Department of Mathematics and Computer Science
Statistics Group

Structure Learning in Multiple Time Series

Master Thesis

Martin de Quincey

Supervisors:	Assessment Committee Members:
dr. Rui Castro	dr. Rui Castro

version 0.1

Eindhoven, December 2021

Abstract

This is the abstract.

Vivamus vehicula leo a justo. Quisque nec augue. Morbi mauris wisi, aliquet vitae, dignissim eget, sollicitudin molestie, ligula. In dictum enim sit amet risus. Curabitur vitae velit eu diam rhoncus hendrerit. Vivamus ut elit. Praesent mattis ipsum quis turpis. Curabitur rhoncus neque eu dui. Etiam vitae magna. Nam ullamcorper. Praesent interdum bibendum magna. Quisque auctor aliquam dolor. Morbi eu lorem et est porttitor fermentum. Nunc egestas arcu at tortor varius viverra. Fusce eu nulla ut nulla interdum consectetuer. Vestibulum gravida. Morbi mattis libero sed est.

Preface

This is the preface.

Vivamus vehicula leo a justo. Quisque nec augue. Morbi mauris wisi, aliquet vitae, dignissim eget, sollicitudin molestie, ligula. In dictum enim sit amet risus. Curabitur vitae velit eu diam rhoncus hendrerit. Vivamus ut elit. Praesent mattis ipsum quis turpis. Curabitur rhoncus neque eu dui. Etiam vitae magna. Nam ullamcorper. Praesent interdum bibendum magna. Quisque auctor aliquam dolor. Morbi eu lorem et est porttitor fermentum. Nunc egestas arcu at tortor varius viverra. Fusce eu nulla ut nulla interdum consectetuer. Vestibulum gravida. Morbi mattis libero sed est.

Contents

List of Figures

List of Tables

List of Definitions

$A \otimes B$ - The *Kronecker product* of two matrices $A \in \mathbb{R}^{m \times n}$ and $B \in \mathbb{R}^{p \times q}$. Performing this operation yields an $mp \times nq$ matrix.

I_n - The *identity matrix* with dimensions $n \times n$.

n - Total number of samples per variable.

p - Total number of nodes in a graph; Total number of variables in a dataset; Number of time series in our dataset.

P_{DS} - A *doubly stochastic matrix.* A matrix $P_{DS} \in \mathbb{R}^{p \times p}$ is doubly stochastic if and only if both all its rows as its columns sum to one. More specifically,

$$\sum_{i=1}^{p} p_{ij} = \sum_{j=1}^{p} p_{ij} = 1 \qquad \forall i, j = 1, \cdots, p.$$

P_{perm} - A *permutation matrix.* A matrix $P_{perm} \in \{0, 1\}^{p \times p}$ is a permutation matrix if and only if both all of its rows and columns contain exactly one non-zero entry. This only non-zero entry must be equal to one.

T - Number of time steps in a time series.

$\mathrm{Tr}(A)$ - The *Trace* of a square matrix $A \in \mathbb{R}^{p \times p}$ is defined to be the sum of its diagonal entries. More specifically,

$$\mathrm{Tr}(A) = \sum_{i=1}^{p} A_{ii}.$$

$vec(A)$ - The *vectorization operation* applied on a matrix $A \in \mathbb{R}^{m \times n}$. This operation transforms the matrix A by stacking the n columns of $A_{\cdot,i}, i = 1, \cdots, n$ on top of each other. More specifically,

$$vec(A) = (a_{11}, a_{21}, \cdots, a_{m1}, a_{12}, \cdots, a_{m2}, \cdots, a_{1n}, \cdots, a_{mn})^T.$$

$X_{t,i}$ - The value of time series i at time step t.

$X_{\cdot,i}$ - The ith *time series*, often of dimension T, the number of time steps. When there is no ambiguity possible, we might use the notation $X_{\cdot,i} \equiv X_i$.

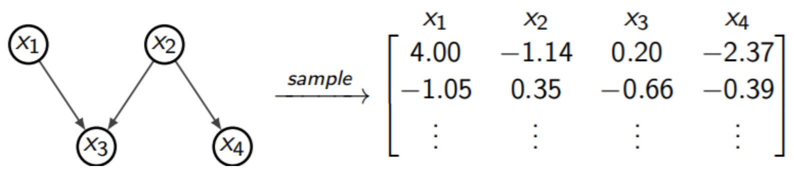
$X_{t,\cdot}$ - The p dimensional vector $\{X_{t,1}, \ldots, X_{t,p}\}$, containing the data values of our p time series. When there is no ambiguity possible, we might use the notation $X_{t,\cdot} \equiv X_t$.

$\mathbf{X}$ - The *data matrix*, often a subset of $\mathbb{R}^{T \times p}$, consisting of p time series of length T.

Chapter 1

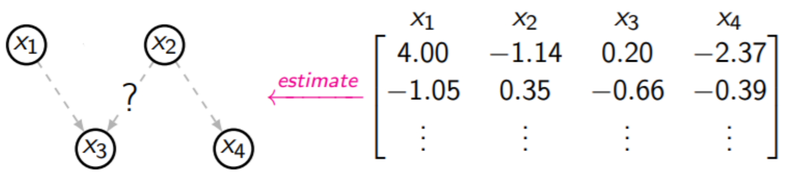
Introduction

This thesis explores the interesting domain of *structure learning*. In many scenarios, we have a graphical model, and we take *samples* generated by this graphical model. This process of sampling is depicted in Figure 1.1. In structure learning, the focus is on the opposite direction. Given a set of samples, the goal is to infer the structure of the graphical model that generated these samples. In other words, were are interested in learning the structure of the graphical model that generated our samples. This process of structure learning is depicted in Figure 1.2.



x_1
x_2
x_3
x_4
sample
$$\begin{array}{cccc} x_1 & x_2 & x_3 & x_4 \\ \hline 4.00 & -1.14 & 0.20 & -2.37 \\ -1.05 & 0.35 & -0.66 & -0.39 \\ \vdots & \vdots & \vdots & \vdots \end{array}$$

Figure 1.1: Visualization of *sampling* data from a graphical model [10].



x_1
x_2
?
x_3
x_4
estimate
$$\begin{array}{cccc} x_1 & x_2 & x_3 & x_4 \\ \hline 4.00 & -1.14 & 0.20 & -2.37 \\ -1.05 & 0.35 & -0.66 & -0.39 \\ \vdots & \vdots & \vdots & \vdots \end{array}$$

Figure 1.2: Visualization of *estimating* the structure of a graphical model from data [10].

Applications. Structure learning is an important concept that with many applications, predominantly concerning complex systems. In a world where systems are becoming evermore complex, it is important to have a suitable graphical model that captures its behavior. Nowadays, systems are so complex that they defy human intuition.

An example of a system so complex that it defies our intuition are genetic regulatory systems. Through these systems, we can understand which genes are expressed where in the organism, and to which extent. Most of these genetic regulatory systems involve many interlocking positive and negative feedback loops, making these systems so complex that they defy human intuition. More information on such complex networks in gene regulatory systems can be found in [6].

Apart from complex biological systems, even man-made systems can become so complex that they will defy human intuition. Based on human knowledge alone, it is not guaranteed that a man-made complex system will perfectly adhere to this intended structure. Furthermore, these man-made systems have become so complex that it is impossible for one person to comprehend its complete behavior. Examples of such complex machines are abundant throughout the tech industry, such as the EUV lithography systems manufactured by ASML.

Therefore, we see that there is a need for *data driven* approaches to construct models for these systems. The approaches will be particularly effective in the era of big data, where data through measurements in genetic regulatory systems or through sensors in man-made machines is widely available.

Motivation. Having seen some interesting applications for structure learning, let us now consider the benefits of having learned the structure of such a complex system. Consider a large and complex machine with many interacting components. When the machine breaks, it might be so complex that finding the issue can take a long time and require disassembling the machine completely. By using sensors, it is possible to know that something has gone wrong in the machine. However, this can be uninformative. The fact that a sensor reports anomalous values does not imply that the root cause is anywhere near this sensor. This is where structure learning comes into play. Suppose that we have analyzed the sensors of the working machine for a long enough period of time. If we can learn the structure of these sensors, very important information can be retrieved from this structure. First of all, whenever the machine breaks, we can look at which sensors produce anomalous results. Indeed, the most logical approach is still to first inspect the components around these sensors. However, if the issue does not lie there, we can use our learned structure to propose other components that are likely to be the root cause of the problem. Therefore, structure learning can be very useful for *root cause analysis* in complex systems.

Another advantage of using structure learning for complex machines is that we can verify whether indeed in theory non-interacting components indeed have no interaction in practice. If we can physically decompose the machine into multiple sections, each with their own function, the learned structure should adhere to this physical structure as well. Should this not be the case, then we know that something went wrong in either design or manufacturing process. Hence, structure learning can be a useful tool for *system validation* as well.

Objective. The objective throughout this thesis is to develop data driven approaches that can learn the structure of complex systems. More precisely, we are interested in learning *predictive causal networks*. The learned network describes the directed causal relations in between the nodes. Note that with causal relations we merely mean predictive causality, in other words, whether the values corresponding to one node are helpful in predicting another node. A visualization of the objective is given in Figure 1.2.

Outline. This thesis is structured as follows. In Chapter 3, important definitions and notions are introduced that are deemed necessary to read through this thesis. In Chapter 4, we will discuss several existing methods for structure learning. These existing methods range from a variety of different techniques, and hence all have their niche in the current state of the art literature. In Chapter 5, we will introduce novel methods that we have researched during this thesis. Our methods will be benchmarked against the aforementioned methods in Chapter.

Chapter 2

Problem Setting

Data notation

Consider the setting where we observe $p \in \mathbb{N}$ continuous variables over a duration of time $T \in \mathbb{N}$. We denote these variables by our *data matrix*

$$\mathbf{X} \in \mathbb{R}^{p \times T}.$$

Here, T can be seen as our time horizon. At each time step $t = 1, \ldots, T$, we observe our p variables, yielding real-valued measurements $X_{t,1}, X_{t,2}, \ldots, X_{t,p}$. Let $X_{t,\cdot} \in \mathbb{R}^p$ denote the p dimensional vector of measurements of our p variables at a specific time t. Furthermore, let $X_{\cdot,i} \in \mathbb{R}^T$ denote the T measurements of variable $i = 1, \ldots, p$, ordered from $t = 1$ until $t = T$. We also denote by $X_{\cdot,i}$ the ith time series in our data matrix $\mathbf{X}$, and $\{X_{\cdot 1}, \ldots, X_{\cdot,p}\}$ the set of our p time series. When there is no ambiguity possible, the notations $X_t \equiv X_{t,\cdot}$ and $X_i \equiv X_{\cdot,i}$ might be utilized to ease notation.

We assume that this data matrix $\mathbf{X}$ has been generated by some joint distribution $p(\mathbf{X})$. However, this joint distribution is unknown, and all that is available to determine this joint distribution is our data matrix $\mathbf{X}$. The goal is partly to recover this joint distribution $p(\mathbf{X})$, but we are predominantly interested in the graphical model that corresponds to this distribution $p(\mathbf{X})$. The next paragraph will explain what this graphical model will look like.

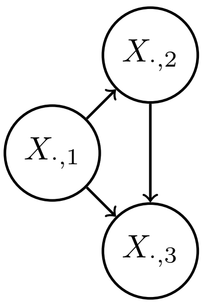
Graph representation

Given our data matrix $\mathbf{X}$, let us consider the directed graph representation $G(\mathbf{X}) = (V, A)$ consisting of p vertices. So, $|V| = p$, and the set of arcs or directed edges is $A = \{(x, y) \mid x, y \in V\}$. Vertex i refers to , $X_{\cdot,i}$, the ith time series in our data matrix. Therefore, a directed edge (i, j) indicates a directed relationship from $X_{\cdot,i}$ to $X_{\cdot,j}$. In the context of this thesis, such a directed edge implies a *directed predictive power*. In other words, the variable or time series $X_{\cdot,i}$ is helpful in predicting $X_{\cdot j}$. When there is no edge (i, j), both variables do not seem helpful in predicting each other. We say that $X_{\cdot,i}$ and $X_{\cdot,j}$ are conditionally independent.

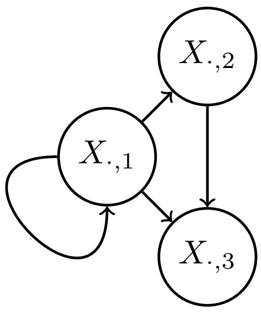
We can also denote our directed edge set A as an *adjacency matrix* $A \in \{0, 1\}^{p \times p}$, where an entry A_{ij} is equal to one if there is an directed edge (i, j), and zero otherwise. In mathematical notation,

$$A_{ij} = \begin{cases} 1 & \text{if } (i,j) \in A, \\ 0 & \text{otherwise.} \end{cases}$$

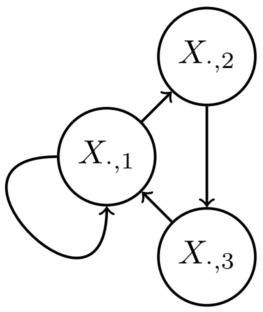
The goal of this thesis is to build such a directed graph $G(\mathbf{X}) = (V, A)$ based on our data matrix $\mathbf{X}$, under one extra constraint. The directed graph $G(\mathbf{X})$ must be *acyclic*. Nevertheless, our definition of acyclic deviates slightly from the literature. Given that our data matrix consists of p time series, it is not unlikely that a time series $X_{\cdot,i}$ will be helpful in predicting itself. Therefore, we do allow for *self-loops*, which are cycle of length 1. Therefore, when we say that $G(\mathbf{X})$ must



(a) The graph G_a.



(b) The graph G_b.



(c) The graph G_c

Figure 2.1: Three different graphs with $p = 3$ vertices. G_a consists of three edges $(1, 2)$, $(1, 3)$, and $(2, 3)$. As it does not contain any cycles, G_a is a DAG. G_b consists of the same edges, but now a self-loop $(1, 1)$ is added. G_b is not a DAG in the standard graph theory literature, as it contains a cycle of length 1. As mentioned before, we allow for cycles of length 1 and therefore, we consider G_b to be a DAG throughout this thesis. G_c consists of the same edges as G_b, but the edge $(1, 3)$ is flipped, resulting in the edge edge $(3, 1)$. Consequently, G_c contains a cycle of length 3 now, which we do not consider to be a DAG.

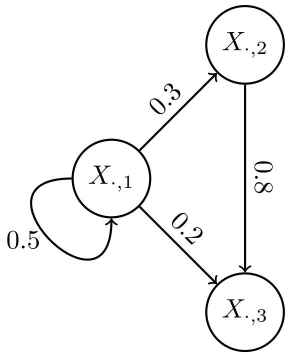


Figure 2.2: The weighted graph G_b from Figure 2.1a with weights W_b from Equation 2.1.

be acyclic, we require that it does not contain any cycle of length *greater* than 1. We call such a directed acyclic graph a DAG for short throughout this thesis. Three examples of graphs are given in Figure 2.1.

Furthermore, we also want to quantify the predictive power in the edge (i, j). Rather than simply stating whether there is a directed edge (i, j) or not, we want to assign a weight $w(i, j)$ as well to this edge. This weight quantifies the *strength* of the edge (i, j). We assign such a weight to all directed edges (i, j). Therefore, we will end up with a *weighted* adjacency matrix W, which is equal to

$$W_{ij} = \begin{cases} w(i,j) \cdot A_{ij} & \text{if } A_{ij} = 1, \\ 0 & \text{otherwise.} \end{cases}$$

We use the abbreviation WAM for the weighted adjacency matrix. For example, the adjacency matrix of the graph G_b depicted in Subfigure 2.1b is

$$A_b = \begin{pmatrix} 1 & 1 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}, \qquad W_b = \begin{pmatrix} 0.5 & 0.3 & 0.2 \\ 0 & 0 & 0.8 \\ 0 & 0 & 0 \end{pmatrix}. \tag{2.1}$$

A visualisation of G_b with the weighted adjacency matrix is given in Figure 2.2.

