# A Bioconductor package for investigation of network heterogeneity from high-dimensional data

Nicolas Städler*, Frank Dondelinger†

May 15, 2016

## 1 Introduction

Data analysis in systems biology and medicine often requires analysing data whose dynamics can be described as a network of observed and unobserved variables. A simple example is a protein signalling network in a cell.

Simplifying the process greatly, signalling proteins known as kinases can be unphosphorylated (inactive) or phosphorylated (active). Cell signalling uses the phosphorylation machinery to pass messages from the exterior of the cell to the interior where they will be acted upon. This message passing is achieved via a relay of kinases and other proteins (the signalling pathway), which can be thought of as a network.

Numerous software packages exist for reconstructing networks from observational data (e.g. [**?**], [**?**], [**?**], [**?**]). However, most of these packages assume that there is a single underlying network. Package `nethet` was designed with the intent of handling heterogeneous datasets arising from a collection of (possibly related) networks.

Take for example protein measurements of breast cancer tumor cells. It is known that there exist several subtypes of breast cancer with different molecular profiles [**?**]. We might be interested in whether the signalling pathways (networks) reconstructed from two subtypes are statistically different. If they are not, then we might want to identify new subtypes that present different molecular profiles, and reconstruct the networks for each identified subtype. The `nethet` package contains functionalities to tackle all of these tasks.

To the best of our knowledge, `nethet` is currently the only implementation of statistical solid methodology enabling the analysis of network heterogeneity from high-dimensional data. Package `nethet` combines several implementations of recent statistical innovations useful for estimation and comparison of networks in a heterogeneous, high-dimensional setting. In particular, we provide code for formal two-sample testing in Gaussian graphical models (differential network and GGM-GSA; [**?**], [**?**]) and make a novel network-based clustering algorithm available (mixed graphical lasso, [**?**]).

## 2 Statistical setup

We consider independent samples $X_i \in \mathbb{R}^p$ $(i = 1, \ldots, n)$, measuring $p$ molecular variables. We assume that the collected data can be divided into $K$ different groups. Let $S_i \in \{1, \ldots, K\}$ be the group assignment of sample $i$, denote with $n_k$ the group specific sample size and write $\mathbf{X}_k$ for the $n_k \times p$ data matrix consisting of all samples belonging to group $k$.

To describe networks we use Gaussian graphical models (*GGMs*, [**?**]). These models use an undirected graph (or network) to describe probabilistic relationships between variables. For each group $k$, we assume that $\mathbf{X}_k$ is sampled from a multivariate Gaussian distribution with (unknown) mean $\mu_k$ and (unknown) $p \times p$ concentration matrix $\Omega_k = \Sigma_k^{-1}$. The

*staedler.n@gmail.com
†fdondelinger.work@gmail.com

matrix $\Omega_k$ defines the group-specific graph $G_k$ via

$$(j, j') \in E(G_k) \Leftrightarrow \Omega_{k;jj'} \neq 0,$$
$$j, j' \in \{1, \ldots, p\} \text{ and } j \neq j',$$

where $E(G)$ denotes the edge set of graph $G$.

Learning of networks $G_k$ is a so-called high-dimensional statistical problem. We employ regularization to learn sparse, parsimonious networks and thereby control over-fitting. In particular, we use the popular *graphical Lasso* [**?**, **?**]. Frequently the group assignments $S_i$, as well as the number of groups $K$, are unknown at the outset and have to be inferred simultaneously with the group-specific mean vectors and networks. The method *mixglasso*, implemented in this package, is a novel tool for high-dimensional, network-based clustering. It is based on a finite mixture of GGMs and employs an adaptive and automatic penalization scheme [**?**].

Network inference is subject to statistical uncertainty and observed differences between estimated networks may be due to noise in the data and variability in estimation rather than any true difference in underlying network topology. Testing hypotheses of the form

$$\mathbf{H_0} : G_k = G_{k'}, \quad k, k' \in \{1, \ldots, K\},\ k \neq k'$$

is challenging. We build upon a recent approach called *differential network* [**?**, **?**] which allows formal two-sample testing in high-dimensional GGMs.

## 3 Package functionalities

The package consists of the following main parts:

- Simulation functions for creating synthetic data from the underlying Gaussian mixture (network) model.
- Network inference using the `het_cv_glasso` function for reconstructing heterogeneous networks from data with the graphical Lasso [**?**] when the group structure is known.
- High-dimensional hypothesis testing capabilities, including the `diffnet` functions implementing a statistical test for whether the networks underlying a pair of dataset are different, the `ggmgsa` functions allowing for differential gene set testing and the `diffregr` functions testing whether two high-dimensional regression models are statistically different [**?**, **?**].
- The `mixglasso` functions implementing a network-based clustering and reconstruction algorithm also based on the graphical Lasso, for unknown group structure [**?**].
- Plotting and export functions for displaying and saving the results of the analysis in a sensible way.

