Project

Kentaro Kato (1851049)

At which age, baseball players can hit the biggest number of homeruns.

The goal of this project is to predict the age that brings the highest number of homeruns by using hierarchical Bayesian models. This time, I define homerun rate as the number of homeruns over the number of at-bats (the a player's turn at batting except for Walk or Hit-by-pitch).

On this project, I used MLB (Major League Baseball) data from the site https://www.retrosheet.org/gamelogs/index.html

Import libraries

```
set.seed(1234)
library(dplyr)
```

```
## Warning: package 'dplyr' was built under R version 3.5.2
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(ggplot2)
library(rstan)
```

```
## Warning: package 'rstan' was built under R version 3.5.2
```

```
## Loading required package: StanHeaders
```

```
## Warning: package 'StanHeaders' was built under R version 3.5.2
```

```
## rstan (Version 2.19.2, GitRev: 2e1f913d3ca3)
```

```
## For execution on a local, multicore CPU with excess RAM we recommend calling
## options(mc.cores = parallel::detectCores()).
## To avoid recompilation of unchanged Stan programs, we recommend calling
## rstan_options(auto_write = TRUE)
```

```
library(bayesplot)
```

```
## Warning: package 'bayesplot' was built under R version 3.5.2
```

```
## This is bayesplot version 1.7.0
```

```
## - Online documentation and vignettes at mc-stan.org/bayesplot
```

```
## - bayesplot theme set to bayesplot::theme_default()
```

```
##    * Does _not_ affect other ggplot2 plots
```

```
##    * See ?bayesplot_theme_set for details on theme setting
```

```
library(bridgesampling)
```

```
## Warning: package 'bridgesampling' was built under R version 3.5.2
```

Import data

```
# This is data about batting
bat <- read.csv("Batting.csv", header = TRUE, stringsAsFactors = FALSE)
bat <- as.data.frame(bat)
dim(bat)
```

```
## [1] 105861     22
```

```
# This is data about players
players <- read.csv("People.csv", header = TRUE)
dim(players)
```

```
## [1] 19617    24
```

```
head(bat)
```

```
##    playerID yearID stint teamID lgID  G  AB  R  H X2B X3B HR RBI SB CS BB
## 1 abercda01   1871     1    TRO <NA>  1   4  0  0   0   0  0   0  0  0  0
## 2  addybo01   1871     1    RC1 <NA> 25 118 30 32   6   0  0  13  8  1  4
## 3 allisar01   1871     1    CL1 <NA> 29 137 28 40   4   5  0  19  3  1  2
## 4 allisdo01   1871     1    WS3 <NA> 27 133 28 44  10   2  2  27  1  1  0
## 5 ansonca01   1871     1    RC1 <NA> 25 120 29 39  11   3  0  16  6  2  2
## 6 armstbo01   1871     1    FW1 <NA> 12  49  9 11   2   1  0   5  0  1  0
##   SO IBB HBP SH SF GIDP
## 1  0  NA  NA NA NA    0
## 2  0  NA  NA NA NA    0
## 3  5  NA  NA NA NA    1
## 4  2  NA  NA NA NA    0
## 5  1  NA  NA NA NA    0
## 6  1  NA  NA NA NA    0
```

Data Preprocessing

As the goal I mentioned, we need only the data about age, homerun, and at-bat. In this section, I extract these data from original data.

```
# Add "age" data to bat dataframe
bat$birthYear <- players$birthYear[match(bat$playerID, players$playerID)]
bat$age <- bat$yearID - bat$birthYear

# Add PA(Plate Apearance) column
bat[is.na(bat)] <- 0
bat$PA  <- bat$AB + bat$BB + bat$HBP + bat$SH + bat$SF # Plate Appearance

# Get sum of total plate appearance for each player's baseball life
bat <- bat %>%
  group_by(playerID) %>%
  mutate(PA_Carrer = sum(PA))

# Cosinder only players born after 1960 and with more than 1000 PA in total
new_bat <- subset(bat, birthYear > 1960 & PA_Carrer > 1000 )

# Extract only playerID, AB, HR and age
my_data <- new_bat %>%
  select(playerID, AB, HR, age)
head(my_data)
```

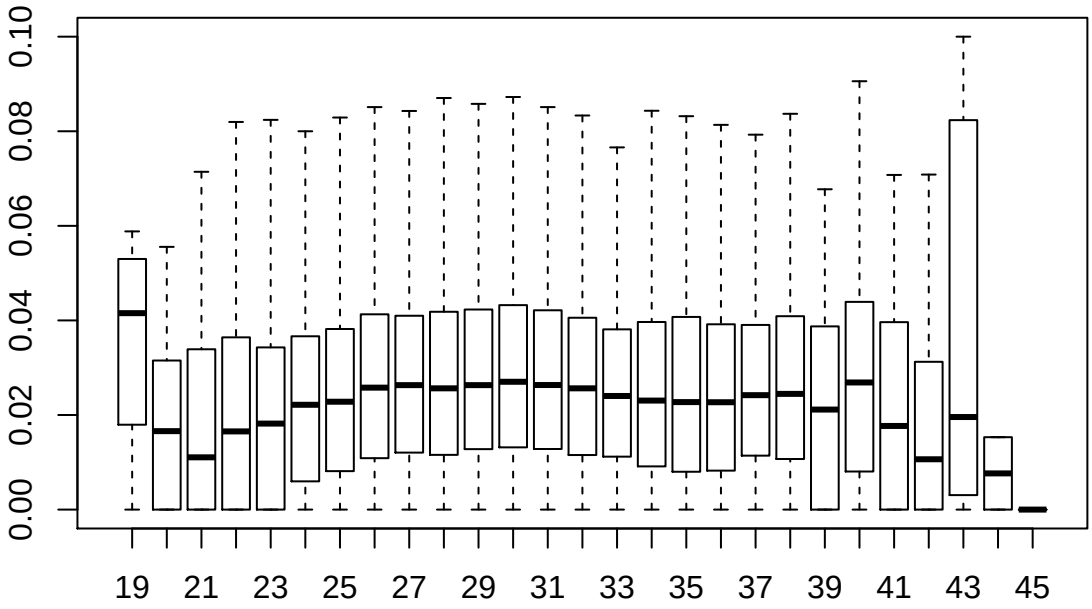
```
## # A tibble: 6 x 4
## # Groups:   playerID [5]
##   playerID     AB    HR   age
##   <chr>     <int> <int> <dbl>
## 1 manrifr01    28     0    20
## 2 bellira01     2     0    21
## 3 mattido01    12     0    21
## 4 bellira01     1     0    22
## 5 colesda01    92     1    21
## 6 daultda01     3     0    21
```

Data Analysis

Finally, we are ready to explore the dataset. Before doing any bayesian analysis, we will take a look at the original data.

Boxplot

```
p_HR <- my_data %>%
  mutate(rate = HR/AB)
boxplot(rate ~ age, data = p_HR, outline=F)
```



rate of homerun by age groups

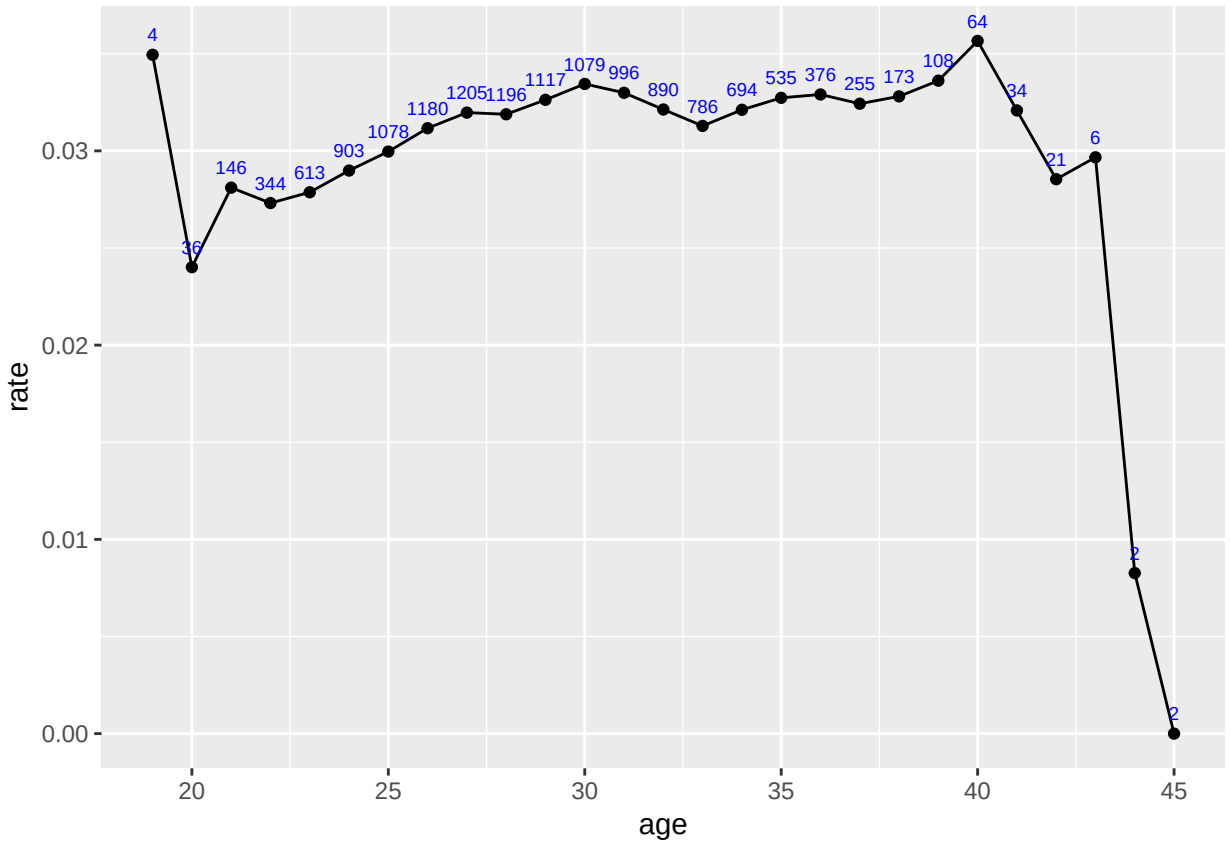
```
p_HR <- my_data %>%
  group_by(age) %>%
  mutate(num.HR = sum(HR)) %>%
  mutate(num.AB = sum(AB)) %>%
  mutate(rate = num.HR/num.AB) %>%
  mutate(count = n())

p_HR <- p_HR %>%
  select(age, num.HR, num.AB, rate, count) %>%
  distinct()

p_HR <- as.data.frame(p_HR)
p_HR <- p_HR[order(p_HR["age"]), ]
p_HR <- select(p_HR, age, rate, count)
rownames(p_HR) <- NULL
head(p_HR)
```

```
##   age       rate count
## 1  19 0.03494624     4
## 2  20 0.02401423    36
## 3  21 0.02810577   146
## 4  22 0.02731352   344
## 5  23 0.02786924   613
## 6  24 0.02898672   903
```

```
ggplot(p_HR, aes(x = age)) +
  geom_point(aes(y = rate), stat = "identity") +
  geom_line(aes(y = rate), stat = "identity", color = "black") +
  geom_text(aes(y = rate, label = count), color = "blue", size = 2.5, vjust = -1)
```



According to the graph, at first, players 29 years old have quite high probability of homeruns. From 20 years old, the rate is slight increasing up to 30. After 30, it relatively remains at the similar level even if it suddenly go up to the top rate at age 40. But before concluding the relationship between ages and homeruns with this result, we have to deal with some concerns.

1) Differences among players are not considered

It seems that players with aged between 35 and 40 likely hit as much homeruns as younger players. This occurs because ,around these age, most of players retire and only good players still play.

2) There might be a big noise

Look at the point of 40 years old. This is the highest point overall. It is hardly to imagine that 40-year-old players have the most strong power among all players to hit homeruns. There may be a prominent player who hit many homeruns with very few chances.

3) The number of data is different among ages

For example, after the age 40, there are a few players because a lot of players retire before 40 years old. Thus, the distribution of these age is not trusted compared to other ages.

Outstanding problems

Shortly, we can see the problem at age 19, 40, 45.

- at age 19, the probability is quite high while it is 0 at age 45. It may be because the number of data is small.

- at age 40, the probability reaches suddenly top overall. We can guess that there are very prominent players although the other players, who has the same age, already retired.

Age mapping

To implement the bayesian modeling, I arrange the number for each age.

```
# age mapping
age.map <- data.frame(matrix(ncol = 2, nrow = 0))
age_data <- sort(unique(my_data$age))
for (i in 1:length(age_data)){
  new <- data.frame(i, age_data[i])
  age.map <- rbind(age.map, new)
}
colnames(age.map) <- c("num", "age")
my_data$age.num <- age.map$num[match(my_data$age, age.map$age)]
head(my_data)
```

```
## # A tibble: 6 x 5
## # Groups:   playerID [5]
##   playerID     AB    HR   age age.num
##   <chr>     <int> <int> <dbl>   <int>
## 1 manrifr01    28     0    20       2
## 2 bellira01     2     0    21       3
## 3 mattido01    12     0    21       3
## 4 bellira01     1     0    22       4
## 5 colesda01    92     1    21       3
## 6 daultda01     3     0    21       3
```

Hierarchical Bayesian models

Rstan

Stan is a new-ish language that offers a more comprehensive approach to learning and implementing Bayesian models that can fit **complex data structures**.

First, I tried to solve this problem by rjags, but it did not work because I have got some software problem which I can not fix by myself. That is why I decide to use Rstan not Rjapgs.

A Stan program has three required "blocks" (other blocks, which is not required, exist as followed in next chunk):

1. **data** block : where you declare the data types, their dimensions, any restrictions (i.e. upper = or lower = , which act as checks for Stan), and their names. Any names you give to your Stan program will also be the names used in other blocks.
2. **parameter** block : This is where you indicate the parameters you want to model, their dimensions, restrictions, and name. For a linear regression, we will want to model the intercept, any slopes, and the standard deviation of the errors around the regression line.
3. **model** block : This is where you include any sampling statements, including the ???likelihood??? (model) you are using. The model block is where you indicate any prior distributions you want to include for your parameters.

To use Rstan, We need to set some parameters to run it.

- **number of iterations**: 1000. The total lenghth of the Markov Chain. I set only 1000 because the effective sample sizes of most parameters are less than 1000.
- **number of chains**: 2
- **warmup (aka burn-in)**: 200. The number of iterations to discard at the beginning

```
# Rstan parameters
my.iter <- 1000
my.warmup <- 500
my.chains <- 2
my.refresh <- my.iter/10
num.lag <- my.iter/10 # for autocorrelation
```

Data

This is the data for simulation

```
my.data = list(N     = nrow(my_data),
               N_age = length(unique(my_data$age)),
               HR    = my_data$HR,
               AB    = my_data$AB,
               age   = my_data$age.num)
```

Graph after MCMC

I create a function to plot a graph after MCMC

```
plot_graph <- function(fit){
  # draw estimation (p_age)
  p_age.summary <- summary(fit, pars = c("p_age"),
                           probs = c(0.025, 0.25, 0.5, 0.75, 0.975))$summary
  p_age.summary <- p_age.summary[, c("2.5%", "25%","50%", "75%", "97.5%" )]
  p_HR_graph <- cbind(p_HR, p_age.summary)
  colnames(p_HR_graph) <- c("age", "p_data","count", "p_2.5", "p_25", "p_50", "p_75", "p_97.5")
  a <- 2
  a

  ggplot(p_HR_graph, aes(x = age)) +
    geom_point(aes(y = p_data), stat = "identity") +
    geom_line(aes(y = p_data), stat = "identity", color = "black") +
    geom_line(aes(y = p_2.5), stat = "identity", color = "red", alpha = 0.3) +
    geom_line(aes(y = p_25), stat = "identity", color = "red", alpha = 0.6) +
    geom_point(aes(y = p_50), stat = "identity", color = "red", alpha = 1) +
    geom_line(aes(y = p_50), stat = "identity", color = "red") +
    geom_line(aes(y = p_75), stat = "identity", color = "red", alpha = 0.6) +
    geom_line(aes(y = p_97.5), stat = "identity", color = "red", alpha = 0.3)
}
```

Model 1 (Considering only age without beta)

I define the first bayesian model cosidering only age below.

$$HR[i] \sim \text{Binomial}(AB[i], p[i])$$
$$p[i] = \text{Logistic}(r_{age}[age[i]])$$

$$\text{where} \quad i = 1, 2, ..., N$$
$$r_{age}[j] \sim \text{Normal}(0, s_{age})$$
$$\text{where} \quad j = 1, 2, ..., N_{age}$$

The number of homeruns HR follows binomial distribution with parameters the number of at-bats AB and the probability of homeruns at age i.

Also the probability of homeruns at age p_i follows logistic function with the variable depending on the age of the player r_{age}.

And, I defined the factor of homerun depending on the ages r_{age} follows normal distribution with mean is 0 and standard deviation is s_{age}

Logistic Regression

Logistic Regression is used when the dependent variable(target) is binary number. In this case, I need to know whether the batter hit homerun (1) or not (0).

```
# Model

# The data block reads external information.
# data {
#     int N;
#     int N_age;
#     int HR[N];
#     int AB[N];
#     int age[N];
# }
#
# The parameters block defines the sampling space.
# parameters {
#     real<lower=0> s_age; # float
#     vector[N_age] r_age;
# }
#
# The transformed parameters block allows for parameter processing before the posterior is computed.
# transformed parameters {
#     vector[N] p;
#
#     for (i in 1:N)
#         p[i] = inv_logit(r_age[age[i]]);
# }
#
# In the model block we define our posterior distributions.
```

```
# model {
#     r_age ~ normal(0, s_age);
#     HR ~ binomial(AB, p);
# }
#
# The generated quantities block allows for postprocessing.
# generated quantities {
#     real p_age[N_age];
#     for (i in 1:N_age)
#         p_age[i] = inv_logit(r_age[i]);
# }
```

```
fit1 <- stan(file = "model_2.stan", data = my.data,
             chains = my.chains, warmup = my.warmup, iter = my.iter,
              cores = 2, refresh = my.refresh)
```

Summary 1

Now we can print the results of our model.

```
list_of_draws <- rstan::extract(fit1)
print(names(list_of_draws))
```

```
## [1] "s_age" "r_age" "p"     "p_age" "lp__"
```

These are the parameters that we can see the trace from MCMC using Rstan. Let's take a look only β, s_{age}, r_{age}.

Statistics

For models fit using MCMC, also included in the summary are the **mean**, the Monte Carlo standard error (**se_mean**), the standard deviation (**sd**), **quantiles** (2.5%, 25%, 50%, 75%, 97.5%) the effective sample size (**n_eff**), and the R-hat statistic (**Rhat**).

- **se_mean**: The Monte Carlo standard error is the uncertainty about a statistic in the sample due to sampling error. The estimated standard deviation of a parameter divided by the square root of the number of effective samples

$$MCSE = \frac{sd}{\sqrt{N_{eff}}}$$

- **n_eff**: The number of effective samples. The effective sample size is an estimate of the sample size required to achieve the same level of precision if that sample was a simple random sample.
- **Rhat** : In equilibrium, the distribution of samples from chains should be the same regardless of the initial starting values of the chains. When these are at or near 1, the chains have converged. Values greater than 1.1 indicate inadequate convergence.

(resourse: https://github.com/stan-dev/cmdstan/blob/develop/src/cmdstan/stansummary.cpp#L119)

```r
paras1 <- c("s_age")
for (i in 1:27){
  para <- paste("r_age[", i, sep="")
  para <- paste(para, "]", sep="")
  paras1 <- c(paras1, para)
}
print(fit1,  pars = paras1, digits=4)
```

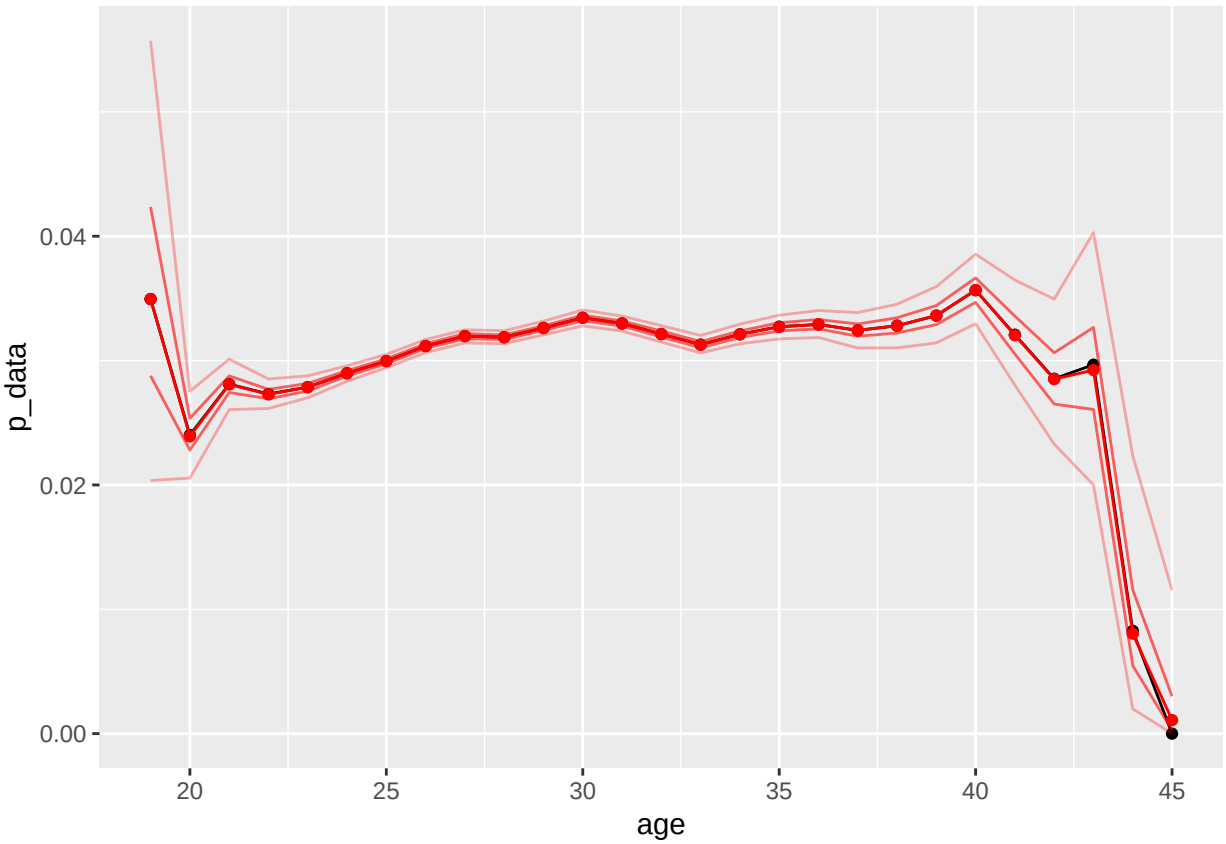
```
## Inference for Stan model: model_2.
## 2 chains, each with iter=1000; warmup=500; thin=1;
## post-warmup draws per chain=500, total post-warmup draws=1000.
##
##              mean se_mean     sd     2.5%     25%     50%     75%   97.5%
## s_age      3.8810  0.0177 0.5761   2.9558  3.4629  3.8388  4.1995  5.2296
## r_age[1]  -3.3281  0.0089 0.2818  -3.8737 -3.5189 -3.3181 -3.1182 -2.8303
## r_age[2]  -3.7066  0.0020 0.0786  -3.8639 -3.7576 -3.7095 -3.6492 -3.5645
## r_age[3]  -3.5444  0.0014 0.0379  -3.6211 -3.5682 -3.5439 -3.5192 -3.4722
## r_age[4]  -3.5731  0.0006 0.0223  -3.6175 -3.5871 -3.5739 -3.5588 -3.5282
## r_age[5]  -3.5525  0.0004 0.0163  -3.5841 -3.5637 -3.5524 -3.5413 -3.5190
## r_age[6]  -3.5118  0.0003 0.0115  -3.5350 -3.5196 -3.5116 -3.5043 -3.4905
## r_age[7]  -3.4778  0.0002 0.0094  -3.4961 -3.4846 -3.4776 -3.4715 -3.4591
## r_age[8]  -3.4367  0.0002 0.0089  -3.4545 -3.4427 -3.4365 -3.4306 -3.4193
## r_age[9]  -3.4106  0.0002 0.0090  -3.4283 -3.4172 -3.4105 -3.4042 -3.3943
## r_age[10] -3.4136  0.0002 0.0087  -3.4304 -3.4192 -3.4136 -3.4075 -3.3966
## r_age[11] -3.3897  0.0002 0.0089  -3.4073 -3.3956 -3.3897 -3.3841 -3.3722
## r_age[12] -3.3642  0.0002 0.0101  -3.3842 -3.3711 -3.3641 -3.3576 -3.3446
## r_age[13] -3.3784  0.0002 0.0095  -3.3976 -3.3846 -3.3783 -3.3721 -3.3597
## r_age[14] -3.4049  0.0003 0.0111  -3.4261 -3.4127 -3.4050 -3.3971 -3.3836
## r_age[15] -3.4329  0.0003 0.0121  -3.4552 -3.4410 -3.4331 -3.4249 -3.4089
## r_age[16] -3.4057  0.0003 0.0127  -3.4307 -3.4141 -3.4056 -3.3973 -3.3799
## r_age[17] -3.3868  0.0004 0.0153  -3.4178 -3.3970 -3.3869 -3.3763 -3.3575
## r_age[18] -3.3801  0.0005 0.0175  -3.4142 -3.3919 -3.3804 -3.3682 -3.3460
## r_age[19] -3.3957  0.0007 0.0228  -3.4417 -3.4108 -3.3956 -3.3795 -3.3511
## r_age[20] -3.3838  0.0007 0.0281  -3.4415 -3.4024 -3.3838 -3.3638 -3.3306
## r_age[21] -3.3583  0.0010 0.0351  -3.4282 -3.3811 -3.3582 -3.3339 -3.2891
## r_age[22] -3.2972  0.0010 0.0406  -3.3789 -3.3260 -3.2969 -3.2690 -3.2162
## r_age[23] -3.4092  0.0021 0.0701  -3.5462 -3.4581 -3.4084 -3.3598 -3.2740
## r_age[24] -3.5301  0.0026 0.1072  -3.7365 -3.6036 -3.5286 -3.4547 -3.3184
## r_age[25] -3.5092  0.0055 0.1773  -3.8918 -3.6206 -3.5027 -3.3881 -3.1704
## r_age[26] -4.8548  0.0201 0.6108  -6.2162 -5.2035 -4.8150 -4.4478 -3.7785
## r_age[27] -7.1625  0.0905 1.9437 -12.0112 -8.1441 -6.8207 -5.8069 -4.4504
##           n_eff   Rhat
## s_age      1060 0.9983
## r_age[1]   1013 0.9994
## r_age[2]   1579 0.9985
## r_age[3]    716 0.9998
## r_age[4]   1512 0.9989
## r_age[5]   1482 1.0027
## r_age[6]   1519 0.9995
## r_age[7]   2106 0.9994
## r_age[8]   1595 0.9993
## r_age[9]   1979 1.0007
## r_age[10]  2181 0.9982
```

```
## r_age[11]  2106 0.9989
## r_age[12]  2030 0.9993
## r_age[13]  2527 0.9983
## r_age[14]  1071 1.0038
## r_age[15]  1669 0.9985
## r_age[16]  1363 0.9994
## r_age[17]  1315 0.9989
## r_age[18]  1041 0.9980
## r_age[19]  1067 0.9988
## r_age[20]  1422 0.9989
## r_age[21]  1160 0.9983
## r_age[22]  1609 0.9993
## r_age[23]  1120 0.9992
## r_age[24]  1636 0.9982
## r_age[25]  1038 0.9987
## r_age[26]   921 0.9984
## r_age[27]   461 1.0012
##
## Samples were drawn using NUTS(diag_e) at Fri Sep 20 21:36:40 2019.
## For each parameter, n_eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor on split chains (at
## convergence, Rhat=1).
```

n_eff is quite high. Even if I set this number higher, It showed much higher one. I can say this model is not nice.

Graph after MCMC

```
plot_graph(fit1)
```



This graph is almost same as initial graph. We can say that this model does not do anything. Next, we will add a constant beta to make a better model.

Model 2 (Considering only age with beta)

I define the first bayesian model cosidering only age below.

$$HR[i] \sim \text{Binomial}(AB[i], p[i])$$
$$p[i] = \text{Logistic}(\beta + r_{age}[age[i]])$$

$$\text{where} \quad i = 1, 2, ..., N$$
$$r_{age}[j] \sim \text{Normal}(0, s_{age})$$
$$\text{where} \quad j = 1, 2, ..., N_{age}$$

This time, I changed the probability of homeruns at age p_i follows logistic function with constant β and the variable depending on the age of the player r_{age}.

```
# Model

# The data block reads external information.
# data {
#     int N;
```

```
#     int N_age;
#     int HR[N];
#     int AB[N];
#     int age[N];
# }
#
# The parameters block defines the sampling space.
# parameters {
#     real beta;            # float
#     real<lower=0> s_age;
#     vector[N_age] r_age;
# }
#
# The transformed parameters block allows for parameter processing before the posterior is computed.
# transformed parameters {
#     vector[N] p;
#
#     for (i in 1:N)
#         p[i] = inv_logit(beta + r_age[age[i]]);
# }
#
# In the model block we define our posterior distributions.
# model {
#     r_age ~ normal(0, s_age);
#     HR ~ binomial(AB, p);
# }
#
# The generated quantities block allows for postprocessing.
# generated quantities {
#     real p_age[N_age];
#     for (i in 1:N_age)
#         p_age[i] = inv_logit(beta + r_age[i]);
# }
```

```
fit2 <- stan(file = "model_1.stan", data = my.data,
             chains = my.chains, warmup = my.warmup, iter = my.iter,
              cores = 2, refresh = my.refresh)
```

```
## Warning: Bulk Effective Samples Size (ESS) is too low, indicating posterior means and medians may be
## Running the chains for more iterations may help. See
## http://mc-stan.org/misc/warnings.html#bulk-ess
```

Summary 2

Now we can print the results of our model.

```
list_of_draws <- rstan::extract(fit2)
print(names(list_of_draws))
```

```
## [1] "beta"  "s_age" "r_age" "p"     "p_age" "lp__"
```

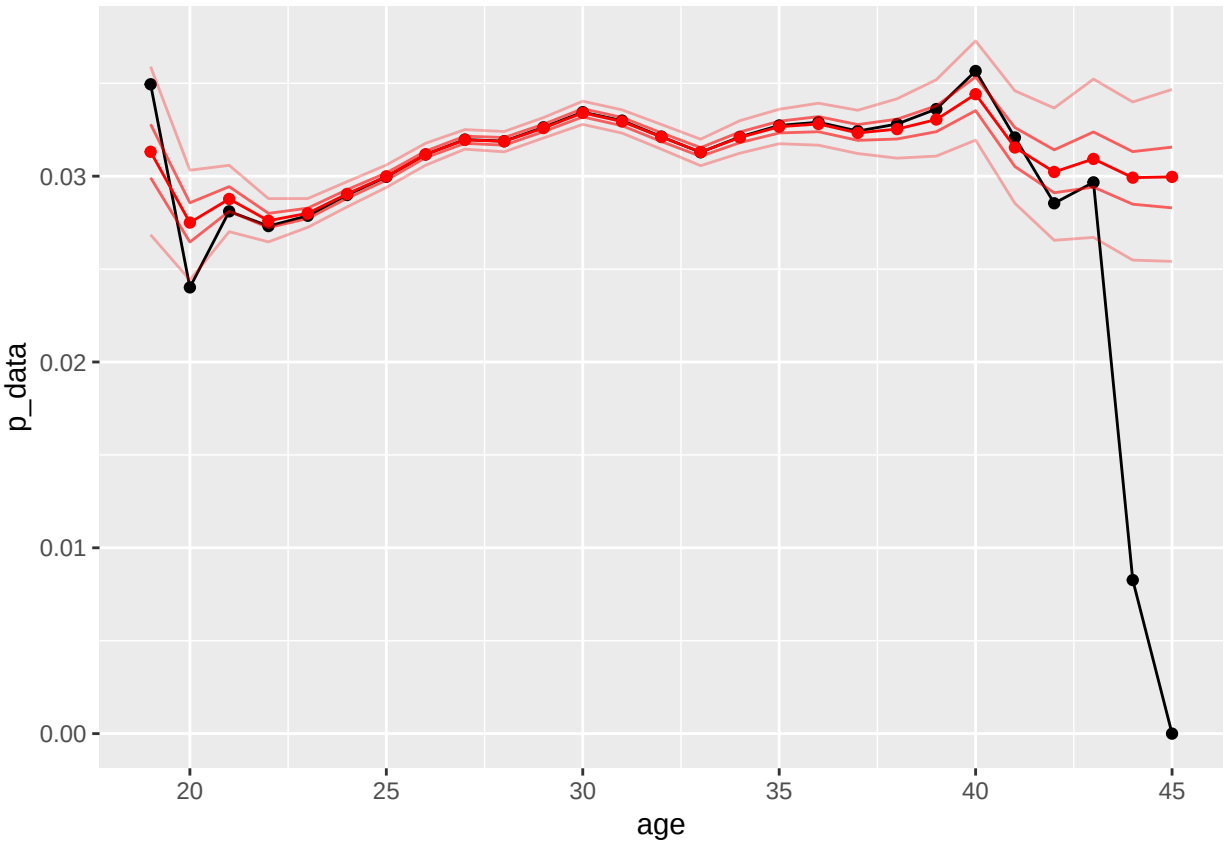
```r
paras <- c("beta", "s_age")
for (i in 1:27){
  para <- paste("r_age[", i, sep="")
  para <- paste(para, "]", sep="")
  paras <- c(paras, para)
}
print(fit2,  pars = paras, digits=4)
```

```
## Inference for Stan model: model_1.
## 2 chains, each with iter=1000; warmup=500; thin=1;
## post-warmup draws per chain=500, total post-warmup draws=1000.
##
##              mean se_mean     sd    2.5%     25%     50%     75%   97.5%
## beta      -3.4414  0.0014 0.0160 -3.4737 -3.4517 -3.4419 -3.4309 -3.4102
## s_age      0.0762  0.0008 0.0155  0.0509  0.0651  0.0744  0.0854  0.1104
## r_age[1]   0.0095  0.0025 0.0721 -0.1331 -0.0364  0.0093  0.0567  0.1549
## r_age[2]  -0.1264  0.0024 0.0575 -0.2449 -0.1650 -0.1235 -0.0867 -0.0247
## r_age[3]  -0.0784  0.0016 0.0359 -0.1461 -0.1020 -0.0793 -0.0522 -0.0093
## r_age[4]  -0.1203  0.0016 0.0267 -0.1759 -0.1372 -0.1194 -0.1024 -0.0703
## r_age[5]  -0.1056  0.0015 0.0216 -0.1494 -0.1192 -0.1063 -0.0914 -0.0620
## r_age[6]  -0.0684  0.0014 0.0199 -0.1068 -0.0807 -0.0687 -0.0561 -0.0296
## r_age[7]  -0.0353  0.0013 0.0186 -0.0730 -0.0478 -0.0344 -0.0233 -0.0010
## r_age[8]   0.0048  0.0014 0.0184 -0.0324 -0.0067  0.0055  0.0168  0.0392
## r_age[9]   0.0307  0.0015 0.0182 -0.0038  0.0186  0.0300  0.0425  0.0682
## r_age[10]  0.0277  0.0014 0.0184 -0.0075  0.0162  0.0287  0.0400  0.0623
## r_age[11]  0.0511  0.0014 0.0180  0.0155  0.0395  0.0507  0.0627  0.0870
## r_age[12]  0.0765  0.0014 0.0187  0.0408  0.0644  0.0764  0.0889  0.1121
## r_age[13]  0.0620  0.0014 0.0189  0.0276  0.0494  0.0612  0.0739  0.1010
## r_age[14]  0.0354  0.0014 0.0191  0.0006  0.0222  0.0349  0.0483  0.0727
## r_age[15]  0.0089  0.0014 0.0194 -0.0290 -0.0037  0.0092  0.0217  0.0466
## r_age[16]  0.0349  0.0015 0.0214 -0.0060  0.0210  0.0340  0.0489  0.0786
## r_age[17]  0.0527  0.0015 0.0217  0.0082  0.0379  0.0533  0.0671  0.0956
## r_age[18]  0.0574  0.0014 0.0240  0.0138  0.0416  0.0561  0.0730  0.1055
## r_age[19]  0.0429  0.0015 0.0252 -0.0064  0.0255  0.0424  0.0590  0.0951
## r_age[20]  0.0486  0.0015 0.0294 -0.0049  0.0297  0.0483  0.0667  0.1062
## r_age[21]  0.0659  0.0016 0.0357 -0.0069  0.0430  0.0669  0.0897  0.1337
## r_age[22]  0.1083  0.0019 0.0426  0.0275  0.0786  0.1083  0.1358  0.1944
## r_age[23]  0.0166  0.0019 0.0513 -0.0840 -0.0177  0.0178  0.0543  0.1085
## r_age[24] -0.0287  0.0021 0.0600 -0.1509 -0.0659 -0.0274  0.0120  0.0860
## r_age[25] -0.0064  0.0025 0.0709 -0.1505 -0.0535 -0.0051  0.0436  0.1264
## r_age[26] -0.0427  0.0028 0.0768 -0.1975 -0.0884 -0.0381  0.0076  0.0991
## r_age[27] -0.0387  0.0035 0.0840 -0.2001 -0.0952 -0.0369  0.0175  0.1142
##           n_eff   Rhat
## beta        135 1.0034
## s_age       373 1.0006
## r_age[1]    807 1.0023
## r_age[2]    595 1.0043
## r_age[3]    512 1.0005
## r_age[4]    271 1.0035
## r_age[5]    203 1.0054
## r_age[6]    207 1.0013
## r_age[7]    191 1.0024
## r_age[8]    186 1.0039
```

```
## r_age[9]    153 1.0055
## r_age[10]   163 1.0031
## r_age[11]   175 1.0039
## r_age[12]   175 1.0029
## r_age[13]   173 1.0037
## r_age[14]   193 1.0034
## r_age[15]   183 1.0041
## r_age[16]   198 1.0037
## r_age[17]   205 1.0027
## r_age[18]   287 1.0021
## r_age[19]   291 1.0021
## r_age[20]   404 0.9981
## r_age[21]   482 1.0009
## r_age[22]   520 1.0021
## r_age[23]   761 1.0010
## r_age[24]   785 1.0007
## r_age[25]   828 1.0016
## r_age[26]   770 0.9992
## r_age[27]   564 1.0031
##
## Samples were drawn using NUTS(diag_e) at Fri Sep 20 21:38:30 2019.
## For each parameter, n_eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor on split chains (at
## convergence, Rhat=1).
```

Graph after MCMC

```
plot_graph(fit2)
```



We have solved the third problem I mentioned before. Although we saw that 19-year-old players can bring a lot of homeruns and 45-year-old players can bring no homeruns at all because of the lack of data, now we predict that the number of homeruns stay at the similar level of near age. However, other than these specific ages, we still get almost same distribution.

