Social Network Analysis Home Assignment 2

Shupeyko Nikita

due date - 15.05.2016 23:59

Contents

Graph models. Centrality metrics **1**

Task 1. Your social network 1

Task 2. Flickr network 12

Graph models. Centrality metrics

Task 1. Your social network

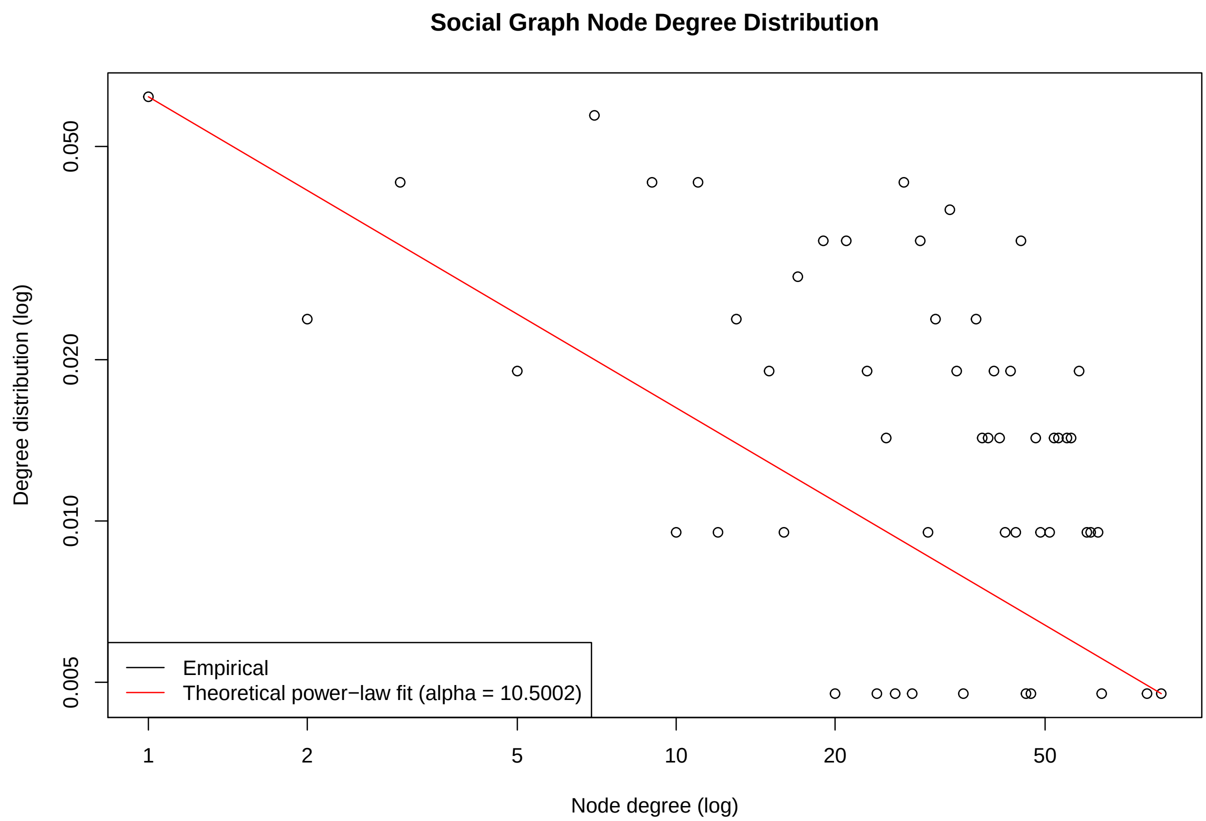
For the first task, you have to load your vk.com network. Please follow the instructions posted on the course wiki or user VK Application. For FB users try to use NetVizz. If you did it correctly, you should have a GraphML file with your own network. Read it to R:

```
# Social graphs ("Social Graph") acquired through
# [Export friends graph](vk.com/app3861133) VK app.

# Friends graph relative path
# (note the file is not available on remote due to privacy issues --
# feel free to generate one on your own):
g_vkf.rel.path = "vk-friends-20049226.gexf"
# Friends-and-followers graph relative path:
g_vkff.rel.path = ""

g.rel.path = g_vkf.rel.path

g = gexf.to.igraph(read.gexf(g.rel.path))

g.n = vcount(g)
g.m = ecount(g)
g.avg.deg = 2*g.m / g.n

g.v.names = names(V(g))

# 'Anonimizing' g:
g.adj.mat = get.adjacency(g)
g.adj.mat.names = 1:g.n
colnames(g.adj.mat) = g.adj.mat.names
rownames(g.adj.mat) = colnames(g.adj.mat)
g = graph.adjacency(g.adj.mat, add.colnames = T)

g.deg = degree(g)
```

