Premier league 2017-2018

Introducing the Vanilla Model

We are interested in modelling the English Premier League season 2017-2018 and more specifically the goals scored by each team in a round robin championship. English championship is one of the hardest championships to model, since it's the most competitive one. The competitiveness of the league can be illustrated in the picture below, where it's conspicuous that anybody can beat anybody!

The response variable here is the number of scored goals by each team within a time interval (the length of each championship game), thus we have count data. Therefore, a reasonable choice would be the use of a

Poisson regression model. We will focus on implementing a double Poisson model, which consists of one target variable for the scored goals of the home team and another one for the scored goals of the away team. These models are often called Poisson log linear models due to the canonical link, which is the log of the mean. Our primary focus will be on the Vanilla model, a simple and basic log linear model originally introduced by Maher (1982), also used by other authors, such as Lee (1997) and Karlis and Ntzoufras (2000).

The model is given by

$\mathrm{Y_{ij}}$ ~ Poisson(λ_{ik}) for j=1,2

log(λ_{i1}) = mu + home + $\mathrm{att}_{\mathrm{HT_i}}$ + $\mathrm{def}_{\mathrm{AT_i}}$,

log(λ_{i2}) = mu + $\mathrm{att}_{\mathrm{AT_i}}$ + $\mathrm{def}_{\mathrm{HT_i}}$, for i = 1,2,…,n

Notation

- n: the number of games.
- mu: constant parameter, which denotes an overall level of log expected goals scored in away games.
- home: Encapsulates the home effect denoted by the difference between the log expected goals scored when two teams of equal strength compete with each other. Binary for home and away teams (1 for home teams, zero otherwise).
- $\mathrm{HT_i}$: Home team i
- $\mathrm{AT_i}$: Away team i
- $\mathrm{att_k}$ stands for the attacking abilities of k team and $\mathrm{def_k}$ stands for the defensive abilities of k team for k = 1,2,...,K

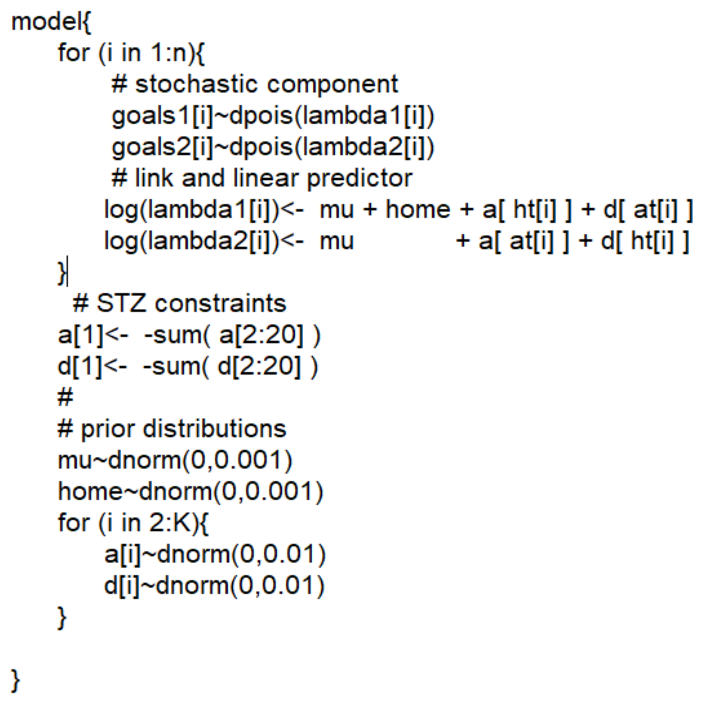
For the attacking and the defensive parameters, we use the sum to zero constraints in order to make the model identifiable and compare the ability of each team with an overall level of attacking and defensive abilities. Hence, we set:

$\sum_{\mathrm{k}=1}^{\mathrm{K}} \mathrm{att_k}$ = 0 and $\sum_{\mathrm{k}=1}^{\mathrm{K}} \mathrm{def_k}$

According to this parametrization all parameters have a straightforward interpretation! Thus:

- Att/def: Attacking/Defensive parameters can be interpreted as deviations of the attacking and respectfully defensive abilities from the average level in the league. Hence, a positive attacking parameter indicates that the team under consideration has an offensive performance that is better than the average level of the teams competing in the league. Likewise, a negative defensive parameter indicates that the team under consideration has a defensive performance that is better than the average level of the teams competing in the league.
- K: The number of teams in the data set under consideration.

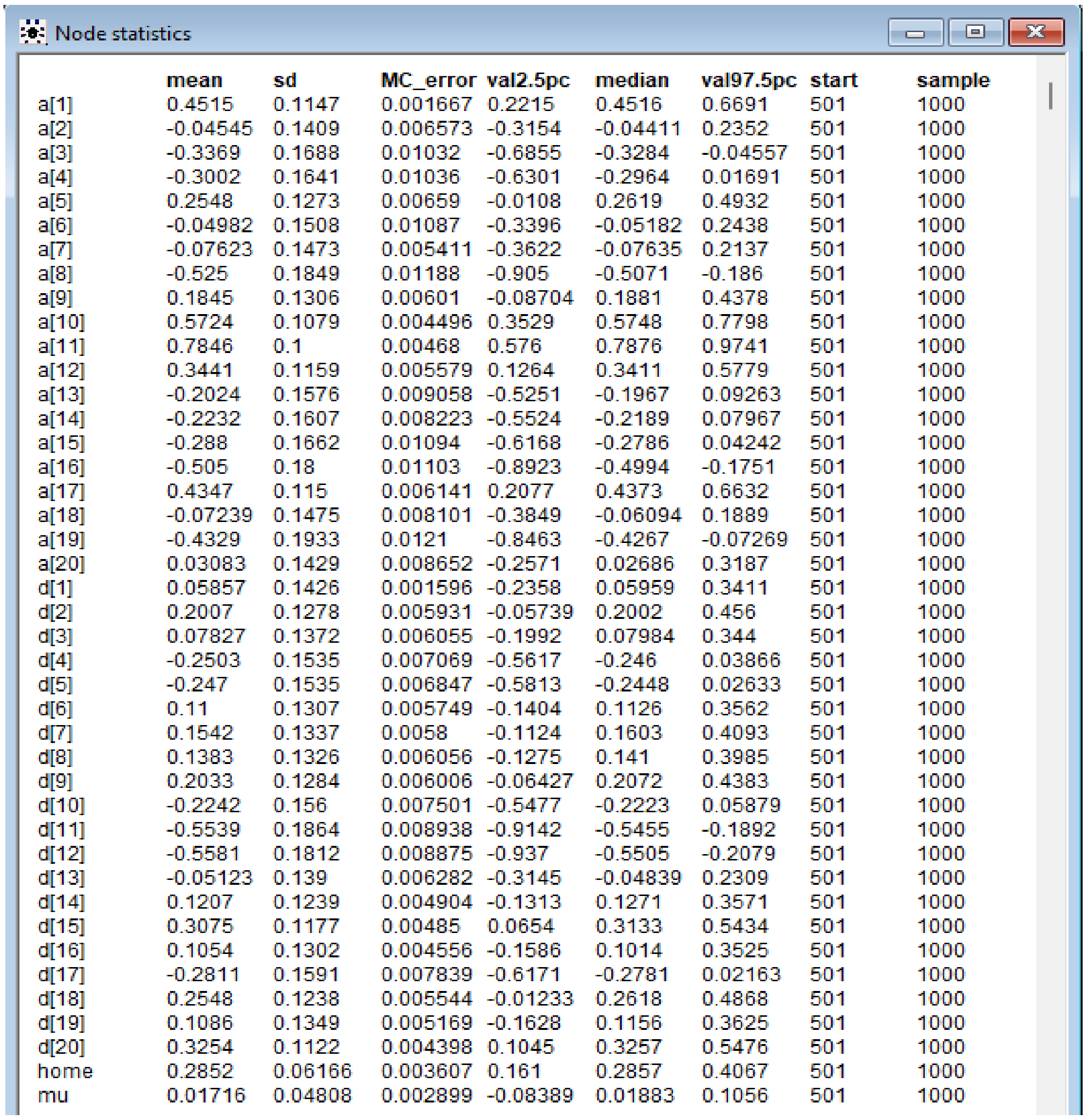
The model structure in OpenBUGS is as follows:



```
model{
    for (i in 1:n){
        # stochastic component
        goals1[i]~dpois(lambda1[i])
        goals2[i]~dpois(lambda2[i])
        # link and linear predictor
       log(lambda1[i])<-  mu + home + a[ ht[i] ] + d[ at[i] ]
       log(lambda2[i])<-  mu            + a[ at[i] ] + d[ ht[i] ]
    }
     # STZ constraints
    a[1]<-  -sum( a[2:20] )
    d[1]<-  -sum( d[2:20] )
    #
    # prior distributions
    mu~dnorm(0,0.001)
    home~dnorm(0,0.001)
    for (i in 2:K){
        a[i]~dnorm(0,0.01)
        d[i]~dnorm(0,0.01)
    }

}
```

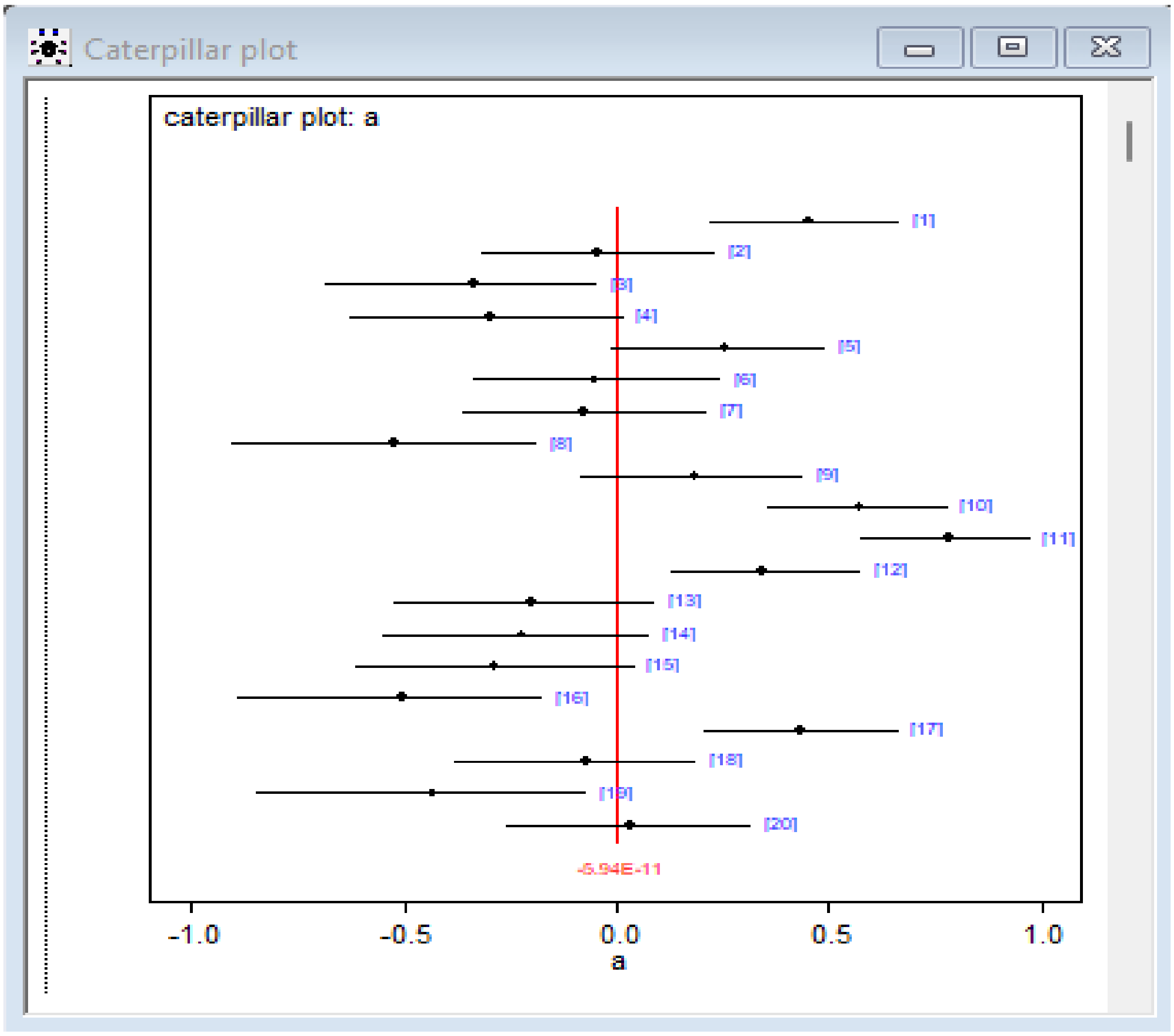
Inference based on the final model

Posterior summaries of the Poisson log-linear model parameters are provided below:

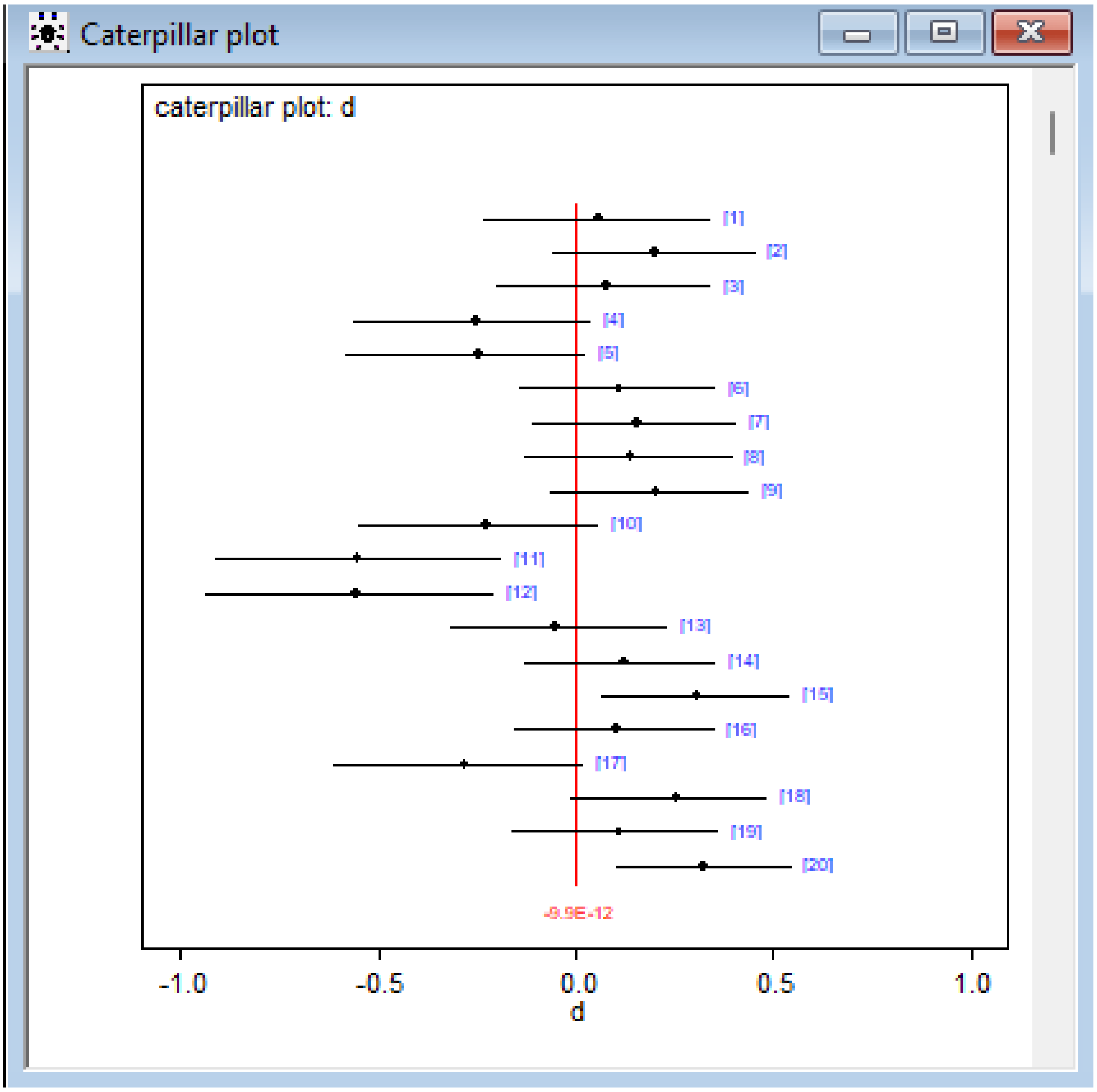


```
         mean      sd       MC_error  val2.5pc  median    val97.5pc start  sample
a[1]     0.4515    0.1147   0.001667  0.2215    0.4516    0.6691    501    1000
a[2]     -0.04545  0.1409   0.006573  -0.3154   -0.04411  0.2352    501    1000
a[3]     -0.3369   0.1688   0.01032   -0.6855   -0.3284   -0.04557  501    1000
a[4]     -0.3002   0.1641   0.01036   -0.6301   -0.2964   0.01691   501    1000
a[5]     0.2548    0.1273   0.00659   -0.0108   0.2619    0.4932    501    1000
a[6]     -0.04982  0.1508   0.01087   -0.3396   -0.05182  0.2438    501    1000
a[7]     -0.07623  0.1473   0.005411  -0.3622   -0.07635  0.2137    501    1000
a[8]     -0.525    0.1849   0.01188   -0.905    -0.5071   -0.186    501    1000
a[9]     0.1845    0.1306   0.00601   -0.08704  0.1881    0.4378    501    1000
a[10]    0.5724    0.1079   0.004496  0.3529    0.5748    0.7798    501    1000
a[11]    0.7846    0.1      0.00468   0.576     0.7876    0.9741    501    1000
a[12]    0.3441    0.1159   0.005579  0.1264    0.3411    0.5779    501    1000
a[13]    -0.2024   0.1576   0.009058  -0.5251   -0.1967   0.09263   501    1000
a[14]    -0.2232   0.1607   0.008223  -0.5524   -0.2189   0.07967   501    1000
a[15]    -0.288    0.1662   0.01094   -0.6168   -0.2786   0.04242   501    1000
a[16]    -0.505    0.18     0.01103   -0.8923   -0.4994   -0.1751   501    1000
a[17]    0.4347    0.115    0.006141  0.2077    0.4373    0.6632    501    1000
a[18]    -0.07239  0.1475   0.008101  -0.3849   -0.06094  0.1889    501    1000
a[19]    -0.4329   0.1933   0.0121    -0.8463   -0.4267   -0.07269  501    1000
a[20]    0.03083   0.1429   0.008652  -0.2571   0.02686   0.3187    501    1000
d[1]     0.05857   0.1426   0.001596  -0.2358   0.05959   0.3411    501    1000
d[2]     0.2007    0.1278   0.005931  -0.05739  0.2002    0.456     501    1000
d[3]     0.07827   0.1372   0.006055  -0.1992   0.07984   0.344     501    1000
d[4]     -0.2503   0.1535   0.007069  -0.5617   -0.246    0.03866   501    1000
d[5]     -0.247    0.1535   0.006847  -0.5813   -0.2448   0.02633   501    1000
d[6]     0.11      0.1307   0.005749  -0.1404   0.1126    0.3562    501    1000
d[7]     0.1542    0.1337   0.0058    -0.1124   0.1603    0.4093    501    1000
d[8]     0.1383    0.1326   0.006056  -0.1275   0.141     0.3985    501    1000
d[9]     0.2033    0.1284   0.006006  -0.06427  0.2072    0.4383    501    1000
d[10]    -0.2242   0.156    0.007501  -0.5477   -0.2223   0.05879   501    1000
d[11]    -0.5539   0.1864   0.008938  -0.9142   -0.5455   -0.1892   501    1000
d[12]    -0.5581   0.1812   0.008875  -0.937    -0.5505   -0.2079   501    1000
d[13]    -0.05123  0.139    0.006282  -0.3145   -0.04839  0.2309    501    1000
d[14]    0.1207    0.1239   0.004904  -0.1313   0.1271    0.3571    501    1000
d[15]    0.3075    0.1177   0.00485   0.0654    0.3133    0.5434    501    1000
d[16]    0.1054    0.1302   0.004556  -0.1586   0.1014    0.3525    501    1000
d[17]    -0.2811   0.1591   0.007839  -0.6171   -0.2781   0.02163   501    1000
d[18]    0.2548    0.1238   0.005544  -0.01233  0.2618    0.4868    501    1000
d[19]    0.1086    0.1349   0.005169  -0.1628   0.1156    0.3625    501    1000
d[20]    0.3254    0.1122   0.004398  0.1045    0.3257    0.5476    501    1000
home     0.2852    0.06166  0.003607  0.161     0.2857    0.4067    501    1000
mu       0.01716   0.04808  0.002899  -0.08389  0.01883   0.1056    501    1000
```

We can observe that node 11 which corresponds to the season's 2017-2018 champions, Manchester City, has by far the best (i.e. the highest) attacking abilities, while their defensive abilities are tied first (i.e. the lowest) along with node 12, which corresponds to Manchester United, that was second-best in the final rankings. In a game where two average teams are competing each other, then the expected number of goals are equal to 1.35 for the home team and 1.01 for the away team, resulting in an increase of 33% of each team scoring mean when playing in its home field! Let us have a better look on the attacking of each team by using caterpillar plots (boxplots as well as density strips can be found at the appendix!):



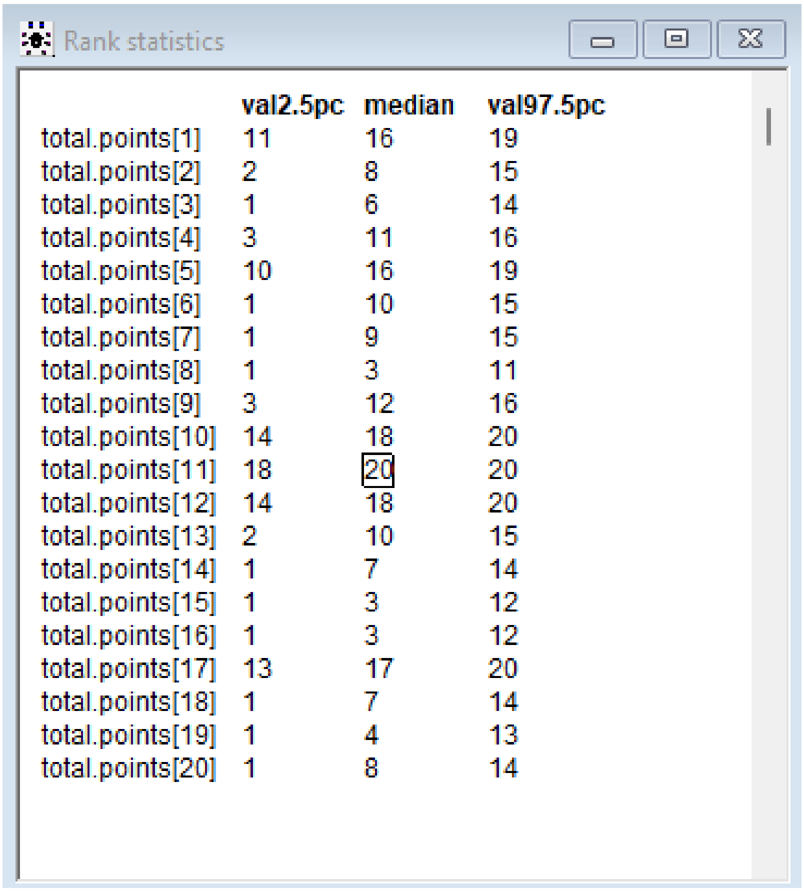
The figure above indicates the 95% posterior interval for the team attacking abilities. Teams that cross 0 means that their attacking abilities are close to the average attacking abilities of all teams of the championship. We observe that nodes 11,10,1,17,12 do not cross 0 and are positive, thus we can infer that their attacking abilities are way better than the average attacking abilities of all teams of the championship. Those nodes correspond to the following teams respectively: Manchester City (having scored 106 goals being the best attack of the league by far), Liverpool (having scored 84 goals being the second best on the goals scored), Arsenal (having scored 74 goals being the third best on the goals scored), Tottenham (having scored 74 goals being tied with Arsenal as the third best on the goals scored) and Manchester United (having scored 68 goals being the fourth best on the goals scored). From the teams mentioned above, Manchester City were the undisputed champions of the season 2017-2018, Manchester United finished second, Tottenham finished third, Liverpool finished fourth while Arsenal finished sixth due to their worse defensive abilities. Nodes 8,16,19,3 correspond to the following teams respectively: Huddersfield (having scored 28, the least number of goals), Swansea (having scored 28 tied first worst attacking ability), West Brom (having scored 31 goals second worst attacking ability), Brighton (having scored 34 goals being the team with the third worst attacking ability). From the teams mentioned above at the 2017-2018 season, Swansea and West Brom were relegated while Brighton and Huddersfield made it due to their defensive abilities. Now, let us have a look at the caterpillar plots (boxplots and density strips can be found at the appendix!) of the defensive abilities of the teams:



The figure above indicates the 95% posterior intervals for the team defensive parameters. Teams that cross 0 mean that their defensive abilities are quite similar to the average defensive abilities of the teams that are included in the round robin championship. We observe that nodes 11 and 12 do not cross 0 and are negative (hence they are much better in comparison to the average defensive abilities of the teams included in the premier league's 2017-2018 championship). These nodes refer to Manchester City (having the best defensive abilities of the season, conceding 27 goals) and Manchester United (having the second-best defensive abilities of the season, conceding 28 goals) respectively. As regards nodes 15 and 20 that are positive and away from zero (since they do not cross it) they correspond to Stoke City and West Ham respectively. Stoke City was relegated, and a major factor was that they had the worst defending abilities in the league since they conceded 68 goals. The same number of goals was conceded by West Ham as well, but they weren't relegated due to their attacking abilities!

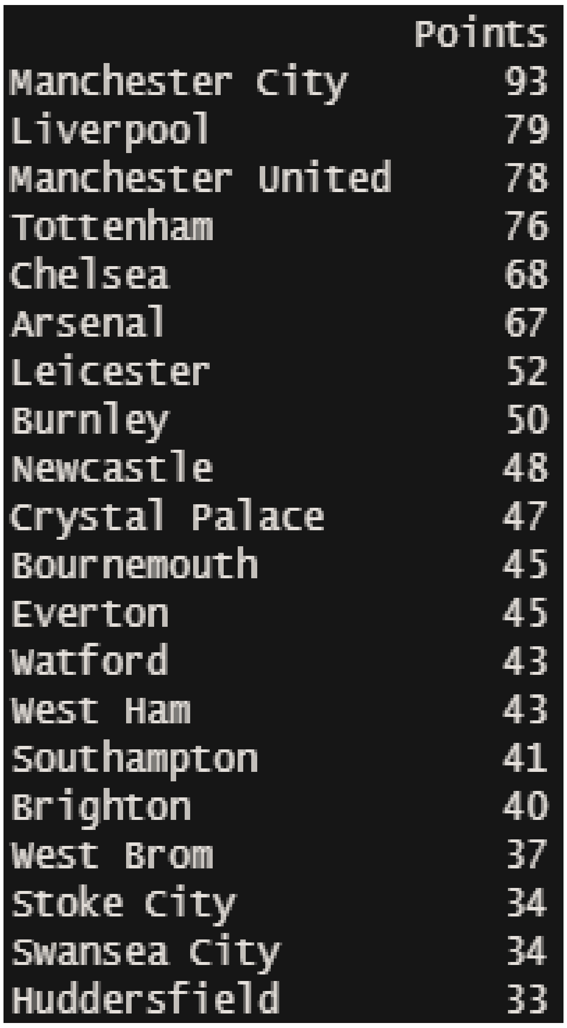
Regeneration of the full league using the Vanilla model

Interest also lies in reconstructing the league using the predictive distribution. Such practice is useful to evaluate whether the final observed ranking was plausible under the fitted model. It can be interpreted as the uncertainty involved in the final ranking if the league is repeated and the model is true (in sample prediction); see Karlis and Ntzoufras (2008) for more information. In order to reconstruct the full table in OpenBUGS, we need to replicate the full scores in a tabular KxK format and then calculate the number of points for each team. The replicated league is calculated using the syntax that can be found in the appendix. In this syntax, points1 and points2 calculate the number of points in each game for the home and the away teams, respectively (arranged in i=1,2,…,K rows and j=1,2,…,K columns). In the total points we calculate the total number of points for each team i earned in home games (sum of i row of node <<points1>>) and in away games (sum of j column of node <<points2>>). Diagonal elements points1[i,i] and points2[i,i] are removed since they refer to each team playing against itself. Posterior summaries of the total predicted earned points are obtained as usual (by using the sample monitor tool), while posterior summaries of the ranks are obtained using the rank monitor tool. The results are summarized after 5000 iterations kept (1000 iterations burn-in period were removed) and can be depicted below:

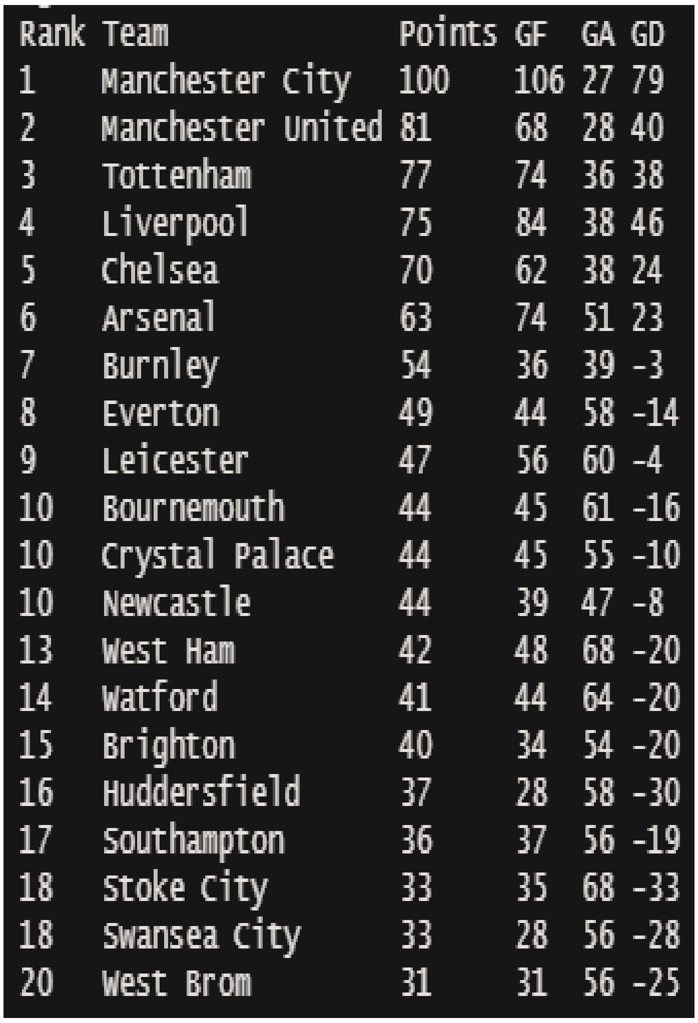


```
                  val2.5pc  median  val97.5pc
total.points[1]   11        16      19
total.points[2]   2         8       15
total.points[3]   1         6       14
total.points[4]   3         11      16
total.points[5]   10        16      19
total.points[6]   1         10      15
total.points[7]   1         9       15
total.points[8]   1         3       11
total.points[9]   3         12      16
total.points[10]  14        18      20
total.points[11]  18        20      20
total.points[12]  14        18      20
total.points[13]  2         10      15
total.points[14]  1         7       14
total.points[15]  1         3       12
total.points[16]  1         3       12
total.points[17]  13        17      20
total.points[18]  1         7       14
total.points[19]  1         4       13
total.points[20]  1         8       14
```

Ranks refer to the total number of points in ascending order, hence 20 refers to the team with the highest number of collected points, in our case it's node 11, which corresponds to Manchester City (i.e. the champions!). On the other hand, 1 refers to the team with the lowest number of collected points (i.e. the worst team in the league). In our case the nodes with the least points collected are 15,16,8 which correspond to the teams (Stoke, Swansea and Huddersfield) that were relegated! The reproduced league (number of points collected by each team) as well as the actual league can be depicted as follows (left picture corresponds to the reproduced league while right picture refers to the actual league):



```
                       Points
Manchester City            93
Liverpool                  79
Manchester United          78
Tottenham                  76
Chelsea                    68
Arsenal                    67
Leicester                  52
Burnley                    50
Newcastle                  48
Crystal Palace             47
Bournemouth                45
Everton                    45
Watford                    43
West Ham                   43
Southampton                41
Brighton                   40
West Brom                  37
Stoke City                 34
Swansea City               34
Huddersfield               33
```



```
Rank Team              Points GF  GA GD
1    Manchester City   100    106 27 79
2    Manchester United 81     68  28 40
3    Tottenham         77     74  36 38
4    Liverpool         75     84  38 46
5    Chelsea           70     62  38 24
6    Arsenal           63     74  51 23
7    Burnley           54     36  39 -3
8    Everton           49     44  58 -14
9    Leicester         47     56  60 -4
10   Bournemouth       44     45  61 -16
10   Crystal Palace    44     45  55 -10
10   Newcastle         44     39  47 -8
13   West Ham          42     48  68 -20
14   Watford           41     44  64 -20
15   Brighton          40     34  54 -20
16   Huddersfield      37     28  58 -30
17   Southampton       36     37  56 -19
18   Stoke City        33     35  68 -33
18   Swansea City      33     28  56 -28
20   West Brom         31     31  56 -25
```

We can infer that the Vanilla model regenerated right 8/20 of the teams in the league, hence indicating a predictive ability about 40% as regards the 2017-2018 Premier league. Moreover, there's an example of prediction of the outcomes as well as the probabilities of each outcome regarding the last 2 matches of the Premier league 2017-2018, in the appendix (plus a prediction of the last 10 matches using the GLM approach)!

Note: In the appendix there's the regeneration of the full league using the Negative Binomial model (preferred versus the Vanilla model) as well.

