Simulating molecular regulatory networks using qpgraph

Inma Tur[1,3], Alberto Roverato[2] and Robert Castelo[1]

March 15, 2016

1. Universitat Pompeu Fabra, Barcelona, Spain.
2. Università di Bologna, Bologna, Italy.
3. Now at Kernel Analytics, Barcelona, Spain.

1 Introduction

The theoretical substrate used by *qpgraph* to estimate network models of molecular regulatory interactions is that of graphical Markov models (GMMs). A useful way to understand the underpinnings of these models is to simulate them and simulate data from them. More importantly, these simulated data may serve the purpose of verifying properties of GMM estimation procedures, such as correctness or asymptotic behavior. Here we illustrate the functionalities of *qpgraph* to perform these simulations. If you use them in your own research, please cite the following article:

> Tur, I., Roverato, A. and Castelo, R. Mapping eQTL networks with mixed graphical Markov models. *Genetics*, 198(4):1377-1393, 2014.

The interface provided by *qpgraph* tries to comply with the available functions in the base R *stats* package for simulating data from probability distributions and the names of functions described below for the purpose of simulating graphs, models and data follow the convention:

> r<$objectclass$>(n, ...)

where <$objectclass$> refers to the class of object (in a broad sense, not just a formal S3 or S4 class) being simulated and n is the number of observations to simulate. Except for the case of data, since the simulated observations are other than R atomic types of objects, when $n > 1$, these functions return simulated observations in the form of a list with n elements.

2 Simulation of graphs

An undirected graph G is a mathematical object defined by a pair of sets $G = (V, E)$ where $V = \{1, \ldots, p\}$ is the vertex set and $E \subseteq (V \times V)$ is the edge set. In the context of GMMs *labeled* undirected graphs are employed to represent conditional (in)dependences among random variables (r.v.'s) $X = \{X_1, \ldots, X_p\}$ indexed by the vertices in V whose values occur on equal footing. Stepwise data generating processes can be represented by directed graphs. In the context of GMMs one may also consider so-called *marked* graphs, which are graphs with *marked* vertices grouped into two subsets, one associated to discrete variables and another to continuous ones. A graph with a single type of vertices, i.e., that is not marked, it is also called *pure*. Diferent types of graphs determine different GMM classes. For a comprehensive description of different GMM classes and more elaborate descriptions of the terminology and notation used in this vignette the reader may consult the book of **?**.

The first step to simulate a GMM consists of simulating its associated graph. While there are many R packages that provide procedures to simulate graphs, *qpgraph* provides its own minimal functionality for this purpose tailored to ease the downstream simulation of GMMs. This functionality allows the user to simulate undirected graphs according to two main criteria, the type of graph (pure or marked) and the type of model to simulate the random graph.

The simplest type of model to simulate random undirected graphs is the so-called Erdős-Rényi which is generated by either including an edge between every pair of vertices with a pre-specified probability or drawing a graph uniformly at random among those with a pre-specified number of edges. In the context of exploring the performance of GMM structure estimation procedures under different degrees of sparseness of the underlying graph, it is handy to work with the so-called d-regular graphs (**?**). These are graphs with a constant degree vertex d which, in one hand, make the graph density a linear function of d and, on the other hand, bound the smallest minimal separator between any two vertices (**?**, see pg. 2646).

To specify the parameters that define one specific type of graph we want to simulate *qpgraph* provides the following functions that build parameter objects which can be used afterwards to simulate graphs through a single call to the function `rgraphBAM()`:

```
> library(qpgraph)
> args(erGraphParam)
```

```
function (p = 4L, m = 4L, prob = NA_real_, labels = as.character(1:p))
NULL
```

```
> args(erMarkedGraphParam)
```

```
function (pI = 1L, pY = 3L, m = 4L, prob = NA_real_, Ilabels = paste0("I",
    1:pI), Ylabels = paste0("Y", 1:pY))
NULL
```

```
> args(dRegularGraphParam)
```

```
function (p = 4L, d = 2L, exclude = as.integer(NULL), labels = as.character(1:p))
NULL
```

```
> args(dRegularMarkedGraphParam)
```

```
function (pI = 1L, pY = 3L, d = 2L, exclude = as.integer(NULL),
    Ilabels = paste0("I", 1:pI), Ylabels = paste0("Y", 1:pY))
NULL
```

As we can see from their default values, a single call without arguments already define some small graphs on 5 vertices:

```
> erGraphParam()
```

```
  Erdos-Renyi pure graph parameter object
  No. of pure vertices: 4
  No. of edges: 4
  Vertex labels: 1, 2, 3, 4
```

```
> erMarkedGraphParam()
```

```
  Erdos-Renyi marked graph parameter object
  No. of marked vertices: 4
  No. of dot (I) vertices: 1
  No. of circle (Y) vertices: 3
  No. of edges: 4
  Dot (I) vertex labels: I1
  Circle (Y) vertex labels: Y1, Y2, Y3
```

```
> dRegularGraphParam()
```

```
  d-regular pure graph parameter object
  No. of pure vertices: 4
  Constant degree: 2
  Vertex labels: 1, 2, 3, 4
```

```
> dRegularMarkedGraphParam()
```

```
  d-regular marked graph parameter object
  No. of marked vertices: 4
  No. of dot (I) vertices: 1
  No. of circle (Y) vertices: 3
  Constant degree: 2
  Dot (I) vertex labels: I1
  Circle (Y) vertex labels: Y1, Y2, Y3
```

The objects returned by these functions belong to different S4 classes derived from the following two main ones, *graphParam* and *markedGraphParam*:

```
> showClass("graphParam")
```

```
Class "graphParam" [package "qpgraph"]

Slots:

Name:          p    labels
Class:   integer character

Known Subclasses: "erGraphParam", "dRegularGraphParam"
```

```
> showClass("markedGraphParam")
```

```
Class "markedGraphParam" [package "qpgraph"]

Slots:

Name:         pI        pY   Ilabels   Ylabels
Class:   integer   integer character character

Known Subclasses: "erMarkedGraphParam", "dRegularMarkedGraphParam"
```

While this level of detail is not crucial for the end-user, knowing the distinction between these two main types of graph parameter objects, *graphParam* and *markedGraphParam*, may help to get more quickly acquainted with the type of arguments we need in calls described below to simulate GMMs.

