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

version 0.4

Eindhoven, June 2022

Contents

Chapter 5

Continuous Approaches

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

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

Although the results of the three methods discussed in Chapter 4 seem quite satisfactory, these permutation-based approaches have several drawbacks. First of all, searching over the space of permutation matrices P is quite difficult. The space is combinatorial in nature, making it hard to navigate. Secondly, the search space of permutation matrices is exponential with respect to the number of variables. There exist $p!$ different permutations on p variables. Therefore, the search space grows exponentially fast when the number of variables p increases. We indeed saw in Section 4.1 that exhaustively searching all permutations was infeasible for more than ten variables. To circumvent this, we settled for a suboptimal solution that could be found much faster. Nevertheless, especially when p gets large, we expect the performance of these permutation-based approaches to decrease.

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

We will discuss three approaches in this chapter. First of all, we will discuss a method close to permutation-based approaches in Section 5.1, where we relax the combinatorial search space of permutations matrices to a continuous search space. Secondly, in Section 5.2 we will explore an approach similar to the authors of NO TEARS [48] who discovered a novel approach to enforce acyclicity using a continuous function. Lastly, we propose a new method in Section **??** that enforces acyclicity using LASSO and systematically increasing the penalty parameter until the inferred structure is acyclic.

5.1 Relaxing the space of permutation matrices.

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

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

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

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

Interestingly, $\mathcal{P}_{\text{DS}}$ is the *convex hull* of P_{perm}. This implies that every doubly stochastic matrix can be written as a convex combination of permutation matrices, as the Birkhoff-von Neuman Theorem states [5, 43]. $\mathcal{P}_{\text{DS}}$ is therefore also known as the *Birkhoff polytope.* The vertices of the Birkhoff polytope correspond to permutation matrices, and all the interior points and points on a line segment are doubly stochastic matrices, but not permutation matrices. A 2D-visualization of the Birkhoff polytope in three dimensions is given in Figure 5.1

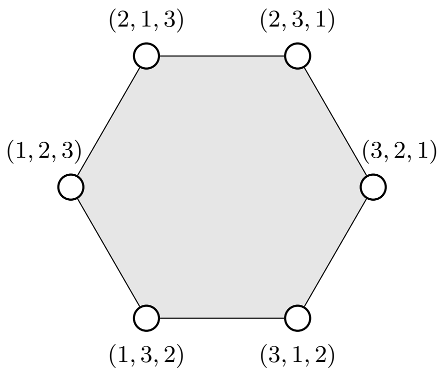

Figure 5.1: A two-dimensional representation of the Birkhoff polytope for three variables. Each of the $3! = 6$ vertices represent a permutation matrix, and all points on the edges or in the gray interior of the Birkhoff polytope represent doubly stochastic matrices.

Not needed

Figure 5.1 also allows for the so-called "rubber band analogy" to explain the notion of a convex hull. Consider the elements of $\mathcal{P}_{\text{perm}}$ to be nails sticking out of a surface, corresponding to the vertices of convex hull. Suppose we wrap a tight rubber band around these nails. The surface that falls inside the region of the rubber band corresponds to the convex hull of this set, which in our case is equivalent to the set of doubly stochastic matrices.

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

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

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

When we constrained ourselves to the space of permutation matrices $\mathcal{P}_{\text{perm}}$, we could enforce acyclicity by rewriting W as $W = P^T UP$, where P is a permutation matrix and U is an upper triangular matrix. However, now that we have relaxed to the space of doubly stochastic matrices, this decomposition is not accurate anymore. Rather than requiring the transpose of P, we now require its inverse. Therefore, we require

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

for an upper triangular matrix U.

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

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

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

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

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

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

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

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

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

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

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

Equation 5.5 and Equation 5.7 illustrate the problems that arise from our decomposition. For permutation matrices, computing the inverse of P was as easy as computing its transpose. For doubly stochastic matrices, this is not as simple anymore. Although we have resolved the issues caused by the combinatorial search space of permutation matrices, we are now left with invertibility issues. ~~We have exchanged our combinatorial constraint problems encountered in Chapter 4 for these invertibility issues.~~ Nevertheless, let us investigate how serious these issues are in practice.

Cost function for finite samples. To find a suitable doubly stochastic matrix P and an upper triangular matrix U, let us again consider minimizing the log-likelihood, which is proportional to

$$\begin{aligned} -\log\left(\mathbb{P}_W\left(\mathbf{X}\right)\right) &\propto \sum_{t=2}^{T} \left\|X_{t,\cdot} - X_{t-1,\cdot}P^{-1}UP\right\|_2^2 \\ &\propto \frac{1}{T-1}\sum_{t=2}^{T} \left\|X_{t,\cdot} - X_{t-1,\cdot}P^{-1}UP\right\|_2^2 \end{aligned} \tag{5.8}$$

For ease of notation, let us define Equation 5.8 as the cost function $C(P, U)$ that we want to minimize,

$$C(P,U) = \frac{1}{T-1}\sum_{t=2}^{T} \left\|X_{t,\cdot} - X_{t-1,\cdot}P^{-1}UP\right\|_2^2. \tag{5.9}$$

Equation 5.9 also corresponds to the mean squared error of $\mathbf{X}$ given our matrices P and U in this VAR(1) setting.

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

$$\begin{aligned} \mathbb{E}\left[C(P,U)\right] &= \lim_{T\to\infty} C(P,U) \\ &= \lim_{T\to\infty}\frac{1}{T-1}\left\|X_{t,\cdot} - X_{t-1,\cdot}P^{-1}UP\right\|_2^2 \\ &= \mathbb{E}\left[\left\|X_{t,\cdot} - X_{t-1,\cdot}P^{-1}UP\right\|_2^2\right] \\ &= \operatorname{Tr}\left(\mathbb{E}\left[\left(X_{t,\cdot} - X_{t-1,\cdot}P^{-1}UP\right)\left(X_{t,\cdot} - X_{t-1,\cdot}P^{-1}UP\right)^T\right]\right) \\ &= \operatorname{Tr}\left(\mathbb{E}\left[\hat{\varepsilon}_t\hat{\varepsilon}_t^T\right]\right), \end{aligned} \tag{5.10}$$

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

So, for the infinite sample case, we see that we can take the expectation of just a single time step. This in fact corresponds to the trace of the matrix obtained by multiplying the estimated error $\hat{\varepsilon}_t$ with its transpose $\hat{\varepsilon}_t^T$.

Why another paragraph?

Derivation of the cost function for infinite samples We can further decompose Equation 5.10 to gain more insights into the construction of our cost function. Using the definition of the covariance, we have that

$$\begin{aligned} \operatorname{Tr}\left(\mathbb{E}\left[\hat{\varepsilon}_t\hat{\varepsilon}_t^T\right]\right) &= \operatorname{Tr}\left(\mathbb{V}\left(\hat{\varepsilon}_t\right) - \mathbb{E}\left[\hat{\varepsilon}_t\right]\mathbb{E}\left[\hat{\varepsilon}_t\right]^T\right) \\ &= \operatorname{Tr}\left(\mathbb{V}\left(\hat{\varepsilon}_t\right)\right) - \operatorname{Tr}\left(\mathbb{E}\left[\hat{\varepsilon}_t\right]\mathbb{E}\left[\hat{\varepsilon}_t\right]^T\right) \\ &= \operatorname{Tr}\left(\mathbb{V}\left(\hat{\varepsilon}_t\right)\right) - \operatorname{Tr}\left(\mathbf{0}\mathbf{0}^T\right) \qquad (5.11) \\ &= \operatorname{Tr}\left(\mathbb{V}\left(X_{t,\cdot} - X_{t-1,\cdot}P^{-1}UP\right)\right) \\ &= \operatorname{Tr}\left(\mathbb{V}\left(X_{t,\cdot} - X_{t-1,\cdot}W\right)\right), \qquad (5.12) \end{aligned}$$

where we have used the fact that $\mathbb{E}\left[X_{t,.}\right] = \mathbf{0}$ in Equation 5.11.

So, we see that the expected mean squared error of our cost function is equal to the trace of the covariance of $\hat{\varepsilon}_t$, the difference between the actual value $X_{t,\cdot}$ and our predicted value $X_{t-1,\cdot}W$. So, we only need to derive the covariance of $X_{t,\cdot} - X_{t-1,\cdot}W$. Assuming that

$$X_{t,\cdot} = X_{t-1,\cdot}W^* + \varepsilon_t, \qquad \varepsilon_t \sim \mathcal{N}_p\left(\mathbf{0}, I_p\right), \tag{5.13}$$

we can derive Equation 5.12 further into

$$\begin{aligned}\mathbb{V}\left(X_{t,\cdot} - X_{t-1,\cdot}W\right) &= \mathbb{V}\left(X_{t-1,\cdot}W^* + \varepsilon_t - X_{t-1,\cdot}W\right)\\ &= \mathbb{V}\left(X_{t-1,\cdot}(W^* - W) + \varepsilon_t\right)\\ &= \mathbb{V}\left(X_{t-1,\cdot}(W^* - W)\right) + \mathbb{V}\left(\varepsilon_t\right)\\ &= (W^* - W)^T\mathbb{V}(X_{t-1,\cdot})(W^* - W) + I_p. \end{aligned} \tag{5.14}$$

What remains now is to derive the covariance of $X_{t-1,\cdot}$, or equivalently, the covariance $X_{t,\cdot}$.

