Making graphs in R

Using the **qgraph** package

Sacha Epskamp

University of Amsterdam
Department of Psychological Methods

15-10-2013

All codes in these slides were run using R version 3.0.1 (2013-05-16) and **qgraph** version 1.2.3 and were made on

```
Linux 3.8.0-29-generic x86_64 #42-Ubuntu SMP Tue Au
```

Get the latest version of R from www.r-project.org and the latest version of **qgraph** from CRAN:

```
install.packages("qgraph", dep = TRUE)
```

Make sure you can load qgraph:

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

And that you have version 1.2.3 or higher:

```
packageDescription("qgraph")$Version

## [1] "1.2.3"
```

If this fails, make sure you have the latest (2.15) version of R and that all depended/imported/suggested packages are installed (see CRAN).

Defaults

Note that the following defaults are set for this presentation:

```
options(
  qgraph = list(
    border.width = 2,
    asize = 8,
    unCol = "black",
    vsize = 10,
    esize = 3)
)
```

So the codes can create different looking graphs on your screen!

Help on R

Do it yourself. . .

- For basic understanding of R: Read through a R manual!
  - How do I make a matrix?
  - How do I index an object?
  - What is a list?
  - Try a short one first! (R for beginners)
- Help on how to use a function: Use the `?` function (e.g. `?matrix`)
  - How do I define a matrix by row?
  - How do I set `mean()` to omit NA's?
- Find a certain function: Use the `??` function
  - What is a function to reduce a string to a certain amount of characters?
    - `??trim`
- Or use google!

Help on R

. . . or ask for help!

- Stackoverflow websites (see next slide)
- For problems concerning specific packages: Mail the maintainer
- For short questions, you can use Twitter or Google+ with hashtag #rstats

Stackexchange

Stackexchange is a series of free question and answer websites on many different topics. Two are very useful for whenever you get stuck in **R**:

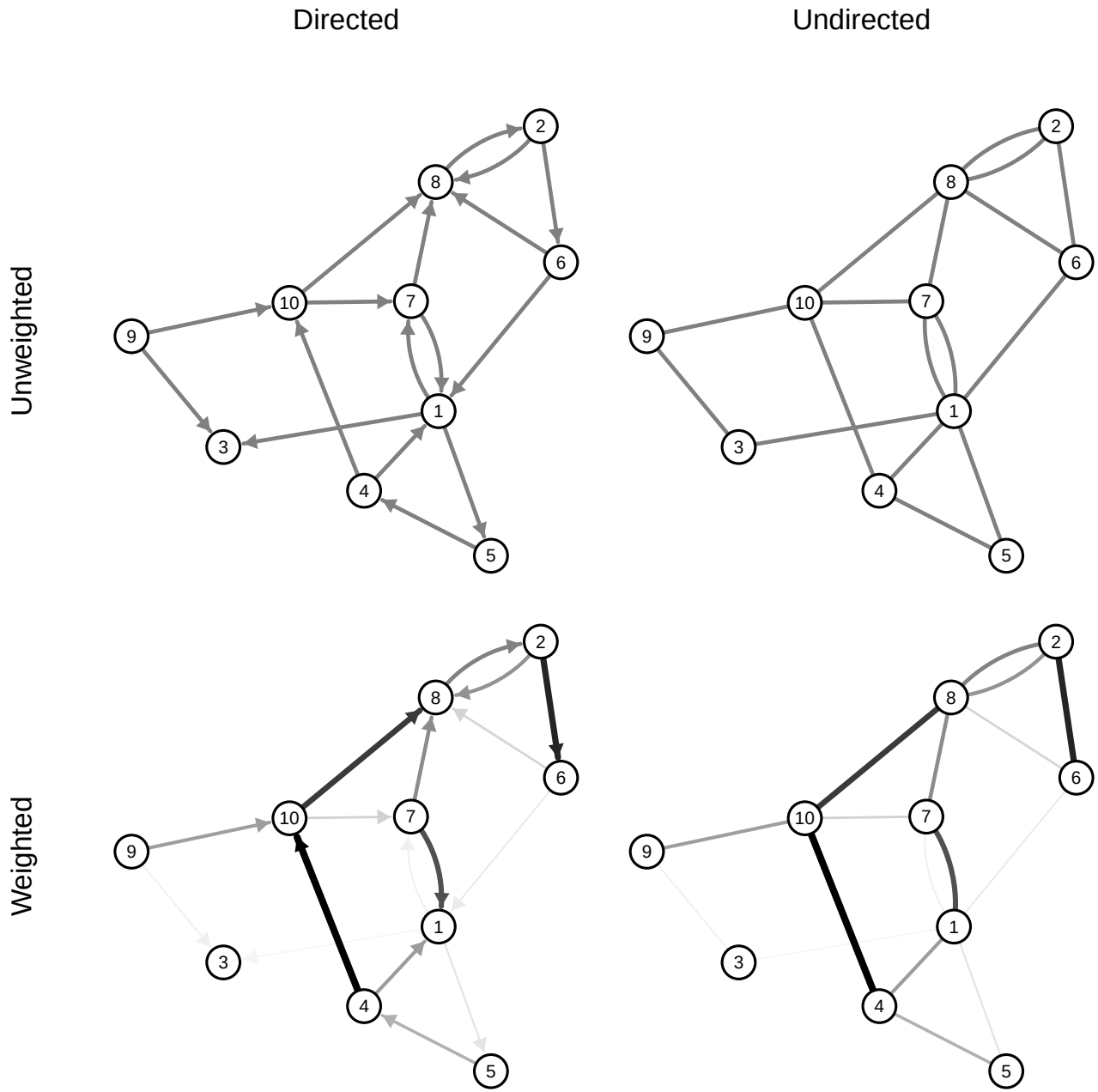
For programming technical questions regarding **R** see:
`http://stackoverflow.com/`

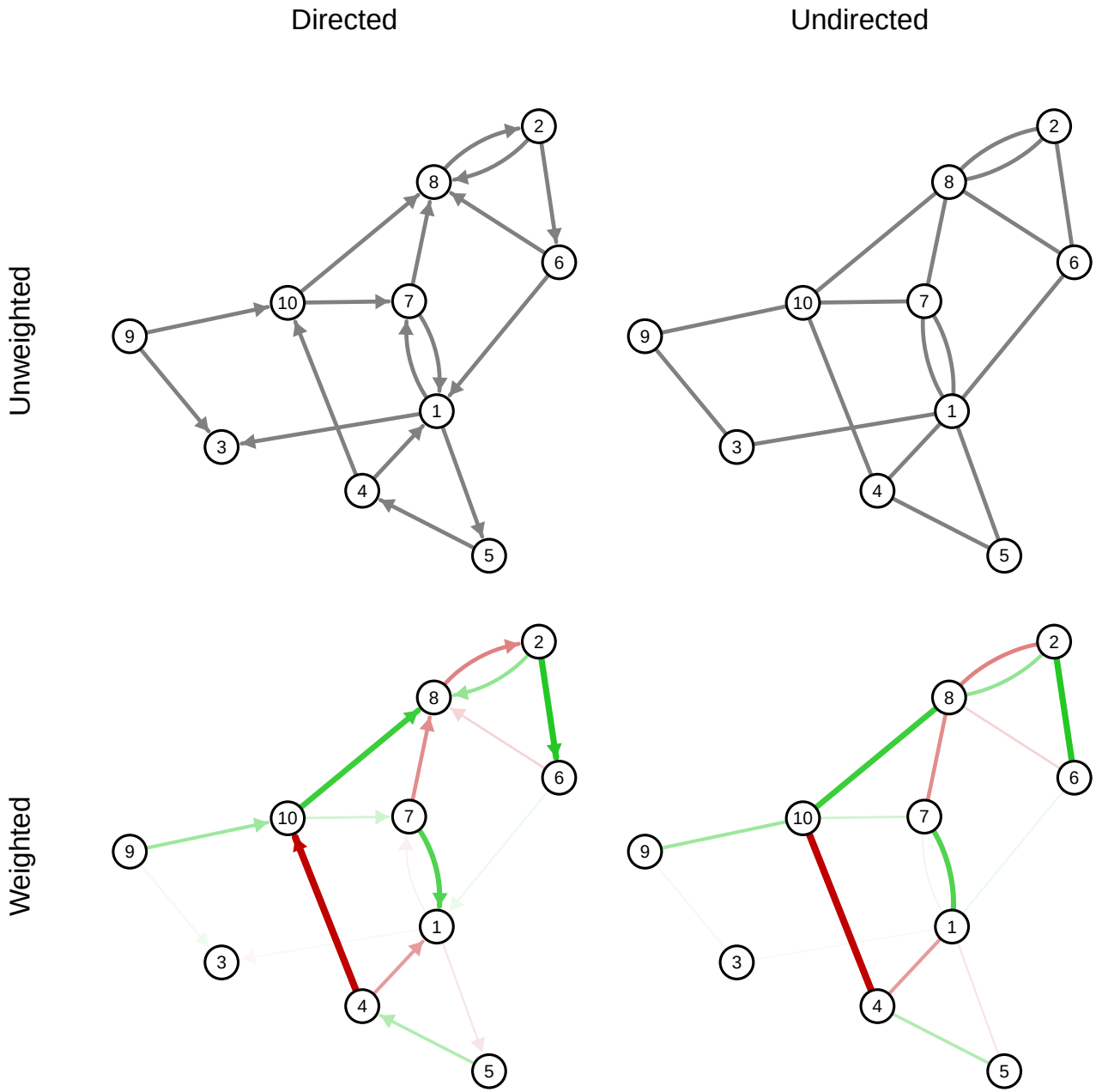
For statistical questions regarding **R** see:
`http://crossvalidated.com/`

In both of these make sure you use the tag `r` and include a reproducible example:
`http://stackoverflow.com/q/5963269/567015`

Graphs

- A graph is a *network* that consists of n nodes (or vertices) that are connected with m edges.
- Each edge can have a *weight* indicating the strength of that connection
- An edge can be directed (have an arrow) or undirected


