Network analysis using R

Gianluca Campanella

Contents

Setup

Let's start by loading the packages we'll be using later.

You can install any missing packages using `install.packages` or the RStudio GUI.

```
library("corpcor")
library("igraph")
library("poweRlaw")
library("tidyverse")
```

Brief introduction to igraph

igraph is an efficient network analysis toolkit implemented in C, with a convenient R interface.

We'll briefly review some of the main functions of `igraph`, before delving into an '-omics' application. For a longer introduction to `igraph`, check out Katherine Ognyanova's *Network Analysis and Visualization with R and igraph*.

Graphs can be specified using the function `graph`. For example, let's create an undirected graph with three vertices and three edges.

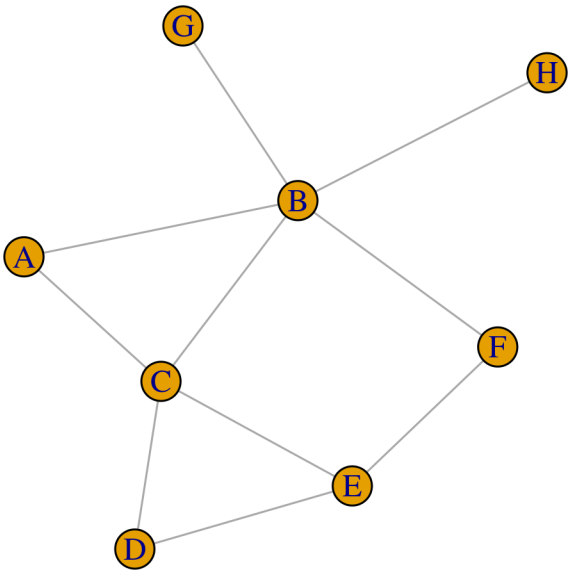
```
g1 <- graph(edges = c(1, 2, 2, 3, 3, 1), directed = FALSE)
```

Graphs can also be specified using `graph_from_literal`.

```
g2 <- graph_from_literal(A-B-C-D-E-F, A-C-E, B-F:G:H)
```

Small graphs can be plotted using `plot`.

```
plot(g2)
```



Vertices and edges can be accessed using the functions `V` and `E`, respectively.

```
V(g2)
```

```
## + 8/8 vertices, named, from 0b354f5:
## [1] A B C D E F G H
```

```
E(g2)
```

```
## + 10/10 edges from 0b354f5 (vertex names):
##  [1] A--B A--C B--C B--F B--G B--H C--D C--E D--E E--F
```

Alternatively, the adjacency matrix representation is also available.

```
g2[]
```

```
## 8 x 8 sparse Matrix of class "dgCMatrix"
##   A B C D E F G H
## A . 1 1 . . . . .
## B 1 . 1 . . 1 1 1
## C 1 1 . 1 1 . . .
## D . . 1 . 1 . . .
## E . . 1 1 . 1 . .
## F . 1 . . 1 . . .
## G . 1 . . . . . .
## H . 1 . . . . . .
```

Correlation networks

We'll use data from the study *Autoantibody profiling of sera from individuals with Systemic Lupus Erythematosus (SLE) by protein microarray* (E-MTAB-5900 on ArrayExpress).

The original data can be found in the `input_data` folder. The dataset we'll be using was generated using the script `build_dataset.R`, and is stored in the `datasets` folder.

Let's start by loading the data.

```
sle <- read_rds("datasets/sle_proteomics.rds")
```

Next, we perform some basic exploratory analyses.

```
sle %>%
    select(-starts_with("P")) %>%
    head()
```

```
## # A tibble: 6 x 4
##      id country ethnicity case
##   <int> <chr>   <chr>     <lgl>
## 1     1 UK      Caucasian TRUE
## 2     2 UK      Caucasian TRUE
## 3     3 UK      Caucasian TRUE
## 4     4 UK      Caucasian TRUE
## 5     5 UK      Caucasian TRUE
## 6     6 UK      Caucasian TRUE
```

```
sle %>%
    group_by(country, ethnicity) %>%
    summarize(
        n = n(),
        n_cases = sum(case),
        cases_pct = mean(case)
    )
```

```
## # A tibble: 5 x 5
## # Groups:   country [?]
##   country ethnicity          n n_cases cases_pct
##   <chr>   <chr>          <int>   <int>     <dbl>
## 1 UK      Afro-Caribbean   179      87     0.486
## 2 UK      Caucasian        192      94     0.490
## 3 USA     Afro-American     40      21     0.525
## 4 USA     Caucasian         72      34     0.472
## 5 USA     Hispanic          63      30     0.476
```