1. Degree distribution

First, plot degree distribution of your network in log-log scales:

```
# For further power-law fitting correctness, let degree n <-> degree n+1:
g.deg.plus1 = g.deg + 1

# Obtaining empirical degree distribution:
g.deg.plus1.rng.min = min(g.deg.plus1)
g.deg.plus1.rng.max = max(g.deg.plus1)
g.deg.plus1.rng = as.numeric(g.deg.plus1.rng.min:g.deg.plus1.rng.max)
g.deg.plus1.dist = degree.distribution(g)[g.deg.plus1.rng]

# Constructing theoretical power-law degree distribution:
g.pl.fit = power.law.fit(g.deg.plus1, implementation = "plfit")
func_pl = function(x)
    return ((g.pl.fit$alpha-1) * g.pl.fit$xmin^(g.pl.fit$alpha-1) * x^(-g.pl.fit$alpha))

# For the sake of clarity, the empirical and theoretical
# degree distributions are plotted side-by-side
plot(g.deg.plus1.dist, log = "xy",
     main = "Social Graph Node Degree Distribution", xlab = "Node degree (log)",
     ylab = "Degree distribution (log)")
par(new = T)
curve(func_pl, from = g.deg.plus1.rng.min, to = g.deg.plus1.rng.max, log="xy",
      main = "", xlab = "", ylab = "", col = "red", axes = F, add = F)
par(new = F)
legend("bottomleft",
       legend = c("Empirical", sprintf("Theoretical power-law fit (alpha = %.4f)", g.pl.fit$alpha)),
       col = c('black', 'red'), lty = 1)
```



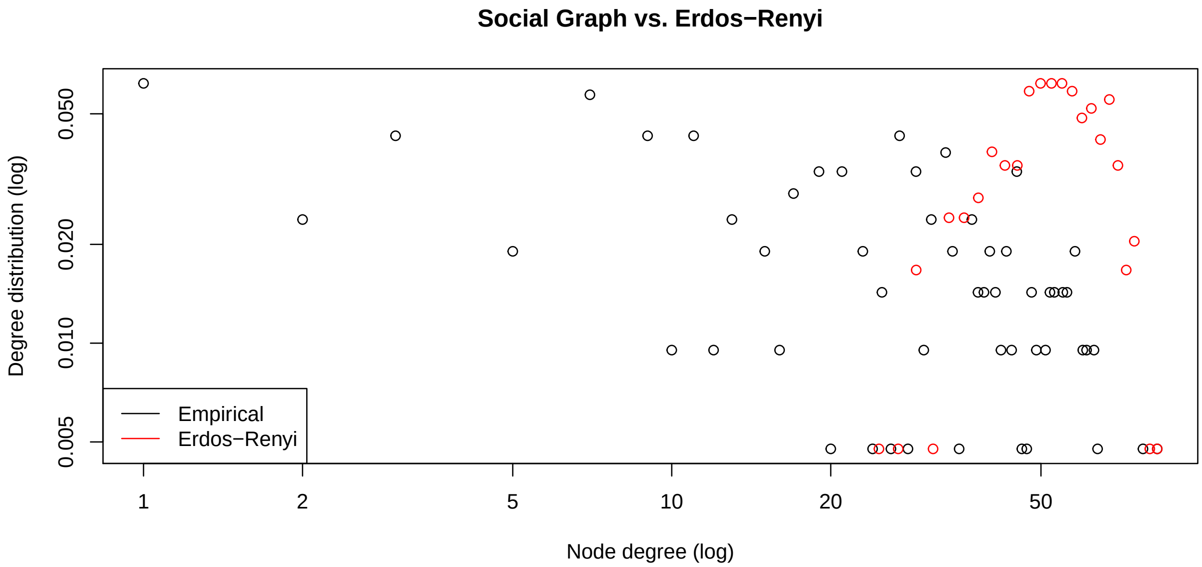
Is there any correspondence between actual degree distribution of your network and the Power Law distribution? If not, explain why.

*According to KS test results, **Power Law fits Social Graph** node degree distribution **with the probability of 0.9994**.*

Now, let's see how it would look if it was random. Produce Erdos-Renyi graph matching your real network (same number of nodes and same average degree). Compare it with your degree distribution.

```
# Since both the original and the produced graph are required to have the same number
# of nodes and average node degree, the number of Erdos-Renyi graph edges, by necessity,
# equals the latter in the original graph.
g_erg = sample_gnm(n = g.n, m = g.m, directed = T)

g_erg.n = vcount(g_erg)
g_erg.m = ecount(g_erg)

g_erg.deg.dist = degree.distribution(g_erg)

plot(g.deg.plus1.dist, log = "xy",
     main = "Social Graph vs. Erdos-Renyi", xlab = "Node degree (log)",
     ylab = "Degree distribution (log)")
par(new = T)
plot(g_erg.deg.dist, log="xy",
     main = "", xlab = "", ylab = "", col="red", axes = F)
par(new = F)
legend("bottomleft", legend = c("Empirical", "Erdos-Renyi"),
```

```
    col = c('black', 'red'), lty = 1)
```



2. Compute centrality metrics

Compute for your network:

- degree centrality
- closeness centrality
- betweenness centrality
- eigenvector centrality
- Bonacich power centrality
- Alpha centrality

