Psychological Networks Summer School
Day 2, part 2: Drawing networks

Sacha Epskamp

University of Amsterdam
Department of Psychological Methods

05-07-2016
Get the latest version of R from www.r-project.org and the latest version of `qgraph` from CRAN:

```r
install.packages("qgraph", dep=TRUE)
```
Make sure you can load qgraph:

```r
library("qgraph")
```

And that you have version 1.3.4 or higher:

```r
packageDescription('qgraph')$Version
```

```r
## [1] "1.3.4"
```
If this fails, make sure you have the latest (3.4.0) version of R and that all depended/imported/suggested packages are installed (see CRAN).
The `qgraph()` function

- The main function in `qgraph` is `qgraph()`
  - Most other functions are either wrapping functions using `qgraph()` or functions used in `qgraph()`

- The `qgraph()` function requires only one argument (`input`)

- A lot of other arguments can be specified, but these are all optional

Usage:

```
qgraph( input, ... )
```
Weights matrices

• The input argument is the input. This can be a weights matrix.

• A weights matrix is a square $n$ by $n$ matrix in which each element indicates the relationship between two variables.

• Any relationship can be used as long as:
  • A 0 indicates no relationship
  • Absolute negative values are similar in strength to positive values

• We will first look at unweighted graphs, in which case the weights matrix is the same as an adjacency matrix.
  • A 1 indicates a connection
  • A 0 indicates a connection
  • Rows indicate the node of origin
  • Columns indicate the node of destination
  • By default the diagonal is omitted
  • By default, a symmetrical weights matrix is interpreted as an unweighted graph.
Weights matrices

```r
input <- matrix(c(0,1,1,
                  0,0,1,
                  0,0,0),3,3,byrow=TRUE)
print(input)
```

```
## [,1] [,2] [,3]
## [1,]  0  1  1
## [2,]  0  0  1
## [3,]  0  0  0
```

```r
cgraph(input)
```
Weights matrices

Exercise: Create this graph

The layout should be right automatically, only use one argument in `qgraph()`
Weights matrices

To make this graph, we need this matrix:

```r
# To make this graph, we need this matrix:

## 
## [1,]  0  1  0  0  0  0  0  0
## [2,]  0  0  1  0  0  0  0  0
## [3,]  0  0  0  1  0  0  0  0
## [4,]  0  0  0  0  1  0  0  0
## [5,]  0  0  0  0  0  1  0  0
## [6,]  0  0  0  0  0  0  1  0
## [7,]  0  0  0  0  0  0  0  1
## [8,]  1  0  0  0  0  0  0  0
```
Weights matrices

These matrices become quite large, so manually defining the matrix is not effective. So some tricks are needed to make the matrix:

```r
input <- matrix(0, 8, 8)
input[1, 2] <- 1
input[2, 3] <- 1
input[3, 4] <- 1
input[4, 5] <- 1
input[5, 6] <- 1
input[6, 7] <- 1
input[7, 8] <- 1
input[8, 1] <- 1
```
Weights matrices

```r
print(input)

##
## [1,] 0 1 0 0 0 0 0 0
## [2,] 0 0 1 0 0 0 0 0
## [3,] 0 0 0 1 0 0 0 0
## [4,] 0 0 0 0 1 0 0 0
## [5,] 0 0 0 0 0 1 0 0
## [6,] 0 0 0 0 0 0 1 0
## [7,] 0 0 0 0 0 0 0 1
## [8,] 1 0 0 0 0 0 0 0
```
You can also change matrices manually (doesn’t work in RStudio):

```r
input <- matrix(0, 8, 8)
fix(input)
```

Or read the matrix from a text file!
Weights matrices

First make the matrix in a spreadsheet program (here LibreOffice)

```
<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
```

