TU/e 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 0.4

Eindhoven, June 2022

Abstract

This is the abstract.

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

Preface

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

First and foremost, I would like to acknowledge 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 in this journey, Rui helped me in either getting over these hurdles, or circumvent 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 the necessary distraction next to my academic endeavours. Our pleasant social interactions provided a nice balance between my social life and academic studies, not just during my graduation project, but throughout all six years that I have spent here at the Eindhoven University of Technology.

Contents

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, the linear time-dependent graphical model from Definition 2.3. 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.

The ML estimate is relevant for the whole chapter. Make a small intro for the chapter out of it

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 the 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

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

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

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

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

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

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,

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

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 proportional to

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

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,

$$\hat{W} = \arg\max \mathbb{P}_W\left(\mathbf{X}\right) = \arg\min -\log\left(\mathbb{P}_W\left(\mathbf{X}\right)\right). \tag{4.7}$$

Now, taking the logarithm of Equation 4.5 yields

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

where $A \propto B$ means that A is proportional to B, in the sense that the matrix $\hat{W}$ that minimizes A is the same matrix as the one that minimizes B. 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-norms of the difference between $X_{t,\cdot}$ and $X_{t-1,\cdot}W$ for $t = 2,\ldots,T$. Therefore, we know that the maximum likelihood estimator $\hat{W}$ coincides with the *ordinary least squares* estimator, which is given by

$$\hat{W} = \left(XX^T\right)^{-1}X^TY, \tag{4.9}$$

where $X \in \mathbb{R}^{T-1\times p}$ represents the first $T-1$ time steps of $\mathbf{X}$, and $Y \in \mathbb{R}^{T-1\times p}$ represents the last $T-1$ time steps of $\mathbf{X}$.

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 lest squares estimation technique, is also one the most common technique for estimating the coefficient matrix of a VAR(1) model [18].

However, such a maximum likelihood estimate does not take acyclicity into account. Therefore, we see that such ~~a maximum likelihood approach~~ does not comply with our acyclicity constraint. 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 i a unique index $\pi(i)$ in the ordering. 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,

$$\mathcal{G} \text{ is acyclic} \iff X_i \text{ precedes } X_j \text{ in } \pi \text{ for all arcs from } X_i \text{ to } X_j. \tag{4.10}$$

sounds like there are multiple arcs from X_i to X_j

Having such a permutation π of the variables is equivalent to having an acyclic structure $\mathcal{G}$ of the variables.

Equivalence does here not mean 1 to 1 correspondence. Make sure to explain somewhere properly the relation between a permutation and a DAG. This comes only (partially) later with part about the size of the search space. Make clear: Given a DAG -> we get a class of permutation matrices that can represent that DAG

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

$$\pi = (3, 1, 2) \iff P = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0. \end{pmatrix}. \tag{4.11}$$

Given a permutation matrix -> We still have to decide which of the possible edges do actually exist

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

$$X_{t,\cdot} = X_{t-1,\cdot}W + \varepsilon_t, \qquad \varepsilon_t \sim \mathcal{N}\left(\mathbf{0}, I_p\right). \tag{4.12}$$

The model defined by Equation 4.12 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 a lower triangular matrix T such that

$$W = P^T U P. \tag{4.13}$$

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 $X_i \rightarrow X_j$ 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 $u_{ij} > 0$ if and only if $i \geq j$, we have that $u_{ij} > 0$ if and 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. Interestingly, 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. Recall that the negative log-likelihood was proportional to

$$-\log\left(\mathbb{P}_W\left(\mathbf{X}\right)\right) \propto \sum_{t=2}^{T} \left\|X_{t,\cdot} - X_{t-1,\cdot}W\right\|_2^2. \tag{4.14}$$

Now, for more concise notation, let $X \in \mathbb{R}^{T-1\times p}$ represent the first $T-1$ time lags of $\mathbf{X}$ and $Y \in \mathbb{R}^{T-1\times p}$ represent the last $T-1$ time lags of $\mathbf{X}$. Then, corresponding maximum likelihood estimator of W is

$$\hat{W} = \left(XX^T\right)^{-1} X^T Y, \tag{4.15}$$

which coincides with the *ordinary least squares* estimate of W.

XP^T?

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

$$-\log\left(\mathbb{P}_U\left(\mathbf{X};P\right)\right) \propto \sum_{t=2}^{T} \left\| X_{t,\cdot} - X_{t-1,\cdot}P^T U P \right\|_2^2, \tag{4.16}$$

which is the same as

$$-\log\left(\mathbb{P}_U\left(\mathbf{X};P\right)\right) \propto \sum_{t=2}^{T} \left\| P X_{t,\cdot} - P X_{t-1,\cdot} U \right\|_2^2, \tag{4.17}$$

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

$$\hat{U} = \left(X_P X_P^T\right)^{-1} X_P^T Y_P, \tag{4.18}$$

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

$$\hat{U}_{\cdot,1:i} = \left(X_{P,\cdot,1:i} X_{P,\cdot,1:i}^T\right)^{-1} X_{P,\cdot,1:i}^T Y_{P,\cdot,i}. \tag{4.19}$$

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`.

In less mathematical terms, we see that 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: Upper triangular matrix U 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 $X$
$U \leftarrow \mathbf{O}_{p\times p}$                                  ▷ Initialize $U$ as $p \times p$ zero matrix

$X \leftarrow \mathbf{X}_P[1 : T-1, \cdot]$
$Y \leftarrow \mathbf{X}_P[2 : T, \cdot]$