## 4 Simulate data

In order to demonstrate the functionalities of the package, we will first simulate data from a Gaussian mixture model with a known covariance structure. The *nethet* package includes code for generating random covariance matrices with a given sparsity, and for simulating from a Gaussian mixture model with given means and covariances. The function `sim_mix_networks` provides a convenient wrapper for both:

```
# Specify number of simulated samples and dimensionality
n = 100
p = 25

# Specify number of components of the mixture model and mixture probabilities
n.comp = 4

mix.prob = c(0.1, 0.4, 0.3, 0.2)
```

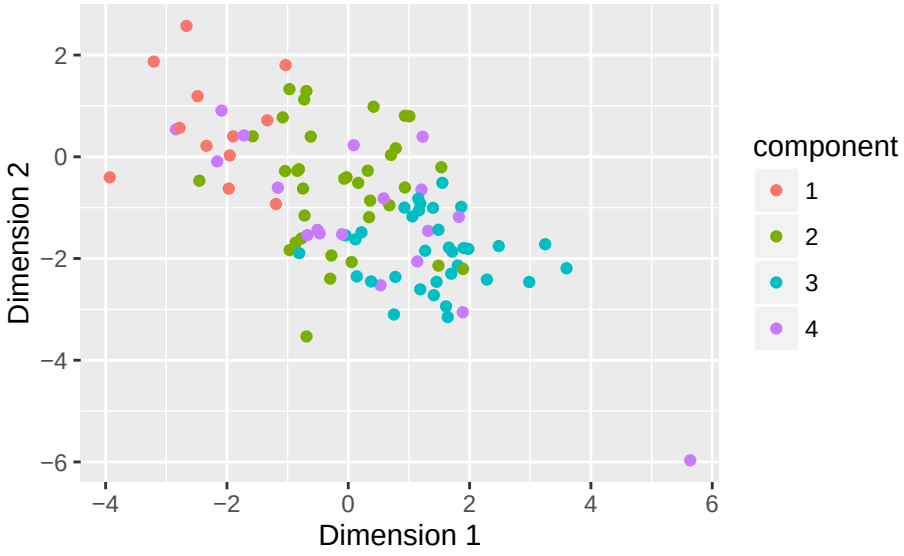
```
# Specify sparsity in [0,1], indicating fraction of off-diagonal zero entries.
s = 0.9


# Generate networks with random means and covariances. Means will be drawn from
# a standard Gaussian distribution, non-zero covariance values from a
# Beta(1,1) distribution.
sim.result = sim_mix_networks(n, p, n.comp, s, mix.prob)
```

The data is contained in `sim.result$data`, and the components that each data point belongs to are contained in `sim.result$comp`. Let's check that the mixture probabilities are correct and then plot the first two dimensions of the data. Note that we do not expect these to be well-separated in any way.

```
print(table(sim.result$comp)/n)
```

```
##
##    1    2    3    4
## 0.12 0.35 0.34 0.19
```

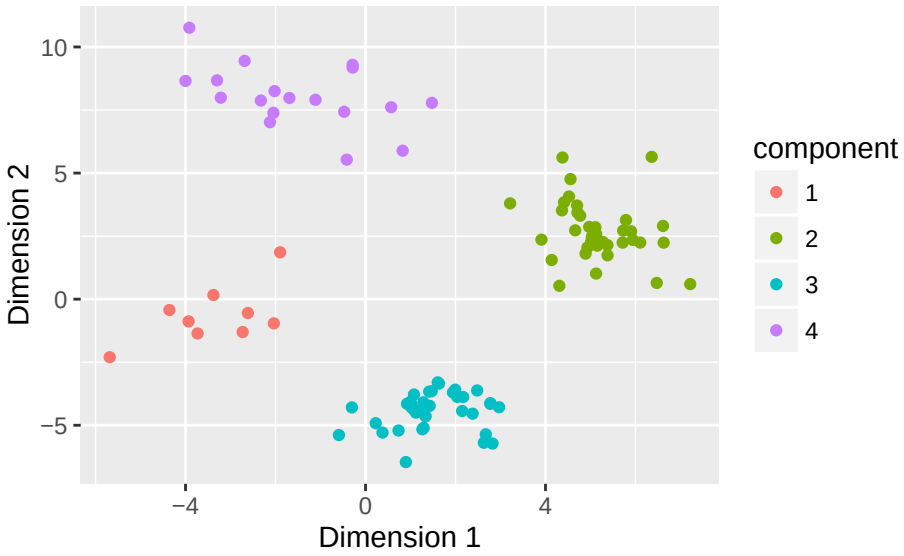
```
component = as.factor(sim.result$comp)

library('ggplot2')
qplot(x=sim.result$data[,1], y=sim.result$data[,2],
      colour=component) +
  xlab('Dimension 1') +
  ylab('Dimension 2')
```




The means and covariances of the data are contained in `sim.result$Mu` and `sim.result$Sig`. If desired, they can also be specified when calling `sim_mix_networks`.

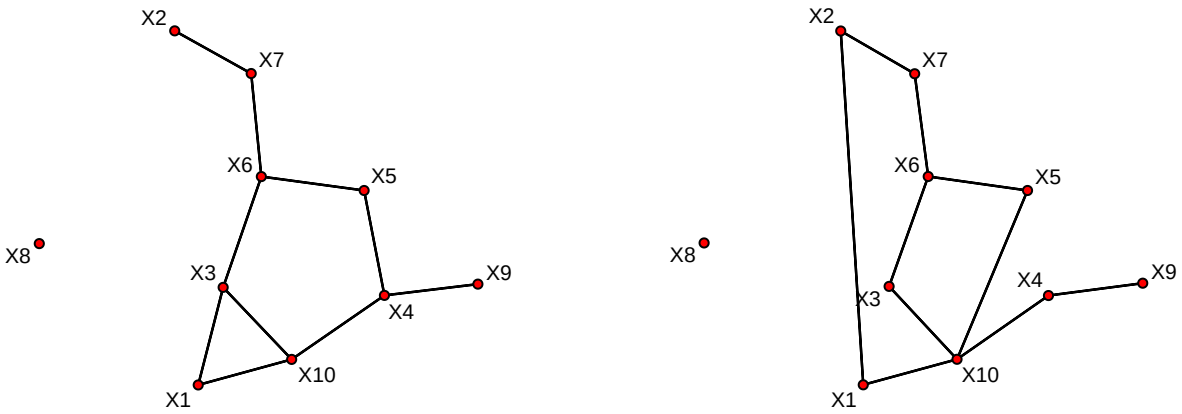
```
# Generate new dataset with the same covariances, but different means
sim.result.new = sim_mix_networks(n, p, n.comp, s, mix.prob, Sig=sim.result$Sig)

component = as.factor(sim.result.new$comp)

qplot(x=sim.result.new$data[,1], y=sim.result.new$data[,2],
      colour=component) +
```

```
  xlab('Dimension 1') +
  ylab('Dimension 2')
```




