An Analysis of adjusted ERA for starting pitchers in Major League Baseball

Taylor Last

STAT 4350 - Bayesian Statistics
University of Georgia
May 5th, 2021

1 Introduction

Baseball is the sport of statistics. The analysis and projection of professional baseball has been an area of interest among analysts within the baseball community as well as academic studies for a long time. Whether it be using advanced statistical methods to evaluate how good a player is and how much he should be paid or simply using stats to set your fantasy baseball lineup, statistics in baseball are very widely used.

One of the most highly debated topics in baseball statistics is determining how to evaluate pitchers performance. Overall, evaluating a team of batters is relatively easy. There is one player hitting for the team, so every batting statistic that occurs is related to that specific player. However, every defensive statistic does not belong to an individual player. A pitcher could perform very well and give up a lot of runs because his defense played poorly. Also, a pitcher could perform very poorly, but his defense could play well. There are statistics to distinguish what the pitcher is responsible for and what the defense is responsible for, but they are not all mutually exclusive. In this paper, we look at several statistics that attempt to quantify pitcher performance by team.

To conduct the evaluation of different team pitching, we used several statistics from FanGraphs.com, which is the most highly renown public source for pitching statistics. The analysis performed in this paper displays how much a team's starting pitching is underperforming or overperforming so far in the 2021 season. The variables used in the model include the following details:

- *ERA-* : Adjusted Earned Runs per nine innings. An earned run is a run that is scores against the pitcher without the benefit of an error. The ERA adjusted variable considers ball park conditions and league averages.
- *HR/9*: Home Runs per nine innings
- *BABIP*: Batting Average on Balls in Play. The rate at which the pitcher allows a hit when the ball is put in play, calculated as (Hits - HomeRuns)/(AtBats - StrikeOuts - HomeRuns + SacrificeFly).
- *LOB%*:Left on base percentage. Percentage of pitcher's own base runners that they strand over the course of a season.
- *WAR*: Wins Above Replacement. A comprehensive statistic that estimates the number of wins a player has been worth to his team compared to a freely available player such as a minor league free agent based on his FIP.
- *K% - BB%*: Strikeout Percentage Minus Walk Percentage. The percentage differential between K% and BB%, often a better indicator of performance than K/BB, which can be skewed by very low walk rates.
- *WHIP* : Walks Plus Hits Per Inning Pitched. The average number of base runners allowed via hit or walk per inning.
- *WIN%*: Win Percentage. The pitcher's wins divided by his wins+losses.

2 Exploratory Data Analysis

To explore the data, we first looked at the distribution for each variable to ensure that there were no anomalies or any influential points that could skew the data significantly.

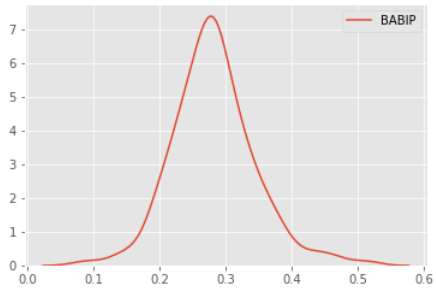


Figure 1: BABIP Distribution

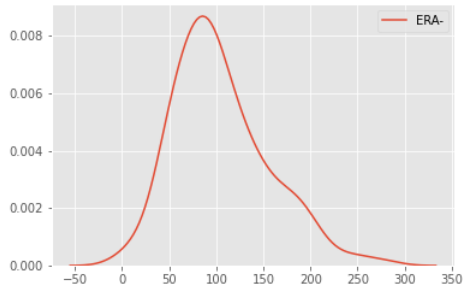


Figure 2: ERA- Distribution

Looking at Figure 1, we see that the BABIP distribution is well behaved and approximately normal. More importantly, we see in Figure 2 that the ERA- distribution is also approximately normal. It appears to be slightly skewed right, but for interpretability of ERA-, it was not transformed. The other predictors in the data don't have any influential outliers that would severely decrease the accuracy of our model. Also, the other predictors appear to follow an approximately normal distribution.

The next aspect of the data we explored was the correlation matrix of all the predictors and the response variable.

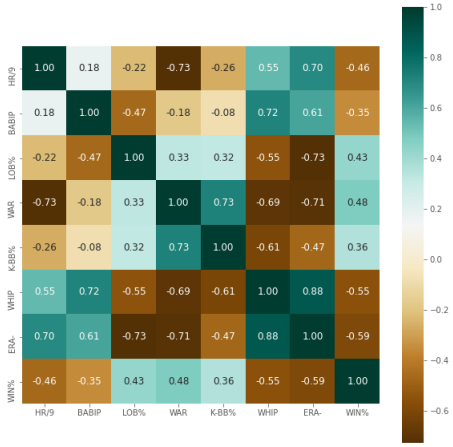


Figure 3: Correlation Heatmap

Looking at the correlation matrix in Figure 3, we see that some of the predictors are heavily correlated with the response. This intuitively makes sense because ERA is calculated from earned runs and these statistics attempt to quantify why an earned run happens or doesn't happen. Also, there appear to be some multicollinearity in the dataset between some of the predictors. It makes sense that some of the predictors because they are derived from some of the same basic statistics. Since we are attempting to include most of the variance that explains ERA-, we keep this subset of variables even though some are relatively highly correlated with each other.

To adequately estimate the expected adjusted ERA for each team, we took the sum of the variables across each team. This allowed us to estimate the ERA for a certain team and the uncertainty generated by different pitchers.

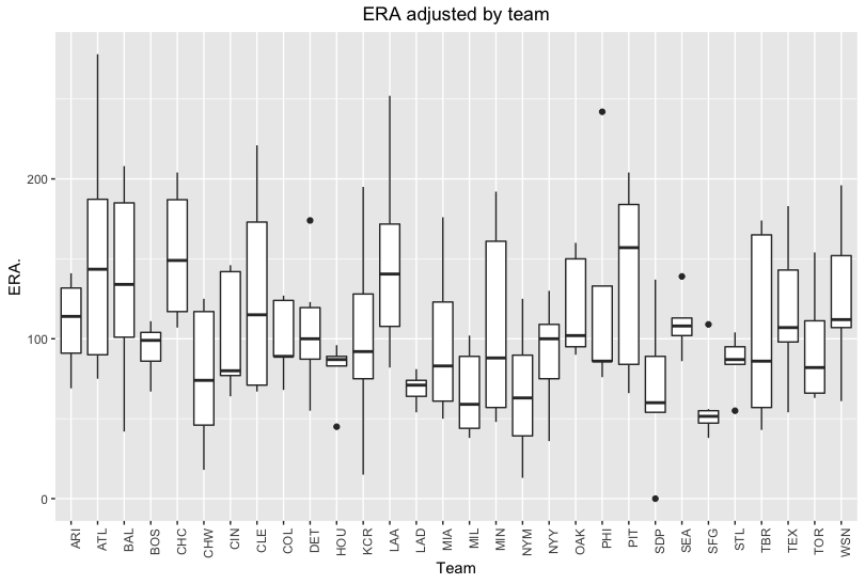


Figure 4: Correlation Heatmap

Looking at Figure 4, we see how well each team's starting pitching has been for 2021. Teams like the The Houston Astros, The Los Angeles Dodgers, and The San Francisco Giants have adjusted ERAs that are below the baseline of 100 with small amounts of variance, while teams like The Chicago Cubs and Pittsburgh Pirates have adjusted ERAs above the baseline with higher amounts of variance. While this chart is useful to evaluate how well a team is pitching, it doesn't answer our question if their pitching is underperforming or outperforming. In order to find out the answer to our question, we must fit our mixed linear model.

3 Methodology

3.1 Hierarchical Bayesian Linear Regression

The Hierarchical Bayesian Linear Regression allowed us to fit our predictors on adjusted ERA while accounting for the team-specific effect. The model uses a series of smaller models to specify the prior

distributions of the parameters. First, we specify the sampling model to account for team specific adjusted ERA where j represents each team and i represents each pitcher in the team:

$$Y_{ij}|\alpha, \beta, \sigma^2 \sim N(\mu_{ij}, \sigma^2)$$

Then we specify the priors for the coefficients of the predictors (β) and the random team effect (α).

$$\beta_1 ... \beta_8 \sim N(0, 10^{10})$$
$$\alpha_1 ... \alpha_J | \sigma_\alpha^2 \sim N(0, \sigma_\alpha^2)$$

Finally, we specify the priors for the between-team variance (σ_α^2) and the within-team variance (σ^2):

$$\sigma_\alpha^2 \sim IG(.01, .01)$$
$$\sigma^2 \sim IG(.01, .01)$$

In this model, we assume that the predictors have the same relationship with the response, regardless of team. We specify the coefficients of the predictors (β) like we did because we want to let the data control the posterior. This prior provides the model with very little intuition as to what the true value of (β) is (similar to uniform distribution). To build in the group structure in the model, we specify the team effects (α) as a normal model centered around 0 with the variance equal to the between-team variance. Specifying our model this way encourages shrinkage of team effects, so if we don't have a lot of information, the team effects will be pushed towards zero. We assume all of the parameters that we specify as priors are all independent.

3.2 Model Implementation

Since we cannot obtain a closed form posterior distribution for this model, we have to sample from our posterior using Monte Carlo Markov Chains. However, we do have full conditional distributions in closed form, so we are able to use Gibbs Sampler to generate approximate samples from the posterior. This was extremely helpful when running our JAGS model because it was much quicker and less computationally expensive than using the Metropolis-Hasting algorithm. We are able to use a Gibbs Sampler in R to sample from the posterior using the R package JAGS. The methods to specify a hierarchical linear regression model in R are as followed:

1. Create a data list - The data list includes any variable you want to use in the model. It will include all the predictors and response as well as sample size and amount of predictors.

2. Specify the list of parameters to be monitored - This list includes all the parameters we want JAGS to track. In our model, we specified α, β, σ_α^2, σ^2, and ICC.

3. Specify the initial values for parameters - These values are for the parameters you are looking to track. In our model, we specified a list of 0 for α, a list of 0 for β, and 1 for both σ_α^2 and σ^2.

4. Specify parameters for running JAGS algorithm -

    - number of steps to "tune" the samplers – 10000
    - number of steps to "burn-in" the samplers – 20000
    - number of chains to run – 3

- total number of steps in chains to save – 50000
- number of steps to "thin" – 10
- steps per chain – 166667

5. Initialize the model and use the specified model file to create the model
6. Burn in the model using the specified amount of burn in steps. This allows the samples we retain to already have converged.
7. Run the MCMC algorithm to receive samples from the posterior.
8. Diagnose plot convergence by looking at density plots and trace plots (In Appendix)
9. Extract the posterior samples into a matrix to view model results
10. Assess the model

4 Results

4.1 Predictors' Effects

The results we obtain from the Hierarchical Bayesian Linear Model are similar to the results from the Bayesian Linear Model because the Bayesian Linear model aggregates the predictors together. Since our model specifies random effects for the intercepts and not the slopes, our predictors will be the same and the aggregated intercept will not be too different.

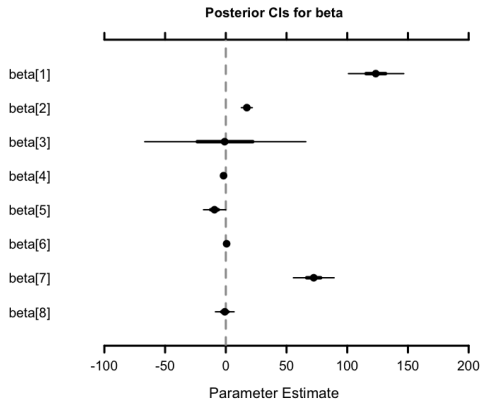


Figure 5: Posterior Credible Intervals: (β)

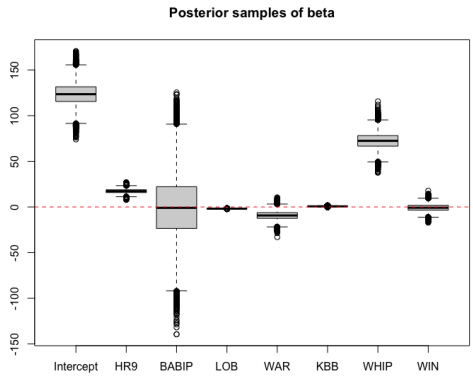


Figure 6: Posterior Samples: (β)

Looking at the posterior credible intervals for β and the boxplot for each β, we can see that *WHIP* seems to have the largest effect on *ERA-*. The variables that don't include zero in their .95 posterior credible interval are *HR/9*, *LOB*, *WAR*, and *WHIP*. This indicates that they are significant in explaining the variance in *ERA-*. The posterior mean for *WHIP* was 72.37313, which means that a 1 unit increase in *WHIP* will lead to a 72.37313 increase in *ERA-*. Although this coefficient is positive, it is a negative effect on the response because an increase in *ERA-* is bad for a pitcher. The posterior mean for *WAR* was -9.323148, so a 1 unit increase in *WAR* will lead to a 9.323148 decrease in *ERA-*. Increasing a pitcher's *WAR* indicates an increase in performance.

4.2 Random Team Effects

Specifying a team layer in the model allows us to quantify how MLB teams' pitchers are performing relative to how they are supposed to be performing based on the predictors in our model. Our hierarchical model allows us to rank every MLB team based on a scale showing how they are underperforming or outperforming compared to the rest of the league.

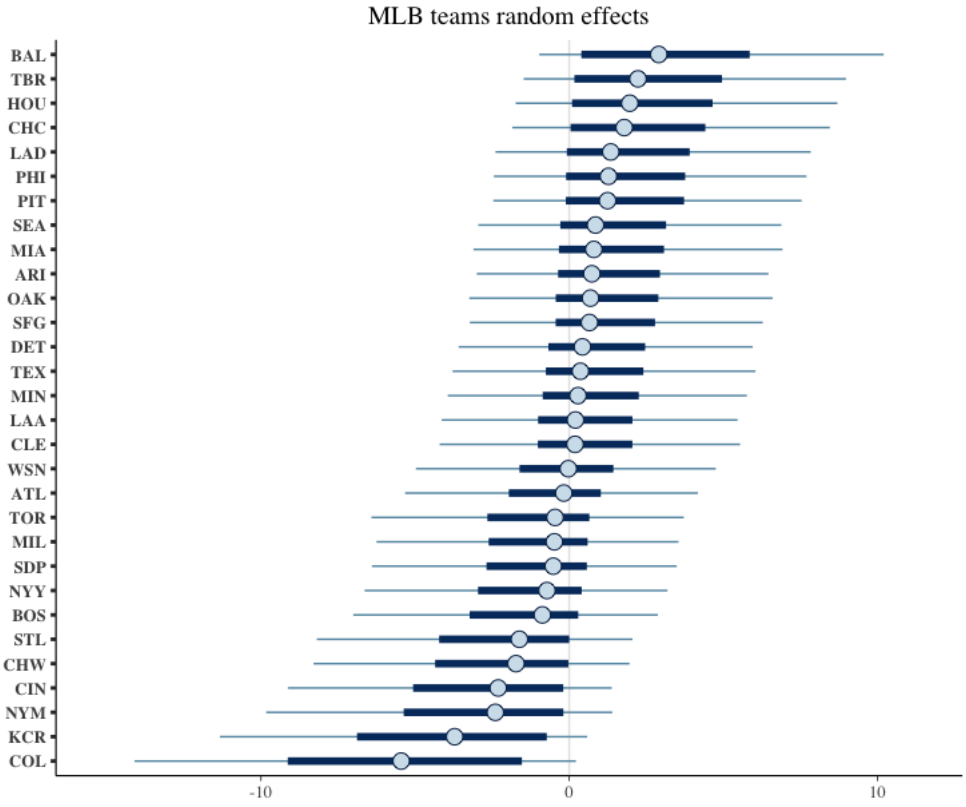


Figure 7: MLB Random Effects

Looking at Figure 7, we can see the posterior intervals for $\alpha_1 ... \alpha_j$. The rankings in Figure 7 indicate that teams near the top have starting pitchers with *ERA-s* that are higher than the baseline predicted *ERA-s*, while teams near the bottom represent starting pitchers with *ERA-s* that are lower than the baseline predicted *ERA-s*. In theory, this means that pitchers on teams near the top are underperforming compared to the baseline, while pitchers on teams near the bottom are outperforming compared to the baseline since it is better for *ERA-* to be lower.