We now focus on the protein-protein correlation network (irrespective of disease status), which we estimate using the shrinkage estimator implemented in the corpcor package.

```
cor_all <- sle %>%
           select(starts_with("P")) %>%
           as.matrix() %>%
           cor.shrink()
```

```
## Estimating optimal shrinkage intensity lambda (correlation matrix): 0.0415
```

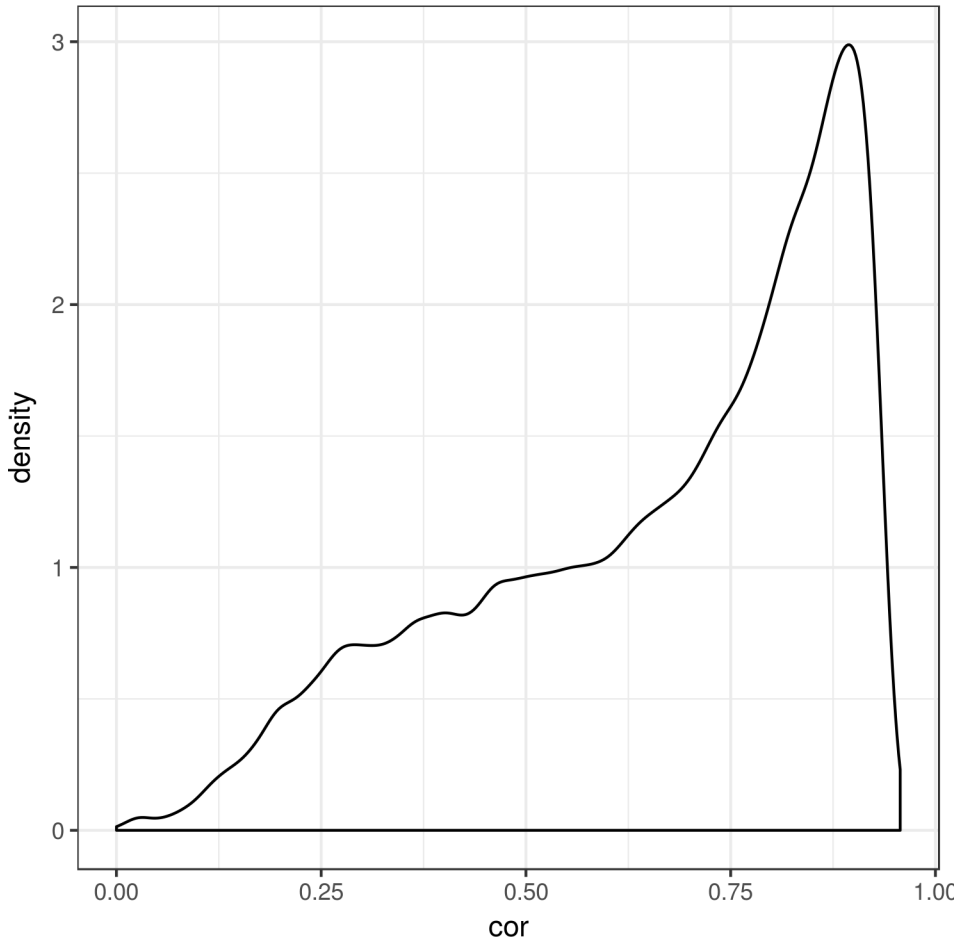
We have a few options to turn this correlation matrix into a graph:

- Use the correlation coefficients directly as weights
- Use the absolute value of the correlation coefficients as weights
- Dichotomise with respect to an arbitrary (absolute) correlation threshold

We'll illustrate the last method using an arbitrary threshold based on the quantiles of the absolute correlation coefficient distribution.

```
abs_cors <- abs(cor_all[upper.tri(cor_all)])
```

```
ggplot(tibble(cor = abs_cors), mapping = aes(x = cor)) +
    geom_density() +
    theme_bw()
```



```
adj_matrix <- ifelse(abs(cor_all) > quantile(abs_cors, 0.95), 1, 0)

g3 <- graph_from_adjacency_matrix(adj_matrix, mode = "undirected")
```

The graph we've obtained is not necessarily connected.