When the covariance matrices for the components are not specified in advance, the `sim_mix_networks` function implicitly assumes that they are generated independently of each other. In order to test the `diffnet` functions, we also want to be able to generate simulated data from pairs of networks that present some common edges. The `generate_2networks` function is used to generate pairs of networks with an arbitrary overlap.

```
## Sample size and number of nodes
n <- 40
p <- 10

## Specify sparse inverse covariance matrices,
## with number of edges in common equal to ~ 0.8*p
gen.net <- generate_2networks(p,graph='random',n.nz=rep(p,2),
                              n.nz.common=ceiling(p*0.8))

invcov1 <- gen.net[[1]]
invcov2 <- gen.net[[2]]

plot_2networks(invcov1,invcov2,label.pos=0,label.cex=0.7)
```




## 5 Network estimation with known group labels

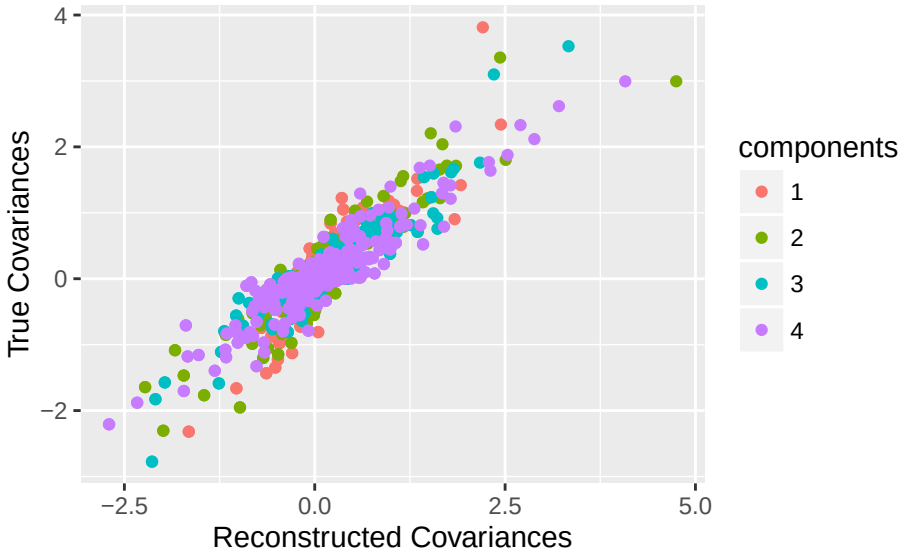
If it is known a priori to which component each sample belongs, then the problem of reconstructing the network reduces to a simple application of the graphical Lasso to each component. For convenience, we have included a wrapper function `het_cv_glasso` in `nethet` that applies the graphical Lasso [**?**] to each component in a heterogeneous dataset with specified component labels. The penalisation hyperparameter is tuned individually for each component using cross-validation.

To demonstrate `het_cv_glasso`, we will generate some data in the same way as in the previous section:

```
n = 100
p = 25

# Generate networks with random means and covariances.
sim.result = sim_mix_networks(n, p, n.comp, s, mix.prob)

test.data = sim.result$data
test.labels = sim.result$comp

# Reconstruct networks for each component
networks = het_cv_glasso(data=test.data, grouping=test.labels)
```

One way of checking if the reconstructed networks are sensible is plotting the covariance matrices used for generating the networks against the reconstructed covariance matrices.

```
# Component labels for covariance values
components = as.factor(rep(1:n.comp, each=p^2))

qplot(x=c(networks$Sig), y=c(sim.result$Sig),
                      colour=components) +
  xlab('Reconstructed Covariances') +
  ylab('True Covariances')
```




# 6 High-dimensional two-sample testing

We have demonstrated how to use our package to estimate networks from heterogeneous data. Often, we would like to perform a statistical comparison between networks. *Differential network* allows formal hypothesis testing regarding network differences. It is based on a novel and very general methodology for high-dimensional two-sample testing. Other useful tools based on this technology are *GGM-GSA* ("multivariate gene-set testing based on GGMs") and *differential regression* which allows formal two-sample testing in the high-dimensional regression model. For details on this methodology we refer the reader to [**?**, **?**].

## 6.1 Differential network

Let us consider datasets generated from GGMs $G_1$ and $G_2$ respectively. We would like to know whether networks inferred from these datasets differ in a statistical significant manner, that is we would like to test the hypothesis

$$\mathbf{H}_0: \quad G_1 = G_2.$$

The function `diffnet_multisplit` uses repeated sample splitting to address this task. The main steps are:

1. Both datasets are randomly split into two halves: the "*in*-" and "*out*-sample".
2. Networks are inferred using only the *in*-sample ("screening step").
3. Based on the *out*-sample, a p-value is computed which compares the networks obtained in step 2 ("cleaning step").
4. Steps 1-3 are repeated many times (e.g. 50 times); the resulting p-values are aggregated and the final aggregated p-value is reported.

