R Notebook

Projet Unsupervised Learning

Let's Simulate a directed graph with K = 8 nodes using Erdos-Renyi's model with p = 0.4. We will do so using the package igraph: The model can be written probabilisticaly as follows:

$$X_{ij}|\{Z_i = k, Z_j = l\} \sim Bern(0.4)$$

where a 1 in the matrix corresponds to website i pointing toward website j. Note that in this setting we will assume that a website cannot point to itself(i.e the diagonal of the matrix X is 0).

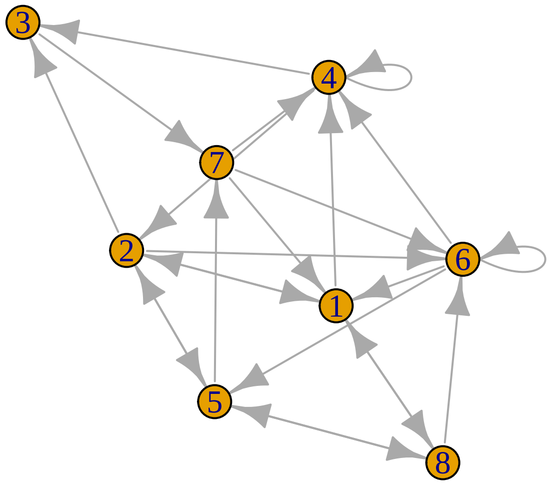
```r
set.seed(20222023)
library(igraph)
```

```
##
## Attachement du package : 'igraph'

## Les objets suivants sont masqués depuis 'package:stats':
##
##     decompose, spectrum

## L'objet suivant est masqué depuis 'package:base':
##
##     union
```

```r
library(ggraph)
```

```
## Le chargement a nécessité le package : ggplot2
```

```r
library(ggplot2)
G1 = sample_gnp(n = 8, p = .4, directed = T, loops = TRUE)
X = as.matrix(as_adjacency_matrix(G1))
#pdf('mtg.pdf')
p1=plot(G1)
```



```
#dev.off()
X
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
## [1,]    0    1    0    1    0    0    0    1
## [2,]    1    0    1    0    1    1    0    0
## [3,]    0    0    0    0    0    0    1    0
## [4,]    0    1    1    1    0    0    0    0
## [5,]    0    1    0    0    0    0    1    1
## [6,]    1    0    0    1    1    1    0    0
## [7,]    1    0    0    1    0    1    0    0
## [8,]    1    0    0    0    1    1    0    0
```

Let's write the transition matrix using the random surfer's model with probability $p = 1 - \epsilon$ of chosing the available hyperlinks and probability ϵ of choosing a website randomly out of the 8 available websites:

Using the above hypothesis the transition matrix can be written as

$$\mathbf{A} = (1-\epsilon)\mathbf{M} + \frac{\epsilon}{K}\mathbf{I_8}.$$

Where

$$M_{ij} = \frac{X_{ij}}{\sum_j X_{ij}}, \forall\, 1 \leq i, j \leq 8;$$

```
TransMat = function(adjMat, epsilon){
  K = ncol(adjMat)
  A = (1- epsilon)* (adjMat / rowSums(adjMat)) + (epsilon/K) * diag(K)
  return(A)
}
```

Let's fix $\epsilon = 0.05$, and assuming that the Markov Chain is aperiodic and irreducible let's compute the stationary distribution:

Recall for an aperiodic and irreducible $\mathcal{MC}$ finding the stationary distribution amounts to solving the equation $\pi\mathbf{A} = \pi$. Furthermore, it is also equivalent to finding the eigenvector of $\mathbf{A}^\top$ associated to the eigenvalue $\lambda = 1$ and renormalizing it (i. e divide each element of the vector by the sum of all of its elements)

```
A = TransMat(adjMat = X, epsilon = 0.05)
v = eigen(t(A))$vectors[, 1]
Pi = Re(v)/sum(Re(v))
round(Pi, 3)
```

```
## [1] 0.138 0.140 0.097 0.185 0.095 0.138 0.128 0.078
```

