2019-04-01

Representation and Reconstruction of Linear, Time-Invariant Networks

Nathan Scott Woodbury
Brigham Young University

Representation and Reconstruction of Linear, Time-Invariant Networks

Nathan Scott Woodbury

A dissertation submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

Sean C. Warnick, Chair
Mark Kenneth Transtrum
David Wingate
Bryan Stuart Morse
Tony R. Martinez

Department of Computer Science

Brigham Young University

ABSTRACT

Representation and Reconstruction of Linear, Time-Invariant Networks

Nathan Scott Woodbury
Department of Computer Science, BYU
Doctor of Philosophy

Network reconstruction is the process of recovering a unique structured representation of some dynamic system using input-output data and some additional knowledge about the structure of the system. Many network reconstruction algorithms have been proposed in recent years, most dealing with the reconstruction of strictly proper networks (i.e., networks that require delays in all dynamics between measured variables). However, no reconstruction technique presently exists capable of recovering both the structure and dynamics of networks where links are proper (delays in dynamics are not required) and not necessarily strictly proper.

The ultimate objective of this dissertation is to develop algorithms capable of reconstructing proper networks, and this objective will be addressed in three parts. The first part lays the foundation for the theory of mathematical representations of proper networks, including an exposition on when such networks are well-posed (i.e., physically realizable). The second part studies the notions of *abstractions* of a network, which are other networks that preserve certain properties of the original network but contain less structural information. As such, abstractions require less a priori information to reconstruct from data than the original network, which allows previously-unsolvable problems to become solvable. The third part addresses our original objective and presents reconstruction algorithms to recover proper networks in both the time domain and in the frequency domain.

Keywords: dynamic systems, networks, network reconstruction, target specificity, system identification, learning, linear systems, system representations, state space models, generalized state space models, dynamical structure functions, dynamical network functions, DSF, DNF, multi-DSF, feedback, well-posedness, algebraic loops, representability, abstraction, immersion, identifiability, informativity, data, proper systems, strictly proper systems, causality, strict causality, frequency domain, time domain

ACKNOWLEDGMENTS

I would like to give a special thanks to Arne Dankers, one of the co-authors on the papers that became Chapters 2 and 3. His work was some of the earliest into the notion of well-posedness of DSFs, and his questions opened up our direction of research into the meaning, computation, and representability of abstractions and immersions. Thanks also to Vasu Chetty, whose network reconstruction work formed the foundation for Chapter 4. I am grateful for the time he took to teach me what I needed to know about reconstruction. I would like to thank my adviser, Sean Warnick, for supporting me in this project, for teaching me, and for providing a wonderful educational experience. I would not be where I am today without his direction and guidance. Thanks also to my committee and to the members of the IDeA Labs at BYU for their advice and critiques, which led me to improve and to grow.

I would also like to thank my family, and especially my beautiful fiancée, Sarah, for their kindness, love, and support throughout this process. Finally, I give thanks to my God for His love and His blessings which I have felt throughout this process.

Table of Contents

List of Figures

List of Algorithms

Chapter 1

Introduction and Motivation

This work is concerned with structured (or network) representations of dynamic systems, and, in particular, network representations of linear and time-invariant (LTI) systems. Our model of choice to represent dynamic LTI networks is the *dynamical structure function* (DSF) and its generalization called the *dynamical network function* (DNF).

1.1 Notions of Structure and a Spectrum of Models

LTI systems can be represented by many models, each containing different levels of structural information, where we define structure as the causal dependencies and independencies between manifest and latent variables. Two such models, the *state space model* and the *transfer function*, are well-known and have been studied for decades in various engineering disciplines.

The transfer function is a matrix of operators that map a set of input signals to a set of output signals. In other words, it is a black-box model that defines the causal relationship from each input into the LTI system and the resulting outputs.

The state space model is a differential-algebraic system of equations containing the mapping of the inputs into latent (unmeasured) internal states, the mapping of the internal states to themselves, and the mapping from the inputs and the internal states to the outputs. As such, there are more causal relationships modeled in the state space model than the transfer function, and we say that the state space model contains more structural information than the transfer function.

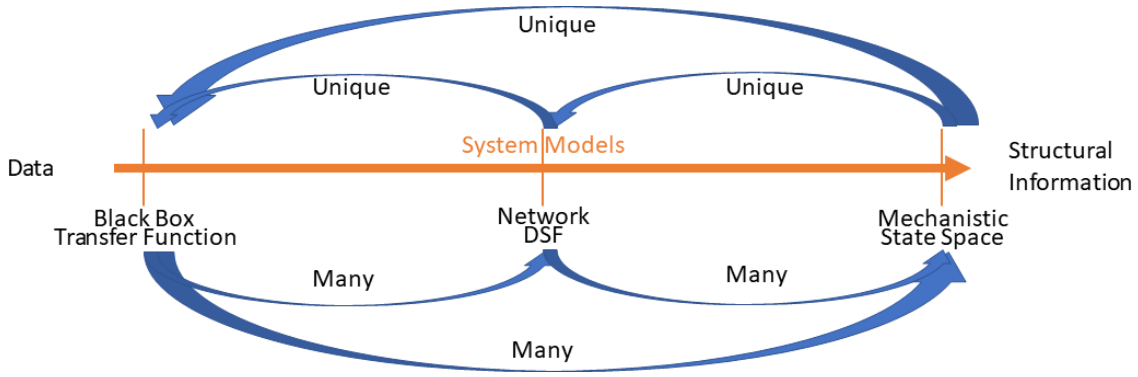


Figure 1.1 The transformations from a more-structured model to a less-structured model are unique where the transformations from a less-structured model to a more-structured model are not.

The DSF and the DNF are each a factorization of the transfer function. Each edge in these networks shows the mapping of input to output which is independent of all other inputs and outputs in the network [1, 2]. In [3], it is shown that there are DSF representations of LTI networks that contain as much structural information as the state space model, other networks that contain as little information as the transfer function, and still other networks that contain many intermediate levels of information. Due to this flexibility in structural representation, the DSF has become the representation of choice in many applications, such as finance, biochemistry, and the security of cyber-physical systems [4–8].

It is known (see, for instance [1, 3] and Chapters 2 and 4) that the transformation from a more structured model to a less structured model is unique. However, for any given model that is less structured, there exist many (typically infinitely many) more structured models that we can transform into that less structured model. See Figure 1.1.

The problem of discovering any more-structured model from a less-structured model such that the complexity of the more-structured model is minimized across the set of all possible such models is known as the *minimal realization problem*. The problem of recovering a specific and more-structured model given a less-structured model and additional a priori knowledge or assumptions about the network is known as *network reconstruction*.

More specifically, suppose we are given the following:

- Data
- A model class, where a set of parameters distinguishes model instances
- A metric comparing predicted model outputs to data

A *learning* problem is the problem of selecting the unique model in the given model class (i.e., some set of parameters) that best fits the data according to the metric. This problem is widely studied in many fields (e.g., machine learning in computer science, econometrics in economics, regression in mathematics) and can be considered one of the most important problems in science.

System identification is a broad subclass of learning problems where the model class is a set of dynamic models, and the data is the set of manifest variables in the dynamic system of interested measured over some time horizon. Much of the theory in system identification revolves around finding black-box or input-output models such as the transfer function. The network reconstruction problem is a subclass of system identification problems where the model class is a structured model, such as the DSF or a state space model.

The reason that most of the system identification theory focuses on input-output models is that input-output data is only sufficient to recover an input-output model. No matter how long we watch the data or how rich the data we receive is, we will never be able to recover anything more than an input-output model. Moreover, as discussed previously, there are many (almost always infinitely-many) structured models corresponding to any single input-output model. Thus additional information–specifically information about the structure of the network–is required to choose a unique structured model. This additional a priori information is known as the *identifiability conditions* (or informativity conditions in some works) required for reconstruction. The identifiability conditions take the form of knowing that specific edges in the network are either zero or linear combinations of other edges.

Furthermore, as the amount of structural information contained within a model increases, the amount of a priori information required by the identifiability conditions likewise

increases. Hence the power and utility of using the DSF as a model of an LTI system, we can tune the amount of structural information contained within the model according to the amount of a priori information we have merely by measuring more or less of the system. Thus we can choose a level of abstraction (see Chapter 3) that is cheaper to learn than a state space model and yet contains more structural information than a black box (transfer function) model.

Note that, in many scenarios, knowing an unstructured model, such as the transfer function, is sufficient for the desired problem (e.g., a predictive modeling or a control problem). However, having a structured model does provide the following benefits that are unavailable for black-box models (and these benefits provide the key motivation for this work):

- **Mechanistic Insight:** A significant part of science revolves around the construction of mechanistic models that improve our understanding of the world around us. Mechanistic models are inherently structured, and give us insight into the fundamental principles that drive the behaviors that we observe. Network Reconstruction is capable of providing mechanistic insight that is unavailable in black box models.
- **Local Sensitivity Analysis:** A structured model allows us to analyze how local perturbations in the network can affect local and global behavior. For instance, in [6, 9–12], it is shown that a small perturbation to local edges within a DSF can cause cascading failures across the entire network, causing it to go unstable. These papers demonstrate how to find which edges are vulnerable to such an attack and how much energy is required on each edge to cause the network to fail. These papers also provide information on how to design such networks so that they are secure to such attacks.
- **Local Control:** Suppose that we only have access to a piece of a network, and wish to control one manifest variable without our attack being detected by another (sometimes called a stealthy attack). Then a structured representation of the system can tell us if this is possible, and if so, how to do it. It can also tell us how to prevent such an attack. See [13] for more information.

- **Robust Control:** The robust control problem, often performed on a black-box model, assumes that our model of a given system is imperfect. It then seeks to control the system by controlling not only the given model but all other models within some distance specified by our uncertainty about the model. Unfortunately, sometimes this distance is too big making the control problem impossible. Moreover, sometimes, even if control is possible, it is too conservative, leading to inefficiencies in operation.

  However, if we had a structured model instead of a black-box model, we could potentially fine-tune the ball of uncertainty around model by reducing the distance in specific dimensions that are certain. Thus, a structured model would allow us to build less-conservative robust controllers. This concept is related to the vulnerability analysis described previously; however, to the author's knowledge, it has never been formally explored.

A study of the network reconstruction problem is the main topic of Chapter 4 in this work.

Example 1.1.1: A Graphical View of Structure

Suppose that we are seeking to model an LTI system containing three inputs $u = [u_1, u_2, u_3]'$ and three outputs $y = [y_1, y_2, y_3]'$. Suppose that this LTI system is modeled with the following state space model containing six states $x = [x_1, \ldots, x_6]'$:

$$\begin{aligned} x(t+1) &= Ax(t) + Bu(t), \\ y(t) &= Cx(t) + Du(t), \end{aligned} \tag{1.1}$$

where

$$A = \begin{bmatrix} \frac{3}{4} & 0 & 0 & 0 & 0 & \frac{6}{5} \\ -\frac{1}{10} & -\frac{7}{20} & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{17}{20} & -1 & 0 & 0 \\ 0 & -\frac{73}{100} & 0 & \frac{19}{20} & 0 & 0 \\ 0 & 0 & \frac{43}{100} & 0 & -\frac{3}{5} & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{5} & \frac{11}{20} \end{bmatrix}, \quad B = \begin{bmatrix} \frac{7}{5} & 0 & 0 \\ 0 & -\frac{1}{4} & 0 \\ 0 & 0 & \frac{3}{4} \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \tag{1.2}$$

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \quad D = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}.$$

Note that matrix A defines the computational (or causal) mappings from the current state $x(t)$ to the next state $x(t+1)$. The matrix B defines the computational mappings from the current input $u(t)$ to the next state $x(t+1)$. The matrix C defines the computational mappings from the current state $x(t)$ to the current output $y(t)$. And the matrix D defines the computational mappings from the current input $u(t)$ to the current outputs $y(t)$.

As such, we can draw a graph defining the computational (causal) dependencies within this network. In this graph, there are 12 nodes, one for each input, output, and state. An edge between some state node x_i and some state node x_j exists if and only if $A_{ji} \neq 0$. An edge between some input node u_i and some state node x_j exists if and only if $B_{ji} \neq 0$. An edge between some state node x_i and some output node y_j exists if and only if $C_{ji} \neq 0$. And an edge between some input u_i and some output y_j exists if and only if $D_{ji} \neq 0$ (and since $D = 0$ for this example, no such edges will exist). Hence, we draw the graph shown in Figure 1.2.

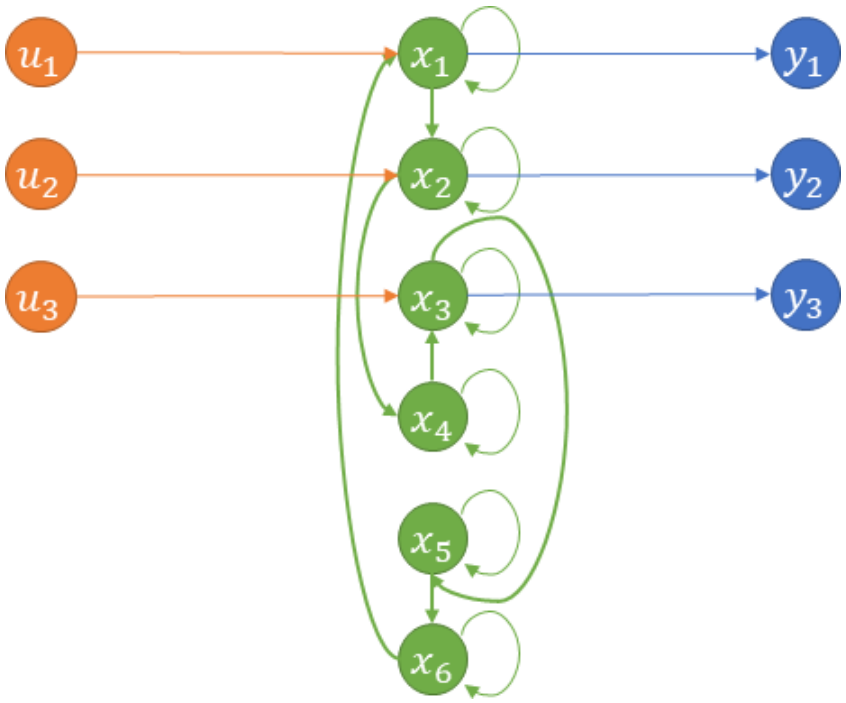


Figure 1.2 A graph of the state space model defined by (1.1) and (1.2. Green nodes are the states x, orange nodes are the inputs u, and blue nodes are the outputs y. Green edges show the computational (causal) dependencies within states as defined by A. Orange edges show the computational dependencies from inputs to states defined by B. Blue edges show the computational dependencies from states to outputs defined by C.

Notice from the definition of C that $y_1 = x_1$, $y_2 = x_2$, and $y_3 = x_3$. Thus we can discuss the direct computational dependencies from $y_i = x_i$ to $y_j = x_j$. From Figure 1.2 that there exists a dependency from $y_3 = x_3$ to $y_1 = x_1$ that does not pass through $y_2 = x_2$ or any u_i (though it does pass through x_5 and x_6. Likewise, there exists a dependency from u_1 to y_1 that does not pass through $y_2 = x_2$, $y_3 = x_3$, or any other u_j. However, there does not exist a dependency from $y_1 = x_1$ to $y_3 = x_3$ that does not first pass through $y_2 = x_2$.

Another graph that we can draw shows an edge from u_i to u_j if and only if there is some path from u_i to y_j in Figure 1.2. Notice that there exists some path from every input to every output, and so this graph (Figure 1.3) is full.

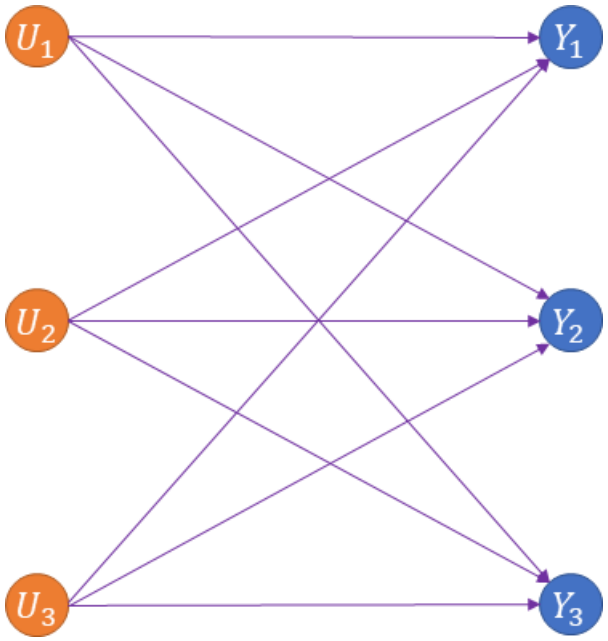


Figure 1.3 A graph of all of the dependencies from the inputs U to outputs Y shown in Figure 1.2. This is also the graphical representation of the transfer function G (1.5) representation of this example system, where an edge from input U_i to output Y_j exists if and only if $G_{ji}(z) \neq 0$.

