Structure Learning in High-Dimensional Time Series Data

de Quincey, Martin G.

Award date:
2022

EINDHOVEN UNIVERSITY OF TECHNOLOGY

Department of Mathematics and Computer Science
Statistics Group

Structure Learning in High-Dimensional Time Series Data

Master Thesis

Martin de Quincey

Supervisors:
dr. Rui Castro
dr. Alex Mey

Assessment Committee Members:
dr. Rui Castro
dr. Alex Mey
dr. Jacques Resing

version 1.2

Eindhoven, July 2022

Abstract

Living in the age of *big data*, we have an abundance of data available to help us perform complex tasks such as image recognition, object detection, and autonomous driving. Furthermore, correlations and dependencies between different variables in the data allow us to, for example, make predictions such as weather forecasts. However, in many applications, it is of greater interest to investigate how the variables in our data influence each other, for examples, how a prescription drug influences the well-being of a patient.

This master's thesis explores this interesting domain of structure learning. Most notably, we are interested in a time series setting, where measurements across successive time steps are heavily dependent. In today's world with an increasing abundance of available data, we want to understand how the values of one variable influence the future values of another variable, thus learning the relations between the variables and the structure of our data. For this, we assume that the structure can be characterized by a graphical model, where we additionally assume acyclicity of the graphical model to reduce the search space. We are predominantly interested in the high-dimensional setting, where our time series data consist of multiple dozens of variables.

We propose several methods to learn the structure of time series data, which can be categorized into three types of approaches: permutation-based, continuous, and iterative. For each type of approach, we propose several algorithms and provide numerous examples to explain their inner workings. Furthermore, we will carefully investigate these approaches to highlight and explain their strengths and weaknesses. We conduct several experiments based on simulations and real-life data to assess the performance of our methods against the state-of-the-art. Our results indicate that our methods achieve similar, if not better, performance. Permutation-based approaches seem preferred for low-dimensional time series, and iterative approaches seem preferred for high-dimensional time series.

Preface

This master's thesis contains the academic endeavors of the graduation project done by Martin de Quincey from September 2021 until July 2022. This thesis was written to complete two master's studies at the Eindhoven University of Technology. This thesis marks the completion of both the master's program "Computer Science and Engineering" with a specialization in "Data Science in Engineering", as well as the master's program"Industrial Applied Mathematics".

First, I would like to acknowledge my gratitude towards my supervisor dr. Rui Castro for his extensive guidance and assistance throughout this final project. Rui has helped greatly in shaping the project from the very beginning, and providing different perspectives and directions throughout the final project. Whenever hurdles arose on this journey, Rui helped me in either getting over these hurdles or circumventing them all together. His honest opinion on my efforts has been greatly appreciated. Furthermore, his feedback on the writing and structure of this thesis greatly helped in shaping and rounding the story that I wanted to convey.

Furthermore, I would like to acknowledge my appreciation to dr. Alex Mey for his valuable input into the project as well. During our weekly meetings, his feedback and insightful perspectives have been of great value to me throughout this final project. Without his support, it would have been much more difficult for me to complete my thesis.

Additionally, I would like to thank dr. Jacques Resing for being part of the assessment committee and being a critical reader of this thesis.

Lastly, I would also like to thank my colleagues, friends and family for keeping me motivated. They provided a pleasant distraction next to my academic endeavors. Our social interactions provided a nice balance between my academic studies and social life, not just during my graduation project, but throughout all six years that I have spent here at the Eindhoven University of Technology.

As a final remark, all the code related to this project has been uploaded to GitHub and is accessible under the GNU General Public License v3.0. This repository contains, among others, all algorithms, examples, datasets, and experiments conducted throughout this final project. The link to the GitHub repository is `https://github.com/Martindeq98/final-project`. Feel free to contact me through this link as well if you have questions, improvements, comments, etc.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Living in the age of *big data*, we have an abundance of data available to help us perform complex tasks such as image recognition, object detection, and autonomous driving. These enormous amounts of data are also connected. Correlations and dependencies between different variables in the data allow us, for example, to make predictions such as weather forecasts. However, in many applications, it is of greater interest to investigate how the variables in our data influence each other. This is exactly what this master's thesis aims to investigate. In this master's thesis, we will be exploring the interesting domain of *structure learning*. In today's world with an increasing abundance of available data, we wish to understand how the variables in our data influence each other, thereby learning the relations between the variables and the structure in our data.

Furthermore, the learned structure of our collected data must be simple to visualize. The goal of structure learning is to gain a deeper understanding, which is only possible when the structure of the data can be easily visualized. These structures must be easy to interpret, for else they have no added value. For this reason, we will focus on *graphical models*. A graphical model represents structure by means of a graph encoding relations between variables. It is a statistical model that explains how the data is generated, thereby also taking into account the relations and structure of the data. Such a graphical model can easily be parameterized by a graph or network of the data. In such a way, the relations between variables in the data are clearly visualized.

Running Example. Let us consider the classical "sprinkler" example from Pearl [55] to explain the notion of structure learning and how we can represent the learned structure as a network. We will use this running example to explain numerous concepts regarding structure learning throughout this thesis. In this sprinkler example, we have collected data regarding the slipperiness of our pavement. We have simultaneously measured the following five variables:

- X_1: The seasonality at the moment.
- X_2: Whether the sprinkler installation is turned on.
- X_3: Whether it rains.
- X_4: How wet the pavement is.
- X_5: How slippery the pavement is.

The objective is to learn the structure between these five variables. We want to gain a deeper understanding of how the variables influence each other. This allows us to answer questions such as "How will the seasonality affect the slipperiness of my pavement?" and "Does the seasonality influence the rainfall?"

In this scenario, we can use our domain knowledge to derive the relationships between these variables. In general, we know that the season influences whether the sprinkler installation is turned on; the sprinklers tend to be used more often in the summer than in the winter. We can

capture this directed relationship by using the notation $X_1 \to X_2$. Furthermore, we know that it rains more often in the autumn than in the summer, and therefore we expect the seasonality to also influence the rainfall ($X_1 \to X_3$). Note that both these relationships are directed; turning on the sprinklers or having a rainy day will not influence which season we are currently in. In turn, we know that when the sprinklers are on or when it rains, then the pavement will become wet ($X_2 \to X_4$ and $X_3 \to X_4$), which in turn makes the pavement slippery ($X_4 \to X_5$).

We have learned the *structure* of these five variables X_i using domain knowledge. We can visualize these relations using nodes for the variables and arcs between these nodes to represent the directed relationships. This results in a graphical network, which has been visualized in Figure 1.1.

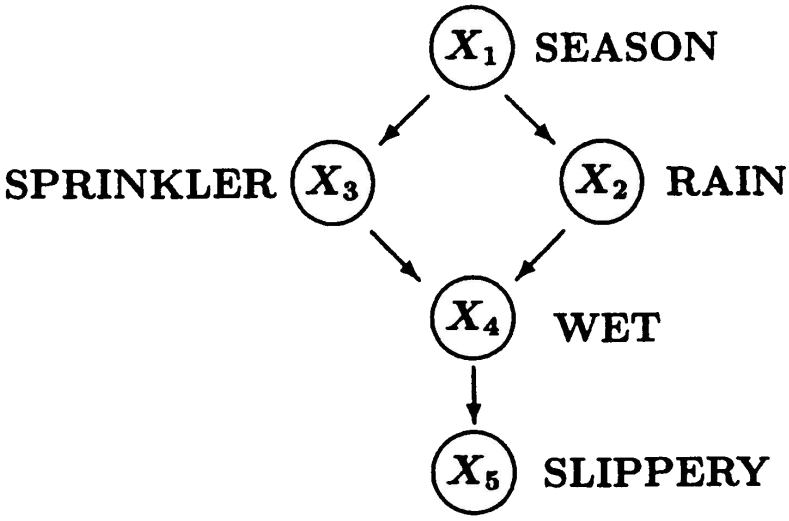


Figure 1.1: The graphical model corresponding to the sprinkler example, taken from [55].

Note that this graphical network only visualizes the *direct* relationships between variables. We know that the seasonality will affect the wetness of the pavement. However, this relationship is indirect through the sprinkler and the rainfall.

Unlike the example sketched above, we will not assume any domain knowledge throughout this thesis. Instead of domain knowledge, we only have observational data at our disposal to infer such a structure. For the sprinkler example, observational data could be collected twice a day for a full year, yielding 730 measurement tuples $(X_1, X_2, X_3, X_4, X_5)_k$, $k = 1, \ldots, 730$. The objective would then be to recover the structure of the graphical model shown in Figure 1.1 using these measurements.

Note that using only observational data creates numerous difficulties, such as not having the right amount of data or not having access to the desired variables. Recalling the sprinkler example, if we had only collected data during the winter, we probably would not have seen the relationships regarding the sprinkler variable X_3. Furthermore, if we had forgotten to measure a variable such as the rain variable X_2, we would have failed to capture the full structure regarding the slipperiness of our pavement.

Having sketched the main objective using the sprinkler example, let us consider some applications where structure learning is important.

Applications. Structure learning is an important concept with many applications, predominantly with regard to complex systems. In a world where systems are becoming ever more complex, it is important to have a suitable graphical model that captures its behavior. Although domain knowledge is generally available for man made machines, they can still be so complex that they defy human intuition. This increased level of complexity arises from having machines with an almost uncountable number of different components. Furthermore, all these components interact

with many other components, and each component will respond differently to such an interaction. Therefore, it is difficult to determine the relationships between all these components using only domain knowledge. If we were able to infer the structure of all these components using observational data, we would gain a deeper understanding of how the machine works, how it interacts, and most importantly how to keep it operational.

A group of systems so complex they defy our intuition are gene regulatory systems. Modeling such systems allows us to understand which genes are expressed in which part of the organism and to which extent. Most of these gene regulatory systems involve many interlocking positive and negative feedback loops, making these systems so complex that they defy human intuition. Furthermore, domain knowledge in such gene regulatory systems is generally not available, yet observational data is quite easily obtainable. More information on such complex networks in gene regulatory systems can be found in [37]. Having a graphical model that represents the structure of such a complex regulatory system can provide crucial information to biologists or pharmacists. Biologists can better understand what, for example, causes an illness and consequently improve the medical care a patient receives. Pharmacists in turn can create more appropriate drugs that improve the medical treatment a patient receives.

In addition to complex biological systems, even man-made systems can become so complex that they will defy human intuition. Based solely on human knowledge, 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 a single person cannot comprehend their complete behavior. Examples of such complex machines are abundant throughout the tech industry, such as the EUV lithography systems manufactured by ASML [32] and modern cars such as Tesla models. With an abundance of sensors and components in such a Tesla car, learning how components of the autopilot interact in dangerous scenarios is of vital importance for a company such as Tesla. This structure also provides insight into how the Tesla autopilot makes decisions based on the information it receives.

Therefore, we see that there is an opportunity 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 are 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 down, it might be so complex that finding the problem requires disassembling the machine completely, which is an expensive and time-consuming process. The placement of sensors in machines can help diagnose which components of the machine exhibit anomalous behavior. However, the fact that a sensor records unexpected 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. Learning the structure of the interacting components can then provide crucial information. First, whenever the machine breaks, we can look at which sensors produce anomalous data. Indeed, the most logical approach is still to first inspect the components around these sensors. However, if the problem is not located 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 useful for automated *root cause analysis* in complex systems.

As an example, suppose that we have a machine consisting of seven components, each accompanied by a sensor labeled a through g. Furthermore, suppose we have learned the structure shown in Figure 1.2. The machine malfunctions and needs to be shut down and based on sensor data we have seen that variables c, g, and j produce anomalous data, indicating that something might have gone wrong there.

We could completely break down the machine and investigate all the components where the sensors have recorded unusual data. But what if no problem is found near any of these sensors, what should be the next step? Should we check all seven components and in what order? Here

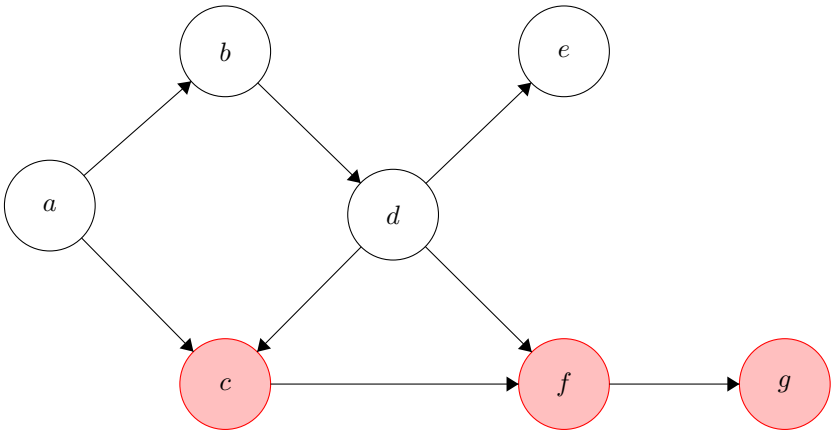


Figure 1.2: Toy example of seven variables a through g with their corresponding structure. The nodes representing the variables c, f, and g are colored red, indicating that their sensors record abnormal behavior.

we can take advantage of the learned structure shown in Figure 1.2. Based on the dependencies between the components, it is reasonable to assume that the malfunction occurred in component c and that this caused the anomalous data in the subsequent components. The learned structure allows for a concrete pinpoint of the likely root cause of the malfunction.

Furthermore, should there still be a malfunction, we can easily find the next likely component by following the structure of the machine. We can go "up" the chain of dependencies by verifying components a and d, or we can go down "down" the chain of dependencies by verifying the components f and g. Based on the structure, we can also deduce that it is unlikely that component e was the actual root cause; something we could not deduce without a learned structure. Therefore, we see that this structure allows us to perform a simple yet sensible *root cause analysis* of our broken machine. More complex methods exist to locate root causes, but many rely on learning the network or structure of the complex system beforehand [15, 41, 54].

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

True causality versus predictive causality. Throughout this thesis, we will make a clear distinction between *predictive* causal relations and regular causal relations. True causality, as for example introduced by the framework of Pearl [55], is impossible to detect from observational data alone. According to the framework of Pearl, causality can be discovered using *interventions*. Suppose that X causes Y. This means that if we intervene on Y, for example by fixing its value, then the value of X does not change. However, if we intervene on X, then the value of Y will change. Using interventions, we can discover that changing the value of X will change the value of Y, but changing the value of Y will not change the value of X. Under this interventionist position of Pearl, we say that X has a causal influence on Y. However, not all agree on this notion of causality [10]. In fact, these disagreements remain to this day, and the accompanying theory remains fiercely debated, as mentioned in [55].

Therefore, we have decided to steer away from the notion of true causality, and primarily focus on predictive causality. Instead of looking for relations where X causes Y, we are satisfied when we have found relations where X *is useful in predicting* the future of Y. This notion is similar to *Granger causality*, where we test whether one time series X is useful in forecasting another time

series Y [29]. More formally, Granger causality means that the past of a time series X provides a significant amount of additional information to predict the time series Y. The notion of Granger causality was introduced in 1969 and continued on the work of Wiener [84] and remains a common notion of causality to this day. Using the notion of usefulness in prediction allows us to safely steer away from the philosophical questions and complex notation that accompany the field of causality.

To illustrate this example, consider a simple graphical model based on two variables, the temperature T and the altitude Y. Based on our understanding of geography and physics, we know that the direction of true causation should be $Y \to T$. If we ascend a mountain, the temperature T will decrease as the atmosphere becomes thinner and thinner. On the other hand, if the temperature T increases in the summer, the altitude will not increase, so $T \not\to Y$. Even in this two-dimensional example, we see the problems that arise from the notion of "true causation". In fact, it is not altitude that causes the temperature to decrease; it is a more complex scientific phenomenon regarding atmospheric pressure. Therefore, reasoning about true causation is difficult, especially when not all the important variables are available. Furthermore, purely based on observational data, how can we learn the true causal direction of the relation between the temperature T and the altitude Y? To safely steer away from these questions, we are satisfied when we have discovered the relation that the altitude Y is useful in predicting the temperature T.

Influence of Time The notion of *time* is crucial in this thesis. As the title suggests, we are primarily investigating time series. Sensor data from complex systems, for example, are time series, as we are recording several measurements over time. Furthermore, the sprinkler example was also time series data, as we said we would measure the five variables twice a day for a full year. In these time series data, past values of a variable can provide useful information for the future. As in the sprinkler example, if it was raining today, we can expect that the sprinklers will not be turned on soon, as the grass has already been watered. We can also deduce that if the sprinkler is on today it most likely will not be on tomorrow, as the grass has already been watered. Therefore, the present value of one node can be useful in predicting the future value of itself or another node. This notion of time is also helpful in determining the direction of the relationships. We mentioned that it is difficult to determine the direction of influence between variables purely based on observational data. However, if there is a time delay between the action and the result, the direction of the arc is easier to determine. As in the sprinkler example, we know that there is an arc from the sprinkler to wetness because if we *first* turn on the sprinkler, *then* the pavement will get wet. In this time series setting, all directions of the arcs will go from the past to the present; as we know, the present cannot influence the past.

Cycles Furthermore, we exclude the existence of cycles or feedback loops in the graphical model for several reasons. Firstly, we hope to recover a structure that is simple to understand. Cycles in a structure make them more difficult to understand as it becomes more difficult to see which variables influence which variables. Furthermore, acyclic structures are less cluttered with overlapping arcs, making the visualization easier to interpret. Additionally, by enforcing the graph to be acyclic, the total number of possible structures shrinks dramatically. For instance, if we require acyclicity and discover the arc from seasonality (X_1) to rainfall (X_2), we can immediately disregard all structures that contain an arc from X_2 to X_1, thereby greatly reducing the search space. Especially when there are tens or hundreds of variables, it is crucial to have a structure that is simple to interpret, and acyclicity enhances the interpretability of such a structure.

Second, even when such a cycle would occur in the predictive causal network, we can still be interested in the most suitable direction. For example, the value of gold might influence the value of silver and the other way around, so such cycles may appear in reality. Nevertheless, when we exclude the existence of a cycle between two variables, we can detect the direction corresponding to the *strongest* relationship, which will also yield interesting insights.

Lastly, the existence of cycles means that there are infinitely long paths in the structure, which greatly complicate inference problems in these networks. For example, the joint distribution that

adheres to this cyclic structure may not exist. This is a known technical issue with such graphical models, which will be discussed in further detail in Chapter 2. Having no joint distribution adhering to such a cyclic structure is just one of the several problems, which are mentioned in more detail in [78].

Research Objective The research objective throughout this thesis is to develop data-driven approaches that learn the structure of complex systems given some set of observational data. Such a structure is often displayed as a *predictive causal network* consisting of nodes and arcs, as in Figure 1.1. These arcs represent predictive causal relations rather than true causal relations. We are satisfied when we have found relations that imply "usefulness in prediction", rather than true causality. Our approaches are mainly focused on time series, as the direction of the arcs is more intuitive to determine. Furthermore, the inferred structure should be free of cycles or feedback loops, such that if there is a path $X \to ... \to Y$ then there is no path $Y \to ... \to X$.

Outline. This thesis is structured as follows. In Chapter 2, we will continue with formalizing the notion of structure learning with a mathematical framework. Furthermore, we will define more concretely the exact objective we are trying to achieve. In Chapter 3, we will discuss several existing methods for structure learning. The chosen methods use varying interesting techniques to discover the structure of the data and contain state-of-the-art methods that have recently been developed, as well as more classical approaches. 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 the subsequent three chapters, we will introduce methods that we have developed and were inspired by the methods discussed in Chapter 3. Chapter 4 will introduce methods based on explicitly searching for a suitable *structure* of the variables. Chapter 5 will developed methods that rely on continuous optimization approaches under acyclicity constraints. In Chapter 6, we will discuss methods that iteratively construct the structure, arc by arc. The methods covered in these three chapters will be evaluated and compared on both simulated and real-life data in Chapter 7. Lastly, we conclude the thesis by summarizing our contributions, discussing the limitations of our work, and providing directions for future work in Chapter 8. A schematic overview of the upcoming chapters is given in Figure 1.3.

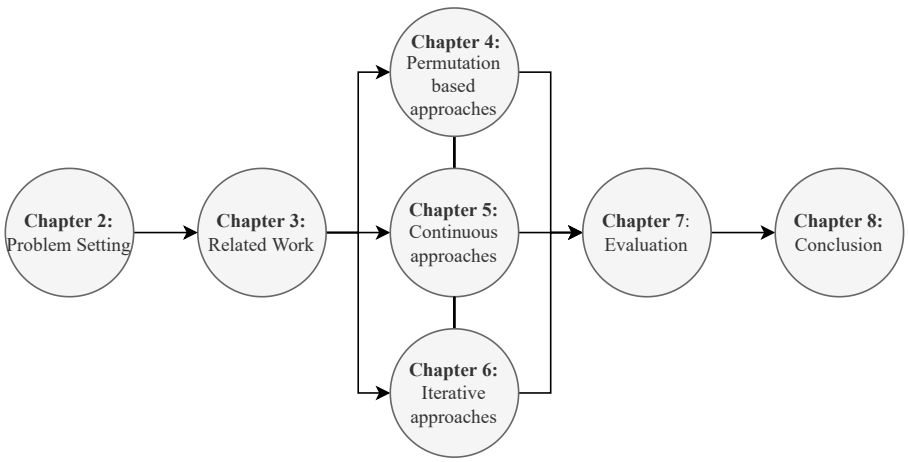


Figure 1.3: Schematic overview of the upcoming chapters in this thesis.

Chapter 2

Problem Setting

In Chapter 1, we have introduced the concepts and intuition behind structure learning. Having provided some motivating examples and applications, let us now formalize our research objective in greater detail. To recap where we left off, the goal is to learn the structure of an observational dataset and recover directed relationships between the variables in our dataset. Throughout this thesis, we will define our given dataset as a data matrix of the form

$$\mathbf{X} \in \mathbb{R}^{T \times p}. \tag{2.1}$$

This dataset contains T measurements of our p variables $X_1, \ldots, X_p$. Let $X_{\cdot,j}, j = 1, \ldots, p$ represent the jth column of $\mathbf{X}$, corresponding to the T measurements of the jth variable. Analogously, $X_{t,\cdot}$ represents the tth row of $\mathbf{X}$, corresponding to the p measurements of the variables at time step t, where $t = 1, \ldots, T$.

Given this data matrix $\mathbf{X}$, we now want to learn the *structure*, which can be captured by the relationships between the p different variables. From a probabilistic perspective, we model $\mathbf{X}$ as a realization of a random matrix with joint distribution

$$\mathbb{P}(\mathbf{X}) = \mathbb{P}\left(X_{1,1} = x_{1,1}, \ldots, X_{1,p} = x_{1,p}, \ldots X_{T,1} = x_{T,1}, \ldots, X_{T,p} = x_{T,p}\right), \tag{2.2}$$

where x_{ij} represents the ith measurement of the jth variable. Learning the structure then corresponds to recovering the joint distribution $\mathbb{P}(\mathbf{X})$ using solely the observed data matrix $\mathbf{X}$.

However, this joint probability does not directly translate to a structure or network regarding our p variables. For this, we require some modeling assumptions, for example, that our data has been generated by a graphical model. Let us first consider a graphical model that does not take influences between different time steps into account.

Time-independent graphical models. Disregarding the notion of time first, a common assumption is to *factorize* the probability distribution. If we assume that the T measurements of the p variables are independent, then the joint probability function in Equation 2.2 factorizes nicely to

$$\mathbb{P}\left(\mathbf{X}\right) = \prod_{t=1}^{T} \mathbb{P}\left(X_{t,1}, X_{t,2}, \ldots, X_{t,p}\right). \tag{2.3}$$

As we assume the measurements are independent over time, another conventional assumption is that the measurements X_t are also identically distributed for all $t = 1, \ldots T$. Therefore, let us consider $\mathbb{P}\left(X_1, \ldots, X_p\right)$, where $(X_1, \ldots, X_p)$ corresponds to a p-dimensional vector with the time index t omitted. It is reasonable to assume that the value of any variable X_i does not depend on all other variables, but merely on a subset of the other variables. For example, the rainfall is only influenced by the season, not by the sprinkler installation or the wetness of the pavement. Therefore, let us define the *parent* set of X_j, abbreviated as $\mathrm{Pa}\left(X_j\right) \subseteq \{X_1, \ldots, X_p\} \setminus X_j$, as the

set of variables that influence the value for X_j. The probability distribution as written in Equation 2.3 then factorizes even further to

$$\mathbb{P}(X_1, X_2, \ldots, X_p) = \prod_{j=1}^{p} \mathbb{P}(X_j | \mathrm{Pa}(X_j)). \tag{2.4}$$

Such a factorization of time-independent probability distribution $\mathbb{P}(X_1, \ldots, X_p)$ is further clarified using the sprinkler example in Example 2.1.

Example 2.1 Factorization of the time-independent sprinkler example.

To see a concrete example, let us consider the sprinkler example visualized in Figure 1.1. We see that the parent set of a node X_j corresponds to all the variables X_i such that there is an arc from X_i to X_j We have that the probability distribution $P(X_1, X_2, X_3, X_4, X_5)$ can be factorized as

$$\mathbb{P}(X_1, X_2, X_3, X_4, X_5) = \mathbb{P}(X_1)\mathbb{P}(X_2|X_1)\mathbb{P}(X_3|X_1)\mathbb{P}(X_4|X_2, X_3)\mathbb{P}(X_5|X_4). \tag{2.5}$$

Intuitively, this means that for example, variable X_5, representing the slipperiness of the pavement is independent of all other variables, given X_4, representing the wetness of the pavement.

We see that this assumption of factorizing the joint probability distribution allows for a clear graphical representation. We can draw a graph $G = (V, A)$, where the p variables are represented by the nodes V. Furthermore, the dependence relations are given by the set of arcs $A = (V \times V)$, where there is an arc from node X_i to X_j if and only if X_i is a parent of X_j. This type of graphical model is called a *Bayesian network*. Note that the network drawn in Figure 1.1 exactly corresponds to the graphical model associated with the factorization of Equation 2.5. This factorization also shows why the exclusion of cycles is natural in these Bayesian networks. If the graphical model contained cycles, then there would not be such a closed-form solution to the factorized joint probability distribution as in Equation 2.5. Another important modeling assumption to keep in mind is that these Bayesian networks do not take into account the notion of time. They represent instantaneous relationships.

Now that we have explained how a factorization in the general form of Equation 2.4 can be visualized in a graphical model, we need some more modeling assumptions. One assumption is that we only need the value of the parents of X_i to model X_i. In addition to the structure, we need to characterize the conditional distribution of X_i given its parent set $\mathrm{Pa}(X_i)$. As the set of all possible conditional distributions is too broad, a simple yet common model is to assume a *linear* relation between X_i and its parent set, yielding a linear graphical model. We will introduce such a linear graphical model in the next paragraph.

A linear graphical model for Bayesian networks. A common method to model such a Bayesian network is to assume that the relationship between variables is *linear*. In other words, the variable X_j is modeled as a linear combination of its parent set $\mathrm{Pa}(X_j)$,

$$X_j = \sum_{X_i \in \mathrm{Pa}(X_j)} X_i w_{ij} + \varepsilon_j, \qquad j = 1, \ldots, p. \tag{2.6}$$

where ε_j are independent random variables. Here, $w_{ij} \in \mathbb{R}$ represents the weight associated with the arc $X_i \to X_j$. A notationally more convenient way is to say that $w_{ij} = 0$ if $X_i \notin \mathrm{Pa}(X_j)$, and non-zero otherwise. Then, we can also write Equation 2.6 as

$$X_j = \sum_{i=1}^{p} X_i w_{ij} + \varepsilon_j = X w_j^T + \varepsilon_j, \qquad j = 1, \ldots, p. \tag{2.7}$$

where $w_j = (w_{1j}, w_{2j}, \ldots, w_{pj})$ represents the row-vector containing these weights. Furthermore, w_j^T corresponds to the transpose of w_j so that w_j^T corresponds to a column-vector. As X_j cannot be a parent of itself, we can already deduce that $w_{jj} = 0$.

More concisely, we can write Equation 2.7, consisting of p equations, into one vector X as

$$X = XW + \varepsilon, \tag{2.8}$$

where $W = \left(w_1^T, w_2^T, \ldots, w_p^T\right)$ and $\varepsilon = (\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_p)$. Here, w_i^T again represents the transpose of the row-vector w_i. Note that the diagonal of W contains only zeros, as $X_i \notin \text{Pa}(X_i)$.

A crucial remark for Equation 2.8 to correctly specify a generative model is to assume that there are no cyclic dependencies induced by W. For example, if $X_1 = 0.5X_2 + \varepsilon_1$ and $X_2 = 0.8X_1 + \varepsilon_2$, we cannot generate data according to this model. Therefore, let us briefly deviate to explain the notion of acyclicity.

Acyclicity. As mentioned in the introduction, we will explicitly enforce our recovered structure to be *acyclic*. Enforcing acyclicity greatly reduces the search space of possible structures; if we know that $X_1 \to X_2$, we can immediately disregard all structures that also have the relation $X_1 \leftarrow X_2$. Furthermore, as we have seen, the generative model for Equation 2.7 does not exist if the structure is cyclic. Therefore, we see that this notion of acyclicity is crucial.

Luckily, the notion of acyclicity extends quite easily to graphical models. Suppose we have a graphical model, which is either a time-independent graphical model as given in Equation 2.8 or a time-dependent graphical model, which will be discussed later on in this chapter. The corresponding coefficient matrix W parameterized this graphical model. The support of our inferred matrix W, $\text{supp}(W) = \{(i,j) \mid w_{ij} \neq 0\}$, corresponds to the adjacency matrix of the graph summarizing the learned structure. Therefore, requiring the structure to be acyclic is equivalent to enforcing acyclicity of $G(W)$, the graph induced by the support of W.

In our setting, a graph is defined to be *acyclic* if there are no paths $X_i \to \ldots \to X_i$ for any $i = 1, \ldots, p$, corresponding to a sequence of at least two unique transitions such that we start at a node X_i and return to the same node X_i. Since this definition for time-series graphical models is central to our thesis, we have made this statement more explicit in Definition 2.1.

Definition 2.1 Acyclicity, Directed Acyclic Graph with Time-Dependency.

Let $W \in \mathbb{R}^{p \times p}$ be a real-valued matrix. Furthermore, let $G(W)$ be the graph induces by the matrix W, where we draw an arc from variable X_i to variable X_j if and only if $w_{ij} \neq 0$. We define a cycle of length k to be a sequence of k unique arc transitions such that we both start and end at the same variable: $X_i \to \ldots \to X_i$ for some $i = 1, \ldots, p$.

Then, we say that $G(W)$ is *acyclic* if and only if there do not exist any cycles of length greater than one. Therefore, so-called self-loops $X_i \to X_i$ are allowed for time-dependent graphical models. We call such a graph $G(W)$ a *directed acyclic graph* or DAG for short. We will use the name `DAGs` to refer to the set of all directed acyclic graphs.

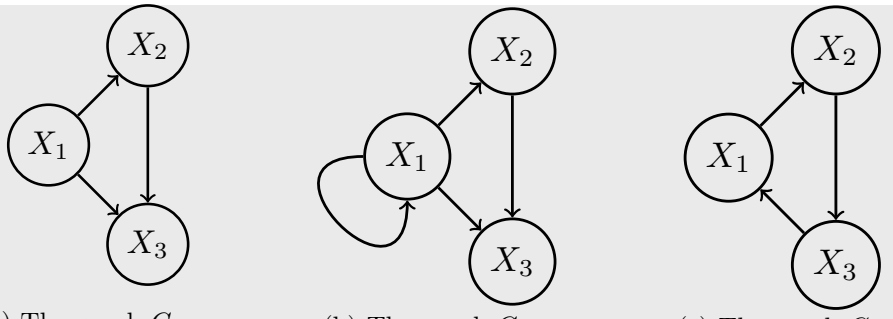
Three graphs that further clarify our notion of acyclicity have been given in Example 2.2.

Example 2.2 Two directed acyclic graphs and one directed cyclic graph.

Let us further clarify the notion of acyclicity by looking at the graphical models corresponding to the three adjacency matrices

$$W_a = \begin{pmatrix} 0 & 1 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}, \qquad W_b = \begin{pmatrix} 1 & 1 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}, \qquad W_c = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}.$$

We now have chosen weights with a value of 1 for simplicity, but note that we could have chosen any non-zero value instead of the value 1. The corresponding summary graphs $G_a = G(W_a), G_b = G(W_b)$, and $G_c = G(W_c)$ have been visualized in Figure 2.1.



(a) The graph G_a. (b) The graph G_b. (c) The graph G_c.

Figure 2.1: Three different graphs G_a, G_b, and G_c on $p = 3$ vertices.

Let us inspect the graphs and see which ones are acyclic according to Definition 2.1. G_a consists of three edges $X_1 \to X_2$, $X_1 \to X_3$, and $X_2 \to X_3$. As it does not contain any cycles, G_a is a DAG. G_b consists of the same edges, but now a self-loop $X_1 \to X_1$ 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 $X_1 \to X_3$ is flipped, resulting in the edge $X_1 \leftarrow X_3$. Consequently, G_c contains a cycle of length 3 now, which we do not consider to be a DAG. In conclusion, $G_a, G_b \in \texttt{DAGs}$, and $G_c \notin \texttt{DAGs}$.

Having defined the notion of acyclicity that will be used throughout this thesis, let us return to defining the linear graphical model for Bayesian networks. Note that if W is acyclic, Equation 2.8, reiterated here as

$$X = XW + \varepsilon, \tag{2.9}$$

now correctly specifies a generative model. We can first generate the variable without any parents, after which we can generate the next variable whose parents have already been generated, etc. Such a generative model is called a linear Structural Equation Model, often shortened as linear SEM. As this model is mentioned often throughout this thesis, the definition has been further clarified in Definition 2.2.

Definition 2.2 Linear Structural Equation Model (linear SEM).

Let $X \in \mathbb{R}^p$ be a real-valued vector containing the values of p variables $X_1, X_2, \ldots, X_p$. If we assume that X has been generated according to a linear Structural Equation Model (linear SEM), then we assume the following generative distribution:

$$X = XW + \varepsilon, \tag{2.10}$$

where $W \in \mathbb{R}^{p \times p}$ is the coefficient matrix that represents the linear relationships between the p variables. Its coefficients w_{ij} denote the linear contribution of variable X_i to variable X_j. Furthermore, the p-dimensional vector ε denotes the error or noise component, which we assume is a Gaussian random variable with zero mean and covariance matrix Σ,

$$\varepsilon \sim \mathcal{N}(\mathbf{0}, \Sigma), \tag{2.11}$$

which is independent of X, that is, $\varepsilon \perp\!\!\!\perp X$. As the noise is a Gaussian random variable, this model is often called a linear Gaussian SEM in the literature. Throughout this thesis, we will stick to the more concise notation of a linear SEM. For simplicity, we often assume that $\Sigma = I_p$. Note that this linear SEM models *instantaneous* relationships, so there is no notion of time. Furthermore, as we cannot have any contribution of a variable X_i to itself, we fix the diagonal of W to equal zero.

Equations 2.10 and 2.11 together define the linear SEM.

In literature, this linear SEM is often used to model a Bayesian network. However, as mentioned before, these linear SEMs are only capable to model *instantaneous* relationships, meaning they fail to capture any notion of time. Therefore, let us now change our focus to graphical models to capture temporal relationships.

Time-dependent graphical models. Taking into account the notion of time, we know that there will be dependency between the variables across time. Therefore, the joint distribution does not factorize as nicely as in Equation 2.3. However, we can use the general product rule of joint distributions to conclude that

$$\begin{aligned}\mathbb{P}(\mathbf{X}) &= \mathbb{P}\left(X_{1,\cdot}= x_{1,\cdot}, X_{2,\cdot}= x_{2,\cdot}, \dots X_{T,\cdot}= x_{T,\cdot}\right)\\ &= P(X_{1,\cdot}= x_{1,\cdot})P(X_{2,\cdot}= x_{2,\cdot}, \dots, X_{T,\cdot}= x_{T,\cdot}|X_{1,\cdot}= x_{1,\cdot})\\ &= P(X_{1,\cdot}= x_{1,\cdot})P(X_{2,\cdot}= x_{2,\cdot}|X_{1,\cdot}= x_{1,\cdot})P(X_{3,\cdot}= x_{3,\cdot}, \dots, X_{T,\cdot}= x_{T,\cdot}|X_{2,\cdot}= x_{2,\cdot}, X_{1,\cdot}= x_{1,\cdot})\\ &= P(X_{1,\cdot}= x_{1,\cdot})\prod_{t=2}^{T} P(X_{t,\cdot}= x_{t,\cdot}|X_{1,\cdot}= x_{1,\cdot}, \dots, X_{t-1,\cdot}= x_{t-1,\cdot}),\end{aligned} \tag{2.12}$$

where $X_{t,\cdot}$ represents the p measurements of our p variables at time step t. As stated in Equation 2.12, $X_{t,\cdot}$ depends on all $t-1$ previous time steps. This structure is quite difficult to model, and therefore we assume all variables $X_{t,\cdot}$ only depend on the *previous* time step $t-1$. Under this strict assumption, Equation 2.12 factorizes as

$$\mathbb{P}(\mathbf{X}) = P(X_{1,\cdot}= x_{1,\cdot})\prod_{t=2}^{T} P(X_{t,\cdot}= x_{t,\cdot}|X_{t-1,\cdot}= x_{t-1,\cdot}). \tag{2.13}$$

Let us now consider the conditional probability $\mathbb{P}\left(X_{t,\cdot}|X_{t-1,\cdot}\right)$, or equivalently when we again include subscripts for the p variables, $\mathbb{P}\left(X_{t,1}, \dots, X_{t,p}|X_{t-1,1}X_{t-1,p}\right)$. A further assumption is that the value of $X_{t,1}$ does not depend on the previous values of *all* p variables, but merely a subset of these p variables. Referring back to the sprinkler example, the amount of rainfall did not depend on the sprinkler installation or the wetness of the pavement; it was only influenced by the seasonality. Therefore, we assume that there is only a subset of the p variables $X_{t-1,\cdot}$ that influence the value of the variable $X_{t,i}$. We again denote this set by the parent set of $X_{t,i}$, or $\mathrm{Pa}\left(X_{t,i}\right)$. Then, Equation 2.13 is further simplified by removing the dependency of $X_{t,i}$ on variables not in the parent set $\mathrm{Pa}\left(X_{t,i}\right)$,

$$\mathbb{P}(X_{t,1}, X_{t,2}, \dots X_{t,p}) = \prod \mathbb{P}(X_{t,i}|\mathrm{Pa}(X_{t,i})). \tag{2.14}$$

As can be seen from Equation 2.13, we assume the parent set only consists of variables from the previous time step, that is, $\mathrm{Pa}(X_{t,i}) \subseteq \{X_{t-1,1}, X_{t-1,2}, \dots, X_{t-1,p}\}$. Note that $X_{t,i}$ can depend on its previous measurement $X_{t-1,i}$. In fact, the most recent past $X_{t-1,i}$ can often be very helpful in predicting $X_{t,i}$. We can again visualize this probability distribution by drawing arcs from $X_{t-1,i}$ to $X_{t-1,j}$ if and only if $X_{t-1,j} \in \mathrm{Pa}\left(X_{t,i}\right)$. Therefore, the parent sets $\mathrm{Pa}\left(X_{t,i}\right)$ for $i = 1, \dots p$ define the arc set corresponding to the graphical model. Such a factorization has been further explained with the time-dependent sprinkler example in Example 2.3.

Example 2.3 Factorization of the time-dependent sprinkler example.

To see an example, let us again consider the sprinkler example where the five variables have been measured over a time. It is reasonable to assume that the previous value $X_{t,i}$ will influence the present value $X_{t,i}$, for example, if the season variable at $X_{t-1,1}$ corresponds to summer, then that will be very useful in predicting $X_{t,1}$, which will most likely also correspond to summer. Therefore, we know that all five variables will be influenced by their past, $X_{t-1} \subseteq \mathrm{Pa}(X_{t,i})$ for $i = 1, 2, \dots, 5$. Furthermore, we will assume that the deduced relations from Figure 1.1 still apply. Therefore, the factorization for the time series setting will be similar to Equation 2.5, but now we include the time index and assume the past values

$X_{t-1,\cdot} = (X_{t-1,1}, \ldots, X_{t-1,p})$ influence the present values $X_{t,\cdot} = (X_{t,1}, \ldots X_{t,p})$. Therefore, the joint probability can be factorized as

$$\begin{aligned}\mathbb{P}(X_{t,1}, X_{t,2}, X_{t,3}, X_{t,4}, X_{t,5}) &= \prod_{i=1}^{5} \mathbb{P}(X_{t,i} | \mathrm{Pa}(X_{t,i})) \\ &= \mathbb{P}(X_{t,1}|X_{t-1,1})\mathbb{P}(X_{t,2}|X_{t-1,1}, X_{t-1,2})\mathbb{P}(X_{t,3}|X_{t-1,1}, X_{t-1,3}) \\ &\quad \mathbb{P}(X_{t,4}|X_{t-1,2}, X_{t-1,3}, X_{t-1,4})\mathbb{P}(X_{t,5}|X_{t-1,4}X_{t-1,5}). \quad (2.15)\end{aligned}$$

Just as with the Bayesian network, an advantage of these model assumptions is that this factorization again allows for an intuitive graphical display. Once again, we can visualize the p variables over time as nodes, and draw arcs $X_{t-1,i} \to X_{t,j}$ to denote that $X_{t,j} \in \mathrm{Pa}\,(X_{t-1,i})$. As these relations hold for any time index t, we only need to draw the graph corresponding to the time steps $t-1$ and t. The time-dependent *window* graph for the sprinkler example is shown in Figure 2.2. Alternatively, we can only show the p nodes once and an arc from node X_i to node X_j implies an arc from $X_{t-1,i}$ to $X_{t,j}$. This *summary* graph is shown in Figure 2.3.

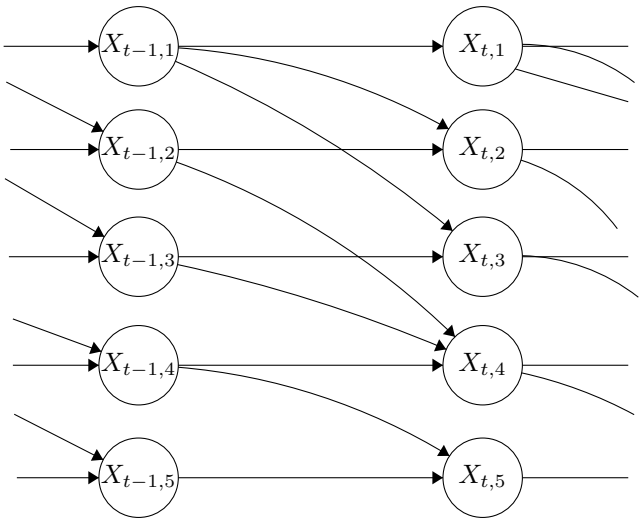


Figure 2.2: Time-Dependent graph depicting the factorized probability distribution in Equation 2.15 from the sprinkler example.

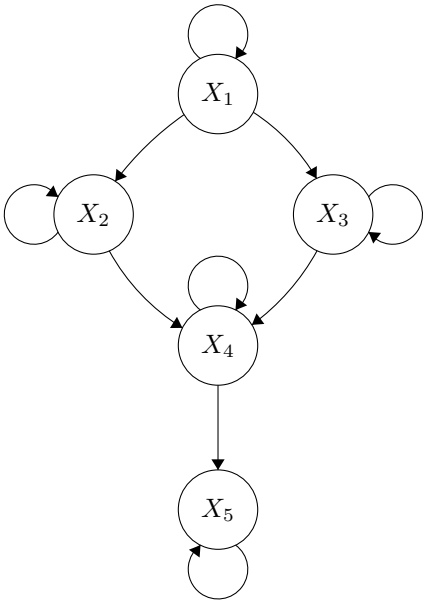


Figure 2.3: Time-Dependent summary graph depicting the factorized probability distribution in Equation 2.15 from the sprinkler example.

To fully specify the model requirements, in addition to the structure given by the factorization from Equation 2.14, we need a characterization of the conditional distribution of X_i given its parent set $\mathrm{Pa}\,(X_i)$. As the set of all possible conditional distributions is too broad, a simple yet common model is to assume a *linear* relation between X_i and its time-lagged parent set, yielding a linear graphical model. We will introduce such a linear graphical model in the next paragraph.

A linear generative model with time dependencies. We have now translated the goal of structure learning into learning the unknown joint distribution $\mathbb{P}(\mathbf{X})$. To allow a clear graphical representation of the joint distribution, we assume that this joint distribution can be factorized, such that each variable $X_{t,i}$ only depended on its parent set $\mathrm{Pa}\,(X_{t,i})$. This allows a clear graphical model, where there is an arc from node X_i to X_j in the summary graph if and only if $X_i \in \mathrm{Pa}\,(X_j)$.

Now, let us assume the conditional distribution of X_i given its parent set $\mathrm{Pa}(X_i)$ is *linear* with respect to the parent set. This simple yet common model can be written as

$$X_{t,j} = \sum_{X_{t-1,i} \in \mathrm{Pa}(X_{t,j})} X_{t-1,i} w_{ij} + \varepsilon_j, \tag{2.16}$$

where ε_j represent independent random variables. Furthermore, $w_{ij} \in \mathbb{R}$ represents the linear weight associated with the arc $X_{t-1,i} \to X_{t,j}$.

A more convenient notation is to say that $w_{ij} = 0$ if $X_{t-1,i} \notin \mathrm{Pa}(X_{t,j})$. Then, we can also write

$$X_{t,j} = \sum_{i=1}^{p} X_{t-1,i} w_{ij} + \varepsilon_{t,j} = X_{t-1,\cdot} w_j^T + \varepsilon_{t,j}, \tag{2.17}$$

where w_j represents the row vector containing the weights $(w_{1j}, w_{2j}, \ldots, w_{pj})$. Again, we assume that $\varepsilon_{t,j}$ are all independent random variables. Instead of looking at just one variable $X_{t,j}$, we can also combine all p variables into one vector X_t and the corresponding generative model will be

$$X_{t,\cdot} = X_{t-1,\cdot} W + \varepsilon_t, \tag{2.18}$$

where $W = \left(w_1^T, w_2^T, \ldots, w_p^T\right)$ and $\varepsilon_t = (\varepsilon_{t,1}, \varepsilon_{t,2}, \ldots, \varepsilon_{t,p})$.

We have decomposed the learning of the structure of the joint distribution into the learning of the weights in this coefficient matrix W. By construction, these coefficients w_{ij} correspond to the weight associated with the relationship $X_{t-1,i} \to X_{t,j}$. If $w_{ij} = 0$, then we say that there is no such arc, or equivalently, $X_{t-1,i} \notin \mathrm{Pa}(X_{t,j})$.

The generative model described in Equation 2.18 corresponds to a so-called *Vector AutoRegression* model with a lag of order one, short-hand a VAR(1) model. As this type of model is central to our thesis, we will further clarify this model in Definition 2.3.

Definition 2.3 Vector AutoRegression Model of order 1 (VAR(1)).

Let p be the number of variables, and T be the number of time steps. If we assume that $\mathbf{X} \in \mathbb{R}^{T \times p}$ has been generated according to a VAR (1) model, then this means that we assume the following generative distribution:

$$X_{t,\cdot} = X_{t-1,\cdot} W + \varepsilon_t, \tag{2.19}$$

for $t = 2, \ldots, T$. Furthermore, $W \in \mathbb{R}^{p \times p}$ is the coefficient matrix that represents the linear relationships between the p variables. In fact, w_{ij} denotes the linear contribution of variable $X_{t-1,i}$ to variable $X_{t,j}$.

Furthermore, we assume that the initial value for X_1 is either given or follows the stationary distribution of X_t, such that $\mathbb{E}[X_1] = \mathbb{E}[X_t]$ and $\mathbb{V}(X_1) = \mathbb{V}(X_t)$. Additionally, we assume that the noise or error component ε_t is normally distributed with mean zero and some covariance matrix $\Sigma \in \mathbb{R}^{p \times p}$,

$$\varepsilon_t \sim \mathcal{N}(0, \Sigma), \tag{2.20}$$

for $t = 2, \ldots, T$. Furthermore, we assume that the noise components are independent of each other, $\varepsilon_t \perp\!\!\!\perp \varepsilon_{t'}$ for $t \neq t'$. Note that ε_t and $\mathbf{X}$ are dependent, as ε_t is a component of $X_{t',\cdot}$ for $t' \geq t$. Generally, we assume that $\Sigma = I_p$.

Equations 2.19 and 2.20, together with an initial value X_1 and the assumptions on ε_t, define the VAR(1) model.

An important note is that for the VAR(1) model as defined in Definition 2.3, acyclicity of the structure induced by W is not required for the generative model to be well-defined. Nevertheless, we do require the structure induced by W to be acyclic for several reasons. Arguably, the most important reason is that enforcing acyclicity greatly reduces the set of possible structures, thus reducing the search space.

Summarizing remarks. To summarize, so far we have recast the structure learning problem to a density estimation problem, where we wish to recover the joint probability distribution $\mathbb{P}(\mathbf{X})$. By making certain assumptions about the factorization of the data, we have either a time-independent graphical model or a time-dependent graphical model. Additionally, by assuming a linear model with Gaussian noise, we derive the time-independent linear SEM from Definition 2.2 and the time-dependent VAR(1) model from Definition 2.3, respectively. Both linear generative models can be parameterized by a single coefficient matrix W. We require the structure to be acyclic, meaning that the graph induced by W must be acyclic as per Definition 2.1.

A logical next step would be to estimate a suitable matrix W for either one of the graphical models, while ensuring the structure induced by W is acyclic as per Definition 2.1. For this, there are several interesting approaches, each having a different perspective. In Chapter 3, we will explain the concepts with accompanying examples, as well as several algorithms to learn acyclic structures from either time-independent data or time-dependent data.

Chapter 3

Related Work

In Chapter 1, we have introduced the notion of *structure learning* on a conceptual level. We have discussed several applications and motivating examples to underline the usefulness of learning a structure associated with the model to have given rise to the data matrix $\mathbf{X}$. In Chapter 2, the objective of this thesis has been made more explicit with corresponding mathematical notation. We are interested in learning the joint probability of the data matrix $\mathbf{X}$, casting the problem to a *density estimation* task. We have made several assumptions on the factorization of the joint probability of $\mathbf{X}$, most notably that each variable X_i only depends on its parent set $\mathrm{Pa}(X_i)$. Methods to learn such a suitable factorization can be decomposed into three main categories.

The first main category was also the first method used for structure learning. This type of approach was first proposed by Verma and Pearl in [56] where edge constraints were derived using conditional independence tests on multiple subsets of variables, together with their developed framework of causal inference. These edge constraints together formed a skeleton of undirected edges, from which directed relationships could be deduced. We categorize these approaches as *constraint-based* approaches, which will be discussed in greater detail in Section 3.1.

The second approach to learning a directed acyclic structure is by exploiting asymmetries in the noise. Without any additional assumptions on the data, determining the direction of instantaneous relations between variables is impossible [80]. In other words, we can detect that an arc exists in either direction, yet it is impossible to determine whether $X_i \to X_j$ or $X_i \leftarrow X_j$. When we make some assumptions about the stochastic relation between X_i and X_j, we can determine the directionality of the arc. As these methods rely on assumptions about the distribution of the noise attributed to the variables, this approach is called *noise-based*. A deeper explanation of the principles behind noise-based approaches will be discussed in Section 3.2.

The third category is named *score-based* approaches, which has seen the most developments in the past years. Score-based approaches assess the validity of a given structure $\mathcal{G}$ by assigning, as the name suggests, a score to each structure based on how well $\mathcal{G}$ fits the data $\mathbf{X}$. A scoring function $S(\mathbf{X}, \mathcal{G})$ determines this score. Consequently, the optimal structure is the structure $\mathcal{G}^*$ that maximizes the scoring function $S(\mathbf{X}, \mathcal{G})$ while remaining acyclic,

$$\mathcal{G}^* = \underset{\text{structures } \mathcal{G}}{\arg\max}\ S(\mathbf{X}, \mathcal{G}) \text{ such that } \mathcal{G} \in \texttt{DAGs}. \tag{3.1}$$

Unfortunately, maximizing such a score function over the set of directed acyclic graphs is NP-hard for most interesting scoring functions [13]. Therefore, we cannot expect to find $\mathcal{G}^*$ in polynomial time. Nevertheless, researchers have developed several interesting approaches that efficiently search for a suboptimal yet satisfactory structure. Furthermore, useful techniques have been developed to find an optimal structure $\mathcal{G}^*$ as efficient as possible, albeit still exponential in running time. We will cover several of these score-based approaches in Section 3.3.

3.1 Constraint-Based Approaches

Constraint-based approaches were among the first types of approaches to identify causal relationships in the causal framework introduced by Pearl [54]. Constraint-based approaches consist of, as the name suggests, identifying *edge constraints* using conditional independence tests.

Identifying the skeleton. First, we identify the skeleton of the structure using conditional independence tests. These conditional independence tests can only detect undirected relations, so the skeleton contains only undirected edges. Let $V = \{X_1, \ldots, X_p\}$ be the set of all p variables. Furthermore, we denote by $X_i \perp\!\!\!\perp X_j \mid S$ that variable X_i and X_j are conditionally independent given a set of variables S. We denote conditional dependence by $X_i \not\perp\!\!\!\perp X_j \mid S$. An undirected edge between two variables X_i and X_j exists if and only if they are conditionally dependent for all subsets S not containing X_i and X_j,

$$\text{edge } (X_i, X_j) \text{ in the skeleton of } \mathcal{G} \iff X_i \not\perp\!\!\!\perp X_j \mid S \qquad \forall\, S \subseteq V \setminus \{X_i, X_j\}. \tag{3.2}$$

In other words, there does not exist any set S without X_i and X_j such that X_i and X_j are conditionally independent; the conditional dependence cannot be removed.

A more intuitive perspective is the converse of Equation 3.2. If we can find a set S such that X_i and X_j are conditionally independent, then the undirected edge (X_i, X_j) is not included in the skeleton,

$$\exists\, S \subseteq V \setminus \{X_i, X_j\} : X_i \perp\!\!\!\perp X_j \mid S \Rightarrow \text{edge } (X_i, X_j) \text{ not in the skeleton of } \mathcal{G}. \tag{3.3}$$

We can derive the skeleton of direct dependencies by starting with a fully connected undirected graph and removing edges using the rule in Equation 3.3. In real-life data, we can use any conditional independence test such as the Fisher-z's test [24] to determine whether the conditional dependence is significant. However, this requires a large amount of conditional independence tests, which causes issues as the accuracy of skeleton recovery decreases [44]. Furthermore, conditional independence tests require many samples which may not always be available [65]. These are the two main drawbacks of constraint-based approaches.

Identifying immoralities. Having identified the skeleton of our causal network, we still need to orient the edges. Constrained-based approaches use the fact we can deduce the orientation of edges from already derived conditional (in)dependencies and orientations. Firstly, we can orient edges by identifying *immoralities*. Suppose that we have discovered using independence tests that $X - Y - Z$, meaning that there is a direct dependency between X and Y and between Y and Z. Furthermore, assume that X and Z are independent conditioned on some subset that does not include Y. We then call the triple (X, Y, Z) an *immorality*, *v-structure*, or *collider* [49].

Now, X and Z are independent given some set of variables not including Y, yet there is a direct dependency between both X and Y and Z and Y. Therefore, Y cannot cause X and Z and thus we can deduce that X and Z both cause Y. Therefore, we can orient such an immorality as $X \rightarrow Y \leftarrow Z$.

Further orientation of edges. Once redundant edges have been removed from the complete graph and all immoralities have been oriented using conditional independence tests, we can use a set of rules based on the skeleton and the conditioning sets S to learn the correct orientation of the other edges. A simple set of just four rules that are proven to be both sound and complete is called the "Meek rules" [47]. The Meek rules orient edges so that we do not introduce any additional immoralities or any directed cycles in the structure.

Sometimes, deducing the correct orientation of the edge is not possible. Therefore, constraint-based approaches often output a maximally oriented graph where some edges remain undirected. Such a graph is called a *Completed Partially Directed Acyclic Graph* (CPDAG). It represents the *Markov Equivalence Class* (MEC) containing all possible directed acyclic graphs with the same joint probability distribution, in the sense that they are statistically indistinguishable.

Pseudocode. The pseudocode for the constraint-based approach has been given in Algorithm 3.1. The two initial constraint-based approaches are SGS, proposed by Spirtes, Glymour, and Scheines [68], and Inductive Causation (IC), proposed by Verma and Pearl [55]. Although SGS technically outputs the set of possible directed acyclic graphs and IC would return the CPDAG, both outputs boil down to the same Markov Equivalence Class.

Improvements over these two algorithms have been proposed in the subsequent years. The PC-algorithm [69], named after Peter Spirtes and Clarke Glymour, two of the three authors of the SGS algorithm, returns the exact same output as the SGS or IC algorithm, but is significantly faster. The PC-algorithm differs in how it approaches step 2. Instead of trying all possible sets $S \subseteq V \setminus \{X_i, X_j\}$, it employs a more efficient approach by first trying smaller sets of variables adjacent to X_i and X_j, such that the skeleton can be discovered using fewer independence tests.

Algorithm 3.1: Pseudocode of the constraint-based approach.

Input: A data matrix $\mathbf{X} = [X_1, X_2, \ldots, X_p]$.
Output: A maximally oriented CPDAG $\mathcal{G}$, corresponding to a set of statistically indistinguishable causal models of $\mathbf{X}$.

1: let $\mathcal{G}$ be a complete undirected graph
2: remove all undirected edges (i, j) in $\mathcal{G}$ if there exists a set $S \subseteq V \setminus \{X_i, X_j\}$ such that X_i and X_j are conditionally independent given S, $X_i \perp\!\!\!\perp X_j \mid S$
3: orient all *immoralities* in the remaining graph structure $\mathcal{G}$
4: **while** we can orient edges in $\mathcal{G}$ **do**
5: orient all edges if the reverse direction would introduce an immorality
6: orient all edges if the reverse direction would introduce a directed cycle
7: **end while**
8: **return** the maximally oriented CPDAG $\mathcal{G}$

Example. To see how constraint-based approaches work, let us consider the following example on five variables, inspired by [49]. The ground truth has been visualized in Figure 3.3.

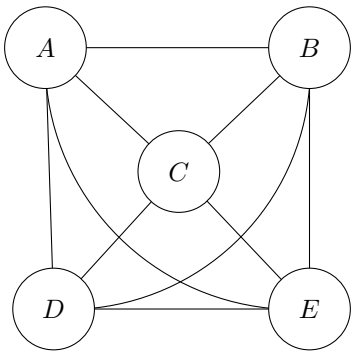


Figure 3.1: Complete undirected graph.

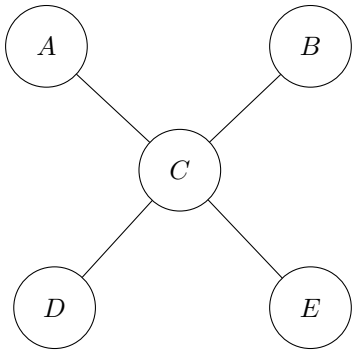


Figure 3.2: Skeleton inferred by conditional independence tests.

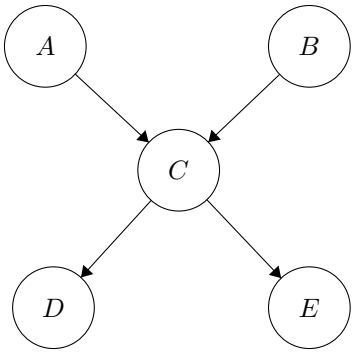


Figure 3.3: Ground truth, also recovered using a constraint-based approach.

To obtain the correct graph using a constraint-based approach, we will first learn the *skeleton* of the structure. The skeleton consists of all edges that are dependent even after we condition on all other subsets of variables. To identify such a skeleton, we will start with a fully connected graph in Figure 3.1 and remove edges $X_i - X_j$ if we can find a set S such that $X_i \perp\!\!\!\perp X_j \mid S$. That is, we are looking for a set such that X_i and X_j are independent when we condition on the variables in the set S.

Assuming our conditional independence tests agree with the ground truth, we will remove the edge $A - B$, as A and B are conditionally independent, even given the empty set. Additionally, we will remove all edges from either A or B to either D or E, as all four pairs of variables are conditionally independent given C. Lastly, we know that D and E are independent given C as well. Therefore, we can successfully recover the skeleton, assuming our conditional independence tests are correct.

Now, to orient the edges, we will first see if we can detect any immoralities. Consider the triple $A - C - B$. We have verified that there is no edge between A and B. Furthermore, we did not require conditioning on C to ensure that A and B were conditionally independent. Therefore, the triplet $A - C - B$ is considered an immorality, and we can orient these edges as $A \to C \leftarrow B$. Note that any other triple $\ldots - C - \ldots$ in the skeleton would not have been such an immorality, as we required conditioning on C to achieve conditional independence.

Subsequently, we can orient the other two edges as well. If we assumed orientation $C \to D$, then we would have introduced an additional immorality $A \to C \leftarrow D$, which would have been incorrect, as mentioned before. Therefore, according to the Meek rules, the proper orientations are $C \to D$ and $C \to E$, which corresponds to the ground truth.

PCMCI: Extending to time series data. The SGS/IC and PC algorithm were all developed for time-independent data and were therefore only able to model instantaneous relations. The PC-algorithm can also be extended to time series data. However, we cannot use the PC-algorithm directly, as the autocorrelations in time series data lead to high false positive rates for the conditional independence tests [63].

The PCMCI algorithm was introduced for time series data to overcome this issue. First, the PC-algorithm is used to remove edges by checking for conditional independence between time-lagged variables, even though this may lead to a many false positives due to autocorrelation. The remaining edges are assessed using Momentary Conditional Independence (MCI) tests to (hopefully) address these false positives. These MCI tests are used to verify whether a time lagged-variable $X^i_{t-\tau}$ and a non time-lagged variable X^j_t are independent when we condition on both parent sets identified by the PC-algorithm without the time-lagged variable $X^j_{t-\tau}$,

$$\text{MCI} : X^i_{t-\tau} \perp\!\!\!\perp X^j_t \mid \text{Pa}\left(X^i_{t-\tau}\right), \text{Pa}\left(X^j_t\right) \setminus X^j_{t-\tau}. \tag{3.4}$$

Note that here, the subscript represents the time index, and the superscript represents the variable index. The orientation of edges is done using the same rule set as the PC-algorithm, but as the future cannot cause the past, we know that all edges should be oriented from past to future, so $X^i_t \to X^j_{t'}$ if $t < t'$. The initial PCMCI algorithm could not detect instantaneous causal relations, but this has been made possible by extending the algorithm in [62].

Several other constraint-based algorithms exist, mostly focused on improving the computational performance of the PC-algorithm. The Fast Causal Inference (FCI) algorithm was proposed in 1993 [68] and the Really Fast Causal Inference (RFCI) algorithm was proposed in 2012 [14]. The FCI algorithm enjoys several time series extensions in the form of the SVAR-FCI algorithm [45] and the tsFCI algorithm [22]. However, these methods also suffer from the aforementioned drawbacks of constraint-based approaches, namely that a large number of conditional independence tests limits their ability to consider a large number of variables, especially when a limited number of samples is available.

3.2 Noise-Based Approaches

As mentioned before, it is impossible to determine the directionality of the effect without certain noise assumptions. In other words, the corresponding structure is then *unidentifiable*. Let us explain this phenomenon in the following example.

Example of unidentifiability of a linear bivariate Gaussian model. Consider a linear Gaussian SEM on two variables which is defined as follows:

$$X = \varepsilon_X, \qquad \varepsilon_X \sim \mathcal{N}(0, 1)\,. \tag{3.5}$$
$$Y = 0.8X + \varepsilon_Y, \qquad \varepsilon_Y \sim \mathcal{N}(0, 0.36)\,. \tag{3.6}$$

Note that the second argument of the normal distribution corresponds to the variance, not the standard deviation. These values have been chosen such that both X and Y have a mean equal to zero and unit variance.

Equation 3.5 and Equation 3.6 define a generative model capturing the relation $X \to Y$. However, when we look purely at observational data, we cannot distinguish between the model from defined by Equation 3.5 and Equation 3.6 and the model given by

$$X = 0.8Y + \tilde{\varepsilon}_X, \qquad \tilde{\varepsilon}_X \sim \mathcal{N}(0, 0.36) \tag{3.7}$$
$$Y = \tilde{\varepsilon}_Y, \qquad \tilde{\varepsilon}_Y \sim \mathcal{N}(0, 1) \tag{3.8}$$

In both scenarios, X, Y are both standard normal random variables, where their joint distribution is a bivariate normal distribution,

$$\begin{pmatrix} X \\ Y \end{pmatrix} \sim \mathcal{N}\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 1.0 & 0.8 \\ 0.8 & 1.0 \end{bmatrix} \right). \tag{3.9}$$

Therefore, both models result in the same joint distribution of X and Y. This has also been shown in Figure 3.4 and Figure 3.5, where we have visualized the joint distributions of both models.

