UCLA

ECE 232 - Large Scale Social and Complex Networks: Design and Algorithms
Spring 2021

Project 1: Random Graphs and Random Walks

Authors:
Swapnil Sayan Saha (UID: 605353215)
Grant Young (UID: 505627579)

GENERATING RANDOM NETWORKS

Question 1(a):

In this question, we are asked to plot the degree distribution of several undirected random networks with varying probabilities of drawing an edge between two arbitrary vertices, as well as comparison between empirical and theoretical mean and variance of degree distributions. To create the random networks, we use the `sample_gnp()` function from the `igraph` library in R, which creates an Erdos-Renyi random network with n nodes, with the edge formation probability between two vertices being p.

An Erdos-Renyi network is created by randomly cutting each edge within a fully connected graph with probability 1-p. An Erdos-Renyi network is said to be undirected if there is no direction associated with the edges of the graph. For undirected graphs, the degree of a vertex v_i is defined as:

$$deg(v_i) = ||\{(v_i, v_j) \in E\}||$$

where, E is the set of edges of the graph. The degree distribution of the graph, which is the probability distribution function of the random variable X, denoting the degree of a randomly selected vertex of the graph, is given by:

$$\mathcal{P}(X = k) = \frac{\text{number of vertices with degree } k}{\text{total number of vertices}}$$

For an Erdos-Renyi network, $\mathcal{P}(X = k)$ is defined as a binomial distribution:

$$\mathcal{P}(X = k) = \begin{pmatrix} n \\ k \end{pmatrix} p^k (1-p)^{n-k}$$

As p increases, the average degree α of an Erdos-Renyi network increases linearly as well. As a result, for increasing values of p and constant n, we expect the degree distribution to shift towards the right.

$$\alpha = np$$

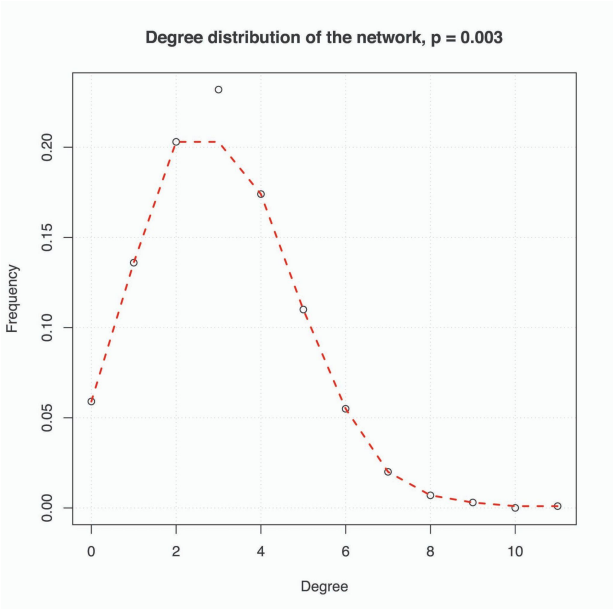
If n is large compared to α, then $\mathcal{P}(X = k)$ follows a Poisson distribution, with the curve being skewed towards the right for low mean:

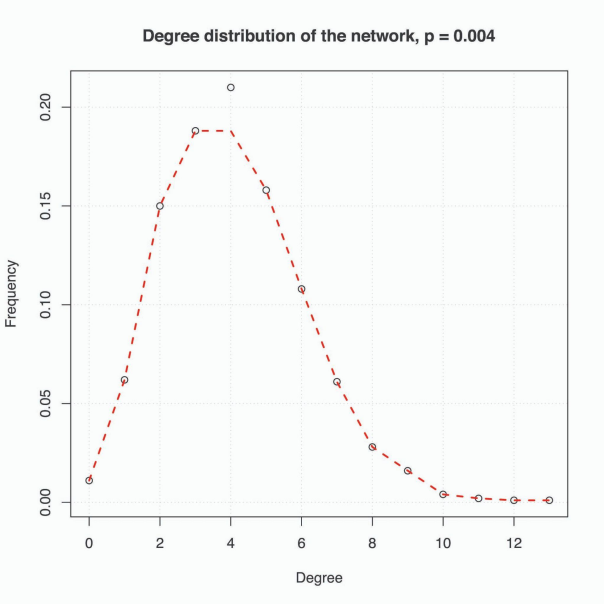
$$\mathcal{P}(X = k)_{\alpha << n} = \frac{e^{-\alpha}\alpha^k}{k!}$$

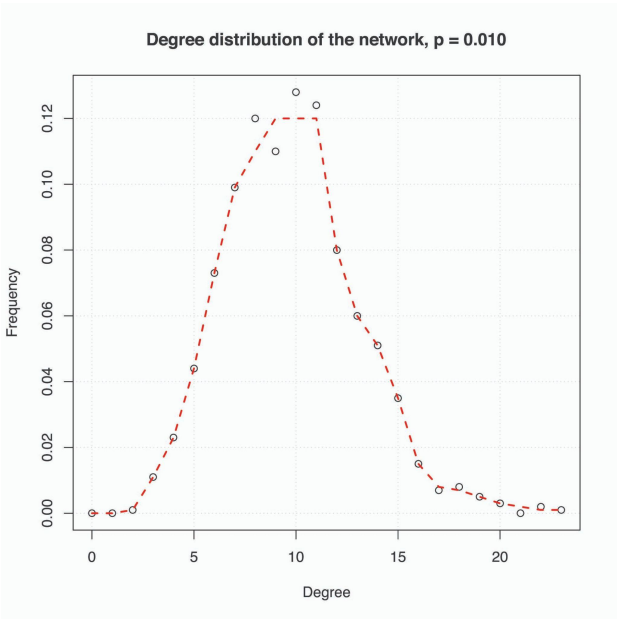
In addition, as p increases, the variance of the distribution should increase as well:

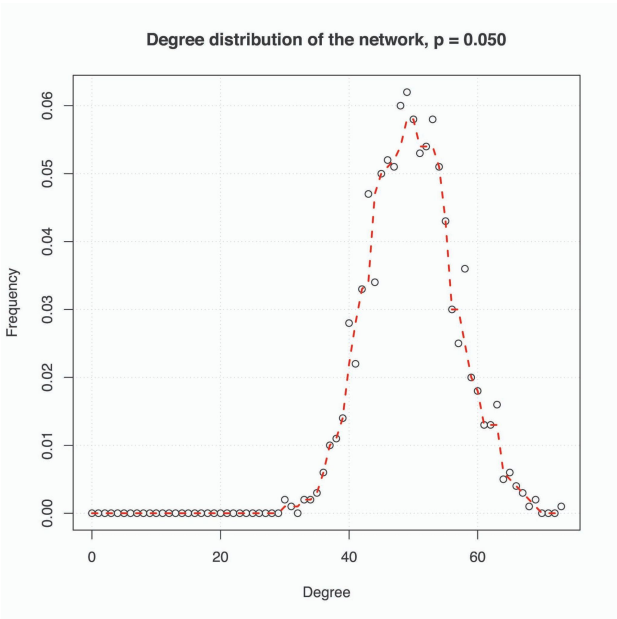
$$\sigma^2 = np(1-p)$$

Figure 1 shows the degree distributions for an undirected Erdos-Renyi random network with 1000 vertices and various values of p (0.003, 0.004, 0.01, 0.05 and 0.1).









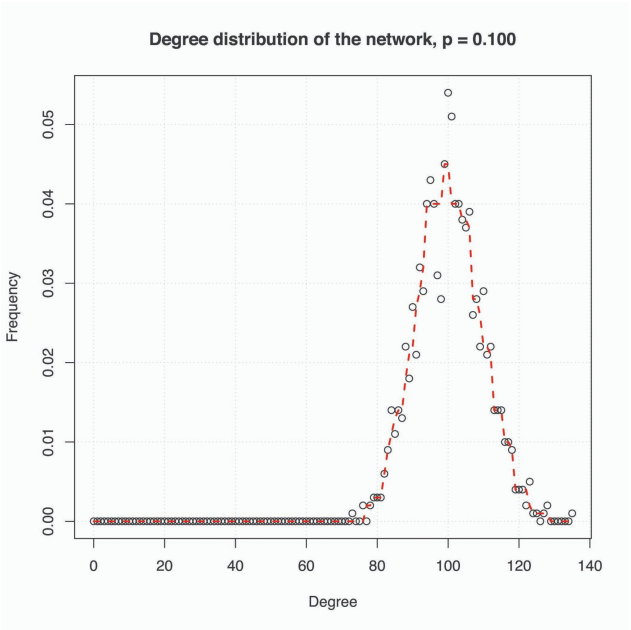


Figure 1: Degree distributions for undirected random networks with 1000 vertices and varying values of p.

From Figure 1, we can observe that for small values of p, the degree distribution follows a Poisson distribution with a right skew as explained earlier. For larger values of p, the distribution follows the expected Binomial distribution. As α increases with increasing values of p, the normalized frequency (or probability) of the α decreases, while the variance of the distribution σ^2 increases (the Q-value of the distribution decreases). In other words, with increasing values of p, we would expect the degree distribution to be wider with a larger mean, with the normalized frequency of α decreasing. Table 1 shows the theoretical and experimental values of the mean and variance of the five degree distributions shown in Figure 1.

Table 1: Theoretical and empirical mean and variance of undirected random networks with varying values of p

p (n = 1000)	$\alpha = np$	α (experimental)	$\sigma^2 = np(1-p)$	σ^2 (experimental)
0.003	3	3.048	2.991	3.054751
0.004	4	4.086	3.984	3.892496
0.01	10	9.776	9.9	10.372196
0.05	50	49.88	47.5	44.820420
0.1	100	100.35	90	85.502186

From Table 1, we can see that the theoretical and experimental values of mean and variance for the undirected random networks with varying values of p are in close agreement with each other (~ < 5% error).

Question 1(b):

In this question, we are asked to find out if all instances of the five undirected random networks are connected or not, along with their numerical probability of being connected. In addition, we are asked to find the giant connected component (GCC) and its diameter for a non-connected random network. An undirected graph is connected if there exists a path between any pair of vertices within the graph. Likewise, an undirected graph is disconnected if there exists at least one pair of vertices that does not have a path between them. For a disconnected graph, the connected component with the highest number of vertices is called the GCC. Table 2 shows the probability of connectedness, whether all random realizations of the network are connected or not and the diameter, number of vertices and number of edges of GCC if a random realization is not connected. We created 1000 random undirected networks for each value of p to calculate the probability of connectedness. From Table 2, we see that as the p increases, the probability of connectedness increases non-monotonically. This is because the connectivity of Erdos-Renyi network depends on np, as shown in Figure 2.

Table 2: Connectedness and GCC properties of undirected random networks with varying values of p

p (n = 1000)	All realizations connected?	Probability of connectedness	Diameter of GCC (if not connected)	Number of vertices of GCC (if not connected)	Number of edges of GCC (if not connected)
0.003	No	0	13	949	1570
0.004	No	0	11	977	2052
0.01	No	0.96	6	999	5162
0.05	Yes	1	-	-	-
0.1	Yes	1	-	-	-

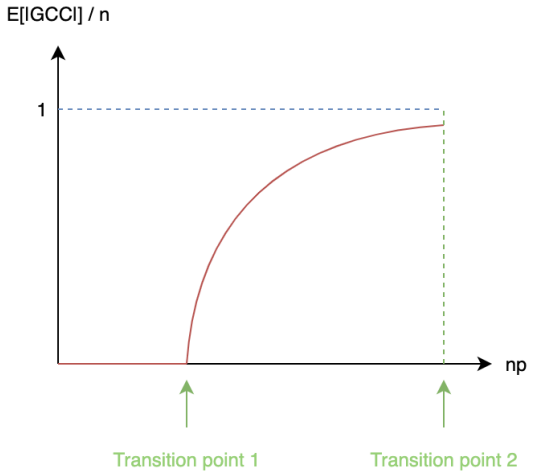


Figure 2: Expected size of GCC versus np for an undirected Erdos-Renyi network.

At transition point 2, we have a network that is fully connected (probability of connectedness is 1). According to Cayley's theorem, at transition point 2:

$$np = c\ln(n)$$

$$p = c\frac{\ln(n)}{n}$$

By observation, for $n = 1000$ and the observed undirected random networks, if $c > 1 \approx 1.00001$, then $p = 0.007$, which is slightly smaller than 0.01. Thus, we should get a fully connected Erdos-Renyi network if $p > 0.007$ for $n = 1000$. This makes sense, because the probability of connectedness when $p = 0.01$ is 0.96, which is very close to 1. Beyond $p = 0.01$, we see that all realizations of the Erdos-Renyi network are fully connected.

In addition, we observe that the diameter of non-connected GCC decreases as p increases, with both the connectivity among the vertices as well as number of vertices of GCC increasing with increasing values of p. From Figure 2, at transition point 2, the expected size of the GCC should be equal to the number of nodes of the network. For $p = 0.01 \approx 0.007$, we see that the GCC of a random network has 999 out of a possible of 1000 nodes. In addition, the diameter of the GCC is approximated by:

$$D \approx \frac{ln(n)}{ln(\alpha)}$$

As p increases, the average degree α of an undirected Erdos Renyi network increases, causing the diameter of the GCC to decrease.

Question 1(c):

In this question, we are asked to find the empirical values of transition points 1 and 2 shown in Figure 2 in last question, along with an empirical plot of the expected size of GCC versus p for undirected Erdos-Renyi networks. We sweeped p from 0 to 0.01 in steps of 0.0001, as we empirically proved in the last question that the network is almost surely connected when p = 0.01 (according to Cayley's theorem). For each value of p, we created 100 random realizations of Erdos-Renyi networks. Figure 3 shows the expected size of GCC versus p for $n = 1000$, as well as scatter plots of normalized GCC sizes for all 100 realizations for each value of p.

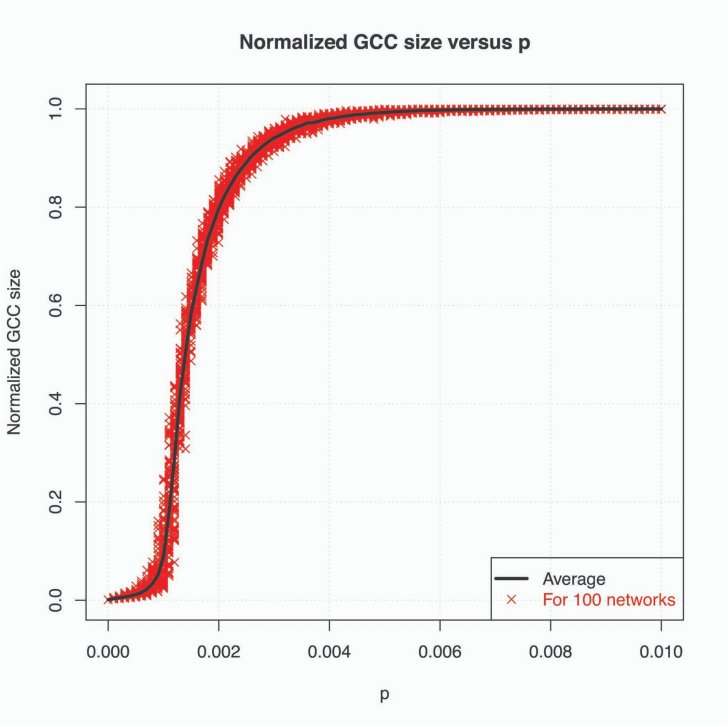


Figure 3: Normalized and expected GCC sizes versus p for undirected Erdos-Renyi network ($n = 1000$).

From Figure 3, we can see that the expected GCC size increases non-monotonically with increasing values of p, similar to the phase transitions in the theoretical plot in Figure 2. Similar to the theoretical plot, we can see that the curve eventually converges to the total number of vertices of the graph, with the GCC starting to appear at transition point 1.

I. Theoretically, the value of p at which GCC starts to appear (transition point 1) is given by:

$$p_{T1} = \frac{1}{n} = \mathcal{O}\left(\frac{1}{n}\right)$$

From the above equation, we observe that the GCC should start to emerge when $p = 0.001$ for $n = 1000$. At transition point 1, the criterion of emergence, or the expected GCC size is given by:

$$\mathbb{E}(|GCC|)_{T1} \propto \sqrt{n} \approx \epsilon \cdot n^2 p$$

We choose ϵ to be 0.0001. Thus, our criterion of emergence is defined as the value of p at which the expected GCC size is 0.1 (or 10% of the maximum GCC size). From Figure 2, the empirical value of p at transition point 1 is 0.0011. This matches closely with the theoretical value of 0.001.

II. Theoretically, the value of p at which the expected GCC size is almost the same as the number of vertices of the network (transition point 2) is given by:

$$p_{T2} = c\frac{\ln(n)}{n} = \mathcal{O}\left(\frac{\ln(n)}{n}\right)$$

From the above equation, the Erdos-Renyi network is surely connected (more than 99% of the nodes belong to GCC), when $p = 0.007$ for $n = 1000$ and $c = 1.00001$. Empirically, the value of p that we obtain is 0.0073, which is quite close to the theoretical value.

Question 1(d):

I. In this question, we are asked to plot the expected size of GCC for undirected Erdos-Renyi networks versus n, where $p = c/n$, average degree of nodes, c is equal to 0.5 and n ranges from 100 to 10000. For each value of n, we create 100 random realizations of Erdos-Renyi networks. Figure 4 shows the plot of expected GCC size versus n for $c = 0.5$.

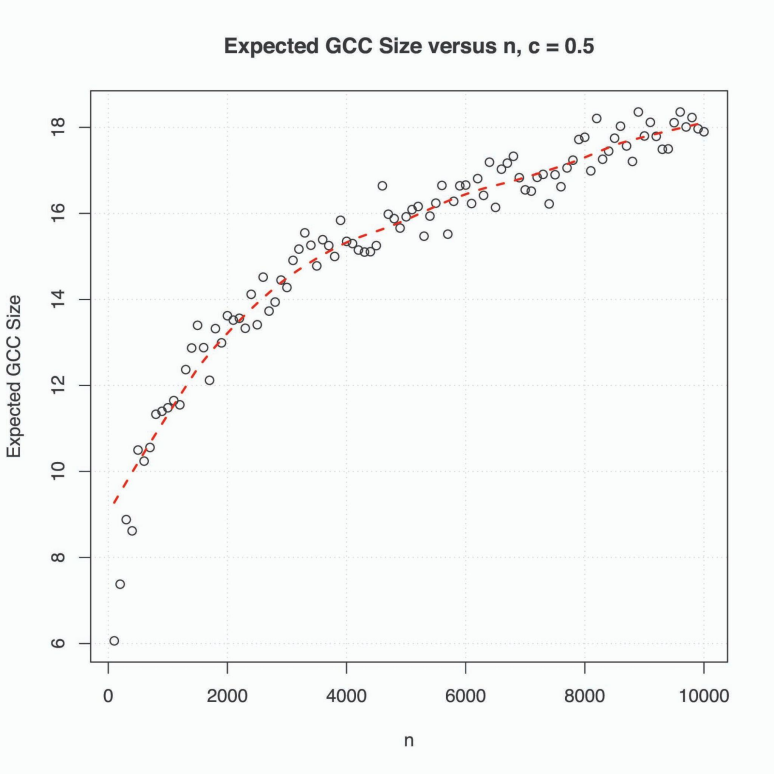


Figure 4: Expected GCC size versus n, average degree of network = 0.5.

From Figure 4, we see that as n increases, the expected GCC size increases and converges to a constant value as $n \to \infty$, following a natural logarithmic trend. As n increases, since c is constant, p decreases. If $np = c < 1$, then an undirected Erdos-Renyi network will almost surely have no connected components of size larger than $\mathcal{O}(ln(n))$. In other words, for $c < 1$, with high probability, the expected size of a connected component is given as:

$$\mathbb{E}(s) < ln(n)$$

For $c < 1$, every connected component in an Erdos-Renyi network has at most one-cycle, with the number of vertices being a growing logarithmic function. The GCC in such a graph is an isolated tree with $\mathcal{O}(ln(n))$ vertices.

II. In this question, we are asked to plot the expected size of GCC for undirected Erdos-Renyi networks versus n, where $p = c/n$, average degree of nodes, c is equal to 1 and n ranges from 100 to 10000. For each value of n, we create 100 random realizations of Erdos-Renyi networks. Figure 5 shows the plot of expected GCC size versus n for $c = 1$.

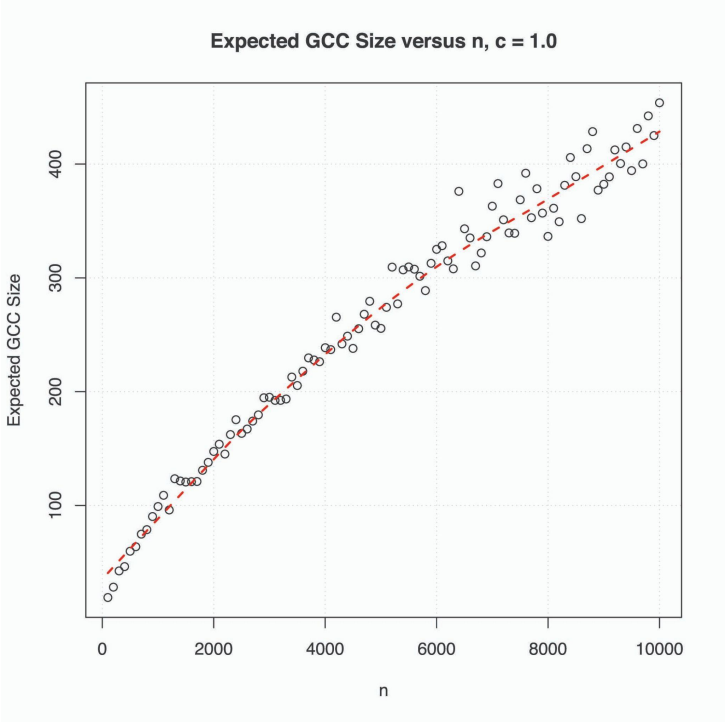


Figure 5: Expected GCC size versus n, average degree of network = 1.

From Figure 5, we see that as n increases, the expected GCC size increases. We also observe that the range of expected GCC sizes is higher for c = 1 than for c = 0.5. This is because the expected GCC size of an Erdos-Renyi network is dependent on np and given by:

$$\mathbb{E}(|GCC|) = \mathcal{O}(\epsilon \cdot np) = \mathcal{O}(\epsilon \cdot c)$$

Thus, as the average degree of the network increases, the expected GCC size also increases. For $np = c = 1$, an Erdos-Renyi network will almost surely have a largest component whose size is of order $n^{2/3}$ or $\sqrt{n}$. In other words:

$$\mathbb{E}(|GCC|)_{c=1} = \mathcal{O}(\sqrt{n}) \vee \mathcal{O}(n^{2/3})$$

As a result, we see that as n increases, the observed trend in expected GCC size increases on the order of square root of n.

III. In this question, we are asked to plot the expected size of GCC for undirected Erdos-Renyi networks versus n, where $p = c/n$, average degree of nodes, c is equal to 1.1, 1.2 and 1.3 and n ranges from 100 to 10000. For each value of n, we create 100 random realizations of Erdos-Renyi networks. Figure 6 shows the plot of expected GCC size versus n for c = 1.1, 1.2and 1.3 on the same plot.

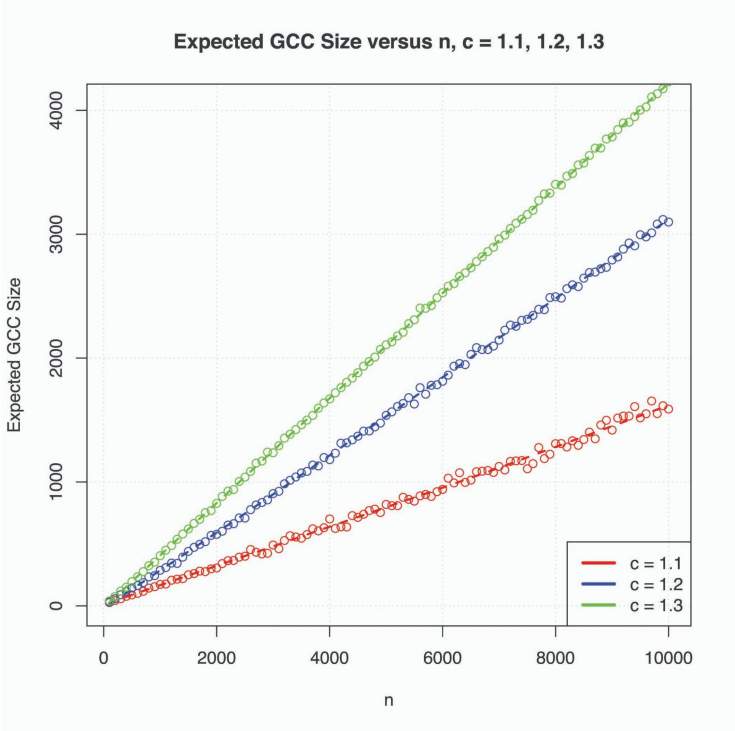


Figure 6: Expected GCC size versus n, average degree of network = 1.1, 1.2 and 1.3.

From Figure 6, we see that as n increases, the expected GCC size increases linearly. For each of the plots, as c increases, the expected GCC size for the same value of n increases as well. This is because:

$$\mathbb{E}(|GCC|) = \mathcal{O}(\epsilon \cdot np) = \mathcal{O}(\epsilon \cdot c)$$

Thus, as the average degree of the network increases, the expected GCC size also increases. Now, for $c > 1$, the expected size of GCC is a linear function of n.

$$\mathbb{E}(|GCC|) = \mathcal{O}(n) = \epsilon \cdot (np)n = \epsilon \cdot cn$$

If $c > 1$, then an Erdos-Renyi network will almost surely have a unique giant component containing a positive fraction of the vertices.

IV. Relation between expected GCC size and n for each case:

- $c = 0.5 (c < 1): \ \mathbb{E}(|GCC|) = \epsilon \cdot c \cdot \ln(n)$
- $c = 1: \ \mathbb{E}(|GCC|) = (\epsilon \cdot c \cdot \sqrt{n}) \vee (\epsilon \cdot c \cdot n^{2/3})$
- $c = 1.1, 1.2, 1.3 (c > 1): \ \mathbb{E}(|GCC|) = \epsilon \cdot cn$

Question 2(a):

In this question, we are to create an undirected random network with 1000 vertices and preferential attachment model, where each new node attaches to 1 old node (m, out degree of the vertices). We use the `barabasi.game()` function from the `igraph` library in R, which creates a scale-free (exhibits power-law degree distributions) and evolving (number of vertices in the graph grows with time) network according to the Barabasi-Albert model.

Preferential attachment refers to the fact that nodes with higher degrees or connectedness are more likely to receive edges with newer nodes. The incoming node either needs to know the degrees of other nodes in the network (global knowledge) or exploit local knowledge via random walk, where sufficiently large number of steps are walked upon randomly until a terminal node is selected, with probabilistic guarantees that the nodes are selected preferentially. Since each new node is always attached to a connected node in the existing network, the final network is theoretically always connected by construction. The algorithm is as follows:

- At time step i, a vertex/node v_i joins the network with m edges (m in our case is 1).
- At each edge, v_i chooses a node v_j proportional to the degree of v_j and makes m such edges. This is usually done through random walk.

Figure 7 shows the generated preferential attachment network.

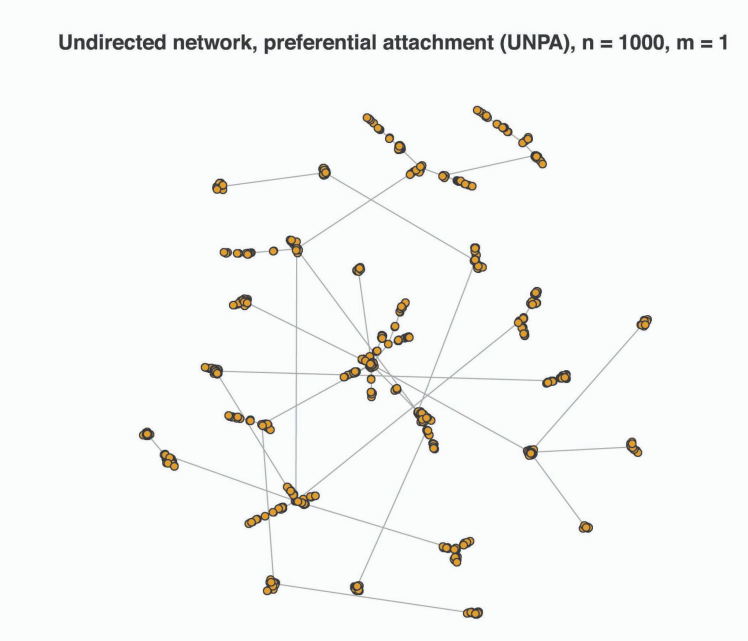


Figure 7: Undirected random network with preferential attachment model, n = 1000, m = 1.

To test whether the network is always connected or not in practice, we generated 100 random undirected networks with preferential attachment model and checked their connectedness using `is_connected()` function. All 100 networks were connected.

Question 2(b):

In this question, we are asked to use the fast greedy method to find the community structure and modularity of the generated network in Question 2(a).

Community structure of a graph is defined as a clustering of the vertices in the network such that the number of inter cluster edges is much smaller than the number of intra cluster edges. In other words, community structures segregate regions of high connectedness from the overall sparse network structure, with a simple example shown in Figure 8.

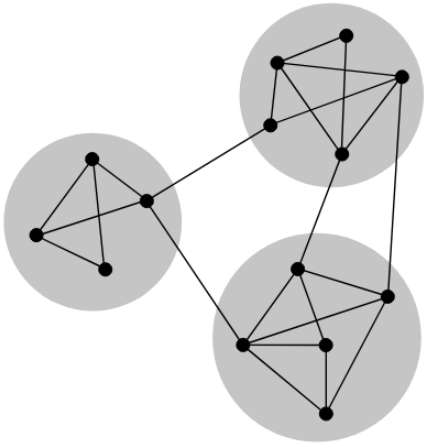


Figure 8: Community structure showing 3 clusters of high connectedness in a random graph.

To find community structures, we use Mark Newman modularity index. For a particular cluster C_i, the modularity index m_{C_i} is given as:

$$m_{C_i} = f_{C_i} - r_{C_i}$$

where, f_{C_i} is the number of edges in C_i and r_{C_i} is the expected number of edges in C_i if the network was created randomly with the same degree distribution. Expanding the equation:

$$m_{C_i} = \sum_{i,j \in C_i} \left(A_{ij} - \frac{k_i k_j}{2m} \right)$$

where, A refers to the node-node incidence matrix, k refers to the degree of nodes selected for random stub (dangling edges) matching and m is the number of edges in r_{C_i}. For a network P partitioned into k disjoint clusters, the modularity index of the network is given as:

$$Q(P) = \sum \frac{1}{2m} \sum_{i,j \in C_i} \left(A_{ij} - \frac{k_i k_j}{2m} \right)$$

The node-node incidence matrix is a mathematical representation of a graph with the rows and columns corresponding to the vertices and edges of a graph respectively. To construct A:

- If an edge j is leaving from node i, A_{ij} is +1.
- If an edge j is entering towards node i, A_{ij} is -1 (for undirected graphs, -1 and +1 will just be 1).
- If an edge j is neither entering towards node i nor leaving, A_{ij} is 0.

The maximum value of $Q(P)$ is 1. The goal of the clustering algorithm is to find the network partitioning with the highest modularity index. Intuitively, modularity measures the strength of the network division into modules with strong interconnections (community structures). The higher the modularity, the larger is the number of edges within communities, while the communities are sparsely connected to each other.

The greedy algorithm for finding out the modularity index is as follows:

- Start with n clusters.
- Compute the change in $Q(P)$ if a pair of clusters among all the existing clusters is merged and perform the merge if $Q(P)$ increases after merging.
- Repeat step 2 until no significant changes in $Q(P)$ are observed.

Figure 9 shows the community structure for the network generated in Question 2(a). Figure 10 shows the number of nodes in each of the communities.

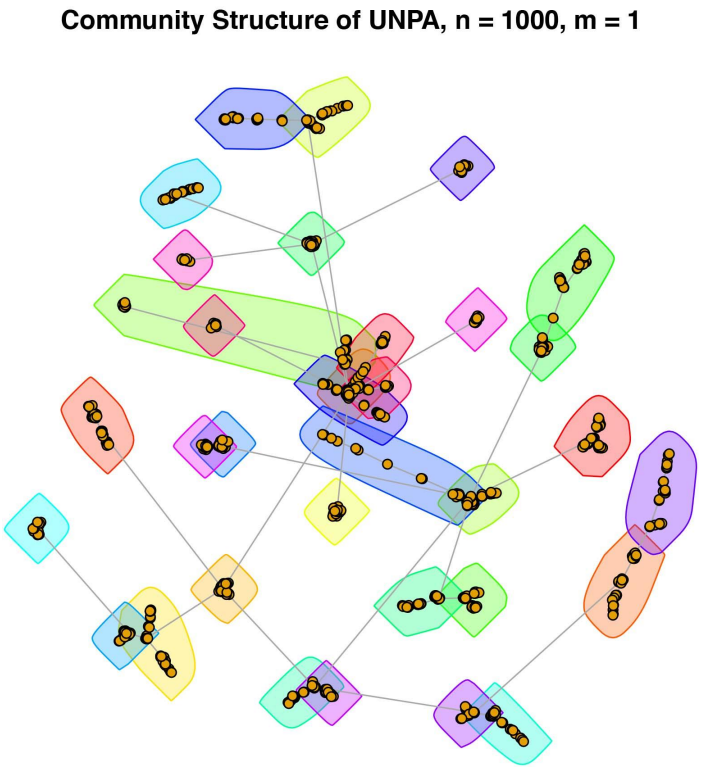


Figure 9: Community structure of an undirected network with preferential attachment, n = 1000, m = 1.

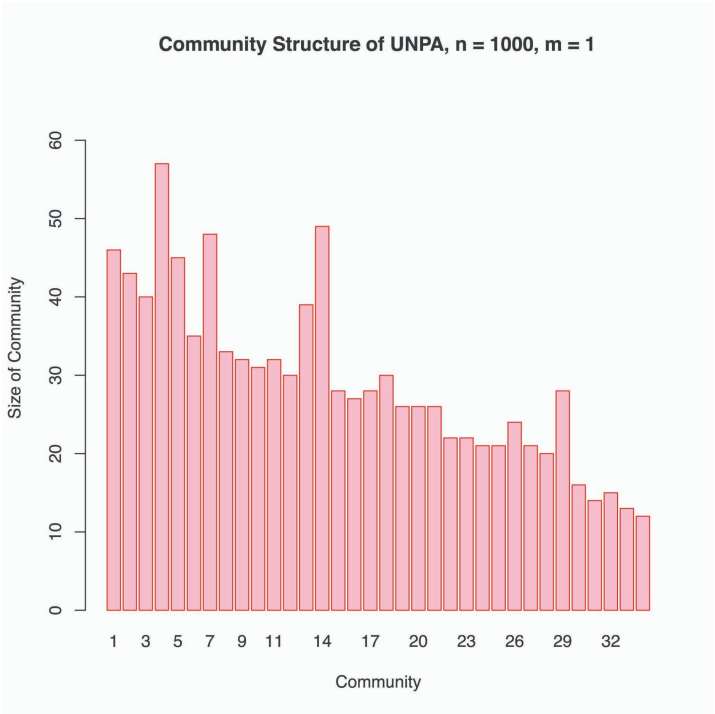


Figure 10: Number of vertices in each community within the undirected random preferential attachment network, $n = 1000$, $m = 1$.

From Figure 10, we see that the greedy method finds communities of various sizes, ranging from 12 to 57 nodes. The modularity of the network is 0.933042.

Question 2(c):

In this question, we are asked to use the fast greedy method to compare the community structure and modularity of the undirected random network with preferential attachment from Question 2(a) with another preferential attachment network but with 10000 nodes. Figure 11 shows the generated network and the community structure and Figure 12 shows the number of nodes in each of the communities.

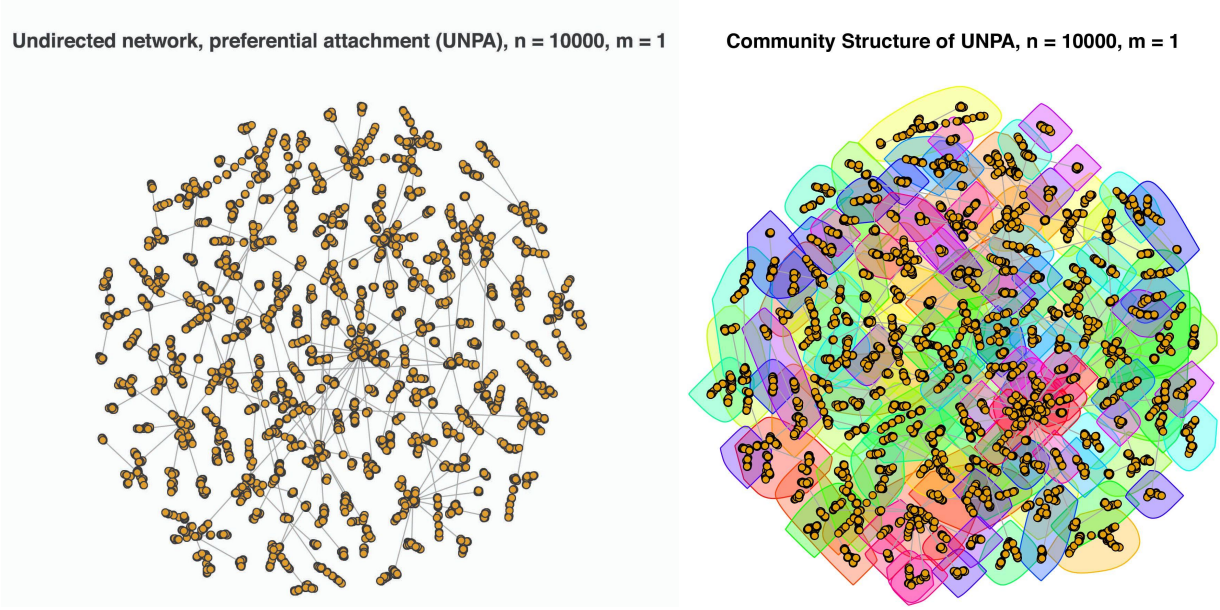


Figure 11: (Left) Undirected random network with preferential attachment model, n = 10000, m = 1. (Right) Community structure of the undirected network with preferential attachment, n = 1000, m = 1.

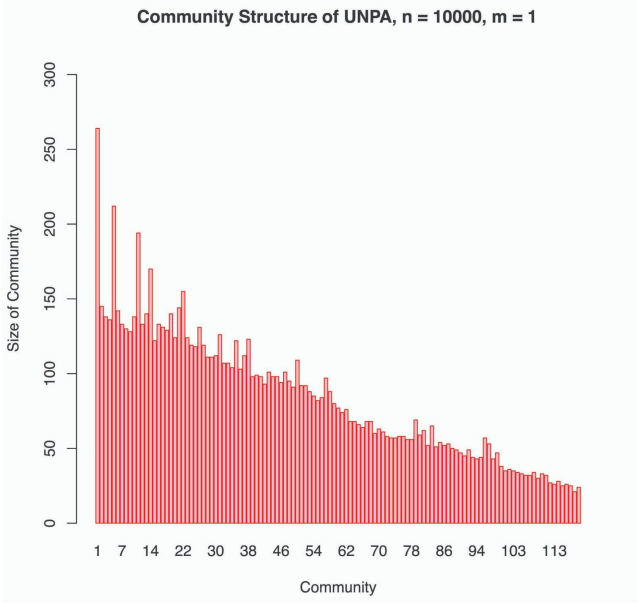


Figure 12: Number of vertices in each community within the undirected random preferential attachment network, n = 10000, m = 1.

The modularity of this network is 0.977433. From Figures 9-12, we see that there are more communities in the network with higher number of nodes (10000) compared to the smaller network (1000 nodes). In addition, the number of vertices in each community is significantly higher as well. The modularity index is also higher for the larger network over the smaller one, indicating that the larger network is better capable of being partitioned into communities with strong intra-community connectedness and sparse inter-community connectedness. This is expected because with more number of vertices, the average number of edges for nodes with high preference is going to be higher than the expected number of edges for the same high preference nodes in a smaller network. The probability of strong intra-community connections is greater for large networks over small networks, and likewise, the extent of sparsity among communities are amplified in large networks, as newer nodes are more likely to be paired with high preference nodes having a larger degree than the high preference nodes in a small network.

Question 2(d):

In this question, we are asked to plot the degree distributions in log-log scale and estimate the slope of the plots using linear regression for both the preferential attachment models found in Question 2(a) and 2(c). The preferential attachment model is an example of a power law network, which exhibits the following fat-tailed degree distribution:

$$P_k \propto \frac{1}{k^\gamma} = \frac{1}{k^\gamma \sum_{k=1}^{k_{\max}} k^{-\gamma}}, \quad k \in 1, 2, 3, \ldots, k_{\max}, \quad \gamma > 0$$

$-\gamma$ is the gradient of the $\log P_k$ versus $\log k$ curve, which is linear. For preferential attachment networks, $-\gamma$ is 3, i.e.

$$P_{k_{\mathrm{PA}}} \propto \frac{1}{k^3}$$

Figure 13 shows the plot of the degree distributions for the two preferential attachment models, along with best fit linear regression lines.

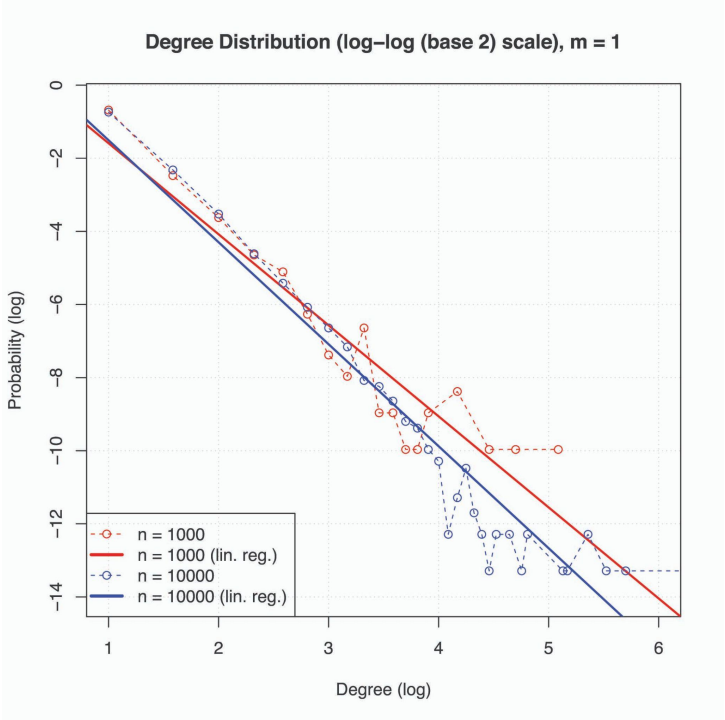


Figure 13: Degree distribution (in log-log scale) for undirected random networks with preferential attachment, n = 1000, 10000 and m = 1.

The negative slope of the linear regression line for n = 1000 is 2.5, and the negative slope of the linear regression line for n = 10000 is 2.8, both of which are approximately equal to 3 as expected. We also observe that the exponent for n = 10000 is closer to 3 than n = 1000. This is because the larger network has more number of

vertices to better approximate the power-law degree distribution for preferential attachment model probabilistically over the smaller network. Since a large network has a higher number of intra-community connections, the probability that the degree k of a randomly chosen node being bigger for large networks over small networks is high as well. Mathematically, the steady-state degree distribution is given as:

$$\lim_{k\to\infty} \propto \frac{1}{k^3}$$

From the above equation, we can see that larger networks follow power-law distribution more closely than smaller ones.

Question 2(e):

In this question, we are asked to plot the degree-distribution on log-log scale of a neighbouring node of a randomly sampled vertex in the two networks generated in Question 2(d), which is shown in Figure 14.

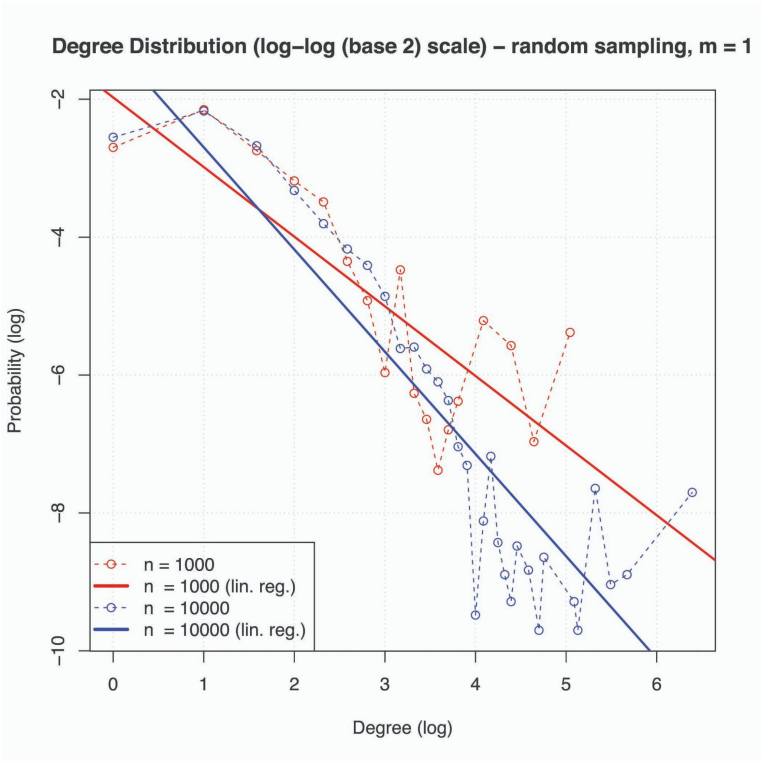


Figure 14: Degree distribution (in log-log scale) for a neighbouring node of a randomly selected node for undirected random networks with preferential attachment, n = 1000, 10000 and m = 1.

From Figure 14, we can see that both the distributions are roughly linear. The negative slope of the linear regression line for n = 1000 is 1.01, and the negative slope of the linear regression line for n = 10000 is 1.5. This is not close to the theoretical value of the exponent in node degree distribution for preferential attachment models, which is 3. Randomly sampling only one neighbouring node does not follow the node degree distribution because:

- At least $\ln(t)$ steps are required in random walk for converging to the node with the highest steady-state degree, where t is the size of the network. A single step is insufficient when t is 1000 or 10000.
- Since we are sampling one node at random, both the expected degree and variance are unbounded, leading to γ being in the range of $1 < \gamma \leq 2$. Since the expected degree is unbounded, the degree distribution will change much more slowly compared to node degree distribution, where the gradient γ is in the range $2 < \gamma \leq 3$ and the expected degree is bounded.
- If we pick a large number of nodes,

$$\lim_{m \to \infty} \mathcal{N}_k = P_k$$

However, if m is small (e.g., 1), then the number of nodes $\mathcal{N}$ picked with degree k does not equal to the probability P that a randomly picked node has degree k. Thus, the relationship between empirical average degree and true average degree of the network breaks down:

$$\left(\mathbb{E}_m(deg) = \frac{1}{m}\sum_{k=1}^{k_{\max}} k \times \mathcal{N}_k\right) \neq \left(\mathbb{E}(deg)\sum_{k=1}^{k_{\max}} k \times P_k\right) \forall (m << \infty)$$

Question 2(f):

In this question, we are asked to show the relationship between the age of nodes and their expected degree for an undirected random network with preferential attachment with timesteps ranging from 1 to 1000. Figure 15 shows the plot.

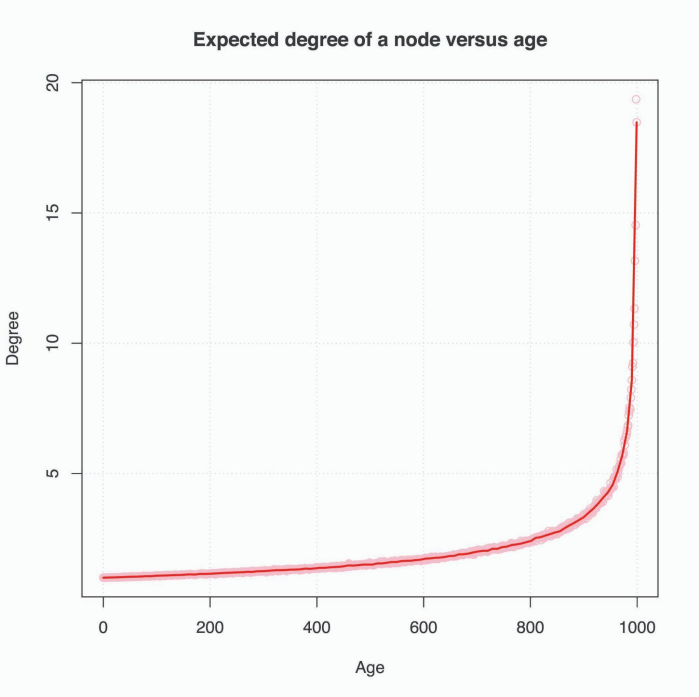


Figure 15: Expected degree versus age of node for undirected random network with preferential attachment, n = 1000 and m = 1.

From Figure 15, we see that the expected degree of a node increases monotonically as it gets older. This is expected because in preferential attachment models, nodes with higher degrees or connectedness are more likely to receive edges with newer nodes. Since older nodes are more likely to be connected with newer nodes with time, the degree of older nodes increase with each timestep. Theoretically, the average degree of ith node (node added at time step i) after time step t is given by:

$$k(i,t) = m\sqrt{\frac{t}{i}}$$

From above equation, we can see that $k(i,t)$ increases monotonically with $(1/i)$. If $i = 1$, then $k_{\max}(t) = m\sqrt{t}$. We see that the maximum expected degree our node has reached is within the maximum limit (for m = 1, $k_{\max}(1000)$ = 31.6, whereas empirical observation is around 20). In addition, if we mirror the plot horizontally and consider the x-axis of the graph as i instead of t, then we can see that $k(i,t)$ will decrease as the value of i increases (older nodes will have higher average degrees while newer nodes will have lower average degrees).

Question 2(g):

In this question, we are asked to repeat Question 2(a) to 2(f) for $m = 2$ and $m = 5$. Figure 16 shows the plot of the four undirected random networks with preferential attachments generated for this question.

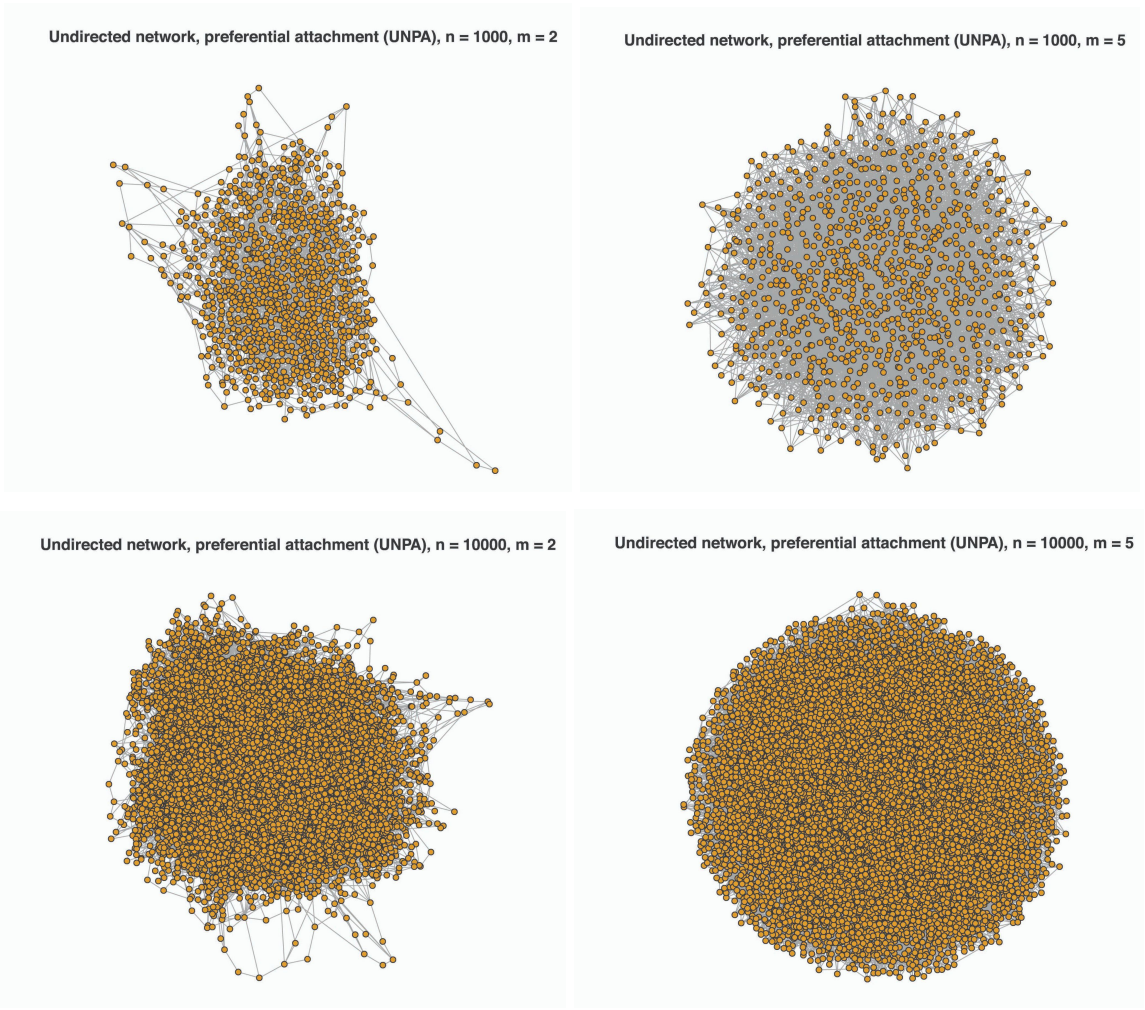


Figure 16: Undirected random network with preferential attachment model, n = 1000 and 10000, m = 2 and 5.

To test whether the networks are always connected or not in practice, we generated 100 random undirected networks with preferential attachment model for each value of m with n = 1000 and checked their connectedness using `is_connected()` function. All 100 networks for each value of m were connected. Since each new node is always attached to a connected node in the existing network, the final network with preferential attachment model is theoretically always connected by construction.

Figure 17 shows the community structures for the four networks. Figure 18 shows the number of nodes in each of the communities for the four networks.

