SDS - Homework1 B

Group 00: Michele Meo, Lorenzo Ceccomancini

People have the power… law!

Network simulation and analysis

We want to simulate the structure of a big network using a preferential attachment mechanism in order to study some features of the real networks, such as the WWW. Since the number of nodes needed to simulate a real network is high, in this case 10^6, we decided to represent the network through a single vector. This vector is composed by the output node of each edge of the graph, obviously this is possible in this special case, where each node of the graph connects to a single another node: since each node is created in sequence, if we need to do it, we are able to reconstruct the graph by looking at this *output_links* vector and the sequence of integer numbers $\{1, \dots, 10^6\}$. This simple structure allows us to build the network with 10^6 nodes in an efficient way.

Using this approach we just risk to lose information about nodes with null degree, but we will see that we can evaluate their total number and we just need this information.

In the following function we create the network using the structure mentioned before and implementing a preferential attachment mechanism, where we sample, with equal probability, the new attached node among the node already present in the network or among the output node of an already existing edge. This allows us to give a heavier weight to the nodes with an higher degree at each sample step of the simulation.

input: *N* the number of nodes

output: *edges* a vector composed by the output of each edge of network

```
create_Net = function(N){

  edges = c(2, 3, 4, 1)

  for( node in 5:N){

    coin = sample(c(TRUE, FALSE), 1, replace=TRUE, prob = c(.5, .5))

    if( coin == TRUE){

      attached_node = sample(1:(node-1), 1, replace=TRUE)
      edges[node] = attached_node

    } else {

      copied_output_node = sample(edges, 1, replace=TRUE)
      edges[node] = copied_output_node
    }

    #if(node%%1000 == 0) print( cat(node, "nodes attached!"))
  }

  return(edges)
}
```

In the following function we evaluate all the features we need to set up a suitable analysis of the network. First of all we need the number of nodes related to each degree, we compute it by using the **table** function. We can even recover the total number of nodes with null degree, that is *N - norm_factor* in the code. Then we decided to work on relative frequencies of each nodes number per degree, so we normalize all by *N*. Finally we evaluate the logarithm and the cumulative of degree and frequency to after fit a power law distribution and plot the complimentary cumulative function.

input: *edges* the vector of output's links generated with the function above

output: *nodes_per_degree* a dataframe with all the features we need for the analysis

```
degree_data = function(edges){

  degree = table(edges)
  norm_factor = length(degree)
  nodes_per_degree = table(degree)

  nodes_per_degree = as.data.frame(sort(nodes_per_degree, decreasing=TRUE))

  nodes_per_degree$norm_freq = nodes_per_degree$Freq/N

  nodes_per_degree$log_degree = log10(as.double(nodes_per_degree$degree))
  nodes_per_degree$log_freq = log10(as.double(nodes_per_degree$norm_freq))

  norm_freq_for_cumulative = nodes_per_degree$norm_freq
  norm_freq_for_cumulative[1] = norm_freq_for_cumulative[1] + (N-norm_factor)/N
  nodes_per_degree$cum = cumsum(norm_freq_for_cumulative)

  return(nodes_per_degree)
}
```

We report the function used to plot the empirical degree distribution in a log-log plot. Note that in a log-log plot the degree 0 has no sense, so we don't need information about this nodes in this specific case.

```
loglog_distro = function(df, slope, intercept, lam, colore){

  p = ggplot(df, aes(x = as.double(degree), y = as.double(norm_freq))) +
    geom_point(colour=colore, alpha=0.7)

  plaw_fit = function(x) (intercept)*x^(slope)

  p = p + geom_function(fun = plaw_fit, colour=colore) +
    labs(title="Degree distribution", x="Degree K", y="# nodes with degree K" ) +
    scale_x_log10(breaks = trans_breaks("log10", function(x) 10^x),
                  labels = trans_format("log10", math_format(10^.x))) +
    scale_y_log10(breaks = trans_breaks("log10", function(x) 10^x),
                  labels = trans_format("log10", math_format(10^.x))) +
    annotation_logticks() + theme_bw()
  print(p)
}
```

