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

Contents

Chapter 6

Iterative Approaches

This is the third and final chapter where we introduce approaches to learn an acyclic structure of a data matrix $\mathbf{X}$. In Chapter 4 we have seen that we can decompose our coefficient matrix W into a permutation matrix P and an upper triangular matrix U, and consequently optimize over the discrete space of permutation matrices. In Chapter 5, we have discussed approaches that enforce acyclicity through continuous constraints. These continuous constraints were enforced using the space of doubly stochastic matrices in Section 5.1, a function $h(W)$ that equals zero if and only if W corresponds to an acyclic structure in Section 5.2, and a lasso-approach with an increasing penalty parameter in Section 5.3.

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

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

There are several interesting procedures to update the coefficient matrix W one arc at the time. The first choice we have is the *direction* of the iterative approach. Firstly, we can start with an empty graph, and iteratively add arcs that maximize some selection criterion. These are called *forward* iterative approaches. Secondly, we can also iteratively update the coefficient matrix in the opposite direction. We can start with a fully connected graph, or a fully estimated matrix W, and iteratively remove the arc that maximizes some deletion criterion. Such procedures are called *backward* iterative approaches.

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

A second advantage is that they are quite efficient. As soon as an arc has been added, this arc will remain in the structure. In the opposite direction, when an arcs has been removed, this arc will never reappear in the structure. In the worst case, we only need to iterate over all arcs once. Now, if we can evaluate our selection or deletion criterion efficiently, such an iterative approach might be more tractable for a large number of time series.

Lastly, we can interpret the order in which arcs are selected as an ordering of how important the arcs in the coefficient matrix are. The sooner an arc is added, the more important we consider it to be. Similarly for a backwards procedure, the later the arc is removed, the more important we consider it to be. As we will see later in this chapter, having such an ordering can be helpful in fine tuning the graphical model such that we only keep arcs that are considered important enough.

Outline of this chapter. In this chapter, we will be discussing various iterative procedures, both in the forward and backwards direction. Furthermore, we will be investigating different criteria to decide whether to include a coefficient in the forwards direction, or whether to remove a coefficient in the backwards direction.

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

In Section 6.4, we will be investigating in greater detail how we should decide on a sensible complexity of the structure. A model with too few arcs does not capture all the necessary behavior, but too much arcs results in an unnecessarily complex model. As these iterative approaches update the coefficient matrix one arc at the time, we can interpret this as an ordering of importance of the arcs, which we can use to select a suitable number of arcs in our graph. We will be investigating bootstrapping approaches in Subsection 6.4.1 and cross-validation approaches in Subsection 6.4.2.

Although cross-validation approaches are unconventional for time series data due to the fact that there is quite some dependencies between, for example between consecutive time steps, we will still evaluate their performance and see that they perform surprisingly well. To gain more insights as to why cross-validation seems to be an effective approach, we will conduct a more theoretical analysis on cross-validation using a simple one-dimensional setting in Section 6.5.

Determining whether the structure of W is acyclic. All iterative approaches require an efficient method to determine whether the structure characterized by W, or equivalently $G(W)$, the graph induced by W, is acyclic. Therefore, let us consider two methods to determine whether a coefficient matrix corresponds to an acyclic structure.

The first method that is most readily available to use is to use a suitable trace exponential function, as discussed in Section 5.2. A suitable trace exponential function would be

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

where $\tilde{W}$ corresponds to the matrix W with the diagonal entries set to zero, and p corresponds to the number of variables or equivalently, the number of nodes in the structure. Computing the trace exponential can be done in $\mathcal{O}(p^3)$ running time [4], so it is not the most efficient approach, but suitable implementations are readily available in standard code libraries.

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

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

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

Pseudo code to verify the existence of a topological ordering of the variables given a coefficient matrix W has been provided in Algorithm 6.1. Such an algorithm is more frequently known as a topological sort, and this approach is quite similar to Kahn's algorithm [33].

Algorithm 6.1: Topological Sort

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

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

We apply this topological sort algorithm on two coefficient matrices to check whether they correspond to an acyclic structure in Example 6.1

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

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

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

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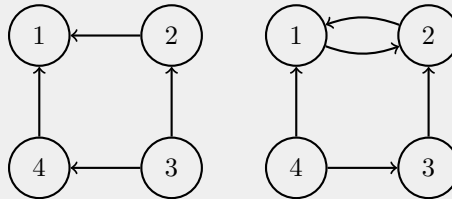

Figure 6.1: Visualization of the graphs induces by the coefficient matrices W_1 (left) and W_2 (right) from Example 6.1. Self-loops of both graphs have been removed.

For W_1, after removing self-loops, the third node has no incoming arcs, as the third column is empty. Hence, we can remove node 3 from the graph, and add it to our topological ordering. Afterwards, we see that we can remove either node 2 or node 4. In such a tie, we can choose one arbitrarily, but for consistency, let us select the node that comes first lexicographically, so we remove node 2 and its outgoing arcs. Now, we see that we can remove node 4, and then we can remove node 1, yielding a topological ordering of $\pi = (3, 2, 4, 1)$, meaning that W_1 corresponds to an acyclic structure. We see that such a topological ordering need not be unique, as $\pi = (3, 4, 2, 1)$ is also a valid topological ordering for W_1.

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

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

6.1 Using Orthogonal Matching Pursuit

The first iterative method that we will be discussing is a forward iterative procedure based on the Orthogonal Matching Pursuit (OMP) algorithm. OMP is nowadays commonly known as a sparse regression technique, but was first introduced in [38] as a signal recovery algorithm. Other names for OMP are *Greedy Least Squares* regression [59] and *Forward Greedy Selection*. Some literature also incorrectly refers to OMP as Orthogonal Least Squares [14, 23], but they are two slightly different algorithms, as also discussed in greater detail in [9].

Origin of OMP. OMP was first proposed in 199?[38] to recover an encoded signal $x \in \mathbb{R}^n$ from a noisy compressed vector $y \in \mathbb{R}^m$, where $m \ll n$. A large signal x has been compressed using a compression matrix $A \in \mathbb{R}^{m \times n}$ as

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

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

Interestingly, although there are infinitely many solutions x', there is only one solution $\hat{x}$ which is maximally sparse [34], meaning that of all x' such that $Ax' = y$, the solution $\hat{x}$ contains the fewest non-zero coefficients. To find $\hat{x}$, we need to solve the problem

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

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

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

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

Both problems in Equation 6.3 and Equation 6.4 are in fact NP-hard, see [22] Problem MP-5, page 246. Therefore, we know that there most likely does not exist an efficient algorithm to retrieve $\hat{x}$.

Therefore, instead of trying to find the $\hat{x}$, researchers have developed algorithms that approximately solved the problem in Equation 6.4. An effective method to approximately solve this sparse signal recovery problem is the aforementioned OMP algorithm, where we iteratively add the element that maximizes the correlation with the current residual to the support of $\hat{x}$.

OMP in the regular regression setting. Apart from the signal processing community, OMP is now also widely used in machine learning, even in for example text classification [48] and image reconstruction [24]. Suppose that we have a data matrix $X \in \mathbb{R}^{T \times p}$ of response variables and an explanatory variable $y \in \mathbb{R}^T$. Suppose we want to perform a linear regression in the form of

$$y = Xw + \varepsilon. \qquad (6.5)$$

For this, we need to estimate our coefficient vector w.

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

Nevertheless, OMP seems suitable to recover sparse coefficient vectors w. The algorithm now approximately solves the NP-hard problem

$$\hat{w} = \min_w ||w||_0 \text{ subject to } w \in \{w \mid ||Xw - y||_2 \le \epsilon\}. \qquad (6.6)$$

Equation 6.6 is in fact equivalent to Equation 6.4 apart from different notation to accommodate the machine learning regression setting.

The orthogonal matching pursuit Algorithm. Now that we have explained the origin of Orthogonal Matching Pursuit, let us now discuss how Orthogonal Matching Pursuit manages to recover a sparse coefficient vector w.

We start with an empty coefficient vector $w^{(0)}$, where the number in the superscript denotes the current iteration of the algorithm. Furthermore, we denote $\Lambda^{(0)}$ as the support of $w^{(0)}$, corresponding to all non-zero entries in $w^{(0)}$. This support set is initially empty. We then calculate which covariate variable $X_i \in \mathbb{R}^T$, or equivalently which column i of X is most correlated with the current residual, which can be calculated as $r^{(0)} = y - Xw^{(0)} = y$. With most correlated, we mean that the angle between the two vectors X_i and $r^{(0)}$ that is closest to either 0 degrees or 180 degrees. The index of the corresponding column vector can be calculated as

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

As $||r^{(0)}||_2$ is the same for all i variables, we do not need to include this quantity in Equation 6.7 to find the column $i^{(k)}$ corresponding to the column covariate $X_{i^{(k)}}$ most correlated with the current residual.

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

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

Having a new estimate of $w^{(1)}$, we can update the residual as $r^{(1)} = y - Xw^{(1)}$ and again find the covariate most correlated with the current residual, add the corresponding index to the support set, and re-estimate the coefficient vector. We repeat this process for multiple iterations $k = 2, \ldots$, until no significant gain is made, that is, when all correlations are smaller than some threshold ϵ. If even adding the most suitable column of X as a response variable does not yield the desired gain, we stop the algorithm and output our coefficient vector $w^{(k-1)}$, consisting of $k - 1$ non-zero coefficients.

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

Algorithm 6.2: Original Orthogonal Matching Pursuit Algorithm

Input: Covariates $X = [X_1, \ldots, X_p] \in \mathbb{R}^{T \times p}$, response variable $y \in \mathbb{R}^T$, threshold ϵ.

Output: A coefficient vector $w \in \mathbb{R}^p$.

1: $\tilde{X}_j \leftarrow X_j / \|X_j\|_2$, the normalized basis for $j = 1, \ldots, p$.
2: $\Lambda^{(0)} \leftarrow \varnothing$
3: $w^{(0)} \leftarrow \mathbf{0} \in \mathbb{R}^p$
4: $r^{(0)} \leftarrow y - X w^{(0)}$
5:
6: **for** $k = 1, 2, \ldots$ **do**
7:      $i^{(k)} \leftarrow \arg\max_i \left| \langle \tilde{X}_i, r^{(k-1)} \rangle \right|$ $\triangleright$ find column X_i most correlated with $r^{(k-1)}$
8:      **if** $\max_i \left| \langle \tilde{X}_i, r^{(k-1)} \rangle \right| > \epsilon$ **then** $\triangleright$ update $\Lambda^{(k-1)}, w^{(k-1)}, r^{(k-1)}$
9:          $\Lambda^{(k)} \leftarrow \Lambda^{(k-1)} \cup \{i^{(k)}\}$
10:          $X^{(k)} \leftarrow X$, with column $X_j^{(k)}$ all zeros if $j \notin \Lambda^{(k)}$.
11:          $w^{(k)} \leftarrow \left(X^{(k)} X^{(k)^T} \right)^{-1} X^{(k)^T} y$ $\triangleright$ OLS solution restricted to $\Lambda^{(k)}$
12:          $r^{(k)} \leftarrow y - X w^{(k)}$
13:      **else**
14:          **return** $w^{(k-1)}$
15:      **end if**
16: **end for**

Casting orthogonal matching pursuit to our setting. In the previous paragraph, we have shown how Orthogonal Matching Pursuit can be used to obtain a sparse coefficient vector w in the regression setting

$$y = Xw + \varepsilon. \tag{6.9}$$

However, as you can see, Algorithm 6.2 is suitable for a coefficient vector $w \in \mathbb{R}^p$ and a response vector $y \in \mathbb{R}^T$. However, we consider a coefficient matrix $W \in \mathbb{R}^{p \times p}$ and a response matrix $\mathbf{X}_{2:T,\cdot} \in \mathbb{R}^{T-1 \times p}$ consisting of p response variables. Recall that in our scenario, we assume that,

$$\mathbf{X}_{2:T,\cdot} = \mathbf{X}_{1:T-1,\cdot} W + \varepsilon_{2:T}. \tag{6.10}$$

Hence, we require some modifications before we can apply OMP to our setting. We will discuss three possible modifications.

Approach 1: p independent OMP methods One approach is to separately apply the OMP algorithm p times, once for each response variable. In our setting, we have for one of the j time series that

$$y_j = \mathbf{X}_{2:T,j} \in \mathbb{R}^{T-1}, \qquad X = \mathbf{X}_{1:T-1,\cdot} \in \mathbb{R}^{T-1 \times p}, \qquad w_j = w_{\cdot,j} \in \mathbb{R}^p. \tag{6.11}$$

Now, can apply the OMP algorithm described in Algorithm 6.2 seperately on all p time series, thereby obtaining the p columns $w_{\cdot,j}$ of W. However, we lose the dependency between the p columns. This becomes impractical later as we require the coefficient matrix W to be acyclic. For this, we need to consider the coefficients of all columns simultaneously.

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

$$\mathbf{X}_{2:T,\cdot} - \mathbf{X}_{1:T-1,\cdot} W = \varepsilon_{2:T} \in \mathbb{R}^{T-1 \times p} \iff y' - X'w' = \text{vec}(\varepsilon_{2:T}), \tag{6.12}$$

where the the vectorized dimensions will be

$$y' \in \mathbb{R}^{(T-1)\cdot p}, \qquad X' \in \mathbb{R}^{(T-1)\cdot p \times p^2}, \qquad w' \in \mathbb{R}^{p^2}. \tag{6.13}$$

Rewrite response matrix $\mathbf{X}_{2:T,\cdot}$ to response vector y'. To rewrite our response matrix $\mathbf{X}_{2:T,\cdot} \in \mathbb{R}^{(T-1)\times p}$ to a response vector $y' \in \mathbb{R}^{(T-1)\cdot p}$, We will vertically stack the p columns to get a single column vector $y \in \mathbb{R}^{p\cdot(T-1)}$. In mathematical notation,

$$y' = \mathrm{vec}(\mathbf{X}_{2:T,1}, \mathbf{X}_{2:T,2}, \ldots, \mathbf{X}_{2:T,p-1}, \mathbf{X}_{2:T,p}) \tag{6.14}$$

$$= \Big(\underbrace{X_{2,1}, \ldots, X_{T,1}}_{\mathbf{X}_{2:T,1}}, \underbrace{X_{2,2}, \ldots, X_{T,2}}_{\mathbf{X}_{2:T,2}}, \ldots, \underbrace{X_{2,p-1}, \ldots, X_{T,p-1}}_{\mathbf{X}_{2:T,p-1}}, \underbrace{X_{2,p}, \ldots, X_{T,p}}_{\mathbf{X}_{2:T,p}} \Big)^T. \tag{6.15}$$

Rewrite coefficient matrix W to coefficient vector w'. Secondly, we will vectorize our coefficient matrix $W \in \mathbb{R}^{p\times p}$ to a coefficient vector $w' \in \mathbb{R}^{p^2}$ by vertically stacking the columns $w_{\cdot,j}$. Therefore, we have

$$w' = \mathrm{vec}(w_{1,\cdot}, w_{2,\cdot}, \ldots, w_{p-1,\cdot}, w_{p,\cdot}) \tag{6.16}$$

$$= (w_{11}, \ldots, w_{p1}, \ldots, w_{1p}, \ldots, w_{pp})^T. \tag{6.17}$$

Rewrite covariate matrix $\mathbf{X}_{1:T-1,\cdot}$ to covariate matrix X'. Lastly, we need to transform our covariates $\mathbf{X}_{1:T-1,\cdots}$ to match our response vector y'. Therefore, our new covariate matrix must be of the form $X' \in \mathbb{R}^{(T-1)\cdot p \times p^2}$ such that

$$\mathrm{vec}\left(\mathbf{X}_{1:T-1,\cdot} W\right) = X'w'. \tag{6.18}$$

We achieve this by repeating the covariate matrix $\mathbf{X}_{1:T-1,\cdot} \in \mathbb{R}^{(T-1)\times p}$ a total of p times along the diagonal. The other entries in the matrix X' are all equal to zero. In matrix notation, we get

$$X' = \begin{pmatrix} \mathbf{X}_{1:T-1,\cdot} & \mathbf{O} & \cdots & \mathbf{O} & \mathbf{O} \\ \mathbf{O} & \mathbf{X}_{1:T-1,\cdot} & \cdots & \mathbf{O} & \mathbf{O} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \mathbf{O} & \mathbf{O} & \cdots & \mathbf{X}_{1:T-1,\cdot} & \mathbf{O} \\ \mathbf{O} & \mathbf{O} & \cdots & \mathbf{O} & \mathbf{X}_{1:T-1,\cdot} \end{pmatrix} \in \mathbb{R}^{(T-1)\cdot p \times p^2}. \tag{6.19}$$

A more concise mathematical notation for Equation 6.19 using the Kronecker product is

$$X' = I_p \otimes \mathbf{X}_{1:T-1,\cdot} \in \mathbb{R}^{(T-1)\cdot p \times p^2}. \tag{6.20}$$

To conclude, we see that we have rewritten our matrix setting to a vectorized setting.

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

> **Example 6.2** Rewrite our matrix formulation to a vectorized setting.
>
> To explain the rewriting of the our matrix notation to vector notation more clearly, we will consider the following example. Suppose that we have four samples of two variables in our data matrix X. Hence, $T = 4$ and $p = 2$, and $X \in \mathbb{R}^{4\times 2}$.
>
> As our response variables, we will use the first $T-1$ time steps of $\mathbf{X}$, which we define as $\mathbf{X}' \in \mathbb{R}^{T-1\times p}$. Furthermore, we use the last $T-1$ time steps as explanatory variables, which we define as $\mathbf{Y} \in \mathbb{R}^{T-1\times p}$.
>
> The data matrix $\mathbf{X}$ that we will be using in this example, as well as the corresponding

covariate matrix $\mathbf{X}'$ and response matrix $\mathbf{Y}$ have been given in Equation 6.21

$$\mathbf{X} = \begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \\ 7 & 8 \end{pmatrix}, \qquad \mathbf{X}' = \begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix}, \qquad \mathbf{Y} = \begin{pmatrix} 3 & 4 \\ 5 & 6 \\ 7 & 8 \end{pmatrix}. \tag{6.21}$$

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

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

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

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

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

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

Approach 3: Modifying the OMP algorithm itself. Approach 2 suffers from the drawback that the dimensions of our data matrix X' will get unnecessarily big, and the information in $\mathbf{X}'$ is redundantly copied a total of p times. Furthermore, the required memory to allocate the X' is cubic with respect to the number of variables, which is rather wasteful.

Therefore, it might be more practical to modify the OMP algorithm such that we can regress on a matrix of coefficients W, rather than a vector of coefficients w. This was accomplished by adding an additional dimension. Rather than an index i, we now have two indices i and j, corresponding to the ith row of the jth column of W, or equivalently, the influence of the ith covariate on the jth response variable.

We see that rather than one support set $\Lambda^{(k)}$, one coefficient vector $w^{(k)}$, and one residual $r^{(k)}$, we now require p of each, one for every response variable j. We will use the subscript j to denote to which response variable they correspond.

Additionally, as we now have p residuals $r_i^{(k)}$ as well, we cannot exclude $\|r_i; (k)\|_2$ as in Equation 6.7 to find the covariate most correlated with the residual of the corresponding response variable. Therefore, we are now interested in finding the covariate X_i and the residual $r_j^{(k)}$ of the jth response variable such that their correlation is maximized,

$$\left(i^{(k)}, j^{(k)} \right) = \arg\max_{i,j} \frac{\left| \langle X_i, r^{(k)} \rangle \right|}{\|X_i\|_2 \|r_j^{(k)}\|_2} = \arg\max_{i,j} \frac{\left| \langle \tilde{X}_i, r^{(k)} \rangle \right|}{\|r_j^{(k)}\|_2}, \qquad \text{where } \tilde{X}_i = \frac{X_i}{\|X_i\|_2}. \tag{6.22}$$

Another interesting point is that adding one index i to $\Lambda_j^{(k)}$, the support set of the jth explanatory variable at iteration k, and consequently re-estimating $w_j^{(k)}$, does not influence the other response variables in any way. The support sets $\Lambda_{j'}^{(k)}$, coefficient columns $w_{j'}^{(k)}$, and residuals $r_{j'}^{(k)}$ remain unchanged for $j' \neq j$. So we only need to re-estimate the coefficient column $w_j^{(k)}$ and recompute the residuals $r_j^{(k)}$. Luckily, we can do this in the exact same way as in the standard OMP algorithm described in Algorithm 6.2, where we had a single response variable.

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

Algorithm 6.3: Orthogonal Matching Pursuit Algorithm for Multiple Reponse Variables

Input: Covariate matrix $X = [X_1, \ldots, X_p] \in \mathbb{R}^{T-1 \times p}$, response matrix $Y = [Y_1, \ldots, Y_p] \in \mathbb{R}^{T-1 \times p}$, threshold ϵ.
Output: A coefficient matrix $W \in \mathbb{R}^{p \times p}$.

