Network models from time-course omics data – practical

Wessel N. van Wieringen[1,2] & Viktorian Miok[1]

April 21, 2017

| [1]Dept. of Epidemiology & Biostatistics
| Amsterdam Public Health research institute
| VU University medical center Amsterdam

| [2]Dept. of Mathematics
| Vrije Universiteit Amsterdam

| w.vanwieringen@vumc.nl (mailto:w.vanwieringen@vumc.nl)

0. Preliminaries

This practical aims to acquaint the reader with the R package `ragt2ridges`, which implements the methodology described in Miok, Wilting, and van Wieringen (2017) and forthcoming follow-up work for network reconstruction from time-course data.

The `ragt2ridges` package augments the `rags2ridges` package (Peeters, Bilgrau, and van Wieringen (2017)) with functionality for time-course studies. Being its sibling `ragt2ridges` mimicks `rags2ridges` in the function names.

The R code included builds up: it assumes code of the previous code blocks have been executed without error.

1. The VAR(1) model

The VAR(1) model is a simple description of the dynamics among a collection of random variables over time. It explains the vector of observations on these variables at the next time points by a linear combination of the current observations on these variables plus an error. The linear combination is represented by a matrix A with elements referred to as autoregression parameters. The error is modelled by a zero-centered multivariate normal distribution with a general covariance matrix.

The model is often percieved as comprising an endogeneous and exogeneous part of a system. The latter is simply modelled by the errors. These are referred to as innovations as they renew the process at each time step. The endogeneous part amounts to the matrix A that descibes how innovations are propagated within the system.

The VAR(1) model is represented graphically by a time-series chain graph. This graph depicts all

temporal and contemporaneous relationships (represented by directed and undirected edges, respectively) between the random variables (represented by nodes). Temporal edges indicate a direct effect of one random variable at one time point on another random variable at the next time point. Temporal edges correspond to non-zero elements of the autoregression coefficient matrix A. The contemporaneous edges indicate an “instantaneous” (i.e., within a time point) and undirected relationship between two random variables. Contemporaneous edges correspond to non-zero elements of the error precision matrix (i.e., the inverse error covariance matrix).

1.1. Libraries

Activate required libraries:

```
# load libraries
library(Biobase)
library(lattice)
library(longitudinal)
library(rags2ridges)
library(ragt2ridges)
library(SparseTSCGM)
```

1.2. Data

The practical uses an *in vitro* oncogenomics study with a longitudinal experimental design that serves to unravel the dynamic interactions among genes during cervical carcinogenesis. The human papilloma virus (HPV), a carcinogenic entity, is inserted into normal cells, yielding an immortalized cell line that faithfully mimicks cervical cancer development morphologically and genetically. As the transfected cell line goes through distinct phenotypic phases, cells are profiled transcriptomically at eight time points uniformly distributed over the transformation process. The experiment is quadruplicated, i.e. four cell lines are subjected to this procedure.

The data of the aforementioned experiment provided through the `ragt2ridges` -package have been preprocessed and are limited to those transcripts that map to the P53 signalling pathway (as defined by the KEGG repository, Ogata et al. (1999)). In all, the data comprises expression levels of 64 transcripts measured at 8 time points in 4 cell lines. For more details on the experiment, data and its preprocessing, see Wilting et al. (2016).

The observed changes in the pathway’s transcript levels over time are modelled by the VAR(1) model. This should shed light on the dynamics of the P53 signalling pathway after HPV transfection.

Load and reformat data for later use:

```
# load data
data(hpvP53)

# reformat data
Y <- longitudinal2array(t(exprs(hpvP53rna)))

# zero center data, variate- and cell line-wise
Y <- centerVAR1data(Y)
```

The object `Y` contains the data in array format. Its rows, columns and slices represent the molecular entities, time points and cell lines, respectively.

1.2.1 Data subsetting

If desired, e.g. for speed, the pathway may be subsetted (remove the "##Dont run:" at the beginning of each line):

```
##Dont run: # specify subset to be analyzed
##Dont run: id        <- sample(1:dim(Y)[1], 10)

##Dont run: # subset both extracted and original data
##Dont run: Y         <- Y[id, , , drop=FALSE]
##Dont run: hpvP53rna <- hpvP53rna[id, ]
```

The `hpvP53rna` -object is also subsetted as it provides the names of the molecular entities, which are used later for plotting purposes.

