SNA Final Project

AUTHOR
Chelsea Rodrigues– 23200333

```r
#Loading necessary libraries
library(igraph)
```

```
Warning: package 'igraph' was built under R version 4.3.3


Attaching package: 'igraph'

The following objects are masked from 'package:stats':

    decompose, spectrum

The following object is masked from 'package:base':

    union
```

```r
library(blockmodels)
```

```
Warning: package 'blockmodels' was built under R version 4.3.3
```

```r
library(Bergm)
```

```
Warning: package 'Bergm' was built under R version 4.3.3

Loading required package: ergm

Warning: package 'ergm' was built under R version 4.3.3

Loading required package: network

Warning: package 'network' was built under R version 4.3.3


'network' 1.18.2 (2023-12-04), part of the Statnet Project
* 'news(package="network")' for changes since last version
* 'citation("network")' for citation information
* 'https://statnet.org' for help, support, and other information


Attaching package: 'network'

The following objects are masked from 'package:igraph':

    %c%, %s%, add.edges, add.vertices, delete.edges, delete.vertices,
    get.edge.attribute, get.edges, get.vertex.attribute, is.bipartite,
    is.directed, list.edge.attributes, list.vertex.attributes,
    set.edge.attribute, set.vertex.attribute
```

```
'ergm' 4.6.0 (2023-12-17), part of the Statnet Project
* 'news(package="ergm")' for changes since last version
* 'citation("ergm")' for citation information
* 'https://statnet.org' for help, support, and other information

'ergm' 4 is a major update that introduces some backwards-incompatible
changes. Please type 'news(package="ergm")' for a list of major
changes.
```

```
library(intergraph)
```

```
Warning: package 'intergraph' was built under R version 4.3.3
```

```
library(latentnet)
```

```
Warning: package 'latentnet' was built under R version 4.3.3


'latentnet' 2.11.0 (2024-02-19), part of the Statnet Project
* 'news(package="latentnet")' for changes since last version
* 'citation("latentnet")' for citation information
* 'https://statnet.org' for help, support, and other information
NOTE: BIC calculation prior to latentnet 2.7.0 had a bug in the calculation of the effective
number of parameters. See help(summary.ergmm) for details.
NOTE: Prior to version 2.8.0, handling of fixed effects for directed networks had a bug: the
covariate matrix was transposed.
```

```
#Loading the data
load("C:/Users/chels/Downloads/SNA/data_eurovision.RData")
```

Question 1: Degree Calculations

Calculate the binary in-degrees and out-degrees (using X) and their weighted counterparts (using Y). Find out the 5 most voted countries, and calculate the number of countries that voted for each of the 5 most voted countries.

```
# Binary degrees using X
in_degrees_binary <- colSums(X)
out_degrees_binary <- rowSums(X)

# Weighted degrees using Y
in_degrees_weighted <- colSums(Y)
out_degrees_weighted <- rowSums(Y)

# Ensure country labels are assigned to in_degrees_weighted
names(in_degrees_weighted) <- countries_legend$country
rownames(X) <- countries_legend$country
colnames(X) <- countries_legend$country

# Now get the top 5 voted countries
```

```r
top_5 <- sort(in_degrees_weighted, decreasing = TRUE)[1:5]
top_5_countries <- names(top_5)

# Calculate the number of countries that voted for each top country
votes_for_top_5 <- colSums(X[, top_5_countries])

cat("\nTop 5 voted countries\n")
```

```

Top 5 voted countries
```

```r
print(top_5_countries)
```

```
[1] "Sweden"   "Italy"    "Ukraine"  "Russia"   "Bulgaria"
```

```r
cat("\nThe number of countries that voted for each top country\n")
```

```

The number of countries that voted for each top country
```

```r
print(votes_for_top_5)
```

```
  Sweden    Italy  Ukraine   Russia Bulgaria 
      28       29       27       18       17 
```

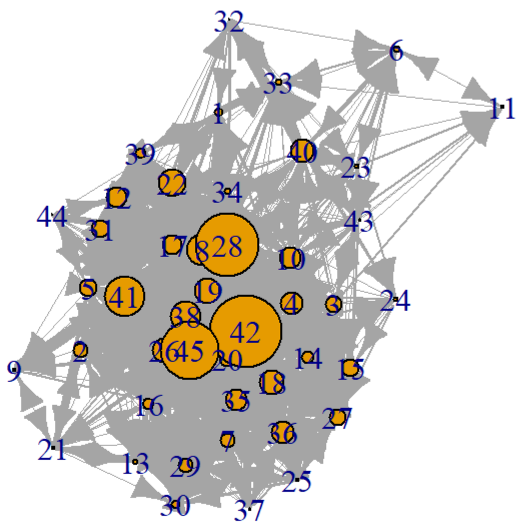
Question 2: Visualizing Network Y

Create a clear visual representation of the network Y, where the size of the nodes indicates the weighted degree, the edge width indicates the number of votes, and the countries labels are shown.

```r
#Y is adjacency matrix with weighted values
graph_Y <- graph.adjacency(Y, mode = "directed", weighted = TRUE)
```

```
Warning: `graph.adjacency()` was deprecated in igraph 2.0.0.
ℹ Please use `graph_from_adjacency_matrix()` instead.
```

