Final Project Stat4DS2+DS

Pol Ribó León -1840853

29 de julio de 2019

INTRODUCTION

Every year, the NBA (National Basketball Association) hosts what it is called the NBA Draft, where each team picks a player coming out from college. Of course, teams try to spot who might be their best choice, and to do this, besides the sharp and technical eye of the scouts, there are a few tests previous to Draft Day, all applied to the upcoming players regarding different aspects such as athelticism or physical condition. A good performance in these tests can be critical for a player to be picked high in the chart and thus, play in a team that really wants him and earn a better base salary. In this sense, there is a notable difference between players selected in the so-called first draft round(so-calle lottery picks), which consists in the first 30 best players picked, and the second round, composed by the second best 30 players.

The purpose of this project is to use a regression model in order to gain insight on how the results of the tests affect a player's chances to be picked in the first or second round.

The data was taken from the site https://data.world/achou/nba-draft-combine-measurements

DATASET ANALYSIS

The dataset cointains the results of 443 players that between 2009 and 2017 took the tests before the Draft Day and were selected either in the 1st or the 2nd round.

```
data = read.csv('nba_draft_combine_all_years (1).csv', header=T, sep=';')
head(data)
```

```
##   Round Wingspan Vertical..Max. Body.Fat Sprint
## 1     1    83.25           35.5      8.2   3.28
## 2     1    81.00           37.0      5.1   3.18
## 3     1    82.25           35.0      4.4   3.14
## 4     1    83.50           34.0      8.5   3.27
## 5      1     86.50             33.0       5.2    3.35
## 6      1     86.75             28.0       5.5    3.55
```

VARIABLE EXPLANATION:

The response variable “Round” follows a Bernoulli distribution where Yi= 1 if the ith player was elected in the 1st draft round let Yi = 0 if on the other hand, it was selected in the 2nd round.

The other variables from which we will infer are:

-**Wingspan**: wingspan of the player, very popular measurement nowadays due to its importance in the player’s optimal antropometrics.

-**Vertical Jump Max**: the vertical jump is a test of an athlete’s explosive leg power. In this case, it concerns the vertical maximmum jump(without running).

-**Body fat**: The body fat percentage (BFP) of a player, which is the total mass of fat divided by total body mass, multiplied by 100.

-**Sprint**: time to sprint over the distance of three quarters of the court, measured in seconds.

First, we will check if the dataset has any missing values.

```
sum(is.na(data))
```

```
## [1] 0
```

As we see, the dataset contains no missing values.

DATASET OVERVIEW

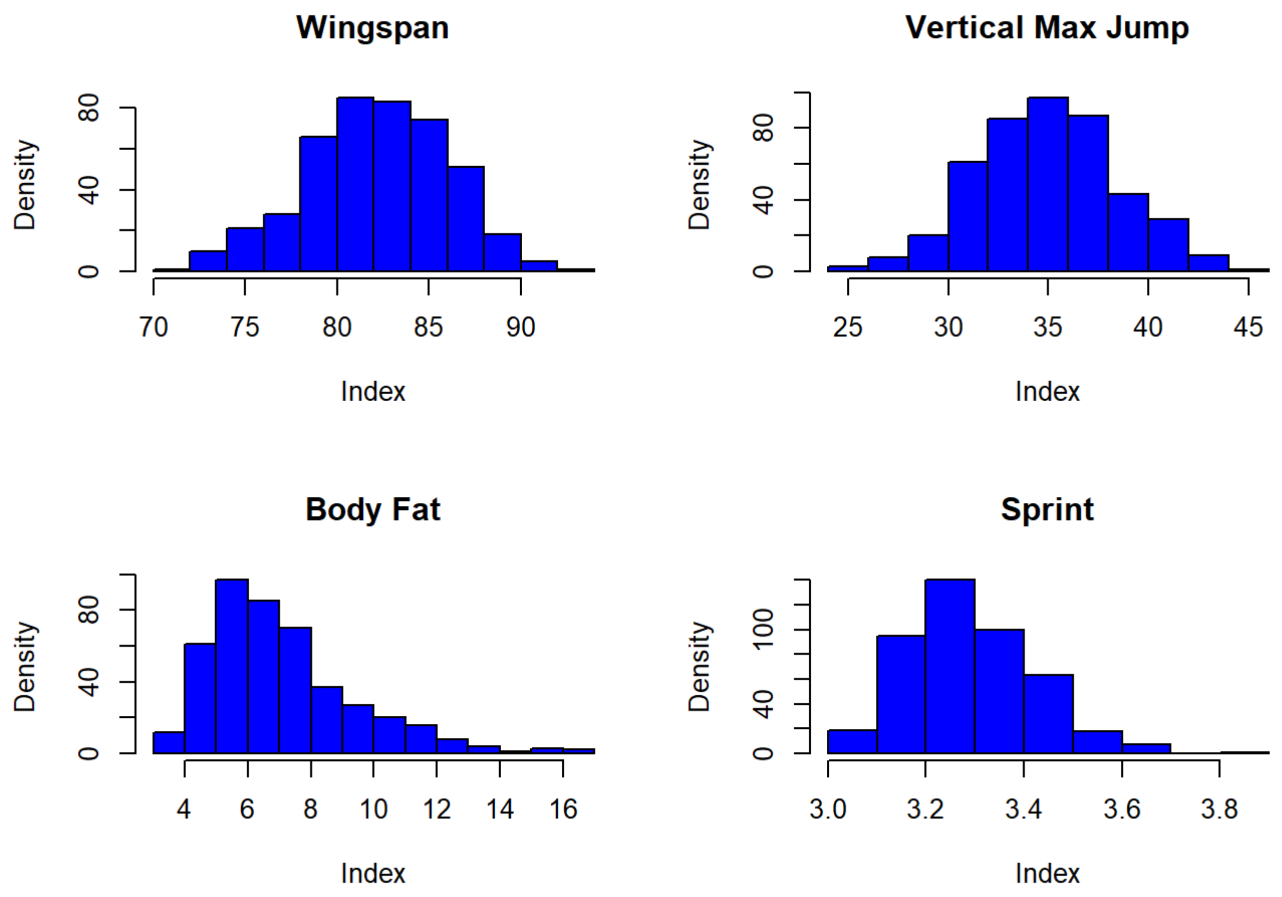
```
summary(data)
```

```
##      Round           Wingspan     Vertical..Max.     Body.Fat
##  Min.   :0.0000   Min.   :70.75   Min.   :25.00   Min.   : 3.200
##  1st Qu.:0.0000   1st Qu.:79.75   1st Qu.:32.50   1st Qu.: 5.400
##  Median :1.0000   Median :82.50   Median :35.00   Median : 6.600
##  Mean   :0.6433   Mean   :82.36   Mean   :35.17   Mean   : 7.143
##  3rd Qu.:1.0000   3rd Qu.:85.25   3rd Qu.:37.50   3rd Qu.: 8.200
##  Max.   :1.0000   Max.   :92.50   Max.   :44.50   Max.   :16.400
##      Sprint
##  Min.   :3.01
##  1st Qu.:3.20
##  Median :3.28
##  Mean   :3.30
##  3rd Qu.:3.38
##  Max.   :3.81
```

```r
#assessing variance
sapply(data, var)
```

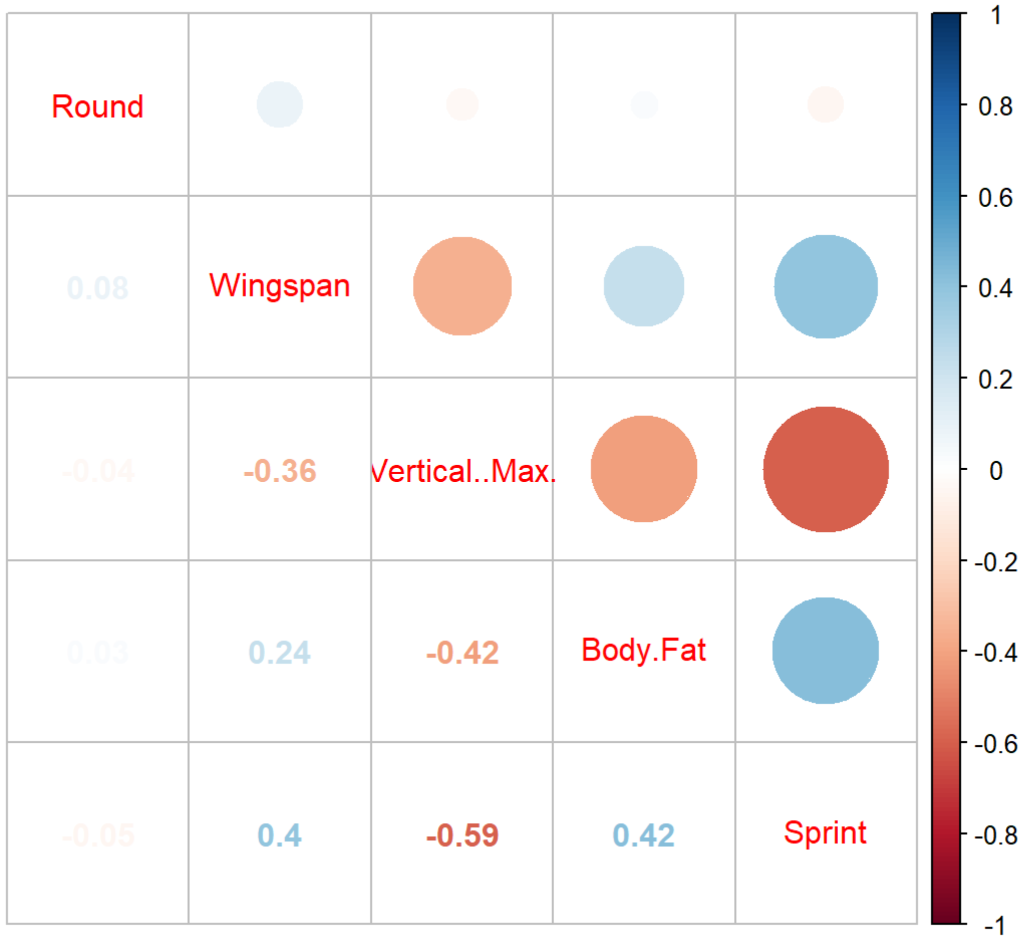
```
##          Round       Wingspan Vertical..Max.       Body.Fat         Sprint
##     0.22997252    14.77666961    12.63452856     5.52730284     0.01647327
```

```r
par(mfrow=c(2,2))
hist(main='Wingspan',data$Wingspan,xlab='Index',ylab='Density',col='blue')
hist(main='Vertical Max Jump',data$Vertical..Max.,xlab='Index',ylab='Density',col='blue')
hist(main='Body Fat',data$Body.Fat,xlab='Index',ylab='Density', col='blue')
hist(main='Sprint',data$Sprint,xlab='Index',ylab='Density',col='blue')
```



