17560.39 15603.40
```

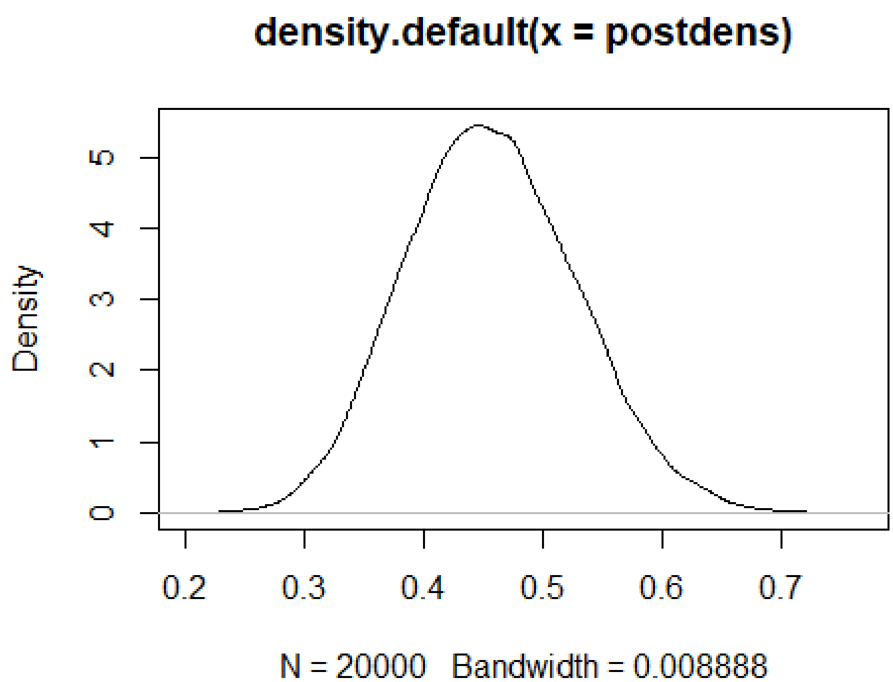
5. If you could actually produce the diagnostics, you will notice that the R-L diagnostic for σ^2_{batch} and σ_{batch} are higher than the R-L diagnostics for σ^2_{error} and σ_{error}. In fact, the batch standard deviation and variance R-L diagnostics are borderline unacceptable. Why do you think the estimates involving the error of replication within a batch are easier to estimate than the batch to batch variability?

The estimates of variation between batches are more difficult to estimate because the batch to batch variability is an estimate of the differences between groups. This creates correlation between observations because they don't have as much independence. The within batch variance is easier because it is measured on single values, not esimates of groups.

6. In statistics, we often are interested in something called the Intraclass correlation or ICC. For this problem, the ICC would be calculated as $\sigma^2_{batch}/(\sigma^2_{batch} + \sigma^2_{error})$. Using your chains, plot the estimated posterior density of the ICC for this problem.

```
colnames(chains)

  [1] "serror"  "sbatch"  "mu[1]"   "mu[2]"   "mu[3]"   "mu[4]"   "mu[5]"
  [8] "mu[6]"   "s2error" "s2batch" "lp__"

postdens <- NA
for(i in 1:nrow(chains)){
  postdens[i] <- chains[i,10]/(chains[i,10]+chains[i,9])
}
```

```
plot(density(postdens))
```

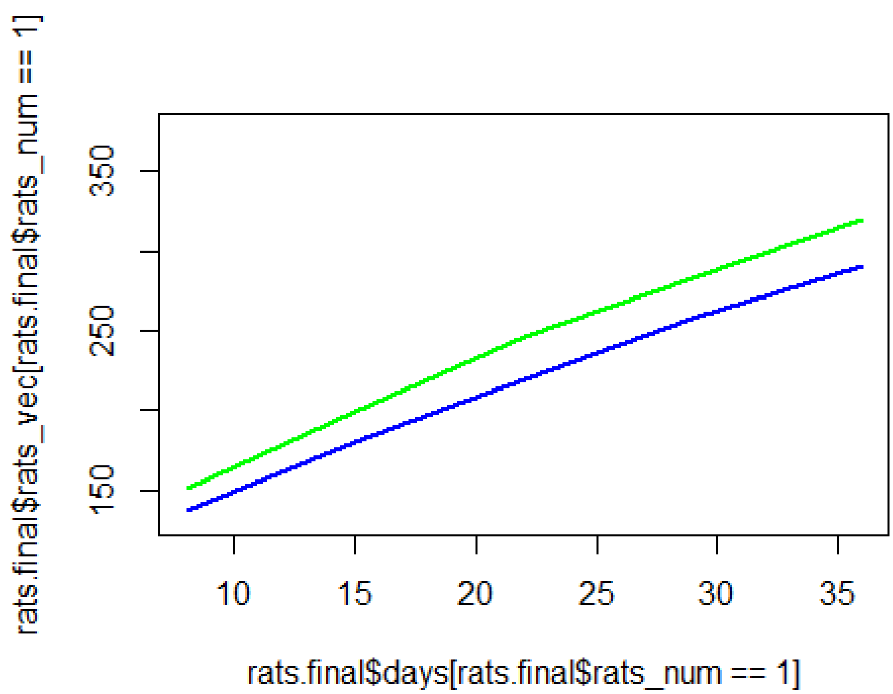


The next set of problems will use a data set that shows the growth of rats. The data file is called **rats.dat** and contains five columns, the weight of each animal at day 8, 15, 22, 29, and 36. Use JAGS as the modeling software for this group of problems.

7. Read in the data and plot the growth of rat1 and rat29 on the same axes.

```
rats <- read.table("rats.dat", header = TRUE)
rats$wt8. <- as.numeric(substr(rats$wt8.,0,3))
rats$wt15. <- as.numeric(substr(rats$wt15.,0,3))
rats$wt22. <- as.numeric(substr(rats$wt22.,0,3))
rats$wt29. <- as.numeric(substr(rats$wt29.,0,3))
rats$wt36. <- as.numeric(substr(rats$wt36,0,3))



dim(rats)

 [1] 30  6

rat_num <- 1:30
rats <- cbind(rats,rat_num)

rats_vec <- c(rats$wt8., rats$wt15., rats$wt22., rats$wt29., rats$wt36.)
```

```
rats_num <- rep(rat_num, 5)
days <- rep(c(8,15,22,29,36), each = 30)
rats.final <- as.data.frame(cbind(rats_vec, rats_num, days))

plot(rats.final$days[rats.final$rats_num == 1],
rats.final$rats_vec[rats.final$rats_num == 1], col = "green", type = 'l', lwd
= 2, ylim = c(min(rats.final$rats_vec),max(rats.final$rats_vec)))
lines(rats.final$days[rats.final$rats_num ==
29],rats.final$rats_vec[rats.final$rats_num == 29], col = "blue", lwd = 2)
```



8. Hopefully, you can see that the growth is fairly linear, but differs from rat to rat. Write the JAGS code to estimate the growth as a linear regression with no regard for the different rats. That is, ignore the fact that the data points represent 30 different animals, and treat all the data as independent. Use priors of normal(mean=0,precision=.0001) for both β_0 and β_1. Use a gamma(shape=2,rate=.01) as the prior for σ^2_{error}. Use the same number of iterations, burnin, thin, chains, and seed that you used in problem 2. What is DIC for this model? (Note: you may find it easier to do this problem if you rearrange the data.)

DIC is 1263.8

```
set.seed(1234)
library(R2jags)

mdl <- "

model {
```

```
  for(i in 1:150){
    weight[i] ~ dnorm(mu[i], 1/s2error)
    mu[i] <- beta0 + beta1*days[i]
  }

  # Priors

  beta0 ~ dnorm(0, 0.0001)
  beta1 ~ dnorm(0, 0.0001)
  s2error ~ dgamma(2, 0.01)

}
"

writeLines(mdl, 'fit2.txt')

weight = rats.final$rats_vec
days = rats.final$days

data.jags <- c('weight', 'days')
parms <- c('beta0' , 'beta1', 's2error')

fit2 <- jags(data= data.jags, parameters.to.save = parms,
                  model.file = 'fit2.txt', inits = NULL,
                  n.iter = 10500, n.thin = 2, n.burnin = 500, jags.seed =
1234,
                  n.chains = 4)

 module glm loaded

 Compiling model graph
    Resolving undeclared variables
    Allocating nodes
 Graph information:
    Observed stochastic nodes: 150
    Unobserved stochastic nodes: 3
    Total graph size: 319

 Initializing model

(fit2)

 Inference for Bugs model at "fit2.txt", fit using jags,
  4 chains, each with 10500 iterations (first 500 discarded), n.thin = 2
  n.sims = 20000 iterations saved
          mu.vect sd.vect    2.5%     25%     50%     75%   97.5%  Rhat
 beta0    106.477   3.211 100.213 104.310 106.470 108.635 112.815 1.001
 beta1      6.189   0.134   5.926   6.099   6.190   6.279   6.451 1.001
 s2error  261.843  30.318 208.704 240.339 259.395 280.935 327.454 1.001
```

```
deviance 1260.905   2.421 1258.137 1259.132 1260.294 1261.988 1267.038 1.001
         n.eff
beta0    17000
beta1    13000
s2error   8300
deviance 20000

For each parameter, n.eff is a crude measure of effective sample size,
and Rhat is the potential scale reduction factor (at convergence, Rhat=1).