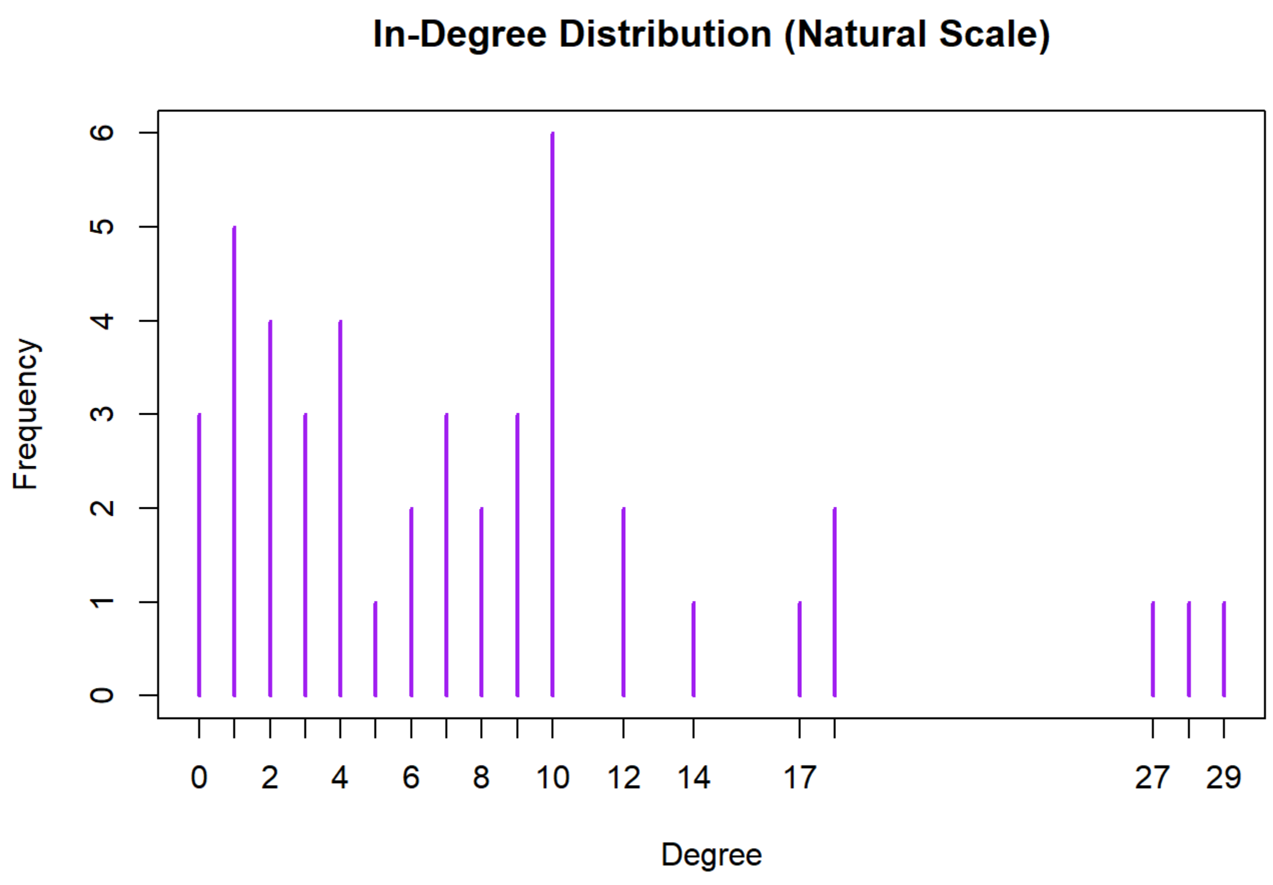
```r
# Scale node size
V(graph_Y)$size <- in_degrees_weighted / max(in_degrees_weighted) * 30
# Scale edge width
E(graph_Y)$width <- E(graph_Y)$weight / max(E(graph_Y)$weight) * 10
#plotting the graph
plot(graph_Y)
```



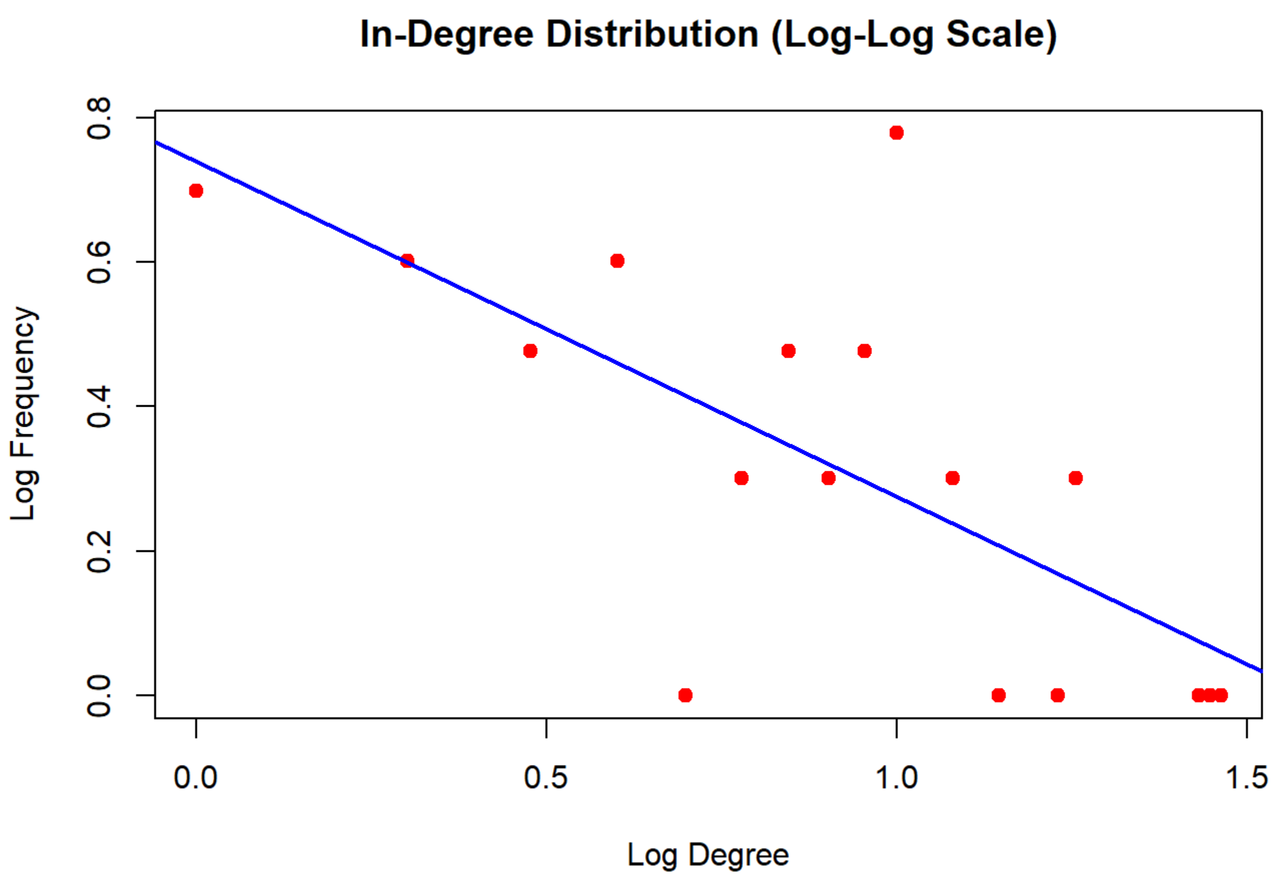
Question 3: Degree Distribution in Natural and Log-Log Scale

Consider the distribution of the binary in-degrees of X. Create a graphical representation of the in-degree distribution both in natural scale and in log-log scale. Is there any evidence of a powerlaw behaviour in the tail of the distribution?

```
in_degrees_binary_distribution <- table(in_degrees_binary)

# plotting Natural scale plot
plot(in_degrees_binary_distribution, main = "In-Degree Distribution (Natural Scale)", xlab = "
```



```r
# Converting to log scale
log_log_scale <- log10(as.numeric(names(in_degrees_binary_distribution)))
log_freq <- log10(as.numeric(in_degrees_binary_distribution))

# Filter out NA, NaN, and Inf values from the log-transformed data
valid_indices <- !is.na(log_log_scale) & !is.na(log_freq) &
                 is.finite(log_log_scale) & is.finite(log_freq)

# Perform a linear regression on the valid log-log data
fit <- lm(log_freq[valid_indices] ~ log_log_scale[valid_indices])

# Plot Log-log scale plot with valid data
plot(log_log_scale[valid_indices], log_freq[valid_indices],
     main = "In-Degree Distribution (Log-Log Scale)",
     xlab = "Log Degree",
     ylab = "Log Frequency",
     col = "red", pch = 19)

# Add the regression line (linear fit) to the plot
abline(fit, col = "blue", lwd = 2)
```



Yes there is slight evidence of a powerlaw behaviour in the tail of the distribution as most points are near to the line.