Finally, the function `rgraphBAM()` simulates one or more random graphs as objects of the class *graphBAM* defined in the *graph* package. Its arguments are:

```
> args(rgraphBAM)
```

```
function (n, param, ...)
NULL
```

where `n` is the number of graphs to simulate (default `n=1`) and `param` is an object generated by one of the previous parameter functions. A couple of minimal examples are:

```
> rgraphBAM(erGraphParam())
```

```
A graphBAM graph with undirected edges
Number of Nodes = 4
Number of Edges = 4
```

```
> rgraphBAM(n=2, dRegularGraphParam())
```

```
[[1]]
A graphBAM graph with undirected edges
Number of Nodes = 4
Number of Edges = 4
```

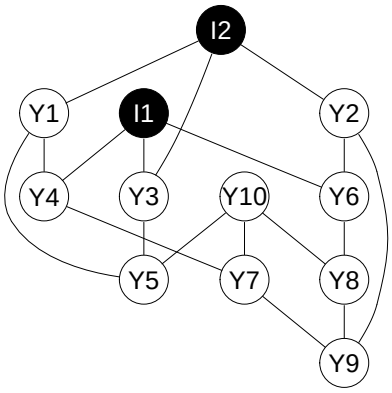

Figure 1: A random 3-regular marked undirected graph.

```
[[2]]
A graphBAM graph with undirected edges
Number of Nodes = 4
Number of Edges = 4
```

In a slightly more ellaborate example, if we would like to simulate a d-regular graph on 2 discrete vertices and 5 continuous ones with a constant degree $d = 3$, we would make the following call to `rgraphBAM()`, which we previously seed to enable the reader reproducing the same graph shown here:

```
> set.seed(1234)
> g <- rgraphBAM(dRegularMarkedGraphParam(pI=2, pY=10, d=3))
> plot(g)
```

where the last `plot` function call is defined (overloaded) in the *qpgraph* package to ease plotting the graph which is displayed in Figure **??**. This function uses the plotting capabilities from the *Rgraphviz* package and further arguments, such as `layoutType`, can be passed downstream to the *Rgraphviz* plotting function to fine tune the display of the graph.

3 Simulation of undirected Gaussian GMMs

Undirected Gaussian GMMs are multivariate normal models on continuous r.v.'s $X_V = \{X_1, \ldots, X_p\}$ determined by an undirected graph $G = (V, E)$ with $V = \{1, \ldots, p\}$ and $E \subseteq (V \times V)$. In particular, the zero-pattern of the inverse covariance matrix corresponds to the missing edges in G (**?**). Therefore, simulating this type of GMM amounts to simulate a covariance matrix whose inverse matches the missing edges of a given, or simulated, undirected graph in its zero pattern and whose scaled covariance matches a given marginal correlation on the cells corresponding to present edges. This can be easily accomplished with *qpgraph* using the function `rUGgmm`:

```
> args(rUGgmm)
```

```
function (n, g, ...)
NULL
```

where `n` corresponds to the number of undirected Gaussian GMMs we want to simulate (default `n=1` and `g` corresponds to either a *graphParam* object, a *graphBAM* object or a matrix. This depends on whether we want to simulate both the graph and the covariance matrix underlying the GMM, by providing a *graphParam* object, or we just want to simulate the covariance matrix given a graph specified as either a *graphBAM* object, an squared and symmetric adjacency matrix or a two-column matrix describing an edge set. Examples of these are the following:

```
> rUGgmm(dRegularGraphParam(p=4, d=2))
```

```
  Undirected Gaussian graphical Markov model
  with 4 r.v. and 4 edges.
```

```
> rUGgmm(matrix(c(0, 1, 0, 1,
+                 1, 0, 1, 0,
+                 0, 1, 0, 1,
+                 1, 0, 1, 0), nrow=4, byrow=TRUE))
```

```
  Undirected Gaussian graphical Markov model
  with 4 r.v. and 4 edges.
```

```
> rUGgmm(matrix(c(1, 2,
+                 2, 3,
+                 3, 4,
+                 4, 1), ncol=2, byrow=TRUE))
```

```
  Undirected Gaussian graphical Markov model
  with 4 r.v. and 4 edges.
```

These three calls to `rUGgmm()` return objects of class *UGgmm* containing undirected Gaussian GMMs with an underlying graph structure formed by a single undirected cycle on four vertices. The elements of an *UGgmm* object can be quickly explored with the `summary()` function call:

```
> set.seed(12345)
> gmm <- rUGgmm(dRegularGraphParam(p=4, d=2))
> summary(gmm)
```

```
  Undirected Gaussian graphical Markov model
  with 4 r.v. and 4 edges.

  Graph density: 67%

  Degree distribution of the undirected graph:
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
      2       2       2       2       2       2

  Distribution of marginal correlations for present edges:
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.3352  0.5963  0.7587  0.6958  0.8582  0.9305

  Distribution of partial correlations for present edges:
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
-0.8236 -0.6656 -0.5352 -0.4232 -0.2928  0.2014
```

and the individual elements that are available to the user can be accessed as if it were a *list* object:

```
> class(gmm)
```

```
[1] "UGgmm"
attr(,"package")
[1] "qpgraph"
```

```
> names(gmm)
```

```
[1] "X"     "p"     "g"     "mean"  "sigma"
```

```
> gmm$X
```

```
[1] "1" "2" "3" "4"
```

```
> gmm$p
```

