Convergence Diagnostics

Sarah Hunter

5/20/2020

Convergence Diagnostics

While we can never *prove* convergence, we can provide enough evidence against non-convergence. We do this with several visual checks and diagnostic tests that can be similar to those used in time series, checking for serial autocorrelation. We will look at the following diagnostics in this lab:

- Visual Checks
- Geweke
- Gelman and Rubin
- Heidelberger-Welch
- Raftery-Lewis

The Model

First, we need to load the data and estimate the Bayesian model. I am using the Angell data from the `car` package for this example In these data, the dependent variable is the "morality of cities".

```
#packages
library(foreign)
library(R2jags)
```

```
## Loading required package: rjags
```

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

```
## Linked to JAGS 4.3.0
```

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

```
##
## Attaching package: 'R2jags'
```

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

```
library(lattice)
library(devtools)
```

```
## Loading required package: usethis
```

```
#data
library(car)
```

```
## Loading required package: carData
```

```r
data(Angell)
summary(Angell)
```

```
##      moral           hetero         mobility     region
##  Min.   : 4.20   Min.   :10.60   Min.   :12.10   E : 9
##  1st Qu.: 8.70   1st Qu.:16.90   1st Qu.:19.45   MW:14
##  Median :11.10   Median :23.70   Median :25.90   S :14
##  Mean   :11.20   Mean   :31.37   Mean   :27.60   W : 6
##  3rd Qu.:13.95   3rd Qu.:39.00   3rd Qu.:34.80
##  Max.   :19.00   Max.   :84.50   Max.   :49.80
```

```r
#remove 4th column
angell.1<-Angell[,-4]
head(angell.1)
```

```
##            moral hetero mobility
## Rochester   19.0   20.6     15.0
## Syracuse    17.0   15.6     20.2
## Worcester   16.4   22.1     13.6
## Erie        16.2   14.0     14.8
## Milwaukee   15.8   17.4     17.6
## Bridgeport  15.3   27.9     17.5
```

```r
#Define vectors of the data matrix for JAGS
moral<-angell.1$moral
hetero<-angell.1$hetero
mobility<-angell.1$mobility
N<-length(angell.1$moral)

#Read in Angell data for JAGS
angell.data<-list("moral", "hetero", "mobility", "N")
angell.data<-list(moral=Angell$moral, hetero=Angell$hetero, mobility=Angell$mobility, N=length(Angell$m

#The model function
angell.model.jags<-function(){
for(i in 1:N){
    moral[i]~dnorm(mu[i], tau)
    mu[i]<-alpha + beta1*hetero[i] + beta2*mobility[i]
}


alpha~dnorm(0,.01)
beta1~dnorm(0, .1)
beta2~dnorm(0, .1)
tau~dgamma(.1,.01)

}

#Parameters of the JAGS model that you will monitor
angell.params<-c("alpha", "beta1", "beta2")

#Define starting values
angell.inits<-function(){
    list("alpha"=c(20), "beta1"=c(-0.1), "beta2"=c(-.02))
}
```

```
#OR

inits1<-list("alpha"=0, "beta1"=0, "beta"=0)
inits2<-list("alpha"=1, "beta1"=1, "beta2"=1)
angell.inits<- list(inits1, inits2)

#Fit the model in JAGS, if you saved model in wd as "angell.model.jags"
angellfit<-jags(data=angell.data, inits=angell.inits, angell.params, n.chains=2,
                n.iter=9000, n.burnin=1000, model=angell.model.jags)
```

```
## module glm loaded
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 43
##    Unobserved stochastic nodes: 4
##    Total graph size: 265
```

```
## Warning in jags.model(model.file, data = data, inits = init.values, n.chains =
## n.chains, : Unused initial value for "beta" in chain 1
```

```
## Initializing model
```

```
print(angellfit)
```

```
## Inference for Bugs model at "/var/folders/dx/pkx9cyj1089843ljm_jzj3pm0000gn/T//RtmpNMvm3s/modele06b5
##  2 chains, each with 9000 iterations (first 1000 discarded), n.thin = 8
##  n.sims = 2000 iterations saved
##          mu.vect sd.vect    2.5%     25%     50%     75%   97.5%  Rhat n.eff
## alpha     19.656   1.192  17.202  18.878  19.641  20.462  21.974 1.002  2000
## beta1     -0.106   0.017  -0.140  -0.118  -0.107  -0.095  -0.071 1.004   430
## beta2     -0.186   0.035  -0.256  -0.210  -0.186  -0.163  -0.118 1.002   870
## deviance 192.433   2.868 188.851 190.303 191.756 193.970 199.837 1.002  1000
##
## 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 = 4.1 and DIC = 196.5
## DIC is an estimate of expected predictive error (lower deviance is better).
```

```
#Converting model results to mcmc and data frames

angellfit.mcmc<-as.mcmc(angellfit)

angellfit.mat<-as.matrix(angellfit.mcmc)
angellfit.df<-as.data.frame(angellfit.mat)
```

Visual Convergence Diagnostics

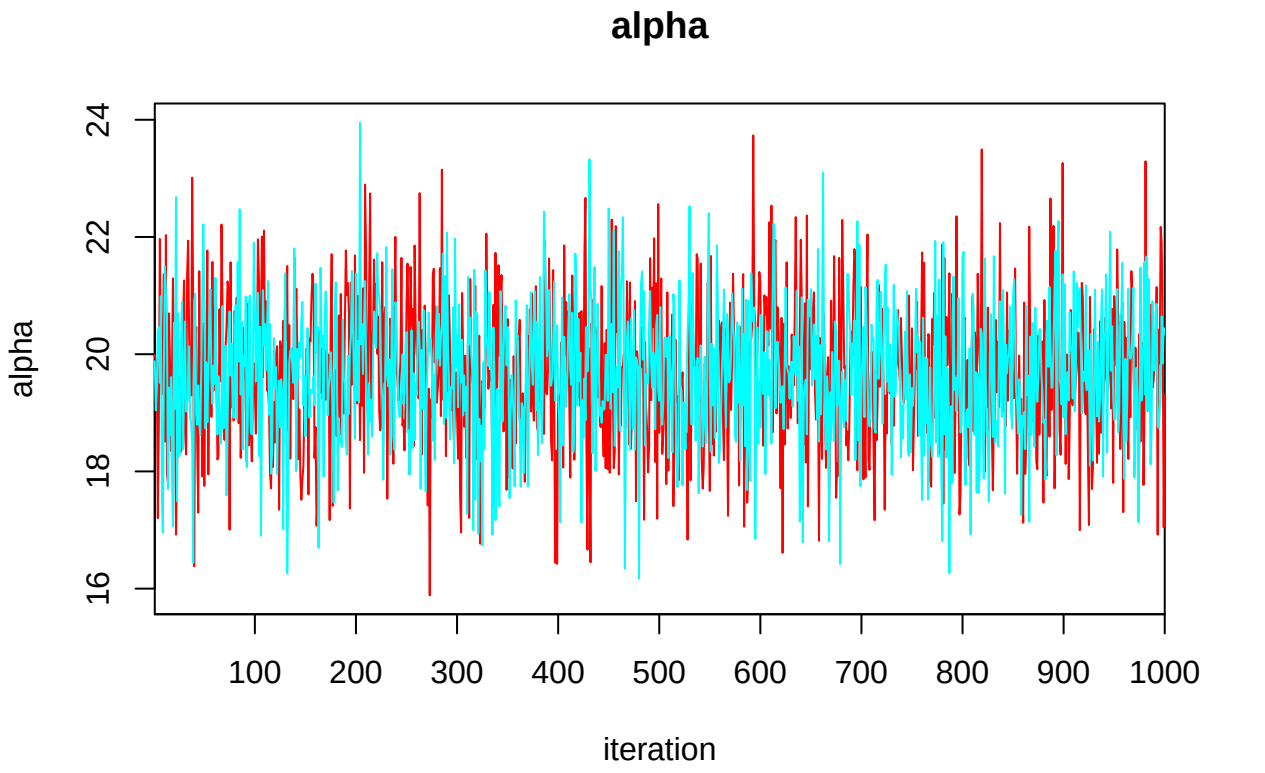
You can use a set of three figures that can help establish evidence against non-convergence. I will show each in turn.