Let's simulate a sequence of 1000 clicks of the random web surfer (we will start from webpage 2).

```
set.seed(20222023)
n = 1000
Sequance = rep(0, n)
Sequance[1] = 2
for (i in 2:n){
  Sequance[i] = sample(1:8, size = 1, prob = A[Sequance[i-1],])
}
```

- Let's estimate the transition matrix $\hat{\mathbf{A}}$.

```
temp = table(Sequance[1:n-1],Sequance[2:n])
A_hat = temp/rowSums(temp)
A_hat
```

```
##
##               1           2           3           4           5           6
##   1 0.007751938 0.418604651 0.000000000 0.279069767 0.000000000 0.000000000
##   2 0.206666667 0.026666667 0.280000000 0.000000000 0.253333333 0.233333333
##   3 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000
##   4 0.000000000 0.327956989 0.349462366 0.322580645 0.000000000 0.000000000
##   5 0.000000000 0.344827586 0.000000000 0.000000000 0.011494253 0.000000000
##   6 0.208633094 0.000000000 0.000000000 0.287769784 0.194244604 0.309352518
##   7 0.318840580 0.000000000 0.000000000 0.369565217 0.000000000 0.311594203
##   8 0.380952381 0.000000000 0.000000000 0.000000000 0.333333333 0.285714286
##
##               7           8
##   1 0.000000000 0.294573643
##   2 0.000000000 0.000000000
##   3 1.000000000 0.000000000
##   4 0.000000000 0.000000000
##   5 0.356321839 0.287356322
##   6 0.000000000 0.000000000
##   7 0.000000000 0.000000000
##   8 0.000000000 0.000000000
```

- Let's estimate the stationary distribution and compare it to the one computed in the previous question. For comparison we will compute the $\updownarrow_1$ norm of their difference.

```
v_hat = eigen(t(A_hat))$vectors[, 1]
Pi_hat = Re(v_hat)/sum(Re(v_hat))
round(Pi_hat, 3)
```

```
## [1] 0.129 0.149 0.107 0.188 0.087 0.139 0.138 0.063
```

```
print("The l1 norm of their difference is:")
```

```
## [1] "The l1 norm of their difference is:"
```

```
sum(abs(Pi - Pi_hat))
```

```
## [1] 0.06615326
```

```
print("The forward KL divergence is:")
```

```
## [1] "The forward KL divergence is:"
```

```
sum(Pi*log(Pi/Pi_hat))
```

```
## [1] 0.003661656
```

Web Communities

```
A_mat = matrix(c(0.7, 0.2, 0.1, 0.25, 0.7, 0.05, 0.1, 0.1, 0.8), byrow = T, nrow = 3)

B = matrix(c(0.2, 0, 0.1, 0, 0.1, 0.3, 0.1, 0.3, 0, 0.1, 0.2, 0.2, 0.2, 0, 0.1, 0.3, 0, 0, 0.1, 0.1, 0.

hidden_states = c('Sport', 'Culture', 'Beauty')

emissions = c('Abdominaux', 'Cosmetiques', 'Livres', 'Age', 'Force', 'Endurance', 'Resilience', 'Creme'
```

```r
new_B=matrix(nrow=55, ncol = 3)
B_couples=matrix(nrow=55, ncol = 3)

k=1
#all couples
for (i in 1:10){
    for (j in 1:10){
        if (i<=j){
            new_B[k, ]=B[i, ]*B[j, ]

        k=k+1


        }


    }
}
#Taking care of repeated couple (by brute force)
mask=c(1, 11, 20, 28, 35, 41, 46, 50, 53, 55)
B_couples[mask,]=new_B[mask,]
B_couples[2:10, ]=2*new_B[2:10, ]
B_couples[12:19, ]=2*new_B[12:19, ]
B_couples[21:27, ]=2*new_B[21:27, ]
B_couples[29:34, ]=2*new_B[29:34, ]
B_couples[36:40, ]=2*new_B[36:40, ]
B_couples[42:45, ]=2*new_B[42:45, ]
B_couples[47:49, ]=2*new_B[47:49, ]
B_couples[51:52, ]=2*new_B[51:52, ]
B_couples[54, ]=2*new_B[54, ]
colSums(B_couples)#it sums to 1
```