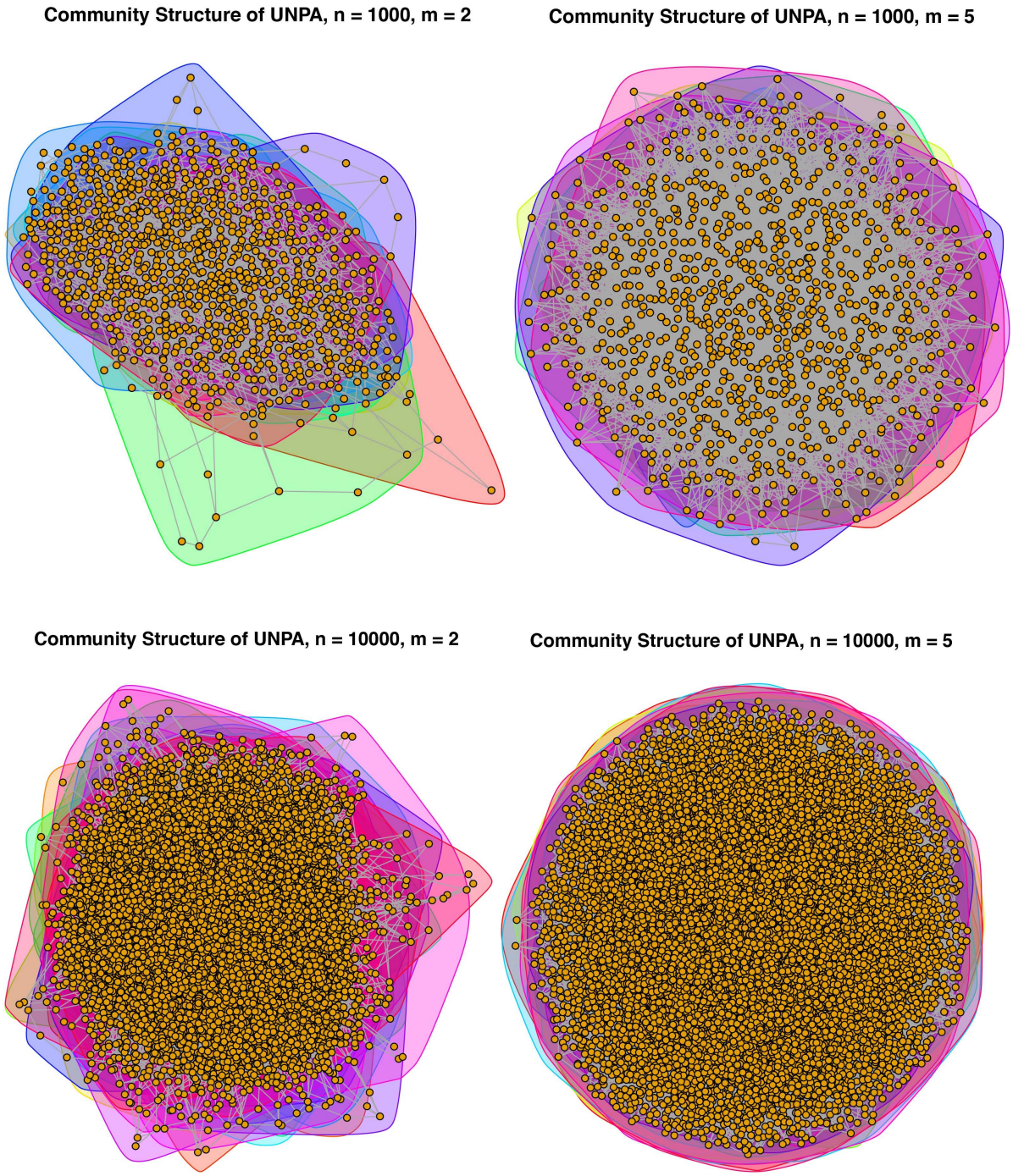


Figure 17: Community structures of undirected networks with preferential attachment, n = 1000 and 10000, m = 2 and 5.

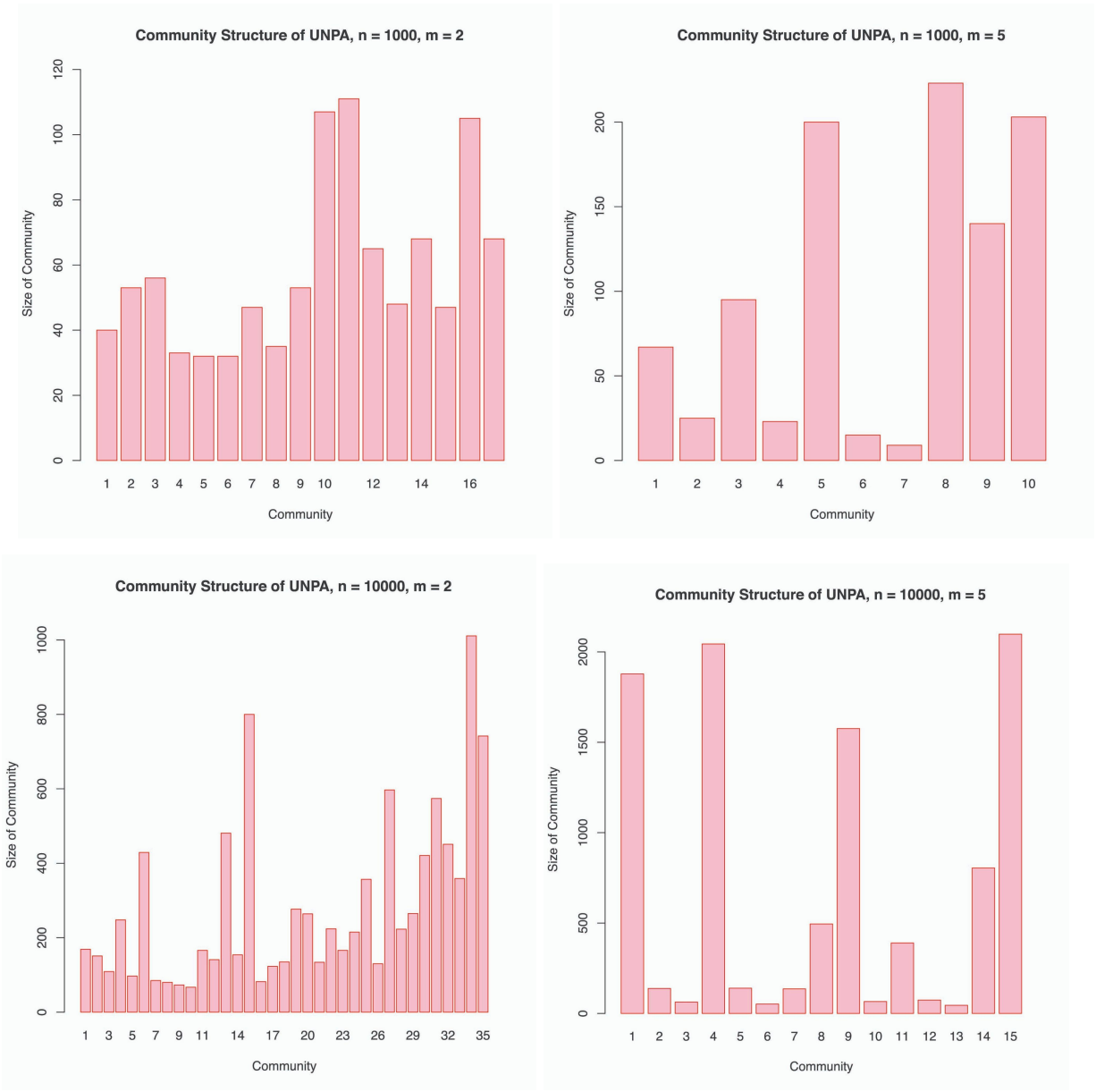


Figure 18: Number of vertices in each community within the undirected random preferential attachment networks, n = 1000 and 10000, m = 2 and 5.

The modularities of the networks are as follows:

- n = 1000, m = 2: 0.520994
- n = 10000, m = 2: 0.533746
- n = 1000, m = 5: 0.277776
- n = 10000, m = 5: 0.271406

For reference, the modularities for m = 1 were as follows:

- n = 1000, m = 1: 0.933042
- n = 10000, m = 1: 0.977433

From Figures 9-12 and 17-18, we make two observations:

- *Effect of* n: We see that there are more communities in the network with higher number of nodes (10000) compared to the smaller network (1000 nodes). In addition, the number of vertices in each community is significantly higher as well. The modularity index is also higher for the larger network over the smaller one, indicating that the larger network is better capable of being partitioned into communities with strong intra-community connectedness and sparse inter-community connectedness. This is expected because with more number of vertices, the average number of edges for nodes with high preference is going to be higher than the expected number of edges for the same high preference nodes in a smaller network. The probability of strong intra-community connections is greater for large networks over small networks, and likewise, the extent of sparsity among communities are amplified in large networks, as newer nodes are more likely to be paired with high preference nodes having a larger degree than the high preference nodes in a small network.

- *Effect of* m: We observe that as the value of m increases, the modularity index drops and the number of communities drop significantly, with a few communities having a large number of vertices. Intuitively, a higher value of m indicates that an incoming node is connected to a larger number of older nodes. While this should result in strong intra-community connectedness, the global sparsity among different communities is lost due to the connectedness requirement brought on by high values of m, resulting in edges being formed among otherwise distinct clusters and hence weakening the community structures. Mathematically,

  $$Q(P) = \sum \frac{1}{2m} \sum_{i,j \in C_i} \left(A_{ij} - \frac{k_i k_j}{2m} \right)$$

  If the number of edges in the network m (not to be confused with the actual m we are talking about, which is the number of old nodes the incoming node connects with) increases, then $Q(P)$ dtops. As a result, less clusters are formed in the overall graph.

Figure 19 shows the plot of the degree distributions for the four preferential attachment models, along with best fit linear regression lines. The negative slopes of the linear regression lines are as follows:

- n = 1000, m = 2: 2.1246
- n = 10000, m = 2: 2.518
- n = 1000, m = 5: 2.095
- n = 10000, m = 5: 2.238

For reference, the negative slopes of the linear regression lines for $m = 1$ were as follows:

- n = 1000, m = 1: 2.5
- n = 10000, m = 1: 2.8

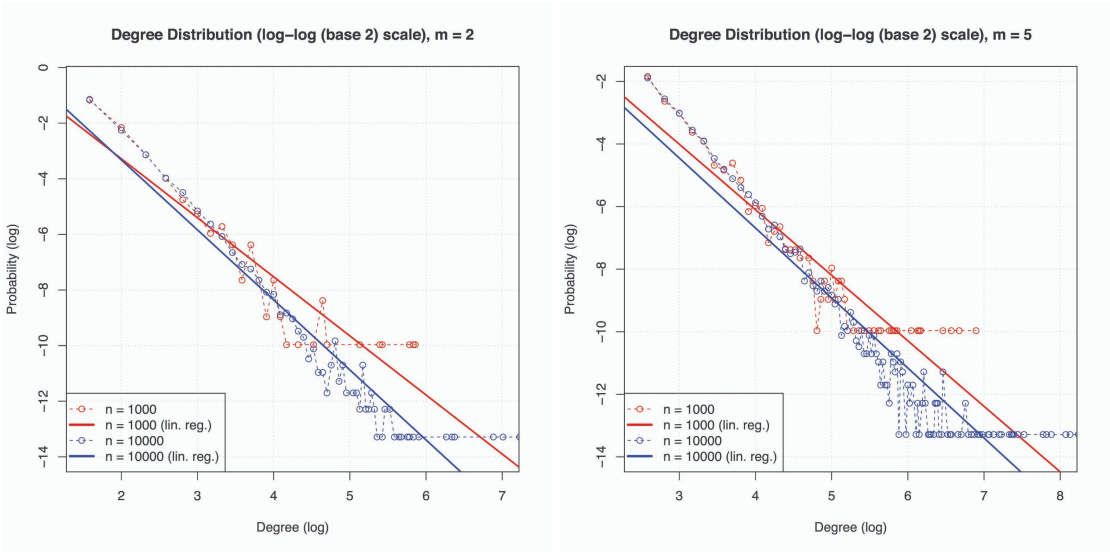


Figure 19: Degree distribution (in log-log scale) for undirected random networks with preferential attachment, n = 1000 and 10000, m = 2 and 5.

From Figure 13 and 19, we make three observations:

- The trends are roughly linear in the log-log scale.
- *Effect of* n: Ideally, the power-law exponent (negative gradient of log-log degree distribution plot) γ of an undirected random network with preferential attachment should be 3. We observe that the power-law exponent for n = 10000 is closer to 3 than n = 1000 for all three values of m. This is because larger networks have more number of vertices to better approximate the power-law degree distribution for preferential attachment model probabilistically over the smaller networks. Since a large network has a higher number of intra-community connections, the probability that the degree k of a randomly chosen node being bigger for large networks over small networks is high as well. Mathematically, the steady-state degree distribution is given as:

$$\lim_{k\to\infty} \propto \frac{1}{k^3}$$

  From the above equation, we can see that larger networks follow power-law distribution more closely than smaller ones.
- *Effect of* m: We observe that as m increases, the negative slope of the linear regression line decreases. This means that the degree distribution deviates from the preferential attachment model when incoming nodes are added to more number of older vertices. One way to explain this is that as m increases, the expected degree of a node for the same degree increases. Mathematically, for preferential attachment networks:

$$\text{Average degree} = 2m$$

Since expected degree is now closer to ∞ with higher values of m (unbounded degree distribution), the value of γ drops and moves in the $1 < \gamma \leq 2$ region from $2 < \gamma \leq 3$ region, leading to a dense network with a significant number of high degree nodes.

The degree-distribution on log-log scale of a neighbouring node of a randomly sampled vertex for preferential attachment models with $m = 2$ and $m = 5$ are shown in Figure 20.

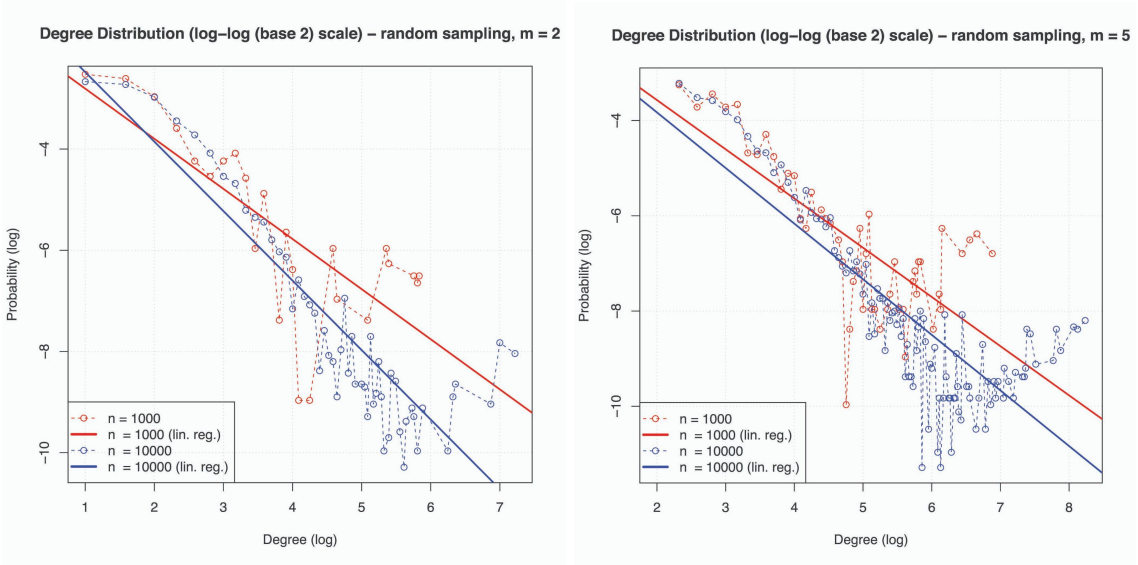


Figure 20: Degree distribution (in log-log scale) for a neighbouring node of a randomly selected node for undirected random networks with preferential attachment, n = 1000 and 10000, m = 2 and 5.

The negative slopes of the linear regression lines are as follows:

- n = 1000, m = 2: 0.9904
- n = 10000, m = 2: 1.373
- n = 1000, m = 5: 1.033
- n = 10000, m = 5: 1.168

For reference, the negative slopes of the linear regression lines for $m = 1$ were as follows:

- n = 1000, m = 1: 1.01
- n = 10000, m = 1: 1.5

From Figures 14 and 20, we can make several observations:

- The distributions are roughly linear.
- The value of γ is not close to the theoretical value of the exponent in node degree distribution for preferential attachment models, which is 3. Randomly sampling only one neighbouring node does not follow the node degree distribution because:

- At least $\ln(t)$ steps are required in random walk for converging to the node with the highest steady-state degree, where t is the size of the network. A single step is insufficient when t is 1000 or 10000.
- Since we are sampling one node at random, both the expected degree and variance are unbounded, leading to γ being in the range of $1 < \gamma \leq 2$. Since the expected degree is unbounded, the degree distribution will change much more slowly compared to node degree distribution, where the gradient γ is in the range $2 < \gamma \leq 3$ and the expected degree is bounded.
- If we pick a large number of nodes,

  $$\lim_{m \to \infty} \mathcal{N}_k = P_k$$

  However, if m is small (e.g., 1), then the number of nodes $\mathcal{N}$ picked with degree k does not equal to the probability P that a randomly picked node has degree k. Thus, the relationship between empirical average degree and true average degree of the network breaks down:

  $$\left(\mathbb{E}_m(deg) = \frac{1}{m} \sum_{k=1}^{k_{\max}} k \times \mathcal{N}_k \right) \neq \left(\mathbb{E}(deg) \sum_{k=1}^{k_{\max}} k \times P_k \right) \forall (m << \infty)$$

- Effect of m: We observe that as m increases, the negative slope of the linear regression line decreases. This means that the degree distribution deviates from the preferential attachment model when incoming nodes are added to more number of older vertices. The explanation is the similar as for the overall degree distribution. As m increases, the expected degree of a randomly selected node for the same degree increases. Mathematically, for preferential attachment networks:

  $$\text{Average degree} = 2m$$

  Since expected degree is now closer to ∞ with higher values of m (unbounded degree distribution), the value of γ drops.

The relationship between the age of nodes and their expected degree for undirected random networks with preferential attachment with timesteps ranging from 1 to 1000 with $m = 2$ and $m = 5$ are shown in Figure 21. From Figures 15 and 21, we can make several observations:

- For all values of m, we see that the expected degree of a node increases monotonically as it gets older. This is expected because in preferential attachment models, nodes with higher degrees or connectedness are more likely to receive edges with newer nodes. Since older nodes are more likely to be connected with newer nodes with time, the degree of older nodes increase with each timestep. Theoretically, the average degree of ith node (node added at time step i) after time step t is given by:

  $$k(i,t) = m\sqrt{\frac{t}{i}}$$

From above equation, we can see that $k(i,t)$ increases monotonically with $(1/i)$. If $i = 1$, then $k_{\max}(t) = m\sqrt{t}$. In addition, if we mirror the plot horizontally and consider the x-axis of the graph as i instead of t, then we can see that $k(i,t)$ will decrease as the value of i increases (older nodes will have higher average degrees while newer nodes will have lower average degrees).

- Effect of m: As m increases, the expected degree for the same node for same age increases. This is because:

$$\text{Average degree} = 2m$$

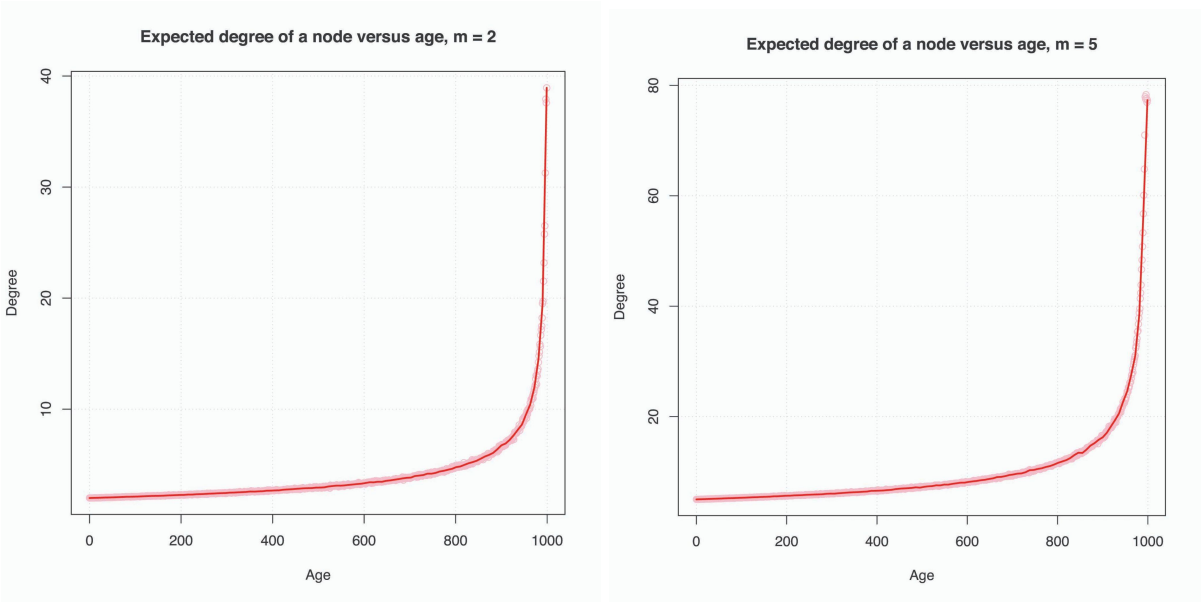


Figure 21: Expected degree versus age of node for undirected random networks with preferential attachment, n = 1000,, m = 2 and 5.

Question 2(h):

In this question, we are asked to generate a preferential attachment network, take its degree sequence and generate a new network with the same degree sequence using stub-matching procedure via the `sample_degseq()` function, along with finding out the modularities and community structures. We use three modes to generate the graph via stub-matching, namely 'simple', 'simple, no multiple' and 'Viger-Latapy". Figure 22 shows the plots of the generated random networks with preferential attachments generated for this question.

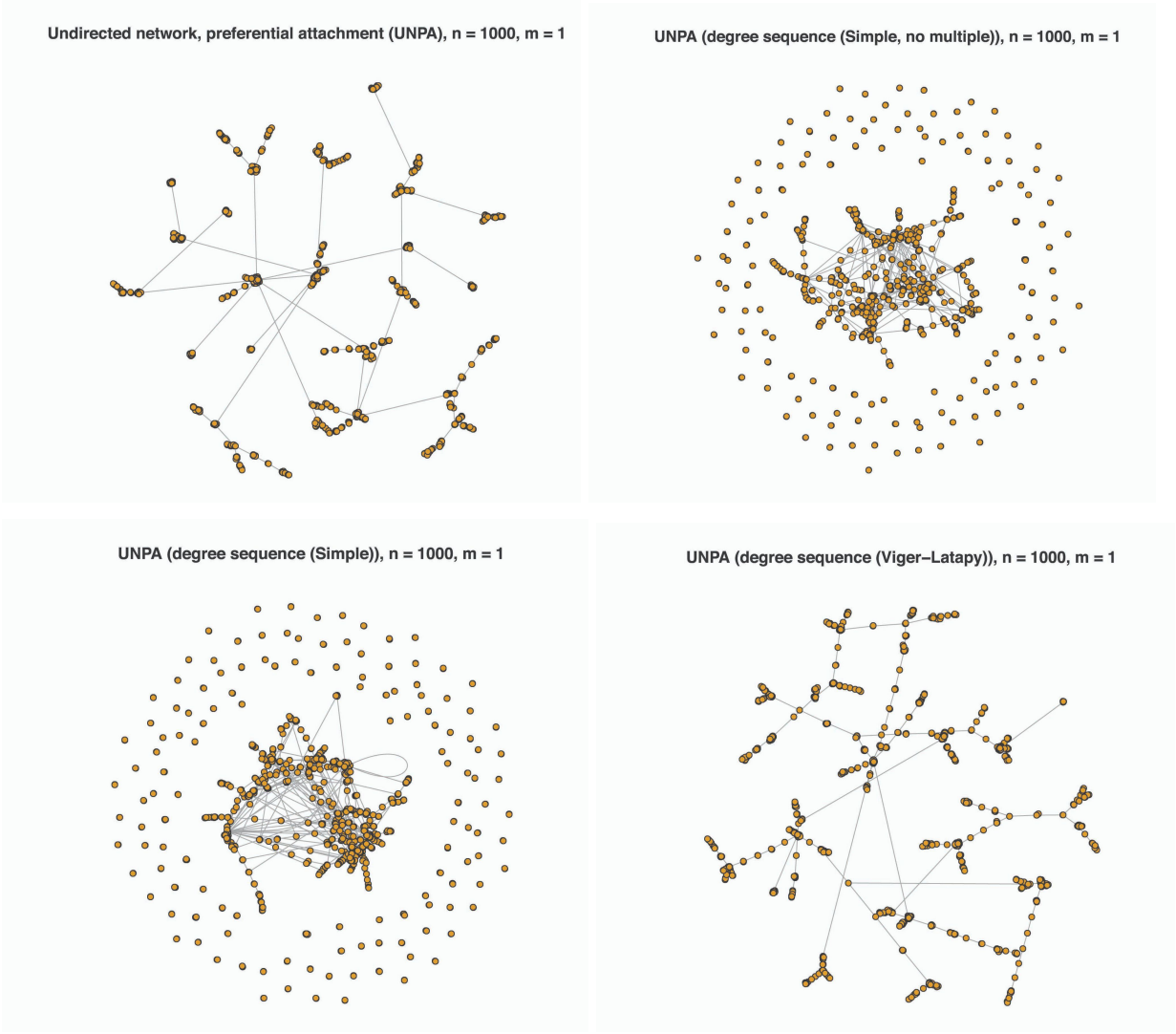


Figure 22: Undirected random network with preferential attachment model and generated networks from the degree sequence of the original network using random-stub matching for various stub-matching methods.

Figure 23 shows the community structures for all the networks. Figure 24 shows the number of nodes in each of the communities for all the networks.

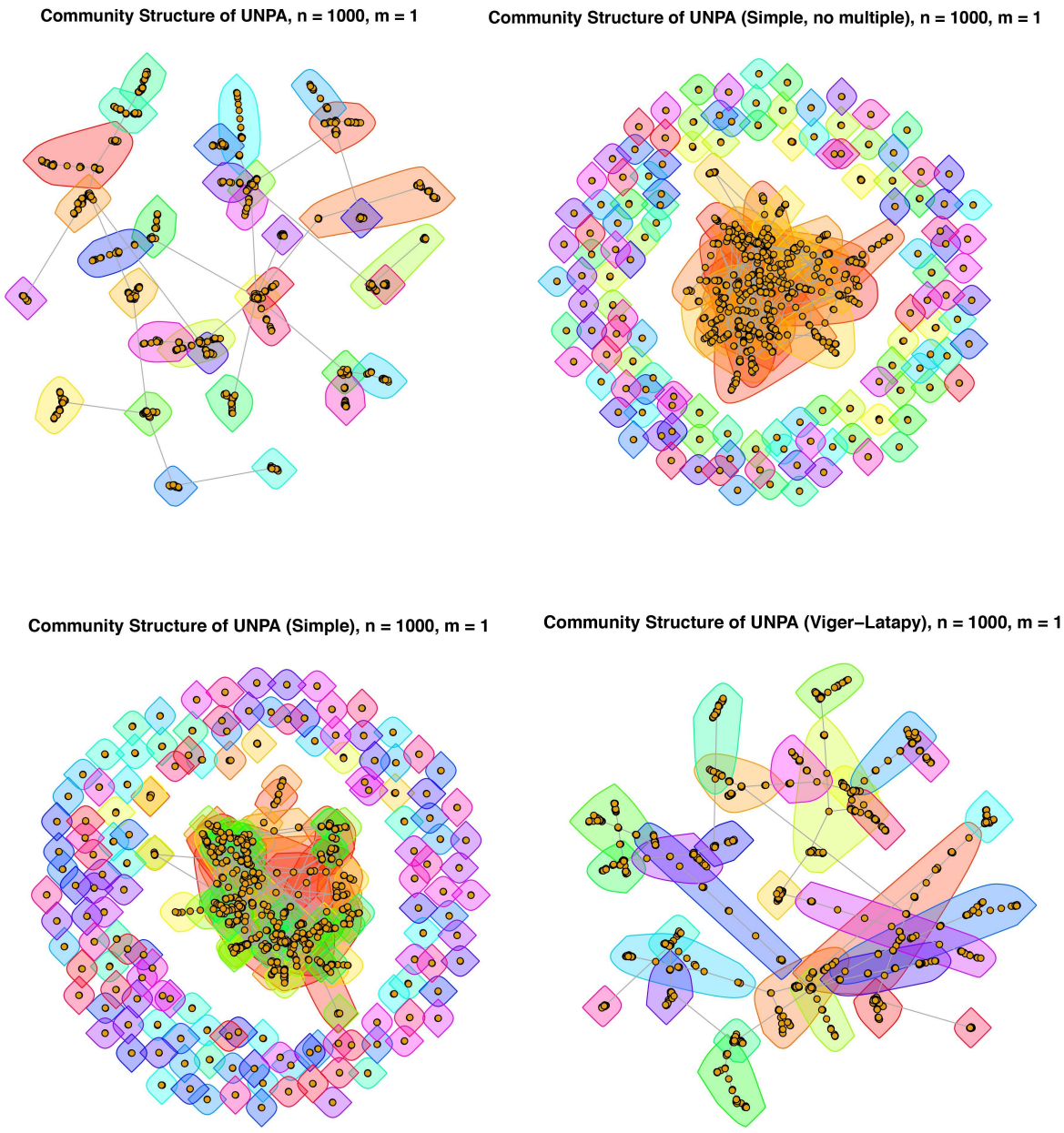


Figure 23: Community structures of undirected network with preferential attachment and generated networks from the degree sequence of the original network using random-stub matching for various stub-matching methods.

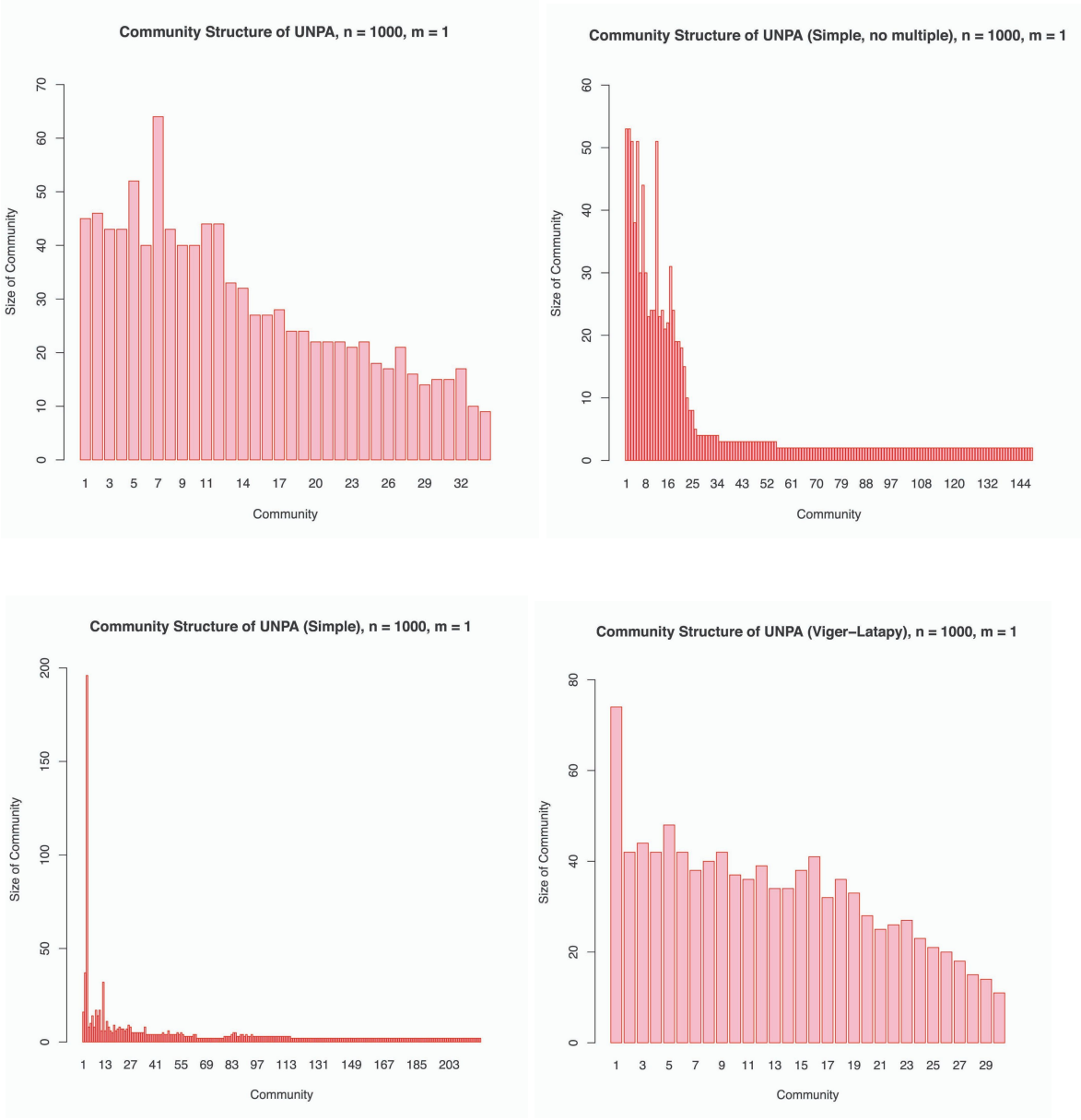


Figure 24: Number of vertices in each community within the undirected random preferential attachment network and generated networks from the degree sequence of the original network using random-stub matching for various stub-matching methods.

The modularity of the four networks are as follows:

- Undirected random preferential attachment network: 0.930920
- Stub-matching from degree sequence (simple): 0.737921
- Stub-matching from degree sequence (simple, no multiple): 0.834776
- Stub-matching from degree sequence (Viger-Latapy): 0.932926

The three methods for performing stub-matching are described below:

- The “simple” method connects the stubs without considering the formation of loops. This technique can result in generation of loop edges or multiple edges from a given degree sequence. Fast-greedy community detection may fail to operate on graphs generated through this method and require walk-trap community detection.
- The “simple, no multiple” method avoids multiple and loop edges and restarts the generation from scratch if loops are detected. Fast-greedy community detection works on this method, but the method does not guarantee uniform sampling from the space of all possible graphs.
- The “Viger-Latapy” method is a specialized generator which creates a simple undirected graph (may be unconnected) from the given degree sequence with some rewiring and Monte-Carlo sampling to form randomized connected graphs.

Since Viger-Latapy is a specialized form of stub-matching, we ignore it for future comparisons with the preferential attachment model, and instead consider the “simple” and “simple, no multiple” as vanilla stub-matching methods.

In stub-matching from degree sequences, we randomly sample n numbers from the degree distribution of our choice, e.g. power law degree distribution to get the nodes of the network along with their degrees. The value of the ith sample is the degree of the ith node. Afterwards, we randomly match the stubs to connect the vertices of the network by edges. From Figure 23 and 24, we observe that a significant number of vertices for the network generated via stub-matching are unconnected and forming their own communities, while the network generated via preferential attachment model show local regions of high-connectedness with sparse connections among individual communities. The number of communities are also significantly higher for stub-matching over preferential attachment. This is reflected in the modularity indices of the networks, with preferential attachment models having a higher modularity than stub-matching models, indicating that the preferential attachment network is better capable of being partitioned into communities with strong intra-community connectedness and sparse inter-community connectedness over stub-matching networks. In addition, since each new node is always attached to a connected node in the existing network for preferential attachment models, the final network is always guaranteed to be connected by construction. However, this is not the case for stub-matching from degree sequence, as the process is random and does not necessarily require connection to older nodes. Furthermore, stub-matching can lead to the formation of loops, which can make community detection difficult using greedy procedures. The only advantage of stub-matching is the level of control over the degree sequence that the user has, allowing the user to generate networks with custom degree distributions.

Question 3(a):

In this question, we are asked to plot the degree distributions in log-log scale and estimate the slope of the plots using linear regression to find the power-law exponent for a modified preferential attachment model that penalizes the age of a node. Figure 25 shows the generated network and Figure 26 shows the degree distribution plot for the network in question.

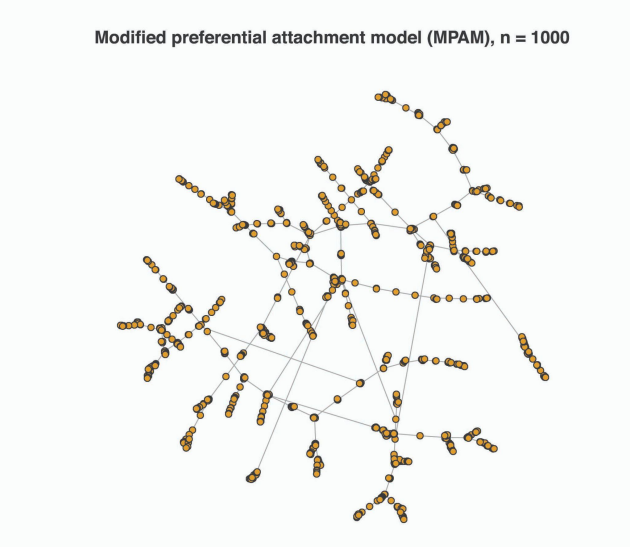


Figure 25: Undirected random network with modified preferential attachment model, n = 1000

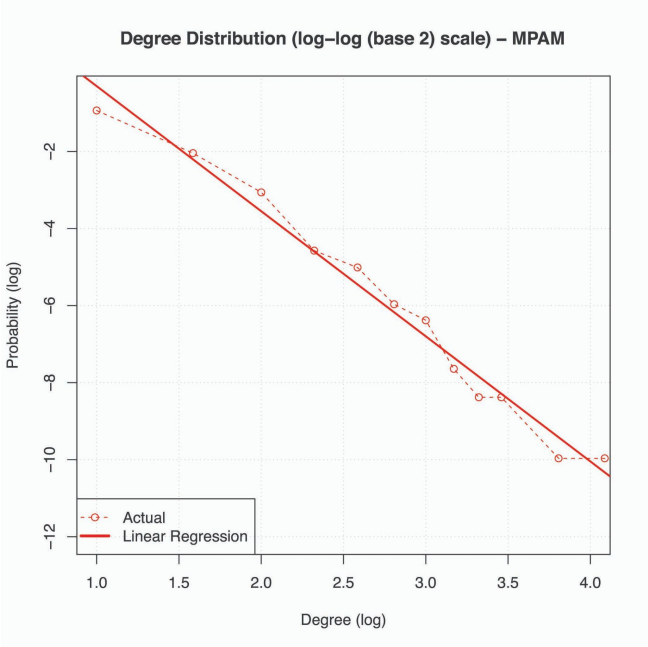


Figure 26: Degree distribution (in log-log scale) for undirected random network with modified preferential attachment, n = 1000.

The negative slope of the linear regression line for n = 1000 is 3.248, hence the power law exponent is 3.248. This is approximately equal to 3 as expected for preferential attachment models. We observe that the degree distribution is much more linear than vanilla preferential attachment models in the log-log scale, with a lower value on the upper bound of the expected degree of the node. This is observed from the empirical value of the power-law exponent, which is > 3, indicating both bounded variance and bounded mean. This means that the modified preferential attachment model gives rise to a sparse network (average degree is bounded) with a finite number of high-degree nodes (bounded variance) and a degree-distribution tail that is not as heavy as vanilla preferential attachment models.

Question 3(b):

In this question, we are asked to find the community structure and the modularity of the network generated in Question 3(a). Figure 27 shows the community structure and Figure 28 shows the number of nodes in each of the communities.

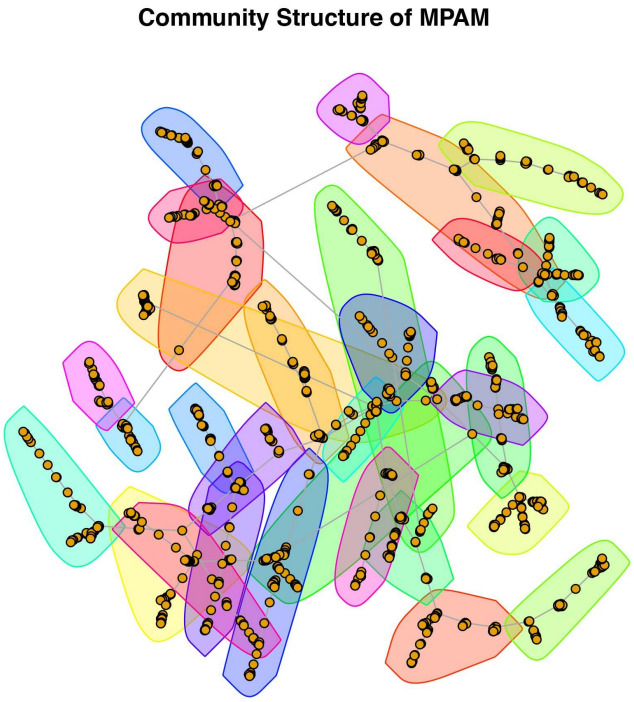


Figure 27: Community structure of the undirected network with modified preferential attachment, n = 1000.

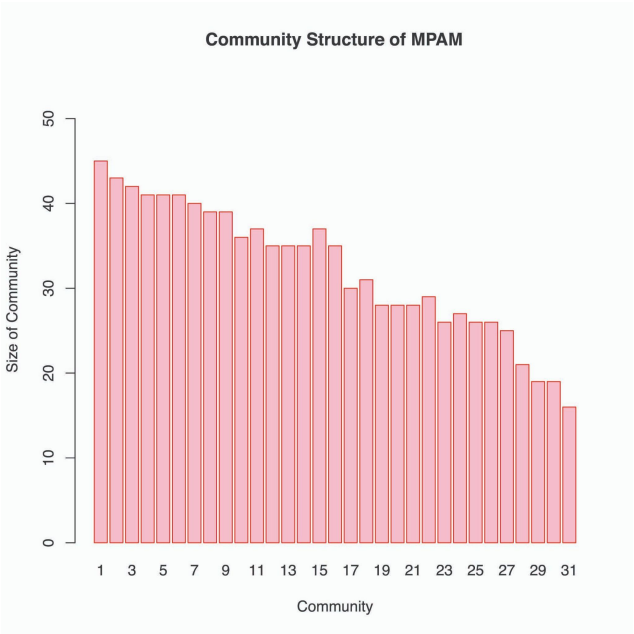


Figure 28: Number of vertices in each community within the undirected network with modified preferential attachment network, n = 10000

The modularity of the network is 0.935772, which is similar to that of the vanilla preferential attachment networks. From Figures 10 and 28, we see that the number of vertices in each community are much more homogeneous for the modified model over vanilla preferential attachment networks. This is expected as the modified model penalizes the network generator from selecting nodes that are too old and encourages attachment of incoming nodes to moderately aged nodes, yielding clusters with homogenous number of nodes.

RANDOM WALK ON NETWORKS

Question 1(a):

In this question, we are asked to create an undirected random network (Erdos-Renyi network) with 1000 nodes, and the probability p for drawing an edge between any pair of nodes equal to 0.01. Figure 29 shows the generated network, created using `erdos.renyi.game()` function in `igraph` library in R.

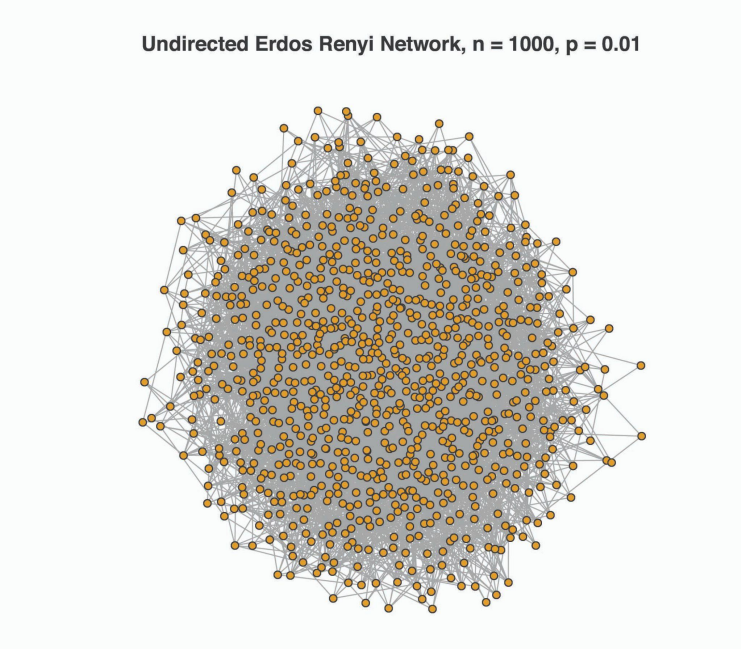


Figure 29: Undirected Erdos-Renyi network, n = 1000, p = 0.01.

Question 1(b):

In this question, we are asked to plot the average distance (shortest path length), $< s(t) >$ of a random walker from his starting point at step t versus t, and the variance $< \sigma^2(t) >$ of this distance versus t. Figure 30 shows the plots. We walked 100 steps on the giant connected component of a random graph if the graph is not connected, else we walked on the entire graph. We performed the random walk on 1000 random realizations.

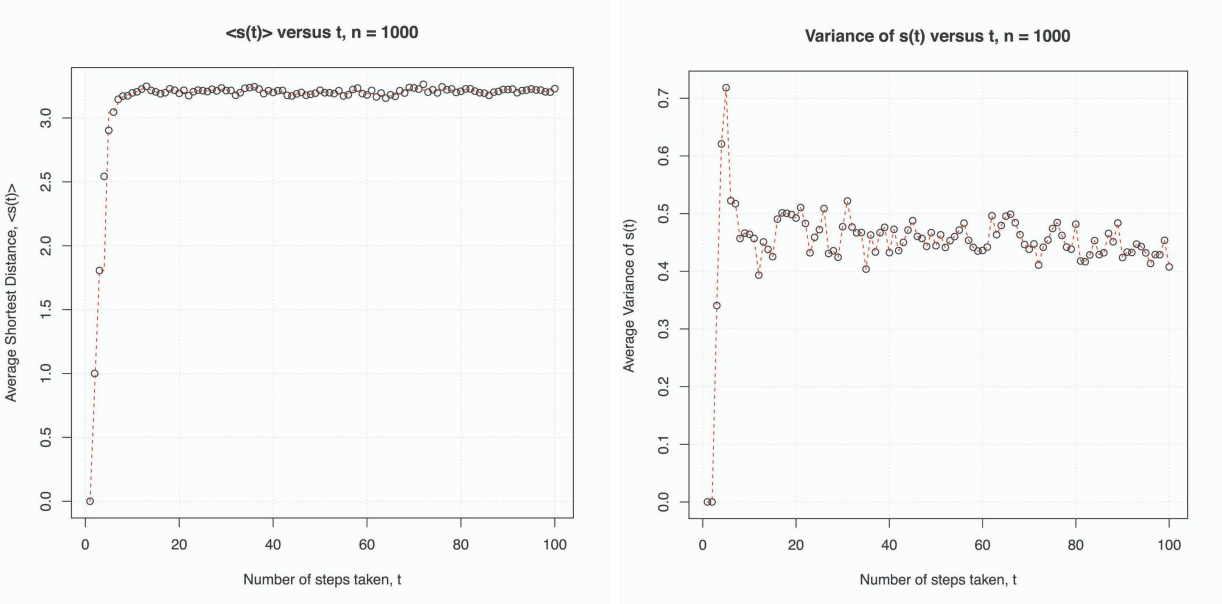


Figure 30: (Left) Plot of $< s(t) >$ versus t, (Right) Plot of $< \sigma^2(t) >$ versus t for an undirected Erdos-Renyi network with 1000 nodes and p = 0.01.

From Figure 30, we see that the average distance of the walker increases initially and then converges to a steady-state value of $\approx$ 3.2 when $t \approx 10$. The variance rises initially and then falls to a steady-state value of $\approx$ 0.42 at around the same time step of 10. This is expected, because for any non-bipartite graph, doing a random walk for a sufficiently large number of steps provides the steady state node occupancy probability vector π. Mathematically:

$$(P^T)^k \pi(o) \approx \pi, \quad \pi = \frac{1}{\mathbf{1}^T A \mathbf{1}} A\mathbf{1}$$

Since there is no preferential attachment, the walker is likely to end on a random node within the circular region corresponding to the average shortest distance of the nodes in the entire network from a random initial node.

The diameter of the network is 5.

Question 1(c):

The degree distributions for the graph and the terminal nodes are shown in Figure 31.

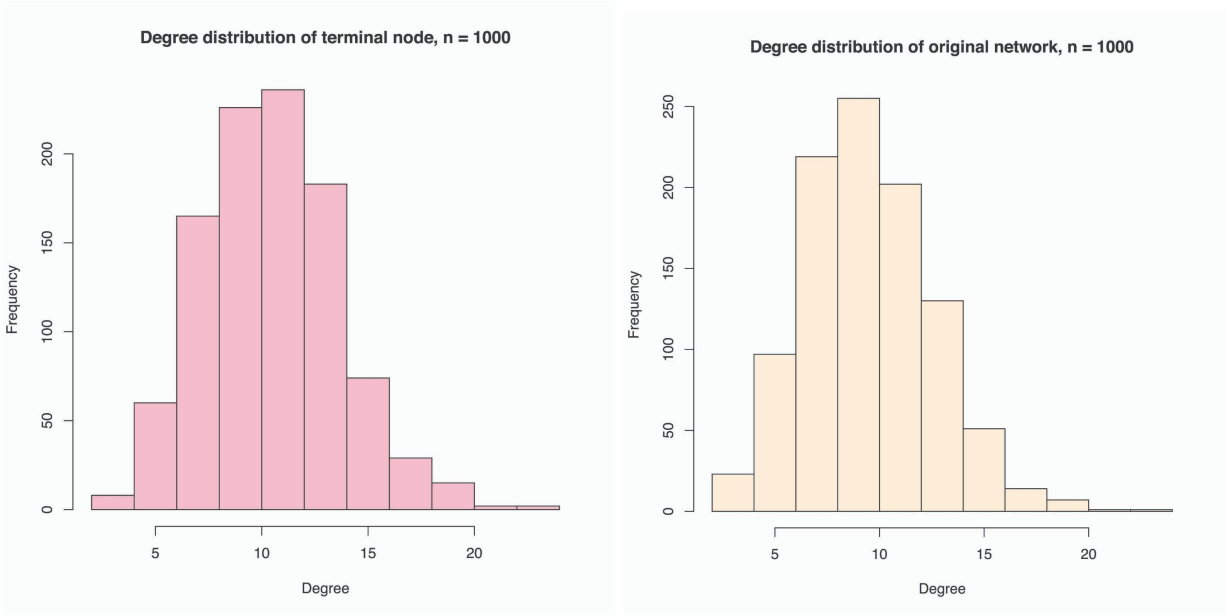


Figure 31: Degree distribution of (Left) terminal node after random walk (Right) entire graph.

From Figure 31, we observe that the degree distribution of the terminal node follows the degree distribution of the entire network (Binomial for Erdos-Renyi networks) with similar mean and variance.

Question 1(d):

In this question, we are asked to repeat Question 1(b) for undirected random networks with 10000 nodes. Figure 32 shows the plots. We walked 100 steps on the giant connected component of a random graph if the graph is not connected, else we walked on the entire graph. We performed the random walk on 1000 random realizations.

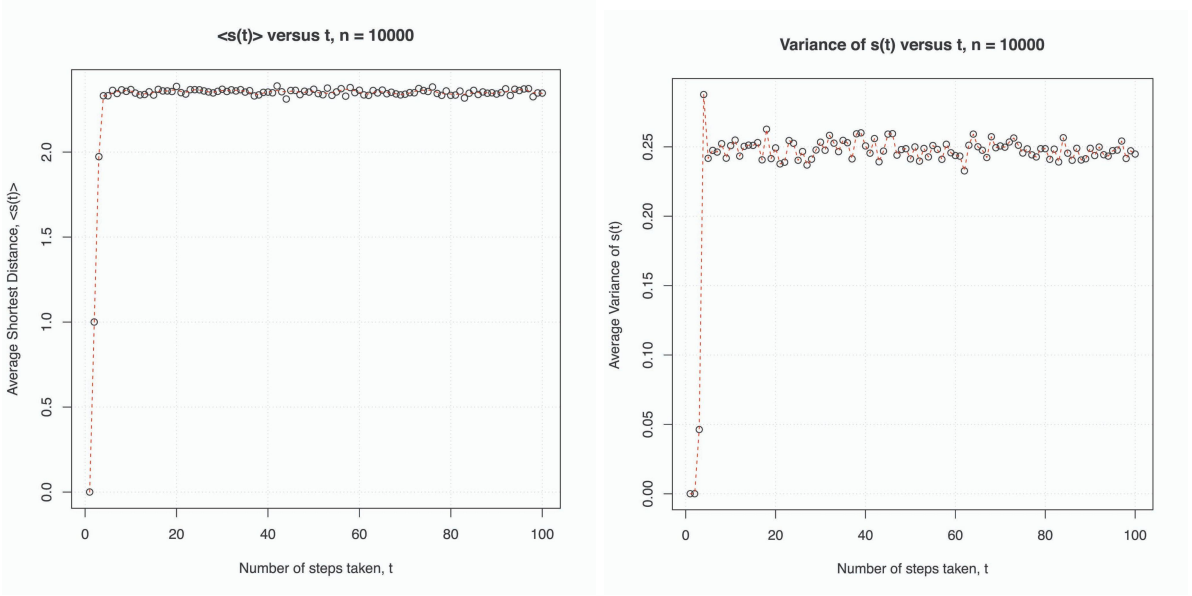


Figure 32: (Left) Plot of $< s(t) >$ versus t, (Right) Plot of $< \sigma^2(t) >$ versus t for an undirected Erdos-Renyi network with 10000 nodes and p = 0.01.

The diameter of this network is 3, compared to 5 for the network with 1000 nodes. We can make several observations from the plots in Figure 30 and 32. Since the diameter of the larger network (3) is shorter over the smaller network (5), the steady-state average shortest distance is also smaller (2.4 versus 3.2) for the larger network. In addition, the number of steps required for convergence is smaller for the larger network over the smaller network (5 steps versus 10). The counter-intuitive value of diameters can be explained by the fact that a larger network has a high-degree of connectedness with nodes tightly packed close to each other. Thus, the random walker overall covers a smaller distance for the same number of steps within a large network compared to a smaller one, where the nodes are more spread out. The steady-state variance is lower for the larger network over the smaller network (0.25 versus 0.42), because the average distances between nodes in the larger network are more homogenous over the smaller network.

Question 2(a):

In this question, we are to create an undirected random network with 1000 vertices and preferential attachment model and $m = 1$. The network is shown in Figure 33.

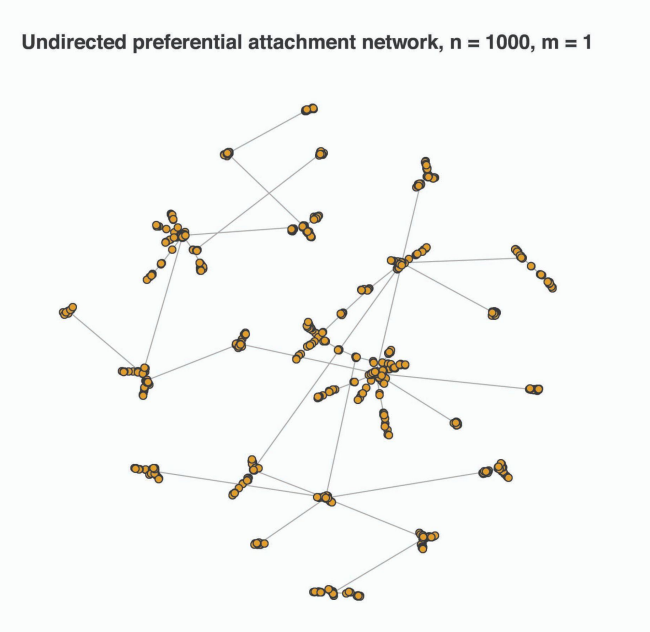


Figure 33: Undirected random network with preferential attachment model, $n = 1000$, $m = 1$.

Question 2(b):

In this question, we are asked to plot the average distance (shortest path length), $< s(t) >$ of a random walker from his starting point at step t versus t, and the variance $< \sigma^2(t) >$ of this distance versus t. Figure 34 shows the plots. We walked 1000 steps on the graph. We performed the random walk on 1000 random realizations.