The goal of this thesis is not only to determine the underlying DAG, but also determine the weights of the directed edges. The problem can therefore informally be stated as follows:

Given $\mathbf{X}$, what is the "most suitable" WAM W such that $G(\mathbf{X})$ is a DAG (self-loops allowed)?

Note that the "most suitable" is put between quotes to indicate that this is certainly not a mathematically sound formulation. The elaboration on "most suitable" will come later in this chapter.

Difficulties

Having given a first explanation of the problem at hand, let us first discuss some difficulties that have already arisen. Learning the directed acyclic structure of $\mathbf{X}$ is quite challenging for multiple reasons.

First of all, when a variable $X_{\cdot,i}$ is useful in predicting $X_{\cdot,j}$, the converse could be true as well. Hence, $X_{\cdot,j}$ could also be useful in predicting $X_{\cdot,i}$. However, our graph $G(\mathbf{X})$ cannot contain cycles of length greater than 1, so it cannot contain both arcs. Therefore, we need not only know whether $X_{\cdot,i}$ is useful in predicting $X_{\cdot,j}$, but also whether $X_{\cdot,i}$ is *more* useful in predicting $X_{\cdot,i}$ than vice versa. In other words, apart from detecting whether there is a dependency between two variables, we also need to determine the most suitable *directionality* of the dependency.

Secondly, the total number of possible DAGs grows super-exponentially with p, the number of time variables. In fact, when we have only ten variables, the total number of possible DAGs already is a staggering $4.2 \cdot 10^{18}$ []. In other words, the search space of our problem is super-exponential. Hence, simply trying all possible directed acyclic graphs is not a tractable solution. Furthermore, we have not even discussed yet how to find a suitable WAM W given this adjacency matrix A. In fact, Chickering et al. has shown that the problem at hand is NP-hard [], meaning that it is highly unlikely for a method to exist that will solve this problem exactly in polynomial time. In the current literature, algorithms that can solve this problem exactly are only tractable for tens of time series.

Formal Problem Statement. Let us concretize the problem statement further. We have explained what we are looking for, but what remains unclear is what the "most suitable" W is for our data matrix $\mathbf{X}$. This had been left blank before is because there is no universally best choice of W. There are many different *cost functions* that can be used to determine whether a choice of W is "good". For now, we abstract from the cost function, and denote this by $C(W)$, to indicate that the only parameter we optimize over is the weighted adjacency matrix W. Now, we define the "most suitable" W as a matrix that minimizes the given cost function $C(W)$. Therefore, the overarching goal of this thesis is to find

$$W^* = \underset{W \in \imath \times \imath}{\arg\min}\, C(W) \text{ such that } G(W) \text{ is a DAG (self loops allowed).}$$

Lastly, we need to define what cost function $C(W)$ we will use. This will be done in the next paragraph, as there exist multiple suitable cost functions.

Cost functions $C(W)$ In order to determine whether a weighted adjacency matrix W is a good fit for our data matrix $\mathbf{X}$, we need some sort of cost function $C(W)$ that quantifies the goodness of fit. Therefore, let us consider some possible cost functions.

The first possibility of a valid cost function is the *negative log likelihood function* of the data $\mathbf{X}$. The likelihood function described the likelihood of the data $\mathbf{X}$ being generated by some statistical model. The joint distribution $\hat{p}(W; \mathbf{X})$ associated with this statistical model this can be described by a set of parameters, in our scenario a matrix W. We can optimize this joint distribution with respect to W. The larger $p(W; \mathbf{X})$, the more likely the data matrix $\mathbf{X}$ has been generated by the statistical model parametrized by W. Hence, we want to maximize $p(W; \mathbf{X})$ with respect to W.

Equivalently, we can minimize the negative log-likelihood, which is obtained by taking the natural logarithm of $p(W;\mathbf{X})$, and changing the sign:

$$\ell(W;\mathbf{X}) = -\ln\left(p\left(W;\mathbf{X}\right)\right).$$

A second possibility is also known as the mean squared error (MSE), which is the average of the sum of squared residual errors. In the literature, it is also known as the L_2 loss function. Three equivalent notations are

$$\begin{aligned} MSE(\hat{\mathbf{X}};\mathbf{X}) &= \frac{1}{T}\|\mathbf{X}-\hat{\mathbf{X}}\|_F^2 && (2.2)\\ &= \frac{1}{T}\sum_{t=1}^{T}\left\|X_{\cdot,t}-\hat{X}_{\cdot,t}\right\|_2^2 && (2.3)\\ &= \frac{1}{T}\sum_{t=1}^{T}\sum_{i=1}^{p}\left(X_{i,t}-\hat{X}_{i,t}\right)^2. && (2.4)\end{aligned}$$

where $\hat{\mathbf{X}}$ is our estimate of the data matrix $\mathbf{X}$. Analogously, $\hat{X}_{\cdot,t}$ is our estimate for $X_{\cdot,t}$, and $\hat{X}_{i,t}$ is our estimate for $X_{i,t}$.

Another possibility is the mean absolute error (MAE), which is the average of the sum of the residuals in absolute value. Another name for the mean absolute error is L_1 loss. The formula for the mean absolute error is

$$\begin{aligned} \mathrm{MAE}(\hat{\mathbf{X}};\mathbf{X}) &= \frac{1}{T}\sum_{t=1}^{T}\|X_{\cdot,t}-\hat{X}_{\cdot,t}\|_1 && (2.5)\\ &= \frac{1}{T}\sum_{t=1}^{T}\sum_{i=1}^{p}\left|X_{i,t}-\hat{X}_{i,t}\right|. && (2.6)\end{aligned}$$

Regularization An extra feature that we prefer our weighted adjacency matrix W to have is that it is *sparse*. This means that we want W to achieve a small loss and at the same time, we prefer for W to have few non-zero entries. This is often achieved by adding regularization. Adding an L_1 regularization penalty to the weighted adjacency matrix W. In mathematical notation, this means that we add an L_1 penalization with respect to W, $\rho(W)$, which is defined as

$$\rho(W) = \lambda\|W\|_1 := \lambda\left(\sum_{i=1}^{p}\sum_{j=1}^{p}|W_{ij}|\right),$$

where $\lambda \geq 0$ determines how strong the penalty will be for having non-zero entries.

Let us now define our objective more formally. When we say we are looking for the "most suitable" weighted adjacency matrix W, we mean that this W minimizes some cost function $C(W)$. This cost function can be any sensible combination of the loss functions and regularization techniques mentioned above. For now, we stick to the following cost function:

$$C(W,\lambda) = MSE(\hat{\mathbf{X}};\mathbf{X}) + \lambda\|W\|_1. \tag{2.7}$$

In Equation 2.7, We have omitted the dependency of our cost function on $\mathbf{X}$, as we can not optimize the cost function over a fixed data matrix $\mathbf{X}$. Note that we do not have $\mathbf{X}$ as a parameter of our cost function, as our estimator $\hat{\mathbf{X}}$ will be parametrized by the weighted adjacency matrix W.

Estimating the data matrix $\hat{\mathbf{X}}$ Since we have defined our cost functions in the previous paragraph, the new variable $\hat{\mathbf{X}}$ has appeared, which is the *estimator* for our data matrix $\mathbf{X}$. In this paragraph, we will explain which model we will use to estimate our data. For this, we use a vector autoregressive model of order 1, or VAR(1) for short. Our estimator is defined as

$$\hat{X}_{t,\cdot} = X_{t-1,\cdot}W, \qquad t = 2,...,T \tag{2.8}$$

where $X_{t-1,\cdot} \in |$ are the actual values of the previous timestep, and $W \in \mathbb{R}^{p\times p}$ is our weighted adjacency matrix. We see that we have no estimator for the first time step, as we have no previous time step. Therefore, the mean squared error is equal to

$$\begin{aligned} MSE(\hat{\mathbf{X}};\mathbf{X}) &= \frac{1}{T-1}\sum_{t=2}^{T} \|X_t - X_{t-1}W\|_2^2 \\ &= \frac{1}{T-1}\left\|\mathbf{X}_{[2:T]} - \mathbf{X}_{[1:T]}W\right\|_F^2 . \end{aligned}$$

From Equation 2.8, we see that our estimator $\hat{X}_{t,i}$ is in fact a linear combination of the p time series at time step $t-1$. Therefore, we can only expect our estimator to capture linear relations. Furthermore, we can only expect our estimator to capture from one time step ago. This restricts our estimator quite heavily. However, the simplicity yields a clear graphical model, namely a single directed acyclic graph. This is an unavoidable trade-off; the more complex we make our estimator $\hat{\mathbf{X}}$, the more complex it is to describe the relations between the time series.

To conclude the problem setting, consider the data matrix $\mathbf{X} \in \mathbb{R}^{T\times p}$. Given this data matrix, we are interested in capturing the directed relations between these p time series. The collection of these directed relations is captured in a weighted adjacency matrix W of a graph $G(W)$ on p nodes, one node for each time series. We constrain this collection of relations W by requiring $G(W)$ to be a directed acyclic graph with self-loops allowed. We consider W^* to be the weighted adjacency matrix that optimally captures the relationships, where we define optimally to be

$$W^* = \underset{W\in\mathbb{R}^{p\times p}}{\arg\min}\frac{1}{T-1}\|\mathbf{X} - \mathbf{X}W\|_F^2. \tag{2.9}$$

Therefore, we can conclude this chapter with the formal notation of our problem setting:

$$\text{Given } \mathbf{X}\text{, find } W^* = \underset{\substack{W\in\mathbb{R}^{p\times p},\\ G(W) \text{ a DAG.}}}{\arg\min} \quad \frac{1}{T-1}\sum_{t=2}^{T}\|X_t - X_{t-1}W\|_2^2 + \lambda\|W\|_1.$$

The next chapter will discuss some of the existing literature that focuses on solving variants of this problem.

Sidenote: We can extend the estimator to e.g. a VAR(k) model, but then we do not have a clear graphical model (arrows from all of the k previous time steps to the current time step). However, perhaps we can "condense" this into one matrix W by e.g. summing up the entries to get a quantifiable way of "predictive power". However, how do you enforce acyclicity on this?

Chapter 3

Preliminaries

3.1 Graph Theory

Graph.

Directed Acyclic Graph.

3.2 Models

3.2.1 Vector AutoRegression Model

A Vector AutoRegressive (VAR) model

$$X_t = \sum_{i=1}^{p} A_i X_{t-i} + e$$

3.2.2 Structural Equation Model

$$X = AX + e$$

3.2.3 Structural Vector AutoRegression Model

A Structural Vector AutoRegression Model (SVAR) model consists of both a SEM and a VAR part. We can write this as

$$X = AX + \sum_{p=1}^{p} A_i X_{t-1} + e.$$

Chapter 4

Previous Work

- *Exact methods:* These methods will find an exact solution to the problem. However, their running time will be exponential. These methods are only tractable for at most a dozen variables.
- *Approximate methods:* Given that exact methods are only applicable to a doze nodes, we need to accept a trade-off: methods that can are much faster than exponential, however, they can not guarantee the best solution. So, these methods could for example be applied to hundreds of nodes, but they cannot guarantee that the solution it finds is anywhere near the optimal solution.

4.1 Exact methods

4.1.1 GOBLNIP

4.1.2 Dynamic Programming

4.2 Approximate methods

4.2.1 NOTEARS

NOTEARS (*Non-combinatorial Optimization via Trace Exponential and Augmented lagRangian for Structure learning*) is a method developed by Zheng et al [9] from the Carnegie Mellon University. As in this thesis, the topic of interest is structure learning: inferring a directed acyclic graph from data. The authors focus on a linear Structural Equation Model of the form

$$X = WX + z.$$

The most difficult hurdle to overcome is the combinatorial constraint that the inferred structure must be a directed acyclic graph. In their paper, they present a novel strategy. Rather than solving the (partly) combinatorial optimization problem, they translate it to an equivalent continuous optimization problem.

$$\begin{aligned} &\min_{W\in\mathbb{R}^{d\times d}} && F(W) \\ &\text{subject to} && G(W)\in \mathsf{DAGs} \end{aligned} \quad\Longleftrightarrow\quad \begin{aligned} &\min_{W\in\mathbb{R}^{d\times d}} && F(W) \\ &\text{subject to} && h(W)=0, \end{aligned} \tag{4.1}$$

Now, the function $F(\cdot)$ that the authors try to optimize is the least-squares loss plus an ℓ_1 regularization on the weighted adjacency matrix to encourage sparsity of the DAG. In mathematical notation, this is equal to

$$F(W) = \frac{1}{2n}\|\mathbf{X}-\mathbf{X}W\|_F^2 + \lambda\|W\|_1.$$

The continuous function that enforces acyclicity is

$$h(W) = \operatorname{Tr}\left(e^{(W\circ W)}\right) - d.$$

Here $W \circ W$ represents the *Hadamard product* of two matrixes. In other words, it is the element-wise square of the matrix W. The authors note four reasons for choosing this function as $h(W)$. First of all,

Algorithm 1 NOTEARS

```
1: Input: Initial guess $(W_0, \alpha_0)$, progress rate $c \in (0,1)$, tolerance $\epsilon > 0$, threshold $\omega > 0$.
2: while F door $t = 0, 1, 2, \dots$:
3:     Solve primal $W_{t+1} \leftarrow \arg\min_W L^\rho(W, \alpha_t)$ with $\rho$ such that $h(W_{t+1}) < ch(W_t)$.
4:     Dual ascent $\alpha_{t+1} \leftarrow \alpha_t + \rho h(W_{t+1})$.
5:     If $h(W_{t+1}) < \epsilon$, set $\tilde{W_{\mathsf{ECP}}} = W_{t+1}$ and break.
6: end while
7: Return the thresholded matrix $\widehat{W} := \tilde{W_{\mathsf{ECP}}} \circ 1(|\tilde{W_{\mathsf{ECP}}}| > \omega)$.
```

$h(W) = 0$ if and only if W represents the weighted adjacency matrix of a DAG, and the value of $h(W)$ represents the "DAG"ness of W. Furthermore, the function as well as its gradient are smooth and can also be easily evaluated.

Optimization The authors use the *augmented Lagrangian* method to solve the right-hand side of Equation 4.1. To solve the augmented Lagrangian, an iterative two-step procedure is proposed. This pseudocode for this procedure is given in Algorithm 1. First, we optimize the augmented Lagrangian with respect to W (Line 3 of Algorithm 1). Secondly, we optimize the augmented Lagrangian with respect to the Lagrange multiplier α (Line 4 of Algorithm 1). The augmented Lagrangian with a penalty parameter $\rho > 0$ and a Lagrange multiplier α can be written as

$$L^\rho(W, \alpha) = F(W) + \frac{\rho}{2}|h(W)|^2 + \alpha h(W). \tag{4.2}$$

Equation 4.2 can be solved using a dual ascent method with respect to W and α. The dual function with the Lagrange multiplier α is given by

$$D(\alpha) = \min_{W \in \mathbb{R}^{d \times d}} L^\rho(W, \alpha). \tag{4.3}$$

To optimize Equation 4.2.1, the authors use the L-BFGS-B optimization method [11]. They used the implementation in the Python package `scipy` [1, 2, 8]. Now that we have found a local minimizer W_α^* of Equation , the next step is to find a local solution to the dual problem

$$\max_{\alpha \in \mathbb{R}} \quad D(\alpha). \tag{4.4}$$

Luckily, $D(\alpha)$ is linear with respect to alpha, and its derivative is simply $\nabla D(\alpha) = h(W_\alpha^*)$. The authors propose to simply do one step of dual gradient ascent with step size ρ. The dual gradient ascent is given by the formula

$$\alpha_{t+1} = \alpha_t + \rho \nabla D(\alpha). \tag{4.5}$$

The algorithm that the authors propose is an iterative procedure of two steps. First, optimize the augmented Lagrangian with respect to W using the L-BFGS-B optimization method. Secondly, optimize the augmented Lagrangian with respect to the Lagrange multiplier α using Equation 4.5. This iterative procedure is continued until a matrix W_{ECP} is found such that $h(W)$ is sufficiently small (Line 5 of Algorithm 1). This is the solution of the equality-constrained program (ECP), the right-hand side of 4.1.