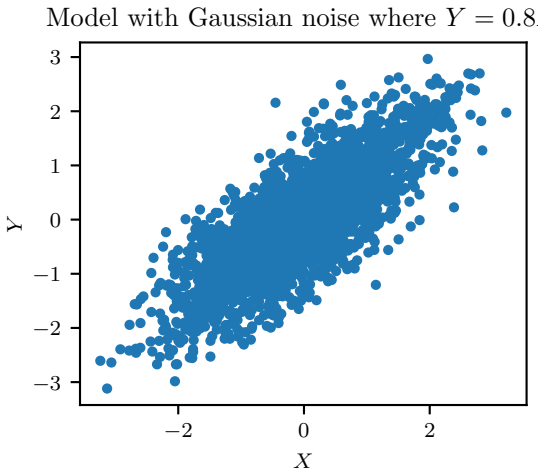


Figure 3.4: Scatter plot of the linear Gaussian model defined by Equations 3.5 and 3.6.

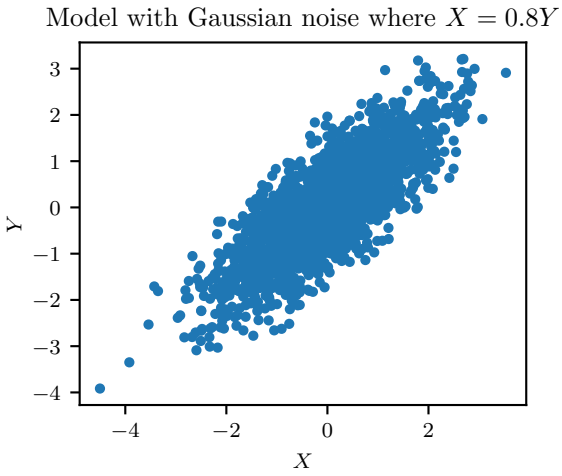


Figure 3.5: Scatter plot of the linear Gaussian model defined by Equations 3.7 and 3.8.

As both linear Gaussian models have the same joint distribution, the "correct" model is *unidentifiable*. The general statement that such a linear Gaussian model is *unidentifiable* has been shown in [66].

Example of identifiability of a linear model with uniform noise. However, the directionality is detectable as soon as we make some assumptions about the noise variables $\varepsilon, \tilde{\varepsilon}$. Let us see how we can discover the correct direction in the following example.

Let us consider the scenario where our noise components follow a *non-Gaussian* distribution, such as the uniform distribution. The noise residuals are now uniformly distributed in such a way that the mean and variance of X and Y do not change,

$$X = \varepsilon_X, \qquad \varepsilon_X \sim U\left(-\sqrt{3}, \sqrt{3}\right). \tag{3.10}$$

$$Y = 0.8X + \varepsilon_Y, \qquad \varepsilon_Y \sim U\left(-0.6\sqrt{3}, 0.6\sqrt{3}\right). \tag{3.11}$$

Furthermore, the covariance matrix of the joint probability does not change compared to Equation 3.9, but now the joint probability does not follow a Gaussian distribution anymore. Nevertheless, the regression coefficient obtained using a least-squares estimation procedure will still equal 0.8 (in the population setting) for both orientations. The scatter plots have been provided in Figure 3.6 and Figure 3.7.

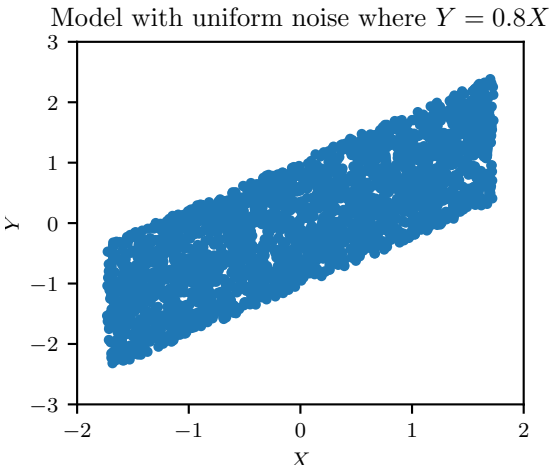


Figure 3.6: Scatter plot of the linear model $Y = 0.8X$ with uniform noise.

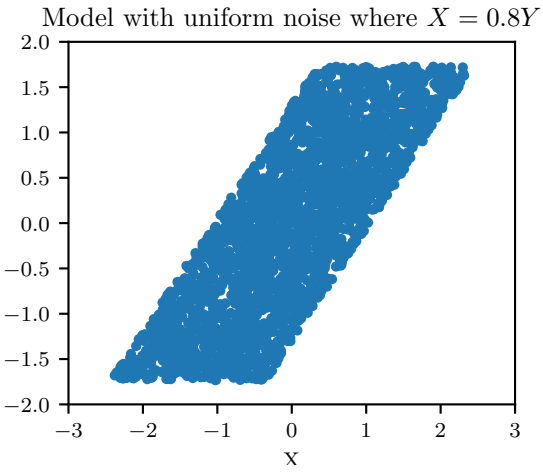


Figure 3.7: Scatter plot of the linear model $X = 0.8Y$ with uniform noise.

Both scatter plots show the linear relationship between X and Y. Nevertheless, we see a clear difference. Figure 3.7 shows a steeper linear relationship between X and Y. Therefore, the two models now seem to exhibit a different *joint distribution* of X and Y.

Now that we have shown there is a difference in joint distribution between these two models, let us consider how we can recover the *correct* direction of influence. For this, we will assume that the correct direction is $X \to Y$, and the additive noises ε_X and ε_Y are independent uniform random variables, as defined by Equations 3.10 and 3.11. To verify the correct direction, we can regress Y on X, yielding approximately the regression line $Y = 0.8X$. Furthermore, we can also regress X on Y, yielding the regression line $X = 0.8Y$. Note that both regression coefficients are the same because the variance parameters were carefully selected. These regression lines have been visualized in Figure 3.8. Note that the black regression line in the correct direction sits nicely in the middle of the points in the scatter plot, indicating no correlation between the residual and the explanatory variable X. However, the red regression line for $Y \to X$ seems too low for small values of X and too high for large values of X, indicating that the residuals are correlated with the explanatory variable Y. This can be further visualized by looking at the *residuals* of the corresponding regressions, which have been plotted in Figure 3.9 and Figure 3.10.

We see that the residuals in the correct direction $X \to Y$ are independent of X and Y, whereas the residuals in the incorrect direction $Y \to X$ are *dependent* on X and Y. As we assumed the residuals to be independent of X and Y, the only plausible direction is indeed $X \to Y$. Note, however, that when the relations between variables are non-linear, we can identify the correct direction even with Gaussian noise, as the joint probability distributions of the two generative models are not identical such as in Equation 3.9.

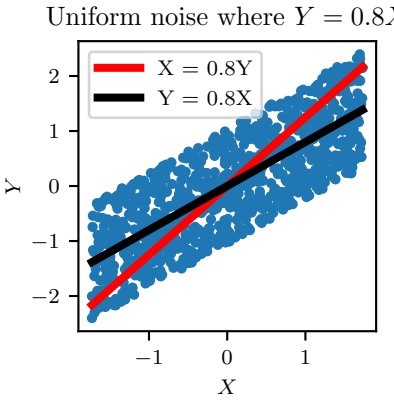


Figure 3.8: Scatter plot of the linear model $Y = 0.8X$ with uniform noise. Both regression lines $X \to Y$ (black) and $Y \to X$ (red) have been plotted as well.

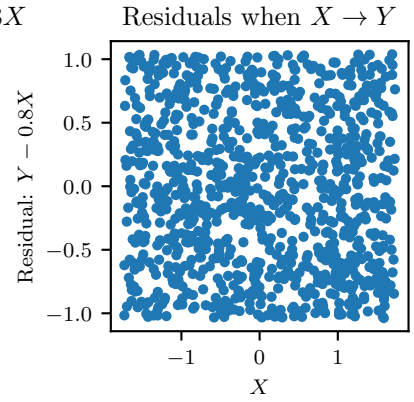


Figure 3.9: Scatter plot of the residuals of the linear model $X = 0.8Y$ with uniform noise, corresponding to the distance to the black line in Figure 3.8.

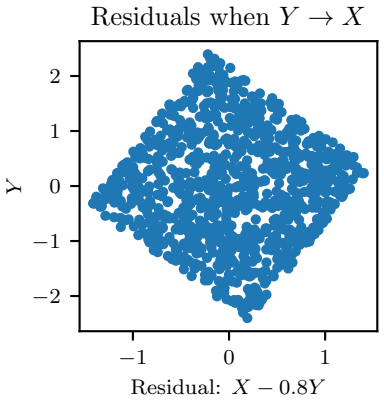


Figure 3.10: Scatter plot of the residuals of the linear model $Y = 0.8X$ with uniform noise, corresponding to the distance to the red line in Figure 3.8.

LiNGAM. Assuming that our residuals follow some non-Gaussian distribution, we can determine whether $X \to Y$ or $X \leftarrow Y$ by verifying whether the residuals are independent. The *Linear Non-Gaussian Acyclic Model* (LiNGAM) [66] utilizes this phenomenon to infer the directed structure between variables.

LiNGAM assumes a linear non-Gaussian model with instantaneous interactions,

$$\mathbf{x} = \mathbf{B}\mathbf{x} + \mathbf{e}, \tag{3.12}$$

where $\mathbf{x}$ is their notation for a data matrix $\mathbf{X} \in \mathbb{R}^{T \times p}$, and $\mathbf{e} \in \mathbb{R}^{T \times p}$ is the random variable that captures the random noise. We can also rewrite Equation 3.12 as

$$\mathbf{x} = \mathbf{A}\mathbf{e}, \tag{3.13}$$

where $\mathbf{A} = (\mathbf{I} - \mathbf{B})^{-1}$. The first step in LiNGAM is to perform an independent component analysis (ICA) to obtain an estimate $\tilde{\mathbf{W}}$ of $\mathbf{W} = \mathbf{A}^{-1}$. This $\mathbf{W}$ needs to be permuted and normalized to obtain an estimate $\hat{\mathbf{B}}$ of $\mathbf{B}$. Lastly, we can derive a causal ordering from this matrix $\tilde{\mathbf{B}}$ which yields a suitable estimate for $\mathbf{B}$.

VARLiNGAM: Extending to time series data. We can extend the LiNGAM setting to a time series setting using the VARLiNGAM approach [35]. Instead of only instantaneous relations, we also model time-lagged relationships through a VAR(τ_{max}) model,

$$\mathbf{x}_t = \sum_{i=0}^{\tau_{max}} \mathbf{A}_i \mathbf{x}_{t-i} + \mathbf{e}. \tag{3.14}$$

We can rewrite this model without an instantaneous effect as

$$\mathbf{x}_t = \sum_{i=1}^{\tau_{max}} \mathbf{M}_i \mathbf{x}_{t-i} + \mathbf{e}, \tag{3.15}$$

where $\mathbf{A}_i = (I - \mathbf{A}_0)\,\mathbf{M}_i$.

We can fit such a VAR(τ_{max}) model on our data $\mathbf{x}$, which yields residuals $\mathbf{e}$. On these residuals, we can do a LiNGAM analysis as mentioned in the paragraph above, which yields an instantaneous coefficient matrix for the instantaneous relations $\hat{\mathbf{A}}_0$. From $\hat{\mathbf{A}}_0$ and $\hat{\mathbf{M}}_i$, in turn, we can compute $\hat{\mathbf{A}}_i = \left(I - \hat{\mathbf{A}}_0\right) \hat{\mathbf{M}}_i$. Using this VARLiNGAM approach, we can estimate the instantaneous effects while considering information from past values.

3.3 Score-Based Structure Learning

Score-based approaches assess the validity of a given structure by assigning a score to each structure. This score is determined by a scoring function $S(\mathbf{X}, \mathcal{G})$. Consequently, the optimal structure is the structure $\mathcal{G}^*$ that maximizes the scoring function $S(\mathbf{X}, \mathcal{G})$, while remaining acyclic,

$$\mathcal{G}^* = \underset{\text{structures } \mathcal{G}}{\arg\max}\ S(\mathbf{X}, \mathcal{G}) \text{ such that } \mathcal{G} \in \texttt{DAGs}. \tag{3.16}$$

Scoring functions that are often used in the literature are the Bayesian Information Criterion (BIC) and the Akaike Information Criterion (AIC), which are both based on the likelihood function of the data.

Globally Optimal Bayesian Network learning using Integer Linear Programming

Finding the structure $\mathcal{G}^*$ that maximizes the scoring function $S(\mathbf{X}, \mathcal{G})$ has been shown to be NP-hard for most interesting scoring functions and therefore, we cannot expect to solve this efficiently for more than a couple of dozens of nodes. Nevertheless, several approaches exist that can quite efficiently find the optimal structure given a scoring function $S(\mathbf{X}, \mathcal{G})$. One of the most well-known exact solvers is known as GOBNILP [16], which is short for *Globally Optimal Bayesian Network learning using Integer Linear Programming*.

Its key to success relies on rewriting the score-based approach. Instead of directly maximizing a scoring function $S(\mathbf{X}, \mathcal{G})$ such that $\mathcal{G}$ is acyclic, they rewrite the problem in Equation 3.16 to a binary Integer Linear Program (ILP). A binary ILP consists of an objective function to maximize subject to a set of constraints, where the assignment variables can only attain the value zero or one, hence the name Integer. By cleverly defining the assignment variables and constraints, we can create an ILP such that the optimal value of the ILP corresponds to the optimal value of the problem in Equation 3.16.

The Integer Linear Program. The Integer Linear Program has been given below. An explanation of the objective function in Equation 3.17 and the sets of constraints in Equations 3.18 - 3.20 follow afterwards.

$$\text{maximize} \quad \sum_{i \in V, J \in \mathcal{P}(i)} c_{i \leftarrow J} x_{i \leftarrow J} \tag{3.17}$$

$$\sum_{J \in \mathcal{P}(i)} x_{i \leftarrow J} = 1 \qquad \forall\ i \in V \tag{3.18}$$

$$\sum_{i \in C} \sum_{J \in \mathcal{P}(i): J \cap C = \emptyset} x_{i \leftarrow J} \geq 1 \qquad \forall\ C \subseteq V, \tag{3.19}$$

$$x_{i \leftarrow J} \in \{0, 1\} \qquad \forall\ i \in V, J \in \mathcal{P}(i) \tag{3.20}$$

The authors first define *binary variables* $x_{i \leftarrow J}$ for each variable $i \in V$, where V corresponds to the set of variables, and each possible parent set $J \in \mathcal{P}(i)$. Here, $\mathcal{P}(j)$ corresponds to all possible parent sets of j, which is equal to the power set of $V \setminus \{i\}$. As each variable has $2^{|V|-1}$ different parent sets, we have an exponential number of binary variables $x_{i \leftarrow J}$. Note that these variables can only attain a value of either zero or one, as is shown in Equation 3.20. If $x_{i \leftarrow J} = 0$, then J is not the exact parent set of variable i. If $x_{i \leftarrow J} = 1$, J corresponds to the parent set of variable i. Now, $c_{i \leftarrow J}$ corresponds to the scoring function when we use the variables in the parent set J to predict the values of variable i.

Continuing with the constraints, we know that each variable i must have precisely one parent set. This set of constraints corresponds to Equation 3.18. Summing over all $0 - 1$ variables representing all possible parent sets of variable i should yield exactly the value 1, indicating that exactly one parent set has been assigned to each variable.

The second set of constraints in Equation 3.19 ensures that there are no cycles in the structure. For example, variable 1 cannot be in the parent set of variable 2 when variable 2 is already in the

parent set of variable 1. This constraint has been cleverly written in Equation 3.19. Every subset $C \subseteq V$ of the variables must contain at least one vertex that has no parent in that subset. To see why this is true, if we would find a subset $C \subseteq V$ such that all vertices have a parent in the subset, then a cycle must exist.

Solving the Integer Linear Program. An advantage of casting this score-based learning problem to an ILP is that there is no need to reinvent any new solver for the problem. Nowadays, highly optimized off-the-shelf ILP solvers exist thanks to decades of research into ILP solvers [4]. However, as the number of assignment variables $x_{i \leftarrow J}$ and constraints are exponential. An optimal solution may take a long time to acquire.

Note that GOBNILP employs many additional tricks to speed up the computations which are not covered here. Furthermore, many improvements have been proposed to increase the computational performance of GOBNILP. The GOBNILP algorithm is actively maintained, and new versions are continuously developed to include the newest performance enhancements, such as different formulations of the constraints [72] and a branch-and-cut approach to reduce the search space [3]. Furthermore, a Python version `pygobnilp` has been introduced as well.

NOTEARS

NOTEARS (*Non-combinatorial Optimization via Trace Exponential and Augmented lagRangian for Structure learning*) is a method developed by Zheng et al. [87]. Rather than trying to find an optimal acyclic structure, they settle for a suboptimal structure with the benefit that their method can scale up to approximately one hundred nodes.

The authors assume a linear structural equation model of the form

$$X = W^T X + z, \tag{3.21}$$

where $X = (X_1, \dots X_p) \in \mathbb{R}^p$ is defined a random vector and $z = (z_1, \dots, z_p)$ defines a random noise vector. Their goal is to infer the matrix $W \in \mathbb{R}^{p \times p}$ from n independent samples of X, forming a data matrix $\mathbf{X} \in \mathbb{R}^{n \times p}$. The most difficult hurdle to overcome is the combinatorial constraint that the inferred structure $G(W)$ must be a directed acyclic graph. In their paper, they present a novel strategy to enforce acyclicity of the structure. Rather than solving the (partly) combinatorial optimization problem on the left-hand side of Equation 3.30, they translate it to an equivalent continuous optimization problem on the right-hand side of Equation 3.30. For any continuous scoring function $F(W)$, they show the equivalence

$$\begin{aligned} \min_{W \in \mathbb{R}^{p \times p}} \quad & F(W) \\ \text{subject to} \quad & G(W) \in \mathsf{DAGs} \end{aligned} \iff \begin{aligned} \min_{W \in \mathbb{R}^{p \times p}} \quad & F(W) \\ \text{subject to} \quad & h(W) = 0, \end{aligned} \tag{3.22}$$

for some function $h(\cdot)$ that enforces acyclicity such that $h(W) = 0$ if and only if $G(W)$ is a directed acyclic graph. Now, the function $F(\cdot)$ that the authors attempt to optimize is the least-squares loss plus an ℓ_1 regularization on the weighted adjacency matrix to encourage sparsity. In mathematical notation, their scoring functions equals

$$F(W) = \frac{1}{2n} \|\mathbf{X} - \mathbf{X}W\|_F^2 + \lambda \|W\|_1, \tag{3.23}$$

where $\mathbf{X} \in \mathbb{R}^{n \times p}$ represents a data matrix of n samples that are p-dimensional. The continuous function $h(W)$ that enforces acyclicity corresponds to

$$h(W) = \operatorname{Tr}\left(e^{(W \circ W)}\right) - p = \operatorname{Tr}\left(\sum_{k=1}^{\infty} \frac{(W \circ W)^k}{k!}\right). \tag{3.24}$$

Here, $\operatorname{Tr}(\cdot)$ represents the trace operator which sums all diagonal entries of its argument, and $W \circ W$ represents the *Hadamard product* of two matrices. In other words, it is the element-wise square of the matrix W.

To explain the function h in greater detail, let us first remark that the trace of $(W \circ W)^k$ corresponds to a weighted sum of all cycles of length k in the matrix $W \circ W$. Therefore, the matrix $W \circ W$ does not contain any cycles of length k if and only if $\text{Tr}\left((W \circ W)^k\right) = 0$. Therefore, $W \circ W$ does not contain any cycles if and only if $h(W) = 0$.

Optimization. The authors propose the *augmented Lagrangian* method to solve the right-hand side of Equation 3.30, which now contains an augmented quadratic penalty,

$$\min_{W \in \mathbb{R}^{p \times p}} F(W) + \frac{\rho}{2}|h(W)|^2 \quad \text{subject to } h(W) = 0, \tag{3.25}$$

where ρ represents the penalty parameter initially equal to one that is iteratively increased.

A Lagrange multiplier α is included to adhere to the constraint $h(W) = 0$, yielding the augmented Lagrangian

$$L^\rho(W, \alpha) = F(W) + \frac{\rho}{2}|h(W)|^2 + \alpha h(W). \tag{3.26}$$

An iterative dual ascent procedure is proposed to find a stationary point of the augmented Lagrangian. Initially, the penalty parameter ρ is equal to one, and the Lagrange multiplier α is equal to zero. For these values, a local optimum W_1 is found using the L-BFGS-B optimization method [90]. If this optimization has provided enough progress with respect to the acyclicity constraint, that is, $h(W_1) < ch(W_0)$ for some hyperparameter $c \in (0, 1)$, then the Lagrange multiplier α is increased and again the L-BFGS-B algorithm is used to find a local optimum W_2. If not, then ρ is iteratively increased until $h(W_1) < ch(W_0)$. When a suitable local optimum W_1 has been found, the Lagrange multiplier α_0 is increased by $\rho h(W_1)$.

Continuing this process, the penalty parameter ρ and Lagrange multiplier α_t keep increasing and local optima are recovered such that $h(W_t)$ is closer and closer to zero. After several iterations, we have found a local optimum to Equation 3.25 such that $h(W)$ is sufficiently close to zero, say no more than some value ϵ.

The solution to the equality constrained program (ECP), W_{ECP} will be thresholded as a final procedure. All entries of W_{ECP} that are smaller in absolute value than some threshold ω will be set to zero to reduce the number of false positives [89]. Experiments done by the authors demonstrate that thresholding increases the accuracy in structure learning, and in their experiments, a threshold value of $\omega = 0.30$ has been proposed. The pseudocode for the NOTEARS procedure has been given in Algorithm 3.2.

Algorithm 3.2: NOTEARS

Input: initial guess W_0, progress rate $c \in (0, 1)$, tolerance $\epsilon > 0$, threshold $\omega > 0$.
Output: an acyclic coefficient matrix $\widehat{W} \in \mathbb{R}^{p \times p}$.

1: **for** $t = 0, 1, 2, \ldots$ **do**:
2: solve primal $W_{t+1} \leftarrow \arg\min_W L^\rho(W, \alpha_t)$ with ρ such that $h(W_{t+1}) < ch(W_t)$
3: dual ascent $\alpha_{t+1} \leftarrow \alpha_t + \rho h(W_{t+1})$
4: if $h(W_{t+1}) < \epsilon$, set $W_{\mathsf{ECP}} = W_{t+1}$ and **break**
5: **end for**
6: **return** the thresholded matrix $\widehat{W} := W_{\mathsf{ECP}} \circ 1(|W_{\mathsf{ECP}}| > \omega)$

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 is no surprise. 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 3.2 to the global optimum found using an exact solver. In this scenario, the authors have used the GOBNILP [16] program to find the global minimum. Quite often, the local optimum was close to the global optimum, demonstrating that the non-convexity is not a large issue in practice.

DYNOTEARS: Extending to time series data. We have seen that NOTEARS only allows for *instantaneous* relationships. However, this can easily be extended to also allow for *time-lagged* relationships. DYNOTEARS [52] assumes that, apart from instantaneous relations, there are also linear time-lagged relations.

The extension of DYNOTEARS to NOTEARS is similar to the extension of VARLiNAM to LiNGAM, which we discussed in Section 3.2. Instead of assuming the instantaneous linear model

$$X = W^T X + z, \tag{3.27}$$

we now also model time-lagged linear relations, up to a time lag of τ_{max},

$$X_{t,\cdot} = X_{t,\cdot} W + \sum_{i=1}^{\tau_{max}} A_i X_{t-i,\cdot} + z, \tag{3.28}$$

where z again represents zero-mean random noise.

Instead of minimizing the cost function $F(W)$ of Equation 3.23 subject to $h(W) = 0$, we now minimize the cost function

$$F'(W) = \frac{1}{2n} \sum_{t=\tau_{max}+1}^{T} \left\| X_{t,\cdot} - X_{t,\cdot} W - \sum_{i=1}^{\tau_{max}} A_i X_{t-i,\cdot} \right\|_2^2 + \lambda_1 \|W\|_1 + \lambda_2 \sum_{i=1}^{\tau_{max}} \|A_i\|_1 , \tag{3.29}$$

where W captures the instantaneous relations, and A_i captures the relations that are time-lagged by i time steps. Additionally, We see that we have added the time-lagged linear relations to the model and an extra sparsity penalty to the autoregressive coefficient matrices A_i. Furthermore, n here corresponds to the effective sample size, for which we have $n = p(T - \tau_{max})$. In their paper, they have $n = p(T + 1 - \tau_{max})$, but their time series also start at time index $t = 0$ rather than our convention of $t = 1$, which explains this small difference in notation.

Just as in NOTEARS, we can enforce acyclicity using a continuous optimization approach by using the translated problem,

$$\begin{aligned} \min_{W \in \mathbb{R}^{p \times p}} \quad & F'(W) \\ \text{subject to} \quad & G(W) \in \mathsf{DAGs} \end{aligned} \iff \begin{aligned} \min_{W \in \mathbb{R}^{p \times p}} \quad & F'(W) \\ \text{subject to} \quad & h(W) = 0, \end{aligned} \tag{3.30}$$

Note that DYNOTEARS does not enforce acyclicity on the autoregressive coefficient matrices A_i. However, this would be possible if we modify the acyclicity constraint function $h(\cdot)$.

Other TEARS. The approach first introduced in NOTEARS has sparked a great interest in such methods, and dozens of papers that mimic or improve on the continuous optimization constraint of NOTEARS have been published. As mentioned, DYNOTEARS extends upon NOTEARS by allowing for linear time-lagged relationships. In addition, there are several other algorithms that improve upon NOTEARS.

Firstly, researchers have sought better functions $h(\cdot)$ that enforce acyclicity. The authors of NO FEARS [83] analyze the acyclicity constraint function $h(\cdot)$ used in NOTEARS and conclude that their method of solving Equation 3.30 using the augmented Lagrangian method does not always converge to a feasible solution. In other words, the solution may not satisfy the constraint $h(W) = 0$. They propose another similar function that enforces acyclicity, and they show that their newly proposed method achieves the accuracy of NOTEARS in the sense that NOTEARS recovers a structure closer to the ground truth.

Second, the authors of NO-BEARS [42] also employ a different approach to verify acyclicity. Instead of investigating the trace exponential of $W \circ W$, the authors of NO BEARS use the spectral radius of $W \circ W$ to enforce acyclicity. Furthermore, they allow for more complex models by modeling polynomial relations between variables instead of linear relationships.

Lastly, the authors of NOTEARS propose an extension of their own approach. Instead of modeling linear relationships, they propose a neural network such that non-linear relationships can be modeled [88]. A similar approach is also employed in [85], but they use a variational autoencoder to model nonlinear relations and their acyclicity constraint is formulated slightly different.

Chapter 4

Permutation-Based Approaches

Although enforcing acyclicity in our model enhances interpretability and reduces the search space of possible graphical models significantly, it is nevertheless quite difficult to enforce. To recall our setting, we are interested in recovering the joint probability distribution of our data $\mathbf{X} \in \mathbb{R}^{T \times p}$, which we can represent as a graphical model that denotes the structure $\mathcal{G}$ of the collected data $\mathbf{X}$.

Let us now assume that our data matrix $\mathbf{X}$ has been generated according to a VAR(1) model as defined in Definition 2.3, which allows us to model linear time-lagged relations between the variables. Given such a data matrix $\mathbf{X}$, we are interested in learning a suitable coefficient matrix W that characterizes this VAR(1) model. However, a crucial remark is that the structure induced by W must be *acyclic* as defined in Definition 2.1. Nevertheless, let us first drop this requirement and see how we would estimate W if we need not enforce acyclicity.

Learning a cyclic matrix W using maximum likelihood estimation. Maximum likelihood estimation is a well-known estimation technique, where we estimate the parameters of a statistical model given a data matrix $\mathbf{X}$ such that the model characterizes by these parameters model was the most probable model to have generated $\mathbf{X}$. In our setting, the statistical model corresponds to a VAR(1) model.

In more mathematical terms, we first need to derive the *likelihood function* which corresponds to the joint density of $\mathbf{X}$ assuming $\mathbf{X}$ has been generated by a VAR(1) model. As the parameters of the VAR(1) model are fully specified by the coefficient matrix W, we can write this joint density as $\mathbb{P}_W(\mathbf{X})$. Under the assumptions of the VAR(1) model, we can write this likelihood function as

$$\mathbb{P}_W(\mathbf{X}) = \mathbb{P}\left(X_{1,\cdot}\right) \prod_{t=2}^{T} \mathbb{P}\left(X_{t,\cdot} \mid X_{t-1,\cdot}\right). \tag{4.1}$$

Now, assuming that all noise components ε_t are independent and follow a Gaussian distribution, and that the initial value X_1 is known, the likelihood function is a product of Gaussian distributions. In particular, let us consider the distribution of the conditional random variable

$$X_{t,\cdot} \mid X_{t-1,\cdot}. \tag{4.2}$$

Since we have assumed that $X_{t,\cdot} = X_{t-1,\cdot}W + \varepsilon_t$, where $\varepsilon_t \sim \mathcal{N}(\mathbf{0}, I_p)$, we can deduce that

$$\mathbb{E}\left[X_{t,\cdot} \mid X_{t-1,\cdot,}\right] = X_{t-1,\cdot}W, \qquad \mathbb{V}\left(X_{t,\cdot} \mid X_{t-1,\cdot,}\right) = I_p. \tag{4.3}$$

As ε_t is a Gaussian random variable, we know that the the conditional random variable in Equation 4.2 follows a Gaussian distribution with aforementioned mean and covariance,

$$X_{t,\cdot} \mid X_{t-1,\cdot} \sim \mathcal{N}\left(X_{t-1,\cdot}W, I_p\right). \tag{4.4}$$

Knowing these conditional distributions $\mathbb{P}\left(X_{t,\cdot}|X_{t-1,\cdot}, W\right)$, the likelihood function $\mathbb{P}_W\left(\mathbf{X}\right)$ is equal to

$$\begin{aligned}\mathbb{P}_W\left(\mathbf{X}\right) &= \prod_{t=2}^{T} \mathbb{P}\left(X_t|X_{t-1,\cdot}\right) \\ &= \prod_{t=2}^{T} \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2} \cdot \left(X_{t,\cdot} - X_{t-1,\cdot}W\right)^T \left(X_{t,\cdot} - X_{t-1,\cdot}W\right)\right). \end{aligned} \tag{4.5}$$

Now, we are interested in finding the matrix W corresponding to the VAR(1) model that is most probable to have generated $\mathbf{X}$, without requiring any acyclicity of the structure for now. Therefore, we are looking for the coefficient matrix $\hat{W}$ that maximizes the likelihood function, or equivalently, the coefficient matrix $\hat{W}$ that maximizes Equation 4.5. This matrix is also known as the maximum likelihood estimator,

$$\hat{W} = \underset{W \in \mathbb{R}^{p \times p}}{\arg\max}\, \mathbb{P}_W\left(\mathbf{X}\right). \tag{4.6}$$

Finding this maximum likelihood estimator $\hat{W}$ can also be achieved by *minimizing* the negative log-likelihood,

$$\hat{W} = \underset{W \in \mathbb{R}^{p \times p}}{\arg\max}\, \mathbb{P}_W\left(\mathbf{X}\right) = \underset{W \in \mathbb{R}^{p \times p}}{\arg\min}\, -\log\left(\mathbb{P}_W\left(\mathbf{X}\right)\right). \tag{4.7}$$

To see this, let us take the logarithm of Equation 4.5 yields and multiply by minus one to obtain

$$\begin{aligned} -\log\left(\mathbb{P}_W\left(\mathbf{X}\right)\right) &= -\log\left(\prod_{t=2}^{T} \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2} \cdot \left(X_{t,\cdot} - X_{t-1,\cdot}W\right)^T \left(X_{t,\cdot} - X_{t-1,\cdot}W\right)\right)\right) \\ &= -\sum_{t=2}^{T} \log\left(\frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2} \cdot \left(X_{t,\cdot} - X_{t-1,\cdot}W\right)^T \left(X_{t,\cdot} - X_{t-1,\cdot}W\right)\right)\right) \\ &= -\sum_{t=2}^{T} \log\left(\frac{1}{\sqrt{2\pi}}\right) - \sum_{t=2}^{T}\left(-\frac{1}{2} \cdot \left(X_{t,\cdot} - X_{t-1,\cdot}W\right)^T \left(X_{t,\cdot} - X_{t-1,\cdot}W\right)\right) \\ &= \frac{1}{2}\sum_{t=2}^{T} \log\left(2\pi\right) + \frac{1}{2}\sum_{t=2}^{T} \left\|X_{t,\cdot} - X_{t-1,\cdot}W\right\|_2^2. \end{aligned} \tag{4.8}$$

Now, as the negative log-likelihood is equal to the squared 2-norm of the difference between $X_{t,\cdot}$ and $X_{t-1,\cdot}W$ for $t = 2, \ldots, T$, multiplied by 0.5 plus some constant shift, the maximum likelihood estimator corresponds to

$$\hat{W} = \underset{W \in \mathbb{R}^{p \times p}}{\arg\max}\, \mathbb{P}_W\left(\mathbf{X}\right) = \underset{W \in \mathbb{R}^{p \times p}}{\arg\min} \sum_{t=2}^{T} \left\|X_{t,\cdot} - X_{t-1,\cdot}W\right\|_2^2. \tag{4.9}$$

Therefore, we see that maximizing the likelihood of $\mathbb{P}_W\left(\mathbf{X}\right)$ is equivalent to minimizing the sum of the squared 2-norm of the difference between $X_{t,\cdot}$ and $X_{t-1,\cdot}W$ for $t = 2, \ldots, T$.

To find the matrix $\hat{W}$ that minimizes this sum in Equation 4.9, let us first define $X \in \mathbb{R}^{T-1 \times p}$ as the first $T-1$ time steps of $\mathbf{X}$, and $Y \in \mathbb{R}^{T-1 \times p}$ as the last $T-1$ time steps of $\mathbf{X}$. Then, we have that

$$\underset{W \in \mathbb{R}^{p \times p}}{\arg\min} \sum_{t=2}^{T} \left\|X_{t,\cdot} - X_{t-1,\cdot}W\right\|_2^2 = \underset{W \in \mathbb{R}^{p \times p}}{\arg\min}\, \left\|Y - XW\right\|_F^2, \tag{4.10}$$

where $\left\|\cdot\right|_F$ denotes the Frobenius norm. We can compute the matrix derivative of Equation 4.10 with respect to W and set it to zero. As $\nabla \left\|X\right\|_F^2 = 2X$, we have that

$$\nabla_W \left\|Y - XW\right\|_F^2 = -2X^T\left(Y - XW\right) = 2X^TXW - 2X^TY, \tag{4.11}$$

where X^T denotes the transpose of X. Now, the matrix derivative in Equation 4.11 is equal to the zero matrix if and only if

$$X^T XW = X^T Y \Rightarrow W = \left(X^T X\right)^{-1} X^T Y. \tag{4.12}$$

Therefore, we know that the maximum likelihood estimator $\hat{W}$ coincides with the so-called *ordinary least squares* estimator, which is given by

$$\hat{W} = \left(X^T X\right)^{-1} X^T Y. \tag{4.13}$$

Such a maximum likelihood estimation technique would be a sensible approach to find the most suitable coefficient matrix W for our VAR(1) model $\mathbf{X}$. Such a maximum likelihood estimation technique or equivalently, an ordinary least squares estimation technique, is one of the most common techniques for estimating the coefficient matrix of a VAR(1) model [34].

However, such a maximum likelihood estimate does not take acyclicity into account. However, we shall see that there is an interesting decomposition that allows us to employ maximum likelihood estimation, while ensuring that the inferred structure is acyclic. As the chapter suggests, this decomposition is based on the ordering or *permutation* of the p variables. This notion will be introduced in the next paragraph.

Permutations in a structure. Searching over this space of directed acyclic graphs is quite difficult. An interesting approach to enforce this acyclicity is by using a *permutation* π. A permutation π of the p variables is represented by a function that assigns each variable X_i a unique index $\pi(i)$ in the ordering. We say that a variable X_i *precedes* a variable X_j in the permutation π if and only if $\pi(i) < \pi(j)$. Enforcing such an ordering can help in enforcing acyclicity, as we see in Proposition 4.1.

Proposition 4.1 (Squires et al., 2019)

A structure $\mathcal{G}$ of the variables is acyclic if and only if there exists a permutation π of the variables such that an arc (X_i, X_j) exists in $\mathcal{G}$ only if X_i precedes X_j in the permutation, that is, $\pi(i) < \pi(j)$.

$$\mathcal{G} \text{ is acyclic } \iff X_i \text{ precedes } X_j \text{ in } \pi \text{ for all arcs } (i,j) \text{ in } \mathcal{G}. \tag{4.14}$$

Given a directed acyclic graph $\mathcal{G}$, we can find at least one but possibly several permutations that are compatible with $\mathcal{G}$. Conversely, if we can find a permutation π such that, for all arcs (i, j) in $\mathcal{G}$, we have that $\pi(i) < \pi(j)$, then we know that $\mathcal{G}$ is a directed acyclic graph. Additionally, if we know that π is compatible with some acyclic graph $\mathcal{G}$, then we still have to decide which of the possible arcs do actually exist

We see that we can enforce acyclicity by enforcing a permutation of the variables. Consequently, when we only allow directed relationships that are compatible with the permutation π, the structure $\mathcal{G}$ will be acyclic. In other words, we only allow edges from a variable X_i to a variable X_j only if the variable X_i precedes the variable X_j in the permutation π.

We can also write the permutation π as a permutation matrix P, where $p_{ij} = 1$ if $j = \pi(i)$ and $p_{ij} = 0$ otherwise. In other words, the entry in the ith row and the jth column is equal to one if and only if the ith variable comes in place j in the ordering. For example, the permutation where we place the third variable first in the ordering, $\pi = (3, 1, 2)$, can be written as

$$\pi = (3,1,2) \iff P = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0. \end{pmatrix}. \tag{4.15}$$

As such approaches require an explicit permutation of the variables, we denote these types of approaches as *permutation-based* approaches.

Permutations in a linear model. To cast these permutation-based approaches in our linear setting, we again consider the VAR(1) model as per Definition 2.3. To reiterate, we assume that our data matrix $\mathbf{X} \in \mathbb{R}^{T \times p}$ has been generated as

$$X_{t,\cdot} = X_{t-1,\cdot}W + \varepsilon_t, \qquad \varepsilon_t \sim \mathcal{N}(\mathbf{0}, I_p). \tag{4.16}$$

The model defined by Equation 4.16 is parametrized by a coefficient matrix W. We require the graph induced by W, which we denote by $G(W)$, to be acyclic. Enforcing acyclicity of the graph induced by W can be done quite easily using *permutation matrices*. The permutation matrix P enforces an ordering or permutation π of the variables. We can use such a permutation matrix P to characterize acyclicity of $G(W)$ similar to Proposition 4.1. This statement is given in Proposition 4.2.

Proposition 4.2 (Bühlmann, 2013)

A matrix W is compatible with a direct acyclic graph $G(W)$ if and only if there exists a permutation matrix P and an upper triangular matrix U such that

$$W = P^T U P. \tag{4.17}$$

Here the permutation matrix P corresponds to a permutation π such that for all arcs (X_i, X_j) in $G(W)$, X_i precedes X_j in π.

From Proposition 4.2, we see that we can decompose our coefficient matrix W into two parts, a permutation matrix P and an upper triangular matrix U. This ordering π implies that if X_i precedes X_j, then we can only have the arc $X_i \to X_j$ and not the arc $X_i \leftarrow X_j$.

The upper triangular matrix U represents the arcs in our graphical model. The upper triangular part of the matrix can be non-zero, the diagonal included. However, all entries in the lower triangular part must be equal to zero. Therefore, we see that the entry u_{12} corresponds to the weight of the arc from $X_{\pi(1)}$ to $X_{\pi(2)}$, where $X_{\pi(i)}$ represents the ith variable in the ordering. As $u_{ij} > 0$ only if $i \geq j$, we have that $u_{ij} > 0$ only if variable X_j does not precede variable X_i. Therefore, the corresponding upper triangular matrix U is consistent with the permutation defined by the permutation matrix P.

Outline of this chapter. So far in this chapter, we have explained how we can derive the *maximum likelihood estimator* of W given our data matrix $\mathbf{X}$. However, acyclicity of the structure is not enforced. Therefore, we have introduced the notion of a *permutation* π of our p variables or equivalently, a permutation matrix P that specifies an ordering of our p variables. Given such a permutation matrix P, we can easily enforce acyclicity by only allowing arcs from X_i to X_j if X_i precedes X_j in the ordering. For this, Proposition 4.2 provides a useful decomposition of the coefficient matrix W that enforces acyclicity.

The next part of this chapter will be devoted to maximum likelihood estimation methods that utilize the decomposition of the coefficient matrix W into a permutation matrix P and an upper triangular matrix U. Using this decomposition, we implicitly enforce acyclicity of $G(W)$, the structure induced by W. We call these types of approaches *permutation-based* approaches, as their success is based on the use of permutation matrices. In Section 4.1, we will first describe a naive approach that iterates over all possible permutation matrices and learns a suitable upper triangular matrix U for each permutation using maximum likelihood estimation. However, this exhaustive approach is infeasible for more than a dozen variables.

Therefore, we need a way to trade off the running time of our algorithm against the performance of our algorithm. Therefore, we will discuss a method to find a suboptimal yet suitable ordering by using a random walk on the space of permutation matrices in Section 4.2. We improve on the random walk by creating a more guided search method using the Metropolis-Hastings algorithm in Section 4.3.

4.1 Exhaustive permutation search

Instead of searching for a suitable acyclic structure induced by W, we can now iterate over all sensible decompositions (P, U) of W. Luckily, finding a suitable matrix U for a given P is relatively simple. We will first discuss how to find such a suitable matrix U given a permutation matrix P. For a suitable matrix U, we will consider the maximum likelihood estimator of U. We have named the algorithm that finds the maximum likelihood estimator of U given a permutation matrix P as `Order-OLS`, since it finds the most likely matrix U given a permutation or ordering of the variables. Let us first discuss how to compute this maximum likelihood estimator.

Order-OLS. Once we are given a permutation matrix P, there is a closed-form solution of the maximum likelihood estimate of the upper triangular matrix U that adheres to this permutation P. Again, let X represent the first $T-1$ times steps of $\mathbf{X}$, and Y the last $T-1$ time steps of $\mathbf{X}$. Then, recall that the negative log-likelihood was equal to

$$-\log\left(\mathbb{P}_W\left(\mathbf{X}\right)\right) = \frac{1}{2}\sum_{t=2}^{T}\log\left(2\pi\right) + \frac{1}{2}\sum_{t=2}^{T}\left\|Y - XW\right\|_F^2. \tag{4.18}$$

Given a permutation matrix P, the log-likelihood corresponding to U is given by

$$-\log\left(\mathbb{P}_U\left(\mathbf{X}; P\right)\right) = \frac{1}{2}\sum_{t=2}^{T}\log\left(2\pi\right) + \frac{1}{2}\sum_{t=2}^{T}\left\|Y - XP^TUP\right\|_F^2, \tag{4.19}$$

which is the same as stating that

$$-\log\left(\mathbb{P}_U\left(\mathbf{X}; P\right)\right) = \frac{1}{2}\sum_{t=2}^{T}\log\left(2\pi\right) + \frac{1}{2}\sum_{t=2}^{T}\left\|YP^T - XP^TU\right\|_F^2, \tag{4.20}$$

where we have reordered the p variables rather than the rows and columns U. If we were allowed to estimate all coefficients of U, the maximum likelihood estimate would be

$$\hat{U} = \left(X_P^TX_P\right)^{-1}X_P^TY_P, \tag{4.21}$$

where $X_P = XP^T$ and $Y_P = YP^T$ represent the permuted data matrices.

However, we we are only allowed to estimate the upper triangular part of the matrix U when maximizing the likelihood. When estimating the first column of U, we are only allowed to estimate the first coefficient u_{11}, corresponding to regressing the first variable in X_P on the first variable in Y_P. For the ith column of U, we can only estimate the first i coefficients, corresponding to the influence of the first i variables in X_P, denoted by $X_{P,\cdot,1:i} \in \mathbb{R}^{T-1\times i}$ on the ith variable in Y_P, denoted by $Y_{P,\cdot,i} \in \mathbb{R}^{T-1}$. Then, the maximum likelihood estimates of the first i coefficients in the ith column of $\hat{U}$ correspond to

$$\hat{U}_{1:i,i} = \left(X_{P,\cdot,1:i}^TX_{P,\cdot,1:i}\right)^{-1}X_{P,\cdot,1:i}^TY_{P,\cdot,i}. \tag{4.22}$$

This maximum likelihood estimate coincides with the *ordinary least squares* solution that adheres to the permutation or order of the variables, hence the name `Order-OLS`. Note that as we estimate i coefficients for the ith column, the coefficient matrix $\hat{U}$ will have $\sum_{i=1}^{p} i = p(p+1)/2$ nonzero coefficients.

In less mathematical terms, we regress the first i time-lagged variables in the permutation on the ith variable in the permutation. The corresponding estimates form the first i coefficients in the ith column of the upper triangular matrix U. Doing this for all p columns of U yields the maximum likelihood estimator of U while adhering to the permutation induced by P. The pseudocode for `Order-OLS` has been given in Algorithm 4.1.

Algorithm 4.1: `Order-OLS`$(\mathbf{X}, P)$

Input: Data matrix $\mathbf{X} \in \mathbb{R}^{T \times p}$, Permutation matrix P.
Output: Coefficient matrix W that maximizes the likelihood of $\mathbf{X}$, while respecting the permutation induced by P.

$\mathbf{X}_P \leftarrow \mathbf{X}P^T$ ▷ permute columns of $\mathbf{X}$
$U \leftarrow \mathbf{O}_{p \times p}$ ▷ initialize U as $p \times p$ zero matrix

$X \leftarrow \mathbf{X}_P[1 : T-1, \cdot]$ ▷ first $T-1$ time steps of $\mathbf{X}_P$
$Y \leftarrow \mathbf{X}_P[2 : T, \cdot]$ ▷ last $T-1$ time steps of $\mathbf{X}_P$

for $i = 1$ until p **do**
  $X_i \leftarrow X[\cdot, 1 : i]$ ▷ first i variables in P
  $Y_i \leftarrow Y[\cdot, i]$ ▷ ith variable in P
  $U[1 : i, i] \leftarrow \left(X_i^T X_i\right)^{-1} X_i^T Y_i$ ▷ OLS estimate for column i of U
end for

$W \leftarrow P^T U P$ ▷ transpose rows and columns of U
return W

Example 4.1 `Order-OLS` with three variables.

Let us give an example of how `Order-OLS` works. Suppose we have three variables, and we want to find the coefficient matrix that maximizes the likelihood given our data matrix $\mathbf{X}$, while respecting the ordering $\pi = (3, 1, 2)$, corresponding to the permutation matrix

$$P = \begin{pmatrix} e_3 \\ e_1 \\ e_2 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}. \tag{4.23}$$

To estimate the first column of U, $X_{P,\cdot,1}$ corresponding to the first $T-1$ values of the first variable in the permutation, so $\mathbf{X}_{1:T-1,3}$. Furthermore, $Y_{P,\cdot,1}$ corresponds to the last $T-1$ values in the first variable of the permutation, so $\mathbf{X}_{2:T,3}$. Therefore, the first column of U contains only one estimated parameter, u_{11}, for which the maximum likelihood estimator is

$$u_{11} = \left(X_{P,\cdot,1}^T X_{P,\cdot,1}\right)^{-1} X_{P,\cdot,1}^T Y_{P,\cdot,1}. \tag{4.24}$$

For the second column of U, we can do the same. We will use the time-lagged values of X_3 and X_1 to predict the values of X_1, yielding the two parameters

$$\begin{pmatrix} u_{12} \\ u_{22} \end{pmatrix} = \left(X_{P,\cdot,1:2}^T X_{P,\cdot,1:2}\right)^{-1} X_{P,\cdot,1:2}^T Y_{P,\cdot,2}, \tag{4.25}$$

where $X_{P,\cdot,1:2} = (\mathbf{X}_{1:T-1,3}, \mathbf{X}_{1:T-1,1})$, and $Y_{P,\cdot,2} = \mathbf{X}_{2:T,1}$.

For the third and final column of U, we can estimate all three parameters. The three parameters that maximize the likelihood given $\mathbf{X}$ are

$$\begin{pmatrix} u_{13} \\ u_{23} \\ u_{33} \end{pmatrix} = \left(X_{P,\cdot,1:3}^T X_{P,\cdot,1:3}\right)^{-1} X_{P,\cdot,1:3}^T Y_{P,\cdot,3}, \tag{4.26}$$

where $X_{P,\cdot,1:3} = (\mathbf{X}_{1:T-1,3}, \mathbf{X}_{1:T-1,1}, \mathbf{X}_{1:T-1,2})$, and $Y_{P,\cdot,3} = \mathbf{X}_{2:T,3}$. This yields the matrix U that maximizes the likelihood given $\mathbf{X}$ while adhering to the permutation induced by P.

As a final procedure, we permute the rows and columns of U,

$$W = P^T U P. \tag{4.27}$$

This acyclic matrix W maximizes the likelihood given $\mathbf{X}$ while respecting the ordering induced by P. Equivalently, this matrix W minimizes the sum of the squared 2-norm of the differences between $X_{t,\cdot}$ and $X_{t-1,\cdot}$, as we have shown in Equation 4.18.

Exhaustive Search. We have a method to find the optimal U given a permutation matrix P. We now perform an exhaustive search over all possible permutation matrices $P \in \mathcal{P}_{perm}$, where $\mathcal{P}_{perm}$ is the space of all permutation matrices. Trying all permutation matrices, we can simply select the permutation matrix P that yields the acyclic coefficient matrix W that maximizes the likelihood. In Equation 4.9, we have seen that this is equal to minimizing the sum of the squared two norms of the errors. Equivalently, we can minimize the mean squared error, which is simply the average of the squared two norms,

$$\texttt{MSE}(W) = \frac{1}{T-1} \sum_{t=2}^{T} \|X_{t,\cdot} - X_{t-1,\cdot} W\|_2^2 . \tag{4.28}$$

To find the most suitable matrix W that respects a permutation matrix P, we use the `Order-OLS` algorithm as described in Algorithm 4.1. The exhaustive procedure is given in Algorithm 4.2.

Algorithm 4.2: Exhaustive-Search($\mathbf{X}$)

Input: Data matrix $\mathbf{X} \in \mathbb{R}^{T \times p}$.
Output: Coefficient matrix W that maximizes the likelihood of the $\mathbf{X}$ while remaining acyclic, $W^* = \underset{W \in \mathbb{R}^{p \times p}}{\arg\max}\, \mathbb{P}_W(\mathbf{X})$ such that $G(W) \in \texttt{DAGs}$.

```
1:  W_best ← I
2:  MSE_best ← ∞
3:
4:  for every permutation matrix P do
5:      W ← Order-OLS(X, P)
6:      if MSE(W) < MSE_best then
7:          MSE_best ← MSE(W)
8:          W_best ← W
9:      end if
10: end for
11:
12: return W_best
```

Search space of permutation matrices. An issue of the exhaustive search described in Algorithm 4.2 can be seen in line 4, where we iterate over the complete search space of permutation matrices. When we have p variables, a total of $p!$ possible permutations exists, which is not tractable when p is large. Even for $p = 10$, a total of $10! \approx 3.6$ million upper triangular matrices U need to be estimated, which implies that a total of 36 million columns need to be estimated, corresponding to 36 million regressions.

Nevertheless, the search space has been greatly reduced. Let $G(p)$ be the number of possible directed acyclic graphs with p nodes. The value of $G(p)$ is given by the recurrence

$$G(p) = \sum_{k=1}^{n} (-1)^{k+1} \binom{p}{k} 2^{k(p-k)} G(p-k), \tag{4.29}$$

which was first given by Robinson in [61]. Note that our definition of a directed acyclic graph allows for self-loops, so we expect the number of graphs that we consider to be directed acyclic graphs to be slightly larger. Even for just ten nodes, the number of possible directed acyclic graphs is equal to $G(10) \approx 4.2 \cdot 10^{18}$. However, there are "only" $10! \approx 3.6 \cdot 10^6$ 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. Therefore, the search space has been reduces dramatically, approximately by a factor 10^{12}.

The exhaustive algorithm has been implemented in Python and unfortunately does not support more than ten variables. Nevertheless, investigating this problem in relatively small dimensions provides an interesting starting point which can lead to insightful discoveries. When applying the algorithm to eight variables and one thousand samples per variable, the algorithm takes approximately one minute to return a matrix W.

As we try all permutation matrices P, the exhaustive approach is guaranteed to return the coefficient matrix W that maximizes the likelihood $\mathbb{P}_W(\mathbf{X})$, while ensuring $G(W)$ is acyclic. This showcases the trade-off between the running time of our algorithm and the quality of its output. The algorithm outputs the acyclic coefficient matrix W that maximizes the likelihood at the cost of having an exponential running time.

Number of compatible permutations. When the true coefficient matrix is sparse, we often do not need to do an exhaustive search over all permutation matrices to find the optimal solution. We only need to find a permutation compatible with the graph $G(W)$ to estimate all non-zero coefficients in W. For example, if two subsequent variables do not depend on each other, that is, $w_{ij} = w_{ji} = 0$ for $j \neq i$, then it does not matter which variable precedes the other in the permutation. Hence, there could be multiple permutations matrices P such that $P^T U P = W$. An example where we count the number of suitable permutations is discussed in Example 4.2.

Example 4.2 Counting the total number of suitable orderings.

Let us give an example of what we mean when we say that multiple permutation matrices will yield the optimal solution. Suppose that our data is generated according to a VAR(1) model with five variables and coefficient matrix

$$W = \begin{pmatrix} 0.5 & 0.5 & 0.0 & 0.0 & 0.5 \\ 0.0 & 0.5 & 0.0 & 0.5 & 0.0 \\ 0.0 & 0.0 & 0.5 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.5 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.5 \end{pmatrix}.$$

For ease of notation, our matrix W is already an upper triangular matrix. As the value of the coefficients is not important here, all coefficients simply have a value of 0.5.

Now, the ordering $\pi = (1, 2, 3, 4, 5)$ is a permutation such that we can estimate all non-zero coefficients of W, as $W = I^T U I$ for some upper triangular matrix U. Hence, using `Order-OLS` on $\mathbf{X}$ with permutation I allows us to estimate all true coefficients in W.

However, this is not the only possible permutation. Our third variable only depends on itself, X_3 can be at any index in the permutation π and our matrix W remains upper triangular. This increases the number of compatible orderings to 5. Furthermore, since the fourth and fifth variable do not depend on each other, we can interchange them in any ordering. Lastly, since the fifth variable only depends on the first variable, we only need to ensure that the first variable precedes the fifth variable in the ordering.

Out of the $5! = 120$ permutations, there are a total of 20 permutation matrices P such that $W = P^T U P$ for some upper triangular matrix U. Therefore, we often need not check *all* possible permutations, especially when the coefficient matrix W is sparse.

The question that arises now is the following: "Given a matrix W, how many different permutations of its rows and columns are there that yield an upper triangular matrix?" In other words, how many permutations are there such that `Order-OLS` can estimate all non-zero coefficients of W? Unfortunately, this is a difficult question to answer. In fact, this problem is #P-Complete [7], meaning that this problem is even harder than an NP-Complete problem. Therefore, the only statement we can make that for a sparse W, there are most likely quite some valid orderings of the variables. How many there are is, unfortunately, often too difficult to say.

As a sidenote, a sparse matrix does not always imply that there exist many possible permutations. Consider the matrix with non-zero coefficients on the diagonal above the main diagonal, which is also called the *superdiagonal*:

$$W = \begin{pmatrix} 0.0 & 0.5 & 0.0 & \cdots & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.5 & \cdots & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & \cdots & 0.0 & 0.0 & 0.0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0.0 & 0.0 & 0.0 & \cdots & 0.0 & 0.5 & 0.0 \\ 0.0 & 0.0 & 0.0 & \cdots & 0.0 & 0.0 & 0.5 \\ 0.0 & 0.0 & 0.0 & \cdots & 0.0 & 0.0 & 0.0 \end{pmatrix}.$$

As each variable depends on the preceding variable, we cannot have any compatible ordering other than $P = I$. Therefore, the number of suitable permutations is equal to 1, although we have a sparse matrix with only $p - 1$ non-zero coefficients.

Let us now discuss the exhaustive algorithm step-by-step. For this, we will be considering a simple three-dimensional time series in Example 4.3.

Example 4.3 Applying the exhaustive method on a three-dimensional data matrix.

Consider the three-dimensional data shown in Figure 4.1. This data has been generated using the formula

$$X_{t,\cdot} = X_{t,\cdot} W + \varepsilon_t, \tag{4.30}$$

where

$$W = \begin{pmatrix} 0.8 & 0.0 & 0.0 \\ 0.5 & 0.8 & -0.5 \\ 0.0 & 0.0 & 0.8 \end{pmatrix}, \qquad \varepsilon_t \sim \mathcal{N}(\mathbf{0}, I_3).$$

We define $X_{1,\cdot} = \varepsilon_1$ so that $X_{t,\cdot}, t \geq 2$ follows from Equation 4.30. We simulate for a total of $T = 100$ time steps. The corresponding time series has been visualized in Figure 4.1.

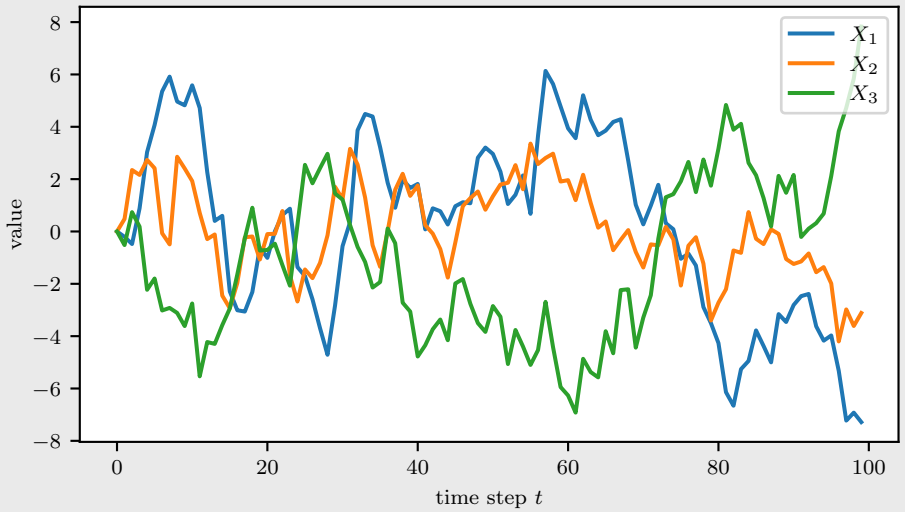


Figure 4.1: Visualization of the three variables X_1, X_2, X_3 of Example 4.3.

Our exhaustive search as described in Algorithm 4.2 will simply iterate over all possible permutation matrices P, of which there are a total of $3! = 6$. For each permutation matrix, we calculate the upper triangular matrix U that maximizes the likelihood $\mathbb{P}_U(\mathbf{X}; P)$ while respecting the permutation matrix P using `Order-OLS`$(\mathbf{X}, P)$ described in Algorithm 4.1.

Let us first consider where we use the identity matrix as our permutation $P_1 = I$, and hence, we can only estimate coefficients from the upper triangular part of W. Using `Order-OLS`, we get that

$$W_1 = \begin{pmatrix} 0.96 & 0.01 & -0.08 \\ 0.0 & 0.80 & -0.42 \\ 0.0 & 0.0 & 0.77 \end{pmatrix},$$

with an accompanying mean squared error of 3.421. We see that the relation $X_2 \to X_1$ cannot be captured appropriately using this ordering, as $X_2 \to X_1$ does not respect the ordering given by P_1.

Now, let us consider a permutation P such that `Order-OLS` can estimate all true coefficients of W. Two permutations satisfy this condition, which are

$$P_2 = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \qquad P_3 = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}.$$

Applying `Order-OLS` on these two permutation matrices P_2 and P_3 yield the respective coefficient matrices

$$W_2 = \begin{pmatrix} 0.79 & 0.00 & -0.08 \\ 0.55 & 0.81 & -0.42 \\ 0.00 & 0.00 & 0.77 \end{pmatrix}, \qquad W_3 = \begin{pmatrix} 0.77 & 0.00 & 0.00 \\ 0.55 & 0.81 & -0.48 \\ -0.02 & 0.00 & 0.83 \end{pmatrix},$$

with accompanying mean squared errors of 2.906 and 2.930 respectively. As we can see, both W_2 and W_3 estimate all non-zero coefficients in W and both achieve a lower mean squared error than W_1, where we could not estimate w_{12}. However, one achieves a slightly lower mean squared error. As our data is corrupted with noise, non-true coefficients are also estimated with small coefficients that correlate with the noise in the data. Apparently, the spurious correlation $X_3 \to X_1$, corresponding to coefficient w_{31} yields slightly better predictive performance than the spurious correlation $X_1 \to X_3$. This is not very problematic, but we would like our method to generalize well to similar data matrices X. Therefore, we would like to *regularize* our solution matrix W_2 or W_3 such that only the true coefficients are estimated. We will discuss a naive yet effective approach at the end of this chapter in Section 4.4.

For the time being, we do not take this regularization into account, and are only interested in the matrix W_i that achieves the largest likelihood, or equivalently, minimizes the mean squared error. Table 4.1 shows the resulting mean squared errors of all $3! = 6$ permutation matrices.

Table 4.1: Table showing the mean squared errors of all six possible permutations for Example 4.3. A lower mean squared error implies a larger likelihood, indicating a better model fit.

Permutation	(1, 2, 3)	(1, 3, 2)	(2, 1, 3)	(2, 3, 1)	(3, 1, 2)	(3, 2, 1)
Mean Squared Error	3.421	3.724	2.906	2.930	3.927	3.413

Looking at Table 4.1, we indeed see that the lowest mean squared error is achieved by respecting the ordering P_2. Using this exhaustive search, we have found the matrix W_2 that maximizes the likelihood of the data while remaining acyclic.

There are some techniques to improve the computational efficiency of the exhaustive search. We can *cache* certain computations such that these need not be redone. For example, suppose we first compute the upper triangular matrix U respecting the permutation $\pi_1 = (1, 2, 3, \ldots, p-1, p)$, resulting in coefficient matrix W_{π_1}. Now, suppose we want to compute the upper triangular matrix U respecting the slightly different permutation matrix $\pi_2 = (1, 2, 3, \ldots, p, p-1)$. Instead of completely recomputing the upper triangular matrix U, we only need to re-estimate the final two columns, as the ordering did not change for the first $p-2$ variables. If we iterated over all permutation matrices in a smart way, we can re-use a large portion of the upper triangular matrices U, thereby greatly reducing the computation time.

Nevertheless, the exponential nature of the exhaustive search makes this technique unsuitable for more than a dozen variables. Therefore, let us now consider combinatorial approaches that do not require an exponential amount of running time, thereby sacrificing the quality of the returned matrix W.

4.2 Random Walk

In Section 4.1, we have seen that we can find our optimal matrix W by trying all possible orderings of our p variables. We refer to the optimal matrix W as the matrix W that maximizes the likelihood of our data matrix $\mathbf{X}$ such that $G(W)$ is acyclic. Unfortunately, there are $p!$ of those permutations, where p is the number of variables or equivalently, the number of rows or columns in W. Therefore, we cannot expect to find the optimal permutation within a reasonable amount of time.

Nevertheless, we can try to find a reasonably good permutation without checking all the possible permutation matrices. For this, we can do a *random walk* through the space of permutation matrices $\mathcal{P}_{\text{perm}}$. For each permutation matrix P we visit, we apply `Order-OLS`$(\mathbf{X}, P)$ to see whether this permutation matrix P is a good fit for our data $\mathbf{X}$. We remember the best permutation matrix P, accompanied by the most fitting upper triangular matrix U. After having tried enough permutation matrices P, we terminate the random walk. For this, we can either fix the number of permutations we visit to a certain number N or alternatively, we set the maximum time that can elapse during our random walk to a fixed time of $t_{\max}$.

For such a random walk, we require two components. First of all, we require a method to determine to which state we will transition next. Secondly, we require an initial state from where we start the random walk. These two components will be discussed in the following two paragraphs.

Transitions probabilities. A random walk is a naive method to traverse any discrete search space. From a given state, we transition to another state according to some transition probability. In the setting of permutation matrices, we transition from our current permutation matrix P_i to the next matrix P_{i+1} according to some transition probability. In a random walk, the next step, or in our setting the next permutation matrix, is often chosen uniformly at random from a set of candidates. That is, the next permutation matrix P_{i+1} is chosen from some distribution, conditioned on P_i,

$$\mathbb{P}\left(P_{i+1} = P' \mid P_i = P\right). \tag{4.31}$$

We will propose two different approaches to define the set of candidates.