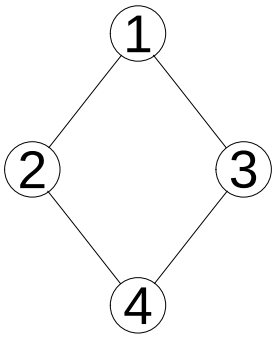

Figure 2: A 4-cycle undirected graph.

```
[1] 4
```

```
> gmm$g
```

```
A graphBAM graph with undirected edges
Number of Nodes = 4
Number of Edges = 4
```

```
> gmm$mean
```

```
[1] 0 0 0 0
```

```
> gmm$sigma
```

```
4 x 4 Matrix of class "dspMatrix"
          1         2         3         4
1 1.1854830 0.8687971 1.0001179 0.4717917
2 0.8687971 0.9152916 0.7026960 0.6299566
3 1.0001179 0.7026960 0.9745280 0.3188662
4 0.4717917 0.6299566 0.3188662 0.9285910
```

We can also directly plot the *UGgmm* object to see the underlying undirected graph and, in this particular example, note how the zeroes of the inverse covariance match the missing edges shown in Figure **??**.

```
> plot(gmm)
> round(solve(gmm$sigma), digits=1)
```

```
     1    2    3    4
1  9.5 -3.4 -7.2  0.0
2 -3.4  5.9  0.0 -2.3
3 -7.2  0.0  8.2  0.9
4  0.0 -2.3  0.9  2.3
```

Further arguments to `rUGgmm()` can be the desired mean marginal correlation derived from the cells of the covariance matrix that match the present edges in the underlying graph (`rho=0.5`), the minimum tolerance at which the iterative matrix completion algorithm that builds the covariance matrix stops (`tol=0.001`) and whether the function should report progress on the calculations (`verbose=FALSE`). It is important to set the latter argument `verbose=TRUE` when we want to simulate an undirected Gaussian GMM with more than, let's say, 200 vertices, since around that number of vertices and beyond the simulation of the covariance matrix becomes computationally demanding, specially when the underlying graph is not very sparse. Further technical information on the algorithms employed to simulate the covariance matrix can be found in the help pages of the *qpgraph* functions `qpG2Sigma()`, `qpRndWishart()`, `qpIPF()` and `qpHTF()` which are called by the procedures described here.

Finally, to simulate multivariate normal observations from the undirected Gaussian GMM we just need to use the `rmvnorm()` function from the *mvtnorm* package which is overloaded in the *qpgraph* package to take directly an *UGgmm* object, as follows:

```
> rmvnorm(10, gmm)
```

```
               1           2           3          4
 [1,]  1.8203501  0.93389886  1.432134453 -0.1917191
 [2,]  0.3276189 -0.04621885  0.004281742  0.8140006
 [3,]  1.2455715  0.92583029  1.462412069 -0.1502647
 [4,] -1.0649416 -1.43363508  0.025905355 -1.0810069
 [5,]  0.8215409  0.92693851  0.457389128  1.0147731
 [6,]  3.6015946  2.99690308  3.073395574  1.4028879
 [7,] -0.3178426  0.04528419 -0.970752819  1.4148364
 [8,] -0.9758867 -0.22685242 -1.549907669 -0.7448294
 [9,]  1.6800263  0.99249248  1.773751270 -0.6969221
[10,] -0.0824054  0.09107758 -0.560575698 -0.2894250
```

Note that with sufficient data we can directly recover the zero-pattern of the inverse covariance matrix:

```
> set.seed(123)
> X <- rmvnorm(10000, gmm)
> round(solve(cov(X)), digits=1)
```

```
   1    2    3    4
1  9.6 -3.4 -7.3  0.0
2 -3.4  5.9  0.0 -2.2
3 -7.3  0.0  8.1  0.9
4  0.0 -2.2  0.9  2.3
```

```
> round(solve(gmm$sigma), digits=1)
```

```
   1    2    3    4
1  9.5 -3.4 -7.2  0.0
2 -3.4  5.9  0.0 -2.3
3 -7.2  0.0  8.2  0.9
4  0.0 -2.3  0.9  2.3
```

However, such a sample size would be exceptional and for more limited sample size but still with $p < n$ the user may use the *qpgraph* function `qpPAC()` which performs zero-partial correlation tests and when $p \gg n$, then the user may estimate non-rejection rates **?** with the `qpNrr()` function and simplify the saturated model such that it may become possible to apply `qpPAC()`.

Obviously, gene expression data, either from microarrays or log-transformed count data, are far from being multivariate normal. However, many available methods for estimating molecular regulatory networks from expression data, such as *qpgraph*, make such an assumption and simulating data from undirected Gaussian GMMs can help us to test these methods under a controlled experiment, learning their basic properties and obvious pitfalls with such a clean data.

