# Simulation and estimation of partition models - Example script with random data

Marion Hoffman

28/09/2020

## 0. Load dependencies

The following packages and scripts are required to fully run this example script. The details of the main functions can be found in the code itself, where comments help finding out what functions do and what arguments are for.

```
library(ERPM)

library(numbers) # for calculations of Bell numbers
library(gmp) # for calculations of Stirling numbers
library(mclust) # for calculations of Rand indexes

library(ggplot2) # for plots
theme_set(theme_minimal())
```

## 1. Make some general calculations

### 1.1 Calculate the size of the space of partitions

For reasonable values of n, we can calculate exactly the total number of possible partitions (this is the Bell number), for example with n = 6,

```
n <- 6
bell(n)
```

```
## [1] 203
```

or the number of partitions with k = 2 groups for example,

```
k <- 2
Stirling2(n,k)
```

```
## Big Integer ('bigz') :
## [1] 31
```

or the number of partitions with groups of size comprised bewteen two values size_min and size_max,

```
size_min <- 2
size_max <- 4
Bell_constraints(n,size_min,size_max)
```

```
## [1] 40
```

or the number of partitions with k = 2 groups and groups of size comprised bewteen two values size_min and size_max.

```
Stirling2_constraints(n,k,size_min,size_max)
```

```
## [1] 25
```

### 1.2 Calculate the expected size of a random partition

We can also compute the average size (i.e., the number of groups) of a partition (under a null model),

```
compute_averagesize(n)
```

```
## [1] 3.293727
```

### 1.3 Enumerate all partitions

For a low number of nodes (below 10 for example), one could enumerate all possible partitions,

```
all_partitions <- find_all_partitions(n)
```

and look at the number of partitions with a certain size structure (for example, how many partitions have 2 groups of 3, or 6 groups of 1?).

```
counts_partition_classes <- count_classes(all_partitions)
```

### 1.4 Calculate a distance measure between two partitions

One can use the Rand distance to evaluate the distance between two partitions, for example:

## 2. Simulate partitions

### 2.1 Define nodesets and attributes

Here we define an arbitrary set of n = 6 nodes with attributes, and an arbitrary covariate matrix.

```
n <- 6
nodes <- data.frame(label = c("A","B","C","D","E","F"),
                    gender = c(1,1,2,1,2,2),
                    age = c(20,22,25,30,30,31))
friendship <- matrix(c(0, 1, 1, 1, 0, 0,
                       1, 0, 0, 0, 1, 0,
                       1, 0, 0, 0, 1, 0,
                       1, 0, 0, 0, 0, 0,
                       0, 1, 1, 0, 0, 1,
                       0, 0, 0, 0, 1, 0), 6, 6, TRUE)
```

### 2.2 Define a model and simulate

First, we need to choose the effects we want to include (see manual for all effect names). For example we set four (which is of course not reasonable for 6 nodes):

```
effects <- list(names = c("num_groups","same","diff","tie"),
                   objects = c("partition","gender","age","friendship"))
objects <- list()
objects[[1]] <- list(name = "friendship", object = friendship)
```

and we can set parameter values for each of these effects.

```
parameters <- c(-0.2,0.2,-0.1,0.5)
```

Now we can generate our simulated sample, by setting the desired additional parameters for the Metropolis sampler and choosing a starting point for the chain (first.partition):

```
nsteps <- 100
sample <- draw_Metropolis_single(theta = parameters,
                          first.partition = c(1,1,2,2,3,3),
                          nodes = nodes,
                          effects = effects,
                          objects = objects,
                          burnin = 100,
                          thining = 10,
                          num.steps = nsteps,
                          neighborhood = c(0,1,0),
                          numgroups.allowed = 1:n,
                          numgroups.simulated = 1:n,
                          sizes.allowed = 1:n,
                          sizes.simulated = 1:n,
                          return.all.partitions = T)
```