First of all, we can just simply pick a permutation matrix at random from the $p!$ possible candidates. Note that this also includes our current state, from which we do not gain any information by revisiting this permutation. Therefore, it may be wise to exclude already visited permutation matrices from our set of matrices. However, when p is large, the probability of visiting the exact same permutation matrix is quite small, and remembering all visited permutation matrices may take quite some storage without improving the algorithm significantly. So, a suitable transition probability would be

$$\mathbb{P}\left(P_{i+1} = P' \mid P_i = P\right) = \frac{1}{p!} \text{ for all permutation matrices } P' \in \mathcal{P}_{\text{perm}}. \tag{4.32}$$

where $\mathcal{P}_{\text{perm}}$ is the set of all possible permutation matrices on p variables. Note that the conditioning in Equation 4.32 is unnecessary, as the transition probability does not depend on the current state.

As a second selection method, we can define the set of candidates as the set of permutation matrices that are "adjacent" to our current permutation matrix P. We define the set of permutation matrix that are "adjacent" to P as $\mathcal{P}_{\text{adj}(P)}$, the set of permutation matrices that can be reached by swapping just two variables. In mathematical notation, we have that

$$\mathcal{P}_{\text{adj}(P)} = \{P' \mid \text{swap row } i \text{ and } j \text{ of } P,\, i = 1, \dots p-1,\, j = i+1, \dots, p.\}. \tag{4.33}$$

In total, there are $\binom{p}{2} = p(p+1)/2$ of such permutation matrices P' for any permutation matrix P. We pick one permutation matrix P' uniformly at random from this set, that is, with probability $2/(p(p+1))$. Our transition probabilities would then be

$$\mathbb{P}\left(P_{i+1} = P' \mid P_i = P\right) = \begin{cases} \frac{2}{p(p+1)} & \text{for all permutation matrices } P' \in \mathcal{P}_{\text{adj}(P)}, \\ 0 & \text{otherwise.} \end{cases}. \tag{4.34}$$

where $\mathcal{P}_{\text{adj}(P)}$ is the set of all permutation matrices on p variables that are one swap of two variables away from P. Note that there are more choices for the transition probabilities. For example, we can consider only swapping variables adjacent in the ordering, resulting in $p-1$ permutation matrices to choose from. Nevertheless, we will only stick to these two for now.

Initial state P_0. Another important choice is the starting permutation of the random walk. The random walk must start at some permutation matrix P_0, from which we will sample new permutation matrices from our conditional distribution $\mathbb{P}\left(P_{i+1} \mid P_i\right)$. Note that for our first choice in Equation 4.32, the initial permutation matrix, or in fact the current state P_i, does not matter at all, as all $p!$ permutation matrices are equally likely to be the next permutation matrix.

For the second case, given in Equation 4.34, where we only consider transitioning to "adjacent" permutation matrices P', the initial starting point P_0 greatly determines which permutation matrices we will visit. For example, suppose that we start at $P_0 = I_p$. A large number of transitions is required to reach the opposite permutation, so the backwards identity matrix

$$J_p = \begin{pmatrix} 0 & 0 & \cdots & 0 & 1 \\ 0 & 0 & \cdots & 1 & 0 \\ \vdots & \vdots & \iddots & \vdots & \vdots \\ 0 & 1 & \cdots & 0 & 0 \\ 1 & 0 & \cdots & 0 & 0 \end{pmatrix}.$$

In fact, we require approximately $p!$ transitions to reach J_p if our initial state is I_p. Therefore, starting with a permutation "far" from a suitable permutation could result in a poorly performing random walk.

Having explained the two key components of the random walk, let us consider the pseudocode for such a method in the next paragraph.

The Random Walk Algorithm. Both versions of the random walk are given in Algorithm 4.3. Note that they differ in their sampling distribution $\mathbb{P}\left(P_{i+1} \mid P_i\right)$. For the first version, the completely random walk, we use Equation 4.32 to determine the next permutation P_{i+1}. For the second version, where we only consider permutation matrices P' adjacent to P, we use Equations 4.33 and 4.34.

Apart from the data matrix $\mathbf{X}$, we also require an initial permutation P_0 and a termination condition, for example the maximum number of steps N. As an initial condition, we can simply pick the identity matrix I_p, and for the N any integer value we prefer. Picking N large will on average result in a more suitable matrix W, but the algorithm also requires more time.

Algorithm 4.3: Random walk

Input: Data matrix $\mathbf{X}$, Initial permutation P_0, maximum number of steps N.
Output: The acyclic coefficient matrix $W \in \mathbb{R}^{p \times p}$ that achieves the highest likelihood found using the random walk.

```
W_best ← P_0
MSE_best ← MSE(P_0)
for n = 1 until N do
    P_n ← P_next(P_{n−1})
    W ← Order-OLS(P_n)
    if MSE(W) < MSE_best then
        MSE_best ← MSE(W)
        W_best ← W
    end if
end for
return W_best
```

Example on ten variables. Let us consider an example on ten variables, resulting in $10! \approx 3.6$ million permutation matrices. Such a large search space cannot quickly be exhausted, so let us see whether the random walk can find a suitable acyclic matrix W.

Example 4.4 Random Walk on a 10-dimensional example.

We consider the data matrix $\mathbf{X} \in \mathbb{R}^{100 \times 10}$ visualized in Figure 4.2, generated by a VAR(1) model as defined in Definition 2.3 on ten variables and one hundred samples. We have constructed a sparse acyclic coefficient matrix W, which has 25 off-diagonal non-zero coefficients with value 0.5. All p variables have an autoregressive coefficient uniformly from the range $[0.6, 0.8]$.

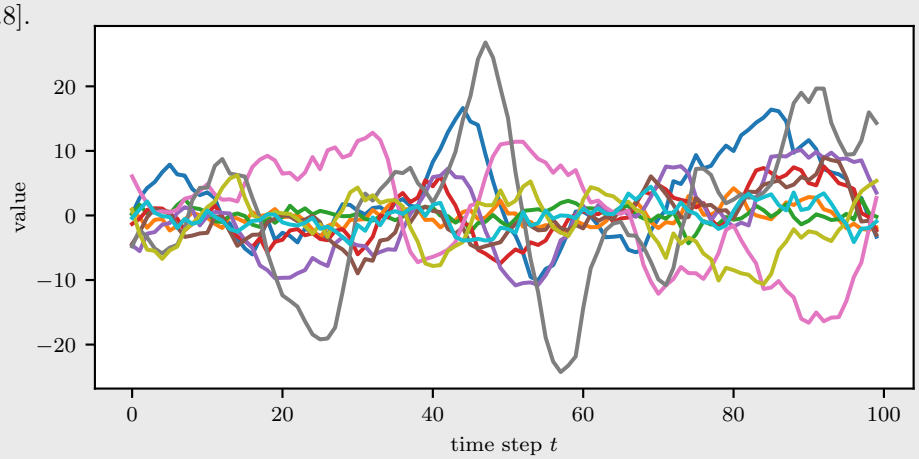


Figure 4.2: Visualization of the data matrix $\mathbf{X}$ of Example 4.4.

We will explore just 1% of the search space with no information on the correct permutation matrix. Therefore, we will use Algorithm 4.3 with $P_0 = I_{10}$ and $N = p!/100 \approx 3.6 \cdot 10^4$, yielding a running time of approximately one minute rather than two hours for the exhaustive approach. We will also run the exhaustive algorithm to obtain the acyclic matrix W^* that maximizes the likelihood. We will use both methods of sampling the next permutation matrix, and the returned matrices W_1 and W_2 achieve a mean squared error of

$$\texttt{MSE}(W^*) = 9.426, \qquad \texttt{MSE}(W_1) = 10.567, \qquad \texttt{MSE}(W_2) = 10.245. \tag{4.35}$$

Both random walks achieve a performance quite close to the exhaustive search, despite having only explored roughly 1% of the search space.

4.3 Using the Metropolis-Hastings Algorithm

The performance of the random walk discussed in Section 4.2 is rather encouraging, especially since the random walk blindly transitions from one permutation matrix to another, regardless of its suitability. We can improve on the random walk by devising a more guided search, where we will not transition to a poorly fitting permutation matrix in an uninformative manner.

In this section, we will apply the Metropolis-Hastings approach. The approach was named after N. Metropolis, one of the authors that introduced this method in 1953 [48]. The method was further investigated by W. K. Hastings in 1970 [31], from which the conjunction originates. It is a *Markov-Chain Monte Carlo* method, where we sample the next permutation matrix based on the transition probabilities of the corresponding Markov Chain. Such methods have shown to be successful in efficiently traversing a complex search space. For example, in [11], the objective was to find a suitable hierarchical clustering of variables. The authors reformulated such a hierarchical clustering to a binary tree, and subsequently employed a Metropolis-Hastings approach to search for the tree T that maximizes the likelihood profile $\mathcal{L}(\mathbf{x} \mid T)$.

Relation to the random walk. The random walk can also be seen as a Markov Chain Monte Carlo method, albeit quite a simple one. The probability of transitioning from a permutation matrix P to a permutation matrix P' solely depends on P, and not on how well P' fits the data $\mathbf{X}$. The Metropolis-Hastings algorithm is a more sophisticated Markov Chain Monte Carlo method that takes into account how well P' fits the data. Recall, the transition probabilities of the second version of the random walk we proposed,

$$q_{P,P'} = \mathbb{P}(P_{i+1} = P' \mid P_i = P) = \begin{cases} \frac{2}{p(p+1)} & \text{for all permutation matrices } P' \in \mathcal{P}_{\text{adj}(P)}, \\ 0 & \text{otherwise.} \end{cases}. \quad (4.36)$$

where $\mathcal{P}_{\text{adj}(P)}$ is the set of all permutation matrices on p variables that are one swap of two variables away from P. For ease of notation, we have defined $q_{P,P'} = \mathbb{P}(P_{i+1} = P' \mid P_i = P)$. Let us now devise a random walk that also takes into account how well P' fits $\mathbf{X}$.

Probability of acceptance. A large disadvantage of the random walk is that there is no *guidance* towards a suitable permutation matrix. There is no guarantee you will get closer to a well-fitting permutation matrix. Even worse, for the regular random walk, transitioning to a much worse state just as likely as transitioning to a better state. This is where the Metropolis-Hastings approach improves over the random walk. Instead of blindly agreeing with the proposed next permutation matrix P', we will first evaluate the suitability of P' before transitioning to it.

Nevertheless, we still want to be able to visit permutation matrices that are slightly worse to avoid being stuck with a locally optimal permutation matrix P. Therefore, we need to have an appropriate trade-off between *exploitation*, greedily picking permutation matrices that are more suitable, and *exploration*, trying some permutation matrices that are less suitable to explore the search space of permutation matrices $\mathcal{P}_{\text{perm}}$.

To allow for both exploitation and exploration, the Metropolis-Hastings method uses a *probability of acceptance* $\alpha_{P,P'}$, which we use to decide whether to accept the transition of P to P'. This probability of acceptance is based on some metric that assesses the suitability of a permutation matrix P. We can use the likelihood of the data given our permutation matrix P as such a suitability metric. Given this permutation matrix P, we find the acyclic matrix W_P that maximizes the likelihood of $\mathbf{X}$, while respecting the permutation matrix P using the `Order-OLS` algorithm with $\mathbf{X}$ and P as input. Consequently, we know from Equation 4.5 that the corresponding likelihood of $\mathbf{X}$ given P equal to

$$\mathcal{L}(\mathbf{X} \mid P) = \prod_{t=2}^{T} \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2} \cdot (X_{t,\cdot} - X_{t-1,\cdot}W_P)^T (X_{t,\cdot} - X_{t-1,\cdot}W_P)\right), \quad (4.37)$$

where W_P is the acyclic matrix that maximizes the likelihood given $\mathbf{X}$, while respecting the permutation matrix P. W_P can be retrieved using `Order-OLS`$(\mathbf{X}, P)$, described in Algorithm 4.1.

We can use $\mathcal{L}(\mathbf{X} \mid P)$ to quantify the suitability of P. If we have a new candidate matrix P', we can compare how well both permutation matrices fit the data. Now, if $\mathcal{L}(\mathbf{X} \mid P') > \mathcal{L}(\mathbf{X} \mid P)$, then P' fits the data better than P. It would then be wise to indeed accept P' as our next state. On the other hand, if $\mathcal{L}(\mathbf{X} \mid P')$ is smaller than $\mathcal{L}(\mathbf{X} \mid P)$, then P' fits the data worse than P. Then, we should be somewhat hesitant to transition to P'. Nevertheless, to explore the search space, sometimes we still want to be able to transition to a slightly worse fitting P'. Therefore, let us define $\alpha_{P,P'}$, the probability of accepting P' over P, as

$$\alpha_{P,P'} = \begin{cases} \min\left\{\frac{\mathcal{L}(\mathbf{X}|P')}{\mathcal{L}(\mathbf{X}|P)}, 1\right\} & \text{if } \mathcal{L}(\mathbf{X} \mid P) > 0 \\ 1 & \text{otherwise.} \end{cases} \tag{4.38}$$

We see that $\alpha_{P,P'}$ represents the fraction of likelihoods of permutation matrices P' and P. If P' fits $\mathbf{X}$ better than P fits $\mathbf{X}$, the fraction will be larger than one, and therefore the probability needs to be thresholded to one. If P' fits $\mathbf{X}$ worse than P fits $\mathbf{X}$, the fraction will be between zero and one. It is then still possible to transition to P', but this transition will not always happen.

However, as the likelihood is a value extremely close to zero and is often unpractical to work with, we will use the mean squared error, as defined in Equation 4.28 as a surrogate for the likelihood. As a lower mean squared error implies a larger likelihood, we interchange the nominator and the denominator such that our proposed probability of acceptance is

$$\alpha_{P,P'} = \begin{cases} \min\left\{\frac{\texttt{MSE}(W_P)}{\texttt{MSE}(W_{P'})}, 1\right\} & \text{if } \texttt{MSE}(W_{P'}) > 0 \\ 1 & \text{otherwise.} \end{cases} \tag{4.39}$$

Just as in Equation 4.38, we always transition to a permutation matrix P that yields a higher likelihood, or equivalently, a smaller mean squared error. If the permutation matrix P' achieves a lower likelihood, or equivalently a larger mean squared error, then we base our acceptance probability on the ratio of the mean squared errors.

Metropolis-Hasting transition probabilities. Given the probability of selecting P' as the next possible permutation matrix and the probability of then accepting P' over the current matrix P, we can define the transition probabilities for our Metropolis-Hastings algorithm.

Firstly, the permutation matrix P' has been selected uniformly at random from the set of potential candidates according to the conditional sampling distribution given in Equation 4.36. Secondly, we evaluate how well P' fits $\mathbf{X}$ compared to P by computing the mean squared error of $\mathbf{X}$ when using the coefficient matrices W_P and $W_{P'}$. Based on the fraction, we compute the probability of acceptance according to Equation 4.45. Then, the probability of actually transitioning to P' is given by

$$\mathbb{P}\left(P_{i+1} = P' \mid P_i = P\right) = q_{P,P'} \cdot \alpha_{P,P'} \quad \text{for } P \neq P'. \tag{4.40}$$

Consequently, the probability of *not* transitioning to a new permutation matrix corresponds to

$$\mathbb{P}\left(P_{i+1} = P \mid P_i = P\right) = 1 - \sum_{P \in \mathcal{P}_{\text{adj}(P)}} q_{P,P'} \cdot \alpha_{P,P'}. \tag{4.41}$$

The Metropolis-Hastings algorithm is a random walk using the sampling distribution given in Equations 4.40 and 4.41. This random walk can remain in the same state P for multiple time steps if the considered matrices P' poorly fit $\mathbf{X}$.

We can again use an iteration limit N as a stopping criterion. A similar stopping criterion is to define a *time limit* $t_{\max}$. That is, we keep sampling permutation matrices for at most $t_{\max}$ seconds. A disadvantage of using a time limit $t_{\max}$ over an iteration limit N is that the time limit depends on the device used to evaluate the algorithm. A more powerful computer can perform more iterations of the while loop than a less powerful computer per second. Therefore, an iteration limit N is fairer when the performance is evaluated over different devices.

The pseudocode has been given in Algorithm 4.4.

Algorithm 4.4: Metropolis-Hastings.

Input: A data matrix $\mathbf{X} \in \mathbb{R}^{T \times p}$, An initial $P_0 \in \mathcal{P}_{\text{perm}}$, a stopping criterion.
Output: A matrix $W \in \mathbb{R}^{p \times p}$ such that $G(W)$ is acyclic.

$P \leftarrow P_0$ ▷ Initialize P and W
$W \leftarrow$ `Order-OLS`$(\mathbf{X}, P)$
`W_best` $\leftarrow W$
`MSE_best` $\leftarrow$ `MSE`(`W_best`)
while the stopping criterion is not met **do**
    $P' \leftarrow$ `P_next`(P) ▷ sample next P'
    $W' \leftarrow$ `Order-OLS`$(\mathbf{X}, P)$
    $\alpha \leftarrow \min\{$`MSE`(W) / `MSE`$(W'), 1\}$ ▷ evaluate transition rule
    $P \leftarrow P'$ with probability α
    $W \leftarrow$ `Order-OLS`$(\mathbf{X}, P)$ ▷ evaluate next permutation P
    if `MSE`$(W) <$ `MSE_best` **then**
        `MSE_best` $\leftarrow$ `MSE`(W)
        `W_best` $\leftarrow W$
    end if
end while
return `W_best` ▷ return best solution found so far

Examples. To see whether the Metropolis-Hastings is indeed an improvement over the regular random walk discussed in Section 4.2, we will consider two examples here. We will apply the Metropolis-Hastings algorithm on the same ten-dimensional data matrix as in Example 4.4, as well as a sparser setting.

Example 4.5 Metropolis-Hastings on ten variables with a dense coefficient matrix W.

Let us consider the data matrix $\mathbf{X}$ of Example 4.4. We could try all $10! \approx 3.6$ million permutation matrices, but just as in Example 4.4, we will only explore a subset of the search space, now even as few as $0.01\% = 360$ permutations.

As W is quite dense, there are most likely few permutation matrices P that such that `Order-OLS` can estimate all non-zero coefficients. We can count the number of permutation matrices P compatible with W by counting the number of permutation matrices P such that

$$P^T W P \text{ is upper triangular.} \tag{4.42}$$

Computing this number reveals that only 153 permutation allow us to estimate all coefficients in W. Therefore, we cannot expect to find a permutation matrix such that all 35 non-zero coefficients can be estimated, but perhaps a permutation matrix that estimates only say 30 non-zero coefficients may be satisfactory.

Let us first consider the how the mean squared error of the best matrix found so far changes over the 360 iterations. These trajectories have been plotted as a function of the iteration number i in Figure 4.3.

The initial choice $P_0 = I$ was not a sensible choice, as the mean squared error was quite high. Therefore, we improve quickly and often in the first fifty iterations. Then, the number of iterations between improvements increases, and also the difference in mean squared error between improvements decreases. This is as expected, as you can improve more easily at the start, where the current best solution is farther from optimal. We also see that the Metropolis-Hastings approach seems to have the upper hand in this example, as it finds a more suitable permutation matrix than the random walk, and also more quickly.

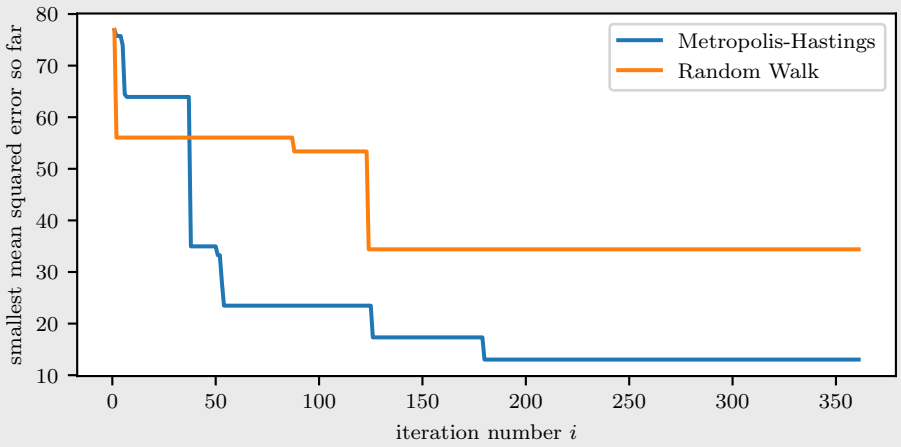


Figure 4.3: Trajectory of the smallest mean squared error found so far.

Let us now consider the mean squared error corresponding to each of the *visited* permutation matrices, rather than the best permutation matrix found so far. This allows us to compare how well both the random walk and the Metropolis-Hastings algorithm choose their next permutation matrix P'. The mean squared errors corresponding to all visited permutation matrices by both algorithms have been plotted in Figure 4.4.

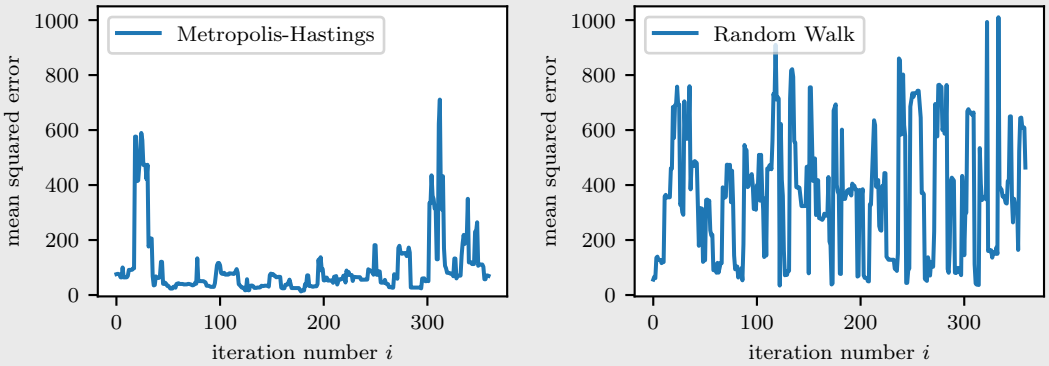


Figure 4.4: The mean squared error of to the permutation matrix at iteration number i.

The Metropolis-Hastings algorithm has far fewer spikes, indicating that it selects permutation matrices that achieve higher likelihoods. Nevertheless, the Metropolis-Hastings algorithm occasionally chooses a poor fitting permutation matrix, showcasing that it sometimes will explore in seemingly less favorable directions to escape local optima.

The random walk selects much poorer fitting permutation matrices. On average, the permutation matrix picked by the Metropolis Hastings algorithm achieves a mean squared error of approximately 94, whereas the mean for the random walk is approximately 330. Comparing medians, the median for the Metropolis-Hastings algorithm is approximately 64, whereas the median for the random walk is approximately 260. In conclusion, the Metropolis-Hastings algorithm chooses candidates with a better likelihood, which makes sense as the Metropolis-Hastings algorithm bases its probability of acceptance on the likelihood.

Density of the coefficient matrix W. In Example 4.5, the Metropolis-Hastings seems more sensible than the random walk. Some less suitable permutation matrices are explored, but most of the time, suitable permutation matrices are chosen to evaluate. The Metropolis-Hastings algorithm works well here is because W has quite a large number of arcs, so the signal-to-noise ratio in $\mathbf{X}$ is high. This in turns means that trying a less suitable permutation matrix will result in a much poorer likelihood. The Metropolis-Hastings algorithm can utilize this and therefore will often not transition to such a matrix. This guided search allows the Metropolis-Hastings algorithm to find suitable permutation matrices faster than the random walk.

Now suppose W is sparser, meaning it contains fewer arcs. Then, the signal-to-noise ratio will be much smaller, so the difference between the best and worst permutation matrix will also be smaller. Therefore, the acceptance probability of a relatively less suitable matrix will be larger. For this, we can compare Figure 4.4 and Figure 4.6. For the denser coefficient matrix W, the ratio between a suitable and an unsuitable permutation permutation matrix is large. However, for a sparse coefficient matrix W, this ratio is much smaller, so the Metropolis-Hastings algorithm will have less of an advantage over the random walk, as its guidance is less strong.

Example 4.6 The original Metropolis-Hastings algorithm on a sparse coefficient matrix.

Secondly, let us consider a sparser example, where the signal-to-noise ratio is much smaller. Instead of 25 off-diagonal elements, the matrix W now has 10 off-diagonal elements. Therefore, we also expect more permutation matrices to be suitable. Therefore, the number of iterations required in this example will most likely be less.

Indeed, there are a total of 2940 suitable permutation matrices, which is much more as in Example 4.5. Therefore we expect a random walk to find such a suitable permutation matrix after fewer iterations. Let us again consider the trajectory of the mean squared error corresponding to the most suitable permutation matrix found so far for the random walk and the Metropolis-Hastings approach. These trajectories have been plotted in Figure 4.5.

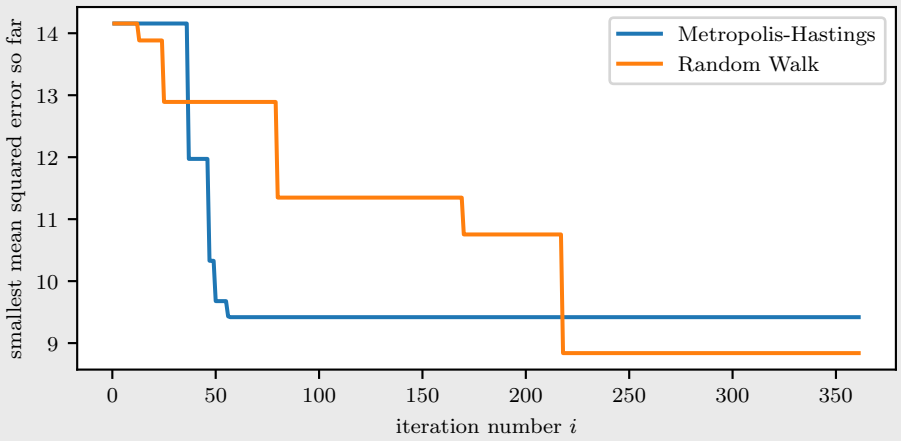


Figure 4.5: Trajectory of the smallest mean squared error found so far.

The performance of the Metropolis-Hastings algorithm is not much better than the random walk anymore. Due to the sparser coefficient matrix W, a poorly chosen permutation matrix does not result in a poor likelihood due to a lower signal-to-noise ratio. To inspect this hypothesis further, let us consider the mean squared errors of the permutation matrices of the Metropolis-Hastings and the random walk algorithm at each iteration in Figure 4.6.

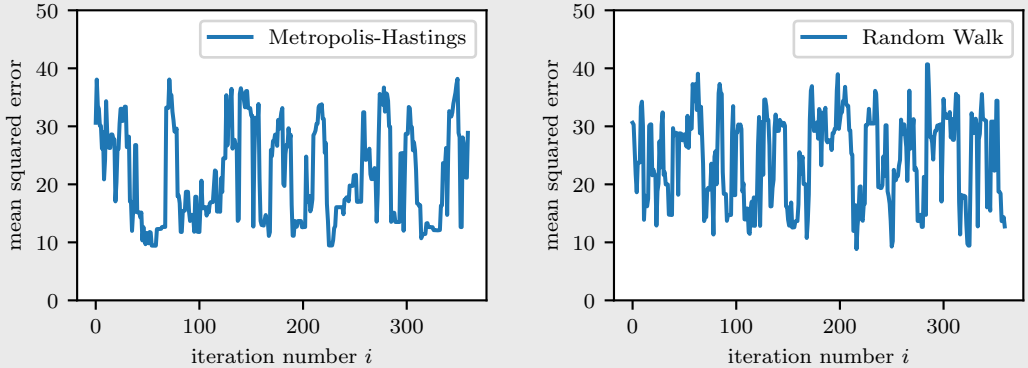


Figure 4.6: Mean squared errors of to the permutation matrix visited at iteration i.

The worst mean squared error is about 40, which is only four times as large as the best mean squared error. In the previous example, the largest mean squared error was one hundred times larger than the smallest mean squared error. Here, this small difference means that the probability of accepting a poor permutation matrix is larger, about one in four, compared to one in one hundred in the previous example. As the mean squared errors are closer together, the Metropolis-Hastings algorithm has difficulties navigating through the search space.

Nevertheless, the Metropolis-Hastings algorithm does pick slightly more suitable permutation matrices on average. The mean and median of the mean squared error for the Metropolis-Hastings are 22 and 20 respectively, compared to 25 and 26 respectively for the random walk. Therefore, the Metropolis-Hastings algorithm is slightly better, but the improvement over the random walk is less significant for this sparser coefficient matrix W.

Fine tuning the probability of acceptance. As defined in Equation 4.45, $\alpha_{P,P'}$ represents the probability of accepting the new permutation matrix P' over the current permutation matrix P. This probability of acceptance does not seem to perform much better than the random walk when the coefficient matrix W is sparse, as we have seen in Example 4.6. This probability of acceptance is not set in stone, and many approaches exist to define such a probability of acceptance. The probability of acceptance should be fine-tuned such that we can capitalize on well-fitting permutation matrices. On the other hand, we should also be able to explore less suitable permutation matrices to make sure we sufficiently explore the search space of permutation matrices.

Translating the mean squared errors. A simple improvement is a translation of the mean squared errors. In Example 4.5, suitable permutation matrices achieve a mean squared error of around ten, whereas poorly suitable permutation matrices achieve a mean squared error of the order of several hundreds. Therefore, the probability of accepting such a poor permutation matrix P' over a well suitable permutation matrix P is then of the order of one hundredths. Nevertheless, in Example 4.6, we see that this bandwidth is much smaller due to a sparser matrix W. Now, the likelihoods range from ten to forty, meaning that we will now often pick one of the lesser suitable permutation matrices. A method to circumvent this is to *translate* the mean squared errors.

Although we do not know the smallest mean squared error a priori, we know that the mean squared error of the (cyclic) ordinary least squares solution is a suitable lower bound. Let W_{OLS} be the ordinary least squares solution, which corresponds to

$$W_{\text{OLS}} = \left(XX^T\right)^{-1} X^T Y, \tag{4.43}$$

where X and Y again correspond to the first and last $T-1$ time steps of $\mathbf{X}$, respectively. The corresponding translated mean squared error then is

$$\texttt{MSE}'(W_P) = \texttt{MSE}(W_P) - \texttt{MSE}(W_{\text{OLS}}). \tag{4.44}$$

The *translated* probability of acceptance $\alpha'_{P,P'}$ then becomes

$$\alpha'_{P,P'} = \begin{cases} \min\left\{\frac{\texttt{MSE}'(W_P)}{\texttt{MSE}'(W_{P'})}, 1\right\} & \text{if } \texttt{MSE}'(W_{P'}) > 0 \\ 1 & \text{otherwise.} \end{cases} \tag{4.45}$$

In Example 4.6, the translated mean squared errors now range from slightly above zero to thirty. Therefore, the fractional differences are higher, decreasing the probability of accepting a poor permutation matrix P'.

Raising the probability of acceptance to the power k. Another method is to raise the probability of acceptance $\alpha_{P,P'}$ to some power $k \geq 0$. Therefore, the probability of acceptance is then equal to

$$\alpha'_{P,P'} = \alpha^k_{P,P'}, \qquad k \geq 0. \tag{4.46}$$

For $k > 1$, raising $\alpha_{P,P'}$ to the kth power will decrease all probabilities of acceptance, but will be harsher on smaller probabilities. Such a transformation will result in a more strict permutation matrix acceptance, resulting in a more exploitative search, where the algorithm is inclined to capitalize on suitable permutation matrices. For $0 < k < 1$, small probabilities will be increased relatively more than larger probabilities. Therefore, using such a tuning parameter $0 < k < 1$ will result in a less strict permutation matrix acceptance, allowing for a more exploratory search.

Greedy acceptance probability. Lastly, we can opt for a greedy search. That is, we accept the transition if and only if the permutation matrix P' is strictly better than the permutation matrix P. That is, the *greedy* probability of acceptance is given by

$$\alpha'_{P,P'} = \begin{cases} 1 & \text{if } \texttt{MSE}(P') < \texttt{MSE}(P) \\ 0 & \text{otherwise.} \end{cases}. \tag{4.47}$$

We only transition to more suitable permutation matrices, and worse permutation matrices are never explored. This also means that it is impossible to escape local optima. Nevertheless, let us return to the setting of Example 4.6 with our approaches to fine-tune the probability of acceptance.

Example 4.7 Fine-tuning the probability of acceptance of Example 4.6.

Let us consider these different acceptance probabilities in the setting of Example 4.6. First, we will investigate translating the mean squared errors by subtracting the mean squared error of the ordinary least squares solution. Second, we will investigate raising the acceptance probability to the power of two. Third, we will also raise the *translated* acceptance probability to the power of two. Lastly, we will consider the greedy acceptance probability, where we only transition when the likelihood of the new permutation matrix is strictly better. We have plotted the trajectories of the smallest mean squared error so far in Figure 4.7.

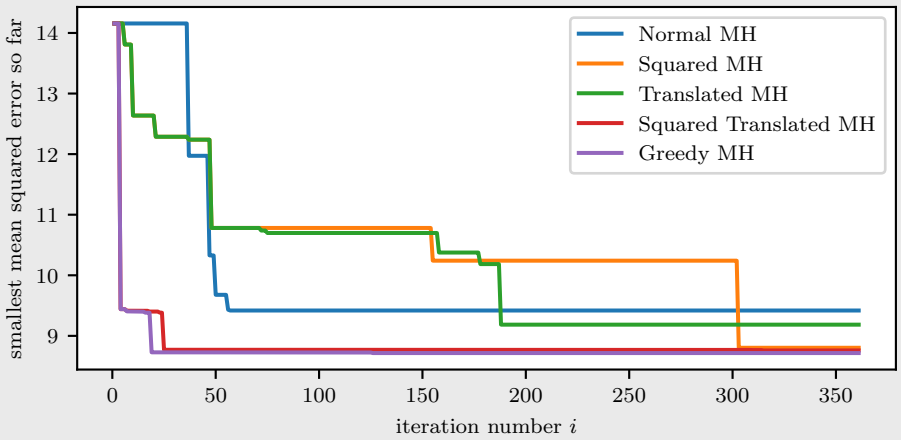


Figure 4.7: Trajectory of the smallest mean squared error found so far for the Metropolis-Hastings algorithm with five different probabilities of acceptance.

First, let us remark that randomness plays a large role, so the comparison should be taken with a grain of salt. Nevertheless, we see that the squared translated and greedy probability of acceptance seem to work the best in the beginning. Nevertheless, after approximately 25 iterations, there is no improvement. The other probabilities of acceptance achieve a larger mean squared error, although the squared probability of acceptance also finds a well-fitting permutation matrix around iteration number 300. The tendency is that the greedier the method, the faster we find a relatively suitable permutation matrix. However, we need to be careful that such a greedy method will not get stuck in a local optimum.

To investigate the exploratory nature of the four new acceptance probabilities, we have plotted the mean squared errors at each iteration in Figure 4.8. The squared and translated probabilities of acceptance remain quite explorative, as many unsuitable permutation matrices are transitioned to. On the other hand, the greedy probability of acceptance is not explorative at all, as less suitable permutation matrices are never transitioned to. Somewhere in between lies the squared translated probability of acceptance, which tries both suitable and unsuitable permutation matrices.

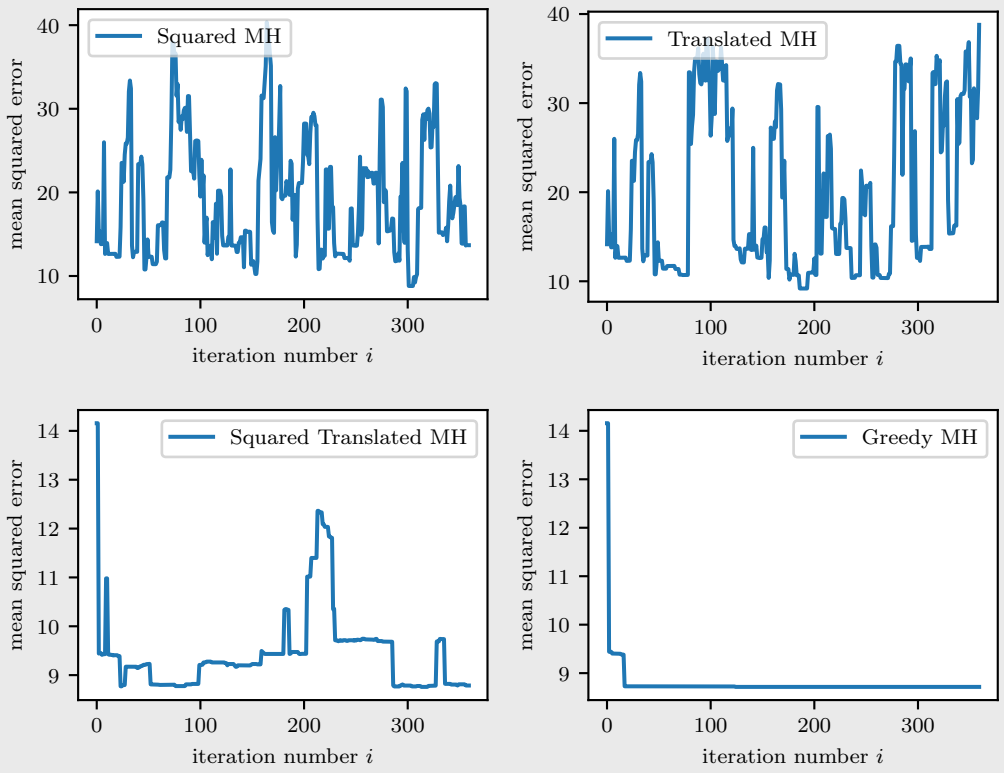


Figure 4.8: Mean squared errors of the permutation matrix at each iteration number i for the Metropolis-Hastings algorithm for different probabilities of acceptance.

Table 4.2: Mean and median mean squared errors corresponding to the 360 permutation matrices visited by the six methods we considered. We have considered the random walk that only considered adjacent permutation matrices (RW) and the regular Metropolis-Hastings algorithm (MH) from Example. Additionally, we have investigated the Metropolis-Hastings algorithm with a translated (MH-Tr), squared (MH-Sq), squared-translated (MH-Sq-Tr), and a greedy (MH-G) transition probability.

	RW	MH	MH-Sq	MH-Tr	MH-Sq-Tr	MH-G
Mean	24.5	21.7	19.2	19.7	9.4	8.8
Median	26.4	20.2	17.5	15.4	9.2	8.7

We see in Table 4.2 that the random walk explores the most of all six methods. Squaring the probability of acceptance seems to decrease the explorativeness a little, and translating the probability of acceptance decreases the level of exploration slightly more. The combination of the two results in even less exploration, whereas the greedy probability of acceptance does not explore less suitable permutation matrices at all.

4.4 Selecting a suitable model complexity

This chapter has so far been devoted to finding a suitable ordering of the p variables in the form of a permutation matrix P. Given this permutation matrix, we use a maximum likelihood approach to find the most probable matrix W that is compatible with this ordering.

As we have seen, maximizing the likelihood yields a complete directed acyclic graph consisting of $p(p+1)/2$ arcs. However, a large portion of these arcs originate from spurious correlations with noise, yielding only a tiny increase in the likelihood. Therefore, we can remove quite some arcs without a significant decrease in performance. Such a "pruned" structure is less complex, which is crucial for making the structure easier to interpret.

Thresholding. The aforementioned spurious correlations that create an overly complex structure are often represented by small weights w_{ij}, which we can omit without a significant decrease in likelihood. A naive yet effective method is to set all coefficients whose absolute values are less than some threshold ϵ to zero. So, the coefficients of W will be thresholded as

$$\tilde{w}_{ij} = \mathbf{1}\left\{|w_{ij}| \geq \epsilon\right\}, \tag{4.48}$$

yielding the thresholded matrix $\tilde{W}$.

However, we still need to pick such a threshold value ϵ. We could, for example, choose $\epsilon = 0.50$, but this might remove important coefficients whose absolute values are smaller than 0.50. If we set a lower threshold of $\epsilon = 0.10$, then we might fail to remove unimportant coefficients whose absolute value is larger than 0.10 due to spurious correlations.

Example 4.8 Selection a suitable number of arcs using thresholding.

Consider the same dataset $\mathbf{X} \in \mathbb{R}^{100\times 10}$ as in Example 4.6, consisting of ten variables and 25 arcs. We expect our permutation-based methods to find a suitable coefficient matrix W that respects some permutation matrix P. However, without any thresholding, W will consist of $\binom{10}{2} = 55$ arcs, where we expect 30 arcs to be there just because of spurious correlations. Using the exhaustive search algorithm, we see that the most probable acyclic matrix W achieves a mean squared error of 9.28.

Now, let us see what happens to the number of arcs, or equivalently, the number of non-zero coefficients as we increase the threshold ϵ. Furthermore, we will also consider how the mean squared error increases as we increase ϵ. Both plots are shown in Figure 4.9.

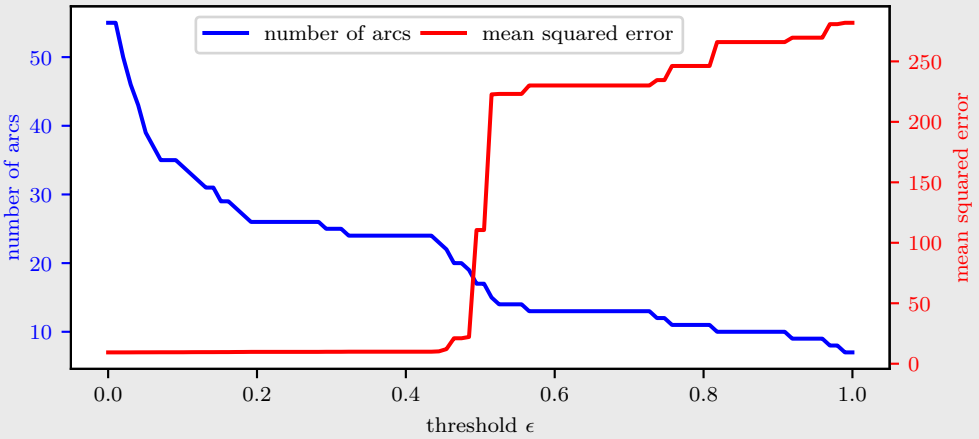


Figure 4.9: Number of arcs and mean squared error as a function of the threshold ϵ.

Initially increasing the threshold ϵ greatly reduces the number of non-zero coefficients, at the cost of only a minor increase in likelihood. We have found a sweet spot at around $\epsilon = 0.4$, where the number of arcs is greatly reduced from 55 to 25 without a significantly loss in predictive performance. If we increase the threshold further, we see that the number of arcs decreases. However, the mean squared error increases significantly, indicating that we pruned too many arcs. Therefore, we see that using such a threshold can greatly reduce the complexity of the structure while only losing a small portion of predictive performance. However, it is difficult to select a suitable threshold ϵ beforehand, and the size of a coefficient may not be indicative of its predictive performance.

Chapter 5

Continuous Approaches

In Chapter 4, we have considered *permutation-based* approaches, where we have decomposed the problem of finding an acyclic matrix W into finding a suitable ordering π of the variables or, equivalently, a permutation matrix P. Subsequently, we find an upper triangular matrix U compatible with this permutation matrix P using the decomposition

$$W = P^T U P. \tag{5.1}$$

Although the results of the three methods discussed in Chapter 4 seem promising, these permutation-based approaches have several drawbacks. First, searching over the space of permutation matrices P is difficult. The space is combinatorial, making it hard to navigate. Secondly, the search space of permutation matrices is exponential with respect to the number of variables. There exist $p!$ different permutations on p variables. Therefore, the search space grows exponentially fast when the number of variables p increases. We saw that exhaustively searching all permutations was infeasible for more than ten variables. To circumvent this, we settled for a suboptimal solution that we could find much faster. Nevertheless, we expect the performance of these permutation-based approaches to decrease when p gets large.

Outline of this chapter. This chapter focuses on continuous approaches that do not iterate over all possible permutations. Rather than optimizing over a search space that is combinatorial, such as the set of permutation matrices, we are now interested in solutions that optimize over a *continuous* search space. Instead of enforcing acyclicity using combinatorial constraints, we are now investigating whether we can enforce acyclicity using continuous constraints. These approaches will hopefully be more suitable for high-dimensional settings. As we are now trying to learn the structure of our data matrix $\mathbf{X}$ using continuous optimization methods, we have named this chapter "continuous approaches".

We will discuss three approaches in this chapter. First of all, we will discuss a method close to permutation-based approaches in Section 5.1, where we relax the combinatorial search space of permutations matrices to a continuous search space. Secondly, in Section 5.2 we will explore an approach similar to the authors of NOTEARS [87] who discovered a novel approach to enforce acyclicity using a continuous function. Lastly, we propose a new method in Section 5.3 that enforces acyclicity using LASSO and systematically increasing the penalty parameter until the inferred structure is acyclic.

5.1 Relaxing the space of permutation matrices.

Recall that the permutation-based approaches in Chapter 4 utilize the matrix decomposition given in Equation 5.1. Although this nicely enforces an acyclic structure, its scalability poses some drawbacks. The method discussed in this section hopes to resolve the issue that the space of permutation matrices is both combinatorial and exponential with respect to the number of variables p.

The Birkhoff polytope. One approach is to extend or relax the space of permutation matrices to some continuous search space, thereby removing the combinatorial constraint that P is a permutation matrix. A natural extension of the discrete set of permutation matrices $\mathcal{P}_{\text{perm}}$ is the continuous set of *doubly stochastic* matrices, which we denote by $\mathcal{P}_{\text{DS}}$. A matrix P is said to be *doubly stochastic* if and only if all its elements are non-negative and all of its rows and columns sum to one. In mathematical notation,

$$P \text{ is doubly stochastic } \iff p_{ij} \geq 0 \text{ and } \sum_{j=1}^{p} p_{ij} = \sum_{i=1}^{p} p_{ij} = 1 \;\forall\; i, j = 1, \ldots, p. \tag{5.2}$$

Clearly, $\mathcal{P}_{\text{perm}} \subseteq \mathcal{P}_{\text{DS}}$, as each permutation matrix is doubly stochastic.

Interestingly, $\mathcal{P}_{\text{DS}}$ is the *convex hull* of P_{perm}. This implies that every doubly stochastic matrix can be written as a convex combination of permutation matrices, as the Birkhoff-von Neuman Theorem states [6, 79]. $\mathcal{P}_{\text{DS}}$ is therefore also known as the *Birkhoff polytope*. The vertices of the Birkhoff polytope correspond to permutation matrices, and all the interior points and points on a line segment are doubly stochastic matrices, but not permutation matrices. A 2D-visualization of the Birkhoff polytope in three dimensions is given in Figure 5.1

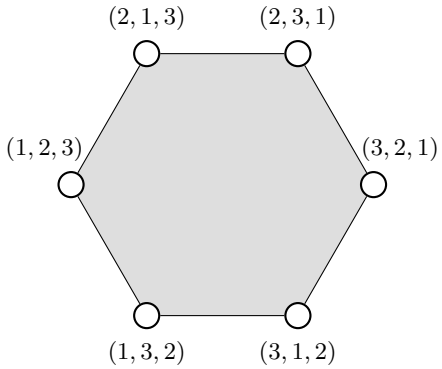


Figure 5.1: A two-dimensional representation of the Birkhoff polytope for three variables. Each of the $3! = 6$ vertices represent a permutation matrix, and all points on the edges or in the gray interior of the Birkhoff polytope represent doubly stochastic matrices.

Model Reformulation. Having proposed this relaxation, let us see how this changes the decomposition of our coefficient matrix W. Recall from Chapter 2 that we have defined our time-dependent graphical model as a VAR(1) model, which we have defined in Definition 2.3. We have a data matrix $\mathbf{X} \in \mathbb{R}^{T \times p}$, and assume the generative model

$$X_{t,\cdot} = X_{t-1,\cdot} W + \varepsilon_t, \qquad t = 2, \ldots, T \tag{5.3}$$

where we assume $X_{1,\cdot}$ is known, and $W \in \mathbb{R}^{p \times p}$ is a coefficient matrix characterizing the acyclic structure of the joint distribution $\mathbb{P}(\mathbf{X})$. Furthermore, ε_t represents the random Gaussian noise, which we assume is independent of the other noise variables, $\varepsilon_t \perp\!\!\!\perp \varepsilon_{t'}$ for $t \neq t'$.

When we constrained ourselves to the space of permutation matrices $\mathcal{P}_{\text{perm}}$, we could enforce acyclicity by rewriting W as $W = P^T U P$, where P is a permutation matrix and U is an upper

triangular matrix. However, now that we have relaxed to the space of doubly stochastic matrices, this decomposition is not accurate anymore. Rather than requiring the transpose of P, we now require its inverse. Therefore, we require

$$W = P^{-1}UP \text{ where } P \in \mathcal{P}_{\text{DS}}, \text{ rather than } W = P^{T}UP \text{ where } P \in \mathcal{P}_{\text{perm}},$$

for an upper triangular matrix U.

We could have also used the left-hand notation for permutation matrices, as $P^T = P^{-1}$ for $P \in \mathcal{P}_{\text{perm}}$. However, for doubly stochastic matrices, often $P^T \neq P^{-1}$. So, the decomposition under our relaxation corresponds to

$$X_{t,\cdot} = X_{t-1,\cdot}P^{-1}UP + \varepsilon_t, \tag{5.4}$$

where $P \in \mathcal{P}_{\text{DS}}$ and U is an upper triangular matrix.

Issues arising with the matrix inverse. Unfortunately, relaxing our problem to the set of doubly stochastic matrices has introduced new problems related to calculating the inverse of P.

First of all, the inverse of P does not exist for all doubly stochastic matrices P. For example, consider the doubly stochastic matrix

$$P = \begin{pmatrix} 0.333\ldots & 0.333\ldots & 0.333\ldots \\ 0.333\ldots & 0.333\ldots & 0.333\ldots \\ 0.333\ldots & 0.333\ldots & 0.333\ldots \end{pmatrix}. \tag{5.5}$$

We see that all three rows are exactly equal, and therefore all three are linearly dependent. This in turn implies that the matrix only has rank 1 and therefore its inverse is undefined. We see that not all doubly stochastic matrices have a defined inverse. Therefore, the model reformulation as in Equation 5.4 is not always well-specified.

Secondly, even when the matrix is theoretically invertible, we can still encounter numerical issues in practice. Let ϵ be an arbitrarily small number. Consider now the slightly different matrix

$$P = \begin{pmatrix} 0.333 - \epsilon & 0.333 + \epsilon & 0.333 \\ 0.333 + \epsilon & 0.333 - \epsilon & 0.333 \\ 0.333 & 0.333 & 0.333 \end{pmatrix}. \tag{5.6}$$

Theoretically, for any $\epsilon \neq 0$, this matrix is non-singular, as all the rows are linearly independent. However, from a numerical perspective, its inverse is quite unstable. Even for a moderately large $\epsilon = 0.05$, the inverse of P is equal to

$$P^{-1} = \begin{pmatrix} 4.5e15 & 4.5e15 & -9.0e15 \\ 4.5e15 & 4.5e15 & -9.0e15 \\ -9.0e15 & -9.0e15 & 1.8e16 \end{pmatrix}. \tag{5.7}$$

Equation 5.5 and Equation 5.7 illustrate the problems that arise from our decomposition. For permutation matrices, computing the inverse of P was as easy as computing its transpose. For doubly stochastic matrices, this is not as simple anymore. Although we have resolved the issues caused by the combinatorial search space of permutation matrices, we are now left with invertibility issues. Nevertheless, let us investigate how serious these issues are in practice.

Cost function for finite samples. To find a suitable doubly stochastic matrix P and an upper triangular matrix U, let us again consider minimizing the mean squared error. For ease of notation, let us define the mean squared error as the cost function $C(P, U)$ that we want to minimize with respect to P and U,

$$C(P, U) = \frac{1}{T-1}\sum_{t=2}^{T} \left\| X_{t,\cdot} - X_{t-1,\cdot}P^{-1}UP \right\|_2^2. \tag{5.8}$$

.

Cost function for infinite samples. Let us first investigate the performance of this relaxation approach in an easier setting, the infinite-sample setting. Rather than having a finite sample of T time steps, we assume we have infinite data. In mathematical notation, this means that we can take the expectation of one arbitrary time step t,

$$\begin{aligned}
\mathbb{E}\left[C(P,U)\right] &= \lim_{T\to\infty} C(P,U) \\
&= \lim_{T\to\infty} \frac{1}{T-1}\sum_{t=2}^{T} \left\|X_{t,\cdot} - X_{t-1,\cdot}P^{-1}UP\right\|_2^2 \\
&= \mathbb{E}\left[\left\|X_{t,\cdot} - X_{t-1,\cdot}P^{-1}UP\right\|_2^2\right] \\
&= \mathrm{Tr}\left(\mathbb{E}\left[\left(X_{t,\cdot} - X_{t-1,\cdot}P^{-1}UP\right)\left(X_{t,\cdot} - X_{t-1,\cdot}P^{-1}UP\right)^T\right]\right) \\
&= \mathrm{Tr}\left(\mathbb{E}\left[\hat{\varepsilon}_t\hat{\varepsilon}_t^T\right]\right), \qquad (5.9)
\end{aligned}$$

where we have defined $\hat{\varepsilon}_t = X_{t,\cdot} - X_{t-1,\cdot}P^{-1}UP$. Furthermore $\mathrm{Tr}\left(\cdot\right)$ denotes the *trace*, corresponding to the sum of all diagonal entries of its argument.

We can further decompose Equation 5.9 to gain more insights into the construction of our cost function. Using the definition of the covariance, we have that

$$\begin{aligned}
\mathrm{Tr}\left(\mathbb{E}\left[\hat{\varepsilon}_t\hat{\varepsilon}_t^T\right]\right) &= \mathrm{Tr}\left(\mathbb{V}\left(\hat{\varepsilon}_t\right) - \mathbb{E}\left[\hat{\varepsilon}_t\right]\mathbb{E}\left[\hat{\varepsilon}_t\right]^T\right) \\
&= \mathrm{Tr}\left(\mathbb{V}\left(\hat{\varepsilon}_t\right)\right) - \mathrm{Tr}\left(\mathbf{0}\mathbf{0}^T\right) \qquad (5.10) \\
&= \mathrm{Tr}\left(\mathbb{V}\left(X_{t,\cdot} - X_{t-1,\cdot}P^{-1}UP\right)\right) \\
&= \mathrm{Tr}\left(\mathbb{V}\left(X_{t,\cdot} - X_{t-1,\cdot}W\right)\right), \qquad (5.11)
\end{aligned}$$

where we used the linearity of the trace operator and the fact that $\mathbb{E}\left[X_{t,\cdot}\right] = \mathbf{0}$ in Equation 5.10.

Now, assuming that the data has been generated by a VAR(1) model with data generating matrix W^*,

$$X_{t,\cdot} = X_{t-1,\cdot}W^* + \varepsilon_t, \qquad \text{where } \varepsilon_t \sim \mathcal{N}_p\left(\mathbf{0}, I_p\right), \qquad (5.12)$$

we can derive Equation 5.11 further into

$$\begin{aligned}
\mathbb{V}\left(X_{t,\cdot} - X_{t-1,\cdot}W\right) &= \mathbb{V}\left(X_{t-1,\cdot}W^* + \varepsilon_t - X_{t-1,\cdot}W\right) \\
&= \mathbb{V}\left(X_{t-1,\cdot}(W^* - W) + \varepsilon_t\right) \\
&= \mathbb{V}\left(X_{t-1,\cdot}(W^* - W)\right) + \mathbb{V}\left(\varepsilon_t\right) \\
&= (W^* - W)^T\mathbb{V}(X_{t-1,\cdot})(W^* - W) + I_p. \qquad (5.13)
\end{aligned}$$

What remains now is to derive the covariance of $X_{t-1,\cdot}$, or equivalently, the covariance of $X_{t,\cdot}$. As we assume that all ε_t are independent Gaussian random variables, we know that $X_{t,\cdot}$ will follow a Gaussian distribution as well. From existing literature on Vector AutoRegressive models [92], we know that that we can derive the covariance matrix as

$$\mathrm{vec}\left(\mathbb{V}\left(X_{t,\cdot}\right)\right) = \left(I_{p^2} - \left(W^T \otimes W^T\right)\right)^{-1}\mathrm{vec}\left(I_p\right), \qquad (5.14)$$

where $\mathrm{vec}(\cdot)$ represents the *vectorization operation*, which stacks all columns into large one column, and $\otimes$ represents the *Kronecker product*.

We conclude that the expected value of the mean squared error for our VAR(1) model equals

$$\begin{aligned}
\mathbb{E}\left[C(P,U)\right] &= \mathrm{Tr}\left(\mathbb{V}\left(X_{t,\cdot} - X_{t-1,\cdot}W\right)\right) && \text{by Equation 5.11} \\
&= \mathrm{Tr}\left((W^* - W)^T\mathbb{V}(X_{t,\cdot})(W^* - W) + \mathbb{V}\left(I_p\right)\right) && \text{by Equation 5.13} \\
&= \mathrm{Tr}\left((W^* - W)^T\mathbb{V}(X_{t,\cdot})(W^* - W)+\right) + p, && (5.15)
\end{aligned}$$

where Equation 5.15 follows from the linearity of the trace operator.

Global minimum of $\mathbb{E}[C(P,U)]$ Let us take a closer look at the global minimum of $\mathbb{E}[C(P,U)]$. We first remark that $\mathbb{V}(X_{t,.})$ is positive semi-definite, as is any covariance matrix. Furthermore, $\mathbb{V}(X_{t,.})$ is positive definite if it has full rank. Assuming the covariance matrix is indeed full rank, then by the definition of positive definiteness we have that for any W, W' s.t. $W \neq W'$,

$$(W - W')^T \mathbb{V}(X_{t,.})(W - W') > 0. \tag{5.16}$$

Hence, we can remark that the global optimum of $\mathbb{E}[C(P,U)]$ is attained only when $W = W'$. Therefore, we can *only* attain the global minimum when we have that our estimated matrix W is exactly equal to the data generating matrix W^*.

In conclusion, $\mathbb{E}[C(P,U)]$ is minimized if and only if

$$P \in \mathcal{P}_{\text{DS}},\ U \text{ upper triangular such that } P^{-1}UP = W^*. \tag{5.17}$$

The corresponding global minimum then is equal to

$$\text{Tr}\left(\mathbb{V}\left(\varepsilon_t\right)\right) = \text{Tr}\left(I_p\right) = p. \tag{5.18}$$

This is a promising result for relaxation of the search space. Although we have extended the search space to the space of doubly stochastic matrices, we have not introduced doubly stochastic matrices that can achieve a lower mean squared error than permutation matrices. Therefore, this relaxation seems sensible.

Example 5.1 Two (P,U) pairs that can compose W.

Nevertheless, although there is only one global minimum with respect to W, there can still be multiple pairs of (P,U) that attain this global minimum. Let us consider the scenario where

$$W = \begin{pmatrix} 0.5 & 0.3 & 0.2 \\ 0.0 & 0.4 & 0.0 \\ 0.0 & 0.0 & 0.6 \end{pmatrix}.$$

We can decompose this W into two (P,U) pairs,

$$P_1 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix},\ U_1 = \begin{pmatrix} 0.5 & 0.3 & 0.2 \\ 0.0 & 0.4 & 0.0 \\ 0.0 & 0.0 & 0.6 \end{pmatrix}, \qquad P_2 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix},\ U_2 = \begin{pmatrix} 0.5 & 0.2 & 0.3 \\ 0.0 & 0.6 & 0.0 \\ 0.0 & 0.0 & 0.4 \end{pmatrix}.$$

The reason for this is that there are multiple permutations compatible with W, namely $(1,2,3)$ and $(1,3,2)$, as there is no relation 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.

Finding a local optimum of $\mathbb{E}[C(P,U)]$ using gradient descent. Now that the $\mathbb{E}[C(P,U)]$ has been derived and analyzed, the next topic of interest is inferring the true matrix W by finding a suitable doubly stochastic matrix P and an upper triangular matrix U. A simple yet efficient approach to infer these parameters is by performing a *gradient descent* with respect to the matrices P and U. Using gradient descent, we can find a stationary point where all partial derivatives with respect to the coefficients in P and U are equal to zero. Hopefully, such a stationary point corresponds to a suitable coefficient matrix W. For ease of notation, we will drop the expectation for now, and simply refer to $\mathbb{E}[C(P,U)]$ as $C'(P,U)$,

$$C'(P,U) = \mathbb{E}\left[C(P,U)\right]. \tag{5.19}$$

The procedure for gradient descent is as follows. We iteratively update the entries of the matrices P and U by changing the corresponding values in the direction of the steepest descent. One by one, step by step, we update the entries such that the corresponding partial derivatives decrease towards zero. If we keep updating these entries, we will eventually end up in a stationary point

where all partial derivatives of the parameters are equal to zero. Instead of searching for a point where $\nabla_P C'(P,U)$ and $\nabla_U C'(P,U)$ are *exactly* equal to the zero matrix, we are satisfied when the gradient is sufficiently small, say when the squared 2-norm of all coefficients in P and U is below some threshold ϵ. The pseudocode of this procedure is provided in Algorithm 5.1.

Algorithm 5.1: Coordinate gradient descent without Lagrange multipliers.

Input: Data matrix $\mathbf{X} \in \mathbb{R}^{T \times p}$, a step size $\eta > 0$, a gradient threshold $\epsilon > 0$.
Output: A coefficient matrix $W \in \mathbb{R}^{p \times p}$.

```
initialize doubly stochastic Matrix P.
initialize upper triangular Matrix U.
while ‖∇_P C'(P,U)‖_F^2 + ‖∇_U C'(P,U)‖_F^2 ≥ ε do
    P ← P − η∇_P C'(P,U)
    U ← U − η∇_U C'(P,U)
end while
return P^{-1}UP
```

To perform this gradient descent, the gradients of $C'(P,U)$ with respect to our input matrices P and U need to be derived. We will only state the gradients here, and refer the interested reader to Appendix A.2. For the doubly stochastic matrix P, the coefficients of the gradient $\nabla_P C'(P,U)$ are equal to

$$\left(\nabla_P C'(P,U)\right)_{ij} = -2\mathrm{Tr}\left(\mathbb{V}\left(X_{t,\cdot}\right)\left(W^* - W\right)^T\left(-P^{-1}J^{ij}P^{-1}UP + P^{-1}UJ^{ij}\right)\right), \tag{5.20}$$

where J^{ij} represents the matrix containing only zeros, apart from the value at the ith row and jth column, which has value one. Furthermore, we use the composition $W = P^{-1}UP$ for more concise notation. For the upper triangular matrix U, we will fix the gradient of U to zero for all lower triangular entries. Then, the partial derivatives in our gradient $\nabla_U C'(P,U)$ can be computed as

$$\Big(\nabla_U C'(P,U)\Big)_{ij} = \begin{cases} \Big(-2(W - W^*)\mathbb{V}\left(X_{t,\cdot}\right)\Big)_{ij} & \text{if } i \leq j, \\ 0 & \text{otherwise.} \end{cases} \tag{5.21}$$

Adhering to the constraints of P. Recall that P needs to be doubly stochastic, meaning that all entries must be nonnegative and all its rows and columns must sum to one. When performing this gradient descent, these constraints need to be taken into consideration. A standard way for this is by introducing *Lagrange multipliers*. We introduce p^2 inequality constraints

$$0 \leq p_{ij}, \qquad i,j = 1,\dots p, \tag{5.22}$$

corresponding to p^2 *nonnegative* Lagrange multipliers

$$\lambda_{\text{ineq},i,j} = p_{ij}, \qquad i,j = 1,\dots,p. \tag{5.23}$$

Furthermore, we introduce $2p$ equality constraints to ensure all rows and columns sum to one,

$$\sum_{k=1}^{p} p_{ik} = \sum_{k=1}^{p} p_{kj} = 1, \qquad i,j = 1,\dots,p, \tag{5.24}$$

corresponding to $2p$ Lagrange multipliers

$$\lambda_{\text{eq},1,i} = \sum_{k=1}^{p} p_{ik} - 1, \qquad \lambda_{\text{eq},2,j} = \sum_{k=1}^{p} p_{kj} - 1, \qquad i,j = 1,\dots,p. \tag{5.25}$$

The Lagrangian function now becomes

$$\mathcal{L}(P,U,\lambda) = C'(P,U) - \sum_{i=1}^{p}\sum_{j=1}^{p}\lambda_{\text{ineq},i,j}p_{ij} - \sum_{i=1}^{p}\lambda_{\text{eq},1,i}\left(\sum_{k=1}^{p}p_{ik}-1\right) - \sum_{j=1}^{p}\lambda_{\text{eq},2,j}\left(\sum_{i=1}^{p}p_{kj}-1\right). \quad (5.26)$$

We see that $\mathcal{L}(P,U,\lambda)$ is equal to $C'(P,U)$ if and only if for each constraint, we have that either the constraint is satisfied or its corresponding Lagrange multiplier is equal to zero. Consequently, minimizing the Lagrangian using gradient descent with respect to P and U will yield a local optimum of $C'(P,U)$, where P indeed corresponding to a doubly stochastic matrix.