4 Simulation of homogeneous mixed GMMs

Mixed GMMs are GMMs for multivariate data defined by mixed discrete and continuous r.v.'s, $X = \{I, Y\}$ where $I = \{I_1, \ldots, I_{p_I}\}$ denote discrete r.v.'s and $Y = \{Y_1, \ldots, Y_{p_Y}\}$ denote continuous ones. This class of GMMs are determined by marked graphs $G = (V, E)$ with p marked vertices $V = \Delta \cup \Gamma$ where $\Delta = \{1, \ldots, p_I\}$ are plotted with dots and index the discrete r.v.'s in I and $\Gamma = \{1, \ldots, p_Y\}$ are denoted by circles and index the continuous r.v.'s in Y.

In the context of molecular regulatory networks and, particularly, of genetical genomics data where we associate discrete r.v.'s to genotypes and continuous ones to expression profiles, we make the assumption that discrete genotypes affect

gene expression and not the other way around. Under this assumption, we will consider the underlying graph G not only with mixed vertices but also with mixed edges, where some are directed and represented by arrows and some are undirected. More concretely G will be a partially-directed graph with arrows pointing from discrete vertices to continuous ones and undirected edges between continuous vertices. From this restricted definition of a partially-directed graph it follows inmediately that there are no semi-directed (direction preserving) cycles and allows one to interpret these GMMs also as *chain graphs*, which are graphs formed by undirected subgraphs connected by directed edges (**?**). Currently, the *graph* and *Rgraphviz* packages, in which *qpgraph* relies to handle and plot graph objects, do not directly allow one to define and work with partially-directed graphs. However, in the functionality described below *qpgraph* tries to hide to the user complications derived from this fact.

A second important assumption *qpgraph* makes is that the joint distribution of the r.v.'s in X is a conditional Gaussian distribution (also known as CG-distribution) by which continuous r.v.'s follow a multivariate normal distribution $\mathcal{N}_{|\Gamma|}(\mu(i), \Sigma(i))$ conditioned on the joint levels $i \in \mathcal{I}$ from the discrete variables in I.

A third and final assumption is that the conditional covariance matrix is constant across $i \in \mathcal{I}$, the joint levels of I, i.e., $\Sigma(i) \equiv \Sigma$. This implies that we are actually simulating the so-called *homogeneous* mixed GMMs. In the context of genetical genomics data, this assumption implies that genotype alleles affect only the mean expression level of genes and not the correlations between them.

Two restrictions currently constain further the type of mixed GMMs we can simulate with *qpgraph*. The first one is that discrete variables are simulated under marginal independence between them and the second one is that every continuous variable cannot be associated to more than one discrete variable. As we shall see below, the first restriction does not apply when simulating expression quantitative trait loci data in experimental crosses, as genotype marker data is simulated by another package, the *qtl* ackage (**?**). We are working to remove the second restriction in the near future and enable multiple discrete variables having linear additive effects and interaction effects, on a common continuous variable.

Similarly to how we did it with undirected Gaussian GMMs, simulating mixed GMMs is done with a call to the function `rHMgmm()`:

```
> args(rHMgmm)
```

```
function (n, g, ...)
NULL
```

where `n` corresponds to the number of mixed GMMs we want to simulate (default `n=1` and `g` corresponds to either a *markedGraphParam* object, a *graphBAM* object or a matrix. Note that the first assumption made before enables specifying the underlying partially-directed graph just as we did for an undirected one, since directed edges are completely determined by the vertex type at their endpoints (discrete to continuous). Examples of these are the following:

```
> rHMgmm(dRegularMarkedGraphParam(pI=1, pY=3, d=2))
```

```
  Homogeneous mixed graphical Markov model
  with 1 discrete and 3 continuous r.v., and 4 edges.
```

```
> rHMgmm(matrix(c(0, 1, 0, 1,
+                 1, 0, 1, 0,
+                 0, 1, 0, 1,
+                 1, 0, 1, 0), nrow=4, byrow=TRUE), I=1)
```

```
  Homogeneous mixed graphical Markov model
  with 1 discrete and 3 continuous r.v., and 4 edges.
```

```
> rHMgmm(matrix(c(1, 2,
+                 2, 3,
+                 3, 4,
+                 4, 1), ncol=2, byrow=TRUE), I=1)
```

```
  Homogeneous mixed graphical Markov model
  with 1 discrete and 3 continuous r.v., and 4 edges.
```

These three calls to `rHMgmm()` return objects of class *HMgmm* containing homogenous mixed GMMs with an underlying graph structured formed by one discrete vertex pointing to two continous ones which are themselves forming an undirected connected component with a fourth vertex, all together forming an undirected cycle on four vertices. Just as with *UGgmm* objects, the elements of an *HMgmm* object can be quickly explored with the `summary()` function call:

```
> set.seed(12345)
> gmm <- rHMgmm(dRegularMarkedGraphParam(pI=1, pY=3, d=2))
> summary(gmm)

  Homogeneous mixed graphical Markov model
  with 1 discrete and 3 continuous r.v., and 2 edges.

  Graph density: 33% (all edges) 33% (mixed edges) 67% (continuous edges)

  Degree distribution of the vertices in the graph:
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
      2       2       2       2       2       2

  Distribution of marginal correlations for present continuous edges:
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.7822  0.8063  0.8304  0.8304  0.8545  0.8786

  Distribution of partial correlations for present continuous edges:
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
-0.7536 -0.6937 -0.6339 -0.6339 -0.5740 -0.5142

  Distribution of additive linear effects for present mixed edges:
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
      1       1       1       1       1       1
```

and again the individual elements that are available to the user can be accessed as if it were a *list* object:

```
> class(gmm)

[1] "HMgmm"
attr(,"package")
[1] "qpgraph"

> names(gmm)

 [1] "X"     "I"     "Y"     "p"     "pI"    "pY"    "g"     "mean"  "sigma"
[10] "a"     "eta2"

> gmm$X

[1] "I1" "Y1" "Y2" "Y3"

> gmm$I

[1] "I1"

> gmm$Y

[1] "Y1" "Y2" "Y3"

> gmm$p

[1] 4

> gmm$pI

[1] 1
```