### 2.3 Trace plots

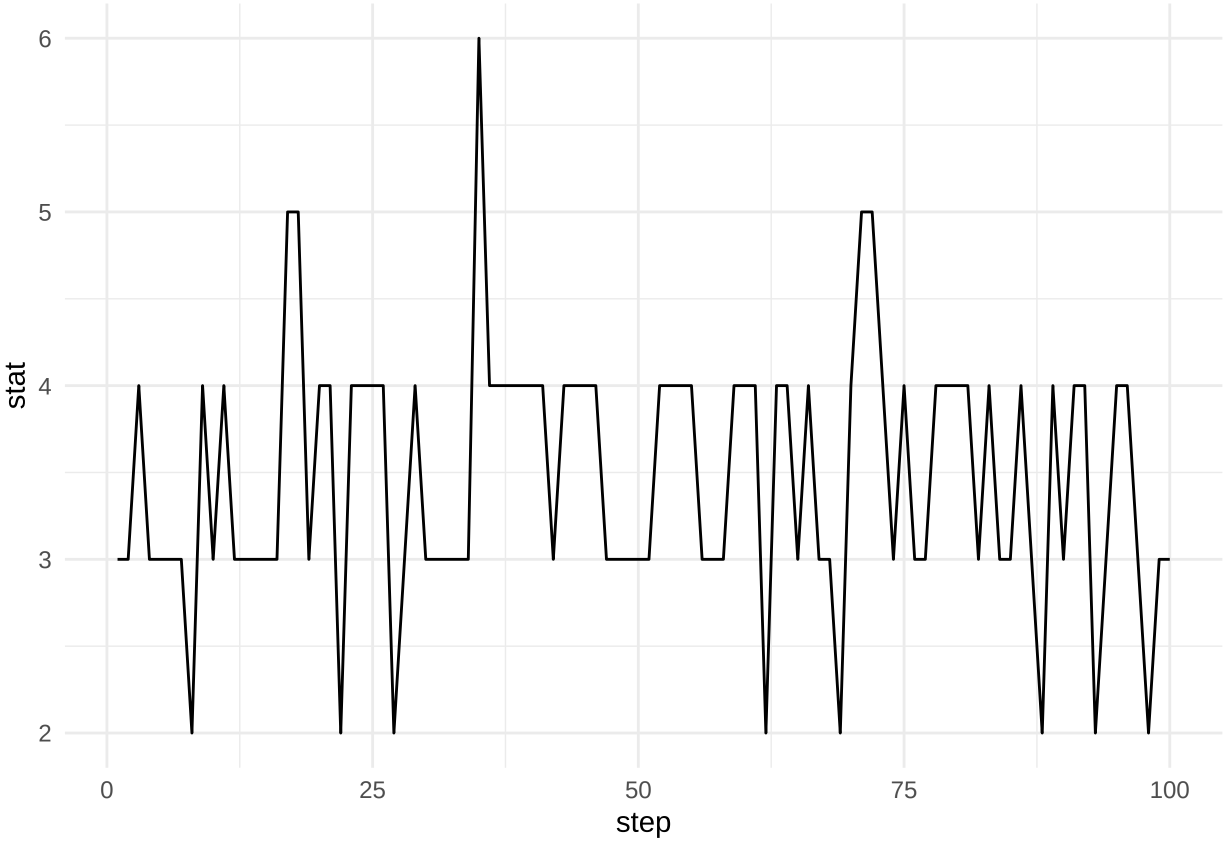
We can check the mixing of the chain with autocorrelations (here it's not good enough yet):

```
s <- 1
cor(sample$draws[1:(nsteps-1),s], sample$draws[2:nsteps,s]) # should be below 0.4!
```

```
## [1] 0.1222756
```

and with the trace plots;

```
s <- 1
ggplot(data = data.frame(step = 1:nsteps,
                         stat = sample$draws[,s])) +
  geom_line(aes(x=step,y=stat))
```



## 3. Estimate for an observed partition

### 3.1 Define the observation

```
partition <- c(1,1,2,2,2,3)
```

### 3.2 Estimate

The average number of groups expected at random is lower than 3 (see section 1), so let's set an initial estimate for the number of groups parameter negative, and leave the others to zero. The burnin and thining are chosen from trace plots to have a good mixing. For phase 1, we don't need a large sample, because it is just for the scaling matrix which is not very sensitive to the number of samples. For phase 2, we have to define a number of subphases and the number of iterations in the subphases. There will be (2.52)^k iterations in each subphase k, multiplied by a constant multiplication factor, that we need to choose big enough so that statistics are crossed. When approaching the correct estimates, we can increasingly reduce the number of steps of phase 2. For phase 3, we need a lot of samples to have correct estimates of the final distribution. When we approach the right estimates, it is the most important phase, and its results should be re-used to restart the estimation from a good starting point. The multiplication factor and gain factor were chosen as in the ERGM book, but they can be adjusted by looking at the evolution of parameters in phase 2.

```
startingestimates <- c(-2,0,0,0)
estimation <- estimate_ERPM(partition,
                          nodes,
                          objects,
                          effects,
                          startingestimates = startingestimates,
                          burnin = 100,
                          thining = 20,
                          length.p1 = 500, # number of samples in phase 1
                          multiplication.iter.p2 = 20,  # multiplication factor for the number of itera
                          num.steps.p2 = 4, # number of phase 2 subphases
                          length.p3 = 1000) # number of samples in phase 3
```

```
## Observed statistics

## 32122

## Burn-in

## 100

## Thining

## 20

## [1] "Autocorrelations in phase 1:"
## [1]  0.007908988 -0.038374075 -0.020415395 -0.048044405
## [1] "Covariance matrix"
##             [,1]       [,2]      [,3]       [,4]
## [1,]   0.4605852 -0.8334269 -11.77437 -0.8419399
## [2,]  -0.8334269  2.7198397  27.75166  2.2936273
## [3,] -11.7743727 27.7516633 446.82554 27.4317595
## [4,]  -0.8419399  2.2936273  27.43176  2.7368978
## [1] "Invert scaling matrix"
##            [,1]        [,2]         [,3]        [,4]
## [1,] 4.41004803  0.35780742  0.055539258  0.40009645
## [2,] 0.35780742  0.80867774 -0.017048721 -0.31740361
## [3,] 0.05553926 -0.01704872  0.004987327 -0.01489191
## [4,] 0.40009645 -0.31740361 -0.014891908  0.79604696
## [1] "Estimated statistics after phase 1"
## [1]  2.156  2.960 40.954  2.976
## [1] "Estimates after phase 1"
##             [,1]
## [1,] -0.61999241
## [2,]  0.32907343
## [3,] -0.06662667
## [4,]  0.29662734
## Difference to estimated statistics after phase 2, step 1NULL
## [1] 0.04347826 0.08695652 5.65217391 0.00000000
## Estimates after phase 2, step 1NULL
##
## [1,] -0.68010480
## [2,]  0.42490340
## [3,] -0.09797422
## [4,]  0.40539806
## Difference to estimated statistics after phase 2, step 2NULL
## [1] -0.1153846  0.4423077  5.9423077  0.2307692
## Estimates after phase 2, step 2NULL
##
## [1,] -1.0157451
## [2,]  0.2678798
## [3,] -0.1236522
## [4,]  0.5847800
## Difference to estimated statistics after phase 2, step 3NULL
## [1] -0.0312500  0.1093750  3.2109375  0.1640625
## Estimates after phase 2, step 3NULL
##
## [1,] -1.3382506
## [2,]  0.2951254
## [3,] -0.1439831
## [4,]  0.5314177
## Difference to estimated statistics after phase 2, step 4NULL
## [1] -0.0619195  0.1764706  3.1517028  0.1764706
## Estimates after phase 2, step 4NULL
##
## [1,] -1.3890893
## [2,]  0.2849936
## [3,] -0.1588861
## [4,]  0.5178520
## [1] "Autocorrelations in phase 3:"
## [1] 0.12608535 0.09004815 0.06584050 0.09675212
## [1] "Estimated statistics after phase 3"
## [1]  2.933  2.060 13.555  2.185
## [1] "Estimates after phase 3"
##
## [1,] -1.3890893
## [2,]  0.2849936
## [3,] -0.1588861
## [4,]  0.5178520
```

```
# get results table
estimation
```

```
##       effect     object        est   std.err sig          t        conv
## 1 num_groups  partition -1.3890893 2.0705389     -0.6708830 -0.09282022
## 2       same     gender  0.2849936 1.1911649      0.2392562  0.04959078
## 3       diff        age -0.1588861 0.1335204     -1.1899769  0.16998581
## 4        tie friendship  0.5178520 1.8856636      0.2746259  0.20213901
```