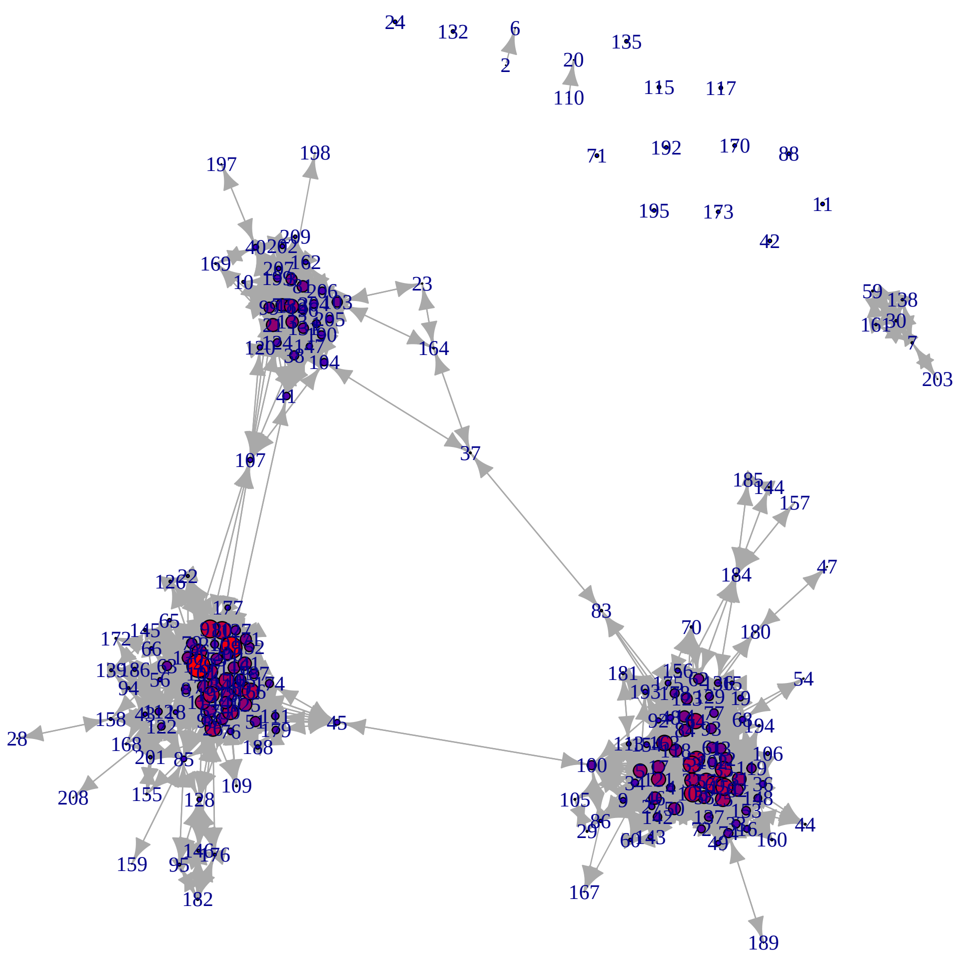
```
g.deg.cent = g.deg
g.cls.cent = closeness(g)
g.btw.cent = betweenness(g)
g.ev.cent = eigen_centrality(g)
g.bp.cent = bonpow(g)
g.alpha.cent = alpha.centrality(g)
```

Output six plots corresponding to six centrality metrics you've computed:

- Use first names of your friends as node labels on the graph (you may hide this information if you wish – change it by integer ID).
- Keep the same layout of the network.
- Make node sizes and colours proportional to the respective centrality metrics.

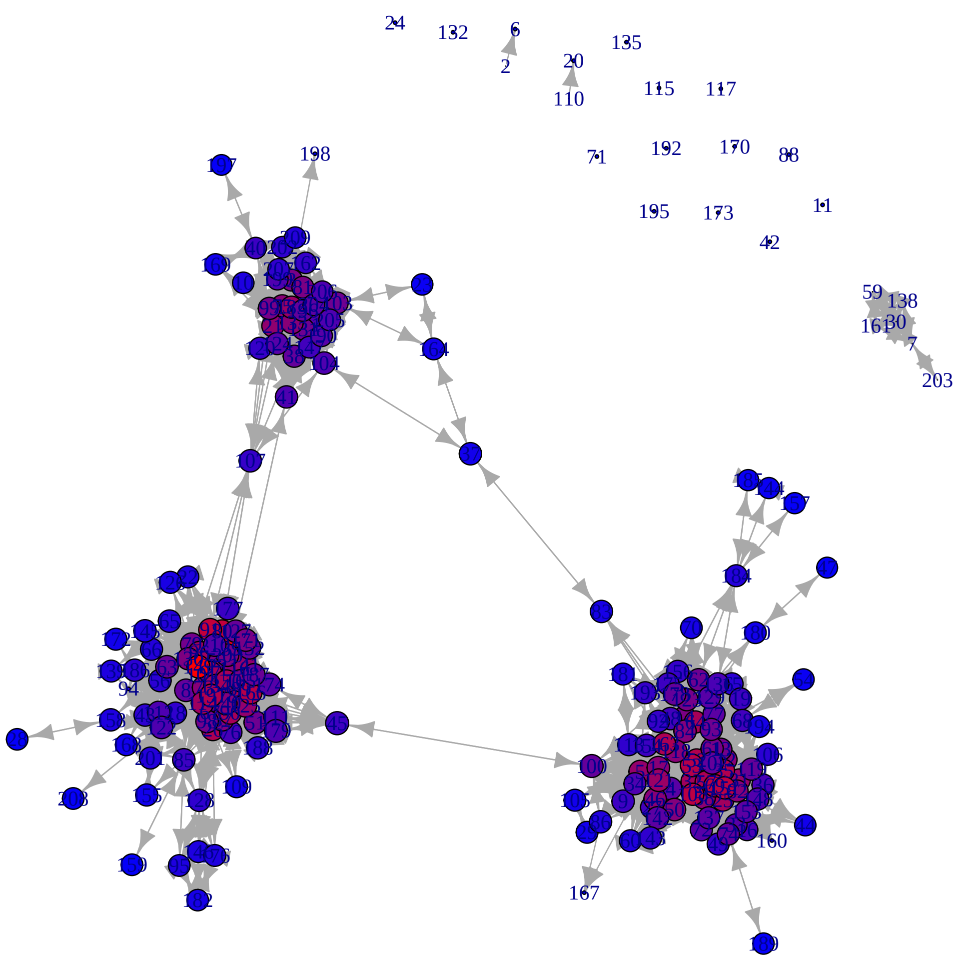
```r
g.cent.th = 100
g.cent.palette = colorRampPalette(c("blue", "red"))
g.cent.lyt = layout.fruchterman.reingold(g)
g.cent.vclr = g.cent.palette(g.cent.th)[as.numeric(cut(g.deg.cent, breaks = g.cent.th))]

func_scale01 = function(l) scale(l, center = min(l), scale = max(l) - min(l))
func_cent.vert.size = function(l) 0 + 5 * func_scale01(l)

plot(g, layout = g.cent.lyt,
     main = "Social Graph Degree Centrality",
     vertex.color = g.cent.vclr, vertex.size = func_cent.vert.size(g.deg.cent))
```