The final part of the algorithm is *thresholding*. This means that all entries of W_{ECP} that are smaller than some threshold ω will be set to zero. Thresholding reduce the number of false positives []. Furthermore, as this algorithm is a numerical procedure, there will be inevitably some numerical precision errors near machine precision, so setting a threshold of for example $\omega = 10^{-8}$ would solve that issue as well. Experiments done by the authors also demonstrate that thresholding increases the accuracy in structure learning.

To conclude, the approach described in NOTEARS is a fundamentally novel approach that transforms the combinatorial constraint $G(W) \in \mathsf{DAGs}$ into a continuous constraint $h(W) = 0$. Then using a quite simple iterative two-step procedure using the augmented Lagrangian method, a local minimum could be found using existing solvers. Note that only a local minimum is guaranteed, and not a global minimum, as the search space is non-convex. Given that the problem at hand is NP-hard, this comes as no surprise. Should this method always find a global minimum, then an algorithm would exists that solves an NP-hard problem in polynomial time, which has never been found before.

However, the fact that only a local minimum is guaranteed does not seem to be a detrimental problem. The authors have compared the local optimum found using Algorithm 4.1 to the global optimum found using an exact program. In this scenario, the authors have used the GOBNILP program to find the global minimum. Quite often, the local optimum was close to the global optimum, which demonstrates that the non-convexity is not a large issue.

4.2.2 Relaxing to the Birkhoff Polytope

The paper *Learning Bayesian Networks through Birkhoff Polytope: A Relaxation Method* by Dallakyan et al. [5] describes a method that is similar to our research direction. The authors assume a linear structural equation model (SEM), which is of the form

$$X = BX + \varepsilon,$$

where $\varepsilon = (\varepsilon_1, \cdots, \varepsilon_p)^T$, $\varepsilon_j \sim \mathcal{N}\left(0, \omega_j^2\right), j = 1, \cdots, p$. We denote B to be the *weighted adjacency matrix*, and $\Omega = \text{diag}(w_1^2, \cdots, w_p^2)$ to be the *inverse covariance matrix*. This weighted adjacency matrix B defines the structure of the underlying directed acyclic graph $\mathcal{G}$. A non-zero entry b_{ij} represents an edge from variable X_j to variable X_i in $\mathcal{G}$.

The distribution of X, where X adheres to a linear SEM, is well known, namely

$$X \sim \mathcal{N}_p(\mathbf{0}, (I - B)^T \Omega^{-1} (I - B)).$$

The negative log-likelihood of can be written as

$$\begin{aligned}
l(B, \Omega\ |\mathbf{X}) &= \frac{1}{2}\text{Tr}\left(\mathbf{X}^T\mathbf{X}(I-B)\Omega^{-1}(I-B)\right) + \frac{n}{2}\log|\Omega| \\
&= \frac{1}{2}\text{Tr}\left(P\underbrace{\mathbf{X}^T\mathbf{X}}_{=nS}P^T\underbrace{(I-B_\pi)\Omega_\pi^{-T/2}}_{=L^T}\underbrace{\Omega_\pi^{-1/2}(I-B_\pi)}_{=L}\right) - n\log\underbrace{|\Omega_\pi|^{-1/2}}_{=|L|=\prod_{j=1}^p L_{jj}} \\
&= \frac{1}{2}\text{Tr}\left(PSP^TL^TL)\right) - \sum_{j=1}^p \log L_{jj} \\
&=: l(L, P|\mathbf{X})
\end{aligned}$$

An important note to make is that $l(L, P|\mathbf{X})$ is *permutation invariant*, in other words,

$$l(L, P|\mathbf{X}) = l(L, I|\mathbf{X})\ \forall \text{ permutation matrices } P.$$

To overcome this permutation invariance, the authors use the same approach as []. They regularize such that sparse DAGs are more preferred by adding a minimax concave penalty (MCP) $\rho(\cdot)$. This MCP is applied to all lower triangular entries of L. The final non-convex penalized score function that the authors consider is

$$Q(L, P) = \min_{L,P}\left(l(L, P|\mathbf{X}) + \sum_{1\leq j\leq i\leq p}\rho\left(|L_{ij}|; \lambda\right)\right).$$

In order to optimize , the authors propose an iterative approach; first optimize P, then optimize L. This is done until we have a maximum number of iterations k_{max} has been reached.

Algorithm 2 RRCF algorithm

```
1: input:
2: $\lambda, k_{max} \leftarrow$ Tuning Parameter, iteration
3: $L^{(0)}, P^{(0)} \leftarrow$ Initial matrices
4: while $k < k_{max}$:
5:     $\hat{P}^{(k)} = \arg\min_{P\in\mathcal{P}_p} Q_{RRCF}(L^{(k-1)}, P)$
6:     $\hat{L}^{(k)} = \arg\min_{L\in\mathcal{L}_p} Q_{RRCF}(L, P^{(k)})$
7:     $k = k + 1$
8: Output: $(\hat{L}, \hat{P})$
```

To optimize P (line 5 of Algorithm 2), the authors relax the constraints from the set of permutation matrices to the set of doubly stochastic matrices. If we only omit the terms that are constant in with respect to P, then we get that we need to minimize.

However,

To optimize L (line 6 of Algorithm 2), the authors propose a cyclic coordinatewise algorithm.

Algorithm 3 Cholesky Factor Estimation

```
1: input:
2: $k_{max}, \mathbf{X}, \lambda, \gamma, \epsilon$
3: $L^{(0)} \leftarrow$ Initial Cholesky factor
4: For i = 1,2,..., p
5:     $\beta^i = \arg\min_{\beta_i} Q_{RRCF,i}(\beta^i)$ via Algorithm ??
6: Construct $L \in \mathcal{L}_p$ by setting its non-zero values as $\beta^i$
7: Output: Lower diagonal matrix L
```

They have compared their method, which they have coined the Relaxed Regularized Cholesky Factor (RRCF) Framework, to three other existing methods; ARCS, CCDr, and NOTEARS. In the experiments, RRCF achieved results compatitble with these other methods.

However, the RRCF Framework suffers from some drawbacks. First of all, three tuning parameters, , need to be tuned. This can be computationally expensive, especially for high-dimensional datasets.

4.2.3 Using a genetic algorithm

The paper *Inferring large graphs using ℓ_1-penalized likelihood* by Champion et al. [4] proposes GADAG (*Genetic Algorithm for learning Directed Acyclic Graph*) to retrieve the underlying structure of a linear Gaussian Structural Equation Model, which is of the form

$$X = XG_0 + \varepsilon,$$

where $\varepsilon = (\varepsilon_1, \cdots, \varepsilon_p)^T$, $\varepsilon_j \sim \mathcal{N}\left(0, \sigma_j^2\right), j = 1, \cdots, p$. We denote G_0 to be the *weighted adjacency matrix.* This weighted adjacency matrix G_0 encodes the structure of the underlying directed acyclic graph $\mathcal{G}_0$. A non-zero entry g_{ij} represents an edge from variable X_i to variable X_j in $\mathcal{G}$.

To estimate this underlying graph G_0, the authors propose to use a penalized maximum likelihood method, which is of the form

$$\hat{G} = \arg\min\left(\ell(G) + \lambda \text{pen}\,(G)\right).$$

Omitting terms that are constant with respect to our variables, we get the following

$$\hat{G} = \arg\min\left(\frac{1}{n}\|X(I-G)\|_F^2 + \lambda\|G\|\right).$$

$$\left(\hat{P}, \hat{T}\right) = \arg\min\left(\frac{1}{n}\left\|X(I - PTP^T)\right\|_F^2 + \lambda\|T\|\right).$$

Minimizing P

4.2.4 PC Algorithm

4.2.5 Greedy Equivalent Search

Chapter 5

Method

We will assume that our data has been generated by a VAR(1) model, which is defined as

$$X_{t+1} = X_t W + E, \tag{5.1}$$

where $E \sim \mathcal{N}(\mathbf{0}, \Sigma)$. The matrix E is the random Gaussian noise, and the matrix W is as coefficient matrix. Here, t as index represents the time step which ranges from $t = 1$ until some time horizon $t = n$. Note that a more standard notation would be

$$X_{t+1} = W' X_t + E. \tag{5.2}$$

Nevertheless, both equations are equivalent for $W' = W^T$. However, the notation $X_t W$ is more easily interpretable from a graphical point of view.

For the sake of simplicity, we assume that our initial value $X_0 = \mathbf{0}$. Note, however, that we could have picked any finite p-dimensional value as initial value. Changing this would have only shifted our VAR(1) model, thereby only changing the mean of X_t to X_0.

The coefficient matrix W encodes the structure of our model. If an entry w_{ij} is equal to zero, that means that the variable $X_{i,t}$ has no influence in determining the value of the variable $X_{j,t}$ for any time t. However, if w_{ij} is in absolute value larger than zero, the variable $X_{i,t}$ is used in determining the value of the variable $X_{j,t}$. The larger w_{ij} in absolute value, the greater the influence of variable X_i on variable X_j.

We can visualize the coefficient matrix W as a graph $G(W)$. A non-zero entry w_{ij} indicates that there is an edge from X_i to X_j. This is the reason why we have opted for the notation of Equation 5.1 rather than the notation of Equation 5.2. Had we used the notation from Equation 5.2, then the corresponding weighted adjacency matrix would be W^T. From now on, let us denote $G(W)$ the weighted directed graph that is induced by the weighted adjacency matrix W.

Example, let us consider the VAR(1) model with

$$W =),(\qquad \Sigma =).($$

The corresponding graph $G(W)$ is depicted in Figure. Furthermore, a sample of 100 timesteps is depicted in Figure .

The largest difficulty to overcome throughout this thesis is that the weighted adjacency matrix W must be such that $G(W)$ is a directed acyclic graph.

5.1 Continuous Optimization Approaches

5.1.1 Decomposition of the Weighted Adjacency Matrix W

To constrain ourselves to matrices W such that the inferred DAG is acyclic, we recall this useful statement:

$$G(W) \text{ is a DAG} \iff \exists \text{ a permutation matrix P s.t. } P^T W P \text{ is lower triangular.}$$

An easy way to see this is that if $G(W)$ is a DAG, then there must be at least one one node x which does not have any incoming arcs. This node x will be the first entry in our permutation. Now, remove x and all its outgoing arcs from the graph, leaving us with a graph G'. We can now iterate this process, and find a new node y which does not have any incoming arcs in $G'(W)$. After this iterative procedure, we have found a permutation P such that P^TWP is lower triangular.

Although this might not be the most useful statement, a similar statement can be used in our advantage.

$$G(W) \text{ is a DAG } \iff \exists \text{ a lower triangular matrix } A, \text{ a permutation matrix P s.t. } P^TAP = W.$$

We can prove this statement by construction. Let $G(W)$ be the graph induced by the weighted adjacency matrix W. As $G(W)$ is a DAG, we can decompose W using a permutation matrix P^TAP such that P^TAP is lower triangular. Denote $A = P^TWP$, then we can revert this by using the same permutation matrix P. Trivially, we have that $PAP^T = PP^TWPP^T = IWI = W$.

The main reason for choosing this decomposition is that we do not have to worry about explicitly requiring $G(W)$ to be acyclic. We have changed the formulation of our problem in such a way this constraint is implicitly required. As long as we require A to be lower triangular and P to be a permutation matrix, we have automatically satisfied the acyclicity constraint.

The decomposition removes the explicit requirement that W must be acyclic, however, other hurdles arise that need to be overcome. As in any mathematical problem, circumventing one issue will create another issue. The question that arises then is which issue is the “least worrying”. A large amount of this thesis is dedicated to answering this question; what other issues do arise because of this composition?

One issue that still remains is the combinatorial search space of permutation matrices. Finding the most suitable permutation matrix P has now become the largest issue. For a total of p time series, there are $p!$ possible permutations and hence, $p!$ possible permutation matrices. This search space is exponential, meaning that an exhaustive search will not be feasible for a reasonably large number of time series. Nevertheless, the search space has been greatly reduced. Let $G(n)$ be the number of possible directed acyclic graphs with n nodes. The value of $G(n)$ is given by the recurrence

$$G(n) = \sum_{k=1}^{n}(-1)^{k+1}\binom{n}{k}2^{k(n-k)}G(n-k),$$

which was first given by Robinson in [7]. Even for just ten nodes, the number of possible dags is equal to $G(10) \approx 4.2\cdot 10^{18}$, a staggering number. However, there are “only” $10! \approx 3.6$ million different permutations. Hence, the combinatorial search space of permutation matrices is far smaller than the search space of directed acyclic graphs, albeit still exponential in size with respect to the number of variables.

Interestingly, during this thesis, two other papers were discovered that have used a similar decomposition. These two papers are mentioned and discussed in Subsection and Subsection . Given that two recent papers use a comparable strategy and consequently achieve results competitive with the current state of the art, there are mixed feelings. On one hand, it showcases the sensibility of the decomposition. On the other hand, it shows that the approach is not completely new.

5.1.2 Relax the permutation matrix constraint

Nevertheless, the combinatorial nature of the search space poses some problems. Continuous optimization procedures are unsuitable for these types of problems, and therefore combinatorial approaches would be required. Therefore, we propose a method to alleviate ourselves from the combinatorial search space. We relax the search space from the set of permutation matrices to the convex hull of permutation matrices. This convex hull is also known as the *Birkhoff polytope.*

Model definition

Therefore, we assume the model

$$X_{t+1} = P^TTPX_t + E, \tag{5.3}$$

where P is a permutation matrix and T is a lower triangular matrix. We can easily circumvent the acyclity constraint by estimating solely the lower triangular entries of T along with a permutation matrix P.

Note that a special property of a permutation matrix P is that its inverse is equal to its tranpose, i.e., $P^T_{perm} = P^{-1}_{perm}$. For reasons that will become more obvious later on, let us rewrite Equation to

$$X_{t+1} = P^{-1}TPX_t + E. \tag{5.4}$$

We propose to use the L_2-norm as a cost function:

$$C(A,P) = \frac{1}{T-1}\sum_{t=1}^{T}\left\|X_t - \hat{X}_{t\cdot}\right\|_2^2 \tag{5.5}$$

Here, X_t are the actual values of the time series at time t, and $\hat{X}_t$ are the estimates of the time series. Under our model assumptions, the cost function boils down to

$$C(A,P) = \frac{1}{T-1}\sum_{t=1}^{T}\left\|X_t - P^{-1}APX_{t-1\cdot}\right\|_2^2 \tag{5.6}$$

It is important to note some characteristics of our cost function here. First of all, the inverse of the matrix P makes Equation 5.7 a *non-convex* optimization problem.