We get some intermediary objects calculated in the different phases

```
objects.phase1 <- estimation$objects.phase1
objects.phase2 <- estimation$objects.phase2
objects.phase3 <- estimation$objects.phase3
```

The convergence should be as small as possible, below 0.1 for example. We can retry to estimate the model starting from the result of phase 3 and using the covariance calculated in phase 3 (then phase 1 is skipped). It can be good to reduce the gain factor, and make the subphases longer, or lengthen phase 3:

```
startingestimates <- estimation$results$est
startingcovariance <- objects.phase3$inv.zcov
startingscaling <- objects.phase3$inv.scaling
estimation <- estimate_ERPM(partition,
                          nodes,
                          objects,
                          effects,
                          startingestimates = startingestimates,
                          gainfactor = 0.05,
                          burnin = 500,
                          thining = 20,
                          length.p1 = 100,
                          multiplication.iter.p2 = 50,
                          num.steps.p2 = 3,
                          length.p3 = 2000,
                          inv.zcov = startingcovariance,
                          inv.scaling = startingscaling)
```

```
## Observed statistics
```

```
## 32122
```

```
## Burn-in
```

```
## 500
```

```
## Thining
```

```
## 20
```

```
## Difference to estimated statistics after phase 2, step 1NULL
## [1] 0.01851852 0.11111111 0.92592593 0.14814815
## Estimates after phase 2, step 1NULL
##
## [1,] -1.4230011
## [2,]  0.2083192
## [3,] -0.1568031
## [4,]  0.4809442
## Difference to estimated statistics after phase 2, step 2NULL
## [1] -0.0234375  0.0234375  1.8984375  0.0781250
## Estimates after phase 2, step 2NULL
##
## [1,] -1.3365718
## [2,]  0.2901097
## [3,] -0.1663249
## [4,]  0.4820529
## Difference to estimated statistics after phase 2, step 3NULL
## [1] -0.012500  0.053125  0.828125  0.081250
## Estimates after phase 2, step 3NULL
##
## [1,] -1.6247329
## [2,]  0.3302758
## [3,] -0.1728464
## [4,]  0.3915407
## [1] "Autocorrelations in phase 3:"
## [1] 0.150520138 0.063297818 0.001525184 0.065434827
## [1] "Estimated statistics after phase 3"
## [1]  2.8875  2.1240 13.5540  2.1720
## [1] "Estimates after phase 3"
##
## [1,] -1.6247329
## [2,]  0.3302758
## [3,] -0.1728464
## [4,]  0.3915407
```

```
# get results table
estimation
```