Social Graph Degree Centrality



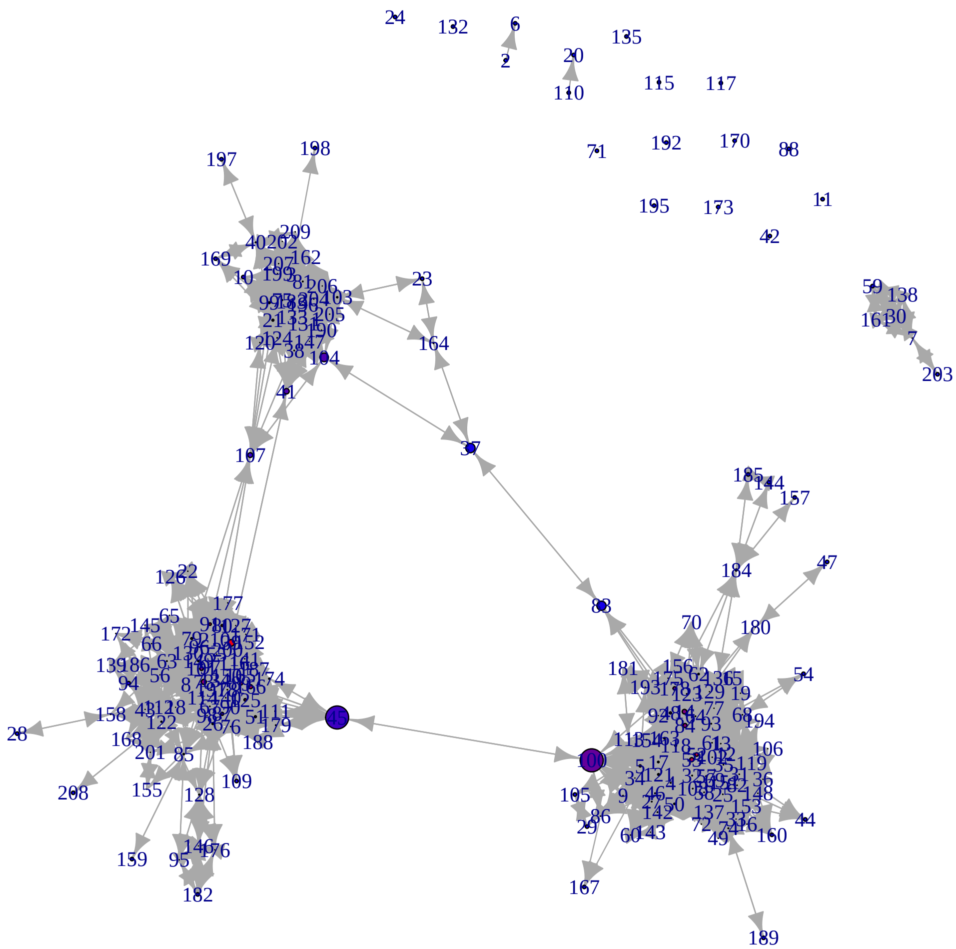
```r
plot(g, layout = g.cent.lyt,
     main = "Social Graph Closeness Centrality",
     vertex.color = g.cent.vclr, vertex.size = func_cent.vert.size(g.cls.cent))
```

