hierarchical bayes model

Lapo Santi

2023-12-07

Hierarchical bayesian model

$$
\begin{align}
&K \sim \text{Pois}(\lambda_K = 1 \mid K > 0) \quad \text{for } 0 < K \leq K^{\max} \tag{1}\\
&a \sim U(0.5, 1) \tag{2}\\
&U_1, \dots, U_K \sim (iid) \quad U(0.5, a) \tag{3}\\
&U_{(k)} \sim f_{U(k)}(u) = \frac{n_0!}{(k-1)!(n_0-k)!}\left(\frac{u-0.5}{a-0.5}\right)^{k-1}\left(\frac{a-u}{a-0.5}\right)^{n_0-k}\frac{1}{a-0.5}, \text{ for } 0.5 \leq u \leq a. \tag{4}\\
&\sigma^2 \sim f(\sigma^2) = \text{TruncExp}(\lambda_{\sigma^2} = 1 \mid 0 < \sigma^2 < U^{\min} \cdot (1 - U^{\min})), \quad \text{where } U^{\min} = \min\left(\mathbf{U}_{(1:K)}\right) \tag{5}\\
&p_{(k)} \sim f(p_{(k)} \mid \mathbf{U}_{(1:K)}, \sigma^2) = Beta(\tilde{\alpha}_k, \tilde{\beta}_k) \quad \text{where } \tilde{\alpha}_k, \tilde{\beta}_k : \begin{cases} \mathbb{E}\left(p_k\right) = U_{(k)} \\ \mathbb{V}ar\left(p_k\right) = \sigma^2 \end{cases} \tag{6}\\
&z_i \sim f(z_i \mid \gamma_1, \dots, \gamma_K) = Mult(n = 1, \gamma_1, \dots, \gamma_K) \tag{7}\\
&\gamma_i \sim f(\gamma_i \mid A_1, \dots, A_K) = Dir(\gamma_i \mid A_1, \dots, A_K) \text{ where } A = \sum_{k=1}^{K} A_k = K \cdot A_k = 1 \tag{8}\\
&y_{ij} \sim f(y_{ij} \mid \boldsymbol{z}, p) = Bin\left(n_{ij}, p_{z_i, z_j}\right) \tag{9}
\end{align}
$$

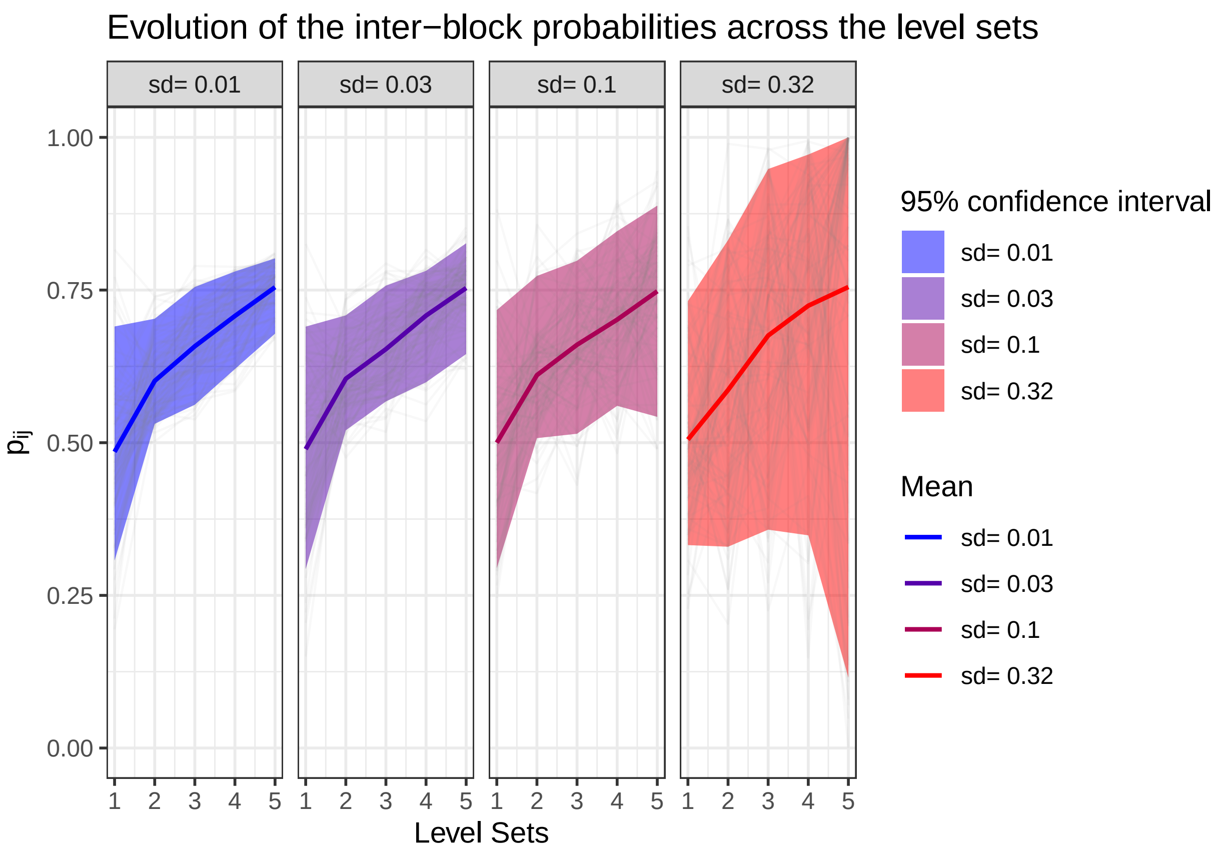
which leads us to:

$$
\begin{align}
&K \sim \text{Pois}(\lambda_K = 1 \mid K > 0) \quad \text{for } 0 < K \leq K^{\max} \tag{10}\\
&a \sim U(0.5, 1) \tag{11}\\
&\mathbf{U}_{(1:K)} \mid a \sim K! \cdot \left(\frac{1}{a-0.5}\right)^K \tag{12}\\
&\sigma^2 \sim f(\sigma^2) = \text{TruncExp}(\lambda_{\sigma^2} = 1 \mid \sigma^2 < (U^{\min} \cdot (1 - U^{\min}))), \quad \text{where } U^{\min} = \min\left(\mathbf{U}_{(1:K)}\right) \tag{13}\\
&p_{(1)}, \dots, p_{(K-1)} \mid \mathbf{U}_{(1:K)}, \sigma^2 \underset{ind}{\sim} \prod_{k=1}^{K-1} Beta(\tilde{\alpha}_k, \tilde{\beta}_k) \tag{14}\\
&z_{(1)}, \dots, z_{(n)} \mid \gamma_1, \dots, \gamma_K \underset{ind}{\sim} \frac{\Gamma(\sum_k n_k + 1)}{\prod_k \Gamma(n_k + 1)} \prod_{k=1}^{K} \gamma_k^{n_k} \tag{15}\\
&\gamma_1, \dots, \gamma_K \mid \alpha_1, \dots, \alpha_K = \frac{1}{\mathrm{B}(\boldsymbol{\alpha})} \prod_{k=1}^{K} \gamma_k^{\alpha_k - 1} \tag{16}\\
&Y \mid \boldsymbol{z}, P \sim \prod_{p=1}^{K-1} \prod_{q=p+1}^{K} p_{pq}^{\bar{y}_{pq}} \cdot (1 - p_{pq})^{\bar{n}_{pq} - \bar{y}_{pq}} \tag{17}
\end{align}
$$

```r
# Print the palettes

my_df <- data.frame(level_set = 0, mean_l = 0, perc5 = 0, perc95=0, sd = 0,sample_number=0)
my_combinations = c(0.0001,0.001,0.01,0.1)

for(var_i in 1:length(my_combinations)){
  n_samples=100
  for(sample in 1:n_samples){
    K=5
    a = .80
    U = runif(5,0.5,a)
    x = rgamma(1 ,shape = 1,scale = 1)
    sigma_squared = my_combinations[var_i]
    U_k = sort(U)

    beta_params = beta_mean_var(U_k,rep(sigma_squared,K) )
    a_k = beta_params$alpha
    b_k = beta_params$beta



    P_array = array(0,dim=c(K,K,1))
    P = matrix(0,K,K)
    for(k in 0:(K-1)){
      for(i in 1:(K-1)){
        for(j in (i+1):K){
          if((j-i)==k){
            P[i,j]= rbeta(1,a_k[k+1],b_k[k+1])
          }
        }
      }
    }


    P = P +  lower.tri(P)*(1-t(P))
    diag(P)<- rbeta(K,1,1)
    P_array[,,1]<- P



    gen_samples = generalized_levels(P_array,K = K,N = 1,diag0.5 = F)



    for(e in 1:length(gen_samples)){
      my_df_i<- data.frame(level_set = e,
                           mean_l = mean(gen_samples[[e]]),
                           perc5 = quantile(gen_samples[[e]],.05),
                           perc95 = quantile(gen_samples[[e]],.95),
                           sd = paste('sd=',round(sqrt(sigma_squared),2)),
                           sample_number = sample)

      my_df <- rbind(my_df,my_df_i)
    }
  }
}

my_df <- my_df[-1,]




my_df_sum <- my_df %>% group_by(level_set, sd) %>% summarize(mean= mean(mean_l), perc5 = quantile(mean_l
```

```
## `summarise()` has grouped output by 'level_set'. You can override using the
## `.groups` argument.
```

```
my_df_m<- my_df %>% left_join(my_df_sum, by= c('sd','level_set'))

# Number of colors in the palette
num_colors = length(my_combinations)

# Blue to Red color palette with alpha=0
palette_alpha_0 <- colorRampPalette(c("blue", "red"))(num_colors)

# Blue to Red color palette with alpha=0.5
palette_alpha_0.5 <- adjustcolor(palette_alpha_0, alpha.f = 0.5)

# plotting the level sets
ggplot(my_df_m, aes(x = level_set, y = mean_l, group = sample_number)) +
  facet_grid(~sd) +
  geom_ribbon(aes(ymin = perc5.y, ymax = perc95.y,fill=sd, group=sd)) +
  geom_path(color='gray50', linewidth = 0.4, alpha = 0.05) +
  geom_line(aes(y = mean,color=sd,group=sd),linewidth = .8) +
  scale_color_manual(values = palette_alpha_0) +
  scale_fill_manual(values = palette_alpha_0.5) +
  theme_bw()+
  labs(title = "Evolution of the inter-block probabilities across the level sets",  # Add your desired
       x = "Level Sets",  # Add your desired X-axis label
       y = expression(p[ij]),  # Add your desired Y-axis label
       fill = "95% confidence interval",
       colour = 'Mean')
```

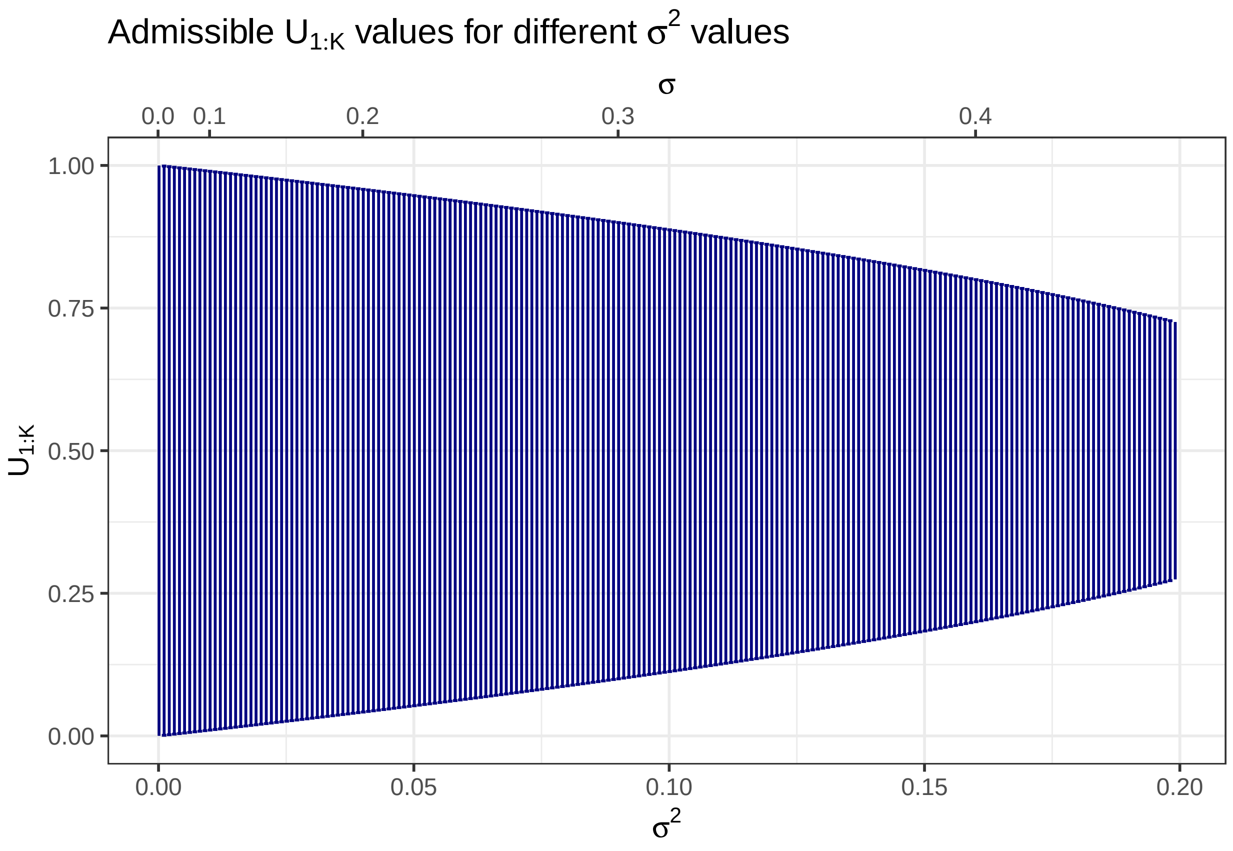


Here we explore the relationship between $\mathbf{U}_{(1:K)}$ and σ^2. Since we know that $\sigma^2 < \mathbf{U}_{(1:K)}(1 - \mathbf{U}_{(1:K)})$, this implies that