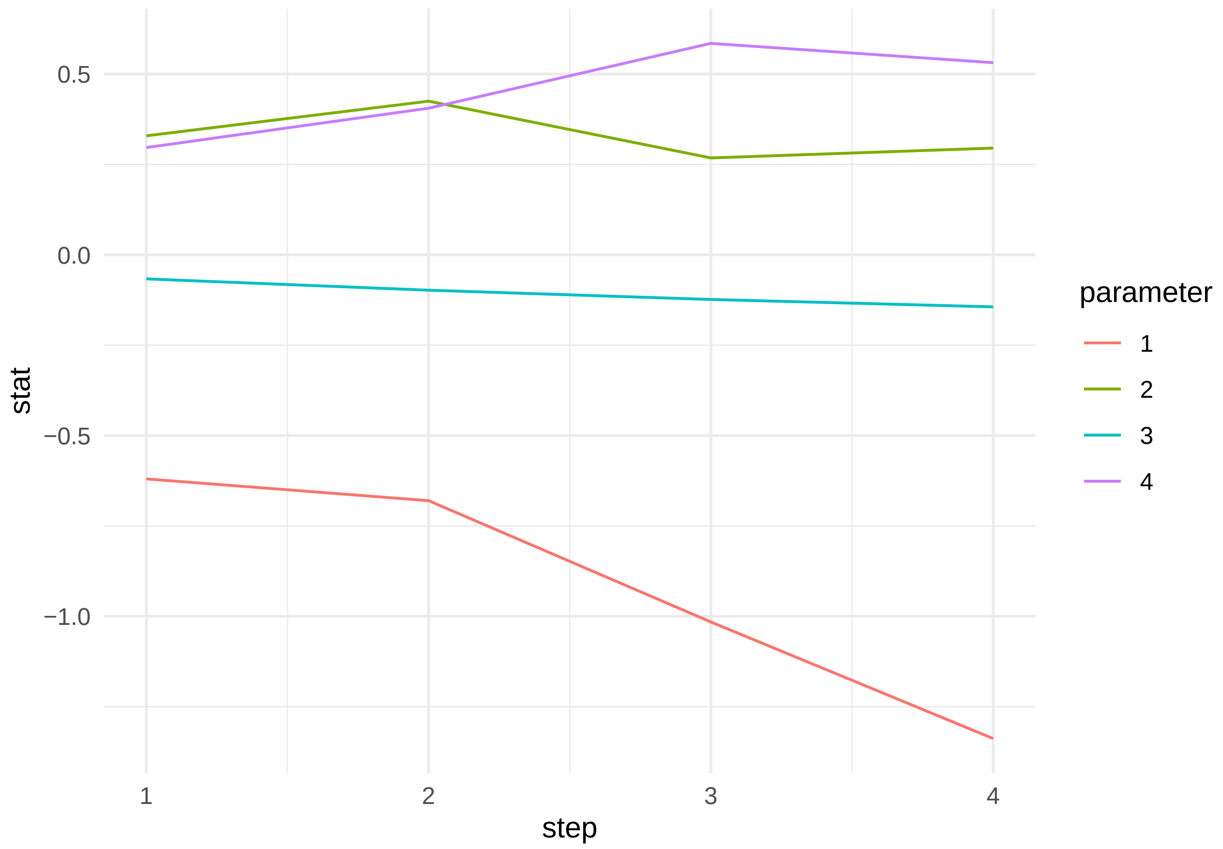
```
##       effect     object        est   std.err sig          t       conv
## 1 num_groups  partition -1.6247329 2.1479104     -0.7564249 -0.1616931
## 2       same     gender  0.3302758 1.2077984      0.2734528  0.1027869
## 3       diff        age -0.1728464 0.1425587     -1.2124578  0.1816995
## 4        tie friendship  0.3915407 1.8876439      0.2074230  0.1887000
```

Convergence improved!

### 3.3 Estimate plots

To assess convergence problems, we can have a look at how estimates evolve during phase 2.

```
ggplot(data = data.frame(step = 1:nrow(objects.phase2$estimates),
                         stat = array(objects.phase2$estimates),
                         parameter = as.character(rep(seq(1,ncol(objects.phase2$estimates)), each=nrow(
  geom_line(aes(x=step,y=stat,colour=parameter))
```