We now illustrate the use of `diffnet_multisplit` with an example. We consider GGMs (i.e. inverse covariance matrices) previously generated in Section **??**.

```
## Set seed
set.seed(1)

## Sample size and number of nodes
p <- 30

## Specify sparse inverse covariance matrices
```

```
gen.net <- generate_2networks(p,graph='random',n.nz=rep(p,2),
                              n.nz.common=ceiling(p*0.8))
invcov1 <- gen.net[[1]]
invcov2 <- gen.net[[2]]

## Get corresponding correlation matrices
cor1 <- cov2cor(solve(invcov1))
cor2 <- cov2cor(solve(invcov2))
```

We start with generating data under the "null-scenario" where both datasets have the same underlying network.

```
## Generate data under null hypothesis
library(mvtnorm) # To generate multivariate Gaussian random samples

## Sample size
n <- 70

x1 <- rmvnorm(n,mean = rep(0,dim(cor1)[1]), sigma = cor1)
x2 <- rmvnorm(n,mean = rep(0,dim(cor1)[1]), sigma = cor1)
```

Then, we run a differential network analysis:

```
## Run diffnet (under null hypothesis)
dn.null <- diffnet_multisplit(x1,x2,b.splits=1,verbose=FALSE)
```

We obtain the p-value 0.8505236, which is stored in `dn.null$ms.pval`.

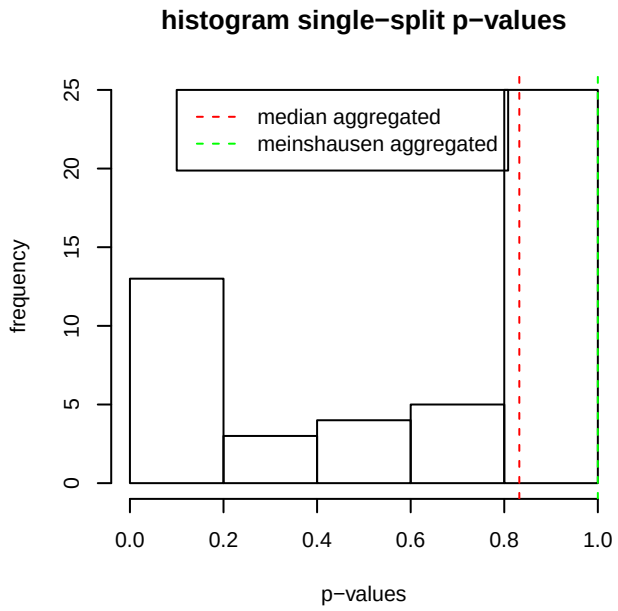
The same analysis can be performed for data generated under the alternative hypothesis.

```
## Generate data under alternative hypothesis (datasets have different networks)
x1 <- rmvnorm(n,mean = rep(0,dim(cor1)[1]), sigma = cor1)
x2 <- rmvnorm(n,mean = rep(0,dim(cor1)[2]), sigma = cor2)

## Run diffnet (under alternative)
dn.altn <- diffnet_multisplit(x1,x2,b.splits=1,verbose=FALSE)
```

The resulting p-value is 0.715561 which indicates a highly significant network difference.

The variable `b.splits` specifies the number of data splits used in the differential network procedure. The p-values in the previous examples were obtained using only a single data split (`b.splits=1`). P-values heavily depend on the random split of the data. This amounts to a "p-value lottery". To get stable and reproducible results we therefore would typically choose a larger number for the variable `b.split` and report the aggregated p-value.

```
## Typically we would choose a larger number of splits
# Use parallel library (only available under Unix) for computational efficiency
if(.Platform$OS.type == "unix") {
        dn.altn <- diffnet_multisplit(x1,x2,b.splits=50,verbose=FALSE,mc.flag=TRUE)
} else {
  dn.altn <- diffnet_multisplit(x1,x2,b.splits=25,verbose=FALSE,mc.flag=FALSE)
}

par(cex=0.7)
plot(dn.altn, cex=0.5) # histogram over 50 p-values
```



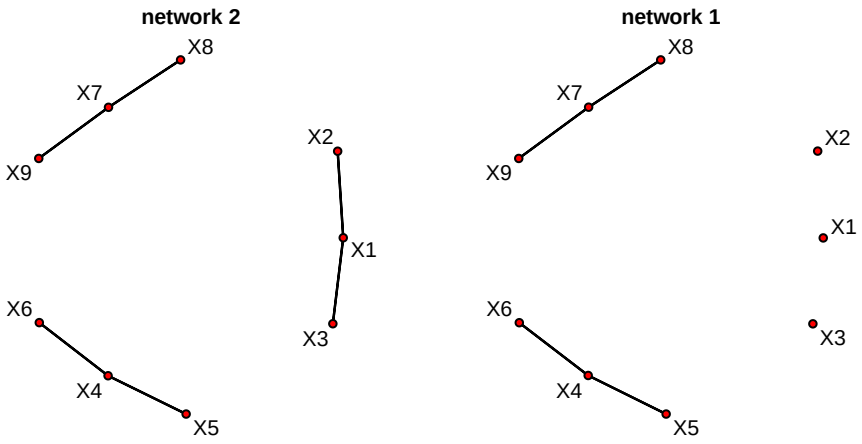

```
cat('p-value:',dn.altn$medagg.pval,'\n') # median aggregated p-value
```

```
## p-value: 0.8322191
```