Question 4: Undirected Graph Analysis

Create an undirected version of X, called X_und, whereby X_und[i,j] is 1 if either X[i,j] = 1 or X[j,i] = 1 (otherwise X_und[i,j] = 0). For X_und: calculate the average total degree, the clustering coefficient, the average path length. Also, calculate the average degree of the neighbours of a node as a function of the node's degree, and comment on the mixing with respect to the degrees.

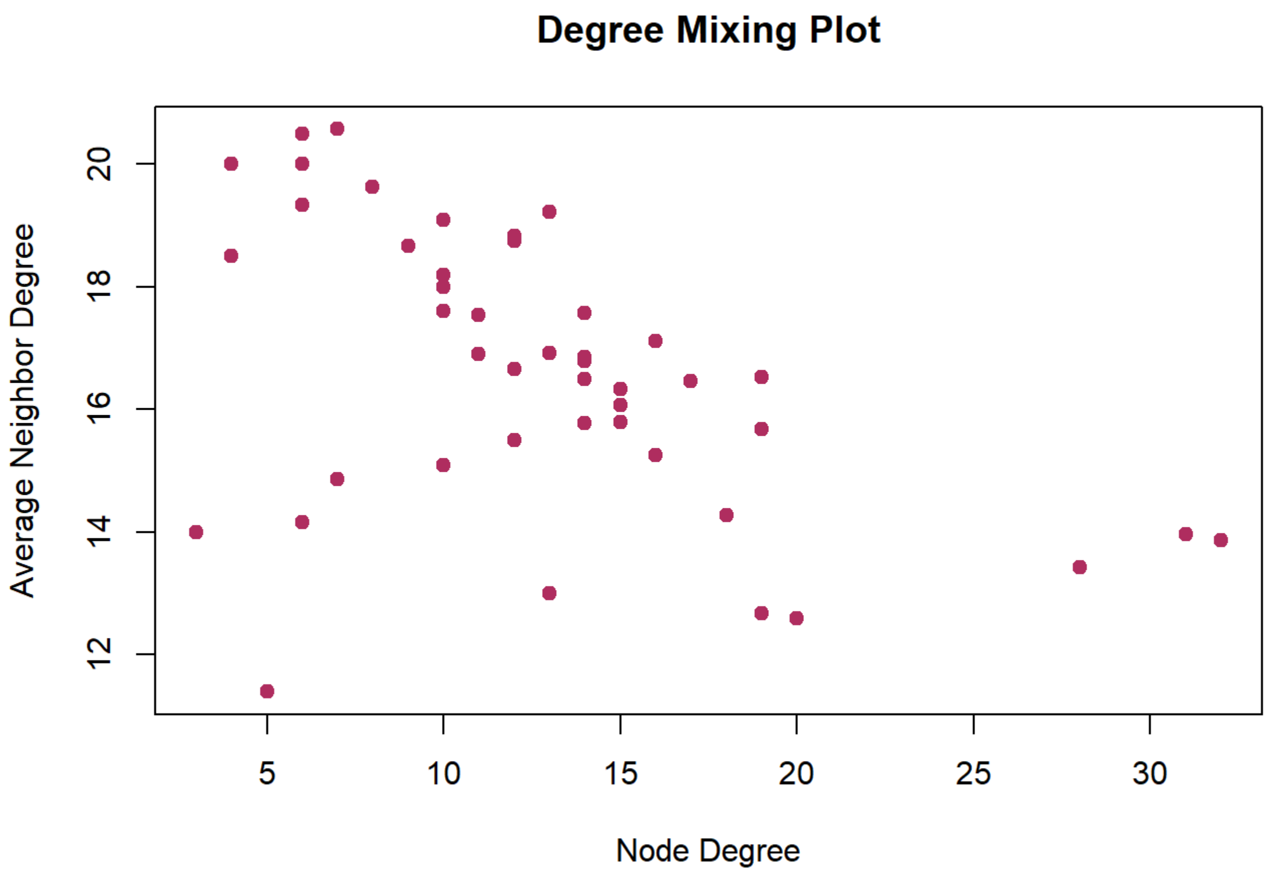
```r
X_und <- as.matrix((X + t(X)) > 0)

# Average total degree
avg_total_degree <- mean(rowSums(X_und))

# Clustering coefficient
clustering_coeff <- transitivity(graph.adjacency(X_und, mode = "undirected"), type = "global")

# Average path length
avg_path_length <- average.path.length(graph.adjacency(X_und, mode = "undirected"))
```

```
Warning: `average.path.length()` was deprecated in igraph 2.0.0.
ℹ Please use `mean_distance()` instead.
```

```
# Degree mixing
deg <- degree(graph.adjacency(X_und, mode = "undirected"))
avg_neighbor_deg <- graph.knn(graph.adjacency(X_und, mode = "undirected"))$knn
```

```
Warning: `graph.knn()` was deprecated in igraph 2.0.0.
ℹ Please use `knn()` instead.
```

```
plot(deg, avg_neighbor_deg, xlab = "Node Degree", ylab = "Average Neighbor Degree", main = "De
```



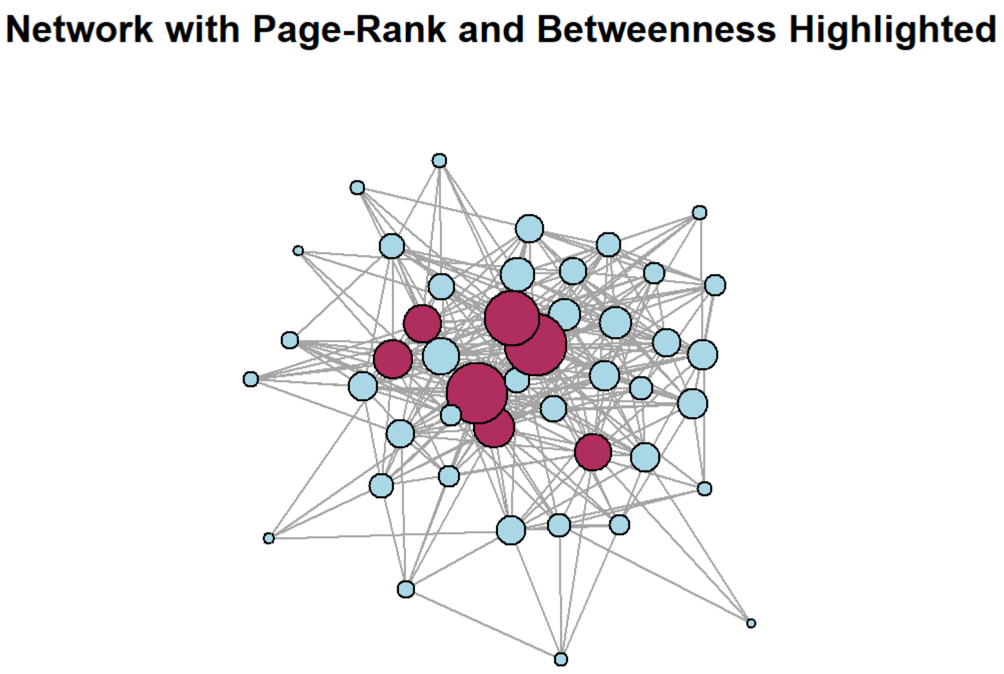
Question 5: Page-Rank and Betweenness Centrality