The model could be used to determine how each team's starting pitchers will perform moving forward. If one believes that teams will regress to the overall predicted *ERA-*, then we would see a future decrease in the performance of Colorado (increase in *ERA-*) and a future increase in the performance of Baltimore (decrease in *ERA-*). A team with an *ERA-* of 95 can be interpreted as 5 percent better than the league average, which is 100. Therefore, Colorado's predicted *ERA-* is

significantly lower than the baseline (-5.789455), which can be interpreted as 5.789% better than the predicted league average. This is the largest deviation between team-predicted *ERA-* and overall league predicted *ERA-*. There are no effects bigger than this due to the shrinkage of team effects. Intuitively, it makes sense that these estimates are pushed towards zero because we don't have a lot of pitcher data since MLB games have been underway for only one month.

4.3 Intraclass Correlation

The deviation between team-predicted *ERA-* and the predicted league baseline can be represented by the intraclass correlation (*ICC*). The *ICC* is composed of the between-group variance and the within-group variance:

$$\frac{\sigma_\alpha^2}{\sigma^2 + \sigma_\alpha^2}$$

In our model, the median posterior estimate for intraclass correlation was 0.1044485. We used median for the point estimate because the distribution of the posterior for *ICC* is skewed right (shown in appendix), and median is a more robust estimate for skewed data. The *ICC* of 0.1044485 indicates that 10.44% of the total variation in the model is attributed to team effects. This is a low number and indicates that there is not a large amount of between-team variance. This could be attributed to the shrinkage effect in the model.

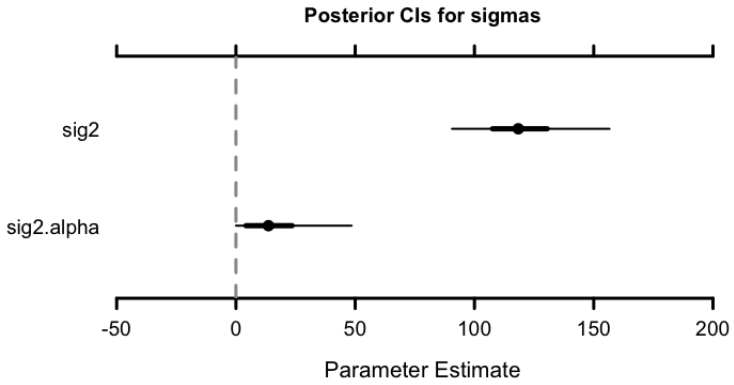


Figure 8: Variance: Between-group vs. Within-group

In Figure 8, we can see the difference between the between-group variance (σ_α^2) and within-group variance (σ^2). This is the reason for the small *ICC*.

5 Conclusion

The Hierarchical Bayesian Linear Regression model constructed in this analysis generates very useful results when evaluating how an MLB team's starting pitchers are performing compared to how they should be performing. The model compares each specific team's predicted *ERA-* to the overall

predicted *ERA-* based on the model predictors. It can be used to show whether teams are underperforming or outperforming the baseline. To compare it other studies conducted, we can look at FanGraph's ERA vs. Expected ERA and FIP vs. Expected FIP. FIP stands for Field Independent Pitching, and it is another way to evaluate pitchers that is scaled to mimic ERA.

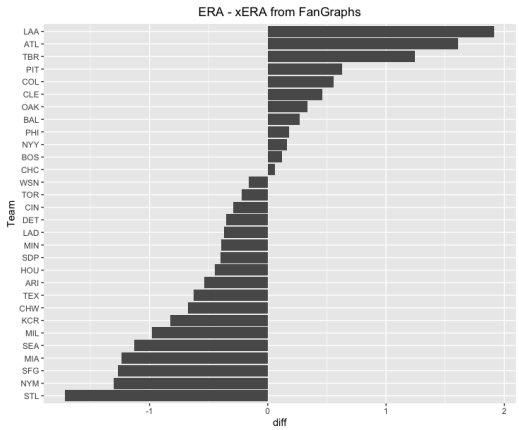


Figure 9: ERA - Expected ERA

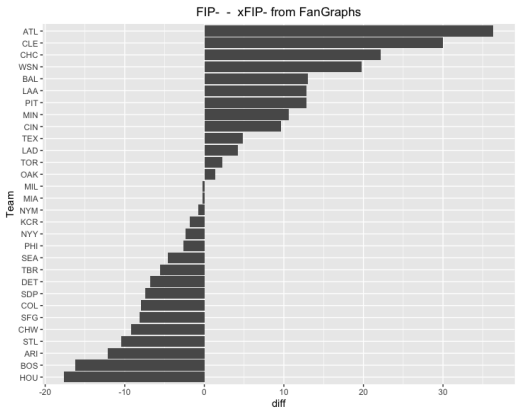


Figure 10: FIP - Expected FIP

The results from Figure 9 and Figure 10 are interpreted differently than the results in this analysis because they are comparing an estimated ERA and FIP to the true ERA and FIP for each team. However, the analysis in this paper and FanGraph's analysis can be compared in that both show whether a team's starting pitching is underperforming or outperforming and by how much compared to other teams. FanGraph gives intuition into how much higher or lower ERA or FIP is than it should be, but our analysis, using *ERA-*, is more interpretable when comparing teams because you can quantify what percentage a certain team is underperforming or outperforming another team. *ERA-* also standardizes ERA by ballpark. Looking at the results from our analysis and FanGraph's, we see many similarities and a few big differences. Teams such as The Chicago Cubs and Tampa Bay Rays seem to be near the top (underperforming), while teams such as The New York Mets and Chicago White Sox appear to be near the bottom (outperforming). However, we see a few key differences between our model and FanGraph's. Atlanta seems to have the largest discrepancy. Our model shows Atlanta near the baseline, while FanGraph's plots show Atlanta severely underperforming. The differences could be due to different variables used to predict ERA or that our response variable is scaled differently. Overall, our model is relatively similar to FanGraph's.

In conclusion, the Hierarchical Bayesian Linear Regression model is a great way to evaluate how well teams are pitching. It provides results that are easily interpretable and accurate. Using hierarchical modeling within the Bayesian framework could be used on many other baseball statistics such as batting average or win probability. The model is very interesting and a great way to interpret baseball statistics.

6 References

[1] https://www.mlb.com/glossary/standard-stats/

[2] https://library.fangraphs.com/pitching/complete-list-pitching/

7 Appendix

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
std = pd.read_csv("std_mlb_csv")
adv = pd.read_csv("adv_mlb_csv")
```

In [3]:
```
# ERA+/ERA = adv['Team','Name','ERA+','xERA']
# ERA+/xERA/'diff' = ERA+/ERA['ERA'] - ERA+/xERA['xERA']
# ERA+/xERA.groupby('Team').mean()
```

In [4]:
```
# FIP/xFIP = adv['Team','Name','FIP-','xFIP-']
# FIP/xFIP/'diff' = FIP/xFIP['FIP-'] - FIP/xFIP['xFIP-']
# FIP/xFIP.groupby('Team').mean().to_csv('FIP_vs_xFIP.csv')
```

In [5]:
```
baseball = pd.merge(std,adv,how = 'inner',on=['Name','playerid'])
```

In [6]:
```
baseball[["BB%","K%","OPS","SO","GS"]] = baseball[["BB%","K%","OPS","GS"]].astype(int)
```

In [7]:
```
baseball[["GS0%_a","GS0%","BB/IP%","K%","BB%"]] = baseball[["GS0%_a","GS0%","BB/IP%","K%","BB%"]].replace('[%]', '', regex=True).astype(float)
```

In [8]:
```
baseball = baseball.drop(baseball.columns[baseball.columns.str.contains('_1y')],axis=1)
```

In [9]:
```
baseball.columns
```

Out[9]:
```
Index(['Name', 'Team_a', 'G', 'IP', 'SO', 'GS', 'TBF', 'WJ%_a', 'BB/K_a',
       'BABIP_a', 'LOB%_a', 'GS0%_a', 'BB/IP%', 'xFA (pi)', 'ERA_a',
       'xERA', 'FIP-_a', 'xFIP-_a', 'WAR', 'playerid', 'K/BB%', 'K%', 'BB%',
       'K-BB%', 'xFA', 'WHIP', 'ERA+', 'FIP-', 'xFIP-', 'K-BB%', 'ERA+_a',
       'FIP+_a', 'xFIP+_a', 'xFIP', 'Team'],
      dtype='object')
```

In [10]:
```
baseball.rename(columns = {'FA/S_a':'FA/S','Team_a':'Team','BB/K_a':'BB/S','WJ%_a':'WJ%','BABIP_a':'BABIP','LOB%_a':'LOB%','ERA_a':'ERA','FIP_a':'FIP','xFIP_a':'xFIP',inplace = True)
```

In [11]:
```
baseball.columns
```

Out[11]:
```
Index(['Name', 'Team', 'G', 'IP', 'SO', 'GS', 'TBF', 'WJ%', 'BB/S', 'BB/IP',
       'BABIP', 'LOB%', 'GS0%', 'BB/IP%', 'xFA (pi)', 'ERA', 'xERA', 'FIP-',
       'xFIP-', 'WAR', 'playerid', 'K/BB%', 'K%', 'BB%', 'K-BB%', 'xFA',
       'WHIP', 'ERA+', 'FIP-', 'xFIP-', 'K-BB%', 'ERA+', 'FIP+', 'xFIP+',
       'xFIP', 'Team'],
      dtype='object')
```

In [12]:
```
baseball = baseball.drop(['xERA','ERA','GS'],axis=1)
```

In [13]:
```
baseball["ERA"] = baseball["R"] / baseball["W"] * baseball["G"])
baseball = baseball.drop(['W','L'],axis=1)
```

In [14]:
```
baseball = baseball.fillna(0)
baseball.isnull().sum()
```

Out[14]:
```
Name        0
Team        0
G           0
IP          0
SO          0
GS          0
TBF         0
WJ%         0
BB/S        0
BB/IP       0
BABIP       0
LOB%        0
GS0%        0
BB/IP%      0
xFA (pi)    0
ERA         0
xERA        0
FIP-        0
xFIP-       0
WAR         0
playerid    0
K/BB%       0
K%          0
BB%         0
K-BB%       0
xFA         0
WHIP        0
ERA+        0
FIP-        0
xFIP-       0
K-BB%       0
ERA+        0
FIP+        0
xFIP+       0
xFIP        0
Team        0
dtype: int64
```

In [15]:
```
baseball.corr()['ERA+']
```

Out[15]:
```
G0          -0.190252
IP          -0.476180
SO          -0.451564
BB/S        0.445544
BB/IP       0.700000
BABIP       0.609750
LOB%        -0.721463
GS0%        -0.000000
BB/IP%      0.406187
xFA (pi)    -0.036043
ERA         0.997026
FIP         0.753731
xFIP        0.503763
WAR         -0.509743
playerid    -0.005991
K/BB        -0.296059
K%          -0.384402
BB%         0.313877
K-BB%       -0.472184
xFA         0.036043
WHIP        0.882555
ERA+        1.000000
FIP-        0.751264
xFIP-       0.507447
FIP+        0.751264
xFIP+       0.504553
W3Sh        -0.596398
Name: ERA+, dtype: float64
```

In [16]:
```
baseball = baseball.drop(['ERA','SIERA','R+P','wFIP+','FIP+','wFIP','FIP','R/9'],axis=1)
```

In [17]:
```
baseball
```

Out[17]:

	Name	Team	GS	IP	BB/9	HR/9	BABIP	LOB%	GB%	HR/FB	...	WAR	playerid	K/BB	K%	BB%	K-BB%	AVG	WHIP	E
0	Gerrit Cole	NYY	6	37.2	0.72	0.24	0.315	78.8	36.5	2.9	...	2.4	13125	20.67	44.3	2.1	42.1	0.176	0.72	
1	Jacob deGrom	NYM	5	35.0	1.03	0.51	0.241	87.2	38.3	7.7	...	2.1	10954	14.75	48.0	3.3	44.7	0.134	0.57	
2	Corbin Burnes	MIL	5	29.1	0.00	0.31	0.273	73.9	53.6	5.3	...	1.8	19361	49.00	45.4	0.0	45.4	0.132	0.55	
3	Tyler Glasnow	TBR	6	37.2	2.63	0.48	0.230	84.6	40.8	6.1	...	1.7	14374	5.09	39.2	7.7	31.5	0.144	0.80	
4	Clayton Kershaw	LAD	6	38.2	1.16	0.70	0.262	81.2	49.0	8.1	...	1.4	2036	7.80	26.0	3.3	22.7	0.207	0.91	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
148	Patrick Corbin	WSN	5	23.1	5.40	2.70	0.294	63.3	41.9	25.9	...	-0.5	9323	1.29	16.5	12.8	3.7	0.290	1.76	
149	Tarik Skubal	DET	4	16.1	6.06	3.86	0.196	75.6	17.0	21.9	...	-0.6	22267	1.18	16.9	14.3	2.6	0.242	1.65	
150	Drew Smyly	ATL	4	19.0	3.32	4.26	0.245	48.8	32.8	27.3	...	-0.6	11760	2.57	25.7	8.0	12.6	0.275	1.53	
151	Logan Allen	CLE	5	15.2	4.02	4.02	0.277	57.3	48.1	36.8	...	-0.6	18555	1.71	16.0	9.3	6.7	0.303	1.72	
152	Kyle Hendricks	CHC	5	22.2	3.18	3.97	0.328	82.1	31.6	35.7	...	-0.7	12049	2.83	19.4	7.4	12.0	0.327	1.76	

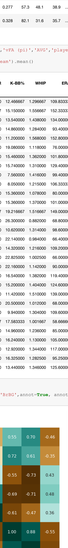
153 rows × 21 columns

In [18]:
```
baseball = baseball.drop(['GS','IP','xFA (pi)','AVG','playerid','K/BB','K%','BB%','BB/S','GB%','BB/IP%'],axis=1)
team_pitching = baseball.groupby('Team').mean()
team_pitching
```

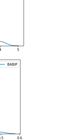
Out[18]:

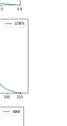
	HR/9	BABIP	LOB%	WAR	K-BB%	WHIP	ERA	WIN%
Team								
ARI	1.058333	0.298833	71.816667	0.300000	12.466667	1.299667	109.833333	0.572222
ATL	2.116333	0.319000	69.116667	0.116667	10.150000	1.553000	152.333333	0.388889
BAL	1.758000	0.307805	75.000000	0.240000	13.540000	1.438000	134.000000	0.450000
BOS	0.836800	0.303400	71.740000	0.540000	14.860000	1.284000	93.400000	0.600000
CHC	1.906933	0.316233	70.433000	0.366667	11.200000	1.356800	102.800000	0.583333
CHW	2.082000	0.294500	69.780000	0.360000	19.080000	1.118000	79.000000	0.683333
CIN	1.416000	0.308400	78.320000	0.350000	15.460000	1.362000	101.800000	0.483333
CLE	2.056000	0.308833	71.250000	0.266667	15.716000	1.310000	129.400000	0.445000
COL	0.996000	0.291000	68.600000	0.320000	7.560000	1.416000	99.400000	0.516667
DET	1.235667	0.284167	69.866667	0.300000	8.050000	1.370000	106.333333	0.290556
HOU	2.854000	0.298000	79.620000	0.360000	15.360000	1.370000	80.200000	0.566667
KCR	1.100000	0.302000	71.840000	0.360000	15.360000	1.370000	101.000000	0.523333
LAA	1.301667	0.329667	63.033333	0.266667	19.216667	1.516667	149.000000	0.361111
LAD	1.160000	0.302200	81.800000	0.660000	26.300000	0.880000	68.800000	0.783333
MIA	1.134000	0.273200	78.380000	0.400000	10.620000	1.314000	98.600000	0.186667
MIL	0.852000	0.252900	78.340000	0.800000	22.140000	0.994000	66.400000	0.700000
MIN	1.584000	0.302250	74.940000	0.350000	14.320000	1.216000	109.200000	0.570000
NYM	0.750000	0.264000	74.150000	0.925000	22.825000	1.020500	68.000000	0.462500
NYY	1.320000	0.291426	76.460000	0.880000	22.160000	1.142000	90.000000	0.526667
OAK	1.020000	0.327600	75.080000	0.300000	16.540000	1.360000	119.400000	0.466000
PHI	1.874000	0.310200	71.340000	0.640000	15.200000	1.424000	122.600000	0.383333
PIT	1.436000	0.296000	69.300000	0.180000	11.420000	1.515000	139.000000	0.300000
SDP	1.766000	0.312400	78.080000	0.540000	20.300000	1.212000	88.000000	0.466000
SEA	1.180000	0.298800	73.220000	0.180000	9.940000	1.530000	101.000000	0.333333
SFG	0.598667	0.258833	83.183333	0.816667	17.583333	1.001667	58.666667	0.750000
STL	0.717000	0.283000	78.140000	0.640000	14.960000	1.136000	85.000000	0.566667
TBR	0.886000	0.267600	68.800000	0.560000	16.240000	1.130000	105.000000	0.419667
TEX	1.513000	0.302200	74.160000	0.140000	12.920000	1.344000	117.000000	0.466667
TOR	1.347000	0.299000	78.673000	0.270000	16.320000	1.282500	95.250000	0.458333
WSN	1.986000	0.259000	73.120000	0.100000	13.440000	1.346000	125.000000	0.383333

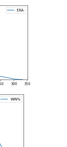
In [19]:
```
# baseball.to_csv('baseball.csv')
fig=plt.figure(figsize = (12,12))
sns.heatmap(baseball.corr(),cmap = 'BrBG',annot=True, annot_kws={"size":12},square=True,fmt = ".2f")
plt.savefig("era_corr")
```



In [20]:
```
def plot_var_kde(col,df):
    ''' kde_kde(col,df)'''
for col in baseball.columns[1:]:
    plot_var_kde(col,baseball)
    plt.show()
plt.style.use('ggplot')
```







In [21]:
```
plot_var_kde('SIERA',baseball)
plt.savefig("ERA_SIERA")
```


In [22]:
```
plot_var_kde('WHIP',baseball)
plt.savefig("ERA_WHIP")
```

In [23]:
```
plot_var_kde('ERA-',baseball)
plt.savefig('ERA')
```

In [ ]:
```

```