We do need to recompute the gradients, however, as we add Lagrange multipliers who are also multiplied by P. The gradient of the Lagrangian dual with respect to U remain the same, so

$$\nabla_U \mathcal{L}(P,U,\lambda) = \nabla_U C'(P,U). \quad (5.27)$$

The gradient of the Lagrangian with respect to P will now be

$$(\nabla_P \mathcal{L}(P,U,\lambda))_{ij} = (\nabla_P C'(P,U))_{ij} - \lambda_{\text{ineq},i,j} - \lambda_{\text{eq},1,i} - \lambda_{\text{eq},2,j}. \quad (5.28)$$

We have now derived all necessary components to perform gradient descent with the accompanying constraints on P. The pseudocode is depicted in Algorithm 5.2.

Algorithm 5.2: Coordinate Gradient Descent With Lagrange Multiplies

Input: Data matrix $\mathbf{X} \in \mathbb{R}^{T\times p}$, a step size $\eta > 0$, a gradient threshold $\epsilon > 0$.
Output: A coefficient matrix $W \in \mathbb{R}^{p\times p}$.

initialize doubly stochastic Matrix P
initialize upper triangular Matrix U
initialize Lagrange multipliers λ using Equation 5.23 and Equation 5.25
while $\|\nabla_P \mathcal{L}(P,U,\lambda)\|_F^2 + \|\nabla_U \mathcal{L}(P,U,\lambda)\|_F^2 \geq \epsilon$ **do**
    compute $\nabla_P \mathcal{L}(P,U,\lambda)$ using Equation 5.20 and Equation 5.28
    compute $\nabla_U \mathcal{L}(P,U,\lambda)$ using Equation 5.21 and Equation 5.27
    $P \leftarrow P - \eta \nabla_P \mathcal{L}(P,U,\lambda)$
    $U \leftarrow U - \eta \nabla_U \mathcal{L}(P,U,\lambda)$
    update Lagrange multipliers λ using Equation 5.23 and Equation 5.25
end while
return $P^{-1}UP$

Gradient descent with acyclic W^*. Let us first apply this algorithm on a relatively simple example to see whether this approach is sensible. If the data generating matrix W^* is indeed acyclic, Algorithm 5.2 should output a suitable acyclic matrix W, decomposed into a doubly stochastic matrix P and an upper triangular matrix U. At the very least, we hope to find a stationary point of Equation 5.26 corresponding to an acyclic structure. Additionally, we hope that the doubly stochastic matrix P will closely resemble a permutation matrix.

Example 5.2 Gradient descent with acyclic W
Let us consider the three dimensional data generating matrix

$$W^* = \begin{pmatrix} 0.85 & 0.55 & 0.50 \\ 0.00 & 0.75 & 0.20 \\ 0.00 & 0.00 & 0.42 \end{pmatrix}.$$

We perform the gradient descent method in Algorithm 5.2 until the squared 2-norm of all partial derivatives is smaller than $\epsilon = 10^{-3}$, after which retrieve our estimated matrices P and U, and therefore also W,

$$P = \begin{pmatrix} 0.55 & 0.15 & 0.29 \\ 0.19 & 0.42 & 0.38 \\ 0.20 & 0.45 & 0.39 \end{pmatrix}, \quad U = \begin{pmatrix} 0.63 & 0.15 & 0.29 \\ 0.00 & 0.51 & 0.37 \\ 0.00 & 0.00 & 0.90 \end{pmatrix}, \quad W = \begin{pmatrix} 0.83 & 0.54 & 0.50 \\ 0.00 & 0.77 & 0.19 \\ 0.03 & -0.03 & 0.43 \end{pmatrix}.$$

The matrix W is close to the true matrix, but not quite. The reason for this is that we are not exactly in a local optimum, as the gradient is slightly off from zero. If we iterated further until the gradient was smaller, we expect to reach this global optimum exactly. However, this convergence was rather slow, and seemed to stagnate after a couple of minutes.

A closer inspection into the partial derivatives of P provide the explanation why gradient descent has difficulties converging closer to a stationary point. In Figure 5.2, the partial derivative of $\mathcal{L}(P, U, \lambda)$ with respect to p_{31} has been shown.

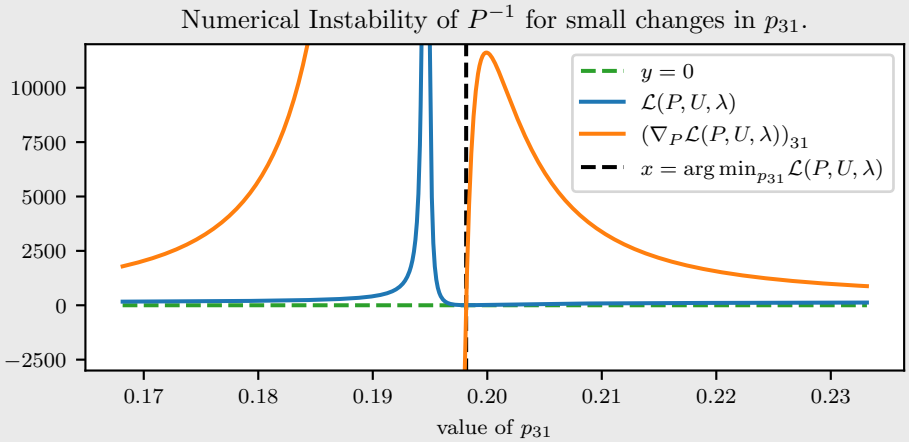


Figure 5.2: Value of the Lagrangian function and its partial derivative with respect to p_{31}, where the minimum of the Lagrangian coincides with the root of the partial derivative.

Figure 5.2 reveals the singularity issues of P. As the two bottom rows of P are similar, slightly tweaking p_{31} results in two linearly dependent rows of P so that the entries of the inverse matrix attain extremely large values, just as in Equation 5.7. When $p_{31} \approx 0.198$, the partial derivative of the Lagrangian with respect to p_{31} is equal to 0.015. However, if we increase the value of p_{31} by 0.002, the partial derivative has become larger than 10,000, which indicates that minor changes to p_{31} result in a substantial change in the corresponding partial derivative. This makes it difficult to exactly find a stationary point of $\mathcal{L}(P, U, \lambda)$. Nevertheless, the coefficient matrix W is reasonably close to the true coefficient matrix W^*.

Example 5.2 showcases two problems of with the gradient descent algorithm. First of all, we see that, although W^* is indeed acyclic, the corresponding doubly stochastic matrix P does not resemble a permutation matrix. We had hoped that a local optimum for W would result in a doubly stochastic matrix similar to a permutation matrix. Unfortunately, this is not the case.

Secondly, the singularity issues that arise with the inverse of P show that finding a local optimum is quite difficult, even in just three dimensions. Small changes to a coefficient of P may result in substantial differences in the partial derivative of the Lagrangian. These two issues raise the question whether this method will be suitable for higher dimensions, as we are already encountering quite some difficulties in just three dimensions.

Coordinate Gradient Descent with cyclic W^*. As in real life, we should also investigate what happens when the data generating matrix W^* does not correspond to an acyclic structure. Then, although we are unable to find a perfect fit, we should strive to find a suitable acyclic matrix W. Therefore, let us see what happens when we apply gradient descent in a cyclic setting. Hopefully, the returned permutation matrix P and upper triangular matrix U compose into an acyclic structure.

Example 5.3 Gradient descent with cyclic W.

Consider a simple three dimensional settings where

$$W^* = \begin{pmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.5 \\ 0.0 & 0.5 & 0.0 \end{pmatrix}.$$

Although this is not a very meaningful VAR(1) model, it showcases a simple three-dimensional setting where we have a cyclic matrix W^*. We perform gradient descent on the coefficients of P and U as described in Algorithm 5.2 with an appropriately small step size η until the squared 2-norm of all partial derivatives is smaller than $\epsilon = 10^{-3}$.

In the end, we see our estimated matrices are

$$P = \begin{pmatrix} 0.04 & 0.48 & 0.48 \\ 0.03 & 0.49 & 0.48 \\ 0.93 & 0.03 & 0.04 \end{pmatrix}, \quad U = \begin{pmatrix} 0.50 & 0.99 & 0.00 \\ 0.00 & -0.50 & 0.04 \\ 0.00 & 0.00 & 0.00 \end{pmatrix}, \quad W = \begin{pmatrix} 0.00 & 0.00 & 0.00 \\ 0.01 & 0.00 & 0.50 \\ 0.01 & 0.50 & 0.00 \end{pmatrix}.$$

Clearly, P is not similar to a permutation matrix. Furthermore, the matrix W is not acyclic.

The decomposition does not provide the intended behavior. We had hoped for an acyclic W, but this is not the case. The matrices P and U together allow for enough freedom such that $W = P^{-1}UP$ is cyclic, even when U is strictly upper triangular.

Unfortunately, as we see in Example 5.3, although U is indeed upper triangular by construction, the additional parameters in P allow for a doubly stochastic matrix P such that we can compose a matrix W that does not correspond to an acyclic structure. As we can still construct cyclic matrices W under this decomposition, this relaxation method seems unsuitable to enforce acyclicity of W.

Potential improvements. Although this decomposition did not provide the intended behavior, we have tried some potential improvements. As these ideas did not significantly improve the approach, we will only describe them briefly without additional derivations.

First of all, we can circumvent the inequality constraints imposing that

$$p_{ij} \geq 0, \qquad i, j = 1, \ldots, p, \tag{5.29}$$

by transforming the matrix P by using the element-wise *sigmoid* function,

$$\sigma(p_{ij}) = \frac{1}{1 + \exp(-p_{ij})}. \tag{5.30}$$

As $0 \leq \sigma(p_{ij}) \leq 1$ for all real values of p_{ij}, we can use this transformation to alleviate ourselves of all the inequality constraints.

Secondly, we have also investigated using surrogates for the inverse that do not have the same singularity issues as the inverse. We can, for example, use the the transpose over the determinant as a surrogate,

$$P^{-1} \approx \frac{P^T}{\det(P)}. \tag{5.31}$$

Lastly, we can also consider adding a penalty that forces P closer to a permutation matrix. We can, for example, use the Frobenius norm to achieve this, as $\|P\|_F = \sqrt{p}$ if $P \in \mathcal{P}_{\text{perm}}$, and $\|P\|_F \leq \sqrt{p}$ if $P \in \mathcal{P}_{\text{DS}}$. Therefore, adding a penalty of the form

$$\lambda \left(\sqrt{p} - \|P\| \right), \qquad \lambda \geq 0, \tag{5.32}$$

forces the doubly stochastic matrix closer to a permutation matrix P. Adding such a penalty seemed to work well for a similar relaxation method discussed in [17].

Unfortunately, all three proposals did not solve the problems that we have encountered using this approach, and therefore were not further pursued.

5.2 Applying NOTEARS to VAR(1) models.

In this section, we will take a closer look at the NOTEARS [87] approach introduced in Chapter 3. Originally, NOTEARS managed to approximately solve the left-hand side of Equation 5.33 by finding a local minimum of the continuous reformulation, which corresponds to the right-hand side of Equation 5.33,

$$\begin{aligned} \min_{W \in \mathbb{R}^{p \times p}} \quad & F(W) \\ \text{subject to} \quad & G(W) \in \texttt{DAGs} \end{aligned} \iff \begin{aligned} \min_{W \in \mathbb{R}^{p \times p}} \quad & F(W) \\ \text{subject to} \quad & h(W) = 0, \end{aligned} \tag{5.33}$$

where $F : \mathbb{R}^{p \times p} \to \mathbb{R}$ is a scoring function that quantifies the suitability of the matrix W given our data matrix $\mathbf{X} \in \mathbb{R}^{T \times p}$. The novelty of this approach is the introduction of a function $h(W)$ that is zero if and only if W is acyclic.

The authors of NOTEARS published their code on GitHub: `https://github.com/xunzheng/notears`. We can reuse most components of their code to apply the NOTEARS method for VAR(1) models. We only need to modify the scoring function F, as well as the function $h : \mathbb{R}^{p \times p} \to \mathbb{R}$ that captures the acyclicity of the coefficient matrix.

Changes to the scoring function F. The original scoring function F of NOTEARS is

$$F(W) = \ell(W; \mathbf{X}) + \lambda \|W\|_1 = \frac{1}{2T} \|\mathbf{X} - \mathbf{X}W\|_F^2 + \lambda \|W\|_1, \tag{5.34}$$

where $\|W\|_1$ corresponds to the sum of the absolute values of W. As we prefer sparse solutions for W as well, we can keep the sparsity penalty $\lambda \|W\|_1$. However, their defined Least Squares (LS) loss $\ell(W; \mathbf{X})$ is different for our model. As we are looking for a matrix W such that $X_{t-1,\cdot}W$ is close to $X_{t,\cdot}$, the LS loss in our setting is

$$\ell'(W; \mathbf{X}) = \frac{1}{2(T-1)} \left\| \mathbf{X}_{[2:T]} - \mathbf{X}_{[1:T-1]} W \right\|_F^2, \tag{5.35}$$

where $\mathbf{X}_{[i:j]}$ means that we include all p variables of the first i to j time steps, with both endpoints included. Consequently, the gradient of the least squares loss with respect to W is equal to

$$\nabla_W \ell'(W; \mathbf{X}) = -\frac{1}{T-1} \mathbf{X}_{[1:T-1]}^T \left\| \mathbf{X}_{[2:T]} - \mathbf{X}_{[1:T-1]} W \right\|_F, \tag{5.36}$$

where the T in subscript denotes the number of time steps, and the T in super script denotes the transpose. This yields the scoring function $F'(W)$ for our VAR(1) setting,

$$F'(W) = \ell'(W; \mathbf{X}) + \lambda \|W\|_1 = \frac{1}{2(T-1)} \left\| \mathbf{X}_{[2:T]} - \mathbf{X}_{[1:T-1]} W \right\|_F^2 + \lambda \|W\|_1. \tag{5.37}$$

Changes to the DAG-ness function $h(W)$. The function $h(W)$ from NOTEARS was designed such that a root of h corresponds to a directed acyclic graph,

$$h(W) = 0 \iff G(W) \text{ is a DAG.} \tag{5.38}$$

Originally, the function they proposed was defined as

$$h(W) = \operatorname{Tr}\left(e^{W \circ W}\right) - p = \sum_{k=1}^{\infty} \frac{1}{k!} \operatorname{Tr}\left((W \circ W)^k\right), \tag{5.39}$$

where $\circ$ represents the Hadamard-product, or the element-wise multiplication of two matrices of equal dimension. Furthermore, $\operatorname{Tr}(\cdot)$ represents the trace operator, which sums all elements on the diagonal.

We can interpret the function $h(W)$ as a weighted sum of all cycles in $W \circ W$ of length two or greater. Under this intuition, we see that $\operatorname{Tr}\left(e^{W \circ W}\right) = 0$ if and only if there are no cycles in the matrix $W \circ W$. If there are no cycles in $W \circ W$, then there are also no cycles in W. Such a function $h(\cdot)$ is also called a *Trace Exponential* function, corresponding to the third and fourth letters of the NO**TE**ARS acronym.

Casting such a function $h(\cdot)$ into our setting, recall that we do allow for self-loops in our VAR(1) model. Therefore, we need to adjust the function $h(W)$ accordingly. The required modification is quite simple. We simply set all diagonal entries of W to zero before evaluating $h(W)$. In other words, we propose the following function to enforce acyclicity for VAR(1) models:

$$h'(W) = \operatorname{Tr}\left(e^{\tilde{W} \circ \tilde{W}}\right) - p, \tag{5.40}$$

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

$$\tilde{w}_{ij} = \begin{cases} w_{ij} & \text{if } i \neq j \\ 0, & \text{if } i = j. \end{cases} \tag{5.41}$$

Changes to the set-up. Apart from these two fundamental changes, some other minor changes to the set-up were required for the NOTEARS method to be suitable for VAR(1) models. First of all, NOTEARS was designed for a linear SEM, similar to Definition 2.2. Therefore, the diagonal entries of W were constrained to zero by the optimization procedure. However, as we want the method to be able to estimate diagonal entries of W as well, we have removed these constraints from the optimization procedure. Lastly, we have added a method that simulates data according to our VAR(1) model.

Let us now consider two examples to showcase the performance of NOTEARS on VAR(1) models and how the regularization parameter λ can be used.

NOTEARS with acyclic W^*. Let us first consider a simple three-dimensional model where the coefficient matrix W^* that generates the data is acyclic.

Example 5.4 NOTEARS on a three-dimensional VAR(1) model with acyclic W^*.

Let us consider the data matrix $\mathbf{X} \in \mathbb{R}^{100 \times 3}$ visualized in Figure 5.3, consisting of three time series of one hundred time steps each. The data has been generated according to a VAR(1) model with data generating matrix

$$W^* = \begin{pmatrix} 0.50 & -0.65 & 0.00 \\ 0.00 & 0.50 & -0.25 \\ 0.00 & 0.00 & 0.40 \end{pmatrix}.$$

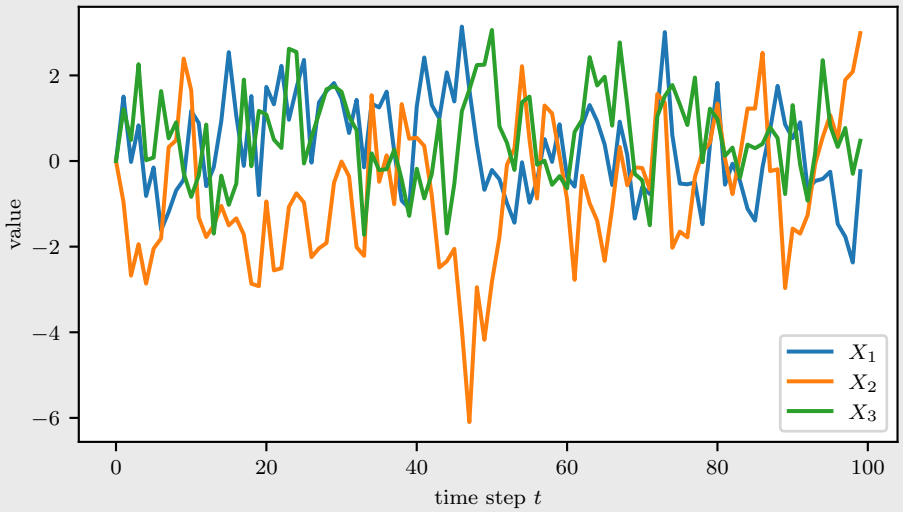


Figure 5.3: Visualization of the three variables X_1, X_2, X_2 of Example 5.4.

Applying the modified NOTEARS method using no regularization, we obtain the matrix

$$W_{\text{NOTEARS},\lambda=0} = \begin{pmatrix} 0.53 & -0.64 & 0.03 \\ 0.00 & 0.57 & -0.20 \\ 0.00 & 0.00 & 0.40 \end{pmatrix},$$

which indeed corresponds to a DAG under our definitions. Furthermore, we see that the coefficients of W are indeed quite close to the true coefficients of W^*.

If we increase the regularization parameter to $\lambda = 0.05$, we see that NOTEARS outputs

$$W_{\text{NOTEARS},\lambda=0.05} = \begin{pmatrix} 0.50 & -0.61 & 0.00 \\ 0.00 & 0.56 & -0.19 \\ 0.00 & 0.00 & 0.37 \end{pmatrix}.$$

Therefore, NOTEARS recovers the true support of W^* for $\lambda = 0.05$.

Note that due to the regularization penalty, all coefficients in $W_{\text{NOTEARS},\lambda=0.05}$ are biased slightly towards zero.

NOTEARS with cyclic W^*. Another interesting property of NOTEARS is that it is robust against model mismatches, that is, it is able to output a sensible acyclic matrix W, even when the data generating matrix W^* is cyclic. The reason for this is that the constraint $h'(W) = 0$ simultaneously enforces acyclicity, while we are also trying to minimize the loss function $F'(W)$. An example that showcases this robustness is discussed in Example 5.5.

Example 5.5 NOTEARS on three-dimensional VAR(1) data with cyclic W^*.

Consider a data matrix $\mathbf{X} \in \mathbb{R}^{3\times 100}$, consisting of three time series of one hundred time steps. The data has been generated according to a VAR(1) model with coefficient matrix

$$W^* = \begin{pmatrix} 0.35 & 0.40 & 0.00 \\ -0.50 & 0.30 & 0.00 \\ 0.00 & 0.00 & 0.50 \end{pmatrix}.$$

We see that W^* does not correspond to a cyclic structure, as both w^*_{12} and w^*_{21} are non-zero. The data matrix $\mathbf{X}$ has been visualized in Figure 5.4.

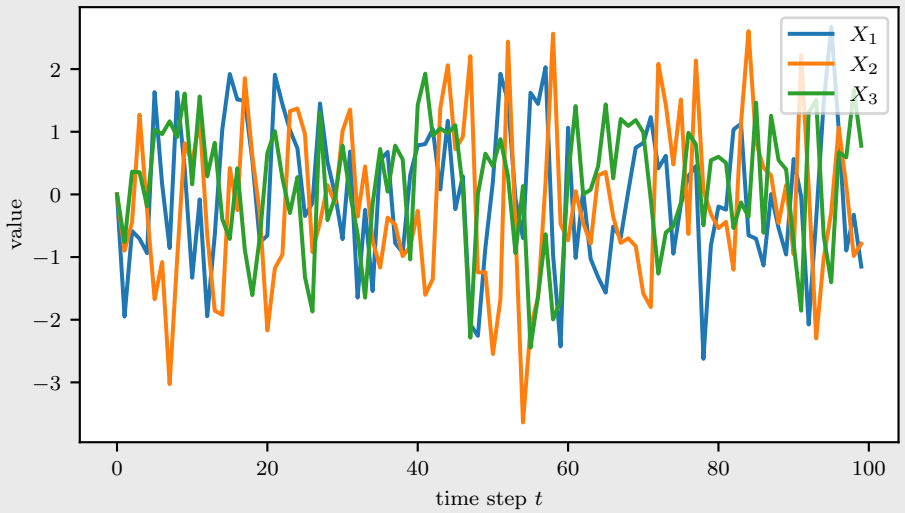


Figure 5.4: Visualization of the three variables X_1, X_2, X_2 of Example 5.5.

Applying NOTEARS without any regularization, so $\lambda = 0$, yields the matrix

$$W_{\text{NOTEARS},\lambda=0} = \begin{pmatrix} 0.28 & 0.00 & -0.17 \\ -0.49 & 0.32 & 0.00 \\ 0.00 & 0.00 & 0.32 \end{pmatrix}.$$

We see that we indeed already have a directed acyclic graph, so the acyclicity constraint forces NOTEARS to decide between setting either w_{21} or w_{12} to zero. We see that w_{13} is non-zero, whereas w^*_{13} was equal to zero.

If we increase the regularization parameter to $\lambda = 0.25$, then NOTEARS algorithm outputs the matrix

$$W_{\text{NOTEARS},\lambda=0.25} = \begin{pmatrix} 0.09 & 0.00 & 0.00 \\ -0.34 & 0.17 & 0.00 \\ 0.00 & 0.00 & 0.07 \end{pmatrix}.$$

We see that NOTEARS with $\lambda = 0.25$ manages to recover the true coefficients of W^*, except for the coefficient that violates the DAG-ness constraint. Therefore, it seems that NOTEARS also performs well on VAR(1) models, even with slight model mismatches.

5.3 Using a LASSO approach.

Another continuous approach to find a hopefully suitable directed acyclic graph $G(W)$ is by using a LASSO (Least Absolute Shrinkage and Selection Operator) approach, first introduced in 1996 [73]. Let us first describe the original regression setting.

Suppose we have a set of N paired p-dimensional features X and 1-dimensional labels Y in the form of $\{(x_i, y_i)\}_{i=1}^N$. Normally, LASSO is a regression technique that aims to solve, for a given $k \in \mathbb{R}$,

$$\beta_k = \underset{\beta \in \mathbb{R}^p}{\arg\min} \left\{ \frac{1}{N} \sum_{i=1}^{N} \left(y_i - x_i \beta^T\right)^2 \right\} \text{ subject to } \sum_{i=1}^{p} |\beta_i| \leq k. \tag{5.42}$$

Another way to look at Equation 5.42, is by consider its *Lagrangian* formulation. Now, for a given $\lambda \in \mathbb{R}$, rather than a given k, we are trying to find

$$\beta_\lambda = \underset{\beta \in \mathbb{R}^p}{\arg\min} \left\{ \frac{1}{N} \sum_{i=1}^{N} \left(y_i - x_i \beta^T\right)^2 + \lambda \|\beta\|_1 \right\}. \tag{5.43}$$

A great advantage of LASSO is that it has been studied extensively and has nice geometrical properties. Most importantly, the penalization of the $\|\beta\|_1$ provides a sparse solution for β, which is crucial to us as well as we are interested in recovering sparse structures.

LASSO in our setting. Our setting requires a coefficient matrix W rather than a coefficient vector β. Therefore, we should aim to solve

$$W_\lambda = \underset{W \in \mathbb{R}^{p \times p}}{\arg\min} \left\{ \frac{1}{T-1} \sum_{t=2}^{T} \|X_t - X_{t-1} W\|_2^2 + \lambda \|W\|_1 \right\}. \tag{5.44}$$

where $\|W\|\|_1$ represents the sum of the absolute value of the entries in W.

We enforce acyclicity by increasing the penalty parameter λ until the structure is acyclic. For small values for, we expect a dense cyclic matrix W. However, for large values for λ, we expect a sparse matrix W. Somewhere between these extremes exists a suitable value for λ, such that $G(W)$ is a directed acyclic graph and W fits $\mathbf{X}$ well.

To formalize, let W_λ be the solution to Equation 5.44 for a given data matrix $\mathbf{X}$ and a penalty parameter λ. We are looking for the smallest penalty parameter λ^* such that W_{λ^*} is acyclic,

$$\lambda^* = \min_{\lambda \geq 0} \lambda \text{ such that } W_\lambda \text{ from Equation 5.44 corresponds to a DAG.}$$

Finding λ^* using binary search. Assume we have an efficient algorithm to compute the solution of Equation 5.44. To find λ^*, We first solve Equation 5.44 for $\lambda = 1$. If $G(W_\lambda)$ is not acyclic, we know that λ^* is larger than 1. To find an upper bound for λ^*, we simply double λ at every iteration until $G(W_\lambda)$ is a DAG, indicating that λ^* is contained in the interval $(l, r] = (\frac{\lambda}{2}, \lambda]$.

To get arbitrarily close to λ^*, we iteratively half the interval and continue with the interval in which λ^* is contained. If $G(W_\lambda)$ is a DAG for $\lambda = (l + r)/2$, we know that λ^* is contained in the lower half of the interval, and hence, we continue with $(l, \lambda]$. If $G(W_\lambda)$ is not a DAG, we know that λ^* resides in the upper half of the interval and we continue with $(\lambda, r]$. We continue this process until the interval is of arbitrarily small length, say of a width smaller than ϵ. We then return W_r. The pseudocode for this binary search approach, which we call `DAG-LASSO`, is given in Algorithm 5.3.

Algorithm 5.3: `DAG-LASSO`$(\mathbf{X}, \epsilon)$

Input: A data matrix $\mathbf{X} \in \mathbb{R}^{T \times p}$, maximum distance of ϵ to the true λ^*.
Output: A penalty value λ such that $G(W_\lambda)$ is a DAG and λ is at less than ϵ away from the smallest λ^* such that $G(W^*_\lambda)$ is a DAG.

```
l ← 0
r ← 1
Compute W_r using Equation 5.44
while G(W_r) is not a DAG do                ▷ Find an upperbound for λ*
    l ← r
    r ← 2r
    Compute W_r using Equation 5.44
end while
λ ← r
while r − l ≥ ϵ and W_λ is not a DAG do     ▷ Iteratively half the interval (l, r]
    λ ← (r+l)/2
    Compute W_λ using Equation 5.44
    if G(W_λ) is a DAG then
        r ← λ
    else
        l ← λ
    end if
end while
return W_r using Equation 5.44.
```

Number of required iterations. The first while loop will run $\max\{\lceil \log_2(\lambda^*) \rceil, 0\}$ times. In the second while loop, we half the width of the interval at each iteration, which is initially of size $r - l = \frac{r}{2}$, where $r = \max\{2^{\lceil \log_2(\lambda^*) \rceil}, 1\}$. Therefore, the width of the interval is $w = \max\{2^{\lceil \log_2(\lambda^*) \rceil - 1}, 1\}$ If we require an interval width of size ϵ, then the second while loop will be executed a total of $\lceil \log_2(w/\epsilon) \rceil$ times. Therefore, after approximately $\mathcal{O}\left(\log_2(\lambda^*/\epsilon)\right)$ iterations, we will have a value for λ that is less than ϵ away from the true value of λ^*.

Solution Path of `LASSO-DAG`. A common visualization of the LASSO algorithm is the so-called *solution path* [74]. We can plot each of the values of the coefficients in W_λ for each value for λ.

For $\lambda = 0$, we expect all coefficients to be quite large, and for a large enough value for λ, all coefficients in the matrix W_λ will be equal to zero, as the penalization is so severe. For all values for λ inbetween, we can plot all the values of all coefficients of W_λ. We expect each coefficient in W_λ to converge to zero as a function of λ. Two examples of such solution paths are given in Example 5.6 and Example 5.7.

Interestingly, the lasso-path of a coefficient w_{ij} is *continuous* and *piecewise linear*. A more formal description of the solution path for the LASSO is given in [74]. In our scenario, changing a value in one column of W_λ only affects the other values in the same column. Therefore, we expect kinks in the solution path of w_{ij} only when another arc w_{kj} has been shrunk to zero.

Furthermore, the rate at which the coefficients converge to zero is indicative of the importance of the corresponding arc. If a coefficient is small, yet converges to zero slowly, then this arc is quite important.

`DAG-LASSO` with acyclic coefficient matrix W^*. Let us assess the performance of `DAG-LASSO` in Example 5.6, where the data has been generated using an acyclic coefficient matrix W.

Example 5.6 `DAG-LASSO` on three-dimensional data with acyclic W^*.

To showcase how `DAG-LASSO` works, let us consider an example based on a data matrix $\mathbf{X} \in \mathbb{R}^{100\times 3}$ consisting of three time series of one hundred time steps. The data $\mathbf{X}$ is exactly the same as in Example 5.4, which has been generated according to a VAR(1) model using the data generating matrix W^* given in the left-hand side of Equation 5.45.

Now, for $\lambda = 0$, we simply get the maximum likelihood estimator which coincides with the ordinary least squares estimator W_{OLS} in the right-hand side of Equation 5.45.

$$W^* = \begin{pmatrix} 0.50 & -0.65 & 0.00 \\ 0.00 & 0.50 & -0.25 \\ 0.00 & 0.00 & 0.40 \end{pmatrix}, \qquad W_{\text{OLS}} = \begin{pmatrix} 0.50 & -0.62 & 0.01 \\ -0.02 & 0.54 & -0.16 \\ -0.14 & -0.11 & 0.32 \end{pmatrix}. \tag{5.45}$$

We see that W_{OLS} does not correspond to an acyclic structure.

Increase the regularization penalty to $\lambda = 0.1$ yields $W_{\lambda=0.1}$, which has been given on the left-hand side of Equation 5.46. Although many coefficients in $W_{\lambda=0.1}$ are close to zero, its structure is still not acyclic, indicating that $\lambda = 0.1$ is not large enough to enforce acyclicity.

Applying `DAG-LASSO` with $\epsilon = 0.01$ suggests that $\lambda^* = 0.14$, resulting in the coefficient matrix W_{λ^*} on the right-hand side of Equation 5.46.

$$W_{\lambda=0.1} = \begin{pmatrix} 0.50 & -0.62 & 0.00 \\ -0.02 & 0.52 & -0.13 \\ -0.03 & -0.03 & 0.24 \end{pmatrix}, \qquad W_{\lambda^*=0.14} = \begin{pmatrix} 0.39 & -0.54 & 0.00 \\ 0.00 & 0.51 & -0.11 \\ 0.00 & 0.00 & 0.21 \end{pmatrix}. \tag{5.46}$$

`DAG-LASSO` recovers the true support of W^*, although the values are slightly biased towards zero. Let us also investigate the solution path of the coefficients of W_λ, which has been given in Figure 5.5.

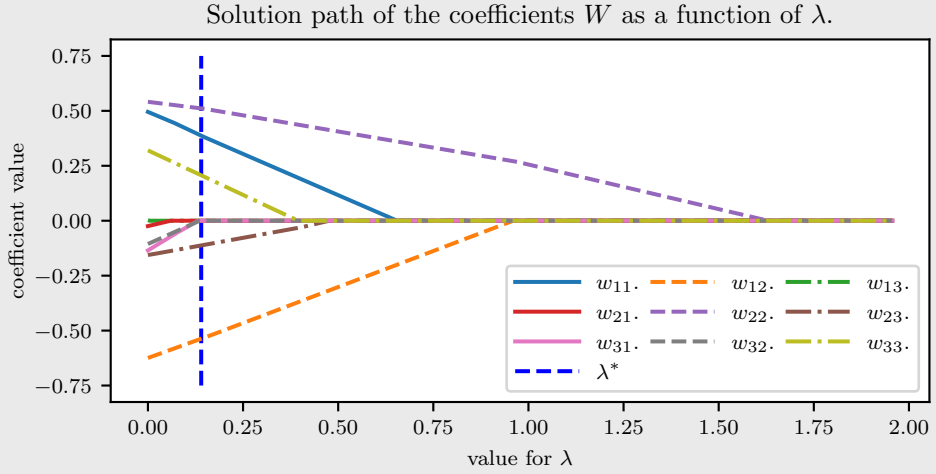


Figure 5.5: Visualization of the solution path for the coefficients of W_λ of Example 5.6.

The blue vertical line in Figure 5.5 indicates that acyclicity is first established for $\lambda^* = 0.14$. The four unimportant arcs w_{13}, w_{21}, w_{31}, and w_{32} have been shrunk to zero, yielding an acyclic coefficient matrix W.

The solution path of a coefficient is also a suitable indicator of its importance. The coefficient w_{23} has the same order of magnitude as the four unimportant arcs in W_{OLS}. Nevertheless, the solution path of w_{23} is less steep than the other arcs and has been shrunk to zero only for $\lambda \approx 0.68$. Even more, w_{23} has been shrunk to zero later than w_{33}, indicating that w_{23} is more important than w_{33}, despite being twice as small in magnitude.

`LASSO-DAG` with a cyclic coefficient matrix W^*. Let us now assess `DAG-LASSO` when we violate the acyclicity assumption by introducing an additional arc.

Example 5.7 `DAG-LASSO` on three-dimensional data with model mismatch.

Consider the same three-dimensional time series with one hundred time steps as in Example 5.5. The coefficient matrix W^* is provided on the left-hand side of Equation 5.47.

$$W^* = \begin{pmatrix} 0.35 & 0.40 & 0.00 \\ -0.50 & 0.30 & 0.00 \\ 0.00 & 0.00 & 0.50 \end{pmatrix}, \qquad W_{\lambda=0} = \begin{pmatrix} 0.28 & 0.42 & -0.18 \\ -0.51 & 0.32 & 0.00 \\ -0.08 & 0.00 & 0.31 \end{pmatrix}. \tag{5.47}$$

Let us first remark that W^* does not correspond to an acyclic structure as w_{12} and w_{21} are non-zero, in fact even quite large. For $\lambda = 0$, we get the matrix $W_{\lambda=0}$ in the right-hand side of Equation that resembles W^* but is also cyclic.

Therefore, let us consider what happens for an already quite large penalty parameter $\lambda = 0.25$, and for the smallest penalty parameter such that W_λ corresponds to an acyclic structure, $\lambda^* = 0.51$. Both are given in Equation 5.48.

$$W_{\lambda=0.25} = \begin{pmatrix} 0.09 & 0.21 & 0.00 \\ -0.36 & 0.16 & 0.00 \\ 0.00 & 0.00 & 0.05 \end{pmatrix}, \qquad W_{\lambda^*=0.51} = \begin{pmatrix} 0.00 & 0.00 & 0.00 \\ -0.19 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.00 \end{pmatrix}. \tag{5.48}$$

The matrix W_{λ^*} does not resemble W^* anymore, except that its non-zero coefficient corresponds to the seemingly most important coefficient w^*_{21}.

Let us investigate the solution path of `DAG-LASSO` applied on our data matrix $\mathbf{X}$, which has been plotted in Figure 6.3.

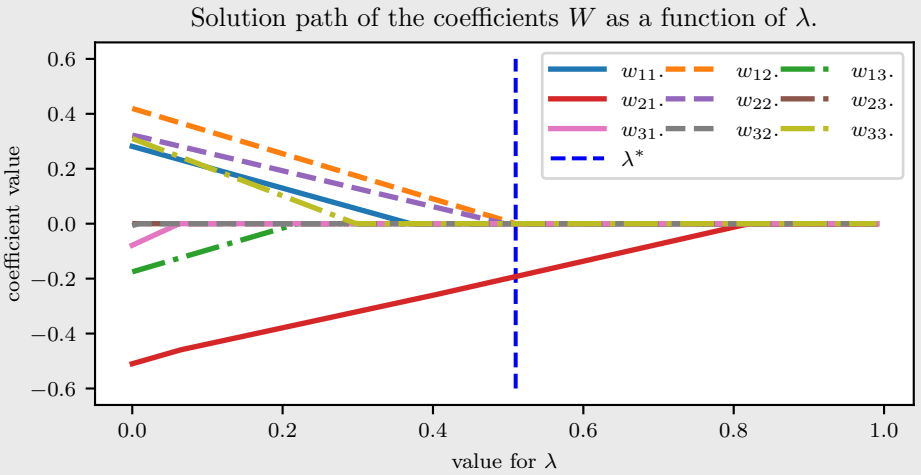


Figure 5.6: Visualization of the solution path for the coefficients of W_λ of Example 5.7.

The three unimportant edges w_{23}, w_{31}, and w_{32} converge to zero quite fast, and for $\lambda \approx 0.25$ we recover the true structure. To retrieve an acyclic structure, we first shrink w_{33}, w_{11}, and w_{22} to zero, such that only the cycle (w_{12}, w_{21}) remains. Now, a "death-match" between the two remaining arcs commences, after which we are left with an acyclic matrix W_{λ^*} at $\lambda^* \approx 0.51$ only containing one non-zero coefficient.

Disadvantages We see that `DAG-LASSO` indeed finds an acyclic structure by using a progressively increasing LASSO penalty. However, there are two disadvantages of using this approach.

One small disadvantage is that the coefficients in W_{λ^*} are *biased* towards zero, as we have seen in both Example 5.6 and Example 5.7. We can overcome this bias by re-estimating the non-zero coefficients of W_{λ^*}, as also highlighted in [30].

So, define the support of W_{λ^*} as

$$\text{supp}(W) = \{(i, j) \mid w_{ij} \neq 0\}. \tag{5.49}$$

We are interested in the matrix $\hat{W}$ that maximizes the likelihood given $\mathbf{X}$, where we can only estimate the coefficients in the support of W_{λ^*}. This constrained maximum likelihood estimator $\hat{W}$ corresponds to the constrained least square solution, where we can only regress the time-lagged variables corresponding to the non-zero coefficients in $\text{supp}(W_{\lambda^*})$. We can do this in a similar approach as the `Order-OLS` algorithm described in Section 4.1.

To estimate the coefficients to predict the jth variable, we can only use the variables $X_{\cdot,i}$ such that $(i, j) \in \text{supp}(W_{\lambda^*})$. Let $\mathbf{X}'_j \in \mathbb{R}^{T-1 \times p}$ be the constrained feature matrix where the ith column of $\mathbf{X}'_j$ consists of the first $T-1$ time steps of variable $X_{\cdot,i}$ if $(i, j) \in \text{supp}(W_{\lambda^*})$, and all zeros otherwise. Furthermore, let $Y_{\cdot,j} \in \mathbb{R}^{T-1}$ be a column vector corresponding to the last $T-1$ time steps of variable j. Then, the columns of the constrained ordinary least squares estimate $\hat{W}$ are equal to

$$\hat{W}_{\cdot,j} = \left(\mathbf{X}'_j {\mathbf{X}'_j}^T\right)^{-1} {\mathbf{X}'_j}^T Y_{\cdot,j}, \qquad j = 1, \ldots, p. \tag{5.50}$$

This coefficient matrix maximizes the likelihood, or equivalently minimizes the mean squared error, where only the support of W_{λ^*} could be re-estimated.

A second and more serious disadvantage is that `DAG-LASSO` does not provide a sensible solution when the data-generating matrix W^* is cyclic, as we have seen in Example 5.7. The value for λ will be increased until the structure is acyclic, but one cycle of length two can cause the inferred structure to consist of only a single arc.

Chapter 6

Iterative Approaches

This is the third and last methodological chapter where we introduce approaches that learn an acyclic structure of a data matrix $\mathbf{X}$. In Chapter 4 we have seen that we can decompose our coefficient matrix W into a permutation matrix P and an upper triangular matrix U, and consequently optimize over the discrete space of permutation matrices. In Chapter 5, we have discussed approaches that enforce acyclicity through continuous constraints.

The methods discussed in the previous two chapters estimate all desired parameters in the coefficient matrix W simultaneously. In Chapter 4, we simultaneously estimated all $p(p+1)/2$ coefficients in the upper triangular matrix U. In Chapter 5, we also estimated all parameters at the same time subject to some continuous constraints. The aforementioned methods were *global* in the sense that they estimated the graphical model as a whole, without considering any arcs individually. An alternative perspective is that the aforementioned methods were *single-step* methods, as the estimation procedure only consisted of one single step.

In this chapter, we will discuss methods that employ a *local* approach, where we *iteratively* construct an acyclic coefficient matrix by updating one arc at the time, hence the name "iterative approaches". Another name would be *multi-step* approaches, as one arc is updated at each step.

Several iterative procedures exist. The first choice is the *direction* of the iterative approach. We can start with an empty graph and iteratively add arcs that maximize some selection criterion. These are called *forward* iterative approaches. Secondly, we can iteratively update the coefficient matrix in the opposite direction, yielding a *backward* approach. We can start with a fully connected graph and iteratively remove the arc that maximizes some deletion criterion. These two directions can be combined in a forward-backward approach, where we first construct some coefficient matrix after which we remove some arcs that were redundant in hindsight.

An advantage of these iterative approaches is that we can enforce acyclicity step-by-step. Instead of discovering that the fully estimated matrix does not correspond to an acyclic structure, the step-by-step approach allows us to pinpoint exactly *when* and *where* acyclicity was violated.

For the single direction approaches, another advantage is that they are quite efficient. As soon as an arc has been added, this arc will remain in the structure. In the opposite direction, when an arc has been removed, this arc will never reappear in the structure. Now, if we can evaluate our selection or deletion criterion efficiently, such an iterative approach might be more tractable for a large number of variables. Lastly, we can interpret the order in which arcs are selected as an ordering of the importance of the arcs. The sooner an arc is added, the more important we consider it to be. As we will see later, such an ordering helps in fine-tuning the graphical model such that we only keep arcs that are considered important.

However, there are several drawbacks to iterative approaches that should also be addressed. First of all, such iterative approaches are unable to recover from an arc addition or deletion that was a bad choice in hindsight. Furthermore, the fact that one solitary arc may seem important at one iteration does not imply that the arc is important in the global sense. When we jointly search for two arcs, the notion of "important" can radically change. Therefore, this notion of importance is at most a heuristic and should be taken with a grain of salt.

Outline of this chapter. In this chapter, we will be discussing various iterative procedures, both in the forward and backward direction. Furthermore, we will be investigating different selection and deletion criteria.

In Section 6.1, we will be discussing a forward iterative procedure that uses the correlation with the current residual as the selection criterion. The linear regression variant is called Orthogonal Matching Pursuit. In Section 6.2, we will be investigating a backward iterative procedure that uses the coefficient size as the removal criterion. In Section 6.3, we will be discussing other iterative procedures, as well as a trick to improve the performance of backward iterative approaches.

We will be investigating how we should decide on a suitable complexity of the structure in Section 6.4. A model with too few arcs does not capture all the useful relations, but too much arcs results in an unnecessarily complex model. We will be investigating bootstrapping approaches in Subsection 6.4.1 and a leave-one-out cross-validation approach in Subsection 6.4.2.

Several cross-validation techniques exists for time series, such as block cross-validation. However, such a technique is rather clunky as it relies on certain assumptions such as stationarity and parameter choices such as the block size. Therefore, we will be investigating a leave-one-out cross-validation approach although it seems inadequate for time series. We demonstrate in Section 6.5 using a small simulation study and a theoretical analysis that this approach seems appropriate for deciding on a sensible complexity of the structure.

Determining whether the structure of W is acyclic. All iterative approaches require an efficient method to determine whether the structure characterized by W is acyclic. The first method that is most readily available is to use a suitable trace exponential function, as discussed in Section 5.2. A suitable trace exponential function would be

$$h(W) = \operatorname{Tr}\left(e^{\tilde{W}}\right) - p = \sum_{k=1}^{\infty} \frac{1}{k!} \tilde{W}^k, \tag{6.1}$$

where $\tilde{W}$ corresponds to the matrix W with the diagonal entries set to zero, and p corresponds to the number of variables. The structure is acyclic if and only if $h(W) = 0$. Computing the trace exponential can be done in $\mathcal{O}(p^3)$ running time [1], so it is not the most efficient approach, but suitable implementations are readily available in standard code libraries.

Secondly, we can detect cycles by verifying the existence of a *topological ordering* π compatible with the induced graph $G(W)$. We say an ordering π is compatible if, for any pair of variables X_i, X_j, there there does not exist an arc from node X_j to node X_i if X_i precedes the X_j in the topological ordering. This notion of a compatible ordering is in fact equivalent to the permutation π or permutation matrix P discussed in Chapter 4. Therefore, we can find such a topological ordering if and only if the structure is acyclic.

There exists efficient implementations to compute a topological ordering in $\mathcal{O}(p^2)$. We can first remove all self-loops from the graph, as they do not violate the acyclicity assumption in our setting. We then look for a node with no incoming arcs. If no such node exists, then there must be a cycle in our graph. If there exists a node with no incoming arcs, then we can remove this node and all its outgoing arcs, and this node will be the first node in our topological ordering. We continue this process and if at some stage we are unable to find a node without any incoming arcs, then there exists a cycle in the graph. If we can continue this process until there are no nodes left, then we know there exists a topological ordering and hence, the structure is acyclic.

If we consider the corresponding coefficient matrix W, then a node j having no incoming arcs is equivalent to finding a column j of W which contains all zeros. Additionally, removing this node j from the graph is equivalent to deleting the jth row and column of W.

Pseudocode to verify the existence of a topological ordering of the variables given a coefficient matrix W is provided in Algorithm 6.1. Such an algorithm is more frequently known as a topological sort, and this approach is quite similar to Kahn's algorithm [38].

Algorithm 6.1: Topological sort.

Input: A coefficient matrix $W \in \mathbb{R}^{p \times p}$ corresponding to a directed graph $G(W)$.
Output: A topological ordering $\pi = (\pi_1, ..., \pi_p)$ of the nodes $i = 1, \ldots, p$ if W corresponds to an acyclic structure, or False if W corresponds to a cyclic structure.

1: remove self-loops of W
2: initialize π as an empty list
3:
4: **for** $i = 1$ until p **do**
5: find a node j with no incoming arcs. ▷ jth column of W contains all zeros
6: **if** no such node is found **then**
7: **return** False ▷ $G(W)$ is not a DAG
8: **else**
9: append such a node j to topological ordering π.
10: remove node j and its outgoing arcs ▷ delete jth row and column of W
11: **end if**
12: **end for**
13:
14: **return** the topological ordering π ▷ $G(W)$ is a DAG

Example 6.1 Topological sort on two adjacency matrices W_1 and W_2.

To see how this algorithm works, let us consider two 4-dimensional adjacency matrices

$$W_1 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 \end{pmatrix}, \qquad W_2 = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \end{pmatrix}.$$

The corresponding graphs are visualized in Figure 6.1, where we have already removed all self-loops, according to line 1 of Algorithm 6.1, for convenience and conciseness of the graphs.

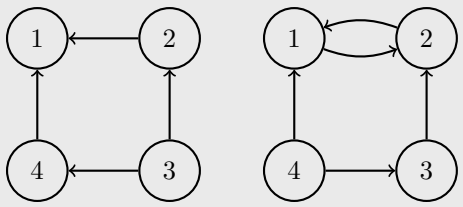


Figure 6.1: Visualization of the graphs induces by the coefficient matrices W_1 (left) and W_2 (right) from Example 6.1. Self-loops of both graphs have been removed.

After removing the self-loops of W_1, we can remove node 3 and add it to our topological ordering. Then, we can remove either node 2 or node 4. For consistency, let us select the node that comes first lexicographically. Next, we can remove node 4, and then node 1, yielding a topological ordering $\pi = (3, 2, 4, 1)$, so that W_1 corresponds to an acyclic structure. A topological ordering need not be unique, as $\pi = (3, 4, 2, 1)$ would also be valid.

For W_2, we see that after removing self-loops, we can first remove the fourth node, and then the third node. However, both node 1 and node 2 have an incoming arc from each other. Hence, we cannot find a topological ordering so we conclude that W_2 does not correspond to an acyclic structure.

Now that we have an efficient method to determine whether a coefficient matrix W characterizes the structure of a directed acyclic graph, let us discuss several iterative methods.

6.1 Using Orthogonal Matching Pursuit

Orthogonal Matching Pursuit (OMP) is an algorithm nowadays commonly known as a sparse regression technique, but was popularized in [46] as a sparse signal recovery algorithm to recover an encoded signal $x \in \mathbb{R}^n$ from a noisy compressed vector $y \in \mathbb{R}^m$, where $m \ll n$. A large signal x has been compressed using a compression matrix $A \in \mathbb{R}^{m \times n}$ as

$$y = Ax. \tag{6.2}$$

The goal now is to recover x given A and y. To recover x, we this we need to solve a system of m linear equations with n unknowns. When $m < n$, the system of linear equations is underdetermined, meaning there are infinitely many solutions for x. Therefore, recovering the true x is impossible.

Although there are infinitely many solutions x', we can try to find a solution $\hat{x}$ that is maximally sparse, meaning that of all x' such that $Ax' = y$, the solution $\hat{x}$ contains the fewest non-zero coefficients. To find $\hat{x}$, we need to solve the problem

$$\hat{x} = \arg\min_x ||x||_0 \text{ such that } Ax = y. \tag{6.3}$$

Where $||x||_0$ represents the number of non-zero coefficients in x.

However, with real life data, it is often not realistic to assume that Ax is *exactly* equal to y. Therefore, we allow for a small tolerance ϵ to cope with the noise in the measurements. The problem then becomes, for a given tolerance ϵ,

$$\hat{x} = \arg\min_x ||x||_0 \text{ such that } ||Ax - y||_2 \leq \epsilon. \tag{6.4}$$

Both problems in Equation 6.3 and Equation 6.4 are in fact NP-hard, see [25] Problem MP-5, page 246.

Therefore, instead of trying to find the $\hat{x}$, researchers have developed algorithms that approximately solve the problem in Equation 6.4. An effective method to approximately solve this sparse signal recovery problem is the aforementioned OMP approach.

OMP in the linear regression setting. Apart from the signal processing community, OMP is now also widely used in machine learning, even in for example text classification [67] and image reconstruction [28]. Suppose that we have a data matrix $X \in \mathbb{R}^{T \times p}$ of explanatory variables and a response variable $y \in \mathbb{R}^T$. Suppose we want to perform a linear regression in the form of

$$y = Xw + \varepsilon, \tag{6.5}$$

where ε represents a zero-mean random noise variable independent of X.

Now, to obtain a sparse coefficient vector w, then OMP is an effective approach, although it is now used in a different setting. We do not necessarily have that y is a compressed version of w, and the size of w is generally smaller than the size of y, which was the other way around in the signal recovery setting.

Tropp [75] has shown in the noiseless setting that OMP will exactly recover the true coefficients in w, assuming the following condition holds. Let X_{OPT} represent the columns of X that are included in the optimal representation of the signal, and $X_{\text{N-OPT}}$ the columns that are not included. Then, OMP will recover the correct coefficients in w if

$$\max_{\text{columns } X_i \text{ of } X_{\text{N-OPT}}} \left\| \left(X_{\text{OPT}}^T X_{\text{OPT}} \right)^{-1} X_{\text{OPT}}^T X_i \right\|_1 < 1. \tag{6.6}$$

Additionally, Zhang has shown in [86] for the setting with noise that OMP recovers all true coefficients under some more involved assumptions, such as sufficiently large coefficient sizes and a suitable stopping criterion.

The OMP algorithm. To recover a sparse coefficient vector w, the Orthogonal Matching Pursuit algorithm starts with an empty coefficient vector $w^{(0)}$, where the number in the superscript denotes the current iteration of the algorithm. Furthermore, we denote $\Lambda^{(0)}$ as the support of $w^{(0)}$, containing all indices that correspond to non-zero entries in $w^{(0)}$. We then calculate which explanatory variable $X_i \in \mathbb{R}^T$, or equivalently which column i of X is most correlated with the current residual $r^{(0)} = y - Xw^{(0)} = y$. Most correlated implies that the angle between the vectors X_i and $r^{(0)}$ is closest to either 0 degrees or 180 degrees. The index of the corresponding column vector can be calculated as

$$i^{(1)} = \arg\max_i \left| \left\langle \tilde{X}_i, r^{(0)} \right\rangle \right|, \qquad \text{where } \tilde{X}_i = \frac{X_i}{\|X_i\|_2}. \tag{6.7}$$

As $\left\|r^{(0)}\right\|_2$ is the same for all i variables, we need not include this quantity in Equation 6.7.

We add the index $i^{(1)}$ to the support set $\Lambda^{(1)}$ and update the coefficient vector $w^{(1)}$ by computing the ordinary least squares solution with indices restricted to $\Lambda^{(1)}$. This estimate can be calculated by first defining $X^{(1)}$ as our matrix of explanatory variables X, but with all columns X_j of $X^{(1)}$ set to zero if $j \notin \Lambda^{(1)}$. Then, the ordinary least squares estimate corresponds to

$$w^{(1)} = \left({X^{(1)}}^T X^{(1)} \right)^{-1} {X^{(1)}}^T y. \tag{6.8}$$

We update the residual as $r^{(1)} = y - Xw^{(1)}$ and again find the column of X most correlated with the current residual, add its index to the support set, and re-estimate the coefficient vector. We repeat this process for multiple iterations $k = 2, \ldots$, until all correlations are smaller than some threshold ϵ. We then return the coefficient vector $w^{(k-1)}$, consisting of $k-1$ non-zero coefficients.

The pseudocode of orthogonal matching pursuit is described in Algorithm 6.2.

Algorithm 6.2: Original Orthogonal Matching Pursuit algorithm

Input: Explanatory variables $X = [X_1, \ldots, X_p] \in \mathbb{R}^{T \times p}$, response variable $y \in \mathbb{R}^T$, a threshold ϵ.
Output: A coefficient vector $w \in \mathbb{R}^p$.

$\tilde{X}_j \leftarrow X_j / ||X_j||_2$, the normalized basis for $j = 1, ..., p$.
$\Lambda^{(0)} \leftarrow \varnothing$
$w^{(0)} \leftarrow \mathbf{0} \in \mathbb{R}^p$
$r^{(0)} \leftarrow y - Xw^{(0)}$

for $k = 1, 2, \ldots$ **do**
  $i^{(k)} \leftarrow \arg\max_i \left| \langle \tilde{X}_i, r^{(k-1)} \rangle \right|$ ▷ find column X_i most correlated with $r^{(k-1)}$
  if $\max_i \left| \langle \tilde{X}_i, r^{(k-1)} \rangle \right| > \epsilon$ **then** ▷ update $\Lambda^{(k-1)}$, $w^{(k-1)}$, $r^{(k-1)}$
    $\Lambda^{(k)} \leftarrow \Lambda^{(k-1)} \cup \{i^{(k)}\}$
    $X^{(k)} \leftarrow X$, with column $X_j^{(k)}$ all zeros if $j \notin \Lambda^{(k)}$.
    $w^{(k)} \leftarrow \left(X^{(k)} {X^{(k)}}^T \right)^{-1} {X^{(k)}}^T y$ ▷ OLS solution restricted to $\Lambda^{(k)}$
    $r^{(k)} \leftarrow y - Xw^{(k)}$
  else
    return $w^{(k-1)}$
  end if
end for

Casting orthogonal matching pursuit to our setting. Algorithm 6.2 is suitable for a coefficient vector $w \in \mathbb{R}^p$ and a response vector $y \in \mathbb{R}^T$. However, we consider a coefficient matrix $W \in \mathbb{R}^{p \times p}$ and a response matrix $\mathbf{X}_{2:T,\cdot} \in \mathbb{R}^{T-1 \times p}$ consisting of p response variables,

$$\mathbf{X}_{2:T,\cdot} = \mathbf{X}_{1:T-1,\cdot} W + \varepsilon_{2:T}. \tag{6.9}$$

We will propose three possible modifications to the OMP algorithm.

Approach 1: p independent OMP methods One approach is to separately apply the OMP algorithm p times, once for each response variable, thereby obtaining the p columns $w_{\cdot,j}$ of W. In our setting, we have for one of the j time series that

$$y_j = \mathbf{X}_{2:T,j} \in \mathbb{R}^{T-1}, \qquad X = \mathbf{X}_{1:T-1,\cdot} \in \mathbb{R}^{T-1 \times p}, \qquad w_j = w_{\cdot,j} \in \mathbb{R}^p. \tag{6.10}$$

However, we lose the dependency between the p columns. This becomes impractical later as we require the coefficient matrix W to be acyclic. For this, we need to consider the coefficients of all columns simultaneously.

Approach 2: cast our matrix notation to vector notation. A more suitable approach is to cast our matrix notation to vector notation. That is, we construct a response vector y', explanatory variables X', and a coefficient vector w' such that

$$\mathbf{X}_{2:T,\cdot} - \mathbf{X}_{1:T-1,\cdot} W = \varepsilon_{2:T} \in \mathbb{R}^{T-1 \times p} \iff y' - X'w' = \text{vec}(\varepsilon_{2:T}), \tag{6.11}$$

where the the vectorized dimensions will be

$$y' \in \mathbb{R}^{(T-1)\cdot p}, \qquad X' \in \mathbb{R}^{(T-1)\cdot p \times p^2}, \qquad w' \in \mathbb{R}^{p^2}. \tag{6.12}$$

The complete derivation to vectorize the the matrix notation has been given in Appendix A.1.

A small example how on rewriting this matrix notation to a vectorized setting is given in Example 6.2

Example 6.2 Rewrite our matrix formulation to a vectorized setting.

To explain the rewriting of our matrix notation to vector notation more clearly, we will consider the following example. Suppose that we have four samples of two variables in our data matrix $\mathbf{X}$. Hence, $T = 4$ and $p = 2$, and $\mathbf{X} \in \mathbb{R}^{4\times 2}$.

As our response variables, we will use the first $T-1$ time steps of $\mathbf{X}$, which we define as $\mathbf{X}' \in \mathbb{R}^{T-1\times p}$. Furthermore, we use the last $T-1$ time steps as explanatory variables, which we define as $\mathbf{Y} \in \mathbb{R}^{T-1\times p}$.

The data matrix $\mathbf{X}$ that we will be using in this example, as well as the corresponding covariate matrix $\mathbf{X}'$ and response matrix $\mathbf{Y}$ have been given in Equation 6.13

$$\mathbf{X} = \begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \\ 7 & 8 \end{pmatrix}, \qquad \mathbf{X}' = \begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix}, \qquad \mathbf{Y} = \begin{pmatrix} 3 & 4 \\ 5 & 6 \\ 7 & 8 \end{pmatrix}. \tag{6.13}$$

Let us now cast this matrix setting to a vectorized setting.

First, we get y' by vectorizing $\mathbf{Y}$, yielding

$$y' = (3, 5, 7, 4, 6, 8)^T \in \mathbb{R}^{T-1\cdot p}.$$

To get X', we repeat the block $\mathbf{X}'$ a total of p times along the diagonal, yielding

$$X' = \begin{pmatrix} \mathbf{X}' & \mathbf{O} \\ \mathbf{O} & \mathbf{X}' \end{pmatrix} = \begin{pmatrix} 1 & 2 & 0 & 0 \\ 3 & 4 & 0 & 0 \\ 4 & 5 & 0 & 0 \\ 0 & 0 & 1 & 2 \\ 0 & 0 & 3 & 4 \\ 0 & 0 & 5 & 6 \end{pmatrix} \in \mathbb{R}^{(T-1)\cdot p \times p^2}.$$

The corresponding coefficient vector w' now exactly corresponds to the vectorized coefficient matrix $\text{vec}(W)$.

Approach 3: Extending the OMP algorithm itself. Approach 2 suffers from the drawback that X' is redundantly copied a total of p times. Furthermore, the memory required to allocate X' is cubic with respect to the number of variables, which is rather wasteful.

Therefore, it is more practical to extend the OMP algorithm such that we can estimate a coefficient matrix W, rather than a coefficient vector w. Instead of one index i, we now have two indices i and j, corresponding to the ith row of the jth column of W. This extension yields p support sets $\Lambda_j^{(k)}$, coefficient vectors $w_j^{(k)}$, and residuals $r_j^{(k)}$. Here, the subscript j denotes to which response variable they belong.

Additionally, as we now have p residuals $r_i^{(k)}$, we cannot exclude $\|r_i^{(k)}\|_2$ as in Equation 6.7 to find the explanatory variable X_i most correlated with the residual $r_j^{(k)}$. The new selection criterion is

$$\left(i^{(k-1)}, j^{(k)}\right) = \arg\max_{i,j} \frac{\left|\left\langle \mathbf{X}_{1:T-1,i}, r_j^{(k-1)} \right\rangle\right|}{\|\mathbf{X}_{1:T-1,i}\|_2 \, \|r_j^{(k-1)}\|_2} \tag{6.14}$$

We only need to re-estimate the coefficient column $w_j^{(k)}$ and recompute the residuals $r_j^{(k)}$, which can be done in the same way as in the standard OMP algorithm. The other support sets, coefficient columns, and residuals remain unchanged for $j' \neq j^{(k)}$.

The pseudocode for the orthogonal matching pursuit algorithm extended to multiple response variables has been given in Algorithm 6.3. To apply it to our setting, the explanatory variables corresponds to $\mathbf{X}_{1:T-1,\cdot}$, and the response variables corresponds to $\mathbf{X}_{2:T,\cdot}$.

Algorithm 6.3: Orthogonal Matching Pursuit algorithm for multiple response Variables

Input: explanatory variables $X = [\mathbf{X}_{1:T-1,1}, \ldots, \mathbf{X}_{1:T-1,p}] \in \mathbb{R}^{T-1 \times p}$, response matrix $Y = [\mathbf{X}_{2:T,1}, \ldots, \mathbf{X}_{2:T,p}] \in \mathbb{R}^{T-1 \times p}$, threshold ϵ.
Output: A coefficient matrix $W \in \mathbb{R}^{p \times p}$.

$\tilde{X}_j \leftarrow X_j / ||X_j||_2$, the normalized basis for $j = 1, ..., p$.
$\Lambda_j^{(0)} \leftarrow \varnothing, j = 1, \ldots, p$
$w_j^{(0)} \leftarrow \mathbf{0} \in \mathbb{R}^p, j = 1, \ldots, p$
$r_j^{(0)} \leftarrow Y_i - X w_i^{(0)}, j = 1, \ldots, p$
for $k = 1, 2, \ldots$ **do**
    $(i^{(k)}, j^{(k)}) \leftarrow \arg\max_{i,j} \left|\langle \tilde{X}_i, r_j^{(k-1)} \rangle\right| / \|r_j^{(k-1)}\|_2$ ▷ get most correlated pair
    if $\max_{i,j} \left|\langle \tilde{X}_i, r_j^{(k-1)} \rangle\right| / \|r_j^{(k-1)}\|_2 > \epsilon$ **then**
        $\Lambda_j^{(k)} \leftarrow \Lambda_j^{(k-1)} \cup \{(i^{(k)}\}$ ▷ update $\Lambda_j^{(k-1)}$, $w_j^{(k-1)}$, $r_j^{(k-1)}$ for j
        $X^{(k)} \leftarrow X$, with column $X_i^{(k)}$ all zeros if $i \notin \Lambda^{(k)}$.
        $w_j^{(k)} \leftarrow \left(X^{(k)} X^{(k)^T}\right)^{-1} X^{(k)^T} Y_j$ ▷ OLS solution restricted to $\Lambda_j^{(k)}$
        $r_j^{(k)} \leftarrow Y_j - X w^{(k)}$
        $\Lambda_{j'}^{(k)} \leftarrow \Lambda_{j'}^{(k-1)}$ for $j' \neq j$ ▷ copy over $\Lambda_{j'}^{(k-1)}, w_{j'}^{(k-1)}, r_{j'}^{(k-1)}$ for $j \neq j'$
        $w_{j'}^{(k)} \leftarrow w_{j'}^{(k-1)}$ for $j' \neq j$
        $r_{j'}^{(k)} \leftarrow r_{j'}^{(k-1)}$ for $j' \neq j$
    else
        return the matrix $W^{(k-1)} = \left(w_1^{(k-1)}, w_2^{(k-1)}, \ldots, w_p^{(k-1)}\right)$
    end if
end for

Enforcing acyclicity of W. In its current form, Algorithm 6.3 does not enforce acyclicity. We can enforce acyclicity by first checking if the arc (i, j) will introduce a cycle. To check for a cycle, we can do a topological sort on W or, its support matrix $\mathbf{\Lambda}^{(k)} = \left(\Lambda_1^{(k)}, \Lambda_2^{(k)}, \ldots, \Lambda_p^{(k)}\right)$, so that we need not estimate $W^{(k)}$ before knowing if arc (i, j) will introduce a cycle.