Directed
Undirected
Unweighted
Weighted
1
2
3
4
5
6
7
8
9
10


Directed
Undirected
Unweighted
Weighted
1
2
3
4
5
6
7
8
9
10

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

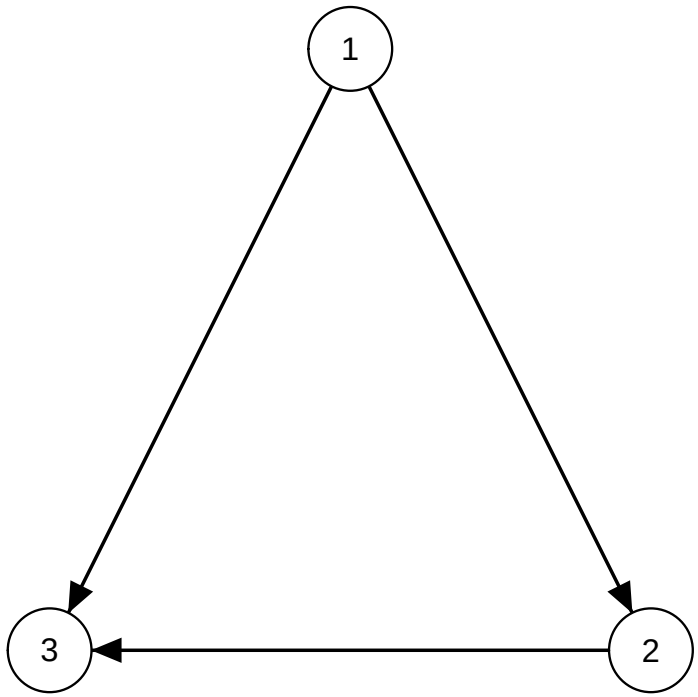
- The `input` argument is the input. This can be an *weights matrix*
- A weights matrix is a square *n* by *n* matrix in which each element indicates the relationship between two variables
- Any relationship can be used as long as:
  - A 0 indicates no relationship
  - Absolute negative values are similar 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 symetrical weights matrix is interpreted as an unweighted graph

Weights matrices

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

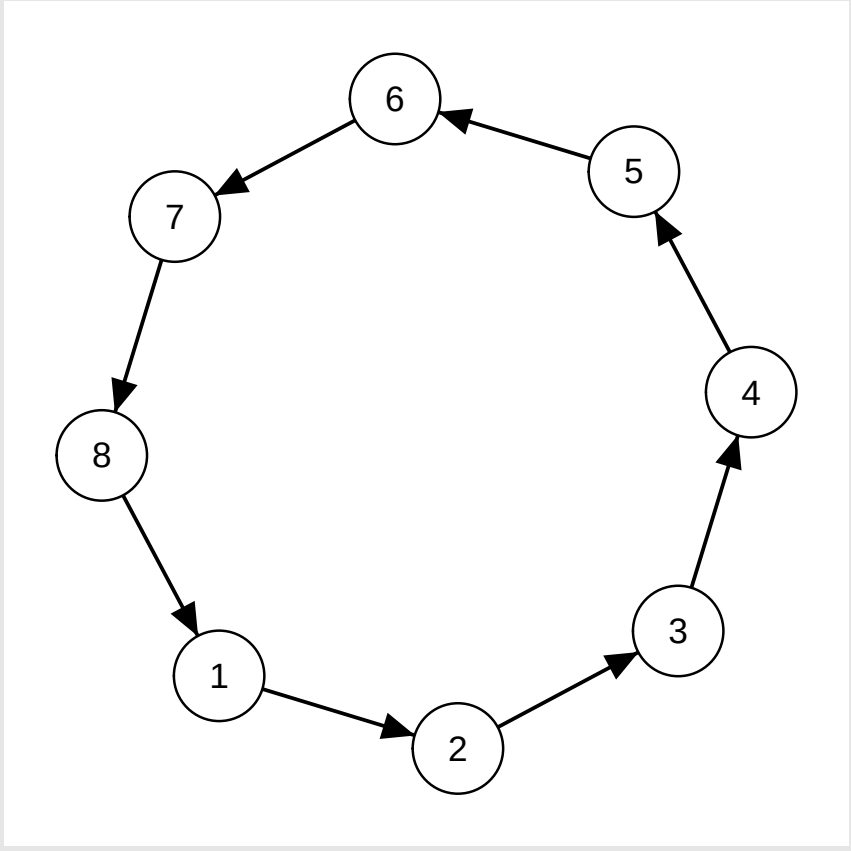
```
##      [,1] [,2] [,3]
## [1,]    0    1    1
## [2,]    0    0    1
## [3,]    0    0    0
```

```
qgraph(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:

```
##       [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
## [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:

```
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

```
print(input)
```

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

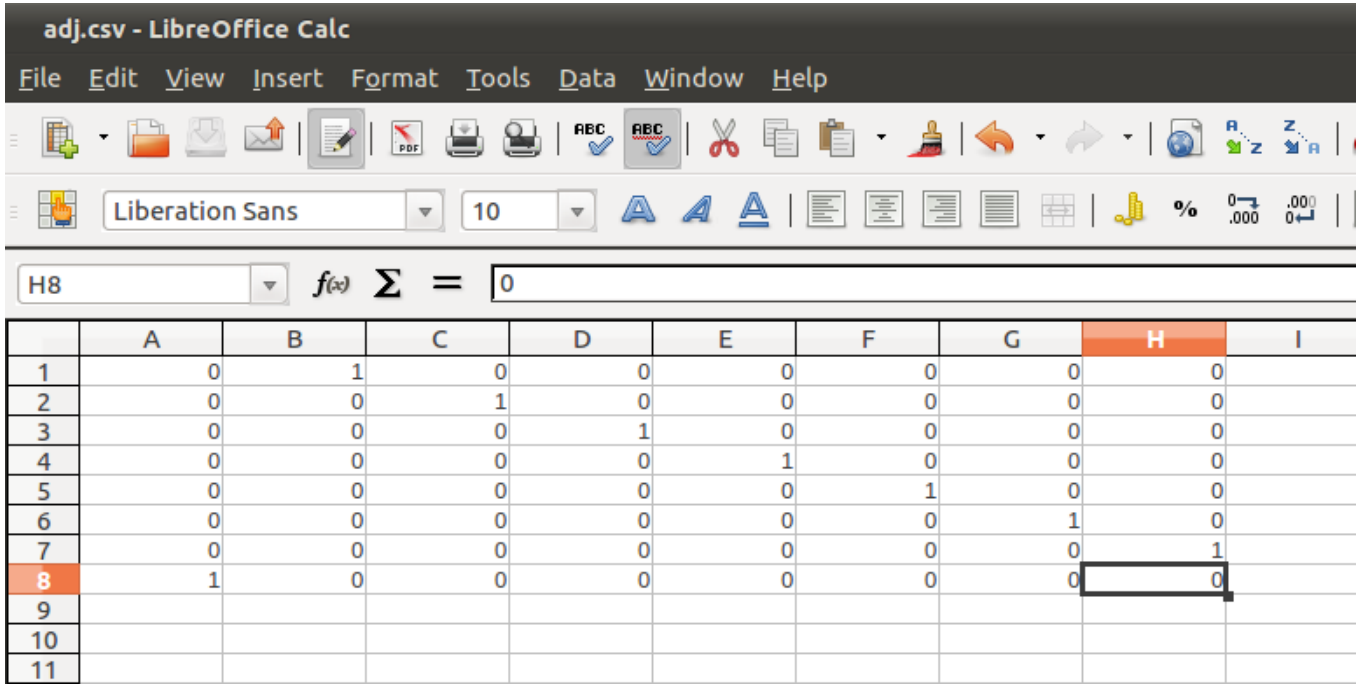
You can also change matrices manually (doesn't work in RStudio):

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

Or read the matrix from a text file!

Weights matrices

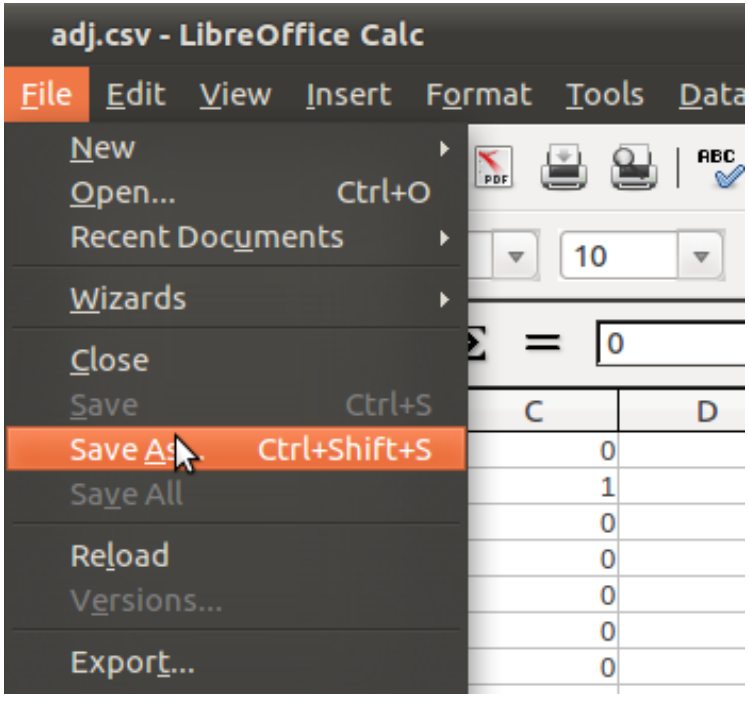
First make the matrix in a spreadsheet program (here LibreOffice)



	A	B	C	D	E	F	G	H	I
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	
9									
10									
11									

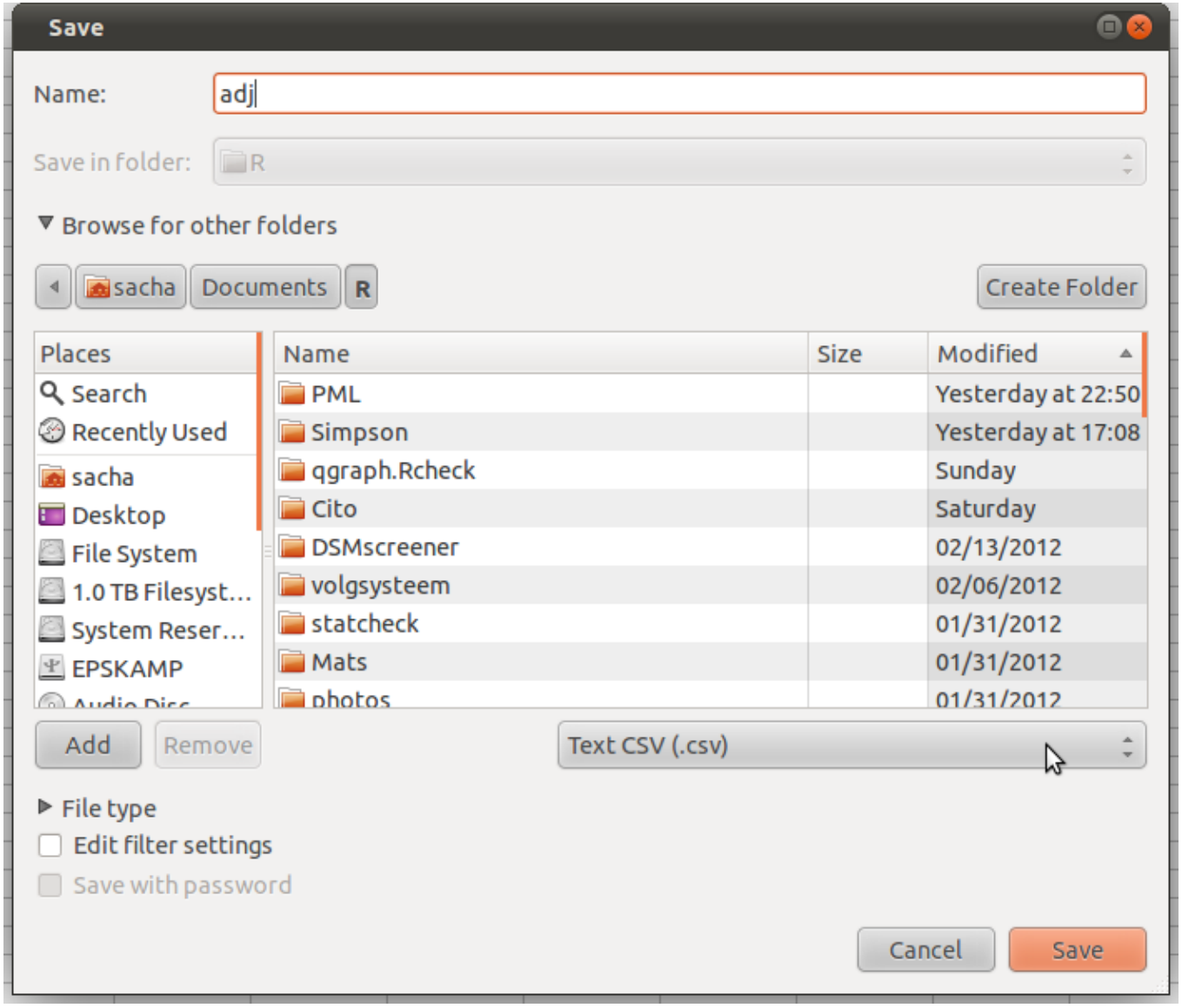
Weights matrices

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()`:

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

##   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
```

