Simulation Study

Lapo Santi

2023-03-08

In this part, I want to check that the model is correcly specified and that there are no coding mistakes, at last in the probability part.

My first step in this direction is to check that the likelihood is maximized by the true (in this case, known) parameter value.

In this model, the unknown parameters are $\mathbf{z}, \mathbf{K}, \mathbf{P}$. The initial check that I do is to fix two parameters at the time, in order to get an univariate likelihood, which will facilitate computations and the visualization with a plot.

First, I generate a fake tournament and store the output.

The full dataset will look as follows:

Inference

In this section, I want to break down the Metropolis-Hastings code in 3 pieces, more manageable and simple.

Each step, described in the Simplified_model file, is taken separately, as a separate inferential exercise. To do that, I fix the other two parameters of the posterior, therefore making it univariate. Given that the posterior has three different parameters, namely z,p and K, I fix two of them at the time.

Fixing p and K

In this section, I want to infer z, by fixing p and K equal to the true, known, values . Here below is the code:

```r
# computing the likelihood
get_A = function(nij,yij,pij){
  return(sum(lchoose(nij,yij)+(yij*log(pij))+((nij- yij)*log(1-pij))))}

#computing the prior on z|K
dir_multinom_d<- function(K,N,n_k, log=T){
  gamma = rep(1,K)
  alpha_post <- alpha + n_k
  A <- sum(alpha)
  N_post <- sum(alpha_post)
  marginal <- gamma(A) / gamma(N_post + A) * prod(gamma(table(z) + alpha) / gamma(alpha))
  if(log==T){
  return(log(marginal))}else{return(marginal)}
}



dir_multinom_r<- function(K,N,gamma=1){
  alpha=rep(gamma,K) # value of alpha, here chosen at random
  p=rgamma(K,alpha) # pre-simulation of the Dirichlet
  y=sample(1:K,n,prob=p/sum(p),rep=TRUE) # Multinomial
  return(y)}

N_iter = 30000

#each column contains one of the samples
z_seq = matrix(0, N, N_iter)
#keeping track of the posterior distribution
A_seq = matrix(0, N_iter,1)

labels_available = 1:K_true

z_proposal = sample(labels_available,1)
init = kmeans(data.frame(synth_matches$n_ij,synth_matches$y_ij), K_true)$cluster
z_current = data.frame(id = synth_players$id, z = init)

z_seq[,1]<- init
df_current = df_aux_fast(synth_matches,z_current,synth_p)
n_current = table(z_current$z)
A_seq[1]<- get_A(df_current$n_ij,df_current$y_ij,df_current$p_ij)+dir_multinom_d(N,K_true,n_current)
acc.count=0
pb=txtProgressBar(min=1,max=N_iter)
j=2

while(j<N_iter+1){
  setTxtProgressBar(pb, j)
  z_prime = z_current
  df_prime = df_current
  n_prime = table(z_prime$z)
  for(ii in 1:N){

    z_scanning = z_prime
    k_proposal = sample(labels_available,1)

    z_scanning[which(z_scanning$id == z_scanning$id[ii]),]$z = k_proposal
    df_scanning = df_aux_fast(synth_matches,z_scanning,synth_p)

    n_scanning = table(z_scanning$z)

    r = (get_A(df_scanning$n_ij,df_scanning$y_ij,df_scanning$p_ij )+ dir_multinom_d(N,K_true,n_scanning

    u=runif(1)
    if(log(u)<r){
      acc.count=acc.count+1
      z_prime <- z_scanning
      df_prime = df_aux_fast(synth_matches,z_prime,synth_p)
    }
  }
  z_current <- z_prime
  n_current = table(z_current$z)
  z_seq[,j-1] = z_prime$z
  A_seq[j-1] = get_A(df_prime$n_ij,df_prime$y_ij,df_prime$p_ij )+ dir_multinom_d(N,K_true,n_scanning)
  j=j+1

}