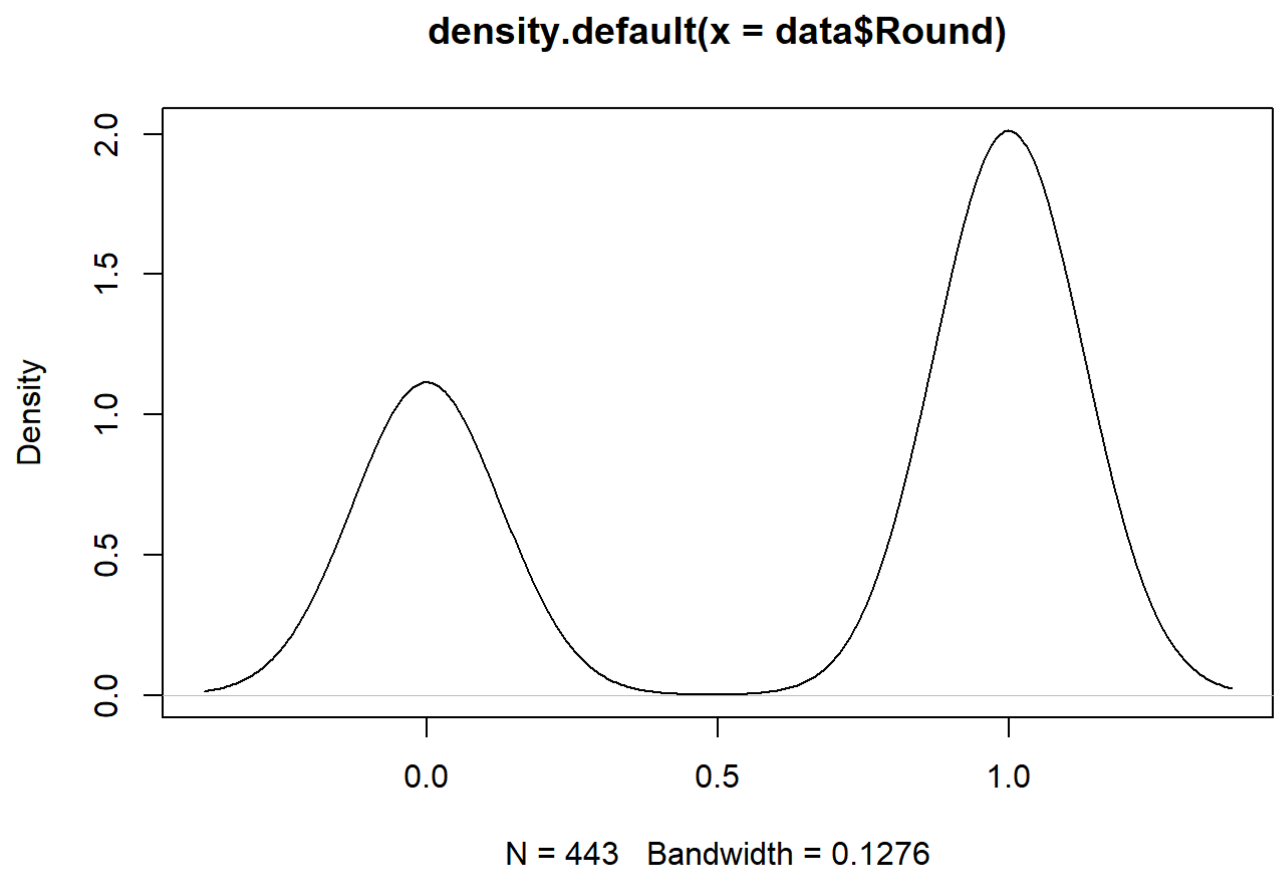
As we see, all variables seem to follow a normal distribution, although **Body Fat** and **Sprint** have fat-tails, making them positevely skewed.

```
#Assessing correlation
corr = cor(data)
corrplot.mixed(corr)
```



While the response variable is almost not correlated, there is a high correlation between the intercept variables, especially with the variable **Sprint**. This, more likely, may turn into multicolinearity. Multicolinearity occurs when independent variables in a regression model are correlated, and it can cause problems when fitting the model, interpreting the results and assessing the importance of the variables.

```
d<- density(data$Round) # returns the density data
plot(d) # plots the results
```



Round ~Bernoulli(p_i) $p_i = P(\text{Round} = 1) p_i = P(\text{Round} = 1)$

MODEL

The model we will try will be a logit model:

$$\text{logit}(p_i) = \alpha_0 + \alpha_1 * x_{1,i} + \alpha_2 * x_{2,i} + \alpha_{12} * x_{1,i} * x_{2,i} + b_i$$

In our case:

$$\alpha_i = \alpha_0 + \alpha_1 \text{wingspan}_i + \alpha_2 \text{vertical\_jump}_i + \alpha_3 \text{body\_fat}_i + \alpha_4 \text{sprint}_i$$

The model using WinBUGS language, is:

```
model_1 = "model{

  #Likelihood
    for(i in 1:n){
    y[i] ~ dbern(p[i])
    logit(p[i]) <- a0+alpha[1]*wingspan[i]+alpha[2]*vertical_max[i]+alpha[3]*body_fat[i]+alpha[4]*sprint[i]
   }

   #Priors
   a0 ~ dnorm(0.0, 1)
   for(z in 1:4){
    alpha[z] ~ dnorm(0.0, 1)
   }
  }"
```

```
#Data
datalist = list(y  = data$Round,
  wingspan = data$Wingspan,
  vertical_max=data$Vertical..Max.,
  body_fat=data$Body.Fat,
  sprint=data$Sprint,
  n=nrow(data))
```

```
inits_list = list(alpha = c(0,0,0,0))

mod1 = jags.model(textConnection(model_1),  inits = inits_list, data=datalist, n.chains=3)
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
```

```
##    Observed stochastic nodes: 443
##    Unobserved stochastic nodes: 5
##    Total graph size: 3379
##
## Initializing model
```

Here the model is run for 10000 iterations

```
params=c("a0", "alpha")
update(mod1, 1e3)




mod1_log = coda.samples(model=mod1,
                        variable.names=params,
                        n.iter = 10000)

mod1_logch = as.mcmc(do.call(rbind, mod1_log))
```

MCMC DIAGNOSTICS

The results give the posterior means, posterior standard deviations, and posterior quantiles for each variable. The ‘naive’ standard error is the standard error of the mean, which captures simulation error of the mean rather than posterior uncertainty.

```
summary(mod1_log)
```

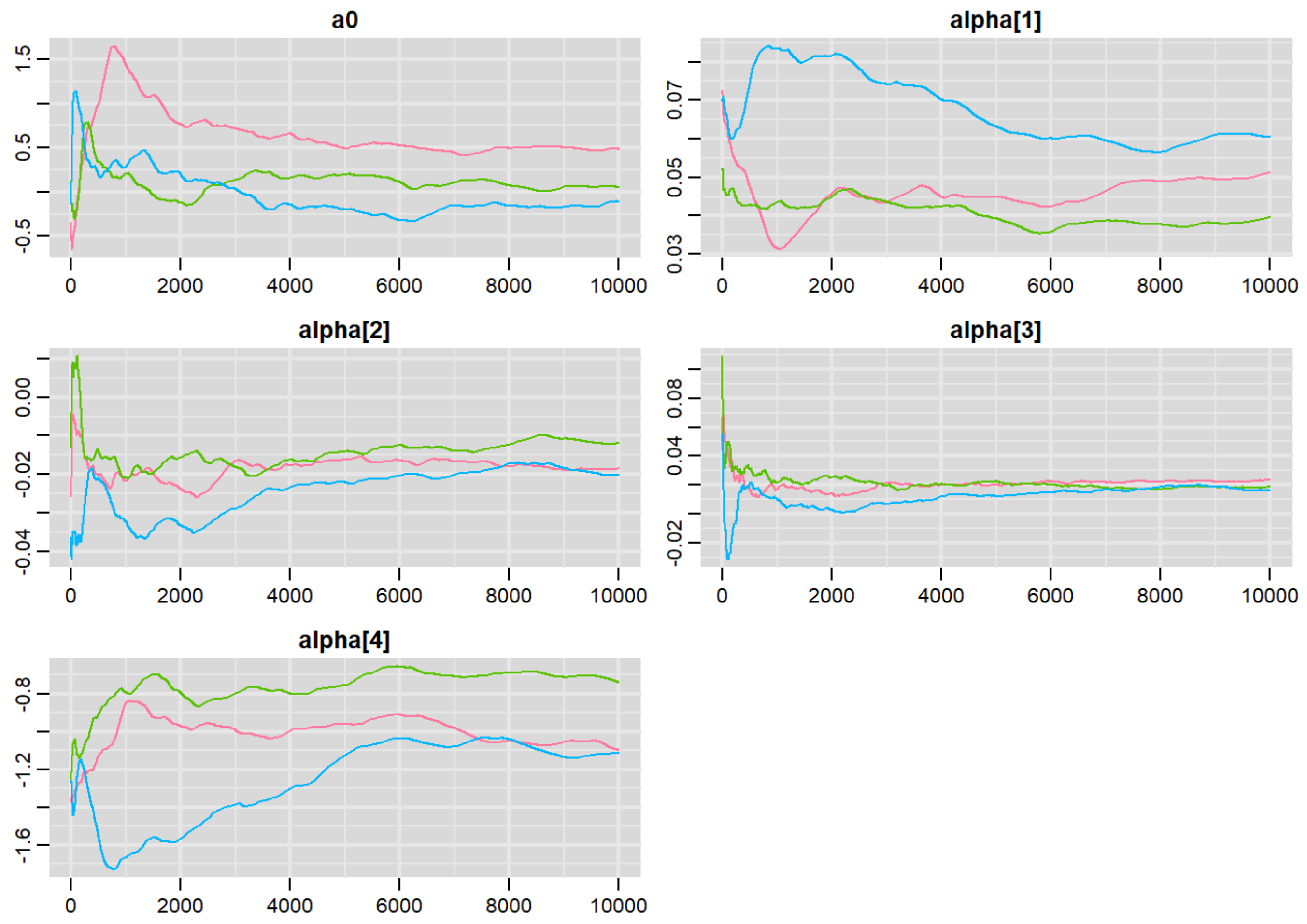
```
##
## Iterations = 2001:12000
## Thinning interval = 1
## Number of chains = 3
## Sample size per chain = 10000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
```

```
##                Mean      SD  Naive SE Time-series SE
## a0         0.14421 1.00864 0.0058234       0.089281
## alpha[1]  0.05051 0.02367 0.0001367       0.003784
## alpha[2] -0.01692 0.02423 0.0001399       0.001866
## alpha[3]  0.01940 0.04747 0.0002741       0.001557
## alpha[4] -0.98409 0.52580 0.0030357       0.075897
##
## 2. Quantiles for each variable:
##
##                 2.5%      25%      50%        75%   97.5%
## a0        -1.769376 -0.52049  0.09589  7.818e-01 2.22119
## alpha[1]  0.006889  0.03355  0.04849  6.647e-02 0.09883
## alpha[2] -0.064163 -0.03368 -0.01730 -8.678e-05 0.03078
## alpha[3] -0.073201 -0.01300  0.01934  5.205e-02 0.11131
## alpha[4] -2.059074 -1.32485 -0.94923 -6.219e-01 0.01596
```

After having run the model, a check on the reliability of the approximated posterior parameters is needed, as well as an assessment on the convergence of the algorithm. Convergence refers to the idea that eventually the Gibbs Sampler or the MCMC algorithm will eventually reach a stationary distribution. From this point on, it will stay in this distribution. As so, if the values obtained converge, then it is safe to say that the distribution from where the samples are drawn is the correct one. TO do that, different techniques will be used.