Social Graph Closeness Centrality



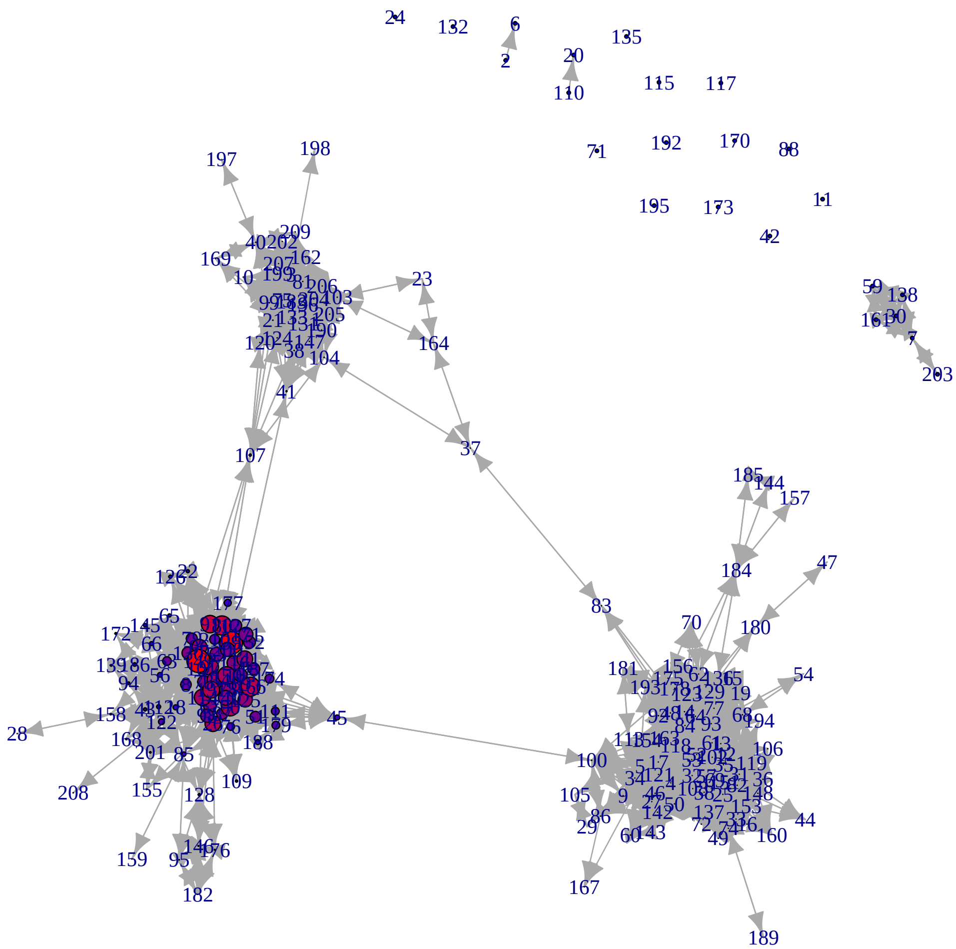

```r
plot(g, layout = g.cent.lyt,
     main = "Social Graph Betweeness Centrality",
     vertex.color = g.cent.vclr, vertex.size = func_cent.vert.size(g.btw.cent))
```

Social Graph Betweeness Centrality



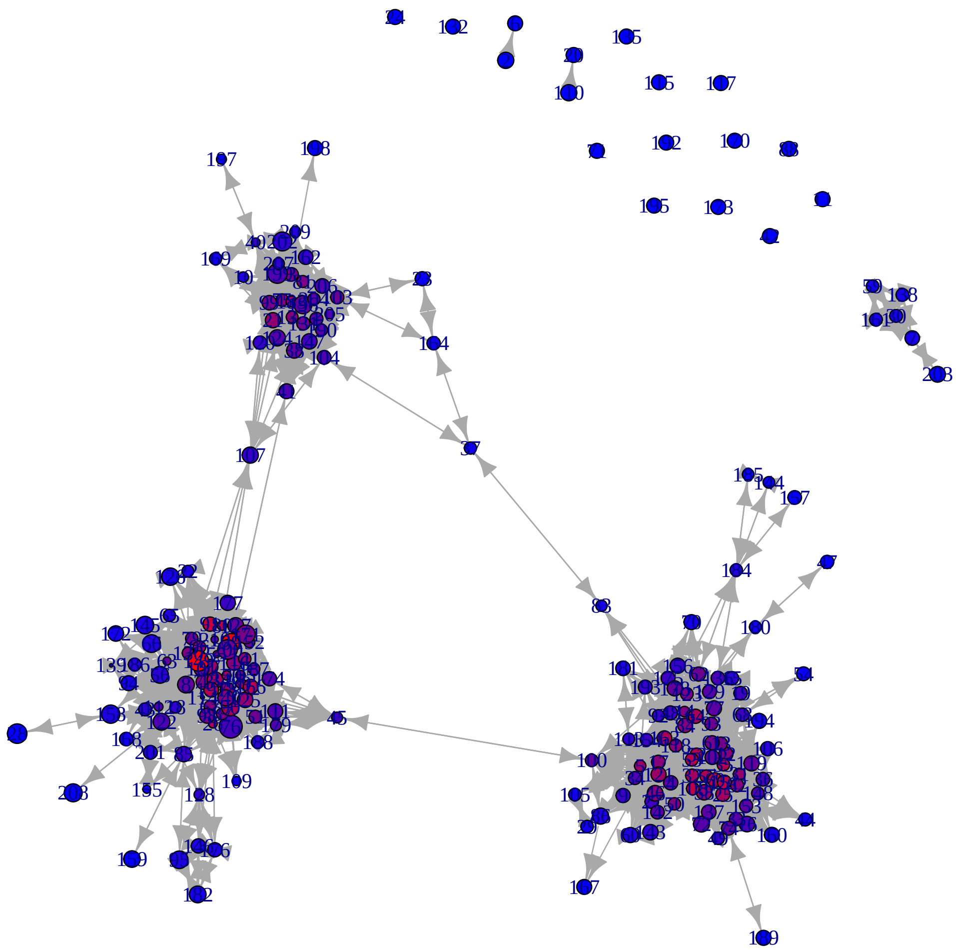
```r
plot(g, layout = g.cent.lyt,
     main = "Social Graph Eigenvector Centrality",
     vertex.color = g.cent.vclr, vertex.size = func_cent.vert.size(g.ev.cent$vector))
```

