Emiliano Díaz

Causality and Surrogate Variable Analysis

Submission Date: July 8 2016

Adviser: Prof. Dr. Marloes Maathuis
I would like to express my gratitude to Prof. Dr. Marloes Maathuis for all her patience and guidance.
Abstract

Gene expression depends on thousands of factors and we usually only have access to tens or hundreds of observations of gene expression levels meaning we are in a high-dimensional setting. Additionally we don’t always observe or care about all the factors. However, many different gene expression levels depend on a set of common factors. By observing the joint variance of the gene expression levels together with the observed primary variables (those we care about) Surrogate Variable Analysis (SVA) seeks to estimate the remaining unobserved factors. The ultimate goal is to assess whether the primary variable (or vector) has a significant effect on the different gene expression levels, but without estimating unobserved factors first the various regression models and hypothesis tests are dependent which complicates significance analysis.

In this work we define a class of additive gene expression structural equation models (SEMs) which are convenient for modeling gene expression data and which provides a useful framework to understand the various steps of the SVA methodology. We justify the use of this class from a modeling viewpoint but also from a causality viewpoint by exploring the independence and causality properties of this class and comparing to the biologically driven data assumptions. For this we use some of the theory that has been developed elsewhere on graphical models and causality. We then give a detailed description of the SVA methodology and its implementation in the R package sva referring each step to different parts of the additive gene expression SEM defined previously.

Given the possible dependency of the primary variable with unobserved factors, recovering these factors accurately presents several issues. SVA tries to solve some of them by obtaining unobserved factors that are allowed to be correlated to the primary variable, however the exact form of this dependency is not modeled and so is not necessarily accurate. We perform simulation experiments and sensibility analysis to assess the performance of SVA, comparing it to other methods designed for the same purpose, and to identify for which parameters of the additive gene expression SEM, SVA performs better. In general, we found SVA performs comparably well at estimating unobserved factors to SVDR and much better than all other methods at capturing the dependence of these factors to the primary variable. Additionally the significance analysis performed were shown to be much closer to being valid for SVA than those for other methods. However, for the base scenario simulated this did not translate into significant improvement in the accuracy of the estimation of the effect of the primary variable on gene expression levels. Using sensitivity analysis we found that, assuming that performance measures vary independently with respect to the different parameters, for gene expression SEMS with low number of unobserved factors, high level of gene expression variables, high sparsity (low number of edges in the SEM), a lower variance for unobserved factors than for the primary variable and low complexity for the signals from the primary variable to the unobserved factors and to the gene expression
level variables, SVA has superior performance to the other methods considered, including in terms of the accuracy of the estimation of the effect of the primary variable on gene expression levels.
## Contents

1 Introduction ............................................. 1  
  1.1 Structure ........................................... 1  
  1.2 Surrogate Variable Analysis ......................... 1  

2 Causality .................................................. 5  
  2.1 DAGs and SEMs ........................................ 5  
  2.2 Causality ............................................. 13  
  2.3 Additive Gene Expression SEMs ....................... 15  

3 SVA Methodology .......................................... 19  
  3.1 Overview ............................................. 19  
  3.2 Estimation of \( \text{span}(\{c_l\}) \) .................... 19  
      3.2.1 Estimation procedure ............................ 19  
      3.2.2 Estimation of basis function model ............. 22  
      3.2.3 Factorization of residuals using SVD ............ 23  
      3.2.4 Parallel Analysis ................................ 26  
  3.3 Estimation of \( \text{span}(\{h_k\}) \) .................... 30  
      3.3.1 General description ............................. 30  
      3.3.2 Finding the signature of \( c_l \) ................ 31  
      3.3.3 FDR based significance analysis ................. 34  

4 Simulation Experiments ................................... 43  
  4.1 Methods to be compared ............................... 43  
  4.2 Low-dimensional experiments ......................... 44  
      4.2.1 Design ........................................... 44  
      4.2.2 Result for one repetition ........................ 45  
      4.2.3 Results for 1000 repetitions .................... 50  
  4.3 High-dimensional experiments ......................... 53  
      4.3.1 Design .......................................... 53  
      4.3.2 Results for 100 repetitions ..................... 54  
      4.3.3 Sensitivity analysis ............................. 55  

5 Summary ................................................... 75  
  5.1 Future Work .......................................... 77  

Bibliography ............................................... 79  

Appendices ................................................. 79  

Appendix A Proofs ......................................... 81  
  A.1 Proof of Proposition 2.3.0.2 ......................... 81  

Appendix B R Code .......................................... 83  
  B.1 Functions ............................................ 83  
  B.2 Script ................................................ 104
List of Figures

2.1 Gene expression SEM ................................. 7
2.2 A simple SEM ........................................ 9
2.3 DAGs $G$ in gene expression class .................. 12
2.4 Non-identifiability ................................... 13
2.5 CPDAG of a Markov equivalence class ............... 13
2.6 Additive gene expression DAG ........................ 16

3.1 3 steps of SVA estimation ............................ 19
3.2 Reduced equation model DAG .......................... 20
3.3 examples of p-value distribution ....................... 34
3.4 estimation of $\pi_0$ ................................... 36
3.5 $\pi_0$ as a function of $\lambda$ ......................... 37
3.6 $FDR(t)$ .............................................. 38
3.7 $q^*(t)$ ............................................... 39

4.1 DAG of simulated low dimensional SEM ................ 45
4.2 Comparison of $f_{x_j}(y)$ using SVA to the real $f_{x_j}$ .... 46
4.3 Comparison of $E[x_j|y, h_{k_{sep}}]$ for the vanilla and SVA methods .... 47
4.4 A comparison of the residuals produced with the four methods, van, SVDX, SVDR and SVA ....................... 48
4.5 Tukey-Anscombe residual plot, with $\sum_{k=1}^K \beta_{kj}h_k$ on the x-axis .... 49
4.6 Evaluation of performance for low-dim sem with 1000 repetitions: $c_l$ and $h_k$ node span estimation, dependence between $h_k$ nodes and $y$ and $f_{x_j}$ estimation ................................................. 52
4.7 Evaluation of performance for high-dim sem with 100 repetitions: $c_l$ and $h_k$ node span estimation, dependence between $h_k$ nodes and $y$ and $f_{x_j}$ estimation ................................................. 54
4.8 Sensitivity of parameter $J$ ............................. 56
4.9 Sensitivity of parameters $(K, J)$ ....................... 58
4.10 Sensitivity of parameter $p_{0k}$ ......................... 59
4.11 Sensitivity of parameter $p_{0j}$ ......................... 61
4.12 Sensitivity of parameter $p_{0\beta}$ ....................... 62
4.13 Sensitivity of parameters $(p_{ij}, P_{d-sep})$ ............ 64
4.14 Sensitivity of parameter $\sigma_{c_l}$ ...................... 66
4.15 Sensitivity of parameter $\sigma_{N_{x_j}}$ .................... 68
4.16 Sensitivity of parameter $d_{\text{max}}$ .................... 70
4.17 Sensitivity of parameter $n$ ............................ 72
Chapter 1

Introduction

1.1 Structure

The report is organized as follows. In Section 1.2 we describe the problem Surrogate Variable Analysis (SVA) tries to solve in its original gene expression context.

Chapter 2 introduces the basic graphical models, structural equation models (SEMs) and causality concepts and results necessary to frame the gene expression modeling problem in the context of causality. We also define a class of structural equation models, which we call additive gene expression SEMs. This class of SEMs can be used to model gene expression and forms the basis for the SVA methodology. We conclude the section by deriving some of the properties of this class.

In Chapter 3 the SVA methodology is described in detail. Section 3.1 gives a general overview of the methodology which consists of three main steps: estimation of the span of the \( c_l \) and \( h_k \) variables, respectively, from the additive gene expression SEM, and estimation of effects \( f_{x_j} \) of \( y \) on \( x_j \). Sections 3.2 and 3.3 give a detailed description of the first two steps.

In Chapter 4 we evaluate the performance of the SVA methodology by applying it to simulated data. Section 4.1 describes other methods that can be used to estimate the gene expression SEM. In Sections 4.2 and 4.3 we will apply these methods, together with SVA, in order to provide benchmarks for the SVA methodology. Section 4.2 includes the results of low dimensional experiments, where the number of gene expression variables is less than 10. The results of high dimensional experiments, where the number of gene expression variables in the order of 1000, are included in Section 4.3. This Section also includes sensitivity analysis on certain simulation parameters to gauge the effectiveness of SVA in different data environments.

In Chapter 5 we summarize the key findings and list possible future lines of investigation.

1.2 Surrogate Variable Analysis

Gene expression is the biological process by which the information contained in a gene is used to produce material such as proteins or RNA. The degree to which this process occurs
can be measured on a continuous scale and we call this measurement the gene expression level.

It has been shown that genetic, environmental, demographic and other factors have an effect on gene expression levels. Often it is of interest to study the effect of one, or a group, of primary variables on the expression levels of a group of genes. Other unmodeled variables which also have an effect on gene expression may not be studied explicitly because measurements are not available or because it is inconvenient to do so - the relationship between the unmodeled variable and gene expression maybe too complex or limited sample size may restrict the number of variables that can be used as predictors. The following model can be used to describe this situation:

\[ x_j = f_{x_j}(y) + \sum_{l=1}^{L} \gamma_l g_l + N_{x_j} \]  \hspace{1cm} (1.2.0.1)

Where,

- \( j \in \{1, ..., J\} \),  
- \( x_1, ..., x_J \) are the gene expression levels of the \( J \) genes of interest,
- \( y \) is the primary variable (which may be a vector) of interest, the variable whose effect on the gene expression levels \( \mathbf{z} = (x_1, ..., x_J)^T \) we want to study,
- \( g_l := \sum_{p=1}^{P} g_{lp}(w_p) \) and \( w_1, ..., w_P \) are the unmodeled variables, and
- \( N_{x_j} \) is a white noise random process.

The choice of an additive model is quite general as it has been shown in ? that an appropriate choice of non-linear basis, i.e. the right choice of functions \( f_{x_j}, g_{11}, ..., g_{lp} \), can be used to accurately represent even complicated non-linear functions of the variables \( y, w_1, ..., w_p \). Notice that the above model is a multivariate regression model since there are \( J \) response variables involved.

**Example 1.2.0.1 (Disease state and age).** As a simple illustrative example take the hypothetical human expression study proposed in ?. The disease state of a certain tissue is the primary variable. Additionally, changes in expression are also influenced by the age of the individuals. The expression level of some genes depends on the disease state, the expression level of other genes depends on age and for others still, expression level depends on both variables. We are only interested in the influence of disease state on expression level. The age of the individuals corresponding to the collected samples is unknown to us. The corresponding model is:

\[ x_j = f_{x_j}(y) + \sum_{l=1}^{L} \gamma_l g_l + N_{x_j} \]  \hspace{1cm} (1.2.0.2)

Where,

- \( j \in \{1, ..., J\} \),  
- \( x_1, ..., x_J \) are the gene expression levels of the \( J \) genes of interest,
1.2 Surrogate Variable Analysis

- $y \in \{0, 1\}$ is the primary variable, disease state,
- $g_l := g_l(w)$ and $w$ is the age of the individual, and
- $N_{x_j}$ is a white noise random process.

Since we can’t observe $w_1, ..., w_p$ normal regression techniques can’t be used to estimate the model. We also know from multiple regression that if we omit the variables $w_1, ..., w_p$ from the model, i.e. if we estimate the simplified model:

$$x_j = f_{x_j}(y) + \epsilon_j \quad (1.2.0.3)$$

we will obtain a model with systematic error such that $\mathbb{E}[\epsilon_{ij}] \neq 0$. Additionally, since we are dealing with a multivariate regression model, omitting the variables $w_1, ..., w_p$ leads to correlation among residuals $\epsilon_i$ and $\epsilon_j$, for $i \neq j$ which complicates significance analysis.

To avoid this we need not estimate $g_1, ..., g_L$ individually, it suffices to estimate $\sum_{l=1}^L g_l$. We could, for example, estimate variables $h_1, ..., h_K$ as long as there exist $\beta_1, ..., \beta_K$ such that $\sum_{k=1}^K \beta_k h_k = \sum_{l=1}^L g_l$. This is what the Surrogate Variable Analysis methodology (SVA) proposed in \cite{?} tries to do. It aims to estimate surrogate variables $h_1, ..., h_K$ which generate the same linear space of the unmodeled factors $g_1, ..., g_L$ so that we can then accurately estimate $f_{x_1}, ..., f_{x_J}$ and produce valid significance analysis.

The variables $h_1, ..., h_K$ must include signal from other sources than the primary variable $y$, i.e. from certain unobserved $w_1, ..., w_p$. However, the linear space generated by $h_1, ..., h_K$ need not be orthogonal to $f_{x_j}(y)$ so the estimation of the surrogate variables must allow for potential overlap in signal with the primary variable. The model to estimate is then:

$$x_j = f_{x_j}(y) + \sum_{k=1}^K \beta_k h_k + N_{x_j} \quad (1.2.0.4)$$

Where,

- $j \in \{1, ..., J\}$,
- $x_1, ..., x_J$ are the gene expression levels of the $J$ genes of interest,
- $y$ is the primary variable (which may be a vector) of interest, the variable whose effect on the gene expression levels $x = (x_1, ..., x_J)^T$ we want to study,
- $N_{x_j}$ is a white noise random process,
- $h_k$ are surrogate variables such that $\sum_{k=1}^K \beta_k h_k = \sum_{l=1}^L g_l$, to be estimated using SVA,
- $h_k$ can’t be modelled as $h_k = f(y) + N$, where $N$ is a white noise process, i.e. it must include signal from unmodeled variables, and
- typically $\text{Cor}(f_{x_j}(y), h_k) \neq 0$ for some $k$.

There are an estimated 20,000-25,000 human protein-coding genes while typically, the cost of measuring gene expression levels means we only have available in the order of tens or
hundreds of samples with the gene expression level and primary variable measurements. This means we are in a high-dimensional setting where $J \gg n$, where $n$ is the sample size. Any estimation technique designed to estimate model 1.2.0.4 must take this into account, as does SVA.

In SVA as in techniques such as Canonica Correlation Analysis (CCA) and Reduced Rank Regression (RRR) response variables (in this case gene expression levels) are considered simultaneously. Where as in CCA and RRR the characterization of the space spanned by the response variables is used to estimate model parameters, in the case of SVA it is used to recover the effects of unmodeled variables so as to produce an analysis that essentially includes all relevant variables.
Chapter 2

Causality

In the relationship between, on the one hand, the primary variable $y$ and unmodeled variables $w_1, ..., w_p$ and, on the other, the gene expression levels $x_1, ..., x_J$ is not explicitly described as causal in nature. Nevertheless, emphasis is made on measuring the effect of the first group of variables on the second, suggesting an underlying causal structure. Additionally, the SVA methodology can be succinctly described and justified within the framework of Causality if we assume these relationships to be causal. We give some definitions and results necessary to frame the problem in this setting. The rest of this section follows closely.

2.1 DAGs and SEMs

Causal relations are defined in the context of structural equation models (SEMs). Every SEM $\mathcal{M}$ has an associated graph $\mathcal{G}$ that summarizes the functional dependencies of the SEM. As we will see each SEM induces a unique joint probability distribution $\mathbb{P}_X$ for the variables in the SEM. We will be interested in SEMs whose graph is a directed acyclic graph (DAG). In this case the graph also represents, through the d-separation relation, the independence relations of $\mathbb{P}_X$ and can be used to refute claims about causal relationships between variables.

**Definition 2.1.0.1** (Directed acyclic graph (DAG)). A directed acyclic graph (DAG) is a tuple $\mathcal{G} := (\mathcal{V}, \mathcal{E})$ where $\mathcal{V}$ is a finite set of nodes and the set of edges $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is such that if $v, w \in \mathcal{V}$ and $v \neq w$ then $(v, v) \notin \mathcal{E}$ and if $(v, w) \in \mathcal{E}$ then $(w, v) \notin \mathcal{E}$.

**Definition 2.1.0.2** (D-separation). Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a DAG, with $\mathcal{V} = \{x_1, ..., x_p\}$. Then:

i. **Notation.** We may refer to node $x_{i_k}$ or a sequence of nodes $x_{i_1}, ..., x_{i_n} \in \mathcal{V}$ by their indices, e.g. node $i_k$ or nodes $i_1, ..., i_n$. If there is an edge between node $i$ and $j$, i.e. if $(i, j) \in \mathcal{E}$, we write $i \to j$ (or $j \leftarrow i$). If $\mathcal{V}$ includes nodes labeled with different letters we don’t use the indices to refer to them as, for example, when $\mathcal{V} = \{y, h_1, ..., h_K, x_1, ..., x_J\}$
ii. **Parent.** A node \(i\) is a parent of \(j\) in DAG \(G\) if \(i \to j\). The set of parents of node \(j\) is denoted \(PA_j\).

iii. **Path.** There is a path between nodes \(i_1\) and \(i_n\) if there exists a sequence of nodes \(i_1, i_2, ..., i_n \in V\) such that either \(i_k \to i_{k+1}\) or \(i_{k+1} \to i_k\) for \(k = 1, ..., n - 1\).

iv. **Directed path.** There is a directed path between nodes \(i_1\) and \(i_n\) if there exists a sequence of nodes \(i_1, i_2, ..., i_n \in V\) such that \(i_k \to i_{k+1}\) for \(k = 1, ..., n - 1\).

v. **Descendent.** We say node \(i_k\) is a descendent of node \(i_j\) if there is a directed path from \(i_j\) to \(i_k\). The sets of descendent and non-descendant nodes to node \(i\) in DAG \(G\) are denoted \(DE_i\) and \(ND_i\) respectively.

vi. **Collider.** A node \(i_k\) is a collider in a path \(i_1, ..., i_n\) if \(i_{k-1} \to i_k \leftarrow i_{k+1}\). The structure \(i_{k-1} \to i_k \leftarrow i_{k+1}\) is referred to as a \(v\)-structure.

vii. **Blocking set.** The set \(S\) blocks the path from \(i_1\) to \(i_n\) if either:

- \(i_k \in S \land i_{k-1} \to i_k \leftrightarrow i_{k+1} \land i_{k-1} \leftarrow i_k \to i_{k+1}\), or
- \(i_k \notin S, DE_{ik} \cap S = \emptyset \land i_{k-1} \to i_k \leftarrow i_{k+1}\) (i.e. \(i_k\) is a collider).

viii. **D-separation.** Let \(A, B, S \subseteq V\) disjoint. \(A\) is d-separated from \(B\) by \(S\), if \(S\) blocks all paths between nodes in \(A\) and nodes in \(B\). We denote this as \(\text{Ad-sep}_G B\) by \(S\).

ix. **D-connection.** We say that \(A\) is d-connected from \(B\) by \(S\), if \(A\) is not d-separated from \(B\) by \(S\). We denote this as \(\text{Ad-conn}_G B\) by \(S\).

**Definition 2.1.0.3 (Structural equation model (SEM)).** A structural equation model (SEM) for variables \(X = \{x_1, ..., x_p\}\) is defined as a tuple \(M := (S, P^N)\) where \(S = \{S_1, ..., S_p\}\), a collection of \(p\) equations, and \(P^N\), a joint probability distribution, are such that:

\[
S_j : x_j = f_j(x_{PA_j}, N_j) \quad \text{(2.1.0.1)}
\]

\[
P^N = P^{N_1}...P^{N_p} = P^{N_1}...P^{N_p} \quad \text{(2.1.0.2)}
\]

Where,

- \(j \in \{1, ..., p\}\),
- \(x_{PA_j}\) is the set of variables upon which \(x_j\) functionally depends, where \(PA_j \subseteq \{1, ..., p\}\) denotes the indices of those variables; we require the functions \(f\) to really depend on the set of parent variables \(x_{PA_j}\), i.e. for all \(x_i \in x_{PA_j}\) and for all values of \(x_{PA_j \backslash i}\) and \(N_j\) there must exist values \(x^1_i\) and \(x^2_i\) such that \(f_j(x^1_i, x_{PA_j \backslash i}, N_j) \neq f_j(x^2_i, x_{PA_j \backslash i}, N_j)\). If this is not the case we may simply redefine \(x_{PA_j} \leftarrow x_{PA_j \backslash i}\),
- \(N_j\) are random noise variables, and
- \(P^N\) is the joint distribution of noise variables, which we require to be jointly independent.

**Definition 2.1.0.4 (Graph of a SEM).** The graph of a SEM for variables \(X = \{x_1, ..., x_p\}\) is \(G = (V, E)\) where \(V = \{1, ..., p\}\) and \(i \to j\) if and only if \(i \in PA_j\). If the resulting graph \(G\) is a DAG then \(PA^G_j = PA_j \forall j\). From now on we only consider SEMs that induce a DAG graph.
Example 2.1.0.5 (A gene expression SEM). As an example of a SEM consider $\mathcal{M} = (S, P)$ where $X = \{y, h_1, ..., h_K, x_1, ..., x_J\}$ and the joint noise distribution is a multivariate normal composed of independent standard normal variables $N \sim N_p(0, I_p)$ with $p = K + J + 1, j \in \{1, ..., J\}, k \in \{1, ..., K\}$ and $S$ is such that:

$$
S_y : y = N_y \tag{2.1.0.3}
$$

$$
S_{h_k} : h_k = f_{h_k}(y, N_{h_k}) \tag{2.1.0.4}
$$

$$
S_{x_j} : x_j = f_{x_j}(y, h_1, ..., h_k, N_x) \tag{2.1.0.5}
$$

If $J = 3$ and $K = 2$ the DAG $G$ corresponding to this SEM is:

![Figure 2.1: Gene expression SEM](image)

Proposition 2.1.0.6 (Distribution of a SEM). A SEM for variables $X = \{x_1, ..., x_p\}$, with an acyclic structure (i.e. a DAG graph), defines a unique distribution $P^X$ over these variables.

**Proof.** See Proposition 2.1.2 in ?.

This means that by specifying the SEM we implicitly specify the probability distribution $P^X$. However, as we will see the SEM also includes additional information that allows us to determine which dependence relations between the variables in $X$ are causal (and incidentally, other information that allows us to make counterfactual statements).

We have seen that a SEM $\mathcal{M}$ induces a unique graph $G$ and joint probability distribution $P^X$. We now study the relationship between the DAG of a SEM and its joint probability distribution to see to what degree the DAG structure accurately describes the independence relations encoded in the joint probability distribution.

Definition 2.1.0.7 (Markov properties). We say that the joint distribution $P^X$ (which we require to have a density $p$ with respect to some product measure) has the following Markov properties with respect to the DAG $G$ over variables $X$ if it satisfies the corresponding conditions:

i. the global Markov property if $A \text{ d-sep}_G B \mid C \Rightarrow A \indep B \mid C$ where $A, B, C \in V$ are disjoint.
ii. the local Markov property if \( x_j \perp \! \! \! \perp_{D, G} P_{A_j} \forall j \in \mathcal{V} \).

iii. the Markov factorization property if \( p(\mathcal{X}) = \prod_{j=1}^{p} p(x_j|x_{PA_j}) \) in which case we say that \( P^X \) factorizes over \( G \).

Notice that the global Markov property implies that:

\[
A \text{ d-conn}_G B \text{ by } C \iff A \not\perp\!\!\!\!\perp B | C \tag{2.1.0.6}
\]

**Proposition 2.1.0.8** (Equivalence of Markov properties). If \( P^X \) has a density \( p \) (with respect to a product measure), then all Markov properties of 2.1.0.7 are equivalent.

*Proof.* See Theorem 3.27 in ?.

Let \( \mathcal{I}(G) \) denote the set of all independencies implied by the d-separations in \( G \) when we equate \( A \text{ d-sep}_G B \text{ by } C \) to \( A \not\perp\!\!\!\!\perp B | C \). Then \( G \) is an I-map for a set of independencies \( \mathcal{I} \) if:

\[
\mathcal{I}(G) \subseteq \mathcal{I} \tag{2.1.0.7}
\]

A DAG \( G \) is said to be an I-map for a joint probability distribution \( P^X \) if and only if \( P^X \) is Markov with respect to \( G \). This is represented as:

\[
\mathcal{I}(G) \subseteq \mathcal{I}(P^X) \tag{2.1.0.8}
\]

Where:

- \( \mathcal{I}(P^X) \) denotes all independencies in \( P^X \).

So if \( P^X \) is Markov with respect to a DAG \( G \) it means d-separation on \( G \) is a sound procedure for obtaining the independencies in \( P^X \): we can read-off independencies from the graph.

**Proposition 2.1.0.9** (The joint distribution of a SEM is Markov). If \( P^X \) and \( G \) are induced by a SEM \( M \) then \( P^X \) is Markov with respect to \( G \).

*Proof.* See Theorem 1.4.1 in ?.

This means that the DAG \( G \) of any SEM \( M \) is an I-map for its joint distribution \( P^X \). Using that figure 2.1 is an I-map for the SEM of example 2.1.0.5 we can now deduce, for example, that \( y \perp h_1 | x_1, x_2, x_3 \) and \( x_2 \perp x_3 | h_1, h_2, y \). The faithfulness property of a DAG \( G \) will allow us to check if d-separation is a complete procedure for finding the independencies of \( P^X \).
Definition 2.1.0.10 (Faithfulness). We say that the joint probability distribution $P^X$ is faithful with respect to DAG $G$ over variables $X$ if it satisfies the following condition:

$$A \text{ d-sep}_G B \text{ by } C \iff A \perp \perp B | C$$  \hspace{1cm} (2.1.0.9)

Notice that the faithfulness property implies that:

$$A \text{ d-conn}_G B \text{ by } C \Rightarrow A \nmid \nmid B | C$$  \hspace{1cm} (2.1.0.10)

If a joint distribution $P^X$ is faithful with respect to DAG $G$ we write:

$$I(G) \supseteq I(P^X)$$  \hspace{1cm} (2.1.0.11)

This means that if a joint distribution $P^X$ is faithful with respect to DAG $G$ we are able to read-off dependencies from the graph. If a joint distribution $P^X$ is Markov and faithful with respect to a DAG $G$ we say $G$ is a perfect I-map for $P^X$ and write:

$$I(G) = I(P^X)$$  \hspace{1cm} (2.1.0.12)

In this case d-separation on $G$ is a sound and complete procedure for obtaining the independencies in $P^X$ and we can read-off dependencies and independencies from the graph. Unfortunately, neither the fact that a joint distribution $P^X$ factorizes over $G$ nor the fact that a DAG $G$ and a joint distribution $P^X$ are induced by the same SEM $M$ guarantees that $P^X$ is faithful with respect $G$. We can verify this using a simplified version of example 2.1.0.5:

Example 2.1.0.11 (A simple SEM). Consider the SEMs $M = (S, P^N)$ where $X = \{y, h, x\}$, the joint noise distribution is a multivariate normal composed of independent standard normal variables $N \sim N_3(0, I_3)$ and $S$ is such that:

$$S_y : y = N_y$$  \hspace{1cm} (2.1.0.13)

$$S_h : h = f_h(y, N_h) = ay + N_h$$  \hspace{1cm} (2.1.0.14)

$$S_x : x = f_x(y, h, N_x) = by + ch + N_x$$  \hspace{1cm} (2.1.0.15)

The DAG $G$ corresponding to this SEM is:

![Figure 2.2: A simple SEM](image-url)
First notice that \( X \sim N_3(0, \Sigma) \) where the covariance matrix \( \Sigma \) depends on \( a, b \) and \( c \). We may reexpress the equation \( S_x \) as:

\[
x = by + ch + N_x = bN_y + c(aN_y + Nh) = (b + ca)N_y + cNh
\]  \tag{2.1.0.16}

From which we can calculate the covariance between \( y \) and \( x \):

\[
\text{Cov}(y, x) = \text{Cov}(N_y, (b + ca)N_y + cNh) = (b + ca)\text{Var}(N_y) = b + ca
\]  \tag{2.1.0.17}

Where we have used the bi-linear properties of the covariance and the fact that the noise variables \( N_y \) and \( Nh \) are independent. Now if \( a, b \) and \( c \) are such that \( b + ca = 0 \), \( y \) and \( x \) are not correlated and since \( X \) has a multivariate normal distribution this means that \( y \) is independent of \( x \). This in turn implies that if \( a, b \) and \( c \) are such that \( b + ca = 0 \) then \( \mathcal{G} \) is not faithful to \( P_X \).

