Getting started

Installation of the main packages

In [234…
```
# you have to first use install.packages('igraph') and install.packages('ggp
library('igraph')
library('ggplot2')
library('igraphdata')
```

Sampling strategies

Average degree estimation

1/ Create a function *Random_sampling(g,nb.sample)* which samples uniformly *nb.sample* nodes in a graph defined by the adjacency matrix g. When a node i is sampled, it will also provide the information of its degree. You can find a code to complete below.

In [235…
```
Random_sampling<-function(g,nb.sample){
    I<-dim(g)[1]
    nodes <- sample(1:I, nb.sample, replace = TRUE)
    degree.vec<-c()
    for(i in nodes){
        degree.vec<-c(degree.vec,sum(g[i,]))
    }
    return(list(nodes,degree.vec))
}
```

2/ Create a function *Degree_Random_sampling(g,nb.sample)* which samples *nb.sample nodes* and where the probability of sampling a node i is given by $\frac{d_i}{\sum_{j=1}^{I} d_j}$, where d_i is the degree of node i. When a node i is sampled, it will also provide the information of its degree.

In [236…
```
Degree_Random_sampling<-function(g,nb.sample){
    I<-dim(g)[1]
    deg.vector<-colSums(g)
    probabilities <- deg.vector / sum(deg.vector)
    nodes <- sample(1:I, nb.sample, replace = TRUE, prob = probabilities)
    degree.vec<-c()
    for(i in nodes){
        degree.vec<-c(degree.vec,sum(g[i,]))
    }
    return(list(nodes,degree.vec))
}
```

3/ Create a function *Random_walk_Random_sampling(g,nb.sample)* which samples *nb.sample nodes* according to a simple random walk.

Reminder: The random walk starts to a node sampled uniformly at random. More precisely if g is the adjacency matrix of the graph with I nodes, the simple random on the graph defined by g is the Markov chain with:

- the state space equal to $\{1, \dots, I\}$;
- the transition matrix P given by:

$$P(i,j) = \begin{cases} \frac{1}{\mathrm{degree}(i)} & \text{if } g(i,j) = 1, \\ 0 & \text{otherwise,} \end{cases} \quad \forall i, j \in \{1, \dots, I\}.$$

In [237…

```
Random_walk_Random_sampling<-function(g,nb.sample){
    I<-dim(g)[1]
    # Initialization
    nodes<-c()
    i0<- sample(1:I, 1)# initialization of the random walk
    nodes<-c(nodes,i0)
    degree.vec<-c()
    degree.vec<-c(degree.vec,sum(g[i0,]))
    for(n in 1:nb.sample){
        neighbors <- which(g[i0, ] == 1)
        if (length(neighbors) > 0) {
            i1 <- sample(neighbors, 1)
        } else {
            i1 <- i0  # Stay in the same node if no neighbors
        }

        if(sum(nodes==i1)>0){
            nodes <- nodes
            degree.vec <- degree.vec
        }
        else{
            nodes <- c(nodes,i1)
            degree.vec <- c(degree.vec,sum(g[i1,]))
        }
        i0 <- i1
    }
    return(list(nodes,degree.vec))
}
```

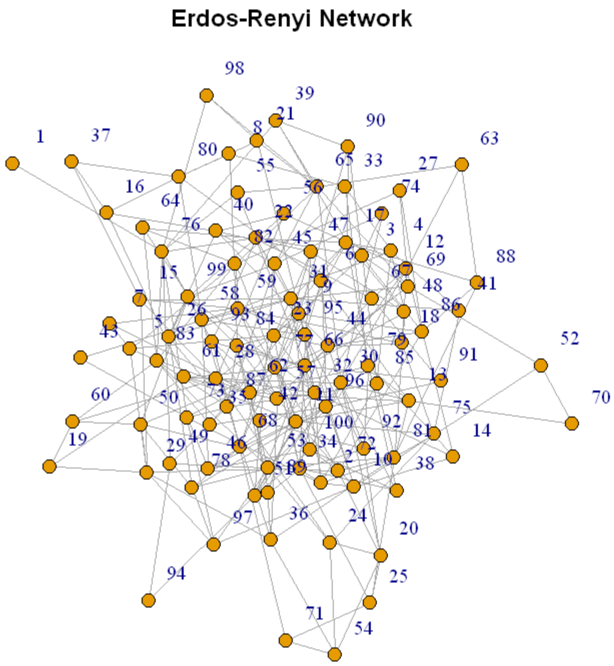
3/ Simulate one Erdos-Renyi network with n=100 and p=0.06 using the function *erdos.renyi.game* . Plot the graph. What is the average degree of the graph?