Model 3 (considering time series)

$$HR[i] \sim \text{Binomial}(AB[i], p[i])$$
$$p[i] = \text{Logistic}(\beta + r_{age}[age[i]])$$
$$\text{where} \quad i = 1, 2, ..., N$$

$$r_{age}[j] \sim \text{Normal}(r_{age}[j-1], s_{age})$$
$$\sum_j r_{age}[j] = 0$$

$$\text{where} \quad j = 1, 2, ..., N_{age}$$

```
# Model
# data {
#     int N;
#     int N_age;
#     int HR[N];
#     int AB[N];
```

```
#     int age[N];
# }
#
# parameters {
#     real beta;
#     real<lower=0> s_age;
#     vector[N_age] r_age;
# }
#
# transformed parameters {
#     vector[N] p;
#
#     for (i in 1:N)
#         p[i] = inv_logit(beta + r_age[age[i]]);
# }
#
# model {
#     r_age[1] ~ normal(-sum(r_age[2:N_age]), 0.001);
#     r_age[2:N_age] ~ normal(r_age[1:(N_age-1)], s_age);
#     HR ~ binomial(AB, p);
# }
#
# generated quantities {
#     real p_age[N_age];
#     for (i in 1:N_age)
#         p_age[i] = inv_logit(beta + r_age[i]);
# }
```

```
fit3 <- stan(file = "model_3.stan", data = my.data,
             chains = my.chains, warmup = my.warmup, iter = my.iter,
             cores = 2, refresh = my.refresh)
```

```
## Warning: There were 153 transitions after warmup that exceeded the maximum treedepth. Increase max_t
## http://mc-stan.org/misc/warnings.html#maximum-treedepth-exceeded
```

```
## Warning: Examine the pairs() plot to diagnose sampling problems
```

```
## Warning: Bulk Effective Samples Size (ESS) is too low, indicating posterior means and medians may be
## Running the chains for more iterations may help. See
## http://mc-stan.org/misc/warnings.html#bulk-ess
```

```
## Warning: Tail Effective Samples Size (ESS) is too low, indicating posterior variances and tail quant
## Running the chains for more iterations may help. See
## http://mc-stan.org/misc/warnings.html#tail-ess
```

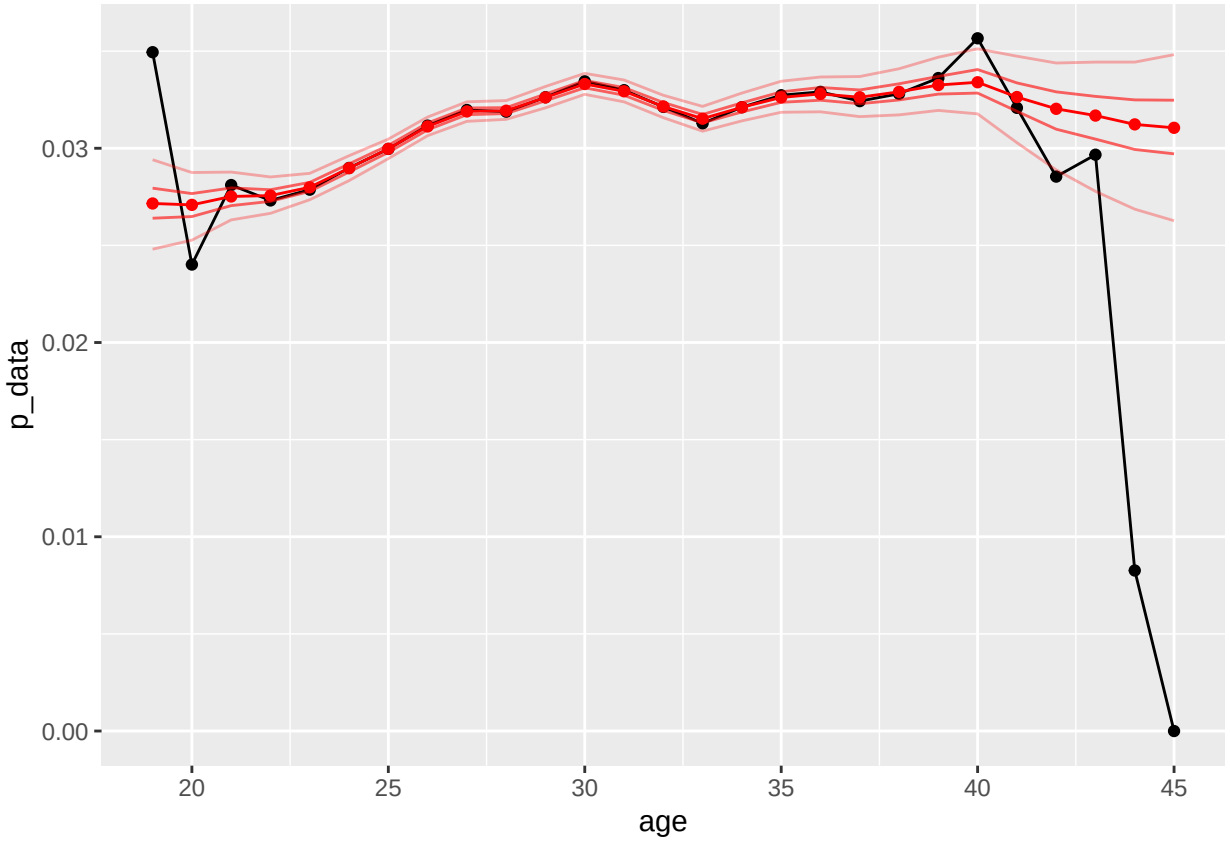
Summary 3 (Diagnostics)

```
print(fit3,  pars = paras, digits = 4)
```

```
## Inference for Stan model: model_3.
## 2 chains, each with iter=1000; warmup=500; thin=1;
## post-warmup draws per chain=500, total post-warmup draws=1000.
##
##              mean se_mean     sd    2.5%     25%     50%     75%   97.5%
## beta      -3.4392  0.0011 0.0113 -3.4658 -3.4459 -3.4380 -3.4310 -3.4212
## s_age      0.0292  0.0005 0.0069  0.0192  0.0245  0.0283  0.0327  0.0456
## r_age[1]  -0.1400  0.0017 0.0417 -0.2214 -0.1669 -0.1408 -0.1138 -0.0586
## r_age[2]  -0.1431  0.0014 0.0318 -0.2017 -0.1638 -0.1431 -0.1227 -0.0824
## r_age[3]  -0.1265  0.0012 0.0235 -0.1720 -0.1430 -0.1255 -0.1108 -0.0811
## r_age[4]  -0.1241  0.0011 0.0187 -0.1595 -0.1373 -0.1242 -0.1115 -0.0882
## r_age[5]  -0.1076  0.0010 0.0157 -0.1361 -0.1187 -0.1083 -0.0968 -0.0766
## r_age[6]  -0.0725  0.0012 0.0157 -0.0994 -0.0833 -0.0742 -0.0614 -0.0389
## r_age[7]  -0.0379  0.0012 0.0144 -0.0639 -0.0477 -0.0385 -0.0294 -0.0070
## r_age[8]   0.0009  0.0012 0.0141 -0.0247 -0.0088  0.0003  0.0100  0.0284
## r_age[9]   0.0264  0.0012 0.0142  0.0010  0.0169  0.0256  0.0353  0.0573
## r_age[10]  0.0278  0.0011 0.0137  0.0035  0.0178  0.0271  0.0367  0.0561
## r_age[11]  0.0494  0.0012 0.0144  0.0235  0.0397  0.0491  0.0584  0.0798
## r_age[12]  0.0711  0.0012 0.0143  0.0444  0.0615  0.0707  0.0797  0.1045
## r_age[13]  0.0592  0.0012 0.0145  0.0328  0.0494  0.0584  0.0687  0.0892
## r_age[14]  0.0345  0.0011 0.0143  0.0092  0.0251  0.0333  0.0433  0.0664
## r_age[15]  0.0140  0.0011 0.0146 -0.0122  0.0032  0.0131  0.0241  0.0439
## r_age[16]  0.0332  0.0012 0.0158  0.0047  0.0220  0.0325  0.0429  0.0693
## r_age[17]  0.0497  0.0012 0.0167  0.0184  0.0374  0.0486  0.0619  0.0819
## r_age[18]  0.0548  0.0011 0.0178  0.0224  0.0426  0.0542  0.0659  0.0938
## r_age[19]  0.0498  0.0009 0.0171  0.0178  0.0384  0.0493  0.0615  0.0846
## r_age[20]  0.0581  0.0009 0.0193  0.0212  0.0452  0.0577  0.0707  0.1006
## r_age[21]  0.0693  0.0010 0.0208  0.0308  0.0550  0.0684  0.0834  0.1136
## r_age[22]  0.0747  0.0013 0.0251  0.0292  0.0583  0.0733  0.0904  0.1270
## r_age[23]  0.0489  0.0018 0.0283 -0.0110  0.0313  0.0498  0.0669  0.1054
## r_age[24]  0.0261  0.0031 0.0370 -0.0538  0.0023  0.0291  0.0514  0.0902
## r_age[25]  0.0122  0.0042 0.0470 -0.0933 -0.0154  0.0177  0.0435  0.0902
## r_age[26] -0.0018  0.0055 0.0571 -0.1266 -0.0347  0.0026  0.0375  0.0909
## r_age[27] -0.0067  0.0061 0.0655 -0.1513 -0.0422 -0.0029  0.0378  0.1046
##           n_eff   Rhat
## beta        101 1.0218
## s_age       192 1.0102
## r_age[1]    605 1.0047
## r_age[2]    508 1.0058
## r_age[3]    411 0.9988
## r_age[4]    294 0.9999
## r_age[5]    236 1.0017
## r_age[6]    184 1.0108
## r_age[7]    150 1.0147
## r_age[8]    144 1.0167
## r_age[9]    147 1.0165
## r_age[10]   148 1.0144
## r_age[11]   146 1.0128
## r_age[12]   143 1.0158
## r_age[13]   154 1.0206
## r_age[14]   165 1.0127
## r_age[15]   167 1.0078
## r_age[16]   185 1.0069
## r_age[17]   180 1.0045
```

```
## r_age[18]   276 1.0055
## r_age[19]   377 1.0021
## r_age[20]   419 1.0080
## r_age[21]   461 1.0090
## r_age[22]   393 1.0061
## r_age[23]   241 0.9992
## r_age[24]   141 1.0099
## r_age[25]   123 1.0100
## r_age[26]   109 1.0147
## r_age[27]   117 1.0127
##
## Samples were drawn using NUTS(diag_e) at Fri Sep 20 21:48:52 2019.
## For each parameter, n_eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor on split chains (at
## convergence, Rhat=1).
```

```
plot_graph(fit3)
```



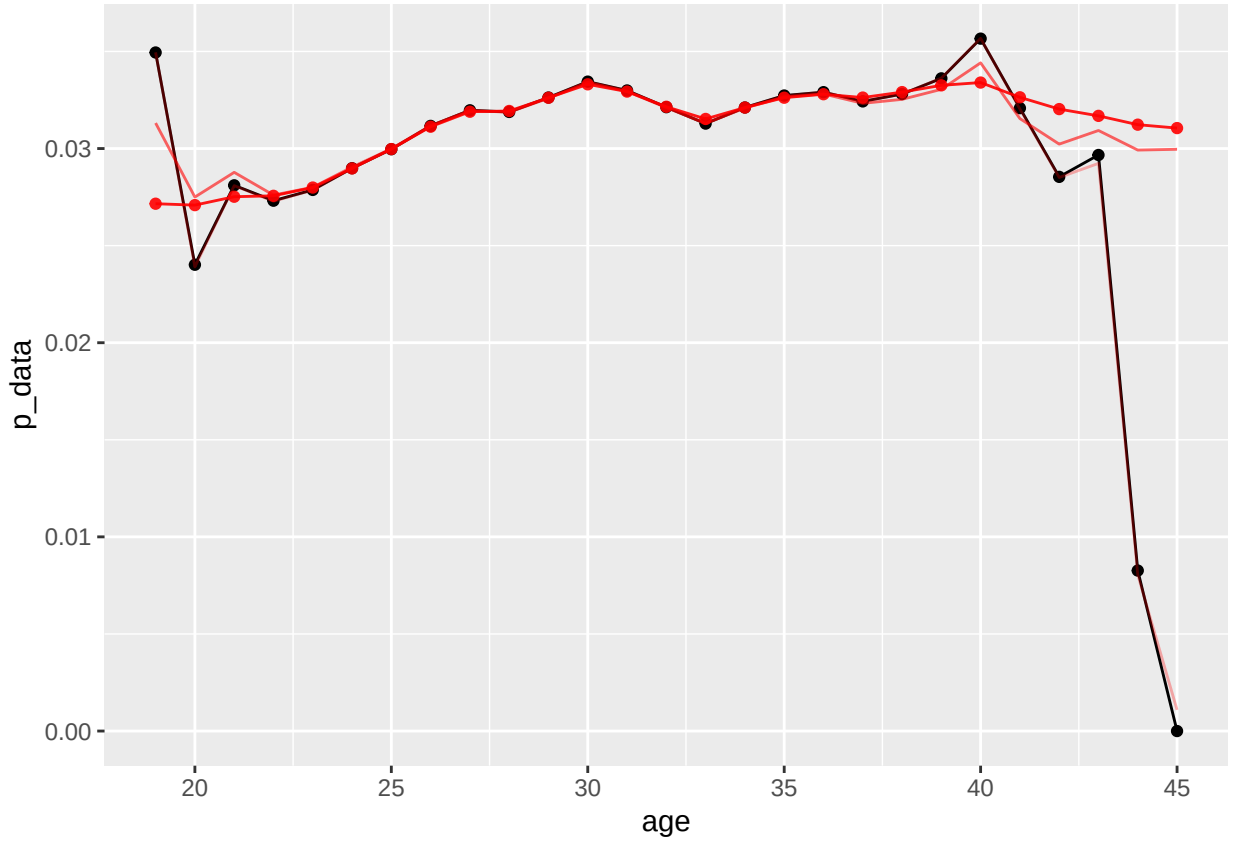
Model Comparison

```
p_age.summary1 <- summary(fit1, pars = c("p_age"),  probs = 0.5)$summary
p_age.summary1 <- p_age.summary1[, "50%"]
p_HR1 <- cbind(p_HR, p_age.summary1)
```

```
p_age.summary2 <- summary(fit2, pars = c("p_age"),  probs = 0.5)$summary
p_age.summary2 <- p_age.summary2[, "50%"]
p_HR2 <- cbind(p_HR1, p_age.summary2)

p_age.summary3 <- summary(fit3, pars = c("p_age"),  probs = 0.5)$summary
p_age.summary3 <- p_age.summary3[, "50%"]
p_HR3 <- cbind(p_HR2, p_age.summary3)
colnames(p_HR3) <- c("age", "p_data","count", "model1", "model2", "model3")

ggplot(p_HR3, aes(x = age)) +
  geom_point(aes(y = p_data), stat = "identity") +
  geom_line(aes(y = p_data), stat = "identity", color = "black") +
  geom_line(aes(y = model1), stat = "identity", color = "red", alpha = 0.3) +
  geom_line(aes(y = model2), stat = "identity", color = "red", alpha = 0.6) +
  geom_point(aes(y = model3), stat = "identity", color = "red", alpha = 0.9) +
  geom_line(aes(y = model3), stat = "identity", color = "red", alpha = 0.9)
```



Marginal Likelihood

Computing the (log) marginal likelihoods via the bridge_sampler function is now easy: we only need to pass the stanfit objects which contain all information necessary. I use silent = TRUE to suppress printing the number of iterations to the console

```r
# compute log marginal likelihood via bridge sampling for all models
bridge1 <- bridge_sampler(fit1, silent = TRUE)
bridge2 <- bridge_sampler(fit2, silent = TRUE)
bridge3 <- bridge_sampler(fit3, silent = TRUE)

print(bridge1)
```

```
## Bridge sampling estimate of the log marginal likelihood: -568335.9
## Estimate obtained in 7 iteration(s) via method "normal".
```

```r
print(bridge2)
```

```
## Bridge sampling estimate of the log marginal likelihood: -568254.9
## Estimate obtained in 7 iteration(s) via method "normal".
```

```r
print(bridge3)
```

```
## Bridge sampling estimate of the log marginal likelihood: -568248.6
## Estimate obtained in 7 iteration(s) via method "normal".
```

Bayes Factor

To compare the null model and the alternative model, we can compute the Bayes factor by using the bf function. In our case, we compute BF01, that is, the Bayes factor which quantifies how much more likely the data are under the null versus the alternative model

```r
# compute Bayes factor
BF1_2 <- bf(bridge2, bridge1)
print(BF1_2)
```

```
## Estimated Bayes factor in favor of bridge2 over bridge1: 145555397536624958082311524886511616.00000
```

```r
BF2_3 <- bf(bridge3, bridge2)
print(BF2_3)
```

```
## Estimated Bayes factor in favor of bridge3 over bridge2: 561.96091
```

We can see each model gets better based on these bayes factor.

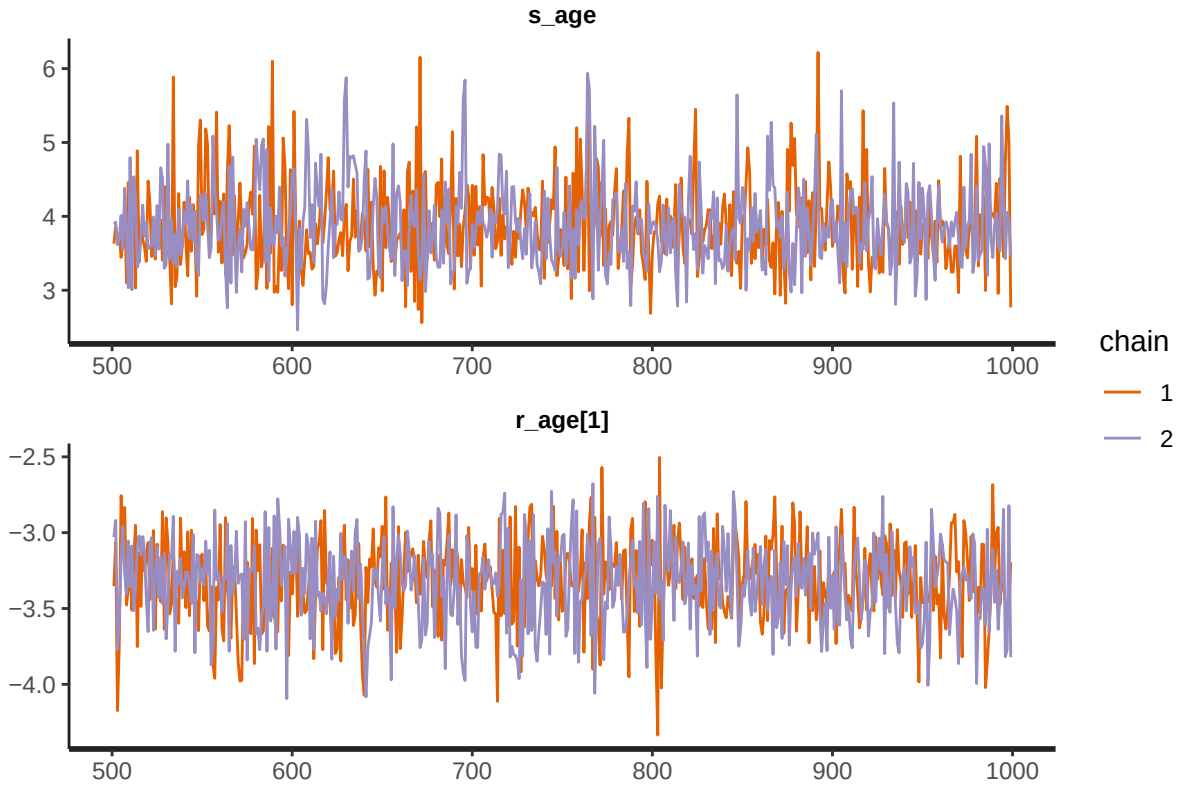
Diagnostic

Traceplot

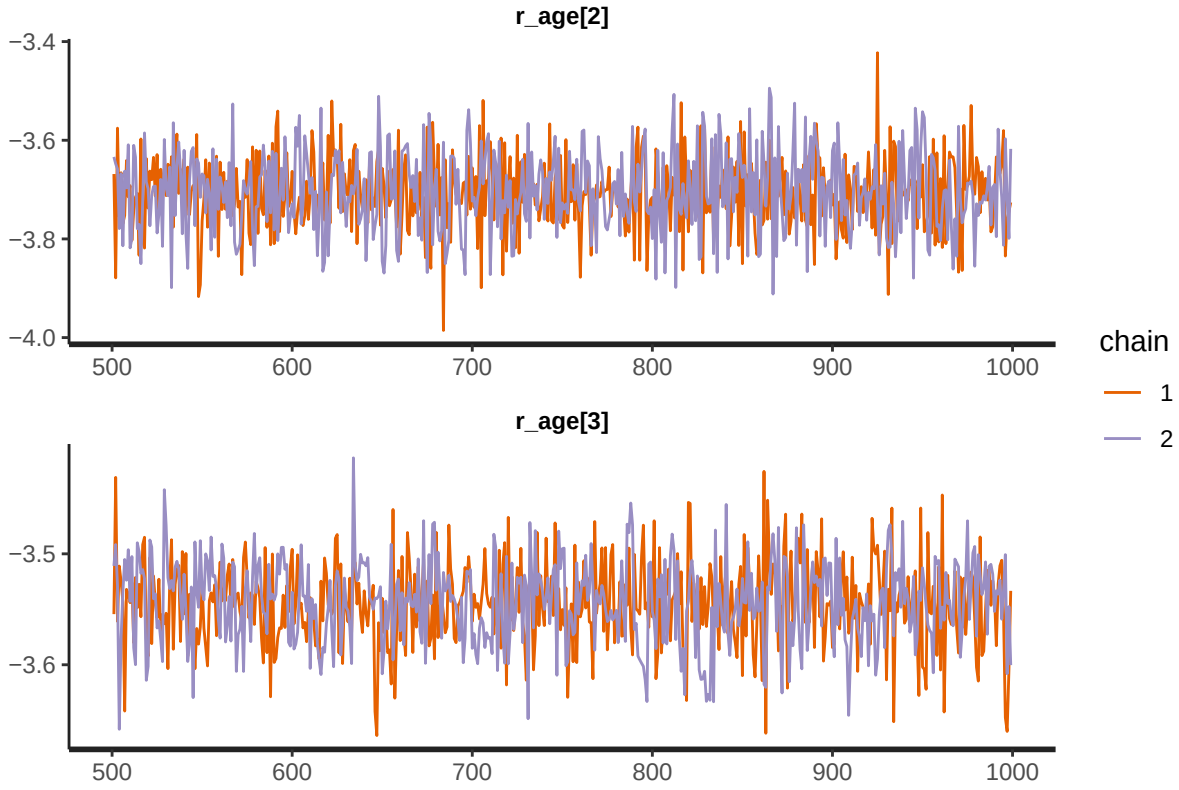
Trace plots are a time series of sampler iterations. The trace plots show the value of a variable across the monitored iteractions of the MCMC chain.

In particular I print both the chain without warmup (burn-in), the first one is the red one, while the second one is blue.

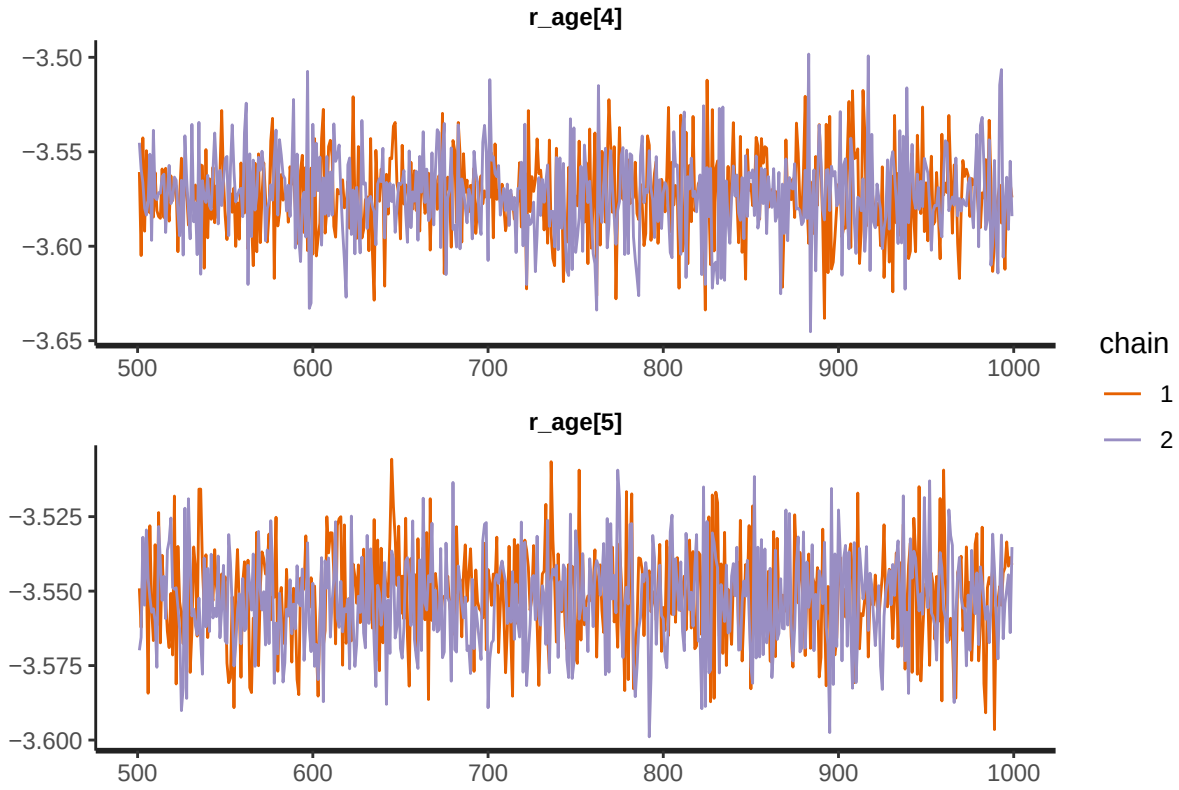
```
traceplot(fit1, pars = c(paras1[1],  paras1[2]),  inc_warmup = FALSE, nrow = 2)
```



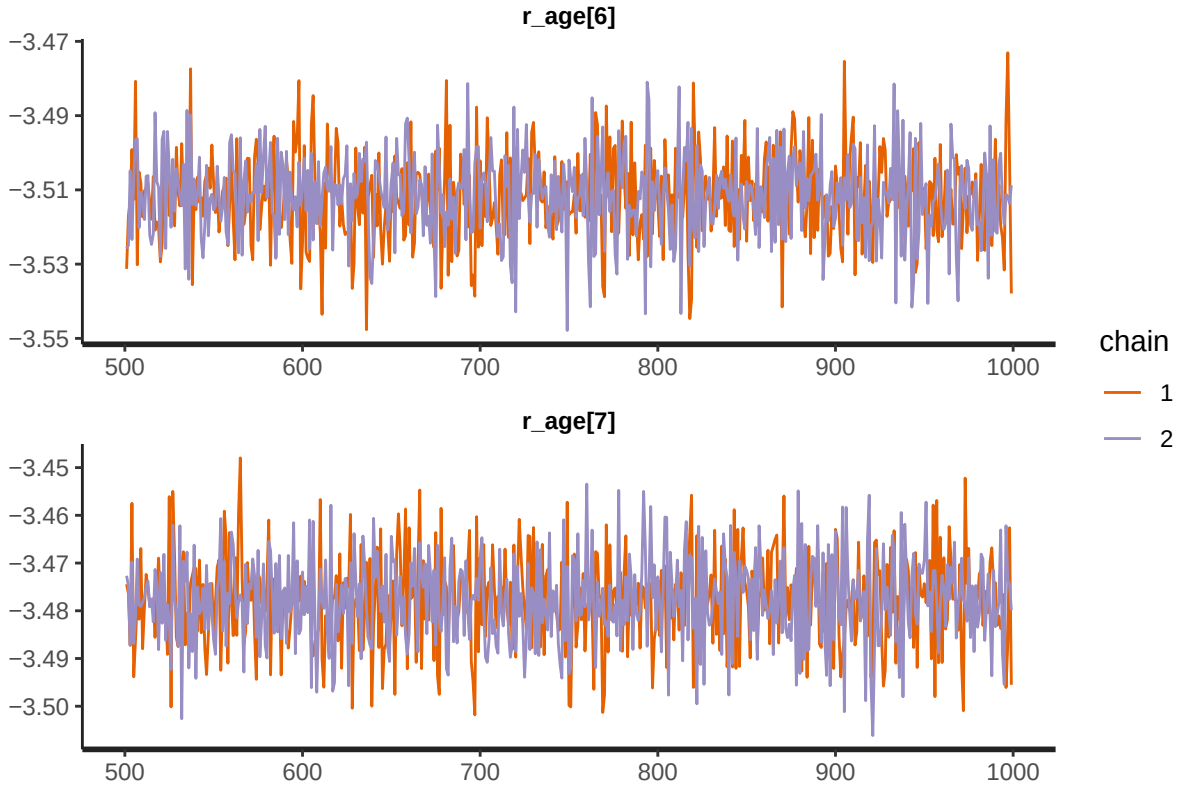

```
traceplot(fit1, pars = c(paras1[3],  paras1[4]),  inc_warmup = FALSE, nrow = 2)
```



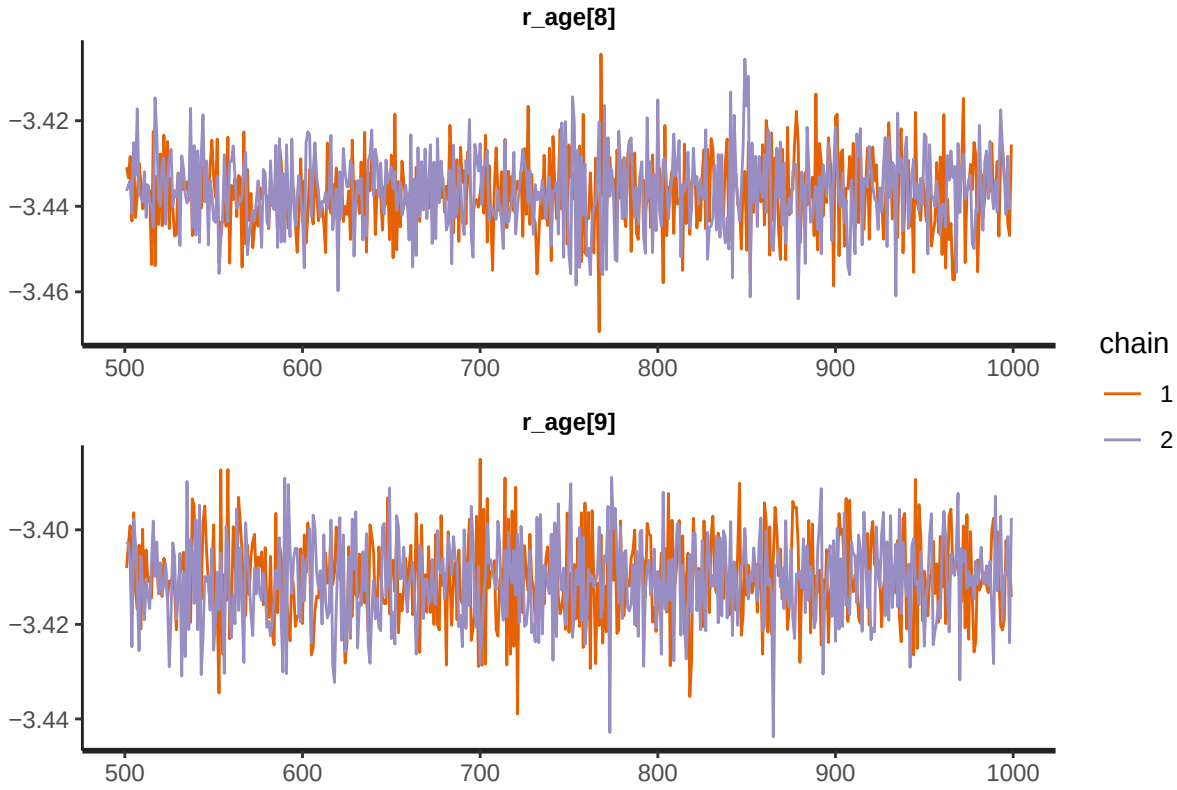
```
traceplot(fit1, pars = c(paras1[5],  paras1[6]),  inc_warmup = FALSE, nrow = 2)
```



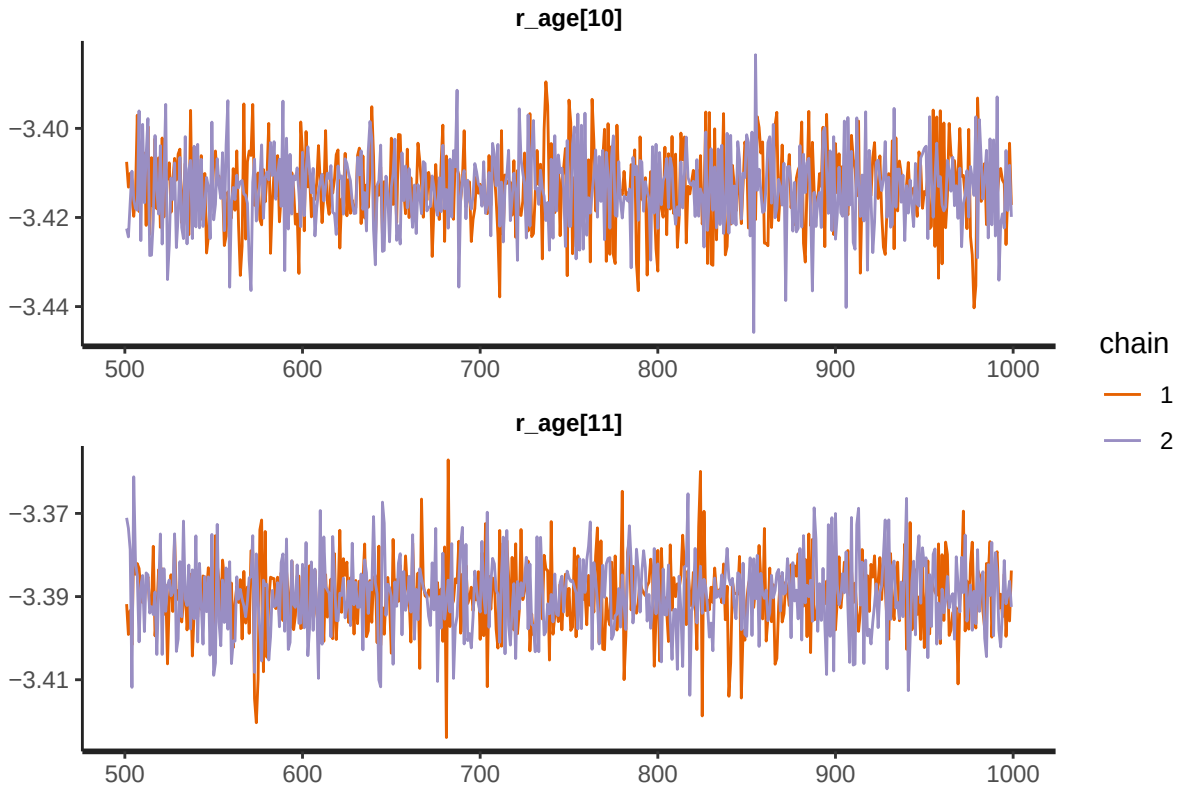

```
traceplot(fit1, pars = c(paras1[7],  paras1[8]),  inc_warmup = FALSE, nrow = 2)
```



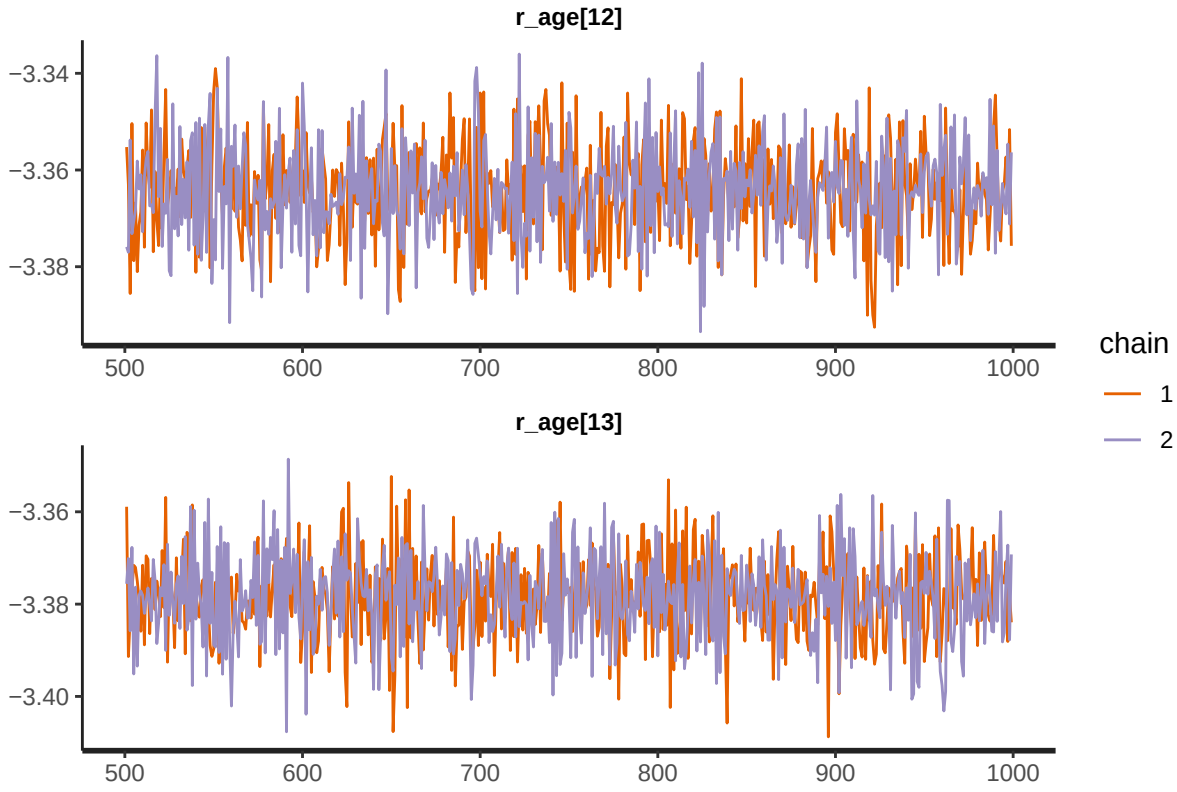
```
traceplot(fit1, pars = c(paras1[9],  paras1[10]), inc_warmup = FALSE, nrow = 2)
```



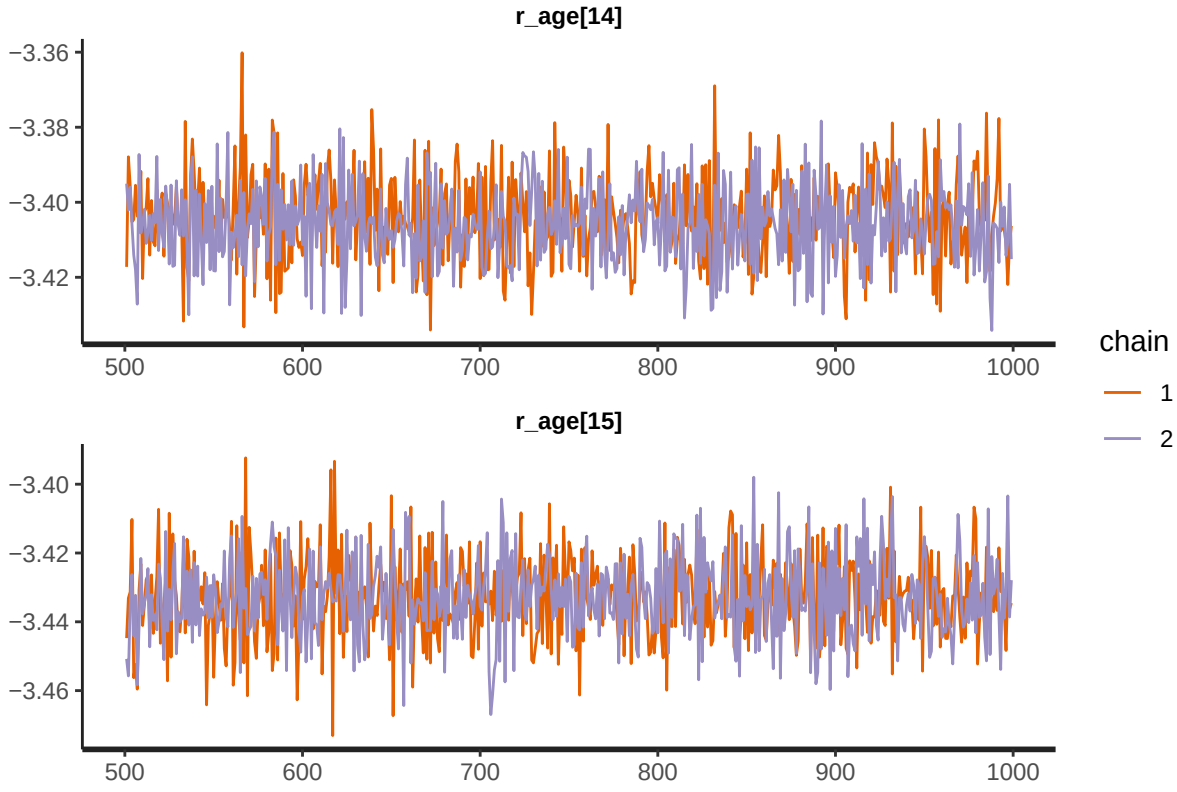

```
traceplot(fit1, pars = c(paras1[11], paras1[12]), inc_warmup = FALSE, nrow = 2)
```



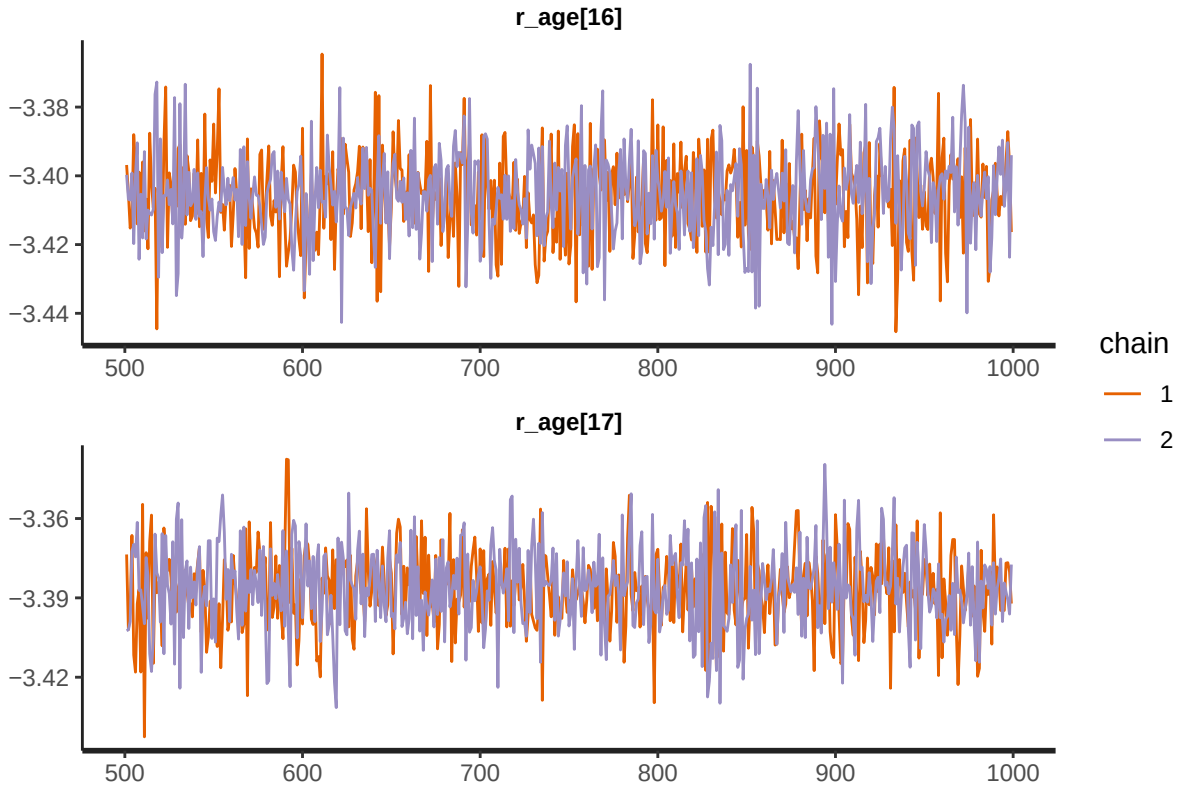
```
traceplot(fit1, pars = c(paras1[13], paras1[14]), inc_warmup = FALSE, nrow = 2)
```