Social Graph Eigenvector Centrality



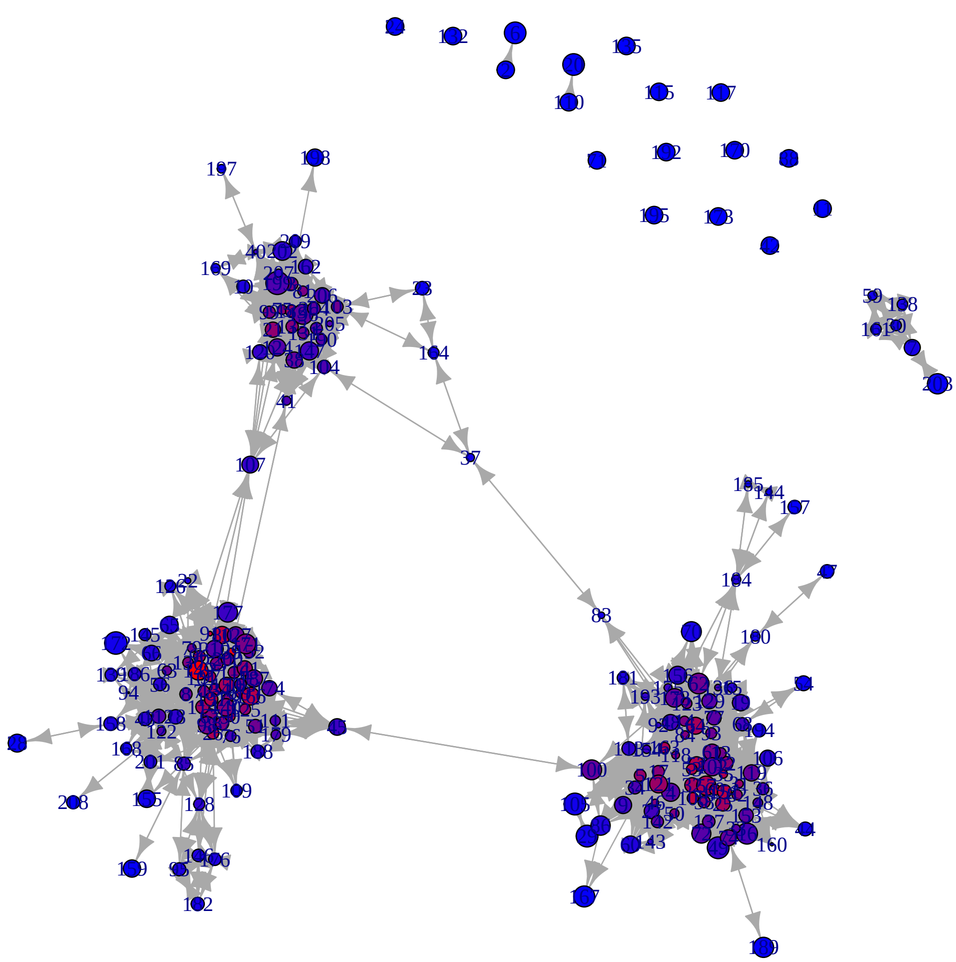

```r
plot(g, layout = g.cent.lyt,
     main = "Social Graph Bonachich Centrality",
     vertex.color = g.cent.vclr, vertex.size = func_cent.vert.size(g.bp.cent))
```

Social Graph Bonachich Centrality



```r
plot(g, layout = g.cent.lyt, main = "Social Graph Alpha Centrality",
     vertex.color = g.cent.vclr, vertex.size = func_cent.vert.size(g.alpha.cent))
```

Social Graph Alpha Centrality



Now, output top ten nodes in each ranking. Again, print only first names in your table to keep privacy:

```
func_topN = function(n, l) which( l > sort(l)[ length(l)-n ] )
func_top10 = function(l) func_topN(10, l)
func_before.space = function(s) sub("\\s.*", "", s)


options(xtable.comment = F)

func_print.custom.xtable = function(m)
    print(xtable(m), floating = F, type = "latex", include.rownames = F)

g.deg.cent.top10 = func_top10(g.deg.cent)
g.deg.cent.top10.mat = cbind(func_before.space( g.v.names[g.deg.cent.top10] ), g.deg.cent.top10)
```

```
colnames(g.deg.cent.top10.mat) = c("Node", "Degree Centrality")
func_print.custom.xtable(g.deg.cent.top10.mat)
```

Node	Degree Centrality
Tanyushka	39
Tatyana	53
Nastya	69
Lizaveta	80
Andrey	91
Veronika	108
Evgeny	149
Danya	151
Maria	166

```
g.cls.cent.top10 = func_top10(g.cls.cent)
g.cls.cent.top10.mat = cbind(func_before.space( g.v.names[g.cls.cent.top10] ), g.cls.cent.top10)
colnames(g.cls.cent.top10.mat) = c("Node", "Closeness Centrality")
func_print.custom.xtable(g.cls.cent.top10.mat)
```

Node	Closeness Centrality
Dima	45
Tatyana	51
Stanislav	73
Sergey	87
Alexey	89
Ivan	100
Alexander	125
Maria	166
Tatyana	174

```
g.btw.cent.top10 = func_top10(g.btw.cent)
g.btw.cent.top10.mat = cbind(func_before.space(g.v.names[g.btw.cent.top10]), g.btw.cent.top10)
colnames(g.btw.cent.top10.mat) = c("Nodes", "Betweeness Centrality")
func_print.custom.xtable(g.btw.cent.top10.mat)
```