In [238…

```
# Simulate Erdos-Renyi network
n <- 100
p <- 0.06
er_graph <- erdos.renyi.game(n, p)

# Plot the graph
```

```
plot(er_graph, main = "Erdos-Renyi Network", vertex.label.dist = 2, vertex.s

# Calculate the average degree
average_degree <- mean(degree(er_graph))
cat("Average Degree:", average_degree, "\n")
```

```
Average Degree: 5.86
```



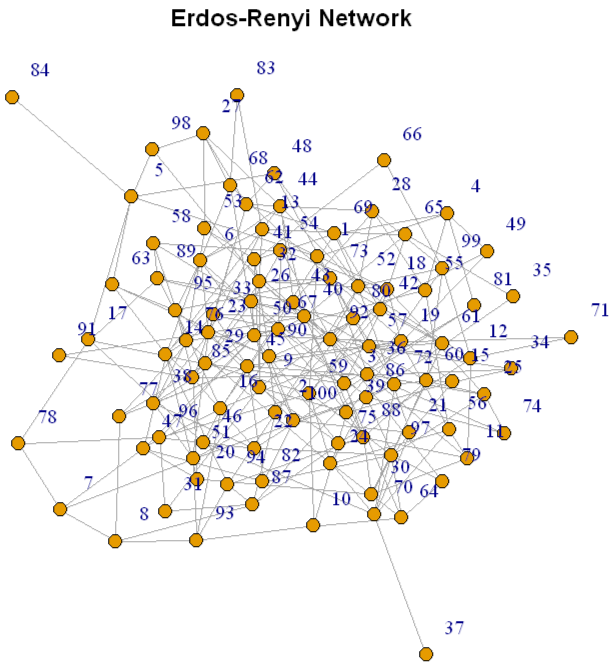
4/ Use the *Random_sampling()*, *Degree_Random_sampling()* and *Random_walk_Random_sampling()* function to extract 10 nodes with their respective degree on the same Erdos-Renyi network. What is the average degree? Repeat 10 times the same experiment and compute the mean and the variance of the average degree. What do you notice? Are your observations consistent with the theory seen during the class?

In [239…

```
# Function for Erdos-Renyi network creation
generate_er_network <- function(n, p) {
    #set.seed(42)
    er_graph <- erdos.renyi.game(n, p)
    return(er_graph)
}

# Function to repeat the experiment and compute mean and variance
repeat_experiment <- function(graph, sampling_function, nb_repeats) {
    avg_degrees <- numeric(nb_repeats)

    for (i in 1:nb_repeats) {
        result <- sampling_function(graph, 100)
        avg_degrees[i] <- mean(result[[2]])
    }
```

```
    mean_degree <- mean(avg_degrees)
    variance_degree <- var(avg_degrees)

    return(c(mean_degree, variance_degree))
}

# Simulate Erdos-Renyi network
n <- 100
p <- 0.06
er_graph <- generate_er_network(n, p)
# Plot the graph
plot(er_graph, main = "Erdos-Renyi Network", vertex.label.dist = 2, vertex.s

# Calculate the average degree
average_degree <- mean(degree(er_graph))
cat("Average Degree:", average_degree, "\n")

# Obtain adjacency matrix
er_g = as_adjacency_matrix(er_graph)
er_g <- as.matrix(er_g)

# Extract 10 nodes with their respective degrees using each sampling method
result_random <- repeat_experiment(er_g, Random_sampling, 10)
result_degree_random <- repeat_experiment(er_g, Degree_Random_sampling, 10)
result_random_walk <- repeat_experiment(er_g, Random_walk_Random_sampling, 1

cat("Random Sampling - Mean:", result_random[1], "Variance:", result_random[
cat("Degree Random Sampling - Mean:", result_degree_random[1], "Variance:",
cat("Random Walk Random Sampling - Mean:", result_random_walk[1], "Variance:
```

```
Average Degree: 5.64
Random Sampling - Mean: 5.539 Variance: 0.04107667
Degree Random Sampling - Mean: 6.647 Variance: 0.04935667
Random Walk Random Sampling - Mean: 6.191555 Variance: 0.02396268
```



It is observed that the random sampling is unbiased, as it is expected since nodes are chosen with equal probability. However, this is not the case in the Degree random sample method or the random walk, because these methods consist of a respondent-based sampling. There is then a bias and a different estimator should be used for the average degree.

It can also be verified that if the number of extracted nodes grows to infinity all variances tend to 0, but only the random sampling is unbiased.