If adding (i, j) indeed introduces a cycle, we exclude the aforementioned arc (i, j) that introduced the cycle from ever being considered again. We can argue that arc (i, j) was the least important arc in the cycle anyhow. We use "anti-support" sets F_j that contain all entries i such arcs (i, j) will introduce a cycle. If a cycle is indeed introduced, we remove i from Λ_j and add i to the set of forbidden arcs F_j. We decrement k so that $W^{(k)}$ will still correspond to the matrix with k estimated coefficients. Now, instead of looking for any $(X_i, r_j^{(k-1)})$ pair that maximizes Equation 6.14, we are now interested in finding the pair such that also $i \notin F_j$. The pseudocode for this modified version of OMP, abbreviated as `DAG-OMP`, is described in Algorithm 6.4.

Algorithm 6.4: `DAG-OMP`

textbfInput: explanatory variables $X = [\mathbf{X}_{1:T-1,1}, \ldots, \mathbf{X}_{1:T-1,p}] \in \mathbb{R}^{T-1 \times p}$,
response matrix $Y = [\mathbf{X}_{2:T,1}, \ldots, \mathbf{X}_{2:T,p}] \in \mathbb{R}^{T-1 \times p}$, threshold ϵ.
Output: An acyclic coefficient matrix $W \in \mathbb{R}^{p \times p}$.

$\tilde{X}_j \leftarrow X_j / ||X_j||_2$, the normalized basis for $j = 1, ..., p$.
$\Lambda_i^{(0)} \leftarrow \varnothing, i = 1, \ldots, p$
$w_i^{(0)} \leftarrow \mathbf{0} \in \mathbb{R}^p, i = 1, \ldots, p$
$r_i^{(0)} \leftarrow Y_i - X w_i^{(0)}, i = 1, \ldots, p$
$F_i^{(0)} \leftarrow \varnothing, i = 1, \ldots, p$
for $k = 1, 2, \ldots$ **do**
  $(i, j) \leftarrow \underset{(i,j): i \notin F_j^{(k-1)}}{\arg\max} \left|\langle \tilde{X}_i, r_j^{(k-1)} \rangle\right| / \|r_j^{(k-1)}\|_2$ ▷ get most correlated pair
  if $\underset{(i,j): i \notin F_j^{(k-1)}}{\max} \left|\left\langle \tilde{X}_i, r_j^{(k-1)} \right\rangle\right| / \|r_j^{(k-1)}\|_2 > \epsilon$ **then**
    $\Lambda_j^{(k)} \leftarrow \Lambda_j^{(k-1)} \cup \{i\}$. ▷ try adding arc (i, j)
    if $G\left(\left(\Lambda_1^{(k)}, \Lambda_2^{(k)}, \ldots, \Lambda_p^{(k)}\right)\right)$ is not acyclic **then**
      $X^{(k)} \leftarrow X$, with column $X_i^{(k)}$ all zeros if $i \notin \Lambda^{(k)}$.
      $w_j^{(k)} \leftarrow \left(X^{(k)} X^{(k)^T}\right)^{-1} X^{(k)^T} Y_j$ ▷ OLS solution restricted to $\Lambda_j^{(k)}$
      $r_j^{(k)} \leftarrow Y_j - X w_j^{(k)}$
      $\Lambda_{j'}^{(k)} \leftarrow \Lambda_{j'}^{(k-1)}$ for $j' \neq j$ ▷ Copy over $\Lambda_{j'}^{(k-1)}, w_{j'}^{(k-1)}, r_{j'}^{(k-1)}$
      $w_{j'}^{(k)} \leftarrow w_{j'}^{(k-1)}$ for $j' \neq j$
      $r_{j'}^{(k)} \leftarrow r_{j'}^{(k-1)}$ for $j' \neq j$
      $F_{j'}^{(k)} \leftarrow F_{j'}^{(k-1)}$ for $j' \neq j$
    else ▷ Exclude arc (i, j) from being considered again
      $\Lambda_j^{(k)} \leftarrow \Lambda_j^{(k-1)} \setminus \{i\}$.
      $F_j^{(k)} \leftarrow F_j^{(k-1)} \cup \{i\}$.
      $k \leftarrow k - 1$
    end if
  else
    return $W^{(k-1)} = \left(w_1^{(k-1)}, w_2^{(k-1)}, \ldots, w_p^{(k-1)}\right)$
  end if
end for

Applying DAG-OMP on data generated by an acyclic W^*. Having gone quite some lengths to translate the orthogonal matching pursuit algorithm to our setting, let us consider the effectiveness of this iterative approach.

Example 6.3 DAG-OMP on three variables.

Let us apply DAG-OMP on the same three-dimensional setting as in Example 5.4 and Example 5.6, where each time series consists of one hundred time steps each, and the data generating matrix was

$$W^* = \begin{pmatrix} 0.50 & -0.65 & 0.00 \\ 0.00 & 0.50 & -0.25 \\ 0.00 & 0.00 & 0.40 \end{pmatrix}.$$

To ease notation, we will use the matrix $\mathbf{C}^{(k)} \in \mathbb{R}^{3\times 3}$ to denote the correlations at the start of iteration k, and $W^{(k)}$ to denote the coefficient matrix at the end of iteration k,

$$c_{ij}^{(k)} = \frac{\left|\left\langle \mathbf{X}_{1:T-1,i}, r_j^{(k-1)} \right\rangle\right|}{\|\mathbf{X}_{1:T-1,i}\|_2 \, \|r_j^{(k-1)}\|_2}, \qquad W^{(k)} = \begin{pmatrix} w_1^{(k)} & w_2^{(k)} & w_3^{(k)} \end{pmatrix}. \tag{6.15}$$

If arc (i, j) has been included or explicitly excluded, we will leave the corresponding entry c_{ij} blank. Computing which covariate X_i is most correlated with the residual of the response variable Y_j results the correlations $\mathbf{C}^{(1)}$ given on the left-hand side of Equation 6.16.

$$\mathbf{C}^{(1)} = \begin{pmatrix} 0.53 & 0.62 & 0.21 \\ 0.23 & 0.71 & 0.43 \\ 0.07 & 0.36 & 0.50 \end{pmatrix} \Rightarrow W^{(1)} = \begin{pmatrix} 0.00 & 0.00 & 0.00 \\ 0.00 & 0.71 & 0.00 \\ 0.00 & 0.00 & 0.00 \end{pmatrix} \tag{6.16}$$

Arc $(2, 2)$ corresponds to the largest residual correlation. Estimating the corresponding coefficient using constrained ordinary least squares yields $w_{22} = 0.72$. Now, the updated correlations are given on the left-hand side of Equation 6.17.

$$\mathbf{C}^{(2)} = \begin{pmatrix} 0.53 & 0.55 & 0.21 \\ 0.23 & & 0.43 \\ 0.07 & 0.16 & 0.50 \end{pmatrix} \Rightarrow W^{(2)} = \begin{pmatrix} 0.00 & -0.64 & 0.00 \\ 0.00 & 0.57 & 0.00 \\ 0.00 & 0.00 & 0.00 \end{pmatrix} \tag{6.17}$$

The residual correlations in the first and third columns have not changed after estimating w_{12}, as the corresponding coefficients and residuals remain unaltered. On the other hand, the correlations in column 2 have decreased and the coefficient w_{22} has been re-estimated.

In the next three iterations, we add the arcs $(1, 1), (3, 3)$, and $(2, 3)$, respectively.

$$\mathbf{C}^{(3)} = \begin{pmatrix} 0.53 & & 0.21 \\ 0.23 & & 0.43 \\ 0.07 & 0.12 & 0.50 \end{pmatrix} \Rightarrow W^{(3)} = \begin{pmatrix} 0.53 & -0.64 & 0.00 \\ 0.00 & 0.57 & 0.00 \\ 0.00 & 0.00 & 0.00 \end{pmatrix}, \tag{6.18}$$

$$\mathbf{C}^{(4)} = \begin{pmatrix} & & 0.21 \\ 0.07 & & 0.43 \\ 0.05 & 0.12 & 0.50 \end{pmatrix} \Rightarrow W^{(4)} = \begin{pmatrix} 0.53 & -0.64 & 0.00 \\ 0.00 & 0.57 & 0.00 \\ 0.00 & 0.00 & 0.50 \end{pmatrix}, \tag{6.19}$$

$$\mathbf{C}^{(5)} = \begin{pmatrix} & & 0.12 \\ 0.07 & & 0.30 \\ 0.05 & 0.12 & \end{pmatrix} \Rightarrow W^{(5)} = \begin{pmatrix} 0.53 & -0.64 & 0.00 \\ 0.00 & 0.57 & -0.20 \\ 0.00 & 0.00 & 0.40 \end{pmatrix}. \tag{6.20}$$

At this stage, we have recovered all true coefficients of W^*.

If the threshold value ϵ would have been between 0.12 and 0.30, we would have recovered the true support of W^* and no arc more. However, knowing such a threshold value beforehand is generally not possible. Therefore, we continue until we have estimated a full acyclic coefficient matrix consisting of $p(p+1)/2 = 6$ coefficients. At the next iteration, we have

$$\mathbf{C}^{(6)} = \begin{pmatrix} & & 0.03 \\ 0.07 & & \\ 0.05 & 0.12 & \end{pmatrix} \Rightarrow W^{(6)} = \begin{pmatrix} 0.53 & -0.64 & 0.03 \\ 0.00 & 0.57 & -0.20 \\ 0.00 & 0.00 & 0.40 \end{pmatrix}. \tag{6.21}$$

The correlation corresponding to arc $(3, 2)$ is maximal. However, arc $(2, 3)$ is already included in our structure, so adding arc $(2, 3)$ would introduce a cycle. Hence, we exclude $(3, 2)$ from consideration, and consider the second-largest entry, corresponding to arc $(1, 2)$. However this arcs would also introduce a cycle, and the third-best choice, arc $(3, 1)$, would introduce a cycle of length three. Therefore, our only option is to include arc $(1, 3)$, yielding the final matrix on the right-hand side of Equation 6.21.

The DAG-OMP algorithm is able to estimate a coefficient matrix W that fits $\mathbf{X}$ reasonably well. Its iterative approach also gives the insight which edges were deemed most important, and which edges were deemed less important. Furthermore, acyclicity was enforced at each iteration, although intervention was only necessary when we added the final arc.

Applying DAG-OMP on data generated by a cyclic W^*. Let us now verify how suitable DAG-OMP is when the data generating matrix W^* is cyclic. We have seen several continuous methods such as the gradient descent method in Section 5.1 and DAG-LASSO in Section 5.3 struggle with cyclic structures, as their global approaches do not consider arcs individually.

Example 6.4 DAG-OMP on three variables with cyclic W.

Let us apply DAG-OMP on the same three-dimensional setting as in Example 5.5 and Example 5.7, where each time series consisted of one hundred time steps and the data generating matrix

$$W^* = \begin{pmatrix} 0.35 & 0.40 & 0.00 \\ -0.50 & 0.30 & 0.00 \\ 0.00 & 0.00 & 0.50 \end{pmatrix}$$

corresponds to a cyclic structure.

To ease notation, we will again use the matrix $\mathbf{C}^{(k)} \in \mathbb{R}^{3\times 3}$ to denote the correlations computed at the start of iteration k, and $W^{(k)}$ to denote the coefficient matrix at iteration k. The first matrix of correlations $\mathbf{C}^{(0)}$ is given on the left-hand side of Equation 6.22.

$$\mathbf{C}^{(1)} = \begin{pmatrix} 0.31 & 0.35 & 0.19 \\ 0.57 & 0.32 & 0.07 \\ 0.05 & 0.09 & 0.32 \end{pmatrix} \Rightarrow W^{(1)} = \begin{pmatrix} 0.00 & 0.00 & 0.00 \\ -0.50 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.00 \end{pmatrix}. \tag{6.22}$$

Arc $(2, 1)$ corresponds to the largest residual correlation. Estimating the corresponding coefficient using constrained ordinary least squares yields $w_{21} = -0.50$. Now, the updated residual correlations are given on the left-hand side of Equation 6.23.

$$\mathbf{C}^{(2)} = \begin{pmatrix} 0.34 & 0.35 & 0.19 \\ & 0.32 & 0.07 \\ 0.09 & 0.09 & 0.32 \end{pmatrix} \Rightarrow W^{(2)} = \begin{pmatrix} 0.28 & 0.00 & 0.00 \\ -0.50 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.00 \end{pmatrix}. \tag{6.23}$$

After adding arc $(2, 1)$, the reversed arc $(1, 2)$ is deemed most important. However, this would create a cycle of length two, so we exclude arc $(1, 2)$ and select the arc that corresponds to the second-largest correlation, which is arc $(1, 1)$, resulting in the coefficient $w_{11} = 0.28$. In the coming iterations, we need not consider the correlation corresponding to arc $(2, 1)$.

In the next three iterations, we add the arcs $(3, 3), (2, 2)$, and $(1, 3)$, respectively.

$$\mathbf{C}^{(3)} = \begin{pmatrix} & & 0.19 \\ & 0.32 & 0.07 \\ 0.09 & 0.09 & 0.32 \end{pmatrix} \Rightarrow W^{(3)} = \begin{pmatrix} 0.28 & 0.00 & 0.00 \\ -0.50 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.32 \end{pmatrix}. \tag{6.24}$$

$$\mathbf{C}^{(4)} = \begin{pmatrix} & & 0.21 \\ & 0.32 & 0.01 \\ 0.09 & 0.09 & \end{pmatrix} \Rightarrow W^{(4)} = \begin{pmatrix} 0.28 & 0.00 & 0.00 \\ -0.50 & 0.32 & 0.00 \\ 0.00 & 0.00 & 0.32 \end{pmatrix}. \tag{6.25}$$

$$\mathbf{C}^{(5)} = \begin{pmatrix} & & 0.21 \\ & & 0.01 \\ 0.09 & 0.02 & \end{pmatrix} \Rightarrow W^{(5)} = \begin{pmatrix} 0.28 & 0.00 & -0.18 \\ -0.49 & 0.32 & 0.00 \\ 0.00 & 0.00 & 0.32 \end{pmatrix}. \tag{6.26}$$

To add the sixth arc to $W^{(5)}$, the corresponding correlations are

$$\mathbf{C}^{(6)} = \begin{pmatrix} & & 0.01 \\ 0.09 & 0.02 & \end{pmatrix} \Rightarrow W^{(6)} = \begin{pmatrix} 0.28 & 0.00 & -0.19 \\ -0.49 & 0.32 & 0.02 \\ 0.00 & 0.00 & 0.31 \end{pmatrix}. \tag{6.27}$$

The arcs $(3, 1)$ and $(3, 2)$ both introduce a cycle, so we will exclude those, and settle for the least important arc $(2, 3)$, as it is the only arc that does not introduce a cycle.

In Example 6.4, we have seen that this iterative approach is indeed suitable for estimating an acyclic coefficient matrix, even when the data generating matrix W^* was cyclic. Due to its iterative approach, where one arc is considered at the time, we can effectively enforce acyclicity of the inferred structure.

Speeding up `DAG-OMP`. Several approaches exist to improve the computational efficiency of `DAG-OMP`. First, if we decide to include the arc (i, j), where $i \neq j$, then we can already exclude the arc (j, i), as adding arc (j, i) would introduce a cycle of length two. This would approximately half the number of residual correlations we need to compute.

A more significant improvement can be made by reconsidering the constrained ordinary least squares computation in line 12 of Algorithm 6.4. Computing w_j requires three matrix multiplications, two $(p \times T-1) \times (T-1 \times p)$ matrix multiplication and one $(p \times p) \times (p \times p)$ multiplication, as well as inverting a $p \times p$ matrix. We can speed this up significantly by precomputing the relevant inner products and storing them in the matrices $\mathbf{\Psi}$ and $\mathbf{K}$, as is also proposed in [43]. We define

$$\mathbf{\Psi} = \mathbf{X}_{1:T-1,\cdot}^T \mathbf{X}_{1:T-1,\cdot} \in \mathbb{R}^{p \times p}, \qquad \mathbf{K} = \mathbf{X}_{1:T-1,\cdot}^T \mathbf{X}_{2:T,\cdot} \in \mathbb{R}^{p \times p}. \tag{6.28}$$

Here, $\mathbf{\Psi}$ represents the *Gram* matrix of the columns of $\mathbf{X}_{1:T-1,\cdot}$, so $\mathbf{\Psi}_{ij} = \langle \mathbf{X}_{1:T-1,i}, \mathbf{X}_{1:T-1,j} \rangle$, and $\langle \cdot, \cdot \rangle$ represents the inner product of two vectors. Similarly, $\mathbf{K}_{ij} = \langle \mathbf{X}_{1:T-1,i}, \mathbf{X}_{2:T,j} \rangle$.

We can avoid the two $(p \times T-1) \times (T-1 \times p)$ matrix multiplications using our matrices $\mathbf{\Psi}$ and $\mathbf{K}$. Let $\mathbf{\Psi'}$ be the matrix containing the entries of $\mathbf{\Psi}$, but the entries Ψ'_{ij} and Ψ'_{ji} are set to zero if i is contained in Λ_j. Then, computing the constrained ordinary least squares solution of the j column vector $w_{\cdot,j}$ corresponds to

$$w_{\cdot,j} = \mathbf{\Psi'}^{-1} \mathbf{K}_{\cdot,\mathbf{j}}, \tag{6.29}$$

which requires one $(p \times p) \times (p \times p)$ matrix multiplication and computing the inverse of a $p \times p$ matrix. These matrices $\mathbf{\Psi}$ and $\mathbf{K}$ need to be computed beforehand, but we can reuse these matrices for every iteration, thereby greatly reducing the computational costs.

Additional tricks exist to speed up the algorithm, mostly focused on speeding up the constrained least squares computations. Several approaches are discussed in [91] to compute the constrained ordinary least squares estimate more efficiently, such as using the matrix inversion lemma [33], a Cholesky decomposition approach [18], or a QR-factorization approach [76].

6.2 Using a backward Iterative Procedure

In Section 6.1, we have constructed an algorithm that can learn a sparse matrix by using an iterative forward procedure called orthogonal matching pursuit. We have first extended the orthogonal matching pursuit algorithm to multiple response variables, after which we added an additional condition that enforces the estimated coefficient matrix W to be acyclic.

Instead of forward iterative approaches, we can also construct a coefficient matrix in the opposite direction. We start with a dense coefficient matrix, and based on some deletion criterion, we prune the coefficient matrix until we have an acyclic structure.

Introduction to `DAG-OLS`. Let us describe a simple yet effective approach to estimate a suitable acyclic matrix using a backward iterative procedure. Recall at the beginning of Chapter 4 that the maximum likelihood estimator of the data generating matrix W^* can be found using an ordinary least squares estimation. Let $\mathbf{X}_{2:T,\cdot}$ represent the first $T-1$ time steps of the data matrix $\mathbf{X}$, and let $\mathbf{X}_{2:T,\cdot}$ represent the last $T-1$ time steps of the data matrix $\mathbf{X}$. Then, the maximum likelihood estimator of $\mathbf{X}$ corresponds to the ordinary least squares (OLS) estimator,

$$W_{\text{OLS}} = \left(\mathbf{X}^T_{1:T-1,\cdot}\mathbf{X}_{1:T-1,\cdot}\right)^{-1}\mathbf{X}^T_{1:T-1,\cdot}\mathbf{X}_{2:T,\cdot}, \tag{6.30}$$

where the T in the subscript represents the number of time steps T, and the T in the superscript represents the transpose.

As we had also encountered in Chapter 4, the maximum likelihood estimator W_{OLS} does not necessarily correspond to an acyclic structure. Therefore, we must modify W_{OLS}.

A naive yet effective solution is to iteratively set the smallest nonzero coefficient to zero until the coefficient matrix is acyclic. Such an approach was also just one step of the algorithm described in [66]. We first compute the ordinary least squares estimate, after which we iteratively set the nonzero coefficient w_{ij} that is smallest in absolute value to zero. As this is a procedure to transform the graph induced by the ordinary least squares estimate into a directed acyclic graph, we have called this algorithm `DAG-OLS`. The pseudocode of `DAG-OLS` is given in Algorithm 6.6.

Algorithm 6.5: `DAG-OLS`.

Input: A data matrix $\mathbf{X} \in \mathbb{R}^{T\times p}$.
Output: A coefficient matrix $W \in \mathbb{R}^{p\times p}$ such that W is a acyclic.

$W \leftarrow \left(\mathbf{X}^T_{1:T-1,\cdot}\mathbf{X}_{1:T-1,\cdot}\right)^{-1}\mathbf{X}^T_{1:T-1,\cdot}\mathbf{X}_{2:T,\cdot}$
while W contains a cycle **do**
    $i, j \leftarrow \arg\min_{i,j}\{|w_{ij}| \mid w_{ij} \neq 0\}$
    $w_{ij} \leftarrow 0$
end while
return W

Although this algorithm may seem quite naive, the maximum likelihood estimator would have been a sensible choice if we did not need to enforce acyclicity. In addition, the OLS estimate W_{OLS} is a *consistent* estimator for the data generating coefficient matrix W^*. This means that when $\mathbf{X}$ has been generated according to a VAR(1) model, then the maximum likelihood estimate converges to the true matrix as the number of samples T tends to infinity, $W_{\text{OLS}} \overset{T\to\infty}{\longrightarrow} W^*$. Therefore, when T is large enough, if a coefficient in W^* is equal to zero, then the corresponding coefficient in W_{OLS} will be close to zero as well.

Applying `DAG-OLS` on data generated by an acyclic W^*. Encouraged by the statement on the consistency of the ordinary least squares estimate, let us see whether this consistency is also useful in a finite sample setting.

If the data generating matrix W^* is acyclic, we expect the ordinary least squares solution W_{OLS} to estimate all coefficients rather well, up to some slight deviations due to noise. Therefore, the only coefficients which have a significant contribution will form an acyclic structure which `DAG-OLS` will hopefully recover.

Example 6.5 `DAG-OLS` on three dimensional data generated by acyclic W^*.

Let us consider the same data matrix $\mathbf{X} \in \mathbb{R}^{100\times 3}$ as in Example 6.3, consisting of three time series of one hundred time steps each. The data has been generated according to a VAR(1) model with data generating matrix

$$W^* = \begin{pmatrix} 0.50 & -0.65 & 0.00 \\ 0.00 & 0.50 & -0.25 \\ 0.00 & 0.00 & 0.40 \end{pmatrix}.$$

The ordinary least squares estimate, or equivalently, the maximum likelihood estimate W_{OLS}, has been given on the left-hand side of Equation 6.31.

$$W_{\text{OLS}} = \begin{pmatrix} 0.50 & -0.62 & 0.01 \\ -0.02 & 0.54 & -0.16 \\ -0.14 & -0.11 & 0.32 \end{pmatrix} \qquad W_{\texttt{DAG-OLS}} = \begin{pmatrix} 0.50 & -0.62 & 0.00 \\ 0.00 & 0.54 & -0.16 \\ 0.00 & 0.00 & 0.32 \end{pmatrix}. \tag{6.31}$$

Applying `DAG-OLS`, we will iteratively set the non-zero coefficient that is smallest in absolute value to zero until the structure is acyclic. For this, we set the coefficients w_{13}, w_{21}, w_{32}, and lastly w_{31} to zero. The final matrix $W_{\texttt{DAG-OLS}}$ has been given on the right-hand side of Equation 6.31. We see that the solution contains all the true coefficients of W^*.

Applying `DAG-OLS` on data generated by a cyclic W^*. Let us now consider a cyclic data generating matrix W^*. Just as in Example 6.5, we expect the ordinary least squares estimate to be reasonably close to W^*. However, as W^* is acyclic, we cannot estimate all coefficients of W^*. Therefore, a large number of true arcs may be removed until we have an acyclic structure.

We need to "break" all cycles in W_{OLS}, and `DAG-OLS` does that by setting the smallest coefficient in that cycle to zero. However, all coefficients that are smaller in absolute value will first be removed, which could make the final coefficient matrix much sparser than we might prefer.

Example 6.6 `DAG-OLS` on three dimensional data generated by a cyclic W^*.

Let us consider the same data matrix $\mathbf{X} \in \mathbb{R}^{100\times 3}$ as in Example 6.4, consisting of three time series of one hundred time steps each. The data has been generated according to a VAR(1) model with data generating matrix

$$W^* = \begin{pmatrix} 0.35 & 0.40 & 0.00 \\ -0.50 & 0.30 & 0.00 \\ 0.00 & 0.00 & 0.50 \end{pmatrix}.$$

Computing the ordinary least squares estimate, or equivalently, the maximum likelihood estimator yields the matrix W_{OLS} given on the left-hand side of Equation 6.32.

$$W_{\text{OLS}} = \begin{pmatrix} 0.28 & 0.42 & -0.18 \\ -0.50 & 0.33 & 0.01 \\ -0.02 & -0.03 & 0.32 \end{pmatrix} \qquad W_{\texttt{DAG-OLS}} = \begin{pmatrix} 0.00 & 0.00 & 0.00 \\ -0.50 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.00 \end{pmatrix}. \tag{6.32}$$

`DAG-OLS` will iteratively set the non-zero coefficient that is smallest in absolute value to zero until the structure is acyclic. This means that we set the coefficients w_{23}, w_{31}, w_{32}, w_{13}, w_{11}, w_{33}, w_{22}, and lastly w_{21}, to zero. The final matrix $W_{\texttt{DAG-OLS}}$ on the right-hand side of Equation 6.32 contains only one non-zero entry, the entry w_{12} that was largest in absolute value in W_{OLS}.

On using the coefficient size as a deletion criterion. The deletion criterion of DAG-OLS is to iteratively set the smallest nonzero coefficient to zero. The reasoning behind this criterion is that we want to remove the least important coefficients first before we start removing more important coefficients. As it is difficult to know how important a coefficient is, we naively use the magnitude of the coefficient as an indicator of its importance. Although a simple an efficient strategy, this is not always a clever choice.

A method that set coefficients to zero based on their predictive performance is the DAG-LASSO algorithm described in Section 5.3, where we used a penalty parameter λ to shrink the coefficients in W_λ until acyclicity has been reached. From a predictive viewpoint, DAG-LASSO is more capable of recovering the important coefficients of W^* than DAG-OLS. DAG-LASSO shrinks all coefficients proportional to their predictive performance.

Example 6.7 DAG-OLS versus DAG-LASSO on two dimensional data.

Consider the data matrix $\mathbf{X} \in \mathbb{R}^{100\times 2}$, consisting of two time series with 100 time steps each. The data has been generated according to a VAR(1) model with coefficient matrix

$$W^* = \begin{pmatrix} 0.95 & -0.25 \\ 1.00 & 0.00 \end{pmatrix}.$$

The time series are plotted in Figure 6.2.

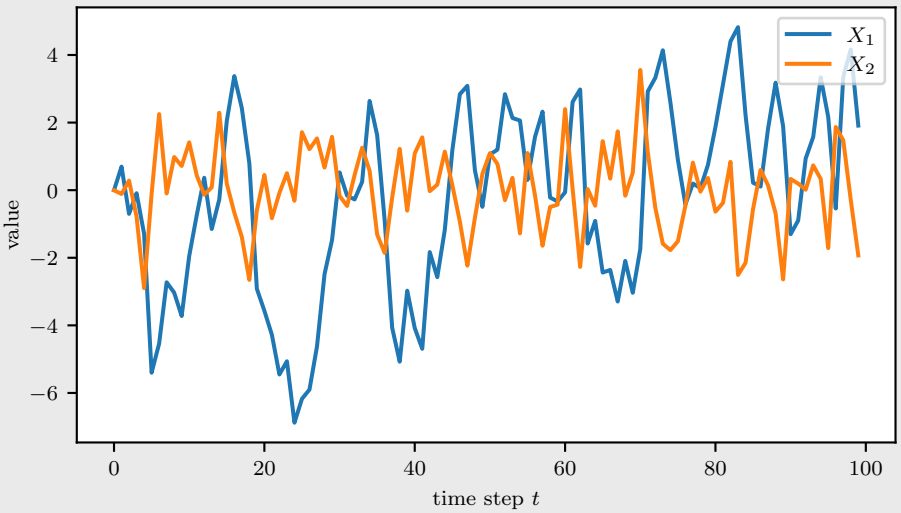


Figure 6.2: Visualization of the two variables X_1, X_2 of Example 6.7

W^* does not correspond to an acyclic structure, as both off-diagonals contain non-zero entries. To obtain an acyclic structure, we must either set w_{12} or w_{21} to zero. Preferably, we set the least important coefficient to zero.

From simply inspecting coefficient value, we would say that w_{21} is the most important coefficient. However, w_{21} corresponds to a contribution of $1.00X_{t-1,2}$ to $X_{t,1}$, whereas the value of $X_{t,1}$ is mostly determined by its autoregressive component $0.95X_{t-1,1}$. Furthermore, as $w_{22} = 0.00$, we have that the $X_{t-1,2}$ is simply the noise coefficient $\varepsilon_{t-1,2}$. Therefore, although w_{21} is the largest coefficient, its contribution is quite modest. It is likely that the contribution attributed to w_{12} is greater, although its size is four times smaller.

$$W_{\text{OLS}} = \begin{pmatrix} 1.03 & -0.11 \\ 1.19 & 0.14 \end{pmatrix}, \; W_{\text{DAG-OLS}} = \begin{pmatrix} 0.00 & 0.00 \\ 1.19 & 0.00 \end{pmatrix}, \; W_{\text{DAG-LASSO}} = \begin{pmatrix} 0.65 & -0.10 \\ 0.00 & 0.00 \end{pmatrix} \tag{6.33}$$

DAG-OLS estimates the matrix in the middle of Equation 6.33. All coefficients except the least important coefficient w_{21} have been removed. DAG-LASSO estimates the matrix on the right-

hand side of Equation 6.33. The DAG-LASSO approach obtains a more suitable coefficient matrix W, where the "correct" choice between w_{12} and w_{21} is made.

The improvement of DAG-LASSO can also be seen in the solution path of the coefficients of W_λ in Figure 6.3. For $\lambda = 0$, we obtain the ordinary least squares estimate, with w_{21} indeed the largest coefficient in magnitude. However, if we increase the penalty parameter, then the coefficient w_{21} decreases much quicker in size than other coefficients. Even the coefficient w_{12}, which was more than four times smaller in magnitude than w_{21}, "survives" longer than w_{21}, until we have an acyclic graph for $\lambda^* \approx 1.19$.

Solution path of the coefficients W as a function of λ.

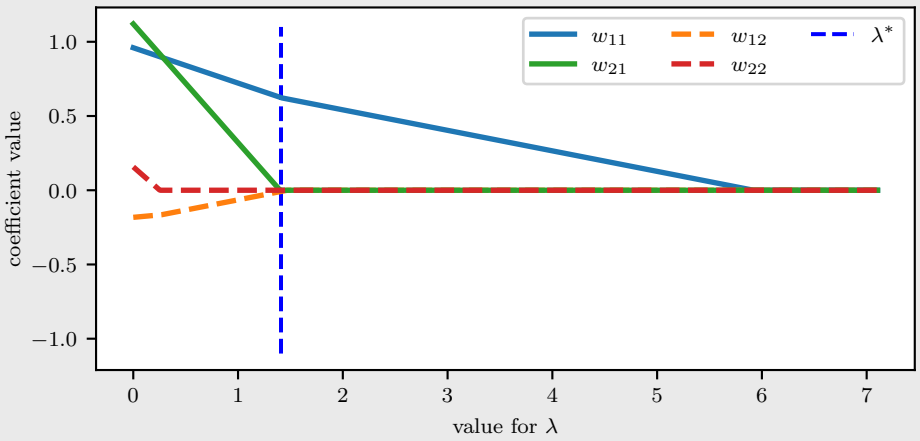


Figure 6.3: Visualization of the solution path of W_λ for Example 6.7.

This example shows that the coefficient size may not be an appropriate indicator of its predictive importance.

6.3 Several Other Iterative Approaches.

So far, we have discussed two iterative approaches. Firstly we have introduced DAG-OMP in Section 6.1, where we iteratively add the coefficient w_{ij} such that the covariate X_i has the largest correlation with the residual of the response variable Y_j, while ensuring the structure remains acyclic. Secondly, we have described DAG-OLS in Section 6.2, where we start with the ordinary least squares estimate and iteratively set the smallest nonzero coefficient to zero until acyclicity has been reached.

In this section, we will first be discussing whether we can also use these two approaches in the opposite direction. Furthermore, we will be introducing a useful trick that improves the performance of backward iterative procedures by only removing arcs that are contained in a cycle, a so-called *backward-violations first* approach.

Forward DAG-OLS. We can also employ DAG-OLS as a forwards procedure. Originally, DAG-OLS first estimates a full coefficient matrix W_{OLS}, after which arcs are iteratively removed until the structure is acyclic.

Now, rather than doing a backward iterative procedure, we can use the coefficient size as a selection criterion for the forward iterative procedure. The algorithm for this is similar to DAG-OMP, but instead of maximizing the residual correlation, we would maximize the coefficient size after refitting. That is, we estimate all coefficients in our matrix W and add the arc corresponding to the coefficient with the largest absolute value.

This would require one model fit each iteration. However, we have seen in Example 6.7 that the magnitude of a coefficient size may not be the best metric to assess the importance of the coefficient. Therefore, it seems that this forward variant of DAG-OLS would perform strictly worse than DAG-OMP, whereas both algorithms would require an equal amount of model fits.

Backward DAG-OMP. Similarly, we could investigate DAG-OMP in the opposite direction, thereby doing a backward iterative approach. We would start with a fully estimated matrix W. Then, we would remove the coefficient that yields the largest residual correlation after its removal. However, to compute these correlations, we first need to refit the model without using this coefficient.

Now, we require a new model fit for each arc we are considering. At iteration k, a total of $p^2 - k + 1$ arcs could be removed. Furthermore, it is possible that we have an acyclic matrix as late as iteration $p^2 - 1$, for example when the two most important arcs form a cycle of length two. Therefore, in the worst case, this means that we need to refit the model $\sum_{k=1}^{p^2-1} p^2 - (k-1) = \mathcal{O}(p^4)$ times. Even for just 10 variables, this yields a total of $5,049$ model fits. Therefore, we see that the backward variant of DAG-OMP is less efficient than the forward variant.

Nevertheless, similar approaches to a backward orthogonal matching pursuit exist, such as the Backward Optimized Orthogonal Matching Pursuit (BOOMP) approach [2]. However, this method is designed rather to sparsify the solution obtained by a forward orthogonal matching pursuit approach. The authors in [2] propose update rules such that the less model fits are required and the model fits are computationally less demanding.

Furthermore, although the backward procedure of DAG-OMP is a bit cumbersome, we still expect this backward procedure to outperform DAG-OLS, as the removal criterion is more appropriate. Therefore, if the number of variables is manageable, say only a couple of dozens of nodes, then we expect this method to be tractable as well.

6.3.1 A Backward-Violators First Approach.

As we have seen in Example 6.6, the backward iterative procedure DAG-OLS is not suitable when the data generating matrix W^* is cyclic. This is a shortcoming of the continuous method DAG-LASSO as well, as we saw in the previous chapter in Example 5.7. However, for iterative approaches, there exists a clever trick to make these backward iterative approaches robust against such model mismatches, that is, when we need to estimate an acyclic coefficient matrix W, yet W^* is cyclic.

Just as before, we start with the dense ordinary least squares estimate W_{OLS}. Next, we only consider removing arcs that violate the acyclicity assumption. If acyclicity is violated due to some cycle of length two, then it is unnecessary to remove an arbitrary arc that does not violate the acyclicity assumption. In Example 6.7, we should have only considered removing coefficients w_{12} and w_{21}. Using this violators-first approach, we would not have considered removed w_{11}, as this arc did not introduce any cycles. Nevertheless, we would have still removed w_{21} although w_{21} is the more important coefficient, but that problem is inherent to DAG-OLS rather than the backward iterative approach.

Finding the arcs that violate the acyclicity constraint. To employ this trick of only removing edges that violate the acyclicity assumption, we require an efficient procedure to retrieve all arcs that violate the acyclicity assumption.

An initial idea is to first count all cycles in the directed graph, and then remove all arcs that are contained in at least one cycle. Efficient algorithms and their implementations exist, one example being Johnson's algorithm [36]. However, the number of cycles grows exponentially with the number of nodes. To see this, suppose we have a fully connected directed graph. To get a cycle of length k, we can start at any arbitrary node, and traverse to $k-1$ other unique nodes arbitrarily, before returning to the first node. There are $p!/(p-k)!$ ways to gets such an ordered sequence of length k when there are p nodes. However, each cycle will be counted k times, once with the first node as "starter", once with the second node as "starter", etc. Therefore, the number of cycles of length k in a fully connected directed graph is equal to $p!/((p-k)!k)$, yielding a total number of cycles of

$$\sum_{k=2}^{p} \frac{p!}{(p-k)!k}. \tag{6.34}$$

Even for a complete directed graph on 10 nodes, the number of cycles is approximately $1.1 \cdot 10^6$. Therefore, iterating over the number of cycles is inefficient, as the running time will be exponential with respect to the number of nodes when the graph is dense.

A more tractable approach is to iterate over all arcs and for each arc, determine whether it belongs to a cycle. The method most readily available to us is a trace exponential function $h(W)$. In the beginning of this chapter, we have proposed the trace exponential function $h(W)$ to verify whether a coefficient matrix W corresponds to an acyclic structure,

$$h(W) = \operatorname{Tr}\left(e^{\tilde{W}}\right) - p = \sum_{k=1}^{\infty} \frac{1}{k!} \tilde{W}^k. \tag{6.35}$$

Here, $\tilde{W}$ corresponds to the matrix W with the diagonal entries set to zero. To verify whether arc (i, j) was part of any cycle, we can compare the values of $h(W)$ and $h\big(W^{-ij}\big)$, where W^{-ij} represents the matrix where we have set w_{ij} to zero. Then,

$$\text{arc } (i,j) \text{ is contained in any cycle } \iff h(W) - h\big(W^{-(i,j)}\big) \neq 0. \tag{6.36}$$

Evaluating this trace exponential function $h(W)$ can be done in $\mathcal{O}(p^3)$, and efficient implementations such as [1] are readily available in standard libraries. As there are $\mathcal{O}(p^2)$ possible arcs, finding all arcs that violate acyclicity using this approach requires $\mathcal{O}(p^5)$ time.

More efficiently, we can also detect whether an arc is contained in any cycle by using an arc traversal algorithm such as depth-first search. We start by first traversing the arc (i, j) and then continue a depth-first search. If we manage to return to the node i from which we started, we know that arc (i, j) must be contained in at least one cycle. Performing this depth-first search starting from an arc (i, j) requires traversing all arcs once. As there are p^2 possible arcs, finding all arcs that violate acyclicity requires approximately $\mathcal{O}(p^4)$ running time.

The `DAG-OLS` algorithm revisited. Let us now use this trick to overcome the issues with `DAG-OLS`. Instead of looking for the non-zero coefficient w_{ij} that is smallest in magnitude, we are now looking for the coefficient w_{ij} that is smallest in magnitude *and* violates the acyclicity constraint, in other words, is contained in a cycle. We now call this algorithm `DAG-OLS-V`, where the `V` stands for the word "violators".

Algorithm 6.6: `DAG-OLS-V`.

Input: A data matrix $\mathbf{X} \in \mathbb{R}^{T \times p}$.
Output: A coefficient matrix W such that the corresponding structure is acyclic.

$W \leftarrow \left(\mathbf{X}_{1:T-1}^T \mathbf{X}_{1:T-1}\right)^{-1} \mathbf{X}_{1:T-1}^T \mathbf{X}_{2:T}$
while W contains a cycle **do**
    $A_C \leftarrow$ set of all arcs in the graph induced by W that are contained in a cycle
    $i, j \leftarrow \arg\min_{(i,j) \in A_C} \{|w_{ij}| \mid w_{ij} \neq 0\}$
    $w_{ij} \leftarrow 0$
end while
return W

Example 6.8 Applying `DAG-OLS-V` on data generated by an W^*

We again consider the data matrix $\mathbf{X} \in \mathbb{R}^{100\times 3}$ corresponding to the same cyclic coefficient matrix as in Example 6.6, where

$$W^* = \begin{pmatrix} 0.35 & 0.40 & 0.00 \\ -0.50 & 0.30 & 0.00 \\ 0.00 & 0.00 & 0.50 \end{pmatrix}, \qquad W_{\text{OLS}} = \begin{pmatrix} 0.28 & 0.42 & -0.18 \\ -0.50 & 0.33 & 0.01 \\ -0.02 & -0.03 & 0.32 \end{pmatrix}.$$

Recall that `DAG-OLS` obtained a matrix where the only non-zero coefficient was $w_{21} = -0.50$.

Now, let us consider `DAG-OLS-V`. In the beginning, all coefficients except for the diagonal entries are contained in a cycle. As w_{23} is smallest in absolute value, we will set this coefficient to zero. In the next iteration, the the remaining five off-diagonal entries are still contained in a cycle, so we now set w_{31} to zero. The other four coefficients continue to be contained in a cycle, so now coefficient w_{32} will be set to zero.

Now, the arc $(1, 3)$ is not contained in a cycle anymore, so we will not no longer consider removing it. The remaining two arcs to consider are arc $(1, 2)$ and arc $(2, 1)$, so we will set w_{12} to zero, after which the structure is acyclic. This yield the matrix

$$W_{\texttt{DAG-OLS-V}} = \begin{pmatrix} 0.28 & 0.00 & -0.18 \\ -0.50 & 0.33 & 0.00 \\ 0.00 & 0.00 & 0.32 \end{pmatrix}.$$

`DAG-OLS-V` recovers four out of the five coefficients of W^*, which is the maximum amount of correct arcs to include in an acyclic structure, as there is a cycle of length two in W^*.

Using backward-violators first to detect a cyclic W^*. We can also use this modified backward direction to detect whether we have a model mismatch, that is, whether we are trying to recover an acyclic representation of a cyclic matrix W^*. Using a backward-violators first approach, the last arc we remove can be considered as the most important violating arc. To detect a cyclic coefficient matrix W^*, let us first describe how to acquire an ordering of importance of the coefficients using a forwards search, a backward search, and a backward-violators first approach.

For the forward (`F`) approach, we can acquire an ordering of the coefficients by considering the matrices $W_{\texttt{F}}^{(k)}$, where k refers to the iteration number and the number of non-zero coefficients. For the backward (`B`) approach, we can acquire a reverse ordering of the coefficients by considering the coefficient matrix $W_{\texttt{B}}^{(p^2-k)}$, where k refers to the iteration number, which is equal to the number of coefficients that have been set to zero. Therefore, the sequence of $(W_{\texttt{B}}^{(k)})_{k=0}^{p^2}$ corresponds to the reverse order in which the coefficients have been removed. Note, however, that not all matrices in this sequence will correspond to acyclic coefficient matrices.

For the backward-violators first (`B-V`) approach, we apply the same strategy for the backward approach, but we first only remove arcs that are contained in a cycle. Then, once we have an acyclic coefficient matrix, we iteratively remove the least important coefficient just as in the backward approach. The reverse order in which the coefficients have been removed then corresponds to an ordering of importance $(W_{\texttt{B-V}}^{(k)})_{k=0}^{p^2}$.

Example 6.9 Using directions to detect cyclicity of W^* using `DAG-OMP`.

Let us consider a data matrix $\mathbf{X}^{100\times 5}$ consisting of five time series, generated according to a VAR(1) model with

$$W^* = \begin{pmatrix} 0.50 & 0.00 & 0.00 & 0.00 & 0.00 \\ 0.00 & 0.50 & 0.00 & 0.00 & 0.00 \\ -0.47 & 0.47 & 0.50 & 0.83 & 0.00 \\ 0.00 & 0.00 & -0.83 & 0.50 & 0.00 \\ 0.53 & -0.55 & 0.00 & 0.00 & 0.50 \end{pmatrix}, W_{\text{OLS}} = \begin{pmatrix} 0.60 & -0.05 & 0.05 & 0.03 & 0.09 \\ 0.00 & 0.51 & 0.01 & 0.02 & 0.16 \\ -0.41 & 0.51 & 0.51 & 0.79 & 0.00 \\ 0.00 & -0.11 & -0.75 & 0.46 & -0.02 \\ 0.48 & -0.50 & -0.20 & -0.07 & 0.59 \end{pmatrix}.$$

We will use Orthogonal Matching Pursuit using the three aforementioned directions: Forward (`-F`), Backward (`-B`), and Backward-Violations first (`-B-V`). These result in the coefficient matrices in Equation 6.37.

$$W_{\texttt{F}} = \begin{pmatrix} 0.58 & -0.05 & 0.65 & 0.01 & 0.00 \\ 0.00 & 0.51 & 0.00 & 0.00 & 0.00 \\ 0.00 & 0.51 & 0.46 & 0.79 & 0.00 \\ 0.00 & -0.11 & 0.00 & 0.47 & 0.00 \\ 0.59 & -0.50 & 0.52 & 0.08 & 0.57 \end{pmatrix}, W_{\texttt{B}} = \begin{pmatrix} 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.00 & 0.81 & 0.00 \\ 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \end{pmatrix}.$$

$$W_{\texttt{B-V}} = \begin{pmatrix} 0.60 & 0.00 & 0.00 & 0.00 & 0.00 \\ 0.00 & 0.46 & 0.00 & 0.00 & 0.00 \\ -0.41 & 0.51 & 0.48 & 0.80 & 0.00 \\ 0.00 & 0.00 & 0.00 & 0.46 & 0.00 \\ 0.48 & -0.55 & -0.23 & 0.00 & 0.57 \end{pmatrix}. \tag{6.37}$$

We use the aforementioned approach to obtain an ordering of importance for each of the three procedures, which we denote by $(W_{\texttt{F}}^{(k)})_{k=0}^{15}$, $(W_{\texttt{V}}^{(k)})_{k=0}^{25}$, and $(W_{\texttt{B-V}}^{(k)})_{k=0}^{25}$. We have computed the mean squared errors when using the coefficient matrix $W^{(k)}$ on $\mathbf{X}$ to see how the predictive performance changes. The mean squared errors have been plotted in Figure 6.4. Furthermore, we have also visualized until which value of k the coefficient matrices was acyclic for the backward procedures.

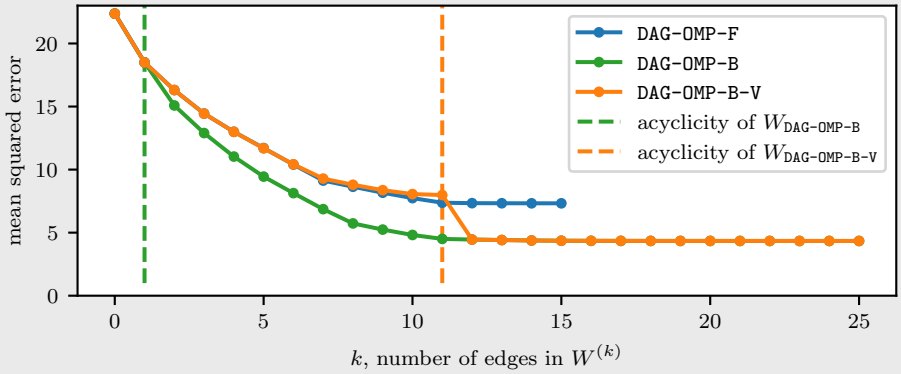


Figure 6.4: Visualization of the solution path for the coefficients of W of Example 5.7

For $k = 0$, the matrices achieve a low predictive performance for all three methods, as the matrix contains no arcs. As the number of coefficients increases, the predictive performance increases as well. The backward procedure seems to perform the best, but note that acyclicity is only attained for $k \leq 1$, so the backward procedure does not provide an acyclic coefficient matrices for $k \geq 2$.

The forward and backward-violations first method achieve a comparable predictive performance. However, at iteration $k = 12$, the mean squared error of the backward-violations first approach significantly decreases. It first removes all violators that are less important, but when 12 arcs remain in $W_{\texttt{DAG-OMP-B-V}}$, the choice between w_{34} and w_{43} had to be made, which meant that a large part of the predictive performance was lost.

This "jump" or "drop", depending from which side we are looking, between $k = 11$ and $k = 12$ of `DAG-OMP-B-V` combined with the large discrepancy in predictive performance between `DAG-OMP-F` and `DAG-OMP-B` indicates that the original structure was not acyclic.

We have seen in Example 6.9 that these iterative approaches give more insights into the importance of individual edges than the permutation based methods from Chapter 4 or the continuous approaches from Chapter 5.

6.4 Selecting a suitable number of arcs.

As mentioned in Chapter 1, a crucial aspect is that the inferred structure W must also be simple to interpret. Therefore, we prefer to have as few arcs as possible in the structure, yet the structure still adequately captures the relations between our variables. However, the discussed methods tend to estimate more arcs than required.

Example 6.10 Selecting a suitable number of arcs in a ten-dimensional setting.

Consider the ten-dimensional dataset $\mathbf{X} \in \mathbb{R}^{100\times 10}$ visualized in Figure 6.5 that we will be using throughout the examples in this section. The data has been generated using an acyclic coefficient matrix W^*, consisting of 25 arcs, of which 15 arcs are off-diagonal.

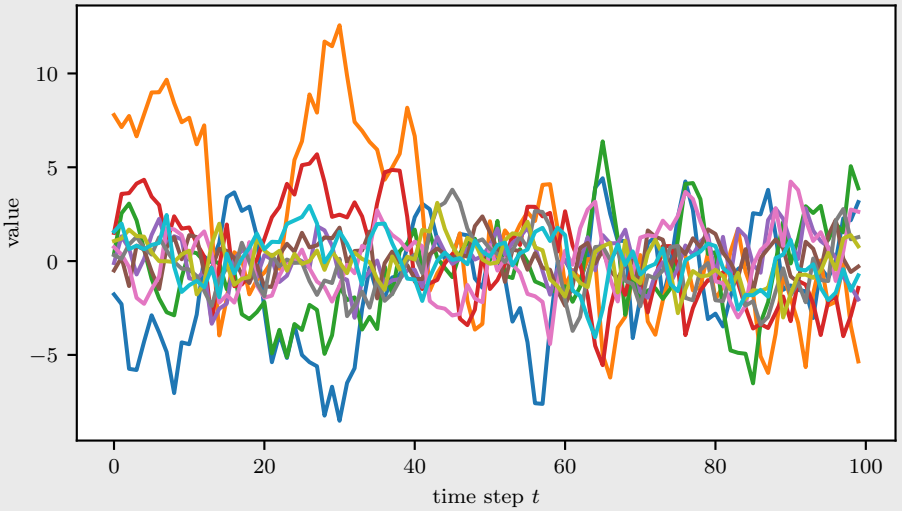


Figure 6.5: Visualization of the data matrix $\mathbf{X} \in \mathbb{R}^{100\times 10}$ from Example 6.10.

We can use `DAG-OMP` to infer an acyclic coefficient matrix W that hopefully resembles the data generating matrix W^*. If we set the threshold to zero, we will estimate a dense directed acyclic graph, consisting of 55 arcs, whereas the true W^* only contained 25 arcs.

As we were interested in recovering this sparse acyclic structure, we require a principled approach to determine which arcs are important enough to be included. As a visual approach, we can plot how the predictive performance of the coefficient increases as we add more arcs. Let $W^{(k)}$ represent the coefficient matrix at iteration k of `DAG-OMP`, containing k non-zero coefficients. In Figure 6.6, we have plotted the mean squared error as a function of $W^{(k)}$.

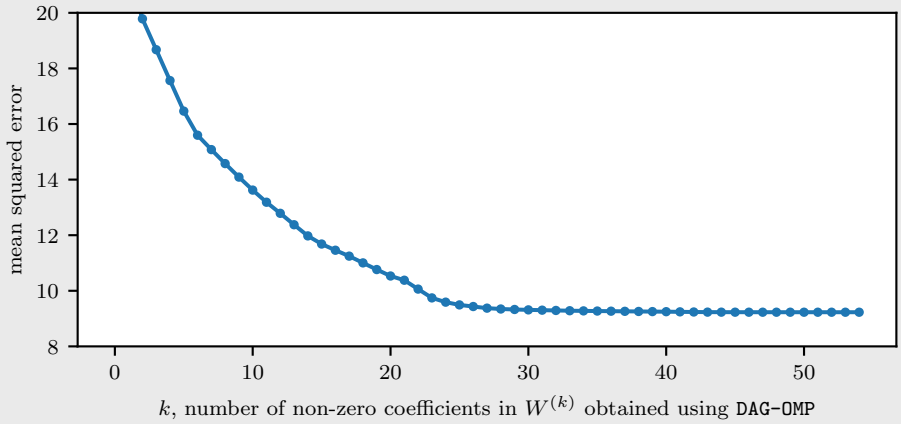


Figure 6.6: Mean squared error of $W^{(k)}$ on $\mathbf{X}$ as a function of k.

The first arcs were indeed quite important, as the mean squared error significantly decreases. However, the predictive gain seems to decrease, and after $k = 25$, the mean squared error seems to plateau. Therefore, we see that the predictive performance of $W^{(25)}$ is almost as good as the predictive performance of $W^{(55)}$, indicating that $W^{(25)}$ is preferred over $W^{(55)}$. The graphical model of $W^{(25)}$ is less complex, in the sense that is contains much fewer arcs, while still achieving almost the same predictive performance.

Next, let us consider some approaches that can help us in selecting a suitable subset of arcs from an acyclic coefficient matrix W, such that the graphical model becomes less complex while still retaining a suitable predictive performance.

6.4.1 Bootstrapping

Predicting the coefficients of W would have been much easier if we had more data. However, obtaining more data can be quite difficult in real life or outright impossible. Luckily, there exists an interesting approach to obtain "new" data by using a bootstrapping approach.

In such a bootstrapping approach, we use our available data matrix $\mathbf{X}$ and our estimated statistical model to resample "new" data. This idea is first introduced in [21]. We will be investigating two methods that utilize this bootstrapping procedure. The first method relies on using bootstrapping to construct *marginal confidence intervals* of the coefficients in W. The second method relies on using bootstrapping to verify the *recoverability* of coefficients in W.

Bootstrapping for VAR(1) models. We will consider the following bootstrapping procedure for VAR(1) models. A great book covering bootstrapping methods for VAR(1) models is "Resampling Methods for Dependent Data" by Lahiri [40].

The bootstrapping method is simple, yet effective. Suppose we have a dataset $\mathbf{X} \in \mathbb{R}^{T \times p}$ that has been generated by a VAR(1) model,

$$X_{t,\cdot} = X_{t-1,\cdot}W^* + \varepsilon_t, \tag{6.38}$$

where $\varepsilon_t \in \mathbb{R}^p, t = 2, \ldots, T$ are all independent and identically multivariate Gaussian random variables with zero mean and covariance matrix Σ. Furthermore, we assume that we are in a stationary setting, so we will generate $X_{1,\cdot}$ according to its stationary distribution, which is $\mathcal{N}(\mathbf{0}, B)$, where

$$\text{vec}(B) = \left(I_{p^2} - (W^* \otimes W^*)^T\right)^{-1} \text{vec}\left(\Sigma\right). \tag{6.39}$$

Now, the bootstrapping method described by Lahiri is as follows. First, we use any method discussed so far to retrieve an acyclic estimator W for the true coefficient matrix W^*. Then, we compute the estimated residuals

$$\hat{\varepsilon}_t = X_{t,\cdot} - X_{t-1,\cdot}W. \tag{6.40}$$

Using the assumption that our data has been generated according to Equation 6.38, we know that

$$\begin{aligned} \hat{\varepsilon}_t &= X_{t,\cdot} - X_{t-1,\cdot}W. && (6.41)\\ &= \varepsilon_t - X_{t-1,\cdot}\left(W^* - W\right). && (6.42) \end{aligned}$$

Assuming W^* was acyclic as well, we expect our estimate W to be "close" to the true estimate W^*. Therefore, we expect $\hat{\varepsilon}_t$ to be "close" to the true residual ε_t. Then, sampling from $\{\hat{\varepsilon}\}_{t=2}^T$ seems to be a good alternative when there is no other way to retrieve new data $\mathbf{X}'$ or residuals ε'_t.

As we require the residuals to be centered around zero, we will first subtract the average from our estimated residuals $\hat{\varepsilon}_t$,

$$\tilde{\varepsilon}_t = \hat{\varepsilon}_t - \bar{\varepsilon}, \qquad \text{where } \bar{\varepsilon} = \frac{1}{T-1}\sum_{t=2}^{T}\hat{\varepsilon}_t. \tag{6.43}$$

where $\bar{\varepsilon}$ represents the sample mean of the estimated residuals.

Next, we can start sampling *with replacement* from our set of centered residuals $\{\tilde{\varepsilon}_t\}_{t=2}^{T}$. Suppose we want to resample a VAR(1) model with T_2 samples. First, we generate T_2 "bootstrapped" residuals $\{\varepsilon'_t\}_{t=1}^{T_2}$ by repeatedly selecting one of the centered residuals with uniform probability,

$$\varepsilon'_i = \tilde{\varepsilon}_t \text{ with probability } \frac{1}{T-1} \; \forall \; t = 2, \dots, T, \; i = 1, \dots, T_2 \; . \tag{6.44}$$

Now, given these new residuals, we can generate our data $\mathbf{X}'$ using our estimated coefficient matrix W,

$$X'_{t,\cdot} = X'_{t-1,\cdot} W + \varepsilon'_t, \qquad 2 \leq t \leq T_2, \tag{6.45}$$

with initial state $X'_{1,\cdot} = \varepsilon'_1$. The procedure of generating a new data matrix $\mathbf{X}' \in \mathbb{R}^{T_2 \times p}$, called the autoregressive bootstrapping procedure, is given in Algorithm 6.7.

Algorithm 6.7: Bootstrapping for VAR(1) models

Input: A data matrix $\mathbf{X} \in \mathbb{R}^{T \times p}$, an estimate $W \in \mathbb{R}^{p \times p}$ for the coefficient matrix.
Output: A bootstrapped data matrix $\mathbf{X}' \in \mathbb{R}^{T \times p}$.

1: compute the residuals

$$\hat{\varepsilon}_t = X_{t,\cdot} - X_{t-1,\cdot} W, \; t = 2, \dots T.$$

2: center the residuals

$$\tilde{\varepsilon}_t = \hat{\varepsilon}_t - \frac{1}{T-1} \sum_{t=2}^{T} \hat{\varepsilon}_t, \; t = 2, \dots T.$$

3: sample $2T$ new residuals ε'_t uniformly at random from $\{\tilde{\varepsilon}_t\}_{t=2}^{T}$ with replacement
4: set $X'_{1,\cdot} = \varepsilon'_1$
5: generate the rest of the bootstrapped data matrix $\mathbf{X}'$ as

$$X'_{t,\cdot} = W X'_{t-1,\cdot} + \varepsilon'_t, \; t = 2, \dots, 2T.$$

6: **return** the final T time steps of $\mathbf{X}'$ to ensure stationarity

Note that the bootstrapped data matrix $\mathbf{X}'$ is not in its stationary distribution from the first measurement $X'_{1,\cdot}$ immediately. As $X'_{1,\cdot} = \varepsilon'_1$, these values may be much closer to zero than is expected of $\mathbf{X}'$ in its stationary distribution. Therefore, we generate a bootstrap sample of length $2T$ and then only use the last T samples. Then, the bootstrapped data matrix $\mathbf{X}'$ is hopefully in its stationary distribution from the outset. For coefficient matrices who are close to being non-stationary, we expect this to be particularly effective.

Another approach would be to use either the first value $X_{1,\cdot}$ or the value at any random time step $X_{t,\cdot}$ as the initial value. However, we do not know whether the the original data matrix $\mathbf{X}$ was in its stationary distribution from the beginning.

Example 6.11 Bootstrapping VAR(1) model until stationarity.

Suppose we want to bootstrap a new data matrix $\mathbf{X}' \in \mathbb{R}^{50 \times 3}$ based on a data matrix $\mathbf{X} \in \mathbb{R}^{50 \times 3}$ and an estimated coefficient matrix

$$\hat{W} = \begin{pmatrix} 0.9 & 0.0 & 0.0 \\ -0.5 & 0.9 & 0.0 \\ 0.5 & 0.5 & 0.9 \end{pmatrix}.$$

Now, using Algorithm 6.7, we would simply set $X'_{1,\cdot} = \varepsilon'_1$. However, this means that the first values of $\mathbf{X}'$ are not near its stationary distribution, as the values are set artificially close to zero. Some time steps may be required until the data matrix $\mathbf{X}'$ is in its stationary distribution. This behavior is showcased in Figure 6.7.

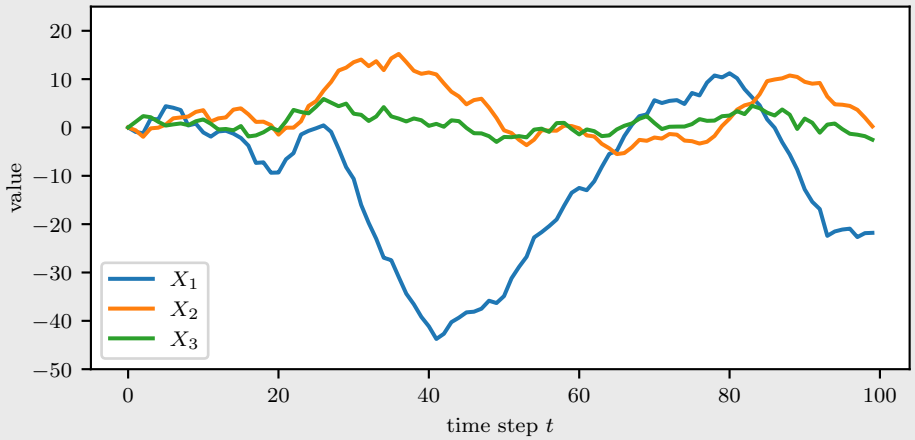


Figure 6.7: Visualization of the bootstrapped sample $\mathbf{X}'$ from Example 6.11, showcasing that $\mathbf{X}'$ is not in its stationary distribution from the beginning.

Some time steps are required until $\mathbf{X}'$ reaches its stationary distribution, say $t = 30$. If we want $\mathbf{X}' \in \mathbb{R}^{50\times 3}$ to start in its stationary distribution, we should not select the first 50 time steps of $\mathbf{X}'$, but rather the final 50 time steps to get a stationary bootstrapped data matrix $\mathbf{X}'$. For this reason, we bootstrap a data matrix $\mathbf{X}'$ consisting of $2T$ time steps, and use only the final T time steps of $\mathbf{X}'$, as is depicted in the final line of Algorithm 6.7.

We have seen that we can sensibly generate a data matrix $\mathbf{X}'$ using an autoregressive bootstrapping procedure as described in Algorithm 6.7. To solve the problem addressed in the introduction of this chapter, we need to employ this bootstrapping procedure to determine the important arcs in W. We have developed two approaches that utilize this bootstrapping procedure.

Using bootstrapping to determine marginal confidence intervals of W. One approach is to use this bootstrap approach to compute marginal confidence intervals of the coefficients of W. If we bootstrap a total of N bootstrap samples $\{\mathbf{X}'^{(n)}\}_{n=1}^{N}$, we can use one of the developed methods to compute a total of N acyclic coefficient matrix $\{W^{(n)}\}_{n=1}^{N}$. We can then calculate the $\alpha/2$th and $(100-\alpha/2)$th percentiles of each coefficient w_{ij} in W, yielding us an empirical $(1-\alpha)$%-confidence interval of each coefficient of W. If the value 0 is contained in the confidence interval, then we can argue that this coefficient is irrelevant and can therefore be set to zero.

The corresponding pseudocode has been given in Algorithm 6.8.

Algorithm 6.8: Using bootstrapping to recover marginal confidence intervals.