Concluding remarks

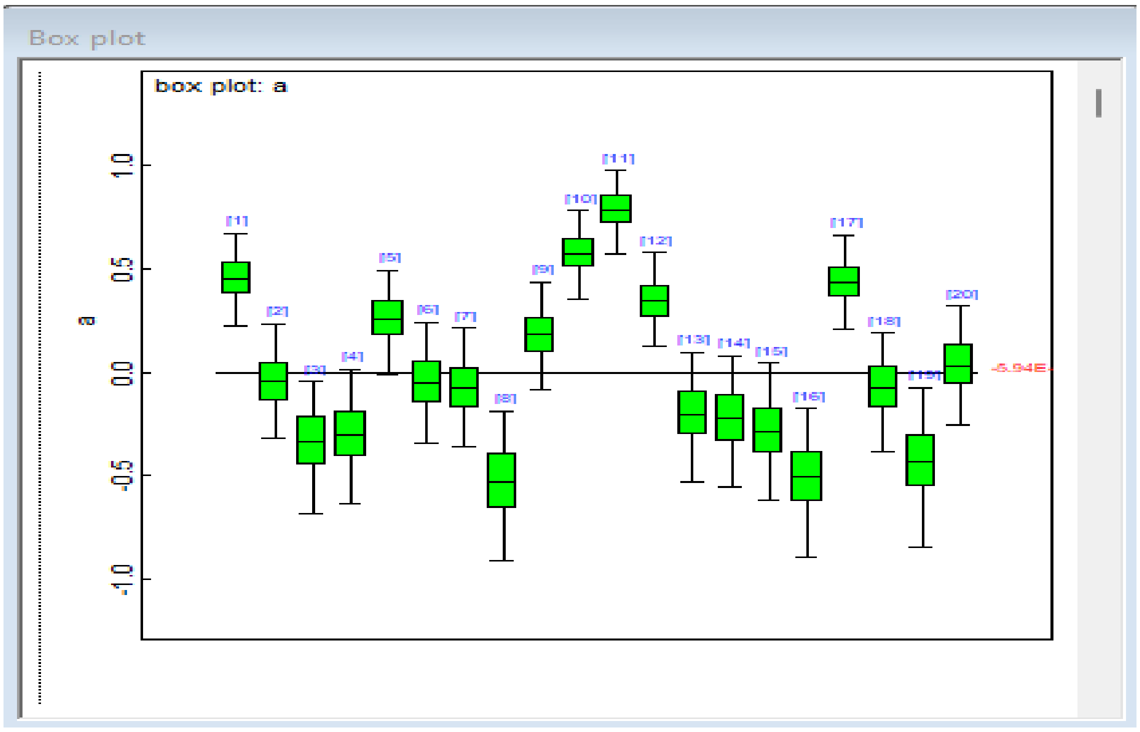
The Vanilla model performs well overall considering its simplicity, plus it can be quite handy as it can be used to compare any model we try to fit. If our fitted model (with added covariates) outperforms the Vanilla model, that's an indication that we are on the right path, while we should check whether our fitted model is parsimonious (the Vanilla model is parsimonious, due to its few covariates!). Moreover, the football assumptions should be taken seriously into consideration since if they are violated there are a few possible remedies. For instance, in most leagues there's observed an excess of draws. Usually, models for goals do not predict very well the draws. In practice, we have an excess of draws in comparison to the predicted one, especially for the 0-0 and 1-1 draws. A possible remedy is to draw an inflation component such as the models proposed by:

- Dixon and Coles (1997) who added extra probabilities in these scores.
- Karlis and Ntzoufras (2003) who implemented a diagonal inflated bivariate Poisson model
- Karlis and Ntzoufras (2008) who implemented the zero inflated model for the goal differences.

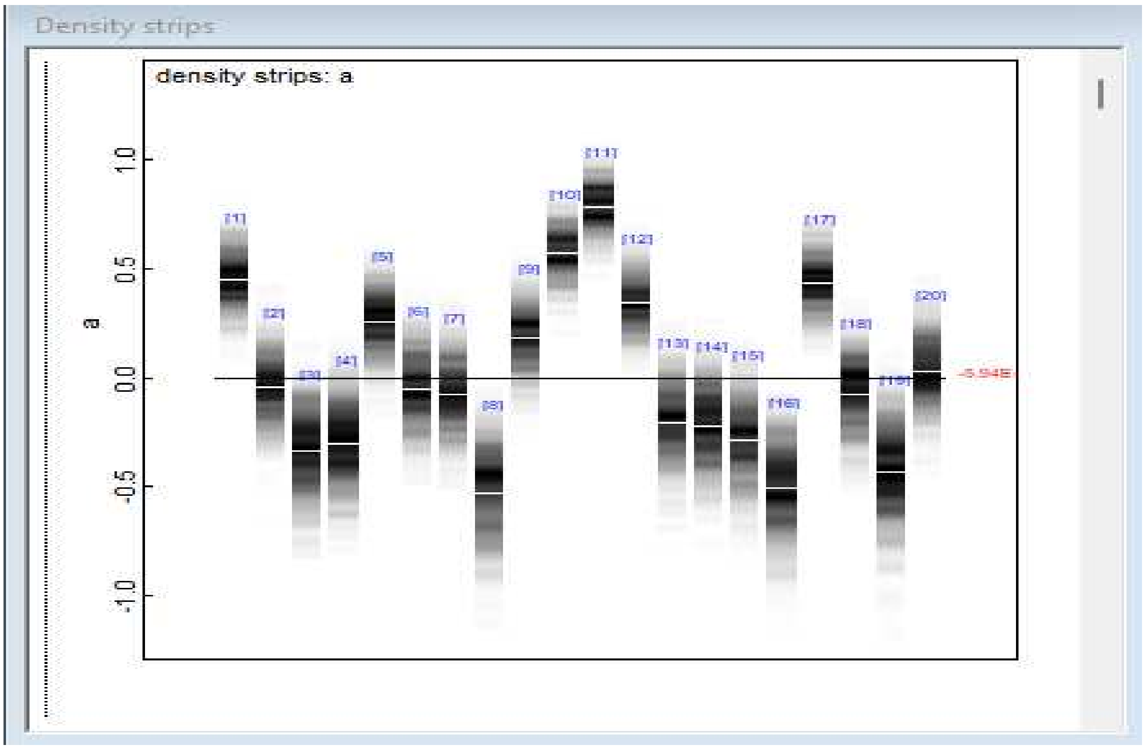
Models using covariates can be further elaborated. There are different covariates for Team/Player performance or for prediction. A method for selecting the best covariates is LASSO. A different approach could be to try fitting the dynamic abilities of the teams. This assumes that abilities are not constant across the season. Usually an AR(1) structure is used (hierarchical model). This is the modern trend in literature, plus it's more realistic since teams have different behaviors over specific time intervals (i.e. Manchester City is a completely different team after it was bought by the Saudi Arabians). On the other hand, it is naturally more complicated and harder to fit! Last but not least, understanding the characteristics of football is of paramount importance in order to construct/fit a good model. The choice for the final model would be the Negative Binomial one, due to the lower DIC that was produced (for more information and predictions made using the NB model check the appendix!).

Appendix

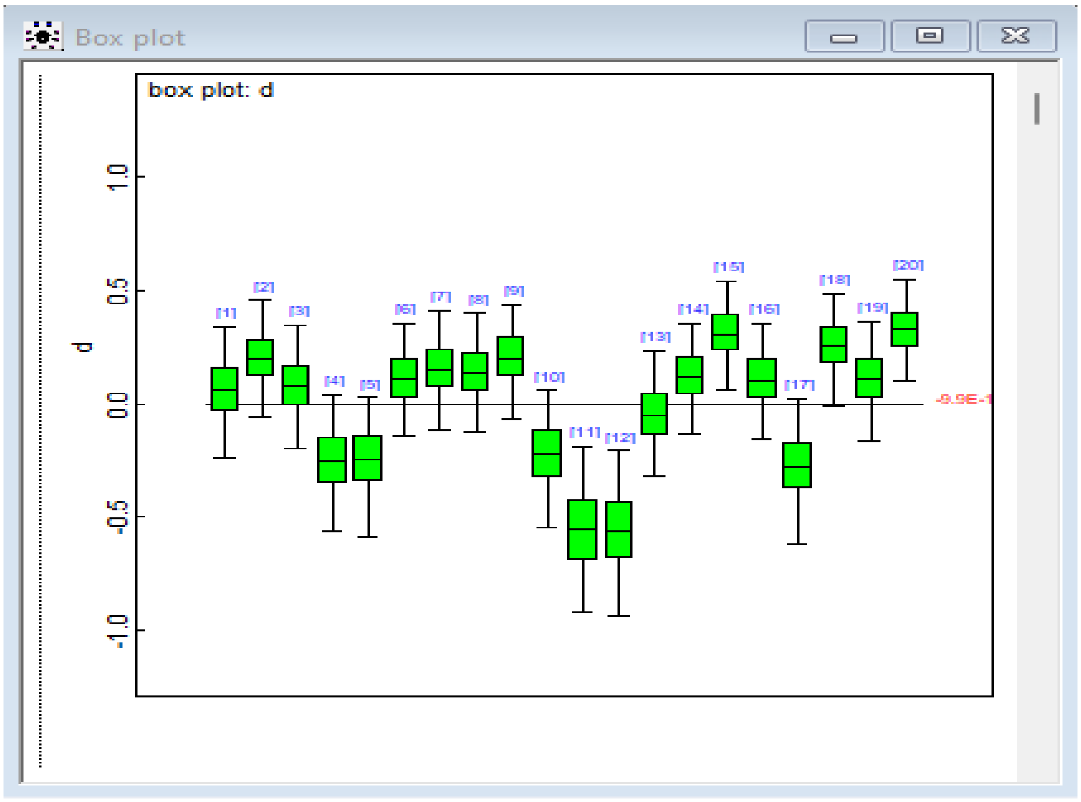
Boxplots of the attacking abilities of the Premier league 2017-2018 teams



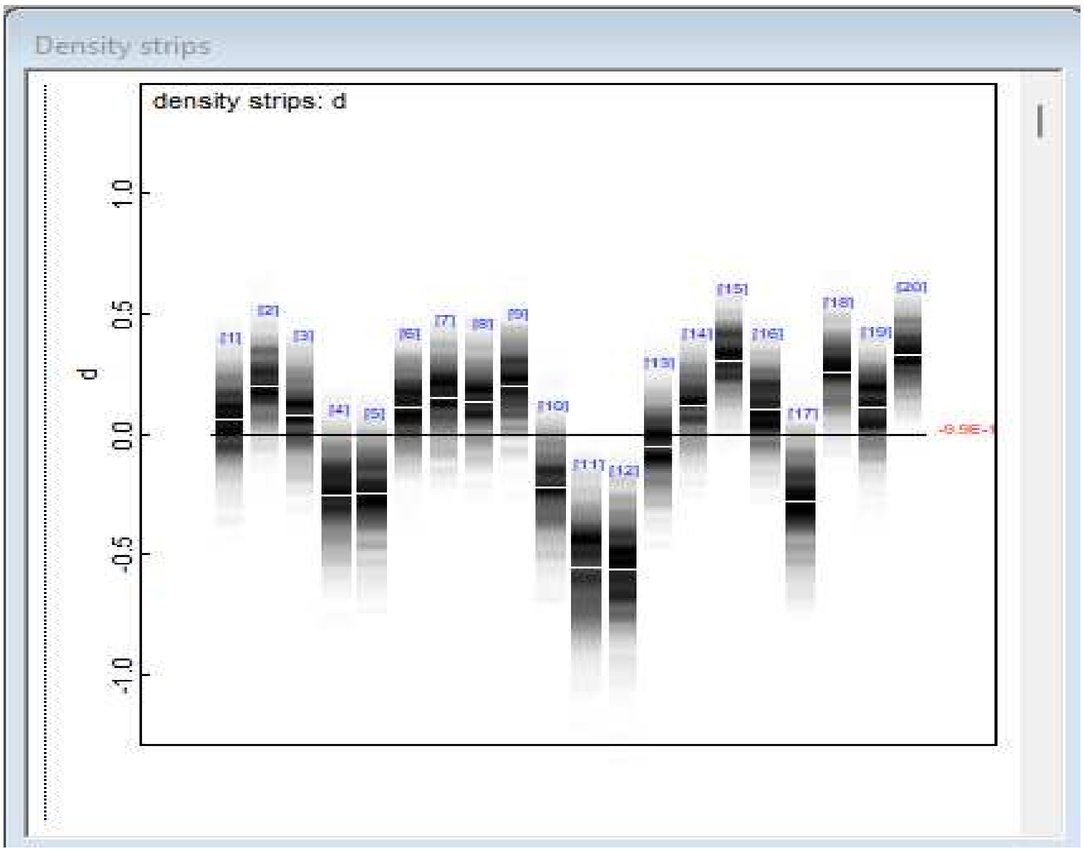
Density strips of the attacking abilities of the Premier league 2017-2018 teams



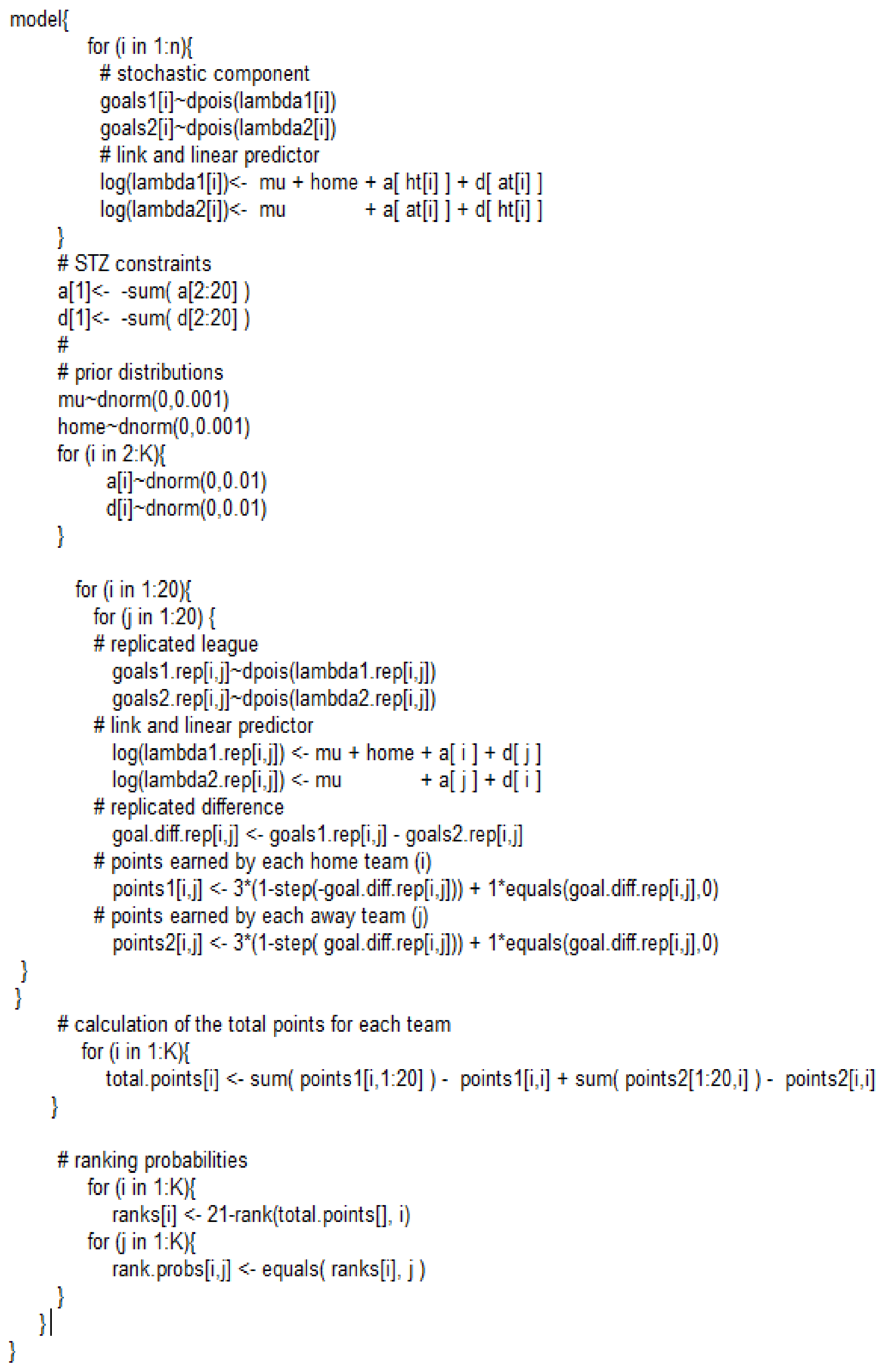
Boxplots of the defensive abilities of the Premier league 2017-2018 teams



Density strips of the defensive abilities of the Premier league 2017-2018 teams

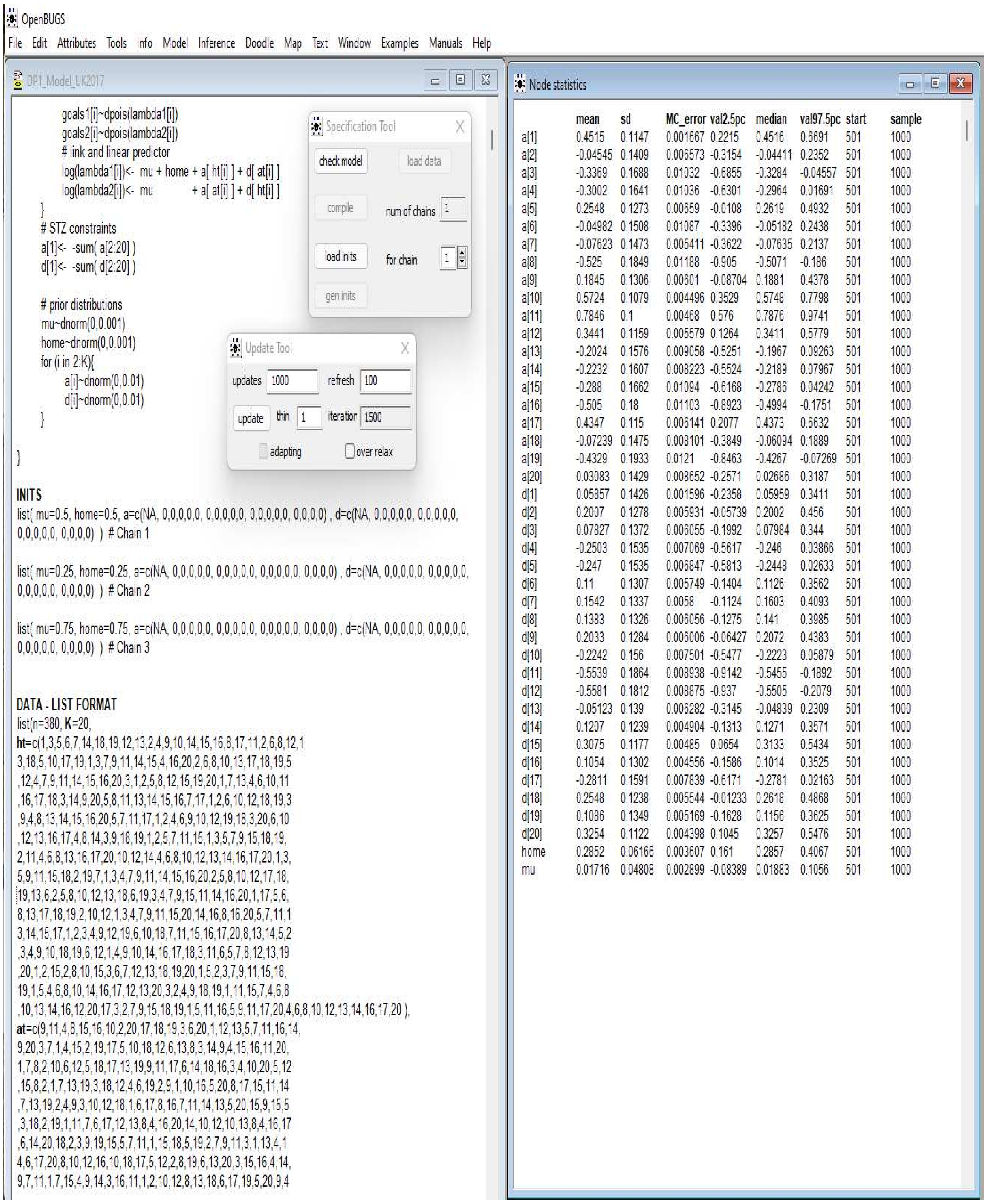


Replicated league syntax



```
model{
            for (i in 1:n){
              # stochastic component
              goals1[i]~dpois(lambda1[i])
              goals2[i]~dpois(lambda2[i])
              # link and linear predictor
              log(lambda1[i])<-  mu + home + a[ ht[i] ] + d[ at[i] ]
              log(lambda2[i])<-  mu             + a[ at[i] ] + d[ ht[i] ]
       }
       # STZ constraints
       a[1]<-  -sum( a[2:20] )
       d[1]<-  -sum( d[2:20] )
       #
       # prior distributions
       mu~dnorm(0,0.001)
       home~dnorm(0,0.001)
       for (i in 2:K){
              a[i]~dnorm(0,0.01)
              d[i]~dnorm(0,0.01)
       }

          for (i in 1:20){
            for (j in 1:20) {
            # replicated league
               goals1.rep[i,j]~dpois(lambda1.rep[i,j])
               goals2.rep[i,j]~dpois(lambda2.rep[i,j])
            # link and linear predictor
               log(lambda1.rep[i,j]) <- mu + home + a[ i ] + d[ j ]
               log(lambda2.rep[i,j]) <- mu             + a[ j ] + d[ i ]
            # replicated difference
               goal.diff.rep[i,j] <- goals1.rep[i,j] - goals2.rep[i,j]
            # points earned by each home team (i)
               points1[i,j] <- 3*(1-step(-goal.diff.rep[i,j])) + 1*equals(goal.diff.rep[i,j],0)
            # points earned by each away team (j)
               points2[i,j] <- 3*(1-step( goal.diff.rep[i,j])) + 1*equals(goal.diff.rep[i,j],0)
  }
 }
       # calculation of the total points for each team
           for (i in 1:K){
               total.points[i] <- sum( points1[i,1:20] ) -  points1[i,i] + sum( points2[1:20,i] ) -  points2[i,i]
      }

       # ranking probabilities
            for (i in 1:K){
                ranks[i] <- 21-rank(total.points[], i)
            for (j in 1:K){
                rank.probs[i,j] <- equals( ranks[i], j )
       }
    }|
}
```

After checking the model, loading the data, compiling it, and loading the initial values we can set the desired number of iterations for the algorithm. Our primary focus now, lies in the convergence of the algorithm. This term refers to whether the algorithm has reached its equilibrium (target) distribution. If this is true, then the generated sample comes from the correct target distribution. Hence, monitoring the convergence of the algorithm is essential for producing results from the posterior distribution of interest. There are many ways to monitor convergence. The simplest way is to monitor MC error since small values of this error will indicate that we have calculated the quantities of interest with precision. Let us have a look on the MC errors produced in the simple Vanilla model after running 1000 iterations (500 burn in period were discarded):



num of chains 1
for chain 1
updates 1000
refresh 100
thin 1
iteration 1500

```
            goals1[i]~dpois(lambda1[i])
            goals2[i]~dpois(lambda2[i])
            # link and linear predictor
            log(lambda1[i])<-  mu + home + a[ ht[i] ] + d[ at[i] ]
            log(lambda2[i])<-  mu          + a[ at[i] ] + d[ ht[i] ]
      }
      # STZ constraints
      a[1]<-  -sum( a[2:20] )
      d[1]<-  -sum( d[2:20] )

      # prior distributions
      mu~dnorm(0,0.001)
      home~dnorm(0,0.001)
      for (i in 2:K){
            a[i]~dnorm(0,0.01)
            d[i]~dnorm(0,0.01)
      }

}
```

INITS

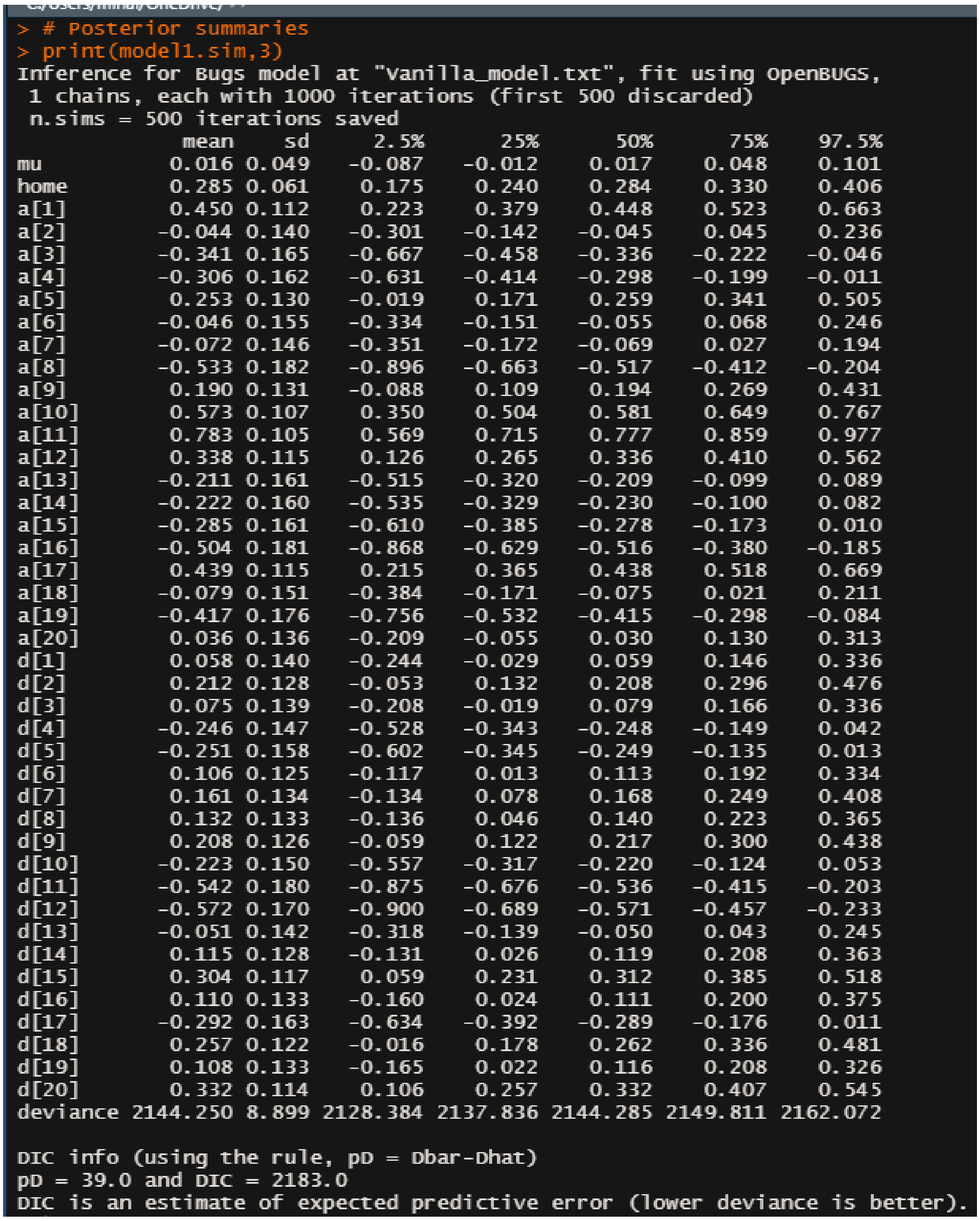
```
list( mu=0.5, home=0.5, a=c(NA, 0,0,0,0,0, 0,0,0,0,0, 0,0,0,0,0, 0,0,0,0) , d=c(NA, 0,0,0,0,0, 0,0,0,0,0,
0,0,0,0,0, 0,0,0,0)  )  # Chain 1

list( mu=0.25, home=0.25, a=c(NA, 0,0,0,0,0, 0,0,0,0,0, 0,0,0,0,0, 0,0,0,0) , d=c(NA, 0,0,0,0,0, 0,0,0,0,0,
0,0,0,0,0, 0,0,0,0)  )  # Chain 2

list( mu=0.75, home=0.75, a=c(NA, 0,0,0,0,0, 0,0,0,0,0, 0,0,0,0,0, 0,0,0,0) , d=c(NA, 0,0,0,0,0, 0,0,0,0,0,
0,0,0,0,0, 0,0,0,0)  )  # Chain 3
```

DATA - LIST FORMAT

```
list(n=380, K=20,
ht=c(1,3,5,6,7,14,18,19,12,13,2,4,9,10,14,15,16,8,17,11,2,6,8,12,1
3,18,5,10,17,19,1,3,7,9,11,14,15,4,16,20,2,6,8,10,13,17,18,19,5
,12,4,7,9,11,14,15,16,20,3,1,2,5,8,12,15,19,20,1,7,13,4,6,10,11
,16,17,18,3,14,9,20,5,8,11,13,14,15,16,7,17,1,2,6,10,12,18,19,3
,9,4,8,13,14,15,16,20,5,7,11,17,1,2,4,6,9,10,12,19,18,3,20,6,10
,12,13,16,17,4,8,14,3,9,18,19,1,2,5,7,11,15,1,3,5,7,9,15,18,19,
2,11,4,6,8,13,16,17,20,10,12,14,4,6,8,10,12,13,14,16,17,20,1,3,
5,9,11,15,18,2,19,7,1,3,4,7,9,11,14,15,16,20,2,5,8,10,12,17,18,
19,13,6,2,5,8,10,12,13,18,6,19,3,4,7,9,15,11,14,16,20,1,17,5,6,
8,13,17,18,19,2,10,12,1,3,4,7,9,11,15,20,14,16,8,16,20,5,7,11,1
3,14,15,17,1,2,3,4,9,12,19,6,10,18,7,11,15,16,17,20,8,13,14,5,2
,3,4,9,10,18,19,6,12,1,4,9,10,14,16,17,18,3,11,6,5,7,8,12,13,19
,20,1,2,15,2,8,10,15,3,6,7,12,13,18,19,20,1,5,2,3,7,9,11,15,18,
19,1,5,4,6,8,10,14,16,17,12,13,20,3,2,4,9,18,19,1,11,15,7,4,6,8
,10,13,14,16,12,20,17,3,2,7,9,15,18,19,1,5,11,16,5,9,11,17,20,4,6,8,10,12,13,14,16,17,20 ),
at=c(9,11,4,8,15,16,10,2,20,17,18,19,3,6,20,1,12,13,5,7,11,16,14,
9,20,3,7,1,4,15,2,19,17,5,10,18,12,6,13,8,3,14,9,4,15,16,11,20,
1,7,8,2,10,6,12,5,18,17,13,19,9,11,17,6,14,18,16,3,4,10,20,5,12
,15,8,2,1,7,13,19,3,18,12,4,6,19,2,9,1,10,16,5,20,8,17,15,11,14
,7,13,19,2,4,9,3,10,12,18,1,6,17,8,16,7,11,14,13,5,20,15,9,15,5
,3,18,2,19,1,11,7,6,17,12,13,8,4,16,20,14,10,12,10,13,8,4,16,17
,6,14,20,18,2,3,9,19,15,5,7,11,1,15,18,5,19,2,7,9,11,3,1,13,4,1
4,6,17,20,8,10,12,16,10,18,17,5,12,2,8,19,6,13,20,3,15,16,4,14,
9,7,11,1,7,15,4,9,14,3,16,11,1,2,10,12,8,13,18,6,17,19,5,20,9,4
```

```
        mean      sd       MC_error  val2.5pc  median    val97.5pc start  sample
a[1]    0.4515    0.1147   0.001667  0.2215    0.4516    0.6691    501    1000
a[2]    -0.04545  0.1409   0.006573  -0.3154   -0.04411  0.2352    501    1000
a[3]    -0.3369   0.1688   0.01032   -0.6855   -0.3284   -0.04557  501    1000
a[4]    -0.3002   0.1641   0.01036   -0.6301   -0.2964   0.01691   501    1000
a[5]    0.2548    0.1273   0.00659   -0.0108   0.2619    0.4932    501    1000
a[6]    -0.04982  0.1508   0.01087   -0.3396   -0.05182  0.2438    501    1000
a[7]    -0.07623  0.1473   0.005411  -0.3622   -0.07635  0.2137    501    1000
a[8]    -0.525    0.1849   0.01188   -0.905    -0.5071   -0.186    501    1000
a[9]    0.1845    0.1306   0.00601   -0.08704  0.1881    0.4378    501    1000
a[10]   0.5724    0.1079   0.004496  0.3529    0.5748    0.7798    501    1000
a[11]   0.7846    0.1      0.00468   0.576     0.7876    0.9741    501    1000
a[12]   0.3441    0.1159   0.005579  0.1264    0.3411    0.5779    501    1000
a[13]   -0.2024   0.1576   0.009058  -0.5251   -0.1967   0.09263   501    1000
a[14]   -0.2232   0.1607   0.008223  -0.5524   -0.2189   0.07967   501    1000
a[15]   -0.288    0.1662   0.01094   -0.6168   -0.2786   0.04242   501    1000
a[16]   -0.505    0.18     0.01103   -0.8923   -0.4994   -0.1751   501    1000
a[17]   0.4347    0.115    0.006141  0.2077    0.4373    0.6632    501    1000
a[18]   -0.07239  0.1475   0.008101  -0.3849   -0.06094  0.1889    501    1000
a[19]   -0.4329   0.1933   0.0121    -0.8463   -0.4267   -0.07269  501    1000
a[20]   0.03083   0.1429   0.008652  -0.2571   0.02686   0.3187    501    1000
d[1]    0.05857   0.1426   0.001596  -0.2358   0.05959   0.3411    501    1000
d[2]    0.2007    0.1278   0.005931  -0.05739  0.2002    0.456     501    1000
d[3]    0.07827   0.1372   0.006055  -0.1992   0.07984   0.344     501    1000
d[4]    -0.2503   0.1535   0.007069  -0.5617   -0.246    0.03866   501    1000
d[5]    -0.247    0.1535   0.006847  -0.5813   -0.2448   0.02633   501    1000
d[6]    0.11      0.1307   0.005749  -0.1404   0.1126    0.3562    501    1000
d[7]    0.1542    0.1337   0.0058    -0.1124   0.1603    0.4093    501    1000
d[8]    0.1383    0.1326   0.006056  -0.1275   0.141     0.3985    501    1000
d[9]    0.2033    0.1284   0.006006  -0.06427  0.2072    0.4383    501    1000
d[10]   -0.2242   0.156    0.007501  -0.5477   -0.2223   0.05879   501    1000
d[11]   -0.5539   0.1864   0.008938  -0.9142   -0.5455   -0.1892   501    1000
d[12]   -0.5581   0.1812   0.008875  -0.937    -0.5505   -0.2079   501    1000
d[13]   -0.05123  0.139    0.006282  -0.3145   -0.04839  0.2309    501    1000
d[14]   0.1207    0.1239   0.004904  -0.1313   0.1271    0.3571    501    1000
d[15]   0.3075    0.1177   0.00485   0.0654    0.3133    0.5434    501    1000
d[16]   0.1054    0.1302   0.004556  -0.1586   0.1014    0.3525    501    1000
d[17]   -0.2811   0.1591   0.007839  -0.6171   -0.2781   0.02163   501    1000
d[18]   0.2548    0.1238   0.005544  -0.01233  0.2618    0.4868    501    1000
d[19]   0.1086    0.1349   0.005169  -0.1628   0.1156    0.3625    501    1000
d[20]   0.3254    0.1122   0.004398  0.1045    0.3257    0.5476    501    1000
home    0.2852    0.06166  0.003607  0.161     0.2857    0.4067    501    1000
mu      0.01716   0.04808  0.002899  -0.08389  0.01883   0.1056    501    1000
```
`