Weights matrices

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)

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

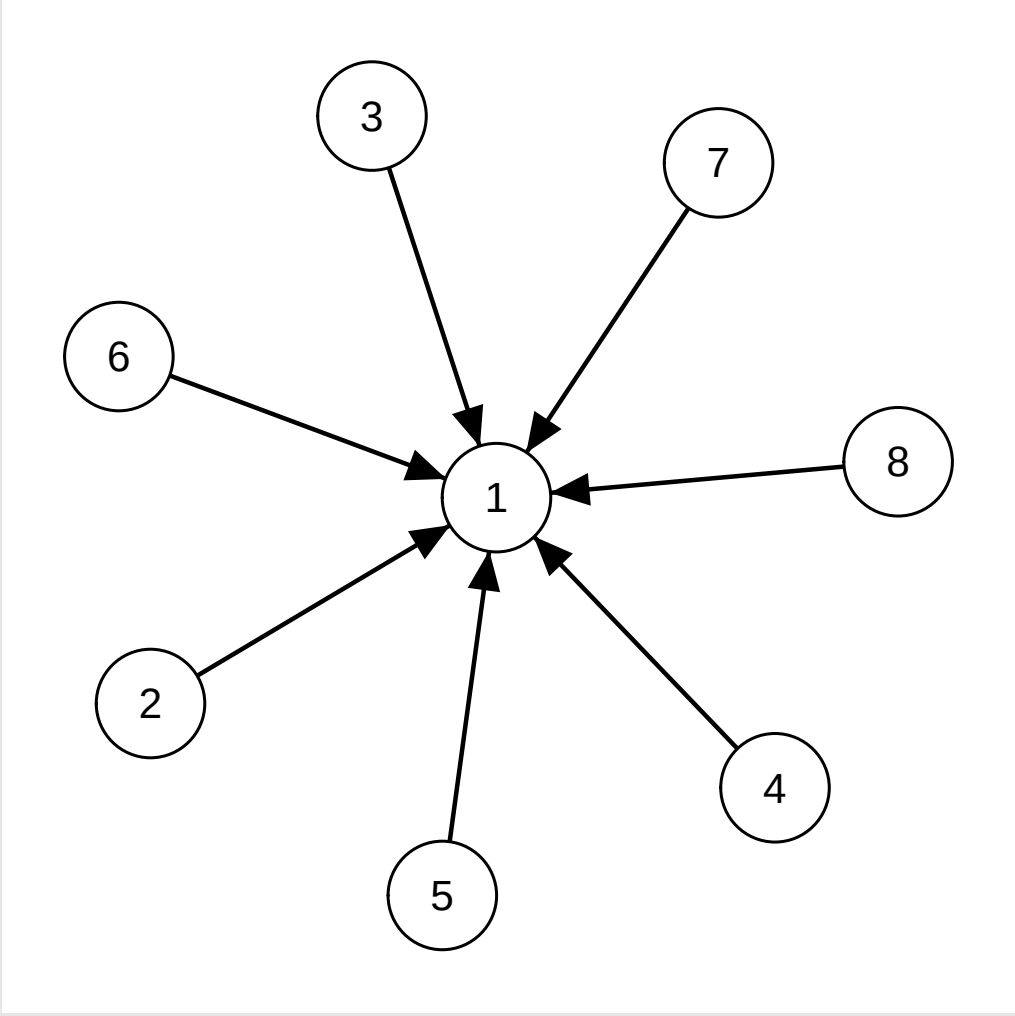
Weights matrices

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

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



Weights matrices

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

Edgelists

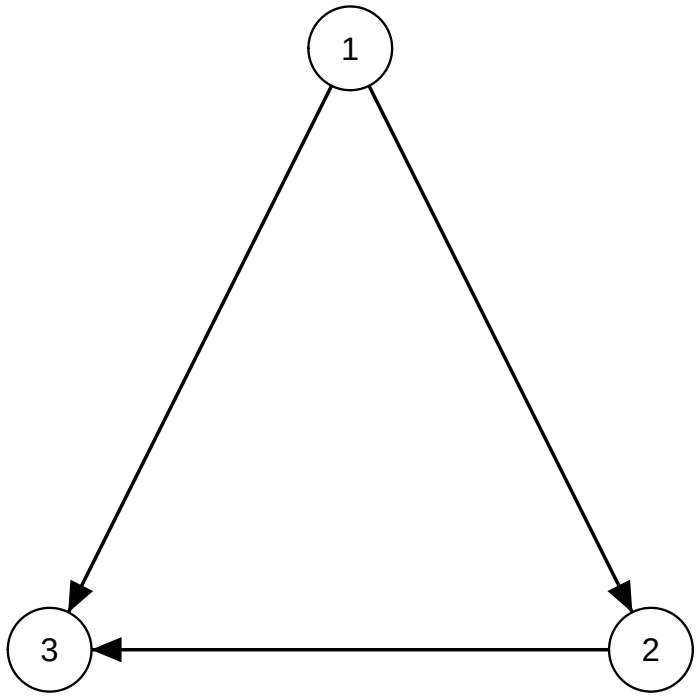
- The `input` argument can also be an *Edgelist*
- An Edgelist is not a list, but a matrix or dataframe
- 2 columns and a row for each edge
- The first column indicates the node of origin
- The second column indicates the node of destination
- These nodes must be indicated with an integer between 1 and *n*
  - The highest integer is interpreted as the number of nodes in the graph
    - can be changed with the `nNodes` argument
  - If an integer is missing it is considered a node without edges
  - Since 0.4.8 the edgelist can also contain characters
    - If the `labels` argument (vector containing the label of each node) is not specified these characters are used as labels
    - if `labels` is specified then each label not used is interpreted as a node without edges

Edgelists

```
qgraph(E)
```

```
E <- matrix(c(
  1,2,
  1,3,
        2,3),ncol=2,byrow=TRUE)
print(E)

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

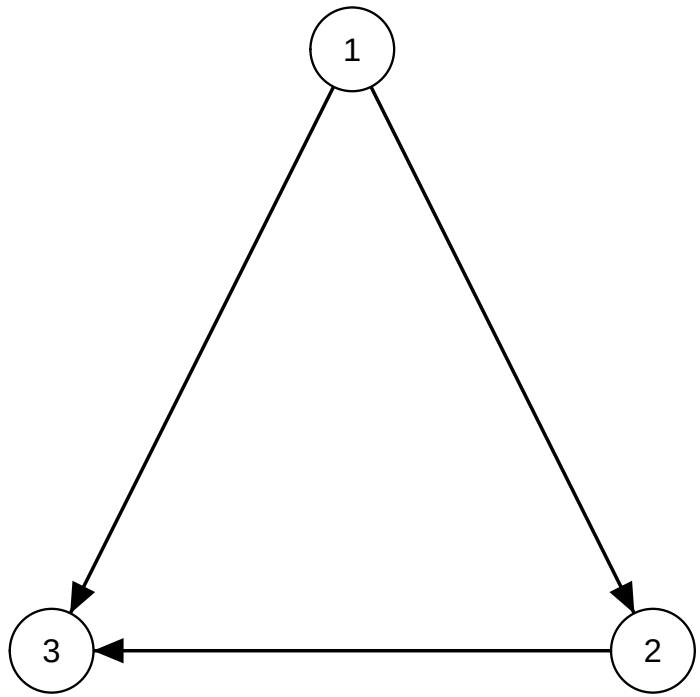


Edgelists

```
qgraph(E)
```

```
E <- data.frame(
        from = c(1,1,2),
        to = c(2,3,3))
print(E)

##   from to
## 1    1  2
## 2    1  3
## 3    2  3
```

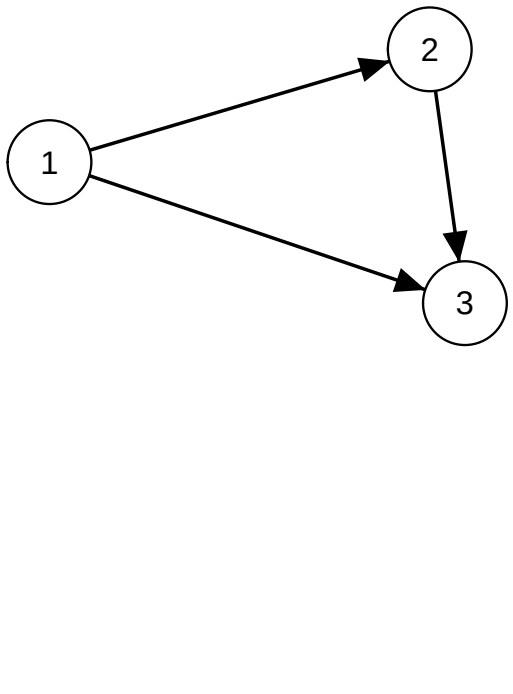


Edgelists

```
qgraph(E, nNodes = 4)
```

```
E <- data.frame(
   from = c(1,1,2),
          to = c(2,3,3))
print(E)