### 3.4 Goodness of fit

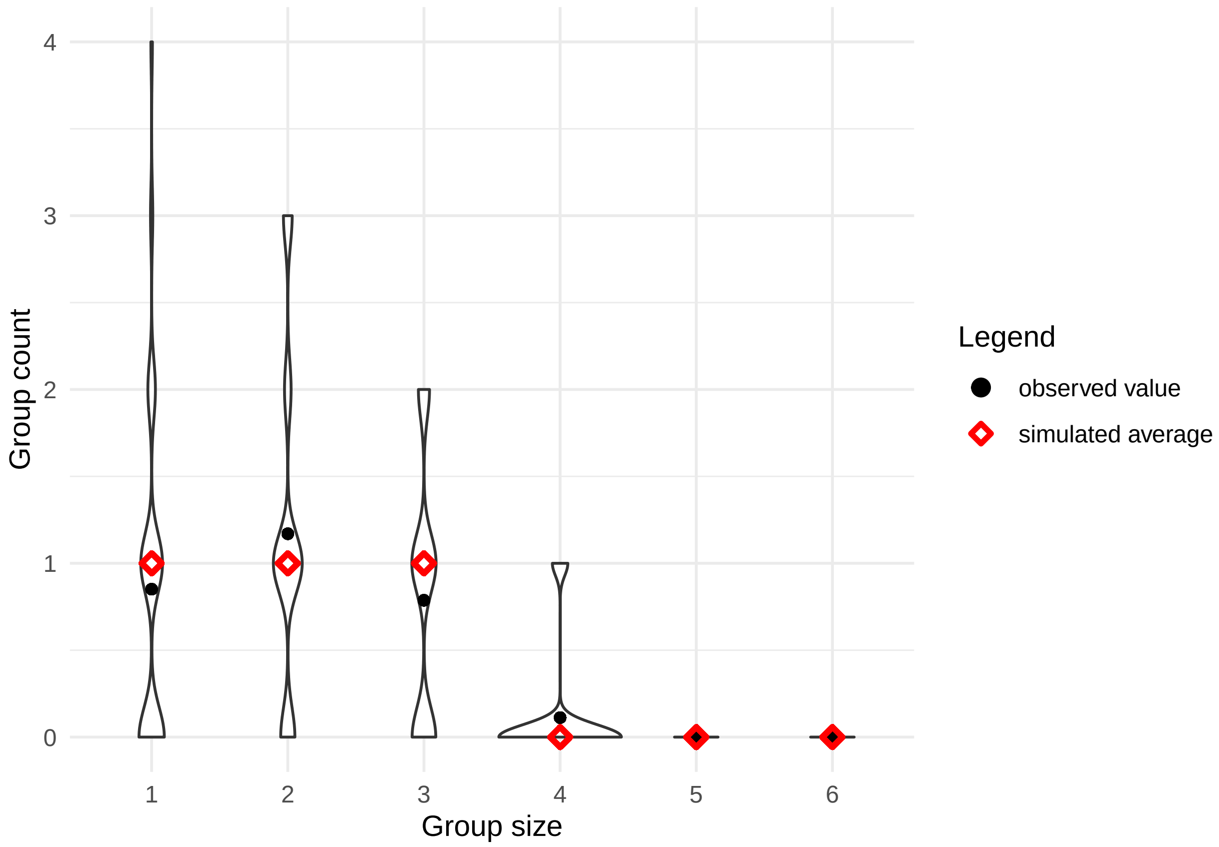
We can check how the model reproduces statistics of the observed data. First we simulate the estimated model (with the option of returning all partitions!):

```
nsimulations <- 1000
simulations <- draw_Metropolis_single(theta = estimation$results$est,
                          first.partition = partition,
                          nodes = nodes,
                          effects = effects,
                          objects = objects,
                          burnin = 100,
                          thining = 20,
                          num.steps = nsimulations,
                          neighborhood = c(0,1,0),
                          sizes.allowed = 1:n,
                          sizes.simulated = 1:n,
                          return.all.partitions = T)
```

We can first check the group size distribution:

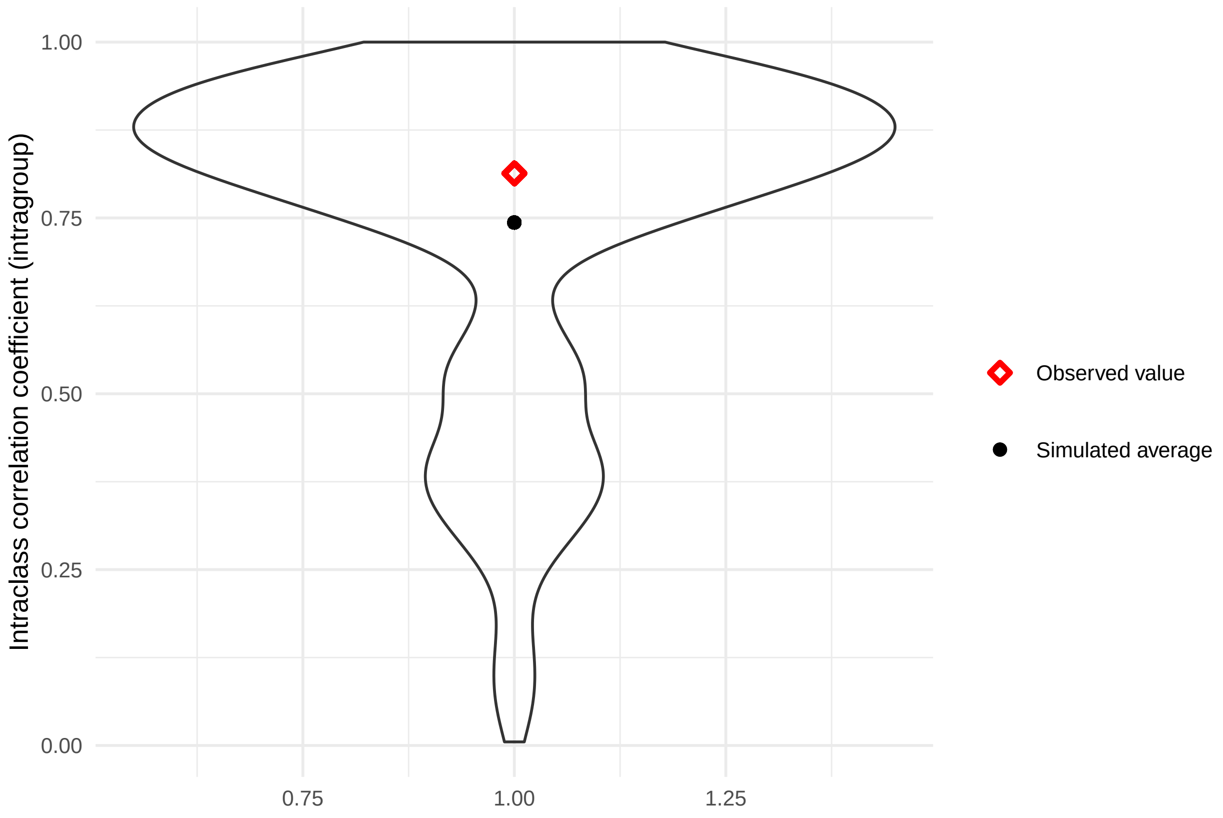
```
observedsizes <- rep(0,n)
for(size in 1:n){
  observedsizes[size] <- length(which(table(partition)==size))
}
```

```
allsizes <- matrix(0,nrow=nsimulations,ncol=n)
meansizes <- matrix(0,n)
for(size in 1:n){
  for(simu in 1:nsimulations){
    allsizes[simu,size] <- length(which(table(simulations$all.partitions[simu,])==size))
  }
  meansizes[size] <- mean(allsizes[,size])
}

df <- data.frame(simulation = 1:nsimulations,
                 simulated_stat = array(allsizes),
                 size = as.character(rep(seq(1,n),each=nsimulations)),
                 observed_stat = rep(observedsizes,each=nsimulations),
                 mean_stat = rep(meansizes,each=nsimulations))
ggplot(df, aes(factor(size), simulated_stat)) +
    geom_violin() +
    geom_point(aes(x=factor(size),y=mean_stat, colour = "simulated", shape= "simulated")) +
    geom_point(aes(x=factor(size),y=observed_stat, colour = "observed", shape= "observed"), stroke= 1.5
    labs(x = "Group size",
         y = "Group count",
         color="Legend",
         shape="Legend") +
    scale_color_manual(values = c(simulated="black",observed="red"), labels=c("observed value","simulate
    scale_shape_manual(values = c(simulated=19,observed=5), labels=c("observed value","simulated averag
    theme(legend.position = "right")
```