Let's define a function `largest_component` to extract the largest connected component of the graph, and apply it to our object.

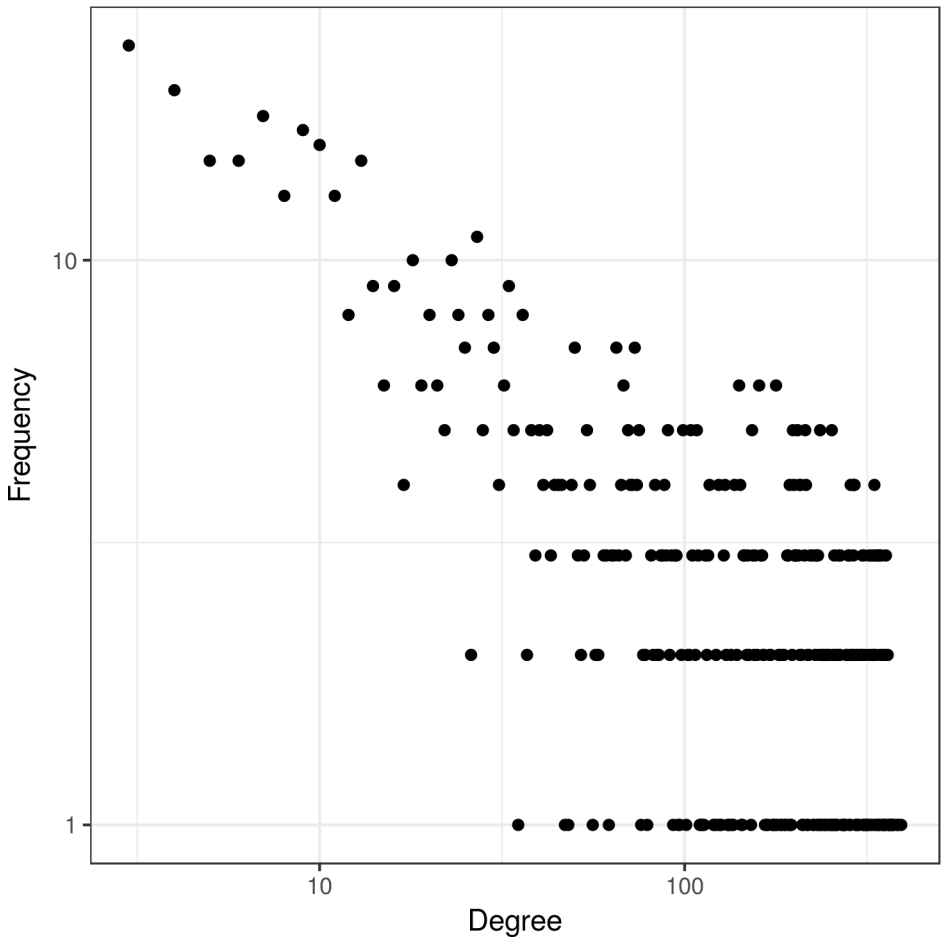
```
largest_component <- function(graph, ...) {
    cs <- components(graph, ...)
    induced_subgraph(graph, which(cs$membership == which.max(cs$csize)))
}

lc3 <- largest_component(g3)
```

We'll also define a function to plot the degree distribution (in log-log scale), and a function to fit a discrete power law and a discrete log-normal distribution to the degrees (using the package `poweRlaw`).