##    from to
## 1     1  2
## 2     1  3
## 3     2  3
```

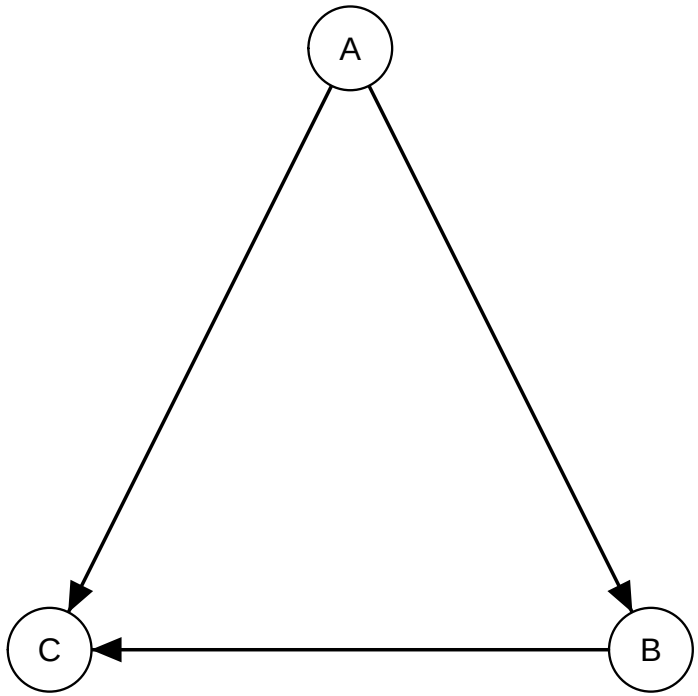


Edgelists

```
qgraph(E)
```

```
E <- data.frame(
   from = c("A","A","B"),
          to = c("B","C","C"))
print(E)

##   from to
## 1    A  B
## 2    A  C
## 3    B  C
```

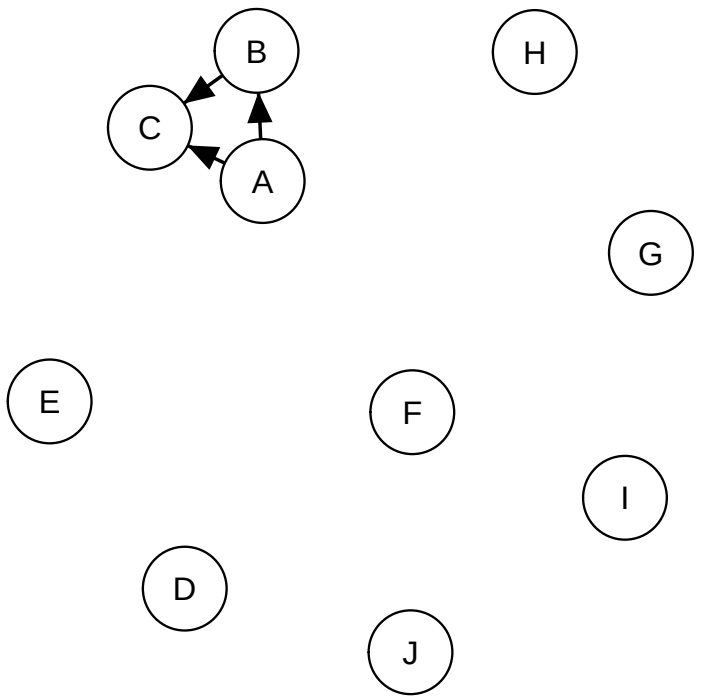


Edgelists

```
qgraph(E,
  labels = LETTERS[1:10])
```

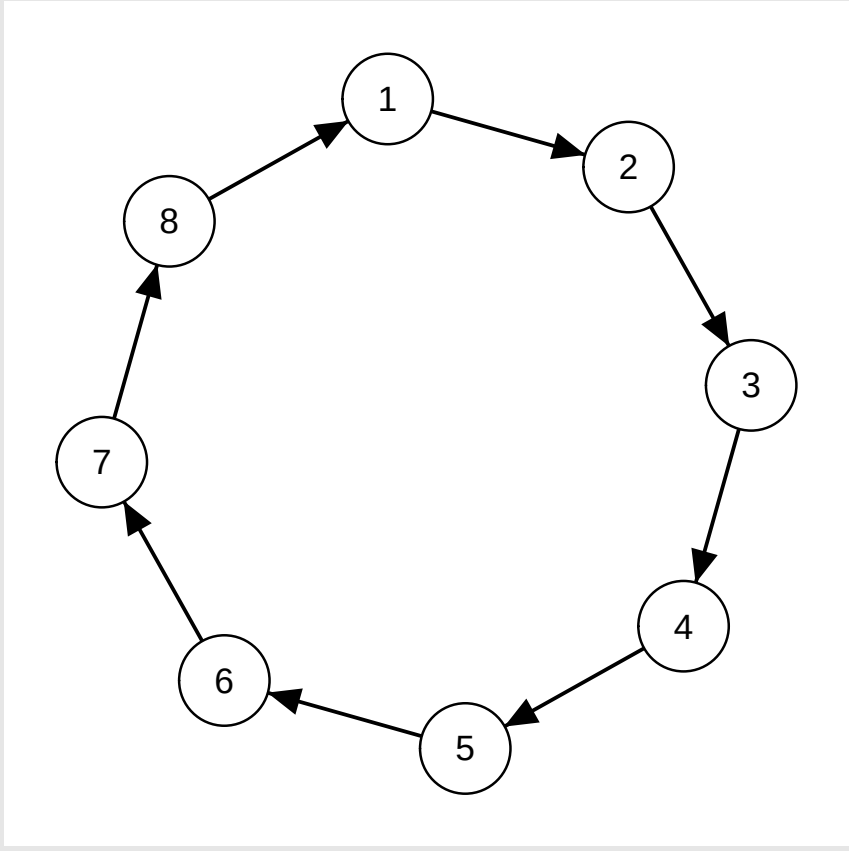
```
E <- data.frame(
        from = c("A","A","B"),
        to = c("B","C","C"))
print(E)

##   from to
## 1    A  B
## 2    A  C
## 3    B  C
```



Edgelists

Exercise 4: Create this graph, using an edgelist



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

Edgelists

```
E <- data.frame(from = 1:8,
    to = c(2:8, 1))
print(E)
```

```
##   from to
## 1    1  2
## 2    2  3
## 3    3  4
## 4    4  5
## 5    5  6
## 6    6  7
## 7    7  8
## 8    8  1
```

The `directed` argument

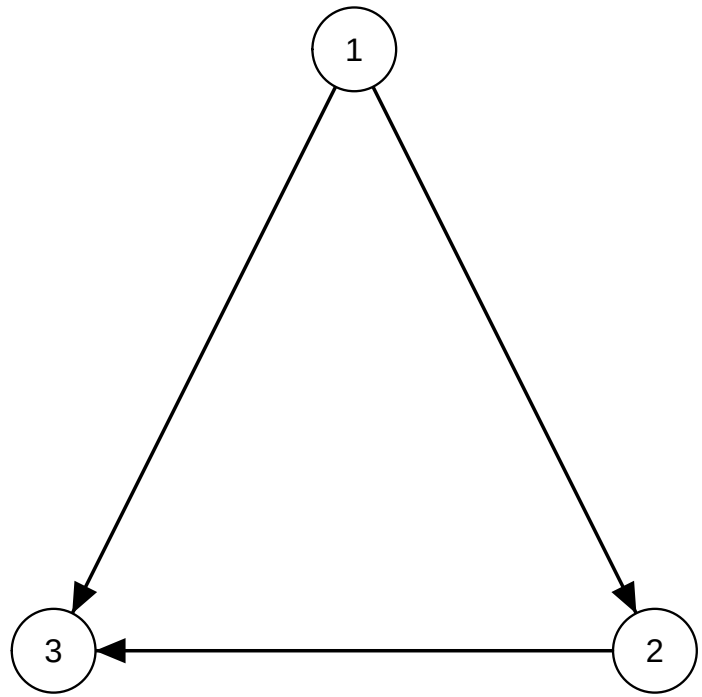
- An edgelist will always return a directed graph, and a weights matrix only a directed graph if it is assymetrical
- The `directed` argument can be used to force a directed (TRUE) or undirected (FALSE) graph
- This can also be specified per edge in a vector (edgelist) or matrix (weights matrix)

The `directed` argument

```
qgraph(E)
```

```
E <- data.frame(
        from = c(1,1,2),
        to = c(2,3,3))
print(E)

##   from to
## 1    1  2
## 2    1  3
## 3    2  3
```

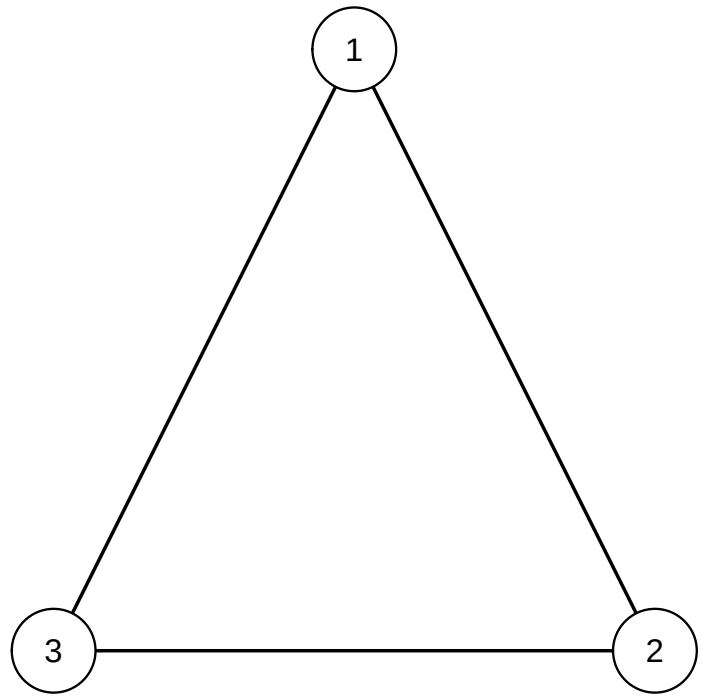


The `directed` argument

```
qgraph(E,directed=FALSE)
```

```
E <- data.frame(
        from = c(1,1,2),
        to = c(2,3,3))
print(E)