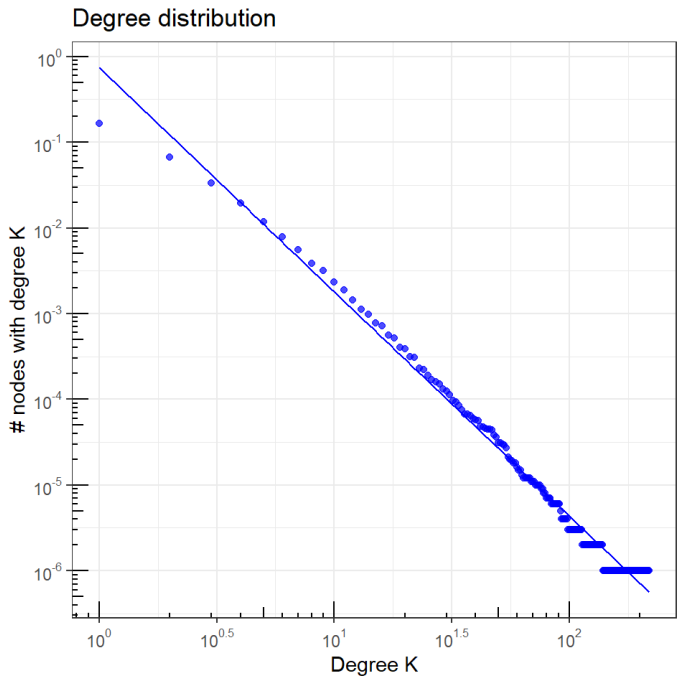
We report the function used to plot the complimentary cumulative degree distribution. Note that here we need the total number of nodes with degree 0, that we rebuilt above, to plot the CCDF. Given K the degree random variable, since we decided to define the CDF as $CDF = P(K \leq k)$ for a specific k, we plot the complimentary cumulative degree distribution as $CCDF = P(K > k)$.

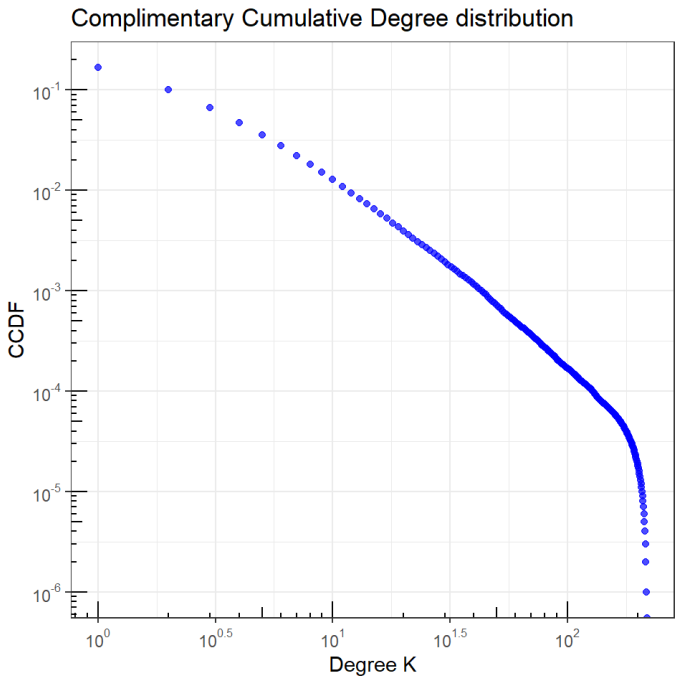
```
loglog_cum = function(df, colore){

  p = ggplot(df, aes(x = as.double(degree),
                     y = 1 - as.double(cum))) +
    geom_point(colour=colore, alpha=0.7) +
    labs(title="Complimentary Cumulative Degree distribution",
         x="Degree K", y="CCDF" ) +
    scale_x_log10(breaks = trans_breaks("log10", function(x) 10^x),
                  labels = trans_format("log10", math_format(10^.x))) +
    scale_y_log10(breaks = trans_breaks("log10", function(x) 10^x),
                  labels = trans_format("log10", math_format(10^.x))) +
    annotation_logticks() + theme_bw()

  print(p)
}
```