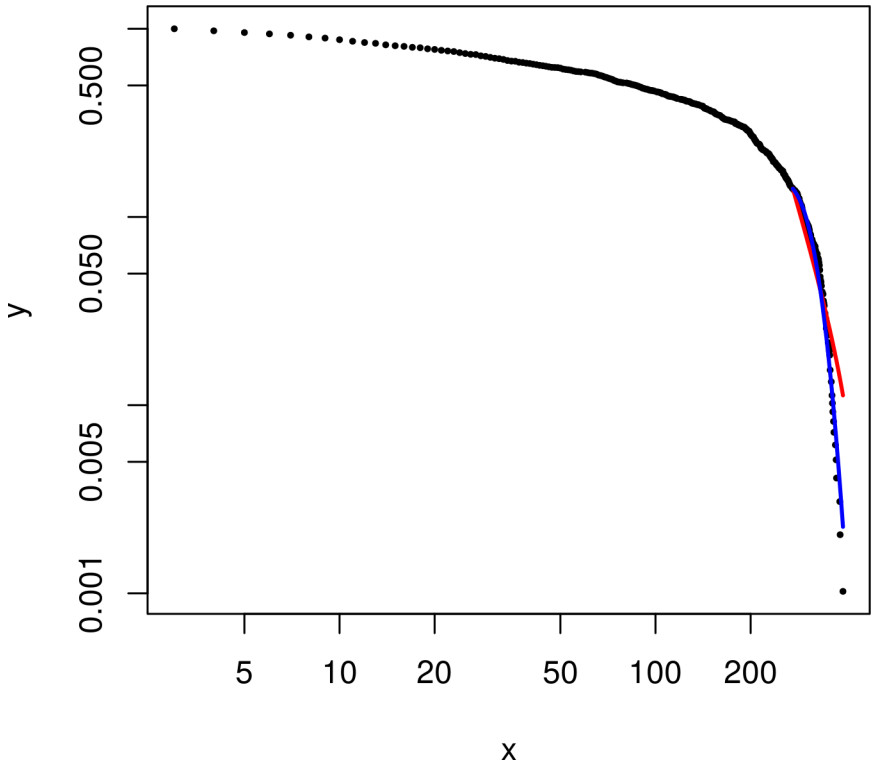
```
plot_degree_distribution <- function(graph) {
    ggplot(tibble(degree = degree(graph)), mapping = aes(x = degree)) +
        geom_point(stat = "count") +
        scale_x_continuous("Degree", trans = "log10") +
        scale_y_continuous("Frequency", trans = "log10") +
        theme_bw()
}

fit_degree_distribution <- function(graph) {
    degrees <- degree(graph)

    pl <- displ$new(degrees)
    pl$setXmin(estimate_xmin(pl))

    lnorm <- dislnorm$new(degrees)
    lnorm$setXmin(pl$getXmin())
    lnorm$setPars(estimate_pars(lnorm))

    plot(pl, pch = 20, cex = 0.5)
    lines(pl, col = "red", lwd = 2)
    lines(lnorm, col = "blue", lwd = 2)

    compare_distributions(pl, lnorm)$p_two_sided
}
```

Let's apply these functions to `lc3`.

```
plot_degree_distribution(lc3)
```




```
fit_degree_distribution(lc3)
```



```
## [1] 6.906611e-05
```

It appears that the discrete log-normal distribution provides a better fit in this case.

Differential correlation networks

In case-control studies we're often interested in identifying correlates of disease status. We can extend this idea to networks by investigating differences between the protein-protein correlation networks observed under each condition.

```
cor_cases <- sle %>%
             filter(case) %>%
             select(starts_with("P")) %>%
             as.matrix() %>%
             pcor.shrink()
```

```
## Estimating optimal shrinkage intensity lambda (correlation matrix): 0.0533
```

```
cor_controls <- sle %>%
                filter(!case) %>%
                select(starts_with("P")) %>%
                as.matrix() %>%
                pcor.shrink()
```