$$\frac{1}{2}(1 - \sqrt{1 - 4\sigma^2} < \mathbf{U}_k < \frac{1}{2}(1 + \sqrt{1 - 4\sigma^2} \quad \text{for } k = 1, \ldots, K - 1$$

```
#several variances
sigma_squared_combinations= seq(0.0001,0.2,0.001)

#lower bound function
lb_f = function(sigma_squared){
  if(sigma_squared>0){
    lb<- 0.5*(1-sqrt(1-4*sigma_squared))
    return(lb)}
}
#upper bound function
ub_f = function(sigma_squared){
  if(sigma_squared>0){
    ub<- 0.5*(1+sqrt(1-4*sigma_squared))
    return(ub)}
}

sigma_U_rel_df <- data.frame(
  sigma = sqrt(sigma_squared_combinations),
  sigma_squared = sigma_squared_combinations,
  lb = as.numeric(lapply(sigma_squared_combinations, lb_f)),
  ub = as.numeric(lapply(sigma_squared_combinations, ub_f))
)
#this function is needed to avoid the ggplor error: function non monotonic
```

```r
my_f <- Vectorize(function(x) {
  if (x < 0) return(x/1e10)
  sqrt(x)
})


ggplot(sigma_U_rel_df, aes(x = sigma_squared, ymin = lb, ymax = ub)) +
  geom_errorbar(color = 'navy') +
  theme_bw() +
  labs(
    title = expression(paste("Admissible ", U[1:K], " values for different ", sigma^2 ,' values')),
    x = expression(sigma^2),
    y = expression(U[1:K])
  ) +
  scale_x_continuous(  limits = c(0,max(sigma_squared_combinations)),
                       sec.axis = sec_axis(~ my_f(.), name =  expression(sigma))
  )
```



Collapsing P parameter

We have that the likelihood, rewritten over blocks, takes the following expression:

$$f_Y(y \mid z, p) = \prod_{p=1}^{K} \prod_{q=1}^{K} \bar{\lambda}_{pq} p_{pq}^{\bar{y}_{pq}} \left(1 - p_{pq}\right)^{\bar{m}_{pq}} \tag{18}$$

where

$$\bar{y}_{pq} = \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} y_{ij} \quad \mathbb{I}(z_i = p)\, \mathbb{I}(z_j = q) \tag{19}$$

$$\bar{m}_{pq} = \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} (n_{ij} - y_{ij})\, \mathbb{I}(z_i = p)\, \mathbb{I}(z_j = q) \tag{20}$$

$$\bar{\lambda}_{pq} = \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} \begin{pmatrix} n_{ij} \\ y_{ij} \end{pmatrix} \mathbb{I}(z_i = p)\, \mathbb{I}(z_j = q) \tag{21}$$

where $\bar{y}_{pq}$ is the number of links between block p and block q, $\bar{m}_{pq}$ is the number of missing links between the two blocks, and finally $\bar{\lambda}_{pq}$ computes the number of ways in which we could have obtained that number of links out of the total amount of possibilities.

The problem with (18) is that the product runs over all possible pairs of blocks. Instead, we would like to have the product spanning just the upper triangular P entries to obtain an expression that would allow us to integrate out the parameter P.

We start by rewriting (18) as follows:

$$\begin{aligned} f_Y(y \mid z, p) &= \prod_{p=1}^{K} \prod_{q=1}^{K} \bar{\lambda}_{pq} p_{pq}^{\bar{y}_{pq}} (1 - p_{pq})^{\bar{m}_{pq}} && (22) \\ &= \prod_{p=1}^{K} \prod_{q=p}^{K} \bar{\lambda}_{pq} p_{pq}^{\bar{y}_{pq}} (1 - p_{pq})^{\bar{m}_{pq}} \cdot \prod_{p=2}^{K} \prod_{q=1}^{K-1} \bar{\lambda}_{pq} p_{pq}^{\bar{m}_{qp}} (1 - p_{pq})^{\bar{y}_{qp}} && (23) \\ &= \prod_{p=1}^{K} \bar{\lambda}_{pp} p_{pp}^{\bar{y}_{pp}} (1 - p_{pp})^{\bar{m}_{pp}} \cdot \prod_{p=1}^{K-1} \prod_{q=p+1}^{K} \bar{\lambda}_{pq} \cdot \bar{\lambda}_{qp} p_{pq}^{\bar{y}_{pq} + \bar{m}_{qp}} (1 - p_{pq})^{\bar{m}_{pq} + \bar{y}_{qp}} && (24) \end{aligned}$$

since $p_{pq} = 1 - p_{qp}$

The prior on the P entries takes the following expression

$$f_p\left(p_{pq} \mid \tilde{\alpha}_k, \tilde{\beta}_k, k\right) = \begin{cases} \prod_{p=1}^{K-1} \prod_{q=p+1}^{K} \frac{1}{\mathrm{Beta}\left(\tilde{\alpha}_k, \tilde{\beta}_k\right)} p_{pq}^{\tilde{\alpha}_k - 1} (1 - p_{pq})^{\tilde{\beta}_k - 1} & \text{for } 0 < q - p < K - 1 \\ \prod_{p=q} Beta(1,1) = [Beta(1,1)]^K & \text{for } q - p = 0 \end{cases} \tag{25}$$

for $\tilde{\alpha}_k > 0, \tilde{\beta}_K > 0$, where for $q - p > 0$

$$\tilde{\alpha}_k = \left(\frac{1 - U_{(k)}}{\sigma^2} - \frac{1}{U_k} \right) \cdot \mu^2 \tag{26}$$

$$\tilde{\beta}_k = \tilde{\alpha}_k \cdot \left(\frac{1}{U_{(k)}} - 1 \right) \tag{27}$$

so that

$$\mathbb{E}\left(p^{(l)}\right) = U_{(k)} \tag{28}$$

$$\mathrm{Var}\left(p^{(l)}\right) = \sigma^2 \tag{29}$$

where $p^{(l)} := \{p_{pq} : q - p = l \quad \text{for } l = 1, \dots, K-1\}$, denoted as *Level Set*.

Instead, for $q - p = 0$, that is over $p^{(0)}$, we set $\tilde{\alpha}_k = \tilde{\beta}_k = 1$, so that the distribution of the level set $p^{(0)}$, that is, the diagonal of the P matrix is a uniform distribution with

$$\mathbb{E}\left[p^{(0)}\right] = 1/2 \tag{30}$$

$$\mathbb{V}ar\left[p^{(0)}\right] = 1/12 \tag{31}$$

The, using the beta-binomial conjugacy, we marginalize out P, obtaining:

$$f_Y\left(Y \mid \mathbf{z}, \mathbf{U}_{(1),\dots,(K)}, \sigma^2, a\right) = \prod_{p=1}^{K-1} \prod_{q=p+1}^{K-1} \left(\bar{\lambda}_{pq}\bar{\lambda}_{qp} \frac{B\left(\tilde{\alpha}_k + \bar{y}_{pq} + \bar{m}_{qp}, \tilde{\beta}_k + \bar{m}_{pq} + \bar{y}_{qp}\right)}{B\left(\tilde{\alpha}_k, \tilde{\beta}_k\right)} \right) \prod_{p=1}^{K} \bar{\lambda}_{pp} \frac{B\left(1 + \bar{y}_{pp}, 1 + \bar{m}_{pp}\right)}{B\left(1, 1\right)}$$

where we use the notation $f_Y\left(y \mid z, \mathbf{U}_{(1),\dots,(K)}, \sigma^2, a\right)$ to explicit the dependency of $\tilde{\alpha}_k, \tilde{\beta}_k$ upon $\{\mathbf{U}_{(1),\dots,(K)}, \sigma^2, \alpha\}$.

