Bayesian Analysis on effectiveness of Progabide in treating Epilepsy Seizure

Statistics For Data Science II Project

Roberta Parisi - roberta.parisi2@gmail.com - 1548814

Abstract

The aim of this work is to carry out a fully Bayesian Analysis and study the effect of a drug, progabide, in some epilectic patient. Specifically ……

1. Dataset

1. Dataset

The dateset contains clinical trials conducted on 59 patients suffering from epilepsy. A total of 31 randomly chosen patients received an anti-epileptic drug (progabide) while the remaining 28 received a placebo in addition to standard chemotherapy. The data have 5 longitudinal measurements:

- A *baseline seizure count* of the 8 weeks prior to being randomized to treatment;
- Four measurement for the next 8 weeks, where each of them is repeated every two weeks and represents the *count of seizure* in that period (referred to as CS).

The period length of the variable *baseline* differs from the other variables (8 weeks vs 2 weeks). To make the comparison possible I decided to divide the count values of the baseline by 4, so that I have an average for a period length of 2 weeks (I will refer to this new variable as BS_4).

Let's observe more in details the variables in the dataset:

- Treatment group of the patiente (0 if placebo / 1 if progabide, called *Trt*);
- Baseline seizures counts, the seizure count before the trial starts (*base*);
- The visit period (First two weeks, Second two weeks, Third two week or Fourth two weeks) ;
- The age of the patient (*Age*);
- The seizure count for each patient and periods (y);
- A dummy variable which flag with the value 1 the 4th visit of every patient and with 0 the others (*V4*).

Descriptive Analysis

The median seizure count before to start the trial (BS_4) was 5.5 and the average was of almost 8 (7.8), that became respectively 4.75 and 7.69 for the patients group that are receiving the placebo and 6 and 7.903 for the ones under treatment. Below we can see the baseline seizure count respectively for the entire group, for the placebo group and for the progabide group:

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   1.500   3.000   5.500   7.805  10.250  37.750

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   1.500   2.750   4.750   7.696  11.938  27.750

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   1.750   3.375   6.000   7.903   9.500  37.750
```

In the following plot we can see the seizure trend in the four visits period compared to the baseline $\frac{SC_t}{BS_4}$: Fig. 1 suggests that the use of progabide has a positive response on the patient. We can notice that the seizure count ratio through the four periods has a decreasing trend for the patients assuming the drug. On the other hand, the placebo group is not generally decreasing, even though their ratio distribution presents several fluctuations. A notable difference can be noticed in the median value of the ratio distribution: for the placebo group is always bigger than 1, which means that the number of seizure tends to grow with respect to the baseline. Below we can see the main statistics for each visit period, respectively both for the placebo group and the progabide group:

- Visit period 1:

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.0000  0.7395  1.0986  1.3082  1.3636  5.6000

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.0000  0.3618  0.7273  0.8449  1.1270  2.8387
```

- Visit period 2:

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.0000  0.7219  1.0909  1.2190  1.5824  5.2000

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.0000  0.5227  0.7368  1.0298  1.2566  4.1818
```

- Visit period 3:

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.0000  0.4904  1.0045  1.0629  1.1889  5.5273

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.0000  0.2642  0.6316  0.8591  1.0714  3.4545
```

- Visit period 4:

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.0000  0.6667  1.0680  1.0996  1.3333  3.3333

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.0000  0.4195  0.7273  0.7443  1.0357  2.0645
```

It's important to point out that there are differences in the seizure counts within patients but also between patients over time. Specifically, one of the patient seems to have an extreme number of seizure counts at all the time points compared to the other one. Another patient registered a considerably high number of seizures at the third visit with respect to his/her own standards. Even so, Thall and Vail (1990) find no clinical basis to tag this two patients as an extreme case, thereby I am not going to mark this patients as outlier and I am going to use the whole dataset as it is.

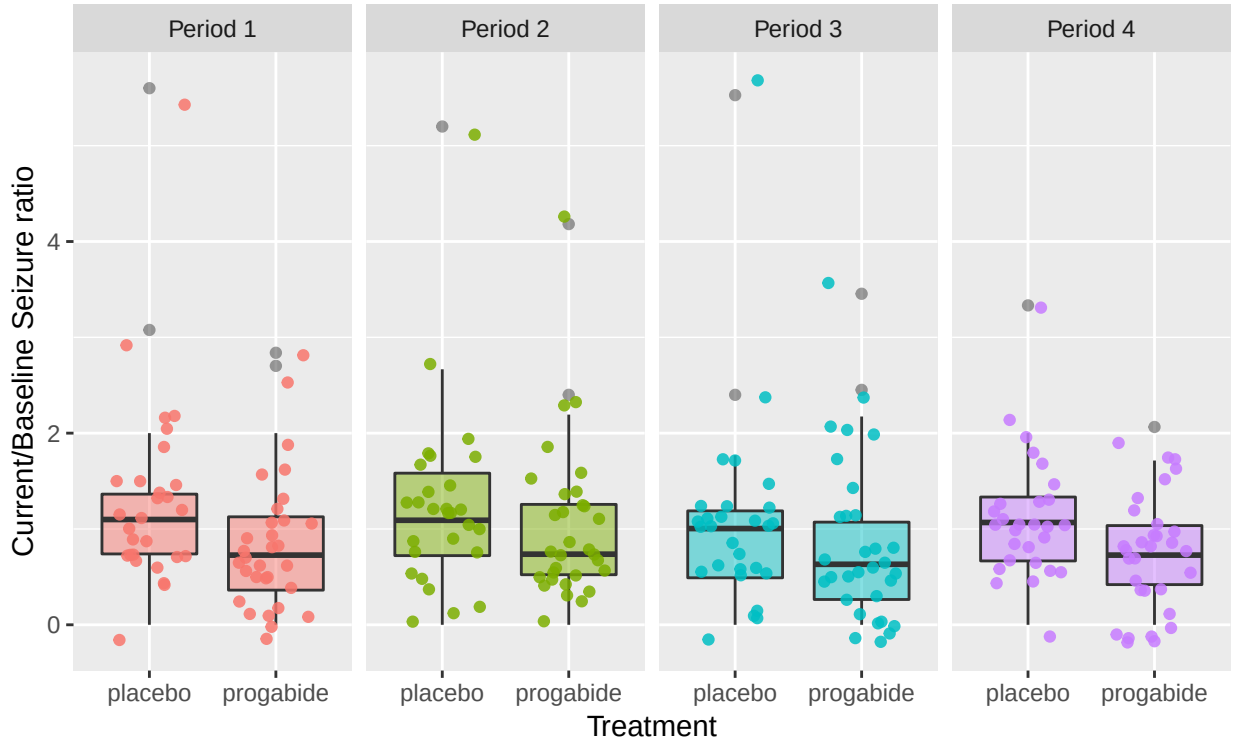


Figure 1: Rate of current seizure count over the baseline seizure count. The baseline is considerd divided by 4 since it's referring to an 8 weeks period.

2. Model definition

The main question of interest is whether taking this anti-convulsant drug can reduce the number of epileptic seizures compared to placebo. Usually for count data a poisson model can represent a good choice, but in many applications (especially in longitudinal and/or biomedical data) such a simple functional relationship is inadequate to handle the heterogeneity of the data.

2.1 Model 0: Basic Poisson

Let's try to fit a simple poisson model considering the index j for the the observation level (1, ..., 59), the index k for the visit period (from 1 to 4) and few transformations applied to the variables: to *Baseline Seizure Count (BS_4)* and *age* I have applied the natural logarithm, as used in other studies (i.e. Breslow and Clayton, 1993 and also Thall and Vail, 1990).

$$Y_{j,k} \sim Pois(\mu_{j,k})$$

$$log(\mu_{j,k}) = \alpha_0 + \beta_1 log(BS_{4j}) + \beta_2 Trt_j + \beta_3 (log(BS_{4j}) * T_j) + \beta_4 log(Age_j) + \beta_5 V_{4k}$$

Prior:

$$B_i \sim Norm(0, 0.001) \qquad \forall \quad i = 1, ..., 5$$

BUGS MODEL

```
poissonModelString = "model
{
  for(j in 1 : N) {
  for(k in 1 : T) {
  log(mu[j, k]) <- a0 + beta.Base * log.Base4[j]
  + beta.Trt * Trt[j]
  + beta.BT * BT[j]
  + beta.Age * log.Age[j]
  + beta.V4 * V4[k]
  y[j, k] ~ dpois(mu[j, k])
  }
  BT[j] <- Trt[j] * log.Base4[j] # interaction
  log.Base4[j] <- log(Base[j] / 4)
  log.Age[j] <- log(Age[j])
  }

  # priors:
  a0 ~ dnorm(0.0,1.0E-3)
  beta.Base ~ dnorm(0.0,1.0E-3)
  beta.Trt ~ dnorm(0.0,1.0E-3);
  beta.BT ~ dnorm(0.0,1.0E-3)
  beta.Age ~ dnorm(0.0,1.0E-3)
  beta.V4 ~ dnorm(0.0,1.0E-3)
  # re-calculate intercept on original scale:
  alpha0 <- a0 - beta.Base - beta.Trt
  - beta.BT - beta.Age  - beta.V4
}"
writeLines(poissonModelString , "basicPoisson.txt")
```

JAGS MODEL

```
library(rjags)
library(R2jags)

set.seed(200908)
parametersPoisson = c("beta.Age", "beta.BT","beta.Base","beta.Trt","beta.V4","a0")

basicPoissonJags = jags(model.file = "basicPoisson.txt",
                        parameters.to.save =  parametersPoisson,
                        data = data_jags, n.chains=3)
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 236
##    Unobserved stochastic nodes: 6
##    Total graph size: 897
```

```
##
## Initializing model
```

```
basicPoissonJags
```

```
## Inference for Bugs model at "basicPoisson.txt", fit using jags,
##  3 chains, each with 2000 iterations (first 1000 discarded)
##  n.sims = 3000 iterations saved
##             mu.vect sd.vect     2.5%      25%      50%      75%    97.5%
## a0           -2.643   0.415   -3.454   -2.913   -2.636   -2.372   -1.852
## beta.Age      0.866   0.116    0.636    0.789    0.865    0.942    1.092
## beta.BT       0.516   0.059    0.397    0.476    0.517    0.556    0.630
## beta.Base     0.973   0.046    0.888    0.942    0.973    1.001    1.061
## beta.Trt     -1.269   0.147   -1.548   -1.368   -1.270   -1.171   -0.976
## beta.V4      -0.212   0.054   -0.322   -0.248   -0.210   -0.175   -0.107
## deviance  1648.257  44.013 1638.077 1642.093 1645.181 1649.147 1660.540
##             Rhat n.eff
## a0         1.002  1300
## beta.Age   1.003   820
## beta.BT    1.001  3000
## beta.Base  1.007   330
## beta.Trt   1.001  3000
## beta.V4    1.014   160
## deviance   1.001  3000
##
## 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 = 969.2 and DIC = 2617.5
## DIC is an estimate of expected predictive error (lower deviance is better).
```

```
burn = 3000
iter = 20000

update(basicPoissonJags$model, burn)


#let's check for convergence
gelman.diag(as.mcmc(basicPoissonJags))
```

```
## Potential scale reduction factors:
##
##           Point est. Upper C.I.
## a0              1.00       1.01
## beta.Age        1.00       1.01
## beta.Base       1.01       1.02
## beta.BT         1.00       1.00
## beta.Trt        1.00       1.00
## beta.V4         1.01       1.05
## deviance        1.00       1.00
##
## Multivariate psrf
```

```
##
## 1.02
```

```
#let's see the autocorrelation
autocorr.diag(as.mcmc(basicPoissonJags))
```

```
##                a0   beta.Age   beta.Base      beta.BT    beta.Trt
## Lag 0   1.00000000 1.00000000 1.000000000  1.000000000  1.00000000
## Lag 1   0.74525311 0.72197058 0.754565864  0.692502682  0.75801705
## Lag 5   0.21185721 0.17314532 0.302807933  0.225363945  0.29357693
## Lag 10  0.06857778 0.05042056 0.102783262  0.075611266  0.08736456
## Lag 50  0.01986830 0.04191100 0.008205789 -0.008550144 -0.03803051
##            beta.V4    deviance
## Lag 0   1.00000000 1.000000000
## Lag 1   0.71188880 0.339793091
## Lag 5   0.17038309 0.042800365
## Lag 10  0.05389717 0.001426978
## Lag 50 -0.03396311 0.002784430
```

```
# i am going to add a thining factor of 10
```

In the first model, created with 3 chains, as we can see from the values of the *R.hat* of the results above the parameters converged.

The n.eff is a crude measure of effective sample size (*ESS*), that is the number of effectively independent draws from the posterior distribution that the Markov chain is equivalent to. When the *ESS* of a parameter is small (<100) then the estimate of the posterior distribution of that parameter will be poor with consequently large standard deviation for the parameter. On the other hand, a too big *ESS* (>10000) can represent a waste of computational resources. In our case the values are big enough, even though much different from each other.

The diagnostic obtained using the *gemalman.diag* function gives a factor of 1 for each parameter. This means that between variance and within chain variance are equal and so that the chains don't have notable difference between each other.

Thanks to the autocorrelation diagnostic we can see that we need a lot more iterations or a thining factor, since the autocorelation values are to high (they starts to become better in the lag 5, but still a little bit higher than what I like to see). I chose to use a thining factor since it allows me to obtaine the same results without wasting additional computational time and memory. Specifically I chose to use an thining factor of 10, since it's where the autocorellation starts to be significantly low for each parameters.

SIMULATION AND DIC COMPUTATION

```
library(coda)
basicPoissonSim = coda.samples(model = basicPoissonJags$model,
                               variable.names = parametersPoisson,
                               n.iter = iter, thin = 10)
summary(basicPoissonSim)
```

```
##
## Iterations = 4010:24000
## Thinning interval = 10
```

```
## Number of chains = 3
## Sample size per chain = 2000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##              Mean      SD  Naive SE Time-series SE
## a0        -2.6271 0.39564 0.0051077      0.0053150
## beta.Age   0.8630 0.11342 0.0014643      0.0015217
## beta.BT    0.5146 0.06051 0.0007812      0.0008160
## beta.Base  0.9723 0.04141 0.0005347      0.0005861
## beta.Trt  -1.2682 0.14894 0.0019228      0.0020635
## beta.V4   -0.2126 0.05397 0.0006968      0.0006955
##
## 2. Quantiles for each variable:
##
##              2.5%     25%     50%     75%   97.5%
## a0        -3.4079 -2.9006 -2.6284 -2.3501 -1.8637
## beta.Age   0.6435  0.7832  0.8632  0.9407  1.0852
## beta.BT    0.3983  0.4729  0.5141  0.5550  0.6368
## beta.Base  0.8932  0.9445  0.9718  0.9997  1.0548
## beta.Trt  -1.5645 -1.3687 -1.2667 -1.1664 -0.9809
## beta.V4   -0.3206 -0.2483 -0.2123 -0.1763 -0.1084
```

```
#let's compute the dic for this value
dicBasicPoisson = dic.samples(basicPoissonJags$model,
                              n.iter= iter,
                              thin = 10)
dicBasicPoisson
```

```
## Mean deviance:  1646
## penalty 5.898
## Penalized deviance: 1652
```

Looking at these simulations we can see that the model obtained low standard error, which gives more reliability to our parameters estimation.

That being said, the DIC value obtained for this model (3353) point out that the model itself is not good enough to explain the variability of our data. It's clear that this model is not taking in consideration some really important aspect of this data. As I said before, a really important aspect that must be taken in consideration it's related to the differences that we could notice within patients but also between a specific patient over time. This is because in our data is present a problem of overdispersion.

2.2 Overdispersion in epileptic data

The standard Poisson model implies that the mean and variance are equal, this implication is usually restrictive because count data samples often have the variance either lower than the mean (the so called underdispersion) or greater than the mean (also known as overdispersion).

In this case, as shown from the plot below (Fig. 2) that represent the density distributions of the seizure count in the four visit period, an high overdispersion is present, indeed the mean seizure count is 8.26 (specifically for the 4 period is respectively: 8.94, 8.36, 8.44, 7.31) whereas the variance is equal to 152.68 (for the 4 visit period is respectively: 220.08, 103.79, 200.18, 93.11). Therefore, using the Poisson model in

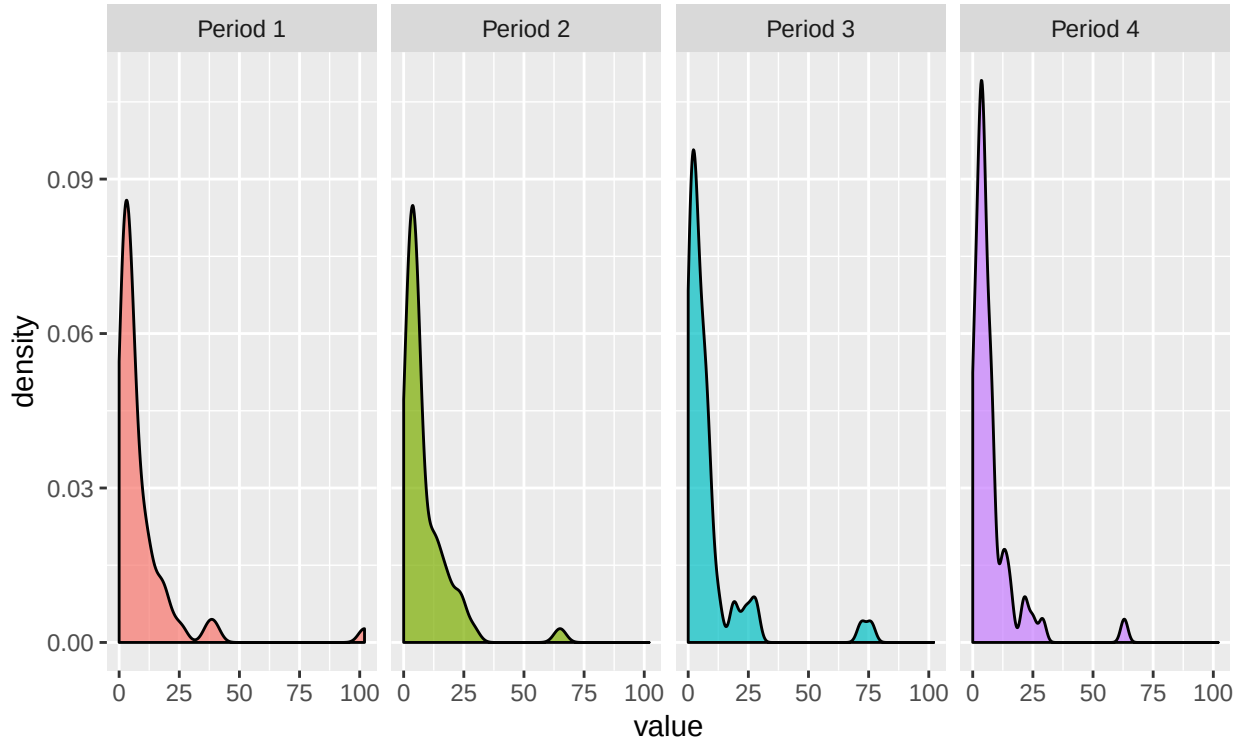


Figure 2: Density of the number of seizure for each visit period.

its basic form would not take in account this feature in the correct way. Let's take some test to proof that in this data are overdispersed:

```
library(AER)
frequentistPoissonModel = glm(y ~ trt + lbase + lage+ V4 + lbase:trt,
                              data = epil,
                              family = poisson)
dispersiontest(frequentistPoissonModel)
```

```
##
##  Overdispersion test
##
## data:  frequentistPoissonModel
## z = 3.085, p-value = 0.001018
## alternative hypothesis: true dispersion is greater than 1
## sample estimates:
## dispersion
##   4.209248
```

Here, with this first overdispersion test, we clearly see that there is evidence of overdispersion which could also be proof by fitting another frequentist model with a different distribution family: the quasi-poisson, a quasi-families which augment the normal poisson by adding a dispersion parameter (Poisson data -> $\bar{Y} = s^2_Y$ whilst QuasiPoisson data -> $\bar{Y} = \tau * s^2_Y$ where τ is the overdispersion parameter).

```r
frequentist_quasipoisson = glm(y ~ trt + lbase + lage+ V4 + lbase:trt, data = epil,family="quasipoisson

# dispersion coefficient:
summary(frequentist_quasipoisson)$dispersion
```

```
## [1] 4.413874
```

```r
# significance for overdispersion computed with the chi-square:
pchisq(summary(frequentist_quasipoisson)$dispersion * frequentistPoissonModel$df.residual, frequentistP
```

```
## [1] 4.873097e-99
```

The dispersion parameter of this model (4.41) validate the result obtained in the overdispension test, with a really high level of significance as the χ^2 attest.

When talking about overdispersion is important also to consider the ratio of 0 value in our data, which regularly in count data has an higher incidence than the expected from the Poisson model:

```r
#proportion of 0's in the data
sum(epil$y==0)/length(epil$y)
```

```
## [1] 0.09745763
```

```r
set.seed(1289)
#proportion of 0's expected from a Poisson distribution
mu <- mean(epil$y)
poisson_distribution <- rpois(10000, mu)
sum(poisson_distribution  == 0) /length(poisson_distribution )
```

```
## [1] 2e-04
```

The observed data has a higher proportion of zero with respect to the one expected, this means that our data could be zero inflated.

How to deal with overdisperision

To account for overdispersion a general solution is to use a quasi-poisson distribution, which is not estimated via maximum likelihood that means that we can't use either AIC value (we can't compute it) nor the deviance (is the same of a simple poisson).

Other way to aproach to count data with overdispersion is the use of a **negative binomial model** ($NB(p, r)$), which technically count the number of failures before the first success, and helps with overdispersion caused by an unmeasured latent variable. The parameter r, similarly to the τ parameter of the quasi-poisson model, represent a dispersion parameter. In this model, the variance is a function of his mean that is $var(Y) = \mu + \mu^2/r$, and as this dispersion parameter gets larger and larger, the variance converges to the same value as the mean, and the Negative Binomial turns into a Poisson distribution.

As I already said before, in this data there are difference in the seizure counts within patients but also between patients over time, and this could justify the use of random effects parameters in the model(either in a simple poisson or a negative binomial):

- A first random effect for the subject, hence a random effect for each individual observation;
- Another random effect related to the subject combined to the visit, hence a random effect for each individual observation and each individual time period.

To avoid the zero-inflation problem we can apply to the poisson or the negative binomial a zero-inflated model, and therefore a solution could be using the ZIP model with random effects or the ZINB model.

2.3 Model 1: Negative Binomial

Let's define the model:

$$Y_{j,k} \sim NegBinom(p_{j,k}, r_{j,k})$$

where $p_{j,k}$ is the probability of *"success"* of the patient in that period visit and is equal to

$$r_{j,k}/(r_{j,k} + \lambda_{j,k}$$

and r_{j,k} is the overdispersion parameter

$$r_{j,k} \sim U(0.0001, 1000)$$

To overcome high autocorrelation and helps the convergence of the markov chains I standardized each covariates by its mean, in this way I can ensure approximate prior independence between the regression coefficients:

$$log(\lambda_{j,k}) = \alpha_0 + \beta_1(log(BS_{4j}) - mean(log(BS_4))) + \beta_2(Trt_j - mean(Trt)) + \beta_3(log(BS_{4j}) * T_j) + \beta_4(log(Age_j) - mean(log(Age)))$$

Prior:

$$B_i \sim Norm(0, 0.001) \qquad \forall \quad i = 1, ..., 5$$

] ### BUGS MODEL

```
NegativeBinomialModelString = "model{
  # likelihood
  for(j in 1 : N) {
  for(k in 1 : T) {


  log(lambda[j, k]) <- a0 + beta.Base * (log.Base4[j] - log.Base4.bar)
  + beta.Trt * (Trt[j] - Trt.bar)
  + beta.BT * (BT[j] - BT.bar)
  + beta.Age * (log.Age[j] - log.Age.bar)
  + beta.V4 * (V4[k] - V4.bar)
  p[j, k] <- r[j,k]/(r[j,k]+lambda[j,k])
  y[j, k] ~ dnegbin( p[j,k], r[j,k])
  r[j, k] ~ dunif(0.0001,1000)
  }
  BT[j] <- Trt[j] * log.Base4[j] # interaction
  log.Base4[j] <- log(Base[j] / 4)
  log.Age[j] <- log(Age[j])
  }
```

```
  #covariate means:
  log.Age.bar <- mean(log.Age[])
  Trt.bar <- mean(Trt[])
  BT.bar <- mean(BT[])
  log.Base4.bar <- mean(log.Base4[])
  V4.bar <- mean(V4[])
  # priors:

  a0 ~ dnorm(0.0,1.0E-4)
  beta.Base ~ dnorm(0.0,1.0E-4)
  beta.Trt ~ dnorm(0.0,1.0E-4);
  beta.BT ~ dnorm(0.0,1.0E-4)
  beta.Age ~ dnorm(0.0,1.0E-4)
  beta.V4 ~ dnorm(0.0,1.0E-4)

  # re-calculate intercept on original scale:
  alpha0 <- a0 - beta.Base * log.Base4.bar - beta.Trt * Trt.bar
  - beta.BT * BT.bar - beta.Age * log.Age.bar - beta.V4 * V4.bar