Calculate the Page-Rank centrality scores and betweenness centrality scores on X_und. Plot the network with node's size proportional to Page-Rank, and highlight with a different color the nodes that are in the top 10 according to both centralities.

```
# Convert adjacency matrix to undirected graph
graph_und <- graph.adjacency(X_und, mode = "undirected")

# Calculate Page-Rank centrality scores
page_rank <- page.rank(graph_und)$vector
```

```
Warning: `page.rank()` was deprecated in igraph 2.0.0.
ℹ Please use `page_rank()` instead.
```

```r
# Calculate Betweenness centrality scores
betweenness <- betweenness(graph_und)

# Identify the top 10 nodes for both PageRank and Betweenness centrality
top_10 <- intersect(order(page_rank, decreasing = TRUE)[1:10], order(betweenness, decreasing =

# Set node sizes proportional to Page-Rank centrality
V(graph_und)$size <- page_rank * 500

# Highlight nodes in the top 10 for both Page-Rank and Betweenness centrality
V(graph_und)$color <- ifelse(V(graph_und) %in% top_10, "maroon", "lightblue")

# Plot the graph with node size proportional to Page-Rank and top nodes highlighted
plot(graph_und, vertex.label = NA, main = "Network with Page-Rank and Betweenness Highlighted"
```



Question 6: E-R Random Graph Model

Find the numerical value of the maximum likelihood estimator of the parameter p of a E-R random graph model for X_und. Calculate the average degree k. Hypothesise an asymptotical setting where the number of nodes N increases and the average degree k remains constant. Point out the asymptotic value for the clustering coefficient and determine the asymptotic regime for the giant component's size

```r
k <- mean(rowSums(X_und))
p <- k / nrow(X_und)  # Maximum likelihood estimator for p
cat("Estimated k:", k, "\n")
```

```
Estimated k: 12.88889
```

```r
cat("Estimated p:", p, "\n")
```

```
Estimated p: 0.2864198
```

```r
# Asymptotic values
asymptotic_clustering <- p
giant_component_regime <- ifelse(k > 1, "Giant component exists", "No giant component")
cat("\n")
```

```r
cat(giant_component_regime)
```

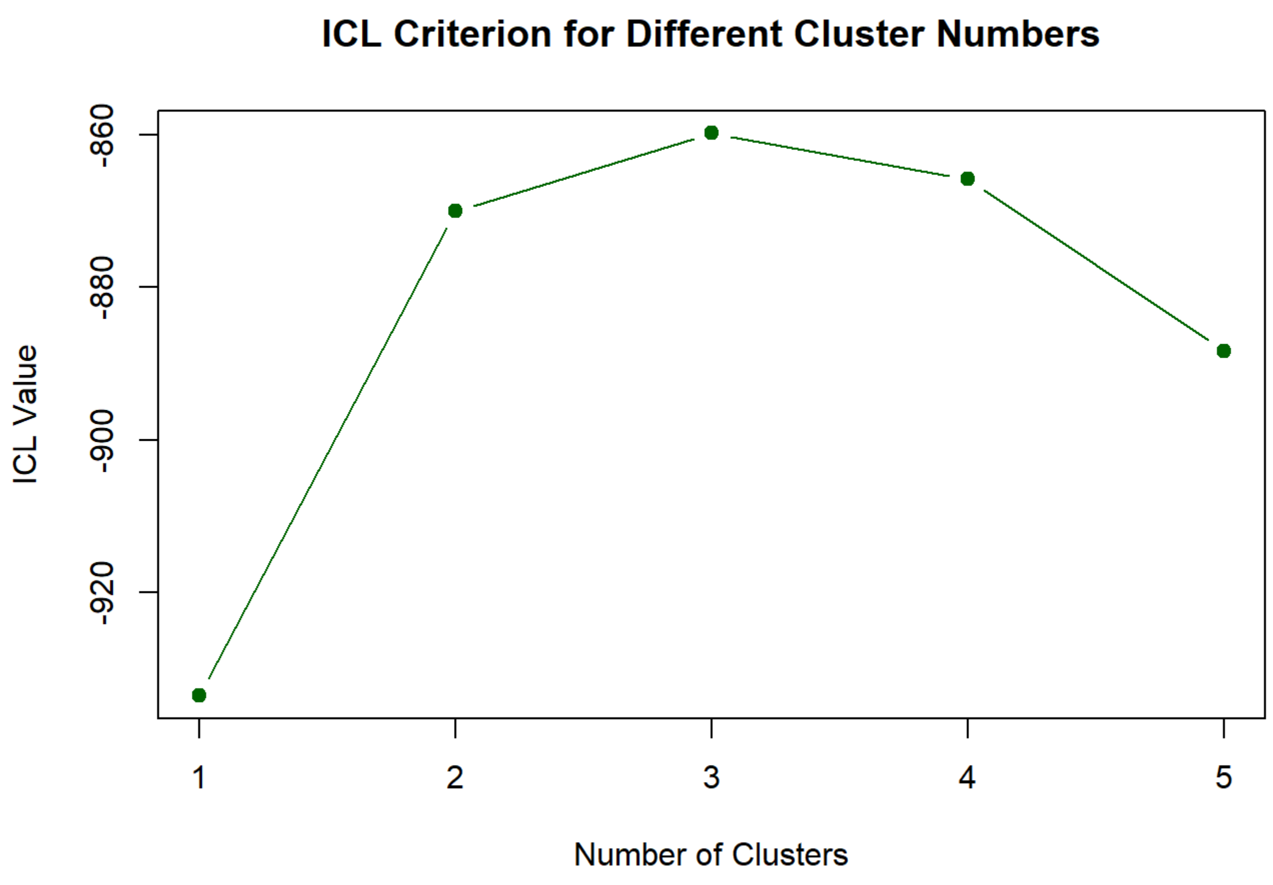
```
Giant component exists
```

Question 7: Stochastic Block Model

Using the package blockmodels, fit a Stochastic Block Model on X (directed network). You should choose the best model according to the Integrated Completed Likelihood criterion. Discuss the results: is there any evidence of assortative or disassortative mixing with respect to the cluster labels? Discuss the connectivity behaviour of the various groups.

```r
X_matrix <- as.matrix(X)
# Fitting a Stochastic Block Model using blockmodels for different numbers of clusters (K)
set.seed(123)
sbm_fit <- BM_bernoulli("SBM", X_matrix, verbosity = 0, plotting = "none")

# Running the SBM for a range of cluster numbers and select the best model based on ICL
sbm_fit$estimate()
```

```r
# Plotting the ICL values to visually check the best number of clusters
plot(sbm_fit$ICL, type = "b", main = "ICL Criterion for Different Cluster Numbers", xlab = "Nu
```



```r
# Selecting the model with the highest ICL value
best_model <- which.max(sbm_fit$ICL)

# Extract the clustering results for the best model
cluster_map_method <- sbm_fit$memberships[[best_model]]$map()

str(cluster_map_method)
```

```
List of 1
 $ C: int [1:45] 1 1 3 3 1 1 3 2 1 3 ...
```

```r
clusters <- cluster_map_method
cluster_table <- table(clusters)
print("Cluster distribution:")
```

```
[1] "Cluster distribution:"
```

```r
cat("\n")
```

```r
print(cluster_table)
```

```
C
 1  2  3 
26  5 14 
```

```r
# Analyzeing the connectivity pattern between groups
connectivity_matrix <- sbm_fit$model_parameters[[best_model]]$pi
print("Connectivity matrix between groups:")
```

```
[1] "Connectivity matrix between groups:"
```

```r
cat("\n")
```

```r
print(connectivity_matrix)
```

```
           [,1]      [,2]      [,3]
[1,] 0.08322375 0.5525297 0.1528354
[2,] 0.12891989 0.3987611 0.1719772
[3,] 0.05172608 0.5377108 0.3976003
```