1.3. Data exploration

Before embarking on a rather complicated description of the data first explore these data visually. A good starting point is a time-course plot, which plots the values of a variate over time. This may be done using the `plotVAR1data` -function, which uses different colors and line styles for variates and individuals, respectively.

Plot time-courses:

```
# plot time-courses of all variates
plotVAR1data(Y)
```

This is rather messy and may be limited to a single variate.

Plot single time-course:

```
# plot time-courses of a single variate
plotVAR1data(Y[5, , , drop=FALSE])
```

This variate shows nice concordant behaviour over the individuals.

With many variates different visualizations may offer a better impression of their joint behaviour. For

instance, a heatmap may be insightful, possible accompanied by prior clustering of the data variate-wise.

Draw a heatmap of the first cell line data, with prior K-means clustering applied:

```
# K-means clustering of the variates
cellLine <- 1
kClust   <- kmeans(Y[, , cellLine], centers=7, nstart=100)$cluster

# heatmap of reshuffled data
edgeHeat(Y[unlist(lapply(1:max(kClust),
                         function(id, clusters){ which(clusters==id) },
                         kClust)), , cellLine])
```

The consistent dynamic behaviour among the variates within a cluster becomes more apparent.

Draw time-course plot for a cluster:

```
# plot time-courses of smallest cluster
plotVAR1data(Y[kClust==which.min(table(kClust)), , , drop=FALSE])
```

Clearly, the variates from the selected cluster show nice concordant time-courses.

1.4. Penalty selection

The VAR(1) model parameters are estimated by means of ridge penalized likelihood maximization. To this end the log-likelihood is augmented with two ridge penalties: the sums of the squares of the elements of both parameters. Each sum is multiplied by its own ridge penalty parameter (which is then common to all parameter elements). For given ridge penalty parameters the resulting penalized log-likelihood is maximized. The parameter values for which this maximum is attained are the ridge estimates. Cf. Miok, Wilting, and van Wieringen (2017) and van Wieringen and Peeters (2016) for more details.

Ridge penalized likelihood maximization requires to make an informed choice of the ridge penalty parameters: one for the autoregression coefficient matrix A and one for the precision matrix. Within the `ragt2ridges` package this is done by means of cross-validation. The penalty parameters that maximize the leave-one-out cross-validated (LOOCV) log-likelihood are considered to be optimal.

Search for optimal penalty parameters:

```
# find optimal penalty parameters
optLambdas <- optPenaltyVAR1(Y, lambdaMin =c(0.01, 0.00001),
                             lambdaMax =c(1000, 1),
                             lambdaInit=c(100 , 0.1))

# print optimal penalty parameters
print(optLambdas)
```

The convergence of the build-in R optimizer behind the `optPenaltyVAR1` -function needs to be

assessed. This is done through visual inspection of the contour plot of the LOOCV log-likelihood.

Specify grid and evaluate the LOOCV log-likelihood at these grid points:

```
# define grid for both penalty parameters
lambdaAgrid <- seq(400,    450, length.out=20)
lambdaPgrid <- seq(0.001, 0.01, length.out = 20)

# evaluate LOOCV log-likelihood over the grid
LOOCVres    <- loglikLOOCVcontourVAR1(lambdaAgrid, lambdaPgrid,
                                      Y, verbose=FALSE)
```

Plot the LOOCV log-likelihood contour and add the optimal penalty parameters:

```
# plot LOOCV log-likelihood somewhat nicer
contour(lambdaAgrid, lambdaPgrid, LOOCVres$llLOOCV, xlab="lambdaA",
        ylab="lambdaP", main="cross-validated log-likelihood", nlevels=25)

# add optimal penalty parameters as red dot
points(optLambdas[1], optLambdas[2], pch=20, cex=2, col="red")
```

Assess whether the selected penalty parameters are indeed close to the optimum of the LOOCV log-likelihood.