}"
writeLines(NegativeBinomialModelString, "NBmodel.txt")
```

JAGS MODEL

```
parametersNB = c("beta.Age", "beta.BT","beta.Base","beta.Trt","beta.V4","a0")

negBinomialJags= jags(model.file = "NBmodel.txt",
                      parameters.to.save =  parametersNB,
                      data = data_jags,
                      n.chains= 3,
                      n.thin = 10,
                      n.burnin = 8000,
                      n.iter = 30000)
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 236
##    Unobserved stochastic nodes: 242
##    Total graph size: 1708
##
## Initializing model
```

```
negBinomialJags
```

```
## Inference for Bugs model at "NBmodel.txt", fit using jags,
##  3 chains, each with 30000 iterations (first 8000 discarded), n.thin = 10
##  n.sims = 6600 iterations saved
##            mu.vect sd.vect     2.5%      25%      50%      75%    97.5%
```

```
## a0          1.632   0.035    1.564    1.609    1.633    1.656    1.698
## beta.Age    0.749   0.135    0.483    0.658    0.747    0.839    1.017
## beta.BT     0.501   0.078    0.346    0.449    0.501    0.553    0.653
## beta.Base   0.951   0.050    0.853    0.917    0.950    0.984    1.049
## beta.Trt   -1.234   0.184   -1.600   -1.360   -1.231   -1.110   -0.874
## beta.V4    -0.066   0.062   -0.186   -0.108   -0.066   -0.024    0.054
## deviance 1334.212  15.569 1304.697 1323.446 1333.834 1344.299 1366.039
##            Rhat n.eff
## a0        1.001  5900
## beta.Age  1.001  5800
## beta.BT   1.001  6600
## beta.Base 1.001  6600
## beta.Trt  1.001  3900
## beta.V4   1.001  6600
## deviance  1.010   240
##
## 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 = 120.2 and DIC = 1454.4
## DIC is an estimate of expected predictive error (lower deviance is better).
```

```
burn = 3000
iter = 35000

update(negBinomialJags$model, burn)


#let's check for convergence
gelman.diag(as.mcmc(negBinomialJags))
```

```
## Potential scale reduction factors:
##
##           Point est. Upper C.I.
## a0                 1       1.00
## beta.Age           1       1.01
## beta.Base          1       1.00
## beta.BT            1       1.00
## beta.Trt           1       1.00
## beta.V4            1       1.00
## deviance           1       1.02
##
## Multivariate psrf
##
## 1.01
```

```
#let's see the autocorrelation
autocorr.diag(as.mcmc(negBinomialJags))
```

```
##                    a0    beta.Age   beta.Base      beta.BT   beta.Trt
## Lag 0    1.0000000000  1.00000000  1.00000000  1.000000000  1.0000000
## Lag 10   0.0868105477  0.11032170  0.36175766  0.674674557  0.6580800
```

```
## Lag 50   0.0493875815  0.05544991  0.11309464  0.164246431  0.1577745
## Lag 100  0.0002080422  0.01628726  0.02796683  0.043103396  0.0431627
## Lag 500 -0.0015428875 -0.01162591 -0.01191788 -0.005559435 -0.0113556
##              beta.V4   deviance
## Lag 0    1.000000000 1.00000000
## Lag 10   0.018838797 0.43590852
## Lag 50   0.024970263 0.21337934
## Lag 100  0.006565404 0.14230966
## Lag 500 -0.001765139 0.04067749
```

```
# i am going to add a thining factor of 10
```

The negative Binomial model needs an higher burn-in value to reach the convergence, with a starting value of 8000 and with a number of iteration equal to 35000 I implemented this model. Both the Rhat and the n.eff values are fine, every parameters reached the convergence and the ESS are big enough. The convergence of the parameters is confirmed by the gelman diagnostic too, and trough the autocorrelation diagnostic I decide to increase the thining factor from 10 to 50.

SIMULATION AND DIC COMPUTATION

```
negBinomialSim= coda.samples(model = negBinomialJags$model,
                             variable.names = parametersNB,
                             n.iter = 30000,
                             thin = 50)

summary(negBinomialSim)
```

```
##
## Iterations = 33050:63000
## Thinning interval = 50
## Number of chains = 3
## Sample size per chain = 600
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##               Mean      SD  Naive SE Time-series SE
## a0         1.63497 0.03601 0.0008487       0.000858
## beta.Age   0.74769 0.13669 0.0032219       0.003519
## beta.BT    0.50925 0.07895 0.0018609       0.002147
## beta.Base  0.94414 0.05077 0.0011966       0.001304
## beta.Trt  -1.25666 0.18665 0.0043994       0.005061
## beta.V4   -0.06596 0.06064 0.0014294       0.001426
##
## 2. Quantiles for each variable:
##
##              2.5%     25%      50%     75%    97.5%
## a0         1.5667  1.6110  1.63546  1.6594  1.70661
## beta.Age   0.4797  0.6539  0.74335  0.8436  1.01702
## beta.BT    0.3539  0.4556  0.51046  0.5620  0.65765
## beta.Base  0.8480  0.9096  0.94382  0.9784  1.04558
```

```
## beta.Trt  -1.6177 -1.3833 -1.25522 -1.1331 -0.88882
## beta.V4   -0.1794 -0.1073 -0.06593 -0.0249  0.05236
```

```
dicNegBin = dic.samples(negBinomialJags$model, n.iter = iter, thin = 50)

dicNegBin
```

```
## Mean deviance:  1345
## penalty 20.61
## Penalized deviance: 1365
```

The DIC value starts dropping out, the negative association of the treatmente variable seems to be really strong, which means that using progabide is associated with a decrease of the seizure count.

2.3 Negative Binomial with Random Effects

As already said before, when repeated responses are observed, correlation can be incorporated in the model via a common random effect for all measurements referring to the same individual. This introduces a marginal correlation between repeated data, while interpretation is based on the conditional means. Let's introduce the Random Effects parameter and define this new model:

$$Y_{j,k} \sim NegBinom(p_{j,k}, r_{j,k})$$

where p_{j,k} is the probability of *"success"* of the patient in that period visit and is equal to

$$r_{j,k}/(r_{j,k} + \lambda_{j,k}$$

and r is the overdispersion parameter

$$r_{j,k} \sim U(0.0001, 1000)$$

To overcome high autocorrelation and helps the convergence of the markov chains I standardized each covariates by its mean, in this way I can ensure approximate prior independence between the regression coefficients:

$$log(\lambda_{j,k}) = \alpha_0 + \beta_1(log(BS4_j) - mean(log(BS_4))) + \beta_2(Trt_j - mean(Trt)) + \beta_3(log(BS4_j) * T_j) + \beta_4(log(Age_j) - mean(log(Age)))$$

where b_1 represent the random effects related to the individual patient:

$$b_{1,j} \sim Norm(0, tau.b_1)$$

with $tau.b_1 \sim Gamma(0.001, 0.001)$

and the its standard deviation equal to $\sigma = \frac{1}{tau.b_1}$

whilst b represent the combined random effects related to the patient and the single visit period:

$$b_{j,k} \sim Norm(0, tau.b)$$

with $tau.b \sim Gamma(0.001, 0.001)$

and its standard deviation equal to $\sigma = \frac{1}{tau.b}$

Prior:

$$B_i \sim Norm(0, 0.001) \qquad \forall \quad i = 1, ..., 5$$

BUGS MODEL

```
NegativeBinomialREModelString = "model{
  # likelihood
  for(j in 1 : N) {
  for(k in 1 : T) {


  log(lambda[j, k]) <- a0 + beta.Base * (log.Base4[j] - log.Base4.bar)
  + beta.Trt * (Trt[j] - Trt.bar)
  + beta.BT * (BT[j] - BT.bar)
  + beta.Age * (log.Age[j] - log.Age.bar)
  + beta.V4 * (V4[k] - V4.bar)
  + b1[j] + b[j, k]

  b[j, k] ~ dnorm(0.0, tau.b); # subject*visit random effects
  p[j,k] <- r[j,k]/(r[j,k]+lambda[j,k])
  y[j, k] ~ dnegbin( p[j,k], r[j,k])
  r[j,k] ~ dunif(0.0001,1000)
  }

  b1[j] ~ dnorm(0.0, tau.b1) # subject random effects
  BT[j] <- Trt[j] * log.Base4[j] # interaction
  log.Base4[j] <- log(Base[j] / 4)
  log.Age[j] <- log(Age[j])
  }

  #covariate means:
  log.Age.bar <- mean(log.Age[])
  Trt.bar <- mean(Trt[])
  BT.bar <- mean(BT[])
  log.Base4.bar <- mean(log.Base4[])
  V4.bar <- mean(V4[])
  # priors:
  #r ~ dunif(0.0001,1000)
  a0 ~ dnorm(0.0,1.0E-4)
  beta.Base ~ dnorm(0.0,1.0E-4)
  beta.Trt ~ dnorm(0.0,1.0E-4);
  beta.BT ~ dnorm(0.0,1.0E-4)
  beta.Age ~ dnorm(0.0,1.0E-4)
  beta.V4 ~ dnorm(0.0,1.0E-4)

  tau.b1 ~ dgamma(1.0E-3,1.0E-3)
  sigma.b1 <- 1.0 / sqrt(tau.b1)
  tau.b ~ dgamma(1.0E-3,1.0E-3)
  sigma.b <- 1.0 / sqrt(tau.b)

# re-calculate intercept on original scale:
  alpha0 <- a0 - beta.Base * log.Base4.bar - beta.Trt * Trt.bar
  - beta.BT * BT.bar - beta.Age * log.Age.bar - beta.V4 * V4.bar

}"
writeLines(NegativeBinomialREModelString, "NBREmodel.txt")
```

JAGS MODEL

```
parametersNBre = c("beta.Age", "beta.BT","beta.Base","beta.Trt","beta.V4","a0", "sigma.b", "sigma.b1",

negBinomialREJags= jags(model.file = "NBREmodel.txt",
                        parameters.to.save =  parametersNBre,
                        data = data_jags,
                        n.chains= 3,
                        n.thin = 10,
                        n.burnin = 8000,
                        n.iter = 30000)
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 236
##    Unobserved stochastic nodes: 539
##    Total graph size: 2247
##
## Initializing model
```

```
negBinomialREJags
```

```
## Inference for Bugs model at "NBREmodel.txt", fit using jags,
##  3 chains, each with 30000 iterations (first 8000 discarded), n.thin = 10
##  n.sims = 6600 iterations saved
##             mu.vect sd.vect     2.5%      25%      50%      75%    97.5%
## a0            1.576   0.078    1.421    1.525    1.578    1.627    1.727
## beta.Age      0.503   0.365   -0.215    0.257    0.503    0.745    1.227
## beta.BT       0.377   0.209   -0.038    0.240    0.373    0.513    0.795
## beta.Base     0.864   0.140    0.589    0.770    0.865    0.958    1.140
## beta.Trt     -1.008   0.408   -1.801   -1.276   -1.005   -0.740   -0.225
## beta.V4      -0.102   0.087   -0.272   -0.160   -0.102   -0.044    0.066
## sigma.b       0.347   0.048    0.255    0.315    0.346    0.378    0.444
## sigma.b1      0.501   0.071    0.376    0.452    0.497    0.545    0.651
## tau.b         8.833   2.767    5.071    7.004    8.345   10.085   15.339
## tau.b1        4.228   1.222    2.360    3.362    4.054    4.900    7.060
## deviance   1047.337  20.248 1008.684 1033.447 1046.982 1060.581 1088.305
##             Rhat n.eff
## a0         1.002  2700
## beta.Age   1.003   830
## beta.BT    1.034    65
## beta.Base  1.020   110
## beta.Trt   1.030    73
## beta.V4    1.001  6600
## sigma.b    1.002  2600
## sigma.b1   1.001  4800
## tau.b      1.002  2600
## tau.b1     1.001  4800
## deviance   1.002  1700
##
```

```
## 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 = 204.8 and DIC = 1252.1
## DIC is an estimate of expected predictive error (lower deviance is better).
```

```
burn = 3000
iter = 35000

update(negBinomialREJags$model, burn)


#let's check for convergence
gelman.diag(as.mcmc(negBinomialREJags))
```

```
## Potential scale reduction factors:
##
##           Point est. Upper C.I.
## a0              1.00       1.01
## beta.Age        1.00       1.01
## beta.Base       1.01       1.05
## beta.BT         1.02       1.06
## beta.Trt        1.02       1.06
## beta.V4         1.00       1.00
## deviance        1.00       1.01
## sigma.b         1.00       1.00
## sigma.b1        1.00       1.00
## tau.b           1.00       1.00
## tau.b1          1.00       1.00
##
## Multivariate psrf
##
## 1.02
```

```
#let's see the autocorrelation
autocorr.diag(as.mcmc(negBinomialREJags))
```

```
##                 a0    beta.Age  beta.Base    beta.BT  beta.Trt
## Lag 0   1.000000000  1.00000000 1.00000000 1.00000000 1.0000000
## Lag 10  0.507589391  0.53742083 0.81372940 0.95030355 0.9146006
## Lag 50  0.008790787  0.07025923 0.45127172 0.77278420 0.7127541
## Lag 100 0.003750349  0.01154782 0.27718209 0.59323334 0.5482986
## Lag 500 0.023381157 -0.02731354 0.02684285 0.05633578 0.0596093
##             beta.V4    deviance      sigma.b    sigma.b1        tau.b
## Lag 0    1.000000000  1.00000000  1.000000000 1.000000000  1.000000000
## Lag 10   0.025177614  0.11411006  0.486025461 0.152838534  0.533330060
## Lag 50  -0.008106076  0.02282315  0.046542593 0.004968676  0.055244677
## Lag 100 -0.014148353 -0.02226635 -0.015164161 0.013402254 -0.018690342
## Lag 500  0.020746508  0.01931551 -0.009427516 0.009941305  0.005710541
##             tau.b1
## Lag 0   1.000000000
## Lag 10  0.162206321
```

```
## Lag 50  0.005669362
## Lag 100 0.018685855
## Lag 500 0.002210448
```

```
# i am going to add a thining factor of 10
```

SIMULATION AND DIC COMPUTATION

```
negBinomialRESim= coda.samples(model = negBinomialREJags$model,
                               variable.names = parametersNBre,
                               n.iter = iter,
                               thin = 50)

summary(negBinomialRESim)
```

```
##
## Iterations = 33050:68000
## Thinning interval = 50
## Number of chains = 3
## Sample size per chain = 700
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##              Mean      SD  Naive SE Time-series SE
## a0         1.5786 0.07730 0.0016868       0.001726
## beta.Age   0.4680 0.37132 0.0081030       0.009063
## beta.BT    0.3490 0.21600 0.0047136       0.013614
## beta.Base  0.8810 0.14086 0.0030739       0.006725
## beta.Trt  -0.9490 0.42440 0.0092611       0.026405
## beta.V4   -0.1040 0.08841 0.0019293       0.001930
## sigma.b    0.3477 0.04563 0.0009958       0.001002
## sigma.b1   0.4996 0.06860 0.0014970       0.001497
## tau.b      8.7273 2.49135 0.0543657       0.059051
## tau.b1     4.2385 1.19335 0.0260411       0.026037
##
## 2. Quantiles for each variable:
##
##               2.5%     25%     50%      75%    97.5%
## a0         1.42500  1.5250  1.5811  1.63216  1.73080
## beta.Age  -0.28323  0.2278  0.4674  0.70507  1.19899
## beta.BT   -0.06899  0.1982  0.3428  0.48857  0.78908
## beta.Base  0.59147  0.7860  0.8844  0.97542  1.15622
## beta.Trt  -1.81526 -1.2362 -0.9403 -0.66121 -0.15562
## beta.V4   -0.28042 -0.1604 -0.1056 -0.04287  0.06517
## sigma.b    0.26472  0.3170  0.3476  0.37667  0.44039
## sigma.b1   0.37947  0.4510  0.4943  0.54291  0.64511
## tau.b      5.15620  7.0481  8.2761  9.94950 14.27043
## tau.b1     2.40286  3.3927  4.0923  4.91564  6.94447
```

```
dicNegBinRE = dic.samples(negBinomialREJags$model, n.iter = iter, thin = 50)

dicNegBinRE
```

```
## Mean deviance:  1047
## penalty 124.8
## Penalized deviance: 1172
```

2.4 Poisson with random effects: Poisson Log-Normal Model

Model formulation:

$$Y_{j,k} \sim Pois(\mu_{j,k})$$

$$log(\mu_{j,k}) = \alpha_0 + \beta_1 log(BS_{4j}) + \beta_2 Trt_j + \beta_3 (log(BS_{4j}) * T_j) + \beta_4 log(Age_j) + \beta_5 V_{4k} + b_{j,k} + b_{1j}$$

that is also equivalent to

$$Y_{j,k} \sim Pois(\lambda_{j,k} = e^b e^{b_1} \mu_{j,k})$$

where, as for the Negative Binomial with Random Effects, b_1 represent the random effects related to the individual patient and b the combined random effects related to the patient and the single visit period:

$$b_{1,j} \sim Norm(0, tau.b_1)$$

with $tau.b_1 \sim Gamma(0.001, 0.001)$

and the its standard deviation equal to $\sigma = \frac{1}{tau.b_1}$

$$b_{j,k} \sim Norm(0, tau.b)$$

with $tau.b \sim Gamma(0.001, 0.001)$

and its standard deviation equal to $\sigma = \frac{1}{tau.b}$

Prior:

$$B_i \sim Norm(0, 0.001) \qquad \forall \quad i = 1, ..., 5$$

BUGS MODEL

```
poissonREModelString = "model{
  for(j in 1 : N) {
  for(k in 1 : T) {
  log(mu[j, k]) <- a0 + beta.Base * (log.Base4[j] - log.Base4.bar)
  + beta.Trt * (Trt[j] - Trt.bar)
  + beta.BT * (BT[j] - BT.bar)
  + beta.Age * (log.Age[j] - log.Age.bar)
  + beta.V4 * (V4[k] - V4.bar)
  + b1[j] + b[j, k]
  y[j, k] ~ dpois(mu[j, k])
  b[j, k] ~ dnorm(0.0, tau.b); # subject*visit random effects
```

```
  }
  b1[j] ~ dnorm(0.0, tau.b1) # subject random effects
  BT[j] <- Trt[j] * log.Base4[j] # interaction
  log.Base4[j] <- log(Base[j] / 4)
  log.Age[j] <- log(Age[j])
  }

  #covariate means:

  log.Age.bar <- mean(log.Age[])
  Trt.bar <- mean(Trt[])
  BT.bar <- mean(BT[])
  log.Base4.bar <- mean(log.Base4[])
  V4.bar <- mean(V4[])

  # priors:
  a0 ~ dnorm(0.0,1.0E-4)
  beta.Base ~ dnorm(0.0,1.0E-4)
  beta.Trt ~ dnorm(0.0,1.0E-4);
  beta.BT ~ dnorm(0.0,1.0E-4)
  beta.Age ~ dnorm(0.0,1.0E-4)
  beta.V4 ~ dnorm(0.0,1.0E-4)
  tau.b1 ~ dgamma(1.0E-3,1.0E-3)
  sigma.b1 <- 1.0 / sqrt(tau.b1)
  tau.b ~ dgamma(1.0E-3,1.0E-3)
  sigma.b <- 1.0/ sqrt(tau.b)

  # re-calculate intercept on original scale:
  alpha0 <- a0 - beta.Base * log.Base4.bar - beta.Trt * Trt.bar
  - beta.BT * BT.bar - beta.Age * log.Age.bar - beta.V4 * V4.bar
}"
writeLines(poissonREModelString, "poissonREmodel.txt")
```

JAGS MODEL

```
parameterPOISre = c("beta.Age", "beta.BT","beta.Base","beta.Trt","beta.V4","a0", "sigma.b", "sigma.b1",

poissonREJags = jags(model.file ="poissonREmodel.txt",
                     parameters.to.save =  parameterPOISre,
                     data = data_jags,
                    n.chains = 3)
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 236
##    Unobserved stochastic nodes: 303
##    Total graph size: 1538
##
## Initializing model
```

```
poissonREJags
```

```
## Inference for Bugs model at "poissonREmodel.txt", fit using jags,
##  3 chains, each with 2000 iterations (first 1000 discarded)
##  n.sims = 3000 iterations saved
##             mu.vect sd.vect     2.5%      25%      50%      75%    97.5%
## a0            1.574   0.080    1.418    1.519    1.575    1.628    1.727
## beta.Age      0.478   0.373   -0.276    0.227    0.480    0.729    1.183
## beta.BT       0.349   0.217   -0.079    0.206    0.346    0.497    0.770
## beta.Base     0.879   0.139    0.614    0.787    0.874    0.968    1.166
## beta.Trt     -0.948   0.430   -1.784   -1.245   -0.951   -0.656   -0.110
## beta.V4      -0.101   0.089   -0.275   -0.162   -0.102   -0.042    0.070
## sigma.b       0.365   0.042    0.285    0.336    0.364    0.392    0.454
## sigma.b1      0.502   0.070    0.378    0.453    0.496    0.545    0.653
## tau.b         7.820   1.870    4.847    6.523    7.533    8.861   12.270
## tau.b1        4.208   1.170    2.345    3.370    4.065    4.875    6.997
## deviance   1036.228  19.651 1000.214 1022.620 1036.244 1049.089 1076.029
##             Rhat n.eff
## a0         1.006   360
## beta.Age   1.001  3000
## beta.BT    1.002  1900
## beta.Base  1.001  3000
## beta.Trt   1.001  3000
## beta.V4    1.013   170
## sigma.b    1.025   140
## sigma.b1   1.004   610
## tau.b      1.025   140
## tau.b1     1.004   610
## deviance   1.010   220
##
## 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 = 191.4 and DIC = 1227.7
## DIC is an estimate of expected predictive error (lower deviance is better).
```

```
burn = 5000
update(poissonREJags$model, burn)


#let's check for convergence
gelman.diag(as.mcmc(poissonREJags))
```

```
## Potential scale reduction factors:
##
##           Point est. Upper C.I.
## a0              1.01       1.02
## beta.Age        1.00       1.00
## beta.Base       1.00       1.00
## beta.BT         1.00       1.00
## beta.Trt        1.00       1.00
## beta.V4         1.01       1.04
```

```
## deviance        1.01       1.03
## sigma.b         1.03       1.07
## sigma.b1        1.00       1.01
## tau.b           1.02       1.06
## tau.b1          1.00       1.01
##
## Multivariate psrf
##
## 1.03
```

```
#let's see the autocorrelation
autocorr.diag(as.mcmc(poissonREJags))
```

```
##                 a0    beta.Age   beta.Base      beta.BT    beta.Trt
## Lag 0  1.000000000  1.00000000 1.000000000  1.000000000  1.00000000
## Lag 1  0.200579813  0.14867456 0.083336508  0.146306432  0.21353002
## Lag 5  0.129661792  0.09320971 0.028677404  0.065373639  0.11450707
## Lag 10 0.064045729  0.02384341 0.020936930  0.031221301  0.06357158
## Lag 50 0.003802036 -0.04616423 0.002296173 -0.006681348 -0.01620699
##           beta.V4    deviance    sigma.b    sigma.b1      tau.b
## Lag 0  1.00000000  1.00000000 1.00000000  1.00000000 1.00000000
## Lag 1  0.47060831  0.50896209 0.75702450  0.50397167 0.74634424
## Lag 5  0.19873090  0.13646383 0.45231918  0.21374011 0.44337743
## Lag 10 0.08592826  0.09129920 0.25301675  0.10417072 0.24772687
## Lag 50 0.04157094 -0.02116388 0.02879932 -0.04291211 0.02434012
##             tau.b1
## Lag 0   1.00000000
## Lag 1   0.52825916
## Lag 5   0.23454967
## Lag 10  0.11715975
## Lag 50 -0.03675341
```

```
# i am going to add a thining factor of 10
```

SIMULATION AND DIC COMPUTATION

```
iter = 30000

poissonRESim= coda.samples(model = poissonREJags$model,
                           variable.names = parameterPOISre,
                           n.iter = iter,
                           thin = 50)

summary(poissonRESim)
```

```
##
## Iterations = 7050:37000
## Thinning interval = 50
## Number of chains = 3
## Sample size per chain = 600
##
```

```
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##               Mean      SD  Naive SE Time-series SE
## a0          1.5754 0.08031 0.0018929       0.001892
## beta.Age    0.4770 0.36024 0.0084908       0.008492
## beta.BT     0.3478 0.21392 0.0050420       0.004892
## beta.Base   0.8792 0.13860 0.0032669       0.003205
## beta.Trt   -0.9487 0.42003 0.0099001       0.009189
## beta.V4    -0.1014 0.08713 0.0020536       0.001936
## sigma.b     0.3603 0.04119 0.0009709       0.001012
## sigma.b1    0.4983 0.07011 0.0016525       0.001698
## tau.b       8.0127 1.87023 0.0440818       0.045355
## tau.b1      4.2745 1.25405 0.0295583       0.030908
##
## 2. Quantiles for each variable:
##
##                2.5%     25%     50%     75%    97.5%
## a0          1.41222  1.5272  1.5748  1.6274  1.73260
## beta.Age   -0.24746  0.2348  0.4792  0.7129  1.20850
## beta.BT    -0.07346  0.2068  0.3484  0.4791  0.78510
## beta.Base   0.60518  0.7872  0.8786  0.9711  1.15781
## beta.Trt   -1.78680 -1.2289 -0.9394 -0.6623 -0.13974
## beta.V4    -0.27267 -0.1595 -0.1010 -0.0451  0.07209
## sigma.b     0.28469  0.3317  0.3587  0.3870  0.44331
## sigma.b1    0.36913  0.4506  0.4922  0.5407  0.64850
## tau.b       5.08849  6.6761  7.7728  9.0871 12.33799
## tau.b1      2.37780  3.4204  4.1276  4.9245  7.33924
```