The above SEM can actually be considered to be a class of SEMs parametrized by \( a, b \) and \( c \). Although for many classes of SEMs, such as the one above, faithfulness cannot be guaranteed, a somewhat weaker condition called causal minimality can sometimes be guaranteed.

**Definition 2.1.0.12** (Causal minimality). We say that a joint probability distribution \( P_X \) satisfies causal minimality with respect to the DAG \( \mathcal{G} \) over variables \( X \) if it is Markov with respect to \( \mathcal{G} \), but not with respect to \( \mathcal{G}' \), where \( \mathcal{G}' \) is a proper subgraph of \( \mathcal{G} \).

We say that a DAG \( \mathcal{G} \) is a minimal I-map of \( P_X \) if the following two conditions hold:

\[
\mathcal{I}(\mathcal{G}) \subseteq \mathcal{I}(P_X) \tag{2.1.0.18}
\]

\[
\mathcal{I}(\mathcal{G}') \not\subseteq \mathcal{I}(P_X) \tag{2.1.0.19}
\]

where \( \mathcal{G}' \) is a proper subgraph of \( \mathcal{G} \). The idea is that, if \( \mathcal{G} \) satisfies causal minimality, while a d-connection in the graph does not imply the corresponding dependency, if we take off any of the edges of DAG \( \mathcal{G} \), to obtain the proper subgraph \( \mathcal{G}' \), then we create new independencies which are not in \( P_X \) i.e. all the edges in \( \mathcal{G} \) contribute to the Markov property of \( P_X \) with respect to \( \mathcal{G} \). We now present a result that will help to discern if the joint distribution of a given SEM satisfies causal minimality with respect to its DAG.

**Proposition 2.1.0.13** (Causal minimality condition for SEMs). Let the joint distribution \( P_X \) be Markov with respect to a DAG \( \mathcal{G} \) over variables \( X = \{y, x_1, ..., x_p\} \) as is the case with the distribution and graph of a SEM. Assume the joint distribution \( P_X \) has a density with respect to some product measure. Then:

\( P_X \) satisfies causal minimality with respect to \( \mathcal{G} \) \iff \( \forall x_j \) and \( \forall y \in PA^\mathcal{G}_j \ x_j \nleftrightarrow y \mid PA^\mathcal{G}_j \setminus \{y\} \)

**Proof.** See Appendix A.2.5 in ?.
2.1 DAGs and SEMs

We now define general a class of SEMs which we assume can be used to accurately model the gene expression causal process. We don’t assume this class of SEMs represents the actual data generating process as we know this is not likely to be additive, for example.

**Definition 2.1.0.14 (Class of gene expression SEMs).** We define the class of gene expression SEMs as the collection of SEMs for variables \( X \cup A = \{y, h_1, ..., h_K, x_1, ..., x_J\} \cup A \) such that:

i.) \( S_{x_j} : x_j = f_{x_j}(y) + \sum_{k=1}^{K} \beta_{kj} h_k + N_{x_j} \forall j \in \{1, ..., J\} \)

ii.) \( P_X \) is such that \( y \perp h_k \forall k \in \{1, ..., K\} \)

iii.) \( A \) is the set of variables that are not parents of \( x_1, ..., x_J \) but form part of the SEM.

The variables in \( A \) represent the part of the SEM which we are not so interested in but which we might need to estimate to understand the rest of the SEM. Specifically we are interested in estimating the functions \( f_{x_j} \), which represent the the direct effect that \( y \) has on \( x_j \). In this work we have assumed that the gene expression level \( x_j \) does not depend on any other expression level \( x_k \) with \( k \in \{1, ..., J\} \setminus j \), however this is not implicit in the SVA methodology. It could be that a more appropriate SEM model for the \( x_j \) variables has the equations:

\[
S_{x_j} : x_j = f_j(x_{\setminus j}) + f_{x_j}(y) + \sum_{k=1}^{K} \beta_{kj} h_k + N_{x_j} \quad (2.1.0.20)
\]

\( \forall j \in \{1, ..., J\} \) where \( x_{\setminus j} \) is a vector which includes all \( x_k \) variables such that \( k \in \{1, ..., J\} \setminus j \). In this case the SVA methodology still works but does not estimate the direct effect of \( y \) on \( x_j \) rather the filtered effect of \( y \) on \( x_j \): the effect of \( y \) that passes through the unobserved variables \( h_k \) has been filtered out.

Notice that since the joint probability distributions \( P_X \) of all SEMs are Markov with respect their DAGs \( G \), if \( A \) d-conn \( G \) by \( C \) then \( A \nmid B \mid C \): we can satisfy condition 2 above by assuring that \( y \) d-conn \( h_k \) by \( \emptyset \). We give some examples of DAGs that correspond to SEMs belonging to class 2.1.0.14.
Notice that in all the examples it holds that \( y \text{d-conn}\_G h_k \) by \( \emptyset \), so for a SEM with any of these DAGs, \( \mathcal{G} \) is an I-map of \( \mathbb{P}^X \). The above class of SEMs is very large: the number of nodes is not parametrized and the functional form of the equations \( S_{h_k} \) is not determined. This leads to an identifiability problem which we discuss here.

Before we can estimate the parameters or functions of a SEM we must specify its structure, which means specifying the arguments of each equation in \( \mathcal{S} \) or equivalently estimating the induced graph \( \mathcal{G} \).

One way to approach the problem is not to assume any specific form of SEM, first learn the graph structure and then model each equation using observations from \( \mathbb{P}^X \). The way we can learn the graph structure is by performing some form of independence test. Assume we have an infinite number of observations from \( \mathbb{P}^X \), then we can estimate the right set of independencies \( \mathcal{I}(\mathbb{P}^X) \) every time, which corresponds to having an independence oracle. Even in this situation, identifying most graph structures is impossible because different graphs can represent the same set of independencies: there exist \( \mathbb{P}^X \) and \( \mathcal{G}_1 \neq \mathcal{G}_2 \) such that \( \mathcal{I}(\mathcal{G}_1) \subseteq \mathcal{I}(\mathbb{P}^X) \) and \( \mathcal{I}(\mathcal{G}_2) \subseteq \mathcal{I}(\mathbb{P}^X) \).

**Example 2.1.0.15** (Non-identifiability). As an example consider the variables \( x, y \) and \( z \) and suppose we want to find the DAG that is an I-map for \( \mathcal{I} = \{ x \perp \! \! \! \! \perp y | z \} \). Even for such a simple example there are 3 graphs that are I-maps for \( \mathcal{I} \):
2.2 Causality

Figure 2.4: Non-identifiability

Something similar occurs in the example of figure 2.3 where the four DAGs are I-maps for sets of dependencies all of which include $y \not\perp h_k$. This means that a SEM with any of the four DAGs can induce a $\mathbb{P}^X$ where $y \not\perp h_k$.

**Definition 2.1.0.16** (Markov equivalence class). The Markov equivalence class of DAGs, $\mathcal{K}$, for a set of independencies $\mathcal{I}$ is the set of DAGs $\mathcal{G}$ over variables $X$, such that $\mathcal{I}(\mathcal{G}) = \mathcal{I}$.

The Markov equivalence class of a DAGs, $\mathcal{K}$, can be represented using a completed directed acyclic graph (CPDAG) where:

1. The set of nodes of the graph is $X$,
2. Directed edges correspond to nodes $i, j$ which in all DAGs $\mathcal{G} \in \mathcal{K}$ are joined either with a $i \rightarrow j$ edge or a $i \leftarrow j$ edge, and
3. Undirected edges correspond to nodes $i, j$ which in some DAGs $\mathcal{G} \in \mathcal{K}$ are joined with a $i \rightarrow j$ edge and in others with $ai \leftarrow j$ edge.

**Example 2.1.0.17** (CPDAG of a Markov equivalence class). The CPDAG representing the Markov equivalence class which contains the three DAGs shown in figure is:

![Figure 2.5: CPDAG of a Markov equivalence class](image)

We can obtain all the DAGs $\mathcal{G}$ in Markov equivalence class $\mathcal{K}$ by directing undirected edges in all possible ways that don’t create cycles in the graph.

2.2 Causality

Using the observational distribution $\mathbb{P}^X$ of an unknown SEM $\mathcal{M}$ we can only identify the correct DAG $\mathcal{G}$ up to its Markov equivalence class. This is why we deal with DAGs in the context of SEMs, instead of as models in themselves, because they allow us to define *interventional distributions*. Through intervention distributions we are able to define causal relationships. This gives us another tool in establishing the directionality of edges and thus the possibility of identifying the correct DAG structure. We don’t always have access
to observations from the interventional distribution - in fact SVA works on the assumption that we don’t - however sometimes we can express the interventional distribution \( P_{X}^{\tilde{M}} \) in terms of the observational distribution \( P_{X}^{M} \) (for example through instrumental variables or by adjusting) meaning we can establish causal relationships.

A second approach to establishing the causal structure is to assume that the SEM belongs to a more restricted class. Depending on the class of SEM assumed there are results available establishing whether the identifiability of the DAG is possible or not, and techniques for actually finding it (see Chapter 4 in ? for an overview). The approach of SVA is more akin to this second approach. In fact, the assumptions that will be made about the underlying SEM in definition 2.3.0.1 almost completely determine the DAG structure.

We now study intervention distributions and causality so that we may give a causal interpretation of SVA. We are also interested in defining causal relationships so as to verify that the proposed SEM classes satisfy our assumption about the causal nature of the relationship between primary and unmodeled variables and the gene expression level variables.

**Definition 2.2.0.1 (Intervention distribution).** Consider a distribution \( P_{X}^{M} \) induced by a SEM \( M = (S, P^{N}) \). If we replace one (or more) of the equations \( S_{i} \in S \) we obtain a new SEM \( \tilde{M} \), a new joint distribution \( P_{X}^{\tilde{M}} \), and new marginals \( P_{X_{i}}^{\tilde{M}} \) which we call intervention distributions and denote:

\[
P_{X_{i}}^{\tilde{M}} = P_{X_{i}}^{M | do(x_{j} = \tilde{f}(\tilde{PA}_{j}, \tilde{N}_{j}))}
\]

Where:

- Some noise variables have been replaced but \((\tilde{N}_{1}, ..., \tilde{N}_{p})\) is still mutually independent,

- \( \tilde{PA}_{j} \) can be any new set of nodes in \( V \) as long as there are no cycles in \( \tilde{G} \) but usually \( \tilde{PA}_{j} = \{\} \) (\( x_{j} \) is deterministic or just noise) or \( PA_{j} = PA_{j} \) (the DAG \( G \) stays the same but we change the noise or the function \( f \))

- When \( \tilde{f}(\tilde{PA}_{j}, \tilde{N}_{j}) \) puts a point mass on a real value \( b \) we simply write \( P_{X_{i}}^{M | do(x_{j} = b)} \)

If we already know the SEM \( M \) and the induced distribution \( P_{X}^{M} \) we can obtain the interventional distribution \( P_{X_{i}}^{\tilde{M}} \) from the above definition. If we don’t know the SEM, are learning the DAG structure, and have access to data generating mechanism, we can perform interventions. This allows us to obtain observations from the corresponding interventional distributions and obtain the independence relations therein by performing independence tests. If we know the set of independencies in the interventional distribution, \( I(P_{X_{i}}^{\tilde{M}}) \), we can establishing causal relations between variables through definition 2.2.0.2 and proposition 2.2.0.3.

**Definition 2.2.0.2 (Total causal effect).** Given a SEM \( M \) there is a (total) causal effect from \( x \) to \( y \) if and only if \( x \perp \not\perp y \) in \( P_{X_{i}}^{M | do(x = \tilde{N}_{x})} \) for some variable \( \tilde{N}_{x} \).
2.3 Additive Gene Expression SEMs

Proposition 2.2.0.3 (Equivalent causality queries). Given a SEM \( \mathcal{M} \), the following are equivalent:

i.) There is a causal effect from \( x \) to \( y \).

ii.) \( \exists x_1, x_2 : x_1 \neq x_2 \) such that \( \mathbb{P}_{\mathcal{M}}[y|do(x=x_1)] \neq \mathbb{P}_{\mathcal{M}}[y|do(x=x_2)] \).

iii.) \( \exists x_1 \) such that \( \mathbb{P}_{\mathcal{M}}[y|do(x=x_1)] \neq \mathbb{P}_{\mathcal{M}}[y] \).

iv.) \( x \perp \!
\!\!\!\perp y \) in \( \mathbb{P}_{\mathcal{M}}[X|\text{do}(x=\tilde{N}_x)] \) for all \( \tilde{N}_x \) such that \( \tilde{p}(x) > 0 \) \( \forall x \). i.e. the pdf of \( \tilde{N}_x \) has full support.

Proof. See Appendix A.2.1 in ?.

Assume we have learned the Markov equivalence class \( \mathcal{K} \) of DAGs that are I-maps to \( \mathbb{P}^X \). The following result can help us to identify the DAG structure, from within \( \mathcal{K} \), by establishing the directionality of undirected edges in the CPDAG.

Proposition 2.2.0.4 (Directed paths and causality). The following statements relate the DAG \( \mathcal{G} \) of a SEM \( \mathcal{M} \) to whether the relationships between its variables are causal or not.

i. If there is no directed path from \( x \) to \( y \), then there is no causal effect.

ii. If there is a directed path from \( x \) to \( y \), there may not be a causal effect.

Proof. See Appendix A.2.2 in ?.

Notice that an equivalent form of statement i. is that if there is a causal effect from \( y \) to \( x \) then there must be a directed path from \( y \) to \( x \). Suppose we have identified the DAG \( \mathcal{G} \) up to the CPDAG that represents \( \mathcal{K} \) using observations from \( \mathbb{P}^X \). Using observations from \( \mathbb{P}_{\mathcal{M}}^X \) we can establish if variable \( y \) causes variable \( x \). If \( y \) causes \( x \) we can then use the above proposition to direct undirected edges by making sure there is at least one directed path from \( y \) to \( x \) in \( \mathcal{G} \).

2.3 Additive Gene Expression SEMs

We now define a class of gene expression SEMs that is more restricted than the one defined in 2.1.0.14. Below we argue that we may use this SEM to model the variables \( X = \{y, h_1, ..., h_K, x_1, ..., x_J\} \) since the distribution \( \mathbb{P}^X \) that both SEMs induce is essentially the same: in this case we use non-identifiability in our favour to model a set of variables with the simpler SEM.

Definition 2.3.0.1 (Class of additive gene expression SEMs). We define the class of additive gene expression SEMs as the collection of SEMs for variables \( X \cup C = \{y, c_1, ..., c_L, h_1, ..., h_K, x_1, ..., x_J\} \) such that:

i.) \( S_{h_k} : h_k = f_{h_k}(y) + \sum_{l=1}^{L} \gamma_l c_l + N_{h_k} \forall k \in \{1, ..., K\} \)
ii.) \( S_{x_j} : x_j = f_{x_j}(y) + \sum_{k=1}^{K} \beta_{kj} h_k + N_{x_j} \forall j \in \{1, \ldots, J\} \)

**Remark:** Since we really care about modeling \( x_j \) and the term \( f_{x_j}(y) \) can be made to include any constant term, we assume without loss of generality that

- \( \mathbb{E}[c_l] = \mathbb{E}[f_{h_k}(y)] = \mathbb{E}[h_k] = \mathbb{E}[N_{h_k}] = \mathbb{E}[N_{x_j}] = 0 \) and
- \( \mathbb{V}[c_l] = \mathbb{V}[h_k] = 1 \)

Clearly the class of additive gene expression SEMs is a subset of the class of gene expression SEMs of definition 2.1.0.14 since \( h_k \) depends on \( y \). Since we assume that the larger class of gene expression SEMs can be used to accurately model the gene expression causal process why can we use this reduced class to model the data? We again use the justification, from \( ? \), proposed for the use of model 2.3.0.1 for modeling gene expression data. Since \( y \) and \( h_k \) must be dependent by condition ii.) of Definition 2.1.0.14 we know that for the right choice of non-linear basis \( \{c_l\}_{l=1,...,L} \) we can model \( h_k \) as:

\[
h_k = f_{h_k}(y) + \sum_{l=1}^{L} \gamma_l c_l + N_{h_k} \quad (2.3.0.1)
\]

Where,

- \( k \in \{1, \ldots, K\} \),
- \( c_l := \sum_{p=1}^{P} c_{lp}(z_p) \) and \( z_1, \ldots, z_P \) are unmodeled variables upon which \( h_k \) also depends, and
- \( N_{h_k} \) is a white noise random process.

This means we can replace condition ii.) in 2.1.0.14 with equation \( S_{h_k} \) of Definition 2.1.0.14.

The DAG corresponding to a SEM from the above class where \( L = 3 \), \( K = 2 \) and \( J = 3 \), and where \( \gamma_{lk}, \beta_{kj} > 0 \forall l, k, j \) is:

![Additive gene expression DAG](image)

**Figure 2.6:** Additive gene expression DAG

The \( f_{h_k}, f_{x_j}, \gamma_{lk} \) and \( \beta_{kj} \) edge labels do not form part of the DAG and are included to represent the SEM graphically.
Notice that the DAG $G$ of any SEM in the class above will be the same as that shown in the figure up to choice of $L$, $K$ and $J$, and deletion of edges (in the case that $\gamma_{lk} = 0$ or $\beta_{kj} = 0$ for any given $l$, $k$ and $j$, or that any of the $f_{xj}$ or $f_{hk}$ functions are constant in $y$). This means that we have essentially identified the causal structure of the underlying SEM by assuming it belongs to the above class.

Also notice that if we use an alternative definition for a SEM, where the joint noise distribution is not necessarily mutually independent then we can define a class of additive gene expression SEMs without the need for the $c_l$ variables, and instead allow the $N_{hk}$ variables to be mutually dependent.

Using the definitions and results presented in this Section we can establish the following properties for the additive gene expression class.

**Proposition 2.3.0.2** (Properties of class of additive gene expression SEMs). Suppose the SEM $M$ belongs to the class of additive gene expression SEMs and that $G$ and $P_{X \cup C}$ are its induced DAG and joint distribution, respectively. Then:

1. $y \perp \perp c_l$ and $c_l \perp \perp c_m$ for all $l, m \in \{1, \ldots, L\}$.
2. $P_{X \cup C}$ satisfies causal minimality with respect to $G$.
3. If $\beta_{kj} \neq 0$ then $h_k$ is a cause of $x_j$.
4. If $f_{xj}(y) + \sum_{k=1}^{K} \beta_{kj} f_{hk}(y) \neq 0$ then $y$ is a cause of $x_j$.

*Proof.* See Appendix A. \qed

As we can see the class of additive gene expression SEMs satisfy our assumption that $y$ and $h_k$ are causes of $x_j$. The causal effect of $y$ on $x_j$ can be described in terms of the interventional distribution $P_{M,do(y=y_1)}(x_j)$. Although we have assumed that in the context of SVA we don’t have access to observations from this distribution, we can express the interventional distribution in terms of the observational distribution if we can find a valid adjustment set:

$$p_{M,do(y=y_1)}(x_j) = \int_Z p_{M}(x_j|y,Z)p_{M}(Z)dZ$$  \hspace{1cm} (2.3.0.2)

where $Z$ is a valid adjustment set of variables. Although we don’t study why adjustment works or what is a criterion for a set of variables constituting a valid adjustment set, we mention that the $c_l$ variables in the additive gene expression SEMs constitute a valid adjustment set for determining $p_{M,do(y=y_1)}(x_j)$. In Section 3.1.2 of \cite{???} the adjustment principle is explained and valid adjustment criterions derived. Using the adjustment set $Z = \{c_1, \ldots, c_L\}$ we can obtain $p_{M,do(y=y_1)}(x_j)$ from the observational distributions $p_{M}(x_j|y,c_1,\ldots,c_L)$ and $p_{M}(c_1,\ldots,c_L)$. This means we don’t need to estimate the $h_k$ variables to obtain the causal effect of $y$ on $x_j$. However, as we mentioned following Definition 2.1.0.14 we are only interested in the part of the effect that $y$ has on $x_j$ which does not go through the unmodeled factors $h_k$. For this we must first estimate the variables $h_k$ so that we may filter out the part of the effect that $y$ has on $x_j$ that goes through $h_k$. The SVA methodology developed by \cite{???} provides a way to estimate the variables $h_k$. 

Chapter 3

SVA Methodology

3.1 Overview

Given \( n \) i.i.d. observations of \( y \) and \( x_j \) for \( j \in \{1, \ldots, J\} \) the SVA methodology uses the model class 2.3.0.1 to fit the data. Although the goal is to estimate the effects \( f_{x_j} \) of \( y \) on \( x_j \), to do this the variables \( \{h_k\}_{k=1,\ldots,K} \), or rather their span, must be estimated so that their effect on \( y \) can be filtered out. To estimate the span of \( \{h_k\}_{k=1,\ldots,K} \) it is necessary to first estimate the span of \( \{c_l\}_{l=1,\ldots,L} \) since the \( c_l \) variables are causes of the \( h_k \) variables.

The basic steps of the SVA methodology are thus:

1. Estimate the span of \( \{c_l\}_{l=1,\ldots,L} \),
2. Estimate the span of \( \{h_k\}_{k=1,\ldots,K} \), and
3. Fit \( S_{x_j} \) equations from 2.3.0.1 to estimate \( f_{x_j} \).

![Figure 3.1: 3 steps of SVA estimation](image)

3.2 Estimation of \( \text{span}(\{c_l\}) \)

3.2.1 Estimation procedure

If we combine equations \( S_{x_j} \) and \( S_{h_k} \) from 2.3.0.1 we have
$x_j = f_{x_j}(y) + \sum_{k=1}^{K} \beta_{kj} h_k + N_{x_j}$ (3.2.1.1)

$x_j = f_{x_j}(y) + \sum_{k=1}^{K} \beta_{kj} (f_{h_k}(y)) + \sum_{l=1}^{L} \gamma_{lk} c_l + N_{h_k} + N_{x_j}$ (3.2.1.2)

$x_j = f_{x_j}(y) + \sum_{k=1}^{K} \beta_{kj} (f_{h_k}(y)) + \sum_{k=1}^{K} \beta_{kj} \sum_{l=1}^{L} \gamma_{lk} c_l + \sum_{k=1}^{K} \beta_{kj} N_{h_k} + N_{x_j}$ (3.2.1.3)

$x_j = (f_{x_j}(y) + \sum_{k=1}^{K} \beta_{kj} f_{h_k}(y)) + \sum_{l=1}^{L} c_l (\sum_{k=1}^{K} \beta_{kj} \gamma_{lk}) + (\sum_{k=1}^{K} \beta_{kj} N_{h_k} + N_{x_j})$ (3.2.1.4)

$x_j = f_j(y) + \sum_{l=1}^{L} \alpha_{lj} c_l + N_j$ (3.2.1.5)

Where,

- $f_j(y) := f_{x_j}(y) + \sum_{k=1}^{K} \beta_{kj} f_{h_k}(y)$,
- $\alpha_{lj} := \sum_{k=1}^{K} \beta_{kj} \gamma_{lk}$, and
- $N_j := \sum_{k=1}^{K} \beta_{kj} N_{h_k} + N_{x_j}$

The last expression suggests a reduced equation model class for variables $y$, $c_k$ and $x_j$ with the following set of equations:

$S_{x_j} : x_j = f_j(y) + \sum_{l=1}^{L} \alpha_{lj} c_l + N_j$ (3.2.1.6)

and with the following associated DAG

Notice that since $N_j = \sum_{k=1}^{K} \beta_{kj} N_{h_k} + N_{x_j}$ the noise variables $N_j$ are not independent which means the above equation model is not a SEM as defined in 2.1.0.3, and we have,
with the theory shown here, no guarantee that the above DAG is an I-map of \( P^K \). However for the same noise realizations of \( N_y \) and \( N_{ck} \), and for the above definition of the \( \alpha_{lj} \) parameters, the above derivation shows that the reduced model would generate exactly the same realizations of \( x_j \). This means that, as in the additive gene expression SEMs, \( y \perp c_l \) for all \( l \in \{1,\ldots,L\} \) which implies that for the model:

\[
x_j = f_j(y) + \sum_{l=1}^{L} \alpha_{lj}c_l + N_j
\]  

we can estimate \( f_j \) separately from \( \alpha_{lj} \) by fitting the model:

\[
x_j = f_j(y) + \epsilon_j
\]  

and we can expect to have an equivalent estimate for \( f_j \), in terms of bias and variance, by fitting model 3.2.1.8 as if we had fit model 3.2.1.7. If we define the residuals \( r_j \) of model 3.2.1.7 as:

\[
r_j := x_j - f_j(y) = \sum_{l=1}^{L} \alpha_{lj}c_l + N_j
\]  

or in vector form

\[
r := x - f(y) = A^T c + N
\]  

where

- \( r = (r_1, \ldots, r_J)^T \),
- \( x = (x_1, \ldots, x_J)^T \),
- \( f(y) = (f_1(y), \ldots, f_J(y))^T \)
- \( N = (N_1, \ldots, N_J)^T \),
- \( c = (c_1, \ldots, c_L)^T \), and
- \( A \in \mathbb{R}^{L \times J} \) with \( A_{lj} = \alpha_{lj} \).

then we can estimate the span of \( \{c_l\}_{l=1}^{L} \) by factorizing the data matrix \( R \):

\[
R := X - F
\]  

where

- \( X \in \mathbb{R}^{n \times J} \) and \( X_{ij} \) is the i-th observation of variable \( x_j \),
• $Y \in \mathbb{R}^n$ and $Y_i$ is the $i$-th observation of variable $y$,
• $F \in \mathbb{R}^{n \times J}$, $F_{ij}$ is the $i$-th observation of $\hat{f}_j(y)$, that is $\hat{f}_j$ applied to $Y_i$, where $\hat{f}_j$ is the estimation of $f_j$ obtained by fitting model 3.2.1.8.

In Section 3.2.2 we explore the way the functions $f_j$ are estimated in the SVA methodology of ? . Once the matrix $F$ and then $R$ are obtained we factorize $R$ as:

$$R = CA + E$$ (3.2.1.12)

where
• $C \in \mathbb{R}^{n \times L}$ and $C_{il}$ is the estimate for $i$-th unobserved realization of variable $c_l$,
• $\Lambda \in \mathbb{R}^{L \times J}$ is the estimate of $A$ from 3.2.1.10 and
• $E \in \mathbb{R}^{n \times J}$ and $E_{ij}$ is the estimate for the $i$-th unobserved realization of variable $N_j$.

The chosen factorization should reflect the fact that the set of $c_l$ and $N_j$ variables are mutually independent.

In SVA this factorization is obtained by applying singular value decomposition (SVD) to $R$ as is detailed in Section 3.2.3. Before applying SVD, the number of $c_l$ variables, $L$, must be estimated. In SVA this is done by using parallel analysis as described in Section 3.2.4.

3.2.2 Estimation of basis function model

To estimate the functions $f_j$ the SVA methodology prescribes using standard basis models. The task is then to choose a feature mapping $\phi$:

$$\phi : \mathbb{R} \to \mathbb{R}^p$$

$$y \mapsto (\phi_1(y), ..., \phi_p(y))^T$$

assuming that $y$ is univariate so that:

$$f_j(y) = \mathbb{E}[x_j|y] := \sum_{k=1}^{p} \beta_{jk}\phi_k(y)$$ (3.2.2.3)

and estimating $f_j$ for $j \in \{1, ..., J\}$ reduces to estimating the matrix $\beta \in \mathbb{R}^{J \times p}$. Least squares estimation is then applied to the regression model

$$x_j = \sum_{k=1}^{p} \beta_{jk}\phi_k(y) + \epsilon$$ (3.2.2.4)

or in vector form
3.2 Estimation of $\text{span}(\{c_l\})$

\[ X_j = \Phi \beta_j + \epsilon \quad (3.2.2.5) \]

Where

- $X_j \in \mathbb{R}^n$ is the j-th column of the observation matrix $X$,
- $\beta_j \in \mathbb{R}^p$ is the j-th row of $\beta$ as a column vector,
- $\Phi \in \mathbb{R}^{n \times p}$ and $\Phi_{ik} = \phi_k(y_i)$.