Nodes	Betweeness Centrality
Anastasia	14
Anatoly	37
Tanyushka	39
Grigory	41
Dima	45
Stanislav	73
Anton	83
Ivan	100
Insaf	104
Daniil	107

```
g.ev.cent.top10 = func_top10(g.ev.cent$vector)
g.ev.cent.top10.mat = cbind(func_before.space(g.v.names[g.ev.cent.top10]), g.ev.cent.top10)
colnames(g.ev.cent.top10.mat) = c("Nodes", "Eigenvector Centrality")
func_print.custom.xtable(g.ev.cent.top10.mat)
```

Nodes	Eigenvector Centrality
Tanyushka	39
Lizaveta	80
Alexey	89
Oleg	90
Andrey	91
Amir	114
Evgeny	149
Elena	150
Maria	166
Dima	191

```
g.bp.cent.top10 = func_top10(g.bp.cent)
g.bp.cent.top10.mat = cbind(func_before.space(g.v.names[g.bp.cent.top10]), g.bp.cent.top10)
colnames(g.bp.cent.top10.mat) = c("Nodes", "Bonachich Centrality")
func_print.custom.xtable(g.bp.cent.top10.mat)
```

Nodes	Bonachich Centrality
Ekaterina	28
Tanyushka	39
Andrey	76
Evgeny	149
Ekaterina	158
Dmitry	171
Joonatan	196
Ali-BarÄ±Å	199
Dmitry	200
Tomas	202

```
g.alpha.cent.top10 = func_top10(g.alpha.cent)
g.alpha.cent.top10.m = cbind(func_before.space(g.v.names[g.alpha.cent.top10]), g.alpha.cent.top10)
colnames(g.alpha.cent.top10.m) = c("Nodes", "Alpha Centrality")
func_print.custom.xtable(g.alpha.cent.top10.m)
```

Nodes	Alpha Centrality
Anya	6
Katerina	16
Sergey	20
Anastasia	29
Anna	49
Jane	62
Georgy	105
Anastasia	167
Stanislav	172
Ali-BarÄ±Å	199

Daniil has predictably revealed the highest betweeness centrality in my social graph since he is one of my most communicative buddies, who indeed preserves strong ties within our group.

Task 2. Flickr network

Please download flickr.mat

Data contains sparse matrix A and list of user names. This is a denser part of the Flickr photo sharing site friendship graph from 2006. Edge direction corresponds to friendship requests (following). Some of the links are reciprocal, others not.

It's a Matlab file. How to deal with it in R? There is a package `R.matlab`. Please install it and call `library(R.matlab)`

Now use `readMat` function to read the file and extract adjacency matrix and a list of user names:

```
flickr = readMat("flickr.mat")
flickr.mat = as.matrix(flickr[1]$A)
flickr.names = flickr[2]$names
```

Look at user names. You might want to remove spaces from the names. Use a function `gsub` to remove them:

```
flickr.names = gsub(" ", "", flickr.names)
```

Now create a graph, output the number of vertices and edges:

```
f_g = graph.adjacency(flickr.mat)

f_g.n = vcount(f_g)
f_g.m = ecount(f_g)

f_g.v.names = flickr.names

paste0("Vertex count: ", f_g.n)
```

```
## [1] "Vertex count: 15724"
```

```
paste0("Edge count: ", f_g.m)
```

```
## [1] "Edge count: 510983"
```

Compute in- and out- degree centralities, PageRank, Hubs and Authorities for this network:

```
f_g.indeg.cent = degree(f_g, mode = "in")
f_g.outdeg.cent = degree(f_g, mode = "out")
f_g.pr = page.rank(f_g)
f_g.hub = hub.score(f_g)
f_g.auth = authority.score(f_g)
```

Print top ten names in each ranking:

```
f_g.indeg.cent.top10 = func_top10(f_g.indeg.cent)
f_g.indeg.cent.top10.m = cbind(f_g.v.names[f_g.indeg.cent.top10], f_g.indeg.cent.top10)
colnames(f_g.indeg.cent.top10.m) = c("Node", "In-degree Centrality")
func_print.custom.xtable(f_g.indeg.cent.top10.m)
```

Node	In-degree Centrality
awfulsara	1465
DrJoanne	2129
drp	3747
BombDog	4013
Ivan	5629
deborahlattimore	7534
SimonPais	8235
antimethod	10846
MaDGi®L	12813

```
f_g.outdeg.cent.top10 = func_top10(f_g.v.names)
f_g.outdeg.cent.top10.mat = cbind(f_g.v.names[f_g.outdeg.cent.top10], f_g.outdeg.cent.top10)
colnames(f_g.outdeg.cent.top10.mat) = c("Node", "Out-degree Centrality")
func_print.custom.xtable(f_g.outdeg.cent.top10.mat)
```

Node	Out-degree Centrality
zzztimbo	2475
zyphichore	3918
zusty	4820
Zzilazz	7165
Zumk	8080
zzle	9825
zweidel	10917
zzcoyote	11962
Zzzzt!Zzzzt!	13318
zwwfc	14320

```
f_g.pr.top10 = func_top10(f_g.pr$vector)
f_g.pr.top10.mat = cbind(f_g.v.names[f_g.pr.top10], f_g.pr.top10)
colnames(f_g.pr.top10.mat) = c("Node", "Pagerank")
func_print.custom.xtable(f_g.pr.top10.mat)
```

Node	Pagerank
awfulsara	1465
DrJoanne	2129
drp	3747
BombDog	4013
cymagen	4965
Ivan	5629
deborahlattimore	7534
SimonPais	8235
antimethod	10846
MaDGi®L	12813

```
f_g.hub.top10 = func_top10(f_g.hub$vector)
f_g.hub.top10.mat = cbind(f_g.v.names[f_g.hub.top10], f_g.hub.top10)
colnames(f_g.hub.top10.mat) = c("Node", "Hub")
func_print.custom.xtable(f_g.hub.top10.mat)
```