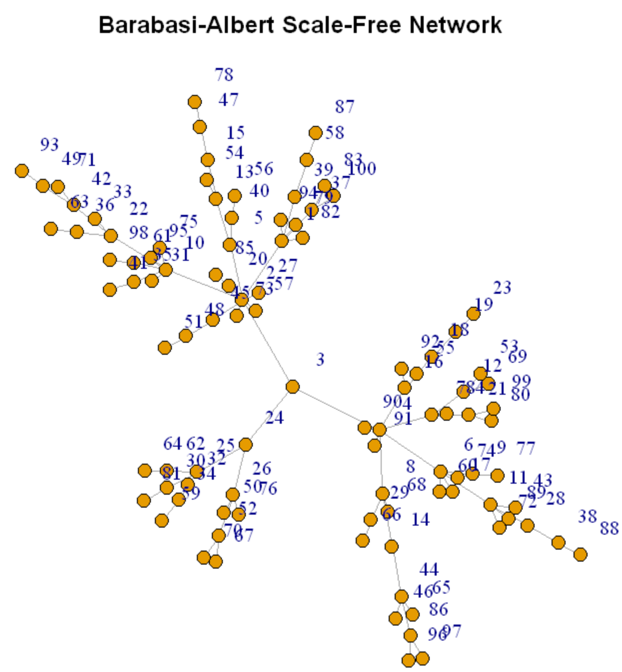
5/ Generate a scale-free graphs according to the Barabasi-Albert model (using the function *sample_pa*). Plot the graph. Repeat the simulations proposed in question 4/ for this new graph. What do you observe?

In [240…

```
# Function for Barabasi-Albert graph creation
generate_scale_free_graph <- function(n, m) {
    set.seed(42)
    sf_graph <- sample_pa(n, m, directed = FALSE)
    return(sf_graph)
}

# Generate Barabasi-Albert scale-free graph
n_sf <- 100
m_sf <- 1
sf_graph <- generate_scale_free_graph(n_sf, m_sf)

# Calculate the average degree
average_degree <- mean(degree(sf_graph))
```

```
cat("Average Degree:", average_degree, "\n")

# Obtain adjacency matrix
sf_g = as_adjacency_matrix(sf_graph)
sf_g <- as.matrix(sf_g)

# Plot the scale-free graph
plot(sf_graph, main = "Barabasi-Albert Scale-Free Network", vertex.label.dis

# Extract 10 nodes with their respective degrees using each sampling method
result_sf_random <- repeat_experiment(sf_g, Random_sampling, 10)
result_sf_degree_random <- repeat_experiment(sf_g, Degree_Random_sampling, 1
result_sf_random_walk <- repeat_experiment(sf_g, Random_walk_Random_samplinc

cat("Random Sampling - Mean:", result_sf_random[1], "Variance:", result_sf_r
cat("Degree Random Sampling - Mean:", result_sf_degree_random[1], "Variance:
cat("Random Walk Random Sampling - Mean:", result_sf_random_walk[1], "Variar
```

```
Average Degree: 1.98
Random Sampling - Mean: 2.049 Variance: 0.01436556
Degree Random Sampling - Mean: 3.222 Variance: 0.06481778
Random Walk Random Sampling - Mean: 2.55876 Variance: 0.002291535
```

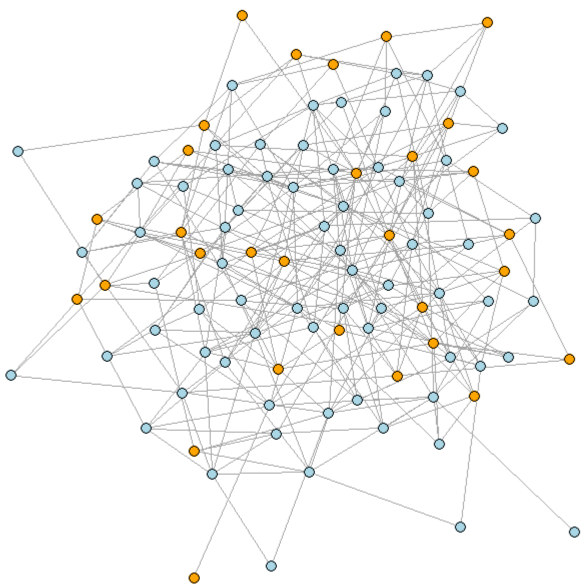


Here, regarding bias, the same results previously found remain true. However, a smaller variance is found, which is reasonable taking into account that the variance of the degree of each node is lower (i.e. all nodes have a more similar degree than in the previous graph).