The well known least squares solution for $\beta_j$ is

\[ \beta_j = (\Phi^T \Phi)^{-1} \Phi^T X_j \quad (3.2.2.6) \]

and the fitted values $F$ are

\[ F = HX = \Phi (\Phi^T \Phi)^{-1} \Phi^T X \quad (3.2.2.7) \]

where $H$ is the hat matrix. Notice that since we choose one mapping $\phi$ to define all $f_j$ then the hat matrix $H$, which only depends on $\Phi$, is the same for all the models to be fitted. In the function sva of the R package sva, which implements the SVA methodology, the hat matrix $H$ must be provided as input so that the choice of basis defined by $\phi$ and the estimation of functions $f_j$ is done independently by the user. The matrix $F$ is then calculated by the function sva as $F = H X$. Notice that this means that no regularization is applied to any of the $J$ regression models to avoid overfitting. This is because since $J$ is large, each regression model would need its separate regularization parameter $\lambda$ which would be unfeasible - for example because cross validation would need to be performed for each model. This means that we must regularize by choosing a simple mapping $\phi$. This is especially true since the number of observations $n$ will usually be in the order of 10 meaning a complex $f_j$ will tend to overfit the data.

3.2.3 Factorization of residuals using SVD

Suppose $J > n$. If we apply SVD to the residual matrix $R$ we have:

\[ R = Q D S^T \quad (3.2.3.1) \]

Where

- $D \in \mathbb{R}^{n \times J}$ and $D = \text{diag}\{d_1, \ldots, d_n\}$ where $d_1 \geq d_2 \geq \ldots \geq d_n \geq 0$, $d_i$ are the singular values of $R$ and $d_i^2$ are the eigenvalues of $R^T R$ and $RR^T$,
- $Q \in \mathbb{R}^{n \times n}$ and its columns $\{q_i\}_{i=1, \ldots, n}$ are an orthonormal basis for $\mathbb{R}^n$ and the eigenvectors of $RR^T = Q D S^T S D Q^T = Q D^2 Q^T$ corresponding to the eigenvalues $\{d_i^2\}_{i=1, \ldots, n}$. 
• $S \in \mathbb{R}^{J \times J}$ and its columns $\{s_i\}_{i=1,\ldots,J}$ are an orthonormal basis for $\mathbb{R}^J$ and the first $n$ are the eigenvectors of $R^TR = SDQ^TQDS^T = SD^2S^T$ corresponding to the eigenvalues $\{d_i^2\}_{i=1,\ldots,n}$.

In the following we suppose that we know exactly $L \leq n$ of the singular values $d_i$ are significantly different from zero.

**Left SVD**

Let $W = Q^TR$ then

$$R = QW = \begin{pmatrix} Q_1 & Q_2 \end{pmatrix} \begin{pmatrix} W_1 \\ W_2 \end{pmatrix} = Q_1W_1 + Q_2W_2$$ (3.2.3.2)

Where

• $Q \in \mathbb{R}^{n \times n}$, $Q_1 \in \mathbb{R}^{n \times L}$, $Q_2 \in \mathbb{R}^{n \times (n-L)}$ and

• $W \in \mathbb{R}^{n \times J}$, $W_1 \in \mathbb{R}^{L \times J}$, $W_2 \in \mathbb{R}^{(n-L) \times J}$.

So if we make $C := Q_1$, $\Lambda := W_1$ and $E := Q_2W_2$ we have an estimate for the factorization of the observed residuals $R$ in terms of the estimated realizations of $c_l$ and $N_j$ variables as in 3.2.1.10 and 3.2.1.12. Some of the properties of this factorization are:

1. Since the unobserved realizations of $c_l$ are estimated as the l-th left eigenvector of $R$, which is orthonormal, the estimated realizations have a sample variance of 1 which is consistent with definition 2.3.0.1.

2. Since the columns of $C$ are the left eigenvectors of $R$ and we assume the expected value of $c_l$ to be zero, the estimated realizations of $c_l$ have zero sample correlation with the estimated realizations of $c_m$ for $l \neq m$:

$$\widehat{\text{Corr}}(c_l, c_m) = q_l^T q_m = 0$$ (3.2.3.3)

This is consistent with property 1. of 2.3.0.2 however it is not sufficient to satisfy it since zero correlation does not imply independence except in the case of normally distributed $c_l$ variables.

3. The estimated realizations of $N_j$ are given by the j-th column of $E := Q_2W_2$ which is $Q_2w_{2j}$ where $w_{2j}$ is the j-th column of $W_2$. Since the columns of $C$ are the left eigenvectors of $R$ and we assume the expected value of $c_l$ to be zero, the estimated realizations of $c_l$ have zero sample correlation with the estimated realizations of $N_j$:

$$\widehat{\text{Corr}}(c_l, N_j) = q_l^T (Q_2w_{2j}) = (q_l^T Q_2)w_{2j} = 0$$ (3.2.3.4)

This is consistent with definition 2.3.0.1 however it is not sufficient to satisfy it since zero correlation does not imply independence except in the case of normally distributed $c_l$ and $N_j$ variables.
3.2 Estimation of \( \text{span}\{c_l\} \)

Right SVD

Let \( Y = RS \) then:

\[
R = YS^T = \begin{pmatrix} Y_1 & Y_2 \end{pmatrix} \begin{pmatrix} S_1^T \\ S_2^T \end{pmatrix} = Y_1S_1^T + Y_2S_2^T
\]

(3.2.3.5)

Where

- \( Y \in \mathbb{R}^{n \times J}, Y_1 \in \mathbb{R}^{n \times L}, Y_2 \in \mathbb{R}^{n \times (J-L)} \) and
- \( S \in \mathbb{R}^{J \times J}, S_1 \in \mathbb{R}^{J \times L}, S_2 \in \mathbb{R}^{J \times (J-L)} \).

So if we make \( C := Y_1, \Lambda := S_1 \) and \( E := Y_2S_2 \) we have an estimate for the factorization of the observed residuals \( R \) in terms of the estimated realizations of \( c_l \) and \( N_j \) variables as in 3.2.1.10 and 3.2.1.12. Notice that the matrix \( Y \) contains the principal components of the residuals \( R \). Some of the properties of this factorization are:

1. Since the unobserved realizations of \( c_l \) are estimated as the \( l \)-th principal component of \( R \), the estimated realizations have a sample variance of \( d_l^2 \). This is inconsistent with definition 2.3.0.1 which assumes that the \( c_l \) variables have a variance of 1, but we may easily fix this by using \( Z = YD^{-1} = RSD^{-1} \) so that \( C := Z_1 \) instead of \( Y_1 \).

2. Since the columns of \( C \) are the first \( L \) principal components of \( R \) and we assume the expected value of \( c_l \) to be zero, the estimated realizations of \( c_l \) have zero sample correlation with the estimated realizations of \( c_m \) for \( l \neq m \):

\[
\overline{\text{Corr}}(c_l, c_m) = (Rs_l)^T(Rs_m) = s_l^TR^TRs_m = s_l^TSD^2S^Ts_m = e_l^TD^2e_m = 0 \quad (3.2.3.6)
\]

Where \( e_l \) is the \( l \)-th canonical vector. This is consistent with property 1. of 2.3.0.2 however it is not sufficient to satisfy it since zero correlation does not imply independence except in the case of normally distributed \( c_l \) variables.

3. The estimated realizations of \( N_j \) are given by the \( j \)-th column of \( E := Y_2S_2^T \) which is \( Y_2\tilde{s}_{2j} \) where \( \tilde{s}_{2j} \) is the \( j \)-th row of \( S_2 \) as a column vector. Since the columns of \( C \) are the first \( L \) principal components of \( R \) and we assume the expected value of \( c_l \) to be zero, the estimated realizations of \( c_l \) have zero sample correlation with the estimated realizations of \( N_j \):

\[
\overline{\text{Corr}}(c_l, N_j) = y_l^T(Y_2\tilde{s}_{2j}) = (Rs_l)^T(Y_2\tilde{s}_{2j})
\]

(3.2.3.7)

\[
= s_l^TR^TRs_{2j} = s_l^TSD^2S^Ts_{2j}
\]

(3.2.3.8)

\[
= e_lD^2(S^TS)s_{2j} = d_l^2e_l \left( \begin{matrix} 0_L \times (J-L) \\ I_{(J-L) \times (J-L)} \end{matrix} \right) \tilde{s}_{2j} = 0 \quad (3.2.3.9)
\]

Where \( y_l \) is the \( l \)-th column of \( Y_1 \) and \( l \in \{1, \ldots, L\} \). This is consistent with definition 2.3.0.1 however it is not sufficient to satisfy it since zero correlation does not imply independence except in the case of normally distributed \( c_l \) and \( N_j \) variables.
Now from the SVD factorization of $R$ and from left and right SVD factorizations above we have that:

$$R = YS^T = QW = QDS^T$$  \hspace{1cm} (3.2.3.10)

So that

$$Y = RS = QWS = QD$$  \hspace{1cm} (3.2.3.11)

$$YD^{-1} = Q$$  \hspace{1cm} (3.2.3.12)

So we see that performing right and left SVD is equivalent up to standardization of the estimated $c_l$ variables. Since $Y$ is the matrix of principal components of $R$ we also see that this factorization corresponds to obtaining the principal components and then standardizing them.

Equations 3.2.1.10 and 3.2.1.12 suggest that factor analysis may be a more suitable technique for extracting factors from the matrix $R$. However since in our setting $J > n$ factor analysis is not possible since it relies on the spectral decomposition of the covariance matrix $R^T R = SDQ^T QDS^T = SD^2 S^T$ which in this case is not positive definite.

### 3.2.4 Parallel Analysis

The SVA methodology uses a parallel analysis method based on $?$ to estimate the number factors $L$ needed to approximate the residual matrix as:

$$R \approx CA$$  \hspace{1cm} (3.2.4.1)

where

- $C \in \mathbb{R}^{n \times L}$ and
- $\Lambda \in \mathbb{R}^{L \times J}$ is the estimate of $A$ from 3.2.1.10.

We assume that the $\{r_j\}_{j=1,...,J}$ residual variables are mutually independent irrespective of the particular marginal distribution of each $r_j$ variable. Under this assumption a realization of $r_j$ is not linked to any particular realization of $r_k$ for $j \neq k$ meaning we can permute observations of the columns of $R$ without altering the underlying joint distribution of $\{r_j\}_{j=1,...,J}$. This assumption also implies that the population singular values for $R$ (i.e. if $n \to \infty$) are $d_j = \sigma_{\psi(j)}$ where $\sigma_j$ is the standard deviation of $r_j$ and $\psi(j)$ is such that:

$$\psi(i) < \psi(j) \Rightarrow \sigma_{\psi(i)} \geq \sigma_{\psi(j)}$$  \hspace{1cm} (3.2.4.2)

One method to choose the number of factors, for example in the context of PCA, is to choose the number of singular values such that:
3.2 Estimation of $\text{span}(\{c_l\})$

\[ d_j^2 \geq \hat{\sigma}^2_{\psi(j)} \] (3.2.4.3)

Where $\hat{d}_j$ is the estimate of the population singular value $d_j$ made by applying SVD to $R$ and $\hat{\sigma}^2_{\psi(j)}$ is the sample variance of $r_{\psi(j)}$. In the context of PCA this corresponds to choosing the number of principal components such that their corresponding eigenvalue is greater than one in the case that the data matrix has been standardized. We choose only the principal components that account for more of the variance than we would expect under the assumption of independence, or in other words we choose only those principal components that summarize the information of more than one variable. However, for limited data settings, the variance of $\hat{d}_j$ and $\hat{\sigma}^2_{\psi(j)}$ means that it is possible for $d_j^2 \geq \hat{\sigma}^2_{\psi(j)}$ even when $d_j^2 < \sigma^2_{\psi(j)}$. To take this variance into account parallel analysis prescribes permuting the values of each column of $R$ several times so that we obtain a distribution for each singular value under the independence assumption. For each column $j$, $B$ permutations $\pi_{ij}(k)$ are generated where $i \in \{1, ..., B\}$ indicates the permutation number, $j \in \{1, ..., J\}$ the column to which it will be applied and $k \in \{1, ..., n\}$ is the row index argument:

\[
\pi_{ij} : \{1, ..., n\} \rightarrow \{1, ..., n\} \quad (3.2.4.4)
\]

\[
(1, ..., n) \mapsto (\pi_{ij}(1), ..., \pi_{ij}(n)) \quad (3.2.4.5)
\]

In this way we obtain $B$ realizations of the residual matrix $R$ which we denote $R^b_i$. For each $R^b_i$ we apply SVD so that we have $B$ estimates $d^b_{ij}$ of $d_j$ for $i \in \{1, ..., B\}$ and $j \in \{1, ..., J\}$ under the assumption of independence. We then compare the estimated proportion of variance explained by each principal component under the assumption of independence to the estimated proportion of variance explained without this assumption, i.e. with the non-permuted $R$ matrix:

\[
\hat{\nu}_j = \frac{d_j^2}{\sum_{j=1}^{J} d_j^2} \quad \text{(3.2.4.6)}
\]

\[
\hat{\nu}^b_{ij} = \frac{(d^b_{ij})^2}{\sum_{j=1}^{J} (d^b_{ij})^2} \quad \text{(3.2.4.7)}
\]

If $d_j^2 \geq \sigma^2_{\psi(j)}$ then we expect that for most realizations $i \in \{1, ..., B\}$:

\[
\hat{\nu}_j \geq \hat{\nu}^b_{ij} \quad \text{(3.2.4.8)}
\]

If this is the case we say the corresponding singular value $d_j$ is significant. To obtain an estimate for $L$ we count the number of significant singular values.

Other considerations taken into account by ? in their implementation of the parallel analysis algorithm suggested by ? are:
1. **Degrees of freedom of model**: We know that:

\[ R = X -HX \] (3.2.4.9)

so that

\[ \text{rank}(R) = \text{rank}(X) - \text{rank}(HX) \] (3.2.4.10)

and if we assume that \( n < J \) and \( X \) has full rank then

\[ \text{rank}(R) = n - \text{rank}(H) = n - \text{tr}(H) \] (3.2.4.11)

since \( H \) is an idempotent matrix. So we only consider \( M := n - \lfloor \text{tr}(H) \rfloor \geq L \) non-zero singular values as potentially significant.

2. **Orthogonal residuals**: Since least squares is used to fit models 3.2.2.5 the residuals should be, by construction, orthogonal to the predictor variables: each column of \( R \) should be orthogonal to the linear span of \( \Phi \). However, by permuting the residuals in each column of \( R \) to create \( R^b_i \) we create a linear dependence between the columns of \( R^b_i \) and the linear span of \( \Phi \). This is fixed by projecting the permuted residuals \( R^b_i \) on the linear span of \( \Phi \) using the hat matrix to create the corrected residuals \( \tilde{R}^b_i \):

\[ \tilde{R}^b_i := R^b_i - HR^b_i \] (3.2.4.12)

3. **Quasi hypothesis test**: To decide if a singular value \( d_j \) is significant the proportion \( p^b_j \) of bootstrap estimates \( \hat{\nu}^b_{ij} \) such that \( \hat{\nu}^b_{ij} \geq \hat{\nu}_j \) is calculated:

\[ p^b_j := \frac{\sum_{i=1}^{B} \mathbb{1}\{\hat{\nu}^b_{ij} \geq \hat{\nu}_j\}}{B} \] (3.2.4.13)

and the null hypothesis that \( d^b_j \) is not significant is rejected if \( p^b_j < 0.1 \). The cutoff value 0.1 is arbitrarily chosen and since the distribution of the test statistic \( p^b_j \) is unknown the significance or power of this test is unknown.

4. **Non-increasing singular values**: Singular values \( d_j \) are non-increasing, meaning \( \nu_j \) should be non-increasing and, for a large enough \( B \), \( p^b_j \) non-decreasing. However because of the randomness of the bootstrap estimates and the limited data setting, the \( p^b_j \) values may not be non-decreasing. A corrected value \( \tilde{p}^b_j \) is calculated:

\[ \tilde{p}^b_j := \max(p^b_{j-1}, p^b_j) \] (3.2.4.14)

for \( j \in \{2, ..., M\} \).

With these consideration in mind we present the pseudo code for the parallel analysis algorithm described in ? and implemented as the default method in the function \texttt{num.sv} of the R package \texttt{sca}. 

---

**SVA Methodology**
Parallel analysis algorithm of $\text{span}\{\{c_l\}\}$

1. Estimate matrices $X$, $F$, $R$ and the rank $M$ of $R$:
   
   $H = \Phi(\Phi^T\Phi)^{-1}\Phi^T$ \hspace{1cm} (3.2.4.15)
   
   $F = HX$ \hspace{1cm} (3.2.4.16)
   
   $R = X - F$ \hspace{1cm} (3.2.4.17)
   
   $M = n - \lceil\text{tr} \ H \rceil$ \hspace{1cm} (3.2.4.18)

2. Estimate singular values $d_j$ by performing SVD on $R$:
   
   $R = QDS^T$ \hspace{1cm} (3.2.4.19)
   
   where $D = \text{diag}\{\hat{d}_1, ..., \hat{d}_n\}$.

3. Estimate the proportion of variance explained by first $M$ principal components:
   
   $\hat{\nu}_j = \frac{\hat{d}_j^2}{\sum_{j=1}^{n} \hat{d}_j^2}$ \hspace{1cm} (3.2.4.20)
   
   for $j \in \{1, ..., M\}$.

4. Calculate bootstrap estimates $\hat{d}_{ij}^b$ for $i \in \{1, ..., B\}$ and $j \in \{1, ..., M\}$.
   
   For $i = 1, ..., B$
   
   a. Generate $J$ permutations $\pi_1, ..., \pi_J$ one for each column of $R$.
   
   b. Permute columns of $R$ according to these permutations to create $R_i^b$:
   
   $$ R_i^b = \begin{pmatrix} R_{\pi_1(1),1} & R_{\pi_2(1),2} & \cdots & R_{\pi_J(1),J} \\ \vdots & \vdots & \ddots & \vdots \\ R_{\pi_1(n),1} & R_{\pi_2(n),2} & \cdots & R_{\pi_J(n),J} \end{pmatrix} $$ \hspace{1cm} (3.2.4.21)
   
   c. Project each column of $R_i^b$ onto the columns of $\Phi$ and calculate adjusted residuals:
   
   $$ \tilde{R}_i^b = R_i^b - HR_i^b $$ \hspace{1cm} (3.2.4.22)
   
   d. Apply SVD to $\tilde{R}_i^b$ to obtain estimates $\hat{d}_{ij}^b$ under independence assumption.
   
   $$ \tilde{R}_i^b = Q_i^b D_i^b (S_i^b)^T $$ \hspace{1cm} (3.2.4.23)
   
   where $D_i^b = \text{diag}\{\hat{d}_1^b, ..., \hat{d}_n^b\}$. 
e. Estimate the proportion of variance explained by first $M$ principal components:

$$\hat{\nu}_{ij}^b = \frac{(\hat{d}_{ij}^b)^2}{\sum_{j=1}^{J}(\hat{d}_{ij}^b)^2}$$

(3.2.4.24)

for $j \in \{1, ..., M\}$.

5. Calculate test statistics: For each singular value $d_j$, $j \in \{1, ..., M\}$ calculate the proportion of bootstrap estimates for which the variance explained is higher than the non-permuted estimate:

$$p_j^b := \frac{\sum_{i=1}^{B} \mathbb{1}\{\hat{\nu}_{ij}^b \geq \nu_j\}}{B}$$

(3.2.4.25)

6. Correct test statistics $\{p_1^b, ..., p_M^b\}$ to ensure they are non-decreasing:

$$\tilde{p}_j^b := \max(p_{j-1}^b, p_j^b)$$

(3.2.4.26)

7. Calculate the number of of singular values for which the corresponding corrected proportion of bootstrap estimates $p_j^b$ is lower than 0.1.

$$L = \sum_{j=1}^{M} \mathbb{1}\{\tilde{p}_j^b \leq 0.1\}$$

(3.2.4.27)

### 3.3 Estimation of $\text{span}\{\{h_k\}\}$

#### 3.3.1 General description

We can observe the $x_j$ variables and have estimated the span of the $c_l$ variables. We now want to estimate the span of the $h_k$ variables. The $x_j$ variables depend on three quantities - $c_l$, $f_{h_k}(y)$ and $f_{x_j}(y)$ - while the $h_k$ variables only depend on $c_l$ and $f_{h_k}(y)$. If we can somehow filter out the $f_{x_j}(y)$ signature from the $x_j$ variables we could then reconstruct $h_k$ from these filtered variables. The problem is it is impossible to separate the $f_{x_j}(y)$ and $f_{h_k}(y)$ signals since they both depend on $y$. The strategy will be to build an $h_k$ variable for every $c_l$ variable which contains the signal from $c_l$ but which is also allowed to contain signal from $y$ even if it is not exactly the $f_{h_k}(y)$ signal we would like it to have. This is important since not including the overlap in signal between $y$ and $h_k$ can lead to biased estimate of $h_k$ as is pointed out in the third paragraph of the section titled comparison with existing methods of 7. The basic algorithm has the following steps, which we will elaborate on in the subsequent subsections:

For $l \in \{1, ..., L\}$

i.) Identify the subset $S \subset \{1, ..., J\}$ such that $x_j$ for $j \in S$ includes the signature of $c_l$. 


3.3 Estimation of $\text{span}(\{h_k\})$

ii.) Form an enriched matrix $X^S \in \mathbb{R}^{n \times |S|}$ which includes the observations of the $x_j$ variables such that $j \in S$.

iii.) Factorize enriched matrix $X^S$ into $M$ factors using left SVD of section 3.2.3 and the parallel analysis method of section 3.2.4.

iv.) Set estimate of $h_l$ as the factor with highest absolute correlation with $c_l$. Let $q_1, ..., q_M$ be the eigen vectors of $X^S(X^S)^T$ corresponding to eigenvalues $d_1^2 \geq d_2^2 \geq ... \geq d_M^2$

$$i^* = \arg \max_{i \in \{1, ..., M\}} \text{Corr}(h_l, q_i)$$

$$\hat{h}_l = q_{i^*}$$

Before detailing what it means for a variable to include the signature of another variable and how this is done in the SVA methodology we remark on some advantages and disadvantages of the algorithm described above. We assume that the method for finding the variables $x_j$ that contain the signature of $c_l$ is adequate.

Advantages

- A lot of the $f_{x_j}$ signal can be filtered out especially if $J >> L$ and the size of $S$ is small (because a lot of the $\gamma_{lk}$ and $\beta_{kj}$ parameters are equal to zero),

- If for all the $j \in S$, there exists $k$ such that $\beta_{kj} \neq 0$ then the $x_j$ contain the signal of $f_{h_k}$.

Disadvantages

- The $f_{x_j}$ signal for $j \in S$ is not filtered out,

- By taking the factor from the enriched matrix most correlated with $c_l$ we ensure that the estimated span of $h_k$ includes the signal from $c_l$ but not necessarily the signal from $f_{h_k}$ as that may be concentrated in the other factors,

- In general it is only clear that the estimated span of $h_k$ allows for the inclusion of signal from $y$ and that if $J$ is large and $|S|$ small, it may filter out most of the $f_{x_j}(y)$ signal, however it isn’t clear how succesful it is in including the $f_{h_k}(y)$ signal.

3.3.2 Finding the signature of $c_l$

To find the set $S$ with the indices of the variables $x_j$ which have the signature of $c_l$ the following regression models are fitted:

$$x_j = \beta_{j0} + \beta_{j1}c_l + \epsilon_j$$

for $j \in \{1, ..., J\}$. We say that $x_j$ contains the signature of $c_l$ if we can reject the null hypothesis $H_{j0} : \beta_{j1} = 0$. This means that the set $S$ is defined as:

$$S := \{j : H_{j0} \text{ is rejected}\}$$
Clearly $x_j$ also depend on $y$ so that the hypothesis tests and corresponding p-values are dependent and a conventional significance analysis will be invalid. It is not clear to the author why the SVA methodology does not use the following model:

$$x_j = f_j(y) + \beta_j c_j + N_j$$

(3.3.2.3)

instead of model 3.3.2.1. However, even in this case the hypothesis tests would be dependent since as we saw in 3.2.1.5 the error term $N_j$ is composed of error terms $N_{hk}$ which are common across different $j$:

$$N_j = \sum_{k=1}^{K} \beta_{kj} N_{hk} + N_{xj}$$

(3.3.2.4)

There are two difficulties with this multiple hypotheses strategy, one of which we have touched upon (dependence of hypotheses). Before exploring these difficulties further, we classify the $m$ hypotheses of a multiple hypotheses problem according to the ground truth of the hypothesis and whether we reject them or not:

| truth  | rejected | not rejected | total |
|--------|----------|--------------|-------|
| null   | $F$      | $m_0 - F$    | $m_0$ |
| alternative | $T$   | $m_1 - T$    | $m_1$ |
| total  | $S$      | $m - S$      | $m$   |

1. **Multiple hypotheses**: When we perform a single hypothesis $\phi$ and reject the null if the p-value is below a certain threshold we are controlling the probability of making a Type I error:

$$\mathbb{P}[\phi(X) = 1|H_0] \leq \alpha$$

(3.3.2.5)

where $X$ is the sample upon which the hypothesis is based. If we perform multiple hypotheses $\phi_i$ and reject analogously this corresponds to controlling the *false positive rate* (FPR):

$$FPR = \frac{\mathbb{E}[F]}{m} \leq \alpha$$

(3.3.2.6)

However this does not provide effective control of the number of Type I errors we make. Typically, this is fixed by controlling the *family wise error rate* (FWER) instead of the false positive rate:

$$FWER = \mathbb{P}[F \geq 1] \leq \alpha$$

(3.3.2.7)
3.3 Estimation of \( \text{span}(\{h_k\}) \)

This can be achieved with the p-values by using the Bonferroni correction. Other methods exist which also control the FWER while increasing the power of the overall procedure. However, for many high dimensional applications this compound error measure is too strict because in these cases we often don’t care that we make more than a few Type I errors so long as the proportion of these errors \( F \), relative to the total number rejected hypotheses \( S \) is small. This is the case when we try and find the variables \( x_j \) that include the signature of a certain variable \( c_l \). So long as the proportion of false discoveries \( F \), those variables \( x_j \) which we declare contain the signature of \( c_l \) but which in reality don’t, are a small proportion of the total discoveries \( S \) then we are satisfied. This is why the SVA methodology, in performing the \( m = J \) tests as described in Section 3.3.1 seeks to control the false discovery rate (FDR):

\[
FDR = \mathbb{E}\left[ \frac{F}{S \lor 1} \right] = \mathbb{E}\left[ \frac{F}{S} \B | S > 0 \right] \mathbb{P}[S > 0] \leq \alpha
\]

(3.3.2.8)

Where the “\( \lor 1 \)” part takes care of the possibility that \( S = 0 \) by setting the whole quotient to zero.

2. **Dependent hypotheses tests**: We actually know that since the \( x_j \) variables depend on \( y \) for some \( j \in \{1, \ldots, J\} \) the hypotheses are dependent and so any joint significance analysis, which controls either the FWER or the FDR, will be invalid in principle. However, if the signal \( f_h(y) \) is weak compared to that of \( c_l \) then the significance may still be valid. To gauge whether this is the case we examine the p-value distribution. Qualitatively speaking it is simple to distinguish between valid and invalid p-value distributions. We may assume that each p-value has the following mixture distribution:

\[
P_i \sim \pi_0 F_0 + \pi_1 F
\]

(3.3.2.9)

where

- \( \pi_0 \) is the proportion of null hypothesis tests \( H_i \),
- \( \pi_1 = 1 - \pi_0 \) is the proportion of alternative hypothesis tests \( H_i \),
- \( F_0 \) is the distribution of a uniform random variable with support on \([0, 1]\) since assuming a one-sided hypothesis test we have that:

\[
\mathbb{P}[P_i \leq x | H_0] = \mathbb{P}[F_0(T) \leq p] = \mathbb{P}[T \leq F_0^{-1}(p)] = F_0(F_0^{-1}(p)) = p
\]

(3.3.2.10)

and,

- \( F \) is a right-skewed probability distribution since it is the distribution of alternative p-values which have a greater probability of being on the low end of the \([0, 1]\) interval.

Graphically the a valid p-value distribution should be the superposition of a uniform distribution and a right-skewed distribution. Figure 3.3 shows examples of valid and invalid p-distributions.
The SVA methodology uses a *point-based* method - to be explored in Section 3.3.3 - for controlling the FDR which:

I. Is still valid under *weak dependence* of the hypotheses tests (see remark D of ? or ? for a definition of weak dependence) and,

II. In case of an invalid p-value distribution caused by dependence between the hypotheses tests automatically *detects* this so that the number of variables $x_j$ estimated as having the signature of $c_l$ is set to zero.

