Ratings

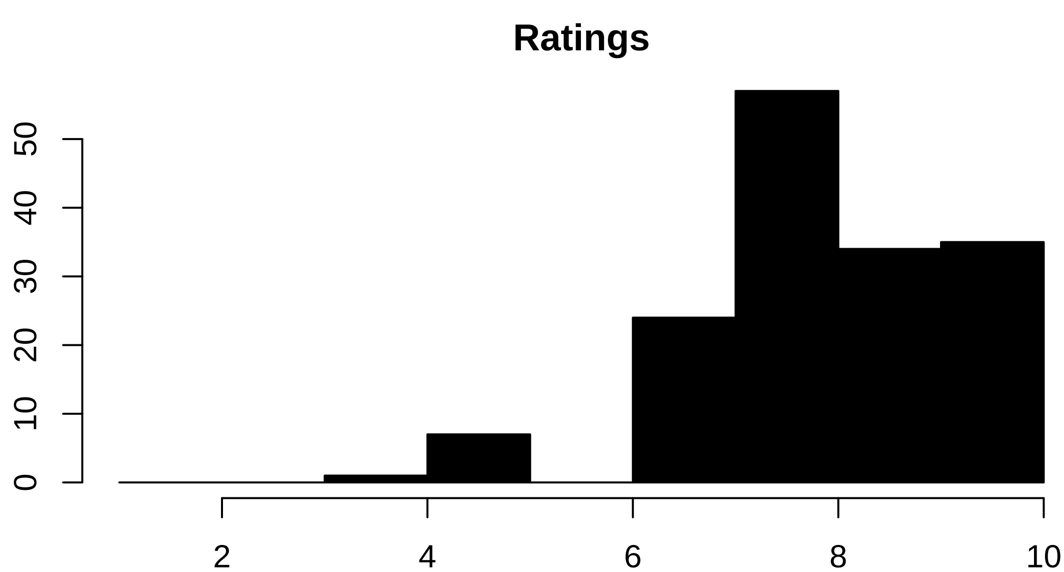
Cecilia Martinez Oliva

1. Dataset

I have analysed 185973 books rated by 77805 users, with 433671 rating. In order to avoid too many missing values, I shrinked the sample in a cluster where each book was rated by at least 50% of the users.

```
# data
setwd('D:/DataScience/SDS2/Books')
set.seed(1234)
r = as.matrix(read.csv("Books.csv")[2:12])
NUsers = dim(r)[1]
NBooks = dim(r)[2]
cat(paste('N users:', NUsers, '\n'))
cat(paste('N books:', NBooks, '\n'))
cat(paste('Missing values:',sum(is.na(r))/length(c(r))),'\n')
c('Mean' = mean(r, na.rm = T), 'SD' = sd(r, na.rm = T))
par(mar = c(2,2,2,2))
hist(r, col = 'black', breaks = 1:10, main = 'Ratings')
```

```
## N users: 20
## N books: 11
## Missing values: 0.281818181818182
##      Mean        SD
## 8.348101 1.281556
```



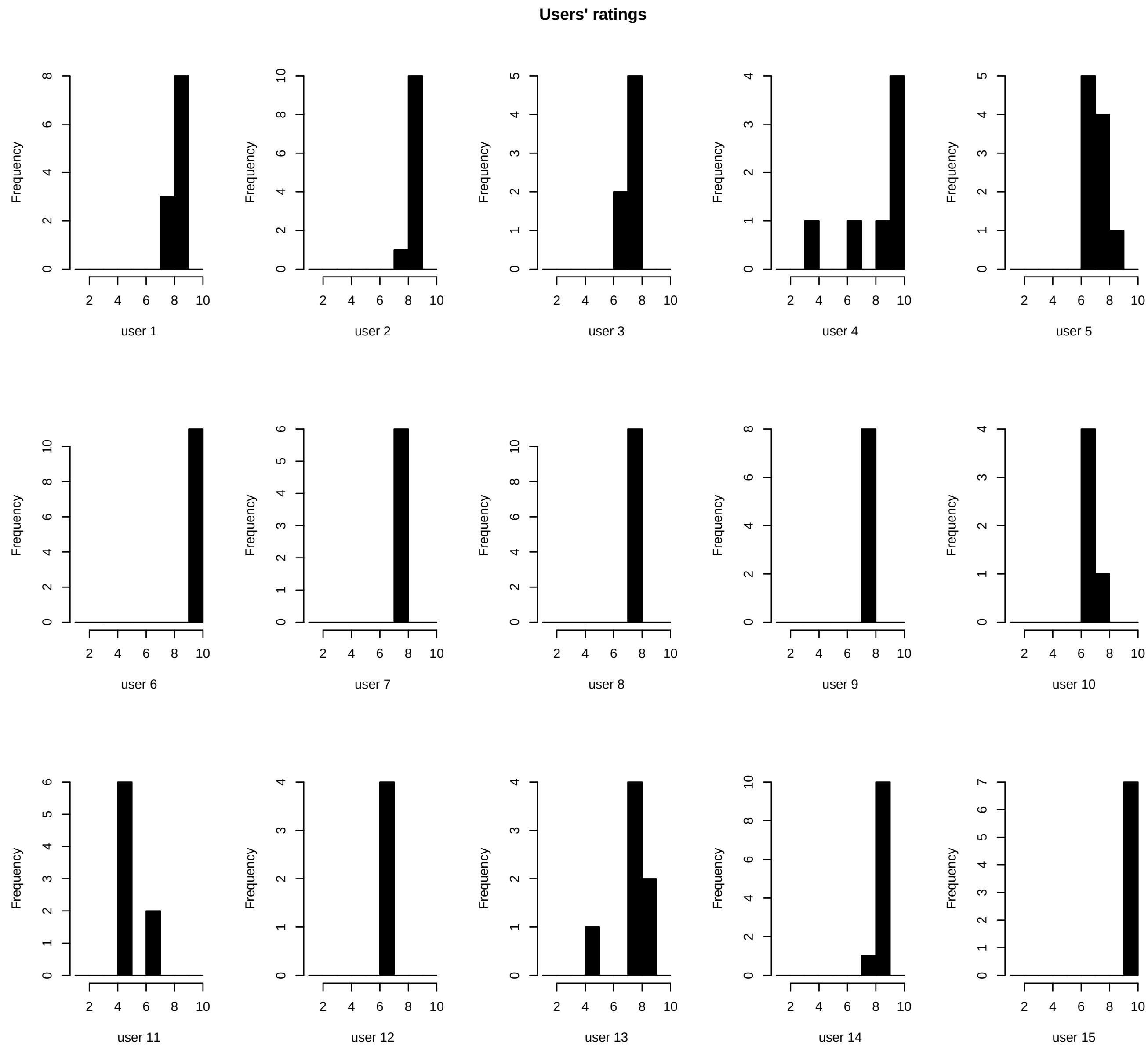
I obtained a dataset with 20 users, 11 books, and 28% of missing values. The average rate is 8.34 with standard deviation of 1.28.

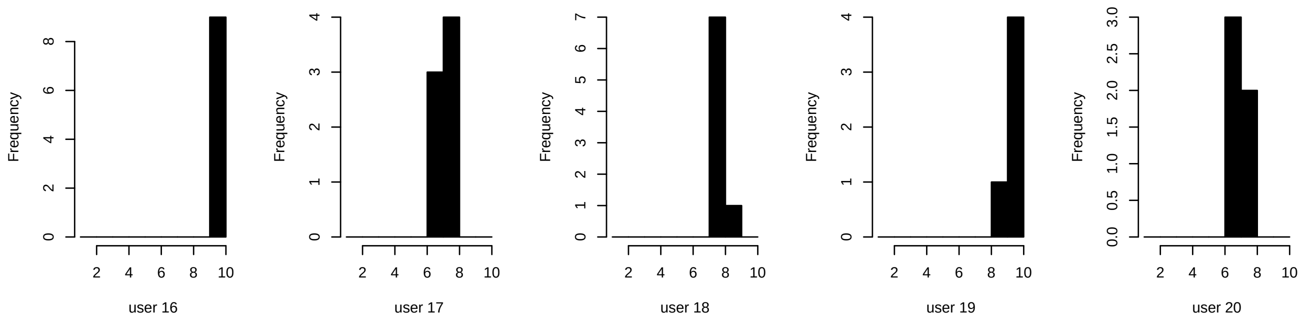
1.1 Users

```
par(mfrow=c(1,5))
v = rep(NA, dim(r)[1])
hist(r[1,], col = 'black', breaks = 1:10, xlab = paste('user',1), main='')
```

```
v[1] = mean(r[1,], na.rm = T)
title("\n Users' ratings", outer=TRUE)
for(i in 2:dim(r)[1]) {
  hist(r[i,], col = 'black', breaks = 1:10,
       xlab = paste('user',i), main='')
  v[i] = mean(r[i,], na.rm = T)
}
cat(paste('SD:', sd(v), '\n'))
```

```
## SD: 1.15317989899602
```



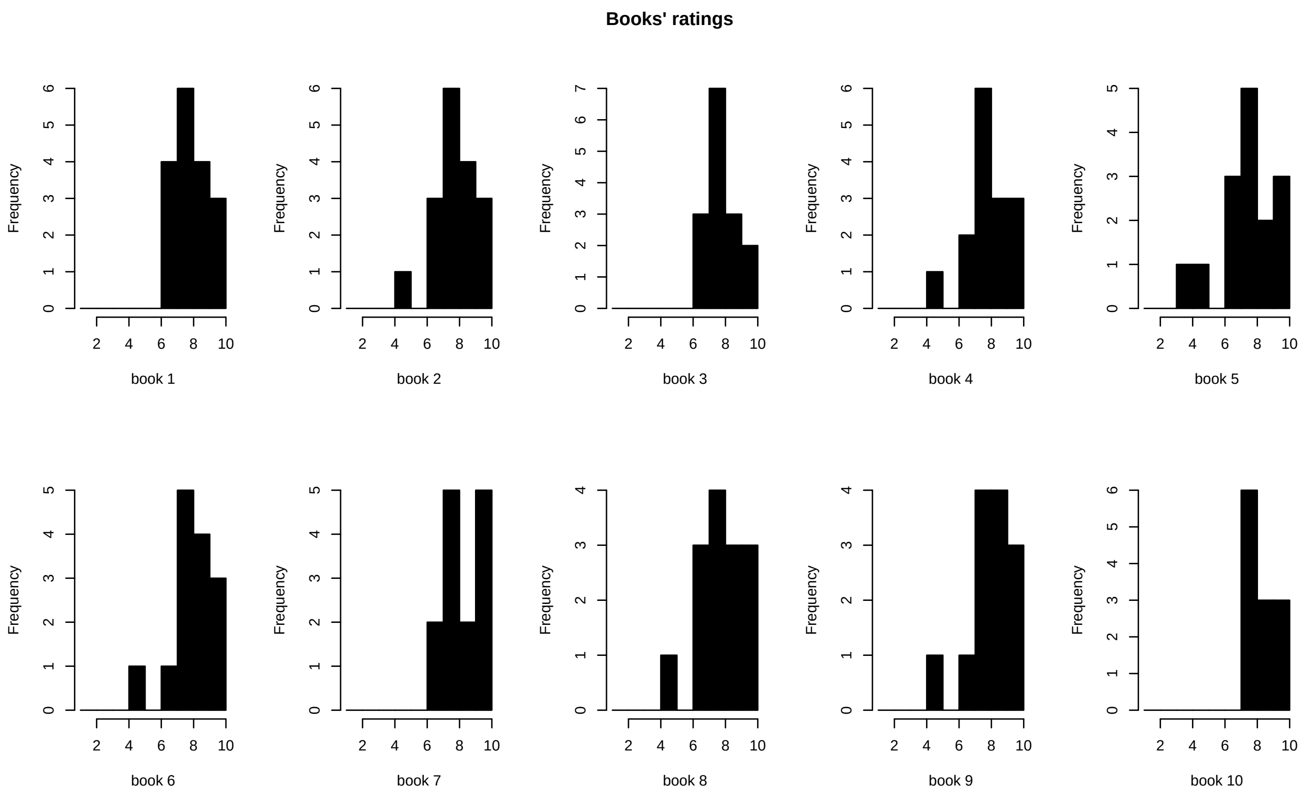


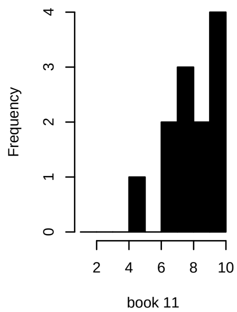
About 50% of the users have given the same rating to all the books they read. The standard deviation between the average ratings of the users is 1.15.

1.2 Books

```r
par(mfrow=c(1,5))
v = rep(NA, dim(r)[2])
hist(r[,1], col = 'black', breaks = 1:10, xlab = paste('book',1), main='')
v[1] = mean(r[,1], na.rm = T)
title("\n Books' ratings", outer=TRUE)
for(i in 2:(dim(r)[2])) {
  hist(r[,i], col = 'black', breaks = 1:10,
       xlab = paste('book',i), main='')
  v[i] = mean(r[,i], na.rm = T)
}
cat(paste('SD:', sd(v), '\n'))
```

```
## SD: 0.243610107008042
```



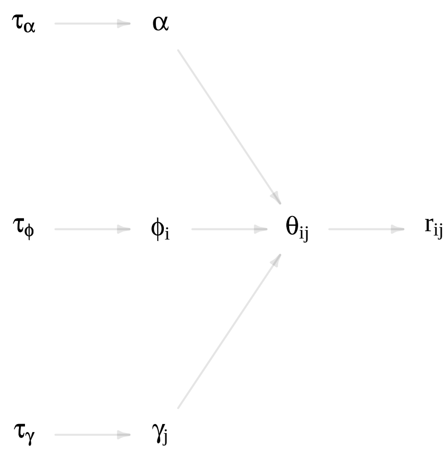



```
ratings = r-1 # da 0 a 9
user.m = apply(ratings,1,mean, na.rm=T)
book.m = apply(ratings,2,mean, na.rm=T)
ratings = structure(.Data = c(ratings), .Dim = c(NUsers,NBooks))
```

Ratings given to each book are more heterogeneous and the standard deviation between the average ratings of the books is 0.24, lower than the users' one (1.15).

2. Model: random effects on users and books

We can suppose that each rating given by a user to a book depends both on the user and the book. ϕ_i is the influence of user i on rating$_{ij}$, while γ_j is the influence of book j. α is a constant.
We suppose that ϕs, γs and α are generated by normal distributions with 0 mean and τ_u, τ_b and τ_α precisions, respectively.
τ_u, τ_b, and τ_α have gamma's prior distributions.
The original ratings had values from 1 to 10 so we want to predict ratings-1 distributed as a binomial distribution with n=9 and p=θ_{ij}. I am going to call them ratings$_{ij}$ instead of ratings$_{ij}$-1 for semplicity. Logit of θij is equal to α+ϕ_i+γ_j to allow θ_{ij} to take values in (0,1).



$$rating_{ij} \sim Binom(9, \theta_{ij}) \qquad\qquad where \quad logit(\theta_{ij}) = \alpha + \phi_i + \gamma_j$$

$$\alpha \sim N(0, \tau_\alpha), \qquad \tau_\alpha \sim Gamma(a, b)$$

$$\phi_i \sim N(0, \tau_u), \qquad \tau_u \sim Gamma(a, b) \qquad\qquad for \;\; i = 1, ..Nusers$$

$$\gamma_j \sim N(0, \tau_b), \qquad \tau_b \sim Gamma(a, b) \qquad\qquad for \;\; j = 1, ..Nbooks$$

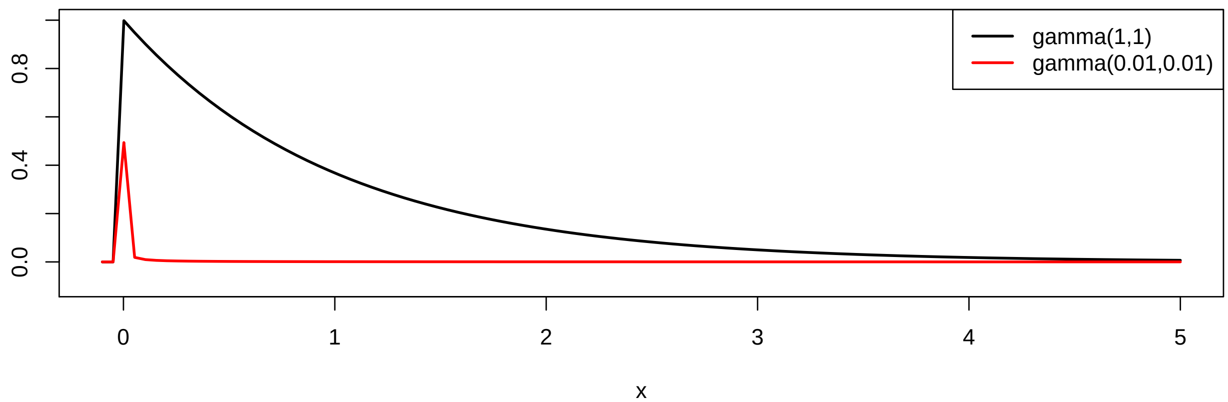
MODEL

```
model {
     for (i in 1:Nusers) {
          for (j in 1:Nbooks) {
               ratings[i,j]~dbin(theta[i,j],9)
               logit.theta[i,j] <- alpha + phi[i] + gamma[j]
               theta[i,j] <- exp(logit.theta[i,j])/(1+exp(logit.theta[i,j]))
          }
          phi[i]~dnorm(0, tau.u)
     }
     for(j in 1:Nbooks) {
          gamma[j]~dnorm(0, tau.b)
     }
     alpha~dnorm(0, tau.alpha)
     tau.u~dgamma(a.u, b.u)
```

```
    tau.b~dgamma(a.b, b.b)
    tau.alpha~dgamma(a.alpha, b.alpha)
}
```

We can try two different parametrizations for the hyperparameters of the gammas' distributions to estimate the α's, ϕs' and γs' precisions.

```
curve(dgamma(x,1,1), xlim = c(-.1,5), ylim = c(-.1,1), lwd = 2, ylab = '')
curve(dgamma(x,0.001,0.001), add = T, col = 'red', lwd = 2)
legend('topright', c('gamma(1,1)', 'gamma(0.01,0.01)'),
                    col = c('black', 'red'), lwd = c(2,2), cex = 1)
```