To compute it, we use the following function:

```r
llike_integrated_out = function(lambdabar, ybar,mbar,K,alpha_k, beta_k){
  llik= matrix(0,K,K)
  for(p in 1:K){
    for(q in p:K){
      if(q-p != 0){
        llik[p,q]<-  log(lambdabar[p,q]) + log(lambdabar[q,p]) +
          lbeta(alpha_k[q-p] + ybar[p,q]+ mbar[q,p],beta_k[q-p] +
                  ybar[q,p]+ mbar[q,p]) - lbeta(alpha_k[q-p], beta_k[q-p])
      }else{
        llik[p,p]<-  log(lambdabar[p,q]) +lbeta(1 + ybar[p,q],1
                                            + ybar[q,p]) - lbeta(1, 1)
      }
    }
  }

  return(sum(llik))
}
```

Collapsing the prior on γ

From this model we can write the joint distribution of $(\boldsymbol{z}, \boldsymbol{\gamma})$ as:

$$\begin{aligned} f(\boldsymbol{z}, \boldsymbol{\gamma}) &= f(\boldsymbol{z}|\boldsymbol{\gamma}) f(\boldsymbol{\gamma}|\boldsymbol{\alpha}) \\ &= \frac{1}{B(\boldsymbol{\alpha})} \gamma^{n_1}\gamma^{n_2}\dots\gamma^{n_K} \prod_{i=1}^{K} (\gamma_i^{\alpha_i - 1}) \\ &= \frac{1}{B(\boldsymbol{\alpha})} \prod_{i=1}^{K} (\gamma_i^{n_i + \alpha_i - 1}) \end{aligned} \tag{32}$$

where $n_k = \sum_{i=1}^{N} \mathbb{I}\left(z_i = k\right)$ is the number of items in block k. Now, we can marginalize out $\boldsymbol{\gamma}$ exploiting the Dirichlet-Categorial conjugacy

$$
\begin{aligned}
f_{\boldsymbol{z}}(\boldsymbol{z} \mid \boldsymbol{\alpha}) &= \int \mathrm{p}(\boldsymbol{z}, \boldsymbol{\gamma})\boldsymbol{\gamma} \\
&= \frac{\sum_{i=1}^{K}\Gamma(\alpha_i)}{\prod_{i=1}^{K}\Gamma(\alpha_i)} \int \gamma_i^{n_i+\alpha_i-1} d\gamma \qquad (33) \\
&= \frac{\sum_{i=1}^{K}\Gamma(\alpha_i)}{\prod_{i=1}^{K}\Gamma(\alpha_i)} \frac{\prod_{i=1}^{K}\Gamma(\alpha_i+n_i)}{\Gamma(\sum_{i=1}^{K}\alpha_i+n_i)} \\
&= \frac{\Gamma(\sum_{i=1}^{K}\alpha_i)}{\Gamma(\sum_{i=1}^{K}\alpha_i+n_i)} \prod_{i=1}^{K} \frac{\Gamma(\alpha_i+n_i)}{\Gamma(\alpha_i)} \\
&= \frac{\Gamma(A)}{\Gamma(A+N)} \prod_{i=1}^{K} \frac{\Gamma(\alpha_i+n_i)}{\Gamma(\alpha_i)} \qquad (34)
\end{aligned}
$$

where we have recognised in (33) the kernel of a Dirichlet$(n_i+\alpha_i)$ and where (34) is known as Dirichlet-Multinomial distribution and we denote it as $DM(\boldsymbol{\alpha})$

The full proportional posterior distribution

$$f\left(\boldsymbol{z}, \sigma^2, a, \mathbf{U}_{(1:K)}, K \mid Y\boldsymbol{\alpha}\right) \propto \qquad (35)$$

$$f\left(Y \mid \boldsymbol{z}, \sigma^2, a, \mathbf{U}_{(1:K)}, K\right) \cdot f(\boldsymbol{z} \mid \boldsymbol{\alpha}) \cdot f\left(\mathbf{U}_{(1:K)} \mid a, K\right) \cdot f\left(\sigma^2 \mid U^{\min}\right) \cdot f(a) \cdot f(K) = \qquad (36)$$

$$\underbrace{\prod_{p=1}^{K-1}\prod_{q=p+1}^{K-1}\left(\bar{\lambda}_{pq}\bar{\lambda}_{qp}\frac{B\left(\tilde{\alpha}_k+\bar{y}_{pq}+\bar{m}_{qp}, \tilde{\beta}_k+\bar{m}_{pq}+\bar{y}_{qp}\right)}{B\left(\tilde{\alpha}_k, \tilde{\beta}_k\right)}\right)\prod_{p=1}^{K}\bar{\lambda}_{pp}\frac{B\left(1+\bar{y}_{pp}, 1+\bar{m}_{pp}\right)}{B\left(1,1\right)}}_{f\left(Y\mid\boldsymbol{z},\sigma^2,a,\mathbf{U}_{(1:K)},K\right)} \cdot \underbrace{\frac{\Gamma(\boldsymbol{\alpha})}{\Gamma(\boldsymbol{\alpha}+n)}\prod_{i=1}^{K}\frac{\Gamma(\alpha_i+n_i)}{\Gamma(\alpha_i)}}_{f(\boldsymbol{z}\mid\boldsymbol{\alpha})} \qquad (37)$$

$$\underbrace{\cdot K! \cdot \left(\frac{1}{a-0.5}\right)^K}_{f\left(\mathbf{U}_{(1:K)}\mid a,K\right)} \cdot \underbrace{\frac{e^{-\sigma^2}}{1-\exp\left(-U^{\min}(1-U^{\min}\right)}}_{f(\sigma^2)} \underbrace{\frac{1}{2}}_{f(a)} \cdot \underbrace{\frac{1}{K!(e-1)}}_{f(K)} \qquad (38)$$

Passing to the log:

$$
\begin{aligned}
&\log\left(f\left(\boldsymbol{z}, \sigma^2, a, \mathbf{U}_{(1:K)}, K \mid Y\boldsymbol{\alpha}\right)\right) \propto \\
&\sum_{p=1}^{K-1}\sum_{q=p+1}^{K-1}\left(\log(\bar{\lambda}_{pq}) + \log(\bar{\lambda}_{qp})\log(B\left(\tilde{\alpha}_k+\bar{y}_{pq}+\bar{m}_{qp}, \tilde{\beta}_k+\bar{m}_{pq}+\bar{y}_{qp}\right)) - \log(B\left(\tilde{\alpha}_k, \tilde{\beta}_k\right))\right) + \\
&\sum_{p=1}^{K}(\log(\bar{\lambda}_{pp}) + \log\left(B\left(1+\bar{y}_{pp}, 1+\bar{m}_{pp}\right)\right) - \log\left(B\left(1,1\right)\right)) + \log(\Gamma(\boldsymbol{\alpha})) - \log(\Gamma(\boldsymbol{\alpha}+n)) + \qquad (39) \\
&\sum_{i=1}^{K}\left[\log\left(\Gamma\left(\alpha_i+n_i\right)\right) - \log\left(\Gamma\left(\alpha_i\right)\right)\right] + \log(K!) + K\cdot(-\log(a-0.5)) - \sigma^2 - \\
&\log\left(1-\exp\left(-U^{\min}(1-U^{\min})\right)\right) + \log(1/2) - \log(K!) - \log(e-1)
\end{aligned}
$$