1. Assortative vs. Disassortative Mixing:

   - Assortative Mixing: Occurs when nodes within the same cluster have higher connection probabilities (i.e., higher values on the diagonal of the connectivity matrix).
   - Disassortative Mixing: Occurs when nodes are more likely to connect across clusters (i.e., higher values in the off-diagonal elements).

   Noticeably, the off-diagonal values between Clusters 1 and 2 (0.553) and between Clusters 3 and 2 (0.538) are much higher than the diagonal values. This suggests disassortative mixing because nodes are more likely to connect across different clusters rather than within the same cluster. The network demonstrates disassortative mixing, where nodes are more likely to connect across different clusters rather than within their own cluster.

2. Connectivity Behavior Between Groups:

   Cluster 2 appears to act as a bridge, facilitating connections between the larger clusters (Clusters 1 and 3). The overall structure suggests that while clusters are defined, the interactions are predominantly cross-cluster, indicating that the network's communication or influence flow is more inter-group than intra-group.

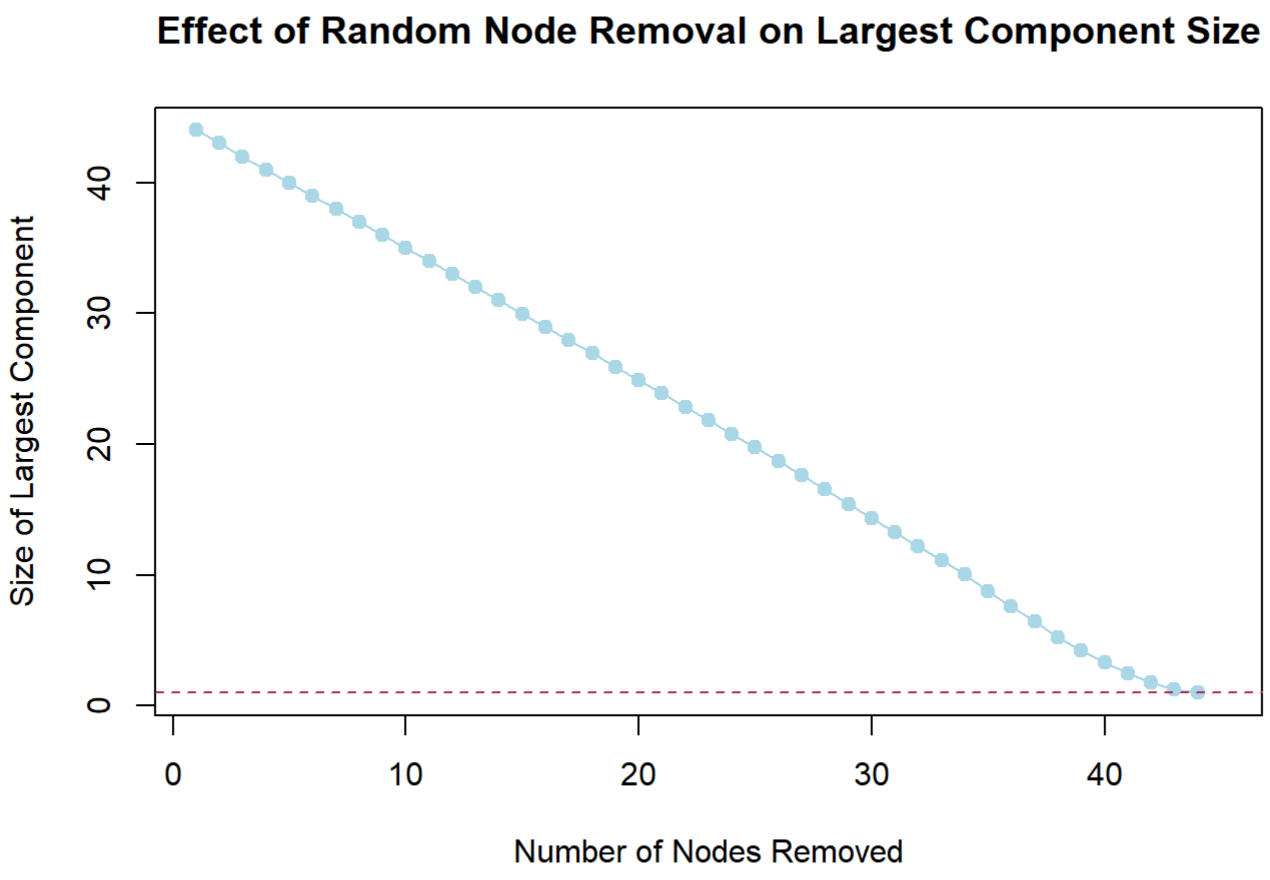
Question 8: Random Node Removal Simulation

Perform a simulation study where the nodes of X_und are removed sequentially and uniformly at random. Consider the size of the largest component and monitor how this changes during the process. Comment on the results.

```r
# Convert the directed network X into an undirected network
X_und <- as.matrix((X + t(X)) > 0)

# Create an igraph object from the undirected adjacency matrix
graph_und <- graph.adjacency(X_und, mode = "undirected")

# Function to simulate the node removal process
simulate_node_removal <- function(graph, num_simulations = 1) {
  num_nodes <- vcount(graph)
  largest_components <- matrix(0, nrow = num_simulations, ncol = num_nodes)

  for (sim in 1:num_simulations) {
    # Randomly shuffle the node indices for this simulation
    nodes <- sample(V(graph))
    current_graph <- graph

    for (i in seq_along(nodes)) {
      # Remove the node using the name or ID, which stays consistent across graphs
      current_graph <- delete_vertices(current_graph, nodes[i]$name)

      # Record the size of the largest connected component
      largest_components[sim, i] <- max(components(current_graph)$csize)
    }
  }

  # Calculate the average size of the largest component across all simulations
  avg_largest_components <- colMeans(largest_components)

  return(avg_largest_components)
}

# Run the simulation (e.g., with 100 simulations)
set.seed(123)  # For reproducibility
avg_largest_component_sizes <- simulate_node_removal(graph_und, num_simulations = 100)
```

```
Warning in max(components(current_graph)$csize): no non-missing arguments to
max; returning -Inf

Warning in max(components(current_graph)$csize): no non-missing arguments to
max; returning -Inf

Warning in max(components(current_graph)$csize): no non-missing arguments to
max; returning -Inf

Warning in max(components(current_graph)$csize): no non-missing arguments to
max; returning -Inf

Warning in max(components(current_graph)$csize): no non-missing arguments to
max; returning -Inf

Warning in max(components(current_graph)$csize): no non-missing arguments to
max; returning -Inf

Warning in max(components(current_graph)$csize): no non-missing arguments to
max; returning -Inf

Warning in max(components(current_graph)$csize): no non-missing arguments to
max; returning -Inf

Warning in max(components(current_graph)$csize): no non-missing arguments to
max; returning -Inf

Warning in max(components(current_graph)$csize): no non-missing arguments to
Warning in max(components(current_graph)$csize): no non-missing arguments to
max; returning -Inf
```

```
# Plot the results
plot(avg_largest_component_sizes, type = "o", col = "lightblue",pch=19,
     xlab = "Number of Nodes Removed", ylab = "Size of Largest Component",
     main = "Effect of Random Node Removal on Largest Component Size")
abline(h = 1, col = "maroon", lty = 2)
```



