Simulation and estimation of Exponential Random Partition Models - Example script with random data

Marion Hoffman

30/04/2020

0. Load dependencies

The following packages and scripts are required to fully run this example script.

```
library(ggplot2)
library(numbers)
library(combinat)
library(gmp)
library(clues)

current_wd <- getwd()
filesLoad <- list.files(paste(current_wd,"/function_ERPM",sep=""), pattern = "R$")
lapply(filesLoad, function(x) source(paste(current_wd,"/function_ERPM/",x,sep="")))

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:

```
p1 <- all_partitions[1,]
p2 <- all_partitions[2,]
rand_12 <- adjustedRand(p1,p2,randMethod="Rand")
```

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)
```

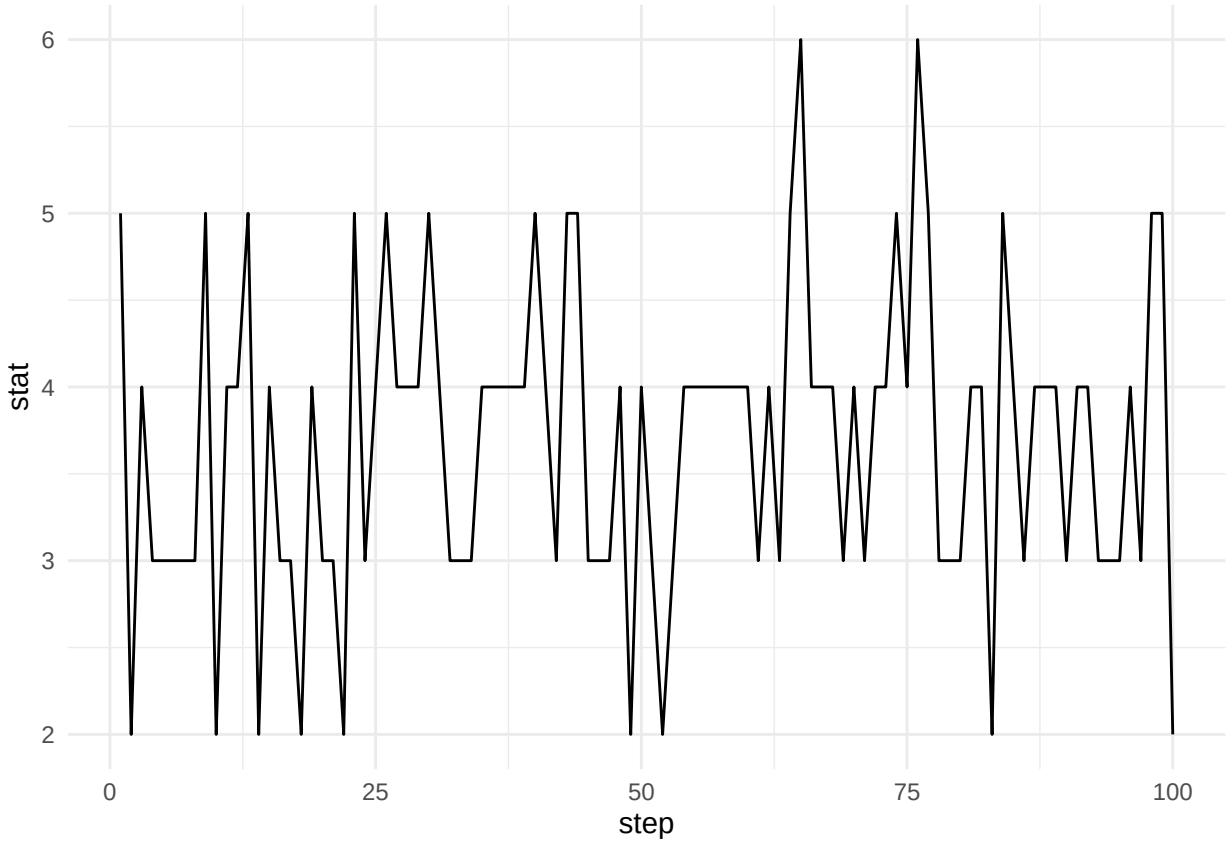
Now we can generate our simulated sample, by setting the desired additional parameters for the Metropolis sampler and choosing a starting point for the chain (first.partition):

```
nsteps <- 100
sample <- draw_Metropolis(theta = parameters,
                          first.partition = 1:n,
                          nodes = nodes,
                          effects = effects,
                          objects = objects,
                          burnin = 100,
                          thining = 10,
                          num.steps = nsteps,
                          mini.steps = "normalized",
                          neighborhood = 2,
                          sizes.allowed = 1:n,
                          sizes.simulated = 1:n)
```