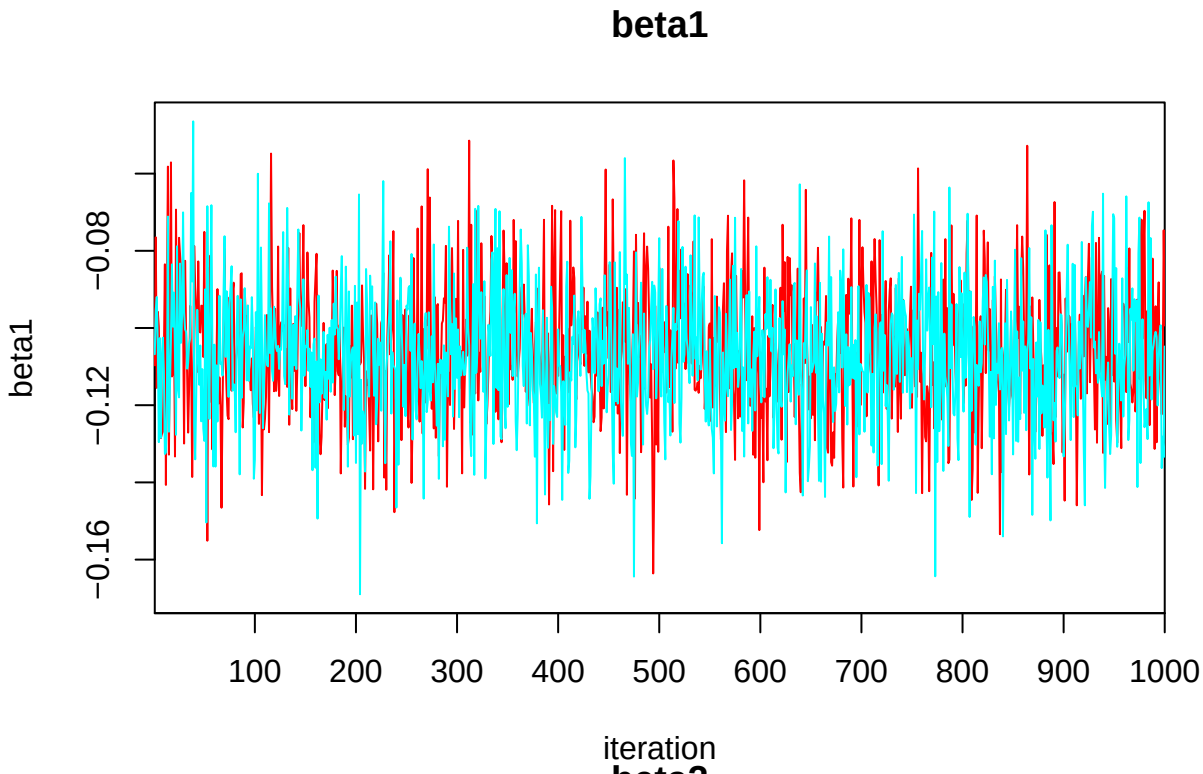
Traceplots

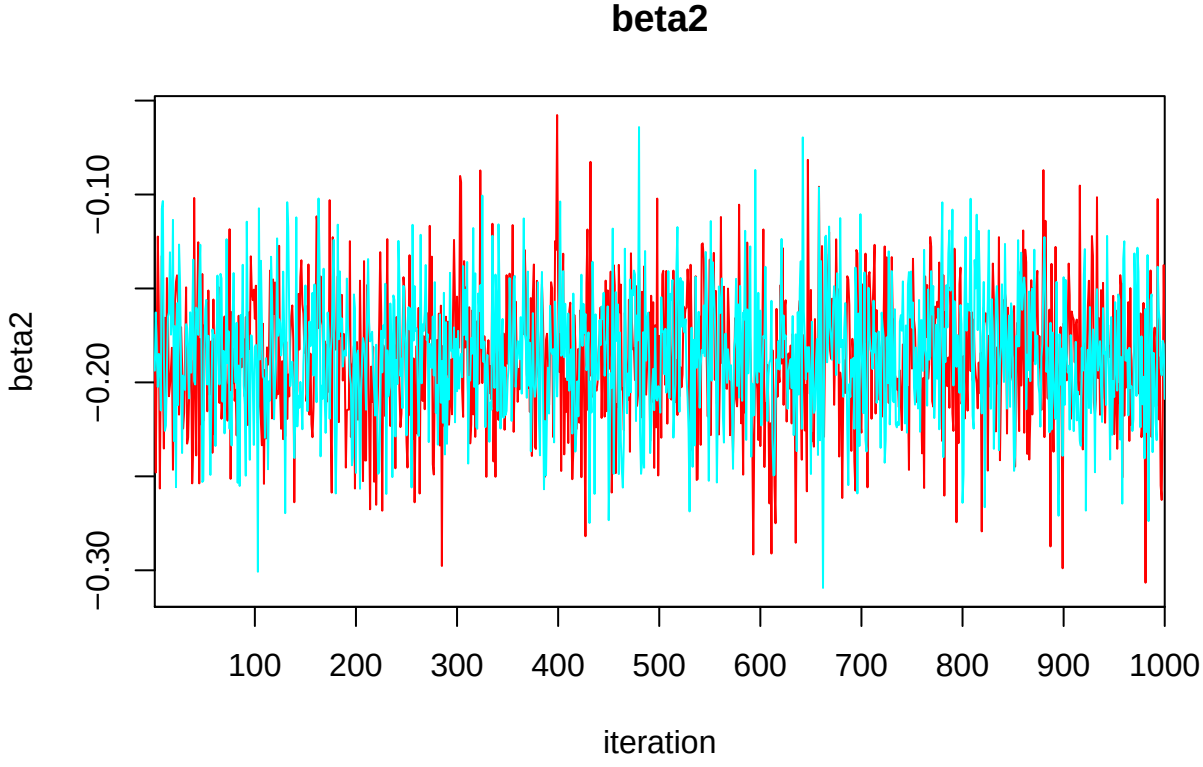
Traceplots are the most common convergence diagnostic tool used. It is especially useful as a first method because it is generally easy to use and quick to interpret. Many packages have traceplots that you can use, include the `R2jags` library. However, those are frankly ugly and only show up one at a time. However, they are preferable when you have a large number of parameters and the other plots are smushed.