DIC info (using the rule, pD = var(deviance)/2)
pD = 2.9 and DIC = 1263.8
DIC is an estimate of expected predictive error (lower deviance is better).
```

Raftery-Lewis Diagnostics and effective sample sizes look good.

```
sims <- as.mcmc(fit2)
chains <- as.matrix(sims)
sims <- as.mcmc(chains)
raftery.diag(sims)


Quantile (q) = 0.025
Accuracy (r) = +/- 0.005
Probability (s) = 0.95

          Burn-in  Total Lower bound  Dependence
          (M)      (N)   (Nmin)       factor (I)
 beta0    2        3771  3746         1.010
 beta1    2        3680  3746         0.982
 deviance 2        3725  3746         0.994
 s2error  3        4223  3746         1.130

effectiveSize(sims)

   beta0    beta1 deviance  s2error
20000.00 20000.00 17157.06 16690.14
```

9. Now please make this into a hierarchical model for the intercepts. That is, you are considering the rats to be a random sample of all possible rats, and you are drawing the individual rat intercepts from a normal population with mean $\mu_{intercepts}$ and variance $\sigma^2_{intercepts}$. Use a normal prior for $\mu_{ntercepts}$ with mean 0 and precision .0001. Use a gamma prior for $\sigma^2_{intercepts}$ with shape 2 and rate .01. Use the same prior as in problem 8 for β_1. Use a gamma prior for σ^2_{error} with shape 2 and rate .05. Use the same number of iterations, burnin, thin, chains, and seed that you used in problem 8. What is the DIC for this model?

The DIC for this model is 1098.0

```
set.seed(1234)
library(R2jags)

mdl <- "

model {

  for(i in 1:150){
    weight[i] ~ dnorm(mu[i], 1/s2error)
    mu[i] <- beta0[rat[i]] + beta1*days[i]
  }

  for(i in 1:30){
      beta0[i] ~ dnorm(muint, s2int)
  }

  # Priors
  s2int ~ dgamma(2, 0.01)
  muint ~ dnorm(0,0.0001)

 s2error ~ dgamma(2,0.05)
 beta1 ~ dnorm(0, 0.0001)

}
"

writeLines(mdl, 'fit3.txt')

weight = rats.final$rats_vec
days = rats.final$days
rat = rats.final$rats_num

data.jags <- c('weight', 'days', 'rat')
parms <- c('beta0' , 'beta1', 's2error', 'muint','s2int')

fit3 <- jags(data= data.jags, parameters.to.save = parms,
                  model.file = 'fit3.txt', inits = NULL,
                  n.iter = 10500, n.thin = 2, n.burnin = 500, jags.seed =
1234,
                  n.chains = 4)

 Compiling model graph
    Resolving undeclared variables
    Allocating nodes
 Graph information:
    Observed stochastic nodes: 150
    Unobserved stochastic nodes: 34
    Total graph size: 646
```

```
 Initializing model

(fit3)

 Inference for Bugs model at "fit3.txt", fit using jags,
  4 chains, each with 10500 iterations (first 500 discarded), n.thin = 2
  n.sims = 20000 iterations saved
           mu.vect sd.vect    2.5%     25%     50%     75%   97.5%
Rhat
 beta0[1]   103.929   3.811   96.458  101.371  103.932  106.493  111.375
1.001
 beta0[2]   111.477   3.834  104.001  108.879  111.490  114.046  119.049
1.001
 beta0[3]   115.920   3.845  108.518  113.327  115.883  118.511  123.504
1.001
 beta0[4]    96.873   3.819   89.366   94.328   96.900   99.427  104.352
1.001
 beta0[5]    95.950   3.841   88.448   93.374   95.952   98.534  103.563
1.001
 beta0[6]   113.333   3.830  105.734  110.789  113.359  115.907  120.789
1.001
 beta0[7]    93.133   3.845   85.644   90.574   93.099   95.725  100.671
1.001
 beta0[8]   112.028   3.823  104.537  109.477  112.039  114.580  119.544
1.001
 beta0[9]   145.553   3.946  137.718  142.911  145.580  148.231  153.185
1.001
 beta0[10]   84.052   3.881   76.366   81.457   84.054   86.686   91.641
1.001
 beta0[11]  121.483   3.845  113.929  118.890  121.502  124.054  128.956
1.001
 beta0[12]   92.619   3.860   84.996   90.005   92.585   95.214  100.269
1.001
 beta0[13]  106.258   3.834   98.765  103.701  106.237  108.843  113.776
1.001
 beta0[14]  131.090   3.866  123.434  128.513  131.089  133.679  138.660
1.001
 beta0[15]  106.646   3.826   99.060  104.101  106.639  109.193  114.071
1.001
 beta0[16]  109.021   3.800  101.581  106.476  109.015  111.588  116.528
1.001
 beta0[17]   96.451   3.857   88.881   93.837   96.501   99.068  103.906
1.001
 beta0[18]  104.423   3.839   96.974  101.851  104.392  106.966  112.117
1.001
 beta0[19]  117.224   3.855  109.664  114.653  117.228  119.809  124.721
1.001
 beta0[20]  105.511   3.825   97.961  102.927  105.549  108.098  113.010
1.001
```

```
 beta0[21]  112.201   3.849  104.641  109.594  112.182  114.787  119.803
1.001
 beta0[22]   89.805   3.884   82.127   87.232   89.795   92.423   97.465
1.001
 beta0[23]   92.944   3.829   85.396   90.381   92.977   95.504  100.534
1.001
 beta0[24]  108.914   3.832  101.406  106.328  108.911  111.501  116.368
1.001
 beta0[25]   98.694   3.799   91.263   96.148   98.704  101.264  106.136
1.001
 beta0[26]  117.413   3.813  109.876  114.822  117.421  119.997  124.848
1.001
 beta0[27]  117.786   3.803  110.417  115.210  117.744  120.364  125.366
1.001
 beta0[28]  106.816   3.808   99.376  104.259  106.799  109.381  114.202
1.001
 beta0[29]   82.724   3.890   75.050   80.090   82.739   85.285   90.366
1.001
 beta0[30]  105.366   3.831   97.846  102.782  105.347  107.934  112.970
1.001
 beta1        6.188   0.068    6.056    6.142    6.187    6.233    6.322
1.001
 muint      106.482   2.946  100.650  104.501  106.484  108.442  112.135
1.001
 s2error     67.095   8.476   52.556   61.149   66.419   72.411   85.808
1.001
 s2int        0.006   0.002    0.003    0.005    0.006    0.007    0.010
1.001
 deviance  1058.226   8.922 1042.917 1051.867 1057.363 1063.801 1078.038
1.001
           n.eff
 beta0[1]  20000
 beta0[2]  20000
 beta0[3]  20000
 beta0[4]  16000
 beta0[5]  20000
 beta0[6]  20000
 beta0[7]  20000
 beta0[8]  20000
 beta0[9]  20000
 beta0[10] 11000
 beta0[11] 15000
 beta0[12] 20000
 beta0[13] 20000
 beta0[14] 20000
 beta0[15] 20000
 beta0[16] 20000
 beta0[17] 18000
 beta0[18] 17000
 beta0[19] 20000
```

```
beta0[20] 20000
beta0[21]  7100
beta0[22] 17000
beta0[23] 20000
beta0[24] 20000
beta0[25] 11000
beta0[26]  7700
beta0[27] 20000
beta0[28] 20000
beta0[29] 18000
beta0[30]  9600
beta1     20000
muint     20000
s2error   19000
s2int     20000
deviance  13000

For each parameter, n.eff is a crude measure of effective sample size,
and Rhat is the potential scale reduction factor (at convergence, Rhat=1).