2.3 Trace plots

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

```
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 don't need to sample much either. 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(-1,0,0,0)
estimation <- estimate_ERPM(partition,
                            nodes,
                            objects,
                            effects,
                            startingestimates = startingestimates,
                            burnin = 100,
                            thining = 20,
                            length.p1 = 100,
                            min.iter.p2 = 10,
                            max.iter.p2 = 200,
                            num.steps.p2 = 6,
                            length.p3 = 1000)
```

```
## [1] "Observed statistics"
## [1]  3  2 12  2
## [1] "Burn-in"
## [1] 100
## [1] "Thining"
## [1] 20
## [1] "Estimated statistics after phase 1"
## [1]  2.16  3.44 48.96  3.44
## [1] "Estimates after phase 1"
##             [,1]
## [1,] -0.43552993
## [2,] -0.18756790
## [3,] -0.05121611
## [4,] -0.18964494
## Estimated statistics after phase 2, step 1NULL
##  [,1] [,2] [,3] [,4]
##     4    1   18    0
## Estimates after phase 2, step 1NULL
##
## [1,] -0.84065838
## [2,] -0.06600653
## [3,] -0.06332476
## [4,] -0.05025269
```

```
## Estimated statistics after phase 2, step 2NULL
##  [,1] [,2] [,3] [,4]
##     5    0    9    0
## Estimates after phase 2, step 2NULL
##
## [1,] -0.87286575
## [2,] -0.03957321
## [3,] -0.07101826
## [4,] -0.03655456
## Estimated statistics after phase 2, step 3NULL
##  [,1] [,2] [,3] [,4]
##     5    1    2    1
## Estimates after phase 2, step 3NULL
##
## [1,] -0.97551911
## [2,] -0.02604806
## [3,] -0.07249847
## [4,] -0.01765188
## Estimated statistics after phase 2, step 4NULL
##  [,1] [,2] [,3] [,4]
##     4    1    6    2
## Estimates after phase 2, step 4NULL
##
## [1,] -1.012013247
## [2,] -0.011229629
## [3,] -0.073180542
## [4,] -0.004128098
## Estimated statistics after phase 2, step 5NULL
##  [,1] [,2] [,3] [,4]
##     2    2   38    1
## Estimates after phase 2, step 5NULL
##
## [1,] -1.0330532775
## [2,] -0.0013938384
## [3,] -0.0742350373
## [4,]  0.0003770204
## Estimated statistics after phase 2, step 6NULL
##  [,1] [,2] [,3] [,4]
##     3    1   20    1
## Estimates after phase 2, step 6NULL
##
## [1,] -1.0279583990
## [2,]  0.0005499205
## [3,] -0.0747254215
## [4,]  0.0022770345
## [1] "Estimated statistics after phase 3"
## [1]  3.300  1.653 18.809  1.652
## [1] "Estimates after phase 3"
##
## [1,] -1.0279583990
## [2,]  0.0005499205
## [3,] -0.0747254215
## [4,]  0.0022770345
##       effect     object           est    std.err sig           t       conv
```

```
## 1 num_groups  partition -1.0279583990 0.02833377 *** -36.28031919  0.3348242
## 2       same     gender  0.0005499205 0.03921899       0.01402179 -0.2797906
## 3       diff        age -0.0747254215 0.42540744      -0.17565612  0.5061488
## 4        tie friendship  0.0022770345 0.03649970       0.06238502 -0.3015018
```

```
estimation$results
```

```
##       effect     object           est    std.err       conv
## 1 num_groups  partition -1.0279583990 0.02833377  0.3348242
## 2       same     gender  0.0005499205 0.03921899 -0.2797906
## 3       diff        age -0.0747254215 0.42540744  0.5061488
## 4        tie friendship  0.0022770345 0.03649970 -0.3015018
```