```
dicPoisRE = dic.samples(poissonREJags$model, n.iter = iter, thin = 50)

dicPoisRE
```

```
## Mean deviance:  1038
## penalty 121.9
## Penalized deviance: 1160
```

2.5 Zero-Inflated Negative Binomial

In this model we have:

$$Y_{j,k} \sim NegBinom(p_{j,k}, r_{j,k})$$

where p[j,k] is the probability of *"success"* of the patient in that period visit and is equal to

$$r_{j,k}/(r_{j,k} + \lambda_{j,k} * (1 - z_{j,k})$$

, r is the overdispersion parameter

$$r_{j,k} \sim U(0.0001, 1000)$$

and

$$z_{j,k} \sim Bern(psi) \quad with \quad psi \sim Unif(0, 1)$$

represent the zero To overcome high autocorrelation and helps the convergence of the markov chains I standardized each covariates by its mean, in this way I can ensure approximate prior independence between the regression coefficients:

$$log(\lambda_{j,k}) = \alpha_0 + \beta_1(log(BS_{4j}) - mean(log(BS_4))) + \beta_2(Trt_j - mean(Trt)) + \beta_3(log(BS_{4j}) * T_j) + \beta_4(log(Age_j) - mean(log(Age)))$$

where b_1 represent the random effects related to the individual patient:

$$b_{1,j} \sim Norm(0, tau.b_1)$$

with $tau.b_1 \sim Gamma(0.001, 0.001)$

and the its standard deviation equal to $\sigma = \frac{1}{tau.b_1}$

whilst b represent the combined random effects related to the patient and the single visit period:

$$b_{j,k} \sim Norm(0, tau.b)$$

with $tau.b \sim Gamma(0.001, 0.001)$

and its standard deviation equal to $\sigma = \frac{1}{tau.b}$

Prior:

$$B_i \sim Norm(0, 0.001) \qquad \forall \quad i = 1, ..., 5$$

```
ZINBmodelString = "model{
 # likelihood
  for(j in 1 : N) {
  for(k in 1 : T) {


  log(lambda[j, k]) <- a0 + beta.Base * (log.Base4[j] - log.Base4.bar)
  + beta.Trt * (Trt[j] - Trt.bar)
  + beta.BT * (BT[j] - BT.bar)
  + beta.Age * (log.Age[j] - log.Age.bar)
  + beta.V4 * (V4[k] - V4.bar)
  + b1[j] + b[j, k]

  b[j, k] ~ dnorm(0.0, tau.b); # subject*visit random effects
  p[j,k] <- r[j,k]/(r[j,k]+lambda[j,k]*(1-z[j,k]))
  y[j, k] ~ dnegbin( p[j,k], r[j,k])
  r[j,k] ~ dunif(0.0001,1000)
  z[j,k] ~ dbern(psi)
  }

  b1[j] ~ dnorm(0.0, tau.b1) # subject random effects
  BT[j] <- Trt[j] * log.Base4[j] # interaction
  log.Base4[j] <- log(Base[j] / 4)
  log.Age[j] <- log(Age[j])
  }

  #covariate means:
  log.Age.bar <- mean(log.Age[])
  Trt.bar <- mean(Trt[])
```

```
  BT.bar <- mean(BT[])
  log.Base4.bar <- mean(log.Base4[])
  V4.bar <- mean(V4[])
  # priors:

  a0 ~ dnorm(0.0,1.0E-4)
  beta.Base ~ dnorm(0.0,1.0E-4)
  beta.Trt ~ dnorm(0.0,1.0E-4);
  beta.BT ~ dnorm(0.0,1.0E-4)
  beta.Age ~ dnorm(0.0,1.0E-4)
  beta.V4 ~ dnorm(0.0,1.0E-4)
  psi ~ dunif(0, 1)

  tau.b1 ~ dgamma(1.0E-3,1.0E-3)
  sigma.b1 <- 1.0 / sqrt(tau.b1)
  tau.b ~ dgamma(1.0E-3,1.0E-3)
  sigma.b <- 1.0/ sqrt(tau.b)

# re-calculate intercept on original scale:
  alpha0 <- a0 - beta.Base * log.Base4.bar - beta.Trt * Trt.bar
  - beta.BT * BT.bar - beta.Age * log.Age.bar - beta.V4 * V4.bar

}"
writeLines(ZINBmodelString, "ZINBmodel.txt")
```

JAGS MODEL

```
parameterZINBre = c("beta.Age", "beta.BT","beta.Base","beta.Trt","beta.V4","a0", "sigma.b", "sigma.b1"

ZINBreJags = jags(model.file = "ZINBmodel.txt",
                  parameters.to.save =  parameterZINBre,
                  data = data_jags,
                  n.chains=3,
                  n.thin = 10 ,
                  n.burnin = 8000,
                  n.iter = 35000)
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 236
##    Unobserved stochastic nodes: 776
##    Total graph size: 2956
##
## Initializing model
```

```
ZINBreJags
```

```
## Inference for Bugs model at "ZINBmodel.txt", fit using jags,
```

```
##  3 chains, each with 35000 iterations (first 8000 discarded), n.thin = 10
##  n.sims = 8100 iterations saved
##            mu.vect sd.vect    2.5%     25%      50%      75%    97.5%
## a0           1.631   0.077   1.473   1.581    1.634    1.683    1.778
## beta.Age     0.429   0.352  -0.261   0.192    0.429    0.666    1.123
## beta.BT      0.313   0.196  -0.074   0.188    0.312    0.441    0.708
## beta.Base    0.880   0.133   0.613   0.790    0.880    0.969    1.139
## beta.Trt    -0.914   0.396  -1.699  -1.178   -0.910   -0.650   -0.118
## beta.V4     -0.106   0.084  -0.268  -0.162   -0.106   -0.050    0.058
## psi          0.039   0.018   0.010   0.026    0.037    0.050    0.079
## sigma.b      0.301   0.049   0.203   0.271    0.301    0.333    0.398
## sigma.b1     0.478   0.069   0.358   0.429    0.472    0.520    0.628
## tau.b       12.093   5.200   6.308   8.994   11.053   13.658   24.323
## tau.b1       4.660   1.373   2.536   3.694    4.487    5.422    7.800
## deviance  1013.046  22.032 971.360 998.221 1012.243 1027.071 1058.640
##            Rhat n.eff
## a0        1.002  1400
## beta.Age  1.001  8100
## beta.BT   1.015   210
## beta.Base 1.011   270
## beta.Trt  1.012   280
## beta.V4   1.001  8100
## psi       1.001  4100
## sigma.b   1.004  1300
## sigma.b1  1.001  6700
## tau.b     1.004  1300
## tau.b1    1.001  6700
## deviance  1.001  4000
##
## 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 = 242.7 and DIC = 1255.7
## DIC is an estimate of expected predictive error (lower deviance is better).
```

```
burn = 10000
update(ZINBreJags$model, burn)


#let's check for convergence
gelman.diag(as.mcmc(ZINBreJags))
```

```
## Potential scale reduction factors:
##
##           Point est. Upper C.I.
## a0              1.00       1.01
## beta.Age        1.00       1.00
## beta.Base       1.02       1.05
## beta.BT         1.03       1.07
## beta.Trt        1.02       1.06
## beta.V4         1.00       1.00
## deviance        1.00       1.00
## psi             1.00       1.00
```

```
## sigma.b         1.00       1.01
## sigma.b1        1.00       1.00
## tau.b           1.04       1.05
## tau.b1          1.00       1.00
##
## Multivariate psrf
##
## 1.02
```

```
#let's see the autocorrelation
autocorr.diag(as.mcmc(ZINBreJags))
```

```
##                   a0     beta.Age  beta.Base   beta.BT   beta.Trt
## Lag 0    1.000000000 1.0000000000 1.00000000 1.0000000 1.00000000
## Lag 10   0.500304020 0.5200775923 0.80225423 0.9433575 0.90676311
## Lag 50   0.045497053 0.0708766579 0.43196190 0.7503728 0.68806472
## Lag 100  0.001710350 0.0388299534 0.27559978 0.5528675 0.51416161
## Lag 500 -0.002580782 0.0008357945 0.03261586 0.0487620 0.03986502
##              beta.V4     deviance          psi     sigma.b   sigma.b1
## Lag 0    1.000000000  1.000000000  1.000000000  1.00000000 1.00000000
## Lag 10   0.040367978  0.071467541  0.044145094  0.60417115 0.16233943
## Lag 50  -0.006312995  0.018105131  0.007929240  0.15329007 0.01172869
## Lag 100  0.018156391 -0.012325822 -0.004411985  0.04534983 0.02113615
## Lag 500  0.005256267  0.009540987 -0.016541412 -0.01110158 0.00871416
##               tau.b      tau.b1
## Lag 0    1.00000000 1.000000000
## Lag 10   0.70292814 0.171802887
## Lag 50   0.27490405 0.011061546
## Lag 100  0.08762315 0.013621564
## Lag 500 -0.01940319 0.006601413
```

```
# i am going to add a thining factor of 10
```

SIMULATION AND DIC COMPUTATION

```
ZINBreSim= coda.samples(model = ZINBreJags$model,
                            variable.names = parameterZINBre,
                            n.iter = iter,
                            thin = 10)

summary(ZINBreSim)
```

```
##
## Iterations = 45010:75000
## Thinning interval = 10
## Number of chains = 3
## Sample size per chain = 3000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
```

```
##                Mean      SD  Naive SE Time-series SE
## a0          1.63262 0.07644 0.0008057      0.0013793
## beta.Age    0.42149 0.35152 0.0037053      0.0070489
## beta.BT     0.30489 0.20125 0.0021214      0.0133511
## beta.Base   0.88743 0.13387 0.0014111      0.0056343
## beta.Trt   -0.89651 0.40317 0.0042498      0.0245651
## beta.V4    -0.10521 0.08244 0.0008689      0.0008907
## psi         0.03882 0.01726 0.0001819      0.0001996
## sigma.b     0.30244 0.04720 0.0004975      0.0010723
## sigma.b1    0.47844 0.06939 0.0007314      0.0008925
## tau.b      11.84077 4.29232 0.0452451      0.1004435
## tau.b1      4.65411 1.40372 0.0147965      0.0182610
##
## 2. Quantiles for each variable:
##
##                2.5%      25%      50%      75%    97.5%
## a0          1.47985  1.58197  1.63423  1.68320  1.77815
## beta.Age   -0.28241  0.19465  0.42596  0.65429  1.11029
## beta.BT    -0.09344  0.17099  0.30255  0.44060  0.69659
## beta.Base   0.62328  0.79747  0.88910  0.97829  1.14890
## beta.Trt   -1.68988 -1.16763 -0.89337 -0.62873 -0.10108
## beta.V4    -0.26673 -0.16015 -0.10376 -0.05040  0.05346
## psi         0.01048  0.02642  0.03704  0.04936  0.07746
## sigma.b     0.20922  0.27072  0.30235  0.33352  0.39585
## sigma.b1    0.35530  0.43031  0.47428  0.52226  0.62616
## tau.b       6.38182  8.99006 10.93926 13.64406 22.84408
## tau.b1      2.55053  3.66632  4.44561  5.40044  7.92153
```

```
dicZinbRE = dic.samples(ZINBreJags$model, n.iter = iter, thin = 10)

dicZinbRE
```

```
## Mean deviance:  1013
## penalty NaN
## Penalized deviance: NaN
```

2.6 Zero-Inflated Poisson

BUGS MODEL

```
ZIPmodelString = "model{
 for(j in 1 : N) {
 for(k in 1 : T) {
 mu[j,k] <- lambda[j,k] *(1- z[j, k])+ z[j, k] *0.00001
  log(lambda[j, k]) <- a0 + beta.Base * (log.Base4[j] - log.Base4.bar)
  + beta.Trt * (Trt[j] - Trt.bar)
  + beta.BT * (BT[j] - BT.bar)
  + beta.Age * (log.Age[j] - log.Age.bar)
  + beta.V4 * (V4[k] - V4.bar)
  + b1[j] + b[j, k]
  y[j, k] ~ dpois(mu[j, k])
```

```
  b[j, k] ~ dnorm(0.0, tau.b); # subject*visit random effects
  z[j,k] ~ dbern(psi)
  }
  b1[j] ~ dnorm(0.0, tau.b1) # subject random effects
  BT[j] <- Trt[j] * log.Base4[j] # interaction
  log.Base4[j] <- log(Base[j] / 4)
  log.Age[j] <- log(Age[j])
  }

  #covariate means:
  log.Age.bar <- mean(log.Age[])
  Trt.bar <- mean(Trt[])
  BT.bar <- mean(BT[])
  log.Base4.bar <- mean(log.Base4[])
  V4.bar <- mean(V4[])
  # priors:
  a0 ~ dnorm(0.0,1.0E-4)
  beta.Base ~ dnorm(0.0,0.01)
  beta.Trt ~ dnorm(0.0,1.0E-2);
  beta.BT ~ dnorm(0.0,1.0E-2)
  beta.Age ~ dnorm(0.0,1.0E-2)
  beta.V4 ~ dnorm(0.0,1.0E-2)
  tau.b1 ~ dgamma(1.0E-3,1.0E-3); sigma.b1 <- 1.0 / sqrt(tau.b1)
  tau.b ~ dgamma(1.0E-3,1.0E-3); sigma.b <- 1.0/ sqrt(tau.b)
  psi ~ dunif(0, 1)

  # re-calculate intercept on original scale:
  alpha0 <- a0 - beta.Base * log.Base4.bar - beta.Trt * Trt.bar
  - beta.BT * BT.bar - beta.Age * log.Age.bar - beta.V4 * V4.bar
}"
writeLines(ZIPmodelString, "ZIPmodel.txt")
```

JAGS MODEL

```
parameterZIPre = c("beta.Age", "beta.BT","beta.Base","beta.Trt","beta.V4","a0", "sigma.b", "sigma.b1",

ZIPreJags = jags(model.file = "ZIPmodel.txt",
                 parameters.to.save =  parameterZIPre,
                 data = data_jags,
                 n.chains=3,
                 n.thin = 10 ,
                 n.burnin = 5000,
                 n.iter = 25000)
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 236
##    Unobserved stochastic nodes: 540
##    Total graph size: 2721
```

```
##
## Initializing model
```

```
ZIPreJags
```

```
## Inference for Bugs model at "ZIPmodel.txt", fit using jags,
##  3 chains, each with 25000 iterations (first 5000 discarded), n.thin = 10
##  n.sims = 6000 iterations saved
##             mu.vect sd.vect    2.5%     25%      50%      75%    97.5%
## a0            1.624   0.077   1.467   1.572    1.626    1.676    1.775
## beta.Age      0.419   0.353  -0.288   0.185    0.417    0.649    1.132
## beta.BT       0.290   0.207  -0.117   0.151    0.290    0.421    0.711
## beta.Base     0.888   0.133   0.624   0.803    0.889    0.977    1.152
## beta.Trt     -0.873   0.409  -1.700  -1.133   -0.869   -0.601   -0.089
## beta.V4      -0.104   0.084  -0.267  -0.160   -0.105   -0.048    0.063
## psi           0.039   0.017   0.010   0.027    0.037    0.050    0.079
## sigma.b       0.322   0.042   0.245   0.293    0.321    0.349    0.409
## sigma.b1      0.477   0.069   0.356   0.429    0.472    0.521    0.621
## tau.b        10.143   2.786   5.986   8.202    9.722   11.630   16.623
## tau.b1        4.677   1.399   2.591   3.687    4.498    5.444    7.907
## deviance   1001.281  21.262 960.499 986.554 1000.981 1015.080 1043.912
##             Rhat n.eff
## a0         1.004   690
## beta.Age   1.006   390
## beta.BT    1.030    84
## beta.Base  1.015   230
## beta.Trt   1.029    88
## beta.V4    1.001  6000
## psi        1.001  5200
## sigma.b    1.002  2700
## sigma.b1   1.001  4500
## tau.b      1.002  2700
## tau.b1     1.001  4500
## deviance   1.002  1400
##
## 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 = 225.8 and DIC = 1227.1
## DIC is an estimate of expected predictive error (lower deviance is better).
```

```
burn = 10000
update(ZIPreJags$model, burn)


#let's check for convergence
gelman.diag(as.mcmc(ZIPreJags))
```

```
## Potential scale reduction factors:
##
##           Point est. Upper C.I.
## a0              1.00       1.01
```

```
## beta.Age        1.01       1.03
## beta.Base       1.02       1.05
## beta.BT         1.06       1.18
## beta.Trt        1.06       1.18
## beta.V4         1.00       1.00
## deviance        1.00       1.01
## psi             1.00       1.00
## sigma.b         1.00       1.00
## sigma.b1        1.00       1.00
## tau.b           1.00       1.00
## tau.b1          1.00       1.00
##
## Multivariate psrf
##
## 1.05
```

```
#let's see the autocorrelation
autocorr.diag(as.mcmc(ZIPreJags))
```

```
##                  a0    beta.Age  beta.Base    beta.BT   beta.Trt
## Lag 0    1.00000000 1.000000000 1.00000000 1.00000000 1.00000000
## Lag 10   0.52709640 0.539476080 0.82133522 0.95493338 0.92019572
## Lag 50   0.04531059 0.078069627 0.47584852 0.79653793 0.72500142
## Lag 100 -0.01001113 0.018753439 0.33218224 0.63220100 0.58303559
## Lag 500 -0.02559044 0.007516367 0.06449013 0.08645023 0.08837808
##              beta.V4     deviance          psi       sigma.b     sigma.b1
## Lag 0    1.000000000  1.000000000  1.000000000  1.0000000000  1.000000000
## Lag 10   0.041531390  0.009911159  0.027670840  0.4428660285  0.149757205
## Lag 50  -0.010284030  0.008635539  0.007842058  0.0227381265  0.036549872
## Lag 100  0.002754400 -0.013374444 -0.010203016  0.0006828865 -0.004828793
## Lag 500 -0.006740524  0.005789238  0.003621168 -0.0039516391  0.002433669
##                tau.b      tau.b1
## Lag 0    1.000000000 1.000000000
## Lag 10   0.447361989 0.146390131
## Lag 50   0.021165488 0.025819725
## Lag 100  0.006366401 0.005479905
## Lag 500 -0.013720095 0.003499450
```

```
# i am going to add a thining factor of 10
```

SIMULATION AND DIC COMPUTATION

```
ZIPreSim= coda.samples(model = ZIPreJags$model,
                           variable.names = parameterZIPre,
                           n.iter = iter,
                           thin = 10)

summary(ZIPreSim)
```

```
##
## Iterations = 35010:65000
```

```
## Thinning interval = 10
## Number of chains = 3
## Sample size per chain = 3000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##               Mean      SD  Naive SE Time-series SE
## a0         1.62884 0.07598 0.0008009      0.0014373
## beta.Age   0.42387 0.35524 0.0037446      0.0070868
## beta.BT    0.28757 0.22125 0.0023322      0.0167927
## beta.Base  0.88745 0.14302 0.0015075      0.0073472
## beta.Trt  -0.86339 0.43514 0.0045868      0.0308094
## beta.V4   -0.10557 0.08398 0.0008852      0.0009141
## psi        0.03941 0.01731 0.0001824      0.0002040
## sigma.b    0.32106 0.04148 0.0004372      0.0007235
## sigma.b1   0.47779 0.06957 0.0007334      0.0009206
## tau.b     10.21166 2.77847 0.0292876      0.0496221
## tau.b1     4.66711 1.39634 0.0147187      0.0183925
##
## 2. Quantiles for each variable:
##
##               2.5%     25%      50%      75%    97.5%
## a0         1.48077  1.5785  1.62946  1.67885  1.77698
## beta.Age  -0.26902  0.1861  0.42335  0.66489  1.11435
## beta.BT   -0.19734  0.1572  0.30192  0.43405  0.68472
## beta.Base  0.62159  0.7911  0.88244  0.97603  1.18563
## beta.Trt  -1.66182 -1.1543 -0.88836 -0.59412  0.08836
## beta.V4   -0.26848 -0.1625 -0.10623 -0.05049  0.06264
## psi        0.01112  0.0268  0.03749  0.05010  0.07795
## sigma.b    0.24506  0.2924  0.31963  0.34832  0.40457
## sigma.b1   0.35424  0.4283  0.47300  0.52090  0.62833
## tau.b      6.10957  8.2423  9.78803 11.69851 16.65099
## tau.b1     2.53296  3.6854  4.46977  5.45151  7.96915
```

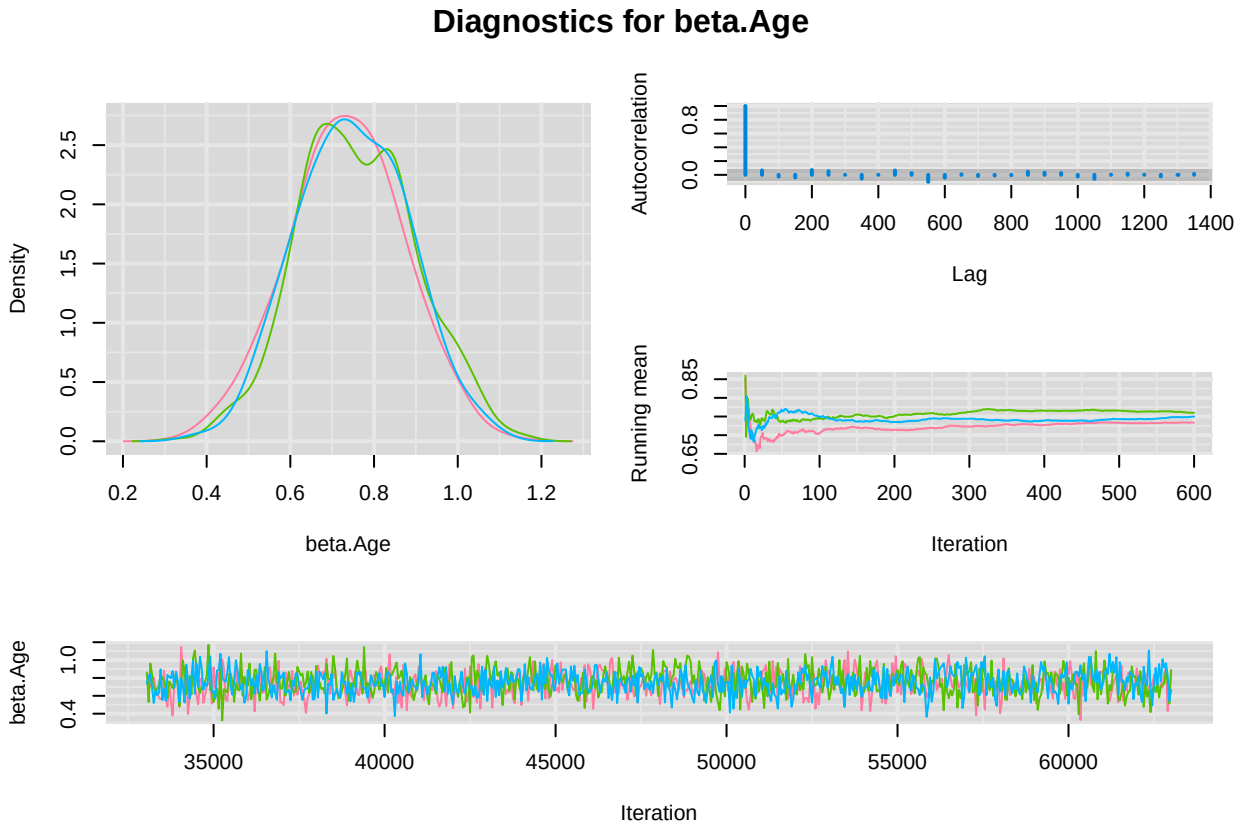
```
dicZipRE = dic.samples(ZIPreJags$model, n.iter = iter, thin = 10)