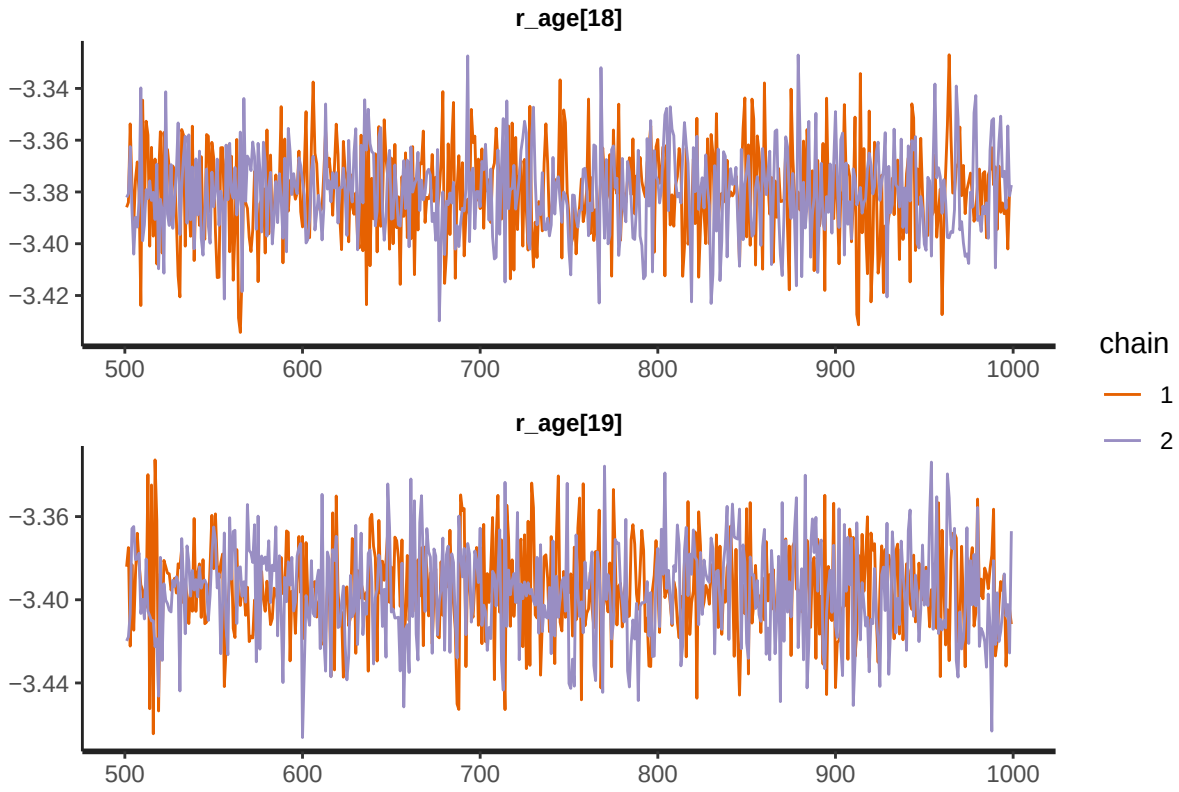

```
traceplot(fit1, pars = c(paras1[15], paras1[16]), inc_warmup = FALSE, nrow = 2)
```



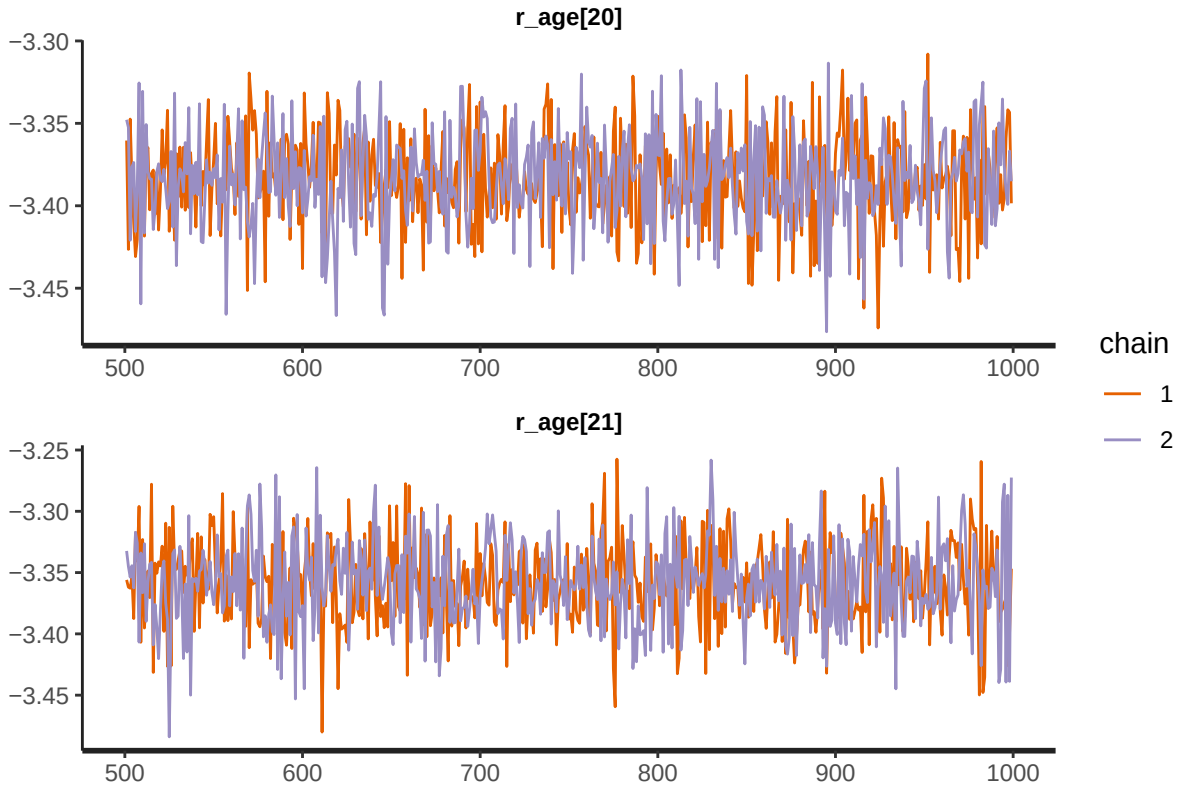
```
traceplot(fit1, pars = c(paras1[17], paras1[18]), inc_warmup = FALSE, nrow = 2)
```



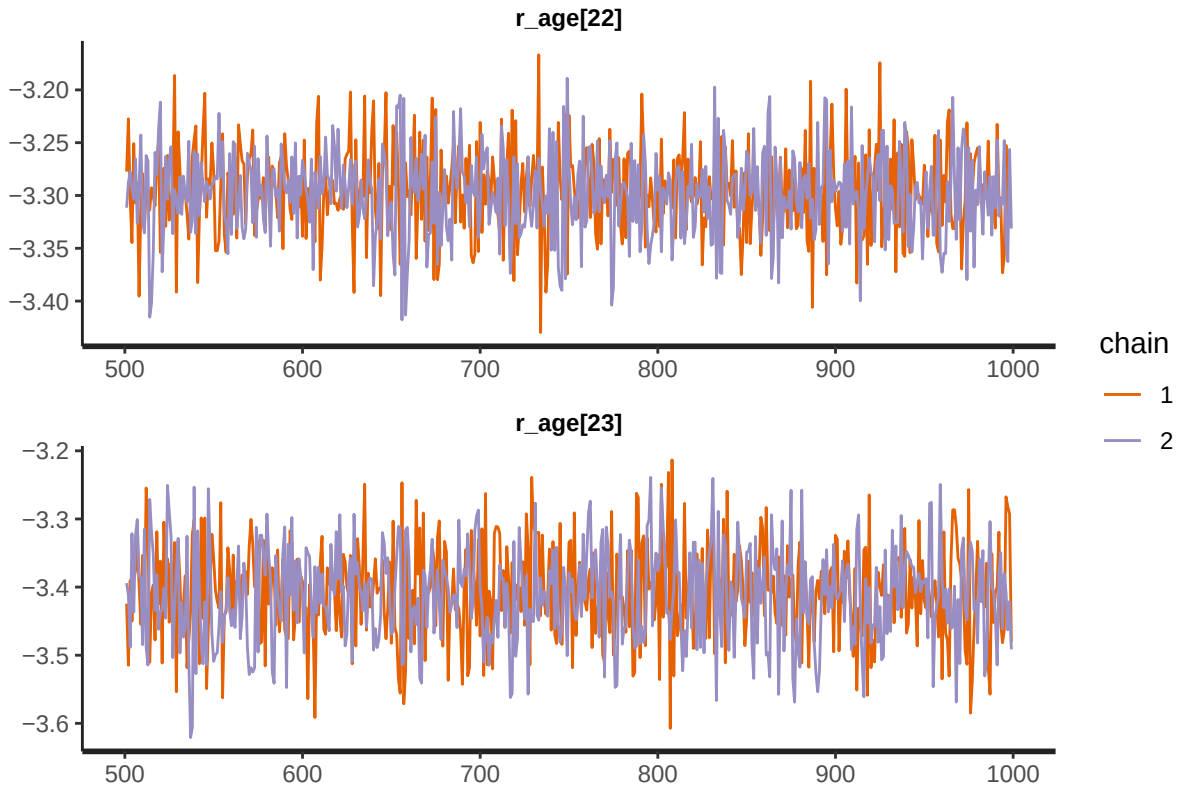

```
traceplot(fit1, pars = c(paras1[19], paras1[20]), inc_warmup = FALSE, nrow = 2)
```



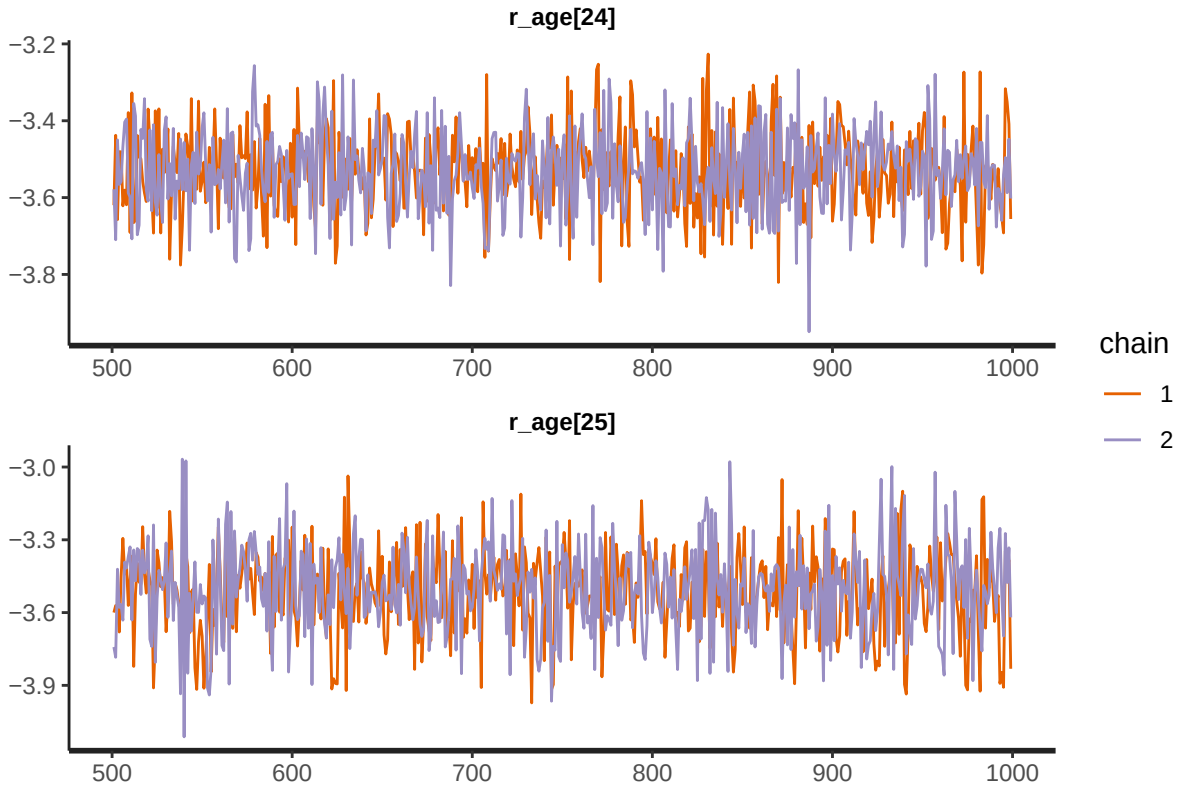
```
traceplot(fit1, pars = c(paras1[21], paras1[22]), inc_warmup = FALSE, nrow = 2)
```



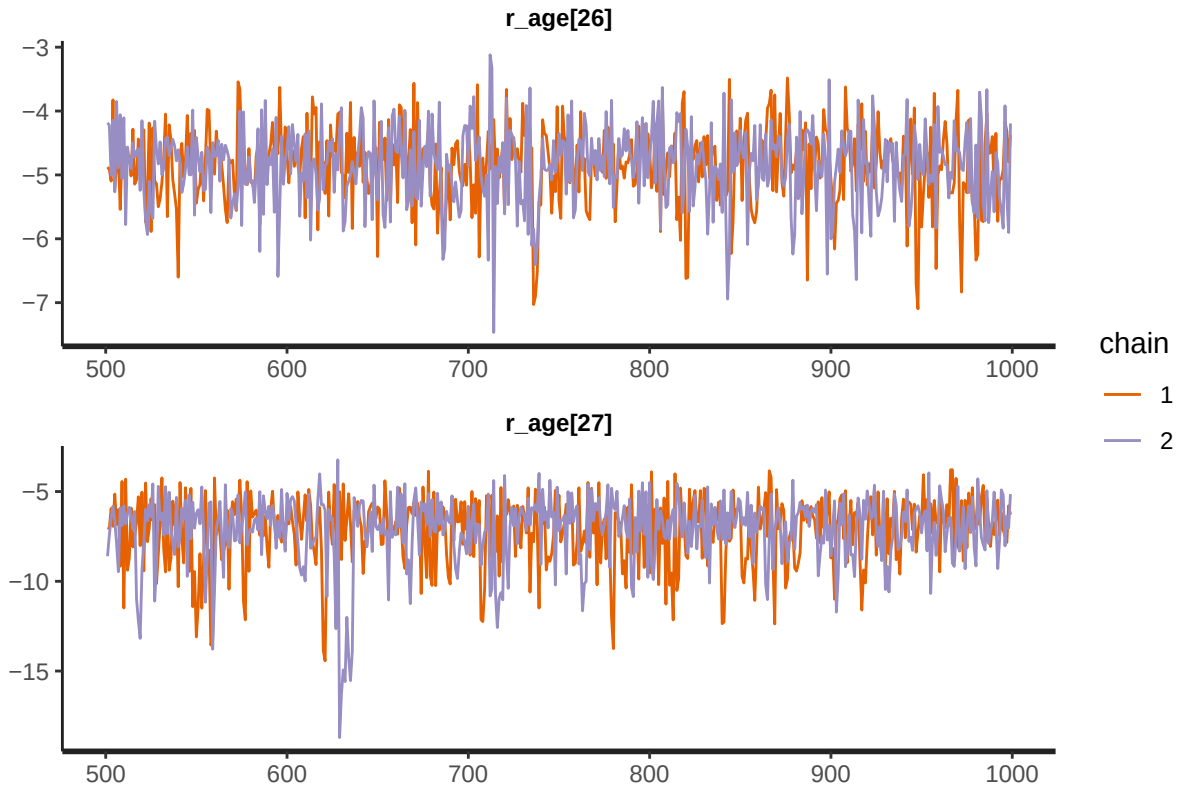

```
traceplot(fit1, pars = c(paras1[23], paras1[24]), inc_warmup = FALSE, nrow = 2)
```



```
traceplot(fit1, pars = c(paras1[25], paras1[26]), inc_warmup = FALSE, nrow = 2)
```




```
traceplot(fit1, pars = c(paras1[27], paras1[28]), inc_warmup = FALSE, nrow = 2)
```



Autocorrelation

MCMC samples are dependent. This does not effect the validity of inference on the posterior if the samplers has time to explore the posterior distribution, but it does affect the efficiency of the sampler.

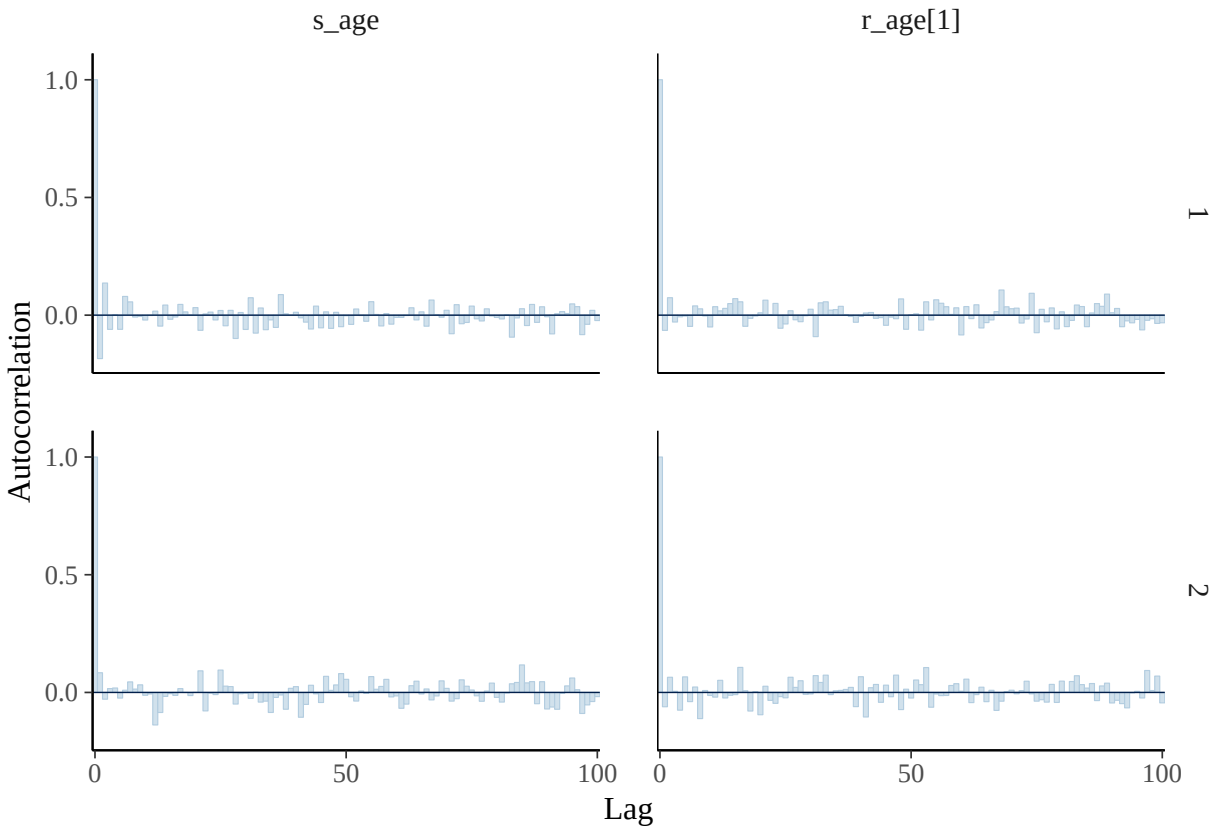
In other words, highly correlated MCMC samplers requires more samples to produce the same level of Monte Carlo error for an estimate.

If we have a sequence of random variables X_1, X_2, ... that are separated in time, as we did with the introduction to Markov chains, we can also think of the concept of autocorrelation, correlation of X_t with some past or future variable X_{t-l}. Formally,it is define as

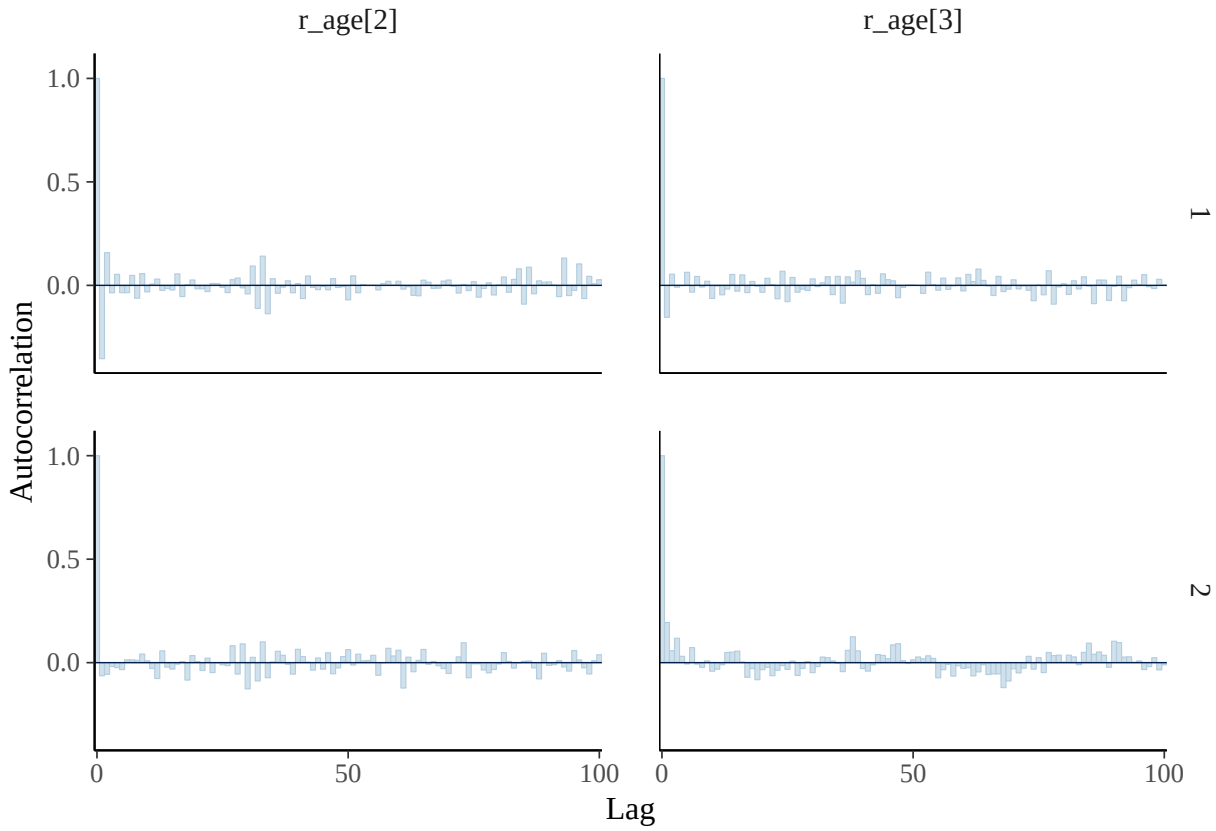
$$ACF(X_t, X_{t-l}) = \frac{\text{Cov}(X_t, X_{t-l})}{\sqrt{\text{Var}(X_t)\text{Var}(X_t)}}$$

If the sequence is stationary, so that the joint distribution of multiple Xs does not change with time shifts, then autocorrelation for two variables does not depend on the exact times t and $t - l$, but rather on the distance between them, l. That is why the autocorrelation plots in the lesson on convergence of MCMC calculate autocorrelation in terms of lags.

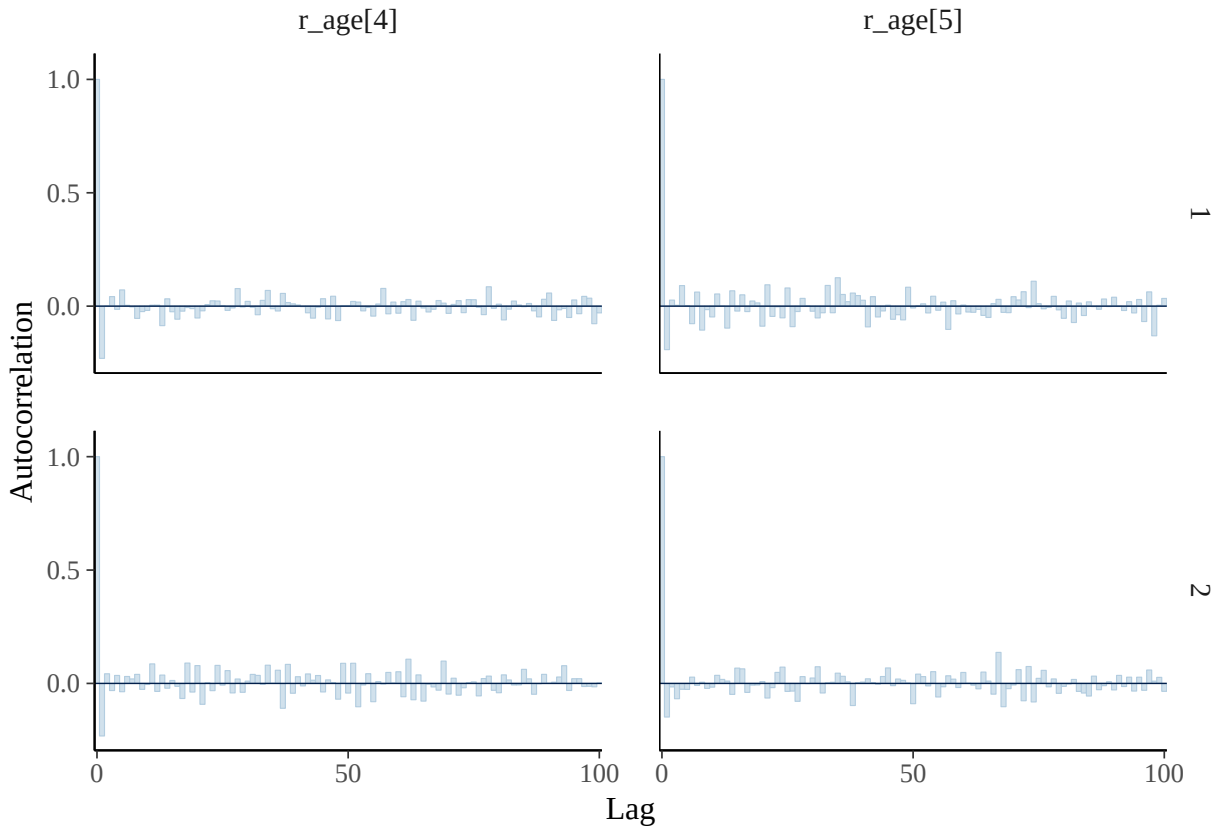
```
mcmc_acf_bar(fit1, pars = c(paras1[1],  paras1[2]),   lags = num.lag)
```



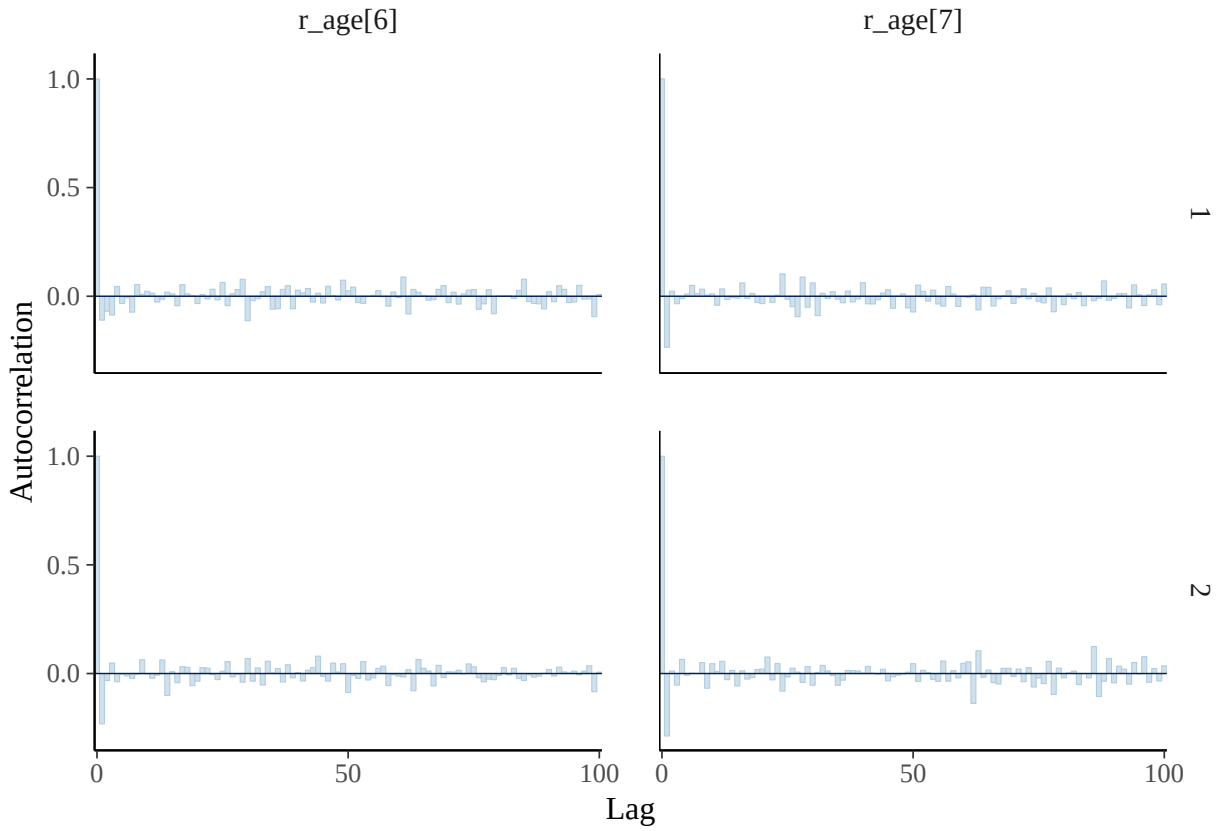

```
mcmc_acf_bar(fit1, pars = c(paras1[3],  paras1[4]),   lags = num.lag)
```



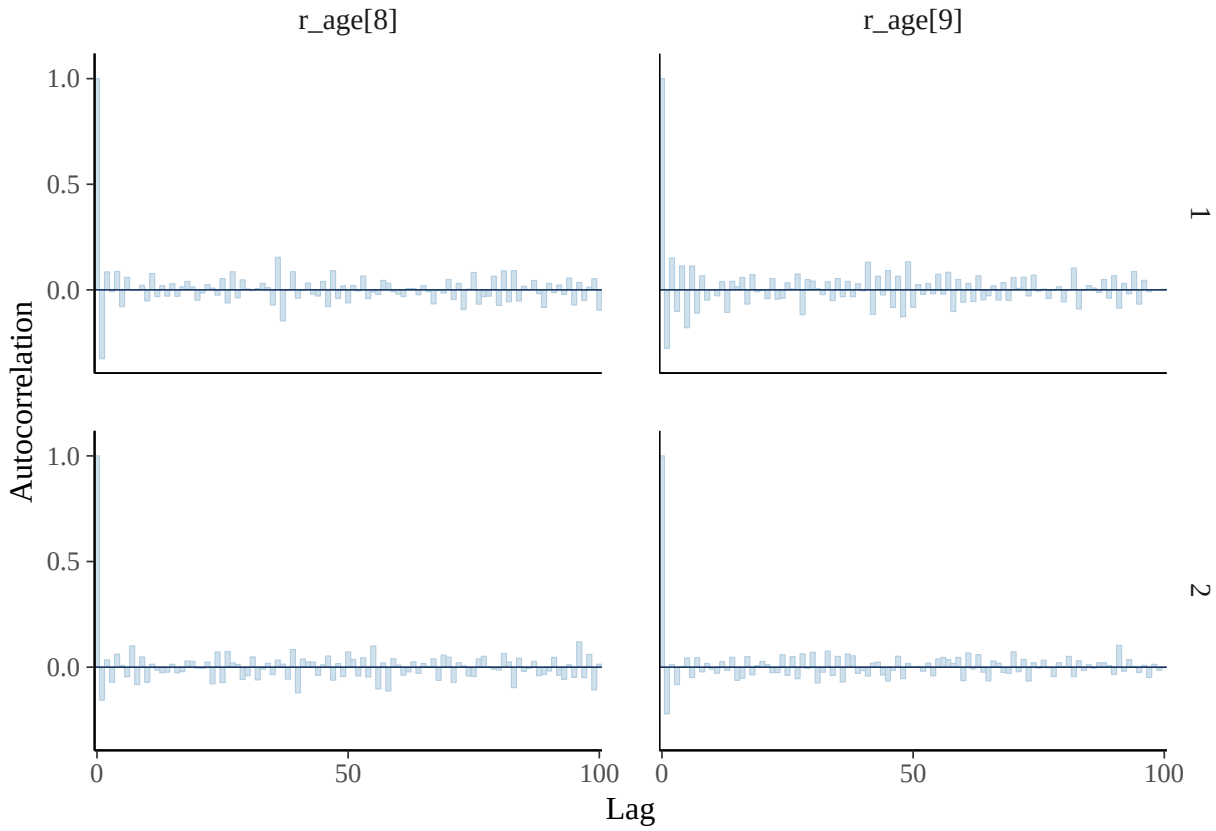

```
mcmc_acf_bar(fit1, pars = c(paras1[5],  paras1[6]),   lags = num.lag)
```



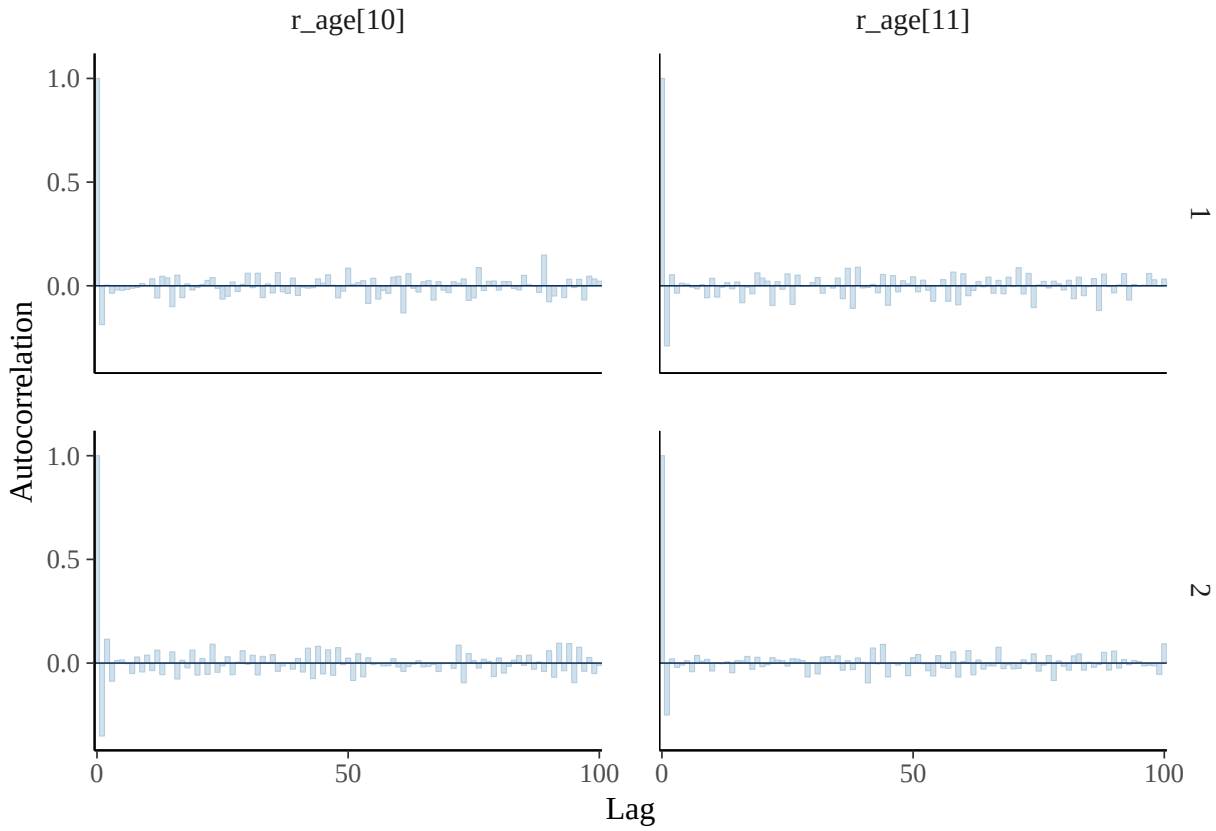

```
mcmc_acf_bar(fit1, pars = c(paras1[7],  paras1[8]),   lags = num.lag)
```



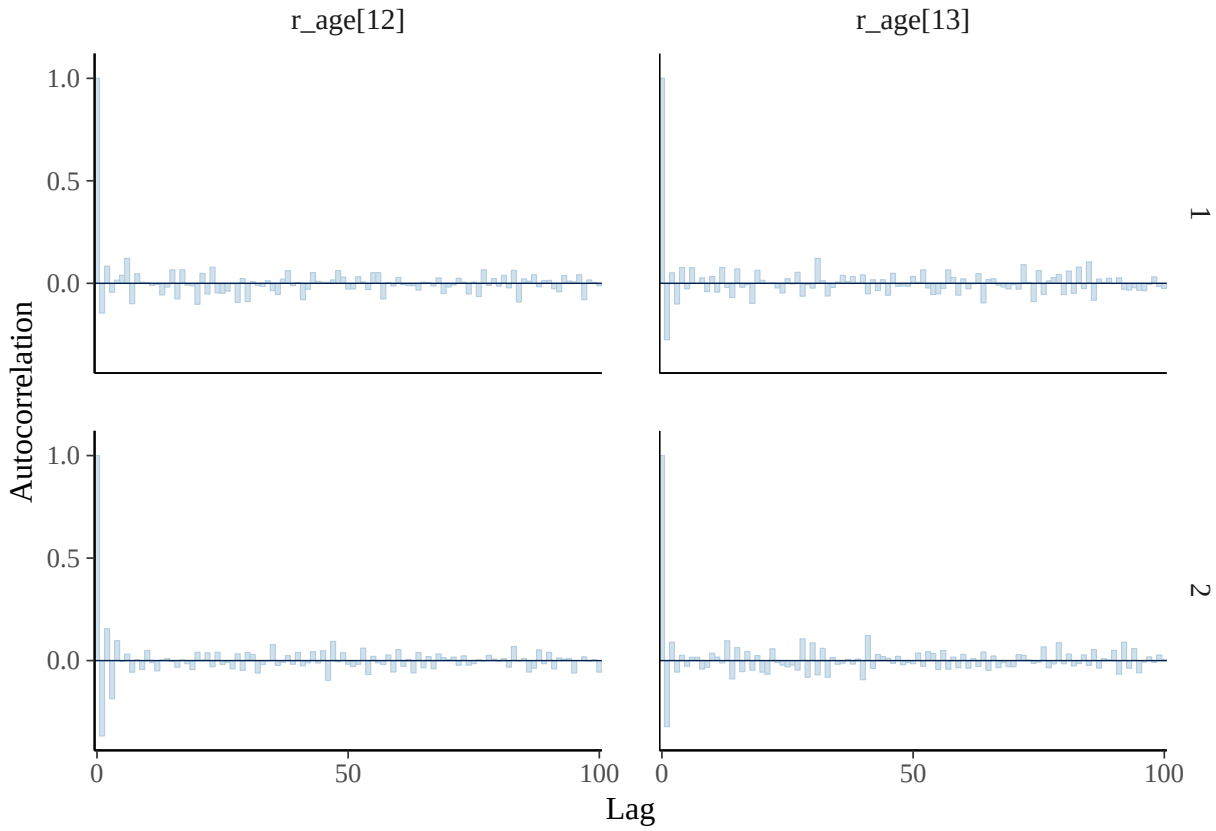

```
mcmc_acf_bar(fit1, pars = c(paras1[9],  paras1[10]),  lags = num.lag)
```



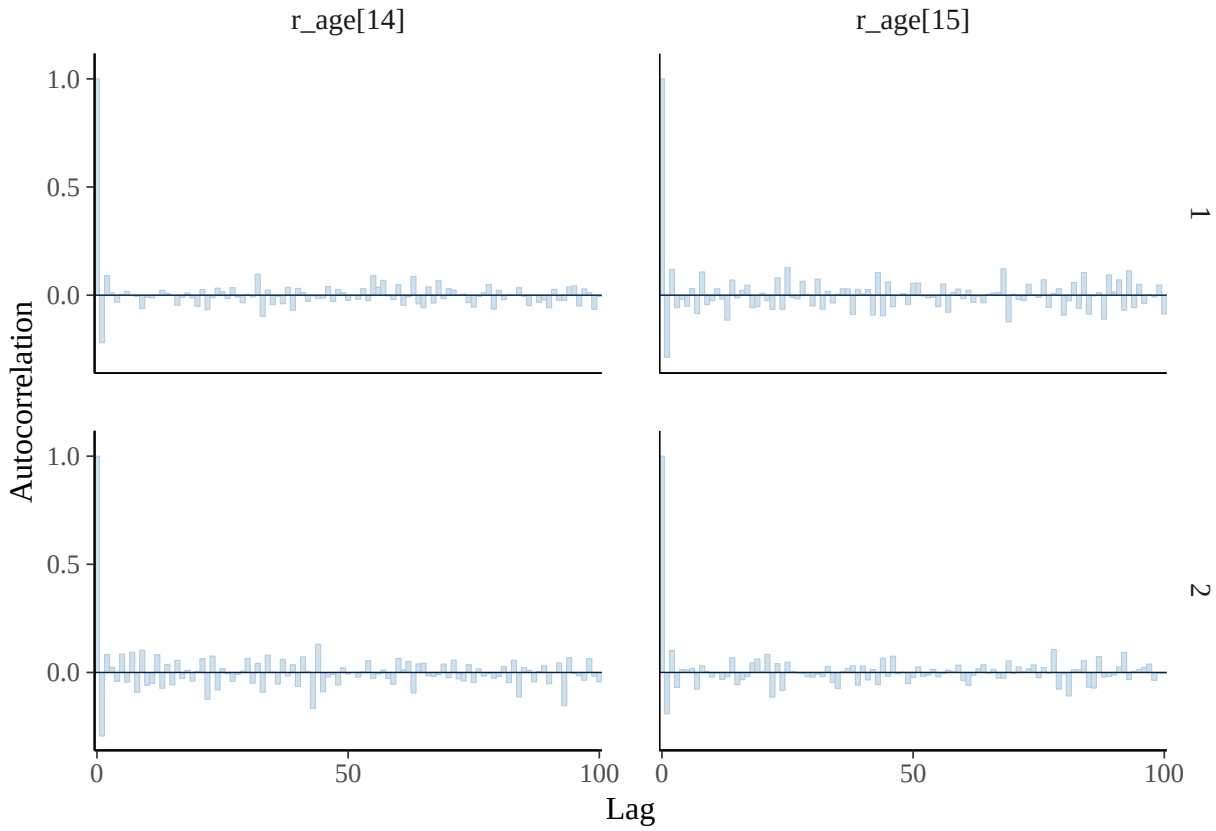

```
mcmc_acf_bar(fit1, pars = c(paras1[11], paras1[12]),  lags = num.lag)
```