We can try 4 different models. In the first one and the third one the hyperparameters for the Gamma distributions are a=b=0.001. In order to balance the weigths of ϕ and γ, in the second model and the fourth one the hyperparameters are set as a=b=1. Finally to achieve better autocorrelations for the α parameter we can use the R2WinBugs to perform over-relaxation in the third and fourth model.

```
library(R2jags)
library(R2WinBUGS)
data1 = list(Nusers = NUsers, Nbooks = NBooks, ratings = ratings,
             a.u = .001, b.u = .001, a.b = .001, b.b = .001,
             a.alpha = .001, b.alpha = .001)
data2 = list(Nusers = NUsers, Nbooks = NBooks, ratings = ratings,
             a.u = 1, b.u = 1, a.b = 1, b.b = 1,
             a.alpha = 1, b.alpha = 1)

inits = list(alpha = 0,
             phi = rep(0,NUsers),
             gamma = rep(0,NBooks),
             tau.u = 0.1, tau.b = 0.1, tau.alpha = 0.1)
mod1 = jags(data = data1,
           inits = list(inits),
           n.chain = 1,
           parameters.to.save=c("tau.u", "tau.b", "tau.alpha",
                                "phi","gamma","alpha"),
           model.file="model01.txt",
           n.iter = 10000,
           n.burnin = 0,
           n.thin = 1)
```

```r
mod2 = jags(data = data2,
           inits = list(inits),
           n.chain = 1,
           parameters.to.save=c("tau.u", "tau.b", "tau.alpha",
                                 "phi","gamma","alpha"),
           model.file="model01.txt",
           n.iter = 10000,
           n.burnin = 0,
           n.thin = 1)
dir = 'C:/Users/cecin/Downloads/winbugs14_unrestricted/WinBUGS14'
wb.mod = bugs(data = data1,
              inits = list(inits),
              n.chains = 1,
              parameters.to.save=c("tau.u", "tau.b", "tau.alpha",
                                    "phi","gamma","alpha"),
              model.file = 'model01.txt',
              n.iter = 10000,
              n.burnin = 0,
              n.thin = 1,
              bugs.directory = dir,
              over.relax = T)

wb.mod2 = bugs(data = data2,
              inits = list(inits),
              n.chains = 1,
              parameters.to.save=c("tau.u", "tau.b", "tau.alpha",
                                    "phi","gamma","alpha"),
              model.file = 'model01.txt',
              n.iter = 10000,
              n.burnin = 0,
              n.thin = 1,
              bugs.directory = dir,
              over.relax = T)
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 158
##    Unobserved stochastic nodes: 97
##    Total graph size: 2055
##
## Initializing model
##
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 158
##    Unobserved stochastic nodes: 97
##    Total graph size: 2055
##
## Initializing model
```

```r
d = '----------------------------------------------------------------------\n'
print.jags.output2 = function(model1,model2,par,sd=F,r=2, t='') {
  m = par
  for(p in 1:(76-nchar(m))) {m=paste(m,'-',sep='')}
  cat(paste(m,'\n'))
```

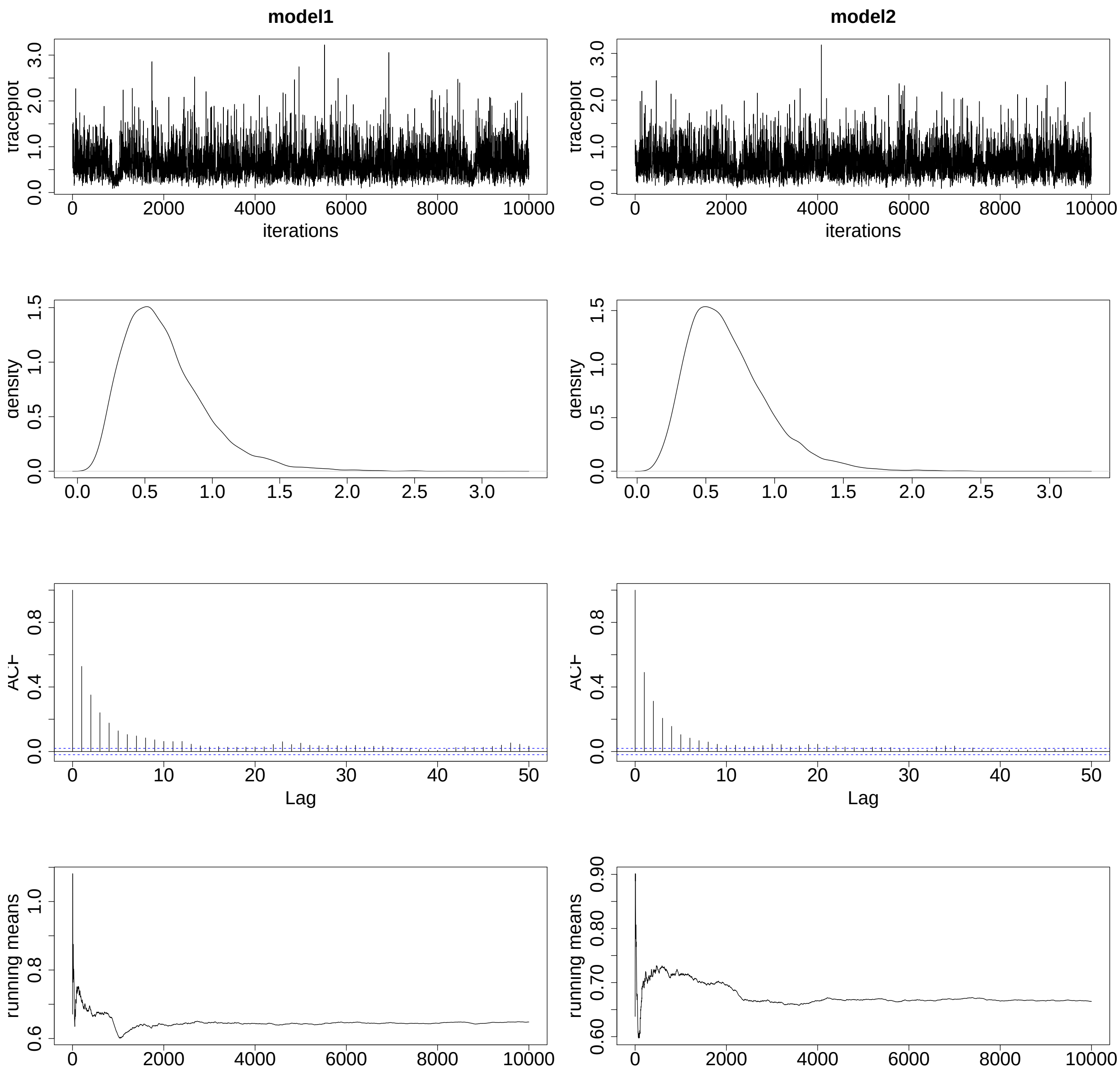
```
  if(sd) {
    print(c('mean1' = round(model1$mean[[par]], digits=r),
            'as.sd1' = round(1/sqrt(model1$mean[[par]]), digits=r),
            'sd1' = round(model1$sd[[par]], digits=r),
            'mean2' = round(model2$mean[[par]],digits=r),
            'as.sd1' = round(1/sqrt(model2$mean[[par]]), digits=r),
            'sd2' = round(model2$sd[[par]], digits=r)))
  }
  else{
    print(c('mean1' = round(model1$mean[[par]], digits=r),
            'sd1' = round(model1$sd[[par]], digits=r),
            'mean2' = round(model2$mean[[par]],digits=r),
            'sd2' = round(model2$sd[[par]], digits=r)))
  }
  par(mfrow = c(1,2))
  par(oma=c(0,0,2,0))
  plot(model1$sims.array[,1,par], type='l', xlab='iterations',
       ylab='traceplot', main='model1', cex.lab=2, cex.axis=2, cex.main=2)
  title(t, cex.main=2, outer=TRUE)
  plot(model2$sims.array[,1,par], type='l', xlab='iterations',
       ylab='traceplot', main='model2', cex.lab=2, cex.axis=2, cex.main=2)
  par(oma=c(0,0,0,0))
  n1 = length(model1$sims.array[,1,par])
  n2 = length(model2$sims.array[,1,par])
  plot(density(model1$sims.array[,1,par]), type='l', xlab='',
       ylab='density', main='', cex.lab=2, cex.axis=2, cex.main=2)
  plot(density(model2$sims.array[,1,par]), type='l', xlab='',
       ylab='density', main='', cex.lab=2, cex.axis=2, cex.main=2)
  acf(model1$sims.array[,1,par], lag.max=50,
     main='', cex.lab=2, cex.axis=2, cex.main=2)
  acf(model2$sims.array[,1,par], lag.max=50,
     main='', cex.lab=2, cex.axis=2, cex.main=2)
  plot(cumsum(model1$sims.array[,1,par])/1:n1, type="l", xlab = '',
       ylab='running means', main='', cex.lab=2, cex.axis=2, cex.main=2)
  plot(cumsum(model2$sims.array[,1,par])/1:n2, type="l", xlab = '',
       ylab='running means', main='', cex.lab=2, cex.axis=2, cex.main=2)
}

print.jags.output1 = function(model,par,t='',sd=F,r=2, wb=F) {
  if(wb==F) {
    model = model$BUGSoutput
  }
  m = par
  for(p in 1:(76-nchar(m))) {m=paste(m,'-',sep='')}
  cat(paste(m,'\n'))
  if(sd) {
    print(c('mean1' = round(model$mean[[par]], digits=r),
            'as.sd1' = round(1/sqrt(model$mean[[par]]), digits=r),
            'sd1' = round(model$sd[[par]], digits=r)))
  }
  else{
    print(c('mean1' = round(model$mean[[par]], digits=r),
            'sd1' = round(model$sd[[par]], digits=r)))
  }
  par(mfrow = c(1,2))
  #par(mar = c(2,2,6,2))
  par(oma=c(0,0,2,0))
  n = length(model$sims.array[,1,par])
```

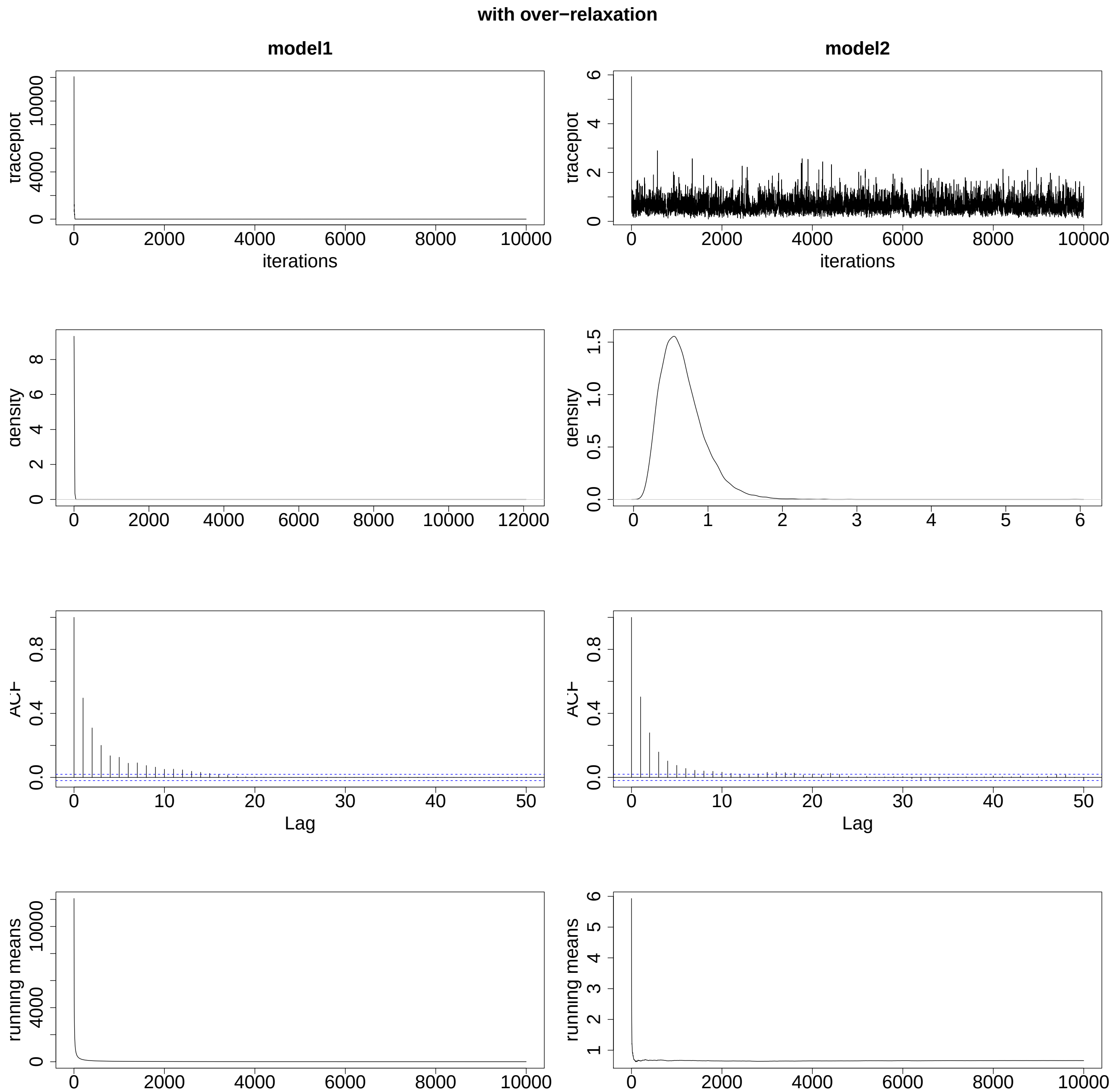
```
  plot(model$sims.array[,1,par], type='l', xlab='iterations',
       ylab='traceplot', main='', cex.lab=2, cex.axis=2, cex.main=2)
  title(t, cex.main=2, outer=TRUE)
  plot(density(model$sims.array[,1,par]), type='l', xlab='',
       ylab='density', main='', cex.lab=2, cex.axis=2, cex.main=2)
  acf(model$sims.array[,1,par], lag.max=50,
      main='', cex.lab=2, cex.axis=2, cex.main=2)
  plot(cumsum(model$sims.array[,1,par])/1:n, type="l", xlab = '',
       ylab='running means', main='', cex.lab=2, cex.axis=2, cex.main=2)
}

phi.gamma.jags.output = function(models, mnames, r=3) {
  pars = c('phi','gamma')
  for(par in pars) {
    m = par
    for(p in 1:(76-nchar(m))) {m=paste(m,'-',sep='')}
    cat(paste(m,'\n'))
    app = c()
    n = c()
    for(m in 1:length(models)) {
      app = c(app,c(round(mean(models[[m]]$mean[[par]]), digits=r),
                    round(sd(models[[m]]$mean[[par]]), digits=r)))
      n = c(n, c(paste('mean',m,sep='')), c(paste('sd',m,sep='')))
    }
    names(app) = n
    print(app)
  }
  par(mfrow = c(1,2))
  cols = c('green','blue','darkgreen')
  lwds = c(1,2,1)
  pchs = c(4,3,3)
  cexs = c(2,2,2)
  for(par in pars) {
    names = c('model1')
    colors = c('lightblue')
    ls = c(2)
    ps = c(4)
    cs = c(2)
    label = eval(parse(text=paste('expression(',par,')')))
    plot(models[[1]]$mean[[par]], pch=4, cex=2, col='lightblue',
         xlab=label, ylab='', ylim=c(-2,2), lwd=2,
         main = '', cex.lab = 2)
    for(m in 2:length(models)) {
      points(models[[m]]$mean[[par]], pch=pchs[m-1], cex=cexs[m-1],
             col=cols[m-1], lwd=lwds[m-1])
      #names = c(names,paste('model',m,sep=''))
      names = mnames
      colors = c(colors,c(cols[m-1]))
      ls=c(ls,c(lwds[m-1]))
      ps=c(ps,c(pchs[m-1]))
      cs=c(cs,c(cexs[m-1]))
    }
    legend('topleft', names, col=colors, pch=ps, pt.cex=cs,
           pt.lwd=ls)
  }
}
```

2.1 tau.u

```
print.jags.output2(mod1[[2]],mod2[[2]],'tau.u',sd=T)
print.jags.output2(wb.mod,wb.mod2, 'tau.u',t='with over-relaxation',sd=T)
```

```
## tau.u-----------------------------------------------------------------------
##   mean1 as.sd1    sd1  mean2 as.sd1    sd2
##    0.65   1.24   0.32   0.67   1.23   0.30
## tau.u-----------------------------------------------------------------------
##   mean1 as.sd1    sd1  mean2 as.sd1    sd2
##    3.70   0.52 139.86   0.66   1.23   0.30
```