```
############################
# Taylor Last
# STAT 4350 FINAL PROJECT
############################
library(ggplot2)
library(rjags)
setwd('/Users/taylorlast/Documents/UGA_FourthYear/STAT_4350/Final Project')
baseball = read.csv('baseball.csv')

#########################################################################
## Hierarchical linear model with team-specific random intercept   ##
##                                                                 ##
#########################################################################
## Load team covariates
attach(baseball)

## More exploratory data analysis
# By team denomination
ggplot(data=baseball,aes(x=Team,y=ERA.))+
  geom_boxplot()+
  theme(axis.text.x = element_text(angle = 90, vjust = 1, hjust = 1),plot.title =
element_text(hjust = 0.5))+
  ggtitle('ERA adjusted by team')

n = dim(baseball)[1]
p = dim(baseball)[2] - 3
n.teams = length(unique(Team))

team_list = as.factor(Team)

## (1) Create data list
dataList <- list(
  "n" = n,
  "p" = p,
  "n.teams" = n.teams,
  "Y" = ERA.,
  "HR9" = HR.9,
  "BABIP" = BABIP,
  "LOB" = LOB.,
  "WAR" = WAR,
  "KBB" = K.BB.,
  "WHIP" = WHIP,
  "WIN" = WIN.,
  "Team" = as.factor(Team))
```

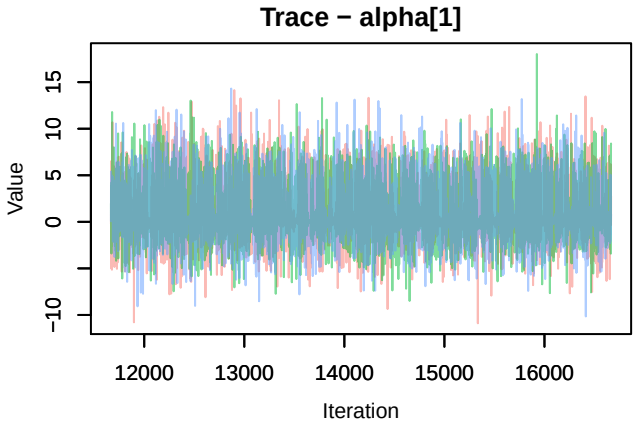
```
## (2) Specify list of parameter(s) to be monitored
parameters <- c("alpha","beta","sig2","sig2.alpha","icc")

## (3) Specify initial values for parameter(s) in Metropolis-Hastings algorithm
initsValues <- list(
  "alpha" = rep(0,n.teams),
  "beta" = rep(0,p),
  "tau2" = 1,
  "tau2.alpha" = 1
)

## (4) Specify parameters for running Metropolis-Hastings algorithm
adaptSteps <- 10000             # number of steps to "tune" the samplers
burnInSteps <- 20000            # number of steps to "burn-in" the samplers
nChains <- 3                  # number of chains to run
numSavedSteps <- 50000          # total number of steps in chains to save
thinSteps <- 10                 # number of steps to "thin" (1 = keep every step)
nIter <- ceiling((numSavedSteps*thinSteps)/nChains)  # steps per chain

## (5) Create, initialize, and adapt the model
# This will require you to create a separate .txt file which specifies
# the model
jagsModel <- jags.model("baseball.txt",
                        data = dataList,
                        inits = initsValues,
                        n.chains = nChains,
                        n.adapt = adaptSteps)

## (6) Burn-in the algorithm
if(burnInSteps>0){
  cat( "Burning in the MCMC chain...\n")
  update(jagsModel, n.iter = burnInSteps)
}

## (7) Run MCMC algorithm
cat("Sampling final MCMC chain...\n" )
codaSamples <- coda.samples(jagsModel,
                            variable.names = parameters,
                            n.iter = nIter,
                            thin = thinSteps)

## (8) Diagnose convergence and plot posterior densities
# par(ask=T)
# plot(codaSamples)
```

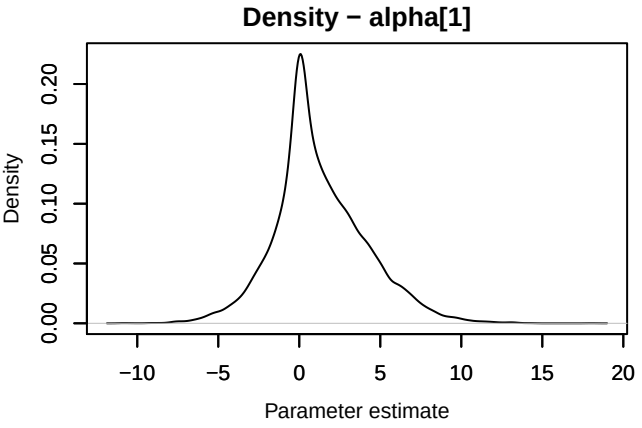
```
## (9) Calculate numerical summaries for the posterior samples
summary(codaSamples)

## (10) Retrieve posterior samples for later use
mcmcChain <- as.matrix(codaSamples)
library(MCMCvis)


MCMCtrace(codaSamples,ISB = FALSE ,
          exact = TRUE,
          pdf = FALSE)

# Examine the posterior distribution of the team random effect
alphaSamples <- matrix(NA, dim(mcmcChain)[1], n.teams)
for(i in 1:n.teams){
  alphaSamples[,i]  <- mcmcChain[, paste("alpha[",i,"]", sep="")]
}
levels(team_list)
par(mfrow=c(1,1), ask=F)
boxplot(as.data.frame(alphaSamples),
        names=as.character(1:n.teams),
        main="Posterior samples of alphas",
        xlab="Team")
abline(h=0)

teams_df = as.matrix(data.frame(num = c(1:30),team = sort(unique(Team))))

rank.alpha <- matrix(NA, dim(mcmcChain)[1], n.teams)
for(i in 1:dim(mcmcChain)[1]){
  rank.alpha[i,] <- rank(alphaSamples[i,])
}
avg.rank <- rank(apply(rank.alpha, 2, mean))


# par(mfrow=c(1,1), ask=F)
# plot(c(-15,15), c(1,30), type='n', axes=F, xlab="", ylab="", main="Team Rankings")
# axis(1,at=seq(-20, 20, length=5))
# for(i in 1:n.teams){
#   lines(quantile(alphaSamples[,i], c(0.025,0.975)), rep(avg.rank[i],2))
#   points(mean(alphaSamples[,i]), avg.rank[i], pch=19)
#   text(15, avg.rank[i], i, cex=.5)
# }

# ranked_teams = vector()
#
```

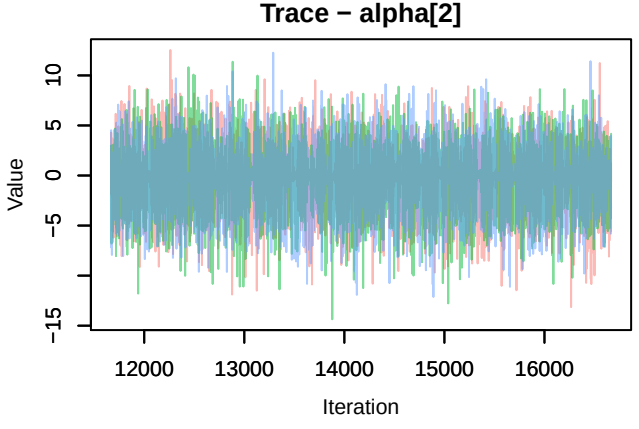
```
#
# for(i in 1:n.teams){
#   for(j in 1:n.teams){
#     if (avg.rank[j] == i){
#       ranked_teams[i]<- teams_df[j,2]
#     }
#   }
# }
# ranked_teams

ranked_teams_idx = vector()
for(i in 1:n.teams){
  for(j in 1:n.teams){
    if (avg.rank[j] == i){
      ranked_teams_idx[i]<- teams_df[j,1]
    }
  }
}
ranked_teams_idx = trimws(ranked_teams_idx)

# Sorted Alphas
temp_vec=vector()
for(i in 1:n.teams){
  temp_vec[i] = paste0('alpha[',ranked_teams_idx[i],']')
}

library(bayesplot)

# Plot the alphas to see random effects
mcmc_intervals(codaSamples,regex_pars = '^[alpha]')+
  scale_y_discrete(
    labels =
      c('alpha[1]' = levels(team_list)[1],
        'alpha[2]' = levels(team_list)[2],
        'alpha[3]' = levels(team_list)[3],
        'alpha[4]' = levels(team_list)[4],
        'alpha[5]' = levels(team_list)[5],
        'alpha[6]' = levels(team_list)[6],
        'alpha[7]' = levels(team_list)[7],
        'alpha[8]' = levels(team_list)[8],
        'alpha[9]' = levels(team_list)[9],
        'alpha[10]' = levels(team_list)[10],
        'alpha[11]' = levels(team_list)[11],
        'alpha[12]' = levels(team_list)[12],
        'alpha[13]' = levels(team_list)[13],
```

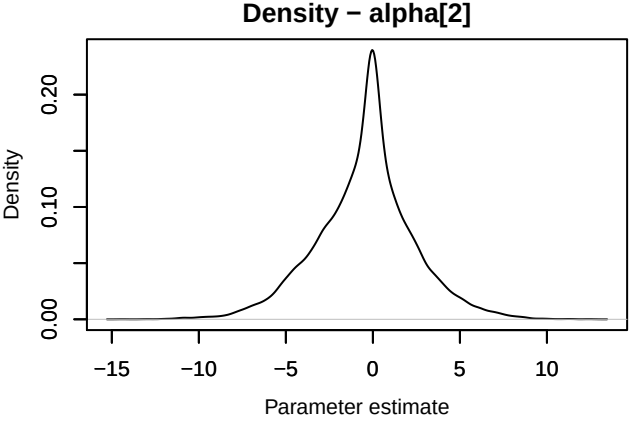
```
        'alpha[14]' = levels(team_list)[14],
        'alpha[15]' = levels(team_list)[15],
        'alpha[16]' = levels(team_list)[16],
        'alpha[17]' = levels(team_list)[17],
        'alpha[18]' = levels(team_list)[18],
        'alpha[19]' = levels(team_list)[19],
        'alpha[20]' = levels(team_list)[20],
        'alpha[21]' = levels(team_list)[21],
        'alpha[22]' = levels(team_list)[22],
        'alpha[23]' = levels(team_list)[23],
        'alpha[24]' = levels(team_list)[24],
        'alpha[25]' = levels(team_list)[25],
        'alpha[26]' = levels(team_list)[26],
        'alpha[27]' = levels(team_list)[27],
        'alpha[28]' = levels(team_list)[28],
        'alpha[29]' = levels(team_list)[29],
        'alpha[30]' = levels(team_list)[30]),
    limits = c(temp_vec)
  )+
  ggtitle('MLB teams random effects')+
  theme(plot.title = element_text(hjust = 0.5))

era.xera = read.csv('era_vs_expected.csv')
era.xera = era.xera[order(era.xera$diff),]
era.xera$Team <- factor(era.xera$Team, levels = era.xera$Team[order(era.xera$diff)])
ggplot(data = era.xera, aes(x = Team,y=diff))+
  geom_bar(stat='identity')+
  coord_flip()+
  ggtitle('ERA - xERA from FanGraphs')+
  theme(plot.title = element_text(hjust = 0.5))


fip.xfip = read.csv('FIP_vs_xFIP.csv')
fip.xfip = fip.xfip[order(fip.xfip$diff),]
fip.xfip$Team <- factor(fip.xfip$Team, levels = fip.xfip$Team[order(fip.xfip$diff)])
ggplot(data = fip.xfip, aes(x = Team,y=diff))+
  geom_bar(stat='identity')+
  coord_flip()+
  ggtitle('FIP-  -  xFIP- from FanGraphs')+
  theme(plot.title = element_text(hjust = 0.5))



# Boxplot of posteriors for beta
boxplot(mcmcChain[,31:38], main = "Posterior samples of beta",
```

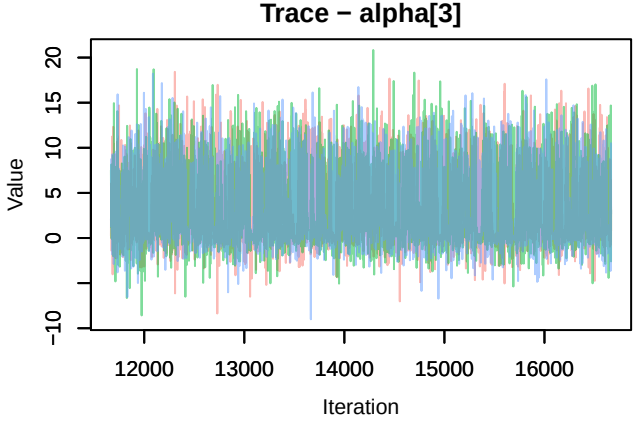
```
        names=c("Intercept",
                "HR9",
                "BABIP",
                "LOB",
                "WAR",
                "KBB",
                "WHIP",
                "WIN"))
abline(h = 0, lty = 2, col="red")