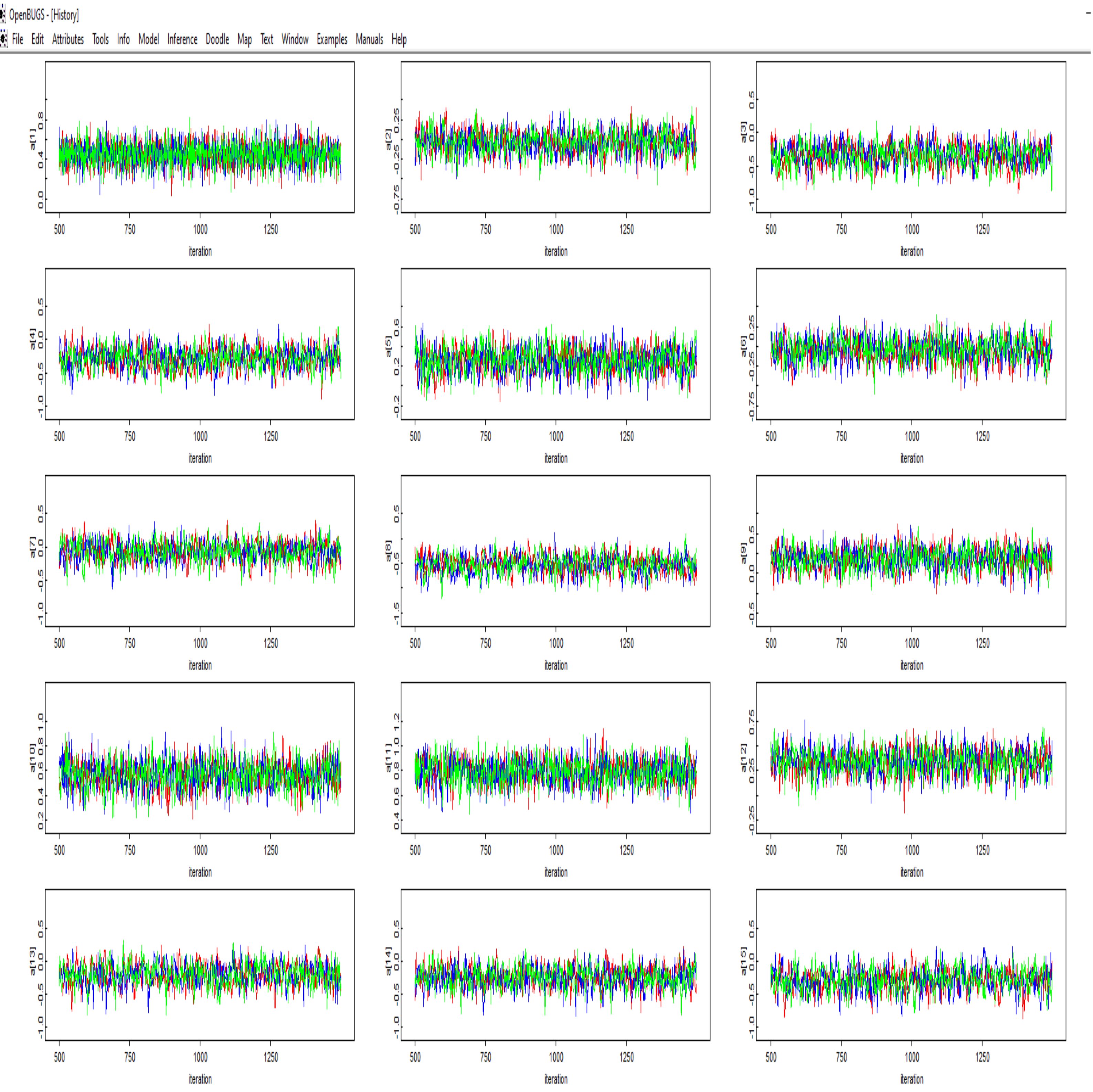
We notice that the MC errors are quite small so there are indications of convergence! Let’s have a look on the same posterior summaries using RStudio after running the same number of iterations:

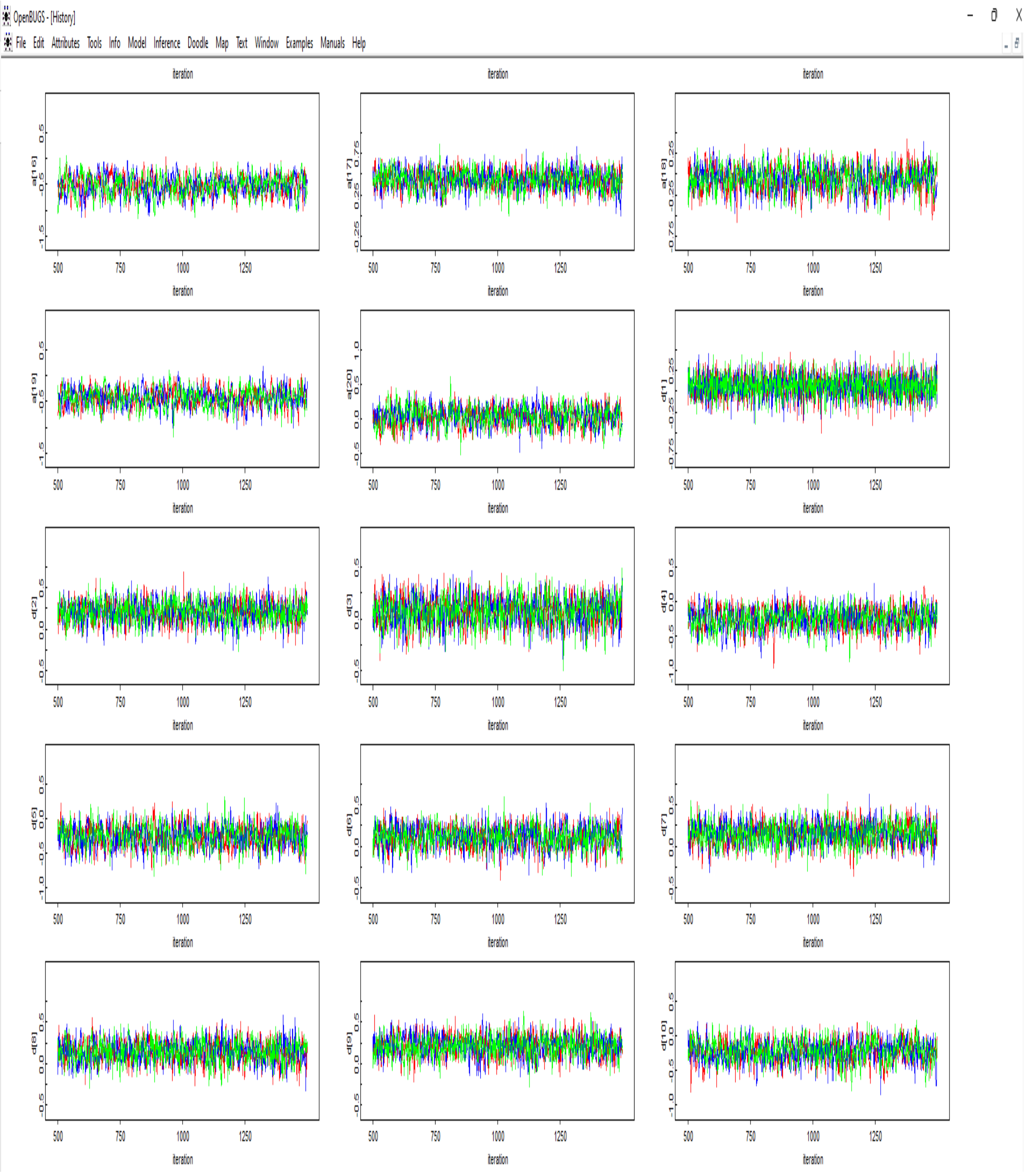


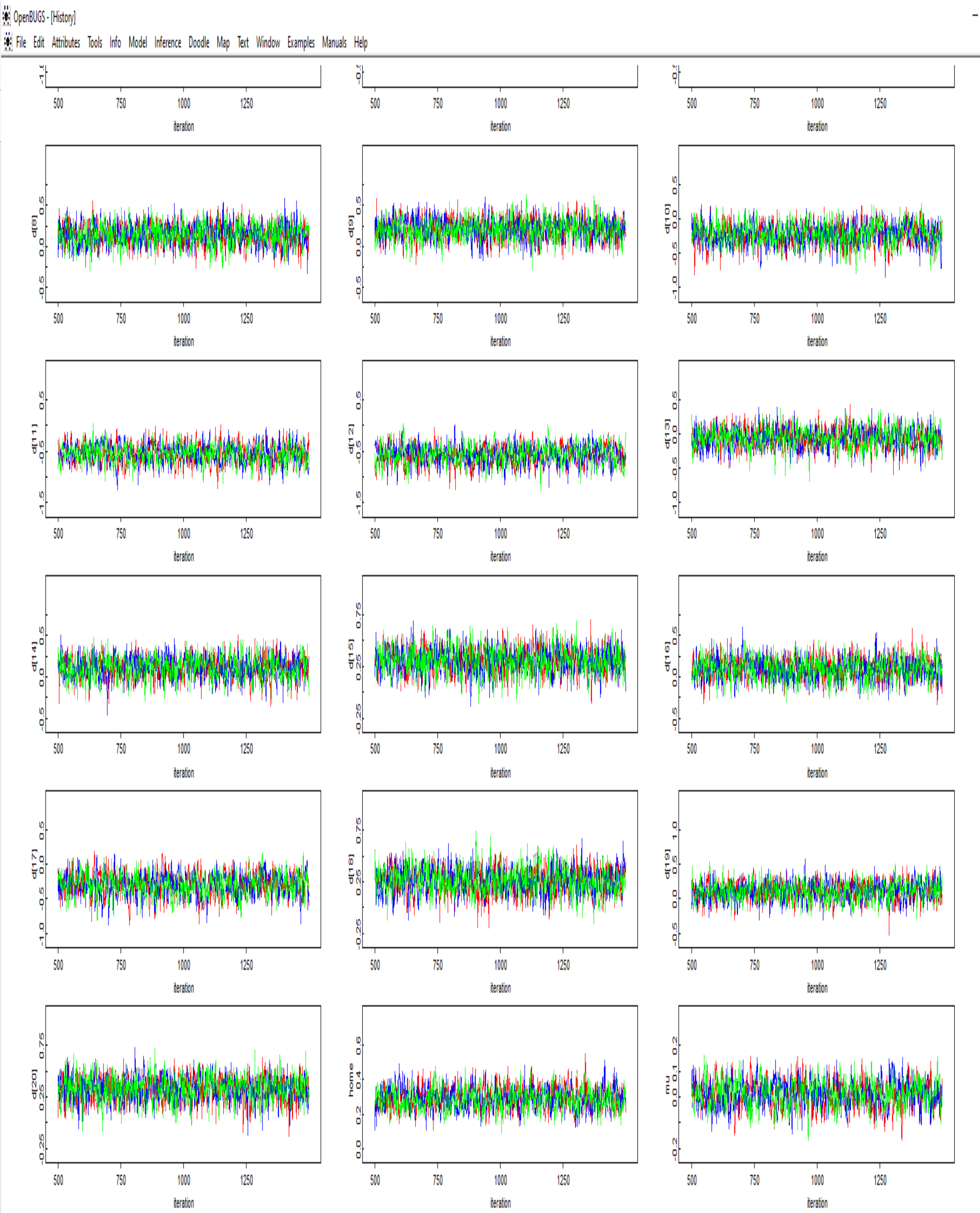
```
> # Posterior summaries
> print(model1.sim,3)
Inference for Bugs model at "Vanilla_model.txt", fit using OpenBUGS,
 1 chains, each with 1000 iterations (first 500 discarded)
 n.sims = 500 iterations saved
             mean    sd     2.5%      25%      50%      75%    97.5%
mu          0.016 0.049   -0.087   -0.012    0.017    0.048    0.101
home        0.285 0.061    0.175    0.240    0.284    0.330    0.406
a[1]        0.450 0.112    0.223    0.379    0.448    0.523    0.663
a[2]       -0.044 0.140   -0.301   -0.142   -0.045    0.045    0.236
a[3]       -0.341 0.165   -0.667   -0.458   -0.336   -0.222   -0.046
a[4]       -0.306 0.162   -0.631   -0.414   -0.298   -0.199   -0.011
a[5]        0.253 0.130   -0.019    0.171    0.259    0.341    0.505
a[6]       -0.046 0.155   -0.334   -0.151   -0.055    0.068    0.246
a[7]       -0.072 0.146   -0.351   -0.172   -0.069    0.027    0.194
a[8]       -0.533 0.182   -0.896   -0.663   -0.517   -0.412   -0.204
a[9]        0.190 0.131   -0.088    0.109    0.194    0.269    0.431
a[10]       0.573 0.107    0.350    0.504    0.581    0.649    0.767
a[11]       0.783 0.105    0.569    0.715    0.777    0.859    0.977
a[12]       0.338 0.115    0.126    0.265    0.336    0.410    0.562
a[13]      -0.211 0.161   -0.515   -0.320   -0.209   -0.099    0.089
a[14]      -0.222 0.160   -0.535   -0.329   -0.230   -0.100    0.082
a[15]      -0.285 0.161   -0.610   -0.385   -0.278   -0.173    0.010
a[16]      -0.504 0.181   -0.868   -0.629   -0.516   -0.380   -0.185
a[17]       0.439 0.115    0.215    0.365    0.438    0.518    0.669
a[18]      -0.079 0.151   -0.384   -0.171   -0.075    0.021    0.211
a[19]      -0.417 0.176   -0.756   -0.532   -0.415   -0.298   -0.084
a[20]       0.036 0.136   -0.209   -0.055    0.030    0.130    0.313
d[1]        0.058 0.140   -0.244   -0.029    0.059    0.146    0.336
d[2]        0.212 0.128   -0.053    0.132    0.208    0.296    0.476
d[3]        0.075 0.139   -0.208   -0.019    0.079    0.166    0.336
d[4]       -0.246 0.147   -0.528   -0.343   -0.248   -0.149    0.042
d[5]       -0.251 0.158   -0.602   -0.345   -0.249   -0.135    0.013
d[6]        0.106 0.125   -0.117    0.013    0.113    0.192    0.334
d[7]        0.161 0.134   -0.134    0.078    0.168    0.249    0.408
d[8]        0.132 0.133   -0.136    0.046    0.140    0.223    0.365
d[9]        0.208 0.126   -0.059    0.122    0.217    0.300    0.438
d[10]      -0.223 0.150   -0.557   -0.317   -0.220   -0.124    0.053
d[11]      -0.542 0.180   -0.875   -0.676   -0.536   -0.415   -0.203
d[12]      -0.572 0.170   -0.900   -0.689   -0.571   -0.457   -0.233
d[13]      -0.051 0.142   -0.318   -0.139   -0.050    0.043    0.245
d[14]       0.115 0.128   -0.131    0.026    0.119    0.208    0.363
d[15]       0.304 0.117    0.059    0.231    0.312    0.385    0.518
d[16]       0.110 0.133   -0.160    0.024    0.111    0.200    0.375
d[17]      -0.292 0.163   -0.634   -0.392   -0.289   -0.176    0.011
d[18]       0.257 0.122   -0.016    0.178    0.262    0.336    0.481
d[19]       0.108 0.133   -0.165    0.022    0.116    0.208    0.326
d[20]       0.332 0.114    0.106    0.257    0.332    0.407    0.545
deviance 2144.250 8.899 2128.384 2137.836 2144.285 2149.811 2162.072

DIC info (using the rule, pD = Dbar-Dhat)
pD = 39.0 and DIC = 2183.0
DIC is an estimate of expected predictive error (lower deviance is better).
```

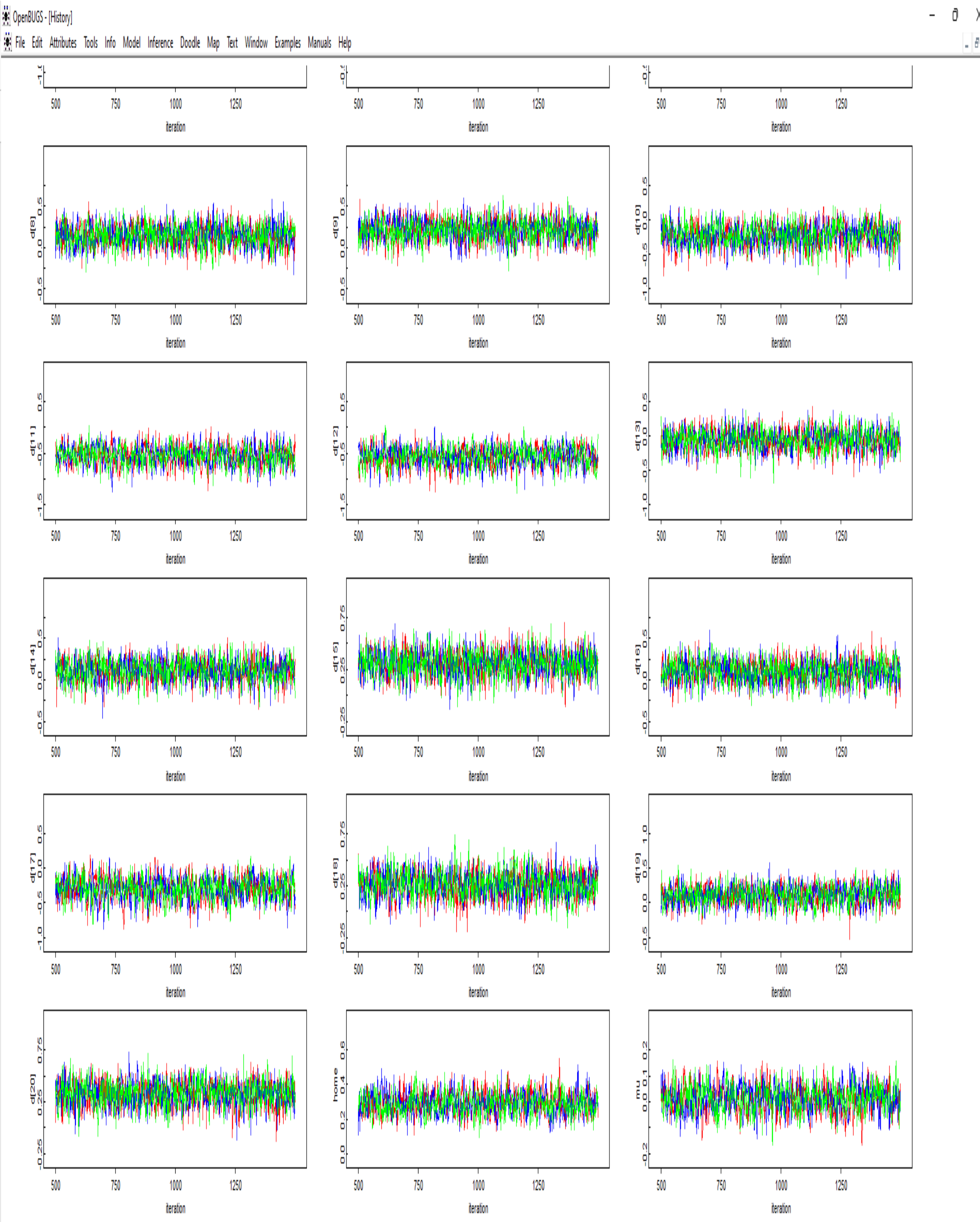
We observe that the DIC (Deviance Information Criterion) is valued at 2183.

The convergence of the chain can also be checked visually using trace plots (plots of the iterations versus the generated values) obtained by the History option in the sample monitor tool:

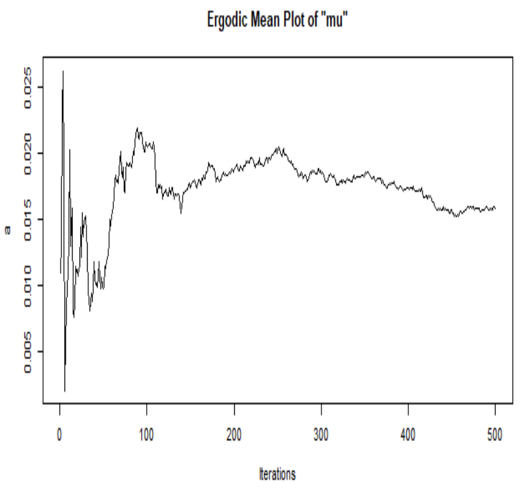

OpenBUGS - [History]
File Edit Attributes Tools Info Model Inference Doodle Map Text Window Examples Manuals Help
a[1]
a[2]
a[3]
a[4]
a[5]
a[6]
a[7]
a[8]
a[9]
a[10]
a[11]
a[12]
a[13]
a[14]
a[15]
500 750 1000 1250
iteration

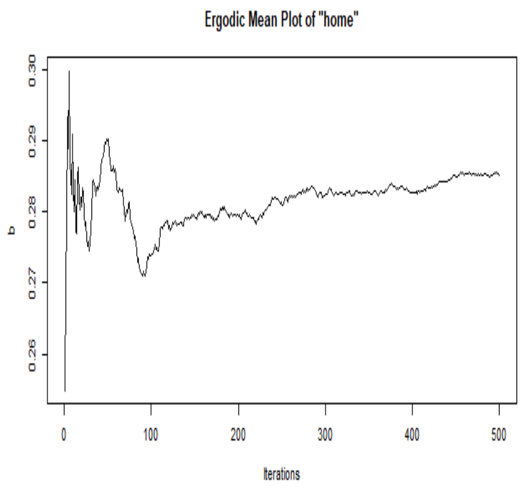

OpenBUGS - [History]
File Edit Attributes Tools Info Model Inference Doodle Map Text Window Examples Manuals Help
iteration
a[16]
a[17]
a[18]
a[19]
a[20]
d[1]
d[2]
d[3]
d[4]
d[5]
d[6]
d[7]
d[8]
d[9]
d[10]
500
750
1000
1250

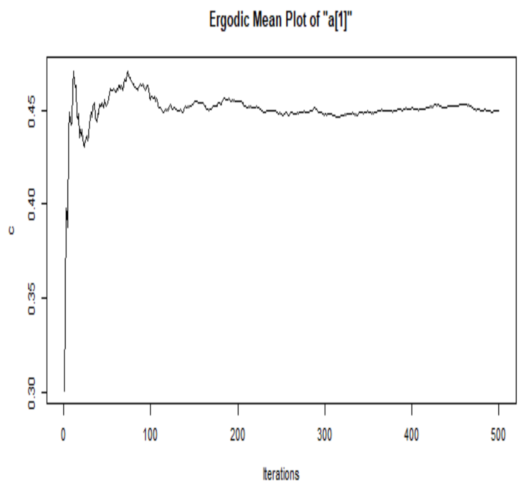

OpenBUGS - [History]
File Edit Attributes Tools Info Model Inference Doodle Map Text Window Examples Manuals Help
d[8]
d[9]
d[10]
d[11]
d[12]
d[13]
d[14]
d[15]
d[16]
d[17]
d[18]
d[19]
d[20]
home
mu
iteration
500 750 1000 1250

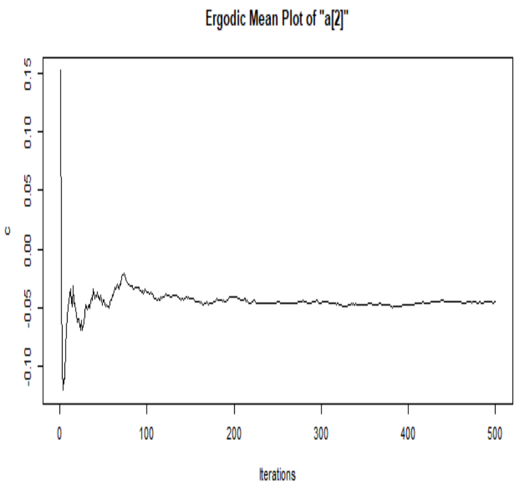

OpenBUGS - [History]
File Edit Attributes Tools Info Model Inference Doodle Map Text Window Examples Manuals Help
d[8]
d[9]
d[10]
d[11]
d[12]
d[13]
d[14]
d[15]
d[16]
d[17]
d[18]
d[19]
d[20]
home
mu
iteration
500 750 1000 1250

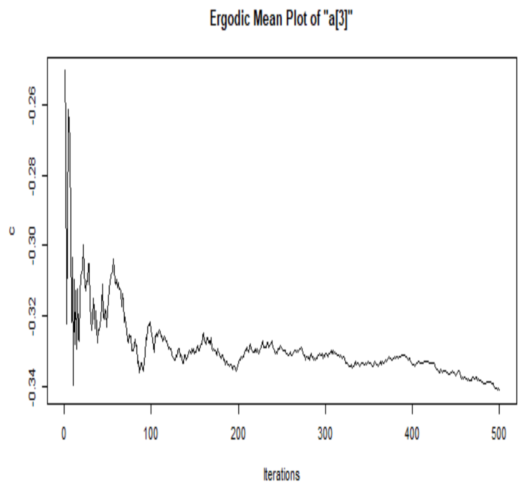
Another useful plot is produced by depicting the evolution of the ergodic mean of a quantity over the number of iterations. The term ergodic mean refers to the mean value until the current iteration. If the ergodic mean stabilizes after some iterations, then this is an indication of the convergence of the algorithm. Let’s have a glimpse on the ergodic mean plots to have more indications about the (much) desired convergence:

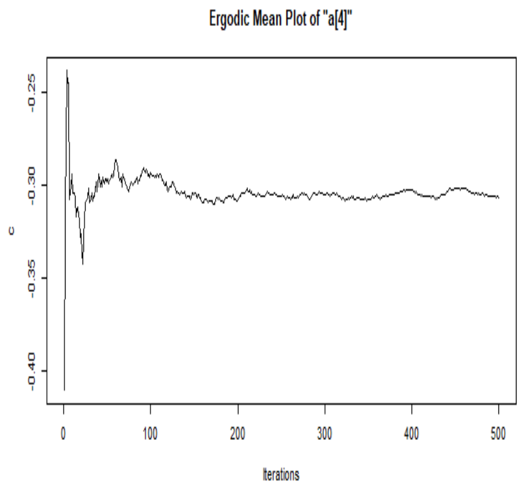


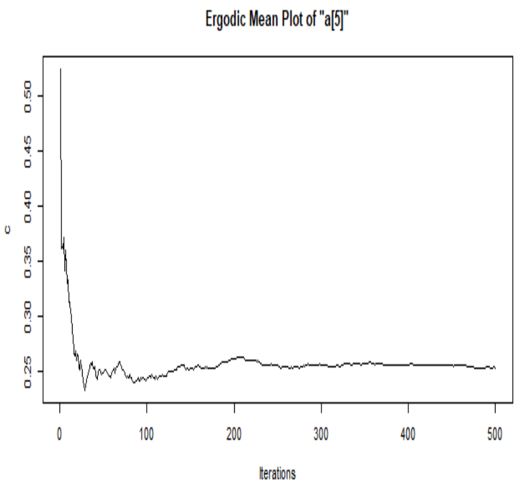


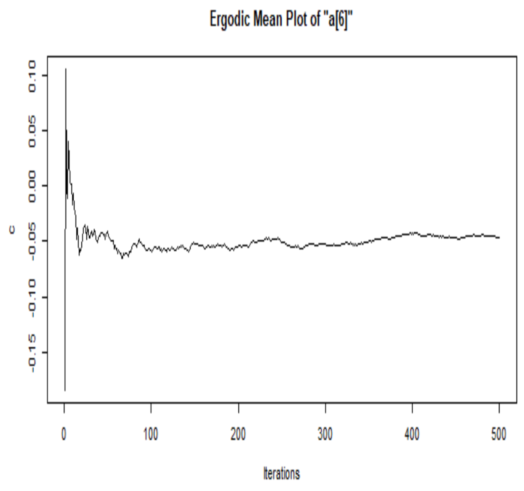


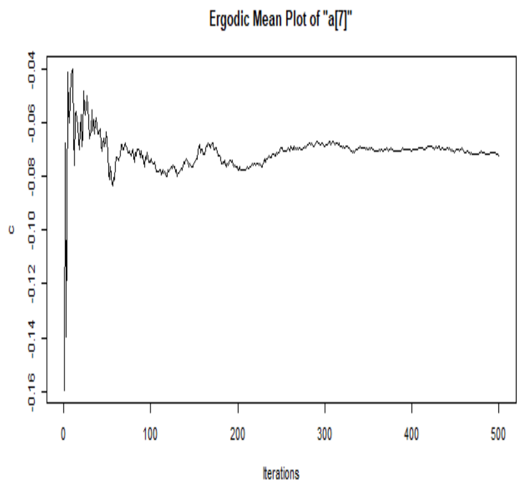




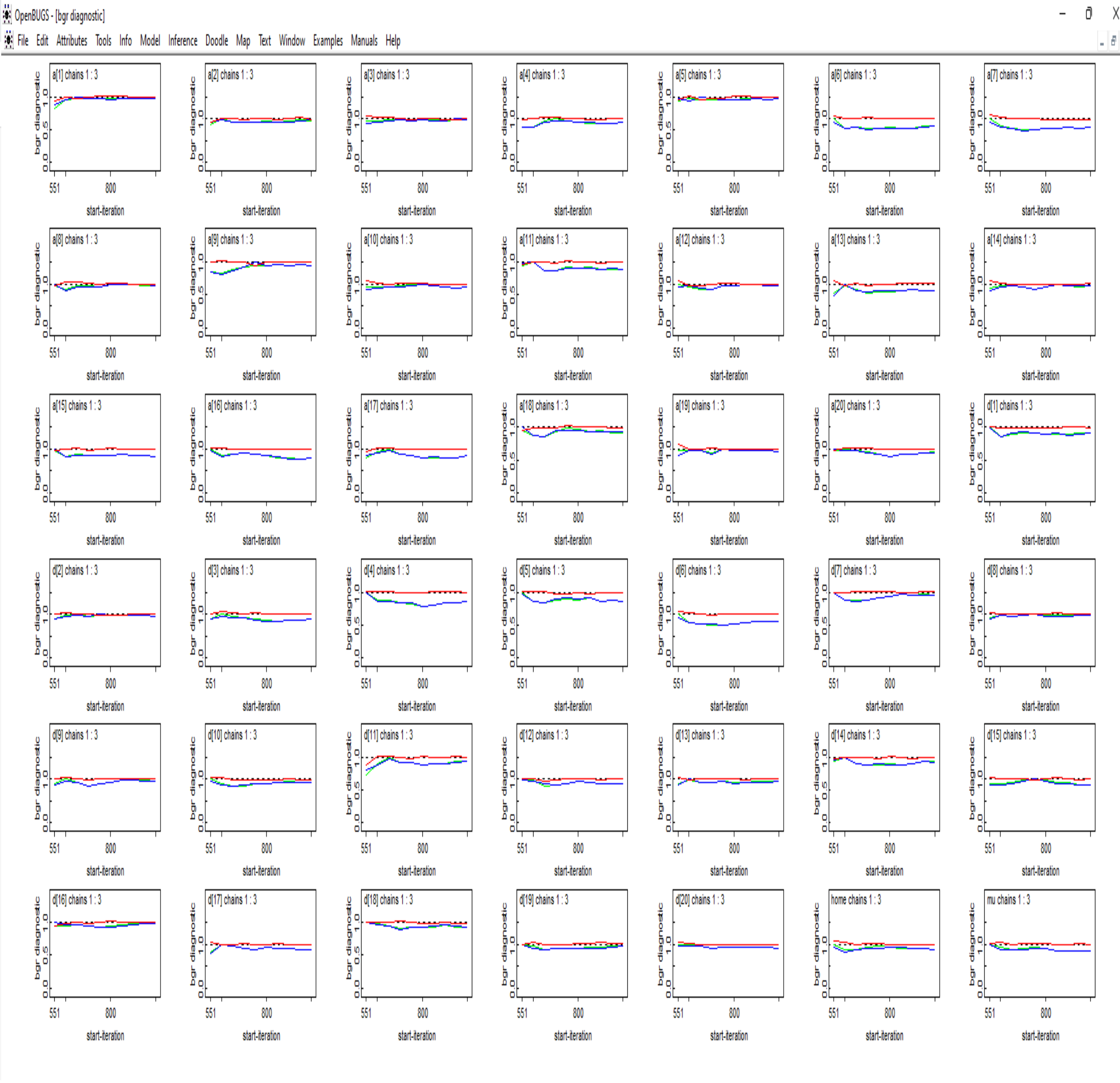




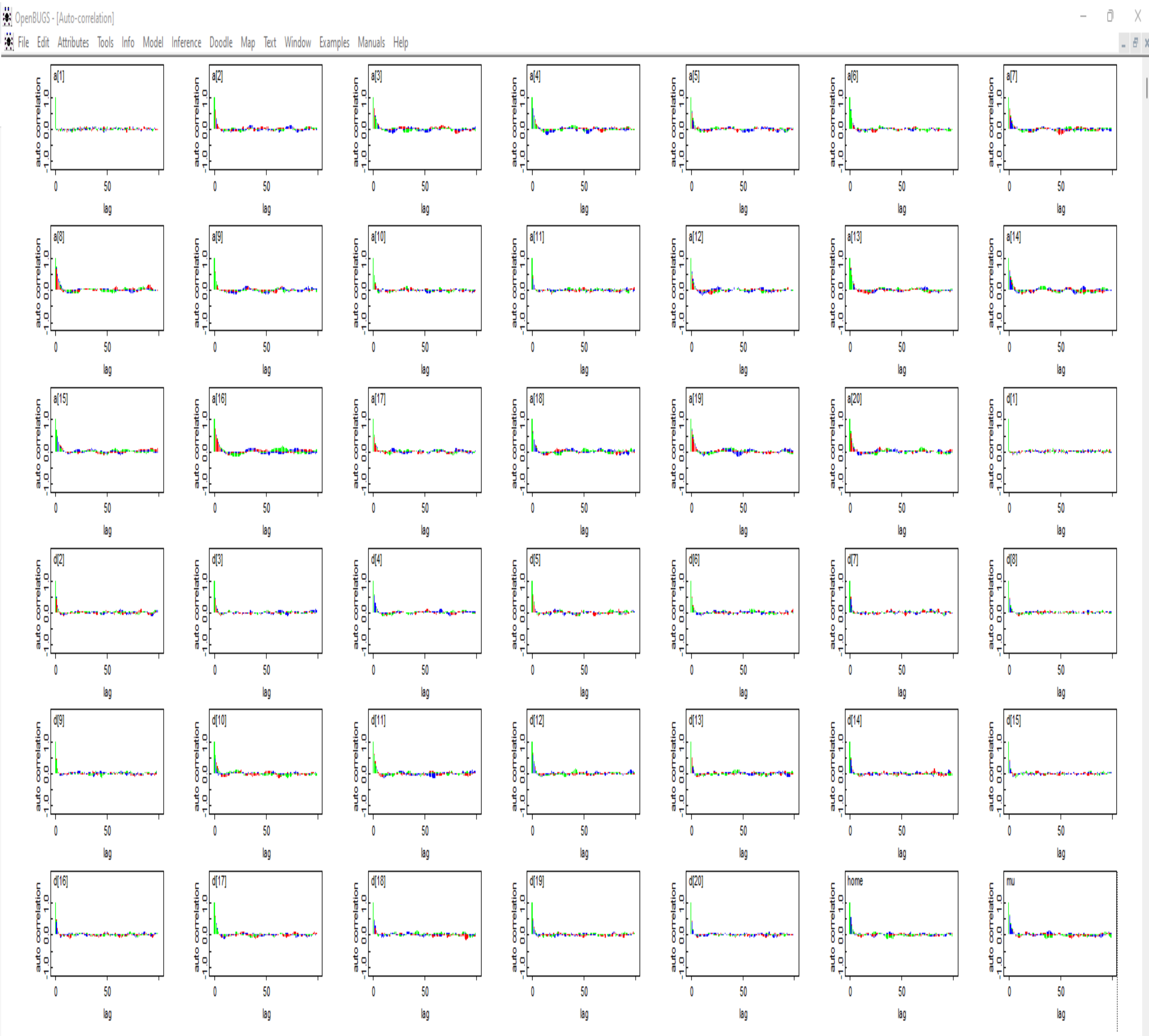




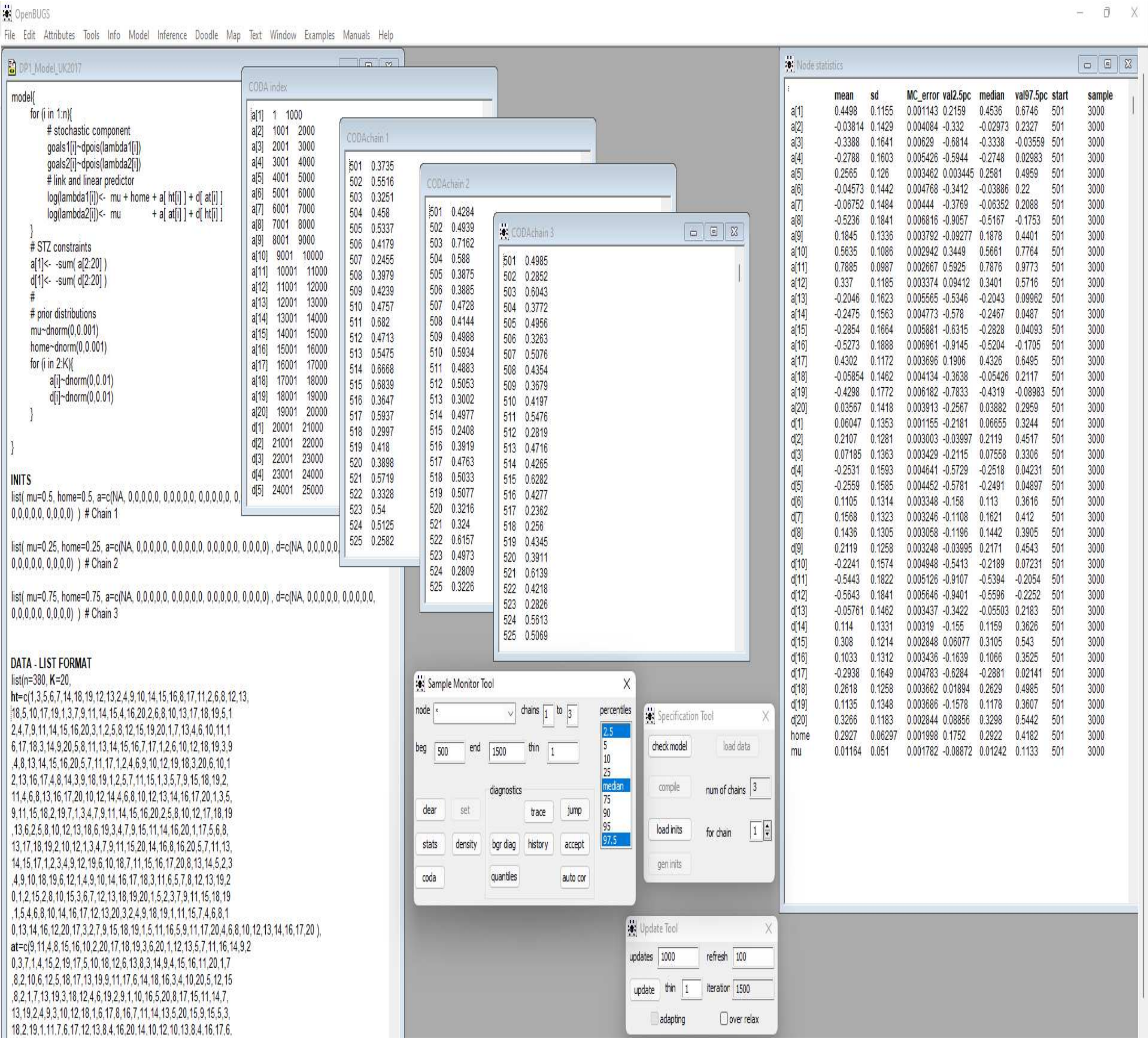
A different way to check the convergence of the algorithm, is based on the bgr diag option (formal convergence diagnostic tool) in OpenBUGS, since we specified 3 chains that should finally converge at the same point, fact that can be depicted below:


OpenBUGS - [bgr diagnostic]
File Edit Attributes Tools Info Model Inference Doodle Map Text Window Examples Manuals Help
a[1] chains 1 : 3
bgr diagnostic
start-iteration
551
800
mu chains 1 : 3
home chains 1 : 3
d[1] chains 1 : 3

We can further monitor convergence using the autocorrelations plots. If autocorrelations are low, then convergence is obtained in a relatively low number of iterations! They are indeed low, as it is depicted below:


OpenBUGS - [Auto-correlation]
File Edit Attributes Tools Info Model Inference Doodle Map Text Window Examples Manuals Help
a[1] a[2] a[3] a[4] a[5] a[6] a[7]
a[8] a[9] a[10] a[11] a[12] a[13] a[14]
a[15] a[16] a[17] a[18] a[19] a[20] d[1]
d[2] d[3] d[4] d[5] d[6] d[7] d[8]
d[9] d[10] d[11] d[12] d[13] d[14] d[15]
d[16] d[17] d[18] d[19] d[20] home mu
auto correlation
1.0 0.0 -1.0
0 50
lag

Finally, we can use the coda package in R to obtain additional model diagnostics. Files required by CODA are opened in separate windows that can be saved and imported in R as follows. Firstly, we obtain them by selecting option << coda >> in OpenBUGS sample menu:



```
model{
     for (i in 1:n){
          # stochastic component
          goals1[i]~dpois(lambda1[i])
          goals2[i]~dpois(lambda2[i])
          # link and linear predictor
          log(lambda1[i])<-  mu + home + a[ ht[i] ] + d[ at[i] ]
          log(lambda2[i])<-  mu          + a[ at[i] ] + d[ ht[i] ]
     }
     # STZ constraints
     a[1]<-  -sum( a[2:20] )
     d[1]<-  -sum( d[2:20] )
     #
     # prior distributions
     mu~dnorm(0,0.001)
     home~dnorm(0,0.001)
     for (i in 2:K){
           a[i]~dnorm(0,0.01)
           d[i]~dnorm(0,0.01)
     }

}

INITS
list( mu=0.5, home=0.5, a=c(NA, 0,0,0,0,0, 0,0,0,0,0, 0,0,0,0,0, 0,
0,0,0,0,0, 0,0,0,0) ) # Chain 1

list( mu=0.25, home=0.25, a=c(NA, 0,0,0,0,0, 0,0,0,0,0, 0,0,0,0,0, 0,0,0,0) , d=c(NA, 0,0,0,0,0,
0,0,0,0,0, 0,0,0,0) ) # Chain 2

list( mu=0.75, home=0.75, a=c(NA, 0,0,0,0,0, 0,0,0,0,0, 0,0,0,0,0, 0,0,0,0) , d=c(NA, 0,0,0,0,0, 0,0,0,0,0,
0,0,0,0,0, 0,0,0,0) ) # Chain 3


DATA - LIST FORMAT
list(n=380, K=20,
ht=c(1,3,5,6,7,14,18,19,12,13,2,4,9,10,14,15,16,8,17,11,2,6,8,12,13,
18,5,10,17,19,1,3,7,9,11,14,15,4,16,20,2,6,8,10,13,17,18,19,5,1
2,4,7,9,11,14,15,16,20,3,1,2,5,8,12,15,19,20,1,7,13,4,6,10,11,1
6,17,18,3,14,9,20,5,8,11,13,14,15,16,7,17,1,2,6,10,12,18,19,3,9
,4,8,13,14,15,16,20,5,7,11,17,1,2,4,6,9,10,12,19,18,3,20,6,10,1
2,13,16,17,4,8,14,3,9,18,19,1,2,5,7,11,15,1,3,5,7,9,15,18,19,2,
11,4,6,8,13,16,17,20,10,12,14,4,6,8,10,12,13,14,16,17,20,1,3,5,
9,11,15,18,2,19,7,1,3,4,7,9,11,14,15,16,20,2,5,8,10,12,17,18,19
,13,6,2,5,8,10,12,13,18,6,19,3,4,7,9,15,11,14,16,20,1,17,5,6,8,
13,17,18,19,2,10,12,1,3,4,7,9,11,15,20,14,16,8,16,20,5,7,11,13,
14,15,17,1,2,3,4,9,12,19,6,10,18,7,11,15,16,17,20,8,13,14,5,2,3
,4,9,10,18,19,6,12,1,4,9,10,14,16,17,18,3,11,6,5,7,8,12,13,19,2
0,1,2,15,2,8,10,15,3,6,7,12,13,18,19,20,1,5,2,3,7,9,11,15,18,19
,1,5,4,6,8,10,14,16,17,12,13,20,3,2,4,9,18,19,1,11,15,7,4,6,8,1
0,13,14,16,12,20,17,3,2,7,9,15,18,19,1,5,11,16,5,9,11,17,20,4,6,8,10,12,13,14,16,17,20 ),
at=c(9,11,4,8,15,16,10,2,20,17,18,19,3,6,20,1,12,13,5,7,11,16,14,9,2
0,3,7,1,4,15,2,19,17,5,10,18,12,6,13,8,3,14,9,4,15,16,11,20,1,7
,8,2,10,6,12,5,18,17,13,19,9,11,17,6,14,18,16,3,4,10,20,5,12,15
,8,2,1,7,13,19,3,18,12,4,6,19,2,9,1,10,16,5,20,8,17,15,11,14,7,
13,19,2,4,9,3,10,12,18,1,6,17,8,16,7,11,14,13,5,20,15,9,15,5,3,
18,2,19,1,11,7,6,17,12,13,8,4,16,20,14,10,12,10,13,8,4,16,17,6,
```

CODA index

```
a[1]   1      1000
a[2]   1001   2000
a[3]   2001   3000
a[4]   3001   4000
a[5]   4001   5000
a[6]   5001   6000
a[7]   6001   7000
a[8]   7001   8000
a[9]   8001   9000
a[10]  9001   10000
a[11]  10001  11000
a[12]  11001  12000
a[13]  12001  13000
a[14]  13001  14000
a[15]  14001  15000
a[16]  15001  16000
a[17]  16001  17000
a[18]  17001  18000
a[19]  18001  19000
a[20]  19001  20000
d[1]   20001  21000
d[2]   21001  22000
d[3]   22001  23000
d[4]   23001  24000
d[5]   24001  25000
```

CODAchain 1

```
501  0.3735
502  0.5516
503  0.3251
504  0.458
505  0.5337
506  0.4179
507  0.2455
508  0.3979
509  0.4239
510  0.4757
511  0.682
512  0.4713
513  0.5475
514  0.6668
515  0.6839
516  0.3647
517  0.5937
518  0.2997
519  0.418
520  0.3898
521  0.5719
522  0.3328
523  0.54
524  0.5125
525  0.2582
```

CODAchain 2

```
501  0.4284
502  0.4939
503  0.7162
504  0.588
505  0.3875
506  0.3885
507  0.4728
508  0.4144
509  0.4988
510  0.5934
511  0.4883
512  0.5053
513  0.3002
514  0.4977
515  0.2408
516  0.3919
517  0.4763
518  0.5033
519  0.5077
520  0.3216
521  0.324
522  0.6157
523  0.4973
524  0.2809
525  0.3226
```

CODAchain 3

```
501  0.4985
502  0.2852
503  0.6043
504  0.3772
505  0.4956
506  0.3263
507  0.5076
508  0.4354
509  0.3679
510  0.4197
511  0.5476
512  0.2819
513  0.4716
514  0.4265
515  0.6282
516  0.4277
517  0.2362
518  0.256
519  0.4345
520  0.3911
521  0.6139
522  0.4218
523  0.2826
524  0.5613
525  0.5069
```

Node statistics

```
        mean      sd       MC_error  val2.5pc  median    val97.5pc start  sample
a[1]    0.4498    0.1155   0.001143  0.2159    0.4536    0.6746    501    3000
a[2]    -0.03814  0.1429   0.004084  -0.332    -0.02973  0.2327    501    3000
a[3]    -0.3388   0.1641   0.00629   -0.6814   -0.3338   -0.03559  501    3000
a[4]    -0.2788   0.1603   0.005426  -0.5944   -0.2748   0.02983   501    3000
a[5]    0.2565    0.126    0.003462  0.003445  0.2581    0.4959    501    3000
a[6]    -0.04573  0.1442   0.004768  -0.3412   -0.03886  0.22      501    3000
a[7]    -0.06752  0.1484   0.00444   -0.3769   -0.06352  0.2088    501    3000
a[8]    -0.5236   0.1841   0.006816  -0.9057   -0.5167   -0.1753   501    3000
a[9]    0.1845    0.1336   0.003792  -0.09277  0.1878    0.4401    501    3000
a[10]   0.5635    0.1086   0.002942  0.3449    0.5661    0.7764    501    3000
a[11]   0.7885    0.0987   0.002667  0.5925    0.7876    0.9773    501    3000
a[12]   0.337     0.1185   0.003374  0.09412   0.3401    0.5716    501    3000
a[13]   -0.2046   0.1623   0.005565  -0.5346   -0.2043   0.09962   501    3000
a[14]   -0.2475   0.1563   0.004773  -0.578    -0.2467   0.0487    501    3000
a[15]   -0.2854   0.1664   0.005881  -0.6315   -0.2828   0.04093   501    3000
a[16]   -0.5273   0.1888   0.006961  -0.9145   -0.5204   -0.1705   501    3000
a[17]   0.4302    0.1172   0.003696  0.1906    0.4326    0.6495    501    3000
a[18]   -0.05854  0.1462   0.004134  -0.3638   -0.05426  0.2117    501    3000
a[19]   -0.4298   0.1772   0.006182  -0.7833   -0.4319   -0.08983  501    3000
a[20]   0.03567   0.1418   0.003913  -0.2567   0.03882   0.2959    501    3000
d[1]    0.06047   0.1353   0.001155  -0.2181   0.06655   0.3244    501    3000
d[2]    0.2107    0.1281   0.003003  -0.03997  0.2119    0.4517    501    3000
d[3]    0.07185   0.1363   0.003429  -0.2115   0.07558   0.3306    501    3000
d[4]    -0.2531   0.1593   0.004641  -0.5729   -0.2518   0.04231   501    3000
d[5]    -0.2559   0.1585   0.004452  -0.5781   -0.2491   0.04897   501    3000
d[6]    0.1105    0.1314   0.003348  -0.158    0.113     0.3616    501    3000
d[7]    0.1568    0.1323   0.003246  -0.1108   0.1621    0.412     501    3000
d[8]    0.1436    0.1305   0.003058  -0.1196   0.1442    0.3905    501    3000
d[9]    0.2119    0.1258   0.003248  -0.03995  0.2171    0.4543    501    3000
d[10]   -0.2241   0.1574   0.004948  -0.5413   -0.2189   0.07231   501    3000
d[11]   -0.5443   0.1822   0.005126  -0.9107   -0.5394   -0.2054   501    3000
d[12]   -0.5643   0.1841   0.005646  -0.9401   -0.5596   -0.2252   501    3000
d[13]   -0.05761  0.1462   0.003437  -0.3422   -0.05503  0.2183    501    3000
d[14]   0.114     0.1331   0.00319   -0.155    0.1159    0.3626    501    3000
d[15]   0.308     0.1214   0.002848  0.06077   0.3105    0.543     501    3000
d[16]   0.1033    0.1312   0.003436  -0.1639   0.1066    0.3525    501    3000
d[17]   -0.2938   0.1649   0.004783  -0.6284   -0.2881   0.02141   501    3000
d[18]   0.2618    0.1258   0.003662  0.01894   0.2629    0.4985    501    3000
d[19]   0.1135    0.1348   0.003686  -0.1578   0.1178    0.3607    501    3000
d[20]   0.3266    0.1183   0.002844  0.08856   0.3298    0.5442    501    3000
home    0.2927    0.06297  0.001998  0.1752    0.2922    0.4182    501    3000
mu      0.01164   0.051    0.001782  -0.08872  0.01242   0.1133    501    3000
```

Sample Monitor Tool

node * chains 1 to 3
beg 500 end 1500 thin 1
percentiles: [✓ 2.5] 5 10 25 [✓ median] 75 90 95 [✓ 97.5]

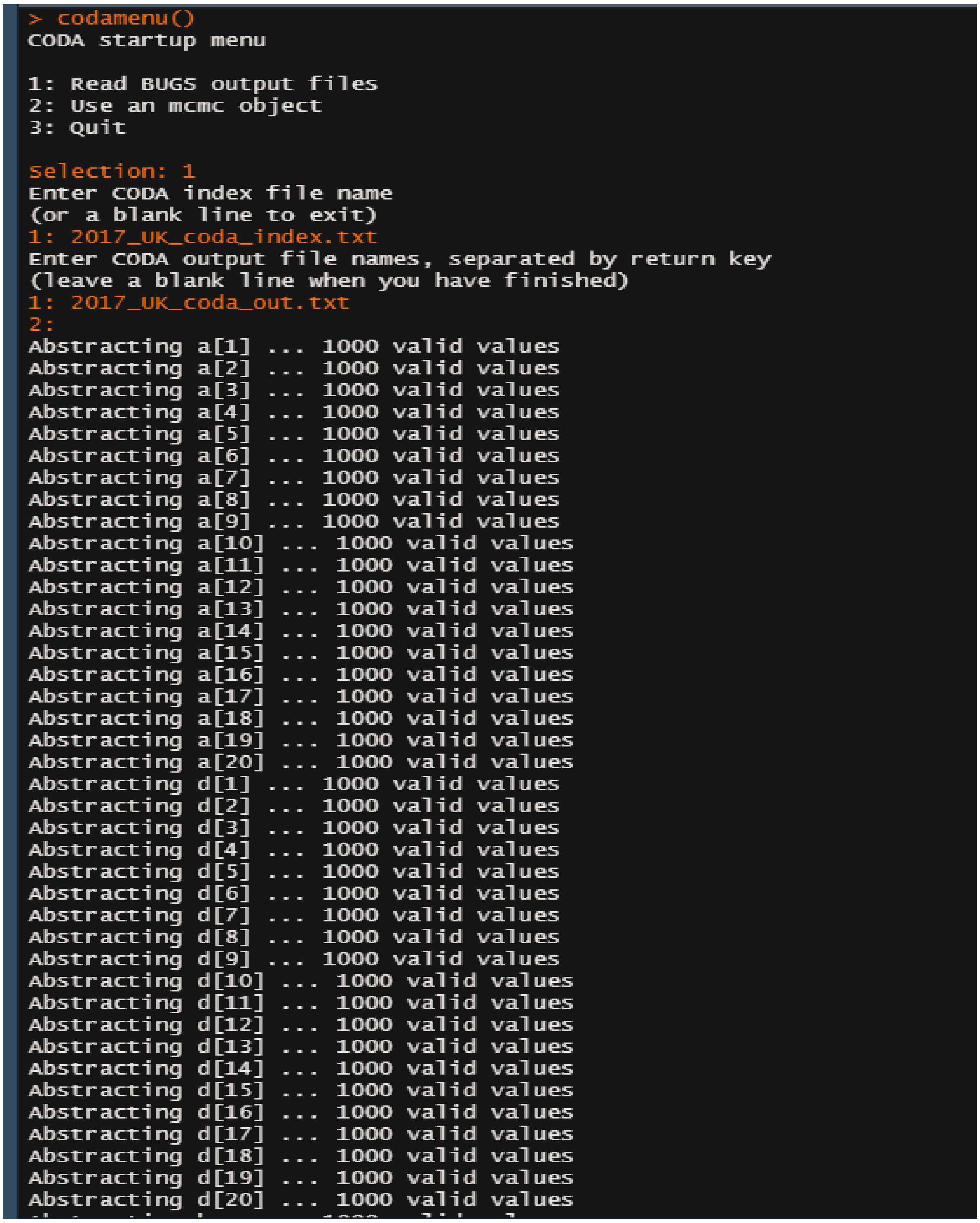
Specification Tool

num of chains 3
for chain 1

Update Tool

updates 1000 refresh 100
thin 1 iteration 1500

Using the option CODA in OpenBUGS we observe a set of windows with the sampled values in a format compatible to the one used by CODA software. CODA is an add-in package for Splus and R that is used for checking convergence of the algorithm using a variety of diagnostics. Now, let’s import them in R:

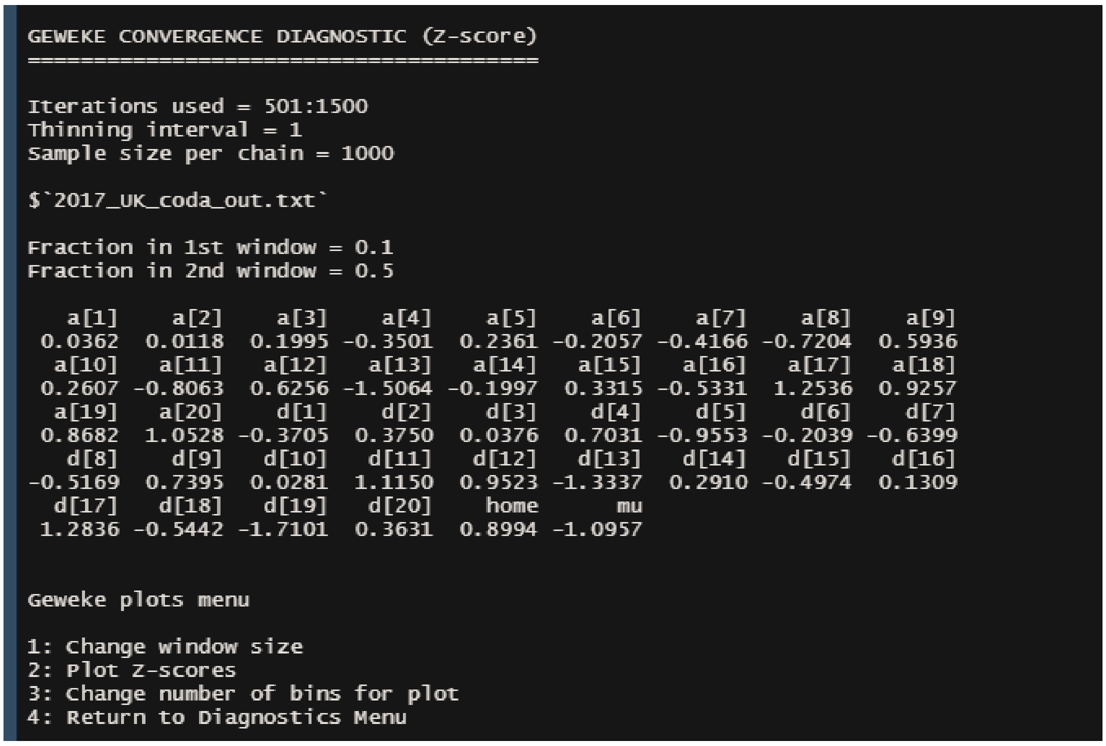


```
> codamenu()
CODA startup menu

1: Read BUGS output files
2: Use an mcmc object
3: Quit

Selection: 1
Enter CODA index file name
(or a blank line to exit)
1: 2017_UK_coda_index.txt
Enter CODA output file names, separated by return key
(leave a blank line when you have finished)
1: 2017_UK_coda_out.txt
2:
Abstracting a[1] ... 1000 valid values
Abstracting a[2] ... 1000 valid values
Abstracting a[3] ... 1000 valid values
Abstracting a[4] ... 1000 valid values
Abstracting a[5] ... 1000 valid values
Abstracting a[6] ... 1000 valid values
Abstracting a[7] ... 1000 valid values
Abstracting a[8] ... 1000 valid values
Abstracting a[9] ... 1000 valid values
Abstracting a[10] ... 1000 valid values
Abstracting a[11] ... 1000 valid values
Abstracting a[12] ... 1000 valid values
Abstracting a[13] ... 1000 valid values
Abstracting a[14] ... 1000 valid values
Abstracting a[15] ... 1000 valid values
Abstracting a[16] ... 1000 valid values
Abstracting a[17] ... 1000 valid values
Abstracting a[18] ... 1000 valid values
Abstracting a[19] ... 1000 valid values
Abstracting a[20] ... 1000 valid values
Abstracting d[1] ... 1000 valid values
Abstracting d[2] ... 1000 valid values
Abstracting d[3] ... 1000 valid values
Abstracting d[4] ... 1000 valid values
Abstracting d[5] ... 1000 valid values
Abstracting d[6] ... 1000 valid values
Abstracting d[7] ... 1000 valid values
Abstracting d[8] ... 1000 valid values
Abstracting d[9] ... 1000 valid values
Abstracting d[10] ... 1000 valid values
Abstracting d[11] ... 1000 valid values
Abstracting d[12] ... 1000 valid values
Abstracting d[13] ... 1000 valid values
Abstracting d[14] ... 1000 valid values
Abstracting d[15] ... 1000 valid values
Abstracting d[16] ... 1000 valid values
Abstracting d[17] ... 1000 valid values
Abstracting d[18] ... 1000 valid values
Abstracting d[19] ... 1000 valid values
Abstracting d[20] ... 1000 valid values
```

Coda provides four diagnostic tests suggested respectively by Geweke (1992), Gelman and Rubin (1992), Raftery and Lewis (1992), and Heidelberger and Welch (1992).

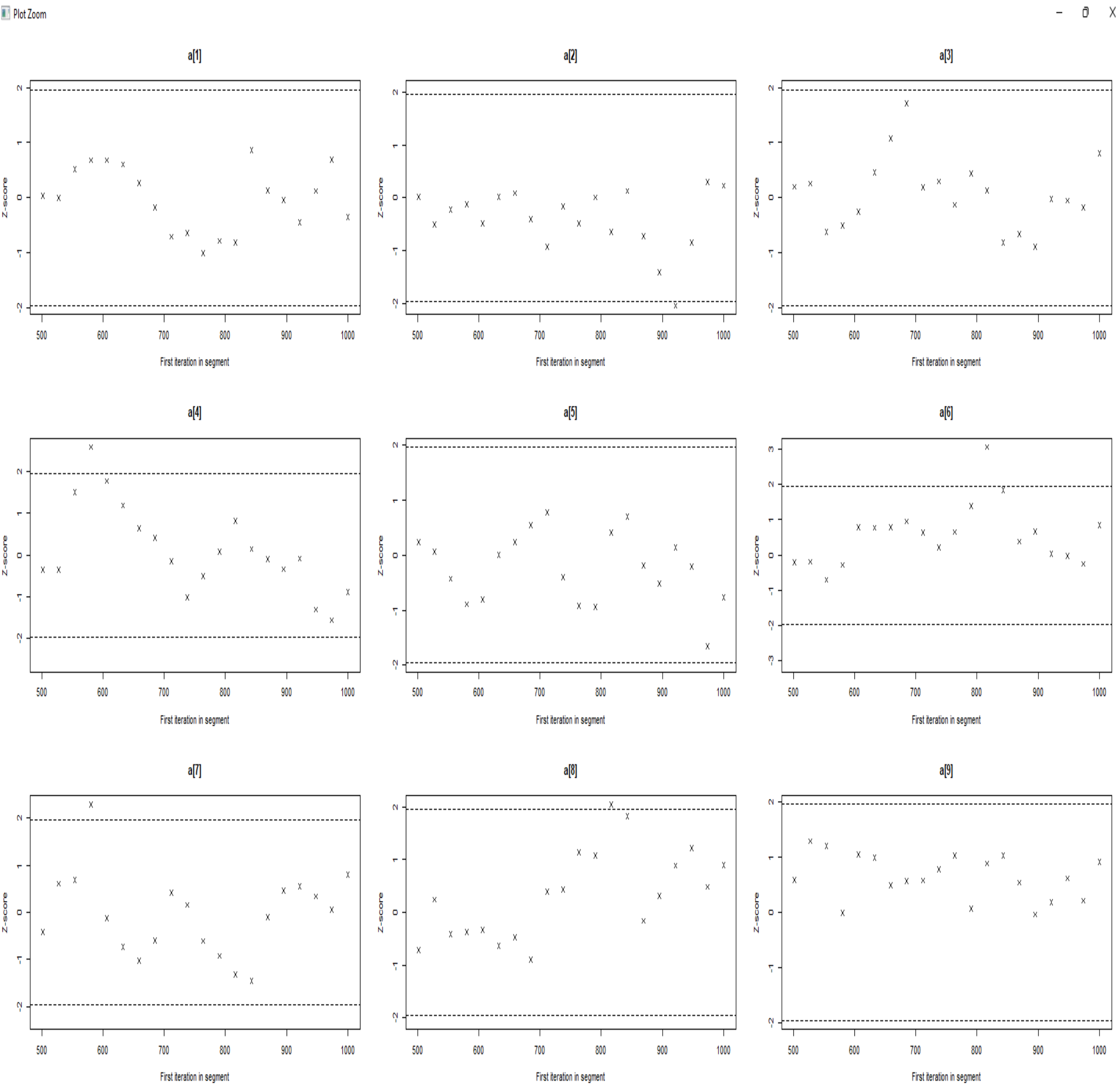
The Geweke diagnostic

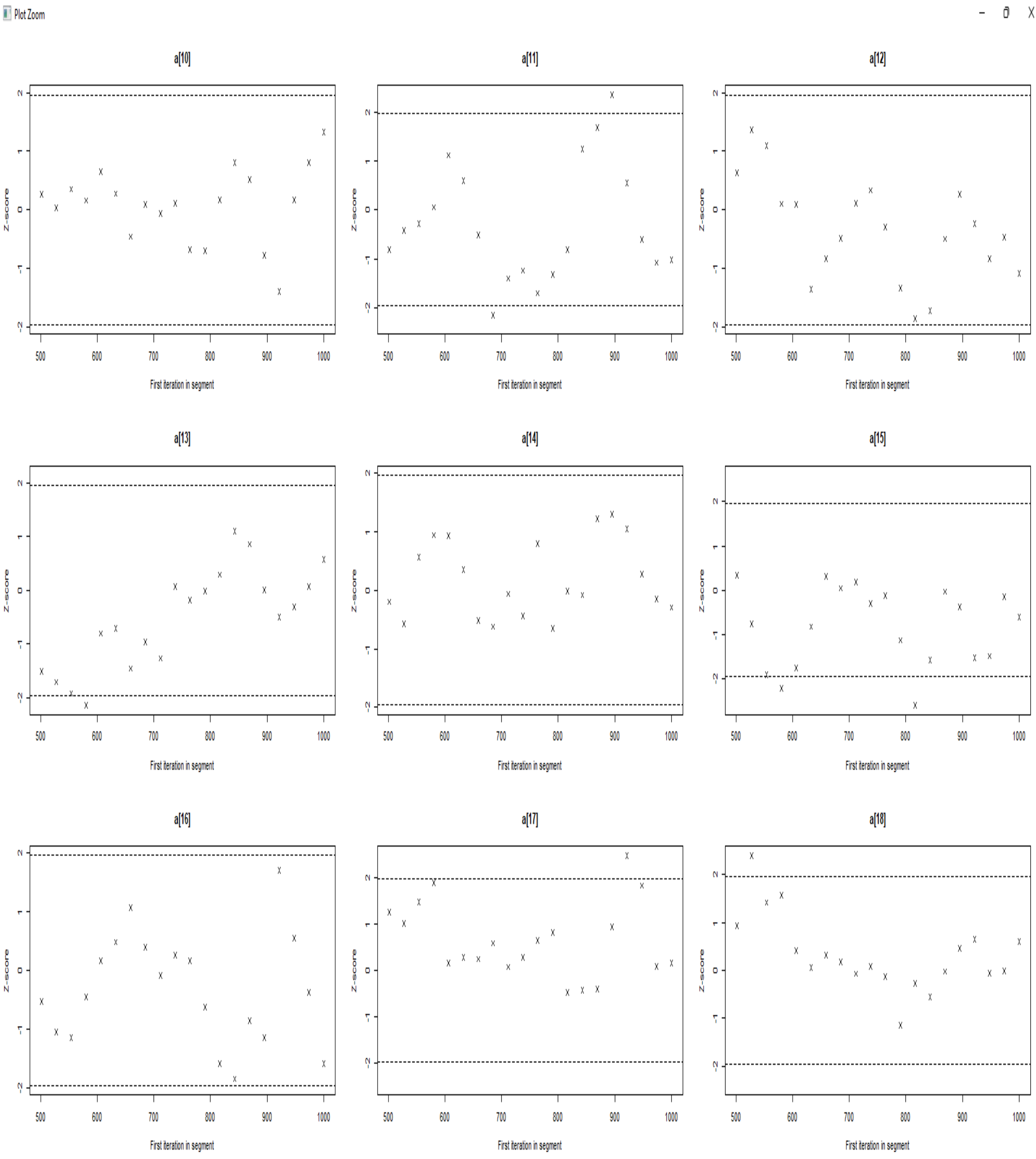
Geweke (1992) suggested a diagnostic test for checking the convergence of the mean of each parameter separately from the sampled values of a single chain. To construct this test, he proposed viewing the set of simulated values, obtained by the MCMC output, as a time series. This diagnostic applies a simple Z test to check whether the means estimated from two different subsamples of the total MCMC output are equal. These subsamples refer to observations coming from the beginning and the end of the generated chain. Hence, if the means at the beginning and the end of the total MCMC output are rejected, then convergence of the chain cannot be assumed. Coda compares by default the initial 10% and the last 50% of the total iterations. Since Z asymptotically follows the standardized normal distribution, values that lie in its tails provide an indication of nonconvergence. To be more specific, parameters with |Z| > 2 indicate differences in the means between the first and the last set of iterations and hence nonconvergence. Nevertheless, because of the type I error of classical significance tests, in multiparameter models, we allow 5% of the calculated Zs to lie outside this range. Let us have a look at the output in R:

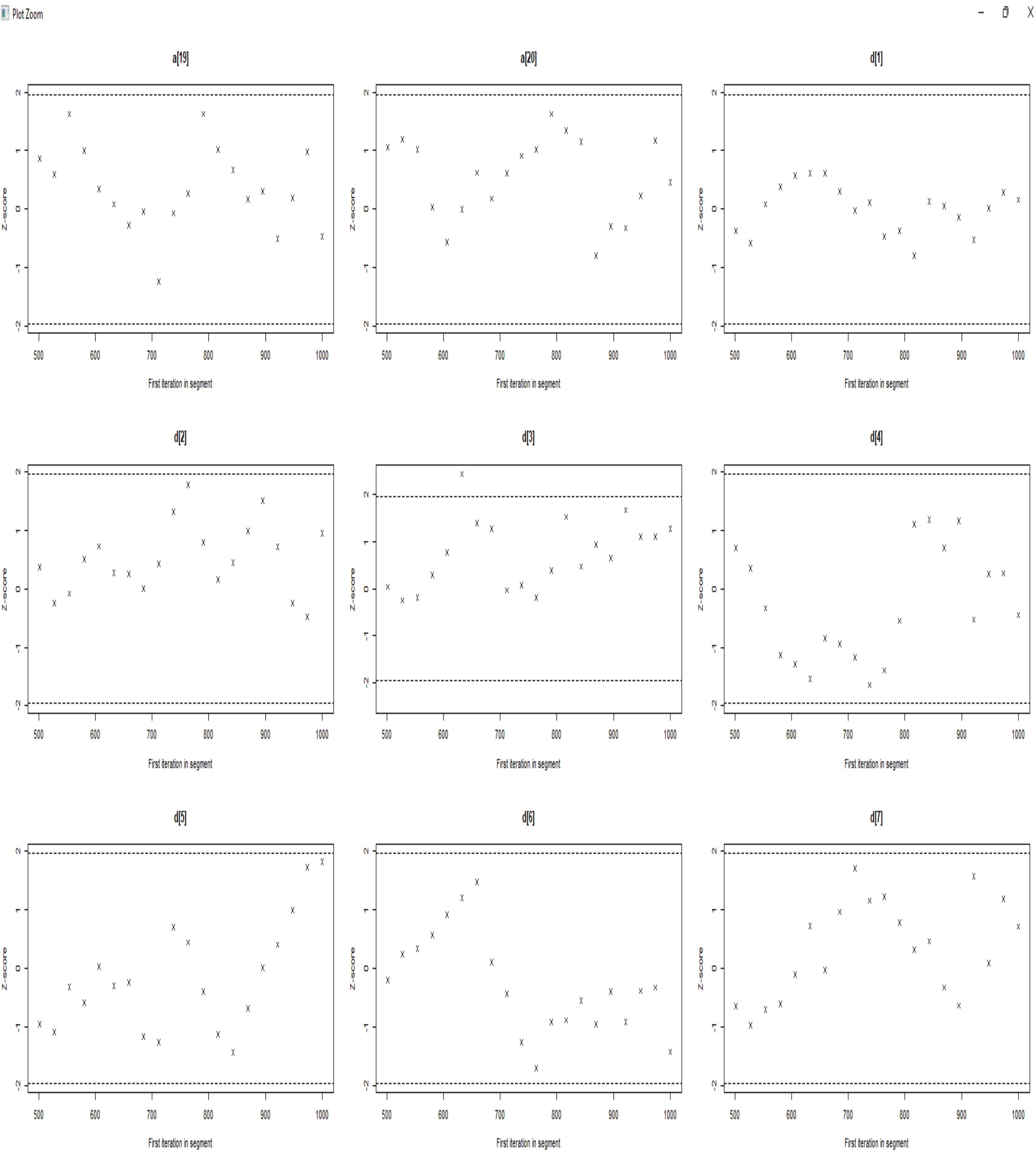


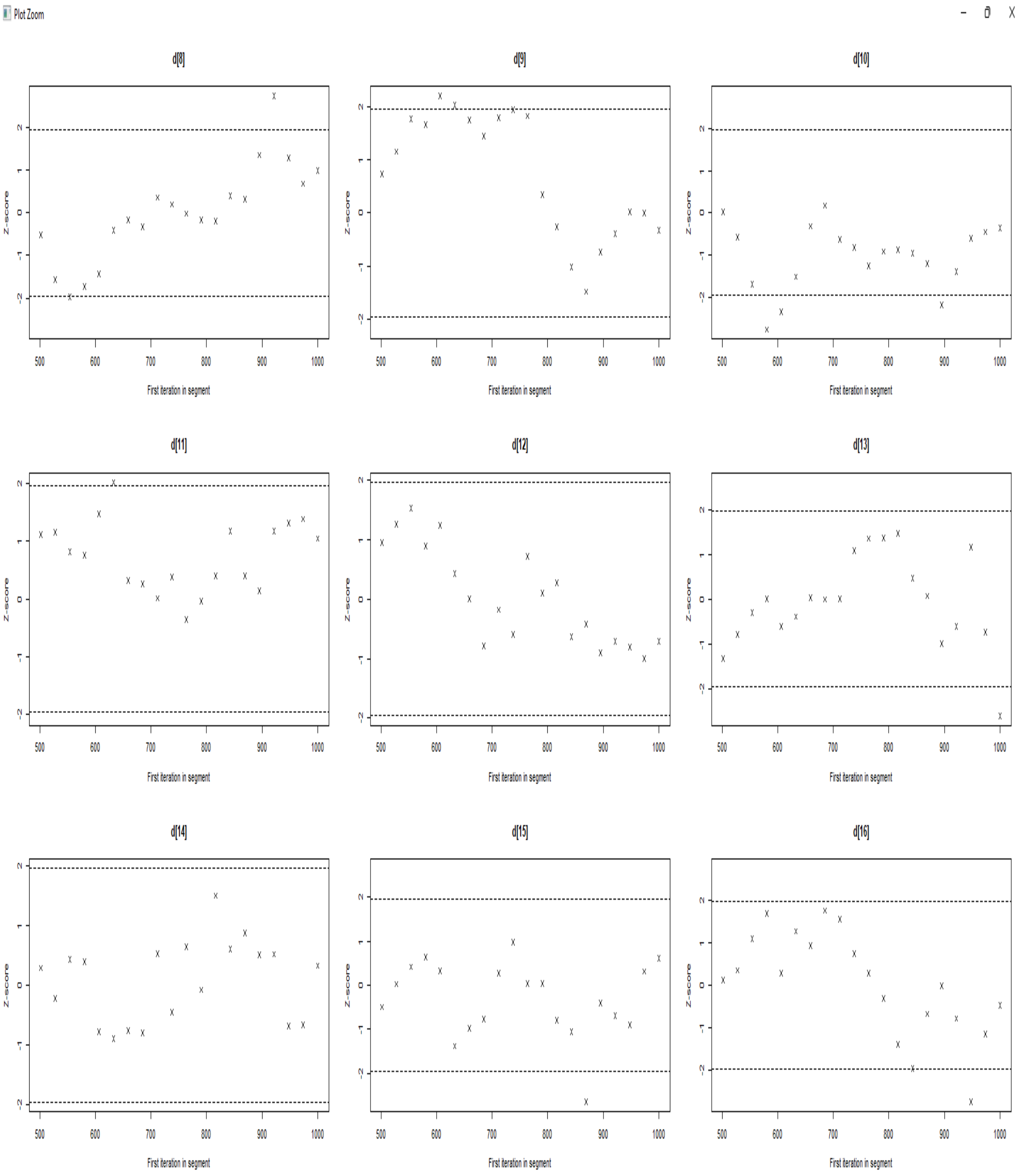
```
GEWEKE CONVERGENCE DIAGNOSTIC (Z-score)
=======================================

Iterations used = 501:1500
Thinning interval = 1
Sample size per chain = 1000

$`2017_UK_coda_out.txt`

Fraction in 1st window = 0.1
Fraction in 2nd window = 0.5

   a[1]    a[2]    a[3]    a[4]    a[5]    a[6]    a[7]    a[8]    a[9]
 0.0362  0.0118  0.1995 -0.3501  0.2361 -0.2057 -0.4166 -0.7204  0.5936
  a[10]   a[11]   a[12]   a[13]   a[14]   a[15]   a[16]   a[17]   a[18]
 0.2607 -0.8063  0.6256 -1.5064 -0.1997  0.3315 -0.5331  1.2536  0.9257
  a[19]   a[20]    d[1]    d[2]    d[3]    d[4]    d[5]    d[6]    d[7]
 0.8682  1.0528 -0.3705  0.3750  0.0376  0.7031 -0.9553 -0.2039 -0.6399
   d[8]    d[9]   d[10]   d[11]   d[12]   d[13]   d[14]   d[15]   d[16]
-0.5169  0.7395  0.0281  1.1150  0.9523 -1.3337  0.2910 -0.4974  0.1309
  d[17]   d[18]   d[19]   d[20]    home      mu
 1.2836 -0.5442 -1.7101  0.3631  0.8994 -1.0957


Geweke plots menu

1: Change window size
2: Plot Z-scores
3: Change number of bins for plot
4: Return to Diagnostics Menu
```

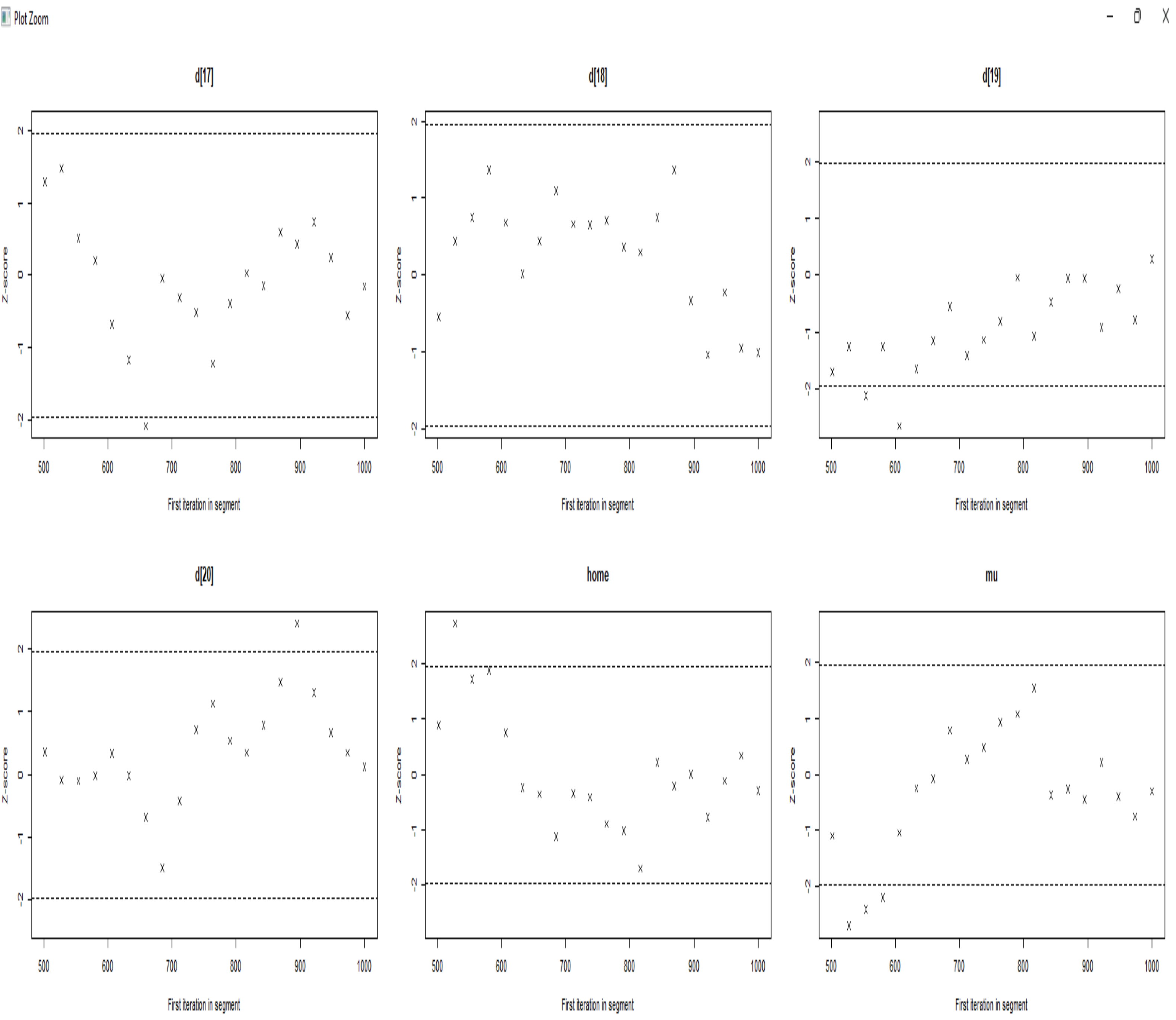
We observe that all values lie within the range of [-2,2] hence we can assume convergence! Let us have a look at the plotted Z scores as well:




Plot Zoom
a[10]
a[11]
a[12]
a[13]
a[14]
a[15]
a[16]
a[17]
a[18]
Z-score
First iteration in segment
500
600
700
800
900
1000
-2
-1
0
1
2


Plot Zoom
a[19]
a[20]
d[1]
d[2]
d[3]
d[4]
d[5]
d[6]
d[7]
Z-score
First iteration in segment
500
600
700
800
900
1000
-2
-1
0
1
2

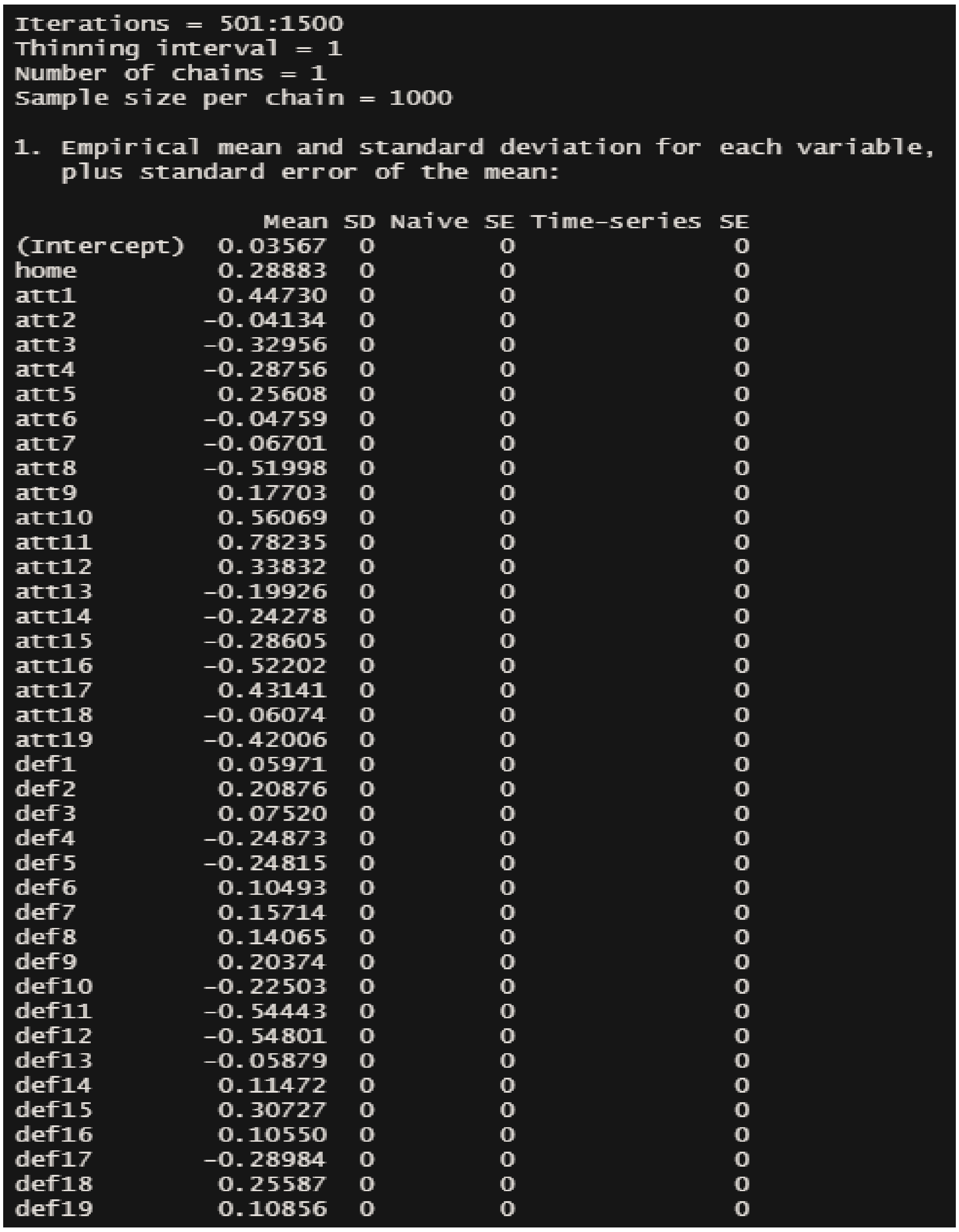

Plot Zoom
d[8]
d[9]
d[10]
d[11]
d[12]
d[13]
d[14]
d[15]
d[16]
Z-score
First iteration in segment
500
600
700
800
900
1000
-2
-1
0
1
2



We observe that all values lie within the range of [-2,2] (allowing 5% of the calculated Zs to lie outside this range) hence we can assume once again convergence! For more information, see: Ntzoufras, (2009) "Bayesian modeling using WinBUGS".

MCMC package output

We can have a look at the output of the MCMC package in RStudio as well:



```
Iterations = 501:1500
Thinning interval = 1
Number of chains = 1
Sample size per chain = 1000

1. Empirical mean and standard deviation for each variable,
   plus standard error of the mean:

                 Mean SD Naive SE Time-series SE
(Intercept)  0.03567  0        0              0
home         0.28883  0        0              0
att1         0.44730  0        0              0
att2        -0.04134  0        0              0
att3        -0.32956  0        0              0
att4        -0.28756  0        0              0
att5         0.25608  0        0              0
att6        -0.04759  0        0              0
att7        -0.06701  0        0              0
att8        -0.51998  0        0              0
att9         0.17703  0        0              0
att10        0.56069  0        0              0
att11        0.78235  0        0              0
att12        0.33832  0        0              0
att13       -0.19926  0        0              0
att14       -0.24278  0        0              0
att15       -0.28605  0        0              0
att16       -0.52202  0        0              0
att17        0.43141  0        0              0
att18       -0.06074  0        0              0
att19       -0.42006  0        0              0
def1         0.05971  0        0              0
def2         0.20876  0        0              0
def3         0.07520  0        0              0
def4        -0.24873  0        0              0
def5        -0.24815  0        0              0
def6         0.10493  0        0              0
def7         0.15714  0        0              0
def8         0.14065  0        0              0
def9         0.20374  0        0              0
def10       -0.22503  0        0              0
def11       -0.54443  0        0              0
def12       -0.54801  0        0              0
def13       -0.05879  0        0              0
def14        0.11472  0        0              0
def15        0.30727  0        0              0
def16        0.10550  0        0              0
def17       -0.28984  0        0              0
def18        0.25587  0        0              0
def19        0.10856  0        0              0
```

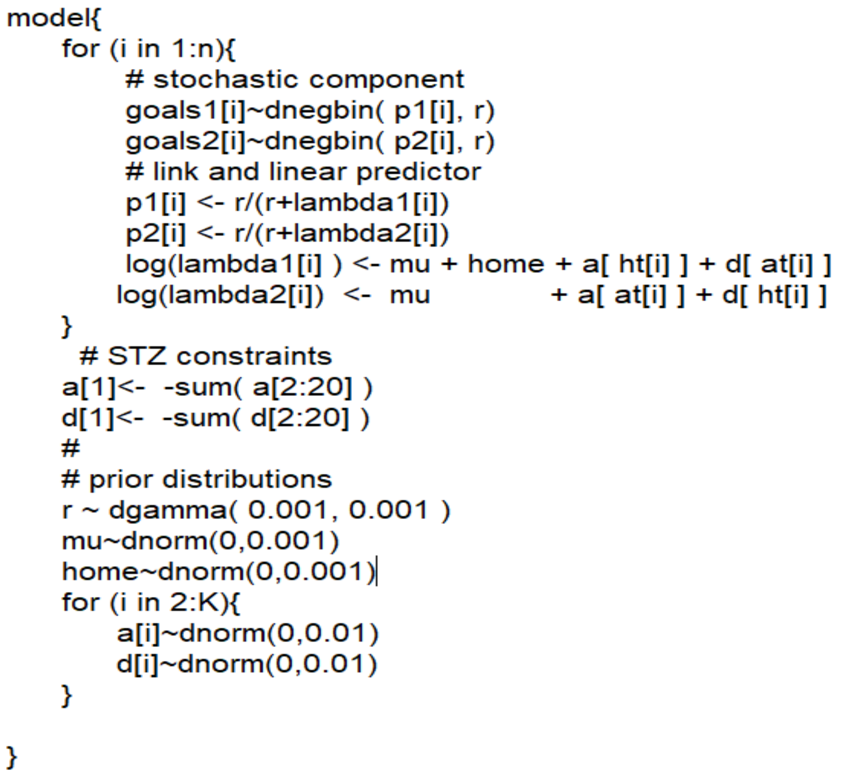
We observe that the results produced are quite similar to those obtained after using OpenBUGS.

Basic football modeling assumptions

As regards the Poisson regression model adopted for the goals scored by each team, it conceals two important assumptions that are questionable in the sports modeling literature: the independence between home and away goals and the equality between the mean and the variance. However, that is not the case in terms of the football data, as there’s observed overdispersion. Furthermore, small dependence (yet significant) has been observed and therefore should be added. Possible remedies include using:

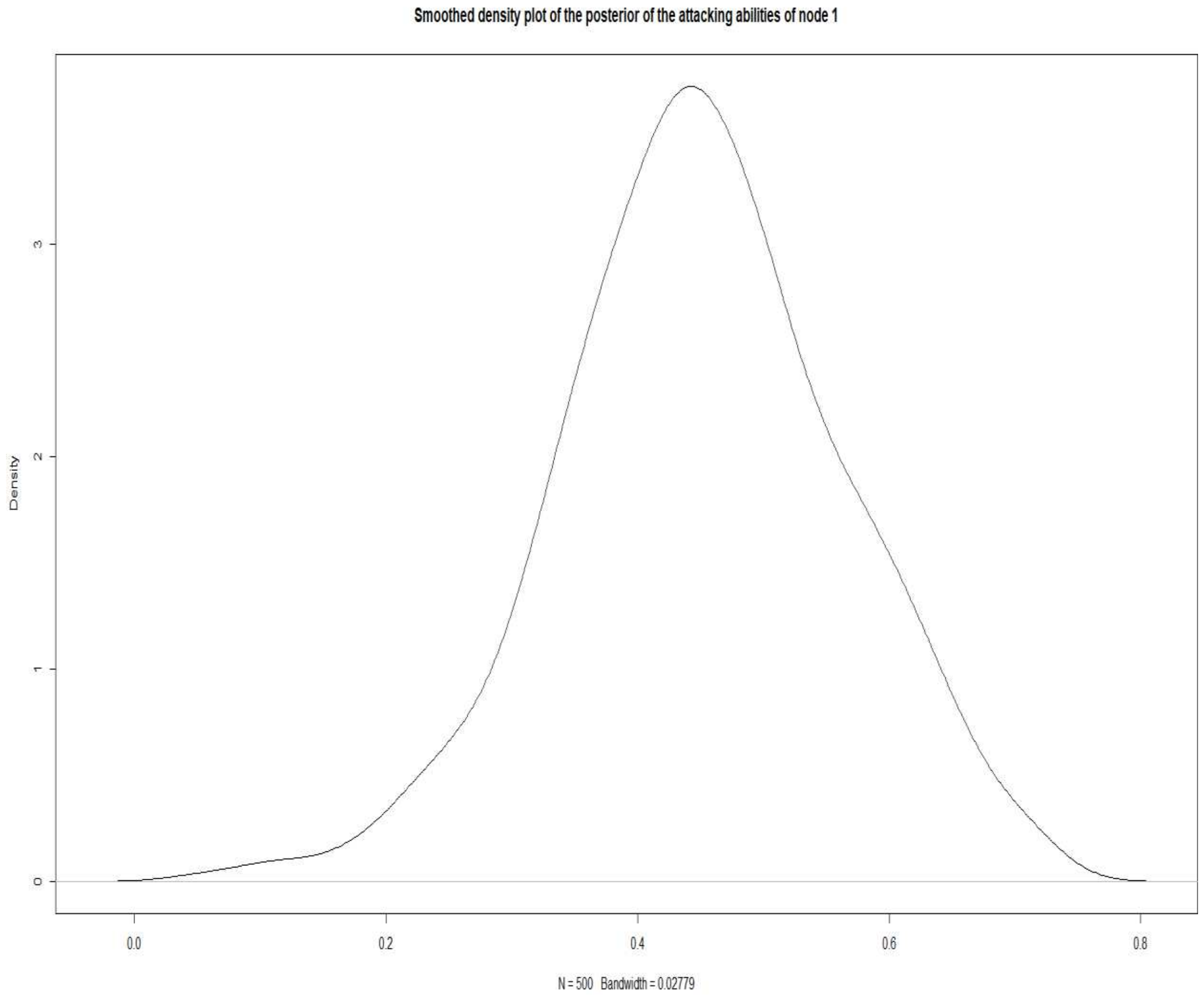
- Bivariate Poisson model (Karlis and Ntzoufras, 2003, JRSSD)
- Poisson Difference/Skellam distribution (Karlis and Ntzoufras, 2008)
- Random effects models (normal random effects do not work very well for football data as we see later on)
- Copula models

To account for the overdispersion a possible remedy is using the Negative Binomial distribution assumption. The Negative Binomial model can be depicted below:

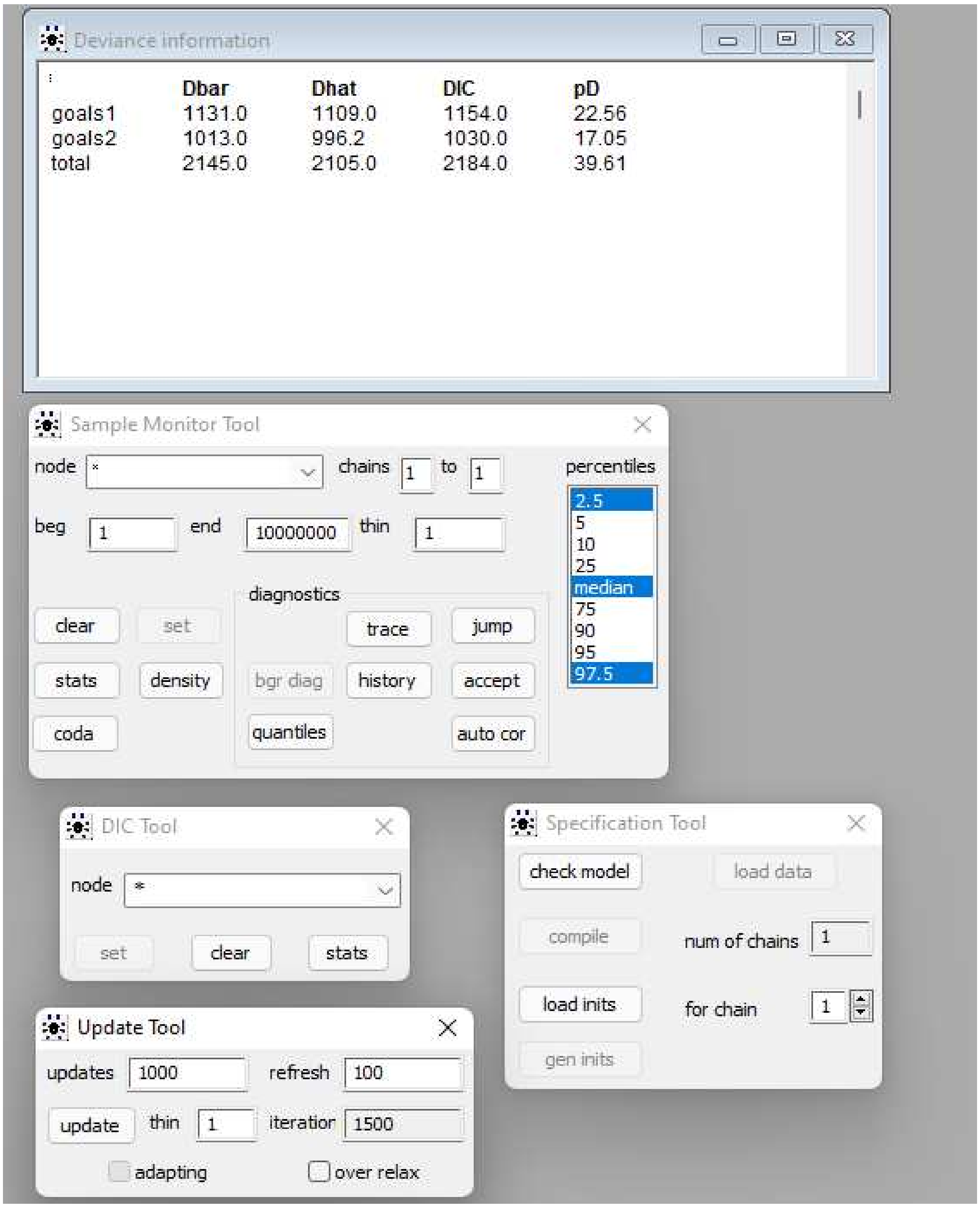


```
model{
    for (i in 1:n){
        # stochastic component
        goals1[i]~dnegbin( p1[i], r)
        goals2[i]~dnegbin( p2[i], r)
        # link and linear predictor
        p1[i] <- r/(r+lambda1[i])
        p2[i] <- r/(r+lambda2[i])
        log(lambda1[i] ) <- mu + home + a[ ht[i] ] + d[ at[i] ]
       log(lambda2[i])  <-  mu           + a[ at[i] ] + d[ ht[i] ]
    }
     # STZ constraints
    a[1]<-  -sum( a[2:20] )
    d[1]<-  -sum( d[2:20] )
    #
    # prior distributions
    r ~ dgamma( 0.001, 0.001 )
    mu~dnorm(0,0.001)
    home~dnorm(0,0.001)
    for (i in 2:K){
        a[i]~dnorm(0,0.01)
        d[i]~dnorm(0,0.01)
    }

}
```