dicZipRE
```

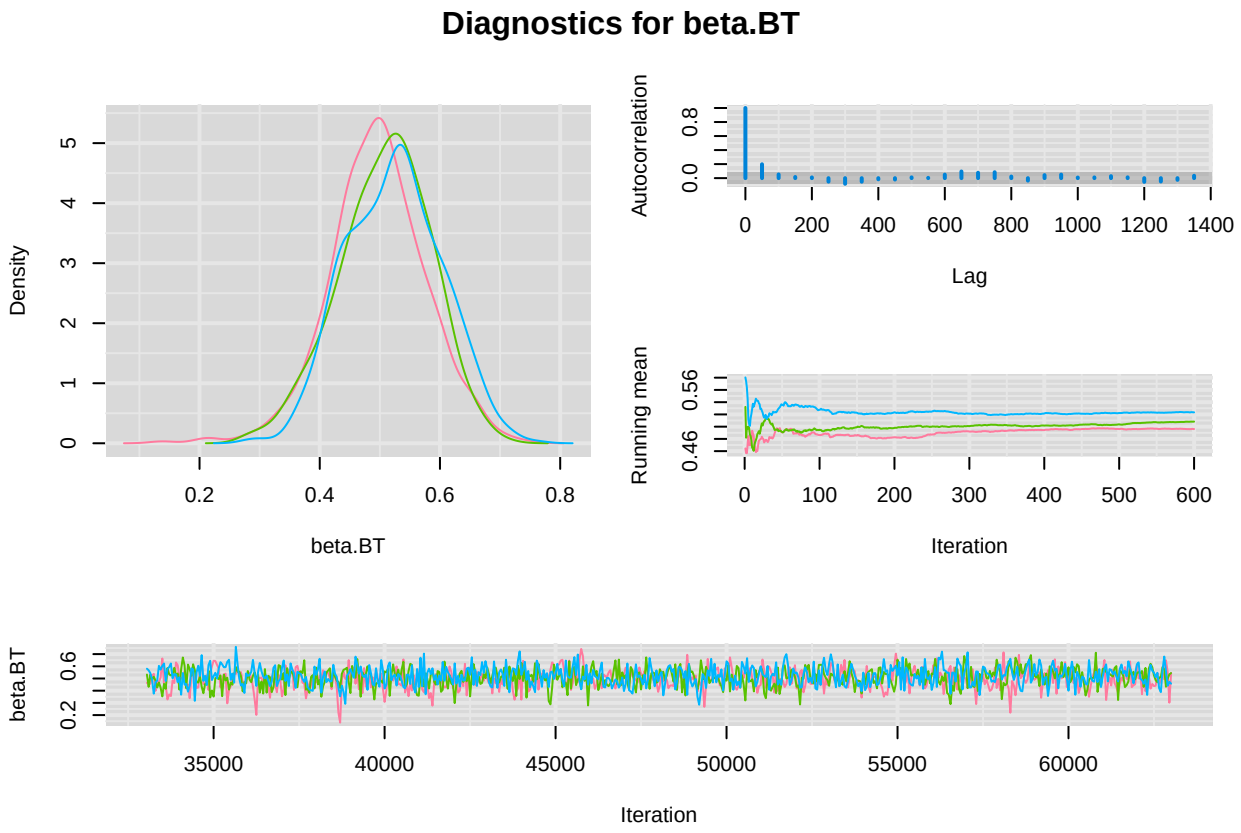
```
## Mean deviance:  1001
## penalty 212.1
## Penalized deviance: 1214
```

3. MCMC Diagnostic

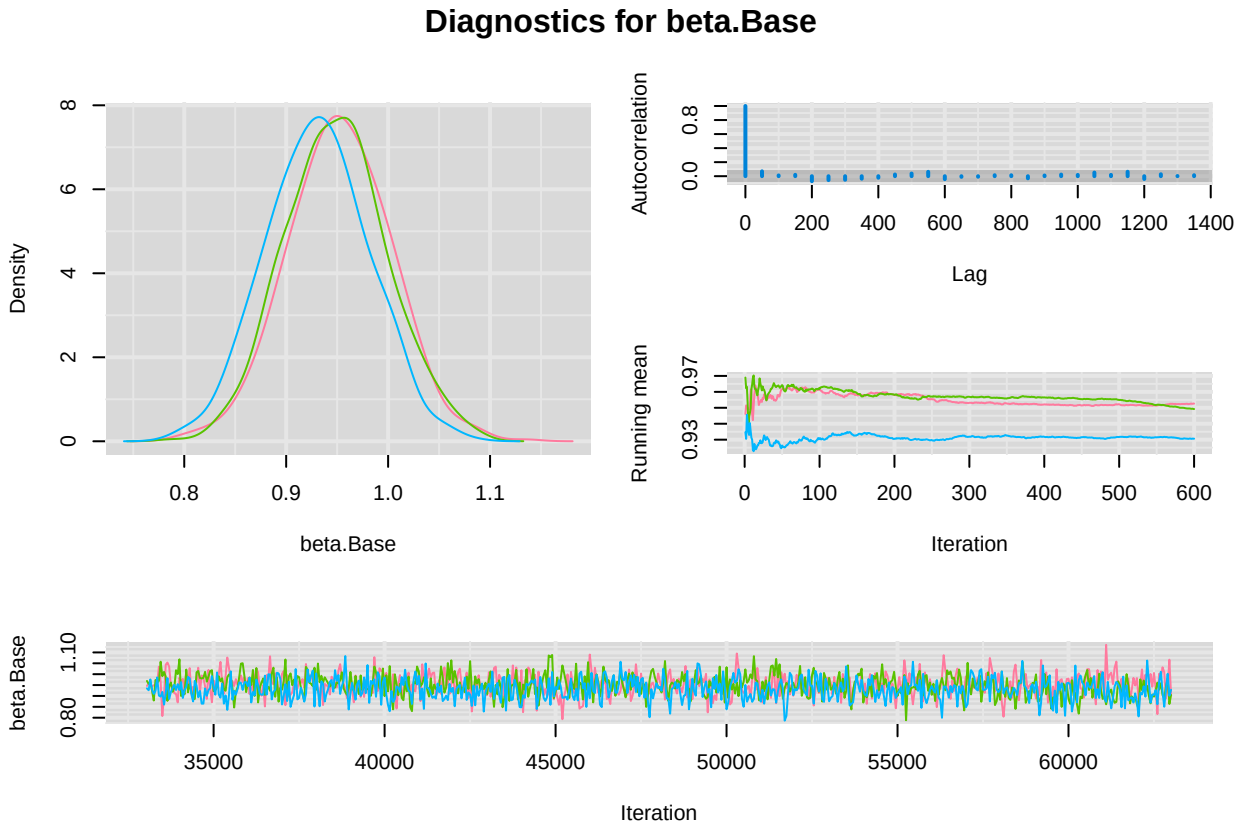
```
library(mcmcplots)
mcmcplot1(negBinomialSim[,"beta.Age",drop=FALSE])
```



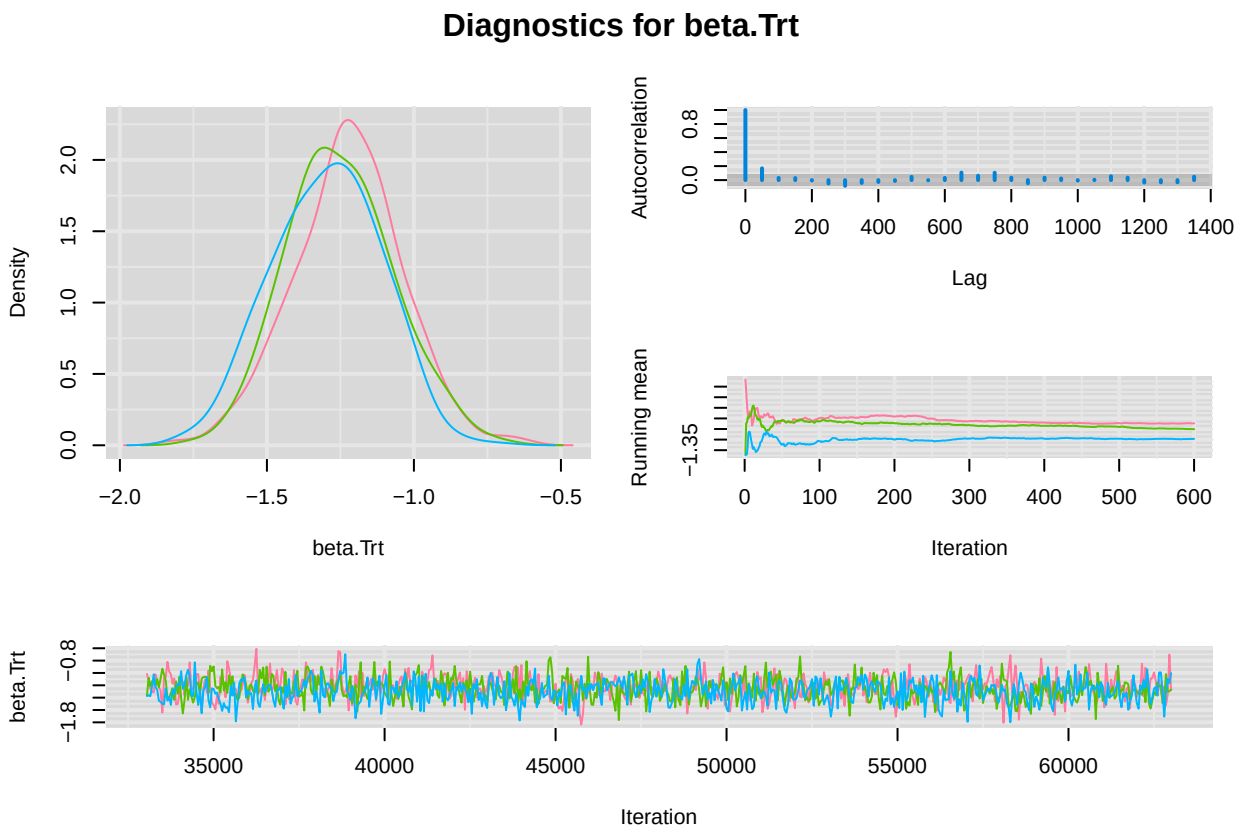

```
mcmcplot1(negBinomialSim[,"beta.BT",drop=FALSE])
```



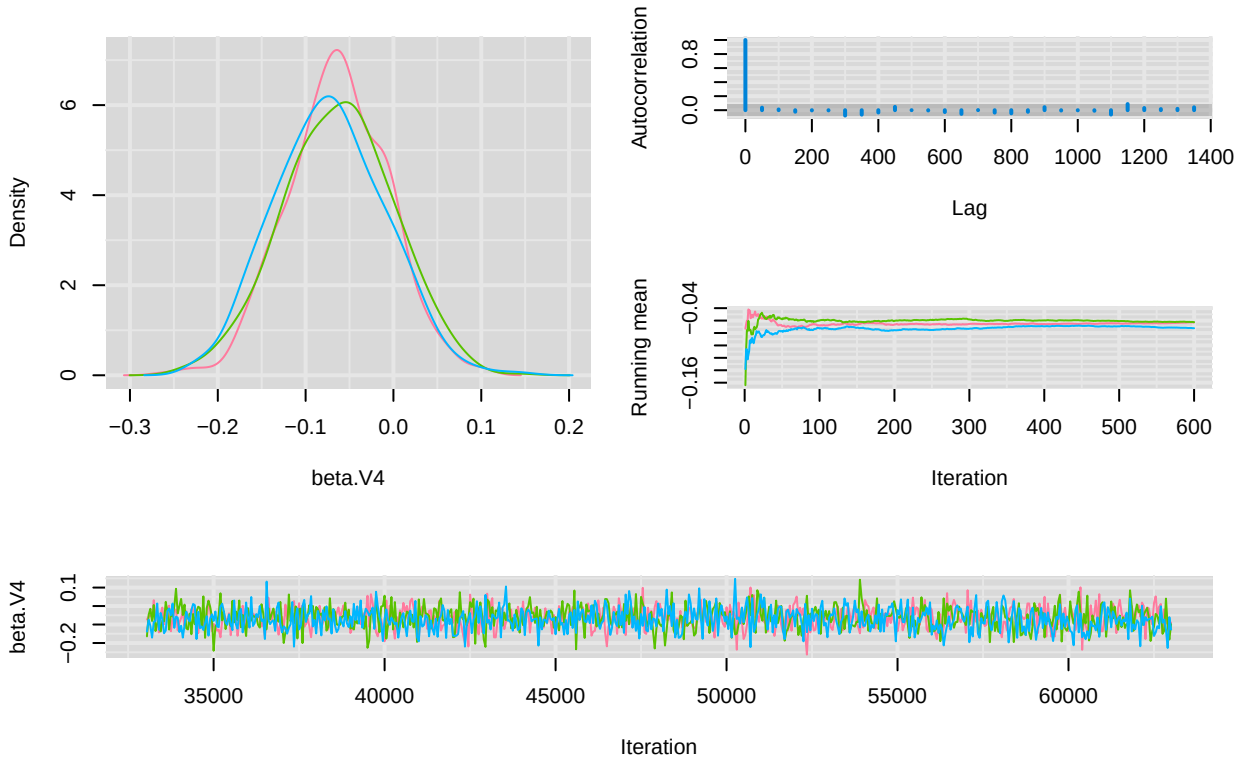

```
mcmcplot1(negBinomialSim[,"beta.Base",drop=FALSE])
```



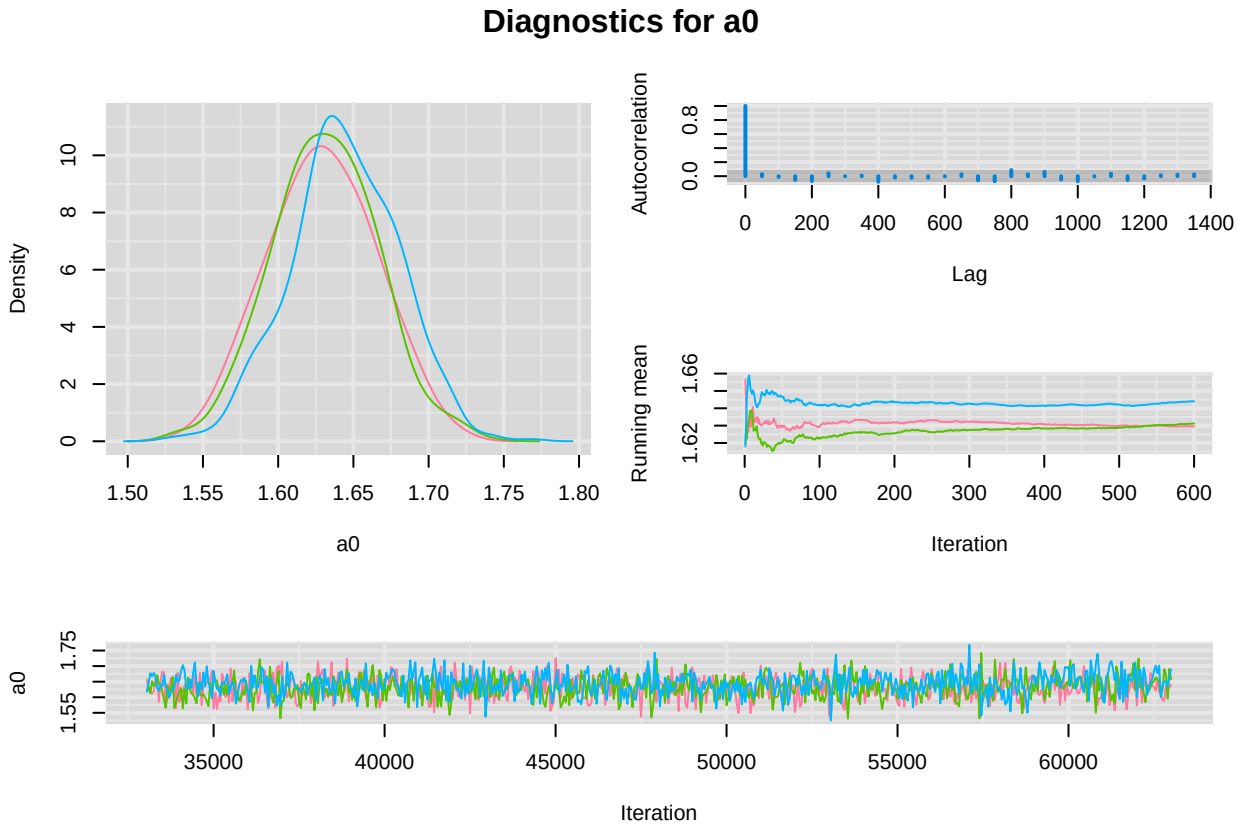

```
mcmcplot1(negBinomialSim[,"beta.Trt",drop=FALSE])
```



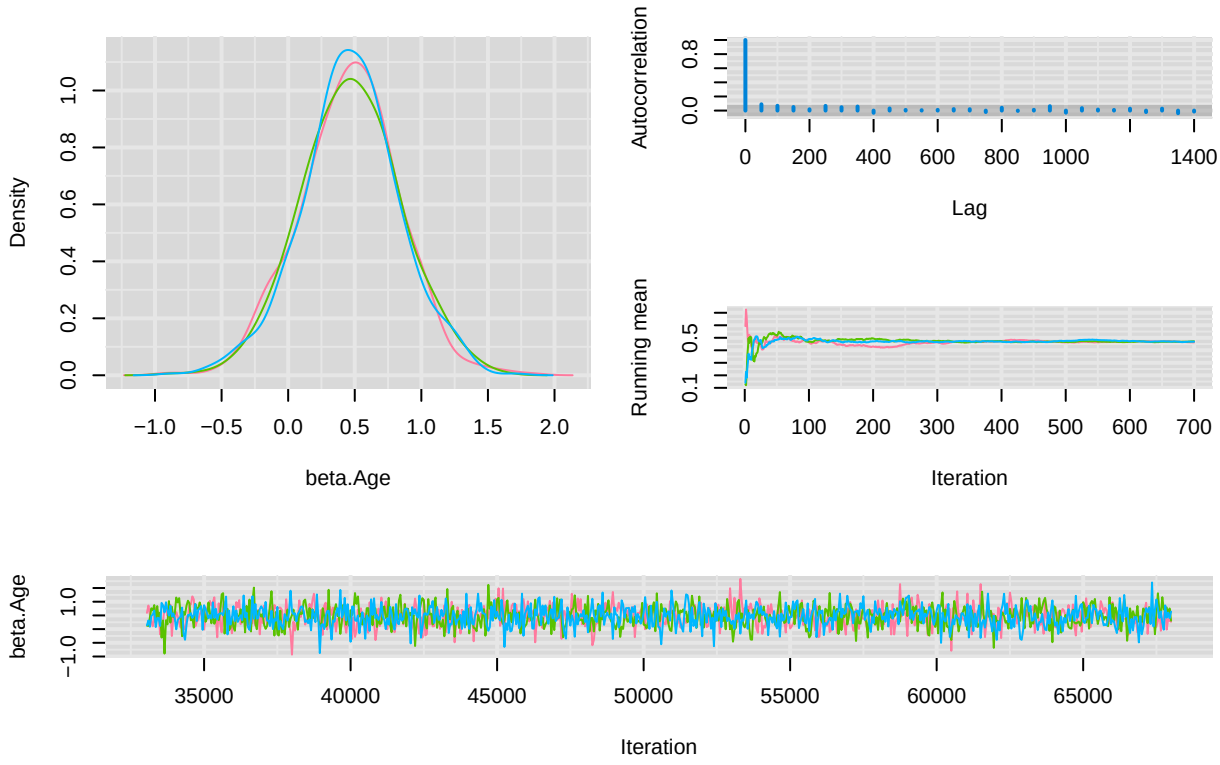

```
mcmcplot1(negBinomialSim[,"beta.V4",drop=FALSE])
```



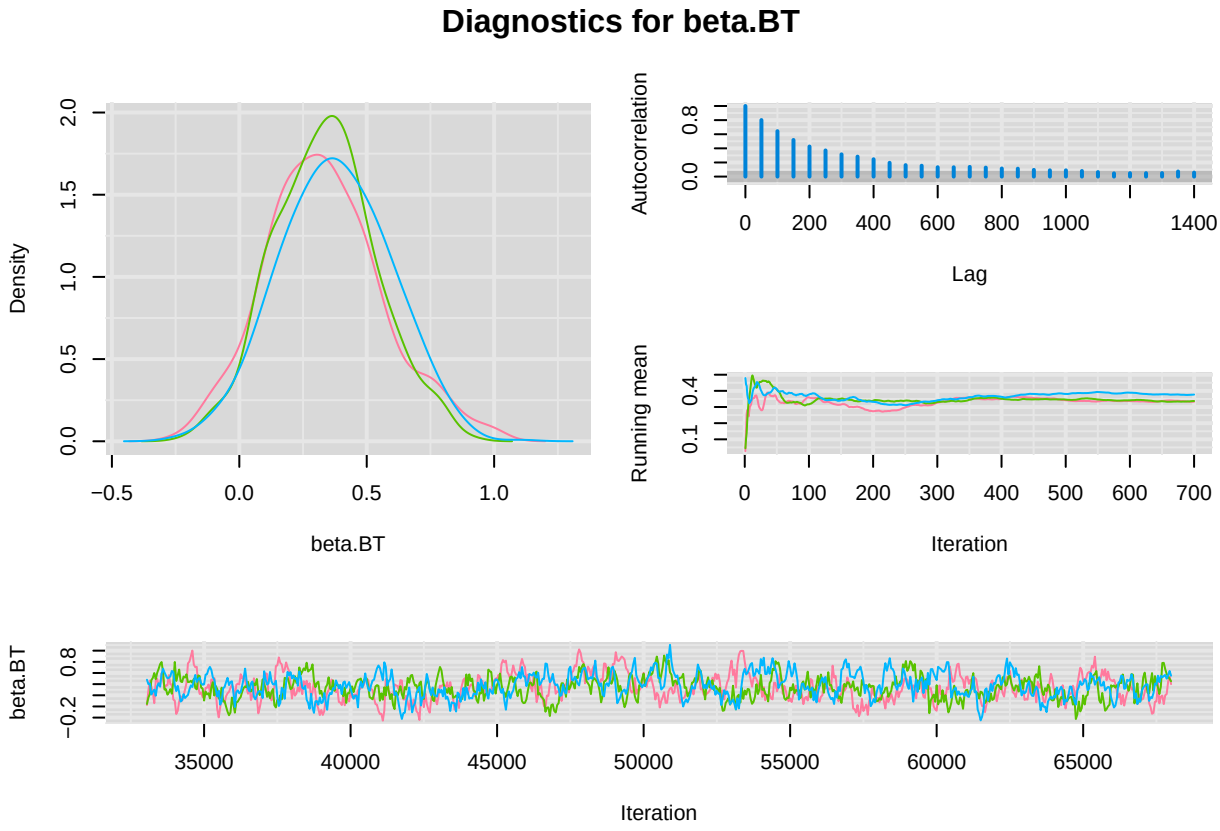

```
mcmcplot1(negBinomialSim[,"a0",drop=FALSE])
```



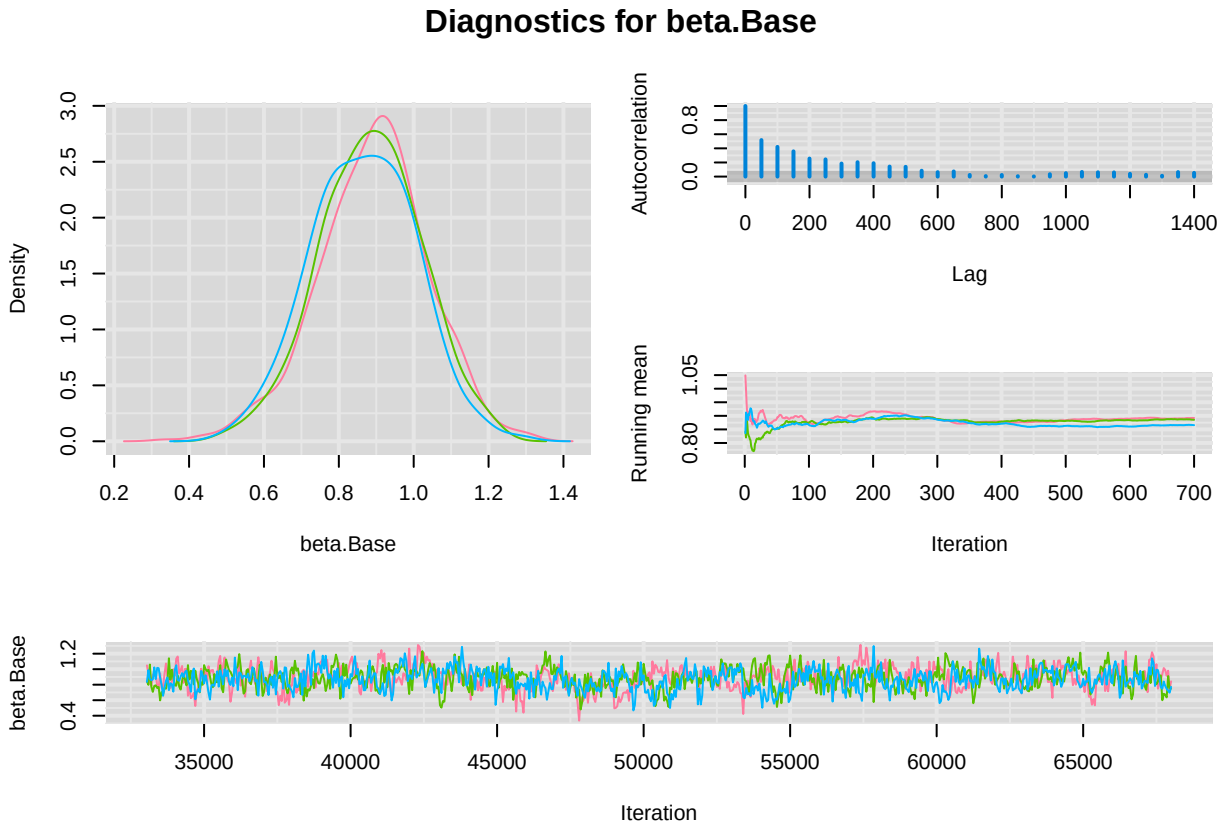