Furthermore, the inverse of the matrix P can create serious non-singularity issues. When P is a permutation matrix, then P^{-1} is simply equal to P^T. However, the inverse of P is not defined always defined. For example, the matrix

$$P = \begin{pmatrix} 0.333 & 0.333 & 0.333 \\ 0.333 & 0.333 & 0.333 \\ 0.333 & 0.333 & 0.333 \end{pmatrix}$$

does not have an inverse, meaning that our cost function is not well defined for singular matrices.

Furthermore, even when the matrix is non-singular, we can still encounter issues. Let ε be an arbitrarily small number. Consider the matrix

$$P = \begin{pmatrix} 0.333-\varepsilon & 0.333+\varepsilon & 0.333 \\ 0.333+\varepsilon & 0.333-\varepsilon & 0.333 \\ 0.333 & 0.333 & 0.333 \end{pmatrix}.$$

This matrix is non-singular, as all the rows are linearly independent. However, from a numerical point of view, the matrix is numerically unstable. For $\varepsilon = 0.05$, the inverse of P is equal to

$$P^{-1} = \begin{pmatrix} 4.5e15 & 4.5e15 & -9.0e15 \\ 4.5e15 & 4.5e15 & -9.0e15 \\ -9.0e15 & -9.0e15 & 1.8e16 \end{pmatrix}.$$

This example illustrates the singularity problems that arise from our decomposition. Although we have relieved ourselves from the issues of the combinatorial search space of permutation matrices, we are now left with the singularity issues of P. We have traded our combinatorial constraint problem in for this singularity problem.

Population setting

Let us first analyze this model in the population setting. In this setting, we assume we have the entire population as data, meaning that we can take rather than having n samples, we have $n \to \infty$ samples.

Cost function In order to estimate these matrices A and P, we use an L_2-norm of the difference between the actual value $X_{t,val}$ and the predicted value $X_{t,pred}$. Assume the data has been generated by some (unknown) A^*, P^*, i.e.,

$$X_{t+1,val} = P^{*T}A^*P^*X_{t,val} + E,$$

where E again represents the added noise to our model. Now assume that we have our two estimated matrices A and P. The corresponding cost function in the population setting is equal to

$$C(P,A,P^*,A^*,E) = \mathbb{E}\left[\|X_{t,val} - X_{t,pred}\|_2^2\right] \tag{5.7}$$

Derivation of the cost function We can decompose this cost function to gain more insights into whether this model makes sense. The $2 - norm$ of a p dimensional variable $X \in ı$ is

$$\|X\|_2 = \sum_{i=1}^{p} \sqrt{|x_i|\|}.^2$$

Hence, we have that

$$\begin{aligned} C(P, A, P^*, A^*, E) &= \mathbb{E}\left[\|X_{t,val} - X_{t,pred}\|_2^2\right] \\ &= \mathbb{E}\left[\sum_{i=1}^{p} (X_{t,val,i} - X_{t,pred,i})^2\right] \\ &= \operatorname{Tr}\left(\mathbb{V}\left(X_{t,val} - X_{t,pred}\right)\right). \end{aligned} \tag{5.8}$$

So, we need to derive the covariance of $X_{t,val} - X_{t,pred}$. Recall that

$$X_{t+1,val} = \underbrace{P^{*-1}A^*P^*}_{=B^*} X_{t-1,val} + E, \qquad X_{t=1,pred} = \underbrace{P^{-1}AP}_{=B} X_{t,val}.$$

Hence,

$$\begin{aligned} \mathbb{V}\left(X_{t+1,val} - X_{t+1,pred}\right) &= \mathbb{V}\left(B^* X_{t,val} + E - BX_{t,val}\right) && \text{By Equation 5.4} \\ &= \mathbb{V}\left((B^* - B)X_{t,val} + E\right) && \text{Rearrange terms} \\ &= \mathbb{V}\left((B^* - B)X_{t,val}\right) + \mathbf{V}\left(E\right) && \text{By independence of } E \\ &= (B^* - B)\mathbf{V}(X_{t,val})(B^* - B)^T + \Sigma. && \text{Take } B^* - B \text{ out of } \mathbb{V}\left(\cdot\right) \end{aligned} \tag{5.9}$$

What remains now is to derive the $\mathbf{V}\left(X_{t,val}\right)$.

Deriving $\mathbf{V}\left(X_{t,val}\right)$. As we assume that E is a Gaussian, we know that $X_{t,val}$ will be a Gaussian as well. Furthermore, as we assume Our data is stationary with mean of $\mathbf{0}$, we know that $X_{t,val} \sim \mathcal{N}(\mathbf{0}, \Sigma_X)$. Note that this unconditional distribution does not depend on t. What is left is to estimate Σ_X. From existing literature on variational autoregressive models, we can easily derive the covariance of X_t. From e.g., [], we know that

$$X_{t+1} = AX_t + E \iff vec(\Sigma_X) = \left(I_{n^2} - (A \otimes A)\right)^{-1} vec(\Sigma),$$

where $vec(\cdot)$ represents the *vectorization operation*, and $\otimes$ represents the *kronecker product*.

Example 5.1. To make this notation clearer, consider this two-dimensional example where

$$A = \begin{pmatrix} 0.5 & 0 \\ 0.3 & 0.4 \end{pmatrix}, \qquad \Sigma = \begin{pmatrix} 1 & 0 \\ 0 & 2 \end{pmatrix}.$$

Performing the kronecker product $A \otimes A$ and subtracting that from I_{n^2} yields

$$\begin{aligned} I_4 - (A \otimes A) &= \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} - \begin{pmatrix} 0.5\begin{pmatrix} 0.5 & 0 \\ 0.3 & 0.4 \end{pmatrix} & 0\begin{pmatrix} 0.5 & 0 \\ 0.3 & 0.4 \end{pmatrix} \\ 0.3\begin{pmatrix} 0.5 & 0 \\ 0.3 & 0.4 \end{pmatrix} & 0.4\begin{pmatrix} 0.5 & 0 \\ 0.3 & 0.4 \end{pmatrix} \end{pmatrix} \\ &= \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} - \begin{pmatrix} 0.25 & 0 & 0 & 0 \\ 0.15 & 0.2 & 0 & 0 \\ 0.15 & 0 & 0.2 & 0 \\ 0.09 & 0.12 & 0.12 & 0.16 \end{pmatrix} \\ &= \begin{pmatrix} 0.75 & 0 & 0 & 0 \\ -0.15 & 0.8 & 0 & 0 \\ -0.15 & 0 & 0.8 & 0 \\ -0.09 & -0.12 & -0.12 & 0.84 \end{pmatrix}. \end{aligned}$$

Then, calculating its inverse and multiplying it with the factorization of Σ yields

$$\begin{aligned}\left(I_n^2 - (A\otimes A)\right)^{-1} vec(\Sigma) &= \begin{pmatrix} 0.75 & 0 & 0 & 0 \\ -0.15 & 0.8 & 0 & 0 \\ -0.15 & 0 & 0.8 & 0 \\ -0.09 & -0.12 & -0.12 & 0.84 \end{pmatrix}^{-1} \begin{pmatrix} 1 \\ 0 \\ 0 \\ 2 \end{pmatrix} \\ &\approx \begin{pmatrix} 1.33 \\ 0.25 \\ 0.25 \\ 2.60 \end{pmatrix}.\end{aligned}$$

Therefore, we end this example by noting that the covariance of X_t is

$$\Sigma_X = \begin{pmatrix} 1.33 & 0.25 \\ 0.25 & 2.60 \end{pmatrix}.$$

Now, returning to deriving the value of $C(P, A, P^*, A^*, E)$ in the population setting, we have that

$$vec\left(\mathbb{V}\left(X_{t,val}\right)\right) = \left(I_{n^2} - B^* \otimes B^*\right)^{-1} vec(\Sigma).$$

Putting it all together We conclude that the expected cost function in the population setting equals

$$\begin{aligned} C(P, A, P^*, A^*, E) &= \mathrm{Tr}\left(\mathbb{V}\left(X_{t+1,val} - X_{t+1,pred}\right)\right) && \text{By Equation 5.8} \\ &= \mathrm{Tr}(\mathbb{V}((B^* - B)X_{t,val} + E) && \text{By Equation 5.9} \\ &= \mathrm{Tr}\left(\mathbb{V}((B^* - B)X_{t,val}) + \mathbf{V}(E)\right) && \text{By independence of } E \\ &= \mathrm{Tr}\left((B^* - B)\mathbb{V}(X_{t,val})(B^* - B)^T\right) + \mathrm{Tr}\left(\Sigma\right). && \text{Take } B^* - B \text{ out} \end{aligned}$$

Global minimum of $C(A, P)$ Let us take a closer look at this cost function. The first question that arises is *what is the global minimum of this cost function?*. For this, we first remark that $\mathbb{V}(X_{t,val})$ is positive semi-definite. Furthermore, $\mathbf{V}(X_{t,val})$ is positive definite if it has full rank. Assuming our covariance is indeed full rank, we have that, for any B^*, B s.t. $B^* \neq B$,

$$(B^* - B)\mathbb{V}(X_{t,val})(B^* - B)^T > 0$$

and hence

$$C(P, A, P^*, A^*) = \mathrm{Tr}(\Sigma) \iff B^* = B.$$

In words, we can *only* attain the global minimum when we have exactly that our estimated matrix B is equal to the data generating matrix B^*.

This is a promising result for the population setting. We know that we can never attain a cost value smaller than the trace of Σ, and this value is only attained for A, P such that $P^{-1}AP = B^*$. Nevertheless, although there is only one global minimum with respect to B, there can still be many pairs of (A, P) to this specific value of B. Let us consider the scenario where

$$B^* = \begin{pmatrix} 0.5 & 0 & 0 \\ 0.3 & 0.4 & 0 \\ 0.2 & 0.0 & 0.6 \end{pmatrix}.$$

We only achieve our global minimum when $B = B^*$, but there are two pairs A_1, P_1 and A_2, P_2 that achieve this. The first pair is

$$A_1 = \begin{pmatrix} 0.5 & 0 & 0 \\ 0.3 & 0.4 & 0 \\ 0.2 & 0.0 & 0.6 \end{pmatrix}, P_1 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix},$$

and the second pair is

$$A_2 = \begin{pmatrix} 0.5 & 0 & 0 \\ 0.2 & 0.6 & 0 \\ 0.3 & 0.0 & 0.4 \end{pmatrix}, P_2 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}.$$

The reason for this is that there are multiple permutations possible, namely $(1, 2, 3)$ and $(1, 3, 2)$, as there is no dependence between our second and third variable. The fact that there are two ways to attain the global minimum is not a problem, however, as these are both compatible with the underlying graph and thus equally valid permutations.

A larger problem would be when the global minimum can also be achieved by a matrix P that is not a permutation matrix, but a doubly stochastic matrix. This would have problematic consequences, as the decoupling would make little sense then.

In short, an open question is, given a matrix $B^* = P^{-1}AP$, where P is a permutation matrix and A is a lower triangular matrix, does there exist a *doubly stochastic* matrix $\hat{P}_{DS}$ and a lower triangular matrix $\hat{A}$ such that $B^* = \hat{P}_{DS}^{-1}\hat{A}\hat{P}_{DS}$?. At the moment, the conjecture is that this is not the case, so the global optimum will only be reached using a valid permutation matrix P and a lower triangular matrix A.

Local minimum of $C(A, P)$ Now that we have seen that a global minimum of $C(A, P)$ indeed constitutes of a valid permutation matrix P and a lower triangular matrix A, the next question is whether there are any local optima that we can get stuck in. Given that our cost function is non-convex with respect to our matrix P, this suggests the presence of undesirable local optima. Furthermore, what is the nature of these local optima? Recall that a local optimum is where the gradient of the cost function is zero.

We indeed see that there are unpreferable local optima. Let us take a closer look at the nature of this local optimum. If we plot the cost functions and the gradients with respect to our entries in A, we see that the cost function is quadratic with respect to A, and hence the gradient is linear with respect to A. We see that all gradients are indeed zero for our values of A, indicating that it is indeed a local optimum with respect to A.

Let us now take a closer look at the cost function and the gradients with respect to our matrix P. As we can see, the matrix P is not a permutation matrix, but a doubly stochastic matrix. We already see that both the gradient and the cost function are not nice behaving functions. They have spikes which reach values up to $1e7$, meaning that the gradient is incredibly large for those values of p_{ij}. Another interesting thing we see is that most of the entries in p_{ij} are close to these spikes. Therefore, we see that the local optima occur where the doubly stochastic matrix P is very close to being non-singular. When inspecting the eigenvalues of P, we see that the first eigenvalue is equal to $-1e-3$, so being very close to zero. This is a major hurdle which is very difficult to overcome.

A solution that we would like to get is e.g. $A^* = B^*$, and P is the identity matrix. However, another global minimum exists as well, with

$$A =, P = .$$

Indeed, computing

Unfortunately, inferring the proper permutation matrix is the hardest part of this problem. For a total of p nodes, there are $p!$ permutations possible and therefore exponential in the number of nodes. The way we try to infer this is by *relaxing* the combinatorial constraint that P must be a permutation matrix. Instead of constraining P to be a permutation matrix, we only require P to be *doubly stochastic* matrix. In more precise words, we extend our search space from the set of permutation matrices to the Birkhoff polytope, the set of all doubly stochastic matrices. In this polytope, each of the vertices represents a matrix. Figure gives a visualization of the Birkhoff polytope for three dimensions.

We define the cost function to be

$$C(P, A, P^*, A^*, \Sigma^*) = \tag{5.10}$$

$$C(A, P, A^*, P^*) = \|X_{t,val} - X_{t,pred}\|_2^2,$$

where we use the following value to predict X_t:

$$X_{t,pred} = P^{-1}APX_{t-1}$$

In reality, there is some underlying P^*, A^* used to generate $X_{t,val}$:

$$X_{t,val} = P^{*-1}A^*P^*X_{t-1} + \Sigma.$$

Gradient of the cost function

Now that the cost function has been derived and analysed, the next topic of interest is inferring P^* and A^*. One of the most straightforward methods to infer these parameters is by performing *gradient descent.*

For this, we need to derive the partial derivatives of $C(P, A, P^*, A^*, E)$ with respect to its parameters. As only A and P here can be estimated, we write the cost function as $C(A, P)$. Let us first consider the gradient with respect to A.

Gradient of the cost function with respect to A. The gradient of $C(A, P)$ can be quite easily derived by using matrix derivatives. Furthermore, we know that computing the trace is a linear operation and thus, we can interchange the derivative and the trace.

$$\begin{aligned}
\frac{\partial C(P,A)}{\partial A_{ij}} &= \frac{\partial \mathrm{Tr}\left(\Sigma + (B^* - B)\,\Sigma_X\,(B^* - B)^T\right)}{\partial A_{ij}} && \text{(Fill in derived expected cost)} \\
&= \frac{\partial \mathrm{Tr}\,(\Sigma)}{\partial A_{ij}} + \frac{\partial \mathrm{Tr}\left((B^* - B)\,\Sigma_X\,(B^* - B)^T\right)}{\partial A_{ij}} && \text{(Separate sum of derivatives)} \\
&= \frac{\partial \mathrm{Tr}\,(\Sigma)}{\partial A_{ij}} + \mathrm{Tr}\left(\frac{\partial\,(B^* - B)\,\Sigma_X\,(B^* - B)^T}{\partial A_{ij}}\right) && \text{(Interchange trace and derivative)} \\
&= 0 + \mathrm{Tr}\left(\frac{\partial\,(B^* - B)\,\Sigma_X\,(B^* - B)^T}{\partial A_{ij}}\right) && \text{(Derivative of } \Sigma \text{ is 0)} \\
&= 2\mathrm{Tr}\left(\Sigma_X (B^* - B)^T \frac{\partial B^* - B}{\partial A_{ij}}\right) && \text{(Use chain rule)} \\
&= -2\mathrm{Tr}\left(\Sigma_X (B^* - B)^T P^{-1} J^{ij} P\right), && \text{(Work out last derivative)}
\end{aligned}$$

where

$$J^{ij}_{kl} = \begin{cases} 1 & k = i, l = j \\ 0 & \text{otherwise.} \end{cases}$$

In other words, J^{ij} is the matrix with zeros everywhere, except for the entry of the ith row and jth column, which is equal to one.