### 3.3.3 FDR based significance analysis

Two related quantities to the FDR are the *positive false discovery rate* (pFDR)

\[
pFDR = \mathbb{E}\left[ \frac{F}{S} \mid S > 0 \right]
\]  \hspace{1cm} (3.3.3.1)

and the *marginal false discovery rate* (mFDR)

\[
mFDR = \frac{\mathbb{E}[F]}{\mathbb{E}[S]}
\]  \hspace{1cm} (3.3.3.2)

As ? surmises, there is some disagreement as to which quantity is more adequate to control however, for settings when there is a large number of hypothesis and the probability that $S = 0$ is negligible FDR, pFDR and mFDR are similar.

There are two approaches to control FDR:

1. **FDR Control.** Fix FDR at a certain level $\alpha$ and come up with a data-dependent thresholding rule such as the *FDR controlling algorithm* proposed by ? and proven to control FDR at level $\alpha$ by ?.
2. **Point estimate of FDR.** Fix a p-value threshold \( t \) for rejecting hypothesis and estimate the corresponding \( FDR(t) \) conservatively. *Q-values* and *local false discovery rates* (lFDR) represent two point estimate approaches for controlling FDR.

? show that under a certain choice for the estimation of \( m_0 \) and threshold \( t \) the FDR controlling algorithm and q-value approach are equivalent.

We first focus on the q-value point estimation method and then go on to explore the local false discovery rate which is used in the R package `sva` to control the FDR. Let:

\[
F(t) = \# \{ \text{null } P_i \leq t : i = 1, \ldots, m \} \\
S(t) = \# \{ P_i \leq t : i = 1, \ldots, m \}
\]  

(3.3.3.3) \hspace{1cm} (3.3.3.4)

Where \( P_i \) are i.i.d. random variables. We want to estimate FDR in terms of the threshold \( t \). For \( m \) large we have that

\[
FDR(t) \approx pFDR(t) \approx mFDR(t) = \frac{\mathbb{E}[F(t)]}{\mathbb{E}[S(t)]}
\]

(3.3.3.5)

We can estimate \( \mathbb{E}[S(t)] \), the expected number of significant variables simply as the observed number of significant variables:

\[
\hat{\mathbb{E}}[S(t)] = \# \{ p_i \leq t, i = 1, \ldots, m \}
\]

(3.3.3.6)

Now

\[
\mathbb{E}[F(t)] = \mathbb{E}[\# \{ \text{null } P_i \leq t \}] = m_0 t
\]

(3.3.3.7)

since \( P_i | H_0 \sim U[0, 1] \). We can estimate \( m_0 \), the total number of null hypotheses, in terms of the total number of hypotheses \( m \) and the proportion of null hypotheses \( \pi_0 \):

\[
\hat{m}_0 = \hat{\pi}_0 m
\]

(3.3.3.8)

We estimate \( \pi_0 \) using the p-value histogram. P-values greater than a certain threshold, say \( \lambda \), will be mostly null p-values. We can estimate the proportion of null p-values as the height of the histogram for values larger than \( \lambda \) divided by the average height of the histogram.
To the right of some point $\lambda$ close to 1 there are almost only null p-values so we could use this $\lambda$ to calculate $\pi_0$. However, the closer we get to $\lambda = 1$ the less data points we use to compute $\hat{\pi}_0$ so the the higher the variance of the estimator: there is a bias-variance trade-off in choosing $\lambda$. For the histogram of figure 3.4 we may plot the estimate $\hat{\pi}_0$:

$$\hat{\pi}_0(\lambda) = \frac{\# \{ p_i > \lambda \}}{m(1 - \lambda)} \quad (3.3.3.9)$$
3.3 Estimation of \( \text{span}(\{h_k\}) \)

As prescribed in ? a natural cubic spline \( \hat{f}(\lambda) \) is fitted to \( \hat{\pi}_0(\lambda) \) and \( \pi_0 \) is estimated as \( \hat{f}(1) \). This method borrows strength across \( \hat{\pi}_0(\lambda) \) giving a balance between variance and bias as is explained in remark B of ?. Notice that if we have an invalid p-value distribution such as that of figure 3.3b then \( \hat{\pi}_0 \geq 1 \). If we set values larger than 1 to 1, then as mentioned at the end of Section 3.3.2 we have a way of automatically detecting invalid p-value distributions and setting the number of discoveries to zero.

Going back to our estimate of \( FDR(t) \) we have:

\[
\overline{FDR}(t) = \frac{\hat{\mathbb{E}}[F(t)]}{\hat{\mathbb{E}}[S(t)]} = \frac{m\hat{\pi}_0 t}{\# \{ p_i \leq t, i = 1, \ldots, m \}} \tag{3.3.3.10}
\]

Notice that if we threshold p-values with \( t = 1 \) then \( FDR(1) = \hat{\pi}_0 \), in other words if we fail to reject all hypothesis our FDR will be \( \hat{\pi}_0 \).

Since \( FDR(t) \) is a non-increasing function, we can’t simply choose a level \( \alpha \) and then find a \( t \) such that if we threshold the p-values with a value of \( t \) or lower we guarantee that that \( FDR(t) \leq \alpha \). This means the p-value loses its meaning of expressing the significance of each feature because a feature might have a lower p-value, but thresholding at the lower p-value might induce a higher FDR. For illustrative purposes suppose we have 6 hypothesis tests with the following p-values which we have ordered for convenience:
| Hypothesis | p-value |
|------------|---------|
| $p_0 :=$   | 0       |
| $p_1$      | 0.05    |
| $p_2$      | 0.1     |
| $p_3$      | 0.2     |
| $p_4$      | 0.35    |
| $p_5$      | 0.6     |
| $p_6$      | 0.85    |
| $p_7 :=$   | 1       |

We can see that we could threshold p-values at $t_1$ or $t_2$ and obtain the same $FDR$. We need a new definition for a value which expresses the significance of each feature. This value is called the *q-value*. The q-value of a feature is the minimum FDR that can be attained if we include a feature (call it significant):

$$q_i = q(p_i) := \min_{t \geq p_i} FDR(t) = \min_{t \geq p_i} pFDR(t)$$ (3.3.3.11)

This definition ensures we can use q-values to control FDR since we choose the thresholding value that minimizes FDR and in doing so implicitly define a strictly increasing function $q^*$ which we can use to control FDR. Define the function $q^*$ as:
\[ q^*(t) := \inf_{s \in [t,1]} \overline{FDR}(s) \quad (3.3.3.12) \]

Also note that

\[ \overline{FDR}(t) = \frac{m\hat{\pi}_0 t}{\# \{p_i \leq t, i = 1, \ldots, m \}} = m\hat{\pi}_0 t \sum_{i=1}^{m} \frac{1}{i} [p_i(t) - p_{i+1}(t)](t) \quad (3.3.3.13) \]

for \( t \in [p(1), 1] \) and

\[ \frac{d\overline{FDR}(t)}{dt} = m\hat{\pi}_0 \sum_{i=1}^{m} \frac{1}{i} [p_i(t) - p_{i+1}(t)](t) \quad (3.3.3.14) \]

for \( t \in [p(1), 1] \setminus \{p_1, \ldots, p_m\} \)

We can see how \( q^*(t) \) can be used to set the p-value threshold \( t \) and to obtain the significance q-values for each feature graphically.

![Graph](image)

Figure 3.7: \( q^*(t) \)

Note that since:
\[
p_i \leq t \iff q_i = q^*(p_i) \leq q^*(t) \leq \hat{FDR}(t) = \alpha \iff q_i \leq \alpha \tag{3.3.3.15}
\]

we can threshold the q-values themselves at \( \alpha \) to control the \( FDR \) at this level.

To find the variables \( x_j \) which contain the signature of \( c_l \) then, one could either choose the \( \hat{m}_0 = \hat{\pi}_0 \ast m \) variables \( x_j \) with the lowest p-values for the hypothesis \( H_{j0} : \beta_{j1} = 0 \) or equivalently calculate the q-values and threshold at \( \alpha \) the desired level \( \alpha \). For reasons that are not clear to the author, in the implementation of the R package \texttt{sda} (specifically function \texttt{edge.lfdr}), another point estimate approach is used based on local false discovery rates (lFDR). This approach follows from a Bayesian interpretation of q-values.

In Theorem 1 of \cite{?}, \cite{?} proves that the positive false discovery rate for a given threshold \( t \) for the p-value, \( pFDR(t) \) is equal to the conditional probability that the corresponding hypothesis is null given that the p-value is below the threshold:

\[
pFDR(t) = \mathbb{E} \left[ \frac{F(t)}{S(t)} \middle| S(t) > 0 \right] = \mathbb{P}[H_i = 0 | P_i \leq t] \tag{3.3.3.16}
\]

This allows us to interpret the \( pFDR(t) \) as a posterior Bayesian Type I error where the prior probability of a hypothesis being null is \( \pi_0 \). Since

\[
q_i = q(p_i) = \min_{t \geq p_i} pFDR(t) \tag{3.3.3.17}
\]

we can interpret the q-value as a posterior Bayesian p-value. We can then express \( pFDR(t) \) as:

\[
pFDR(t) = \mathbb{P}[H = 0 | P \leq t] = \int \mathbb{P}[H = 0 | P = p] dG(p | p \leq t) \tag{3.3.3.18}
\]

Where \( G \) is the distribution of \( P_i \) from \ref{3.3.2.9}. We then define the local false discovery rate \( lFDR(p) \) of a hypothesis as:

\[
lFDR(p) := \mathbb{P}[H = 0 | P = p] \tag{3.3.3.19}
\]

Notice that this is not a function of the p-value thresholding level \( t \) and so it gives a measure of significance of the p-value without taking into account the multiple hypotheses being carried out. Using Bayes’ Theorem we have

\[
lFDR(p) = \mathbb{P}[H = 0 | P = p] = \frac{f_{P_0}(p) \pi_0}{f_P(p)} = \frac{\pi_0}{f_P(p)} \tag{3.3.3.20}
\]

where
• \( f_P(p) \) is the density of the p-values and

• \( f_{P_0}(p) = 1 \) is the density of the null p-values which we know is uniform by 3.3.2.10.

To estimate \( lFDR(p) \) we estimate \( \pi_0 \) as was illustrated in figures 3.4 and 3.5. To estimate \( f_P(p) \) standard kernel-density estimators can be used however these tend not to perform very well for random variables with bounded support, and so in the R package sva (specifically function `edge.lfdr`), the p-values are first transformed so that the underlying random variable has unbounded support. A standard change of variable theorem (see for example pg 153 of ?) is then used to calculate the density of the P-values in terms of the density of the transformed P-values which we can estimate effectively using kernel-density estimation. Let \( S = \Phi^{-1}(P) \) where \( \Phi \) is the standard normal cumulative distribution function, with associated \( \phi \) probability density function, then:

\[
    f_P(p) = \frac{f_S(s)}{\phi(s)} = \frac{f_S(\Phi^{-1}(p))}{\phi(\Phi^{-1}(p))}
\]

The estimate for \( lFDR(p) \) is then:

\[
    \hat{lFDR}(p) = \hat{\pi}_0 \frac{\hat{f}_S(\Phi^{-1}(p))}{\hat{f}_S(\Phi^{-1}(p))}
\]
Chapter 4

Simulation Experiments

4.1 Methods to be compared

Given \( n \) i.i.d. observations of \( y \) and \( x_j \) for \( j \in \{1, ..., J\} \) simulated from model 1.2.0.4, following \(?\), we will fit 4 different models to the data:

\[
x_j = f(y) + \epsilon_j \quad (4.1.0.1)
\]

\[
x_j = f_{x_j}(y) + \sum_{k=1}^{K-1} \beta_{kj} \hat{h}^{svdx}_k + \epsilon_j \quad (4.1.0.2)
\]

\[
x_j = f_{x_j}(y) + \sum_{k=1}^{M} \beta_{kj} \hat{h}^{svdr}_k + \epsilon_j \quad (4.1.0.3)
\]

\[
x_j = f_{x_j}(y) + \sum_{k=1}^{N} \beta_{kj} \hat{h}^{sva}_k + \epsilon_j \quad (4.1.0.4)
\]

\[
(4.1.0.5)
\]

for \( j \in \{1, ..., J\} \) where

- \( f(y) \) and \( f_{x_j}(y) \) are polynomials of the form \( a_1y + a_2y^2 + ... + a_n y^n \),

- \( \hat{h}^{svdx}_k \) is the estimate of \( h_k \) obtained by performing SVD on the observation matrix \( X \) (instead of on residual matrix \( R \)) to obtain \( K \) factors, as per sections 3.2.3 and 3.2.4. The factor \( h_k \) with largest absolute correlation with \( y \) is removed and the remaining \( K-1 \) factors are included as covariates. The idea is that if we exclude the strongest part of the \( y \) signal we will only be left with the unobserved \( h_k \) signal, however the other factors may still include undesirable \( f_{x_j}(y) \) signal and the removed factor could also include desirable \( f_{h_k}(y) \) signal,

- \( \hat{h}^{svdr}_k \) is the estimate of \( h_k \) obtained by performing SVD on the observation matrix \( R \) as in section 3.2.3 to obtain \( M \) estimates for the \( c_l \) factors. The estimates for \( c_l \) are plugged in as the \( h_k \) estimates. In this case we completely filter out all \( y \) signal including \( f_{h_k}(y) \) signal meaning that \( \hat{h}^{svdr}_k \) are uncorrelated to \( y \) which may lead to bias in the estimate of \( h_k \), and
• $\hat{h}_{sva}$ is the estimate of $h_k$ obtained by performing SVA on the observations of $y$ and $x_j$.

The four models are forthwith referred to as *vanilla* (or *van*), SVDX, SVDR and SVA. The main objective is to see if the SVA method has significant advantages in estimating $f_{x_j}(y)$ for $j \in \{1, ..., J\}$. This also entails that valid significance analysis is produced for the joint hypotheses tests $H_{j0} : f_{x_j}(y) = 0$.

4.2 Low-dimensional experiments

4.2.1 Design

As a first approximation we simulated $n = 100$ observations of the following low-dimensional SEM so as to follow the performance of each estimate $\hat{f}_{x_j}$:

- $J = K = L = 4$
- $y = N_y \sim N(0, 1)$
- $c_l = N_c_l \sim N(0, 1)$ for $l \in \{1, ..., 4\}$
- $h_k = f_{h_k}(y) + c_l$ for $k \in \{1, ..., 4\}$
- $f_{h_1}(y) = f_{h_2}(y) = 0$
- $f_{h_3}(y) = -0.28y + 1.29y^2$
- $f_{h_4}(y) = 1.54y + 0.59y^2$
- $x_j = f_{x_j}(y) + \sum_{k=1}^{K} \beta_{kj} h_k + N_{x_j}$ for $j \in \{1, ..., 4\}$
- Non-zero $\beta_{kj}$ values were generated randomly with standard normal generator:
  - $f_{x_1}(y) = f_{x_2}(y) = 0$
  - $f_{x_3}(y) = -0.92y$
  - $f_{x_4}(y) = 1.24y - 1.48y^2$, and
  - $N_{x_j} \sim N(0, 1)$ for $j \in \{1, ..., 4\}$

The DAG for this SEM is the following:
We chose a SEM with this DAG so that we get variety in the way $x_j$ depend on $y$:

1. $x_1$: depends on $y$ only through $f_{h_4}$,
2. $x_2$: doesn’t depend on $y$,
3. $x_3$: depends on $y$ through $f_{h_3}$ and $f_{x_3}$, and
4. $x_4$: depends on $y$ only through $f_{x_4}$.

Moreover for the estimation of $f_j(y)$ and $f_{x_j}(y)$ we chose polynomials of the form:

$$g(y) = a_1 y + a_2 y^2$$  \hspace{1cm} (4.2.1.1)

### 4.2.2 Result for one repetition

For $x_j$ with $j \in \{1, 2, 3, 4\}$ we display the results of the simulation with the following graphs:

- A comparison of $\hat{f}_{x_j}(y)$ using SVA to the real $f_{x_j}$,
- A comparison of $\hat{E}[x_j|y, h_k^{sva}]$ for the vanilla and SVA methods,
- A comparison of the residuals produced with the four methods, van, SVDX, SVDR and SVA, and
- A Tukey-Anscombe residual plot, with $\sum_{k=1}^{K} \beta_{kj} h_k$ on the x-axis.
\( \hat{f}_{x_j}(y) \) using SVA

![Graphs comparing \( \hat{f}_{x_j}(y) \) using SVA to the real \( f_{x_j} \)](image)

We can see that the SVA method does fairly well in estimating \( f_{x_j} \) when it has similar complexity to the polynomial being used to estimate it (in this case a quadratic polynomial). This is because the SVA method does not apply regularization since this would not be feasible for high dimensional cases. Since we are using quadratic polynomials to estimate first, \( f_j \) and subsequently, \( f_{x_j} \), we overfit the data.
4.2 Low-dimensional experiments

The SVA estimation seems to have very poor performance for \( \hat{E}[x_j|y, \hat{h}_k^{\text{sva}}] \) where overfitting was worse. This could be because the overfitting of \( f_1(y) \) leads to bad estimation of the \( \beta_{k1} \) coefficients to compensate. This shows up in the graph with a green SVA line that is noisier than the underlying true \( E[x_1|y, h_k] \).
Comparison of residuals for 4 methods

Figure 4.4: A comparison of the residuals produced with the four methods, van, SVDX, SVDR and SVA

A good estimation method would show residuals similar to those of the real errors $N_{x_j}$. None of the methods show consistency in this respect across the four $r_j$. Again for $r_4$ where the true model complexity is similar to the basis function model fitted SVA appears to do well.
4.2 Low-dimensional experiments

Tukey-Anscombe plots

![Tukey-Anscombe plots](image)

Figure 4.5: Tukey-Anscombe residual plot, with $\sum_{k=1}^{K} \beta_{kj} h_k$ on the x-axis

In general we see that SVA outperforms the vanilla method, and especially for $r_3$ and $r_4$ where the true model complexity is similar to the basis function model being fitted.

If the significance analysis is done properly, if it is valid, then under the null hypotheses $H_{j0} : f_{x_j}(y) = 0$ p-values should be distributed uniformly. In this case, with only one repetition of the experiment, there are only 4 p-values per method (one for each $j$) so that confirming uniformity is not possible. However in some cases it is clear that we can rule out the possibility of the p-values being uniformly distributed. The following table shows the p-values for each $j$ and method. These were obtained by simulating under the null.
Simulation Experiments

| $j$ | sva | svdx | svdr | van |
|-----|-----|------|------|-----|
| 1   | 0.820 | 0.000 | 0.000 | 0.000 |
| 2   | 0.230 | 0.000 | 0.016 | 0.390 |
| 3   | 0.235 | 0.000 | 0.000 | 0.014 |
| 4   | 0.804 | 0.000 | 0.016 | 0.055 |

It seems that only the SVA method produces null p-values which could potentially be uniformly distributed.

4.2.3 Results for 1000 repetitions

We repeated the above simulation experiments this time for $M = 1000$ repetitions and tracked the following variables to measure the performance of the four methods:

1. **$c_l$-node span estimation.** % overlap between estimated $c_l$-node span and real $c_l$-node span. Using canonical correlation analysis (CCA) we measured what % of the linear span of $c_l$ and $\hat{c}_l$ is shared. This measures our ability to accurately estimate the $c_l$-node span. Since SVA is the only method that estimates the span of $c_l$ differently to the span of $h_k$ we only measure this for the SVA method.

2. **$h_k$-node span estimation.** % overlap between estimated $h_k$-node span and real $h_k$-node span (using CCA). In this case we are able to compare the SVA, SVDR and SVDX methods since each has an estimate for $h_k$ ($\hat{h}_{sva}$, $\hat{h}_{svdr}$ and $\hat{h}_{svdx}$ as described in Section 4.1). This measures our ability to accurately estimate the $h_k$-node span.

3. **Dependence between $h_k$ nodes and $y$.** R2 between real $y$ and $h_k$ nodes minus R2 between real $y$ and estimated $h_k$ nodes. The R2 corresponds to a simple linear regression with $y$ as the independent variable and $h_k$ or $\hat{h}_k$ as the dependent variables. This measures our ability to accurately model the dependence between $h$ and $y$ (to linear approximation).

4. **$f_{x_j}$ estimation.** Mean absolute error in estimation of $f_{x_j}$ calculated as:

$$\frac{\sum_{i=1}^{n} |f_{x_j}(y_i) - \hat{f}_{x_j}(y_i)|}{n} \quad (4.2.3.1)$$

The ultimate goal of all four methods is to estimate $f_{x_j}(y)$ accurately and this measure helps us evaluate this.

5. **Valid significance analysis.** We perform a nested Kolmogorov-Smirnov hypothesis test (nested KS test) to see if, under the null hypotheses $H_{j0} : f_{x_j}(y) = 0$ for $j \in \{1, ..., J\}$, the p-values are distributed for this hypothesis test are distributed uniformly. This measures whether we are performing valid significance analysis.

**Kolmogorov-Smirnov test** Suppose we have a sample of $n$ realizations from a random variable $X$. We want to perform the hypothesis test $H_0 : X_i \sim F$ vs. $H_a : X_i \sim F' \neq F$. We may use the Kolmogorov-Smirnov statistic $D_n$ for $F$. We know $\sqrt{n}D_n$ converges in distribution to a Kolmogorov random variable $K$ and so can perform an asymptotic test with this statistic:
\[ D_n = \sup_x |F_n(x) - F(x)| \] (4.2.3.2)

\[ \sqrt{n}D_n \overset{d}{\to} K \] (4.2.3.3)

where

- \( F_n \) is the empirical distribution function: \( F_n(x) = \frac{1}{n} \sum_{i=1}^{n} \mathbb{1}_{[-\infty,x]}(x_i) \) and

- \( K \) is a Kolmogorov random variable

For a large enough \( n \) if \( \sqrt{n}D_n > K_\alpha \), where \( \mathbb{P}[K \leq K_\alpha] = 1 - \alpha \), then \( H_0 \) is rejected.

**Nested KS test**

The idea of the nested KS test is that for a given repetition of the experiment \( m \) we have \( J \) p-values (call these outer p-values) and use the test statistic \( D_n \) for \( F \sim U[0,1] \) to test for uniformity. Under the null \( H_{outer}^m : \{ J \text{ outer p-values uniform} \} \) the p-value (call this a nested p-value) for this test is again uniform. If we collect the \( M \) nested p-values from \( M \) KS tests, one for each repetition of the \( J \) hypothesis tests \( H_{mj0} : f_{x_j}(y) = 0 \), then under the null they should be distributed uniformly. We perform another nested KS test \( H_{nested}^0 : \{ M \text{ nested p-values uniform} \} \) to conclude on the uniformity of the nested p-value distribution. As is mentioned in the section definition of a correct procedure of \( ? \) the nested KS test is more robust to chance fluctuations as a set of individual outer p-values corresponding to a repetition \( m \), \( H_{mj0} \), \( j \in \{1,\ldots,J\} \) may not be uniform due to chance fluctuations, but this will only contribute one of \( M \) nested p-values.

The following figure shows the first four indicators for the results of the 1000 repetitions of our low-dimensional simulation experiment.
The graphs above show that not only does SVA produce estimates of the span of $h_k$ with a large overlap with the real span of $h_k$ but it also manages to capture the relationship between $h_k$ and $y$ better than the other methods. However, this did not translate into more accurate estimates for $f_{x_j}$. Let’s see how valid the significance analysis for each method is by looking at the nested KS test statistic and p-values.

|       | sva | svdx | svdr | van |
|-------|-----|------|------|-----|
| KS statistic | 0.308 | 0.312 | 0.527 | 0.265 |
| p-value     | 0.000 | 0.000 | 0.000 | 0.000 |

Although in all cases we reject the hypothesis that p-values under null $H_{j0}: f_{x_j}(y)$ are
distributed uniformly, and so in all cases significance analyses are invalid, we are closer
to not rejecting for vanilla method. Since the vanilla method also produced the most
accurate estimates of $f_x$, it seems that the SVA methodology does not perform well in this
low-dimensional setting.

### 4.3 High-dimensional experiments

#### 4.3.1 Design

We now explore a high-dimensional setting more akin to the gene expression setting for
which SVA was designed. We simulated $M = 100$ repetitions of experiments with $n = 100$
realizations. The simulated SEM had the following characteristics:

- $J = 1000$, $K = L = 10$
- $y \sim N_y(0, 1)$
- $c_l \sim N(c_l) = N(0, 1)$ for $l \in \{1, \ldots, 10\}$
- $h_k = f_{hk}(y) + c_l$ for $k \in \{1, \ldots, 10\}$
- $f_{hk}(y) = b_k y$
- $x_j = f_{x_j}(y) + \sum_{k=1}^{K} \beta_{kj} h_k + N_{x_j}$ for $j \in \{1, \ldots, 1000\}$
- Non-zero $\beta_{kj}$ values were generated randomly with standard normal generator.
- $f_{x_j}(y) = a_j y$
- $N_{x_j} \sim N(0, 1)$ for $j \in \{1, \ldots, 1000\}$
- Non-zero $a_j$ and $b_k$ values were generated randomly with standard normal generator.

We control the sparsity of the SEM, the number of edges it has, with four parameters:

i. $p_{0j}$: the proportion of $j \in \{1, \ldots, J\}$ such that $f_{x_j}(y) = 0$. We set this to 0.5. The $j$ are chosen uniformly at random from $\{1, \ldots, J\}$. Denote this randomly

selected set as $J_0$

ii. $p_{0k}$: the proportion of $k \in \{1, \ldots, K\}$ such that $f_{hk}(y) = 0$. We set this to 0.5. The $k$ are chosen uniformly at random from $\{1, \ldots, K\}$. Denote this randomly

selected set as $K_0$

iii. $p_{0j}$: the minimum proportion of $(k, j) \in \{1, \ldots, K\} \times \{1, \ldots, J\}$ such that $\beta_{kj} = 0$. We set this to 0.5. The pairs $(k, j)$ are chosen uniformly at random from $\{1, \ldots, K\} \times \{1, \ldots, J\}$. Denote this randomly

selected set as $B_0$

iv. $p_{d-sep}$: the proportion of $j \in \{1, \ldots, J\}$ such that $x_j \ d-sep \ y \ by \emptyset$. We set this to 0.25. For this to be possible we need $p_{d-sep} \leq p_{0j}$. We sample $[J * p_{d-sep}]$

uniformly at random from $J_0$. Denote this randomly selected set as $J_{d-sep} \subset J_0$. Then for all $(k, j)$ such that $k \notin K_0$ and $j \in J_{d-sep}$ we set $\beta_{kj} = 0$. Notice that the actual proportion of $(k, j) \in \{1, \ldots, K\} \times \{1, \ldots, J\}$ such that $\beta_{kj} = 0$ is $\frac{|J_0 \cup J_{d-sep}|}{J}$, which we do not control and can be larger than $p_{0j}$ for a given

random selection of $J_0$ and $B_0$. 


4.3.2 Results for 100 repetitions

The following figure shows the first four indicators for the results of the 100 repetitions of our high-dimensional simulation experiment.

Figure 4.7: Evaluation of performance for high-dim sem with 100 repetitions: $c_l$ and $h_k$ node span estimation, dependence between $h_k$ nodes and $y$ and $f_{x_j}$ estimation

In this case the estimation of the span of the $c_l$ nodes with SVA is between 98.8% and 99.9% a lot higher than for the low dimensional experiment. The estimation of the span of $h_k$ nodes is clearly better with SVA and SVDR than with SVDX. In the median SVA is superior to SVDR in this respect although there is much more variance across the different $M$ repetitions. For modeling of the dependence between $y$ and $h_k$ in the high dimensional setting, SVA really stands out as for more than 95% of repetitions the R2
difference between the real \( y \) and \( h_k \) and the real \( y \) and estimated \( h_k \) was of 0.25 or less while for SVDR and SVDX the difference was of 0.5 or more for 90% of repetitions. The result of this in terms of the accuracy of \( f_{x_j} \) estimation is not clear-cut as for the median all methods perform similarly but SVA displays much greater variance across repetitions having much better accuracy for some and much worse for others. Let’s see how valid the significance analysis for each method is by looking at the nested KS test statistic and p-values.

|       | sva | svdx | svdr | van |
|-------|-----|------|------|-----|
| KS statistic | 0.099 | 0.637 | 0.622 | 0.457 |
| p-value     | 0.000 | 0.000 | 0.000 | 0.000 |

Although in all cases we reject the hypothesis that p-values under null \( H_{j0} : f_{x_j}(y) \) are distributed uniformly, and so in all cases significance analyses are invalid, we are closer to not rejecting for SVA method. The modeling of the relationship between the \( y \) and \( h_k \) nodes and the validity of the significance analysis show some evidence that SVA is superior to the other three methods considered.