1: $\tilde{X}_j \leftarrow X_j / \|X_j\|_2$, the normalized basis for $j = 1, \ldots, p$.
2: $\Lambda_j^{(0)} \leftarrow \varnothing, j = 1, \ldots, p$
3: $w_j^{(0)} \leftarrow \mathbf{0} \in \mathbb{R}^p, j = 1, \ldots, p$
4: $r_j^{(0)} \leftarrow Y_i - X w_i^{(0)}, j = 1, \ldots, p$
5:
6: **for** $k = 1, 2, \ldots$ **do**
7:      $(i^{(k)}, j^{(k)}) \leftarrow \arg\max_{i,j} \left| \langle \tilde{X}_i, r_j^{(k-1)} \rangle \right| / \|r_j^{(k-1)}\|_2$ ▷ get most correlated pair
8:      **if** $\max_{i,j} \left| \langle \tilde{X}_i, r_j^{(k)} \rangle \right| / \|r_j^{(k-1)}\|_2 > \epsilon$ **then**
9:          $\Lambda_j^{(k)} \leftarrow \Lambda_j^{(k-1)} \cup \{i^{(k)}\}$ ▷ update $\Lambda_j^{(k-1)}$, $w_j^{(k-1)}$, $r_j^{(k-1)}$ for j
10:          $X^{(k)} \leftarrow X$, with column $X_i^{(k)}$ all zeros if $i \notin \Lambda^{(k)}$.
11:          $w_j^{(k)} \leftarrow \left(X^{(k)} X^{(k)^T} \right)^{-1} X^{(k)^T} Y_j$ ▷ OLS solution restricted to $\Lambda_j^{(k)}$
12:          $r_j^{(k)} \leftarrow Y_j - X w^{(k)}$
13:          $\Lambda_{j'}^{(k)} \leftarrow \Lambda_{j'}^{(k-1)}$ for $j' \neq j$ ▷ Copy over $\Lambda_{j'}^{(k-1)}$, $w_{j'}^{(k-1)}$, $r_{j'}^{(k-1)}$ for $j \neq j'$
14:          $w_{j'}^{(k)} \leftarrow w_{j'}^{(k-1)}$ for $j' \neq j$
15:          $r_{j'}^{(k)} \leftarrow r_{j'}^{(k-1)}$ for $j' \neq j$
16:      **else**
17:          **return** the matrix $W^{(k-1)} = \left(w_1^{(k-1)}, w_2^{(k-1)}, \ldots, w_p^{(k-1)} \right)$
18:      **end if**
19: **end for**

Enforce DAG-ness of W. Now that we are able to apply the Orthogonal Matching Pursuit algorithm described in Algorithm 6.3 to multiple response variables simultaneously, there is still one hurdle to overcome that is central in this thesis. The graph induced by the coefficient matrix W must be acyclic. However, in its current form, Algorithm 6.3 does not enforce acyclicity.

Luckily, acyclicity is quite easily enforced when using forward iterative approaches. At each iteration k, when we add a new coefficient or arc to our matrix W, we can first check if adding this arc (i, j) will create a cycle in the graph. If adding (i, j) indeed introduces a cycle, then we have two options:

1. We terminate the algorithm just before the first cycle was created. We then return the matrix $W^{(k-1)}$. However, this may result in a premature termination of the procedure. If the coefficient corresponding to the largest correlation introduces a cycle, then there could still be other important coefficients that do not introduce cycles.

2. We revert back to the matrix $W^{(k-2)}$. We exclude the aforementioned arc (i, j) that introduced the cycle from ever being considered again. We can argue that arc (i, j) was the least important arc in the cycle anyhow. After this arc has been disregarded, we continue our algorithm and find the arc corresponding to the most correlated $(X_i, r_j^{(k-1)})$ pair that does not introduce a cycle. The algorithm continues until all candidate arcs do not yield a sufficient gain or when there are only arcs left that will violate the acyclicity assumption of W.

To enforce acyclicity, we can keep track of additional sets F_j that contains all entries i such that adding arcs (i, j) will introduce a cycle in the graph. We can also regard this set F_j as all forbidden entries i that are excluded from ever being added to $\Lambda_j^{(k)}$, again as they introduce a cycle.

To check for a cycle, we can do a topological sort on W or, its support matrix $\mathbf{\Lambda}^{(k)} = \left(\Lambda_1^{(k)}, \Lambda_2^{(k)}, \ldots, \Lambda_p^{(k)} \right)$, so that we do not need to estimate $W^{(k)}$ before knowing if it will introduce a cycle.

If a cycle is indeed introduced, we remove i from Λ_j and add i to the set of forbidden arcs F_j. We decrement k so that $W^{(k)}$ will still correspond to the matrix with k estimated coefficients. Now, instead of looking for any $(X_i, r_j^{(k-1)})$ pair that maximize Equation 6.22, we are now interested in finding the pair such that $i \notin F_j$ as well. The pseudocode for this modified version of OMP, abbreviated as `DAG-OMP`, is described in Algorithm 6.4.

Algorithm 6.4: `DAG-OMP`

Input: Covariates $X = [X_1, \ldots, X_p] \in \mathbb{R}^{T-1 \times p}$, response variables $Y = [Y_1, \ldots, Y_p] \in \mathbb{R}^{T-1 \times p}$, threshold ϵ.
Output: An acyclic coefficient matrix $W \in \mathbb{R}^{p \times p}$.

1: $\tilde{X}_j \leftarrow X_j / ||X_j||_2$, the normalized basis for $j = 1, \ldots, p$.
2: $\Lambda_i^{(0)} \leftarrow \varnothing, i = 1, \ldots, p$
3: $w_i^{(0)} \leftarrow \mathbf{0} \in \mathbb{R}^p, i = 1, \ldots, p$
4: $r_i^{(0)} \leftarrow Y_i - X w_i^{(0)}, i = 1, \ldots, p$
5: $F_i^{(0)} \leftarrow \varnothing, i = 1, \ldots, p$
6:
7: **for** $k = 1, 2, \ldots$ **do**
8:      $(i, j) \leftarrow \underset{(i,j) : i \notin F_j^{(k-1)}}{\arg\max} |\langle \tilde{X}_i, r_j^{(k-1)} \rangle| / ||r_j^{(k-1)}||_2$ $\triangleright$ get most correlated pair
9:      **if** $\underset{(i,j) : i \notin F_j^{(k-1)}}{\max} \left| \left\langle \tilde{X}_i^T, r_j^{(k-1)} \right\rangle \right| / ||r_j^{(k-1)}||_2 > \epsilon$ **then**
10:          $\Lambda_j^{(k)} \leftarrow \Lambda_j^{(k-1)} \cup \{i\}$. $\triangleright$ try adding arc (i, j)
11:          **if** $G\left(\left(\Lambda_1^{(k)}, \Lambda_2^{(k)}, \ldots, \Lambda_p^{(k)} \right) \right)$ is not acyclic **then**
12:              $X^{(k)} \leftarrow X$, with column $X_i^{(k)}$ all zeros if $i \notin \Lambda^{(k)}$.
13:              $w_j^{(k)} \leftarrow \left(X^{(k)} X^{(k)^T} \right)^{-1} X^{(k)^T} Y_j$ $\triangleright$ OLS solution restricted to $\Lambda_j^{(k)}$
14:              $r_j^{(k)} \leftarrow Y_j - X w^{(k)}$
15:              $\Lambda_{j'}^{(k)} \leftarrow \Lambda_{j'}^{(k-1)}$ for $j' \neq j$ $\triangleright$ Copy over $\Lambda_{j'}^{(k-1)}, w_{j'}^{(k-1)}, r_{j'}^{(k-1)}$
16:              $w_{j'}^{(k)} \leftarrow w_{j'}^{(k-1)}$ for $j' \neq j$
17:              $r_{j'}^{(k)} \leftarrow r_{j'}^{(k-1)}$ for $j' \neq j$
18:              $F_{j'}^{(k)} \leftarrow F_{j'}^{(k-1)}$ for $j' \neq j$
19:          **else** $\triangleright$ Exclude arc (i, j) from being considered again
20:              $\Lambda_j^{(k)} \leftarrow \Lambda_j^{(k-1)} \setminus \{i\}$.
21:              $F_j^{(k)} \leftarrow F_j^{(k-1)} \cup \{i\}$.
22:              $k \leftarrow k - 1$
23:          **end if**
24:      **else**
25:          **return** $W^{(k-1)} = \left(w_1^{(k-1)}, w_2^{(k-1)}, \ldots, w_p^{(k-1)} \right)$
26:      **end if**
27: **end for**

Applying `DAG-OMP` on data generated by an acyclic W^*. Having gone quite some lengths to translate the orthogonal matching pursuit algorithm to our setting, let us consider the effectiveness of this iterative approach. Firstly, let us consider the same recurring data matrix $\mathbf{X}$ from Example 5.6 and Example 5.8.

Example 6.3 `DAG-OMP` on three variables.

Let us apply `DAG-OMP` on the same three-dimensional setting as in Example 5.6 and Example 5.8, where each time series consists of one hundred time steps each, and the data generating matrix was

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

For ease of notation, we will define $X = \mathbf{X}_{1:T-1,\cdot} \in \mathbb{R}^{T-1 \times p}$ as the covariate matrix and $Y = \mathbf{X}_{2:T,\cdot} \in \mathbb{R}^{T-1 \times p}$ as the response matrix. Furthermore, we will use the matrix $\mathbf{C}^{(k)} \in \mathbb{R}^{3 \times 3}$ to denote the correlations computed at the end of iteration k, and $W^{(k)}$ to denote the coefficient matrix at iteration k,

$$c_{ij}^{(k)} = \frac{\left| \langle X_i, r^{(k-1)} \rangle \right|}{\|X_i\|_2 \|r_j^{(k-1)}\|_2}, \qquad W^{(k)} = \begin{pmatrix} w_1^{(k)} & w_2^{(k)} & w_3^{(k)} \end{pmatrix}. \tag{6.23}$$

If arc (i,j) has been included or explicitly excluded, we will leave the corresponding entry c_{ij} blank. Computing which covariate X_i is most correlated with the residual of the response variable Y_j results the correlations $\mathbf{C}^{(0)}$ given in the left-hand side of Equation 6.24.

$$\mathbf{C}^{(0)} = \begin{pmatrix} 0.53 & 0.62 & 0.21 \\ 0.23 & 0.71 & 0.43 \\ 0.07 & 0.36 & 0.50 \end{pmatrix} \Rightarrow W^{(1)} = \begin{pmatrix} 0.00 & 0.00 & 0.00 \\ 0.00 & 0.71 & 0.00 \\ 0.00 & 0.00 & 0.00 \end{pmatrix} \tag{6.24}$$

Arc $(2,2)$ corresponds to the largest residual correlation. Estimating the corresponding coefficient using constrained ordinary least squares yields $w_{22} = 0.72$. Now, the updated residual correlations are given in the right-hand side of Equation 6.25.

$$\mathbf{C}^{(1)} = \begin{pmatrix} 0.53 & 0.55 & 0.21 \\ 0.23 & & 0.43 \\ 0.07 & 0.16 & 0.50 \end{pmatrix} \Rightarrow W^{(2)} = \begin{pmatrix} 0.00 & -0.64 & 0.00 \\ 0.00 & 0.57 & 0.00 \\ 0.00 & 0.00 & 0.00 \end{pmatrix} \tag{6.25}$$

The residual correlations corresponding to the first and third columns have not changed after estimating w_{22}, as the corresponding coefficients and residuals remain unaltered as well. On the other hand, the correlations in column 2 have decreased. After refitting $w_2^{(2)}$, the second coefficient $w_{22}^{(2)}$ has also changed.

Let us quickly go over the next three steps, as nothing new happens here. We add the arcs $(1,1), (3,3)$, and $(1,3)$ in that order.

$$\mathbf{C}^{(2)} = \begin{pmatrix} 0.53 & & 0.21 \\ 0.23 & & 0.43 \\ 0.07 & 0.12 & 0.50 \end{pmatrix} \Rightarrow W^{(3)} = \begin{pmatrix} 0.53 & -0.64 & 0.00 \\ 0.00 & 0.57 & 0.00 \\ 0.00 & 0.00 & 0.00 \end{pmatrix} \tag{6.26}$$

$$\mathbf{C}^{(3)} = \begin{pmatrix} & & 0.21 \\ 0.07 & & 0.43 \\ 0.05 & 0.12 & 0.50 \end{pmatrix} \Rightarrow W^{(4)} = \begin{pmatrix} 0.53 & -0.64 & 0.00 \\ 0.00 & 0.57 & 0.00 \\ 0.00 & 0.00 & 0.50 \end{pmatrix} \tag{6.27}$$

$$\mathbf{C}^{(4)} = \begin{pmatrix} & & 0.12 \\ 0.07 & & 0.30 \\ 0.05 & 0.12 & \end{pmatrix} \Rightarrow W^{(5)} = \begin{pmatrix} 0.53 & -0.64 & 0.00 \\ 0.00 & 0.57 & -0.20 \\ 0.00 & 0.00 & 0.40 \end{pmatrix} \tag{6.28}$$

At this stage, we have recovered all true coefficients of W^*.

Now, suppose that our threshold value for ϵ would have been between 0.12 and 0.30, we would have perfectly recovered the true support of W^* and no arc more. However, knowing such a threshold value beforehand is generally not possible. Therefore, let us simply continue until we have estimated a full acyclic coefficient matrix, which consists of $p(p+1)/2 = 6$ coefficients.

Furthermore, we did not need to enforce acyclicity of $W^{(k)}$ so far. At this iteration, however, the residuals are

$$\mathbf{C}^{(6)} = \begin{pmatrix} & & 0.03 \\ 0.07 & & \\ 0.05 & 0.12 & \end{pmatrix} \tag{6.29}$$

The correlation corresponding to arc $(3,2)$ is maximal. However, arc $(2,3)$ is already included in our structure, so adding arc $(2,3)$ would introduce a cycle. Hence, we exclude $(3,2)$ from consideration, and consider the second-largest entry, corresponding to arc $(1,2)$. However this arcs would also introduce a cycle, and the third-best choice, arc $(3,1)$, would introduce a cycle of length three. Therefore, our only option is to include arc $(1,3)$, yielding the final matrix

$$W_{\texttt{DAG-OMP}} = \begin{pmatrix} 0.53 & -0.64 & 0.03 \\ 0.00 & 0.57 & -0.20 \\ 0.00 & 0.00 & 0.40 \end{pmatrix}. \tag{6.30}$$

Note that if we were able to continue adding arcs, then we would remember that arcs $(2,3),(3,2)$, and $(1,2)$ would introduce cycles, and we would again not consider them. We do not need to compute the corresponding correlations for these arcs anymore, as they will never be considered.

We have seen that the `DAG-OMP` algorithm seems to be a suitable approach for estimating an acyclic coefficient matrix. Its iterative approach also gives the insight which edges were deemed most important, and which edges were deemed less important.

Furthermore, acyclicity was enforced at each iteration, but as the data generating matrix W^* was acyclic, acyclicity was only enforced when we needed to add the final arc.

Applying `DAG-OMP` on data generated by an acyclic W^*. Let us now verify how suitable `DAG-OMP` is when the data generating matrix W^* is acyclic. We have seen several continuous methods such as the gradient descent method in Section 5.1 and `DAG-LASSO` in Section 5.3 struggle with cyclic structures, as their global approaches do not consider arcs individually.

Therefore, let us consider the same recurring data matrix $\mathbf{X}$ from Example 5.7 and Example 5.9, where the data generating coefficient matrix W^* was cyclic.

Example 6.4 `DAG-OMP` on three variables with cyclic W.

Let us apply `DAG-OMP` on the same three-dimensional setting as in Example 5.7 and Example 5.9, where each time series consisted of one hundred time steps and the data generating matrix was equal to

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

corresponds to a cyclic structure.

For ease of notation, we will be using the matrix $\mathbf{C}^{(k)} \in \mathbb{R}^{3 \times 3}$ to denote the correlations computed at the end of iteration k, and $W^{(k)}$ to denote the coefficient matrix at iteration k, just as in Example 6.3. Furthermore, we will again use X to denote the covariates $\mathbf{X}_{1:T-1,\cdot}$ and Y to denote the response matrix $\mathbf{X}_{2:T,\cdot}$.

Computing which covariate X_i is most correlated with the residual of the response variable Y_j results the correlations $\mathbf{C}^{(0)}$ given in the left-hand side of Equation 6.31.

$$\mathbf{C}^{(0)} = \begin{pmatrix} 0.31 & 0.35 & 0.19 \\ 0.57 & 0.32 & 0.07 \\ 0.05 & 0.09 & 0.32 \end{pmatrix} \Rightarrow W^{(1)} = \begin{pmatrix} 0.00 & 0.00 & 0.00 \\ -0.50 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.00 \end{pmatrix}. \tag{6.31}$$

Arc $(2,1)$ corresponds to the largest residual correlation. Estimating the corresponding coefficient using constrained ordinary least squares yields $w_{21} = -0.50$. Now, the updated residual correlations are given in the left-hand side of Equation 6.32.

$$\mathbf{C}^{(1)} = \begin{pmatrix} 0.34 & 0.35 & 0.19 \\ & 0.32 & 0.07 \\ 0.09 & 0.09 & 0.32 \end{pmatrix} \Rightarrow W^{(2)} = \begin{pmatrix} 0.28 & 0.00 & 0.00 \\ -0.50 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.00 \end{pmatrix}. \tag{6.32}$$

After adding arc $(2,1)$, the reversed arc $(1,2)$ is deemed the most important. However, this would cause a cycle of length two, and therefore, we exclude arc $(1,2)$, and select the arc that corresponds to the second-largest correlation, which is arc $(1,1)$, resulting in the coefficient $w_{11} = 0.28$ In the coming iterations, we will not include the correlation corresponding to arc $(2,1)$, as we have deduced that will introduce a cycle.

Let us quickly go over the next three steps, as nothing new happens here. We add the arcs $(3,3), (2,2)$, and $(1,3)$ in that order.

$$\mathbf{C}^{(2)} = \begin{pmatrix} & & 0.19 \\ & 0.32 & 0.07 \\ 0.09 & 0.09 & 0.32 \end{pmatrix} \Rightarrow W^{(3)} = \begin{pmatrix} 0.28 & 0.00 & 0.00 \\ -0.50 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.32 \end{pmatrix}. \tag{6.33}$$

$$\mathbf{C}^{(3)} = \begin{pmatrix} & & 0.21 \\ & 0.32 & 0.01 \\ 0.09 & 0.09 & \end{pmatrix} \Rightarrow W^{(4)} = \begin{pmatrix} 0.28 & 0.00 & 0.00 \\ -0.50 & 0.32 & 0.00 \\ 0.00 & 0.00 & 0.32 \end{pmatrix}. \tag{6.34}$$

$$\mathbf{C}^{(4)} = \begin{pmatrix} & & 0.21 \\ & & 0.01 \\ 0.09 & 0.02 & \end{pmatrix} \Rightarrow W^{(5)} = \begin{pmatrix} 0.28 & 0.00 & -0.18 \\ -0.49 & 0.32 & 0.00 \\ 0.00 & 0.00 & 0.32 \end{pmatrix}. \tag{6.35}$$

To add the sixth arc to $W^{(5)}$, the corresponding correlations are

$$\mathbf{C}^{(5)} = \begin{pmatrix} & & 0.01 \\ 0.09 & 0.02 & \end{pmatrix}. \tag{6.36}$$

We see that the remaining arcs are $(3,1), (3,2)$, and arc $(2,3)$, in their order of correlation. However, arcs $(3,1)$ and $(3,2)$ both introduce a cycle, we will not select any of those, and settle for the least important arc $(2,3)$, as this arc is the only arc that does not introduce a cycle.

This yields the final matrix

$$W_{\texttt{DAG-OMP}} = \begin{pmatrix} 0.53 & -0.64 & 0.03 \\ 0.00 & 0.57 & -0.20 \\ 0.00 & 0.00 & 0.40 \end{pmatrix}. \tag{6.37}$$

In Example 6.3, we have seen that this iterative approach is indeed suitable for estimating an acyclic coefficient matrix, even when the data generating matrix W^* was was cyclic. This statement is in line with what we had argued in the beginning of this chapter. Due to its iterative approach, where one arc is considered at the time, we can effectively enforce acyclicity of the inferred structure.

Speeding up `DAG-OMP`. Although `DAG-OMP` is already quite fast, there are several techniques to increase the efficiency of OMP and therefore, also `DAG-OMP`.

Two minor improvements can be made by first remarking that that adding an arc (i, j) will only change the coefficients corresponding to the same column j. Therefore, we do not need to re-estimate the full coefficient matrix, but only the jth column. Consequently, we only need to update the residuals and correlations corresponding to the jth column. Secondly, if we decide to include the arc (i, j), where $i \neq j$, then we can already exclude the arc (j, i), as adding arc (j, i) would introduce a cycle of length two. This would approximately half the number of residual correlations we need to compute.

A more significant improvement can be made by reconsidering the constrained ordinary least squares computation in line 13 of Algorithm 6.4. Computing w_j requires three matrix multiplications, two $(p \times T - 1) \times (T - 1 \times p)$ matrix multiplication and one $(p \times p) \times (p \times p)$ multiplication. Furthermore, a $p \times p$ matrix needs to be inverted as well. We can speed this up significantly by precomputing the relevant inner products and store them in the matrices $\mathbf{\Psi}$ and $\mathbf{K}$, as is also mentioned in [37]. We define

$$\mathbf{\Psi} = \mathbf{X}_{1:T-1,\cdot}^T \mathbf{X}_{1:T-1,\cdot} \in \mathbb{R}^{p \times p}, \qquad \mathbf{K} = \mathbf{X}_{1:T-1,\cdot}^T \mathbf{X}_{2:T,\cdot} \in \mathbb{R}^{p \times p}. \tag{6.38}$$

Here, $\mathbf{\Psi}$ represents the *Gram* matrix of the columns of $\mathbf{X}_{1:T-1,\cdot}$, so $\mathbf{\Psi}_{ij} = \langle \mathbf{X}_{1:T-1,i}, \mathbf{X}_{1:T-1,j} \rangle$, where $\langle \cdot, \cdot \rangle$ represents the inner product of two vectors. In a similar fashion, we have that $\mathbf{K}_{ij} = \langle \mathbf{X}_{1:T-1,i}, \mathbf{X}_{2:T,j} \rangle$.