Input: A data matrix $\mathbf{X} \in \mathbb{R}^{T\times p}$, an coefficient matrix estimate $W \in \mathbb{R}^{p\times p}$, a total number of bootstrap samples N, and a confidence level $0 \leq \alpha < 0.5$.
Output: $W_L, W_U \in \mathbb{R}^{p\times p}$, containing the bootstrapped $\alpha/2$th and $1-\alpha/2$th percentiles of the coefficient matrix W.

1: sample N bootstrapped data matrices $\mathbf{X}'^{(n)}, n = 1,\ldots,N$ with length T using Algorithm 6.7 with data matrix $\mathbf{X}$ and coefficient matrix W
2: estimate N the coefficient matrix $W^{(n)}$ on the bootstrapped data matrices $\mathbf{X}^{(n)}$
3: $W_L \leftarrow$ the $\alpha/2$th percentile for each coefficient from the set $\{W^{(n)}\}_{n=1}^{N}$
4: $W_U \leftarrow$ the $(1-\alpha/2)$th percentile for each coefficient from the set $\{W^{(n)}\}_{n=1}^{N}$
5: **return** W^L, W^U

We can use these percentiles to determine whether an arc or coefficient in W is significantly different from zero. The coefficients of the thresholded matrix $W_{\text{threshold}}$ are then

$$w_{\text{threshold},ij} = \begin{cases} 0 & \text{if } w_{ij}^L \leq 0 \leq w_{ij}^U, \\ w_{ij} & \text{otherwise.} \end{cases} \tag{6.46}$$

Example 6.12 Using bootstrapped confidence intervals to recover important arcs.

Let us consider the four dimensional setting with 250 time steps visualized in Figure 6.8, where $\mathbf{X}$ has been generated according to a VAR(1) model, where the data generating matrix W^* has been given on the left-hand side of Equation 6.47.

Furthermore, assume that we have estimated the matrix W on the right-hand of Equation 6.47 using `DAG-OMP`. Note, however, that we could have used any method that estimates an acyclic coefficient matrix. We see that the correct arcs have been recovered, but there are also several arcs which originate from spurious correlation with the noise.

$$W^* = \begin{pmatrix} 0.85 & 0.00 & 0.00 & 0.00 \\ 0.00 & 0.85 & 0.00 & 0.00 \\ -0.28 & 0.26 & 0.85 & 0.00 \\ 0.46 & 0.00 & 0.00 & 0.85 \end{pmatrix}, \quad W = \begin{pmatrix} 0.85 & 0.00 & 0.00 & 0.00 \\ 0.03 & 0.90 & 0.00 & -0.01 \\ -0.29 & 0.19 & 0.90 & 0.01 \\ 0.50 & 0.00 & 0.00 & 0.83 \end{pmatrix} \tag{6.47}$$

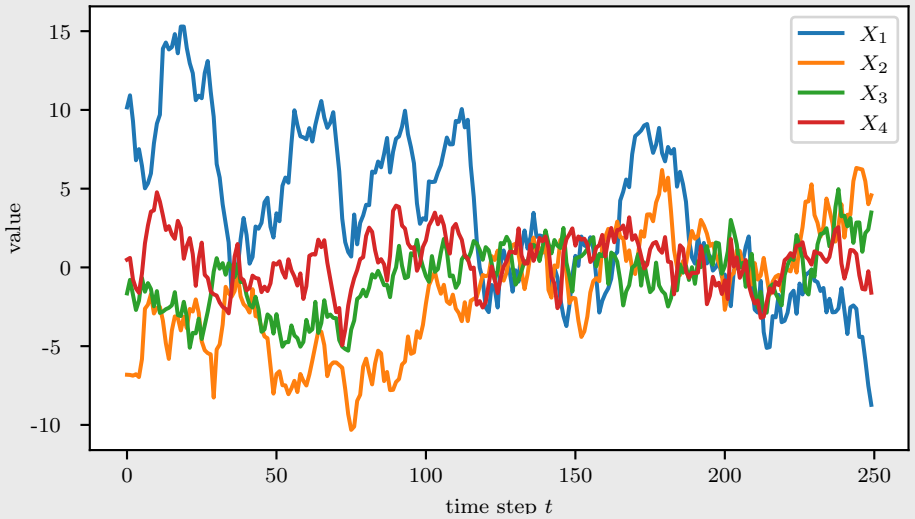


Figure 6.8: Visualization of the data matrix $\mathbf{X}$ from Example 6.12.

We will now generate 1000 bootstrapped data matrices $\{\mathbf{X}'^{(n)}\}_{n=1}^{1000}$ and use `DAG-OMP` again to estimate a suitable coefficient matrix for each bootstrapped data matrix, yielding 1000 coefficient matrices $\{W^{(n)}\}_{n=1}^{1000}$. Constructing the 10th and 90th percentile for each coefficient w_{ij} using $\{W^{(n)}\}_{n=1}^{N}$ yields percentile matrices in Equation 6.48.

$$W_L = \begin{pmatrix} 0.82 & -0.03 & 0.00 & 0.00 \\ 0.08 & 0.84 & 0.00 & -0.04 \\ -0.35 & 0.14 & 0.84 & 0.05 \\ 0.43 & -0.06 & -0.06 & 0.75 \end{pmatrix}, W_U = \begin{pmatrix} 0.88 & 0.02 & 0.00 & 0.00 \\ 0.00 & 0.92 & 0.00 & 0.02 \\ -0.23 & 0.26 & 0.93 & 0.06 \\ 0.59 & 0.07 & 0.06 & 0.88 \end{pmatrix}. \tag{6.48}$$

Keeping the coefficients where the value 0 lies completely outside the percentile range yields the thresholded matrix

$$W_{\text{threshold}} = \begin{pmatrix} 0.85 & 0.00 & 0.00 & 0.00 \\ 0.00 & 0.90 & 0.00 & 0.00 \\ -0.29 & 0.19 & 0.90 & 0.00 \\ 0.50 & 0.00 & 0.00 & 0.83 \end{pmatrix},$$

which indeed retrieves all the correct entries.

Bootstrapping to verify the recoverability of coefficients. A second approach to distinguish true coefficients from noise is by inspecting the rate of *recoverability* of the coefficients. The intuition is that if an entry in the coefficient matrix W is important, then we will also recover this element after bootstrapping. For this, we require an ordering of the arcs from most important to least important. To obtain such an ordering, we can for example use a forward iterative approach such as `DAG-OMP`, or we can iteratively remove coefficients from an acyclic structure using a backward approach as we have done in Example 6.9. The pseudocode is given in Algorithm 6.9.

Algorithm 6.9: Using bootstrapping to inspect the recoverability of edges.

Input: A list of coefficient matrices $(W_k)_{k=0}^{K}$, representing the coefficient matrix W after the kth iteration of the `DAG-OMP` algorithm, and N, the number of bootstrap samples per coefficient matrix W_k.
Output: A list `arcs_missed`, representing the average number of arcs in W_k that were not recovered using `DAG-OMP` on bootstrapped data matrices.

1: let `arcs_missed` be a list of length K
2: **for** each coefficient matrix W_k **do**
3: use W_k to generate N bootstrapped data matrices $X^{(n)}, n = 1, \dots, N$
4: use `DAG-OMP` to estimate the coefficient matrix $\hat{W}_k^{(n)}$ for each $X^{(n)}$, where $\hat{W}_k^{(n)}$ is obtained by terminating `DAG-OMP` after k iterations
5: let B_k and $\hat{B}_k^{(n)}$ be the support matrices of W_k and $\hat{W}_k^{(n)}$, respectively
6: compute the average number of arcs in W_k that were not recovered,

$$\texttt{arcs\_missed}[k] = \frac{1}{2N} \sum_{n=1}^{N} \left\| B_k - \hat{B}_k^{(n)} \right\|_0$$

7: **end for**
8: **return** `arcs_missed`

Example 6.13 Using bootstrapping to inspect the recoverability of arcs.

Let us consider the same four-dimensional setting on $T = 250$ time steps as in Example 6.12. After applying `DAG-OMP` on $\mathbf{X}$, we obtain a sequence of coefficient matrices $(W_k)_{k=0}^{10}$, where W_k corresponds to the coefficient matrix at iteration k of the `DAG-OMP` algorithm.

Now, for each matrix W_k, we bootstrap 100 data matrices $\{\mathbf{X}_k^{(n)}\}_{n=1}^{100}$, and use `DAG-OMP` to obtain 100 coefficient matrices $\{W_k^{(n)}\}_{n=1}^{100}$ that contain exactly k arcs. We then compute the average number of arcs that were not recovered. Plotting the average number of missed arcs as a function of k, the number of arcs, yields the plot in Figure 6.9.

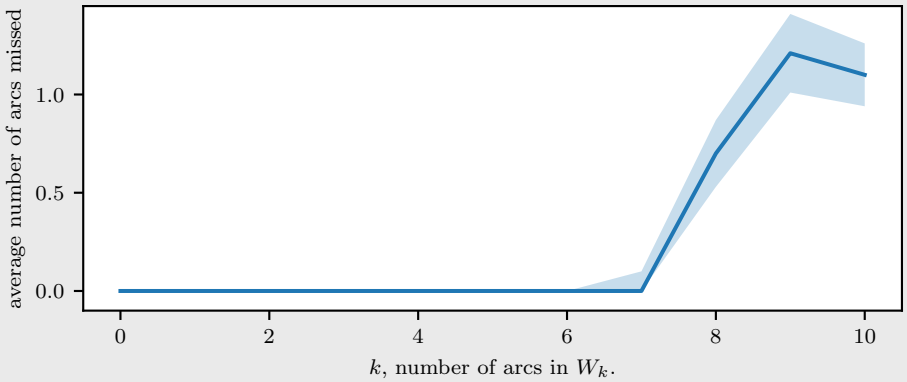


Figure 6.9: Plot of `arcs_missed` from Example 6.13.

The number of missed arcs remains close to zero until we have recovered all true arcs at $k = 7$. Intuitively, this means that these arcs were important enough to be recovered. Once we consider W_8, an arc is often missed, approximately 80% of the time. This implies that arcs 8, 9, and 10 were all not significant enough to be recovered, indicating that W_7 would be the most suitable matrix. Indeed, the support of W_7 is equal to the support of the matrix $W_{\text{threshold}}$ we recovered using the confidence interval approach in Example 6.12.

Bootstrapping performance in higher dimensions Having explained the two bootstrapping approaches to distinguish true edges from noise in Example 6.12 and Example 6.13, let us now consider higher-dimensional settings with fewer time steps T.

Example 6.14 Applying the bootstrapping approaches in the high-dimensional setting.

Let us consider a ten-dimensional time series with one hundred time steps from Example 6.10. The data generating matrix W^* consists of 25 arcs, of which 15 are off-diagonal. Applying `DAG-OMP` yields a sequence of coefficient matrices $(W_k)_{k=0}^{55}$, where the index k corresponds to the number of non-zero entries in W_k. Referring back to the mean squared error achieved by each W_k in Figure 6.5, we indeed see that the mean squared error plateaus around $k = 25$, corresponding to the total number of edges in W^*.

We can first apply the bootstrapping approach to compute the 2.5th and 97.5th percentile of each coefficient in W. We have again used 1000 bootstrapped data matrices containing an equal number of time steps as $\mathbf{X}$. If we remove all arcs where the value zero lies inside its confidence interval, we managed to exactly recover all true coefficients and no arc more.

Secondly, we can use bootstrapping to investigate the recoverability of the arcs in each matrix W_k. The average number of arcs that were missed has been plotted in Figure 6.10 as a function of k. The first 25 arcs were deemed important enough to be recovered almost always. From $k = 25$ onward, the number of missed arcs steadily increases, indicating that these arcs were not important enough to be recovered consistently. This also indicates that there are indeed approximately 25 true arcs, and thus W_{25} here seems the best trade-off between predictive performance and model complexity.

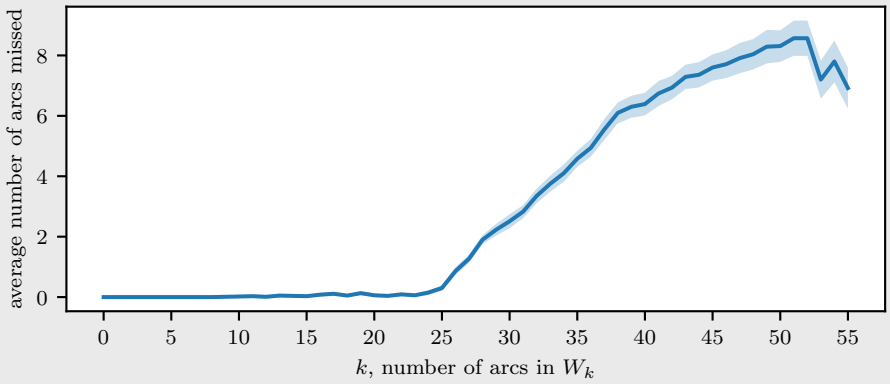


Figure 6.10: Plot `arcs_missed` from Example 6.14, showcasing the steep incline at $k = 25$.

6.4.2 Cross-Validation

If the data were independent, then cross-validation would be a sensible choice. As first introduced in [71], cross-validation is a technique where we split the available data in a training set and a validation set. We would train on the training set, and then verify the predictive performance on the unseen validation set. For cross-validation, we use the same data matrix to randomly generate several train and validation sets, and average the performance of the different models on their corresponding validation sets. As the data is reused as both training data and validation data several times, we call such an approach cross-validation.

However, in our setting, we do not have independent data. The measurements of the current time step $X_{t,\cdot}$ depend heavily on the previous time step $X_{t-1,\cdot}$. Therefore, if we split our data and train on the training data, then the validation set does not technically contain unseen data, as some of the information of the training set has "bled" into the validation set. Therefore, cross-validation should be done carefully when the data contains dependencies. Several principles of proper validation, the "do's and don'ts" of cross-validation, are explained in greater detail in [23].

Nevertheless, several cross-validation techniques for autoregressive models have been employed before, and the validity of cross-validation on AR(1) models seems reasonable under certain assumptions, such as having a stationary time series, having a consistent estimator, and making sure that the noise components form a Martingale difference sequence [5]. Therefore, we will naively disregard the dependencies. Even though cross-validation is only known to be sensible for independent data, we show empirically that this approach seems suitable for model selection in dependent data.

Leave-one-out cross-validation. We will naively regard our data matrix $\mathbf{X} \in \mathbb{R}^{T\times p}$ as a set $D = \{X_{t-1,\cdot}, X_{t,\cdot}\}_{t=2}^{T}$, consisting of $T-1$ independent features and labels.

For each of the $T-1$ pairs, we will construct the training set as $D_{\text{Tr}}^{(-t)} = D \setminus \{(X_{t-1,\cdot}, X_{t,\cdot})\}$, and the validation set will simply be the left out pair $D_V^{(t)} = \{(X_{t-1,\cdot}, X_{t,\cdot})\}$. As we leave out one pair each time, this approach is called "leave-one-out cross-validation". We estimate a suitable coefficient matrix $W^{(-t)}$ using the data in training set $D_{\text{Tr}}^{(t)}$. We then compute the leave-one-out cross-validation score of $W^{(-t)}$ on the validation set. The score for the validation set $D_V^{(t)}$ then will be

$$\text{LOOCV}_t\left(W^{(-t)}\right) = \left\| X_{t,\cdot} - X_{t-1,\cdot}W^{(-t)} \right\|_2^2, \qquad t = 2,\dots,T. \tag{6.49}$$

Now, the average LOOCV score for W is the average across the $T-1$ validation sets,

$$\text{LOOCV}(W) = \frac{1}{T-1}\sum_{t=2}^{T} \text{LOOCV}_t\left(W^{(-t)}\right). \tag{6.50}$$

We expect that true coefficient will decrease the leave-one-out cross-validation score, as the arc provides a significant increase to the predictive performance. However, estimated coefficients which were equal to zero in W^* are estimated because of spurious correlations with the noise. Although these correlations may be apparent in D_{Tr}, these will most likely not appear in D_V. Therefore, we expect the leave-one-out cross-validation score to decrease as long as we add correct arcs to W_k, and to increase when we start adding incorrect arcs to W_k.

Therefore, there will be a specific value k^* that minimizes this LOOCV score. This value for k^* will hopefully correspond to the number of non-zero coefficients in W^*, such that selecting W_{k^*} corresponds to the most suitable matrix. A great advantages of such an approach is that there is no additional parameter that needs to be chosen.

Example 6.15 Cross-Validation for VAR(1) model selection.

Consider the data matrix $\mathbf{X} \in \mathbb{R}^{100\times 10}$ which we have also used in Example 6.14. The coefficient matrix W^* consists of 25 coefficients, of which 15 are off-diagonal.

We split our data matrix $D = \{\{(X_{t-1,\cdot}, X_{t,\cdot})\}\}_{t=2}^{100}$ into 99 folds of the form $D_t = \{(X_{t-1,\cdot}, X_{t,\cdot})\}$ one for each pair, where $t = 2,\dots,100$. We will use these folds to create 99 combinations of train and validation sets,

$$D_{\text{Tr}}^{(-t)} = D \setminus \{(X_{t-1,\cdot}, X_{t,\cdot})\}, \qquad D_V^{(t)} = \{(X_{t-1,\cdot}, X_{t,\cdot})\}, \qquad t = 2,\dots,T.$$

For each training set we use `DAG-OMP` to obtain a sequence of acyclic coefficient matrices $\left(W_k^{(-t)}\right)_{k=0}^{55}$ and compute the average leave-one-out cross-validation score on $D_V^{(-t)}$ for each coefficient matrix W_k,

$$\text{LOOCV}(W_k) = \frac{1}{T-1} \sum_{t=2}^{T} \left\| X_{t,\cdot} - X_{t-1} W_k^{(-t)} \right\|_2^2 .$$

These leave-one-out cross-validation scores $\text{LOOCV}(W_k)$ are plotted as a function of k, the number of non-zero coefficients in the coefficient matrix W_k in Figure 6.11.

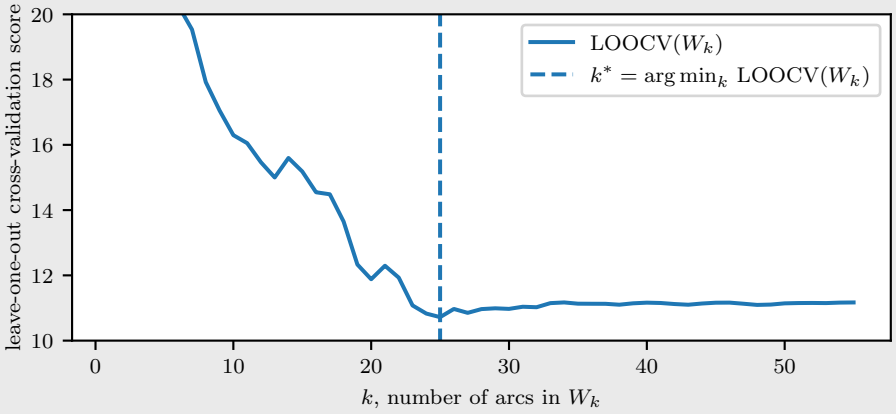


Figure 6.11: Plot of the leave-one-out cross-validation scores as a function of k, the number of arcs in W_k. The smallest leave-one-out cross-validation score was attained for $k = 25$.

We see that the cost value first decreases quite steeply, after which the cost function is minimized for $k = 25$ arcs. After $k = 25$, we first see that the cost slightly increases until around $k = 35$, after which the cost function remains approximately constant. Therefore, we see that a total of $k = 25$ non-zero coefficients seems to be the correct number of arcs. Therefore, we will now apply `DAG-OMP` on the full dataset $\mathbf{X}$, and return the coefficient matrix after iteration $k = 25$. Using this approach, we recovered exactly all true coefficients of W^*, and no arc more, just as both bootstrapping approaches.

6.5 An Analysis of Cross-Validation for AR(1) models.

As we have seen, the leave-one-out cross-validation technique seems to be a suitable approach to regularize our coefficient matrix W. This seems counter-intuitive, as cross-validation is often not advised in time-series models due to the dependency between different time steps [8]. Therefore, we devote this section to analyze the performance of cross-validation in a much simpler setting.

6.5.1 AR(1) setting without mean.

Suppose that we are dealing with a one-dimensional data matrix $X \in \mathbb{R}^T$ which has been generated by an auto regressive (AR) model of order 1,

$$X_t = a_0 X_{t-1} + \varepsilon_t. \tag{6.51}$$

The parameter a_0 represents the auto regressive coefficient, and is assumed to be less than one in absolute value to ensure that the time series is stationary. Furthermore, ε_t represents random noise, which we assume to be independent standard Gaussian random variables,

$$\varepsilon_t \sim \mathcal{N}(0,1) \qquad \forall\, t = 2, \ldots, T. \tag{6.52}$$

To make sure the data matrix X is in its stationary distribution from the very start, we sample X_1 according to its stationary distribution,

$$X_1 \sim \mathcal{N}\left(0, \frac{1}{1-a_0^2}\right). \tag{6.53}$$

Assume we know that X has been generated by an AR(1) model with auto regressive coefficient a_0. We would like to verify whether the data has actually been generated by a_0,

$$H_0 : a = a_0 \text{ versus the alternative hypothesis } H_1 : a \neq a_0.$$

We propose cross-validation to verify H_0, which is an unorthodox approach. We will compare the negative log-likelihood when using $a = a_0$ to the average negative log-likelihood of using $\{a^{(-t)}\}_{t=2}^{T}$ the leave-one-out cross-validation estimates for a. Let $-2LL_0(X)$ represent the negative log-likelihood of a_0 given the data matrix X, multiplied by two,

$$-2LL_0(X) = -2\sum_{t=2}^{T} \log\left(-\frac{1}{\sqrt{2\pi}}\right) + \sum_{t=2}^{T} \left(X_t - a_0 X_{t-1}\right)^2, \tag{6.54}$$

and $-2LL_{\text{LOOCV}}(X)$ the average negative log-likelihood given X of the leave-one-out cross-validation estimates, multiplied by two,

$$-2LL_{\text{LOOCV}}(X) = -2\sum_{t=2}^{T} \log\left(-\frac{1}{\sqrt{2\pi}}\right) + \sum_{t=2}^{T} \left(X_t - \hat{a}^{(-t)} X_{t-1}\right)^2. \tag{6.55}$$

Here, $\hat{a}^{(-t)}$ denotes the maximum likelihood estimate of a given the data without (X_{t-1}, X_t),

$$\hat{a}^{(-t)} = \frac{\sum_{i=2}^{T} X_{i-1}X_i - X_{t-1}X_t}{\sum_{i=2}^{T} X_i^2 - X_{t-1}^2}. \tag{6.56}$$

We accept the null-hypothesis H_0 if $-2LL_0(X) \leq -2LL_{\text{LOOCV}}(X)$, and reject the null-hypothesis otherwise. Let us do some simulations to verify how often we correctly accept the null-hypothesis.

Simulation Study. For varying values of a_0 and T, we have simulated a total of N data matrices $X^{(n)} \in \mathbb{R}^T, n = 1 \ldots, N$ according to this AR(1) model. For each data matrix, we have computed $-2LL_0(X)$ and $-2LL_{\text{LOOCV}}(X)$, and counted how often we correctly accept H_0, i.e., $-2LL_{\text{LOOCV}}(X) \leq -LL_0(X)$. The ratio of correct acceptances of H_0 is then equal to

$$RCH_0(a, T) = \frac{1}{N}\sum_{n=1}^{N} \mathbf{1}\left\{-2LL_0\left(X^{(n)}\right) \leq -2LL_{\text{LOOCV}}\left(X^{(n)}\right)\right\}. \tag{6.57}$$

The value $RCH_0(a, T)$ has been computed for the values $a \in \{0.0, 0.3, 0.6, 0.9, 0.95, 0.99, 0.995\}$ and $T \in \{25, 50, 100, 250, 500, 1000\}$. For each (a_0, T) pair, we used $N = 1{,}000{,}000$ data matrices $X^{(n)}$ to estimate $RCH_0(a_0, T)$. The values $RCH_0(a_0, T)$ have been plotted as a function of a_0 in Figure 6.12 and as a function of T in Figure 6.13.

We accept the null-hypothesis using cross-validation quite often, ranging from approximately 80% to 85% of the time. Therefore, cross-validation seems to be sensible for model selection, even when the data is dependent.

Let us first consider the ratio of correct H_0 acceptances as a function of the autoregressive coefficient a_0. For $a = 0.0$ until $a = 0.900$, the ratio of correct acceptance of H_0 remains constant for the different values of T. However, as a_0 tends closer and closer to one, the dependency increases, and we see that the ratio of correct acceptance of H_0 decreases slightly. This decrease is especially visible for smaller values of T.

Secondly, let us investigate the ratio of correct H_0 acceptances as a function of T. When we have more time steps, the ratio of correct acceptance of H_0 increases. For $a = 0.0$ until $a = 0.95$, the increase happens quite soon, and already attains its maximum around $T = 250$, after which it remains constant. For values of a_0 closer to one, there is more dependency between the time steps in X, and therefore we also see that the ratio increases more slowly. Nevertheless, these ratios also seem to converge to the same value.

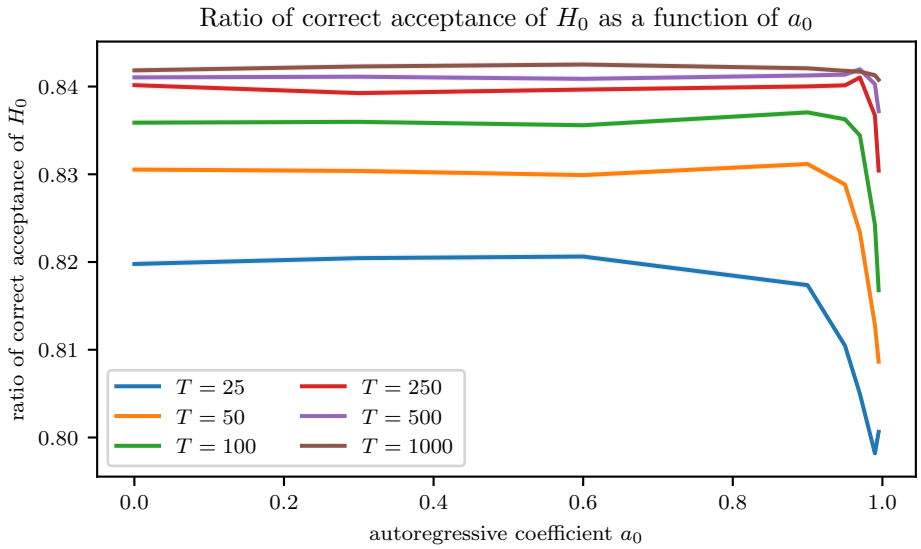


Figure 6.12: Plot of $RCH_0(a_0, T)$ as a function of the autoregressive coefficient a_0 for varying values of the number of time steps T.

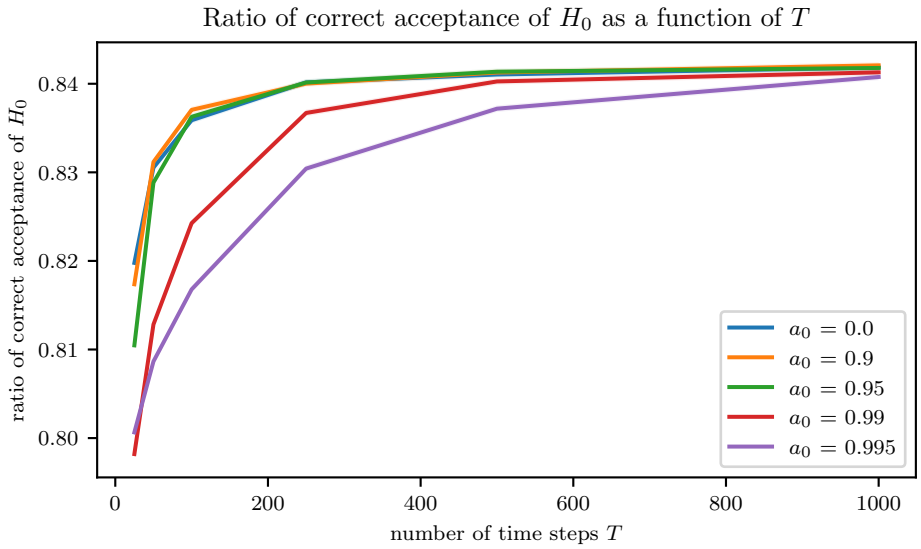


Figure 6.13: Plot of $RCH_0(a_0, T)$ as a function of the number of time steps T for varying values of the autoregressive coefficient a_0.

Asymptotic value of $RCH_0(a_0, T)$. What is especially interesting is that these ratios do not converge to one, yet they seem to converge to some value around 0.84, regardless of the value for $a_0 < 1$. For values of a_0 closer to one, convergence requires a larger number of time steps T, but also then the ratio converges to approximately 0.84.

Our conjecture regarding this asymptotic value is provided in Conjecture 6.1.

Conjecture 6.1 Asymptotic value of $RCH(a_0, T)$

Let $|a_0| < 1$. Furthermore, let X be generated according to an AR(1) model. Then, we conjecture that the difference between the two negative log-likelihoods as defined in Equation 6.54 and Equation 6.55 converges to a chi-squared random variable with one degree of freedom, shifted two units to the left. That is,

$$\lim_{T\to\infty} -2LL_0(X) + 2LL_{\text{LOOCV}}(X) \sim \chi_1^2 - 2. \tag{6.58}$$

Moreover, let $RCH_0(a_0, T)$ denote the ratio of correct acceptance of H_0 as defined in Equation 6.60. Then, we conjecture the value of $RCH_0(a_0, T)$ converges to $\mathbb{P}(Q \leq 2)$ as T tends to infinity, where Q is a chi-squared random variable with one degree of freedom,

$$\lim_{T\to\infty} \mathbb{E}\left[RCH_0(a_0, T)\right] = \text{const}, \quad \text{where const} = \mathbb{P}(Q \leq 2) \text{ and } Q \sim \chi_1^2. \tag{6.59}$$

We want to stress that this is a conjecture, not a theorem or lemma, as a formal argument is not provided. Nevertheless, we will provide convincing arguments that support Conjecture 6.1.

Now, assume that Equation 6.58 is indeed true. Then,

$$\begin{aligned}
\lim_{T\to\infty} \mathbb{E}\left[RCH_0(a, T)\right] &= \lim_{T\to\infty} \mathbb{E}\left[\mathbf{1}\left\{-2LL_0\left(X\right) \leq -2LL_{\text{LOOCV}}\left(X\right)\right\}\right] \\
&= \lim_{T\to\infty} \mathbb{P}\left(-2LL_0\left(X\right) \leq -2LL_{\text{LOOCV}}\left(X\right)\right) \\
&= \lim_{T\to\infty} \mathbb{P}\left(-2LL_0(X) + 2LL_{\text{LOOCV}}(X) \leq 0\right) \\
&= \mathbb{P}\left(Q \leq 2\right), \qquad \text{where } Q \sim \chi_1^2.
\end{aligned}$$

Using the cumulative distribution function of the chi-squared distribution, we see that

$$\mathbb{P}(Q \leq 2) = \frac{\gamma(1/2, 1)}{\Gamma(1/2)} = \frac{1}{\sqrt{\pi}} \sum_{k=0}^{\infty} \frac{e^{-1}}{1/2 \cdot (1/2+1) \cdots \cdots \cdot (1/2+k)} \approx 0.8427,$$

which indeed closely resembles the limit we see in Figure 6.13.

To further verify our conjecture, we consider the histogram of $-2LL_0(X) + 2LL_{\text{LOOCV}}(X)$ for $a = 0.5$ and $T = 10{,}000$. We generate 50,000 data matrices $X \in \mathbb{R}^T$ and compute $-2LL_0(X) + 2LL_{\text{LOOCV}}(X)$. These values are plotted in Figure 6.14, along with the probability density function of $Q \sim \chi_1^2 - 2$.

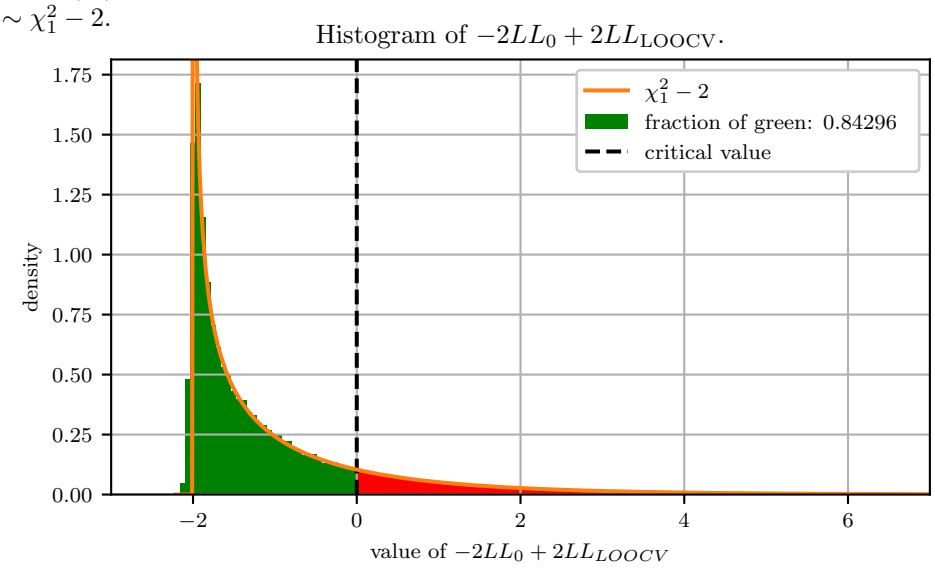


Figure 6.14: Histogram of $-2LL_0(X) + 2LL_{\text{LOOCV}}(X)$, computed from 50,000 data matrices $X \in \mathbb{R}^T$ with $T = 10{,}000$ using an AR(1) model with $a = 0.5$.

The random variable $-2LL_0(X) + 2LL_{\text{LOOCV}}(X)$ indeed perfectly follows a chi squared distribution with one degree of freedom, shifted two units to the left. Here, a value smaller than zero indicates that $-2LL_0(X) \leq -2LL_{\text{LOOCV}}(X)$, indicating that we correctly accept reject H_0. The bars where $-2LL_0(X) \leq -2LL_{\text{LOOCV}}(X)$ have been plotted in green and the ratio of green bars indeed corresponds to a value of approximately 0.84296, close to the value $\mathbb{P}(Q \leq 2) \approx 0.8427$.

A more careful derivation that further supports our claim in Conjecture 6.1 has been provided in Appendix A.3.

Modifying the type-I error Equation 6.59 unveils an interesting relation between the leave-one-out cross-validation score and hypothesis testing. We can use the leave-one-out cross-validation score to determine whether a time series X has indeed been generated using the auto regressive coefficient a_0. Without changing the leave-one-out cross-validation score, we know that the ratio of correctly accepting H_0 is equal to $P(Q \leq 2) \approx 0.8427$, where $Q \sim \chi^2_1$. However, we can modify this to adjust the confidence of correctly accepting the null-hypothesis, otherwise known as the type-1 error α. For example, if we prefer a type-II error of $\alpha = 0.05$, that means we are looking for a k such that $\mathbb{P}(Q \leq k) = 0.95 \Rightarrow k \approx 3.8414$ by looking up the quantiles of the chi-squared distribution. We could achieve this by changing our decision rule to

$$RCH_0(a, T) = \frac{1}{N} \sum_{n=1}^{N} \mathbf{1} \left\{ -2LL_0 \left(X^{(n)} \right) \leq -2LL_{\text{LOOCV}} \left(X^{(n)} \right) + 1.8414 \right\}. \tag{6.60}$$

6.5.2 AR(1) Setting with mean.

We will now investigate the setting in which the autoregressive model consisted of two parameters, an autoregressive coefficient a_0 and a mean of b_0. Now, the model is defined as

$$X_t = b_0 + a_0(X_{t-1} - b_0) + \varepsilon_t. \tag{6.61}$$

Again, ε_t represents random noise, which we assume to be identically and independently standard normal distributed,

$$\varepsilon_t \sim \mathcal{N}(0, 1) \; \forall \; t = 2, \ldots, T.$$

To make sure the data matrix X is in its stationary distribution from the very start, we sample X_1 according to its stationary distribution,

$$X_1 \sim \mathcal{N}\left(b_0, \frac{1}{1 - a_0^2} \right).$$

The setting is now as follows. Assume X has been generated by an AR(1) model with auto regressive coefficient a_0 and mean b_0. We would now like to verify whether the data has actually been generated by a_0,

$$H_0 : a = a_0 \text{ versus the alternative hypothesis } H_1 : a \neq a_0.$$

We will again use cross-validation, hoping it will be effective just an for the AR(1) setting without any mean. The difference between the previous setting is that we now need to estimate b as well.

Deriving $-2LL_0(X)$. We will compare the negative log-likelihood when using $a = a_0$ multiplied by two against the average negative log-likelihood of using $\{a^{(-t)}\}_{t=1}^{T-1}$, the leave-one-out cross-validation estimates for a. However, we now also need to estimate b.

Let $-2LL_0(X)$ represent the negative log-likelihood, multiplied by two, of a_0 given the data matrix X, which is now defined as

$$-2LL_0(X) = -2 \sum_{t=2}^{T} \log \left(-\frac{1}{\sqrt{2\pi}} \right) + \sum_{t=2}^{T} \left((X_1 - \hat{b}_0^{(-t)}) - a_0 \left(X_{t-1} - \hat{b}_0^{(-t)} \right) \right)^2, \tag{6.62}$$

where $\hat{b}_0^{(-t)}$ corresponds to the leave-one-out maximum likelihood estimator given a_0 and X, where we have left out the pair (X_{t-1}, X_t),

$$\hat{b}_0^{(-t)} = \frac{\left(\sum_{k=2}^{T} X_k - X_t\right) - a_0 \left(\sum_{k=1}^{T-1} X_k - X_t\right)}{(T-2)(1-a_0)}. \tag{6.63}$$

Deriving $-2LL_{\text{LOOCV}}(X)$. Deriving the negative log-likelihood using leave-one-out cross-validation to derive both $\hat{a}^{(-t)}$ and $\hat{b}^{(-t)}$ is slightly more involved. We cannot use Equation 6.63, as we do not have an estimate for $\hat{a}^{(-t)}$. If we had known $\hat{a}^{(-t)}$, then the estimate for $\hat{b}^{(-t)}$ would be

$$\hat{b}^{(-t)} = \frac{\left(\sum_{k=2}^{T} X_k - X_t\right) - \hat{a}^{(-t)} \left(\sum_{k=1}^{T-1} X_k - X_t\right)}{(T-2)(1-\hat{a}^{(-t)})}. \tag{6.64}$$

Analogously, if we had known $\hat{b}^{(-t)}$, then the estimate for $\hat{a}^{(-t)}$ would be

$$\hat{a}^{(-t)} = \frac{\sum_{i=2}^{T} \left(X_{i-1} - \hat{b}^{(-t)}\right)\left(X_i - \hat{b}^{(-t)}\right) - \left(X_{t-1} - \hat{b}^{(-t)}\right)\left(X_t - \hat{b}^{(-t)}\right)}{\sum_{i=2}^{T} \left(X_i - \hat{b}^{(-t)}\right)^2 - \left(X_{t-1} - \hat{b}^{(-t)}\right)^2}. \tag{6.65}$$

Now, we could do an iterative approach, where we will initially estimate $\hat{b}$ without $\hat{a}^{(-t)}$ as $\hat{b}^{(-t)} \approx (T-2)^{-1} \sum_{i=2}^{T} X_i - X_t$. Then, we iteratively use Equation 6.64 and Equation 6.65 to update our leave-one-out coefficients $\hat{a}^{(-t)}$ and $\hat{b}^{(-t)}$ until the negative log-likelihood has converged. Then, the corresponding negative log-likelihood, multiplied by two, corresponds to

$$-2LL_{\text{LOOCV}}(X) = -2\sum_{t=2}^{T} \log\left(-\frac{1}{\sqrt{2\pi}}\right) + \sum_{t=2}^{T} \left(\left(X_t - \hat{b}^{(-t)}\right) - \hat{a}^{(-t)}\left(X_{t-1} - \hat{b}^{(-t)}\right)\right)^2. \tag{6.66}$$

We again accept the null-hypothesis if the negative log-likelihood under H_0 is smaller than when we would use the leave-one-out cross-validation score, that is, when $-2LL_0(X) \leq -2LL_{\text{LOOCV}}$.

Simulation Study. Let us first investigate how b_0 affects the ratio of correct acceptance. For varying values of b_0 and a fixed value $a_0 = 0.5$ and $T = 100$, we have generated a total of $N = 10{,}000$ data matrices $X^{(n)} \in \mathbb{R}^T, n = 1 \ldots, N$ of length T according to this AR(1) model.

For each data matrix, we compute $-2LL_0(X)$ and $-2LL_{\text{LOOCV}}(X)$ according to Equation 6.62 and Equation 6.66, respectively. Subsequently, we count how often we correctly accept H_0, i.e., $-2LL_0(X) \leq -2LL_{\text{LOOCV}}(X)$.

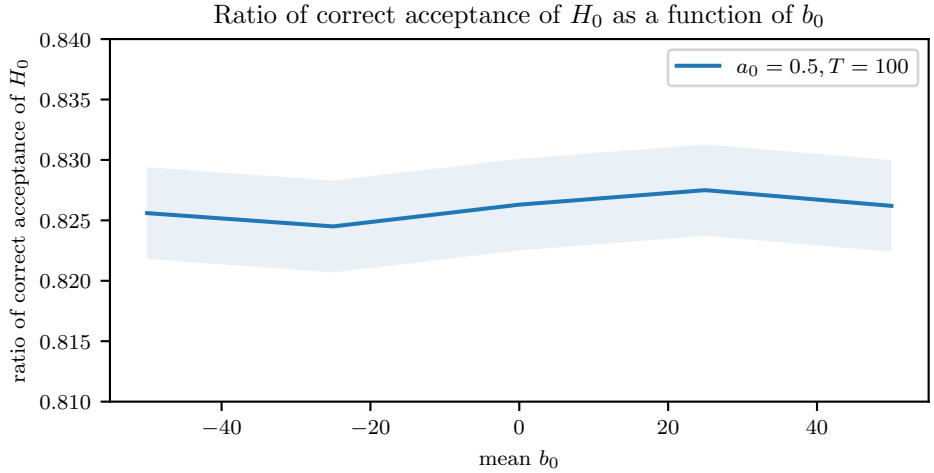


Figure 6.15: Plot of $RCH_0(a_0, b_0, T)$ for varying values of the mean b_0. Shaded parts correspond to the standard errors.

The value for b_0 does not influence the value for $RCH_0(a_0, b_0, T)$. We know that both $\hat{b}_0^{(-t)}$ and $\hat{b}^{(-t)}$ are both estimated using cross-validation using Equation 6.63 and Equation 6.64, respectively, albeit with a different value for a. Therefore, as the estimation procedures are similar, it is not unexpected that the value for b_0 does not affect $RCH_0(a_0, b_0, T)$.

Now, let us see how the values for a and T influence the value of $RCH_0(a_0, b_0, T)$. Based on the analysis in Subsection 6.5.1, we expect the value of $RCH_0(a_0, b_0, T)$ to decrease when a_0 is close to one or when T is small. We have again generated $N = 10{,}000$ data matrices for each (a_0, T) pair in the two dimensional range of values for $a_0 \in \{0.0, 0.25, 0.6, 0.9, 0.95, 0.99, 0.995\}$ and $T \in \{25, 50, 100, 250, 500, 1000\}$. The values $RCH_0(a_0, b_0, T)$ have been plotted as a function of a_0 in Figure 6.16 and as a function of T in Figure 6.17. As the value for b_0 did not have any effect on $RCH_0(a_0, b_0, T)$, we have fixed $b_0 = 0$, but will still be estimated.

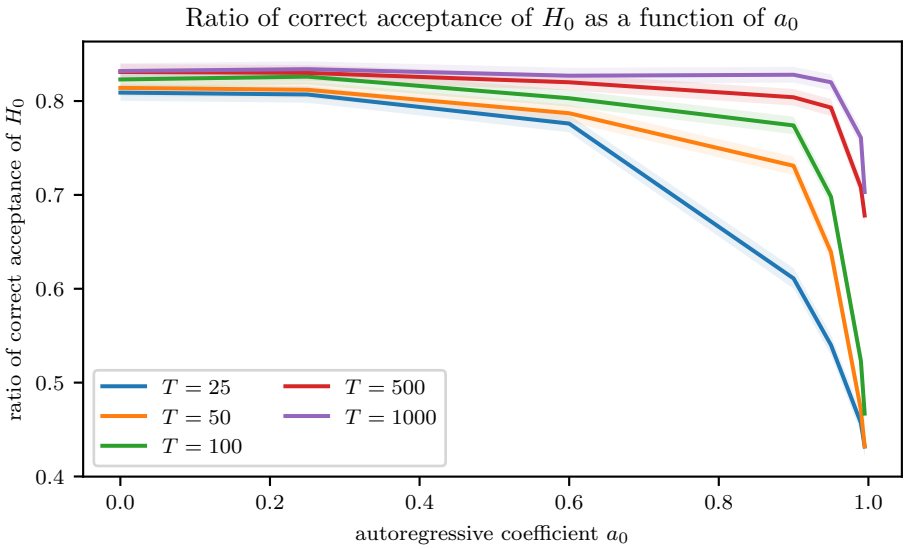


Figure 6.16: Plot of $RCH_0(a_0, b_0, T)$ as a function of the autoregressive coefficient a_0 for varying values of the number of time steps T. Shaded parts correspond to the standard errors.

Just as in Subsection 6.5.1, we see that the ratio of correct acceptance of H_0 is is slightly larger than 0.80 when either a_0 is close to zero or when T is large. However, the ratio of correct acceptance of H_0 decreases quite quickly when a_0 gets closer to zero, especially when T is small. The ratio of correct acceptance of H_0 can also get much smaller than in the previous subsection. As we need to estimate b in both models, the additional parameter in the leave-one-out cross-validation score allows for more flexibility. So, the leave-one-out cross-validation score $-2LL_{\text{LOOCV}}$ is now more often smaller than $-2LL_0$, resulting in a lower value for $RCH_0(a_0, b_0, T)$. When we have more samples, the ratio of correct acceptance of H_0 remains constant around 0.84 for quite some time until $a_0 = 0.9$.

Secondly, let us investigate the ratio of correct H_0 acceptances as a function of T. The corresponding values for $RCH_0(a_0, b_0, T)$ a a function of T have been plotted in Figure 6.17. When we have more time steps, the ratio of correct acceptance of H_0 increases, just as in the scenario where we had no mean. Especially for values of a_0 close to one, we see that the ratio of correct acceptance of H_0 increases when we increase the number of samples. When we have more data, this method is more capable of correctly accepting H_0.

Interestingly, we again see that this ratio does not converge to one, but again some value around 0.84. This is most likely due to the same behavior we have seen in Subsection 6.5.1, where this difference between the negative log-likelihoods, multiplied by two, seems to converge to a chi-squared distribution, shifted two units to the right.

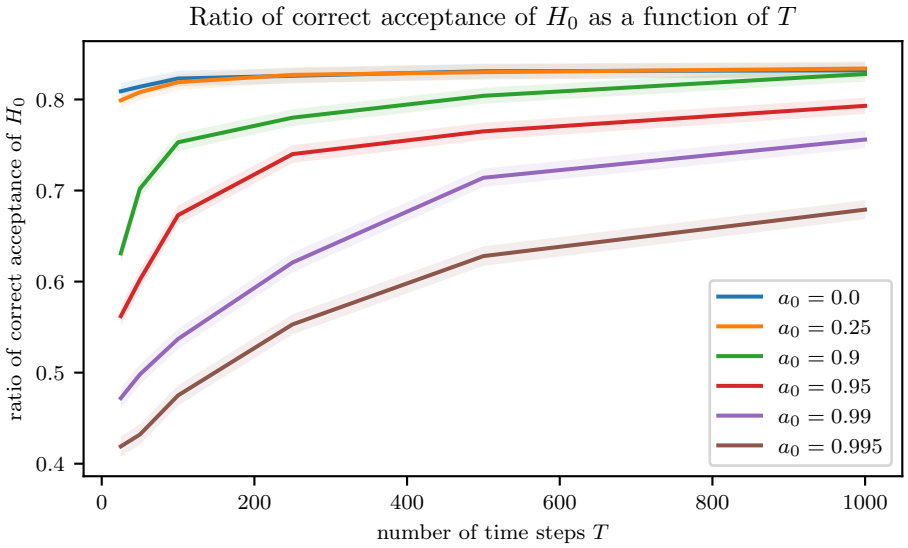


Figure 6.17: Plot of $RCH_0(a_0, b_0, T)$ as a function of the number of time steps T for varying values of the autoregressive coefficient a_0, where $b_0 = 0$. For each (a_0, T) pair, we have conducted a total of $N = 10^4$ simulations. Shaded parts correspond to the standard errors.

Chapter 7

Evaluation

Throughout Chapters 4, 5, and 6, we have presented several methods for learning an acyclic structure that characterizes the linear dependencies between the variables in time series data $\mathbf{X} \in \mathbb{R}^{T \times p}$. We have already highlighted the advantages and disadvantages of most methods in their respective chapters using toy examples, but the methods have not been objectively and quantitatively compared to each other. As no theoretical results have been provided, the methods will be evaluated using both simulated and real-life data.

We will predominantly evaluate our models on time series data, but we will also briefly discuss the performance of our methods in independent data in Section 7.2.

Methods that will be evaluated. We have developed several methods. However, some methods will not be evaluated for several reasons. For example, the exhaustive approach in Section 4.1 is not tractable for more than ten variables. Furthermore, the method where we relaxed the set of permutation matrices to the Birkhoff polytope in Section 5.1 will not be evaluated, as the method did not properly enforce acyclicity and convergence was rather slow.

We will be evaluating the random walk approach as discussed in Section 4.2, where we can only transition to the permutations that are one transposition away, as defined in Equation 4.33 and Equation 4.34. Furthermore, we will be evaluating the regular Metropolis-Hastings approach discussed in Section 4.3, as well as the greedy variant, where we only transition to permutations that achieve a strictly larger likelihood than the current permutation. We will terminate the permutation based approaches after one thousand permutations have been evaluated.

From the continuous-based methods, we will be evaluating the NOTEARS approach modified for VAR(1) models as discussed in Section 5.2, as well as the DAG-LASSO approach as discussed in Section 5.3. Lastly, from the iterative approaches, we will be evaluating the DAG-OMP approach from Section 6.1, and the DAG-OLS-V approach, where we will be using the "violators-first" approach as discussed in Subsection 6.3.1.

This yields a total of seven methods that we will be evaluating throughout this chapter.

Performance criteria. We will evaluate the aforementioned methods based on both structural performance criteria, where we will assess how closely the estimated matrix W resembles to the ground truth W^*, as predictive performance criteria, where we will investigate how well W can be used to predict $\mathbf{X}$.

Structural performance criteria are widely used to assess the performance of methods in the structure learning community, for example in [20, 26, 80]. Such structural performance criteria give insights into how similar the structures of W and W^* are, without considering the data matrix $\mathbf{X}$. We consider each entry in W^* to be equally important. Whether the coefficient in W^* is equal to 0.01 or 1.00, from a structural point of view, failing to recover any of these coefficients is considered equally problematic.

From a predictive point of view, it is unimportant how well W resembles W^*. All arcs could be incorrectly recovered, as long as W is useful in prediction for our data matrix $\mathbf{X}$.

Structural Hamming Distance (SHD). The Structural Hamming Distance (SHD) is a performance metric that captures both the number of false positives (an incorrectly recovered arc) and the number of false negatives (and incorrectly missed arc). The SHD has first been used in [77] to compare adjacency matrices of directed graphs. The structural hamming distance between two adjacency matrices W^* and W is defined as the smallest number of arc additions, deletions, and reversals in order to transform the graph $G(W)$ into the graph $G(W^*)$.

Let $\text{supp}(W)$ be the set of all indices corresponding to non-zero indices in W. Furthermore, let $\overline{\text{supp}}(W)$ denote the set of all indices corresponding to zero entries in W. Furthermore, let $\tilde{W}$ represent the coefficient matrix W with the coefficients on the diagonal set to zero. Then, the SHD can be computed as

$$\begin{aligned} \text{SHD}(W^*, W) &= \#\text{arc additions} + \#\text{arc deletions} + \#\text{arc reversals} \\ &= |\overline{\text{supp}}(W) \cap \text{supp}(W^*)| + |\text{supp}(W) \cap \overline{\text{supp}}(W^*)| - |\text{supp}(\tilde{W}^T) \cap \text{supp}(W^*)|, \end{aligned} \tag{7.1}$$

An arc reversal is counted as both a required arc addition and a required arc deletion in the two components of Equation 7.1. Therefore, the third component of Equation 7.1 subtracts one mistake for each incorrectly directed off-diagonal arc. To ensure only off-diagonal arcs are counted in the third component, we use $\tilde{W}$ rather than W.

On one hand, the SHD quantifies how many arcs of W^* are contained in W, and on the other hand, how many missing arcs of W^* are also not contained in W. This combination is useful as we often want a trade-off between the number of true positives and the number of true negatives. We can "recover" all arcs with a matrix W that only has nonzero coefficients, and we can "recover" all true negatives with a matrix W that does not have any nonzero coefficients. To get an optimal SHD, however, we require W to contain exactly all arcs of W^* and no more. As the SHD considers an incorrect arc direction as only one mistake, we see that the SHD is also more lenient with respect to arc discovery, as a metric such as the accuracy regards this as two mistakes.

Empirical risk Empirical risk is a performance criteria that quantifies the usefulness of W for prediction. For our VAR(1) model, we are interested in assessing how close $X_{t,\cdot}$ is to our prediction $X_{t-1,\cdot}W$. The empirical risk of a coefficient matrix W on a data matrix $\mathbf{X} \in \mathbb{R}^{T \times p}$ is defined as

$$R_{\text{emp}}(W) = \frac{1}{T-1} \sum_{t=2}^{T} \|X_{t,\cdot} - X_{t-1,\cdot}W\|_2^2 \tag{7.2}$$

This is in fact equivalent to the mean squared error, as was defined in Equation 4.28.

A lower empirical risk of W indicates that it was more likely that $\mathbf{X}$ has been generated by a VAR(1) model characterized by W, as was also explained in the introduction of Chapter 4.

True Risk. Although the empirical risk adequately assesses how suitable a coefficient matrix W predicts $\mathbf{X}$, we also want to assess whether W achieve a low empirical risk on similar data $\mathbf{X}'$.

Therefore, we also consider the *true risk*. The true risk can only be evaluated when the data generating model is available. When we generate data according to a VAR(1) model,

$$X_{t,\cdot} = X_{t-1,\cdot}W^* + \varepsilon_t. \tag{7.3}$$

Assuming the generated time series is stationary, that is, $\mathbb{E}[X_{t,\cdot}] = \mathbb{E}[X_{t',\cdot}]$ and $\mathbb{V}(X_{t,\cdot}) = \mathbb{V}(X_{t',\cdot})$ for all t, t', then the true risk of W is calculated by taking the expectation of a single prediction,

$$\begin{aligned} R(W) &= \mathbb{E}\left[\|X_{t,\cdot} - X_{t-1,\cdot}W\|_2^2\right] \\ &= \text{Tr}\left((W^* - W)^T \mathbb{V}(X_{t,\cdot})(W^* - W) + \mathbb{V}(\varepsilon_t)\right), \end{aligned} \tag{7.4}$$

where we can compute the covariance of $\mathbb{V}(X_{t,\cdot})$ as

$$\text{vec}\left(\mathbb{V}(X_{t,\cdot})\right) = \left(I_{p^2} - (W^* \otimes W^*)^T\right)^{-1} \text{vec}\left(\mathbb{V}(\varepsilon_t)\right). \tag{7.5}$$

Note that the true risk is lower bounded by $\text{Tr}(\mathbb{V}(\varepsilon_t))$, which will be equal to p throughout all simulations.

7.1 Time Series Experiments

Having defined the necessary performance criteria to evaluate our methods, we propose the following three types of time series experiments. First, we will simulate an optimal setting in Subsection 7.1.1 where $\mathbf{X}$ has been generated by a VAR(1) model with an *acyclic* coefficient matrix W^*. This is the exact setting we are assuming, and therefore we expect our methods to perform well.

Secondly, we will simulate a sub-optimal setting where the data $\mathbf{X}$ has been generated by a VAR(1) model, but the coefficient matrix W^* is *cyclic*. Therefore, we cannot expect to find an acyclic coefficient matrix W that exactly resembles W^*. Nevertheless, we hope our methods will retrieve a reasonably suitable coefficient matrix W. The results of these experiments will be discussed in Subsection 7.1.2.

Lastly, we will verify our methods on *real-life* data in Subsection 7.1.3. In real-life settings, the VAR(1) modeling assumption might be violated. However, applying our methods on real-life data could provide an interesting use case that highlights the directed relations between the variables.

Generating W^* and $\mathbf{X}$. To generate the true coefficient matrix W^*, we first specify the number of variables p. For the time series experiments, the p auto regressive coefficients on the diagonal will be set to 0.5. This value is rather low, but necessary to ensure stationarity when there are numerous off-diagonal entries, especially for large values of p. Lastly, a total of s off-diagonal arcs will be set to 0.5 such that W^* corresponds to an acyclic structure in Subsection 7.1.1, and to a cyclic structure in Subsection 7.1.2.

Given our coefficient matrix W^*, we can generate our data matrices $\mathbf{X} \in \mathbb{R}^{T \times p}$ by generating T samples according to a VAR(1) model as defined in Definition 2.3. The noise variables are all Gaussian random variables with mean zero and identity covariance. To ensure stationarity, $X_{1,\cdot}$ will have a covariance corresponding to its stationary distribution.

Experimental Setups. There are many parameters with respect to the data generating process that we can consider. As we are predominantly interested in high-dimensional time series, we will carefully investigate the influence of p in the small sample setting and the large sample setting. Additionally, we can change the sparsity of the coefficient matrix W by altering s, the number of off-diagonal arcs. We will consider the following range of parameters.

- The number of variables $p \in \{5, 10, 15, 25, 50\}$, ranging from low to high-dimensional.
- The number of time steps $T \in \{100, 1000\}$, corresponding to few samples and many samples.
- The number of arcs $s \in \{3p, 5p\}$, corresponding to an average of three outgoing arcs per variable (sparse setting) and five outgoing arcs per variable (dense setting), respectively. The number of arcs will be thresholded to $p(p-1)/2$ as that is the maximum number of off-diagonal arcs in a directed acyclic graph.

For each triple (p, T, s), we generate ten datasets to obtain reliable estimates. We will compute the mean and corresponding standard errors of our three performance metric to express uncertainty.

For all methods, we will compute the empirical risk $R_{\text{emp}}(W)$ as defined in Equation 7.2. Subsequently, we will threshold all coefficients in W with an absolute value smaller than $\epsilon = 0.30$ to zero in order to obtain a suitable number of arcs. Based on this thresholded matrix, we will compute SHD according to Equation 7.1. Lastly, we re-estimate the non-zero coefficients of the thresholded matrix to compute the true risk $R(W)$ as defined in Equation 7.4.

Justification for the threshold ϵ Although thresholding is often employed to select a suitable number of arcs [82, 87, 89], some justification is required for the seemingly arbitrary choice for the threshold value $\epsilon = 0.30$. A fair argument is that this threshold has been selected such that it sits nicely between the 0.0 and 0.5, the values of the coefficients in W^*. Therefore, using such a threshold may sketch an overly optimistic picture of the methods. Furthermore, the size of a coefficient is not necessarily a suitable indicator of its performance, so there is reasoning to not do thresholding at all.

However, our reasoning is that we wanted to create a level playing field to assess the structure recovery of all methods, and applying the same threshold to all methods seemed a fair approach. Apart from DAG-LASSO, no other methods shrink their coefficients. Nevertheless, we have investigated the influence of the threshold value ε in Appendix B.5. We have analyzed how the results change for the values $\epsilon \in \{0.0, 0.05, 0.15, 0.3, 0.4\}$. We saw that, although the threshold indeed significantly affects the performance of all methods, the relative performance between the methods remained unchanged. Therefore, the choice of ϵ might influence the absolute results, but the relative performance results of all methods remains valid.

7.1.1 Simulated VAR(1) data with an acyclic coefficient matrix W^*

Let us consider the setting where we have a sparse coefficient matrix, meaning that each variable has an average of three incoming arcs. For the results on dense acyclic coefficient matrices, we refer the interested reader to Section B.2 in the appendix.

The results for the true risk are given in Table 7.1 and have also been plotted with standard errors in Figure 7.1 and Figure 7.2. The results for the Structural Hamming Distance are given in Table 7.2, Figure 7.3, and Figure 7.4. We refer the reader to Appendix B.1 for the empirical risk.

Table 7.1: Average true risk $R(W)$ as a function of p for $T = 100$ and $T = 1000$, where $s = 3p$ and W corresponds to an acyclic structure. A lower true risk indicates a better predictive performance.

	$T = 100$					$T = 1000$				
Method	$p = 5$	$p = 10$	$p = 15$	$p = 25$	$p = 50$	$p = 5$	$p = 10$	$p = 15$	$p = 25$	$p = 50$
Random Walk	**5.26**	12.70	20.85	41.30	109.70	**5.02**	11.92	19.41	37.87	89.57
MH-Regular	**5.26**	12.47	20.85	41.28	109.54	**5.02**	11.87	20.35	37.91	95.08
MH-Greedy	5.38	11.41	17.02	28.95	68.01	**5.02**	10.46	15.47	25.67	54.62
NOTEARS	**5.26**	10.59	**16.08**	**27.08**	**61.03**	**5.02**	**10.04**	15.07	**25.10**	**50.19**
DAG-LASSO	12.32	51.51	68.16	148.08	353.65	10.72	46.02	69.44	136.20	290.74
DAG-OMP	**5.26**	11.45	17.51	27.88	63.64	**5.02**	10.52	15.85	25.34	50.62
DAG-OLS-V	5.28	**10.55**	16.15	27.61	128.07	**5.02**	**10.04**	**15.06**	**25.10**	**50.19**

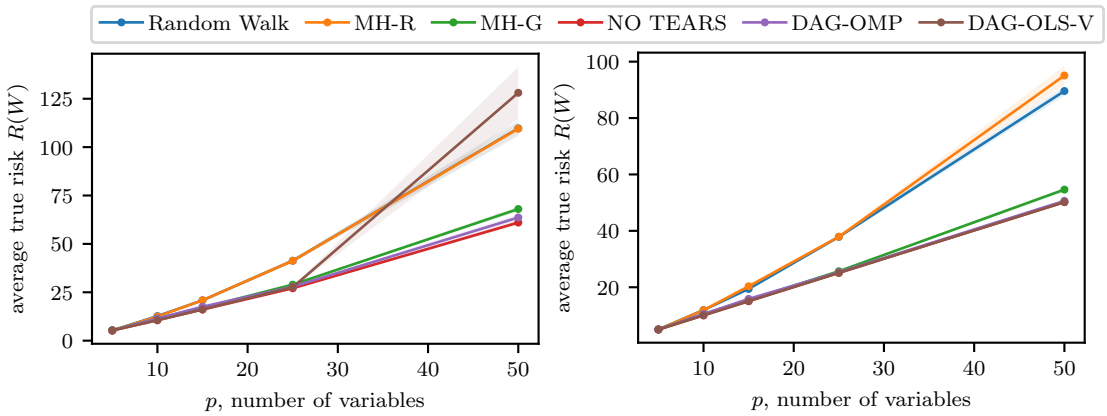


Figure 7.1: Plot of the average true risk as a function of p, where $T = 100$ for the methods in Table 7.1, excluding DAG-LASSO.

Figure 7.2: Plot of the average true risk as a function of p, where $T = 1000$ for the methods in Table 7.1, excluding DAG-LASSO.

DAG-LASSO achieves by far the poorest true risk. As only DAG-LASSO shrinks its coefficients, thresholding will unfairly penalize this method. Although this is true, we have conducted these experiments for several threshold values in Appendix B.5. Even with the most favorable threshold, DAG-LASSO was not on par with the other methods. Its global approach removes too many coefficients when one false coefficient is important due to spurious correlations. As the results of DAG-LASSO skew the plots, we do include them results in the figures.

The random walk and the regular Metropolis-Hastings approach achieve a decent true risk for $p \in \{5, 10, 15\}$, after which the performance decreases. This is most likely due to the exponential increase of the search space. Trying only 1000 permutations is enough to try all permutations for $p = 5$ and a reasonable subset for $p = 10$ and $p = 15$. However, for $p = 25$, we only cover a minuscule portion of $25!/1000 \approx 10^{-20}\%$ of the search space, so these permutation-based approaches will decrease in performance as p increases. The regular Metropolis-Hastings approach is not much better than the random walk, most likely because the coefficient matrix is relatively sparse, as discussed in Example 4.6. On the other hand, the greedy Metropolis-Hastings approach performs surprisingly well, even though only a minuscule fraction of the search space has been explored. The exploitative decision rule seems to allow for an efficient traversal of the search space.