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:

```
startingestimates <- estimation$results$est
startingcovariance <- estimation$objects.phase3$inv.zcov
startingscaling <- estimation$objects.phase3$inv.scaling
estimation <- estimate_ERPM(partition,
                            nodes,
                            objects,
                            effects,
                            startingestimates = startingestimates,
                            multiplicationfactor = 30,
                            gainfactor = 0.1,
                            mini.steps = "normalized",
                            burnin = 100,
                            thining = 20,
                            length.p1 = 100,
                            min.iter.p2 = 10,
                            max.iter.p2 = 200,
                            num.steps.p2 = 3,
                            length.p3 = 2000,
                            inv.zcov = startingcovariance,
                            inv.scaling = startingscaling)
```

```
## [1] "Observed statistics"
## [1]  3  2 12  2
## [1] "Burn-in"
## [1] 100
## [1] "Thining"
## [1] 20
## Estimated statistics after phase 2, step 1NULL
##  [,1] [,2] [,3] [,4]
##     2    4   34    4
## Estimates after phase 2, step 1NULL
##
## [1,] -0.98175980
## [2,]  0.07324356
## [3,] -0.08181982
## [4,]  0.04583823
## Estimated statistics after phase 2, step 2NULL
##  [,1] [,2] [,3] [,4]
##     3    2    8    2
## Estimates after phase 2, step 2NULL
```

```
##
## [1,] -0.91308574
## [2,]  0.06668289
## [3,] -0.08851863
## [4,]  0.08838757
## Estimated statistics after phase 2, step 3NULL
##  [,1] [,2] [,3] [,4]
##     5    1    2    1
## Estimates after phase 2, step 3NULL
##
## [1,] -0.94548971
## [2,]  0.07183291
## [3,] -0.09163247
## [4,]  0.11378040
## [1] "Estimated statistics after phase 3"
## [1]  3.3335  1.5850 16.9365  1.5995
## [1] "Estimates after phase 3"
##
## [1,] -0.94548971
## [2,]  0.07183291
## [3,] -0.09163247
## [4,]  0.11378040
##       effect     object         est    std.err sig           t       conv
## 1 num_groups  partition -0.94548971 0.01948920 *** -48.5135287  0.3826369
## 2       same     gender  0.07183291 0.02598509  **   2.7643889 -0.3571156
## 3       diff        age -0.09163247 0.27489547      -0.3333357  0.4015472
## 4        tie friendship  0.11378040 0.02430731 ***   4.6809131 -0.3684263
```