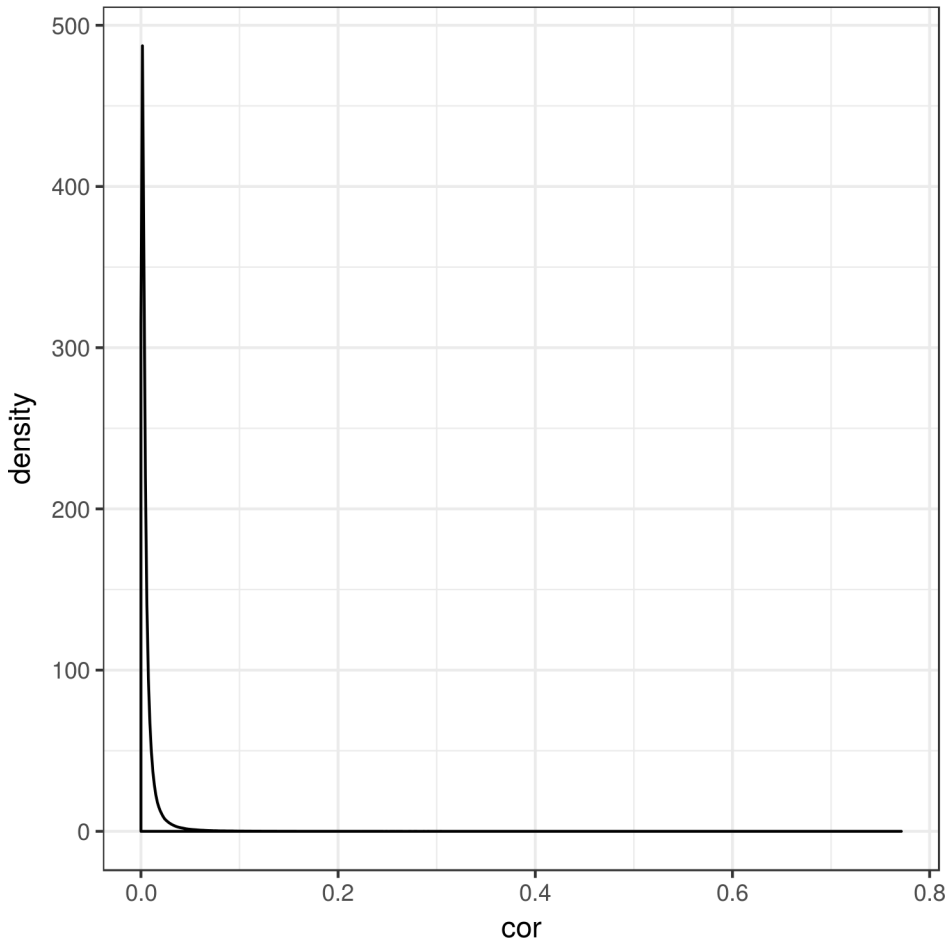
```
## Estimating optimal shrinkage intensity lambda (correlation matrix): 0.131
```

```
cor_diff <- cor_cases - cor_controls
```

We'll again dichotomise with respect to an arbitrary threshold based on the quantiles of the absolute correlation difference distribution.

```
abs_diffs <- abs(cor_diff[upper.tri(cor_diff)])
```

```
ggplot(tibble(cor = abs_diffs), mapping = aes(x = cor)) +
    geom_density() +
    theme_bw()
```



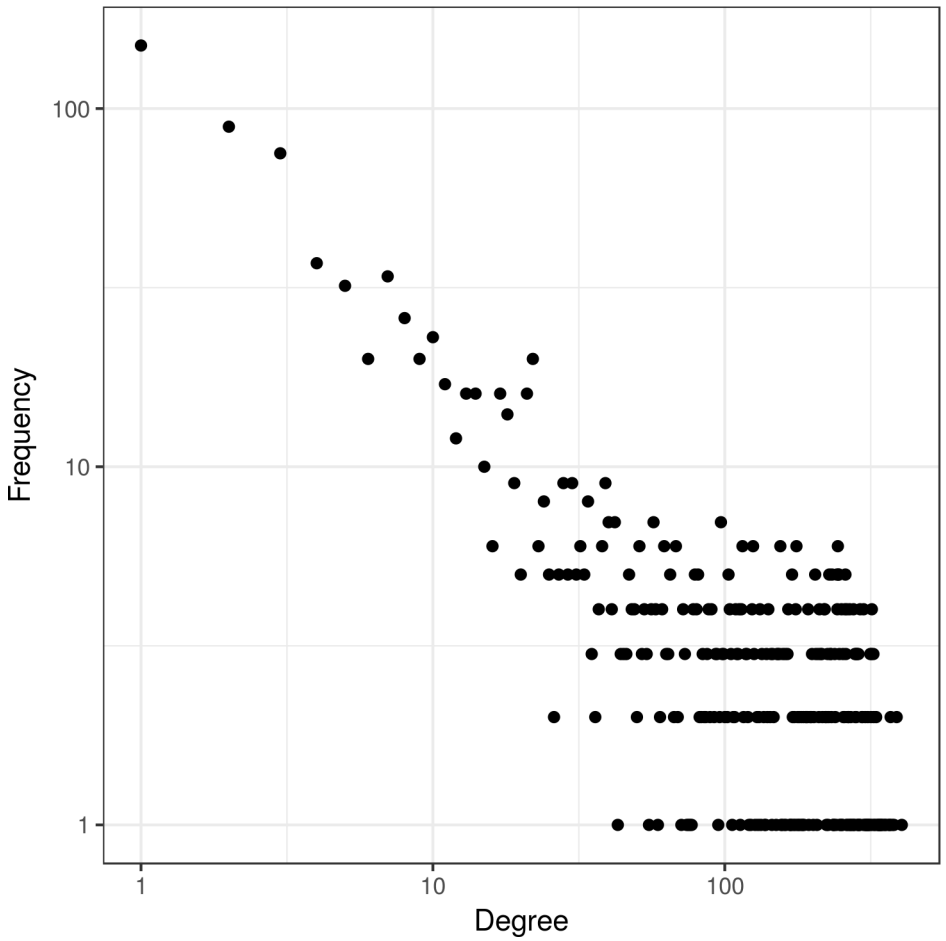

```
adj_matrix <- ifelse(abs(cor_diff) > quantile(abs_diffs, 0.95), 1, 0)
```

```
g4 <- graph_from_adjacency_matrix(adj_matrix, mode = "undirected")
```