The LOOCV log-likelihood contour plot may indicate that the penalty parameters selected by the `optPenaltyVAR1` -function do not match with the optimum of the LOOCV log-likelihood. One option to resolve this is to use another build R optimizer. The optimizer to be used by the `optPenaltyVAR1` -function is specified through its `optimizer` argument. Persistence of the failure after a change of optimizer may be due to either a flat optimum of the LOOCV log-likelihood or a poor choice of the initial penalty parameter values. The latter may be fixed by a more informed initial guess which can be obtained from the contour plot with a rather coarse grid for screening.

1.5. Penalized estimation

Having decided on the value of the penalty parameters, the VAR(1) model parameters can be obtained by ridge maximum likelihood estimation.

Estimate and extract the VAR(1) model parameters:

```
# fit the model
VAR1hat        <- ridgeVAR1(Y=Y,
                            lambdaA=optLambdas[1],
                            lambdaP=optLambdas[2])

# extract parameter estimates
Ahat           <- VAR1hat$A
Phat           <- VAR1hat$P

# add row and column names
rownames(Ahat) <- colnames(Ahat) <- rownames(Phat) <-
                  colnames(Phat) <- rownames(hpvP53rna)
```

Inspect ridge estimates of the VAR(1) model parameters (A and precision matrix) through heatmaps:

```
# heatmap of estimate of A
edgeHeat(Ahat, main="ridge estimate of A")
```

```
# heatmap of partial correlation matrix estimate
edgeHeat(pcor(Phat), main="ridge partial correlation estimate", diag=FALSE)
```

Note that not the precision matrix of the innovations, but the corresponding partial correlation matrix is displayed. The latter forms the basis for the selection of the contemporaneous edges.

The code above illustrates the computation of the ridge maximum likelihood estimates of the VAR(1) model parameters. Although the code is (reasonably) optimized to provide a fast implementation, it may be considered slow in cases with parameter dimensions larger than (say) hundreds. As an alternative one may then consider to estimate the autoregression coefficient matrix A by means of ridge penalized sum-of-squares minimization. This is specified through setting `fitA="ss"` when invoking the `ridgeVAR1` -function. The same should be done in the optimal penalty parameter selection (the `optPenaltyVAR1` and `loglikLOOCVcontourVAR1` functions then propagate this internally to the `ridgeVAR1` -function).

Should one have prior knowledge, e.g. from a pilot experiment or a publicly available external data set on a comparable model system, on either the autoregression coefficient matrix A or the precision matrix, these can be provided as suggestions in the ridge penalized estimation of the VAR(1) model through the options `targetA` and `targetP` in the `ridgeVAR1` -function. The provided parameter suggestions appear in the ridge penalty and their estimates are shrunken towards the corresponding suggestion for large values of the penalty parameters .

It may be that the endogeneous part of the VAR(1) model is an adequate enough description of the system, leaving no residual correlation among the innovations. The VAR(1) model is then to be fitted with a diagonal error covariance matrix. This is specified through setting `diagP=TRUE` when calling the `ridgeVAR1` -function.

1.6 Stability analysis

A rather big VAR(1) model has thus been estimated from a relatively small amount of data. For reassurance assess the sensitivity of the obtained estimate. This is done through the study of the parameter estimates obtained from a perturbed version of the data set. Here perturbation amounts to the one-by-one removal of a single design point from the data set. From the thus perturbed data set the VAR(1) model parameters are re-estimated (including updating of the penalty parameters through LOOCV). Ideally, the resulting perturbed parameter estimates resemble – in some relevant sense – the original one.

Assess stability (WARNING: be patient, takes a while):

```
# leave each time point out once and re-estimate
Aperturb <- numeric()
for (u in 1:((dim(Y)[2]-1)*dim(Y)[3])){
  # leave one sample out
    unbalanced <- cbind(rep(2:dim(Y)[2], dim(Y)[3]),
                        sort(rep(1:dim(Y)[3], dim(Y)[2]-1)))[u, , drop=FALSE]

    # choose penalty parameters and estimate model
    newLambda <- optPenaltyVAR1(Y,
                                lambdaMin =c(100, 0.001),
                                lambdaMax =c(1000, 1),
                                lambdaInit=optLambdas,
                                optimizer ="optim",
                                unbalanced=unbalanced)

    # refit
    Aperturb  <- cbind(Aperturb,
                       as.numeric(ridgeVAR1(Y,
                                            lambdaA   =newLambda[1],
                                            lambdaP   =newLambda[2],
                                            unbalanced=unbalanced)$A))
}

# select hundred equidistant autoregression parameters
ids <- match(ceiling(seq(1, nrow(Aperturb), length.out=100)),
             rank(Ahat))
```