We can avoid the two $(p \times T - 1) \times (T - 1 \times p)$ matrix multiplications using our matrices $\mathbf{\Psi}$ and $\mathbf{K}$. Let $\mathbf{\Psi}'$ be the matrix containing the entries of $\mathbf{\Psi}$, but the entries Ψ'_{ij} and Ψ'_{ji} are set to zero if i is contained in Λ_j. Then, computing the constrained ordinary least squares solution of the j column vector $w_{\cdot,j}$ corresponds to

$$w_{\cdot,j} = \mathbf{\Psi}'^{-1} \mathbf{K_j}, \tag{6.39}$$

which only requires one $(p \times p) \times (p \times p)$ matrix multiplication computing the inverse of a $p \times p$ matrix. In all fairness, the matrices $\mathbf{\Psi}$ and $\mathbf{K}$ needed to be computed beforehand, but we can reuse these matrices for every iteration. This speeds up the computation significantly.

Now, there are additional clever tricks to speed up the algorithm that we have not pursued. Most tricks that improve the efficiency of orthogonal matching pursuit focus on speeding up the constrained least squares computations. Several approaches are discussed in [63] to compute the constrained ordinary least squares estimate more efficiently, such as using the matrix inversion lemma [29], a Cholesky decomposition approach [19], or a QR-factorization approach [52].

6.2 Using a Backwards Iterative Procedure

In Section 6.1, we have constructed an algorithm that can learn a sparse matrix representation by using an iterative forward procedure called orthogonal matching pursuit. We have first extended the orthogonal matching pursuit algorithm to multiple response variables, after which we added an additional condition that enforces the estimated coefficient matrix W to be acyclic.

Instead of forward iterative approaches, we can also construct a coefficient matrix in the opposite direction. We start with a dense coefficient matrix, and based on some deletion criterion, we prune the coefficient matrix until we have an acyclic structure.

Introduction to `DAG-OLS`. Let us describe a simple yet effective approach to estimate a suitable acyclic matrix using a backwards iterative procedure. Recall at the beginning of Chapter 4 that the maximum likelihood estimator of the data generating matrix W^* can be found using an ordinary least squares estimation. Let $\mathbf{X}_{2:T,\cdot}$ represent the first $T-1$ time steps of the data matrix $\mathbf{X}$, and let $\mathbf{X}_{2:T,\cdot}$ represent the last $T-1$ time steps of the data matrix $\mathbf{X}$. Then, the maximum likelihood estimator of $\mathbf{X}$ corresponds to the ordinary least squares (OLS) estimator,

$$W_{\text{OLS}} = \left(\mathbf{X}_{1:T-1,\cdot}^T \mathbf{X}_{1:T-1,\cdot}\right)^{-1} \mathbf{X}_{1:T-1,\cdot}^T \mathbf{X}_{2:T,\cdot}, \tag{6.40}$$

where the T in the subscript represents the number of time steps T, and the T in the superscript represents the transpose.

As we had also encountered in Chapter 4, the maximum likelihood estimator W_{OLS} does not necessarily correspond to an acyclic structure. Therefore, we must somehow modify W_{OLS} for it to represent a DAG.

A naive yet effective solution is to iteratively set the smallest nonzero coefficient to zero until the coefficient matrix is acyclic. Such an approach was also just one step of the algorithm described in [47], where in the final step they iteratively set the nonzero coefficient that is smallest in absolute value in their coefficient matrix $\hat{\mathbf{B}}$ to zero until $\hat{\mathbf{B}}$ corresponded to an acyclic structure.

We can apply this approach from [47] on the ordinary least squares estimate W_{OLS}. We first compute the ordinary least squares estimate, after which we iteratively set the nonzero coefficient w_{ij} that is smallest in absolute value to zero. As this is a procedure to transform the graph induces by the ordinary least squares estimate into a directed acyclic graph, we have called this algorithm `DAG-OLS`. The pseudocode of `DAG-OLS` is given in Algorithm 6.6.

Algorithm 6.5: `DAG-OLS`.

Input: A data matrix $\mathbf{X} \in \mathbb{R}^{T \times p}$.
Output: A coefficient matrix $W \in \mathbb{R}^{p \times p}$ such that W is a acyclic.

1: $W \leftarrow \left(\mathbf{X}_{1:T-1,\cdot}^T \mathbf{X}_{1:T-1,\cdot}\right)^{-1} \mathbf{X}_{1:T-1,\cdot}^T \mathbf{X}_{2:T,\cdot}$
2: **while** W contains a cycle **do**
3: $\quad i, j \leftarrow \arg\min_{i,j} \{|w_{ij}| \mid w_{ij} \neq 0\}$
4: $\quad w_{ij} \leftarrow 0$
5: **end while**
6: **return** W

Although this algorithm may seem quite naive, the ordinary least squares estimator, or equivalently, the maximum likelihood estimator, would have been a sensible choice if we did not need to enforce acyclicity. Interestingly, the OLS estimate W_{OLS} is a *consistent* estimator for the data generating coefficient matrix W^*. This means that when $\mathbf{X}$ has been generated according to a VAR(1) model, then the maximum likelihood estimate converges to the true matrix as the number of samples T tends to infinity, $W_{\text{OLS}} \overset{T \to \infty}{\longrightarrow} W^*$.

Therefore, when T is large enough, if a coefficient in W^* is equal to zero, then the corresponding coefficient in W_{OLS} will be close to zero as well. Therefore, by removing the smallest elements first, we will hopefully recover the true coefficient matrix W^*.

Applying `DAG-OLS` on data generated by an acyclic W^*. Encouraged by the statement on the consistency of the ordinary least squares estimate, let us see whether this consistency is also useful in a finite sample setting.

Suppose the data generating matrix W^* is acyclic. Then, we would first expect the ordinary least squares solution W_{OLS} to estimate all coefficients rather well up to some slight deviations due to noise. Now, as W^* is acyclic, the only coefficients which should have a significant contribution will form an acyclic structure which `DAG-OLS` should hopefully be able to retrieve.

Example 6.5 `DAG-OLS` on three dimensional data generated by acyclic W^*.

Let us consider the same data matrix $\mathbf{X} \in \mathbb{R}^{100 \times 3}$ as in Example 5.8, 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

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

Computing the ordinary least squares estimate, or equivalently, the maximum likelihood estimate W_{OLS}, which has been given in the left-hand side of Equation 6.41

$$W_{\text{OLS}} = \begin{pmatrix} 0.50 & -0.62 & 0.00 \\ -0.02 & 0.54 & -0.16 \\ -0.14 & -0.11 & 0.32 \end{pmatrix} \qquad W_{\text{DAG-OLS}} = \begin{pmatrix} 0.50 & -0.62 & 0.00 \\ 0.00 & 0.54 & -0.16 \\ 0.00 & 0.00 & 0.32 \end{pmatrix}. \qquad (6.41)$$

Now, applying `DAG-OLS`, we will iteratively set the non-zero coefficient that is smallest in absolute value to zero until the structure is acyclic. For this, we set the following coefficients to zero, in order, w_{13}, w_{21}, w_{32}, and lastly w_{31} to break the cycle of length three. The final matrix $W_{\text{DAG-OLS}}$ has been given in the right-hand side of Equation 6.41. We see that the solution contains all the true coefficients of W^*.

Applying `DAG-OLS` on data generated by a cyclic W^*. Let us now consider a cyclic data generating matrix W^*. Just as in Example 6.5, we expect the ordinary least squares estimate to be reasonably close to W^*. However, as W^* is acyclic, we cannot estimate all coefficients of W^*. Therefore, a large number of true arcs may be removed until we have an acyclic structure.

In fact, we will surely remove as many arcs as there are coefficients that are smaller in absolute value than the smallest coefficient corresponding to an in a cycle. We need to "break" all cycles in W_{OLS}, and `DAG-OLS` does that by setting the smallest coefficient in that cycle to zero. However, all coefficients that are smaller in absolute value will first be removed, which could make the final coefficient matrix much sparser than we might prefer.

Example 6.6 `DAG-OLS` on three dimensional data generated by a cyclic W^*.

Let us consider the same data matrix $\mathbf{X} \in \mathbb{R}^{100 \times 3}$ as in Example 5.9, consisting of three time series of one hundred time steps each. The data has been generated according to a VAR(1) model with data generating matrix

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

Computing the ordinary least squares estimate, or equivalently, the maximum likelihood estimator yields the matrix W_{OLS} given in the left-hand side of Equation 6.42.

$$W_{\mathrm{OLS}} = \begin{pmatrix} 0.28 & 0.42 & -0.18 \\ -0.50 & 0.33 & 0.01 \\ -0.02 & -0.03 & 0.32 \end{pmatrix} \qquad W_{\mathtt{DAG\text{-}OLS}} = \begin{pmatrix} 0.00 & 0.00 & 0.00 \\ -0.50 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.00 \end{pmatrix}. \qquad (6.42)$$

Now, applying $\mathtt{DAG\text{-}OLS}$, we will iteratively set the non-zero coefficient that is smallest in absolute value to zero until the structure is acyclic. For this, we set the following coefficients to zero, in order, $w_{23}, w_{31}, w_{32}, w_{13}, w_{11}, w_{33}, w_{22}$, and lastly w_{21} to break the last remaining cycle of length two. We see that the final matrix $W_{\mathtt{DAG\text{-}OLS}}$ in the right-hand side of Equation 6.42 contains only one non-zero entry, corresponding to the entry w_{12} that was largest in absolute value in W_{OLS}.

On using the coefficient size as a deletion criterion. The deletion criterion of $\mathtt{DAG\text{-}OLS}$ is to iteratively set the smallest nonzero coefficient to zero. The reasoning behind this criterion is that we want to remove the least important coefficients first before we start removing more important coefficients. As it is difficult to know how important a coefficient is, we naively use the magnitude of the coefficient as an indicator of its importance. However, as we will see, this is not always a clever choice. Nevertheless, it is a simple and efficient strategy.

A method that set coefficients to zero based on their predictive performance was the $\mathtt{DAG\text{-}LASSO}$ algorithm described in Section 5.3, where we used a penalty parameter λ to shrink the coefficients in W_λ in a continuous manner until acyclicity has been reached.

From a predictive viewpoint, $\mathtt{LASSO\text{-}DAG}$ is more capable of recovering the important coefficients of W^* than $\mathtt{DAG\text{-}OLS}$. $\mathtt{LASSO\text{-}DAG}$ shrinks all coefficients proportionally to their predictive performance. Therefore, we expect $\mathtt{LASSO\text{-}DAG}$ to be better at prioritizing important edges. This is also showcased in Example 6.7.

Example 6.7 $\mathtt{DAG\text{-}OLS}$ versus $\mathtt{LASSO\text{-}DAG}$ on two dimensional data.

Let us exemplify the shortcomings of $\mathtt{DAG\text{-}OLS}$ compared to $\mathtt{LASSO\text{-}DAG}$ which was discussed in Section 5.3. Consider the data matrix $\mathbf{X} \in \mathbb{R}^{100 \times 2}$, consisting of two time series with 100 time steps each. The data has been generated according to a VAR(1) model with coefficient matrix

$$W^* = \begin{pmatrix} 0.95 & -0.25 \\ 1.00 & 0.00 \end{pmatrix}.$$
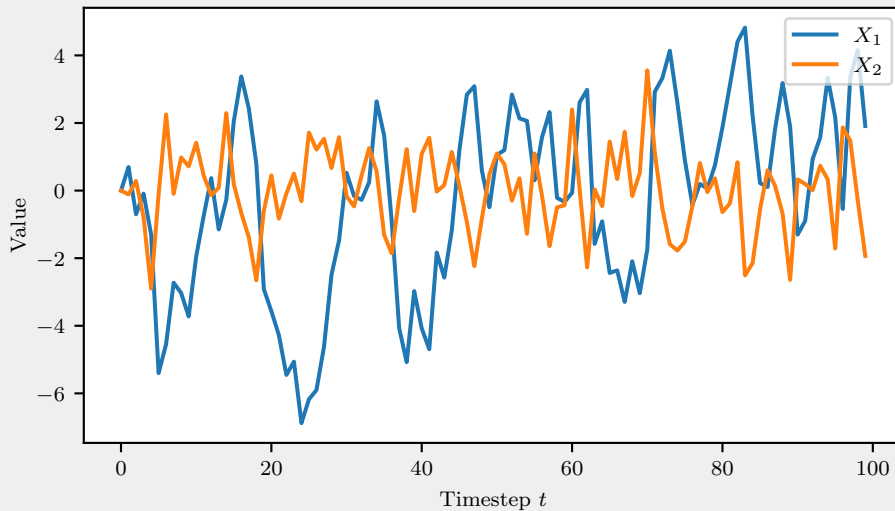
The two time series are plotted in Figure 6.2.

Figure 6.2: Visualization of the three variables X_1, X_2 of Example 6.7

Let us first remark that W^* does not correspond to an acyclic structure, as both off-diagonals contain non-zero entries. Therefore, to get an acyclic structure, we must either set w_{12} or w_{21} to zero. Preferably, we set the least important coefficient to zero.

From simply inspecting coefficient value, we would say that w_{21} is the most important coefficient. However, w_{12} corresponds to a contribution of $1.00X_{t-1,2}$ to $X_{t,1}$, whereas the value of $X_{t,1}$ is mostly determined by its autoregressive component $0.95X_{t-1,1}$. Furthermore, as $w_{22} = 0.00$, we have that the $X_{t-1,2}$ is simply the noise coefficient $\varepsilon_{t-1,2}$. Therefore, although w_{22} is the largest coefficient, its contribution is in fact quite modest. It is likely that the contribution attributed to w_{12} is greater, although its size is four times as small.

$$W_{\text{OLS}} = \begin{pmatrix} 1.03 & -0.25 \\ 1.19 & 0.01 \end{pmatrix}, \ W_{\text{DAG-OLS}} = \begin{pmatrix} 0.00 & 0.00 \\ 1.19 & 0.00 \end{pmatrix}, \ W_{\text{DAG-LASSO}} = \begin{pmatrix} 0.65 & -0.10 \\ 0.00 & 0.00 \end{pmatrix} \quad (6.43)$$

Were we to use `DAG-OLS`, we would estimate the matrix $W_{\text{DAG-OLS}}$ in the middle of Equation 6.43. Indeed, all coefficients except w_{21} have been removed. If we used `DAG-LASSO`, which shrinks the coefficients proportional to their importance, we would get the matrix $W_{\text{DAG-LASSO}}$ in the right-hand side of Equation 6.43.

We see that `DAG-LASSO` approach provides are more suitable coefficient matrix W, where the "correct" choice between w_{12} and w_{21} is made. `DAG-OLS`, however, only retains the least important non-zero coefficient of W^*.

The improvement of `DAG-LASSO` can also be seen in the solution path of the coefficients of W_λ in Figure 6.3. We see that for $\lambda = 0$, we obtain the ordinary least squares estimate, with w_{21} indeed the largest coefficient in magnitude. However, if we increase the penalty parameter, then we see that the coefficient w_{21} also decreases in size much more steeply than all other coefficients. Even the coefficient w_{12}, which was more than four times smaller in magnitude than w_{12}, "survives" longer than w_{12}, until we have an acyclic graph with penalty parameter $\lambda^* = 1.19$.
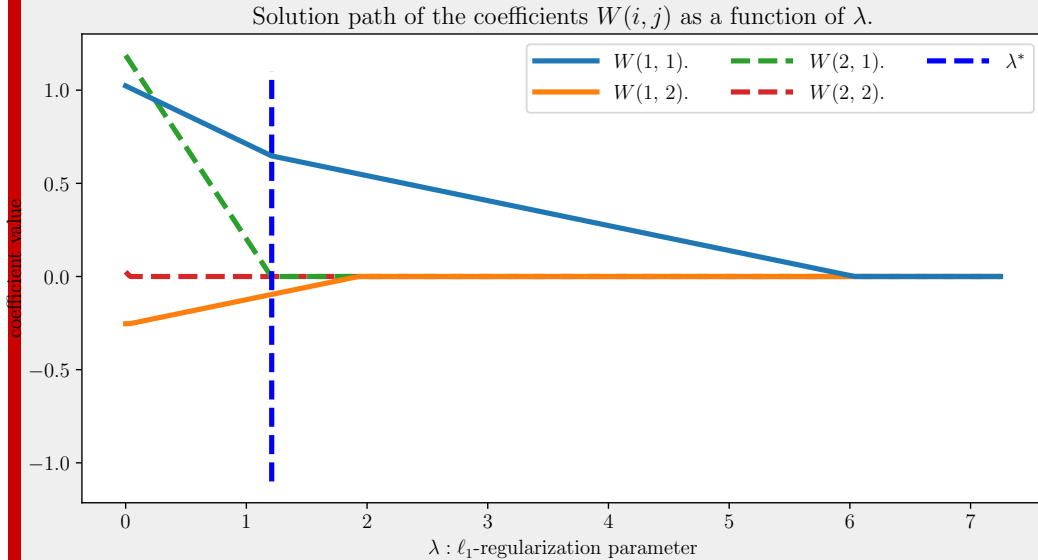

Solution path of the coefficients $W(i,j)$ as a function of λ.

Figure 6.3: Visualisation of the solution path of W_λ for Example 6.7.

This example shows that the coefficient size may not be an effective indicator of its predictive importance. This can be seen in the solution path of W, where the coefficient w_{21} has been shrunk to zero the earliest, despite being largest in magnitude in the ordinary least squares estimate.

6.3 Several Other Iterative Approaches.

So far, we have discussed two iterative approaches. Firstly we have introduced `DAG-OMP` in Section 6.1, where we iteratively add the coefficient w_{ij} such that the covariate X_i has the largest correlation with the residual of the response variable Y_j, while ensuring the structure remains acyclic. Secondly, we have described `DAG-OLS` in Section 6.2, where we start with the ordinary least squares estimate and iteratively set the smallest nonzero coefficient to zero until acyclicity has been reached.

In this section, we will be first be discussing whether we can also use these two approaches in the opposite direction. Furthermore, we will be introducing a nice trick that improves the performance of backwards iterative procedures by only removing arcs that are contained in a cycle, a so-called *backwards-violations first* approach.

Forward `DAG-OLS`. We can also employ DAG-OLS as a forwards procedure. Originally, `DAG-OLS` first estimates a full coefficient matrix W_{OLS}, after which arcs are iteratively removed until the structure is acyclic.

Now, rather than doing a backwards iterative procedure, we can use the coefficient size as a selection criterion for the forward iterative procedure. The algorithm for this is similar to `DAG-OMP`, but instead of maximizing for correlation with residuals, we would maximize the coefficient size after refitting. That is, for each possible arc (i, j) that we can add, we estimate the matrix with the additional coefficient w_{ij}, and compute the magnitude $|w_{ij}|$. We do this for all possible arcs, and select the arc (i, j) such that w_{ij} has the largest magnitude.

From the textual description, we already see that this forwards procedure feels less natural. Instead of doing one model fit, we now require to do as many model fits as arcs we are considering. At iteration k, we need to choose the best arc out of potentially $p^2 - (k - 1)$ arcs. Therefore, in the worst case, this means that we need to refit the model $\sum_{k=1}^{p(p+1)/2} p^2 - (k - 1) = \mathcal{O}(p^4)$ times. Even for just ten variables, this requires $4,015$ model fits, which is significantly more than just $p(p + 1)/2$ model fits for `DAG-OMP`.

Additionally, we have seen in Example 6.7 that the magnitude of a coefficient size may not be the best metric to assess the importance of the coefficient. Therefore, it seems that this forward variant of `DAG-OLS` seems less preferable than `DAG-OMP` in almost every aspect.

Backward `DAG-OMP`. Similarly, we could investigate `DAG-OMP` in the opposite direction, thereby doing a backwards iterative approach. We would then start with a fully estimated matrix W_{p^2}, and iteratively set the coefficient w_{ij} to zero that is most correlated with the residual after we have removed w_{ij}. However, to compute these correlations, we first need to refit the model without using this coefficient.

Just as in the forward `DAG-OLS` procedure, we require a new model fit for each arc we are considering. In the worst case, we need to consider removing one of the $p^2 - (k - 1)$ arcs at iteration k. Furthermore, it is possible that we have an acyclic matrix as late as iteration $p^2 - 1$, for example when the two most important arcs form a cycle of length two. Therefore, in the worst case, this means that we need to refit the model $\sum_{k=1}^{p^2-1} p^2 - (k - 1) = \mathcal{O}(p^4)$ times. Even for just 10 variables, this yields a total of $5,049$ model fits. Therefore, we see that the backward variant of `DAG-OMP` is not very efficient.

Nevertheless, similar approaches to a backwards orthogonal matching pursuit exist, such as the Backward Optimized Orthogonal Matching Pursuit (BOOMP) approach [5]. However, this method is designed rather to sparsify the solution obtained by a forward orthogonal matching pursuit approach. The authors in [5] propose update rules such that the less model fits are required and the model fits are computationally less demanding.

Furthermore, although the backwards procedure of `DAG-OMP` is a bit cumbersome, we still expect this backwards procedure to outperform `DAG-OLS`, as the removal criterion is more sophisticated. Therefore, if the number of variables is manageable, say only a couple of dozens of nodes, then we expect this method to be tractable as well.