Analysis:

1. Gradual Decline: First of all, the value of the largest component index reduces gradually and consistently from 44 to 39 while the nodes are being eliminated.

2. Accelerated Fragmentation: For the component size, it is getting more apparent that the size is shrinking around node 12 at 32. 99. When more nodes are taken off the network, the largest component begins to fragment into even smaller components.

3. Rapid Collapse: Starting the node number 34 onwards, the largest component has a steep downfall. From this it is apparent that the network is no longer connected as a single large unit yet it is fragmenting into small units or nodes.

4. Negative Infinity (-Inf): The appearance of -Inf implies that, finally, Here, at last, all the components of the network are divided in such a way that there are no more components to quantify.

The plot of the simulation reveals that, in the case of randomly removing some nodes, the small-world network is fairly immune at the beginning, after which the size of the largest component drops sharply as soon as a number of nodes is taken out. The results thus suggest the need for the monitoring of connectivity especially in those networks, where there is need to always ensure that a large connected component is preserved.

Question 9: Targeted Node Removal Simulation

Perform a simulation study where the nodes of X_und are removed sequentially using one of 3 methods: • by degree • by eigenvector centrality • by Page-Rank centrality in each of the methods the node with largest value is removed at each iteration. In case of ties, use smaller number-label first. Monitor how the size of the largest component changes and discuss on the results. Remove nodes until no nodes remain.

```r
# Convert the directed network X into an undirected network
X_und <- as.matrix((X + t(X)) > 0)
graph_und <- graph.adjacency(X_und, mode = "undirected")

# Function to simulate node removal based on a centrality measure
simulate_targeted_removal <- function(graph, measure) {
  num_nodes <- vcount(graph)
  largest_components <- numeric(num_nodes)

  current_graph <- graph

  for (i in 1:num_nodes) {
    # Calculate the specified centrality measure
    if (measure == "degree") {
      centrality <- degree(current_graph)
    } else if (measure == "eigenvector") {
      centrality <- evcent(current_graph)$vector
    } else if (measure == "pagerank") {
      centrality <- page_rank(current_graph)$vector
    }

    # Get the node with the highest centrality value
    # In case of ties, choose the node with the smallest index
    max_node <- which.max(centrality)

    # Remove the node
    current_graph <- delete_vertices(current_graph, max_node)

    # Record the size of the largest connected component
    largest_components[i] <- max(components(current_graph)$csize)
  }

  return(largest_components)
}

# Run simulations for each centrality measure
set.seed(123)  # For reproducibility
degree_removal <- simulate_targeted_removal(graph_und, "degree")
```

```
Warning in max(components(current_graph)$csize): no non-missing arguments to
max; returning -Inf
```

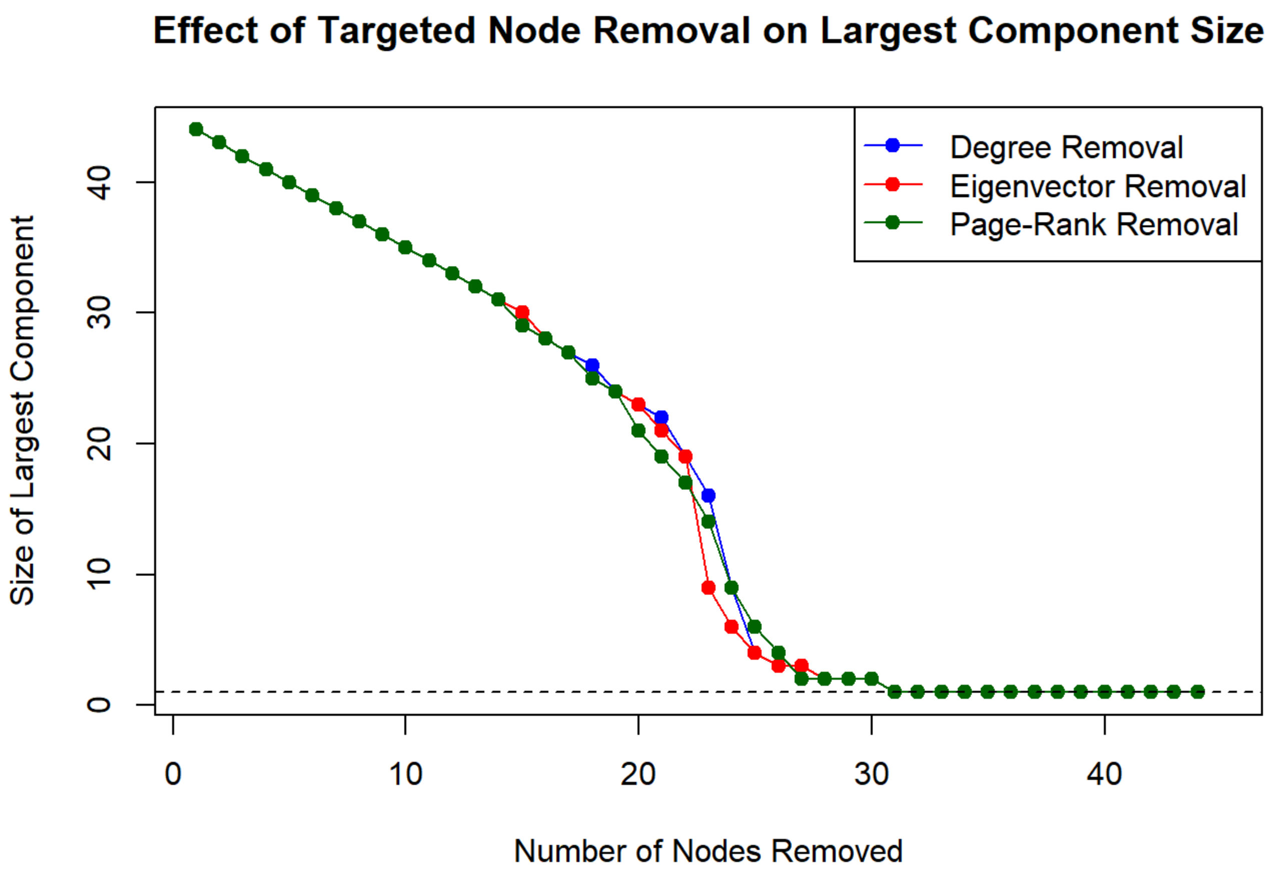
```
eigenvector_removal <- simulate_targeted_removal(graph_und, "eigenvector")
```

```
Warning: `evcent()` was deprecated in igraph 2.0.0.
ℹ Please use `eigen_centrality()` instead.
no non-missing arguments to max; returning -Inf
```

```
pagerank_removal <- simulate_targeted_removal(graph_und, "pagerank")
```

```
Warning in max(components(current_graph)$csize): no non-missing arguments to
max; returning -Inf
```

```
# Plot the results
plot(degree_removal, type = "o", col = "blue",pch=19, xlab = "Number of Nodes Removed", ylab =
lines(eigenvector_removal, type = "o", col = "red",pch=19)
lines(pagerank_removal, type = "o", col = "darkgreen",pch=19)
legend("topright", legend = c("Degree Removal", "Eigenvector Removal", "Page-Rank Removal"), c
abline(h = 1, col = "black", lty = 2)  # Mark when the largest component becomes trivial
```