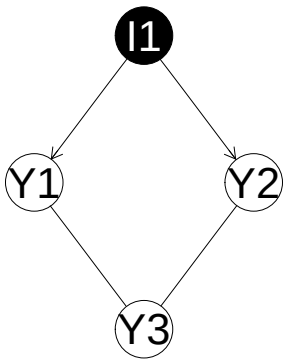

Figure 3: Homogeneous mixed graphical (chain) model with one discrete variable and three continuous ones forming an undirected cycle on four vertices.

```
> gmm$pY
```

```
[1] 3
```

```
> gmm$g
```

```
A graphBAM graph with undirected edges
Number of Nodes = 4
Number of Edges = 4
```

```
> gmm$mean()
```

```
         Y1        Y2        Y3
1 0.4720734 0.9669291 0.7242007
2 1.4720734 1.9669291 1.7934027
```

```
> gmm$sigma
```

```
3 x 3 Matrix of class "dspMatrix"
          Y1        Y2        Y3
Y1 0.3986118 0.6302970 0.4720734
Y2 0.6302970 2.1100639 0.9669291
Y3 0.4720734 0.9669291 0.7242007
```

```
> gmm$a
```

```
Y1 Y2 Y3
 1  1 NA
```

```
> gmm$eta2
```

```
        Y1         Y2         Y3
0.35494969 0.09638361         NA
```

We can also directly plot the *HMgmm* object and *qpgraph* will use the necessary instructions from the *graph* and *Rgraphviz* libraries to display a partially-directed graph as the one shown in Figure **??**.

```
> plot(gmm)
```

Further arguments to `rHMgmm()` are all we described previously for the `rUGgmm()` function plus the desired additive linear effect (`a=1`) of the discrete levels (alleles in the genetics context) on the continuous variables (genes in the genetics context). To simulate conditional multivariate normal observations from the homogeneous mixed GMM we use the `rcmvnorm()` function, which uses its pure continuous counterpart `rmvnorm()` from the *mvtnorm* package, but which is defined in the *qpgraph* package as it needs to calculate the corresponding conditional mean vectors $\mu(i)$:

```
> rcmvnorm(10, gmm)
```

```
   I1          Y1          Y2          Y3
1   1  0.69288009  0.78201846  0.84474686
2   1 -0.07511924 -2.25118979 -0.07484308
3   1  0.30898513  2.16129006  0.45379595
4   1  0.44373890  1.73438953  0.69000584
5   1  0.03144977  1.35426519  0.95480862
6   1 -0.34614962  0.13665093 -0.45059696
7   2  1.45888312  1.52702362  1.81217248
8   1  0.17087970  0.19890662  0.67766218
9   1  0.32224897  0.05164234  1.09247428
10  2  1.39579255  0.68223679  1.48814816
```

Note that with sufficient data we can directly recover the zero-pattern of the inverse *conditional* covariance matrix:

```
> set.seed(123)
> X <- rcmvnorm(10000, gmm)
> csigma <- (1/10000)*sum(X[, gmm$I] == 1)*cov(X[X[, gmm$I]==1, gmm$Y]) +
+           (1/10000)*sum(X[, gmm$I] == 2)*cov(X[X[, gmm$I]==2, gmm$Y])
> round(solve(csigma), digits=1)
```

```
     Y1   Y2   Y3
Y1 11.1  0.0 -7.2
Y2  0.0  1.2 -1.6
Y3 -7.2 -1.6  8.3
```

```
> round(solve(gmm$sigma), digits=1)
```

```
     Y1   Y2   Y3
Y1 11.0  0.0 -7.2
Y2  0.0  1.2 -1.6
Y3 -7.2 -1.6  8.2
```

and that the sample sample mean vectors and additive effects approach the ones specified in the model according to the mixed associations between the discrete and continuous variables:

```
> smean <- apply(X[, gmm$Y], 2, function(x, i) tapply(x, i, mean), X[, gmm$I])
> smean
```

```
         Y1       Y2        Y3
1 0.4637072 0.942311 0.7123249
2 1.4657407 1.985921 1.7951299
```

```
> gmm$mean()
```

```
         Y1        Y2        Y3
1 0.4720734 0.9669291 0.7242007
2 1.4720734 1.9669291 1.7934027
```

```
> abs(smean[1, ] - smean[2, ])
```

```
      Y1       Y2       Y3
1.002033 1.043610 1.082805
```

```
> gmm$a
```

```
Y1 Y2 Y3
 1  1 NA
```

5 Simulation of eQTL models of experimental crosses

We illustrate in this section how we can use *qpgraph* in conjunction with the *qtl* package (**?**) to simulate expression quantatitve trait loci (eQTL) models of experimental crosses and data from them. This functionality employs the previously described procedures to simulate an homogeneous mixed GMM that represents the underlying model of regulatory *cis*-eQTL, *trans*-eQTL and gene-gene associations, although this fact appears hidden to the user.

More concretely, the *qpgraph* package defines an object class called *eQTLcross* which basically holds a genetic map of the genotype markers (as defined by the *map* class in the *qtl* package from **?**) and an homogeneous mixed GMM defining the underlying molecular regulatory network that connects genotypes with genes and genes themselves, where we use the term *gene* to refer to a gene expression profile.