From the emission matrix of key words to that of couples

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

Sampling from the model(HMM)

```r
set.seed(20222023)
n_couples = 55
n_states = 3
Z = matrix(rep(0, 30*100), nrow = 30)
XX = matrix(rep(0, 30*100), nrow = 30)
pi_start = c(0.1, 0.3, 0.6)
for (i in 1:30){
  Z[i, 1] = sample(1:n_states, size = 1, prob = pi_start)
  XX[i, 1] = sample(1:n_couples, size = 1, prob = B_couples[, Z[i, 1]])
}

for (i in 1:30){
  for (j in 2:100){
    Z[i, j] = sample(1:n_states, size = 1, prob = A_mat[Z[i, j-1], ])
    XX[i, j] = sample(1:n_couples, size = 1, prob = B_couples[, Z[i, j]])
  }
}
```

Let's implement the Baum-Welch algorithm and use it to estimate the parameters of the model using the data simulated from the previous question.

```
#implementation of the logsumexp trick
logsumexp <- function(logx) {
# compute \log(\sum exp(logx)) by rescaling it by m = \max(logx)
# indeed : \log(\sum exp(logx)) = m + \log(\sum exp(logx - m))
# This ensures an exp(0) somewhere in the sum
m = max(logx)
return(m + log(sum(exp(logx - m))))
}


#The forward algorithm as explained in the appendix
forward = function(emis, trans, initial, data){
    #A function that implements the forward algorithm
    #initialization
    K = ncol(trans)
    T = ncol(data)
    n = nrow(data)
    epsilon = 1e-300#to avoid taking log of 0 if it happens
    logalph = array(rep(0, K*T*n), dim = c(n, K, T))

    for (i in 1:n){
    #fill in the first element in logspace
    logalph[i, , 1] = log(initial) + log(emis[data[i, 1], ] + epsilon)
    }

    for (t in 2:T){
        for (k in 1:K){
            for (i in 1:n){
                #the argument for the log sum exp function
                arg =  log(trans[, k]+epsilon) + logalph[i, , t-1]
                logalph[i, k, t] = log(emis[data[i, t], k]+epsilon) + logsumexp(arg)
    }
    }
    }

    return(logalph)
}

#The backward algorithm as explained in the appendix
backward = function(emis, trans, initial, data){
    #A function that implements the backward algorithm
    #initialization
    K = ncol(trans)
    T = ncol(data)
```