MEAN PLOTS

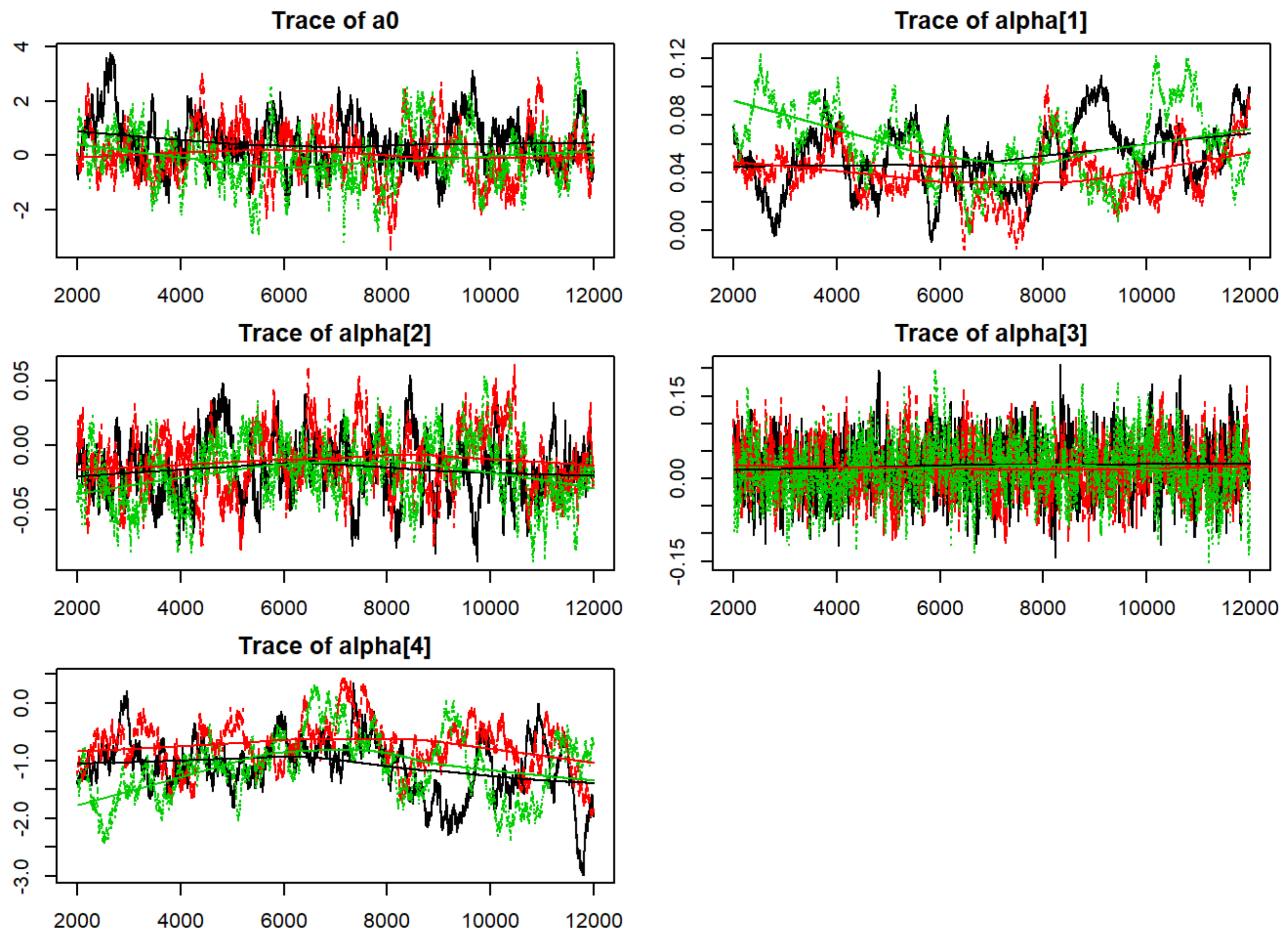
```
rmeanplot(mod1_log)
```



As it is seen, the running mean plots show that except for the ‘Body Fat’ variable, the chains don’t converge, so a good mixing is not accomplished as the means are not independent. Also, in case of stationarity it should stabilize with in while increasing k.

TRACE PLOTS

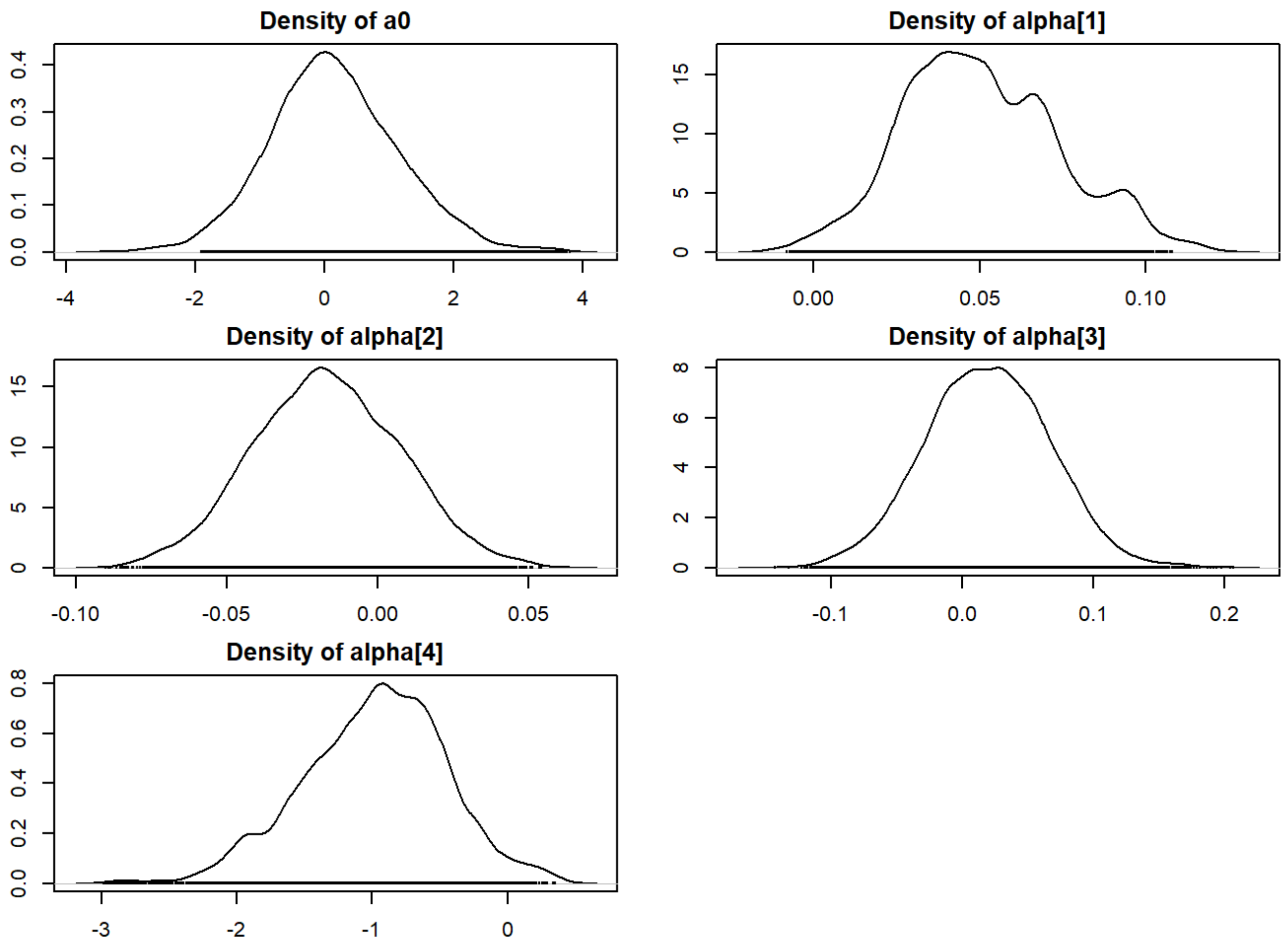
```
par(mar = rep(2, 4))
plot(mod1_log, density=FALSE,trace=TRUE)
```



As we can easily see, only the variable ‘Body Fat’ shows a solid random behavior, as all the other variables have trends in their sample spaces. So,the model doesn’t converge.

DENSITY PLOTS

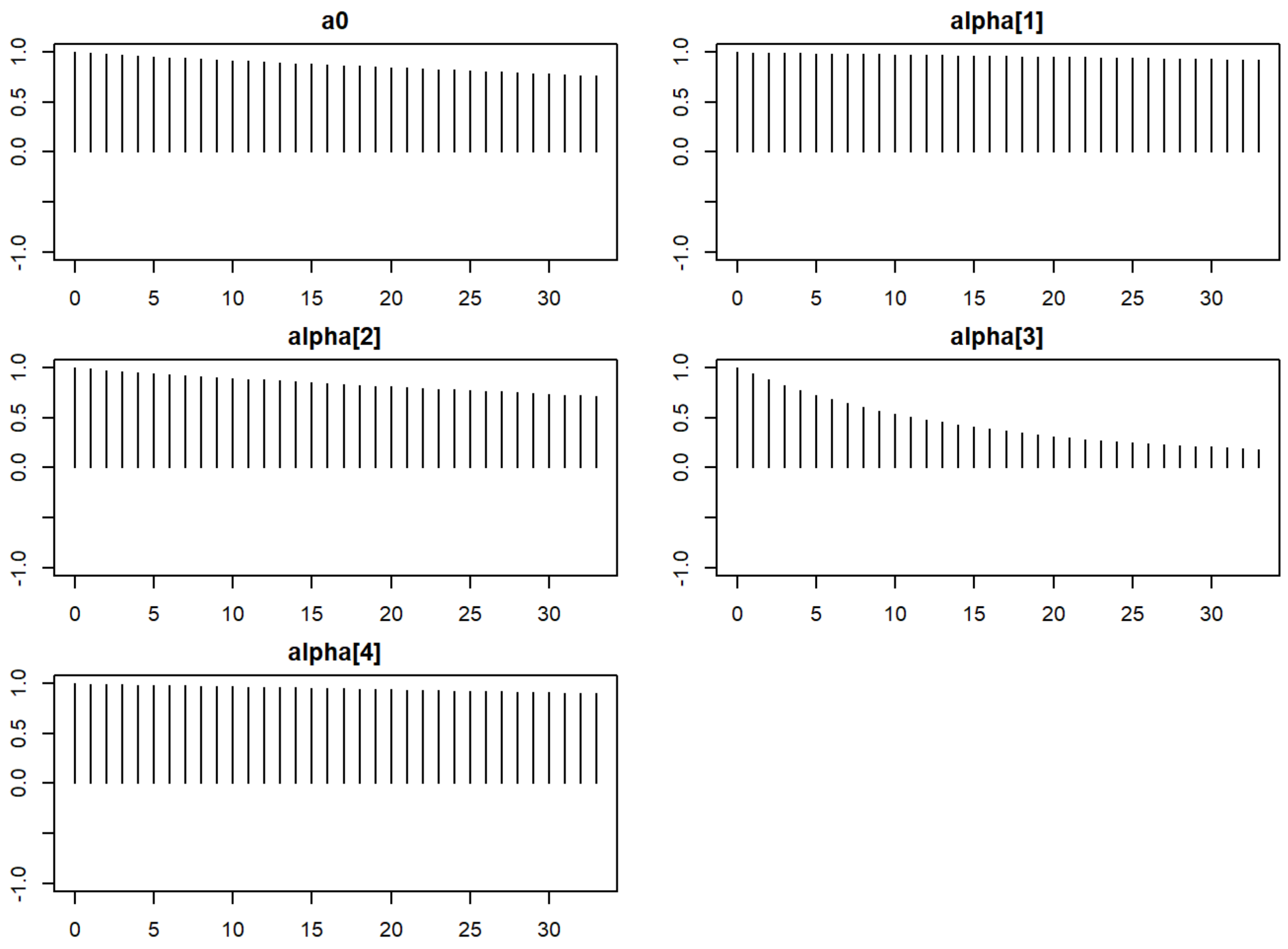
```
par(mar = rep(2, 4))
plot(mod1_log, density=TRUE,trace=FALSE)
```