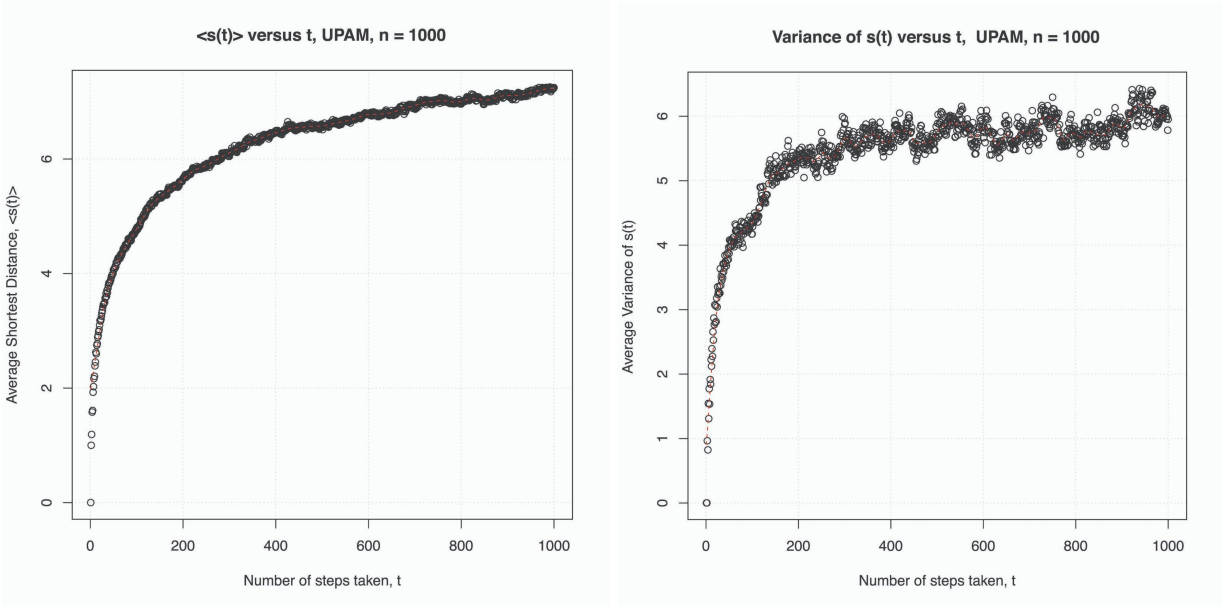


Figure 35: (Left) Plot of $< s(t) >$ versus t, (Right) Plot of $< \sigma^2(t) >$ versus t for an undirected network with preferential attachment with 1000 nodes and m = 1.

From Figure 35, we see that the average distance of the walker as well as the variance increases non-monotonically with the number of steps taken, with the rate of change of rise decreasing with more number of steps. Compared to Erdos-Renyi network, the shortest distance does not converge to a steady-state distance, but follows the following relation:

$$< s(t) >= \mathcal{O}(\sqrt{t})$$

This is because unlike Erdos-Renyi networks, Barabasi-Albert networks are preferentially attached. Doing a random walk for a sufficiently large number of steps on scale-free networks gives the steady state occupancy vector, which is used to pick the nodes in a preferential manner. Nodes with higher degree have a higher probability of getting selected as the terminal node. As a result, the implicit goal of the walker here is to not end up at a random node, but at a node with a very high degree. With more number of steps, the walker gets further away from the initial node as it reaches nodes possibly in other communities but with a higher degree than the nodes within the community the walker started in. On the other hand, in Erdos-Renyi networks, the walker usually stays within a bounded circle of the initial node corresponding to the average shortest path of the entire network, as there is no preferential attachment. The diameter of the network is 18.

Question 2(c):

The degree distributions for the graph and the terminal nodes are shown in Figure 36.

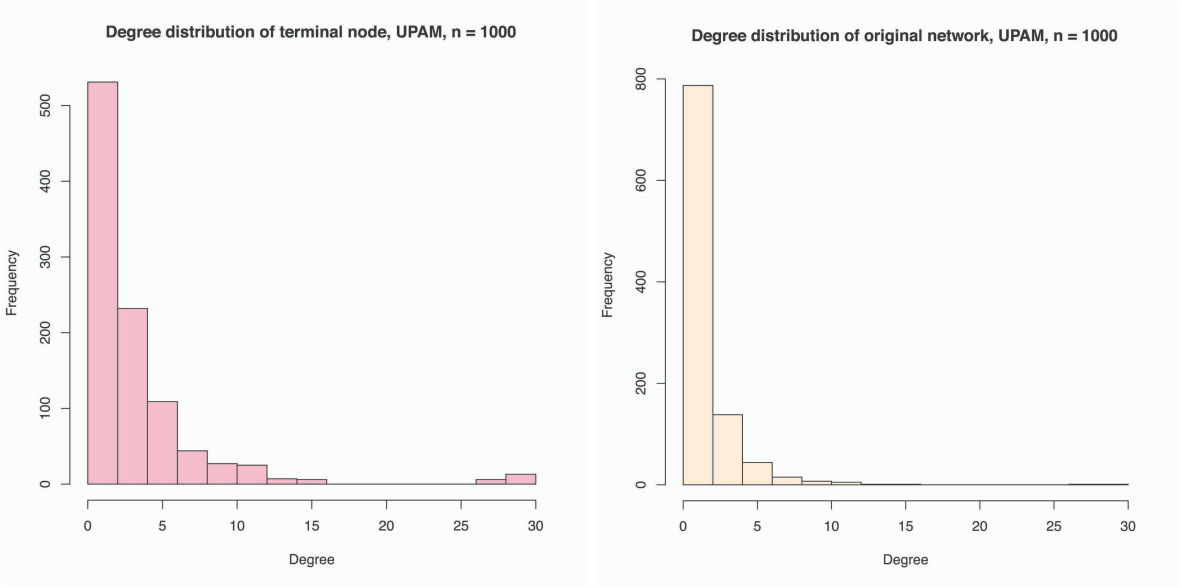


Figure 36: Degree distribution of (Left) terminal node after random walk (Right) entire graph.

From Figure 36, we observe that the degree distribution of the terminal node follows the degree distribution of the entire network (fat-tailed power-law distribution):

$$P_k \propto \frac{1}{k^\gamma} = \frac{1}{k^\gamma \sum_{k=1}^{k_{\max}} k^{-\gamma}}, \quad k \in 1, 2, 3, ..., k_{\max}, \quad \gamma > 0$$

This is expected because for preferential attachment models, doing a random walk for a sufficiently large number of steps will give the steady state occupancy vector, which can be used to pick the nodes in a preferential manner. Nodes with higher degree have a higher probability of getting selected as the terminal node.

Question 2(d):

In this question, we are asked to repeat Question 2(b) for Barabasi-Albert networks with 100 and 10000 nodes. Figure 37 and 38 shows the plots. We performed the random walks on 1000 random realizations of each network.

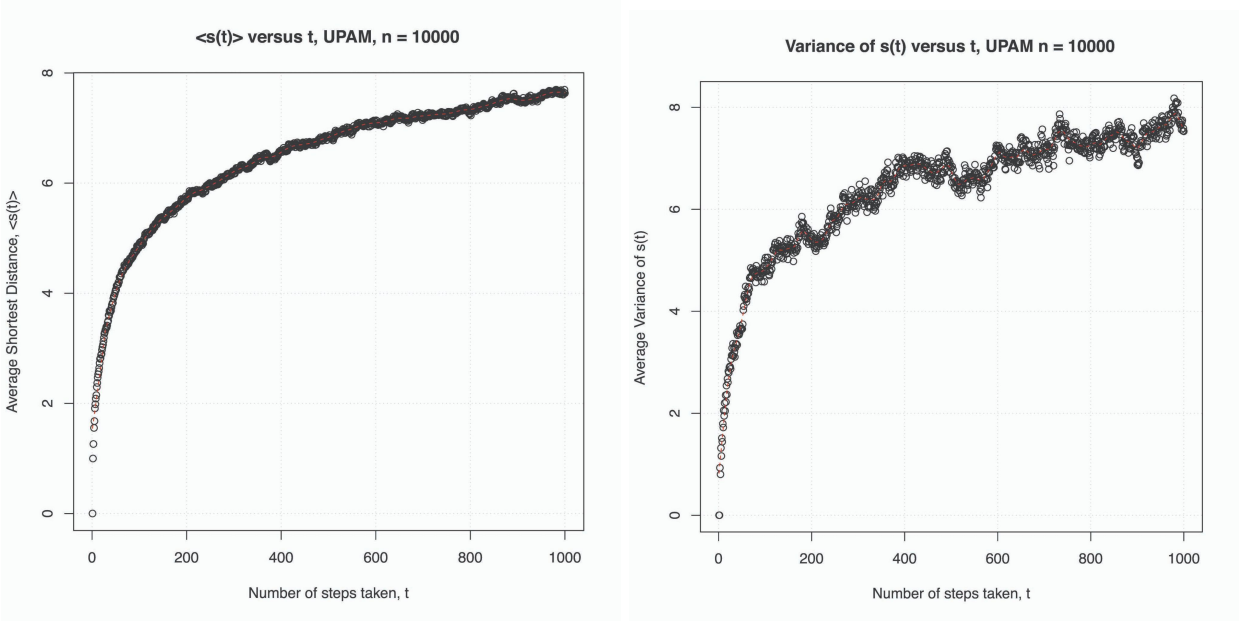


Figure 37: (Left) Plot of $< s(t) >$ versus t, (Right) Plot of $< \sigma^2(t) >$ versus t for an undirected network with preferential attachment with 10000 nodes and m = 1.

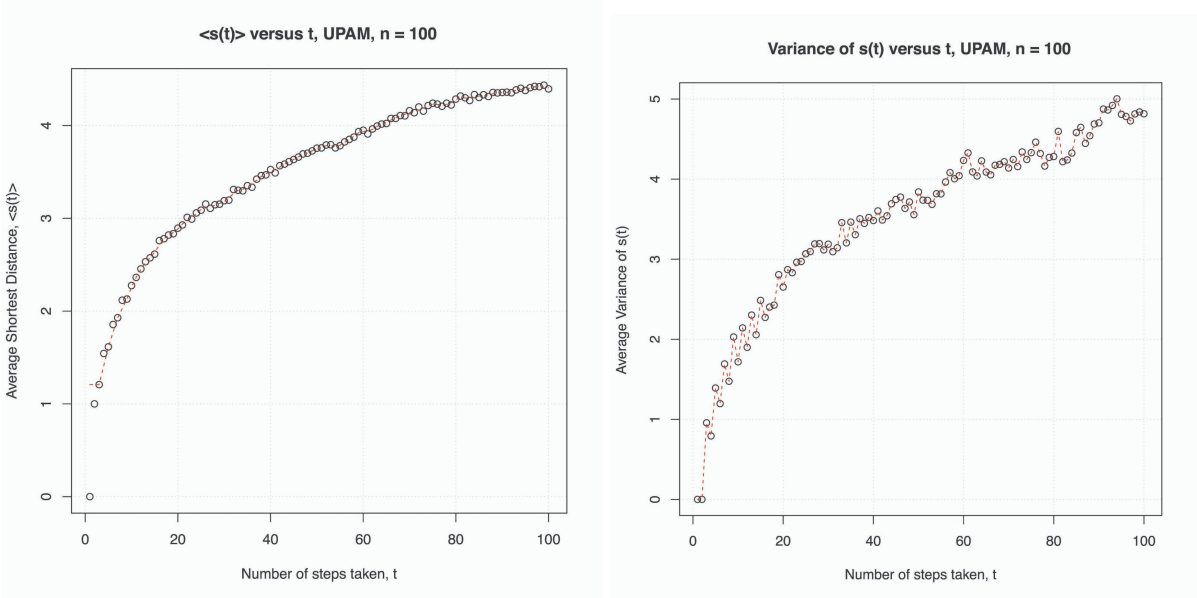


Figure 38: (Left) Plot of $< s(t) >$ versus t, (Right) Plot of $< \sigma^2(t) >$ versus t for an undirected network with preferential attachment with 100 nodes and m = 1.

The diameter of the preferential attachment networks are as follows:

- 100 nodes: 13
- 1000 nodes: 18 (from Question 2(b))
- 10000 nodes: 29

We can make several observations from the plots in Figure 35, 37 and 38: The average path length (as well as the variance) is higher for the larger networks (more number of vertices and larger diameter). This is because the average path length scales logarithmically with the size of the network:

$$< s(t) >= \mathcal{O}\left(\frac{\ln n}{\ln \ln n}\right)$$

Unlike Erdos-Renyi networks, the diameter of the network does not decrease with increasing number of nodes but rather increases. This is because larger networks are better capable of being partitioned into communities with strong intra-community connectedness and sparse inter-community connectedness. This is expected because with more number of vertices, the average number of edges for nodes with high preference is going to be higher than the expected number of edges for the same high preference nodes in a smaller network. The probability of strong intra-community connections is greater for large networks over small networks, and likewise, the extent of sparsity among communities are amplified in large networks, as newer nodes are more likely to be paired with high preference nodes having a larger degree than the high preference nodes in a small network. Since there are more communities, random walkers on larger networks are more likely to end up at a high-degree node that is far away from the initial community compared to small networks. As mentioned earlier, the implicit goal of the walker in preferential attachment models is to not end up at a random node that is closest to the initial node, but at a node with a very high degree. With more number of nodes, the uncertainty in the shortest path between terminal and initial nodes is also higher, resulting in a higher variance of shortest path values for larger networks and requiring more number of steps to reach a stable terminal node.

Question 3(a):

In this question, we are asked to use random walk to simulate PageRank without teleportation. The PageRank algorithm is used by the Google search engine and exploits the graphical structure of the web to compute global importance scores (called pagerank scores) that can be used to influence the ranking of search results.

We create two directed preferential attachment networks with 1000 nodes and $m = 4$. To prevent the walker from being trapped in a node with no outbounding edges, we merge the two networks using `add_edges()` by adding the edges of the second graph to the first graph (edge list obtained via `as_edgelist()`) with a shuffling of the indices of the nodes of the second network using `permute()`. Afterwards, we are asked to measure the probability that the walker visits each node. Only the visits beyond $ln(n)$ steps are recorded, as $ln(n)$ is the number of steps required for exponential convergence. We perform 100 random walks, each with 1000 steps. The node visit probability is given by:

$$\text{Node Visit Probabilities} = \frac{\text{Visited Nodes}}{100 \times (1000 - \ln(n))}$$

The plot of the probability that the walker visits each node, as well as the plot of the visit probability versus degree of nodes, are shown in Figure 39.

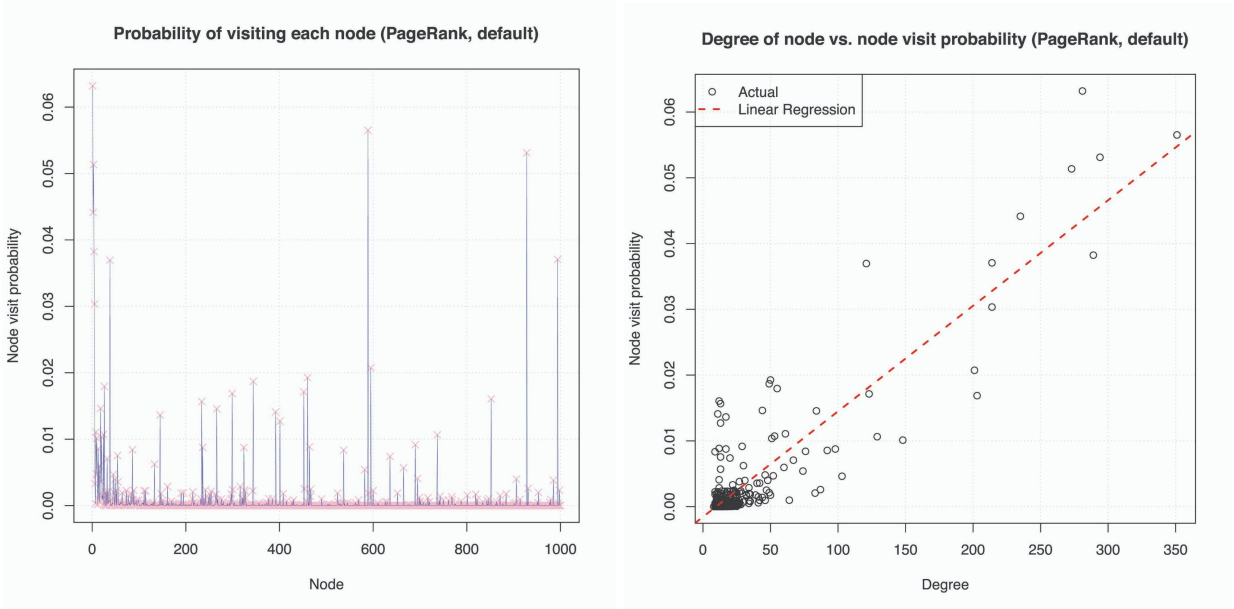


Figure 39: (Left): Probability of visiting each node, (Right): Degree of node versus node visit probability.

From Figure 39 (Left), we can see that some of the nodes have a higher probability of being visited than others. This is explained by the plot in Figure 39 (Right), which shows that there is a strong positive linear relationship (correlation coefficient: 0.902458) between the degree of nodes and the node visit probability. In other words,

nodes with a higher degree are more likely to be visited by the random walker than others. This is expected because in preferential attachment networks, doing random walk for a sufficiently large number of steps gives the steady state occupancy vector, which is used to pick the nodes in a preferential manner. Nodes with higher degree have a higher probability of getting selected by the walker. The implicit goal of the walker in preferential attachment models is to end up at a node with a very high extent of connectedness. Mathematically, the pagerank score vector π (steady-state node occupancy vector) for strongly connected nodes in the directed networks is given by:

$$\pi = P^T\pi, \;\; \pi(i) = \sum_{j=1}^{|V|} P_{ji}\pi(j). \;\; , i = [1, |V|]$$

$$P = (\Sigma_{\text{out}})^{-1}A, \; \Sigma_{\text{out}} = \text{diag}(k_{\text{out}}(1)^{-1}, k_{\text{out}}(2)^{-1}, , k_{\text{out}}(|V|)^{-1})$$

Therefore,

$$\pi(i) = \sum_{j=1}^{|V|} \frac{1}{k_{\text{out}}(j)} A_{ji}\pi(j)$$

If the degree of certain nodes are large, the probability of going from one node to those high degree nodes P_{ij} is large (from the self-consistency equation), which leads to the page rank score of those nodes being large as well. Thus, those nodes are visited by the walker more often. The gradient of the line in Figure 39 (right) is 0.0001605.

Question 3(b):

In this question, we are asked to use random walk on the network created in Question 3(a) to simulate PageRank with teleportation, which is a trick used to find the pagerank scores of vertices in the networks that are not strongly connected. This allows nodes with low-degrees to get a chance to be visited by the walker. With teleportation, a random walker teleports to a random node with probability α and follows one of the outgoing edges with probability $(1-\alpha)$. We choose α to be 0.15. The plot of the probability that the walker visits each node, as well as the plot of the visit probability versus degree of nodes, are shown in Figure 40.

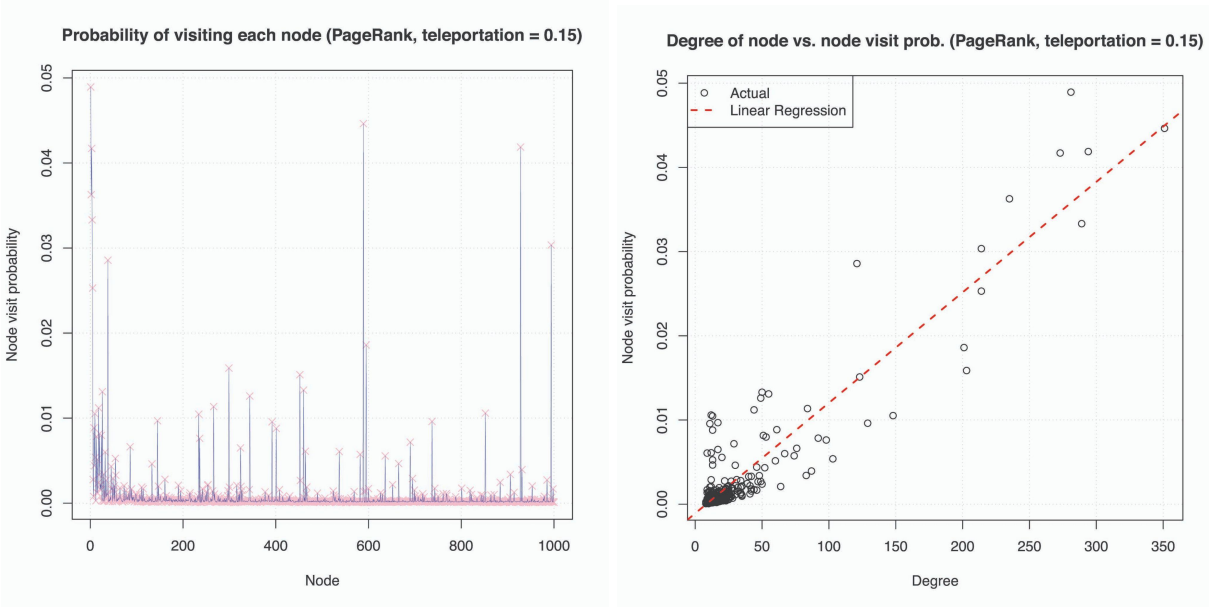


Figure 40: (Left): Probability of visiting each node, (Right): Degree of node versus node visit probability.

Similar to the plots in Figure 39, from Figure 40 (Left), we can see that some of the nodes still have a higher probability of being visited than others. This is explained by the plot in Figure 40 (Right), which shows that there is a strong positive linear relationship (correlation coefficient: 0.930428) between the degree of nodes and the node visit probability. We have already explained why there is a linear relationship between the degree of nodes and node visit probability in Question 3(a). However, the slope of the linear regression line (Figure 40 (Right)) with teleportation is 0.0001313, which is lower than the slope of the line without teleportation (Figure 39 (Right)), which is 0.0001605. This means that with teleportation activated, the probability of visiting only highly connected nodes has reduced, and now, the lower degree nodes are also being visited by the random walker. This can also be explained by the lower value of maximum node visit probability in Figure 40 (Left), which is ~ 0.05, compared to the plot in Figure 39 (Right), which is ~0 06. Mathematically,

$$\pi(i) = (1-\alpha)\sum_{j=1}^{|V|}\frac{1}{k_{\text{out}}(j)}A_{ji}\pi(j) + \frac{\alpha}{|V|}$$

$(1-\alpha)$ reduces the influence of the first term in the above equation, thereby reducing the probability of node visits to nodes with high connectedness.

Question 4(a):

In this question, we are asked to use random walk on the network created in Question 3(a) to simulate personalized PageRank with teleportation, where the teleportation probability to each node is proportional to its PageRank and not $1/N$ like in Question 3(b). The plot of the probability that the walker visits each node, as well as the plot of the visit probability versus degree of nodes, are shown in Figure 41.

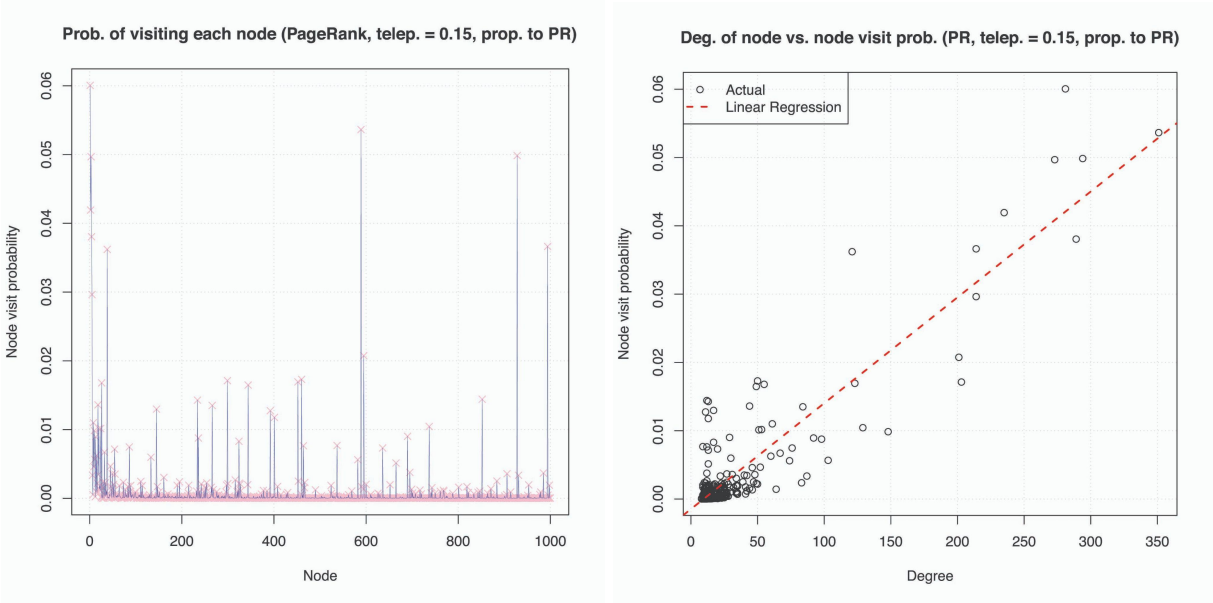


Figure 41: (Left): Probability of visiting each node, (Right): Degree of node versus node visit probability.

Similar to the plots in Figure 39, from Figure 40 (Left), we can see that some of the nodes still have a higher probability of being visited than others. This is explained by the plot in Figure 40 (Right), which shows that there is a strong positive linear relationship (correlation coefficient: 0.910936) between the degree of nodes and the node visit probability. We have already explained why there is a linear relationship between the degree of nodes and node visit probability in Question 3(a) and Question 3(b). The key difference is the slope of the line:

- Pagerank, no teleportation (Question 3(a)): 0.0001605
- Pagerank, with teleportation (Question 3(b)): 0.0001313
- Personalized Pagerank, with teleportation: 0.000155

We see that the slope of the line for personalized Pagerank is slightly lower than the slope of the line for vanilla Pagerank with teleportation. This means that with teleportation activated, the probability of visiting only highly connected nodes has reduced, and now, some of the lower degree nodes are also being visited by the random walker. However, pagerank with vanilla teleportation has an even smaller slope, indicating that the degree of nodes being visited by the random walker in personalized Pagerank even with teleportation is still higher than pagerank with vanilla teleportation. This is because teleportation in personalized Pagerank is happening to those nodes with a high Pagerank value, effectively cancelling out the effect of vanilla teleportation to nodes with a low

Pagerank score (low Pagerank score signifies lower degree of nodes). We can think of personalized Pagerank with teleportation being in the middle of Pagerank with and without teleportation. Mathematically,

$$\pi(i) = (1-\alpha)\sum_{j=1}^{|V|}\frac{1}{k_{\text{out}}(j)}A_{ji}\pi(j) + \alpha\sum_{j=1}^{|V|}\pi^*(i)\pi(j)$$

$$= (1-\alpha)\sum_{j=1}^{|V|}\frac{1}{k_{\text{out}}(j)}A_{ji}\pi(j) + \alpha\pi^*(i)$$

$$\pi^*(i) = \sum_{j=1}^{|V|}P_{ji}\pi(j)$$

From the equation above, we see that personalized Pagerank is intuitively similar to Pagerank with no teleportation. This is why the node visit probability plots from Question 3(a) are also similar to the plots in this question, with peak probability in both cases being ~0.06.

Question 4(b):

In this question, we are asked to repeat Question 4(a) if teleportations land only on those two nodes with equal probabilities having median Pagerank. The plot of the probability that the walker visits each node, as well as the plot of the visit probability versus degree of nodes, are shown in Figure 42. Figure 43 provides a graphical summary of how the Pagerank values of each node are affected, with blue showing Pagerank values of the original network and red showing the Pagerank values after teleportation to node with median Pagerank.

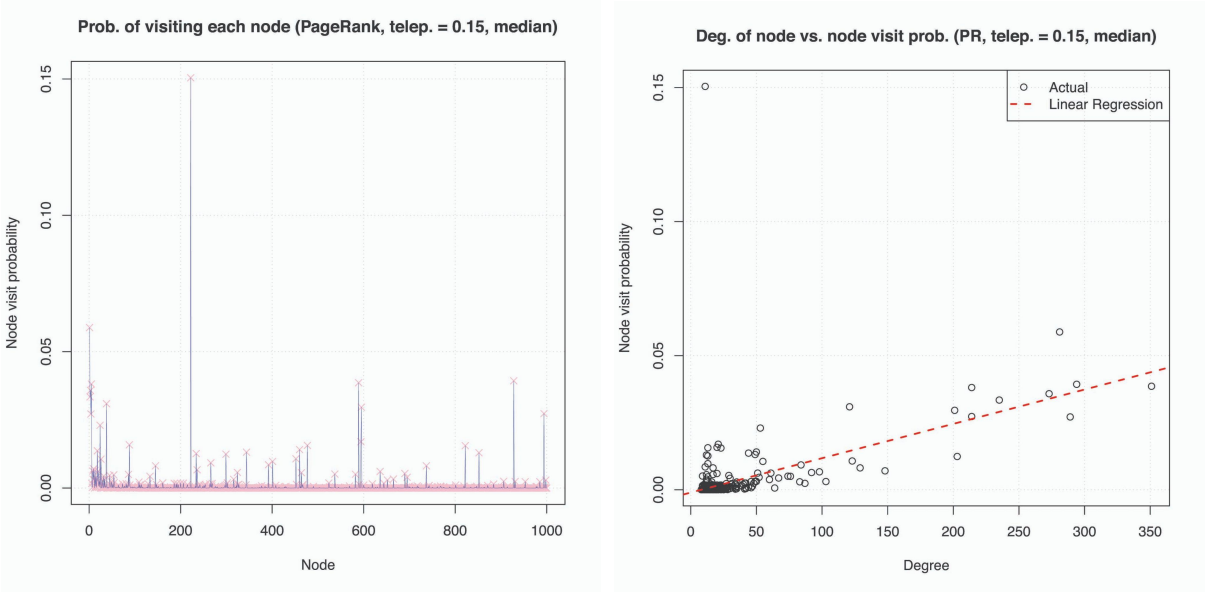


Figure 42: (Left): Probability of visiting each node, (Right): Degree of node versus node visit probability.

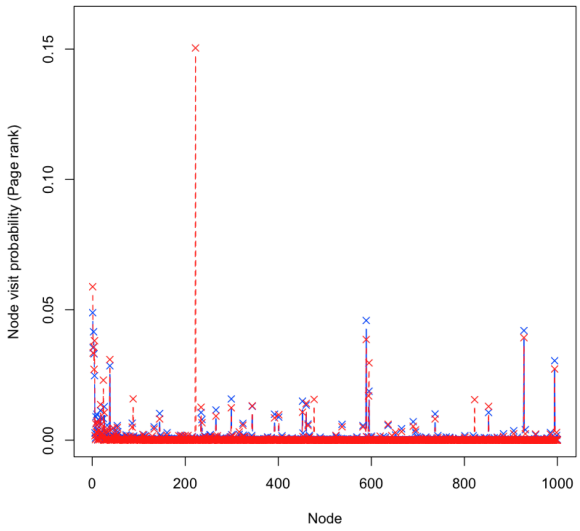


Figure 43: Pagerank values for (blue) entire network (red) after teleportation to node with median Pagerank.

From Figure 42, we see that only the node with the median Pagerank value has a high probability of being visited. The probability of visiting that node is exactly 0.15, which is equal to the teleportation probability

$\alpha = 0.15$. The correlation coefficient for the plot in Figure 42 (Right) is 0.563365, which indicates that the expected linear relationship between degree of node and node visit probability is no longer true if we consider the case of teleporting to the median Pagerank node. This is because earlier, the peak probability of visiting even the highest degree nodes were ~0.05-0.06, which has now more than doubled for the node with median Pagerank. Otherwise, the higher degree nodes still have a higher chance of being visited as shown in Figure 42 (Right).

In Figure 43, we see a large boost in Pagerank score for the node with median Pagerank, corresponding to the node where teleportation is happening. This spike is absent in the original network. This causes redistribution of the Pagerank score of other nodes, decreasing their Pagerank scores slightly to accommodate the increase in Pagerank score of the teleportation node. In addition, some of the nodes which receive in-link edges from the teleportation node had their pagerank scores amplified as well. Mathematically,

$$\begin{aligned}
\pi(i) &= (1-\alpha)\sum_{j=1}^{|V|}\frac{1}{k_{\text{out}}(j)}A_{ji}\pi(j) + \alpha\sum_{j=1}^{|V|}\frac{1}{|T|}\pi(j) \\
&= (1-\alpha)\sum_{j=1}^{|V|}\frac{1}{k_{\text{out}}(j)}A_{ji}\pi(j) + \frac{\alpha}{|T|}, i \in T \\
&= (1-\alpha)\sum_{j=1}^{|V|}\frac{1}{k_{\text{out}}(j)}A_{ji}\pi(j), i \notin T
\end{aligned}$$

We can see that the teleportation nodes in the set $|T|$ (called trusted nodes) is getting a Pagerank score boost, including those nodes that are connected to the teleportation node.

Question 4(c):

The original Pagerank equation is given by:

$$\pi(i) = \sum_{j=1}^{|V|} \frac{1}{k_{\text{out}}(j)} A_{ji} \pi(j)$$

The personalized Pagerank with teleportation to trusted nodes is given by:

$$\pi(i) = (1-\alpha) \sum_{j=1}^{|V|} \frac{1}{k_{\text{out}}(j)} A_{ji} \pi(j) + \alpha \sum_{j=1}^{|V|} \frac{1}{|T|} \pi(j)$$

To account for self-reinforcement, we can combine the two equations as follows:

$$\pi(i) = (1-\alpha) \sum_{j=1}^{|V|} \frac{1}{k_{\text{out}}(j)} A_{ji} \pi(j) + \frac{\alpha}{|T|}, i \in T$$

$$\pi(i) = (1-\alpha) \sum_{j=1}^{|V|} \frac{1}{k_{\text{out}}(j)} A_{ji} \pi(j), i \notin T$$

Project1_Saha_Young

April 14, 2021

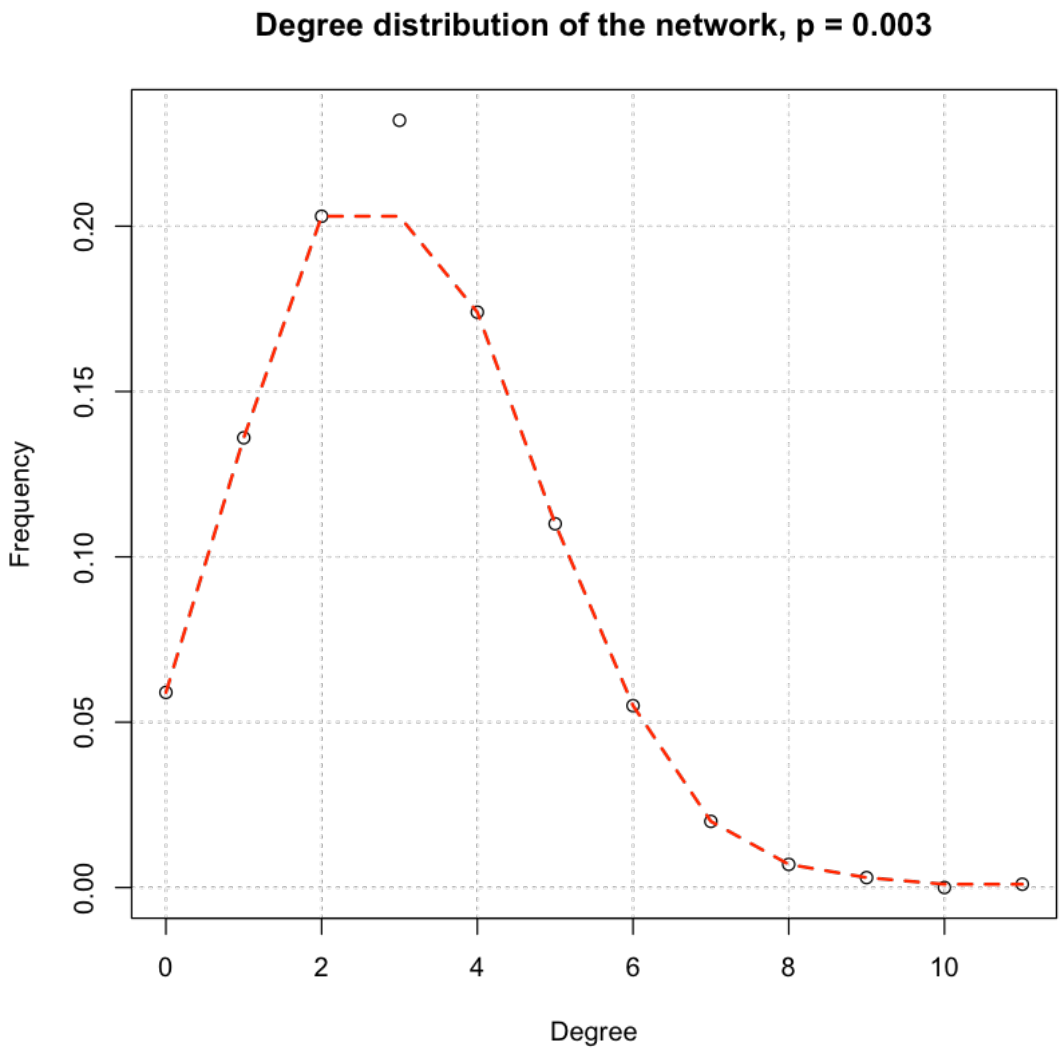
[10]:
```
library(igraph)
library(resample)
library(textTinyR)
library(Matrix)
library(pracma)
```

1 Generating Random Networks

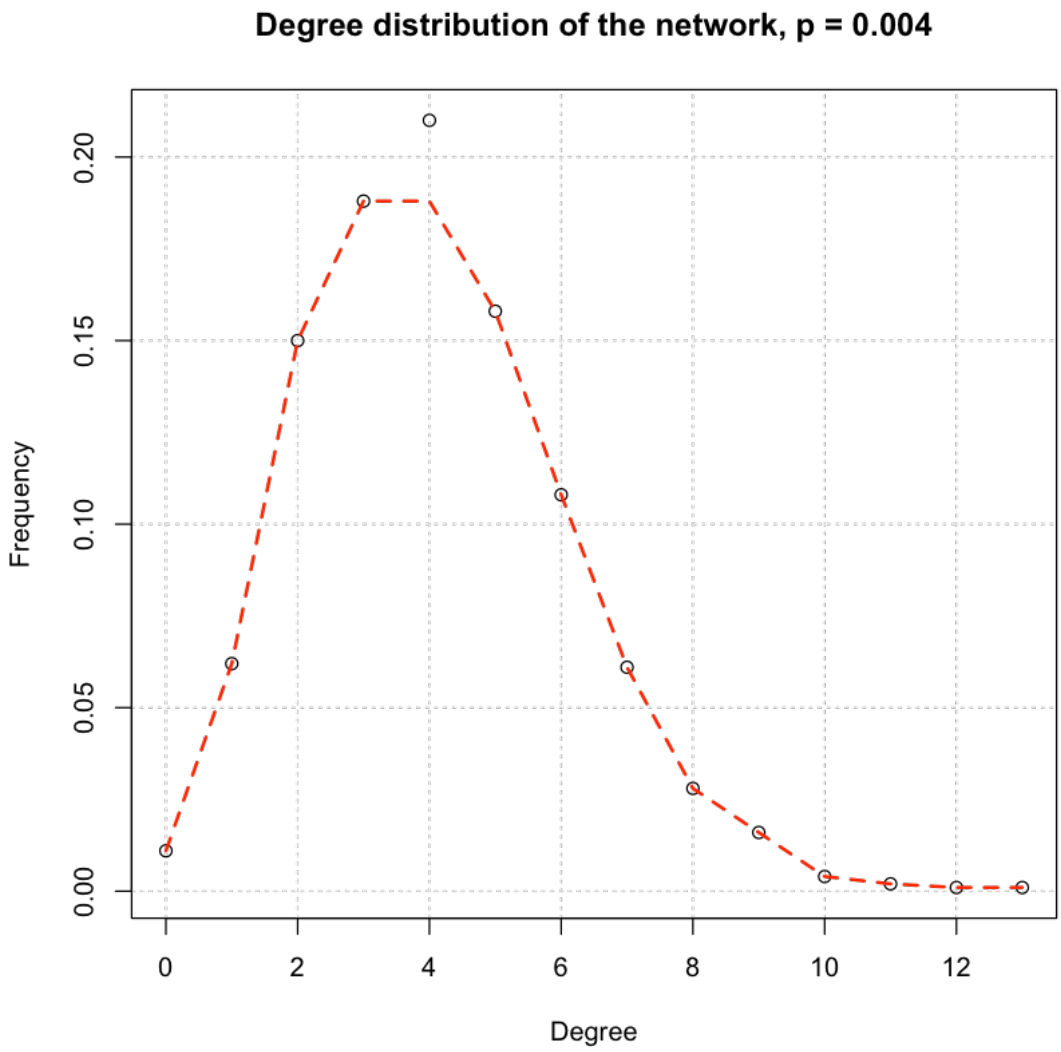
1.1 Question 1a

[2]:
```
pList <- c(0.003, 0.004, 0.01, 0.05, 0.1)
i = 1
for (p in pList) {
    g <- sample_gnp(1000,p,directed = FALSE)
    plot(seq_along(degree.distribution(g)) - 1, degree.distribution(g),
         main=sprintf("Degree distribution of the network, p = %0.
 ↪3f",p),xlab="Degree",ylab="Frequency",grid())
    lines(seq_along(degree.distribution(g)) - 1, smooth(degree.
 ↪distribution(g),kind = '3RS3R'),lwd = 2,col='red',lty=2)
    dev.copy2eps(file=sprintf('Q1a%d.eps',i))
    print(sprintf("p=%.3f: Mean=%f, Variance=%f", p, mean(degree(g)),␣
 ↪var(degree(g))))
    i <- i+1
}
```

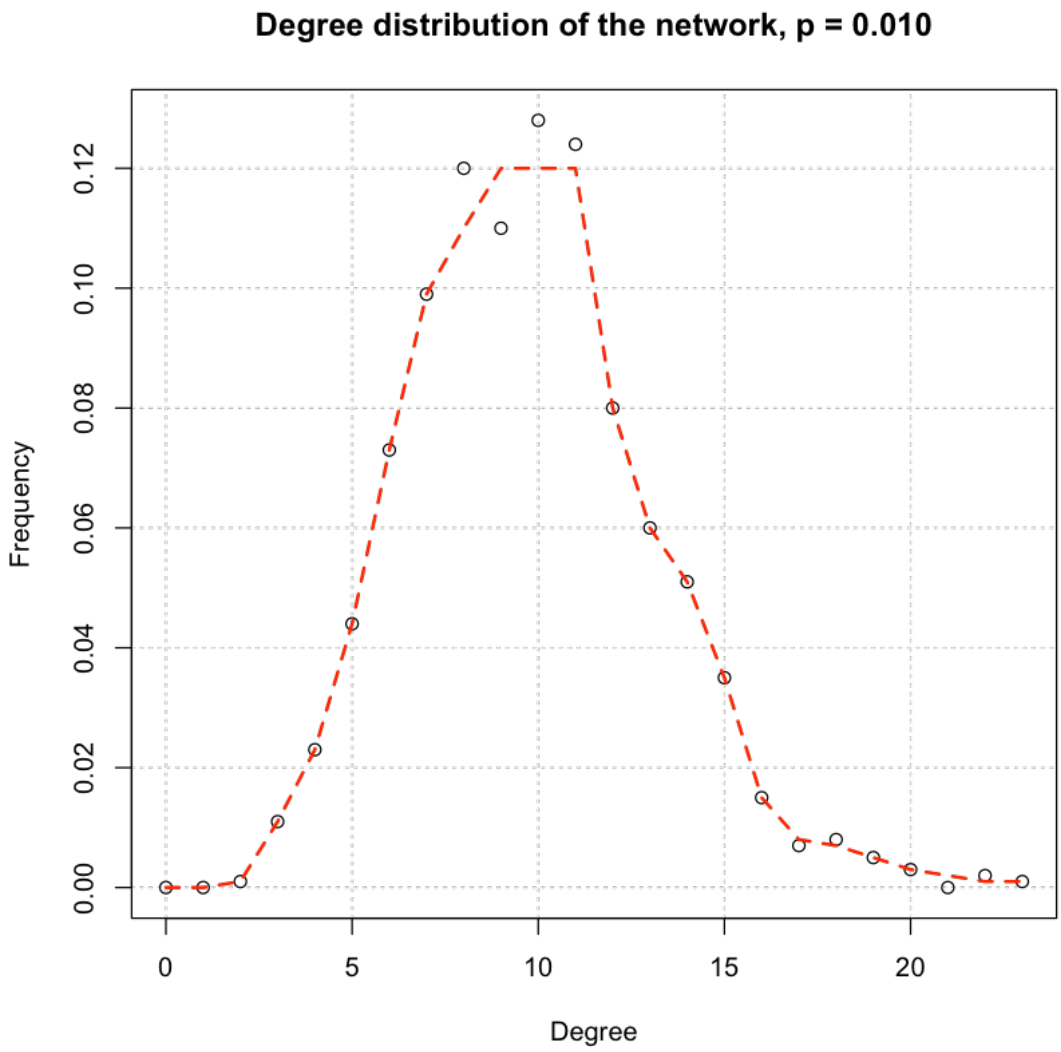
```
[1] "p=0.003: Mean=3.048000, Variance=3.054751"
```



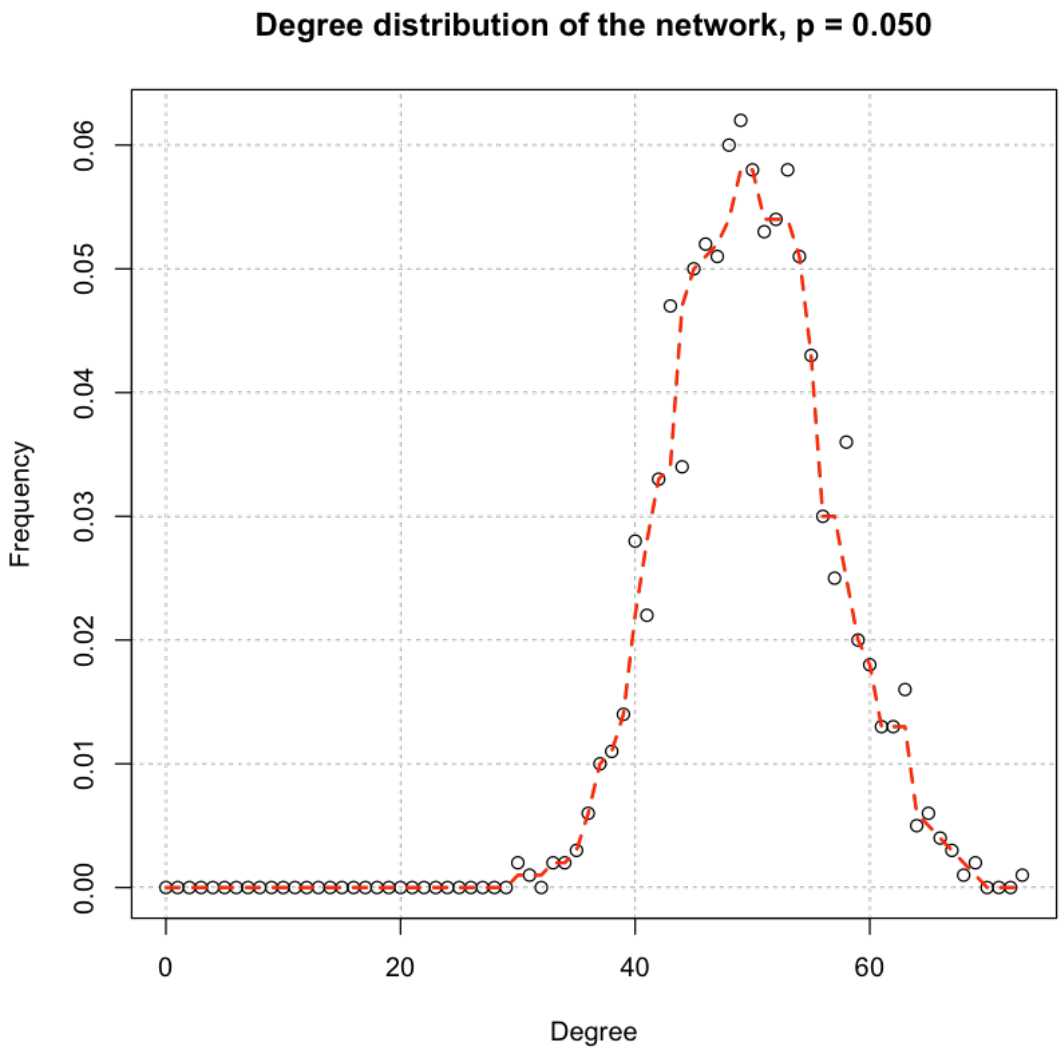

```
[1] "p=0.004: Mean=4.086000, Variance=3.892496"
```



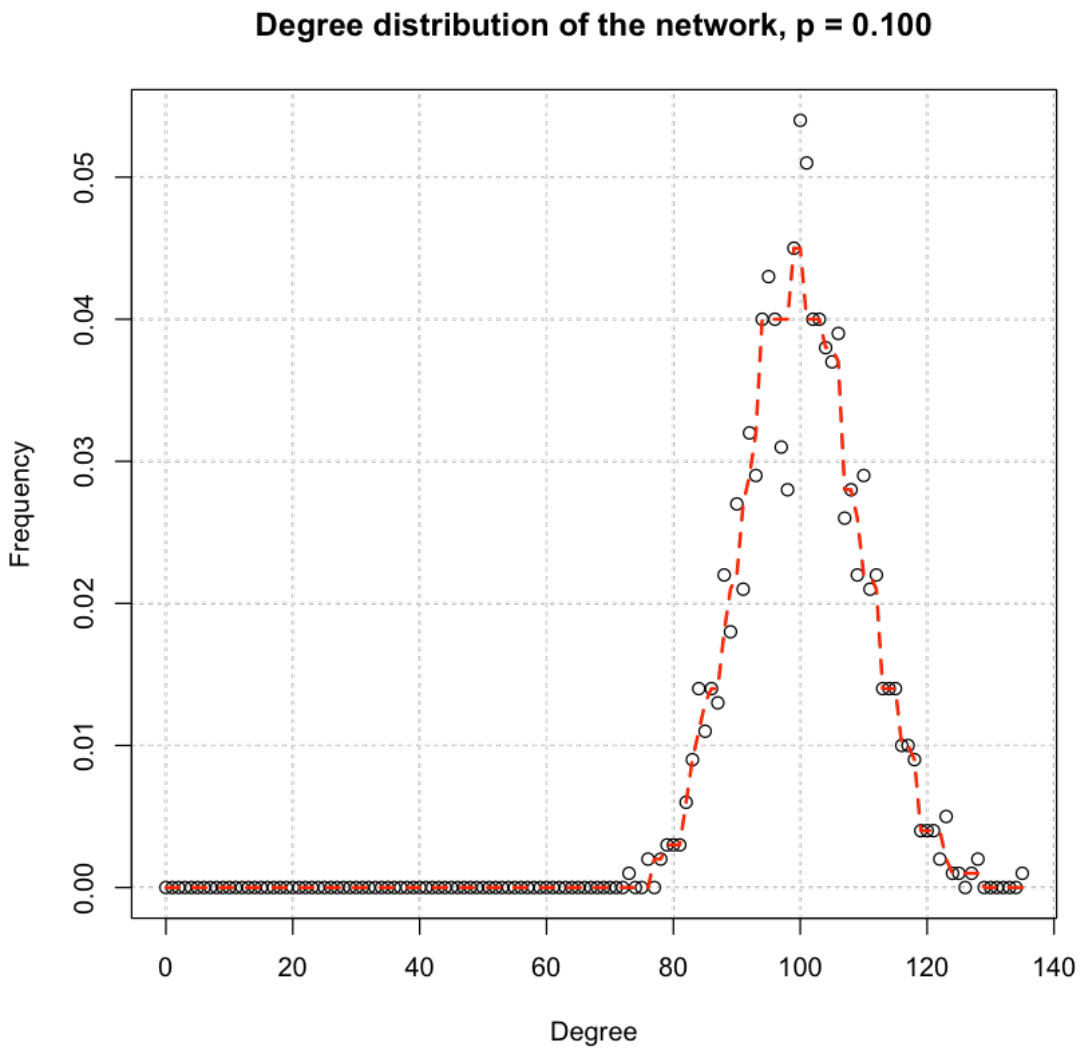

```
[1] "p=0.010: Mean=9.776000, Variance=10.372196"
```




```
[1] "p=0.050: Mean=49.880000, Variance=44.820420"
```




```
[1] "p=0.100: Mean=100.354000, Variance=85.502186"
```



1.2 Question 1b

[3]:
```
for (p in pList) {
    gccflag = 1
    count = 0
    diameterGCC = -1
    GCCnode = -1
    GCCedge = -1
    for (i in 1:1000) {
        g <- sample_gnp(1000,p,directed = FALSE)
        if (is.connected(g)){
            count <- count + 1
        }
```

```
        else if (!is.connected(g)){
            if(gccflag){
                g.components <- clusters(g)
                ix <- which.max(g.components$csize)
                g.giant <- induced.subgraph(g, which(g.components$membership ==␣
↪ix))
                diameterGCC <- diameter(g.giant, directed = FALSE)
                GCCnode <- vcount(g.giant)
                GCCedge <- ecount(g.giant)
                gccflag = 0
            }
        }
    }
    print(sprintf("Probability of connectedness for p = %0.3f: %f", p, count/
↪1000))
    print(sprintf("Diameter of GCC for a random non-connected network: %d (-1␣
↪means no non-connected network exists)", diameterGCC))
    print(sprintf("Number of nodes of GCC for a random non-connected network:␣
↪%d (-1 means no non-connected network exists)", GCCnode))
    print(sprintf("Number of edges of GCC for a random non-connected network:␣
↪%d (-1 means no non-connected network exists)", GCCedge))
    ␣
↪print("----------------------------------------------------------------------------------
}
```

```
[1] "Probability of connectedness for p = 0.003: 0.000000"
[1] "Diameter of GCC for a random non-connected network: 13 (-1 means no non-
connected network exists)"
[1] "Number of nodes of GCC for a random non-connected network: 949 (-1 means no
non-connected network exists)"
[1] "Number of edges of GCC for a random non-connected network: 1570 (-1 means
no non-connected network exists)"
[1] "----------------------------------------------------------------------------
------------------------------------------------"
[1] "Probability of connectedness for p = 0.004: 0.000000"
[1] "Diameter of GCC for a random non-connected network: 11 (-1 means no non-
connected network exists)"
[1] "Number of nodes of GCC for a random non-connected network: 977 (-1 means no
non-connected network exists)"
[1] "Number of edges of GCC for a random non-connected network: 2052 (-1 means
no non-connected network exists)"
[1] "----------------------------------------------------------------------------
------------------------------------------------"
[1] "Probability of connectedness for p = 0.010: 0.960000"
[1] "Diameter of GCC for a random non-connected network: 6 (-1 means no non-
connected network exists)"
[1] "Number of nodes of GCC for a random non-connected network: 999 (-1 means no
```

```
non-connected network exists)"
[1] "Number of edges of GCC for a random non-connected network: 5162 (-1 means
no non-connected network exists)"
[1] "----------------------------------------------------------------------------
------------------------------------------------"
[1] "Probability of connectedness for p = 0.050: 1.000000"
[1] "Diameter of GCC for a random non-connected network: -1 (-1 means no non-
connected network exists)"
[1] "Number of nodes of GCC for a random non-connected network: -1 (-1 means no
non-connected network exists)"
[1] "Number of edges of GCC for a random non-connected network: -1 (-1 means no
non-connected network exists)"
[1] "----------------------------------------------------------------------------
------------------------------------------------"
[1] "Probability of connectedness for p = 0.100: 1.000000"
[1] "Diameter of GCC for a random non-connected network: -1 (-1 means no non-
connected network exists)"
[1] "Number of nodes of GCC for a random non-connected network: -1 (-1 means no
non-connected network exists)"
[1] "Number of edges of GCC for a random non-connected network: -1 (-1 means no
non-connected network exists)"
[1] "----------------------------------------------------------------------------
------------------------------------------------"
```