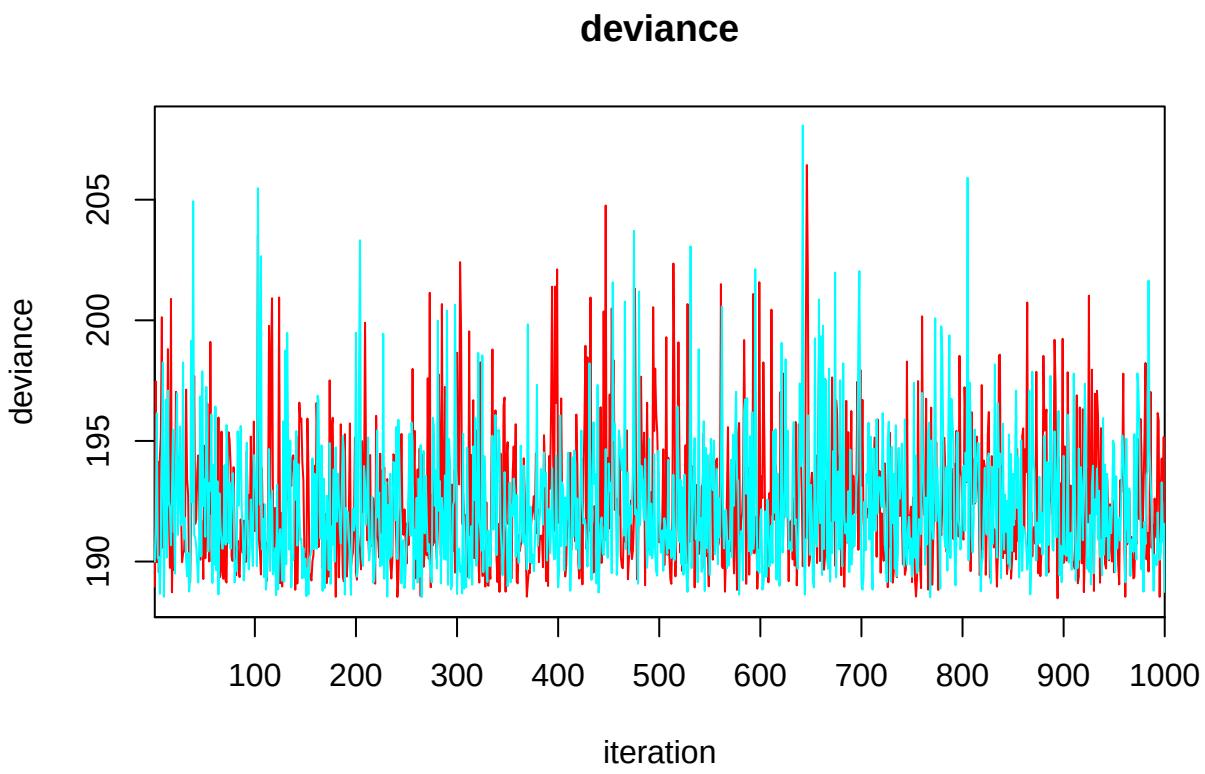
To interpret these plots, you are looking for a good "mixing" between the two chains. We usually say that we are looking for "fuzzy caterpillars".

```
#Using R2jags
traceplot(angellfit)
```