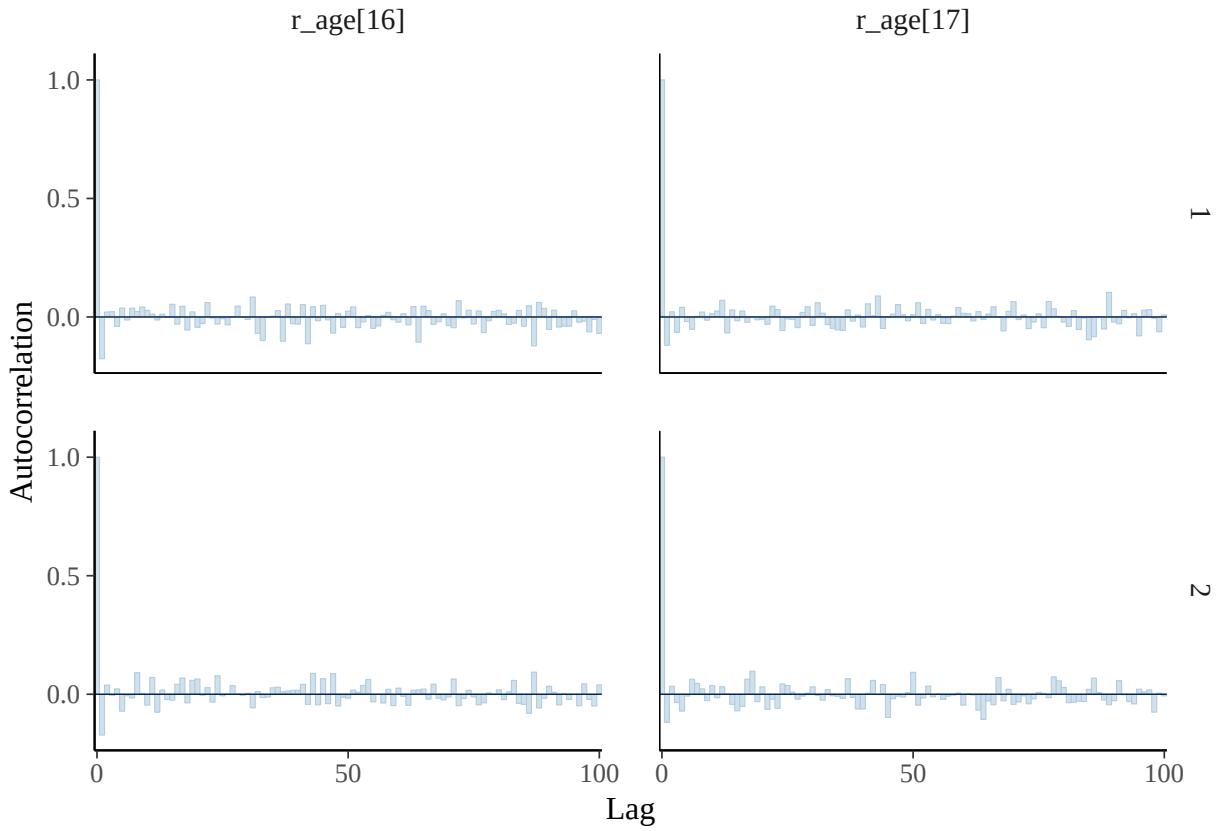

```
mcmc_acf_bar(fit1, pars = c(paras1[13], paras1[14]),  lags = num.lag)
```



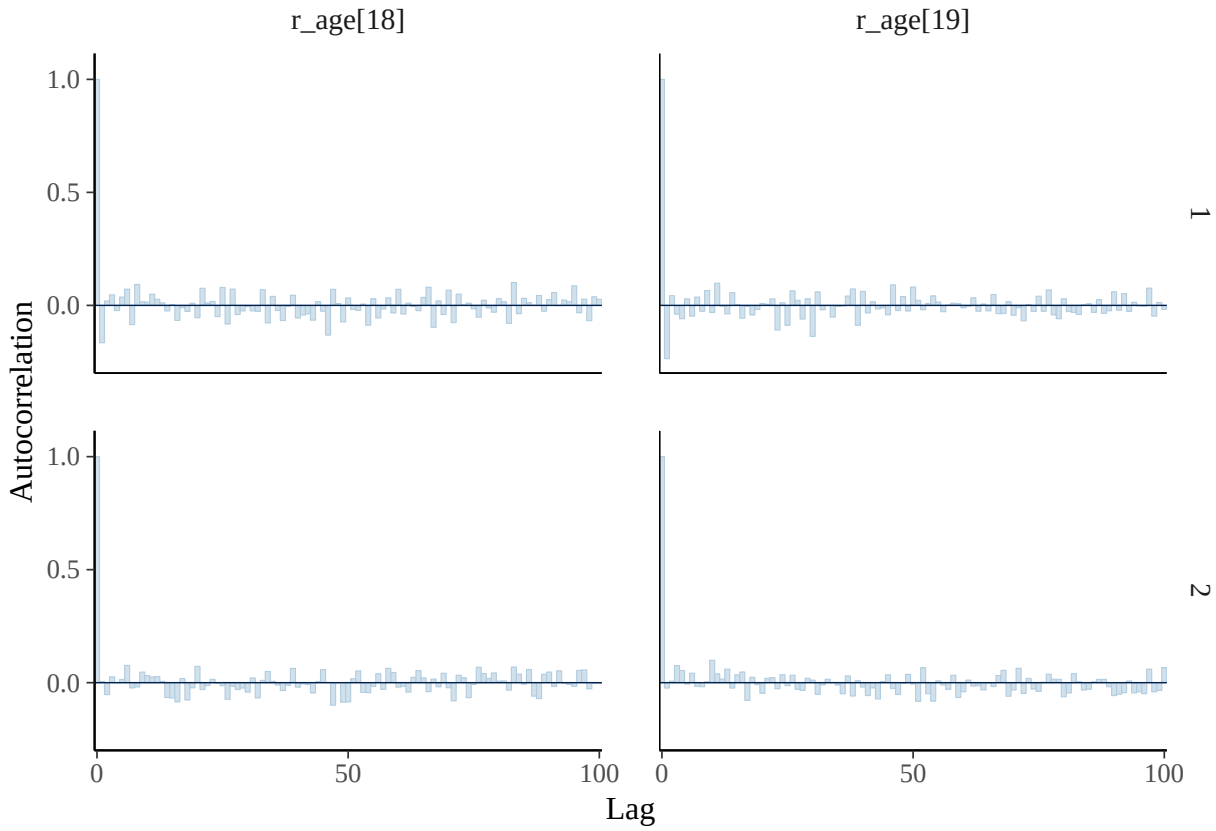

```
mcmc_acf_bar(fit1, pars = c(paras1[15], paras1[16]),  lags = num.lag)
```



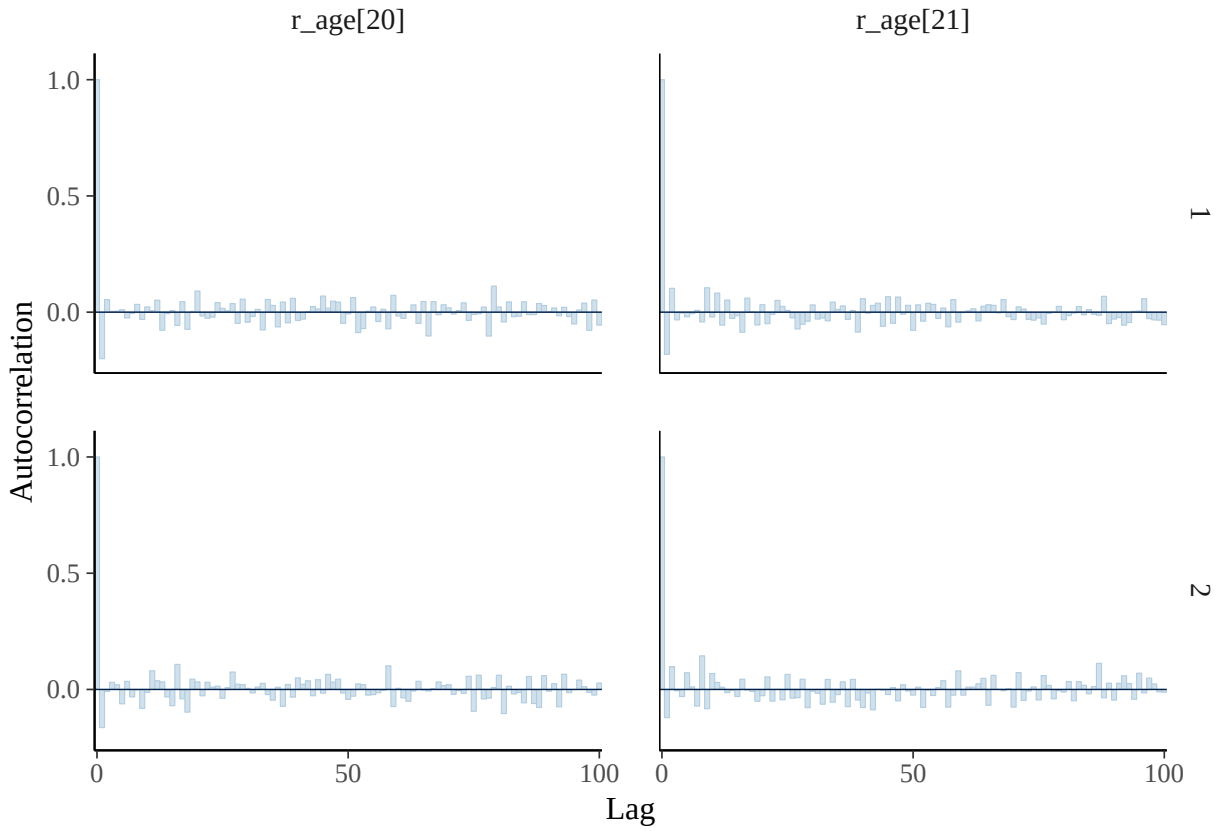

```
mcmc_acf_bar(fit1, pars = c(paras1[17], paras1[18]),  lags = num.lag)
```



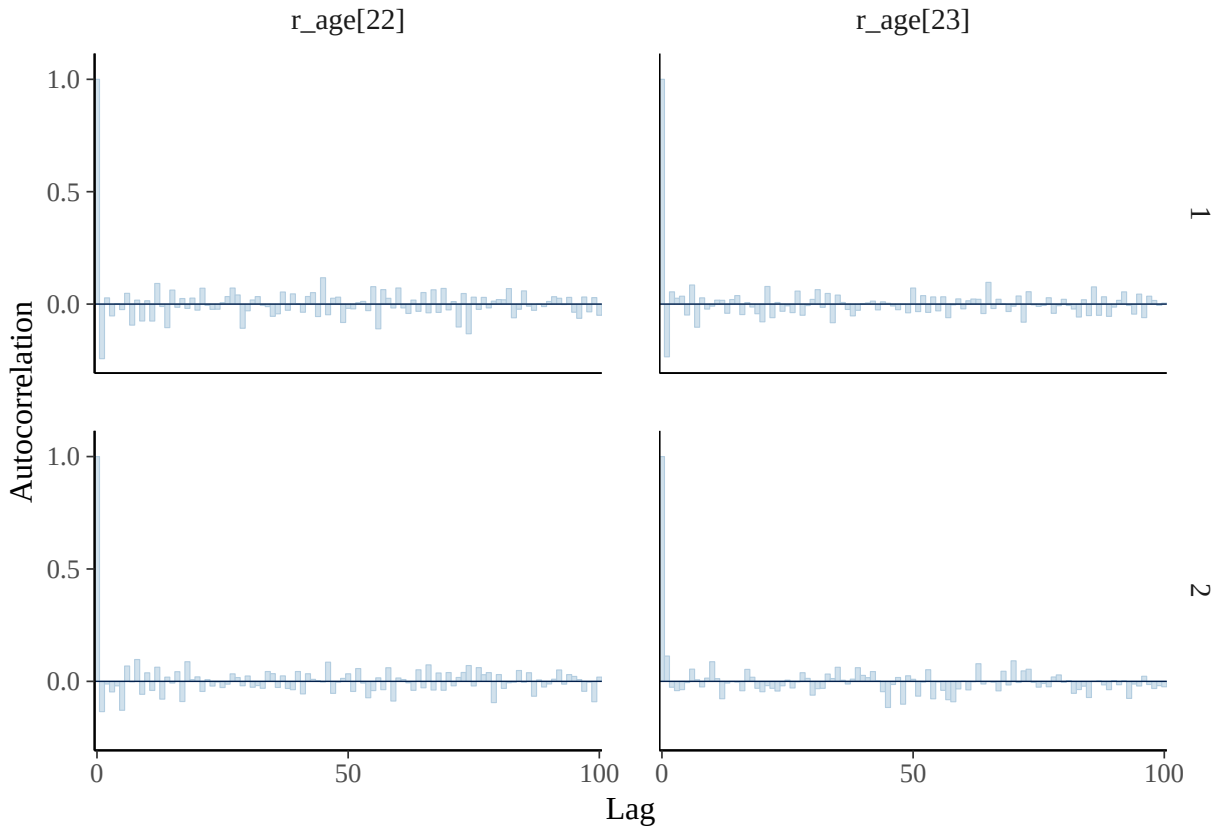

```
mcmc_acf_bar(fit1, pars = c(paras1[19], paras1[20]),  lags = num.lag)
```



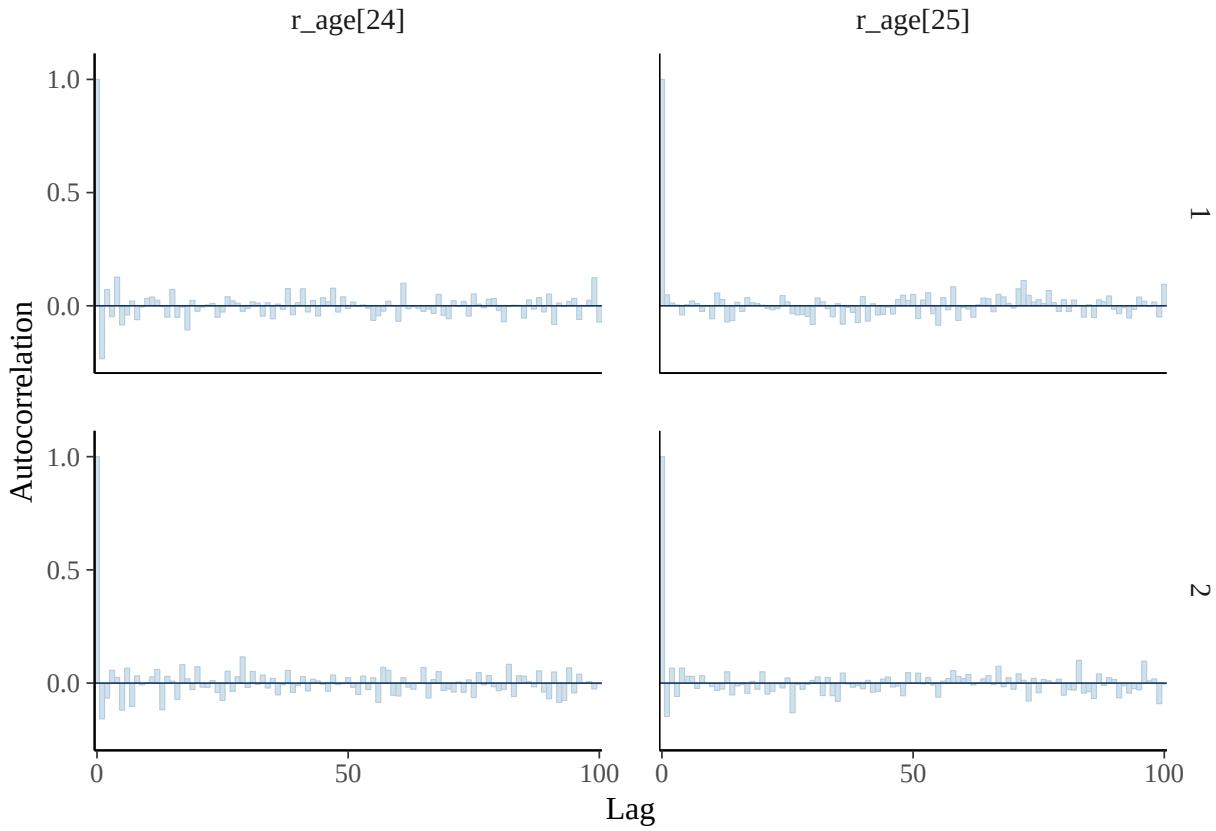

```
mcmc_acf_bar(fit1, pars = c(paras1[21], paras1[22]),  lags = num.lag)
```



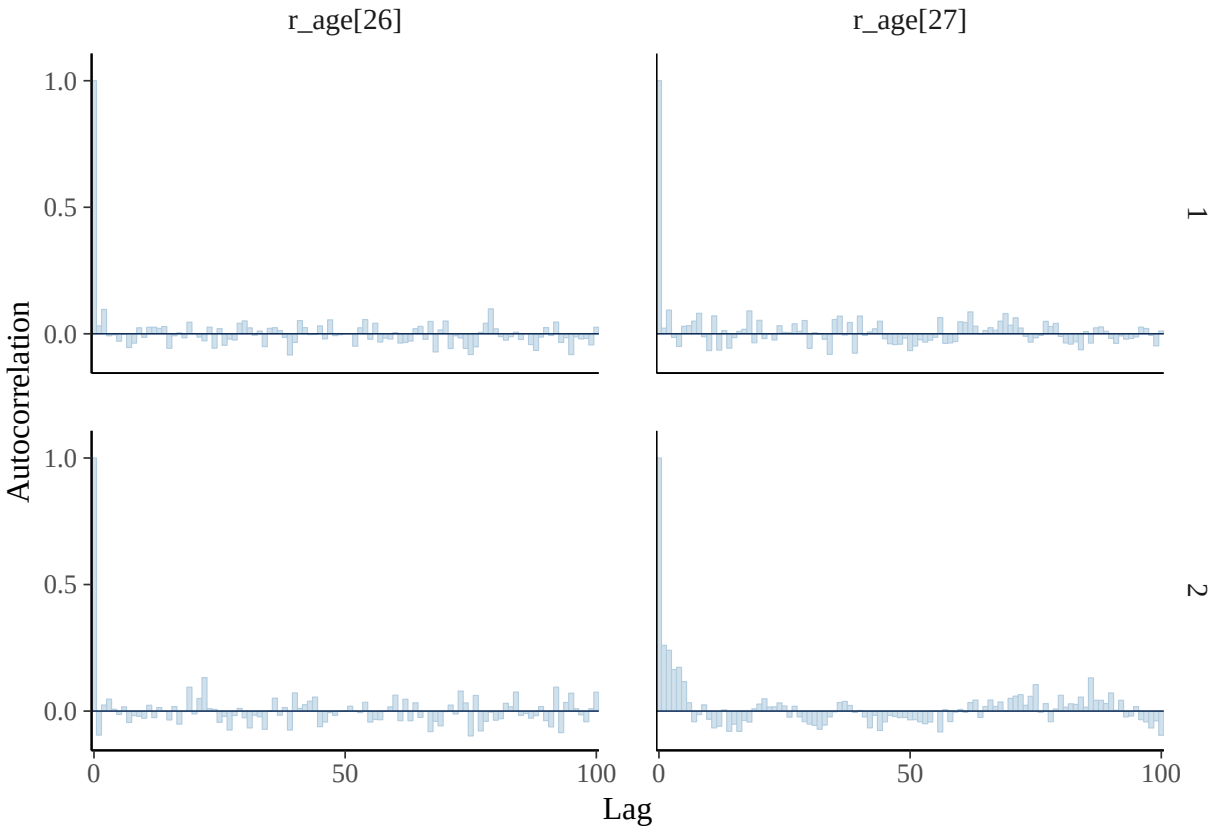

```
mcmc_acf_bar(fit1, pars = c(paras1[23], paras1[24]),  lags = num.lag)
```




```
mcmc_acf_bar(fit1, pars = c(paras1[25], paras1[26]),  lags = num.lag)
```



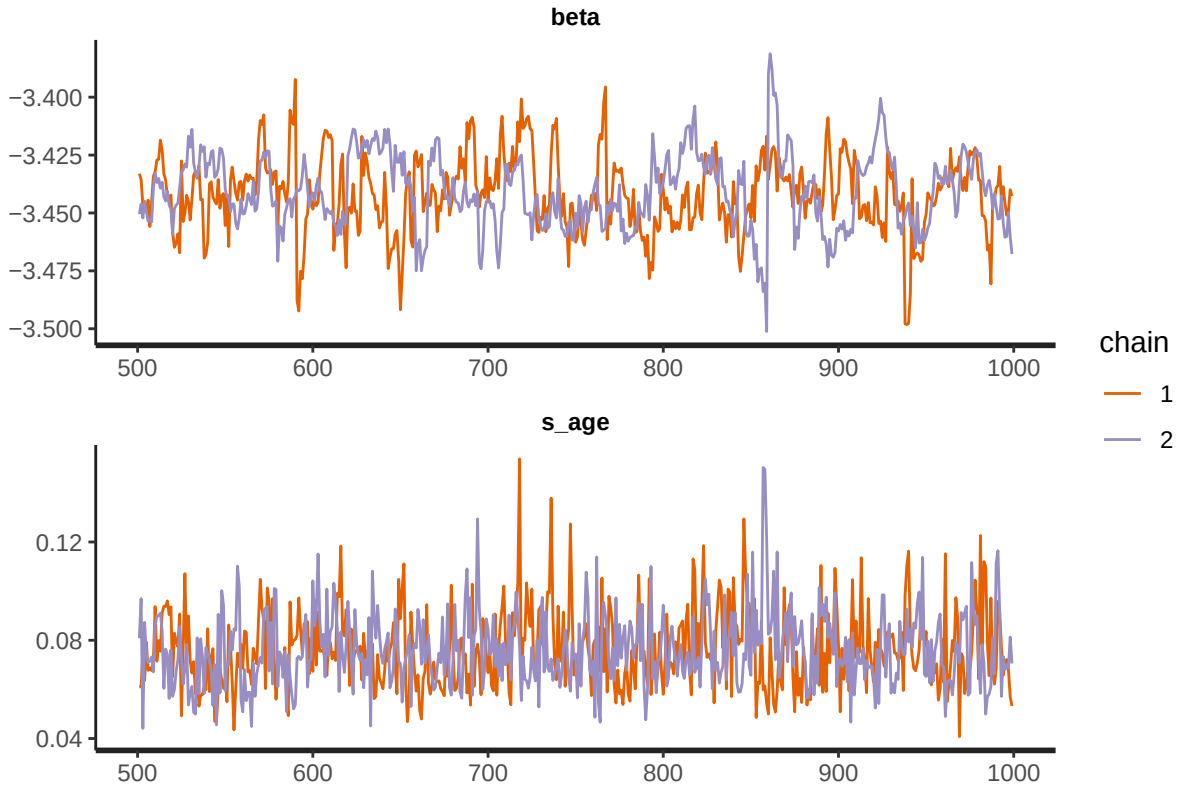

```
mcmc_acf_bar(fit1, pars = c(paras1[27], paras1[28]),  lags = num.lag)
```



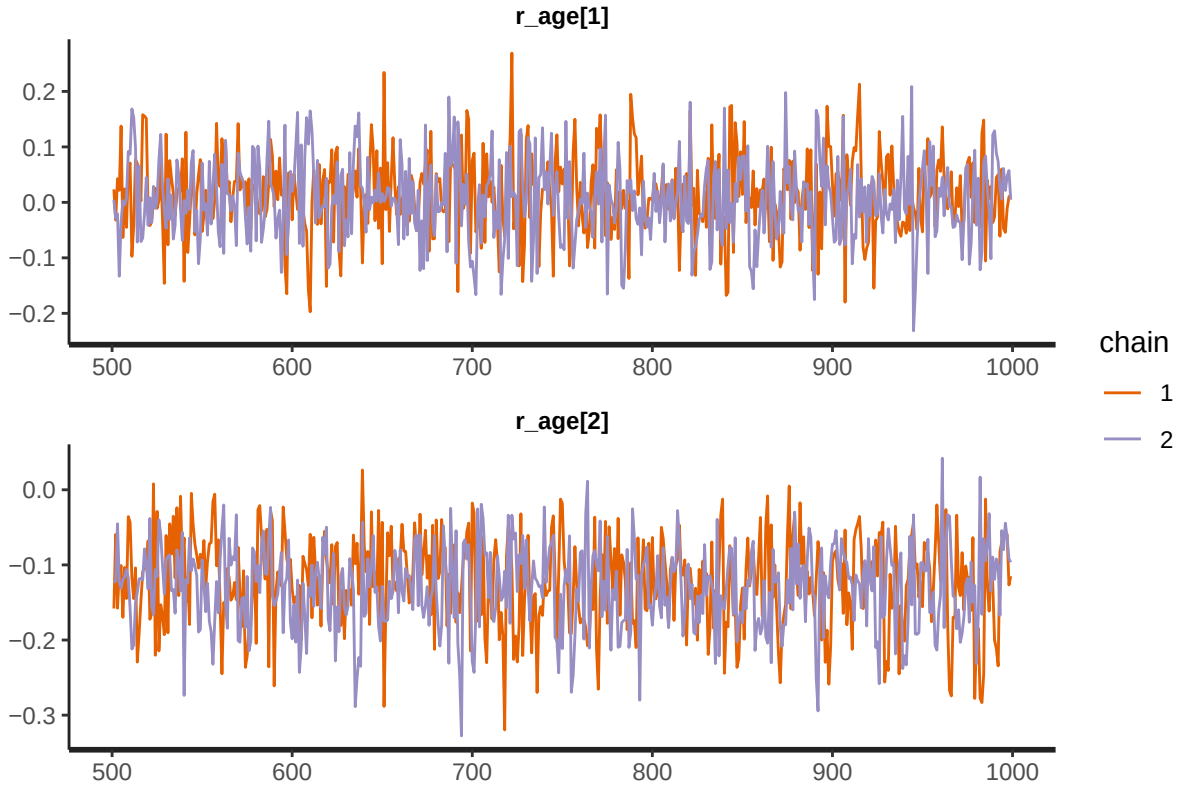
When the lag is small, the autocorrelation is relatively higher than those of bigger lags.

Traceplot (model2)

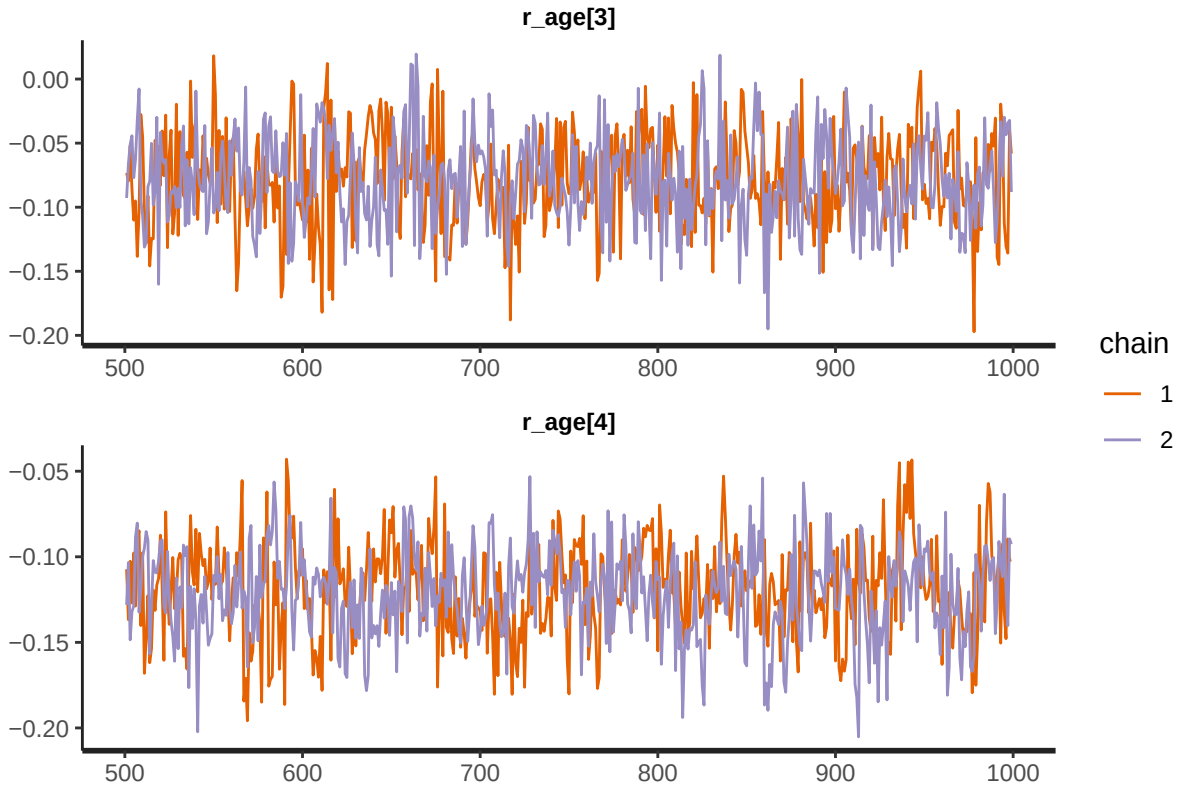
```
traceplot(fit2, pars = c(paras[1],  paras[2]),  inc_warmup = FALSE, nrow = 2)
```



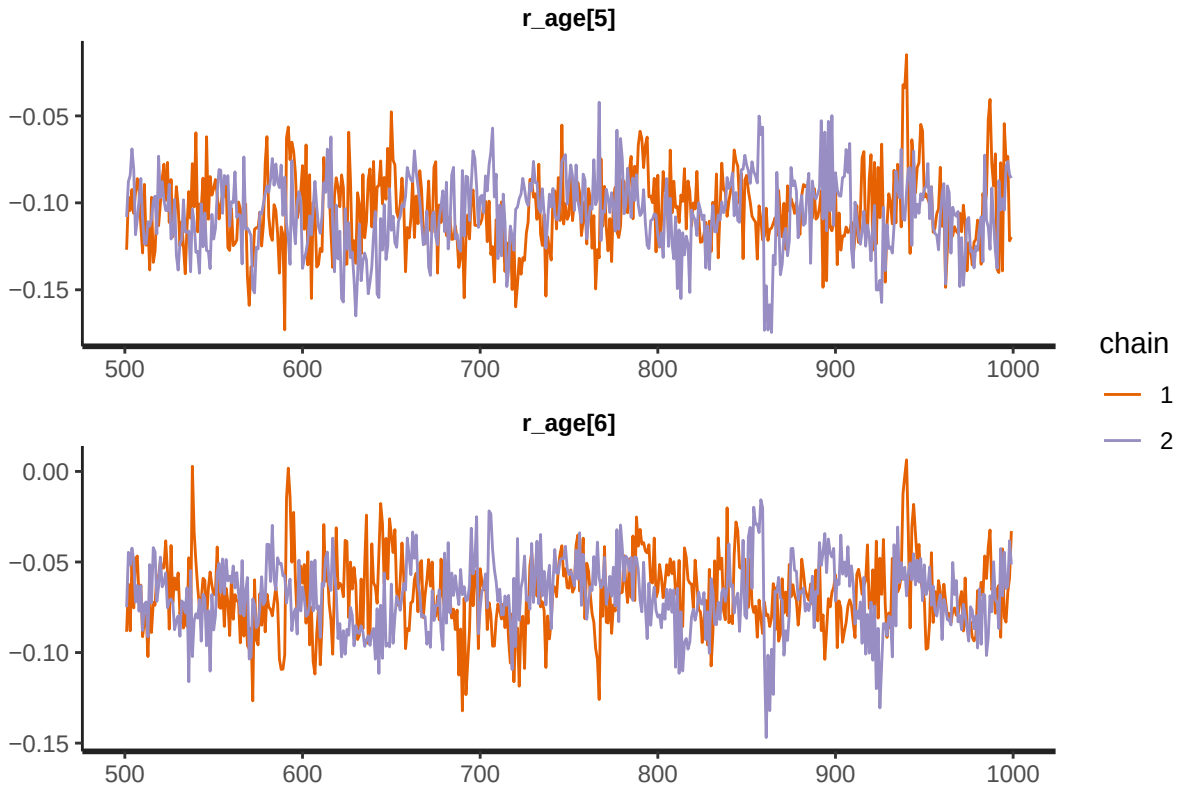

```
traceplot(fit2, pars = c(paras[3],  paras[4]),  inc_warmup = FALSE, nrow = 2)
```



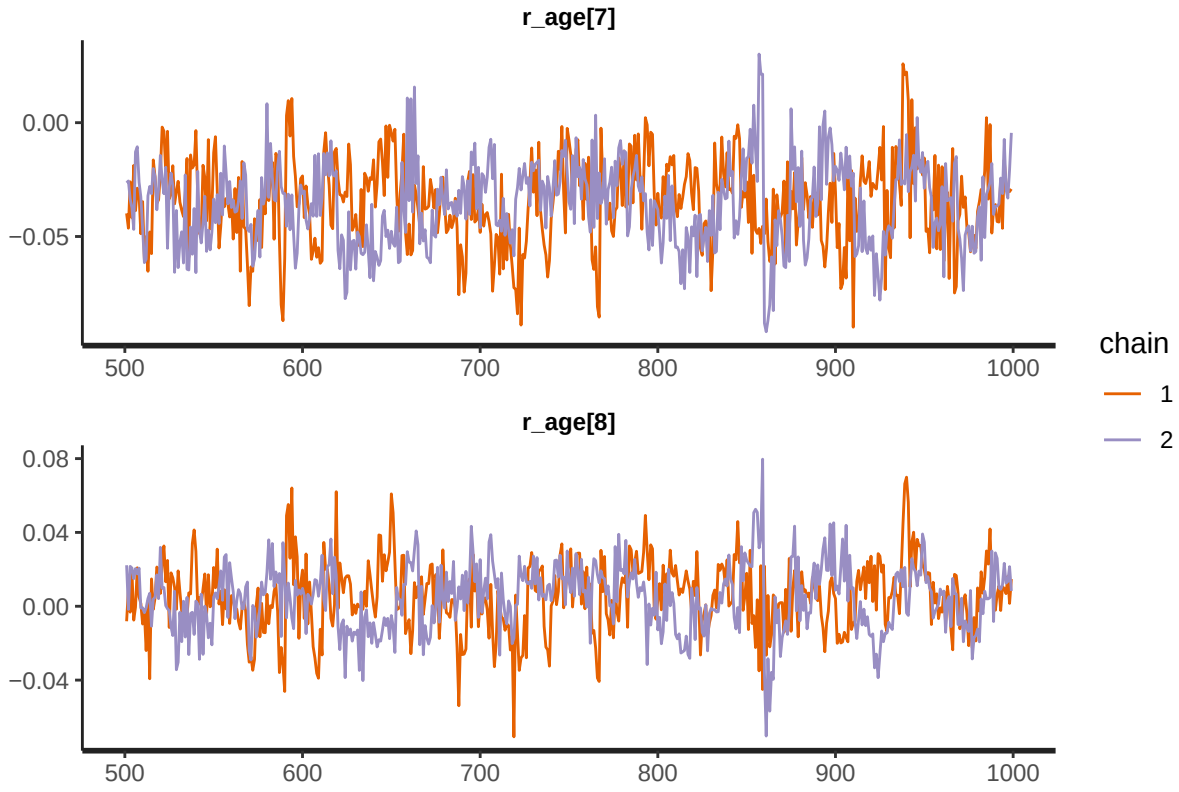
```
traceplot(fit2, pars = c(paras[5],  paras[6]),  inc_warmup = FALSE, nrow = 2)
```



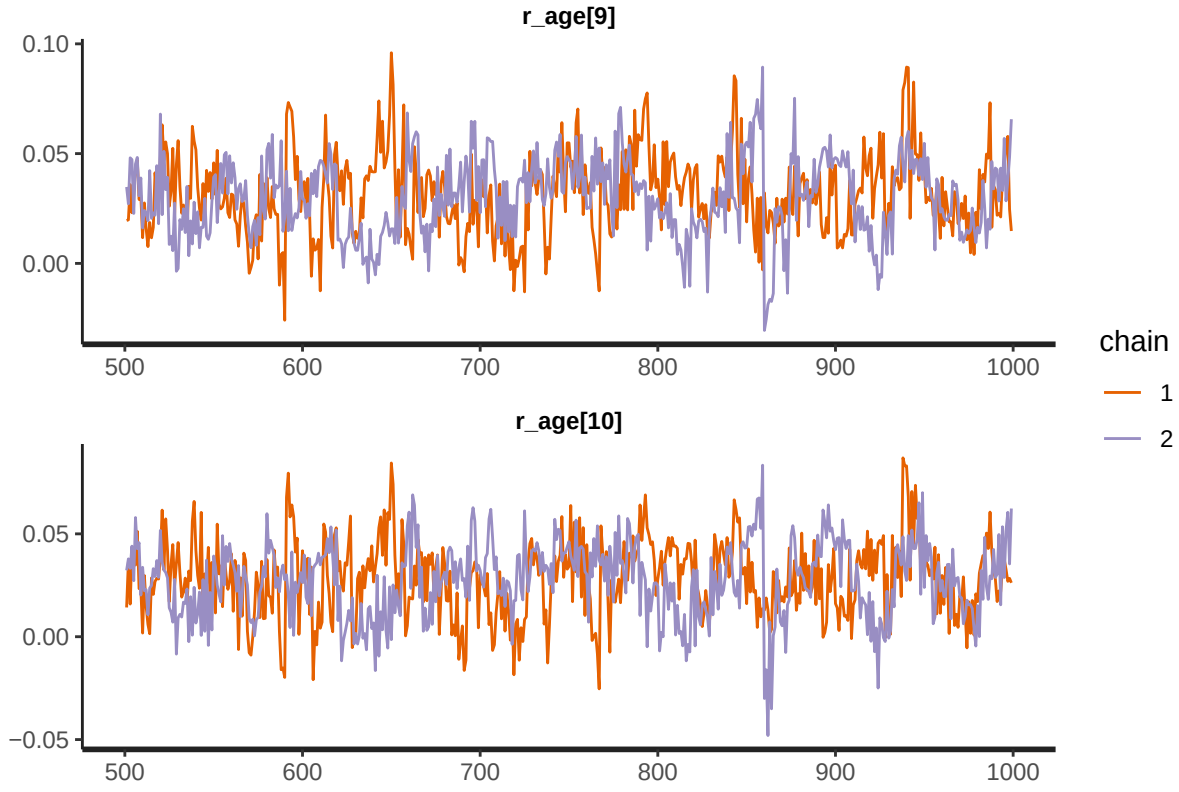

```
traceplot(fit2, pars = c(paras[7],  paras[8]),  inc_warmup = FALSE, nrow = 2)
```



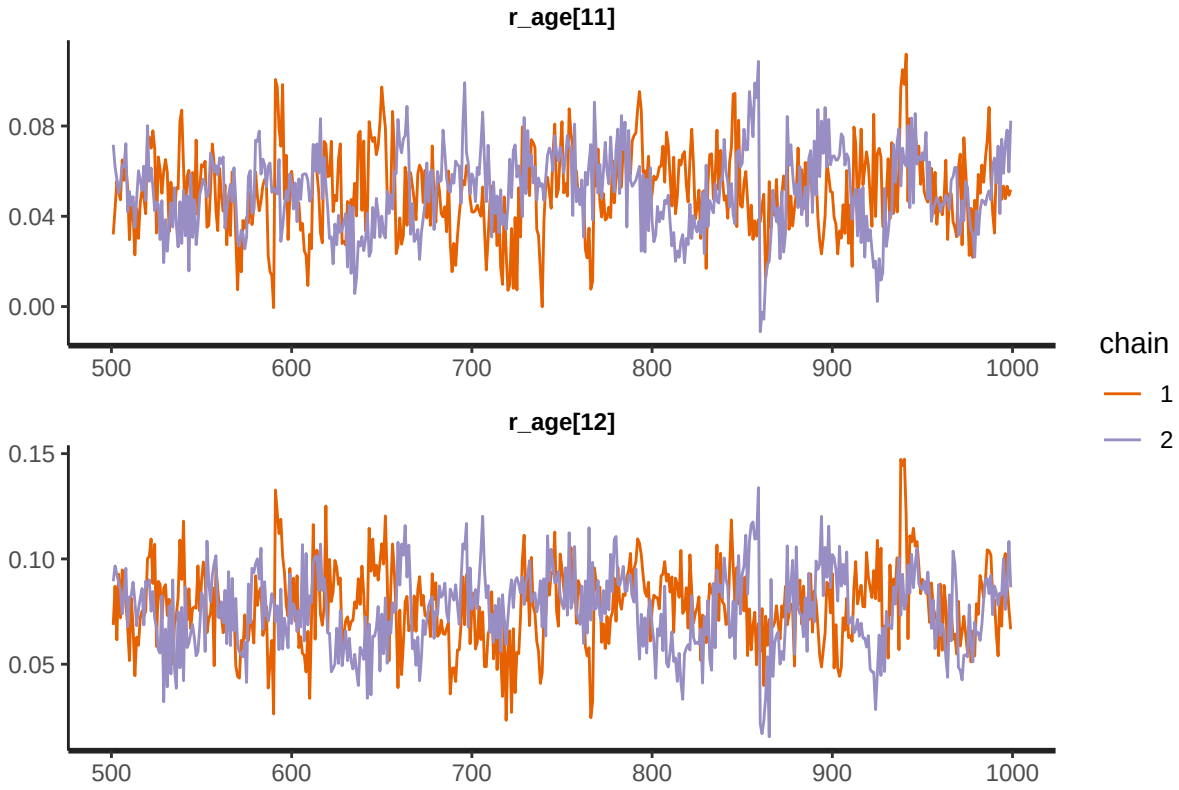
```
traceplot(fit2, pars = c(paras[9],  paras[10]), inc_warmup = FALSE, nrow = 2)
```