The transformation from the discrete-time state space model to the transfer function is well-known and accomplished by taking the $\mathcal{Z}$-transform of (1.1). The transfer function defines the relationship between U and Y (the Laplace transform of u and y respectively) as

$$Y = G(z)U, \tag{1.3}$$

where $G(z)$ is a matrix of proper rational functions (i.e., a polynomial divided by another polynomial where the degree of the polynomial in the numerator is no greater than the degree of the polynomial in the denominator). We have that the transfer function of this example is

$$G(z) = C(zI - A)^{-1}B + D \tag{1.4}$$

$$= \begin{bmatrix} \frac{n_{11}}{d} & \frac{n_{12}}{d} & \frac{n_{13}}{d} \\ \frac{n_{21}}{d} & \frac{n_{22}}{d} & \frac{n_{23}}{d} \\ \frac{n_{31}}{d} & \frac{n_{32}}{d} & \frac{n_{33}}{d} \end{bmatrix}, \quad \text{where} \tag{1.5}$$

$$d = z^6 - 2.150z^5 + 0.825z^4 + 0.939z^3 - 0.622z^2 - 0.06z + 0.07,$$

$$n_{11} = 1.4z^5 - 1.96z^4 - 0.315z^3 + 1.078z^2 - 0.622z - 0.131,$$

$$n_{12} = -0.014z^4 + 0.025z^3 - 0.005z^2 - 0.009z + 0.004,$$

$$n_{13} = -0.1z^2 - 0.001z + 0.003,$$

$$n_{21} = -0.019,$$

$$n_{22} = -0.25z^5 + 0.625z^4 - 0.425z^3 - 0.086z^2 + 0.186z - 0.050,$$

$$n_{23} = -0.183z^3 + 0.128z^2 + 0.067z - 0.045,$$

$$n_{31} = 0.077z^2 - 0.046z - 0.026,$$

$$n_{32} = -0.001z + 0.001,$$

$$n_{33} = 0.75z^5 - 0.975z^4 - 0.210z^3 + 0.526z^2 - 0.020z - 0.062.$$

The graph of the transfer function, which plots an edge from U_i to Y_j if and only if $G_{ji}(z) \neq 0$ is precisely the graph in Figure 1.3 described previously.

We can also collect all of the dependencies in Figure 1.4 and draw an alternate graph where an edge between U_i and Y_j or between Y_i and Y_j if and only if there exists a dependency from i to j independent of all other inputs U or outputs Y. It is up to the designer to include the self-loops in this graph, and we choose not to in this example. This graph is known as the *signal structure* of this dynamic network (see Figure 1.4). Note that, often, the signal structure also contains weights on the edges of the graph which are equal to the operators in the DSF.

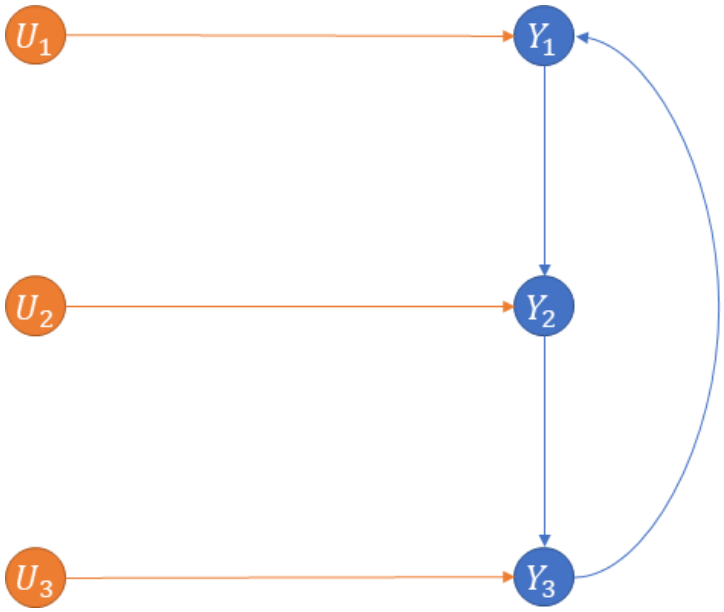


Figure 1.4 A graph of all of the **direct** dependencies (in contrast to all dependencies as shown in Figure 1.3 from the inputs U to outputs Y shown in Figure 1.2. This is also the graphical representation of the DSF $(Q(z), P(z))$ (1.7) representation of this example system, where an edge from input U_i to output Y_j exists if and only if $P_{ji}(z) \neq 0$ and an edge from output Y_i to output Y_j exists if and only if $Q_{ji}(z) \neq 0$.

If we follow the procedure contained in [1] and Chapter 2 of this work to convert the state space model in (1.1) to a DSF, we get the relationship

$$Y = Q(z)Y + P(z)U, \tag{1.6}$$

where U and Y are again the $\mathcal{Z}$-transform of the inputs U and the outputs Y respectively and where $Q(z)$ and $P(z)$ are both matrices of rational functions (as was the transfer

function) given by

$$
\begin{aligned}
Q(z) &= \begin{bmatrix} 0 & 0 & \frac{1032}{25(4z-3)(5z+3)(20z-11)} \\ -\frac{2}{20z+7} & 0 & 0 \\ 0 & \frac{292}{(20z-19)(20z-17)} & 0 \end{bmatrix}, \quad \text{and} \\
P(z) &= \begin{bmatrix} \frac{28}{5(4z-3)} & 0 & 0 \\ 0 & -\frac{5}{20z+7} & 0 \\ 0 & 0 & \frac{15}{20z-17} \end{bmatrix}.
\end{aligned} \tag{1.7}
$$

We can draw a graph of this DSF where an edge from U_i to Y_j exists if and only if $P_{ji}(z) \neq 0$ and an edge from Y_i to Y_j exists if and only if $Q_{ji}(z) \neq 0$. Notice that this graph is precisely the signal structure given in Figure 1.4. Thus, the graphical representation of a DSF is a signal structure and the DSF represents the direct causal dependencies between and within inputs and outputs. If we had chosen to find a DNF instead of a DSF, the graph would have been the same as this signal structure, but where we chose to include the self-loops as well. This highlights the difference between the DNF and the DSF: the DNF includes self-loops where the DSF does not. See Section 3.4 for more details.

Also notice that there is a clear ring structure between Y_1, Y_2, and Y_3 in the signal structure. This ring is also apparent in the graph of the state space model, though there are many additional details graphed as well; as such, the state space model contains more structural information than this DSF. In contrast, no cycles between outputs can ever be seen in the transfer function. In this example, it can only show that every output is eventually dependent on every input; as such, the transfer function contains less structural information than this DSF.

1.2 Proper and Strictly Proper Networks

As shown in Example 1.1.1, the DSF is parameterized by the pair $(Q(z), P(z))$ which defines the relationship between inputs U and outputs Y as

$$Y = Q(z)Y + P(z)U. \tag{1.8}$$

In the frequency domain, both $Q(z)$ and $P(z)$ are matrices of proper rational functions. Matrix $Q(z)$ defines the causal dependencies of Y's on other Y's and matrix $P(z)$ defines the causal dependencies of Y's on U's.

Suppose that we are dealing with a discrete-time system, and suppose that each $Q_{ji}(z)$ is *strictly proper* (and similarly for entries in $P(z)$). Then the computation of Y_j is dependent on the past of Y_i but is independent on the present or the future of Y_i. For instance, suppose that $y_j(t)$ and $y_i(t)$ are computed at the discrete times $t = 0, 1, \ldots, \infty$. If we interpret our network probabilistically, then this means that

$$\mathbb{P}\left(y_j(t) \,\middle|\, y_i(0), y_i(1), \ldots, y_i(\infty)\right) = \begin{cases} \mathbb{P}\left(y_j(t) \,\middle|\, y_i(0), y_i(1), \ldots, y_i(t-1)\right) & \text{if } Q_{ji}(z) \neq 0 \\ \mathbb{P}\left(y_j(t)\right) & \text{if } Q_{ji}(z) = 0 \end{cases} \tag{1.9}$$

If every entry in $Q(z)$ and $P(z)$ are strictly proper, then we say that the DSF $(Q(z), P(z))$ is strictly proper.

On the other hand, if $Q_{ij}(z)$ is *proper* (and similarly for entries in $P(z)$), then the computation of Y_j is dependent on the past *and* the present of Y_i, but is independent of the future of Y_i. For instance, suppose that $y_j(t)$ and $y_i(t)$ are computed at the discrete times $t = 0, 1, \ldots, \infty$. The probabilistic interpretation of this gives

$$\mathbb{P}\left(y_j(t) \,\middle|\, y_i(0), y_i(1), \ldots, y_i(\infty)\right) = \begin{cases} \mathbb{P}\left(y_j(t) \,\middle|\, y_i(0), y_i(1), \ldots, y_i(t)\right) & \text{if } Q_{ji}(z) \neq 0 \\ \mathbb{P}\left(y_j(t)\right) & \text{if } Q_{ji}(z) = 0 \end{cases} \tag{1.10}$$

If every entry in $Q(z)$ and $P(z)$ are proper, then we say that the DSF $(Q(z), P(z))$ is proper.

Note that the set of strictly proper DSFs is a strict subset of the set of proper DSFs. In particular, if $Q(z)$ is proper, then it is equal to the sum of a strictly proper matrix and a static matrix (i.e., a matrix consisting only of real-valued entries instead of rational functions). In this dissertation, we are concerned primarily with the set of networks that are proper, but not strictly proper. We are especially concerned with those DSFs that have at least one entry in $Q(z)$ that is proper but not strictly proper.

Example 1.2.1: Proper and Strictly Proper Networks

Suppose that our network is given by

$$\begin{bmatrix} Y_1 \\ Y_2 \end{bmatrix} = \begin{bmatrix} 0 & Q_{21}(z) \\ Q_{12}(z) & 0 \end{bmatrix} \begin{bmatrix} Y_1 \\ Y_2 \end{bmatrix} + \begin{bmatrix} P_{11}(z) & 0 \\ 0 & P_{22}(z) \end{bmatrix}. \tag{1.11}$$

Suppose also, for simplicity in plotting, that each Y_i is dependent on values of Y_j, U_i, and U_j at no more than one time-step in the past (such a situation arises if the DSF is a full-state DSF as defined in Definition 3.6.2; see Remark 4.2.1). The graph of this network is given on the left of Figures 1.5-1.7.

Suppose that we "unwrap" the network to show the dependency of the computation of each $y_i(t)$ on current and previous values of $y_j(t)$ and $u_i(t)$. Note that if we were to draw the more general case where each $Q_{ij}(z)$ and $P_{ij}(z)$ were strictly proper and dependent on all past values (for instance, if we weren't dealing with a full-state DSF), then there would be many more edges in this graph connecting each $y_i(t)$ to *all* previous values of $y_j(t)$ and $u_i(t)$. This complexity highlights one reason why we choose to use the DSF in this work–the DSF leverages operator theory to write down a more abbreviated version of the network and to make the analysis of the network much more simple). The graph showing these computational dependencies is given in Figure 1.5. Note also that the graph in Figure 1.5 is a directed and acyclic polytree.

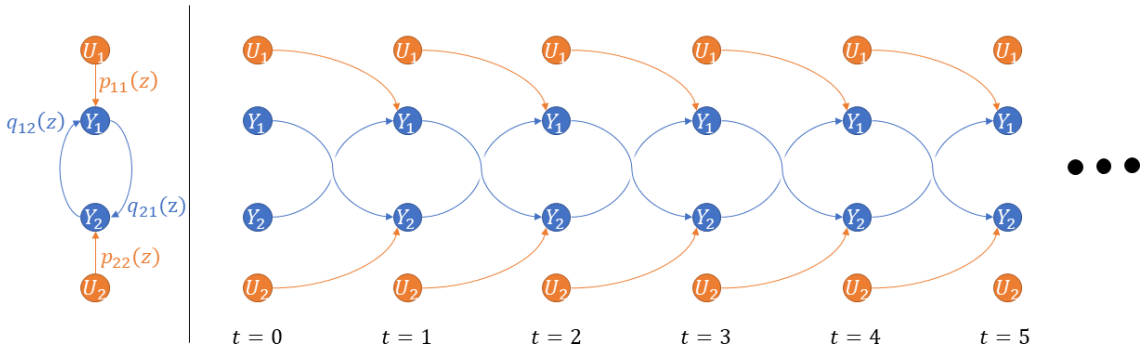


Figure 1.5 A graph of the structural and temporal dependencies of (1.11) where every entry in $Q(z)$ and $P(z)$ are strictly proper. On the left, we have a graph of $Q(z)$ and $P(z)$, and on the right, we have "unwrapped" $Q(z)$ and $P(z)$ to show the temporal dependence of each U and Y on previous values of U and Y. Since $Q(z)$ and $P(z)$ are strictly proper, each node at time t is dependent only on nodes strictly in the past of t. Blue edges are the strictly proper components of $Q(z)$ and orange edges are the strictly proper components of $P(z)$.

If $P(z)$ were instead proper and not necessarily strictly proper, then we must add additional edges from each $u_i(t)$ to $y_i(t)$ (Figure 1.6). Notice, however, that such a network is still a directed and acyclic polytree.

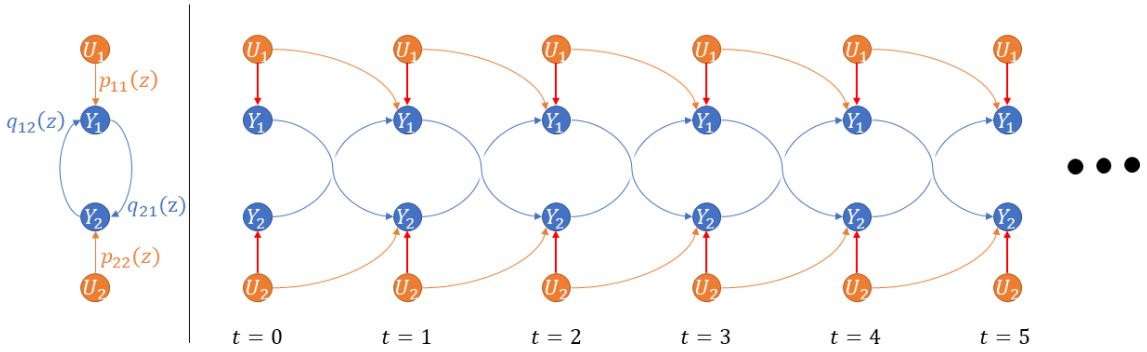


Figure 1.6 A graph of the structural and temporal dependencies of (1.11) where every entry in $Q(z)$ is strictly proper, and every entry of $P(z)$ is proper but not strictly proper. On the left, we have a graph of $Q(z)$ and $P(z)$, and on the right, we have "unwrapped" $Q(z)$ and $P(z)$ to show the temporal dependence of each U and Y on previous and present values of U and Y. Since $Q(z)$ is strictly proper but and $P(z)$ is proper, each node $y_j(t)$ is dependent on $y_i(t)$ $(i \neq j)$ strictly in the past of t, but dependent on $u_1(t)$ and $u_2(t)$ in both the past and the present of t. Blue edges are the strictly proper components of $Q(z)$, orange edges are the strictly proper components of $P(z)$, and red edges are the static components of $P(z)$.

A more interesting case occurs when both $Q_{12}(z)$ and $Q_{21}(z)$ are proper but not strictly proper. In this case, we add a cycle between each $y_1(t)$ and $y_2(t)$ (Figure 1.7). Such a cycle is sometimes called an *algebraic loop* and can lead to inconsistencies in the computation of the system (an issue known as ill-posedness, see Chapter 2). One of the main objectives of this work (Chapter 4) is to provide algorithms capable of identifying such networks using input-output data.

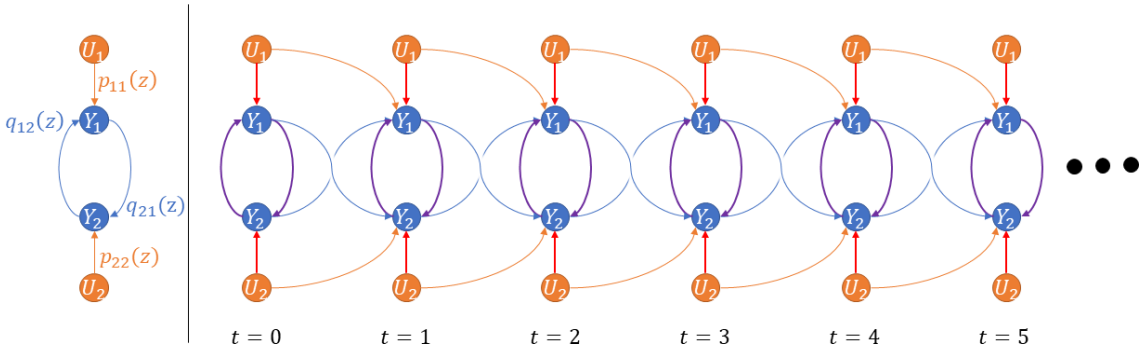


Figure 1.7 A graph of the structural and temporal dependencies of (1.11) where every entry in $Q(z)$ and $P(z)$ are proper and not strictly proper. On the left, we have a graph of $Q(z)$ and $P(z)$, and on the right, we have "unwrapped" $Q(z)$ and $P(z)$ to show the temporal dependence of each U and Y on previous and present values of U and Y. Since $Q(z)$ and $P(z)$ are proper but not strictly proper, each node at time t is dependent on nodes in the past and the present of t. Blue edges are the strictly proper components of $Q(z)$, purple edges are the static components of $Q(z)$, orange edges are the strictly proper components of $P(z)$, and red edges are the static components of $P(z)$.

Due to physical limitations (e.g., the speed of light), it can be argued that proper networks do not exist in nature, but are merely mathematical constructs. These networks, however, often arise when we attempt to model systems using discrete-time models. If the system contains significant dynamics that move more quickly than the sampling time, then those dynamics will be approximated using a static edge.

For instance, consider a network model of a stock market where each signal $y_i(t)$ is a measure of the stock price of security i (e.g., $MSFT$, $GOOG$, $AAPL$, etc.) at time t. Suppose that we only measure stock prices at the end of each day. Then $Q_{MSFT,GOOG}(z)$ represents the causal dependency of the price of $MSFT$ on day t on the prices of $GOOG$ from the beginning of time until day t. Since the prices of both securities change throughout the

day, and since such intra-day changes in $GOOG$ likely cause an intra-day change in $MSFT$, some dynamics are moving more quickly than our daily price measurement. The static term will approximate these dynamics from $GOOG$ to $MSFT$. Higher-frequency sampling will reduce the significance of this static dependence to the limit of infinity (or at least the clock speed of the market mechanism computing prices) where this static dependency is zero.

1.3 Overview

The central theme behind this work is a study of networks that are proper and not necessarily strictly proper, as motivated by the previous section.