Here I write the code to evaluate the function above (39).

```
lprop_posterior <- function(lambdabar, ybar,mbar, a,
                            alpha_vec, n_k,sigma_squared, U_vec,K){

  beta_params = beta_mean_var(U_vec,rep(sigma_squared,K) )
  alpha_k = beta_params$alpha
  beta_k = beta_params$beta



  sum1= llike_integrated_out(lambdabar, ybar,mbar,K,alpha_k,beta_k)
  #prior on z
  sum2 <- ddirichlet_multinomial(N,K,n_k, alpha_vec)
  #prior on U
  sum3 <- lfactorial(K)+K*(-log(a-0.5))
  #prior on sigma^2
  sum4 <- -sigma_squared - log(1-exp(min(U*(1-U))))
  #prior on K
  sum5 <- -lfactorial(K) - log(exp(1) -1)
  #pior on a
  sum6 <- 1/2
  result <- sum1 + sum2 + sum3 + sum4 + sum5 + sum6
  return(result)
}
```

Inference

Update z

Regarding the z parameter:

Update σ^2

Below we report the algorithm employed to update σ^2

Here is the function employed for its computation:

```
#code for the sigma^2 update step within the MH MCMC algorithm

sigma_squared_update = function(lambdabar,ybar,mbar, a, alpha_vec, n_k,
                                sigma_squared, U_vec,K, theta_sigma_squared,
                                acc.count_sigma_squared){

  U_max = max(U_vec)

  #simulating (sigma^2)' from a g ~ truncated normal
  sigma_squared_prime <- rtruncnorm(1,a = 0,b = 0.9,
                                    mean = sigma_squared,
                                    sd = theta_sigma_squared**2)

  #computing the proportional posterior in (sigma^2)'
  prop_posterior_prime <- lprop_posterior(lambdabar, ybar,mbar, a, alpha_vec,
                                          n_k,sigma_squared_prime, U_vec,K)

  #evaluating the proportional posterior in (sigma^2)^(t)
```

Algorithm 1 Metropolis-within-Gibbs update for σ^2

```
Given  $\boldsymbol{z}^{(t)}, (\sigma^2)^{(t)}, a^{(t)}, \mathbf{U}^{(t)}_{(1:K)}, K^{(t)}, Y, N, \boldsymbol{\alpha}$

1.  Sample
```

$$(\sigma^2)' \sim g\left((\sigma^2)^{(t)}, (\tau^2_{\sigma^2})^{(t)} \mid 0, U^{\min}\right) \tag{40}$$

$$= \text{TruncNorm}\left(\mu = (\sigma^2)^{(t)}, \sigma^2 = (\tau^2_{\sigma^2})^{(t)}, lb = 0, ub = U^{\min}(1 - U_{\min})\right) \tag{41}$$

```
2.  Take
```

$$(\sigma^2)^{(t+1)} = \begin{cases} (\sigma^2)^{(t)} & \texttt{with probability } \; 1 - r', \\ (\sigma^2)' & \texttt{with probability } \; r', \end{cases}$$

```
where
```

$$r' = \log(1) \wedge$$

$$\log\left(f\left(\boldsymbol{z}^{(t)}, (\sigma^2)', a^{(t)}, \mathbf{U}^{(t)}_{(1:K)}, K^{(t)} \mid Y \boldsymbol{\alpha}\right)\right) + \log\left(g((\sigma^2)^{(t)})\right) - \tag{42}$$

$$\log\left(f\left(\boldsymbol{z}^{(t)}, (\sigma^2)^{(t)}, a^{(t)}, \mathbf{U}^{(t)}_{(1:K)}, K^{(t)} \mid Y \boldsymbol{\alpha}\right)\right) - \log\left(g((\sigma^2)')\right) \tag{43}$$

```r
  prop_posterior_current<- lprop_posterior(lambdabar, ybar,
                                           mbar, a, alpha_vec, n_k,
                                           sigma_squared_current, U_vec,K)

  #evaluating the proposal density g(sigma^2)')
  log_proposal_prime <- log(dtruncnorm(sigma_squared_prime,a = 0,b = 0.9,
                                     mean = sigma_squared,
                                     sd = theta_sigma_squared**2))

  #evaluating the proposal density g(sigma^2)^(t))
  log_proposal_current <- log(dtruncnorm(sigma_squared,a = 0,b = 0.9,
                                       mean = sigma_squared,
                                       sd = theta_sigma_squared**2))
  #acceptance ratio
  log_r=  prop_posterior_prime + log_proposal_current -
    prop_posterior_current - log_proposal_prime

  #create statements that check conditionion to accept move
  MH_condition_S= min(log_r,0)>=log(runif(1))
  if(MH_condition_S){
    acc.count_sigma_squared = acc.count_sigma_squared+1
    sigma_squared <- sigma_squared_prime
  }


  return(list(acc.moves = acc.count_sigma_squared,
              sigma_squared=sigma_squared))

}
```

Update a

Below we report the algorithm employed to update a

Algorithm 2 Metropolis-within-Gibbs update for a

```
Given  $\boldsymbol{z}^{(t)}, (\sigma^2)^{(t)}, a^{(t)}, \mathbf{U}^{(t)}_{(1:K)}, K^{(t)}, Y, N, \boldsymbol{\alpha}$

1.  Sample
```

$$a' \sim g\left(a^{(t)}, (\tau_a^2)^{(t)} \mid 0, 1\right) \tag{44}$$

$$= \text{TruncNorm}\left(\mu = (a^{(t)}, \sigma^2 = (\tau_a^2)^{(t)}, lb = 0, ub = 1\right) \tag{45}$$

```
2.  Take
```

$$a^{(t+1)} = \begin{cases} a^{(t)} & \texttt{with probability} \quad 1 - r', \\ a' & \texttt{with probability} \quad r', \end{cases}$$

```
where
```

$$r' = \log(1) \wedge$$
$$\log\left(f\left(\boldsymbol{z}^{(t)}, (\sigma^2)^{(t)}, a', \mathbf{U}^{(t)}_{(1:K)}, K^{(t)} \mid Y\boldsymbol{\alpha}\right)\right) + \log\left(g(a^{(t)})\right) - \tag{46}$$
$$\log\left(f\left(\boldsymbol{z}^{(t)}, (\sigma^2)^{(t)}, a^{(t)}, \mathbf{U}^{(t)}_{(1:K)}, K^{(t)} \mid Y\boldsymbol{\alpha}\right)\right) - \log\left(g(a')\right) \tag{47}$$