Gradient with respect to P Deriving the gradient of $C(A, P)$ with respect to P is slightly more involved than with respect to A, but nevertheless easily doable. Analogous to the gradient with respect to A, we first derive

$$\begin{aligned}
\frac{\partial C(P,A)]}{\partial P_{ij}} &= \frac{\partial \mathrm{Tr}\left(\Sigma + (B^* - B)\,\Sigma_X\,(B^* - B)^T\right)}{\partial P_{ij}} && \text{(Fill in derived expected cost)} \\
&= \frac{\partial \mathrm{Tr}\,(\Sigma)}{\partial P_{ij}} + \frac{\partial \mathrm{Tr}\left((B^* - B)\,\Sigma_X\,(B^* - B)^T\right)}{\partial P_{ij}} && \text{(Separate sum of derivatives)} \\
&= \frac{\partial \mathrm{Tr}\,(\Sigma)}{\partial P_{ij}} + \mathrm{Tr}\left(\frac{\partial\,(B^* - B)\,\Sigma_X\,(B^* - B)^T}{\partial P_{ij}}\right) && \text{(Interchange trace and derivative)} \\
&= 0 + \mathrm{Tr}\left(\frac{\partial\,(B^* - B)\,\Sigma_X\,(B^* - B)^T}{\partial P_{ij}}\right) && \text{(Derivative of } \Sigma \text{ is 0)} \\
&= 2\mathrm{Tr}\left(\Sigma_X (B^* - B)^T \frac{\partial B^* - B}{\partial P_{ij}}\right) && \text{(Use chain rule)} \\
&= -2\mathrm{Tr}\left(\Sigma_X (B^* - B)^T \left(-P^{-1} J^{ij} P^{-1} A P + P^{-1} A J^{ij}\right)\right), && \text{(Work out last derivative)}
\end{aligned}$$

where

$$J^{ij}_{kl} = \begin{cases} 1 & k = i, l = j \\ 0 & \text{otherwise.} \end{cases}$$

To verify our derivations, we have plotted the

Inferring A and P

Now that we have a gradient for A and for P, we will describe multiple methods to infer A^* and P^*. A straightforward method to infer A^* in the population setting is as follows. We first estimate A^* without constraining A to be lower triangular. As we know that A^* is the weighted adjacency matrix of an actual directed acyclic graph, we know that there must exist at least one permutation matrix P_{perm} such that $A^* = PT^*P$, where T^* is lower triangular. Below, we give a straightforward algorithm to compute such valid permutation P.

Algorithm 4 Topological Sort

```
**Input:** A matrix $A$ that represents the weighted adjacency matrix of a directed acyclic graph.
**Output:** A permutation matrix $P$ that represents a valid topological ordering of the nodes of $G(A)$.
Set all diagonal entries of $A$ to zero.
while $A$ is nonempty do
    Let $i$ be the index of the row in $A$ with only zeros
    Remove this $i$th row from the matrix $A$
    Remove the $i$th column from the matrix $A$
end while
```

When A^* is the weighted adjacency matrix of a graph that is cyclic, which directed graph would then be the best solution? Is the problem even well-defined then?

Non-population setting

5.1.3 Using Birkhoff's Theorem

Another interesting method is by using Birkhoff's Theorem. Birkhoff's Theorem states the following.

Birkhoff's Theorem. Every doubly stochastic matrix can be rewritten as a *convex combination* of permutation matrices.

In more concrete words, let $P_1, P_2, \cdot, P_{n!}$ be all $n!$ possible permutation matrices. Given a doubly stochastic matrix P_{DS}, there exists non-negative $\lambda_1, ...\lambda_{n!} \in \mathbb{R}$, together summing to one, such that

$$P_{DS} = \sum_{i=1}^{n!} \lambda_i P_i.$$

This interesting result can be utilized in many different settings. Recall our model

$$X_{t+1} = P^{-1}APX_t.$$

Here, P is equal to a doubly stochastic matrix. Inferring these matrices A and P is quite difficult. Especially estimating a permutation matrix P is difficult given the combinatorial search space of all $n!$ permutation matrices. An interesting approach to circumvent this combinatorial search space is by relaxing the constraint from the set of permutation matrices to the set of doubly stochastic matrices. The second question then is how to deal with the new problems that arise when relaxing tho the space of doubly stochastic matrices. For example, the constraints that all entries must be between zero and one, and all rows must sum to one, and all columns must sum to one, require quite a large set of linear inequalities.

One way of dealing with this is by, instead of learning the entries of the douvly stochastic matrix directly, we learn the values λ_i, $i = 1, \cdots, n!$. So, we relax from requiring one specific permutation matrix to a convex combination of the existing permutation matrices. A problem, however, is that there are $n!$ permutation matrices, meaning that we now have a total of $\mathcal{O}(n!)$ permutations, which will not be feasible for large n. However, it is worth investigating this approach for moderately small n.

In more precise notation, let our cost function again by

$$\mathbb{E}\left[\|X_{val} - X_{pred}\|_2^2\right].$$

Then,

$$A, \{\lambda\}_{i=1}^{n!} = \arg \min_{A, \{\lambda\}_{i=1}^{n!}} \mathbb{E}\left[\|X_{val} - X_{pred}\|_2^2\right],$$

where

$$X_{val} = P^{*-1}A^*P^*,$$

and

$$X_{pred} = \left(\sum_{i=1}^{n!} \lambda_i P_i\right)^{-1} A \left(\sum_{i=1}^{n!} \lambda_i P_i\right).$$

5.1.4 As a product of row-swapping matrices

This method is similar to Subsection 5.1.3., "Using Birkhoff's Theorem". In that Subsection, we discussed the method of relaxing the space permutation matrix $\mathcal{P}_{perm}$ to the space of double stochastic matrices $\mathcal{P}_{DS}$. Consequently, we rewrote a double stochastic matrix $P_{DS} \in \mathcal{P}_{DS}$ as a convex combination of permutation matrices. Unfortunately, there are $n!$ permutation matrices and hence, an exponential number of parameters would have to be learnt and updated.

In this section, we discuss another method to relax the space of permutation matrices. For this, we define P_{ij} as the identity matrix with row i and row j swapped. As an example, consider three dimensions. Then, we have that

$$P_{13} = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix}.$$

Now, what is interesting, is that we can write any permutation matrix P as a product of $n-1$ of these row-swapping matrices. To see why this is the case, consider any permutation P. Let π be the ordering corresponding to the permutation matrix P, i.e., $\pi(i)$ contains the index of the row where the ith index is equal to 1. As an example, a permutation matrix P with corresponding order is

$$\begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}, \qquad \pi = (2, 3, 1).$$

Then, this permutation matrix P can be attained by only $n-1=2$ row-swapping matrix multiplications, namely $P_{1,3}P_{2,3}$.

In the more general setting, we can always write any permutation matrix as a product of $n-1$ row-swapping matrices using a greedy approach. The first row-swapping matrix will swap the first row with the row of which the first index is equal to 1, so $P_{1,\pi(1)}$. The first row is now in the correct location. We continue with the subproblem of size $n-1$. The second row-swapping matrix will swap the second row with the row of which the second index is equal to 1, so $P_{2,\pi(2)}$. Clearly, we can do this iteratively until the first $n-1$ rows are in the correct location, which is after at most $n-1$ swaps. If the first $n-1$ rows are in the correct location, we can immediately deduce that the row with the nth index equal to 1 must be in the last row. Therefore, we can write any permutation matrix P as a product of $n-1$ row-swapping matrices.

Of course, these $n-1$ row-swapping matrices need not be the same for any permutation P. For one matrix, we swap row 1 and row 3, whereas for another matrix, we swap row 1 and row 2. Luckily, we know that we can write any matrix using $n-1$ row-swapping matrices, where we pick these from a total of $\binom{n-1}{2}$ matrices.

For the first swap, we can swap row 1 with any of the other $n-1$ rows, yielding $n-1$ possibilities. For the second swap, we swap row 2 with any of the remaining $n-2$ rows. This yields

$$\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} 1 = \sum_{i=1}^{n-1} n - i = \sum_{i=1}^{n-1} i = \frac{(n-1)(n)}{2} = \begin{pmatrix} n-1 \\ 2 \end{pmatrix}.$$

The main point here is that we can write a permutation matrix using only a linear number of row-swapping matrices, and the search space of all these matrices is quadratic. Therefore, the total number of parameters corresponding to this permutation matrix is significantly smaller than our approach involving Birkhoff's theorem, which yielded $n!$ parameters.

Now, let α_{ij} be a variable that is either zero or one, and it denotes whether we use this row-swapping matrix. By the convention we have that $P_{ij}^0 = I$, where I is the identity matrix. Furthermore, we have that $P_{ij}^1 = P_{ij}$.

Therefore, we can write any permutation matrix P as

$$P = \prod_{i=1}^{n-1} \prod_{j=i+1}^{n} P_{ij}^{\alpha_{ij}}, \qquad \alpha_{ij} \in \{0, 1\}.$$

Furthermore, the transpose or inverse is also easily calculated with respect to $P_{ij}^{\alpha_{ij}}$:

$$P^{-1} = \left(\prod_{i=1}^{n-1} \prod_{j=i+1}^{n} P_{ij}^{\alpha_{ij}} \right)^{-1} = \prod_{i=1}^{n-1} \prod_{j=i+1}^{n} (P_{ij}^{\alpha_{ij}})^{-1} = \prod_{i=1}^{n-1} \prod_{j=i+1}^{n} (P_{ij}^{\alpha_{ij}})^{T}.$$

Alternatively, another option is by reversing the order of the row-swapping matrices P_{ij}, so

$$P^{-1} = P^T = \left(\prod_{i=1}^{n-1} \prod_{j=i+1}^{n} P_{ij}^{\alpha_{ij}} \right)^T .$$

Relaxing α We see that we now have parametrized our permutation matrix P by the parameters $\alpha \in \{0,1\}^{\binom{n-1}{2}}$. However, learning these parameters is still not easy, as the α_{ij} are non-continuous. Therefore, we will try to relax this constraint by stating $\alpha_{ij} \in \mathbb{R}$. The hope is that relaxing this constraint means that we can use a continuous optimization procedure to learn our parameters α such as gradient descent. Let us see whether this is possible.

Let us first consider the matrix $P_{ij}^{\alpha_{ij}}$ for $\alpha_{ij} \in \mathbb{R}$. For $\alpha \in \mathbb{R}$, we have a *fractional matrix power.* The fractional matrix power with exponent $\alpha \in \mathbb{R}$ is defined as

$$A^\alpha = e^{\alpha \log(A)}$$

Eigendecomposition of P_{ij} Luckily, our matrices P_{ij} are all quite simple, only two rows are swapped. Therefore, let us consider the eigendecomposition of P_{ij}. We have that the eigenvalues of P_{ij} are

$$\lambda = \left(1, \ldots, 1, \underbrace{-1}_{\text{index } j}, 1, \ldots, 1 \right).$$

Furthermore, the corresponding eigenvector $\mathbf{u}_k$ of λ_k is equal to

$$\mathbf{u}_k = \begin{cases} e_k & \text{if } k \neq i \text{ and } k \neq j \\ \frac{1}{2}\sqrt{2}\,(e_i + e_j) & \text{if } k = i \\ \frac{1}{2}\sqrt{2}\,(e_i - e_j) & \text{if } k = j \end{cases},$$

where e_k denotes the vector with all zeros except a value one on the kth entry. Therefore, we can rewrite P_{ij} in its eigendecomposition

$$P_{ij} = U \Lambda U^T,$$

where U is the matrix with all eigenvectors stacked, and $\Lambda = \operatorname{diag}(\lambda)$.

Closed form of $P_{ij}^{\alpha_{ij}}$ Now, including the matrix exponent is quite easy when we have the eigendecomposition. We have that

$$P_{ij}^{\alpha_{ij}} = U \Lambda^{\alpha_{ij}} U^T.$$

As Λ is a diagonal matrix, its fractional matrix power can be computed by computing the power of all elements separately. As $n-1$ eigenvalues are equal to 1, we know that they remain the same, as $1^{\alpha_{ij}} = 1$ for any value of α_{ij}. Only for the eigenvalue with value -1, we interestingly get a value in the complex plane, as

$$-1^{\alpha_{ij}} = \cos(\pi\alpha_{ij}) + i \sin(\pi\alpha_{ij}).$$

We can put this all together to have a closed form solution for our matrix $P_{ij}^{\alpha_{ij}}$. Consequently, we also have a closed form solution for our matrix P, which is simply the product of all these matrices $P_{ij}^{\alpha_{ij}}$.

An important feature of $P_{ij}^{\alpha_{ij}}$ is that although its elements are complex numbers, $P_{ij}^{\alpha_{ij}}$ is still a doubly stochastic matrix for any value of α_{ij}, meaning that both its rows and columns sum up to 1 (and hence the imaginary part is equal to zero). Furthermore, we see that $P_{ij}^{\alpha_{ij}} = P_{ij}^{\alpha_{ij}+2k}$ for $k \in \mathbb{Z}$. Lastly, to compute its inverse, we can simply take the conjugate transpose of all matrices

$$P^{-1} = \left(\prod_{i=1}^{n-1} \prod_{j=i+1}^{n} P_{ij}^{\alpha_{ij}} \right)^{-1} = \prod_{i=1}^{n-1} \prod_{j=i+1}^{n} (P_{ij}^{\alpha_{ij}})^{-1} = \prod_{i=1}^{n-1} \prod_{j=i+1}^{n} (P_{ij}^{\alpha_{ij}})^{H}.$$

Alternatively, another option is by reversing the order of the row-swapping matrices $P_{ij}^{\alpha_{ij}}$ and taking their complex conjugates, so

$$P^{-1} = P^H = \left(\prod_{i=1}^{n-1} \prod_{j=i+1}^{n} \overline{P_{ij}^{\alpha_{ij}}} \right)^T .$$

Influence of α_{ij} on the cost value Now that we have a proper mathematical overview of how we can decompose this matrix P, let us consider how these α_{ij} will influence the cost function. Recall that throughout this thesis we will use the squared 2-norm of the residual error:

$$C(A,P) = \|X_{[1:T]} - X_{[0:T-1]}P^{-1}AP\|_2^2.$$

Let us denote

$$C(P) = \underset{A\text{lower triangular}}{\arg\min} C(A,P).$$

Given our permutation matrix P, we can easily derive this quantity using least squares. However, linear least squares will likely not work here as our P is not necessary a permutation matrix but a doubly stochastic matrix.

Nevertheless, we see that using certain values of α_{ij} can lead to a cost value smaller than the optimum of the original problem. This is problematic, as the optimum of the new problem does not coincide with the optimum of the old problem. Hence, this relaxation in its current form is not a good fit. A method to harshly penalize the complex part of the solution is necessary.