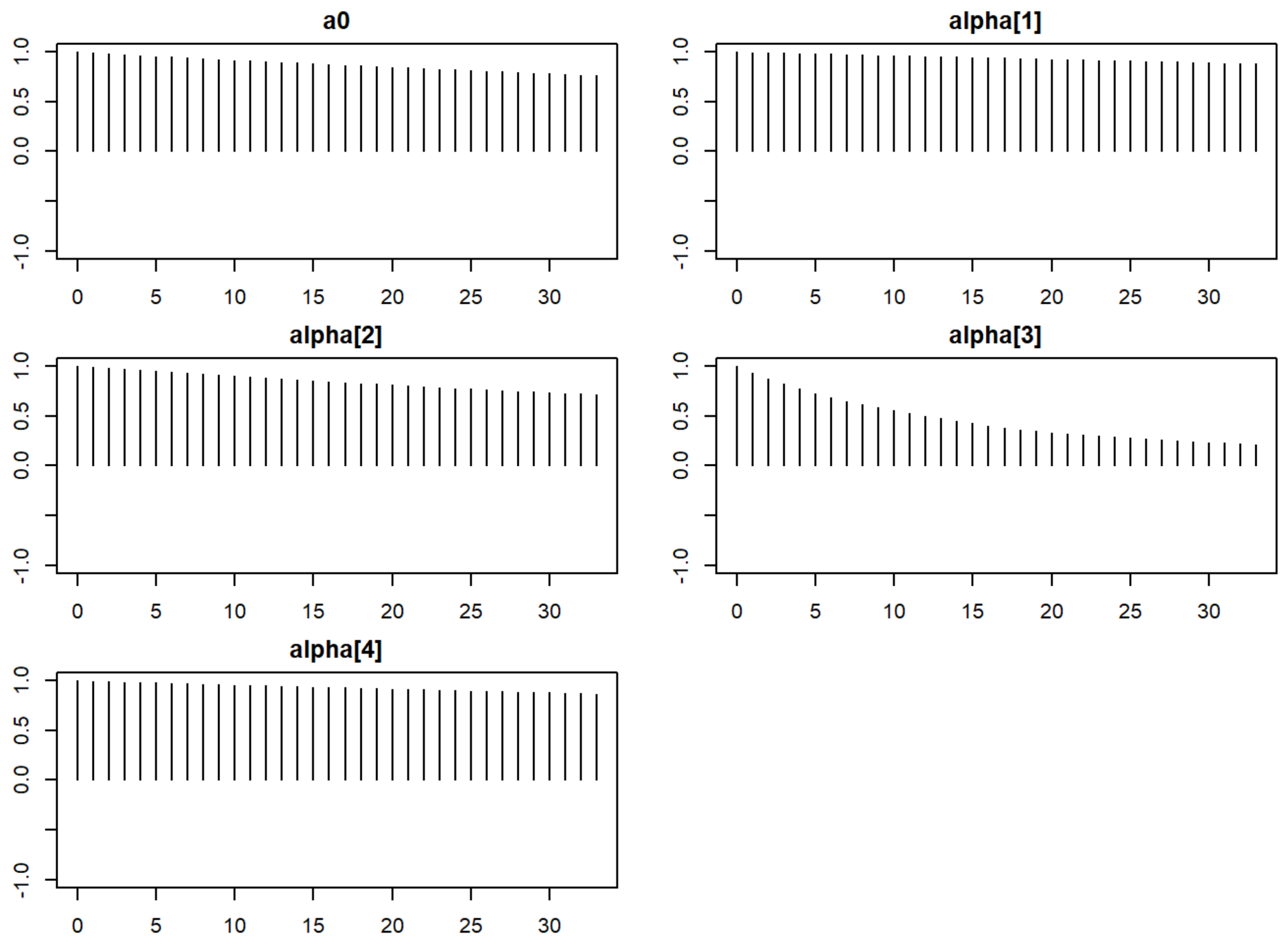
As they look bell-shaped, we can consider as satisfactory and a suitable performance the density plots.

AUTOCORRELATION PLOTS

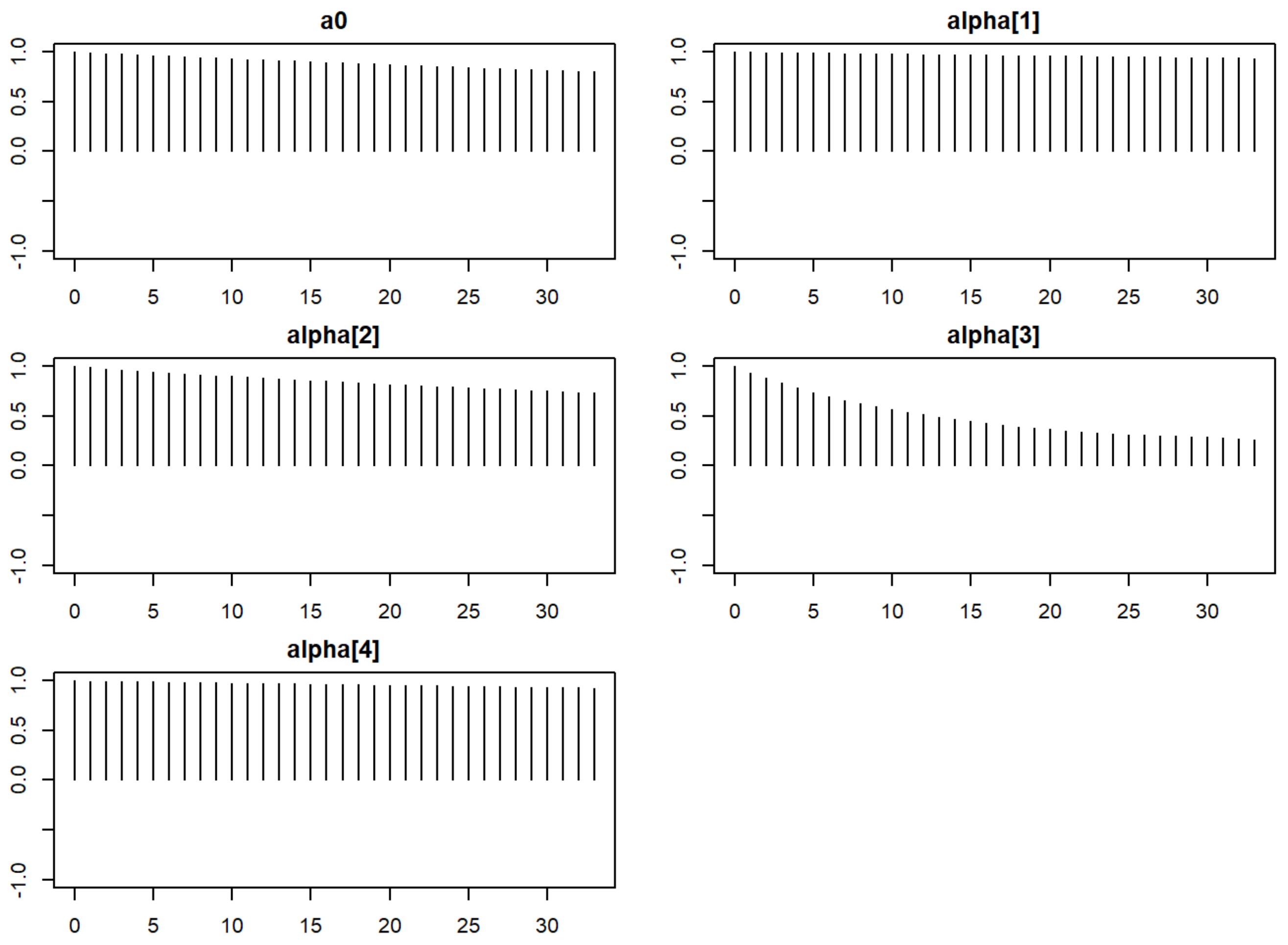
```r
par(mar = rep(2, 4))
autocorr.plot(mod1_log[1])
```



```
autocorr.plot(mod1_log[2])
```



```
autocorr.plot(mod1_log[3])
```



```
autocorr.diag(mod1_log[1])
```

```
##                 a0  alpha[1]  alpha[2]  alpha[3]  alpha[4]
## Lag 0  1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## Lag 1  0.9910676 0.9975796 0.9889745 0.9396436 0.9970116
## Lag 5  0.9566853 0.9879831 0.9458862 0.7295893 0.9850098
## Lag 10 0.9171307 0.9760948 0.8968273 0.5390425 0.9701648
## Lag 50 0.6639762 0.8825432 0.5816413 0.1129705 0.8518443
```

```
autocorr.diag(mod1_log[2])
```

```
##                a0  alpha[1]  alpha[2]  alpha[3]  alpha[4]
## Lag 0  1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## Lag 1  0.9915046 0.9962796 0.9894609 0.9342064 0.9956535
## Lag 5  0.9579163 0.9818366 0.9499687 0.7299314 0.9784901
## Lag 10 0.9183892 0.9643868 0.9022033 0.5566951 0.9571111
## Lag 50 0.6617540 0.8211439 0.6050984 0.1444024 0.8035146
```

```
autocorr.diag(mod1_log[3])
```

```
##                a0  alpha[1]  alpha[2]  alpha[3]  alpha[4]
## Lag 0  1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## Lag 1  0.9930316 0.9981937 0.9887218 0.9380010 0.9976034
## Lag 5  0.9651583 0.9909428 0.9464408 0.7384328 0.9885849
## Lag 10 0.9330694 0.9816791 0.8992104 0.5646253 0.9776079
## Lag 50 0.7159978 0.9061382 0.6402289 0.1965963 0.8967383
```

As we apprecciate, autocorrelation of the parameters is not decreasing as it should to consider that our algorithm converges. As well, the level of autocorrelation of all of our parameters except Body_Fat is very high, which makes it difficult to assess convergence. The cause of autocorrelation is that the parameters in our model may be highly correlated, so the Gibbs Sampler will be slow to explore the entire posterior distribution.

Tests

In this section, Gelman-Rubin test and Heidelberger-Welch test will be performed.

Gelman-Rubin

```
gelman.diag(mod1_log)
```

```
## Potential scale reduction factors:
##
##          Point est. Upper C.I.
## a0               1.05       1.15
## alpha[1]         1.13       1.41
## alpha[2]         1.04       1.13
## alpha[3]         1.01       1.02
## alpha[4]         1.14       1.42
##
## Multivariate psrf
##
## 1.13
```

The potential scale reduction factor for all the variables except Body_Fat is high (should be closer to 1), showing that our chains doesn't converge and we should run them out longer to improve convergence to the stationary distribution.