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

$W \leftarrow P^T U P$                                   ▷ Transpose rows and columns of $A$
return $W$
```

A step-by-step three dimensional example showcasing the inner workings of `Order-OLS` is given in Example 4.1.

Example 4.1 `Order-OLS` with three variables.

Let us give an example of how `Order-OLS` works. Suppose that we have three variables, and we are interested in finding the coefficient matrix that maximizes the likelihood given our data matrix $\mathbf{X}$, while still respecting some permutation matrix P. Suppose we are given an ordering $\pi = (X_3, X_1, X_2)$, corresponding to the permutation matrix

$$P = \begin{pmatrix} e_3 \\ e_1 \\ e_2 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}. \tag{4.20}$$

Let us now consider the `Order-OLS` algorithm. To estimate the first column of U, we have that $X_{P,\cdot,1}$ 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 consists of only one estimated parameter, u_{11}, for which the maximum likelihood estimator is

$$u_{11} = \left(X_{P,\cdot,1} X_{P,\cdot,1}^T\right)^{-1} X_{P,\cdot,1}^T Y_{P,\cdot,1}. \tag{4.21}$$

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

$$\begin{pmatrix} u_{12} \\ u_{22} \end{pmatrix} = \left(X_{P,\cdot,2} X_{P,\cdot,2}^T\right)^{-1} X_{P,\cdot,2}^T Y_{P,\cdot,2}, \tag{4.22}$$

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

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

$$\begin{pmatrix} u_{13} \\ u_{23} \\ u_{33} \end{pmatrix} = \left(X_{P,\cdot,3} X_{P,\cdot,3}^T\right)^{-1} X_{P,\cdot,3}^T Y_{P,\cdot,3}, \tag{4.23}$$

where $X_{P,\cdot,3} = (\mathbf{X}_{1:T-1,3}, \mathbf{X}_{1:T-1,1}, \mathbf{X}_{1:T-1,2})$, 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 back from the permutation $\pi = (3, 1, 2)$ to the standard ordering $(1, 2, 3)$, which is done by

$$W = P^T U P. \tag{4.24}$$

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 4.14.

Exhaustive Search. Now, we have found a way to find the optimal U given a permutation matrix P. What remains is to 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. Having tried all permutation matrices, we can simply select the permutation matrix P that provides the best fitting matrix W. We define the best fitting matrix as the matrix that maximizes the likelihood, or equivalently, that minimizes the mean squared error, where the mean squared error corresponding to a coefficient matrix W is defined as

$$\texttt{MSE}(W) = \frac{1}{T-1} \sum_{t=2}^{T} \|X_{t,\cdot} - X_{t-1,\cdot} W\|_2^2. \tag{4.25}$$

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,

$$W^* = \underset{W \in \mathbb{R}^{p \times p}}{\arg\max}\ \mathbb{P}_W(\mathbf{X}) \text{ such that } G(W) \in \texttt{DAGs}.$$

```
 1: W_best ← $I$
 2: MSE_best ← $\infty$
 3:
 4: for every permutation matrix $P$ do
 5:     $W$ ← Order-OLS($\mathbf{X}, P$)
 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

$$G(p) = \sum_{k=1}^{n} (-1)^{k+1} \binom{p}{k} 2^{k(p-k)} G(p-k), \tag{4.26}$$

which was first given by Robinson in [29]. 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 provide 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.

An upside of the exhaustive search method is that the returned matrix W is guaranteed to be *optimal*. 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 optimal matrix, at the cost of having an exponential running time.

in which sense
optimal

Sometimes you say compatible (which I like better)

Number of suitable permutations. An interesting remark is that sometimes we do not need to do an exhaustive search over all permutation matrices to find the optimal solution, especially when the true matrix W is sparse. In fact, any permutation that is compatible with the graph $G(W)$ would be able to estimate all non-zero coefficients in W. If for example, the fourth and the fifth variable do not depend on each other, so $W_{4,5} = W_{5,4} = 0$, 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 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, it is clear that 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. First of all, our third variable only depends on itself, so this variable can be at any index in the permutation π and our matrix W will still be upper triangular. This already increases the number of appropriate orderings from just 1 to 5. Furthermore, since the fourth and fifth variable do not depend on each other, we can interchange them in which way we like. 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. In total, there are a total of 20 permutation matrices P such that $W = P^T U P$ for some upper triangular matrix U.

This means that out of the $5! = 120$ permutations, 20 permutations are suitable for estimating all true coefficients. Therefore, we often do not need to check *all* possible permutations, especially when the coefficient matrix W is sparse.

In Example 4.2, we have seen that there often are quite some possible permutation matrices P to properly estimate our matrix W. Especially when the matrix W is sparse, there are quite some permutation matrices possible.

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 [5], 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 only the preceding variable, we cannot have any 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.

Having explained some interesting aspects of the exhaustive search, let us 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

$$X_{t,\cdot} = X_{t,\cdot}W + \varepsilon_t, \tag{4.27}$$

where

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

We define $X_{1,\cdot} = \varepsilon_1$ so that $X_{t,\cdot}, t \geq 2$ follows from Equation 4.27. We simulate for a total of $T = 100$ timesteps. The corresponding time series have been visualized in Figure 4.1.

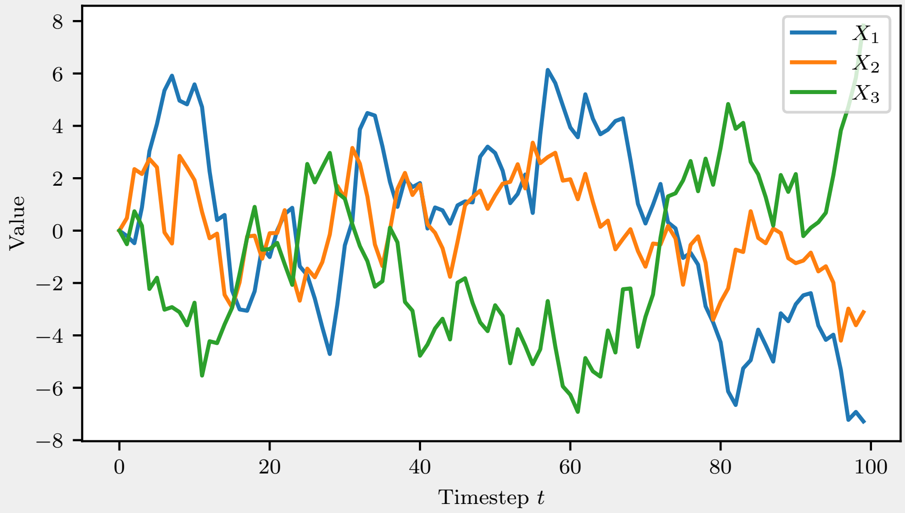

Figure 4.1: Visualisation of the three variables X_1, X_2, X_3 of Example 4.3.

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 $w_{3,1}$ 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 some suitable approaches 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.

Final Remarks. We see that, although the problem we are tackling is NP-hard, the problem becomes quite simple when a suitable ordering or permutation matrix P is provided. We have described an algorithm, `Order-OLS`, that finds the coefficient matrix W that maximizes the likelihood of $\mathbf{X}$, while respecting the ordering induced by P.

Unfortunately, finding a suitable permutation is difficult. Using an exhaustive search, we can iterate over all permutations and we are guaranteed to find the matrix W that maximizes the likelihood function, while enforcing $G(W)$ to be acyclic. Iterating over all possible permutations of our variables unfortunately implies that the running time is exponential with respect to the number of variables p, namely $\mathcal{O}(p!)$. Therefore, we cannot expect to find use this method for more than a dozen of variables.

There are interesting techniques to improve the running time of the exhaustive search. For example, 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, yet thereby sacrificing the quality of the returned matrix W.