To see this, we start with a simple two-dimensional example, as then we only have one parameter α_{ij} and our permutation matrix is of the form

$$P = P_{12}^{\alpha}.$$

Let the true underlying weighted adjacency matrix be

$$W^* = \begin{pmatrix} 0.5 & 0 \\ 0.5 & 0.5 \end{pmatrix},$$

which corresponds to the permutation matrix P^* and lower triangular matrix A^*

$$P^* = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \qquad A^* = \begin{pmatrix} 0.5 & 0 \\ 0.5 & 0.5 \end{pmatrix}.$$

Now, the α_{ij} that we hope to find would be $\alpha_{ij} = 0$, so that $P = P^*$. We sample $T = 50$ samples according to this model. This yields

$$C(A^*, P^*) \approx .$$

Now, what happens if we change our value of α? We have plotted the cost function $C(A^*, P_{12}^{\alpha})$. We see that the cost function behaves nicely with respect to α and the optimum is clearly at $\alpha = 0 + 2k, k \in \mathbb{Z}$. The reason for this is that $P_{ij}^{\alpha_{ij}}$ is cyclic with respect to α with a period of 2 (due to the cyclicty of $\cos(\pi\alpha) + i\sin(\alpha)$. Furthermore, we see that the cost function is symmetric around $\alpha = 1$. The reason for this symmetry is because

$$-1^{1+\alpha} = \cos(\pi + \pi\alpha) + i\sin(\pi + \pi\alpha) = \cos(\pi\alpha) - i\sin(\pi\alpha),$$

and hence

$$P_{ij}^{1+\alpha} = \overline{P_{ij}^{1-\alpha}}$$

. As the squared 2-norm of a complex number is equal for its complex conjugate, we see that

$$C\left(A, P_{ij}^{1+\alpha}\right) = C\left(A, P_{ij}^{1-\alpha}\right)$$

5.1.5 Learning using Lagrange Multipliers

As usual, we are interested in the following model:

$$X_{t+1} = P^{-1}APX_t + E$$

We have

$$C(A,P) = \operatorname{Tr}\left((B^* - B)\Sigma_X(B^* - B)\right),$$

where

$$B = P^{-1}AP.$$

Here, A is lower triangular and P is a doubly stochastic matrix, namely

$$
\begin{aligned}
0 \leq & p_{ij} && \forall\ i, j = 1, \cdots, n \\
\sum_{j=1}^{n} & p_{ij} = 1 && \forall\ i = 1, \cdots, n \\
\sum_{i=1}^{n} & p_{ij} = 1 && \forall\ j = 1, \cdots, n
\end{aligned}
$$

To deal with the $2n$ equality constraints, we use *lagrange multipliers* $\lambda_{row,i}$, $i = 1, \cdots, n$ and $\lambda_{col,j}$, $j = 1, \cdots, n$. The dual function now becomes

$$\mathcal{L}(A, P, \lambda) = C(A, P) - \sum_{i=1}^{n} \lambda_{row,i} \left(\sum_{j=1}^{n} p_{ij} - 1 \right) - \sum_{j=1}^{n} \lambda_{col,j} \left(\sum_{i=1}^{n} p_{ij} - 1 \right).$$

Let

$$q(\lambda) = \inf_{A,P} \mathcal{L}(A, P, \lambda).$$

Then the *dual problem* is

$$\max_{\lambda} q(\lambda).$$

To minimize $\mathcal{L}(A, P, \lambda)$, we will use gradient descent.

Gradient of $\mathcal{L}(A, P\lambda)$

The partial derivatives with respect to entries in A are simple,

$$\frac{\partial \mathcal{L}(A, P_\sigma, \lambda)}{\partial a_{ij}} = \frac{\partial C(A, P)}{\partial a_{ij}}$$

Now, the partial derivative with respect to the entries in p is more involved as it occurs also in exactly two equality constraints, namely in $\lambda_{row,i}$ and in $\lambda_{col,j}$.

$$\frac{\partial \mathcal{L}(A, P_\sigma, \lambda)}{\partial p_{ij}} = \frac{\partial C(A, P)}{\partial p_{ij}} - \lambda_{row,i} - \lambda_{col,j}.$$

Now, as $C(A, P)$ is unfortunately yet inevitably non-convex, we cannot expect to find the infimum $q(\lambda)$. However, let us consider a local minimum $\tilde{q}(\lambda)$, which we will find by gradient descent. Let us start with an initial matrix A_0 (e.g. the non-zero matrix) and an initial unconstrained matrix $P_0 = Z$, where $Z \sim \mathcal{N}(\mathbf{0}, I_{n^2})$.

For a sufficiently small step size, we will do this gradient descent until we have found a stationary point:

$$(A_{t+1}, P_{t+1}) = (A_t, P_t) - \eta\ \nabla \mathcal{L}(A_t, P_t, \lambda)$$

Note that this is an *unconstrained* optimization problem now, which should make things more easy.

Suppose that after a number of iterations, we have found this local minimum with parameters $\tilde{A}$, $\tilde{P}$. Then, we have that

$$\tilde{q}(\lambda) = \mathcal{L}(\tilde{A}, \tilde{P}, \lambda).$$

Now that we have $\tilde{q}(\lambda)$, the solution to the dual problem is

$$\max_{\lambda} \tilde{q}(\lambda),$$

which is again an unconstrained optimization problem, which we can also optimize by maximizing λ's.

$$\tilde{q}(\lambda) = C(\tilde{A}, \tilde{P}) - \sum_{i=1}^{n} \lambda_{row,i} \left(\sum_{j=1}^{n} \tilde{p}_{ij} - 1 \right) - \sum_{j=1}^{n} \lambda_{col,j} \left(\sum_{i=1}^{n} \tilde{p}_{ij} - 1 \right).$$

In order to maximize $\tilde{q}(\lambda)$ with respect to our lagrange multipliers, we note that $\tilde{q}(\lambda)$ is linear with respect to λ, meaning that

$$\frac{\tilde{q}(\lambda)}{\lambda_{row,i}} = \sum_{j=1}^{n} p_{ij} - 1,$$
$$\frac{\tilde{q}(\lambda)}{\lambda_{col,j}} = \sum_{i=1}^{n} p_{ij} - 1.$$

Unfortunately, these derivatives are constant and non-zero with respect to our lagrange multipliers. This means that there are no stationary points, and hence no local optima that we can hope to find using this approach. If a row or column i of our doubly stochastic matrix is less than one, then the gradient will be negative, meaning that we increase $\lambda_{row/col,i}$. If a row or column j of our doubly stochastic matrix is larger than one, then the gradient will be positive, meaning that we decrease $\lambda_{row/col,i}$ indefinitely. Only when a row our column i of the doubly stochastic matrix sums exactly to one, then the gradient will be zero.

This suggests an iterative approach. While the gradient of both are non-zero

- Use gradient descent to find $\tilde{q}(\lambda)$.
- Use one gradient descent step to update λ.

In the end, we have reached a local optimum that also satisfies the constraints. A large advantage of using this lagrangian multiplier method is that we have an unconstrained optimization problem now.

5.1.6 NOTEARS for VAR(1) models

We can use the NOTEARS approach also for VAR(1) models. For this, we need to make a couple of modifications.

Changes to the loss function $\ell(W; \mathbf{X})$. We change the loss from

$$\ell(W; \mathbf{X}) = \|\mathbf{X} - \mathbf{X}W\|_F^2$$

to

$$\ell(W; \mathbf{X}) = \frac{1}{2(n-1)} \|X_{[1:T]} - X_{[0:T-1]}W\|_F^2.$$

Consequently, the gradient then becomes

$$\nabla \ell(W; \mathbf{X}) = -\frac{1}{n-1} X_{[0:T-1]} \|X_{[1:T]} - X_{[0:T-1]}W\|_F.$$

Changes to the DAGness $h(W)$. The function $h(W)$ from NOTEARS was designed such that

$$h(W) = 0 \iff G(W) \text{ is a DAG.}$$

However, for our VAR(1) models, we do allow for self-loops. Therefore, we need to adjust the function $h(W)$ accordingly. Originally, the function was

$$h(W) = e^{W \circ W},$$

where $\circ$ represents the Hadamard-Product, or the element-wise multiplication of two matrices of equal dimension. The required modification is quite simple. We simply set all diagonal entries of W to zero. In other words, we propose the following function for VAR(1) models:

$$h'(W) = e^{\tilde{W} \circ \tilde{W}},$$

where the entries of $\tilde{W}$ are

$$\tilde{w}_{ij} = \begin{cases} w_{ij} & \text{if } i \neq j \\ 0 & \text{if } i = j. \end{cases}$$

Changes to the set-up. Apart from these two fundamental changes, some other minor changes needed to be done to be done to the NOTEARS repository to ensure the method also works for VAR(1) models. Firstly, we have added a method that, given a weighted adjacency matrix W, simulates data according to the VAR(1) model. Secondly, we have added the possibility for self-loops when generating the weighted adjacency matrix W. Lastly, we adjusted the bounds of the optimization procedure such that diagonal entries were not constrained to zero.

Furthermore, the elements in W were in the range of $[-2, -0.5] \cup [0.5, 2.0]$ for the original NOTEARS paper, where they assumed a SEM model. This is perfectly fine in their setting. In the VAR(1) setting, however, we must be extra careful with the diagonal entries. If they are in absolute value larger than one, we get a non-stationary series, which keeps growing. Furthermore, if one of the diagonal entries is negative, then we will see an oscillating behavior, because the sign will change with every timestep. Although this is not necessarily an issue, it is highly unlikely for any real dataset to have such an oscillating behavior. Therefore, we will keep the values on the diagonal between zero and one.

5.1.7 Learning a DAG by learning a permutation

We assume the model

$$\mathbf{X}_t = A\mathbf{X}_{t-1} + \epsilon$$

Now, we are looking for a permutation π, or equivalently, a permutation matrix P, such that PA is strictly lower triangular. Here, A denotes the weighted adjacency matrix (WAM).

$$G(A) \text{ is a DAG } \iff \exists \text{ a permutation matrix P s.t. } PAP^T \text{ is lower triangular.}$$

To find this permutation matrix, we will relax the constraints on P. Rather than requiring P to be a permutation matrix, we will require P to be *doubly stochastic.* This means that every row and every column of P must sum to one. In mathematical notation,

$$\sum_{i=1}^{p} P_{ij} = \sum_{j=1}^{p} P_{ij} = 1, \ \forall\ i = 1, \cdots, p, j = 1, \cdots, p.$$

To give a more concrete example, consider the model defined in Equation 5.11, consisting of three variables X, Y, and Z.

$$\mathbf{X}_t = \begin{pmatrix} X_t \\ Y_t \\ Z_t \end{pmatrix} = \begin{pmatrix} 0.4 & 0.3 & 0.2 \\ 0 & 0.5 & 0.4 \\ 0.9 & 0.0 & 0.0 \end{pmatrix} \begin{pmatrix} X_{t-1} \\ Y_{t-1} \\ Z_{t-1} \end{pmatrix} + \begin{pmatrix} \epsilon_t & 0 & 0 \\ 0 & \epsilon'_t & 0 \\ 0 & 0 & \epsilon''_t \end{pmatrix}, \tag{5.11}$$

Writing out the matrix multiplications yields the model

$$\begin{aligned} X_t &= 0.4X_{t-1} + 0.3Y_{t-1} + 0.2Z_{t-1} + \epsilon_t, \\ Y_t &= 0.5Y_{t-1} + 0.4Z_{t-1} + \epsilon'_t, \\ Z_t &= 0.9Z_{t-1} + \epsilon''_t. \end{aligned}$$

When using arrows to express dependencies in Equation 5.11, will result the graph depicted in Figure 5.1. Self-loops denoting the dependence of the variable on its own past were omitted, and the edge labels denote the weight of each edge.

Clearly, we see that (X, Y, Z) is not a topological ordering, as X_t depends on both Y_{t-1} and Z_{t-1}. However, we see that (Z, Y, X) is a valid topological ordering, as each variable only depends on its predecessors. Therefore, we are interested in learning this topological ordering, or equivalently, learning a permutation matrix P such that PAP^T is lower triangular. Ideally, we would like to retrieve the following P and A:

$$P = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix}, \qquad A = \begin{pmatrix} 0.9 & 0 & 0 \\ 0.4 & 0.5 & 0 \\ 0.2 & 0.3 & 0.4 \end{pmatrix}.$$

Example 5.1.1. Let us consider the model given in Equation 5.12:

$$X_t = aX_{t-1} + \epsilon_t, \qquad Y_t = X_{t-1} + \epsilon'_t, \tag{5.12}$$

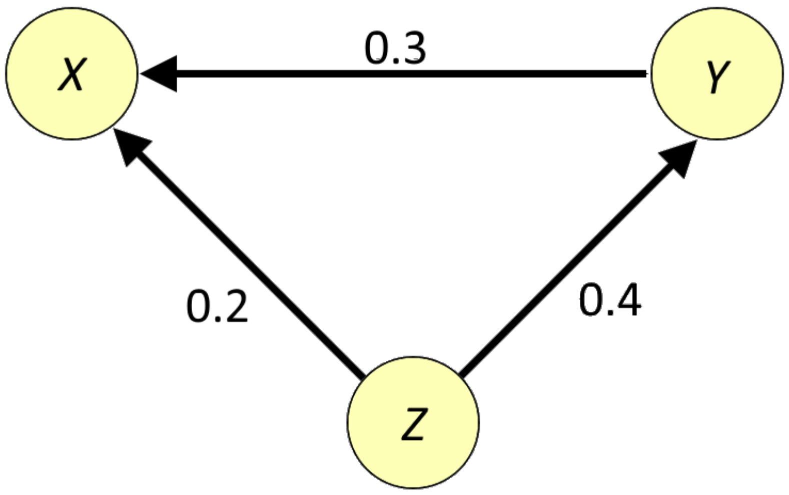


Figure 5.1: Graph of the model defined in Equation 5.11.

where $a \in [0, 1)$, and ϵ_t and ϵ'_t are both independently and identically distributed standard normal variables for all values of $t = 1, ..., T$. This model can be interpreted as X_t being some noisy hidden state variable, and Y_t being a noisy observation of the hidden state X at time $t-1$.

Clearly, X_{t-1} is more useful for predicting Y_t than Y_{t-1} is for predicting X_t. If we know X_{t-1}, then we know Y_t up to some noise ϵ'_t. However, if we know Y_{t-1}, then we know X_{t-1} up to some noise $-\epsilon'_t$, and more importantly, we only know X_t up to this noisy estimate of X_{t-1} multiplied by a, plus an additional noise ϵ_t. Therefore, when we want to decide on the directionality of the predictive effect (so $X \to Y$ or $X \leftarrow Y$), we want our method to choose the direction $X \to Y$.