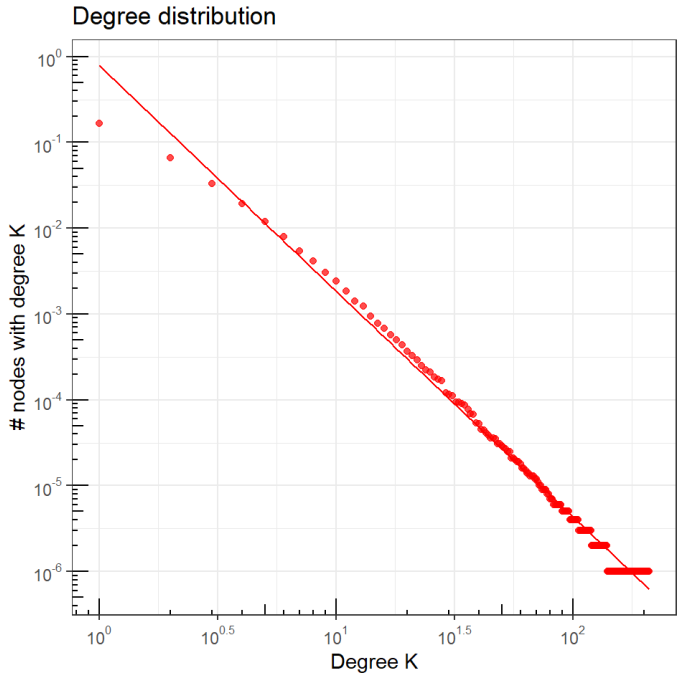
The main part of the code is in the following cell, where we run all the functions defined above in order to create the network, study the network structure through some plots and try to fit the degree distribution with a discrete power law distribution and a poisson distribution. In order to evaluate the power law parameters, we can use the log-log transformation to infer from a linear fit the couple of parameters $(1/\zeta(s), -s)$. Instead, for the poisson distribution we can just evaluate its parameter by $\mu = \mathbb{E}(K)$.

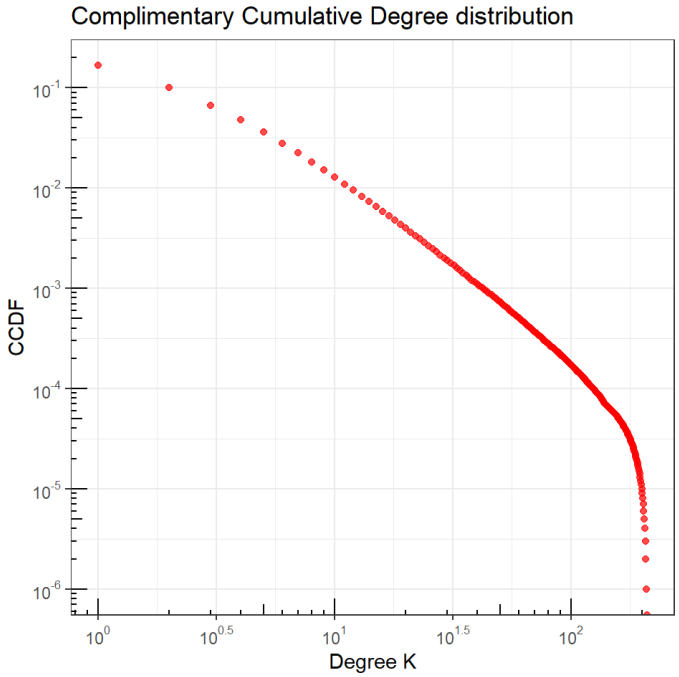
Finally, we plot on each row the degree distribution with its related power law “line” and, next, its CCDF obtained by a single run. We do it for 3 trials.