Chapter 2 is based off of the published works [14, 15]; however, it also contains additional novel research above and beyond those works (see Section 2.1.1 for more details). An abbreviated version of the chapter, at the time of the dissertation defense, was submitted as the stand-alone paper [16].

Chapter 2 begins by introducing (for the first time in literature) the DNF as a generalization of the DSF since analysis is easier over the DNF, and the results quickly translate to the DSF as well. It demonstrates how to transform a state space model into a DNF and then shows that no state space model can generate a $Q(z)$ that is proper but not strictly proper. However, it also shows that an interconnection of state space models can lead to a proper $Q(z)$. Thus, it uses the generalized state space model to represent an interconnection of state space models and shows how to transform this model into a DNF. It also shows that all proper DSFs have a generalized state space model representation.

Chapter 2 then continues by showing that the algebraic loops allowed by proper a $Q(z)$ can lead to a network that is ill-posed, meaning that the signals y either do not exist, are not unique, or are not causally dependent on the other signals in y or u. The primary contribution of this chapter is to provide conditions over $Q(z)$ ensuring that it is well-posed (y exists, is unique, and is proper).

Chapter 3 is based off the published works [3, 15, 17]; however, this chapter also contains additional novel research above and beyond those works (see Section 3.1.3). An abbreviated version of that chapter will be submitted as a stand-alone paper to the IEEE Transactions on Automatic Control.

In Chapter 3, we study abstractions of DNFs, which are other DNFs that share the same input-output behavior and other properties from the original but contain less structural information than the original network. One abstraction, in particular, is known as the *immersion*, which is an abstraction of a DSF which results in another DSF.

Chapter 3 introduces *representability* of abstractions, which is related to well-posedness. It shows that–where well-posedness is the study of the generic existence, uniqueness, and properness of the transfer function found from some network–representability is the study of the generic existence, uniqueness, and properness of the abstraction itself. For each abstraction, it presents conditions of representability and shows that if an abstraction is representable, then it is well-posed if and only if the base network is well-posed.

Finally, Chapter 3 concludes by using the immersion to define a spectrum of models. To do so, it introduces a more general DNF known as the *multi-DNF* and a more general immersion over the DNF known as the *stacked immersion*. It shows that there exists a multi-DNF that contains as much structural information as the state space model. It shows that there also exists a stacked immersion of this multi-DNF with as little structural information as the transfer function. It also shows that all other stacked immersions of this multi-DNF form a partial ordering of structural information ranging from the state space model to the transfer function.

Chapter 4 is based off the works [12, 18]; however, it contains novel results above and beyond those works (which we discuss below). An abbreviated version of this chapter will be submitted as a stand-alone paper to the IEEE Transactions on Automatic Control.

Chapter 4 utilizes the theory built in the previous two chapters to study the network reconstruction problem. It presents two algorithms for solving the network reconstruction

problem, the first in the frequency domain and the second in the time domain, outlining the advantages and disadvantages of each.

The frequency-domain algorithm to reconstruct DSFs existed long before this work (see [19]), and though it is capable of reconstructing proper (and not necessarily strictly proper) networks, all previous work assume that the networks are strictly proper. Here, we demonstrate that the algorithm reconstructs not only proper networks but also networks where entries in $Q(z)$ and $P(z)$ are taken from general fields. We also extend the frequency-domain algorithm to reconstruct DNFs and provide the necessary and sufficient identifiability conditions to solve this problem.

The time-domain algorithm was first presented in [20] and extended in [12, 18], but those works required that the network be a strictly proper DSF. In this chapter, we generalize the algorithm so that it is capable of reconstructing proper DNFs and provide the necessary and sufficient identifiability conditions for this algorithm as well. We also provide, for the first time, necessary conditions under which the input-output data is informative enough to reconstruct the network.

REMARK 1.3.1: To clarify notation, Chapters 2 and 3 build their theory around continuous-time systems. Thus, all rational functions in those chapters (including those in the DSF matrices) are written in terms of the Laplace variable $s \in \mathbb{C}$. In this introduction and in Chapter 4, the discussion centers around discrete-time systems. Thus, all rational functions in these chapters are written in terms of $z \in \mathbb{C}$. While there are semantic differences between s and z dealing with the differences between continuous- and discrete-time systems, most of the analysis in this work applies naturally to both situations, with the time-domain network reconstruction algorithm and the examples above being the only notable exceptions.

As such, to demonstrate that we are agnostic between the frequency and the time domain (and for notational brevity), we often drop the dependency of G, Q, P, W, V (and other matrices of rational functions) on s or z.

Chapter 2

On the Well-Posedness of LTI Dynamic Networks[1]

This chapter is concerned with dynamic networks of linear, time-invariant systems defined over matrices of rational functions in a complex variable. We show that state space models are not sufficient to characterize the full space of interest of these networks; thus we must utilize a more generalized state space model which contains information about the feedback interconnection of multiple state space models. We then define the notion of well-posedness over these networks as a regularity condition over feasible mathematical models for physical systems. With this definition, we characterize conditions over the dynamic networks and their corresponding generalized state space models under which they are well-posed.

2.1 Introduction

We can model the dynamics of linear, time-invariant (LTI) systems in many ways. For instance, we can use state space model to capture both the input-output behavior of the system as well as the internal structure, where internal structure defined as the computational dependencies among the latent and the manifest variables within the network. We can also use the transfer function to capture the input-output behavior only while abstracting away the information about the internal structure of the system.

The *dynamical structure function* (DSF) is a type a dynamic network, another model of dynamic systems. The work [3] shows that these dynamic networks are capable of representing as much structural information as a state space model, as little as a transfer function, or with

[1] A variation of this chapter was submitted to the Transactions on Automatic Control [16].

many intermediate levels of structural information. Because of this flexibility in representation, dynamic networks have increasingly become the representation of choice in applications where knowledge of both the dynamic behavior and the computational structure of the system are required [4–8]. This work is focused on general dynamic networks, which we represent with a generalization of the DSF called the *dynamical network function* (DNF).

When defining a mathematical model to represent a physical dynamic system, it is possible that the model does not make physical sense. For instance, if the inputs do not uniquely determine the outputs of the of the model into that model, it would be sensible not to consider this a reasonable model since this phenomenon cannot occur in the physical system we are modeling [21–25]. We use the notion of *well-posedness* to restrict a set of models to a subset that makes physical sense. As Jan Willems states [21]:

> Well-posedness is essentially a modeling problem. It expresses that a mathematical model is, at least in principle, adequate as a description of a physical system ... Well-posedness thus imposes a regularity condition on feasible mathematical models for physical systems ... In other words, since exact mathematical models would always be well posed, one thus requires this property to be preserved in the modeling.

The purpose of this work is to provide the necessary and sufficient conditions under which a DNF is well-posed.

2.1.1 Related Work and Contributions

This chapter is concerned with the study of well-posedness of DNFs in particular and is a direct extension of [14, 15]. In addition to the material contained in those previous works, this chapter includes the following novel contributions:

(i) The extension of [14, 15] and other works from DSFs into the realm of DNFs (See Remark 2.3.1).

(ii) A proof that state space model can only be converted into a DNF with strictly proper entries in matrix Q (Theorem 2.4.3), which is important since all such models are always well-posed (Corollary 2.5.9).

(iii) The process of converting a generalized state space model to a DNF (Section 2.4.2), with a proof that for any proper (and not necessarily strictly proper) DNF, there exists a generalized state space model that can we can transform into that DNF (Theorem 2.4.10).

(iv) A demonstration that the DNF found from converting a generalized state space model (representing an interconnection of state space models) into a DNF results in the same model that we would find if we converted each sub-model into a DNF first and then interconnected the resultant DNFs (Example 2.4.11).

(v) The connection of well-posedness of DNFs to the well-posedness of generalized state space models (Section 2.5.3 and Theorem 2.5.8).

We acknowledge that the conditions for well-posedness of DSFs (sometimes called linear dynamical networks) contained in Proposition 2.5.5 have been stated prior to [14, 15] and this work (see, for instance, [8, 26]); however, the key novelty of this work is the proof that this condition is a necessary and sufficient implication of Jan Willem's original definition for the well-posedness of networked dynamical systems [21] (see Section 2.5.2) as well as the connection of this condition to the well-posedness of interconnected state space models (see Section 2.5.3).

2.2 Preliminaries

This work is largely concerned with functions of the form $g(s) = \frac{n(s)}{d(s)}$, where $n(s)$ and $d(s)$ are functions in $s \in \mathbb{C}$. If the degree of $n(s)$ is no greater than the degree of $d(s)$, we say

that $q(s)$ is *proper*. If the degree of $n(s)$ is strictly less than the degree of $d(s)$, then we say that $g(s)$ is *strictly proper*.

We are also concerned with matrices consisting of these rational functions. We say that the matrix $G(s) = [g_{ij}(s)]$ is *proper* if every rational function $g_{ij}(s)$ is proper, and likewise $G(s)$ is *strictly proper* if every $g_{ij}(s)$ is strictly proper. In the vein of [27], we define $RP^{m\times n}$ as the set of all $m \times n$ matrices of proper (and not necessarily strictly proper) rational functions. Also, for notational simplicity, we will often drop the dependency of these matrices on s; i.e., we write $G(s)$ simply as G.

We say that $G \in RP^{p\times p}$ is *invertible almost everywhere* if it is singular only for a finite choice of values $s \in \mathbb{C}$. For brevity, from this point forward, when we say that G is invertible, we mean that it is invertible almost everywhere. See the appendix for additional results on functions and matrices of rational functions that will be utilized later in this work.

2.3 Background: Dynamic Networks

Consider N manifest (or visible, or measured) real-valued signals $w(t) = [w_1(t), \ldots, w_N(t)]'$, where $t \in \mathbb{R}$ or $t \in \mathbb{Z}$ (in this work we will focus on $t \in \mathbb{R}$, although all the results hold for $t \in \mathbb{Z}$ as well). The behavior of any such system can be defined by a constraint $f(w) = 0$ for some operator, f, distinguishing allowed values of the manifest variables from disallowed values. In general, there may be many such representations with the same solution set, or behavior. If f is bijective, then we can say that the system is autonomous. However, if f is not injective, then the manifest signals can be partitioned (possibly non-uniquely) into *inputs* and *outputs*, where the outputs are uniquely specified for any choice of input signal. This work considers linear finite-dimensional systems, where a partition can always be found (possibly non-uniquely) for which the outputs are not only uniquely specified given arbitrary inputs, but they are also causally determined from these inputs. Note that a specification of which signals are inputs and which are outputs must be given as additional information–it

can not be derived in general from f–and this partition introduces a notion of causality on the system.

In this work, we are concerned primarily with LTI systems. Thus, we can write the behavior as $f(w) = \bar{W}w = 0$, where $\bar{W}$ is an $N \times N$ matrix of SISO causal operators[2]. Let $p = \operatorname{rank} \bar{W}$ and $m = N - p$. By partitioning w into m inputs[3] u and p outputs y, we can again rewrite this behavior, without loss of generality, as

$$\begin{bmatrix} (I - W) & -V \\ 0 & 0 \end{bmatrix} \begin{bmatrix} y \\ u \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \implies y = Wy + Vu. \tag{2.1}$$

where W is a $p \times p$ matrix and V is a $p \times m$ matrix. We call the tuple (W, V) the *dynamical network function* (DNF) of this system. Matrix V defines the causal dependencies of each output y on each input u, independent of all other manifest variables in the network. Matrix W defines the causal dependencies of each output y on every other output y, again independent of all manifest variables in the network. Thus, the DNF defines a network of causal dependencies among manifest signals. This network can be rigorously characterized by a graph called the *signal structure*, with nodes representing each manifest signal and directed edges labeled with (possibly) dynamic operators characterizing the causal dependency of one signal on another. See Figure 2.1 for examples of signal structures.

REMARK 2.3.1: Previous works (e.g., [1, 14, 15]) contain treatments of the *dynamical structure function* (DSF), which are very similar to the DNF but with one subtle distinction. A DSF (Q, P) defining the relationship $y = Qy + Pu$ is derived from a signal structure where $\bar{W}$ is required to be hollow (meaning $[\bar{W}]_{ii} = 0$ for all i), which implies

[2]For simplicity, we will assume that these operators are continuous-time systems in the frequency domain, meaning that each operator is a rational function in the Laplace variable $s \in \mathbb{C}$. Causality implies that these rational functions are proper. The results in this work also generalize naturally to discrete-time and time-domain operators as well.

[3]Some works (e.g., [1, 28]) further partition inputs based on whether they are controlled, uncontrolled, deterministic, stochastic, etc. We make no such distinction in this work, although we do remind the reader that these signals are all *manifest*, not fictitious (e.g., white noise) as part of a particular modeling framework.

that Q is also hollow. Hollowness implies that there are no self-loops from signals w_i to themselves, or, equivalently outputs y_i to themselves. In this work, we introduce the DNF as a generalization of the DSF by not requiring either $\bar{W}$ or W to be hollow, meaning that self-loops are allowed but not required. In consequence, this set of results apply to all linear dynamical networks, including DNFs and DSFs.

By assuming that $(I - W)$ is invertible and solving for y, we can re-arrange the DNF such that

$$y = \left[(I - W)^{-1}V\right] u \triangleq G_{cl}u, \tag{2.2}$$

where G_{cl} is the closed-loop transfer function characterizing the input-output behavior of this system. Thus we see that the invertibility of $(I - W)$ is important to the characterization of this DNF and its behavior, and, as we will show in this work, the invertibility of $(I - W)$ is also intimately tied to the well-posedness of the DNF.

REMARK 2.3.2: Note that we can find an equivalent behavior by considering the constraint $M\bar{W} = 0$, where M has a proper inverse. It can be shown that all equivalent behaviors result in the same closed-loop transfer function G_{cl}, but will each have a different DNF (W, V). The problem of finding a *unique* DNF from either the behavior or the closed-loop transfer function is known as *network reconstruction*, and since (W, V) is not unique given the behavior, additional a priori knowledge about the network, known as the *identifiability conditions*, is needed to solve this problem. We do not discuss network reconstruction or identifiability conditions in this work; however, treatments of these ideas can be found in [19, 28].

2.4 Structured Representations of LTI Systems

We now show the conversion of a state space model to the DNF (a slight variation of the conversion process from a state space model to a DNF contained in [1]). We then show that we cannot convert any state space model into a DNF with entries in W that are proper and

not necessarily strictly proper. DNFs with proper (and not necessarily strictly proper) entries in W are essential for several applications (see, for instance, [8, 29]) and are the only networks where well-posedness is a concern (see Corollary 2.5.9). Thus, we present the generalized state space (GSS) model (first introduced in [30, 31]) as a representation of multiple state space models connected in feedback and demonstrate the transformation from a GSS model into a DNF. We then show that for any arbitrary proper DNF (W, V), there exists a GSS that can be converted to that DNF, meaning that the set of GSSs is rich enough to define the entire space of proper DNFs where the set of state space models is not.

2.4.1 The State Space Model and the DNF

We begin by demonstrating the conversion from a state space model to a DNF. Suppose that we are given an arbitrary state space model $(\bar{A}, \bar{B}, \bar{C}, \bar{D})$ such that

$$\begin{aligned} \dot{z} &= \bar{A}z + \bar{B}u, \\ y &= \bar{C}z + \bar{D}u, \end{aligned} \tag{2.3}$$

where $\bar{A} \in \mathbb{R}^{n\times n}$, $\bar{B} \in \mathbb{R}^{n\times m}$, $\bar{C} \in \mathbb{R}^{p\times n}$, and $\bar{D} \in \mathbb{R}^{p\times m}$. Let C be full row rank[4] (i.e., $\operatorname{rank} C = p$).

Perform a change of basis over the states in (2.3) such that $x = Tz$ such that

$$T = \begin{bmatrix} \bar{C}' & E_1 \end{bmatrix}' \tag{2.4}$$

and $E_1 \in \mathbb{R}^{n\times(n-p)}$ is any basis of the null space of $\bar{C}$. Note that

$$T^{-1} = \begin{bmatrix} \bar{C}'(\bar{C}\bar{C}')^{-1} & E_1 \end{bmatrix} \triangleq \begin{bmatrix} R_1 & E_1 \end{bmatrix}. \tag{2.5}$$

[4]The work [1] contains the procedure for dealing with a C that is not full row rank; however, any additional insight by considering this case is not insightful for the purposes of this work; therefore, we will not treat it here.

With this definition, we have that

$$\bar{C}T^{-1} = \begin{bmatrix} (\bar{C}\bar{C}')(\bar{C}\bar{C}')^{-1} & CE_1 \end{bmatrix} = \begin{bmatrix} I_p & 0 \end{bmatrix}, \tag{2.6}$$

thus the change of basis will result in a new state space model (A, B, C, D) such that $A = T\bar{A}T^{-1}$, $B = T\bar{B}$, $C = \bar{C}T^{-1} = \begin{bmatrix} I_p & 0 \end{bmatrix}$, and $D = \bar{D}$.