Should we want to compare the two proposed models above (the Vanilla model and the Negative Binomial model), we can use the Deviance Information Criterion (DIC). The DIC can be considered as the Bayesian analog of Akaike's Information Criterion (AIC). Its justification is similar to that for AIC, but the expectations used in its derivation are now with respect to parameters instead of sampling distributions used in the original derivation of the latter. The importance of DIC is that it can be directly calculated from an MCMC output and can be applied in a much wider variety of models, including hierarchical and latent variable models, where the number of estimated parameters is unclear. On the whole, DIC must be used with caution since it assumes that the posterior mean can be used as a "good" summary of central location for description of the posterior distribution. Problems have been reported in cases where posterior distributions are not symmetrical or unimodal. In our case however, that does not apply as we can observe below:



Thus, we can compare the two proposed models via DIC. The DIC value can be obtained directly by using R (R2winBUGS package), or it can be calculated manually in OpenBUGS. Via OpenBUGS, the DIC value for the Vanilla model is depicted below:



```
Deviance information
          Dbar     Dhat     DIC      pD
goals1    1131.0   1109.0   1154.0   22.56
goals2    1013.0   996.2    1030.0   17.05
total     2145.0   2105.0   2184.0   39.61
```

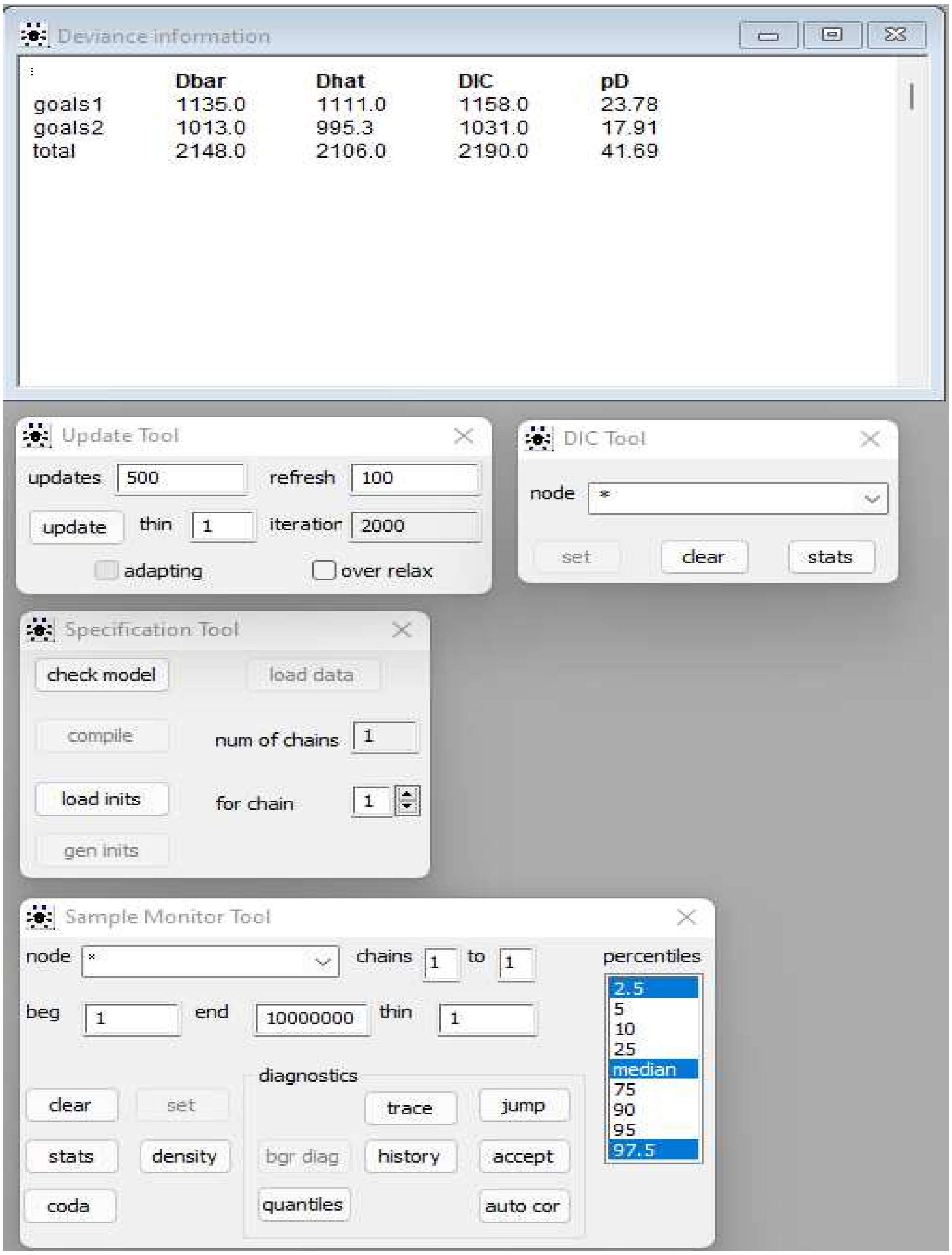
Sample Monitor Tool: node *, chains 1 to 1, beg 1, end 10000000, thin 1, percentiles 2.5, median, 97.5

DIC Tool: node *

Update Tool: updates 1000, refresh 100, thin 1, iteration 1500

Specification Tool: num of chains 1, for chain 1

Now, as regards the Negative Binomial model the DIC value is as follows:



```
            Dbar     Dhat     DIC      pD
goals1      1135.0   1111.0   1158.0   23.78
goals2      1013.0   995.3    1031.0   17.91
total       2148.0   2106.0   2190.0   41.69
```

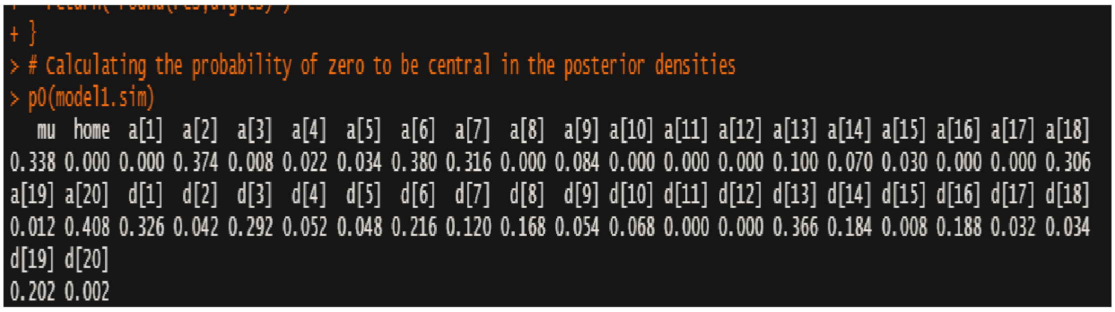
updates 500, refresh 100, thin 1, iteration 2000

DIC Tool node *

num of chains 1, for chain 1

Sample Monitor Tool: node *, chains 1 to 1, beg 1, end 10000000, thin 1, percentiles 2.5, median, 97.5

We observe that the difference in the DIC values of the above two models is less than 5 units so we would report both of those. Now let’s try obtaining the DIC values via the R2winBUGS package. By observing the posterior summaries above we notice that in the Poisson log normal model, the DIC is valued at 2183 while in the Negative Binomial model, the DIC is valued at 2176.3! The model with the minimum DIC is expected to give better short-term predictions in the same logic as AIC. In this case, it seems that the Negative Binomial model performs slightly better than the Vanilla model (which makes sense as the Negative Binomial model accounts for the overdispersion that is observed)! As regards the effect of the covariates on the target variable, we could check whether the posterior distributions lie away from zero, hence implying that the covariates have an important effect on our response variable (the goals scored). Such analysis offers a tool for tracing important variables. Within this analysis, we can calculate the posterior probability p0, which indicates that when zero lies at the center of the distribution, hence p0=1/2, there’s no clear positive or negative effect of the covariate on the target variable. Let’s view the results for the Vanilla model:

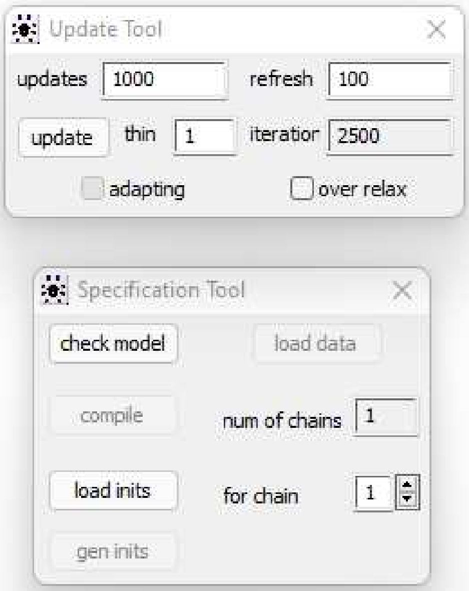


```
+ }
> # Calculating the probability of zero to be central in the posterior densities
> p0(model1.sim)
   mu  home  a[1]  a[2]  a[3]  a[4]  a[5]  a[6]  a[7]  a[8]  a[9] a[10] a[11] a[12] a[13] a[14] a[15] a[16] a[17] a[18]
0.338 0.000 0.000 0.374 0.008 0.022 0.034 0.380 0.316 0.000 0.084 0.000 0.000 0.000 0.100 0.070 0.030 0.000 0.000 0.306
a[19] a[20]  d[1]  d[2]  d[3]  d[4]  d[5]  d[6]  d[7]  d[8]  d[9] d[10] d[11] d[12] d[13] d[14] d[15] d[16] d[17] d[18]
0.012 0.408 0.326 0.042 0.292 0.052 0.048 0.216 0.120 0.168 0.054 0.068 0.000 0.000 0.366 0.184 0.008 0.188 0.032 0.034
d[19] d[20]
0.202 0.002
```

However, p0 ignores the collinearities of the covariates, that being the case in our study, so we won’t make use of it!

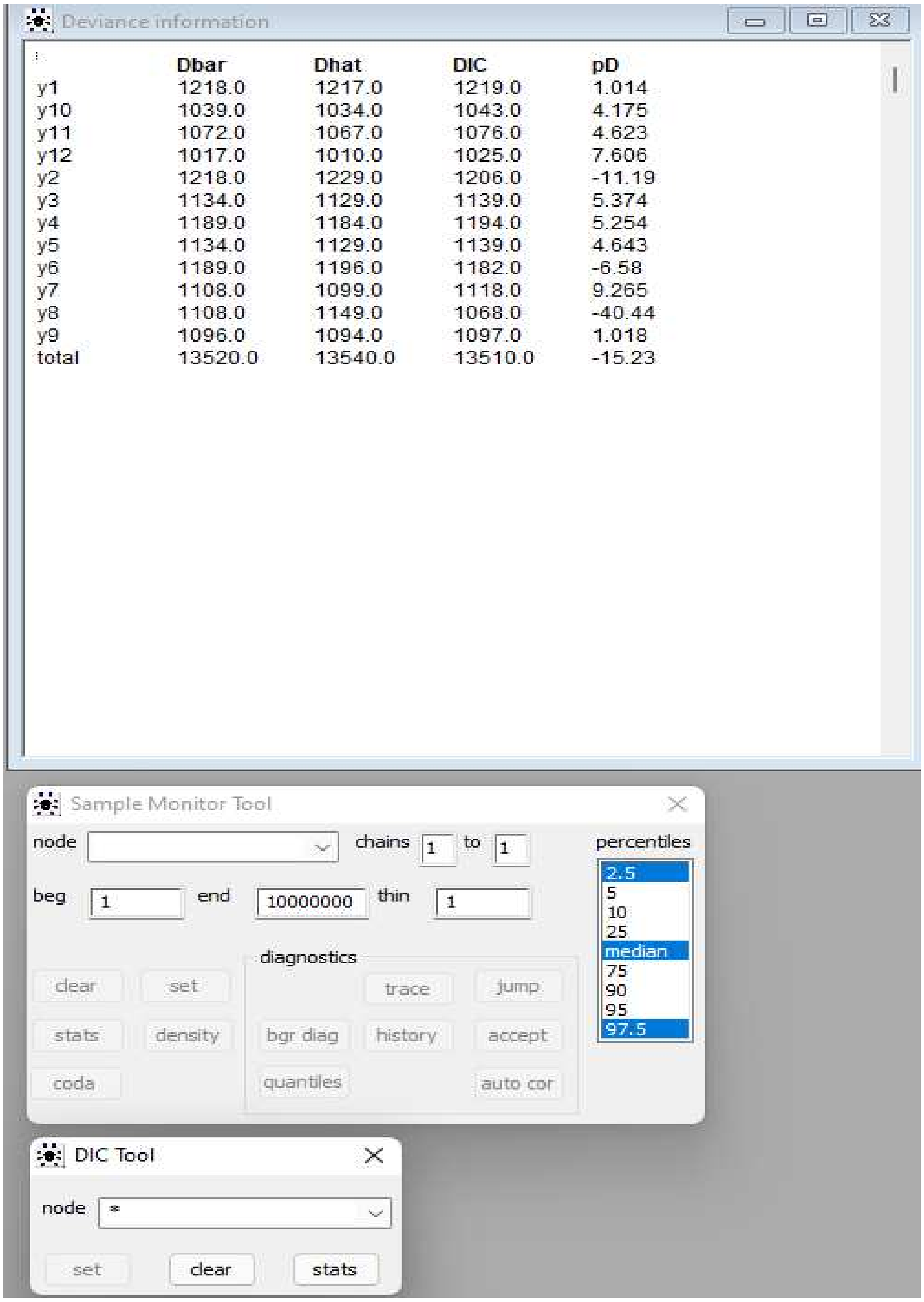
Variable selection via DIC in the Vanilla model

This procedure can be set up within OpenBUGS in a relatively straightforward manner. We only need to define the response variable of each model in a different vector in order to obtain separate DIC values for all fitted models in each cycle of the procedure. The use of the binary vectors (gamma1,gamma2,gamma3 in our case) simplifies the implementation of the procedure in the OpenBUGS. We only need to set up the current model along with the binary indicators and then run the associated OpenBUGS code to identify the best model (gamma optimal) in each cycle of the proposed procedure. The code described above can be depicted below:



```
model{ # Vectors used for the calculation of DIC
 for ( i in 1:n ) {
     y1[i] <- goals1[i]
     y2[i] <- goals1[i]
     y3[i] <- goals1[i]
     y4[i] <- goals1[i]
     y5[i] <- goals1[i]
     y6[i] <- goals1[i]
     y7[i] <- goals1[i]
     y8[i] <- goals1[i]
     y9[i] <- goals2[i]
     y10[i] <- goals2[i]
     y11[i] <- goals2[i]
     y12[i] <- goals2[i]
     y1[i]~dpois( lambda1[i,1] )
     y2[i]~dpois( lambda1[i,2] )
     y3[i]~dpois( lambda1[i,3] )
     y4[i]~dpois( lambda1[i,4] )
     y5[i]~dpois( lambda1[i,5] )
     y6[i]~dpois( lambda1[i,6] )
     y7[i]~dpois( lambda1[i,7] )
     y8[i]~dpois( lambda1[i,8] )
     y9[i]~dpois( lambda2[i,9] )
     y10[i]~dpois( lambda2[i,10] )
     y11[i]~dpois( lambda2[i,11] )
     y12[i]~dpois( lambda2[i,12] )
  }

  # Poisson model likelihood
  for ( k in 1:12 ) {
     for ( i in 1:n ) {
          log( lambda1[i,k] ) <- beta[1,k] + gamma1[k]*beta[2,k]*home + gamma2[k]*beta[3,k]*a[ ht[i] ] +
gamma3[k]*beta[4,k]*d[ at[i] ]
          log( lambda2[i,k] ) <- beta[1,k] + gamma2[k]*beta[3,k]*a[ at[i] ] + gamma3[k]*beta[4,k]*d[ ht[i] ]
     }}

  # STZ constraints
       a[1]<-  -sum( a[2:20] )
       d[1]<-  -sum( d[2:20] )

 # Priors
   home~dnorm(0,0.001)
   for ( k in 1:12) {
      for ( j in 1:4) {
            beta[j,k]~dnorm(0,0.001)
   }}
   for (i in 2:K){
               a[i]~dnorm(0,0.01)
               d[i]~dnorm(0,0.01)
   }
}


# Inits
list( beta=structure( .Data=c(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ), .Dim = c(4,
12) ) , a=c(NA, 0,0,0,0,0, 0,0,0,0,0, 0,0,0,0,0, 0,0,0,0) , d=c(NA, 0,0,0,0,0, 0,0,0,0,0, 0,0,0,0,0, 0,0,0,0) , home=0.5 )
```

Running 2000 iterations (500 discarded) we obtain the following DIC values:



```
Deviance information

        Dbar     Dhat     DIC      pD
y1      1218.0   1217.0   1219.0   1.014
y10     1039.0   1034.0   1043.0   4.175
y11     1072.0   1067.0   1076.0   4.623
y12     1017.0   1010.0   1025.0   7.606
y2      1218.0   1229.0   1206.0   -11.19
y3      1134.0   1129.0   1139.0   5.374
y4      1189.0   1184.0   1194.0   5.254
y5      1134.0   1129.0   1139.0   4.643
y6      1189.0   1196.0   1182.0   -6.58
y7      1108.0   1099.0   1118.0   9.265
y8      1108.0   1149.0   1068.0   -40.44
y9      1096.0   1094.0   1097.0   1.018
total   13520.0  13540.0  13510.0  -15.23
```

Sample Monitor Tool

node: (empty) chains 1 to 1
beg 1 end 10000000 thin 1
percentiles: [✓ 2.5], 5, 10, 25, [✓ median], 75, 90, 95, [✓ 97.5]

DIC Tool

node: *

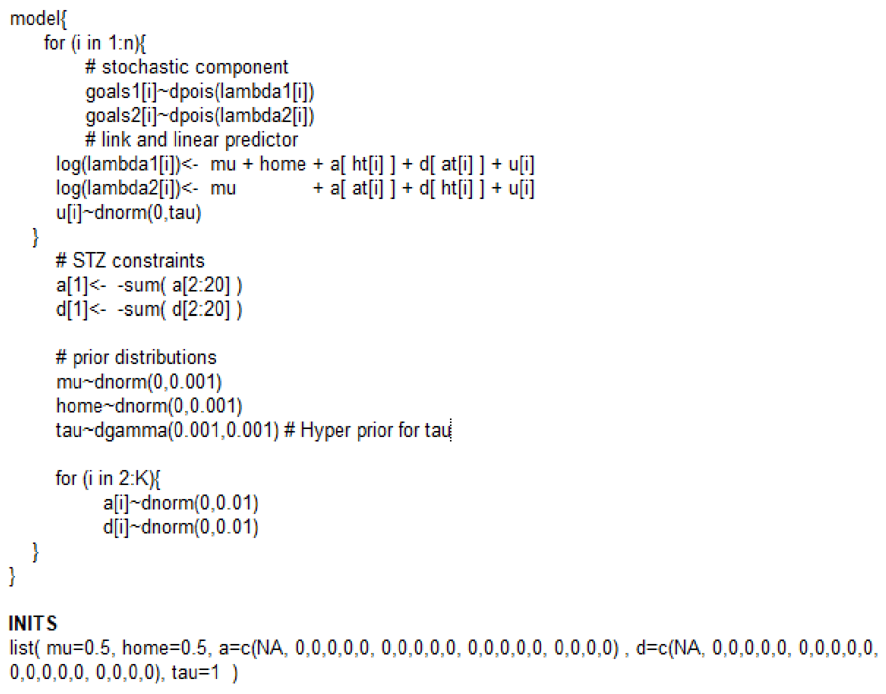
The first 8 (2^3=8) models (y1,…,y8) correspond to all the possible combinations of the covariates (betas) with the binary indicators (gammas) for the log(lambda1), which refers to the goals scored by the home team. The next 4 (2^2) models (y9,…,y12) correspond to all the possible combinations of the covariates (betas) with the binary indicators (gammas) for the log(lambda2), which refers to the goals scored by the away team. The results above indicate that y8 has the lowest DIC value, model which represents the full model for the log(lambda1). As regards log(lambda2), the lowest DIC value is observed at y12, model which consists of the all the possible covariates as well. Hence, we gather that the full models are the best according to DIC.

Random effects in football data

A random effect is an additional random error added in the linear predictor. Parameters in the classical statistics are not constants but random variables. In Bayesian analysis, an additional hierarchical level is added. In our case we can try implementing a random effects model since random effects can also be used when:

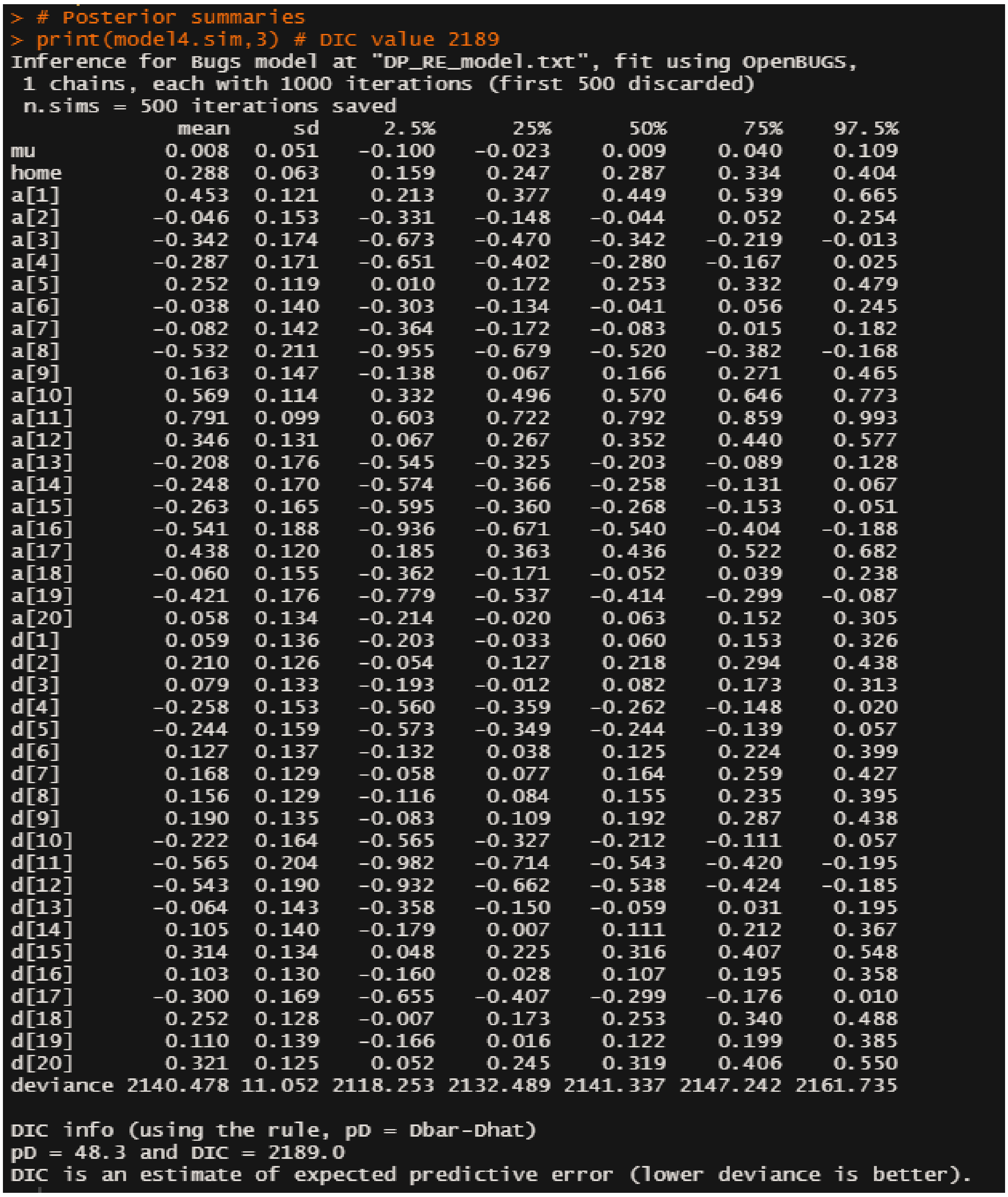
- We want to correct for over-dispersion.
- Introduce correlation between measurements.
- There are repeated measures over the same subjects at the same time.

The random effects model has been implemented in the code below:



```
model{
    for (i in 1:n){
         # stochastic component
         goals1[i]~dpois(lambda1[i])
         goals2[i]~dpois(lambda2[i])
         # link and linear predictor
     log(lambda1[i])<-  mu + home + a[ ht[i] ] + d[ at[i] ] + u[i]
     log(lambda2[i])<-  mu            + a[ at[i] ] + d[ ht[i] ] + u[i]
     u[i]~dnorm(0,tau)
   }
     # STZ constraints
     a[1]<-  -sum( a[2:20] )
     d[1]<-  -sum( d[2:20] )

     # prior distributions
     mu~dnorm(0,0.001)
     home~dnorm(0,0.001)
     tau~dgamma(0.001,0.001) # Hyper prior for tau

     for (i in 2:K){
           a[i]~dnorm(0,0.01)
           d[i]~dnorm(0,0.01)
   }
}

INITS
list( mu=0.5, home=0.5, a=c(NA, 0,0,0,0,0, 0,0,0,0,0, 0,0,0,0,0, 0,0,0,0) , d=c(NA, 0,0,0,0,0, 0,0,0,0,0,
0,0,0,0,0, 0,0,0,0), tau=1  )
```

Let’s view the posterior summaries from RStudio:

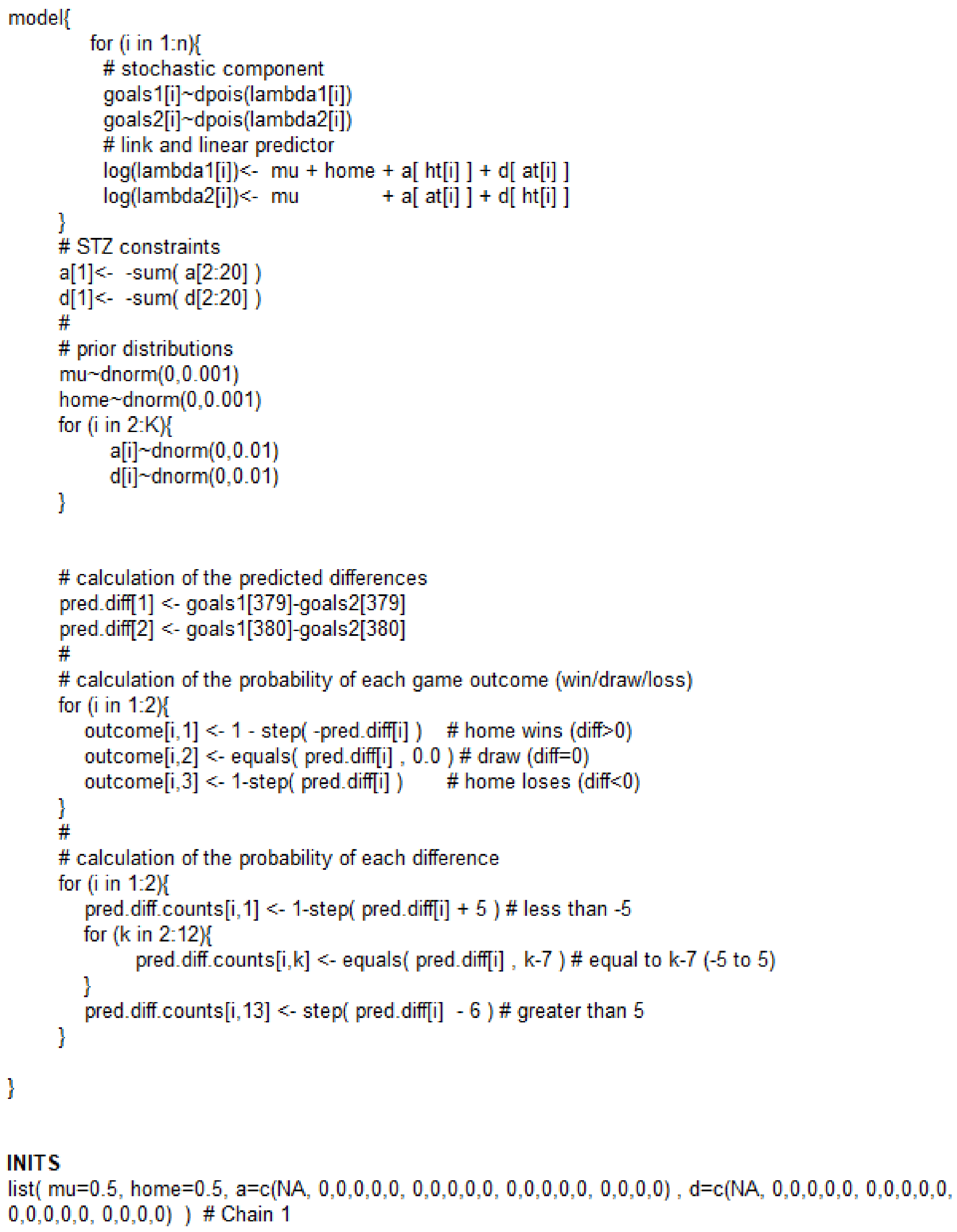


```
> # Posterior summaries
> print(model4.sim,3) # DIC value 2189
Inference for Bugs model at "DP_RE_model.txt", fit using OpenBUGS,
 1 chains, each with 1000 iterations (first 500 discarded)
 n.sims = 500 iterations saved
             mean     sd     2.5%      25%      50%      75%    97.5%
mu          0.008  0.051   -0.100   -0.023    0.009    0.040    0.109
home        0.288  0.063    0.159    0.247    0.287    0.334    0.404
a[1]        0.453  0.121    0.213    0.377    0.449    0.539    0.665
a[2]       -0.046  0.153   -0.331   -0.148   -0.044    0.052    0.254
a[3]       -0.342  0.174   -0.673   -0.470   -0.342   -0.219   -0.013
a[4]       -0.287  0.171   -0.651   -0.402   -0.280   -0.167    0.025
a[5]        0.252  0.119    0.010    0.172    0.253    0.332    0.479
a[6]       -0.038  0.140   -0.303   -0.134   -0.041    0.056    0.245
a[7]       -0.082  0.142   -0.364   -0.172   -0.083    0.015    0.182
a[8]       -0.532  0.211   -0.955   -0.679   -0.520   -0.382   -0.168
a[9]        0.163  0.147   -0.138    0.067    0.166    0.271    0.465
a[10]       0.569  0.114    0.332    0.496    0.570    0.646    0.773
a[11]       0.791  0.099    0.603    0.722    0.792    0.859    0.993
a[12]       0.346  0.131    0.067    0.267    0.352    0.440    0.577
a[13]      -0.208  0.176   -0.545   -0.325   -0.203   -0.089    0.128
a[14]      -0.248  0.170   -0.574   -0.366   -0.258   -0.131    0.067
a[15]      -0.263  0.165   -0.595   -0.360   -0.268   -0.153    0.051
a[16]      -0.541  0.188   -0.936   -0.671   -0.540   -0.404   -0.188
a[17]       0.438  0.120    0.185    0.363    0.436    0.522    0.682
a[18]      -0.060  0.155   -0.362   -0.171   -0.052    0.039    0.238
a[19]      -0.421  0.176   -0.779   -0.537   -0.414   -0.299   -0.087
a[20]       0.058  0.134   -0.214   -0.020    0.063    0.152    0.305
d[1]        0.059  0.136   -0.203   -0.033    0.060    0.153    0.326
d[2]        0.210  0.126   -0.054    0.127    0.218    0.294    0.438
d[3]        0.079  0.133   -0.193   -0.012    0.082    0.173    0.313
d[4]       -0.258  0.153   -0.560   -0.359   -0.262   -0.148    0.020
d[5]       -0.244  0.159   -0.573   -0.349   -0.244   -0.139    0.057
d[6]        0.127  0.137   -0.132    0.038    0.125    0.224    0.399
d[7]        0.168  0.129   -0.058    0.077    0.164    0.259    0.427
d[8]        0.156  0.129   -0.116    0.084    0.155    0.235    0.395
d[9]        0.190  0.135   -0.083    0.109    0.192    0.287    0.438
d[10]      -0.222  0.164   -0.565   -0.327   -0.212   -0.111    0.057
d[11]      -0.565  0.204   -0.982   -0.714   -0.543   -0.420   -0.195
d[12]      -0.543  0.190   -0.932   -0.662   -0.538   -0.424   -0.185
d[13]      -0.064  0.143   -0.358   -0.150   -0.059    0.031    0.195
d[14]       0.105  0.140   -0.179    0.007    0.111    0.212    0.367
d[15]       0.314  0.134    0.048    0.225    0.316    0.407    0.548
d[16]       0.103  0.130   -0.160    0.028    0.107    0.195    0.358
d[17]      -0.300  0.169   -0.655   -0.407   -0.299   -0.176    0.010
d[18]       0.252  0.128   -0.007    0.173    0.253    0.340    0.488
d[19]       0.110  0.139   -0.166    0.016    0.122    0.199    0.385
d[20]       0.321  0.125    0.052    0.245    0.319    0.406    0.550
deviance 2140.478 11.052 2118.253 2132.489 2141.337 2147.242 2161.735

DIC info (using the rule, pD = Dbar-Dhat)
pD = 48.3 and DIC = 2189.0
DIC is an estimate of expected predictive error (lower deviance is better).
```

The DIC for the model with the fixed effects was estimated 2183 while for the random effects model it was estimated 2189, hence we can gather that the random effects model does not perform better in this case and the fixed effects model will perform better in short term predictions.