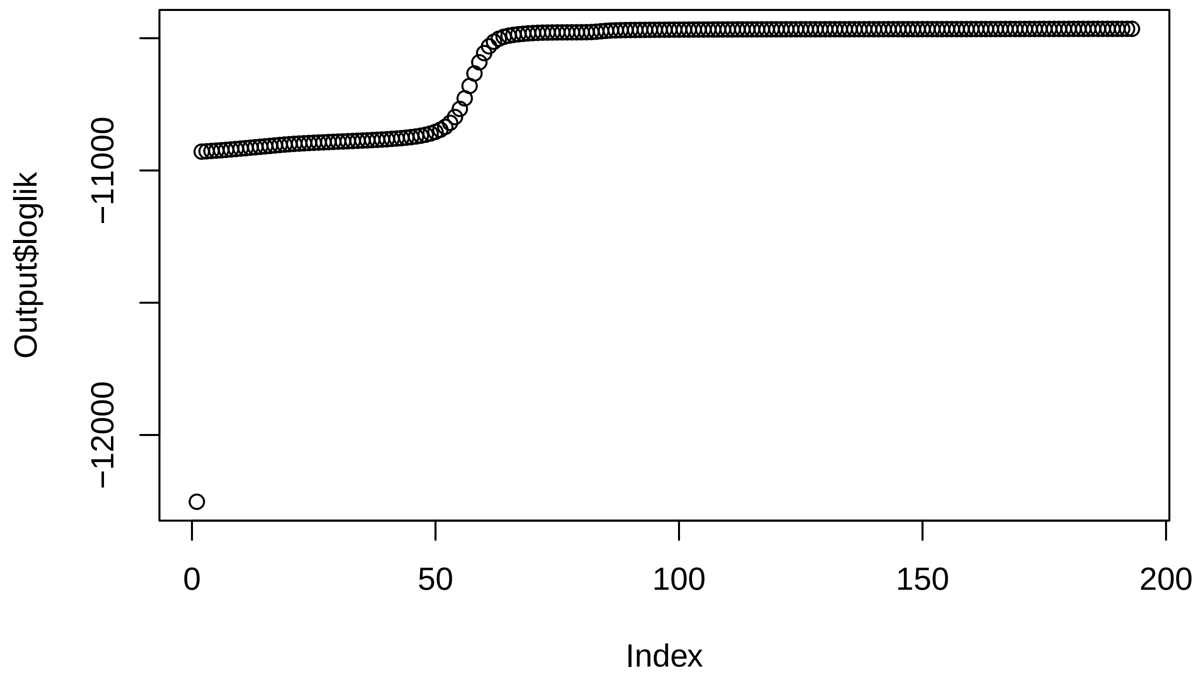
```r
    n = nrow(data)
    epsilon = 1e-300#to avoid taking log of 0 if it happens
    logbeta = array(rep(0, K*T*n), dim = c(n, K, T))

    for (i in 1:n){
        #the last element since beta_T=1 its log is 0
        logbeta[i, , T] = 0 + epsilon
    }

    for (t in (T-1):1){
        for (k in 1:K){
            for (i in 1:n){
                #the argument for the log sum exp function
                arg = log(trans[k, ]+epsilon) + log(emis[data[i, t+1], ]+epsilon) + logbeta[i, ,t+1]
                #fill in logbeta
                logbeta[i, k, t] = logsumexp(arg)
            }
        }
    }
    return(logbeta)

}


#Computing the \log\xi matrix as   explained in the appendix
logKSI = function(emis, trans, initial, data, fr, br){
    #this function computes the log ksis
    #initialization
    K = ncol(trans)
    T = ncol(data)
    n = nrow(data)
    epsilon = 1e-300
    KSI_mat = array(rep(0, n*K*K*(T-1)), dim = c(n, (T-1), K, K))
    #compute the loglikelihood of each sequance
    logP = apply(fr[, , T], 1, logsumexp)
    for (l in 1:K){
        for (k in 1:K){
            for (t in 1:(T-1)){
                for (i in 1:n){
                #fill in the ksi matrix
                KSI_mat[i, t, l, k] = (fr[i, l, t] + br[i, k, t+1] + log(trans[l, k]+epsilon) +
                    log(emis[data[i, t+1], k]+epsilon) - logP[i])
                }
            }
        }
    }
    return(KSI_mat)
}


#all togther we get the Baum-Welch algorithm
Baum_Welch = function(emis, trans, initial, data, n_it, epsilon=1e-6){
```

```r
K = ncol(trans)
T = ncol(data)
n = nrow(data)
brow = nrow(emis)

# Pi_new = rep(0, K)
A_new = matrix(rep(0, K*K), nrow = K)
B_new = matrix(rep(0, brow*K), nrow = brow)
fr = forward(emis, trans, initial, data)
br = backward(emis, trans, initial, data)
KSI_mat = logKSI(emis, trans, initial, data, fr, br)

A_old = trans
B_old = emis
Pi_old = initial
loglik = c(sum(apply(fr[, , T], 1, logsumexp)))

comp = array(rep(0, n*K*T), dim = c(n, K, T))
comp[, , 1:(T-1)] = aperm(apply(KSI_mat, c(1, 2, 3), logsumexp), c(1, 3, 2))
comp[, , T] = apply(KSI_mat[,(T-1), , ], c(1, 3), logsumexp)
comp = exp(comp)
for (iter in 1:n_it){
    #a vectorized formula for pi
    Pi_new = exp(apply(fr[, , 1] + br[, , 1] - apply(fr[, , T], 1, logsumexp), 2, logsumexp)-log(n))
    denom = apply(exp(KSI_mat), c(1, 2, 3), sum)
    nume = exp(KSI_mat)

    # comp = exp(LogTau(B_old , A_old, Pi_old , data, fr, br))#for emission matrix

    for (l in 1:K){
        for(k in 1:K){
            A_new[l, k] = sum(nume[, , l, k])/sum(denom[ , , l])
        }
    }
    for (j in 1:brow){
        for (k in 1:K){
            B_new[j, k] = sum(comp[, k, ]*(XX==j))/sum(comp[, k, ])

        }

    }
    B_old = B_new
    A_old = A_new
    Pi_old = Pi_new
    fr = forward(B_old, A_old, Pi_old, data)
    br = backward(B_old, A_old, Pi_old, data)
    KSI_mat = logKSI(B_old, A_old, Pi_old, data, fr, br)
    loglik = c(loglik, sum(apply(fr[, , T], 1, logsumexp)))
    comp[, , 1:(T-1)] = aperm(apply(KSI_mat, c(1, 2, 3), logsumexp), c(1, 3, 2))
    comp[, , T] = apply(KSI_mat[,(T-1), , ], c(1, 3), logsumexp)
    comp = exp(comp)
    if (abs((loglik[iter+1] - loglik[iter])/loglik[iter])<= epsilon){
    break
```