Question 10: Exponential Random Graph Model

Fit an Exponential Random Graph Model on X_und using the package Bergm and summary statistics: • the number of edges (edges) • the number of two-stars (kstar(2)) • the number of triangles (triangle) You should run the algorithm without changing the sampling parameters, e.g. the number of iterations. Based on the convergence diagnostics plots, comment on whether the results obtained can be considered reliable. Based on the Bayesian goodness of fit plots, express an opinion on whether this may be a good model for the data.

```
# Convert the directed network X into an undirected network
X_und <- as.matrix((X + t(X)) > 0)
# Create an igraph object from the undirected adjacency matrix
graph_und <- graph.adjacency(X_und, mode = "undirected")
# Convert the igraph object to a network object
network_und <- asNetwork(graph_und)
# Fit the Exponential Random Graph Model using Bergm
ergm_fit <- bergm(
  network_und ~ edges + kstar(2) + triangle
)
```

```
 > MCMC start
```

```
print("Summary")
```

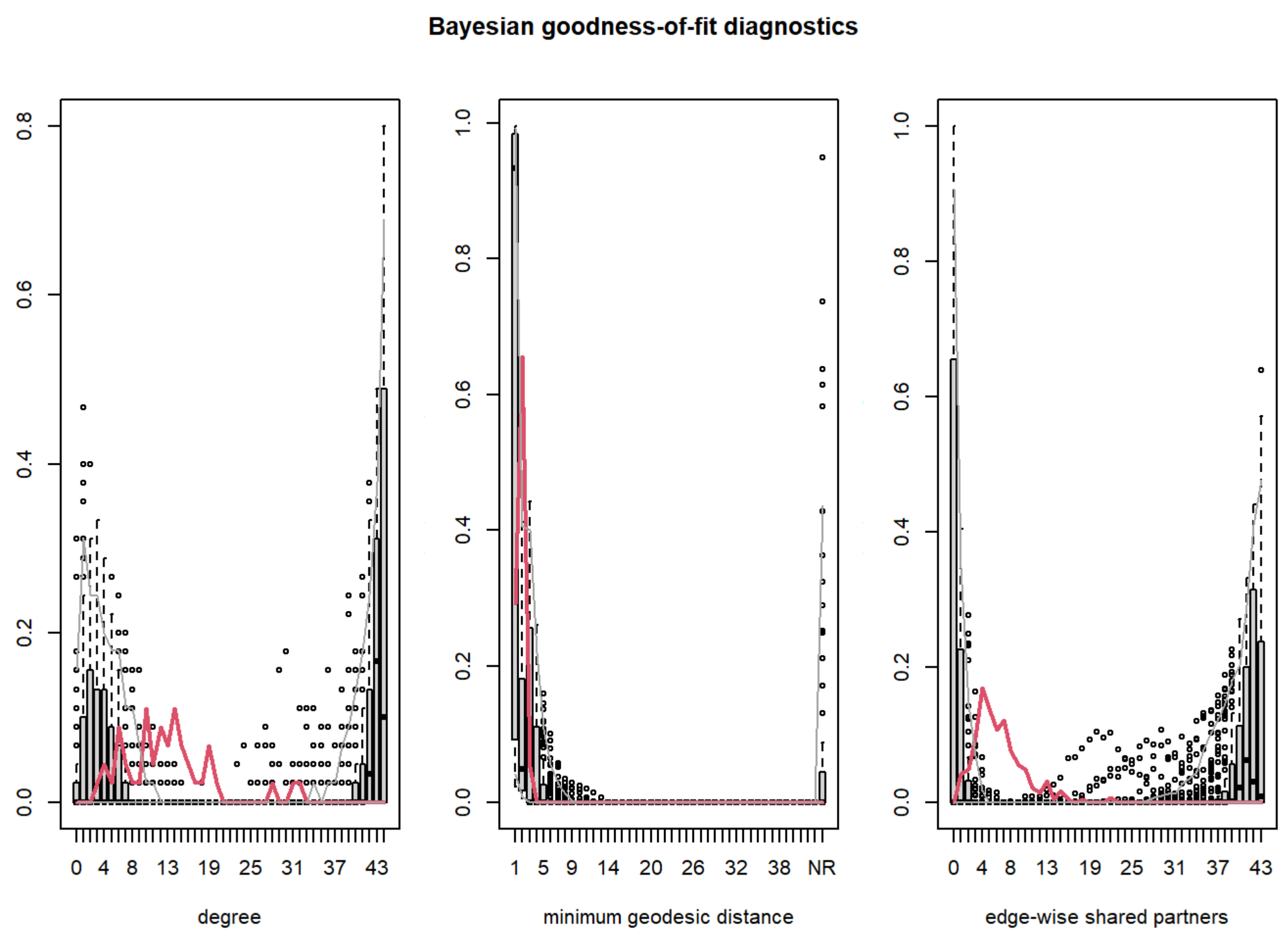
```
[1] "Summary"
```

```
cat("\n")
```

```
print(summary(ergm_fit))
```

```

 Posterior Density Estimate for Model: y ~ edges + kstar(2) + triangle

                          Mean         SD     Naive SE Time-series SE
theta1 (edges)    -2.88846340 0.43317798 0.0055923037    0.113923849
theta2 (kstar2)    0.04944641 0.02754718 0.0003556325    0.006807083
theta3 (triangle)  0.15535325 0.08034553 0.0010372564    0.017594985

                         2.5%         25%         50%        75%      97.5%
theta1 (edges)    -3.754022920 -3.18736067 -2.91153259 -2.5300184 -2.1624257
theta2 (kstar2)    0.004287378  0.02827661  0.04470475  0.0719288  0.0974851
theta3 (triangle)  0.012208813  0.08836354  0.17151974  0.2259078  0.2828638

 Acceptance rate: 0.02


NULL
```

```
# Bayesian goodness-of-fit (GOF) diagnostics
gof_fit <- bgof(ergm_fit)
```