Deriving $\mathbb{V}\left(X_{t,\cdot}\right)$. As we assume that all ε_t are independent Gaussian random variables, we know that $X_{t,\cdot}$ will follow a Gaussian distribution as well. Furthermore, as we assume our data is stationary with mean of $\mathbf{0}$, we know that $X_{t,\cdot} \sim \mathcal{N}(\mathbf{0}, \Sigma_X)$ for some covariance matrix $\Sigma_X \in \mathbb{R}^{p\times p}$. Note that this stationary distribution does not depend on t. What is left is to estimate Σ_X. From existing literature on Vector AutoRegressive models [3], we know that that we can derive the covariance matrix as

You spend quite some time explaining that, yet it does not seem to come back anywhere. Remove?

$$vec(\Sigma_X) = \left(I_{p^2} - (W^T \otimes W^T)\right)^{-1} vec\left(\mathbb{V}\left(\varepsilon_t\right)\right), \tag{5.15}$$

where $vec(\cdot)$ represents the *vectorization operation*, which stacks all columns into large one column. Furthermore, $\otimes$ represents the *Kronecker product*. A small two-dimensional example on how to derive $\mathbb{V}(X_{t,\cdot})$ using Equation 5.15 has been given in Example 5.1.

Example 5.1 Deriving $\mathbb{V}(X_{t,\cdot})$ for a VAR(1) model.

To give a concrete example of how Equation 5.15 can be used to derive $\mathbb{V}\left(X_{t,\cdot}\right)$, consider the two-dimensional VAR(1) model as in Equation 5.4 with

$$W = \begin{pmatrix}0.5 & 0.3\\ 0.0 & 0.4\end{pmatrix}, \qquad \mathbb{V}\left(\varepsilon_t\right) = I_2.$$

Computing the Kronecker product $W^T \otimes W^T$ and subtracting that from I_{p^2} yields

$$\begin{aligned} I_4 - (W^T \otimes W^T) &= \begin{pmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&0&1\end{pmatrix} - \begin{pmatrix} 0.5\begin{pmatrix}0.5&0.0\\0.3&0.4\end{pmatrix} & 0.0\begin{pmatrix}0.5&0.0\\0.3&0.4\end{pmatrix}\\ 0.3\begin{pmatrix}0.5&0.0\\0.3&0.4\end{pmatrix} & 0.4\begin{pmatrix}0.5&0.0\\0.3&0.4\end{pmatrix}\end{pmatrix}\\ &= \begin{pmatrix}0.75&0.00&0.00&0.00\\-0.15&0.80&0.00&0.00\\-0.15&0.00&0.80&0.00\\-0.09&-0.12&-0.12&0.84\end{pmatrix}.\end{aligned}$$

Then, alculating its inverse and multiplying it with the vectorization of I_p yields

$$\begin{aligned}\left(I_{p^2} - (W \otimes W)\right)^{-1} vec\left(\mathbb{V}\left(\varepsilon_t\right)\right) &= \begin{pmatrix}0.75&0.00&0.00&0.00\\-0.15&0.80&0.00&0.00\\-0.15&0.00&0.80&0.00\\-0.09&-0.12&-0.12&0.84\end{pmatrix}^{-1}\begin{pmatrix}1\\0\\0\\1\end{pmatrix}\\ &\approx \begin{pmatrix}1.33\\0.25\\0.25\\2.60\end{pmatrix}.\end{aligned}$$

Therefore, we conclude this example by noting that the covariance of $X_{t,\cdot}$ is equal to

$$\mathbb{V}\left(X_{t,\cdot}\right) = \begin{pmatrix}1.33&0.25\\0.25&2.60\end{pmatrix}.$$

Putting it all together. We conclude that the expected value of the mean squared error for our VAR(1) model equals

$$\begin{aligned}
\mathbb{E}\left[C(P,U)\right] &= \mathrm{Tr}\left(\mathbb{V}\left(X_{t,\cdot} - X_{t-1,\cdot}W\right)\right) && \text{by Equation 5.12} \\
&= \mathrm{Tr}\left((W^* - W)^T \mathbb{V}(X_{t,\cdot})(W^* - W) + \mathbb{V}\left(\varepsilon_t\right)\right) && \text{by Equation 5.14} \\
&= \mathrm{Tr}\left((W^* - W)^T \mathbb{V}(X_{t,\cdot})(W^* - W) +\right) + \mathrm{Tr}\left(\mathbb{V}\left(\varepsilon_t\right)\right), && (5.16)
\end{aligned}$$

where Equation 5.16 follows from the linearity of the trace operator, $\mathrm{Tr}\left(A+B\right) = \mathrm{Tr}\left(A\right) + \mathrm{Tr}\left(B\right)$.

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

$$(W - W')^T \mathbb{V}(X_{t,\cdot})(W - W') > 0. \tag{5.17}$$

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

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

$$P \in \mathcal{P}_{\mathrm{DS}},\ U \text{ upper triangular such that } P^{-1}UP = W^*. \tag{5.18}$$

The corresponding global minimum then is equal to

$$\mathrm{Tr}\left(\mathbb{V}\left(\varepsilon_t\right)\right) = \mathrm{Tr}\left(I_p\right) = p. \tag{5.19}$$

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

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

Example 5.2 Two (P,U) pairs that can compose W.

Let us consider the scenario where

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

We can decompose this W into several multiple (P,U) pairs. The first pair (P_1,U_1) is

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

and the second pair (P_2,U_2) is

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

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

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

$$C'(P,U) = \mathbb{E}\left[C(P,U)\right]. \tag{5.20}$$

The procedure for gradient descent is as follows. We iteratively update the entries of the matrices P and U by changing the corresponding value in the direction of the steepest descent. One by one, step by step, we update the entries such that the corresponding partial derivative decreases towards zero. If we keep updating these entries, we will eventually end up in a stationary point where all partial derivatives of the parameters are equal to zero. Instead of searching for a point where $\nabla_P C'(P,U)$ and $\nabla_U C'(P,U)$ are *exactly* equal to the zero matrix, we are satisfied when the gradient is sufficiently small, say when the squared 2-norm of all coefficients in P and U is below some threshold ϵ. The pseudocode of this procedure is provided in Algorithm 5.1.

Algorithm 5.1: Coordinate gradient descent without Lagrange multipliers.

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

```
1: initialize doubly stochastic Matrix $P$.
2: initialize upper triangular Matrix $U$.
3: while $\|\nabla_P C'(P,U)\|_F^2 + \|\nabla_U C'(P,U)\|_F^2 \geq \epsilon$ do
4:     $P \leftarrow P - \eta\nabla_P C'(P,U)$
5:     $U \leftarrow U - \eta\nabla_U C'(P,U)$
6: end while
7: return $P^{-1}UP$
```

rewrite

An important remark is we need to take into account is that P must be *doubly stochastic.* Therefore, we must add the constraints that all coefficients p_{ij} of P must be non-negative, and all rows and columns of P must sum to one. For this, we will later introduce *Lagrange multipliers*. Nevertheless, let us for now first introduce this method without Lagrange multipliers.

To perform this gradient descent, the gradients of $C'(P,U)$ with respect to our input matrices P and U need to be derived. In the coming two paragraphs, both gradients $\nabla_P C'(P,U)$ and $\nabla_U C'(P,U)$ will be derived. We start with $\nabla_U C'(P,U)$, as that one is easier.

Just state what the deltas are and move derivation to appendix.

Deriving $\nabla_U C'(P,U)$. The gradient of $C'(P,U)$ with respect to U can be quite easily derived by using matrix derivatives. By filling in the derived expected cost from Equation 5.16 and using the linearity of the trace operation, we get that the matrix derivative of $C'(P,U)$ with respect to U is equal to

$$\begin{aligned}
\frac{\partial C'(P,U)}{\partial U} &= \frac{\partial \mathrm{Tr}\left(\mathbb{V}\left(\varepsilon_t\right) + \left(W^* - W\right)^T \Sigma_X \left(W^* - W\right)\right)}{\partial U} \\
&= \frac{\partial \mathrm{Tr}\left(\mathbb{V}\left(\varepsilon_t\right)\right)}{\partial U} + \frac{\partial \mathrm{Tr}\left(\left(W^* - W\right)^T \Sigma_X \left(W^* - W\right)\right)}{\partial U} \\
&= \frac{\partial \mathrm{Tr}\left(\left(W^* - W\right)^T \Sigma_X \left(W^* - W\right)\right)}{\partial U},
\end{aligned} \tag{5.21}$$

where we could remove the first component in Equation 5.21 as the matrix derivative of ε_t with respect to U is zero.