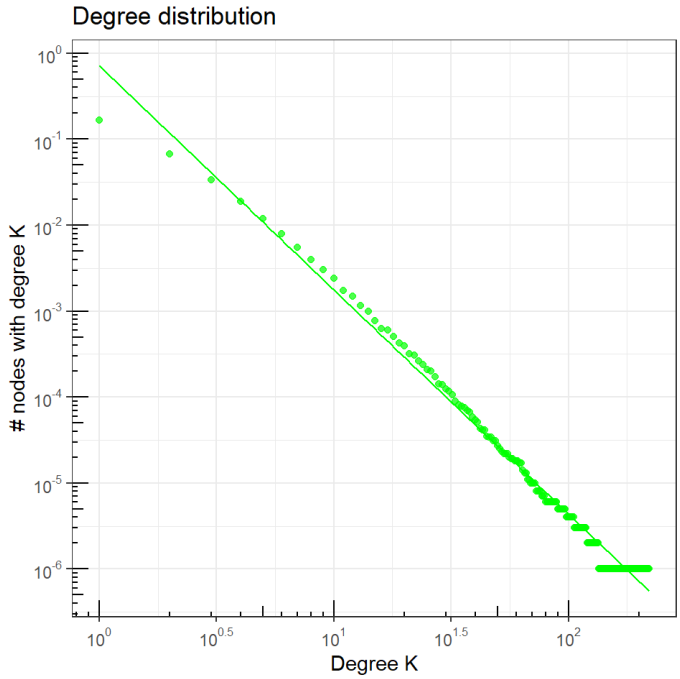
```
set.seed(34745647)
trials = 3
N = 10^6

for(k in 1:trials){

  output_links = create_Net(N)

  mu = mean(table(output_links))   # poisson parameter

  data = degree_data(output_links)
  linearFit = lm(log_freq ~ log_degree, data=data)

  plot_colors = c(rgb(0,0,1), rgb(1,0,0), rgb(0,1,0))

  loglog_distro(data, linearFit$coefficients[2], 10^linearFit$coefficients[1],
                mu,plot_colors[k])
  loglog_cum(data, plot_colors[k])
}
```

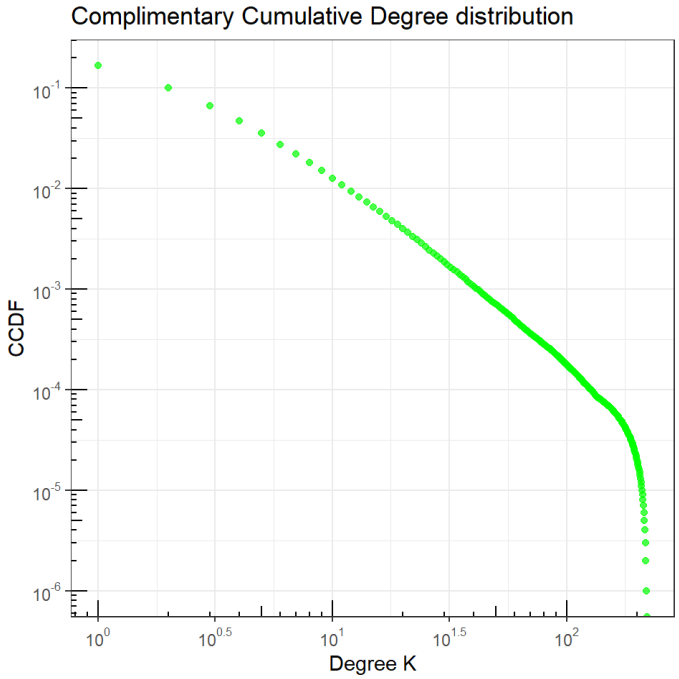

Degree distribution
nodes with degree K
Degree K


Complimentary Cumulative Degree distribution
CCDF
Degree K


Degree distribution
nodes with degree K
Degree K


Complimentary Cumulative Degree distribution
CCDF
Degree K


Degree distribution
nodes with degree K
Degree K


Complimentary Cumulative Degree distribution
CCDF
Degree K

We can see that data seems to follow very well a power law distribution that explains the greater presence of nodes with low grade but it does not preclude the possibility of having nodes with a very high grade, as in many real cases. Instead, the poisson distribution associates a probability to the higher degree so low that these rare cases are virtually impossible: we will try to numerical visualize this after with a statistical test, since we didn't manage to plot the poisson distribution relative to each run with the **ggplot** library.