General: Combine 4.2 and 4.3. Many of the things you almost introduce twice.
This should be possible to explain in a much more compact format.

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 $\mathcal{P}_{perm}$. 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,

$$\mathbb{P}\left(P_{i+1} = P' \mid P_i = P\right). \tag{4.28}$$

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

$$\mathbb{P}\left(P_{i+1} = P' \mid P_i = P\right) = \frac{1}{p!} \text{ for all permutation matrices } P' \in \mathcal{P}_{perm}. \tag{4.29}$$

where $\mathcal{P}_{perm}$ is the set of all possible permutation matrices on p variables. Note that the conditioning in Equation 4.29 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 $\mathcal{P}_{adj(P)}$, the set of permutation matrices that can be reached by swapping just two variables. In mathematical notation, we have that

$$\mathcal{P}_{adj(P)} = \{P' \mid \text{swap row } i \text{ and } j \text{ of } P,\ i = 1, \dots p-1,\ j = i+1, \dots, p.\}. \tag{4.30}$$

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

$$\mathbb{P}\left(P_{i+1} = P' \mid P_i = P\right) = \begin{cases} \frac{2}{p(p+1)} & \text{for all permutation matrices } P' \in \mathcal{P}_{adj(P)}, \\ 0 & \text{otherwise.} \end{cases}. \tag{4.31}$$

where $\mathcal{P}_{adj(P)}$ 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 4.29, 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 next permutation matrix.

For the second case, given in Equation 4.31, 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 $\mathbb{P}(P_{i+1} \mid P_i)$. For the first version, the completely random walk, we use Equation 4.29 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 4.30 and 4.31.

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 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. To further explain the inner workings of the random walk algorithm described in Algorithm 4.3, let us consider an example. For the exhaustive algorithm, we used a three-dimensional example such that we could easily iterate over all permutations. The random walk, however, was designed for scenarios where iterating over all possible permutation matrices was too time consuming. Therefore, we will consider a higher-dimensional example where we have 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 a VAR(1) model as defined in Definition 2.3 on ten variables and one hundred samples, resulting in a data matrix $\mathbf{X} \in \mathbb{R}^{100\times 10}$. We have constructed a moderately sparse coefficient matrix W, which is acyclic by construction and has 25 off-diagonal non-zero coefficients with value 0.5. Furthermore, each of the p time-series has quite a strong autoregressive coefficient, as we generate the diagonal elements by selecting a value in the range $[0.6, 0.8]$ uniformly at random. The corresponding time series is visualized in Figure 4.2.

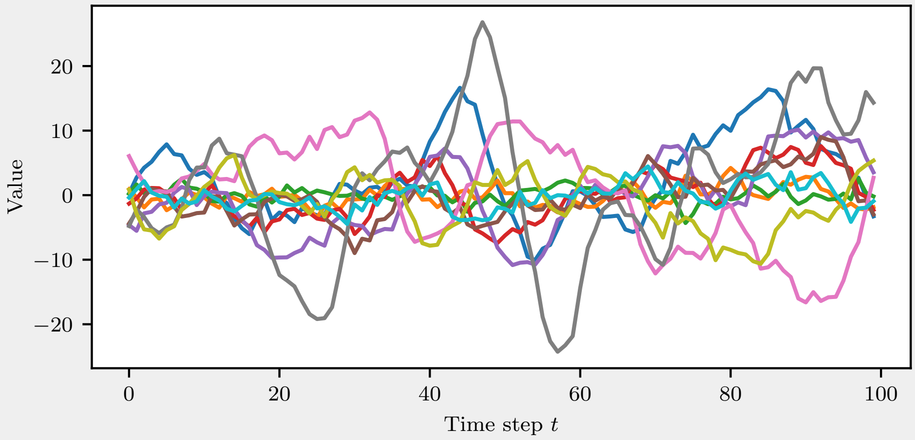

Figure 4.2: Visualisation of the ten time series corresponding to the variables X_1, ..., X_{10} of Example 4.4.

Let us consider exploring approximately just 1% of the search space of permutation matrices 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. For comparison, we will also run the exhaustive algorithm to obtain the acyclic matrix W^* that maximizes the likelihood, or equivalently minimizes the mean squared error. We will use both methods of sampling the next permutation matrix, and the corresponding acyclic matrices W_1 and W_2 that have been returned achieve a mean squared error of

$$\mathtt{MSE}(W^*) = 9.426, \qquad \mathtt{MSE}(W_1) = 10.567, \qquad \mathtt{MSE}(W_2) = 10.245. \tag{4.32}$$

In conclusion, we see that both random walks achieve similar performance. Furthermore, both performances are quite close to the performance of the exhaustive search, despite having only explored roughly 1% of the search space.

Final Remarks. In this section, we have continued upon the exhaustive algorithm from Section 4.1, where all permutation matrices were considered. As this is intractable for even a dozen number of variables, we have devised a random walk algorithm where we randomly explore the search space for a limited amount of time and return the best solution found in this time span. We have devised two approaches to explore the search space of permutation matrices; one where we simply select any permutation matrix uniformly at random, and one where we select a permutation matrix that is just one transposition away from our current permutation matrix. In Example 4.4, we have seen that this method finds a suitable acyclic matrix W that achieves a likelihood almost as high as the exhaustive approach, whereas we have only randomly explored 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 the suitability of this permutation. We can improve on this blind transitioning 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 such an approach, which is also called the Metropolis-Hastings approach. The approach was named after N. Metropolis, who was one of the authors that introduced this method in 1953 [24]. The method was further investigated by W. K. Hastings in 1970 [16], from which the conjunction Metropolis-Hastings 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 quite successful in efficiently traversing a complex search space. For example, the Metropolish-Hastings algorithm worked quite well in [7].