# 50% (thin line) and 95% (thick line) credible interval "caterpillar" plots
# dot is the posterior median
MCMCplot(codaSamples, params = "beta",
         main = "Posterior CIs for beta")

MCMCplot(codaSamples, params = c("sig2",'sig2.alpha'),
         main = "Posterior CIs for sigmas")

# Posterior for intraclass correlation
plot(codaSamples[,39])
median(mcmcChain[,39])
mean(mcmcChain[,3])
mean(mcmcChain[,9])
mean(mcmcChain[,35])
```

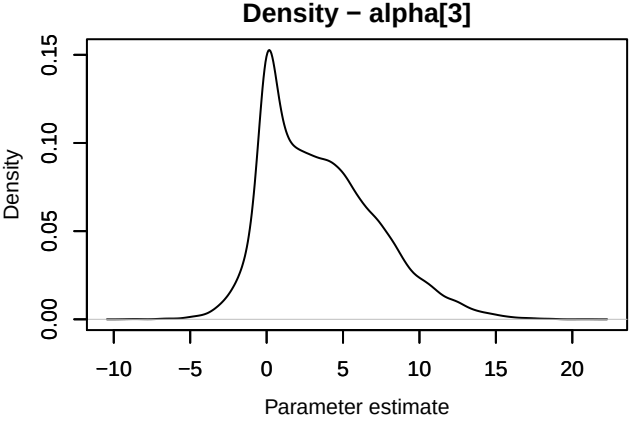

Trace − alpha[1]
Value
Iteration
12000
13000
14000
15000
16000
−10
0
5
10
15

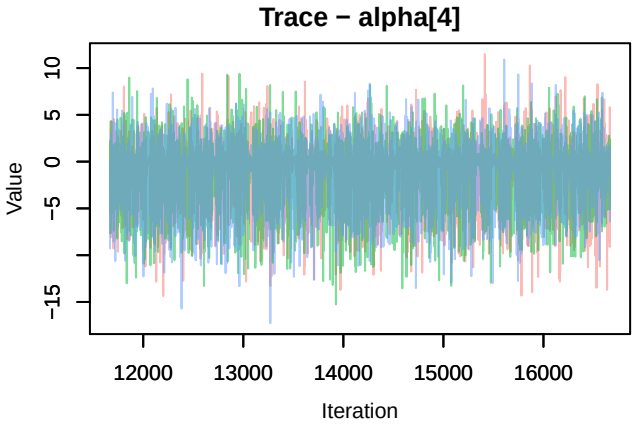

Density − alpha[1]
Density
Parameter estimate
0.00
0.05
0.10
0.15
0.20
−10
−5
0
5
10
15
20

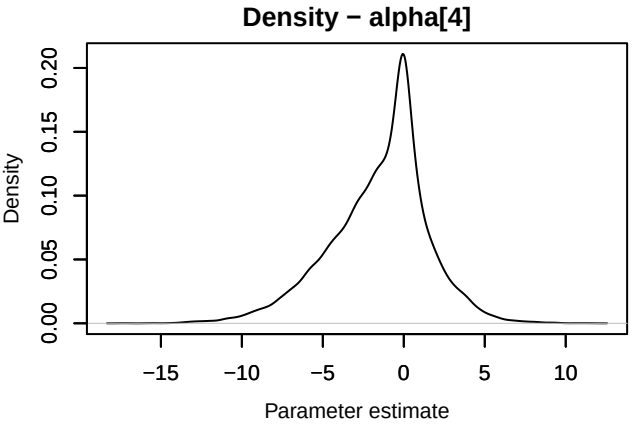

Trace − alpha[2]
Value
Iteration
12000
13000
14000
15000
16000
−15
−5
0
5
10

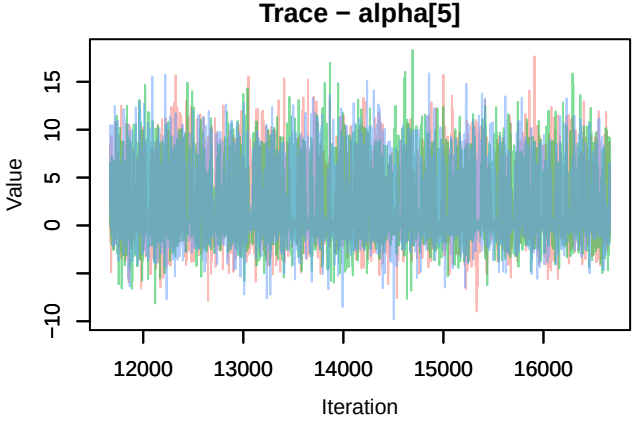

Density − alpha[2]
Density
Parameter estimate
0.00
0.10
0.20
−15
−10
−5
0
5
10

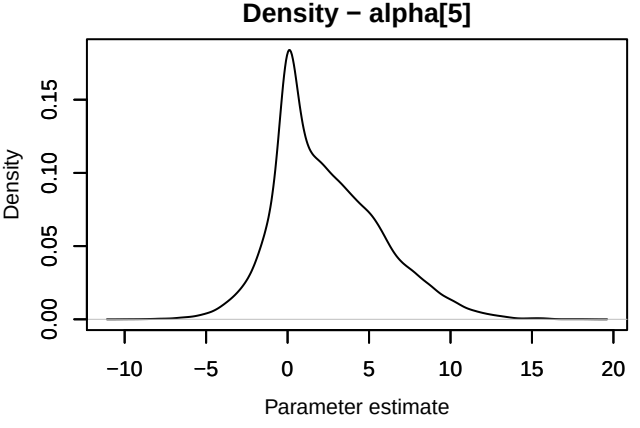

Trace − alpha[3]
Value
Iteration
12000
13000
14000
15000
16000
−10
0
5
10
15
20

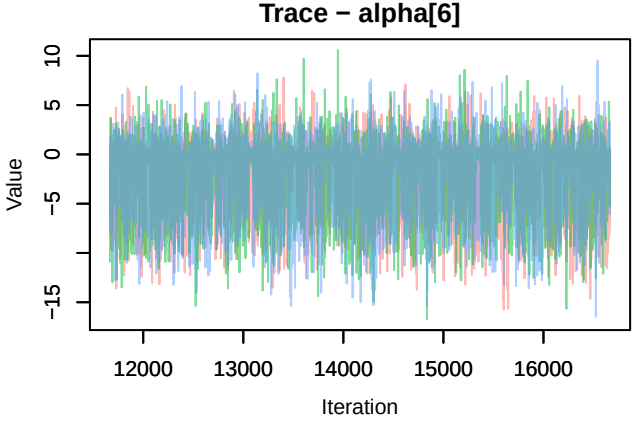

Density − alpha[3]
Density
Parameter estimate
0.00
0.05
0.10
0.15
−10
−5
0
5
10
15
20

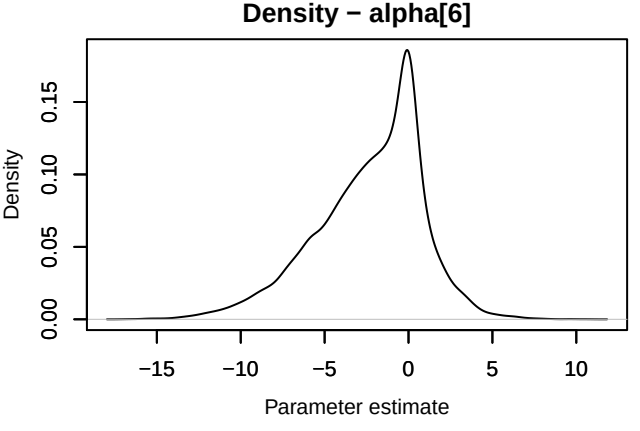

Trace − alpha[4]
Value
10
5
0
−5
−15
12000
13000
14000
15000
16000
Iteration

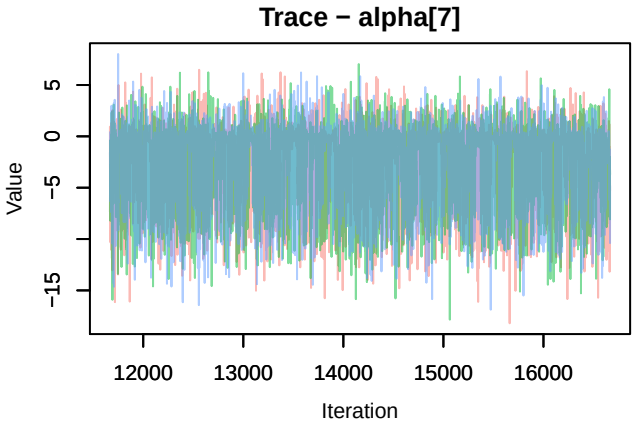

Density − alpha[4]
Density
0.20
0.15
0.10
0.05
0.00
−15
−10
−5
0
5
10
Parameter estimate

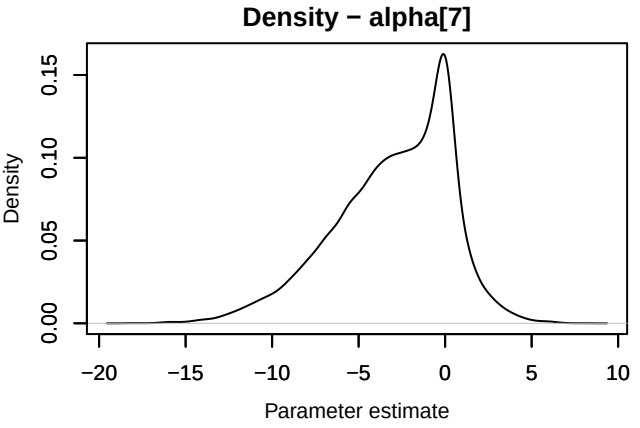

Trace − alpha[5]
Value
15
10
5
0
−10
12000
13000
14000
15000
16000
Iteration

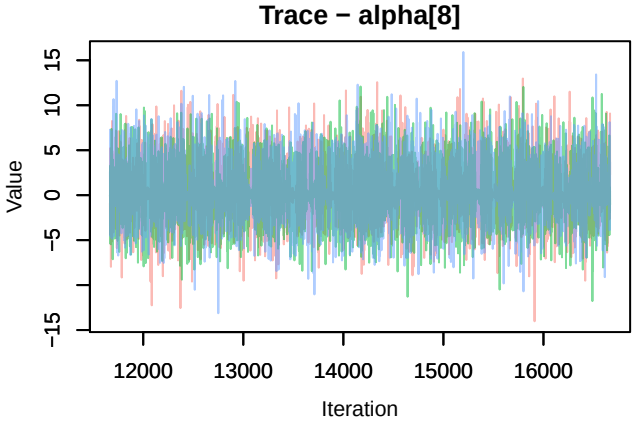

Density − alpha[5]
Density
0.15
0.10
0.05
0.00
−10
−5
0
5
10
15
20
Parameter estimate

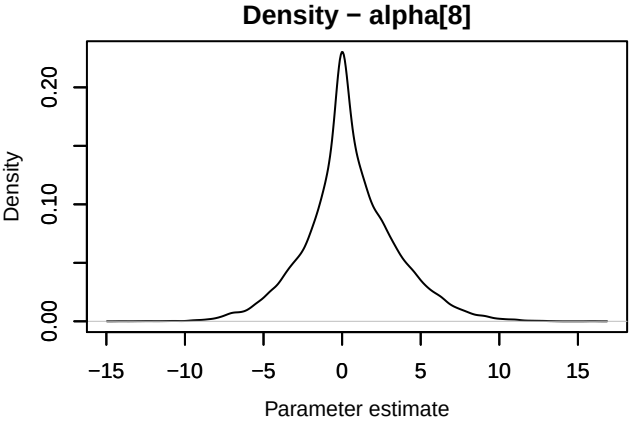

Trace − alpha[6]
Value
10
5
0
−5
−15
12000
13000
14000
15000
16000
Iteration

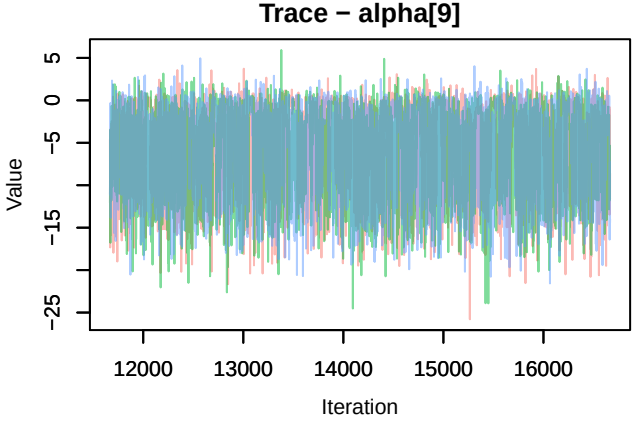

Density − alpha[6]
Density
0.15
0.10
0.05
0.00
−15
−10
−5
0
5
10
Parameter estimate

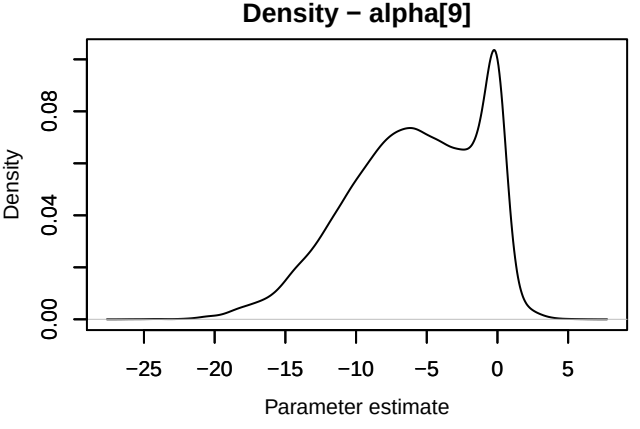

Trace − alpha[7]
Value