6.3.1 A Backwards-Violators First Approach.

As we have seen in Example 6.6, the backwards iterative procedure `DAG-OLS` is not very suitable when the data-generating matrix W^* is cyclic. This is a shortcoming of the continuous method `DAG-LASSO` as well, as we saw in the previous chapter in Example 5.9. However, for iterative approaches, there exists a clever trick to make these backwards iterative approaches robust against such model mismatches, that is, when we need to estimate an acyclic coefficient matrix W, whereas W^* is cyclic.

Just as before, we start with the dense ordinary least squares estimate W_{OLS}, where we estimate every coefficient. In the regular backwards approaches, we would remove the least important arc. However, this means that all arcs that are less important than the arcs that violate acyclicity are removed first. This problem was also highlighted in Example 6.6, where only one coefficient of W^* was recovered.

A clever trick is to only consider removing arcs that violate the acyclicity assumption. If acyclicity is violated due to some arbitrary cycle of length two, then it is unnecessary to remove an arbitrary arc that does not violate the acyclicity whatsoever.

We see instead of looking for the least important arc, we should search for the least important arc *that violates the acyclicity assumption.* To refer back to Example 6.7, we should only consider removing the arcs that violate acyclicity, so we should have only considered removing coefficients w_{12} and w_{21}. Under this approach, but we would have kept w_{11}, as this arc did not violate our acyclicity assumption. Nevertheless, we would have still removed w_{12} although w_{21} is the more important coefficient, but that problem is inherent to `DAG-OLS` rather than the backwards iterative approach.

To employ this trick of only removing edges that violate the acyclicity assumption, we require an efficient procedure to retrieve all arcs that violate the acyclicity assumption.

Finding the arcs that violate the acyclicity constraint. An initial idea is to first count all cycles in the directed graph, and then remove all arcs that contained in at least one cycle. Efficient algorithms and their implementations exist, one example being Johnson's algorithm [31]. However, the number of cycles grows exponentially with the number of nodes. To see this, suppose we have a fully connected directed graph. To get a cycle of length k, we can start at any arbitrary node, and traverse to $k-1$ other unique nodes arbitrarily, before returning to the first node. Now, there are $p!/(p-k)!$ ways to gets such an ordered sequence of length k when there are p nodes. However, each cycle will be counted k times, once with the first node as "starter", once with the second node as "starter", etc. Therefore, the number of cycles of length k in a fully connected directed graph is equal to $p!/((p-k)!k)$. Therefore, the total number of cycles is equal to

$$\sum_{k=2}^{p} \frac{p!}{(p-k)!k}. \tag{6.44}$$

Even for a complete directed graph on 10 nodes, then the number of cycles is approximately $1.1 \cdot 10^6$, meaning that we have already exceeded a million cycles. Therefore, we see that iterating over the number of cycles is inefficient, as the running time will be exponential with respect to the number of nodes when the graph is dense.

A more tractable approach would then be to iterate over the arcs and for each arc, determine whether it belongs to a cycle. The method most readily available to us corresponds to a trace exponential function $h(W)$, which was introduced in Section 5.2. In the beginning of this chapter, we have proposed the following trace exponential function to verify whether a coefficient matrix W corresponds to an acyclic structure,

$$h(W) = \text{Tr}\left(e^{\tilde{W}}\right) - p = \sum_{k=1}^{\infty} \frac{1}{k!}\tilde{W}^k. \tag{6.45}$$

Here, $\tilde{W}$ corresponds to the matrix W with the diagonal entries set to zero.

To see whether arc (i,j) was part of any cycle, we can compare the values of $h(W)$ and $h\big(W^{-ij}\big)$, W^{-ij} represents the matrix where we have set w_{ij} to zero. Then,

$$\text{arc } (i,j) \text{ is contained in any cycle} \iff h(W) - h\big(W^{-(i,j)}\big) \neq 0. \tag{6.46}$$

Evaluating this trace exponential function $h(W)$ can be done in $\mathcal{O}(p^3)$, and efficient implementations such as [4] are readily available in standard libraries. As there are $\mathcal{O}(p^2)$ possible arcs, finding all arcs that violate acyclicity using this approach requires approximately $\mathcal{O}(p^5)$ running time.

More efficiently, we can also detect whether an arc is contained in any cycle by using an arc traversal algorithm such as depth-first search. We start by first traversing the arc (i,j) and then continue a depth-first search. If we manage to return to the node i from which we started, we know that arc (i,j) must be contained in at least one cycle. Performing this depth-first-search starting from an arc (i,j) once requires traversing all arcs. As there are p^2 possible arcs, finding all arcs that violate acyclicity requires approximately $\mathcal{O}(p^4)$ running time.

The `DAG-OLS` algorithm revisited. Let us now use this trick to overcome the issues with `DAG-OLS`. Instead of looking for the non-zero coefficient w_{ij} that is smallest in magnitude, we are now looking for the coefficient w_{ij} that is smallest in magnitude *and* violates the acyclicity constraint, in other words, is contained in a cycle. We now call this algorithm `DAG-OLS-V`, where the `V` stands for the word "violators".

Algorithm 6.6: `DAG-OLS-V`.

Input: A data matrix $\mathbf{X} \in \mathbb{R}^{T \times p}$.
Output: A coefficient matrix W such that the corresponding structure is acyclic.

1: $W \leftarrow \big(\mathbf{X}_{1:T-1}^T \mathbf{X}_{1:T-1}\big)^{-1} \mathbf{X}_{1:T-1}^T \mathbf{X}_{2:T}$
2: **while** W contains a cycle **do**
3:     $A_C \leftarrow$ set of all arcs in the graph induced by W that are contained in a cycle
4:     $i, j \leftarrow \arg\min_{(i,j) \in A_C} \{|w_{ij}| \mid w_{ij} \neq 0\}$
5:     $w_{ij} \leftarrow 0$
6: **end while**
7: **return** W

Let us consider an example that shows the improvement of `DAG-OLS-V` over `DAG-OLS`.

Example 6.8 Applying `DAG-OLS-V` on data generated by an W^*

We again consider the data matrix $\mathbf{X} \in \mathbb{R}^{100 \times 3}$ corresponding to the same cyclic coefficient matrix as in Example 6.6, where

$$W^* = \begin{pmatrix} 0.35 & 0.40 & 0.00 \\ -0.50 & 0.30 & 0.00 \\ 0.00 & 0.00 & 0.50 \end{pmatrix}, \qquad W_{\text{OLS}} = \begin{pmatrix} 0.28 & 0.42 & -0.18 \\ -0.50 & 0.33 & 0.01 \\ -0.02 & -0.03 & 0.32 \end{pmatrix}.$$

Recall that `DAG-OLS` obtained a matrix where the only remaining non-zero coefficient was $w_{21} = -0.50$.

Now, let us consider `DAG-OLS-V`. In the beginning, all coefficients except for the diagonal entries are contained in a cycle. As w_{23} is smallest in absolute value, we will set this coefficient to zero. In the next iteration, the the remaining five off-diagonal entries are still contained in a cycle, so we now set w_{31} to zero. The other four coefficients continue to be contained in a cycle, so now coefficient w_{32} will be set to zero.

Now, the arc $(1, 3)$ is not contained in a cycle anymore, so we will not no longer consider removing this arc. The remaining two arcs to consider are arc $(1, 2)$ and arc $(2, 1)$, and so we will set w_{12} to zero, after which the structure is acyclic. The remaining graph corresponds to

$$W_{\texttt{DAG-OLS-V}} = \begin{pmatrix} 0.28 & 0.00 & -0.18 \\ -0.50 & 0.33 & 0.00 \\ 0.00 & 0.00 & 0.32 \end{pmatrix}.$$

We see that `DAG-OLS-V` recovers four out of the five coefficients of W^*, which is the maximum amount of correct arcs to include in an acyclic structure, as there is a cycle of length two in W^*.

Using backwards-violators first to detect a cyclic W^*. Interestingly, we can also use this modified backwards direction to detect whether we have a model mismatch, that is, whether we are trying to recover an acyclic representation of a cyclic matrix W^*. Using a Backwards-Violators First approach, we will first remove all arcs that violate the acyclicity assumption according to their corresponding correlation with the corresponding residual. Therefore, the last arc the we remove can be considered as the most violating arc. To detect a cyclic coefficient matrix W^*, let us first describe how to acquire an ordering of importance of the coefficients using a forwards search, a backwards search, and a backwards-violators first approach.

For the forward approach, we can acquire an ordering of the coefficients by considering the matrices $W_{\texttt{F}}^{(k)}$, where k refers to the iteration number, which is equal to the number of non-zero coefficients.

For the backward approach, we can acquire a reverse ordering of the coefficients by considering the coefficient matrix $W_{\texttt{B}}^{(p^2-k)}$, where k refers to the iteration number, which is equal to the number of coefficients that have been set to zero. Therefore, the number of non-zero coefficients corresponds to $p^2 - k$. Therefore, the sequence of $(W_{\texttt{B}}^{(k)})_{k=0}^{p^2}$ corresponds to the reverse order in which the coefficients have been removed. Note, however, that not all matrices in this sequence will correspond to acyclic coefficient matrices.

For the backwards-violators first approach, we can apply the same strategy for the backwards approach, but we first only remove arcs that are contained in a cycle. Then, once we have an acyclic coefficient matrix, we iteratively remove the least important coefficient matrix just as in the backwards approach. This again yields a reverse order in which the coefficients have been removed, such that $(W_{\texttt{B-V}}^{(k)})_{k=0}^{p^2}$ corresponds to an ordering of importance.

Example 6.9 `DAG-OMP-V` when W^* is cyclic.

Let us consider a data matrix $\mathbf{X}^{100 \times 5}$ consisting of five time series, generated according to a VAR(1) model with

$$W^* = \begin{pmatrix} 0.50 & 0.00 & 0.00 & 0.00 & 0.00 \\ 0.00 & 0.50 & 0.00 & 0.00 & 0.00 \\ -0.47 & 0.47 & 0.50 & 0.83 & 0.00 \\ 0.00 & 0.00 & -0.83 & 0.50 & 0.00 \\ 0.53 & -0.55 & 0.00 & 0.00 & 0.50 \end{pmatrix}, W_{\text{OLS}} = \begin{pmatrix} 0.60 & -0.05 & 0.05 & 0.03 & 0.09 \\ 0.00 & 0.51 & 0.01 & 0.02 & 0.16 \\ -0.41 & 0.51 & 0.51 & 0.79 & 0.00 \\ 0.00 & -0.11 & -0.75 & 0.46 & -0.02 \\ 0.48 & -0.50 & -0.20 & -0.07 & 0.59 \end{pmatrix}.$$

We will use Orthogonal Matching Pursuit using the three aforementioned direction methods, Forward (`-F`), Backwards (`-B`), and Backwards-Violations first (`-B-V`). These result in the coefficient matrices in Equation 6.47.

$$
W_\text{F} = \begin{pmatrix} 0.58 & -0.05 & 0.65 & 0.01 & 0.00 \\ 0.00 & 0.51 & 0.00 & 0.00 & 0.00 \\ 0.00 & 0.51 & 0.46 & 0.79 & 0.00 \\ 0.00 & -0.11 & 0.00 & 0.47 & 0.00 \\ 0.59 & -0.50 & 0.52 & 0.08 & 0.57 \end{pmatrix}, W_\text{B} = \begin{pmatrix} 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.00 & 0.81 & 0.00 \\ 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \end{pmatrix}.
$$

$$
W_\text{B-V} = \begin{pmatrix} 0.60 & 0.00 & 0.00 & 0.00 & 0.00 \\ 0.00 & 0.46 & 0.00 & 0.00 & 0.00 \\ -0.41 & 0.51 & 0.48 & 0.80 & 0.00 \\ 0.00 & 0.00 & 0.00 & 0.46 & 0.00 \\ 0.48 & -0.55 & -0.23 & 0.00 & 0.57 \end{pmatrix}. \tag{6.47}
$$

We use the aforementioned approach to obtain an ordering of importance for each of the three procedures, which we denote by $(W_\text{F}^{(k)})_{k=0}^{p(p+1)/2}$, $(W_\text{V}^{(k)})_{k=0}^{p^2}$, and $(W_\text{B-V}^{(k)})_{k=0}^{p^2}$. We have computed the mean squared errors of using the coefficient matrix $W^{(k)}$ on $\mathbf{X}$ to see how the predictive performance changes. The mean squared errors have been plotted in Figure 6.4. Furthermore, we have also visualized until which value of k we had acyclicity of the coefficient matrices obtained by the backwards procedures.
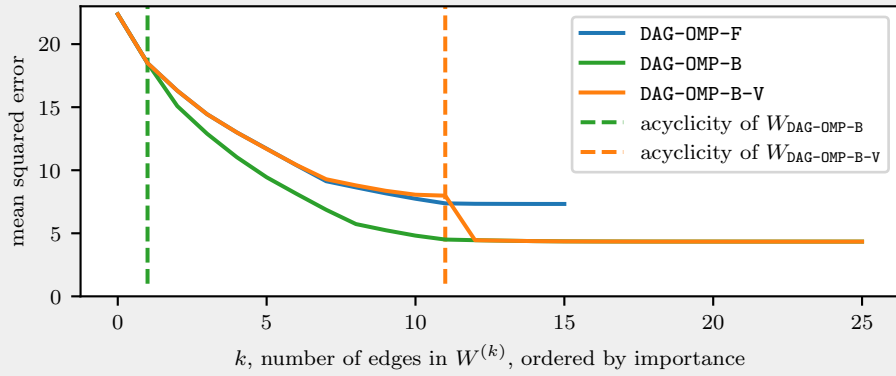

Figure 6.4: Visualization of the solution path for the coefficients of W of Example 5.9

We see that for $k = 0$, the matrices achieve a low predictive performance for all three methods, as the matrix contains no edges. As the number of coefficients increase, we see that the predictive performance increases as well. The backwards procedure seems to perform the best, but note that acyclicity is only attained for $k \leq 1$, so the backwards procedure does not yield any valid coefficient matrices for $k \geq 2$.

We see that the forward and backward-violations first method achieve a comparable predictive performance. However, at iteration $k = 12$, the mean squared error of the backwards-violations first approach significantly decreases. The reason for this lies in its violations-first procedure. It first removes all violators that are less important, but when 12 arcs remain in $W_\text{DAG-OMP-B-V}$, the choice between w_{34} and w_{43} had to be made, which meant that a large part of the predictive performance was lost.

This "jump" or "drop", depending from which side we are looking, between $k = 11$ and $k = 12$ of DAG-OMP-B-V combined with the large discrepancy in predictive performance between DAG-OMP-F and DAG-OMP-B indicates that the original structure was not acyclic.

We have seen in Example 6.9 that these iterative approaches give more insights into the importance of individual edges than the permutation based methods from Chapter 4 or the continuous approaches from Chapter 5.

6.4 Selecting a suitable number of arcs.

As mentioned in Chapter 1, we are looking for a matrix W that corresponds to a graphical model. The matrix W should be such that the induced graph $G(W)$ is a directed acyclic graph. Another crucial aspect is that the inferred structure must also be simple to interpret. Therefore, we prefer to have as few arcs as possible in the graph, yet still adequately capture how the variables influence each other. The iterative methods so far, but also the permutation-based methods in Chapter 4, tend to estimate more edges than required. We will make the objective of selecting a suitable number of arcs more concrete in Example 6.10.

Example 6.10 Selecting a suitable number of arcs in a ten-dimensional setting.

Consider the ten-dimensional dataset $\mathbf{X} \in \mathbb{R}^{100 \times 10}$ that we will be using throughout the examples in this section. The data has been generated using an acyclic coefficient matrix W^*, consisting of 25 arcs, off which 15 arcs are off-diagonal. The data set has been visualized in Figure 6.5.
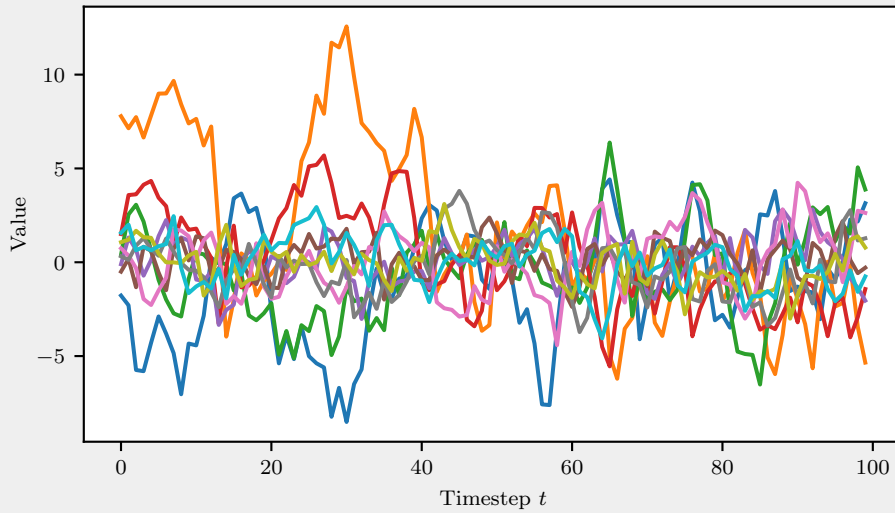

Figure 6.5: Visualization of the data matrix $\mathbf{X} \in \mathbb{R}^{100 \times 10}$ from Example 6.10.

Now, we can use an iterative approach such as `DAG-OMP` to retrieve an acyclic coefficient matrix W that hopefully closely resembles the data generating matrix W^*. If we set the threshold ϵ close to zero, then we will estimate a dense directed acyclic graph, consisting of $\binom{10}{2} = 55$ arcs, whereas the true data generating matrix only consisted of 25 arcs.

As we were interested to recover this sparse acyclic structure, we should find a way to determine which arcs are important enough, and which arcs are deemed unimportant to be included in the structure.

As a first visual approach, we can plot how the predictive performance of the coefficient matrix $W^{(k)}$ increases as we iteratively add more arcs. Let $W^{(k)}$ represent the coefficient matrix at iteration k of `DAG-OMP`, containing k non-zero coefficients, of which $k-1$ were already included in $W^{(k-1)}$. In Figure 6.6, we have plotted the corresponding mean squared error of using $X_{t-1,\cdot} W^{(k)}$ as a predictor for $X_{t,\cdot}$.
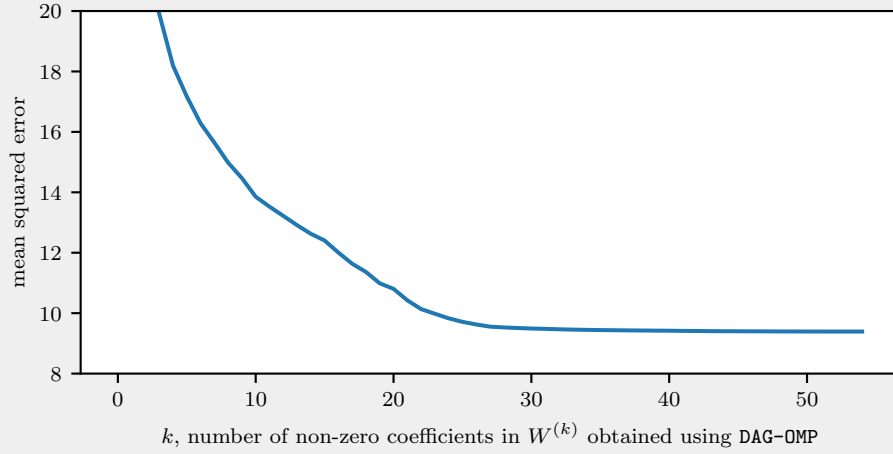
Figure 6.6: Mean squared error of W_k on $\mathbf{X}$ as a function of k, showcasing a decline until $k = 25$, after which the mean squared error seems to plateau until $k = 55$.

We see that the first arcs were indeed quite important, as the mean squared error significantly decreases. However, the predictive gain seems to decrease, and after $k = 25$, the mean squared error seems to plateau. Therefore, we see that the predictive performance of $W^{(25)}$ is almost as good as the predictive performance of $W^{(55)}$. Therefore, we see that $W^{(25)}$ might be a better choice than $W^{(55)}$, as the graphical model is less complex, in the sense that is contains much fewer arcs, while still achieving almost the same predictive performance.

Therefore, let us consider some approaches that can help us in selecting a suitable subset of arcs from an acyclic coefficient matrix W, such that the graphical model becomes less complex while still retaining a suitable predictive performance.

6.4.1 Bootstrapping

We know that predicting the coefficients of W would have been much easier if we had more data. However, obtaining more data can be quite difficult in real life or outright impossible. Luckily, there exists an interesting approach to obtain "new" data by using a bootstrapping approach.

In such a bootstrapping approach, we use our available data matrix $\mathbf{X}$ and our estimated statistical model to resample "new" data. This idea is first introduced in [20]. As there is quite some dependency in $\mathbf{X}$ since it consists of several autoregressive time-series, bootstrapping might be a suitable approach to investigate which arcs are considered important. Therefore, let us consider a suitable bootstrapping procedure for our Vector AutoRegressive models of order 1.

Bootstrapping for VAR(1) models. We consider the following bootstrapping procedure for VAR(1) models. A great book covering bootstrapping methods for VAR(1) models is "Resampling Methods for Dependent Data" by Lahiri [35].