Minimization method. For this simple example with two one-dimensional variables, we can work out our model by hand. For this, we define our minimization problem as follows:

$$\arg \min_{P \subseteq \mathcal{P}, A \in \mathbb{R}^{2\times 2}} \sum_{t=2}^{T} ||P\mathbf{X}_t - AP\mathbf{X}_{t-1}||_2^2. \tag{5.13}$$

Here, $\mathcal{P}$ is the set of all doubly stochastic 2×2 matrices, and A is a 2×2 upper triangular matrix. Interestingly, a two-dimensional doubly stochastic matrix P can be uniquely defined by only one entry. As P must be doubly stochastic, both diagonal entries must be equal, which we define to be p. As both rows and columns must both sum to 1, we know that both off-diagonal entries must be equal to $1 - p$. This value $p \in [0, 1]$ denotes how likely the permutation will be. If p is close to zero, then the permutation (Y, X) is more likely, hence $X \to Y$. If p is close to one, then the permutation (X, Y) is more likely, hence $X \leftarrow Y$. Writing out this two-dimensional scenario yields the following doubly stochastic matrix P and upper triangular matrix A:

$$P = \begin{pmatrix} p & 1-p \\ 1-p & p \end{pmatrix}, \qquad A = \begin{pmatrix} a_{11} & a_{12} \\ 0 & a_{22} \end{pmatrix}.$$

Then, the minimization problem of Equation 5.13 becomes

$$\arg\min_{p\in[0,1], a_{11}, a_{12}, a_{22} \in \mathbb{R}} \sum_{t=2}^{T} \left\| \begin{pmatrix} p & 1-p \\ 1-p & p \end{pmatrix} \begin{pmatrix} X_t \\ Y_t \end{pmatrix} - \begin{pmatrix} a_{11} & a_{12} \\ 0 & a_{22} \end{pmatrix} \begin{pmatrix} p & 1-p \\ 1-p & p \end{pmatrix} \begin{pmatrix} X_{t-1} \\ Y_{t-1} \end{pmatrix} \right\|_2^2. \tag{5.14}$$

To gain a deeper understanding of Equation 5.14, let us consider the boundary values that p can attain. When $p = 0$, this yields the model

$$X_t = a_{22} X_{t-1}, \qquad Y_t = a_{11} Y_{t-1} + a_{12} X_{t-1}. \tag{5.15}$$

When $p = 1$, this yields the model

$$X_t = a_{11} X_{t-1} + a_{12} Y_{t-1}, \qquad Y_t = a_{22} Y_{t-1}. \tag{5.16}$$

If $0 < p < 1$, we will get a mixture of these two equations:

$$\begin{aligned} pX_t + (1-p)Y_t &= a_{11}\left(pX_{t-1} + (1-p)Y_{t-1}\right) + a_{12}\left((1-p)X_{t-1} + pY_{t-1}\right), \\ pY_t + (1-p)X_t &= a_{22}\left((1-p)X_{t-1} + pY_{t-1}\right). \end{aligned} \tag{5.17}$$

Equation 5.12 for $a = 0.8$. Let us consider a concrete instance of Equation **??**. We have generated a time series with time horizon $T = 100$ according to Equation 5.12 with $a = 0.8$. The resulting time series are plotted in Figure 5.2.

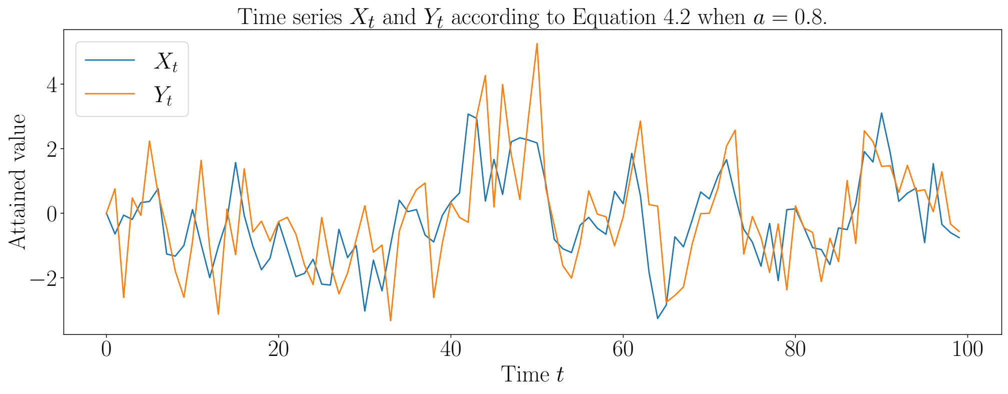


Figure 5.2: Vizualization of time series X_t and Y_t according to Equation 5.12 with $a = 0.8$.

Applying our minimization technique from Equation 5.14, this yielded the following estimates:

$$\hat{p} = 0.35, \qquad \hat{A} = \begin{pmatrix} -1.06 & 1.98 \\ 0 & 0.853 \end{pmatrix}.$$

We see that the value for p is significantly smaller than 0.5, which would indicate that it is better to use X_{t-1} to predict Y_t than to use Y_{t-1} to predict X_t. Hence, our method suggests that the predictive effect $X \to Y$ is larger than the predictive effect $X \leftarrow Y$, just as we would expect. Therefore, this method produces the correct outcome.

Unfortunately, the matrix $\hat{A}$ does not resemble the original matrix A. The reason for this is that our value for p is still quite far away from either zero or one, and hence our model is a mixture of the equations as described in Equation 5.17. When we would force p to be either zero or one, for example by rounding p to the nearest integer, the minimization problem in this scenario becomes

$$\underset{a_{11}, a_{12}, a_{22} \in \mathbb{R}}{\arg\min} \sum_{t=2}^{T} \left\| \begin{pmatrix} Y_t \\ X_t \end{pmatrix} - \begin{pmatrix} a_{11} & a_{12} \\ 0 & a_{22} \end{pmatrix} \begin{pmatrix} Y_t \\ X_t \end{pmatrix} \right\|_2^2. \tag{5.18}$$

Solving Equation yields the following estimate of A:

$$\hat{A} = \begin{pmatrix} 0.01 & 1.01 \\ 0 & 0.80 \end{pmatrix},$$

which is very close to the actual model.

Values of p in Equation 5.12 for different values of a To see the influence of the coupling strength of a, we have tried different values for a and computed the corresponding value for p. These results are shown in Figure 5.3.

We see that as a increases, the value for p decreases. In other words, as we increase the dependence of Y_t on X_{t-1}, the value for p decreases towards zero, meaning the data leans more towards Equation 5.15 rather than Equation 5.16. Given that we generated the data according to Equation 5.15, this is as expected. However, the decrease is not as steep as we would have hoped; the values for p are indeed smaller than 0.5, but also remain larger than 0.3 for all suitable values of a.

Example 5.1.2. *Two-dimensional AR(1) model.*

Now consider the more generic two-dimensional model

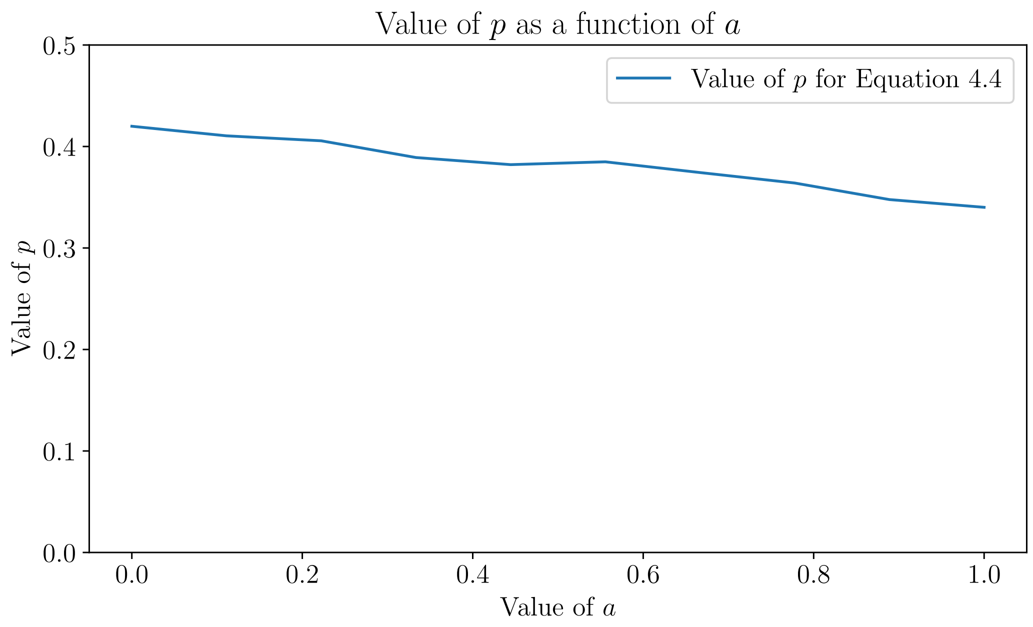


Figure 5.3: Value of p as a function of a in Equation 5.12.

$$\begin{pmatrix} X_t \\ Y_t \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} X_{t-1} \\ Y_{t-1} \end{pmatrix} + \begin{pmatrix} \epsilon_t \\ \epsilon'_t \end{pmatrix}. \tag{5.19}$$

Equation when $a_{11} = a_{22}$ and $a_{12} = a_{21}$. Let us consider the scenario where X and Y predict each other equally.

$$X_t = aX_{t-1} + bY_{t-1} + \epsilon_t,$$
$$Y_t = aX_{t-1} + bY_{t-1} + \epsilon'_t.$$

From this equation, we can clearly see that X_{t-1} predicts Y_t equally well as Y_{t-1} predicts X_t. Hence, we would expect our method to report values of p close to a half. We have created a contour plot in Figure that shows the value of p for a given pair (a, b). Note that the contour plot concerns only pairs (a, b) where $a + b < 1$, as larger values would mean that X and Y would diverge towards either positive or negative infinity.

We see that for all pairs (a, b), the value p remains close to a half. The small deviations of p are most likely caused by the randomness of ϵ_t and ϵ'_t.

Equation when $a_{12} = a_{22} = 0$. We are also interested in the scenario where X is a clear influencer of Y, so

$$X_t = a_{11}X_{t-1} + \epsilon_t$$
$$Y_t = a_{21}X_{t-1} + \epsilon'_t$$

Clearly, X_{t-1} is a useful predictor for Y_t. The question is how much the parameters a_{11} and a_{21} influence the values of p for detecting the direction of this predictive effect. Lower values of a_{21} decrease the predictive effect of X_{t-1} on Y_t, as the noise component ϵ'_t becomes more dominant. Furthermore, lower values of a_{11} result in smaller values of X_t, which also results in the noise component ϵ'_t to become more dominant.

We have created a contour plot in Figure that shows the value of p for a given pair (a_{11}, a_{21}). A brighter more yellow color indicates that p is close to 0.5. A darker more blueish color indicates that p is smaller than 0.5. Figure tells us that indeed, the value of p increases as either a_{11} or a_{21} increases, indicating that the predictive effect $X \to Y$ increases. This is in line with what was expected.

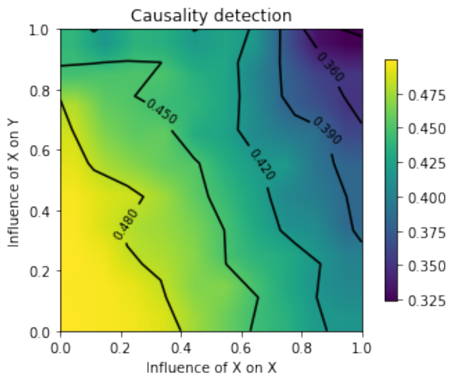


Figure 5.4: first figure

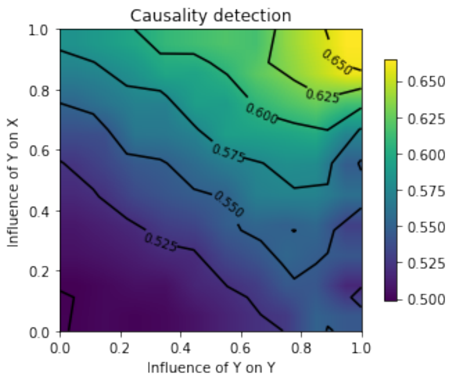


Figure 5.5: second figure

To verify whether our approach is symmetric, we will also investigate where the roles of X and Y are switched around, so now $a_{11} = a_{21} = 0$. As Y_{t-1} is now a more useful predictor of X_t, we expect higher values for p. Just as for the previous example, we expect larger values for both a_{12} and a_{22} to yield larger values for p. We have created a countour plot in Figure that shows the values of p for a given pair (a_{12}, a_{22}).

In Example 1 and Example 2, we have seen that our method seems to perform well in the simple two-dimensional setting, reporting the correct direction of the predictive effect. Furthermore, when there is no predictive effect, the method reports this as well. Lastly, when the predictive effect is equal in both directions, the method does not pick sides. The question remains whether this method will also work in higher dimensions.

Example 5.1.3. *Three dimensions.* Now, consider the three dimensional setting from Equation 5.11. Just as in Example 4.1.1, we expect our method to report $X \leftarrow Y \leftarrow Z$. The main difference between the two and the three dimensional case is that we now need more parameters for our doubly stochastic matrix P. As it turns out, we require $(n-1)^2$ parameters for an $n \times n$ doubly stochastic matrix. If we take all entries except the bottom row and the right-most column as parameters, $\{p_{ij}\}_{i,j=1,\dots,n-1}$, we can deduce all other values. First, we can deduce the right-most entry for each of the top $n-1$ rows, as the entries in each row must sum to one, so $p_{in} = 1 - \sum_{j=1}^{n-1} p_{ij}$. Similarly, we can deduce the bottom entry for each of the left-most $n-1$ columns by $p_{nj} = 1 - \sum_{i=1}^{n-1} p_{ij}$. Lastly, we can deduce the bottom-right right entry by either using the right-most column or the bottom row by $p_{nn} = 1 - \sum_{i=1}^{n-1} p_{in} = 1 - \sum_{j=1}^{n-1} p_{nj}$. Hence, we now require 4 variables, which we denote by $\{p_{ij}\}_{i,j=1,2}$ to uniquely define the three-dimensional doubly stochastic matrix P.

Generating data according to this model and applying our method yielded the following estimates for $\hat{P}, \hat{A}$:

Closest permutation matrix by Hungarian algorithm.

5.2 Combinatorial Approaches

In this section, we will discuss approximate methods that are combinatorial in nature. The difficulty that arises when using this type of method is that the combinatorial search space of DAGs is super exponential. Nevertheless, our approaches here focus on finding a permutation matrix P and a lower triangular matrix A. Given a permutation matrix P, finding a suitable lower triangular matrix is quite simple, for example the closed form least-squares solution. Nevertheless, the total number of permutation matrices on p variables is $p!$, which also grows exponentially with respect to p. This section considers some of these combinatorial methods of different complexities.

5.2.1 Exhaustive Search

A simple and naive method is an exhaustive search over all permutation matrices $P_{perm} \in \mathcal{P}$. As mentioned before, this method is not tractable when we have a large number of variables p. Nevertheless, it is an interesting starting point from which a lot can be learned. An overview of the algorithm is given in Algorithm **??**.

Finding A given P. Given a permutation matrix P that permutes our p variables, the next task is to finding the most suitable matrix A such that $G(W)$ is a DAG with self-loops allowed. An approach is to find

$$A = {}_{A\text{lower triangular}}.$$

Results.

An interesting remark is that we do not need to do an exhaustive search of all permutation matrices to find the optimal solution, especially when the matrix W is sparse. In fact, any permutation that adheres to the topological ordering of the graph $G(W)$ would yield the same optimal solution for a large enough sample size. For example, the graph

5.2.2 Random Walk

5.2.3 Using Markov Chain Monte Carlo

As we have seen in Section, relaxing the search space from the set of permutation matrices to the set of doubly stochastic matrices created numerous issues. Although the search space is now continuous rather than combinatorial, the presence of singular or close to singular doubly stochastic matrices makes it difficult to retrieve a local optimum (A, P) such that P is "close" to a permutation matrix. This section takes one step back and tries a novel approach to find a suitable permutation matrix P.

The approach that we will investigate in this section is a so-called *Markov Chain Monte Carlo* (MCMC) method, inspired by [3].

$$q_{P,P'} = \Pr\left(s_{i+1} = P' \mid s_i = P\right) = \begin{cases} 0 & \text{if } P' \text{ is not within one move from } P \\ \frac{1}{n_P} & \text{otherwise} \end{cases} \tag{5.20}$$

Here, we define "one move" as the interchanging of two variables X_i and X_j. This is done by interchanging the ith row and the jth row or equivalently, interchanging the ith row and the jth column.