Iteration
12000
13000
14000
15000
16000
5
0
−5
−15

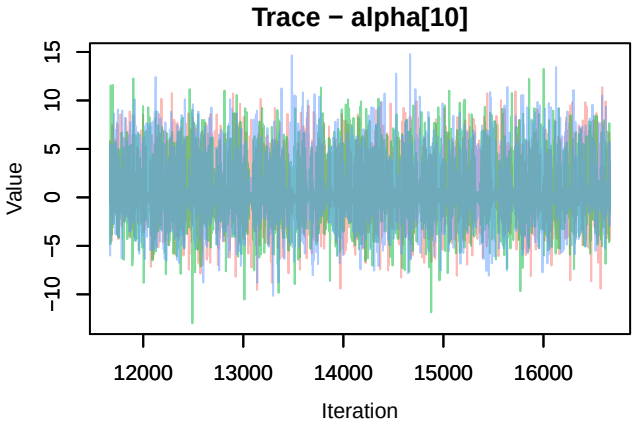

Density − alpha[7]
Density
Parameter estimate
0.00
0.05
0.10
0.15
−20
−15
−10
−5
0
5
10

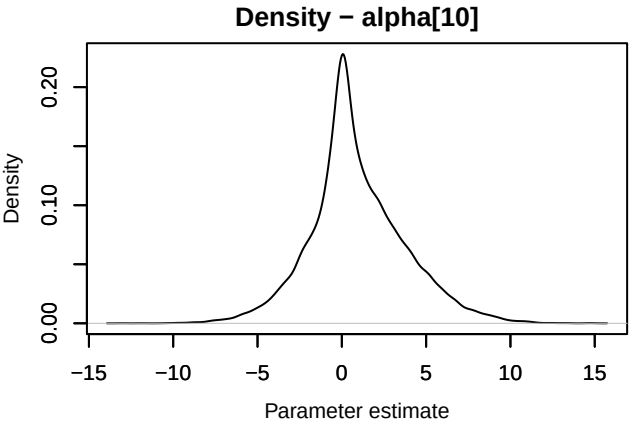

Trace − alpha[8]
Value
Iteration
12000
13000
14000
15000
16000
15
10
5
0
−5
−15

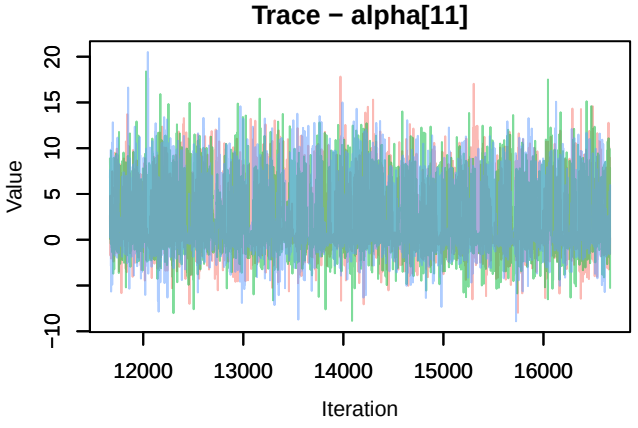

Density − alpha[8]
Density
Parameter estimate
0.00
0.10
0.20
−15
−10
−5
0
5
10
15

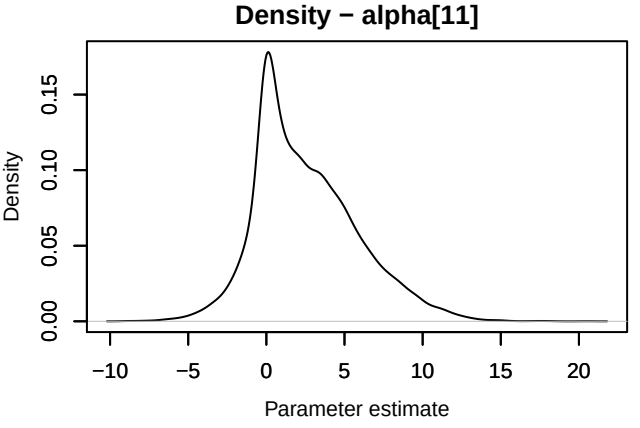

Trace − alpha[9]
Value
Iteration
12000
13000
14000
15000
16000
5
0
−5
−15
−25

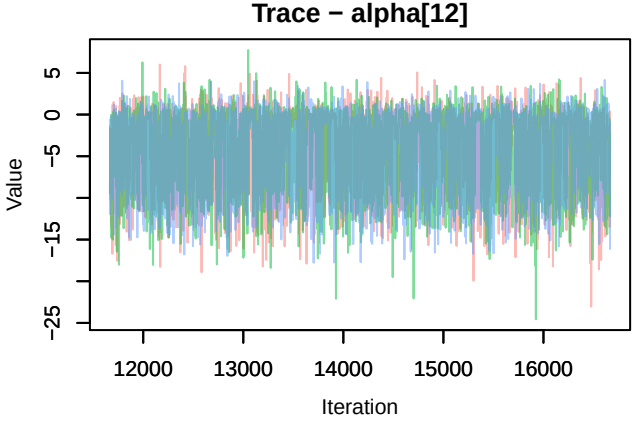

Density − alpha[9]
Density
Parameter estimate
0.00
0.04
0.08
−25
−20
−15
−10
−5
0
5

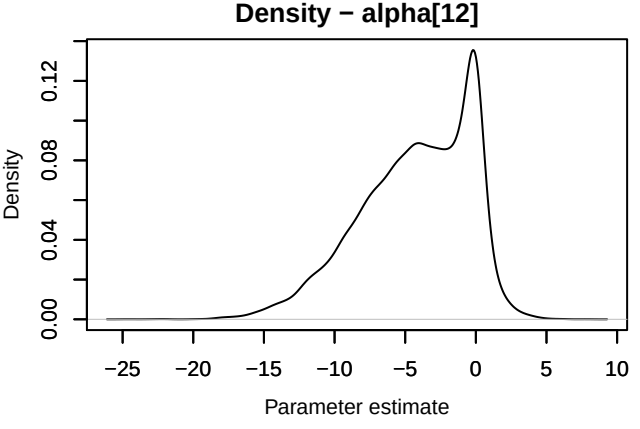

Trace − alpha[10]
Value
Iteration
12000
13000
14000
15000
16000
−10
−5
0
5
10
15

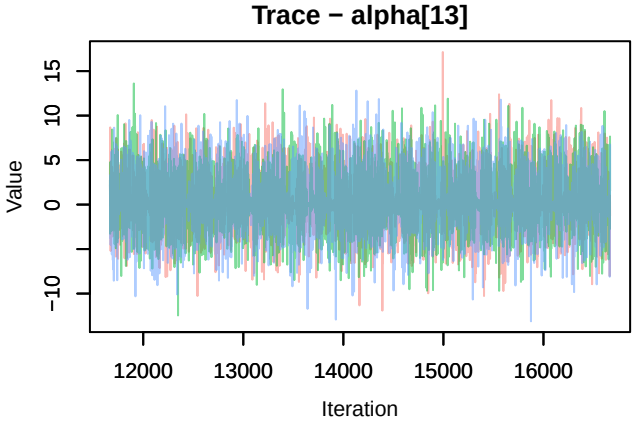

Density − alpha[10]
Density
Parameter estimate
0.00
0.10
0.20
−15
−10
−5
0
5
10
15

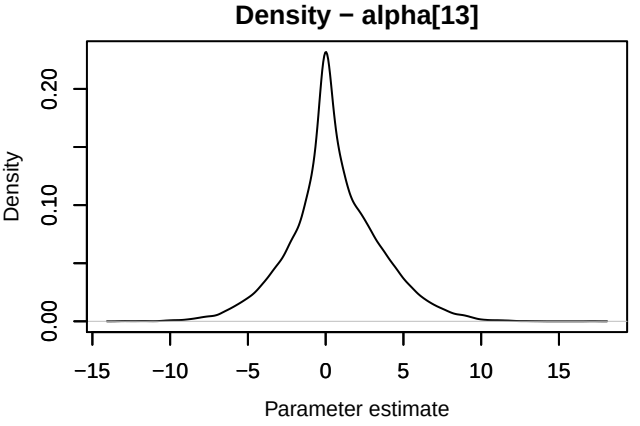

Trace − alpha[11]
Value
Iteration
12000
13000
14000
15000
16000
−10
0
5
10
15
20

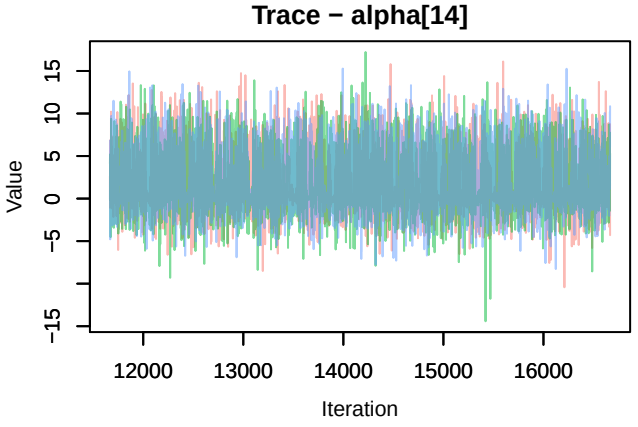

Density − alpha[11]
Density
Parameter estimate
0.00
0.05
0.10
0.15
−10
−5
0
5
10
15
20

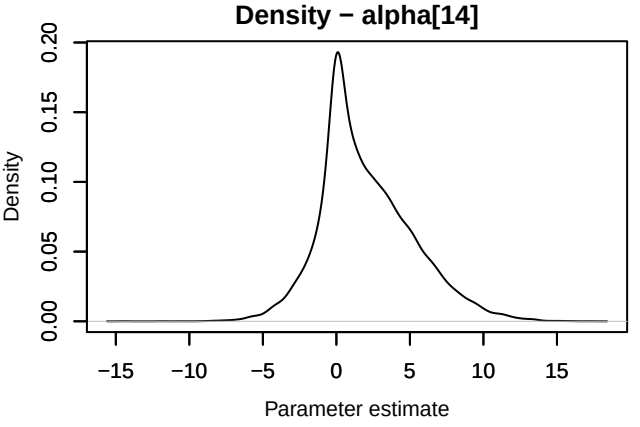

Trace − alpha[12]
Value
Iteration
12000
13000
14000
15000
16000
−25
−15
−5
0
5

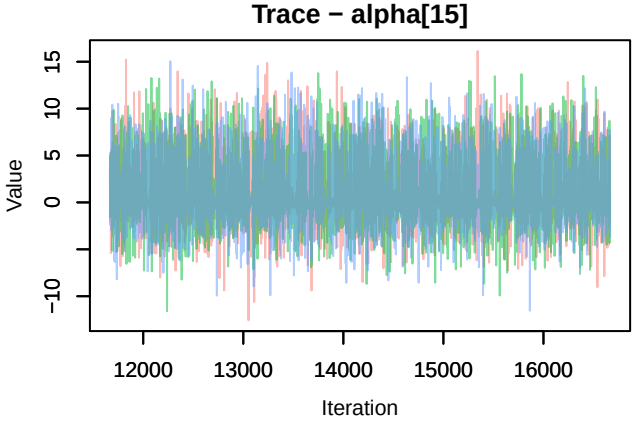

Density − alpha[12]
Density
Parameter estimate
0.00
0.04
0.08
0.12
−25
−20
−15
−10
−5
0
5
10

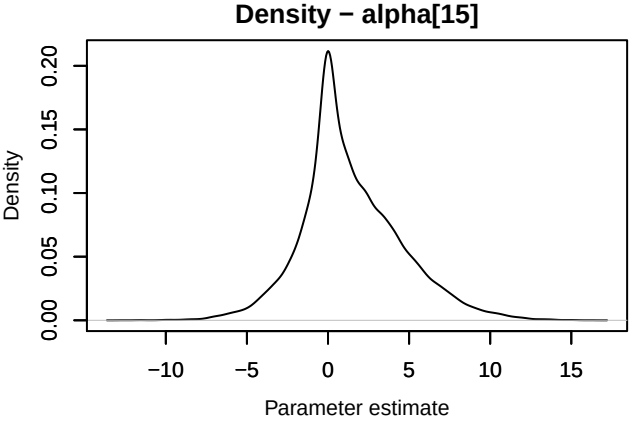

Trace − alpha[13]
Value
Iteration
12000
13000
14000
15000
16000
−10
0
5
10
15

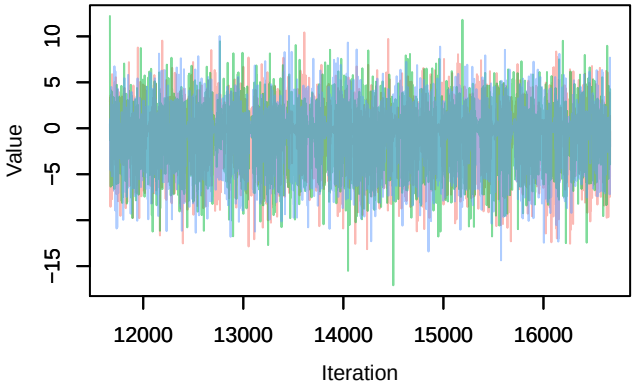

Density − alpha[13]
Density
Parameter estimate
0.00
0.10
0.20
−15
−10
−5
0
5
10
15

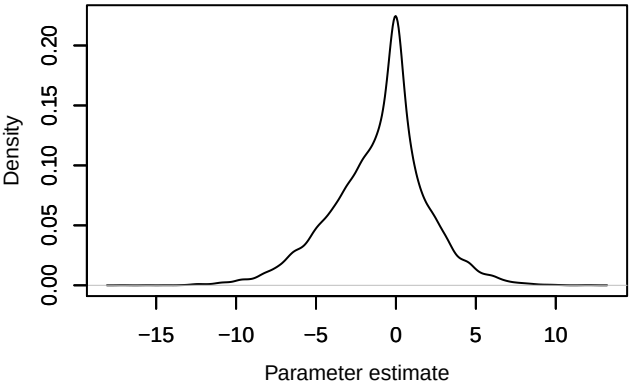

Trace − alpha[14]
Value
Iteration
12000
13000
14000
15000
16000
−15
−5
0
5
10
15

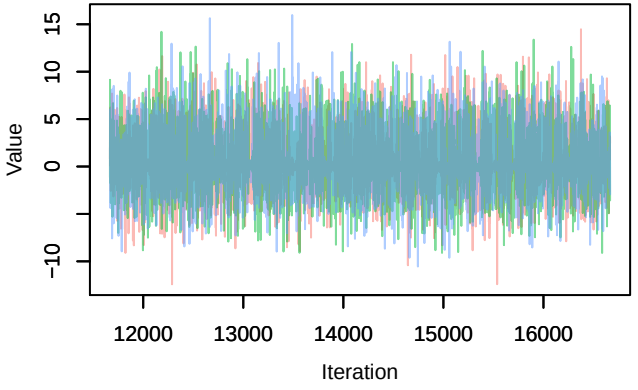

Density − alpha[14]
Density
Parameter estimate
0.00
0.05
0.10
0.15
0.20
−15
−10
−5
0
5
10
15

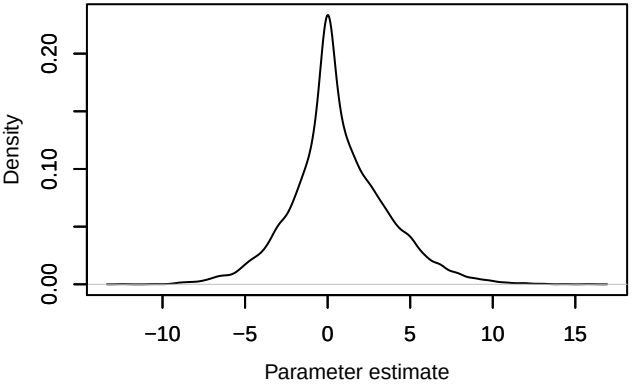

Trace − alpha[15]