##   from to
## 1    1  2
## 2    1  3
## 3    2  3
```



The `directed` argument

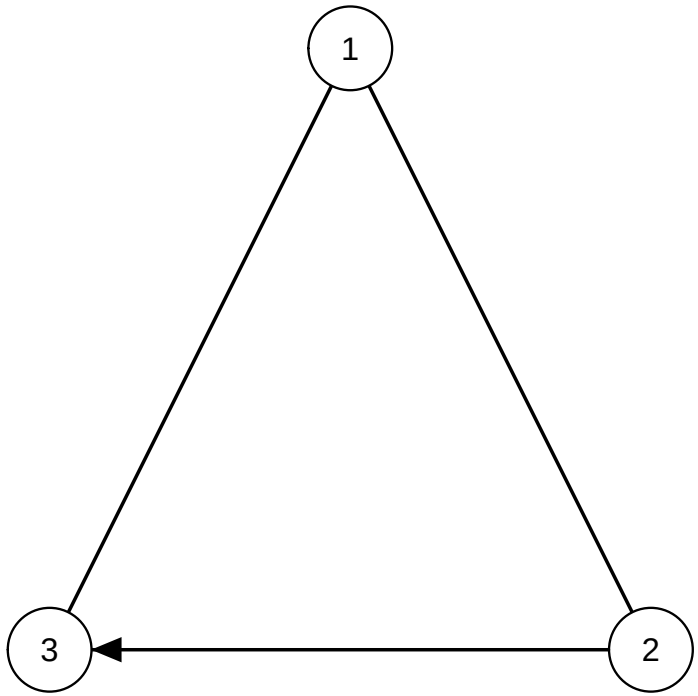
```
print(E)

##   from to
## 1    1  2
## 2    1  3
## 3    2  3

dir <- c(FALSE,FALSE,TRUE)
print(dir)

## [1] FALSE FALSE  TRUE
```

```
qgraph(E,directed=dir)
```

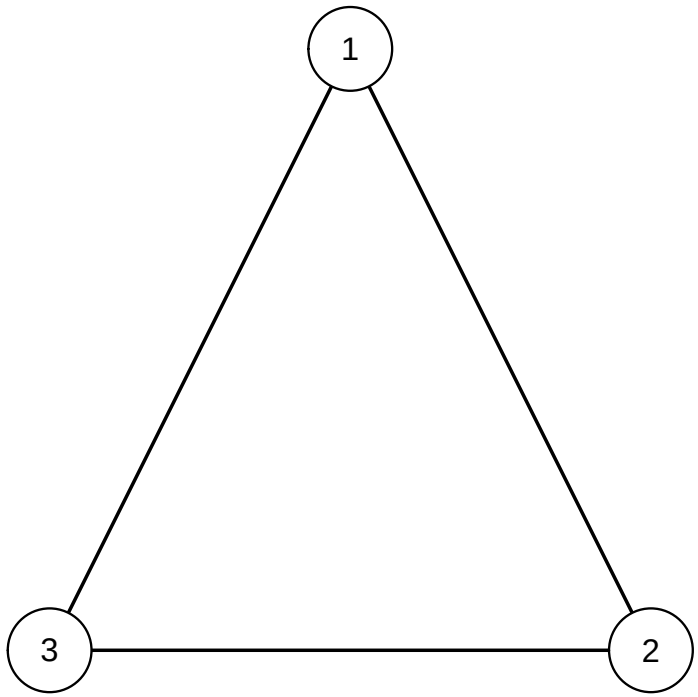


The `directed` argument

```
input<- matrix(1,3,3)
print(input)

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

```
qgraph(input)
```

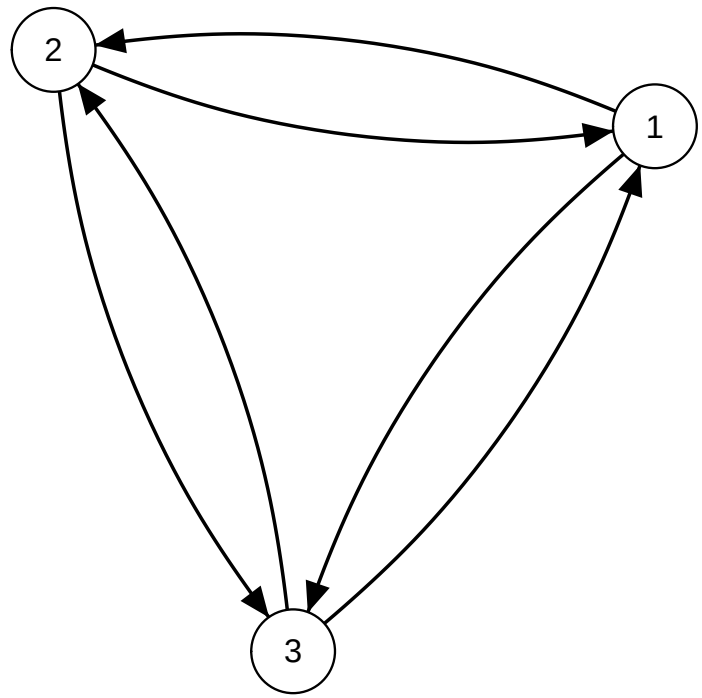


The `directed` argument

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

```
print(input)
```

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



The `directed` argument

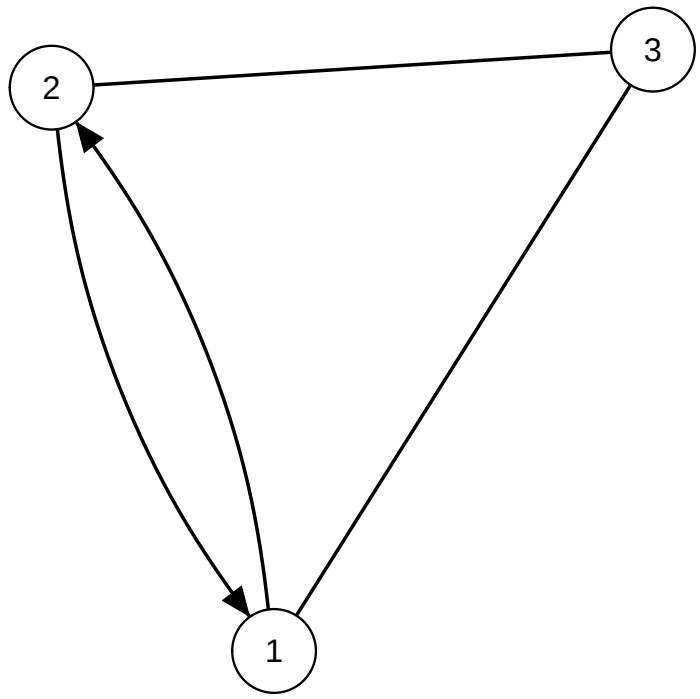
```
print(input)

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

dir <- matrix(c(
           FALSE,TRUE,FALSE,
           TRUE,FALSE,FALSE,
           FALSE,FALSE,FALSE)
  ,3,3,byrow=TRUE)
print(dir)

##       [,1]  [,2]  [,3]
## [1,] FALSE  TRUE FALSE
## [2,]  TRUE FALSE FALSE
## [3,] FALSE FALSE FALSE
```

```
qgraph(input,directed=dir)
```



The `bidirectional` argument

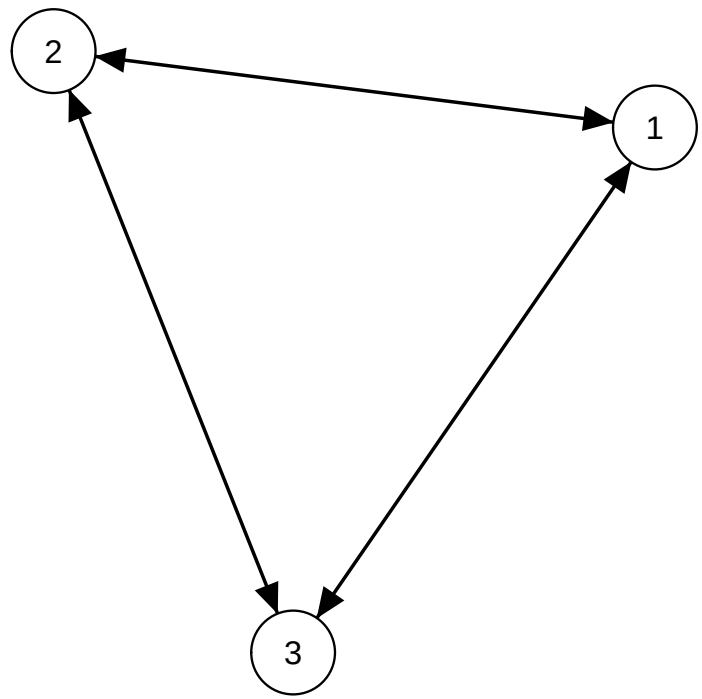
- Multiple directed edges between two nodes are curved
- To change this behavior, `bidirectional` can be set to TRUE
- Can also be a vector (edgelist) or matrix(weights matrix)

The `bidirectional` argument

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

```
print(input)

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



The `bidirectional` argument

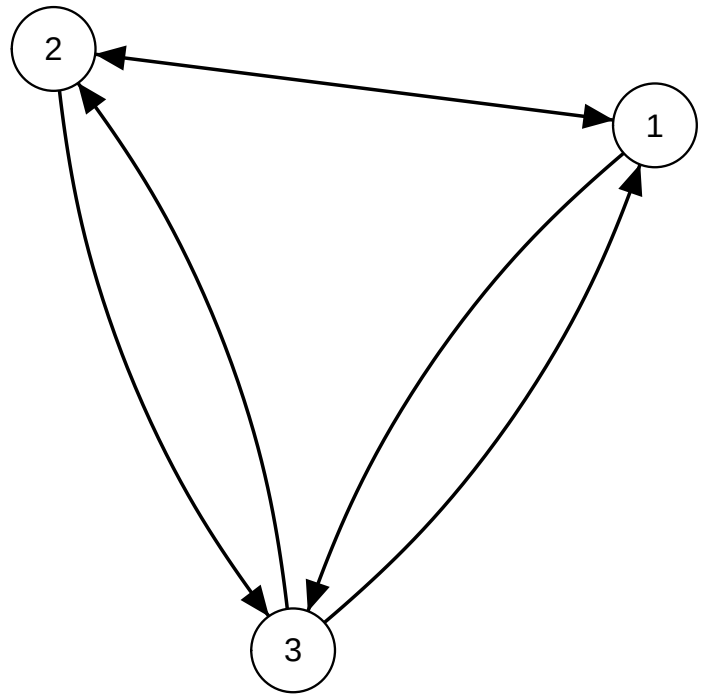
```
print(input)

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

bidir <- matrix(c(
           FALSE,TRUE,FALSE,
           TRUE,FALSE,FALSE,
           FALSE,FALSE,FALSE)
  ,3,3,byrow=TRUE)
print(bidir)