The perturbed parameter estimates are now to be compared against the ones from the original full data set. Here this is done for hundred entries from the autoregression coefficient matrix A. These hundred have been selected from the vectorized original parameter estimate such that they are equidistantly distributed over the full entry range. The (order of the) selected entries of the original estimate of A are plotted against those from all perturbed estimates of this parameter. Ideally, the size of the original and perturbed parameter estimates is concordant.

Inspect sensitivity through a boxplot of original against perturbed estimates:

```
# plot boxplots of the left-out estimates
boxplot(as.numeric(Aperturb[ids,]) ~ rep(1:length(ids), ncol(Aperturb)),
        col ="blue", border="lightblue", pch=20, cex=0.5,
        xlab="original estimate ranking",
        ylab="entry of perturbed A estimate")
```

The boxplot suggests that the ranking, i.e. the relative size of the autoregression parameters, is preserved over the perturbed data sets. The original ridge penalized ML estimates of the VAR(1) model thus seem reasonably thrustworthy, in the sense that the methodology is robust to the applied perturbation of the data set.

1.7. Support determination

Sparse parameter estimates are desirable for many practical purposes. However, ridge estimates are generally not sparse. But many elements of the VAR(1) model parameters may be close to zero. The `ragt2ridges` package provides various methods for thresholding the elements of these parameters and thus inferring which are effectively zero and which are not.

Determine support for the autoregression coefficient matrix A and error precision matrix:

```
# support determination of A
zerosA <- sparsifyVAR1(A=Ahat, SigmaE=symm(solve(Phat)),
                       threshold="top", top=25,
                       statistics=FALSE, verbose=FALSE)$zeros

# support determination of precision matrix
zerosP <- sparsify(Phat, threshold="top", top=10,
                   output="light", verbose=FALSE)$zeros
```

The code above simply selects the largest elements of A and those among the nonredundant elements of the partial correlation matrix of the errors.

The `sparsify` and `sparsifyVAR1` -functions also offer a (slightly) more sophicated empirical Bayes approach, which is specied by the options `threshold="localFDR"` in combination with `FDRcut=0.95` . This approach assumes that the (say) partial correlations stem from a two-component mixture. The two components represent the distributions of partial correlations corresponding to absent and present contemporenous edges. The null component – corresponding to absent edges – is fitted by means of truncated likelihood maximization. Then, the probability of observing a partial correlation – corresponding to an absent edge – equal to or larger than an estimated partial correlation is calculated. This probability may be endowed with a local FDR interpretation (Efron (2004)). The complementary probability indicates whether an edge is interesting. Edges with a complementary probability larger than a significance value are selected. The empirical Bayes approach, however, is only valid for networks with a reasonably sized number of nodes (say) at least $p > 50$ and, consequently, a vector of 2500 regression coefficients. In this illustration p is rather small and a simple edge selection alternative is used instead.

1.8. Re-estimation with inferred support

The above provides the ridge estimates and inferred support of the VAR(1) model parameters. However, the former is not sparse and – strictly – does thus not comply with the latter. It may then be appropriate to re-estimate the parameters such that their estimates obey the inferred support. This requires to redo the search for optimal penalty parameters and (with these at hand) to re-estimate the parameter, in both cases taking the inferred support into account. Hereto the same functions as introduced above are used, now with the inferred support provided.