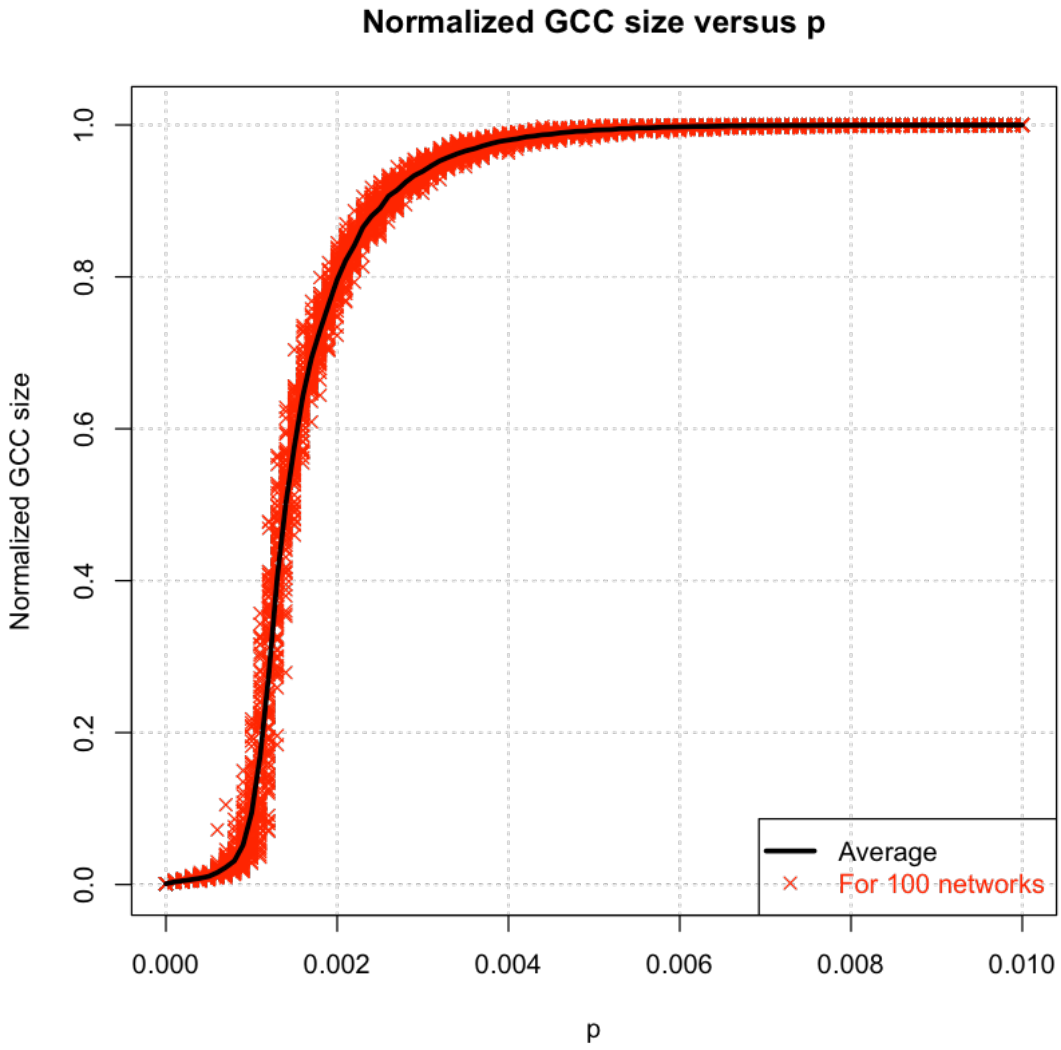
1.3 Question 1c

[9]:
```
pList <- seq(0,0.01,0.0001)
m = matrix(data = 0.0, nrow = 100, ncol = length(pList))
v = matrix(data= 0.0, nrow=length(pList),ncol=1)
j = 1
for (p in pList){
    for (i in 1:100){
    g <- sample_gnp(1000,p,directed = FALSE)
    g.components <- clusters(g)
    m[i,j] <- max(g.components$csize)/1000
    }
    v[j] = mean(m[,j])
    j <- j+1
}
plot(rep(pList[1],100),m[1:100,1],xlim = c(0,0.01), ylim = c(0,1.
 ↪01),xlab="p",main = "Normalized GCC size versus p",ylab="Normalized GCC␣
 ↪size",grid(),pch=4,col='red')
j = 2
for (p in pList[2:length(pList)]){
    points(rep(p,100),m[1:100,j],pch=4,col='red')
    j <- j+1
}
```

```
lines(pList,v,lwd=3)
legend('bottomright', legend = c("Average", "For 100 networks"),
       pch = c(NA, 4), lty = c(1, NA), lwd = c(3,NA),
       col = c(1,'red'), text.col = c(1,'red'))
dev.copy2eps(file='Q1c.eps')
print(sprintf("p at which GCC starts to emerge: %f",pList[min(which(v>0.1))]))
print(sprintf("p at which 99 percent nodes belong to GCC:␣
 ↪%f",pList[mean(which(v>0.99))]))
```

pdf: 2

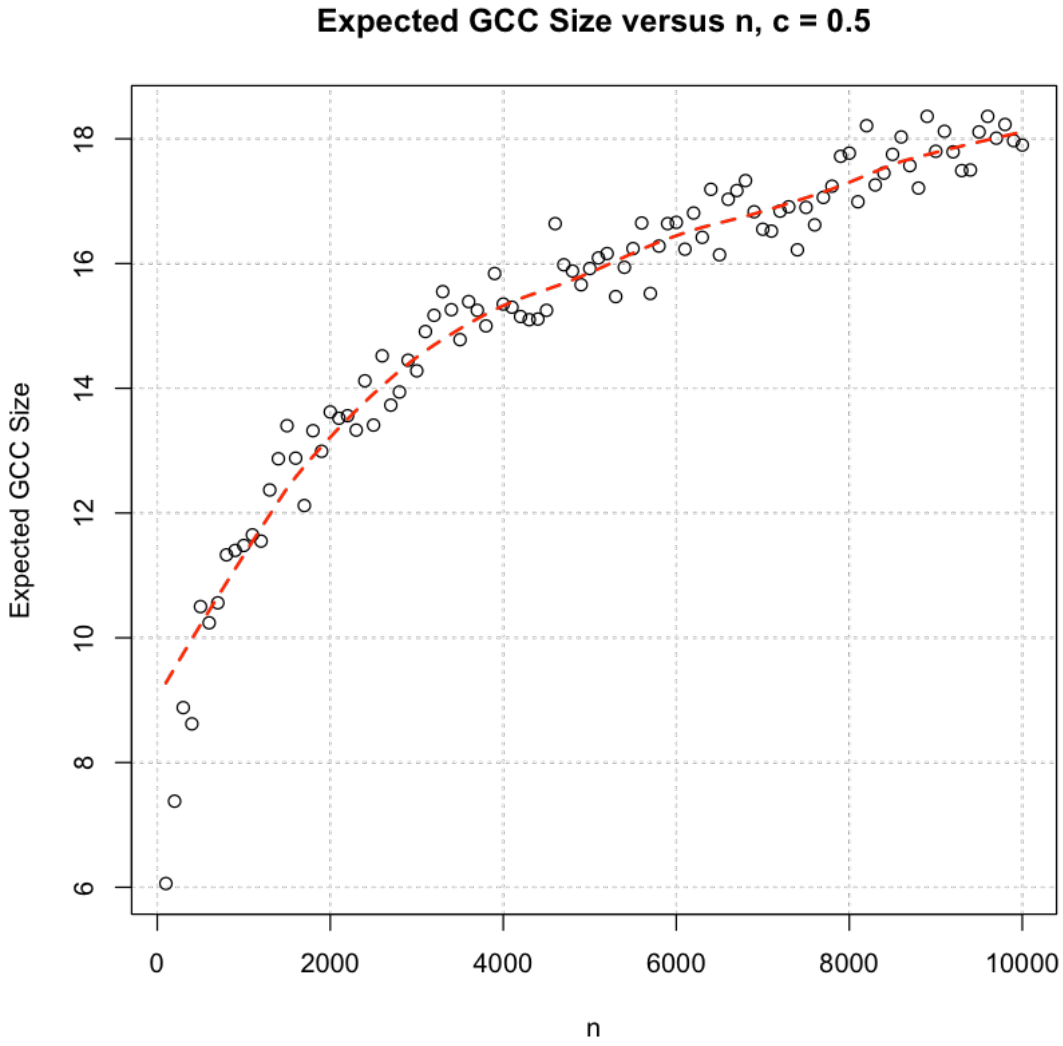
```
[1] "p at which GCC starts to emerge: 0.001100"
[1] "p at which 99 percent nodes belong to GCC: 0.007300"
```



1.4 Question 1d

[5]:
```
nList <- seq(100,10000,100)
q = matrix(data = 0.0, nrow = 100, ncol = length(nList))
r = matrix(data= 0.0, nrow=length(nList),ncol=1)
c = 0.5
j = 1
for (n in nList){
    for (i in 1:100){
    g <- sample_gnp(n,c/n,directed = FALSE)
    g.components <- clusters(g)
    q[i,j] <- max(g.components$csize)
    }
    r[j] <- mean(q[,j])
    j <- j+1
}
plot(nList,r,xlab="n",main = "Expected GCC Size versus n, c = 0.
 ↪5",ylab="Expected GCC Size",grid())
lines(lowess(nList,r,f = 0.3), col="red",lwd = 2,lty=2)
dev.copy2eps(file='Q1da.eps')
```

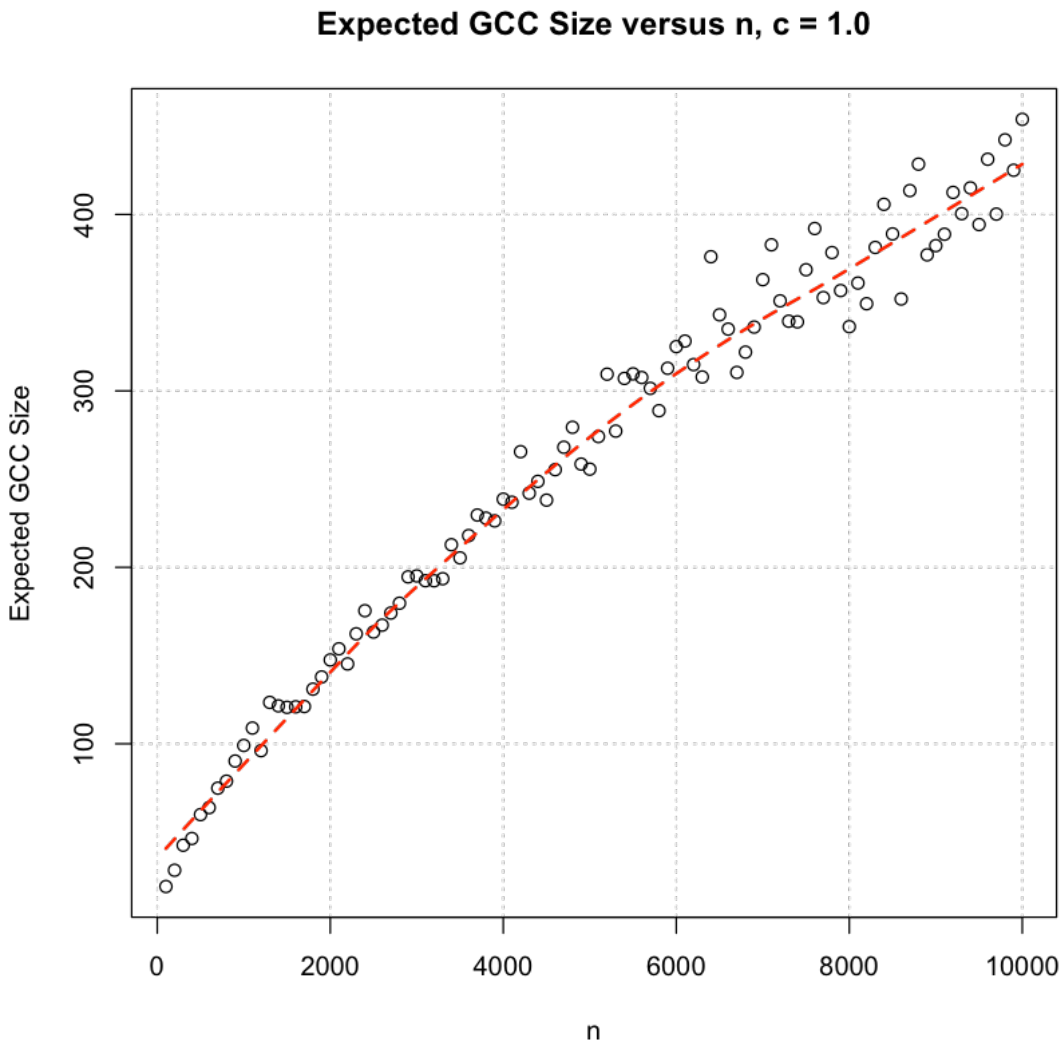
pdf: 2



[6]:
```
nList <- seq(100,10000,100)
q = matrix(data = 0.0, nrow = 100, ncol = length(nList))
r = matrix(data= 0.0, nrow=length(nList),ncol=1)
c = 1.0
j = 1
for (n in nList){
    for (i in 1:100){
    g <- sample_gnp(n,c/n,directed = FALSE)
    g.components <- clusters(g)
    q[i,j] <- max(g.components$csize)
    }
    r[j] <- mean(q[,j])
    j <- j+1
```

```
}
plot(nList,r,xlab="n",main = "Expected GCC Size versus n, c = 1.
 ↪0",ylab="Expected GCC Size",grid())
lines(lowess(nList,r,f = 0.5), col="red",lwd = 2,lty=2)
dev.copy2eps(file='Q1db.eps')
```

pdf: 2

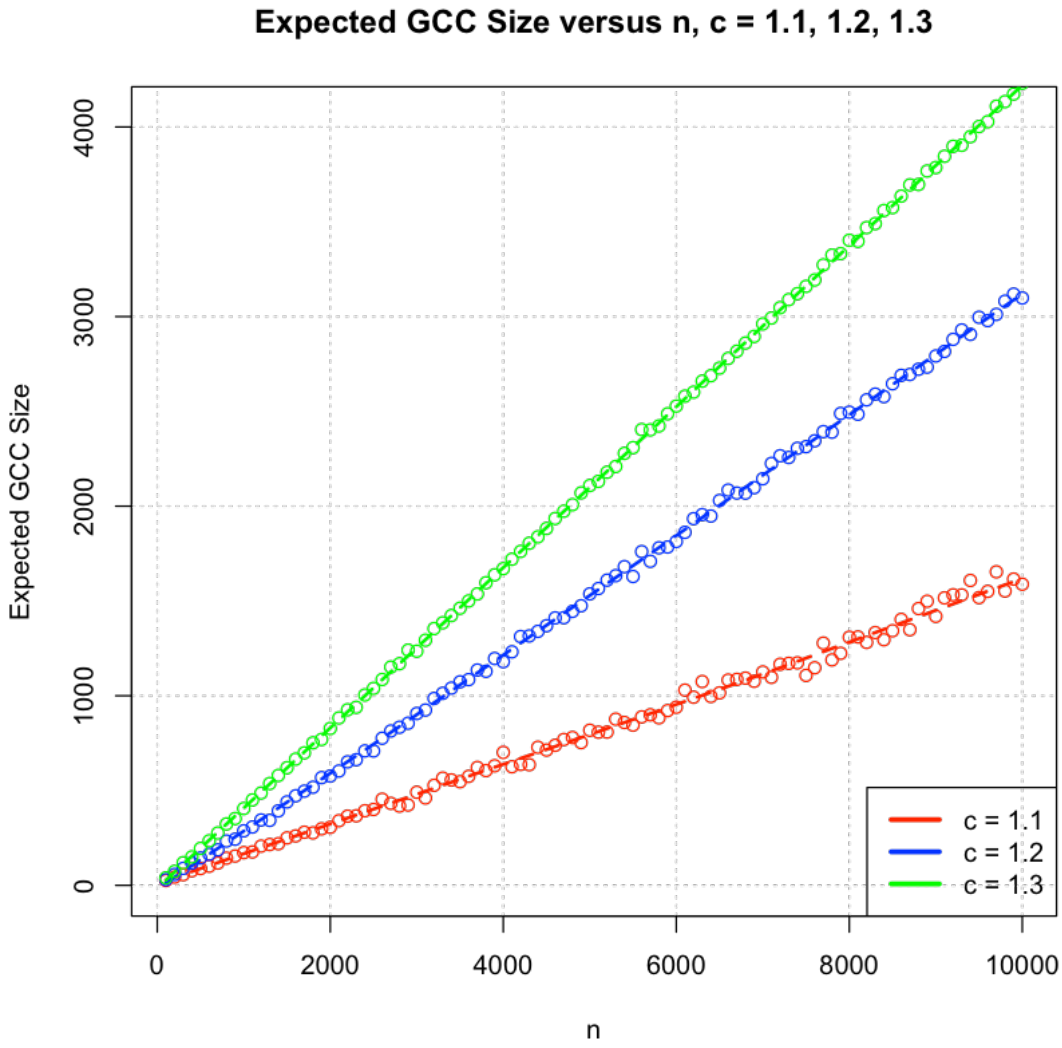


[7]:
```
nList <- seq(100,10000,100)
q = matrix(data = 0.0, nrow = 100, ncol = length(nList))
r = matrix(data= 0.0, nrow=length(nList),ncol=1)
c = 1.1
j = 1
```

```
for (n in nList){
    for (i in 1:100){
    g <- sample_gnp(n,c/n,directed = FALSE)
    g.components <- clusters(g)
    q[i,j] <- max(g.components$csize)
    }
    r[j] <- mean(q[,j])
    j <- j+1
}
plot(nList,r,xlab="n",main = "Expected GCC Size versus n, c = 1.1, 1.2, 1.
 ↪3",ylab="Expected GCC Size",grid(),col="red",ylim = c(0,4050))
lines(lowess(nList,r,f = 0.5), col="red",lwd = 2,lty=2)

q = matrix(data = 0.0, nrow = 100, ncol = length(nList))
r = matrix(data= 0.0, nrow=length(nList),ncol=1)
c = 1.2
j = 1
for (n in nList){
    for (i in 1:100){
    g <- sample_gnp(n,c/n,directed = FALSE)
    g.components <- clusters(g)
    q[i,j] <- max(g.components$csize)
    }
    r[j] = mean(q[,j])
    j <- j+1
}
points(nList,r,xlab="n",col="blue")
lines(lowess(nList,r,f = 0.5), col="blue",lwd = 2,lty=2)

q = matrix(data = 0.0, nrow = 100, ncol = length(nList))
r = matrix(data= 0.0, nrow=length(nList),ncol=1)
c = 1.3
j = 1
for (n in nList){
    for (i in 1:100){
    g <- sample_gnp(n,c/n,directed = FALSE)
    g.components <- clusters(g)
    q[i,j] <- max(g.components$csize)
    }
    r[j] = mean(q[,j])
    j <- j+1
}
points(nList,r,xlab="n",col="green")
lines(lowess(nList,r,f = 0.5), col="green",lwd = 2,lty=2)
legend('bottomright', legend = c("c = 1.1", "c = 1.2", "c = 1.3"),
       lty = c(1, 1, 1), lwd = c(3,3,3),
       col = c('red','blue','green'))
```

```
dev.copy2eps(file='Q1dc.eps')
```

pdf: 2



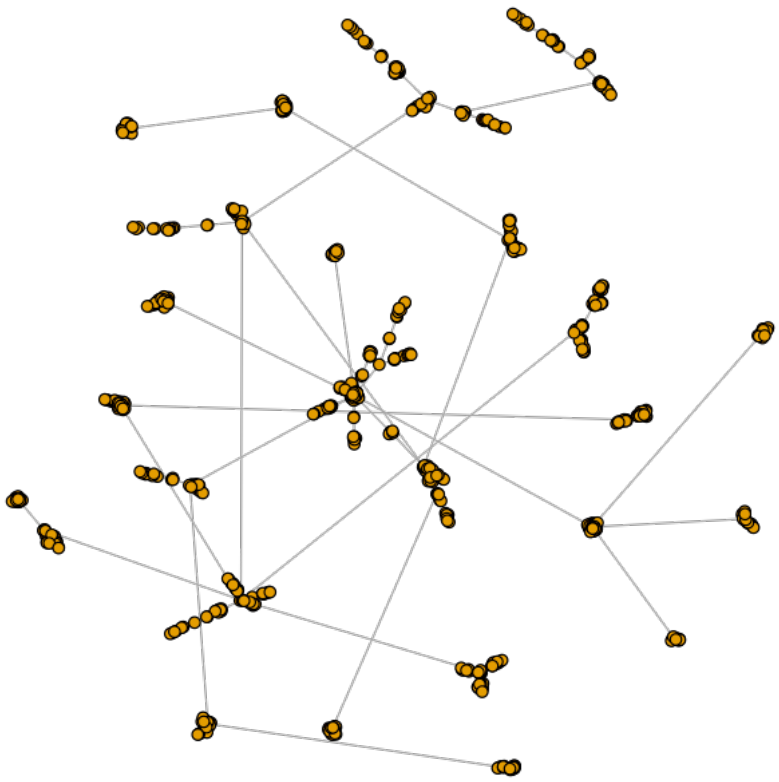
1.5 Question 2a

[8]:
```
count = 0
for (i in 1:100) {
    g <- barabasi.game(1000, m=1, directed=FALSE)
    if (is.connected(g)) {
        count <- count + 1
    }
}
```

```
print(sprintf("Total iterations: 100, Connected graphs: %d",count))
plot(g,vertex.label="",vertex.size=3,main = "Undirected network, preferential␣
 ↪attachment (UNPA), n = 1000, m = 1")
dev.copy2eps(file='Q2a.eps')
```

```
[1] "Total iterations: 100, Connected graphs: 100"
```

pdf: 2

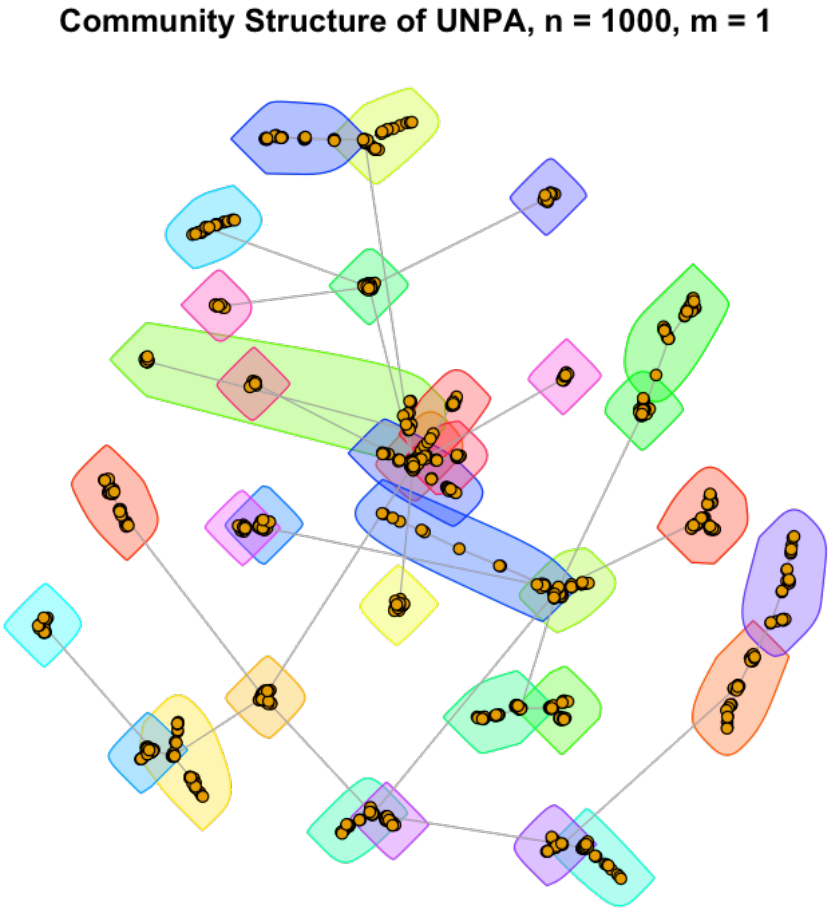


1.6 Question 2b

[16]:
```
comm_struct <- cluster_fast_greedy(g)
mod <- modularity(comm_struct)
print(sprintf("Modularity: %f",mod))
plot(g, mark.groups = groups(comm_struct), vertex.size=3, vertex.
 ↪label="",main="Community Structure of UNPA, n = 1000, m = 1")
dev.copy2pdf(file='Q2ba.pdf')
```

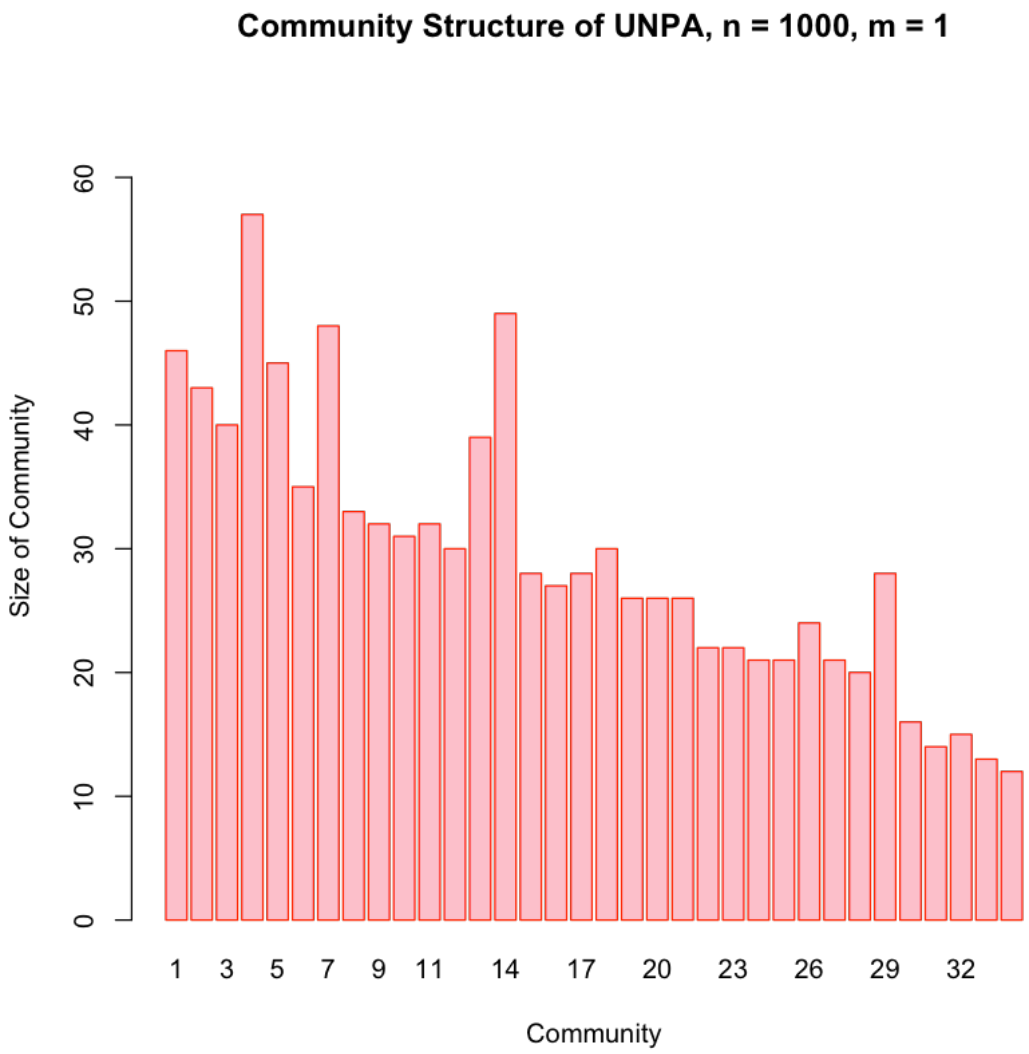
```
[1] "Modularity: 0.933042"
```

pdf: 2



[17]:
```
barplot(as.vector(sizes(comm_struct)),names.arg =␣
↪seq(1,length(comm_struct),1),main="Community Structure of UNPA, n = 1000, m␣
↪= 1",
  xlab="Community",ylab="Size of Community",ylim=c(0,max(as.
↪vector(sizes(comm_struct)))+10),border="red",col="pink")
dev.copy2eps(file='Q2bb.eps')
```

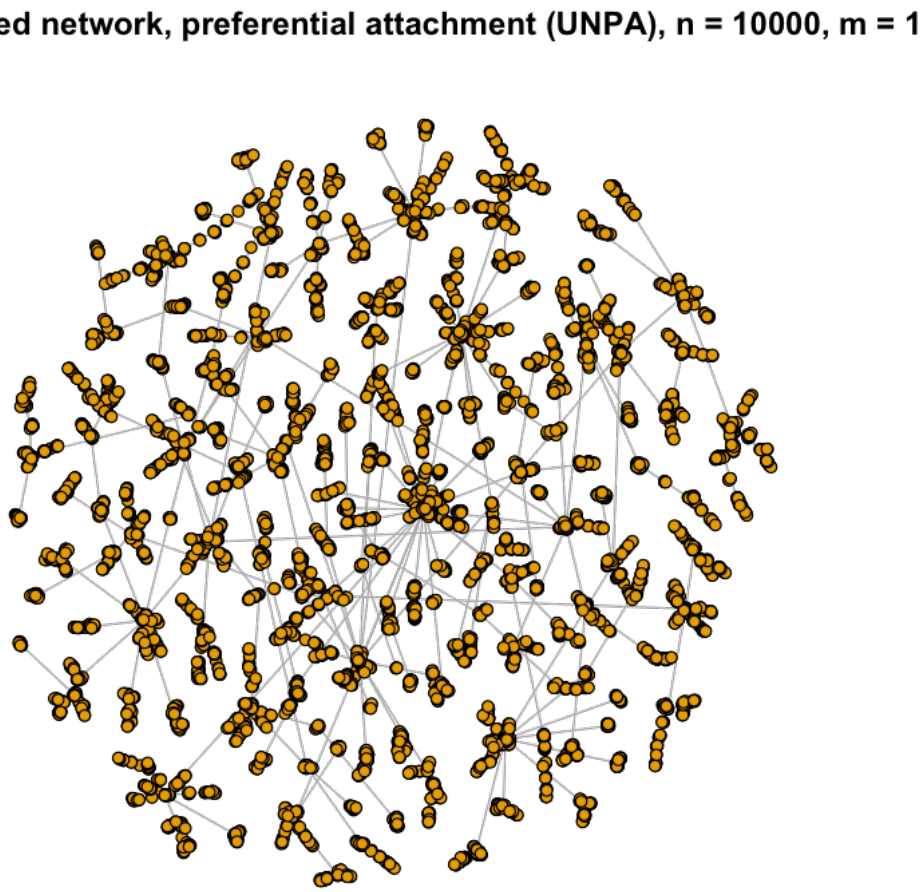
pdf: 2



1.7 Question 2c

[18]:
```
g2 <- barabasi.game(10000, m=1, directed=FALSE)
plot(g2,vertex.label="",vertex.size=3,main = "Undirected network, preferential␣
 ↪attachment (UNPA), n = 10000, m = 1")
dev.copy2eps(file='Q2ca.eps')
```

pdf: 2

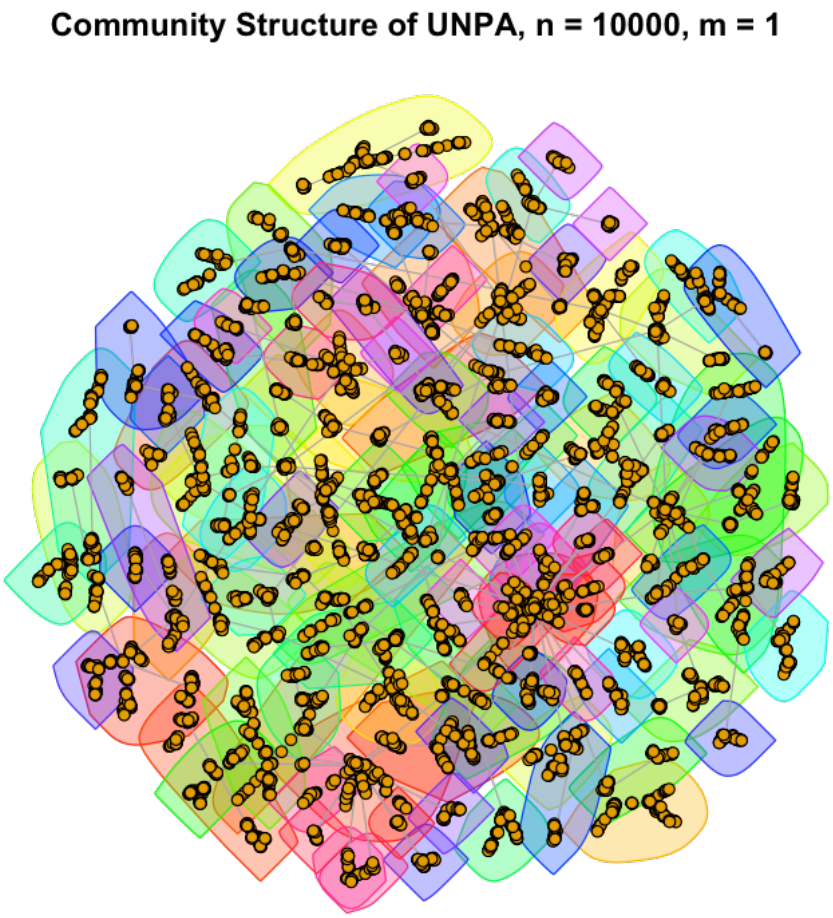


[19]:
```
comm_struct2 <- cluster_fast_greedy(g2)
mod2 <- modularity(comm_struct2)
print(sprintf("Modularity: %f",mod2))
```

```
plot(g2, mark.groups = groups(comm_struct2), vertex.size=3, vertex.
 ↪label="",main="Community Structure of UNPA, n = 10000, m = 1")
dev.copy2pdf(file='Q2cb.pdf')
```

```
[1] "Modularity: 0.977433"
```

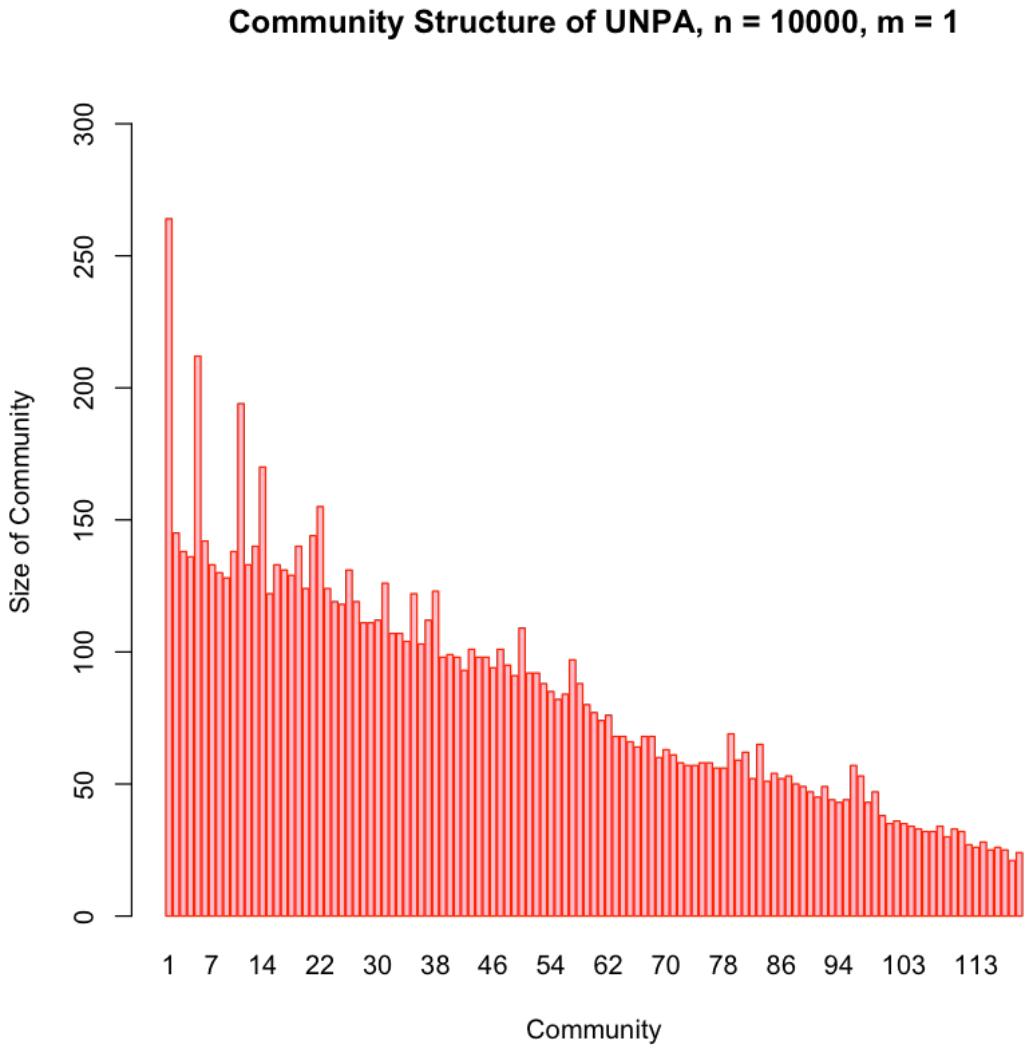
pdf: 2



[20]:
```
barplot(as.vector(sizes(comm_struct2)),names.arg =␣
 ↪seq(1,length(comm_struct2),1),main="Community Structure of UNPA, n = 10000,␣
 ↪m = 1",
   xlab="Community",ylab="Size of Community",ylim=c(0,max(as.
 ↪vector(sizes(comm_struct2)))+50),border="red",col="pink")
```

```
dev.copy2eps(file='Q2cc.eps')
```

pdf: 2

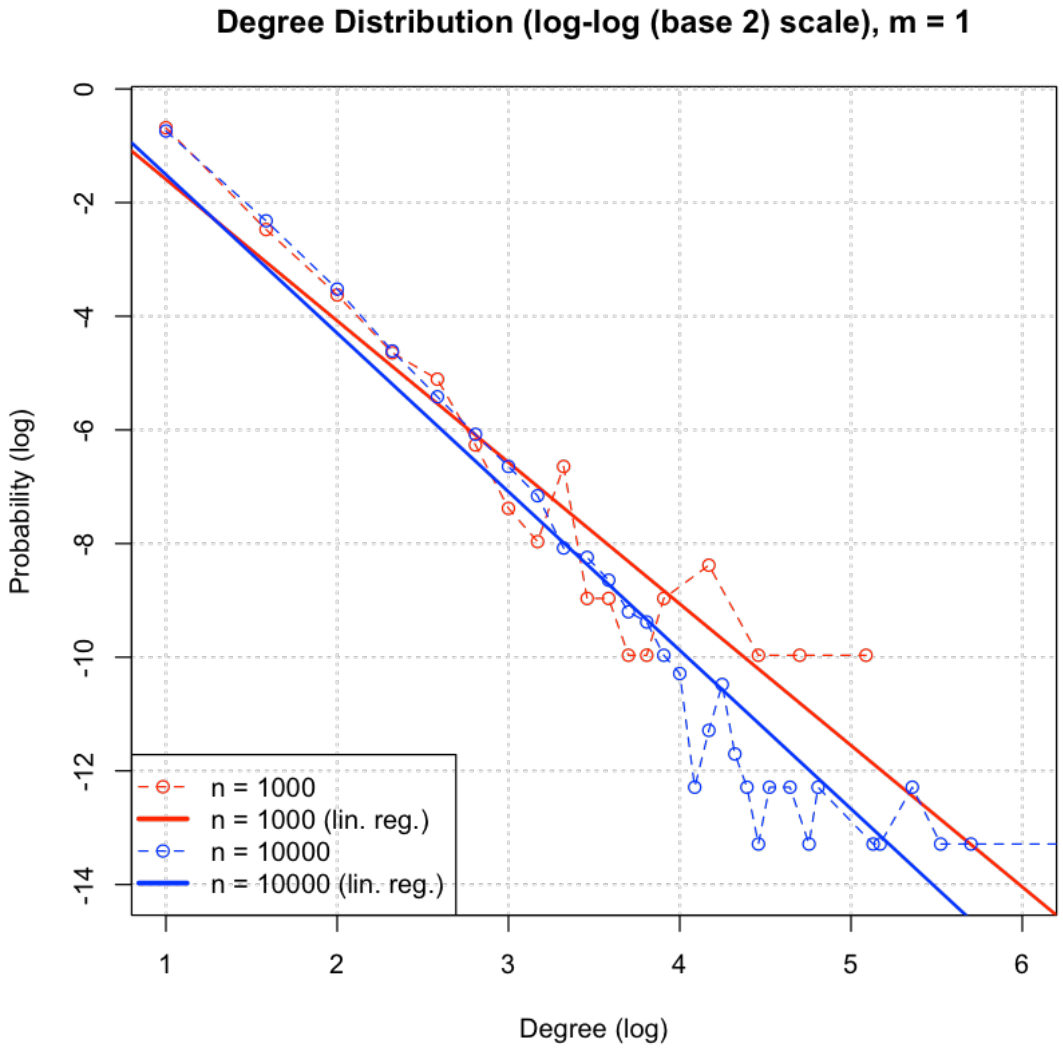


1.8 Question 2d

[21]:
```
deg_dist_1 = degree.distribution(g)
deg_1 <- log2(c(1:length(deg_dist_1)))[which(deg_dist_1 !=0, arr.ind = TRUE)]
dist_1 <- log2(deg_dist_1)[which(deg_dist_1 !=0, arr.ind = TRUE)]
plot(deg_1,dist_1,main="Degree Distribution (log-log (base 2) scale), m = 1",
     xlab="Degree (log)",ylab="Probability␣
 ↪(log)",grid(),col="red",ylim=c(-14,-0.5),xlim=c(1,6))
lines(deg_1,dist_1,lty=2,col="red")
```

```
abline(lm(dist_1 ~ deg_1),col="red",lwd=2)

deg_dist_2 = degree.distribution(g2)
deg_2 <- log2(c(1:length(deg_dist_2)))[which(deg_dist_2 !=0, arr.ind = TRUE)]
dist_2 <- log2(deg_dist_2)[which(deg_dist_2 !=0, arr.ind = TRUE)]
points(deg_2,dist_2,col="blue")
lines(deg_2,dist_2,lty=2,col="blue")
abline(lm(dist_2 ~ deg_2),col="blue",lwd=2)
legend('bottomleft', legend = c("n = 1000", "n = 1000 (lin. reg.)","n = 10000",␣
 ↪"n = 10000 (lin. reg.)"),
       lty = c(2, 1, 2,1), lwd = c(1,3,1,3), pch=c(1,NA,1,NA),
       col = c('red','red','blue','blue'))
dev.copy2eps(file='Q2da.eps')
```

pdf: 2



[22]:
```
print("Slope and intercept for n = 1000:")
print(lm(dist_1 ~ deg_1))
print("Slope and intercept for n = 10000:")
print(lm(dist_2 ~ deg_2))
```

```
[1] "Slope and intercept for n = 1000:"

Call:
lm(formula = dist_1 ~ deg_1)

Coefficients:
(Intercept)        deg_1
     0.9011      -2.4906

[1] "Slope and intercept for n = 10000:"

Call:
lm(formula = dist_2 ~ deg_2)

Coefficients:
(Intercept)        deg_2
      1.283       -2.790
```

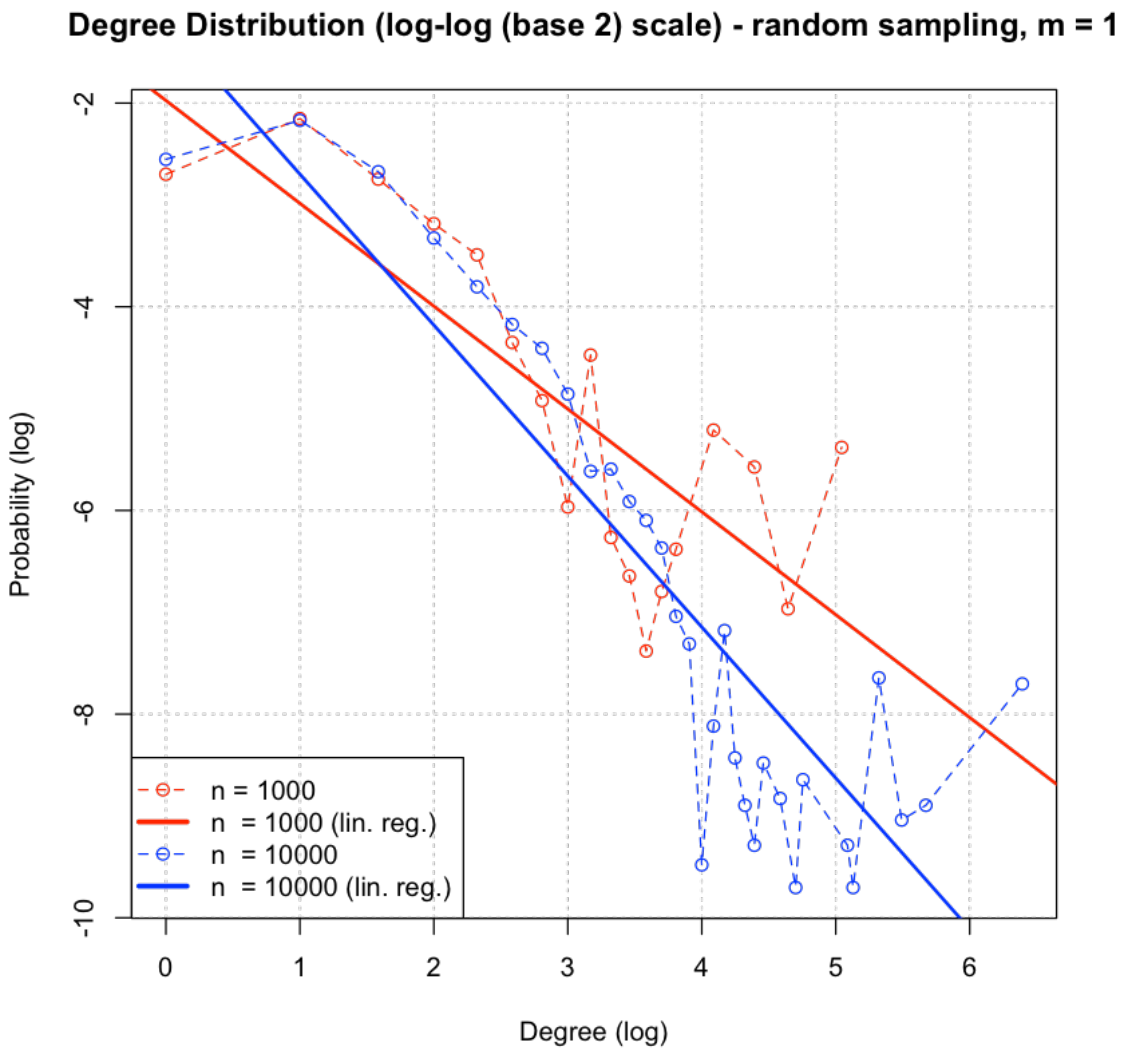
1.9 Question 2e

[23]:
```
deg_n = matrix(data=0.0,nrow=1000,ncol=1)
for (i in 1:1000){
    i_n = sample(1000,1)
    i_j = neighbors(g,i_n)
    if(length(i_j)>1){
        i_j = sample(i_j,1)
    }
    deg_n[i] = degree(g,i_j)
}
dist_n = table(deg_n)
dist_n_1000 = log2(as.vector(dist_n)/1000)
deg_n_1000 = log2(as.numeric(names(dist_n)))

deg_n2 = matrix(data=0.0,nrow=10000,ncol=1)
for (i in 1:10000){
    i_n = sample(10000,1)
    i_j = neighbors(g2,i_n)
    if(length(i_j)>1){
        i_j = sample(i_j,1)
```

```
    }
    deg_n2[i] = degree(g2,i_j)
}
dist_n2 = table(deg_n2)
dist_n_10000 = log2(as.vector(dist_n2)/10000)
deg_n_10000 = log2(as.numeric(names(dist_n2)))
```

[24]:
```
plot(deg_n_1000,dist_n_1000,main="Degree Distribution (log-log (base 2) scale)␣
↪- random sampling, m = 1",
     xlab="Degree (log)",ylab="Probability␣
↪(log)",grid(),col="red",xlim=c(0,max(deg_n_10000)),ylim=c(min(dist_n_10000),max(dist_n_1000
lines(deg_n_1000,dist_n_1000,lty=2,col="red")
abline(lm(dist_n_1000 ~ deg_n_1000),col="red",lwd=2)
points(deg_n_10000,dist_n_10000,col="blue")
lines(deg_n_10000,dist_n_10000,lty=2,col="blue")
abline(lm(dist_n_10000 ~ deg_n_10000),col="blue",lwd=2)
legend('bottomleft', legend = c("n = 1000", "n  = 1000 (lin. reg.)","n  =␣
↪10000", "n  = 10000 (lin. reg.)"),
       lty = c(2, 1, 2,1), lwd = c(1,3,1,3), pch=c(1,NA,1,NA),
       col = c('red','red','blue','blue'))
dev.copy2eps(file='Q2e.eps')
```

pdf: 2



[25]:
```
print("Slope and intercept for n = 1000:")
print(lm(dist_n_1000 ~ deg_n_1000))
print("Slope and intercept for n = 10000:")
print(lm(dist_n_10000 ~ deg_n_10000))
```

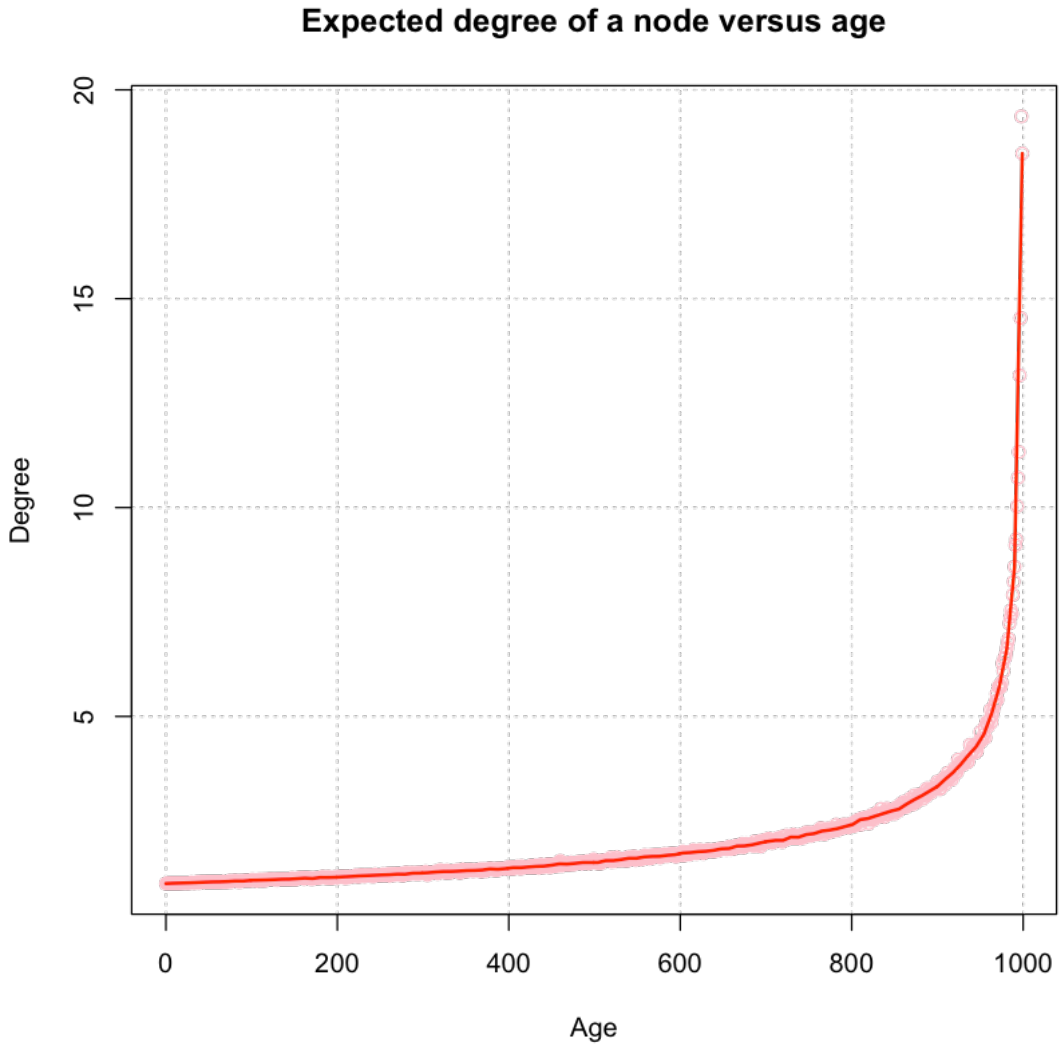
```
[1] "Slope and intercept for n = 1000:"

Call:
lm(formula = dist_n_1000 ~ deg_n_1000)

Coefficients:
(Intercept)   deg_n_1000
     -1.974       -1.010
```

```
[1] "Slope and intercept for n = 10000:"

Call:
lm(formula = dist_n_10000 ~ deg_n_10000)

Coefficients:
(Intercept)  deg_n_10000
     -1.215       -1.483
```