```
mcmcplot1(negBinomialRESim[,"beta.Age",drop=FALSE])
```



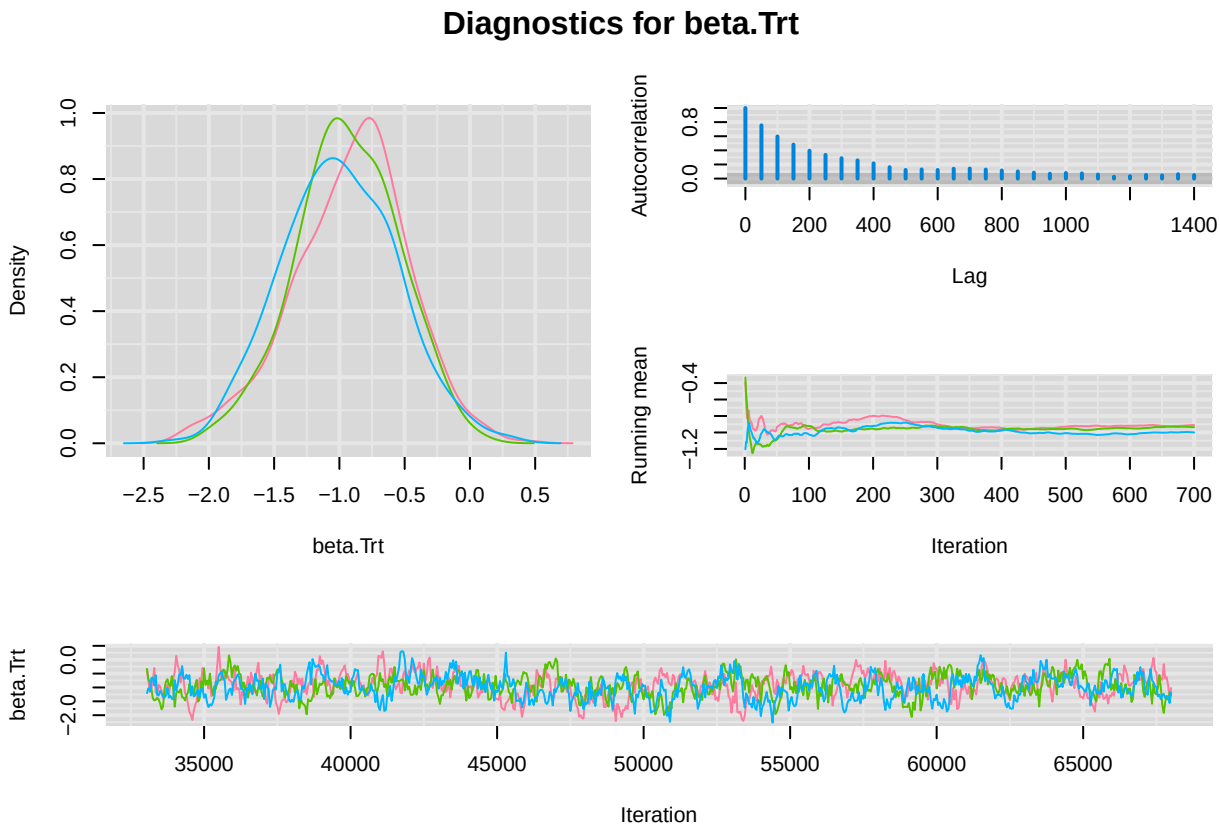

```
mcmcplot1(negBinomialRESim[,"beta.BT",drop=FALSE])
```



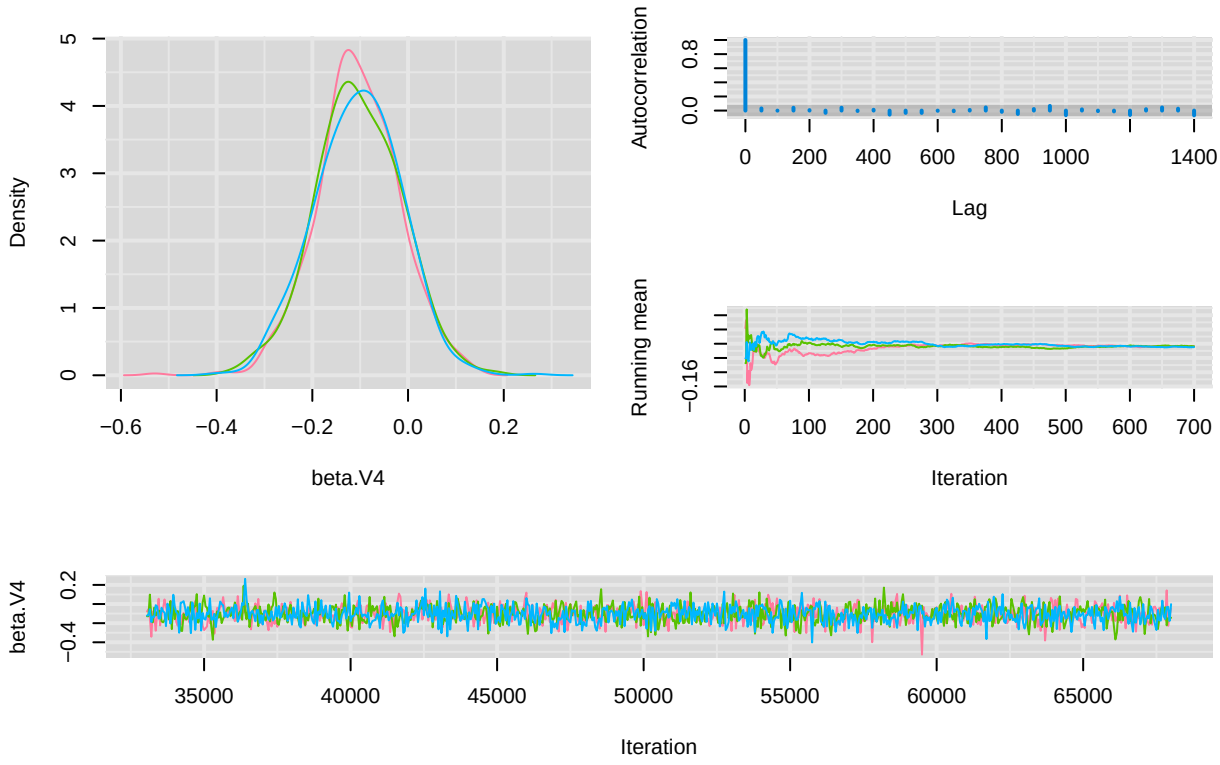

```
mcmcplot1(negBinomialRESim[,"beta.Base",drop=FALSE])
```



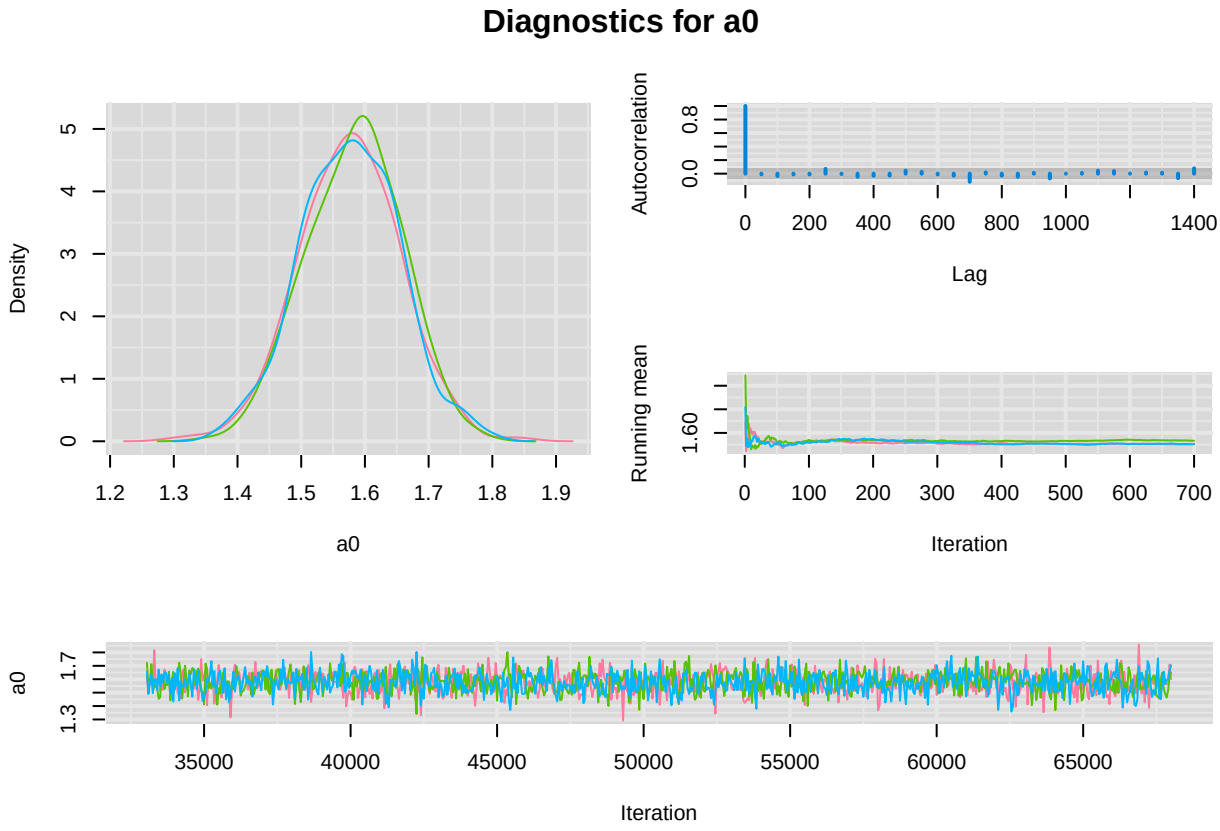

```
mcmcplot1(negBinomialRESim[,"beta.Trt",drop=FALSE])
```



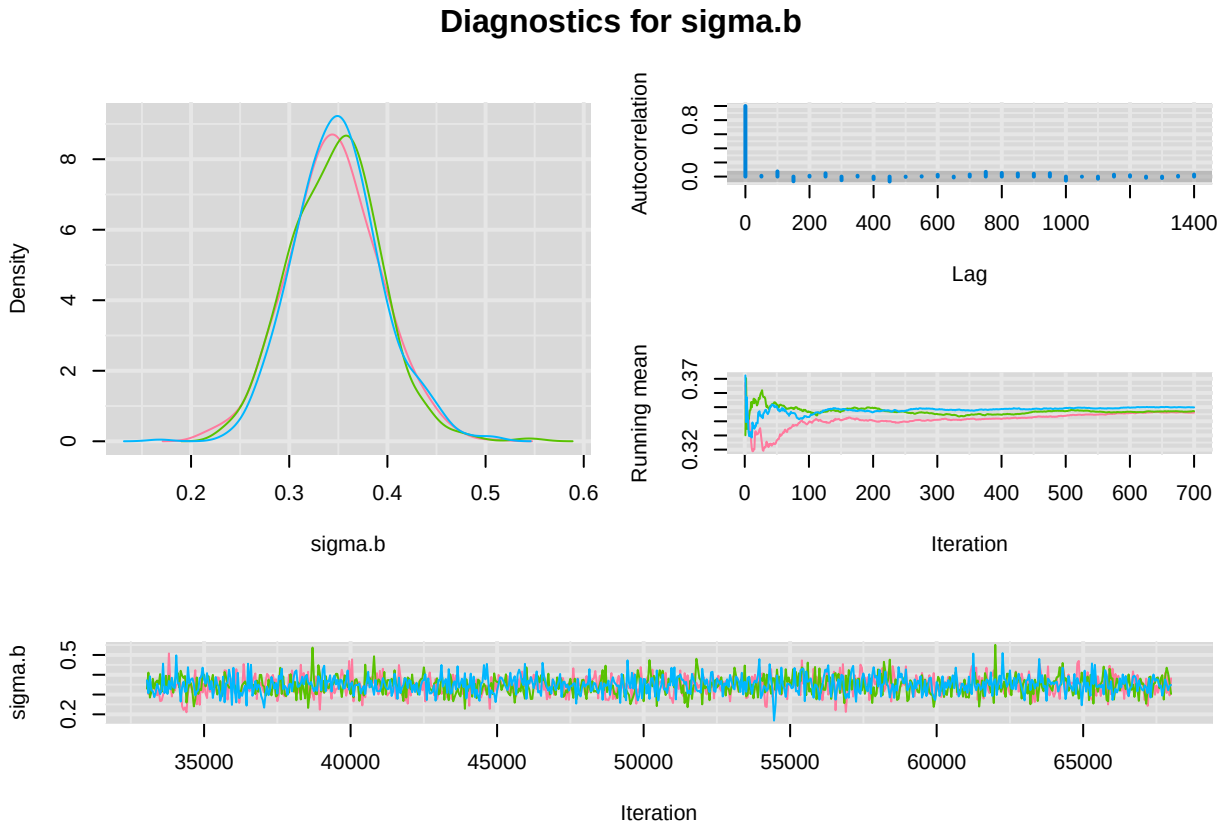

```
mcmcplot1(negBinomialRESim[,"beta.V4",drop=FALSE])
```



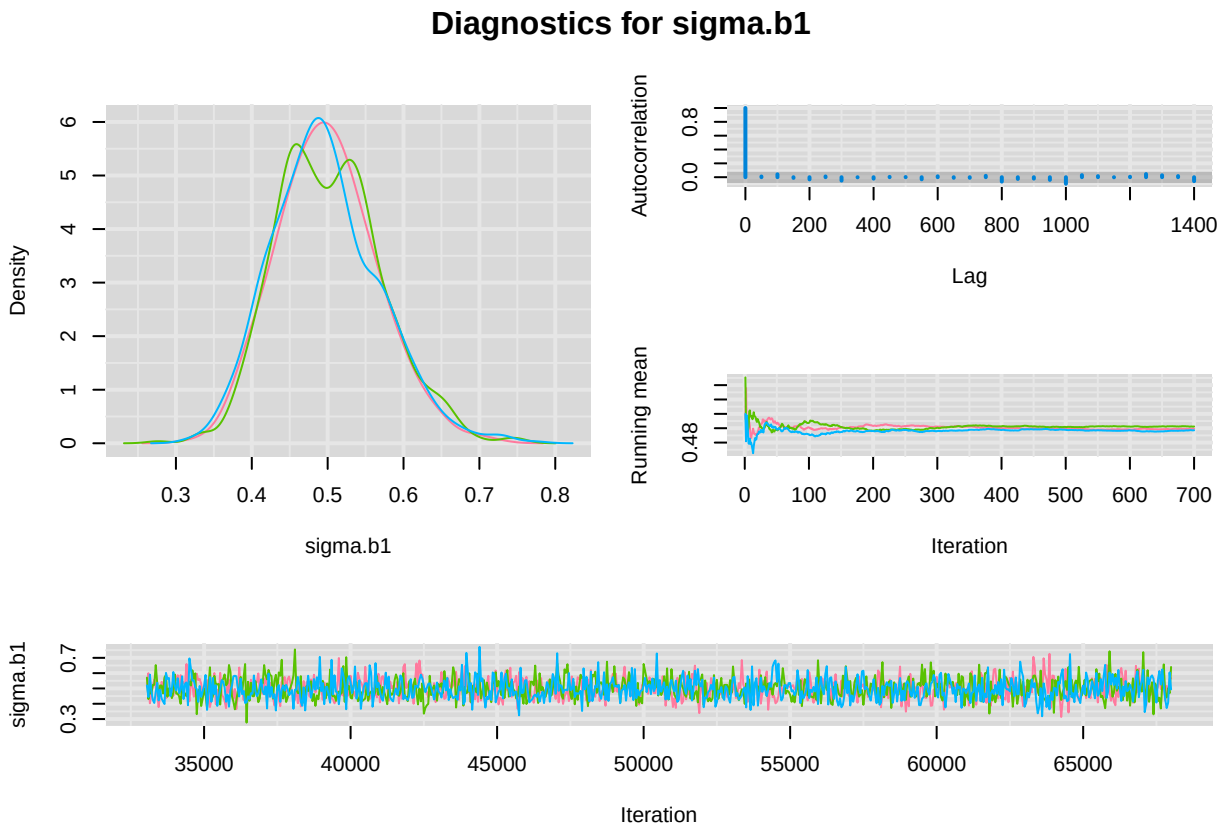

```
mcmcplot1(negBinomialRESim[,"a0",drop=FALSE])
```



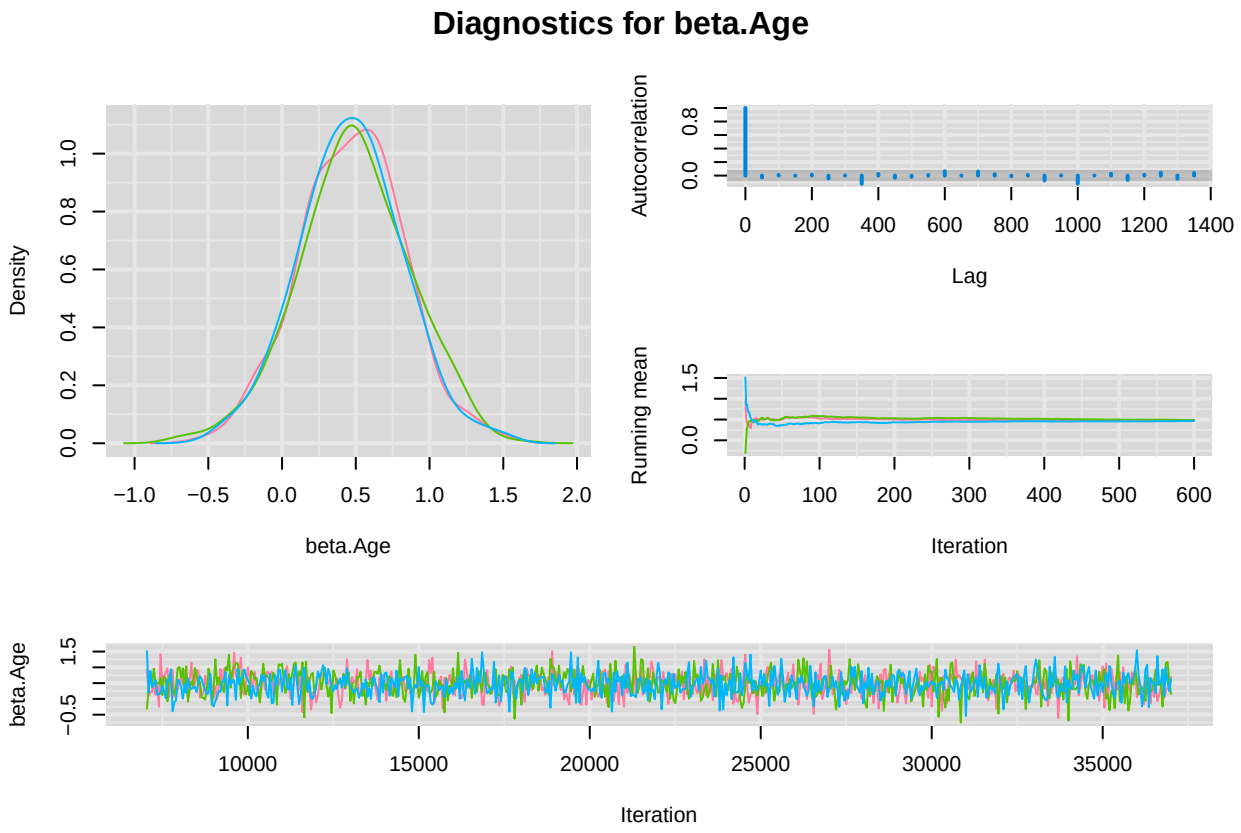

```
mcmcplot1(negBinomialRESim[,"sigma.b",drop=FALSE])
```



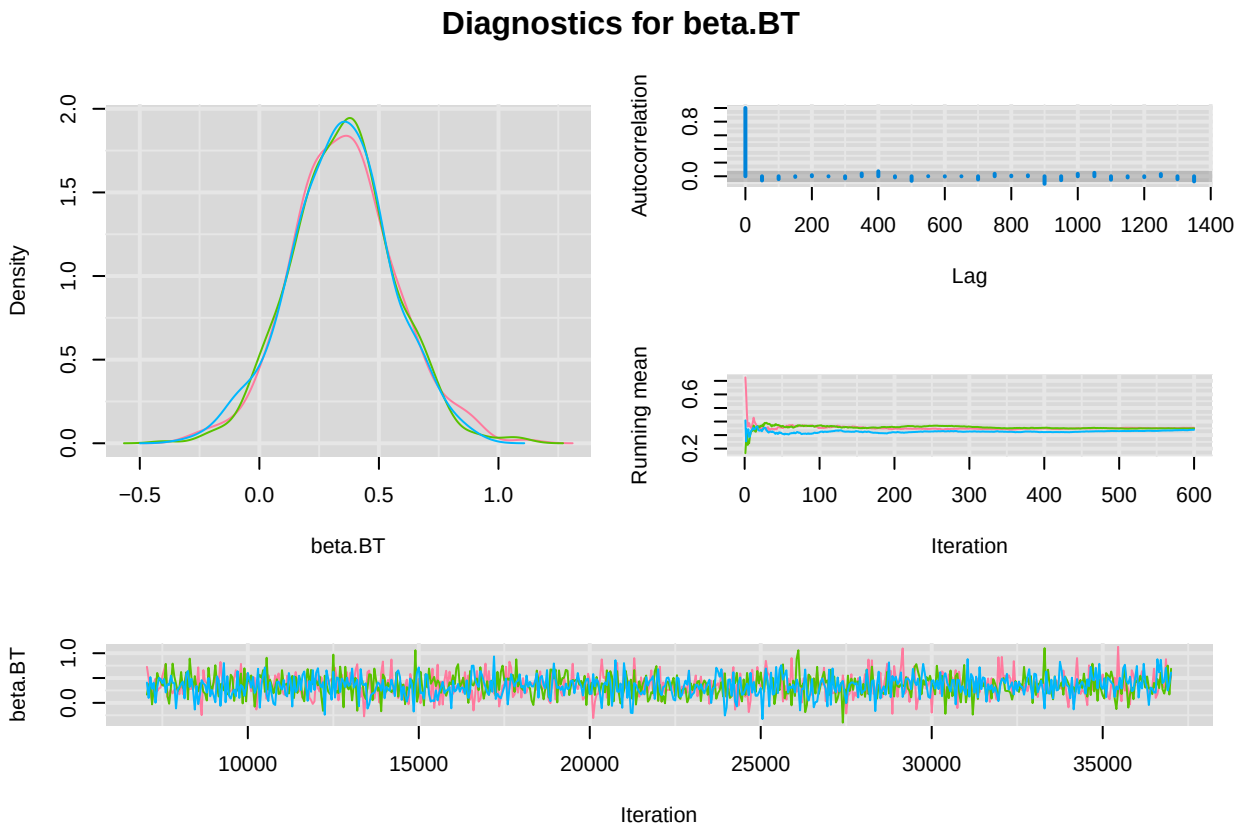

```
mcmcplot1(negBinomialRESim[,"sigma.b1",drop=FALSE])
```



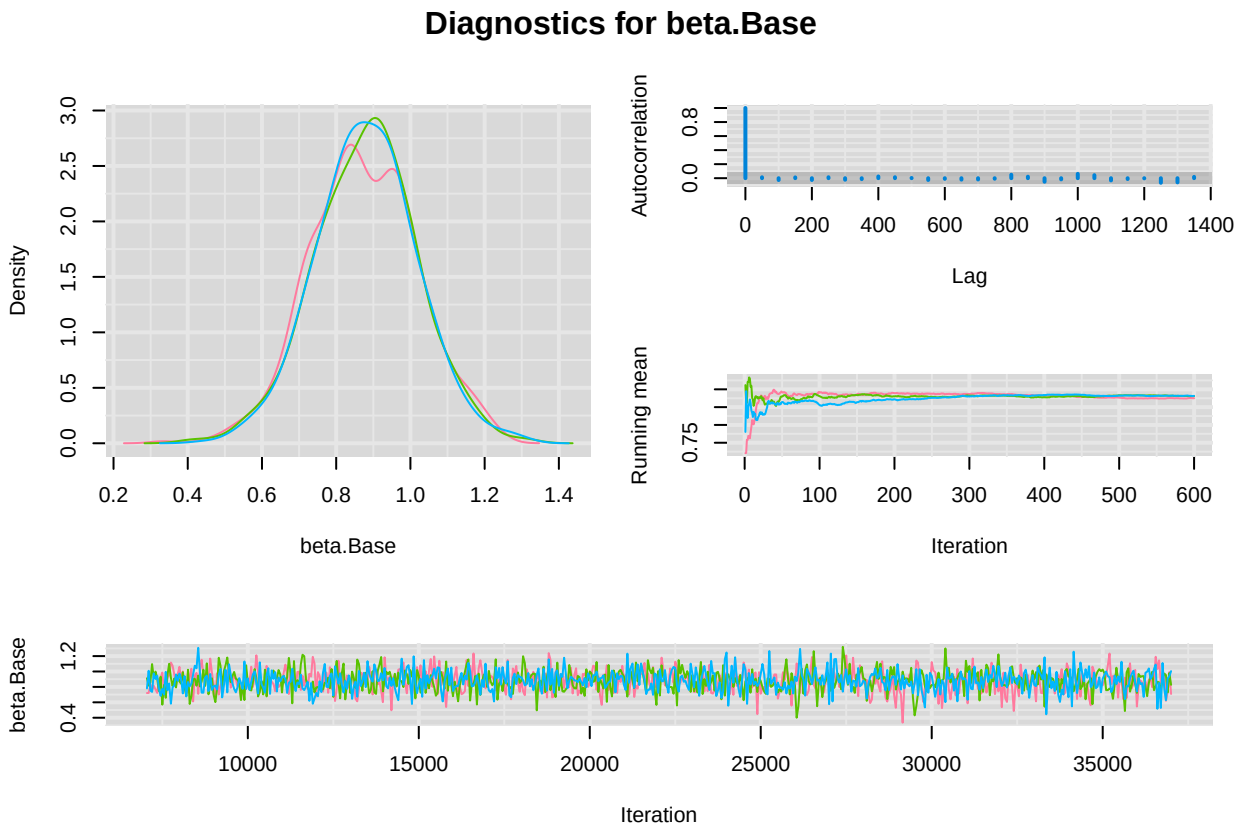