DIC info (using the rule, pD = var(deviance)/2)
pD = 39.8 and DIC = 1098.0
DIC is an estimate of expected predictive error (lower deviance is better).
```

10. Now adapt what you did in problem 9 to make a hierarchical model for slopes as well as intercepts. Use a normal prior for μ_{slopes} with mean 0 and precision .0001. Use a gamma prior for σ^2_{slopes} with shape 1.1 and rate 1. Use the same control parameters as in problem 9. What is the DIC for this model?

DIC is 1066.1

```
set.seed(1234)

mdl4 <- "
  model {

  for (i in 1:150){
    weight[i] ~ dnorm(mu[i], 1/vv)
    mu[i] <- b0[rat[i]] + b1[rat[i]]*days[i]
  }

  for(i in 1:30){
      b0[i] ~ dnorm(mub0, 1/vvint)
      b1[i] ~ dnorm(mub1, 1/vvslp)
  }

  vvint ~ dgamma(2, 0.01)
  mub0 ~ dnorm(0, 0.0001)
```

```
  vvslp ~ dgamma(1.1, 1)
  mub1 ~ dnorm(0, .00001)

  vv ~ dgamma(2, 0.05)
  }

"


writeLines(mdl4, 'fit4.txt')

weight = rats.final$rats_vec
days = rats.final$days
rat = rats.final$rats_num

data.jags <- c('weight', 'days', 'rat')
parms <- c('b0', 'b1', 'vv', 'mub0', 'vvint', 'mub1', 'vvslp')

fit4 <- jags(data= data.jags, parameters.to.save = parms,
                  model.file = 'fit4.txt', inits = NULL,
                  n.iter = 10500, n.thin = 2, n.burnin = 500, jags.seed =
1234,
                  n.chains = 4)

 Compiling model graph
    Resolving undeclared variables
    Allocating nodes
 Graph information:
    Observed stochastic nodes: 150
    Unobserved stochastic nodes: 65
    Total graph size: 826

 Initializing model

fit4

 Inference for Bugs model at "fit4.txt", fit using jags,
  4 chains, each with 10500 iterations (first 500 discarded), n.thin = 2
  n.sims = 20000 iterations saved
          mu.vect sd.vect    2.5%     25%     50%     75%   97.5%  Rhat n.eff
 b0[1]    106.426   5.263  96.131 102.894 106.445 109.936 116.817 1.001  5100
 b0[2]     95.875   5.453  85.209  92.137  95.859  99.600 106.480 1.001 13000
 b0[3]    109.224   5.271  98.908 105.710 109.239 112.745 119.558 1.001  7600
 b0[4]    112.877   5.443 102.275 109.239 112.877 116.553 123.561 1.001  5300
 b0[5]     91.434   5.465  80.566  87.753  91.488  95.103 102.013 1.001  7900
 b0[6]    112.196   5.289 101.825 108.576 112.183 115.766 122.630 1.001  4300
 b0[7]     99.146   5.321  88.611  95.588  99.177 102.699 109.588 1.001  4700
 b0[8]    107.155   5.288  96.825 103.555 107.118 110.761 117.430 1.002  3900
 b0[9]    124.062   5.617 112.997 120.290 124.012 127.803 135.209 1.001  7500
 b0[10]    93.939   5.455  83.237  90.285  93.988  97.663 104.518 1.003  1200
 b0[11]   108.465   5.301  98.200 104.899 108.433 111.962 118.963 1.001 20000
```

b0[12]	96.291	5.371	85.774	92.626	96.349	99.971	106.696	1.001	9100
b0[13]	106.847	5.296	96.317	103.319	106.860	110.405	117.286	1.001	17000
b0[14]	118.208	5.466	107.479	114.583	118.166	121.841	128.955	1.001	13000
b0[15]	119.644	5.539	108.818	115.900	119.604	123.347	130.679	1.002	1800
b0[16]	113.053	5.342	102.602	109.472	113.025	116.602	123.490	1.001	6400
b0[17]	96.805	5.282	86.357	93.251	96.828	100.395	107.161	1.001	8700
b0[18]	110.588	5.253	100.316	107.033	110.528	114.101	121.012	1.001	4200
b0[19]	111.685	5.294	101.288	108.152	111.683	115.220	122.068	1.002	2000
b0[20]	107.950	5.278	97.550	104.434	107.967	111.481	118.295	1.002	2400
b0[21]	107.537	5.344	96.901	103.991	107.521	111.118	117.967	1.001	20000
b0[22]	98.473	5.366	87.902	94.864	98.540	102.117	108.899	1.002	2900
b0[23]	102.813	5.292	92.411	99.270	102.814	106.386	113.144	1.002	3500
b0[24]	113.600	5.312	103.176	110.045	113.536	117.135	124.118	1.001	6100
b0[25]	88.036	5.648	76.986	84.235	88.059	91.855	99.107	1.001	4900
b0[26]	109.357	5.318	98.836	105.843	109.371	112.933	119.735	1.001	9400
b0[27]	120.546	5.454	109.904	116.863	120.512	124.183	131.379	1.001	20000
b0[28]	112.579	5.294	102.167	109.060	112.564	116.097	122.869	1.001	20000
b0[29]	95.877	5.369	85.096	92.305	95.916	99.463	106.394	1.002	3600
b0[30]	106.493	5.277	96.087	102.984	106.485	109.979	116.886	1.001	6700
b1[1]	6.062	0.220	5.624	5.915	6.063	6.211	6.493	1.002	3900
b1[2]	6.946	0.228	6.497	6.792	6.947	7.101	7.391	1.001	6800
b1[3]	6.518	0.220	6.085	6.370	6.517	6.665	6.949	1.001	9700
b1[4]	5.405	0.229	4.953	5.253	5.402	5.559	5.855	1.001	5800
b1[5]	6.400	0.228	5.955	6.245	6.398	6.555	6.846	1.001	20000
b1[6]	6.245	0.222	5.811	6.095	6.247	6.394	6.682	1.002	3500
b1[7]	5.888	0.223	5.452	5.740	5.888	6.037	6.324	1.001	4800
b1[8]	6.428	0.221	5.996	6.278	6.426	6.578	6.867	1.001	4600
b1[9]	7.254	0.236	6.789	7.096	7.256	7.413	7.716	1.001	7700
b1[10]	5.696	0.228	5.255	5.541	5.695	5.850	6.140	1.003	1300
b1[11]	6.826	0.222	6.386	6.676	6.827	6.973	7.263	1.001	20000
b1[12]	6.001	0.224	5.572	5.847	5.999	6.152	6.446	1.001	13000
b1[13]	6.162	0.221	5.728	6.014	6.162	6.310	6.597	1.001	20000
b1[14]	6.827	0.228	6.380	6.676	6.826	6.981	7.278	1.001	7200
b1[15]	5.558	0.231	5.101	5.404	5.559	5.713	6.013	1.002	1900
b1[16]	5.996	0.223	5.562	5.846	5.997	6.145	6.434	1.001	6100
b1[17]	6.166	0.221	5.732	6.020	6.167	6.315	6.602	1.001	5100
b1[18]	5.888	0.219	5.458	5.740	5.889	6.036	6.317	1.001	6600
b1[19]	6.463	0.221	6.030	6.313	6.461	6.612	6.896	1.002	1800
b1[20]	6.070	0.221	5.639	5.923	6.070	6.216	6.509	1.002	2100
b1[21]	6.418	0.224	5.978	6.268	6.418	6.569	6.857	1.001	18000
b1[22]	5.759	0.223	5.327	5.607	5.756	5.910	6.195	1.002	2300
b1[23]	5.700	0.222	5.265	5.552	5.701	5.848	6.136	1.002	4200
b1[24]	5.962	0.222	5.517	5.816	5.964	6.111	6.394	1.002	3800
b1[25]	6.701	0.234	6.240	6.545	6.700	6.859	7.158	1.001	4700
b1[26]	6.583	0.222	6.149	6.433	6.583	6.732	7.023	1.001	11000
b1[27]	6.062	0.228	5.614	5.910	6.064	6.215	6.509	1.001	20000
b1[28]	5.910	0.222	5.480	5.762	5.909	6.058	6.345	1.001	20000
b1[29]	5.538	0.225	5.101	5.386	5.538	5.689	5.983	1.002	4100
b1[30]	6.132	0.221	5.701	5.984	6.132	6.281	6.562	1.001	11000
mub0	106.524	2.374	101.813	104.972	106.523	108.067	111.192	1.001	20000

```
mub1       6.184   0.109   5.967   6.111   6.184   6.256   6.400 1.001 20000
vv        37.607   5.636  28.037  33.609  37.075  41.039  50.038 1.002  3300
vvint    123.468  42.791  58.775  93.106 117.022 146.574 224.960 1.001  4200
vvslp      0.285   0.099   0.141   0.215   0.268   0.336   0.522 1.002  3600
deviance 968.027  14.010 942.968 958.231 967.211 977.056 997.750 1.002  3300

For each parameter, n.eff is a crude measure of effective sample size,
and Rhat is the potential scale reduction factor (at convergence, Rhat=1).