Node	Hub
mrpiink	3821
noahstone	4996
isherwood	6518
sgoralnick	6572
starlet	7380
automat	7954
brynfoto	10487
liquidpixel	11304
lorrainemd	12675
schizoo23	14004

```
f_g.auth.top10 = func_top10(f_g.auth$vector)
f_g.auth.top10.mat = cbind(f_g.v.names[f_g.auth.top10], f_g.auth.top10)
colnames(f_g.auth.top10.mat) = c("Node", "Authority")
func_print.custom.xtable(f_g.auth.top10.mat)
```

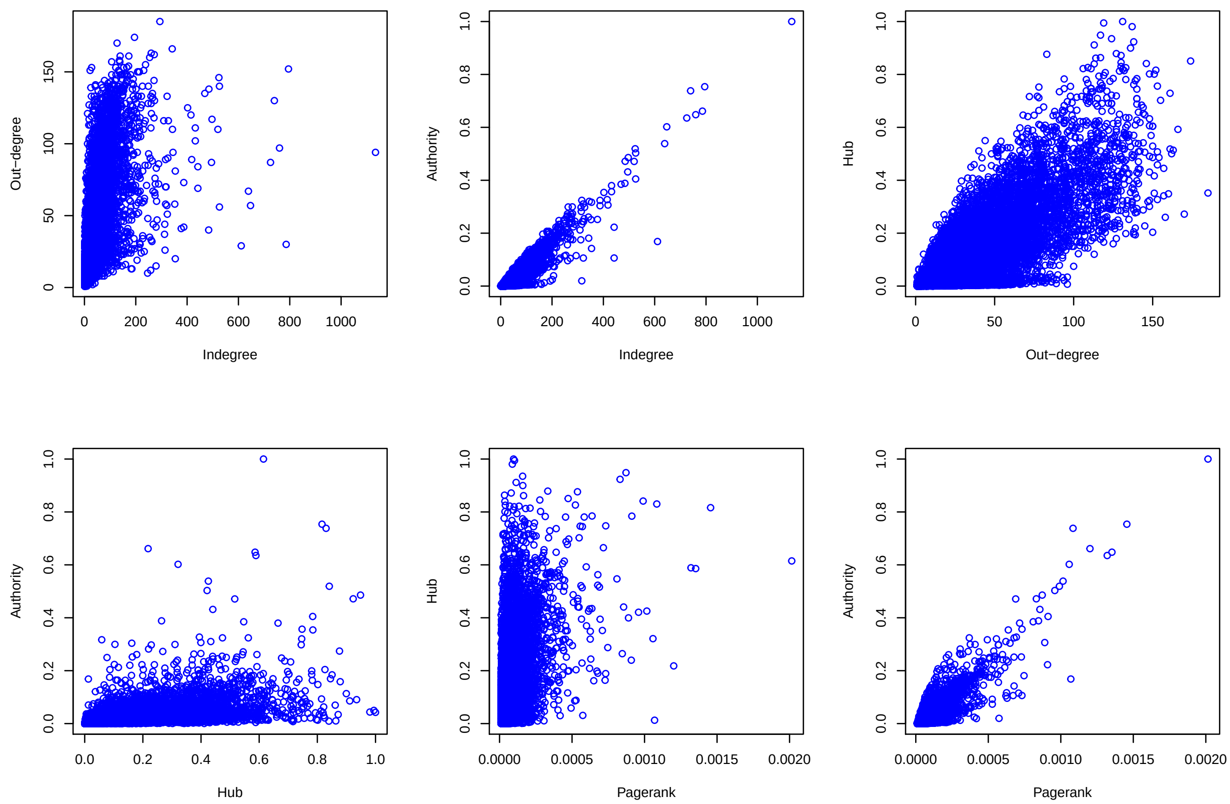
Node	Authority
awfulsara	1465
DrJoanne	2129
drp	3747
BombDog	4013
:Nikola	4301
cymagen	4965
Ivan	5629
deborahlattimore	7534
SimonPais	8235
antimethod	10846

Produce the following plots:

- In-degree centralities versus out-degree centralities
- In-degree centralities versus authorities
- Out-degree centralities versus hubs
- Hubs versus authorities
- PageRank versus hubs
- PageRank versus authorities

```
func_plot.vs = function(m1, lab1, m2, lab2, col = "blue")
    plot(m1, m2, xlab = lab1, ylab = lab2, col = col)

par(mfrow = c(2, 3))

func_plot.vs(f_g.indeg.cent, "Indegree", f_g.outdeg.cent, "Out-degree")

func_plot.vs(f_g.indeg.cent, "Indegree", f_g.auth$vector, "Authority")

func_plot.vs(f_g.outdeg.cent, "Out-degree", f_g.hub$vector, "Hub")

func_plot.vs(f_g.hub$vector, "Hub", f_g.auth$vector, "Authority")
```

```r
func_plot.vs(f_g.pr$vector, "Pagerank", f_g.hub$vector, "Hub")

func_plot.vs(f_g.pr$vector, "Pagerank", f_g.auth$vector, "Authority")

mtext("Centrality Comparison", outer = T)
```



For <in-degree, authority>, <out-degree, hub> and <pagerank, authority> pairs a positive correlation is revealed, as opposed to the rest of those being compared.