**Additional remark:** We observed that in general more variables \( x_j \) for \( j \in \{1, ..., J\} \) are found to be contain the signature of \( c_l \), for \( l \) small than for \( l \) large. This seems to indicate that the first few factors (standardized principal components) are composed of more residuals \( r_j \) variables than the latter factors (or rather the corresponding weights are more even).

The experiments carried out here were somewhat different to those carried out in ? especially in terms of:

- \( f_{x_j} \) and \( f_{h_k} \) complexity: in ? only simple step functions where used while in this case we used linear functions, and
- sparsity of the sems involved: the sems implicitly simulated in ? are somewhat sparser than those simulated here.

In the following sections we explore how the performance of SVA and other methods changes as we change certain parameters of the simulated SEM, including the complexity of the functions \( f_{x_j} \) and \( f_{h_k} \) and the sparsity.

### 4.3.3 Sensitivity analysis

We performed univariate sensitivity analysis on the following parameters of the additive gene expression SEMs:

- Dimension of the additive gene expression SEM, specifically parameters \( K \) and \( J \),
- Sparsity of the additive gene expression SEM, specifically parameters \( p_{0k}, p_{0j}, p_{0\beta} \) and \( p_{d-sep} \),
- Variance of noise variables, specifically parameters \( \sigma_{c_l} \) and \( \sigma_{N_{x_j}} \),
- Complexity of \( f_{x_j} \) and \( f_{h_k} \), specifically the maximum degree of the polynomials, and
- Number of observations.
In the following sections all parameters are as described in Section 4.3.1 unless explicitly stated otherwise.

**Dimension of additive gene expression SEM**

We let the number of gene expression level variables take the following values $J \in \{100, 200, ..., 1000\}$.

![Graphs showing sensitivity of parameter $J$](image)

- (a) $c_l$ node span SVA
- (b) $h_k$ node span SVA, SVDR and SVDX
- (c) $y-h_k$ dependence SVA, SVDR and SVDX
- (d) $f_{x_j}$ estimation SVA, SVDR, SVDX and van
- (e) Validity of significance analysis: Nested KS for SVA, SVDR, SVDX and van

Figure 4.8: Sensitivity of parameter $J$
Increasing $J$ from 100 to 200 seems to improve all 5 measures studied for SVA both in absolute terms and in comparison with the other methods. Between $J = 200$ and $J = 500$ the estimation of the $h_k$-node span for SVA is high and has little variance. For higher values of $J$ it is still as high as for SVDR however there is much greater variance. This seems to be reflected in the accuracy of $f_{x_j}$ estimation since for values less than $J = 500$ SVA is the most accurate method but for higher values the median accuracy is for all methods is similar but SVA has much more variance. For the range of $J$ values analyzed the nested KS statistic shows that significance analysis for SVA is much more valid than for the other three methods.
We let the number of gene expression level variables and the number of unobserved factors take the following values \((K, J) \in \{(4, 40), (37, 370), (70, 700), \ldots, (300, 3000)\}\).

Figure 4.9: Sensitivity of parameters \((K, J)\)

Increasing the number of factors \(K\) while keeping the \(K : J\) ratio equal worsens all five measures for SVA both in absolute terms and in comparison with the other methods. For \(K\) up to a value of 40 SVA has better or similar accuracy in the estimation of \(f_{x_j}\) to the other methods but beyond this value it performs consistently worse.
Sparsity of additive gene expression SEM

We let the proportion of values \( k \in \{1, \ldots, K\} \) such that \( f_{h_k}(y) = 0 \) take the following values \( p_{0k} \in \{0, 0.11, 0.22, \ldots, 0.88, 1\} \).

For values of \( p_{0k} \) of 0.78 or higher \( h_k \)-node span estimation, \( h_k - y \) dependence modeling, \( f_{x_j} \) estimation and validity of significance analysis all improve for SVA both in absolute
terms and relative to the other methods. This seems to indicate that for SVA to work better than the other methods the dependence between the $h_k$-node span and the primary variable $y$ must not be too strong.
We let the proportion of values \( j \in \{1, ..., J \} \) such that \( f_{h_j}(y) = 0 \) take the following values \( p_{0j} \in \{0.25, 0.33, 0.42, 0.5, 0.58, 0.67, 0.75, 0.83, 0.92, 1\} \).

For very high values of \( p_{0j} \), when the primary variable only affects 8\% or less of the gene expression levels, the \( h_k \)-node span estimation and the modeling of the \( h_k - y \) dependency for SVA improves significantly resulting in better accuracy in the estimation of \( f_{x_j} \) for this method than the other methods.
We let the proportion of values \((k, j) \in \{1, \ldots, K\} \times \{1, \ldots, J\}\) such that \(\beta_{kj} = 0\) take the following values \(p_\beta \in \{0, 0.11, 0.22, \ldots, 0.99\}\).

(a) \(c_l\) node span SVA

(b) \(h_k\) node span SVA, SVDR and SVDX

(c) \(y-h_k\) dependence SVA, SVDR and SVDX

(d) \(f_{x_j}\) estimation SVA, SVDR, SVDX and van

(e) Validity of significance analysis: Nested KS for SVA, SVDR, SVDX and van

Figure 4.12: Sensitivity of parameter \(p_0\beta\)

Sparsity in the edges that link \(h_k\) nodes and \(x_j\) nodes, represented by the \(\beta_{kj}\) coefficients improves the accuracy of \(f_{x_j}\) estimation for all four methods. This is due to the fact that the \(h_k\) estimations become less important since, as \(p_0\beta\) increases, the \(x_j\) gene expression levels depend less on these variables. For low values of \(p_0\beta\) the estimation accuracy of
$f_{x_j}$ is similar for all four methods however the variance is much higher for SVA. For high values of $p_{03}$ we start to see higher accuracy and similar variance for SVA compared to the other methods.
We let the proportion of $j \in \{1, ..., J\}$ such that $f_{h_j}(y) = 0$ take the following values $p_{0j} \in \{0, 0.11, 0.22, ..., 0.99\}$ and we let the proportion of $j \in \{1, ..., J\}$ such that $x_j \text{ d-sep}_G y$ by $\emptyset$ be the square of the corresponding $p_{0j}$ value, ie we let $(p_{0j}, p_{d-sep}) \in \{(0, 0), (0.11, 0.0121), (0.22, 0.0484), ..., (0.99, 0.9801)\}$.

The sensitivity of all 5 measures to changes in $p_{d-sep}$ behaves similarly to that for changes to $p_{0j}$: for increases in both parameters (individually) estimation of the span of $h_k$ nodes
and of the $h_k - y$ dependence gets worse, but the accuracy of $f_{x_j}$ estimation gets better. As $p_{d\text{-sep}}$ increases, the gene expression levels depend less on the primary variable $y$ and more on the unobserved factors $h_k$, the opposite of what happens as $p_{0\beta}$ increases. In terms of the accuracy of $f_{x_j}$ estimation, it would seems SVA performs better when either the dependence on $y$ or the dependence on the span of $h_k$ is weak. However, this is hard to explain in this case since for high values of $p_{d\text{-sep}}$, the $x_j$ variables depend predominantly on the unobserved factors $h_k$ but these are now poorly estimated. The poor estimation of the span of $h_k$ nodes is due to a poor estimation of the span of $c_l$ nodes. One reason for this could be that there is overfitting of the $f_j(y)$ functions in the initial step of SVA due to the fact that many $x_j$ variables don’t depend on $y$. This would cause a bad estimation of residuals and therefore of the span of $c_l$ nodes. The fact that estimation $f_{x_j}$ estimation for SVA still improves despite the poor estimation of the span of $h_k$ nodes could be explained by the fact that most functions $f_{x_j} = 0$. 
Variance of noise variables

We let the standard deviation of the \( c_l \) noise variables take the following values \( \sigma_{N_{c_l}} \in \{1/10, 1/5, 1/3, 1/2, 1, 2\} \)

- (a) \( c_l \) node span SVA
- (b) \( h_k \) node span SVA, SVDR and SVDX
- (c) \( y-h_k \) dependence SVA, SVDR and SVDX
- (d) \( f_x \) estimation SVA, SVDR, SVDX and van
- (e) Validity of significance analysis: Nested KS for SVA, SVDR, SVDX and van

Figure 4.14: Sensitivity of parameter \( \sigma_{c_l} \)

Greater variance in the noise of \( c_l \) nodes results in poorer estimation of span of \( h_k \) nodes. This leads to more variance in the estimation of the \( h_k - y \) dependency. For low values
4.3 High-dimensional experiments

(0.1-0.5) of $\sigma_{c_l}$ SVA is the most accurate method for estimating $f_{x_j}$ however for higher values (>0.5) it is worse both in terms of the median and the variance. Although SVA is still the best method for producing valid significance analysis even for high levels of $\sigma_{c_l}$ the quality of the significance analysis drops dramatically as this parameter increases.
We let the standard deviation of the $x_j$ noise variables take the following values $\sigma_{N_{x_j}} \in \{1/10, 1/5, 1/3, 1/2, 1, 2, 5, 10\}$

As the $x_j$ node noise variance, $\sigma_{x_j}$ gets larger it is harder to recover the contribution of the $h_k$ and $c_l$ nodes so the estimation of their span worsens, and consequently that of the $h_k - y$ dependency. However, since the $h_k$ nodes constitute an ever decreasing component of the gene expression levels $x_j$ it is possible to model them with or without an estimate for...
$h_k$ and so all methods perform equally well for high $\sigma_{x_j}$ values. This also means that the significance analysis for the multiple hypothesis $H_{j0} : f_{x_j}(y) = 0$ become less dependent and so significance analysis for all 4 methods becomes more valid.
Complexity of $f_{x_j}$ and $f_{h_k}$

We let the maximum degree for any $f_{x_j}(y)$ or $f_{h_k}(y)$ polynomial function take the following values $d^{\text{max}} \in \{1, 2, 3, 4, 5\}$. This is also the order of the polynomial basis function model used in estimation. **Remark**: the actual degree of any polynomial $f_{x_j}(y)$ for $j \in \{1, ..., J\}$ and $f_{h_k}(y)$ for $k \in \{1, ..., K\}$ is sampled uniformly from the set $\{1, ..., d^{\text{max}}\}$.

![Graphs and plots](image)

(a) $c_l$ node span SVA 
(b) $h_k$ node span SVA, SVDR and SVDX 
(c) $y-h_k$ dependence SVA, SVDR and SVDX 
(d) $f_{x_j}$ estimation SVA, SVDR, SVDX and van

(e) Validity of significance analysis: Nested KS for SVA, SVDR, SVDX and van

Figure 4.16: Sensitivity of parameter $d^{\text{max}}$. 

Simulation Experiments
As the maximum order of the polynomials $f_{x_j}$ and $f_{h_k}$ increases the estimation of the $c_l$-node span for SVA deteriorates probably because the variance of the signal from $y$ to $x_j$ dominates the signal coming from the $c_l$ nodes to $x_j$ making it hard to recover the latter. This in turn means that as complexity of $f_{x_j}$ and $f_{h_k}$ increases the $h_k$-node span and $h_k - y$ dependency estimation deteriorates leading to ever poorer $f_{x_j}$ estimation. However, greater $f_{x_j}$ and $f_{h_k}$ complexity lead to poorer $f_{x_j}$ estimation for all four methods. The validity of significance analysis also deteriorated for all four methods as the complexity of $f_{x_j}$ and $f_{h_k}$ increased. This is probably because the dependence of the gene expression levels $x_j$ on $h_k$ is stronger.
Number of observations

We let the number of observations $n$ simulated for each repetition $m \in \{1, 2, \ldots, 100\}$ take the following values $n \in \{25, 133, 242, \ldots, 1000\}$.

As $n$ increases above 500 observations the performance measures for the SVA estimation do not seem to improve, if anything they get worse. This may relate to what ? refer to as
4.3 High-dimensional experiments

a reversal of the curse of dimensionality where a low dimensional kernel, in this case the $K h_k$ unobserved variables, fully captures the dependence structure in an observed high-dimensional data set, in this case the $x_j$ variables. As $n$ increases the data set of $x_j$ variables approaches a non high-dimensional setting so it may become harder to capture the dependence structure with a low dimensional set of variables.

The sensibility analysis carried out is univariate meaning we can’t choose the best parameter values form each sensitivity analysis to obtain a set of parameters where SVA performs ideally or better than the other methods. However assuming that performance measures vary with respect to the different sensitivity parameters somewhat independently this analysis suggests the following parameters for ideal SVA performance:

- **Dimension**: Low number of unobserved factors $h_k$ and high number of gene expression levels $x_j$ ($J$ 20 to 50 times bigger than $K$ based on sensitivity analysis),
- **Sparsity**: High level of sparsity in gene expression SEM favours a high level of accuracy of $f_{x_j}$ estimation for SVA both in absolut terms and in comparison with other methods ($p_{0k} \geq 0.88$, $p_{0j} \geq 0.92$, $p_{0\beta} \geq 0.88$ and $p_{d-sep} \geq 0.77$ as roughly indicated by sensitivity analysis),
- **Variance of noise**: Sensitivity analysis suggests that lower variance for $c_l$ variables than for $y$ variable and a slightly higher variance for $x_j$ variables than for $y$ variable are ideal conditions for SVA both in absolute terms and relative to other methods (sensitivity analysis suggests $\sigma_{c_l} \leq 0.5$, $\sigma_y = 1$ and $\sigma_{N_{x_j}} = 2$ are good values for example),
- **Complexity of $f_{x_j}$ and $f_{h_k}$**: simpler functions lead to better estimation accross all four methods,
- **Number of observations**: There is some evidence that SVA performs better relative to other methods when $n$ is smaller, probably in relation to $K$ and $J$. 
Chapter 5

Summary

In Chapter 1 we introduced the SVA methodology in the context of the modeling of a large number of gene expression levels, $x_j$ with $j \in \{1, ..., J\}$, in terms of an observed primary variable (or vector) $y$ and unobserved factors $g_l$. We justified the use of an additive model of the form in 2.3.0.1 and argued that we must estimate the variables $g_l$ in order to obtain an unbiased estimate of $f_{x,j}(y)$ which is the ultimate goal. We concluded that it is not necessary to obtain an explicit estimate of $g_l$ for $l \in \{1, ..., L\}$ rather we can estimate variables $h_k$ for $k \in \{1, ..., K\}$ such that they generate the same linear space as the $g_l$ variables.

In Chapter 2 we framed the SVA modeling problem as a SEM, graphical model and causality estimation problem. In Section 2.1 we defined DAGs and SEMs, established their relationship and the properties, such as the Markov property, faithfulness and causal minimality, that determine whether a given DAG is a sound and complete map for the independencies implicit in a corresponding SEM. We defined a class of gene expression SEMs (2.1.0.14) that we assume can be used to model gene expression data but concluded that it is too large since it includes SEMs with different unidentifiable DAG structures. In Section 2.2 we defined the interventional distribution of a SEM and used it to define causal effects. We explored causality as a tool for identifying the DAG structure of a SEM and gave some results that help establish the causal relationships between the variables in a SEM with a given DAG. In Section 2.3 we defined a subset class of SEMs denoted additive gene expression SEMs (2.3.0.1) and justified the use of this smaller class to model gene expression data whose behavior, we assumed previously, is not a problem when working with this class. Using the theory developed throughout the chapter we derive some of the independence and causality properties of the additive gene expression SEMs.

Chapter 3 gives a detailed description of the SVA estimation methodology including its implementation in the R package sva. Section 3.1 gives a general overview of the three main steps involved and how they relate to the class of additive gene expression SEMs defined in Section 2.3. Sections 3.2 and 3.3 give a detailed description of the first two steps. The third step simply involves fitting a standard basis function model so a further description is not given. Section 3.2 shows how the span of $c_l$ for $l \in \{1, ..., L\}$, from the additive gene expression SEM, is estimated. In Section 3.2.1 we showed how the first step of SVA, involving fitting a standard basis function model of the form $x_j = f_j(y) + \epsilon$
Summary

and factorizing the residuals, corresponds to estimating the span of the $c_l$ variables from
the additive gene expression SEM. In Section 3.2.2 we detailed the type of standard basis
function model fitted in the R package sva. Section 3.2.3 describes the SVD factorization of
residuals to estimate the span of the $c_l$ variables and its relationship to principal component
analysis (PCA), while in Section 3.2.4 we described the parallel analysis method used to
select the number of factors to be used and detailed how this method is implemented in the
R package sva. Section 3.3 shows how the span of $h_k$ for $k \in \{1, ..., K\}$ from the additive
gene expression SEM is estimated. Section 3.3.1 describes the filtering procedure in general
and remarks on its limitations. Part of this procedure involves finding the signature of $c_l$
variables in the $x_j$ gene expression level variables. Section 3.3.2 describes the $J$
regressions, and subsequent FDR based significance analysis of the corresponding hypotheses, that is
carried out for this purpose. In Section 3.3.3 we develop various concepts related to FDR
such as positive FDR (pFDR), marginal FDR (mFDR), local FDR (IFDR) and q-values,
and show how they can be used to control the FDR at a certain level $\alpha$. We show how
q-values control FDR and describe a method for estimating them conservatively. We also
show how IFDR is estimated and used to decide which hypotheses tests are significant
(discoveries) in the R package sva, as part of the procedure for finding the signature of a
variable $c_l$ in $x_j$ for $j \in \{1, ..., J\}$.

Chapter 4 includes the simulations carried out to evaluate the performance of the SVA
methodology in different, relevant data environments. Section 4.1 includes the description
of three alternate methods for estimating the effect of $y$ on the $x_j$ variables which serve as
benchmarks for SVA. In Section 4.2 we assess the performance of SVA for a low dimensional
additive gene expression SEM, specifically when $J = K = L = 4$. Section 4.2.1 includes
the details of the simulation design. Section 4.2.2 includes the simulation results for $M = 1$
repetition of a simulation with $n = 100$ observations. The results seem to show that SVA
has good performance when the complexity of $f_{x_j}$ and $f_{h_k}$ is similar to the complexity of
the standard basis model chosen. By the same token there is a clear potential for overfitting
for cases when, for example, the complexity of $f_{x_j}$ for at least one $j \in \{1, ..., J\}$, is less
than the standard basis function model chosen. This is a weakness of the methodology
borne of the fact that regularization for large $J$ is costly. In Section 4.2.3 we perform
$M = 1000$ repetitions of the simulation experiment reporting five metrics which evaluate
different aspects of the SVA methodology:

1. $c_l$-node span estimation,
2. $h_k$-node span estimation,
3. dependence between span of $h_k$ nodes and $y$,
4. $f_{x_j}$ estimation accuracy, and
5. validity of significance analysis.

We conclude that in this low dimensional setting, SVA had good performance for metrics
1-3 and 5, but this did not translate into a more accurate estimation of the $f_{x_j}$ effects.

In Section 4.3 we assess the performance of SVA in a high dimensional setting, specifically
when $J = 1000$ and $K = L = 10$. Section 4.3.1 includes the details of a base simulation
design. Section 4.3.2 includes the results of performing $M = 100$ repetitions of the
simulation experiment each with $n = 100$ observations. The results of this base scenario
show that SVA outperforms the SVDR and SVDX methods in terms of $h_k$-node span
estimation and modeling of the dependence between the span of $h_k$ and $y$ and also in
terms of the validity of the significance analysis but this did not translate into consistently better results in terms of the accuracy of the $f_{x_j}$ estimation. In Section 4.3.3 we perform univariate sensitivity analysis with the scenario described in Section 4.3.1 as the base scenario to explore for what type of additive gene expression SEMs and for what parameter settings SVA performs better. We performed sensitivity analysis on the dimension of the additive gene expression SEM ($K$ and $J$ parameters), the sparsity of the additive gene expression SEM ($p_{0k}$, $p_{0j}$, $p_{0\beta}$ and $p_{0\text{sep}}$ parameters), variance of noise variables ($\sigma_{c_l}$ and $\sigma_{N_xj}$ parameters), complexity of $f_{x_j}$ and $f_{h_k}$ (maximum degree of polynomials) and number of observations. We found that, assuming that performance measures vary independently with respect to the different parameters, for gene expression SEMs with low number of unobserved factors $h_k$, high level of gene expression variables $x_j$ (around 20 to 50 times), high sparsity, a low variance ratio $\frac{\sigma_{c_l}}{\sigma_y}$ and low complexity for the signals $f_{x_j}$ and $f_{h_k}$ SVA has superior performance to the other methods considered.

5.1 Future Work

The following is a list of possible ways to extend or improve the work presented here:

- Define the class of additive gene expression SEMs in terms of the more general case where $x_j$ can depend on $x_k$ for $j \neq k$. Also add this feature to simulation experiments,
- Explore the possibility of using Independent Component Analysis (ICA) to factorize and obtain independent and not only uncorrelated $c_l$ variables,
- Understand why the model $x_j = \beta_0 + \beta_1 c_l + \epsilon$ instead of the model $x_j = f_j(y) + \beta_1 c_l + N_j$ is used to find the variables $x_j$ for $j \in \{1, ..., J\}$ with the signature of $c_l$,
- Understand why local false discovery rates instead of q-values are used to find the variables $x_j$ for $j \in \{1, ..., J\}$ with the signature of $c_l$,
- Perform simulations from non additive genetic expression sems and see how good the approximation is to test the approximation of real life gene expression data generating mechanisms with additive gene expression SEMs,
- In simulation experiments include noise for $h_k$ equations and allow $h_k$ nodes to be affected by more than one $c_l$ variable,
- Investigate if the error shown in low-dim experiment is mainly due to overfitting of $f_j(y)$,
- Perform simulations with default conditions closer to those of the simulation of experiments in ? where the functions $f_{x_j}(y)$and$ f_{h_k}(y)$ are simple step functions of the form $g(y) = \mathbb{1}_{y \leq a} + \mathbb{1}_{y > a}$ and the simulated SEMs are sparser. Perform sensitivity analysis with these default conditions as a starting point,
- Provide performance measures for different categories of $x_j$ variables separately: those that depend on $y$ only, on the span of $h_k$ only or on both,
- Separate sensibility analysis of $f_{x_j}$ complexity from $f_{h_k}$ complexity, and
- Perform a multivariate sensitivity analysis based upon different scenarios for the additive gene expression SEM parameters.
Appendices
Appendix A

Proofs

A.1 Proof of Proposition 2.3.0.2

Proof.

1. All paths from $y$ to $c_l$ or from $c_l$ to $c_m$ must go through at least one of four types of $v$-structures:
   - $y \rightarrow h_k \leftarrow c_s$
   - $y \rightarrow x_{j} \leftarrow h_k$
   - $h_k \rightarrow x_j \leftarrow h_r$ or
   - $c_s \rightarrow h_k \leftarrow c_t$.

   Where $j \in \{1, ..., J\}$, $k, r \in \{1, ..., K\}$ and $s, t \in \{1, ..., L\}$. This means that $y$ is d-separated from $c_l$ by the empty set and $c_l$ is d-separated from $c_m$ by the empty set.

2. Suppose $\mathbb{P}^{XUC}$ does not satisfy causal minimality with respect to $G$. Then by Proposition 2.1.0.13 we have that there must exist a variable $w \in \{x_1, ..., x_J, h_1, ..., h_K\}$ with a parent $z \in PA^G_w \subseteq \{y, h_1, ..., h_K, c_1, ..., c_L\}$ such that $w \perp \perp z|PA^G_w \setminus z$.

   a. Suppose $w = h_k$. This means that $z \in PA^G_w \subseteq \{y, c_1, ..., c_L\}$.

      i. Suppose $z = y$. This means that $h_k \perp \perp y|PA^G_{h_k} \setminus y$, but given $PA^G_{h_k} \setminus y \subseteq \{c_1, ..., c_L\}$ we have that $h_k = f_{h_k}(y) + \sum_{l=1}^{L} \gamma_{lk}c_l = f_{h_k}(y) + b$ where $b$ is a constant. Since $y \in PA^G_{h_k}$ we have that $f_{h_k}(y)$ is not constant. This leads to the contradictory conclusion that $h_k \not\perp \perp y|PA^G_{h_k} \setminus y$.

      ii. Suppose $z = c_l$. This means that $h_k \perp \perp c_l|PA^G_{h_k} \setminus c_l$, but given $PA^G_{h_k} \setminus c_l \subseteq \{y, c_1, ..., c_{l-1}, c_{l+1}, ..., c_L\}$ we have that $h_k = f_{h_k}(y) + \sum_{l=1}^{L} \gamma_{lk}c_l = \gamma_{lk}c_l + b$ where $b$ is a constant. Since $c_l \in PA^G_{h_k}$ we have that $\gamma_{lk} \neq 0$. This leads to the contradictory conclusion that $h_k \not\perp \perp c_l|PA^G_{h_k} \setminus c_l$.

   b. Suppose $w = x_j$. This means that $z \in PA^G_w \subseteq \{y, h_1, ..., h_K\}$.
i. Suppose \( z = y \). This means that \( x_j \perp y | PA_{x_j}^G \setminus y \), but given \( PA_{x_j}^G \setminus y \subseteq \{h_1, ..., h_K\} \) we have that \( x_j = f_{x_j}(y) + \sum_{k=1}^{K} \beta_{kj} h_k = f_{x_j}(y) + b \) where \( b \) is a constant. Since \( y \in PA_{x_j}^G \) we have that \( f_{x_j}(y) \) is not constant. This leads to the contradictory conclusion that \( x_j \not\perp y | PA_{x_j}^G \setminus y \).

ii. Suppose \( z = h_k \). This means that \( x_j \perp h_k | PA_{x_j}^G \setminus h_k \), but given \( PA_{x_j}^G \setminus h_k \subseteq \{y, h_1, ..., h_{k-1}, h_{k+1}, ..., h_K\} \) we have that \( x_j = f_{x_j}(y) + \sum_{k=1}^{K} \beta_{kj} h_k = \beta_{kj} h_k + b \) where \( b \) is a constant. Since \( h_k \in PA_{x_j}^G \) we have that \( \beta_{kj} \neq 0 \). This leads to the contradictory conclusion that \( x_j \not\perp y | PA_{x_j}^G \setminus h_k \).

We conclude that \( P^{X_{\mathcal{C}}} \) satisfies causal minimality.

3. We look at the SEM \( \hat{S} \) that results from replacing the equations \( S_{h_k} \) of Definition 2.3.0.1 with \( \hat{S}_{h_k} = \hat{N}_{h_k} \) where \( \hat{N}_{h_k} \sim \mathcal{N}(0, 1) \), say. This means that for \( j \) such that \( \beta_{kj} \neq 0 \) we have:

\[
x_j = f_{x_j}(y) + \sum_{r=1}^{K} \beta_{rj} h_r + N_{x_j} = f_{x_j}(N_y) + \sum_{r\neq k} \beta_{rj} N_{h_r} + N_{x_j} + \beta_{kj} \hat{N}_{h_k} \quad \text{(A.1.0.1)}
\]

Since \( \beta_{kj} \neq 0 \) and the random noise variables are mutually independent it is clear that \( x_j \not\perp h_k \) in \( P^{X_{\mathcal{C}}} \).