DIC info (using the rule, pD = var(deviance)/2)
pD = 98.1 and DIC = 1066.1
DIC is an estimate of expected predictive error (lower deviance is better).
```

11. Which of the models in 8, 9, and 10 would you prefer? Why?

I prefer the complete hierarchical model from question 10. It has the best DIC and is the best for inference.

12. Regardless of your answer in number 11, rerun the model in problem 10. Make sure that all parameters are being saved. Compute the Raftery-Louis diagnostic for all the parameters. Are there any R-L diagnostics that we should be concerned about?

All of the R-L values are below 3, some are close but aren't too concerning. Effective size is also good.

```
set.seed(1234)

mdl4 <- "
  model {

  for (i in 1:150){
    weight[i] ~ dnorm(mu[i], 1/vv)
    mu[i] <- b0[rat[i]] + b1[rat[i]]*days[i]
  }

  for(i in 1:30){
      b0[i] ~ dnorm(mub0, 1/vvint)
      b1[i] ~ dnorm(mub1, 1/vvslp)
  }

  vvint ~ dgamma(2, 0.05)
  mub0 ~ dnorm(0, 0.0001)

  vvslp ~ dgamma(1.1, 1)
  mub1 ~ dnorm(0, .00001)

  vv ~ dgamma(1.1, 0.5)
  }

"
```

```
writeLines(mdl4, 'fit4.txt')

weight = rats.final$rats_vec
days = rats.final$days
rat = rats.final$rats_num

data.jags <- c('weight', 'days', 'rat')
parms <- c('b0', 'b1', 'vv', 'mub0', 'vvint', 'mub1', 'vvslp')

fit4 <- jags(data= data.jags, parameters.to.save = parms,
                  model.file = 'fit4.txt', inits = NULL,
                  n.iter = 10500, n.thin = 2, n.burnin = 500, jags.seed =
1234,
                  n.chains = 4)

 Compiling model graph
    Resolving undeclared variables
    Allocating nodes
 Graph information:
    Observed stochastic nodes: 150
    Unobserved stochastic nodes: 65
    Total graph size: 826

 Initializing model

(fit4)

 Inference for Bugs model at "fit4.txt", fit using jags,
  4 chains, each with 10500 iterations (first 500 discarded), n.thin = 2
  n.sims = 20000 iterations saved
         mu.vect sd.vect    2.5%     25%     50%     75%   97.5%  Rhat n.eff
 b0[1]   106.547   4.710  97.264 103.388 106.551 109.721 115.780 1.001  7800
 b0[2]    95.389   4.958  85.736  92.069  95.420  98.726 105.187 1.002  2200
 b0[3]   108.903   4.715  99.686 105.717 108.936 112.076 118.131 1.002  2600
 b0[4]   113.740   4.790 104.341 110.479 113.753 116.973 123.142 1.001 20000
 b0[5]    91.299   4.927  81.643  88.002  91.260  94.562 101.048 1.001  4900
 b0[6]   112.271   4.675 103.147 109.126 112.225 115.323 121.524 1.001 20000
 b0[7]    99.534   4.766  90.280  96.324  99.558 102.726 108.950 1.001 20000
 b0[8]   106.999   4.714  97.762 103.823 106.993 110.166 116.232 1.002  3800
 b0[9]   122.700   4.936 112.959 119.413 122.708 126.022 132.336 1.001  5000
 b0[10]   94.452   4.826  85.007  91.212  94.419  97.736 104.024 1.001 12000
 b0[11]  107.740   4.818  98.341 104.528 107.730 110.957 117.141 1.002  3600
 b0[12]   96.562   4.801  87.128  93.347  96.534  99.788 106.057 1.002  1700
 b0[13]  106.710   4.734  97.388 103.518 106.724 109.855 116.115 1.001  5400
 b0[14]  117.373   4.805 107.924 114.149 117.359 120.637 126.792 1.001  7700
 b0[15]  120.275   4.909 110.627 117.025 120.298 123.550 129.907 1.002  4100
 b0[16]  113.113   4.729 103.873 109.891 113.123 116.310 122.402 1.001 20000
 b0[17]   97.083   4.753  87.871  93.832  97.072 100.311 106.410 1.001  6000
 b0[18]  110.940   4.686 101.698 107.798 110.956 114.101 120.172 1.001 20000
```

```
b0[19]   111.333   4.723 102.040 108.160 111.377 114.531 120.594 1.001  6300
b0[20]   108.143   4.723  98.949 104.998 108.161 111.289 117.486 1.001  5900
b0[21]   107.335   4.685  98.106 104.135 107.350 110.525 116.380 1.002  3100
b0[22]    99.002   4.755  89.768  95.786  98.951 102.203 108.437 1.001 10000
b0[23]   103.288   4.757  94.005 100.041 103.269 106.523 112.575 1.001  5600
b0[24]   113.664   4.759 104.356 110.461 113.658 116.878 122.950 1.001  4500
b0[25]    87.521   5.001  77.781  84.168  87.530  90.852  97.491 1.002  3800
b0[26]   108.955   4.789  99.599 105.746 108.970 112.151 118.323 1.003  1300
b0[27]   120.607   4.816 111.259 117.362 120.592 123.829 130.059 1.001 12000
b0[28]   112.734   4.738 103.434 109.541 112.708 115.935 122.040 1.002  2200
b0[29]    96.491   4.819  86.963  93.267  96.450  99.748 105.917 1.001 13000
b0[30]   106.560   4.702  97.217 103.438 106.605 109.709 115.743 1.001  6400
b1[1]      6.057   0.199   5.664   5.924   6.055   6.190   6.448 1.001  9500
b1[2]      6.972   0.208   6.564   6.833   6.972   7.110   7.382 1.002  2100
b1[3]      6.533   0.199   6.142   6.399   6.532   6.665   6.926 1.002  3100
b1[4]      5.365   0.202   4.965   5.229   5.367   5.501   5.760 1.001 18000
b1[5]      6.407   0.207   5.995   6.269   6.409   6.546   6.807 1.001  5800
b1[6]      6.244   0.197   5.855   6.112   6.244   6.374   6.632 1.001 15000
b1[7]      5.870   0.200   5.478   5.735   5.871   6.005   6.258 1.001 20000
b1[8]      6.435   0.198   6.049   6.301   6.433   6.567   6.825 1.001  5000
b1[9]      7.319   0.208   6.912   7.180   7.316   7.459   7.733 1.001  5600
b1[10]     5.671   0.203   5.264   5.536   5.671   5.808   6.072 1.001 13000
b1[11]     6.864   0.202   6.470   6.727   6.863   7.000   7.260 1.001  5400
b1[12]     5.989   0.201   5.592   5.855   5.990   6.125   6.376 1.002  2600
b1[13]     6.166   0.199   5.774   6.033   6.166   6.300   6.556 1.001  6900
b1[14]     6.868   0.203   6.467   6.731   6.868   7.004   7.265 1.001 10000
b1[15]     5.527   0.207   5.120   5.388   5.526   5.666   5.935 1.002  3200
b1[16]     5.992   0.199   5.606   5.857   5.992   6.127   6.385 1.001 20000
b1[17]     6.154   0.200   5.761   6.018   6.154   6.290   6.542 1.001  7300
b1[18]     5.872   0.198   5.485   5.738   5.870   6.004   6.267 1.001 20000
b1[19]     6.478   0.199   6.089   6.343   6.477   6.613   6.870 1.001  7900
b1[20]     6.062   0.198   5.670   5.928   6.062   6.194   6.451 1.001  5400
b1[21]     6.428   0.197   6.042   6.295   6.428   6.563   6.813 1.002  3100
b1[22]     5.733   0.201   5.336   5.599   5.735   5.870   6.121 1.001  9600
b1[23]     5.678   0.199   5.287   5.545   5.678   5.814   6.065 1.001  6000
b1[24]     5.956   0.200   5.565   5.819   5.956   6.091   6.352 1.001  6000
b1[25]     6.725   0.209   6.309   6.587   6.724   6.865   7.130 1.002  4000
b1[26]     6.603   0.202   6.207   6.468   6.603   6.737   6.997 1.003  1500
b1[27]     6.056   0.201   5.664   5.922   6.056   6.191   6.450 1.001 17000
b1[28]     5.901   0.199   5.513   5.765   5.901   6.036   6.290 1.002  2300
b1[29]     5.508   0.203   5.103   5.373   5.509   5.645   5.906 1.001  9000
b1[30]     6.130   0.199   5.740   5.998   6.128   6.263   6.523 1.001  4800
mub0     106.541   2.026 102.543 105.197 106.555 107.873 110.540 1.001 20000
mub1       6.185   0.108   5.972   6.113   6.185   6.256   6.397 1.001 16000
vv        29.176   3.470  23.094  26.733  28.923  31.367  36.715 1.001  6600
vvint     90.589  24.694  50.382  73.060  88.038 105.297 146.468 1.001 13000
vvslp      0.297   0.099   0.154   0.228   0.281   0.347   0.531 1.001  9200
deviance 957.692  12.219 935.717 949.027 957.009 965.644 983.198 1.001  6600

For each parameter, n.eff is a crude measure of effective sample size,
```

```
 and Rhat is the potential scale reduction factor (at convergence, Rhat=1).