Estimation in Hidden Populations

The graph generated below contains nodes with two states (1 and 0). Only 30 nodes are in state 1. We want to implement the estimator seen during the class that will help us estimate the proportion of individual in state 1.

In [241…
```
graph <- erdos.renyi.game(100, 0.07, type = "gnp")
sub.pop <- sample(1:100)[1:30]
g <-  as.matrix(as_adjacency_matrix(graph))
V(graph)$color<-"lightblue"
V(graph)$color[sub.pop]<-"orange"
plot(graph, vertex.label= NA, edge.arrow.size=0.02,vertex.size = 4, xlab = "
```



Random Network: G(N,p) model

6/ Transform the *Random_walk_Random_sampling(g,nb.sample)* function to *Random_walk_Random_sampling_modified(g,sub.pop,nb.sample)* where when a node i is sampled, it will provide the information of its degree and its state.

In [242…
```
Random_walk_Random_sampling_modified<-function(g, sub.pop, nb.sample){
    I<-dim(g)[1]
    # Initialization
    nodes<-c()
    i0<- sample(1:I, 1) # initialization of the random walk
    nodes<-c(nodes,i0)
    degree.vec<-c()
    state.vec<-c()
    degree.vec<-c(degree.vec,sum(g[i0,]))
    state.vec<-c(state.vec, as.integer(as.logical(i0 %in% sub.pop)))

    for(n in 1:nb.sample){
```

```
        neighbors <- which(g[i0, ] == 1)
        if (length(neighbors) > 0) {
            i1 <- sample(neighbors, 1)
        } else {
            i1 <- i0  # Stay in the same node if no neighbors
        }

        if(sum(nodes==i1)>0){
            nodes <- nodes
            degree.vec <- degree.vec
            state.vec <- state.vec
        }
        else{
            nodes <- c(nodes,i1)
            degree.vec <- c(degree.vec, sum(g[i1,]))
            state.vec <- c(state.vec, as.integer(as.logical(i1 %in% sub.pop)
        }
        i0 <- i1
    }
    return(list(nodes,degree.vec, state.vec))
}
```

In [243…

```
sub.pop
```

4 · 5 · 92 · 72 · 31 · 3 · 14 · 51 · 97 · 58 · 63 · 75 · 87 · 34 · 23 · 66 · 55 · 9 · 100 · 70 · 95 · 60 · 13 · 50 · 2 · 69 · 49 · 85 · 15 · 40

In [244…

```
Test <- Random_walk_Random_sampling_modified(g,sub.pop,100)
Test
```

1. 35 · 29 · 36 · 76 · 21 · 89 · 77 · 63 · 1 · 58 · 24 · 98 · 47 · 59 · 10 · 81 · 15 · 85 · 55 · 50 · 9 · 100 · 92 · 22 · 90 · 44 · 32 · 56 · 49 · 14 · 70 · 62 · 39 · 61 · 38 · 23 · 96 · 4 · 72 · 51 · 42 · 69 · 60 · 8 · 18 · 26 · 17 · 46 · 41 · 86 · 3 · 45 · 37 · 65 · 48 · 74
2. 10 · 5 · 6 · 8 · 6 · 5 · 6 · 5 · 10 · 6 · 6 · 7 · 14 · 8 · 9 · 8 · 8 · 7 · 4 · 11 · 6 · 6 · 7 · 4 · 9 · 10 · 3 · 4 · 12 · 8 · 10 · 11 · 7 · 10 · 5 · 6 · 7 · 3 · 5 · 7 · 2 · 6 · 4 · 8 · 7 · 8 · 10 · 11 · 4 · 3 · 5 · 6 · 6 · 5 · 6 · 5
3. 0 · 0 · 0 · 0 · 0 · 0 · 0 · 1 · 0 · 1 · 0 · 0 · 0 · 0 · 0 · 0 · 1 · 1 · 1 · 1 · 1 · 1 · 1 · 0 · 0 · 0 · 0 · 0 · 1 · 1 · 1 · 0 · 0 · 0 · 0 · 1 · 0 · 1 · 1 · 1 · 0 · 1 · 1 · 0 · 0 · 0 · 0 · 0 · 0 · 0 · 1 · 0 · 0 · 0 · 0 · 0

7/ Implement the estimator P_{10} and P_{01} seen during the class. You can complete the following code. Reminder:

$$P_{0,1} = \frac{\sum_{k=1}^{K} 1_{x(k)=0} 1_{x(k+1)=1}}{\sum_{k=1}^{K} 1_{x(k)=0}},$$

where $x(k)$ is the state of the node visited at the k-th step of the random walk and $1_.$ is the indicator function.