NOTEARS, DAG-OMP, and DAG-OLS perform quite well. Note that the optimal coefficient matrix will still yield a true risk of p due to the noise, indicating that NOTEARS and DAG-OMP perform close to optimal. DAG-OLS-V, quite surprisingly, is on par with NOTEARS and DAG-OMP, yet performs poorly for $p = 50, T = 100$. We think that the initial OLS estimate is too inaccurate in such a high-dimensional small-sample setting. Furthermore, all methods benefit from having a larger sample size T, although the permutation-based approaches seem to benefit relatively less. The bottleneck for permutation-based approaches was not the sample size, but rather the iteration limit.

Table 7.2: Average structural hamming distance (SHD) as a function of p for $T = 100$ and $T = 1000$, where $s = 3p$ and W is acyclic. A lower SHD indicates a better structural performance.

	$T = 100$					$T = 1000$				
Method	$p = 5$	$p = 10$	$p = 15$	$p = 25$	$p = 50$	$p = 5$	$p = 10$	$p = 15$	$p = 25$	$p = 50$
Random Walk	**0.3**	5.4	11.2	29.5	97.5	**0.0**	4.5	9.1	22.6	57.7
MH-Regular	**0.3**	5.2	11.2	30.6	99.6	**0.0**	4.3	10.8	20.8	58.4
MH-Greedy	0.7	2.3	2.4	6.1	33.2	**0.0**	1.1	0.8	1.2	6.0
NOTEARS	**0.3**	0.4	**0.7**	**1.9**	**20.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**
DAG-LASSO	12.1	37.2	54.5	92.8	188.1	9.1	36.1	55.0	91.9	184.6
DAG-OMP	**0.3**	1.6	1.4	3.6	22.3	**0.0**	0.6	0.9	0.4	0.3
DAG-OLS-V	0.4	**0.3**	0.8	2.9	58.1	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**

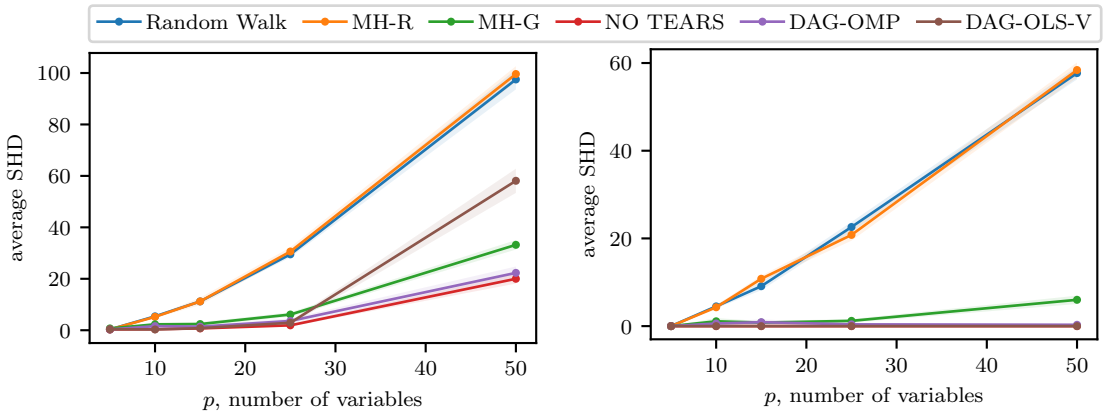


Figure 7.3: Plot of the structural hamming distance as a function of p, where $T = 100$ for the methods in Table 7.2, excluding DAG-LASSO.

Figure 7.4: Plot of the structural hamming distance as a function of p, where $T = 1000$ for the methods in Table 7.2, excluding DAG-LASSO.

Considering the structural hamming distance, we see comparable results as with the true risk. All methods improve as we have more samples, most notably the DAG-OLS-V approach as it relies on a suitable initial ordinary least squares estimate. The sharp increase in SHD from $p = 25$ to $p = 50$ in the small-sample setting highlights the increase in difficulty of recovering a suitable structure increases when the number of dimensions increases.

The random walk and the regular Metropolis-Hastings approach perform quite poor compared to the other methods. The greedy Metropolis-Hastings approach performs quite well, but NOTEARS and DAG-OMP perform slightly better, especially for larger values of p. DAG-OLS-V again performs surprisingly well apart from the setting where $T = 100$ and $p = 50$. In the large-sample setting, NOTEARS and DAG-OLS-V perfectly recover the acyclic structure.

Using cross-validation to determine a suitable number of arcs

To create a level playing field for the permutation-based methods, we have used thresholding to select a suitable number of arcs. This thresholding is rather unjustified, as it might benefit some methods over others. Furthermore, the threshold was chosen such that it sits nicely in between zero and the value of the true coefficients, which is information we do not have in practice.

Therefore, we will also provide the true risk and structural hamming distance where we have not applied any thresholding. We apply cross-validation for the iterative methods and the regularization parameter $\lambda = 0.1$ for NOTEARS, which the authors proposed. The results are shown in Table 7.3 and Table 7.4. We obtain an ordering of the arcs for DAG-OLS-V by repeating its deletion criterion until the graph is empty. We could also apply such a deletion criterion to the permutation-based approaches, after which we could use cross-validation as well. However, we consider the ordering an advantage of the iterative approach, and we consider "lending" this feature to the permutation-based approaches unfair. Furthermore, this would have been computationally infeasible for $p \geq 15$, as the method needs to be applied T times rather than once.

Table 7.3: Average true risk using cross-validation for the iterative approaches and $\lambda = 0.1$ for NOTEARS as a function of p for $T = 100$ and $T = 1000$, where $s = 3p$ and W^* corresponds to an acyclic structure. A lower true risk indicates a better predictive performance.

	$T = 100$					$T = 1000$				
Method	$p = 5$	$p = 10$	$p = 15$	$p = 25$	$p = 50$	$p = 5$	$p = 10$	$p = 15$	$p = 25$	$p = 50$
Random Walk	**5.17**	12.69	20.33	42.26	130.67	**5.02**	11.99	19.15	35.93	80.92
MH-Regular	**5.17**	12.87	20.41	43.98	136.06	**5.02**	11.80	19.73	36.46	84.11
MH-Greedy	**5.17**	11.38	16.98	32.18	96.01	**5.02**	10.29	15.18	26.10	55.39
NOTEARS	**5.17**	**10.63**	16.53	29.52	69.33	**5.02**	**10.04**	**15.08**	**25.15**	**50.36**
DAG-LASSO	8.47	37.04	50.79	101.90	244.59	8.24	29.30	49.07	87.45	205.30
DAG-OMP	**5.17**	11.26	16.92	**27.54**	**58.9**	**5.02**	10.38	15.73	25.36	50.45
DAG-OLS-V	**5.17**	10.64	**16.39**	28.65	63.50	**5.02**	10.06	15.13	25.27	50.93

Table 7.4: Average structural hamming distance using cross-validation for the iterative approaches and $\lambda = 0.1$ for NOTEARS as a function of p for $T = 100$ and $T = 1000$, where $s = 3p$ and W^* is acyclic. A lower SHD indicates a better structural performance.

	$T = 100$					$T = 1000$				
Method	$p = 5$	$p = 10$	$p = 15$	$p = 25$	$p = 50$	$p = 5$	$p = 10$	$p = 15$	$p = 25$	$p = 50$
Random Walk	**0.0**	23.8	74.8	263.2	1177.2	**0.0**	23.6	75.8	267.0	1176.0
MH-Regular	**0.0**	23.8	76.2	266.0	1182.6	**0.0**	22.2	78.8	267.4	1189.0
MH-Greedy	**0.0**	18.2	61.0	229.4	1095.2	**0.0**	16.2	60.2	227.6	1089.4
NOTEARS	**0.0**	7.3	26.0	91.0	370.3	**0.0**	**0.7**	**4.9**	**11.2**	34.6
DAG-LASSO	8.7	37.3	58.2	96.0	205.3	6.8	35.9	57.0	94.7	195.3
DAG-OMP	**0.0**	**4.3**	**8.6**	**20.6**	**44.9**	**0.0**	3.5	9.1	12.7	**17.5**
DAG-OLS-V	**0.0**	7.4	18.1	56.3	166.9	**0.0**	5.8	20.9	60.0	229.0

Cross-validation for iterative methods and using $\lambda = 0.10$ for NOTEARS yield poorer structural performance than thresholding, as we do not have the oracle information of a suitable threshold. However, these regularization approaches yield only a minor decrease in predictive performance compared to thresholding. For the permutation-based approaches, the SHD increases drastically if we do not apply thresholding, and the true risk increases slightly.

In the small sample setting, cross-validation seems preferred over NOTEARS with $\lambda = 0.10$. DAG-OMP performs slightly better than DAG-OLS-V, most likely due to its more sensible selection criterion. In the large sample setting, NOTEARS with default λ seems preferred over cross-validation in terms of structure, and just slightly better in terms of prediction. In both settings, the regularized approaches yield much better results than permutation-based methods. Lastly, DAG-LASSO without thresholding still performs poorest.

7.1.2 Simulated VAR(1) data with a cyclic coefficient matrix W^*

Let us consider the setting where the coefficient matrix W^* is cyclic. Therefore, we can never achieve a structural hamming distance of zero, or a true risk as low as when W^* was acyclic. Nevertheless, it is interesting to see which methods cope best with acyclicity.

The results for the true risk $R(W)$ are given in Table 7.5, Figure 7.5, and Figure 7.6. The results for the Structural Hamming Distance, are given in Table 7.6, Figure 7.7 and Figure 7.8. We refer the reader to Appendix B.3 for the empirical risk.

Table 7.5: Average true risk $R(W)$ as a function of p for $T = 100$ and $T = 1000$, where $s = 3p$ and W^* corresponds to a cyclic structure. A lower true risk indicates a better predictive performance.

	$T = 100$					$T = 1000$				
Method	$p = 5$	$p = 10$	$p = 15$	$p = 25$	$p = 50$	$p = 5$	$p = 10$	$p = 15$	$p = 25$	$p = 50$
Random Walk	**6.79**	13.88	21.32	40.23	95.62	**6.61**	13.10	19.97	36.75	77.33
MH-Regular	**6.79**	**13.80**	20.64	41.22	93.31	**6.61**	13.29	20.12	37.06	79.32
MH-Greedy	7.01	13.85	19.94	35.72	81.01	6.75	13.02	18.39	**31.50**	63.20
NOTEARS	6.83	13.47	**19.57**	**34.58**	74.41	6.66	**12.76**	**18.27**	32.87	**60.35**
DAG-LASSO	20.03	42.85	67.68	129.41	334.85	19.63	42.49	64.84	127.45	268.56
DAG-OMP	7.38	14.35	20.44	35.86	77.72	7.08	13.35	18.65	31.78	60.93
DAG-OLS-V	7.89	14.82	21.21	36.25	**58.18**	7.49	14.02	19.39	33.99	63.09

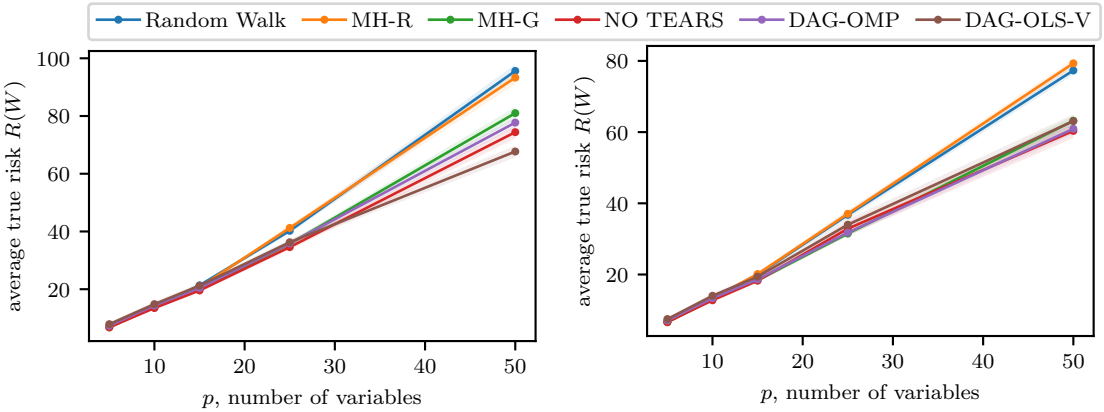


Figure 7.5: Plot of the average true risk as a function of p where $T = 100$ for the methods in Table 7.5, excluding DAG-LASSO.

Figure 7.6: Plot of the average true risk as a function of p where $T = 1000$ for the methods in Table 7.5, excluding DAG-LASSO.

The predictive performance of the six methods seem to be closer than in the acyclic setting, especially for $p \in \{5, 10, 15\}$, with DAG-LASSO being the exception as expected. As the number of variables increases, the permutation-based approaches fall slightly behind, as the number of permutations grows exponentially. For $T = 1000$, the non permutation-based approaches all achieve a similar true risk, which is most likely close to optimal. However, the permutation-based approaches who are not able to efficiently travel the search space seem to benefit very little of this larger sample size. Arguably the simplest method, the DAG-OLS-V algorithm, performs surprisingly well in the $T = 100$ setting, outperforming a state-of-the-art method such as NOTEARS when $p = 50$.

Table 7.6: Average structural hamming distance (SHD) as a function of p for $T = 100$ and $T = 1000$, where $s = 3p$ and W^* is cyclic. A lower SHD indicates a better structural performance.

	$T = 100$					$T = 1000$				
Method	$p = 5$	$p = 10$	$p = 15$	$p = 25$	$p = 50$	$p = 5$	$p = 10$	$p = 15$	$p = 25$	$p = 50$
Random Walk	**3.5**	7.1	12.7	25.0	73.9	**3.3**	5.5	9.7	18.0	37.2
MH-Regular	**3.5**	6.7	11.5	26.7	78.4	**3.3**	5.0	9.9	17.7	40.1
MH-Greedy	3.7	5.8	9.6	15.7	52.7	3.8	5.6	6.1	**8.2**	15.6
NOTEARS	3.7	**5.7**	**8.2**	**13.0**	42.1	**3.3**	**4.9**	**5.8**	9.2	**11.3**
DAG-LASSO	14.3	28.2	41.8	69.7	140.3	14.2	27.9	42.0	69.4	134.9
DAG-OMP	4.6	8.1	10.4	16.2	49.8	4.3	6.3	7.0	10.2	13.8
DAG-OLS-V	4.5	6.8	10.0	14.7	**23.5**	4.6	6.5	7.4	11.8	17.4

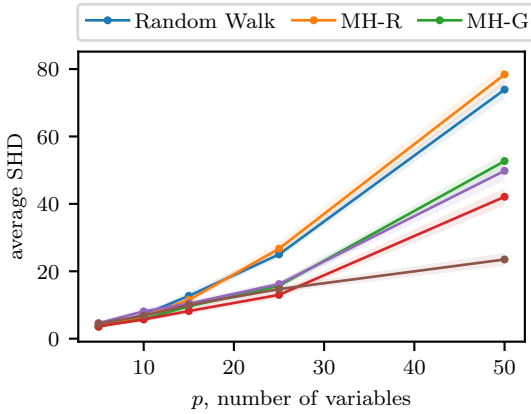


Figure 7.7: Plot of the structural hamming distance as a function of p where $T = 100$ for the methods in Table 7.6, excluding DAG-LASSO.

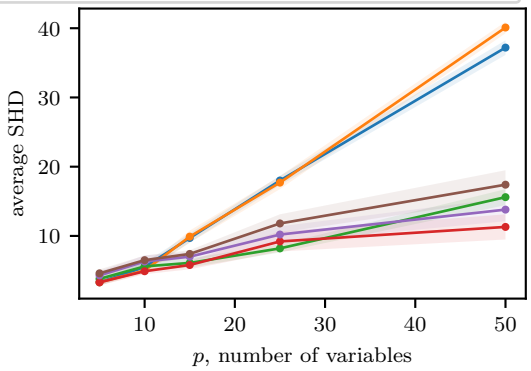


Figure 7.8: Plot of the structural hamming distance as a function of p where $T = 1000$ for the methods in Table 7.6, excluding DAG-LASSO.

From a structural perspective, the random walk and the regular Metropolis-Hastings achieve quite a large structural hamming distance, indicating that their recovered coefficient matrix deviates quite a lot from the true coefficient matrix. For small sample sizes, the DAG-OLS-V approach seems to recover the structure of W^* best, as NOTEARS, DAG-OMP, and the greedy Metropolis-Hastings approach seem to struggle more when $T = 100$ and $p = 50$.

For larger sample sizes, the structural performance increases for all methods, with NOTEARS being slightly better than DAG-OMP, the greedy Metropolis-Hastings approach, and DAG-OLS-V.

7.1.3 Real Life Time Series Data.

In the previous two subsections, we have generated data according to a VAR(1) model, where all noise components were all independently and identically distributed with mean zero and an identity covariance matrix. This setting is quite optimistic, as it perfectly aligns with our model assumptions, apart from the acyclicity assumption in Subsection 7.1.2.

However, more often than not, real-life data does not perfectly align with the model assumptions. Hopefully, these violations are not too problematic and we can still obtain interesting results. Furthermore, these real-life datasets allow us to give meaning to the directed relationships. Rather than $X_1 \to X_2$, these variables have a physical meaning, such as how the sales of one product affect the sales of another product. Therefore, let us consider the Dominick's Finer Foods data.

Dominick's Finer Foods data. Dominick's Finer Foods was a well-established supermarket chain in Chicago which was declared defunct in 2013 [39, 53]. Information regarding the prices, sales and promotions of all their available products was gathered from 1989 until 1999, as part of a partnership with Chicago Booth, the graduate business school of the University of Chicago. For each Dominick's Finer Foods store, the weekly number of units sold has been recorded. Many more attributes have been recorded, such as the demographics of customers, prices of the products, profit margin, deal codes, etc. Nevertheless, we will only focus on the sales of the products.

Over those seven years, more than 3,500 different Universal Product Codes (UPCs) have been tracked, corresponding to specific products sold by Dominick's Finer Foods. These UPCs have been categorized into 29 different categories, ranging from foods such as "cheese" and "crackers", beverages such as "beer" and "bottled juice", and general commodities such as "dish detergent" and "toothbrushes". We have decided to focus on the sixteen consumable categories, which correspond to: "beer", "bottled juice", "canned soup", "canned tuna", "cereal", "cheese", "cookies", "cracker", "front end candy", "frozen dinner", "frozen entree", "frozen juice", "oatmeal", "refrigerated juice", "snack cracker", and "soft drink".

We follow the same approach as [27] and [70] by only considering data from January 1993 until July 1994, yielding 77 weeks of sales numbers for sixteen categories in total. Next the log-differences of the sales have been considered rather than the actual sales to ensure stationarity of the time series. This results in sixteen time series of 76 time steps for each Dominick's Finer Foods store. Now, we are interested in seeing how the *sales* of one category influence the *sales* of another category. We conjecture that similar product categories, such as "bottled juice" and "refrigerated juice" will have some relation, as they are similar categories. We argue that when quite a lot of bottled juice is sold in one week, then this might be at the expensive of refrigerated drinks the next week. Similarly, if the sales of cheese increase this week, then this be associated with an increase of other snacks and unhealthy beverages the next week. Furthermore, we expect no relation to exist between dissimilar categories, such as "canned tuna" and "beer". If customers purchase more beer, we do not expect more canned tuna to be sold in the next week.

We use `DAG-OMP` from Section 6.1 to estimate a dense acyclic structure, after which we use leave-one-out cross-validation to determine a suitable number of arcs. This yields the sparse coefficient matrix W in Figure 7.9, containing nine off-diagonal arcs. We have only drawn variables that had at least one incoming or outgoing arc as to clutter the structure as little as possible.
We see some interesting relations between the product categories. First of all, we see that the sales of cheese seem to affect the future sales of beverages such as soft drinks and beer, as well as frozen dinner. This is in line with the conjecture that snacks such as cheese are frequently bought either together or in quick succession of other unhealthy consumables such as frozen pizzas, beer, or soft drinks.

Furthermore, we also see a directed relationship from refrigerated juice to bottled juice, and from snack cracker to oatmeal. The former could be explained that when customer purchase more refrigerated juice, they see no need to purchase bottled juice in the near future. For the latter directed relationship, no reasonable explanation could be deduced.

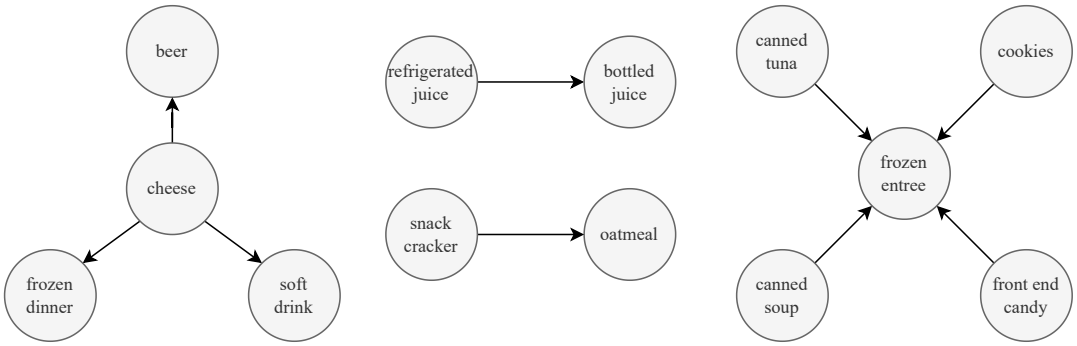


Figure 7.9: Acyclic structure corresponding to the Dominick's Finer Foods data. The structure has been inferred using `DAG-OMP`, and the number of arcs was chosen using leave-one-out cross-validation. Variables with no incoming or outgoing arcs have been omitted.

Lastly, the future sales of frozen entrees is affected by canned tuna, canned soup, cookies, and front end candy. It seems that when customers purchase canned food or sweet snacks such as cookies and candy, then this will affect how much frozen entrees are purchased in the near future.

We would like to stress that all these assumptions should be taken with a grain of salt, as it is difficult to verify whether our findings sensible. Furthermore, the time granularity of one week makes these relations more difficult to interpret. Nevertheless, it provides an interesting use case of how the sales of consumables affect the future sales of other consumables.

7.2 Time-Independent Experiments

So far, all methods and concepts introduced assumed the time-series setting of a VAR(1) model. However, recall that most structure learning methodologies assume *instantaneous* relations, for example through a linear structural equation model as defined in Definition 2.2. Although all methods and examples have been developed for time series data, we will use this section to briefly investigate the performance of our discussed methods on independent data. We will first evaluate the methods based on simulated data in Subsection 7.2.1, after which we will also evaluate our methods on real-life biological data in Subsection 7.2.2.

Modifications to the methods. Luckily, our methods need to be adjusted only slightly. For the permutation-based and iterative approaches, we only need to align the index of the response and explanatory variable, rather than shifting one time index. Furthermore, we must fix the diagonal entries of W to zero, as we cannot use the value of a variable to predict itself. For the continuous-based methods, NOTEARS now becomes the regular method as the authors have proposed in [87]. The DAG-LASSO algorithm, unfortunately, was not suitable for these models as it shrinks all coefficients until no cycle remains.

We also need to modify the predictive performance criteria. The empirical risk $R_{\text{emp}}(W)$ is now computed as

$$R_{\text{emp}}(W) = \frac{1}{T}\sum_{t=1}^{T} \|X_{t,\cdot} - X_{t,\cdot}W\|_2^2\,, \tag{7.6}$$

and the true risk $R(W)$ is calculated just as in Equation 7.4, but now the covariance of $X_{t,\cdot}$ is

$$\mathbb{V}(X_{t,\cdot}) = \left((I_p - W^*)^{-1}\right)^T \left(I_p - W^*\right)^{-1}. \tag{7.7}$$

7.2.1 Simulated Time-Independent Data

Generating W^* and X. For the linear structural equation model, we will generate an acyclic coefficient matrix W^* by setting $s = \min\{3p, p(p-1)/2\}$ coefficients to non-zero, thereby enforcing that the structure remains acyclic and the diagonal coefficients remain zero. The values of the s non-zero coefficients will be sampled uniformly from the range $(-2.0, -0.5) \cup (0.5, 2.0)$.

Given this coefficient matrix W^*, we generate T independent p-dimensional vectors X according to a linear structural equation model (SEM)

$$X = XW^* + \varepsilon, \tag{7.8}$$

where ε is an independent Gaussian random variable with mean zero and as covariance matrix the identity matrix. Additionally, we first need to sample the variables with no incoming arcs, and continue only sampling variables when all its parents have been sampled first, a so-called *ancestor-first* sampling. Generating T of these independent and identically distributed variables $X_{t,\cdot}$, $t = 1, \ldots, T$ yields the data matrix $\mathbf{X} \in \mathbb{R}^{T \times p}$.

The results for the true risk $R(W)$ are given in Table 7.7 and have been plotted in Figure 7.10 and Figure 7.11. The results for the Structural Hamming Distance are given in Table 7.8, Figure 7.12 and Figure 7.13. We refer the reader to Appendix B.4 for the empirical risk.

Table 7.7: Average true risk $R(W)$ as a function of p for $T = 100$ and $T = 1000$, where $s = 3p$ and the data has been generated according to a linear structural equation model. A lower true risk indicates a better predictive performance.

	$T = 100$					$T = 1000$				
Method	$p = 5$	$p = 10$	$p = 15$	$p = 25$	$p = 50$	$p = 5$	$p = 10$	$p = 15$	$p = 25$	$p = 50$
Random Walk	**5.13**	13.12	29.26	62.28	335.55	**5.01**	14.08	30.09	54.89	498.40
MH-Regular	**5.13**	12.68	25.57	66.41	290.89	**5.01**	12.13	25.31	58.90	462.91
MH-Greedy	**5.13**	**10.41**	**15.82**	**27.35**	**67.77**	**5.01**	**10.03**	**15.04**	**25.11**	**56.25**
NOTEARS	5.20	10.82	16.36	30.27	152.6	5.06	10.44	22.55	28.38	153.32
DAG-OMP	5.98	15.34	19.71	37.92	77.2	5.95	15.35	19.08	33.74	74.26
DAG-OLS-V	7.04	26.88	63.67	182.5	2154.0	7.11	34.43	98.66	194.27	2468.56

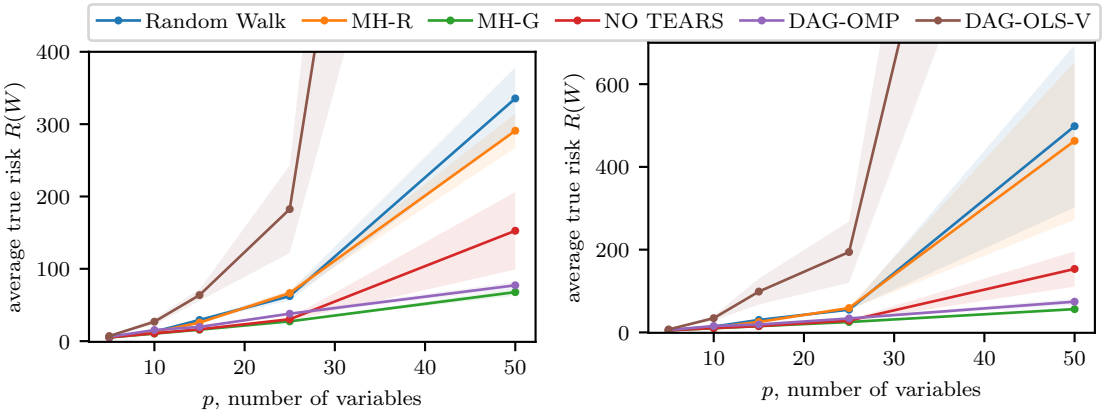


Figure 7.10: Plot of the average true risk as a function of p where $T = 100$ for the methods in Table 7.7.

Figure 7.11: Plot of the average true risk as a function of p where $T = 1000$ for the methods in Table 7.7.

The DAG-OLS-V algorithm achieves the poorest true risk of the six methods, whereas it performed much better on data generated by VAR(1) models. Apparently, its ordinary least squares estimate is not a suitable starting point for linear SEMs. The two explorative permutation-based approaches achieve quite a good performance for $p \in \{5, 10, 15\}$, but this performance decreases as p increases.

The number of permutations scales so fast that such an exploratory approach is not tractable in higher dimensions. Interestingly, the exploitative permutation-based approach seems to be achieving the smallest true risk, significantly smaller than the state-of-the-art method NOTEARS. Furthermore, the DAG-OMP approach seems to be a more suitable approach than NOTEARS when $p = 50$ for both small and large sample sizes, indicating that an iterative approach is relatively more suitable when the number of variables is quite large.

Table 7.8: Average structural hamming distance as a function of p for $T = 100$ and $T = 1000$, where $s = 3p$ and the data has been generated according to a linear structural equation model. A lower SHD indicates a better structural performance.

	$T = 100$					$T = 1000$				
Method	$p = 5$	$p = 10$	$p = 15$	$p = 25$	$p = 50$	$p = 5$	$p = 10$	$p = 15$	$p = 25$	$p = 50$
Random Walk	**0.2**	13.9	43.4	112.0	375.8	**0.0**	12.5	37.9	87.4	261.7
MH-Regular	**0.2**	12.5	40.7	104.9	372.6	**0.0**	10.7	42.3	95.8	260.3
MH-Greedy	**0.2**	**3.1**	**10.4**	**45.7**	270.1	**0.0**	**0.0**	**0.0**	**1.0**	**39.1**
NOTEARS	1.5	6.6	15.3	49.3	220.1	1.2	3.6	6.6	9.9	54.6
DAG-OMP	2.9	15.3	25.8	78.3	272.4	2.6	15.4	25.2	39.0	93.8
DAG-OLS-V	1.9	13.5	30.5	54.7	**138.7**	1.4	15.8	29.5	50.5	125.5

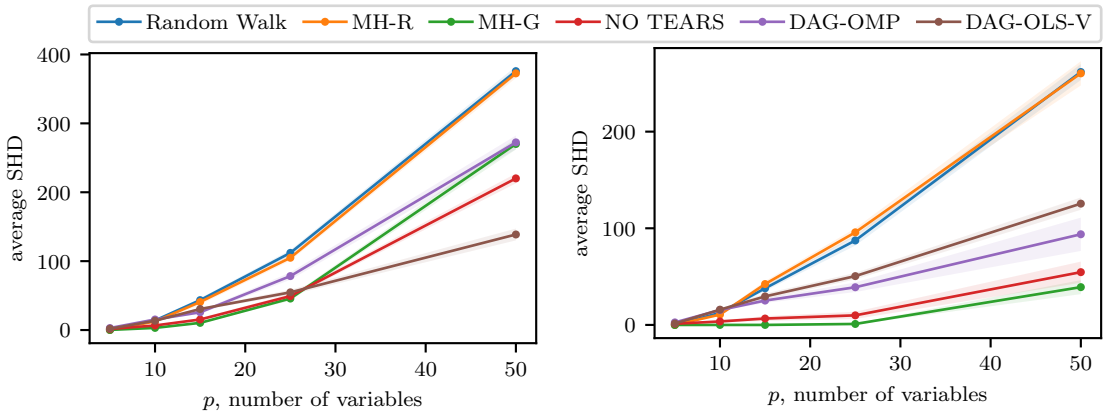


Figure 7.12: Plot of the average structural hamming distance as a function of p where $T = 100$ for the methods in Table 7.8.

Figure 7.13: Plot of the average structural hamming distance as a function of p where $T = 1000$ for the methods in Table 7.8.

DAG-OLS-V now achieves the lowest SHD, which seems to contradict the observation that the method achieved the worst predictive performance. An explanation for this is that DAG-OLS-V estimates a coefficient matrix W that is much too sparse. We can achieve an average SHD of $s = 3p$ simply by returning the zero matrix. Therefore, we need to compare the methods on both predictive and structural performance criteria to ensure that the method is adequate.

The random walk and the regular Metropolis-Hastings algorithm both perform quite poorly. Interestingly, the greedy Metropolis-Hastings approach outperforms the state-of-the-art NOTEARS method also with respect to the structural hamming distance. The DAG-OMP algorithm is slightly worse from a structural perspective, although it performed quite well from a predictive perspective.

7.2.2 Real-Life Time-Independent Data

In this section, we will consider the dataset introduced by Sachs et al. [64]. It contains a total of 7,466 measurements of 11 types of phosphorylated proteins and phospholipids. Sachs et al. used this dataset to learn the causal pathways between these proteins and phospholipids.

This dataset is widely used as a benchmark in the structure learning community because these pathway links are already known from the existing literature. Figure 7.14 depicts a network widely accepted by biologists as a sensible ground truth. However, note that unknown confounders might exist or other factors that have not been discovered yet. Sachs. et al have considered the eleven variables colored in orange. Having such a ground truth widely accepted by biologists, researchers can benchmark their methods against real-life data.

The model reported by Sachs et al is shown in Figure 7.15. They acquired this model using the following procedure. They first removed 30% of the data which they considered to be outliers, then discretized the data and applied a heuristic search to maximize their proposed Bayesian score. Their heuristic searches the space of directed acyclic graphs by randomly modifying (that is, removing, adding, or reversing) an arc without violating the acyclicity assumption. This new directed acyclic graph is accepted if it achieves a better score. To avoid local optima, they occasionally accept a graph that achieves a lower score. Then they employ model averaging by applying this heuristic a total of 500 times, yielding 500 models, and only keep arcs that occur in more than 85% of the models.

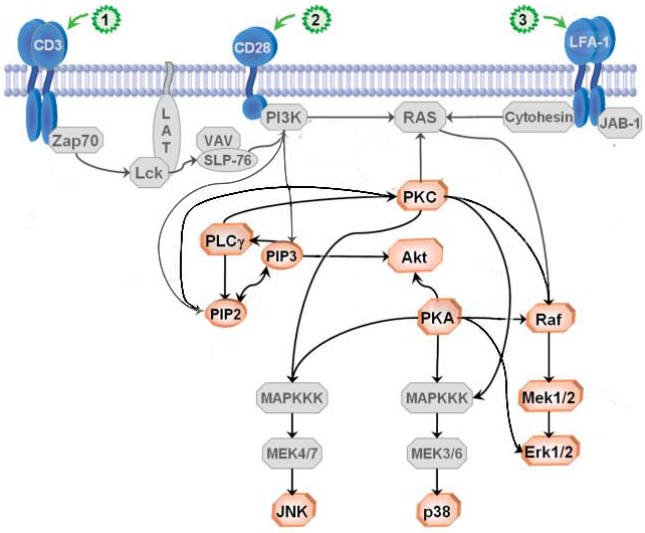


Figure 7.14: Biological overview of the causal pathways widely accepted by biologists. Variables that were included in the dataset are colored in orange. Retrieved from [64].

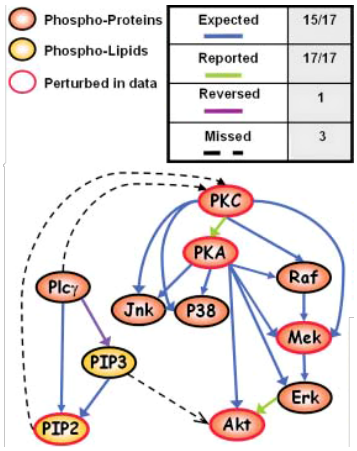


Figure 7.15: The network obtained by Sachs et al. in [64]. Fourteen expected pathways were correctly recovered, one pathway was recovered in the reversed direction, and three pathways were missed. Furthermore, two pathways were reported but were not among the widely accepted pathways.

Some authors use the model reported by Sachs as the ground truth [87, 85], whereas others authors use the original biologists' view as the ground truth [60, 19]. We will be using the same ground truth as NOTEARS, as that is the state-of-the-art method that we will compare our methods to, which corresponds to Figure 7.15. Furthermore, several versions of the dataset exists, such as discretized versions or where outliers have been removed. For clarity, we have selected the original dataset as provided in the supplementary material of their publication.

We have applied our five methods, as well as the state of the art NO TEARS method on this biological dataset. The results are shown in Table 7.9.

Table 7.9: Results of applying our methods as well as NOTEARS on the time-independent protein dataset of Sachs et al. [64]. We have reported the total number of predicted edges, as well as the true positives (TP) out of 20, the structural hamming distance, and the empirical risk. We consider the graph in Figure 7.15 to be the ground truth.

Method	Predicted edges	TP (out of 20)	SHD	$R_{\text{emp}}(W)$
Random Walk	13	6	21	$5.037 \cdot 10^5$
MH-Regular	15	7	21	$5.051 \cdot 10^5$
MH-Greedy	17	**8**	21	$\mathbf{4.998 \cdot 10^5}$
NOTEARS	16	**8**	22	$5.032 \cdot 10^5$
DAG-OMP	17	**8**	21	$5.000 \cdot 10^5$
DAG-OLS-V	14	7	**20**	$5.156 \cdot 10^5$

From Table 7.9, we conclude that all methods seem to achieve similar performance. All methods correctly recover either six, seven, or eight of the causal pathways. Furthermore, the corresponding structural hamming distance is either 20, 21, or 22 for all methods, indicating that their structural performance is similar. Furthermore, all methods achieve a similar empirical risk, with the greedy Metropolis-Hastings approach attaining the lowest empirical risk at $4.998 \cdot 10^5$, and DAG-OLS-V attaining the highest empirical risk at $5.156 \cdot 10^5$.

As the number of variables p is quite low, it is not unexpected that the permutation-based approaches achieve a similar performance to the other methods. Even a naive random walk that has tried 1000 permutations attains a similar performance as a state of the art method such as NOTEARS. Furthermore, as the number of samples is quite large, we expected the iterative approaches to achieve a decent performance.

Chapter 8

Conclusion

In this work, we have discussed the notion of structure learning in time series data. Given a set of time series, the research objective was to learn how the past values of one time series influence the future values of another time series. These directed relationships can be summarized in a graphical model where an arc from node i to j indicates that time series i is useful in predicting future values of time series j. To enhance interpretability and to promote sparsity of the learned graphical model, we have explicitly forbidden the existence of cycles, excluding self-loops.

In Chapter 2, we have formalized the notion of structure learning in time series, and have introduced the VAR(1) model that we have employed to learn an acyclic structure in time series data. Subsequently, we have discussed some interesting state-of-the-art methodologies in Chapter 3 which can learn the structure of time-independent data and how these methodologies can be extended to time series data.

Methodologies. A natural start in Chapter 4 was to consider maximum likelihood estimation. However, such a procedure does not enforce acyclicity. Therefore, we have discussed several permutation-based approaches with an increasing level of complexity. Acyclicity was enforced by first selecting a permutation after which only arcs were permitted that respect the induced permutation. That is, only arcs were permitted from a variable i to another variable j if variable i precedes variable j in the permutation. In Section 4.1, we have investigated an exhaustive approach, where we simply try all possible permutations. As such a method is not tractable for more than ten variables, we have proposed heuristic search algorithms such as a Metropolis-Hastings based searched that do not try all permutations, but rather a subset of all permutations. These methods seem adequate up to a moderate number of variables, based on empirical evidence, but their performance seems to scale poorly with the number of variables.

To deal with such a large number of variables, we considered approaches that relax the original search space to a continuous optimization problem. We have investigated in Section 5.1 relaxing the space of permutation matrices to the space of doubly stochastic matrices, after which we found a local optimum using gradient descent with Lagrange multipliers. Unfortunately, this approach was unsuccessful, as convergence was slow and this method was unable to enforce acyclicity. Alternatively, we have modified the NOTEARS [87] algorithm to learn an acyclic VAR(1) model by using a continuous constraint that enforces acyclicity in Section 5.2. As a last continuous approach, we have used a LASSO-penalty to enforce acyclicity by increasing the penalty parameter until the inferred structure was acyclic in Section 5.3.

Thirdly, we have developed iterative methods where our learned structure is updated one arc at a time rather than all arcs simultaneously. We have first extended the Orthogonal Matching Pursuit algorithm to estimate sparse VAR(1) models, after which we also enforce acyclicity in Section 6.1. A backward ordinary least squares approach was proposed in Section 6.2 and an improvement to backward approaches was discussed in Section 6.3. Additionally, we have developed several approaches to determine a suitable number of arcs in our structure. Too few arcs may cause a significant drop in predictive performance and, on the other hand, too many arcs may

result in an overly complex structure. Therefore, we have developed two approaches that rely on bootstrapping and one approach that relies on cross-validation to learn a suitable number of arcs in Section 6.4. Lastly, as cross-validation was surprisingly effective for model selection despite having dependent data, we have conducted a small theoretical analysis regarding leave-one-out cross-validation on autoregressive models in Section 6.5. There, we have closely investigated a simpler setting to understand why leave-one-out cross-validation, despite being morally wrong in our time series setting, might still be quite useful for model selection. Our investigations suggest an interesting relation between Wilk's theorem and the leave-one-out cross-validation score.

Evaluations. The aforementioned methods have been evaluated and their predictive and structural performance have been compared on simulated data. For the time series setting, we have simulated acyclic VAR(1) models and cyclic VAR(1) models. All methods except for DAG-LASSO performed well, even when the generated VAR(1) model was cyclic. The permutation-based approaches were particularly effective when the number of variables was small. However, the permutation-based approaches dropped in performance as the number of variables increased, as the number of possible permutations increases exponentially with the number of variables. The iterative approaches seemed particularly effective when the number of time series was large and sufficient time steps were available. The NOTEARS approach was effective as well, yet our developed methods were competitive with NOTEARS. We have also briefly investigated the performance of our methodologies on time-independent data. We have used simulated data in the form of a linear structural equation model. Rather surprisingly, we discovered that our methods, especially the greedy Metropolis-Hastings approach, were competitive with a state-of-the-art method such as NOTEARS.

Furthermore, we have applied our methods to real-life data. As a use case, we have investigated whether DAG-OMP was capable of inferring the structure between different grocery products using weekly scanner data of the convenience store chain "Dominick's Finer Foods". Secondly, we have evaluated all our methods on the real-life biological dataset provided by Sachs et al. [64], consisting of causal pathways in protein interactions within a biological cell. All methods evaluated on time-independent data achieved a similar performance on the biological dataset compared to the state-of-the-art method NOTEARS.

8.1 Limitations

Throughout this work, certain assumptions have been made to limit the scope of research. Furthermore, due to time constraints, some concepts in this thesis have not been investigated at the level of detail we had hoped. In this section, we will describe these limitations.

Strictness of the graphical models

The most severe limitation of this thesis is the strictness of the graphical model. Throughout this thesis, we have only considered two types of graphical models, the linear structural equation model (SEM) as per Definition 2.2 and the Vector AutoRegressive model of order 1 (VAR(1)) as per Definition 2.3. The linear SEM only allows for instantaneous linear relationships and the VAR(1) model only allows for linear relationships with a single time lag.

Although it is a realistic assumption that only the past can predict the present, in real-life measurements, instantaneous relations may exist, especially when the time intervals between consecutive measurements is large. If we acquire daily measurements, it is reasonable to assume that the rainfall of that day has influenced the wetness of the pavement that day. Therefore, a time series model should also allow for instantaneous effects.

Furthermore, it is reasonable to assume that more intricate relationships exist than linear relations based on values of exactly one time step ago. Especially in time series, seasonality or the day of the week can play a large role. Suggestions to allow for more intricate relationships will be discussed in the Section 8.2 where we will propose future directions of research.

Model Complexity Selection

From the outset of this thesis, the goal was to recover a structure of $\mathbf{X}$ that captures the relations between the variables, while simultaneously being intuitive to understand. Sparsity of the structure, meaning the graphical model contains few arcs, is crucial for obtaining a simple graphical model. For iterative procedures, quite some different methods have been devised to select an appropriate number of arcs, as iterative procedures provide an ordering of the inferred arcs.

However, for the permutation-based approaches discussed in Chapter 4, little research has been done to sensibly select an appropriate number of arcs. As permutation-based approaches estimate a full directed acyclic graph that respects a given permutation, numerous arcs in W are superfluous. Nevertheless, not many sensible approaches have been discussed that can "prune" such a coefficient matrix W efficiently. Thresholding the coefficient matrix is an effective approach, yet the approach is quite naive as the coefficient size of an arc is not always a suitable indication of its importance. Furthermore, selecting no principled approach has been provided to select such a threshold. We have investigated the impact of the threshold value on the simulation results in Appendix B.5, and the threshold value indeed significantly impacts the structural performance. Therefore, we consider the lack of suitable methods for selecting an appropriate number of arcs for permutation-based methods a limitation of this thesis.

For the continuous-based approaches discussed in Chapter 5, no methods are provided to select an appropriate number of arcs as well. On the one hand, the LASSO-penalty used in NOTEARS and DAG-LASSO does promote some degree of sparsity. However, little research has been done into selecting the correct magnitude of this LASSO-penalty parameter in our time-series setting. We also consider this to be a shortcoming of this work.

8.2 Future Work

Although this thesis can be regarded as quite a comprehensive work, numerous avenues are interesting directions for future work. Due to time constraints, these were not pursued. However, we will provide some initial investigations and pointers which an enthusiastic reader may continue on.

Allowing for more complex models

As mentioned in the previous section, the VAR(1) model is quite a simple model that does not allow for much flexibility. There are several approaches to allow for more complex models.

Incorporating instantaneous relationships. Although instantaneous relations may be unrealistic in theory, the time intervals at which variables are measured can be so far apart that we should also incorporate instantaneous relationships in our model.

A natural extension is a combination of the VAR(1) model from Definition 2.3 and the linear Structural Equation Model from Definition 2.2. Instantaneous relations are captured in the coefficient matrix W_0, and time-lagged relations are captured in W_1,

$$X_{t,\cdot} = X_{t,\cdot}W_0 + X_{t-1,\cdot}W_1 + \varepsilon_t, \tag{8.1}$$

where ε_t is a p-dimensional vector representing the noise. Note that W_0 must be an acyclic coefficient matrix where the diagonal entries are equal to zero, just as for the linear SEM.

Extending to VAR(k) models. Throughout this thesis, we have predominantly focused on *time series* data. As our goal was to infer the structure of a graphical model, a natural first step was the VAR(1) model, where we have but a single coefficient matrix W. However, it is reasonable to assume that a vector of variables $X_{t,\cdot}$ depends on more than just its previous time step $X_{t-1,\cdot}$. Therefore, a VAR(1) model might fail to capture more intricate relations. Extending the proposed methodologies to more complex models would be an interesting direction for future work.

We can quite easily extend the VAR(1) model by considering a VAR(k) model. Here, the values of $X_{t,\cdot}$ do not depend only on the past value, but the past k values,

$$X_{t,\cdot} = \sum_{i=1}^{k} X_{t-1,\cdot} W_i + \varepsilon_t, \tag{8.2}$$

where ε_t represents some zero-mean noise. For a VAR(1) model, we had $k = 1$ and we only had the matrix W_1, which we simply denoted as W. Now, we have k coefficient matrices.

Enforcing acyclicity of W or W_1 was quite intuitive and straightforward, as we could regard this as one network. However, the notion of acyclicity has become ambiguous now that we have k coefficient matrices. First, we could say that the structure is acyclic if and only if all its k coefficient matrices separately are acyclic. However, this could mean that in W_1 we have the arc $(1, 2)$, and W_2 contains the arc $(2, 1)$ such that we still have some form of cyclic dependency. A second approach would be to consider some sort of combined network, where the combination of all coefficient matrices must be acyclic. Consider W', where

$$W' = \sum_{i=1}^{k} |W_i|, \tag{8.3}$$

where $|\cdot|$ represents the element-wise absolute value. Therefore, W' contains an arc (i, j) if and only if the arc (i, j) is contained in at least one of the k coefficient matrices. We then could consider the structure to be acyclic if and only if W' is acyclic. This means that if there is an arc (i, j) in some coefficient matrix, then we know that there is no other path from variable j to variable i, where we can use all arcs from the k coefficient matrices. Such a definition might be more sensible as there can be no cycles in the relations between the variables across different time lags. A similar approach for learning sparse VAR(k) models by decomposing the indices into groups has been investigated in [50], so that may prove a useful starting point.

So, extending the methodologies and acyclicity to higher-order models is not more difficult, but the notion of acyclicity becomes less intuitive. It would be interesting to see how higher-order models are perhaps more suitable to estimate more complex simulated data and real-life time series, and what the advantages and disadvantages of the proposed notions of acyclicity are.

As a side note, instantaneous relationships can also be incorporated in VAR(k) models, yielding a structural VAR(k) model. Then, $X_{t,\cdot}$ can be written as

$$X_{t,\cdot} = X_{t,\cdot} W_0 + \sum_{i=1}^{k} X_{t-1,\cdot} W_i + \varepsilon_t, \tag{8.4}$$

where W_0 characterizes instantaneous relationships, W_i characterizes time-lagged relationships, and ε_t represents random noise. Again, W_0 must be acyclic with zero entries on the diagonal.

Non-linear relationships. Rather than only allowing for linear relationships in the form of a coefficient matrix W, we can also learn non-linear relationships between our variables. Rather than assuming X_j is a linear function of its parent set $\mathrm{Pa}(X)$,

$$X_j = \sum_{X_i \in \mathrm{Pa}(X_j)} w_{ij} X_i, \tag{8.5}$$

we can also assume that X_j is some non-linear function g of its parents,

$$X_j = g(\mathrm{Pa}(X_j)). \tag{8.6}$$

Several methods exist to learn non-linear relationships, such as the recent non-linear version of NOTEARS [88], and DAG-GNN [85] where the authors employ a variational autoencoder. Both utilize neural networks to learn non-linear relationships between variables.

More complex noise structure. Throughout this thesis, we have made the assumption that all noise random variables ε_t are independently and identically distributed as a Gaussian random variable with zero mean and an identity matrix as the covariance matrix,

$$\varepsilon_t \sim \mathcal{N}(\mathbf{0}, I_p). \tag{8.7}$$

However, this assumption is rather strict, as such a homogeneity assumption of the noise variables is often violated in practice. Some variables may exhibit larger fluctuations, or in real life, some variables can be measured more accurately.

For future work, the suitability of the model may increase if we allow for more complex noise structures. Rather than assuming that $\mathbb{V}(\varepsilon_t) = I_p$, we can assume that all noise components are independent yet heterogeneous,

$$\mathbb{V}(\varepsilon_t) = \operatorname{diag}\left(\omega_1^2, \ldots, \omega_p^2\right), \tag{8.8}$$

where ω_i^2 represents the variance associated with the noise of the ith variable. Furthermore, $\operatorname{diag}(\omega_1^2, \ldots, \omega_2^2) \in \mathbb{R}^{p \times p}$ represents the diagonal matrix with ω_i^2 as the ith value along the diagonal. It should be noted, however, that this adds a total of p extra parameters to estimate, which may be problematic when we have few samples. However, as the number of possible arcs is quadratic with respect to the number of variables, the number of parameters does not drastically increase.

We can also assume even more complex noise structures. We can, for example, allow some dependency between noise variables of the same time step. Then, we assume that

$$\mathbb{V}(\varepsilon_t) = \Omega, \tag{8.9}$$

where $\Omega \in \mathbb{R}^{p \times p}$ is positive semi-definite. This yields $p(p-1)/2$ extra parameters over Equation 8.8, so the number of parameters remains quadratic with respect to the number of variables.

Developing Theoretical Guarantees

Although we have provided an interesting conjecture for autoregressive models, this conjecture was not extended to more complex models such as VAR(1) models. Additionally, it would have been nice to provide some theoretical guarantees for some of our methodologies.

Orthogonal Matching Pursuit. For the Orthogonal Matching Pursuit algorithm discussed in Section 6.1, performance guarantees have been provided for the standard regression setting with noise [9, 86] and without noise [75]. However, our setting is more involved, where we have multiple response variables and there are dependencies in the data. Therefore, existing performance guarantees for Orthogonal Matching Pursuit could not be directly applied.

However, it would be of great value to investigate whether we can translate the orthogonal matching pursuit guarantees in some way to the `DAG-OMP` algorithm. Having a formal statement that guarantees that the correct features will be recovered under certain conditions can provide more insights into the performance of `DAG-OMP`. Unfortunately, this was not pursued due to time constraints, but it could be an interesting avenue to investigate.

Greedy Metropolis-Hastings. When we evaluated our methods, the greedy Metropolis-Hastings approach seemed to perform surprisingly well. For the linear SEM with equal variances, some performance guarantees are provided for algorithms that employ a greedy approach such as [12] and [57]. In [12], the authors show that when the data has indeed been generated by a linear SEM and some other conditions are met, for example that the covariance matrix of the data has only nonnegative eigenvalues, then the probability of recovering the true structure converges to one in the infinite sample setting.

Both methods are similar to ours, so we expect a performance guarantee to exist for the greedy Metropolis-Hastings algorithm as well. For example, an asymptotic bound on the number of iterations required to recover the support of the data generating matrix. We expect that these performance guarantees can also be cast in the VAR(1) setting. Providing such a performance guarantee could be an interesting direction for future work.

Nevertheless, both [12] and [57] state that the risk of such a greedy approach is that we might get stuck in a local optimum. Therefore, one of their proposals is to run the algorithm several times from different starting points. Furthermore, a similar discussed in [58] states that their greedy method is not guaranteed to find the best scoring graph due to local optima. So, the greedy nature of our method may cause difficulties when deriving performance guarantees as well.

Sparsity imposed by acyclicity

Another interesting direction for future work that we have not thoroughly explored is how acyclicity could be used to derive tighter theoretical guarantees for known methods such as the LASSO. Well-known theoretical guarantees exist for the LASSO approach in the regression setting. For example, in [81], the authors show that under certain conditions, the LASSO-penalty should be approximately $\lambda \approx \sqrt{\log(p)/n}$ in order to recover the nonzero coefficients. Here, n is the number of independent samples and p is the number of variables. Note that this is for the linear regression setting where all samples are independent, not for a time series setting as we are investigating. Now, what if instead of knowing that the coefficient vector contains only k non-zero coefficients, or when we know that the coefficient *matrix* is acyclic? If our coefficient matrix is acyclic, we know it contains at most $p(p+1)/2$ nonzero parameters, which is already much fewer than the possible p^2 parameters.

The question arises whether the acyclicity assumption can provide tighter bounds on convergence rates, penalty parameter values, etc. The number of possible parameters remains quadratic, so in terms of complexity we have not gained much. However, a more optimistic perspective is by considering the reduction in search space. If we would allow any coefficient matrix, or equivalently any structure on p variables without self-loops, then there are a total of 2^{p^2-p} different structures, as p^2-p entries can either be zero or non-zero. However, we have seen that the number of directed acyclic graphs grows much smaller. The number of possible directed acyclic graphs up to fourteen nodes has been reported in [51]. Now, define R_p as the ratio of the total number of possible directed acyclic graphs on p nodes over the total number of possible directed graphs on p nodes. Then, some values of R_p are

$$R_1 = 1, R_2 = 0.75, \ldots, R_5 = 0.027, \ldots, R_{10} = 3.4 \cdot 10^{-9}, \ldots, R_{14} = 2.3 \cdot 10^{-19}. \tag{8.10}$$

From Equation 8.10, we see that the search space of structures has decreased massively. Even for a moderate total of ten nodes, knowing that the true structure is acyclic has reduced the number of possible structures by a factor 10^9. As acyclicity indeed massively decreases the search space, perhaps this may also imply that we can obtain tighter bounds for several performance guarantees of existing methods. Investigating this in greater depth as future work may provide some interesting novel results.

Bibliography

[1] Awad H. Al-Mohy and Nicholas J. Higham. A new scaling and squaring algorithm for the matrix exponential. *SIAM Journal on Matrix Analysis and Applications*, 31(3):970–989, 2010. 66, 81

[2] M. Andrle, L. Rebollo-Neira, and E. Sagianos. Backward-optimized orthogonal matching pursuit approach. *IEEE Signal Processing Letters*, 11(9):705–708, 2004. 80

[3] Mark Barlett and James Cussens. Advances in bayesian network learning using integer programming. In *Proceedings of the Twenty-Ninth Conference on Uncertainty in Artificial Intelligence*, UAI'13, page 182–191, Arlington, Virginia, USA, 2013. AUAI Press. 23

[4] Mark Bartlett and James Cussens. Integer linear programming for the bayesian network structure learning problem. *Artificial Intelligence*, 244:258–271, 2017. Combining Constraint Solving with Mining and Learning. 23

[5] Christoph Bergmeir, Rob J. Hyndman, and Bonsoo Koo. A note on the validity of cross-validation for evaluating autoregressive time series prediction. *Computational Statistics Data Analysis*, 120:70–83, 2018. 91

[6] Garrett Birkhoff. Three observations on linear algebra. *Univ. Nac. Tacuman, Rev. Ser. A*, 5:147–151, 1946. 49

[7] Graham Brightwell and Peter Winkler. Counting linear extensions is #p-complete. In *Proceedings of the Twenty-Third Annual ACM Symposium on Theory of Computing*, STOC '91, page 175–181, New York, NY, USA, 1991. Association for Computing Machinery. 34

[8] Prabir Burman, Edmond Chow, and Deborah Nolan. A cross-validatory method for dependent data. *Biometrika*, 81:351–358, 1994. 92

[9] T. Tony Cai and Lie Wang. Orthogonal matching pursuit for sparse signal recovery with noise. *IEEE Transactions on Information Theory*, 57(7):4680–4688, 2011. 118

[10] Nancy Cartwright. Are rcts the gold standard? *BioSocieties*, 2(1):11–20, 2007. 4

[11] 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. 39

[12] Wenyu Chen, Mathias Drton, and Y Samuel Wang. On causal discovery with an equal-variance assumption. *Biometrika*, 106(4):973–980, 09 2019. 118, 119

[13] David Maxwell Chickering. *Learning Bayesian Networks is NP-Complete*, pages 121–130. Springer New York, New York, NY, 1996. 15

[14] Diego Colombo, Marloes H. Maathuis, Markus Kalisch, and Thomas S. Richardson. Learning high-dimensional directed acyclic graphs with latent and selection variables. *The Annals of Statistics*, 40(1):294–321, 2012. 18

[15] Gregory F. Cooper. The computational complexity of probabilistic inference using bayesian belief networks. *Artificial Intelligence*, 42(2):393–405, 1990. 4

[16] James Cussens. Bayesian network learning with cutting planes. In *Proceedings of the Twenty-Seventh Conference on Uncertainty in Artificial Intelligence*, UAI'11, page 153–160, Arlington, Virginia, USA, 2011. AUAI Press. 22, 24

[17] Aramayis Dallakyan and Mohsen Pourahmadi. Learning bayesian networks through birkhoff polytope: A relaxation method. *CoRR*, abs/2107.01658, 2021. 56

[18] Ivan Damnjanovic, Matthew E. P. Davies, and Mark D. Plumbley. Smallbox - an evaluation framework for sparse representations and dictionary learning algorithms. In Vincent Vigneron, Vicente Zarzoso, Eric Moreau, Rémi Gribonval, and Emmanuel Vincent, editors, *Latent Variable Analysis and Signal Separation*, pages 418–425, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg. 75

[19] Tristan Deleu, António Góis, Chris Chinenye Emezue, Mansi Rankawat, Simon Lacoste-Julien, Stefan Bauer, and Yoshua Bengio. Bayesian structure learning with generative flow networks. In *ICLR Workshop on Deep Generative Models for Highly Structured Data*, 2022. 112

[20] Vera Djordjilović, Monica Chiogna, and Jiří Vomlel. An empirical comparison of popular structure learning algorithms with a view to gene network inference. *International Journal of Approximate Reasoning*, 88:602–613, 2017. 100

[21] B. Efron. Bootstrap Methods: Another Look at the Jackknife. *The Annals of Statistics*, 7(1):1 – 26, 1979. 85

[22] Doris Entner and Patrik Hoyer. On causal discovery from time series data using fci. *Proceedings of the 5th European Workshop on Probabilistic Graphical Models, PGM 2010*, 09 2010. 18

[23] Kim Esbensen and Paul Geladi. Principles of proper validation: use and abuse of re-sampling for validation. *Journal of Chemometrics*, 24:168 – 187, 03 2010. 91

[24] Ronald Aylmer Fisher et al. 014: On the" probable error" of a coefficient of correlation deduced from a small sample. 1921. 16

[25] Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman amp; Co., USA, 1990. 68

[26] Maxime Gasse, Alex Aussem, and Haytham Elghazel. An experimental comparison of hybrid algorithms for bayesian network structure learning. In Peter A. Flach, Tijl De Bie, and Nello Cristianini, editors, *Machine Learning and Knowledge Discovery in Databases*, pages 58–73, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg. 100

[27] Sarah Gelper, Ines Wilms, and Christophe Croux. Identifying demand effects in a large network of product categories. *Journal of Retailing*, 92(1):25–39, 2016. 108

[28] Hemant S. Goklani, Jignesh N. Sarvaiya, and A. M. Fahad. Image reconstruction using orthogonal matching pursuit (omp) algorithm. In *2014 2nd International Conference on Emerging Technology Trends in Electronics, Communication and Networking*, pages 1–5, 2014. 68

[29] C. W. J. Granger. Investigating causal relations by econometric models and cross-spectral methods. *Econometrica*, 37(3):424–438, 1969. 5

[30] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. Springer New York Inc., New York, NY, USA, 2001. 64

[31] W. K. Hastings. Monte carlo sampling methods using markov chains and their applications. *Biometrika*, 57(1):97–109, 1970. 39

[32] Sander Hofman. Making euv: From lab to fab, Mar 2022. 3

[33] Guoxian Huang and Lei Wang. High-speed signal reconstruction with orthogonal matching pursuit via matrix inversion bypass. pages 191–196, 10 2012. 75

[34] Robin John Hyndman and George Athanasopoulos. *Forecasting: Principles and Practice*. OTexts, Australia, 2nd edition, 2018. 28

[35] Aapo Hyvärinen, Kun Zhang, Shohei Shimizu, and Patrik O. Hoyer. Estimation of a structural vector autoregression model using non-gaussianity. *Journal of Machine Learning Research*, 11(56):1709–1731, 2010. 21

[36] Donald B. Johnson. Finding all the elementary circuits of a directed graph. *SIAM Journal on Computing*, 4(1):77–84, 1975. 80

[37] Hidde De Jong. Modeling and simulation of genetic regulatory systems: A literature review. *Journal of Computational Biology*, 9:67–103, 2002. 3

[38] A. B. Kahn. Topological sorting of large networks. *Commun. ACM*, 5(11):558–562, nov 1962. 66

[39] Wagner A. Kamakura and Wooseong Kang. Chain-wide and store-level analysis for cross-category management. *Journal of Retailing*, 83(2):159–170, 2007. 108

[40] S. N. Lahiri. *Bootstrap Methods*, pages 17–43. Springer New York, New York, NY, 2003. 85

[41] S. L. Lauritzen and D. J. Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society. Series B (Methodological)*, 50(2):157–224, 1988. 4

[42] Hao-Chih Lee, Matteo Danieletto, Riccardo Miotto, Sarah Cherng, and Joel T. Dudley. Scaling structural learning with no-bears to infer causal transcriptome networks. *Pacific Symposium on Biocomputing. Pacific Symposium on Biocomputing*, 25:391–402, 2020. 25

[43] Hanxi Li, Yongsheng Gao, and Jun Sun. Fast kernel sparse representation. In *2011 International Conference on Digital Image Computing: Techniques and Applications*, pages 72–77, 2011. 75

[44] Rami Mahdi and Jason Mezey. Sub-local constraint-based learning of bayesian networks using a joint dependence criterion. *Journal of Machine Learning Research*, 14(13):1563–1603, 2013. 16

[45] Daniel Malinsky and Peter Spirtes. Causal structure learning from multivariate time series in settings with unmeasured confounding. In *Proceedings of 2018 ACM SIGKDD workshop on causal discovery*, pages 23–47. PMLR, 2018. 18

[46] S.G. Mallat and Zhifeng Zhang. Matching pursuits with time-frequency dictionaries. *IEEE Transactions on Signal Processing*, 41(12):3397–3415, 1993. 68

[47] Christopher Meek. Causal inference and causal explanation with background knowledge. In *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*, UAI'95, page 403–410, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc. 16

[48] Nicholas Metropolis, Arianna W. Rosenbluth, Marshall N. Rosenbluth, Augusta H. Teller, and Edward Teller. Equation of State Calculations by Fast Computing Machines. , 21(6):1087–1092, June 1953. 39

[49] Brady Neal. Introduction to causal inference from a machine learning perspective. *Course Lecture Notes*, 2020. 16, 17

[50] William B. Nicholson, David S. Matteson, and Jacob Bien. Varx-l: Structured regularization for large vector autoregressions with exogenous variables. *International Journal of Forecasting*, 33(3):627–651, 2017. 117

[51] OEIS Foundation Inc. The On-Line Encyclopedia of Integer Sequences, 2022. Published electronically at `https://oeis.org/A003024`. 119

[52] Roxana Pamfil, Nisara Sriwattanaworachai, Shaan Desai, Philip Pilgerstorfer, Konstantinos Georgatzis, Paul Beaumont, and Bryon Aragam. Dynotears: Structure learning from time-series data. In Silvia Chiappa and Roberto Calandra, editors, *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, volume 108 of *Proceedings of Machine Learning Research*, pages 1595–1605. PMLR, 26–28 Aug 2020. 25

[53] Koen Pauwels. How retailer and competitor decisions drive the long-term effectiveness of manufacturer promotions for fast moving consumer goods. *Journal of Retailing*, 83(3):297–308, 2007. 108

[54] Judea Pearl. Fusion, propagation, and structuring in belief networks. *Artificial Intelligence*, 29(3):241–288, 1986. 4, 16

[55] Judea Pearl. *Causality: Models, Reasoning and Inference*. Cambridge University Press, USA, 2nd edition, 2009. v, 1, 2, 4, 17

[56] Judea Pearl and Thomas Verma. A theory of inferred causation. In *KR*, 1991. 15

[57] J. Peters and P. Bühlmann. Identifiability of gaussian structural equation models with equal error variances. *Biometrika*, 101(1):219–228, 2014. 118, 119

[58] Jonas Peters, Joris M. Mooij, Dominik Janzing, and Bernhard Schölkopf. Causal discovery with continuous additive noise models. *Journal of Machine Learning Research*, 15:2009–2053, 2014. 119

[59] K. B. Petersen and M. S. Pedersen. The matrix cookbook, October 2008. Version 20081110. 127

[60] Joseph Ramsey and Bryan Andrews. Fask with interventional knowledge recovers edges from the sachs model. *ArXiv*, abs/1805.03108, 2018. 112

[61] 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. 33

[62] Jakob Runge. Discovering contemporaneous and lagged causal relations in autocorrelated nonlinear time series datasets. In *Conference on Uncertainty in Artificial Intelligence*, pages 1388–1397. PMLR, 2020. 18

[63] Jakob Runge, Peer Nowack, Marlene Kretschmer, Seth Flaxman, and Dino Sejdinovic. Detecting and quantifying causal associations in large nonlinear time series datasets. *Science Advances*, 5(11):eaau4996, 2019. 18

[64] Karen Sachs, Omar Perez, Dana Pe'er, Douglas Lauffenburger, and Garry Nolan. Causal protein-signaling networks derived from multiparameter single-cell data. *Science (New York, N.Y.)*, 308:523–9, 05 2005. vii, vii, viii, 112, 113, 115

[65] Rajen Shah and Jonas Peters. The hardness of conditional independence testing and the generalised covariance measure. *Annals of Statistics*, 48, 04 2018. 16

[66] Shohei Shimizu, Patrik O. Hoyer, Aapo Hyv228;rinen, and Antti Kerminen. A linear non-gaussian acyclic model for causal discovery. *Journal of Machine Learning Research*, 7(72):2003–2030, 2006. 19, 21, 76

[67] Konstantinos Skianis, Nikolaos Tziortziotis, and Michalis Vazirgiannis. Orthogonal matching pursuit for text classification. *ArXiv*, abs/1807.04715, 2018. 68

[68] P. Spirtes, C. Glymour, and R. Scheines. *Causation, Prediction, and Search*. MIT press, 2nd edition, 2000. 17, 18

[69] Peter Spirtes and Clark Glymour. An algorithm for fast recovery of sparse causal graphs. *Social Science Computer Review*, 9(1):62–72, 1991. 17

[70] Shuba Srinivasan, Koen Pauwels, Dominique M. Hanssens, and Marnik G. Dekimpe. Do promotions benefit manufacturers, retailers, or both? *Management Science*, 50(5):617 – 629, 2004. Cited by: 177; All Open Access, Green Open Access. 108

[71] M. Stone. Cross-validatory choice and assessment of statistical predictions. *Journal of the Royal Statistical Society. Series B (Methodological)*, 36(2):111–147, 1974. 90

[72] Milan Studený and James Cussens. Towards using the chordal graph polytope in learning decomposable models. *International Journal of Approximate Reasoning*, 88:259–281, 2017. 23

[73] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1):267–288, 1996. 60

[74] Ryan Tibshirani and Jonathan Taylor. The solution path of the generalized lasso. *The Annals of Statistics*, 39, 05 2010. 61, 62

[75] J.A. Tropp. Greed is good: algorithmic results for sparse approximation. *IEEE Transactions on Information Theory*, 50(10):2231–2242, 2004. 68, 118

[76] Joel A. Tropp and Anna C. Gilbert. Signal recovery from random measurements via orthogonal matching pursuit. *IEEE Transactions on Information Theory*, 53(12):4655–4666, 2007. 75

[77] Ioannis Tsamardinos, Laura Brown, and Constantin Aliferis. The max-min hill-climbing bayesian network structure learning algorithm. *Machine Learning*, 65:31–78, 10 2006. 101

[78] Alexander L. Tulupyev and Sergey I. Nikolenko. Directed cycles in bayesian belief networks: Probabilistic semantics and consistency checking complexity. In Alexander Gelbukh, Álvaro de Albornoz, and Hugo Terashima-Marín, editors, *MICAI 2005: Advances in Artificial Intelligence*, pages 214–223, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg. 6

[79] John Von Neumann. A certain zero-sum two-person game equivalent to the optimal assignment problem. *Contributions to the Theory of Games*, 2(0):5–12, 1953. 49

[80] Matthew J. Vowels, Necati Cihan Camgoz, and Richard Bowden. D'ya like dags? a survey on structure learning and causal discovery. *ACM Comput. Surv.*, mar 2022. Just Accepted. 15, 100

[81] Martin J. Wainwright. Sharp thresholds for high-dimensional and noisy sparsity recovery using l1-constrained quadratic programming (lasso). *IEEE Trans. Inf. Theor.*, 55(5):2183–2202, may 2009. 119

[82] Xiangyu Wang, David Dunson, and Chenlei Leng. No penalty no tears: Least squares in high-dimensional linear models. In Maria Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1814–1822, New York, New York, USA, 20–22 Jun 2016. PMLR. 102

[83] Dennis Wei, Tian Gao, and Yue Yu. Dags with no fears: A closer look at continuous optimization for learning bayesian networks. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 3895–3906. Curran Associates, Inc., 2020. 25

[84] Norbert Wiener. The theory of prediction. *Modern mathematics for engineers*, 1956. 5

[85] Yue Yu, Jie Chen, Tian Gao, and Mo Yu. DAG-GNN: DAG structure learning with graph neural networks. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 7154–7163. PMLR, 09–15 Jun 2019. 25, 112, 117

[86] Tong Zhang. On the consistency of feature selection using greedy least squares regression. *Journal of Machine Learning Research*, 10(19):555–568, 2009. 68, 118

[87] Xun Zheng, Bryon Aragam, Pradeep Ravikumar, and Eric P. Xing. Dags with no tears: Continuous optimization for structure learning. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, page 9492–9503, Red Hook, NY, USA, 2018. Curran Associates Inc. 23, 48, 57, 102, 109, 112, 114

[88] Xun Zheng, Chen Dan, Bryon Aragam, Pradeep Ravikumar, and Eric Xing. Learning sparse nonparametric dags. In *International Conference on Artificial Intelligence and Statistics*, pages 3414–3425. PMLR, 2020. 25, 117

[89] Shuheng Zhou. Thresholding procedures for high dimensional variable selection and statistical estimation. In Y. Bengio, D. Schuurmans, J. Lafferty, C. Williams, and A. Culotta, editors, *Advances in Neural Information Processing Systems*, volume 22. Curran Associates, Inc., 2009. 24, 102

[90] 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. 24

[91] Hufei Zhu, Wen Chen, and Yanpeng Wu. Efficient implementations for orthogonal matching pursuit. *Electronics*, 9:1507, 09 2020. 75

[92] Eric Zivot and Jiahui Wang. *Vector Autoregressive Models for Multivariate Time Series*, pages 369–413. Springer New York, New York, NY, 2003. 51

Appendix A

Derivations

A.1 Rewriting the matrix notation to vector notation

In this section, we will provide a derivation to cast our matrix regression notation

$$\mathbf{X}_{2:T,\cdot} = \mathbf{X}_{1:T-1,\cdot} W + \varepsilon_{2:T} \tag{A.1}$$

into a vector regression notation

$$y' = X' w' + \varepsilon \tag{A.2}$$

. To achieve this, we construct a response vector y', explanatory variables X', and a coefficient vector w' such that

$$\mathbf{X}_{2:T,\cdot} - \mathbf{X}_{1:T-1,\cdot} W = \varepsilon_{2:T} \in \mathbb{R}^{T-1 \times p} \iff y' - X' w' = \text{vec}(\varepsilon_{2:T}), \tag{A.3}$$

where the the vectorized dimensions will be

$$y' \in \mathbb{R}^{(T-1)\cdot p}, \qquad X' \in \mathbb{R}^{(T-1)\cdot p \times p^2}, \qquad w' \in \mathbb{R}^{p^2}. \tag{A.4}$$

.