Heidelberger-Welch

```
heidel.diag(mod1_log)
```

```
## [[1]]
##
##          Stationarity start     p-value
##          test         iteration
## a0       passed       1         0.338
## alpha[1] passed       1         0.337
## alpha[2] passed       1         0.791
## alpha[3] passed       1         0.390
## alpha[4] passed       1         0.615
##
##          Halfwidth Mean    Halfwidth
##          test
## a0       failed     0.4851 0.27013
## alpha[1] failed     0.0513 0.01269
## alpha[2] failed    -0.0185 0.00626
## alpha[3] failed     0.0232 0.00534
## alpha[4] failed    -1.0984 0.25555
##
## [[2]]
```

```
##
##            Stationarity start     p-value
##            test         iteration
## a0         passed       1         0.897
## alpha[1] passed       1         0.899
## alpha[2] passed       1         0.440
## alpha[3] passed       1         0.589
## alpha[4] passed       1         0.883
##
##            Halfwidth Mean    Halfwidth
##            test
## a0         failed     0.0548 0.29021
## alpha[1] failed     0.0398 0.00831
## alpha[2] failed    -0.0120 0.00655
## alpha[3] failed     0.0188 0.00505
## alpha[4] failed    -0.7410 0.17600
##
## [[3]]
##
##            Stationarity start     p-value
##            test         iteration
## a0         passed       1         0.7408
## alpha[1] passed       1         0.5918
## alpha[2] passed       1         0.3081
## alpha[3] passed       1         0.0767
## alpha[4] passed       1         0.5269
##
##            Halfwidth Mean    Halfwidth
##            test
## a0         failed    -0.1073 0.34410
## alpha[1] failed     0.0605 0.01628
## alpha[2] failed    -0.0202 0.00619
## alpha[3] failed     0.0161 0.00545
## alpha[4] failed    -1.1129 0.32075
```

As the halfwidth test has been failed for all chains, we reject the null hypothesis that states that the sampled values come from a stationary distribution. In this case, the chain must be run out longer.

Effective Sample Size

```
effectiveSize(mod1_log)
```

```
##        a0  alpha[1]  alpha[2]  alpha[3]  alpha[4]
## 122.47179  39.78896 165.10200 926.92551  48.34437
```

As the definition of the effective sample size states;

$$\text{ESS} = \frac{n}{1 + 2\sum_{k=1}^{\infty} \rho(k)}$$

$$\text{ESS} = \frac{n}{1 + 2\sum_{k=1}^{\infty} \rho(k)}$$

as in all variables except Body_fat autocorrelation between lags(k) decrease so slowly, the ESS is low, which is an indicator that the MCMC model hasn't converged.

IMPROVING CONVERGENCE

There are several ways to try to improve convergence. Here we applied some:

-Standardize the variables so they have mean=0 and var=1. It has been done with the function 'scale'. It involves the creation of a new dataframe called 'data1'.

-Run for more iterations: in this case, and according to the test results of Heidelberger-Welch, the number of iterations will be increased till 100.000.

-Change inits: different value for initial values of the priors will be applied

-Thinning: a thinning of 100k will be applied.

-Apply Burnin: sometimes a low effective size number of samples is just because the chain started in a low-probability region, and found the basin of convergence (the high probability region, or typical set) only later on. To solve, a burn-in of 50.000 will be applied.

NOTE: although eliminating the intercept was considered because the variable 'sprint' was acting like a substitute of it, after trying both models with an without the intercept, the former performed better. On the other hand, as the variables have high autocorreñation, eliminating 'sprint' was also considered.

```
#standardization
wing=scale(data$Wingspan)
vert=scale(data$Vertical..Max.)
body=scale(data$Body.Fat)
sprint=scale(data$Sprint)
```

```
#creation of new dataframe
data1 <- data.frame(data$Round, wing, vert, body,sprint)
head(data1)
```

```
##   data.Round        wing        vert       body     sprint
## 1          1  0.2308693  0.09240165  0.4494468 -0.1544190
## 2          1 -0.3544518  0.51440096 -0.8691289 -0.9335493
## 3          1 -0.0292734 -0.04826478 -1.1668718 -1.2452014
## 4          1  0.2959050 -0.32959765  0.5770509 -0.2323321
## 5          1  1.0763333 -0.61093052 -0.8265942  0.3909721
## 6          1  1.1413690 -2.01759488 -0.6989901  1.9492327
```

```
#assessing mean and variance
mean(data1$wing)
```

```
## [1] -5.75162e-16
```

```
var(data1$wing)
```

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

Running the new model

```
#Data
datalist_new = list(y  = data1$data.Round,
  wingspan = data1$wing,
  vertical_max=data1$vert,
  body_fat=data1$body,
  sprint=data1$sprint,
  n=nrow(data1))
```

```
model_1_new1 = "model{

  #Likelihood
    for(i in 1:n){
    y[i] ~ dbern(p[i])
    logit(p[i]) = a0+alpha[1]*wingspan[i]+alpha[2]*vertical_max[i]+alpha[3]*body_fat[i]+alpha[4]*sprint[i]
   }

   #Priors
   a0 ~ dnorm(0.0, 1.0E-06)
   for(z in 1:4){
    alpha[z] ~ dnorm(0.0, 1.0E-06)
      }
}
"
```

```
inits_list_new = list(alpha = c(0,0,0,0))

mod1_imp = jags.model(textConnection(model_1_new1),  inits = inits_list_new, data=datalist_new, n.chains=3)
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 443
##    Unobserved stochastic nodes: 5
##    Total graph size: 3379
##
## Initializing model
```

```
params=c("a0","alpha")
update(mod1, 1e3)



mod1_log_new = coda.samples(model=mod1_imp,
                        variable.names=params,
                        n.iter = 100000, n.burnin = 50000, thin = 100)

mod1_logch_new = as.mcmc(do.call(rbind, mod1_log_new))
```

MCMC DIAGNOSTICS

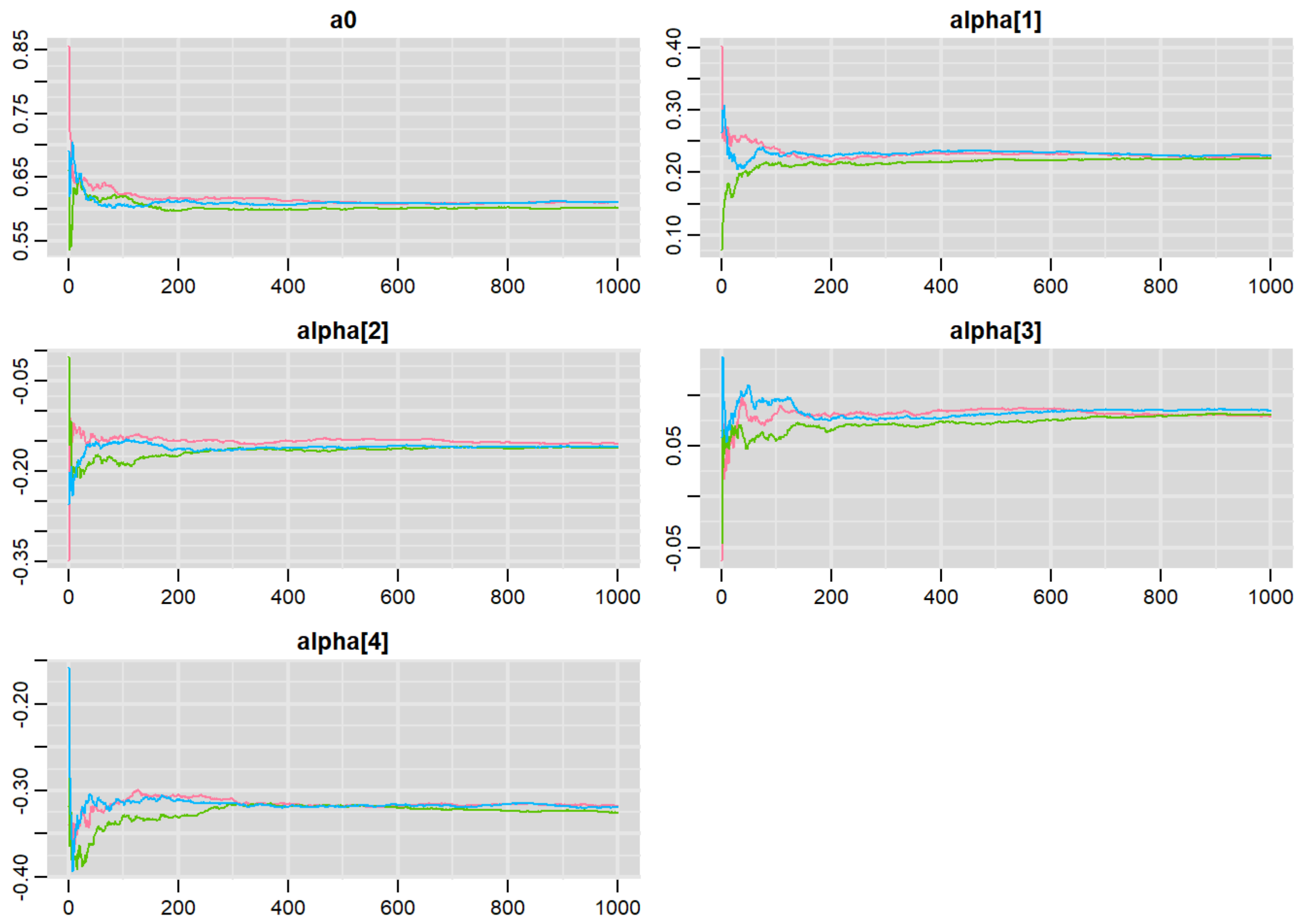
```
summary(mod1_log_new)
```

```
##
## Iterations = 1100:101000
## Thinning interval = 100
## Number of chains = 3
## 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
## a0         0.60768 0.1008 0.001841       0.001863
## alpha[1]   0.22565 0.1145 0.002090       0.002086
## alpha[2] -0.15846 0.1334 0.002436       0.002401
## alpha[3]   0.08129 0.1176 0.002147       0.002140
## alpha[4] -0.32087 0.1342 0.002450       0.002450
```

```
##
## 2. Quantiles for each variable:
##
##                2.5%       25%      50%     75%    97.5%
## a0         0.406455  0.540422  0.60897  0.6723  0.80544
## alpha[1]  0.009683  0.148325  0.22527  0.3028  0.45363
## alpha[2] -0.415378 -0.248646 -0.15991 -0.0666  0.10868
## alpha[3] -0.148140  0.002498  0.07721  0.1594  0.31719
## alpha[4] -0.579709 -0.412291 -0.31795 -0.2270 -0.05785
```