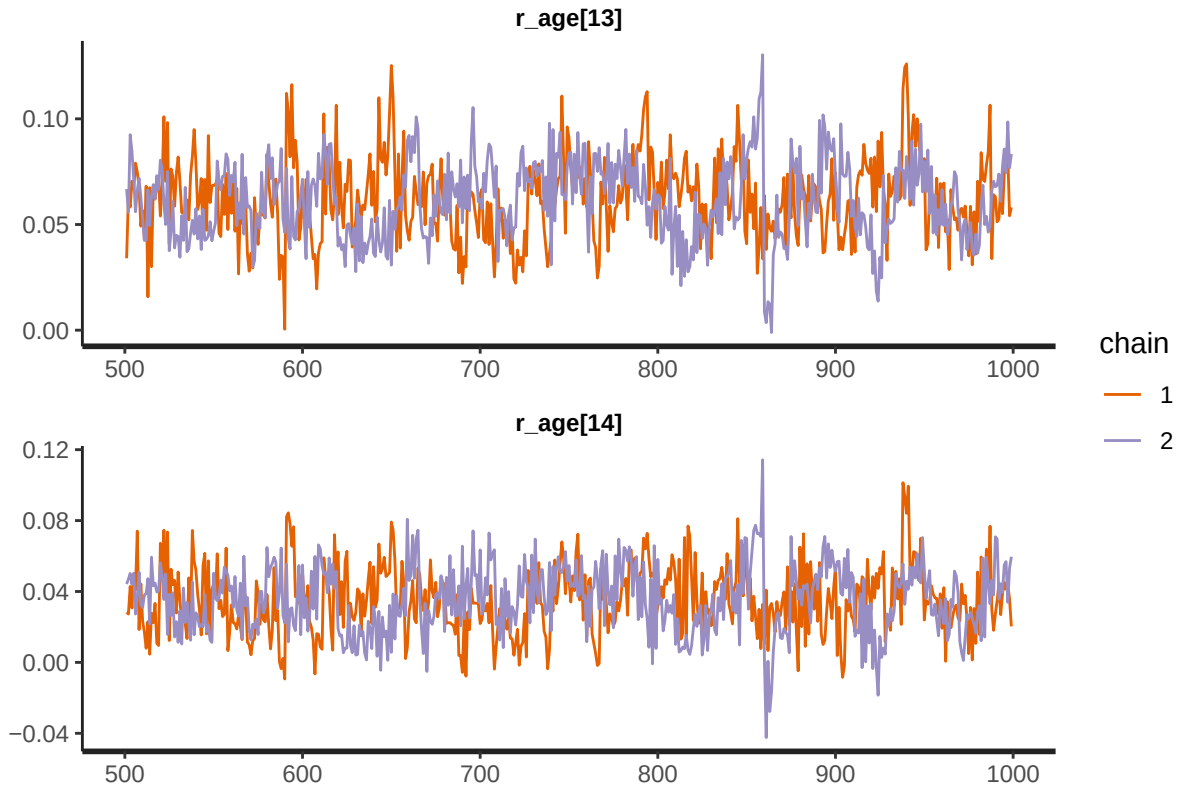

```
traceplot(fit2, pars = c(paras[11], paras[12]), inc_warmup = FALSE, nrow = 2)
```



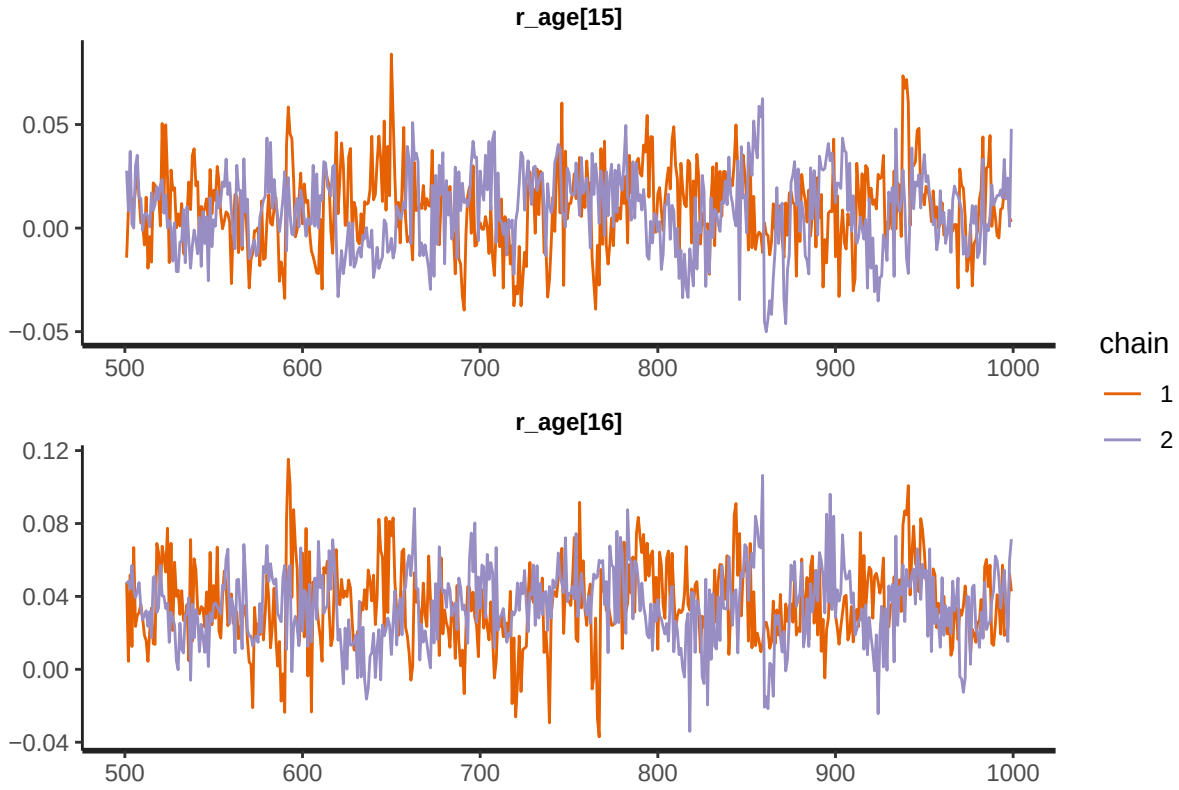
```
traceplot(fit2, pars = c(paras[13], paras[14]), inc_warmup = FALSE, nrow = 2)
```



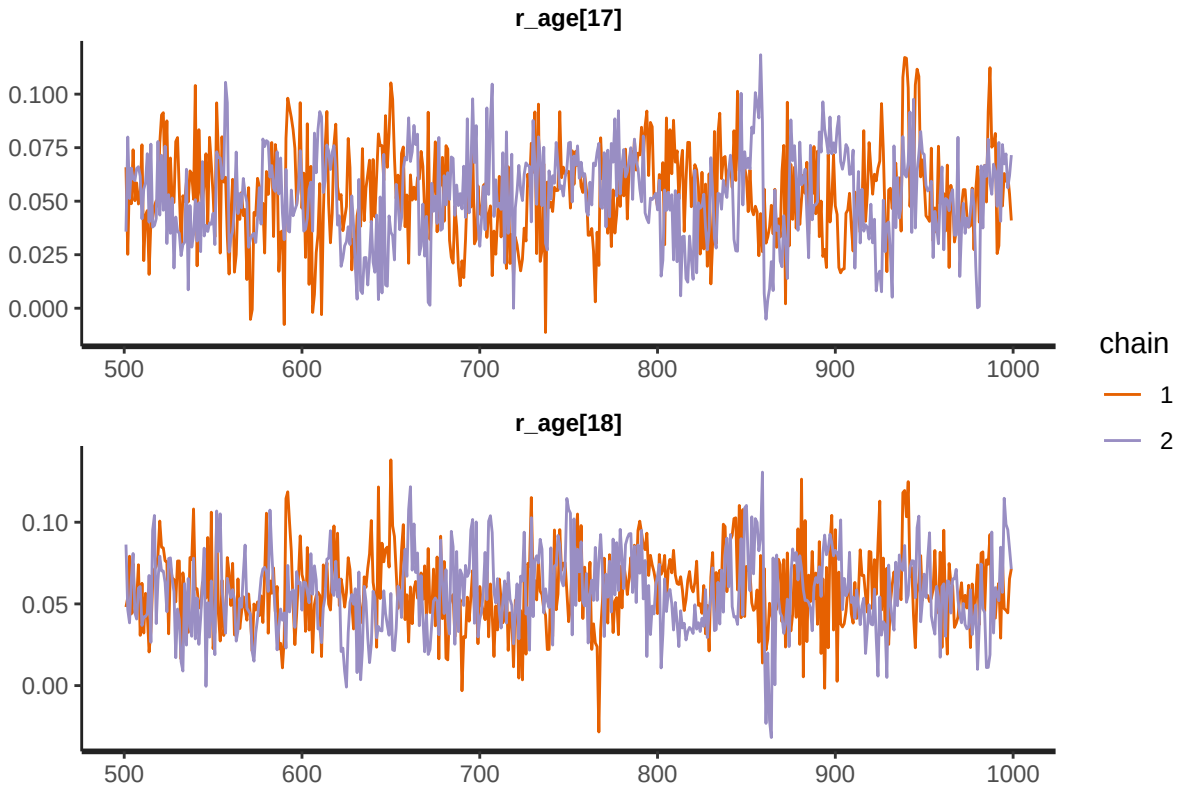

```
traceplot(fit2, pars = c(paras[15], paras[16]), inc_warmup = FALSE, nrow = 2)
```



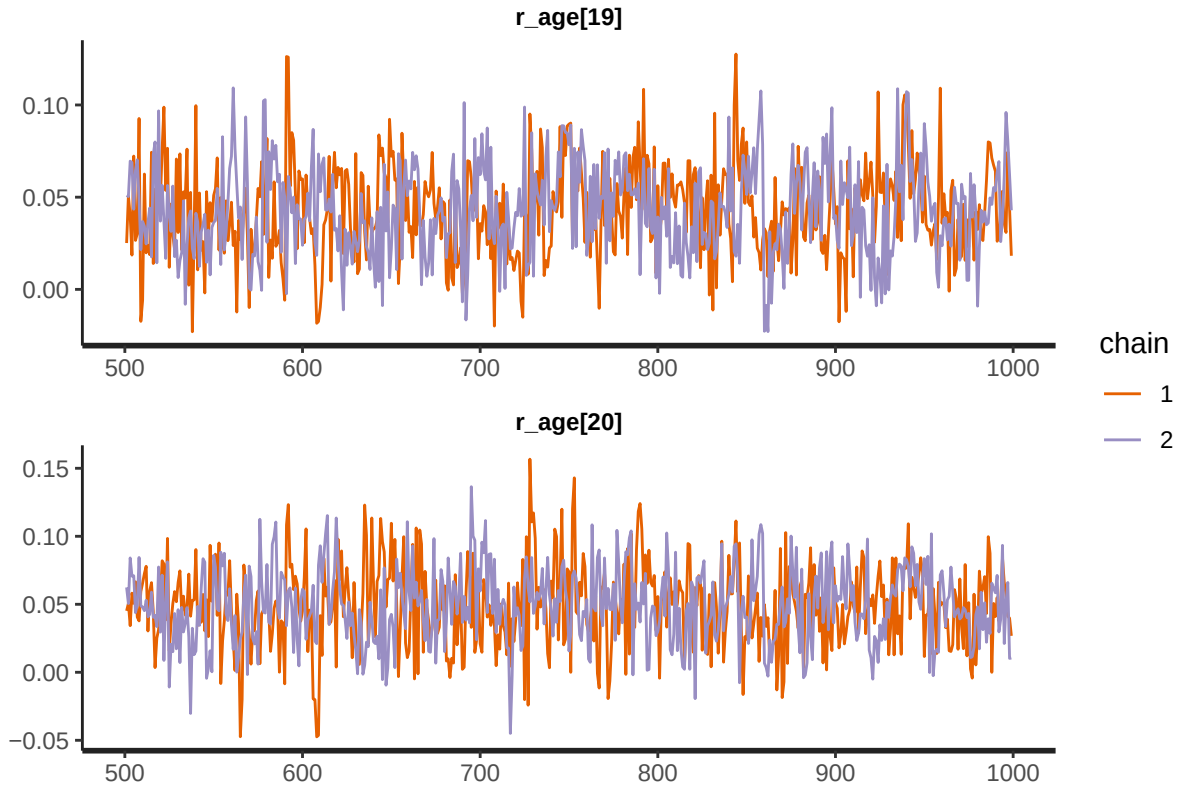
```
traceplot(fit2, pars = c(paras[17], paras[18]), inc_warmup = FALSE, nrow = 2)
```



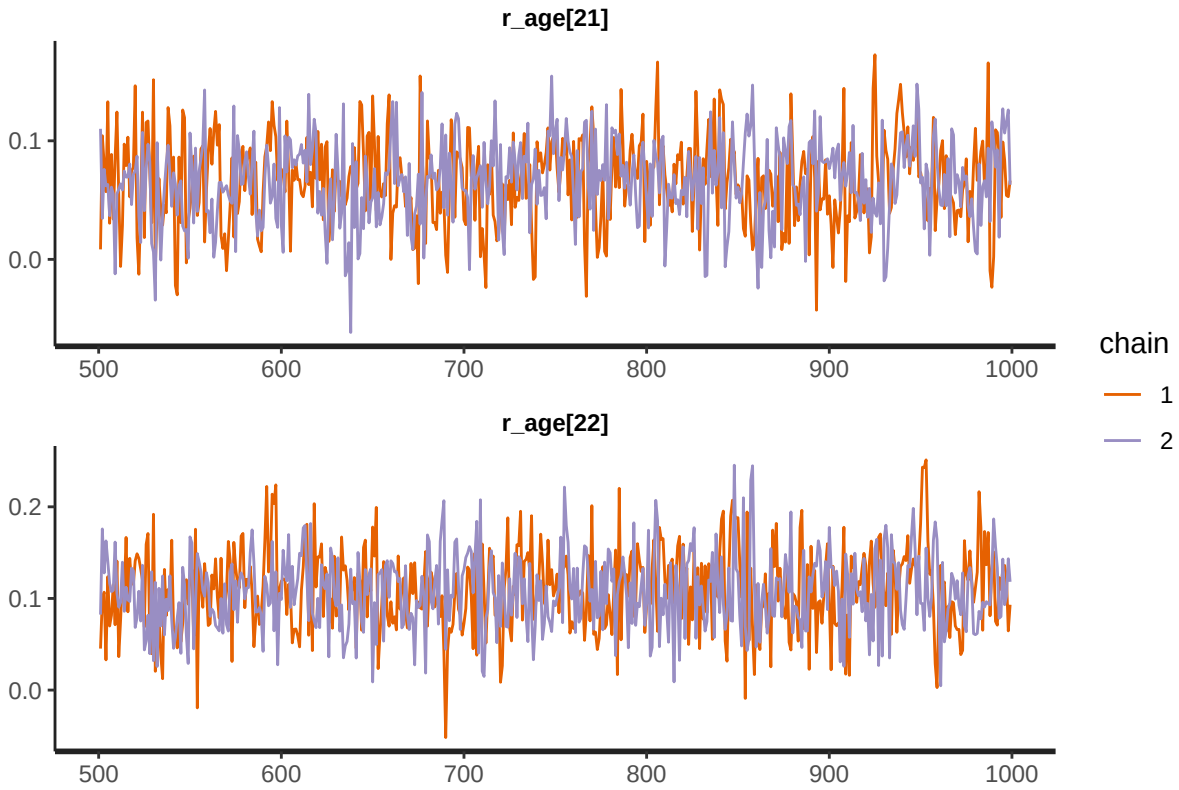

```
traceplot(fit2, pars = c(paras[19], paras[20]), inc_warmup = FALSE, nrow = 2)
```



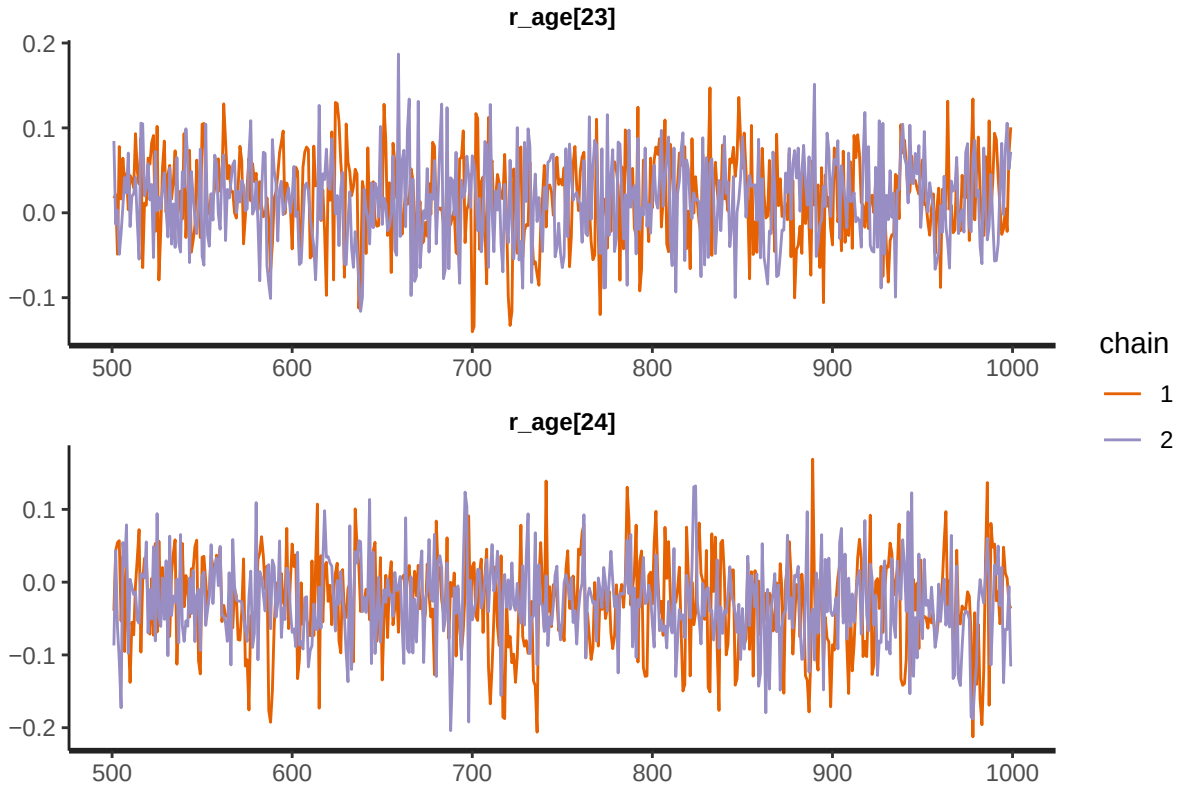
```
traceplot(fit2, pars = c(paras[21], paras[22]), inc_warmup = FALSE, nrow = 2)
```



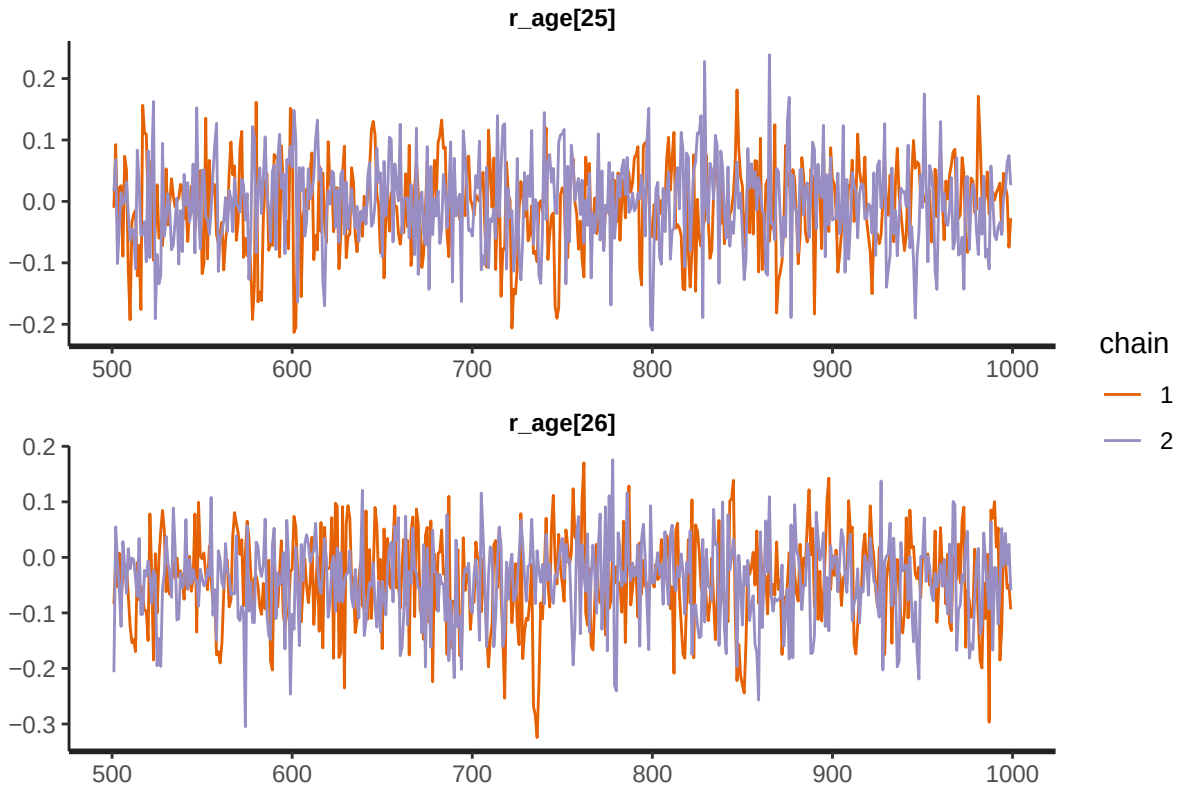

```
traceplot(fit2, pars = c(paras[23], paras[24]), inc_warmup = FALSE, nrow = 2)
```



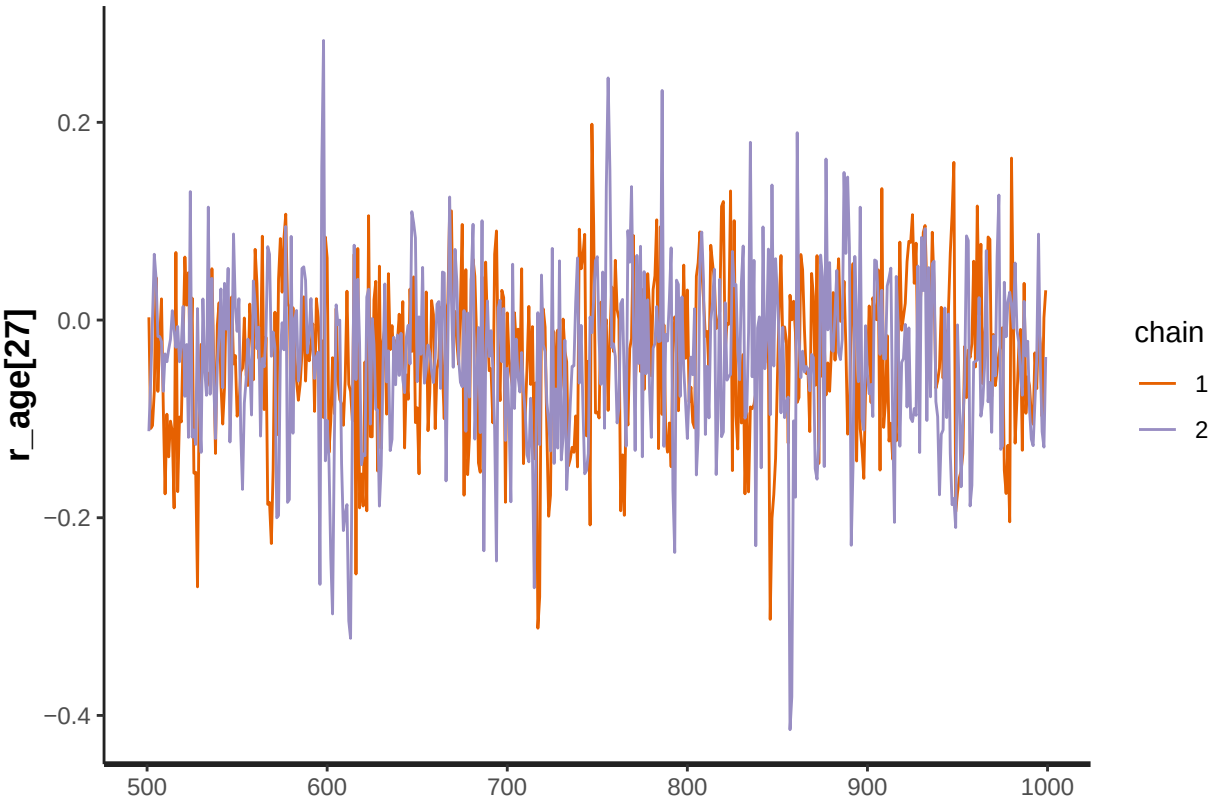
```
traceplot(fit2, pars = c(paras[25], paras[26]), inc_warmup = FALSE, nrow = 2)
```




```
traceplot(fit2, pars = c(paras[27], paras[28]), inc_warmup = FALSE, nrow = 2)
```

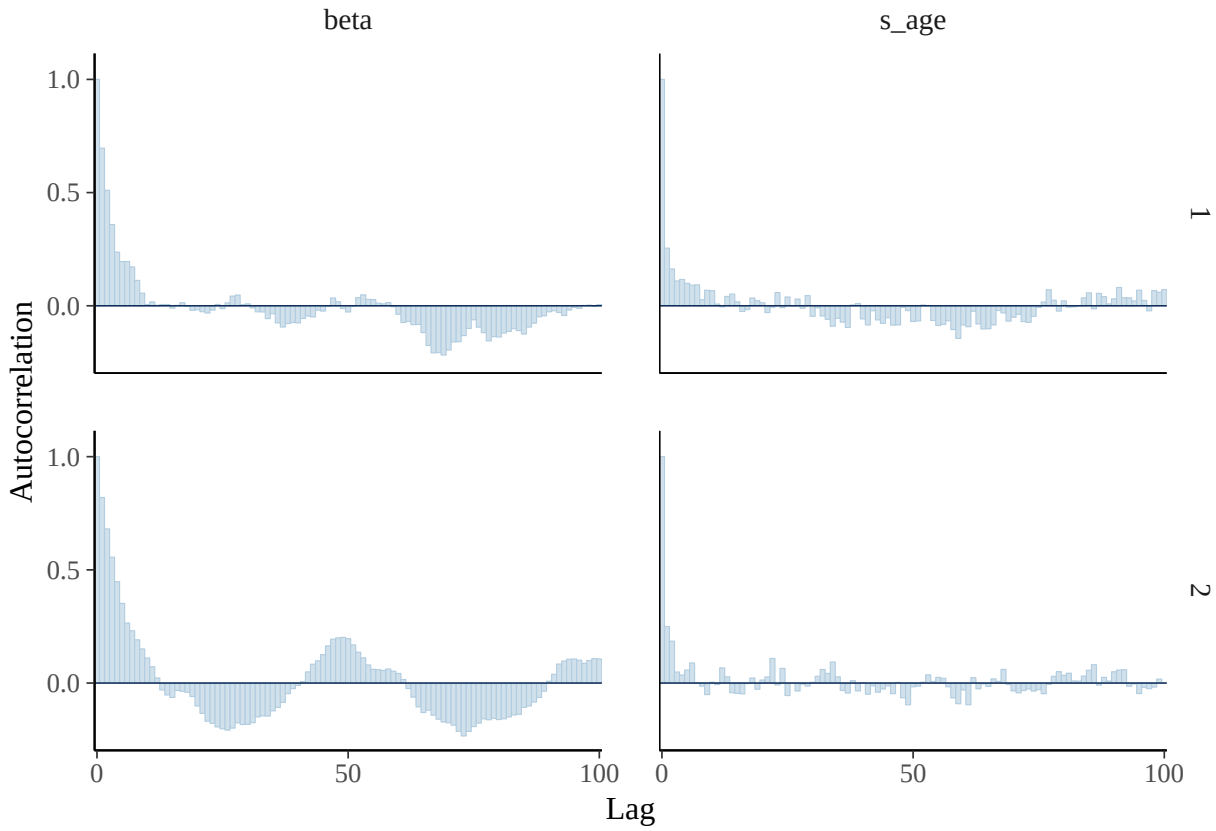


```
traceplot(fit2, pars = paras[29],                inc_warmup = FALSE, nrow = 2)
```

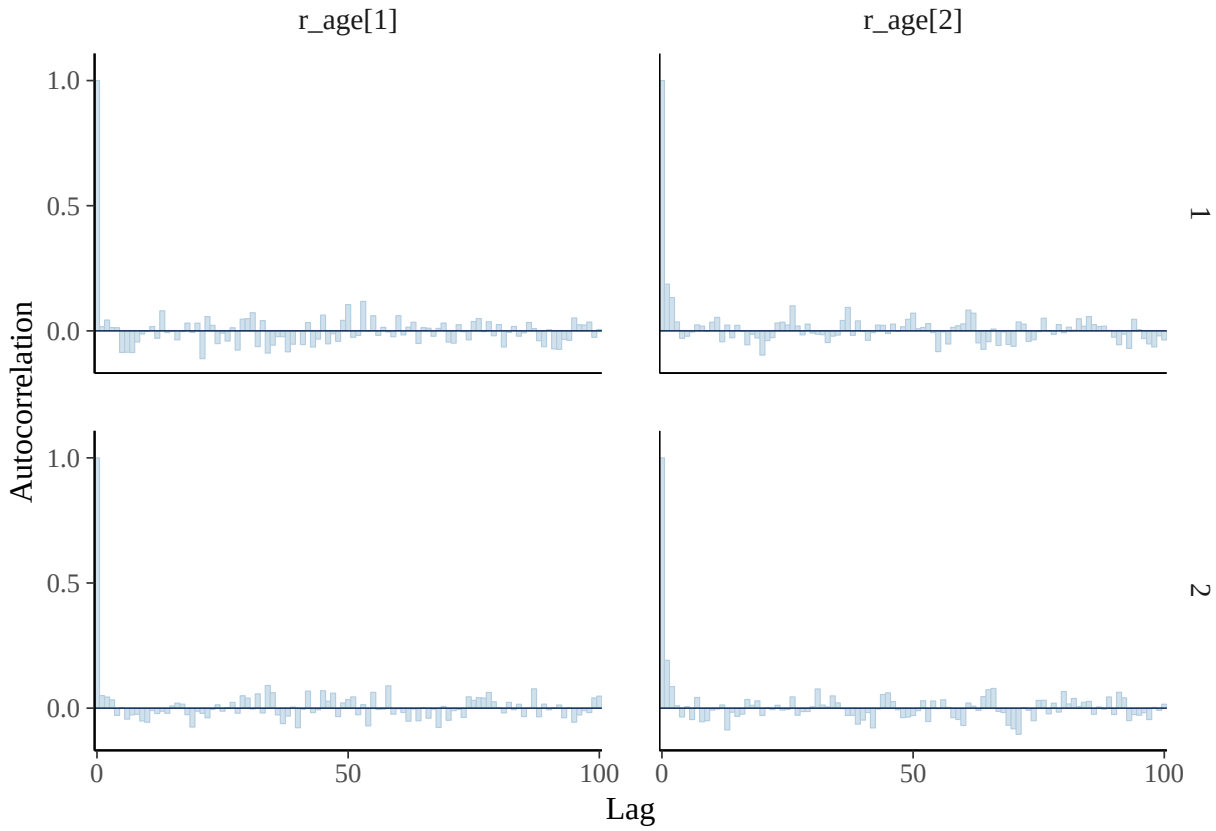


Autocorrelation (model2)

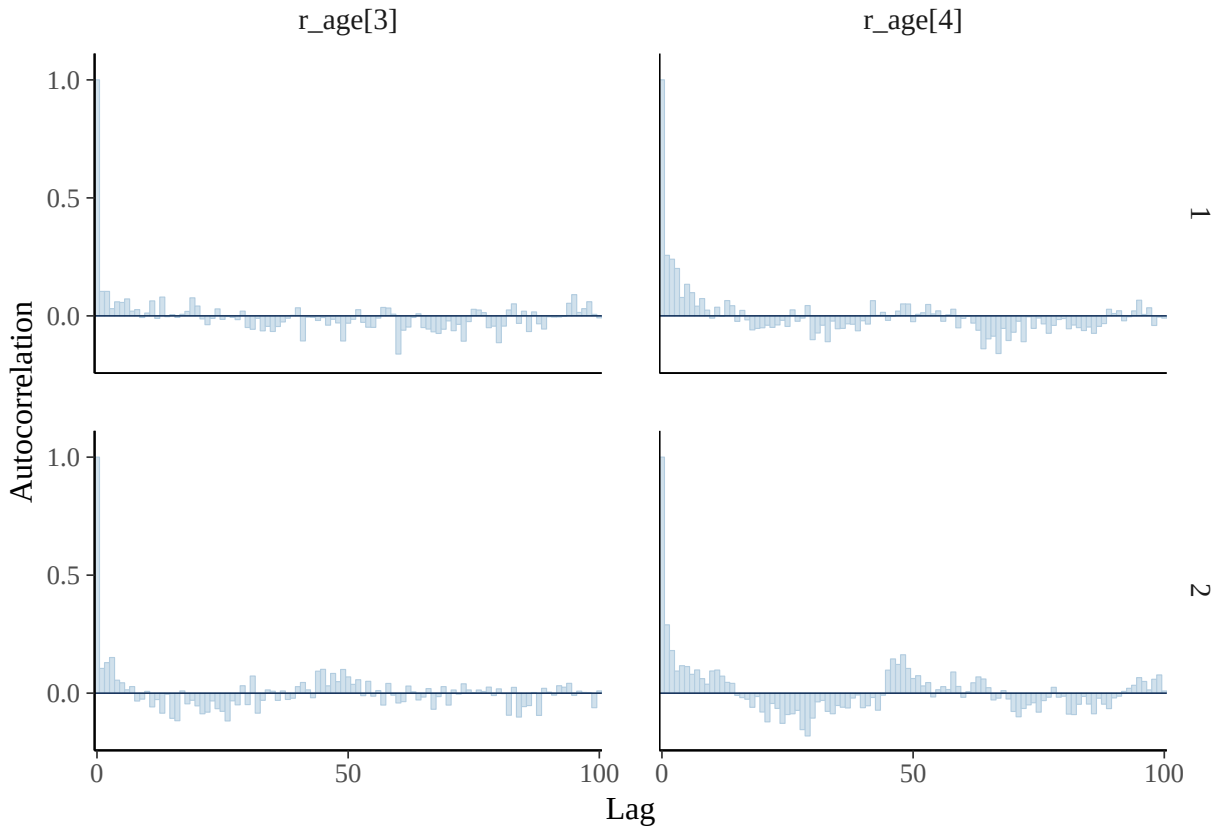
```
mcmc_acf_bar(fit2, pars = c(paras[1],  paras[2]),   lags = num.lag)
```



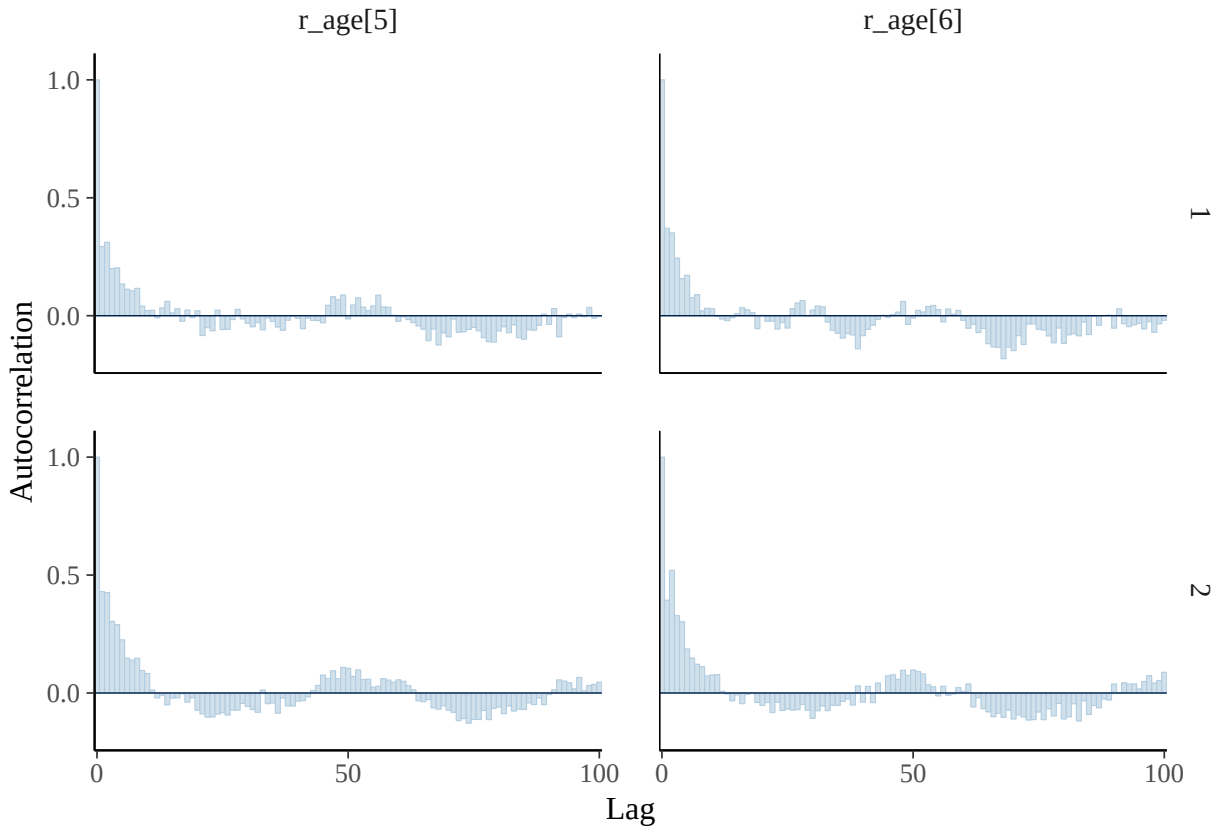

```
mcmc_acf_bar(fit2, pars = c(paras[3],  paras[4]),   lags = num.lag)
```



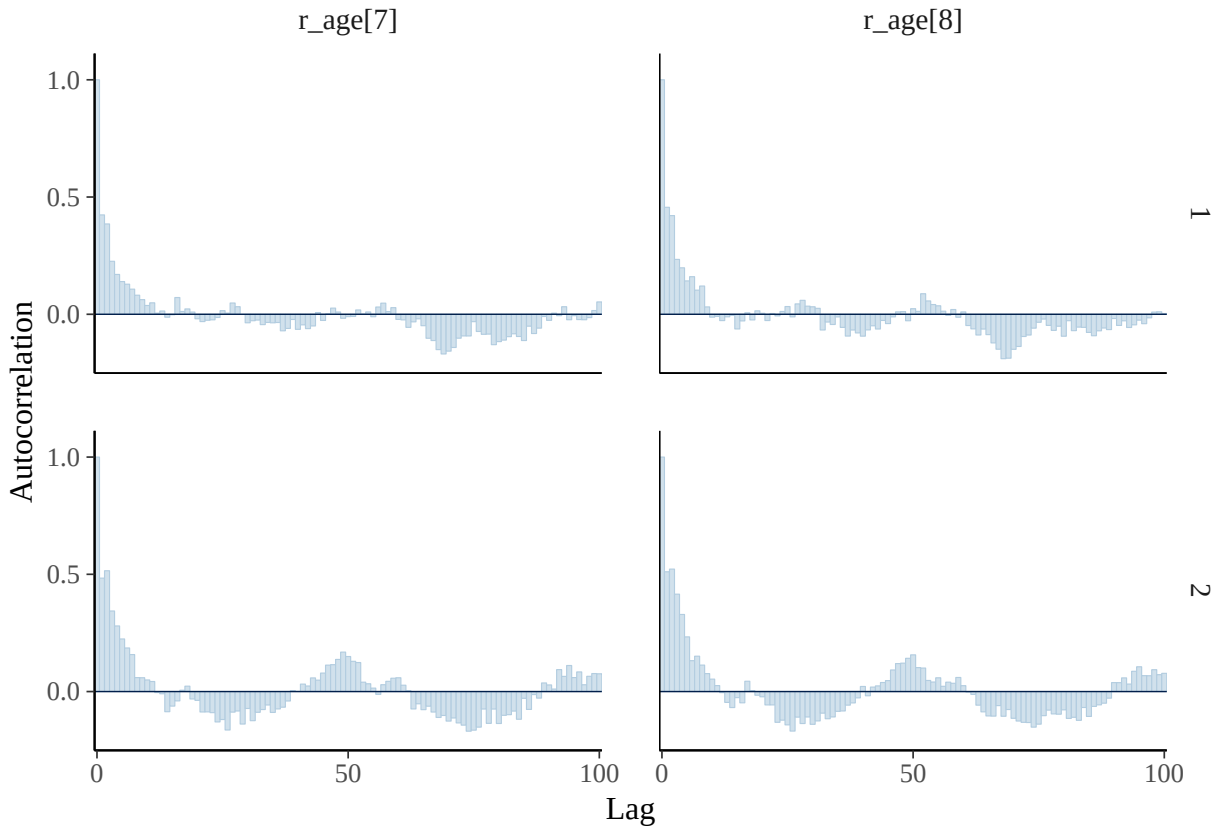

```
mcmc_acf_bar(fit2, pars = c(paras[5],  paras[6]),   lags = num.lag)
```



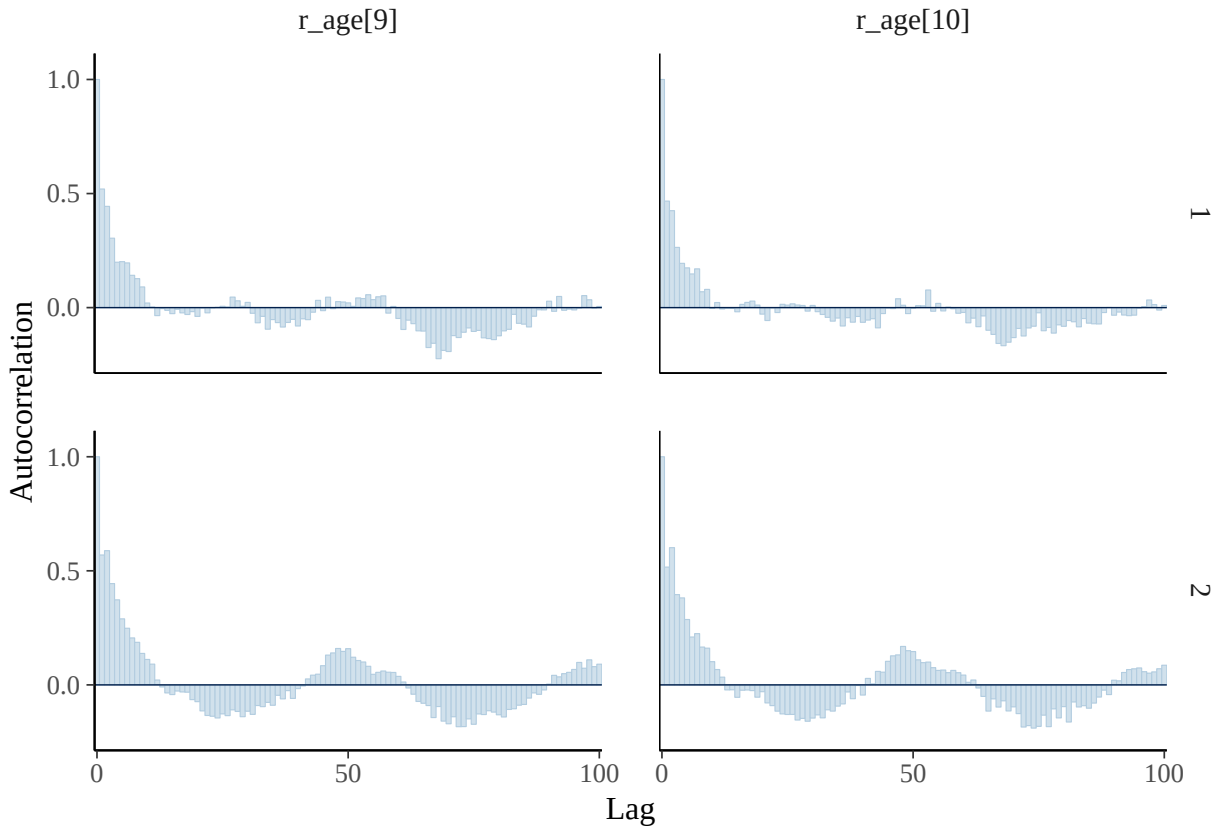

```
mcmc_acf_bar(fit2, pars = c(paras[7],  paras[8]),   lags = num.lag)
```