##       [,1]  [,2]  [,3]
## [1,] FALSE  TRUE FALSE
## [2,]  TRUE FALSE FALSE
## [3,] FALSE FALSE FALSE
```

```
qgraph(input,
  directed=TRUE,
  bidirectional=bidir)
```

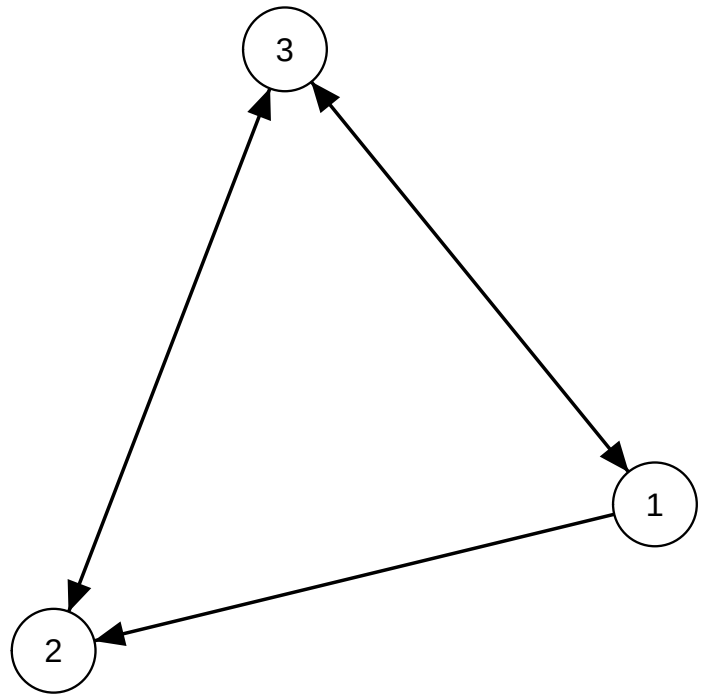


The `bidirectional` argument

```
qgraph(E, bidirectional=TRUE)
```

```
E <- data.frame(
        from = c(1,1,3,2,3),
        to = c(2,3,1,3,2))
print(E)

##   from to
## 1    1  2
## 2    1  3
## 3    3  1
## 4    2  3
## 5    3  2
```



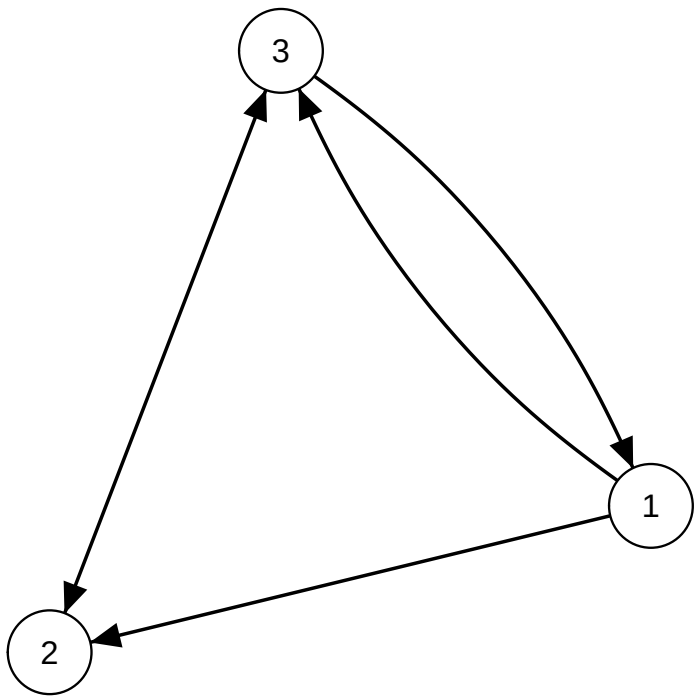
The `bidirectional` argument

```
print(E)

##    from to
## 1     1  2
## 2     1  3
## 3     3  1
## 4     2  3
## 5     3  2

bidir <- c(FALSE,FALSE,
  FALSE,TRUE,TRUE)
```

```
qgraph(E,
  bidirectional=bidir)
```



Arguments for directed graphs

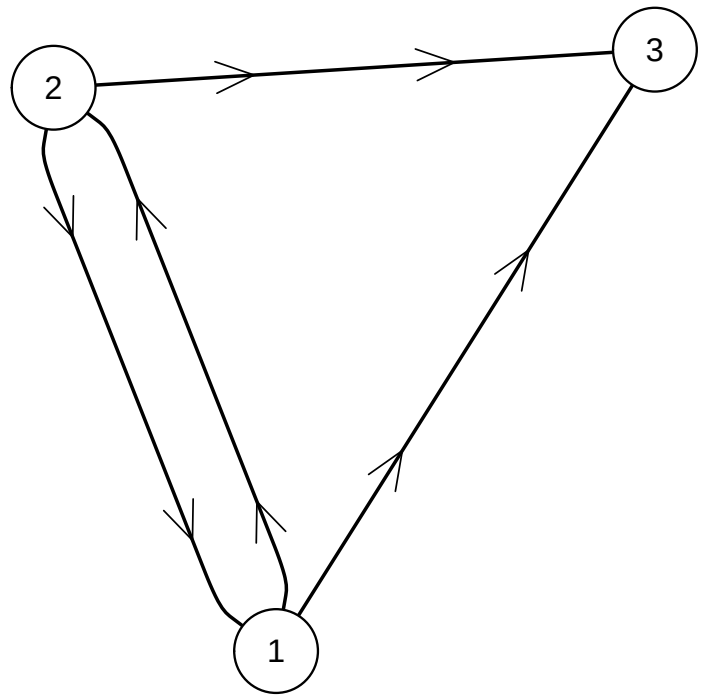
- Two other arguments can be used this way:
  - `curve` to curve each edge
  - `lty` to create dashed lines (not yet in matrix form)
- And finally a few other arguments:

Arguments for directed graphs

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

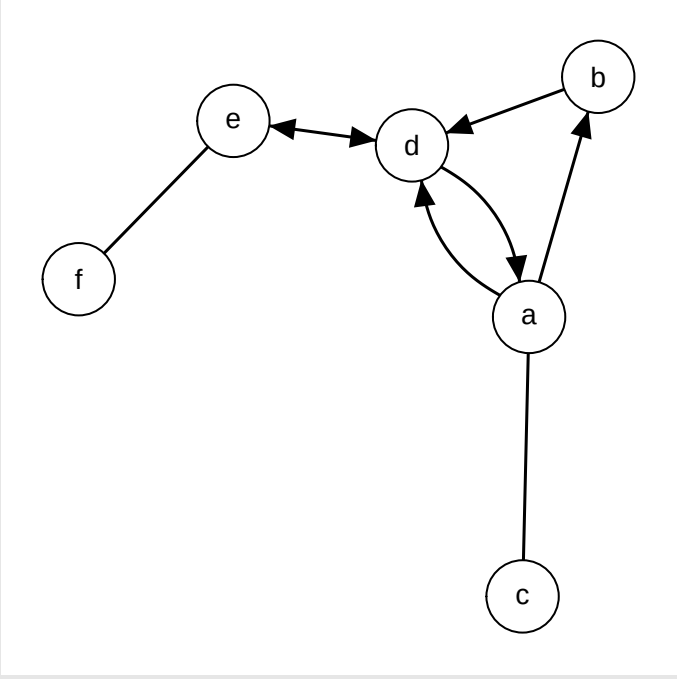
```
##      [,1] [,2] [,3]
## [1,]    0    1    1
## [2,]    1    0    1
## [3,]    0    0    0
```

```
qgraph(input,
  asize=10,
  arrows=2,
  open=TRUE,
  curvePivot=TRUE)
```



Arguments for directed graphs

Exercise: Create this graph



- 6 arrows and 2 unweighted edges, so 8 edges total!
- Look at the helpfiles to lookup the arguments again!

```
?qgraph
```

Weighted graphs

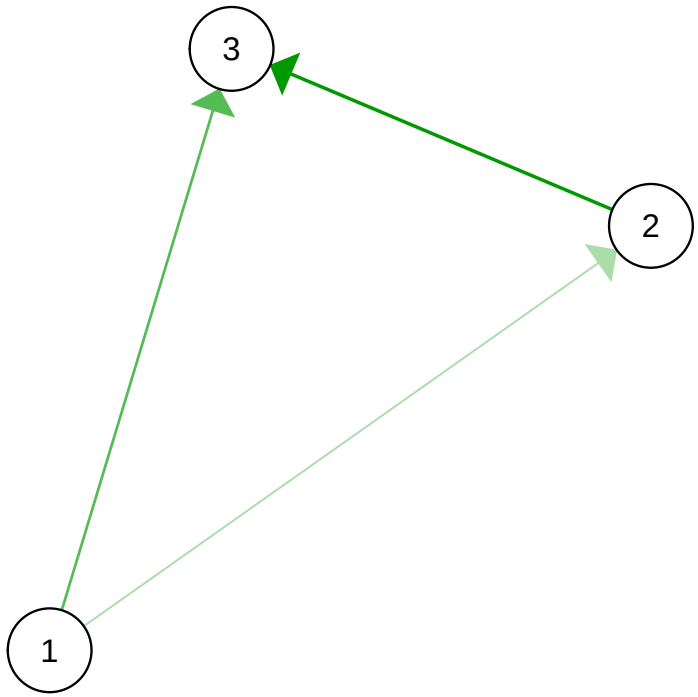
- Specify edge weights to make a graph weighted
  - In an edgelist: Add a third column containing edge weights
  - In a weights matrix: simply specify other values than only zeros and ones
- An edge weigth of 0 indicates no connection
- Positive and negative edge weights must be comparable in strength
- 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
```

```
qgraph(input)
```

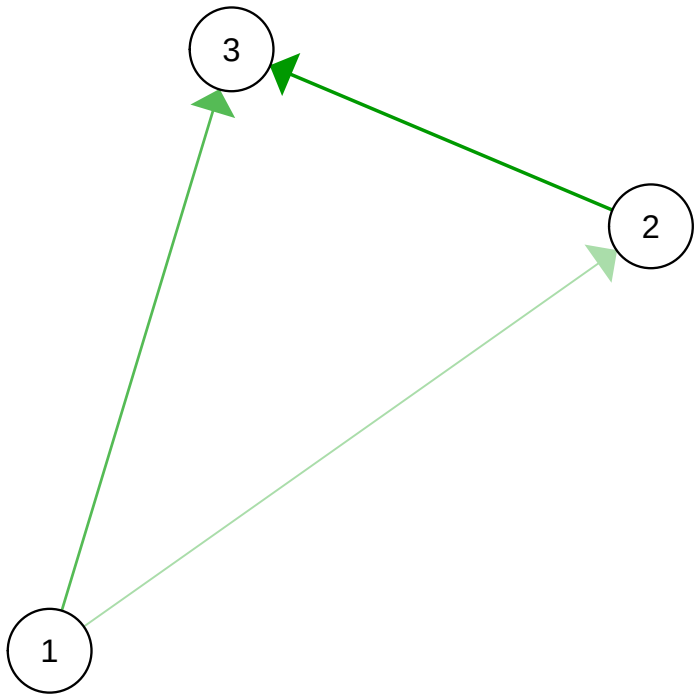


Weighted graphs

```
E <- data.frame(
        from   = c(1, 1, 2, 1),
        to     = c(2, 3, 3, 1),
        weight = c(1, 2, 3, 0))
print(E)
```

```
##   from to weight
## 1    1  2      1
## 2    1  3      2
## 3    2  3      3
## 4    1  1      0
```

```
qgraph(E)
```