beta1
beta1
−0.08
−0.12
−0.16
100
200
300
400
500
600
700
800
900
1000
iteration

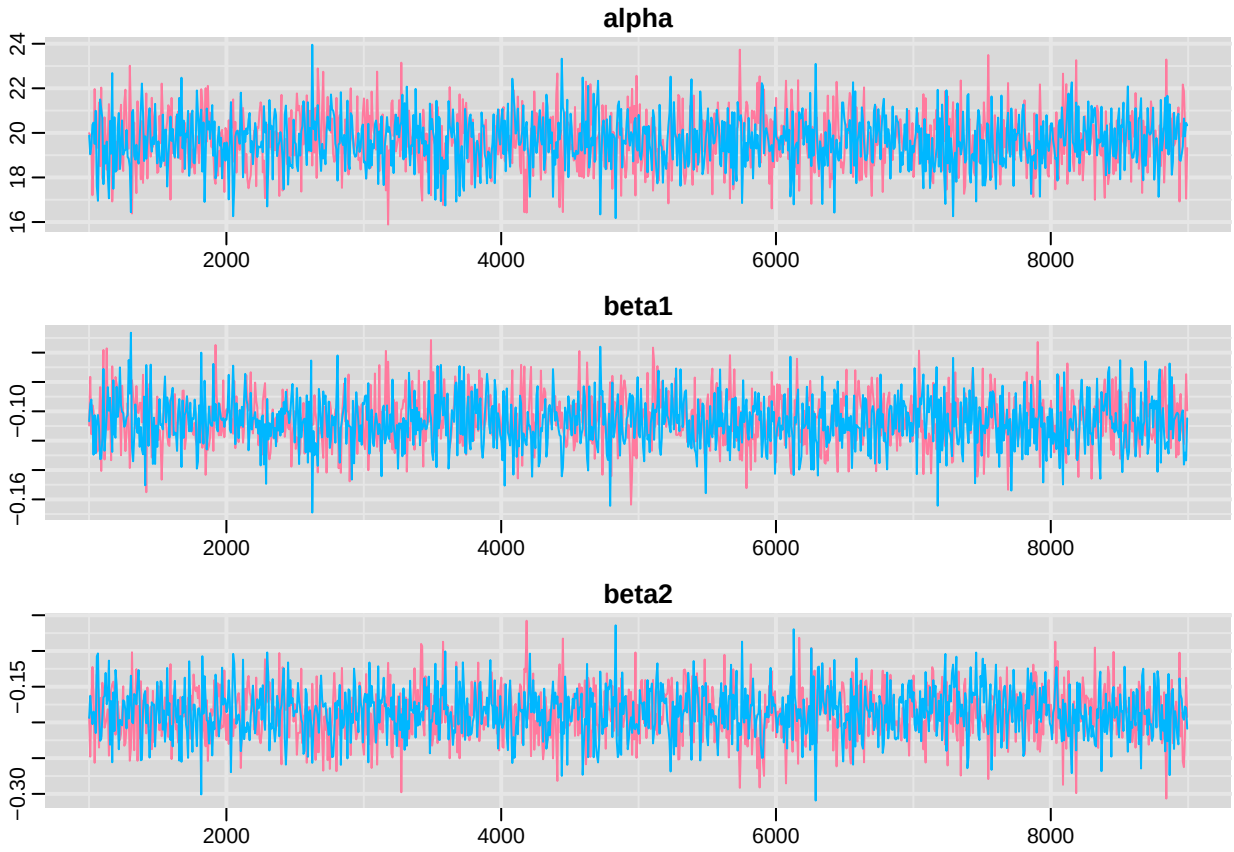

beta2
beta2
−0.10
−0.20
−0.30
100
200
300
400
500
600
700
800
900
1000
iteration




```
#Using mcmcplots
library(mcmcplots)
```

```
## Registered S3 method overwritten by 'mcmcplots':
##   method        from
##   as.mcmc.rjags R2jags
```

```
traplot(angellfit.mcmc, parms = c("alpha", "beta1", "beta2"))
```




```
#Using ggmcmc
library(ggmcmc)
```

```
## Loading required package: dplyr
```

```
##
## Attaching package: 'dplyr'
```

```
## The following object is masked from 'package:car':
##
##     recode
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

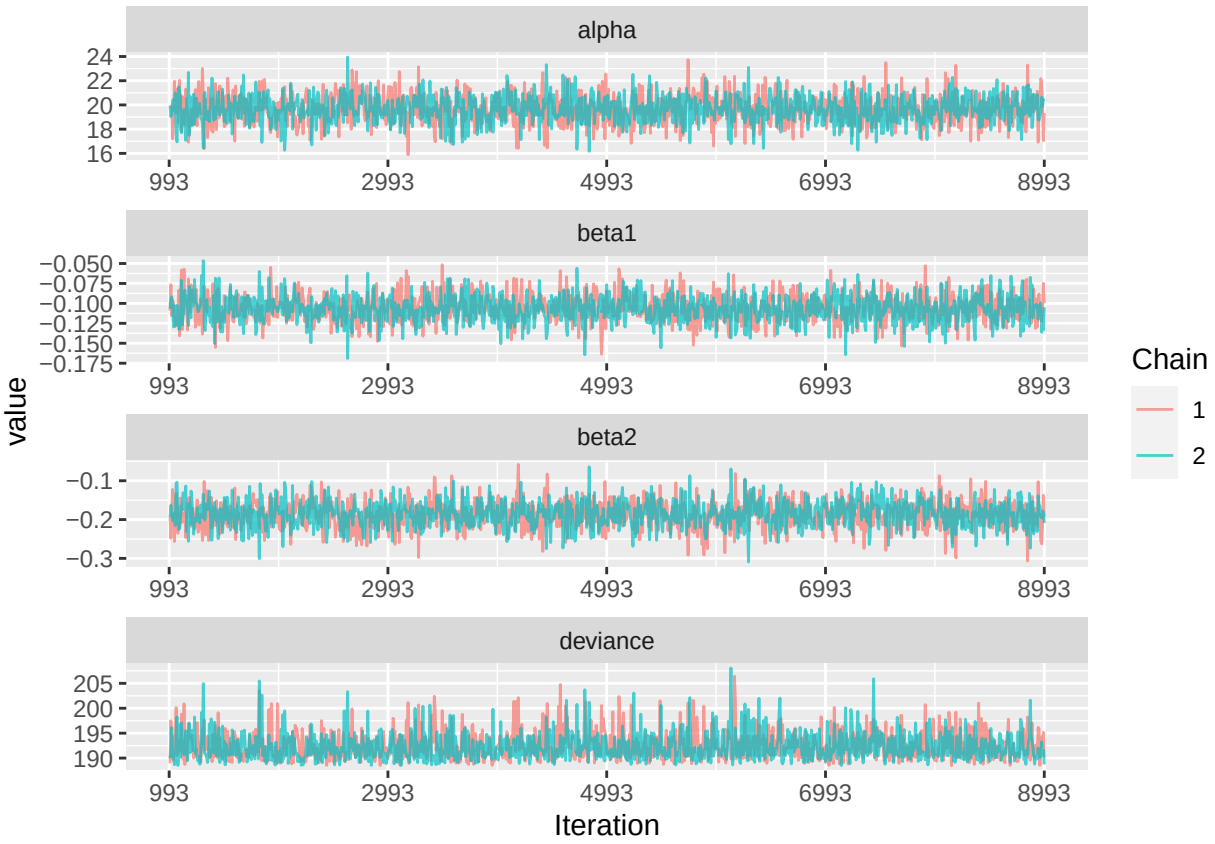
```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
## Loading required package: tidyr
```

```
## Loading required package: ggplot2
```

```
## Registered S3 method overwritten by 'GGally':
##   method from
##   +.gg   ggplot2
```