```
mcmcplot1(poissonRESim[,"beta.Age",drop=FALSE])
```



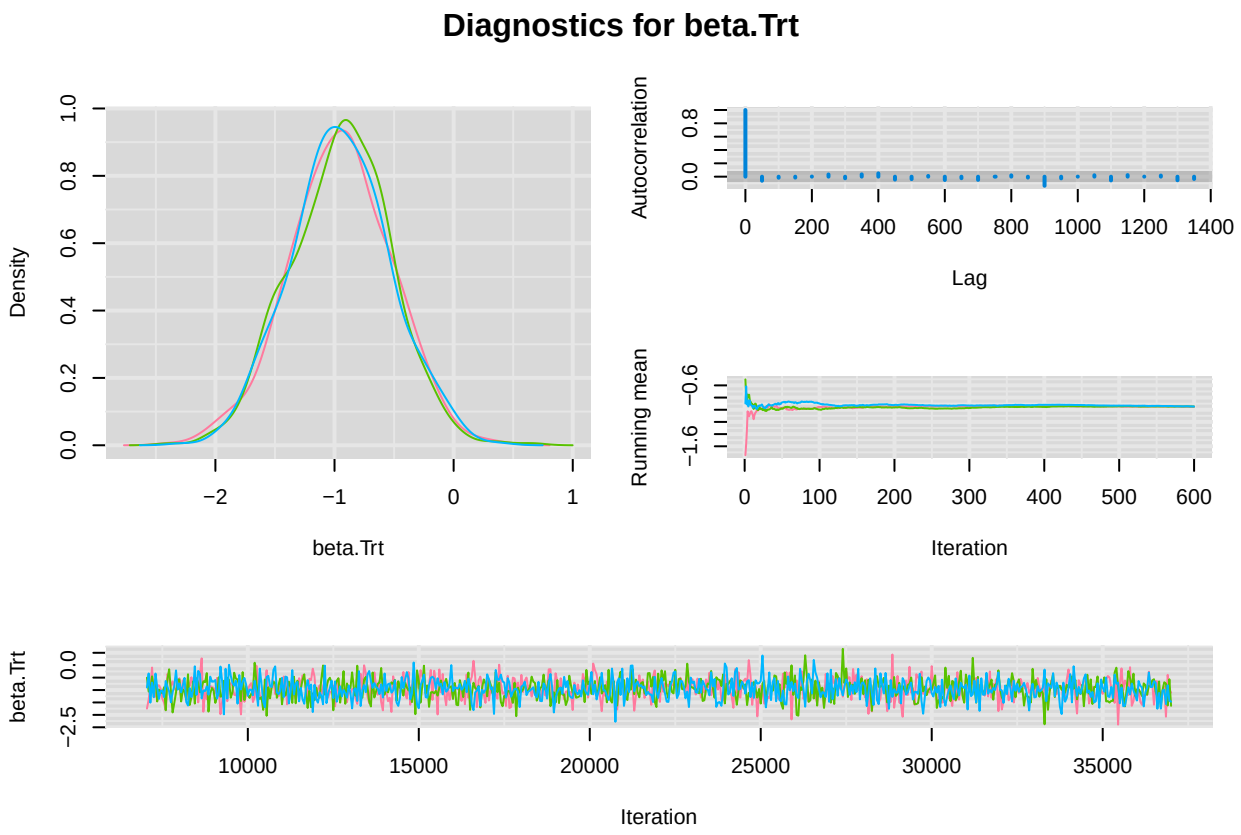

```
mcmcplot1(poissonRESim[,"beta.BT",drop=FALSE])
```



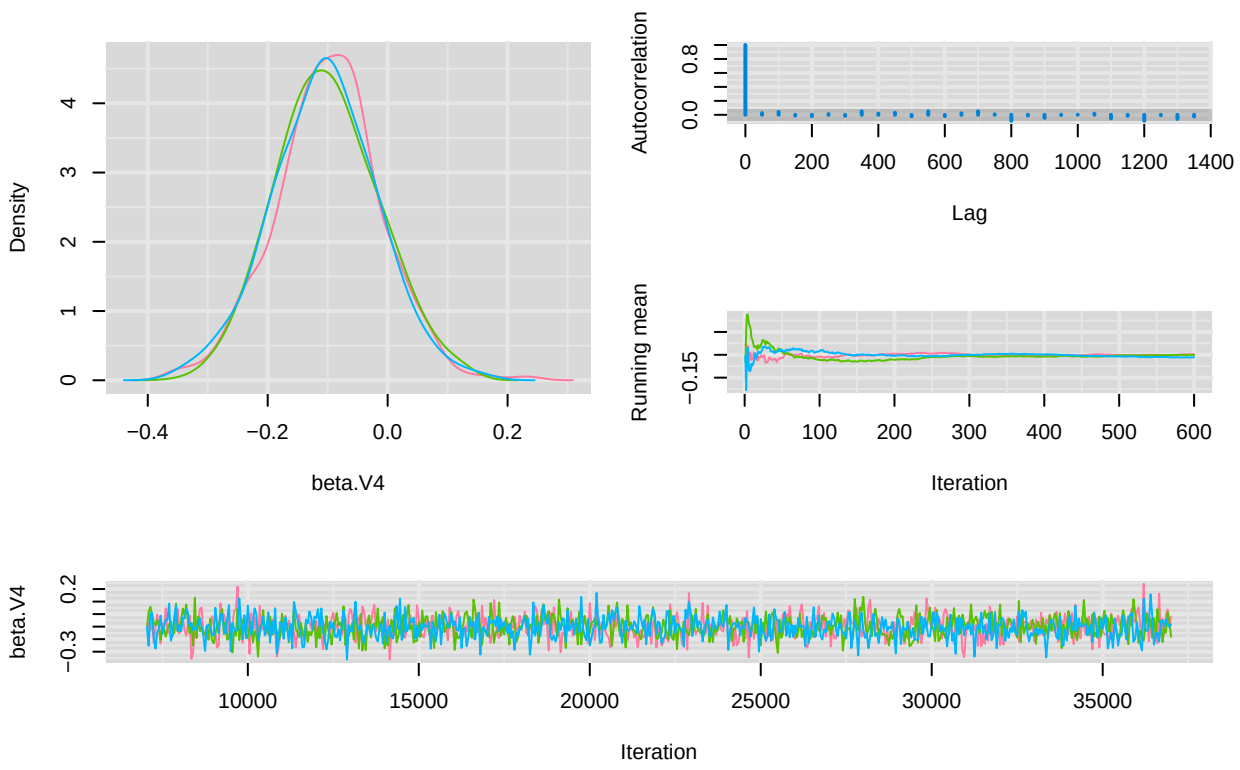

```
mcmcplot1(poissonRESim[,"beta.Base",drop=FALSE])
```



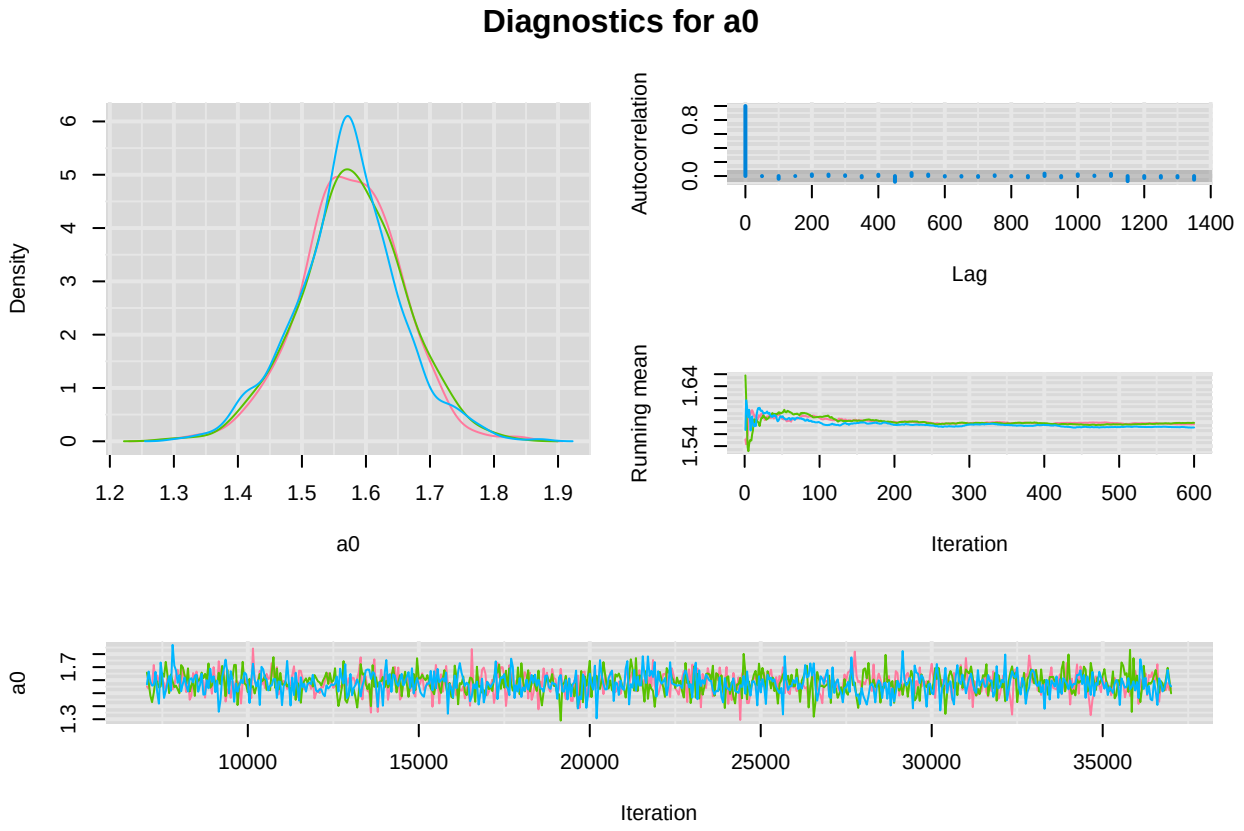

```
mcmcplot1(poissonRESim[,"beta.Trt",drop=FALSE])
```



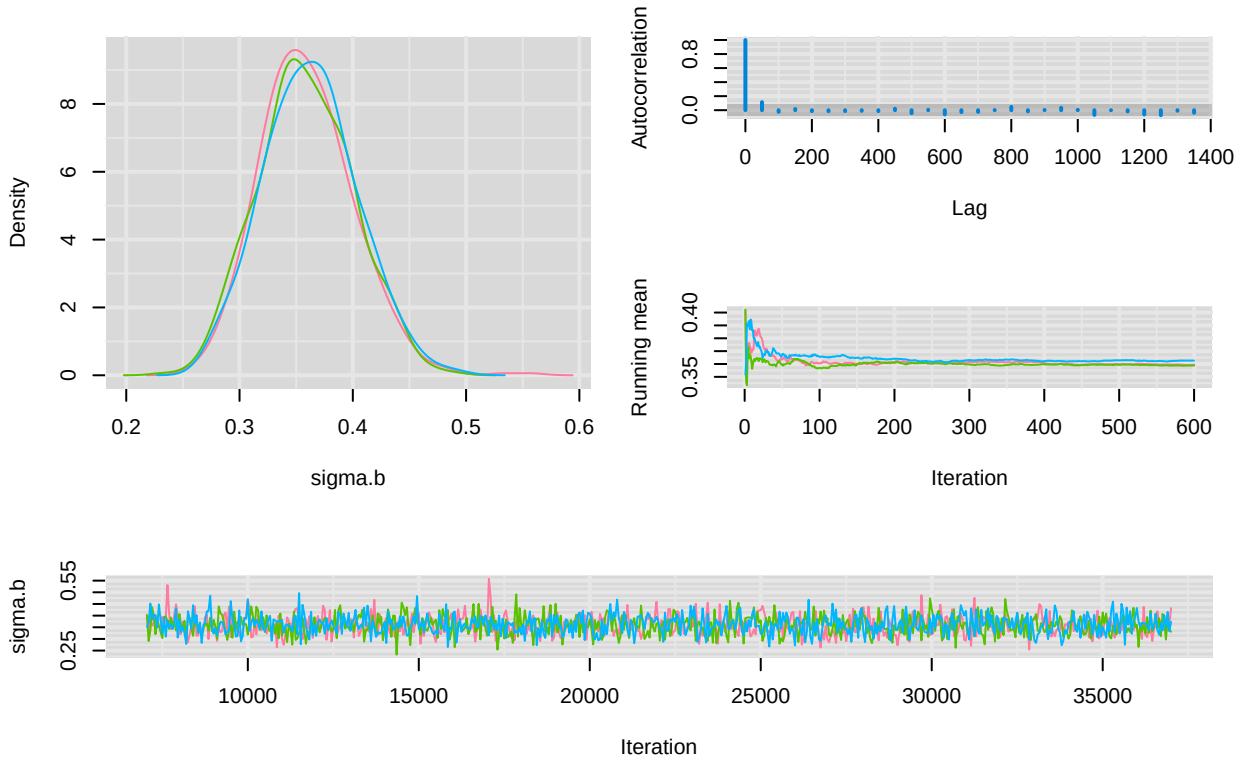

```
mcmcplot1(poissonRESim[,"beta.V4",drop=FALSE])
```



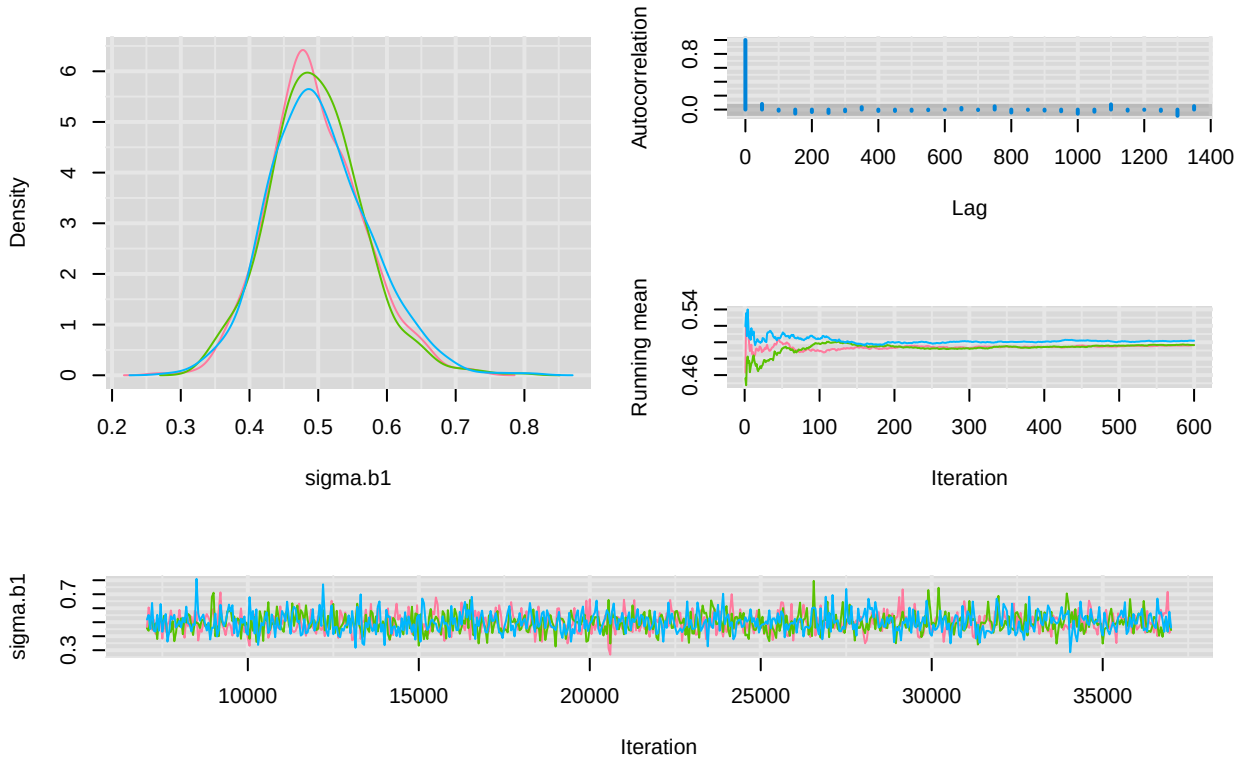

```
mcmcplot1(poissonRESim[,"a0",drop=FALSE])
```



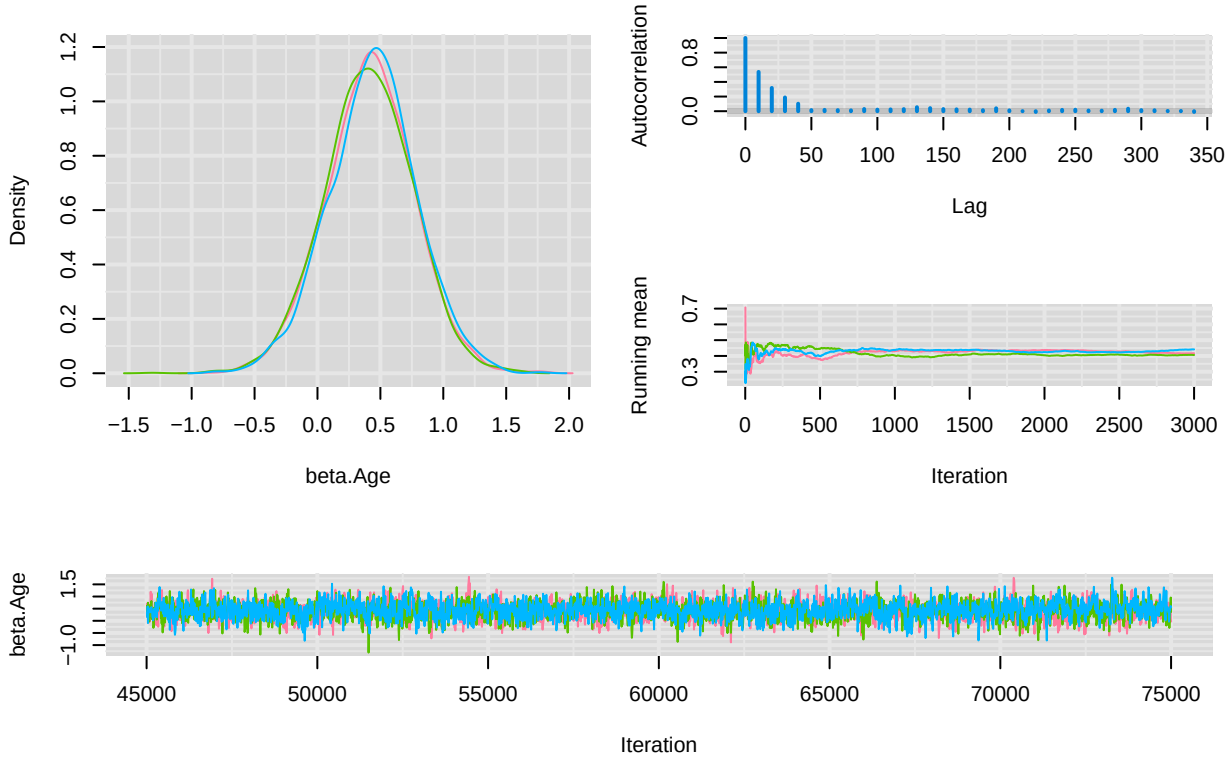

```
mcmcplot1(poissonRESim[,"sigma.b",drop=FALSE])
```



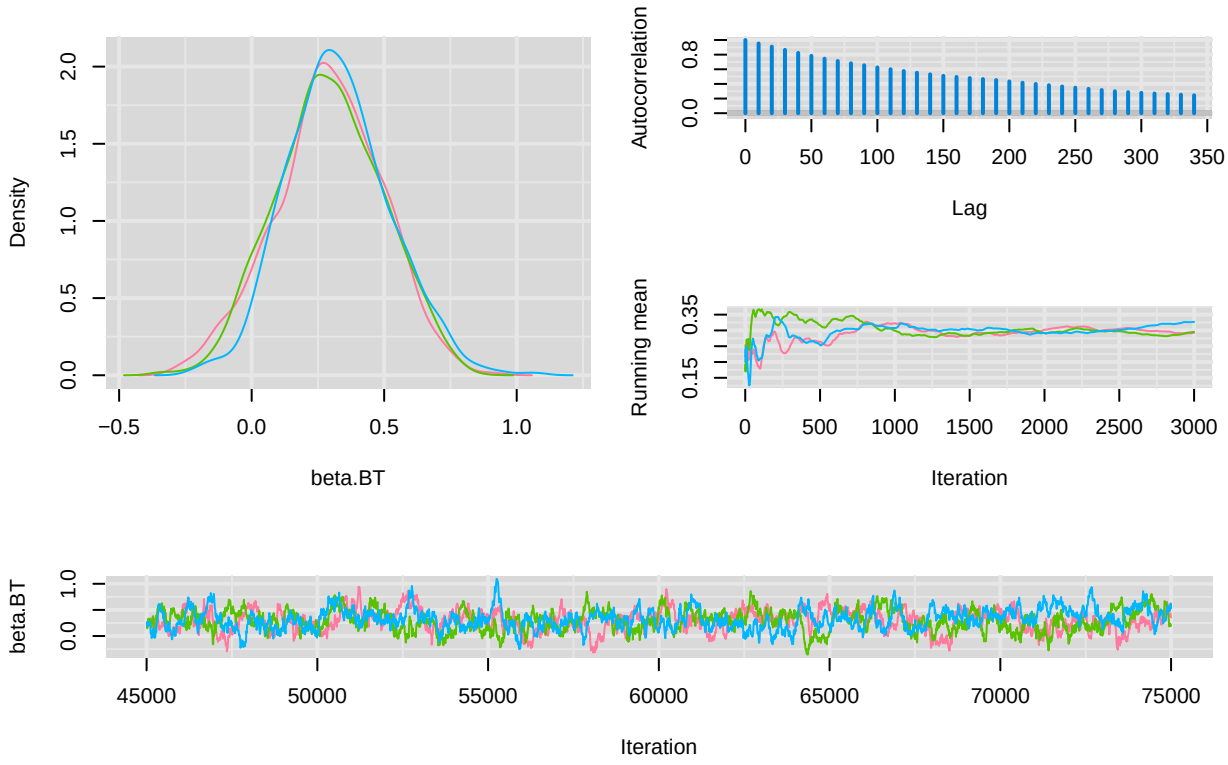