close(pb)
cat("accepted ", signif(100*acc.count/(N_iter*N),2), "%\n",sep="")
rand.index(z_seq[,which(A_seq[1:(N_iter-1)] == max(A_seq[1:(N_iter-1)]))], synth_players$z)

ts.plot(A_seq[1:(N_iter-1)])

library(mcclust)
simil<-pr_cc(z_seq)
diag(simil)=1
minVI(simil)

post_avg <- round(rowSums(z_seq[,c(20000:30000)])/5000)
adj.rand.index(synth_players$z,post_avg)
```

```
###########
###INPUT
#N: number of players
#K:number of blocks
#z: label controlling the assignment of each player to one block
#p_ij: probabilities of victory against different blocks
#n_ij:number of matches between player ij
#y_ij:number of victories of player i against player j
#consider that p_ji ==1 - p_ij
#consider also that y_ji = n_ij - y_ij




N=100
a=1
b=3
K_max=4
M=300
max_clust=3
min_clust=3
max_number_games = 200
N_iter = 100000
############

############
#SIMULATING THE TOURNAMENT
synth = simulating_tournament_test(N=N,alpha = 1,beta = 3,min_clust = min_clust,max_clust = max_clust ,
synth_matches = synth$matches_results
synth_players = synth$z_true
synth_p = synth$p_true
K_true = synth$K_true

test = df_aux_fast(synth_matches,synth_players,synth_p)
n_test = table(synth_players$z)
n_test
```

```
##
##  1  2  3
## 23 54 23
```

```
cor(test$y_ij/test$n_ij, test$p_ij)
```

```
## [1] 0.938093
```

```
###################

#########
#INITIALIZING THE MCMC
##########


labels_available = 1:K_true
z_proposal = sample(labels_available, 1)


z_current = data.frame(id = synth_players$id, z = synth_players$z)
df_current = df_aux_fast(synth_matches, z_current, synth_p)
n_current = table(z_current$z)
A_seq = matrix(0, N_iter, 1)
z_seq = matrix(0, N, N_iter)
z_seq[, 1] =  z_current$z
A_seq[1] = get_A(df_current$n_ij, df_current$y_ij, df_current$p_ij) + dir_multinom_d(N, K_true, n_curren
acc.count = 0
pb = txtProgressBar(min = 1, max = N_iter)
j = 2
max_reassign = 3  # maximum number of nodes to reassign
min_acc_ratio = 0.3  # minimum acceptance ratio
reassign_nodes = 2  # initial number of nodes to reassign

#######
##METROPOLIS-HASTINGS

while (j < N_iter + 1) {
  setTxtProgressBar(pb, j)
  z_prime = z_current
  df_prime = df_current
  n_prime = n_current
  for (ii in 1:N) {
    reassign_nodes <- sample(size = 1, x=c(1:max_reassign), p=c(0.5,seq(from = 0.3,to= 0.01,by= ((0.01 
    if (reassign_nodes == 1) {
      z_scanning = get_proposal1(z_prime, labels_available)
    } else {
      z_scanning = get_proposal2(z_prime, labels_available, reassign_nodes)
    }
    df_scanning = df_aux_fast(synth_matches, z_scanning, synth_p)
    n_scanning = table(factor(z_scanning$z, levels = labels_available))
    r = (get_A(df_scanning$n_ij, df_scanning$y_ij, df_scanning$p_ij) + dir_multinom_d(N, labels_availabl
      (get_A(df_prime$n_ij, df_prime$y_ij, df_prime$p_ij) + dir_multinom_d(N, labels_available, n_prime
    u = runif(1)
    if (log(u) < r) {
      acc.count = acc.count + 1
      z_prime = z_scanning
      df_prime = df_scanning
      n_prime = n_scanning
      A_seq[j] = get_A(df_prime$n_ij, df_prime$y_ij, df_prime$p_ij) + dir_multinom_d(N, labels_availabl
    } else {
      A_seq[j] = A_seq[j - 1]
      # if (acc.count / j < min_acc_ratio && reassign_nodes < max_reassign) {
      #   reassign_nodes = reassign_nodes + 1
      # }
    }
    z_current = z_prime
    df_current = df_prime
    n_current = n_prime
    z_seq[, j - 1] = z_prime$z
    j = j + 1
  }
}
```