Now, partition (A, B, C, D) commensurate with $C = \begin{bmatrix} I_p & 0 \end{bmatrix}$ to get

$$\begin{aligned} \begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} &= \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} B_1 \\ B_2 \end{bmatrix} u, \\ y &= \begin{bmatrix} I & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + Du. \end{aligned} \tag{2.7}$$

Assuming zero initial conditions, take the Laplace transform of (2.7) to get

$$\begin{aligned} \begin{bmatrix} sX_1 \\ sX_2 \end{bmatrix} &= \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \end{bmatrix} + \begin{bmatrix} B_1 \\ B_2 \end{bmatrix} U, \\ Y &= \begin{bmatrix} I & 0 \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \end{bmatrix} + DU. \end{aligned} \tag{2.8}$$

Solve for X_2 in the bottom row of the first equation in (2.8) to yield

$$X_2 = (sI - A_{22})^{-1}A_{21}X_1 + (sI - A_{22})^{-1}B_2U. \tag{2.9}$$

Plug (2.9) into the top row of (2.8) and rearrange to get

$$sX_1 = \left[A_{11} + A_{12}(sI - A_{22})^{-1}A_{21}\right] X_1 + \left[B_1 + A_{12}(sI - A_{22})^{-1}B_2\right] U \tag{2.10}$$

$$\triangleq \tilde{W}X_1 + \tilde{V}U. \tag{2.11}$$

The process, thus far, has been consistent with that outlined in [1], though in that work, they labeled $\tilde{W}$ as W and $\tilde{V}$ as V. We added the tilde in this work to distinguish the intermediate step in (2.11) and the final DNF (W, V). At this point, however, we diverge from the previous works on DSFs. This results in

$$X_1 = \frac{1}{s}\tilde{W}X_1 + \frac{1}{s}\tilde{V}U \triangleq WX_1 + \hat{V}U. \tag{2.12}$$

We return to the procedure outlined in [1] while acknowledging that W here (which corresponds to Q in the DSF) is not hollow. Assuming that $(I - W)$ has an inverse (we will show, in Section 2.5, that $(I - W)^{-1}$ is guaranteed to both exist and be proper since W is strictly proper), solve for $X_1 = (I - W)^{-1}\hat{V}U$ and plug this in to the equation for Y in (2.8), resulting in $Y = (I - W)^{-1}\hat{V}U + DU$. Rearrange to get

$$Y = WY + \left[\hat{V} + (I - W)D\right]U \triangleq WY + VU, \tag{2.13}$$

which forms our final DNF (W, V), where $W \in RP^{p\times p}$ and $V \in RP^{p\times m}$.

Definition 2.4.1: Consistency

Consider a state space model (A, B, C, D) where C is full row rank. If the DNF (W, V) is the result of applying the above process to (A, B, C, D), then we say that (W, V) is *consistent* with (A, B, C, D).

Notice that the choice of T in (2.4) is non-unique. We show, however, that (W, V) is consistent with the transformed state space model found from *every* choice of T that satisfies (2.4).

Theorem 2.4.2 (Invariance to Bases of the Null Space): *Given a state space model $(\bar{A}, \bar{B}, \bar{C}, \bar{D})$ as in (2.3), consider two distinct bases of the null space of $\bar{C}$, E_1 and E_2 with $E_1 \neq E_2$, with corresponding state transformations T_1 and T_2 each given by (2.4). Let (W_1, V_1)*

be the DNF resulting from the transformed system $(A_1, B_1, C_1, D_1) = (T_1\bar{A}T_1^{-1}, T_1\bar{B}, \bar{C}T_1^{-1}, \bar{D})$ *and let* (W_2, V_2) *bet the DNF resulting from the transformed system* $(A_2, B_2, C_2, D_2) = (T_2\bar{A}T_2^{-1}, T_2\bar{B}, \bar{C}T_2^{-1}, \bar{D})$. *Then* $(W_1, V_1) = (W_2, V_2)$. ♢

PROOF We defer the reader to [1], which shows that $\tilde{W}$ and $\tilde{V}$ are invariant to a change of basis on the null space of $\bar{C}$; thus W and V will also be invariant. ■

We have now shown that for any state space model there exists a DNF consistent with that model. However, the question arises about whether all DNFs (W, V) of interest (i.e., all (W, V) where W and V are proper matrices of rational polynomials of the appropriate dimensions). It turns out that the answer is no, that the procedure above can never generate a W that is proper but not strictly proper. We show this in Theorem 2.4.3 below.

Theorem 2.4.3: *Let* (W, V) *be a DNF where* $W \in RP^{p\times p}$ *and* $V \in RP^{p\times m}$, *and where* W *is proper but not strictly proper. Then* (W, V) *is not consistent with any state space model.*♢

PROOF It is sufficient to show that W is always strictly proper when generated from a state space model through the process above. We have that $\tilde{W} = A_{12}(sI - A_{22})^{-1}A_{21} + A_{11}$, thus $\tilde{W} \in RP^{p\times p}$ can be viewed as the transfer function representation of the state space model $(A_{22}, A_{21}, A_{12}, A_{11})$, and is therefore proper (see Lemma 2.7.5). However, $\frac{1}{s}$ is strictly proper. Thus, by Lemma 2.7.4, $W = \frac{1}{s}\tilde{W}$ must be strictly proper. ■

Corollary 2.4.4: *Let* (W, V) *be a DNF where* $V \in RP^{p\times m}$ *and* $W \in RP^{p\times p}$ *and* W *is strictly proper. Then* V *is proper but not strictly proper (a) if and (b) only if* $D \neq 0$ *for any state space model consistent with* (W, V). ♢

PROOF We have that $\tilde{V} = A_{12}(sI - A_{22})^{-1}B_2 + B_1$, thus $V \in RP^{p\times m}$ can be viewed as the transfer function representation of the state space model $(A_{22}, A_{21}, B_2, B_1)$, and is therefore proper (see Lemma 2.7.5). Thus, by Lemma 2.7.4 and since $\frac{1}{s}$ is strictly proper, $\hat{V} = \frac{1}{s}\tilde{V}$ is strictly proper. Also, by Theorem 2.4.3, W is likewise strictly proper.

(a) Suppose that $D \neq 0$. Since W is strictly proper, no diagonal entry of W can equal 1; thus $(I - W)$ must be proper but not strictly proper along the diagonal entries and strictly proper everywhere else. Thus the product of $(I - W)$ with D cannot cancel any of the proper but not strictly proper elements of $(I - W)$, leaving $(I - W)D$ proper but not strictly proper as well. Moreover, since $\hat{V}$ is strictly proper, it cannot cancel any of the proper but not strictly proper elements of $(I - W)D$; thus the sum $V = \hat{V} + (I - W)D$ must be proper but not strictly proper.

(b) Suppose that $D = 0$. Then $V = \hat{V}$ and is strictly proper thus concluding our proof. ■

Theorem 2.4.3 may be somewhat surprising, especially in light of Lemma 2.7.6 which shows that we can generate any proper transfer function from a state space model. However, the result is sensible considering how properness arises in the DNF. Notice that the transfer function $G = C(sI - A)^{-1}B + D$ is proper if and only if $D \neq 0$, and this because D is precisely the static term mapping u to y. This static term is what adds the properness in G, and it also, by Corollary 2.4.4, creates the properness in P. However, there are no static terms mapping y to y or even x to x in the state space model; therefore, it is impossible to create any static maps in W which maps y to y (and, by extension, x to x).

One may ask, therefore, whether we should even allow terms in W that are not strictly proper. However, as we show in Example 2.4.5 below, a W that is proper but not strictly proper can arise very naturally when we connect two DNFs in feedback; therefore, we do wish to build a theoretical foundation allowing for such properness, which is the purpose of the rest of this work.

Example 2.4.5

Consider two DNFs defining the relations $Y_1 = W_1Y_1 + V_1E_1$ and $Y_2 = W_2Y_2 + V_2E_2$. Suppose that both DNFs are generated from state space models, meaning that W_1 and W_2 are strictly proper, though V_1 and V_2 may be proper and not strictly proper. Suppose

that we connect these DNFs in feedback by letting $E_1 = Y_2 + U_1$ and $E_2 = Y_1 + U_2$ for some external signals U_1 and U_2. Then the closed-loop DNF takes the form

$$Y \triangleq \begin{bmatrix} Y_1 \\ Y_2 \end{bmatrix} = \begin{bmatrix} W_1 & V_1 \\ V_2 & W_2 \end{bmatrix} \begin{bmatrix} Y_1 \\ Y_2 \end{bmatrix} + \begin{bmatrix} V_1 & 0 \\ 0 & V_2 \end{bmatrix} \begin{bmatrix} U_1 \\ U_2 \end{bmatrix}$$
$$\triangleq WY + VU, \tag{2.14}$$

which is also a DNF. We have that, since V_1 and V_2 are proper (and not necessarily strictly proper), then W is likewise proper.

The insight that the example above gives is that, by connecting two systems in feedback, we turn inputs u into outputs y. As explained above, static terms in state space models give us static terms in the maping from u to y; thus, since u's are now y's, we can now have static terms in the mapping from y to y allowing W to be proper and not necessarily strictly proper.

2.4.2 The Generalized State Space Model and the DNF

As shown in the previous section, state space models are not rich enough to generate DNFs with W that are proper and not strictly proper. However, we also showed that proper (and not necessarily strictly proper) entries in W arise naturally when interconnecting DNFs in feedback. Thus, if we had a state space representation that preserves information about the feedback interconnection of subsystems (the state space model, unfortunately, does not), we may expect that this representation could generate matrices W that are proper and not necessarily strictly proper. Fortunately, representations do exist that preserve information about the feedback interconnection. Such a representation was introduced in [30, 31], and is called *generalized state space* (GSS) model or the *complete computational structure*. This

representation is characterized by $(A, \hat{A}, B, \bar{A}, \tilde{A}, \bar{B}, C, \bar{C}, D)$ such that:

$$\begin{aligned} \dot{x} &= Ax + \hat{A}w + Bu, \\ w &= \bar{A}x + \tilde{A}w + \bar{B}u, \\ y &= Cx + \bar{C}w + Du. \end{aligned} \tag{2.15}$$

Note that, assuming $(I - \tilde{A})$ is non-singular, the GSS model can be uniquely converted to the state space model by solving for $w = (I - \tilde{A})^{-1}\bar{A}x + (I - \tilde{A})^{-1}\bar{B}u$, and plugging this into the equations for $\dot{x}$ and y. However, the transformation of a GSS model into a state space model will result in the loss of the information about the feedback interconnection within the system.

The auxiliary variables, w, are used to characterize the intermediate computation in the interconnection of subsystems and are thus capable of characterizing the interconnection. We show this in the following examples:

Example 2.4.6: Feedback Interconnection as a GSS

We now illustrate that state space models connected in feedback can be represented as a generalized state space model. Consider two state space models of the form

$$\begin{aligned} \begin{bmatrix} \dot{x}_{i,1} \\ \dot{x}_{i,2} \end{bmatrix} &= \begin{bmatrix} A_{i,11} & A_{i,12} \\ A_{i,21} & A_{i,22} \end{bmatrix} \begin{bmatrix} x_{i,1} \\ x_{i,2} \end{bmatrix} + \begin{bmatrix} B_{i,1} \\ B_{i,2} \end{bmatrix} e_i, \\ y_i &= \begin{bmatrix} I & 0 \end{bmatrix} \begin{bmatrix} x_{i,1} \\ x_{i,2} \end{bmatrix} + D_i e_i. \end{aligned} \tag{2.16}$$

Stack these state equations into a generalized state space model while keeping track of the intricacy interconnections by letting $w_i = y_i$ and $e_i = u_i + y_j$ for $j \neq i$. This gives

$$\begin{bmatrix} \dot{x}_{1,1} \\ \dot{x}_{1,2} \\ \dot{x}_{2,1} \\ \dot{x}_{2,2} \end{bmatrix} = \begin{bmatrix} A_{1,11} & A_{1,12} & 0 & 0 \\ A_{1,21} & A_{1,22} & 0 & 0 \\ 0 & 0 & A_{2,11} & A_{2,12} \\ 0 & 0 & A_{2,21} & A_{2,22} \end{bmatrix} \begin{bmatrix} x_{1,1} \\ x_{1,2} \\ x_{2,1} \\ x_{2,2} \end{bmatrix} + \begin{bmatrix} 0 & B_{1,1} \\ 0 & B_{1,2} \\ B_{2,1} & 0 \\ B_{2,2} & 0 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} + \begin{bmatrix} B_{1,1} & 0 \\ B_{1,2} & 0 \\ 0 & B_{2,1} \\ 0 & B_{2,2} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix},$$

$$\begin{bmatrix} w_1 \\ w_2 \end{bmatrix} = \begin{bmatrix} I & 0 & 0 & 0 \\ 0 & 0 & I & 0 \end{bmatrix} \begin{bmatrix} x_{1,1} \\ x_{1,2} \\ x_{2,1} \\ x_{2,2} \end{bmatrix} + \begin{bmatrix} 0 & D_1 \\ D_2 & 0 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} + \begin{bmatrix} D_1 & 0 \\ 0 & D_2 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix},$$

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} I & 0 \\ 0 & I \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \end{bmatrix}.$$

Note that the resultant model is a generalized state space model of the form (2.15).

Example 2.4.7: Large-Scale Interconnection as a GSS

We now generalize Example 2.4.6 to represent the interconnection of an arbitrary number N of state-space models. For $i = 1, \ldots, N$, let

$$\begin{aligned} \dot{x}_i &= Ax_i + Be_i, \\ y_i &= Cx_i + De_i, \end{aligned} \tag{2.17}$$

with $A \in \mathbb{R}^{n_i \times n_i}$, $B \in \mathbb{R}^{n_i \times m_i}$, $C \in \mathbb{R}^{p_i \times n_i}$, and $D \in \mathbb{R}^{p_i \times m_i}$.

Stack these equations into a generalized state space model while keeping track of the intricacy interconnections by (for $i = 1, \ldots, N$) letting $w_i = y_i$ and, in the manner of [25],

$$e_i = u_i - \sum_{j=1}^{n} H_{ij} y_j, \tag{2.18}$$

where $H \in \mathbb{R}^{p_i \times p_j}$ (note that we allow $j = i$). Let $x = \begin{bmatrix} x_1' & x_2' & \cdots & x_N' \end{bmatrix}'$ and $w = \begin{bmatrix} w_1' & w_2' & \cdots & w_N' \end{bmatrix}'$. Then the interconnected state space model takes the form (2.15) with $A = \operatorname{diag}(A_1, A_2, \cdots, A_N)$, $B = \operatorname{diag}(B_1, B_2, \cdots, B_N)$, $\hat{A} = -B \cdot [H_{ij}]$, $\bar{A} = \operatorname{diag}(C_1, C_2, \cdots, C_N)$, $\bar{B} = \operatorname{diag}(D_1, D_2, \cdots, D_N)$, $\tilde{A} = -\bar{B} \cdot [H_{ij}]$, $C = 0$, $\bar{C} = I$, and $D = 0$.

Example 2.4.7 and Example 2.4.6 highlight a few important ideas about the generalized state space model that is generated from connecting state space models in feedback. First, if we connect individual state space models such that the outputs of the individual models are connected to the inputs of the other models, then the intricacy variables are defined as precisely the union of all outputs of the individual models. Furthermore, by construction, if we choose the output of the interconnected system to be the union of the outputs of the individual models, then we get that the set of outputs of the interconnected system is equivalent to the set of intricacy variables. More briefly, $y = w$ by construction.

Furthermore, as described in the previous section, this paper is concerned only with state space models where C is full row rank. In the interconnected model, we have $\bar{A} = \operatorname{diag}(C_1, C_2, \cdots, C_N)$, thus $\operatorname{rank} \bar{A} = \sum_{i=1}^{N} \operatorname{rank} C_i$, meaning that $\bar{A}$ is also full row rank.

For this paper, we will only consider generalized state space models that have $y = w$ and $\bar{A}$ full row rank, such as those found interconnecting state space models. We call such models *Intricacy-Observed*, which we define formally below.

Definition 2.4.8: Intricacy-Observed GSS

We say that a generalized state space model is *intricacy observed* if it is in the form

$$\begin{aligned} \dot{x} &= Ax + \hat{A}w + Bu, \\ w &= \bar{A}x + \tilde{A}w + \bar{B}u, \\ y &= w, \end{aligned} \tag{2.19}$$

where $\bar{A}$ is full row rank.

We now proceed with the process of converting an Intricacy-Observed GSS into a DNF. Since $\bar{A}$ is full row rank, there exists a change of basis T over w such that $\bar{A}$ is in $\begin{bmatrix} I & 0 \end{bmatrix}$ form (it is the same T that converted C into $\begin{bmatrix} I & 0 \end{bmatrix}$ form in the previous section). Transform (2.19) by T. For simplicity, we overload notation and say that the resultant transformed system (after partitioning the remaining matrices commensurate with $\bar{A}$ and taking advantage of the identity $y = w$) is given as follows:

$$\begin{aligned} \begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} &= \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} \hat{A}_1 \\ \hat{A}_2 \end{bmatrix} y + \begin{bmatrix} B_1 \\ B_2 \end{bmatrix} u, \\ y &= \begin{bmatrix} I & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \tilde{A}y + \bar{B}u. \end{aligned} \tag{2.20}$$

We proceed in much the same manner as in Section 2.4.1. Take the Laplace transform of (2.20) to get

$$\begin{aligned} \begin{bmatrix} sX_1 \\ sX_2 \end{bmatrix} &= \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \end{bmatrix} + \begin{bmatrix} \hat{A}_1 \\ \hat{A}_2 \end{bmatrix} Y + \begin{bmatrix} B_1 \\ B_2 \end{bmatrix} U, \\ Y &= \begin{bmatrix} I & 0 \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \end{bmatrix} + \tilde{A}Y + \bar{B}U. \end{aligned} \tag{2.21}$$

Solve for X_2 in the bottom row of the first equation in (2.21) to get

$$\begin{aligned} X_2 = (sI - A_{22})^{-1} A_{21} X_1 + (sI - A_{22})^{-1} \hat{A}_2 Y \\ + (sI - A_{22})^{-1} B_2 U. \end{aligned} \tag{2.22}$$

Plug (2.22) back in to the top row of the first equation in (2.21) and rearrange to get

$$sX_1 = \left[A_{11} + A_{12}(sI - A_{22})^{-1} A_{21} \right] X_1$$

$$+ \left[\hat{A}_1 + A_{12}(sI - A_{22})^{-1}\hat{A}_2\right] Y$$

$$+ \left[B_1 + A_{12}(sI - A_{22})^{-1}B_2\right] U \tag{2.23}$$

$$\triangleq \tilde{W}X_1 + \tilde{R}Y + \tilde{V}U. \tag{2.24}$$

Divide both sides of (2.24) by s to get

$$X_1 = \frac{1}{s}\tilde{W}X_1 + \frac{1}{s}\tilde{R}Y + \frac{1}{s}\tilde{V}U \tag{2.25}$$

$$\triangleq \hat{W}X_1 + \hat{R}Y + \hat{V}U. \tag{2.26}$$

Since $(I - \hat{W})$ has an inverse (this is always guaranteed for the same reasons it was guaranteed in Section 2.4.1), we can solve for X_1 to get

$$X_1 = (I - \hat{W})^{-1}\hat{R}Y + (I - \hat{W})^{-1}\hat{V}U. \tag{2.27}$$

Plug (2.27) into the equation for Y in (2.21) to get

$$Y = (I - \hat{W})^{-1}\hat{R}Y + (I - \hat{W})^{-1}\hat{V}U + \tilde{A}Y + \bar{B}U. \tag{2.28}$$

Finally, rearrange to get

$$Y = \left[\hat{W} + \hat{R} + (I - \hat{W})\tilde{A}\right] Y + \left[\hat{V} + (I - \hat{W})\bar{B}\right] U \tag{2.29}$$

$$\triangleq WY + VU, \tag{2.30}$$

which forms our final DNF.