1.10 Question 2f

[26]:
```
deg_age = matrix(data=0.0,nrow = 1000,ncol=1)
for (i in 1:1000) {
    g <- barabasi.game(1000, m=1, directed=FALSE)
    deg_age = deg_age+degree(g)
}
deg_age = deg_age/1000
plot(seq(999,0,-1),deg_age,col='pink',ylab="Degree",xlab="Age",main="Expected␣
 ↪degree of a node versus age",grid())
lines(lowess(seq(999,0,-1),deg_age,f = 0.01), col="red",lwd = 2)
dev.copy2eps(file='Q2f.eps')
```

pdf: 2



1.11 Question 2g

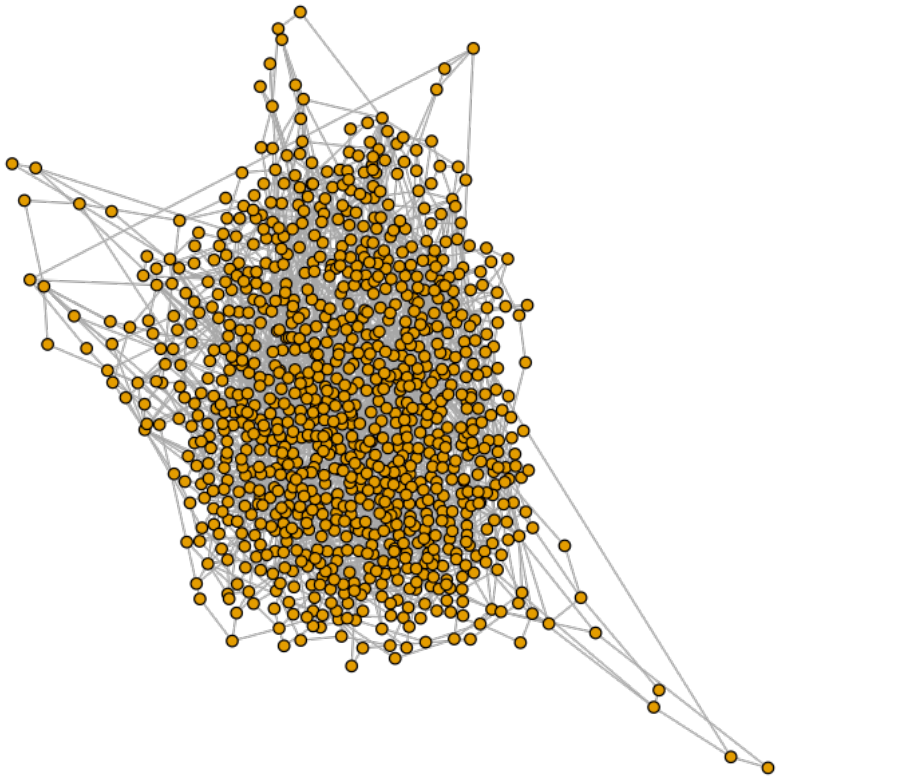
[27]:
```
count = 0
for (i in 1:100) {
    g <- barabasi.game(1000, m=2, directed=FALSE)
    if (is.connected(g)) {
        count <- count + 1
    }
}
print(sprintf("Total iterations: 100, Connected graphs: %d",count))
plot(g,vertex.label="",vertex.size=3,main = "Undirected network, preferential␣
 ↪attachment (UNPA), n = 1000, m = 2")
```

```
dev.copy2eps(file='Q2ga.eps')

count = 0
for (i in 1:100) {
    g2 <- barabasi.game(1000, m=5, directed=FALSE)
    if (is.connected(g2)) {
        count <- count + 1
    }
}
print(sprintf("Total iterations: 100, Connected graphs: %d",count))
plot(g2,vertex.label="",vertex.size=3,main = "Undirected network, preferential␣
 ↪attachment (UNPA), n = 1000, m = 5")
dev.copy2eps(file='Q2gb.eps')
```

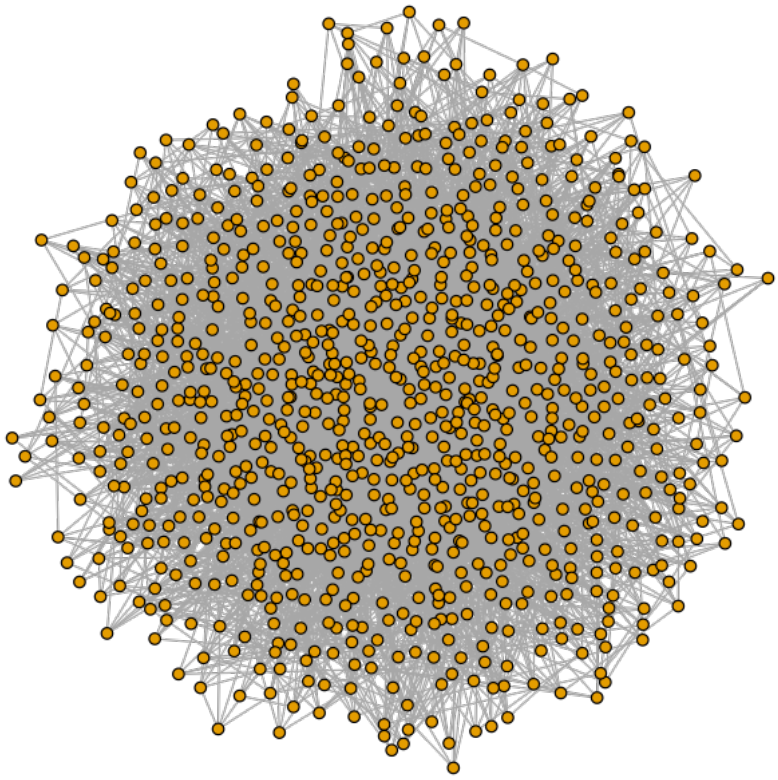
```
[1] "Total iterations: 100, Connected graphs: 100"
```

pdf: 2

```
[1] "Total iterations: 100, Connected graphs: 100"
```


Undirected network, preferential attachment (UNPA), n = 1000, m = 2

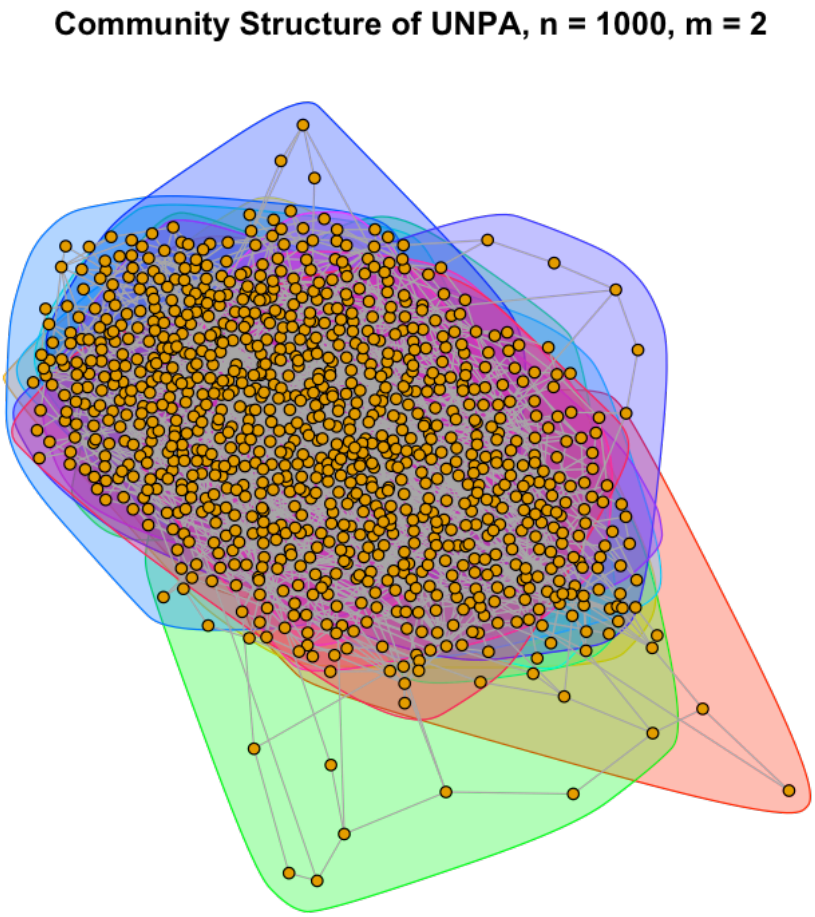
pdf: 2



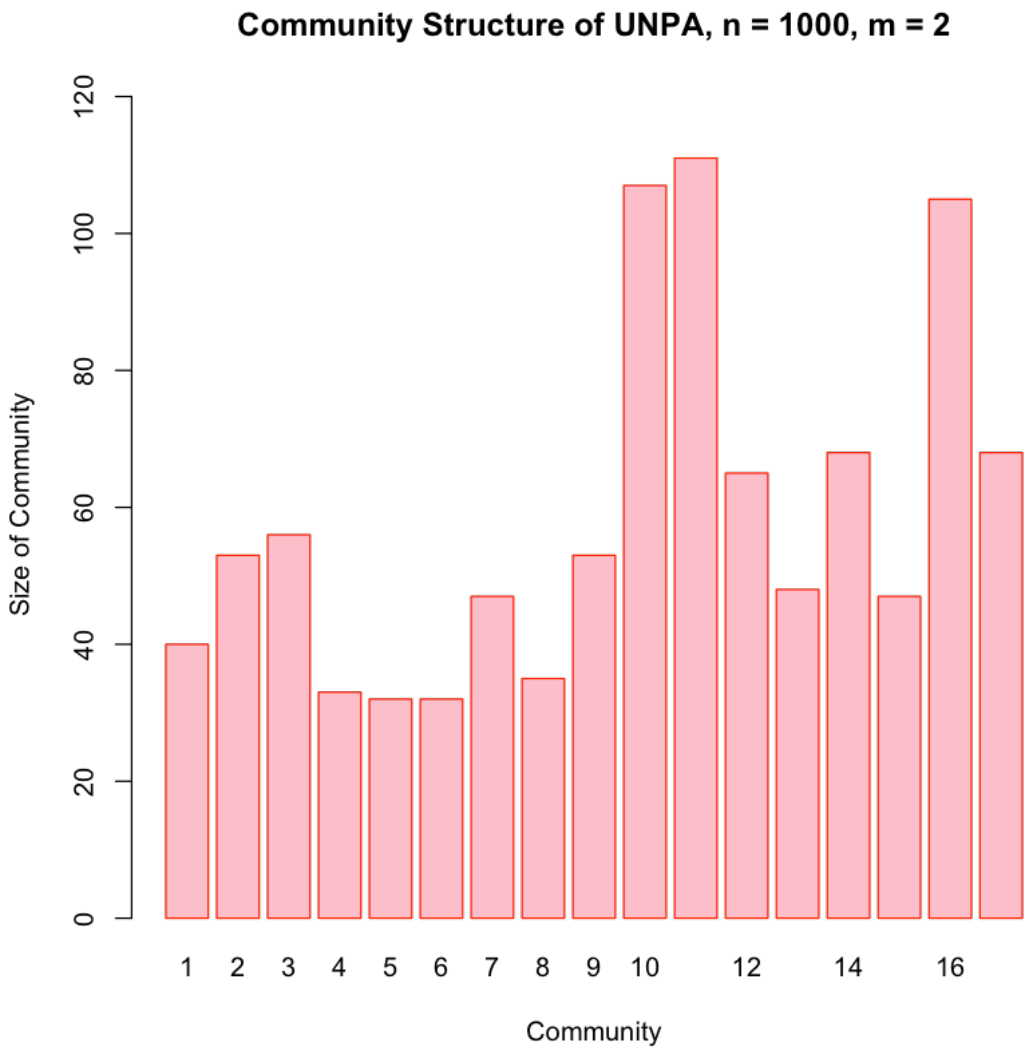
[28]:
```
comm_struct <- cluster_fast_greedy(g)
mod <- modularity(comm_struct)
print(sprintf("Modularity: %f",mod))
plot(g, mark.groups = groups(comm_struct), vertex.size=3, vertex.
 ↪label="",main="Community Structure of UNPA, n = 1000, m = 2")
dev.copy2pdf(file='Q2gc.pdf')
barplot(as.vector(sizes(comm_struct)),names.arg =␣
 ↪seq(1,length(comm_struct),1),main="Community Structure of UNPA, n = 1000, m␣
 ↪= 2",
   xlab="Community",ylab="Size of Community",ylim=c(0,max(as.
 ↪vector(sizes(comm_struct)))+10),border="red",col="pink")
dev.copy2eps(file='Q2gd.eps')
```

```
[1] "Modularity: 0.520994"
```

pdf: 2



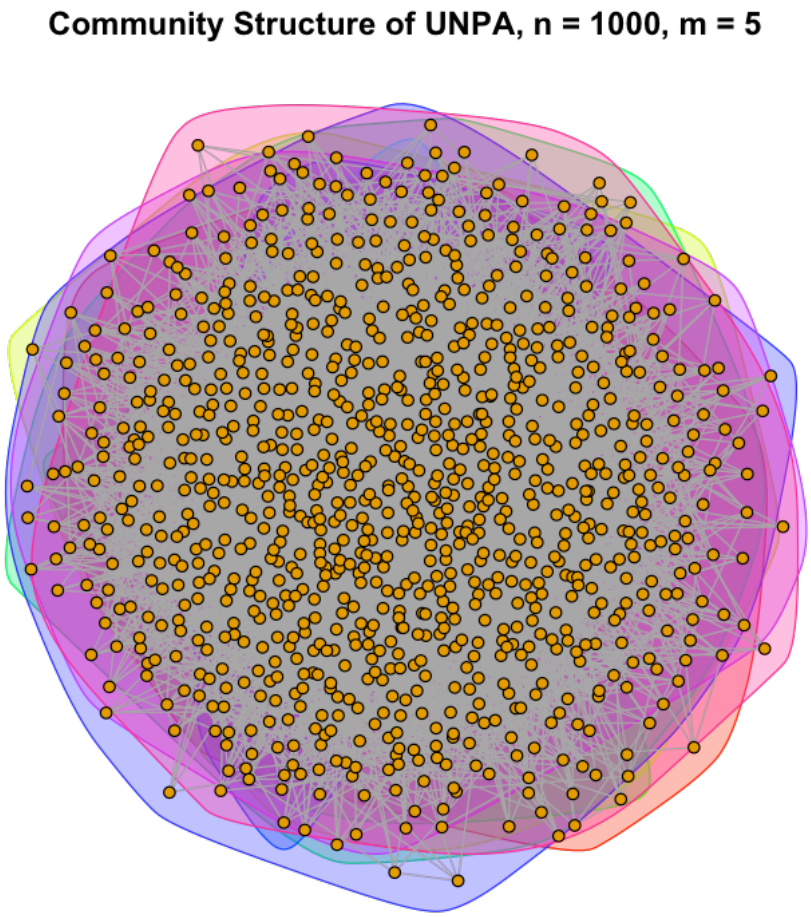
pdf: 2



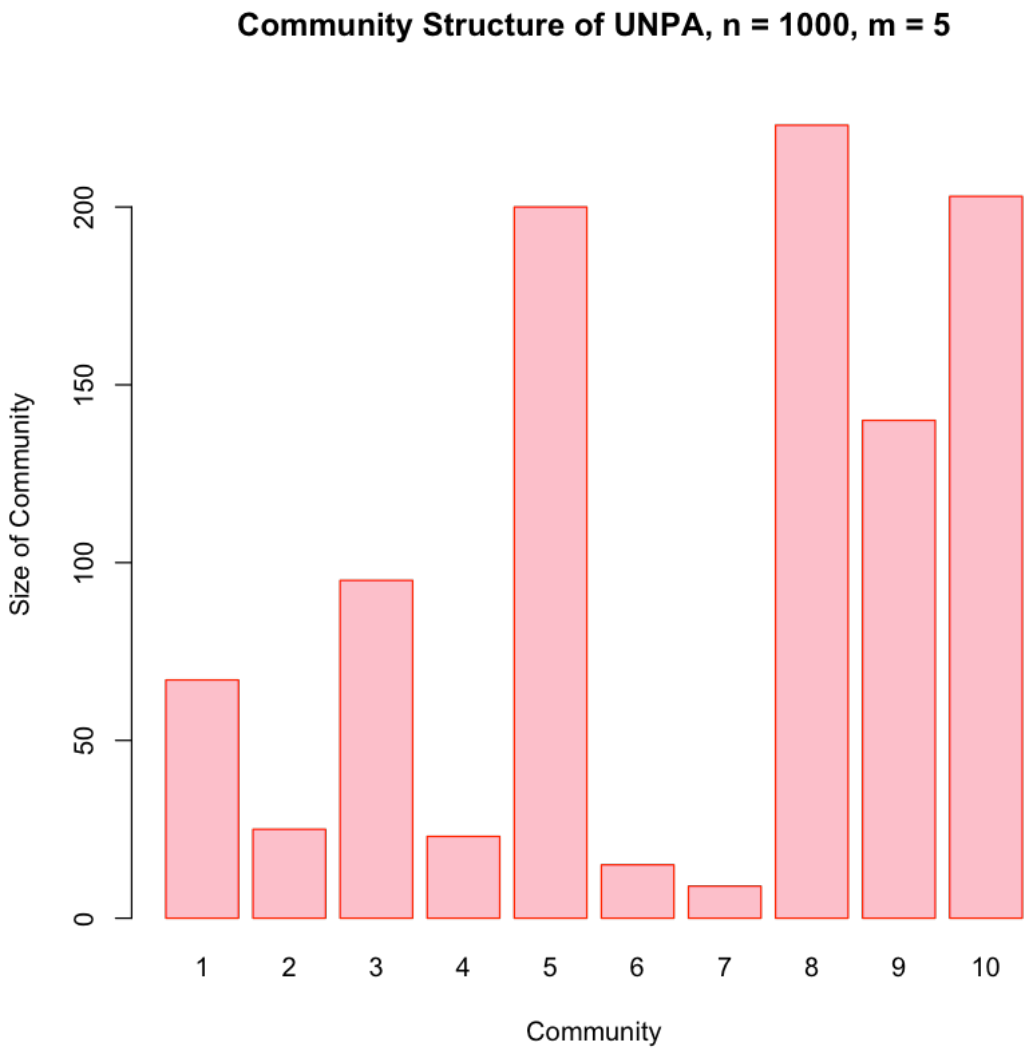
[29]:
```
comm_struct <- cluster_fast_greedy(g2)
mod <- modularity(comm_struct)
print(sprintf("Modularity: %f",mod))
plot(g2, mark.groups = groups(comm_struct), vertex.size=3, vertex.
 ↪label="",main="Community Structure of UNPA, n = 1000, m = 5")
dev.copy2pdf(file='Q2ge.pdf')
barplot(as.vector(sizes(comm_struct)),names.arg =␣
 ↪seq(1,length(comm_struct),1),main="Community Structure of UNPA, n = 1000, m␣
 ↪= 5",
   xlab="Community",ylab="Size of Community",ylim=c(0,max(as.
 ↪vector(sizes(comm_struct)))+10),border="red",col="pink")
dev.copy2eps(file='Q2gf.eps')
```

```
[1] "Modularity: 0.277776"
```

pdf: 2

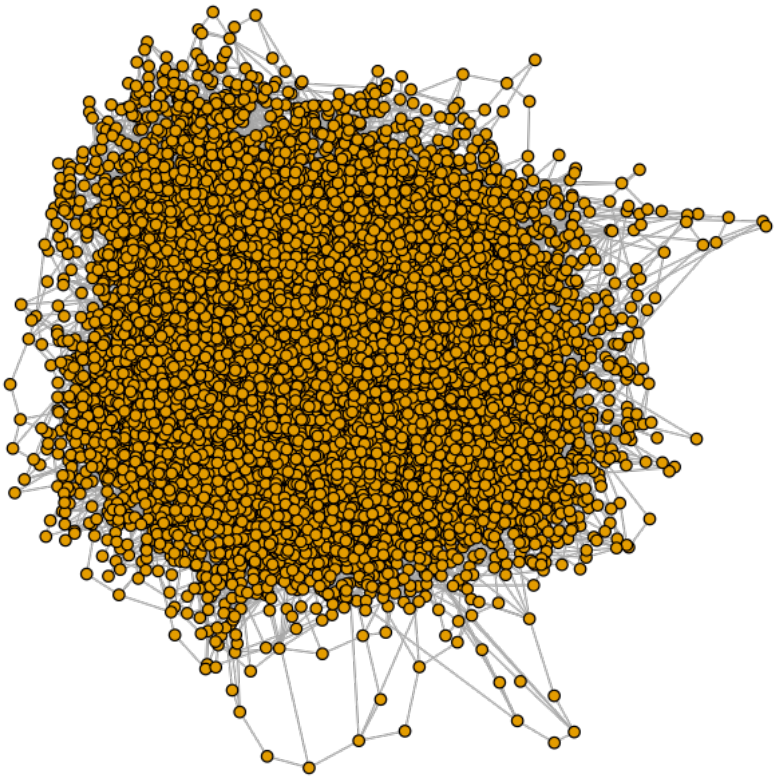


pdf: 2

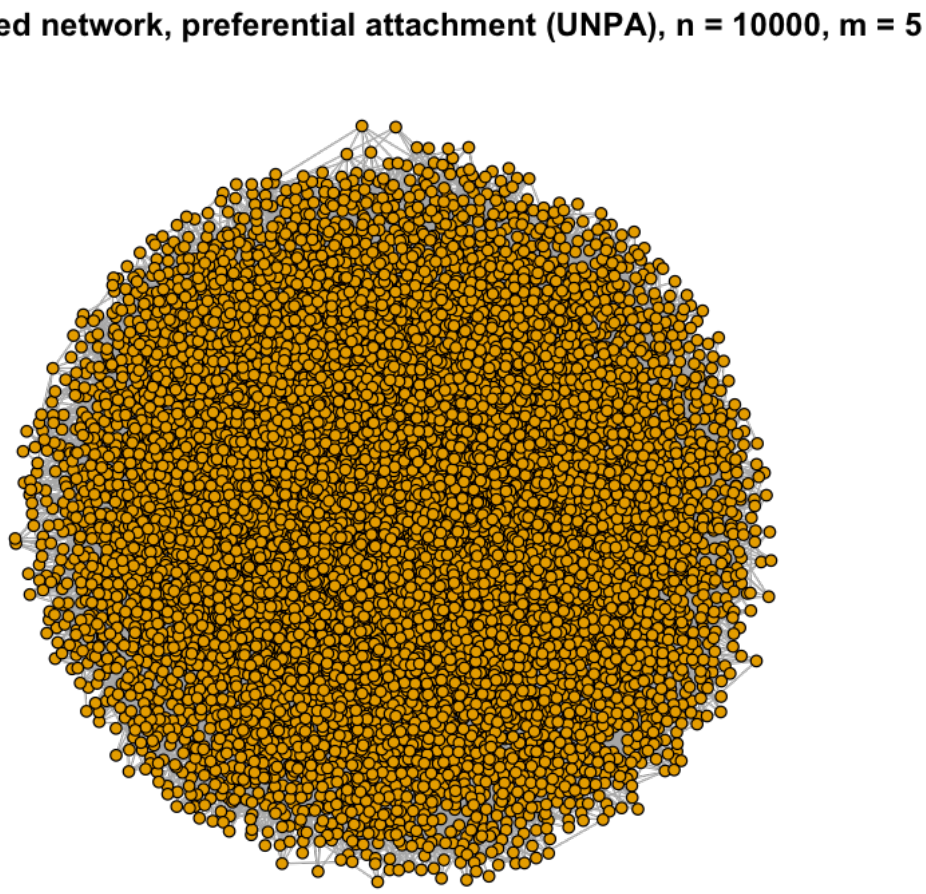


[30]:
```
g3 <- barabasi.game(10000, m=2, directed=FALSE)
plot(g3,vertex.label="",vertex.size=3,main = "Undirected network, preferential␣
 ↪attachment (UNPA), n = 10000, m = 2")
dev.copy2eps(file='Q2gg.eps')

g4 <- barabasi.game(10000, m=5, directed=FALSE)
plot(g4,vertex.label="",vertex.size=3,main = "Undirected network, preferential␣
 ↪attachment (UNPA), n = 10000, m = 5")
dev.copy2eps(file='Q2gh.eps')
```

pdf: 2


Undirected network, preferential attachment (UNPA), n = 10000, m = 2

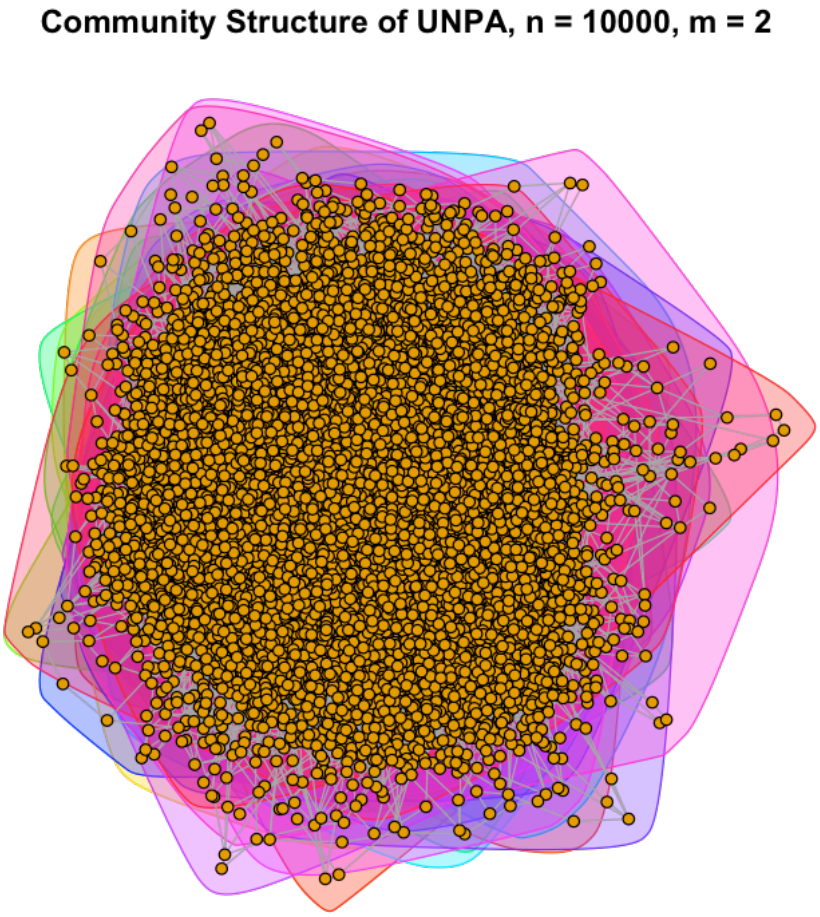
pdf: 2



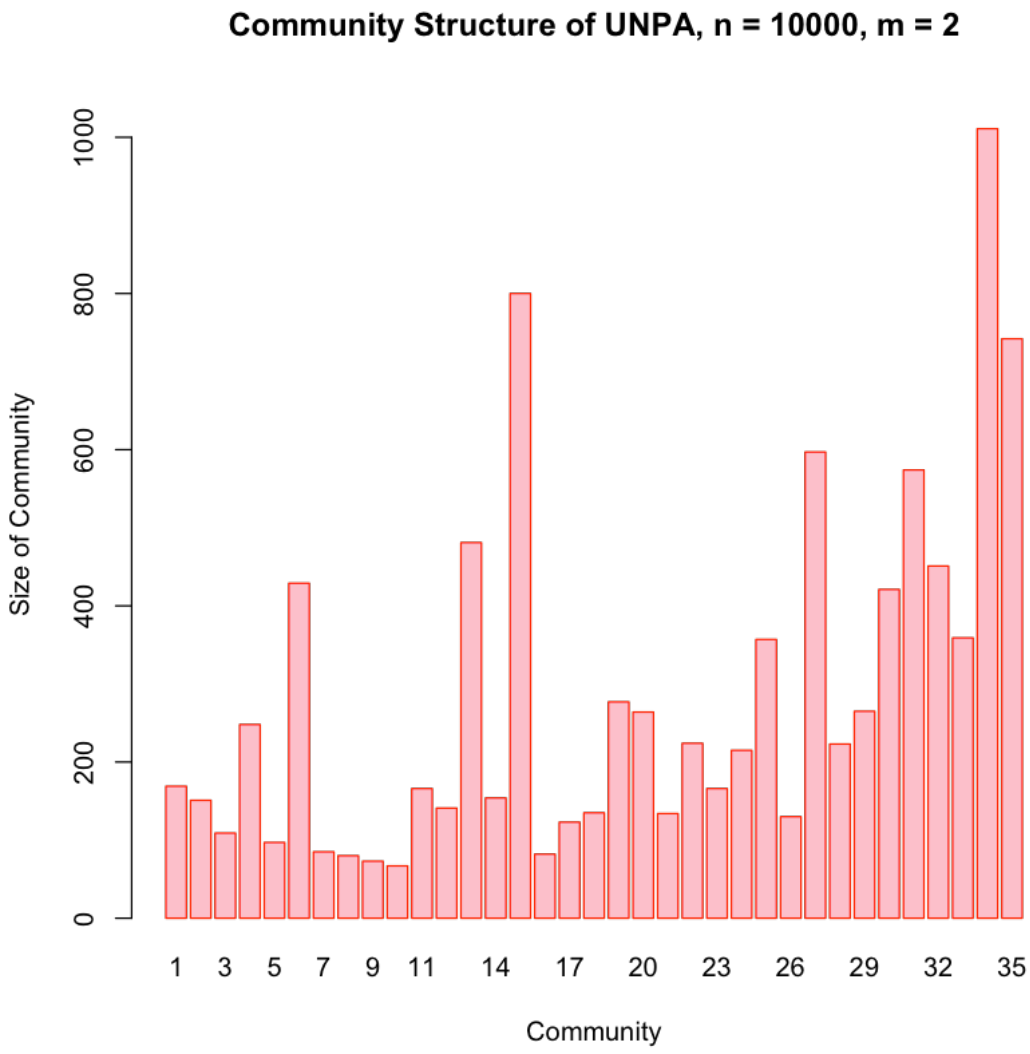
[31]:
```
comm_struct3 <- cluster_fast_greedy(g3)
mod3 <- modularity(comm_struct3)
print(sprintf("Modularity: %f",mod3))
plot(g3, mark.groups = groups(comm_struct3), vertex.size=3, vertex.
 ↪label="",main="Community Structure of UNPA, n = 10000, m = 2")
dev.copy2pdf(file='Q2gi.pdf')

barplot(as.vector(sizes(comm_struct3)),names.arg =␣
 ↪seq(1,length(comm_struct3),1),main="Community Structure of UNPA, n = 10000,␣
 ↪m = 2",
   xlab="Community",ylab="Size of Community",ylim=c(0,max(as.
 ↪vector(sizes(comm_struct3)))+50),border="red",col="pink")
dev.copy2eps(file='Q2gj.eps')
```

```
[1] "Modularity: 0.533746"
```

pdf: 2



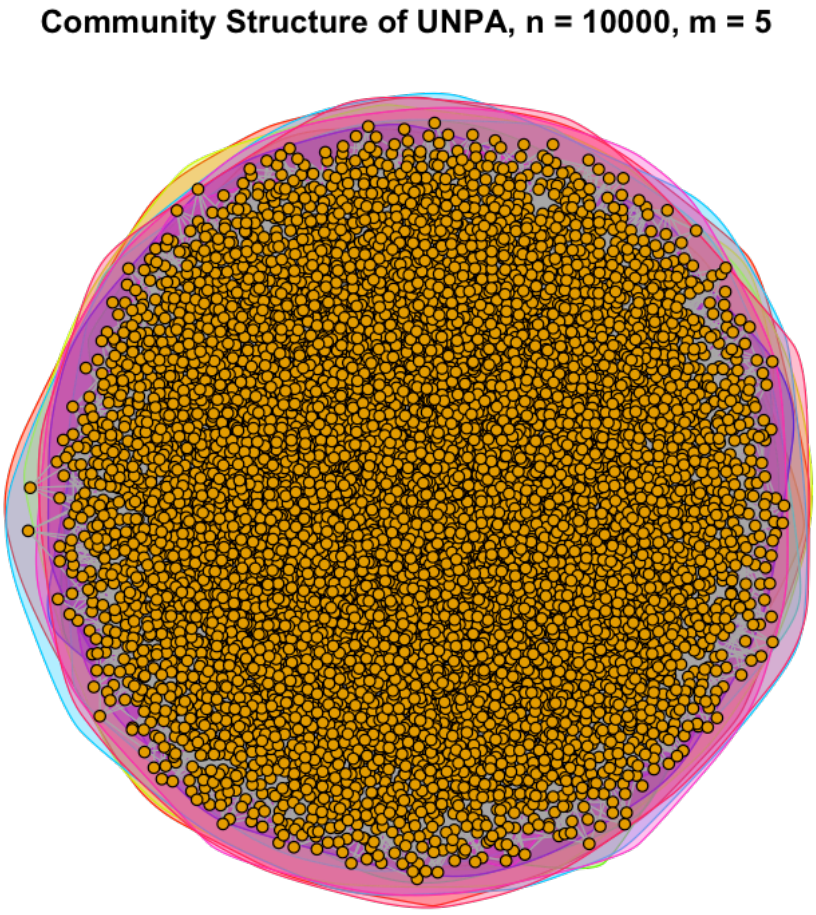
pdf: 2



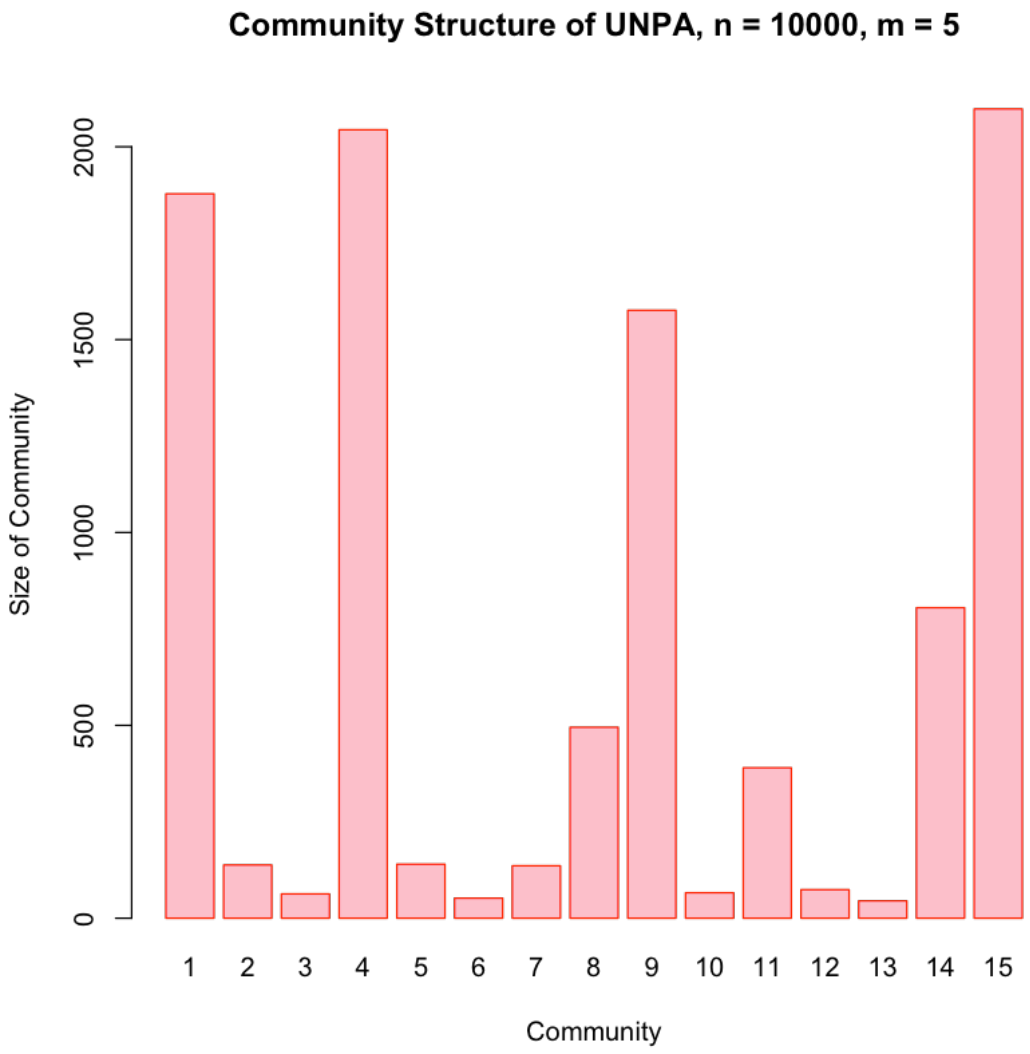
[32]:
```
comm_struct4 <- cluster_fast_greedy(g4)
mod4 <- modularity(comm_struct4)
print(sprintf("Modularity: %f",mod4))
plot(g4, mark.groups = groups(comm_struct4), vertex.size=3, vertex.
 ↪label="",main="Community Structure of UNPA, n = 10000, m = 5")
dev.copy2pdf(file='Q2gk.pdf')

barplot(as.vector(sizes(comm_struct4)),names.arg =␣
 ↪seq(1,length(comm_struct4),1),main="Community Structure of UNPA, n = 10000,␣
 ↪m = 5",
   xlab="Community",ylab="Size of Community",ylim=c(0,max(as.
 ↪vector(sizes(comm_struct4)))+50),border="red",col="pink")
dev.copy2eps(file='Q2gl.eps')
```

```
[1] "Modularity: 0.271406"
```

pdf: 2



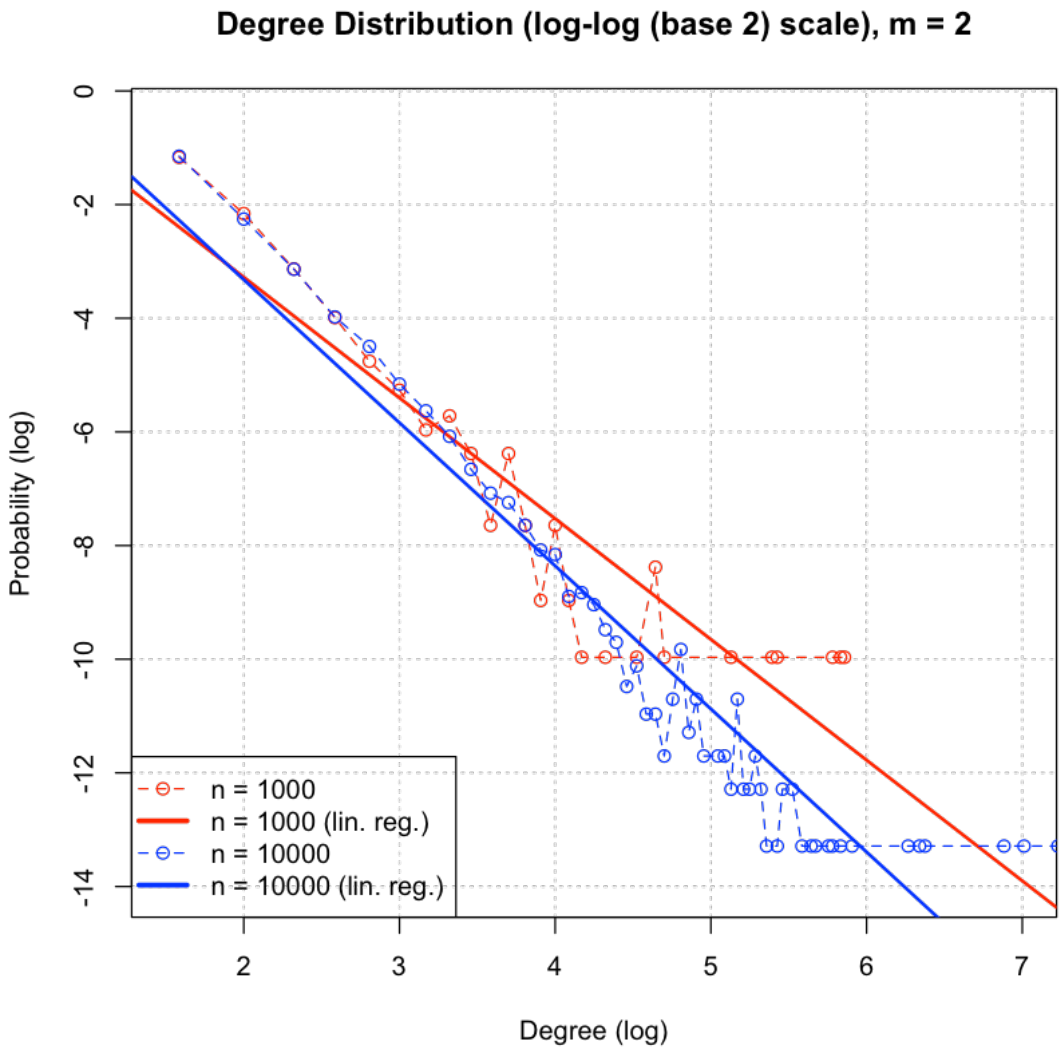
pdf: 2



[33]:
```
deg_dist_1 = degree.distribution(g)
deg_1 <- log2(c(1:length(deg_dist_1)))[which(deg_dist_1 !=0, arr.ind = TRUE)]
dist_1 <- log2(deg_dist_1)[which(deg_dist_1 !=0, arr.ind = TRUE)]
plot(deg_1,dist_1,main="Degree Distribution (log-log (base 2) scale), m = 2",
     xlab="Degree (log)",ylab="Probability␣
 ↪(log)",grid(),col="red",ylim=c(-14,-0.5),xlim=c(1.5,7))
lines(deg_1,dist_1,lty=2,col="red")
abline(lm(dist_1 ~ deg_1),col="red",lwd=2)

deg_dist_2 = degree.distribution(g3)
deg_2 <- log2(c(1:length(deg_dist_2)))[which(deg_dist_2 !=0, arr.ind = TRUE)]
dist_2 <- log2(deg_dist_2)[which(deg_dist_2 !=0, arr.ind = TRUE)]
points(deg_2,dist_2,col="blue")
```

```
lines(deg_2,dist_2,lty=2,col="blue")
abline(lm(dist_2 ~ deg_2),col="blue",lwd=2)
legend('bottomleft', legend = c("n = 1000", "n = 1000 (lin. reg.)","n = 10000",␣
 ↪"n = 10000 (lin. reg.)"),
       lty = c(2, 1, 2,1), lwd = c(1,3,1,3), pch=c(1,NA,1,NA),
       col = c('red','red','blue','blue'))
dev.copy2eps(file='Q2gm.eps')
```

pdf: 2



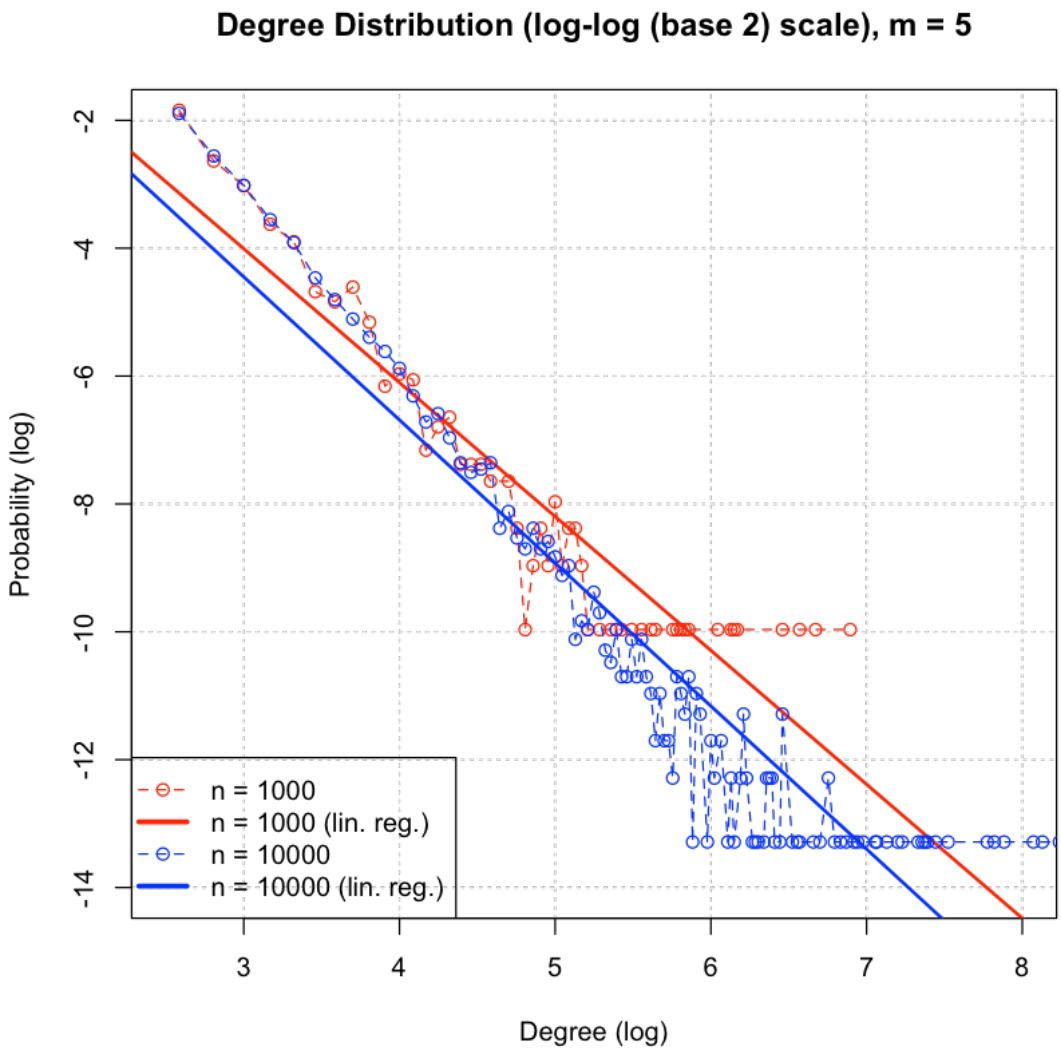
[34]:
```
print("Slope and intercept for n = 1000:")
print(lm(dist_1 ~ deg_1))
print("Slope and intercept for n = 10000:")
```

```
print(lm(dist_2 ~ deg_2))
```

```
[1] "Slope and intercept for n = 1000:"

Call:
lm(formula = dist_1 ~ deg_1)

Coefficients:
(Intercept)        deg_1
     0.9717      -2.1246

[1] "Slope and intercept for n = 10000:"

Call:
lm(formula = dist_2 ~ deg_2)

Coefficients:
(Intercept)        deg_2
      1.715       -2.518
```

[35]:
```
deg_dist_1 = degree.distribution(g2)
deg_1 <- log2(c(1:length(deg_dist_1)))[which(deg_dist_1 !=0, arr.ind = TRUE)]
dist_1 <- log2(deg_dist_1)[which(deg_dist_1 !=0, arr.ind = TRUE)]
plot(deg_1,dist_1,main="Degree Distribution (log-log (base 2) scale), m = 5",
     xlab="Degree (log)",ylab="Probability␣
 ↪(log)",grid(),col="red",ylim=c(-14,-2),xlim=c(2.5,8))
lines(deg_1,dist_1,lty=2,col="red")
abline(lm(dist_1 ~ deg_1),col="red",lwd=2)

deg_dist_2 = degree.distribution(g4)
deg_2 <- log2(c(1:length(deg_dist_2)))[which(deg_dist_2 !=0, arr.ind = TRUE)]
dist_2 <- log2(deg_dist_2)[which(deg_dist_2 !=0, arr.ind = TRUE)]
points(deg_2,dist_2,col="blue")
lines(deg_2,dist_2,lty=2,col="blue")
abline(lm(dist_2 ~ deg_2),col="blue",lwd=2)
legend('bottomleft', legend = c("n = 1000", "n = 1000 (lin. reg.)","n = 10000",␣
 ↪"n = 10000 (lin. reg.)"),
       lty = c(2, 1, 2,1), lwd = c(1,3,1,3), pch=c(1,NA,1,NA),
       col = c('red','red','blue','blue'))
dev.copy2eps(file='Q2gn.eps')
```

pdf: 2



[36]:
```
print("Slope and intercept for n = 1000:")
print(lm(dist_1 ~ deg_1))
print("Slope and intercept for n = 10000:")
print(lm(dist_2 ~ deg_2))
```

```
[1] "Slope and intercept for n = 1000:"

Call:
lm(formula = dist_1 ~ deg_1)

Coefficients:
(Intercept)        deg_1
      2.278       -2.095
```

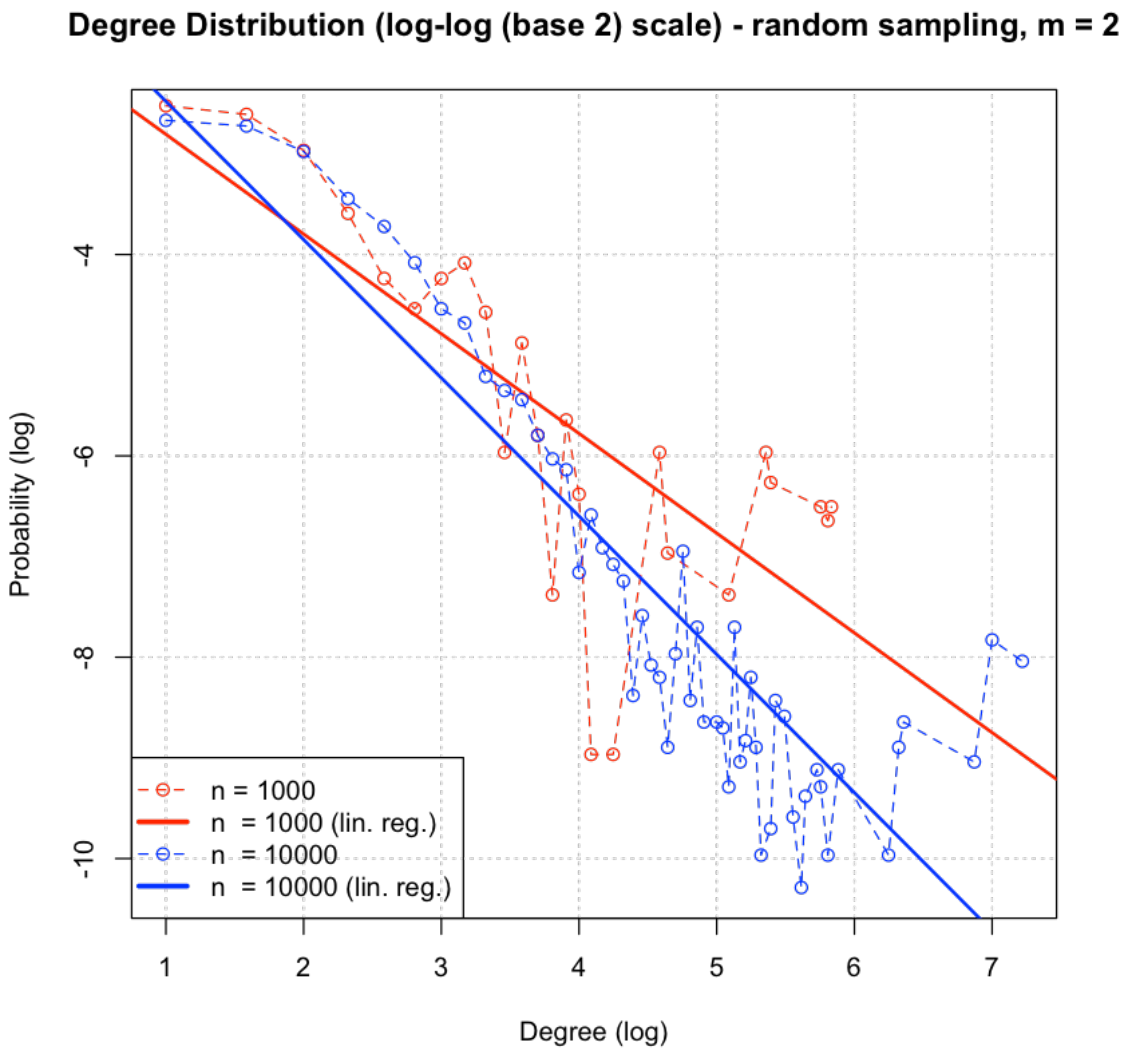
```
[1] "Slope and intercept for n = 10000:"

Call:
lm(formula = dist_2 ~ deg_2)

Coefficients:
(Intercept)        deg_2
      2.264       -2.238
```

[37]:
```
deg_n = matrix(data=0.0,nrow=1000,ncol=1)
for (i in 1:1000){
    i_n = sample(1000,1)
    i_j = neighbors(g,i_n)
    if(length(i_j)>1){
        i_j = sample(i_j,1)
    }
    deg_n[i] = degree(g,i_j)
}
dist_n = table(deg_n)
dist_n_1000 = log2(as.vector(dist_n)/1000)
deg_n_1000 = log2(as.numeric(names(dist_n)))

deg_n2 = matrix(data=0.0,nrow=10000,ncol=1)
for (i in 1:10000){
    i_n = sample(10000,1)
    i_j = neighbors(g3,i_n)
    if(length(i_j)>1){
        i_j = sample(i_j,1)
    }
    deg_n2[i] = degree(g3,i_j)
}
dist_n2 = table(deg_n2)
dist_n_10000 = log2(as.vector(dist_n2)/10000)
deg_n_10000 = log2(as.numeric(names(dist_n2)))

plot(deg_n_1000,dist_n_1000,main="Degree Distribution (log-log (base 2) scale)␣
↪- random sampling, m = 2",
     xlab="Degree (log)",ylab="Probability␣
↪(log)",grid(),col="red",xlim=c(1,max(deg_n_10000)),ylim=c(min(dist_n_10000),max(dist_n_1000
lines(deg_n_1000,dist_n_1000,lty=2,col="red")
abline(lm(dist_n_1000 ~ deg_n_1000),col="red",lwd=2)
points(deg_n_10000,dist_n_10000,col="blue")
lines(deg_n_10000,dist_n_10000,lty=2,col="blue")
abline(lm(dist_n_10000 ~ deg_n_10000),col="blue",lwd=2)
```

```
legend('bottomleft', legend = c("n = 1000", "n  = 1000 (lin. reg.)","n  =␣
 ↪10000", "n  = 10000 (lin. reg.)"),
       lty = c(2, 1, 2,1), lwd = c(1,3,1,3), pch=c(1,NA,1,NA),
       col = c('red','red','blue','blue'))
dev.copy2eps(file='Q2go.eps')
print("Slope and intercept for n = 1000:")
print(lm(dist_n_1000 ~ deg_n_1000))
print("Slope and intercept for n = 10000:")
print(lm(dist_n_10000 ~ deg_n_10000))
```

pdf: 2

```
[1] "Slope and intercept for n = 1000:"

Call:
lm(formula = dist_n_1000 ~ deg_n_1000)

Coefficients:
(Intercept)   deg_n_1000
    -1.8160      -0.9904

[1] "Slope and intercept for n = 10000:"

Call:
lm(formula = dist_n_10000 ~ deg_n_10000)

Coefficients:
(Intercept)  deg_n_10000
     -1.107       -1.373
```