This is some of things you can remove when you merge 4.2 and 4.3

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', given that we are now considering permutation matrix P, solely depends on P and not on how well the permutation matrix P' fits the data $\mathbf{X}$. In this section, we will consider the Metropolis-Hastings algorithm, 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,

$$q_{P,P'} = \mathbb{P}\left(P_{i+1} = P' \mid P_i = P\right) = \begin{cases} \frac{2}{p(p+1)} & \text{for all permutation matrices } P' \in \mathcal{P}_{adj(P)}, \\ 0 & \text{otherwise.} \end{cases} . \tag{4.33}$$

where $\mathcal{P}_{adj(P)}$ 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 $q_{P,P'} = \mathbb{P}\left(P_{i+1} = P' \mid P_i = P\right)$. 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 slightly worse 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

$$\mathcal{L}\left(\mathbf{X} \mid P\right) = \prod_{t=2}^{T} \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2} \cdot \left(X_{t,\cdot} - X_{t-1,\cdot} W_P\right)^T \left(X_{t,\cdot} - X_{t-1,\cdot} W_P\right)\right), \tag{4.34}$$

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

We will 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 much smaller than $\mathcal{L}(\mathbf{X} \mid P)$, then P' fits the data much 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

$$\alpha_{P,P'} = \begin{cases} \min\left\{\frac{\mathcal{L}(\mathbf{X}|P')}{\mathcal{L}(\mathbf{X}|P\ )}, 1\right\} & \text{if } \mathcal{L}(\mathbf{X} \mid P) > 0 \\ 1 & \text{otherwise.} \end{cases} \tag{4.35}$$

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 as the probability is now some value between zero and one, 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 4.25 as a surrogate for the likelihood. As the mean squared error of W on $\mathbf{X}$ is proportional to the likelihood of W given $\mathbf{X}$, we argue that this transformation is not problematic. However, as a lower mean squared error implies a larger likelihood, we interchange the nominator and the denominator ratio such that our proposed probability of acceptance is

$$\alpha_{P,P'} = \begin{cases} \min\left\{\frac{\texttt{MSE}(W_P)}{\texttt{MSE}(W_{P'})}, 1\right\} & \text{if } \texttt{MSE}(W_{P'}) > 0 \\ 1 & \text{otherwise.} \end{cases} \tag{4.36}$$

Just as in Equation 4.35, 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 both mean squared errors.

Metropolis-Hasting transition probabilities. Given the probability of selecting P' as the next possible permutation matrix, and the probability of consequently accepting the P' over the current permutation matrix P, we can define the actual transition probability for our Metropolis-Hastings algorithm. To transition from P' to P, the following two events are required.

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 4.33. Secondly, we evaluate how well P' fits $\mathbf{X}$ compared to P by computing the mean squarer error of $\mathbf{X}$ when using the coefficient W_P and $W_{P'}$. Based on the fraction, we compute the probability of acceptance according to Equation 4.42. Consequently, the probability of actually transitioning to P' is now given by the new transition probabilities

$$\mathbb{P}(P_{i+1} = P' \mid P_i = P) = q_{P,P'} \cdot \alpha_{P,P'} \quad \text{for } P \neq P'. \tag{4.37}$$

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

$$\mathbb{P}(P_{i+1} = P \mid P_i = P) = 1 - \sum_{P \in \mathcal{P}_{adj(P)}} q_{P,P'} \cdot \alpha_{P,P'}. \tag{4.38}$$

The Metropolis-Hastings algorithm is a random walk using the sampling distribution given in Equations 4.37 and 4.38. This random walk can remain in the same state P for multiple time steps if the considered permutation matrices P' poorly fit $\mathbf{X}$. 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 that fits $\mathbf{X}$ well.

```
 1: $P \leftarrow P_0$
 2: $W \leftarrow$ Order-OLS$(\mathbf{X}, P)$
 3: W_best $\leftarrow W$
 4: MSE_best $\leftarrow$ MSE(W_best)
 5:
 6: while the stopping criterion is not met do
 7:     $P' \leftarrow$ P_next$(P)$
 8:     $W' \leftarrow$ Order-OLS$(\mathbf{X}, P)$
 9:     $\alpha \leftarrow \min \{$MSE$(W)$ / MSE$(W'), 1\}$
10:     $P \leftarrow P'$ with probability $\alpha$.
11:     $W \leftarrow$ Order-OLS$(\mathbf{X}, P)$.
12:     if MSE$(W) <$ MSE_best then
13:         MSE_best $\leftarrow$ MSE$(W)$
14:         W_best $\leftarrow W$
15:     end if
16: end while
17:
18: return W_best
```

Make concise explanations next to the lines

In the first four lines, we will initialize our permutation matrix P and coefficient matrix W. Furthermore, we will use these as our best W with highest likelihood so far. Now, while the stopping criterion is not met, we will continue sampling new permutation matrices P' and evaluate them. In line seven through ten, we will sample the new P from our new Markov chain using the transition probabilities given in Equation 4.37 and 4.38. In line eleven through fifteen, we will evaluate our next permutation matrix.

Lastly, once the stopping criterion is met, we will return the best matrix W we have found. We can use the same stopping criteria as for the random walk given in Section 4.2. We can put an *iteration limit* N on the number of loop evaluations allowed. This means that after N samples from the Markov Chain, we will terminate with the best solution that we have found, regardless of how good it is.

A similar stopping criterion is to define a *time limit* $t_{\max}$. That is, we keep sampling permutation matrices until the total time that has elapsed during the while loop has exceeded $t_{\max}$ 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.

Examples. To see whether the Metropolis-Hastings is indeed a significant improvement over the regular random walk discussed in Section 4.2, we will consider two examples here. First, we will discuss the Metropolis-Hastings algorithm where our data generating matrix W is dense. For this, we will use the same ten-dimensional data matrix on 100 samples as in Example 4.4. In Example 4.6, we will use a data generating matrix that is sparser.

Example 4.5 Random Walk on ten variables with a dense coefficient matrix W.

Let us consider an example where we have ten time series of one hundred time steps, so we have a data matrix $\mathbf{X} \in \mathbb{R}^{100\times 10}$. The coefficient matrix W consists of positive diagonal elements, and a total of 25 off-diagonal elements, giving us a total of 35 edges. The matrix W has been created such that $G(W)$ correspond to a directed acyclic graph. 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, even as few as $0.01\% = 360$ permutations this time.

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 such that W is compatible with the ordering by counting the number of permutation matrices P such that

$$P^T W P \text{ is upper triangular.} \tag{4.39}$$

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 during the 360 iterations. The mean squared error corresponding to the best permutation found so far has been plotted as a function of the iteration number i in Figure 4.3.

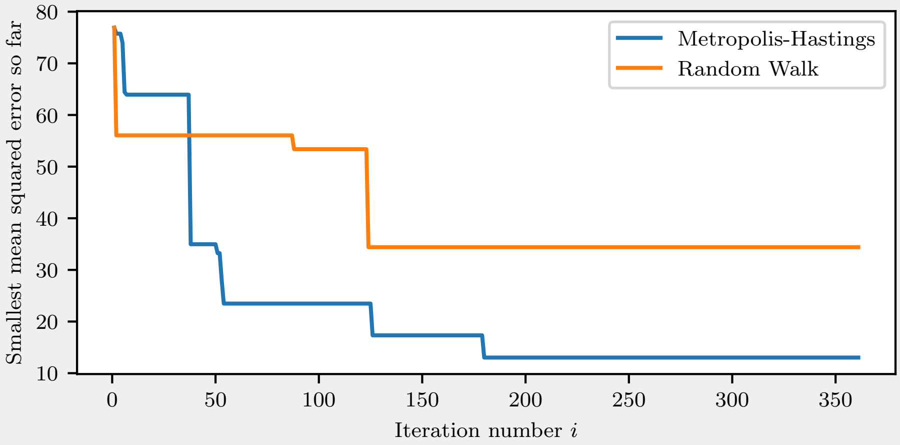

Figure 4.3: Trajectory of the lowest likelihood found so far for the Metropolis-Hastings algorithm and the Random Walk.