Prediction of the last 2 matches in the Premier league 2017-2018 using the Vanilla model (via OpenBUGS)

The syntax of the code used to predict the outcome and the corresponding probabilities can be viewed in the picture below:

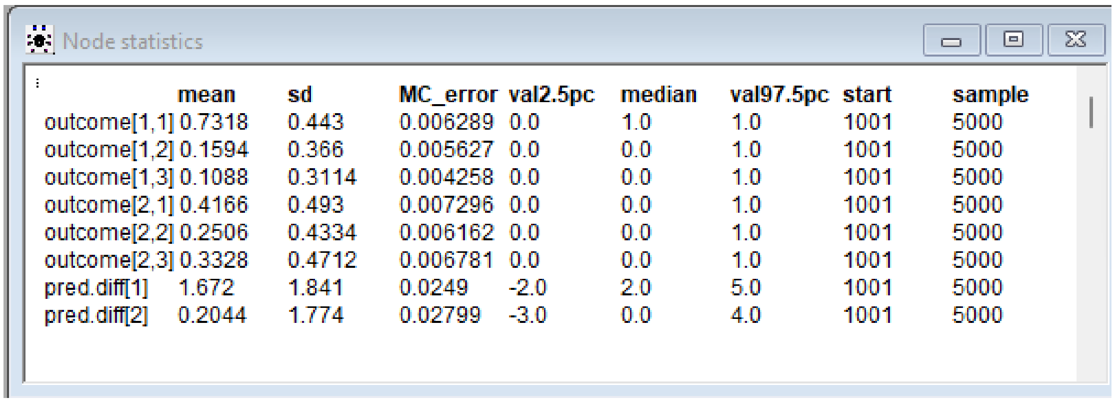


```
model{
        for (i in 1:n){
          # stochastic component
          goals1[i]~dpois(lambda1[i])
          goals2[i]~dpois(lambda2[i])
          # link and linear predictor
          log(lambda1[i])<-  mu + home + a[ ht[i] ] + d[ at[i] ]
          log(lambda2[i])<-  mu        + a[ at[i] ] + d[ ht[i] ]
     }
     # STZ constraints
     a[1]<-  -sum( a[2:20] )
     d[1]<-  -sum( d[2:20] )
     #
     # prior distributions
     mu~dnorm(0,0.001)
     home~dnorm(0,0.001)
     for (i in 2:K){
          a[i]~dnorm(0,0.01)
          d[i]~dnorm(0,0.01)
     }


     # calculation of the predicted differences
     pred.diff[1] <- goals1[379]-goals2[379]
     pred.diff[2] <- goals1[380]-goals2[380]
     #
     # calculation of the probability of each game outcome (win/draw/loss)
     for (i in 1:2){
        outcome[i,1] <- 1 - step( -pred.diff[i] )    # home wins (diff>0)
        outcome[i,2] <- equals( pred.diff[i] , 0.0 ) # draw (diff=0)
        outcome[i,3] <- 1-step( pred.diff[i] )       # home loses (diff<0)
     }
     #
     # calculation of the probability of each difference
     for (i in 1:2){
        pred.diff.counts[i,1] <- 1-step( pred.diff[i] + 5 ) # less than -5
        for (k in 2:12){
              pred.diff.counts[i,k] <- equals( pred.diff[i] , k-7 ) # equal to k-7 (-5 to 5)
        }
        pred.diff.counts[i,13] <- step( pred.diff[i]  - 6 ) # greater than 5
     }

}


INITS
list( mu=0.5, home=0.5, a=c(NA, 0,0,0,0,0, 0,0,0,0,0, 0,0,0,0,0, 0,0,0,0) , d=c(NA, 0,0,0,0,0, 0,0,0,0,0,
0,0,0,0,0, 0,0,0,0)  ) # Chain 1
```

DATA - LIST FORMAT

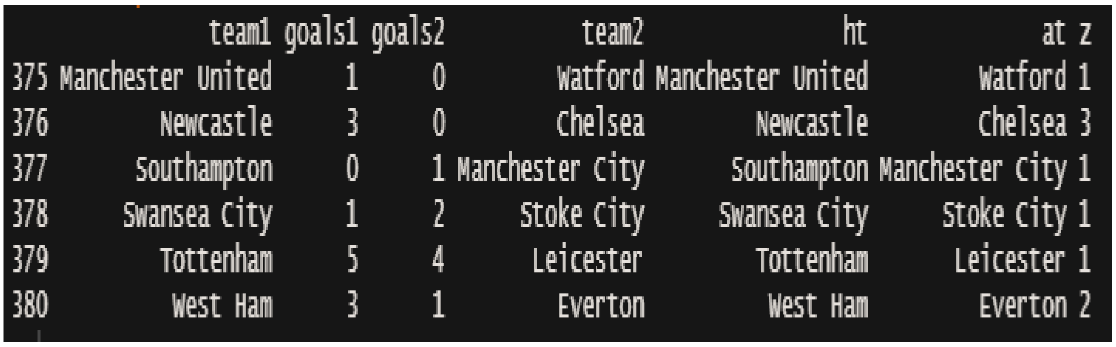
```
list(n=380, K=20,
ht=c(1,3,5,6,7,14,18,19,12,13,2,4,9,10,14,15,16,8,17,11,2,6,8,12,1
3,18,5,10,17,19,1,3,7,9,11,14,15,4,16,20,2,6,8,10,13,17,18,19,5
,12,4,7,9,11,14,15,16,20,3,1,2,5,8,12,15,19,20,1,7,13,4,6,10,11
,16,17,18,3,14,9,20,5,8,11,13,14,15,16,7,17,1,2,6,10,12,18,19,3
,9,4,8,13,14,15,16,20,5,7,11,17,1,2,4,6,9,10,12,19,18,3,20,6,10
,12,13,16,17,4,8,14,3,9,18,19,1,2,5,7,11,15,1,3,5,7,9,15,18,19,
2,11,4,6,8,13,16,17,20,10,12,14,4,6,8,10,12,13,14,16,17,20,1,3,
5,9,11,15,18,2,19,7,1,3,4,7,9,11,14,15,16,20,2,5,8,10,12,17,18,
19,13,6,2,5,8,10,12,13,18,6,19,3,4,7,9,15,11,14,16,20,1,17,5,6,
8,13,17,18,19,2,10,12,1,3,4,7,9,11,15,20,14,16,8,16,20,5,7,11,1
3,14,15,17,1,2,3,4,9,12,19,6,10,18,7,11,15,16,17,20,8,13,14,5,2
,3,4,9,10,18,19,6,12,1,4,9,10,14,16,17,18,3,11,6,5,7,8,12,13,19
,20,1,2,15,2,8,10,15,3,6,7,12,13,18,19,20,1,5,2,3,7,9,11,15,18,
19,1,5,4,6,8,10,14,16,17,12,13,20,3,2,4,9,18,19,1,11,15,7,4,6,8
,10,13,14,16,12,20,17,3,2,7,9,15,18,19,1,5,11,16,5,9,11,17,20,4,6,8,10,12,13,14,16,17,20 ),
at=c(9,11,4,8,15,16,10,2,20,17,18,19,3,6,20,1,12,13,5,7,11,16,14,
9,20,3,7,1,4,15,2,19,17,5,10,18,12,6,13,8,3,14,9,4,15,16,11,20,
1,7,8,2,10,6,12,5,18,17,13,19,9,11,17,6,14,18,16,3,4,10,20,5,12
,15,8,2,1,7,13,19,3,18,12,4,6,19,2,9,1,10,16,5,20,8,17,15,11,14
,7,13,19,2,4,9,3,10,12,18,1,6,17,8,16,7,11,14,13,5,20,15,9,15,5
,3,18,2,19,1,11,7,6,17,12,13,8,4,16,20,14,10,12,10,13,8,4,16,17
,6,14,20,18,2,3,9,19,15,5,7,11,1,15,18,5,19,2,7,9,11,3,1,13,4,1
4,6,17,20,8,10,12,16,10,18,17,5,12,2,8,19,6,13,20,3,15,16,4,14,
9,7,11,1,7,15,4,9,14,3,16,11,1,2,10,12,8,13,18,6,17,19,5,20,9,4
,20,16,7,14,3,1,11,15,6,5,12,19,18,13,8,2,17,10,10,1,6,2,9,19,4
,3,18,12,7,15,20,11,16,8,14,13,17,5,6,9,3,4,1,18,2,12,10,19,13,
16,14,15,20,7,8,17,5,11,7,2,13,15,20,8,19,1,5,12,6,3,16,10,14,9
,4,18,17,11,19,6,18,7,9,10,11,16,8,2,4,14,15,17,6,8,10,13,12,17
,4,16,14,20,9,3,18,2,5,7,11,19,1,15,17,12,5,14,6,10,20,16,4,13,
3,9,7,15,19,2,5,1,11,18,12,16,14,20,6,13,17,4,10,8,14,8,1,3,13,12,2,19,1,3,18,5,11,15,9,7 ),
goals1=c(4,0,2,0,1,0,3,1,4,0,0,0,2,1,3,1,0,1,1,1,1,0,0,2,3,0,2,4,1,1,
3,3,0,1,5,0,2,1,0,2,2,0,1,1,2,0,0,0,0,4,0,2,2,5,0,0,1,2,1,2,0,0
,0,4,2,2,1,2,0,1,1,2,0,7,2,1,2,1,2,1,0,4,2,3,1,1,1,1,2,4,2,0,2,
3,1,0,2,1,2,1,1,0,0,2,0,1,1,3,3,1,2,4,2,2,0,3,4,0,2,2,1,2,1,1,0
,0,1,0,1,4,0,2,2,2,5,1,1,4,2,0,1,1,3,2,1,2,1,0,1,2,1,2,2,2,1,5,
1,1,1,1,1,2,1,0,1,0,1,0,2,0,1,0,1,0,4,0,1,0,1,3,3,1,0,0,2,4,1,3
,1,2,3,2,1,5,2,5,2,0,0,2,2,5,0,2,0,0,1,0,1,2,1,0,3,0,3,1,0,2,2,
1,0,1,1,1,4,2,2,2,4,3,4,0,0,1,2,3,2,1,1,1,0,3,1,0,2,3,1,1,0,2,5
,2,3,1,1,2,2,1,2,4,3,5,1,1,1,2,4,1,0,3,2,4,1,1,4,1,1,0,2,0,2,1,
2,0,4,2,1,2,1,2,2,2,0,2,3,1,0,3,1,0,2,0,5,1,0,1,1,2,1,2,1,3,3,1
,2,1,0,1,2,1,1,1,3,1,2,3,1,3,2,1,1,0,2,1,1,0,1,0,0,2,4,5,1,1,0,
5,0,0,0,2,0,2,1,2,1,1,1,0,1,2,1,5,1,0,0,1,3,3,1,0,1,2,0,4,1,3,0,1,NA,NA ),
goals2=c(3,2,3,3,0,0,3,0,0,2,2,1,0,0,2,0,4,0,2,1,2,2,0,0,0,0,0,0,1,1,
0,1,3,2,0,2,2,0,1,0,1,1,1,1,1,0,6,0,0,0,0,1,3,0,1,4,2,3,0,0,0,1
,4,0,1,2,0,0,1,1,1,1,0,2,0,0,1,1,2,1,3,2,1,0,0,0,2,2,5,1,1,1,2,
0,0,1,3,1,0,0,0,1,1,2,1,4,0,2,1,0,0,0,0,2,2,0,1,4,0,2,1,1,1,0,3
,0,1,1,2,1,0,1,4,2,0,2,0,0,1,3,3,5,1,0,0,1,1,0,1,1,0,2,0,3,0,1,
0,1,2,1,0,1,3,0,0,1,4,4,0,0,0,0,0,3,1,3,4,4,2,1,3,0,3,0,2,0,1,1
,1,3,3,0,1,0,2,2,1,0,1,3,1,0,0,1,0,0,2,0,1,2,2,2,0,1,1,2,2,1,2,
1,0,0,4,1,0,2,0,1,3,0,1,4,1,1,0,1,0,1,1,0,3,1,1,3,1,0,1,1,0,0,1
,1,1,1,1,0,3,1,2,1,1,1,1,0,0,0,1,0,2,0,2,1,1,1,1,0,2,1,1,3,1,1,
0,0,1,0,0,1,0,3,1,0,0,1,0,4,3,0,4,2,1,2,0,2,2,2,3,0,0,2,2,0,0,3
,2,1,0,2,3,2,2,1,2,1,1,2,0,0,3,1,3,1,1,1,1,2,2,0,0,2,1,0,1,0,0,
0,2,0,1,1,1,1,4,0,0,0,1,2,2,1,0,0,0,0,1,1,1,1,0,0,2,0,1,0,0,0,1,2,NA,NA ) )
```

The results generated after running 5000 iterations (1000 iterations were discarded as the burn-in period) are as follows:



```
                mean    sd      MC_error  val2.5pc  median  val97.5pc  start  sample
outcome[1,1]    0.7318  0.443   0.006289  0.0       1.0     1.0        1001   5000
outcome[1,2]    0.1594  0.366   0.005627  0.0       0.0     1.0        1001   5000
outcome[1,3]    0.1088  0.3114  0.004258  0.0       0.0     1.0        1001   5000
outcome[2,1]    0.4166  0.493   0.007296  0.0       0.0     1.0        1001   5000
outcome[2,2]    0.2506  0.4334  0.006162  0.0       0.0     1.0        1001   5000
outcome[2,3]    0.3328  0.4712  0.006781  0.0       0.0     1.0        1001   5000
pred.diff[1]    1.672   1.841   0.0249    -2.0      2.0     5.0        1001   5000
pred.diff[2]    0.2044  1.774   0.02799   -3.0      0.0     4.0        1001   5000
```

The last five matches along with the actual results for match day 38 (the last of match day of the league) can be depicted below:



```
                team1 goals1 goals2           team2                ht              at z
375 Manchester United      1      0         Watford Manchester United         Watford 1
376          Newcastle      3      0         Chelsea         Newcastle         Chelsea 3
377        Southampton      0      1 Manchester City       Southampton Manchester City 1
378       Swansea City      1      2      Stoke City      Swansea City      Stoke City 1
379          Tottenham      5      4       Leicester         Tottenham       Leicester 1
380           West Ham      3      1         Everton          West Ham         Everton 2
```

The prediction refers to the matches:

Tottenham-Leicester

West Ham-Everton

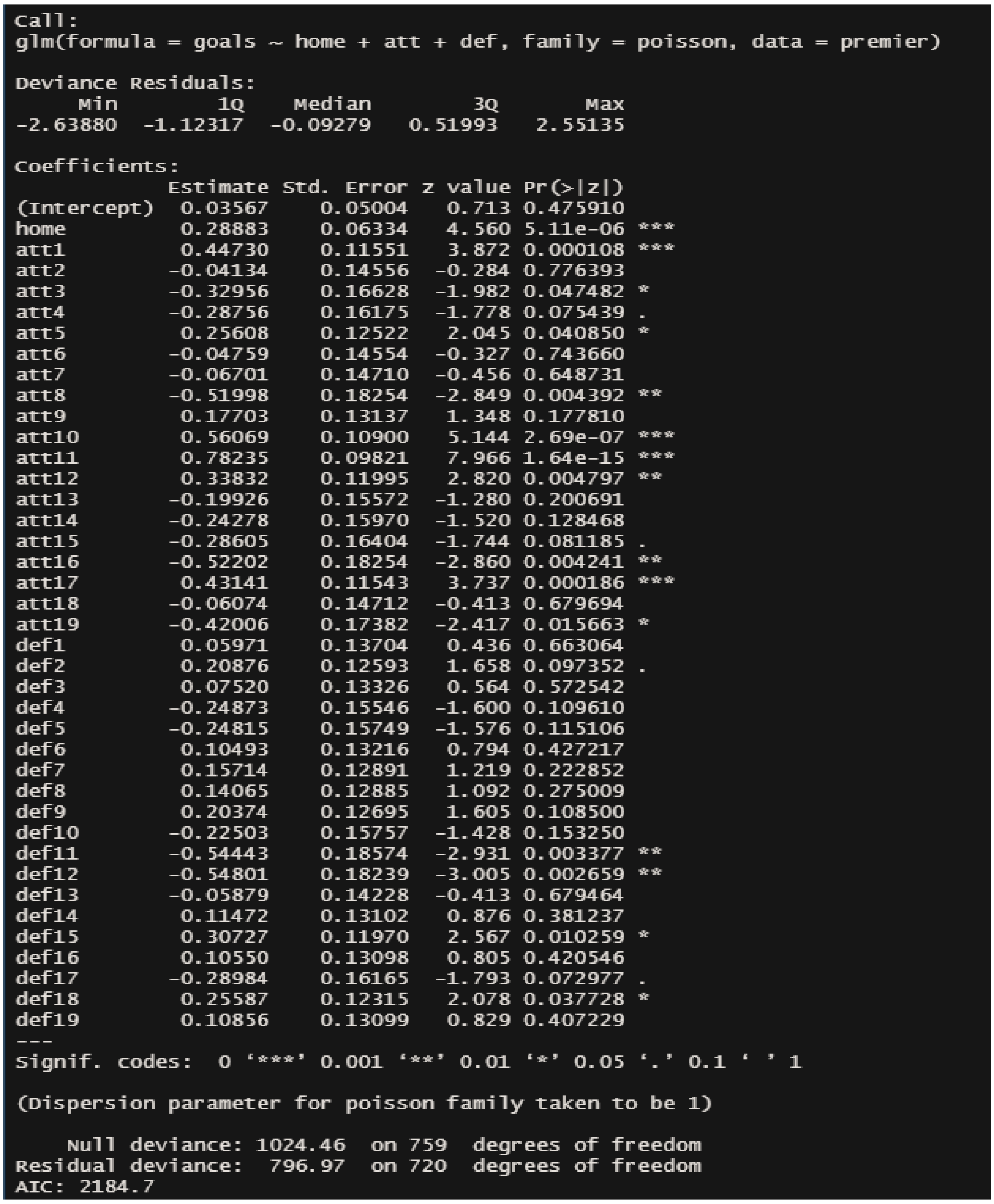
OpenBUGS automatically generates values for the missing goals from the predictive distribution and will provide estimates for each score by monitoring the nodes goals1 and goals2. Outcome probabilities indicate that for the match Tottenham-Leicester, the home's team probability of winning was about 73% (with a posterior mode of difference about 1.64), the probability of draw was about 15% and the probability for the away team to win the match was about 10%. As regards the match West Ham-Everton the home's team probability of winning was about 42% (with a posterior mode of difference about 0.2), the probability of draw was about 25% and the probability for the away team to win the match was about 33%. Thus, we can gather that the Vanilla model did not bad at all in this case!

A different approach using GLM (Generalized Linear Models)

To begin with, we transform the data in order to implement a Poisson glm model:

- One response -> Goals=(goals1, goals2)
- Each game -> 2 lines in the dataset (home teams vs the away team and vice versa!)
- Add home team indicator (binary)

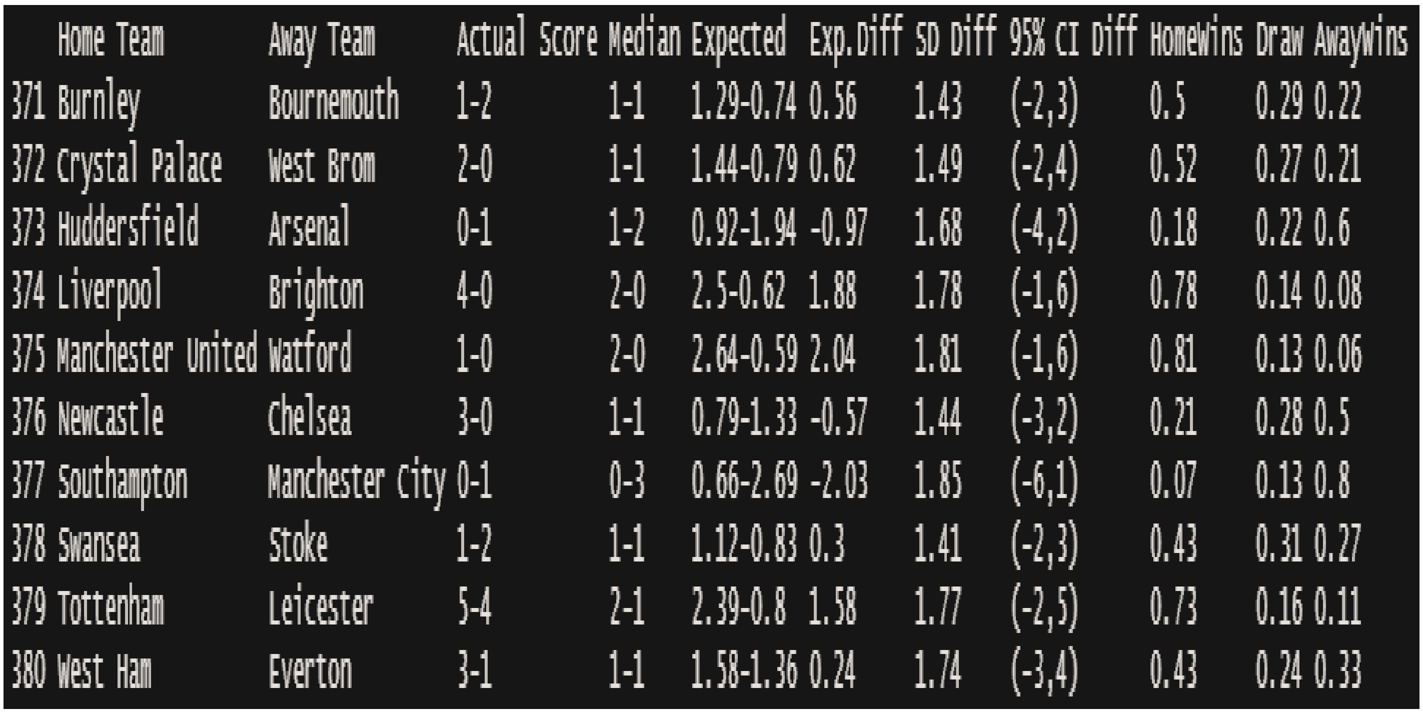
The model parameters of the Vanilla model in R can be depicted in the picture below:



```
Call:
glm(formula = goals ~ home + att + def, family = poisson, data = premier)

Deviance Residuals:
     Min        1Q    Median        3Q       Max
-2.63880  -1.12317  -0.09279   0.51993   2.55135

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)  0.03567    0.05004   0.713 0.475910
home         0.28883    0.06334   4.560 5.11e-06 ***
att1         0.44730    0.11551   3.872 0.000108 ***
att2        -0.04134    0.14556  -0.284 0.776393
att3        -0.32956    0.16628  -1.982 0.047482 *
att4        -0.28756    0.16175  -1.778 0.075439 .
att5         0.25608    0.12522   2.045 0.040850 *
att6        -0.04759    0.14554  -0.327 0.743660
att7        -0.06701    0.14710  -0.456 0.648731
att8        -0.51998    0.18254  -2.849 0.004392 **
att9         0.17703    0.13137   1.348 0.177810
att10        0.56069    0.10900   5.144 2.69e-07 ***
att11        0.78235    0.09821   7.966 1.64e-15 ***
att12        0.33832    0.11995   2.820 0.004797 **
att13       -0.19926    0.15572  -1.280 0.200691
att14       -0.24278    0.15970  -1.520 0.128468
att15       -0.28605    0.16404  -1.744 0.081185 .
att16       -0.52202    0.18254  -2.860 0.004241 **
att17        0.43141    0.11543   3.737 0.000186 ***
att18       -0.06074    0.14712  -0.413 0.679694
att19       -0.42006    0.17382  -2.417 0.015663 *
def1         0.05971    0.13704   0.436 0.663064
def2         0.20876    0.12593   1.658 0.097352 .
def3         0.07520    0.13326   0.564 0.572542
def4        -0.24873    0.15546  -1.600 0.109610
def5        -0.24815    0.15749  -1.576 0.115106
def6         0.10493    0.13216   0.794 0.427217
def7         0.15714    0.12891   1.219 0.222852
def8         0.14065    0.12885   1.092 0.275009
def9         0.20374    0.12695   1.605 0.108500
def10       -0.22503    0.15757  -1.428 0.153250
def11       -0.54443    0.18574  -2.931 0.003377 **
def12       -0.54801    0.18239  -3.005 0.002659 **
def13       -0.05879    0.14228  -0.413 0.679464
def14        0.11472    0.13102   0.876 0.381237
def15        0.30727    0.11970   2.567 0.010259 *
def16        0.10550    0.13098   0.805 0.420546
def17       -0.28984    0.16165  -1.793 0.072977 .
def18        0.25587    0.12315   2.078 0.037728 *
def19        0.10856    0.13099   0.829 0.407229
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for poisson family taken to be 1)

    Null deviance: 1024.46  on 759  degrees of freedom
Residual deviance:  796.97  on 720  degrees of freedom
AIC: 2184.7
```

Prediction for match day 38 (10 games prediction) using the GLM approach

We can view the actual scores, the expected goals scored by each team, the expected difference of goals, the 95% confidence intervals of the goals, plus the probabilities of each outcome in the picture below:

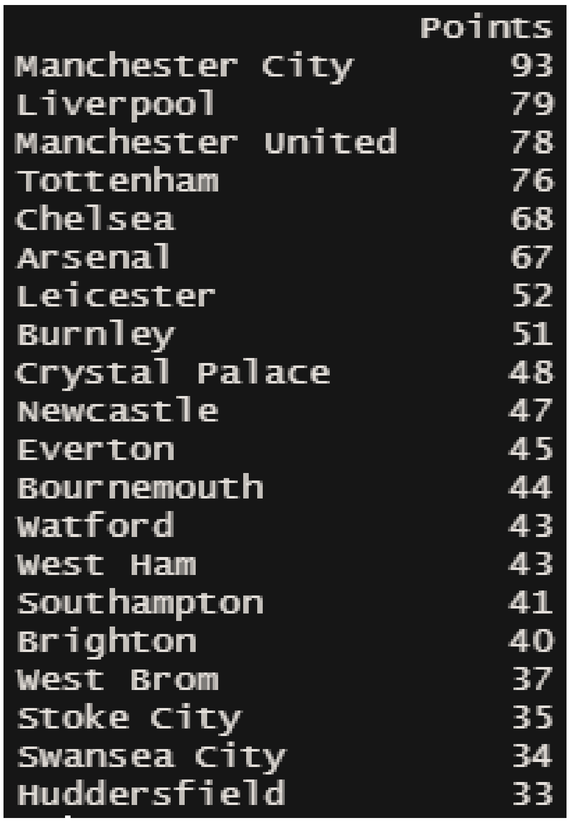


```
    Home Team         Away Team         Actual Score Median Expected  Exp.Diff SD Diff 95% CI Diff HomeWins Draw AwayWins
371 Burnley           Bournemouth       1-2          1-1    1.29-0.74 0.56     1.43    (-2,3)      0.5      0.29 0.22
372 Crystal Palace    West Brom         2-0          1-1    1.44-0.79 0.62     1.49    (-2,4)      0.52     0.27 0.21
373 Huddersfield      Arsenal           0-1          1-2    0.92-1.94 -0.97    1.68    (-4,2)      0.18     0.22 0.6
374 Liverpool         Brighton          4-0          2-0    2.5-0.62  1.88     1.78    (-1,6)      0.78     0.14 0.08
375 Manchester United Watford           1-0          2-0    2.64-0.59 2.04     1.81    (-1,6)      0.81     0.13 0.06
376 Newcastle         Chelsea           3-0          1-1    0.79-1.33 -0.57    1.44    (-3,2)      0.21     0.28 0.5
377 Southampton       Manchester City   0-1          0-3    0.66-2.69 -2.03    1.85    (-6,1)      0.07     0.13 0.8
378 Swansea           Stoke             1-2          1-1    1.12-0.83 0.3      1.41    (-2,3)      0.43     0.31 0.27
379 Tottenham         Leicester         5-4          2-1    2.39-0.8  1.58     1.77    (-2,5)      0.73     0.16 0.11
380 West Ham          Everton           3-1          1-1    1.58-1.36 0.24     1.74    (-3,4)      0.43     0.24 0.33
```

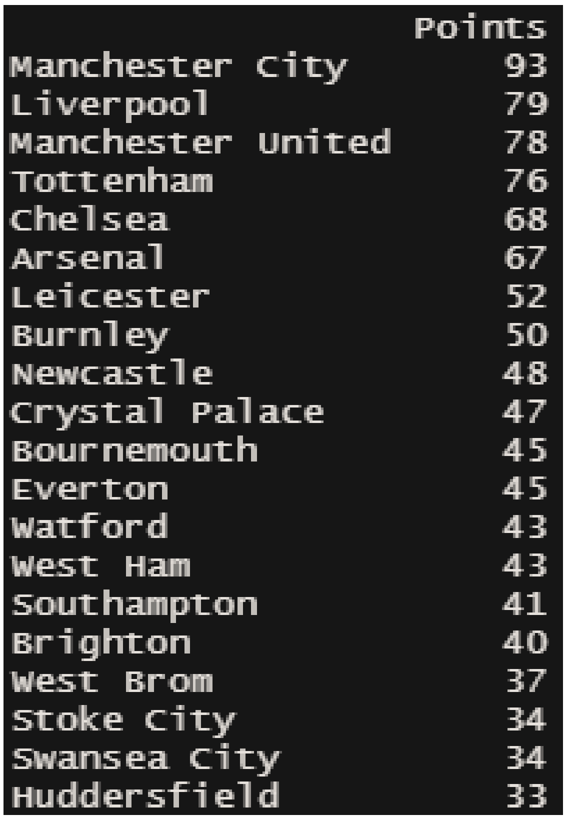
We observe that using the GLM approach to predict the results for the final match day of the Premier league, was about 60% successful (interpreting the expected difference of the goals compared to the actual scores)! The results between the two different approaches (GLM and the Bayesian approach via MCMC) are quite similar because there hasn’t been introduced any additional prior variability.

Regeneration of the full 2017-2018 Premier league using the NB model (via MCMC)

The output of the regeneration of the league can be illustrated in the left picture below: (the right picture depicts the final rankings according to the Vanilla model in order to highlight their differences!)



```
                        Points
Manchester City             93
Liverpool                   79
Manchester United           78
Tottenham                   76
Chelsea                     68
Arsenal                     67
Leicester                   52
Burnley                     51
Crystal Palace              48
Newcastle                   47
Everton                     45
Bournemouth                 44
Watford                     43
West Ham                    43
Southampton                 41
Brighton                    40
West Brom                   37
Stoke City                  35
Swansea City                34
Huddersfield                33
```

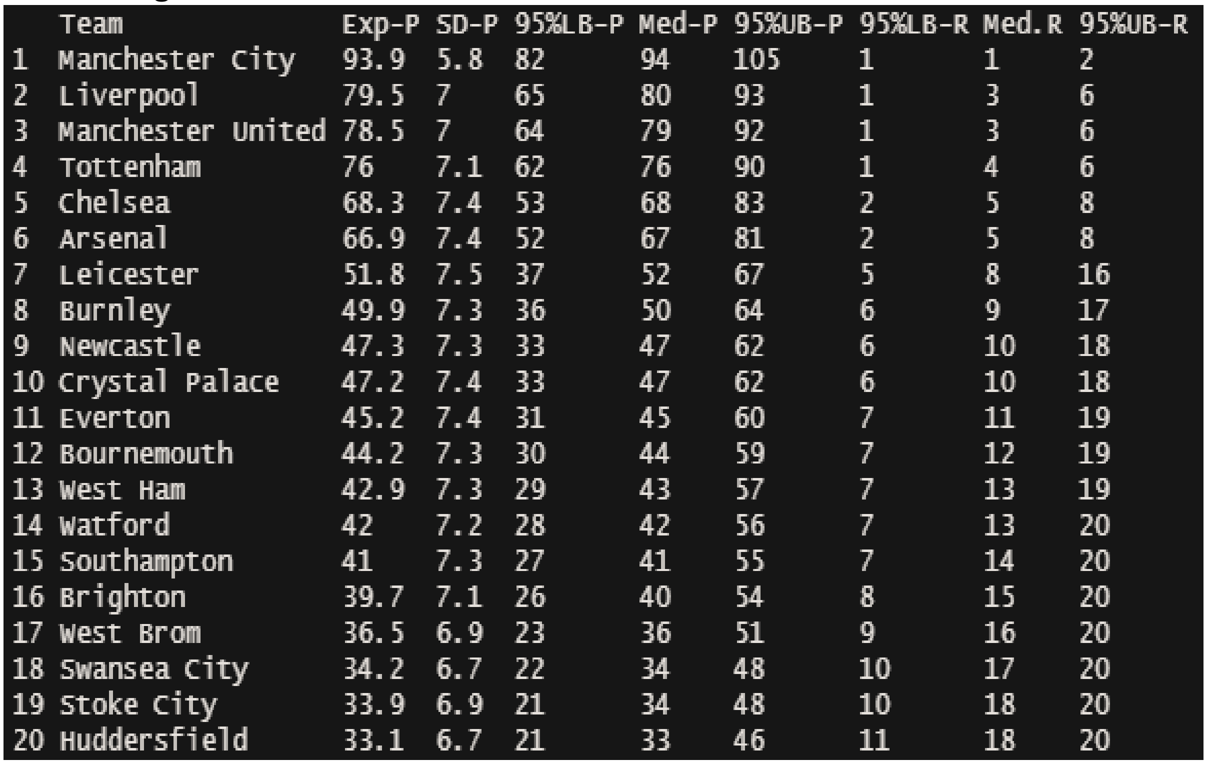


```
                        Points
Manchester City             93
Liverpool                   79
Manchester United           78
Tottenham                   76
Chelsea                     68
Arsenal                     67
Leicester                   52
Burnley                     50
Newcastle                   48
Crystal Palace              47
Bournemouth                 45
Everton                     45
Watford                     43
West Ham                    43
Southampton                 41
Brighton                    40
West Brom                   37
Stoke City                  34
Swansea City                34
Huddersfield                33
```

In comparison with the actual final rankings, we can gather that the predictive ability of the Negative Binomial model is about 35% (the teams that were placed in the position that was indeed their final rank). Thus, the differences with the Vanilla model seem to be minor!