 DIC info (using the rule, pD = var(deviance)/2)
 pD = 74.6 and DIC = 1032.3
 DIC is an estimate of expected predictive error (lower deviance is better).
```

```r
sims <- as.mcmc(fit4)
chains <- as.matrix(sims)
sims <- as.mcmc(chains)
raftery.diag(sims)
```

```
 Quantile (q) = 0.025
 Accuracy (r) = +/- 0.005
 Probability (s) = 0.95

           Burn-in  Total Lower bound  Dependence
           (M)      (N)   (Nmin)       factor (I)
  b0[1]    5        5820  3746         1.55
  b0[10]   5        5870  3746         1.57
  b0[11]   6        9042  3746         2.41
  b0[12]   8        9784  3746         2.61
  b0[13]   5        5896  3746         1.57
  b0[14]   5        5459  3746         1.46
  b0[15]   8        9470  3746         2.53
  b0[16]   6        9710  3746         2.59
  b0[17]   6        9214  3746         2.46
  b0[18]   8        9592  3746         2.56
  b0[19]   8        9384  3746         2.51
  b0[2]    5        6130  3746         1.64
  b0[20]   5        5600  3746         1.49
  b0[21]   8        8920  3746         2.38
  b0[22]   8        9712  3746         2.59
  b0[23]   5        5845  3746         1.56
  b0[24]   5        5721  3746         1.53
  b0[25]   8        10062 3746         2.69
  b0[26]   6        9116  3746         2.43
  b0[27]   6        9266  3746         2.47
  b0[28]   8        8720  3746         2.33
  b0[29]   6        8492  3746         2.27
  b0[3]    6        8664  3746         2.31
  b0[30]   6        9040  3746         2.41
  b0[4]    6        8586  3746         2.29
  b0[5]    5        5820  3746         1.55
  b0[6]    5        5505  3746         1.47
  b0[7]    5        5696  3746         1.52
  b0[8]    5        6103  3746         1.63
  b0[9]    8        9280  3746         2.48
  b1[1]    6        6405  3746         1.71
  b1[10]   6        8692  3746         2.32
```

```
  b1[11]   6        9116  3746         2.43
  b1[12]   6        8942  3746         2.39
  b1[13]   6        9002  3746         2.40
  b1[14]   6        6377  3746         1.70
  b1[15]   6        8758  3746         2.34
  b1[16]   6        9090  3746         2.43
  b1[17]   6        9264  3746         2.47
  b1[18]   6        9366  3746         2.50
  b1[19]   5        5436  3746         1.45
  b1[2]    6        9190  3746         2.45
  b1[20]   5        6051  3746         1.62
  b1[21]   6        8128  3746         2.17
  b1[22]   8        9046  3746         2.41
  b1[23]   5        6157  3746         1.64
  b1[24]   6        9314  3746         2.49
  b1[25]   8        9196  3746         2.45
  b1[26]   8        9640  3746         2.57
  b1[27]   6        9102  3746         2.43
  b1[28]   5        6077  3746         1.62
  b1[29]   6        8270  3746         2.21
  b1[3]    5        5413  3746         1.45
  b1[30]   6        8448  3746         2.26
  b1[4]    8        9864  3746         2.63
  b1[5]    5        5998  3746         1.60
  b1[6]    6        9190  3746         2.45
  b1[7]    6        9634  3746         2.57
  b1[8]    5        5820  3746         1.55
  b1[9]    8        9270  3746         2.47
  deviance 6        8624  3746         2.30
  mub0     3        4373  3746         1.17
  mub1     2        3913  3746         1.04
  vv       3        4428  3746         1.18
  vvint    6        8390  3746         2.24
  vvslp    6        8080  3746         2.16
```

```
effectiveSize(sims)
```

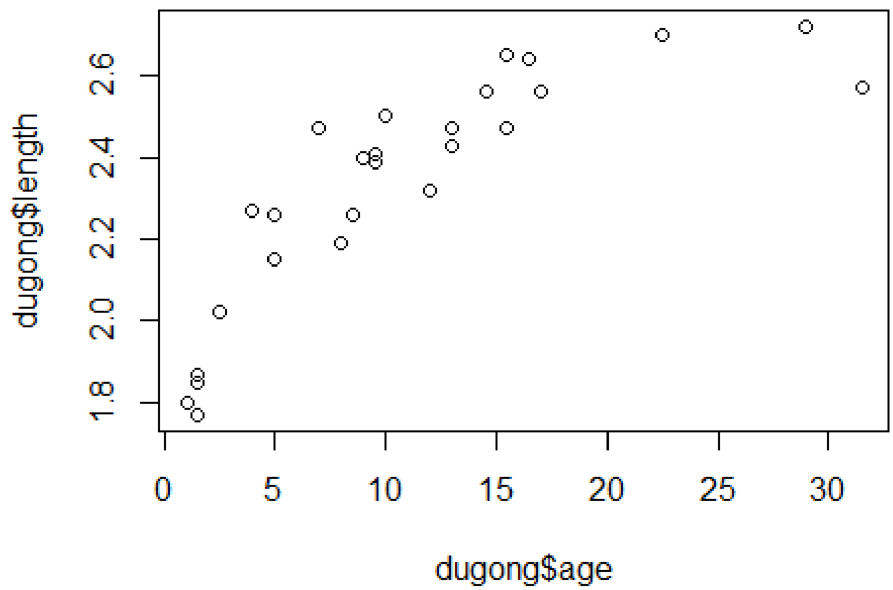
```
     b0[1]    b0[10]    b0[11]    b0[12]    b0[13]    b0[14]    b0[15]
b0[16]
  5466.189  5093.584  5240.858  5346.555  5438.969  5233.957  4646.284
5381.557
    b0[17]    b0[18]    b0[19]     b0[2]    b0[20]    b0[21]    b0[22]
b0[23]
  5145.307  5547.427  5499.401  4908.786  5575.347  5439.207  5340.724
5191.164
    b0[24]    b0[25]    b0[26]    b0[27]    b0[28]    b0[29]     b0[3]
b0[30]
  5246.137  4515.997  5328.565  5230.590  5108.571  5062.985  5463.583
5524.107
     b0[4]     b0[5]     b0[6]     b0[7]     b0[8]     b0[9]     b1[1]
```

```
b1[10]
  5251.947  4990.679  5523.231  5406.980  5555.371  4828.314  5490.613
5178.868
    b1[11]    b1[12]    b1[13]    b1[14]    b1[15]    b1[16]    b1[17]
b1[18]
  5166.014  4875.280  5482.191  5201.444  4805.381  5413.746  5380.925
5484.268
    b1[19]     b1[2]    b1[20]    b1[21]    b1[22]    b1[23]    b1[24]
b1[25]
  5437.671  4901.882  5457.798  5539.374  5375.348  5339.358  5317.545
4468.320
    b1[26]    b1[27]    b1[28]    b1[29]     b1[3]    b1[30]     b1[4]
b1[5]
  5333.690  5270.448  5374.173  5170.412  5303.182  5385.682  5356.538
5063.375
     b1[6]     b1[7]     b1[8]     b1[9]  deviance      mub0      mub1
vv
  5607.994  5304.018  5537.501  4625.677  5748.900  7674.032 11140.096
8228.251
     vvint     vvslp
  5896.695  6968.496
```

For the next set of problems, you will be using the data set **dugong.dat**. These data concern growth of dugongs (sometimes called sea cows, an aquatic mammal found primarily in the Indo-West Pacific). The data file contains two columns, the age of the animal in years, and the length of the animal in meters for 27 individuals. Use JAGS for the models in this section.

13. Read in the data and plot it with age on the x-axis and length on the y-axis.