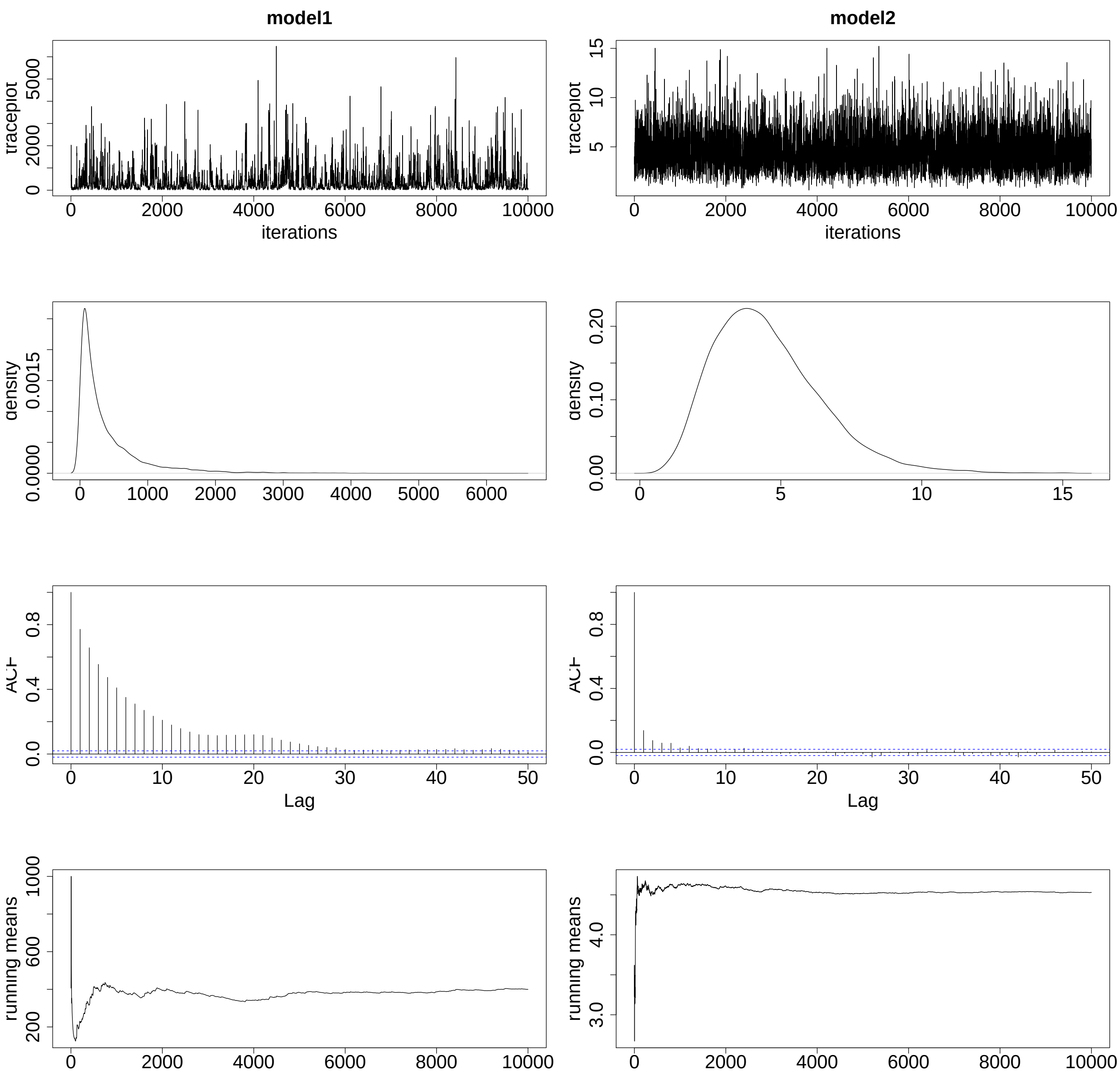


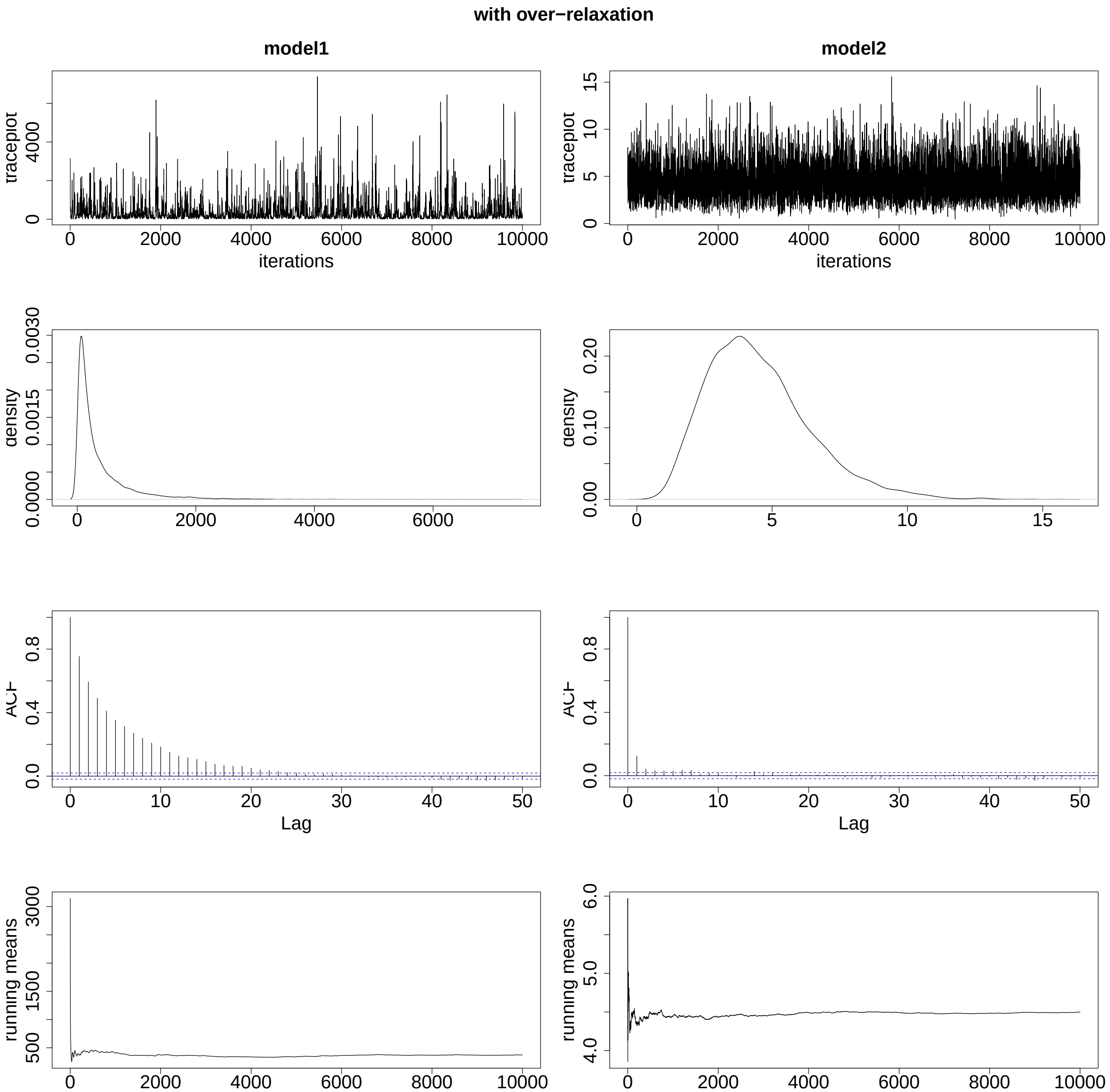
τ_u does not change so much between model1, model2 and model2(wo) but it significantly increases in model1(wo).

2.2 tau.b

```r
print.jags.output2(mod1[[2]],mod2[[2]],'tau.b', sd=T)
print.jags.output2(wb.mod,wb.mod2,'tau.b',t='with over-relaxation',sd=T)
```

```
## tau.b-----------------------------------------------------------------
##  mean1 as.sd1    sd1  mean2 as.sd1    sd2
## 399.35   0.05 521.45   4.53   0.47   1.92
## tau.b-----------------------------------------------------------------
##  mean1 as.sd1    sd1  mean2 as.sd1    sd2
## 373.05   0.05 520.75   4.50   0.47   1.91
```



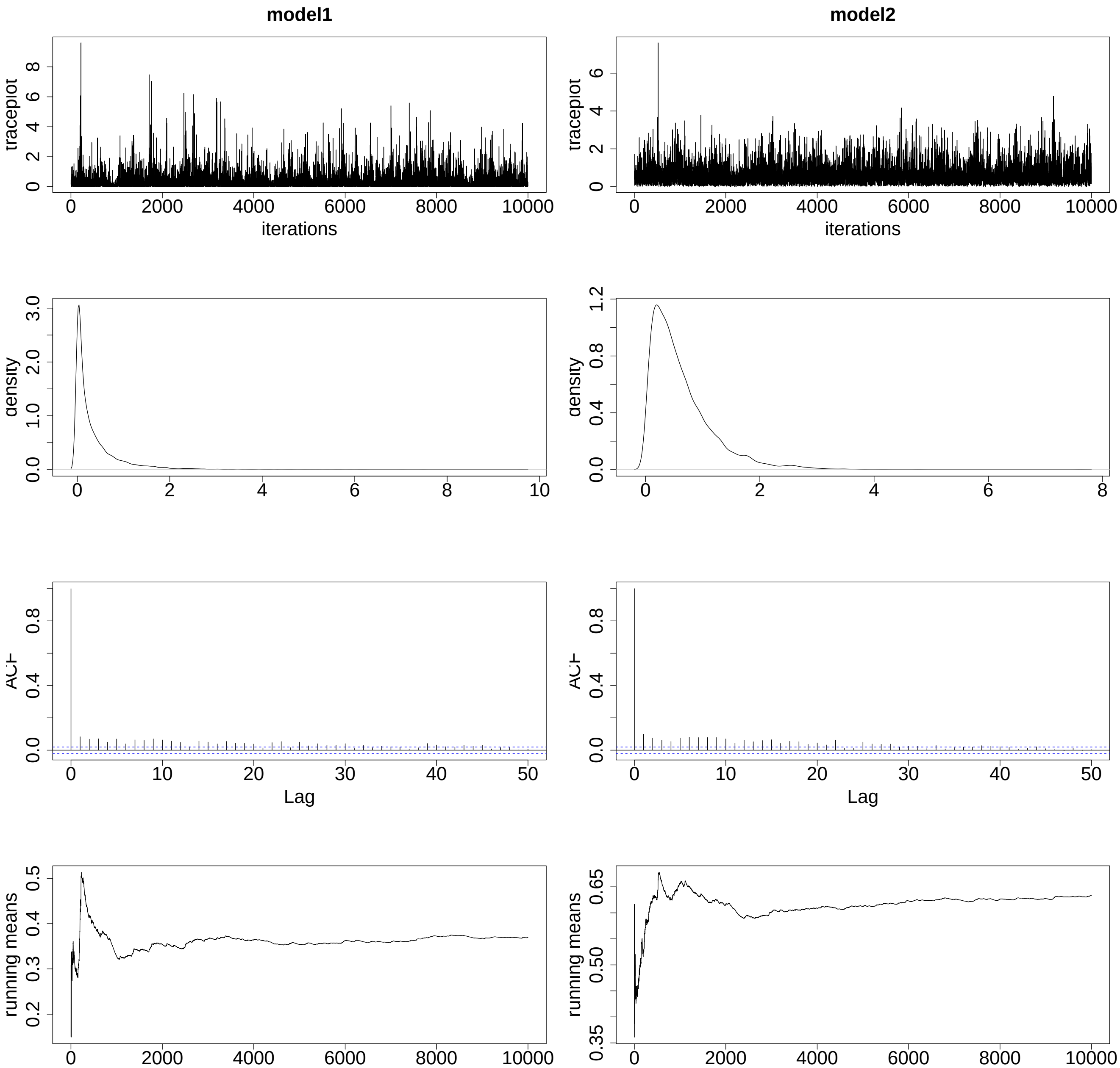


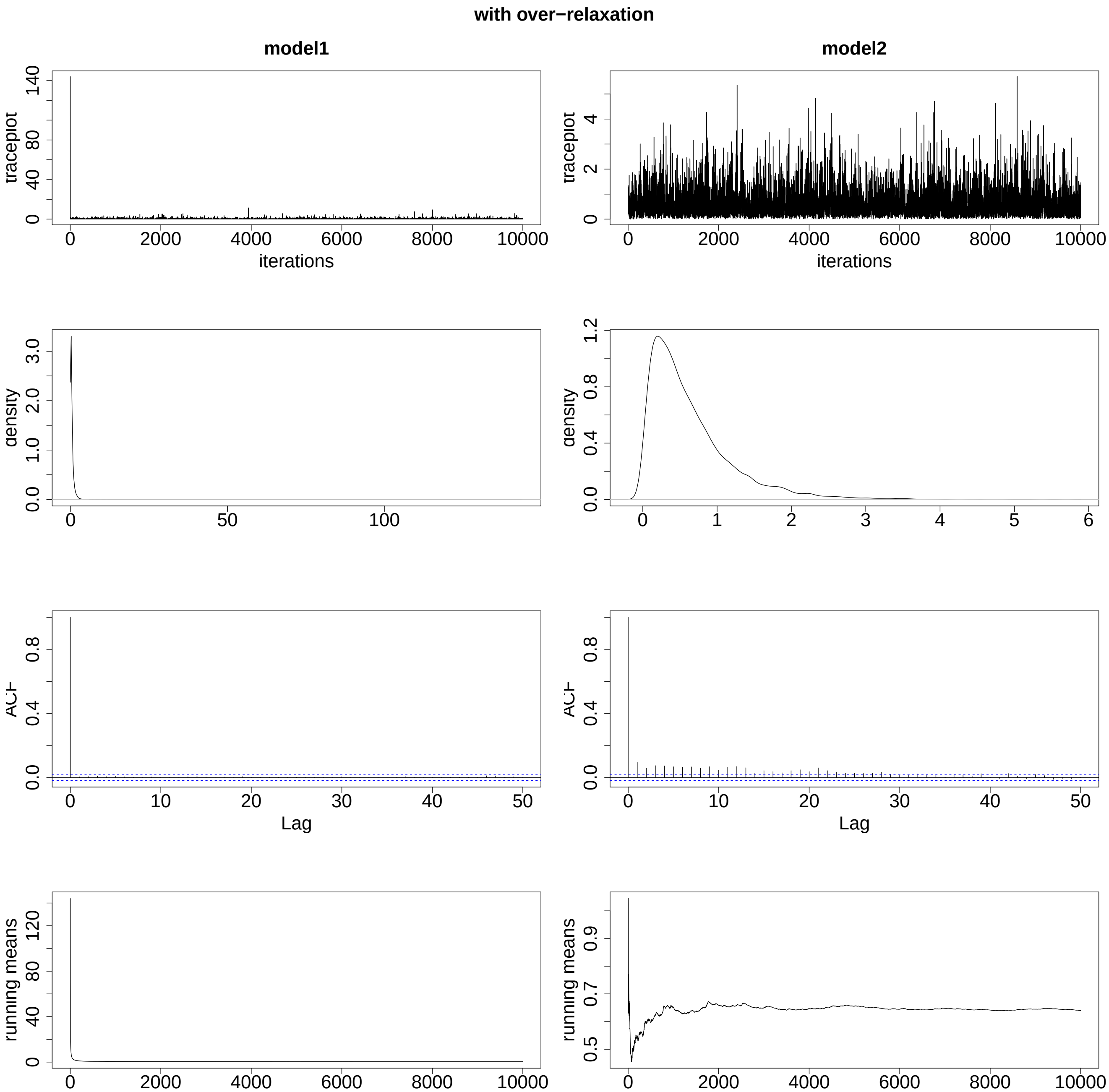
τ_b instead decreases a lot with the second hyperparameters (a=b=1), meaning that there are more differences between the books. Also the standard deviation and autocorrelations decrease.

2.3 tau.alpha

```
print.jags.output2(mod1[[2]],mod2[[2]],'tau.alpha', sd=T)
print.jags.output2(wb.mod,wb.mod2,'tau.alpha',t='with over-relaxation',sd=T)
```

```
## tau.alpha-------------------------------------------------------------------
##   mean1 as.sd1     sd1  mean2 as.sd1     sd2
##    0.37   1.65    0.58   0.63   1.26    0.55
## tau.alpha-------------------------------------------------------------------
##   mean1 as.sd1     sd1  mean2 as.sd1     sd2
##    0.36   1.67    1.54   0.64   1.25    0.57
```

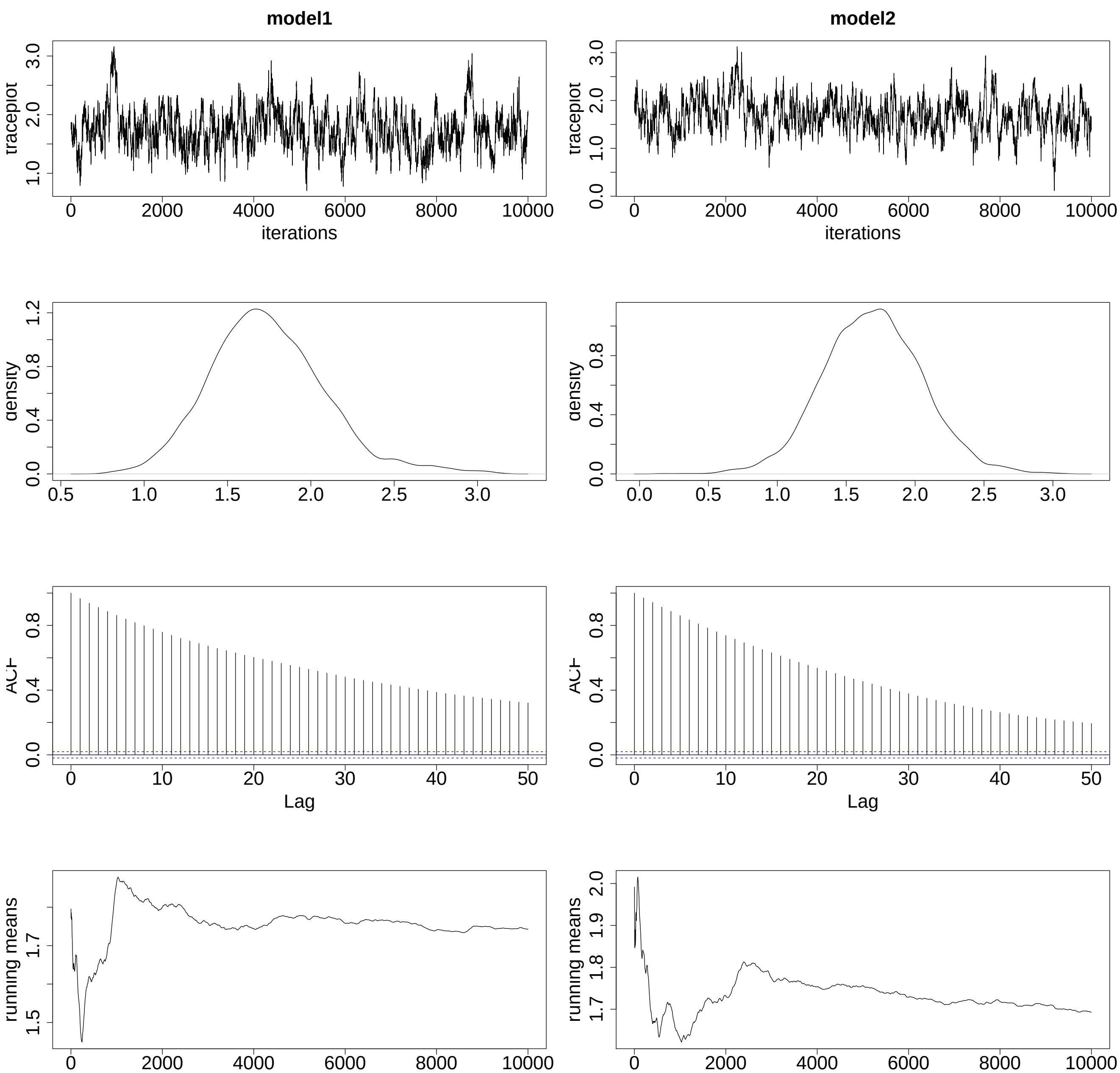


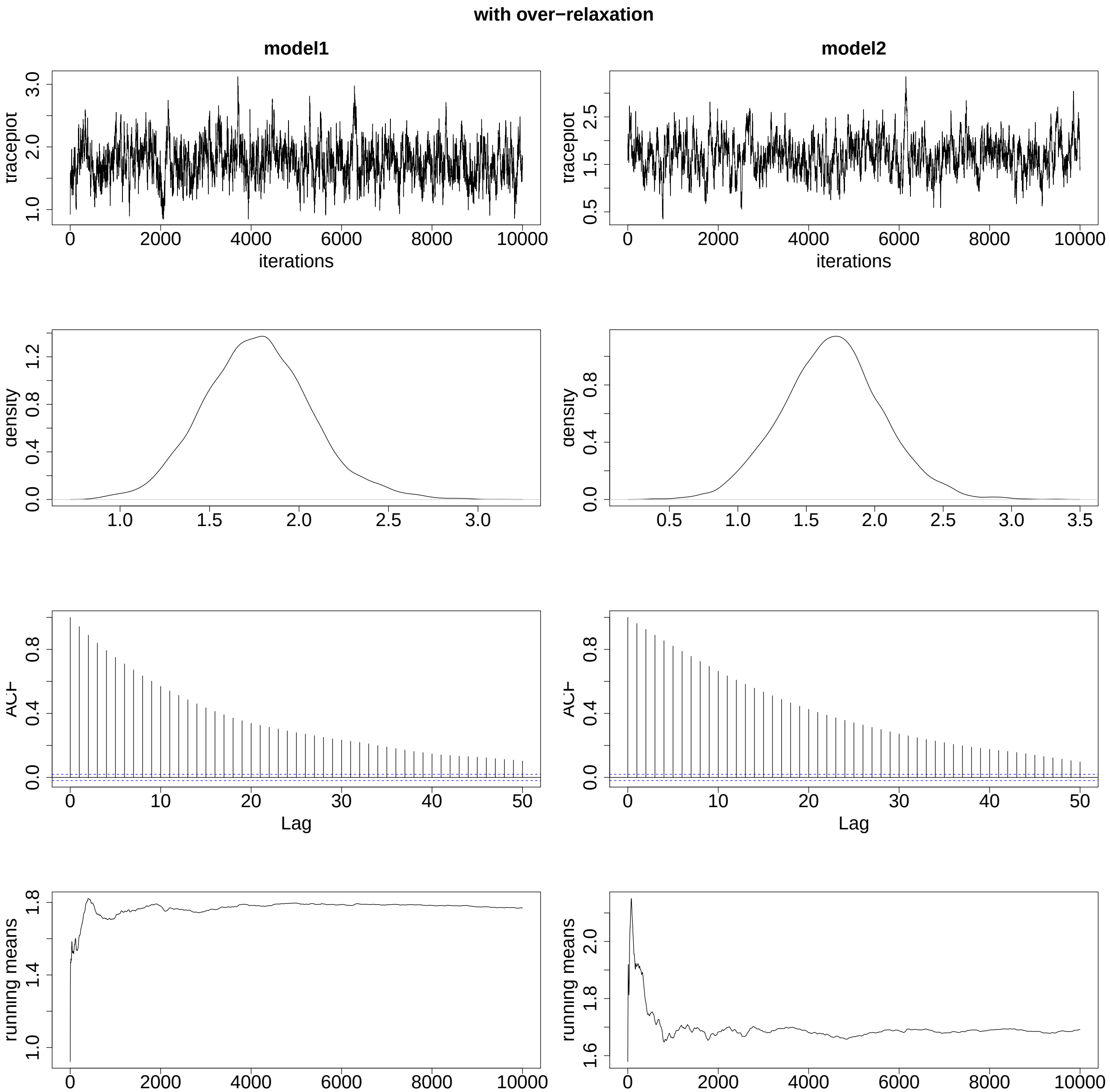

with over−relaxation
model1
model2
traceplot
iterations
density
ACF
Lag
running means