Given a permutation matrix on n variables, there are a total of $\binom{n}{2}$ possible moves. Hence, the possible number of valid moves does not depend on the permutation matrix P.

Let us now define the probability of acceptance $\alpha_{P,P'}$ as

$$\alpha_{P,P'} = \begin{cases} \min\left\{\frac{\mathcal{L}(\mathbf{X}|P')\cdot q_{P',P}}{\mathcal{L}(\mathbf{X}|P)\cdot q_{P,P'}}, 1\right\} & \text{if } \mathcal{L}(\mathbf{X} \mid P)\cdot q_{P,P'} > 0 \\ 1 & \text{otherwise} \end{cases} \tag{5.21}$$

The number of permutation matrices P' that are within one move from P is equal to the total amount of possible variables that we can swap. For any permutation matrix P, n_P is equal to the number of possible pairs given n variables, which is

$$n_P = \binom{n}{2} = \frac{n(n+1)}{2} \quad \forall P \in \mathcal{P}.$$

Now, we have that the likelihood of our data $\mathbf{X}$ given our lower triangular matrix A and our permutation matrix P is

$$\mathcal{L}(\mathbf{X}|A, P) = \frac{1}{2(T-1)}\left\|\mathbf{X}_{[1:T]} - \mathbf{X}_{[0:T-1]}P^TAP\right\|_F^2.$$

However, for a given permutation matrix P, we do not know A. Luckily, the likelihood is only quadratic with respect to A, so $\mathcal{L}(\mathbf{X}|A, P)$ can be easily be optimized with respect to A via for example gradient descent. The gradient of $\mathcal{L}(\mathbf{X}|A, P)$ with respect to A is

$$\nabla_A\mathcal{L}(\mathbf{X}|A, P) = -\frac{1}{(T-1)}P\mathbf{X}_{[0:T-1]}^T\left\|\mathbf{X}_{[1:T]} - \mathbf{X}_{[0:T-1]}P^TAP\right\|_F P^T.$$

Let

$$\mathcal{L}(\mathbf{X}|P) = \arg_A \min \mathcal{L}(\mathbf{X}|A, P).$$

Then, we get that the probability of acceptance $\alpha_{P,P'}$ is equal to

$$\alpha_{P,P'} = \begin{cases} \min\left\{\frac{\mathcal{L}(\mathbf{X}|P')}{\mathcal{L}(\mathbf{X}|P)}, 1\right\} & \text{if } \mathcal{L}(\mathbf{X} \mid P) > 0 \\ 1 & \text{otherwise} \end{cases} \tag{5.22}$$

Now, the transition matrix of the *new* Markov chain $\{r_{P,P'}\}$ is

$$r_{P,P'} = \Pr\left(s_{i+1} = P' \mid s_i = P\right) = \begin{cases} \alpha_{P,P'} q_{P,P'} & \text{if } P' \neq P \\ 1 - \sum_{P' \neq P} \alpha_{P,P'} q_{P,P'} & \text{if } P^{'} = P \end{cases} \tag{5.23}$$

However, we need to find a "good" initialization P_0 from where we start sampling according yo our chain $\{r_{P,P'}\}$.

Initial state of $\{r_{P,P'}\}$. We can use the *ordinary least squares* estimate for W.

However, $\hat{W}$ need not be the WAM of a DAG. We will iteratively set the smallest (in absolute value) entry of $\hat{W}$ to zero until $\hat{W}$ is the WAM of a DAG.

From this initial state, we can already expect to be close to a good guess, especially when we have enough samples, i.e, T is sufficiently large.

5.2.4 Orthogonal Matching Pursuit

In its regular form, the Orthogonal Matching Pursuit can be written in two forms:

$$\arg\min_{w \in \mathbb{R}^p} \|y - Xw\|_2^2 \text{ subject to } ||w||_0 \leq k.$$

Alternatively, we can use the *dual* variant of it, where we are looking for the smallest number of entries that we can add such that our residual error is below some threshold.

However, we see that the current formulation of OMP only allows for one variable $y \in \mathbb{R}^T$. One approach is to do the OMP algorithm p times, one for each variable. In our setting, we have for one time series that

$$y = X_{i,2:T} \in \mathbb{R}^{T-1}, \qquad X = X_{\cdot,1:T-1} \in \mathbb{R}^{p \times T-1}, \qquad w = w_i \in \mathbb{R}^p.$$

Now, we can do this simply for all p time series, thereby getting the p columns w_i. However, we do not know beforehand how much non-zero entries we require per variable. One variable can depend on all p variables, whereas another variable depends only on itself. Therefore, fixing the number of nonzero coefficients for each variable separately will not result in a valid solution.

Rewriting the conventional OMP to matrix form. Another approach is to modify the algorithm to be applied to multiple variables at the same time. Interestingly, this can be done easily by some small modifications. Recall that the conventional notation of OMP is

$$\arg\min_{w \in \mathbb{R}^p} \|y - Xw\|_2^2 \text{ subject to } ||w||_0 \leq k, \tag{5.24}$$

whereas our desired matrix notation is

$$\arg\min_{W \in \mathbb{R}^{p \times p}} \left\|\mathbf{X}_{[2:T]} - \mathbf{X}_{[1:T-1]} W\right\|_F^2 \text{ subject to } ||W||_0 \leq k. \tag{5.25}$$

Unfortunately, the conventional notation in the literature is of the form of Equation 5.24. Nevertheless, we can modify the algorithms to accommodate problems of the form of Equation 5.25. However, an easier approach is to rewrite our data matrix $\mathbf{X}$ and matrix W into the form of Equation 5.24. Then, we know that the vectorized dimensions will be

$$\left\|y' - X'w'\right\|_2^2,$$

with

$$y' \in \mathbb{R}^{p \cdot (T-1)}, \qquad X' \in \mathbb{R}^{p^2 \times p \cdot (T-1)}, \qquad w' \in \mathbb{R}^{p^2}.$$

Our approach is as follows: we will first vertically stack our $y_i \in \mathbb{R}^{T-1}$ to get a column vector $y \in \mathbb{R}^{p\cdot(T-1)}$. In mathematical notation,

$$\begin{aligned} y &= vec(y_1, y_2, \dots, y_p) \\ &= (y_1, y_2, \dots, y_p)^T \\ &= (X_{2,1}, X_{3,1}, \dots, X_{T,1}, \dots, X_{2,p}, X_{3,p}, \dots, X_{T-1,p})^T . \end{aligned}$$

Secondly, we will vectorize our matrix W by vertically stacking the columns $W_{\cdot,i}$. Therefore, we have

$$\begin{aligned} w' &= vec(W) \\ &= (w_1, \dots, w_p) \\ &= (w_{11}, \dots, w_{1p}, \dots, w_{p1}, \dots w_{pp})^T . \end{aligned}$$

Thirdly, we need to define our matrix $X' \in \mathbb{R}^{p^2\times p\cdot(T-1)}$, such that

$$X_{[1:T-1]}W = X'w'.$$

We achieve this by repeating the matrix $X_{[0:T-1]} \in \mathbb{R}^{T-1}$ a total of p times and introducing some matrix blocks containing only zeros.

As an example, consider estimating the ith column of our matrix W. We want the coefficients corresponding to $i-1$ first columns to be multiplied by zero. Therefore, we we simply prepend a block of size $(i-1)\times T-1$ with zeros before the matrix $\mathbf{X}_{[1:T-1]}$. Similarly, we postpend a block of size $(p-i)\times T-1$ containing only zeros at the end of this block. For the i column, this gives us the equality

$$\left(\underbrace{\mathbf{0}, \dots, \mathbf{0},}_{(i-1)\cdot p} \ \mathbf{X}_{[0:T-1]}, \ \underbrace{\mathbf{0}, \dots, \mathbf{0}}_{(p-i)\cdot p} \right) w' = \mathbf{X}_{[1:T-1]} W_i.$$

This way, coefficients of W not corresponding to the ith column will be multiplied by zero, and the coefficients of the ith column will be multiplied by the data matrix $\mathbf{X}_{[1:T-1]}$. In the end, we get a block of size $p\times T-1$ that we use to estimate the coefficients of the ith column.

We create such a $p^2 \times T-1$ block for all p variables, and we stack these vertically, resulting in a $p^2 \times p\cdot(T-1)$ data matrix. In matrix notation, we get

$$X' = \begin{pmatrix} \mathbf{X}_{[1:T-1]} & \mathbf{O} & \cdots & \mathbf{O} & \mathbf{O} \\ \mathbf{O} & \mathbf{X}_{[1:T-1]} & \cdots & \mathbf{O} & \mathbf{O} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \mathbf{O} & \mathbf{O} & \cdots & \mathbf{X}_{[1:T-1]} & \mathbf{O} \\ \mathbf{O} & \mathbf{O} & \cdots & \mathbf{O} & \mathbf{X}_{[1:T-1]} \end{pmatrix} \in \mathbb{R}^{p^2\times p\cdot(T-1)}.$$

A more concise mathematical notation for this using the Kronecker product is

$$X' = I_p \otimes \mathbf{X}_{[1:T-1]} \in \mathbb{R}^{p^2\times T-1}.$$

To conclude, we see that we have rewritten our matrix sparsity problem to a vector sparsity problem:

$$\mathbf{X}_{[2:T]} - \mathbf{X}_{[1:T-1]}W \iff y' - X'w'.$$

5.2.5 Ordinary Least Squares

5.2.6 LASSO

5.3 Miscellaneous Approaches

5.3.1 Learning P^TAP with "moving" A

To quote [], "looking for an optimal T given a fixed P makes sense, but changing P for a fixed T does not." The entry T_{ij} originally capured the effect of $X_{P(j)}$ on $X_{P(i)}$. However, after changing the permutation matrix from P to P', T_{ij} captured the effect of $X_{P'(j)'}$ on $X_{P'(i)}$. These effects can be totally different, hence the previous estimate T_{ij} is now meaningless. However, an approach would be to also shift this matrix T_{ij} along. We are looking for a shift that after changing from P to P', the entry T_{ij} is shifted to $T_{i'j'}$, such that the estimate still remains meaningful for the specific i' and $'j'$, and that T_{ij} now contains the current estimate of $X_{P'(j)}$ on $X_{P'(i)}$. However, for computing the current cost, we will only use the lower triangular part of T, to enforce acyclity.

Chapter 6

Evaluation

This chapter evaluates the methods and the model.

Vivamus vehicula leo a justo. Quisque nec augue. Morbi mauris wisi, aliquet vitae, dignissim eget, sollicitudin molestie, ligula. In dictum enim sit amet risus. Curabitur vitae velit eu diam rhoncus hendrerit. Vivamus ut elit. Praesent mattis ipsum quis turpis. Curabitur rhoncus neque eu dui. Etiam vitae magna. Nam ullamcorper. Praesent interdum bibendum magna. Quisque auctor aliquam dolor. Morbi eu lorem et est porttitor fermentum. Nunc egestas arcu at tortor varius viverra. Fusce eu nulla ut nulla interdum consectetuer. Vestibulum gravida. Morbi mattis libero sed est.

6.1 Performance Criteria

True Positive Rate.

True Negative Rate.

Precision.

Recall.

Precision-Recall Curve.

Area Under Precision-Recall Curve.

Structural Hamming Distance.

6.2 Datasets

6.2.1 Simulated Datasets

Data can be generated in multiple ways: - Hénon map. - Lorenz system. - $\mathbf{X}_t = A\mathbf{X}_{t-1} + I\epsilon_t$

6.2.2 Real Datasets

- https://www.bnlearn.com/bnrepository/ - https://webdav.tuebingen.mpg.de/cause-effect/

Chapter 7

Conclusions

This is the conclusion.

Vivamus vehicula leo a justo. Quisque nec augue. Morbi mauris wisi, aliquet vitae, dignissim eget, sollicitudin molestie, ligula. In dictum enim sit amet risus. Curabitur vitae velit eu diam rhoncus hendrerit. Vivamus ut elit. Praesent mattis ipsum quis turpis. Curabitur rhoncus neque eu dui. Etiam vitae magna. Nam ullamcorper. Praesent interdum bibendum magna. Quisque auctor aliquam dolor. Morbi eu lorem et est porttitor fermentum. Nunc egestas arcu at tortor varius viverra. Fusce eu nulla ut nulla interdum consectetuer. Vestibulum gravida. Morbi mattis libero sed est.

Bibliography

[1] Scipy api reference for `optimize.minimize`. 10

[2] Scipy api reference for the `L-BFGS-B` optimization method. 10

[3] Rui Castro and Robert Nowak. Likelihood based hierarchical clustering and network topology identification. In Anand Rangarajan, Mário Figueiredo, and Josiane Zerubia, editors, *Energy Minimization Methods in Computer Vision and Pattern Recognition*, pages 113–129, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg. 31

[4] Magali Champion, Victor Picheny, and Matthieu Vignes. Inferring large graphs using ℓ_1-penalized likelihood. *Statistics and Computing*, 28(4):905–921, Jul 2018. 12

[5] Aramayis Dallakyan and Mohsen Pourahmadi. Learning bayesian networks through birkhoff polytope: A relaxation method, 2021. 11

[6] Hidde De Jong. Modeling and simulation of genetic regulatory systems: A literature review. *JOURNAL OF COMPUTATIONAL BIOLOGY*, 9:67–103, 2002. 1

[7] R. W. Robinson. Counting unlabeled acyclic digraphs. In Charles H. C. Little, editor, *Combinatorial Mathematics V*, pages 28–43, Berlin, Heidelberg, 1977. Springer Berlin Heidelberg. 14

[8] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020. 10

[9] Xun Zheng, Bryon Aragam, Pradeep Ravikumar, and Eric P. Xing. Dags with no tears: Continuous optimization for structure learning, 2018. 9

[10] Xun Zheng, Bryon Aragam, Pradeep Ravikumar, and Eric P. Xing. Dags with no tears: Continuous optimization for structure learning. presented at neurips 2018, november 2018. 2018. v, v, 1

[11] Ciyou Zhu, Richard H. Byrd, Peihuang Lu, and Jorge Nocedal. Algorithm 778: L-bfgs-b: Fortran subroutines for large-scale bound-constrained optimization. *ACM Trans. Math. Softw.*, 23(4):550–560, December 1997. 10

Appendix A

Appendix 1

Appendix B

List of datasets

List of datasets.

Vivamus vehicula leo a justo. Quisque nec augue. Morbi mauris wisi, aliquet vitae, dignissim eget, sollicitudin molestie, ligula. In dictum enim sit amet risus. Curabitur vitae velit eu diam rhoncus hendrerit. Vivamus ut elit. Praesent mattis ipsum quis turpis. Curabitur rhoncus neque eu dui. Etiam vitae magna. Nam ullamcorper. Praesent interdum bibendum magna. Quisque auctor aliquam dolor. Morbi eu lorem et est porttitor fermentum. Nunc egestas arcu at tortor varius viverra. Fusce eu nulla ut nulla interdum consectetuer. Vestibulum gravida. Morbi mattis libero sed est.

Appendix C

Additional tables

The tables.

Vivamus vehicula leo a justo. Quisque nec augue. Morbi mauris wisi, aliquet vitae, dignissim eget, sollicitudin molestie, ligula. In dictum enim sit amet risus. Curabitur vitae velit eu diam rhoncus hendrerit. Vivamus ut elit. Praesent mattis ipsum quis turpis. Curabitur rhoncus neque eu dui. Etiam vitae magna. Nam ullamcorper. Praesent interdum bibendum magna. Quisque auctor aliquam dolor. Morbi eu lorem et est porttitor fermentum. Nunc egestas arcu at tortor varius viverra. Fusce eu nulla ut nulla interdum consectetuer. Vestibulum gravida. Morbi mattis libero sed est.