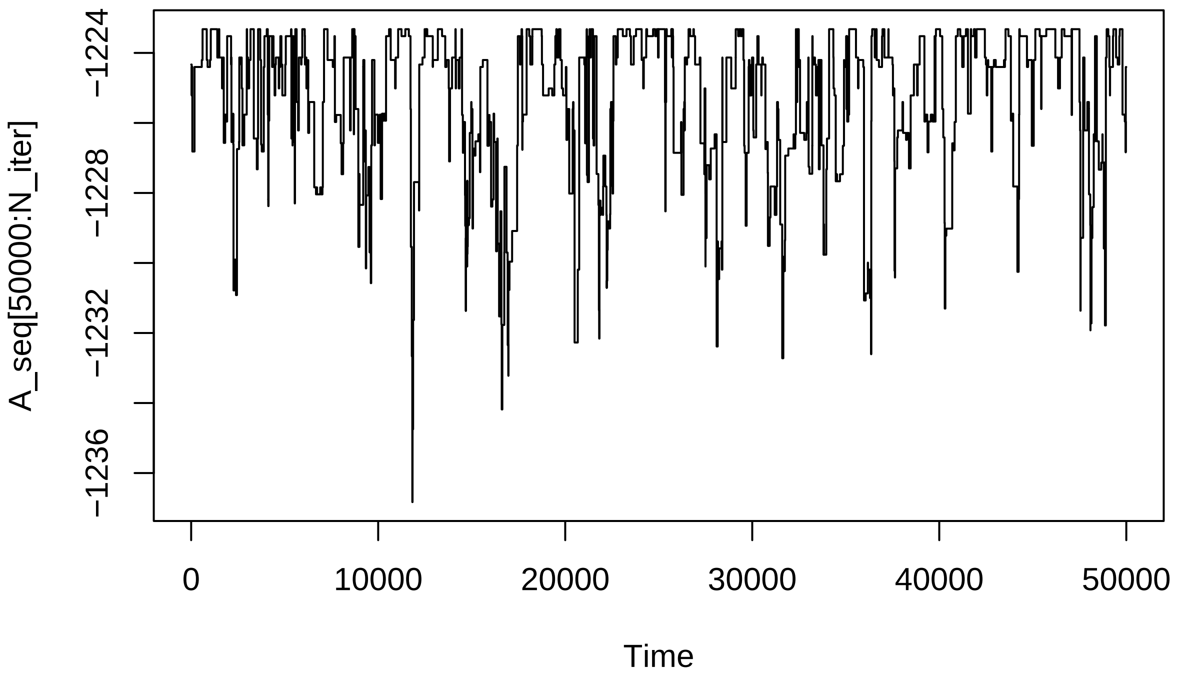
```
## ================================================================================
```

```
##########


#######
#RESULTS
close(pb)
```

```
cat("accepted ", signif(100*acc.count/(N_iter*N),2), "%\n",sep="")
```

```
## accepted 0.061%
```

```
MAP = which(A_seq[1:(N_iter-1)] == max(A_seq[1:(N_iter-1)]))


ts.plot(A_seq[50000:N_iter])
```