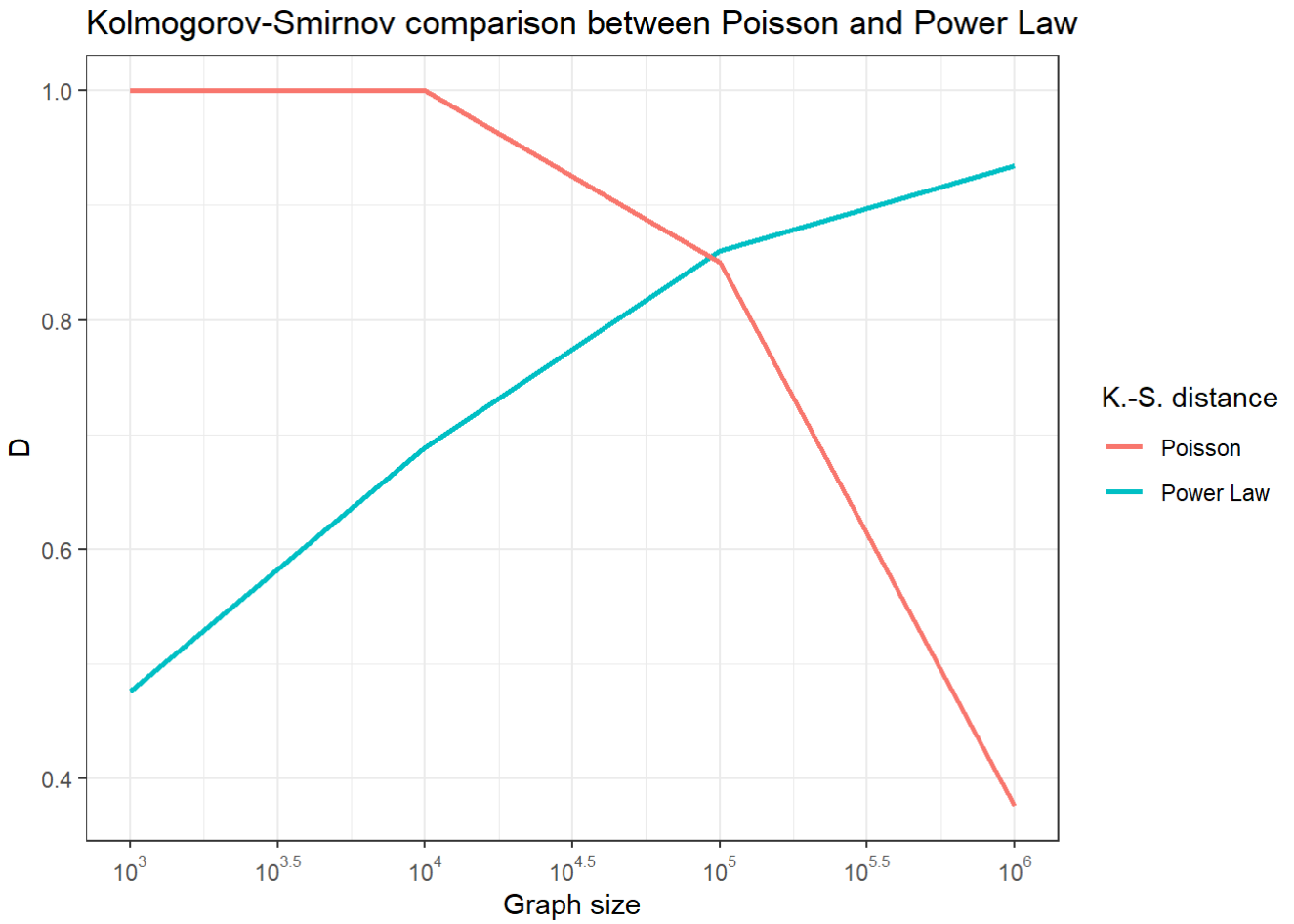
It's even better try to visualize the difference between the popularity of low degree nodes and the rarity of high degree nodes from the CCDF plots, where we can see a significant curvature near the high degrees even in these log-log plots. Moreover, when this high degree nodes are present in the network, they will continue to increase their degree because of the mechanism of preferential attachment.