Here is the function employed for its computation:

```r
#code for the a parameter update step within the MH MCMC algorithm

a_update = function(lambdabar, ybar,mbar, a, alpha_vec, n_k,
                    sigma_squared, U_vec,K, theta_sigma_squared,
                    acc.count_a){



  #simulating (sigma^2)' from a g ~ truncated normal
  a_prime <- rtruncnorm(1,a = 0.5,b = 1,
                        mean = a,
                        sd = theta_a**2)

  #computing the proportional posterior in a'
  prop_posterior_prime <- lprop_posterior(lambdabar, ybar,mbar, a_prime,
                                           alpha_vec, n_k,sigma_squared,
                                           U_vec,K)

  #evaluating the proportional posterior in a^(t)
  prop_posterior_current<- lprop_posterior(lambdabar, ybar,
                                            mbar, a, alpha_vec, n_k,
                                            sigma_squared, U_vec,K)

  #evaluating the proposal density g(a')
  log_proposal_prime <- log(dtruncnorm(a_prime,a = 0,b = 0.9,
                                       mean = sigma_squared,
                                       sd = theta_sigma_squared**2))
  #evaluating the proposal density g(sigma^2)^(t))
  log_proposal_current <- log(dtruncnorm(1,a = 0,b = 0.9,
                                         mean = a,
                                         sd = theta_a**2))
  #acceptance ratio
  log_r=  prop_posterior_prime + log_proposal_current -
    prop_posterior_current - log_proposal_prime

  #create statements that check conditionion to accept move
  MH_condition_S= min(log_r,0)>=log(runif(1))
  if(MH_condition_S){
    acc.count_a = acc.count_a+1
    a <- a_prime
  }


  return(list(acc.moves = acc.count_a,
              sigma_squared=sigma_squared))

}
```

Update U

Below we report the algorithm employed to update $\mathbf{U}^{(t)}_{(1:K)}$

Here is the function employed for its computation:

```
#code for the U parameter update step within the MH MCMC algorithm

U_update = function(lambdabar, ybar,mbar, a, alpha_vec, n_k,
                    sigma_squared, U_vec,K, theta_sigma_squared,
                    acc.count_a){


  for(k in 2:(K+1)){

    U_ext <- c(0.5,U_vec,a)
    U_vec_prime <- U_vec

    #simulating (sigma^2)' from a g ~ truncated normal
    U_prime <- rtruncnorm(1,a = U_ext[k-1],b = U_ext[k+1],
                          mean = U_vec[k],
                          sd = theta_U[k]**2)

    U_vec_prime[k] <- U_prime
    #computing the proportional posterior in a'
    prop_posterior_prime <- lprop_posterior(lambdabar, ybar,mbar, a,
                                            alpha_vec, n_k,sigma_squared,
                                            U_vec_prime,K)

    #evaluating the proportional posterior in a^(t)
    prop_posterior_current<- lprop_posterior(lambdabar, ybar,
                                             mbar, a, alpha_vec, n_k,
                                             sigma_squared, U_vec,K)
```

Algorithm 3 Metropolis-within-Gibbs update for $\mathbf{U}_{(1:K)}^{(t)}$

```
Given  $\boldsymbol{z}^{(t)}, (\sigma^2)^{(t)}, a^{(t)}, \mathbf{U}_{(1:K)}^{(t)}, K^{(t)}, Y, N, \boldsymbol{\alpha}$
for k = 1:K do
    1.  Sample

        $U'_{(k)} \sim g_l = \begin{cases} g_1\left(U_{(1)}^{(t)}, (\tau^2_{U_{(1)}})^{(t)} \mid 0.5, U_{(2)}^{(t)}\right) & \text{if } k = 1 \\ g_2\left(U_{(k)}^{(t)}, (\tau^2_{U_{(k)}})^{(t)} \mid U_{(k-1)}^{(t)}, U_{(k+1)}^{(t)}\right) & \text{if } 1 < k < K \\ g_3\left(U_{(K)}^{(t)}, (\tau^2_{U_{(K)}})^{(t)} \mid U_{(K-1)}, a\right) & \text{if } k = K \end{cases}$   (48)

        $= \begin{cases} \text{TruncNorm}\left(\mu = U_{(1)}^{(t)}, \sigma^2 = (\tau_1^2)^{(t)}, lb = 0.5, ub = U_{(2)}^{(t)}\right) & \text{for } k = 1 \\ \text{TruncNorm}\left(\mu = U_{(k)}^{(t)}, \sigma^2 = (\tau_k^2)^{(t)}, lb = U_{(k-1)}^{(t)}, ub = U_{(k+1)}^{(t)}\right) & \text{for } 1 < k < K \\ \text{TruncNorm}\left(\mu = U_{(K)}^{(t)}, \sigma^2 = (\tau_K^2)^{(t)}, lb = U_{(k-1)}^{(t)}, ub = U_{(K)}^{(t)}\right) & \text{for } k = K \end{cases}$   (49)

    2.  Take

        $U_{(k)}^{(t+1)} = \begin{cases} U_{(k)}^{(t)} & \texttt{with probability} \quad 1 - r', \\ U'_{(k)} & \texttt{with probability} \quad r', \end{cases}$

    where

        $r' = \log(1) \wedge$
        $\log\left(f\left(\boldsymbol{z}^{(t)}, (\sigma^2)^{(t)}, a^{(t)}, \mathbf{U}_{(k_j<k)}^{(t+1)}, \mathbf{U}'_{(k)}, \mathbf{U}_{(k_j>k)}^{(t)}, K^{(t)} \mid Y\boldsymbol{\alpha}\right)\right) + \log\left(g_l(U_k^{(t)})\right) -$   (50)
        $\log\left(f\left(\boldsymbol{z}^{(t)}, (\sigma^2)^{(t)}, a^{(t)}, \mathbf{U}_{(k_j<k)}^{(t+1)}, \mathbf{U}_{(k:K)}^{(t)}, K^{(t)} \mid Y\boldsymbol{\alpha}\right)\right) - \log\left(g_l(U'_k)\right)$   (51)

end for
```

```
    #evaluating the proposal density g(a')
    log_proposal_prime <- log(dtruncnorm(U_prime,a = U_ext[k-1],b = U_ext[k+1],
                                         mean = U_vec[k],
                                         sd = theta_U[k]**2))

    #evaluating the proposal density g(sigma^2)^(t))
    log_proposal_current <- log(dtruncnorm(U_vec[k],a = U_ext[k-1],b = U_ext[k+1],
                                           mean = U_vec[k],
                                           sd = theta_U**2))
    #acceptance ratio
    log_r=  prop_posterior_prime + log_proposal_current -
      prop_posterior_current - log_proposal_prime

    #create statements that check conditionion to accept move
    MH_condition_S= min(log_r,0)>=log(runif(1))
    if(MH_condition_S){
      acc.count_a = acc.count_a+1
      U_vec <- U_prime
    }
  }

  return(list(acc.moves = acc.count_a,
              U_vec=U_vec))

}
```