We see that the initial choice of the identity matrix is not a sensible choice, as the mean squared error was quite high. Therefore, we see that we improve quickly and often in the first fifty iterations. Afterwards, we see that the number of iterations between improvements increases, and also the difference in likelihood 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 in less iterations.

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 4.5.

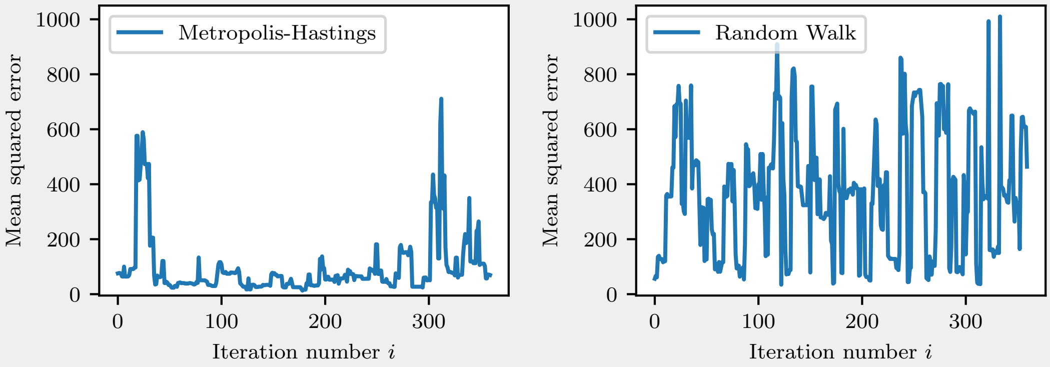

Figure 4.5: The mean squared error corresponding to the permutation matrix at each iteration number i for the Metropolis-Hastings algorithm and the Random Walk algorithm.

We see that the Metropolis-Hastings has far fewer spikes, which indicate that the Metropolis-Hastings algorithm selects permutation matrices that achieve higher likelihoods or equivalently, lower mean squared errors. 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.

Inspecting Figure 4.5, we see that random walk selects a much poorer fitting permutation matrix than the Metropolis-Hastings algorithm. If we were to take the mean of all mean squared errors in Figure 4.5, then the average permutation matrix picked by the Metropolis Hastings algorithm achieves a mean squared error of approximately 94, whereas the mean for the Random Walk algorithm is approximately 330. Let us also compare the medians, as the median is more robust against skewed data. In a same manner, the median for the Metropolis-Hastings algorithm is approximately 64, whereas the median for the random walk algorithm is approximately 260.

We see what we would indeed expect, the Metropolis-Hastings algorithm chooses candidates with a better likelihood. In fact, the likelihood as about two and a half times as small. As the Metropolis-Hastings algorithm indeed bases its probability of acceptance on the likelihood, we would not have expected anything else. Nevertheless, it is good to have a clear example of what this transition probability does in practice.

Density of the coefficient matrix W. As we can see in Example 4.5, the Metropolis-Hastings seems to be 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. One of the reasons why the Metropolis-Hastings algorithm works well here is the density of the coefficient matrix W. As W is quite dense, meaning that W has a large number of edges, the signal to noise ratio in $\mathbf{X}$ is quite high. This in turns means that trying a less suitable permutation matrix will result in a significantly smaller likelihood. The Metropolis-Hastings algorithm is able to utilize this, and therefore it 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, what happens if the original coefficient matrix W is sparse, meaning it has fewer edges? In such a case, the signal to noise ratio is much smaller. Therefore, the difference between the best and worst permutation matrix is also smaller. Therefore, the acceptance probability of a relatively less suitable matrix will be larger. To see how the likelihoods compare, we can compare Figure and Figure, corresponding to the likelihoods of the Metropolis-Hastings algorithm for a dense and a sparse coefficient matrix W, respectively. 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, meaning 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 4.6.

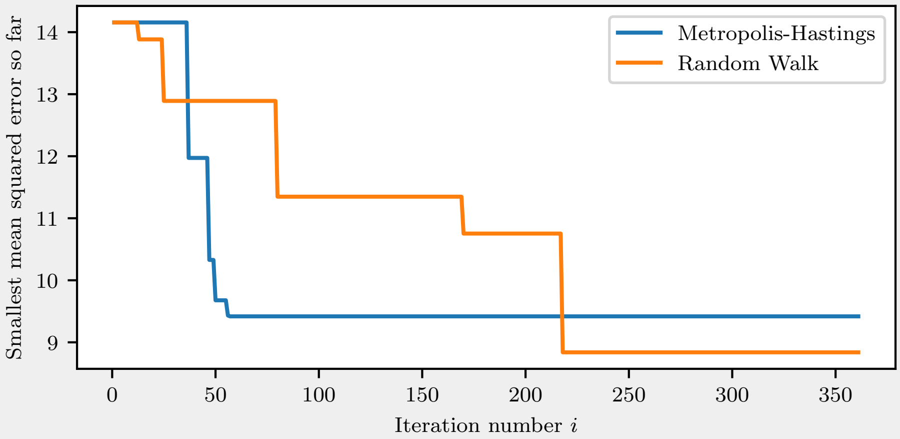

Figure 4.6: Trajectory of the smallest mean squared error found so far for the Metropolis-Hastings algorithm and the Random Walk.

We see that the performance of the Metropolis-Hastings algorithm is not much better than the random walk anymore. The reason for this is that due to the much sparser coefficient matrix W, a poor choice for a permutation matrix does not result in a very poor likelihood due to a lower signal-to-noise ratio. To inspect this hypothesis further, let us consider the mean squared errors corresponding to the permutation matrices of both the Metropolis-Hastings and the random walk algorithm at each iteration. Both plots are given in Figure 4.8.