The bootstrapping method is simple, yet effective. Suppose we have a dataset $\mathbf{X} \in \mathbb{R}^{T \times p}$ that has been generated by a VAR(1) model:

$$X_{t,\cdot} = X_{t-1,\cdot} W + \varepsilon_t, \tag{6.48}$$

where $\varepsilon_t \in \mathbb{R}^p, t = 2, \ldots, T$ are all independent and identically multivariate Gaussian random variables with zero mean and covariance matrix Σ. Furthermore, we assume that we are in a stationary setting, so we will generate $X_{1,\cdot}$ according to its stationary distribution, which is $\mathcal{N}(\mathbf{0}, B)$, where

$$\mathrm{vec}(B) = \left(I_{p^2} - W^T \otimes W^T \right)^{-1} \mathrm{vec}(\Sigma). \tag{6.49}$$

Now, the bootstrapping method described by Lahiri is as follows. First, we use any method discussed so far, to retrieve an acyclic estimator $\hat{W}$ for the true coefficient matrix W. Then, we can compute the estimated residuals

$$\hat{\varepsilon}_t = X_{t,\cdot} - X_{t-1,\cdot}\hat{W}. \tag{6.50}$$

Using the assumption that our data has been generated according to Equation 6.48, we know that

$$\hat{\varepsilon}_t = X_{t,\cdot} - X_{t-1,\cdot}\hat{W}. \tag{6.51}$$

$$= \varepsilon_t - X_{t-1,\cdot}\left(\hat{W} - W\right). \tag{6.52}$$

Assuming W^* was acyclic as well, we expect our estimate $\hat{W}$ to be "close" to the true estimate W. Therefore, we can expect $\hat{\varepsilon}_t$ to be "close" to the true residual ε_t. Then, sampling from $\{\hat{\varepsilon}\}_{t=2}^{T}$ seems to be a good alternative when there is no other way to retrieve new data $\mathbf{X}'$ or new residuals ε_t'.

As we require the residuals to be centered around zero, we will first subtract the average $\bar{\varepsilon}$ from our estimated residuals $\hat{\varepsilon}_t$,

$$\tilde{\varepsilon}_t = \hat{\varepsilon}_t - \bar{\varepsilon}, \qquad \text{where } \bar{\varepsilon} = \frac{1}{T-1}\sum_{t=2}^{T}\hat{\varepsilon}_t. \tag{6.53}$$

where $\bar{\varepsilon}$ represents the sample mean of the estimated residuals, which can be easily computed as

Next, we can start sampling *with replacement* from our set of centered residuals $\{\tilde{\varepsilon}_t\}_{t=2}^{T}$. Suppose we want to resample a VAR(1) model with T_2 samples. First, we generate T_2 "bootstrapped" residuals $\{\varepsilon_t'\}_{t=1}^{T_2}$ by repeatedly selecting one of the centered residuals with uniform probability,

$$\varepsilon_i' = \tilde{\varepsilon}_t \text{ with probability } \frac{1}{T-1} \ \forall \ t = 2, \dots, T, \ i = 1, \dots, T_2 \ . \tag{6.54}$$

Now, given these new residuals, we can generate our data X^* using our estimated coefficient matrix $\hat{W}$,

$$X_{t,\cdot}' = X_{t-1,\cdot}'\hat{W} + \varepsilon_t'^*, \qquad 2 \leq t \leq T_2, \tag{6.55}$$

with initial state $X_{1,\cdot}' = \varepsilon_1'$. The procedure of generating a new data matrix $\mathbf{X}' \in \mathbb{R}^{T_2 \times p}$, called the AutoRegressive Bootstrapping procedure, is giving in Algorithm 6.7.

Algorithm 6.7: Bootstrapping for VAR(1) models

Input: A data matrix $\mathbf{X} \in \mathbb{R}^{T \times p}$, an estimate $W \in \mathbb{R}^{p \times p}$ for the coefficient matrix.
Output: A bootstrapped data matrix $\mathbf{X}' \in \mathbb{R}^{T \times p}$.

1: compute the residuals
$$\hat{\varepsilon}_t = X_{t,\cdot} - X_{t-1,\cdot}W, \ t = 2, \dots T.$$

2: center the residuals
$$\tilde{\varepsilon}_t = \hat{\varepsilon}_t - \frac{1}{T-1}\sum_{t=2}^{T}\hat{\varepsilon}_t, \ t = 2, \dots T.$$

3: sample $2T$ new residuals ε_t' uniformly at random from $\{\tilde{\varepsilon}_t\}_{t=2}^{T}$ with replacement
4: set $X_{1,\cdot}' = \varepsilon_1'$
5: generate the rest of the bootstrapped data matrix $\mathbf{X}'$ as
$$X_{t,\cdot}' = WX_{t-1,\cdot}' + \varepsilon_t', \ t = 2, \dots, 2T.$$

6: **return** the final T time steps of $\mathbf{X}'$ to ensure stationarity.

An important note here is that the bootstrapped data matrix $\mathbf{X}'$ is not in its stationary distribution from the first measurement $X'_{1,.}$ immediately. As $X'_{1,.} = \varepsilon'_1$, these values may be much closer to zero than is expected of $\mathbf{X}'$ in its stationary distribution. Therefore, it may be wise to generate a larger bootstrap sample of say length $2T$ and then only use the last T samples. That way, we are sure to have a bootstrapped data matrix $\mathbf{X}'$ which is indeed stationary from the outset. Nevertheless, the values of the first measurement "die out" geometrically fast in auto regressive models, so we do not expect this to be a significant issue. For coefficient matrices who are close to being non-stationary, because for example the diagonals are close to one, we should generate a larger sample to make sure the last T measurements are indeed taken from a stationary state. An example showcasing this behavior is given in Example 6.11.

Example 6.11 Bootstrapping VAR(1) model until stationarity.

Suppose we want to bootstrap a new data matrix $\mathbf{X}' \in \mathbb{R}^{50 \times 3}$ based on a data matrix $\mathbf{X} \in \mathbb{R}^{50 \times 3}$ and an estimated coefficient matrix

$$\hat{W} = \begin{pmatrix} 0.9 & 0.0 & 0.0 \\ -0.5 & 0.9 & 0.0 \\ 0.5 & 0.5 & 0.9 \end{pmatrix}.$$

Now, using Algorithm 6.7, we would simply set $X'_{1,.} = \varepsilon'_1$. However, this means that the first values of $\mathbf{X}'$ are not year its stationary distribution, as the values are set artificially close to zero. Some time steps may be required until the data matrix $\mathbf{X}'$ is in its stationary distribution. This behavior is showcased in Figure 6.7.
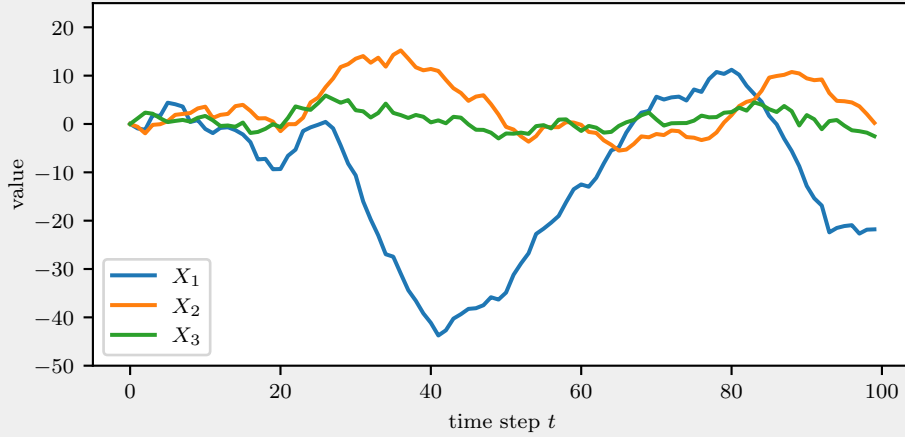

Figure 6.7: Visualization of the bootstrapped sample $\mathbf{X}'$ from Example 6.11, showcasing that $\mathbf{X}'$ is not in its stationary distribution from the very beginning.

As $X'_{1,.}$ is simply a randomly selected centered residual, the time series starts "too close to zero", and it takes some time steps for $\mathbf{X}'$ to reach its stationary distribution, until say $t = 30$. Therefore, if we want $\mathbf{X}' \in \mathbb{R}^{50 \times 3}$ to start already in its stationary distribution, we should not select the first 50 time steps of $\mathbf{X}'$, but rather the final 50 time steps to get a suitable bootstrapped data matrix X^*. For this reason, we bootstrap a data matrix $\mathbf{X}'$ consisting of $2T$ time steps, and use only the final T time steps of $\mathbf{X}'$, as is also depicted in the final line of Algorithm 6.7.

We have seen that we can sensibly generate a data matrix $\mathbf{X}'$ using an auto regressive bootstrapping procedure as described in Algorithm 6.7. To now solve the problem addressed in the introduction of this chapter in Example 6.10, we need to develop a method to use employ bootstrapping procedure to find the important arcs in W. We have developed two approaches that utilize this bootstrapping procedure. The first method relies on using bootstrapping to construct *confidence intervals* of the coefficients in W. The second method relies on using bootstrapping to verify the *recoverability* of coefficients in W.

Using bootstrapping to determine confidence intervals of W. One approach is to use this bootstrap approach to compute confidence intervals of the coefficients in the coefficient matrix W. If we bootstrap a total of N bootstrap samples $\{\mathbf{X}'^{(n)}\}_{n=1}^N$, we can use one of the developed methods to compute a total of N acyclic coefficient matrix $\{W^{(n)}\}_{n=1}^N$. We can then calculate the $\alpha/2$th and $(100 - \alpha/2)$th percentiles of each coefficient w_{ij} in W, yielding us an empirical $(1 - \alpha)\%$-confidence interval of each coefficient in W. If the value 0 is contained in the confidence interval of the corresponding coefficient, then we can argue that this coefficient is irrelevant and can therefore be set to zero.

The corresponding pseudocode has been given in Algorithm 6.8.

Algorithm 6.8: Using bootstrapping to recover confidence intervals.

Input: A data matrix $\mathbf{X} \in \mathbb{R}^{T \times p}$, an coefficient matrix estimate $W \in \mathbb{R}^{p \times p}$, a total number of bootstrap samples N, and a confidence level $0 \leq \alpha < 0.5$.
Output: $W_L, W_U \in \mathbb{R}^{p \times p}$, containing the bootstrapped $\alpha/2$ and $1 - \alpha/2$ percentiles of the coefficient matrix $\hat{W}$.

1: sample N bootstrapped data matrices $\mathbf{X}'^{(n)}, n = 1, \ldots, N$ with length T using Algorithm 6.7 with data matrix $\mathbf{X}$ and coefficient matrix W
2: estimate N the coefficient matrix $W^{(n)}$ on the bootstrapped data matrices $\mathbf{X}^{(n)}$
3: $W_L \leftarrow$ the $\alpha/2$th percentile for each coefficient from the set $\{W^{(n)}\}_{n=1}^N$.
4: $W_U \leftarrow$ the $(1 - \alpha/2)$th percentile for each coefficient from the set $\{W^{(n)}\}_{n=1}^N$.
5: **return** W^L, W^U

We can use these percentiles to determine whether an arc or coefficient in W is significantly different from zero. The coefficients of the thresholded matrix $W_{\text{threshold}}$ are then

$$w_{\text{threshold},ij} = \begin{cases} 0 & \text{if } w_{ij}^L \leq 0 \leq w_{ij}^U, \\ w_{ij} & \text{otherwise.} \end{cases} \tag{6.56}$$

To better understand this method, we have constructed a four-dimensional setting with a moderately large sample size T in Example 6.12.

Example 6.12 Using bootstrapped confidence intervals to recover important arcs.

Let us consider the four dimensional setting with 250 time steps where $\mathbf{X}$ has been generated according to a VAR(1) model, where the data generating matrix W^* has been given in the left-hand side of Equation 6.57.

Furthermore, assume that we have estimated the matrix W in the right-hand of Equation 6.57 using `DAG-OMP`. Note, however, that we could have used any method that estimates an acyclic coefficient matrix. We see that the correct arcs have been recovered, but there are also several arcs which originate from spurious correlation with the noise.

$$W^* = \begin{pmatrix} 0.85 & 0.00 & 0.00 & 0.00 \\ 0.00 & 0.85 & 0.00 & 0.00 \\ -0.28 & 0.26 & 0.85 & 0.00 \\ 0.46 & 0.00 & 0.00 & 0.85 \end{pmatrix}, \quad W = \begin{pmatrix} 0.85 & 0.00 & 0.00 & 0.00 \\ 0.03 & 0.90 & 0.00 & -0.01 \\ -0.29 & 0.19 & 0.90 & 0.01 \\ 0.50 & 0.00 & 0.00 & 0.83 \end{pmatrix} \tag{6.57}$$

The data matrix $\mathbf{X}$ has been visualized in Figure 6.8.

We will now generate 1,000 bootstrapped data matrices $\{\mathbf{X}'^{(n)}\}_{n=1}^{1,000}$ and use `DAG-OMP` again to estimate a suitable coefficient matrix for each bootstrapped data matrix, yielding 1,000 coefficient matrices $\{W^{(n)}\}_{n=1}^{1,000}$. Constructing the 10th and 90th percentile for each coefficient w_{ij} using $\{W^{(n)}\}_{n=1}^N$ yields percentile matrices in Equation 6.58.
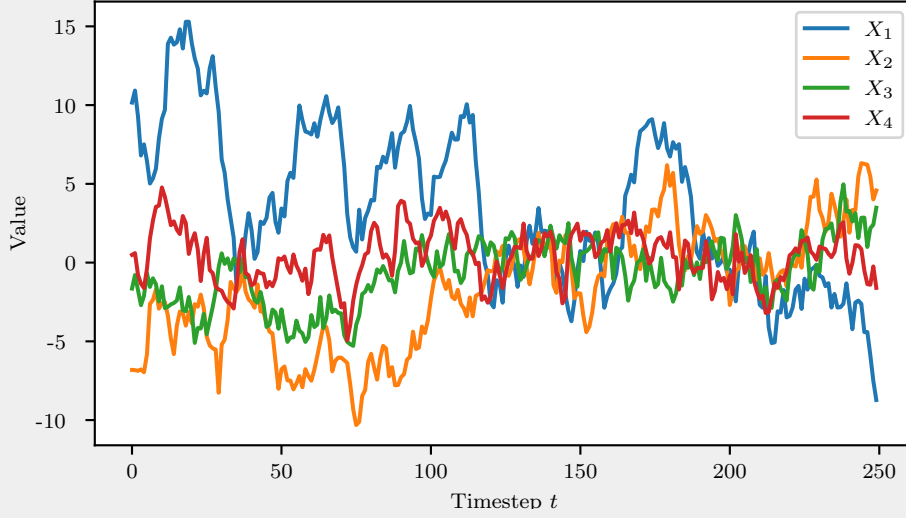

Figure 6.8: Visualization of the data matrix $\mathbf{X}$ from Example 6.12.

$$W_L = \begin{pmatrix} 0.82 & -0.03 & 0.00 & 0.000 \\ 0.08 & 0.84 & 0.00 & -0.04 \\ -0.35 & 0.14 & 0.84 & 0.05 \\ 0.43 & -0.06 & -0.06 & 0.75 \end{pmatrix}, W_U = \begin{pmatrix} 0.88 & 0.02 & 0.00 & 0.00 \\ 0.00 & 0.92 & 0.00 & 0.02 \\ -0.23 & 0.26 & 0.93 & 0.06 \\ 0.59 & 0.07 & 0.06 & 0.88 \end{pmatrix}. \quad (6.58)$$

Keeping the coefficients where the value 0 lies completely outside the percentile range yields the thresholded matrix

$$W_{\text{threshold}} = \begin{pmatrix} 0.85 & 0.00 & 0.00 & 0.00 \\ 0.00 & 0.90 & 0.00 & 0.00 \\ -0.29 & 0.19 & 0.90 & 0.00 \\ 0.50 & 0.00 & 0.00 & 0.83 \end{pmatrix},$$

which indeed retrieves all the correct entries.

Bootstrapping to verify the recoverability of coefficients. A second approach to distinguish true coefficients from noise is by inspecting the rate of *recoverability* of the coefficients. The intuition is that if an entry in the coefficient matrix W is non-zero, then we will also recover this element after bootstrapping. This method is very similar to the confidence interval method, but here we require an ordering of the edges from most important to least important.

To obtain such an ordering, we can for example use a forward iterative approach such as `DAG-OMP`, or we can iteratively remove coefficients from an acyclic structure using a backwards approach as we have done in Example 6.9.

Such an ordering consists of a sequence of coefficient matrices $W = (W_k)^{k=0p}{}^{)/2}$, where the index i represents the coefficient matrix after the ith iteration of `DAG-OMP`. This index also corresponds to the number of non-zero coefficients in W_i. The algorithm description is given in Algorithm 6.9

Algorithm 6.9: Using bootstrapping to inspect the recoverability of edges.

Input: A list of coefficient matrices $(W_k)_{k=0}^K$, representing the coefficient matrix W after the kth iteration of the `DAG-OMP` algorithm, and N, the number of bootstrap samples per coefficient matrix W_k.

Output: A list `arcs_missed`, representing the average number of arcs in W_k that did not belong to the k most important arcs after bootstrapping data matrices $\{X^{(n)}\}_{n=1}^N$ using W_k.

1: let `arcs_missed` be a list of length K
2: **for** each coefficient matrix W_k **do**
3:     use W_k to generate N bootstrapped data matrices $X^{(n)}, n = 1, \ldots, N$
4:     use `DAG-OMP` to estimate the coefficient matrix $\hat{W}_k^{(n)}$ for each $X^{(n)}$, where $\hat{W}_k^{(n)}$ is restricted to the k most important coefficients. That is, we terminate `DAG-OMP` after k iterations
5:     let B_k and $\hat{B}_k^{(n)}$ be the support matrices of W_k and $\hat{W}_k^{(n)}$, respectively
6:     compute the average number of arcs in W_k that were not recovered,

$$\texttt{arcs\_missed}[k] = \frac{1}{2N} \sum_{n=1}^N \left\| B_k - \hat{B}_k^{(n)} \right\|_0 .$$

7: **end for**
8: **return** `arcs_missed`

To better explain the method described in Algorithm 6.9, we will apply this method on the same four-dimensional data matrix $\mathbf{X}$ as described in Example 6.12.

Example 6.13 Using bootstrapping to inspect the recoverability of arcs.

Let us consider the same four-dimensional setting on $T = 250$ time steps as in Example 6.12. After applying `DAG-OMP` on $\mathbf{X}$, we obtain a sequence of coefficient matrices $(W_k)_{k=0}^{10}$, where W_k corresponds to the coefficient matrix at iteration k of the `DAG-OMP` algorithm. Note that $K = 10$, as the densest possible acyclic matrix W_K contains $p(p+1)/2 = 10$ edges.

Now, for each matrix W_k, we bootstrap 100 data matrices $\{\mathbf{X}_k^{(n)}\}_{n=1}^{100}$, use `DAG-OMP` to obtain 100 coefficient matrices $\{W_k^{(n)}\}_{n=1}^{100}$ that contain exactly k arcs. We then compute the average number of arcs that was not recovered by applying `DAG-OMP` on the bootstrapped data matrix. Plotting the average number of missed arcs as a function of k, the number of arcs, yields the plot in Figure 6.9.
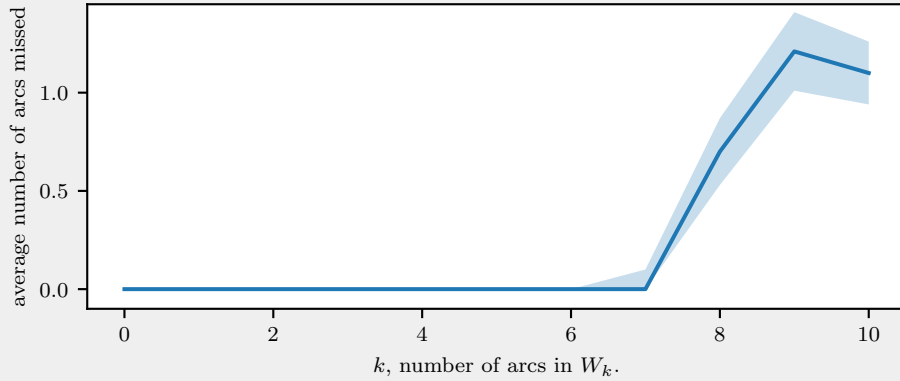

Figure 6.9: Plot of `arcs_missed` from Example 6.13.

The number of missed arcs remains close to zero until we have recovered all true arcs at $k = 7$. Intuitively, this means that these arcs were important enough to be recovered. Once we consider W_8, the matrix containing the first untrue arcs, we see that this arc is often missed, approximately 80% of the time. This implies that arcs 8, 9, and 10 were all not significant enough to be recovered, indicating that W_7 would be the most suitable matrix. Indeed, the support of W_7 is equal to the support of the matrix $W_{\text{threshold}}$ we recovered using the confidence interval approach in Example 6.12. Therefore, we see that this approach recovers all true arcs as well.

Bootstrapping performance in higher dimensions   Having explained the two bootstrapping approaches to distinguish true edges from noise in Example 6.12 and Example 6.13, let us now consider higher-dimensional settings with fewer time steps T.