With this definition, we are prepared to show that the information contained in $\tilde{A}$ and $\bar{B}$ is preserved in the DNF as well, an idea which is important later in this work. We formalize and show this idea in Lemma 2.4.9 below.

Lemma 2.4.9: *Let $(A, \hat{A}, B, \bar{A}, \tilde{A}, \bar{B})$ parameterize some generalized state space model of the form (2.19) and suppose that $(W(s), V(s))$ is the DNF consistent with this model. Then $W(\infty) \triangleq \lim_{s\to\infty} W(s) = \tilde{A}$ and $V(\infty) \triangleq \lim_{s\to\infty} V(s) = \bar{B}$.* ♢

PROOF We have, by (2.30), that $W(s) = \hat{W} + \hat{R} + (I - \hat{W})\tilde{A}$, where $\hat{W}$ and $\hat{R}$ are both strictly proper (see the proof Theorem 2.4.3 for the justification of strict properness). Then, by Lemma 2.7.10, Lemma 2.7.11, and Lemma 2.7.9:

$$\begin{aligned} W(\infty) &= \lim_{s\to\infty} \left[\hat{W} + \hat{R} + (I - \hat{W})\tilde{A}\right] \\ &= \left[\lim_{s\to\infty} \hat{W}\right] + \left[\lim_{s\to\infty} \hat{R}\right] + \left(I - \left[\lim_{s\to\infty} \hat{W}\right] \tilde{A}\right) \\ &= 0 + 0 + (I - 0)\tilde{A} = \tilde{A}. \end{aligned}$$

Similarly, since $\hat{V}$ is strictly proper

$$\begin{aligned} V(\infty) &= \lim_{s\to\infty} \left[\hat{V} + (I - \hat{W})\bar{B}\right] \\ &= \left[\lim_{s\to\infty} \hat{V}\right] + \left(I - \left[\lim_{s\to\infty} \hat{W}\right]\right) \bar{B} \\ &= 0 + (I - 0)\bar{B} = \bar{B}, \end{aligned}$$

as desired. ■

Our purpose for introducing the set of intricacy-observed GSS's was to provide an equivalent set to the set of all possible (proper) DNFs. We have shown above that every generalized state space model of the form (2.19) with $\tilde{A} \neq 0$ results in a proper DNF. We now need to show that every proper DNF has a realization of the form (2.19). We do this in Theorem 2.4.10 below.

Theorem 2.4.10: *Let (W, V) be an arbitrary well-posed*[5] *DNF where $W \in RP^{p\times p}$ and $V \in RP^{p\times m}$. Then a generalized state space model of the form (2.19) exists such that (W, V) is consistent with that model.* ♢

PROOF We show this by construction. By Lemma 2.4.9, we have that $\tilde{A} = W(\infty)$ and $\bar{B} = V(\infty)$. Choose any strictly proper $\hat{R}$ of the appropriate dimensions (e.g. $\hat{R} = (W - \tilde{A})/2$ or $\hat{R} = 0$). We have, by (2.30), that

$$W = \hat{W} + \hat{R} + (I - \hat{W})\tilde{A} = \hat{W}(I - \tilde{A}) + \hat{R} + \tilde{A},$$

thus

$$\hat{W} = ((W - \tilde{A}) - \hat{R})(I - \tilde{A})^{-1}, \tag{2.31}$$

where the assumption of well-posedness guarantees that $(I - \tilde{A})$ is non-singular (see Theorem 2.5.8 in Section 2.5). Note that, since $\tilde{A} = W(\infty)$, $(W - \tilde{A})$ will be strictly proper. Hence, by Lemma 2.7.3, $\hat{W}$ is also strictly proper.

We also have from (2.30) that

$$\hat{V} = V - (I - \hat{W})\bar{B}. \tag{2.32}$$

Note that $\lim_{s\to\infty} \hat{V} = \lim_{s\to\infty}(V - \bar{B}) - \hat{W}\bar{B} = 0$ since $V - \bar{B} = V - V(\infty)$ and $\hat{W}\bar{B}$ are strictly proper, thus $\hat{V}$ must also be strictly proper by Lemma 2.7.9.

Define $\tilde{W} = s\hat{W}$, $\tilde{R} = s\hat{R}$, and $\tilde{V} = s\hat{V}$. Since $\hat{W}$, $\hat{R}$, and $\hat{V}$ are all strictly proper and the product of s with these can only increase the degree of the numerator by one, we have that $\tilde{W}$, $\tilde{R}$, and $\tilde{V}$ will all be proper.

Consider the augmented matrix $\Phi = \begin{bmatrix}\tilde{W} & \tilde{R} & \tilde{V}\end{bmatrix}$. Since Φ is a proper matrix of rational polynomials, we can treat it as a transfer function with a state realization $(A_\Phi, B_\Phi, C_\Phi, D_\Phi)$ such that $\Phi = C_\Phi(sI - A_\Phi)^{-1}B_\Phi + D_\Phi$ (Lemma 2.7.6). Partition $(A_\Phi, B_\Phi, C_\Phi, D_\Phi)$ commen-

[5] See Section 2.5.

surate with Φ and label the partitions as follows:

$$\begin{aligned} A_\Phi &\triangleq A_{22}, & B_\Phi &\triangleq \begin{bmatrix} A_{21} & \hat{A}_2 & B_2 \end{bmatrix}, \\ C_\Phi &\triangleq A_{12}, & D_\Phi &\triangleq \begin{bmatrix} A_{11} & \hat{A}_1 & B_1 \end{bmatrix}. \end{aligned}$$

Construct the following generalized state space model:

$$\begin{aligned} \begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} &= \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} \hat{A}_1 \\ \hat{A}_2 \end{bmatrix} w + \begin{bmatrix} B_1 \\ B_2 \end{bmatrix} u, \\ w &= \begin{bmatrix} I & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \tilde{A}w + \bar{B}u, \\ y &= w. \end{aligned}$$

This model is in the form (2.19), and by construction, it will yield (Q, P) when the process above is followed. ■

Example 2.4.11

We return to Example 2.4.5, where we connected two DNFs in feedback. Let each DNF (Q_i, P_i), $i \in \{1, 2\}$, have the following state realization:

$$\begin{aligned} \begin{bmatrix} \dot{x}_{i,1} \\ \dot{x}_{i,2} \end{bmatrix} &= \begin{bmatrix} A_{i,11} & A_{i,12} \\ A_{i,21} & A_{i,22} \end{bmatrix} \begin{bmatrix} x_{i,1} \\ x_{i,2} \end{bmatrix} + \begin{bmatrix} B_{i,1} \\ B_{i,2} \end{bmatrix} e_i, \\ w_i &= \begin{bmatrix} I & 0 \end{bmatrix} \begin{bmatrix} x_{i,1} \\ x_{i,2} \end{bmatrix} + D_i e_i. \end{aligned} \tag{2.33}$$

Using the process for finding a DNF outlined in Section 2.4.1, we get the following equations for each system i:

$$\begin{aligned}
\tilde{W}_i &= A_{i,11} + A_{i,12}(sI - A_{i,22})^{-1}A_{i,21} \\
\tilde{V}_i &= B_{i,1} + A_{i,12}(sI - A_{i,22})^{-1}B_{i,2} \\
W_i &= \frac{1}{s}\tilde{W}_i \\
\hat{V}_i &= \frac{1}{s}\tilde{V}_i \\
V_i &= \hat{V}_i + (I - W_i)D_i
\end{aligned}$$

Stack the state equations into a generalized state space while keeping track of the intricacy interconnections by letting $y = w$ and $e_i = u_i + w_j$ for $j \neq i$. The resultant generalized state space model is given in Example 2.4.6. Convert this generalized state space model to $\bar{A} = \begin{bmatrix} I & 0 \end{bmatrix}$ form by creating a transformation T that permutes $x_{1,2}$ with $x_{2,1}$. This gives, in the notation for the intricacy-observed GSS:

$$\begin{aligned}
A_{11} &= \begin{bmatrix} A_{1,11} & 0 \\ 0 & A_{2,11} \end{bmatrix}, & A_{12} &= \begin{bmatrix} A_{1,12} & 0 \\ 0 & A_{2,12} \end{bmatrix}, \\
A_{21} &= \begin{bmatrix} A_{1,21} & 0 \\ 0 & A_{2,21} \end{bmatrix}, & A_{22} &= \begin{bmatrix} A_{1,22} & 0 \\ 0 & A_{2,22} \end{bmatrix}, \\
\hat{A}_1 &= \begin{bmatrix} 0 & B_{1,1} \\ B_{2,1} & 0 \end{bmatrix}, & \hat{A}_2 &= \begin{bmatrix} 0 & B_{1,2} \\ B_{2,2} & 0 \end{bmatrix}, \\
B_1 &= \begin{bmatrix} B_{1,1} & 0 \\ 0 & B_{2,1} \end{bmatrix}, & B_2 &= \begin{bmatrix} B_{1,2} & 0 \\ 0 & B_{2,2} \end{bmatrix}, \\
\tilde{A} &= \begin{bmatrix} 0 & D_1 \\ D_2 & 0 \end{bmatrix}, & \bar{B} &= \begin{bmatrix} D_1 & 0 \\ 0 & D_2 \end{bmatrix}.
\end{aligned}$$

Solve for $\tilde{W}$, $\tilde{R}$, and $\tilde{V}$ in (2.24) to give:

$$\tilde{W} = \begin{bmatrix} \tilde{W}_1 & 0 \\ 0 & \tilde{W}_2 \end{bmatrix}, \quad \tilde{V} = \begin{bmatrix} \tilde{V}_1 & 0 \\ 0 & \tilde{V}_2 \end{bmatrix}, \quad \tilde{R} = \begin{bmatrix} 0 & \tilde{V}_1 \\ \tilde{V}_2 & 0 \end{bmatrix},$$

thus,

$$\hat{W} = \begin{bmatrix} W_1 & 0 \\ 0 & W_2 \end{bmatrix}, \quad \hat{V} = \begin{bmatrix} \hat{V}_1 & 0 \\ 0 & \hat{V}_2 \end{bmatrix}, \quad \hat{R} = \begin{bmatrix} 0 & \hat{V}_1 \\ \hat{V}_2 & 0 \end{bmatrix}.$$

We have that

$$W = \hat{W} + \hat{R} + (I - \hat{W})\tilde{A} = \begin{bmatrix} W_1 & \hat{V}_1 + (I - W_1)D_1 \\ \tilde{V}_2 + (I - W_2)D_2 & W_2 \end{bmatrix} = \begin{bmatrix} W_1 & V_1 \\ V_1 & W_1 \end{bmatrix}.$$

Similarly, we have that

$$\begin{aligned} V &= \hat{V} + (I - \hat{W})\bar{B} \\ &= \begin{bmatrix} \hat{V}_1 + (I - \hat{W}_1)\bar{B}_1 & 0 \\ 0 & \hat{V}_2 + (I - \hat{W}_2)\bar{B}_2 \end{bmatrix} = \begin{bmatrix} V_1 & 0 \\ 0 & V_2 \end{bmatrix}. \end{aligned}$$

Thus the final DNF defines the relationship

$$\begin{aligned} Y \triangleq \begin{bmatrix} Y_1 \\ Y_2 \end{bmatrix} &= \begin{bmatrix} W_1 & V_1 \\ V_2 & W_2 \end{bmatrix} \begin{bmatrix} Y_1 \\ Y_2 \end{bmatrix} + \begin{bmatrix} V_1 & 0 \\ 0 & V_2 \end{bmatrix} \begin{bmatrix} U_1 \\ U_2 \end{bmatrix} \\ &\triangleq WY + VU. \end{aligned} \tag{2.34}$$

Notice that Equation (2.34) is the same as Equation (2.14), meaning that the process of converting the generalized state space representing two systems in feedback results in the

same network as that created by connecting the DNFs corresponding to the two systems in feedback.

2.5 Well-Posedness of LTI Network Systems

We now turn our attention to a definition of well-posedness of DNFs and generalized state space models.

2.5.1 Well-Posedness of Large-Scale Interconnected Systems

One of the earliest treatments of studying the well-posedness of networks containing the interconnection of more than two systems is found in [25]. This work is concerned with networks of the form

$$\begin{aligned} e_i &= u_i - \textstyle\sum_{j=1}^{m} H_{ij} y_j, \\ y_i &= G_i e_i. \end{aligned} \tag{2.35}$$

Note that [25] defines and then provides sufficient (but not necessary) conditions for the well-posedness of such networks when H_{ij} and G_i are arbitrary (non-linear) operators. However, to find necessary and sufficient conditions, it then restricts its attention to the case where G_i and H_{ij} are LTI operators (which is also the focus of our work). In this case, [25] gives the following definition for well-poosedness:

Definition 2.5.1: Well-Posedness (Vidyasagar)

The network system (2.35), with every G_i and H_{ij} as an LTI operator, is said to be well-posed if the following conditions hold: l

(V1) For each set of input signals u, there exists a unique set of error signals e and a set of output signals y such that the equations (2.35) are satisfied.

(V2) The dependence of e and y on u is causal.

(V3) For each finite T, the dependence of $P_T e$ and $P_T y$ on $P_T u$ is Lipschitz continuous, where, for any signal x, $P_T x$ is the truncation of x to the interval $[0, T]$.

Since each operator H_{ij} and G_i are LTI, we can write (2.35) as matrix multiplications as follows:

$$\begin{aligned} e &= u - Hy \\ y &= Ge, \end{aligned} \tag{2.36}$$

where $G \in RP^{p\times p}$ is block diagonal[6] and $H \in RP^{p\times p}$ is arbitrary.

The necessary and sufficient conditions, shown in [25], for (2.36) to be well-posed according to Definition 2.5.1 are that

$$\det\left(I + H(\infty)G(\infty)\right) \neq 0. \tag{2.37}$$

In other words, in light of Lemma 2.7.7, (2.36) is well-posed if and only if $(I - HG)$ has a proper inverse.

The conversion of (2.36) to a DNF depends on whether e or y is manifest. If e is manifest, we can plug the bottom row in to the top to get

$$e = u - HGe, \tag{2.38}$$

thus $W = -HG$ and $V = I$. If y is manifest instead, we can plug the top row into the bottom to get

$$y = Gu - GHy, \tag{2.39}$$

thus $W = -GH$ and $V = G$. Since $\det(I + G(\infty)H(\infty)) = \det(I + H(\infty)G(\infty))$ (see Lemma 2.7.12), the conditions of well-posedness for both (2.38) and (2.39) are that $(I - W(\infty))$ has an inverse, or, equivalently, that $(I - W)$ has a proper inverse (Lemma 2.7.7).

[6]Since each G_i is MIMO and arbitrary, arbitrary (full) G can be found by considering an interconnection of exactly one system.

We remark, however, that in both (2.38) and (2.39), the DNF formed takes a very special form. In the first, $V = I$, and in the second, V is square, block diagonal, and a left factor of W. We wish to generalize these conditions to the case where $V \in RP^{p\times m}$ is arbitrary.

Before we tackle the more general conditions, however, we will explore the implications of the well-posedness conditions of (2.38) and (2.39) on the closed-loop transfer function (from inputs to outputs) $G_{cl} = (I - W)^{-1}V$.

Proposition 2.5.2: *Let (W, V) be a proper DNF such that either $V = I$ or V is square, block diagonal, and a left factor of W. Then (W, V) is well-posed if and only if $G_{cl} = (I - W)^{-1}V$ exists, is unique, and is proper.* ♢

Proof In the case where $V = I$, this follows immediately since $G_{cl} = (I - W)^{-1}$, and (W, V) is well-posed if and only if $(I - W)^{-1}$ exists and is proper.

Now we consider the case where V is square, block diagonal, and a left factor of W. In particular, let $V = G \in RP^{p\times p}$ and $W = -GH$ where $H \in RP^{p\times p}$. Suppose that $(I - W) = (I + GH)$ has a proper inverse. Since both $G \in RP^{p\times p}$ and $(I + GH)^{-1} \in RP^{p\times p}$, we have that $G_{cl} = (I + GH)^{-1}G \in RP^{p\times p}$; hence G_{cl} exists, is unique, and is proper.

Now suppose that (Q, P) is not well-posed. We have two sub-cases to consider. The first is where $(I + GH)$ does not have an inverse. In this sub-case, either G_{cl} does not exist or is not unique. The second is where $(I + GH)$ does have an inverse, but the inverse is not proper. In this sub-case, $G_{cl} = (I + GH)^{-1}G$ does exist and is unique, thus we need to show that G_{cl} is improper to conclude the proof.