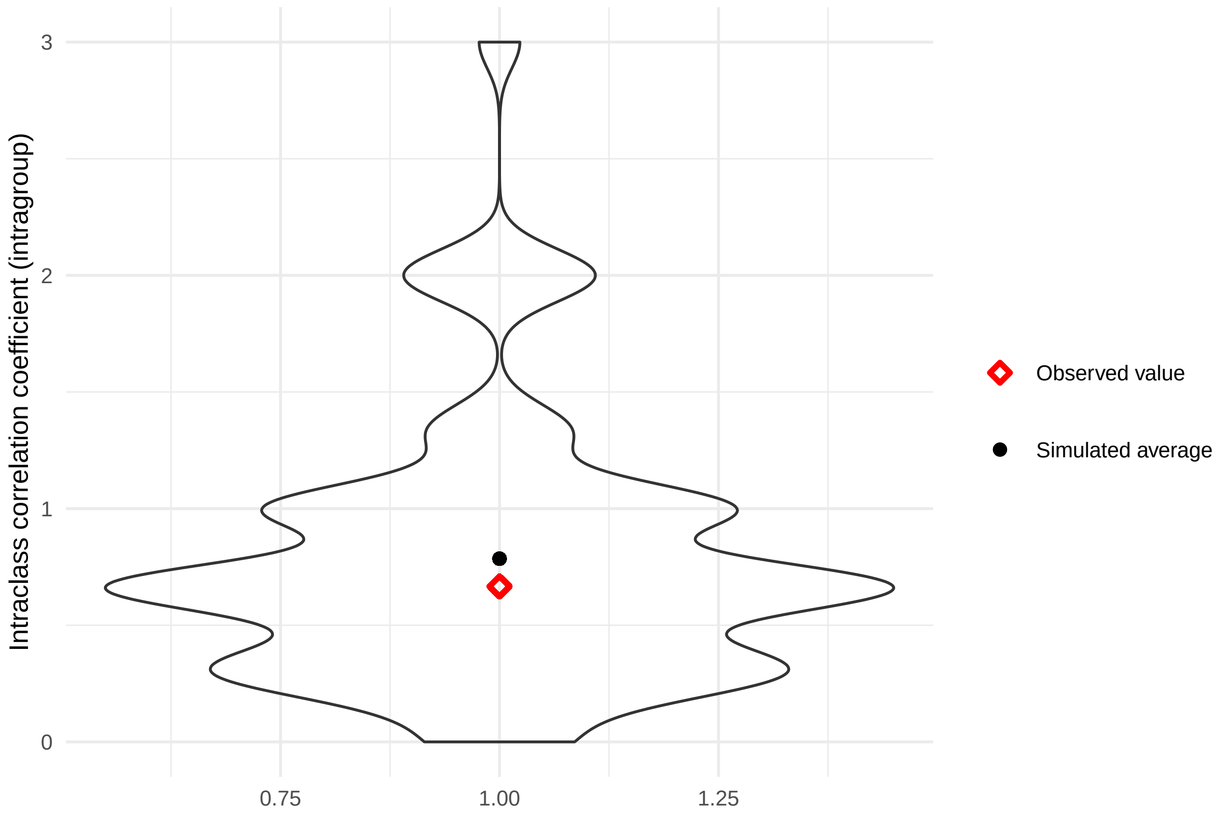
For continuous variables, we can check the intra-class correlation of the variable (linked to the statistic used in the model, but not equal):

```
allicc <- rep(0,nsimulations)
for(simu in 1:nsimulations){
  allicc[simu] <- icc(simulations$all.partitions[simu,],nodes$age)
}

df <- data.frame(simulation = 1:nsimulations,
                 simulated_stat = allicc,
                 observed_stat = rep(icc(partition,nodes$age),nsimulations),
                 mean_stat = rep(mean(allicc,na.rm=T),nsimulations))
ggplot(df) +
    geom_violin(aes(x=1,y=simulated_stat), fill = NA) +
    geom_point(aes(x=1, y=mean_stat, colour = "simulated", shape= "simulated"), stroke= 1.5) +
    geom_point(aes(x=1, y=observed_stat, colour = "observed", shape= "observed"), stroke= 1.5) +
    labs(x = "",
         y = "Intraclass correlation coefficient (intragroup)",
         color="",
         shape="",
         linetype="") +
    scale_color_manual(values = c(observed="red", simulated="black"), labels=c("Observed value","Simula
    scale_shape_manual(values = c(observed=5, simulated=16), labels=c("Observed value","Simulated averag
    theme(legend.position = "right",
          legend.key.height = unit(0.4,"in"),
          text = element_text(size=10))
```