Value
Iteration
12000
13000
14000
15000
16000
−10
0
5
10
15

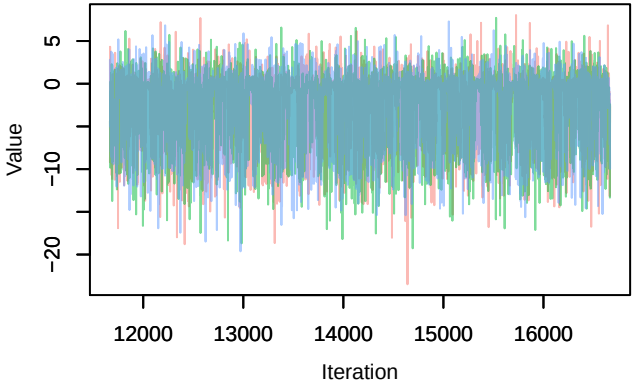

Density − alpha[15]
Density
Parameter estimate
0.00
0.05
0.10
0.15
0.20
−10
−5
0
5
10
15

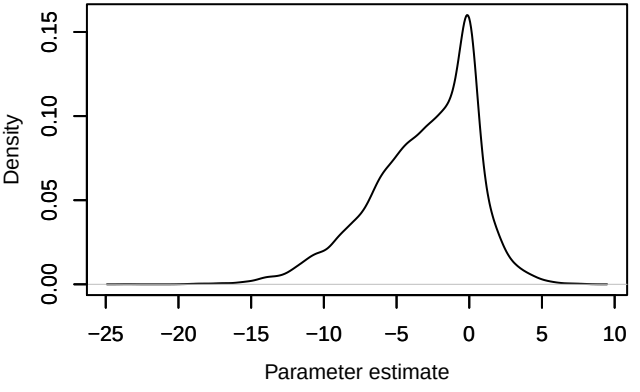

Trace − alpha[16]
Value
10
5
0
−5
−15
12000
13000
14000
15000
16000
Iteration

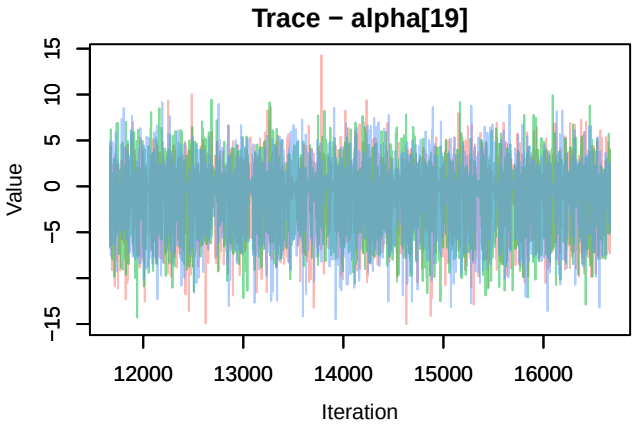

Density − alpha[16]
Density
0.20
0.15
0.10
0.05
0.00
−15
−10
−5
0
5
10
Parameter estimate

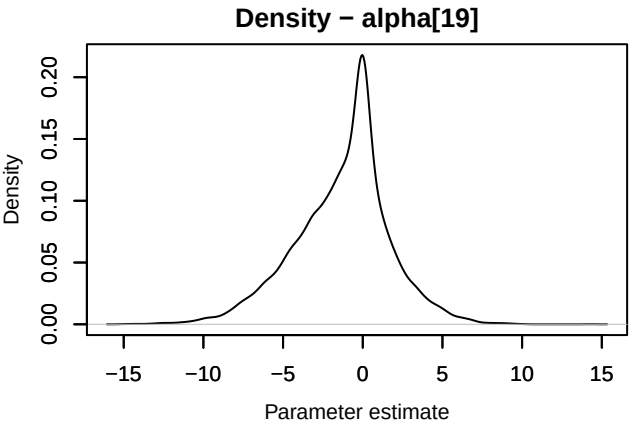

Trace − alpha[17]
Value
15
10
5
0
−10
12000
13000
14000
15000
16000
Iteration

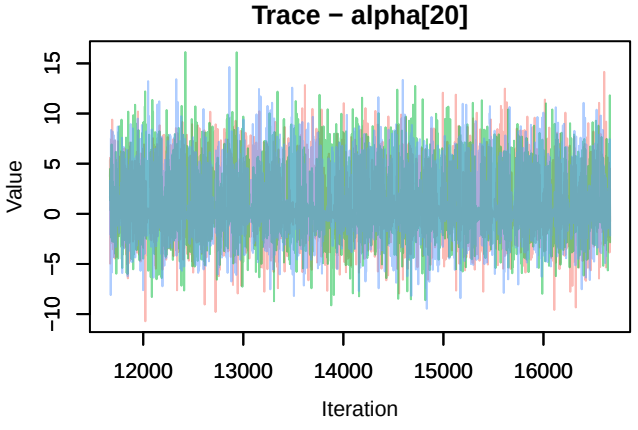

Density − alpha[17]
Density
0.20
0.10
0.00
−10
−5
0
5
10
15
Parameter estimate

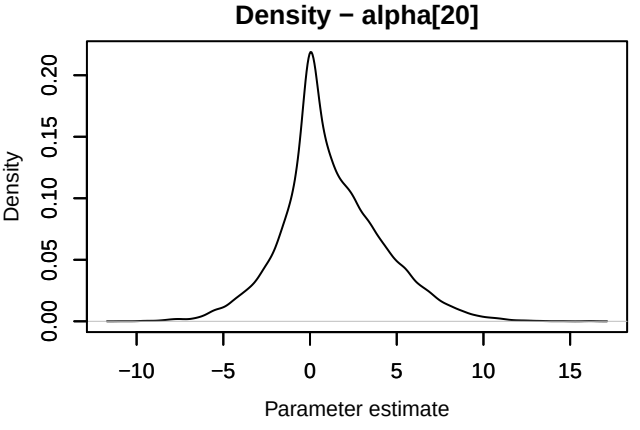

Trace − alpha[18]
Value
5
0
−10
−20
12000
13000
14000
15000
16000
Iteration

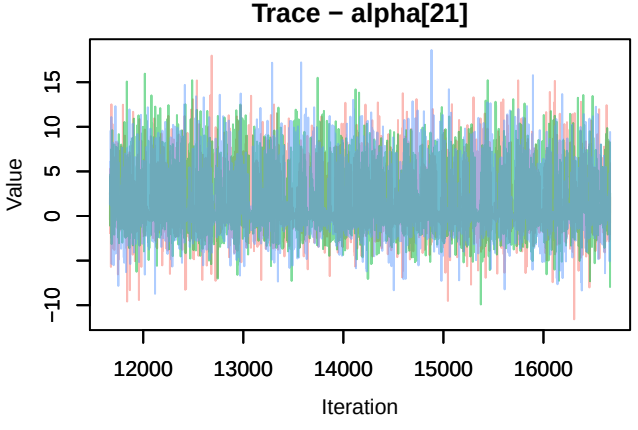

Density − alpha[18]
Density
0.15
0.10
0.05
0.00
−25
−20
−15
−10
−5
0
5
10
Parameter estimate

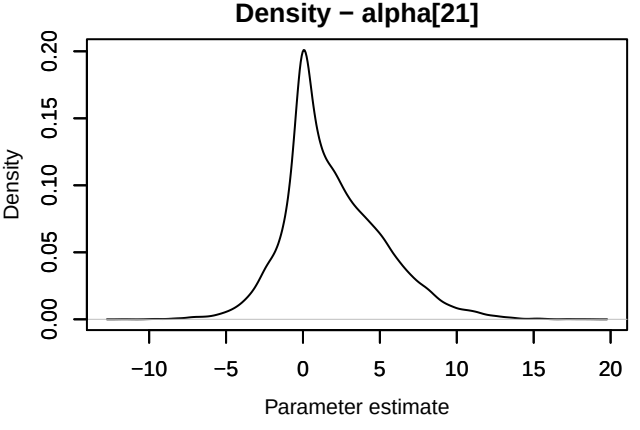

Trace − alpha[19]
Value
Iteration
12000
13000
14000
15000
16000

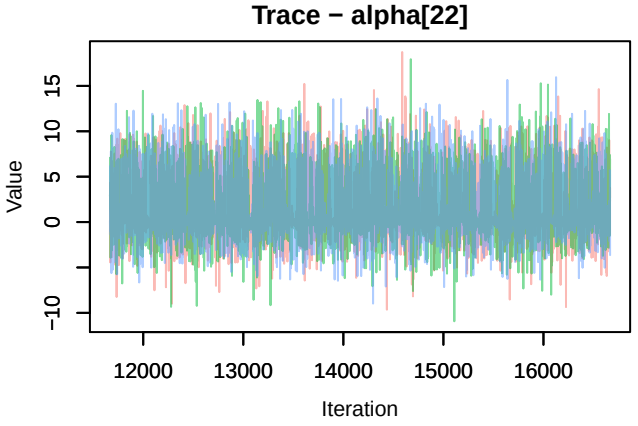

Density − alpha[19]
Density
Parameter estimate

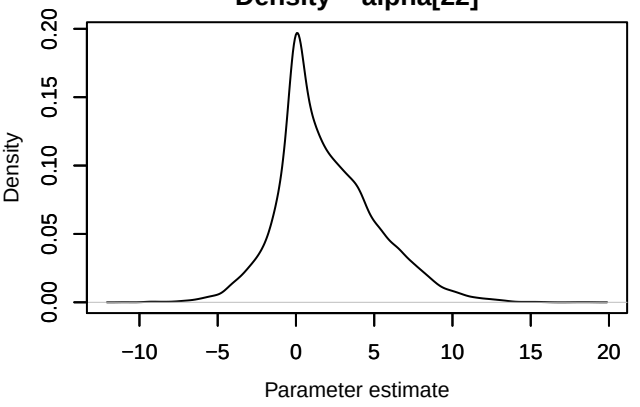

Trace − alpha[20]
Value
Iteration
12000
13000
14000
15000
16000

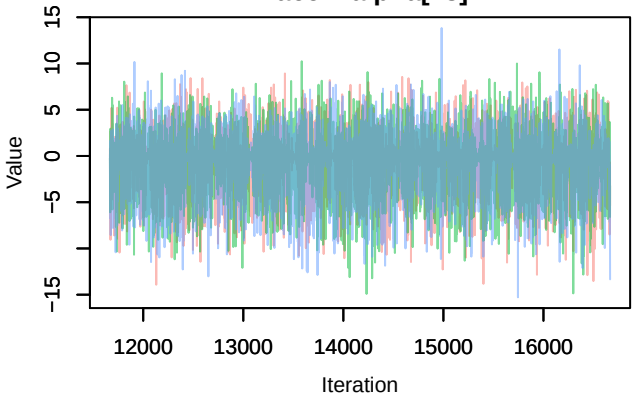

Density − alpha[20]
Density
Parameter estimate

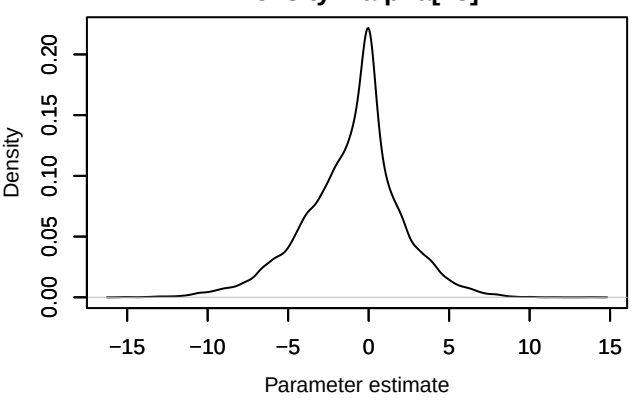

Trace − alpha[21]
Value
Iteration
12000
13000
14000
15000
16000

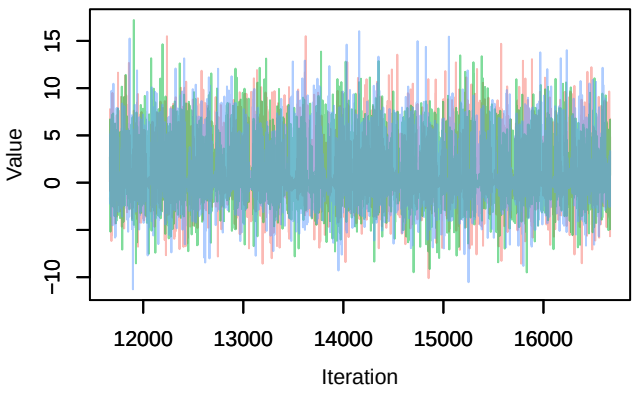

Density − alpha[21]
Density
Parameter estimate

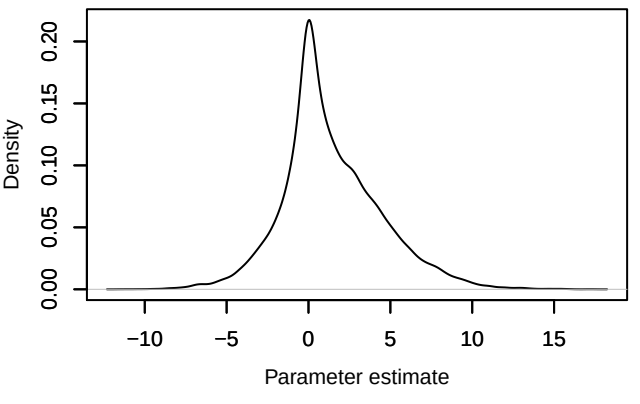

Trace − alpha[22]
Value
Iteration
12000
13000
14000
15000
16000
−10
0
5
10
15

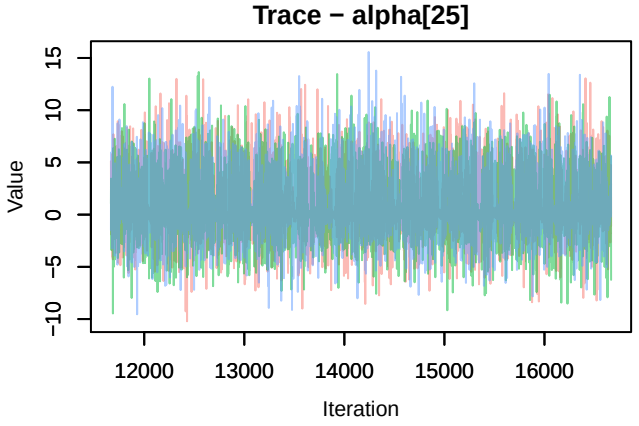

Density − alpha[22]
Density
Parameter estimate
0.00
0.05
0.10
0.15
0.20
−10
−5
0
5
10
15
20

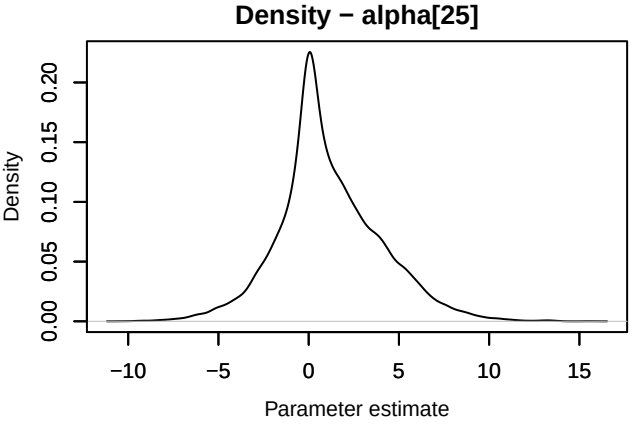

Trace − alpha[23]
Value
Iteration
12000
13000
14000
15000
16000
−15
−5
0
5
10
15

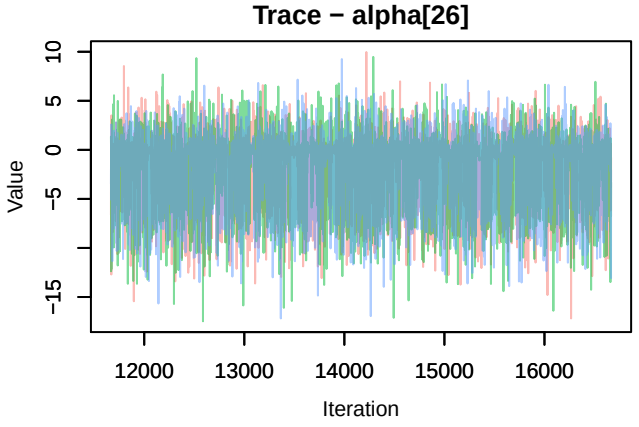

Density − alpha[23]
Density
Parameter estimate