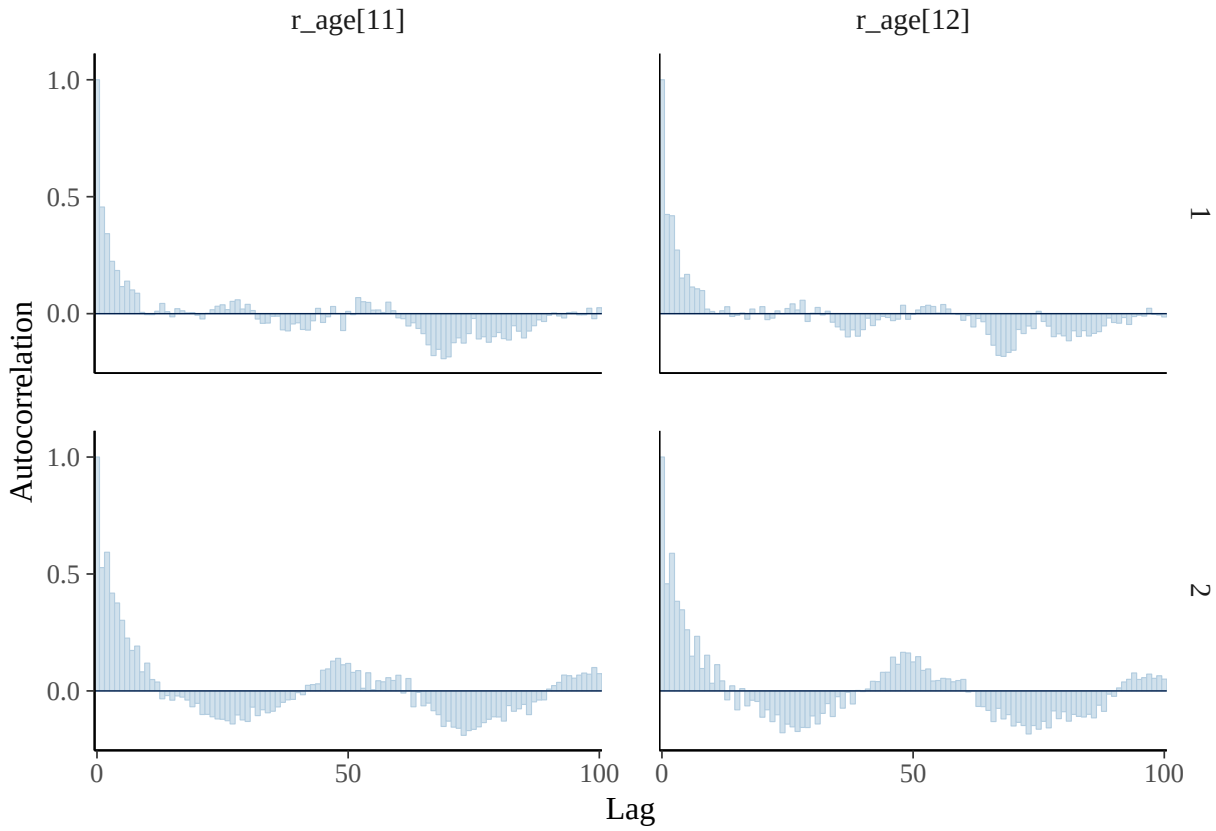

```
mcmc_acf_bar(fit2, pars = c(paras[9],  paras[10]),  lags = num.lag)
```



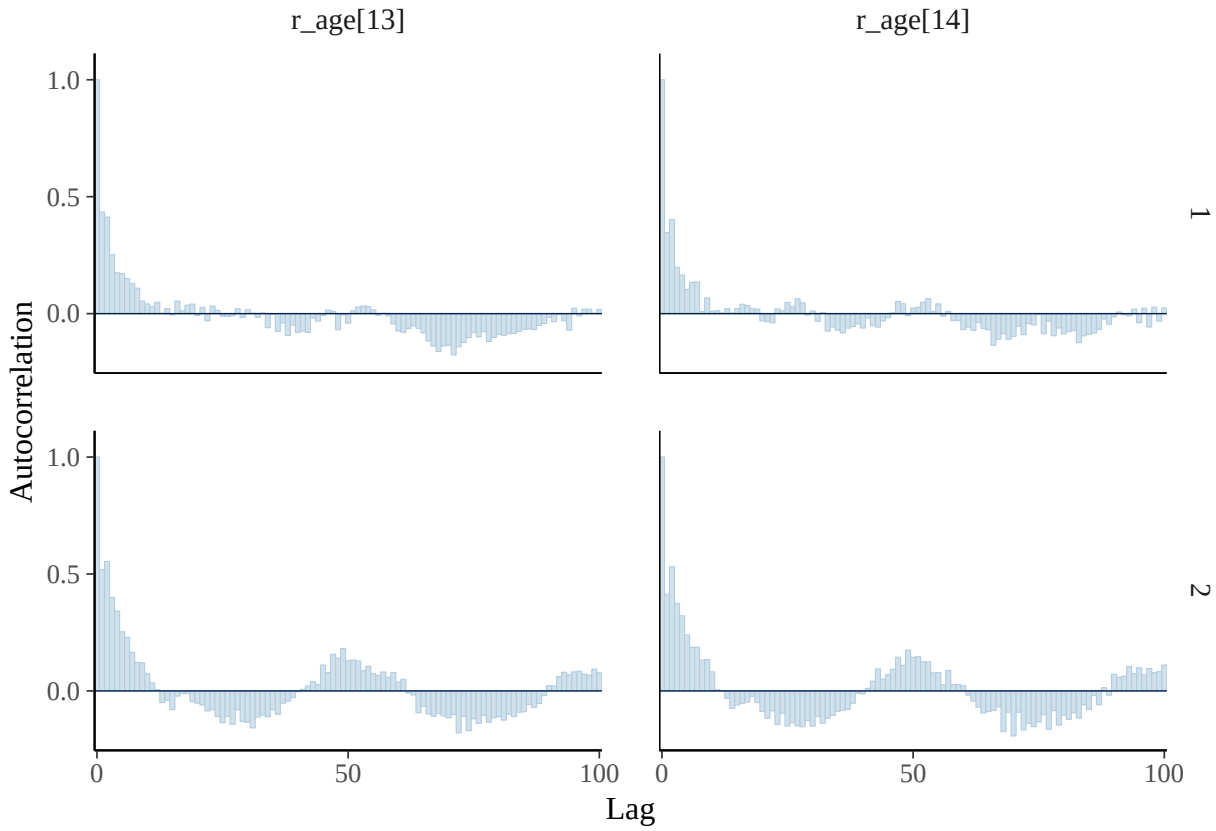

```
mcmc_acf_bar(fit2, pars = c(paras[11], paras[12]),  lags = num.lag)
```



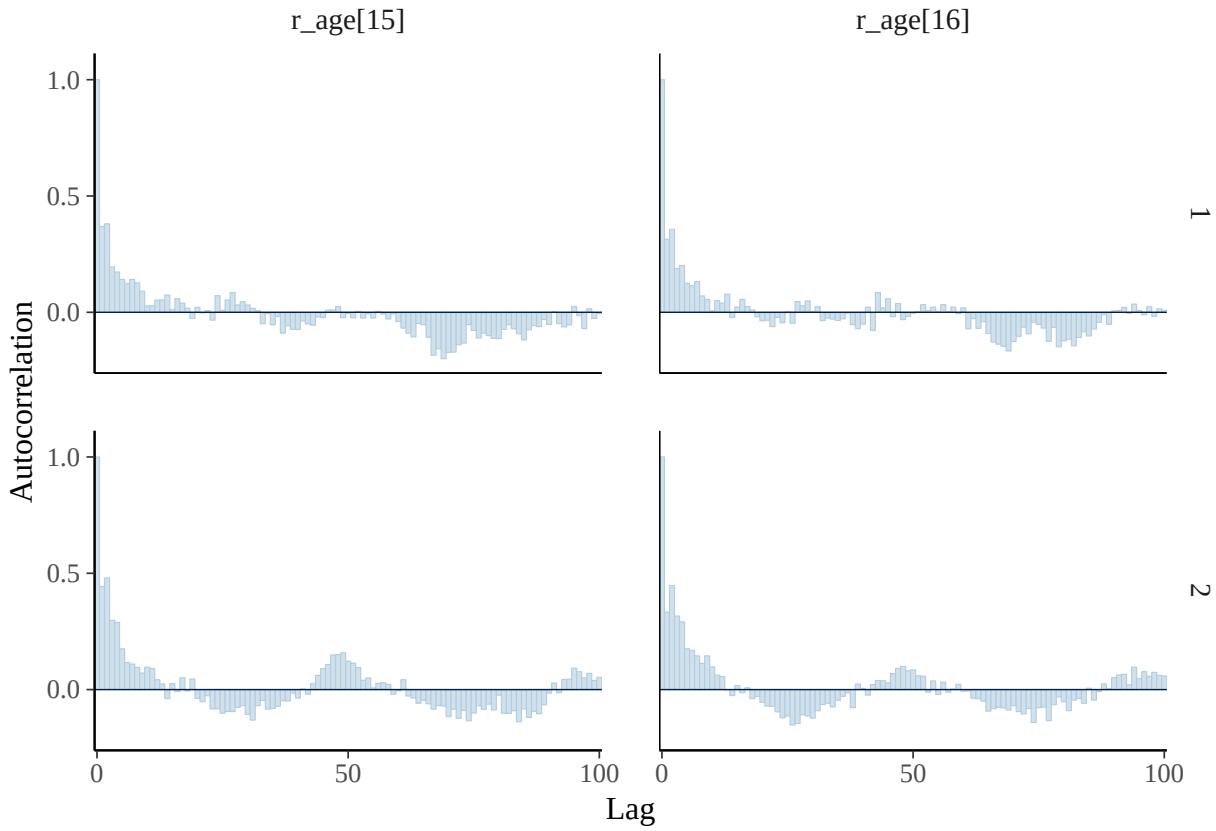

```
mcmc_acf_bar(fit2, pars = c(paras[13], paras[14]),  lags = num.lag)
```



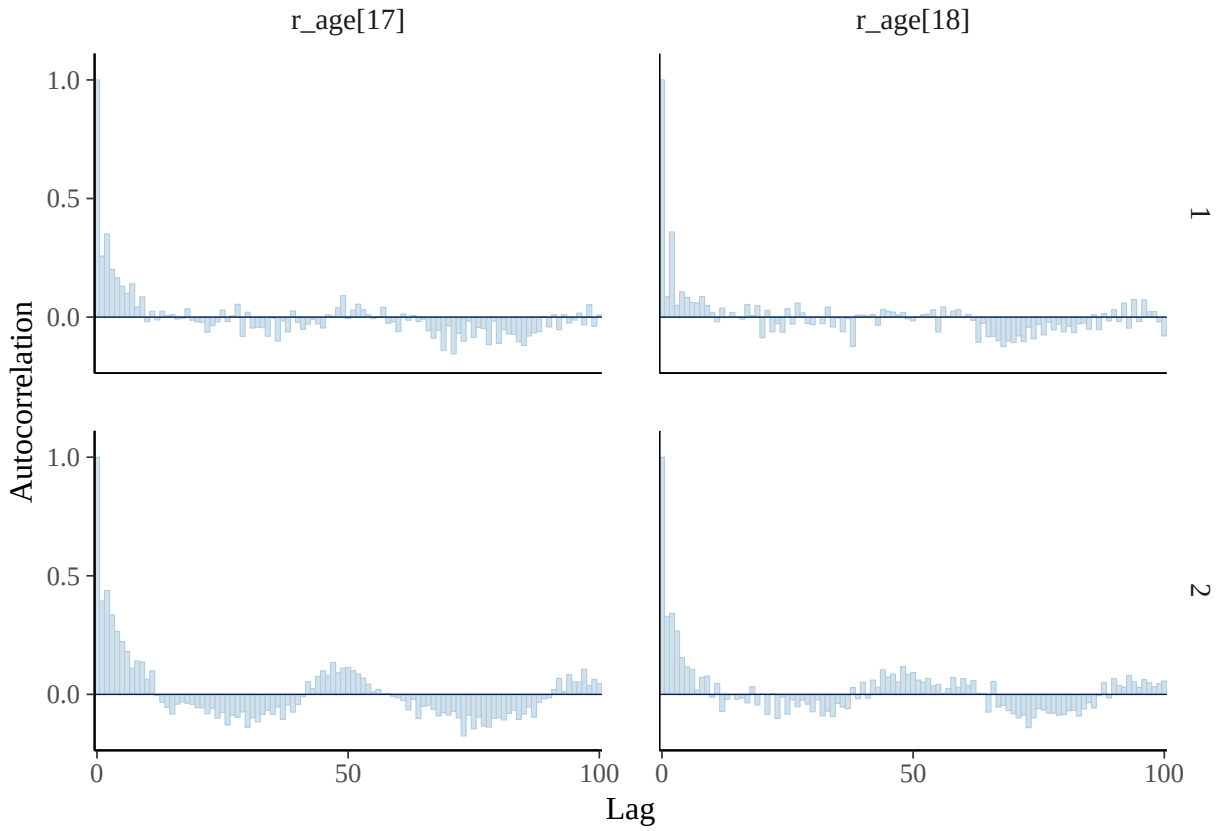

```
mcmc_acf_bar(fit2, pars = c(paras[15], paras[16]),  lags = num.lag)
```



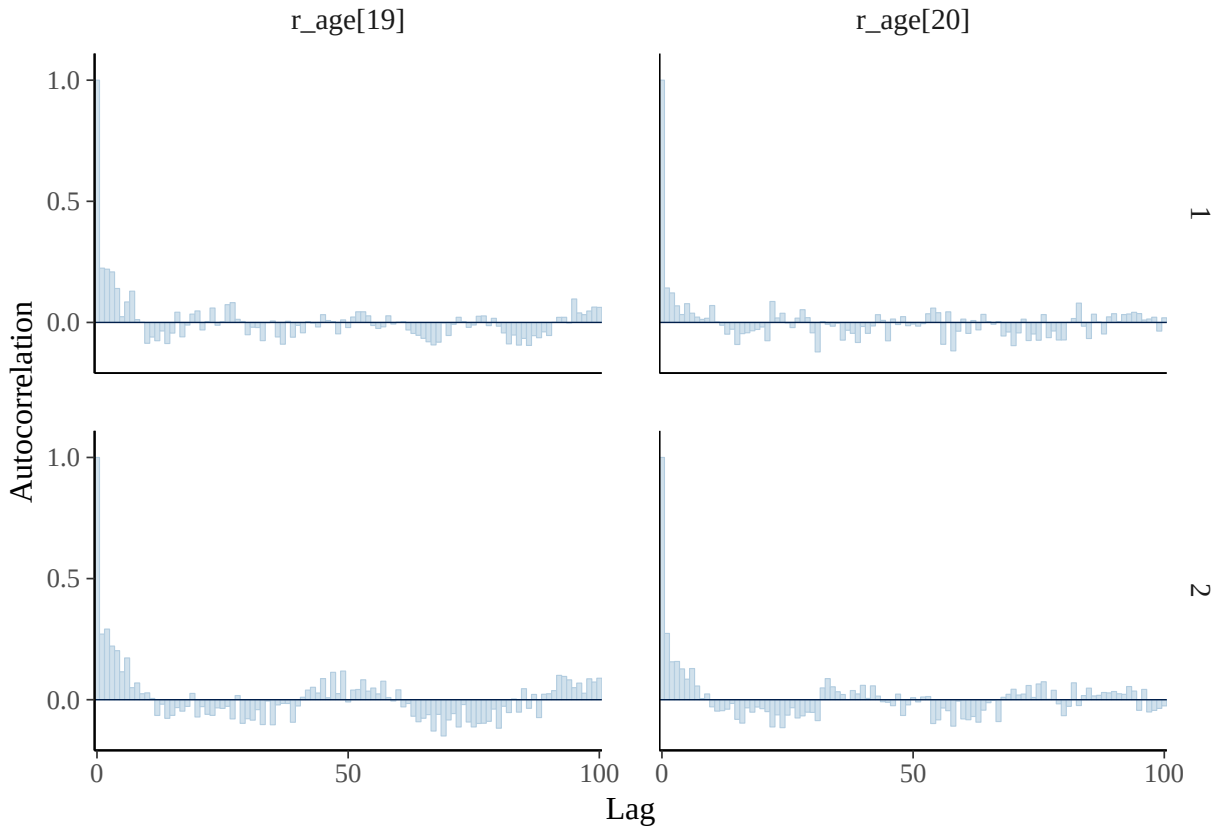

```
mcmc_acf_bar(fit2, pars = c(paras[17], paras[18]),  lags = num.lag)
```



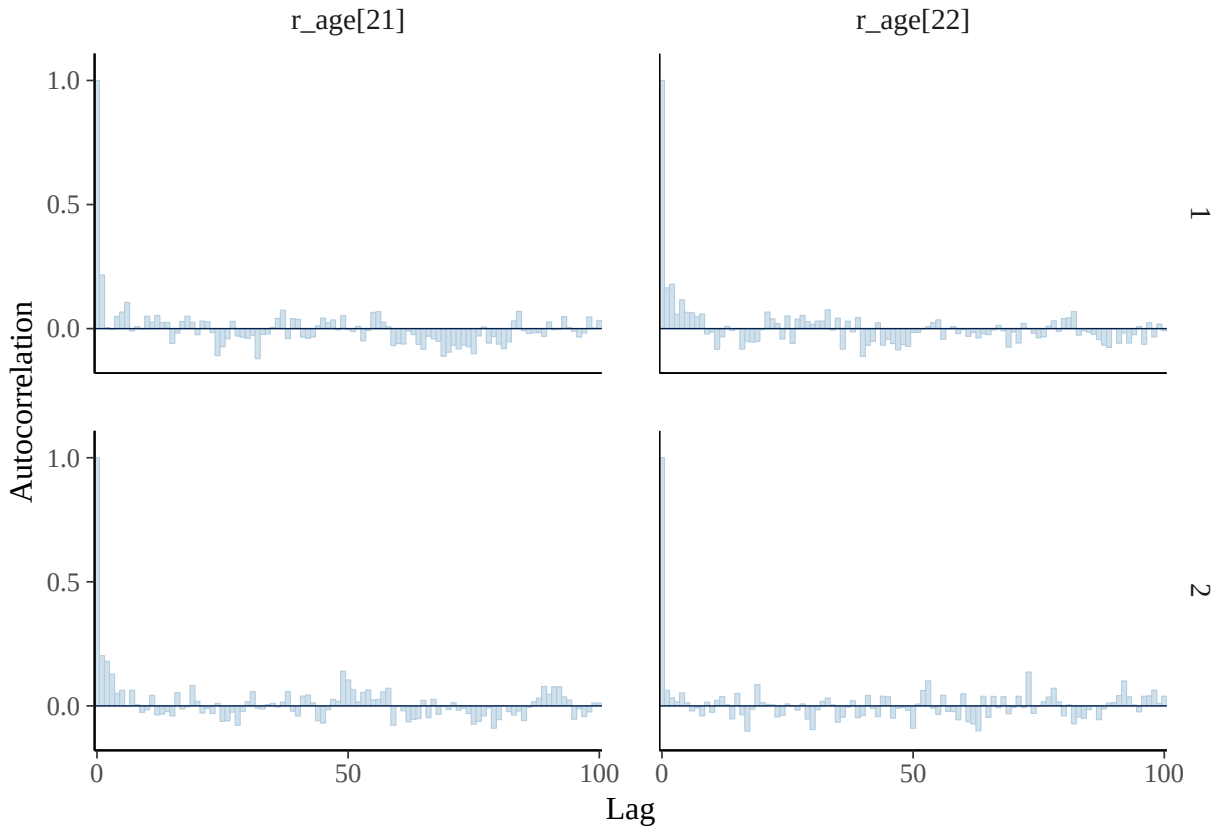

```
mcmc_acf_bar(fit2, pars = c(paras[19], paras[20]),  lags = num.lag)
```



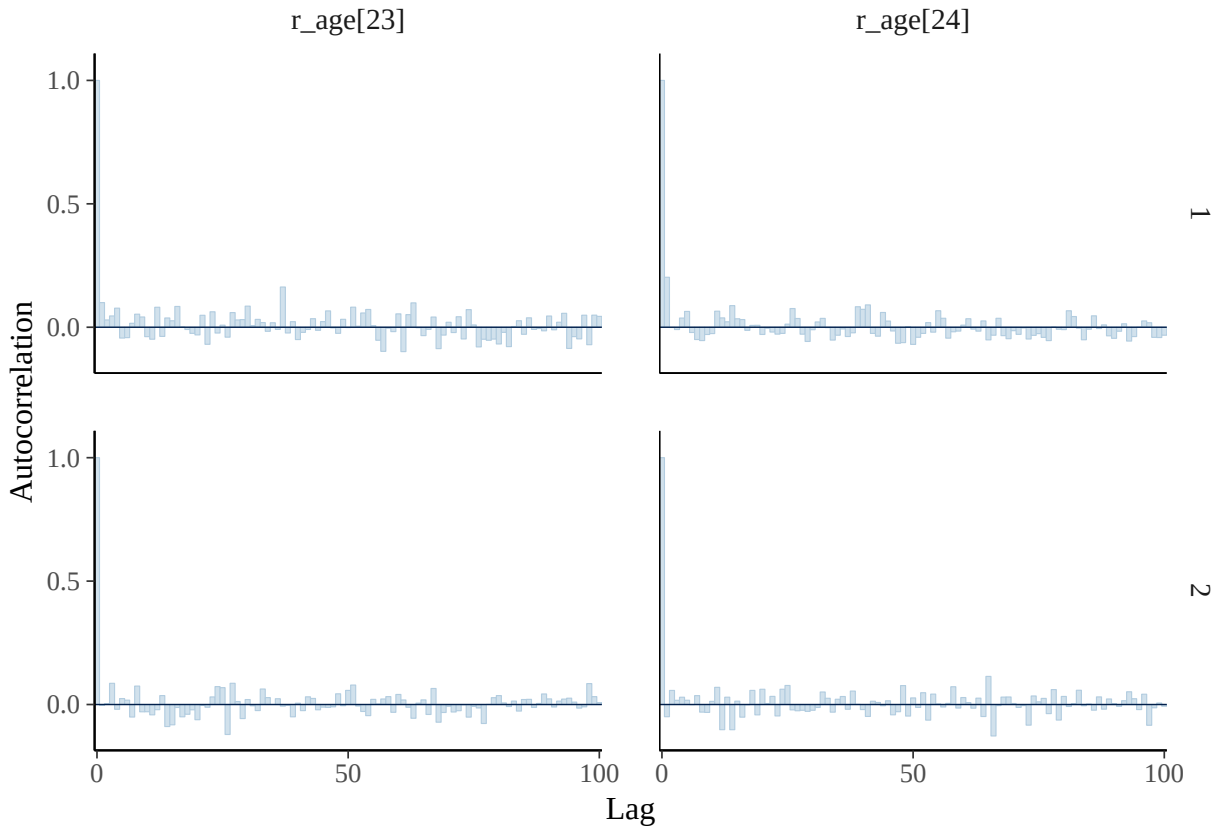

```
mcmc_acf_bar(fit2, pars = c(paras[21], paras[22]),  lags = num.lag)
```



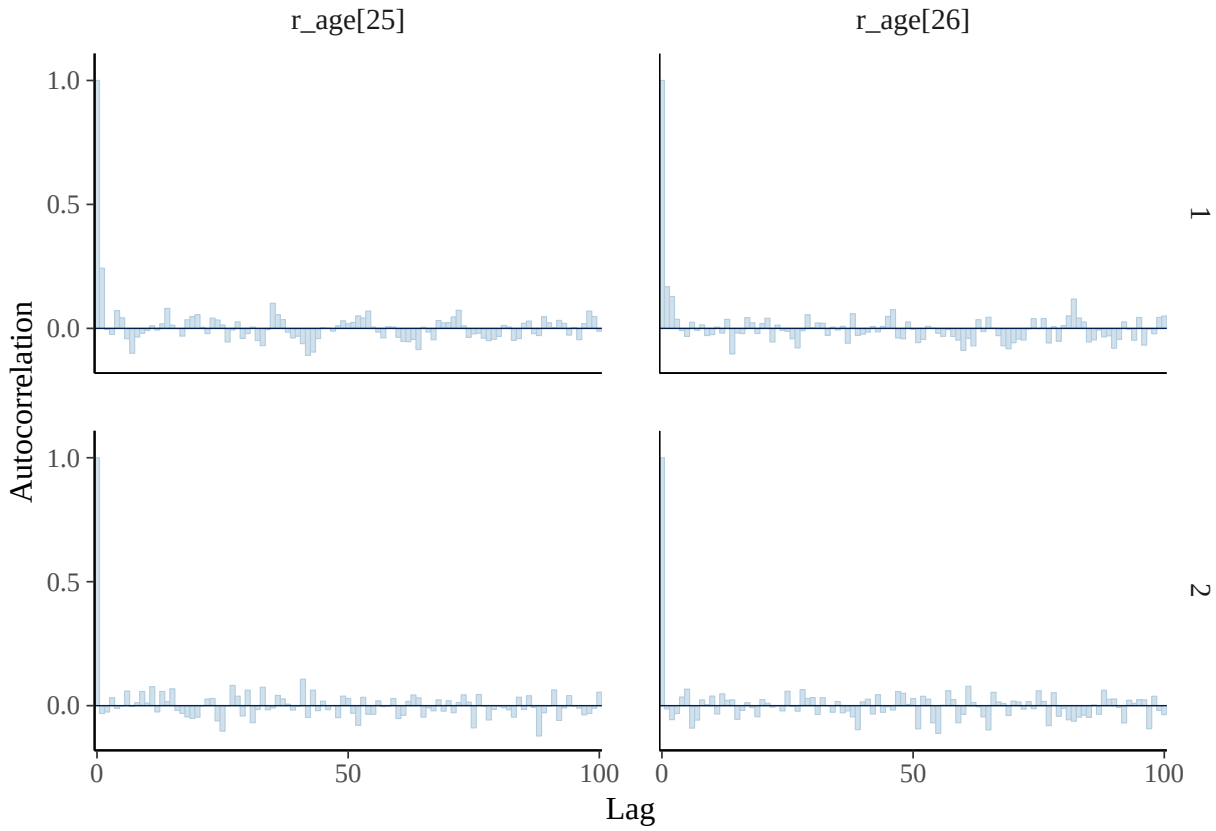

```
mcmc_acf_bar(fit2, pars = c(paras[23], paras[24]),  lags = num.lag)
```



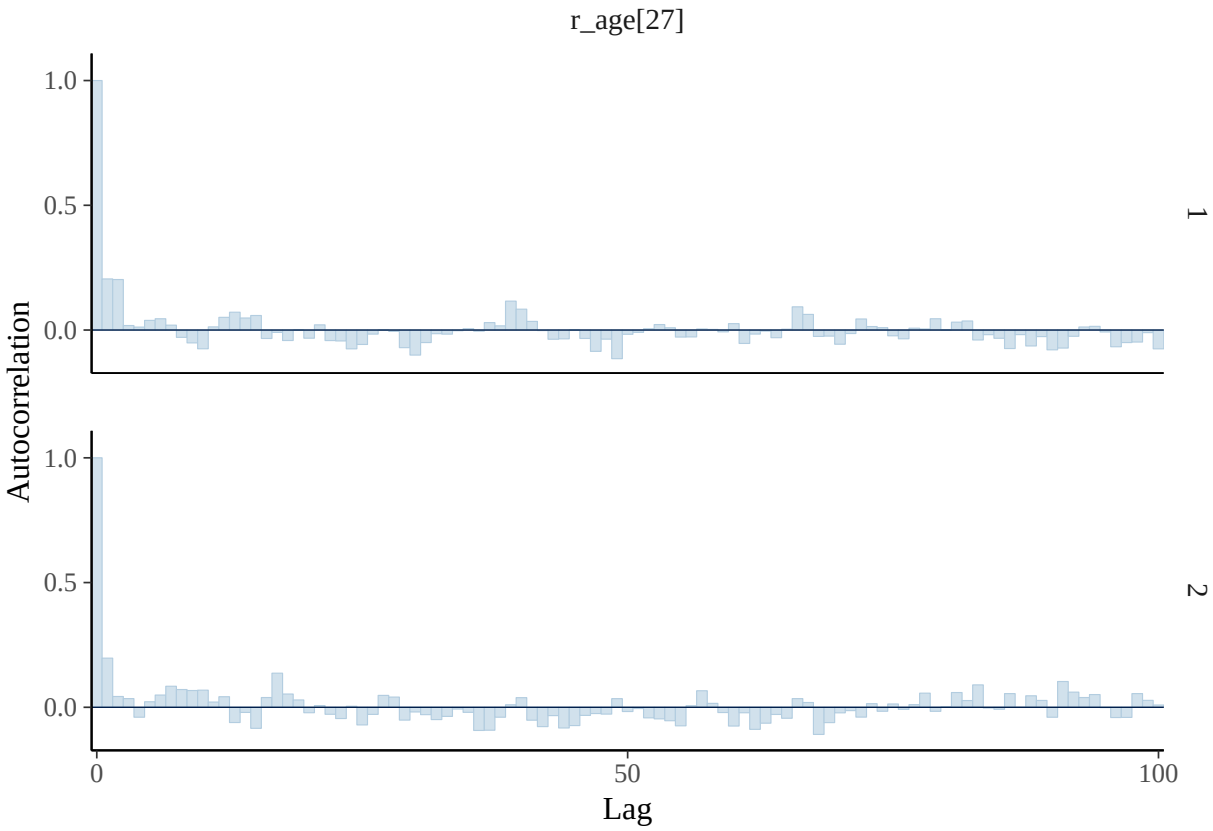

```
mcmc_acf_bar(fit2, pars = c(paras[25], paras[26]),  lags = num.lag)
```




```
mcmc_acf_bar(fit2, pars = c(paras[27], paras[28]),  lags = num.lag)
```

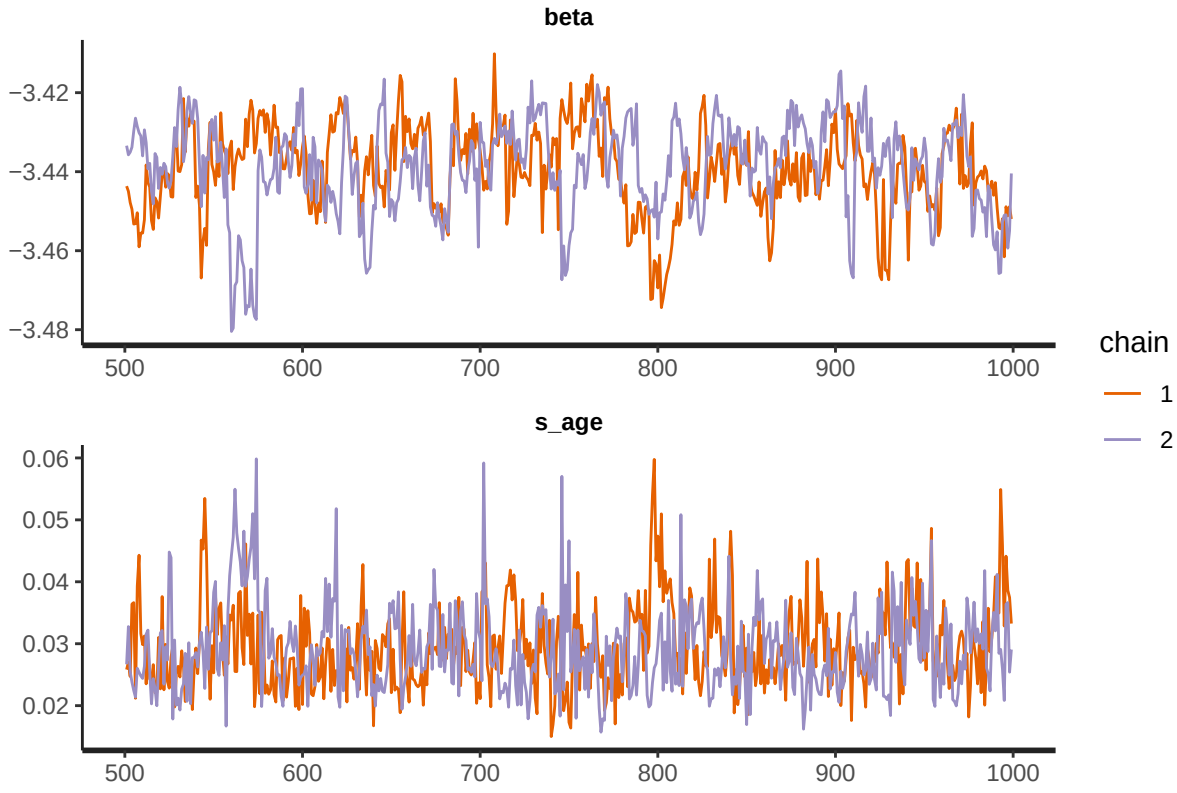



```
mcmc_acf_bar(fit2, pars = paras[29],                lags = num.lag)
```

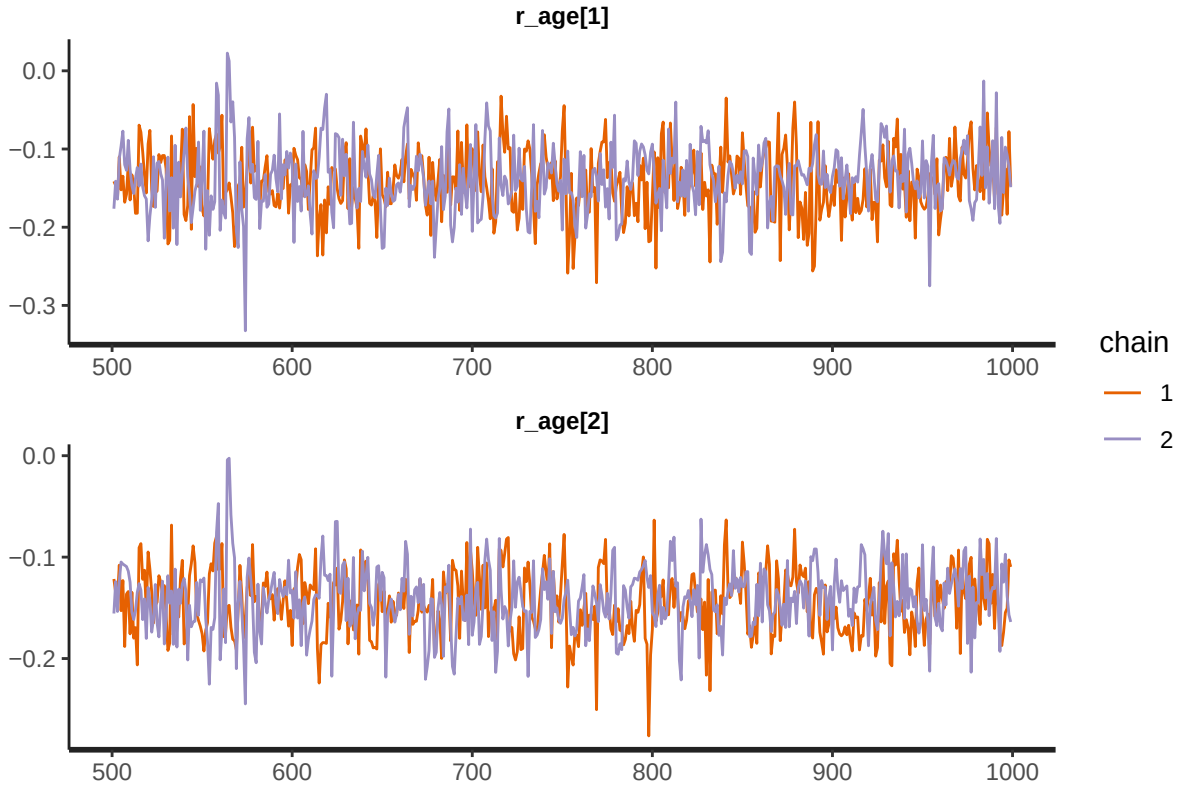


Traceplot (model3)

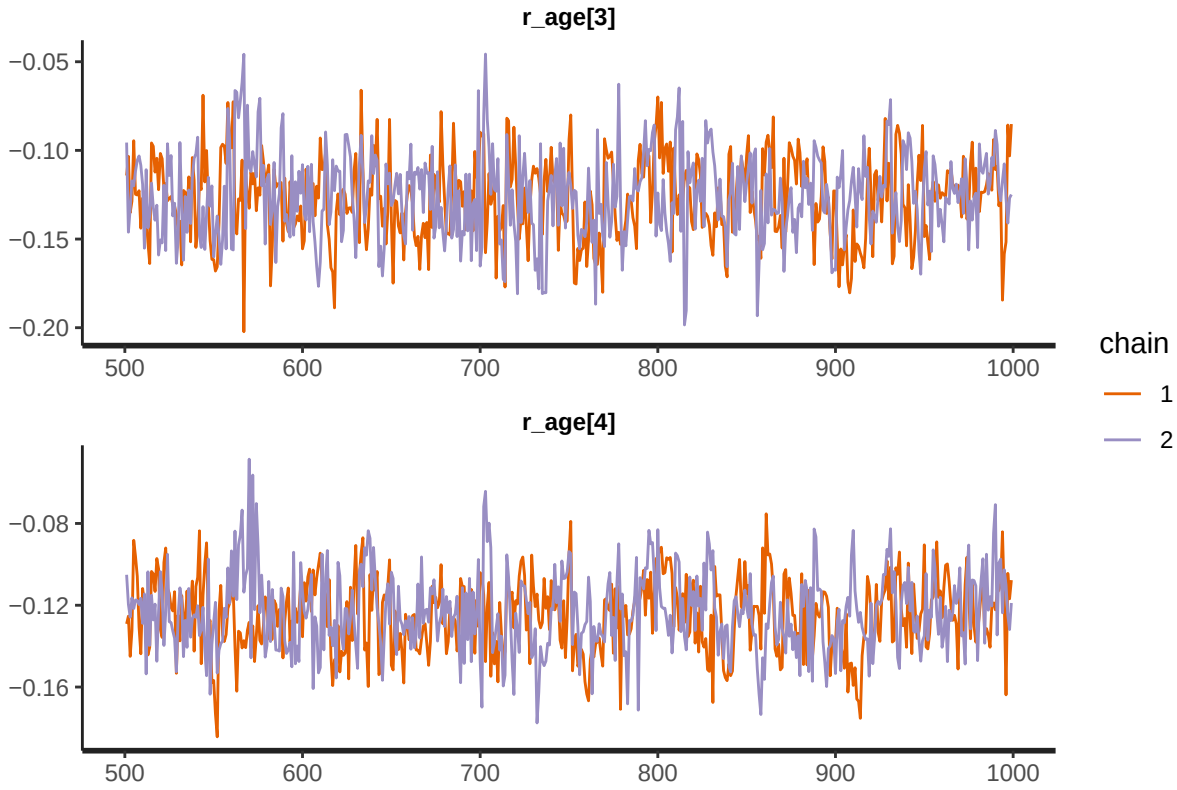
```
traceplot(fit3, pars = c(paras[1],  paras[2]),  inc_warmup = FALSE, nrow = 2)
```



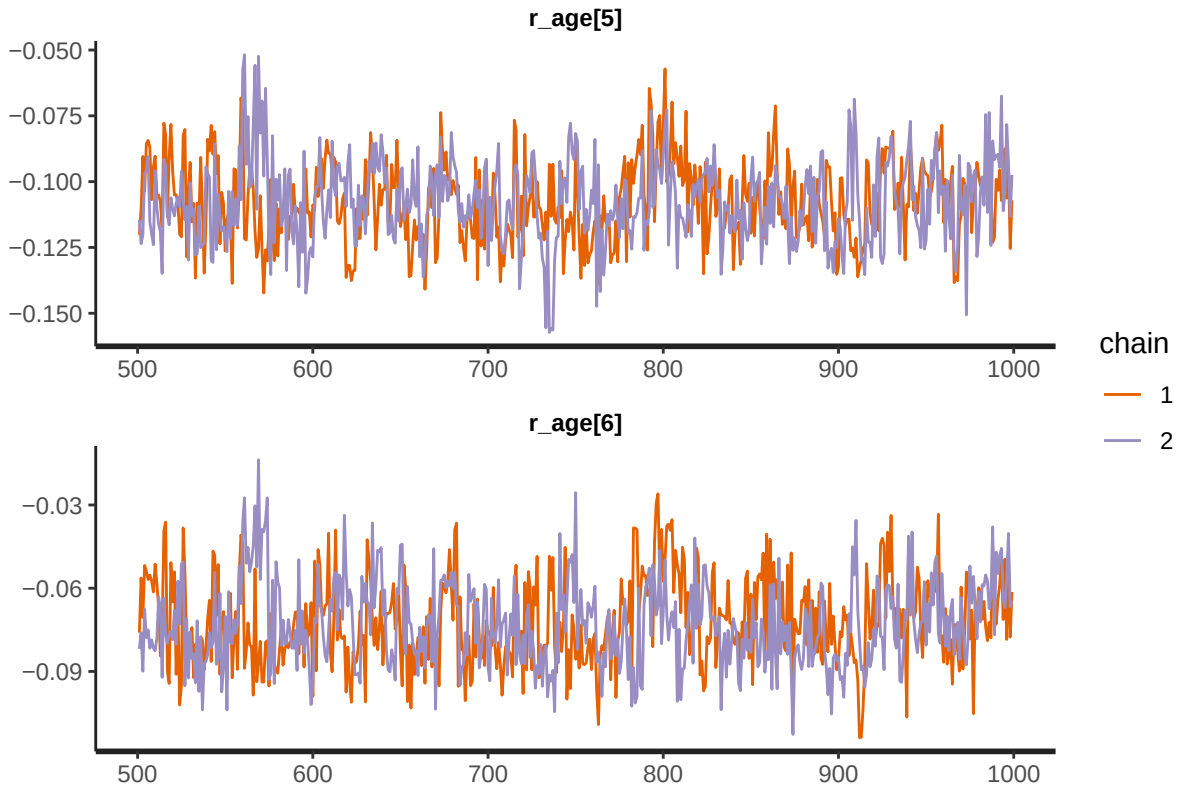

```
traceplot(fit3, pars = c(paras[3],  paras[4]),  inc_warmup = FALSE, nrow = 2)
```