We'll also retrieve the largest component, and plot its degree distribution.

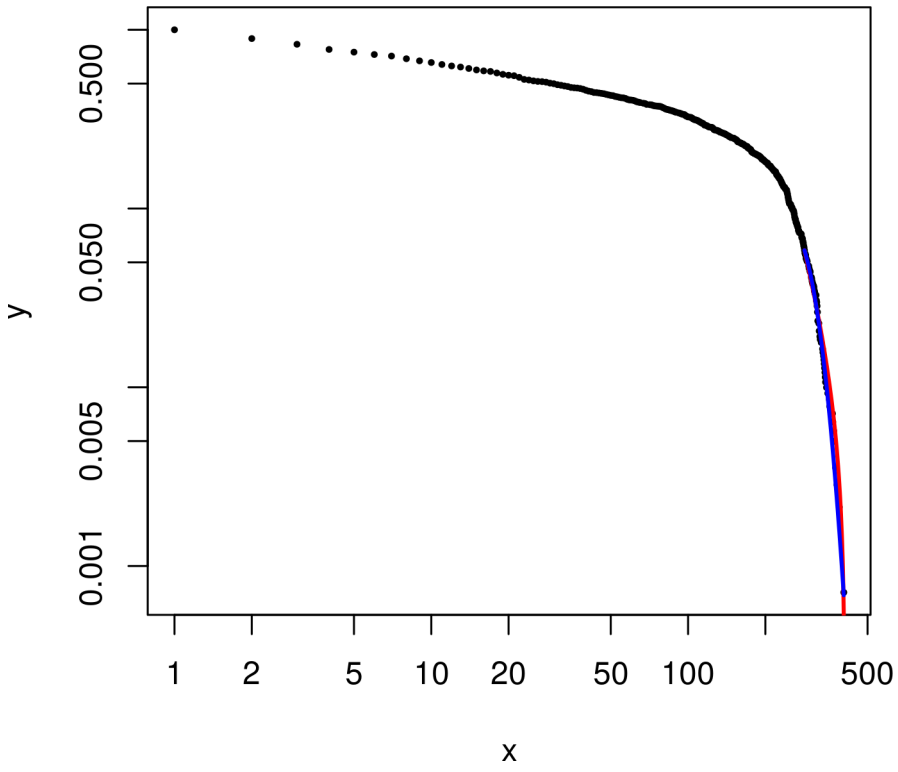
```
lc4 <- largest_component(g4)
```

```
plot_degree_distribution(lc4)
```



Finally, we'll fit a discrete power law and a discrete log-normal distribution to the degrees.

```
fit_degree_distribution(lc4)
```




```
## [1] 0.9359877
```

It appears that both distributions provide similar fits in this case.