Now, let us use Equation (111) of the matrix cookbook [33], which corresponds to

$$\frac{\partial \mathrm{Tr}(X^T B X)}{\partial X} = XB^T + XB, \qquad \text{(Equation 111 of [33])}$$

which is equal to $2XB$ if B is symmetric.

Applying this formula on Equation 5.21 and applying the chain rule yields

$$\begin{aligned}\frac{\partial \mathrm{Tr}\left((W^* - W)^T \Sigma_X (W^* - W)\right)}{\partial U} &= -2(W^* - W)\Sigma_X P^{-1} \frac{\partial \mathrm{Tr}(U)}{\partial U} P \\ &= -2(W^* - W)\Sigma_X. \end{aligned} \tag{5.22}$$

Note, however, that we will fix the gradient of U to be equal to zero for all lower triangular entries. Therefore, we have that the partial derivatives in our gradient $\nabla_U C'(P, U)$ can be computed as

$$\Big(\nabla_U C'(P,U)\Big)_{ij} = \begin{cases} \Big(-2(W - W^*)\Sigma_X\Big)_{ij} & \text{if } i \le j, \\ 0 & \text{otherwise.} \end{cases} \tag{5.23}$$

Deriving $\nabla_P C'(P, U)$. Deriving the gradient of $C'(P, U)$ with respect to P is slightly more involved than with respect to U, but nevertheless doable. Analogous to the gradient with respect to U, we first derive the matrix derivative

$$\begin{aligned}\frac{\partial C'(P,U)}{\partial P} &= 2\mathrm{Tr}\left((W^* - W)\Sigma_X \frac{\partial \mathrm{Tr}\,(W^* - W)}{\partial P}\right) \\ &= -2\mathrm{Tr}\left((W^* - W)\Sigma_X \frac{\partial P^{-1}UP}{\partial P}\right).\end{aligned}$$

What is left is to derive the partial derivatives

$$\frac{\partial P^{-1}UP}{\partial p_{ij}}. \tag{5.24}$$

We can use the product rule to decompose the partial derivative in Equation 5.24 into the two components

$$\frac{\partial P^{-1}UP}{\partial p_{ij}} = \frac{\partial P^{-1}}{\partial p_{ij}} UP + P^{-1}U \frac{P}{\partial p_{ij}}, \tag{5.25}$$

where the second component in Equation 5.25 can be computed as

$$P^{-1}U \frac{P}{\partial p_{ij}} = P^{-1}UJ_{ij}. \tag{5.26}$$

Here, J_{ij} corresponds to the matrix with zeros everywhere, except for the value in the ith row of the jth column, which is equal to one.

The first component in Equation 5.25 requires computing the derivative of the inverse P^{-1}, which also has a closed form. The partial derivative of the inverse matrix P^{-1} can be written as

$$\frac{\partial P^{-1}}{\partial p_{ij}} = -P^{-1} \frac{\partial P}{\partial p_{ij}} P^{-1} = -P^{-1} J^{ij} P^{-1}. \tag{5.27}$$

Hence, the first component in Equation 5.25 is equal to

$$\frac{\partial P^{-1}}{\partial p_{ij}} UP = -P^{-1} J^{ij} P^{-1} UP. \tag{5.28}$$

Putting the two components together, we can conclude that the partial derivative of the cost function $C'(P,U)$ with respect to p_{ij} is equal to

$$\begin{aligned}\frac{\partial C'(P,U)}{\partial p_{ij}} &= -2\text{Tr}\left(\Sigma_X(W^*-W)^T\frac{\partial P^{-1}UP}{\partial p_{ij}}\right)\\ &= -2\text{Tr}\left(\Sigma_X(W^*-W)^T\left(-P^{-1}J^{ij}P^{-1}UP+P^{-1}UJ^{ij}\right)\right), \end{aligned} \tag{5.29}$$

Then, the coefficients of the gradient $\nabla_P C'(P,U)$ are equal to

$$\left(\nabla_P C'(P,U)\right)_{ij} = \left(\frac{\partial C(P,U)}{\partial p_{ij}}\right)_{ij} \tag{5.30}$$

To verify our derivations, we have plotted the partial derivatives of our cost function both with respect to U and P in Example 5.3, where we indeed see that the partial derivatives are equal to zero exactly where $C'(P,U)$ attains its minimum.

Example 5.3 Verifying the gradients $\nabla_P C'(P,U)$ and $\nabla_U C'(P,U)$.
Consider a two-dimensional scenario where

$$P^* = \begin{pmatrix}1 & 0\\ 0 & 1\end{pmatrix}, \qquad U^* = \begin{pmatrix}0.8 & 0.5\\ 0.0 & 0.6\end{pmatrix}.$$

For each entry $p_{ij}, 1 \leq i,j \leq 2$, we have plotted both the cost function $C'(P^*,U^*)$ and the corresponding partial derivative in Figure 5.2. All other values for P^* and U^* the same, and only the corresponding coefficient p_{ij} is changed.

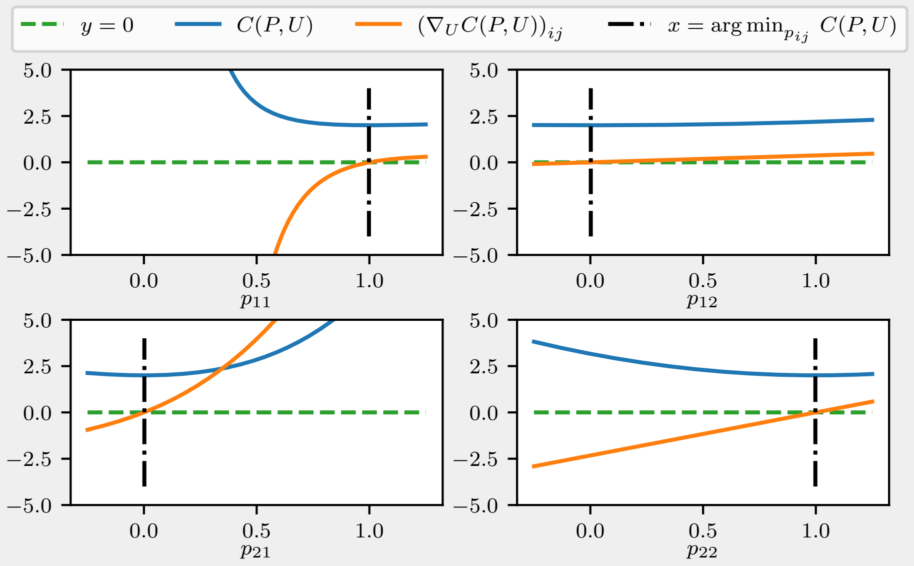

Figure 5.2: Value of the cost function and its partial derivative with respect to p_{ij} as a function of the four coefficients p_{ij} of P.

Indeed, we see that when the cost function $C'(P,U)$ attains its minimum for p_{ij}, the partial derivative with respect to p_{ij} equals zero.

For each upper triangular entry $u_{ij}, 1 \leq i < j \leq 2$, we have plotted the cost function and the corresponding partial derivatives in Figure 5.3.

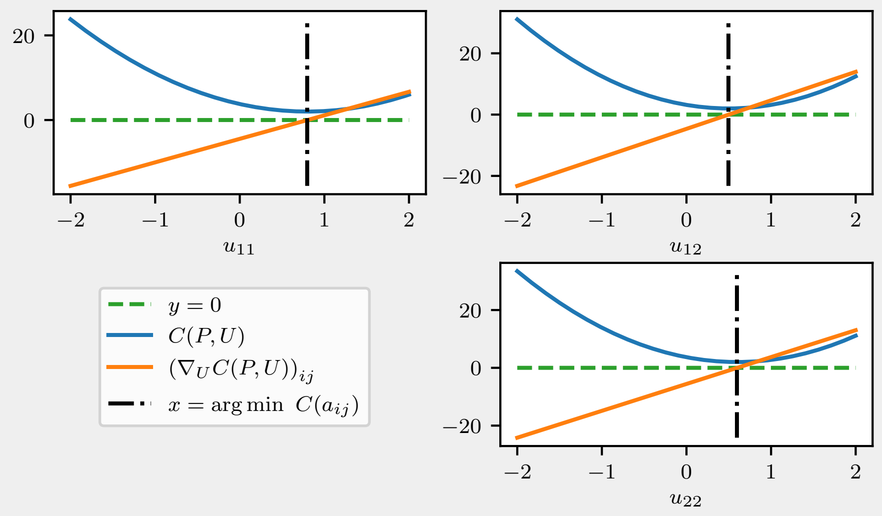

Figure 5.3: Value of the cost function and its partial derivative with respect to u_{ij} as a function of the three upper triangular coefficients u_{ij} of U .

We see that the cost function attains its minimum for u_{ij} when the partial derivative with respect to u_{ij} is equal to zero. Furthermore, we have checked that the slope of the partial derivative indeed corresponds to the rate of change of the cost function.