Search for optimal penalty parameters with inferred support:

```
# format precision support
supportP <- support4ridgeP(zeros=zerosP, nNodes=nrow(Y))

# optimal penalty parameter determination
optLambdas <- optPenaltyVAR1(Y, lambdaMin=c(10^(-5), 10^(-5)),
                             lambdaMax=c(10, 0.1), lambdaInit=c(5, 0.01),
                             zerosA=zerosA, zerosP=zerosP,
                             cliquesP=supportP$cliques,
                             separatorsP=supportP$separators,
                             zerosAfit="sparse")
print(optLambdas)
```

In the above the support of the precision matrix is reformatted as a list of cliques and separators. This requires the support to be chordal (i.e. a particularly structured graph). When this is not the case, it is casted (through triangulation) to be chordal. Effectively, this adds a minimum, but possibly a non-negligible number, of edges to the support of the precision matrix such that the resulting graph is chordal. This process is called chordal embedding. The edges added in chordal embedding can be inspected through the `addedEdges` -slot of the `supportP` -object. The penalty optimization and estimation then proceed with the chordal support.

The convergence of the build-in R optimizer behind the `optPenaltyVAR1` -function needs to be assessed. This is done through visual inspection of the contour plot of the LOOCV log-likelihood.

Specify grid and evaluate the LOOCV log-likelihood at these grid points:

```
# define grid for both penalty parameters
lambdaAgrid <- seq(0.001, 1.5, length.out=20)
lambdaPgrid <- seq(0.001, 0.02, length.out = 20)

# evaluate LOOCV log-likelihood over the grid
LOOCVres    <- loglikLOOCVcontourVAR1(lambdaAgrid, lambdaPgrid,
                                      Y, zerosA=zerosA, zerosP=zerosP,
                                      cliquesP=supportP$cliques,
                                      separatorsP=supportP$separators,
                                      zerosAfit="sparse")
```

Plot the LOOCV log-likelihood contour and add the optimal penalty parameters:

```
# plot LOOCV log-likelihood somewhat nicer
contour(lambdaAgrid, lambdaPgrid, LOOCVres$llLOOCV,
        xlab="lambdaA", ylab="lambdaP", main="cross-validated
        log-likelihood", nlevels=25)

# add optimal penalty parameters as red dot
points(optLambdas[1], optLambdas[2], pch=20, cex=2, col="red")
```

Assess whether the selected penalty parameters are indeed close to the optimum of the LOOCV log-likelihood.

Re-fit the VAR(1) model with inferred support and extract re-estimated parameters:

```
#re-fit the model
VAR1hat        <- ridgeVAR1(Y=Y, lambdaA=optLambdas[1],
                            lambdaP=optLambdas[2], zerosA=zerosA,
                            cliquesP=supportP$cliques,
                            separatorsP=supportP$separators,
                            zerosP=zerosP, zerosAfit="sparse")

# extract parameter estimates
Ahat           <- VAR1hat$A
Phat           <- VAR1hat$P

# add row and column names
rownames(Ahat) <- colnames(Ahat) <- rownames(Phat) <-
                  colnames(Phat) <- rownames(hpvP53rna)
```

Inspect the support of the ridge estimates of the VAR(1) model parameters:

```
# heatmap of support of A
edgeHeat(adjacentMat(Ahat), legend=FALSE, main="inferred support of A")
```

```
# heatmap of support of precision matrix
edgeHeat(adjacentMat(Phat), legend=FALSE, main="inferred support of
                            precision matrix", diag=FALSE)
```

The heatmap of the adjacency matrix of the autoregression coefficient matrix A reveals the presence of “regulators” and “regulatees”. “Regulators” influence many other nodes. This is reflected in the estimate of A by many nonzero elements in the corresponding column. Vice versa, “regulatees” are influenced by many other nodes, which is reflected by many nonzero elements in the corresponding row of A. Simply put: a vertical or horizontal stripy pattern thus hints at a “regulator” or “regulatee”, respectively.

Inspect the ridge estimates of the VAR(1) model parameters (A and precision matrix) through heatmaps:

```
# heatmap of estimate of A
edgeHeat(Ahat, main="ridge re-estimate of A with inferred support")
```

```
# heatmap of support of the partial correlation matrix
edgeHeat(pcor(Phat), main="ridge re-estimate of the partial correlation matr
ix
         with inferred support", diag=FALSE)
```

Again not the precision matrix of the innovations, but the corresponding partial correlation matrix is displayed.