Yes,this may be good model for the data based on bayesian goodness of fit plots

Question 11: Euclidean Latent Position Model

Fit a Euclidean Latent Position Model on X_und with two latent dimensions, using a Bayesian method. Use the BIC method to choose the number of latent clusters (consider up to 3 clusters), and plot the latent space for the optimal model.

```
library(latentnet)

# Fit the Euclidean Latent Position Model (LPM) with Bayesian approach
model_1_cluster <- ergmm(network_und ~ euclidean(d=2, G=1), verbose = FALSE)
```

```
Warning in backoff.check(model, burnin.sample, burnin.control): Backing off:
too few acceptances. If you see this message several times in a row, use a
longer burnin.
```

```
model_2_clusters <- ergmm(network_und ~ euclidean(d=2, G=2), verbose = FALSE)
model_3_clusters <- ergmm(network_und ~ euclidean(d=2, G=3), verbose = FALSE)

# Compare models using BIC
BIC_values <- c(bic.ergmm(model_1_cluster)$Z, bic.ergmm(model_2_clusters)$Z, bic.ergmm(model_3
```

NOTE: It is not certain whether it is appropriate to use latentnet's BIC to select latent space dimension, whether or not to include actor-specific random effects, and to compare clustered models with the unclustered model.

```
best_model_index <- which.min(BIC_values)
best_model <- list(model_1_cluster, model_2_clusters, model_3_clusters)[[best_model_index]]

# Print BIC values and the chosen number of clusters
cat("BIC Values for Models:\n")
```

```
BIC Values for Models:
```

```
print(BIC_values)
```

```
[1] 280.6057 308.5126 295.1507
```

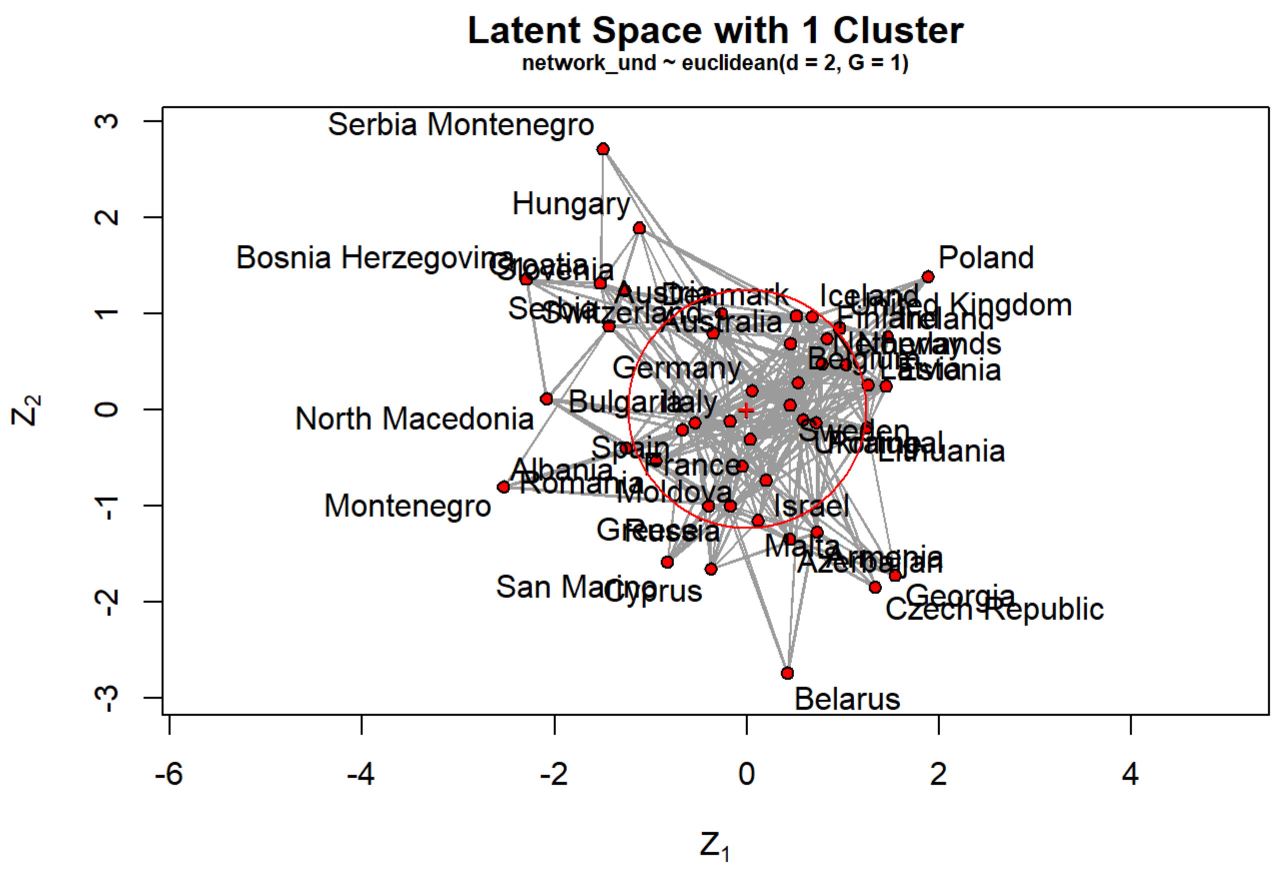
```
cat("\nBest Model:\n")
```

```

Best Model:
```

```
print(summary(best_model))
```

```

==========================
Summary of model fit
==========================

Formula:   network_und ~ euclidean(d = 2, G = 1)
Attribute: edges
Model:     Bernoulli
MCMC sample of size 4000, draws are 10 iterations apart, after burnin of 10000 iterations.
Covariate coefficients posterior means:
            Estimate   2.5%  97.5% 2*min(Pr(>0),Pr(<0))
(Intercept)   1.6057 1.3915 1.8577            < 2.2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Overall BIC:        1198.385
Likelihood BIC:     917.7797
Latent space/clustering BIC:     280.6057

Covariate coefficients MKL:
            Estimate
(Intercept) 0.841757
```

```
# Plot the latent space for the optimal model
num_clusters <- best_model_index
if (num_clusters > 1) {
  # Use pie chart if there are multiple clusters
  plot(best_model, pie = TRUE, labels = TRUE, main = paste("Latent Space with", best_model_ind
} else {
  # Fallback to a simple plot without pie charts for a single cluster
  plot(best_model, labels = TRUE, main = paste("Latent Space with", best_model_index, "Cluster
}
```


Latent Space with 1 Cluster
network_und ~ euclidean(d = 2, G = 1)
Serbia Montenegro
Hungary
Bosnia Herzegovina
Croatia
Slovenia
Poland
Austria
Denmark
Iceland
United Kingdom
Serbia
Switzerland
Australia
Finland
Ireland
Netherlands
Norway
Germany
Belgium
Latvia
Estonia
Bulgaria
Italy
North Macedonia
Sweden
Ukraine
Portugal
Lithuania
Spain
France
Albania
Romania
Moldova
Montenegro
Israel
Greece
Russia
Malta
Armenia
Azerbaijan
San Marino
Cyprus
Georgia
Czech Republic
Belarus
Z_1
Z_2