MEAN PLOTS

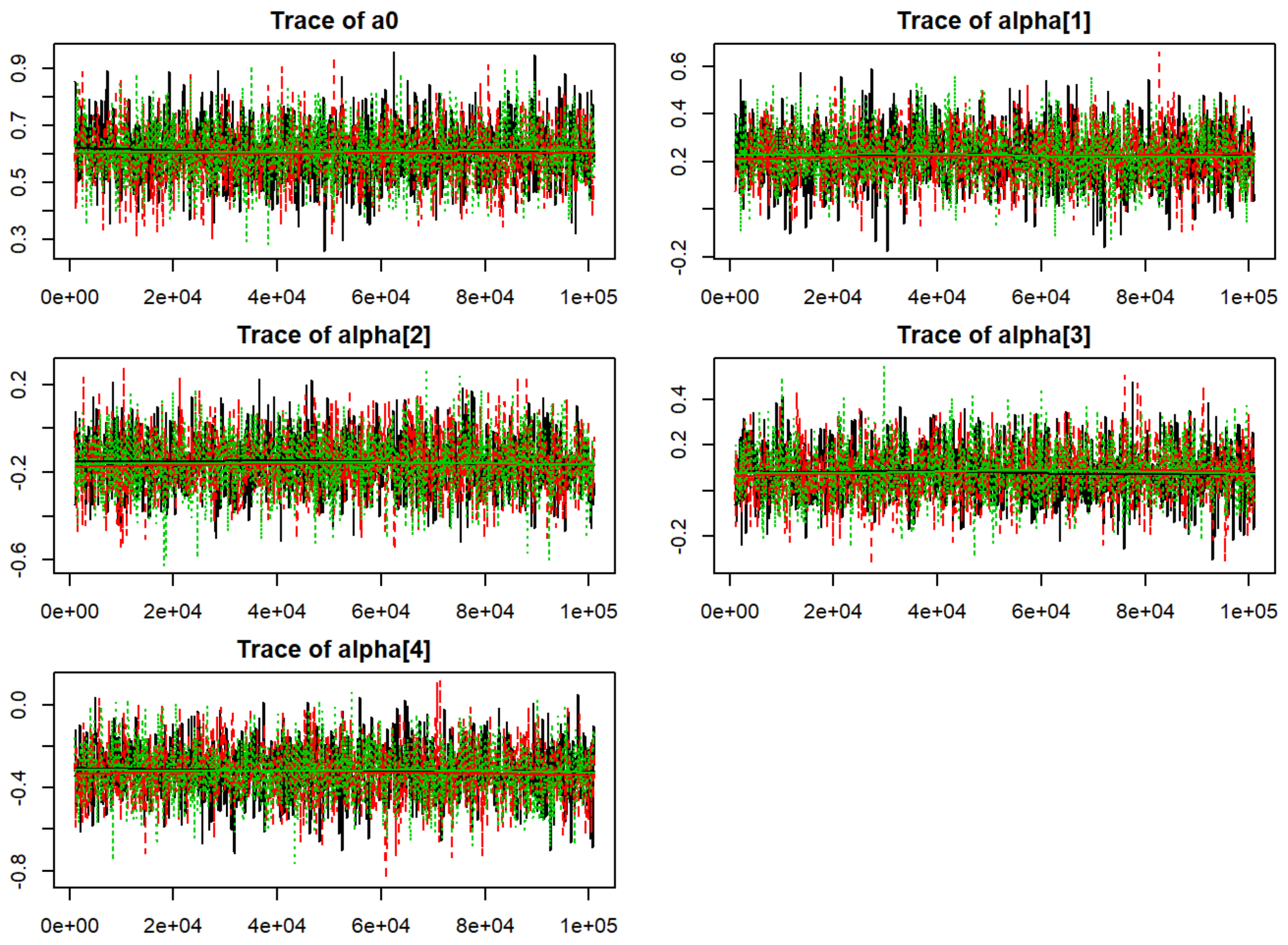
```
rmeanplot(mod1_log_new)
```



Further samples from a parameter’s posterior distribution don’t influence the calculation of the mean, as they converge.

TRACE PLOTS

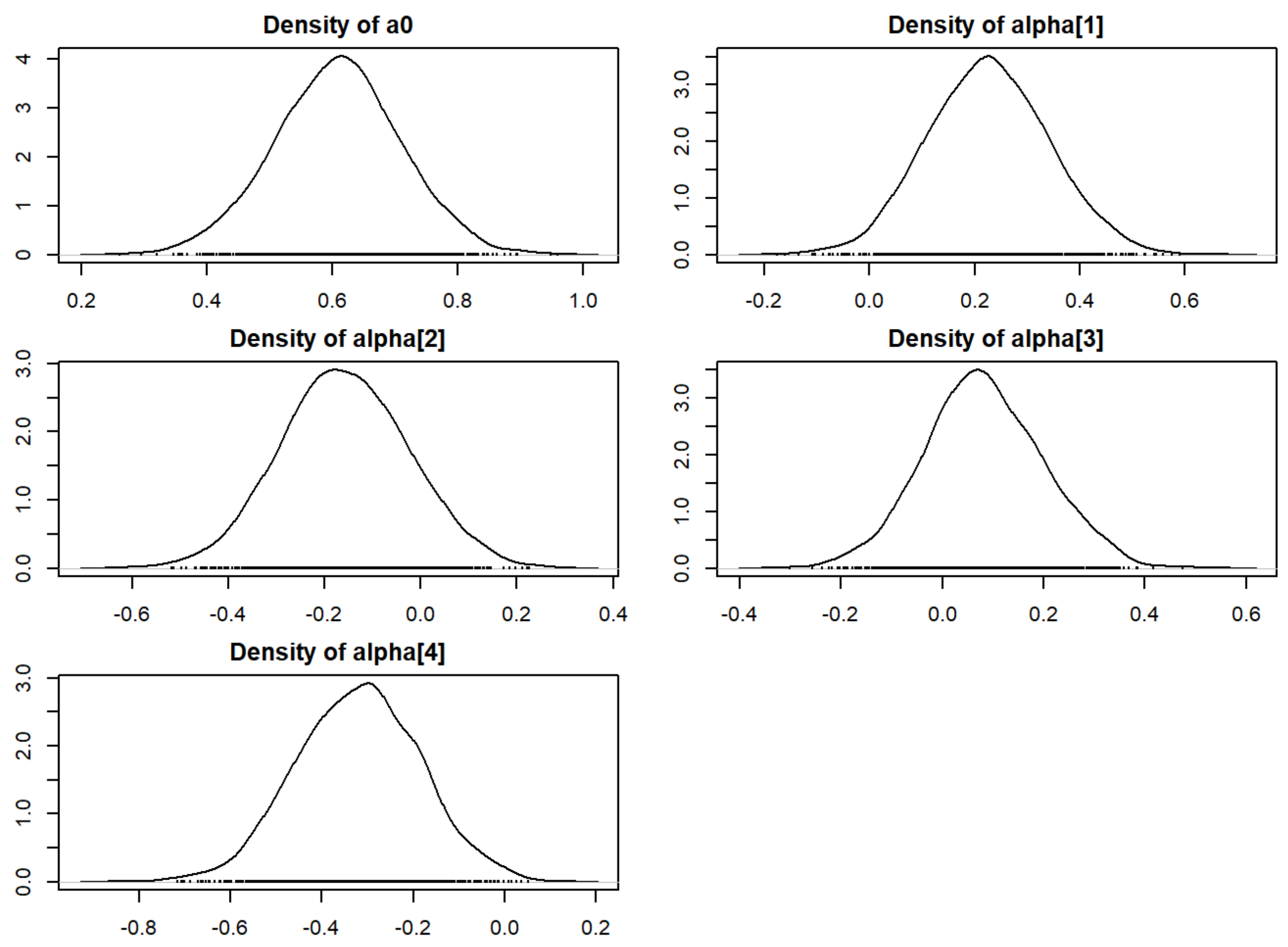
```
par(mar = rep(2, 4))
plot(mod1_log_new, density=FALSE,trace=TRUE)
```



The traceplots move around the mode of the distribution, expressing random behavior(no trend).

DENSITY PLOTS

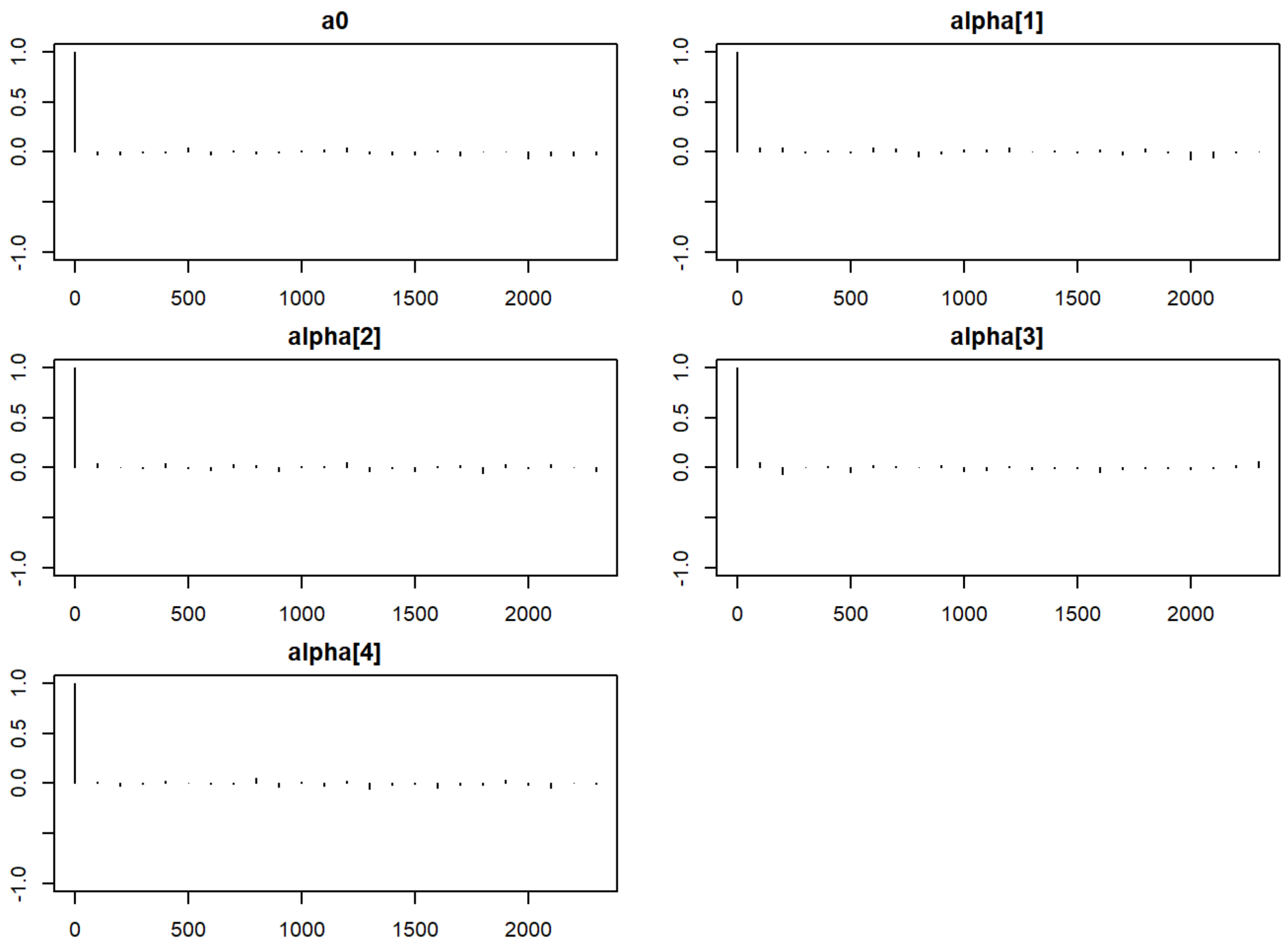
```
par(mar = rep(2, 4))
plot(mod1_log_new, density=TRUE,trace=FALSE)
```