2.4 alpha

```r
print.jags.output2(mod1[[2]],mod2[[2]],'alpha')
print.jags.output2(wb.mod,wb.mod2,'alpha',t='with over-relaxation')
```

```
## alpha-----------------------------------------------------------------------
## mean1    sd1 mean2    sd2
##  1.74  0.35  1.69  0.36
## alpha-----------------------------------------------------------------------
## mean1    sd1 mean2    sd2
##  1.77  0.30  1.69  0.37
```



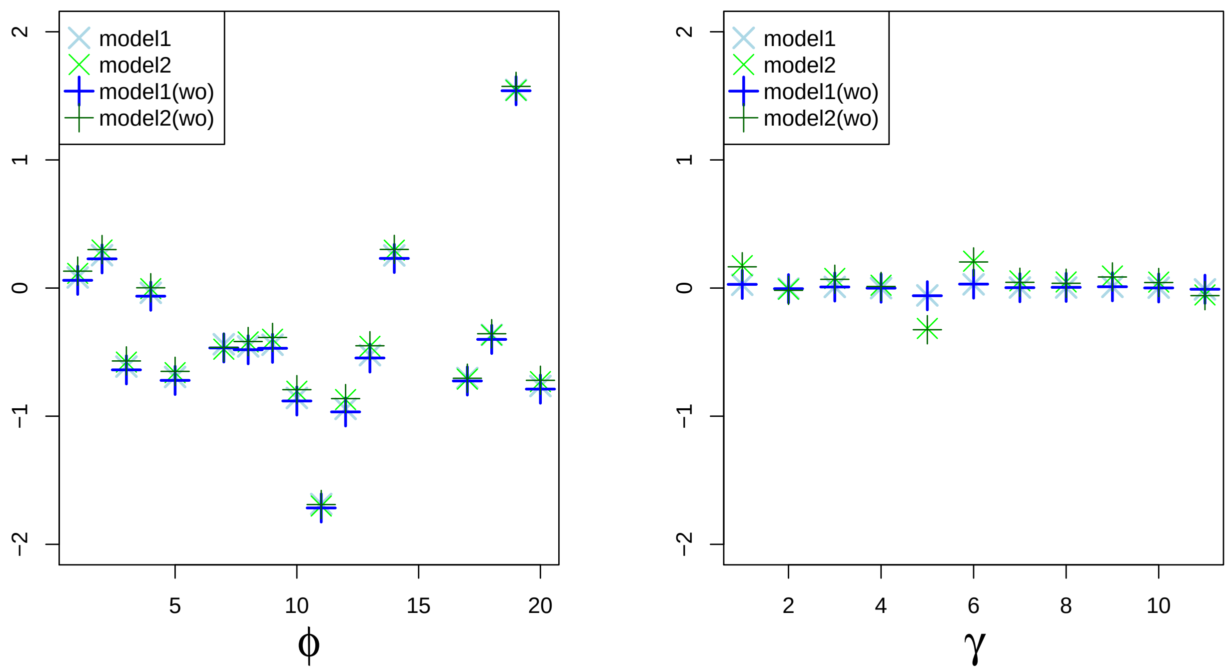


α remains quite stable in the four models. In this case even with lag=50 autocorrelations are high. They are lower with over-relaxation. $Logit(\alpha) \simeq 0.85$ multiplied by 9 is $\simeq 7.65$ which is near to the average rating.

2.5 phi and gamma

```
phi.gamma.jags.output(list(mod1[[2]],mod2[[2]],wb.mod,wb.mod2),
                       mnames = c('model1','model2','model1(wo)',
                                  'model2(wo)'))
cat('\n')
c('DIC1'=mod1[[2]]$DIC, 'DIC2'=mod2[[2]]$DIC, 'DIC1(wo)'=wb.mod$DIC,
  'DIC2(wo)'=wb.mod2$DIC)
```

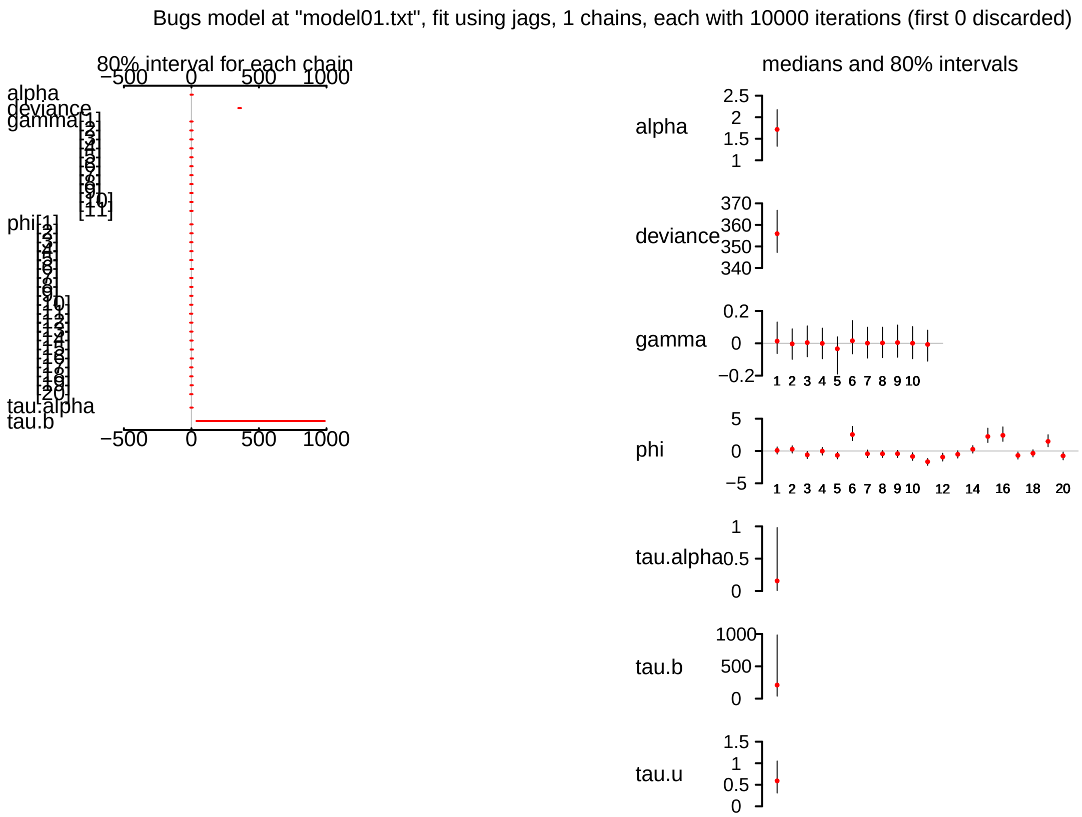
```
## phi------------------------------------------------------------------------
## mean1   sd1 mean2   sd2 mean3   sd3 mean4   sd4
## 0.058 1.233 0.082 1.227 0.034 1.237 0.093 1.230
## gamma----------------------------------------------------------------------
## mean1   sd1 mean2   sd2 mean3   sd3 mean4   sd4
## 0.001 0.023 0.032 0.137 0.001 0.024 0.024 0.138
##
##     DIC1     DIC2 DIC1(wo) DIC2(wo)
## 386.4318 395.5228 377.0440 385.6560
```



Finally we can visualize the different estimations for the ϕs and γs parameters. Model2 has bigger γs then the other models but not so much.

2.6 model1 results

```r
plot(mod1)
```

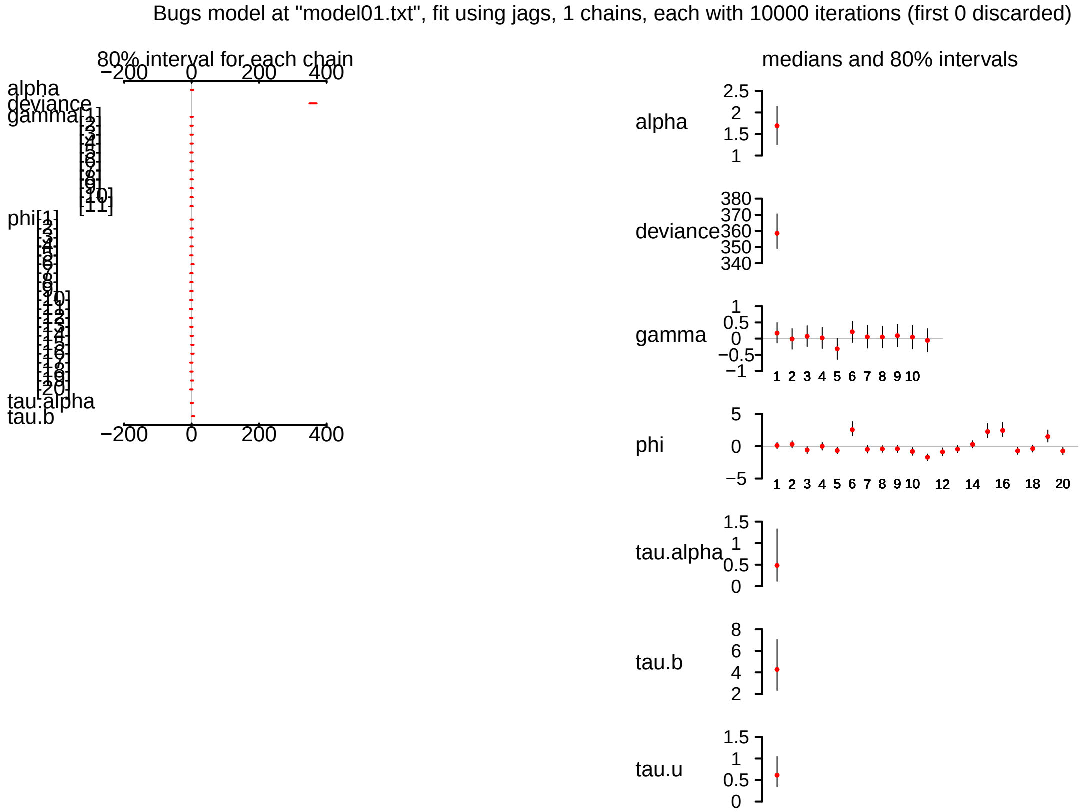


```r
m = 'model1'
for(p in 1:(76-nchar(m))) {m=paste(m,'-',sep='')}
cat(paste(m,'\n'))
mod1[[2]]
DIC1 = mod1[[2]]$DIC
dev1 = mod1[[2]]$mean[['deviance']]
```

```
## model1---------------------------------------------------------------------
## Inference for Bugs model at "model01.txt", fit using jags,
##  1 chains, each with 10000 iterations (first 0 discarded)
##  n.sims = 10000 iterations saved
##              mean    sd  2.5%   25%   50%   75%  97.5%
## alpha         1.7   0.4   1.1   1.5   1.7   2.0    2.5
## deviance    356.6   7.7 343.2 351.0 355.9 361.3  373.3
## gamma[1]      0.0   0.1  -0.1   0.0   0.0   0.1    0.2
## gamma[2]      0.0   0.1  -0.2   0.0   0.0   0.0    0.2
## gamma[3]      0.0   0.1  -0.2   0.0   0.0   0.0    0.2
## gamma[4]      0.0   0.1  -0.2   0.0   0.0   0.0    0.2
## gamma[5]     -0.1   0.1  -0.3  -0.1   0.0   0.0    0.1
## gamma[6]      0.0   0.1  -0.1   0.0   0.0   0.1    0.3
## gamma[7]      0.0   0.1  -0.2   0.0   0.0   0.0    0.2
## gamma[8]      0.0   0.1  -0.2   0.0   0.0   0.0    0.2
## gamma[9]      0.0   0.1  -0.2   0.0   0.0   0.0    0.2
## gamma[10]     0.0   0.1  -0.2   0.0   0.0   0.0    0.2
## gamma[11]     0.0   0.1  -0.2   0.0   0.0   0.0    0.2
## phi[1]        0.1   0.4  -0.8  -0.2   0.1   0.4    0.9
## phi[2]        0.3   0.4  -0.7   0.0   0.3   0.6    1.1
## phi[3]       -0.6   0.4  -1.6  -0.9  -0.6  -0.3    0.2
## phi[4]        0.0   0.5  -1.0  -0.3   0.0   0.3    0.9
## phi[5]       -0.7   0.4  -1.6  -1.0  -0.7  -0.4    0.1
## phi[6]        2.7   0.9   1.2   2.0   2.6   3.2    4.7
## phi[7]       -0.4   0.5  -1.4  -0.7  -0.4  -0.1    0.4
## phi[8]       -0.5   0.4  -1.3  -0.7  -0.4  -0.2    0.3
## phi[9]       -0.4   0.4  -1.4  -0.7  -0.4  -0.2    0.4
## phi[10]      -0.9   0.5  -1.8  -1.2  -0.8  -0.5    0.0
## phi[11]      -1.7   0.4  -2.6  -1.9  -1.7  -1.4   -0.9
## phi[12]      -0.9   0.5  -1.9  -1.2  -0.9  -0.6    0.0
## phi[13]      -0.5   0.5  -1.5  -0.8  -0.5  -0.2    0.3
## phi[14]       0.3   0.5  -0.7   0.0   0.3   0.6    1.1
## phi[15]       2.4   0.9   0.9   1.7   2.2   2.9    4.4
## phi[16]       2.5   0.9   1.1   1.9   2.4   3.0    4.7
## phi[17]      -0.7   0.4  -1.7  -1.0  -0.7  -0.4    0.1
## phi[18]      -0.4   0.4  -1.3  -0.6  -0.4  -0.1    0.5
## phi[19]       1.5   0.8   0.2   1.0   1.5   2.0    3.2
## phi[20]      -0.8   0.5  -1.7  -1.1  -0.8  -0.4    0.1
## tau.alpha     0.4   0.6   0.0   0.0   0.2   0.5    1.9
## tau.b       399.4 521.4  17.2  82.4 210.0 503.8 1861.0
## tau.u         0.6   0.3   0.2   0.4   0.6   0.8    1.4
##
## DIC info (using the rule, pD = var(deviance)/2)
## pD = 29.9 and DIC = 386.4
## DIC is an estimate of expected predictive error (lower deviance is better).
```

2.7 model2 results

```r
plot(mod2)
```

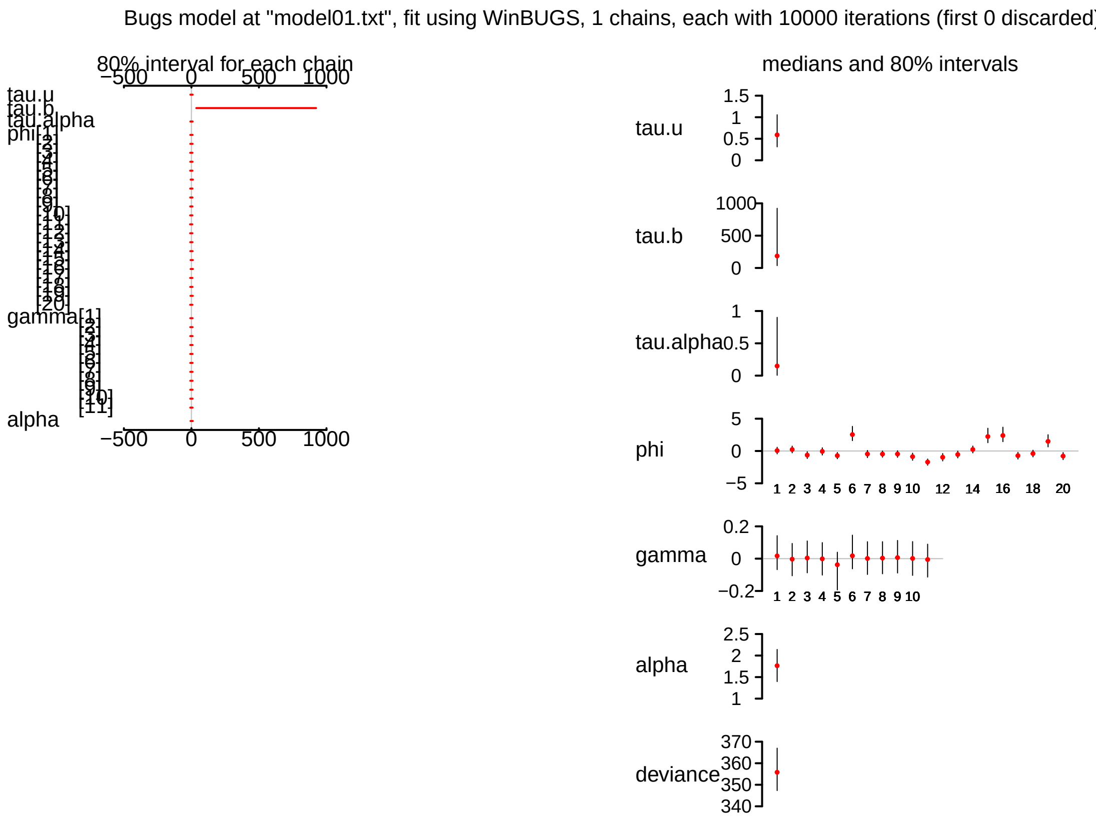


```r
m = 'model2'
for(p in 1:(76-nchar(m))) {m=paste(m,'-',sep='')}
cat(paste(m,'\n'))
mod2[[2]]
DIC2 = mod2[[2]]$DIC
dev2 = mod2[[2]]$mean[['deviance']]
```

```
## model2----------------------------------------------------------------------
## Inference for Bugs model at "model01.txt", fit using jags,
##  1 chains, each with 10000 iterations (first 0 discarded)
##  n.sims = 10000 iterations saved
##            mean  sd  2.5%   25%   50%   75% 97.5%
## alpha       1.7 0.4   1.0   1.5   1.7   1.9   2.4
## deviance  359.3 8.5 344.9 353.3 358.5 364.5 378.2
## gamma[1]    0.2 0.2  -0.3   0.0   0.2   0.3   0.7
## gamma[2]    0.0 0.3  -0.5  -0.2   0.0   0.2   0.5
## gamma[3]    0.1 0.3  -0.4  -0.1   0.1   0.2   0.6
## gamma[4]    0.0 0.3  -0.5  -0.2   0.0   0.2   0.5
## gamma[5]   -0.3 0.3  -0.8  -0.5  -0.3  -0.1   0.2
## gamma[6]    0.2 0.3  -0.3   0.0   0.2   0.4   0.7
## gamma[7]    0.1 0.3  -0.5  -0.1   0.1   0.2   0.6
## gamma[8]    0.0 0.3  -0.5  -0.1   0.0   0.2   0.6
## gamma[9]    0.1 0.3  -0.4  -0.1   0.1   0.3   0.6
## gamma[10]   0.0 0.3  -0.5  -0.1   0.0   0.2   0.6
## gamma[11]  -0.1 0.3  -0.6  -0.2  -0.1   0.1   0.5
## phi[1]      0.1 0.4  -0.7  -0.2   0.1   0.4   0.9
## phi[2]      0.3 0.4  -0.5   0.0   0.3   0.6   1.1
## phi[3]     -0.6 0.4  -1.4  -0.9  -0.6  -0.3   0.3
## phi[4]      0.0 0.5  -0.9  -0.3   0.0   0.3   0.9
## phi[5]     -0.7 0.4  -1.5  -0.9  -0.7  -0.4   0.1
## phi[6]      2.7 0.9   1.3   2.1   2.6   3.2   4.7
## phi[7]     -0.5 0.5  -1.4  -0.8  -0.5  -0.2   0.4
## phi[8]     -0.4 0.4  -1.2  -0.7  -0.4  -0.2   0.4
## phi[9]     -0.4 0.4  -1.2  -0.7  -0.4  -0.1   0.4
## phi[10]    -0.8 0.4  -1.7  -1.1  -0.8  -0.5   0.1
## phi[11]    -1.7 0.4  -2.5  -2.0  -1.7  -1.4  -0.9
## phi[12]    -0.9 0.5  -1.8  -1.2  -0.9  -0.5   0.0
## phi[13]    -0.5 0.4  -1.3  -0.7  -0.5  -0.2   0.4
## phi[14]     0.3 0.4  -0.5   0.0   0.3   0.6   1.2
## phi[15]     2.4 0.9   0.9   1.8   2.3   2.9   4.4
## phi[16]     2.5 0.9   1.1   1.9   2.4   3.0   4.5
## phi[17]    -0.7 0.4  -1.6  -1.0  -0.7  -0.4   0.1
## phi[18]    -0.4 0.4  -1.2  -0.6  -0.4  -0.1   0.5
## phi[19]     1.5 0.7   0.3   1.0   1.5   2.0   3.1
## phi[20]    -0.7 0.4  -1.6  -1.0  -0.7  -0.4   0.1
## tau.alpha   0.6 0.6   0.0   0.2   0.5   0.9   2.1
## tau.b       4.5 1.9   1.7   3.1   4.3   5.6   8.9
## tau.u       0.7 0.3   0.2   0.4   0.6   0.8   1.4
##
## DIC info (using the rule, pD = var(deviance)/2)
## pD = 36.2 and DIC = 395.5
## DIC is an estimate of expected predictive error (lower deviance is better).
```