Rewrite response matrix $\mathbf{X}_{2:T,\cdot}$ to response vector y'. To rewrite our response matrix $\mathbf{X}_{2:T,\cdot} \in \mathbb{R}^{(T-1)\times p}$ to a response vector $y' \in \mathbb{R}^{(T-1)\cdot p}$, We will vertically stack the p columns to get a single column vector $y \in \mathbb{R}^{p\cdot(T-1)}$. In mathematical notation,

$$\begin{aligned} y' &= \text{vec}(\mathbf{X}_{2:T,1}, \mathbf{X}_{2:T,2}, \ldots, \mathbf{X}_{2:T,p-1}, \mathbf{X}_{2:T,p}) && \text{(A.5)} \\ &= \big(\underbrace{X_{2,1}, \ldots, X_{T,1}}_{\mathbf{X}_{2:T,1}}, \underbrace{X_{2,2}, \ldots, X_{T,2}}_{\mathbf{X}_{2:T,2}}, \ldots, \underbrace{X_{2,p-1}, \ldots, X_{T,p-1}}_{\mathbf{X}_{2:T,p-1}}, \underbrace{X_{2,p}, \ldots, X_{T,p}}_{\mathbf{X}_{2:T,p}} \big)^T. && \text{(A.6)} \end{aligned}$$

Rewrite coefficient matrix W to coefficient vector w'. Secondly, we will vectorize our coefficient matrix $W \in \mathbb{R}^{p\times p}$ to a coefficient vector $w' \in \mathbb{R}^{p^2}$ by vertically stacking the columns $w_{\cdot,j}$. Therefore, we have

$$\begin{aligned} w' &= \text{vec}(w_{1,\cdot}, w_{2,\cdot}, \ldots, w_{p-1,\cdot}, w_{p,\cdot}) && \text{(A.7)} \\ &= (w_{11}, \ldots, w_{p1}, \ldots, w_{1p}, \ldots, w_{pp})^T. && \text{(A.8)} \end{aligned}$$

Rewrite explanatory matrix $\mathbf{X}_{1:T-1,\cdot}$ to explanatory matrix X'. Lastly, we need to transform our explanatory matrix $\mathbf{X}_{\mathbf{1:T-1},\cdots}$ to match our response vector y'. Therefore, our new explanatory matrix must be of the form $X' \in \mathbb{R}^{(T-1)\cdot p \times p^2}$ such that

$$\text{vec}\left(\mathbf{X}_{1:T-1,\cdot} W\right) = X' w'. \tag{A.9}$$

We achieve this by repeating the explanatory matrix $\mathbf{X}_{1:T-1,\cdot} \in \mathbb{R}^{(T-1)\times p}$ a total of p times along the diagonal. The other entries in the matrix X' are all equal to zero. In matrix notation, we get

$$X' = \begin{pmatrix} \mathbf{X}_{1:T-1,\cdot} & \mathbf{O} & \cdots & \mathbf{O} & \mathbf{O} \\ \mathbf{O} & \mathbf{X}_{1:T-1,\cdot} & \cdots & \mathbf{O} & \mathbf{O} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \mathbf{O} & \mathbf{O} & \cdots & \mathbf{X}_{1:T-1,\cdot} & \mathbf{O} \\ \mathbf{O} & \mathbf{O} & \cdots & \mathbf{O} & \mathbf{X}_{1:T-1,\cdot} \end{pmatrix} \in \mathbb{R}^{(T-1)\cdot p \times p^2}. \tag{A.10}$$

A more concise mathematical notation for Equation A.10 using the Kronecker product is

$$X' = I_p \otimes \mathbf{X}_{1:T-1,\cdot} \in \mathbb{R}^{(T-1)\cdot p \times p^2}. \tag{A.11}$$

To conclude, we see that we have rewritten our matrix setting to a vectorized setting.

A.2 Deriving the gradients for gradient descent

In the coming two paragraphs, we will be deriving both gradients $\nabla_P C'(P,U)$ and $\nabla_U C'(P,U)$ required for the gradient descent algorithm in Section 5.1. We start with $\nabla_U C'(P,U)$, as that one is easier.

Deriving $\nabla_U C'(P,U)$. The gradient of $C'(P,U)$ with respect to U can be quite easily derived by using matrix derivatives. By filling in the derived expected cost from Equation 5.15 and using the linearity of the trace operation, we get that the matrix derivative of $C'(P,U)$ with respect to U is equal to

$$\begin{aligned} \frac{\partial C'(P,U)}{\partial U} &= \frac{\partial \mathrm{Tr}\left(\mathbb{V}\left(\varepsilon_t\right) + \left(W^* - W\right)^T \mathbb{V}\left(X_{t,\cdot}\right)\left(W^* - W\right)\right)}{\partial U} \\ &= \frac{\partial \mathrm{Tr}\left(\mathbb{V}\left(\varepsilon_t\right)\right)}{\partial U} + \frac{\partial \mathrm{Tr}\left(\left(W^* - W\right)^T \mathbb{V}\left(X_{t,\cdot}\right)\left(W^* - W\right)\right)}{\partial U} \\ &= \frac{\partial \mathrm{Tr}\left(\left(W^* - W\right)^T \mathbb{V}\left(X_{t,\cdot}\right)\left(W^* - W\right)\right)}{\partial U}, \end{aligned} \tag{A.12}$$

where we could remove the first component in Equation A.12 as the matrix derivative of ε_t with respect to U is zero.

Now, let us use Equation (111) of the matrix cookbook [59], which corresponds to

$$\frac{\partial \mathrm{Tr}(X^T B X)}{\partial X} = XB^T + XB, \qquad \text{(Equation 111 of [59])}$$

which is equal to $2XB$ if B is symmetric.

Applying this formula on Equation A.12 and applying the chain rule yields

$$\begin{aligned} \frac{\partial \mathrm{Tr}\left(\left(W^* - W\right)^T \mathbb{V}\left(X_{t,\cdot}\right)\left(W^* - W\right)\right)}{\partial U} &= -2(W^* - W)\mathbb{V}\left(X_{t,\cdot}\right) P^{-1} \frac{\partial \mathrm{Tr}(U)}{\partial U} P \\ &= -2(W^* - W)\mathbb{V}\left(X_{t,\cdot}\right). \end{aligned} \tag{A.13}$$

Note, however, that we will fix the gradient of U to be equal to zero for all lower triangular entries. Therefore, we have that the partial derivatives in our gradient $\nabla_U C'(P,U)$ can be computed as

$$\left(\nabla_U C'(P,U)\right)_{ij} = \begin{cases} \left(-2(W - W^*)\mathbb{V}\left(X_{t,\cdot}\right)\right)_{ij} & \text{if } i \leq j, \\ 0 & \text{otherwise.} \end{cases} \tag{A.14}$$

Deriving $\nabla_P C'(P,U)$. Deriving the gradient of $C'(P,U)$ with respect to P is slightly more involved than with respect to U, but nevertheless doable. Analogous to the gradient with respect to U, we first derive the matrix derivative

$$\begin{aligned}\frac{\partial C'(P,U)}{\partial P} &= 2\text{Tr}\left((W^* - W)\mathbb{V}\left(X_{t,\cdot}\right)\frac{\partial \text{Tr}\left(W^* - W\right)}{\partial P}\right) \\ &= -2\text{Tr}\left((W^* - W)\mathbb{V}\left(X_{t,\cdot}\right)\frac{\partial P^{-1}UP}{\partial P}\right).\end{aligned}$$

What is left is to derive the partial derivatives

$$\frac{\partial P^{-1}UP}{\partial p_{ij}}. \tag{A.15}$$

We can use the product rule to decompose the partial derivative in Equation A.15 into the two components

$$\frac{\partial P^{-1}UP}{\partial p_{ij}} = \frac{\partial P^{-1}}{\partial p_{ij}}UP + P^{-1}U\frac{P}{\partial p_{ij}}, \tag{A.16}$$

where the second component in Equation A.16 can be computed as

$$P^{-1}U\frac{P}{\partial p_{ij}} = P^{-1}UJ_{ij}. \tag{A.17}$$

Here, J_{ij} corresponds to the matrix with zeros everywhere, except for the value in the ith row of the jth column, which is equal to one.

The first component in Equation A.16 requires computing the derivative of the inverse P^{-1}, which also has a closed form. The partial derivative of the inverse matrix P^{-1} can be written as

$$\frac{\partial P^{-1}}{\partial p_{ij}} = -P^{-1}\frac{\partial P}{\partial p_{ij}}P^{-1} = -P^{-1}J^{ij}P^{-1}. \tag{A.18}$$

Hence, the first component in Equation A.16 is equal to

$$\frac{\partial P^{-1}}{\partial p_{ij}}UP = -P^{-1}J^{ij}P^{-1}UP. \tag{A.19}$$

Putting the two components together, we can conclude that the partial derivative of the cost function $C'(P,U)$ with respect to p_{ij} is equal to

$$\begin{aligned}\frac{\partial C'(P,U)}{\partial p_{ij}} &= -2\text{Tr}\left(\mathbb{V}\left(X_{t,\cdot}\right)(W^* - W)^T\frac{\partial P^{-1}UP}{\partial p_{ij}}\right) \\ &= -2\text{Tr}\left(\mathbb{V}\left(X_{t,\cdot}\right)(W^* - W)^T\left(-P^{-1}J^{ij}P^{-1}UP + P^{-1}UJ^{ij}\right)\right),\end{aligned} \tag{A.20}$$

Then, the coefficients of the gradient $\nabla_P C'(P,U)$ are equal to

$$\left(\nabla_P C'(P,U)\right)_{ij} = \frac{\partial C'(P,U)}{\partial p_{ij}} \tag{A.21}$$

To verify our derivations, we have plotted the partial derivatives of our cost function both with respect to U and P in Example A.1, where we indeed see that the partial derivatives are equal to zero exactly where $C'(P,U)$ attains its minimum.

Example A.1 Verifying the gradients $\nabla_P C'(P,U)$ and $\nabla_U C'(P,U)$.

Consider a two-dimensional scenario where

$$P^* = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \qquad U^* = \begin{pmatrix} 0.8 & 0.5 \\ 0.0 & 0.6 \end{pmatrix}.$$

For each entry $p_{ij}, 1 \leq i, j \leq 2$, we have plotted both the cost function $C'(P^*, U^*)$ and the corresponding partial derivative in Figure A.1. All other values for P^* and U^* the same,

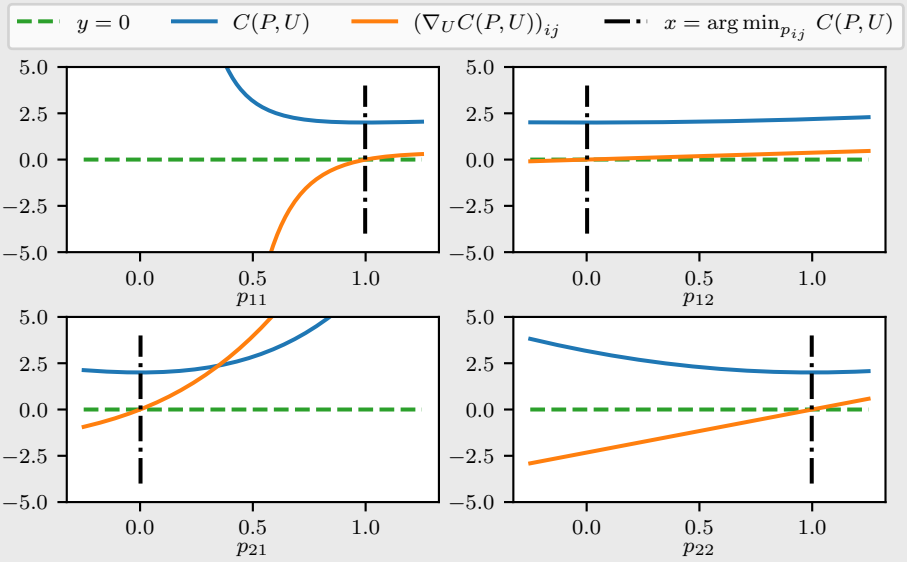


Figure A.1: Value of the cost function and its partial derivative with respect to p_{ij} as a function of the four coefficients p_{ij} of P.

When the cost function $C'(P,U)$ attains its minimum for p_{ij}, the partial derivative with respect to p_{ij} equals zero. For each upper triangular entry $u_{ij}, 1 \leq i < j \leq 2$, we have plotted the cost function and the corresponding partial derivatives in Figure A.2.

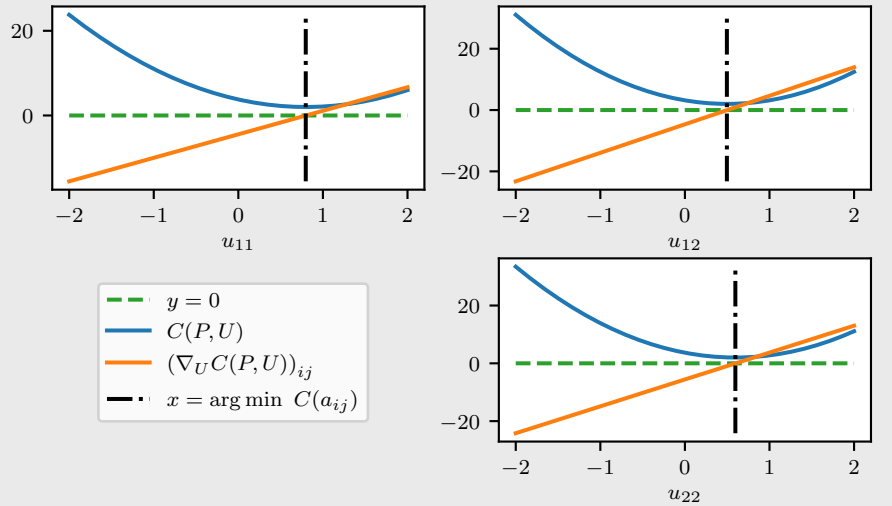


Figure A.2: Value of the cost function and its partial derivative with respect to u_{ij} as a function of the three upper triangular coefficients u_{ij} of U .

The cost function attains its minimum for u_{ij} when the partial derivative with respect to u_{ij} is equal to zero. Furthermore, we have checked that the slope of the partial derivative indeed corresponds to the rate of change of the cost function.

A.3 Difference of the negative log-likelihoods.

In this section of the appendix, we will provide more arguments to support our conjecture stated in Subsection 6.5.1. We will reiterate the conjecture here.

Conjecture A.1 Asymptotic value of $RCH(a_0, T)$

Let $|a_0| < 1$. Furthermore, let X be generated according to an AR(1) model. Then, we conjecture that the difference between the two negative log-likelihoods as defined in Equation 6.54 and Equation 6.55 converges to a chi-squared random variable with one degree of freedom, shifted two units to the left. That is,

$$\lim_{T\to\infty} -2LL_0(X) + 2LL_{\text{LOOCV}}(X) \sim \chi^2_1 - 2. \tag{A.22}$$

Moreover, let $RCH_0(a_0, T)$ denote the ratio of correct acceptance of H_0 as defined in Equation 6.60. Then, we conjecture the value of $RCH_0(a_0, T)$ converges to $\mathbb{P}(Q \leq 2)$ as T tends to infinity, where Q is a chi-squared random variable with one degree of freedom,

$$\lim_{T\to\infty} \mathbb{E}\left[RCH_0(a_0, T)\right] = \text{const}, \quad \text{where const} = \mathbb{P}(Q \leq 2) \;\text{ and }\; Q \sim \chi^2_1. \tag{A.23}$$

Decomposing Equation A.22 To support our claim in Conjecture A.1, we will first rewrite the equation and subsequently invoke Wilk's theorem. Let $-2LL_1(X)$ represent twice the negative log-likelihood of corresponding the maximum likelihood estimator $\hat{a}$,

$$-2LL_1(X) = -2\sum_{t=2}^{T} \log\left(-\frac{1}{\sqrt{2\pi}}\right) + \sum_{t=2}^{T} \left(X_t - \hat{a}X_{t-1}\right)^2,$$

where the maximum likelihood estimator of a_0 can be computed as

$$\hat{a} = \frac{\sum_{t=2}^{T} X_{t-1}X_t}{\sum_{t=2}^{T} X^2_{t-1}}. \tag{A.24}$$

We can rewrite $-2LL_0 + 2LL_{\text{LOOCV}}$ to

$$\begin{aligned}
-2LL_0 + LL_{\text{LOOCV}} &= \sum_{t=2}^{T} \left(X_t - a_0X_{t-1}\right)^2 - \sum_{t=2}^{T} \left(X_t - \hat{a}^{(-t)}X_{t-1}\right)^2 \\
&= \sum_{t=2}^{T} \left(X_t - a_0X_{t-1}\right)^2 - \sum_{t=2}^{T} \left(X_t - \left(\hat{a} - \hat{a} + \hat{a}^{(-t)}\right)X_{t-1}\right)^2 \\
&= \sum_{t=2}^{T} \left(X_t - a_0X_{t-1}\right)^2 - \sum_{t=2}^{T} \left(X_t - \hat{a}X_{t-1} - \left(\hat{a}^{(-t)} - \hat{a}\right)X_{t-1}\right)^2 \\
&= \sum_{t=2}^{T} \left(X_t - a_0X_{t-1}\right)^2 - \sum_{t=2}^{T} \Big(\left(X_t - \hat{a}X_{t-1}\right)^2 \\
&\qquad - 2\left(X_t - \hat{a}X_{t-1}\right)\left(\left(\hat{a}^{(-t)} - \hat{a}\right)X_{t-1}\right) \\
&\qquad + \left(\left(\hat{a}^{(-t)} - \hat{a}\right)X_{t-1}\right)^2 \Big) \\
&= -2LL_0 + 2LL_1 + C_T,
\end{aligned}$$

where

$$C_T = \sum_{t=2}^{T} \Big(-2\left(X_t - \hat{a}X_{t-1}\right)\left(\left(\hat{a}^{(-t)} - \hat{a}\right)X_{t-1}\right) + \left(\left(\hat{a}^{(-t)} - \hat{a}\right)X_{t-1}\right)^2 \Big). \tag{A.25}$$

Let us first consider the negative log-likelihood ratio $-2LL_0(X)+2LL_1(X)$. Wilk's theorem states that under the null-hypothesis,

$$-2LL_0(X) + 2LL_1(X) \sim \chi^2_1. \tag{A.26}$$

From this decomposition, we can see the from where this chi-squared distribution originates. What remains now is to compute the limit of C_T.

Decomposing C_T. Now, let us further dissect this quantity C_T. We are only interested in terms that do not vanish as T gets arbitrarily large. For C_T, that means that we can disregard all components of C_T that are of order $O(1/T)$, roughly indicating that they are upper bounded by c/T for some positive constant c and T sufficiently large.

Intuitively, we expect $\hat{a}^{(-t)}$ and $\hat{a}$ to be close to each other in value. In fact, their difference will be of the order of $O(1/T)$. Therefore, we see that the $\left(a^{(-t)} - \hat{a}\right)^2$ to be of the order $O(1/T^2)$, and therefore their sum over T will be of the order $O(1/T)$, which converges to zero as T tends to infinity. Therefore, we can disregard the last component of the sum.

Secondly, the first component of Equation A.25 can be decomposed into

$$\begin{aligned} -2\sum_{t=2}^{T}\left((X_t - \hat{a}X_{t-1})\left(\left(\hat{a}^{(-t)} - \hat{a}\right)X_{t-1}\right)\right. &= -2\sum_{t=2}^{T}\left((aX_{t-1} + \varepsilon_t - \hat{a}X_{t-1})\left(\left(\hat{a}^{(-t)} - \hat{a}\right)X_{t-1}\right)\right. \\ &= -2\sum_{t=2}^{T}\left(\left(\hat{a}^{(-t)} - \hat{a}\right)\left((a-\hat{a})\,X_{t-1}^2 + X_{t-1}\varepsilon_t\right)\right). \end{aligned} \tag{A.27}$$

We see that the first part of the summation is a product of the difference between a and $\hat{a}$ and the difference between $\hat{a}^{(-t)}$ and $\hat{a}$, both of which are approximately of the order $O(1/T)$, meaning that the product is of the order $O(1/T^2)$. Therefore, summing over these products yields a value of the order $O(1/T)$, which also converges to zero as T tends to infinity. Therefore, we can also disregard the first component of Equation A.27. What remains is the final component,

$$-2\sum_{t=2}^{T}\left(\hat{a}^{(-t)} - \hat{a}\right)X_{t-1}\varepsilon_t. \tag{A.28}$$

Now, the difference between $\hat{a}^{(-t)}$ and $\hat{a}$ is of the order $O(1/T)$, so summing difference over T should give some nonzero constant.

To slim Equation A.28 down even further, let us first remark that we can rewrite $\hat{a}^{(-t)} - \hat{a}$ as

$$\begin{aligned} \hat{a}^{(-t)} - \hat{a} &= -\frac{X_{t-1}}{\sum_{k=2}^{T} X_{k-1}^2 - X_{t-1}^2}\left(X_t - \hat{a}X_{t-1}\right) \\ &= -\frac{X_{t-1}}{\sum_{k=2}^{T} X_{k-1}^2 - X_{t-1}^2}\left((a-\hat{a})\,X_{t-1} + \varepsilon_t\right). \end{aligned}$$

Again, as $(a-\hat{a})$ is of the order of $O(1/T)$, and $1/\left(\sum_{k=2}^{T} X_{k-1}^2 - X_{t-1}^2\right)$ is also of $O(1/T)$, the product of both components will be of the order $O(1/T^2)$. Therefore,

$$\begin{aligned} \hat{a}^{(-t)} - \hat{a} &= -\frac{X_{t-1}}{\sum_{k=2}^{T} X_{k-1}^2 - X_{t-1}^2}\left((a-\hat{a})\,X_{t-1} + \varepsilon_t\right) \\ &= -\frac{X_{t-1}\varepsilon_t}{\sum_{k=2}^{T} X_{k-1}^2 - X_{t-1}^2} + O(1/T^2). \end{aligned} \tag{A.29}$$

Returning back to Equation A.28, using Equation A.29, we get that

$$-2\sum_{t=2}^{T}\left(\hat{a}^{(-t)} - \hat{a}\right)X_{t-1}\varepsilon_t = -2\sum_{t=2}^{T}\left(\frac{(X_{t-1}\varepsilon_t)^2}{X_{t-1}^2 - \sum_{k=2}^{T} X_{k-1}^2}\right) + O(1/T).$$

Let us also quickly remark that the first order component of each entry in the sum is negative, as the nominator is always positive, whereas the denominator is always negative.

To quickly summarize, we have done a careful derivation of C_T, which was given into Equation A.25. We have split C_T into two components, the part which is of order $O(1/T)$, and its leading terms, yielding the decomposition

$$C_T = \underbrace{-2 \sum_{t=1}^{T-1} \left(\frac{(X_t \varepsilon_{t+1})^2}{X_t^2 - \sum_{k=1}^{T-1} X_k^2} \right)}_{:= C_{T,\text{approx}}} + O(1/T), \tag{A.30}$$

So, it seems that $\lim_{T \to \infty} C_T = C_{T,\text{approx}}$.

Comparing C_T to $C_{T,\text{approx}}$. Let us first inspect whether the derivation from C_T to $C_{T,approx}$ was sensible. We generated data from an AR(1) model with autoregressive coefficient $a = 0.9$ and a total of $T = 10,000$ time steps. Now, the exact value for C_T from Equation A.25 was -2.0179, whereas the approximate value $C_{T,\text{approx}}$, where we have removed all components of order $O(1/T)$ after summation, was equal to -2.0182. We see that the derivation was still accurate up to three decimals, so it seems that the derivation was indeed sensible.

To see whether the distribution remains the same for C_T and $C_{T,\text{approx}}$, let us consider histograms for C_T and $C_{T,\text{approx}}$ for a fixed value of the autoregressive coefficient a and the number of time steps T. We have plotted the two histograms consisting of $N = 100{,}000$ samples of C_T and $C_{T,\text{approx}}$ for $a = 0.5$ and $T = 10{,}000$ in Figure A.3.

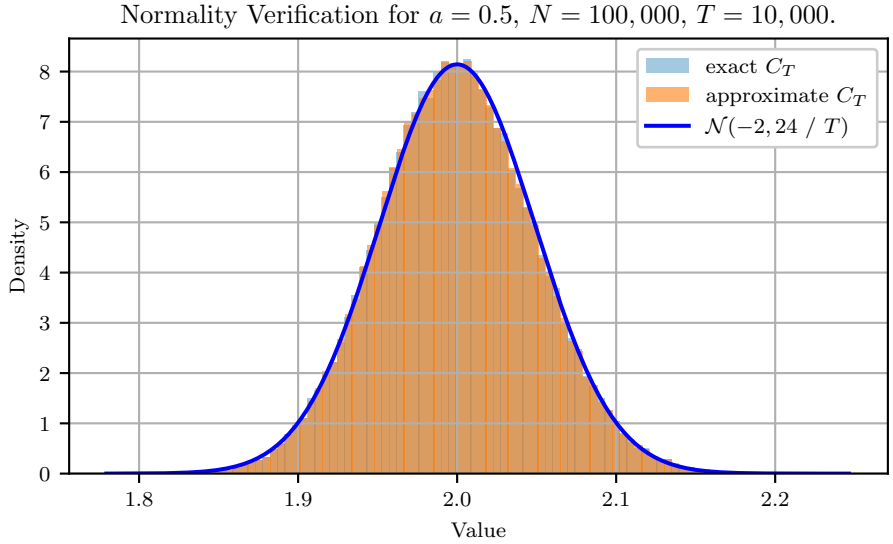


Figure A.3: Histogram visualizing the distribution of C_T and $C_{T,\text{approx}}$ as defined in Equation A.25 and Equation A.30, respectively. We see that the two histograms overlap almost perfectly. Furthermore, we have fitted a normal distribution with a mean of -2 and a variance of $-24/T$, which seems a perfect fit for both C_T and the approximate C_T.

We see that both C_T and $C_{T,\text{approx}}$ follow a normal distribution with a mean of -2 and a variance of $24/T$. Therefore, as T tends to infinity, the variance of C_T will tend to zero. In conclusion, we have that the limit of C_T is equal to

$$\lim_{T \to \infty} C_T = -2. \tag{A.31}$$

Combining this statement with the statement from Equation A.26, where we used Wilk's theorem, we have that

$$\lim_{T \to \infty} -2LL_0 + 2LL_{\text{LOOCV}} = \lim_{T \to \infty} -2LL_0 + 2LL_1 + C_T \sim \chi_1^2 - 2. \tag{A.32}$$

Appendix B

Additional tables and figures

In Chapter 7, the methods discussed in this thesis have been compared using several performance criteria. Furthermore, we have experimented with several settings. The following settings were considered:

- Sparse acyclic VAR(1) models, where the number of off-diagonal arcs was $3p$. Furthermore, we have generated data matrices consisting of few time steps $T = 100$ and many time steps $T = 1000$. The corresponding table and figure of the empirical risk are given in Section B.1.
- Dense acyclic VAR(1) models, where the number of off-diagonal arcs was $5p$. Furthermore, we have generated data matrices consisting of few time steps $T = 100$ and many time steps $T = 1000$. The three corresponding tables and figures of the empirical risk, true risk, and structural hamming distance are given in Section B.2.
- Sparse cyclic VAR(1) models, where the number of off-diagonal arcs was $2p$. Furthermore, we have generated data matrices consisting of few time steps $T = 100$ and many time steps $T = 1000$. The corresponding table and figure of the empirical risk are given in Section B.3.
- Sparse linear structural equation models, where the number of off-diagonal arcs was $2p$. Furthermore, we have generated data matrices consisting of few time steps $T = 100$ and many time steps $T = 1000$. The corresponding table and figure of the empirical risk are given in Section B.4.
- The influence of the threshold for the sparse acyclic setting has been investigated in Section B.5.

B.1 Sparse acyclic VAR(1) models

Table B.1: Average empirical risk $R_{\text{emp}}(W)$ for the aforementioned methods for several values of p and T, where $s = 3p$ and W corresponds to an acyclic structure.

	$T = 100$					$T = 1000$				
Method	$p = 5$	$p = 10$	$p = 15$	$p = 25$	$p = 50$	$p = 5$	$p = 10$	$p = 15$	$p = 25$	$p = 50$
Random Walk	4.81	10.67	17.22	29.62	57.53	4.95	11.57	18.55	34.69	76.29
MH-Regular	4.81	10.74	16.95	30.06	58.18	4.95	11.55	19.42	35.11	79.12
MH-Greedy	4.91	9.64	14.00	21.47	38.03	4.95	10.25	15.15	25.01	51.48
NOTEARS	4.81	9.18	13.46	20.89	34.79	4.95	9.87	14.79	24.55	48.57
DAG-LASSO	10.75	46.94	62.76	123.03	305.72	9.19	38.92	62.57	112.58	270.07
DAG-OMP	4.81	9.76	13.90	21.09	35.85	4.95	10.18	15.39	24.71	48.59
DAG-OLS-V	4.81	9.31	14.30	24.08	110.21	4.95	9.88	14.84	24.75	49.58

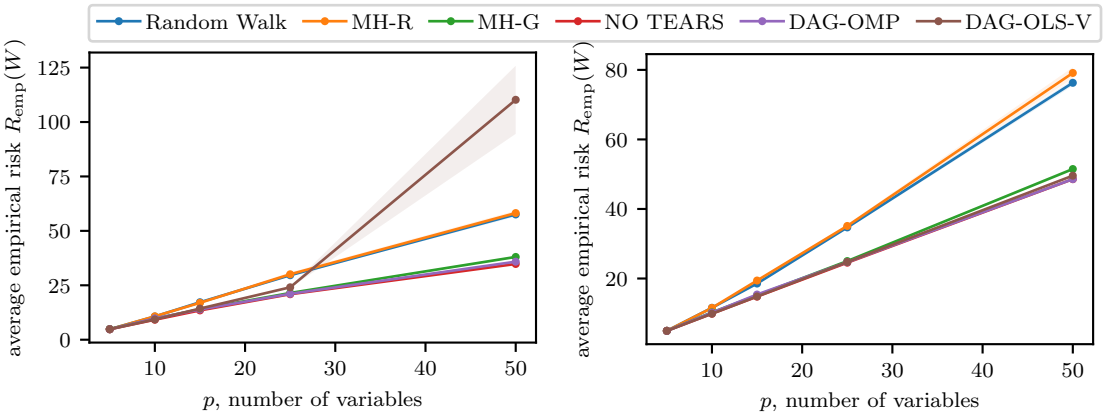


Figure B.1: Plot of the average empirical risk as a function of p, the number of variables, where $T = 100$ for the methods in Table B.1, excluding DAG-LASSO.

Figure B.2: Plot of the average empirical risk as a function of p, the number of variables, where $T = 1000$ for the methods in Table B.1, excluding DAG-LASSO.

B.2 Dense acyclic VAR(1) models

Table B.2: Average empirical risk $R_{\text{emp}}(W)$ as a function of p for $T = 100$ and $T = 1000$, where $s = 5p$ and W corresponds to an acyclic structure.

	$T = 100$					$T = 1000$				
Method	$p = 5$	$p = 10$	$p = 15$	$p = 25$	$p = 50$	$p = 5$	$p = 10$	$p = 15$	$p = 25$	$p = 50$
Random Walk	4.81	12.13	23.05	45.40	100.85	4.95	13.02	26.04	52.87	132.65
MH-Regular	4.81	12.49	23.47	45.49	103.04	4.95	13.65	26.50	53.92	145.3
MH-Greedy	4.81	9.25	14.06	23.36	49.01	4.95	9.87	15.32	27.83	64.72
NOTEARS	4.81	9.25	13.60	21.14	36.04	4.95	9.87	14.83	24.62	50.37
DAG-LASSO	10.75	65.91	332.95	565.14	3004.02	9.19	65.92	255.99	435.44	2661.91
DAG-OMP	4.81	9.63	17.41	23.75	45.55	4.95	9.94	19.37	28.36	65.94
DAG-OLS-V	4.81	9.25	13.89	24.31	212.65	4.95	9.87	14.83	24.71	49.5

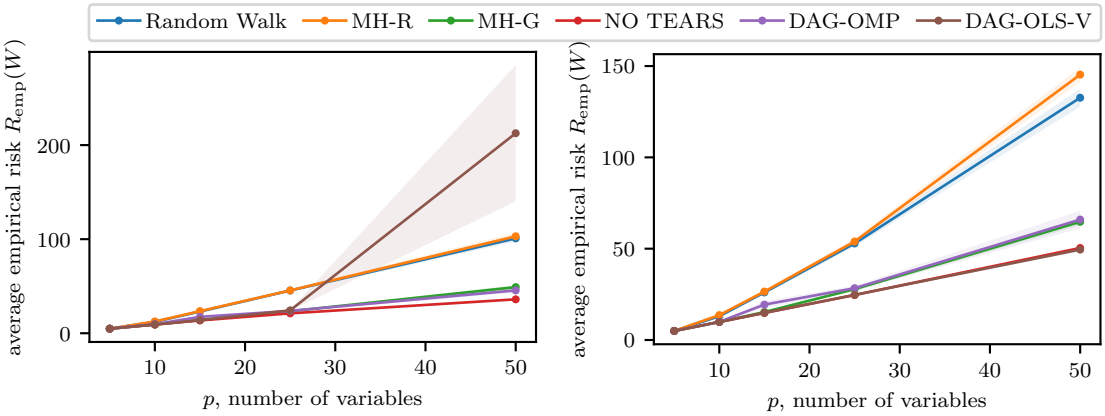


Figure B.3: Plot of the average empirical risk as a function of p, the number of variables, where $T = 100$ for the methods in Table B.2, excluding DAG-LASSO.

Figure B.4: Plot of the average empirical risk as a function of p, the number of variables, where $T = 1000$ for the methods in Table B.2, excluding DAG-LASSO.

Table B.3: Average true risk $R(W)$ as a function of p for $T = 100$ and $T = 1000$, where $s = 5p$ and W corresponds to an acyclic structure.

	$T = 100$					$T = 1000$				
Method	$p = 5$	$p = 10$	$p = 15$	$p = 25$	$p = 50$	$p = 5$	$p = 10$	$p = 15$	$p = 25$	$p = 50$
Random Walk	5.26	14.65	31.45	73.38	270.40	5.02	14.57	29.22	64.67	212.57
MH-Regular	5.26	15.30	30.46	76.33	278.67	5.02	14.58	28.75	68.29	218.29
MH-Greedy	5.26	10.74	17.13	31.30	117.50	5.02	10.06	15.81	29.46	78.15
NOTEARS	5.26	10.74	16.32	27.83	65.44	5.02	10.06	15.09	25.16	50.32
DAG-LASSO	12.32	69.26	481.20	643.96	5183.68	10.72	76.76	459.05	515.09	4645.04
DAG-OMP	5.26	11.39	23.82	39.79	158.59	5.02	10.13	20.82	31.63	100.15
DAG-OLS-V	5.28	10.74	16.32	28.8	301.54	5.02	10.06	15.09	25.16	50.32

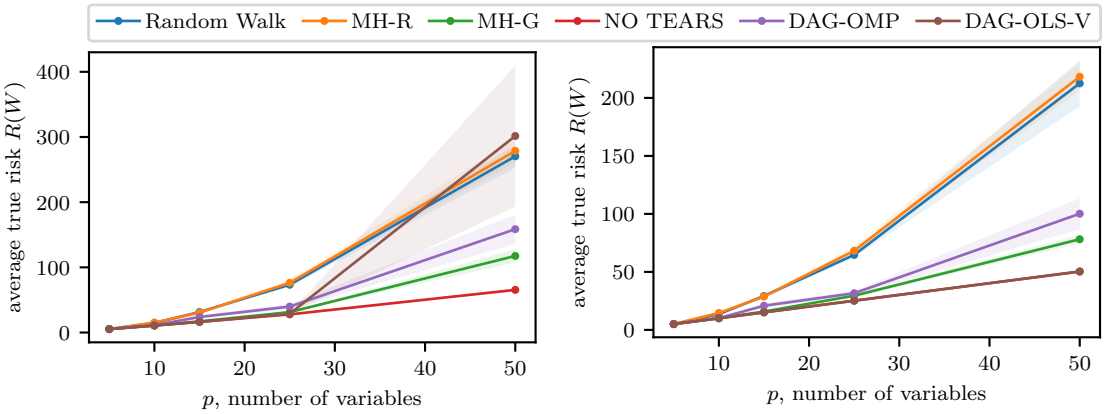


Figure B.5: Plot of the average true risk as a function of p, the number of variables, where $T = 100$ for the methods in Table B.3, excluding DAG-LASSO.

Figure B.6: Plot of the average true risk as a function of p, the number of variables, where $T = 1000$ for the methods in Table B.3, excluding DAG-LASSO.

Table B.4: Average structural hamming distance (SHD) as a function of p for $T = 100$ and $T = 1000$, where $s = 5p$ and W corresponds to an acyclic structure.

	$T = 100$					$T = 1000$				
Method	$p = 5$	$p = 10$	$p = 15$	$p = 25$	$p = 50$	$p = 5$	$p = 10$	$p = 15$	$p = 25$	$p = 50$
Random Walk	0.3	9.2	24.5	61.4	177.4	0.0	8.5	24.6	53.4	118.5
MH-Regular	0.3	10.9	22.9	61.6	183.7	0.0	9.7	22.7	50.8	135.0
MH-Greedy	0.3	0.5	2.2	9.0	67.9	0.0	0.0	1.5	7.3	29.3
NOTEARS	0.3	0.5	0.6	2.0	15.5	0.0	0.0	0.0	0.0	0.0
DAG-LASSO	12.1	50.9	86.8	145.6	294.3	9.1	51.0	85.4	143.5	291.7
DAG-OMP	0.3	1.4	6.4	7.9	62.8	0.0	0.1	4.6	5.5	18.7
DAG-OLS-V	0.4	0.5	0.6	3.1	74.9	0.0	0.0	0.0	0.0	0.0

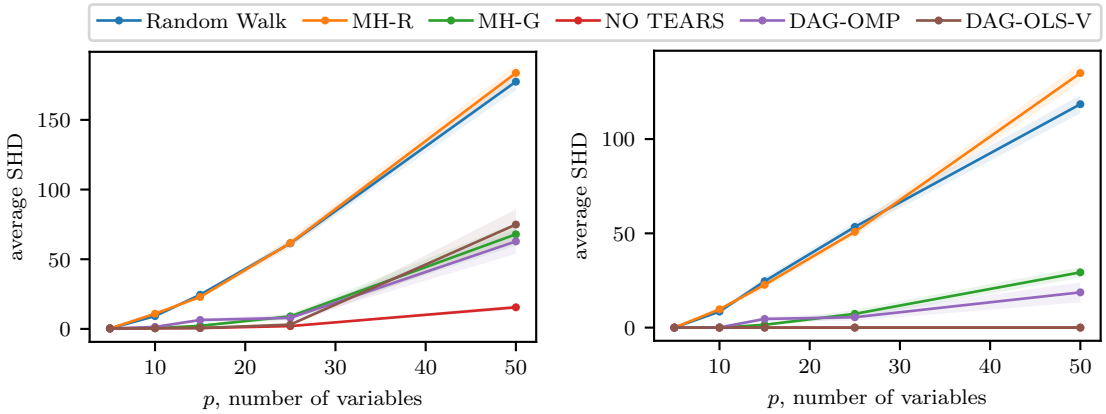


Figure B.7: Plot of the average structural hamming distance as a function of p, the number of variables, where $T = 100$ for the methods in Table B.4, excluding DAG-LASSO.

Figure B.8: Plot of the average structural hamming distance as a function of p, the number of variables, where $T = 1000$ for the methods in Table B.4, excluding DAG-LASSO.

B.3 Sparse cyclic VAR(1) models

Table B.5: Average empirical risk $R_{\text{emp}}(W)$ as a function of p for $T = 100$ and $T = 1000$, where $s = 2p$ and W corresponds to a cyclic structure.

	$T = 100$					$T = 1000$				
Method	$p = 5$	$p = 10$	$p = 15$	$p = 25$	$p = 50$	$p = 5$	$p = 10$	$p = 15$	$p = 25$	$p = 50$
Random Walk	6.48	11.90	17.42	28.92	51.43	6.49	12.54	19.02	33.04	66.88
MH-Regular	6.48	11.77	17.30	29.30	52.08	6.49	12.66	19.15	33.28	67.82
MH-Greedy	6.68	11.47	16.30	25.55	42.77	6.64	12.48	17.72	29.36	57.42
NOTEARS	6.58	11.28	15.99	24.53	40.05	6.51	12.26	17.6	29.52	54.58
DAG-LASSO	20.56	42.37	54.78	103.97	288.88	16.77	35.97	54.48	107.46	222.84
DAG-OMP	7.07	12.03	16.29	25.78	42.25	6.93	13.0	17.87	29.87	55.62
DAG-OLS-V	7.57	13.28	18.96	31.22	58.18	7.24	13.68	18.93	32.96	60.77

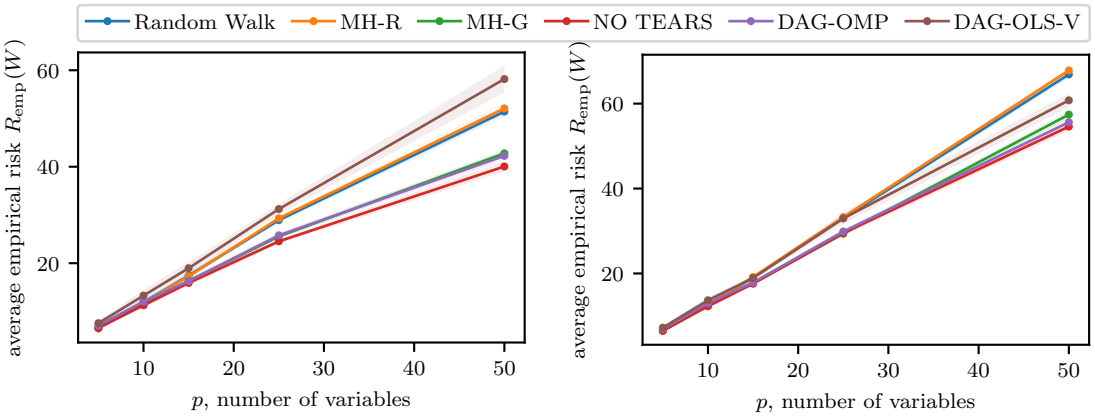


Figure B.9: Plot of the average empirical risk as a function of p, the number of variables, where $T = 100$ for the methods in Table B.5, excluding DAG-LASSO.

Figure B.10: Plot of the average empirical risk as a function of p, the number of variables, where $T = 1000$ for the methods in Table B.5, excluding DAG-LASSO.

B.4 Linear structural equation models.

Table B.6: Average empirical risk $R_{\text{emp}}(W)$ for the aforementioned methods for several values of p and T, where $s = 3p$ and the data has been generated according to a linear structural equation model.

	$T = 100$					$T = 1000$				
Method	$p = 5$	$p = 10$	$p = 15$	$p = 25$	$p = 50$	$p = 5$	$p = 10$	$p = 15$	$p = 25$	$p = 50$
Random Walk	4.97	11.60	24.44	44.08	143.06	5.02	12.30	24.20	46.22	172.68
MH-Regular	4.97	11.45	21.28	45.60	147.32	5.02	11.73	23.18	48.07	160.71
MH-Greedy	4.97	9.59	14.07	22.35	39.35	5.02	9.93	14.91	24.70	53.35
NOTEARS	5.03	17.09	395.64	136.79	2667.6	5.07	11.86	326.30	32.91	1209.85
DAG-OMP	5.65	13.39	16.62	27.65	44.74	5.87	14.12	18.24	30.16	62.24
DAG-OLS-V	6.87	23.62	58.98	175.61	1985.58	7.10	34.19	96.83	191.95	2384.91

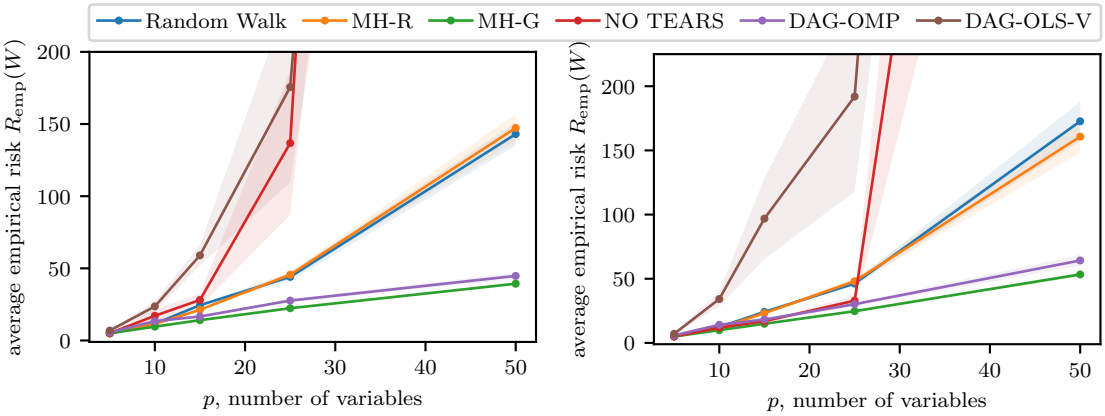


Figure B.11: Plot of the average empirical risk as a function of p, the number of variables, where $T = 100$ for the methods in Table B.6, excluding DAG-LASSO.

Figure B.12: Plot of the average empirical risk as a function of p, the number of variables, where $T = 1000$ for the methods in Table B.6, excluding DAG-LASSO.

B.5 Investigating the influence of the threshold

The experiments from Appendix B.1 have been redone for thresholds $\epsilon \in \{0.00, 0.05, 0.15, 0.30, 0.40\}$ to investigate its influence. The true risk $R(W)$ and structural hamming distance (SHD) for these threshold values are given in Table B.7 and Table B.8.

Table B.7: Average true risk as a function of p for $T = 100$ and $T = 1000$, where $s = 3p$ and W is acyclic for different thresholds ϵ.

		$T = 100$					$T = 1000$				
Threshold	Method	$p = 5$	$p = 10$	$p = 15$	$p = 25$	$p = 50$	$p = 5$	$p = 10$	$p = 15$	$p = 25$	$p = 50$
0.00	Random Walk	5.17	12.69	20.33	42.26	130.67	5.02	11.99	19.15	35.93	80.92
0.00	MH-Regular	5.17	12.87	20.41	43.98	136.06	5.02	11.80	19.73	36.46	84.11
0.00	MH-Greedy	5.17	11.38	16.98	32.18	96.01	5.02	10.29	15.18	26.1	55.39
0.00	NOTEARS	5.17	10.68	16.72	30.7	87.18	5.02	10.06	15.15	25.36	51.47
0.00	DAG-LASSO	8.47	37.04	50.79	101.90	244.59	8.24	29.30	49.07	87.45	205.30
0.00	DAG-OMP	5.17	11.34	17.46	30.84	85.33	5.02	10.39	15.78	25.55	51.65
0.00	DAG-OLS-V	5.17	10.64	16.39	28.65	63.76	5.02	10.06	15.13	25.27	50.92
0.05	Random Walk	5.17	12.65	20.24	41.62	126.33	5.02	11.97	19.12	35.84	80.44
0.05	MH-Regular	5.17	12.83	20.28	43.44	131.58	5.02	11.79	19.71	36.36	83.71
0.05	MH-Greedy	5.17	11.35	16.89	31.81	92.69	5.02	10.27	15.13	25.89	54.44
0.05	NOTEARS	5.17	10.67	16.64	30.32	84.85	5.02	10.05	15.09	25.14	50.46
0.05	DAG-LASSO	8.64	39.05	54.00	105.35	254.44	8.47	31.06	51.21	90.98	216.13
0.05	DAG-OMP	5.17	11.32	17.37	30.46	82.93	5.02	10.38	15.72	25.34	50.62
0.05	DAG-OLS-V	5.17	10.64	16.39	28.64	63.74	5.02	10.05	15.09	25.15	50.44
0.15	Random Walk	5.17	12.56	19.79	39.30	106.11	5.02	12.00	19.26	36.59	84.07
0.15	MH-Regular	5.17	12.78	19.82	41.03	109.51	5.02	11.80	19.85	37.00	87.19
0.15	MH-Greedy	5.17	11.21	16.32	29.02	70.74	5.02	10.27	15.11	25.87	54.56
0.15	NOTEARS	5.17	10.53	16.07	27.41	62.52	5.02	10.04	15.07	25.10	50.19
0.15	DAG-LASSO	9.93	45.21	60.87	117.77	293.00	8.65	35.89	56.65	102.29	245.60
0.15	DAG-OMP	5.17	11.23	16.97	27.79	63.29	5.02	10.39	15.75	25.33	50.43
0.15	DAG-OLS-V	5.17	10.52	16.05	27.26	58.18	5.02	10.04	15.07	25.10	50.19
0.30	Random Walk	5.26	12.71	20.14	40.24	109.74	5.02	12.33	19.7	37.72	88.83
0.30	MH-Regular	5.26	13.12	20.12	41.93	115.55	5.02	12.24	20.26	38.23	93.01
0.30	MH-Greedy	5.26	11.34	16.41	29.09	71.41	5.02	10.27	15.11	26.01	55.76
0.30	NOTEARS	5.26	10.59	16.08	27.08	61.03	5.02	10.04	15.07	25.10	50.19
0.30	DAG-LASSO	12.32	51.51	68.16	148.08	353.65	10.72	46.02	69.44	136.2	290.74
0.30	DAG-OMP	5.26	11.45	17.51	27.88	63.64	5.02	10.52	15.85	25.34	50.62
0.30	DAG-OLS-V	5.26	10.59	16.05	26.71	54.61	5.02	10.04	15.07	25.10	50.19
0.40	Random Walk	5.91	14.11	24.84	47.58	143.70	5.02	12.49	20.64	40.28	94.02
0.40	MH-Regular	5.91	14.38	24.39	50.55	142.37	5.02	12.57	21.18	40.30	96.72
0.40	MH-Greedy	5.91	13.23	20.53	37.17	109.48	5.02	10.33	15.11	26.31	57.72
0.40	NOTEARS	5.91	12.27	19.92	34.21	106.49	5.02	10.04	15.07	25.10	50.19
0.40	DAG-LASSO	13.79	54.4	77.76	177.18	400.56	12.04	48.88	81.05	152.41	334.92
0.40	DAG-OMP	5.91	13.48	21.05	36.13	101.94	5.02	10.61	16.11	25.62	50.68
0.40	DAG-OLS-V	5.91	12.34	19.92	33.05	73.2	5.02	10.04	15.07	25.10	50.19

For $T = 100$, the performance of all methods increases if we increase the threshold. However, for $\epsilon = 0.40$, the performance decreases as the true coefficients are also thresholded. For $T = 1000$, the threshold barely influences the iterative methods and NOTEARS. The permutation-based methods attain slightly higher true risks. For DAG-LASSO, a smaller threshold is better, but the results for DAG-LASSO remain the worst of all methods by quite a margin.

Table B.8: Average structural hamming distance (SHD) as a function of p for $T = 100$ and $T = 1000$, where $s = 3p$ and W is acyclic for different thresholds ϵ.

		$T = 100$					$T = 1000$				
Threshold	Method	$p = 5$	$p = 10$	$p = 15$	$p = 25$	$p = 50$	$p = 5$	$p = 10$	$p = 15$	$p = 25$	$p = 50$
0.00	Random Walk	0.0	23.8	74.8	263.2	1177.2	0.0	23.6	75.8	267.0	1176.0
0.00	MH-Regular	0.0	23.8	76.2	266.0	1182.6	0.0	22.2	78.8	267.4	1189.0
0.00	MH-Greedy	0.0	18.2	61.0	229.4	1095.2	0.0	16.2	60.2	227.6	1089.4
0.00	NOTEARS	0.0	14.3	52.9	204.1	992.0	0.0	9.3	42.3	149.1	699.6
0.00	DAG-LASSO	8.7	37.3	58.2	96.0	205.3	6.8	35.9	57.0	94.7	195.3
0.00	DAG-OMP	0.0	16.0	60.8	225.6	1077.8	0.0	15.4	60.8	225.4	1075.2
0.00	DAG-OLS-V	0.0	7.4	18.1	56.3	169.5	0.0	5.8	20.9	60.0	226.0
0.05	Random Walk	0.0	14.4	43.0	155.8	755.0	0.0	6.0	19.9	62.0	215.8
0.05	MH-Regular	0.0	14.6	42.8	162.0	774.4	0.0	6.2	23.2	60.7	220.7
0.05	MH-Greedy	0.0	11.6	33.8	137.9	691.8	0.0	1.7	4.5	12.7	82.5
0.05	NOTEARS	0.0	9.7	33.0	135.8	700.5	0.0	0.9	4.3	9.5	61.1
0.05	DAG-LASSO	9.0	37.1	55.5	93.8	192.2	7.1	34.9	54.6	89.9	184.3
0.05	DAG-OMP	0.0	9.9	33.9	132.4	674.9	0.0	1.2	6.8	11.6	61.4
0.05	DAG-OLS-V	0.0	7.3	17.4	54.4	168.2	0.0	1.0	3.8	10.3	48.9
0.15	Random Walk	0.0	6.9	16.3	52.1	273.0	0.0	4.8	11.0	29.0	75.5
0.15	MH-Regular	0.0	6.7	14.8	56.4	282.3	0.0	4.9	12.7	28.7	80.2
0.15	MH-Greedy	0.0	3.0	7.4	27.7	204.1	0.0	0.6	0.2	1.9	11.9
0.15	NOTEARS	0.0	1.2	6.2	26.3	191.3	0.0	0.0	0.0	0.0	0.0
0.15	DAG-LASSO	10.2	36.9	54.3	91.5	186.9	7.1	34.9	54.0	89.1	180.9
0.15	DAG-OMP	0.0	2.4	7.2	25.3	172.5	0.0	0.4	1.1	0.5	0.8
0.15	DAG-OLS-V	0.0	0.8	5.5	19.2	94.6	0.0	0.0	0.0	0.0	0.0
0.30	Random Walk	0.3	5.6	9.2	27.2	97.3	0.0	4.8	9.4	22.9	55.4
0.30	MH-Regular	0.3	5.9	9.9	28.7	106.9	0.0	4.2	11.0	23.9	59.8
0.30	MH-Greedy	0.3	2.3	1.3	5.6	39.3	0.0	0.6	0.1	1.6	8.4
0.30	NOTEARS	0.3	0.4	0.7	1.9	20.0	0.0	0.0	0.0	0.0	0.0
0.30	DAG-LASSO	12.1	37.2	54.5	92.8	188.1	9.1	36.1	55.0	91.9	184.6
0.30	DAG-OMP	0.3	1.6	1.4	3.6	22.3	0.0	0.6	0.9	0.4	0.3
0.30	DAG-OLS-V	0.3	0.4	0.6	1.3	6.0	0.0	0.0	0.0	0.0	0.0
0.40	Random Walk	2.3	8.5	15.6	33.1	96.9	0.0	5.6	10.6	25.7	59.0
0.40	MH-Regular	2.3	8.3	15.3	36.5	101.9	0.0	5.6	12.0	27.2	64.4
0.40	MH-Greedy	2.3	5.7	7.5	14.2	50.6	0.0	0.8	0.1	1.9	9.4
0.40	NOTEARS	2.3	3.4	6.3	11.0	37.2	0.0	0.0	0.0	0.0	0.0
0.40	DAG-LASSO	13.3	37.6	55.7	95.1	190.6	12.0	36.7	55.9	93.4	188.0
0.40	DAG-OMP	2.3	4.6	7.3	13.4	39.9	0.0	0.8	1.1	0.5	0.2
0.40	DAG-OLS-V	2.3	3.5	6.3	9.9	23.2	0.0	0.0	0.0	0.0	0.0

For $T = 100$, having no threshold is detrimental for all methods, although DAG-LASSO seems to perform better structurally, although the predictive performance of DAG-LASSO was quite poor. Increasing the threshold significantly improves the structural performance of all methods, except DAG-LASSO whose performance remains approximately the same. However, the performance drops for $\epsilon = 0.40$, most likely because too many coefficients are thresholded.

For $T = 1000$, when we have no threshold, the SHD of all methods is poor, especially the permutation-based methods and DAG-OMP. A threshold of 0.05 slightly improves the SHD, and for larger threshold values, the SHD does not seem to change much.

The value of the threshold drastically affects the structural performance of all methods, yet their relative performance, that is, their performance compared to each other, remains similar.