We'll now compute different centrality measures.

```
centralities <- tibble(
    id = names(V(lc4)),
    degree = degree(lc4, normalized = TRUE),
    closeness = closeness(lc4, normalized = TRUE),
    betweenness = betweenness(lc4, normalized = TRUE),
    eigen = eigen_centrality(lc4)$vector
)
```

First of all, note how different measures of centrality are correlated.

```
cor(centralities %>% select(-id), method = "spearman")
```

```
##                degree closeness betweenness     eigen
## degree      1.0000000 0.9941397   0.9828218 0.9978278
## closeness   0.9941397 1.0000000   0.9775846 0.9968782
## betweenness 0.9828218 0.9775846   1.0000000 0.9766828
## eigen       0.9978278 0.9968782   0.9766828 1.0000000
```

Let's investigate some of the more 'central' vertices.

```
centralities %>%
    arrange(desc(degree))
```

```
## # A tibble: 1,404 x 5
##    id        degree closeness betweenness eigen
##    <chr>      <dbl>     <dbl>       <dbl> <dbl>
##  1 P000213_1  0.288     0.568     0.0146  0.943
##  2 P000201_1  0.277     0.559     0.0152  0.913
##  3 P003218_1  0.277     0.565     0.00629 1.00
##  4 P001280_1  0.271     0.558     0.0111  0.971
##  5 P003029_1  0.267     0.564     0.0203  0.907
##  6 P001964_1  0.264     0.553     0.0106  0.917
##  7 P000252_1  0.264     0.558     0.0195  0.852
##  8 P001384_1  0.263     0.561     0.0153  0.927
##  9 P002104_1  0.261     0.560     0.0117  0.941
## 10 P001894_1  0.260     0.560     0.0143  0.886
## # ... with 1,394 more rows
```

Alternatively, we can compare ranks.

```
centrality_ranks <- centralities %>%
                    mutate_if(is.numeric, funs(min_rank(desc(.))))
```

```
centrality_ranks %>%
    arrange(degree)
```

```
## # A tibble: 1,404 x 5
##    id        degree closeness betweenness eigen
##    <chr>      <int>     <int>       <int> <int>
##  1 P000213_1      1         1          15     3
##  2 P000201_1      2         8          13     8
##  3 P003218_1      2         2          52     1
##  4 P001280_1      4         9          23     2
##  5 P003029_1      5         3           5     9
##  6 P001964_1      6        18          24     7
##  7 P000252_1      6         9           7    29
##  8 P001384_1      8         4          12     5
##  9 P002104_1      9         6          21     4
## 10 P001894_1     10         5          16    16
## # ... with 1,394 more rows
```

Let's focus on proteins that appear amongst the top 3 for any centrality measure.

```
centrality_ranks %>%
    filter_if(is.numeric, any_vars(. <= 3))
```

```
## # A tibble: 8 x 5
##   id          degree closeness betweenness eigen
##   <chr>        <int>     <int>       <int> <int>
## 1 P000213_1        1         1          15     3
## 2 P001280_1        4         9          23     2
## 3 P000165_1       12        11           1    44
## 4 P000201_1        2         8          13     8
## 5 P003029_1        5         3           5     9
## 6 P002239.2_1     34        39           3    86
## 7 P000131_1      149       190           2   253
## 8 P003218_1        2         2          52     1
```

Consulting the array design file on ArrayExpress, we arrive at the following table:

id	UniProt	Gene name
P000131_1	Q13526	*PIN1*
P000165_1	Q92934	*BAD*
P000201_1	O75081	*CBFA2T3*
P000213_1	Q9UHB7	*AFF4*
P001280_1	P54252	*ATXN3*
P002239.2_1	Q9H999	*PANK3*
P003029_1	Q5JRK9	*PAGE2B*
P003218_1	Q06330	*RBPJ*

Whilst interpretation of these results is beyond the scope of this workshop, we note that:

- *PIN1* has been found to be abnormally activated in SLE by Wei et al.
- *PANK3* codes for isoform 3 of pantothenate kinase, the first enzyme in the CoA biosynthetic pathway. CoA deficiency has been hypothesised to be involved in the pathogenesis of SLE by Leung.
- *RBPJ* is part of the Notch signalling pathway, which has been explored as a potential target for SLE treatment by Teachey et al.