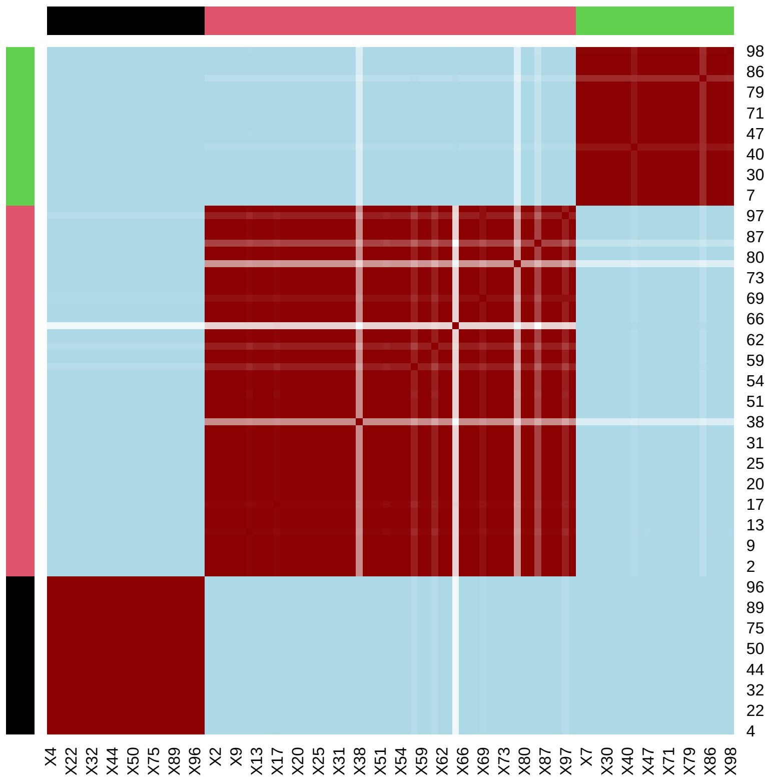
```
similarity<- pr_cc(z_seq[,c(50000:N_iter)])

memb_Z_DP_VI <- minVI(similarity,method="avg",max.k=5)

#POINT ESTIMATES
memb_Z_DP_VI$cl
```

```
##   [1] 1 1 2 3 1 1 2 2 1 3 1 1 1 3 1 1 1 1 1 1 1 3 1 3 1 1 1 3 2 2 1 3 2 1 2 1 3
##  [38] 1 1 2 3 1 2 3 2 3 2 3 2 3 1 1 1 1 2 1 1 3 1 1 1 1 1 1 3 1 1 1 1 1 2 1 1 1
##  [75] 3 1 2 2 2 1 3 1 2 1 2 2 1 3 3 3 1 1 2 2 3 3 1 2 1 3
```

```
z_seq[,MAP[1]]
```

```
##   [1] 2 2 3 1 2 2 3 3 2 1 2 2 2 1 2 2 2 2 2 2 2 1 2 1 2 2 2 1 3 3 2 1 3 2 3 2 1
##  [38] 2 2 3 1 2 3 1 3 1 3 1 3 1 2 2 2 2 3 2 2 1 2 2 2 2 2 2 1 2 2 2 2 2 3 2 2 2
##  [75] 1 2 3 3 3 2 1 2 3 2 3 3 2 1 1 1 2 2 3 3 1 1 2 3 2 1
```

```
similarity_plot(similarity,z_0 = synth_players$z, z_est =synth_players$z)
```



```
## $rowInd
##   [1]   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18
##  [19]  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36
##  [37]  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54
##  [55]  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72
##  [73]  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90
##  [91]  91  92  93  94  95  96  97  98  99 100
##
## $colInd
##   [1]   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18
##  [19]  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36
##  [37]  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54
##  [55]  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72
##  [73]  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90
##  [91]  91  92  93  94  95  96  97  98  99 100
##
## $Rowv
## NULL
##
## $Colv
## NULL
```

Fixing z and K

In this section, I want to infer p, by fixing z and K equal to the true, known, values . Here below is the code:

```
beta_0 = 3
N_iter = 30000

#containers
```

```r
p_seq= array(0,c(K_true,K_true,N_iter))
A_seq = matrix(0,N_iter,1)

p_current <- p_proposal(rbeta(K_true**2,1,1),2,K = K_true)
p_seq[,,1] = p_current
data_current = df_aux_fast(synth_matches,synth_players,p_current)


A_seq[1]= get_A(data_current$n_ij, data_current$y_ij,data_current$p_ij)+get_B(p_current, beta_0)
acc.count=0
pb=txtProgressBar(min=1,max=N_iter)
j=2
while(j<N_iter+1){
  setTxtProgressBar(pb, j)
  data_current = df_aux_fast(synth_matches,synth_players,p_current)
  #proposing a new p
  p_prime = p_proposal(p_seq[,,(j-1)],2,K = K_true)
  p_prime[is.na(p_prime)] <- 0
  p_prime = (1 -t(p_prime*upper.tri(p_prime))) * (lower.tri(p_prime, diag = F)*1) + upper.tri(p_prime, 
  diag(p_prime)=0.5
  #mapping it into the aux dataframe
  data_prime = df_aux_fast(synth_matches,synth_players,p_prime)

  #evaluating the likelihood
  r = (get_A(data_prime$n_ij, data_prime$y_ij,data_prime$p_ij)+get_B(p_prime, beta_0))-
    (get_A(data_current$n_ij, data_current$y_ij,data_current$p_ij)+get_B(p_current, beta_0))

  u=runif(1)
  if(log(u)<r){
    acc.count=acc.count+1
    p_seq[,,j] = p_prime
    A_seq[j] = get_A(data_prime$n_ij, data_prime$y_ij,data_prime$p_ij)+get_B(p_prime, beta_0)
  }else{
    p_seq[,,j] = p_current
    A_seq[j] = get_A(data_current$n_ij, data_current$y_ij,data_current$p_ij)+get_B(p_current, beta_0)
  }
  j=j+1
}
```