The findings above seems to imply that the gradients were derived and implemented correctly.

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

$$0 \leq p_{ij}, \qquad i, j = 1, \ldots p, \tag{5.31}$$

corresponding to p^2 *nonnegative* Lagrange multipliers

$$\lambda_{\text{ineq},i,j} = p_{ij}, \qquad i, j = 1, \ldots, p. \tag{5.32}$$

Furthermore, we introduce $2p$ equality constraints

$$\sum_{k=1}^{p} p_{ik} = \sum_{k=1}^{p} p_{kj} = 1, \qquad i, j = 1, \ldots, p, \tag{5.33}$$

corresponding to $2p$ Lagrange multipliers

$$\lambda_{\text{eq},1,i} = \sum_{k=1}^{p} p_{ik} - 1, \qquad \lambda_{\text{eq},2,j} = \sum_{k=1}^{p} p_{kj} - 1, \qquad i, j = 1, \ldots, p. \tag{5.34}$$

The Lagrangian function now becomes

$$\begin{aligned}\mathcal{L}(P, U, \lambda) &= C'(P, U) \\ &- \sum_{i=1}^{p}\sum_{j=1}^{p} \lambda_{\text{ineq},i,j} p_{ij} - \sum_{i=1}^{p} \lambda_{\text{eq},1,i} \left(\sum_{k=1}^{p} p_{ik} - 1\right) - \sum_{j=1}^{p} \lambda_{\text{eq},2,j} \left(\sum_{i=1}^{p} p_{kj} - 1\right).\end{aligned} \tag{5.35}$$

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

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

$$\nabla_U \mathcal{L}(P, U, \lambda) = \nabla_U C'(P, U). \tag{5.36}$$

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

$$(\nabla_P \mathcal{L}(P, U, \lambda))_{ij} = (\nabla_P C'(P, U))_{ij} - \lambda_{\text{ineq},i,j} - \lambda_{\text{eq},1,i} - \lambda_{\text{eq},2,j}. \tag{5.37}$$

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

Algorithm 5.2: Coordinate Gradient Descent With Lagrange Multipliers

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

```
1:  initialize doubly stochastic Matrix $P$
2:  initialize upper triangular Matrix $U$
3:  initialize Lagrange multipliers $\lambda$ using Equation 5.32 and Equation 5.34
4:  while $\|\nabla_P \mathcal{L}(P, U, \lambda)\|_F^2 + \|\nabla_U \mathcal{L}(P, U, \lambda)\|_F^2 \geq \epsilon$ do
5:      compute $\nabla_P \mathcal{L}(P, U, \lambda)$ using Equation 5.30 and Equation 5.37
6:      compute $\nabla_U \mathcal{L}(P, U, \lambda)$ using Equation 5.23 and Equation 5.36
7:      $P \leftarrow P - \eta \nabla_P \mathcal{L}(P, U, \lambda)$
8:      $U \leftarrow U - \eta \nabla_U \mathcal{L}(P, U, \lambda)$
9:      update Lagrange multipliers $\lambda$ using Equation 5.32 and Equation 5.34
10: end while
11: return $P^{-1}UP$
```

Gradient descent with acyclic W. Let us first apply this algorithm on a relatively simple example to see whether this approach is sensible. If the data generating matrix W^* is indeed acyclic, Algorithm 5.2 should be able to output a suitable acyclic matrix W, consisting of a doubly stochastic matrix P and an upper triangular matrix U. At the very least, we hope to find an acyclic matrix that correspond to a stationary point of Equation 5.35

Additionally, we hope that the doubly stochastic matrix P will closely resemble a permutation matrix. Nevertheless, we do not know yet whether P will indeed correspond to a permutation matrix, perhaps we will find a local optimum where $P \notin \mathcal{P}_{\text{perm}}$. Therefore, let us consider a simple three-dimensional setting in Example 5.4.

Example 5.4 Gradient descent with acyclic W

Let us consider the three dimensional data generating matrix

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

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

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

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

A closer inspection into the partial derivatives of P provide the explanation for why gradient descent has difficulties converging closer to a stationary point. In Figure 5.4, the partial derivative of $\mathcal{L}(P, U, \lambda)$ with respect to p_{31} has been shown.

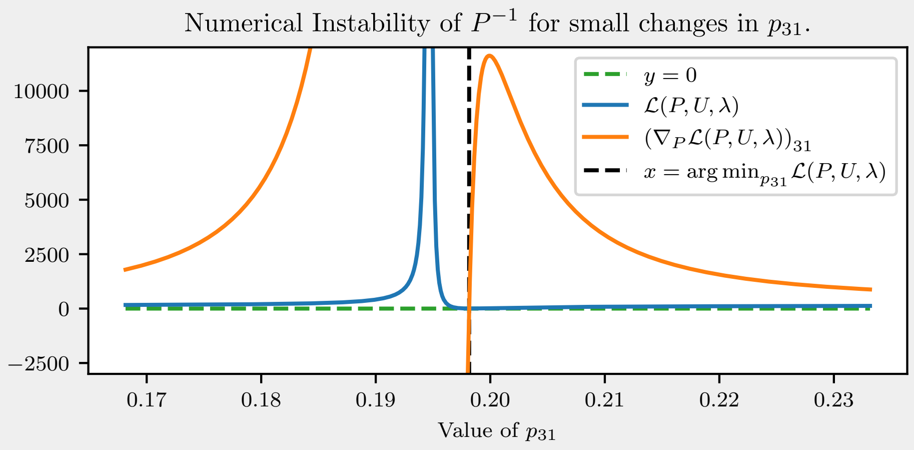

Figure 5.4: Value of the Lagrangian function and its partial derivative with respect to p_{31}, with the minimum of the Lagrangian corresponding to the root of the partial derivative.

Figure 5.4 reveals the singularity issues of P. As the two bottom rows of P are very similar, slightly tweaking p_{31} results in two linearly dependent rows of P so that the entries of the inverse matrix attain extremely large values, just as in Equation 5.7. When $p_{31} \approx 0.198$, the partial derivative of the Lagrangian with respect to p_{31} is equal to 0.015. However, if we increase the value of p_{31} by 0.002, the partial derivative has become larger than 10,000, which indicates that minor changes to p_{31} result in a substantial change in the corresponding partial derivative. This makes it difficult to exactly find a stationary point of $\mathcal{L}(P, U, \lambda)$. Nevertheless, the coefficient matrix W is reasonably close to the true coefficient matrix W^*.

Example 5.4 showcases two problems of with the gradient descent algorithm. First of all, we see that, although W^* is indeed acyclic, the corresponding doubly stochastic matrix P does not resemble a permutation matrix. We had hoped that a global optimum for W would result in a doubly stochastic matrix that was close to a permutation matrix. Unfortunately, this is not the case.

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

Also interesting to consider in other cases, for example for the permutation approaches. (Don't prioritize that though)

Coordinate Gradient Descent with cyclic W. As in real life, we should also investigate what happens when the data generating matrix W^* does not correspond to an acyclic structure. Then, although we are unable to find a perfect fit, we should still strive to find a suitable acyclic matrix $\hat{W}$. Therefore, let us see what happens when we apply gradient descent with Lagrange multipliers in such a setting. Hopefully, the algorithm will return a permutation matrix P and an upper triangular matrix U such that the inferred structure is acyclic.

Example 5.5 Gradient descent with cyclic W.

Consider a simple three dimensional settings where

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

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

In the end, we see our estimated matrices are

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

Clearly, P is nowhere close to a permutation matrix. Furthermore, we see that W is not acyclic at all.

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

Unfortunately, as we see in Example 5.5, although U is indeed upper triangular by construction, the additional parameters in P allow for a doubly stochastic matrix P such that we can compose a matrix W that does not correspond to an acyclic structure. As we can still construct cyclic matrices W under this decomposition, this relaxation method seems unsuitable to enforce acyclicity of W.

Final Remarks. Decomposing the coefficient matrix W into a permutation matrix P and an upper triangular matrix U such that

$$W = P^T UP, \qquad P \in \mathcal{P}_{\text{perm}} \tag{5.38}$$

seems like a sensible way to enforce $G(W)$ to be acyclic. However, the space of permutation matrices $\mathcal{P}_{\text{perm}}$ is exponentially large and discrete, meaning that finding a suitable permutation matrix P is difficult.

Does not seem like there is anything new in here.
You also did not have those summaries on other spots, right?
To me that makes the section unnecessarily long

Therefore, the constraint that $P \in \mathcal{P}_{\text{perm}}$ has been relaxed to $P \in \mathcal{P}_{\text{DS}}$, which represents the space of doubly stochastic matrices, changing our matrix decomposition to

$$W = P^{-1}UP, \qquad P \in \mathcal{P}_{\text{DS}}. \tag{5.39}$$

We have seen that this relaxation does not allow for better solutions, i.e., the global optimum remains the same after relaxation as long as the data generating matrix W^* was acyclic.