Plot the time-series chain graph of the estimated VAR(1) model:

```
# time-series chain graph
graphVAR1(Ahat, Phat, nNames=rownames(Ahat), type="TSCG",
          vertex.label.cex=0.5, vertex.label.font=1, vertex.size=4,
          vertex.label.color.T0="darkblue",
          vertex.label.color.T1="darkblue",
          vertex.frame.color="steelblue", vertex.color.T0="lightblue",
          vertex.color.T1="lightblue", edge.width=1.5, main = "")
```

The contemporaneous and temporal edges connect nodes from the same and different time points, respectively, and correspond to nonzero elements in the ridge estimates of the VAR(1) model parameters, i.e. precision matrix and A, respectively.

Instead of the time-series chain graph one may plot the “global” (i.e., over all time) conditional independence graph. The absence/presence of an edge in this graph indicates that two subprocesses, i.e., two sets of random variables over time, are “globally” in/dependent conditional on the remaining subprocesses. A “global” conditional independence is implied by a zero partial correlation between the two random variables from any two arbitrary time points. The sparse VAR(1) model parameter estimates provide the necessary information to assess this.

Determine the adjacency matrix of the “global” conditional independence graph CIG of the VAR(1) model and plot the resulting graph:

```
# determine adjacency matrix
adjMatCIG <- CIGofVAR1(Ahat, Phat)

# plot global conditional independence graph
graphVAR1(Ahat, Phat, nNames=rownames(Ahat), type="globalPC")
```

1.9. Fit assessment

A good model describes the data well. This is assessed through comparison of the observations with their fits.

Calculate the fit of the estimated VAR(1) model:

```
# calculate fits and fit-vs-observation correlations
Yhat <- array(dim=dim(Y))
for (i in 1:dim(Y)[3]){ Yhat[, -1, i] <- Ahat %*% Y[, -dim(Y)[2], i] }
```

Assess the fit for the 1st molecular entity:

```
# specify molecular entity of interest
entityID <- 1

# format data for plotting
label               <- paste("CellLine", sort(rep(c(1:dim(Y)[3]),
                                                  dim(Y)[2]-1)))
time                <- rep(2:dim(Y)[2], dim(Y)[3])
entityData          <- data.frame(as.numeric(Y[entityID, -1, ]),
                                  time, label)
colnames(entityData) <- c("Data", "TimePoints", "CellLine")

# plot data and fit for each cell line separately
xyplot(Data ~ TimePoints | CellLine, data=entityData,
       layout=c(dim(Y)[3], 1), col="blue", pch=20, aspect=1)
for (i in 1:dim(Y)[3]){
    trellis.focus("panel", i, 1)
    llines(Yhat[entityID, -1, i] ~ c(2:dim(Y)[2]), col="red", lwd=2)
}
trellis.unfocus()
```

Visual inspection for each molecular entity separately may be time consuming when there are many. The model fit can be assessed globally by inspection of the distribution of the correlation between fit and observation per (molecular entity, cell line)-combination. Ideally, this distribution is concentrated on the positive unit interval [0,1], corresponding to positive correlations between fit and observation.

Calculate the (Spearman rank) correlations between fit and observation and plot as histogram:

```
# calculate fits and fit-vs-observation correlations
for (i in 1:dim(Y)[3]){ Yhat[, -1, i] <- Ahat %*% Y[, -dim(Y)[2], i] }
corFit <- numeric()
for (j in 1:dim(Y)[1]){
    slHelper <- numeric()
    for (i in 1:dim(Y)[3]){
      slHelper <- c(slHelper, cor(Yhat[j, -1, i], Y[j, -1, i], m="s"))
    }
    corFit <- rbind(corFit, slHelper)
}

# histogram of the fit-vs-observation correlations
hist(corFit, xlab="Correlation", ylab="Frequency", main="Histogram of the
     correlation fit vs. observation", n=20, col="blue", border="lightblue",
     xlim = c(-1, 1))
```

The number of fit-vs-observation correlations in the histogram need not sum to the total number of possible correlations. Some correlations may equal `NA` and are omitted in the histogram. This is due to the fact that these molecular entities have no incoming temporal edges. Consequently, the fit is constant and its correlation with the observation is undefined.

A new forecaster (e.g., of the weather) is generally benchmarked against two minimally sophicated but common-sense competitors. The first competitor forecasts the next observation by the current one (e.g., tomorrow's weather will be like today's). The other competitor uses the long-term average (e.g., the annual weather trend) as forecast. In the current example the latter may be operationalized as the average of the other cell lines which is then used to predict the observations of the left-out cell line.