```
mcmcplot1(poissonRESim[,"sigma.b1",drop=FALSE])
```



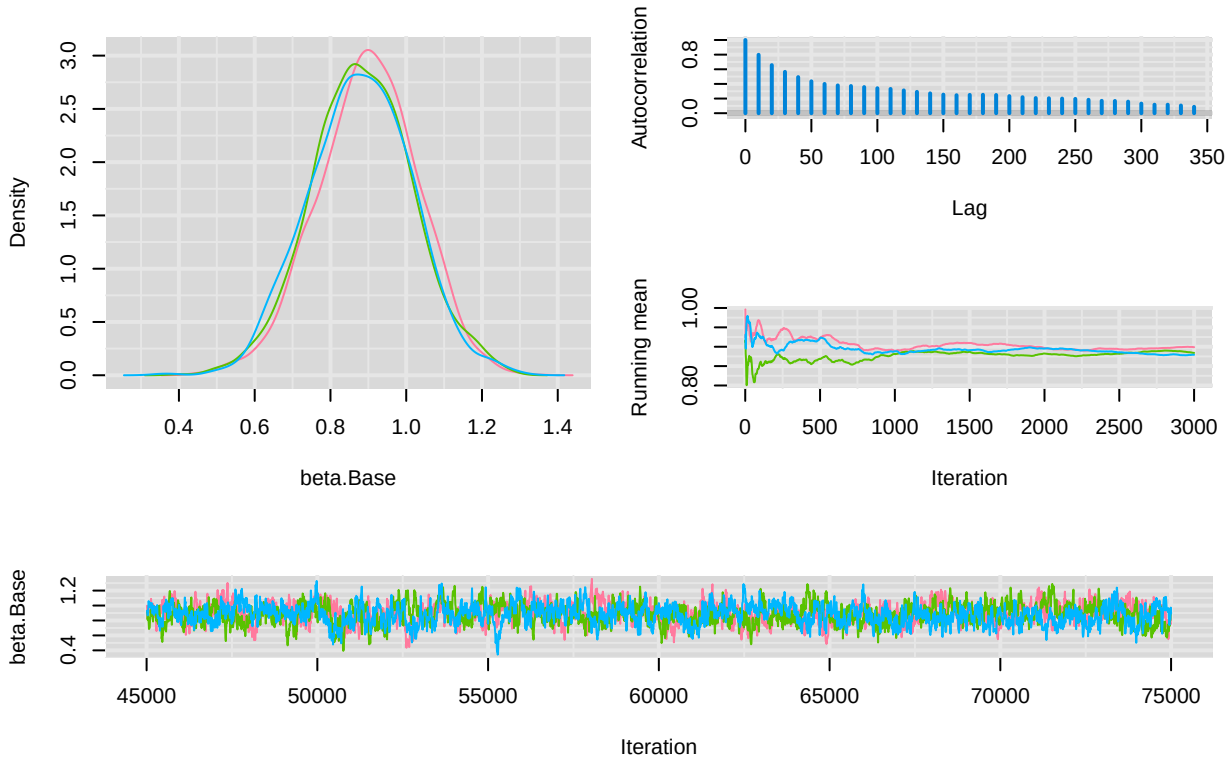

```
mcmcplot1(ZINBreSim[,"beta.Age",drop=FALSE])
```



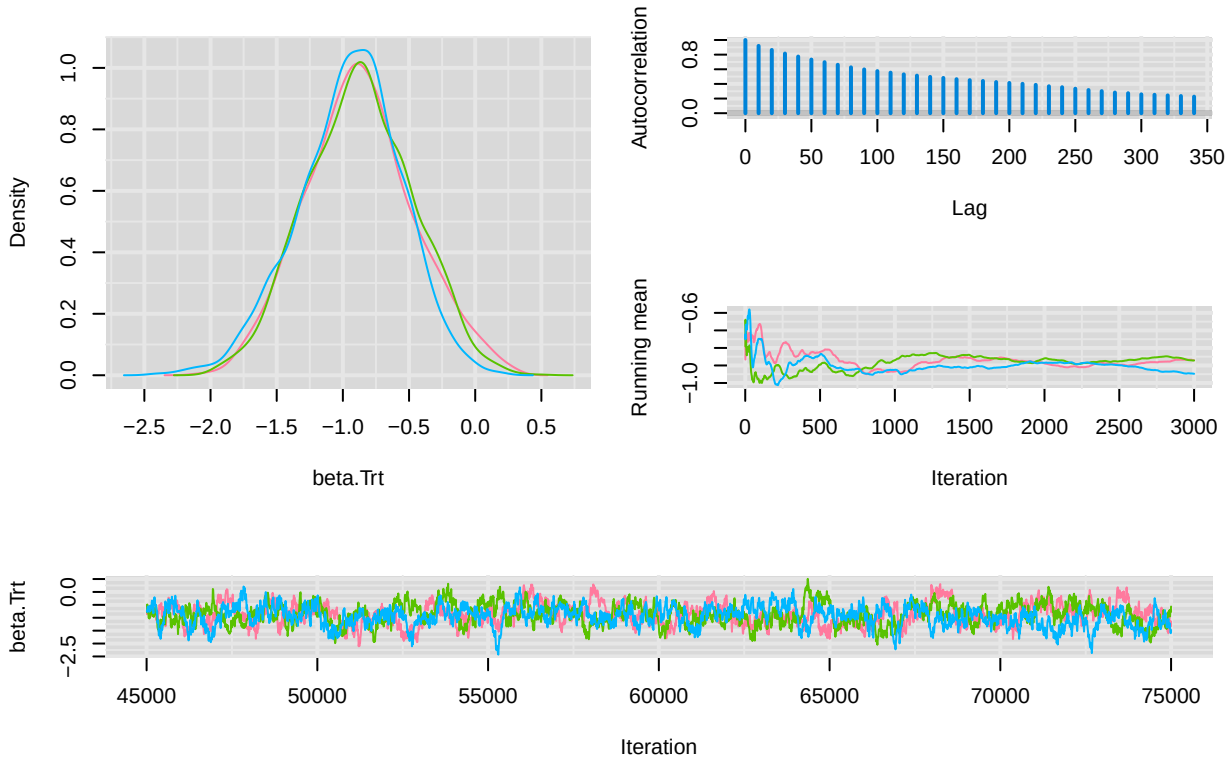

```
mcmcplot1(ZINBreSim[,"beta.BT",drop=FALSE])
```



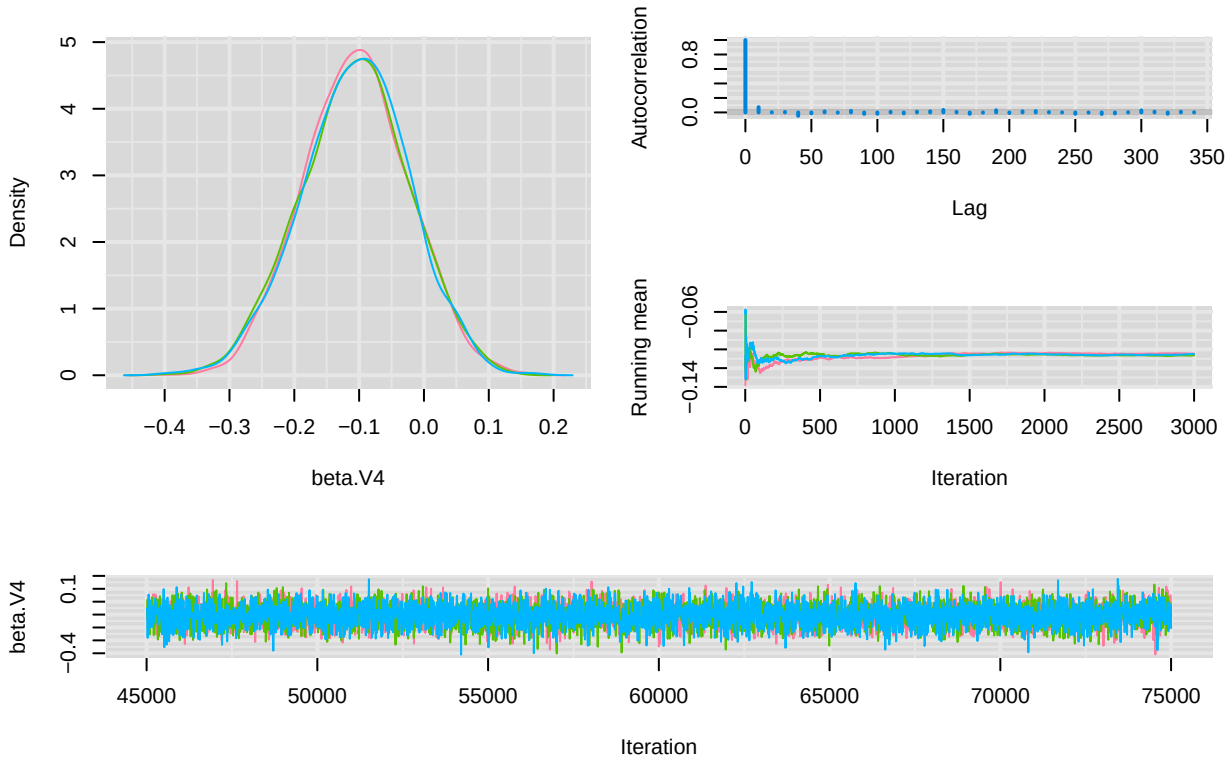

```
mcmcplot1(ZINBreSim[,"beta.Base",drop=FALSE])
```



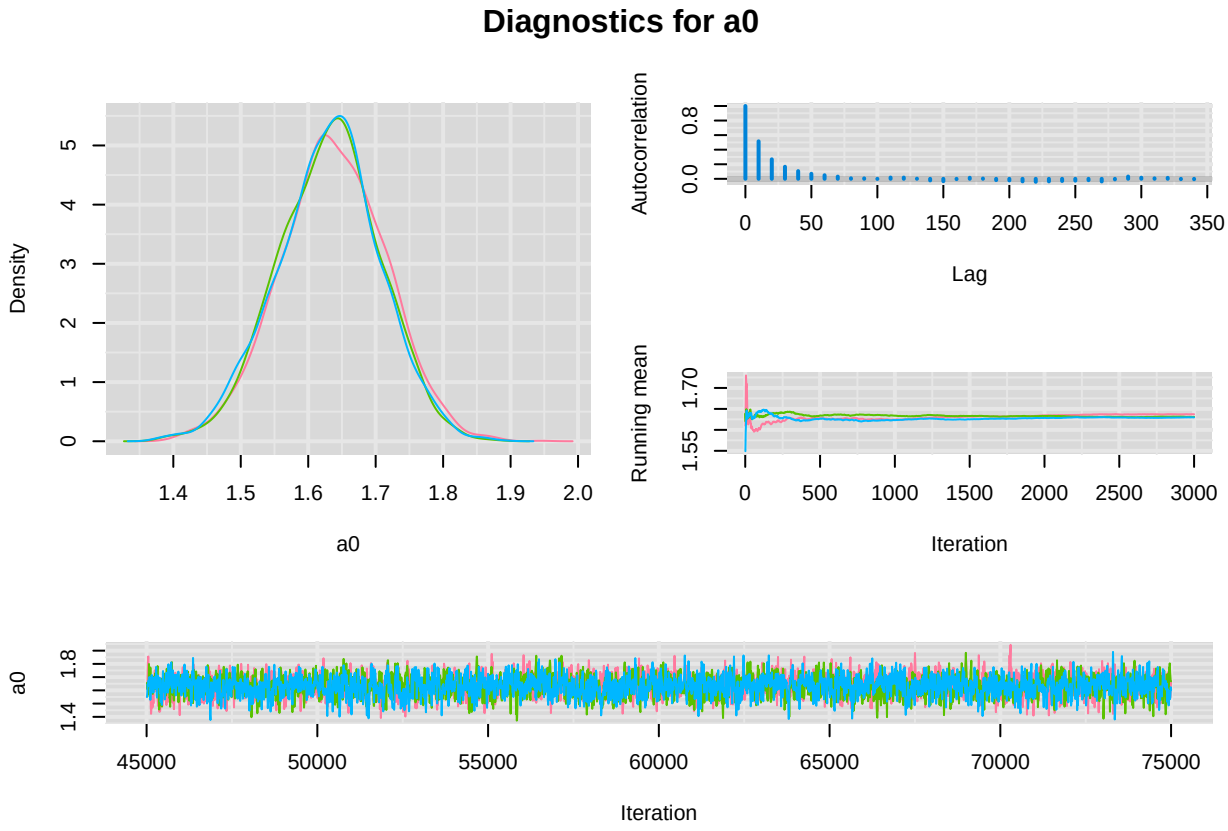

```
mcmcplot1(ZINBreSim[,"beta.Trt",drop=FALSE])
```



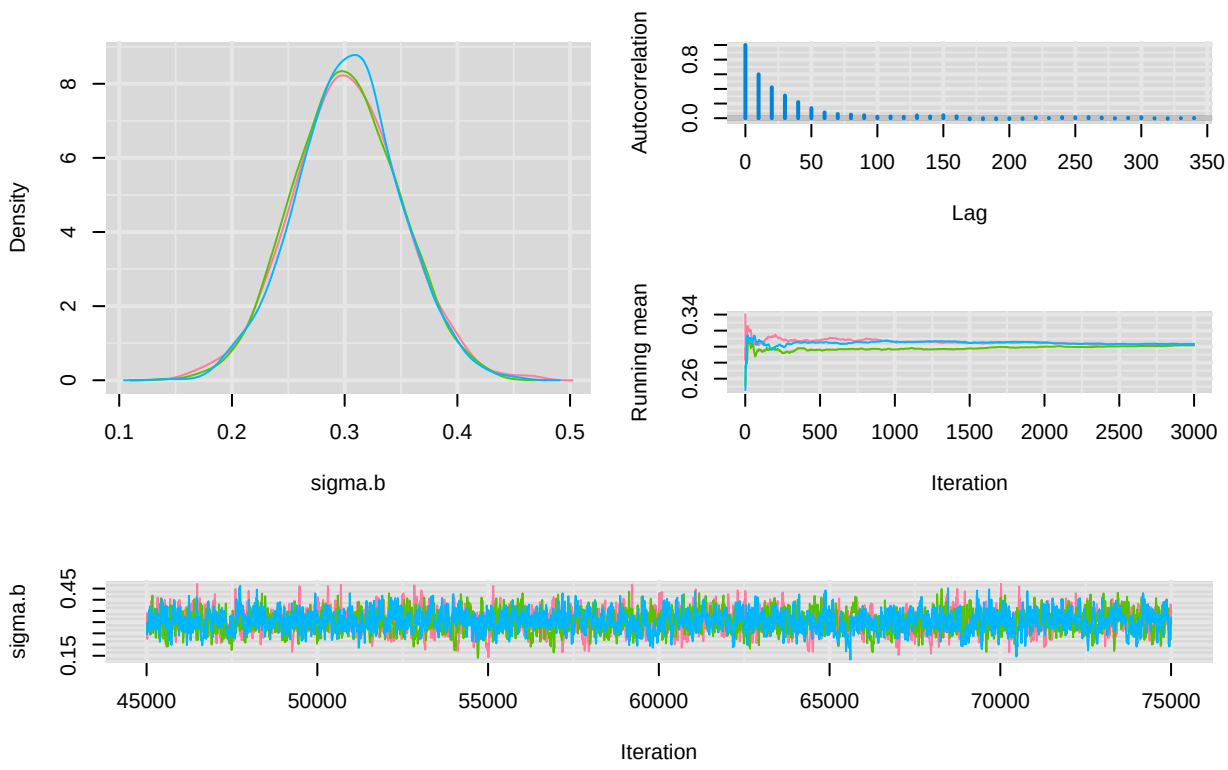

```
mcmcplot1(ZINBreSim[,"beta.V4",drop=FALSE])
```



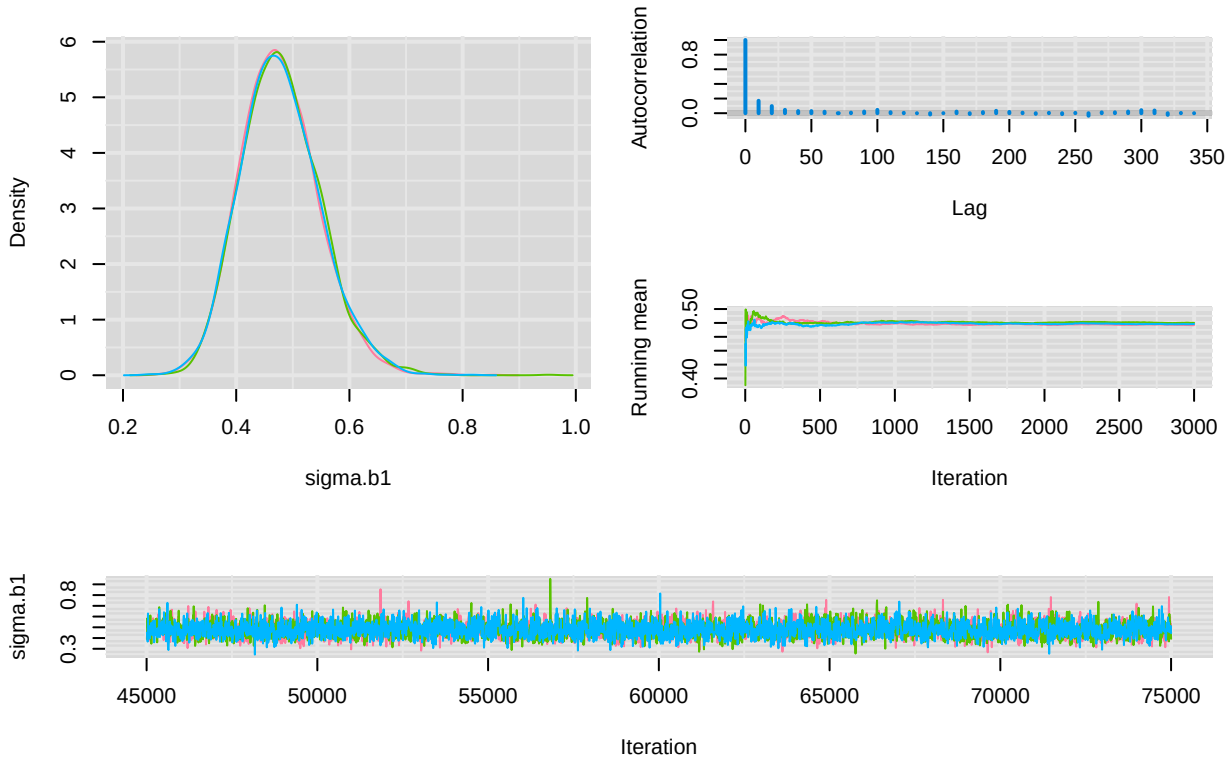

```
mcmcplot1(ZINBreSim[,"a0",drop=FALSE])
```



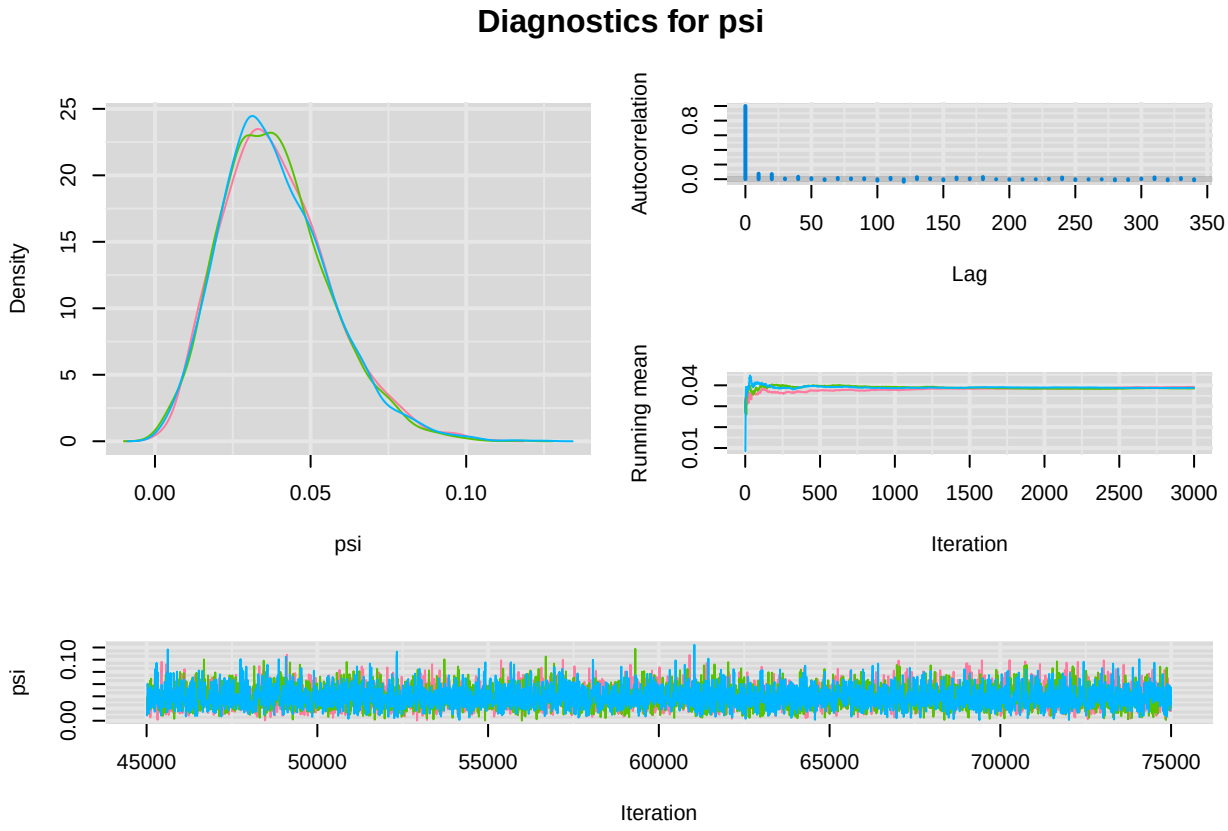

```
mcmcplot1(ZINBreSim[,"sigma.b",drop=FALSE])
```



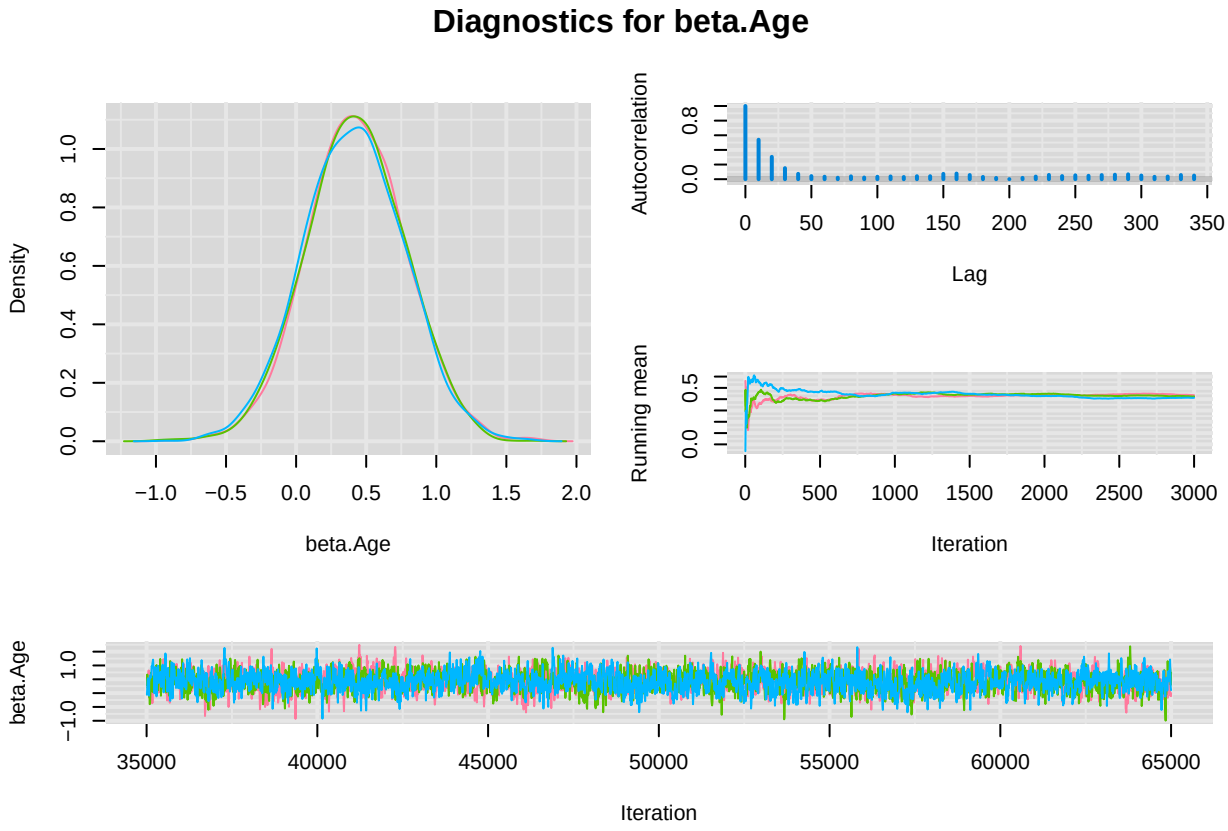