Assume, to the contrary, that G_{cl} is proper. Then, by Lemma 2.7.3 and Lemma 2.7.4, $G_{cl}H - I$ is also proper. We have that

$$\begin{aligned} G_{cl}H - I &= (I + GH)^{-1}GH - I \\ &= (I + GH)^{-1}\left(GH - (I + GH)\right)) \\ &= -(I + GH)^{-1}. \end{aligned}$$

But, we have already assumed that $(I + GH)^{-1}$ is improper, meaning that $-(I + GH)^{-1} = G_{cl}H - I$ is also improper by Lemma 2.7.4, providing our contradiction. ∎

Thus, by Proposition 2.5.2, well-posedness of interconnected systems (Definition 2.5.1) is equivalent to a check to see if the closed-loop transfer function exists, is unique, and is proper. This is sensible since only one of e or y is observed, and thus the existence, uniqueness, and properness of the closed-loop transfer function from u to e (or y) imply conditions (V1)-(V3) and vice-versa.

2.5.2 Well-Posedness of DNFs

As described in the previous section, when analyzing the set of interconnected systems considered by Vidyasagar, well-posedness is equivalent to the existence, uniqueness, and properness of the closed-loop transfer function. However, the set DNFs is a strict generalization of the set of interconnected systems considered by Vidyasagar in that V need not be the identity, a left factor of W, or even square. In this more general case, the invertibility of $(I - W(\infty))$ is a sufficient condition to guarantee the existence, uniqueness, and properness of the closed loop transfer function; however, it is not necessary (see Proposition 2.5.3 below).

Proposition 2.5.3: *Consider a DNF (W, V) where $W \in RP^{p\times p}$ and $V \in RP^{p\times m}$. If $(I - W)$ has a proper inverse, then $G_{cl} = (I - W)^{-1}V$ exists, is unique, and is proper.* ♢

Proof If $(I - W)$ has a proper inverse, then by Lemma 2.7.4, $G_{cl} = (I - W)^{-1}V$ exists, is unique, and is proper. However, to show that this condition is not necessary, we consider a counter example. Let

$$W = \begin{bmatrix} 0 & \frac{s+1}{s+2} \\ \frac{s+1}{s+3} & 0 \end{bmatrix}, \quad V = \begin{bmatrix} \frac{1}{s+4} & 0 \\ 0 & \frac{1}{s+5} \end{bmatrix}. \tag{2.40}$$

Then

$$(I-W)^{-1} = \begin{bmatrix} \frac{(s+2)(s+3)}{(3s+5)} & \frac{(s+1)(s+3)}{(3s+5)} \\ \frac{(s+1)(s+2)}{(3s+5)} & \frac{(s+2)(s+3)}{(3s+5)} \end{bmatrix} \tag{2.41}$$

is improper. However

$$G_{cl} = (I-W)^{-1}V = \begin{bmatrix} \frac{(s+2)(s+3)}{(s+4)(3s+5)} & \frac{(s+1)(s+3)}{(s+4)(3s+5)} \\ \frac{(s+1)(s+2)}{(s+5)(3s+5)} & \frac{(s+2)(s+3)}{(s+5)(3s+5)} \end{bmatrix} \tag{2.42}$$

exists, is unique, and is proper. ■

Note, however, the existence of a proper inverse of $(I-W)$ in Proposition 2.5.3 is not necessary to guarantee that G_{cl} exists, is unique, and is proper. We wish to discover the necessary *and* sufficient conditions for the well-posedness of general DNFs, and we seek to show that the necessary and sufficient conditions are that $(I-W)$ has a proper inverse, as it was in the previous section. However, if we adhere to Definition 2.5.1, and since we only measure signals y and u, the definition would seem to suggest that the necessary and sufficient conditions for well-posedness are that $G_{cl} = (I-W)^{-1}V$ exists, is unique, and is proper.

Nonetheless, as indicated by Proposition 2.5.3, the existence of a proper inverse of $(I-W)$ is sufficient to guarantee well-posedness of (W,V) and it is also necessary for *almost every* (W,V). In order for G_{cl} to exist, be unique, and proper when $(I-W)$ does not have a proper inverse, V needs to be in a way "special." For instance, the counterexample in the proof of Proposition 2.5.3 was generated by forcing V to be strictly proper with degrees in the denominator large enough to cancel the improperness of $(I-W)^{-1}$. Moreover, if we required V to be proper, we would have to engineer it in a way such that we exactly cancel any improperness in $(I-W)^{-1}$. Thus, if our specially-engineered V were to be perturbed slightly in either of these scenarios, then G_{cl} would suddenly become improper.

Definition 2.5.1 was built upon the work of Jan Willems [21], which contains an additional condition for well-posedness that [25] acknowledged but did not include as it was

unnecessary to that work. This additional condition suggests that small perturbations to the model should not result in a well-posed model suddenly becoming ill-posed. This idea is justified in that well-posedness is a statement on whether our model is a good approximation of reality. As our models will never be perfect representations of reality, well-posedness should not be sensitive to errors in our models.

As such, we use an expanded definition of well-posedness[7] over DNFs based on both [21] and [25].

Definition 2.5.4

Let (W, V) be a proper DNF. Then (W, V) is said to be well-posed if:

(W1) The closed-loop transfer function $G_{cl} = (I - W)^{-1}V$ mapping u to y exists and is unique.

(W2) The map from u to y is causal (i.e., G_{cl} is proper)

(W3) For each finite T, the dependence of $P_T y$ on $P_T u$ is Lipschitz continuous, where, again, P_T is the truncation operator

(W4) If (W1)-(W3) hold for (W, V), then small perturbations to W and V do not cause these conditions to fail.

With Definition 2.5.4, we are now prepared to provide the necessary and sufficient conditions for the well-posedness of arbitrary DNFs.

Proposition 2.5.5: *Let (W, V) be an arbitrary DNF with $W \in RP^{p\times p}$ and $V \in RP^{p\times m}$. Then (W, V) is well-posed according to Definition 2.5.4 (a) if and (b) only if $(I - W)$ has a proper inverse.* ♢

Proof (a) Suppose $(I-W)$ has a proper inverse. Then, by Proposition 2.5.3, conditions (W1)-(W3) of Definition 2.5.4 hold. Furthermore, it is well-known that invertible matrices are dense

[7]Note that the works [21] and [32] use all four conditions listed here, where [25] and [23] only use conditions (W1-W3) as described in Section 2.5.1, [22] uses only conditions (W1-W2), and [24] only uses condition (W1).

in the space of square matrices. By Lemma 2.7.7, we have that $\lim_{s\to\infty}(I-W) \in \mathbb{R}^{p\times p}$ is non-singular. Thus if we choose W_δ at random, then, with probability one, $\lim_{s\to\infty}(I-W)-\epsilon W_\delta$ will also be non-singular for any small $\epsilon > 0$, hence $(I-(W+\epsilon W_\delta))$ has a proper inverse by Lemma 2.7.7 and (W1)-(W3) still hold since $V+\epsilon V_\delta$ is proper.

(b) Suppose $(I-W)$ does not have a proper inverse. If the inverse of $(I-W)$ does not exist, then G_{cl} does not exist and condition (W1) fails. If $(I-W)^{-1}V$ is not proper, condition (W2) fails. Suppose, however, that $(I-W)^{-1}$ exists and is improper and that $G_{cl} = (I-W)^{-1}V$ is proper[8]. Thus, there exists an $i,j \in \{1,\dots,p\}$ such that $\left[(I-W)^{-1}\right]_{ij}$ is improper. Let

$$\operatorname{sign}\left(\left[(I-W)^{-1}\right]\right) = \begin{cases} 1 & \text{if } [G_{cl}]_{ik} \geq 0 \\ -1 & \text{otherwise.} \end{cases}$$

Let $[V_\delta]_{k1} = \operatorname{sign}\left([(I-W)^{-1}]\right)$ for $k = 1,\dots,p$. Then $\sum_{k=1}^{p}\left[(I-W)^{-1}\right][V_\delta]_{k1}$ must be improper; hence $(I-W)^{-1}V_\delta$, and therefore $(I-W)^{-1}(V+\epsilon V_\delta)$ for any $\epsilon > 0$ is also improper. Thus condition (W4) fails. ∎

2.5.3 Well-Posedness of the Generalized State Space Model

Since the intricacy-observed generalized state space model represents the interconnection of state space models in feedback, we wish to study the well-posedness of these models as well.

Definition 2.5.6

Interconnected state space models, such as the generalized state space model, are said to be *well-posed* if the output signals y exist and are unique given any choice of inputs u and state variables x [27].

With (2.5.6), we are prepared to give the necessary and sufficient conditions for the well-posedness of a generalized state space model.

[8] As a minor note, in this case, we must have $m > 0$ (where m is the number of columns in V) since if $m = 0$, $G_{cl} = (I-W)^{-1}$ is improper by assumption.

Proposition 2.5.7: *Let* $(A, \hat{A}, B, \begin{bmatrix} I & 0 \end{bmatrix}, \tilde{A}, \bar{B})$ *parameterize an intricacy-observed generalized state space of the form (2.19). Then this GSS is well-posed if and only if* $(I - \tilde{A})$ *is invertible.* ♢

Proof By solving for w and noting that $y = w$, we have that

$$\begin{aligned} \dot{x} &= \left[A + \hat{A}(I - \tilde{A})^{-1}\bar{A}\right] x + \left[B + \hat{A}(I - \tilde{A})^{-1}\bar{B}\right] u, \\ y &= (I - \tilde{A})^{-1}\bar{A}x + (I - \tilde{A})^{-1}\bar{B}u. \end{aligned}$$

If $(I - \tilde{A})^{-1}$ is invertible, then this is a standard state space model and y exists and can be computed uniquely given any x and u. If $(I - \tilde{A})^{-1}$ is not invertible, then there exists a choice of x and u such that y either does not exist or is not unique. ■

From Lemma 2.4.9, notice that, for any DNF $(W(s), V(s))$ and any intricacy-observed generalized state space realization $(A, \hat{A}, B, \bar{A}, \tilde{A}, \bar{B})$ of $(W(s), V(s))$, we have that $(I - W(\infty)) = (I - \tilde{A})$. Thus, Proposition 2.5.7 implies that $(W(s), V(s))$ is well-posed if and only if *every* intricacy-observed GSS realization of $(W(s), V(s))$ is also well-posed.

2.5.4 Summary of Well-Posedness Results

In summary, we can combine the previous results to get the following equivalent statements about the well-posedness of a DNF.

Theorem 2.5.8: *Let* $(W(s), V(s))$ *be a DNF with* $W(s) \in RP^{p\times p}$ *and* $V(s) \in RP^{p\times m}$. *Then the following are equivalent:*

(T1) $(W(s), V(s))$ *is well-posed by Definition 2.5.4.*

(T2) $(I - W(s))$ *has a proper inverse.*

(T3) $(I - W(\infty)) = \lim_{s\to\infty}(I - W(s))$ *is invertible.*

(T4) Every intricacy-observed generalized state space realization of $(W(s), V(s))$ *is well-posed.*

♢

PROOF The equivalence of (T1) and (T2) is given by Proposition 2.5.5. The equivalence of (T2) and (T3) is a restatement of Lemma 2.7.7. The equivalence of (T3) and (T4) follows from Lemma 2.4.9 and Proposition 2.5.7. ■

Corollary 2.5.9: *Let $(W(s), V(s))$ be a proper DNF where all entries in $W(s)$ are strictly proper. Then this DNF is well-posed.* ♢

PROOF Follows immediately from Theorem 2.5.8 and Lemma 2.7.9. ■

To demonstrate the synthesis of ideas contained in Theorem 2.5.8, we return to our running example.

Example 2.5.10

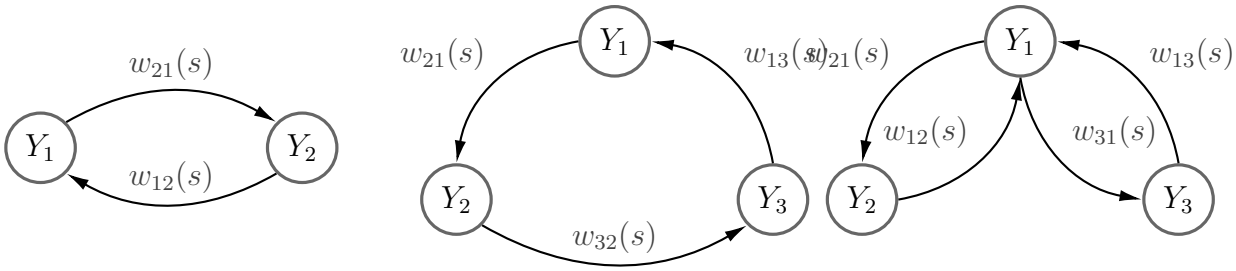
In 2.4.11, we connected two systems in feedback with static terms D_1 and D_2 respectively. The well-known condition for well-posedness of this interconnected systems is that

$$I - \begin{bmatrix} 0 & D_1 \\ D_2 & 0 \end{bmatrix} \tag{2.43}$$

is non-singular, or equivalently, that $\det(I - D_1 D_2) \neq 0$.

Also in Example 2.4.11, we represented this interconnection using a generalized state space, with $I - \tilde{A}$ precisely as given in (2.43); thus the conditions on the well-posedness of the generalized state space are precisely the conditions of well-posedness of the two systems connected in feedback.

Furthermore, we can convert this interconnection into a DNF either by (i) transforming both state space models into a DNF through the process contained in Section 2.4.1 and then connecting the DNFs in feedback (as shown in Example 2.4.5) or by (ii) stacking the state space models into a generalized state space model and converting that into a DNF through the process contained in Section 2.4.2 (as shown in Example 2.4.11). Regardless of the approach, the resultant DNF $(W(s), V(s))$ is the same, with



(a) A network with two links in feedback, which is well-posed if and only if $1 - w_{12}(s)w_{21}(s) \neq 0$.

(b) A network with three links oriented in a ring, which is well-posed if and only if $1 - w_{13}(\infty)w_{21}(\infty)w_{32}(\infty) \neq 0$.

(c) A network four links, which is well-posed if and only if $1 - w_{12}(\infty)w_{21}(\infty) - w_{23}(\infty)w_{32}(\infty) \neq 0$.

Figure 2.1 Example networks and their well-posedness properties as discussed in Sections 2.6.1, 2.6.2, and 2.6.3 respectively.

(for $i \in \{1, 2\}$):

$$W(s) = \begin{bmatrix} W_1(s) & V_1(s) \\ V_2(s) & W_2(s) \end{bmatrix}, \tag{2.44a}$$

$$W_i(s) = \frac{1}{s}\left[A_{i,11} + A_{i,12}(sI - A_{i,22})^{-1}A_{i,21}\right] \tag{2.44b}$$

$$V_i(s) = \hat{V}_i + (I - Q_i)D_i, \quad \text{with} \tag{2.44c}$$

$$\hat{V}_i(s) = \frac{1}{s}\left[B_{i,1} + A_{i,12}(sI - A_{i,22})^{-1}B_{i,2}\right].$$

Since $W_i(s)$ and $\hat{V}_i(s)$ are strictly proper (as expected by Theorem 2.4.3), we have that $W_i(\infty) = 0$ and $V_i(\infty) = D_i$. Thus $I - W(\infty)$ is also precisely (2.43) and the conditions of well-posedness on the DNF given in Theorem 2.5.8 are precisely the same as that on the generalized state space model as well as the feedback interconnection of the component state space models.

2.6 Additional Examples

We now include three additional examples that were first presented in [14] to further illustrate Theorem 2.5.8. These examples show that the conditions of well-posedness contained in Theorem 2.5.8 can lead to non-intuitive results.

2.6.1 Two Links in Feedback

Consider an arbitrary $W(s) \in RP^{2\times 2}$ as shown in Figure 2.1a. Then

$$I - W(\infty) = \begin{bmatrix} 1 & -w_{12}(\infty) \\ -w_{21}(\infty) & 1 \end{bmatrix},$$

where each $w_{ij}(s)$ is a SISO transfer function (i.e., $w_{ij}(\infty)$ is a scalar). In other words, this system contains a cycle at (Y_1, Y_2, Y_1). This system is well-posed if and only if $\det(I-W(\infty)) \neq 0$, which holds if and only if $1 - w_{12}(\infty)w_{21}(\infty) \neq 0$.

Note that this is precisely the well-known condition for well-posedness of two SISO systems connected in feedback. It is also a special case of Example 2.5.10 where, for $i, j \in \{1, 2\}$, $i \neq j$, we have $(W_i, V_i) = (0, w_{ij}(s))$.

2.6.2 A Ring

Let $W(s) \in RP^{3\times 3}$ where links are oriented in a ring such as that shown in Figure 2.1b. Without loss of generality, let

$$(I - W(\infty)) = \begin{bmatrix} 1 & 0 & -w_{13}(\infty) \\ -w_{21}(\infty) & 1 & 0 \\ 0 & -w_{32}(\infty) & 1 \end{bmatrix}.$$

By Theorem 2.5.8, this system is well-posed if and only if $\det(I - W(\infty)) \neq 0$ which is true if and only if $1 - w_{13}(\infty)w_{21}(\infty)w_{32}(\infty) \neq 0$.

2.6.3 The Adjoining of Two Feedback Loops

Let $W(s) \in RP^{3\times 3}$ be defined as shown in Figure 2.1c. Thus

$$I - W(\infty) = \begin{bmatrix} 1 & -w_{12}(\infty) & 0 \\ -w_{21}(\infty) & 1 & -w_{23}(\infty) \\ 0 & -w_{32}(\infty) & 1 \end{bmatrix}.$$

By Theorem 2.5.8, the condition of well-posedness of this system is that $1 - w_{12}(\infty)w_{21}(\infty) - w_{23}(\infty)w_{32}(\infty) \neq 0$.

Notice from Figure 2.1c that this example contains two cycles of the type described in Section 2.6.1, namely (y_1, y_2, y_1) and (y_2, y_3, y_2). However, the condition $(1-w_{12}(\infty)w_{21}(\infty)-w_{23}(\infty)w_{32}(\infty) \neq 0)$ is different than the union of conditions on the individual loops $(1 - w_{12}(\infty)w_{21}(\infty) \neq 0$ *and* $1 - w_{23}(\infty)w_{32}(\infty) \neq 0)$. As such, it is possible for each of the two cycles to be ill-posed, but the overall system adjoining these two cycles at output 2 is actually well-posed.