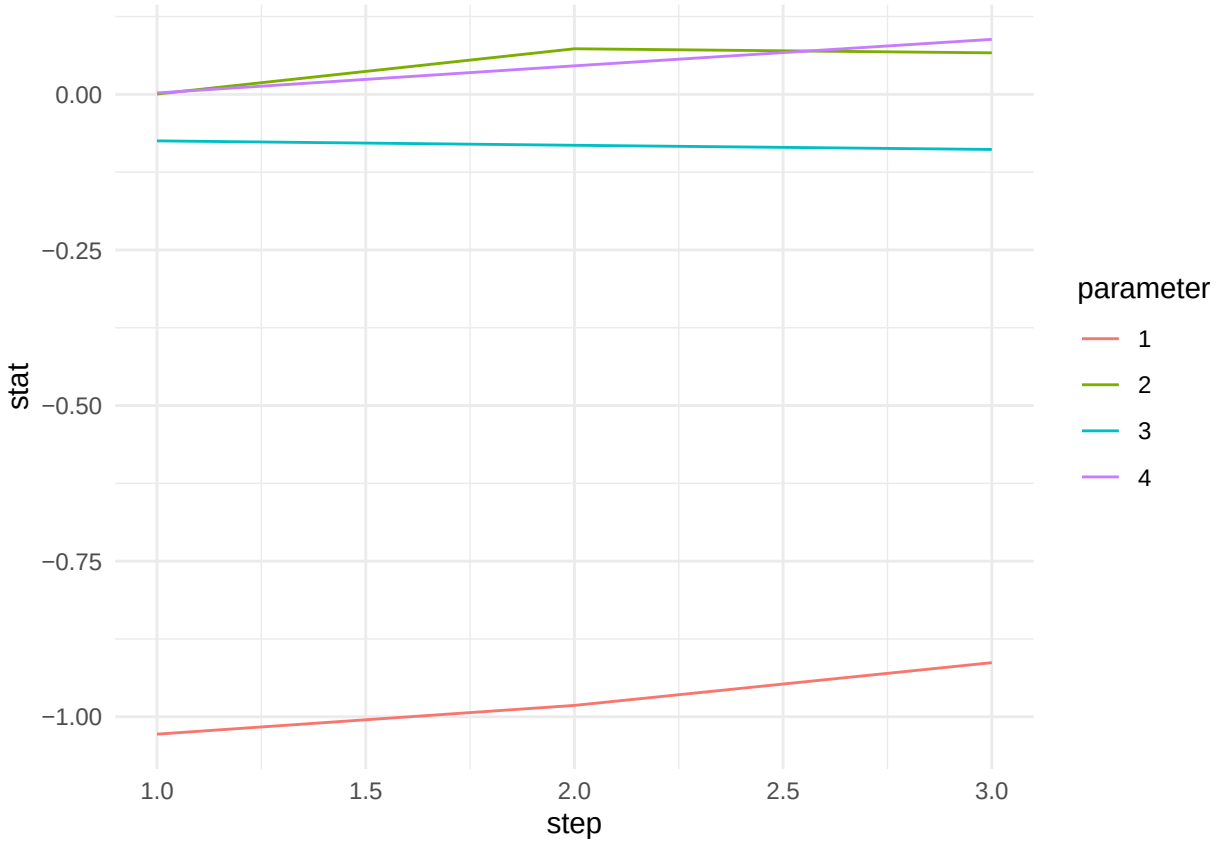
```
estimation$results
```

```
##       effect     object         est    std.err       conv
## 1 num_groups  partition -0.94548971 0.01948920  0.3826369
## 2       same     gender  0.07183291 0.02598509 -0.3571156
## 3       diff        age -0.09163247 0.27489547  0.4015472
## 4        tie friendship  0.11378040 0.02430731 -0.3684263
```

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



3.4 Goodness of fit

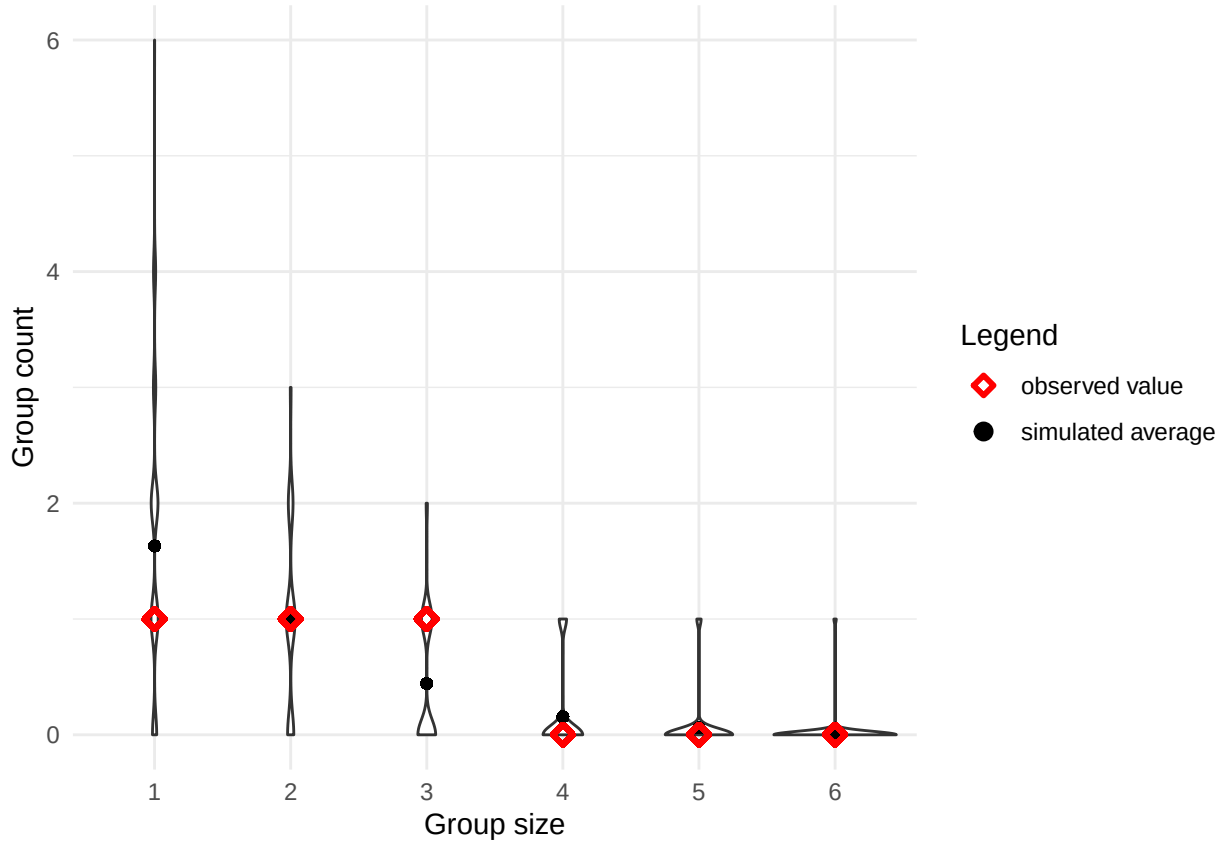
We can check how the model reproduces statistics of the observed data. First we simulate the estimated model (with the option of returning all partitions!):

```
nsimulations <- 1000
simulations <- draw_Metropolis(theta = estimation$results$est,
                               first.partition = partition,
                               nodes = nodes,
                               effects = effects,
                               objects = objects,
                               burnin = 100,
                               thining = 20,
                               num.steps = nsimulations,
                               mini.steps = "normalized",
                               neighborhood = 2,
                               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))
}
```