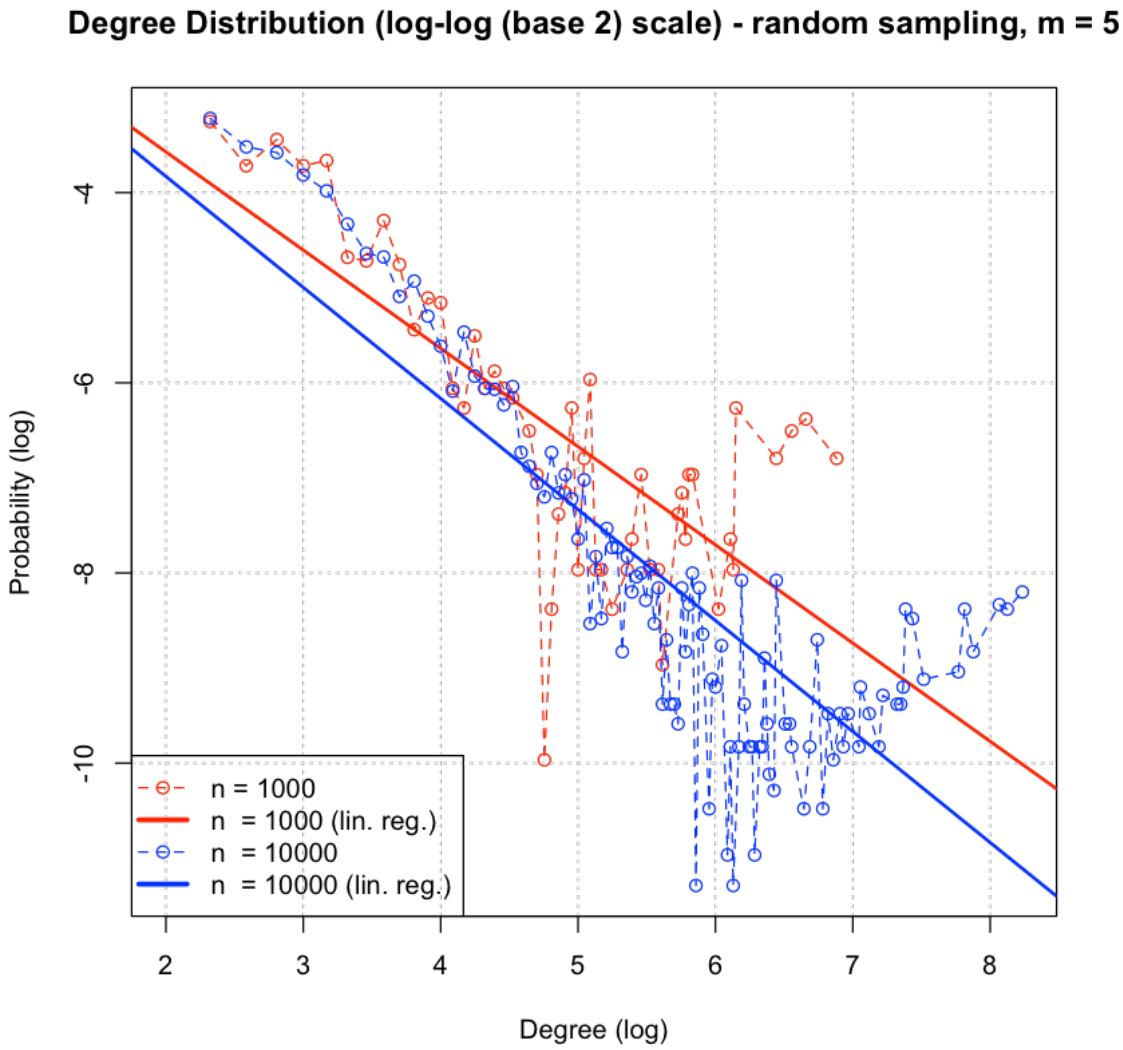
[38]:
```
deg_n = matrix(data=0.0,nrow=1000,ncol=1)
for (i in 1:1000){
    i_n = sample(1000,1)
    i_j = neighbors(g2,i_n)
    if(length(i_j)>1){
        i_j = sample(i_j,1)
    }
    deg_n[i] = degree(g2,i_j)
}
dist_n = table(deg_n)
dist_n_1000 = log2(as.vector(dist_n)/1000)
deg_n_1000 = log2(as.numeric(names(dist_n)))
```

```
deg_n2 = matrix(data=0.0,nrow=10000,ncol=1)
for (i in 1:10000){
    i_n = sample(10000,1)
    i_j = neighbors(g4,i_n)
    if(length(i_j)>1){
        i_j = sample(i_j,1)
    }
    deg_n2[i] = degree(g4,i_j)
}
dist_n2 = table(deg_n2)
dist_n_10000 = log2(as.vector(dist_n2)/10000)
deg_n_10000 = log2(as.numeric(names(dist_n2)))

plot(deg_n_1000,dist_n_1000,main="Degree Distribution (log-log (base 2) scale)␣
 ↪- random sampling, m = 5",
     xlab="Degree (log)",ylab="Probability␣
 ↪(log)",grid(),col="red",xlim=c(2,max(deg_n_10000)),ylim=c(min(dist_n_10000),max(dist_n_1000
lines(deg_n_1000,dist_n_1000,lty=2,col="red")
abline(lm(dist_n_1000 ~ deg_n_1000),col="red",lwd=2)
points(deg_n_10000,dist_n_10000,col="blue")
lines(deg_n_10000,dist_n_10000,lty=2,col="blue")
abline(lm(dist_n_10000 ~ deg_n_10000),col="blue",lwd=2)
legend('bottomleft', legend = c("n = 1000", "n  = 1000 (lin. reg.)","n  =␣
 ↪10000", "n  = 10000 (lin. reg.)"),
       lty = c(2, 1, 2,1), lwd = c(1,3,1,3), pch=c(1,NA,1,NA),
       col = c('red','red','blue','blue'))
dev.copy2eps(file='Q2gp.eps')
print("Slope and intercept for n = 1000:")
print(lm(dist_n_1000 ~ deg_n_1000))
print("Slope and intercept for n = 10000:")
print(lm(dist_n_10000 ~ deg_n_10000))
```

pdf: 2

```
[1] "Slope and intercept for n = 1000:"

Call:
lm(formula = dist_n_1000 ~ deg_n_1000)

Coefficients:
(Intercept)   deg_n_1000
     -1.504       -1.033

[1] "Slope and intercept for n = 10000:"

Call:
lm(formula = dist_n_10000 ~ deg_n_10000)
```

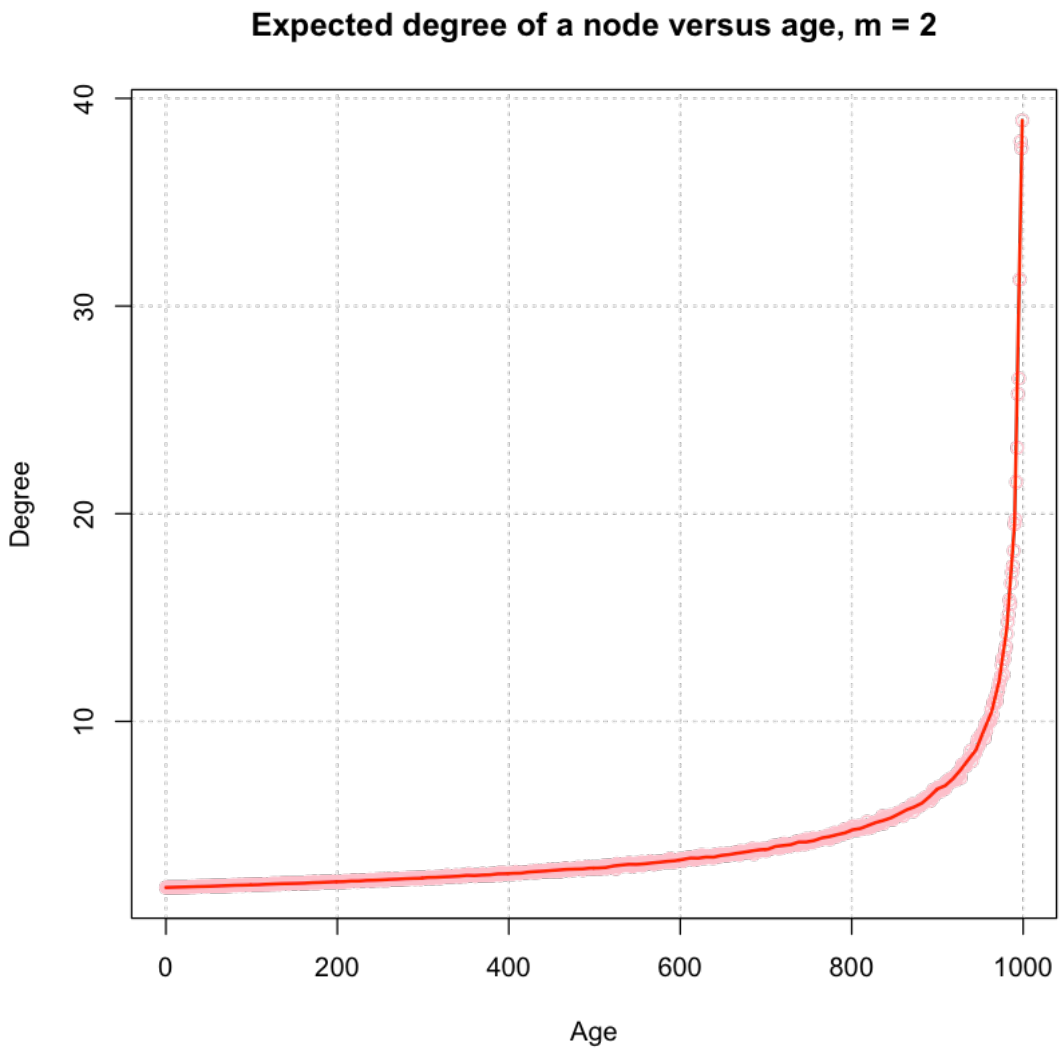
```
Coefficients:
(Intercept)  deg_n_10000
     -1.494       -1.168
```



[39]:
```
deg_age = matrix(data=0.0,nrow = 1000,ncol=1)
for (i in 1:1000) {
    g <- barabasi.game(1000, m=2, directed=FALSE)
    deg_age = deg_age+degree(g)
}
deg_age = deg_age/1000
plot(seq(999,0,-1),deg_age,col='pink',ylab="Degree",xlab="Age",main="Expected␣
 ↪degree of a node versus age, m = 2",grid())
```

```
lines(lowess(seq(999,0,-1),deg_age,f = 0.01), col="red",lwd = 2)
dev.copy2eps(file='Q2gq.eps')
```

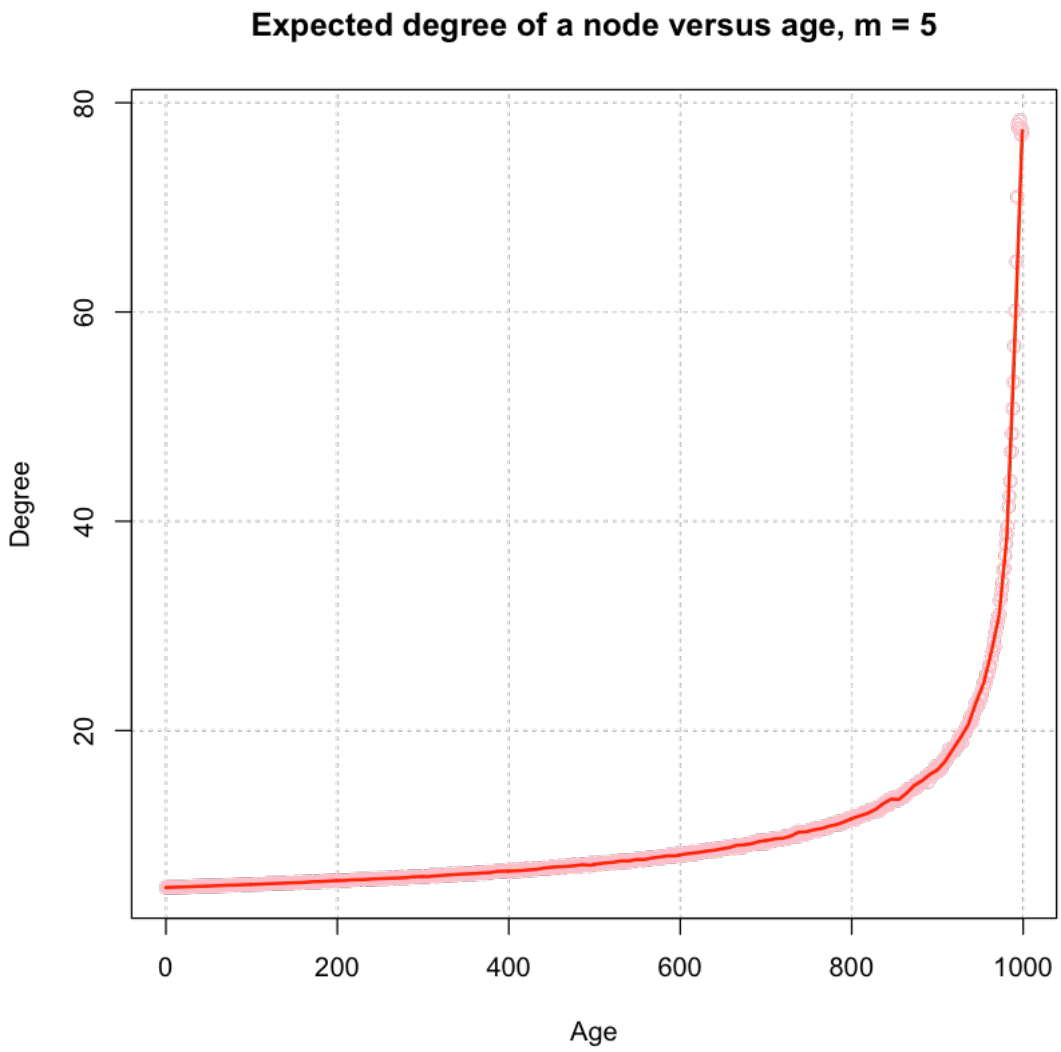
pdf: 2



[40]:
```
deg_age = matrix(data=0.0,nrow = 1000,ncol=1)
for (i in 1:1000) {
    g <- barabasi.game(1000, m=5, directed=FALSE)
    deg_age = deg_age+degree(g)
}
deg_age = deg_age/1000
plot(seq(999,0,-1),deg_age,col='pink',ylab="Degree",xlab="Age",main="Expected␣
 ↪degree of a node versus age, m = 5",grid())
```

```
lines(lowess(seq(999,0,-1),deg_age,f = 0.005), col="red",lwd = 2)
dev.copy2eps(file='Q2gr.eps')
```

pdf: 2



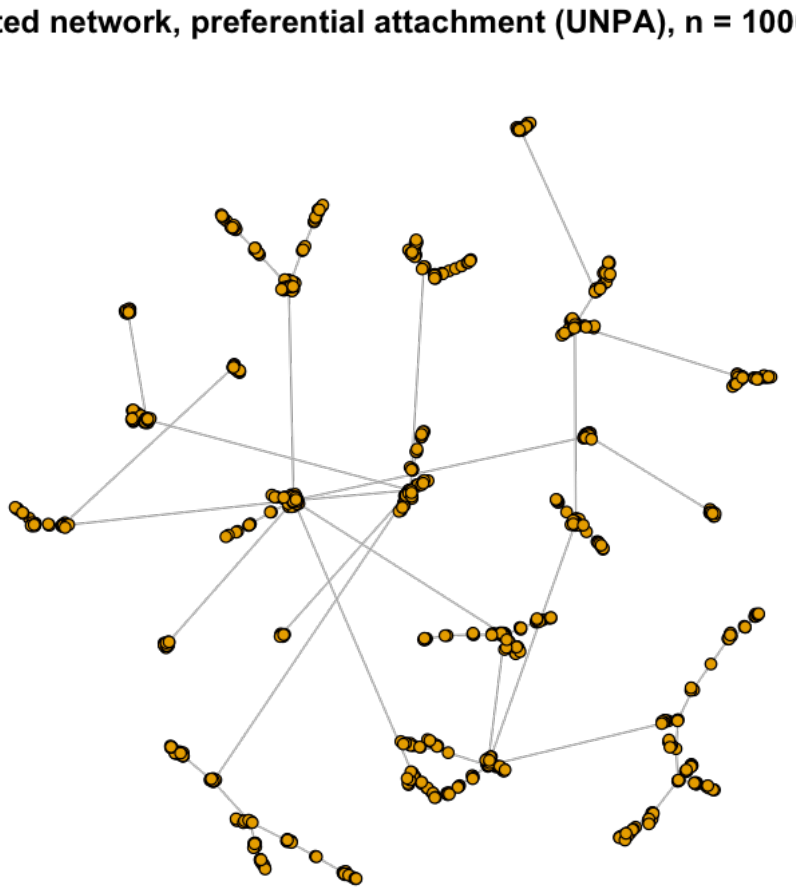
1.12 Question 2h

[41]:
```
g <- barabasi.game(1000, m=1, directed=FALSE)
comm_struct <- cluster_fast_greedy(g)
plot(g,vertex.label="",vertex.size=3,main = "Undirected network, preferential␣
 ↪attachment (UNPA), n = 1000, m = 1")
dev.copy2eps(file='Q2ha.eps')
mod <- modularity(comm_struct)
```

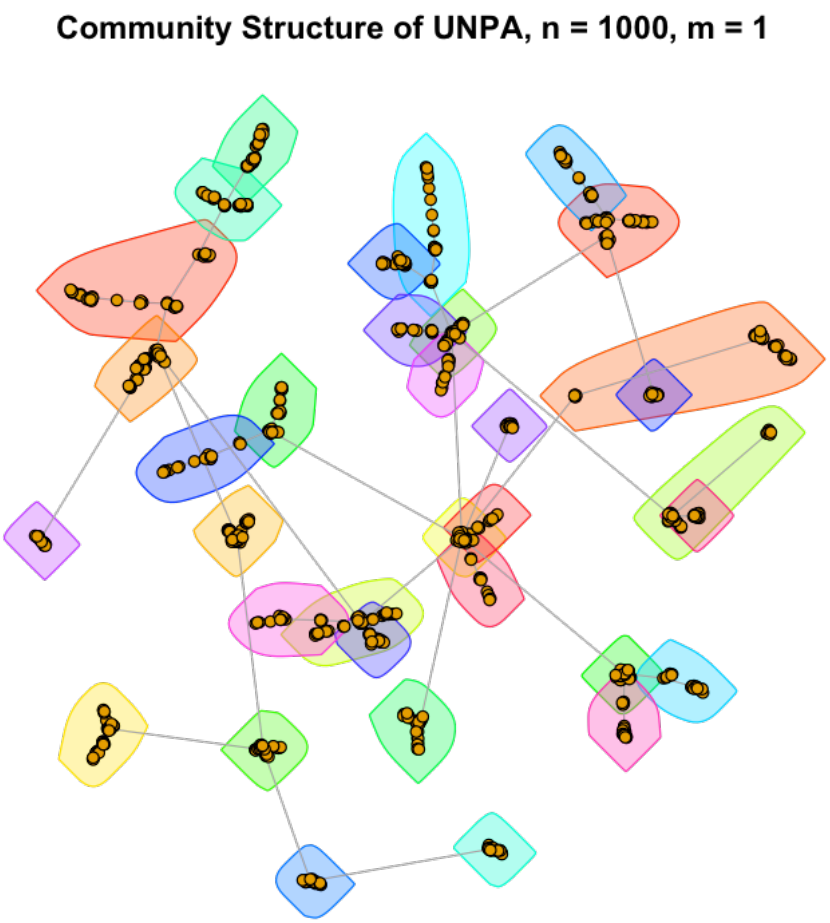
```
print(sprintf("Modularity: %f",mod))
plot(g, mark.groups = groups(comm_struct), vertex.size=3, vertex.
 ↪label="",main="Community Structure of UNPA, n = 1000, m = 1")
dev.copy2pdf(file='Q2hb.pdf')
barplot(as.vector(sizes(comm_struct)),names.arg =␣
 ↪seq(1,length(comm_struct),1),main="Community Structure of UNPA, n = 1000, m␣
 ↪= 1",
   xlab="Community",ylab="Size of Community",ylim=c(0,max(as.
 ↪vector(sizes(comm_struct)))+10),border="red",col="pink")
dev.copy2eps(file='Q2hc.eps')
```

pdf: 2

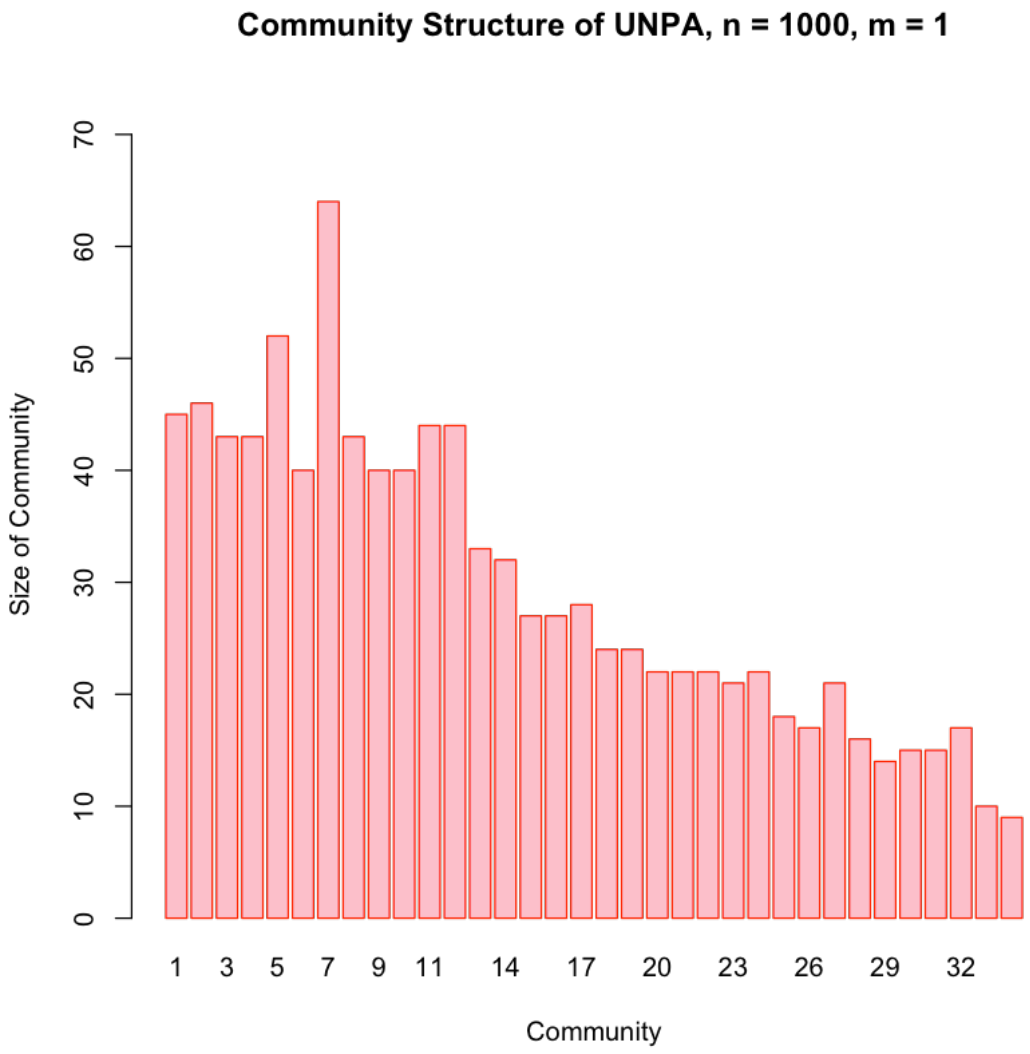
```
[1] "Modularity: 0.930920"
```



pdf: 2



pdf: 2

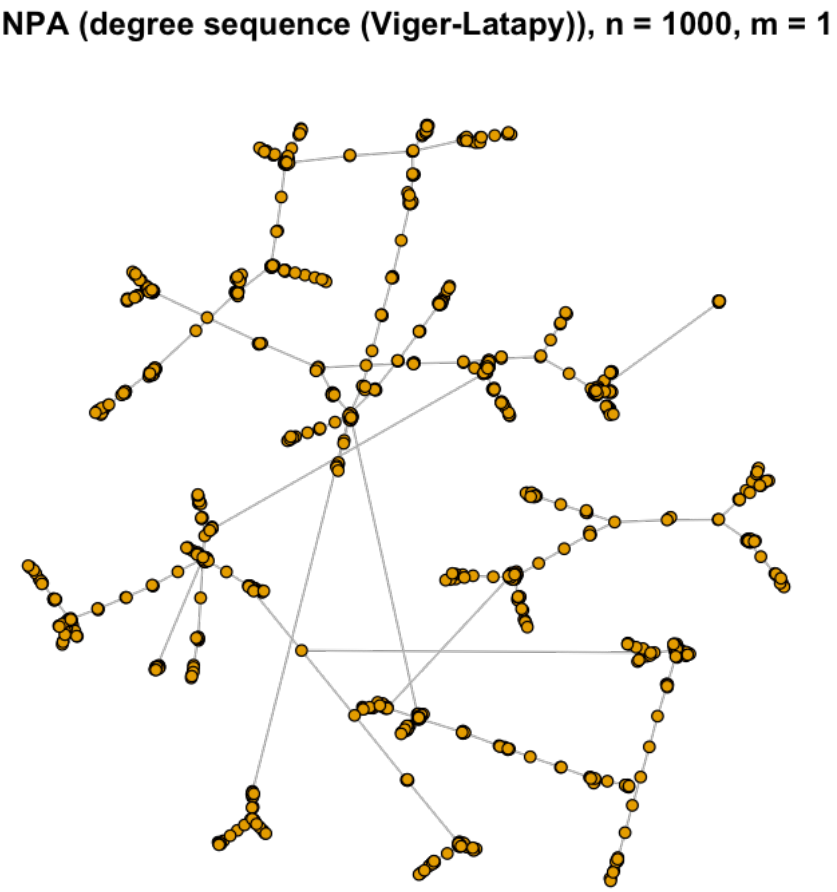


[42]:
```
g2 <- sample_degseq(degree(g),method = 'vl')
comm_struct2 <- cluster_fast_greedy(g2)
plot(g2,vertex.label="",vertex.size=3,main = "UNPA (degree sequence␣
 ↪(Viger-Latapy)), n = 1000, m = 1")
dev.copy2eps(file='Q2hd.eps')
mod <- modularity(comm_struct2)
print(sprintf("Modularity: %f",mod))
plot(g2, mark.groups = groups(comm_struct2), vertex.size=3, vertex.
 ↪label="",main="Community Structure of UNPA (Viger-Latapy), n = 1000, m = 1")
dev.copy2pdf(file='Q2he.pdf')
barplot(as.vector(sizes(comm_struct2)),names.arg =␣
 ↪seq(1,length(comm_struct2),1),main="Community Structure of UNPA␣
 ↪(Viger-Latapy), n = 1000, m = 1",
```

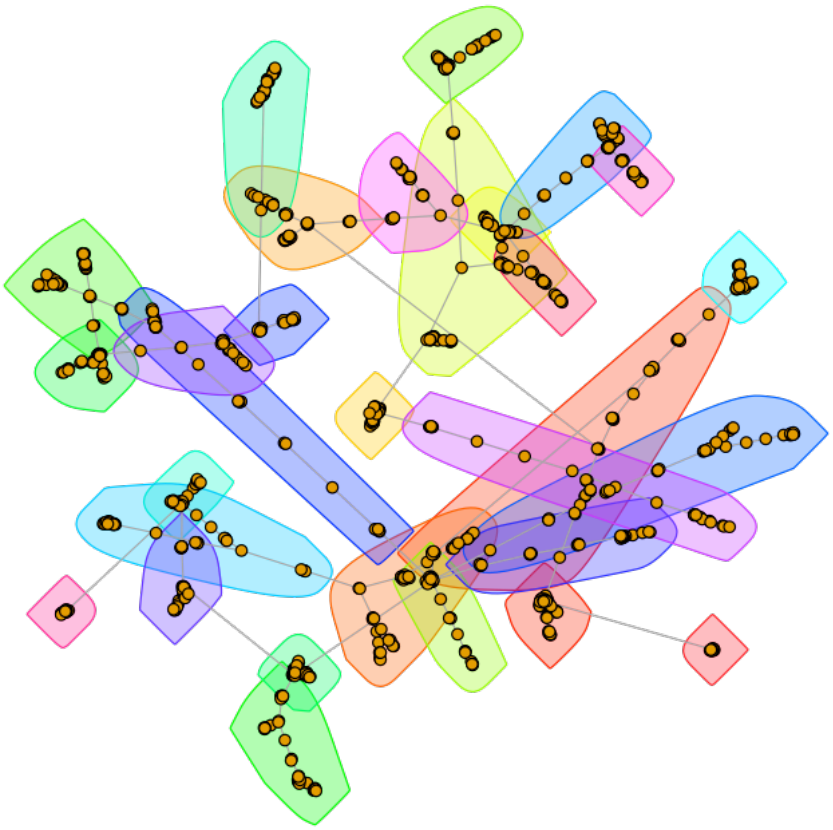
```
   xlab="Community",ylab="Size of Community",ylim=c(0,max(as.
 ↪vector(sizes(comm_struct2)))+10),border="red",col="pink")
dev.copy2eps(file='Q2hf.eps')
```

pdf: 2

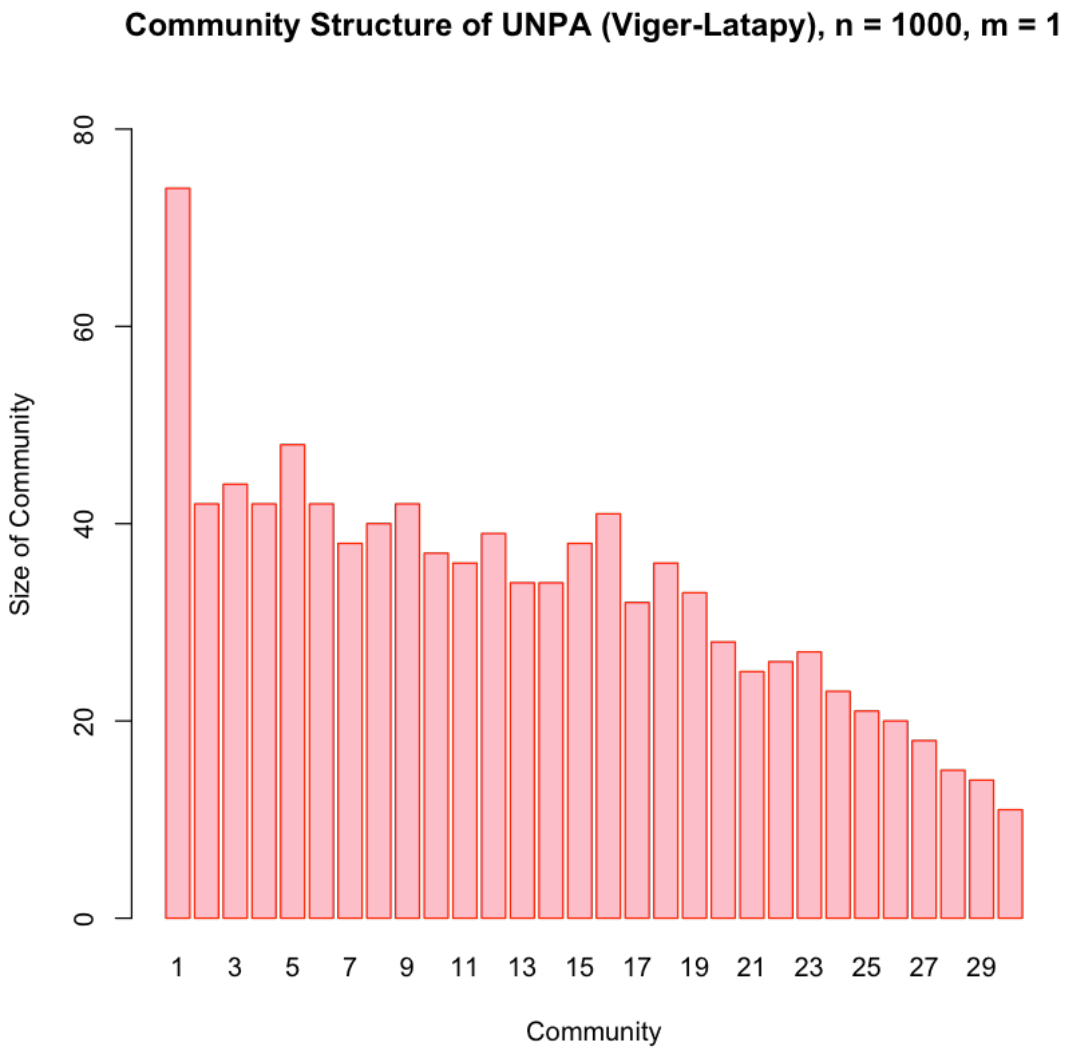
```
[1] "Modularity: 0.932926"
```



pdf: 2



pdf: 2

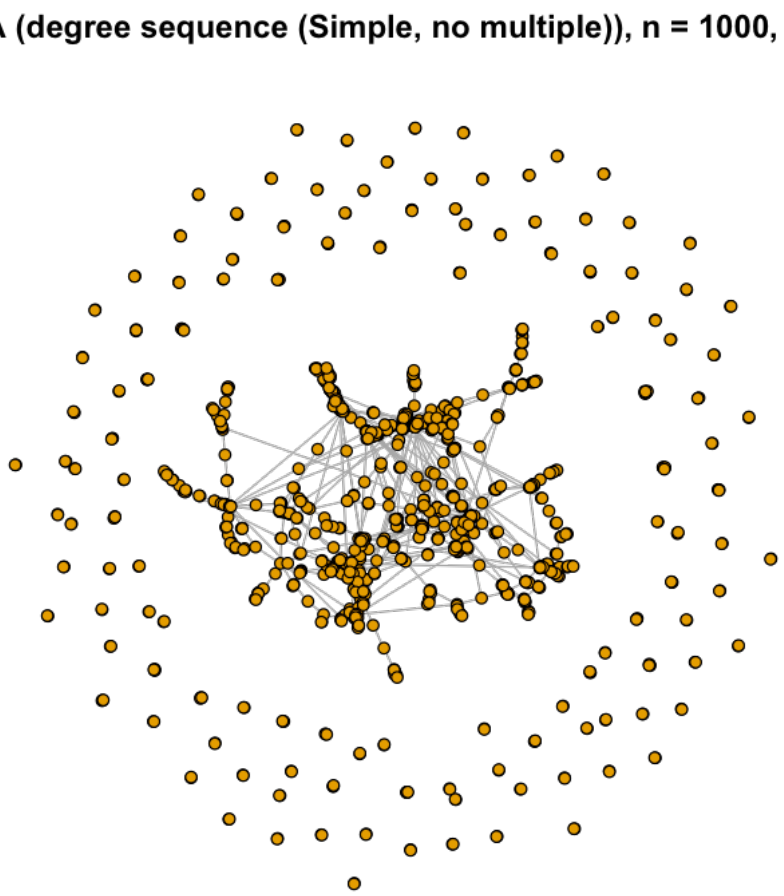


[43]:
```
g3 <- sample_degseq(degree(g),method = 'simple.no.multiple')
comm_struct3 <- cluster_fast_greedy(g3)
plot(g3,vertex.label="",vertex.size=3,main = "UNPA (degree sequence (Simple, no␣
 ↪multiple)), n = 1000, m = 1")
dev.copy2eps(file='Q2hg.eps')
mod <- modularity(comm_struct3)
print(sprintf("Modularity: %f",mod))
plot(g3, mark.groups = groups(comm_struct3), vertex.size=3, vertex.
 ↪label="",main="Community Structure of UNPA (Simple, no multiple), n = 1000,␣
 ↪m = 1")
dev.copy2pdf(file='Q2hh.pdf')
```

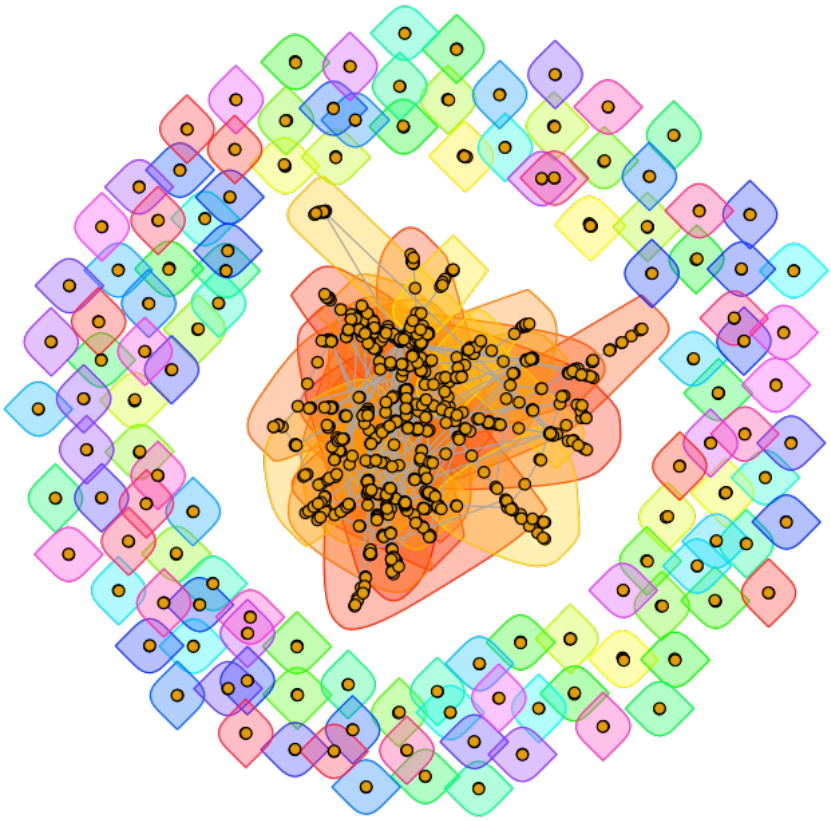
```
barplot(as.vector(sizes(comm_struct3)),names.arg =␣
 ↪seq(1,length(comm_struct3),1),main="Community Structure of UNPA (Simple, no␣
 ↪multiple), n = 1000, m = 1",
   xlab="Community",ylab="Size of Community",ylim=c(0,max(as.
 ↪vector(sizes(comm_struct3)))+10),border="red",col="pink")
dev.copy2eps(file='Q2hi.eps')
```

pdf: 2

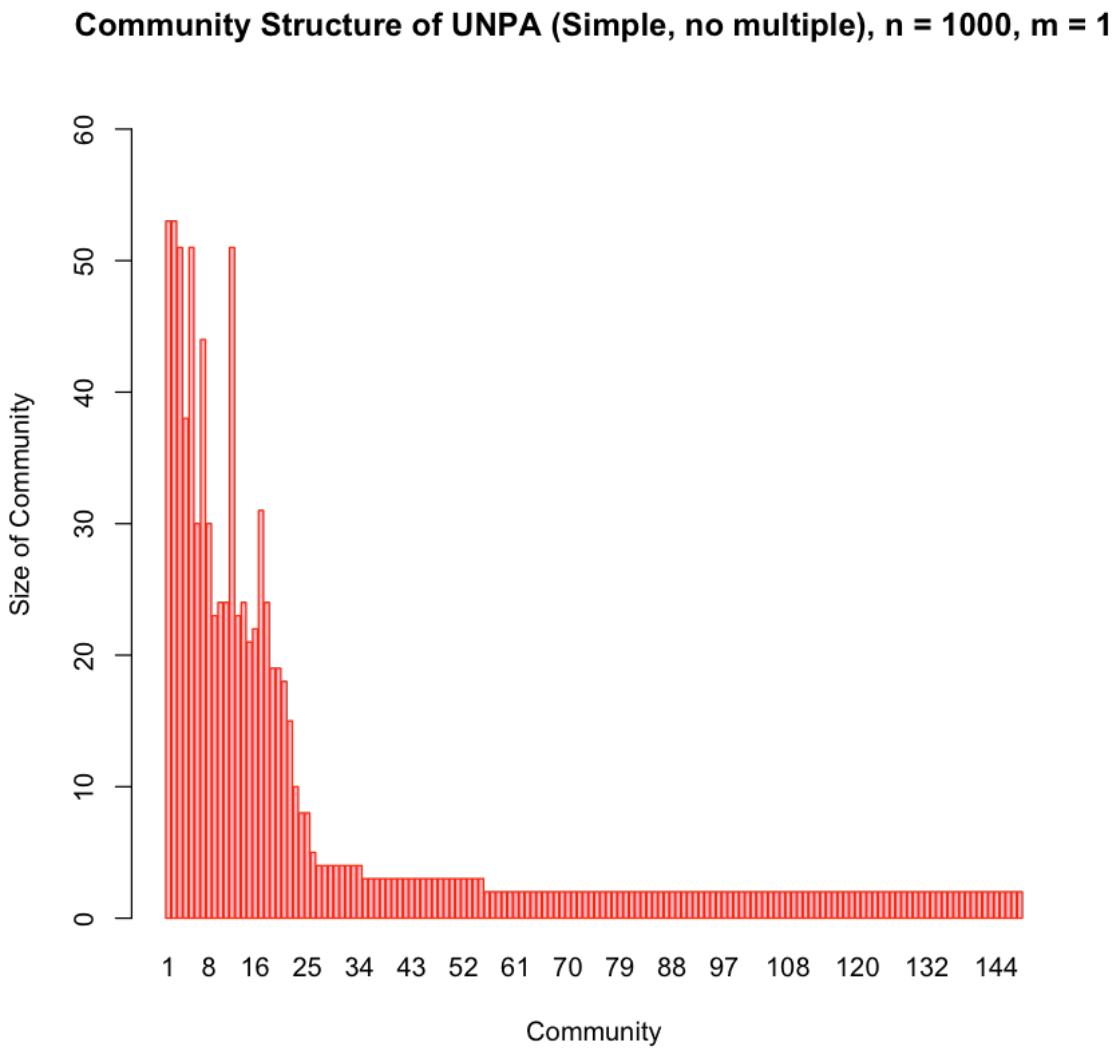
```
[1] "Modularity: 0.834776"
```



pdf: 2



pdf: 2

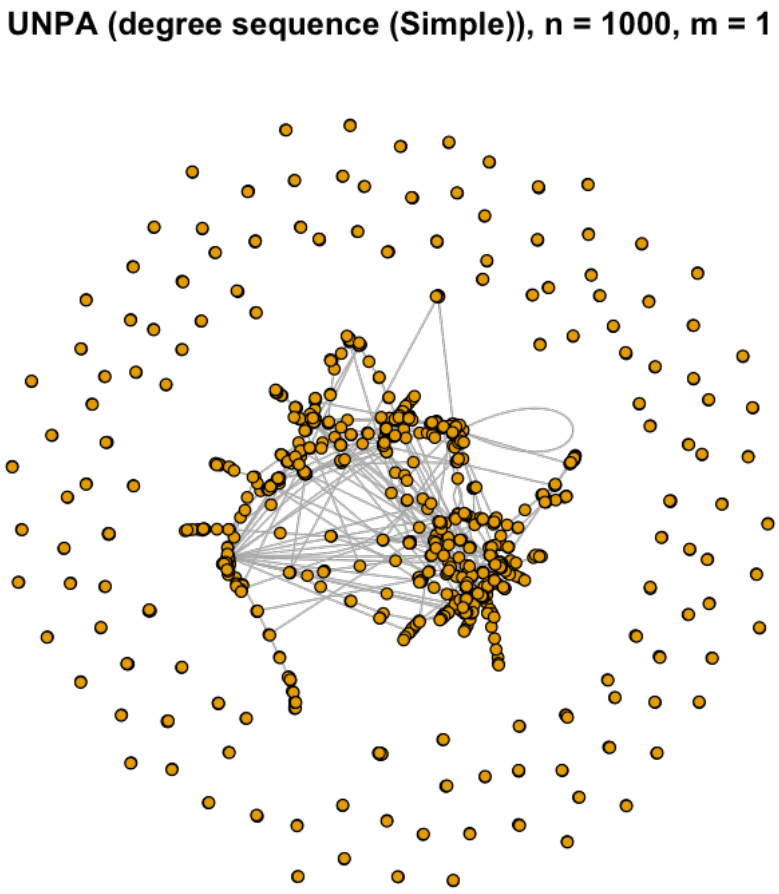


[44]:
```
g4 <- sample_degseq(degree(g),method = 'simple')
comm_struct4 <- walktrap.community(g4)
plot(g4,vertex.label="",vertex.size=3,main = "UNPA (degree sequence (Simple)),␣
 ↪n = 1000, m = 1")
dev.copy2eps(file='Q2hj.eps')
mod <- modularity(comm_struct4)
print(sprintf("Modularity: %f",mod))
plot(g4, mark.groups = groups(comm_struct4), vertex.size=3, vertex.
 ↪label="",main="Community Structure of UNPA (Simple), n = 1000, m = 1")
dev.copy2pdf(file='Q2hk.pdf')
barplot(as.vector(sizes(comm_struct4)),names.arg =␣
 ↪seq(1,length(comm_struct4),1),main="Community Structure of UNPA (Simple), n␣
 ↪= 1000, m = 1",
```

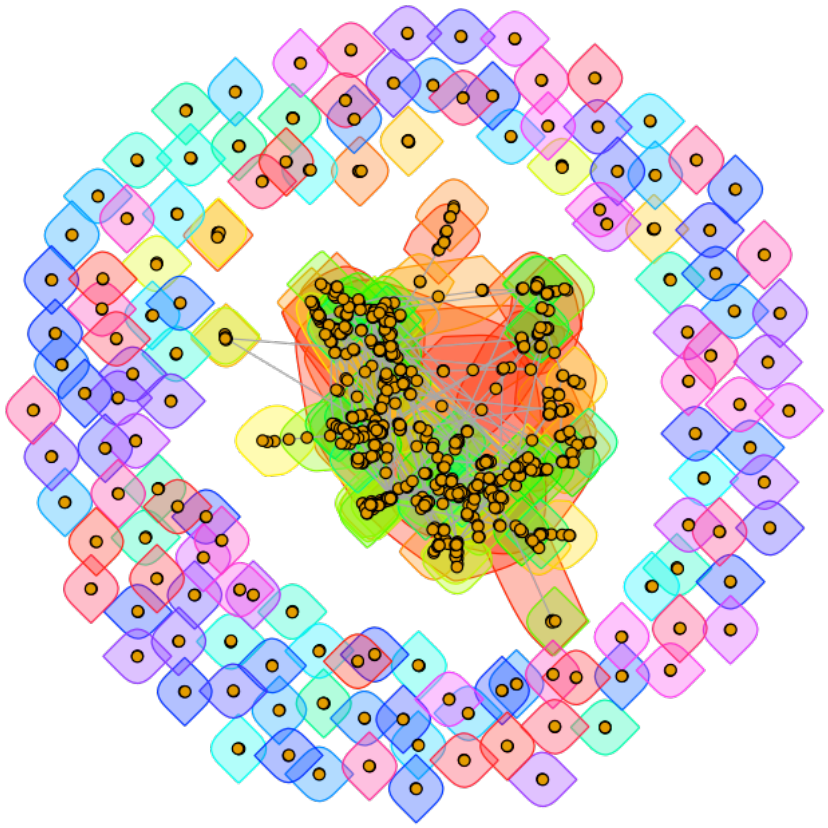
```
   xlab="Community",ylab="Size of Community",ylim=c(0,max(as.
 ↪vector(sizes(comm_struct4)))+10),border="red",col="pink")
dev.copy2eps(file='Q2hl.eps')
```

pdf: 2

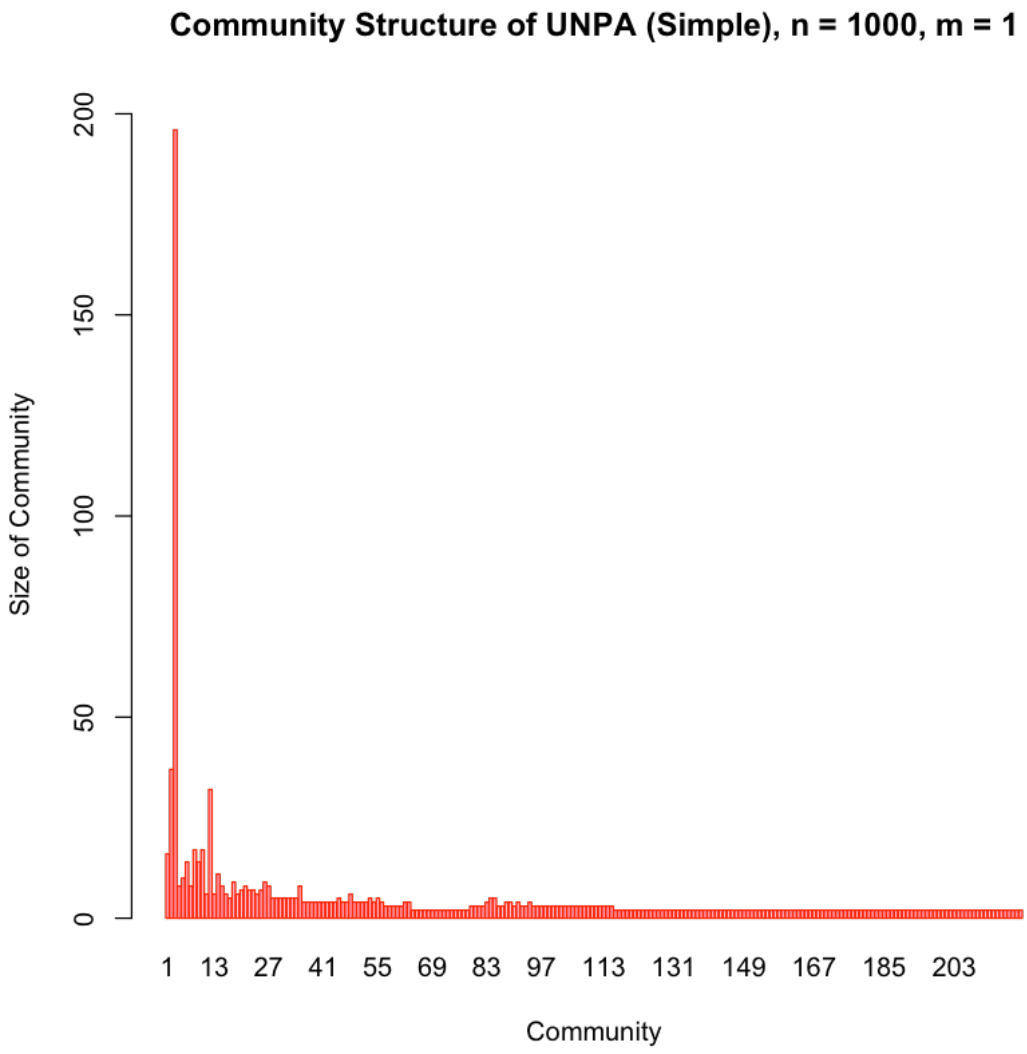
```
[1] "Modularity: 0.737921"
```



pdf: 2



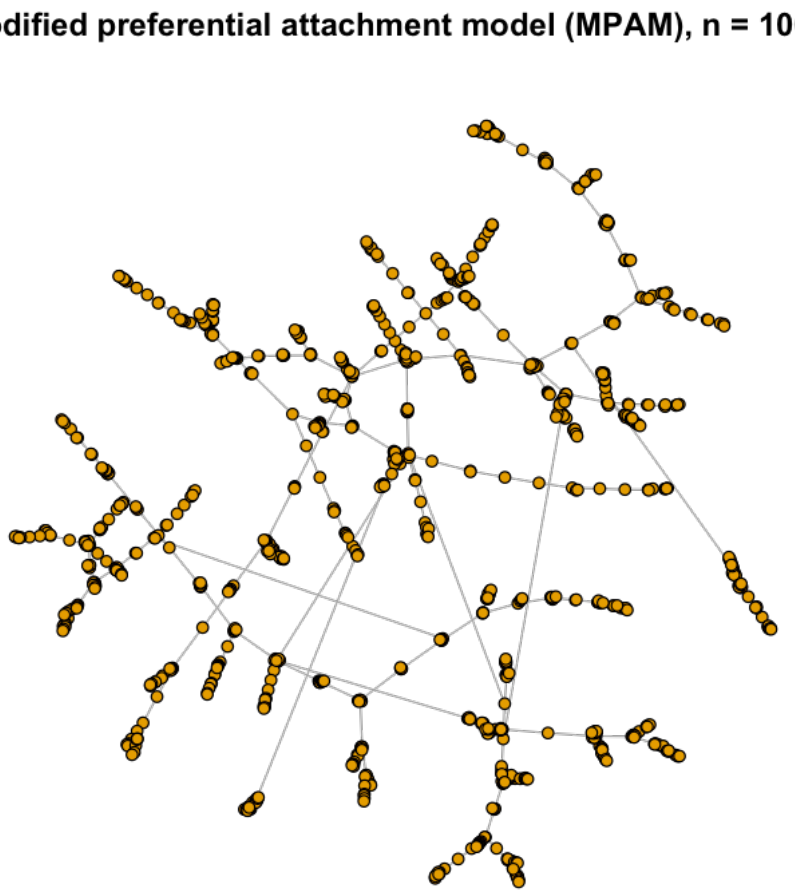
pdf: 2



1.13 Question 3a

[45]:
```
g <- sample_pa_age(n=1000, directed = FALSE, m=1, pa.exp=1, aging.exp=-1,
                   zero.deg.appeal = 1, zero.age.appeal = 0, deg.coef = 1, age.
 ↪coef = 1)
plot(g,vertex.label="",vertex.size=3,main = "Modified preferential attachment␣
 ↪model (MPAM), n = 1000")
dev.copy2eps(file='Q3aa.eps')
```

pdf: 2

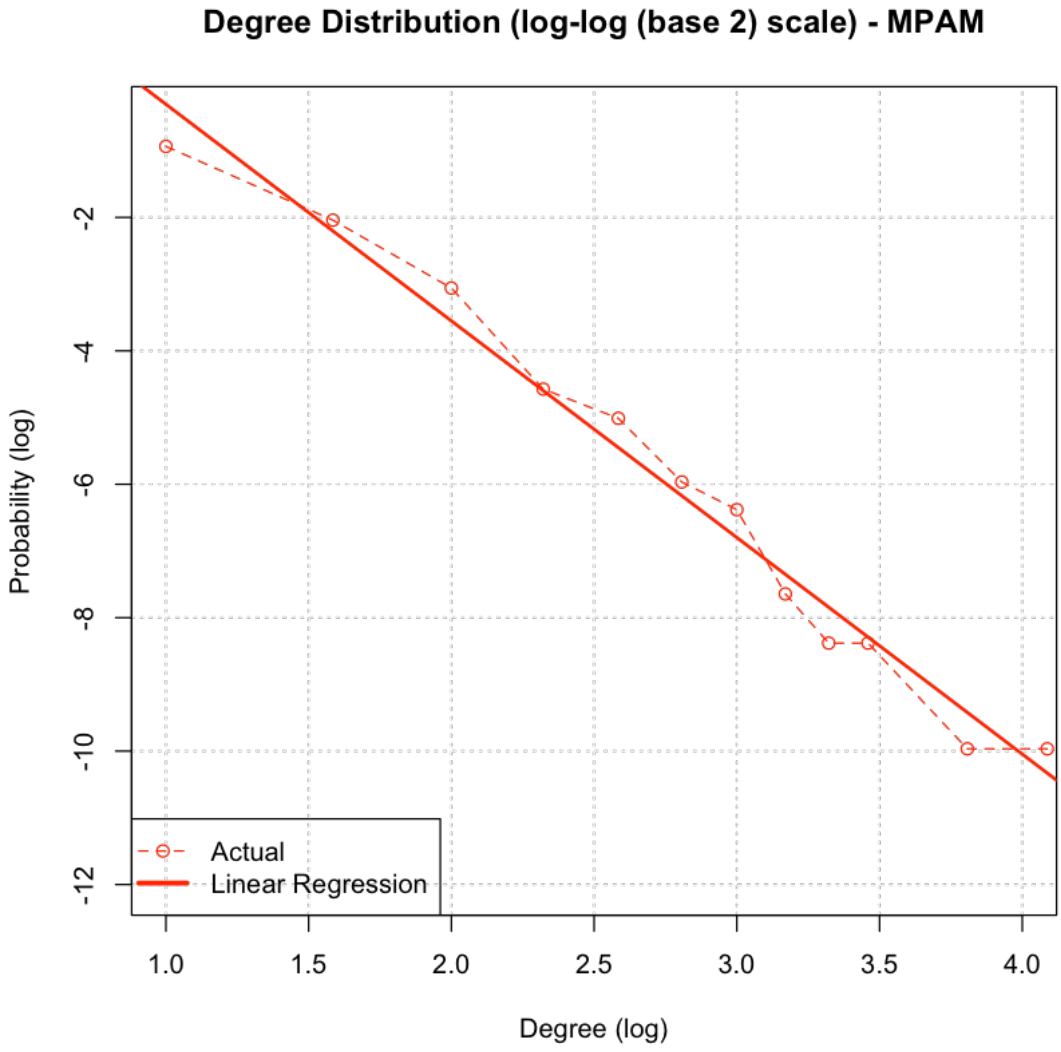


[46]:
```
deg_dist_1 = degree.distribution(g)
deg_1 <- log2(c(1:length(deg_dist_1)))[which(deg_dist_1 !=0, arr.ind = TRUE)]
dist_1 <- log2(deg_dist_1)[which(deg_dist_1 !=0, arr.ind = TRUE)]
plot(deg_1,dist_1,main="Degree Distribution (log-log (base 2) scale) - MPAM",
     xlab="Degree (log)",ylab="Probability␣
 ↪(log)",grid(),col="red",ylim=c(-12,-0.5),xlim=c(1,4))
lines(deg_1,dist_1,lty=2,col="red")
abline(lm(dist_1 ~ deg_1),col="red",lwd=2)

legend('bottomleft', legend = c("Actual","Linear Regression"),
       lty = c(2, 1), lwd = c(1,3), pch=c(1,NA),
       col = c('red','red'))
dev.copy2eps(file='Q3ab.eps')
```

```r
print("Slope and intercept for MPAM:")
print(lm(dist_1 ~ deg_1))
```

pdf: 2

```
[1] "Slope and intercept for MPAM:"

Call:
lm(formula = dist_1 ~ deg_1)

Coefficients:
(Intercept)        deg_1
      2.945       -3.248
```