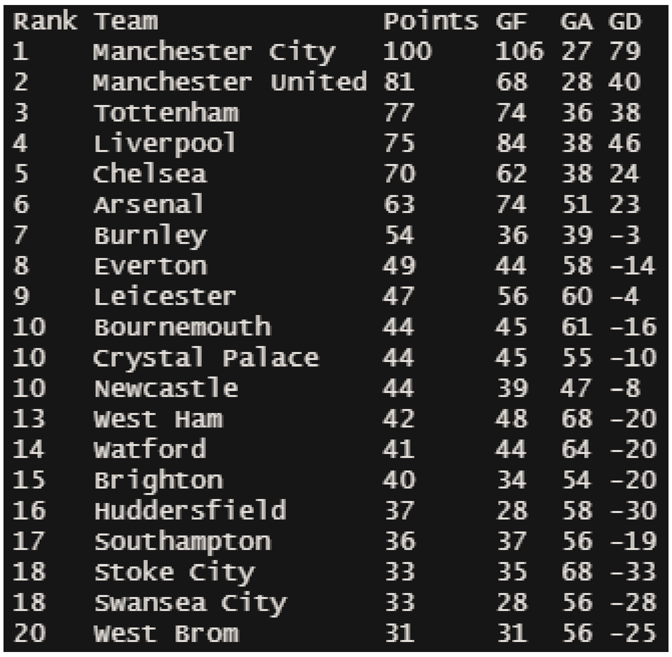
Regeneration of the full 2017-2018 Premier league using the Vanilla model (via GLM)

Running 10000 simulations (parametric bootstrap) we obtain the following results:



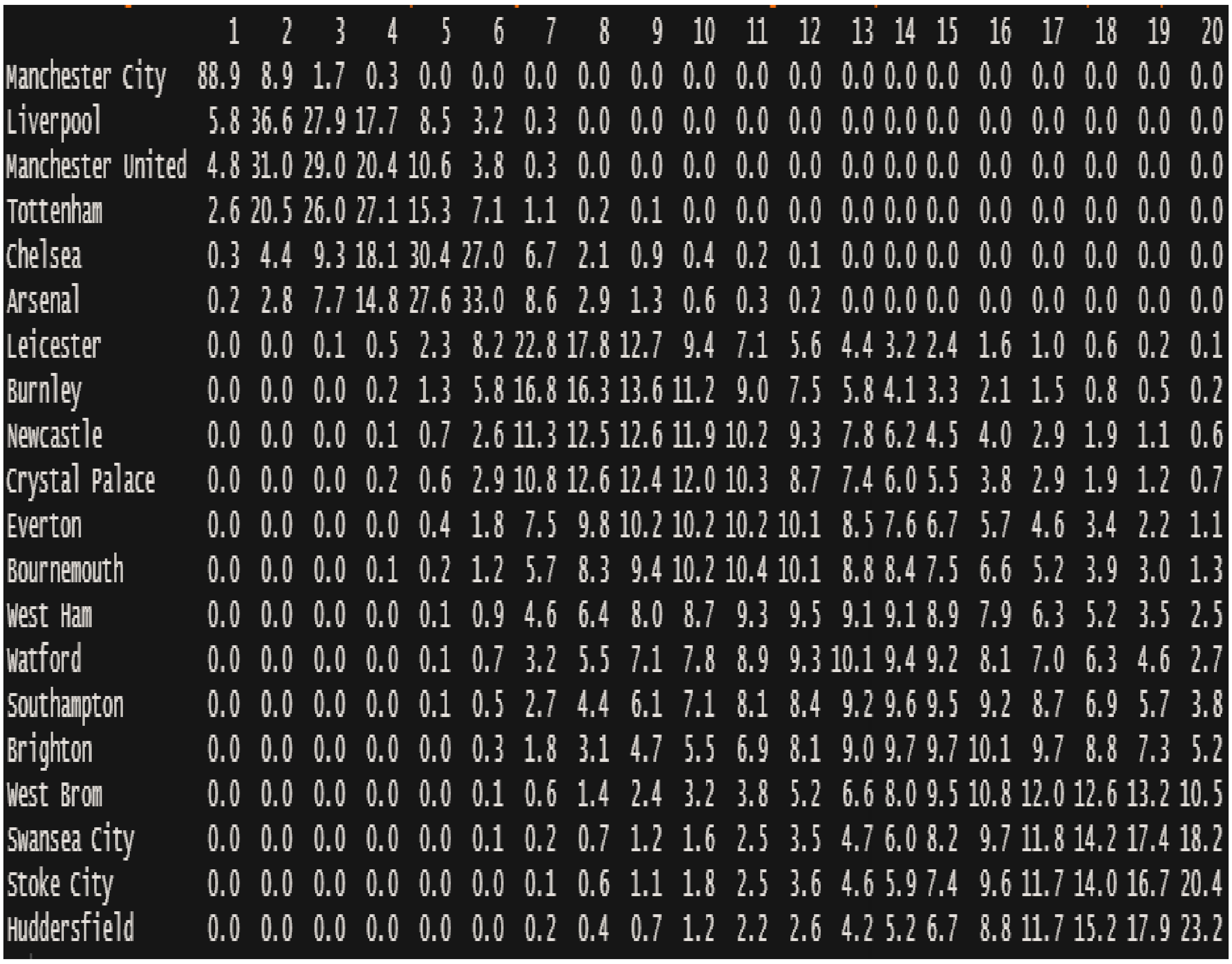
```
   Team              Exp-P SD-P 95%LB-P Med-P 95%UB-P 95%LB-R Med.R 95%UB-R
1  Manchester City   93.9  5.8  82      94    105     1       1     2
2  Liverpool         79.5  7    65      80    93      1       3     6
3  Manchester United 78.5  7    64      79    92      1       3     6
4  Tottenham         76    7.1  62      76    90      1       4     6
5  Chelsea           68.3  7.4  53      68    83      2       5     8
6  Arsenal           66.9  7.4  52      67    81      2       5     8
7  Leicester         51.8  7.5  37      52    67      5       8     16
8  Burnley           49.9  7.3  36      50    64      6       9     17
9  Newcastle         47.3  7.3  33      47    62      6       10    18
10 Crystal Palace    47.2  7.4  33      47    62      6       10    18
11 Everton           45.2  7.4  31      45    60      7       11    19
12 Bournemouth       44.2  7.3  30      44    59      7       12    19
13 West Ham          42.9  7.3  29      43    57      7       13    19
14 Watford           42    7.2  28      42    56      7       13    20
15 Southampton       41    7.3  27      41    55      7       14    20
16 Brighton          39.7  7.1  26      40    54      8       15    20
17 West Brom         36.5  6.9  23      36    51      9       16    20
18 Swansea City      34.2  6.7  22      34    48      10      17    20
19 Stoke City        33.9  6.9  21      34    48      10      18    20
20 Huddersfield      33.1  6.7  21      33    46      11      18    20
```

The simulated league can be compared to the actual one:



```
Rank Team              Points GF  GA GD
1    Manchester City   100    106 27 79
2    Manchester United 81     68  28 40
3    Tottenham         77     74  36 38
4    Liverpool         75     84  38 46
5    Chelsea           70     62  38 24
6    Arsenal           63     74  51 23
7    Burnley           54     36  39 -3
8    Everton           49     44  58 -14
9    Leicester         47     56  60 -4
10   Bournemouth       44     45  61 -16
10   Crystal Palace    44     45  55 -10
10   Newcastle         44     39  47 -8
13   West Ham          42     48  68 -20
14   Watford           41     44  64 -20
15   Brighton          40     34  54 -20
16   Huddersfield      37     28  58 -30
17   Southampton       36     37  56 -19
18   Stoke City        33     35  68 -33
18   Swansea City      33     28  56 -28
20   West Brom         31     31  56 -25
```

We observe that 7 out of the 20 teams were rightly placed, indicating about 35% success percentage. Interest lies in the fact that we can also view the probabilities for each team of the league finishing in each of the 20 positions, as follows:



```
                    1    2    3    4    5    6    7    8    9   10   11   12   13  14  15   16   17   18   19   20
Manchester City  88.9  8.9  1.7  0.3  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0 0.0 0.0  0.0  0.0  0.0  0.0  0.0
Liverpool         5.8 36.6 27.9 17.7  8.5  3.2  0.3  0.0  0.0  0.0  0.0  0.0  0.0 0.0 0.0  0.0  0.0  0.0  0.0  0.0
Manchester United 4.8 31.0 29.0 20.4 10.6  3.8  0.3  0.0  0.0  0.0  0.0  0.0  0.0 0.0 0.0  0.0  0.0  0.0  0.0  0.0
Tottenham         2.6 20.5 26.0 27.1 15.3  7.1  1.1  0.2  0.1  0.0  0.0  0.0  0.0 0.0 0.0  0.0  0.0  0.0  0.0  0.0
Chelsea           0.3  4.4  9.3 18.1 30.4 27.0  6.7  2.1  0.9  0.4  0.2  0.1  0.0 0.0 0.0  0.0  0.0  0.0  0.0  0.0
Arsenal           0.2  2.8  7.7 14.8 27.6 33.0  8.6  2.9  1.3  0.6  0.3  0.2  0.0 0.0 0.0  0.0  0.0  0.0  0.0  0.0
Leicester         0.0  0.0  0.1  0.5  2.3  8.2 22.8 17.8 12.7  9.4  7.1  5.6  4.4 3.2 2.4  1.6  1.0  0.6  0.2  0.1
Burnley           0.0  0.0  0.0  0.2  1.3  5.8 16.8 16.3 13.6 11.2  9.0  7.5  5.8 4.1 3.3  2.1  1.5  0.8  0.5  0.2
Newcastle         0.0  0.0  0.0  0.1  0.7  2.6 11.3 12.5 12.6 11.9 10.2  9.3  7.8 6.2 4.5  4.0  2.9  1.9  1.1  0.6
Crystal Palace    0.0  0.0  0.0  0.2  0.6  2.9 10.8 12.6 12.4 12.0 10.3  8.7  7.4 6.0 5.5  3.8  2.9  1.9  1.2  0.7
Everton           0.0  0.0  0.0  0.0  0.4  1.8  7.5  9.8 10.2 10.2 10.2 10.1  8.5 7.6 6.7  5.7  4.6  3.4  2.2  1.1
Bournemouth       0.0  0.0  0.0  0.1  0.2  1.2  5.7  8.3  9.4 10.2 10.4 10.1  8.8 8.4 7.5  6.6  5.2  3.9  3.0  1.3
West Ham          0.0  0.0  0.0  0.0  0.1  0.9  4.6  6.4  8.0  8.7  9.3  9.5  9.1 9.1 8.9  7.9  6.3  5.2  3.5  2.5
Watford           0.0  0.0  0.0  0.0  0.1  0.7  3.2  5.5  7.1  7.8  8.9  9.3 10.1 9.4 9.2  8.1  7.0  6.3  4.6  2.7
Southampton       0.0  0.0  0.0  0.0  0.1  0.5  2.7  4.4  6.1  7.1  8.1  8.4  9.2 9.6 9.5  9.2  8.7  6.9  5.7  3.8
Brighton          0.0  0.0  0.0  0.0  0.0  0.3  1.8  3.1  4.7  5.5  6.9  8.1  9.0 9.7 9.7 10.1  9.7  8.8  7.3  5.2
West Brom         0.0  0.0  0.0  0.0  0.0  0.1  0.6  1.4  2.4  3.2  3.8  5.2  6.6 8.0 9.5 10.8 12.0 12.6 13.2 10.5
Swansea City      0.0  0.0  0.0  0.0  0.0  0.1  0.2  0.7  1.2  1.6  2.5  3.5  4.7 6.0 8.2  9.7 11.8 14.2 17.4 18.2
Stoke City        0.0  0.0  0.0  0.0  0.0  0.0  0.1  0.6  1.1  1.8  2.5  3.6  4.6 5.9 7.4  9.6 11.7 14.0 16.7 20.4
Huddersfield      0.0  0.0  0.0  0.0  0.0  0.0  0.2  0.4  0.7  1.2  2.2  2.6  4.2 5.2 6.7  8.8 11.7 15.2 17.9 23.2
```

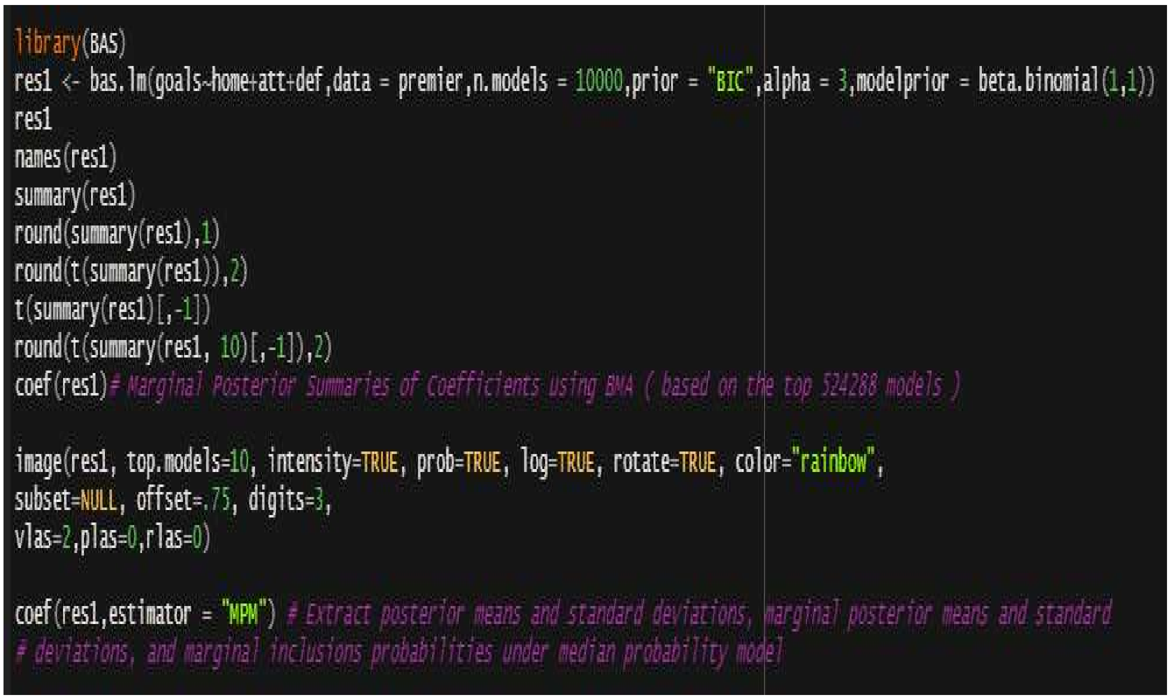
We can tell that Manchester City had 88.9% chance of finishing first, fact that implies that they were by far the best team of the league! Liverpool had 36.6% chance of finishing second, Manchester United had 31% chance of finishing second (United in reality finished clear second!). As regards the bottom of the league, Swansea, Stoke and West Brom had quite high chances of relegating and they didn't disappoint (they actually did disappoint their fans but statistically speaking) as they were relegated indeed!

Variable selection using BAS with RStudio

As regards variable selection, <<BAS>> package in R offers certain advantages:

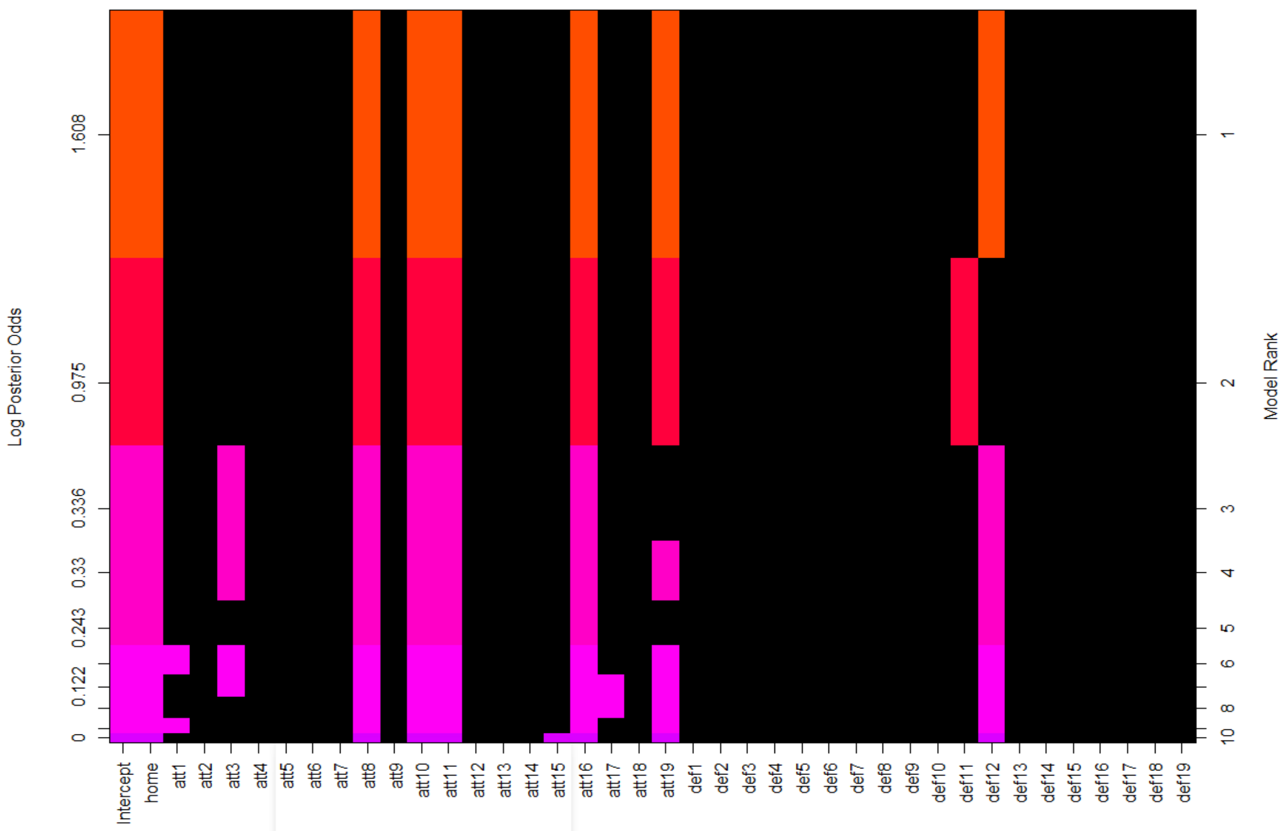
- Implements g-prior, Zellner and Siow and hyper-g
- Uses full enumeration for small spaces and adaptive sampling
- It’s very fast and easy to use
- GLMs can be easily fitted using Laplace approximations

Let’s view the R code using a prior distribution for the regression coefficients based on BIC and a beta binomial(1,1) prior distribution on the models:

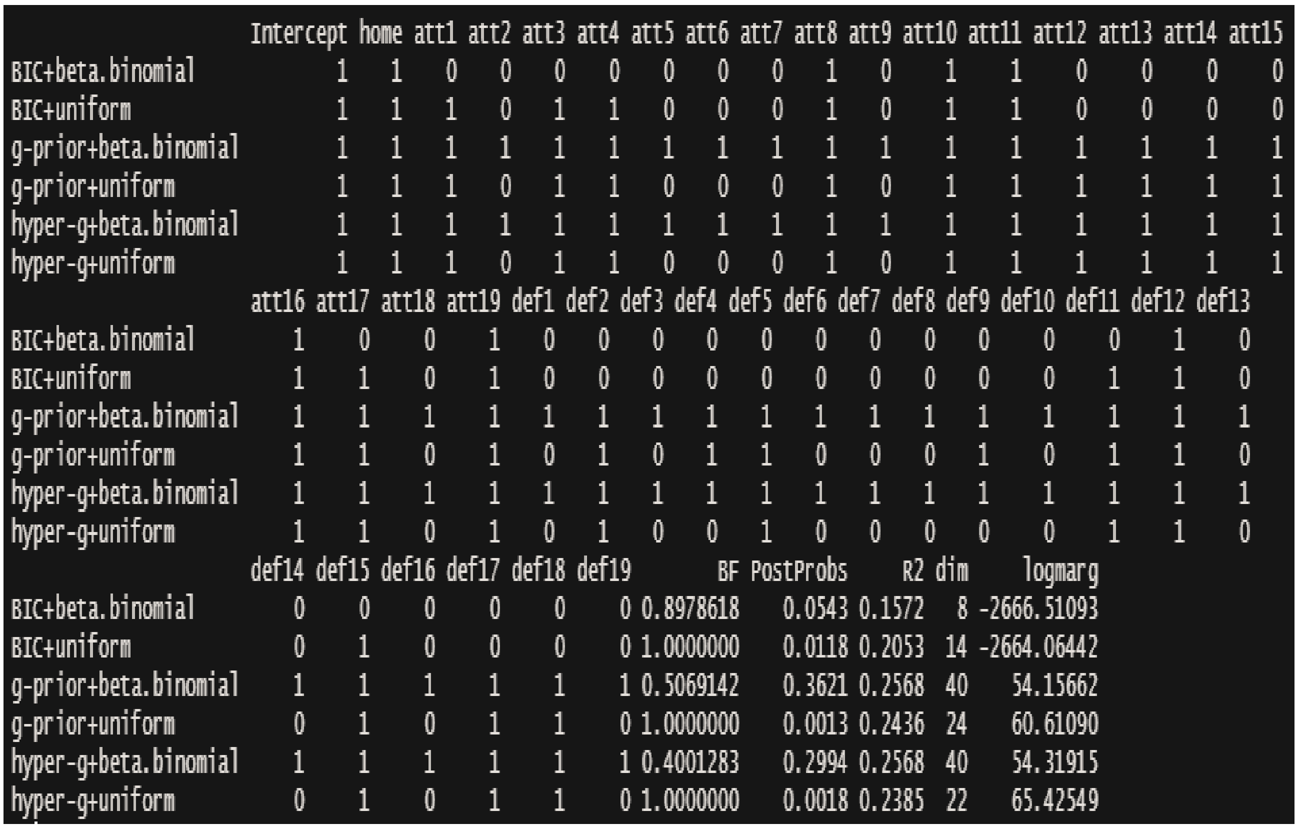


```r
library(BAS)
res1 <- bas.lm(goals~home+att+def,data = premier,n.models = 10000,prior = "BIC",alpha = 3,modelprior = beta.binomial(1,1))
res1
names(res1)
summary(res1)
round(summary(res1),1)
round(t(summary(res1)),2)
t(summary(res1)[,-1])
round(t(summary(res1, 10)[,-1]),2)
coef(res1)# Marginal Posterior Summaries of Coefficients using BMA ( based on the top 524288 models )

image(res1, top.models=10, intensity=TRUE, prob=TRUE, log=TRUE, rotate=TRUE, color="rainbow",
subset=NULL, offset=.75, digits=3,
vlas=2,plas=0,rlas=0)

coef(res1,estimator = "MPM") # Extract posterior means and standard deviations, marginal posterior means and standard
# deviations, and marginal inclusions probabilities under median probability model
```

In order to visualize better the models obtained from the regression illustrated above we can use the following picture that depicts the mostly used covariates (black color indicates the nonexistence of a covariate!):

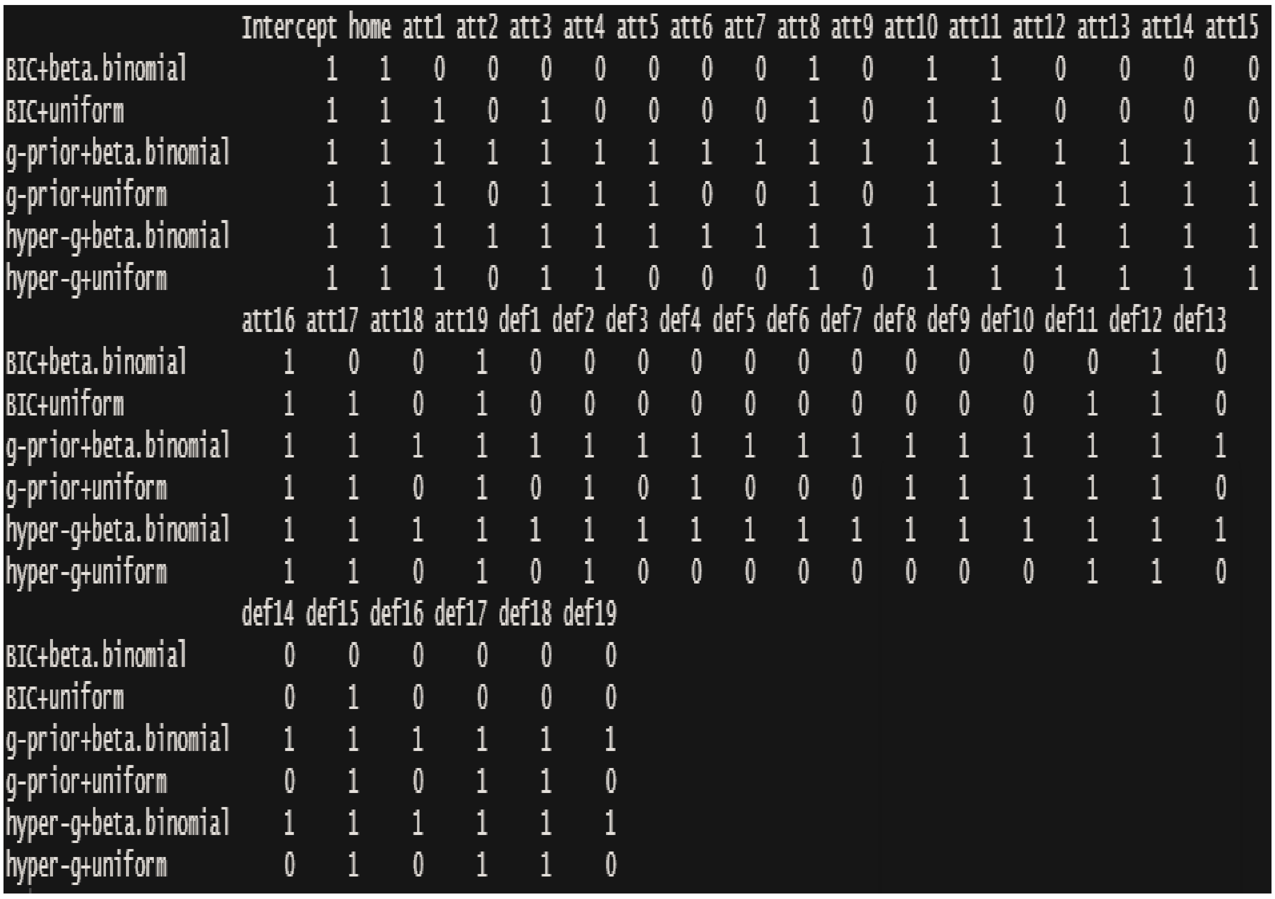


Using BIC, g-prior and the hyper-g prior with the uniform and the beta-binomial distribution on the model space we obtain the following results as regards the MAP (Max a-posteriori probabilities) model:



```
                      Intercept home att1 att2 att3 att4 att5 att6 att7 att8 att9 att10 att11 att12 att13 att14 att15
BIC+beta.binomial             1    1    0    0    0    0    0    0    0    1    0     1     1     0     0     0     0
BIC+uniform                   1    1    1    0    1    1    0    0    0    1    0     1     1     0     0     0     0
g-prior+beta.binomial         1    1    1    1    1    1    1    1    1    1    1     1     1     1     1     1     1
g-prior+uniform               1    1    1    0    1    1    0    0    0    1    0     1     1     1     1     1     1
hyper-g+beta.binomial         1    1    1    1    1    1    1    1    1    1    1     1     1     1     1     1     1
hyper-g+uniform               1    1    1    0    1    1    0    0    0    1    0     1     1     1     1     1     1
                      att16 att17 att18 att19 def1 def2 def3 def4 def5 def6 def7 def8 def9 def10 def11 def12 def13
BIC+beta.binomial         1     0     0     1    0    0    0    0    0    0    0    0    0     0     0     1     0
BIC+uniform               1     1     0     1    0    0    0    0    0    0    0    0    0     0     1     1     0
g-prior+beta.binomial     1     1     1     1    1    1    1    1    1    1    1    1    1     1     1     1     1
g-prior+uniform           1     1     0     1    0    1    0    1    1    0    0    0    1     0     1     1     0
hyper-g+beta.binomial     1     1     1     1    1    1    1    1    1    1    1    1    1     1     1     1     1
hyper-g+uniform           1     1     0     1    0    1    0    0    1    0    0    0    0     0     1     1     0
                      def14 def15 def16 def17 def18 def19        BF PostProbs     R2 dim     logmarg
BIC+beta.binomial         0     0     0     0     0     0 0.8978618    0.0543 0.1572   8 -2666.51093
BIC+uniform               0     1     0     0     0     0 1.0000000    0.0118 0.2053  14 -2664.06442
g-prior+beta.binomial     1     1     1     1     1     1 0.5069142    0.3621 0.2568  40    54.15662
g-prior+uniform           0     1     0     1     1     0 1.0000000    0.0013 0.2436  24    60.61090
hyper-g+beta.binomial     1     1     1     1     1     1 0.4001283    0.2994 0.2568  40    54.31915
hyper-g+uniform           0     1     0     1     1     0 1.0000000    0.0018 0.2385  22    65.42549
```

Added to this, we can view the posterior inclusion probabilities of each covariate based on MP (median probability model) as follows:



```
                      Intercept home att1 att2 att3 att4 att5 att6 att7 att8 att9 att10 att11 att12 att13 att14 att15
BIC+beta.binomial             1    1    0    0    0    0    0    0    0    1    0     1     1     0     0     0     0
BIC+uniform                   1    1    1    0    1    0    0    0    0    1    0     1     1     0     0     0     0
g-prior+beta.binomial         1    1    1    1    1    1    1    1    1    1    1     1     1     1     1     1     1
g-prior+uniform               1    1    1    0    1    1    1    0    0    1    0     1     1     1     1     1     1
hyper-g+beta.binomial         1    1    1    1    1    1    1    1    1    1    1     1     1     1     1     1     1
hyper-g+uniform               1    1    1    0    1    1    0    0    0    1    0     1     1     1     1     1     1
                      att16 att17 att18 att19 def1 def2 def3 def4 def5 def6 def7 def8 def9 def10 def11 def12 def13
BIC+beta.binomial         1     0     0     1    0    0    0    0    0    0    0    0    0     0     0     1     0
BIC+uniform               1     1     0     1    0    0    0    0    0    0    0    0    0     0     1     1     0
g-prior+beta.binomial     1     1     1     1    1    1    1    1    1    1    1    1    1     1     1     1     1
g-prior+uniform           1     1     0     1    0    1    0    1    0    0    0    1    1     1     1     1     0
hyper-g+beta.binomial     1     1     1     1    1    1    1    1    1    1    1    1    1     1     1     1     1
hyper-g+uniform           1     1     0     1    0    1    0    0    0    0    0    0    0     0     1     1     0
                      def14 def15 def16 def17 def18 def19
BIC+beta.binomial         0     0     0     0     0     0
BIC+uniform               0     1     0     0     0     0
g-prior+beta.binomial     1     1     1     1     1     1
g-prior+uniform           0     1     0     1     1     0
hyper-g+beta.binomial     1     1     1     1     1     1
hyper-g+uniform           0     1     0     1     1     0
```

Bayesian Statistics vs Frequent Statistics

Main difference between the two different approaches refers to the point of view both sides have regarding the parameter of interest:

Frequentist: The parameter is not a random variable.

Bayesian: The parameter is a random variable.

One word, huge difference! Let's take a closer look:

Which words tell you who you're dealing with?

What jargon tells you that you've stepped into their territory?

Frequentist: confidence interval, p-value, power, significance.

Bayesian: credible interval, prior, posterior.

What are their goals?

What are they using statistics to change their minds about?
Frequentist: actions to take (default action).
Bayesian: opinions to have (prior belief).

What is the main difference?

Frequentist: the parameter is a fixed quantity (no probability about it).
Bayesian: the parameter is a random variable (no right answer).

What's in it for you?

What do you gain by joining their way of thinking?
Frequentist: it makes sense to talk about your method's quality and "getting the answer right"
Bayesian: intuitive definitions, e.g. credible intervals are what you wish confidence intervals were (but aren't!)

What do you give up?

What do you lose if you choose their side?
Frequentist: the core concepts are harder to wrap your head around (e.g. p_values and confidence intervals have counter-intuitive, wordy definitions) and lazy thinkers make a hash out of them frequent-ly.
Bayesian: you lose the ability to talk about any notion of "right answers" and "method quality" — there's no such thing as statistically significant or rejecting the null hypotheis. There's only "more likely" and "less likely" …from your perspective!

So, which one is better?

Wrong question! The right one to choose depends on how you want to approach your decision-making. For example, if you have no default action, go Bayesian. Without a default action, the Frequentist approach is less practical than the Bayesian approach unless you have special philosophical reasons for invoking the concept of TRUTH in your calculations.

(Note: those last three words are important. We're not talking about the concept of truth in general, but rather about how it's handled in the math that powers these approaches to statistics. The distinction between the camps boils down to whether you treat the parameter of interest as a fixed constant or not.)

Okay…so which one is more objective?

Neither! They're both based on assumptions, so they're fundamentally subjective. The key difference is how they assist decision-making once the decision context has been framed.

Wait, what about sample size? Isn't the Bayesian the way to go with small data?

If you've been hanging out with the ***"Frequentist if there's lots of data, Bayesian if there isn't"*** folks, you might be sold on the idea that you should let sample size decide which camp to go with. Alas, the reasoning behind their advice gets wobbly if you poke it. Yes, it's true that Frequentists spurn babby datasets. If you've got more fingers than examples, they'll almost surely tell you not to bother. Yes, it's true that if you take a Bayesian approach, you can proceed with as little as one (!) datapoint. The math checks out. Sure. You can do it.

…But *should* you?

Being allowed proceed with a pittance of data might be a bug instead of a feature. There are circumstances where you definitely don't want to be doing that. (Statistics isn't alchemy. We're not making gold out of thin air! There's the same amount of data in one datapoint no matter which school of thought you pledge fealty to.) The way to "require less data" is to make bigger assumptions (this holds for both philosophies)… so do take a moment to ponder the nutritional content of your conclusions when your main ingredient isn't data but, essentially, some nonsense you made up. If you take yourself too seriously when working with tiny data, expert Bayesians and Frequentists alike will forget their differences long enough to join in a belly laugh at your expense.

To sum up

Overall, it depends on whether the situation calls for choosing between actions or forming an evidence-based opinion. Don’t pick a side. See them as two different approaches that fit two different styles of decision-making and reasoning, then leave yourself the option of using whichever one suits the mindset/context you find yourself in.

References

Ntzoufras, I., (2008), “*Bayesian Modeling using WinBUGS”, Wiley*

Kozyrkov, C., (2021), “*Statistics: Are you Bayesian or Frequentist?”*