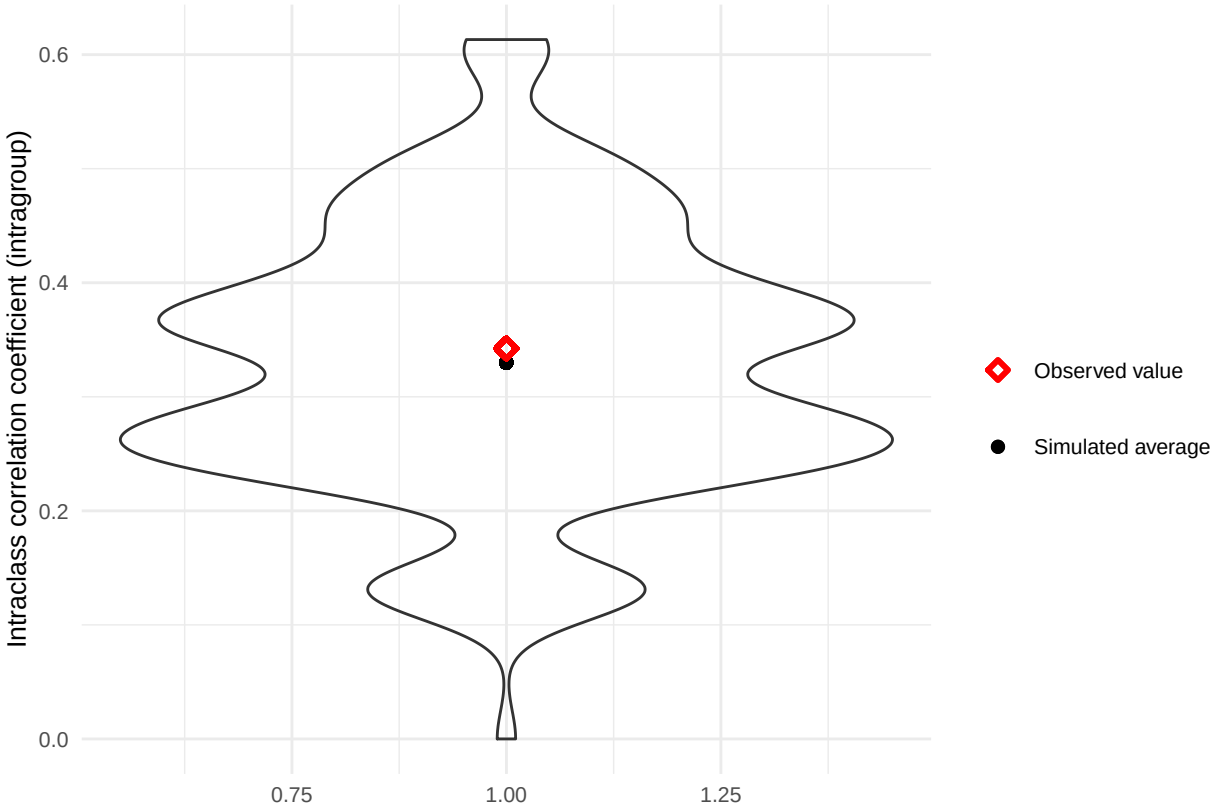
```r
allsizes <- matrix(0,nrow=nsimulations,ncol=n)
meansizes <- matrix(0,n)
for(size in 1:n){
  for(simu in 1:nsimulations){
    allsizes[simu,size] <- length(which(table(simulations$all.partitions[simu,])==size))
  }
  meansizes[size] <- mean(allsizes[,size])
}

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):

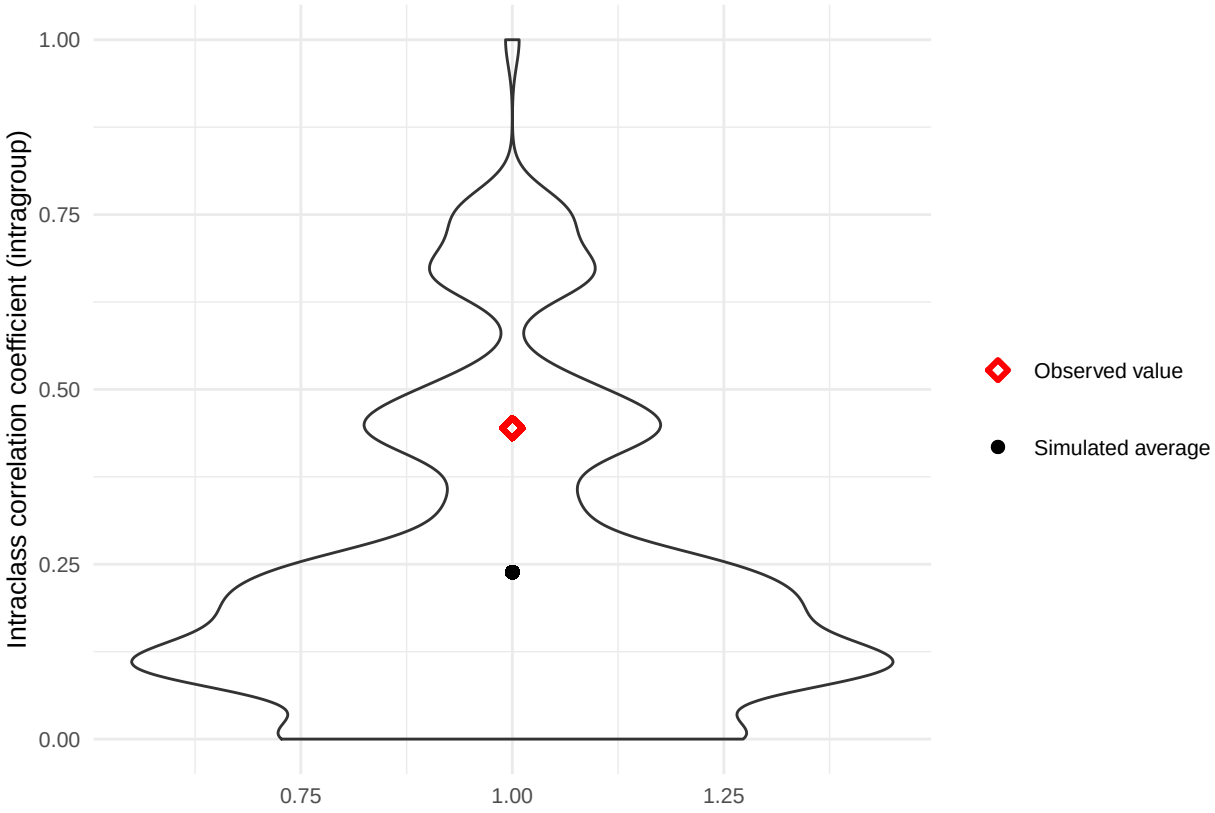
```
allicc <- rep(0,nsimulations)
for(simu in 1:nsimulations){
  allicc[simu] <- computeicc(nodes$age,simulations$all.partitions[simu,])
}