## 6.2 Multivariate gene-set testing based on GGMs

In the case where molecular variables can be grouped into various sets of biologically related features (e.g. gene-sets or pathways), `ggmgsa_multisplit` can be used to perform differential network analyses iteratively for all gene-sets. This allows us to identify gene-sets which show a significant network difference. For illustration we consider data generated from the following networks.

```
## Generate new networks
set.seed(1)
p <- 9 # network with p nodes
n <- 40
hub.net <- generate_2networks(p,graph='hub',n.hub=3,n.hub.diff=1)#generate hub networks
invcov1 <- hub.net[[1]]
invcov2 <- hub.net[[2]]
plot_2networks(invcov1,invcov2,label.pos=0,label.cex=0.7,
               main=c('network 1', 'network 2'),cex.main=0.7)
```




```
## Generate data
library('mvtnorm')
x1 <- rmvnorm(n,mean = rep(0,p), sigma = cov2cor(solve(invcov1)))
x2 <- rmvnorm(n,mean = rep(0,p), sigma = cov2cor(solve(invcov2)))
```

The nodes can be grouped into three gene-sets where only the first has a different underlying network.

```
## Identify groups with 'gene-sets'
gene.names <- paste('G',1:p,sep='')
gsets <- split(gene.names,rep(1:3,each=3))
```

We run GGM-GSA with a single data split (`b.splits=1`) and note that only the p-value for the first gene-set has small magnitude. Again, we would typically use a larger number of data splits in order to obtain stable p-values.

```
## Run GGM-GSA
fit.ggmgsa <- ggmgsa_multisplit(x1,x2,b.splits=1,gsets,gene.names,verbose=FALSE)

library(xtable)
print(xtable(summary(fit.ggmgsa),digits=6))
```

| | medagg.pval | meinshagg.pval |
|---|---|---|
| gs1 | 0.001004 | 0.004010 |
| gs2 | 0.103492 | 0.413552 |
| gs3 | 0.601607 | 1.000000 |

## 6.3 Differential regression

In addition to differential network, this **R**-package also provides an implementation of differential regression. In particular, the function `diffregr_multisplit` allows formal two-sample testing in the high-dimensional regression model. It is also based on sample splitting and is very similar to the previously introduced `diffnet_multisplit`.

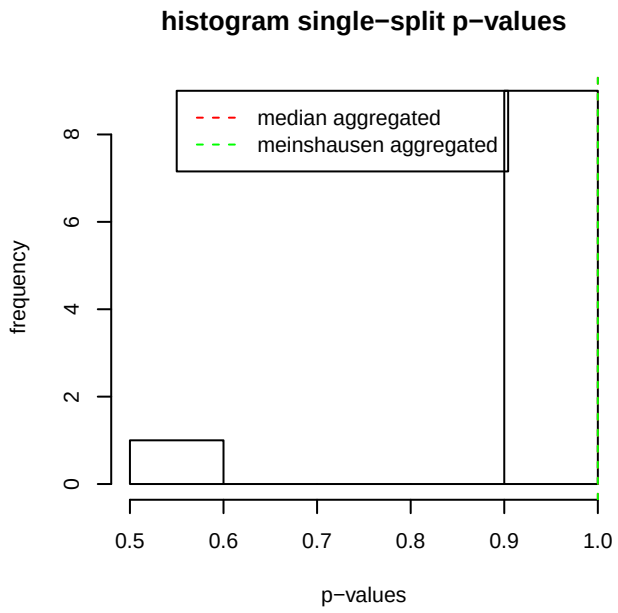
Consider the following sparse regression models.

```
## Number of predictors and sample size
p <- 100
n <- 80

## Predictor matrices
x1 <- matrix(rnorm(n*p),n,p)
```

```
x2 <- matrix(rnorm(n*p),n,p)

## Active-sets and regression coefficients
act1 <- sample(1:p,5)
act2 <- c(act1[1:3],sample(setdiff(1:p,act1),2))
beta1 <- beta2 <- rep(0,p)
beta1[act1] <- 0.7
beta2[act2] <- 0.7
```

We generate data under the null-hypothesis and run differential regression. The histogram shows the distribution of the p-values obtained form ten data splits.

```
## Response vectors under null-hypothesis
y1 <- x1%*%as.matrix(beta1)+rnorm(n,sd=1)
y2 <- x2%*%as.matrix(beta1)+rnorm(n,sd=1)

## Differential regression; b.splits=10
fit.null <- diffregr_multisplit(y1,y2,x1,x2,b.splits=10)
par(cex=0.7)
plot(fit.null,cex=0.5) # histogram of p-values from b.split data splits
```



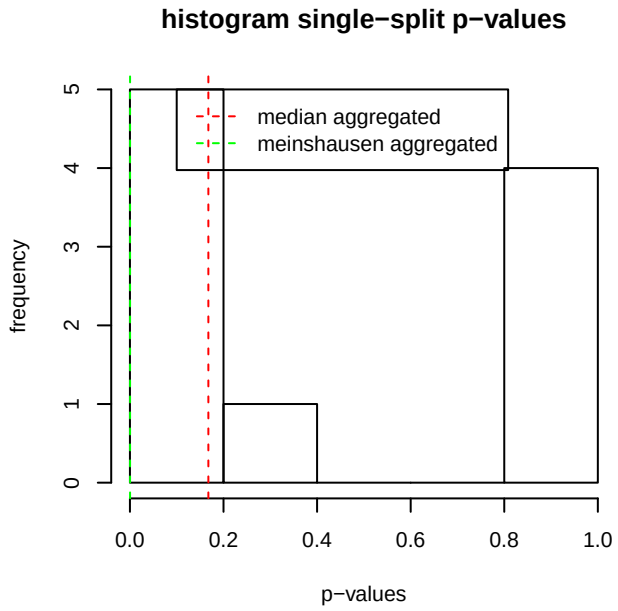

```
cat('p-value: ',fit.null$medagg.pval,'\n') # median aggregated p-value
```