4. Since \( y \) has no parents, intervening on \( y \) corresponds to changing \( N_y \) with some \( \hat{N}_y \) and we have:

\[
x_j = f_{x_j}(y) + \sum_{k=1}^{K} \beta_{kj} h_k + N_{x_j} \quad \text{(A.1.0.2)}
\]
\[
= f_{x_j}(y) + \sum_{k=1}^{K} \beta_{kj} f_{h_k}(y) + \sum_{l=1}^{L} \gamma_{lk} c_l + N_{h_k} + N_{x_j} \quad \text{(A.1.0.3)}
\]
\[
= \left( f_{x_j}(y) + \sum_{k=1}^{K} \beta_{kj} f_{h_k}(y) \right) + \left( \sum_{k=1}^{K} \sum_{l=1}^{L} \beta_{kj} \gamma_{lk} c_l + \sum_{k=1}^{K} \beta_{kj} N_{h_k} \right) + N_{x_j} \quad \text{(A.1.0.4)}
\]
\[
= f_j(y) + \left( \sum_{k=1}^{K} \sum_{l=1}^{L} \beta_{kj} \gamma_{lk} c_l + \sum_{k=1}^{K} \beta_{kj} N_{h_k} \right) + N_{x_j} \quad \text{(A.1.0.5)}
\]
\[
= f_j(\hat{N}_y) + \left( \sum_{k=1}^{K} \sum_{l=1}^{L} \beta_{kj} \gamma_{lk} c_l + \sum_{k=1}^{K} \beta_{kj} N_{h_k} \right) + N_{x_j} \quad \text{(A.1.0.6)}
\]

Where \( f_j(y) := f_{x_j}(y) + \sum_{k=1}^{K} \beta_{kj} f_{h_k}(y) \). Since \( f_j(y) \) is assumed not to be constant and the random noise variables are mutually independent it is clear that \( x_j \perp y \) in \( P^{X_{\mathcal{C}}} \).
Appendix B

R Code

We include the code in R that was used to produce all results in this work.

B.1 Functions

```r
# SVA PACKAGE FUNCTIONS (MODIFIED BY EMILIANO DIAZ)
# A function for estimating surrogate variables with the two step approach of
# Leek and Storey 2007
# Taken from R package sva and modified.
# @param dat : The transformed data matrix with the variables in rows and samples
# in columns
# @param y: Primary variable observations
# @param mod : The model matrix being used to fit the data
# @param n. sv .r: The number of surrogate variables to estimate for SVA and SVDR
# @param n. sv .x: The number of surrogate variables to estimate for SVDX
# @param beta : The matrix of beta coefficients linking hk to aj nodes.
# If passed SVA doesnt have to do regression analysis to figure out enriched
# matrix.
# @param trace : whether or not to print out a trace of steps carried out
# @return sv : The estimated surrogate variables , one in each column
# @return svd .x: the result of performing SVD on X matrix
# @return svd .r:
# @return svd .r. red :
# @return pprob . gam : A vector of the posterior probabilities each gene is
# affected by heterogeneity
# @return pprob .b: A vector of the posterior probabilities each gene is affected
# by mod (this is always null for the two-step approach)
# @return n. sv .r: The number of significant surrogate variables
# @return n. sv .x:
# @return n. sv .r. ini :
# sva .mod ← function (dat , y, mod , n. sv .r, n. sv .x, beta=NULL , trace=FALSE){
# if(trace) print("enters sva.mod function")
# m ← ncol(dat)
# n ← nrow(dat)
# n. sv .r. ini ← n. sv .r
# # Calculate Regression Hat Matrix
# H ← mod %*% solve(t(mod) %*% mod) %*% t(mod)
# # Calculate residuals R = X - S = X - H*E(Y)
```

83
res ← dat - t(H %*% t(dat))
# Singular value decomposition of residuals and data
uu.r ← svd(res)
uu.x ← svd(dat)

# Construction of T statistic for selection of number of eigenvalues.
# However this is actually not used here. We can ignore or even comment
# out next two lines of code.
# ndf ← n - ceiling(sum(diag(H)))
# dstat ← uu$d[1:ndf]^2/sum(uu$d[1:ndf]^2)

# Obtain the n.sv right eigenvectors of residuals - ie. the residual eigengenes
res.sv ← as.matrix(uu.r$v[,1:n.sv.r])
x.sv ← as.matrix(uu.x$v[,1:n.sv.x])
cors ← as.numeric(cor(x.sv, y))
index.max ← which.max(abs(cors))

# For each eigengene, an "enriched" data matrix will be constructed.
# Use . var will indicate which expression genes each data matrix should be
# constructed from.
# pp[,i] includes p-values for regressing each of the m variables against
# residual eigengene i.

if(!is.null(beta)){
  use.var ← t(beta) != 0
} else{
  use.var ← matrix(rep(FALSE, n.sv.r*m), nrow=m, ncol=n.sv.r)
  pp ← matrix(rep(FALSE, n.sv.r*m), nrow=m, ncol=n.sv.r)
  # we search for the gene expression variables x_i, which will make up the
  # enriched data matrix
  # for each residual eigengene e_k
  for(i in 1:n.sv.r) {
    #print(i)
    # We want to test if residual eigengene i is associated with any of the m
    # expression variables
    # so we construct the null model - matrix which corresponds to the model X = mu
    # + epsilon and
    # the model matrix corresponding to the model X = mu + e_i + epsilon
    mod ← cbind(rep(1,n),res.sv[,i])
    mod0 ← cbind(rep(1,n))
    # Obtain the m p-values corresponding to the models X = mu + e_i + epsilon (we
    # regress e_i on each x_j, j=1,...,m)
    pp[,i] ← p.value(dat,mod,mod0)
  }

  # we only use variables whose p-values have a local FDR rate of less than
  # 10%.
  # I don't know exactly what local FDR is but it might be beyond the scope of
  # my project.
  sigVars ← edge.lfdr(p=pp[,i]) < 0.10
  use.var[,i] ← sigVars

  hist(pp[,i], main=paste("num. of xs sig. with d ", i, " = ", sum(sigVars)),
       xlab="p-values",ylab="frequency")
  #lambda.t ← seq(0,0.999, length.out=100)
  #p0.t ← sapply(lamda.t, function(lam) sum(pp[,i] > lam)/(m*(1-lam)))
  #plot(lambda.t, p0.t)
If the number of gene expression variables chosen for which a certain residual eigengene $e_k$ is a factor is less than $m$ then we don’t take into account this residual eigengene and we adjust the number of eigen values. This isn’t in original algorithm. Might have to take this out at least for when $m < n$.

```r
for (i in ncol(use.var):1) {
  if (sum(use.var[,i]) <= 1) {
    use.var ← as.matrix(use.var[, -i])
    n.sv.r ← n.sv.r - 1
    res.sv.red ← matrix(res.sv.red[, -i], n, n.sv.r)
    if (n.sv.r <= 0) {break}
  }
}

mk ← apply(use.var, 2, sum)

# For each residual eigengene we form its corresponding enriched gene expression matrix, perform svd on it, and take the eigengene most correlated with the eigengene.
if (n.sv.r > 0) {
  # Initialize the surrogate variable matrix.
  sv ← matrix(0, nrow = n, ncol = n.sv.r)
  # Center matrix by rows (genes).
  dat ← t(scale(t(dat), scale = FALSE))

  mk.diff ← unique(mk)
  list.track.xs ← lapply(1:length(mk.diff), function(el) as.character())
  list.track.sv ← lapply(1:length(mk.diff), function(el) as.numeric())
  take.off ← as.numeric()
  for (i in 1:n.sv.r) {
    # List the columns associated with each surrogate eigengene.
    indx.mk ← match(mk[i], mk.diff)
    vars.mk ← which(use.var[, i])
    vars.char ← paste(vars.mk, collapse = ".")
    uu.ex ← svd(dat[use.var[, i], ], n, mk[i])
    maxcor ← 0
    # We only consider first n-1 eigengenes from enriched matrix and look for one with the most correlation to corresponding residual eigengene.
    for (j in 1:min(n, mk[i])) {
      # If the number of gene expression variables chosen for which a certain residual eigengene $e_k$ is a factor
      ifelse(mk[i] == 1, v.eig ← uu.ex$u[, j], v.eig ← uu.ex$v[, j])
      if (abs(cor(v.eig, res.sv.red[, i])) > maxcor) {
        maxcor ← abs(cor(v.eig, res.sv.red[, i]))
        sv[, i] ← v.eig
      }
      choose.sv ← j
    }
  }
}
```
if(any(vars.char %in% list.track.xs[[indx.mk]] & choose sve[[indx.mk]])) {
  take.off[c(take.off, i)]
} else {
  list.track.xs[[indx.mk]] <- c(list.track.xs[[indx.mk]], vars.char)
  list.track.sv[[indx.mk]] <- c(list.track.sv[[indx.mk]], choose.sv)
}

if(length(take.off)>0){
  sv <- matrix(sv[,-take.off], n, n.sv.r-length(take.off))
  res.sv.red <- matrix(res.sv.red[,take.off], n, n.sv.r-length(take.off))
  use.var <- matrix(use.var[,-take.off], m, n.sv.r-length(take.off))
  n.sv.r <- n.sv.r - length(take.off)
}

pprob.gam <- use.var %*% rep(1, n.sv.r) > 0
retval <- list(sv=sv, svd.x=x.sv, svd.r=res.sv, svd.r.red=res.sv.red, pprob.gam=pprob.gam, pprob.b=NULL, n.sv.r=n.sv.r, n.sv.x=n.sv.x, n.sv.r.ini=n.sv.r.ini)
if(trace) print("exits sva.mod function")
return(retval)
}

f.pvalue <- function(dat, mod, mod0) {
  n <- dim(dat)[2]
  m <- dim(dat)[1]
  df1 <- dim(mod)[2]
  df0 <- dim(mod0)[2]
  p <- rep(0, m)
  Id <- diag(n)
  resid <- dat %*% (Id - mod %*% solve(t(mod) %*% mod) %*% t(mod))
  rss1 <- rowSums(resid*resid)
  rm(resid)
  resid0 <- dat %*% (Id - mod0 %*% solve(t(mod0) %*% mod0) %*% t(mod0))
  rss0 <- rowSums(resid0*resid0)
  rm(resid0)
  fstats <- (rss0 - rss1)/(df1-df0)/(rss1/(n-df1))
  p <- 1-pf(fstats, df1=(df1-df0), df2=(n-df1))
  return(p)
# A function for calculating the number of surrogate variables to estimate in a model

# This function estimates the number of surrogate variables that should be included
# in a differential expression model. The default approach is based on a permutation
# procedure originally proposed by Buja and Eyuboglu 1992. The function also
# provides
# an interface to the asymptotic approach proposed by Leek 2011 Biometrics.

# @param dat: The transformed data matrix with the variables in rows and samples
# in columns
# @param mod: The model matrix being used to fit the data
# @param method: One of "be" or "leek" as described in the details section
# @param vfilter: You may choose to filter to the vfilter most variable rows
# before performing the analysis
# @param B: The number of permutations to use if method = "be"
# @param seed: Set a seed when using the permutation approach

# @return n.sv: The number of surrogate variables to use in the sva software

num.sv <- function(dat, mod, method = c("be", "leek"), vfilter = NULL, B = 20, seed = NULL) {
  
  # mod is the basis predictors which are functions of y... its in n x p
  # dat is the multivariate X response variables........... its in n x J

  if(!is.null(vfilter)){
    if(vfilter < 100 | vfilter > dim(dat)[1]){
      stop(paste("The number of genes used in the analysis must be between 100 and",dim(dat)[1],"\n"))
    }
    tmpv = rowVars(dat)
    ind = which(rank(-tmpv) < vfilter)
    dat = dat[ind,]
  }

  method <- match.arg(method)
  if(method == "be"){
    if(is.null(seed)){set.seed(seed)}
    warn <- NULL
    n <- ncol(dat)
    m <- nrow(dat)
    H <- mod %*% solve(t(mod) %*% mod) %*% t(mod)
    res <- dat - t(H %*% t(dat))
    uu <- svd(res)

    # rank(R) = rank(X - H*%*%X) = rank((I-H)X) where rank(X) = min(J,n), rank(I-H) =
    # tr(I-H) = tr(I) - tr(H) = n - tr(H)
    # rank(X - HX) <= rank(X) - rank(HX) = min(m,n) -

    # ndf <- min(m,n) - ceiling(sum(diag(H)))
    ndf <- min(min(m,n), ceiling(abs(min(m,n) - sum(diag(H)))))
    dstat = uu$d[1:ndf]^2/sum(uu$d[1:ndf]^2)
    dstat0 = matrix(0, nrow = B, ncol = ndf)

    for(i in 1:B){
      # permute ORIGINAL residuals according to Bth sample permutation
      res0 <- t(apply(res, 1, sample, replace = FALSE))
      # the permutation may have created certain "artificial" dependence between
      # residuals and predictors X, so we apply model again projecting on to X with
      # H matrix and ensuring residuals are again orthogonal to z

    }
  }
}


```r
res0 <- res0 - t(H %*% t(res0))
uu0 <- svd(res0)

# % of variance in corresponding principal component
dstat0[i,] <- uu0$d[1:ndf]^2/sum(uu0$d[1:ndf]^2)

psv <- rep(1, n)
for (i in 1:ndf){
  # for each eigenvalue we see what fraction of the repetitions was higher with
  # the permutations than without. If its
  # truly a non zero eigenvalue then we expect the non-permuted eigen-value to be
  # higher most of the time so the corresponding
  # psv value should be low
  psv[i] <- mean(dstat0[,i] >= dstat[i])
} for (i in 2:ndf){
  # we make sure psv is its non-decreasing: true eigenvalues are decreasing so %
  # of times "random" eigenvalues are bigger than true abs to be
  # increasing
  psv[i] <- max(psv[(i-1)], psv[i])
}

nsv <- sum(psv <= 0.10)
return(as.numeric(list(n.sv = nsv)))
}

else{
  dat <- as.matrix(dat)
dims <- dim(dat)
a <- seq(0, 2, length = 100)
n <- floor(dims[1]/10)
rhat <- matrix(0, nrow = 100, ncol = 10)
P <- (diag(dims[2]) - mod %*% solve(t(mod) %*% mod) %*% t(mod))
for (j in 1:10){
  dats <- dat[1:(j*n),]
  ee <- eigen(t(dats) %*% dats)
sigbar <- ee$values[dims[2]]/(j*n)
  R <- dats %*% P
  wm <- (1/(j*n))*t(R) %*% R - P*sigbar
  ee <- eigen(wm)
v <- c(rep(0, bumpstart), ss[(bumpstart+1):100])
v <- v[order(c(a*(j*n)^(1/3)*dims[2], ee$values), decreasing = TRUE)]
  u <- 1:length(v)
w <- 1:100
  rhat[,j] <- rev((u[v==TRUE]-w))
}
  ss <- rowVars(rhat)
  bumpstart <- which.max(ss > (2*ss[1]))
  start <- which.max(c(rep(1e5, bumpstart), ss[(bumpstart+1):100]) < 0.5*ss[1])
  finish <- which.max(c(ss+c(rep(0, start), rep(1, 100-start)) > ss[1])
  if(finish==1){finish <- 100}
  n.sv <- modefunc(rhat[start:finish,10])
  return(n.sv)
  print(method)
}
```
```r
# ' @return n.sv: The number of surrogate variables to use in the sva software
# '
edge.lfdr <- function (p)
{
    # default parameters
    trunc = TRUE
    monotone = TRUE
    adj = 1.5
    eps = 10^-8
    lambda = 0.8

    # estimate pi_0, the proportion of null genes (out of the m genes)
    pi0 <- mean(p >= lambda)/(1 - lambda)
    pi0 <- min(pi0, 1)

    # calculate number of p values
    n = length(p)

    # we bound p-values below to eps=10^-8
    p = pmax(p, eps)

    # we bound p-values above to 1-eps
    p = pmin(p, 1 - eps)

    # we transform values in (0,1) to (-inf,inf) using F_norm_std
    x = qnorm(p)

    # calculates a non-parametric, gaussian-kernel density estimator for
    # transformed p-values
    myd = density(x, adjust = adj)
    # plot(myd$x, myd$y, type="l")

    # we fit a spline to the kernel-density estimator
    mys = smooth.spline(x = myd$x, y = myd$y)
    # plot(mys$x, mys$y, type="l")

    # we obtain the value of the density at the transformed p-values
    y = predict(mys, x)$y
    # indx <- order(x)
    # plot(x[indx], y[indx], type="l")

    # with dnorm(x)/y we compare how far our non-parametric density is from the
    # normal
    # under the null (model x_i = mu + ek + epsilon) our pvalues are distribute
    # uniformly so x
    # should be distributed normally and dnorm(x)/y should be close to 1
    lfdr = pi0 * dnorm(x)/y

    # by default trunc=TRUE
    if (trunc) {
        lfdr[lfdr > 1] = 1
    }

    # by default monotone=TRUE
    if (monotone) {
        lfdr = lfdr[order(p)]
        lfdr = mono(lfdr)
        lfdr = lfdr[rank(p)]
    }
    return(lfdr)
}
```

---

# A function for making the local false discovery rates of vector of p-values

#'
# @param lfdr: vector of local false discovery rates
#'
# # "
# @return: monotone local false discovery rates
#
mono ← function(lfdr)
{
  .Call("monotone", as.numeric(lfdr), PACKAGE = "sva")
}

# FUNCTIONS BY EMILIANO DIAZ

# A function for simulating the parameters of an genetic expression SEM with polynomial functions fxj and fhk
#
# This function simulates the coefficients of the f_x(y) and f_h(y) polynomial functions
# and also the linear coefficients beta. It does it for the hole SEM: "y" node, K "h" and
# "c" nodes and J "x" nodes.
#
# @param K: The number of "c" and "h" nodes (number of eigengenes)
# @param J: The number of "x" nodes (number of genes)
# @param deg.max: The maximum degree of the polynomial functions
# @param trace: whether or not to print out a trace of steps carried out
#
# @return: sem.lin A list with:
# fhd - a k-length vector with the degree of polynomials 1,...,k
# fh.a - a k x deg.max matrix with the coefficients of polynomials 1,...,k
# fx.d - an m-length vector with the degree of polynomials 1,...,m
# fx.a - an m x deg.max matrix with the coefficients of polynomials 1,...,m
# beta - a k x m matrix with the coefficients of all possible links from h nodes to x nodes.
#
sim.sem ← function(K, J, deg.max, trace=FALSE){
  if(trace) print("enters sim.sem function")
  fh.d ← sample(1:deg.max, size=K, replace=T)
  fh.a ← matrix(rnorm(n=deg.max*K), nrow=K, ncol=deg.max)
  fx.d ← sample(1:deg.max, size=J, replace=T)
  fx.a ← matrix(rnorm(n=deg.max*J), nrow=J, ncol=deg.max)
  beta ← matrix(rnorm(J*K), K, J)
  sem.lin ← list(fh.d=fh.d, fh.a=fh.a, fx.d=fx.d, fx.a=fx.a, beta=beta)
  if(trace) list(fh.d=fh.d, fh.a=fh.a, fx.d=fx.d, fx.a=fx.a, beta=beta)
  return(sem.lin)
}

# In this case we can control the sparsity of the SEM with additional parameters
#
# This function simulates the coefficients of the f_x(y) and f_h(y) polynomial functions
# and also the linear coefficients beta. It does it for the hole SEM: "y" node, K "h" and
# "c" nodes and J "x" nodes.
#
# @param K: The number of "c" and "h" nodes (number of eigengenes)
# @param J: The number of "x" nodes (number of genes)
# @param deg.max: The maximum degree of the polynomial functions
# @param p0_fh: The proportion of fh(y)=0 functions
# @param p0_fz: The proportion of fz(y)=0 functions
# @param p0_isox: The proportion of xs d-separated from y
# @param p0_beta: The minimum proportion of betas = 0
# @param trace: In case you want to pass a specific beta matrix for the sem
#
sim.sem.p0 ← function(K, J, deg.max, p0_fh, p0_fz, p0_isox, p0_beta, trace=FALSE){
  if(trace) print("enters sim.sem.p0 function")
  if(p0_fh < 0) stop("p0_fh must be non-negative")
  if(p0_fz < 0) stop("p0_fz must be non-negative")
  if(p0_isox < 0) stop("p0_isox must be non-negative")
  if(p0_beta < 0) stop("p0_beta must be non-negative")
  if(p0_fh + p0_fz + p0_isox + p0_beta > 1) stop("The proportions must sum to 1")
  if(p0_fh + p0_fz > 1) stop("p0_fh + p0_fz cannot exceed 1")
  if(p0_isox + p0_beta > 1) stop("p0_isox + p0_beta cannot exceed 1")
  if(p0_isox > 0) p0_isox <- 1
  if(p0_beta > 0) p0_beta <- 1
  n_fh <- floor(K * p0_fh)
  n_fz <- floor(J * p0_fz)
  n_isox <- floor(K * p0_isox)
  n_beta <- floor(K * p0_beta)
  fh.d ← sample(1:deg.max, size=n_fh, replace=T)
  fh.a ← matrix(rnorm(n=deg.max*K), nrow=K, ncol=deg.max)
  fx.d ← sample(1:deg.max, size=n_fz, replace=T)
  fx.a ← matrix(rnorm(n=deg.max*J), nrow=J, ncol=deg.max)
  beta ← matrix(rnorm(J*K), K, J)
  sem.lin ← list(fh.d=fh.d, fh.a=fh.a, fx.d=fx.d, fx.a=fx.a, beta=beta)
  if(trace) list(fh.d=fh.d, fh.a=fh.a, fx.d=fx.d, fx.a=fx.a, beta=beta)
  return(sem.lin)
}
# @return sim.sem.sparse A list with:
# fhd - a k-length vector with the degree of polynomials 1,...,k
# fha - a k x deg.max matrix with the coefficients of polynomials 1,...,k
# fxd - an m-length vector with the degree of polynomials 1,...,m
# fxa - an m x deg.max matrix with the coefficients of polynomials 1,...,m
# beta - a k x m matrix with the coefficients of all possible links from h nodes to x nodes
# effect.yh - in case of deg.max=1 (linear functions) the effect of y on h
# effect.yx - in case of deg.max=1 (linear functions) the effect of y on x
#
# sim.sem.sparse ← function(K, J, deg.max, p0_fh=0, p0_fx=0, p_isox=0, p0_beta=0, 
# beta=NULL, trace = FALSE)
# if trace) print("enters sim.sem.sparse function")
# if there are less j such that faj(y)=0 than p_isox*J then we cant keep p_isox* J zjs d-separated from y
# if(p_isox > p0_fx) stop("cannot have more isolated xjs than f0=0")
#number of ks such that fhk(y)=0
#fh.d is the degree of the polynomial. We fix the first numfhds0 as 0
fh.d ← ceiling(p0_fh*K)
#fh.a is the degree of the polynomial. We fix the first numfhds0 as 0
fh.a ← matrix(rnorm(deg.max*K), K, deg.max)
#number of js such that faj(y)=0
#fx.d is the degree of the polynomial. We fix the first numfjds0 as 0
fx.d ← ceiling(p0_fx*J)
#fx.a is the degree of the polynomial. We fix the first numfjds0 as 0
fx.a ← matrix(rnorm(n=deg.max*J), nrow=J, ncol=deg.max)
if(is.null(beta)){
# we create a J*K matrix with the beta coefficients p0_beta% of which are
# randomly zero.
num.beta0 ← ceiling(p0_beta*J*K)
um.beta0 ← J*K - num.beta0
beta.vec ← c(rep(0, num.beta0), rnorm(num.beta.not0))
beta.vec ← beta.vec[sample(1:(J*K), size=J*K, replace=F)]
beta ← matrix(beta.vec, K, J)
# additionally we want p_isox% of the J z's to be d-separated from y. We have already checked that p_isox >= p0_fz so we know that we can achieve this by controlling the betas. For J*p_isox of the j's such that faj=0, we need to make sure that for all k such that fhk != 0 then beta_kj = 0.
#first we calculate the number of j's this is. Since we know the first p0_fz* J js are such that faj=0 we now that the js we are looking for are 1:num.beta1.0
num.beta1.0 ← ceiling(p0_isox*J)
# we just check that not all the fhk =0, because in that case we don't need to do anything as the number of d-separated zjs from y will be p0_fz*J
if(num.fhds0 < K){
# beta has hs in rows and xs in columns
# for rows which correspond to all non zero fhk's then we make sure that
# first num.beta.0 columns have beta_kj = 0
mat.indx <- expand.grid(index.row=(num.h.ds0+1):K, index.col=1:num.beta.0)
beta[as.matrix(mat.indx)] <- 0
}

if(trace) print(paste("%0s beta : ", round(sum(beta==0)/(J*K),2)))

# for linear polynomials we can easily calculate the effect of y on h and
# after that on x
effect.yh <- NULL
effect.yx <- NULL
if(deg.max==1){
effect.yh <- as.numeric(fh.a)
effect.yx <- as.numeric(fx.a) + sapply(1:J, function(i) sum(sapply(1:K, function(j) fh.a[j,i]*beta[j,i])))
}

sem.lin <- list(fh.d=fh.d, fh.a=fh.a, fx.d=fx.d, fx.a=fx.a, beta=beta, effect.yh=effect.yh, effect.yx=effect.yx)
if(trace) print(" exits sim.sem.sparse function")
return(sem.lin)

########################################################################
# ' A function for evaluating multiple polynomials
#'
# ' This function takes multiple polynomials represented in matrix form, one row
# ' per polynomial
# ' one column per degree of the term of the polynomial, and evaluates each
# ' polynomial at
# ' every value of the vector x
#'
# '@param x: a vector of values at which we wish to evaluate the polynomials
# '@param d: the degree of each polynomial, one for each row of a
# '@param a: a matrix with the coefficients of each polynomial, one polynomial
# ' per row
# '@param trace : whether or not to print out a trace of steps carried out
#'
# '@return P: A matrix where the i,j term corresponds to the j-th polynomial
# ' evaluated at the i-th observation
polyn <- function(x, d, a, trace=FALSE){
  if(trace) print("enters polyn function")
  d.max <- dim(a)[2]
  n <- length(x)
  K <- length(d)

  # we create a matrix which indicates for each degree if the corresponding
  # coefficient
  # is zero or non-zero, so that we can set it to zero in a
  Y <- matrix(sapply(d, function(y) c(rep(1,y), rep(0,d.max-y))), dim(a)[2], dim(a)[1])

  # we correct 'a' so that zero-coefficient terms are fixed at zero
  a.aux <- a*t(Y)

  # we create a matrix with x elevated to 1,2,...,deg.max
  X <- t(matrix(sapply(x, function(y) y^(1:d.max)),d.max, n))

  # we evaluate the polynomials by applying the dot product of each row of a.aux
  # with each row of X
  P <- t(sapply(1:n, function(i) sapply(1:K, function(K) sum(X[i,]a.aux[K,]))))
if (trace) print("exits sim.sem function")
return(P)
}

# #######################################################################
# ' A function for simulating an additive genetic expression SEM
# ' This function simulates a sem with the structure described in section 4.3.1 (see DAG therein)
# ' The sem must be specified in a list such as is produced by the functions sim.sem or sim.sem.sparse.
# ' @param sem: an additive gene expression SEM specified in a list.
# ' @param y.c. signal: the ratio of the standard deviation of the y variable to the
# ' standard deviation of the c variables which is 1.
# ' @param trace: whether or not to print out a trace of steps carried out
# ' @return a list with:
# ' y - observations for y variable
# ' cs - observations for c_l variables for all l
# ' hs - observations for h_k variables for all k
# ' xs - observations for x_j variables for all j
# ' xs0 - observations for x_j variables for all j under null: f(x_j)(y)=0.
# ' fx - observations of y evaluated at f(x_j) for all j
# ' exp.xs - expected value of x_j observations given y and h_k observations (i.e. without white noise)
# ' fh - observations of y evaluated at f(h_k) for all k
# ' fg - observations of y evaluated at f(x_j) for all j
# ' nx - observations of N_{x_j} for all j
# ' J ← dim(sem$fx.a)[1]
# ' x ← matrix(rnorm(n,J,0),n,J)
# ' cs ← matrix(rnorm(n*K,0,1),n,K)
# ' fh ← polyn(x,y, d=sem$fh.d, a=sem$fh.a, trace)
# ' hs ← cs + fh
# ' fx ← polyn(y, sem$fx.d, sem$fx.a, trace)
# ' exp.xs ← fx + hs%*%sem$beta
# ' xs ← exp.xs + nx
# ' xs0 ← hs%*%sem$beta + nx
if (trace) print("exits sim.lin function")
return(list(y=y,cs=cs ,hs=hs , xs=xs , xs0 =xs0 , fx=fx , exp.xs=exp.xs , fh=fh , fg=fg , nx=nx))