df <- data.frame(simulation = 1:nsimulations,
                 simulated_stat = allicc,
                 observed_stat = rep(computeicc(nodes$age,partition),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 avera
    theme(legend.position = "right",
          legend.key.height = unit(0.4,"in"),
          text = element_text(size=10))
```

```
## Warning: Removed 19 rows containing non-finite values (stat_ydensity).
```



For binary variables, we can check the intra-group density (linked to the statistic used in the model, but not equal):

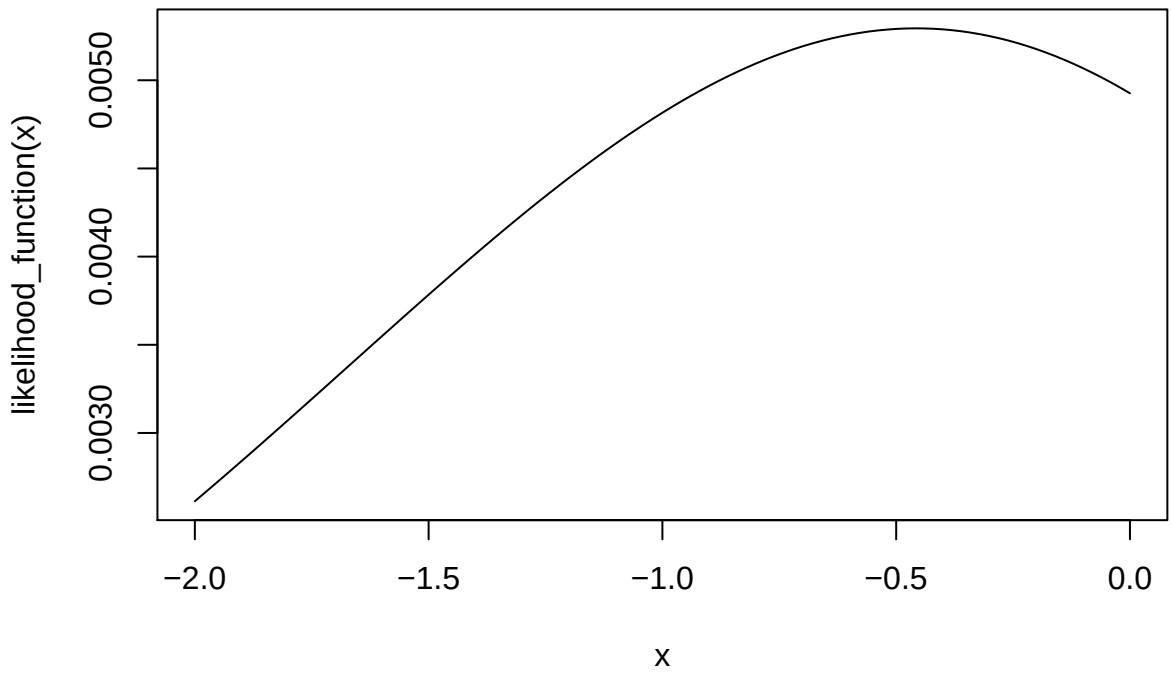
```
samegender <- as.matrix(1-dist(nodes$gender,upper=T,diag=T))
alldensities <- rep(0,nsimulations)
for(simu in 1:nsimulations){
  alldensities[simu] <- computedensity(samegender,simulations$all.partitions[simu,])$average
}

df <- data.frame(simulation = 1:nsimulations,
                 simulated_stat = alldensities,
                 observed_stat = rep(computedensity(samegender,partition)$average,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 avera
    theme(legend.position = "right",
          legend.key.height = unit(0.4,"in"),
          text = element_text(size=10))
```



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)) / calculate_logL_Dirichlet(nodes, x, sizes.al
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)
```

```
## [1] "step 1"
## [1] "step 2"
## [1] "step 3"
```

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()
```