p-value: 1

The following example illustrates differential regression in scenario with different regression models.

```
## Response vectors under alternative-hypothesis
y1 <- x1%*%as.matrix(beta1)+rnorm(n,sd=1)
y2 <- x2%*%as.matrix(beta2)+rnorm(n,sd=1)

## Differential regression (asymptotic p-values)
fit.alt <- diffregr_multisplit(y1,y2,x1,x2,b.splits=10)
par(cex=0.7)
```

```
plot(fit.alt)
```




```
cat('p-value: ',fit.alt$medagg.pval,'\n')
```

p-value: 0.1676274

For differential regression we have the option to compute permutation-based p-values by choosing a number of permutations `n.perm`.

```
## Differential regression (permutation-based p-values; 100 permutations)
fit.alt.perm <- diffregr_multisplit(y1,y2,x1,x2,b.splits=5,n.perm=100)
```

The default option (`n.perm=NULL`) uses an asymptotic approximation to calculate p-values.

# 7 Network estimation and model-based clustering with unknown group labels

Often we do not know a priori which component each sample belongs to. For example in the case of samples corresponding to protein measurements in breast cancer patients, the particular subtype of breast cancer that a patient suffers from may be unknown. In these cases, our package allows for network-based clustering of the samples using the mixture graphical Lasso (mixglasso), which jointly clusters the samples and reconstructs the networks for each group or cluster.

To demonstrate the `mixglasso` function, let us first generate some data in the same way as before, but with means defined to ensure separability of the groups:

```
# Generate networks with random means and covariances.
n = 1000
p = 10
s = 0.9
n.comp = 3

# Create different mean vectors
```

```
Mu = matrix(0,p,n.comp)

# Define non-zero means in each group (non-overlapping)
nonzero.mean = split(sample(1:p),rep(1:n.comp,length=p))

# Set non-zero means to fixed value
for(k in 1:n.comp){
  Mu[nonzero.mean[[k]],k] = -2/sqrt(ceiling(p/n.comp))
}

# Generate data
sim.result = sim_mix_networks(n, p, n.comp, s, Mu=Mu)
```

Now we will run mixglasso on this dataset to retrieve the original clustering and reconstruct the underlying networks.

```
# Run mixglasso
mixglasso.result = mixglasso(sim.result$data, n.comp=3)

# Calculate adjusted rand index to judge how accurate the clustering is
# Values > 0.7 indicate good agreement.
library(mclust, quietly=TRUE)

## Package 'mclust' version 5.2

## Type 'citation("mclust")' for citing this R package in publications.

adj.rand = adjustedRandIndex(mixglasso.result$comp, sim.result$comp)
cat('Adjusted Rand Index', round(adj.rand, digits=2), '\n')

## Adjusted Rand Index 0.63
```

Table **??** shows the cross-tabulation of the number of samples in predicted versus true groups.

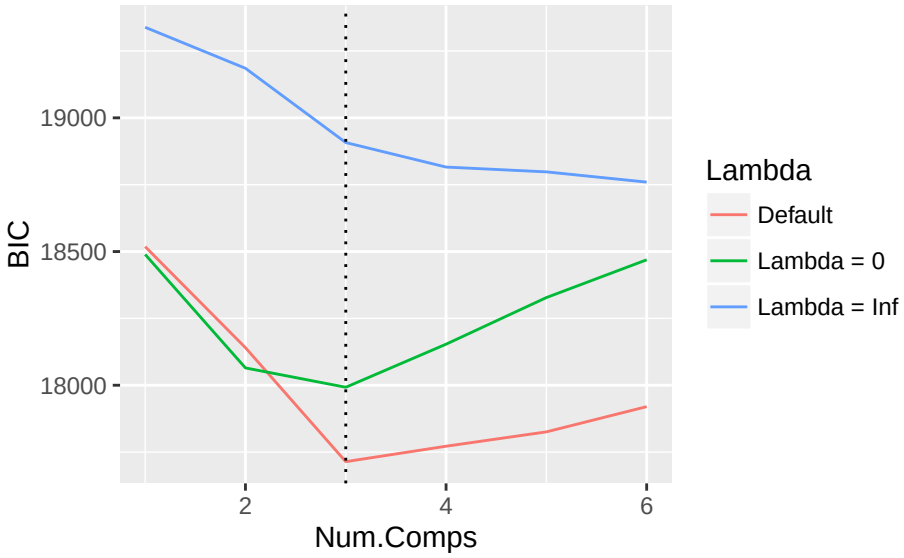
| | A | B | C |
|---|---|---|---|
| 1 | 316 | 12 | 16 |
| 2 | 23 | 37 | 276 |
| 3 | 15 | 267 | 38 |

Table 1: Cross-tabulation of mixglasso clusters (rows) with true group assignments (columns).