\[ f(x) \sum = 0 \]
Weights matrices

Next save as or export
Weights matrices

Save as CSV (comma delimited text file) or tab delimited:
Weights matrices

Read in R (for tab delimited use `read.table()`):

```r
input <- read.csv("adj.csv",header=FALSE)
print(input)
```

```plaintext
## V1  V2  V3  V4  V5  V6  V7  V8
## 1  0  1  0  0  0  0  0  0
## 2  0  0  1  0  0  0  0  0
## 3  0  0  0  1  0  0  0  0
## 4  0  0  0  0  1  0  0  0
## 5  0  0  0  0  0  1  0  0
## 6  0  0  0  0  0  0  1  0
## 7  0  0  0  0  0  0  0  1
## 8  1  0  0  0  0  0  0  0
```
These methods are not reproducible. Scripts should not depend on manual input. An easy way to change this is to first define a matrix. then run `dput()` on the object and use that result in your script:

```
dput(input)
```

```r
## structure(c(0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0), .Dim = c(8L, 8L))
```
Weights matrices

```
input2 <- structure(c(0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0,
1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 1, 0), .Dim = c(8L, 8L))

print(input2)
```

```r
#> [1,] 0  1  0  0  0  0  0  0
#> [2,] 0  0  1  0  0  0  0  0
#> [3,] 0  0  0  1  0  0  0  0
#> [4,] 0  0  0  0  1  0  0  0
#> [5,] 0  0  0  0  0  1  0  0
#> [6,] 0  0  0  0  0  0  1  0
#> [7,] 0  0  0  0  0  0  0  1
#> [8,] 1  0  0  0  0  0  0  0
```
Weights matrices

Exercise: Create this graph

1
2
3
4
5
6
7
8
## Weights matrices

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>[1,]</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>[2,]</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>[3,]</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>[4,]</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>[5,]</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>[6,]</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>[7,]</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>[8,]</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
The **directed** argument

- The **directed** argument can be used to force a directed (TRUE) or undirected (FALSE) graph
- This can also be specified per edge in a matrix
- By default, symmetric weight matrices are undirected and asymmetric weights matrices are directed
The directed argument

\[ \text{qgraph}(\text{input}) \]

\[
\text{input} \leftarrow \text{matrix}(1, 3, 3)
\]

\[
\text{print}(\text{input})
\]

\[
\begin{array}{ccc}
1 & 1 & 1 \\
1 & 1 & 1 \\
1 & 1 & 1
\end{array}
\]
The directed argument

```r
qgraph(input, directed=TRUE)
```

```
print(input)

## [,1] [,2] [,3]
## [1,] 1 1 1
## [2,] 1 1 1
## [3,] 1 1 1
```
The directed argument

input[1,2] <- 0
print(input)

```r
# input matrix
## [,1] [,2] [,3]
## [1,] 1 0 1
## [2,] 1 1 1
## [3,] 1 1 1
```

`qgraph(input)`
Weighted graphs

• Specify edge weights to make a graph weighted
  • In a weights matrix: simply specify other values than only zeros and ones
• An edge weight of 0 indicates no connection
• Positive and negative edge weights must be comparable in strength
• The “length” of an edge is defined as the inverse of the weight.
  • Stronger connected nodes are closer together
  • An edge weight of 0 indicates infinite length
Weighted graphs

```
input <- matrix(c(0, 1, 2,
                  0, 0, 3,
                  0, 0, 0), 3, 3, byrow = TRUE)

print(input)
```

```
## [,1] [,2] [,3]
## [1,] 0  1  2
## [2,] 0  0  3
## [3,] 0  0  0
```
Weighted graphs

```r
input <- matrix(c(0,1,-2,
                  1,0,3,
                  -2,3,0),3,3,byrow=TRUE)

print(input)

##   [,1] [,2] [,3]
## [1,]  0   1  -2
## [2,]  1   0   3
## [3,] -2   3   0
```

```r
qgraph(input)
```
Interpreting qgraph

- Under the default coloring scheme, positive edge weights (here correlations) are shown as green edges and negative edge weights as red.
- An edge weight of 0 is omitted. The wider and more colorful an edge the stronger the edge weight.
To interpret qgraph networks, three values need to be known:

- **Minimum** Edges with absolute weights under this value are omitted
- **Cut** If specified, splits scaling of width and color
- **Maximum** If set, edge width and color scale such that an edge with this value would be the widest and most colorful
No minimum, maximum or cut
No minimum, maximum or cut
Maximum 1
Maximum 1
Maximum must be set to make graphs comparable!
Minimum 0.1
Cut 0.4
Minimum 0.1
Cut 0.4
Maximum 1
By default, above 20 nodes the cut value is automatically chosen to be equal to the maximum of the 75th quantile of absolute edge strengths or the edge strength corresponding to 2n-th edge strength (n being the number of nodes).
Layout modes