We have gone great lengths to derive the gradients of the expected cost $\mathbb{E}[C(P,U)]$ with respect to the matrices P and U. Furthermore, we have included Lagrange multipliers to make sure that P remains a doubly stochastic matrix. Using gradient descent as in Algorithm 5.2, we were able to find stationary points of the expected cost function, while taking the constraints into account.

Unfortunately, acyclic structures were only recovered when the data generating matrix W^* itself is acyclic. When W^* is cyclic, it is possible to compose a cyclic matrix W using a doubly stochastic matrix P and an upper triangular matrix U, although we explicitly required $G(W)$ to be acyclic.

Therefore, Algorithm 5.2 is not suitable for finding acyclic coefficient matrices W, as relaxing to the space of doubly stochastic matrices yields does not enforce acyclicity of $G(W)$.

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

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

$$p_{ij} \geq 0, \qquad i, j = 1, \ldots, p, \tag{5.40}$$

by transforming the matrix P. Instead of working with P directly, we transform P by using the element-wise *sigmoid* function,

$$\sigma(p_{ij}) = \frac{1}{1 + \exp(-p_{ij})}. \tag{5.41}$$

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

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

$$P^{-1} \approx \frac{P^T}{\det(P)}. \tag{5.42}$$

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

also iff?

$$\lambda \left(\sqrt{p} - \|P\|\right), \qquad \lambda \geq 0, \tag{5.43}$$

forces the doubly stochastic matrix closer to a permutation matrix P. Adding such a penalty seemed to work well for a similar relaxation method discussed in [16].

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

5.2 Applying NO TEARS to VAR(1) models.

In this section, we will take a closer look at the NO TEARS [48] approach introduced in Chapter 3. Originally, NO TEARS managed to approximately solve the left-hand side of Equation 5.44 by finding a local minimum of the continuous reformulation, which corresponds to the right-hand side of Equation 5.44,

$$\begin{aligned} \min_{W \in \mathbb{R}^{p \times p}} \quad & F(W) \\ \text{subject to} \quad & G(W) \in \texttt{DAGs} \end{aligned} \iff \begin{aligned} \min_{W \in \mathbb{R}^{p \times p}} \quad & F(W) \\ \text{subject to} \quad & h(W) = 0, \end{aligned} \tag{5.44}$$

Certainly need the definition of h earlier, and if you manage to keep it brief, also the intuition how that captures acylicity

where $F : \mathbb{R}^{p \times p} \to \mathbb{R}$ is a scoring function that quantifies the suitability of the matrix W given our data matrix $\mathbf{X} \in \mathbb{R}^{T \times p}$. The ingenuity of this approach is that $h(W)$ is a function which is equal to zero if and only if $h(W) = 0$. It can be seen as a function that quantifies the "DAG-ness" of a matrix W.

Not sure how genius that is :D

The authors of NO TEARS published the code required to run the original NO TEARS method on GitHub: `https://github.com/xunzheng/notears`. We can reuse most components of their code to apply the NO TEARS method for VAR(1) models. Nevertheless, we do need to make several changes. We need to modify the scoring function F, as well as the function $h : \mathbb{R}^{p \times p} \to \mathbb{R}$ that characterizes the acyclicity of the coefficient matrix.

Changes to the scoring function F. The original scoring function F of NO TEARS is

$$F(W) = \ell(W; \mathbf{X}) + \lambda \|W\|_1 = \frac{1}{2T} \|\mathbf{X} - \mathbf{X}W\|_F^2 + \lambda \|W\|_1, \tag{5.45}$$

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

$$\ell'(W;\mathbf{X}) = \frac{1}{2(T-1)} \left\|\mathbf{X}_{[2:T]} - \mathbf{X}_{[1:T-1]}W\right\|_F^2, \tag{5.46}$$

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

$$\nabla_W \ell'(W;\mathbf{X}) = -\frac{1}{T-1}\mathbf{X}_{[1:T-1]} \left\|\mathbf{X}_{[2:T]} - \mathbf{X}_{[1:T-1]}W\right\|_F. \tag{5.47}$$

Having defined the correct loss function for our setting, we have that the correct scoring function for our VAR(1) setting is

$$F'(W) = \ell'(W;\mathbf{X}) + \lambda \|W\|_1 = \frac{1}{2(T-1)} \left\|\mathbf{X}_{[2:T]} - \mathbf{X}_{[1:T-1]}W\right\|_F^2 + \lambda\|W\|_1. \tag{5.48}$$

Changes to the DAG-ness function $h(W)$. The function $h(W)$ from NO TEARS was designed such that a root of h corresponds to a directed acyclic graph,

$$h(W) = 0 \iff G(W) \text{ is a DAG.} \tag{5.49}$$

Originally, the function they proposed was defined as

$$h(W) = \operatorname{Tr}\left(e^{W\circ W}\right), \tag{5.50}$$

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

As $h(W)$ is equal to

$$\operatorname{Tr}\left(e^{W\circ W}\right) = \sum_{k=0}^{\infty} \frac{1}{k!}\operatorname{Tr}\left((W\circ W)^k\right), \tag{5.51}$$

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

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

$$h'(W) = \operatorname{Tr}\left(e^{\tilde{W}\circ\tilde{W}}\right), \tag{5.52}$$

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

$$\tilde{w}_{ij} = \begin{cases} w_{ij} & \text{if } i \neq j \\ 0, & \text{if } i = j. \end{cases} \tag{5.53}$$

Changes to the set-up. Apart from these two fundamental changes, some other minor changes to the set-up were required for the NO TEARS method to be suitable for VAR(1) models. First of all, NO TEARS was designed for a linear SEM, similar to Definition 2.2. Therefore, the diagonal entries of W were constrained to zero by the optimization procedure. However, as we want the method to be able to estimate diagonal entries of W as well, we needed to remove these constraints from the optimization procedure. Secondly, we have made slight modifications to the data generating process. We allow the diagonal of W to be non-zero. Lastly, we have added a method that simulates data according to our VAR(1) model. same?

Let us now consider two examples to showcase the performance of NO TEARS on VAR(1) models and how the regularization parameter λ can be used.

NO TEARS with acyclic W. Let us first consider a simple three-dimensional model where the coefficient matrix W^* that generates the data is acyclic.

Example 5.6 NO TEARS on a three-dimensional VAR(1) model.

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

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

The data matrix $\mathbf{X}$ has been plotted in Figure 5.5.

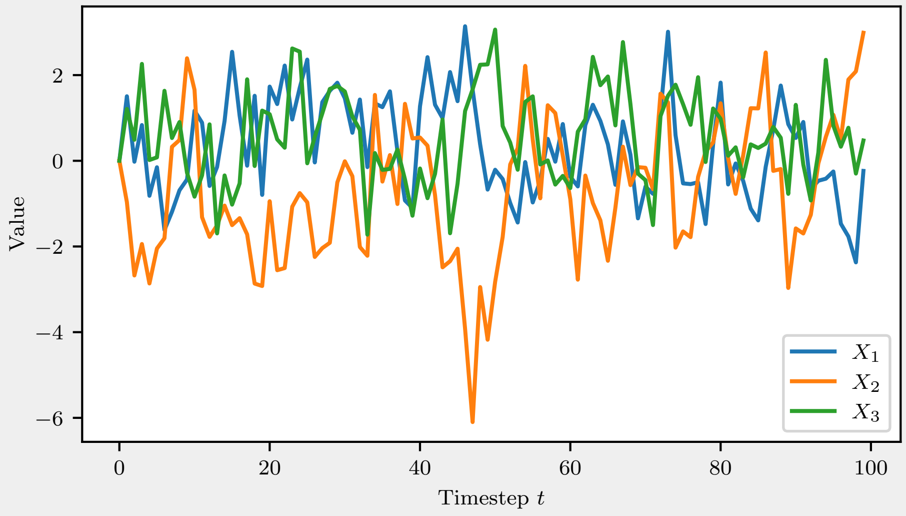

Figure 5.5: Visualization of the three variables X_1, X_2, X_2 of Example 5.6

Applying the modified NO TEARS method using no regularization, we obtain the matrix

$$W_{\text{NO TEARS},\lambda=0} = \begin{pmatrix} 0.53 & -0.64 & 0.03 \\ 0.00 & 0.57 & -0.20 \\ 0.00 & 0.00 & 0.40 \end{pmatrix},$$

which indeed corresponds to a DAG under our definitions. Furthermore, we see that the coefficients of W are indeed quite close to the true coefficients of W^*. However, since we do not apply any regularization, several values in the coefficient matrix are close to zero.

If we increase the regularization parameter to $\lambda = 0.05$, we see that NO TEARS outputs

$$W_{\text{NO TEARS},\lambda=0.05} = \begin{pmatrix} 0.50 & -0.61 & 0.00 \\ 0.00 & 0.56 & -0.19 \\ 0.00 & 0.00 & 0.37 \end{pmatrix}.$$

Therefore, NO TEARS recovers the true support of W^* for $\lambda = 0.05$.