To illustrate this, let

$$I - W(\infty) = \begin{bmatrix} 1 & -.5 & 0 \\ -2 & 1 & -2 \\ 0 & -.5 & 1 \end{bmatrix}.$$

Notice that the cycle (y_1, y_2, y_1) is ill-posed since $1-w_{12}(\infty)w_{21}(\infty) = 1-(.5)(2) = 0$. Likewise, the cycle (y_2, y_3, y_2) is ill-posed since $1-w_{23}(\infty)w_{32}(\infty) = 1-(2)(.5) = 0$. However, the overall system is actually well-posed since $1-w_{12}(\infty)w_{21}(\infty)-w_{23}(\infty)w_{32}(\infty) = 1-(.5)(2)-(2)(.5) = -1 \neq 0$.

It is also possible to adjoin two well-posed cycles to make the overall system ill-posed. To illustrate this, let

$$I - W(\infty) = \begin{bmatrix} 1 & -.5 & 0 \\ -1 & 1 & -1 \\ 0 & -.5 & 1 \end{bmatrix}.$$

Notice that the cycle (y_1, y_2, y_1) is well-posed since $1 - w_{12}(\infty)w_{21}(\infty) = 1 - (.5)(1) = .5 \neq 0$. Likewise, the cycle (y_2, y_3, y_2) is well-posed since $1 - w_{23}(\infty)w_{32}(\infty) = 1 - (1)(.5) = .5 \neq 0$. However, the overall system is actually ill-posed since $1 - w_{12}(\infty)w_{21}(\infty) - w_{23}(\infty)w_{32}(\infty) = 1 - (.5)(1) - (1)(.5) = 0$.

2.7 Appendix

Consider a polynomial $p(s)$ in terms of $s \in \mathbb{C}$. Then the *degree* of $p(s)$, written $\deg p(s)$, is the highest degree of its monomials with non-zero terms. We say that $g(s)$ is a *rational polynomial* if it can be written as $g(s) = \frac{n(s)}{d(s)}$, where $n(s)$ and $d(s)$ are both polynomials. If $\deg n(s) \leq \deg d(s)$, we say that $g(s)$ is *proper*, and if $\deg n(s) < \deg d(s)$, we say that $g(s)$ is *strictly proper*.

Given these definitions and the definitions in Section 2.2, we can list several results on matrices of rational polynomials that we use throughout this work, which are well known. The first several are known and pertain to the closure of arithmetic over proper and strictly proper rational polynomials and matrices of rational polynomials.

Lemma 2.7.1: *Let $p(s)$ and $q(s)$ be rational polynomials. If $p(s)$ and $q(s)$ are proper, then $p(s) + q(s)$ is proper. Furthermore if both are strictly proper, then the sum is likewise strictly proper.* □

PROOF Let $p(s) = \frac{n_p(s)}{d_p(s)}$ and $q(s) = \frac{n_q(s)}{d_q(s)}$ for polynomials $n_p(s)$, $n_q(s)$, $d_p(s)$, $d_q(s)$. By assumption, $\deg n_p(s) \leq \deg d_p(s)$ and $\deg n_q(s) \leq \deg d_q(s)$. Then

$$p(s) + q(s) = \frac{n_p(s)d_q(s) + n_q(s)d_p(s)}{d_p(s)d_q(s)} \triangleq \frac{n(s)}{d(s)}.$$

We have, by properties of polynomial degrees, that

$$\begin{aligned}\deg n(s) &= \max\left(\deg n_p(s) + \deg d_q(s), \deg n_q(s) + d_p(s)\right) \\ &\leq \max\left(\deg d_p(s) + \deg d_q(s), \deg d_q(s) + d_p(s)\right) \\ &= \deg d_p(s) + \deg d_q(s) = \deg d(s),\end{aligned}$$

thus the sum of proper rational polynomials is proper.

To show that the sum of strictly proper rational polynomials is strictly proper, replace the inequalities above with strict inequalities. ■

Lemma 2.7.2: *Let $p(s)$ and $q(s)$ be rational polynomials. If $p(s)$ and $q(s)$ are proper, then $p(s)q(s)$ is proper. Furthermore if one of $p(s)$ and $q(s)$ is strictly proper, then the product is likewise strictly proper.* □

PROOF Let $p(s) = \frac{n_p(s)}{d_p(s)}$ and $q(s) = \frac{n_q(s)}{d_q(s)}$ for polynomials $n_p(s)$, $n_q(s)$, $d_p(s)$, $d_q(s)$. By assumption, $\deg n_p(s) \leq \deg d_p(s)$ and $\deg n_q(s) \leq \deg d_q(s)$. Then

$$p(s)q(s) = \frac{n_p(s)n_q(s)}{d_p(s)d_q(s)} \triangleq \frac{n(s)}{d(s)}.$$

We have, by properties of polynomial degrees, that

$$\deg n(s) = \deg n_p(s) + \deg n_q(s) \leq \deg d_p(s) + \deg d_q(s) = \deg d(s).$$

Thus the product of proper rational polynomials is proper.

Without loss of generality, assume that $p(s)$ is strictly proper and $q(s)$ is proper. Then $\deg n_p(s) < \deg d_p(s)$. We thus have

$$\deg n(s) = \deg n_p(s) + \deg n_q(s) < \deg d_p(s) + \deg d_q(s) = \deg d(s).$$

Thus the product of proper rational polynomials, where one is strictly proper, is likewise strictly proper. ■

Lemma 2.7.3: *Let $M(s)$ and $N(s)$ be proper matrices of rational polynomials, each of dimension $m \times n$. Then $M(s) + N(s)$ is likewise proper. Furthermore, if both $M(s)$ and $N(s)$ are strictly proper, the sum is also strictly proper.* □

PROOF Follows immediately from Lemma 2.7.1 and the fact that $[M(s)+N(s)]_{ij} = [M(s)]_{ij} + [N(s)]_{ij}$. ■

Lemma 2.7.4: *Let $M(s)$ be a matrix of rational polynomials of dimension $p \times m$, and let $N(s)$ be a matrix of rational polynomials of dimension $m \times n$. If $M(s)$ and $N(s)$ are both proper, then $M(s)N(s)$ is proper. Likewise, if one of $M(s)$ and $N(s)$ are strictly proper, then the product is also strictly proper.* □

PROOF This follows immediately from Lemma 2.7.1, Lemma 2.7.2, and the fact that $[M(s)N(s)]_{ij} = \sum_{k=1}^{m} [M(s)]_{ik}[N(s)]_{kj}$. ■

We now discuss the relationship between proper matrices of rational polynomials and state space models. See [27] for a discussion of these results.

Lemma 2.7.5: *Let (A, B, C, D) characterize a system as given by the state space equations*

$$\begin{aligned} \dot{x}(t) &= Ax(t) + Bu(t), \\ y(t) &= Cx(t) + Du(t), \end{aligned} \tag{2.45}$$

with $A \in \mathbb{R}^{n \times n}$, $B \in \mathbb{R}^{n \times m}$, $C \in \mathbb{R}^{p \times n}$, and $D \in \mathbb{R}^{p \times m}$.

Then the rational polynomial $G(s) = C(sI - A)B + D$ representation of that system is proper; i.e., $G(s) \in RP^{p\times m}$. □

PROOF See [27], pp. 92-93. ■

Lemma 2.7.6: *Let $G(s) \in RP^{p\times m}$ be an arbitrary proper matrix of rational polynomials. Then $G(s) = C(sI - A)^{-1}B + D$ for some choice of $A \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{n\times m}$, $C \in \mathbb{R}^{p\times n}$, and $D \in \mathbb{R}^{p\times m}$. In other words, the set $RP^{p\times m}$ of all proper matrices of rational polynomials is equal to the set of state space transfer functions with m inputs and p outputs.* □

PROOF See [27], pp. 93-94. ■

The next result, from [27, 33], gives conditions for when $G(s)$ has a proper inverse.

Lemma 2.7.7: *Let $G(s)$ be a square and proper matrix of rational polynomials. Then the inverse of $G(s)$ exists and is proper (a) if and (b) only if $G(\infty) \triangleq \lim_{s\to\infty} G(s)$ is non-singular.* □

PROOF Since $G(s)$ is proper, by Lemma 2.7.6, it has a state space representation (A, B, C, D) with $A \in \mathbb{R}^{n\times n}$ for some n, $B \in \mathbb{R}^{n\times p}$, $C \in \mathbb{R}^{p\times n}$ and $D \in \mathbb{R}^{p\times p}$, where $G(s) = C(sI - A)^{-1}B + D$. It follows that $G(\infty) = \lim_{s\to\infty} G(s) = D$ (since $(sI - A)^{-1} \to 0$ as $s \to \infty$).

(a) Suppose that $G(\infty) = D$ is non-singular. Then, using the Sherman-Morrison-Woodbury formula, we can express G^{-1} as

$$\begin{aligned} G^{-1}(s) &= \big(C(sI - A)^{-1}B + D\big)^{-1} \\ &= -D^{-1}C(sI - A + BD^{-1}C)^{-1}BD^{-1} + D^{-1} \\ &\triangleq \tilde{C}(sI - \tilde{A})^{-1}\tilde{B} + \tilde{D}, \end{aligned}$$

where $sI - A + BD^{-1}C$ is invertible due to the s terms appearing on the diagonal and only the diagonal. Since $G^{-1}(s)$ is the transfer function representation of the state space $(\tilde{A}, \tilde{B}, \tilde{C}, \tilde{D})$, by Lemma 2.7.5, $G^{-1}(s)$ must be proper.

(b) Let $G(s)$ be proper but suppose that $G(\infty)$ is singular. We must show that either $G(s)$ does not have an inverse or that the inverse of $G(s)$ is improper.

For brevity, we write $G(s) = \hat{G}(s) + D$, where $\hat{G}(s) = C(sI - A)^{-1}B$, where again, A, B, C, and D are of the proper dimensions and $D = G(\infty)$. This time, adding $\hat{G}(s)$ to D could potentially increase the rank of the sum, leaving us with two cases. The first case occurs where the rank of the sum is still deficient, meaning $G(s)$ is not invertible, and no further analysis is needed. The second and more interesting case occurs when $\hat{G}(s)$ added to D increases the rank of the sum sufficiently that $G(s)$ is invertible. We must show that, in this case, $G^{-1}(s)$ is improper.

Assume, to the contrary, that $G^{-1}(s)$ is proper. Then $G^{-1}(s)$ has a state space realization $(\tilde{A}, \tilde{B}, \tilde{C}, \tilde{D})$ such that $G^{-1}(s) = \tilde{C}(sI - \tilde{A})^{-1}\tilde{B} + \tilde{D} \triangleq \tilde{G}(s) + \tilde{D}$. Furthermore, we have that

$$\begin{aligned} G^{-1}(s)G(s) &= \left(\tilde{G}(s) + \tilde{D}\right)\left(\hat{G}(s) + D\right) \\ &= \tilde{G}(s)\hat{G}(s) + \tilde{G}(s)D + \tilde{D}\hat{G}(s) + \tilde{D}D \\ &= I. \end{aligned}$$

Since $\tilde{G}(s)\hat{G}(s)$, $\tilde{G}(s)D$, and $\tilde{D}\hat{G}(s)$ are all strictly proper, their sum must be zero, leaving $\tilde{D} = D^{-1}$. However, this contradicts our original assumption that D is not invertible. Thus $G^{-1}(s)$ must be improper. ∎

We now wish to give results about limits of a proper matrix of rational polynomials $M(s)$ as $s \to \infty$.

Lemma 2.7.8: *Let $g(s)$ be a proper rational polynomial. Then $g(\infty) \triangleq \lim_{s\to\infty} g(s) = 0$ (a) if and (b) only if $g(s)$ is strictly proper.* □

Proof (a) Suppose $g(s)$ is strictly proper. Since the degree of the denominator of $g(s)$ is strictly larger than the degree of the numerator, the limit will approach zero.

(b) Suppose that $g(s)$ is proper but not strictly proper. Since the degree of the denominator of $g(s)$ is equal to the degree of the numerator, the limit will approach $\frac{n}{d} \neq 0$, where n is the coefficient of the highest-order monomial and d is the coefficient of the highest-order monomial in the denominator. ■

Lemma 2.7.9: *Let $M(s)$ be an arbitrary matrix of rational polynomials. Then $M(\infty) \triangleq \lim_{s\to\infty} M(s) = 0$ if and only if $M(s)$ is strictly proper.* □

PROOF Follows immediately from Lemma 2.7.8. ■

Lemma 2.7.10: *Let $M(s)$ and $N(s)$ be proper matrices of rational polynomials. Then $\lim_{s\to\infty} [M(s) + N(s)] = [\lim_{s\to\infty} M(s)] + [\lim_{s\to\infty} N(s)]$.* □

PROOF Follows immediately from the sum rule of limits. ■

Lemma 2.7.11: *Let $M(s)$ and $N(s)$ be proper matrices of rational polynomials. Then $\lim_{s\to\infty} [M(s)N(s)] = [\lim_{s\to\infty} M(s)] \cdot [\lim_{s\to\infty} Ns)]$.* □

PROOF We have, by the sum and product rule of limits,

$$\begin{aligned}\lim_{s\to\infty} [M(s)N(s)]_{ij} &= \lim_{s\to\infty} \sum_k [M(s)]_{ik} [N(s)]_{kj} \\ &= \sum_k \lim_{s\to\infty} [M(s)]_{ik} [N(s)]_{kj} \\ &= \sum_k \left[\lim_{s\to\infty} M(s)\right]_{ik} \left[\lim_{s\to\infty} N(s)\right]_{kj}.\end{aligned}$$

Thus $\lim_{s\to\infty} [M(s)N(s)] = [\lim_{s\to\infty} M(s)] \cdot [\lim_{s\to\infty} Ns)]$. ■

This final result is well-known and is a variation on the matrix determinant lemma.

Lemma 2.7.12: *Let $G, H \in \mathbb{R}^{p\times p}$. Then $\det(I + GH) = \det(I + HG)$.* □

PROOF We have that

$$LMR \triangleq \begin{bmatrix} I & 0 \\ G & I \end{bmatrix} \begin{bmatrix} I+HG & H \\ 0 & I \end{bmatrix} \begin{bmatrix} I & 0 \\ -G & I \end{bmatrix}$$
$$= \begin{bmatrix} I & H \\ 0 & I+GH \end{bmatrix} \triangleq N.$$

Thus

$$\det(LMR) = \det L \cdot \det M \cdot \det R = I \cdot \det(I+HG) \cdot I$$
$$= \det(I+HG) = \det(N) = \det(I+GH),$$

as desired. ∎

Chapter 3

Representability and Abstractions of LTI Dynamic Networks

This chapter establishes the importance of abstractions and realizations of dynamic networks in characterizing the structure and dynamics of systems. Abstractions and realizations generate dynamically equivalent representations of systems with varying degrees of structural detail. In this chapter, we characterize four types of abstractions, which we call the node abstraction, the edge abstraction, the immersion, and the stacked immersion. We give conditions under which these abstractions are representable and well-posed. Furthermore, we use the stacked immersion to show that dynamic networks exist that contain the same level of detail as state space models, that other dynamic networks exist that contain the same level of detail as transfer functions, and that still other dynamic networks exist that are simultaneously abstractions of state space models and realizations of transfer functions; thus containing intermediate levels of structural detail. We include examples illustrating these points and show how we can apply them to the problem of network reconstruction.

3.1 Introduction

We can represent the dynamics of an LTI system in many ways, such as through a state space model or a transfer function. For applications where knowledge of both the dynamic behavior and the computational structure of the system are required, dynamic networks have increasingly become the representation of choice [4–8]. We begin by introducing dynamic networks as represented by the dynamical network function (DNF).

3.1.1 Background: Dynamic Networks

Consider N manifest (or visible, or measured) real-valued signals $w(t) = [w_1(t), \ldots, w_N(t)]'$, where $t \in \mathbb{R}$ or $t \in \mathbb{Z}$ (in this work we will focus on $t \in \mathbb{R}$, although all the results hold for $t \in \mathbb{Z}$ as well). The behavior of any such system can be defined by a constraint $f(w) = 0$ for some operator, f, distinguishing allowed values of the manifest variables from disallowed values. In general, there may be many such representations with the same solution set, or behavior. If f is bijective, then we can say that the system is autonomous. However, if f is not injective, then the manifest signals can be partitioned (possibly non-uniquely) into *inputs* and *outputs*, where the outputs are uniquely specified for any choice of input signal. This work considers linear finite-dimensional systems, where a partition can always be found (possibly non-uniquely) for which the outputs are not only uniquely specified given arbitrary inputs, but they are also causally determined from these inputs. Note that a specification of which signals are inputs and which are outputs must be given as additional information–it can not be derived in general from f–and this partition introduces a notion of causality on the system.

In this work, we are concerned primarily with LTI systems. Thus, we can write the behavior as $f(w) = \bar{W}w = 0$, where $\bar{W}$ is an $N \times N$ matrix of SISO causal operators[1]. Let $p = \operatorname{rank} \bar{W}$ and $m = N - p$. By partitioning w into m inputs[2] u and p outputs y, we can again rewrite this behavior, without loss of generality, as

$$\begin{bmatrix} (I_{p\times p} - W) & -V \\ 0 & 0 \end{bmatrix} \begin{bmatrix} y \\ u \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \implies y = Wy + Vu. \tag{3.1}$$

[1]For simplicity, we will assume that these operators are continuous-time systems in the frequency domain, meaning that each operator is a rational function in the Laplace variable $s \in \mathbb{C}$. Causality implies that these rational functions are proper. The results in this work also generalize naturally to discrete-time and time-domain operators as well.