- The placement of the nodes is specified with the `layout` argument in `qgraph()`.
- This can be a $n$ by 2 matrix indicating the $x$ and $y$ position of each node.
- `layout` can also be given a character indicating one of the two default layouts:
  - If `layout="circular"` the nodes are placed in circles per group (if the `groups` list is specified).
  - If `layout="spring"` the Fruchterman Reingold algorithm is used for the placement.
input <- matrix(1,3,3)
L <- matrix(c(0,1,
             1,1,
             0.5,0),
            ncol=2,byrow=TRUE)
print(L)

##       [,1] [,2]
## [1,] 0.0  1.0
## [2,] 1.0  1.0
## [3,] 0.5  0.0

qgraph(input, layout=L)
L <- matrix(c(0,1,
              1,1,
              0,0), ncol=2, byrow=TRUE)

print(L)

## [,1] [,2]
## [1,] 0 1
## [2,] 1 1
## [3,] 0 0

qgraph(input, layout=L)
Layout matrix

• With the layout matrix the actual layout can be specified
• The scale is not relevant
• `qgraph()` returns a list containing everything needed to make the graph
• This can be used to force another graph based on the layout of the first

\[
Q \leftarrow \texttt{qgraph}(\text{input})
\]
\[
\texttt{qgraph}(\text{input2, layout=Q$layout})
\]
Fruchterman-Reingold layout

- `layout="spring"` uses a force-embedded algorithm (the Fruchterman-Reingold algorithm)
- This is an iterative algorithm.
- The initial layout is a circle
- Then in each iteration:
  - Each node is repulsed by all other nodes
  - Connected nodes are also attracted to each other
  - The maximum displacement weakens each iteration
Load the big 5 dataset:

```r
data(big5)
str(big5)
```

```r
# num [1:500, 1:240] 2 3 4 4 5 2 2 1 4 2 ...
# - attr(*, "dimnames")=List of 2
# ..$: NULL
# ..$: chr [1:240] "N1" "E2" "O3" "A4" ...
```
Big 5

```
qgraph(cor(big5), minimum=0.25)
```
The *groups* argument

- The *groups* indicates which nodes belong together
- Nodes belonging together are...
  - placed in smaller circles (with circular layout)
  - colored in the same color (either rainbow or defined with color)
- Names in the *groups* can be used as legend
- *groups* can even be used to perform a oneline CFA with `qgraph.cfa()`

Either use a factor (a vector with characters) or a list in which each element is a vector containing the number of nodes that belong together
# The groups argument

# List:
groups <- list(A = c(1, 2, 3, 4, 5),
                B = c(6, 7, 8, 9, 10))

# Factor:
              "B", "B", "B", "B", "B")

# Result:
qgraph(matrix(1, 10, 10), groups=groups)
Big 5

data(big5groups)
big5graph <- qgraph(cor(big5), minimum=0.25, groups=big5groups)
Big 5

qgraph(big5graph, layout="spring")
qgraph graphs can **not** be manually rescaled, and hence the RStudio Export function can **not** be used to save qgraph graphs.

For the best result, save graphs in a PDF device!

Note that if a legend is used, the plot is made square by making the width 1.4 times the height of a plot.
Export to PDF

```r
# Open a pdf device:
pdf("nameoffile.pdf")

# Plot stuff:
qgraph(1)

# Close pdf device:
dev.off()

## pdf
## 2
```

(If you get faulty output, make sure to run `dev.off()` enough times until R returns `Null Device`.)
Export to PNG

# Open a pdf device:
png("nameoffile.png")

# Plot stuff:
qgraph(1)

# Close pdf device:
dev.off()

## pdf
## 2

(If you get faulty output, make sure to run dev.off() enough times untill R returns Null Device)
Important qgraph arguments

**minimum**  Omits edge weights with absolute values under this argument

**maximum**  Sets the strongest edge to scale to

**cut**  Splits scaling of color and width

**vsize**  Sets the size of nodes

**esize**  Sets the size of edges

**asize**  Sets the size of arrows

**filetype**  Type of file to save the plot to

**filename**  Name of the file to save the plot to

**groups**  Colors nodes and adds a legend

**nodeNames**  Adds a legend with specific names per node
Contribute to qgraph

The developmental version of qgraph can be found on GitHub (https://github.com/SachaEpskamp/qgraph) and can be installed using devtools

library("devtools")
install_github("qgraph","sachaepskamp")

If you have any ideas on concepts to implement in qgraph or encounter any bugs please post them on GitHub!
Thank you for your attention!