```
dugong <- read.table("dugong.dat")
head(dugong)

   age length
 1 1.0   1.80
 2 1.5   1.85
 3 1.5   1.87
 4 1.5   1.77
 5 2.5   2.02
 6 4.0   2.27

plot(dugong$age, dugong$length)
```



14. You will note that growth is faster for younger animals, and slows as the animal matures. This type of growth is called nonlinear growth, and the simplest curve to describe such growth is: where y_i represents the length of the animal and x_i represents the age of the animal. As you can see, there are three parameters to estimate, a, b, and g. 'a' represents the asymptote or value at which growth stops, 'b' is constrained to be positive, and 'g' is constrained to be between 0 and 1. Use a normal with mean 3 and precision .01 as the prior for parameter a. Use a gamma with shape 1.1 and rate .1 for parameter b. And use a Uniform(0,1) prior for parameter g. You may assume the likelihood is normal. Use a gamma with shape 2 and rate .1 for the prior for σ^2_{error}. Use the same control parameters as we have used in previous problems. What is the DIC of the model? Use the same number of iterations in problem 8 and 2

-38.4

```
set.seed(1234)
mdl5 <- "
  model {

  for (i in 1:27){
    length[i] ~ dnorm(mu[i], 1/vv)
    mu[i] <- (a - b*g^(age[i]))
  }

  g ~ dunif(0,1)
  b ~ dgamma(1.1, .1)
```

```
  a ~ dnorm(3, 0.01)
  vv ~ dgamma(2, 0.1)

  }

"
writeLines(mdl5, "fit5.txt")
length <- dugong$length
age <- dugong$age

data.jags <- c('length', 'age')
parms <- c('a', 'g', 'b', 'vv')

fit5 <- jags(data= data.jags, parameters.to.save = parms,
                  model.file = 'fit5.txt', inits = NULL,
                  n.iter = 10500, n.thin = 2, n.burnin = 500, jags.seed =
1234,
                  n.chains = 4)

 Compiling model graph
    Resolving undeclared variables
    Allocating nodes
 Graph information:
    Observed stochastic nodes: 27
    Unobserved stochastic nodes: 4
    Total graph size: 126

 Initializing model

fit5

 Inference for Bugs model at "fit5.txt", fit using jags,
  4 chains, each with 10500 iterations (first 500 discarded), n.thin = 2
  n.sims = 20000 iterations saved
          mu.vect sd.vect    2.5%     25%     50%     75%   97.5%  Rhat n.eff
 a          2.650   0.082   2.511   2.596   2.643   2.695   2.829 1.004   720
 b          0.979   0.097   0.810   0.921   0.974   1.030   1.164 1.003  8100
 g          0.858   0.043   0.765   0.840   0.864   0.884   0.919 1.031   920
 vv         0.012   0.005   0.006   0.009   0.011   0.014   0.024 1.001 15000
 deviance -47.692   4.310 -52.824 -50.761 -48.712 -45.721 -36.659 1.002 11000

 For each parameter, n.eff is a crude measure of effective sample size,
 and Rhat is the potential scale reduction factor (at convergence, Rhat=1).

 DIC info (using the rule, pD = var(deviance)/2)
 pD = 9.3 and DIC = -38.4
 DIC is an estimate of expected predictive error (lower deviance is better).

sims <- as.mcmc(fit5)
chains <- as.matrix(sims)
```

```
sims <- as.mcmc(chains)
raftery.diag(sims)


 Quantile (q) = 0.025
 Accuracy (r) = +/- 0.005
 Probability (s) = 0.95

          Burn-in  Total Lower bound  Dependence
          (M)      (N)   (Nmin)       factor (I)
 a        36       37614 3746         10.00
 b        5        5921  3746          1.58
 deviance 2        3929  3746          1.05
 g        40       43200 3746         11.50
 vv       3        4028  3746          1.08

effectiveSize(sims)

        a         b  deviance         g        vv
 1019.0890 1554.9033 1846.8274  762.3757 3734.5988
```

15. What is the Raftery-Louis diagnostic for the asymptote parameter?

R-L for aymptote parameter is 10.00.

16. Do you think you have a problem? Why?

This is defnitiely a problem. This indicates that it's not mixing well and is correlated. We will need to run more draws.

17. Rerun the code from problem 14 with 10 times the number of iterations and burnins, and thin by 10. Now what is the R-L diagnostic for a?

Now the R-L diagnostic is 2.29, this is acceptable.

```
set.seed(1234)
mdl5 <- "
  model {

  for (i in 1:27){
    length[i] ~ dnorm(mu[i], 1/vv)
    mu[i] <- (a - b*g^(age[i]))
  }

  g ~ dunif(0,1)
  b ~ dgamma(1.1, .1)
  a ~ dnorm(3, 0.01)
  vv ~ dgamma(2, 0.1)

  }
```

```
"
writeLines(mdl5, "fit5.txt")
length <- dugong$length
age <- dugong$age

data.jags <- c('length', 'age')
parms <- c('a', 'g', 'b', 'vv')

fit5 <- jags(data= data.jags, parameters.to.save = parms,
                  model.file = 'fit5.txt', inits = NULL,
                  n.iter = 10*10500, n.thin = 10, n.burnin = 10*500,
jags.seed = 1234,
                  n.chains = 4)

 Compiling model graph
    Resolving undeclared variables
    Allocating nodes
 Graph information:
    Observed stochastic nodes: 27
    Unobserved stochastic nodes: 4
    Total graph size: 126

 Initializing model

fit5

 Inference for Bugs model at "fit5.txt", fit using jags,
  4 chains, each with 105000 iterations (first 5000 discarded), n.thin = 10
  n.sims = 40000 iterations saved
          mu.vect sd.vect    2.5%     25%     50%     75%   97.5%  Rhat n.eff
 a          2.651   0.080   2.516   2.597   2.643   2.697   2.832 1.001 11000
 b          0.977   0.087   0.811   0.921   0.975   1.031   1.154 1.001 31000
 g          0.860   0.038   0.770   0.840   0.865   0.885   0.919 1.001 12000
 vv         0.012   0.004   0.006   0.009   0.011   0.014   0.023 1.001 40000
 deviance -47.838   4.015 -52.786 -50.774 -48.735 -45.880 -37.790 1.001 16000

 For each parameter, n.eff is a crude measure of effective sample size,
 and Rhat is the potential scale reduction factor (at convergence, Rhat=1).

 DIC info (using the rule, pD = var(deviance)/2)
 pD = 8.1 and DIC = -39.8
 DIC is an estimate of expected predictive error (lower deviance is better).

sims <- as.mcmc(fit5)
chains <- as.matrix(sims)
sims <- as.mcmc(chains)
raftery.diag(sims)


 Quantile (q) = 0.025
```

```
Accuracy (r) = +/- 0.005
Probability (s) = 0.95

          Burn-in  Total Lower bound  Dependence
          (M)      (N)   (Nmin)       factor (I)
 a        6        8580  3746         2.29
 b        2        3929  3746         1.05
 deviance 1        3755  3746         1.00
 g        8        9596  3746         2.56
 vv       2        3779  3746         1.01
```

```
effectiveSize(sims)
```

```
       a        b deviance        g       vv
10499.97 26599.73 23457.74 11160.11 31480.63
```

18. What is the 95% equal tail posterior probability interval for the asymptote parameter?

The 95% equal tail posterior probability interval for the asymptote parameter is (2.52, 2.83).