Power Law VS Poisson distribution

we compare the ability to describe the structure of large networks by comparing power law distribution and poisson distribution numerically through the Kolmogorov-Smirnov test. For the power law distribution we used the library **poweRlaw**. We decided to implement this analysis on different size of the network, in order to understand when each distribution is more suitable then the other or if one distribution is ever better than the other.

The Kolmogorov-Smirnov test evaluates the following distance: $D = sup_x |F_n(x) - F(x)|$, where F_n is the Empirical CDF while the F is the CDF given by a specific distribution on which we want to test our ECDF. So we have to evaluate, for each given network size, the ECDF and the poisson/power_law best fit on data in order to build the test CDF.

```
size = c(10^3, 10^4, 10^5, 10^6)
poisson_D = c()
pl_D = c()

iter = 1

for(s in size){

  output_links = create_Net(s)
  mu = mean(table(output_links))

  data = degree_data(output_links)
  linearFit = lm(log_freq ~ log_degree, data=data)

  test_pois = ks.test(data$cum, ppois(as.double(data$degree), lambda=mu))
  test_pl = ks.test(data$cum, ppldis(as.double(data$degree), xmin=1, alpha=-linearFit$coeffic
ients[2]))

  poisson_D[iter] = test_pois$statistic
  pl_D[iter] = test_pl$statistic

  iter = iter+1
}

test_df = data.frame(size, poisson_D, pl_D)

p = ggplot()+
    geom_line(data=test_df,aes(y=poisson_D, x=size,colour="green"), size=1 )+
    geom_line(data=test_df,aes(y=pl_D, x=size,colour="blue"), size=1) +
    scale_color_discrete(name = "K.-S. distance", labels = c("Poisson", "Power Law")) +
    labs(title="Kolmogorov-Smirnov comparison between Poisson and Power Law",
         x="Graph size", y="D") +
     scale_x_log10(breaks = trans_breaks("log10", function(x) 10^x),
                   labels = trans_format("log10", math_format(10^.x))) + theme_bw()
plot(p)
```