```
mcmcplot1(ZINBreSim[,"sigma.b1",drop=FALSE])
```



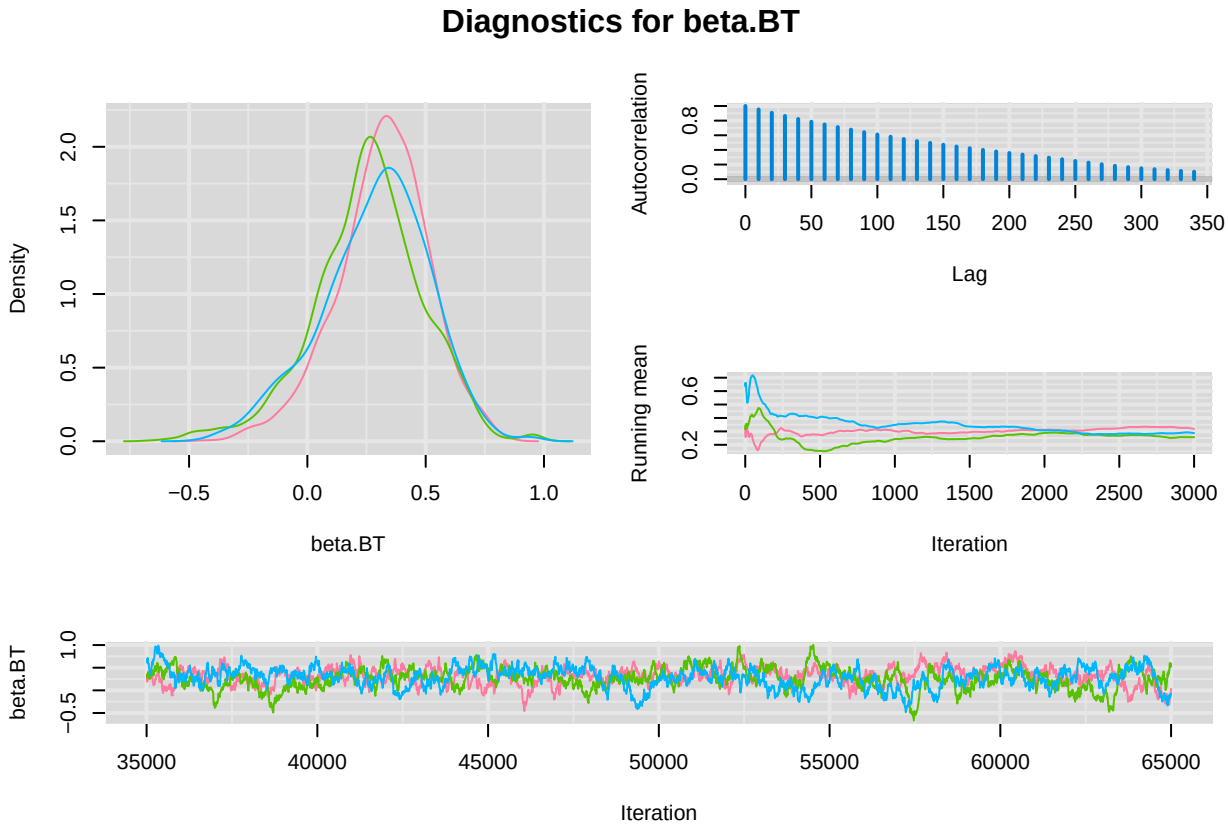

```
mcmcplot1(ZINBreSim[,"psi",drop=FALSE])
```



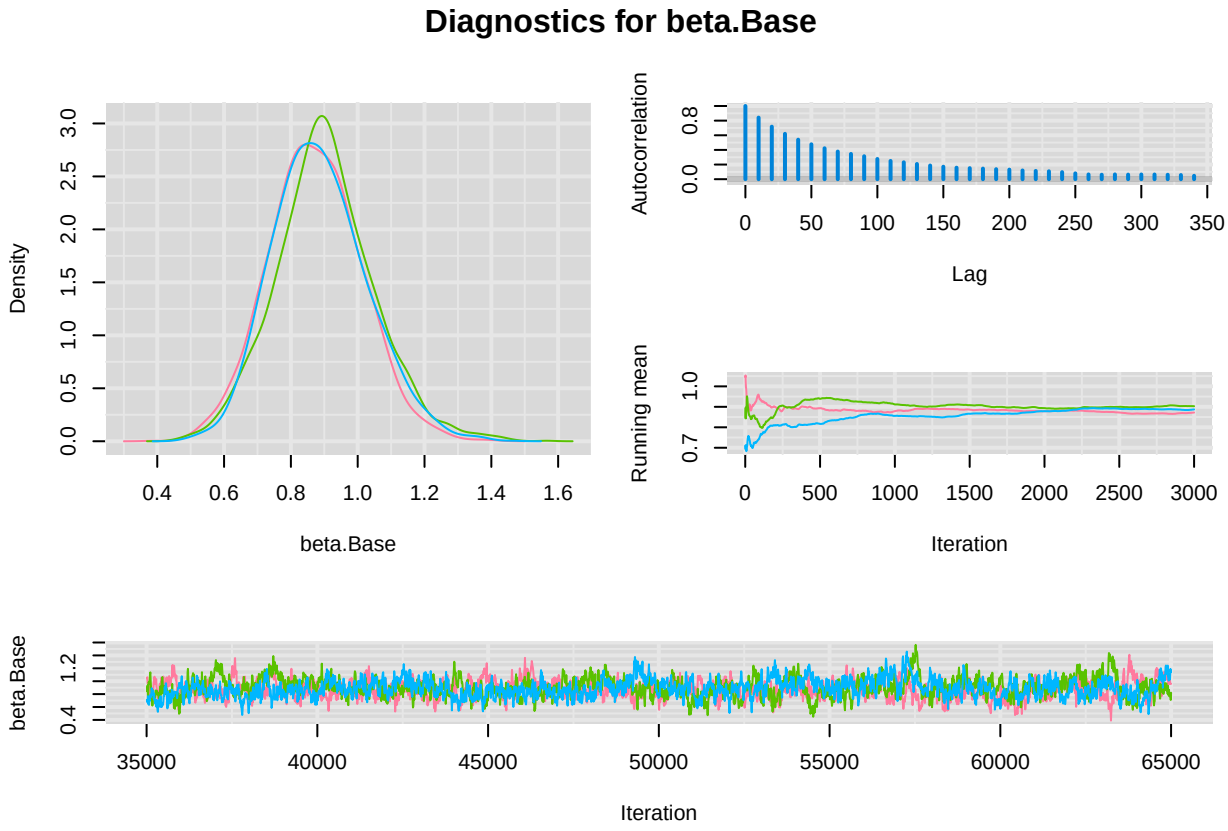

```
mcmcplot1(ZIPreSim[,"beta.Age",drop=FALSE])
```



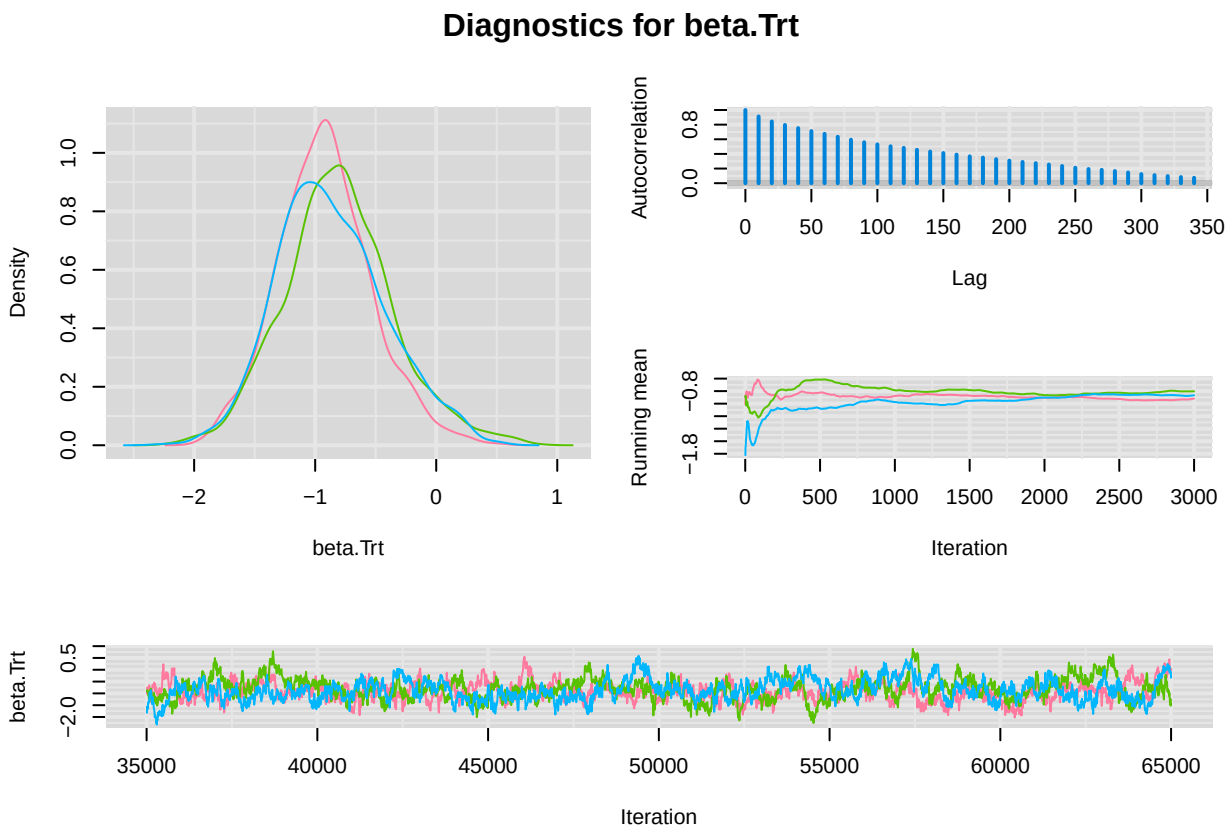

```
mcmcplot1(ZIPreSim[,"beta.BT",drop=FALSE])
```



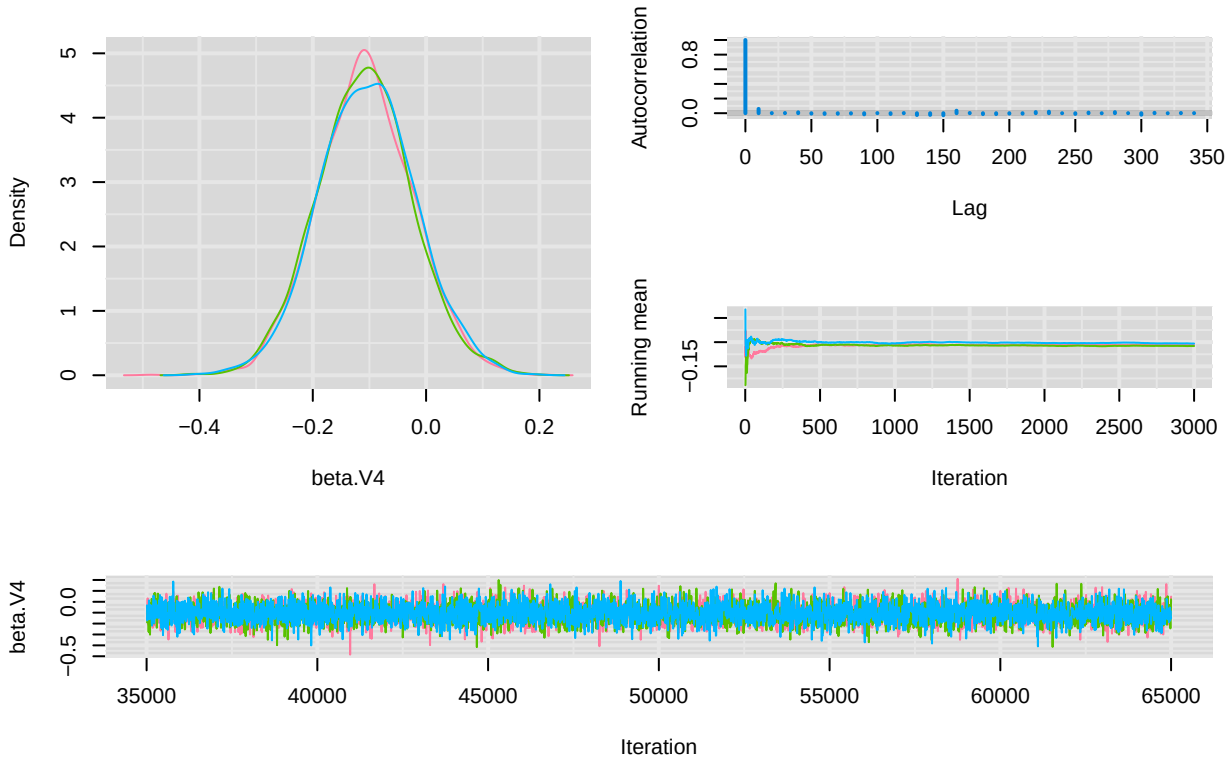

```
mcmcplot1(ZIPreSim[,"beta.Base",drop=FALSE])
```



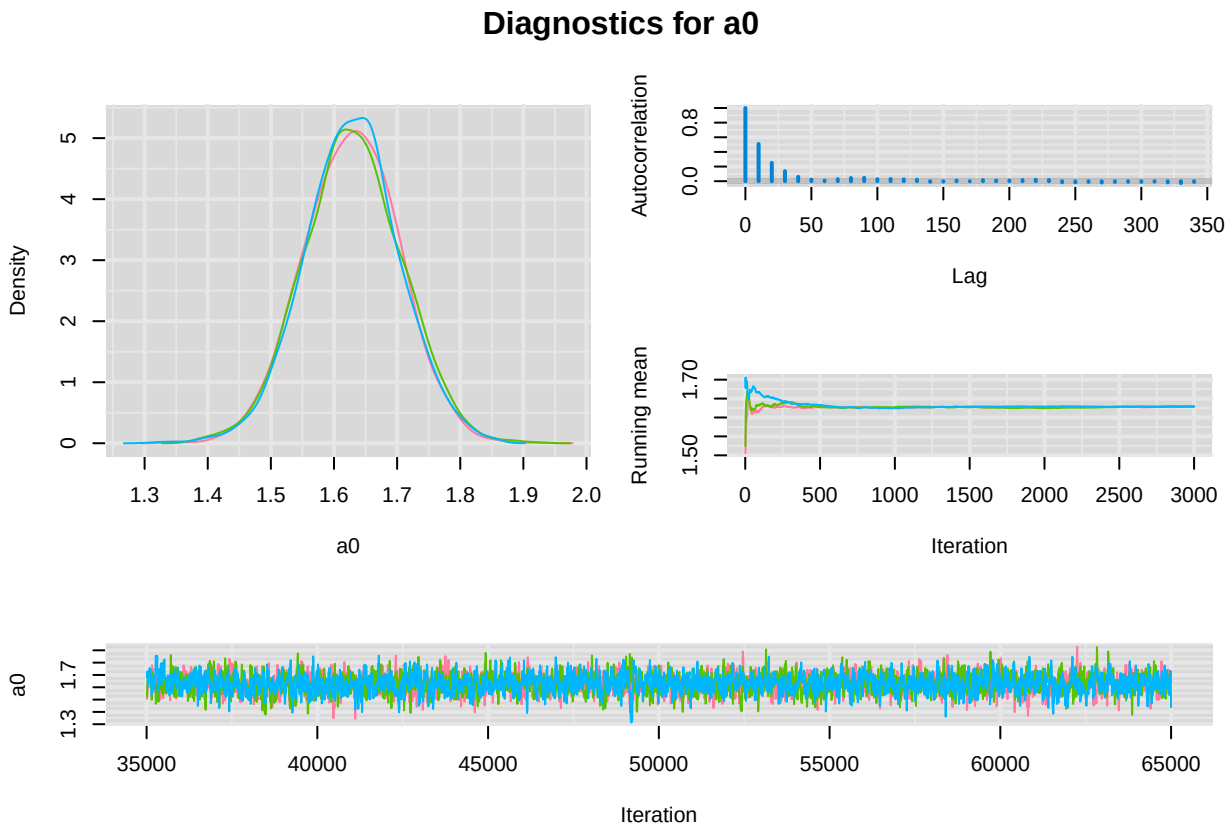

```
mcmcplot1(ZIPreSim[,"beta.Trt",drop=FALSE])
```



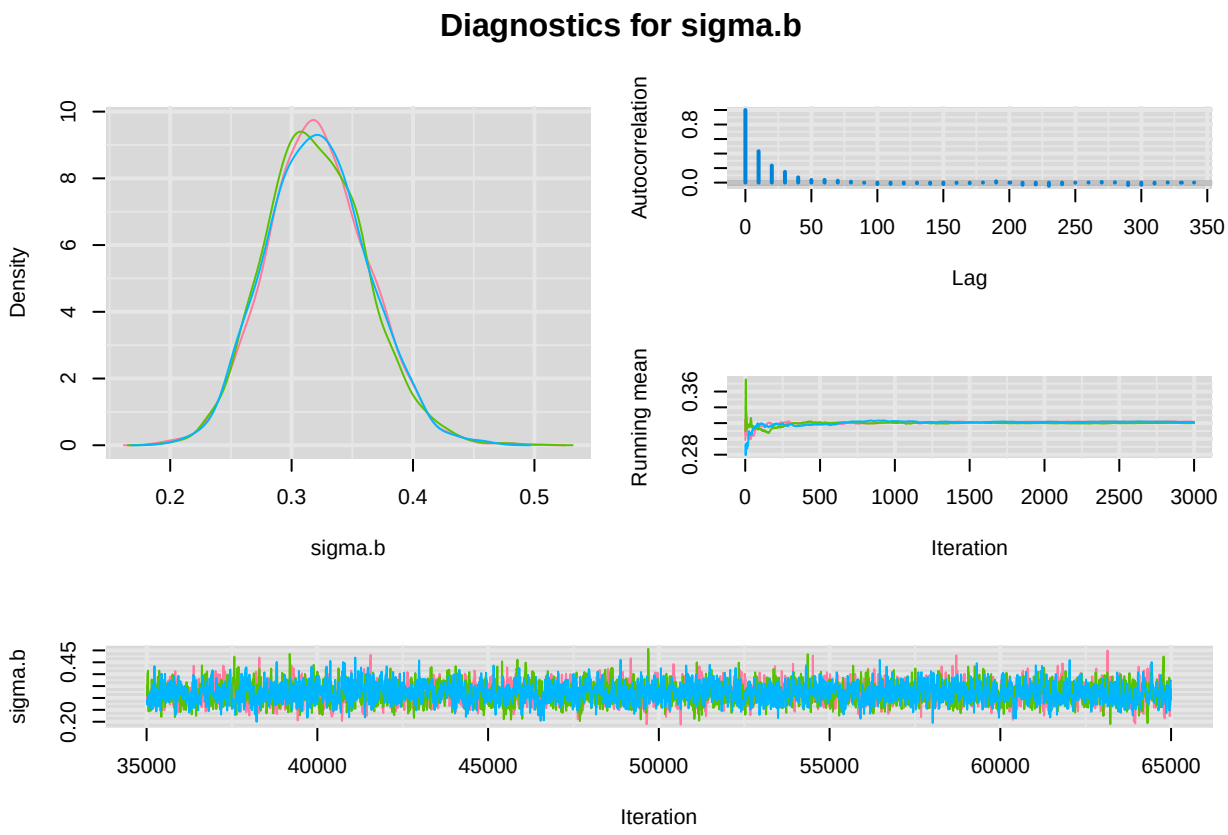

```
mcmcplot1(ZIPreSim[,"beta.V4",drop=FALSE])
```



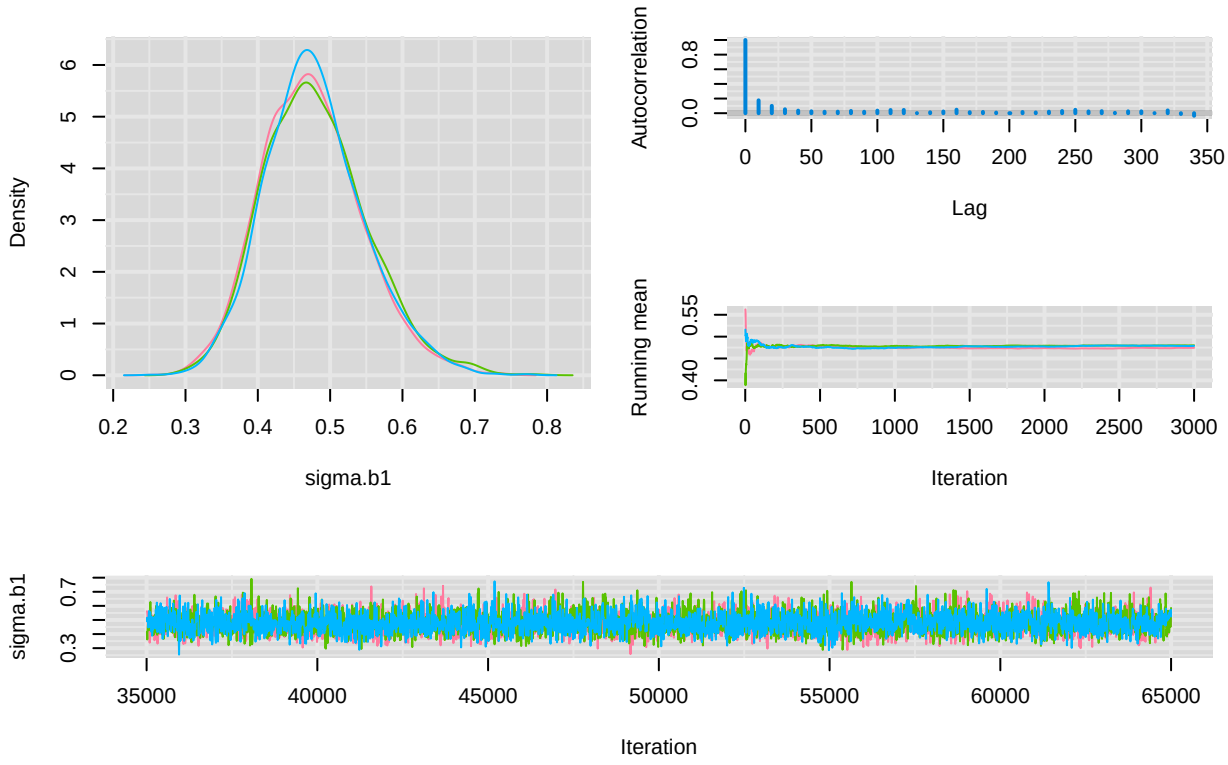

```
mcmcplot1(ZIPreSim[,"a0",drop=FALSE])
```



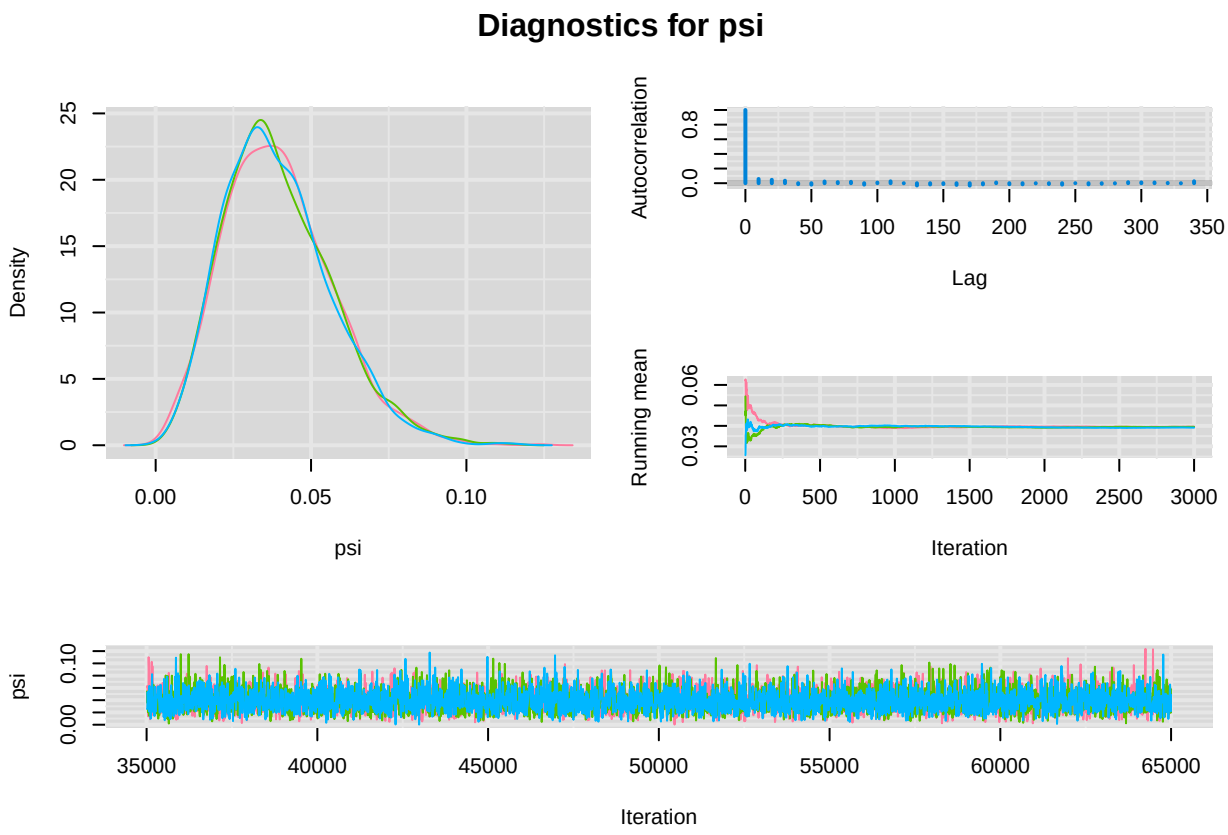

```
mcmcplot1(ZIPreSim[,"sigma.b",drop=FALSE])
```




```
mcmcplot1(ZIPreSim[,"sigma.b1",drop=FALSE])
```




```
mcmcplot1(ZIPreSim[,"psi",drop=FALSE])
```



4. Model Comparison

3.3 Bur

the MCMC sampling chain was adequately mixed and the retained samples independent. The chains appear well mixed and stable the samples are now less auto-correlated and the chains are also arguably mixed a little better.

A combination of normal and independent Gamma random effects. This is the most commonly used form of the combined model in which the normal random effects induce/account for correlation while the Gamma random effects induce/account for overdispersion. It is model (3.16) but with Σi diagonal.