In a similar vein to the way we simulated before graphs and GMMs, we need to create a parameter object that defines the main features of the eQTL model we want to simulate. This is done through the function `eQTLcrossParam()` which by default defines some minimal eQTL model:

```
> eQTLcrossParam()
```

```
  eQTL backcross parameter object defining 20 markers,
  20 genes, 20 cis-eQTL and 0 trans-eQTL.

  cis-eQTL associations occur within 1.0 cM of a gene
  and all eQTL are located at 0.0 cM from a marker.

  Gene network parameters are defined by a

  d-regular pure graph parameter object
  No. of pure vertices: 20
  Constant degree: 2
  Vertex labels: g1, g2, g3, g4, g5, g6 ...
```

```
> args(eQTLcrossParam)
```

```
function (map = do.call("class<-", list(list(`1` = do.call("class<-",
    list(do.call("names<-", list(seq(1, 100, length.out = 20),
        paste0("m", 1:20))), "A"))), "map")), type = "bc", genes = 20,
    cis = 1, trans = as.integer(NULL), cisr = 1, d2m = 0, networkParam = dRegularGraphParam())
NULL
```

To simulate an *eQTLcross* object *qpgraph* provides the function `reQTLcross()` giving as first argument the number of *eQTLcross* objects we want to simulate and a *eQTLcrossParam* object:

```
> reQTLcross(n=2, eQTLcrossParam())
```

```
[[1]]

  eQTL backcross model with 20 markers, 20 genes,
  20 eQTL and 20 gene-gene expression associations.


[[2]]

  eQTL backcross model with 20 markers, 20 genes,
  20 eQTL and 20 gene-gene expression associations.
```

When the first argument `n` is omitted, then `n=1` is assumed by default. Other arguments to `reQTLcross()` are the mean marginal correlation between genes (`rho=0.5`), the magnitude of the linear additive effect of the simulated eQTL

associations (a=1), the minimum tolerance of the matrix completion algorithm that is involved in the construction of the conditional covariance matrix (tol=0.001) and whether progress on the calculations should be shown verbose=FALSE.

To simulate a larger *eQTLcross* object we need to simulate a genetic map using the sim.map() function from the *qtl* package (**?**), which should be loaded first. Since *qpgraph* overloads the *qtl* function sim.cross(), which will be used later to simulate data from an *eQTLcross* object, we will detach *qpgraph* before loading *qtl*, and load *qpgraph* again.

```
> detach("package:qpgraph")
> library(qtl)
> library(qpgraph)
> map <- sim.map(len=rep(100, times=20),
+                n.mar=rep(10, times=20),
+                anchor.tel=FALSE,
+                eq.spacing=FALSE,
+                include.x=TRUE)
```

and using it in combination with a larger number of genes (50) we can easily simulate this larger *eQTLcross* object:

```
> eqtl <- reQTLcross(eQTLcrossParam(map=map, genes=50))
> class(eqtl)
```

```
[1] "eQTLcross"
attr(,"package")
[1] "qpgraph"
```

```
> eqtl
```

```
  eQTL backcross model with 200 markers, 50 genes,
  50 eQTL and 50 gene-gene expression associations.
```

which, as we can see, it corresponds a backcross model with 50 genes each of them associated to a *cis*-eQTL, and with a certain underlying gene network embedded into an homogeneous mixed GMM that forms part of this object and which can be accessed as follows:

```
> eqtl$model
```

```
  Homogeneous mixed graphical Markov model
  with 50 discrete and 50 continuous r.v., and 100 edges.
```

A dot plot describing the eQTL associations along the given genetic map can be obtained by calling the plot function with the *eQTLcross* object as argument. In Figure **??** we see on the right panel such a plot, and on the left panel the genetic map plotted by the plot function defined in the *qtl* package (**?**) to plot genetic maps.

```
> par(mfrow=c(1,2))
> plot(map)
> plot(eqtl, main="eQTL model with cis-associations only")
```

A somewhat more complex eQTL model with *trans* associations can be simulated by using the trans argument as follows:

```
> set.seed(123)
> eqtl <- reQTLcross(eQTLcrossParam(map=map, genes=50,
+                                   cis=0.5, trans=c(5, 5)), a=5)
```

In this call, cis=0.5 indicates that 50% of the genes should have *cis*-eQTL associations and among the remaining ones, 10 will be associated to two *trans*-eQTL affecting 5 genes each of the two. We have also increased the default additive linear effect from the default value a=1 to a=5 which corresponds to a very strong linear additive effect from genotype markers on gene expression. We can examine the *cis* and *trans* associations of the *eQTLcross* object with the ciseQTL() and transeQTL() functions:

```
> head(ciseQTL(eqtl))
```

```
  chrom location  QTL gene a
1     1 70.72429 QTL1   g2 5
```