Example 6.14

Let us consider a ten-dimensional time series with one hundred time steps, so $\mathbf{X} \in \mathbb{R}^{100 \times 10}$. This data matrix $\mathbf{X}$ is the same data matrix as in Example 6.10, where we have sketched the motivating example of selecting a suitable number of arcs. The data generating matrix W^* consists of 25 arcs, of which 15 are off-diagonal arcs.

Attempting to retrieve the coefficient matrix W using for example `DAG-OMP` yields a sequence of coefficient matrices $(W_k)_{k=0}^{55}$, where the index k corresponds to the number of non-zero entries in $\hat{W}_k$. Referring back to the the mean squared error achieved by each W_k in Figure 6.5, we indeed see that the mean squared error plateaus around $k = 25$, corresponding to the total number of edges in W^*.

We can first apply the bootstrapping approach to compute the 2.5th and 97.5th percentile of each coefficient in W. We have again used $1,000$ bootstrapped data matrices containing an equal number of time steps as $\mathbf{X}$. If we threshold all coefficients where the value zero lies inside its confidence interval, we managed to recover exactly all true coefficients and no other edges.

Secondly, we can use bootstrapping to investigate the recoverability of the arcs in each matrix W. The average number of arcs that were missed has been plotted in Figure 6.10 as a function of k. The first 25 arcs were deemed important enough to be recovered almost always. From $k = 25$ onwards, the number of missed arcs steadily increases, indicating that these arcs were not important enough that be recovered consistently. This also indicates that there are indeed approximately 25 true arcs, and thus W_{25} here seems the best trade-off between predictive performance and model complexity. This in accordance with our findings in Example 6.10.
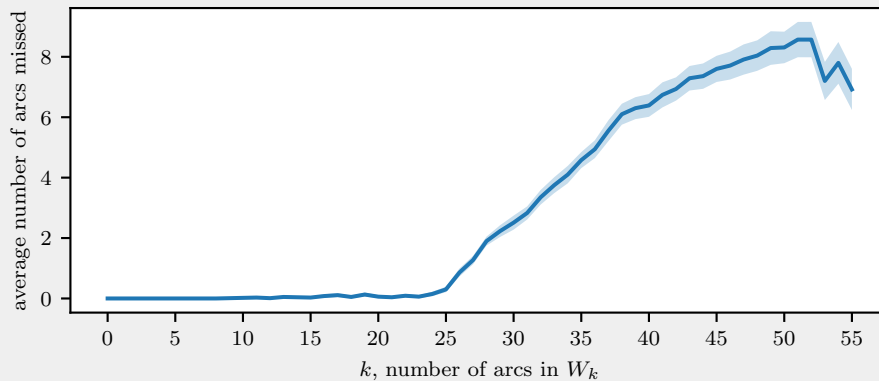

Figure 6.10: Plot `arcs_missed` from Example 6.14, showcasing the steep incline at around $k = 25$.

6.4.2 Cross-Validation

To reiterate Chapter 1, our research objective is to find an acyclic coefficient matrix characterizes how our p variables influence each other. Another crucial aspect of our research objective is that the structure is easy to interpret. Therefore, the structure must not be overly complex in the sense that it contains unnecessary arcs, nor should it have too few arcs that be do not adequately capture how the variables influence each other. Therefore, we need a way to assess whether the model is of the right complexity or equivalently, whether the model contains the right amount of arcs.

If the data were independent, then cross-validation would be a sensible choice. As first introduced in [49], cross-validation is a technique where we split the available data in a training set and a validation set. We would train on the training set, and then verify the predictive performance on the unseen validation set. For cross-validation, we use the same data matrix to randomly generate several train and validation sets, and average the performance of the different models on their corresponding validation sets. As the data is reused as both training data and validation data several times, we call such an approach cross-validation.

However, in our setting, we do not have independent data. For example, the measurements of the current time step $X_{t,\cdot}$ depend heavily on the previous time step $X_{t-1,\cdot}$. Therefore, if we split our data and train on the training data, then the validation set does not technically contain unseen data, as some of the information of the training set has "bled" into the validation set due to the aforementioned dependencies. Therefore, cross-validation when the data contains dependencies should be done carefully. Several principles of proper validation, the "do's and don'ts" of cross-validation, are explained in greater detail in [21].

Nevertheless, several cross-validation techniques for auto regressive models have been employed before, and the validity of cross-validation on AR(1) models seems reasonable under certain assumptions, such as having a stationary time series, having a consistent estimator, and making sure that the noise components form a Martingale difference sequence [7]. Therefore, we will investigate the performance of cross-validation on vector auto regressive models in this subsection.

Cross-validation using n folds. We can regard our data matrix $\mathbf{X} \in \mathbb{R}^{T \times p}$ as a dictionary $D = \{(x_t, y_t)\}_{t=2}^T$, consisting of $T - 1$ pairs of features and labels, where (x_t, y_t) corresponds to $(X_{t-1,\cdot}, X_{t,\cdot})$. Then, we can split this dictionary into two disjoint sets, a training set D_{Tr} and a validation set D_V. We can estimate an acyclic coefficient matrix, for example using `DAG-OMP`, on the pairs included in the training set D_{Tr}. In the independent setting, the validation set D_V is considered to be independent of D_{Tr}, so this dataset could be considered as unseen data. However, in our time series setting, we do not have independent data. If $(X_{1,\cdot}, X_{2,\cdot})$ and $(X_{3,\cdot}, X_{4,\cdot})$ are included in D_{Tr}, then we can exactly deduce the pair $(X_{2,\cdot}, X_{3,\cdot})$, even though we should consider this as unseen data.

This yields a coefficient matrix W, or in the scenario of a forwards iterative procedure such as `DAG-OMP`, a sequence of coefficient matrices $(W_k)_{k=0}^{p(p+1)/2}$, where k corresponds to matrix at iteration k of `DAG-OMP`, containing k non-zero coefficients in W_k. We can then assess the performance of such a matrix W_k on the validation set D_V by computing the cross-validation score,

$$\text{CV}(W_k) = \frac{1}{|D_V|} \cdot \sum_{(X_{t-1,\cdot}, X_{t,\cdot}) \in D_V} \|X_{t,\cdot} - X_{t-1,\cdot} W_k\|_2^2, \tag{6.59}$$

where $|D_V|$ denotes the cardinality of D_V, that is, number of (x_i, y_i) pairs in D_V.

In n-fold cross-validation, we randomly split our dictionary D into n folds of equal length. This yields n smaller dictionaries D_i, $i = 1, \ldots, n$. We will now use $n - 1$ folds to estimate our coefficient matrices $(W_k)_{k=1}^{p(p+1)/2}$, and use the remaining fold D_i as our validation set.

The cross validation score on the remaining fold D_i then corresponds to the mean squared error,

$$\text{CV}_i(W_k) = \frac{1}{|D_i|} \cdot \sum_{(X_{t-1,\cdot}, X_{t,\cdot}) \in D_i} \|X_{t,\cdot} - X_{t-1,\cdot} W_k\|_2^2, \tag{6.60}$$

where $|D_i|$ denotes the size or cardinality of the fold D_i.

Then, the average n-fold cross validation score is equal to

$$\mathrm{CV}(W_k) = \frac{1}{n} \sum_{i=1}^{n} \mathrm{CV}_i(W_k). \tag{6.61}$$

Leave-one-out cross-validation. Now, choosing the number of folds can be quite difficult. If we select too few folds, then our training sets will be relatively smaller, meaning our coefficient matrices will be poorer estimates. Therefore, one could argue that we want to have as many training pairs in the training set as possible.

A second another issue is how we would split our data between these folds. Assume we are looking to split K pairs into n folds of equal size, where K is divisible by n. Then, the total number of unique ways to construct these splits is equal to

$$\frac{K!}{\left(\frac{K}{n}\right)!^n N!}. \tag{6.62}$$

Even for thus 100 samples and 10 folds, there are more than $6 \cdot 10^{85}$ possible orderings. An intuition behind Equation 6.62 is that there are $K!$ ways to order the pairs. Then, within each fold, there are $\binom{K}{n}!$ ways to order the pairs. However, within one pair, the order is irrelevant. As there are k such folds, we need to divide $N!$ by $\binom{K}{n}!^n$. Lastly, as there are n folds, there are additionally $n!$ ways to order the folds, although the ordering of the folds is again irrelevant.

A simple yet effective approach is to circumvent the two aforementioned issues is to consider $T-1$ folds, such that we have one fold for each of the $T-1$ (x_t, y_t) pairs, where $t = 2, \ldots, T$. As we now leave out one sample, this approach is called "leave-one-out cross-validation" (LOOCV).

For each of the $T-1$ pairs, we will construct the training set as $D_{\mathrm{Tr}}^{(-t)} = D \setminus \{(x_t, y_t)\}$, and the validation set will simply be the left out pair $D_V^{(t)} = \{(x_t, y_t)\}$. We then compute the leave one out cross-validation score as

$$\mathrm{LOOCV}_t(W_k) = \left\| X_{t,\cdot} - X_{t-1,\cdot} W^{(-t)} \right\|_2^2, \qquad t = 2, \ldots, T. \tag{6.63}$$

Now, the average LOOCV score for W_k is the average across the $T-1$ folds,

$$\mathrm{LOOCV}(W_k) = \frac{1}{T-1} \sum_{t=2}^{T} \mathrm{LOOCV}_t(W_k). \tag{6.64}$$

We expect that each coefficient in W^* will contribute positively to the leave-one-out cross-validation score, as the arc provides an increase to the predictive performance. However, estimated coefficients which were equal to zero in W^* are estimated because of spurious correlations with the noise. Although these correlations may be apparent in D_{Tr}, these will hopefully not be apparent in D_V. Therefore, we expect the leave-one-out cross-validation score to decrease as long as we add correct arcs to W_k, and we expect to increase when we start adding incorrect arcs to W_k.

Therefore, there will be a specific value for k^* that minimizes this LOOCV score. This value for k^* will hopefully correspond to the number of non-zero coefficients in W^*, such that selecting W_{k^*} corresponds to the most suitable matrix.

Examples Let us investigate whether we can use this leave-one-out cross-validation approach to find a suitable number of non-zero coefficients in W. We would like to include arcs that provide a significant predictive performance, but we would like to exclude arcs that do provide a sufficient increase in the predictive performance. We hope that leave-one-out cross-validation will select a suitable number of arcs for us.

Let us now use such a forward approach to apply cross-validation.

Example 6.15 Cross-Validation for VAR(1) model selection.

Consider the data matrix $\mathbf{X} \in \mathbb{R}^{100 \times 10}$ which we have also used in Example 6.14, where we used coefficient matrix W^* consisting of 25 coefficients, of which 15 coefficients are off-diagonal.

We split our data matrix into 99 folds of the form $D_t = \{(x_t, y_t)\} = \{(X_{t-1,.}, X_{t,.})\}$ folds, one for each pair, where $t = 2, \ldots, 100$. We will use these folds to create 99 combinations of train and validation sets,

$$D_{\text{Tr}}^{(-t)} = D \setminus \{(x_t, y_t)\}, \qquad D_V^{(t)} = \{(x_t, y_t)\}, \qquad t = 2, \ldots, T.$$

For each training set $D_{\text{Tr}}^{(-t)}$, we use `DAG-OMP` to obtain a sequence of acyclic coefficient matrices $\left(W_k^{(-t)}\right)_{k=0}^{p(p+1)/2}$ and compute the average leave-one-out cross-validation score on $D_V^{(-t)}$ for each coefficient matrix W_k,

$$\text{LOOCV}(W_k) = \frac{1}{T-1} \sum_{t=1}^{T-1} \left\| X_{t+1} - X_t W_k^{(-t)} \right\|_2^2.$$

These leave-one-out cross-validation scores $\text{LOOCV}(W_k)$ are plotted as a function of k, the number of non-zero coefficients in the coefficient matrix W_k Figure 6.11.
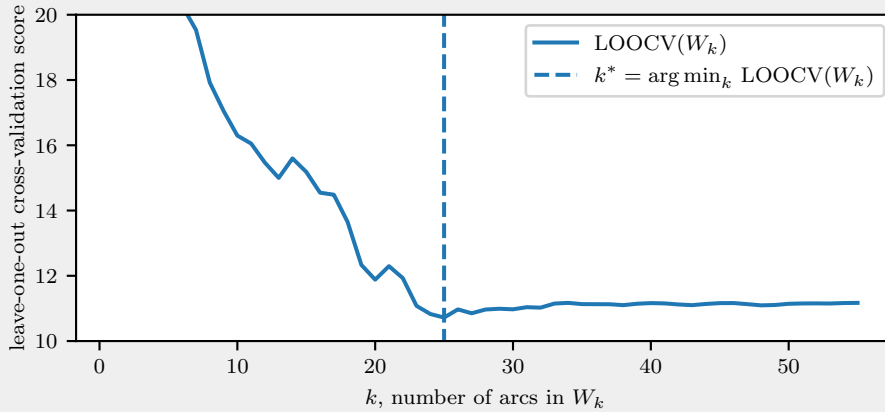

Figure 6.11: Plot of the leave-one-out cross-validation scores as a function of k, the number of edges in W_k. The smallest leave-one-out cross-validation score was attained for $k = 25$.

We see that the cost value first decreases quite steeply, after which the cost function is minimized for $k = 25$ edges. After $k = 25$, we first see that the cost slightly increases until around $k = 35$, after which the cost function remains approximately constant until the final value of $k = 55$. Therefore, we see that a total of $k = 25$ non-zero coefficients seems to be the correct number of arcs. Therefore, we will now apply `DAG-OMP` on the full dataset $\mathbf{X}$, and return the coefficient matrix after iteration $k = 25$. Using this approach, we recovered exactly all true coefficients of W^*, and no arc more, just as both bootstrapping approaches managed to do.

6.5   An Analysis of Cross-Validation for AR(1) models.

As we have seen, the leave-one-out cross-validation technique seems to be a suitable approach to regularize our coefficient matrix W. This seems counter-intuitive, as cross-validation is often not advised in time-series models due to the dependency between different time steps [11]. Therefore, we devote this section to analyze the performance of cross-validation in a much simpler setting.

6.5.1   AR(1) setting without mean.

Suppose that we are dealing with a one-dimensional data matrix $X \in \mathbb{R}^T$ which has been generated by an auto regressive (AR) model of order 1,

$$X_t = a_0 X_{t-1} + \varepsilon_t. \tag{6.65}$$

The parameter a_0 represents the auto regressive coefficient, and is assumed to be less than one in absolute value to ensure that the time series is stationary, $|a_0| < 1$. Furthermore, ε_t represents random noise, which we assume to be identically and independently standard normal distributed,

$$\varepsilon_t \sim \mathcal{N}(0, 1) \qquad \forall\, t = 2, \dots, T. \tag{6.66}$$

To make sure the data matrix X is in its stationary distribution from the very start, we sample X_1 according to its stationary distribution,

$$X_1 \sim \mathcal{N}\left(0, \frac{1}{1 - a_0^2}\right). \tag{6.67}$$

The setting is now as follows. Assume we know that X has been generated by an AR(1) model according to Equation 6.65 with auto regressive coefficient a_0. We would now like to verify whether the data has actually been generated by a_0. In other words, we are interested in the hypothesis test

$$H_0 : a = a_0 \text{ versus the alternative hypothesis } H_1 : a \neq a_0.$$

For this, we propose to use cross-validation, which is an unorthodox approach. We will compare the log-likelihood when using $a = a_0$ to the average log-likelihood of using $\{a^{(-t)}\}_{t=1}^{T-1}$ the leave-one-out cross-validation estimates for a. Let $-2LL_0(X)$ represent the negative log-likelihood of a_0 given the data matrix X, multiplied by two,

$$-2LL_0(X) = -2 \sum_{t=2}^{T} \log\left(-\frac{1}{\sqrt{2\pi}}\right) + \sum_{t=2}^{T} (X_{t, \cdot} - a_0 X_{t-1})^2, \tag{6.68}$$

and $-2LL_{\text{LOOCV}}(X)$ the average negative log-likelihood given X of the leave-one-out cross-validation estimates, multiplied by two,

$$-2LL_{\text{LOOCV}}(X) = -2 \sum_{t=2}^{T} \log\left(-\frac{1}{\sqrt{2\pi}}\right) + \sum_{t=2}^{T} \left(X_{t, \cdot} - \hat{a}^{(-t)} X_{t-1, \cdot}\right)^2. \tag{6.69}$$

Here $\hat{a}^{(-t)}$ represents the maximum likelihood estimate of a given the data matrix X, without using the (X_{t-1}, X_t) pair to estimate a,

$$\hat{a}^{(-t)} = \frac{\sum_{i=2}^{T} X_{i-1} X_i - X_{t-1} X_t}{\sum_{i=2}^{T} X_i^2 - X_{t-1}^2}. \tag{6.70}$$

We accept the null-hypothesis H_0 if $-2LL_0(X) \leq -2LL_{\text{LOOCV}}(X)$, and reject the null-hypothesis otherwise. Let us do some simulations to verify how often we correctly accept the null-hypothesis.

Simulation Study. For varying values of a_0 and T, we have simulated a total of N data matrices $X^{(n)} \in \mathbb{R}^T, n = 1 \dots, N$ according to this AR(1) model with auto regressive coefficient a_0. For each data matrix, we have computed $-2LL_0(X)$ and $-2LL_{\text{LOOCV}}(X)$, and counted how often we correctly accept H_0, i.e., $-2LL_{\text{LOOCV}}(X) \leq -LL_0(X)$. The ratio of correct acceptances of H_0 is then equal to

$$RCH_0(a, T) = \frac{1}{N} \sum_{n=1}^{N} \mathbf{1} \left\{ -2LL_0 \left(X^{(n)} \right) \leq -2LL_{\text{LOOCV}} \left(X^{(n)} \right) \right\}. \qquad (6.71)$$

The value of $RCH_0(a, T)$ has been computed for the two dimensional range of values for $a \in \{0.0, 0.3, 0.6, 0.9, 0.95, 0.99, 0.995\}$ and $T \in \{25, 50, 100, 250, 500, 1000\}$. For each (a_0, T) pair, we used $N = 1,000,000$ data matrices $X^{(n)}$ to reliably estimate $RCH_0(a_0, T)$. The values for $RCH_0(a_0, T)$ have been plotted as a function of a_0 in Figure 6.12 and as a function of T in Figure 6.13.
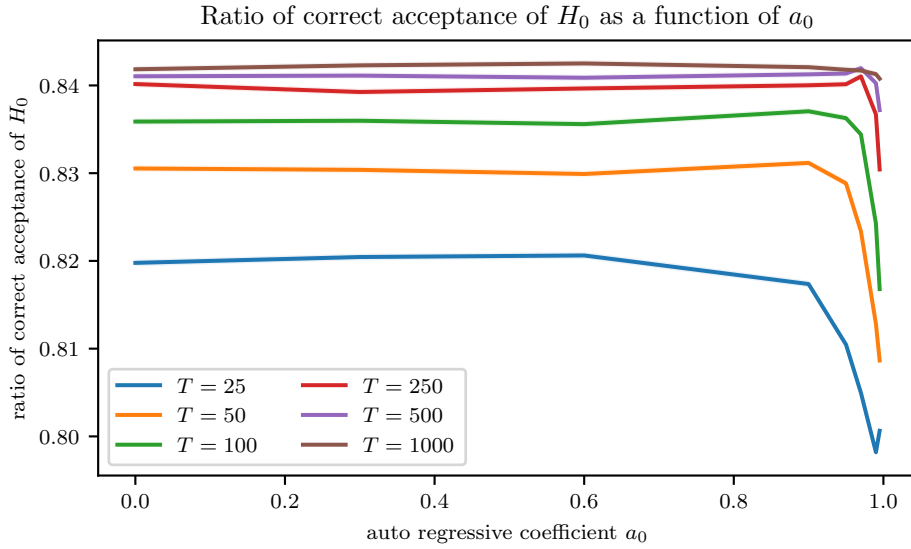

Figure 6.12: Plot of $RCH_0(a_0, T)$ as a function of the auto regressive coefficient a_0 for varying values of the number of time steps T. For each (a_0, T) pair, we have performed the simulation $N = 10^6$ times to get reliable estimates.

Interestingly, we see that quite often, we accept the true null-hypothesis using cross-validation, ranging from approximately 80 to 85% of the time. Therefore, cross-validation seems to be sensible for model selection, even when the data is dependent.

Let us first consider the ratio of correct H_0 acceptances as a function of the auto regressive coefficient a_0. For $a_0 = 0$, we have no dependence at all, and for a closer to one, the dependency increases. We see that for $a = 0.0$ until $a = 0.900$, the ratio of correct acceptance of H_0 remains constant for the different values of T. However, as a_0 tends closer and closer to one, the dependency increases, and we see that the ratio of correct acceptance of H_0 decreases slightly. Note, however, that this drop is quite modest, an absolute decrease of approximately 0.02. This decrease is especially visible for smaller values of T. For larger values of T, such as $T = 1000$, we see that the ratio barely decreases.