Update K

Appendix

Checking that rewriting the likelihood over blocks is correct

First, we are simulating some data and compute the likelihood with the product ranging over individuals $i < j = 1, \ldots, n$

```
# Binomial SBM

#Simulating the from the prior P -------
set.seed(12)
n=20
K=4
a = .70
U = runif(4,0.5,a)
x = rgamma(1 ,shape = 1,scale = 1)
sigma_squared = 0.001
U_k = sort(U)

beta_params = beta_mean_var(U_k,rep(sigma_squared,K) )
a_k = beta_params$alpha
b_k = beta_params$beta

N<- matrix(2,n,n)
diag(N)<-0
```

```
P = matrix(0,K,K)
for(k in 0:(K-1)){
  for(i in 1:(K-1)){
    for(j in (i+1):K){
      if((j-i)==k){
        P[i,j]= rbeta(1,a_k[k+1],b_k[k+1])
      }
    }
  }
}

P = P +  lower.tri(P)*(1-t(P))
diag(P)=0.5

#simulating z
z = matrix(0,n,1)
z<- sample(1:K, n,replace=T)

z_P<- vec2mat_0_P(z,P)

P_nbyn<- calculate_victory_probabilities(z_P,P)

#simulating N
Y_binom <- matrix(0, n,n)
for(i in 1:n){
  for(j in 1:n){
    Y_binom[i,j]<- rbinom(1,N[i,j],P_nbyn[i,j])
  }
}

Y_binom[lower.tri(Y_binom)] = N[lower.tri(N)] - t(Y_binom)[lower.tri(Y_binom)]
diag(Y_binom)<- 0

#--------------

log_lik_f_binom = function(N,Y,z,P, directed=T){
  z_P<- vec2mat_0_P(z,P)
  P_nbyn<- calculate_victory_probabilities(z_P, P)
  if(directed==T){
    #computing the pairwise log-probabilitiees
    bigM = lchoose(N,Y)+(Y* log(P_nbyn)+(N-Y)*log(1 - P_nbyn))
    #remember to subtract the diagonal
    log_lik= sum(bigM) - sum(diag(bigM))
  }else if(directed==F){
    bigM = lchoose(N,Y)+(Y* log(P_nbyn)+(N-Y)*log(1 - P_nbyn))
    #remember to subtract the diagonal
    log_lik= sum(bigM*upper.tri(bigM))
  }
  return(log_lik)
}

# number of victories between block p and block q
ybar_binom1 = t(z_P)%*%(Y_binom*upper.tri(Y_binom))%*%z_P
```

```r
# number of missed victories between block p and block q
n_minus_y1 <- (N-Y_binom)*upper.tri(N)
# number of missed victories between block p and block q
mbar_binom1<- t(z_P)%*%n_minus_y1%*%z_P

coef1 = lchoose(N, Y_binom)*upper.tri(N)
bin_coef1 <- t(z_P)%*%(coef1)%*%z_P


llik_over_blocks_f_binomial = function(bin_coef, ybar, mbar, P){
  #llik<- sum(bin_coef+ ybar*log(P) + (mbar)*log(1-P))
  llik = matrix(0,K,K)
  for(p in 1:K){
    for(q in p:K){
      if(q-p==0){
        llik[p,q] = bin_coef[p,q]+ ybar[p,q]*log(P[p,q]) + mbar[p,q]*log(1-P[p,q])
      }else{
        llik[p,q] = bin_coef[q,p]+ bin_coef[p,q]+(ybar[p,q]+mbar[q,p])*log(P[p,q]) + (mbar[p,q]+ybar[q,
      }
    }
  }
  return(sum(llik))
}

llik_over_blocks_f_binomial(bin_coef1 ,ybar_binom1, mbar_binom1,P)
```

```
## [1] -183.5635
```

```r
log_lik_f_binom(N,Y_binom,z,P,directed=F)
```

```
## [1] -183.5635
```

```r
llike_integrated_out = function(lambdabar, ybar,mbar,K,alpha_k, beta_k){
  llik= matrix(0,K,K)
  for(p in 1:K){
    for(q in p:K){
      if(q-p != 0){
        llik[p,q]<-  log(lambdabar[p,q]) + log(lambdabar[q,p]) +
          lbeta(alpha_k[q-p] + ybar[p,q]+ mbar[q,p],beta_k[q-p] +
                  ybar[q,p]+ mbar[q,p]) - lbeta(alpha_k[q-p], beta_k[q-p])
      }else{
        llik[p,p]<-  log(lambdabar[p,q]) +lbeta(1 + ybar[p,q],1
                                              + ybar[q,p]) - lbeta(1, 1)
      }
    }
  }

  return(sum(llik))
}
```

Checking that the full likelihood works effectively

```r
P_prior_probability = function(P,K,U_vec, sigma_squared){
  beta_params = beta_mean_var(U_vec,rep(sigma_squared,K) )
  alpha_k = beta_params$alpha
  beta_k = beta_params$beta

  p_P = matrix(0,K,K)
  for(p in 1:K){
    for(q in p:K){
      if(q-p != 0){
        p_P[p,q]<-  dbeta(P[p,q],alpha_k[q-p],beta_k[q-p],log = TRUE)
      }else if (p==q){
        p_P[p,p]<-  dbeta(P[p,q],1,1,log = TRUE)
      }
    }
  }

  return(sum(p_P))
}
```