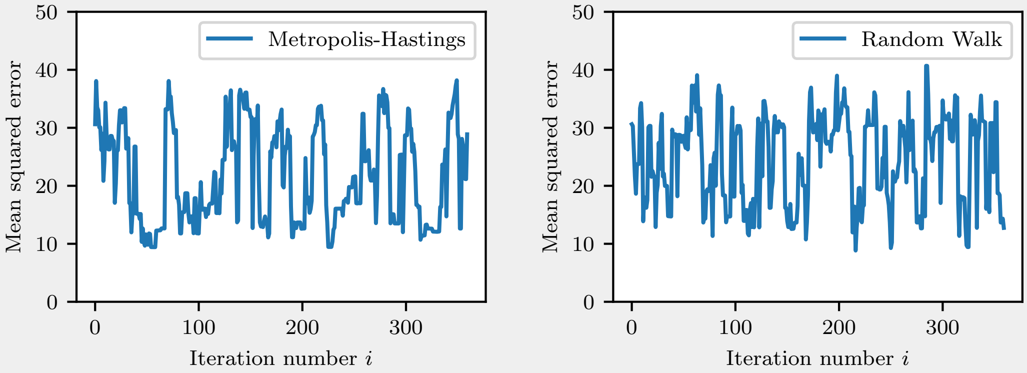

Figure 4.8: Mean squared errors corresponding to the permutation matrix visited at each iteration number i for the Metropolis-Hastings algorithm and the random walk algorithm.

Ee see in the plots that the worst mean squared error is about 90, which is only nine times a large as some of the best likelihoods. This is in stark contrast to the previous example, where the largest mean squared error is almost one hundred times larger than the smallest mean squared error. Such a small difference means that the probability of accepting a poor permutation matrix is now much larger, about one in nine, compared to the one in one hundred of the previous example. Since the likelihoods are now much closer together, it is more difficult for the Metropolis-Hastings algorithm to navigate through the search space of permutation matrices.

Nevertheless, the Metropolis-Hastings algorithm does pick slightly more suitable permutation matrices on average. The mean and median of the mean squared error corresponding to the permutation matrices chosen by the Metropolis-Hastings are 22 and 20 respectively, compared to 25 and 26 respectively for the Random Walk. Therefore, it seems that the Metropolis-Hastings algorithm is still 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 4.42, $\alpha_{P,P}$ represents the probability of accepting the new permutation matrix P' over the current permutation matrix P. This quantity is simply the fraction of the old mean squared error over the new mean squared error, which will be thresholded to one if this fraction is larger than one. Unfortunately, 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. Luckily, this probability of acceptance is not set in stone, and there are many different approaches to define such a probability of acceptance. The probability of acceptance should be finetuned such that we can capitalize on well-fitting permutation matrices. On the other hand, we should also be able to explore seemingly less suitable permutation matrices to make sure we sufficiently explore the search space of permutation matrices.

Translating the mean squared errors. In our setting, 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 ninety, meaning that we will now often pick one of the lesser suitable permutation matrices. A method to circumvent this is to *translate* the likelihoods. Although we do not know the smallest likelihood a priori, we know that a suitable lower bound of the likelihood of all permutation matrices is the likelihood of the ordinary least squares solution.

So, rather than estimating just a permuted upper triangular part of W, we estimate the full matrix W, and use the corresponding likelihood as a lower bound. Therefore, let W_{OLS} be the ordinary least squares solution, which corresponds to

$$W_{\text{OLS}} = \left(XX^T\right)^{-1} X^T Y, \tag{4.40}$$

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

$$\texttt{MSE}'(W_P) = \texttt{MSE}(W_P) - \texttt{MSE}(W_{\text{OLS}}). \tag{4.41}$$

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

$$\alpha'_{P,P'} = \begin{cases} \min\left\{\frac{\texttt{MSE}'(W_P)}{\texttt{MSE}'(W_{P'})}, 1\right\} & \text{if } \texttt{MSE}'(W_{P'}) > 0 \\ 1 & \text{otherwise.} \end{cases} \tag{4.42}$$

Now, the range of likelihoods range from slightly above zero to thirty. Therefore, the fractional differences are much higher. This will decrease 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

$$\alpha'_{P,P'} = \alpha^k_{P,P'}, \qquad k \in \mathbb{R}_{\geq 0}. \tag{4.43}$$

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, thereby resulting in a more exploitative search, where the algorithm is inclined to capitalize on more 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, thereby 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

$$\alpha'_{P,P'} = \begin{cases} 1 & \text{if } \texttt{MSE}(P') < \texttt{MSE}(P) \\ 0 & \text{otherwise.} \end{cases}. \tag{4.44}$$

In such a scenario, we will only transition to more suitable permutation matrices, and worse permutation matrices are never explored. This in turn also means that it is impossible to escape a local optimum. Nevertheless, let us return to the setting of Example 4.6 with our novel approaches to fine tune the probability of acceptance.

Example 4.7 Finetuning the probability of acceptance of Example 4.6.

As mentioned, the standard Metropolis-Hastings algorithm seems less suitable when the coefficient matrix W is sparse. Four simple transformations of the acceptance probability have been proposed to decrease the level of exploration for the Metropolis-Hastings algorithm. In this example, let us consider these four transformations on the setting of Example 4.6. First, we will investigate translating the likelihoods by subtracting $\texttt{MSE}(W_{\text{OLS}})$, the mean squared error of the ordinary least squares solution. Secondly, we will investigate raising the acceptance probability to the power of two, so $k = 2$. Thirdly, we will also try raising the translated acceptance probability to the power of two. Lastly, the greedy acceptance probability is considered, where we only transition when the corresponding likelihood of the new permutation matrix is strictly better.

To compare the suitability of the four proposed probabilities of acceptance against the regular probability of acceptance, we have plotted in Figure 4.9 the trajectories of the smallest likelihood found so far.

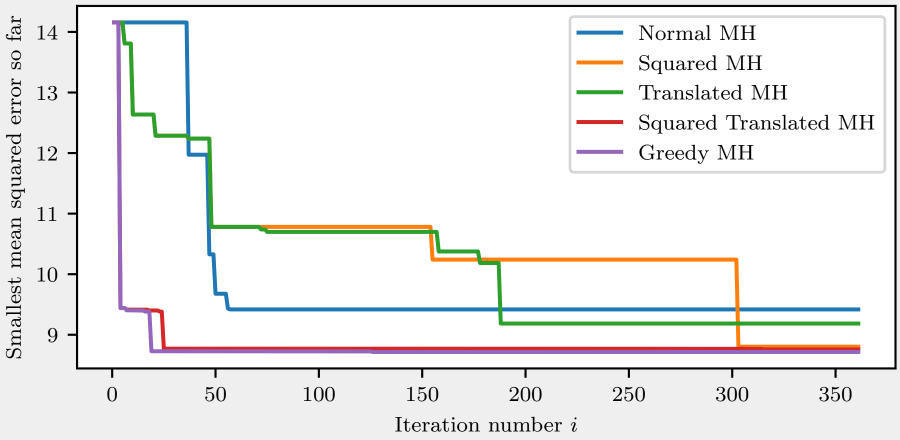

Figure 4.9: Trajectory of the lowest likelihood found so far for the Metropolis-Hastings algorithm with five different probabilities of acceptance.