Secondly, let us investigate the ratio of correct H_0 acceptances as a function of T. The corresponding values for $RCH_0(a_0, T)$ a a function of T have been plotted in Figure 6.13. When we have more time steps, the ratio of correct acceptance of H_0 increases. This increase is especially visible for smaller values of a_0. From $a = 0.000$ until $a = 0.950$, we see that the increase happens quite soon, and already reaches its maximum value around $T = 200$, after which it remains constant. For values of a_0 closer to one, there is more dependency between the time steps in X, and therefore we also see that the ratio increases more slowly. Nevertheless, these ratios also seem to converge to the same value, but a larger number of time steps is required.
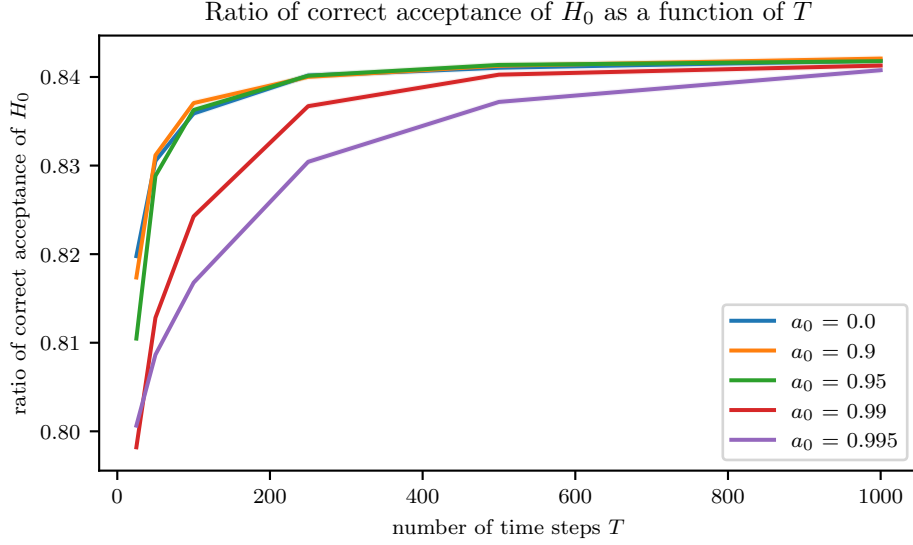
Ratio of correct acceptance of H_0 as a function of T

Legend:
- $a_0 = 0.0$
- $a_0 = 0.9$
- $a_0 = 0.95$
- $a_0 = 0.99$
- $a_0 = 0.995$

Figure 6.13: Plot of $RCH_0(a_0, T)$ as a function of the number of time steps T for varying values of the AutoRegressive coefficient a_0. For each (a_0, T) pair, we have conducted a total of $N = 10^6$ iterations to get reliable estimates.

Asymptotic value of $RCH_0(a_0, T)$. What is especially interesting is that these ratios do not converge to one, yet they seem to converge to some value around 0.84, regardless of the value for $|a_0| < 1$. For values of a_0 closer to one, convergence requires a larger number of time steps T, but also then the ratio converges to approximately 0.84.

Let us find an explanation for this constant. In mathematical terms, we conjecture that

$$\lim_{T \to \infty} \mathbb{E}\left[RCH_0(a, T)\right] = \text{const for } |a| < 1, \qquad \text{where const} \approx 0.84.$$

As it turns out, the difference between the negative log-likelihoods follows a chi-squared distribution with one degree of freedom, shifted two units to the left. That is,

$$\lim_{T \to \infty} -2LL_0(X) + 2LL_{\text{LOOCV}}(X) \sim \chi_1^2 - 2.$$

This would indeed make sense, as then,

$$
\begin{aligned}
\lim_{T \to \infty} \mathbb{E}\left[RCH_0(a, T)\right] &= \lim_{T \to \infty} \frac{1}{N} \sum_{n=1}^{N} \mathbb{E}\left[\mathbf{1}\left\{-2LL_0\left(X^{(n)}\right) \leq -2LL_{\text{LOOCV}}\left(X^{(n)}\right)\right\}\right] \\
&= \lim_{T \to \infty} \frac{1}{N} \sum_{n=1}^{N} \mathbb{P}\left(-2LL_0\left(X^{(n)}\right) \leq -2LL_{\text{LOOCV}}\left(X^{(n)}\right)\right) \\
&= \lim_{T \to \infty} \mathbb{P}\left(-2LL_0\left(X\right) \leq -2LL_{\text{LOOCV}}\left(X\right)\right) \\
&= \lim_{T \to \infty} \mathbb{P}\left(-2LL_0(X) + 2LL_{\text{LOOCV}}(X) \leq 0\right) \\
&= \mathbb{P}\left(Q \leq 2\right),
\end{aligned}
$$

where $Q \sim \chi_1^2$. Now, using the cumulative distribution function of the chi-squared distribution with one degree of freedom, we see that

$$\mathbb{P}(Q \leq 2) = \frac{\gamma(1/2, 1)}{\Gamma(1/2)} = \frac{1}{\sqrt{\pi}} \sum_{k=0}^{\infty} \frac{e^{-1}}{1/2 \cdot (1/2 + 1) \cdots \ldots \cdots (1/2 + k)} \approx 0.8427,$$

which indeed closely resembles the limit we see in Figure 6.13.

To further verify our claims, let us consider the histogram of the random variable $-2LL_0(X) + 2LL_{\text{LOOCV}}(X)$ for $a = 0.5$ and $T = 10{,}000$. For this, we generated $100{,}000$ data matrices $X \in \mathbb{R}^T$, and computed $-2LL_0(X) + 2LL_{\text{LOOCV}}(X)$. These $100{,}000$ samples were then plotted in a histogram, along with the probability density function of $Q \sim \chi_1^2 - 2$ in Figure 6.14.
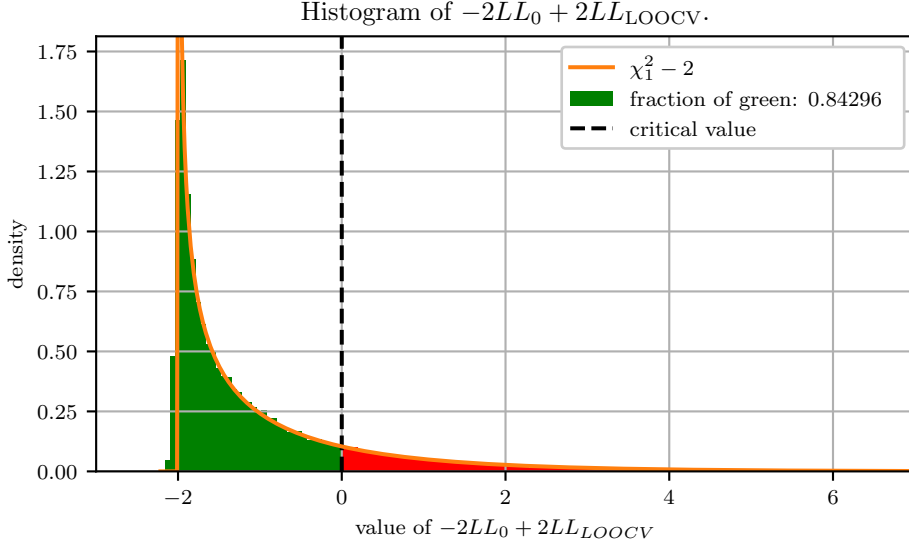

Histogram of $-2LL_0 + 2LL_{\text{LOOCV}}$.

Figure 6.14: Histogram of $-2LL_0(X) + 2LL_{\text{LOOCV}}(X)$, computed from $50{,}000$ data matrices $X \in \mathbb{R}^T$ with $T = 10{,}000$ using an AR(1) model with $a = 0.5$. The probability density function of a chi-squared distribution with one degree of freedom, translated two units to the left has also been plotted.

The random variable $-2LL_0(X) + 2LL_{\text{LOOCV}}(X)$ indeed almost perfectly follows a chi squared distribution with one degree of freedom that has been translated two units to the left. Here, a value smaller than zero indicates that $-2LL_0(X) \leq -2LL_{\text{LOOCV}}(X)$, indicating that we correctly fail to reject H_0. The bars where $-2LL_0(X) \leq -2LL_{\text{LOOCV}}(X)$ have been plotted in green and the ratio of green bars indeed corresponds to a value of approximately 0.8430, close to the value $\mathbb{P}(Q \leq 2) \approx 0.8427$. These simulations indeed support our claim.

A rigorous derivation that shows that

$$\lim_{T \to \infty} -2LL_0(X) + 2LL_{\text{LOOCV}}(X) \sim \chi_1^2 - 2 \tag{6.72}$$

has been given in Appendix A.

Modifying the type-I error Equation 6.72 unveils an interesting relation between the leave-one-out cross-validation score and hypothesis testing. We can use the leave-one-out cross-validation score to determine whether a time series X has indeed been generated using the auto regressive coefficient a_0. Without changing the leave-one-out cross-validation score, we know that the ratio of correctly accepting H_0 is equal to $P(Q \leq 2) \approx 0.8427$, where $Q \sim \chi_1^2$. However, we can modify this to adjust the confidence of correctly accepting the null-hypothesis, otherwise known as the type-1 error α. For example, if we prefer a type-II error of $\alpha = 0.05$, that means we are looking for a k such that $\mathbb{P}(Q \leq k) = 0.95 \Rightarrow k \approx 3.8414$ by looking up the quantiles of the chi-squared distribution. We could achieve this by changing our decision rule to

$$RCH_0(a, T) = \frac{1}{N} \sum_{n=1}^{N} \mathbf{1} \left\{ -2LL_0\left(X^{(n)}\right) \leq -2LL_{\text{LOOCV}}\left(X^{(n)}\right) + 1.8414 \right\}. \tag{6.73}$$

6.5.2 AR(1) Setting with mean.

We have also investigated the setting in which the auto regressive model consisted of two parameters, one auto regressive coefficient a and a mean of b. Now, the model is defined as

$$X_t = b_0 + a_0(X_{t-1} - b_0) + \varepsilon_t. \tag{6.74}$$

Again, ε_t represents random noise, which we assume to be identically and independently standard normal distributed,

$$\varepsilon_t \sim \mathcal{N}(0,1) \ \forall \ t = 2, \ldots, T.$$

To make sure the data matrix X is in its stationary distribution from the very start, we sample X_1 according to its stationary distribution,

$$X_1 \sim \mathcal{N}\left(b_0, \frac{1}{1-a_0^2}\right).$$

The setting is now as follows. Assume we know that X has been generated by an AR(1) model according to Equation 6.74 with auto regressive coefficient a_0 and mean b_0. We would now like to verify whether the data has actually been generated by a_0. In other words, we are interested in the hypothesis test

$$H_0 : a = a_0 \text{ versus the alternative hypothesis } H_1 : a \neq a_0.$$

For this, we will again use cross-validation, hoping it will be effective just an for the AR(1) setting without any mean as in Subsection 6.5.1. The difference between the previous setting is that we now need to estimate b as well.

Deriving $-2LL_0(X)$. We will compare the negative log-likelihood when using $a = a_0$ multiplied by two against the average negative log-likelihood of using $\{a^{(-t)}\}_{t=1}^{T-1}$, the leave-one-out cross-validation estimates for a. However, we now also need to estimate b.

Let $-2LL_0(X)$ represent the negative log-likelihood, multiplied by two, of a_0 given the data matrix X, which is now defined as

$$-2LL_0(X) = -2\sum_{t=2}^{T}\log\left(-\frac{1}{\sqrt{2\pi}}\right) + \sum_{t=2}^{T}\left((X_1 - \hat{b}^{(-t)}) - a_0\left(X_{t-1} - \hat{b}_0^{(-t)}\right)\right)^2, \tag{6.75}$$

where $\hat{b}_0^{(-t)}$ corresponds to the leave-one-out maximum likelihood estimator given a_0 and X, where we have left out the pair (X_{t-1}, X_t),

$$\hat{b}_0^{(-t)} = \frac{\left(\sum_{k=2}^{T} X_k - X_t\right) - a_0\left(\sum_{k=1}^{T-1} X_k - X_t\right)}{(T-2)(1-a_0)}. \tag{6.76}$$

Deriving $-2LL_{\text{LOOCV}}(X)$. Deriving the negative log-likelihood using leave-one-out cross-validation to derive both $\hat{a}^{(-t)}$ and $\hat{b}^{(-t)}$ is slightly more involved. We cannot use Equation 6.76, as we do not have an estimate for $\hat{a}^{(-t)}$, nor do we have an estimate for $\hat{b}^{(-t)}$.

If we had known $\hat{a}^{(-t)}$, then the estimate for $\hat{b}^{(-t)}$ would be

$$\hat{b}^{(-t)} = \frac{\left(\sum_{k=2}^{T} X_k - X_t\right) - \hat{a}^{(-t)}\left(\sum_{k=1}^{T-1} X_k - X_t\right)}{(T-2)(1-\hat{a}^{(-t)})}. \tag{6.77}$$

Analogously, if we had known $\hat{b}^{(-t)}$, then the estimate for $\hat{a}^{(-t)}$ would be

$$\hat{a}^{(-t)} = \frac{\sum_{i=2}^{T}\left(X_{i-1} - \hat{b}^{(-t)}\right)\left(X_i - \hat{b}^{(-t)}\right) - \left(X_{t-1} - \hat{b}^{(-t)}\right)\left(X_t - \hat{b}^{(-t)}\right)}{\sum_{i=2}^{T}\left(X_i - \hat{b}^{(-t)}\right)^2 - \left(X_{t-1} - \hat{b}^{(-t)}\right)^2}. \tag{6.78}$$

Now, we could do an iterative approach, where we will initially estimate $\hat{b}$ without $\hat{a}^{(-t)}$ as $\hat{b}^{(-t)} \approx (T-2)^{-1} \sum_{t=2}^{T} -X_t$. Then, we iteratively use Equation 6.77 and Equation 6.78 to update our leave-one-out coefficients $\hat{a}^{(-t)}$ and $\hat{b}^{(-t)}$ until the negative log-likelihood has converged. Then, the corresponding negative log-likelihood, multiplied by two, corresponds to

$$LL_{\text{LOOCV}}(X) = -2 \sum_{t=2}^{T} \log\left(-\frac{1}{\sqrt{2\pi}}\right) + \sum_{t=2}^{T} \left(\left(X_t - \hat{b}^{(-t)}\right) - \hat{a}^{(-t)}\left(X_{t-1} - \hat{b}^{(-t)}\right)\right)^2. \quad (6.79)$$

We again say that we accept the null-hypothesis if the negative log-likelihood under H_0 is smaller than when we would use the leave-one-out cross-validation score, that is, when $-2LL_0(X) \leq -2LL_{\text{LOOCV}}$.

Simulation Study. Let us first investigate how b_0 affects the ratio of correct acceptance. For varying values of b_0 and a fixed value for both a_0 and T, we have generated a total of N data matrices $X^{(n)} \in \mathbb{R}^T, n = 1 \ldots, N$ of length T according to this AR(1) model with auto regressive coefficient a_0 and mean b_0.

For each data matrix, we have computed $-2LL_0(X)$ and $-2LL_{\text{LOOCV}}(X)$ according to Equation 6.75 and Equation 6.79, respectively. Subsequently, we have counted how often we correctly accept H_0, i.e., $-2LL_0(X) \leq -2LL_{\text{LOOCV}}(X)$. The ratio of correct acceptances of H_0 is then again equal to

$$RCH_0(a_0, b_0, T) = \frac{1}{N} \sum_{n=1}^{N} \mathbf{1}\left\{-2LL_0\left(X^{(n)}\right) \leq -2LL_{\text{LOOCV}}\left(X^{(n)}\right)\right\}. \quad (6.80)$$

The value of $RCH_0(a_0, b_0, T)$ has first been been computed for $a = 0.5$ and $T = 100$, and for values of b_0 in the range of $\{-50, -25, 0, 25, 50\}$. For each value of b_0, we have used $N = 10,000$ data matrices to reliably estimate $RCH_0(a_0, b_0, T)$. The values of $RCH_0(a_0, b_0, T)$ as a function of b_0 have been plotted in Figure 6.15.
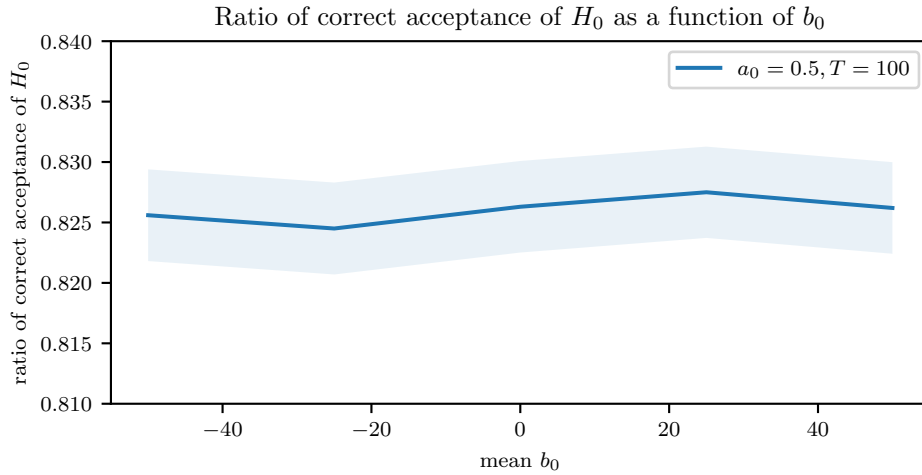

Figure 6.15: Plot of $RCH_0(a_0, b_0, T)$ as a function for varying values of the mean b_0. The auto regressive coefficient has been fixed to $a_0 = 0.5$, and the number of time steps has been fixed to $T = 100$. For each value of b_0, we have performed the simulation $N = 10^4$ times to get reliable estimates. Shaded parts correspond to the standard errors.

We see that the value for b_0 does not influence the value for $RCH_0(a_0, b_0, T)$. We know that both $\hat{b}_0^{(-t)}$ and $\hat{b}^{(-t)}$ are both estimated using cross-validation using Equation 6.76 and Equation 6.77, respectively, albeit with a different value for a. Therefore, as the computation procedures are similar, it is not unexpected that the value for b_0 does not affect $RCH_0(a_0, b_0, T)$.

Now, let us see how the values for a and T influence the value of $RCH_0(a_0, b_0, T)$. Based on the analysis in Subsection 6.5.1, we expect the value of $RCH_0(a_0, b_0, T)$ to decrease when a_0 is close to one or when T is small. We have again generated $N = 10,000$ data matrices for each (a_0, T) pair in the two dimensional range of values for $a_0 \in \{0.0, 0.25, 0.6, 0.9, 0.95, 0.99, 0.995\}$ and $T \in \{25, 50, 100, 250, 500, 1000\}$ to reliably estimate $RCH_0(a_0, b_0, T)$. The values for $RCH_0(a_0, b_0, T)$ have been plotted as a function of a_0 in Figure 6.16 and as a function of T in Figure 6.17. As the value for b_0 did not have any effect on $RCH_0(a_0, b_0, T)$, we have fixed $b_0 = 0$. Note, however, that b_0 will still be estimated for both negative log-likelihoods using leave-one-out cross-validation.
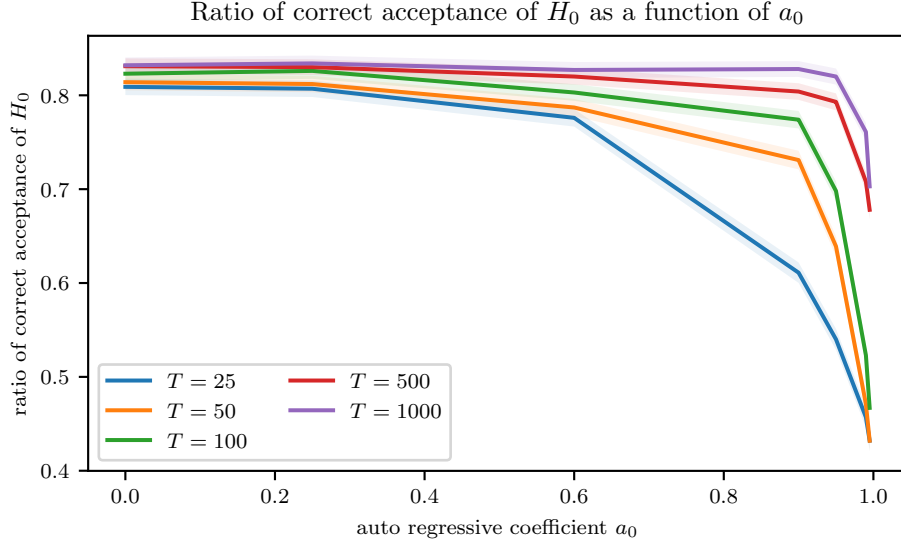

Ratio of correct acceptance of H_0 as a function of a_0

Figure 6.16: Plot of $RCH_0(a_0, b_0, T)$ as a function of the auto regressive coefficient a_0 for varying values of the number of time steps T. For each (a_0, T) pair, we have performed the simulation $N = 10^4$ times to get reliable estimates. Shaded parts correspond to the standard errors.

Just as in Subsection 6.5.1, we see that the ratio of correct acceptance of H_0 is quite high when either a is close to zero or when T is large. This ratio then is slightly larger than 0.80.

However, we see that the ratio of correct acceptance of H_0 decreases quite quickly when a gets closer to zero, especially when T is small. For example, even when $a_0 = 0.9$ and $T = 25$, we see that the ratio of correct acceptance of H_0 has already decreases to approximately 0.6, much smaller than in the previous subsection. When the auto regressive coefficient tends closer to one, we see that the ratios of correct decrease even further, being less than 0.5 for $T \in \{25, 50, 100\}$. Now that both models need to estimate b_0, the ratio of correct acceptance is much smaller when either a is close to one.

Again, we see that when we have more samples, which is the case for $T = 500$ and $T = 1000$, the ratio of correct acceptance of H_0 remains constant around 0.84 for quite some time, approximately until $a_0 = 0.9$. However, when a_0 gets even closer to one, the ratio of correct acceptance of H_0 drops to approximately 0.7.