```
angellfit.gg<-ggs(angellfit.mcmc)
ggs_traceplot(angellfit.gg)
```

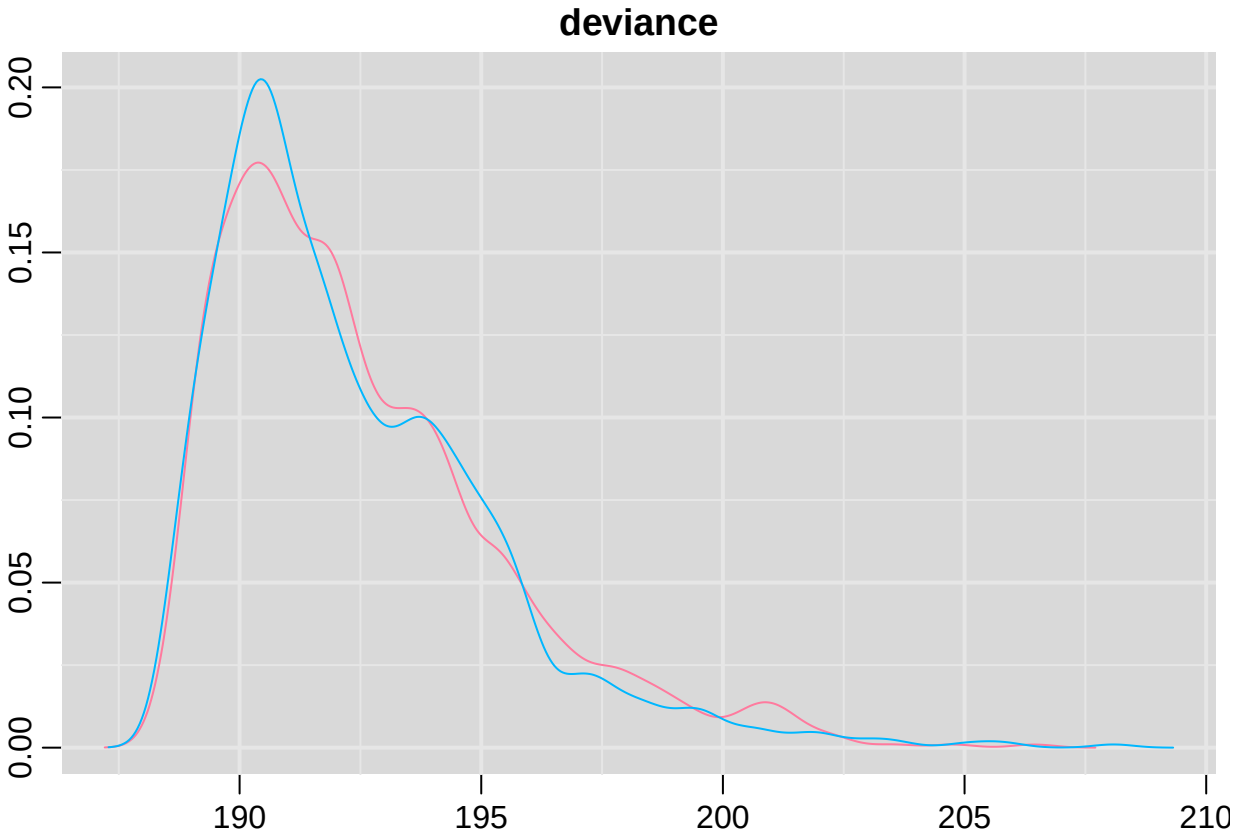


Density Plots

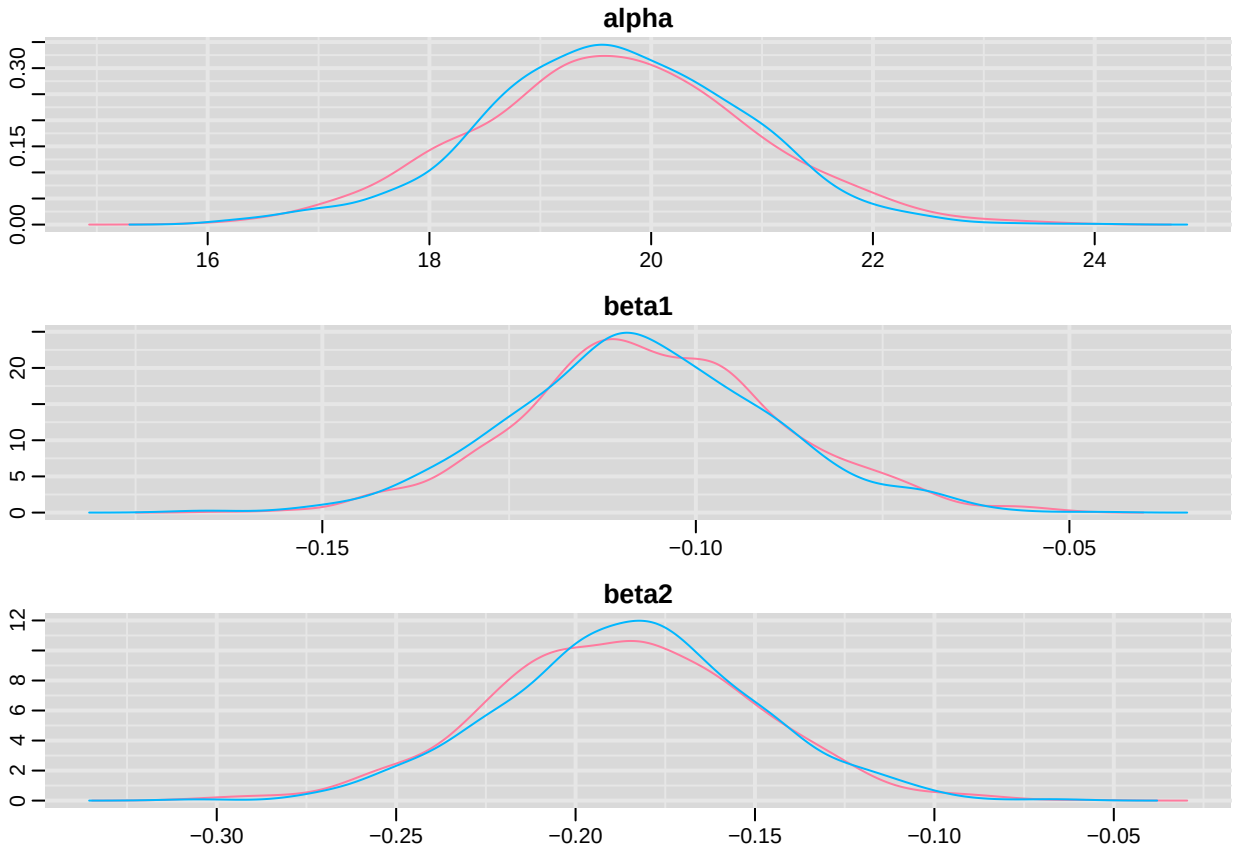
Our old friend the density plot shows up here two. However, these diagnostic plots are a little different because we plot the density of the parameters for each chain. This can also be done with several packages. I am just posting my favorites here.

To interpret these plots, we are looking for sufficient overlap between the two densities. The plots helpfully make the two densities different colors (if you are color blind, let me know and I will figure out how to change the colors).

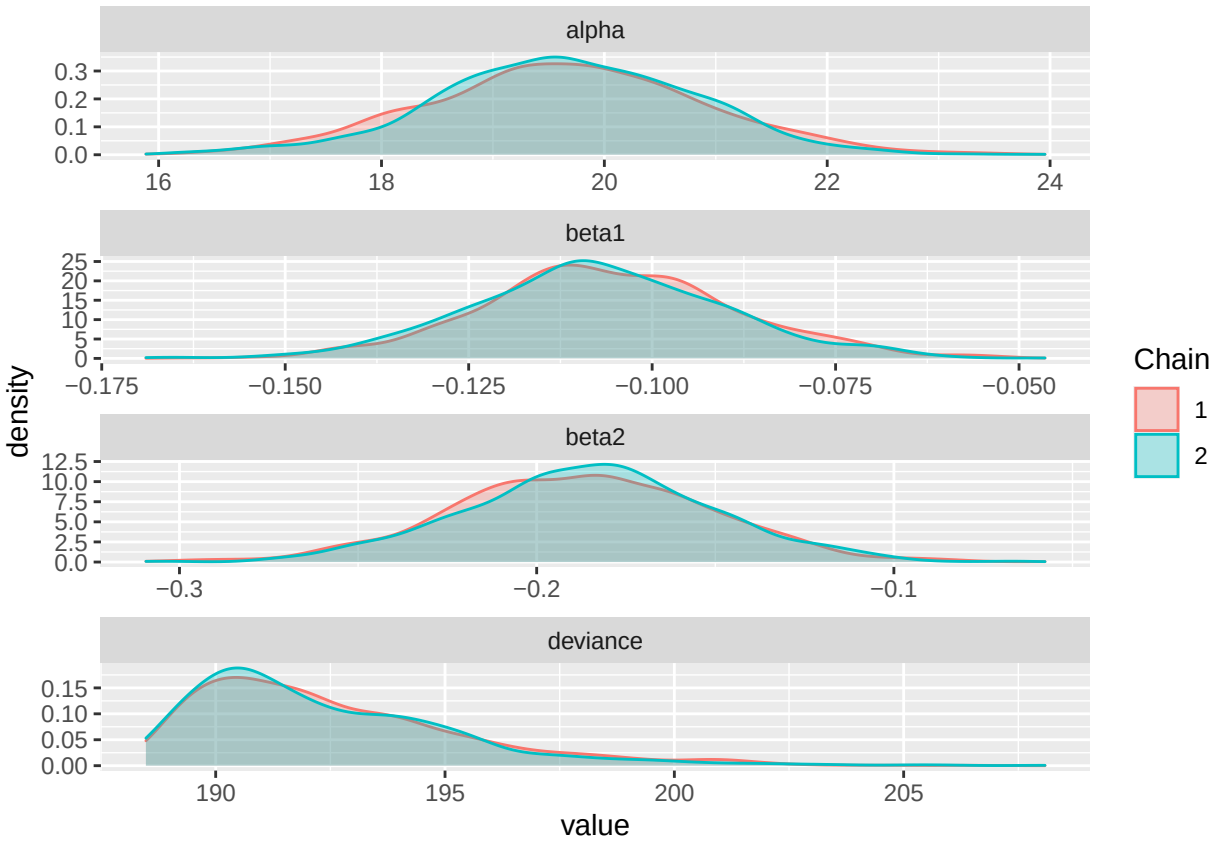
```
#From mcmcplots