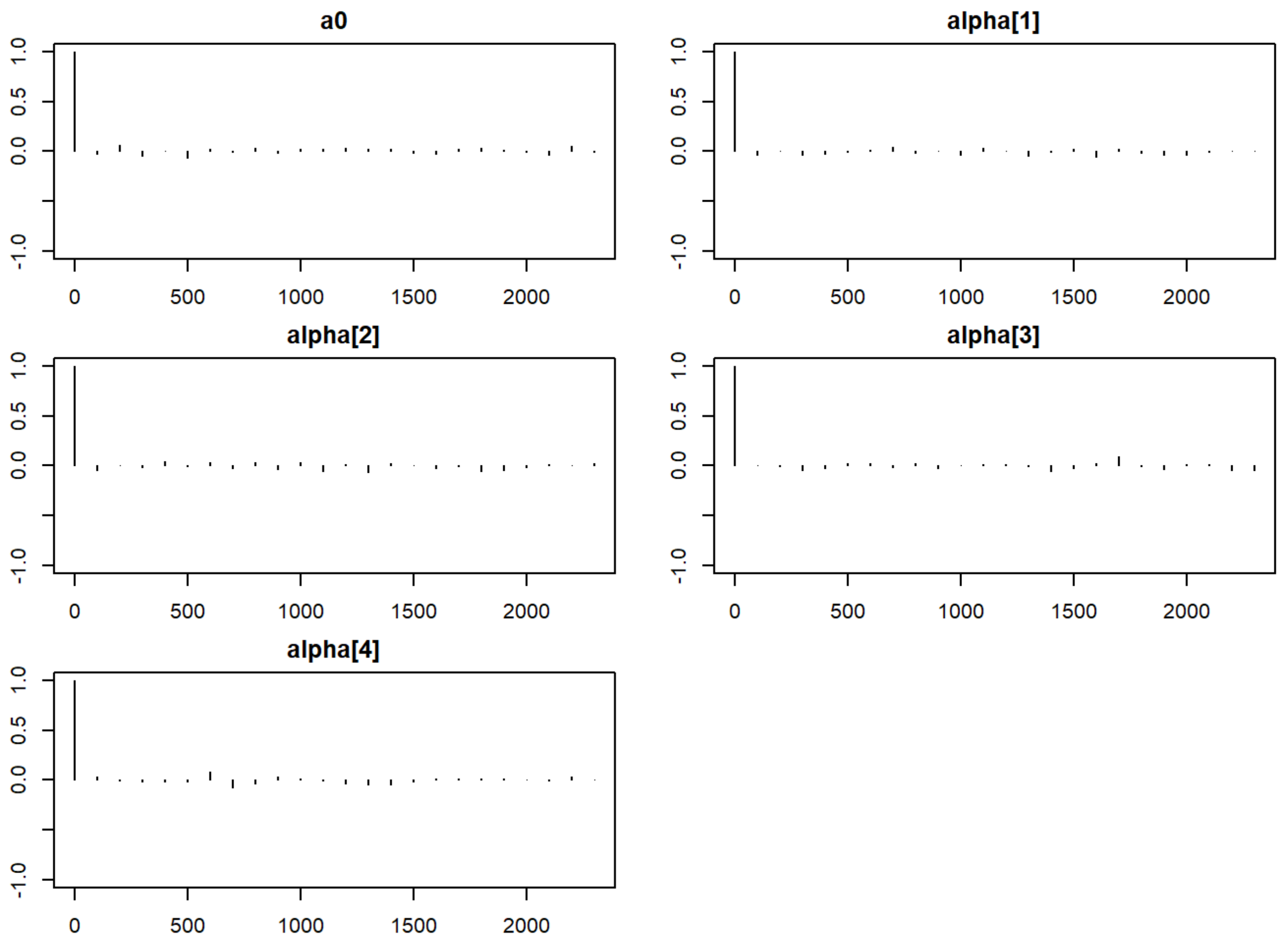
More bell-shaped curves than in the previous model are shown.

AUTOCORRELATION PLOTS

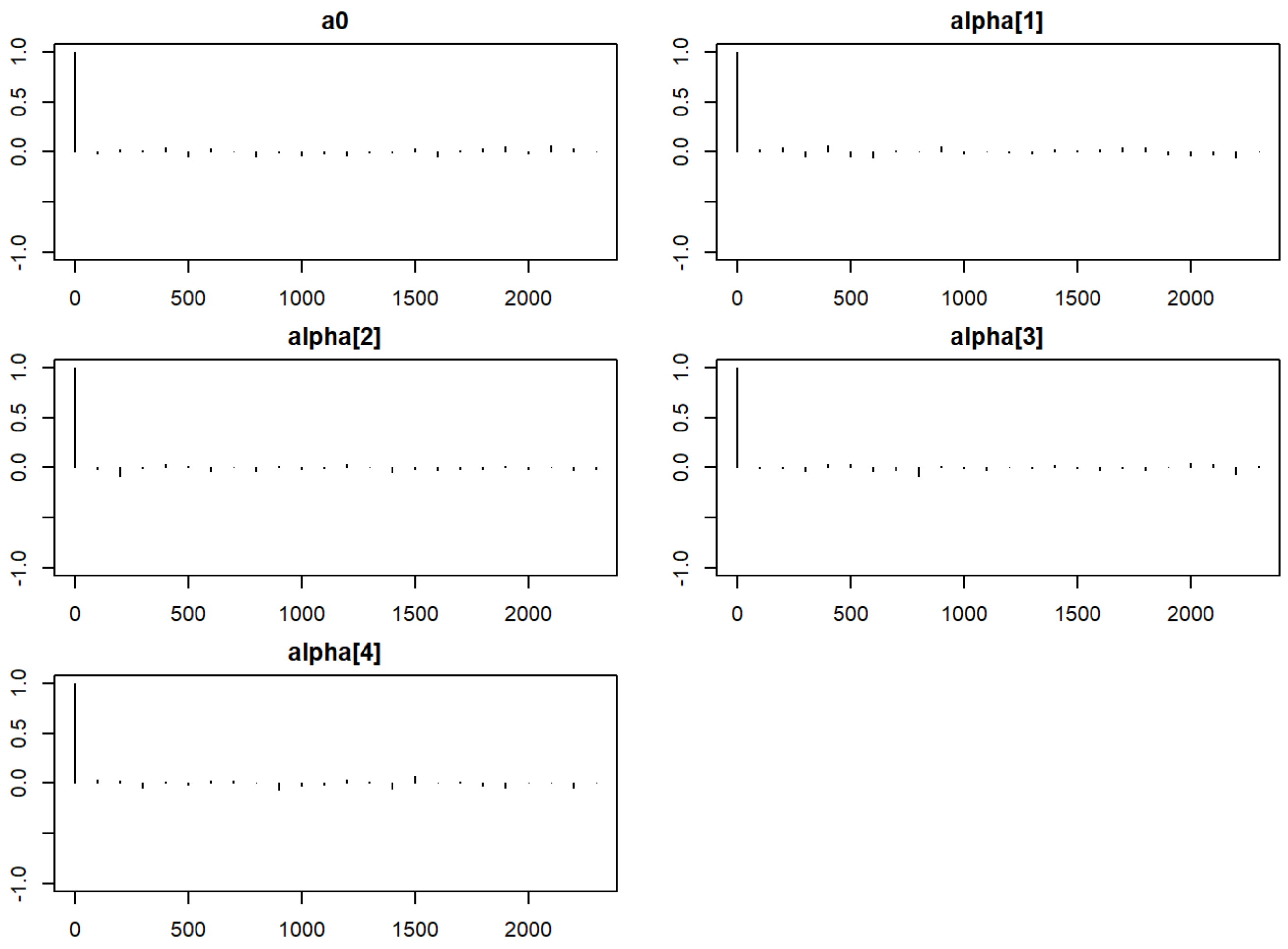
```
par(mar = rep(2, 4))
autocorr.plot(mod1_log_new[1])
```



```
autocorr.plot(mod1_log_new[2])
```



```
autocorr.plot(mod1_log_new[3])
```



```r
autocorr.diag(mod1_log_new[1])
```

```
##                    a0     alpha[1]     alpha[2]    alpha[3]    alpha[4]
## Lag 0     1.00000000  1.000000000  1.000000000  1.00000000 1.000000000
## Lag 100  -0.02097598  0.044126858  0.045295882  0.05408613 0.016772852
## Lag 500   0.04049597 -0.009628047 -0.006683553 -0.04756058 0.005780742
## Lag 1000  0.01676439  0.021241384  0.015200141 -0.03226551 0.016026795
## Lag 5000  0.05433155 -0.014561899  0.042973519 -0.01410750 0.058725216
```

```
autocorr.diag(mod1_log_new[2])
```

```
##                   a0     alpha[1]     alpha[2]     alpha[3]     alpha[4]
## Lag 0     1.00000000  1.000000000  1.000000000  1.000000000  1.000000000
## Lag 100  -0.02555241 -0.033175737 -0.041476281  0.003645635  0.030548580
## Lag 500  -0.06094148 -0.004499377 -0.007185097  0.028442953 -0.014279371
## Lag 1000  0.02414894 -0.031751650  0.029989647  0.001686569  0.010255715
## Lag 5000 -0.02163643 -0.043283217 -0.011779205 -0.042387631  0.004469293
```

```
autocorr.diag(mod1_log_new[3])
```

```
##                   a0    alpha[1]     alpha[2]     alpha[3]    alpha[4]
## Lag 0     1.00000000  1.00000000  1.000000000  1.000000000  1.00000000
## Lag 100  -0.01747868  0.02651220 -0.014869919 -0.006355079  0.03276622
## Lag 500  -0.04092926 -0.03972364  0.014371365  0.034787281 -0.01915068
## Lag 1000 -0.03691274 -0.01173719 -0.019526067 -0.008986719 -0.02911259
## Lag 5000  0.02463742 -0.02316284  0.007036707 -0.014348781 -0.02635062
```

The autocorrelation decreases in an appropiate rate.

Tests

Gelman-Rubin

```
gelman.diag(mod1_log_new)
```

```
## Potential scale reduction factors:
##
##          Point est. Upper C.I.
## a0                1       1.00
## alpha[1]          1       1.00
## alpha[2]          1       1.00
## alpha[3]          1       1.02
## alpha[4]          1       1.01
##
## Multivariate psrf
##
## 1
```

The potential scale reduction factor for all the variables is 1 or close to 1. As well, the multivariate psrf equals 1.

Heidelberger-Welch

```
heidel.diag(mod1_log_new)
```

```
## [[1]]
##
##          Stationarity start     p-value
##          test         iteration
## a0       passed       1         0.470
## alpha[1] passed       1         0.549
## alpha[2] passed       1         0.116
## alpha[3] passed       1         0.132
## alpha[4] passed       1         0.621
##
##          Halfwidth Mean    Halfwidth
##          test
## a0       passed     0.6105 0.00638
## alpha[1] passed     0.2265 0.00733
## alpha[2] passed    -0.1546 0.00840
## alpha[3] passed     0.0789 0.00723
## alpha[4] passed    -0.3179 0.00820
##
## [[2]]
##
##          Stationarity start     p-value
##          test         iteration
## a0       passed       1         0.8070
## alpha[1] passed       1         0.1981
## alpha[2] passed       1         0.6204
## alpha[3] passed       1         0.0562
## alpha[4] passed       1         0.1235
##
##          Halfwidth Mean    Halfwidth
##          test
## a0       passed     0.6017 0.00641
## alpha[1] passed     0.2233 0.00686
## alpha[2] passed    -0.1611 0.00854
## alpha[3] passed     0.0806 0.00731
## alpha[4] passed    -0.3256 0.00851
##
## [[3]]
##
##          Stationarity start     p-value
##          test         iteration
## a0       passed       1         0.525
## alpha[1] passed       1         0.138
## alpha[2] passed       1         0.972
## alpha[3] passed       1         0.410
## alpha[4] passed       1         0.771
##
##          Halfwidth Mean    Halfwidth
##          test
## a0       passed     0.6108 0.00618
## alpha[1] passed     0.2272 0.00705
## alpha[2] passed    -0.1596 0.00747
## alpha[3] passed     0.0844 0.00725
## alpha[4] passed    -0.3191 0.00825
```

As we can see, all test are passed, so we can argue that the draws come from a stationary distribution.