Benchmark the constructed VAR(1) forecaster against these two less sophicated competitors. Hereto replace the two lines in the previous code block by either:

```
# "previous observation"-forecasts
for (i in 1:dim(Y)[3]){ Yhat[, -1, i] <- Y[, -dim(Y)[2], i] }
```

or:

```
# "other cell lines"-forecasts
for (i1 in 1:dim(Y)[3]){
  Yslh <- 0 * Y[, , 1]
  for (i2 in 1:dim(Y)[3]){
    if (i2 != i1){ Yslh <- Yslh + Y[, , i2] }
  }
    Yhat[, , i1] <- Yslh / (dim(Y)[3]-1)
}
```

Compare the fits of the benchmark forecasters to the constructed VAR(1) model, for individual genes as well as by the (histogram) of the fit-vs-observation correlations.

1.10. Down-stream analysis

The network may yield a pretty picture but its potential is generally not immediate to the medical researcher. For his/her acceptance the fitted network/model needs to be translated into tangible numbers or consequences. Some means to this end are provided below.

1.10.1. Node statistics

First efforts to grasp the network tend to concentrate on the calculation of node statistics, e.g., their in- and out-degree and various centrality measures (cf. Newman (2010) for details). On the basis of such node statistics nodes may be prioritized or endowed with an interpretation.

Calculate node statistics:

```
# calculate node-wise network stats
nodeStats <- nodeStatsVAR1(Ahat, Phat, as.table=TRUE)

# show node degree table
print(nodeStats[, 1:7])
```

From the printed table “hubs” (i.e., nodes with many connections) may be delineated. These may be further dissected into “regulators” and “regulatees” (i.e., having a large out- and in-degree, respectively). The other measures (e.g., betweenness, eigen-centrality) may also be used to identify nodes of importance (in some sense) to the network.

1.10.2. Mutual information

The node statistics calculated above only use the reconstructed topology of the time-series chain graph. That is, the actual weights of the edges (i.e., the actual values of the elements of the estimates of A and the error precision matrix) are not used. Incorporation of these weights allows a more attenuated and quantitative assessment of the importance of the nodes in the time-series chain graph.

The mutual information is a generalized correlation measure: the correlation of a random variable at the current time with all random variables at some future time (conditional on all past random variables). As such it is thus an indication of a random variable’s association with the system’s future. Nodes with no out-going edges will not affect random variables at future time points and their mutual information will be zero. In contrast, “regulators” with many out-going edges exert a large influence on other nodes at the next time point and their mutual information is expected to be large.

Calculate node statistics:

```
# specify time lag
lag <- 1

# evaluate mutual informations with specified lag
MIs <- mutualInfoVAR1(Ahat, solve(Phat), lag)
```

The `MIs` -object is a vector with mutual informations. The j-th element of this vector is the mutual information of the j-th random variable and all random variables at `lag` time points.

Study the mutual informations over larger time lags: do they decay? Which nodes are most influential for future observations?

1.10.3. Impulse response

The VAR(1) model describes how a signal coming in through the innovations is propagated through the system. This description may be exploited to predict how a perturbation of the system (e.g., a knock-out or some stimulus) will affect the random variables at future time points. Impulse response analysis provides: it calculates the change in the random variables due to a change in the innovations at a previous time point.

Calculate impulse responses:

```
# specify time lag
lag <- 1

# evaluate impulse response with specified lag
IRs <- impulseResponseVAR1(Ahat, lag)
```

The `IRs` -object is a matrix with impulse responses. The (j_1, j_2)-th element of this matrix is the effect induced in the j_1-th random variable at `lag` time points after a unit change in the j_2-th innovation.

Study the impulse response over larger time lags: do they decay? Also assess through which node a “regulatee” may be repressed best.

1.10.4. Path decomposition analysis

Finally, the VAR(1) model may be used to decompose the covariance between a random variable at the current time point with a random variable at some future time point (conditional on all past random variables) into the contribution of all the paths in the time-series chain graph that connect these two random variables. As such it provides a means to decompose the propagation of a signal (i.e., innovation) into the contribution of each path with the time-series chain graph connecting these node. WARNING: not yet implement!