What if we don't know the true number groups? Luckily, `mixglasso` supports model comparison using BIC [**?**] and minimum description length [**?**]. In the following example we will use BIC to find the correct number of components:

```
# Run mixglasso over a range of numbers of components
mixglasso.result = mixglasso(sim.result$data, n.comp=1:6)

##          -mixglasso: comp too small; min(n_k)= 4.999443
##          -mixglasso: comp too small; min(n_k)= 4.629485

# Repeat with lambda=0 and lambda=Inf for comparison
mixglasso.result.0 = mixglasso(sim.result$data, n.comp=1:6, lambda=0)
mixglasso.result.Inf = mixglasso(sim.result$data, n.comp=1:6, lambda=Inf)

# Aggregate BIC results for plotting
BIC.vals = c(mixglasso.result$bic, mixglasso.result.0$bic,
                                              mixglasso.result.Inf$bic)

lambda.labels = rep(c('Default', 'Lambda = 0', 'Lambda = Inf'), each=6)
```

```
# Plot to verify that minimum BIC value corresponds with true
library(ggplot2)
plotting.frame <- data.frame(BIC=BIC.vals, Num.Comps=rep(1:6, 3), Lambda=lambda.labels)

p <- ggplot(plotting.frame) +
        geom_line(aes(x=Num.Comps, y=BIC, colour=Lambda)) +
        geom_vline(xintercept=3, linetype='dotted')

print(p)
```




We note that mixglasso involves a penalization parameter $\lambda$ which trades off goodness-of-fit and model complexity. We recommend to use the default which employs an adaptive and automatic penalization scheme [**?**]. Note that in this simplified example, $\lambda = 0$ (no penalization) performs well because $n >> p$. $\lambda = \infty$ constrains inverse covariance matrices to be diagonal, hence the inferior performance.

# 8 Plotting and exporting results

Our package includes several functions for plotting and exporting the networks and results that have been obtained.

## 8.1 Plotting results

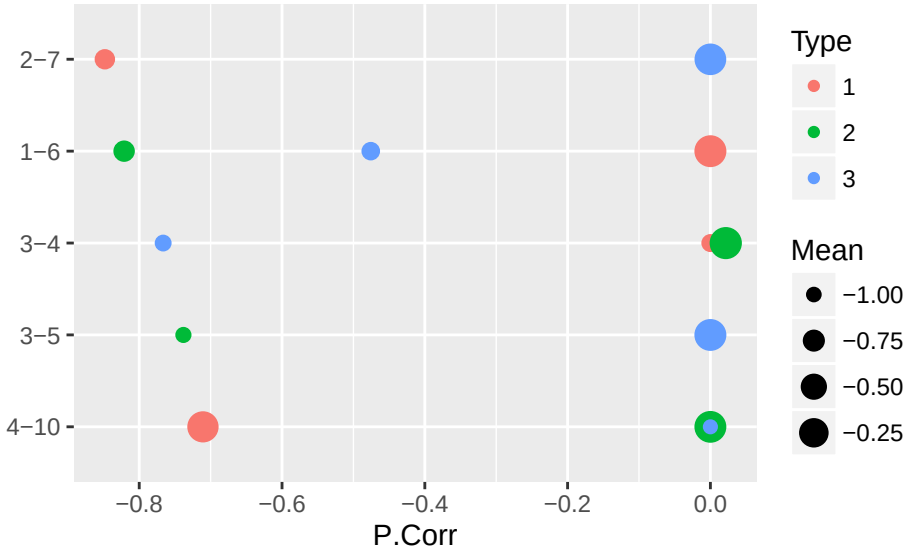
The output of `het_cv_glmnet` and `mixglasso` can be plotted either in network form or as individual edges in the networks. For the network plots, we use the `network` package [**?**]. This is the default plotting when `plot` is invoked on an object of class `nethetclustering`, and produces one global plot showing edges that occur in any group, as well as one plot for each group. For this example we will use the networks and clustering obtained using `mixglasso` in the previous section.

```
# Retrieve best clustering and networks by BIC
mixglasso.clustering = mixglasso.result$models[[mixglasso.result$bic.opt]]

# Plot networks, omitting edges with absolute partial correlation < 0.5 in
# every group.
```

```
# NOTE: Not displayed.
# plot(mixglasso.clustering, p.corrs.thresh=0.5)
```