Note that coefficients in $W_{\text{NO TEARS},\lambda=0.05}$ have become smaller than when we did not apply any regularization. Due to the regularization penalty, all coefficients are biased slightly slightly towards zero. We see that the NO TEARS method not only performs well on structural equation models, but also on this VAR(1) model after some modifications.

NO TEARS with cyclic W. Another interesting property of NO TEARS is that it is robust against model mismatches, that is, when the data generating matrix W^* does not match the model we are considering. To create such a model mismatch, we generate the data according to a cyclic VAR(1) model, although we are only considering acyclic model structures.

can you make precise what is meant with robust here?

Even when the data generating matrix does not correspond to a directed acyclic graph, NO TEARS will still find a suitable acyclic matrix W. The reason for this is that the constraint $h'(W) = 0$ simultaneously enforces acyclicity, while also trying to minimize the loss function. An example that showcases this robustness is discussed in Example 5.7.

Example 5.7 NO TEARS on three-dimensional VAR(1) data with a model mismatch.

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

$$W^* = \begin{pmatrix} 0.35 & 0.40 & 0.0 \\ -0.50 & 0.30 & 0.0 \\ 0.0 & 0.0 & 0.50 \end{pmatrix}.$$

We see that W^* does not correspond to a cyclic structure, as both w^*_{12} and w^*_{21} are non-zero. The data matrix $\mathbf{X}$ has been visualized in Figure 5.6.

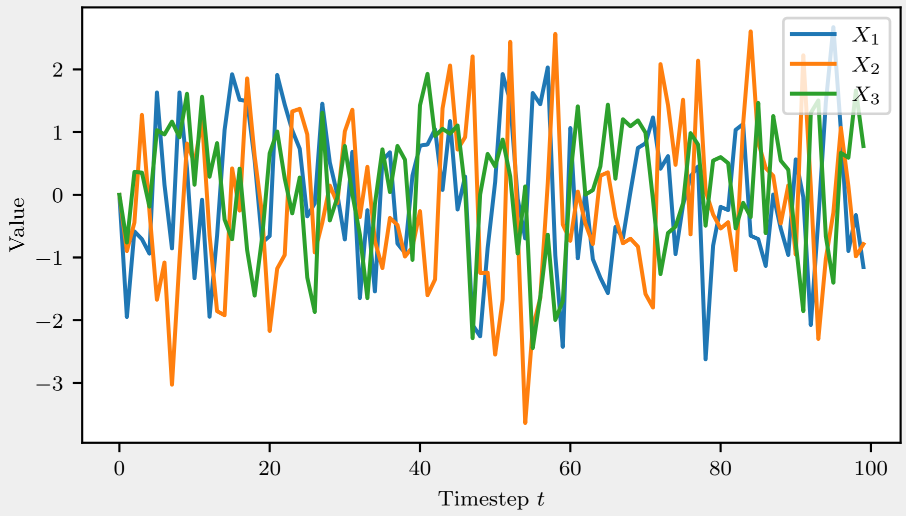

Figure 5.6: Visualization of the three variables X_1, X_2, X_2 of Example 5.7

Applying NO TEARS without any regularization, so $\lambda = 0$, yields the matrix

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

We see that we indeed already have a directed acyclic graph, so the acyclicity constraint forces NO TEARS to decide between setting either w_{21} or w_{12} to zero. We see that w_{13} is non-zero, whereas w^*_{13} was equal to zero.

If we increase the regularization parameter to $\lambda = 0.25$, then NO TEARS algorithm outputs the matrix

$$W_{\text{NO TEARS},\lambda=0.25} = \begin{pmatrix} 0.09 & 0.00 & 0.00 \\ -0.34 & 0.17 & 0.00 \\ 0.00 & 0.00 & 0.07 \end{pmatrix}.$$

We see that NO TEARS with $\lambda = 0.25$ manages to recover the true coefficients of W^*, except for the coefficient that violates the DAG-ness constraint. Therefore, it seems that NO TEARS also performs well on VAR(1) models, even with slight model mismatches.

Similar to before, I would not need that. But if you keep it, make sure those are also in the other method sections.
I also would call those paragraphs 'Summary'.

Final remarks We have seen in the literature that the NO TEARS method was able to recover an acyclic structure of a linear SEM. In this section, we have modified the NO TEARS algorithm to recover acyclic structures of VAR(1) models. Only the loss function $F(W)$ and the acyclicity constraint function $h(W)$ needed to be modified according to Equation 5.48 and Equation 5.52, along with some minor changes to the optimization procedure and the data generating procedure.

Applying NO TEARS on two VAR(1) examples, we indeed see that this NO TEARS method successfully recovers acyclic structures, even when the data generating matrix was cyclic. We have also briefly explored how we could select a value for λ to regularize our recovered coefficient matrix W. We see that increasing the regularization parameter λ helps in recovering the true structure of W^*.

5.3 Using a LASSO approach.

Another continuous approach to find a hopefully suitable directed acyclic graph $G(W)$ is by using a LASSO (Least Absolute Shrinkage and Selection Operator) approach, first introduced in 1996 [38]. Let us first describe the original regression setting.

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

$$\beta_t = \underset{\beta\in\mathbb{R}^p}{\arg\min} \left\{ \frac{1}{N}\sum_{i=1}^{N}\left(y_i - x_i\beta^T\right)^2 \right\} \text{ subject to } \sum_{i=1}^{p}|\beta_i| \leq k. \tag{5.54}$$

Another way to look at Equation 5.54, is by consider its *Lagrangian* formulation. Now, for a given $\lambda \in \mathbb{R}$, rather than a given t, we are trying to find

$$\beta_\lambda = \underset{\beta\in\mathbb{R}^p}{\arg\min} \left\{ \frac{1}{N}\sum_{i=1}^{N}\left(y_i - x_i\beta^T\right)^2 + \lambda\|\beta\|_1 \right\}. \tag{5.55}$$

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

LASSO in our setting. To cast the LASSO into our setting, we require a coefficient matrix W rather than a coefficient vector β. Furthermore, we are given a data matrix $\mathbf{X} \in \mathbb{R}^{T\times p}$, for which we want to do one-step-ahead prediction using one coefficient matrix W for all time steps $t = 2, \dots, T$. Therefore, we should aim to solve

$$W_k = \underset{W\in\mathbb{R}^{p\times p}}{\arg\min} \left\{ \frac{1}{T-1}\sum_{t=2}^{T}\|X_{t,\cdot} - X_{t-1,\cdot}W\|_2^2 \right\} \text{ subject to } \|W\|_1 \leq k, \tag{5.56}$$

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

Just as in the standard regression setting, we can write its Lagrangian counterpart as

$$W_\lambda = \underset{W\in\mathbb{R}^{p\times p}}{\arg\min} \left\{ \frac{1}{T-1}\sum_{t=2}^{T}\|X_t - X_{t-1}W\|_2^2 + \lambda\|W\|_1 \right\}. \tag{5.57}$$

For our setting, we additionally require $G(W)$ to be a directed acyclic graph. A possible approach is to simply increase the penalty parameter λ until $G(W)$ is a DAG. For a small value for λ, we do not severely penalize non-zero coefficients of W. Hence, for small values for λ, we expect a dense graph W. However, for large values for λ, non-zero coefficients in W are severely penalized. In that case, we can expect a sparse matrix W. Somewhere between these extremes exists a suitable value for λ, such that $G(W)$ is a directed acyclic graph and W fits $\mathbf{X}$ well.

To formalize, let W_λ be the solution to Equation 5.57 for a given data matrix $\mathbf{X}$ and a penalty parameter λ. Now, let λ^* be the smallest penalty parameter such that the graph $G(W_{\lambda^*})$ is acyclic. In mathematical notation, we are looking for

$$\lambda^* = \min_{\lambda \geq 0} \lambda \text{ such that } W_\lambda \text{ from Equation 5.57 corresponds to a DAG.}$$

Consequently, the matrix W_{λ^*} would then be the solution to Equation 5.57 with this specific regularization parameter λ^*.

Finding λ^* using binary search. An efficient method to find λ^* is by sensibly trying different values of λ and subsequently computing W_λ using Equation 5.57. Assume we have an efficient algorithm to compute the solution of Equation 5.57. We start by solving equation 5.57 for $\lambda = 1$. If $G(W_\lambda)$ is not acyclic, we know that λ^* is larger than 1. To find an upper bound for λ^*, we simply double λ at every iteration. We double λ until $G(W_\lambda)$ is a DAG, after which we know that λ^* is contained in the interval $(l, r] = (\frac{\lambda}{2}, \lambda]$.

To get arbitrarily close to the value for λ^*, we iteratively half the interval and continue with the interval in which λ^* is contained. If $G(W_\lambda)$ is a DAG for $\lambda = (l + r)/2$, we know that λ^* is contained in the lower half of the interval, and hence, we continue with $(l, \lambda]$. If $G(W_\lambda)$ is not a DAG, we know that λ^* resides in the upper half of the interval and we continue with $(\lambda, r]$.