denplot(angellfit.mcmc, parms = c("deviance"))
```




```
denplot(angellfit.mcmc, parms = c("alpha", "beta1", "beta2"))
```



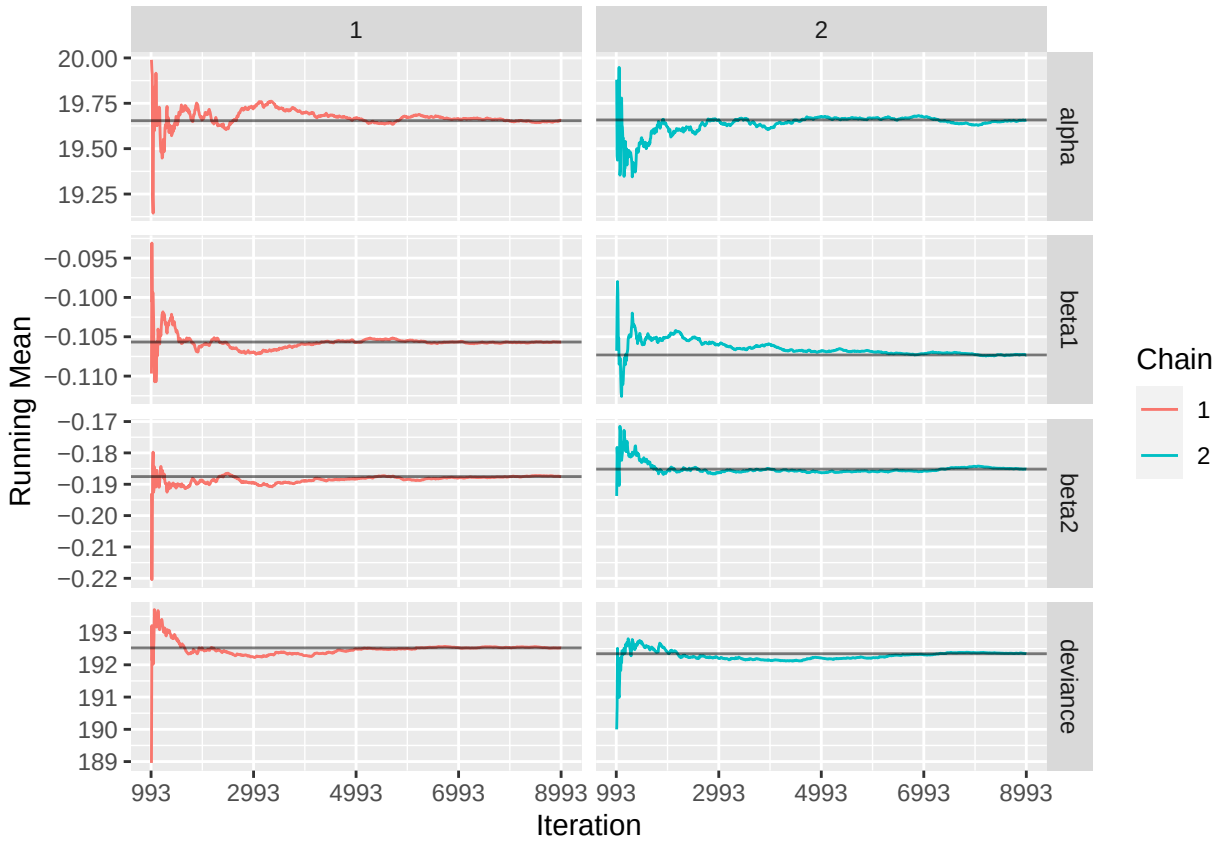

```
#From ggmcmc

ggs_density(angellfit.gg)
```



Running Plots

Running plots show each posterior draw per iteration by variable for each chain. The dark line in this plot indicates the posterior mean for that chain. We want these means to be as equal as possible, or make sure that the chains are exploring the same area. For both chains, we also want to see the draws from the posterior distribution stay as close as possible to the posterior mean. If the observations bounce around too much, it is an indication that the posterior has not reached a stationary distribution.