2.8 model1 with over-relaxation results

```r
plot(wb.mod)
```

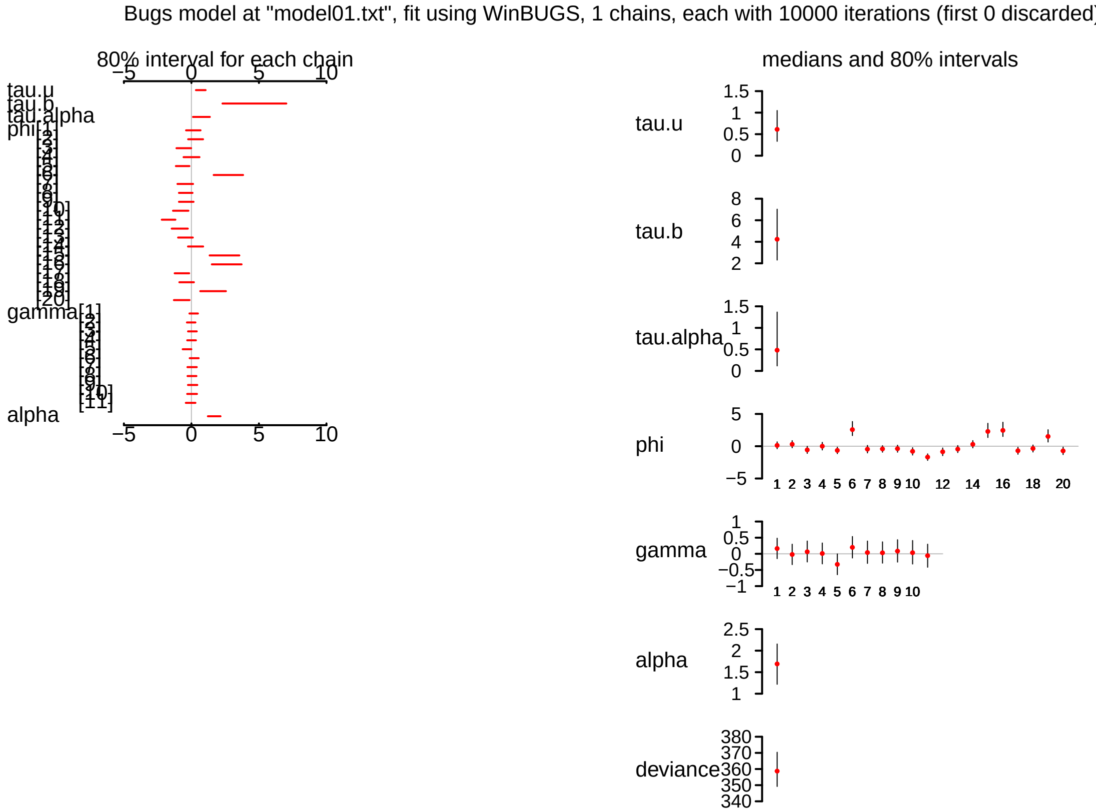


```r
m = 'model1 with over-relaxation'
for(p in 1:(76-nchar(m))) {m=paste(m,'-',sep='')}
cat(paste(m,'\n'))
wb.mod
DIC1wo = wb.mod$DIC
dev1wo = wb.mod$mean[['deviance']]
```

```
## model1 with over-relaxation-------------------------------------------------
## Inference for Bugs model at "model01.txt", fit using WinBUGS,
##  1 chains, each with 10000 iterations (first 0 discarded)
##  n.sims = 10000 iterations saved
##             mean    sd  2.5%   25%   50%   75%  97.5%
## tau.u        3.7 139.9   0.2   0.4   0.6   0.8    1.5
## tau.b      373.1 520.7  17.9  76.7 184.7 455.8 1860.0
## tau.alpha    0.4   1.5   0.0   0.0   0.1   0.4    1.8
## phi[1]       0.1   0.4  -0.7  -0.2   0.1   0.3    0.8
## phi[2]       0.2   0.4  -0.6   0.0   0.2   0.5    1.0
## phi[3]      -0.6   0.4  -1.4  -0.9  -0.6  -0.4    0.2
## phi[4]      -0.1   0.4  -0.9  -0.4  -0.1   0.2    0.8
## phi[5]      -0.7   0.4  -1.5  -1.0  -0.7  -0.5    0.0
## phi[6]       2.6   0.9   1.2   2.0   2.5   3.2    4.7
## phi[7]      -0.5   0.4  -1.3  -0.8  -0.5  -0.2    0.4
## phi[8]      -0.5   0.4  -1.3  -0.7  -0.5  -0.2    0.2
## phi[9]      -0.5   0.4  -1.3  -0.7  -0.5  -0.2    0.3
## phi[10]     -0.9   0.4  -1.7  -1.2  -0.9  -0.6   -0.1
## phi[11]     -1.7   0.4  -2.5  -2.0  -1.7  -1.5   -1.0
## phi[12]     -1.0   0.4  -1.9  -1.3  -1.0  -0.7   -0.1
## phi[13]     -0.5   0.4  -1.4  -0.8  -0.5  -0.3    0.3
## phi[14]      0.2   0.4  -0.6   0.0   0.2   0.5    1.0
## phi[15]      2.3   0.9   0.9   1.7   2.2   2.9    4.4
## phi[16]      2.5   0.9   1.0   1.9   2.4   3.0    4.6
## phi[17]     -0.7   0.4  -1.5  -1.0  -0.7  -0.5    0.1
## phi[18]     -0.4   0.4  -1.2  -0.7  -0.4  -0.1    0.4
## phi[19]      1.5   0.8   0.2   1.0   1.5   2.0    3.2
## phi[20]     -0.8   0.4  -1.7  -1.1  -0.8  -0.5    0.1
## gamma[1]     0.0   0.1  -0.1   0.0   0.0   0.1    0.3
## gamma[2]     0.0   0.1  -0.2   0.0   0.0   0.0    0.2
## gamma[3]     0.0   0.1  -0.2   0.0   0.0   0.0    0.2
## gamma[4]     0.0   0.1  -0.2   0.0   0.0   0.0    0.2
## gamma[5]    -0.1   0.1  -0.3  -0.1   0.0   0.0    0.1
## gamma[6]     0.0   0.1  -0.1   0.0   0.0   0.1    0.3
## gamma[7]     0.0   0.1  -0.2   0.0   0.0   0.0    0.2
## gamma[8]     0.0   0.1  -0.2   0.0   0.0   0.0    0.2
## gamma[9]     0.0   0.1  -0.2   0.0   0.0   0.1    0.2
## gamma[10]    0.0   0.1  -0.2   0.0   0.0   0.0    0.2
## gamma[11]    0.0   0.1  -0.2  -0.1   0.0   0.0    0.2
## alpha        1.8   0.3   1.2   1.6   1.8   2.0    2.4
## deviance   356.8  10.2 343.7 351.0 355.8 361.2  373.6
##
## DIC info (using the rule, pD = Dbar-Dhat)
## pD = 20.2 and DIC = 377.0
## DIC is an estimate of expected predictive error (lower deviance is better).
```

2.9 model2 with over-relaxation

```r
plot(wb.mod2)
```



```r
m = 'model2 with over-relaxation'
for(p in 1:(76-nchar(m))) {m=paste(m,'-',sep='')}
cat(paste(m,'\n'))
wb.mod2
DIC2wo = wb.mod2$DIC
dev2wo = wb.mod2$mean[['deviance']]
```

```
## model2 with over-relaxation-------------------------------------------------
## Inference for Bugs model at "model01.txt", fit using WinBUGS,
##  1 chains, each with 10000 iterations (first 0 discarded)
##  n.sims = 10000 iterations saved
##              mean  sd  2.5%   25%   50%   75% 97.5%
## tau.u         0.7 0.3   0.2   0.4   0.6   0.8   1.4
## tau.b         4.5 1.9   1.6   3.1   4.2   5.6   9.0
## tau.alpha     0.6 0.6   0.0   0.2   0.5   0.9   2.2
## phi[1]        0.1 0.4  -0.7  -0.1   0.1   0.4   1.0
## phi[2]        0.3 0.4  -0.6   0.0   0.3   0.6   1.2
## phi[3]       -0.6 0.4  -1.4  -0.8  -0.6  -0.3   0.3
## phi[4]        0.0 0.5  -0.9  -0.3   0.0   0.3   0.9
## phi[5]       -0.7 0.4  -1.4  -0.9  -0.6  -0.4   0.1
## phi[6]        2.7 0.9   1.2   2.1   2.6   3.2   4.7
## phi[7]       -0.5 0.5  -1.4  -0.8  -0.5  -0.2   0.4
## phi[8]       -0.4 0.4  -1.2  -0.7  -0.4  -0.1   0.4
## phi[9]       -0.4 0.4  -1.2  -0.7  -0.4  -0.1   0.5
## phi[10]      -0.8 0.5  -1.7  -1.1  -0.8  -0.5   0.1
## phi[11]      -1.7 0.4  -2.5  -1.9  -1.7  -1.4  -0.9
## phi[12]      -0.9 0.5  -1.8  -1.2  -0.9  -0.5   0.1
## phi[13]      -0.5 0.4  -1.3  -0.7  -0.5  -0.2   0.4
## phi[14]       0.3 0.4  -0.5   0.0   0.3   0.6   1.2
## phi[15]       2.4 0.9   0.9   1.8   2.3   2.9   4.5
## phi[16]       2.6 0.9   1.1   1.9   2.5   3.1   4.6
## phi[17]      -0.7 0.4  -1.5  -1.0  -0.7  -0.4   0.1
## phi[18]      -0.4 0.4  -1.2  -0.6  -0.4  -0.1   0.5
## phi[19]       1.6 0.8   0.3   1.0   1.5   2.0   3.3
## phi[20]      -0.7 0.5  -1.6  -1.0  -0.7  -0.4   0.2
## gamma[1]      0.2 0.2  -0.3   0.0   0.2   0.3   0.7
## gamma[2]      0.0 0.2  -0.5  -0.2   0.0   0.1   0.5
## gamma[3]      0.1 0.3  -0.4  -0.1   0.1   0.2   0.6
## gamma[4]      0.0 0.3  -0.5  -0.2   0.0   0.2   0.5
## gamma[5]     -0.3 0.3  -0.8  -0.5  -0.3  -0.2   0.2
## gamma[6]      0.2 0.3  -0.3   0.0   0.2   0.4   0.7
## gamma[7]      0.0 0.3  -0.5  -0.1   0.0   0.2   0.6
## gamma[8]      0.0 0.3  -0.5  -0.1   0.0   0.2   0.6
## gamma[9]      0.1 0.3  -0.4  -0.1   0.1   0.3   0.6
## gamma[10]     0.0 0.3  -0.5  -0.1   0.0   0.2   0.6
## gamma[11]    -0.1 0.3  -0.6  -0.2  -0.1   0.1   0.5
## alpha         1.7 0.4   1.0   1.4   1.7   1.9   2.4
## deviance    359.4 8.3 345.0 353.5 358.7 364.6 377.2
##
## DIC info (using the rule, pD = Dbar-Dhat)
## pD = 26.3 and DIC = 385.7
## DIC is an estimate of expected predictive error (lower deviance is better).
```

3. Simulations

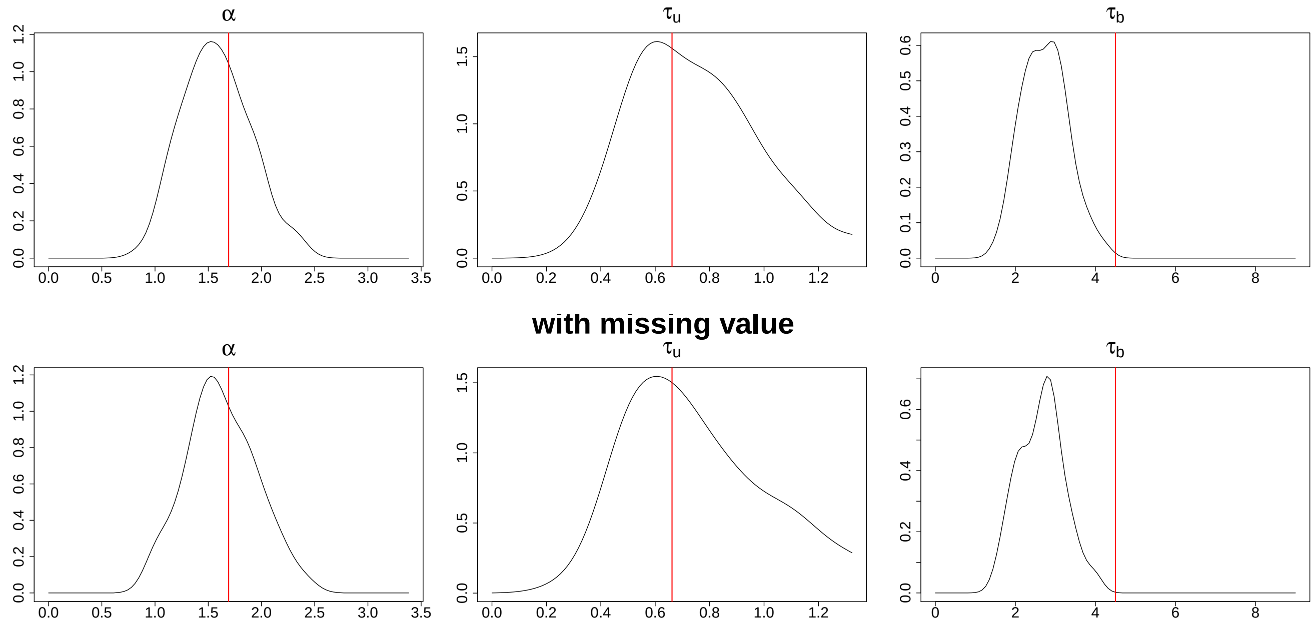
```r
alpha = wb.mod2$mean$alpha
tau.u = wb.mod2$mean$tau.u
```

```
tau.b = wb.mod2$mean$tau.b

run.model = function(p) {
  dir = 'C:/Users/cecin/Downloads/winbugs14_unrestricted/WinBUGS14'
  mod = bugs(data = p$data,
             inits = list(p$inits),
             n.chains = 1,
             parameters.to.save= p$pts,
             model.file = p$file,
             n.iter = p$niter,
             n.burnin = p$nburnin,
             n.thin = p$nthin,
             bugs.directory = dir,
             over.relax = T)
  return(mod)
}

inits = list(alpha = 0,
             phi = rep(0,NUsers),
             gamma = rep(0,NBooks),
             tau.u = 0.1, tau.b = 0.1, tau.alpha = 0.1)
pts = c("tau.u", "tau.b", "tau.alpha", "phi", "gamma", "alpha")
file = 'model01.txt'
data = list(Nusers = NUsers, Nbooks = NBooks,
            a.u = 1, b.u = 1, a.b = 1, b.b = 1,
            a.alpha = 1, b.alpha = 1)

model = list('data'=data, 'inits'=inits, 'pts'=pts, 'file'=file,
             'niter'=1000, 'nburnin'=0, 'nthin'=1)
n = 100
alpha.res = rep(NA,n)
tau.u.res = rep(NA,n)
tau.b.res = rep(NA,n)

alpha.nan.res = rep(NA,n)
tau.u.nan.res = rep(NA,n)
tau.b.nan.res = rep(NA,n)

p = round(.28*(NUsers*NBooks)) # NA

for(i in 1:n) {
  phi.sim = rnorm(NUsers,0,1/sqrt(tau.u))
  gamma.sim = rnorm(NBooks,0,1/sqrt(tau.b))
  ratings.sim = matrix(NA, NUsers, NBooks)
  for(r in 1:NUsers) {
    for(c in 1:NBooks) {
      logit.theta.sim = alpha + phi.sim[r] + gamma.sim[c]
      theta.sim = exp(logit.theta.sim)/(1+exp(logit.theta.sim))
      ratings.sim[r,c] = rbinom(1,9,theta.sim)
    }
  }
  model$data$ratings = structure(.Data = c(ratings.sim),
                                 .Dim = c(NUsers,NBooks))
  #print(model)
  res = run.model(model)
  alpha.res[i] = res$mean$alpha
  tau.u.res[i] = res$mean$tau.u
  tau.b.res[i] = res$mean$tau.b
```

```
  # with NA
  ratings.sim.nan = ratings.sim
  ratings.sim.nan[sample(1:length(ratings))[1:p]] = NA
  model$data$ratings = structure(.Data = c(ratings.sim.nan),
                                 .Dim = c(NUsers,NBooks))
  #print(model)
  res = run.model(model)
  alpha.nan.res[i] = res$mean$alpha
  tau.u.nan.res[i] = res$mean$tau.u
  tau.b.nan.res[i] = res$mean$tau.b
}
```

```
par(mfrow=c(1,3))
par(oma=c(0,0,2,0))
dens.alpha = approxfun(density(alpha.res)$x, density(alpha.res)$y,
                       yleft=0, yright=0)
dens.tau.u = approxfun(density(tau.u.res)$x, density(tau.u.res)$y,
                       yleft=0, yright=0)
dens.tau.b = approxfun(density(tau.b.res)$x, density(tau.b.res)$y,
                       yleft=0, yright=0)
plot(dens.alpha, main = expression(alpha), xlab = '', ylab = '',
     xlim = c(0, 2*alpha), cex.lab=2, cex.axis=2, cex.main=3)
abline(v=alpha, lwd = 1.5, col = 'red')
plot(dens.tau.u, main = expression(tau[u]), xlab = '', ylab = '',
     xlim = c(0, 2*tau.u), cex.lab=2, cex.axis=2, cex.main=3)
abline(v=tau.u, lwd = 1.5, col = 'red')
plot(dens.tau.b, main = expression(tau[b]), xlab = '', ylab = '',
     xlim = c(0, 2*tau.b), cex.lab=2, cex.axis=2, cex.main=3)
abline(v=tau.b, lwd = 1.5, col = 'red')
c('alpha'=round(integrate(dens.alpha, -Inf, alpha)$value,2),
  'tau.u'=round(integrate(dens.tau.u, -Inf, tau.u)$value,2),
  'tau.b'=round(integrate(dens.tau.b, -Inf, tau.b)$value,2))
dens.alpha.nan = approxfun(density(alpha.nan.res)$x, density(alpha.nan.res)$y,
                           yleft=0, yright=0)
dens.tau.u.nan = approxfun(density(tau.u.nan.res)$x, density(tau.u.nan.res)$y,
                           yleft=0, yright=0)
dens.tau.b.nan = approxfun(density(tau.b.nan.res)$x, density(tau.b.nan.res)$y,
                           yleft=0, yright=0)
plot(dens.alpha.nan, main = expression(alpha), xlab = '', ylab = '',
     xlim = c(0, 2*alpha), cex.lab=2, cex.axis=2, cex.main=3)
abline(v=alpha, lwd = 1.5, col = 'red')
title('with missing value', cex.main=4, outer=TRUE)
plot(dens.tau.u.nan, main = expression(tau[u]), xlab = '', ylab = '',
     xlim = c(0, 2*tau.u), cex.lab=2, cex.axis=2, cex.main=3)
abline(v=tau.u, lwd = 1.5, col = 'red')
plot(dens.tau.b.nan, main = expression(tau[b]), xlab = '', ylab = '',
     xlim = c(0, 2*tau.b), cex.lab=2, cex.axis=2, cex.main=3)
abline(v=tau.b, lwd = 1.5, col = 'red')
cat('with missing values\n')
c('alpha'=round(integrate(dens.alpha.nan, -Inf, alpha)$value,2),
  'tau.u'=round(integrate(dens.tau.u.nan, -Inf, tau.u)$value,2),
  'tau.b'=round(integrate(dens.tau.b.nan, -Inf, tau.b)$value,2))
```

```
## alpha tau.u tau.b
##  0.65  0.40  1.00
## with missing values
## alpha tau.u tau.b
##  0.59  0.41  1.00
```



Now we want to simulate the ratings from our model, fixing $\tau_u = 0.67$, $\tau_b = 4.53$ and $\alpha = 1.69$, to test how good we can recover those parameters. We run 100 simulations and then plot the distribution of the results. The red line is the real value of the parameter. The real dataset had 28% of missing values so in the first case we try to recover the parameters without missing values, in the second case instead I added 62 NA randomly. The two estimations are very similar. τ_b is underestimated in both cases.