First of all, let us remark that randomness plays a large role in all five trajectories, so the comparisons should be taken with a grain of salt. Nevertheless, we see that the squared translated and greedy probability of acceptance seems to work the best in the beginning. Nevertheless, after approximately 25 iterations, there is no improvement. The other probabilities of acceptance achieve a higher mean squared error, although the squared probability of acceptance also finds a well-fitting permutation matrix at around iteration number 300. The general 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 different probabilities of acceptance, let us consider the likelihoods of the permutation matrices at each iteration. These are plotted in Figure 4.11 for the four new acceptance probabilities.

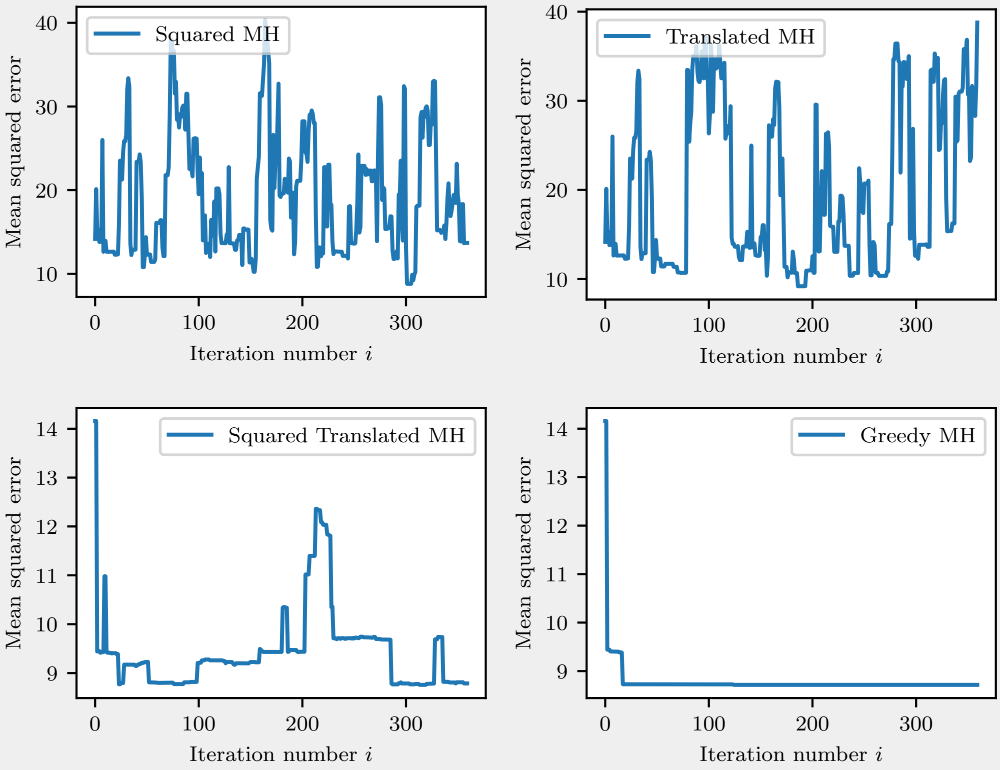

Figure 4.11: Mean squared errors of the permutation matrix at each iteration number i for the Metropolis-Hastings algorithm for different probabilities of acceptance.

We see that the Squared and Translated probabilities of acceptance are still quite explorative, as many unsuitable permutation matrices are transitioned to. On the other hand, the greedy probabability of acceptance is not explorative at all, as less suitable permutation matrix are never transitioned to. Somewhere in between lies the squared translated probability of acceptance, which tries both suitable and unsuitable permutation matrices.

Let us consider the mean and the median of the mean squared errors for the permutation matrices chosen by these four probabilities of acceptance, as well as the original Metropolis-Hastings algorithm and the Random Walk. All these are given in Table 4.2.

Table 4.2: Mean and Median corresponding to the 360 permutation matrices visited by the six methods we considered. We have considered the random walk that only considered adjacenct 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 indeed see that the Random Walk explores the most of all six methods, whereas the greedy probability of acceptance explores the least. 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.

Concluding Remarks. In this section, we have continued upon the random walk algorithm from Section 4.2. Instead of blindly trying permutation matrices, the Metropolis-Hastings algorithm utilizes a more guided search by considering the ratio of the likelihood of the proposed permutation matrix P over the likelihood of the current permutation matrix P. We have seen that this already improves on the random walk approach, especially when the coefficient matrix W is sparse and there is not enough time to visit a large number of permutation matrices.

We have also investigated multiple approaches of finetuning the probability of acceptance. Rather than taking the ratio of mean squared errors, we can also translate this ratio, raise this ratio to a power k, or a combination of both. Furthermore, we have also considered a greedy probability of acceptance. All these probabilities of acceptance are different in their trade-off of exploration and exploitation. We have seen that methods that fully focus in exploration, e.g. a random walk, are inefficient as too many poor permutation matrices are evaluated. On the other hand, a method that fully focuses on exploitation, e.g. the Metropolis-Hastings algorithm with the greedy acceptance probability, is too strict and hence has troubles getting out of local optima. All in all, a method that combines both exploration and exploitation, such as the Metropolis-Hastings algorithm with the translated acceptance probability raised to the power two, is better at weighing the exploration and exploitation.

Generally true, but did not see that in any of the examples

based on what do we make that claim?

4.4 Selecting a suitable model complexity.

Right now 4.4 comes and ends a bit abrupt. Also, we need here some more principled reasoning we regularization makes sense

So far, this chapter has been devoted to finding a suitable ordering or permutation 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.

However, as we have seen, maximizing the likelihood yields a dense directed acyclic graph, consisting of $p(p+1)/2$ edges. However, a large portion of these edges originate from spurious correlations with the noise and are often represented by very small weights w_{ij}. Therefore, we see that we can remove these edges as well without a significant decrease of the likelihood function.

That has to be explicit earlier on

Therefore, this section is devoted to finding suitable approaches to sensibly prune the coefficient matrix W. The model is less complex, yet still achieves a satisfactory likelihood on the data. In this manner, the pruned coefficient matrix will generalize slightly better to similar data, where the spurious correlations in the noise may have disappeared. This will be clarified in Example 4.8.

Example 4.8 Selection a suitable number of edges.