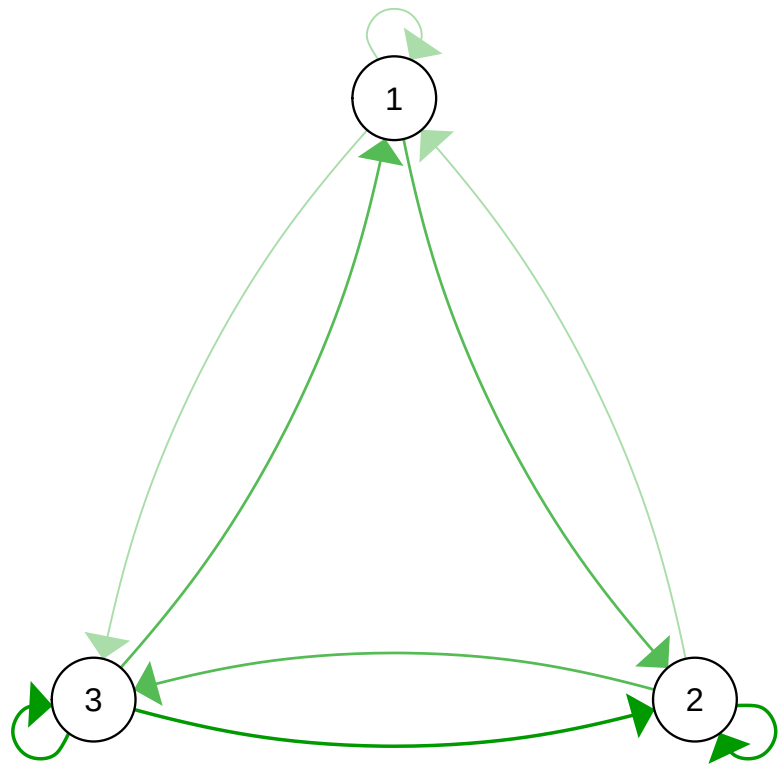
Weighted graphs

Look out, a square matrix is interpreted as weights matrix!

```
E <- data.frame(
        from   = c(1, 1, 2),
        to     = c(2, 3, 3),
        weight = c(1, 2, 3))
print(E)

##   from to weight
## 1    1  2      1
## 2    1  3      2
## 3    2  3      3
```

```
qgraph(E)
```



Layout modes

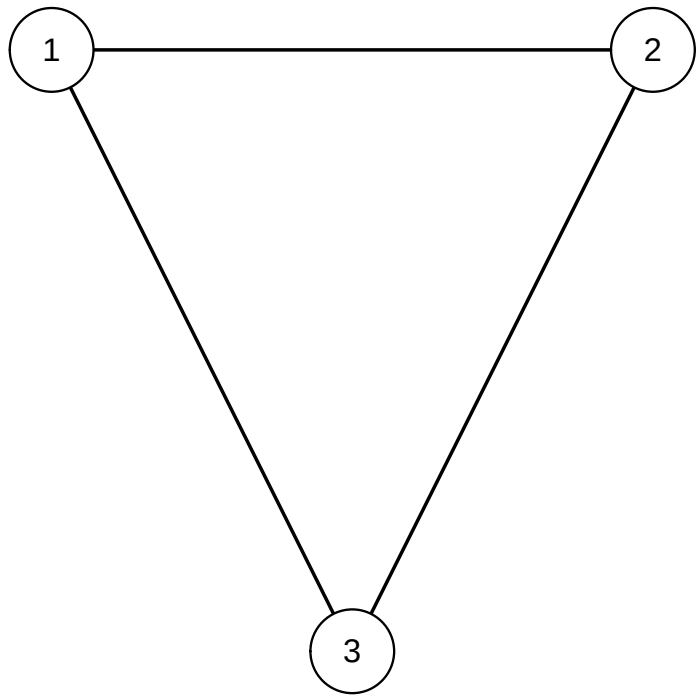
- The placement of the nodes is specified with the `layout` argument in `qgraph()`
- This can be a *n* by 2 matrix indicating the *x* and *y* position of each node
- `layout` can also be given a character indicating one of the two default layouts
  - If `layout="circular"` the nodes are placed in circles per group (if the `groups` list is specified)
  - If `layout="spring"` a force-embedded algorithm (?, ?) is used for the placement
- And a final option is to specify a grid-like layout

Layout matrix

```
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
## [2,]  1.0    1
## [3,]  0.5    0
```

```
qgraph(input, layout = L)
```

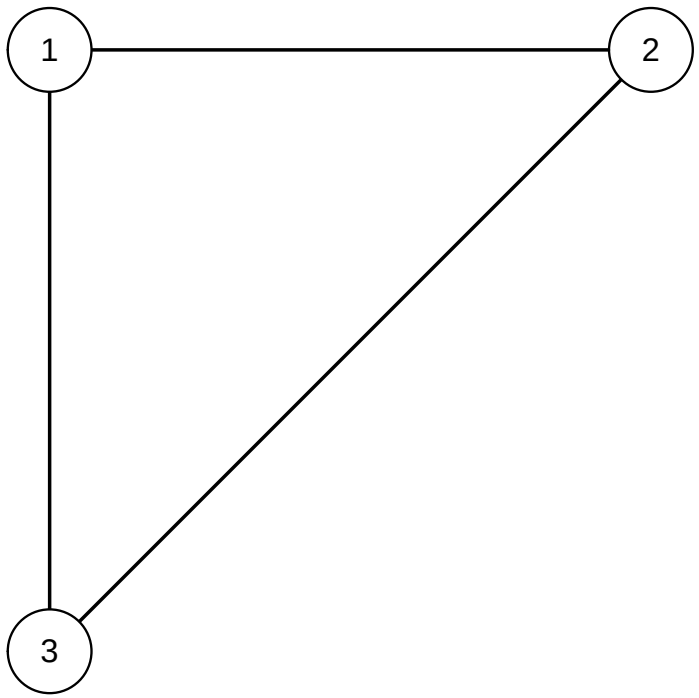


Layout matrix

```
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
```



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

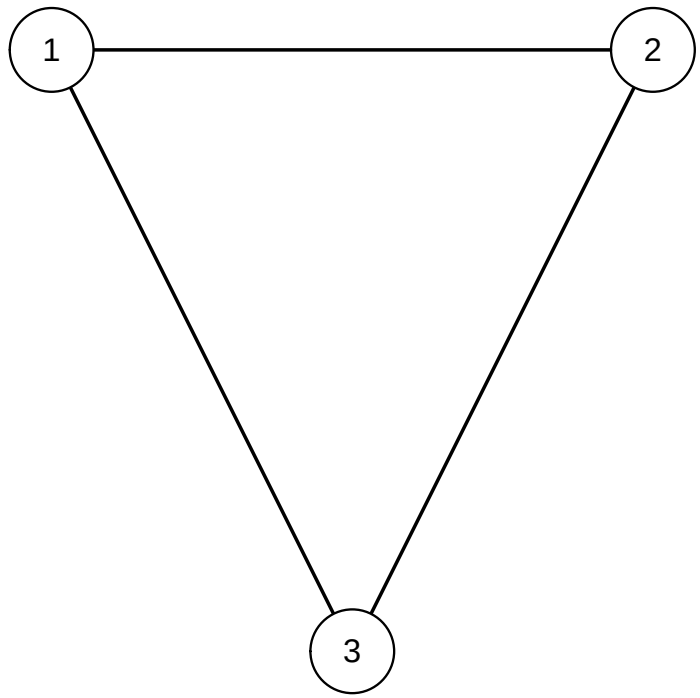
```
Q <- qgraph(input)
qgraph(input2, layout = Q$layout)
```

Grid layout

```
qgraph(input, layout = L)
```

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

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

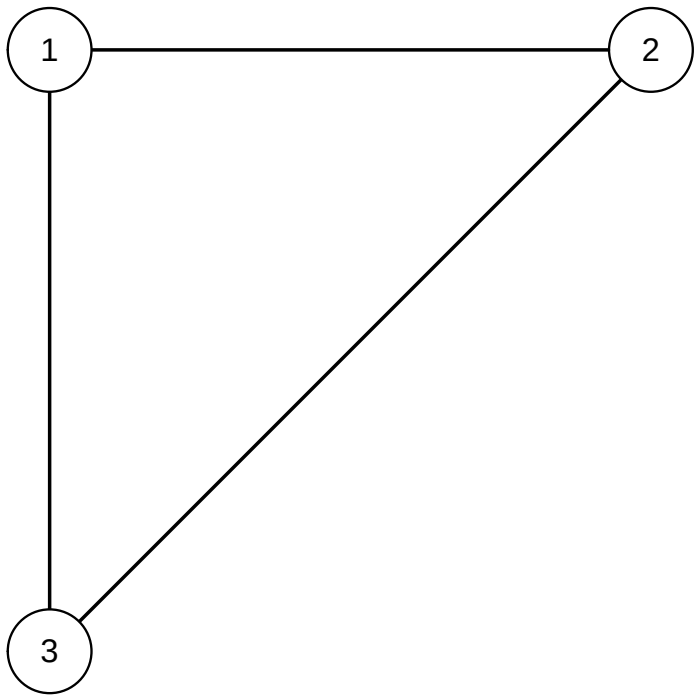


Grid layout

```
qgraph(input, layout = L)
```

```
input <- matrix(1,3,3)
L <- matrix(c(
         1,0,2,
         3,0,0),nrow=2,byrow=TRU
print(L)
```

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



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
- After this process the layout is rescaled to fit the -1 to 1 *xy*-plane
- The unscaled layout is returned as `layout.orig`