In [245…

```
Num10<-c()
Denom10<-c()
for(n in 1:(length(Test[[3]])-1)){
    Denom10<-c(Denom10,Test[[3]][n]==1)
    Num10<-c(Num10,(Test[[3]][n]==1)*(Test[[3]][n+1]==0))
}

Num01<-c()
Denom01<-c()
for(n in 1:(length(Test[[3]])-1)){
    Denom01<- c(Denom01,Test[[3]][n]==0)
    Num01<- c(Num01,(Test[[3]][n]==0)*(Test[[3]][n+1]==1))
}
P10 = sum(Num10)/sum(Denom10)
P01 = sum(Num01)/sum(Denom01)
print(P10)
print(P01)
```

```
[1] 0.4210526
[1] 0.2222222
```

8/ Implement the estimator of the average degree of each subpopulation. Reminder:

$$D_0 = \frac{\sum_{k=1}^{K} 1_{x(k)=0}}{\sum_{k=1}^{K} \frac{1_{x(k)=0}}{deg(i_k)}}$$

where $deg(i_k)$ is the degree of the node visited at the k-th step of the random walk.

In [246…

```
Num1<-c()
Denom1<-c()
Num0<-c()
Denom0<-c()

for(n in 1:(length(Test[[3]])-1)){
    Num0 = c(Num0,Test[[3]][n] == 0)
    Num1 = c(Num1,Test[[3]][n] == 1)
    Denom0 <- c(Denom0,(Test[[3]][n]==0)/(Test[[2]][n]))
    Denom1 <- c(Denom1,(Test[[3]][n]==1)/(Test[[2]][n]))
}

Deg.1<-sum(Num1)/sum(Denom1)
Deg.0<-sum(Num0)/sum(Denom0)

print(Deg.1)
print(Deg.0)
```

```
[1] 5.901573
[1] 5.945273
```

9/ Derive the final estimator for the proportion of individual in state 1. What is happening when the number of observations increases? (Try simulations with 100, 200 and 1000 observations)

In [247…

```
Deg.0*P01/(Deg.0*P01+Deg.1*P10)
```

0.347124603745697

In [248…

```
# All what has been done up to here is wrapped in function estimate_state_pr
# Inputs are the adjacency matrix g, the number of observations nb.samples a
estimate_state_proportion<-function(g, sub.pop, nb.sample){
    random_walk_data <- Random_walk_Random_sampling_modified(g, sub.pop, nb.

    Num10<-c()
    Denom10<-c()
    for(n in 1:(length(random_walk_data[[3]])-1)){
        Denom10<-c(Denom10,random_walk_data[[3]][n]==1)
        Num10<-c(Num10,(random_walk_data[[3]][n]==1)*(random_walk_data[[3]][
    }

    Num01<-c()
    Denom01<-c()
    for(n in 1:(length(random_walk_data[[3]])-1)){
        Denom01<- c(Denom01,random_walk_data[[3]][n]==0)
        Num01<- c(Num01,(random_walk_data[[3]][n]==0)*(random_walk_data[[3]]
    }
    P10 = sum(Num10)/sum(Denom10)
    P01 = sum(Num01)/sum(Denom01)

    Num1<-c()
    Denom1<-c()
    Num0<-c()
    Denom0<-c()

    for(n in 1:(length(random_walk_data[[3]])-1)){
        Num0 = c(Num0,random_walk_data[[3]][n] == 0)
        Num1 = c(Num1,random_walk_data[[3]][n] == 1)
        Denom0 <- c(Denom0,(random_walk_data[[3]][n]==0)/(random_walk_data[[
        Denom1 <- c(Denom1,(random_walk_data[[3]][n]==1)/(random_walk_data[[
    }

    Deg.1<-sum(Num1)/sum(Denom1)
    Deg.0<-sum(Num0)/sum(Denom0)

    return(Deg.0*P01/(Deg.0*P01+Deg.1*P10))
}
```

In [249…

```
# Values for nb.samples
sample_values <- c(100, 200, 1000)

# Iterate over nb.samples values
for (nb_samples in sample_values) {
  result <- estimate_state_proportion(g, sub.pop, nb_samples)
  cat("nb.samples:", nb_samples, "Result:", result, "\n")
}
```

```
nb.samples: 100 Result: 0.3621949
nb.samples: 200 Result: 0.3456746
nb.samples: 1000 Result: 0.335749
```

As the number of observations increases the estimation of the proportion of nodes in state 1 becomes more precise (0.3 is the correct value).