Consider the same dataset $\mathbf{X} \in \mathbb{R}^{100\times 10}$ as in Example 4.5, consisting of ten time series. We expect our permutation-based methods to find a suitable coefficient matrix $\hat{W}$ that respects some permutation matrix P. Assuming that $\hat{W}$ is optimal, $\hat{W}$ will most likely contain all edges that were contained in the matrix W used to generate $\mathbf{X}$. Apart from these true edges, $\hat{W}$ will also contain other edges with small weights, as we estimate a fully upper triangular matrix $\hat{U}$. Using the exhaustive search algorithm, we see that the most probable acyclic matrix $\hat{W}$ achieves a mean squared error of 9.28.

Maximizing the likelihood of $\mathbf{X}$, where we are only allowed to estimate entries that were indeed non-zero in the data-generating matrix W, achieves a mean squared error is 9.30. This is only fractionally smaller, whereas the number of arcs has been reduced from $\binom{10}{2} = 55$ edges to 35 edges. Therefore, we see that we can reduce the model complexity without a significant decrease in likelihood, or equivalently, without a significant increase in mean squared error.

Let us briefly consider a simple method that can be used to select a model of a smaller complexity.

Thresholding. A simple yet effective methods is to simply set all coefficients whose absolute value are smaller than some threshold ϵ to zero. So, the coefficients of $\tilde{W}$ will be thresholded as

$$\tilde{w}_{ij} = \mathbf{1}\left\{|w_{ij}| \geq \epsilon\right\}. \tag{4.45}$$

However, we still need to pick such a threshold value ϵ. We could for example pick $\epsilon = 0.10$, but this might remove important coefficients whose absolute value is smaller than 0.10 or alternatively, we might keep unimportant coefficients whose absolute value is larger than 0.10 due to spurious correlations.

Bibliography

[1] Scipy api reference for `optimize.minimize`. 23

[2] Scipy api reference for the `L-BFGS-B` optimization method. 23

[3] 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. 20

[4] Thomas Blumensath and Mike Davies. Iterative thresholding for sparse approximations. *Journal of Fourier Analysis and Applications*, 14:629–654, 12 2008. 78

[5] 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. 36

[6] Nancy Cartwright. Are rcts the gold standard? *BioSocieties*, 2(1):11–20, 2007. 4

[7] 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. 43

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

[9] David Maxwell Chickering. *Learning Bayesian Networks is NP-Complete*, pages 121–130. Springer New York, New York, NY, 1996. 15

[10] Gregory F. Cooper. The computational complexity of probabilistic inference using bayesian belief networks. *Artificial Intelligence*, 42(2):393–405, 1990. 4

[11] Gregory F. Cooper and Edward Herskovits. A bayesian method for the induction of probabilistic networks from data. *Machine Learning*, 9(4):309–347, Oct 1992. 22

[12] 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. 20

[13] Aramayis Dallakyan and Mohsen Pourahmadi. Learning bayesian networks through birkhoff polytope: A relaxation method, 2021. 25

[14] B. Efron. Bootstrap Methods: Another Look at the Jackknife. *The Annals of Statistics*, 7(1):1 – 26, 1979. 89

[15] C. W. J. Granger. Investigating causal relations by econometric models and cross-spectral methods. *Econometrica*, 37(3):424–438, 1969. 5

[16] W. K. Hastings. Monte carlo sampling methods using markov chains and their applications. *Biometrika*, 57(1):97–109, 1970. 43

[17] Sander Hofman. Making euv: From lab to fab, Mar 2022. 3

[18] Robin John Hyndman and George Athanasopoulos. *Forecasting: Principles and Practice*. OTexts, Australia, 2nd edition, 2018. 30

[19] Hidde De Jong. Modeling and simulation of genetic regulatory systems: A literature review. *JOURNAL OF COMPUTATIONAL BIOLOGY*, 9:67–103, 2002. 3

[20] Mahdi Khosravy, Nilanjan Dey, and Carlos Duque. *Compressive Sensing in Health Care*. 10 2019. vi, 78

[21] S. N. Lahiri. *Bootstrap Methods*, pages 17–43. Springer New York, New York, NY, 2003. 89

[22] 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

[23] S.G. Mallat and Zhifeng Zhang. Matching pursuits with time-frequency dictionaries. *IEEE Transactions on Signal Processing*, 41(12):3397–3415, 1993. 77, 78

[24] 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. 43

[25] 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. 24

[26] Judea Pearl. Fusion, propagation, and structuring in belief networks. *Artificial Intelligence*, 29(3):241–288, 1986. 4

[27] Judea Pearl. *Causality: Models, Reasoning and Inference*. Cambridge University Press, USA, 2nd edition, 2009. v, 1, 2, 4

[28] Judea Pearl and Thomas Verma. A theory of inferred causation. In *KR*, 1991. 16

[29] 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. 35

[30] Prof Carolina Ruiz. Illustration of the k2 algorithm for learning bayes net structures. 22

[31] Marco Scutari, Catharina Elisabeth Graafland, and José Manuel Gutiérrez. Who learns better bayesian network structures: Accuracy and speed of structure learning algorithms. 2018. 15

[32] 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. 18

[33] Ioannis Tsamardinos, Laura Brown, and Constantin Aliferis. The max-min hill-climbing bayesian network structure learning algorithm. *Machine Learning*, 65:31–78, 10 2006. 108

[34] 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

[35] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020. 23

[36] 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, 17

[37] Norbert Wiener. The theory of prediction. *Modern mathematics for engineers*, 1956. 5

[38] Tong Zhang. On the consistency of feature selection using greedy least squares regression. *Journal of Machine Learning Research*, 10(19):555–568, 2009. 77

[39] Xun Zheng, Bryon Aragam, Pradeep Ravikumar, and Eric P. Xing. Dags with no tears: Continuous optimization for structure learning, 2018. 22

[40] 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. 23

Appendix A

Code Snippets

This Appendix contains numerous small snippets of code that were used throughout this thesis. Note that this appendix does not contain the more complicated code associated with the algorithms, but rather simple scripts that aid in the understanding of concepts described throughout this thesis. The code snippets are provided in order of appearance in the thesis, starting from the snippets used in the introduction, to the snippets used in the conclusion.

A.1 Counting the number of permutations suitable to a matrix W

```python
import numpy as np
import itertools

def get_permutations(W):
    """@params: Weighted Adjacency Matrix W as numpy array"""
    # tracks the total number of permutations
    total = 0

    # iterate over all permutations of the identity matrix
    for perm in itertools.permutations(np.identity(np.shape(W)[0])):
        # convert to numpy array
        P = np.array(perm)

        # check if we have an upper triangular matrix
        if np.allclose(P.T @ W @ P, np.triu(P.T @ W @ P)):
            total += 1

    # return the number of suitable permutations
    return total
```

Appendix B

List of datasets

List of datasets.

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

Appendix C

Additional tables

The tables.

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