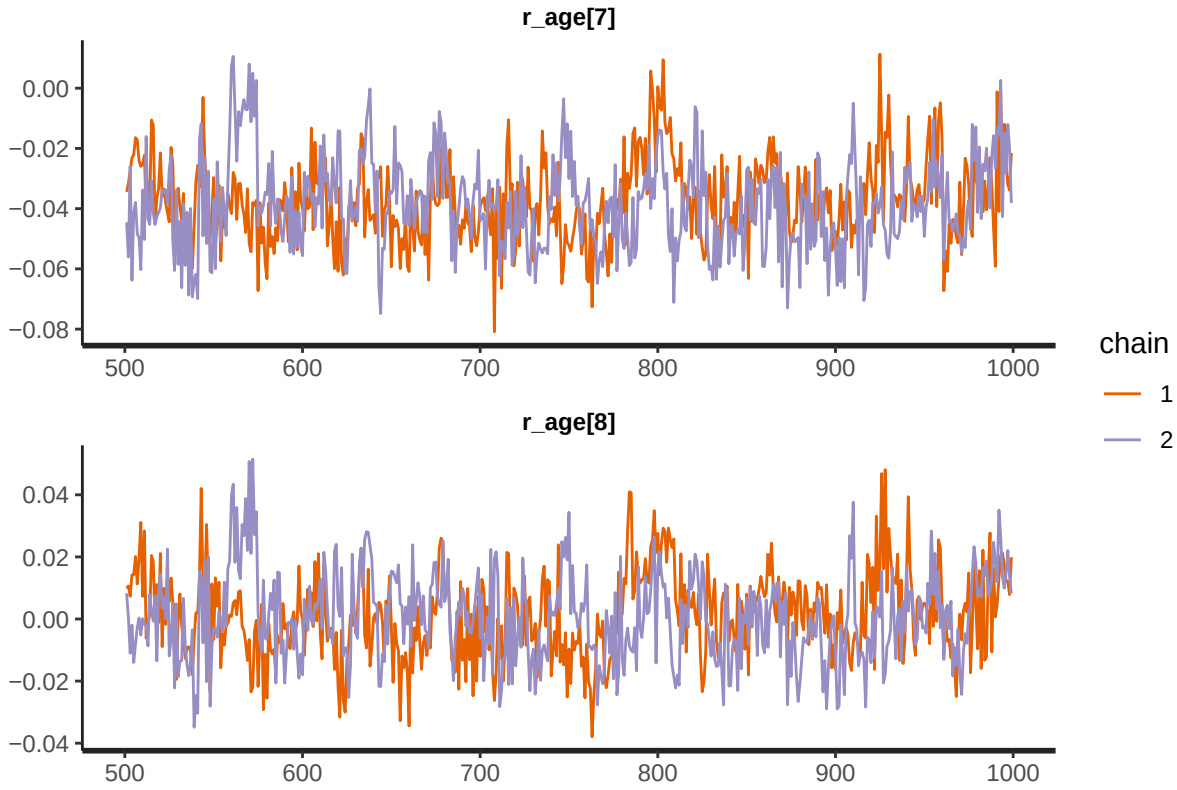
```
traceplot(fit3, pars = c(paras[5],  paras[6]),  inc_warmup = FALSE, nrow = 2)
```



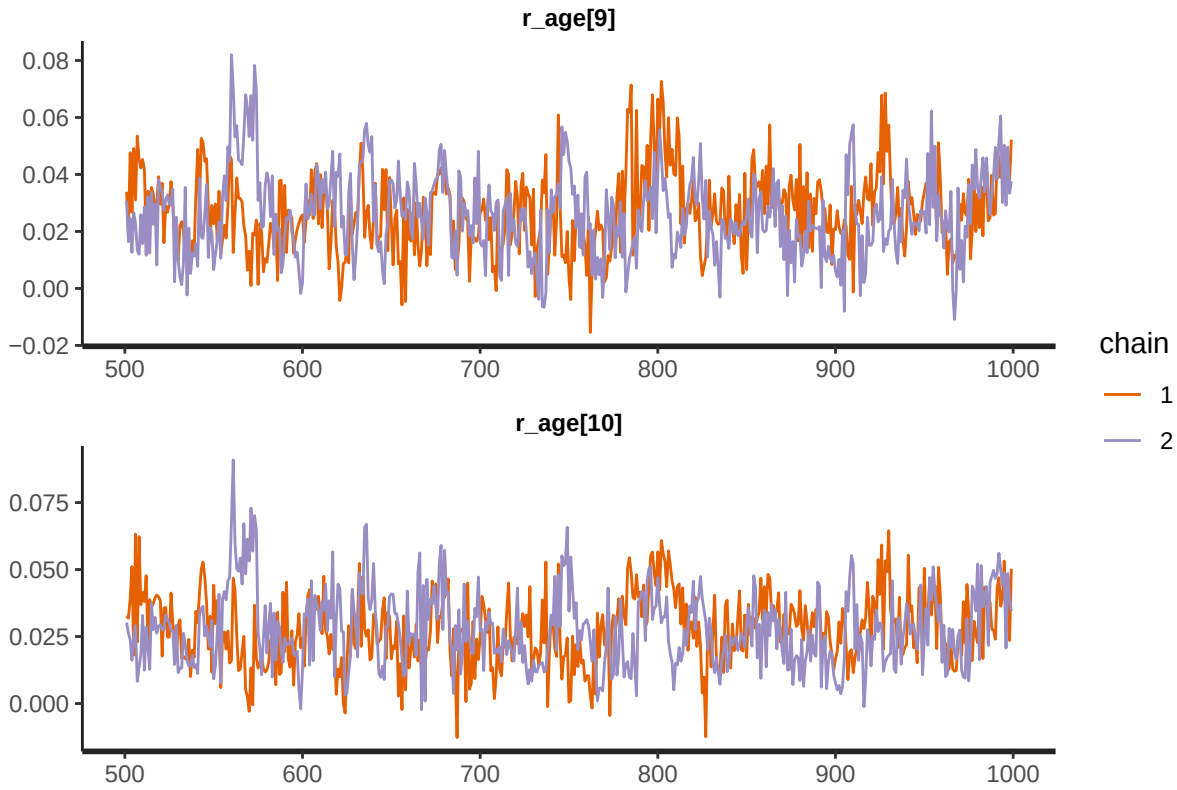

```
traceplot(fit3, pars = c(paras[7],  paras[8]),  inc_warmup = FALSE, nrow = 2)
```



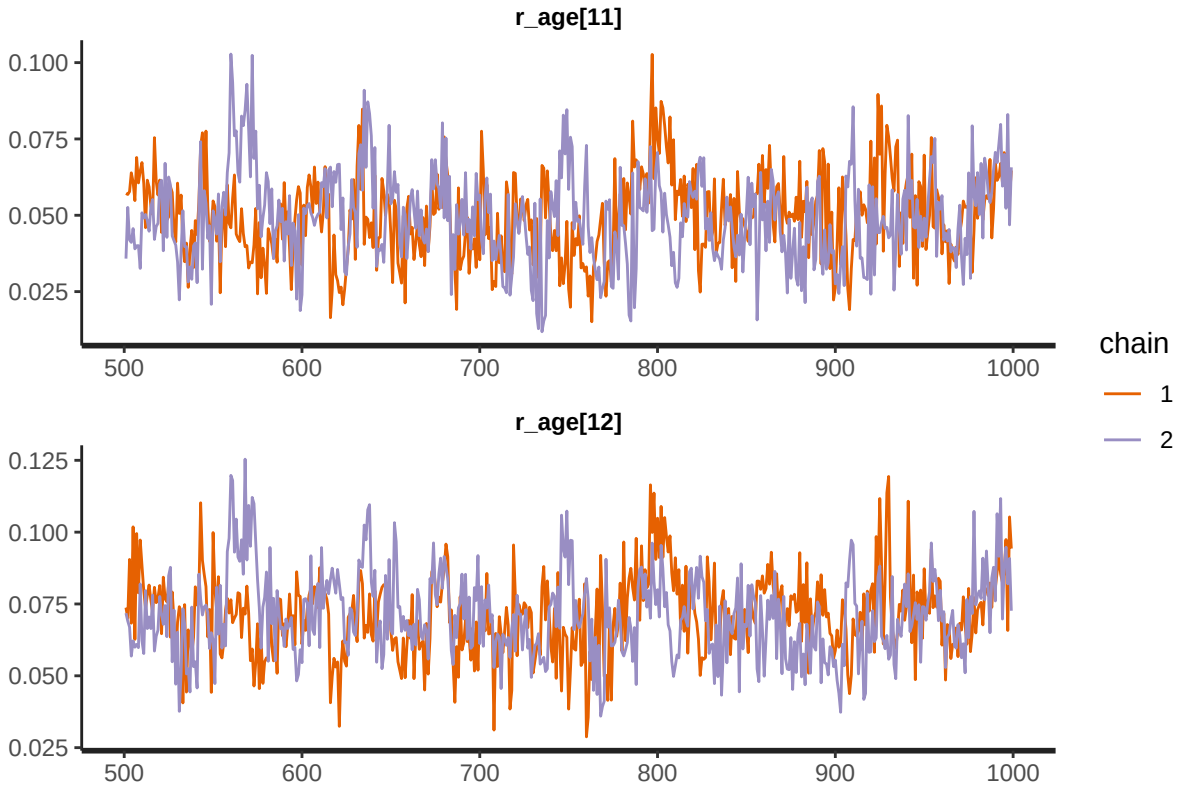
```
traceplot(fit3, pars = c(paras[9],  paras[10]), inc_warmup = FALSE, nrow = 2)
```



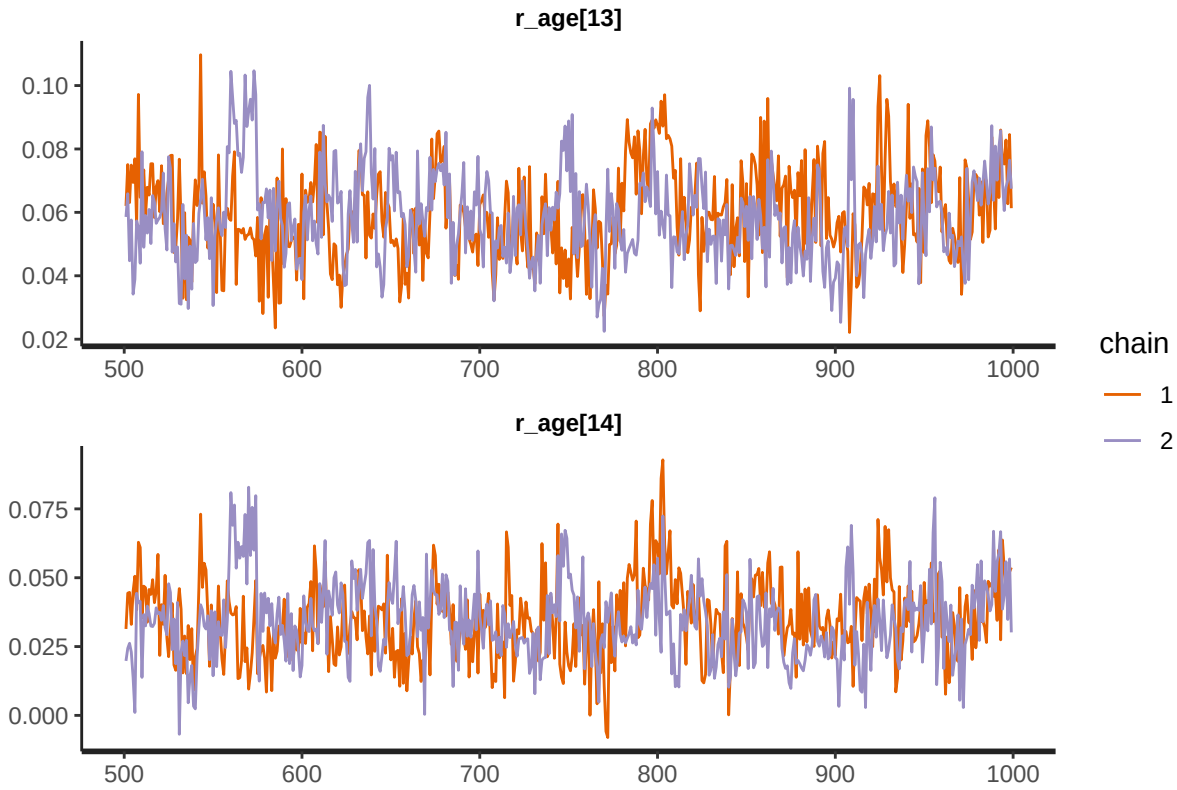

```
traceplot(fit3, pars = c(paras[11], paras[12]), inc_warmup = FALSE, nrow = 2)
```



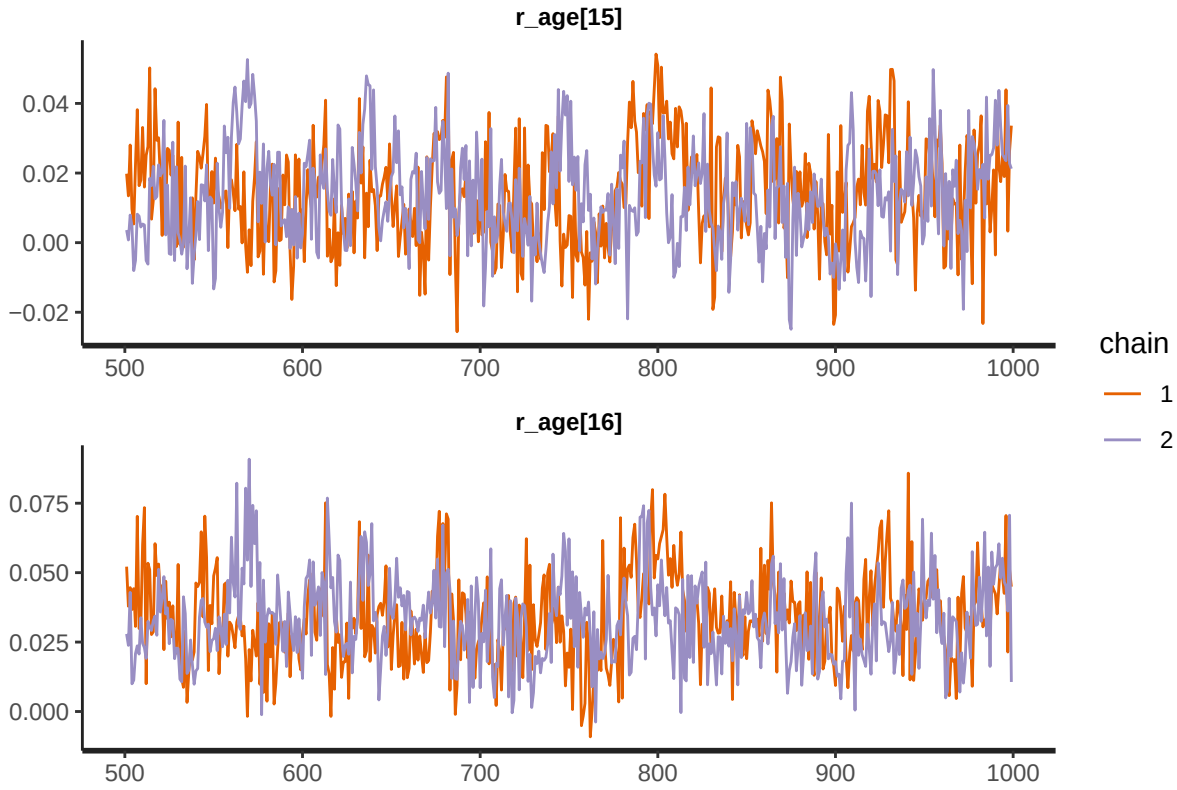
```
traceplot(fit3, pars = c(paras[13], paras[14]), inc_warmup = FALSE, nrow = 2)
```



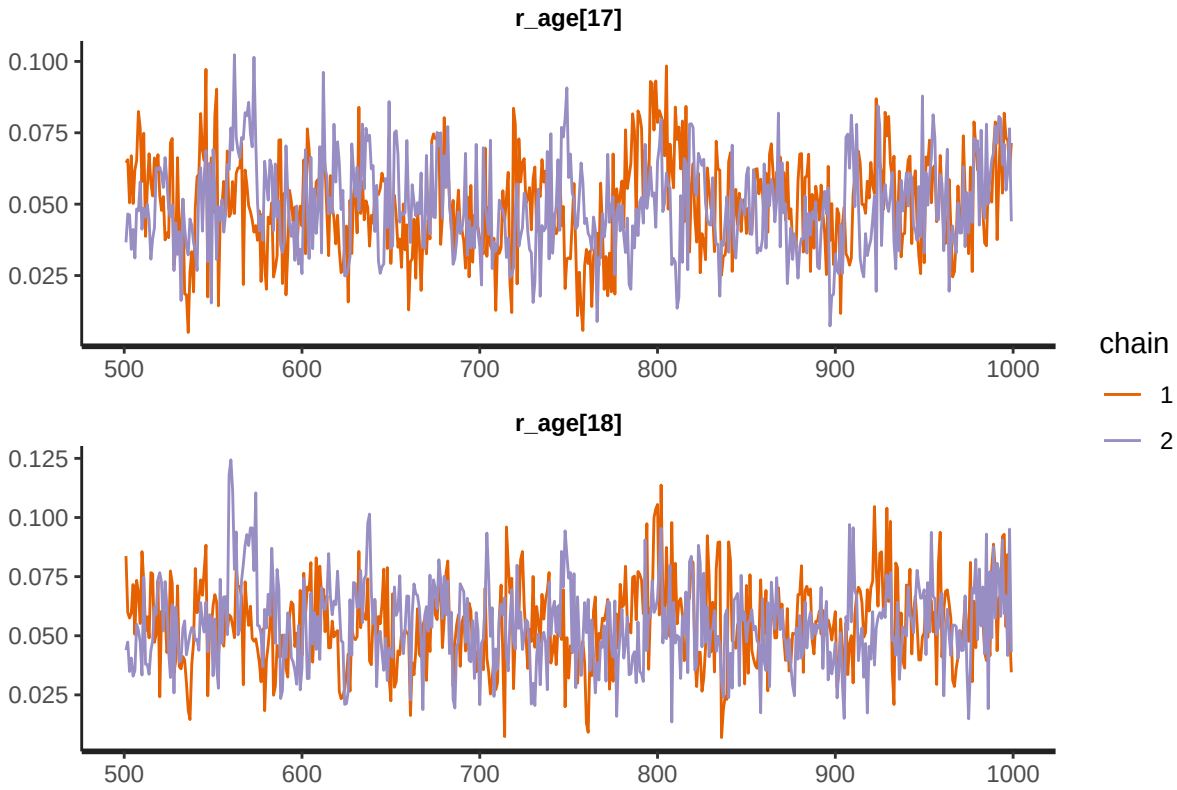

```
traceplot(fit3, pars = c(paras[15], paras[16]), inc_warmup = FALSE, nrow = 2)
```



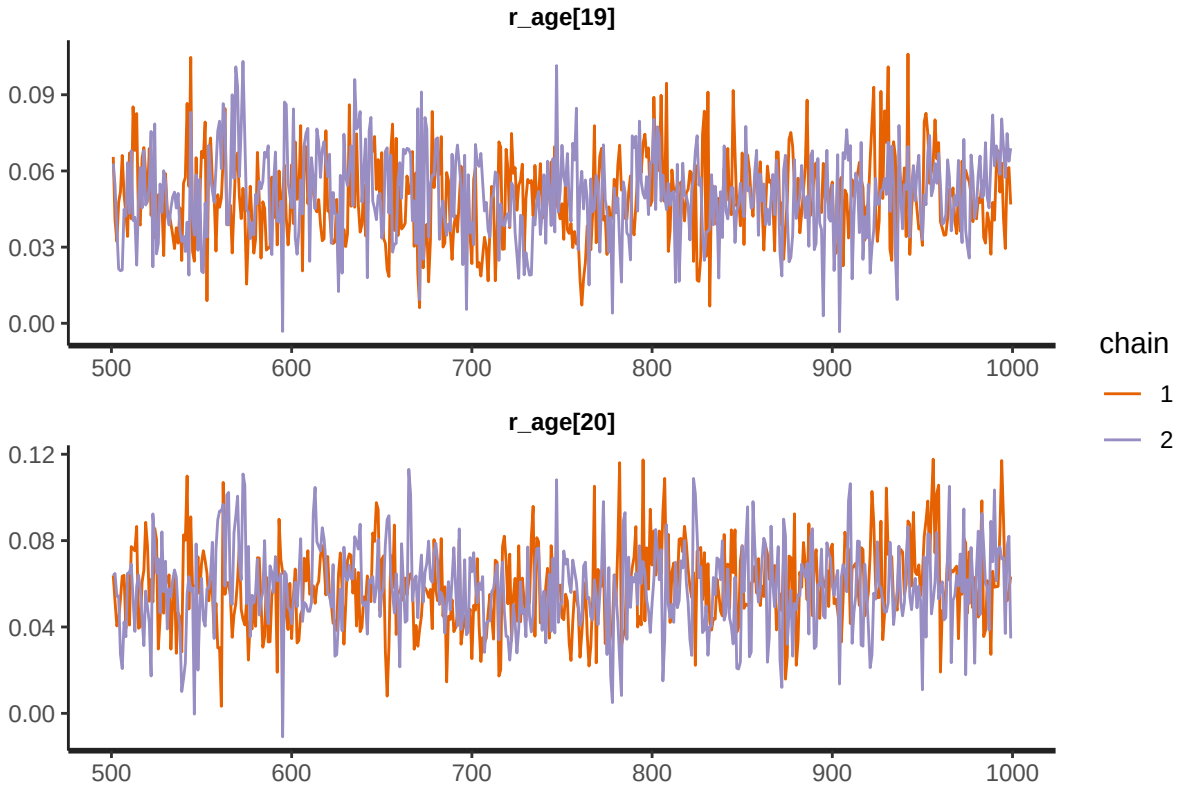
```
traceplot(fit3, pars = c(paras[17], paras[18]), inc_warmup = FALSE, nrow = 2)
```



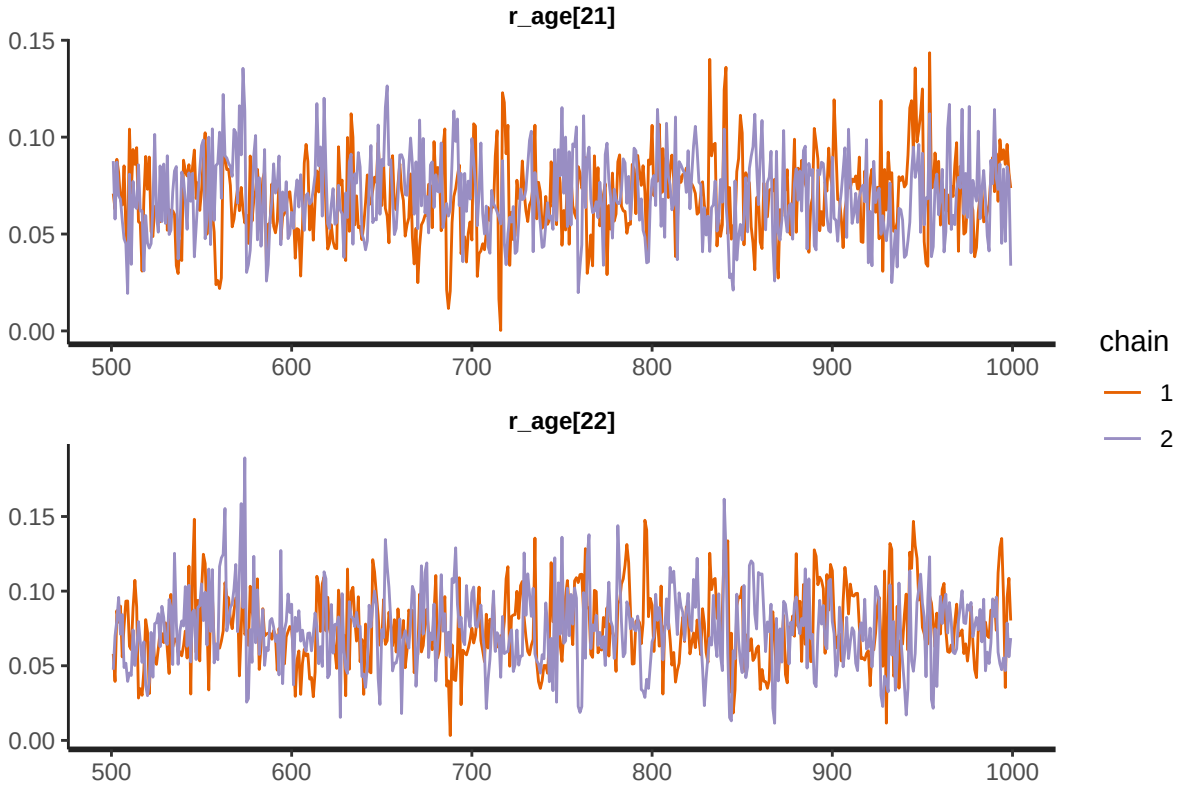

```
traceplot(fit3, pars = c(paras[19], paras[20]), inc_warmup = FALSE, nrow = 2)
```



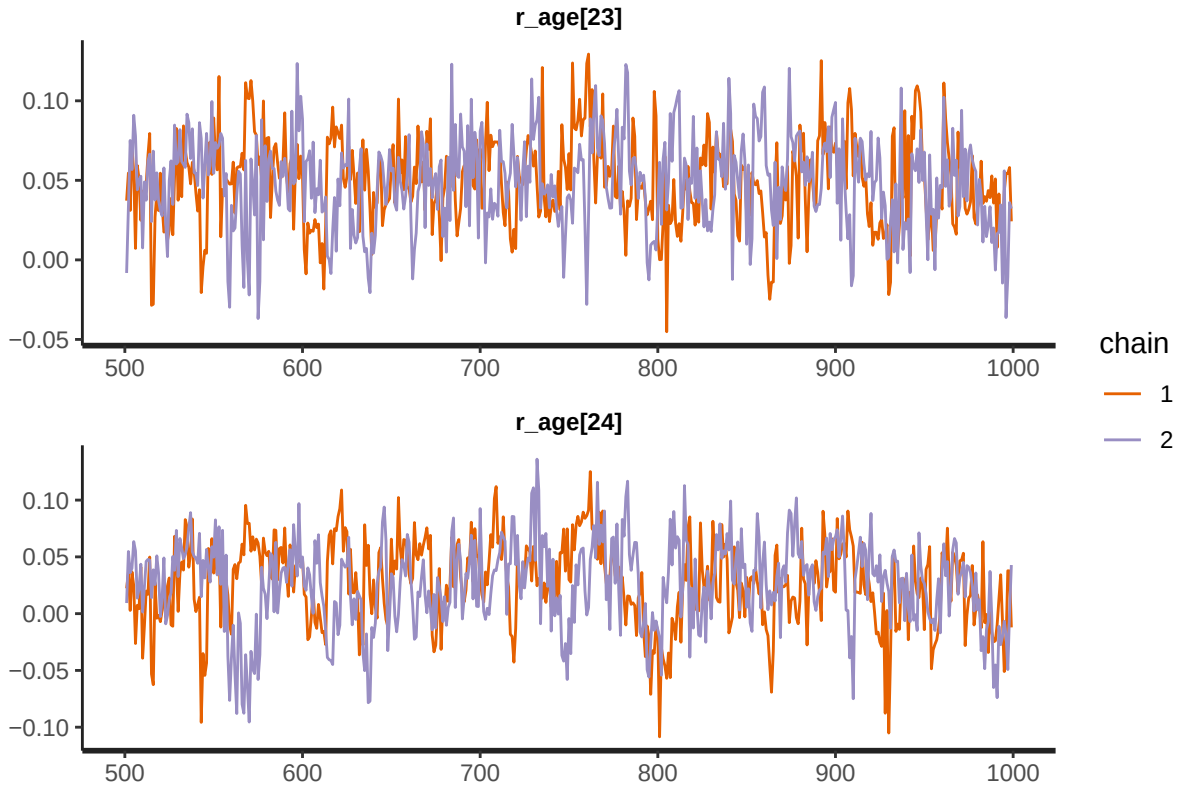
```
traceplot(fit3, pars = c(paras[21], paras[22]), inc_warmup = FALSE, nrow = 2)
```



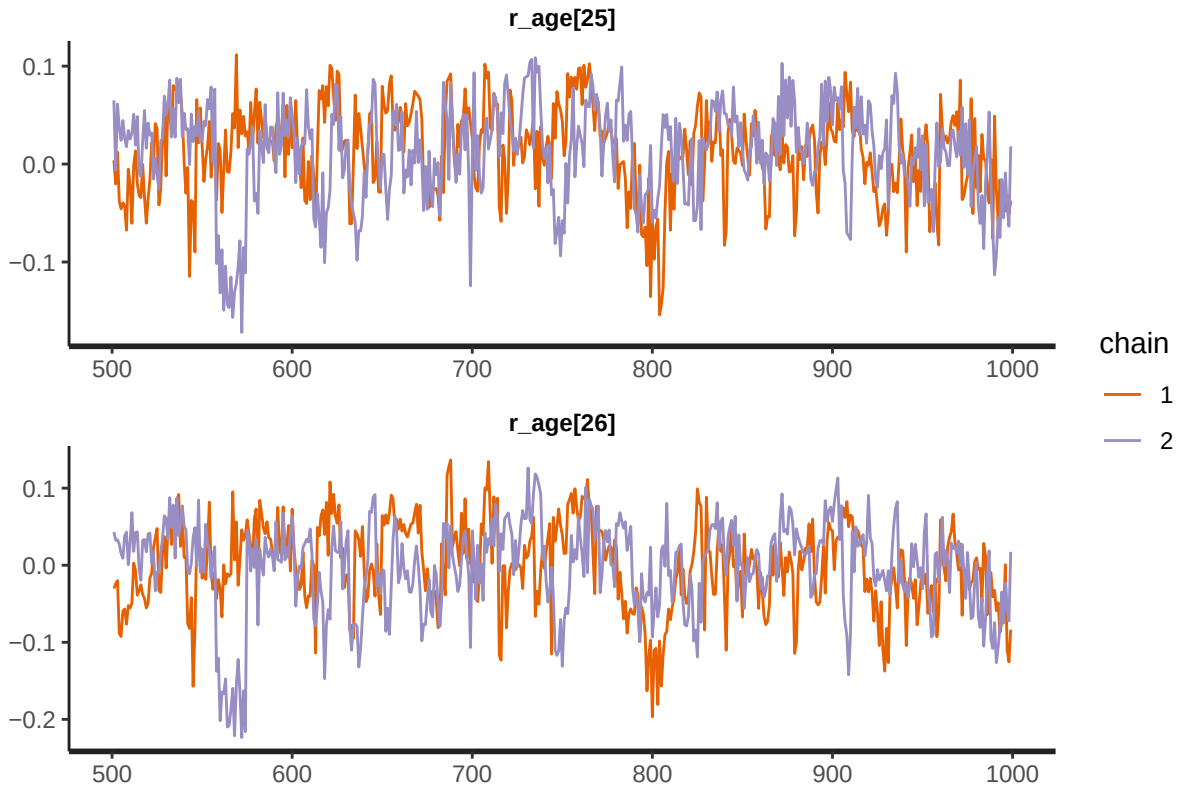

```
traceplot(fit3, pars = c(paras[23], paras[24]), inc_warmup = FALSE, nrow = 2)
```



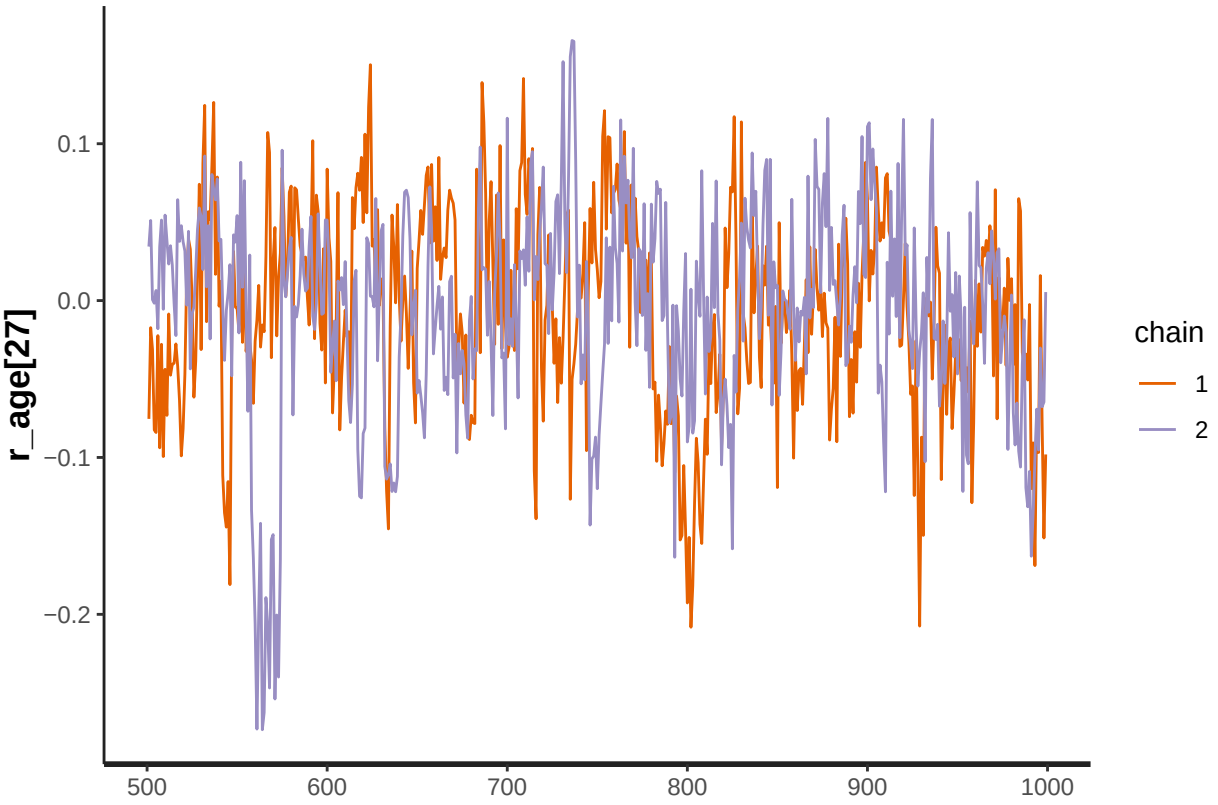
```
traceplot(fit3, pars = c(paras[25], paras[26]), inc_warmup = FALSE, nrow = 2)
```




```
traceplot(fit3, pars = c(paras[27], paras[28]), inc_warmup = FALSE, nrow = 2)
```

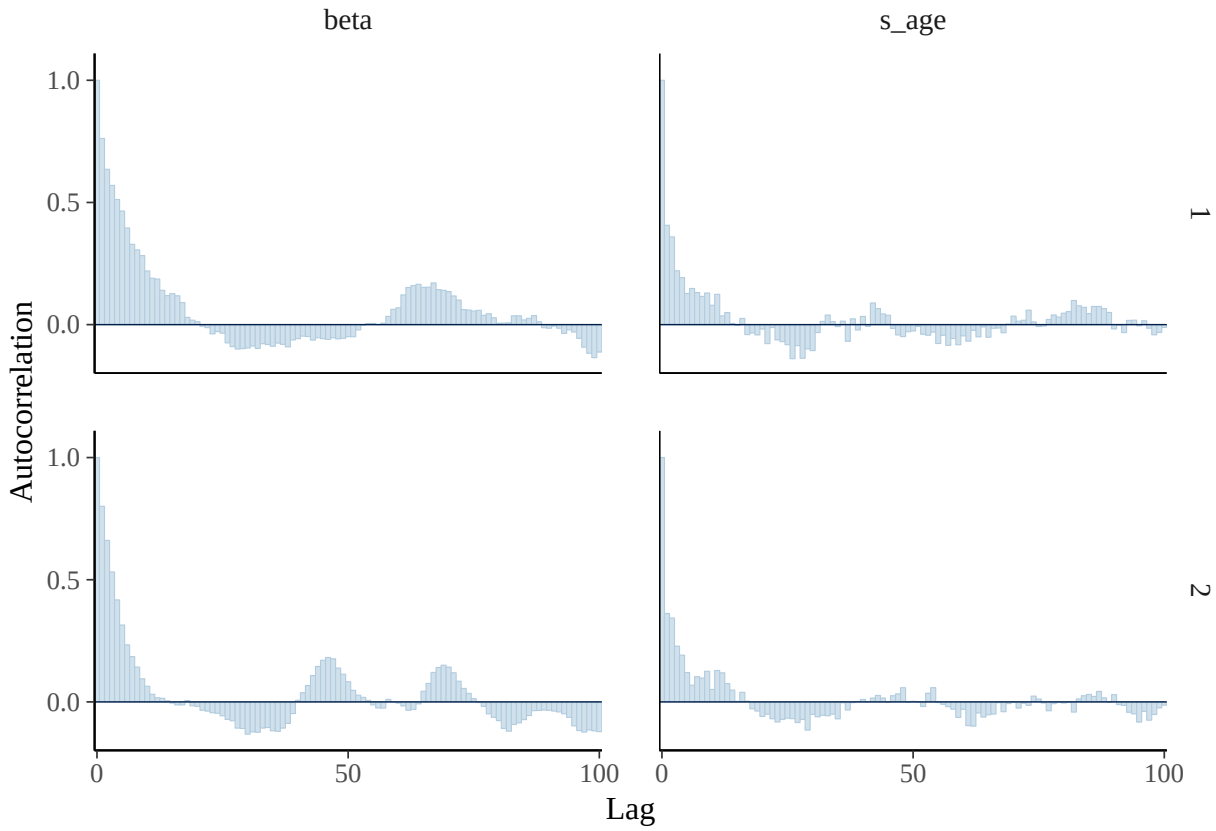


```
traceplot(fit3, pars = paras[29],                inc_warmup = FALSE, nrow = 2)
```

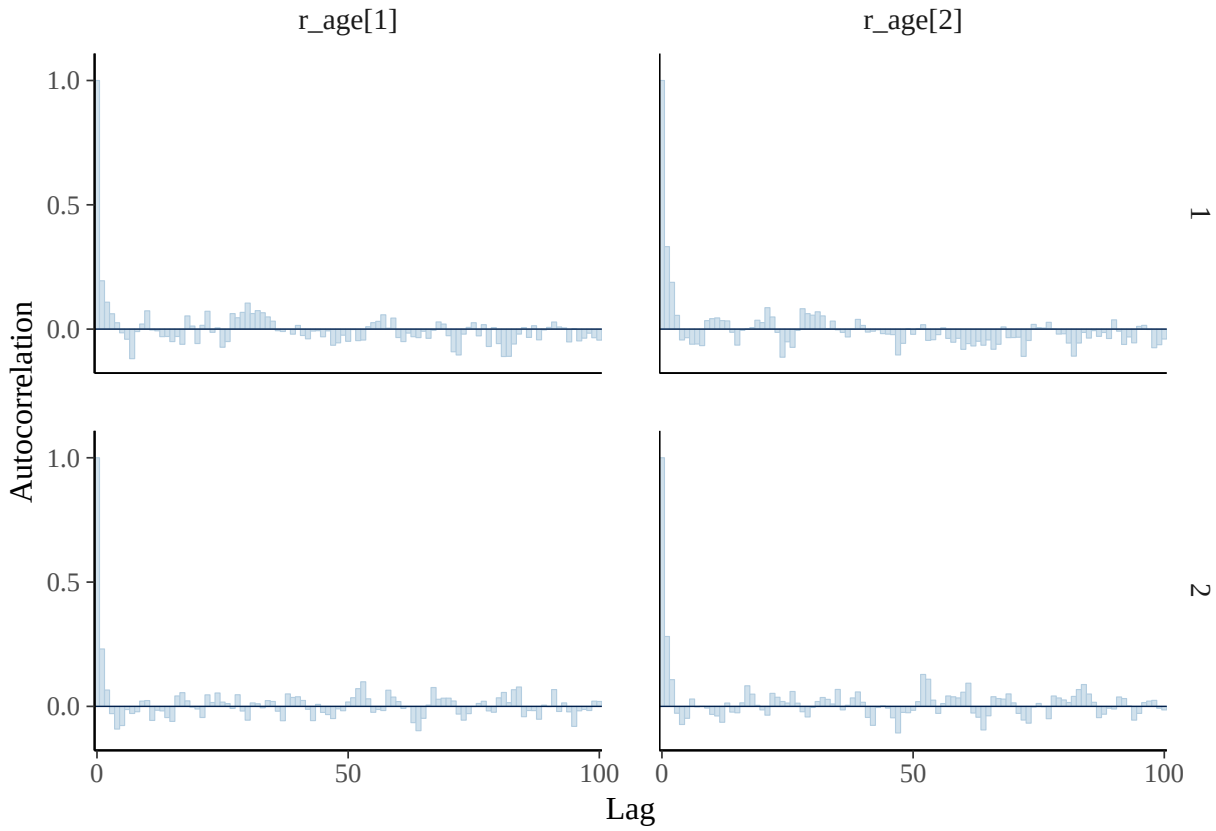


Autocorrelation (model3)