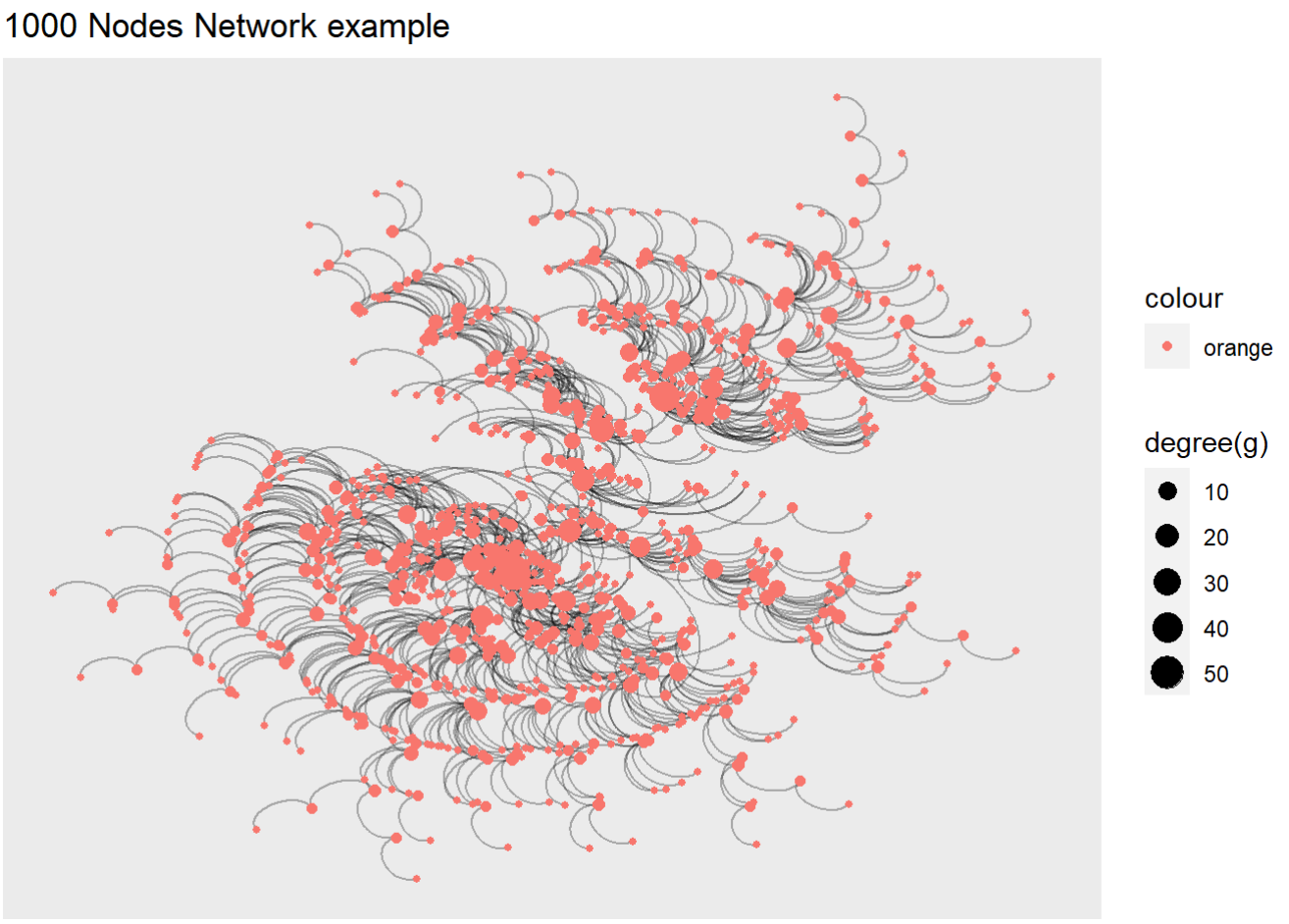


The plot seems to suggest that for not too large network the Poisson distribution describes better the data, instead for bigger network such as 10^5 or 10^6 nodes the power law distribution seems to fit in the best way. We think that for even larger graphs the distance would continue to fall for the power law and to rise for the poisson distribution but we can't demonstrate it for computational limitations.

Graph Visualization

The graph visualization for 10^6 nodes is computationally heavy, so we decided to report the example of a graph with 10^3 nodes built with the function *create_Net*, in which we simulate the preferential attachment. In order to do this we need to rebuild all the information about the graph from the very essential structure that we gave in that function and we did it as described in the beginning of the report.

```
library(ggraph)
N = 1000
nodes = 1:N
links= c()

output_links = create_Net(N)

for(i in 1:N){
  links[2*i-1] = nodes[i]
  links[2*i] = output_links[i]
}

g = make_graph(links, n=N, directed=T)

ggraph(g, layout='kk') + ggtitle("1000 Nodes Network example") +
  geom_edge_arc(alpha=0.3) +
  geom_node_point(show.legend=T, aes( colour='orange',
                                      size=degree(g)))
```



Even with just 10^3 nodes we can start to appreciate the rarity of very high degree nodes with respect to the low degree nodes, which are shown in the outermost layers of this graph.