We can also check the density of pairs with same gender in the groups (linked to the statistic used in the model, but not equal):

```
alldensities <- rep(0,nsimulations)
for(simu in 1:nsimulations){
  alldensities[simu] <- same_pairs(simulations$all.partitions[simu,], nodes$gender, 'avg_pergroup')
}

df <- data.frame(simulation = 1:nsimulations,
                 simulated_stat = alldensities,
                 observed_stat = rep(same_pairs(partition, nodes$gender, 'avg_pergroup'),nsimulations),
                 mean_stat = rep(mean(alldensities,na.rm=T),nsimulations))
ggplot(df) +
    geom_violin(aes(x=1,y=simulated_stat), fill = NA) +
    geom_point(aes(x=1, y=mean_stat, colour = "simulated", shape= "simulated"), stroke= 1.5) +
    geom_point(aes(x=1, y=observed_stat, colour = "observed", shape= "observed"), stroke= 1.5) +
    labs(x = "",
         y = "Intraclass correlation coefficient (intragroup)",
         color="",
         shape="",
         linetype="") +
    scale_color_manual(values = c(observed="red", simulated="black"), labels=c("Observed value","Simula
    scale_shape_manual(values = c(observed=5, simulated=16), labels=c("Observed value","Simulated averag
    theme(legend.position = "right",
          legend.key.height = unit(0.4,"in"),
          text = element_text(size=10))
```

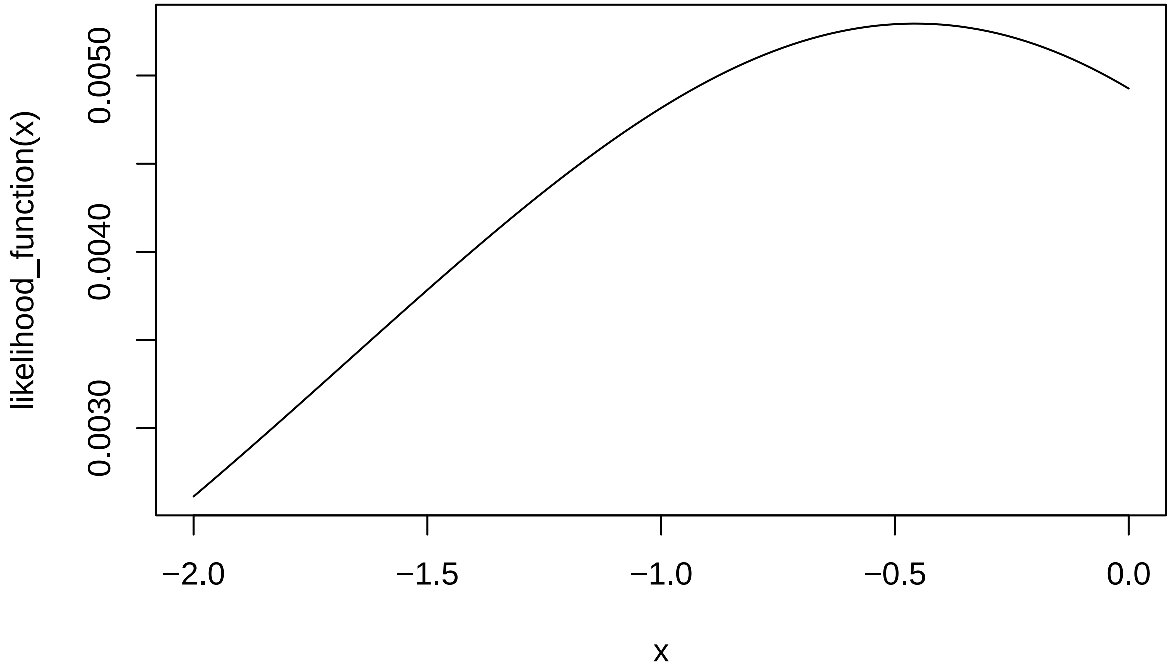


And it is also good to check other statistics even less linked to the model, for example on attributes that are not related to any effect in the model.