We can continue this process until the interval is of arbitrarily small length, say of a width smaller than ϵ. We know that the right endpoint will be at less than ϵ away from the true value for λ^*. We then return W_r with the knowledge that W_r corresponds to a directed acyclic graph. The pseudocode for this binary search, which we call `DAG-LASSO`, is given in Algorithm 5.3.

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

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

```
1:  $l \leftarrow 0$
2:  $r \leftarrow 1$
3:  Compute $W_r$ using Equation 5.57
4:  while $G(W_r)$ is not a DAG do                 ▷ Find an upperbound for $\lambda^*$
5:      $l \leftarrow r$
6:      $r \leftarrow 2r$
7:      Compute $W_r$ using Equation 5.57
8:  end while
9:  $\lambda \leftarrow r$
10: while $r - l \geq \epsilon$ and $W_\lambda$ is not a DAG do    ▷ Iteratively half the interval $(l, r]$
11:     $\lambda \leftarrow \frac{r+l}{2}$
12:     Compute $W_\lambda$ using Equation 5.57
13:     if $G(W_\lambda)$ is a DAG then
14:         $r \leftarrow \lambda$
15:     else
16:         $l \leftarrow \lambda$
17:     end if
18: end while
19: return $W_r$ using Equation 5.57.
```

Number of required iterations. The first while loop will run $\lceil \log_2(\lambda^*) \rceil$ times, after which we have found the correct upper bound for λ^*. Note that if $W_{\lambda=1}$ is already a directed acyclic graphs, so technically the first while loop will be evaluated $\max\{\lceil \log_2(\lambda^*) \rceil, 0\}$ times. rewrite

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

Solution Path of `LASSO-DAG`. A common visualization of the LASSO algorithm is the so-called *solution path* [39]. We can plot each of the values of the coefficients in W_λ for each value for λ.

For $\lambda = 0$, we expect all coefficients to be quite large, and for a large enough value for λ, all coefficients in the matrix W_λ will be equal to zero, as the penalization is so severe. For all values for λ in-between, we can plot all the values of all coefficients of W_λ. We expect each coefficient in W_λ to converge to zero as a function of λ. Two examples of such solution paths are given in Example 5.8 and Example 5.9.

Interestingly, the lasso-path of a coefficient w_{ij} is *continuous* and *piecewise linear*. The two statements imply that the solution path of each coefficient consists of sloped lines with kinks when another coefficient has been shrunk to zero. A more formal description of the solution path for the LASSO is given in [39]. In our scenario, changing a value in one column of W_λ only affects the other values in the same column. Therefore, we expect kinks in the solution path of w_{ij} only when another edge w_{kj} has been shrunk to zero.

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

`DAG-LASSO` on an acyclic matrix W. Let us investigate the inner workings of `DAG-LASSO` in greater detail in Example 5.8, where the data has been generated using an acyclic coefficient matrix W.

Example 5.8 `DAG-LASSO` on three-dimensional data.

To showcase how `DAG-LASSO` works, let us consider an example based on a data matrix $\mathbf{X} \in \mathbb{R}^{100\times 3}$ consisting of three time series of one hundred time steps. The data $\mathbf{X}$ is exactly the same as in Example 5.6, which has been generated according to a VAR(1) model with the data generating matrix W^* given in the left-hand side of Equation 5.58.

Now, for $\lambda = 0$, we simply get the maximum likelihood estimator which coincides with the ordinary least squares estimator W_{OLS} in the right-hand side of Equation 5.58.

$$W^* = \begin{pmatrix} 0.50 & -0.65 & 0.00 \\ 0.00 & 0.50 & -0.25 \\ 0.00 & 0.00 & 0.40 \end{pmatrix}, \qquad W_{\text{OLS}} = \begin{pmatrix} 0.50 & -0.62 & 0.00 \\ -0.02 & 0.54 & -0.16 \\ -0.14 & -0.11 & 0.32 \end{pmatrix}. \tag{5.58}$$

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

If we increase the regularization penalty to $\lambda = 0.1$, we get $W_{\lambda=0.1}$, which has been given in the left-hand side of Equation 5.59. This coefficient matrix is already closer to a sparse matrix, yet is still not acyclic. Furthermore, we see that many of the entries that are zero in the true matrix W^* are quite close to zero in $W_{\lambda=0.1}$. Although many coefficients in $W_{\lambda=0.1}$ are close to zero, its structure is still not acyclic. Therefore, we see that $\lambda = 0.1$ is not a large enough penalty parameter to enforce acyclicity.

Using `DAG-LASSO` with $\epsilon = 0.01$ suggests that $\lambda^* = 0.14$ is the smallest λ such that $G(W_\lambda)$ is acyclic, yielding the coefficient matrix W_{λ^*} in the right-hand side of Equation 5.59.

$$W_{\lambda=0.1} = \begin{pmatrix} 0.50 & -0.62 & 0.00 \\ -0.02 & 0.52 & -0.13 \\ -0.03 & -0.03 & 0.24 \end{pmatrix}, \qquad W_{\lambda^*=0.14} = \begin{pmatrix} 0.39 & -0.54 & 0.00 \\ 0.00 & 0.51 & -0.11 \\ 0.00 & 0.00 & 0.21 \end{pmatrix}. \tag{5.59}$$

We see that `DAG-LASSO` with $\lambda^* = 0.14$ recovers the true support of W^*, although the values are slightly smaller as the penalization parameter biases the coefficients towards zero.

Let us also investigate the solution path of the coefficients of W_λ. The solution path has been given in Figure 5.7.

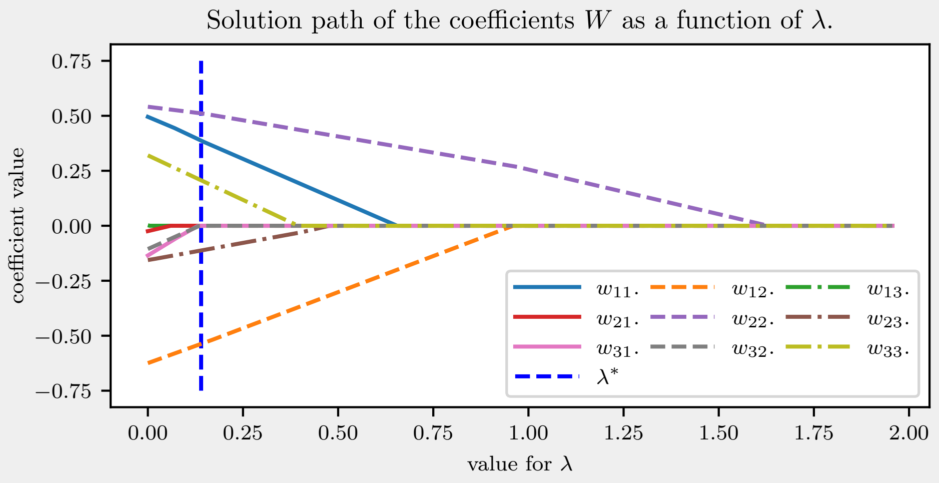

Figure 5.7: Visualization of the solution path for the coefficients of W of Example 5.8.

We see in Figure 5.7 that the value for λ^* is indeed equal to 0.14. Upon close inspection, we see the coefficients w_{13}, w_{21}, w_{31}, and w_{32} have shrunk to zero, after which we have an acyclic coefficient matrix W.

Interestingly, we see that w_{33} was estimated around -0.16, in W_{OLS}, which is in the same order of magnitude as the unimportant edges. Interestingly, we see that the solution path of w_{33} is much more gradual. Even more, w_{33} has shrunk to zero even later than w_{32}, which indicates that w_{33} is more important than w_{32}, despite its coefficient being twice as small in magnitude.

Therefore, we see that the solution path of the coefficients is also a suitable indicator of how important a coefficient is.

`LASSO-DAG` on a cyclic W. Normally, we assume the data is generated with a VAR(1) model and a true underlying coefficient matrix W^*, where $G(W^*)$ is a DAG. However, in real-life, this might not always be the case, and we could have that $G(W^*)$ is *almost* a DAG, but not quite. Let us see what happens to the `DAG-LASSO` algorithm when we introduce one extra edge such that the acyclicity assumption is violated.

Example 5.9 `DAG-LASSO` with model mismatch.