Effective Sample Size

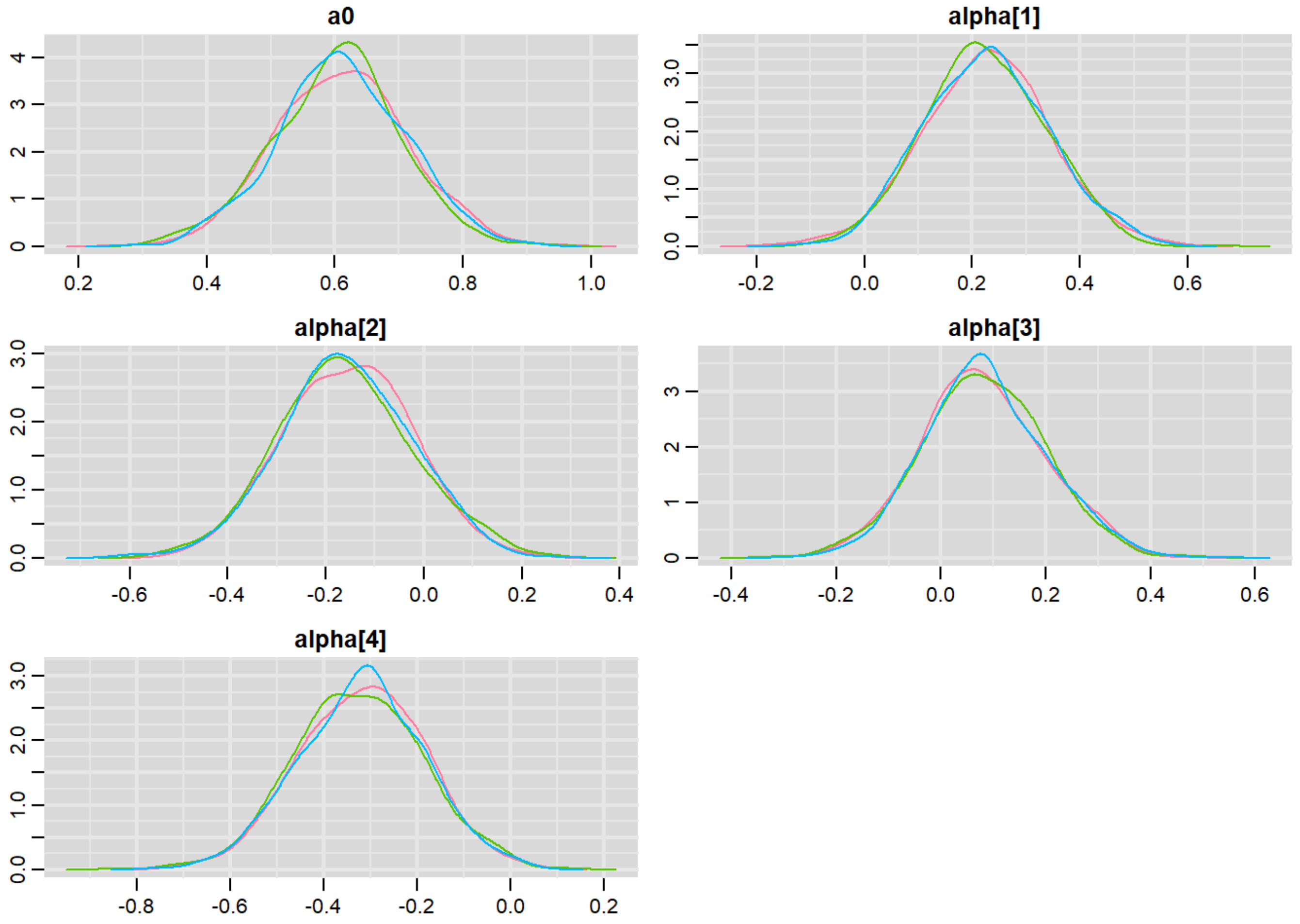
```
effectiveSize(mod1_log_new)
```

```
##       a0 alpha[1] alpha[2] alpha[3] alpha[4]
## 2930.200 3012.526 3126.430 3019.832 3000.000
```

All variables have a big enough Effective Sample Size to assume that the model converges.

POSTERIOR DENSITIES

```
denplot(mod1_log_new, parms= c('a0','alpha'))
```



FREQUENTIST INFERENCE

```
freq_mod = glm(data1$data.Round ~ data1$wing+data1$vert+data1$body+data1$sprint, family=binomial(link="logit"), d
ata = data1)
summary(freq_mod)
```

```
##
## Call:
## glm(formula = data1$data.Round ~ data1$wing + data1$vert + data1$body +
##     data1$sprint, family = binomial(link = "logit"), data = data1)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.7529  -1.3568   0.8455   0.9516   1.2593
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)   0.60231    0.10052   5.992 2.07e-09 ***
## data1$wing    0.22277    0.11136   2.000   0.0455 *
## data1$vert   -0.15615    0.12960  -1.205   0.2283
## data1$body    0.07649    0.11484   0.666   0.5054
## data1$sprint -0.31675    0.13330  -2.376   0.0175 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 577.20  on 442  degrees of freedom
## Residual deviance: 568.54  on 438  degrees of freedom
## AIC: 578.54
##
## Number of Fisher Scoring iterations: 4
```

FREQUENTIST VS BAYESIAN FIRST MODEL

```
comparison1=as.table(cbind(freq_mod$coefficients,colMeans(mod1_logch)))
colnames(comparison1) <- c("Frequentist","Bayesian_first_model")
comparison1
```

```
##                Frequentist Bayesian_first_model
## (Intercept)     0.60230740           0.14420758
## data1$wing      0.22276913           0.05050534
## data1$vert     -0.15614645          -0.01691650
## data1$body      0.07649066           0.01939503
## data1$sprint   -0.31674968          -0.98408737
```

Coefficients differ quite a lot, likely because the bayesian model doesn’t converge.

FREQUENTIST VS BAYESIAN IMPROVED MODEL

```
comparison=as.table(cbind(freq_mod$coefficients,colMeans(mod1_logch_new)))
colnames(comparison) <- c("Frequentist","Bayesian")
comparison
```

```
##                Frequentist    Bayesian
## (Intercept)     0.60230740  0.60767761
## data1$wing      0.22276913  0.22564558
## data1$vert     -0.15614645 -0.15845986
## data1$body      0.07649066  0.08129055
## data1$sprint   -0.31674968 -0.32086536
```

Values look quite similar, probably due to the improvements for convergence used.

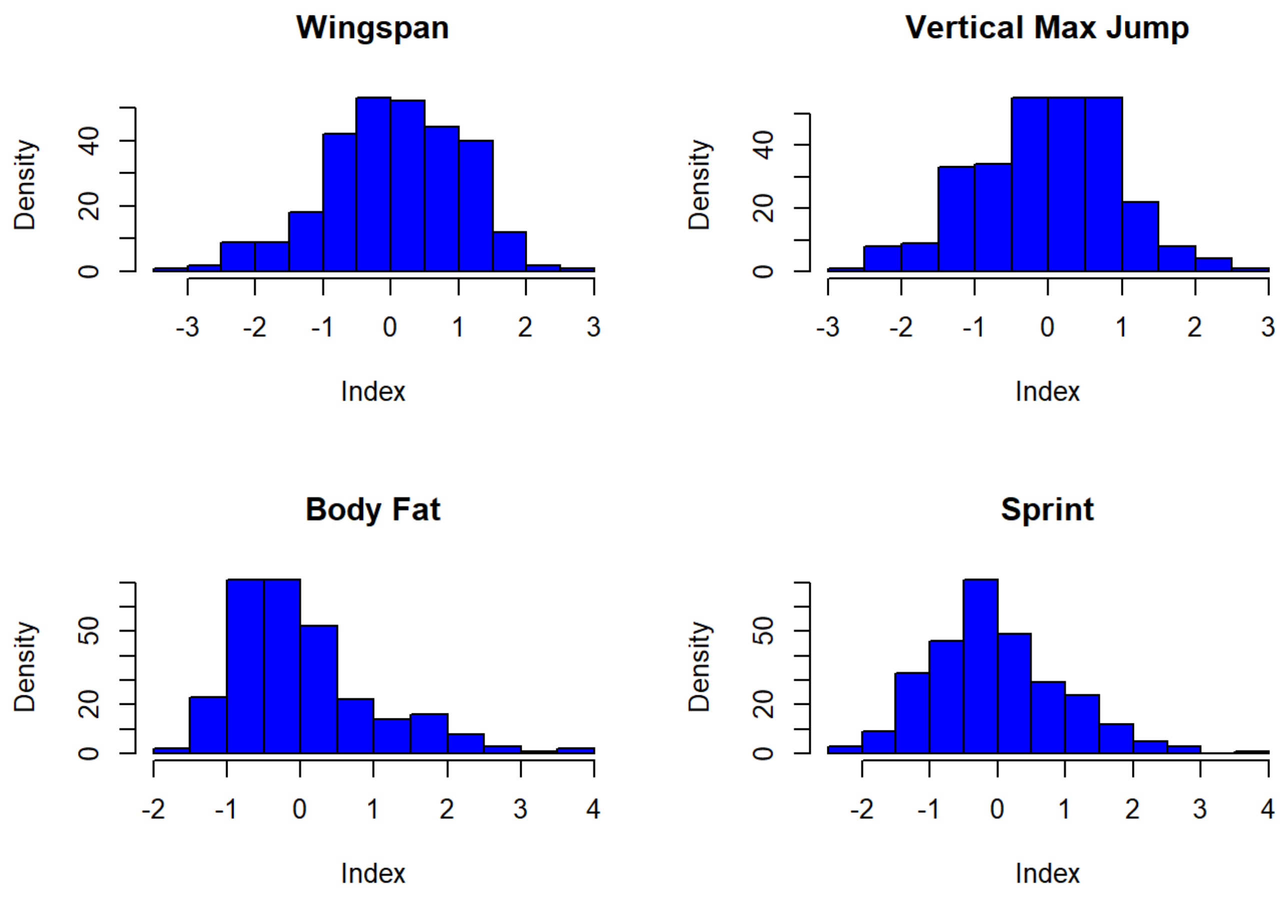
DOES THE MODEL RECOVER SOME FEATURES OF THE OBSERVED DATA?

What happens when we only take values when the response variable equals 1? Do the results match with the coefficients of the bayesian model?

```r
data11 = data1[data1$data.Round == 1, ]
head(data11)
```

```
##   data.Round       wing        vert       body     sprint
## 1          1  0.2308693  0.09240165  0.4494468 -0.1544190
## 2          1 -0.3544518  0.51440096 -0.8691289 -0.9335493
## 3          1 -0.0292734 -0.04826478 -1.1668718 -1.2452014
## 4          1  0.2959050 -0.32959765  0.5770509 -0.2323321
## 5          1  1.0763333 -0.61093052 -0.8265942  0.3909721
## 6          1  1.1413690 -2.01759488 -0.6989901  1.9492327
```

```r
par(mfrow=c(2,2))
hist(main='Wingspan',data11$wing,xlab='Index',ylab='Density',col='blue')
hist(main='Vertical Max Jump',data11$vert,xlab='Index',ylab='Density',col='blue')
hist(main='Body Fat',data11$body,xlab='Index',ylab='Density', col='blue')
hist(main='Sprint',data11$sprint,xlab='Index',ylab='Density',col='blue')
```



```
colMeans(mod1_logch_new)
```

```
##          a0    alpha[1]    alpha[2]    alpha[3]    alpha[4]
##  0.60767761  0.22564558 -0.15845986  0.08129055 -0.32086536
```

In case of the ‘wingspan’ variable, it is interesting to see that its the largest positive value. This might suppose a good starting point to draw an hypothesis stating that wingspan is important when a franchise chooses a player. As well, although both ‘body fat’ and ‘sprint’ present fat tails, its alphas are positive and negative respectively.