```
library(bayestestR)
```

```
Warning: package 'bayestestR' was built under R version 4.0.3
```

```
et95 <- ci(chains[,1], method = "ETI", ci = .95)
et95
```

```
# Equal-Tailed Interval

95% ETI
------------
[2.52, 2.83]
```

19. There is a small posterior probability that the asymptote parameter could be greater than 3. What is that probability?

The probability that the asymptote parameter is greater than 3 is 0.00185

```
colnames(chains)
```

```
[1] "a"        "b"        "deviance" "g"        "vv"
```

```
mean(chains[,1] > 3)
```

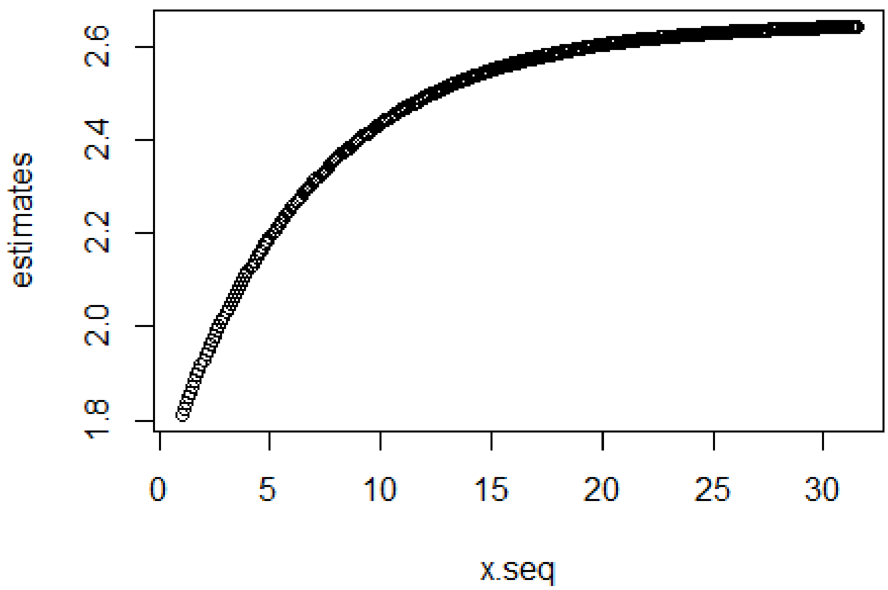
```
[1] 0.00185
```

20. Using the output from the code in problem 17, put a best fit line on your plot of the raw data. Use the posterior means of the parameters for the line.

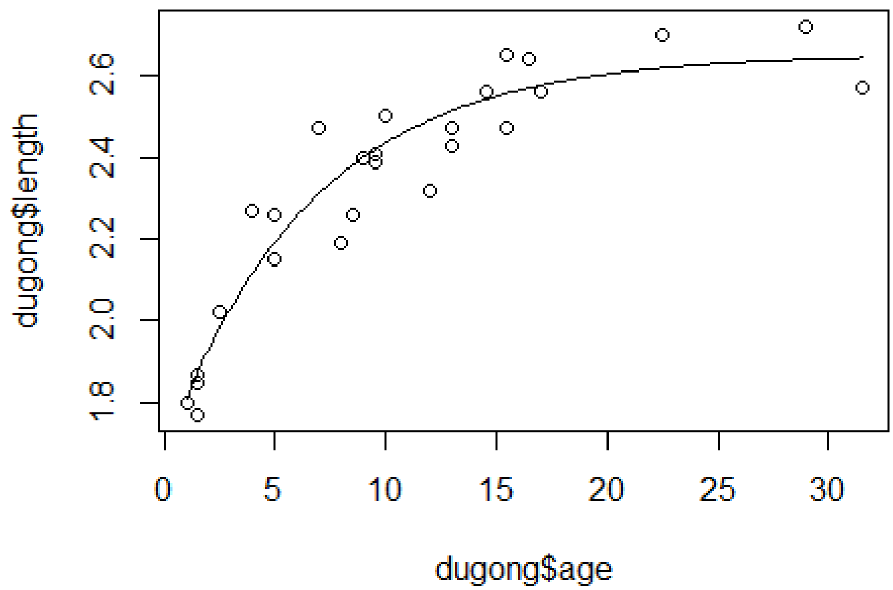
```
colnames(chains)
```

```
[1] "a"        "b"        "deviance" "g"        "vv"
```

```
a <- chains[,1]
b <- chains[,2]
g <- chains[,4]


x.seq <- seq(from = 1, to = 31.5, by = .1)
estimates <- mean(a) - mean(b) * mean(g)^(x.seq)

plot(x.seq, estimates)
```

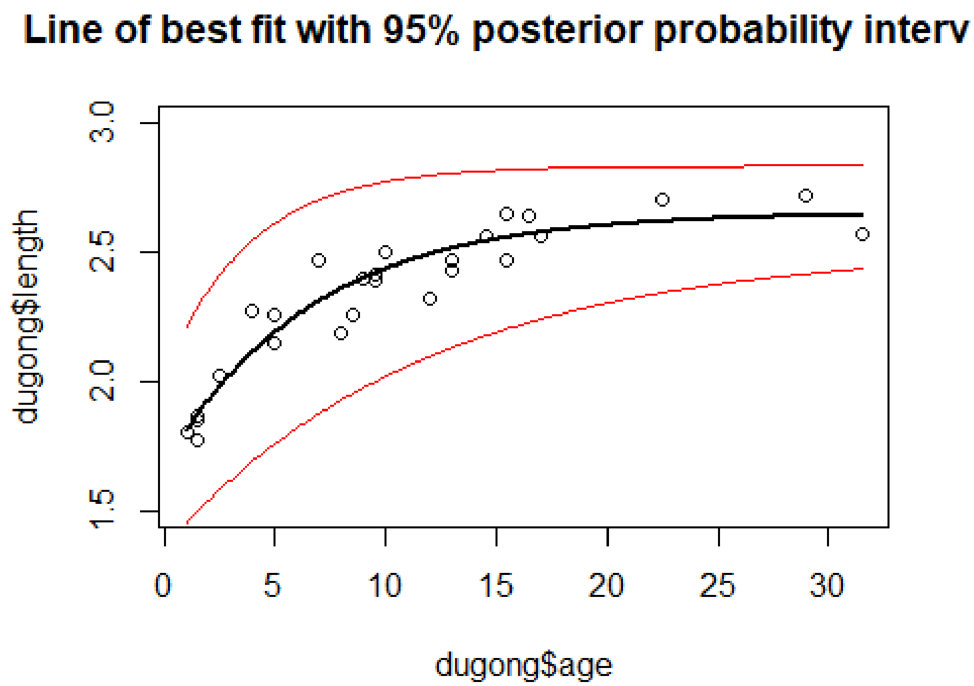



```
plot(dugong$age, dugong$length)
lines(x.seq, estimates)
```



21. Now plot the data points, the best fit line, and 95% posterior probability intervals for the line.

```
quant_a <- quantile(a, probs = c(.025,.975))
quant_b <- quantile(b, probs = c(.025,.975))
quant_g <- quantile(g, probs = c(.025,.975))

estimate_2.5 <-  quant_a[2] - quant_b[1]*quant_g[1]^(x.seq)
estimate_97.5 <-  quant_a[1] - quant_b[2]*quant_g[2]^(x.seq)