Consider a three-dimensional time series with one hundred time steps. The data matrix is equivalent to data matrix used in Example 5.7, which has been generated according to a VAR(1) model with data generating matrix W^*, which has been given in the left-hand side of Equation 5.60.

$$W^* = \begin{pmatrix} 0.35 & 0.40 & 0.00 \\ -0.50 & 0.30 & 0.00 \\ 0.0 & 0.0 & 0.50 \end{pmatrix}, \qquad W_{\lambda=0} = \begin{pmatrix} 0.28 & 0.42 & -0.18 \\ -0.51 & 0.32 & 0.00 \\ -0.08 & 0.00 & 0.31 \end{pmatrix}. \tag{5.60}$$

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

Therefore, let us consider what happens for an already quite large penalty parameter $\lambda = 0.25$, and for the smallest penalty parameter such that W_λ corresponds to an acyclic structure, $\lambda^* = 0.51$. Both are given in Equation 5.61.

$$W_{\lambda=0.25} = \begin{pmatrix} 0.09 & 0.21 & 0.00 \\ -0.36 & 0.16 & 0.00 \\ 0.00 & 0.00 & 0.05 \end{pmatrix}, \qquad W_{\lambda^*=0.51} = \begin{pmatrix} 0.00 & 0.00 & 0.00 \\ -0.19 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.00 \end{pmatrix}. \tag{5.61}$$

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

Apart from the three matrices depicted above, we can also look at the solution path of `DAG-LASSO` applied on our data matrix $\mathbf{X}$. This solution path of all coefficients in W_λ has been plotted in Figure 6.5.

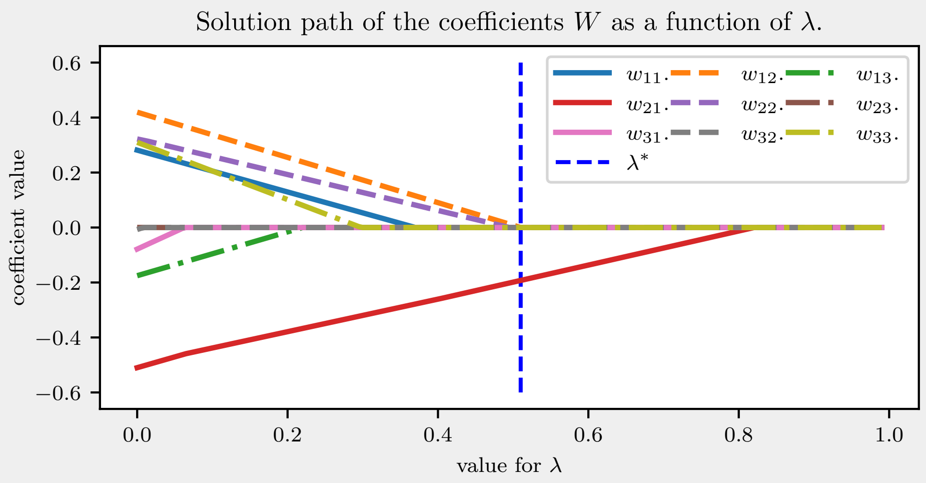

Figure 5.8: Visualization of the solution path for the coefficients of W of Example 5.9

We indeed see that three unimportant edges, w_{23}, w_{31}, and w_{32} converge to zero quite fast. At $\lambda \approx 0.22$, we have recovered the support of the true graph W^*, but we are interested in recovering an acyclic structure. To retrieve an acyclic structure, we see that we first shrink w_{33}, w_{11}, and w_{22} to zero. Now, we still have a cycle of length two, (w_{12}, w_{21}). Since these two edges are deemed the most important, all other edges have been shrunk to zero. Now, a "death-match" between the two remaining edges in the cycle remains, after which we are left with an acyclic matrix W_{λ^*} at $\lambda^* \approx 0.51$, where only $w_{21} \approx -0.19$ remains as non-zero coefficient of W_{λ^*}.

This example showcases that when there is a model mismatch, albeit just one cycle of length two violating the DAG-ness assumption, then `DAG-LASSO` is unable to return a suitable coefficient matrix W_{λ^*}. Its global approach to enforce acyclicity is not effective when the data generating matrix W^* itself was acyclic.

I would not consider that as a large disadvantage, at least regarding structure learning

Those that also come for the other methods later? Would be nice to have the 'disadvantages' somehow streamlined

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

One large disadvantage is that the coefficients in W_{λ^*} are *biased* towards zero, as we have seen in both Example 5.8 and Example 5.9. We can overcome this bias by re-estimating the non-zero coefficients of W_{λ^*}, as also highlighted in [19].

So, define the support of W_{λ^*} as

$$\operatorname{supp}(W) = \{(i,j) \mid w_{ij} \neq 0\}. \tag{5.62}$$

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

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

$$\hat{W}_{\cdot,j} = \left(\mathbf{X}'_j \mathbf{X}'^{T}_j\right)^{-1} \mathbf{X}'^{T}_j Y_{\cdot,j}, \qquad j = 1,\dots,p. \tag{5.63}$$

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

A second and more serious disadvantage is that DAG-LASSO does not provide a sensible solution when the data generating matrix W^* is cyclic, as we have seen in Example 5.9. The value for λ will be increased until the structure is acyclic, but one cycle of length two can cause the inferred structure to consist of only a single arc.

Final Remarks. In this section, we have investigated the effectiveness of using the LASSO penalty to retrieve an acyclic matrix W. We were interested in finding the smallest penalty parameter λ^* such that $G(W)$ corresponds to a directed acyclic graph. This λ^* and consequently W_{λ^*} was quite easily retrieved using a binary search approach, where we search the interval containing λ^* by splitting it in half after each iteration.

We have seen in Example 5.8 that when W is indeed acyclic, the result returned by DAG-LASSO is sensible and fits the data well while remaining acyclic. However, when W is cyclic, as in Example 5.9, then DAG-LASSO fails to provide a sensible matrix W. We have seen that in such a scenario, all edges less important than the edges that violate the acyclicity constraint are removed first. In the extreme case, we are left with only one edge. The global approach of DAG-LASSO to enforce acyclicity makes it unsuitable to estimate acyclic matrices W when the data generating coefficient matrix was cyclic.

Lastly, we have introduced the solution path of the coefficients of W as a function of λ, and have also shown how these can provide useful insights into the importance of the estimated edges in W.

Bibliography

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

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

[3] *Vector Autoregressive Models for Multivariate Time Series*, pages 385–429. Springer New York, New York, NY, 2006. 58

[4] Mark Bartlett and James Cussens. Integer linear programming for the bayesian network structure learning problem. *Artificial Intelligence*, 244:258–271, 2017. Combining Constraint Solving with Mining and Learning. 20

[5] Garrett Birkhoff. Three observations on linear algebra. *Univ. Nac. Tacuman, Rev. Ser. A*, 5:147–151, 1946. 55

[6] Thomas Blumensath and Mike Davies. Iterative thresholding for sparse approximations. *Journal of Fourier Analysis and Applications*, 14:629–654, 12 2008. 80

[7] Graham Brightwell and Peter Winkler. Counting linear extensions is #p-complete. In *Proceedings of the Twenty-Third Annual ACM Symposium on Theory of Computing*, STOC '91, page 175–181, New York, NY, USA, 1991. Association for Computing Machinery. 36

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

[9] 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

[10] 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

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

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

[13] 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

[14] 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

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

[16] Aramayis Dallakyan and Mohsen Pourahmadi. Learning bayesian networks through birkhoff polytope: A relaxation method. *CoRR*, abs/2107.01658, 2021. 67

[17] B. Efron. Bootstrap Methods: Another Look at the Jackknife. *The Annals of Statistics*, 7(1):1 – 26, 1979. 92

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

[19] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. Springer New York Inc., New York, NY, USA, 2001. 76

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

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

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

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

[24] Mahdi Khosravy, Nilanjan Dey, and Carlos Duque. *Compressive Sensing in Health Care*. 10 2019. vi, 79

[25] S. N. Lahiri. *Bootstrap Methods*, pages 17–43. Springer New York, New York, NY, 2003. 92

[26] 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

[27] S.G. Mallat and Zhifeng Zhang. Matching pursuits with time-frequency dictionaries. *IEEE Transactions on Signal Processing*, 41(12):3397–3415, 1993. 78, 80

[28] 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

[29] 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

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

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

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

[33] K. B. Petersen and M. S. Pedersen. The matrix cookbook, October 2008. Version 20081110. 61

[34] 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

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

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

[37] 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

[38] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1):267–288, 1996. 71

[39] Ryan Tibshirani and Jonathan Taylor. The solution path of the generalized lasso. *The Annals of Statistics*, 39, 05 2010. 73

[40] Ioannis Tsamardinos, Laura Brown, and Constantin Aliferis. The max-min hill-climbing bayesian network structure learning algorithm. *Machine Learning*, 65:31–78, 10 2006. 111

[41] 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

[42] 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

[43] John Von Neumann. A certain zero-sum two-person game equivalent to the optimal assignment problem. *Contributions to the Theory of Games*, 2(0):5–12, 1953. 55

[44] 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

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

[46] Tong Zhang. On the consistency of feature selection using greedy least squares regression. *Journal of Machine Learning Research*, 10(19):555–568, 2009. 78

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

[48] Xun Zheng, Bryon Aragam, Pradeep Ravikumar, and Eric P. Xing. Dags with no tears: Continuous optimization for structure learning. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, page 9492–9503, Red Hook, NY, USA, 2018. Curran Associates Inc. 54, 67

[49] 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