### 3.5 Estimated log-likelihood and AIC

Finally, we can estimate the log-likelihood and AIC of the model (useful to compare two models for example). First we need to estimate the ML estimates of a simple model with only one parameter for number of groups (this parameter should be in the model!).

```
likelihood_function <- function(x){ exp(x*max(partition)) / compute_numgroups_denominator(n,x)}
curve(likelihood_function, from=-2, to=0)
```



```
parameter_base <- optimize(likelihood_function, interval=c(-2, 0), maximum=TRUE)
parameters_basemodel <- c(parameter_base$maximum,0,0,0)
```

Then we can get our estimated logL and AIC.

```
logL_AIC <- estimate_logL(partition,
                          nodes,
                          effects,
                          objects,
                          theta = estimation$results$est,
                          theta_0 = parameters_basemodel,
                          M = 3,
                          num.steps = 200,
                          burnin = 100,
                          thining = 20)
 logL_AIC$logL
```

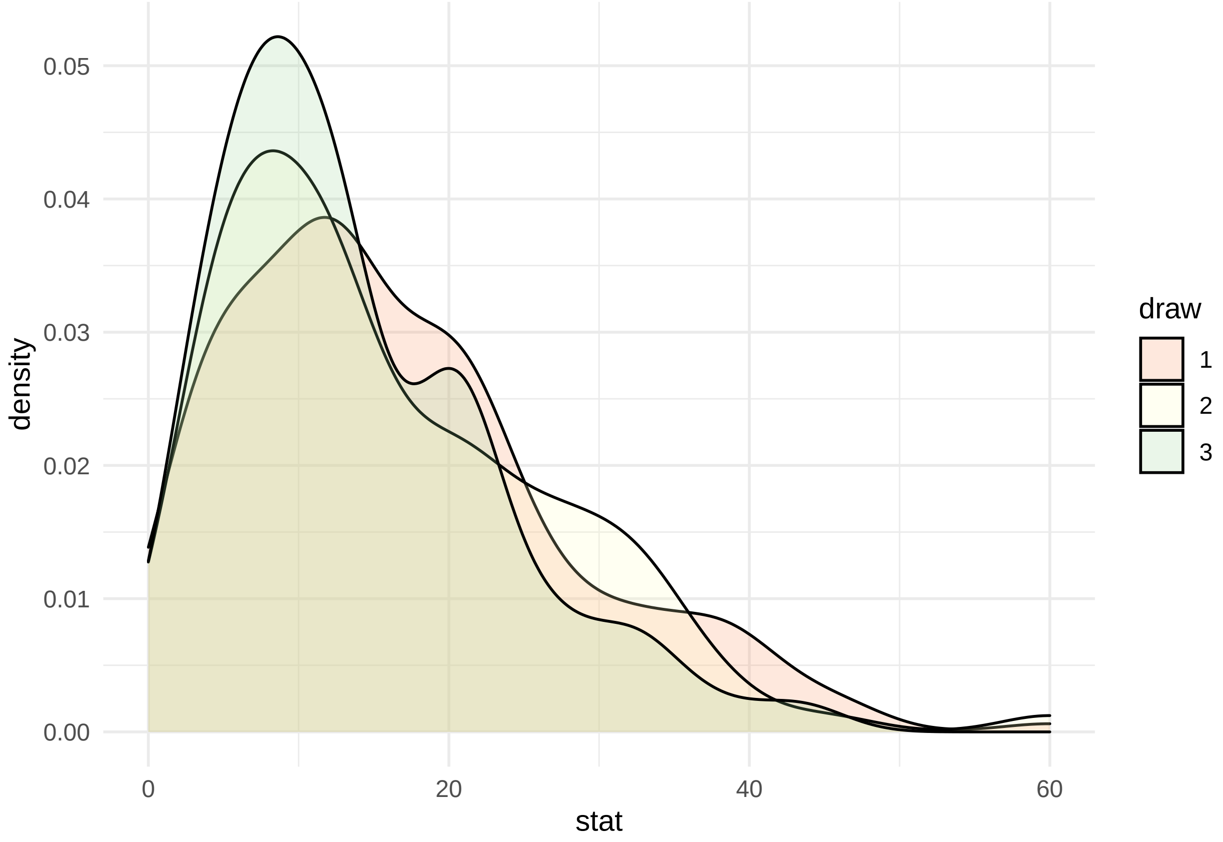
```
##            [,1]
## [1,] -4.314276
```

```
logL_AIC$AIC
```

```
##            [,1]
## [1,] 0.6285522
```

To check that there were enough steps in the algorithm (M), we can plot the statistics distributions of the intermediate models sampled and see whether they overlap (if not, we need more steps). For example for the stat about age:

```
stat <- 3
ggplot(data =  data.frame(stat = c(logL_AIC$draws[[1]]$draws[,stat], logL_AIC$draws[[2]]$draws[,stat], l
             draw = c(rep("1",100), rep("2",100), rep("3",100)))) +
    geom_density(aes(x=stat, fill=draw),alpha=0.2) +
    scale_fill_brewer(palette = "Spectral") +
    theme_minimal()
```



NOTE: with larger data, things can be parallelized. The only available “automatic” way to do it now is to specify the options “cpus” in the estimation function. In that case, phase 1 and 3 samples are parallelized.