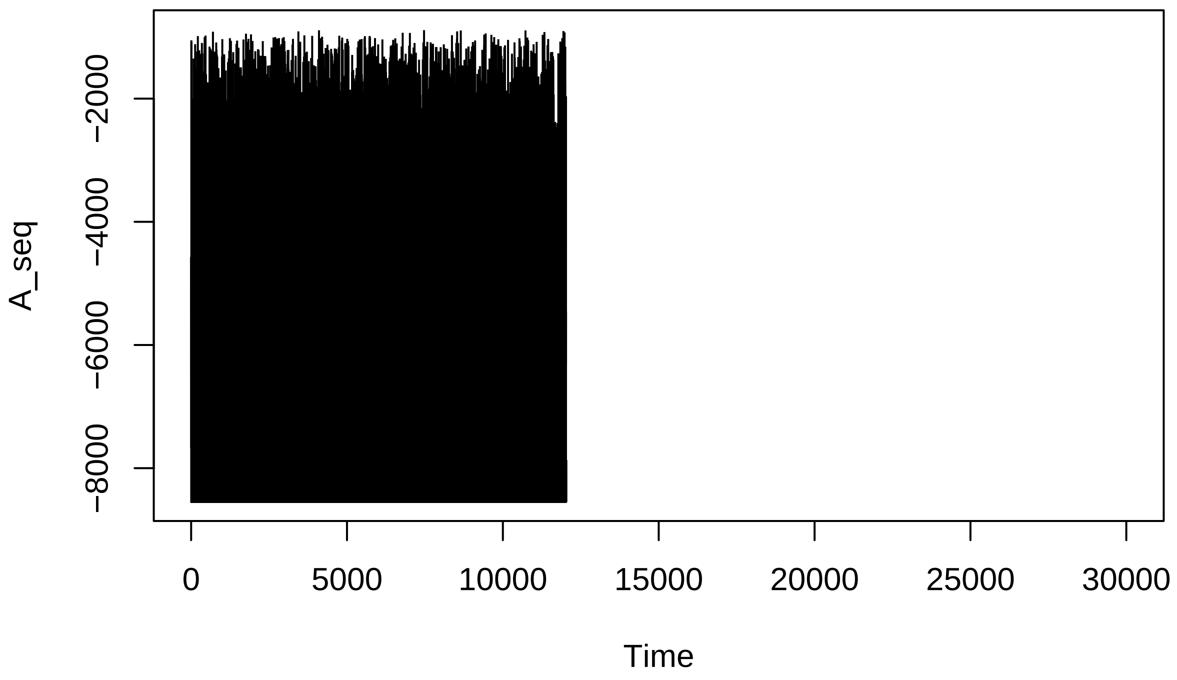
```
## ================================================================================
```

```r
close(pb)
```

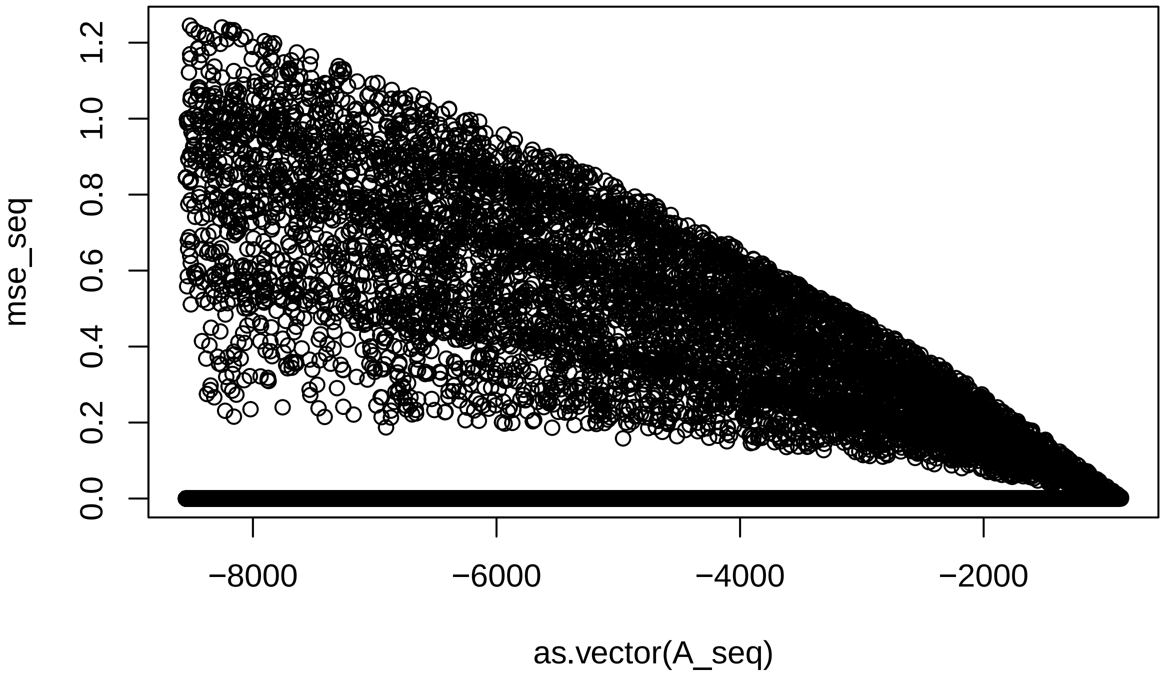
```r
cat("accepted ", signif(100*acc.count/(N_iter),2), "%\n",sep="")
```

```
## accepted 65%
```

```r
#checking the mixing of the distribution
ts.plot(A_seq)
```



```
#looking at the MSE between the MAP and the true one
mse_seq = matrix(0,N_iter,1)
for(i in 20000:N_iter){
  mse_seq[i] = sum((p_seq[,,i] - synth_p)**2)
}

plot(as.vector(A_seq), mse_seq)
```