```
ggs_running(angellfit.gg)
```



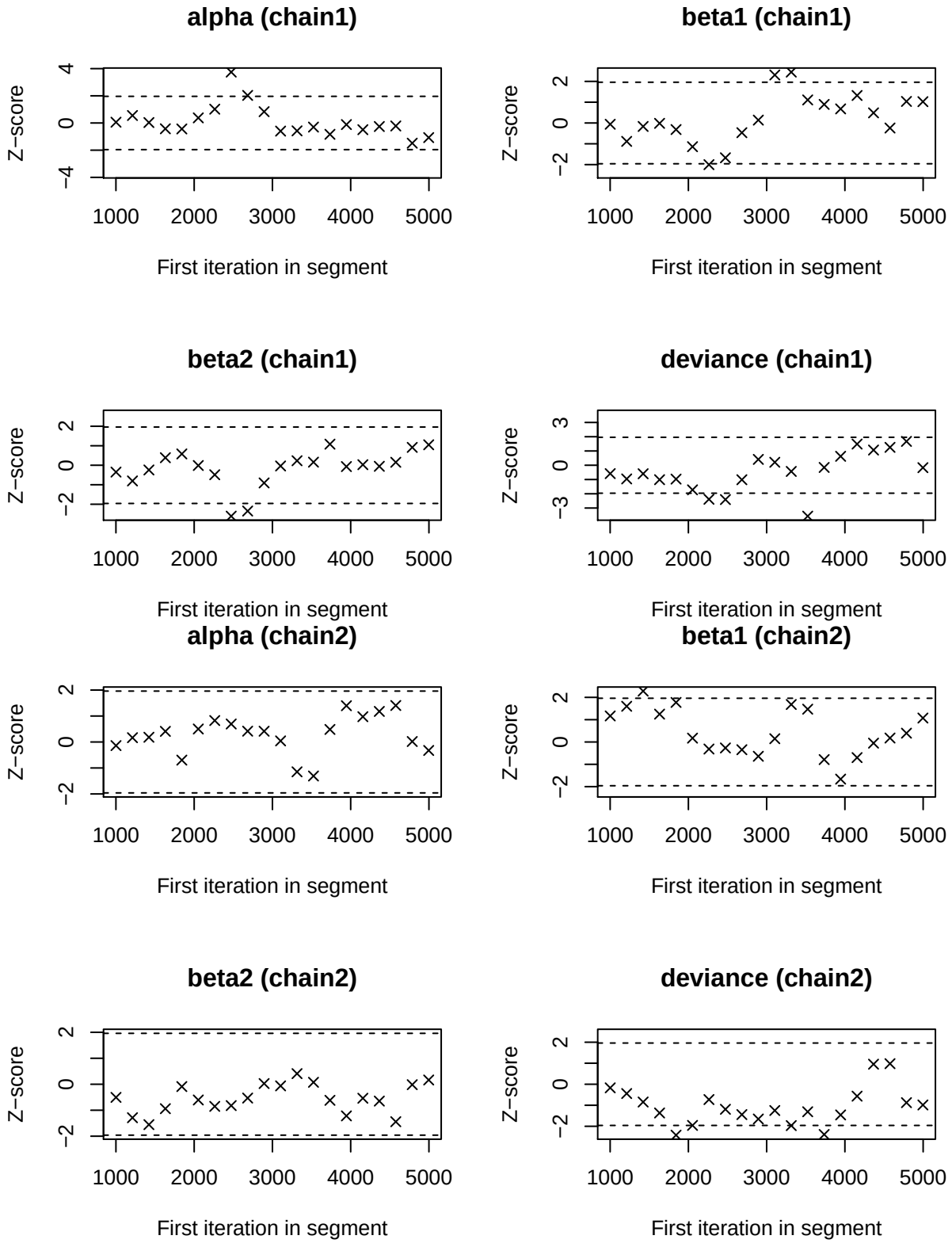
Geweke Test

The Geweke Test compares the posterior mean of the first half of a Markov chain to the second half. Under this test, the null hypothesis that the model has achieved convergence. Therefore, we do not want to reject the null hypothesis. The following code demonstrates how to conduct the Geweke test on your Bayesian model. The Geweke test also comes with a plot version of the results.

```
geweke.diag(angellfit.mcmc)
```

```
## [[1]]
##
## Fraction in 1st window = 0.1
## Fraction in 2nd window = 0.5
##
##    alpha    beta1    beta2 deviance
##  0.05804 -0.06090 -0.35093 -0.58749
##
##
## [[2]]
##
## Fraction in 1st window = 0.1
## Fraction in 2nd window = 0.5
##
##    alpha    beta1    beta2 deviance
##  -0.1385   1.1677  -0.5059  -0.1715
```

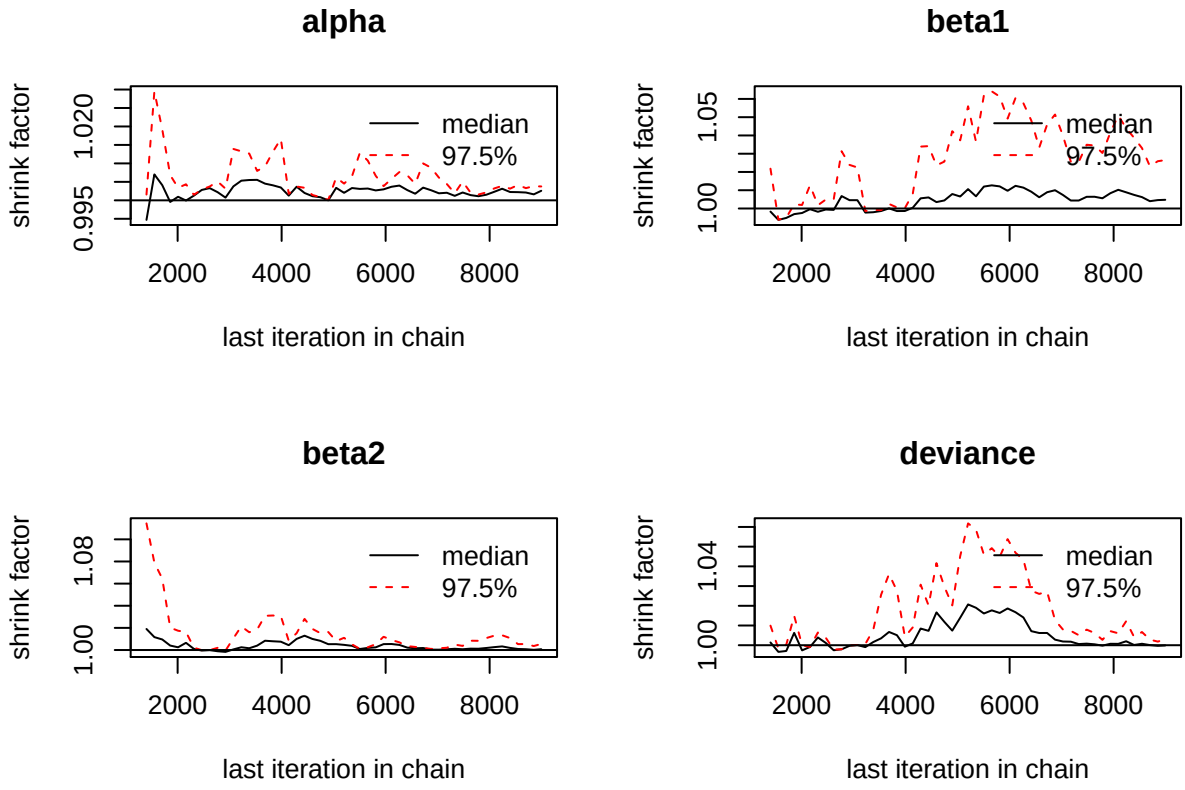
```
geweke.plot(angellfit.mcmc)
```



Gelman and Rubin Test

The Gelman and Rubin test compares the within and between chain variances. It produces a test statistic called "R hat". This statistic is automatically conducted and reported for the parameters every time you estimate a Bayesian model with JAGS. Generally speaking, $\hat{R} > 1$ is cause for concern over non-convergence. There is a separate command that can be used to plot the R hat values, and therefore can find the ones that are above 1.