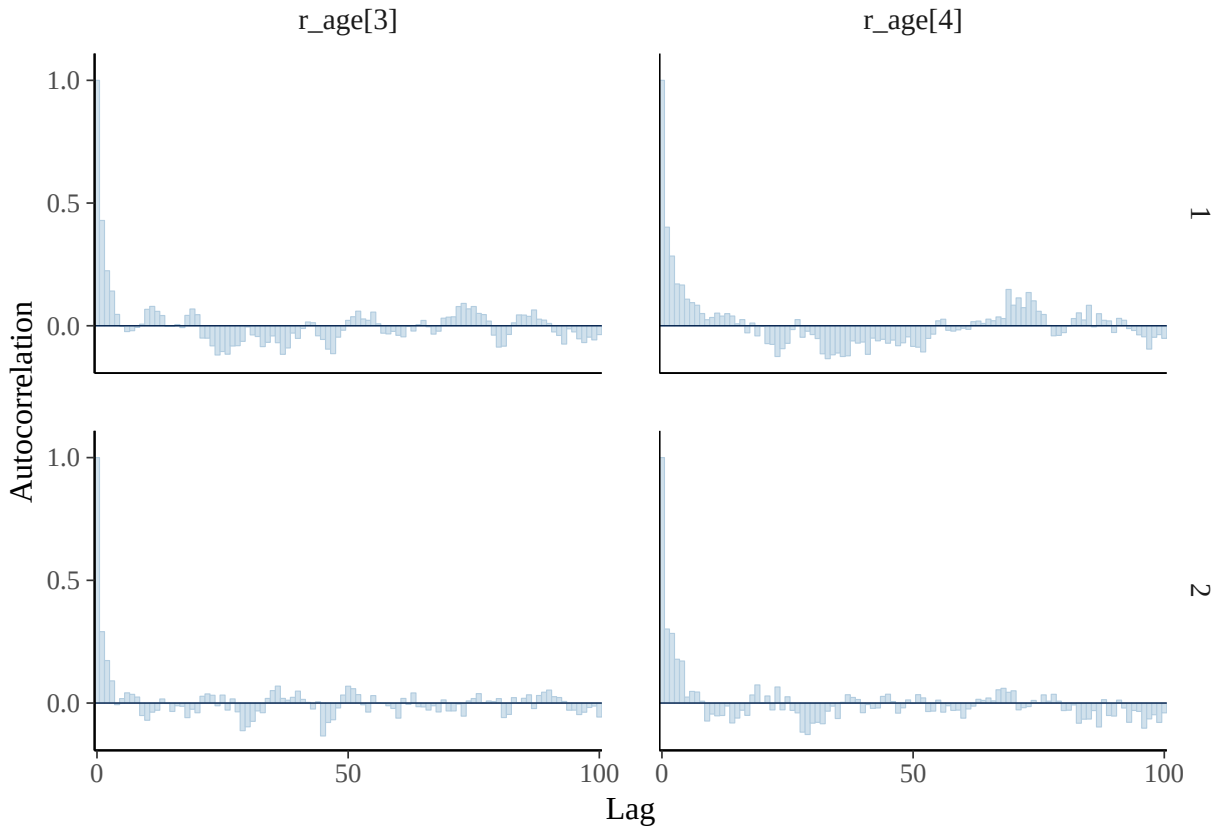
```
mcmc_acf_bar(fit3, pars = c(paras[1],  paras[2]),   lags = num.lag)
```



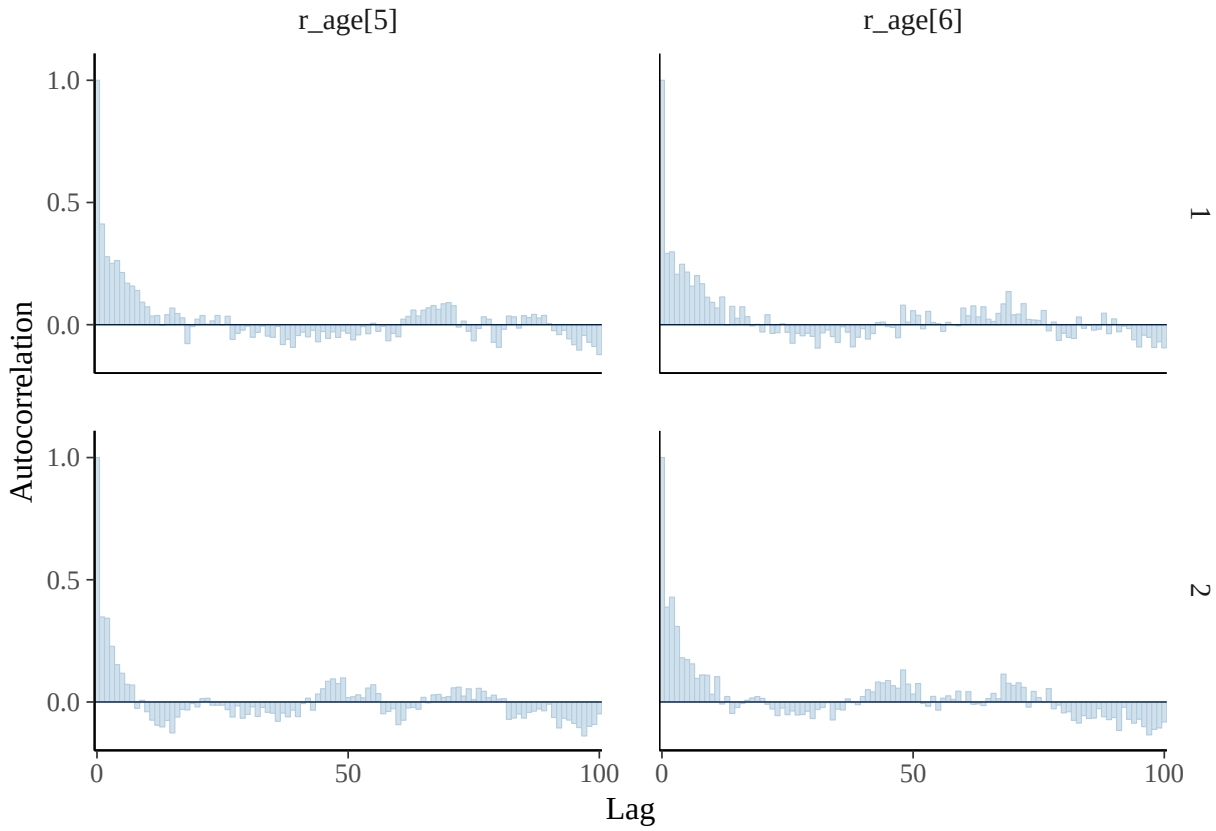

```
mcmc_acf_bar(fit3, pars = c(paras[3],  paras[4]),   lags = num.lag)
```



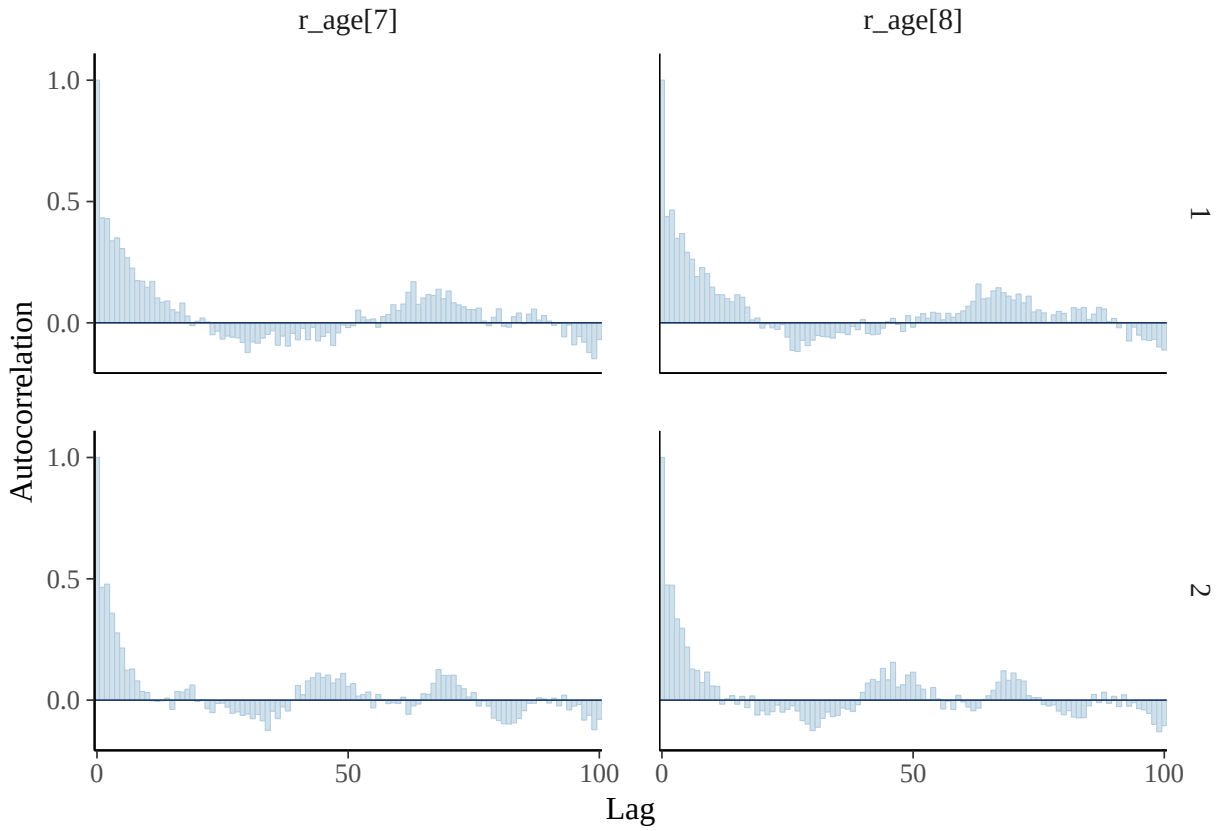

```
mcmc_acf_bar(fit3, pars = c(paras[5],  paras[6]),   lags = num.lag)
```



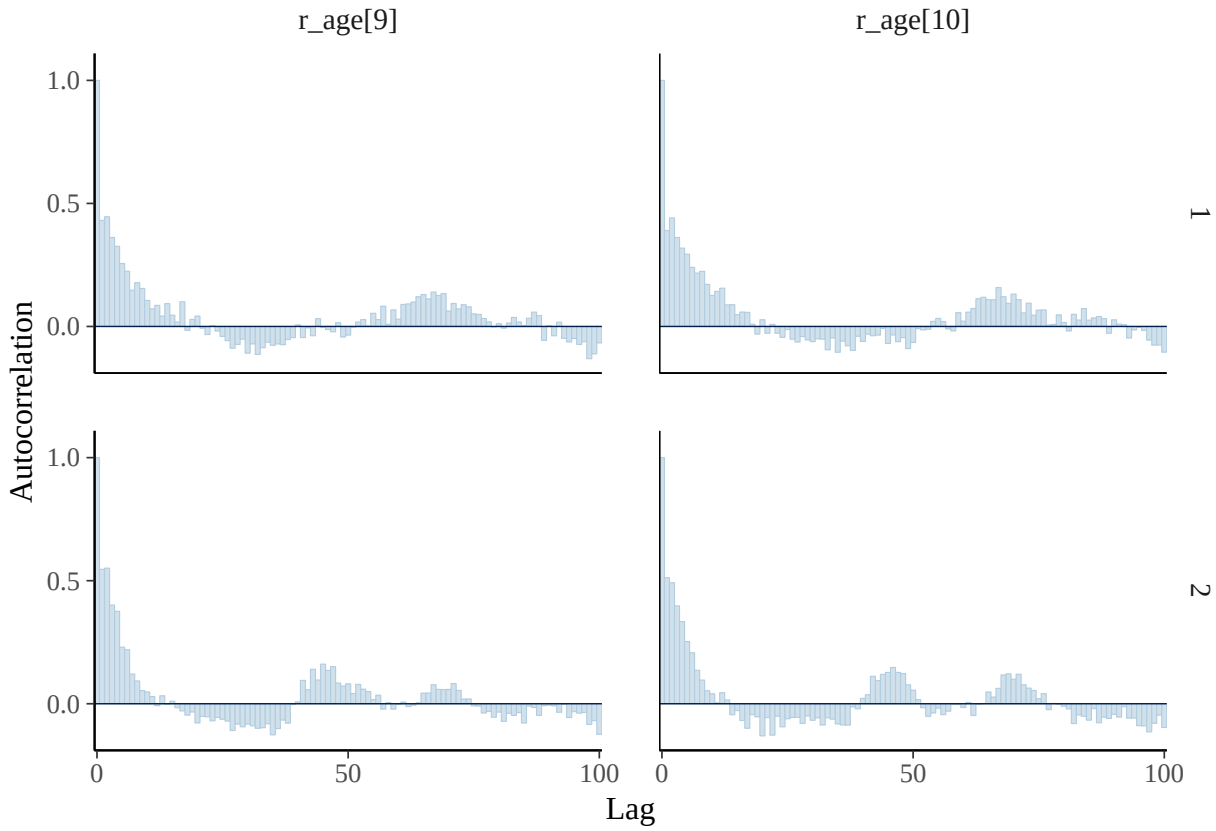

```
mcmc_acf_bar(fit3, pars = c(paras[7],  paras[8]),   lags = num.lag)
```



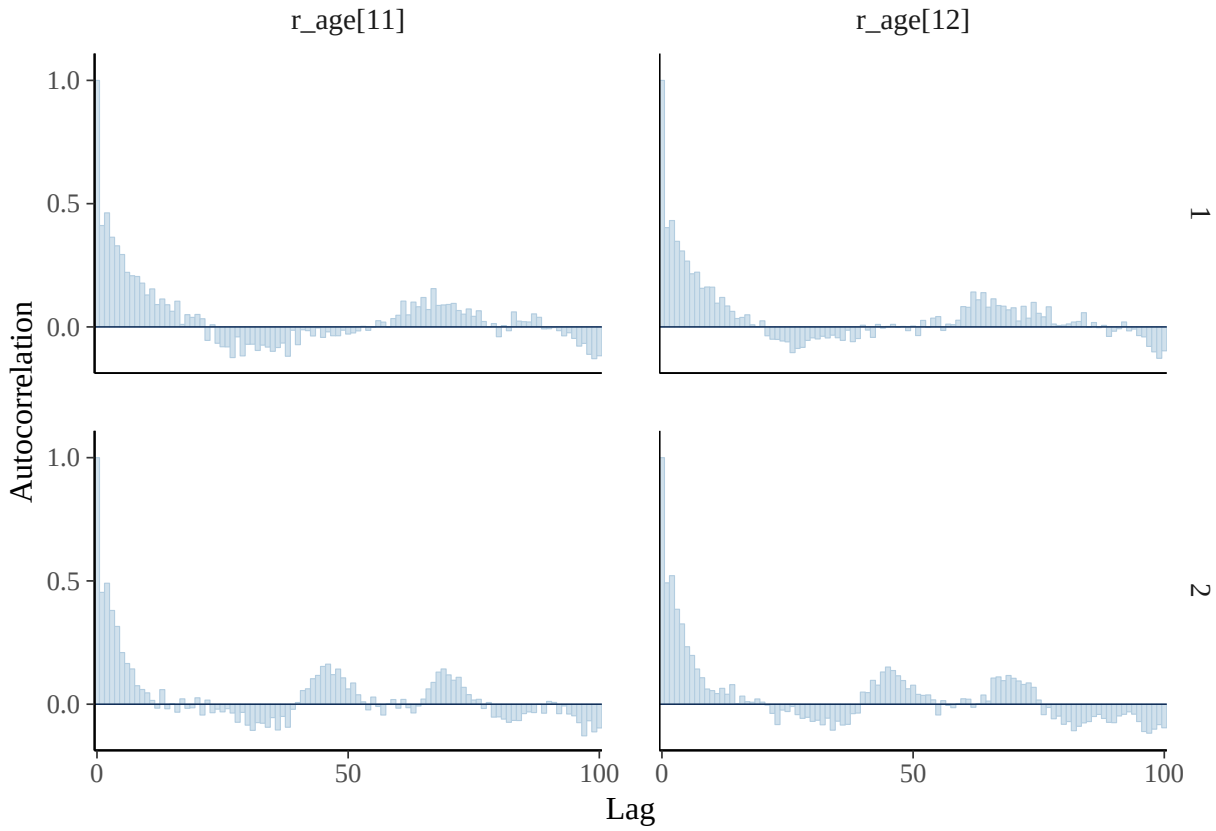

```
mcmc_acf_bar(fit3, pars = c(paras[9],  paras[10]),  lags = num.lag)
```



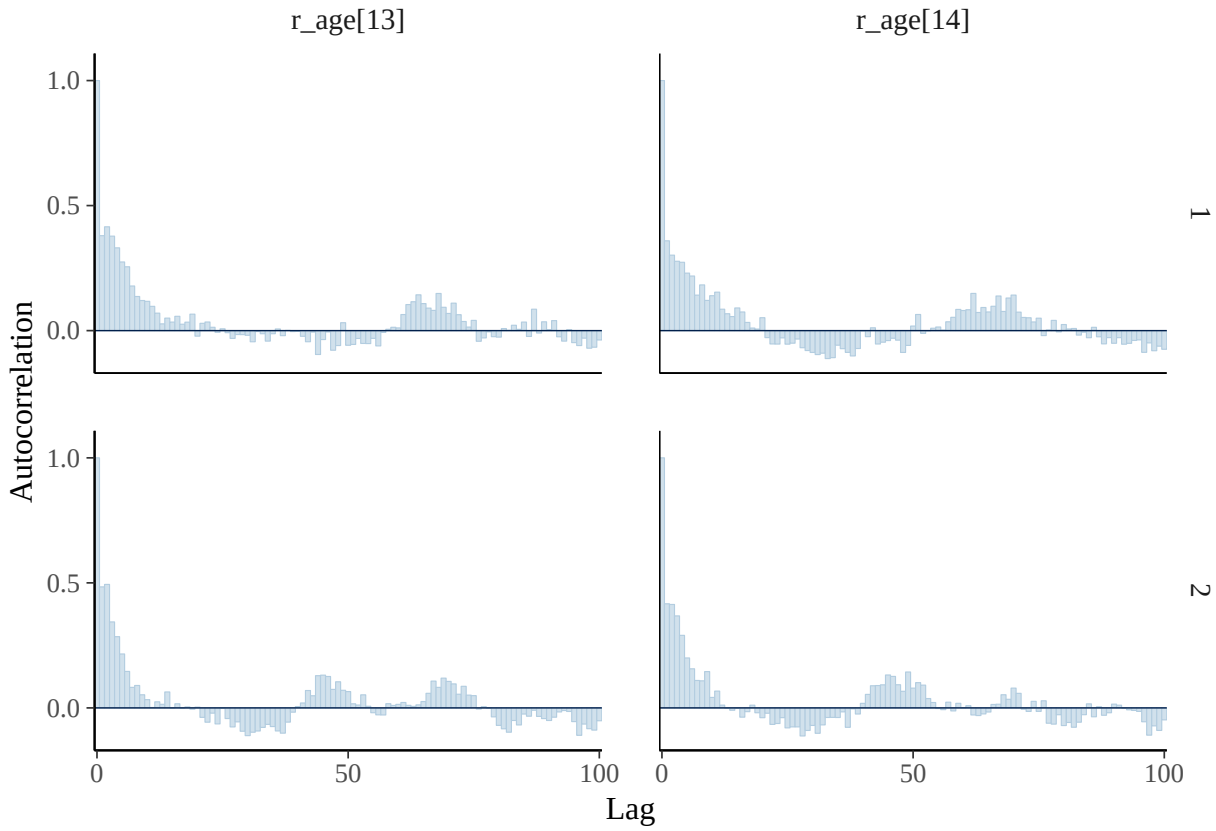

```
mcmc_acf_bar(fit3, pars = c(paras[11], paras[12]),  lags = num.lag)
```



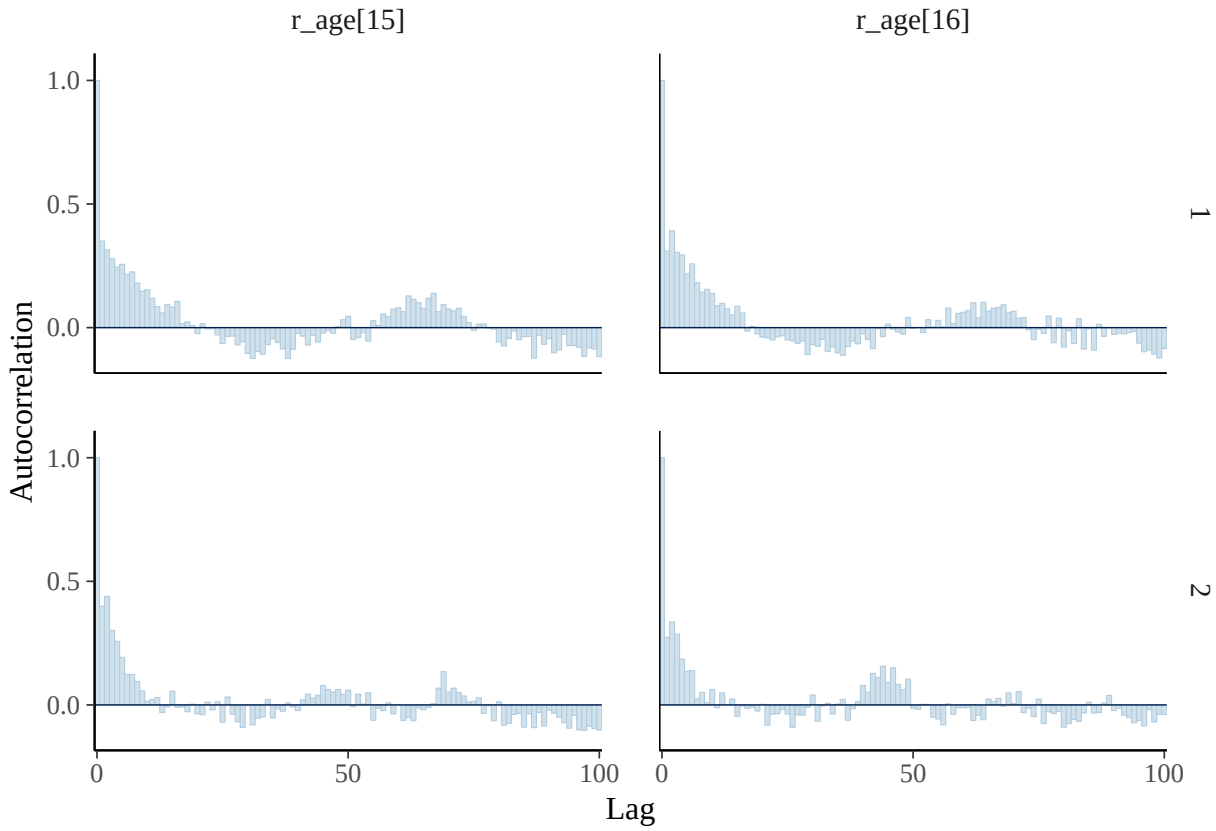

```
mcmc_acf_bar(fit3, pars = c(paras[13], paras[14]),  lags = num.lag)
```



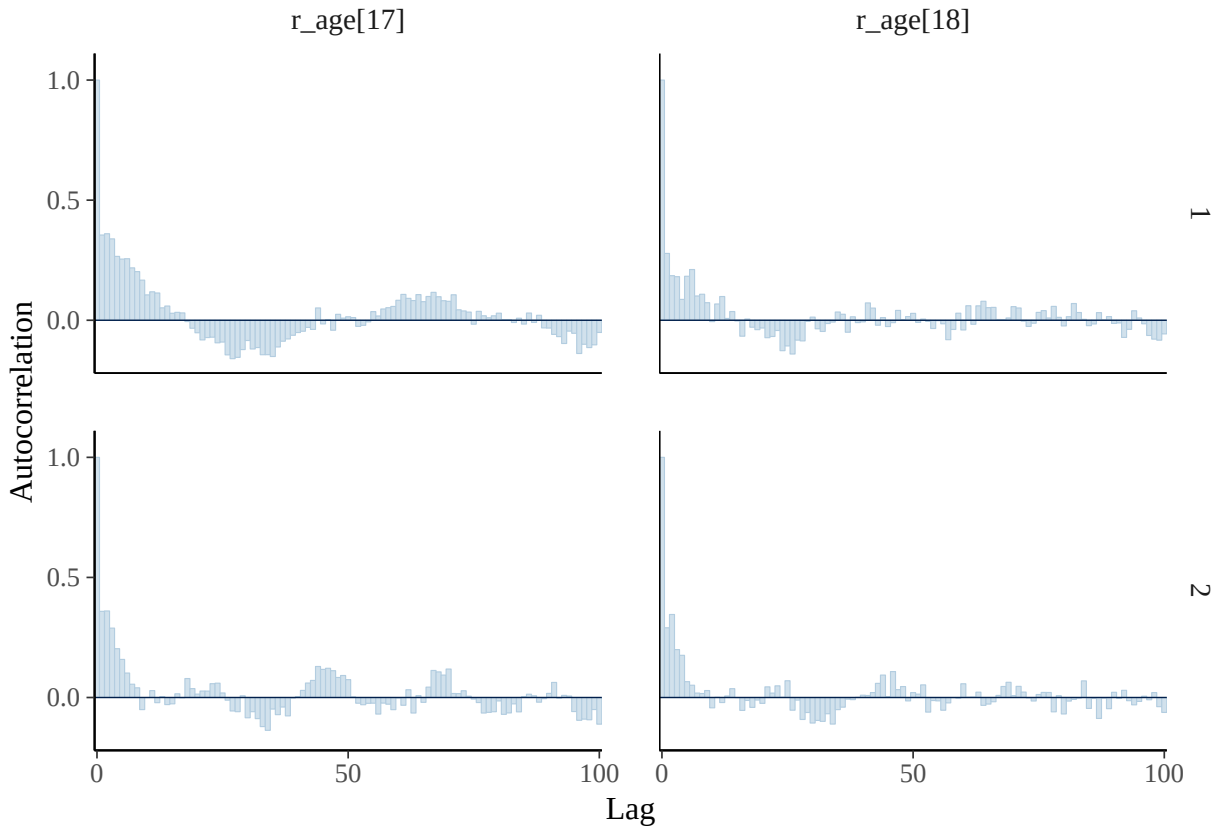

```
mcmc_acf_bar(fit3, pars = c(paras[15], paras[16]),  lags = num.lag)
```



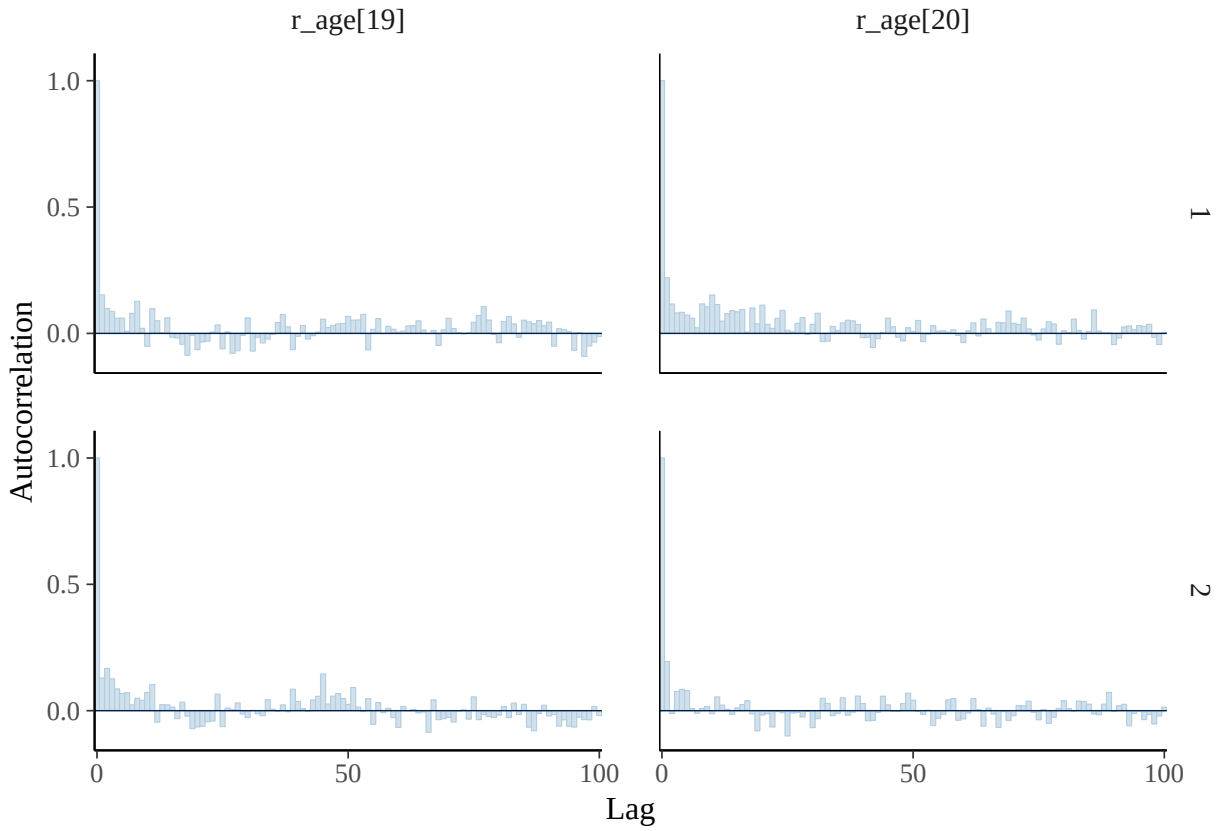

```
mcmc_acf_bar(fit3, pars = c(paras[17], paras[18]),  lags = num.lag)
```



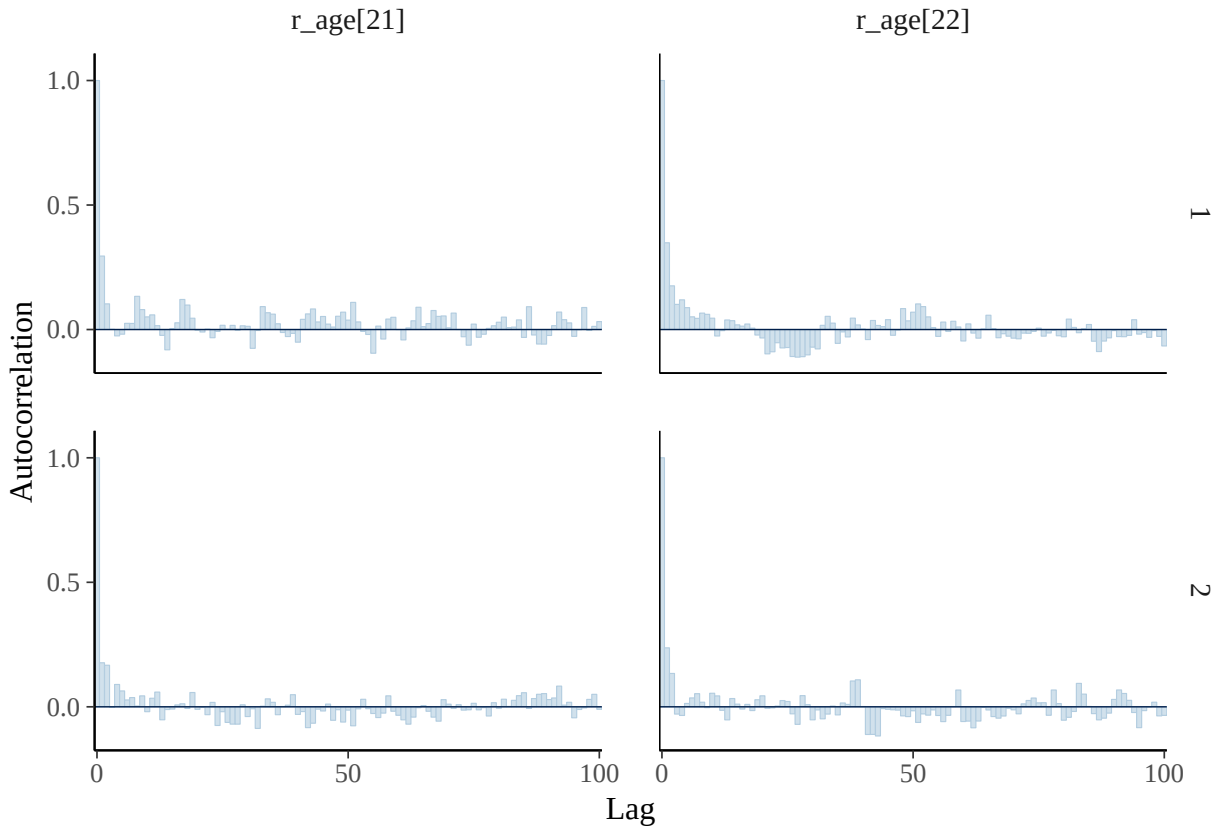

```
mcmc_acf_bar(fit3, pars = c(paras[19], paras[20]),  lags = num.lag)
```



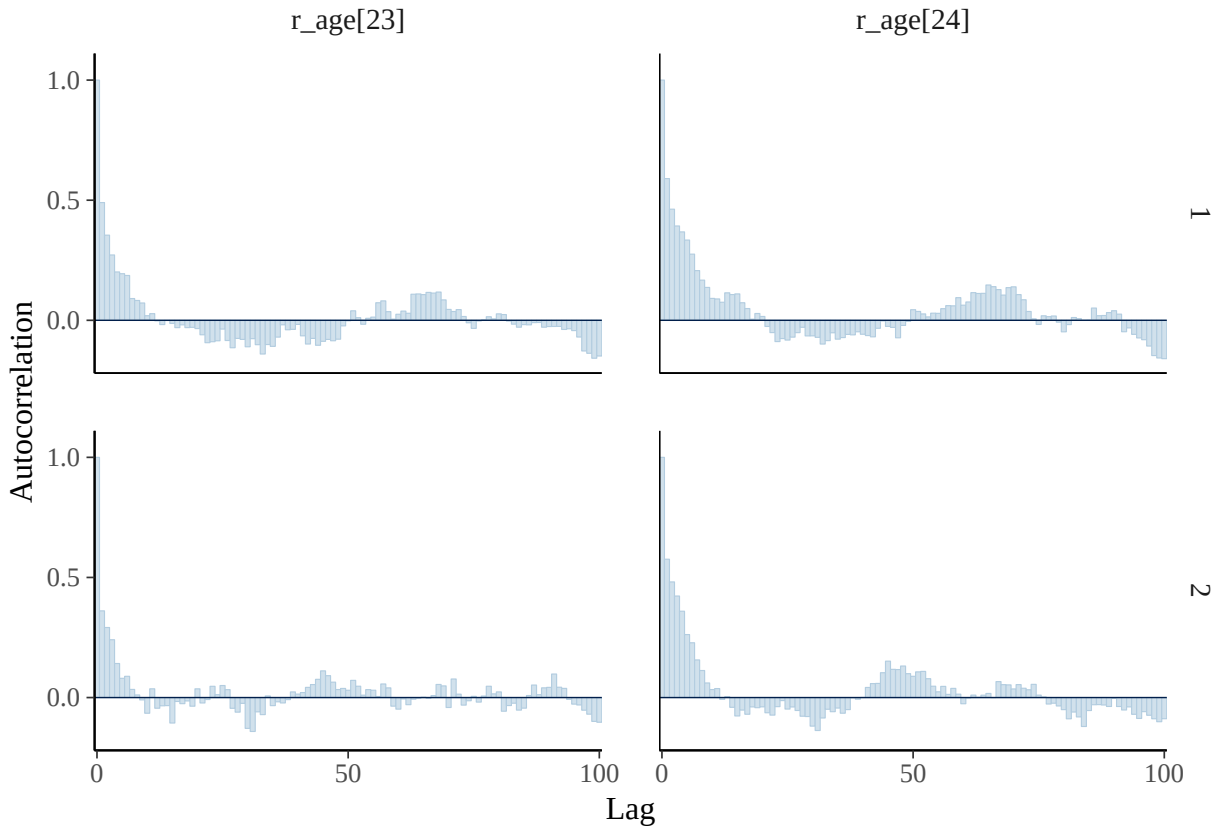

```
mcmc_acf_bar(fit3, pars = c(paras[21], paras[22]),  lags = num.lag)
```



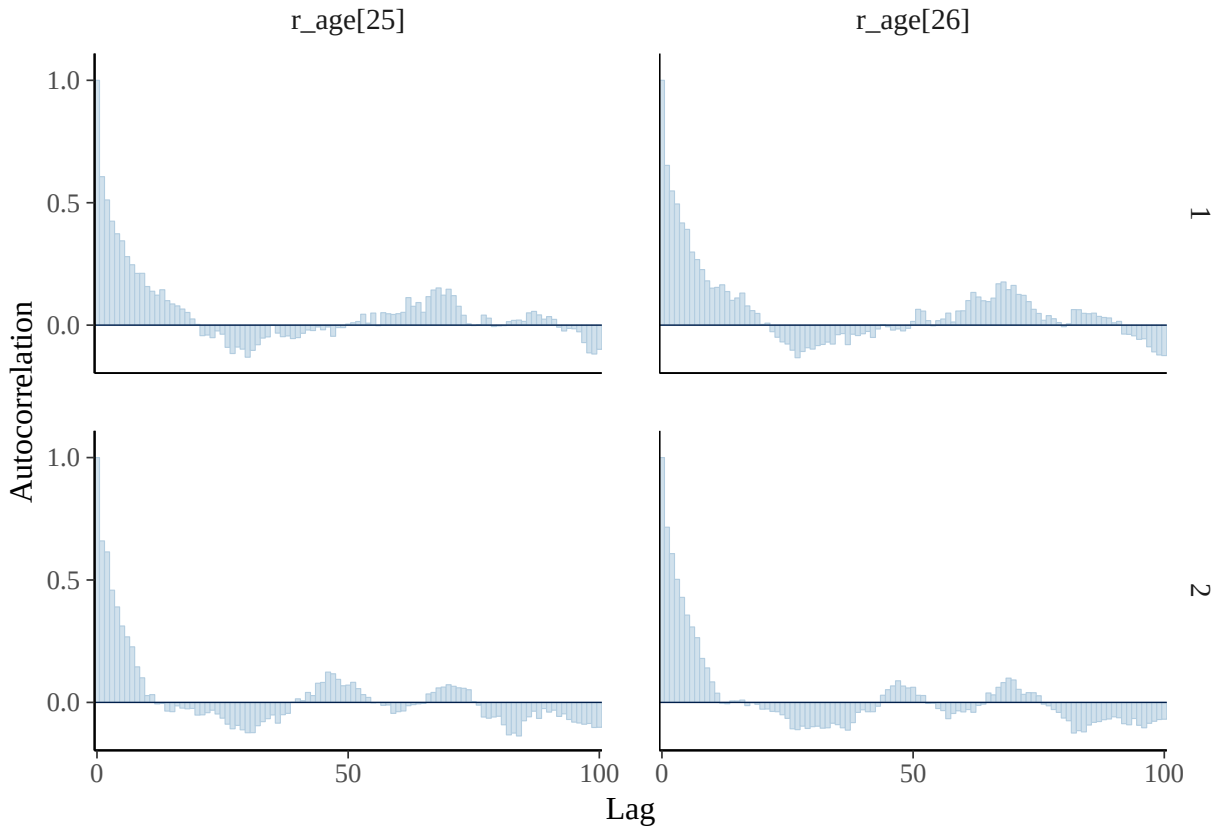

```
mcmc_acf_bar(fit3, pars = c(paras[23], paras[24]),  lags = num.lag)
```



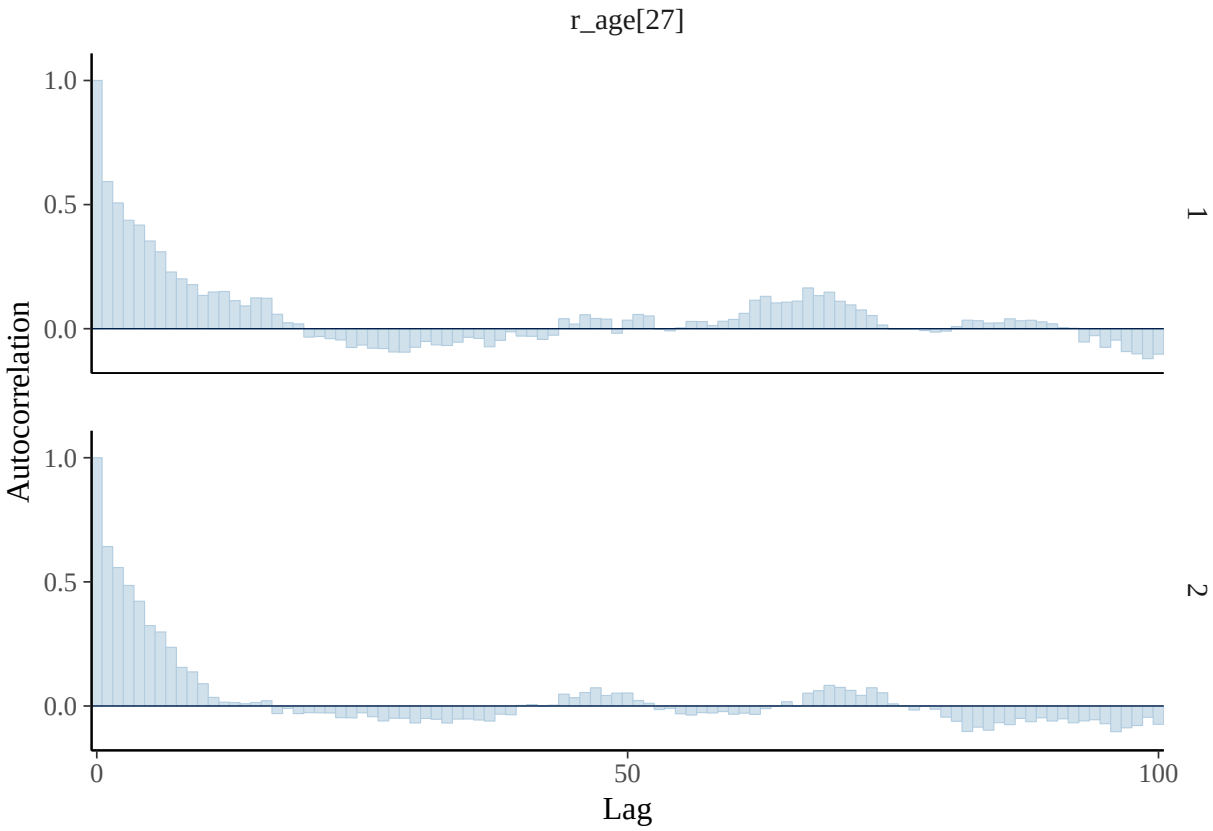

```
mcmc_acf_bar(fit3, pars = c(paras[25], paras[26]),  lags = num.lag)
```




```
mcmc_acf_bar(fit3, pars = c(paras[27], paras[28]),  lags = num.lag)
```




```
mcmc_acf_bar(fit3, pars = paras[29],                lags = num.lag)
```


r_age[27]
Autocorrelation
1.0
0.5
0.0
1
1.0
0.5
0.0
2
0
50
100
Lag