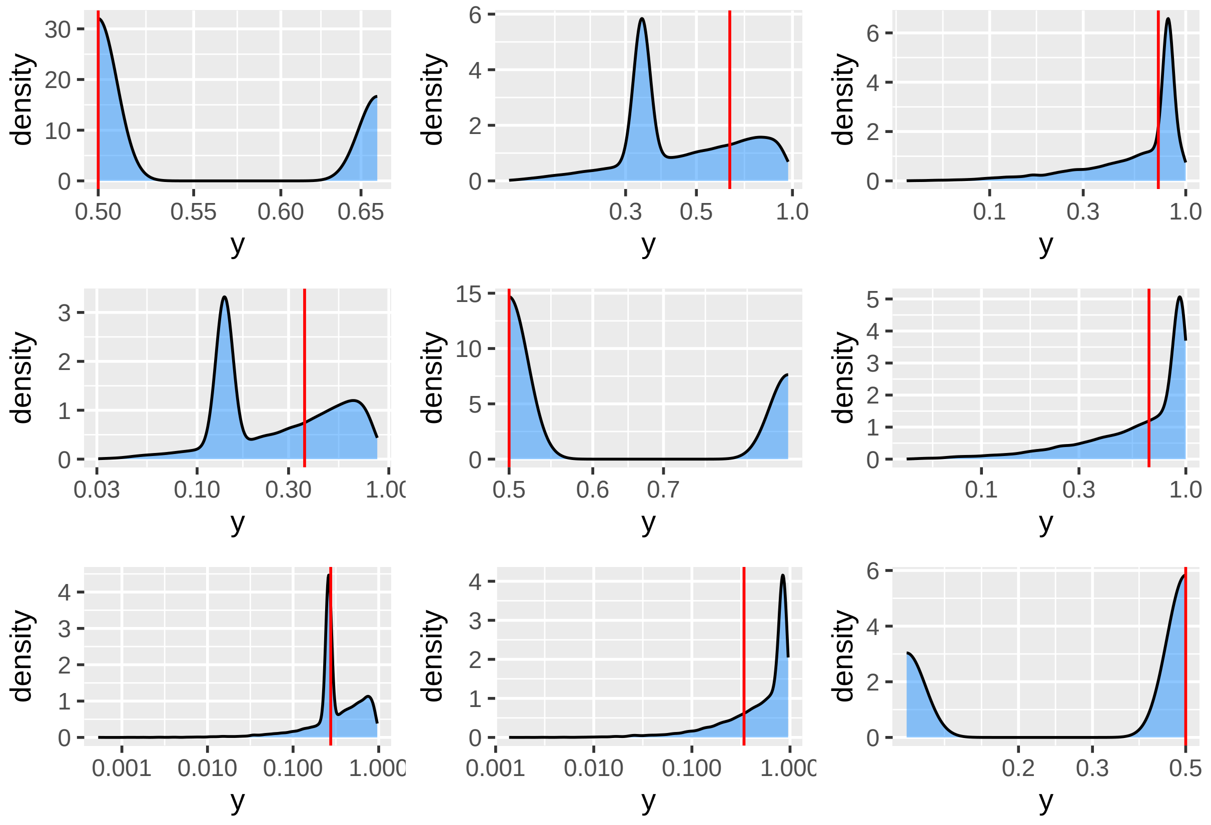


```
burnin_p = p_seq[,,20001:30000]
```

```
sqrt(sum((p_seq[,,which(A_seq==max(A_seq))] - synth_p)**2))
```

```
## [1] 0.05592916
```

```
burnin_p = p_seq[,,20001:30000]

plots = list()
for(i in 1:K_true) {
  for(j in 1:K_true) {
    y_try = data.frame(y = as.vector(burnin_p[i, j,]))
    p1 = ggplot(y_try, aes(y)) +
      geom_density(fill = "dodgerblue", alpha = 0.5) +
      scale_x_log10() +
      geom_vline(xintercept = synth_p[i, j], color = "red")
    plots[[length(plots) + 1]] <- p1
  }
}
p_combined = wrap_plots(plots, ncol = 3, nrow = 3)
p_combined
```