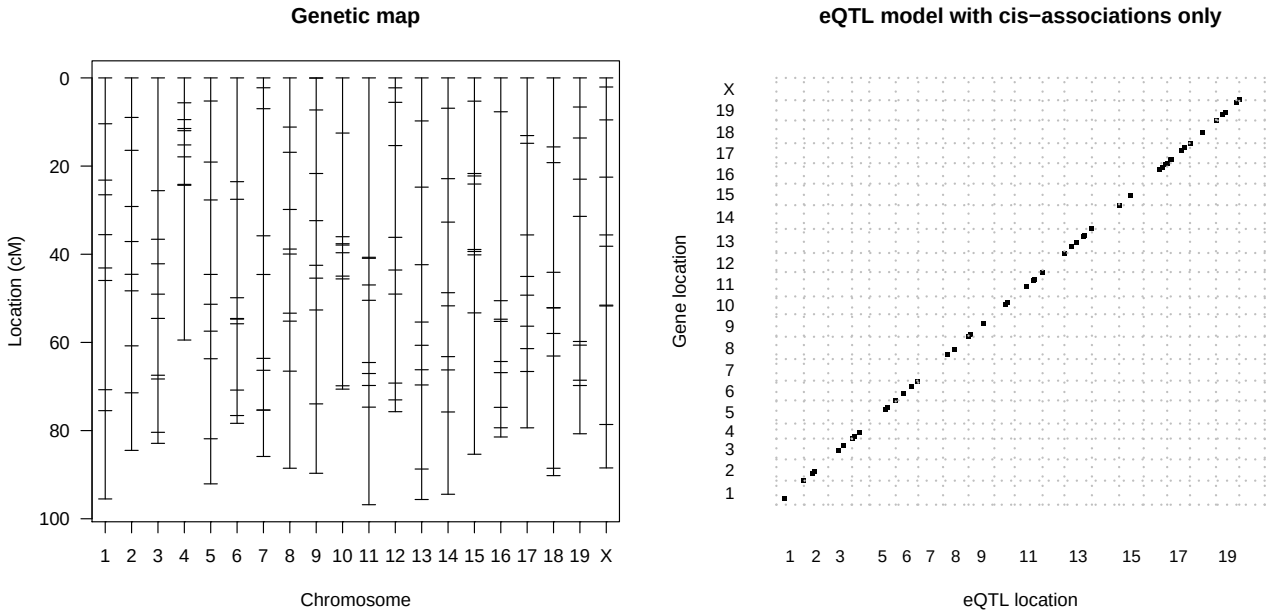

Figure 4: A genetic map simulated with the *qtl* package (**?**) on the left, and on the right, an eQTL model simulated using the genetic map with the *qpgraph* package.

```
2     1 75.47459 QTL2   g3 5
3     2 84.48877 QTL3   g4 5
4     5 51.34596 QTL4  g10 5
5     6 70.81028 QTL5  g15 5
6     6 78.35164 QTL6  g16 5
```

```
> transeQTL(eqtl)
```

```
   chrom location   QTL gene a
1     14 32.70995 QTL17   g6 5
2     14 32.70995 QTL17   g9 5
3     14 32.70995 QTL17  g13 5
4     14 32.70995 QTL17  g14 5
5     14 32.70995 QTL17  g24 5
6     18 88.54678 QTL24   g5 5
7     18 88.54678 QTL24  g18 5
8     18 88.54678 QTL24  g22 5
9     18 88.54678 QTL24  g36 5
10    18 88.54678 QTL24  g37 5
```

and examine where the eQTL associations occur and what genes map to *trans*-eQTL, as shown in Figure **??**.

```
> plot(eqtl, main="eQTL model with trans-eQTL")
```

Using this *eQTLcross* object we can simulate data from the corresponding experimental cross with the function `sim.cross()` from the *qtl* package (**?**) but which is overloaded in *qpgraph* to plug the eQTL associations into the corresponding genetic loci:

```
> set.seed(123)
> cross <- sim.cross(map, eqtl)
> cross
```

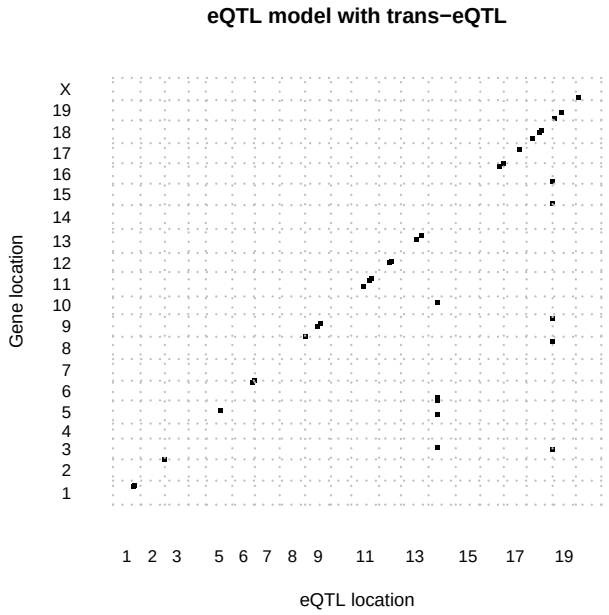

Figure 5: An eQTL model including trans-acting associations simulated using the genetic map from Fig. **??**.

```
This is an object of class "cross".
It is too complex to print, so we provide just this summary.
  Backcross

  No. individuals:    100

  No. phenotypes:     50
  Percent phenotyped: 100

  No. chromosomes:    20
      Autosomes:      1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
      X chr:          X

  Total markers:      200
  No. markers:        10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10
                      10
  Percent genotyped:  100
  Genotypes (%):
        Autosomes:    AA:51.7  AB:48.3
     X chromosome:    AA:50.6  AB:49.4
```

Note that here, the genotype data is simulated by the procedures implemented in the *qtl* package (**?**) and *qpgraph* adds to that simulation the eQTL and gene network associations. Thus, while the `rHMgmm()` function described in the previous section, does not simulate correlated discrete variables, here the genotypes will be correlated according to the input arguments of the `sim.cross()` function in *qtl* (mainly, the `map.function` argument that converts genetic distances into recombination fractions) and which can be passed through the previous call to `sim.cross()`.