plot(dugong$age, dugong$length, ylim = c(1.5,3), main = "Line of best fit
with 95% posterior probability intervals")
lines(x.seq, estimates, lwd = 2)
lines(x.seq, estimate_2.5, col = "red")
lines(x.seq, estimate_97.5, col = "red")
```

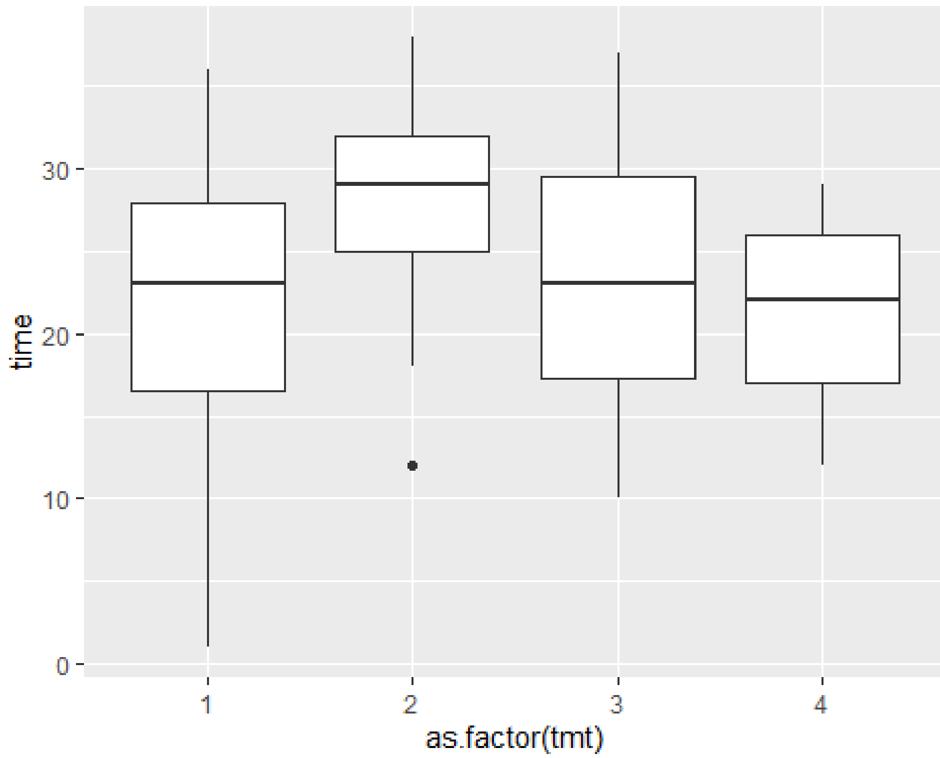


The next data set concerns survival of mice under different treatment conditions. The data file is called **mice.dat** and contains four columns: a mouse id, the treatment (there are four), the number of days the mouse survived, and a censored survival time (that is, the time the mouse survived was not recorded exactly, but the experimenters know the mouse survived at least as long as the time in this column). When you are asked to produce posterior distributions in this section, you should use SAS.

22. Produce a boxplot of the survival time by the treatment conditions. Take out all the censored data prior to making the plot.

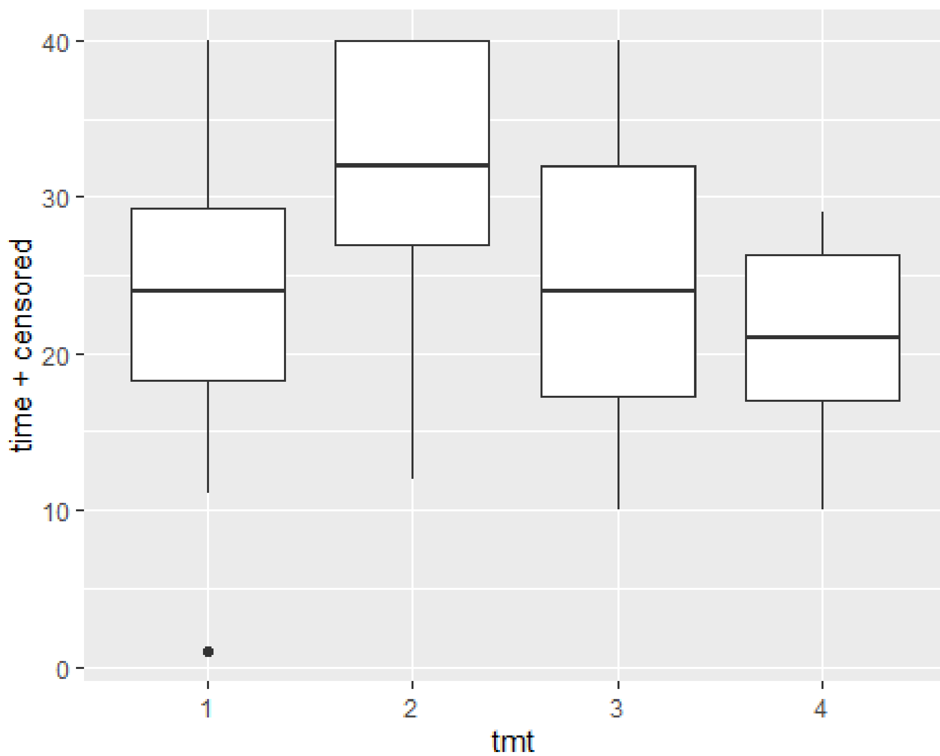
```
library(ggplot2)
mice <- read.table("mice.dat", header = TRUE)
head(mice)

  mid tmt time censored
1   1   1   12        0
2   2   1    1        0
3   3   1   21        0
4   4   1   25        0
5   5   1   11        0
6   6   1   26        0

mice$tmt <- as.factor(mice$tmt)


mice.plot <- mice[mice$censored == 0,]
```

```r
ggplot(mice.plot, mapping = aes(x = as.factor(tmt), y = time)) +
  geom_boxplot()
```



23. Produce another boxplot, this time putting the censored time in place of 0 in the time variable.

```r
ggplot(mice, mapping = aes(x = tmt, y = time+censored)) +
  geom_boxplot()
```



24. We expect survival times to be exponential or gamma, but for these data, we don't, in general, see very long tails. So for this problem, we are going to assume the survival time (likelihood) is normally distributed, with different means, and different variances in each treatment. Assume the prior distributions on the treatment means are all normal with mean 25 and variance 1000. Assume the prior distributions on the treatment variances are gamma with shape 4 and scale 10. Use SAS. Use the following control parameters: nmc=400000 nbi=5000 thin=10 seed=1234. Also use propcov=quanew. What is the mean of the survival time of treatment 2.

Mean survival time of treatment 2 is 27.78.

Sas Code:

data mice; infile 'C:/Users/nateh/Documents/Stat 451/mice.dat' firstobs = 2; input mid tmt time censored; run;

proc mcmc data = mice outpost = 'C:/Users/nateh/Documents/Stat 451/mice.sas7bdat' seed = 1234 nmc = 400000 nbi = 5000 thin = 10 monitor = (*parms*) diagnostics = (rl ess autocorr) dic propcov = quanew; array mu[4] mu1-mu4; array vv[4] vv1-vv4; parms mu:; parms vv:; prior mu: ~ normal(mean= 25, sd = 1000); prior vv: ~ gamma(shape = 4, scale = 10); theta = mu[tmt]; alpha = vv[tmt]; if censored = 0 then ll = logpdf('normal', time, theta, alpha); else ll = logsdf('normal', time, theta, alpha); model general(ll); run;

25. What is the probability that the survival time in treatment 2 exceeds the survival time in treatment 1?

The probability that the survival time for treatment 2 exceeds the survival time for treatment 1 is 0.947.

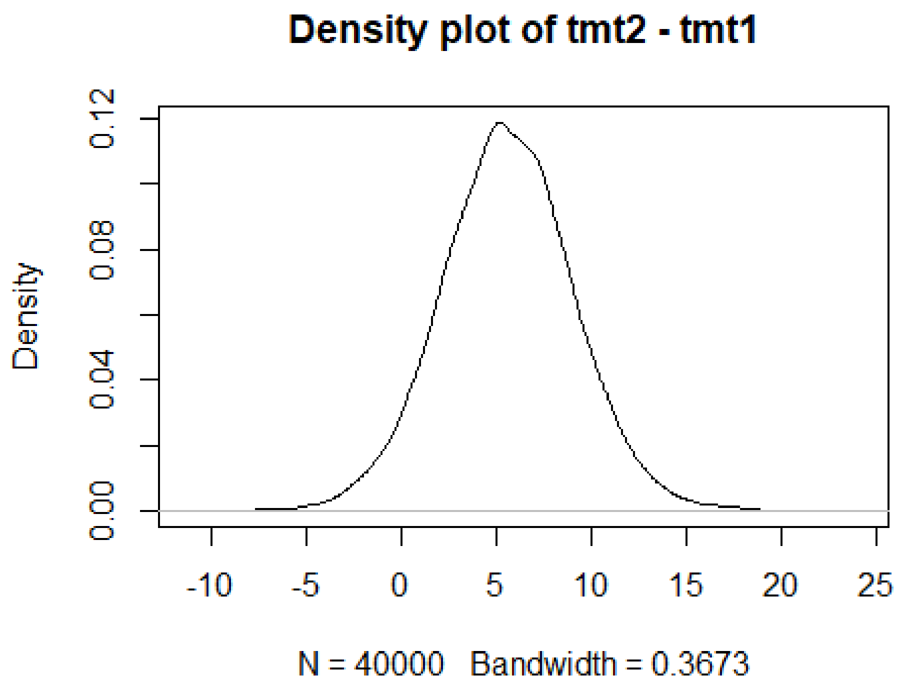
```
library(sas7bdat)
chains <- read.sas7bdat('C:/Users/nateh/Documents/Stat 451/mice.sas7bdat')
sims <- as.mcmc(chains)
raftery.diag(sims)
```

```
 Quantile (q) = 0.025
 Accuracy (r) = +/- 0.005
 Probability (s) = 0.95

            Burn-in  Total Lower bound  Dependence
            (M)      (N)   (Nmin)       factor (I)
  Iteration 6905     6905  3746         1.84
  mu1       4        5134  3746         1.37
  mu2       5        5576  3746         1.49
  mu3       5        5390  3746         1.44
  mu4       4        5265  3746         1.41
  vv1       3        4492  3746         1.20
  vv2       4        4872  3746         1.30
  vv3       4        4933  3746         1.32
  vv4       4        4712  3746         1.26
  LogPrior  5        5858  3746         1.56
  LogLike   8        10620 3746         2.84
  LogPost   8        9938  3746         2.65
```

```
tmt1 <- chains[,2]
tmt2 <- chains[,3]
mean(tmt1<tmt2)
```

```
 [1] 0.94705
```

```
plot(density(tmt2-tmt1), main = "Density plot of tmt2 - tmt1")
```


Density plot of tmt2 - tmt1
Density
0.00
0.04
0.08
0.12
-10
-5
0
5
10
15
20
25
N = 40000   Bandwidth = 0.3673