Secondly, let us investigate the ratio of correct H_0 acceptances as a function of T. The corresponding values for $RCH_0(a_0, T)$ a a function of T have been plotted in Figure 6.17. When we have more time steps, the ratio of correct acceptance of H_0 increases, just as in the scenario where we had no mean. Especially for values of a_0 close to one, we see that the ratio of correct acceptance of H_0 increases when we increase the number of samples. It seems that when we have more data, this method is more capable of correctly accepting H_0.

Interestingly, we again see that this ratio does not converge to one, but again some value around 0.84. This is most likely due to the same behavior we have seen in Subsection 6.5.1, where this difference between the negative log-likelihoods, multiplied by two, converges to a chi-squared distribution, shifted two units to the right.
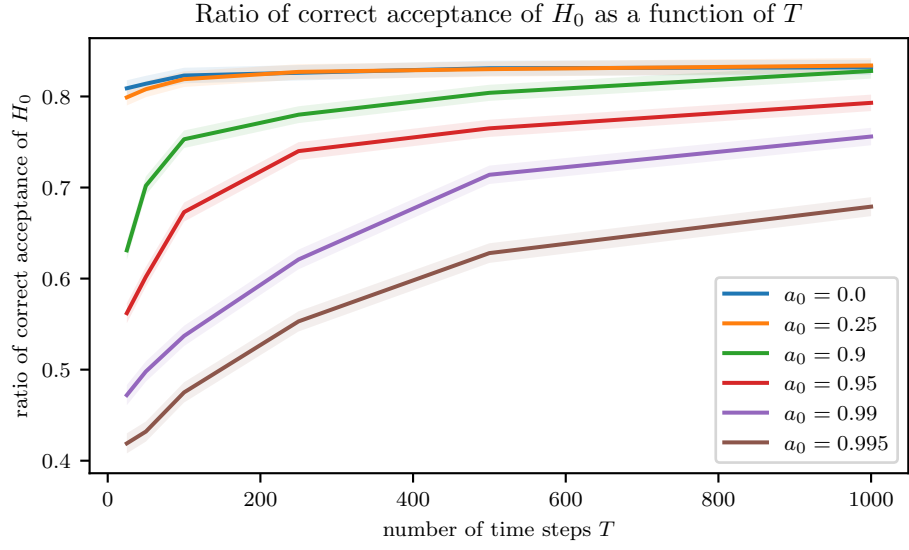
Figure 6.17: Plot of $RCH_0(a_0, b_0, T)$ as a function of the number of time steps T for varying values of the auto regressive coefficient a_0. For each (a_0, T) pair, we have conducted a total of $N = 10^4$ iterations to get reliable estimates. Shaded parts correspond to the standard errors.

Bibliography

[1] Scipy api reference for `optimize.minimize`. 22

[2] Scipy api reference for the `L-BFGS-B` optimization method. 22

[3] *Vector Autoregressive Models for Multivariate Time Series*, pages 385–429. Springer New York, New York, NY, 2006. 54

[4] Awad H. Al-Mohy and Nicholas J. Higham. A new scaling and squaring algorithm for the matrix exponential. *SIAM Journal on Matrix Analysis and Applications*, 31(3):970–989, 2010. 74, 93

[5] M. Andrle, L. Rebollo-Neira, and E. Sagianos. Backward-optimized orthogonal matching pursuit approach. *IEEE Signal Processing Letters*, 11(9):705–708, 2004. 91

[6] 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

[7] Christoph Bergmeir, Rob J. Hyndman, and Bonsoo Koo. A note on the validity of cross-validation for evaluating autoregressive time series prediction. *Computational Statistics Data Analysis*, 120:70–83, 2018. 104

[8] Garrett Birkhoff. Three observations on linear algebra. *Univ. Nac. Tacuman, Rev. Ser. A*, 5:147–151, 1946. 51

[9] Thomas Blumensath and Mike Davies. On the difference between orthogonal matching pursuit and orthogonal least squares. 03 2007. 76

[10] 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. 32

[11] Prabir Burman, Edmond Chow, and Deborah Nolan. A cross-validatory method for dependent data. *Biometrika*, 81:351–358, 1994. 107

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

[13] Rui Castro and Robert Nowak. Likelihood based hierarchical clustering and network topology identification. In Anand Rangarajan, Mário Figueiredo, and Josiane Zerubia, editors, *Energy Minimization Methods in Computer Vision and Pattern Recognition*, pages 113–129, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg. 39

[14] S. CHEN, S. A. BILLINGS, and W. LUO. Orthogonal least squares methods and their application to non-linear system identification. *International Journal of Control*, 50(5):1873–1896, 1989. 76

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

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

[17] 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

[18] Aramayis Dallakyan and Mohsen Pourahmadi. Learning bayesian networks through birkhoff polytope: A relaxation method. *CoRR*, abs/2107.01658, 2021. 63

[19] Ivan Damnjanovic, Matthew E. P. Davies, and Mark D. Plumbley. Smallbox - an evaluation framework for sparse representations and dictionary learning algorithms. In Vincent Vigneron, Vicente Zarzoso, Eric Moreau, Rémi Gribonval, and Emmanuel Vincent, editors, *Latent Variable Analysis and Signal Separation*, pages 418–425, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg. 86

[20] B. Efron. Bootstrap Methods: Another Look at the Jackknife. *The Annals of Statistics*, 7(1):1 – 26, 1979. 97

[21] Kim Esbensen and Paul Geladi. Principles of proper validation: use and abuse of re-sampling for validation. *Journal of Chemometrics*, 24:168 – 187, 03 2010. 104

[22] Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman amp; Co., USA, 1990. 76

[23] M. Gharavi-Alkhansari and T.S. Huang. A fast orthogonal matching pursuit algorithm. In *Proceedings of the 1998 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP '98 (Cat. No.98CH36181)*, volume 3, pages 1389–1392 vol.3, 1998. 76

[24] Hemant S. Goklani, Jignesh N. Sarvaiya, and A. M. Fahad. Image reconstruction using orthogonal matching pursuit (omp) algorithm. In *2014 2nd International Conference on Emerging Technology Trends in Electronics, Communication and Networking*, pages 1–5, 2014. 77

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

[26] 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. 72

[27] W. K. Hastings. Monte carlo sampling methods using markov chains and their applications. *Biometrika*, 57(1):97–109, 1970. 39

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

[29] Guoxian Huang and Lei Wang. High-speed signal reconstruction with orthogonal matching pursuit via matrix inversion bypass. pages 191–196, 10 2012. 86

[30] Robin John Hyndman and George Athanasopoulos. *Forecasting: Principles and Practice*. OTexts, Australia, 2nd edition, 2018. 26

[31] Donald B. Johnson. Finding all the elementary circuits of a directed graph. *SIAM Journal on Computing*, 4(1):77–84, 1975. 92

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

[33] A. B. Kahn. Topological sorting of large networks. *Commun. ACM*, 5(11):558–562, nov 1962. 75

[34] Mahdi Khosravy, Nilanjan Dey, and Carlos Duque. *Compressive Sensing in Health Care*. 10 2019. 76

[35] S. N. Lahiri. *Bootstrap Methods*, pages 17–43. Springer New York, New York, NY, 2003. 97

[36] 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

[37] Hanxi Li, Yongsheng Gao, and Jun Sun. Fast kernel sparse representation. In *2011 International Conference on Digital Image Computing: Techniques and Applications*, pages 72–77, 2011. 86

[38] S.G. Mallat and Zhifeng Zhang. Matching pursuits with time-frequency dictionaries. *IEEE Transactions on Signal Processing*, 41(12):3397–3415, 1993. 76

[39] Nicholas Metropolis, Arianna W. Rosenbluth, Marshall N. Rosenbluth, Augusta H. Teller, and Edward Teller. Equation of State Calculations by Fast Computing Machines. , 21(6):1087–1092, June 1953. 39

[40] 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. 23

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

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

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

[44] K. B. Petersen and M. S. Pedersen. The matrix cookbook, October 2008. Version 20081110. 57

[45] 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. 31

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

[47] 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, 87

[48] Konstantinos Skianis, Nikolaos Tziortziotis, and Michalis Vazirgiannis. Orthogonal matching pursuit for text classification. *ArXiv*, abs/1807.04715, 2018. 77

[49] M. Stone. Cross-validatory choice and assessment of statistical predictions. *Journal of the Royal Statistical Society. Series B (Methodological)*, 36(2):111–147, 1974. 104

[50] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1):267–288, 1996. 67

[51] Ryan Tibshirani and Jonathan Taylor. The solution path of the generalized lasso. *The Annals of Statistics*, 39, 05 2010. 69

[52] Joel A. Tropp and Anna C. Gilbert. Signal recovery from random measurements via orthogonal matching pursuit. *IEEE Transactions on Information Theory*, 53(12):4655–4666, 2007. 86

[53] Ioannis Tsamardinos, Laura Brown, and Constantin Aliferis. The max-min hill-climbing bayesian network structure learning algorithm. *Machine Learning*, 65:31–78, 10 2006. 116

[54] 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

[55] 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. 22

[56] 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. 51

[57] 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

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

[59] Tong Zhang. On the consistency of feature selection using greedy least squares regression. *Journal of Machine Learning Research*, 10(19):555–568, 2009. 76

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

[61] 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. 50, 63

[62] 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. 22

[63] Hufei Zhu, Wen Chen, and Yanpeng Wu. Efficient implementations for orthogonal matching pursuit. *Electronics*, 9:1507, 09 2020. 86

Appendix A

Difference of the negative log-likelihoods

In this chapter of the appendix, we will mathematically verify our empirical findings in Subsection 6.5.1. It seemed that the random variable

$$-2LL_0(X) + 2LL_{\text{LOOCV}}(X) = \sum_{t=2}^{T} (X_t - a_0 X_{t-1})^2 - \sum_{t=2}^{T} \left(X_{t,} - \hat{a}^{(-t)} X_{t-1} \right)^2 \tag{A.1}$$

converged to a chi-squared distribution, shifted two units to the left, when T tends to infinity. That is,

$$\lim_{T \to \infty} -2LL_0(X) + 2LL_{\text{LOOCV}}(X) \sim \chi_1^2 - 2. \tag{A.2}$$

Decomposing Equation A.1 To prove the conjecture stated in Equation A.2, we will first rewrite the equation and subsequently invoke Wilk's theorem. Let $-2LL_1(X)$ represent twice the negative log-likelihood of corresponding the maximum likelihood estimator $\hat{a}$,

$$-2LL_1(X) = -2 \sum_{t=2}^{T} \log \left(-\frac{1}{\sqrt{2\pi}} \right) + \sum_{t=2}^{T} (X_t - \hat{a} X_{t-1})^2,$$

where the maximum likelihood estimator of a_0 can be computed as

$$\hat{a} = \frac{\sum_{t=2}^{T} X_{t-1} X_t}{\sum_{t=2}^{T} X_{t-1}^2}. \tag{A.3}$$

We can rewrite $-2LL_0 + 2LL_{\text{LOOCV}}$ to

$$
\begin{aligned}
-2LL_0 + LL_{\text{LOOCV}} &= \sum_{t=2}^{T} \left(X_t - a_0 X_{t-1}\right)^2 - \sum_{t=2}^{T} \left(X_t - \hat{a}^{(-t)} X_{t-1}\right)^2 \\
&= \sum_{t=2}^{T} \left(X_t - a_0 X_{t-1}\right)^2 - \sum_{t=2}^{T} \left(X_t - \left(\hat{a} - \hat{a} + \hat{a}^{(-t)}\right) X_{t-1}\right)^2 \\
&= \sum_{t=2}^{T} \left(X_t - a_0 X_{t-1}\right)^2 - \sum_{t=2}^{T} \left(X_t - \hat{a} X_{t-1} - \left(\hat{a}^{(-t)} - \hat{a}\right) X_{t-1}\right)^2 \\
&= \sum_{t=2}^{T} \left(X_t - a_0 X_{t-1}\right)^2 - \sum_{t=2}^{T} \Big(\left(X_t - \hat{a} X_{t-1}\right)^2 \\
&\qquad\qquad - 2\left(X_t - \hat{a} X_{t-1}\right)\left(\left(\hat{a}^{(-t)} - \hat{a}\right) X_{t-1}\right) \\
&\qquad\qquad + \left(\left(\hat{a}^{(-t)} - \hat{a}\right) X_{t-1}\right)^2 \Big) \\
&= -2LL_0 + 2LL_1 + C_T,
\end{aligned}
$$

where

$$
C_T = \sum_{t=2}^{T} \left(-2\left(X_t - \hat{a} X_{t-1}\right)\left(\left(\hat{a}^{(-t)} - \hat{a}\right) X_{t-1}\right) + \left(\left(\hat{a}^{(-t)} - \hat{a}\right) X_{t-1}\right)^2 \right). \tag{A.4}
$$

Let us first consider the negative log-likelihood ratio $-2LL_0(X) + 2LL_1(X)$. Wilk's theorem states that under the null-hypothesis,

$$
-2LL_0(X) + 2LL_1(X) \sim \chi_1^2. \tag{A.5}
$$

From this decomposition, we can see the from where this chi-squared distribution originates. What remains now is to compute the limit of C_T.

Decomposing C_T. Now, let us further dissect this quantity C_T. We are only interested in terms that do not vanish as T gets arbitrarily large. For C_T, that means that we can disregard all components of C_T that are of order $o(1/T)$.

Intuitively, we expect $\hat{a}^{(-t)}$ and $\hat{a}$ to be close to each other in value. In fact, their difference will be of the order of $o(1/T)$. Therefore, we see that the $\left(a^{(-t)} - \hat{a}\right)^2$ to be of the order $o(1/T^2)$, and therefore their sum over T will be of the order $o(1/T)$, which converges to zero as T tends to infinity. Therefore, we can disregard the last component of the sum.

Secondly, the first component of Equation A.4 can be decomposed into

$$
\begin{aligned}
-2\sum_{t=2}^{T} \left(\left(X_t - \hat{a} X_{t-1}\right)\left(\left(\hat{a}^{(-t)} - \hat{a}\right) X_{t-1}\right) \right) &= -2\sum_{t=2}^{T} \left(\left(a X_{t-1} + \varepsilon_t - \hat{a} X_{t-1}\right)\left(\left(\hat{a}^{(-t)} - \hat{a}\right) X_{t-1}\right) \right) \\
&= -2\sum_{t=2}^{T} \left(\left(\hat{a}^{(-t)} - \hat{a}\right)\left((a - \hat{a}) X_{t-1}^2 + X_{t-1}\varepsilon_t\right) \right).
\end{aligned}
\tag{A.6}
$$

We see that the first part of the summation is a product of the difference between a and $\hat{a}$ and the difference between $\hat{a}^{(-t)}$ and $\hat{a}$, both of which are approximately of the order $o(1/T)$, meaning that the product is of the order $o(1/T^2)$. Therefore, summing over these products yields a value of the order $o(1/T)$, which also converges to zero as T tends to infinity. Therefore, we can also disregard the first component of Equation A.6. What remains is the final component,

$$
-2\sum_{t=2}^{T} \left(\hat{a}^{(-t)} - \hat{a}\right) X_{t-1}\varepsilon_t. \tag{A.7}
$$

Now, the difference between $\hat{a}^{(-t)}$ and $\hat{a}$ is of the order $o(1/T)$, so summing difference over T should give some constant.

To slim Equation A.7 down even further, let us first remark that we can rewrite $\hat{a}^{(-t)} - \hat{a}$ as

$$\hat{a}^{(-t)} - \hat{a} = -\frac{X_{t-1}}{\sum_{k=2}^{T} X_{k-1}^2 - X_{t-1}^2} \left(X_t - \hat{a} X_{t-1} \right)$$
$$= -\frac{X_{t-1}}{\sum_{k=2}^{T} X_{k-1}^2 - X_{t-1}^2} \left((a - \hat{a}) X_{t-1} + \varepsilon_t \right).$$

Again, as $(a - \hat{a})$ is of the order of $o(1/T)$, and $1/\left(\sum_{k=2}^{T} X_{k-1}^2 - X_{t-1}^2 \right)$ is also of $o(1/T)$, the product of both components will be of the order $o(1/T^2)$. Therefore,

$$\hat{a}^{(-t)} - \hat{a} = -\frac{X_{t-1}}{\sum_{k=2}^{T} X_{k-1}^2 - X_{t-1}^2} \left((a - \hat{a}) X_{t-1} + \varepsilon_t \right)$$
$$= -\frac{X_{t-1}\varepsilon_t}{\sum_{k=2}^{T} X_{k-1}^2 - X_{t-1}^2} + o(1/T^2). \tag{A.8}$$

Returning back to Equation A.7, using Equation A.8, we get that

$$-2\sum_{t=2}^{T} \left(\hat{a}^{(-t)} - \hat{a} \right) X_{t-1}\varepsilon_t = 2\sum_{t=2}^{T} \left(\frac{(X_{t-1}\varepsilon_t)^2}{X_{t-1}^2 - \sum_{k=2}^{T} X_{k-1}^2} \right) + o(1/T).$$

Let us also quickly remark that the first order component of each entry in the sum is negative, as the nominator is always positive, whereas the denominator is always negative.

To quickly summarize, we have done a careful derivation of C_T, which was given into Equation A.4. We have splitted C_T into two components, the part which is of order $o(1/T^2)$, and its leading terms, yielding the decomposition

$$C_T = 2\sum_{t=1}^{T-1} \left(\frac{(X_t\varepsilon_{t+1})^2}{X_t^2 - \sum_{k=1}^{T-1} X_k^2} \right) \tag{A.9}$$
$$\approx C_{T,\text{approx}}, \tag{A.10}$$

where

$$C_{T,\text{approx}} = 2\sum_{t=1}^{T-1} \left(\frac{(X_t\varepsilon_{t+1})^2}{X_t^2 - \sum_{k=1}^{T-1} X_k^2} \right) \tag{A.11}$$

Comparing C_T to $C_{T,\text{approx}}$. Let us first inspect whether the derivation from C_T to $C_{T,approx}$ was sensible. We generated data from an AR(1) model with auto regressive coefficient $a = 0.9$ and a total of $T = 10,000$ time steps. Now, the exact value for C_T from Equation A.4 was -2.0179, whereas the approximate value $C_{T,\text{approx}}$, where we have removed all components of order $o(1/T)$ after summation, was equal to -2.0182. We see that the derivation was still accurate up to three decimals, so it seems that the derivation was indeed sensible.

To see whether the distribution remains the same for C_T and $C_{T,\text{approx}}$, let us consider histograms for C_T and $C_{T,\text{approx}}$ for a fixed value of the auto regressive coefficient a and the number of time steps T. We have plotted the two histograms consisting of $N = 100,000$ samples of C_T and $C_{T,\text{approx}}$ for $a = 0.5$ and $T = 10,000$ in Figure A.1.
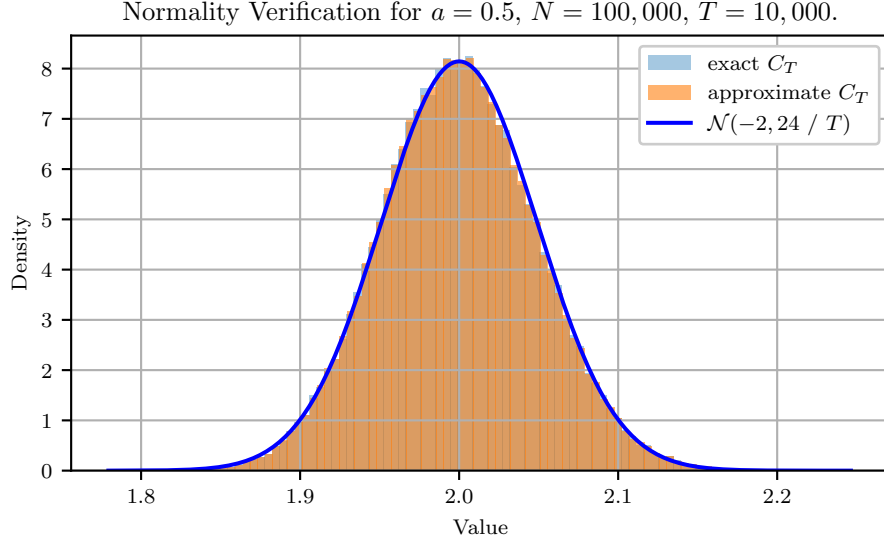
Normality Verification for $a = 0.5$, $N = 100,000$, $T = 10,000$.

Figure A.1: Histogram visualizing the distribution of C_T and $C_{T,\text{approx}}$ as defined in Equation A.4 and Equation A.10, respectively. We see that the two histograms overlap almost perfectly, indicating that the derivation was sensible. Furthermore, we have fitted a normal distribution with a mean of -2 and a variance of $-24/T$, which seems to be a perfect fit for both C_T and the approxiamte C_T.

We see that both C_T and $C_{T,\text{approx}}$ follow a normal distribution with a mean of -2 and a variance of $24/T$. Therefore, as T tends to infinity, the variance of C_T will tend to zero. In conclusion, we have that the limit of C_T is equal to

$$\lim_{T \to \infty} C_T = -2. \tag{A.12}$$

Combining this statement with the statement from Equation A.5, where we used Wilk's theorem, we have that

$$\lim_{T \to \infty} -2LL_0 + 2LL_{\text{LOOCV}} = \lim_{T \to \infty} -2LL_0 + 2LL_1 + C_T \sim \chi_1^2 - 2. \tag{A.13}$$