```
gelman.plot(angellfit.mcmc)
```



Heidelberger-Welch Test

The Heidelberger-Welch test directly tests whether or not the draws for each parameter come from a stationary distribution. Under this test, the null hypothesis is that the variable is from a stationary distribution. Again, like the Geweke Test, we do not want to reject the null hypothesis.

```
heidel.diag(angellfit.mcmc)
```

```
## [[1]]
##
##          Stationarity start     p-value
##          test         iteration
## alpha    passed       1         0.622
## beta1    passed       1         0.748
## beta2    passed       1         0.535
## deviance passed       1         0.274
##
##          Halfwidth Mean    Halfwidth
##          test
```

```
## alpha    passed     19.654 0.07684
## beta1    passed     -0.106 0.00106
## beta2    passed     -0.188 0.00223
## deviance passed    192.523 0.17947
##
## [[2]]
##
##          Stationarity start     p-value
##          test         iteration
## alpha    passed       1         0.768
## beta1    passed       1         0.237
## beta2    passed       1         0.599
## deviance passed       1         0.143
##
##          Halfwidth Mean    Halfwidth
##          test
## alpha    passed     19.658 0.07239
## beta1    passed     -0.107 0.00107
## beta2    passed     -0.185 0.00209
## deviance passed    192.343 0.17595
```

Raftery-Lewis Test

This test is a bit different from the others. It doesn't necessary test for convergence, but rather how many more iterations the model needs before it reaches a stationary distribution. More formally, this test tells the researcher the minimum number of iterations that would be needed to estimate the specified quantile to the desired precision if the samples in the chain were independent.

You can interpret this test by looking at the feasibility of convergence. Generally speaking, you want for the total number of needed iterations to be less than 100,000. Any more than 100,000 gives cause for concern.

```
raftery.diag(angellfit.mcmc)
```

```
## [[1]]
##
## Quantile (q) = 0.025
## Accuracy (r) = +/- 0.005
## Probability (s) = 0.95
##
## You need a sample size of at least 3746 with these values of q, r and s
##
## [[2]]
##
## Quantile (q) = 0.025
## Accuracy (r) = +/- 0.005
## Probability (s) = 0.95
##
## You need a sample size of at least 3746 with these values of q, r and s
```

One command to run them all

There also exists a library that does all of the diagnostics for you with one command. This library is `superdiag` and is very simple to use. You just need to interpret each diagnostic individually, using the above information as a guide.

```
library(superdiag)
```

```
## Loading required package: boa
```

```
superdiag(angellfit.mcmc, burnin=100)
```

```
## Number of chains = 2
## Number of iterations = 1000 per chain before discarding the burn-in period
## The burn-in period = 100 per chain
## Sample size in total = 1800
##
## ********** The Geweke diagnostic: **********
## Z-scores:
##                       chain1    chain 2
## alpha             -0.3605961 -0.4283273
## beta1             -0.2177596  1.3258569
## beta2              0.5384182 -0.1377893
## deviance          -0.6707489 -0.7659940
## Window From Start  0.1000000  0.0820900
## Window From Stop   0.5000000  0.8369900
##
## ********** The Gelman-Rubin diagnostic: **********
## Potential scale reduction factors:
##
##          Point est. Upper C.I.
## alpha             1       1.00
## beta1             1       1.03
## beta2             1       1.02
## deviance          1       1.00
##
## Multivariate psrf
##
## 1.01
##
## ********** The Heidelberger-Welch diagnostic: **********
##
## Chain 1, epsilon=0.1, alpha=0.05
##          Stationarity start     p-value
##          test         iteration
## alpha    passed       1         0.542
## beta1    passed       1         0.693
## beta2    passed       1         0.547
## deviance passed       1         0.314
##
##          Halfwidth Mean    Halfwidth
##          test
## alpha    passed     19.654 0.08054
## beta1    passed     -0.106 0.00111
## beta2    passed     -0.187 0.00237
## deviance passed    192.539 0.20082
##
## Chain 2, epsilon=0.037, alpha=0.1
##          Stationarity start     p-value
##          test         iteration
```

```
## alpha    passed        1        0.733
## beta1    passed        1        0.510
## beta2    passed        1        0.540
## deviance passed       91        0.129
##
##          Halfwidth Mean    Halfwidth
##          test
## alpha    passed     19.661 0.07821
## beta1    passed     -0.108 0.00112
## beta2    passed     -0.185 0.00227
## deviance passed    192.366 0.19348
##
## ********** The Raftery-Lewis diagnostic: **********
##
## Chain 1, converge.eps = 0.001
## Quantile (q) = 0.025
## Accuracy (r) = +/- 0.005
## Probability (s) = 0.95
##
## You need a sample size of at least 3746 with these values of q, r and s
##
## Chain 2, converge.eps = 5e-04
## Quantile (q) = 0.05
## Accuracy (r) = +/- 0.001
## Probability (s) = 0.9
##
## You need a sample size of at least 128514 with these values of q, r and s
```