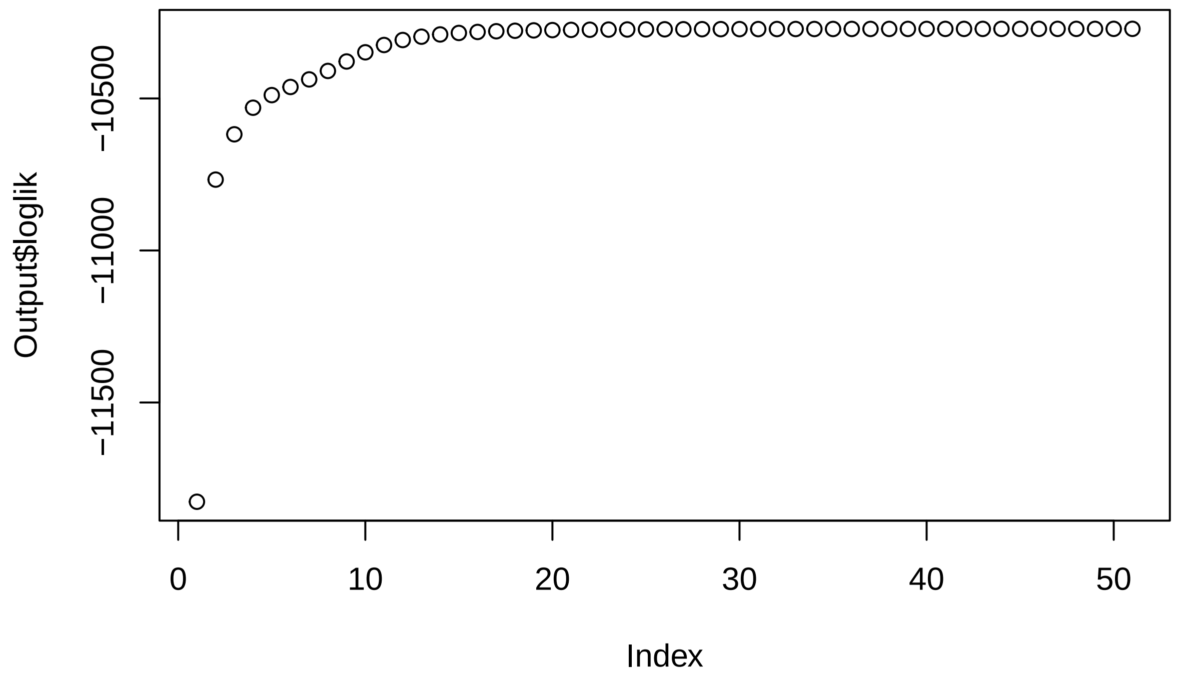
```
        }
    }

    param = list()
    param$Pi = round(Pi_old, 2)
    param$A = round(A_old, 2)
    param$B = round(B_old, 2)
    param$loglik = loglik
    return(param)
}
```