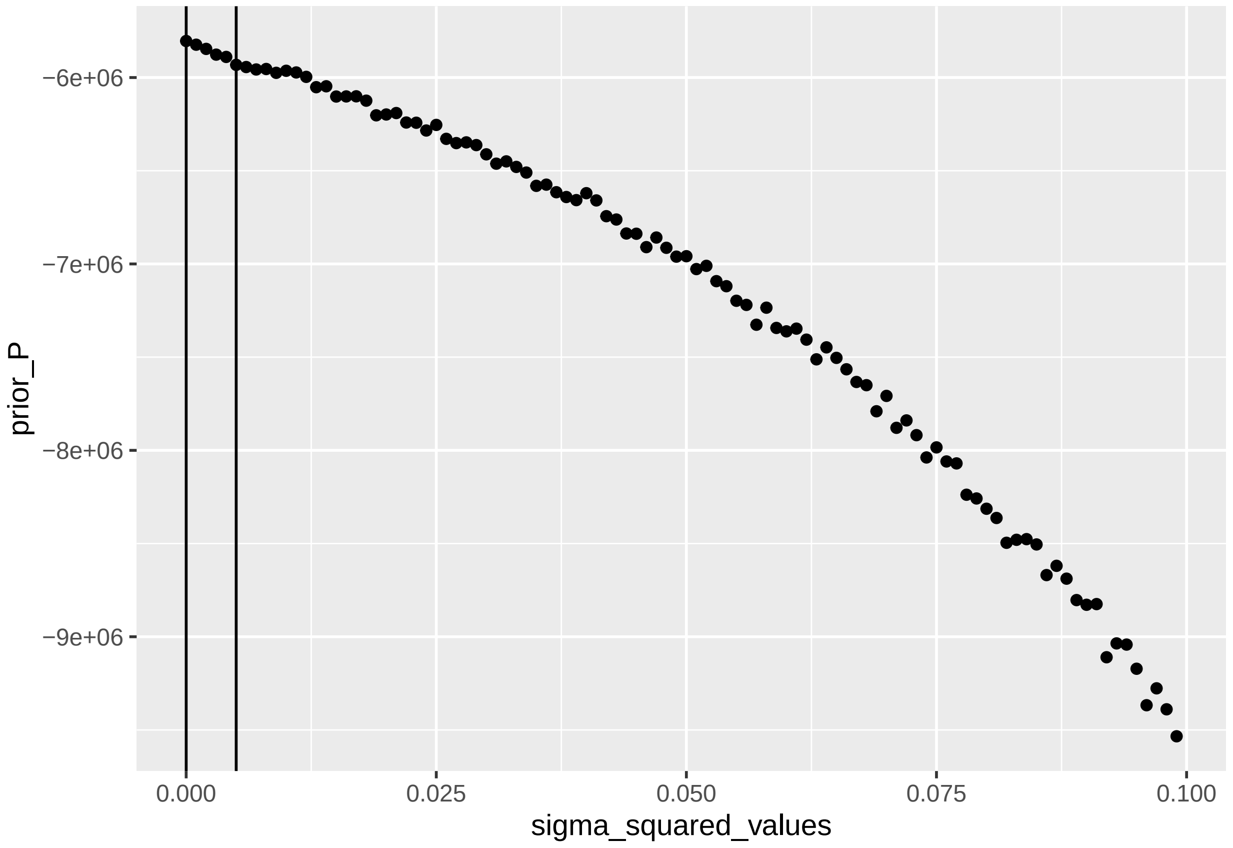
```
N_samples = 1000


set.seed(12)
n=100
N<- matrix(2,n,n)
diag(N)<-0
K= 5
a = .75
sigma_squared = 0.005
P_container = array(0, dim=c(K,K,N_samples))

Y_container= array(0, dim=c(n,n,N_samples))
z = matrix(0,n,1)
z<- sample(1:K, n,replace=T)
U = runif(K,0.5,a)
U_k = sort(U)


for(sample in 1:N_samples){

  beta_params = beta_mean_var(U_k,rep(sigma_squared,K) )

  a_k = beta_params$alpha
  b_k = beta_params$beta


  P = matrix(0,K,K)
  for(k in 0:(K-1)){
    for(i in 1:(K-1)){
      for(j in (i+1):K){
        if((j-i)==k){
          P[i,j]= rbeta(1,a_k[k+1],b_k[k+1])
        }
      }
    }
```

```r
  }

  P = P +  lower.tri(P)*(1-t(P))
  diag(P)= rbeta(K,1,1)
  P_container[,,sample]<-P

  z_P<- vec2mat_0_P(z,P)

  P_nbyn<- calculate_victory_probabilities(z_P,P)

  #simulating N
  Y_binom <- matrix(0, n,n)
  for(i in 1:n){
    for(j in 1:n){
      Y_binom[i,j]<- rbinom(1,N[i,j],P_nbyn[i,j])
    }
  }

  Y_binom[lower.tri(Y_binom)] = N[lower.tri(N)] - t(Y_binom)[lower.tri(Y_binom)]
  diag(Y_binom)<- 0

  Y_container[,,sample]<- Y_binom
}
```

```r
sigma_squared_values <-  seq(0.000001,.1,.001)

prior_P = matrix(0,length(sigma_squared_values),1)

for(try_out in 1:length(sigma_squared_values)){
  prior_container = matrix(0,N_samples,1)

  beta_params = beta_mean_var(U_k,rep(sigma_squared_values[try_out],K) )


  for(try_in in 1:N_samples){

    alpha_k = beta_params$alpha
    beta_k = beta_params$beta

    Y_binom <- Y_container[,,try_in]

    P = matrix(0,K,K)
    for(k in 0:(K-1)){
      for(i in 1:(K-1)){
        for(j in (i+1):K){
          if((j-i)==k){
            P[i,j]= rbeta(1,alpha_k[k+1],beta_k[k+1])
          }
        }
      }
    }

    P = P +  lower.tri(P)*(1-t(P))
    diag(P)= rbeta(K,1,1)
    # number of victories between block p and block q
    ybar = t(z_P)%*%(Y_binom*upper.tri(Y_binom))%*%z_P
    # number of missed victories between block p and block q
    n_minus_y1 <- (N-Y_binom)*upper.tri(N)
    # number of missed victories between block p and block q
    mbar<- t(z_P)%*%n_minus_y1%*%z_P

    coef1 = lchoose(N, Y_binom)*upper.tri(N)
    lambdabar <- t(z_P)%*%(coef1)%*%z_P

    prior_container[try_in]<- llik_over_blocks_f_binomial(bin_coef = lambdabar,ybar = ybar,mbar = mbar,
  }
  prior_P[try_out] <- sum(prior_container)
  print(sum(prior_container))
}
```

```
## [1] -5804046
## [1] -5823946
## [1] -5846449
## [1] -5877142
## [1] -5889469
## [1] -5932128
## [1] -5943770
## [1] -5956780
## [1] -5954202
## [1] -5974747
## [1] -5963774
## [1] -5972732
## [1] -5996358
## [1] -6051809
## [1] -6046934
## [1] -6101926
## [1] -6101344
## [1] -6100763
## [1] -6124089
## [1] -6202629
## [1] -6198254
## [1] -6190553
## [1] -6241281
## [1] -6242532
## [1] -6284157
## [1] -6254227
## [1] -6328997
## [1] -6351726
## [1] -6347806
## [1] -6362420
## [1] -6411991
## [1] -6462075
## [1] -6449705
## [1] -6479576
## [1] -6509708
## [1] -6581024
## [1] -6574622
## [1] -6615311
## [1] -6640985
## [1] -6657567
## [1] -6620420
## [1] -6659178
## [1] -6743814
## [1] -6761710
## [1] -6836768
## [1] -6838022
## [1] -6909798
## [1] -6858362
## [1] -6913376
## [1] -6960851
## [1] -6958478
## [1] -7027588
## [1] -7010166
## [1] -7092384
## [1] -7119405
## [1] -7197657
## [1] -7219736
## [1] -7326153
## [1] -7234699
## [1] -7343362
## [1] -7361607
## [1] -7347105
## [1] -7406224
## [1] -7511841
## [1] -7447503
## [1] -7503907
## [1] -7565174
## [1] -7633152
## [1] -7650451
## [1] -7790448
## [1] -7707885
## [1] -7879215
## [1] -7839416
## [1] -7918564
## [1] -8038230
## [1] -7984170
## [1] -8059612
## [1] -8070086
## [1] -8238118
## [1] -8258235
## [1] -8313384
## [1] -8362837
## [1] -8495958
## [1] -8479892
## [1] -8476397
## [1] -8504826
## [1] -8669264
## [1] -8619525
## [1] -8688423
## [1] -8803324
## [1] -8828516
## [1] -8824770
## [1] -9109903
## [1] -9035709
## [1] -9042073
## [1] -9171726
## [1] -9367294
## [1] -9276738
## [1] -9389203
## [1] -9533927
```

```
a_df = data.frame(sigma_squared_values = sigma_squared_values,prior_probabilities = prior_P)

ggplot(a_df, aes(x = sigma_squared_values, y = prior_P))+
  geom_point()+
  geom_vline(xintercept = sigma_squared)+
  geom_vline(xintercept = sigma_squared_values[which.max(prior_P)])
```


-6e+06
-7e+06
-8e+06
-9e+06
prior_P
sigma_squared_values