4. Diagnostics

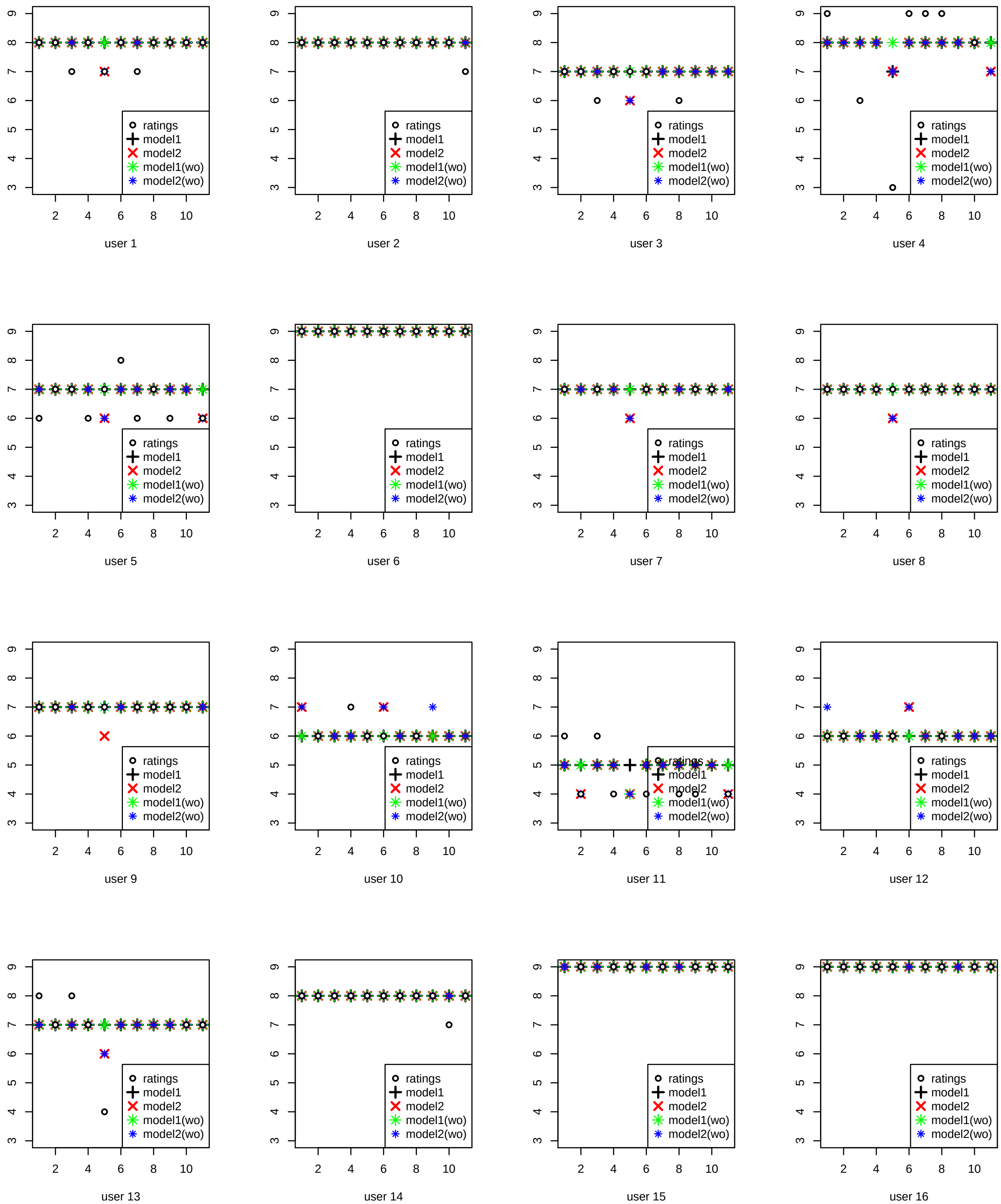
```
n = 10000
ratings.hat1 = matrix(NA, NUsers, NBooks)
ratings.hat2 = matrix(NA, NUsers, NBooks)
ratings.hat3 = matrix(NA, NUsers, NBooks)
ratings.hat4 = matrix(NA, NUsers, NBooks)
alpha.sim1 = mod1$BUGSoutput$sims.array[,1,'alpha']
alpha.sim2 = mod2$BUGSoutput$sims.array[,1,'alpha']
alpha.sim3 = wb.mod$sims.array[,1,'alpha']
alpha.sim4 = wb.mod2$sims.array[,1,'alpha']

for(i in 1:NUsers) {
  phi.sim1 = mod1$BUGSoutput$sims.array[,1,paste('phi[',i,']', sep = '')]
  phi.sim2 = mod2$BUGSoutput$sims.array[,1,paste('phi[',i,']', sep = '')]
  phi.sim3 = wb.mod$sims.array[,1,paste('phi[',i,']', sep = '')]
  phi.sim4 = wb.mod2$sims.array[,1,paste('phi[',i,']', sep = '')]
  for(j in 1:NBooks) {
    gamma.sim1 = mod1$BUGSoutput$sims.array[,1,paste('gamma[',j,']', sep = '')]
    gamma.sim2 = mod2$BUGSoutput$sims.array[,1,paste('gamma[',j,']', sep = '')]
    gamma.sim3 = wb.mod$sims.array[,1,paste('gamma[',j,']', sep = '')]
    gamma.sim4 = wb.mod2$sims.array[,1,paste('gamma[',j,']', sep = '')]
    r1 = rep(NA, n)
    r2 = rep(NA, n)
    r3 = rep(NA, n)
    r4 = rep(NA, n)
    for(k in 1:n) {
      logit.theta1 = alpha.sim1[k] + phi.sim1[k] + gamma.sim1[k]
      logit.theta2 = alpha.sim2[k] + phi.sim2[k] + gamma.sim2[k]
      logit.theta3 = alpha.sim3[k] + phi.sim3[k] + gamma.sim3[k]
      logit.theta4 = alpha.sim4[k] + phi.sim4[k] + gamma.sim4[k]
      theta1 =  exp(logit.theta1)/(1+exp(logit.theta1))
      theta2 =  exp(logit.theta2)/(1+exp(logit.theta2))
      theta3 =  exp(logit.theta3)/(1+exp(logit.theta3))
      theta4 =  exp(logit.theta4)/(1+exp(logit.theta4))
      r1[k] = rbinom(1,9,theta1)
      r2[k] = rbinom(1,9,theta2)
      r3[k] = rbinom(1,9,theta3)
      r4[k] = rbinom(1,9,theta4)
    }
    ratings.hat1[i,j] = round(mean(r1))
    ratings.hat2[i,j] = round(mean(r2))
    ratings.hat3[i,j] = round(mean(r3))
    ratings.hat4[i,j] = round(mean(r4))
  }
}
m = 'MODEL1'
for(p in 1:(76-nchar(m))) {m=paste(m,'-',sep='')}
cat(paste(m,'\n'))
cat(paste('sum of errors:',sum(abs(ratings.hat1-ratings), na.rm=T),
          '\n'))
cat(paste('precision:',
          round(sum((ratings.hat1-ratings)==0, na.rm=T)/sum(is.na(ratings)==F),3),
          '\n'))
m = 'MODEL2'
for(p in 1:(76-nchar(m))) {m=paste(m,'-',sep='')}
cat(paste(m,'\n'))
cat(paste('sum of errors:',sum(abs(ratings.hat2-ratings), na.rm=T),
```

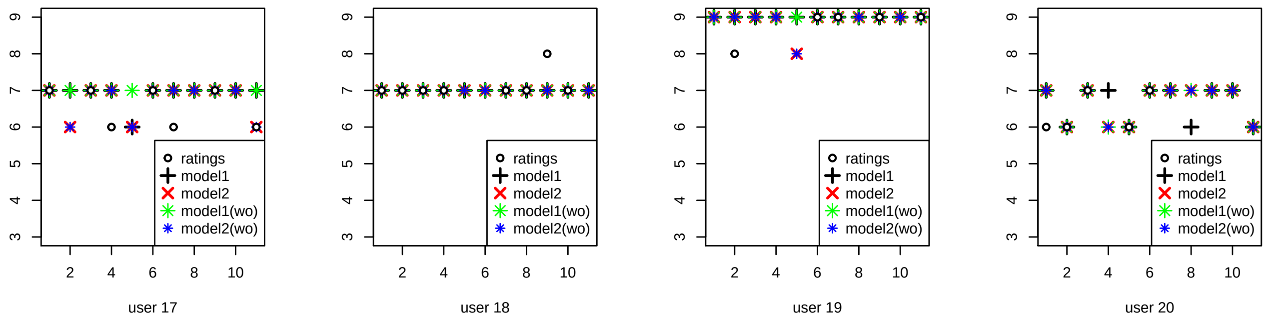
```
          '\n'))
cat(paste('precision:',
          round(sum((ratings.hat2-ratings)==0, na.rm=T)/sum(is.na(ratings)==F),3),
          '\n'))
m = 'MODEL1(wo)'
for(p in 1:(76-nchar(m))) {m=paste(m,'-',sep='')}
cat(paste(m,'\n'))
cat(paste('sum of errors:',sum(abs(ratings.hat3-ratings), na.rm=T),
          '\n'))
cat(paste('precision:',
          round(sum((ratings.hat3-ratings)==0, na.rm=T)/sum(is.na(ratings)==F),3),
          '\n'))

m = 'MODEL2(wo)'
for(p in 1:(76-nchar(m))) {m=paste(m,'-',sep='')}
cat(paste(m,'\n'))
cat(paste('sum of errors:',sum(abs(ratings.hat4-ratings), na.rm=T),
          '\n'))
cat(paste('precision:',
          round(sum((ratings.hat4-ratings)==0, na.rm=T)/sum(is.na(ratings)==F),3),
          '\n'))
```

```
## MODEL1----------------------------------------------------------------------
## sum of errors: 43
## precision: 0.766
## MODEL2----------------------------------------------------------------------
## sum of errors: 42
## precision: 0.766
## MODEL1(wo)------------------------------------------------------------------
## sum of errors: 44
## precision: 0.766
## MODEL2(wo)------------------------------------------------------------------
## sum of errors: 42
## precision: 0.766
```

```
par(mfrow=c(1,4))
plot(ratings.hat1[1,], pch=3, cex=1.5, col='black', lwd=2,
     xlab=paste('user',1), ylab = '', ylim=c(3,9))
points(ratings.hat2[1,], pch=4, cex=1.5, col='red', lwd=2)
points(ratings.hat3[1,], pch=8, cex=1.5, col='green', lwd=1)
points(ratings.hat4[1,], pch=8, cex=1, col='blue', lwd=1)
points(ratings[1,], pch=21, cex=1, bg='white', lwd=1.5)
legend('bottomright', c('ratings','model1','model2','model1(wo)','model2(wo)'),
       col=c('black','black','red','green','blue'), pch=c(21,3,4,8,8),
       pt.cex=c(1,1.5,1.5,1.5,1), pt.lwd = c(1.5,2,2,1,1))
title("\n Users' ratings", outer=TRUE, cex.main =2)
for(i in 2:NUsers) {
  plot(ratings.hat1[i,], pch=3, cex=1.5, col='black', lwd=2,
       xlab=paste('user',i), ylab = '', ylim=c(3,9))
  points(ratings.hat2[i,], pch=4, cex=1.5, col='red', lwd=2)
  points(ratings.hat3[i,], pch=8, cex=1.5, col='green', lwd=1)
  points(ratings.hat4[i,], pch=8, cex=1, col='blue', lwd=1)
  points(ratings[i,], pch=21, cex=1, bg='white', lwd=1.5)
  legend('bottomright', c('ratings','model1','model2','model1(wo)','model2(wo)'),
         col=c('black','black','red','green','blue'), pch=c(21,3,4,8,8),
         pt.cex=c(1,1.5,1.5,1.5,1), pt.lwd = c(1.5,2,2,1,1))
}
```

Users' ratings

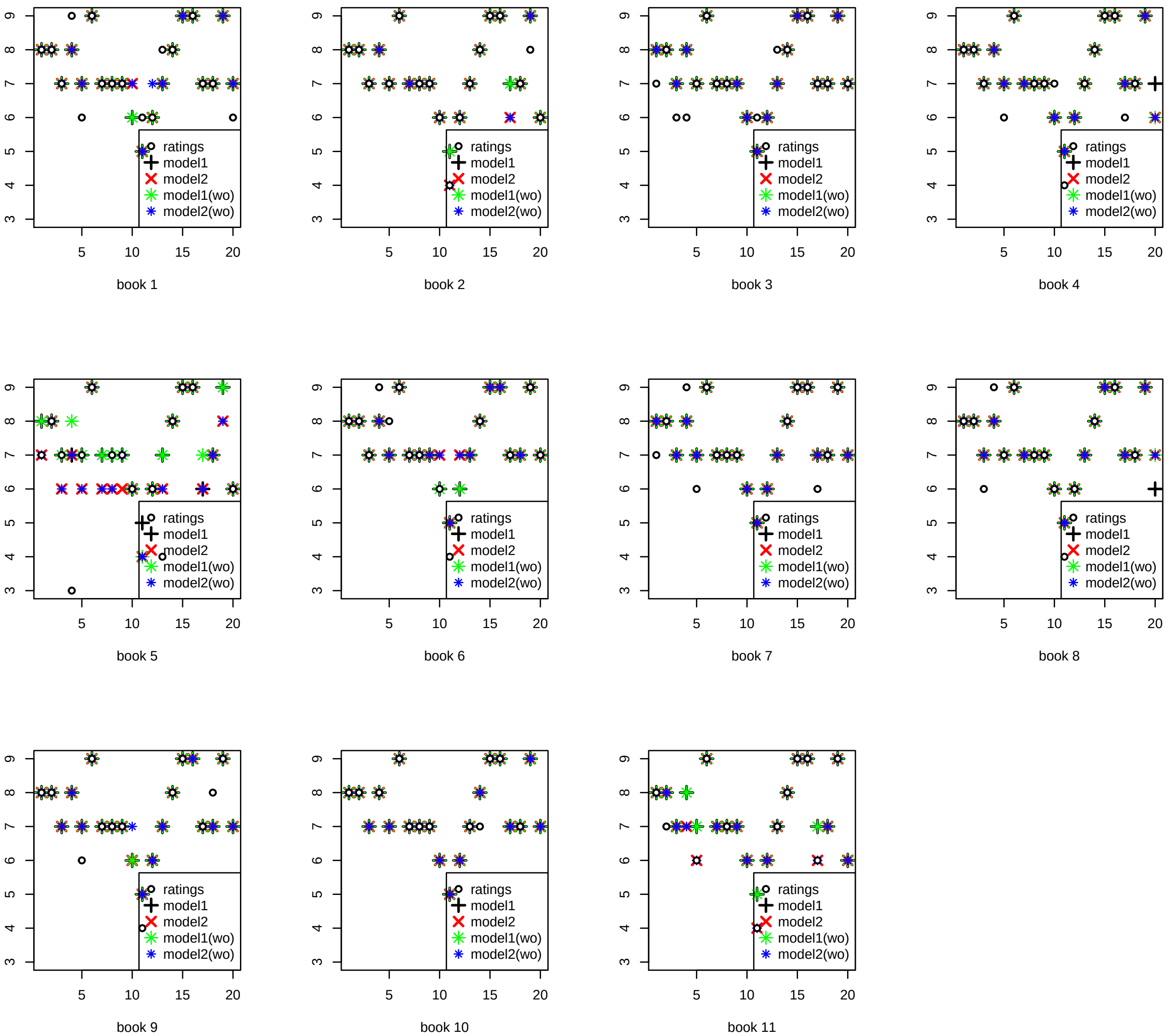




Model2 for book5 has the biggest γ(=-0.3).