0.00
0.05
0.10
0.15
0.20
−15
−10
−5
0
5
10
15

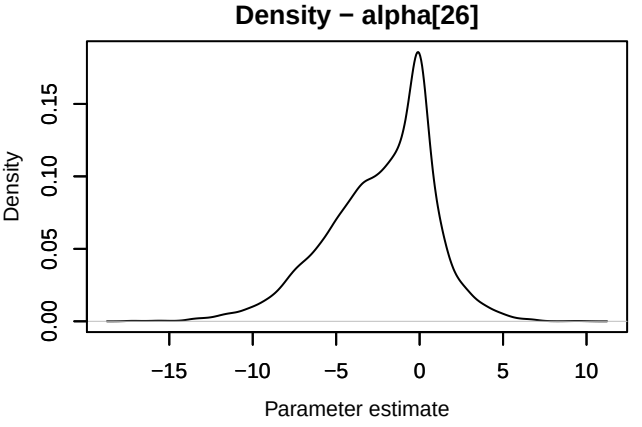

Trace − alpha[24]
Value
Iteration
12000
13000
14000
15000
16000
−10
0
5
10
15

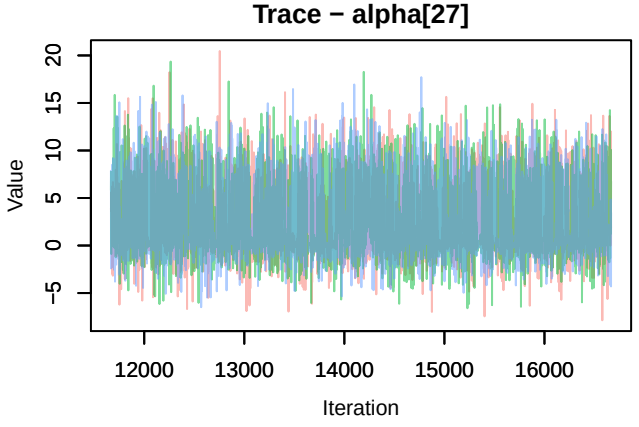

Density − alpha[24]
Density
Parameter estimate
0.00
0.05
0.10
0.15
0.20
−10
−5
0
5
10
15

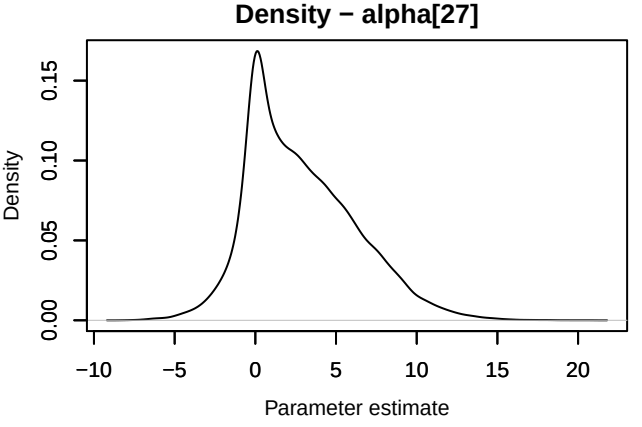

Trace − alpha[25]
Value
Iteration
12000
13000
14000
15000
16000

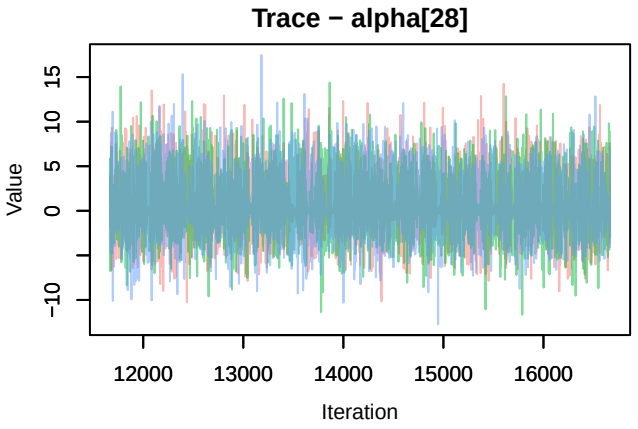

Density − alpha[25]
Density
Parameter estimate

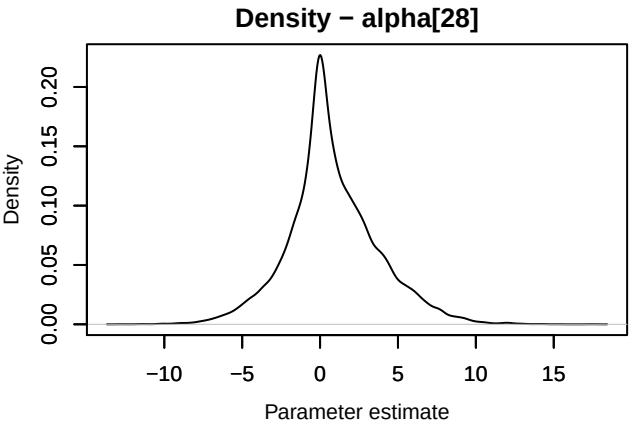

Trace − alpha[26]
Value
Iteration
12000
13000
14000
15000
16000

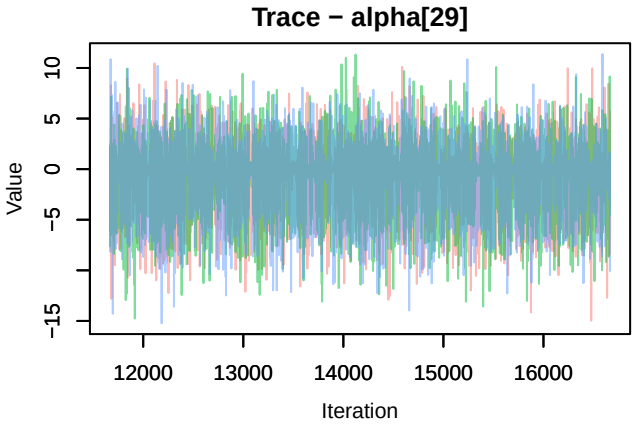

Density − alpha[26]
Density
Parameter estimate

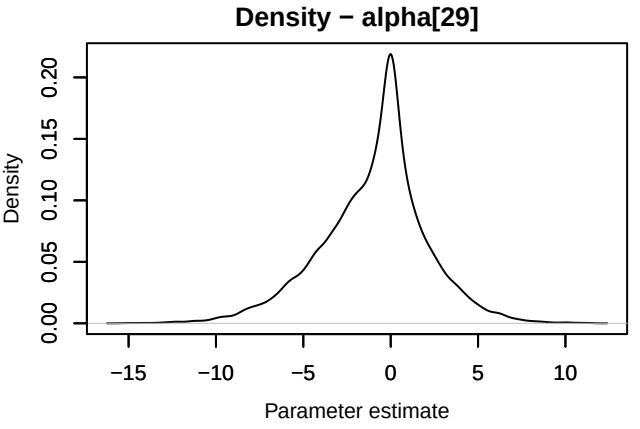

Trace − alpha[27]
Value
Iteration
12000
13000
14000
15000
16000

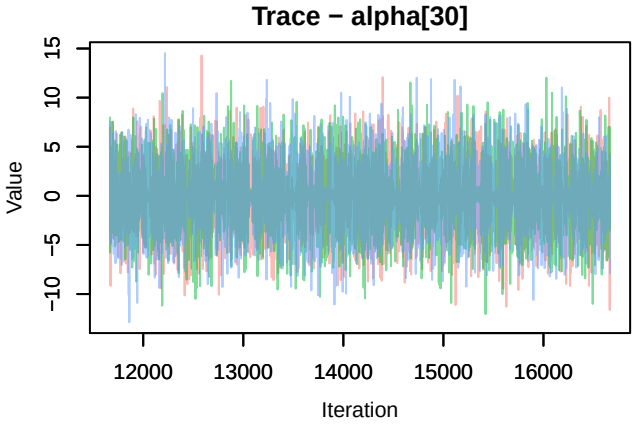

Density − alpha[27]
Density
Parameter estimate

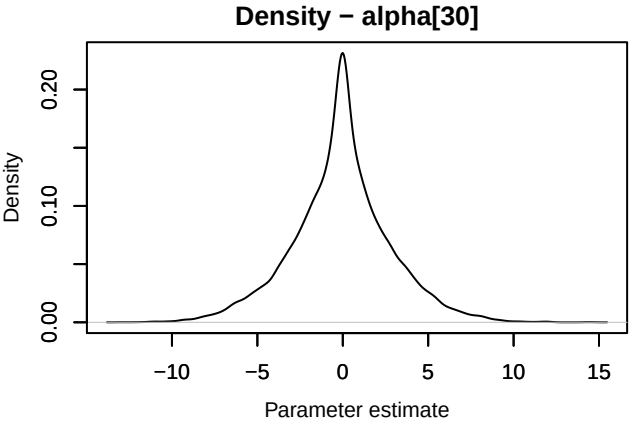

Trace − alpha[28]
Value
Iteration
12000
13000
14000
15000
16000

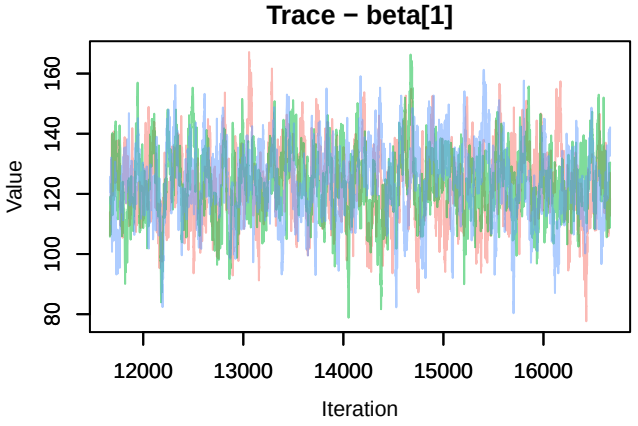

Density − alpha[28]
Density
Parameter estimate

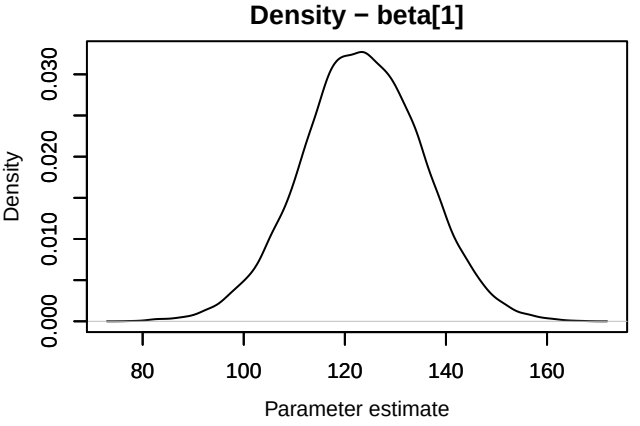

Trace − alpha[29]
Value
Iteration
12000
13000
14000
15000
16000

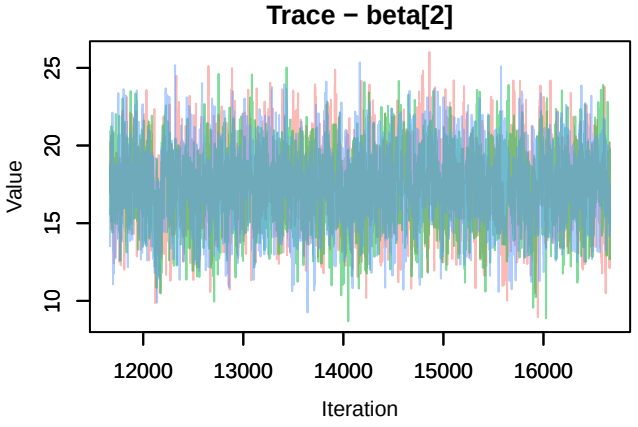

Density − alpha[29]
Density
Parameter estimate

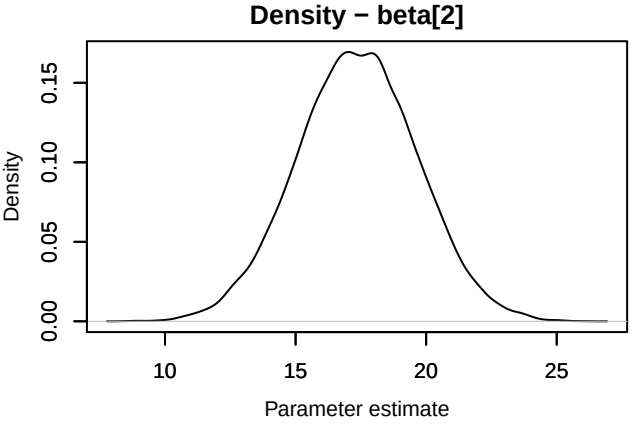

Trace − alpha[30]
Value
Iteration
12000
13000
14000
15000
16000

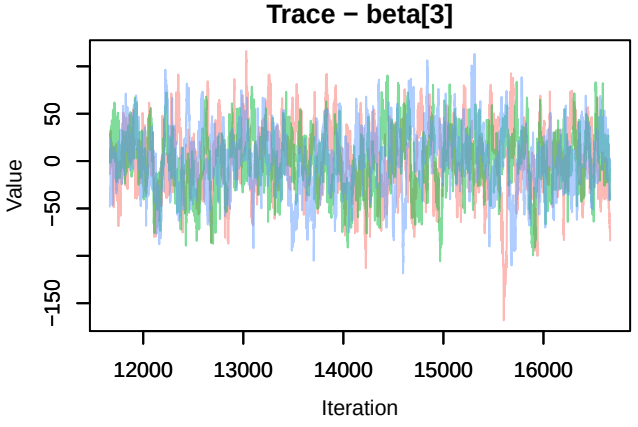

Density − alpha[30]
Density
Parameter estimate

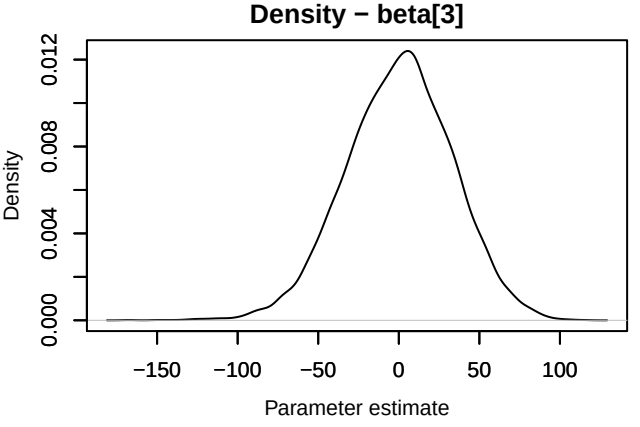

Trace − beta[1]
Value
80
100
120
140
160
12000
13000
14000
15000
16000
Iteration

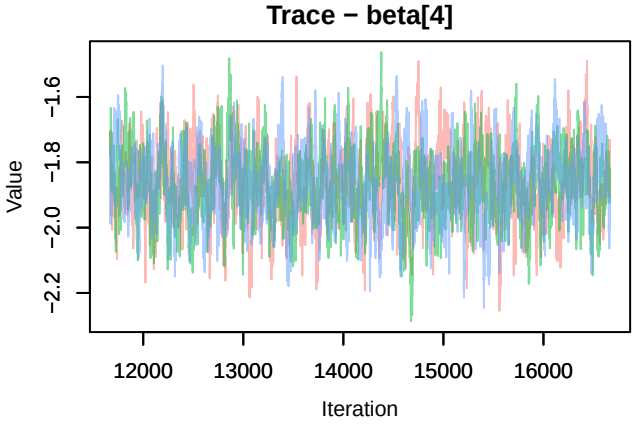

Density − beta[1]
Density
0.000
0.010
0.020
0.030
80
100
120
140
160
Parameter estimate

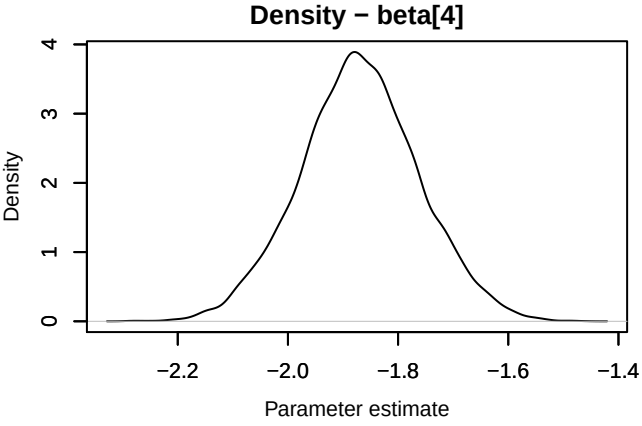

Trace − beta[2]
Value
10
15
20
25
12000
13000
14000
15000
16000
Iteration

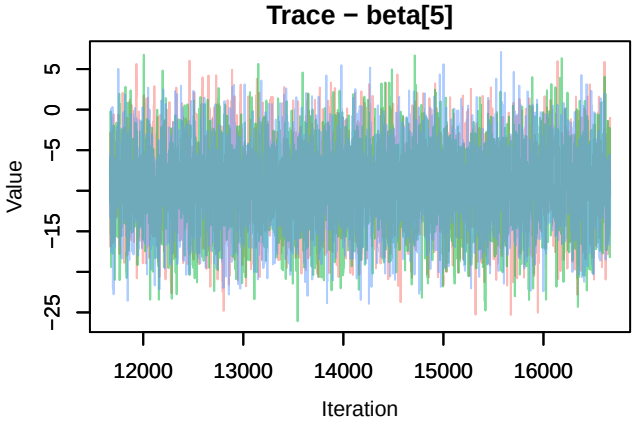

Density − beta[2]
Density
0.00
0.05
0.10
0.15
10
15
20
25
Parameter estimate

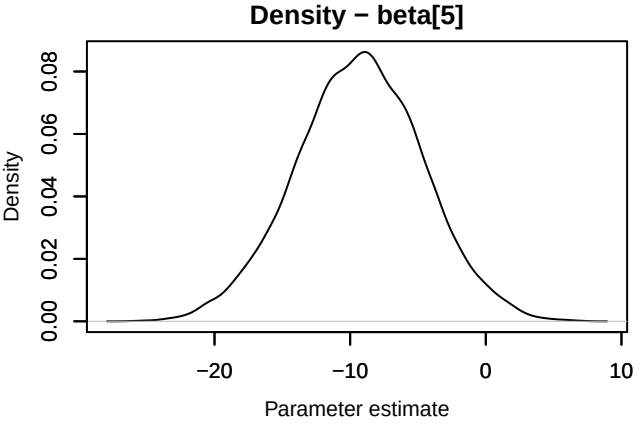