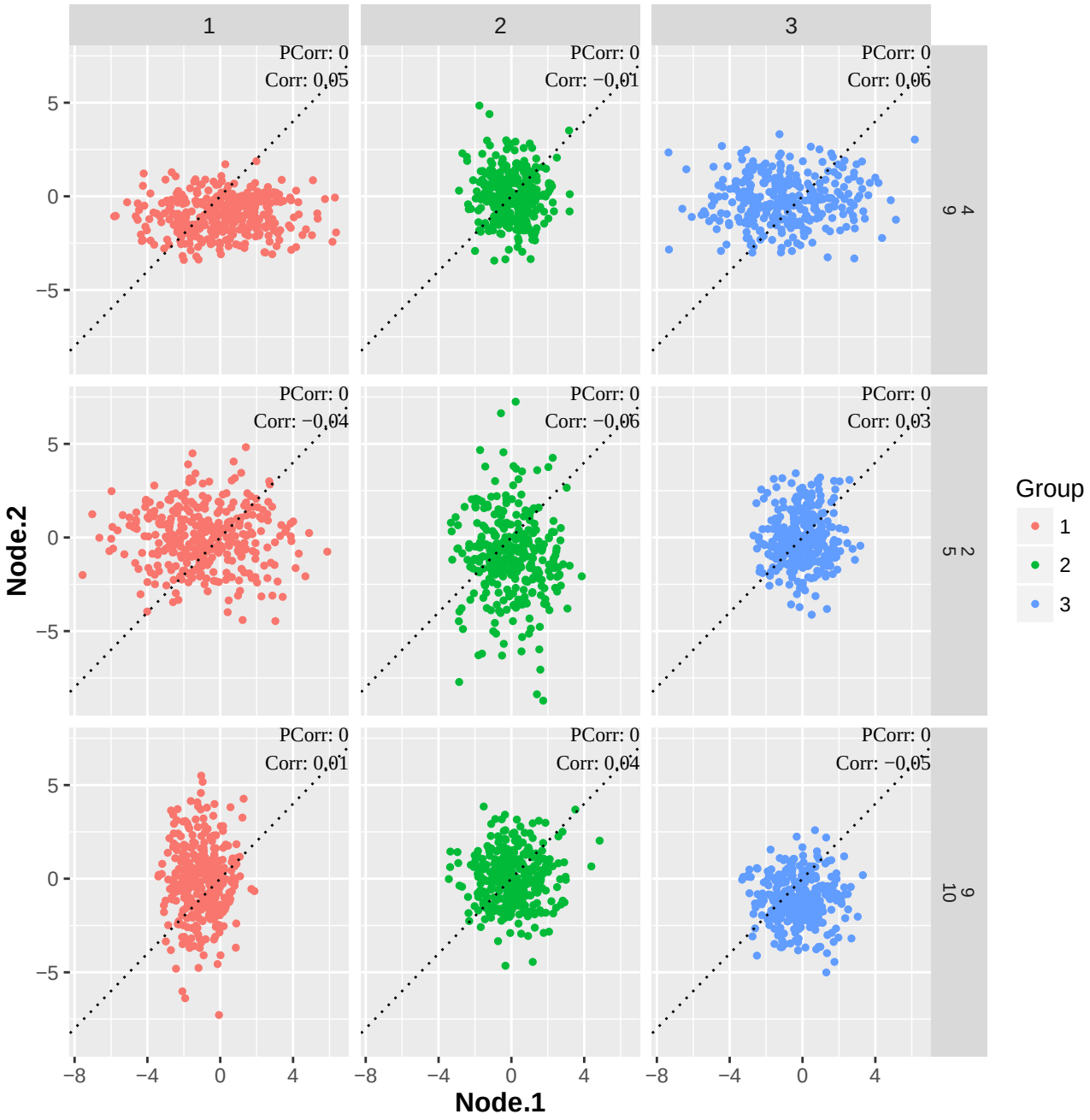
Usually we are only interested in specific edges, and perhaps we wish to compare them among groups. Function `dot_plot` generates a plot with edges above a certain threshold along the y-axis, and one circle for each group showing the smallest mean of the two nodes that make up the edge. We use the `ggplot2` package to make the plots [**?**].

```
# Plot edges, omitting those with absolute partial correlation < 0.5 in every
# group.
g = dot_plot(mixglasso.clustering, p.corrs.thresh=0.5, dot.size.range=c(1,5))
```




Finally, we might want to compare the observed values of the nodes linked by specific edges across groups. Function `scatter_plot` will generate plots for a specified list of edges.

```
# Specify edges
node.pairs = rbind(c(9,10), c(2,5),c(4,9))

# Create scatter plots of specified edges
g = scatter_plot(mixglasso.clustering, data=sim.result$data,
                                              node.pairs=node.pairs, cex=0.5)
```




## 8.2 Exporting Results

Our package offers the option to export the inferred networks as a comma-separated values (CSV) text file. Like the plotting functions, function `export_network` can be invoked on the output of `het_cv_glmnet` and `mixglasso`.

```
# Save network in CSV format, omitting edges with absolute partial correlation
# less than 0.25.
#export_network(mixglasso.clustering, file='nethet_network.csv',
#  p.corrs.thresh=0.25)
```

This creates a CSV file encoding a table with one row for each edge with partial correlation above the threshold, and

columns indicating the nodes linked by the edge, the absolute partial correlation, the sign of the partial correlation, and the group or cluster in which the edge occurred.

If the user wishes to use the Cytoscape [**?**] software to analyse the network further, we note that the output of `export_network` can be loaded into Cytoscape, provided the option `quote=FALSE` is set.

```
# Save network in CSV format suitable for Cytoscape import
#export_network(mixglasso.clustering, file='nethet_network.csv',
#  p.corrs.thresh=0.25, quote=FALSE)
```

```
sessionInfo()
```

```
## R version 3.3.0 (2016-05-03)
## Platform: x86_64-apple-darwin13.4.0 (64-bit)
## Running under: OS X 10.9.5 (Mavericks)
##
## locale:
## [1] C/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
## [1] mclust_5.2    xtable_1.8-2  mvtnorm_1.0-5 ggplot2_2.1.0 nethet_1.4.2
##
## loaded via a namespace (and not attached):
##  [1] ICSNP_1.1-0         Rcpp_0.12.5         highr_0.6           parcor_0.2-6
##  [5] formatR_1.4         plyr_1.8.3          ppls_1.6-1          iterators_1.0.8
##  [9] tools_3.3.0         digest_0.6.9        GeneNet_1.2.13      evaluate_0.9
## [13] gtable_0.2.0        lattice_0.20-33     huge_1.2.7          Matrix_1.2-6
## [17] foreach_1.4.3       igraph_1.0.1        cmprsk_2.2-7        parallel_3.3.0
## [21] CompQuadForm_1.4.1  stringr_1.0.0       knitr_1.13          ICS_1.2-5
## [25] stats4_3.3.0        multtest_2.28.0     grid_3.3.0          glmnet_2.0-5
## [29] Biobase_2.32.0      Epi_2.0             survival_2.39-4     longitudinal_1.1.12
## [33] fdrtool_1.2.15      etm_0.6-2           glasso_1.8          limma_3.28.4
## [37] GSA_1.03            reshape2_1.4.1      corpcor_1.6.8       magrittr_1.5
## [41] splines_3.3.0       BiocGenerics_0.18.0 scales_0.4.0        codetools_0.2-14
## [45] MASS_7.3-45         BiocStyle_2.0.2     colorspace_1.2-6    labeling_0.3
## [49] stringi_1.0-1       survey_3.30-3       network_1.13.0      munsell_0.4.3
## [53] ggm_2.3
```