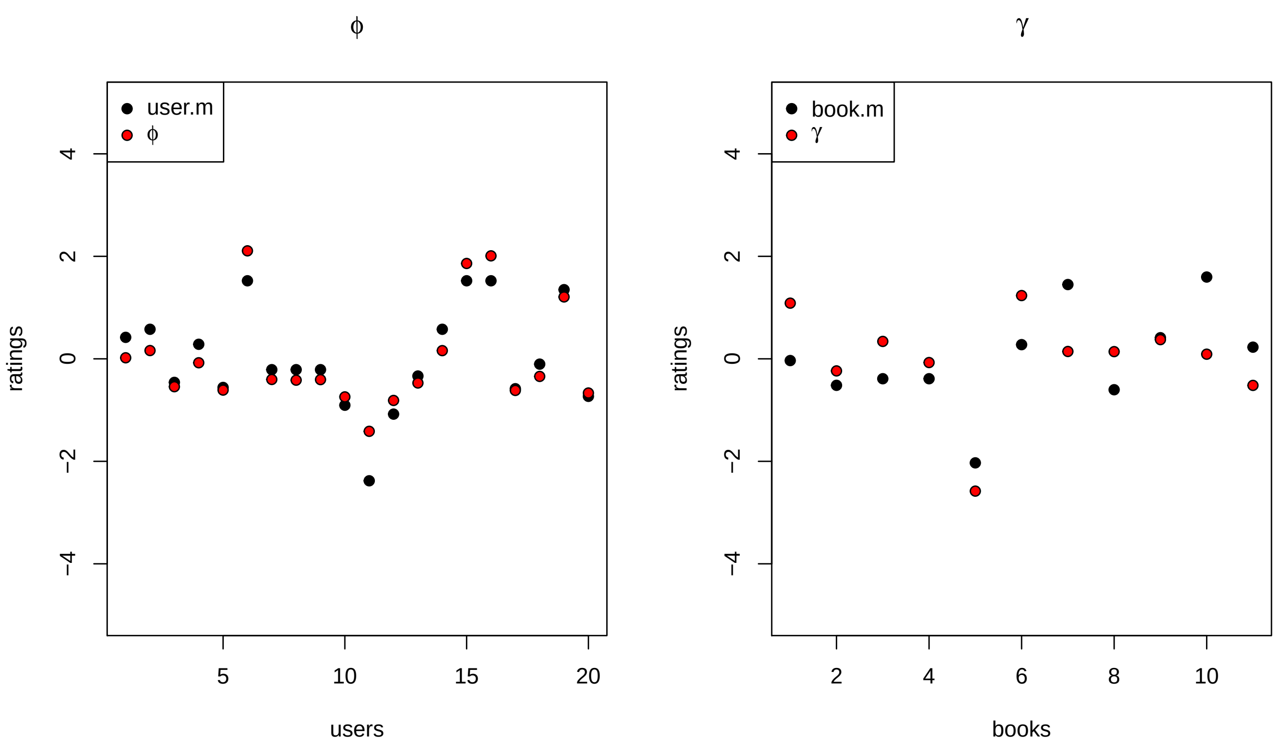
```
par(mfrow=c(1,4))
plot(ratings.hat1[,1], pch=3, cex=1.5, col='black', lwd=2,
     xlab=paste('book',1), ylab = '', ylim=c(3,9))
points(ratings.hat2[,1], pch=4, cex=1.5, col='red', lwd=2)
points(ratings.hat3[,1], pch=8, cex=1.5, col='green', lwd=1)
points(ratings.hat4[,1], pch=8, cex=1, col='blue', lwd=1)
points(ratings[,1], pch=21, cex=1, bg='white', lwd=1.5)
legend('bottomright', c('ratings','model1','model2','model1(wo)','model2(wo)'),
       col=c('black','black','red','green','blue'), pch=c(21,3,4,8,8),
       pt.cex=c(1,1.5,1.5,1.5,1), pt.lwd = c(1.5,2,2,1,1))
title("\n Books' ratings", outer=TRUE, cex.main = 2)
for(i in 2:NBooks) {
  plot(ratings.hat1[,i], pch=3, cex=1.5, col='black', lwd=2,
       xlab=paste('book',i), ylab = '', ylim=c(3,9))
  points(ratings.hat2[,i], pch=4, cex=1.5, col='red', lwd=2)
  points(ratings.hat3[,i], pch=8, cex=1.5, col='green', lwd=1)
  points(ratings.hat4[,i], pch=8, cex=1, col='blue', lwd=1)
  points(ratings[,i], pch=21, cex=1, bg='white', lwd=1.5)
  legend('bottomright', c('ratings','model1','model2','model1(wo)','model2(wo)'),
         col=c('black','black','red','green','blue'), pch=c(21,3,4,8,8),
         pt.cex=c(1,1.5,1.5,1.5,1), pt.lwd = c(1.5,2,2,1,1))
}
```

Books' ratings



5. Users' and books' averages

```
mod = mod1[[2]]
par(mfrow = c(1,2))
phi = mod$mean$phi

plot((user.m-mean(user.m))/sd(user.m), pch=21, cex=1, bg='black',
     ylim=c(-5,5), xlab='users', ylab='ratings',
     main=expression(phi))
points((phi-mean(phi))/sd(phi), pch=21, cex=1, bg='red')
legend('topleft', c('user.m', expression(phi)),
       pch=21, pt.bg=c('black', 'red'), cex=1)


gamma = mod$mean$gamma

plot((book.m-mean(book.m))/sd(book.m), pch=21, cex=1, bg='black',
     ylim=c(-5,5), xlab='books', ylab='ratings',
     main=expression(gamma))
points((gamma-mean(gamma))/sd(gamma), pch=21, cex=1, bg='red')
legend('topleft', c('book.m', expression(gamma)),
       pch = 21, pt.bg=c('black', 'red'), cex=1)
```



We can try to plot together the users' means with the ϕs' estimations, both standardized. The same for the books' means with the γs' estimations. The ϕs are nearer to the respective means.

6. Model: users' and books' averages

In this model we can use ϕ and γ as weights for the user's and book's means.

MODEL

```
model {
     for (i in 1:Nusers) {
          for (j in 1:Nbooks) {
               ratings[i,j] ~dbin(theta[i,j],9)
               logit.theta[i,j] <- alpha + phi*(user.m[i]-7.35) + gamma*(book.m[j]-7.35)
               theta[i,j] <- exp(logit.theta[i,j])/(1+exp(logit.theta[i,j]))
          }
     }
     alpha~dnorm(0, 0.1)
     phi~dnorm(0, 0.1)
     gamma~dnorm(0, 0.1)
}
```

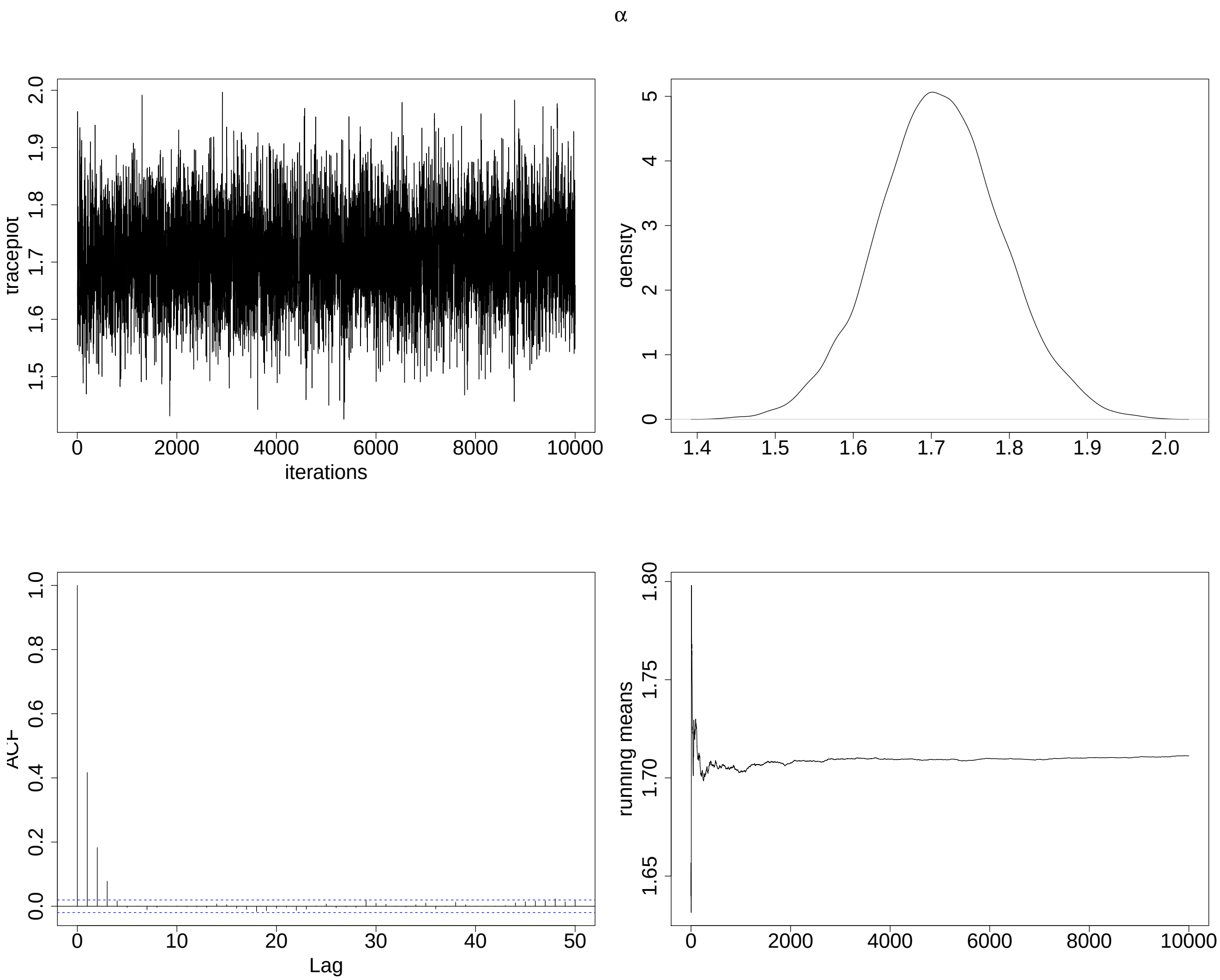
```r
data = list(Nusers = NUsers, Nbooks = NBooks, ratings = ratings,
            user.m = user.m, book.m = book.m)
# model

inits = list(alpha = 1, phi = 0, gamma = 0)
mod = jags(data = data,
           inits = list(inits),
           n.chain = 1,
           parameters.to.save=c("alpha","phi","gamma"),
           model.file="model04.txt",
           n.iter = 10000,
           n.burnin = 0,
           n.thin = 1)
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 158
##    Unobserved stochastic nodes: 65
##    Total graph size: 2232
##
## Initializing model
```

6.1 alpha

```r
print.jags.output1(mod, 'alpha', t=expression(alpha), wb=F)
```

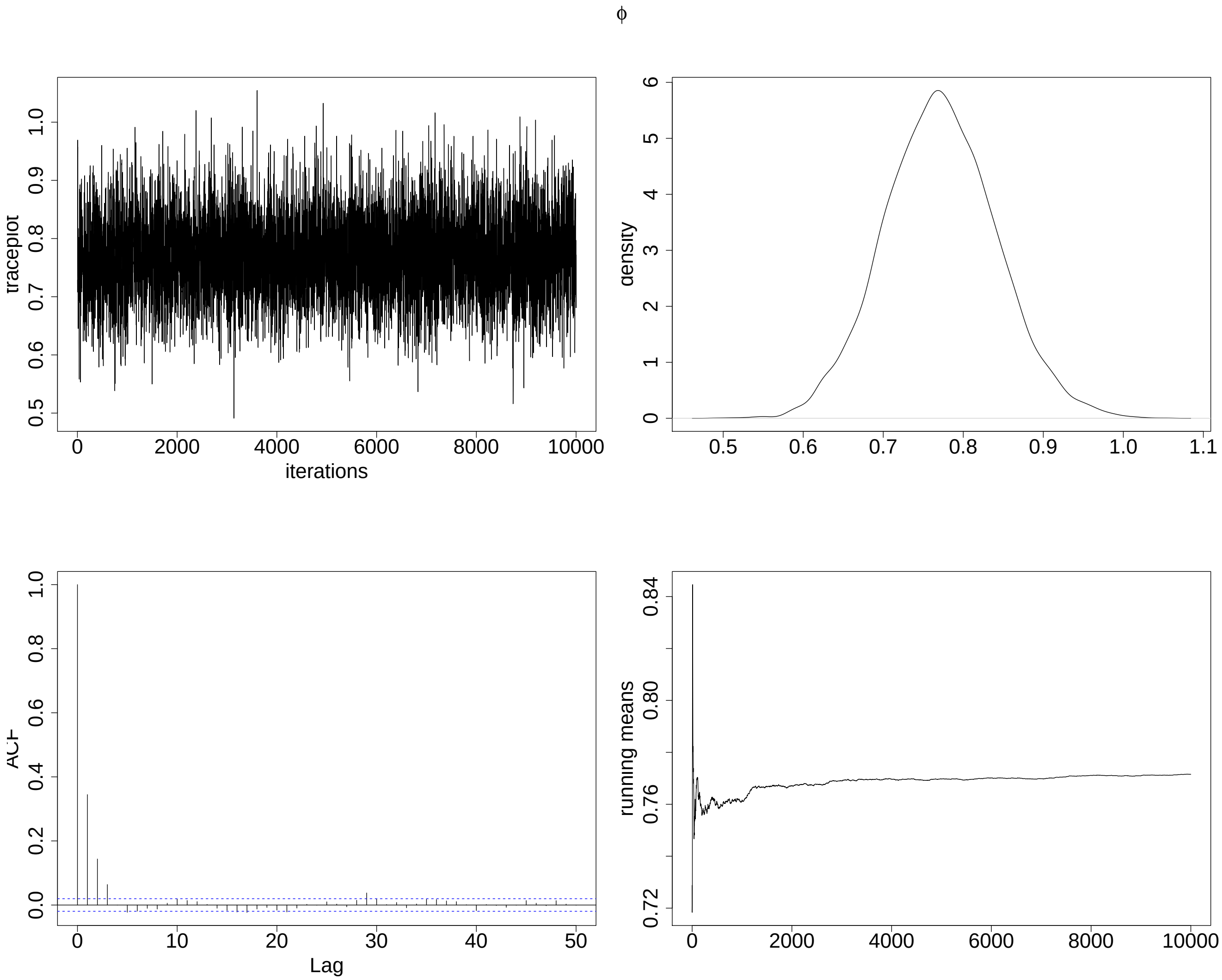
```
## alpha------------------------------------------------------------------------
## mean1   sd1
##  1.71  0.08
```



6.2 phi

```
print.jags.output1(mod, 'phi', t=expression(phi), wb=F)
```

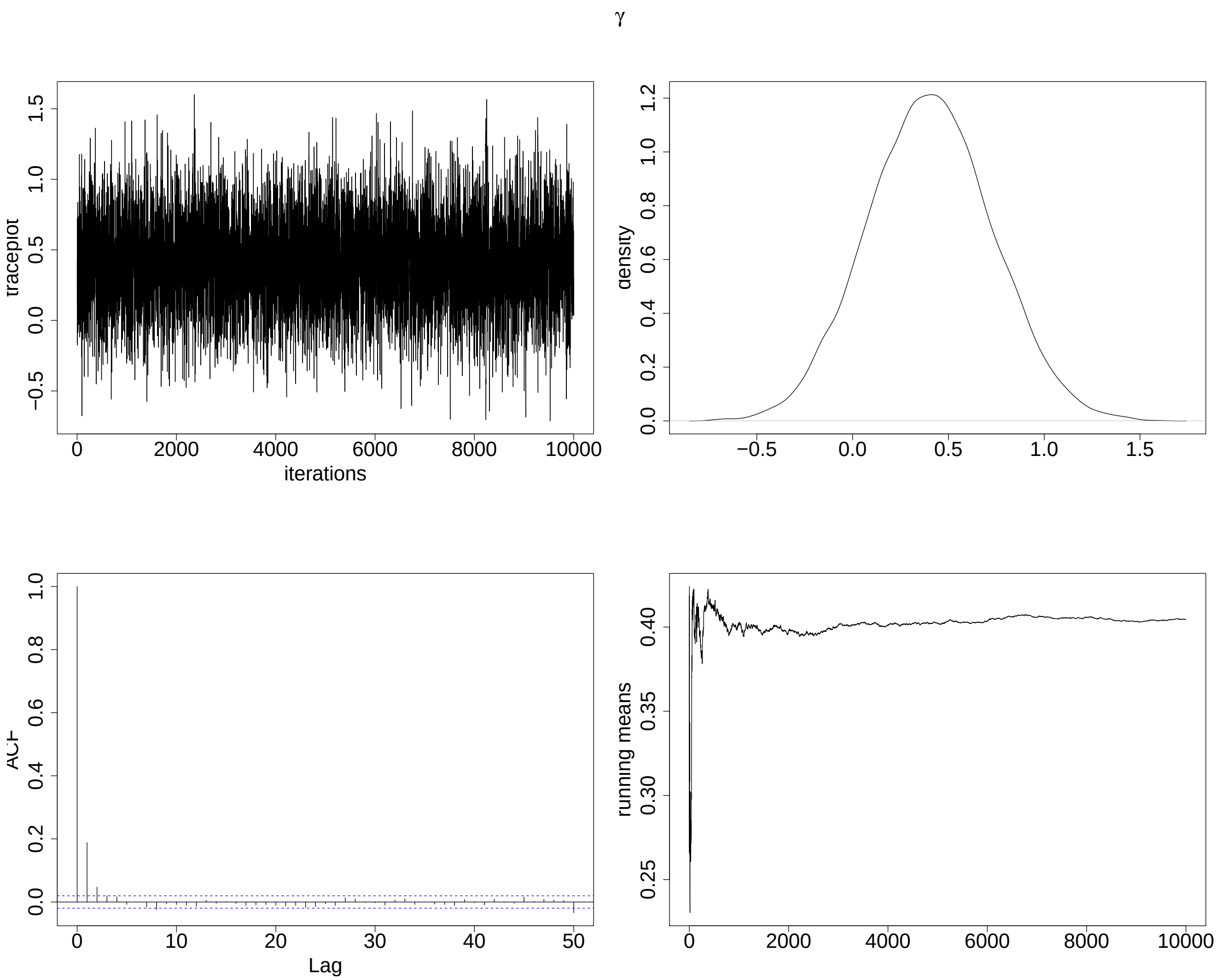
```
## phi-----------------------------------------------------------------
## mean1    sd1
##  0.77   0.07
```



6.3 gamma

```
print.jags.output1(mod, 'gamma', t=expression(gamma), wb=F)
```

```
## gamma-----------------------------------------------------------------
## mean1    sd1
##   0.40   0.33
```



```
cat(paste('DIC', d, mod[[2]]$DIC))
```

```
## DIC -----------------------------------------------------------------------
##  368.457258924328
```

```
cat('model -------------------------------------------------\n')
mod[[2]]
DIC3 = mod[[2]]$DIC
dev3 = mod[[2]]$mean[['deviance']]
```

```
## model -------------------------------------------------
## Inference for Bugs model at "model04.txt", fit using jags,
##  1 chains, each with 10000 iterations (first 0 discarded)
##  n.sims = 10000 iterations saved
##           mean  sd  2.5%   25%   50%   75% 97.5%
## alpha      1.7 0.1   1.6   1.7   1.7   1.8   1.9
## deviance 365.5 2.4 362.8 363.7 364.9 366.6 371.9
## gamma      0.4 0.3  -0.2   0.2   0.4   0.6   1.0
## phi        0.8 0.1   0.6   0.7   0.8   0.8   0.9
##
## DIC info (using the rule, pD = var(deviance)/2)
## pD = 3.0 and DIC = 368.5
## DIC is an estimate of expected predictive error (lower deviance is better).
```