Let's focus on a specific simulated eQTL, concretely on the second one of the following list:

```
> allcis <- ciseQTL(eqtl)
> allcis[allcis$chrom==1, ]
```

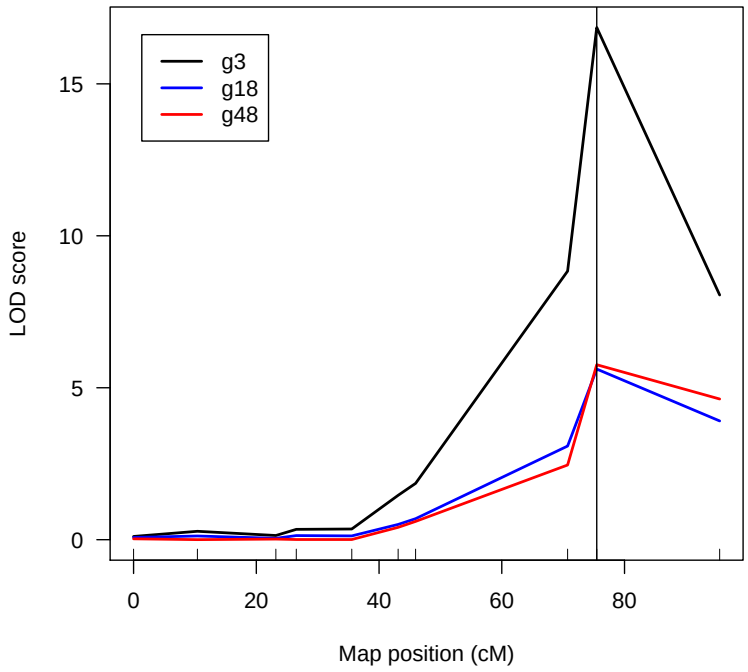

Figure 6: Profile of LOD scores for three genes with direct and indirect eQTL.

```
  chrom location  QTL gene a
1     1 70.72429 QTL1   g2 5
2     1 75.47459 QTL2   g3 5
```

```
> gene <- allcis[2, "gene"]
> chrom <- allcis[2, "chrom"]
> location <- allcis[2, "location"]
```

Find out the genes connected to gene g3 in the underlying regulatory network:

```
> connectedGenes <- graph::inEdges(gene, eqtl$model$g)[[1]]
> connectedGenes <- connectedGenes[connectedGenes %in% eqtl$model$Y]
> connectedGenes
```

```
[1] "g18" "g48"
```

Now, using the *qtl* package (**?**) and its `scanone()` function, we perform a simple QTL analysis by single marker regression using the expression profiles from genes g3, g18, g48 as phenotypes:

```
> out.mr <- scanone(cross, method="mr", pheno.col=c(gene, connectedGenes))
```

By using the plotting functionalities of the *qtl* package (**?**) we can examine the LOD score profile of these three genes on chromosome 1 where the eQTL of gene g3 is located:

```
> plot(out.mr, chr=chrom, ylab="LOD score", lodcolumn=1:3, col=c("black", "blue", "red"), lwd=2)
> abline(v=allcis[allcis$gene == gene, "location"])
> legend("topleft", names(out.mr)[3:5], col=c("black", "blue", "red"), lwd=2, inset=0.05)
```

We can observe in Figure **??** that not only the directly associated gene g3 seems to have an eQTL at position 75.5 cM, but also the genes g18, g48 connected to g3 in the underlying gene network. Using a permutation procedure implemented

in *qtl*, we calculate LOD score threholds that yield genome-wide statistical significant eQTL associations:

```
> out.perm <- scanone(cross, method="mr", pheno.col=c(gene, connectedGenes),
+                     n.perm=100, verbose=FALSE)
> summary(out.perm, alpha=c(0.05, 0.10))
```

```
LOD thresholds (100 permutations)
      g3  g18  g48
5%  2.90 2.63 3.04
10% 2.69 2.47 2.51
```

and examine which genotype markers yield such significant associations to these the expression profiles of these genes:

```
> summary(out.mr, perms=out.perm, alpha=0.05)
```

```
      chr  pos    g3   g18  g48
D1M9    1 75.5 16.85  5.61 5.76
D18M9  18 88.5  4.95 15.20 5.90
```

Notice that the directly associated gene g3 as well as the indirectly associated ones g18, g48 have genome-wide significant LOD scores on the same eQTL located at 75.5 cM in chromosome 1.

6 Session information

```
> toLatex(sessionInfo())
```

- R version 3.2.4 (2016-03-10), `x86_64-apple-darwin13.4.0`
- Locale: `C/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8`
- Base packages: base, datasets, grDevices, graphics, grid, methods, parallel, stats, stats4, utils
- Other packages: AnnotationDbi 1.32.3, Biobase 2.30.0, BiocGenerics 0.16.1, DBI 0.3.1, GenomeInfoDb 1.6.3, IRanges 2.4.8, RSQLite 1.0.0, Rgraphviz 2.14.0, S4Vectors 0.8.11, XML 3.98-1.4, annotate 1.48.0, genefilter 1.52.1, graph 1.48.0, org.EcK12.eg.db 3.2.3, qpgraph 2.4.2, qtl 1.39-5
- Loaded via a namespace (and not attached): BiocParallel 1.4.3, BiocStyle 1.8.0, Biostrings 2.38.4, GenomicAlignments 1.6.3, GenomicFeatures 1.22.13, GenomicRanges 1.22.4, Matrix 1.2-4, RCurl 1.95-4.8, Rsamtools 1.22.0, SummarizedExperiment 1.0.2, XVector 0.10.0, biomaRt 2.26.1, bitops 1.0-6, futile.logger 1.4.1, futile.options 1.0.0, lambda.r 1.1.7, lattice 0.20-33, mvtnorm 1.0-5, rtracklayer 1.30.3, splines 3.2.4, survival 2.38-3, tools 3.2.4, xtable 1.8-2, zlibbioc 1.16.0