Big 5

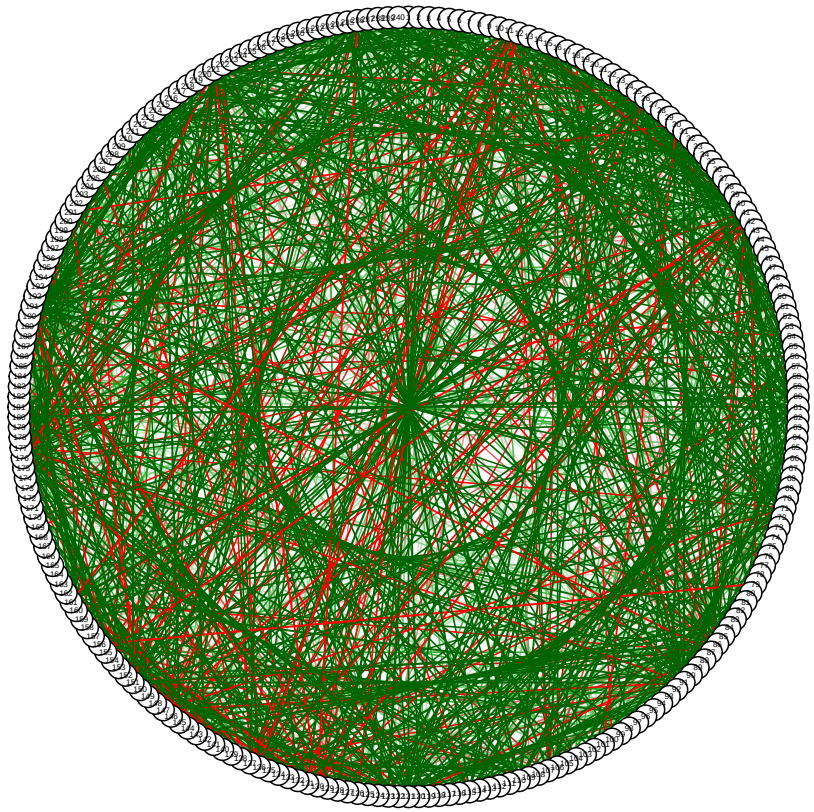
Load the big 5 dataset:

```
data(big5)
str(big5)

##  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)
```



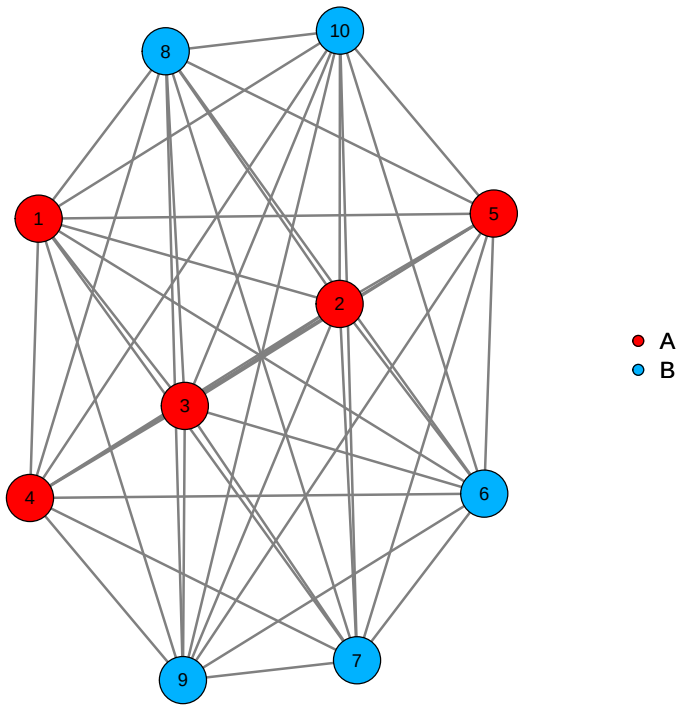
The `groups` argument

- 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

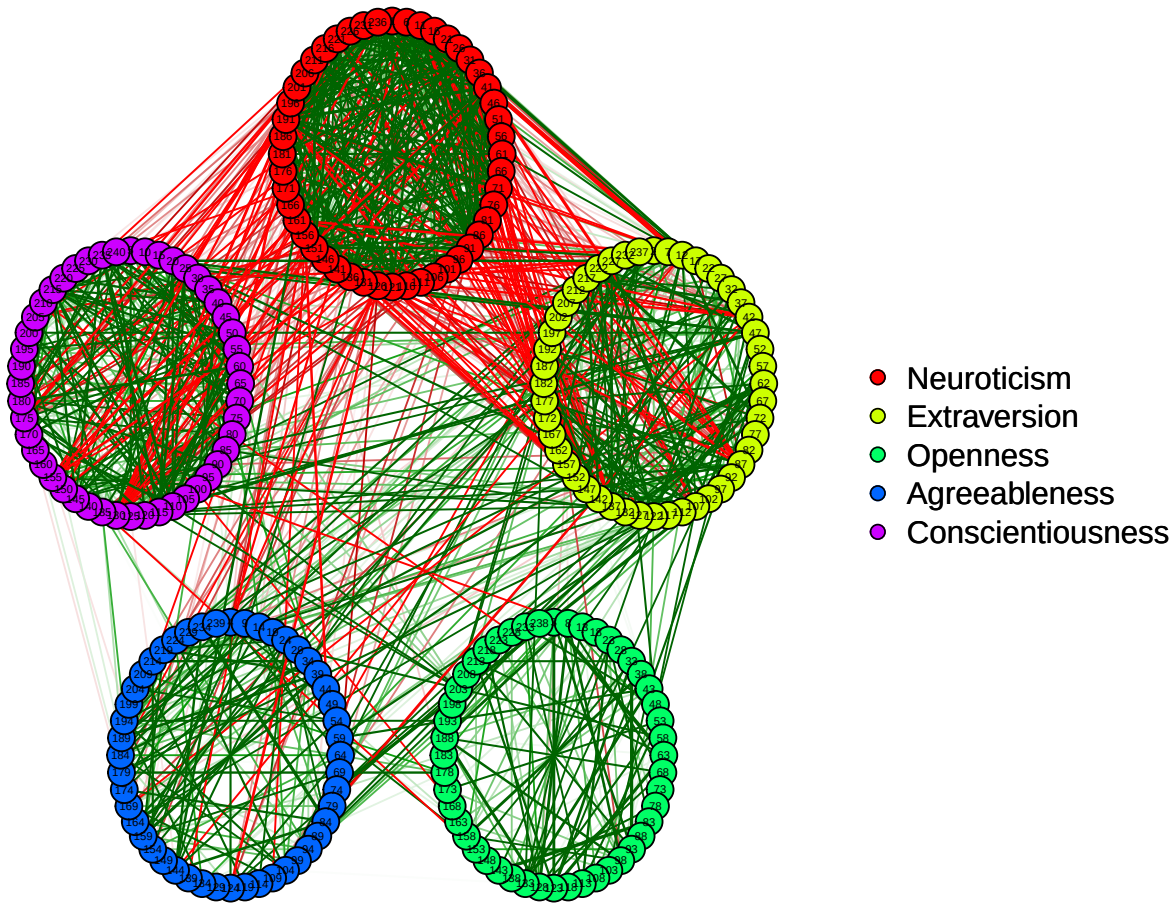
The `groups` argument

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



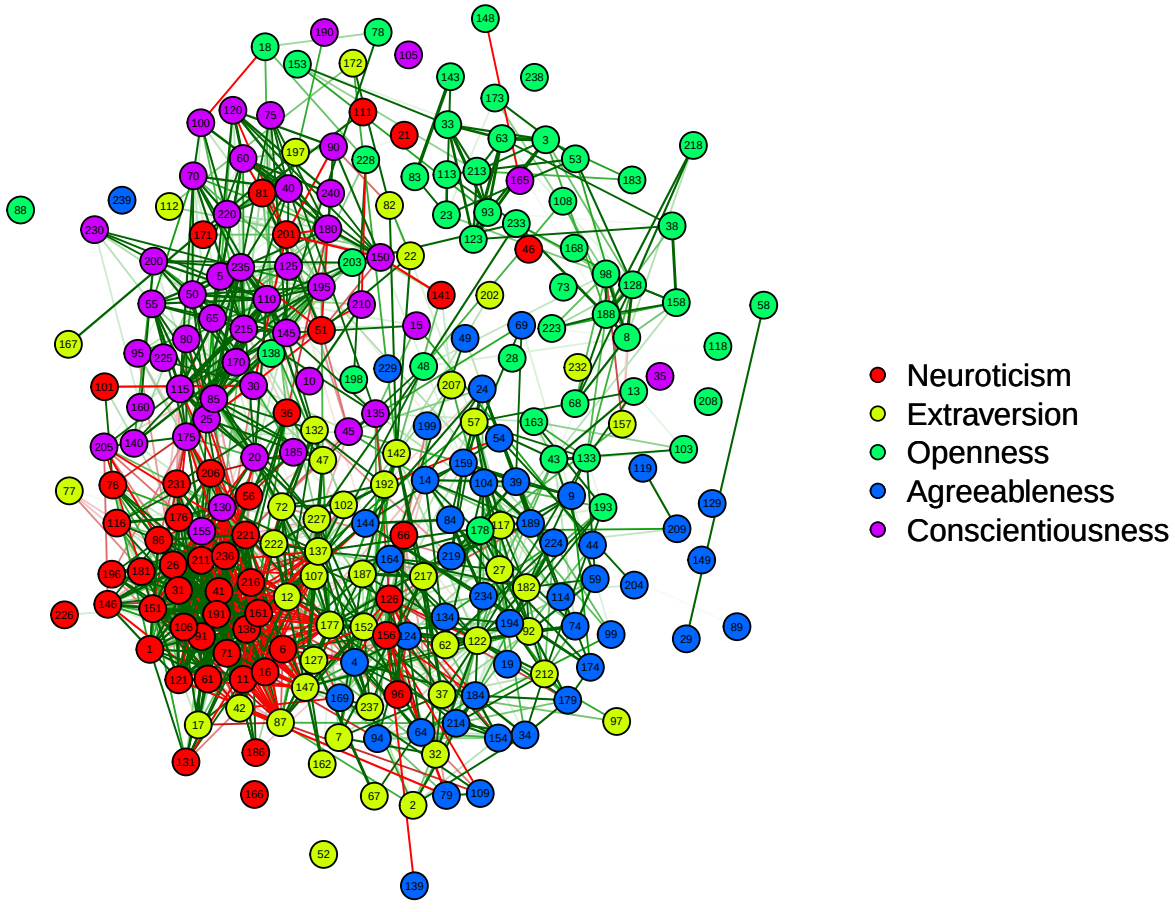
Big 5

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



Big 5

```
qgraph(big5graph, layout = "spring")
```



output

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!

Export to PDF

```
# 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 untill 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`)

Contribute to **qgraph**

The devellopmental 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!