Trace − beta[3]
Value
−150
−50
0
50
12000
13000
14000
15000
16000
Iteration

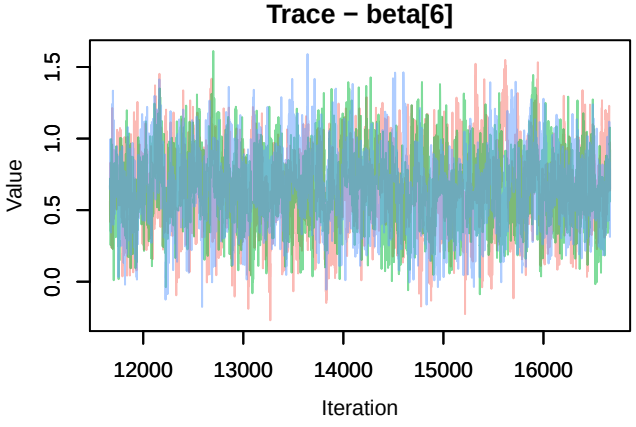

Density − beta[3]
Density
0.000
0.004
0.008
0.012
−150
−100
−50
0
50
100
Parameter estimate

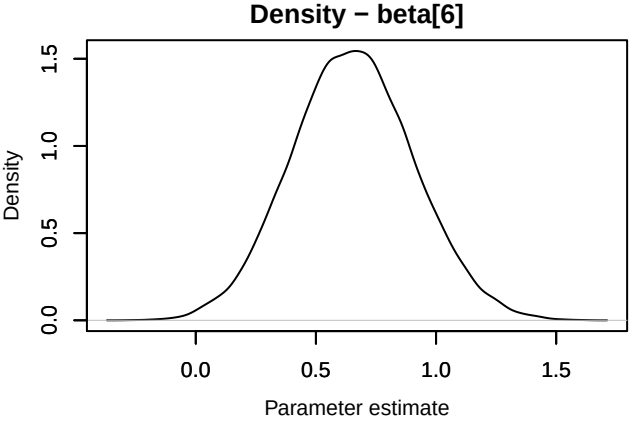

Trace − beta[4]
Value
−1.6
−1.8
−2.0
−2.2
12000
13000
14000
15000
16000
Iteration

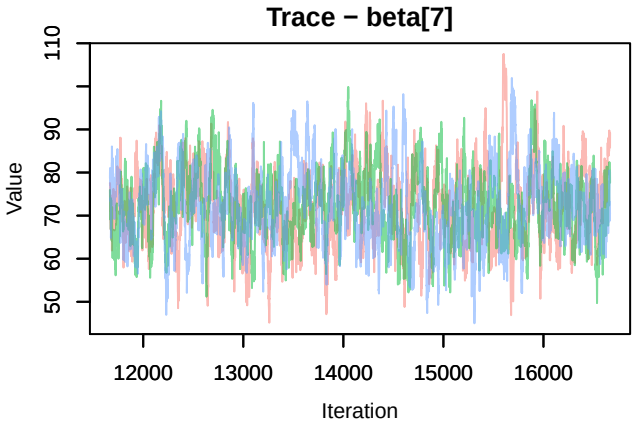

Density − beta[4]
Density
0
1
2
3
4
−2.2
−2.0
−1.8
−1.6
−1.4
Parameter estimate

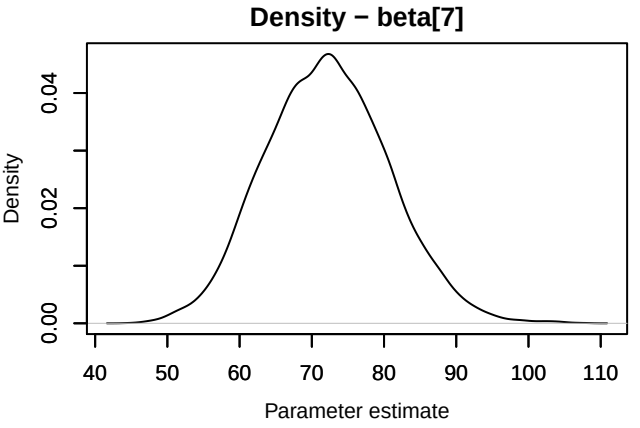

Trace − beta[5]
Value
5
0
−5
−15
−25
12000
13000
14000
15000
16000
Iteration

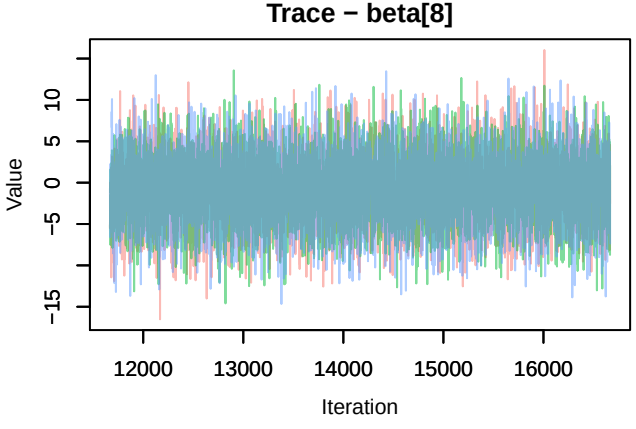

Density − beta[5]
Density
0.00
0.02
0.04
0.06
0.08
−20
−10
0
10
Parameter estimate

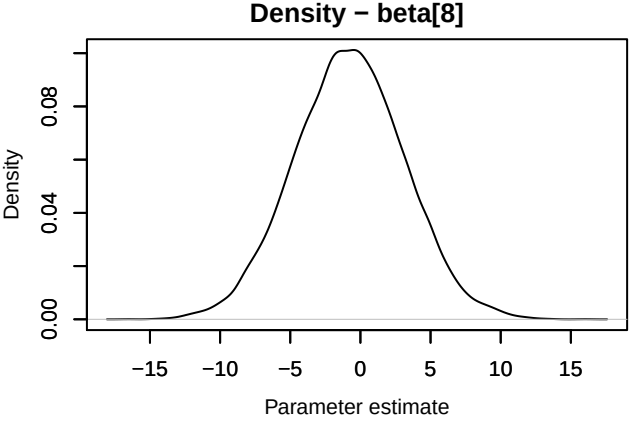

Trace − beta[6]
Value
1.5
1.0
0.5
0.0
12000
13000
14000
15000
16000
Iteration

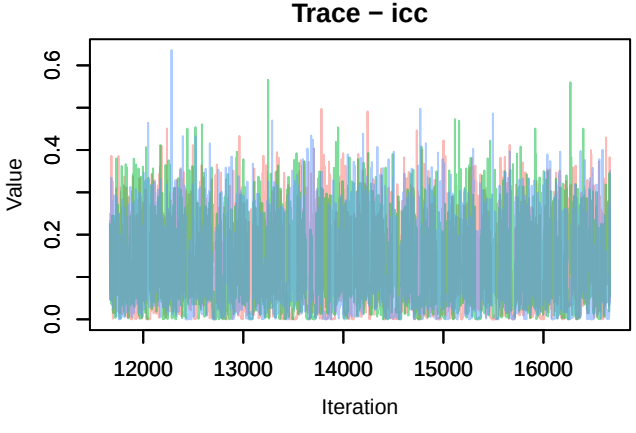

Density − beta[6]
Density
0.0
0.5
1.0
1.5
0.0
0.5
1.0
1.5
Parameter estimate

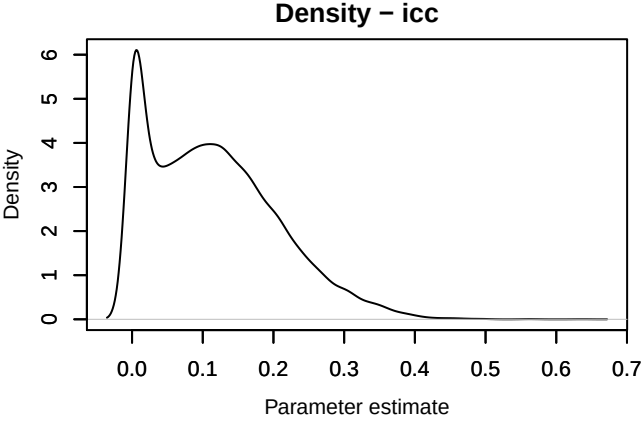

Trace − beta[7]
Value
110
90
80
70
60
50
12000
13000
14000
15000
16000
Iteration

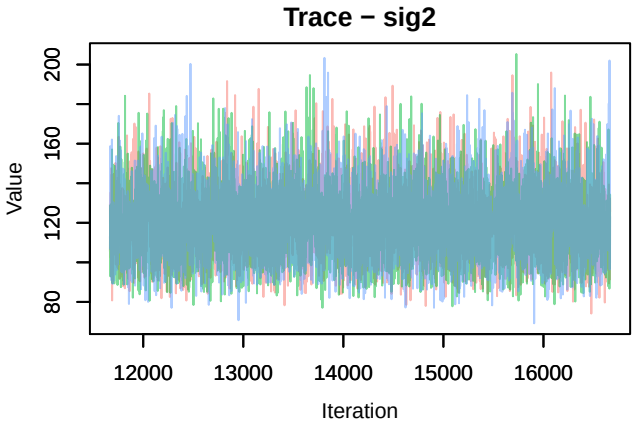

Density − beta[7]
Density
0.04
0.02
0.00
40
50
60
70
80
90
100
110
Parameter estimate

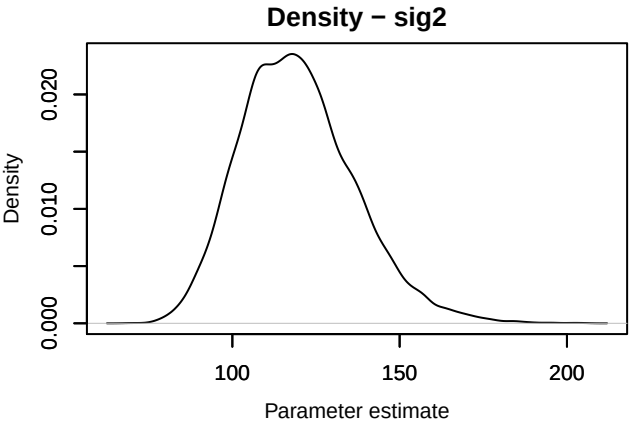

Trace − beta[8]
Value
10
5
0
−5
−15
12000
13000
14000
15000
16000
Iteration

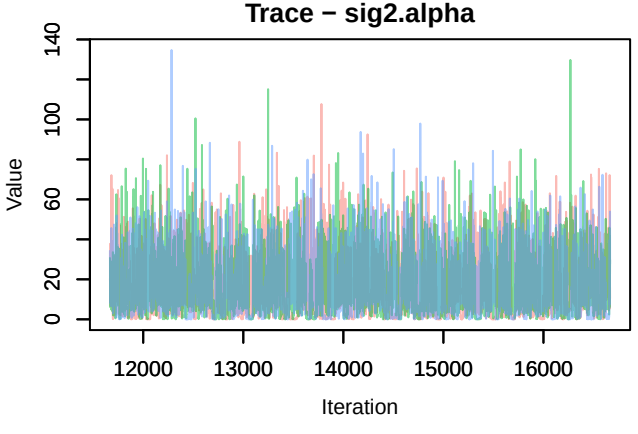

Density − beta[8]
Density
0.08
0.04
0.00
−15
−10
−5
0
5
10
15
Parameter estimate

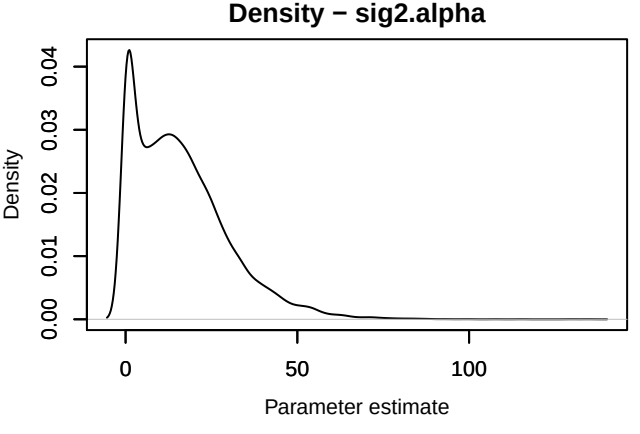

Trace − icc
Value
0.6
0.4
0.2
0.0
12000
13000
14000
15000
16000
Iteration


Density − icc
Density
6
5
4
3
2
1
0
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
Parameter estimate


Trace − sig2
Value
80
120
160
200
12000
13000
14000
15000
16000
Iteration


Density − sig2
Density
0.000
0.010
0.020
100
150
200
Parameter estimate


Trace − sig2.alpha
Value
0
20
60
100
140
12000
13000
14000
15000
16000
Iteration


Density − sig2.alpha
Density
0.00
0.01
0.02
0.03
0.04
0
50
100
Parameter estimate