1.11. Penalized estimation – lasso

Would one be a strong believer in lasso-type methods and its accompanying motivating sparsity assumption, he/she may wish to estimate the VAR(1) model not in ridge but lasso penalized fashion. The `SparseTSCGM` -package, implementing the work of Abegaz and Wit (2013), offers something along these lines. It differs from a full ML lasso estimation procedure in two respects: a) it uses a SCAD – a lasso-relative – penalty to estimate the matrix A, and 2) the matrix A is estimated in penalized sum-of-squares fashion (in stead of penalized maximum likelihood). Furthermore, in contrast to the ridge ML estimation procedure detailed above optimal penalty parameters are chosen not through LOOCV but by means of a Bayesian information criterion.

Fit VAR(1) model by means of the `SparseTSCGM` implemented lasso procedure:

```
# reformat data
Y <- as.longitudinal(array2longitudinal(Y[1:10,,]))

# find BIC-optimal penalty parameters and fit VAR(1) model
VAR1lasso <- sparse.tscgm(Y, optimality = "bic")

# extract parameters
AhatL           <- VAR1lasso$gamma
PhatL           <- VAR1lasso$theta
rownames(AhatL) <- colnames(AhatL) <- rownames(PhatL) <-
                   colnames(PhatL) <- rownames(hpvP53rna)[1:10]
```

For relatively low-dimensional data (i.e., p is smaller than the number of cell lines times the number of time points) the `SparseTSCGM` -implemention of the lasso estimation of the VAR(1) model converges reasonably fast. For high-dimensional data convergence is troublesome.

1.12 Simulate data

So far real data has been used throughout. This serves to illustrate the use of the package. To assess the performance of the implemented methodology, for instance to compare it with other methods, knowledge of the underlying data generating mechanism is needed. To this end the package provides functionality to sample from a VAR(1) model with specified parameters. The `createA` - and `createS` -functions (the latter from the `rags2ridges` -package) construct VAR(1) model parameters, the autoregression coefficient matrix and the (inverse of the) error precision matrix, respectively. With parameters at hand data may be drawn from the VAR(1) model using the `dataVAR1` -function.

Sampling from the VAR(1) model:

```
# set dimensions (p=covariates, n=individuals, T=time points)
p <- 3; n <- 4; T <- 10

# set model parameters
A      <- -createA(p, "clique", nCliques=1, nonzeroA=0.1)
SigmaE <- diag(p)/4

# generate data
Y <- dataVAR1(n, T, A, SigmaE)
```

Acknowledgements

Carel F.W. Peeters provided useful comments on a preliminary version of this practical which led to this improved version.

References

Abegaz, F, and E Wit. 2013. “Sparse time series chain graphical models for reconstructing genetic networks.” *Biostatistics* 13 (3): 586–99.

Efron, B. 2004. “Large-scale simultaneous hypothesis testing.” *Journal of the American Statistical Association* 99 (465).

Miok, V, S M Wilting, and W N van Wieringen. 2017. “Ridge estimation of the VAR(1) model and its time series chain graph from multivariate time-course omics data.” *Biometrical Journal* 59 (1): 172–91.

Newman, M. 2010. *Networks: An Introduction*. Oxford University Press.

Ogata, H, S Goto, K Sato, W Fujibuchi, H Bono, and M Kanehisa. 1999. “KEGG: Kyoto Encyclopedia of Genes and Genomes.” *Nucleic Acids Research* 27: 29–34.

Peeters, C F W, A E Bilgrau, and W N van Wieringen. 2017. “rags2ridges: ridge estimation of

precision matrices from high-dimensional data. R package version 2.2." https://CRAN.R–project.org/package=rags2ridges.

van Wieringen, W N, and C F W Peeters. 2016. "Ridge estimation of the inverse covariance matrix from high-dimensional data." *Computational Statistics and Data Analysis* 103: 284–303.

Wilting, S M, V Miok, A Jaspers, D Boon, H Sørgård, M Lando, B C Snoek, et al. 2016. "Aberrant methylation-mediated silencing of microRNAs contributes to HPV-induced anchorage independence." *Oncotarget* 7 (28): 43805.