# #######################################################################
# ' A function for calculating p-values for J hypothesis tests H_{(j0): f(x_j)(y)=0}
# ' This function calculates the J p-values for multiple hypotheses H_{(j0): f(x_j)(y)=0}
# ' from the regression model x_j = f(x_j)(y) + sum_{l=1}^L beta_{l}*c_l + epsilon_j where j in {1,...,J}
# ' @param mod: The model matrix being used to fit the data including standard
# ' basis variables from standard basis function model f(y) and c_l vars
# ' @param ord: order of the polynomial standard basis model f(y)
# ' @param trace: whether or not to print out a trace of steps carried out
# ' @return p.val: The J pvalues one for each regression model
# ' pval ← function(mod, x, ord, trace=FALSE){
if (trace) print("enters pval function")

n ← dim(x) [1]
C ← diag(dim(mod) [2])
C ← matrix(C [2:(ord+1),, ord, dim(C) [2]])
q ← dim(C) [1]
r ← dim(C) [2]
g ← rep(0, q)
beta_hat ← solve(t(mod) %*% mod) %*% t(mod) %*% x
# print("beta_hat")
# print(beta_hat)
res ← mod %*% beta_hat - x
se2 ← diag(t(res) %*% res) / (n - r)
# to match R's calculation we divide by n - r instead of n - r - 1
# print("se2")
# print(se2)
Fstat ← diag(t(C %*% beta_hat - g) %*% solve(C %*% solve(t(mod) %*% mod) %*% t(C)) %*% (C %*% beta_hat - g)) / (q * se2)
# print("Fstat")
# print(Fstat)
p.val ← 1 - pf(Fstat, df1 = q, df2 = n - r)
if(trace) print("exits pval function")
return(p.val)
}

# #######################################################################
# ' A function for calculating the mean absolute error in the estimation of f_{x_j}(y) at points y_1 ,..., y_n
# ' This function calculates the mean absolute error in the estimation of f_{x_j}(y). The real values of f_{x_j}(y)
# ' are passed and also the estimated ones in the form of the matrix mod which contains the standard basis model for y.
# ' @param mod : The model matrix being used to fit the data consisting of standard basis variables from standard basis function model f(y)
# ' @param x: x_j ariable matrix in reals^{ n x J}
# ' @param fx . real : the real function f_{x_j}(y) evaluated at observed points y_1 ,..., y_n
# ' @param ord : order of the polynomial standard basis model f(y)
# ' @param trace : whether or not to print out a trace of steps carried out
# ' @return err . fx : The mean absolute error of hat_{f_{x_j}(y)} at y_1 ,... , y_n
# '
fx.err ← function (mod, x, fx.real, ord, trace = FALSE){
if(trace) print("enters fx.err function")
beta_hat ← solve(t(mod) %*% mod) %*% t(mod) %*% x
# print("beta_hat")
# print(beta_hat)
fx ← mod [1:(ord+1)] %*% beta_hat [1:(ord+1),]
err.fx ← apply(abs(fx - fx.real), 2, mean)
if(trace) print("exits fx.err function")
return(err.fx)
}

# #######################################################################
# ' A function for calculating the 5 performance measures for the different methods (where possible): SVA, SVDR, SVDX, VANILLA
# '
# ' With the simulated gene expression SEM, the standard basis variables and the estimated surrogate variables
# ' this function constructs the various performance measures: estimation of c_l node span, estimation of h_k node span,
# ' y-span(h_k) dependence, f_{x_j} estimation and validity of significance analysis.
# '
# ' @param sem . sim : simulation of sem which includes a simulation under the null ( f_{x_j}(y)=0 for all j) and under alternative.
# ' @param sva . lin : estimated SVA model, including surrogate variables, with alternative hypothesis simulation.
B.1 Functions

```r
#' @param sva.lin0: estimated SVA model, including surrogate variables, with null hypothesis simulation
#' @param mod: The model matrix being used to fit the data consisting of standard basis variables from standard basis function model \( f(y) \)
#' @param ord: order of the polynomial standard basis model \( f(y) \)
#' @param trace: whether or not to print out a trace of steps carried out

#' @return a list with:
# ccor.cs.sva - % overlap between estimated c\_l-node span and real c\_l-node span. Using canonical correlation analysis (CCA) we measure what % of the linear span of c\_l and \( \hat{c}_l \) is shared. Since SVA is the only method that estimates the span of c\_l differently to the span of h\_k we only measure this for the SVA method.
# ccor.hs - % overlap between estimated h\_k-node span and real h\_k-node span (using CCA). In this case we are able to compare the SVA, SVDR and SVDX methods since each has an estimate for h\_k
# r2yh - \( R^2 \) between real y and h\_k nodes minus \( R^2 \) between real y and estimated h\_k nodes. The \( R^2 \) corresponds to a simple linear regression with y as the independent variable and h\_k or \( \hat{h}_k \) as the dependent variables.
# err.fx - Mean absolute error in estimation of \( f_{(x,j)}(y) \) at \( y_1, \ldots, y_n \)
# pval0 - P-values for hypothesis test \( H_{(j0)}: f_{(x,j)}(y) = 0 \) under null (that is we simulated \( f_{(x,j)}(y) = 0 \) for all \( j \))
# ext.rate - Number of C-nodes extracted by sva algorithm as a % of the initial number detected by num.sv

# measures.sem <- function(sem.sim, sva.lin, sva.lin0, mod, ord, trace=FALSE){
if(trace) print("enters measures.sim")
K <- dim(sem.sim$cs)[2]

# I. sum of K max correlations between real and estimated c-nodes (only for SVA)
# SVA
if(trace) print("calculate c-node overlap")
cancor.cs.sva <- sum(cancor(sem.sim$cs, sva.lin$ svd.r, xcenter = TRUE, ycenter = TRUE)$cor)/K

# II. sum of K max correlations between real and estimated h-nodes (surrogate variables)
# SVA
if(trace) print("calculate h-node overlap")
cancor.hs.sva <- sum(cancor(sem.sim$hs, sva.lin$sv, xcenter = TRUE, ycenter = TRUE)$cor)/K

cancor.hs.svdx <- sum(cancor(sem.sim$hs, sva.lin$svd.x, xcenter = TRUE, ycenter = TRUE)$cor)/K

# SVD.R

cancor.hs.svdr <- sum(cancor(sem.sim$hs, sva.lin$svd.r, xcenter = TRUE, ycenter = TRUE)$cor)/K

# ALL
cancor.hs <- c(sva=cancor.cs.sva, svdx=cancor.hs.svdx, svdr=cancor.hs.svdr)

# III. Measure correlation between y and real h-nodes and y and estimated h-nodes
if(trace) print("estimate y-h correlation")
r2yh.h <- summary(lm(sem.sim$y~sem.sim$hs))$r.squared
rsqv
r2yh.hhat <- summary(lm(sem.sim$y~sva.lin$sv))$r.squared
r2yh.sva <- r2yh.h - r2yh.hhat

cancor.hs.svdx <- summary(lm(sem.sim$y~sva.lin$svd.x))$r.squared
r2yh.svdr <- r2yh.h - r2yh.hhat

# ALL
r2yh <- c(sva=r2yh.sva, svdx=r2yh.svdx, svdr=r2yh.svdr)
```
# IV. Sum of squares of fx estimation for 2 models, and 3 different surrogate variable estimations

```r
if(trace) print("estimate fx estimation error")
```

```r
# SVA
mod.sva <- cbind(mod, sva.lin$sv)
err.fx.sva <- mean(fx.err(mod=mod.sva, x=sem.sim$xs, fx.real=sem.sim$fx, ord, trace))
```

```r
# SVD.X
mod.svdx <- cbind(mod, sva.lin$svd.x)
err.fx.svdx <- mean(fx.err(mod=mod.svdx, x=sem.sim$xs, fx.real=sem.sim$fx, ord, trace))
```

```r
# SVD.R
mod.svdr <- cbind(mod, sva.lin$svd.r)
err.fx.svdr <- mean(fx.err(mod=mod.svdr, x=sem.sim$xs, fx.real=sem.sim$fx, ord, trace))
```

```r
# vanilla
err.fx.van <- mean(fx.err(mod, x=sem.sim$xs, fx.real=sem.sim$fx, ord, trace))
```

```r
# ALL
err.fx <- c(sva=err.fx.sva, svdx=err.fx.svdx, svdr=err.fx.svdr, van=err.fx.van)
```

# V. Null p-values for 2 models, and 3 different surrogate variable estimations

```r
if(trace) print("calculate null p-values")
```

```r
# SVA
mod.sva0 <- cbind(mod, sva.lin0$sv)
pval.sva0 <- pval(mod=mod.sva0, x=sem.sim$xs0, ord, trace)
```

```r
# SVDX
mod.svdx0 <- cbind(mod, sva.lin0$svd.x)
pval.svdx0 <- pval(mod=mod.svdx0, x=sem.sim$xs0, ord, trace)
```

```r
# SVDR
mod.svdr0 <- cbind(mod, sva.lin0$svd.r)
pval.svdr0 <- pval(mod=mod.svdr0, x=sem.sim$xs0, ord, trace)
```

```r
# VANILLA
pval.van0 <- pval(mod=mod, x=sem.sim$xs0, ord, trace)
```

```r
# ALL
pval0 <- cbind(sva=pval.sva0, svdx=pval.svdx0, svdr=pval.svdr0, van=pval.van0)
```

# VI. Number of C-nodes extracted by significance algorithm as a % of the initial number detected by num.sv

```r
ext.rate <- sva.lin$n.sv.r / sva.lin$n.sv.r.ini
```

```r
if(trace) print("exits measures.sim")
return(list(cancor.cs.sva=cancor.cs.sva, cancor.hs=cancor.hs, r2yh=r2yh, err.fx=err.fx, pval0=pval0, ext.rate=ext.rate))
```

---

# A function which fits the four regression models - SVA, SVDR, SVDX and VANILLA - and calculates its various components

```r
# This function fits the four regression models to the simulated sem sem.sim:
# 1) x_j = f_{x_j}(y) + sum_{i=1}^{K} \hat{h}^{SVA}_k + \epsilon_j(SVA)
# 2) x_j = f_{x_j}(y) + sum_{i=1}^{K} \hat{h}^{SVDR}_k + \epsilon_j(SVDR)
# 3) x_j = f_{x_j}(y) + sum_{i=1}^{K} \hat{h}^{SVDX}_k + \epsilon_j(SVDX)
# 4) x_j = f_{x_j}(y) + \epsilon_j(VANILLA)

# Also calculates various model components such as model design matrix, hat matrix,
# expected/estimated x_j, beta_hat, f_{x_j}(y) and f_{x_j}(y) for y1,...,y_n and residuals.

# @param sem.sim: simulation of sem which includes a simulation under the null (f_{x_j}(y)=0 for all j) and under alternative.
# @param sva.lin: estimated SVA model, including surrogate variables, with alternative hypothesis simulation
# @param ord: order of the polynomial standard basis model f(y)
# @return a list with:
# x_hat - an array (reals^{n x 5 x J}) with expected x_j for SVA, SVDR, SVDX, VANILLA and real data (without noise)
functions

B.1 Functions

fg.hat - an array (reals^{n x 3 x J}) with estimated f_{j}(y) for y_{1},...,y_{n} for SVA, VAHILLA and real data

fx.hat - an array (reals^{n x 4 x J}) with estimated f_{x_{j}}(y) for y_{1},...,y_{n} for SVA, SVDR, SVDX and real data

betah.hat - an array (reals^{n x 4 x J}) with estimated sum_{k=1}^{J} k \beta_{kj} for SVA, SVDR, SVDX and real data

res - an array (reals^{n x 5 x J}) with estimated residuals for SVA, SVDR, SVDX, VAHILLA and real data

get.Models <- function(sem.sim, sva.lin, ord){
  reps <- dim(sem.sim$xs)[1]
  J <- dim(sem.sim$xs)[2]

  # Vanilla model
  mod.van <- model.matrix(~poly(sem.sim$y, ord, raw=TRUE))
  H.van <- mod.van %*% solve(t(mod.van) %*% mod.van) %*% t(mod.van)
  x.van.hat <- H.van %*% sem.sim$xs
  beta.van.hat <- solve(t(mod.van) %*% mod.van) %*% t(mod.van) %*% sem.sim$xs
  res.van <- sem.sim$xs - x.van.hat

  # SVA model
  mod.sva <- cbind(mod.van, sva.lin$sv)
  H.sva <- mod.sva %*% solve(t(mod.sva) %*% mod.sva %*% t(mod.sva))
  x.sva.hat <- H.sva %*% sem.sim$xs
  beta.sva.hat <- solve(t(mod.sva) %*% mod.sva) %*% t(mod.sva) %*% sem.sim$xs
  fx.sva.hat <- mod.sva[,1:(ord+1)] %*% beta.sva.hat
  fg.sva.hat <- fx.sva.hat + (sva.lin$sv - sva.lin$svd.r.red) %*% beta.sva.sv.hat
  res.sva <- sem.sim$xs - x.sva.hat
  beta.hs.sva.hat <- sva.lin$sv %*% beta.sva.sv.hat

  # SVD.R model
  mod.svdr <- cbind(mod.van, sva.lin$svd.r)
  H.svdr <- mod.svdr %*% solve(t(mod.svdr) %*% mod.svdr %*% t(mod.svdr))
  x.svdr.hat <- H.svdr %*% sem.sim$xs
  beta.svdr.hat <- solve(t(mod.svdr) %*% mod.svdr) %*% t(mod.svdr) %*% sem.sim$xs
  fx.svdr.hat <- mod.svdr[,1:(ord+1)] %*% beta.svdr.hat
  res.svdr <- sem.sim$xs - x.svdr.hat
  beta.hs.svdr.hat <- sva.lin$svd.r %*% beta.svdr.sv.hat

  # SVD.X model
  mod.svdx <- cbind(mod.van, sva.lin$svd.x)
  H.svdx <- mod.svdx %*% solve(t(mod.svdx) %*% mod.svdx %*% t(mod.svdx))
  x.svdx.hat <- H.svdx %*% sem.sim$xs
  beta.svdx.hat <- solve(t(mod.svdx) %*% mod.svdx) %*% t(mod.svdx) %*% sem.sim$xs
  fx.svdx.hat <- mod.svdx[,1:(ord+1)] %*% beta.svdx.hat
  res.svdx <- sem.sim$xs - x.svdx.hat
  beta.hs.svdx.hat <- sva.lin$svd.x %*% beta.svdx.sv.hat

  # Real
  beta.hs.real.hat <- sem.sim%$hs %*% sem.sim%$beta
  x.hat <- abind(real=sem.sim$exp.xs, van=x.van.hat, sva=x.sva.hat, svdr=x.svdr.hat, svdx=x.svdx.hat, along=3)
  dimnames(x.hat) <- list(reps=1:reps, J=1:J, method=c("real","van","sva","svdr","svdx"))
  fg.hat <- abind(real=sem.sim$fg, van=x.van.hat, sva=fg.sva.hat, along=3)
  dimnames(fg.hat) <- list(reps=1:reps, J=1:J, method=c("real","van","sva"))
  fx.hat <- abind(real=sem.sim$fx, sva=fx.sva.hat, svdr=fx.svdr.hat, svdx=fx.svdx.hat, along=3)
  dimnames(fx.hat) <- list(reps=1:reps, J=1:J, method=c("real","sva","svdr","svdx"))
  betah.hat <- abind(real=beta.hs.real.hat, sva=beta.hs.sva.hat, svdr=beta.hs.svdr.hat, svdx=beta.hs.svdx.hat, along=3)
dimnames(betah.hat) ← list(reps=1:reps, J=1:J, method=c("real","sva","svdr","svdx"))
res ← abind(real=sem.sim$x, van=res.van, sva=res.sva, svdr=res.svdr, svdx=res.svdx, along=3)
dimnames(res) ← list(reps=1:reps, J=1:J, method=c("real","van","sva","svdr","svdx"))
return(list(x.hat=x.hat, fg.hat=fg.hat, fx.hat=fx.hat, betah.hat=betah.hat, res=res))
}
# #######################################################################
# ' Simulates a SEM and then n observations of that SEM once. Calculates surrogate
# variables and performance measures.
#'
# ' This function first simulates a SEM (i.e. simulates the parameters which
# define an additive gene expression SEM) and then performs n observations of that SEM once. It then fits 4 models and
# calculates performance measures.
#'
# ' @param K: The number of "c" and "h" nodes (number of eigengenes)
# ' @param J: The number of "x" nodes (number of genes)
# ' @param deg.max: The maximum degree of the polynomial functions
# ' @param n: the number of observations of the sem to simulate.
# ' @param sigma: the standard deviation of the N_{x_{j}} noise variables.
# ' @param y.c.signal: the ratio of the standard deviation of the y variable to
# the standard deviation of the c variables which is 1.
# ' @param ord: order of the polynomial standard basis model \( f(y) \)
# ' @param sem: an additive gene expression SEM specified in a list.
# ' @param cheat: if TRUE the real beta is passed to sva.lin so that it can
# calculate the enriched matrix based on beta
# ' @param sparse: if TRUE simulates a SEM with controlled sparsity
# ' @param p0_fh: The proportion of fh(y)=0 functions
# ' @param p0_fx: The proportion of fx(y)=0 functions
# ' @param p_isox: The proportion of xs d-separated from y
# ' @param p0_beta: The minimum proportion of betas = 0
# ' @param trace: whether or not to print out a trace of steps carried out
#'
# ' @return msrs.sem: the performance measures for the simulation as produced by
# function measures.sem
#
#'
do. one. rep ← function(K, J, deg.max, n, sigma, y.c.signal, ord, sem=NULL, cheat=FALSE, sparse=FALSE, p0_fh=0, p0_fx=0, p_isox=0, p0_beta=0, trace=FALSE){
  if(trace) print(" enters do. one. rep ")

  # Simulate the SEM from class of SEMs
  if(trace) print(" simulate sem structure ")
  if(is.null(sem)){
    if(!sparse){
      sem.lin ← sim.sem(K, J, deg.max, trace)
    } else{
      sem.lin ← sim.sem.sparse(K, J, deg.max, p0_fh, p0_fx, p_isox, p0_beta, beta =NULL, trace=trace)
    }
  } else{
    sem.lin ← sem
  }

  # Simulate the repetitions of the hole SEM which is now fixed
  if(trace) print("simulate sem realization")
  sem.sim ← sim.lin(sem.lin, n, sigma, y.c.signal, trace)

  # estimate surrogate variables
  mod ← model.matrix(~poly(sem.sim$y, ord, raw=TRUE))

  if(cheat){
    n.sv.r ← min(n,K) # normally should be n.sv.r ← num.sv(dat, mod)
n.sv.x ← min(min(n,J),2*K)  # normally should be n.sv.x ← fa.
beta ← sem.lin$beta  # normally should be NULL

} else{
  n.sv.r ← max(2,num.sv(dat=t(sem.sim$xs), mod))
n.sv.x ← n.sv.r #min(min(n,J),2*K)  #max(3, fa.parallel(t(sem.sim$xs), fa="pc",
              n.iter=20)$ncomp)
  beta ← NULL
}

if(trace) print("estimate sv's")
sva.lin ← sva.mod(dat=t(sem.sim$xs), y=sem.sim$y, mod, n.sv.r=n.sv.r, n.sv.x=
                n.sv.x, beta=beta, trace)
sva.lin0 ← sva.mod(dat=t(sem.sim$x0), y=sem.sim$y, mod, n.sv.r=n.sv.r, n.sv.x=
                n.sv.x, beta=beta, trace)

# Calculate some performance measures for our simulated sem
if(trace) print("calculate performance measures")
msrs.sem ← measures.sem(sem.sim, sva.lin, sva.lin0, mod, ord, trace)
if(trace) print("exits do.one.rep")
return(msrs.sem)

#"' Simulates a SEM and then n observations of that SEM reps times. Calculates
#"' surrogate variables and performance measures
#"' for each rep and then collects and orders them.
#"' This function calls do.one.rep which first simulates a SEM (i.e. simulates the
#"' parameters which define an additive gene expression SEM)
#"' then performs n observations of that SEM. It then fits 4 models and calculates
#"' performance measures. Repeats M times and then collects
#"' and orders all the performance measures.
#"' @param reps : Number of times it will simulate an additive gene expression SEM
#"' with n observations
#"' @param K: The number of "c" and "h" nodes (number of eigengenes)
#"' @param J: The number of "x" nodes (number of genes)
#"' @param deg.max: The maximum degree of the polynomial functions
#"' @param n: the number of observations of the sem to simulate.
#"' @param ord: the standard deviation of the N_{(x_j)} noise variables.
#"' @param y.c.signal: the ratio of the standard deviation of the y variable to the
#"' standard deviation of the c variables which is 1.
#"' @param p0_fh: The proportion of fh(y)=0 functions
#"' @param p0_z: The proportion of z_{(x_j)} variables.
#"' @param sparse: if TRUE the real beta is passed to sva.lin so that it can
#"' calculate the enriched matrix based on beta
#"' @param sparse: if TRUE simulates a SEM with controlled sparsity
#"' @param p0_fh: The proportion of fh(y)=0 functions
#"' @param p0_z: The proportion of z_{(x_j)} variables.
#"' @param trace: whether or not to print out a trace of steps carried out
#"' @return a list with following matrices (matrices in reals^{no.reps x no.
#"' methods}):
# cancor.cs.sva - % overlap between estimated c_i-node span and real c_i-node span
# cancor.hs - % overlap between estimated h_k-node span and real h_k-node span
# r2yh - R^2 between real y and h_k nodes minus R^2 between real y and estimated h_k nodes.
# err.fz - Mean absolute error in estimation of f_(x_j)(y) at y_1,..., y_n
ks.nested - nested KS statistic for hypothesis H0: nested null p-values uniformly distributed

count.no.svs - number of repetitions where no surrogate variable was able to be calculated

est.rate - Number of C-nodes extracted by sva algorithm as a % of the initial number detected by num.sv

# do.n.reps <- function(reps, K, J, deg.max, n, sigma, y.c.signal, ord, sem=NULL, cheat=FALSE, sparse=FALSE, p0_fh=0, p0_fx=0, p_isox=0, p0_beta=0, trace=FALSE){
if(trace) print("enters do.n.reps")

count.no.svs <- 0

count <- 0
mrs.sem.list <- list()
for(i in 1:reps){
  print("***********************************")
  print(paste(" sim : ",i, sep=""))
  print("***********************************")
  res <- try(do.one.rep(K,J, deg.max, n, sigma, y.c.signal, ord, sem, cheat, sparse, p0_fh, p0_fx, p_isox, p0_beta, trace))
  if(inherits(res, "try-error")){
    count.no.svs <- count.no.svs + 1
  } else{
    count <- count + 1
    mrs.sem.list[[count]] <- res
  }
}
cancor.cs.sva <- sapply(mrs.sem.list, function(el) el$cancor.cs.sva)
cancor.hs <- t(sapply(mrs.sem.list, function(el) el$cancor.hs, simplify="array"))
r2yh <- t(sapply(mrs.sem.list, function(el) el$r2yh, simplify="array"))
err.fx <- t(sapply(mrs.sem.list, function(el) el$err.fx, simplify="array"))
pval0 <- sapply(mrs.sem.list, function(el) el$pval0, simplify="array")
dimnames(pval0) <- list(J=1:J, method=c("sva","svdx","svdr","van"), rep=1:count)
ks.outer <- apply(pval0, c("method","rep"), function(el) ks.test(el, "punif", alternative="two.sided")$p.value)
ks.nested <- apply(pval0, "method", function(el){
  test <- ks.test(el, "punif", alternative="two.sided")
  return(c(stat=test$statistic, pval=test$p.value))
})
ext.rate <- sapply(mrs.sem.list, function(el) el$ext.rate)
if(trace) print("exits do.n.reps")
return(list(cancor.cs.sva=cancor.cs.sva, cancor.hs=cancor.hs, r2yh=r2yh, err.fx=err.fx, ks.nested=ks.nested, count.no.svs=count.no.svs, ext.rate=ext.rate))
}

# Performs sensibility analysis by simulating reps repetitions of n observations for each parameter set.
# This function calls do.n.reps once for each row of data.frame params which contains one column for each simulation parameter.

# @param reps: Number of times it will simulate an additive gene expression SEM with n observations
# @param params: A data.frame containing one row per parameter set and one column per parameter (K, J, deg.max, n, sigma_{c.j}, sigma_{c.l}, ord, p_{0k}, p_{0j}, p_{(0)dep} and p_{(0)beta})
# @param sem: an additive gene expression SEM specified in a list.
# @param cheat: if TRUE the real beta is passed to sva.lin so that it can calculate the enriched matrix based on beta
# @param sparse: if TRUE simulates a SEM with controlled sparsity
# @param trace: whether or not to print out a trace of steps carried out
B.1 Functions

# ' @return a list with following matrices (arrays in reals^{ no . reps x no . methods x no . parameter sets }) :
# cancor . cs . sva - % overlap between estimated c_l-node span and real c_l-node span
# cancor . hs - % overlap between estimated h_k-node span and real h_k-node span
# r2yh - R2 between real y and h_k nodes minus R2 between real y and estimated h_k nodes.
# err . fx - Mean absolute error in estimation of f_ (x_j)(y) at y_1,..., y_n
# ks . nested - nested KS statistic for hypothesis H0: nested null p-values uniformly distributed
# count . no . svs - number of repetitions where no surrogate variable was able to be calculated
# ext . rate - Number of C-nodes extracted by sva algorithm as a % of the initial number detected by num . sv

sens . model ← function (reps , params , sem = NULL , cheat = FALSE , sparse = TRUE , trace = FALSE ) {
  res ← list ()
  for (i in 1: dim (params)[1]) {
    print ("******************************")
    print (" parameter set : ", i, sep = "")
    print (" parameters :")
    print (params[i,])
    res[i] ← do . n . reps (reps , K = params$K[i], J = params$J[i], deg . max = params$deg . max[i], n = params$n[i], sigma = params$sigma[i], y . c . signal = params$y . c . signal[i], ord = params$ord[i], sem = sem , cheat = cheat, sparse = sparse, p0_fh = params$p0_fh[i], p0_fx = params$p0_fx[i], p_iso = params$p_iso[i], p0_beta = params$p0_beta[i], trace = trace)
    print ("******************************")
  }
  count . no . svs ← sapply (res , function (el) el$count . no . svs)
  ext . rate ← sapply (res , function (el) el$ext . rate)
  cancor . cs . sva ← sapply (res , function (el) el$cancor . cs . sva).
  cancor . hs ← sapply (res , function (el) el$cancor . hs , simplify = "array")
  r2yh ← sapply (res , function (el) el$r2yh , simplify = "array")
  err . fx ← sapply (res , function (el) el$err . fx , simplify = "array")
  ks . nested ← sapply (res , function (el) el$ks . nested[1 ,], simplify = "array")
  return (list (cancor . cs . sva = cancor . cs . sva, cancor . hs = cancor . hs, r2yh = r2yh, err . fx = err . fx, ks . nested = ks . nested, count . no . svs = count . no . svs, ext . rate = ext . rate))
}

#' Graphs n observations a simulated gene expression SEM and one or more of the different estimation methods for one j in \{1 ,... , J\}

graph . Models ← function (sem . sim , models , select = c(" real"," sva"," svdr"," svdx"," van"), j) {
  x . hat ← models$x . hat[j ,]
  fg . hat ← models$fg . hat[j ,]
  fx . hat ← models$fx . hat[j ,]
  betah . hat ← models$betah . hat[j ,]
  res ← models$res[j ,]
  indx ← order (sem . sim$y)

## Produces 5 graphs comparing different components of each estimated model:

## @param sem . sim: simulation of sem which includes a simulation under the null (f_ (x_j)(y)=0 for all j) and under alternative.
## @param models: list of arrays containing the model components for all methods as produced by function get . Models
## @param j: model x_j = f_ (x_j)(y) + sum_{k=1}^K beta_(k)h_k + epsilon_j with j in \{1 ,... , J\} that one wants to visualize
## @param select: models which one wants to compare. Must be one or more of real, sva, svdr, svdx and van
tab <- data.frame(select = c("real","sva","svdr","svdx","van"), cols = c("black","green","red","brown","blue"), labs=c('real','SVA','SVD.R','SVD.X','vanilla'), stringsAsFactors=F)

#compare SVA, SVD.r, SVD.x and Vanilla model
# 1) E[x|y]
select.aux <- intersect(select, colnames(x.hat))
key <- match(select.aux, tab$select)
rng <- range(cbind(x.hat[,sel],sem.sim$xs[,j]))
plot(sem.sim$y[indx], sem.sim$xs[indx,j], main=paste("E[x_,j,|y]",sep="")
  , xlab="y", ylab="f(y)", ylim=rng)
for(k in 1:length(sel)) lines(sem.sim$y[indx], x.hat[indx, sel[k]], col=tab$cols[key[k]])
legend('bottomright', legend=tab$labs[key], col=tab$cols[key], lwd=rep(3.5, length(sel)))

# 2) f_y
select.aux <- intersect(select, colnames(fg.hat))
key <- match(select.aux, tab$select)
 sel <- match(select.aux, colnames(fg.hat))
 rng <- range(cbind(fg.hat[,sel],sem.sim$xs[,j]))
 plot(sem.sim$y[indx], sem.sim$xs[indx,j], main=paste("f(x)=f_x_,j, (y) + beta*f_h(y)",sep="")
    , xlab="y", ylab="f(y)", ylim=rng)
for(k in 1:length(sel)) lines(sem.sim$y[indx], fg.hat[indx, sel[k]], col=tab$cols[key[k]])
legend('bottomright', legend=tab$labs[key], col=tab$cols[key], lwd=rep(3.5, length(sel)))

# 3) f_x
select.aux <- intersect(select, colnames(fx.hat))
 key <- match(select.aux, tab$select)
 sel <- match(select.aux, colnames(fx.hat))
 rng <- range(cbind(sem.sim$xs[,j], fx.hat[,sel]))
 plot(sem.sim$y[indx], sem.sim$xs[indx,j], main=paste("f_x_,j, (y) ",sep="")
    , xlab="y", ylab="f(y)", ylim=rng)
for(k in sel) lines(sem.sim$y[indx], fx.hat[indx,sel[k]], col=tab$cols[key[k]])
legend('bottomright', legend=tab$labs[key], col=tab$cols[key], lwd=rep(3.5, length(sel)))

# 4) beta*h
select.aux <- setdiff(intersect(select, colnames(betah.hat)),"real")
 key <- match(select.aux, tab$select)
 sel <- match(select.aux, colnames(betah.hat))
 rng <- range(betah.hat[,sel])
 plot(1,1, col="white", main=paste("beta*h_,j, vs beta_h_hat*h_hat_,j",sep="")
    , xlab=paste("beta*h_,j,sep="")
    , ylab=paste("beta_h_hat*h_hat_,j,sep="")
    , ylim=rng, xlin-range(betah.hat[,"real"]))
for(k in 1:length(sel)){
  lines(betah.hat[,"real"], betah.hat[,sel[k]], col=tab$cols[key[k]], type="p"
    , pch=k)
  if(! all(betah.hat[,"real"]==0)) abline(lm(betah.hat[,sel[k]]~betah.hat[,"real"]), col=tab$cols[key[k]])
}
abline(a=0,b=1, col="red")
legend('bottomright', legend=tab$labs[key], col=tab$cols[key], pch=1:length(sel))

# 5) residuals
boxplot(res, main=paste("residuals E[x_,j,|y]"))
 select.aux <- intersect(select, colnames(res))
 key <- match(select.aux, tab$select)
 sel <- match(select.aux, colnames(res))
 rng <- range(res[,sel])
 plot(1,1, col="white", main=paste("Tukey Anscombe residuals E[x_,j,|y]",
    , sep="")
    , xlab="beta*h", ylab="", ylim=rng, xlin-range(betah.hat[,"real"])
    )
B.1 Functions