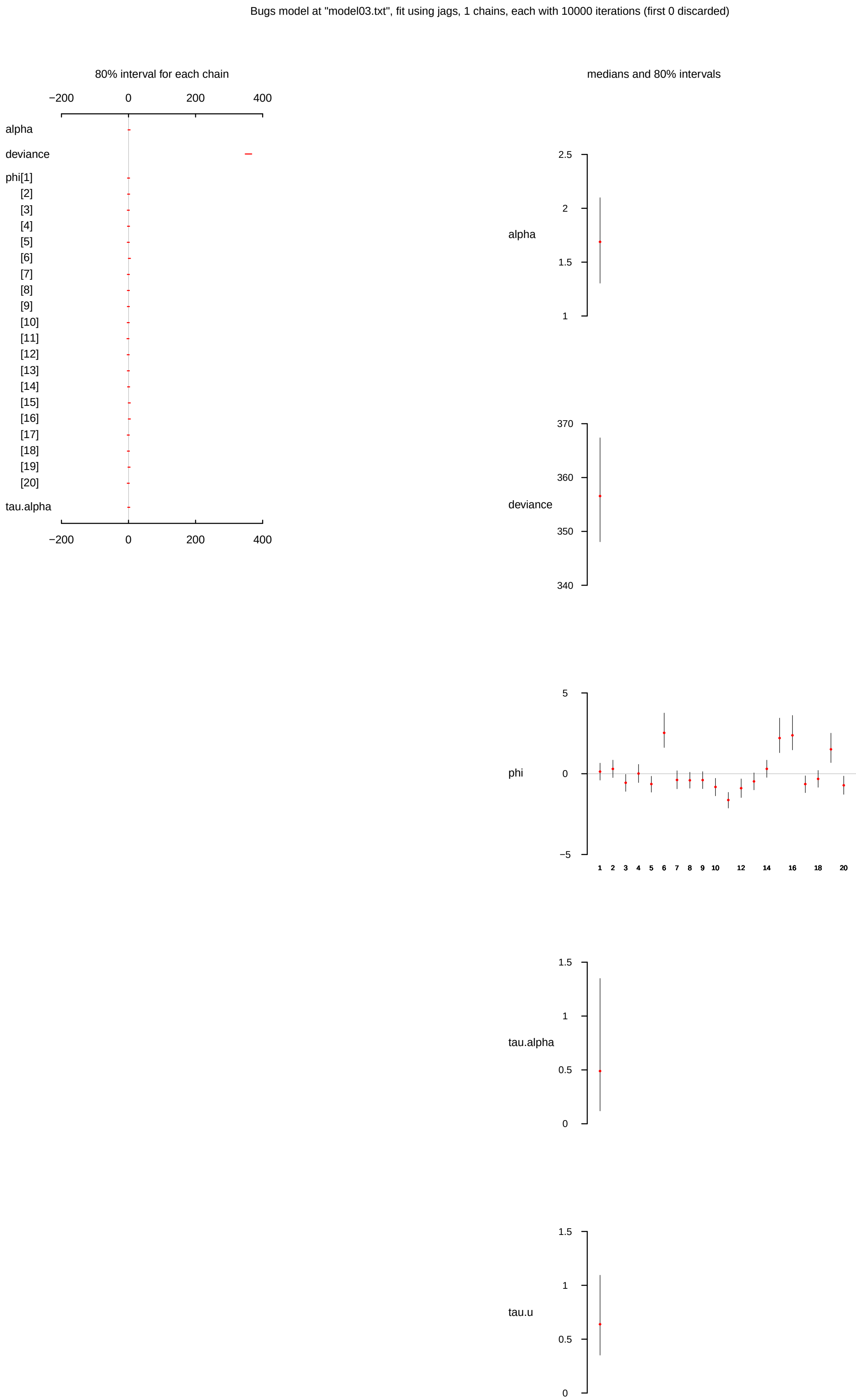
7. Model: no books random effect

MODEL

```
model {
     for (i in 1:Nusers) {
          for (j in 1:Nbooks) {
               ratings[i,j]~dbin(theta[i,j],9)
               logit.theta[i,j] <- alpha + phi[i]
               theta[i,j] <- exp(logit.theta[i,j])/(1+exp(logit.theta[i,j]))
          }
          phi[i]~dnorm(0, tau.u)
     }
     alpha~dnorm(0, tau.alpha)
     tau.u~dgamma(a.u, b.u)
     tau.alpha~dgamma(a.alpha, b.alpha)
}
```

```
data = list(Nusers = NUsers, Nbooks = NBooks, ratings = ratings, a.u = 1,
            b.u = 1, a.alpha = 1, b.alpha = 1)

inits = list(alpha = 0,
             phi = rep(0,NUsers),
             tau.u = 0.1, tau.alpha = 0.1)
mod = jags(data = data,
           inits = list(inits),
           n.chain = 1,
           parameters.to.save=c("tau.u", "tau.alpha",
                                "phi","alpha"),
           model.file="model03.txt",
           n.iter = 10000,
           n.burnin = 0,
           n.thin = 1)
plot(mod)
```

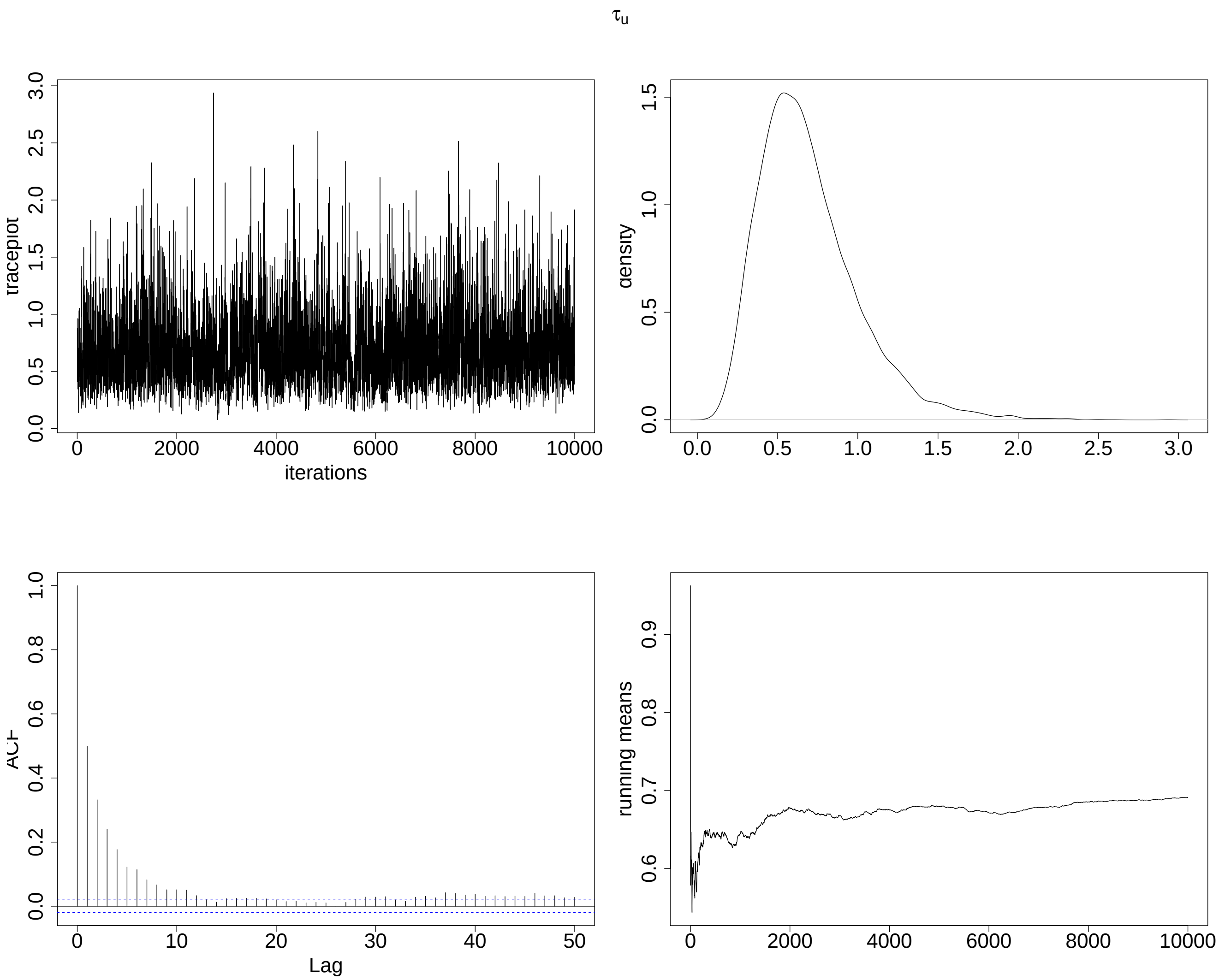
```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 158
##    Unobserved stochastic nodes: 85
##    Total graph size: 1230
##
## Initializing model
```


Bugs model at "model03.txt", fit using jags, 1 chains, each with 10000 iterations (first 0 discarded)
80% interval for each chain
−200
0
200
400
alpha
deviance
phi[1]
[2]
[3]
[4]
[5]
[6]
[7]
[8]
[9]
[10]
[11]
[12]
[13]
[14]
[15]
[16]
[17]
[18]
[19]
[20]
tau.alpha
medians and 80% intervals
alpha
2.5
2
1.5
1
deviance
370
360
350
340
phi
5
0
−5
1 2 3 4 5 6 7 8 9 10 12 14 16 18 20
tau.alpha
1.5
1
0.5
0
tau.u
1.5
1
0.5
0

7.1 tau.u

```r
print.jags.output1(mod, 'tau.u',
                   t=expression(tau[u]), wb=F)
```

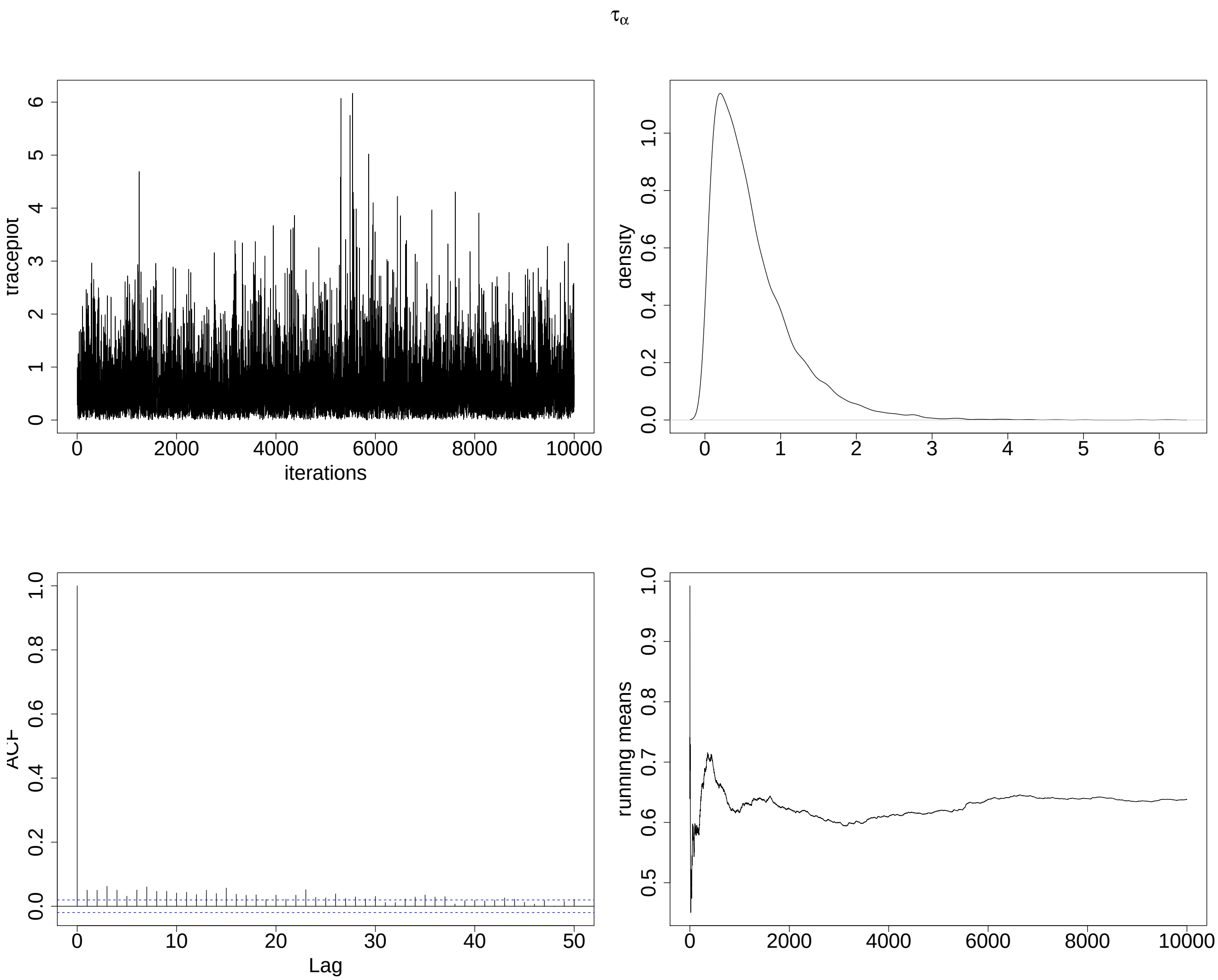
```
## tau.u-----------------------------------------------------------------------
## mean1    sd1
##   0.69   0.31
```



7.2 tau.alpha

```r
print.jags.output1(mod, 'tau.alpha',
                   t=expression(tau[alpha]), wb=F)
```

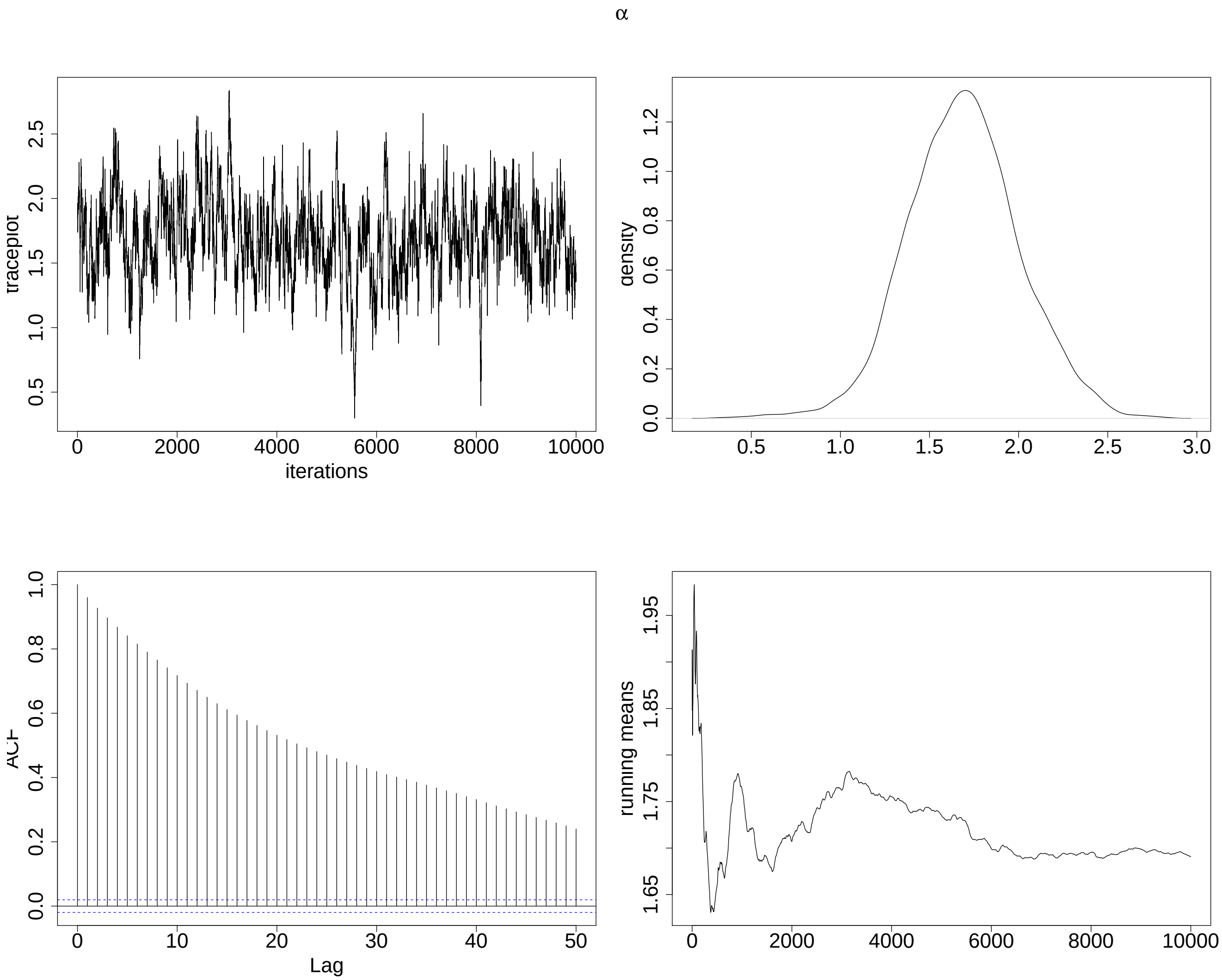
```
## tau.alpha-------------------------------------------------------------------
## mean1   sd1
##  0.64  0.56
```



7.3 alpha

```
print.jags.output1(mod, 'alpha', t=expression(alpha), wb=F)
```

```
## alpha-------------------------------------------------------------------------
## mean1   sd1
##  1.69  0.31
```



```r
cat(paste('phi',d))
c('mean'=mean(mod[[2]]$mean$phi),'sd'=sd(mod[[2]]$mean$phi))
```

```
## phi ----------------------------------------------------------------------
##       mean         sd
## 0.09345992 1.20102569
```

```r
cat(paste('DIC', d, mod[[2]]$DIC))
```

```
## DIC ----------------------------------------------------------------------
##  385.952700304907
```

```r
cat('model ------------------------------------------------\n')
mod[[2]]
DIC4 = mod[[2]]$DIC
dev4 = mod[[2]]$mean[['deviance']]
```

```
## model ------------------------------------------------
## Inference for Bugs model at "model03.txt", fit using jags,
##  1 chains, each with 10000 iterations (first 0 discarded)
##  n.sims = 10000 iterations saved
##             mean  sd  2.5%   25%   50%   75% 97.5%
## alpha        1.7 0.3   1.1   1.5   1.7   1.9   2.3
## deviance   357.3 7.6 344.5 351.8 356.6 361.9 373.9
## phi[1]       0.1 0.4  -0.7  -0.1   0.1   0.4   0.9
## phi[2]       0.3 0.4  -0.5   0.0   0.3   0.6   1.2
## phi[3]      -0.6 0.4  -1.4  -0.8  -0.6  -0.3   0.3
## phi[4]       0.0 0.4  -0.8  -0.3   0.0   0.3   0.9
## phi[5]      -0.6 0.4  -1.4  -0.9  -0.6  -0.4   0.1
## phi[6]       2.6 0.9   1.2   2.0   2.5   3.1   4.5
## phi[7]      -0.4 0.4  -1.3  -0.7  -0.4  -0.1   0.5
## phi[8]      -0.4 0.4  -1.2  -0.7  -0.4  -0.1   0.4
## phi[9]      -0.4 0.4  -1.2  -0.7  -0.4  -0.1   0.4
## phi[10]     -0.8 0.4  -1.7  -1.1  -0.8  -0.5   0.0
## phi[11]     -1.6 0.4  -2.4  -1.9  -1.6  -1.4  -0.9
## phi[12]     -0.9 0.5  -1.8  -1.2  -0.9  -0.6   0.0
## phi[13]     -0.5 0.4  -1.3  -0.7  -0.5  -0.2   0.3
## phi[14]      0.3 0.4  -0.5   0.0   0.3   0.6   1.1
## phi[15]      2.3 0.9   0.9   1.7   2.2   2.8   4.3
## phi[16]      2.5 0.9   1.0   1.9   2.4   3.0   4.4
## phi[17]     -0.6 0.4  -1.5  -0.9  -0.6  -0.4   0.2
## phi[18]     -0.3 0.4  -1.1  -0.6  -0.3   0.0   0.5
## phi[19]      1.6 0.7   0.3   1.1   1.5   2.0   3.2
## phi[20]     -0.7 0.4  -1.6  -1.0  -0.7  -0.4   0.2
## tau.alpha    0.6 0.6   0.0   0.2   0.5   0.9   2.1
## tau.u        0.7 0.3   0.3   0.5   0.6   0.8   1.4
##
## DIC info (using the rule, pD = var(deviance)/2)
## pD = 28.7 and DIC = 386.0
## DIC is an estimate of expected predictive error (lower deviance is better).
```

8. Deviance mean and DIC

	deviance mean	DIC
MODEL1	356.55	386.43
MODEL2	359.33	395.52
MODEL1(wo)	356.82	377.04
MODEL2(wo)	359.37	385.66
MODEL3	365.49	368.46
MODEL4	357.25	385.95

The deviance mean of MODEL4 is slightly bigger than the MODEL1's one, since we have a DIC slightly better due to the reduction of the number of parameters (35 for MODEL1 against 23 for MODEL5). MODEL3 has the worst deviance mean but the best DIC (in this case parameters are just α, ϕ and γ). MODEL1(wo)'s DIC is computed with *pD=Dbar-Dhat*, with *pD= var(deviance)/2* instead we would obtain 356.8+(10.2^2/2)=408.82. The same for MODEL2(wo)'s DIC which is equal to 393.84.