Fixing z and p

In this section, I want to infer K, by fixing p and z equal to the true, known, values . Here below is the code:

```
N_iter = 10000
N=100
```

```r
##Initializing basic quantities----

#interations container
K_seq = rep(NA, N_iter)
#maximum number of blocks allowed
K_max = 6
#quantity updated at each iteration
K_current = sample(1:6,1)
#keeping track of labels
labels_available = seq(1:K_current)

#----

##Actual loop ----
s=0
while(s < N_iter){

  #proposing the add or remove step from a discrete uniform
  add_or_remove=sample.int(2, size=1)

  # if 1 is sampled, then it is add attempt
  add_attempt= add_or_remove==1

  #  ADD only if K_max is not exceeded
  ok1 = K_current<K_max

  # Add accepted with this probability
  ok2 <- K_current/((N+K_current)*(K_current+1)) >= runif(1)

  # we REMOVE only if not less than 1
  ok3 = K_current>=2

  # if there is an empty cluster and remove move is selected, remove it with pr=1
  empty_cluster= any(n_current==0)

  ## insert attempt
  if (ok1 & ok2 & ok3){

    # adding a new label to the existing ones
    K_new = labels_add(LABELS=labels_available)$LABEL_ADDED

    labels_available = labels_add(LABELS=labels_available)$NEW_LABEL_VECTOR

    # update K --- true update
    K_seq[s+1]<-K_current+1
    K_current=K_current+1

    ## adjusting the P matrix

    #Firt case: the new cluster has label 1
    NEW_LABELS_first_position = K_new ==1
    NEW_LABELS_last_position = K_neww ==K
    if(NEW_LABELS_first_position){
      #generating a row vector of length K+1
      new_p  = matrix(0,1,K+1)
      new_p[1] = 0.5
      for(i in (1:K)){
        new_p[i+1] = sample_beta_trunc(1,1,1,max(c(new_p[i],synth_p[1,i])), Inf)}
      p_matrix = cbind((1-new_p[2:(K+1)]),p_matrix)
      new_p = rbind(new_p,p_matrix)
    }else(NEW_LABELS_last_position){
      new_p  = matrix(0,K+1,1)
      new_p[K+1] = 0.5
      for(i in K:1){
        new_p[i] = sample_beta_trunc(1,1,1,max(c(synth_p[i,K],new_p[i+1])), Inf)
      }
      p_matrix = rbind(p_matrix,(1-new_p[1:(K)]))
      new_p = cbind(p_matrix,new_p)
    }


    ## delete attempt
  } else if (!add_attempt & at_least_two & empty_cluster){

    #deliting the empty cluster
    lab_to_delete<-which(n_current==0)[1]
    labels_available<-labels_available[-lab_to_delete]

    # update K --- true update
    K_seq[s+1]<-K_current-1
    K_current=K_current-1
  } else {
    # labels_available are the same
    K_seq[s+1]<-K_current
  }
  s = s+1
}

#----
```

```
p_matrix = synth_p
K=3
new_cluster=3




#Second case: the new cluster has label K

#Third case: the new cluster has label 1<i<K
```