Estimating the parameters using the Baum__Welch

```
#A naive Initialization
K = n_states
J = n_couples
init_pi = rep(1/K, K)
init_A = abs(matrix(rnorm(n=K^2, sd=10), K, K))
init_A = init_A/rowSums(init_A)
init_B = abs(matrix(rnorm(n=K*J, sd=10), K, J))
init_B = init_B/rowSums(init_B)
init_B = t(init_B)

#estimation

Output = Baum_Welch(emis = init_B, trans=init_A, initial = init_pi, data = XX, n_it=500)
plot(Output$loglik)
```



```
#An initialization by perturbing the initial parmaters a little bit

#coded in class

perturbate_matrix = function(X, noise=7e-2) {
# function to add small noise to the true parameter A & B
X_pert = X +
matrix(rnorm(prod(dim(X)), 0, sd=noise),
nrow(X),
ncol(X))
X_pert = abs(X_pert) # enforce positive values !
X_pert = X_pert / rowSums(X_pert)
return(X_pert)
}

init_pi = (pi_start + abs(rnorm(n = 3, sd = 7e-2)))/sum((pi_start + abs(rnorm(n = 3, sd = 7e-2))))
init_A = perturbate_matrix(X = A_mat)
init_B = t(perturbate_matrix(X=t(B_couples)))
Output = Baum_Welch(emis = init_B, trans=init_A, initial = init_pi, data = XX, n_it=500)

plot(Output$loglik)
```



```
#we see that in this case it is better
```

Let's use Viterbi's algorithm to estimate the successive hidden states of the first row in the simulated data

```
#The Viterbi algorithm algorithm
param_test = list(A = A_mat, B = t(B_couples), pi = pi_start)
Viterbi<-function(x, param){
epsilon<-1e-6
K<-nrow(param$A)
n<-length(x)
S<-matrix(0,K,n)
logV<-matrix(-Inf,K,n)
Zest<-rep(0,n)
for (k in 1:K){
logV[k,1]<-log(param$B[k, x[1]]+epsilon)+log(param$pi[k])
}
# Forward
for (t in (2:n))
for (k in (1:K)){
logV[k,t]=max(logV[,t-1]+log(param$A[,k])+log(param$B[k, x[t]]))
S[k,t-1]=which.max(logV[,t-1]+log(param$A[,k])+log(param$B[k, x[t]]))
}
# Back-tracking
Zest[n]<-which.max(logV[,n])

for (t in (n-1):1)
Zest[t]<-S[Zest[t+1],t]
return(Zest)
}

Z1_hat = Viterbi(x=XX[1, ], param=param_test)
```

The percentage of errors made in the estimation of Z1 is

```
100*(sum(Z1_hat != Z[1, ]))/length(Z1_hat)
```

```
## [1] 6
```

We observe above that we only made 6 errors, out the 100 estimated values. Which is not bad at all (using the optimal values).