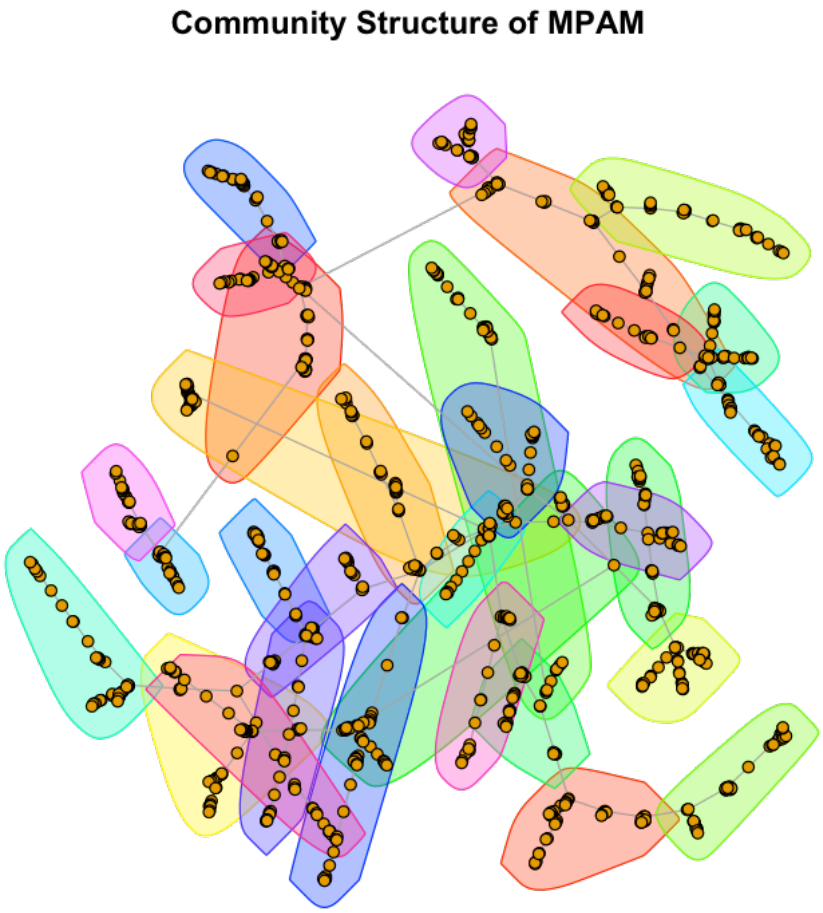


1.14 Question 3b

[47]:
```
comm_struct <- cluster_fast_greedy(g)
mod <- modularity(comm_struct)
print(sprintf("Modularity: %f",mod))
plot(g, mark.groups = groups(comm_struct), vertex.size=3, vertex.
 ↪label="",main="Community Structure of MPAM")
dev.copy2pdf(file='Q3ba.pdf')
```

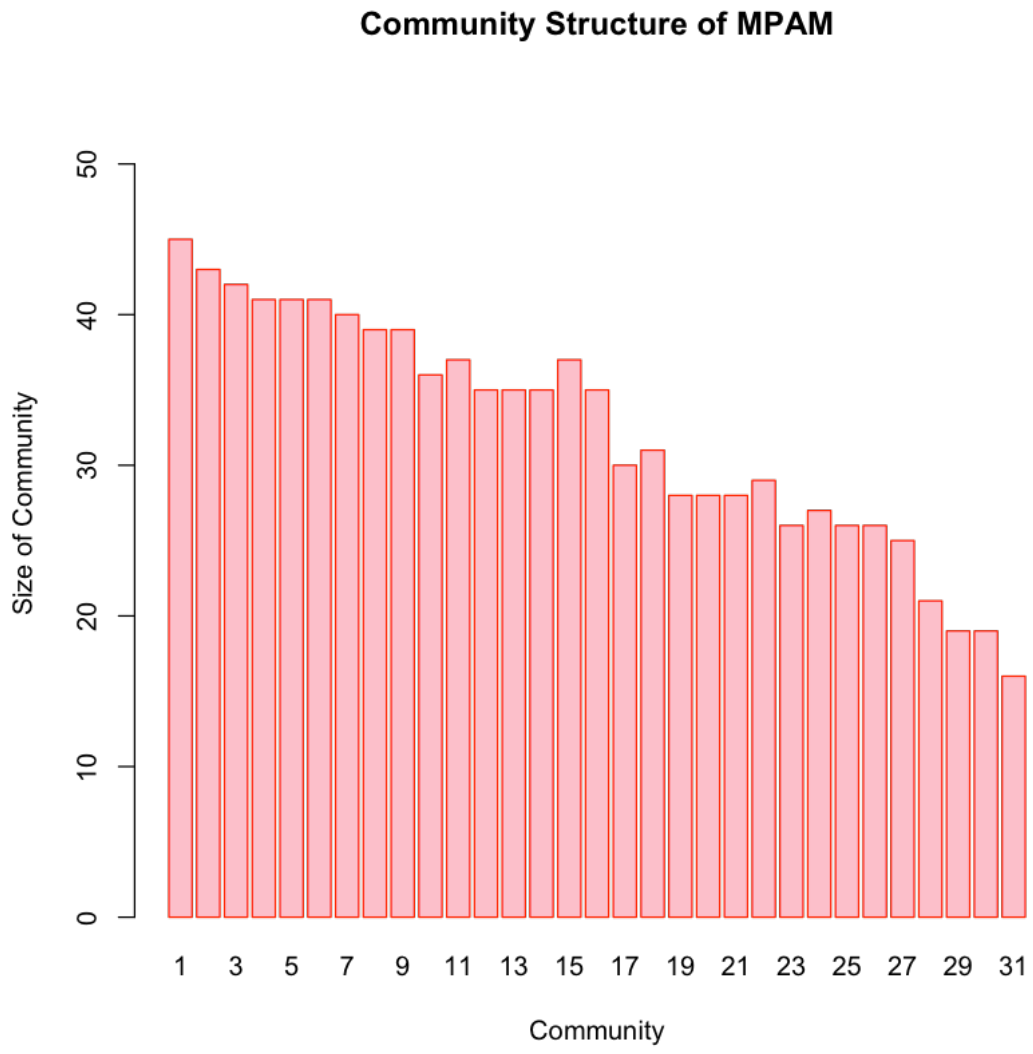
```
[1] "Modularity: 0.935772"
```

pdf: 2



[48]:
```
barplot(as.vector(sizes(comm_struct)),names.arg =␣
↪seq(1,length(comm_struct),1),main="Community Structure of MPAM",
  xlab="Community",ylab="Size of Community",ylim=c(0,max(as.
↪vector(sizes(comm_struct)))+10),border="red",col="pink")
dev.copy2eps(file='Q3bb.eps')
```

pdf: 2

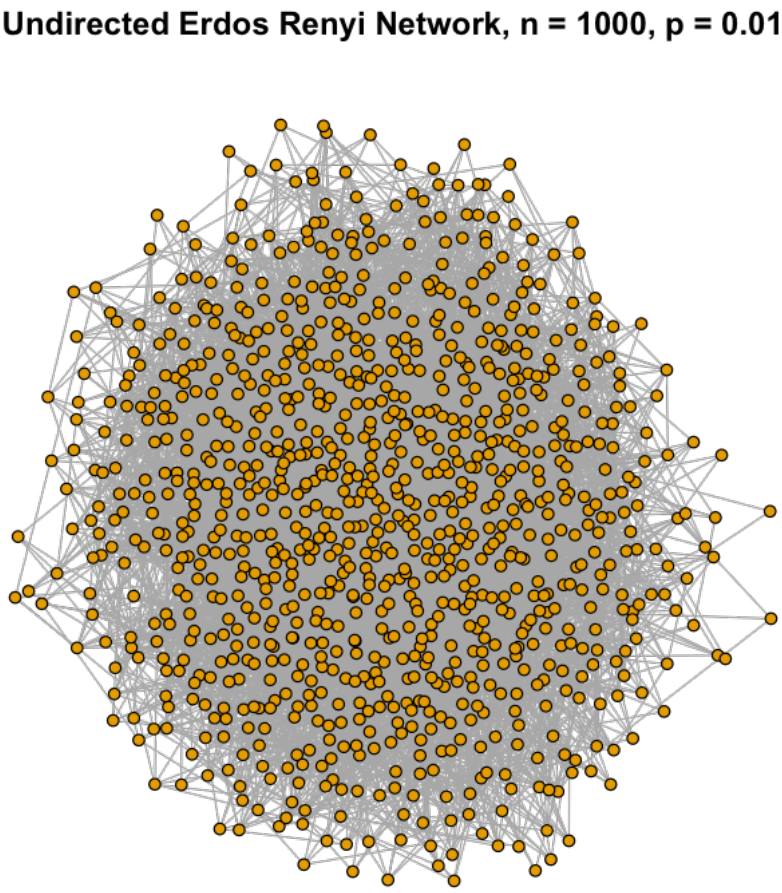


2 Random Walk on Networks

2.1 Question 1a

[49]:
```
g <- erdos.renyi.game(n = 1000,type = "gnp",p=0.01,directed = FALSE)
plot(g,vertex.label="",vertex.size=3,main = "Undirected Erdos Renyi Network, n␣
 ↪= 1000, p = 0.01")
dev.copy2eps(file='2Q1a.eps')
```

pdf: 2

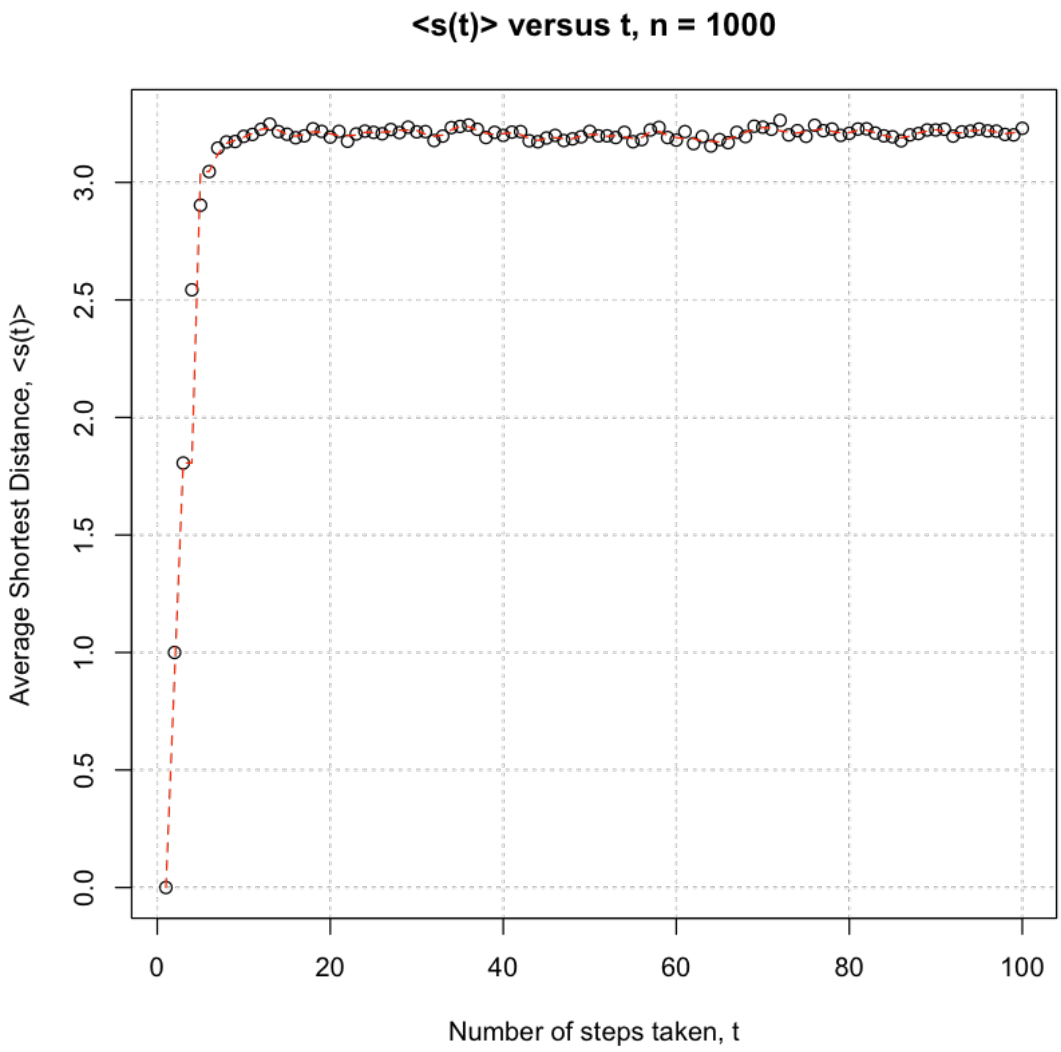


2.2 Question 1b and 1c

[50]:
```
s_t = matrix(data=0.0, nrow=1000, ncol=100)
term_node_deg = matrix(data=0.0,nrow=1000,ncol=1)
for (i in 1:1000){
    if(is_connected(g)){
        gcc <- g
    }
    else{
        g.components <- clusters(g)
        ix <- which.max(g.components$csize)
        g.giant <- induced.subgraph(g, which(g.components$membership == ix))
        gcc <- g.giant
    }
    start = sample(V(gcc), 1)
    walked_nodes = random_walk(gcc, steps=100, start)
    s_t[i,] = shortest.paths(gcc, walked_nodes, start)
    term_node_deg[i,1] = degree(gcc, walked_nodes[length(walked_nodes)])
}
```

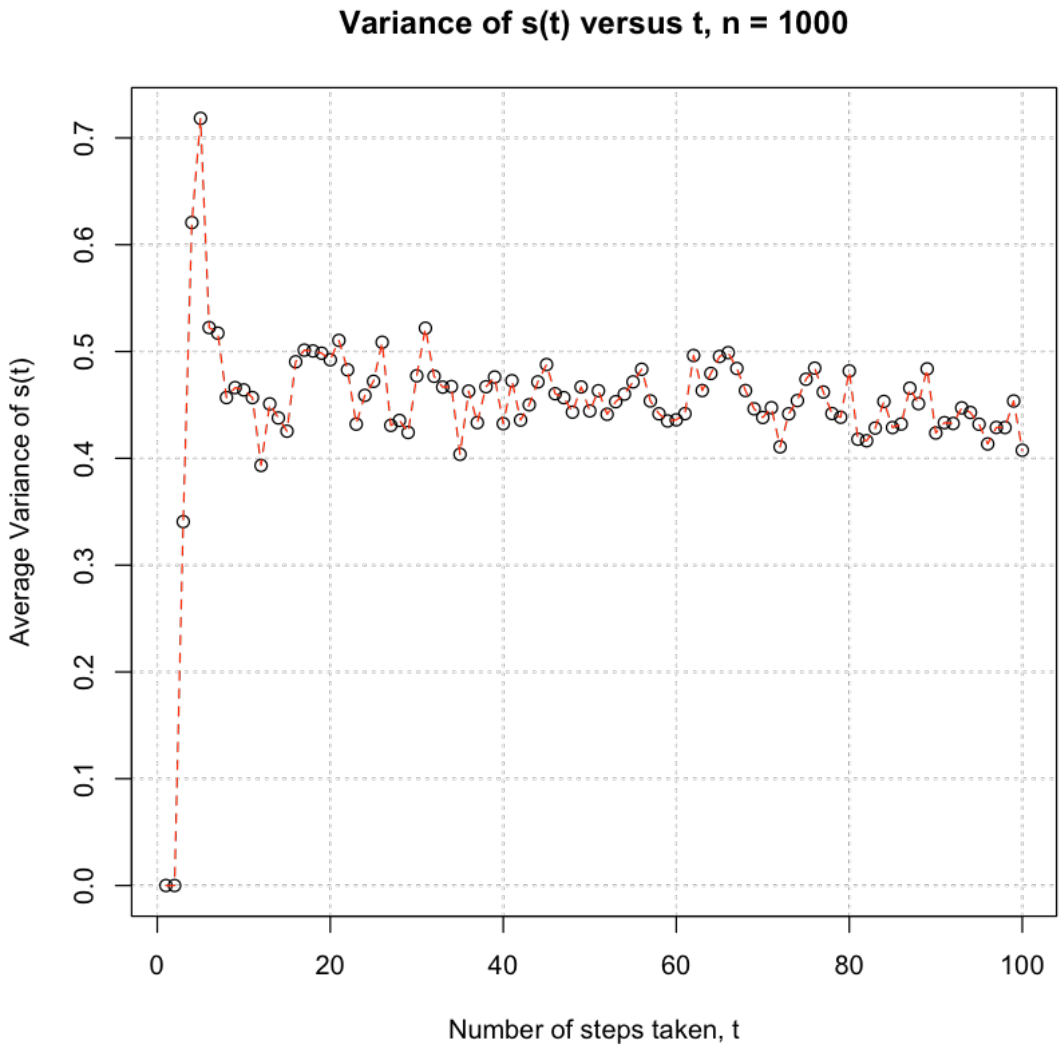
[51]:
```
plot(seq(1,100,1),colMeans(s_t),grid(), xlab = 'Number of steps taken, t', ylab␣
↪= 'Average Shortest Distance, <s(t)>',main="<s(t)> versus t, n = 1000")
lines(lowess(seq(1,100,1),colMeans(s_t),f = 0.04), col="red",lwd = 1,lty=2)
dev.copy2eps(file='2Q1ba.eps')
```

pdf: 2



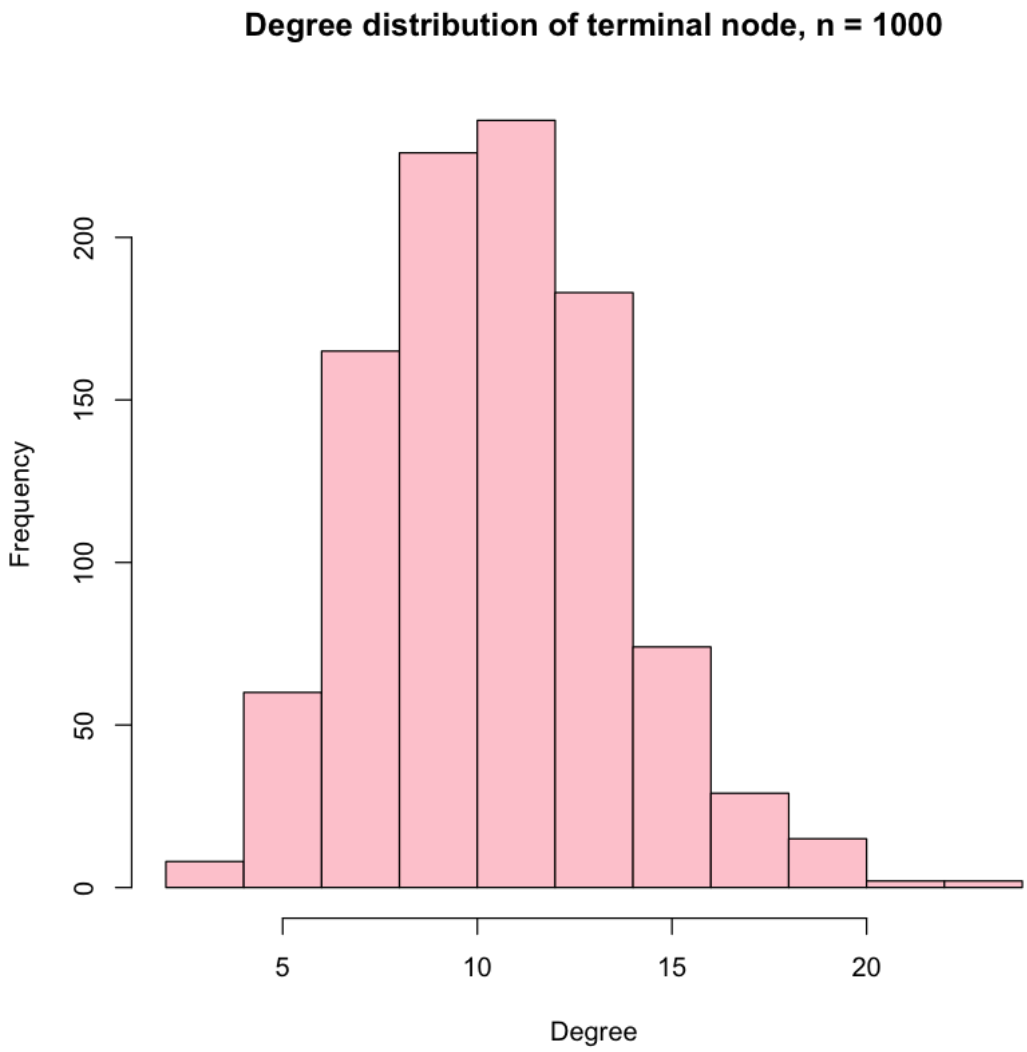
[52]:
```
plot(seq(1,100,1),colVars(s_t),grid(), xlab = 'Number of steps taken, t', ylab
↪= 'Average Variance of s(t)',main="Variance of s(t) versus t, n = 1000")
lines(lowess(seq(1,100,1),colVars(s_t),f = 0.03), col="red",lwd = 1,lty=2)
dev.copy2eps(file='2Q1bb.eps')
```

pdf: 2

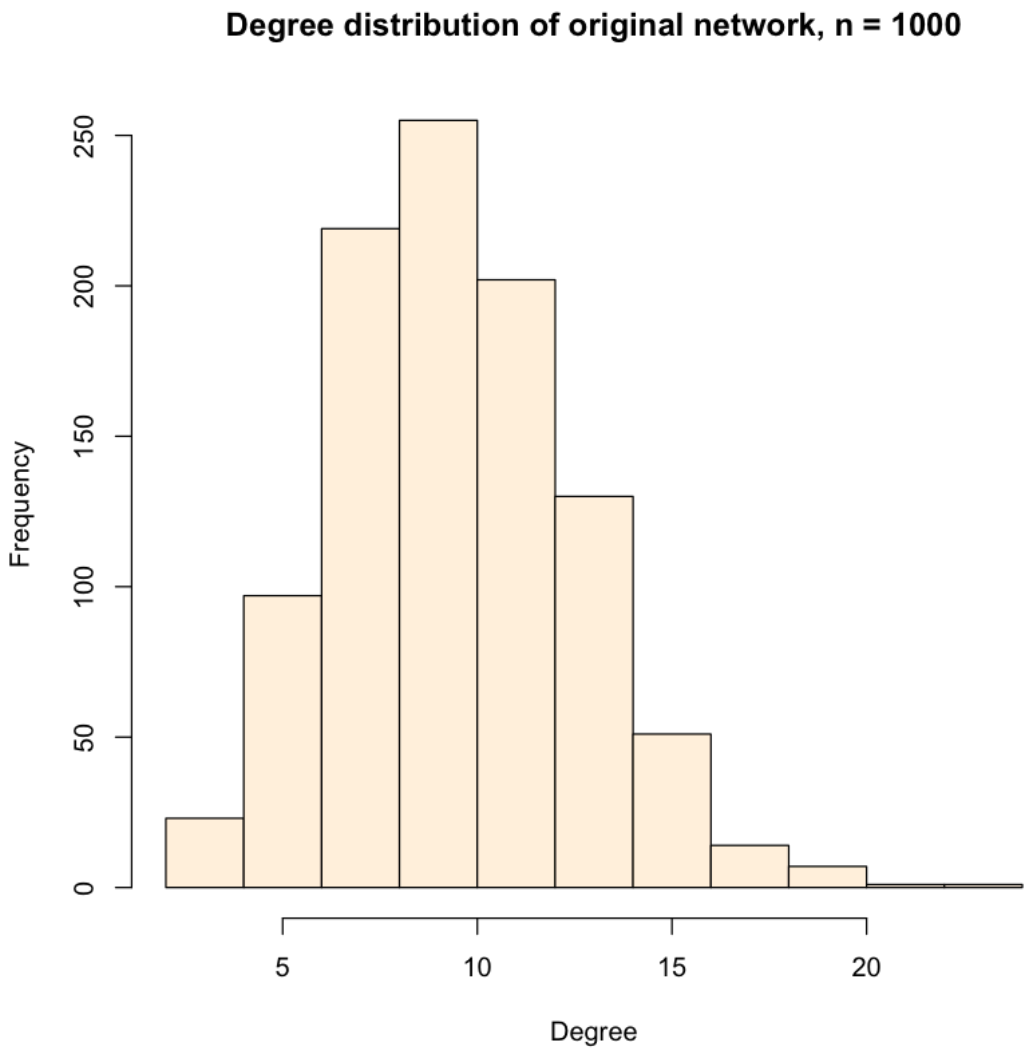


[53]:
```
hist(term_node_deg,col='pink',main="Degree distribution of terminal node, n =␣
↪1000",xlab='Degree',ylab='Frequency')
dev.copy2eps(file='2Q1ca.eps')
hist(degree(g),col='antiquewhite1',main="Degree distribution of original␣
↪network, n = 1000",xlab='Degree',ylab='Frequency')
dev.copy2eps(file='2Q1cb.eps')
```

pdf: 2


Degree distribution of terminal node, n = 1000
Frequency
0
50
100
150
200
5
10
15
20
Degree

pdf: 2



[54]:
```
print(diameter(g))
```

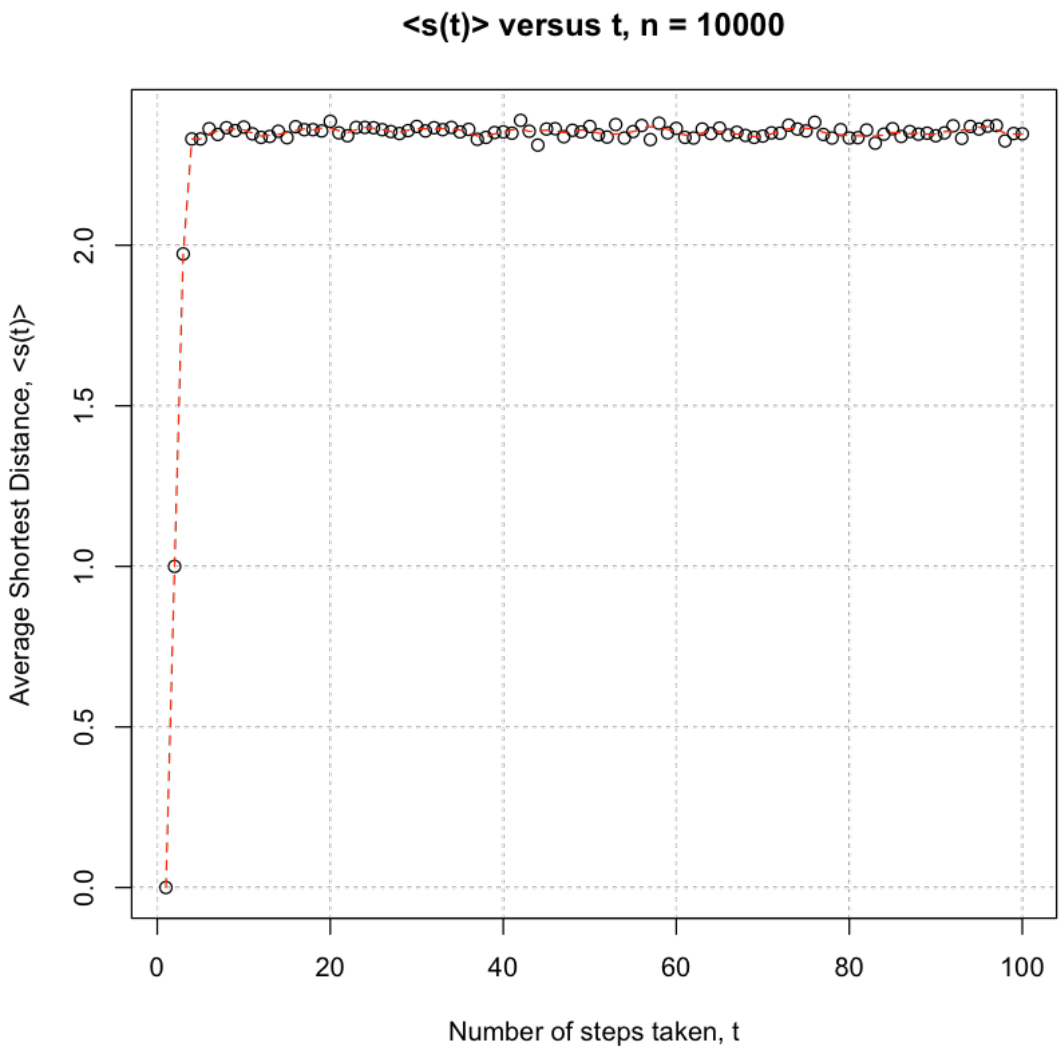
```
[1] 5
```

2.3 Question 1d

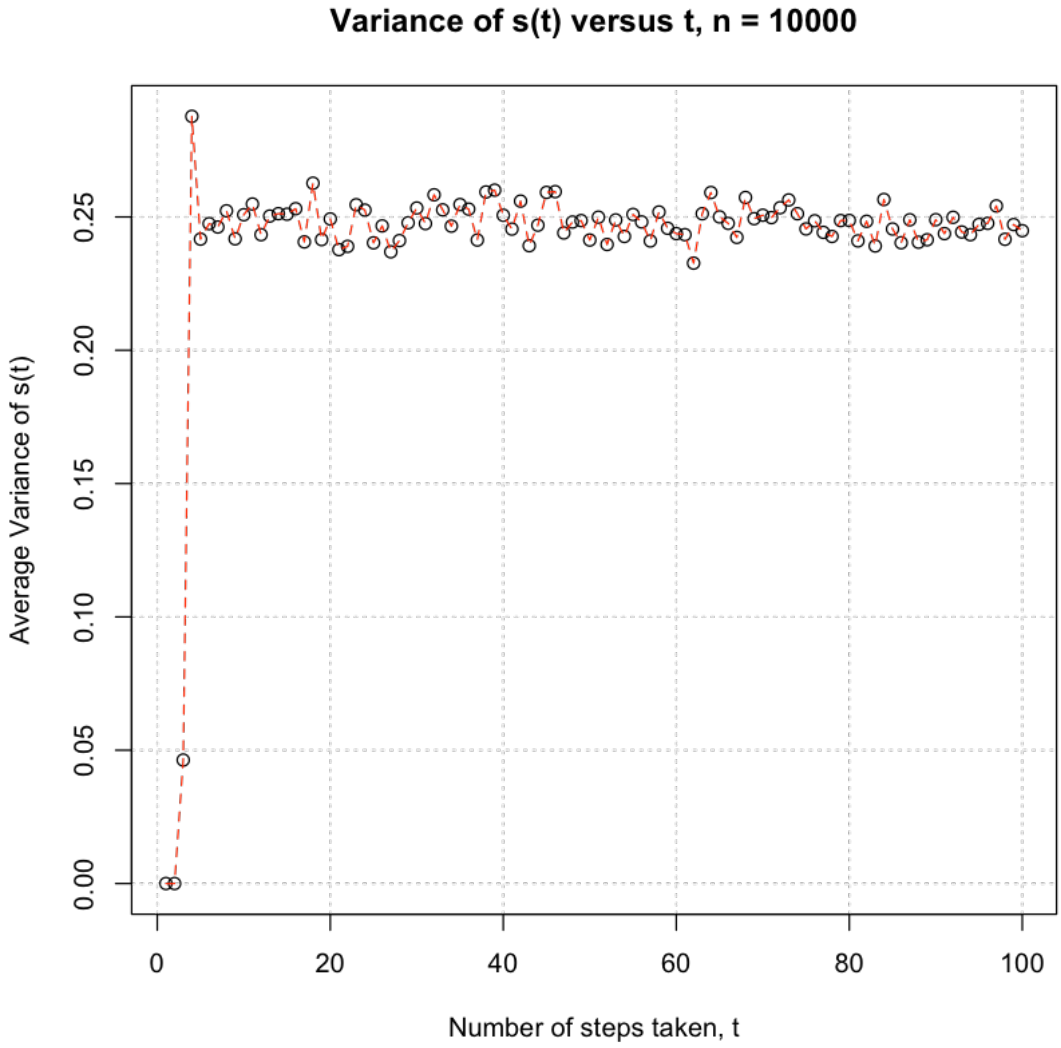
[55]:
```
g <- erdos.renyi.game(n = 10000,type = "gnp",p=0.01,directed = FALSE)
s_t = matrix(data=0.0, nrow=1000, ncol=100)
term_node_deg = matrix(data=0.0,nrow=1000,ncol=1)
for (i in 1:1000){
    if(is_connected(g)){
        gcc <- g
    }
```

```
    else{
        g.components <- clusters(g)
        ix <- which.max(g.components$csize)
        g.giant <- induced.subgraph(g, which(g.components$membership == ix))
        gcc <- g.giant
    }
    start = sample(V(gcc), 1)
    walked_nodes = random_walk(gcc, steps=100, start)
    s_t[i,] = shortest.paths(gcc, walked_nodes, start)
    term_node_deg[i,1] = degree(gcc, walked_nodes[length(walked_nodes)])
}
plot(seq(1,100,1),colMeans(s_t),grid(), xlab = 'Number of steps taken, t', ylab␣
 ↪= 'Average Shortest Distance, <s(t)>',main="<s(t)> versus t, n = 10000")
lines(lowess(seq(1,100,1),colMeans(s_t),f = 0.05), col="red",lwd = 1,lty=2)
dev.copy2eps(file='2Q1da.eps')

plot(seq(1,100,1),colVars(s_t),grid(), xlab = 'Number of steps taken, t', ylab␣
 ↪= 'Average Variance of s(t)',main="Variance of s(t) versus t, n = 10000")
lines(lowess(seq(1,100,1),colVars(s_t),f = 0.03), col="red",lwd = 1,lty=2)
dev.copy2eps(file='2Q1db.eps')
```

pdf: 2


<s(t)> versus t, n = 10000
Average Shortest Distance, <s(t)>
0.0
0.5
1.0
1.5
2.0
0
20
40
60
80
100
Number of steps taken, t

pdf: 2



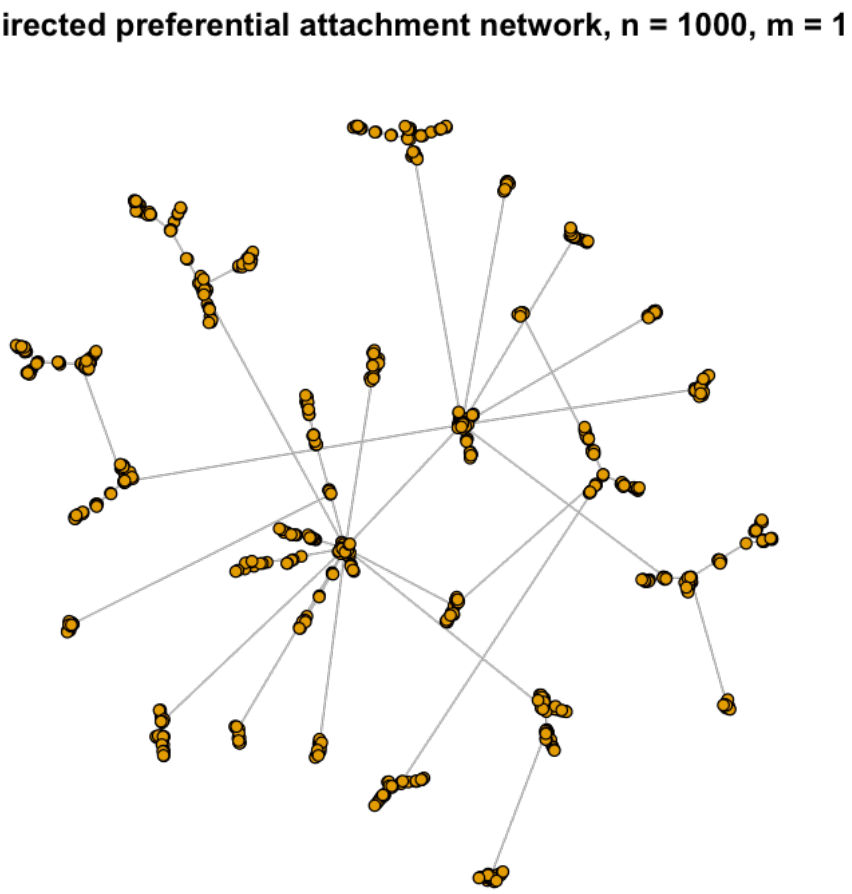
[56]:
```
print(diameter(g))
```

```
[1] 3
```

2.4 Question 2a

[11]:
```
g <- barabasi.game(1000, m=1, directed=FALSE)
plot(g,vertex.label="",vertex.size=3,main = "Undirected preferential attachment␣
 ↪network, n = 1000, m = 1")
dev.copy2eps(file='2Q2a.eps')
```

pdf: 2

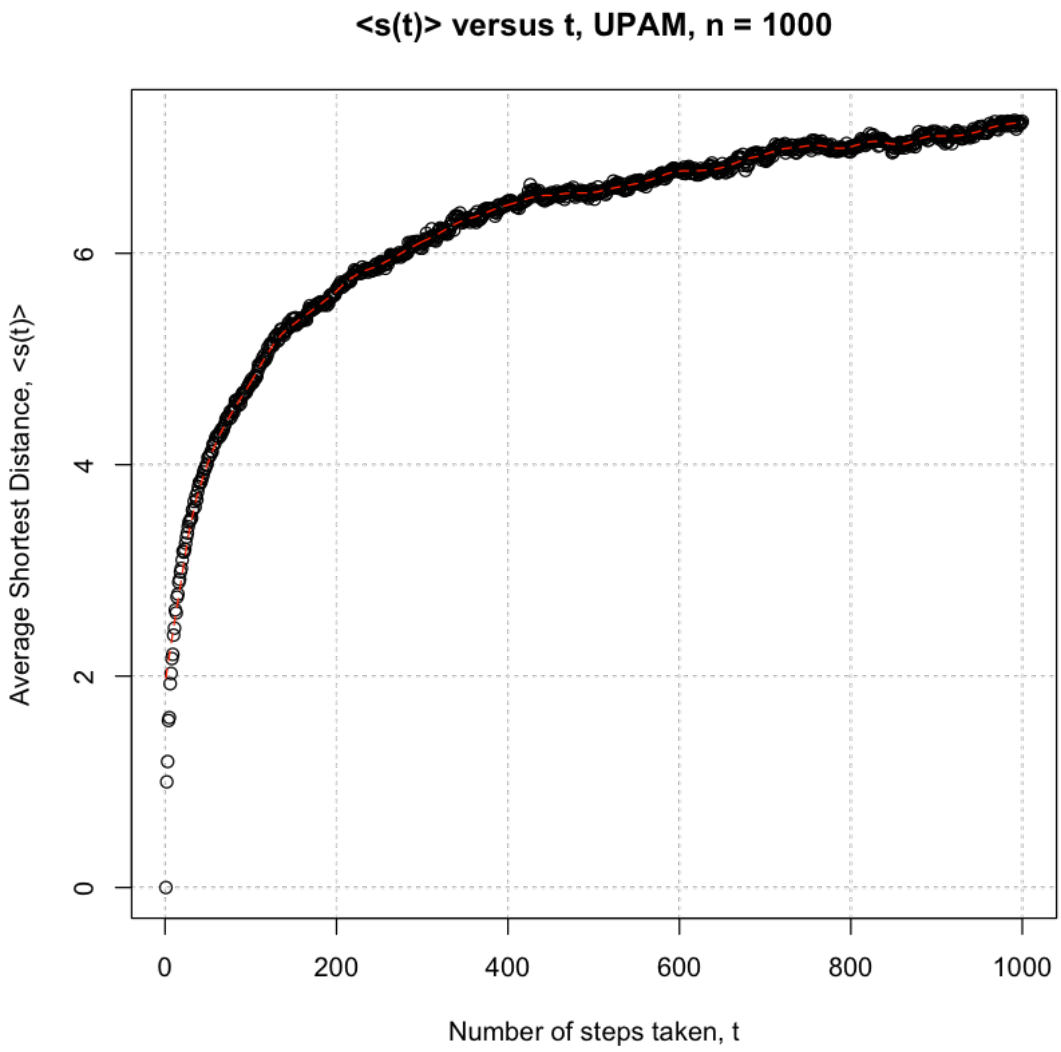


2.5 Question 2b

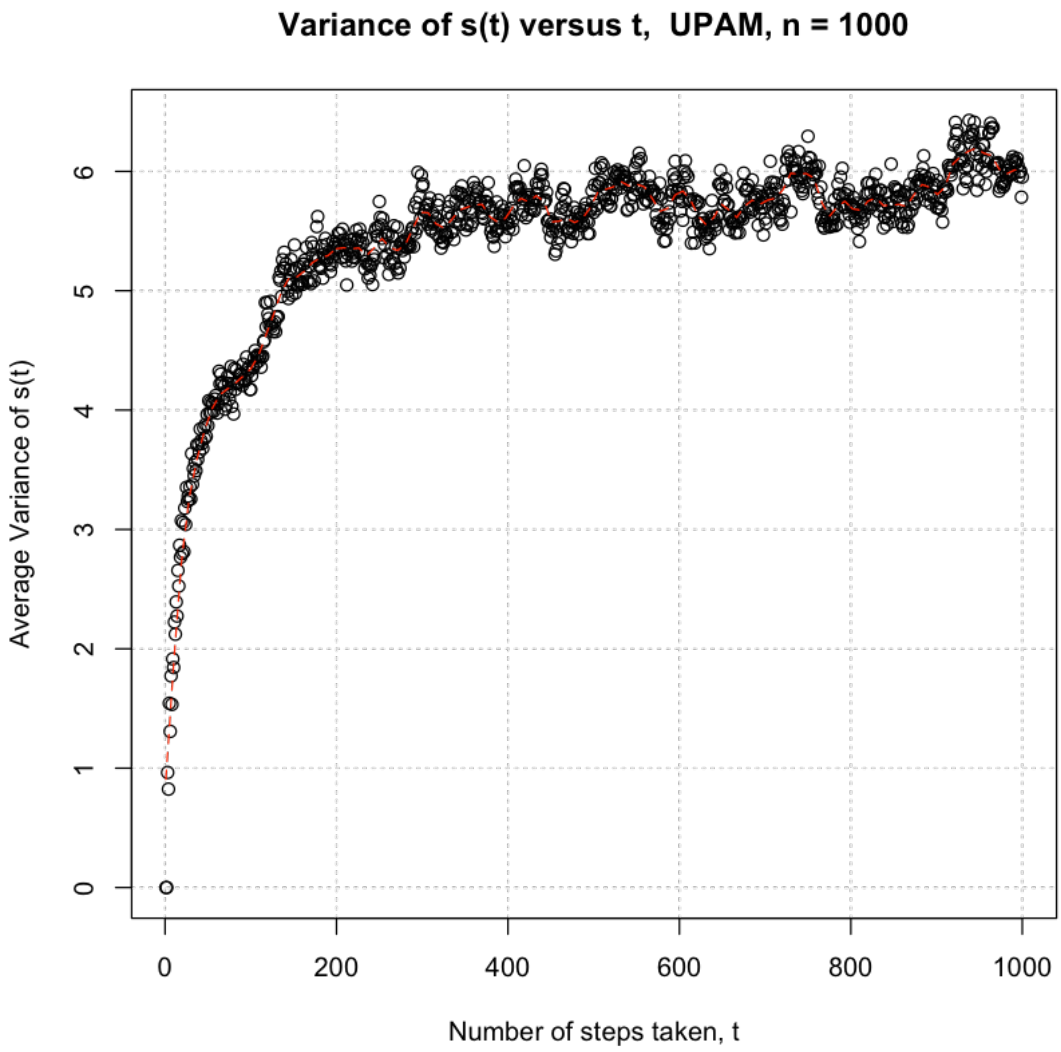
[13]:
```
s_t = matrix(data=0.0, nrow=1000, ncol=1000)
term_node_deg = matrix(data=0.0,nrow=1000,ncol=1)
for (i in 1:1000){
    if(is_connected(g)){
        gcc <- g
    }
    else{
        g.components <- clusters(g)
        ix <- which.max(g.components$csize)
        g.giant <- induced.subgraph(g, which(g.components$membership == ix))
        gcc <- g.giant
```

```
    }
    start = sample(V(gcc), 1)
    walked_nodes = random_walk(gcc, steps=1000, start)
    s_t[i,] = shortest.paths(gcc, walked_nodes, start)
    term_node_deg[i,1] = degree(gcc, walked_nodes[length(walked_nodes)])
}
plot(seq(1,1000,1),colMeans(s_t),grid(), xlab = 'Number of steps taken, t',␣
 ↪ylab = 'Average Shortest Distance, <s(t)>',main="<s(t)> versus t, UPAM, n =␣
 ↪1000")
lines(lowess(seq(1,1000,1),colMeans(s_t),f = 0.05), col="red",lwd = 1,lty=2)
dev.copy2eps(file='2Q2ba.eps')

plot(seq(1,1000,1),colVars(s_t),grid(), xlab = 'Number of steps taken, t', ylab␣
 ↪= 'Average Variance of s(t)',main="Variance of s(t) versus t,  UPAM, n =␣
 ↪1000")
lines(lowess(seq(1,1000,1),colVars(s_t),f = 0.03), col="red",lwd = 1,lty=2)
dev.copy2eps(file='2Q2bb.eps')
```

pdf: 2


<s(t)> versus t, UPAM, n = 1000
Average Shortest Distance, <s(t)>
0
2
4
6
0
200
400
600
800
1000
Number of steps taken, t

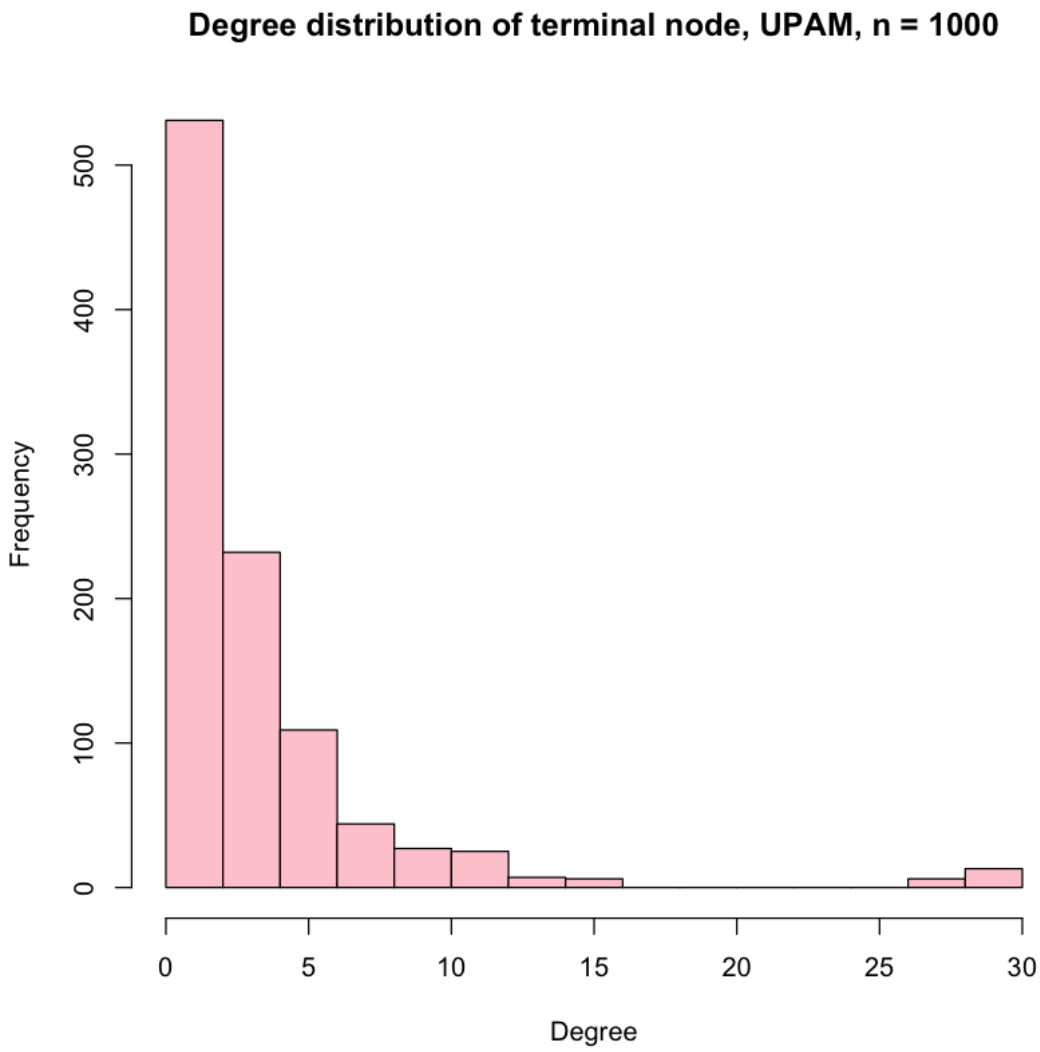
pdf: 2



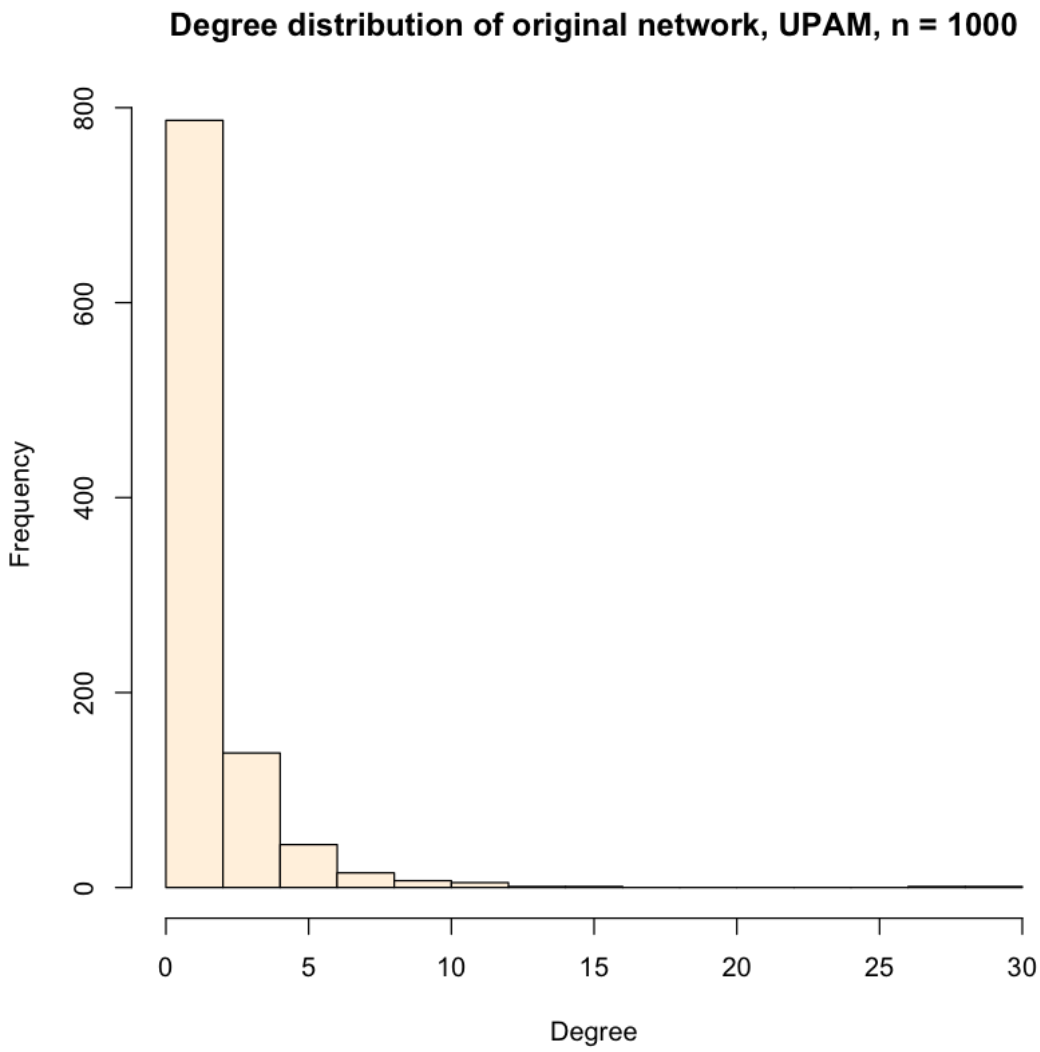
2.6 Question 2c

[14]:
```
hist(term_node_deg,col='pink',main="Degree distribution of terminal node, UPAM,␣
 ↪n = 1000",xlab='Degree',ylab='Frequency')
dev.copy2eps(file='2Q2ca.eps')
hist(degree(g),col='antiquewhite1',main="Degree distribution of original␣
 ↪network, UPAM, n = 1000",xlab='Degree',ylab='Frequency')
dev.copy2eps(file='2Q2cb.eps')
```

pdf: 2


Degree distribution of terminal node, UPAM, n = 1000
Frequency
0
100
200
300
400
500
0
5
10
15
20
25
30
Degree