```r
indx.x <- order(betah.hat[, "real"])
for (k in 1:length(sel)){
  lines(betah.hat[, "real"], res[, sel[k]], type="p", col=tab$cols[key[k]], pch=k)
  if(! all(betah.hat[, "real"]==0)){
    sm <- smooth.spline(betah.hat[indx.x, "real"], res[indx.x, sel[k]])
    lines(sm, col=tab$cols[key[k]])
  }
}
legend('bottomright', legend=tab$labss[key], col=tab$cols[key], pch=1:length(sel), lwd=rep(3.5, length(sel)))
```

# #######################################################################
# ' Produces boxplots of a performance measure for one set of parameters for different methods
#'
# ' It is used to produce plots for all performance measures. Each boxplot corresponds to one method.
#'
# ' @param mat: A matrix with n rows, one per observation, and p columns one per level of the sensibility parameter
#='
# ' @param title: desired title for plot
#'
#'
#'
# boxPlot <- function(mat, title){
#  data <- melt(mat)
#  colnames(data) <- c("rep","method","value")
#  p <- ggplot(data)
#  p <- p + geom_boxplot(data)
#  p <- p + ggtitle(title)
#  p
#}

# #######################################################################
# ' Produces boxplots of a performance measure for one method accross a set of parameters
#'
# ' It is used to produce the sensibility c_l-node estimation plot which only applies to SVA method. Each boxplot corresponds to one parameter
#'
# ' @param mat: A matrix with n rows, one per observation, and p columns one per level of the sensibility parameter
# '
# ' @param labs: labels for x-axis (changing parameter values)
#'
# boxPlot2 <- function(mat, labs, xaxis, yaxis, title){
#  data <- melt(mat)
#  colnames(data) <- c("rep","param","value")
#  data$param <- as.factor(data$param)
#  p <- ggplot(data)
#  p <- p + geom_boxplot(aes(x=param, y=value))
#  p <- p + ggtitle(title)
#  p <- p + scale_x_discrete(breaks=levels(data$param), labels=labs)
#  p <- p + xlab(xaxis) + ylab(yaxis)
#  p
#}
```

# #######################################################################
# ' Produces boxplots of a performance measure for various methods accross a set of parameters
#'
# ' It is used to produce the sensibility h_k-node, h_k-y dependence and f_{z_j} estimation plots which apply to various methods
#'
# ' Each group of boxplots corresponds to one parameter, each color to a different estimation method (SVA, SVDR, SVDX, VEN).
#'
# ' @param mat: An array in reals^{n x num.methods x num.params}
#'
# ' @param labs: labels for x-axis (changing parameter values)
#'
# ' @param yaxis: y-axis title

```
B.2 Script

```r
# SCRIPT

# Simulate the SEM from class of SEMS
seed ← 4
K ← 4
J ← 4
deg.max ← 2
p0_fh ← 0.5
p0_fx ← 0.5
p_isox ← 0
p0_beta ← 0
set.seed(seed)

beta ← matrix(rnorm(k*m),k,m)

mat.indx ← matrix(c(2,3,3,4,4,4,1,2,4,2,4),6,2)
```
beta[mat.indx] ← 0
sem.lin ← sim.sem.sparse(K, J, deg.max, p0_fh, p0_fx, p_isox, p0_beta, beta)

# Simulate the repetitions of the hole SEM which is now fixed
n ← 100
sigma ← 1
y.c.signal ← 1
set.seed(seed)
sem.sim ← sim.lin(sem=sem.lin, n, sigma, y.c.signal)

# estimate surrogate variables
ord ← deg.max
n.sv.r ← max(2, num.sv(dat=t(sem.sim$xs), mod))
n.sv.x ← min(n,m),2*k) #max(3, fa.parallel(t(sem.sim$xs), fa="pc", n.iter =20)$+comp)
beta ← NULL
mod ← model.matrix(~poly(sem.sim$y, ord, raw=TRUE))
sva.lin ← sva.mod(dat=t(sem.sim$xs), y=sem.sim$y, mod, n.sv.r=n.sv.r, n.sv.x=n.sv.x, beta=beta)
sva.lin0 ← sva.mod(dat=t(sem.sim$xs0), y=sem.sim$y, mod, n.sv.r=n.sv.r, n.sv.x=n.sv.x, beta=beta)

# Visualize different models
models ← get.Models(sem.sim, sva.lin, ord)
graph.Models(sem.sim, models, select=c("real","sva","van"), 1)
graph.Models(sem.sim, models, select=c("real","sva","van"), 2)
graph.Models(sem.sim, models, select=c("real","sva","van"), 3)
graph.Models(sem.sim, models, select=c("real","sva","van"), 4)

# Calculate some performance measures for our simulated sem
(mrs.sem ← measures.sem(sem.sim, sva.lin = sva.lin, sva.lin0 = sva.lin0, mod, ord))

# Perform some repetitions
reps ← 1000
K ← 4
J ← 4
deg.max ← 1
p0_fh ← 0.5
p0_fx ← 0.5
p_isox ← 0.25
p0_beta ← 0.5
n ← 100
sigma ← 1
y.c.signal ← 1
ord ← deg.max

set.seed(1)
pm ← proc.time()
mrs.sem.reps ← do.n.reps(reps, K, J, deg.max, n, sigma, y.c.signal, ord, sem=sem.lin, cheat=FALSE, sparse=TRUE, p0_fh=p0_fh, p0_fx=p0_fx, p_isox=p_isox, p0_beta=p0_beta, trace=TRUE)
proc.time() - pm

# 217 seconds for reps = 10,000
# 414 seconds for reps = 20,000

mrs.sem.reps$count.no.svs
hist(mrs.sem.reps$ext.rate, main="c-node extraction rate")
hist(mrs.sem.reps$cancor.cs.sva, main="c-node overlap SVA")
boxPlot(mrs.sem.reps$cancor.hs, "b-node overlap")
boxPlot(mrs.sem.reps$r2yh, "real r2 - est. r2 (y ~ hs)")
boxPlot(mrs.sem.reps$err.fx, "mean absolute error")
mrs.sem.reps$ks.nested

# HIGH DIMENSIONAL SEM

# High Dimensional SEM
# Simulate the SEM from class of SEMS

debug ← 4
K ← 10
J ← 1000
deg.max ← 2
p0_fh ← 0.5
p0_fx ← 0.5
p_isox ← 0.25
p0_beta ← 0.5
set.seed(debug)
sem_lin ← sim_sem.sparse(K, J, deg.max, p0_fh, p0_fx, p_isox, p0_beta)

# Simulate the repetitions of the hole SEM which is now fixed
n ← 100
sigma ← 1
y.c. signal ← 1
set.seed(debug)
sem_sim ← sim_lin(sem = sem_lin, n, sigma, y.c. signal)

# estimate surrogate variables
ord ← deg.max
mod ← model_matrix(~poly(sem_sim$y, ord, raw = TRUE))
n.sv.r ← max(2, num_sv(dat = t(sem_sim$xs), mod))
n.sv.x ← n.sv.r  # max(3, fa.parallel(t(sem_sim$xs), fa = "pc", n.iter = 20)$ncomp)

beta ← NULL

sva_lin ← sva.mod(dat = t(sem_sim$xs), y = sem_sim$y, mod, n.sv.r = n.sv.r, n.sv.x = n.sv.x, beta = beta)
sva_lin0 ← sva.mod(dat = t(sem_sim$xs0), y = sem_sim$y, mod, n.sv.r = n.sv.r, n.sv.x = n.sv.x, beta = beta)

# Visualize different models
models ← get.Models(sem_sim, sva_lin, ord)
graph.Models(sem_sim, models, select = c("real", "sva", "van"), 1)
graph.Models(sem_sim, models, select = c("real", "sva", "van"), 2544)
graph.Models(sem_sim, models, select = c("real", "sva", "van"), 6664)
graph.Models(sem_sim, models, select = c("real", "sva", "van"), 8532)

# Calculate some performance measures for our simulated sem
(msrs_sem ← measures_sem(sem_sim, sva_lin = sva_lin, sva_lin0 = sva_lin0, mod, ord))
hist(msrs_sem$pval0[, "sva"])

ks.test(msrs_sem$pval0[, "sva"], "punif", alternative = "two.sided")
ks.test(msrs_sem$pval0[, "sva"], "punif", alternative = "two.sided")
ks.test(msrs_sem$pval0[, "sva"], "punif", alternative = "two.sided")

# Perform 100 repetitions
reps ← 100
K ← 10
J ← 1000
deg.max ← 1
p0_fh ← 0.5
p0_fx ← 0.5
p0_beta ← 0.5
n ← 100
sigma ← 1
y.c. signal ← 1
ord ← deg.max

set.seed(2)
rm ← proc.time()
msrs_sem_reps ← do.n.reps(reps, K, J, deg.max, n, sigma, y.c. signal, ord, sem = NULL, cheat = FALSE, sparse = TRUE, p0_fh = p0_fh, p0_fx = p0_fx, p_isox = p_isox, p0_beta =
p0_beta, trace=TRUE)
proc.time() - pm

# 40 minutes for reps = 20, K=100, J=10,000
# 6.7 minutes for reps = 20, K=50, J=2,500

mrs.sem.reps$count.no.svs
hist(mrs.sem.reps$ext.rate, main="c-node extraction rate")
hist(mrs.sem.reps$cancor.cs.sva, main="c-node overlap, SVA")
boxPlot(mrs.sem.reps$cancor.hs, "h-node overlap")
boxPlot(mrs.sem.reps$r2yh, "real r2 - est. r2 (y ~ hs")
boxPlot(mrs.sem.reps$err.fx, "mean absolute error")
mrs.sem.reps$ks.nested

# #######################################################################
# SENSIBILITY
# #######################################################################

# size of model (just J)

reps ← 100
K ← 10
J ← round(seq(100, 1000, length.out=10))
deg.max ← 1
n ← 100
sigma ← 1
y.c.signal ← 1
ord ← deg.max
p0_fh ← 0.5
p0_fx ← 0.5
p_isox ← 0.25
p0_beta ← 0.5
params ← expand.grid(K=K, J=J, deg.max=deg.max, n=n, sigma=sigma, y.c.signal=y.c.signal, ord=ord, p0_fh=p0_fh, p0_fx=p0_fx, p_isox=p_isox, p0_beta=p0_beta)

set.seed(2)
pm ← proc.time()
sens.sem.km ← sens.model(reps, params, cheat=FALSE, sparse=TRUE, trace=FALSE)
proc.time() - pm
# 47 mins

labs ← as.character(J)
xaxis ← "J"
yaxis ← "n"
boxPlot2(sens.sem.km$ext.rate, labs, xaxis, "x", title="c-node extraction rate")
boxPlot2(sens.sem.km$cancor.cs.sva, labs, xaxis, "x", title="c-node overlap")
boxPlot3(sens.sem.km$cancor.hs, labs, xaxis, "method", title="h-node overlap")
boxPlot3(sens.sem.km$r2yh, labs, xaxis, "method", title="real r2 - est. r2 (y ~ hs")
boxPlot3(sens.sem.km$err.fx, labs, xaxis, "method", title="mean absolute error")
ks.plot(sens.sem.km$ks.nested, labs, xaxis, yaxis)

# size of model (K and J)

reps ← 100
K ← round(seq(4,300, length.out=10))
J ← round(K/0.1)
deg.max ← 1
n ← 100
sigma ← 1
y.c.signal ← 1
ord ← deg.max
```R
params ← expand.grid(K=K, deg.max=deg.max, n=n, sigma=sigma, y.c.signal=y.c.signal,
                      ord=ord, p0_fh=p0_fh, p0_fx=p0_fx, p_isox=p_isox, p0_beta=p0_beta)
params$J ← round(params$K/0.1)

set.seed(2)

sens.sem.km ← sens.model(reps, params, cheat=FALSE, sparse=TRUE, trace=FALSE)
proc.time() - pm # 178 mins

labs ← paste(K,J, sep="-")
xaxis ← "K-J"
yaxis ← ""

boxPlot2(sens.sem.km$ext.rate, labs, xaxis, "," , title="c-node extraction rate")
boxPlot2(sens.sem.km$cancor.cs.sva, labs, xaxis, "," , title="c-node overlap")
boxPlot3(sens.sem.km$cancor.hs, labs, xaxis, "method", title="h-node overlap")
boxPlot3(sens.sem.km$r2yh, labs, xaxis, "method", title="real r2 - est. r2 (y ~ hs)")
boxPlot3(sens.sem.km$err.fx, labs, xaxis, "method", title="mean absolute error")
ks.plot(sens.sem.km$ks.nested, labs, xaxis, yaxis)

# #############################################################
# sparsity of model p0_h
# #############################################################

reps ← 100
K ← 10
J ← 1000
deg.max ← 1
n ← 100
sigma ← 1
y.c.signal ← 1
ord ← deg.max
p0_fh ← seq(0,1,length.out=10)
p0_fx ← 0.5
p_isox ← 0.25
p0_beta ← 0.5
params ← expand.grid(K=K,J=J, deg.max=deg.max, n=n, sigma=sigma, y.c.signal=y.c.signal,
                      ord=ord, p0_fh=p0_fh, p0_fx=p0_fx, p_isox=p_isox, p0_beta=p0_beta)

set.seed(2)

pm ← proc.time()

sens.sem.sp_fh ← sens.model(reps, params, cheat=FALSE, sparse=TRUE, trace=TRUE)
proc.time() - pm # 78 mins

labs ← as.character(round(p0_fh,2))
xaxis ← "% fh(y)=0"
yaxis ← ""

boxPlot2(sens.sem.sp_fh$ext.rate, labs, xaxis, "," , title="c-node extraction rate")
boxPlot2(sens.sem.sp_fh$cancor.cs.sva, labs, xaxis, "," , title="c-node overlap")
boxPlot3(sens.sem.sp_fh$cancor.hs, labs, xaxis, "method", title="h-node overlap")
boxPlot3(sens.sem.sp_fh$r2yh, labs, xaxis, "method", title="real r2 - est. r2 (y ~ hs)")
boxPlot3(sens.sem.sp_fh$err.fx, labs, xaxis, "method", title="mean absolute error")
ks.plot(sens.sem.sp_fh$ks.nested, labs, xaxis, yaxis)

# #############################################################
# sparsity of model p0_w
# #############################################################

reps ← 100
K ← 10
```
J ← 1000
deg.max ← 1
n ← 100
sigma ← 1
y.c.signal ← 1
ord ← deg.max
p0_fh ← 0.5
p0_fx ← seq(0.25,1,length.out=10)
p_isox ← 0.25
p0_beta ← 0.5
params ← expand.grid(K=K,J=J, deg.max=deg.max, n=n, sigma=sigma, y.c.signal=y.c.
signal, ord=ord, p0_fh=p0_fh, p0_fx=p0_fx, p_isox=p_isox, p0_beta=p0_beta)

set.seed(2)

pm ← proc.time()
sens.sem.sp_fx ← sens.model(reps, params, cheat=FALSE, sparse=TRUE, trace=TRUE)
proc.time() - pm

# 100 reps, 78 mins

labs ← as.character(round(p0_fx,2))
xaxis ← "% fx(y)=0"
yaxis ← ""
boxPlot2(sens.sem.sp_fx$ext.rate, labs, xaxis, "", title="c-node extraction rate")
boxPlot2(sens.sem.sp_fx$cancor.cs.sva, labs, xaxis, "", title="c-node overlap")
boxPlot3(sens.sem.sp_fx$cancor.hs, labs, xaxis, "method", title="h-node overlap")
boxPlot3(sens.sem.sp_fx$r2yh, labs, xaxis, "method", title="real r2 - est. r2 (y ~ hs)")
boxPlot3(sens.sem.sp_fx$err.fx, labs, xaxis, "method", title="mean absolute error")
ks.plot(sens.sem.sp_fx$ks.nested, labs, xaxis, yaxis)

# sparsity of model p0_beta

reps ← 100
K ← 10
J ← 1000
deg.max ← 1
n ← 100
sigma ← 1
y.c.signal ← 1
ord ← deg.max
p0_fh ← 0.5
p0_fx ← 0.5
p_isox ← 0.25
p0_beta ← seq(0,0.99,length.out=10)
params ← expand.grid(K=K,J=J, deg.max=deg.max, n=n, sigma=sigma, y.c.signal=y.c.
signal, ord=ord, p0_fh=p0_fh, p0_fx=p0_fx, p_isox=p_isox, p0_beta=p0_beta)

set.seed(2)

pm ← proc.time()
sens.sem.sp_beta ← sens.model(reps, params, cheat=FALSE, sparse=TRUE, trace=FALSE)
proc.time() - pm

# 75 mins

labs ← as.character(round(p0_beta,2))
xaxis ← "% beta=0"
yaxis ← ""
boxPlot2(mat=sens.sem.sp_beta$ext.rate, labs, xaxis, yaxis="", title="c-node extraction rate")
boxPlot2(sens.sem.sp_beta$cancor.cs.sva, labs, xaxis, "", title="c-node overlap")
boxPlot3(sens.sem.sp_beta$cancor.hs, labs, xaxis, "method", title="h-node overlap")
boxPlot3(sens.sem.sp_beta$r2yh, labs, xaxis, "method", title="real r2 - est. r2 (y ~ hs)")
boxPlot3(sens.sem.sp_beta$err.fx, labs, xaxis, "method", title="mean absolute error")
ks.plot(sens.sem.sp_beta$ks.nested, labs, xaxis, yaxis)
```r
# sparsity of model p_\text{isox}
#
reps ← 100
code
K ← 10
code
J ← 1000
code
deg.max ← 1
code
n ← 100
code
sigma ← 1
code
y.c.signal ← 1
code
ord ← deg.max
code
p0_fh ← 0.5
code
p0_fx ← seq(0, 0.99, length.out=10)
plot(p0_fx, p_isox)
code
p0_beta ← 0.5
code
params ← expand.grid(K=K, J=J, deg.max=deg.max, n=n, sigma=sigma, y.c.signal=y.c.signal, ord=ord, p0_fh=p0_fh, p0_fx=p0_fx, p0_beta=p0_beta)
code
params$p_isox ← params$p0_fx^2
set.seed(2)
code
pm ← proc.time()
code
sens.sem.sp_isox ← sens.model(reps, params, cheat=FALSE, sparse=TRUE, trace=FALSE)
code
proc.time() - pm
# 103 mins

code
labs ← paste(round(p0_fx, 2), round(p_isox, 2), sep="-")
code
xaxis ← "\% fx(y)=0 - \% p_isox=0"
code
yaxis ← ""
code
boxPlot2(sens.sem.sp_isox$ext.rate, labs, xaxis, ",title="c-node extraction rate "")
code
boxPlot2(sens.sem.isox$cancor.cs.sva, labs, xaxis, "", title="c-node overlap")
code
boxPlot3(sens.sem.isox$cancor.hs, labs, xaxis, "method", title="h-node overlap")
code
boxPlot3(sens.sem.isox$r2yh, labs, xaxis, "method", title="real r2 - est. r2 (y \sim hs)")
code
ks.plot(sens.sem.isox$ks.nested, labs, xaxis, yaxis)

code
# y.c.signal
#
reps ← 100
code
K ← 10
code
J ← 1000
code
deg.max ← 1
code
n ← 100
code
sigma ← 1
code
y.c.signal ← c(1/10, 1/5, 1/3, 1/2, 1, 2)
code
ord ← deg.max
code
p0_fh ← 0.5
code
p0_fx ← 0.5
code
p_isox ← 0.25
code
p0_beta ← 0.5
code
params ← expand.grid(K=K, J=J, deg.max=deg.max, n=n, sigma=sigma, y.c.signal=y.c.signal, ord=ord, p0_fh=p0_fh, p0_fx=p0_fx, p_isox=p_isox, p0_beta=p0_beta)
code
set.seed(2)
code
pm ← proc.time()
code
sens.sem.y.c.signal ← sens.model(reps, params, cheat=FALSE, sparse=TRUE, trace=TRUE)
code
proc.time() - pm
# 69 mins

code
labs ← as.character(round(y.c.signal, 2))
```

xaxis ← "y.c.signal"
ynaxis ←"
boxPlot2(sens.sem.y.c.signal$ext.rate, labs, xaxis, ",title="c-node extraction rate")
boxPlot2(sens.sem.y.c.signal$cancor.cs.sva, labs, xaxis, ",title="c-node overlap")
boxPlot3(sens.sem.y.c.signal$cancor.hs, labs, xaxis, "method", title="h-node overlap")
boxPlot3(sens.sem.y.c.signal$r2yh, labs, xaxis, "method", title="real r2 - est. r2 (y ~ hs))
boxPlot3(sens.sem.y.c.signal$err.fx, labs, xaxis, "method", title="mean absolute error")
ks.plot(sens.sem.y.c.signal$ks.nested, labs, xaxis, yaxis)

# nx noise

reps ← 100
K ← 10
J ← 1000
deg.max ← 1
n ← 100
sigma ← c(1/10, 1/5, 1/3, 1/2, 1, 2, 5, 10)
y.c.signal ← 1
ord ← deg.max
p0_fh ← 0.5
p0_fx ← 0.5
p.isox ← 0.25
p0_beta ← 0.5
params ← expand.grid(K=K, J=J, deg.max=deg.max, n=n, sigma=sigma, y.c.signal=y.c.signal, signal, ord=ord, p0_fh=p0_fh, p0_fx=p0_fx, p.isox=p.isox, p0_beta=p0_beta)

set.seed(2)
pm ← proc.time()
sens.sem.sigma ← sens.model(reps, params, cheat=FALSE, sparse=TRUE, trace=TRUE)
proc.time() - pm
# 91 mins

labs ← as.character(round(sigma, 2))
xaxis ← "x-node noise"
yaxis ← ""
boxPlot2(sens.sem.sigma$ext.rate, labs, xaxis, ",title="c-node extraction rate")
boxPlot2(sens.sem.sigma$cancor.cs.sva, labs, xaxis, ",title="c-node overlap")
boxPlot3(sens.sem.sigma$cancor.hs, labs, xaxis, "method", title="h-node overlap")
boxPlot3(sens.sem.sigma$r2yh, labs, xaxis, "method", title="real r2 - est. r2 (y ~ hs))
boxPlot3(sens.sem.sigma$err.fx, labs, xaxis, "method", title="mean absolute error")
ks.plot(sens.sem.sigma$ks.nested, labs, xaxis, yaxis)

# degree of functions fx and fh

reps ← 100
K ← 10
J ← 1000
deg.max ← seq(5)
n ← 100
sigma ← 1
y.c.signal ← 1
ord ← deg.max
p0_fh ← 0.5
p0_fx ← 0.5
p.isox ← 0.25
p0_beta ← 0.5
params ← expand.grid(K=K, J=J, deg.max=deg.max, n=n, sigma=sigma, y.c.signal=y.c.signal, signal, p0_fh=p0_fh, p0_fx=p0_fx, p.isox=p.isox, p0_beta=p0_beta)
params$ord ← params$deg.max
```r
set.seed(2)

pm ← proc.time()
sens.sem.deg ← sens.model(reps, params, cheat=FALSE, sparse=TRUE, trace=TRUE)
proc.time() - pm
# 50 mins

labs ← as.character(round(deg.max,2))
xaxis ← "fx and fh order"
yaxis ← ""
boxPlot2(sens.sem.deg$ext.rate, labs, xaxis, "", title="c-node extraction rate")
boxPlot2(sens.sem.deg$cancor.cs.sva, labs, xaxis, "", title="c-node overlap")
boxPlot3(sens.sem.deg$cancor.hs, labs, xaxis, "method", title="h-node overlap")
boxPlot3(sens.sem.deg$r2yh, labs, xaxis, "method", title="real r2 - est. r2 (y ~ hs)")
boxPlot3(sens.sem.deg$err.fh, labs, xaxis, "method", title="mean absolute error")
ks.plot(sens.sem.deg$ks.nested, labs, xaxis, yaxis)

# ############################
# number of points n
# ############################

reps ← 100
K ← 10
J ← 1000
deg.max ← 1
n ← round(seq(25,1000, length.out=10,0)
sigma ← 1
ord ← deg.max
p0_fh ← 0.5
p0_fx ← 0.5
p_isox ← 0.25
p0_beta ← 0.5
params ← expand.grid(K=K,J=J, deg.max=deg.max, n=n, sigma=sigma, y.c.signal=y.c.
signal, ord=ord, p0_fh=p0_fh, p0_fx=p0_fx, p_isox=p_isox, p0_beta=p0_beta)

set.seed(2)

pm ← proc.time()
sens.sem.n ← sens.model(reps, params, cheat=FALSE, sparse=TRUE, trace=TRUE)
proc.time() - pm
# 31 hours!!!

labs ← as.character(round(n,2))
xaxis ← "sample size"
yaxis ← ""
boxPlot2(sens.sem.n$ext.rate, labs, xaxis, "", title="c-node extraction rate")
boxPlot2(sens.sem.n$cancor.cs.sva, labs, xaxis, "", title="c-node overlap")
boxPlot3(sens.sem.n$cancor.hs, labs, xaxis, "method", title="h-node overlap")
boxPlot3(sens.sem.n$r2yh, labs, xaxis, "method", title="real r2 - est. r2 (y ~ hs)")
boxPlot3(sens.sem.n$err.fh, labs, xaxis, "method", title="mean absolute error")
ks.plot(sens.sem.n$ks.nested, labs, xaxis, yaxis)
```