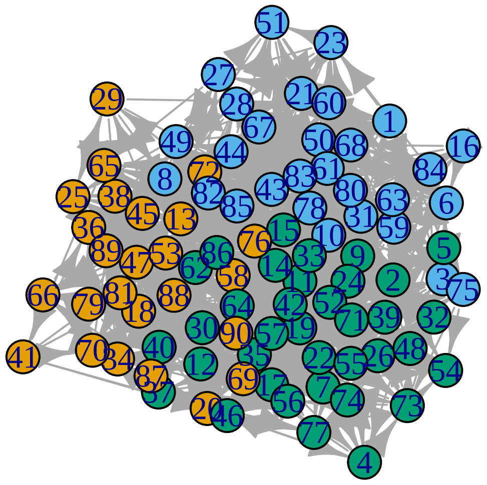
Community Detection

Let there be 90 pages clustered in three groups $\{S, C, B\}$ we will note the $\{1, 2, 3\}$ Let's simulate an adjacency matrix of the directed graph using the stochastic block model described in the report

```
set.seed(20222023)
SBM_sample = function (n=90, pi_start = c(1/3, 1/3, 1/3), alpha=0.15, beta=0.05, loop=FALSE, directed=T

    K = length(pi_start)

    Z = sample(1:K, replace = T, size = n,  prob = pi_start)
    X = matrix(rep(0, n*n), nrow = n)
    Gamma = matrix(rep(alpha, K*K), nrow=K)
    Gamma[upper.tri(Gamma)|lower.tri(Gamma)] = beta

    #if the graph is directed
    if (directed) {

        #if loops are allowed (nodes can point to themselves)
        if (loop) {
            for (i in 1:n){
                for (j in 1:n){
                    X[i, j] = rbinom(1, 1, prob = Gamma[Z[i], Z[j]])
                }
            }
            }

        #if loops are not allowed (nodes cannot point to themselves)
        else{
            for (i in 1:n){
                for (j in 1:n){
                    if (i!=j) { X[i, j] = rbinom(1, 1, prob = Gamma[Z[i], Z[j]]) }
            }
            }
        }
    }

    #if the graph is not directed(thus in this case X is symetric, and loop are not allowed)
    else {
        for (i in 1:n){
            for (j in 1:n){
                if (i<j){
                    X[i, j] = rbinom(1, 1, prob = Gamma[Z[i], Z[j]])
                    X[j, i] = X[i, j]
                }
            }
        }
    }

    simul = list()
    simul$Adjacency = X
    simul$cluster = Z
    return (simul)


}

gra = SBM_sample()
```

Plot the graph

```
#pdf('sbm_comm.pdf')
plot(graph_from_adjacency_matrix(gra$Adjacency,mode="directed"),vertex.color=gra$cluster)
```



```
#dev.off()
```

Let's compute the A^1, and A^2

```
eps = 1/1000
A_one = gra$Adjacency + eps
A_one = A_one / rowSums(A_one)

A_two = (1/90)*matrix(rep(1, 90*90), nrow=90)
rowSums(A_one)
```

```
##  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [39] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [77] 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

```
rowSums(A_two)
```

```
##  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [39] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [77] 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

###Let's sample a sequence of 500 sequence of couples from the model with A_1

```r
set.seed(20222023)
# websites are labeled from 1 to 90
#couples are labeled from 1 to 55 this is to facilitate the computations as explained in the report
group = gra$cluster
Web_1 = rep(0, 500)#webs using A1
cpls_1 = rep(0, 500)#couple using A1
Web_1[1]=7#we begin at website 7
cpls_1[1]=sample(1:n_couples, size=1, prob = B_couples[, group[Web_1[1]]])
for (i in 2:500){
  Web_1[i]=sample(1:90, size=1, prob = A_one[Web_1[i-1], ])
  cpls_1[i]=sample(1:55, size=1, prob = B_couples[, group[Web_1[i]]])
}
```

###Let's sample a sequence of 500 sequence of couples from the model with A_2

```r
set.seed(20222023)
# websites are labeled from 1 to 90
#couples are labeled from 1 to 55 this is to facilitate the computations as explained in the report
group = gra$cluster
Web_2 = rep(0, 500)#webs using A2
cpls_2 = rep(0, 500)#couple using A2
Web_2[1]=7#we begin at website 7
cpls_2[1]=sample(1:n_couples, size=1, prob = B_couples[, group[Web_2[1]]])
for (i in 2:500){
  Web_2[i]=sample(1:90, size=1, prob = A_two[Web_2[i-1], ])
  cpls_2[i]=sample(1:55, size=1, prob = B_couples[, group[Web_2[i]]])
}
```