pdf: 2



[15]:
```
print(diameter(g))
```

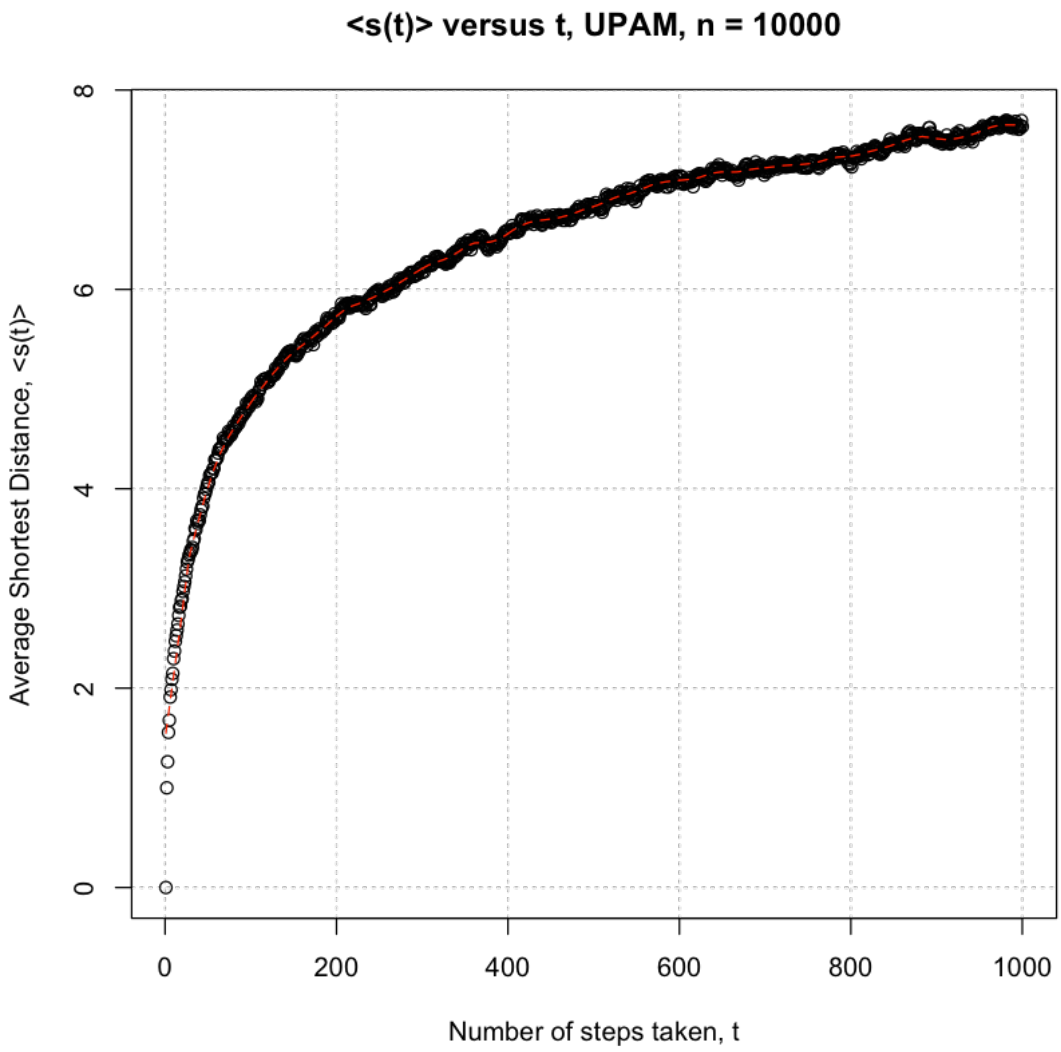
```
[1] 18
```

2.7 Question 2d

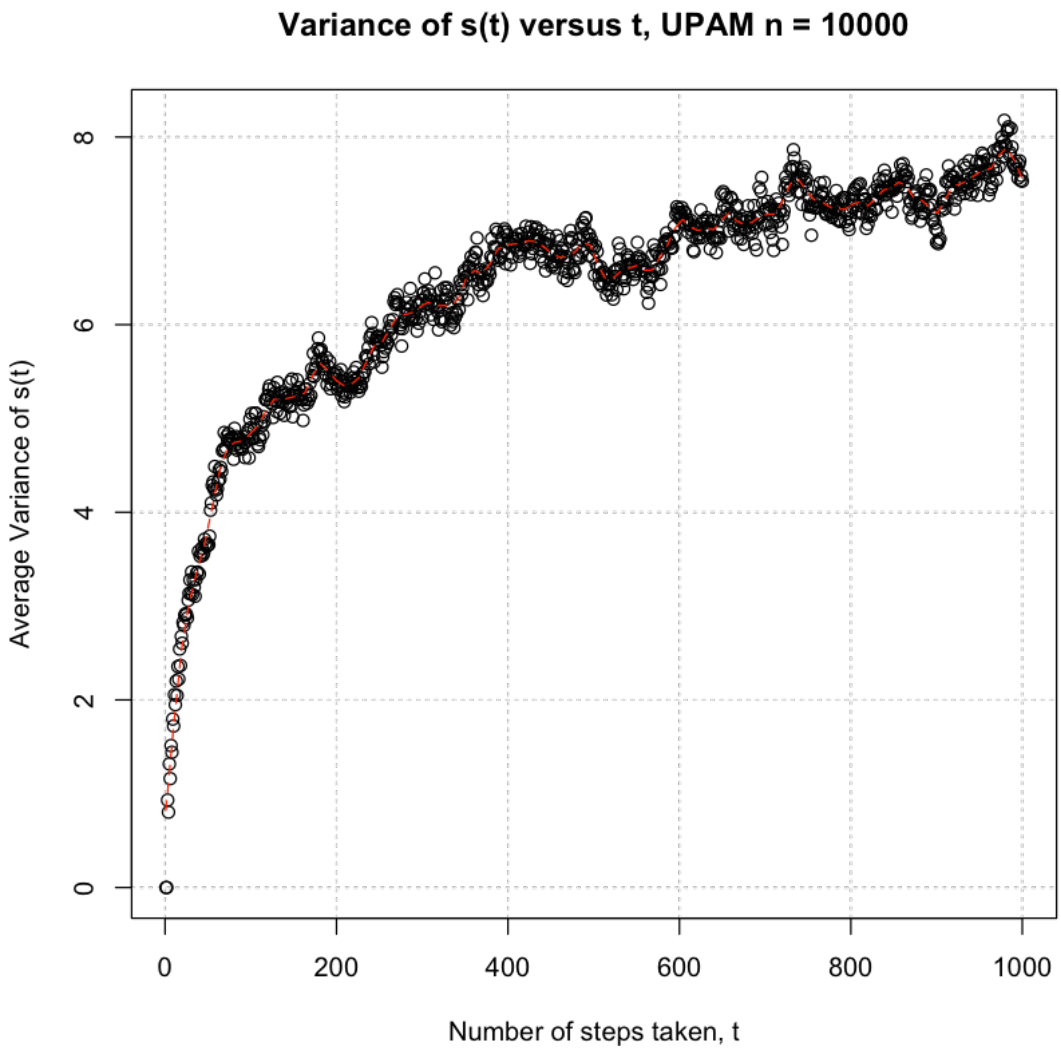
[16]:
```
g <-  barabasi.game(10000, m=1, directed=FALSE)
s_t = matrix(data=0.0, nrow=1000, ncol=1000)
term_node_deg = matrix(data=0.0,nrow=1000,ncol=1)
for (i in 1:1000){
    if(is_connected(g)){
        gcc <- g
    }
```

```
    else{
        g.components <- clusters(g)
        ix <- which.max(g.components$csize)
        g.giant <- induced.subgraph(g, which(g.components$membership == ix))
        gcc <- g.giant
    }
    start = sample(V(gcc), 1)
    walked_nodes = random_walk(gcc, steps=1000, start)
    s_t[i,] = shortest.paths(gcc, walked_nodes, start)
    term_node_deg[i,1] = degree(gcc, walked_nodes[length(walked_nodes)])
}
plot(seq(1,1000,1),colMeans(s_t),grid(), xlab = 'Number of steps taken, t',␣
 ↪ylab = 'Average Shortest Distance, <s(t)>',main="<s(t)> versus t, UPAM, n =␣
 ↪10000")
lines(lowess(seq(1,1000,1),colMeans(s_t),f = 0.05), col="red",lwd = 1,lty=2)
dev.copy2eps(file='2Q2da.eps')

plot(seq(1,1000,1),colVars(s_t),grid(), xlab = 'Number of steps taken, t', ylab␣
 ↪= 'Average Variance of s(t)',main="Variance of s(t) versus t, UPAM n =␣
 ↪10000")
lines(lowess(seq(1,1000,1),colVars(s_t),f = 0.03), col="red",lwd = 1,lty=2)
dev.copy2eps(file='2Q2db.eps')
```

pdf: 2


<s(t)> versus t, UPAM, n = 10000
Average Shortest Distance, <s(t)>
Number of steps taken, t
0
2
4
6
8
0
200
400
600
800
1000

pdf: 2



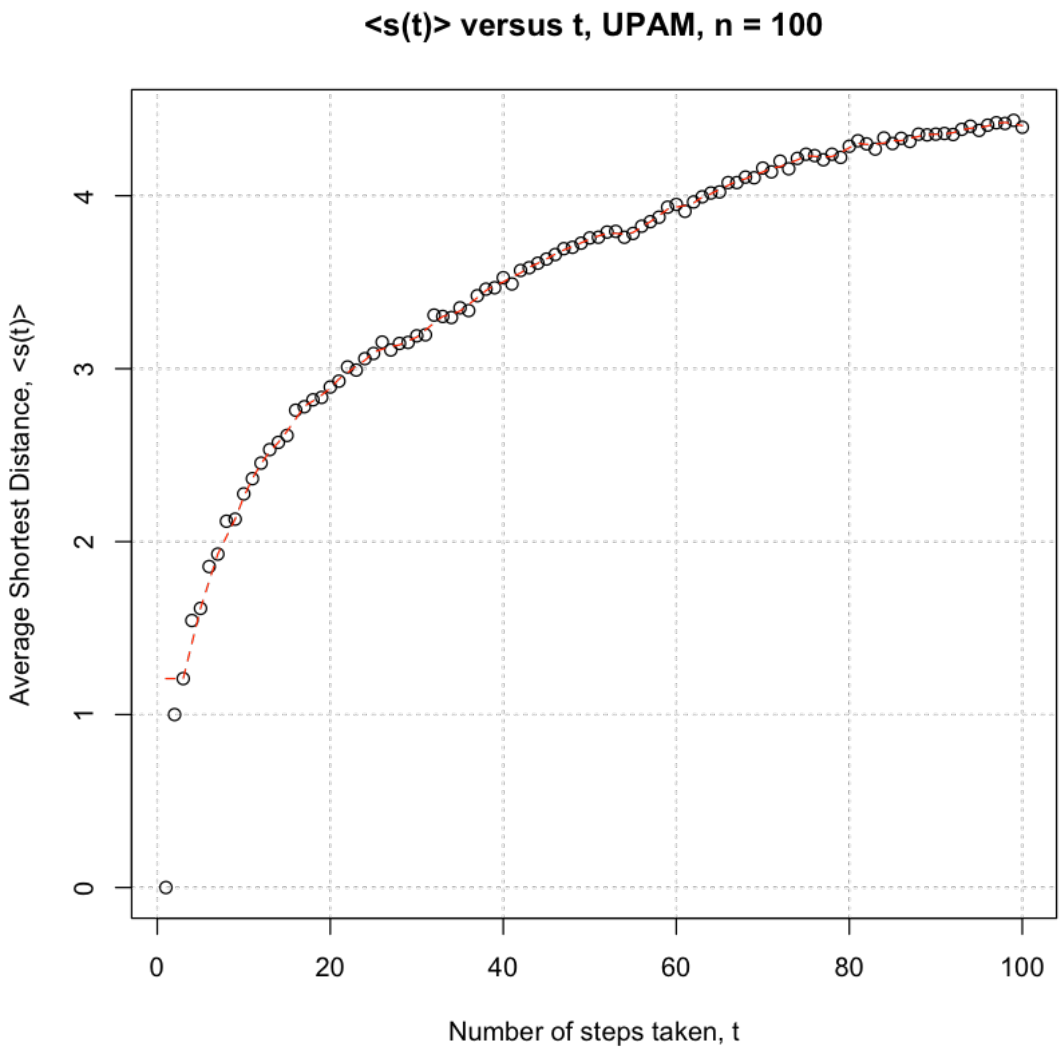
[17]:
```
print(diameter(g))
```

```
[1] 29
```

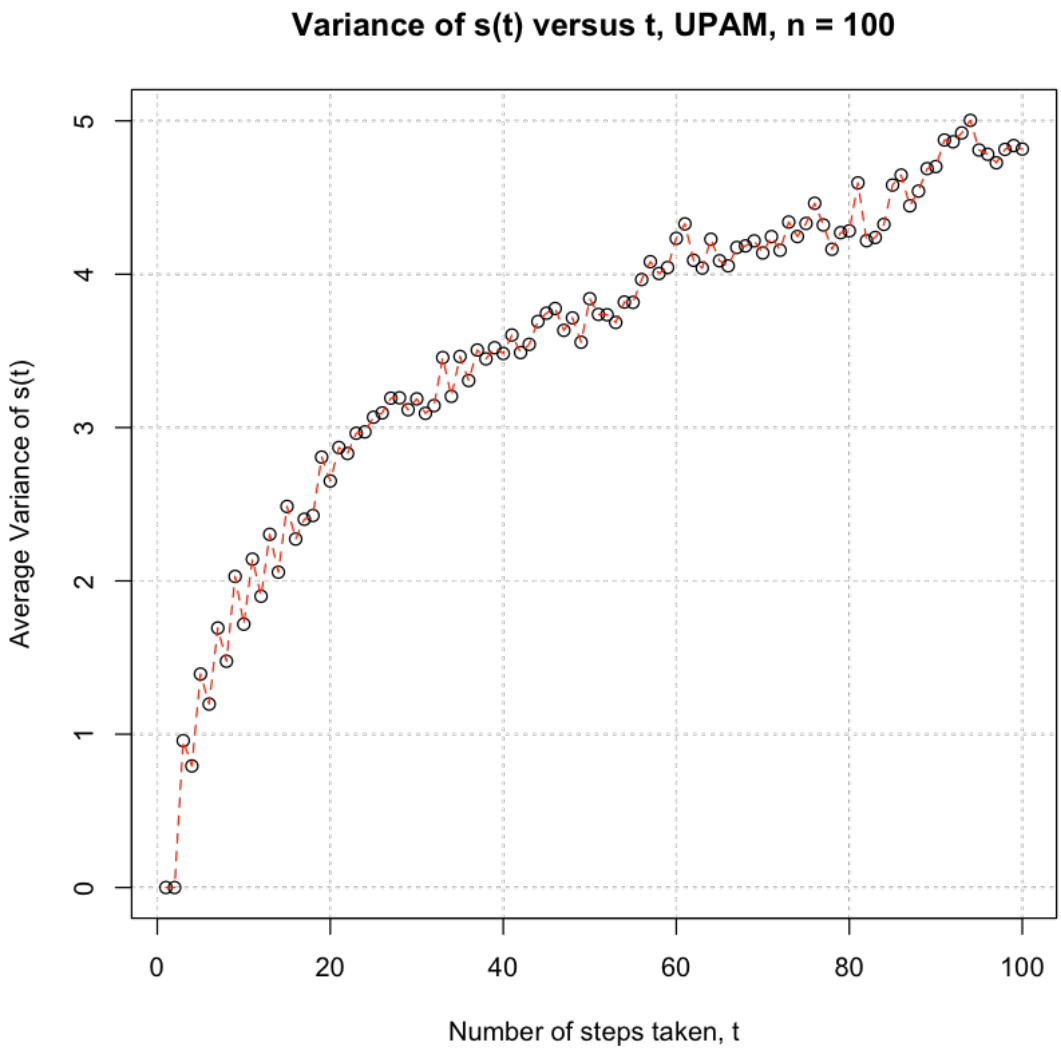
[63]:
```
g <-  barabasi.game(100, m=1, directed=FALSE)
s_t = matrix(data=0.0, nrow=1000, ncol=100)
term_node_deg = matrix(data=0.0,nrow=1000,ncol=1)
for (i in 1:1000){
    if(is_connected(g)){
        gcc <- g
    }
    else{
        g.components <- clusters(g)
```

```
        ix <- which.max(g.components$csize)
        g.giant <- induced.subgraph(g, which(g.components$membership == ix))
        gcc <- g.giant
    }
    start = sample(V(gcc), 1)
    walked_nodes = random_walk(gcc, steps=100, start)
    s_t[i,] = shortest.paths(gcc, walked_nodes, start)
    term_node_deg[i,1] = degree(gcc, walked_nodes[length(walked_nodes)])
}
plot(seq(1,100,1),colMeans(s_t),grid(), xlab = 'Number of steps taken, t', ylab␣
 ↪= 'Average Shortest Distance, <s(t)>',main="<s(t)> versus t, UPAM, n = 100")
lines(lowess(seq(1,100,1),colMeans(s_t),f = 0.05), col="red",lwd = 1,lty=2)
dev.copy2eps(file='2Q2dc.eps')

plot(seq(1,100,1),colVars(s_t),grid(), xlab = 'Number of steps taken, t', ylab␣
 ↪= 'Average Variance of s(t)',main="Variance of s(t) versus t, UPAM, n = 100")
lines(lowess(seq(1,100,1),colVars(s_t),f = 0.03), col="red",lwd = 1,lty=2)
dev.copy2eps(file='2Q2dd.eps')
```

pdf: 2


<s(t)> versus t, UPAM, n = 100
Average Shortest Distance, <s(t)>
Number of steps taken, t
0
1
2
3
4
0
20
40
60
80
100

pdf: 2



[64]:
```
print(diameter(g))
```

```
[1] 13
```

2.8 Question 3a

[102]:
```
create_transition_matrix = function (g){
    vs = V(g)
    n = vcount(g)
    adj = as_adjacency_matrix(g)
    adj[diag(rowSums(adj) == 0)] = 1
    z = matrix(rowSums(adj, , 1))
    transition_matrix = adj / repmat(z, 1, n)
```

```
    return(transition_matrix)
}
```

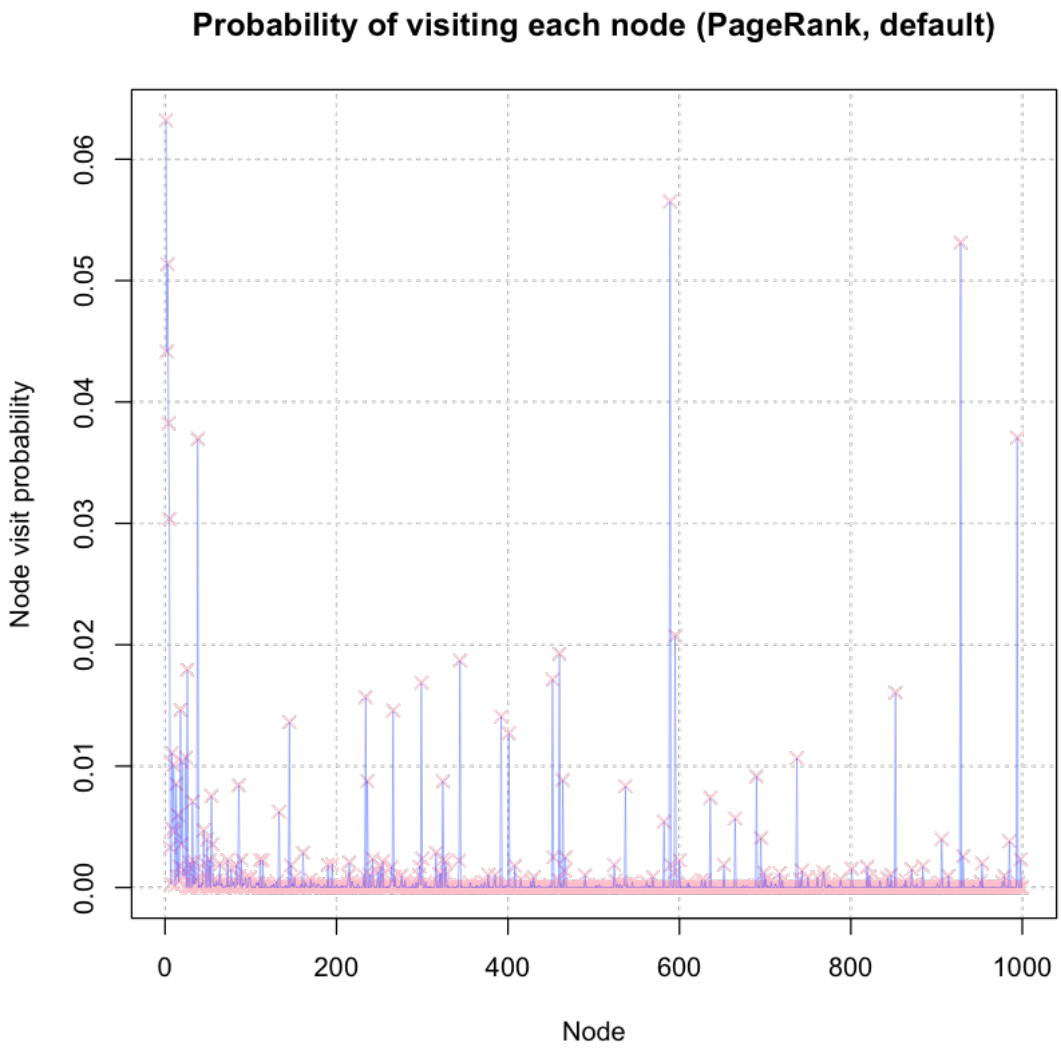
[103]:
```
random_walk_custom = function (g, num_steps, start_node, transition_matrix =␣
↪NULL, alpha=0.0, visit_mode="equal"){
    if(is.null(transition_matrix)){
            transition_matrix = create_transition_matrix(g)
    }
    v = start_node
    walked_nodes = array(data=0.0, length(num_steps))

    if(visit_mode=='equal'){
        visit_probs=array(1/vcount(g), vcount(g))
    }
    else if (visit_mode=="page_rank"){
        pr = page_rank(g, directed=TRUE)$vector
        visit_probs= pr/sum( pr)
    }
    else if (visit_mode=="page_rank_median"){
        pr = page_rank(g, directed=TRUE)$vector
        df = data.frame("idx"=1:vcount(g), "val"=pr)
        df=df[order(df$val),]
        visit_probs=array(0, vcount(g))
        visit_probs[df$idx[(vcount(g) %/% 2):(vcount(g) %/% 2)+1]]=0.5
    }
    else{
        visit_probs=array(1/vcount(g), vcount(g))
    }


    for(i in 1:num_steps){
        if(runif(1)<alpha){
            v = sample(1:vcount(g), 1, prob = visit_probs)
        }
        else{
            PMF = transition_matrix[v, ]
            v = sample(1:vcount(g), 1, prob = PMF)
        }
        walked_nodes[i] = v
    }
    return(walked_nodes)
}
```

[104]:
```
g1 <- barabasi.game(1000, m = 4, directed = TRUE)
g2 <- barabasi.game(1000, m = 4, directed = TRUE)
gf = add_edges(g1, c(t(as_edgelist(permute(g2, sample(vcount(g2)))))))
```

[105]:
```
num_steps = 1000
visited_nodes=array(data=0.0, vcount(gf))
stable_node = ceiling(log(vcount(gf)))
for(i in 1:100){
    start = sample(1:vcount(gf), 1)
    walked_nodes = random_walk_custom(g = gf,num_steps =␣
↪num_steps,transition_matrix = NULL,start_node = start,alpha = 0.0,visit_mode␣
↪= 'equal')
    for (j in 1:length(walked_nodes)) {
        if (j > stable_node) {
            visited_nodes[walked_nodes[j]] = visited_nodes[walked_nodes[j]] + 1
        }
    }
}
node_prob = visited_nodes / ((num_steps-stable_node) * 100)
```

[106]:
```
plot(seq(1,1000,1),node_prob,pch=4, xlab='Node', ylab='Node visit␣
↪probability',main='Probability of visiting each node (PageRank,␣
↪default)',col='pink',grid())
lines(node_prob,lwd=0.3,col="blue")
dev.copy2eps(file='2Q3aa.eps')
```

pdf: 2

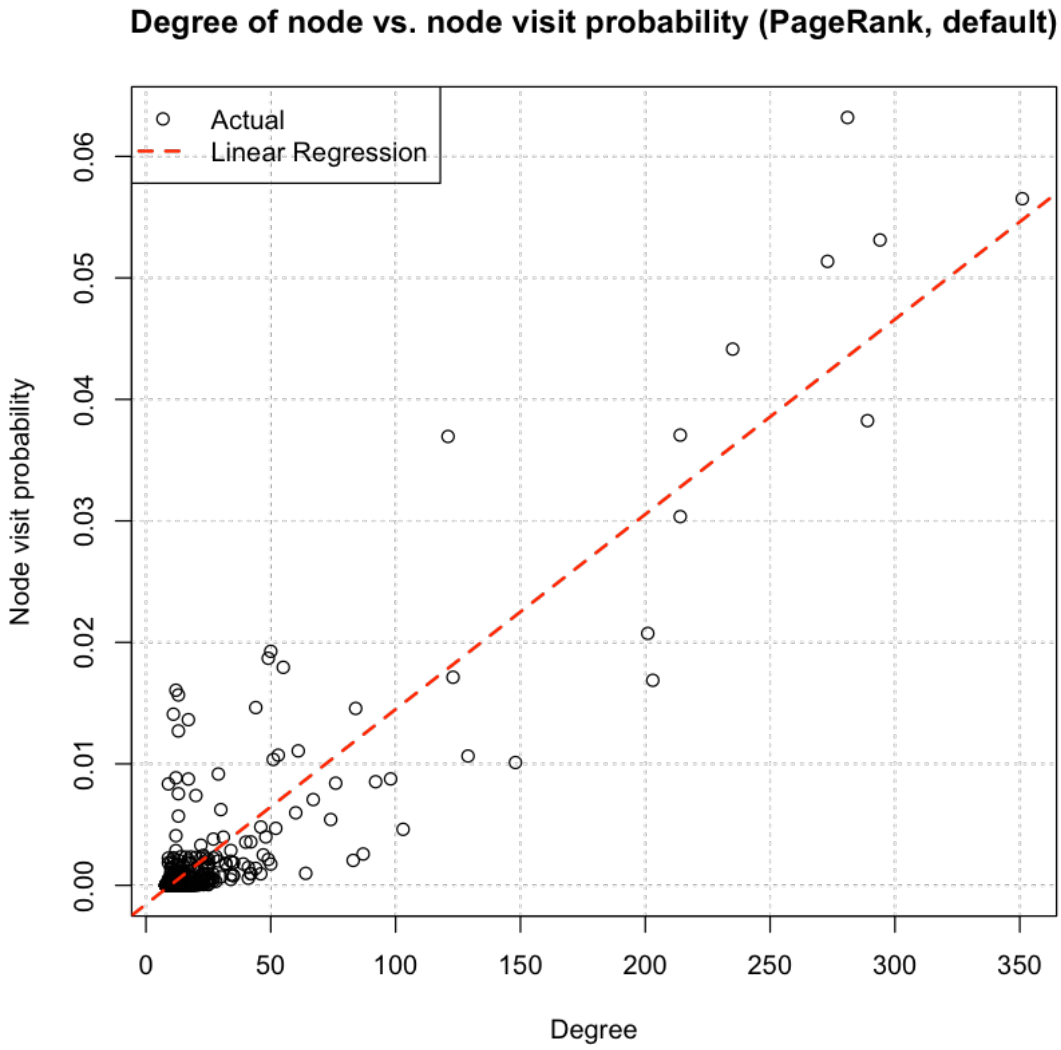


[108]:
```
plot(degree(gf), node_prob , xlab='Degree', ylab='Node visit␣
 ↪probability',main="Degree of node vs. node visit probability (PageRank,␣
 ↪default)",grid())
abline(lm(node_prob ~ degree(gf)),col="red",lwd=2,lty=2)
legend('topleft', legend = c("Actual","Linear Regression"),
       lty = c(NA, 2), lwd = c(1,2), pch=c(1,NA),
       col = c('black','red'))
print(sprintf("Pearson correlation coefficient: %f",cor(degree(gf), node_prob)))
print("Slope and intercept:")
print(lm(node_prob ~ degree(gf)))
dev.copy2eps(file='2Q3ab.eps')
```

```
[1] "Pearson correlation coefficient: 0.902458"
```

```
[1] "Slope and intercept:"

Call:
lm(formula = node_prob ~ degree(gf))

Coefficients:
(Intercept)   degree(gf)
 -0.0015616    0.0001605
```

pdf: 2

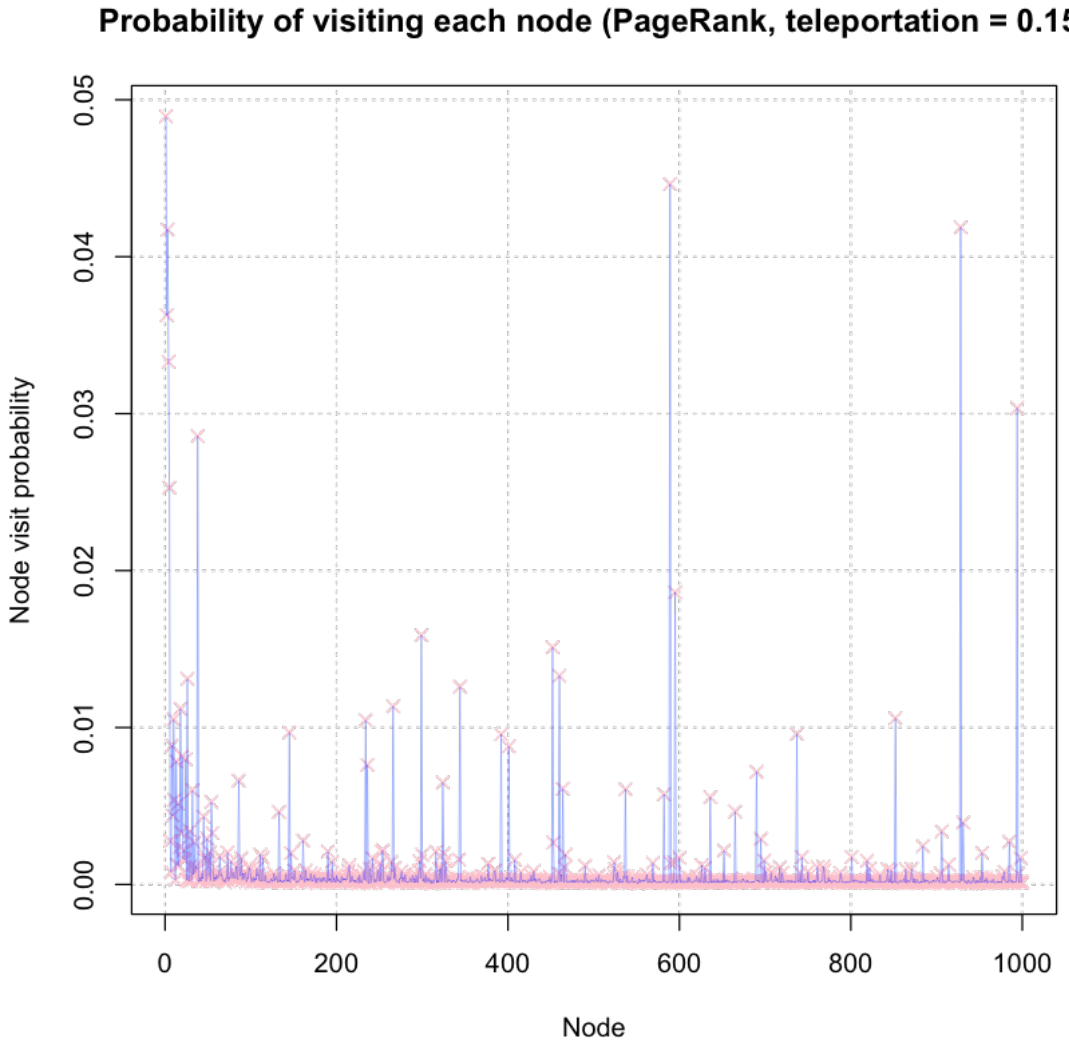


2.9 Question 3b

[109]:
```
num_steps = 1000
visited_nodes=array(data=0.0, vcount(gf))
stable_node = ceiling(log(vcount(gf)))
for(i in 1:100){
    start = sample(1:vcount(gf), 1)
    walked_nodes = random_walk_custom(g = gf,num_steps =␣
↪num_steps,transition_matrix = NULL,start_node = start,alpha = 0.
↪15,visit_mode = 'equal')
    for (j in 1:length(walked_nodes)) {
        if (j > stable_node) {
            visited_nodes[walked_nodes[j]] = visited_nodes[walked_nodes[j]] + 1
        }
    }
}
node_prob = visited_nodes / ((num_steps-stable_node) * 100)
```

[112]:
```
plot(seq(1,1000,1),node_prob,pch=4, xlab='Node', ylab='Node visit␣
↪probability',main='Probability of visiting each node (PageRank,␣
↪teleportation = 0.15)',col='pink',grid())
lines(node_prob,lwd=0.3,col="blue")
dev.copy2eps(file='2Q3ba.eps')
```

pdf: 2

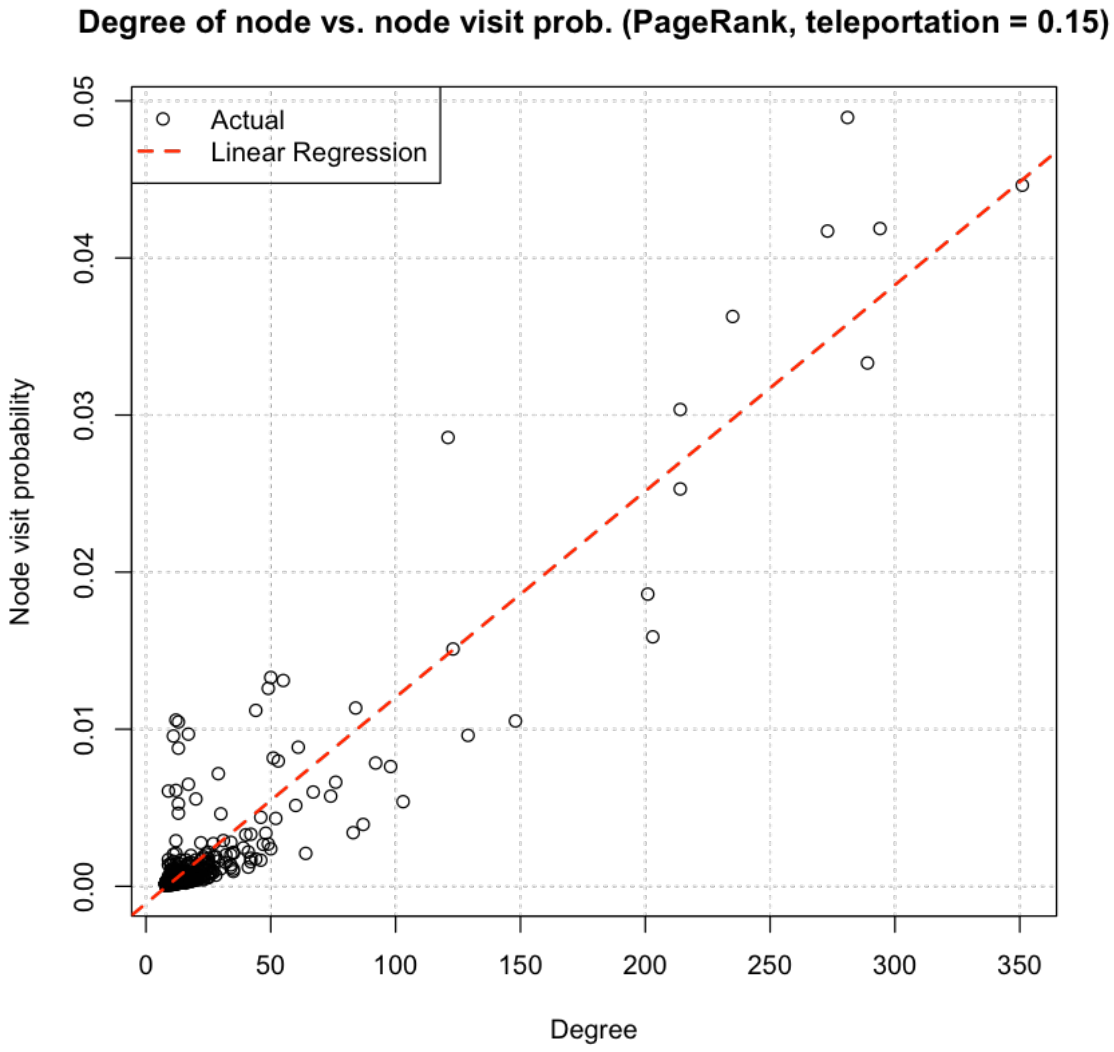


[114]:
```
plot(degree(gf), node_prob , xlab='Degree', ylab='Node visit␣
 ↪probability',main="Degree of node vs. node visit prob. (PageRank,␣
 ↪teleportation = 0.15)",grid())
abline(lm(node_prob ~ degree(gf)),col="red",lwd=2,lty=2)
legend('topleft', legend = c("Actual","Linear Regression"),
       lty = c(NA, 2), lwd = c(1,2), pch=c(1,NA),
       col = c('black','red'))
print(sprintf("Pearson correlation coefficient: %f",cor(degree(gf), node_prob)))
print("Slope and intercept:")
print(lm(node_prob ~ degree(gf)))
dev.copy2eps(file='2Q3bb.eps')
```

```
[1] "Pearson correlation coefficient: 0.930428"
```

```
[1] "Slope and intercept:"

Call:
lm(formula = node_prob ~ degree(gf))

Coefficients:
(Intercept)   degree(gf)
 -0.0010949    0.0001313
```

pdf: 2

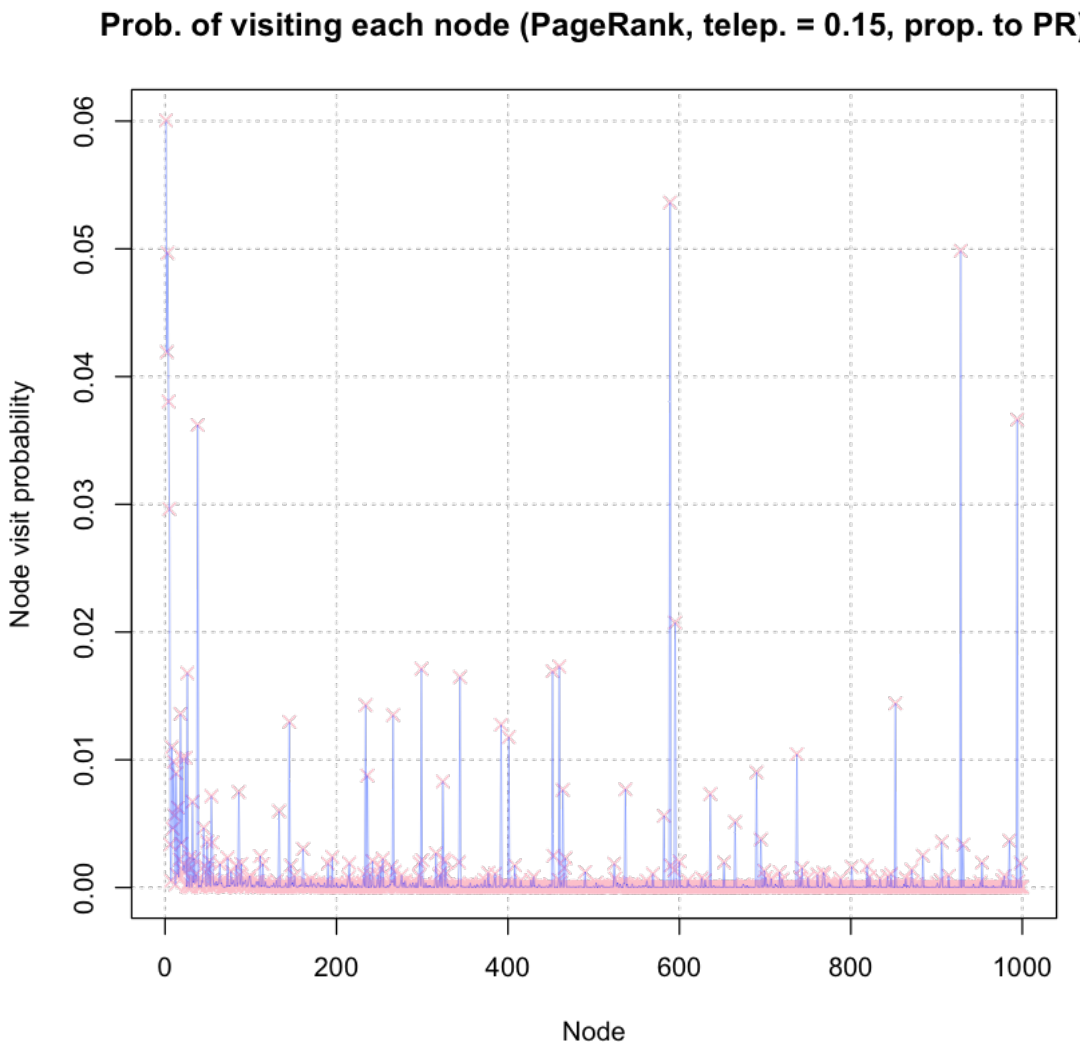


2.10 Question 4a

[115]:
```
num_steps = 1000
visited_nodes=array(data=0.0, vcount(gf))
stable_node = ceiling(log(vcount(gf)))
for(i in 1:100){
    start = sample(1:vcount(gf), 1)
    walked_nodes = random_walk_custom(g = gf,num_steps =␣
↪num_steps,transition_matrix = NULL,start_node = start,alpha = 0.
↪15,visit_mode = 'page_rank')
    for (j in 1:length(walked_nodes)) {
        if (j > stable_node) {
            visited_nodes[walked_nodes[j]] = visited_nodes[walked_nodes[j]] + 1
        }
    }
}
node_prob = visited_nodes / ((num_steps-stable_node) * 100)
```

[116]:
```
plot(seq(1,1000,1),node_prob,pch=4, xlab='Node', ylab='Node visit␣
↪probability',main='Prob. of visiting each node (PageRank, telep. = 0.15,␣
↪prop. to PR)',col='pink',grid())
lines(node_prob,lwd=0.3,col="blue")
dev.copy2eps(file='2Q4aa.eps')
```

pdf: 2

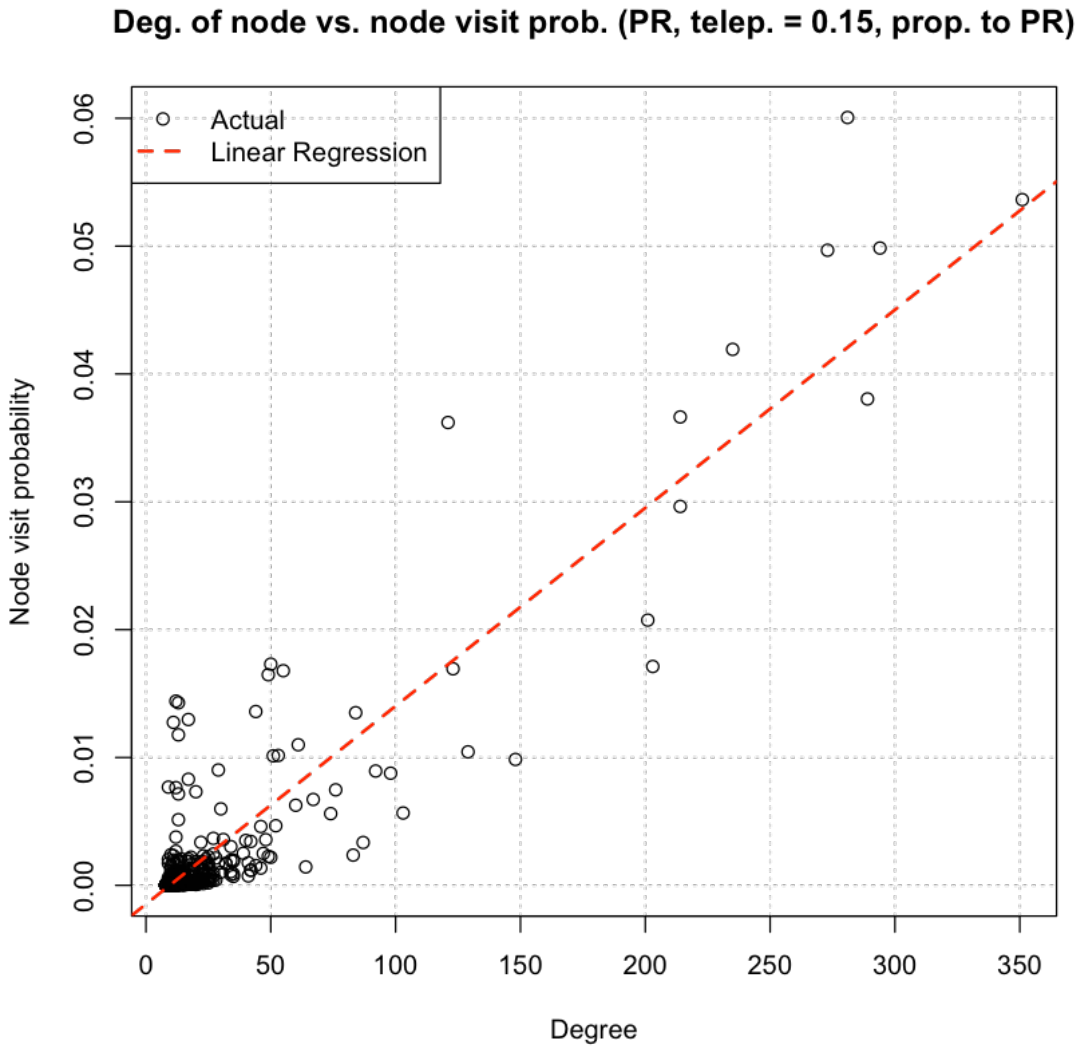


[118]:
```
plot(degree(gf), node_prob , xlab='Degree', ylab='Node visit␣
 ↪probability',main="Deg. of node vs. node visit prob. (PR, telep. = 0.15,␣
 ↪prop. to PR)",grid())
abline(lm(node_prob ~ degree(gf)),col="red",lwd=2,lty=2)
legend('topleft', legend = c("Actual","Linear Regression"),
       lty = c(NA, 2), lwd = c(1,2), pch=c(1,NA),
       col = c('black','red'))
print(sprintf("Pearson correlation coefficient: %f",cor(degree(gf), node_prob)))
print("Slope and intercept:")
print(lm(node_prob ~ degree(gf)))
dev.copy2eps(file='2Q4ab.eps')
```

```
[1] "Pearson correlation coefficient: 0.910936"
```

```
[1] "Slope and intercept:"

Call:
lm(formula = node_prob ~ degree(gf))

Coefficients:
(Intercept)   degree(gf)
  -0.001473     0.000155
```

pdf: 2

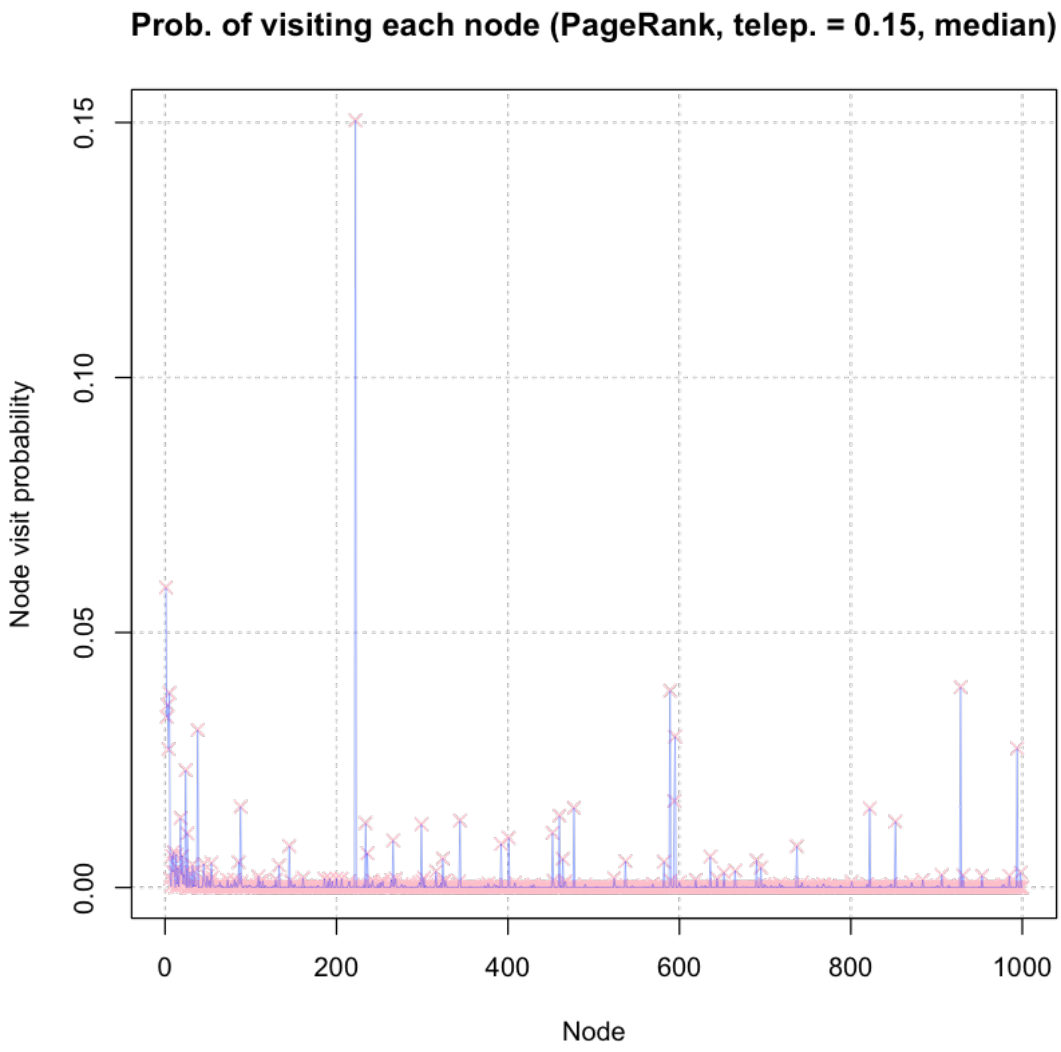


2.11 Question 4b

[119]:
```
num_steps = 1000
visited_nodes=array(data=0.0, vcount(gf))
stable_node = ceiling(log(vcount(gf)))
for(i in 1:100){
    start = sample(1:vcount(gf), 1)
    walked_nodes = random_walk_custom(g = gf,num_steps =␣
↪num_steps,transition_matrix = NULL,start_node = start,alpha = 0.
↪15,visit_mode = 'page_rank_median')
    for (j in 1:length(walked_nodes)) {
        if (j > stable_node) {
            visited_nodes[walked_nodes[j]] = visited_nodes[walked_nodes[j]] + 1
        }
    }
}
node_prob = visited_nodes / ((num_steps-stable_node) * 100)
```

[120]:
```
plot(seq(1,1000,1),node_prob,pch=4, xlab='Node', ylab='Node visit␣
↪probability',main='Prob. of visiting each node (PageRank, telep. = 0.15,␣
↪median)',col='pink',grid())
lines(node_prob,lwd=0.3,col="blue")
dev.copy2eps(file='2Q4ba.eps')
```

pdf: 2

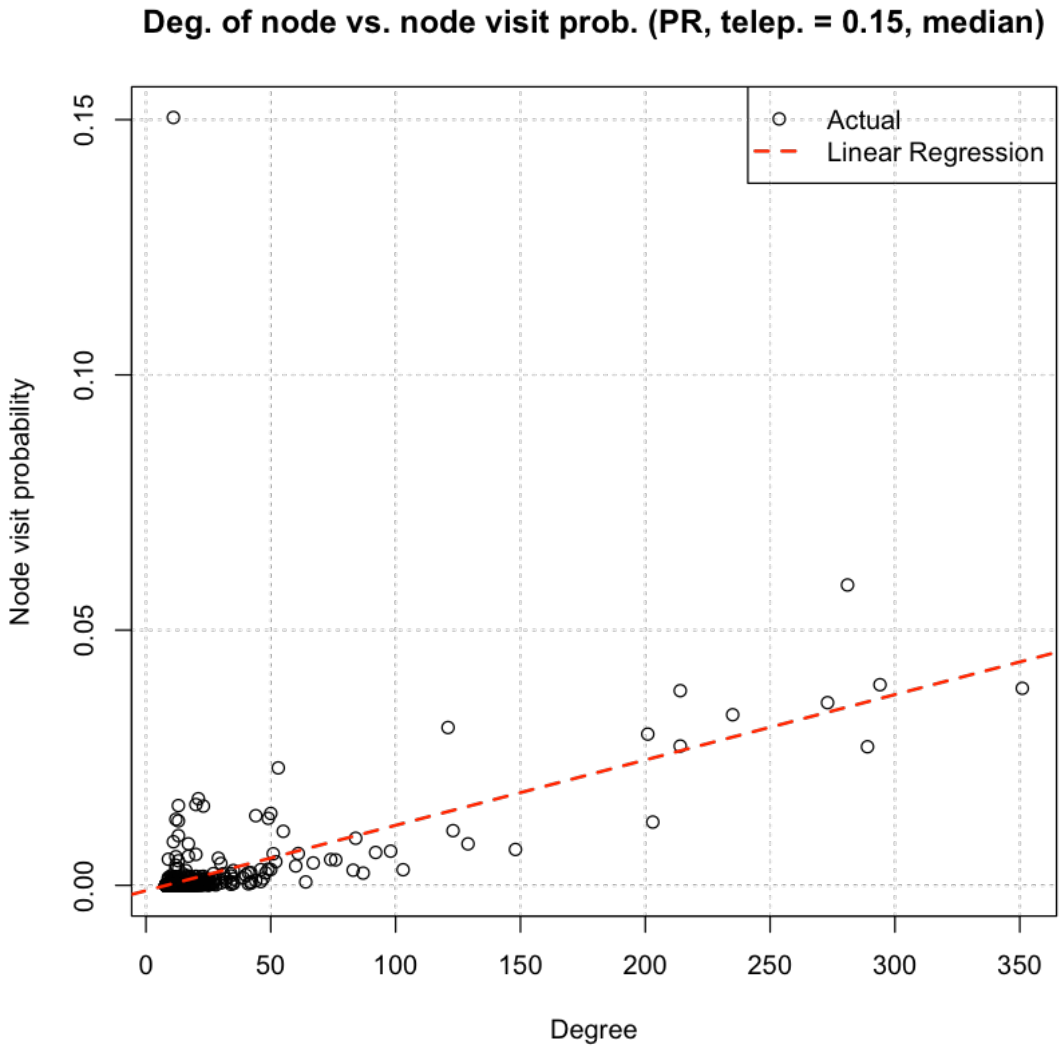


[122]:
```
plot(degree(gf), node_prob , xlab='Degree', ylab='Node visit␣
 ↪probability',main="Deg. of node vs. node visit prob. (PR, telep. = 0.15,␣
 ↪median)",grid())
abline(lm(node_prob ~ degree(gf)),col="red",lwd=2,lty=2)
legend('topright', legend = c("Actual","Linear Regression"),
       lty = c(NA, 2), lwd = c(1,2), pch=c(1,NA),
       col = c('black','red'))
print(sprintf("Pearson correlation coefficient: %f",cor(degree(gf), node_prob)))
print("Slope and intercept:")
print(lm(node_prob ~ degree(gf)))
dev.copy2eps(file='2Q4bb.eps')
```

```
[1] "Pearson correlation coefficient: 0.563365"
```

```
[1] "Slope and intercept:"

Call:
lm(formula = node_prob ~ degree(gf))

Coefficients:
(Intercept)   degree(gf)
  -0.001044     0.000128
```

pdf: 2



[170]:
```
pageranks = page_rank(gf)$vector
```

```
plot(seq(1,1000,1),pageranks,col="blue",type='b',pch=4,ylim=c(0,0.16),␣
 ↪xlab='Node', ylab='Node visit probability (Page rank)')
points(node_prob, col="red",type='b',lty=2,pch=4)
```