[2]Some works (e.g., [1, 28]) further partition inputs based on whether they are controlled, uncontrolled, deterministic, stochastic, etc. We make no such distinction in this work, although we do remind the reader that these signals are all *manifest*, not fictitious (e.g., white noise) as part of a particular modeling framework.

where W is a $p \times p$ matrix and V is a $p \times m$ matrix. We call the tuple (W, V) the *dynamical network function* (DNF) of this system. Matrix V defines the causal dependencies of each output y on each input u, independent of all other manifest variables in the network. Matrix W defines the causal dependencies of each output y on every other output y, again independent of all manifest variables in the network. Thus, the DNF defines a network of causal dependencies among manifest signals. This network can be rigorously characterized by a graph called the *signal structure*, with nodes representing each manifest signal and directed edges labeled with (possibly) dynamic operators characterizing the causal dependency of one signal on another. See Section 3.2.1 for more details and also Figure 3.1 for examples of signal structures.

If W (and hence $\bar{W}$) is hollow (meaning that all hollow entries are zero), then we call the representation the *dynamical structure function* (DSF), and we typically use the notation (Q, P) in place of (W, V) [1, 15]. We will discuss the distinction between DNFs and DSFs further in Section 3.4.2.

By assuming that $(I - W)$ is well-posed (implying that W has a proper inverse [16]) and solving for y, we can re-arrange the DNF such that

$$y = \left[(I - W)^{-1} V\right] u \triangleq G_{cl} u, \tag{3.2}$$

where G_{cl} is the closed-loop transfer function characterizing the input-output behavior of this system.

3.1.2 Abstractions and Realizations of Dynamic Networks

An abstraction of some DNF (W, V) is another dynamic network with equivalent input-output behavior but with *less* structural detail. Similarly, a realization of (W, V) is another dynamic network, again with equivalent input-output behavior but with *more* structural detail. For network reconstruction, this implies that an abstraction is cheaper to reconstruct than the original DNF. Thus, abstractions provide two critical benefits which motivate this work:

- The cost of reconstructing an abstraction (in terms of the amount of a priori structural information required to identify the network uniquely) is strictly less than the cost of reconstructing the original network (see Section 3.2.3).
- Abstractions represent a system at varying levels of detail. In studies of robustness and vulnerability of a networked system (see, for instance, [6, 9, 11]), this allows us to study both the vulnerability at an outsider's level of information (by considering an abstraction with less structural detail) or an insider's level of detail (by considering a realization with more structural detail).

In this work, we discuss four different types of abstractions, the node abstraction, the edge abstraction (with the emphasis on a particular type of edge abstraction called the hollow abstraction), the immersion, and the stacked immersion.

The node abstraction (Section 3.3) is found by reducing the number of manifest variables in the network that still preserves the dynamics of the network and pattern of independence among the remaining manifest variables. The edge abstraction (Section 3.4) is found by removing specific edges in the network while still preserving dynamics and the remaining Boolean structure. The hollow abstraction is a particular edge abstraction that is found by removing all self-loops in the network, and results (as shown in Section 3.4.2) in a DSF instead of a DNF.

If we compose the process for finding a node abstraction with that for finding an edge abstraction, the result is an abstraction that we call the immersion or the immersed network [34, 35]. The *immersion* (which has at times been simply called the *abstraction*) has been previously explored in many ways. The term "immersion" was introduced in [34] in reference to the output elimination procedure described in Section 3.3. The works [34–36] leverage the reduced amount of identifiability conditions required for reconstructing an immersion to identify specific links in the network as opposed to the network as a whole. In [2, 30, 37–39], the DSF itself is described as simultaneously an immersion of a state space model and a realization of a transfer function. This idea is formalized in [40], which discusses

the identifiability of a state space model from a DSF and provides necessary and sufficient conditions under which an immersion creates hidden states in the network that are shared among the edges. The works [14, 15] and Chapter 2 of this dissertation give conditions on when the immersion is proper and well-posed.

We construct the stacked immersion using the immersion as a foundation. A key idea highlighted by this paper is that there exists a DSF, called the *full-state DSF*, that is informationally equivalent to a state space model. Furthermore, there exists a stacked immersion of the full-state DSF–called the final immersion–that is informationally equivalent to the transfer function. There are also many stacked immersions of the full-state DSF and realizations of the final immersion that represent intermediate levels of structural information. Thus, we can describe the DSF (and the multi-DSF, which we introduce in Section 3.6) as a "partial structure representation" of a system.

REMARK 3.1.1: Throughout this work, we use the words "abstraction" and "immersion" as a noun, referring to a specific DNF or DSF derived from a more structurally informative network. Prior works often use the word "abstraction" (and "immersion") as a verb referring to the process converting a DNF into its abstraction. In this work, we will maintain a distinction between the abstraction as a network and the process (or function or method) used to compute the abstraction from the base network.

3.1.3 Related Work and Contributions

This chapter is a direct extension of [3] and a follow-on to [16]; however, it does contain the following novel contributions in addition to those works:

- Generalization of previous results over DSFs into the realm of DNFs.
- Definition of the edge abstraction (Section 3.4).
- Separation of the immersion into the node abstraction and the hollow abstraction, with a discussion on how these relate to DNFs and DSFs (Sections 3.3 and 3.4).

- A definition and discussion of the "independence pattern" of a network, which is the characteristic attribute defining the node abstraction.
- Proof that the process for computing the node abstraction is unique given the definition of the node abstraction.
- Separation of the principles of *representability* and *well-posedness*, with the implication that, assuming that some abstraction (node, hollow, or immersion) of a given base network (DNF or DSF) is representable, the abstraction is well-posed if and only if the base network is also well-posed (see Sections 3.3.4 and 3.4.4).

3.2 Background

Much of this work centers around functions of the form $g(s) = \frac{n(s)}{d(s)}$, where $n(s)$ and $d(s)$ are polynomials in $s \in \mathbb{C}$. If the degree of $n(s)$ is no greater than the degree of $d(s)$, we say that $g(s)$ is *proper*. If the degree of $n(s)$ is strictly less than the degree of $d(s)$, then we say that $g(s)$ is *strictly proper*.

We are also concerned with matrices consisting of these rational functions. We say that the matrix $G(s) = [g_{ij}(s)]$ is *proper* if every rational function $g_{ij}(s)$ is proper, and likewise $G(s)$ is *strictly proper* if every $g_{ij}(s)$ is strictly proper. In the vein of [27], we define $RP^{m\times n}$ as the set of all $m \times n$ matrices of proper (and not necessarily strictly proper) rational functions. Also, for notational simplicity, we will often drop the dependency of these matrices on s; i.e., we write $G(s)$ simply as G.

We say that $G \in RP^{p\times p}$ is *invertible almost everywhere* if it is singular only for a finite choice of values $s \in \mathbb{C}$. For brevity, from this point forward, when we say that G is invertible, we mean that it is invertible almost everywhere.

3.2.1 The Signal Structure Graph

Let (W, V) be a DNF defining the relationship $Y = WY + VU$ with $W \in RP^{p\times p}$, $V \in RP^{p\times m}$, $Y = [y_1 \ \cdots \ y_p]'$, and $U = [u_1 \ \cdots \ u_m]'$. Associated with this DSF is a weighted directed graph $\Gamma(W, V) = (\Gamma_V, \Gamma_E, \Gamma_W)$, where the vertices are defined by the set

$$\Gamma_V = \{y_1, \cdots, y_p\} \cup \{u_1, \cdots, u_m\}, \tag{3.3}$$

and the edges are defined by the set

$$\Gamma_E = \{(y_i, y_j) \mid i, j = 1, \ldots, p\} \cup \{(u_i, y_j) \mid i = 1, \ldots, m, j = 1, \ldots, p\}. \tag{3.4}$$

Furthermore, Γ_W assigns weights to the edges by defining the relation $\Gamma_W : \Gamma_E \to RP$ (note that the edge weights are in RP rather than $\mathbb{R}$ as is common in graph theory) such that

$$\Gamma_W(y_i, y_j) = W_{ji}, \quad \Gamma_W(u_i, y_j) = V_{ji}. \tag{3.5}$$

In other words, the edges are defined by the Boolean structure of W and V and the weights are defined by the entries (operators) in W and V.

We can also import existing graph-theoretic terms and concepts. Of greatest import to this work is the notion of a *walk*, with *net effect* defined over a walk.

Definition 3.2.1: Walk

A *walk* from $y_i \in V$ (or $u_i \in V$) to $y_j \in V$ is a sequence of edges $\mathcal{W} = [(y_i, y_{k_1}) \in E, \ldots, (y_{k_m}, y_j) \in E]$ such that the source of the first edge is y_i and the sink of the last edge is y_j. In a walk, vertices and edges are both allowed to repeat in the sequence.

Definition 3.2.2: Effect

Let $\mathcal{W} = [(y_i, y_{k_1}) \in E, \ldots, (y_{k_m}, y_j) \in E]$ be some walk in the graph of (W, V) from y_i to y_j, and let $\mathcal{J}(\mathcal{W}) = [(i, k_1), \ldots, (k_m, j)]$ index the nodes in this walk. Then we define the *effect* of this walk to be

$$\mathcal{E}(\mathcal{W}) = \prod_{\hat{i},\hat{j} \in \mathcal{J}(\mathcal{W})} W_{\hat{j},\hat{i}}. \tag{3.6}$$

In other words, the effect of some walk is the product of the weights of all edges in the walk.

Since we can interpret edges as a subsystem represented by a SISO transfer function, the effect of a walk is equivalent to the system resulting from connecting all of the subsystems represented by the edges in the walk in sequence.

Definition 3.2.3: Net Effect

Let i, j index two nodes in our graph and let $\bar{S}$ index some subset of the nodes in our graph such that $i \notin \bar{S}$ and $j \notin \bar{S}$. We define the *net effect* from node i to node j with respect to $\bar{S}$ to be the sum of the effects of all of the walks connecting i to j and passing through some subset of the nodes in $\bar{S}$ only (including the empty set).

If $\bar{S}$ is not specified, then the net effect is the sum of the effects of all walks in the graph from i to j.

In the case where the net effect is convergent, we can show that the net effect is from u_i to y_j is $G_{ij} = [(I - W)^{-1}V]_{ij}$. If the net effect is not convergent, we still define it as G_{ij} rather than the divergent sum of all walks [17].

We also define the *independence pattern*, which relates to the Boolean structure of the DNF (W, V).

Definition 3.2.4: Independence Pattern

Edges in the signal structure are essential to the interpretation of the DNF. If there exists a (non-zero) edge from y_i to y_j, this means that there is a component of signal y_j that is computed directly from y_i and independent of all other manifest signals y_k and u_l in the network (and likewise for an edge from u_i to y_j). Furthermore, since W and V are matrices of proper rational polynomials, this computation is causal. However, if the edge from y_i to y_j is zero, this means that either (i) there is no direct computation of y_j from y_i independent from all other manifest signals, or (ii) that the direct computation has been canceled exactly across the latent variables in the network. Thus, the graph of the DSF represents the direct and causal dependence among variables manifest in the network. We call these direct causal dependencies and independencies the *independence pattern* of the network.

3.2.2 Well-Posedness of DNFs

As discussed in [16] and Chapter 2, a DNF model (W, V) is considered to be well-posed if it is a reasonable approximation of reality. Following the definitions in [21, 25], we say that a DNF (as well as a DSF) defining the relationship $Y = WY + VU$ is *well-posed* if the following conditions are satisfied (see Definition 2.5.4):

(W1) Signals in Y exist and are unique given signals in U (i.e., the closed-loop transfer function G_{cl} exists and is unique).

(W2) The dependence of Y on U is causal (i.e., G_{cl} is proper).

(W3) For each finite T and the truncation operator P_T, the dependence of $P_T Y$ on $P_T U$ is Lipschitz continuous.

(W4) If (W1)-(W3) above hold for (W, V), small perturbations to W and V do not cause any of these conditions to fail.

Chapter 2 and the works [14–16] then proceed to provide the necessary and sufficient conditions for well-posedness, which are summarized as follows (see Theorem 2.5.8):

Proposition 3.2.5: *Let $(W(s), V(s))$ be a proper DNF. Then the following statements are equivalent:*

- *$(W(s), V(s))$ is well-posed.*
- *$(I - W(s))$ has a proper inverse.*
- *$(I - W(\infty)) \triangleq \lim_{s\to\infty}(I - W(s))$ is non-singular.* ♢

3.2.3 Network Reconstruction

Let $G \in RP^{p\times m}$ be a transfer function that is either given or identified from data. Network reconstruction is the process of leveraging structural information (which will be defined below) to find the *unique* DNF (W, V) such that[3] $W \in RP^{p\times p}$, $V \in RP^{p\times n}$, and $G = (I - W)^{-1}V$.

As shown in [19, 41, 42], we can recover W and V by writing

$$G = (I - W)^{-1}V = \begin{bmatrix} W & V \end{bmatrix} \begin{bmatrix} G \\ I_p \end{bmatrix} \implies G' = \begin{bmatrix} G' & I_p \end{bmatrix} \begin{bmatrix} W' \\ V' \end{bmatrix} \implies \vec{g} = L\vec{\theta}, \tag{3.7}$$

where $\vec{g} \in RP^{pm}$ is a vectorization of the pm known entries in G, $\vec{\theta} \in RP^{p^2+pm}$ is a vectorization of the $p^2 + pm$ unknown entries in $\begin{bmatrix} W & V \end{bmatrix}'$, and $L \in RP^{pm\times(p^2+pm)}$ is constructed to preserve the transformation $\begin{bmatrix} G' & I_p \end{bmatrix}$ between the new vectorized spaces.

Note every entry in L and $\vec{g}$ is known, so the network reconstruction problem reduces to finding a unique $\vec{\theta}$. However, since $p^2 + pm > pm$, this problem is ill-posed. Hence, we must reduce the dimensions of the columns such that the number of columns is less than or

[3] None of these matrices actually need to be strictly proper, and the network reconstruction methodology presented below works even if the operators are improper or even functions over s that are not rational polynomials. We will focus our discussion here, however, on the case where W and V are matrices of rational polynomials.

equal to pm. We can do so by leveraging *a priori structural information* about the DNF, which may take the following form

- Knowledge that an entry in V or W is zero.
- Knowledge that an entry in V or W is a known linear combination of other unknown entries in V or W.

If we know *at least* p^2 entries in V and W according to the above criteria, then we can recover V and W uniquely. We call the number p^2 the *identifiability index*[4] required for the reconstruction of (W, V).

We can reduce the identifiability index required to reconstruct a network by finding an abstraction of that network and reconstructing the abstraction instead. For instance, a DSF is an abstraction of a DNF where the p diagonal entries of W are zero, which allows us to remove p columns from L reducing the identifiability index from p^2 to $p^2 - p$ (see Section 3.4.5). Moreover, since much of the network reconstruction literature centers on DSFs as opposed to DNFs, $p^2 - p$ often appears as the necessary identifiability index required (see, for instance, [19]).

3.3 Node Abstractions

Suppose that we wish to stop observing or modeling specific output signals in some network. In other words, suppose we wish to reduce the set of variables that are manifest in the system (by "abstracting away" the complementary set). Since we define links in the DNF as the interaction between a source and destination manifest variable, independent from other manifest variables (see Section 3.2.1), the DNF must change in order to reflect our new

[4]Some works, such as [19, 43], refer to this number as the *informativity conditions*. Other works, such as [44, 45], refer to it as the *identifiability conditions* (or number of conditions for the identifiability of the network). As we will note in Remark 4.3.6 (Chapter 4), the identifiability (informativity) conditions specify the set of necessary and sufficient information required to reconstruct the network whereas the identifiability index is a measure of the necessary minimum size of this set.

manifest set without discarding the internal dynamics of the system. We call the result of the transformation where we reduce the number of outputs Y measured a *node abstraction*.

3.3.1 Node Abstraction Definitions

We seek to provide a precise definition for a node abstraction. In short, a node abstraction contains fewer nodes in the network, but is *dynamically equivalent* to the original network and *preserves the independence pattern*. We define a node abstraction using the properties "dynamically equivalent" and "independence-pattern-preserving" here.

Definition 3.3.1: Node Abstraction

Consider the DNF (W, V) with $W \in RP^{p\times p}$ and $V \in RP^{p\times m}$, and let this DNF characterize the relationship $Y = WY + VU$ for m inputs U and p outputs Y. Let Y_S be some subset of the outputs of Y indexed by S. A *node abstraction* of (W, V) with respect to S is a new network (W_S, V_S) where W_S is a matrix of (possibly improper) rational polynomials of dimension $|S| \times |S|$, where V_S is a matrix of (possibly improper) rational polynomials of dimension $|S| \times m$, and where the following properties hold:

- *Dynamic Equivalence:* The relationship $Y_S = W_S Y_S + V_S U$ holds such that

$$\begin{bmatrix} Y_S \\ Y_{\bar{S}} \end{bmatrix} = (I - W)^{-1} VU \implies Y_S = (I - W_S)^{-1} V_S U. \tag{3.8}$$

- *Independence-Pattern Preserving:* Let $(\Gamma_V, \Gamma_E, \Gamma_W) = \Gamma(Q, P)$ be the graph of this DNF as defined in Section 3.2.1 and consider the corresponding graph $(\Gamma_{V_S}, \Gamma_{E_S}, \Gamma_{W_S}) = \Gamma(W_S, V_S)$. For every pair $y_i, y_j \in V_S$, we have that $\Gamma_{W_S}(y_i, y_j)$ is precisely the net effect in $\Gamma(W, V)$ fromn y_i to y_j with respect to $\bar{S}$[a].

[a] Due to potential cycles in (W, V), there may be an infinite number of walks between any two nodes passing through subsets of $\bar{S}$. As such, the "independence-pattern preserving" property can only be defined on networks where net effects with respect to $\bar{S}$ are convergent. Convergence is related to the notion of stability; however, the computation of node abstractions need not be dependent on stability as

the matrix inverses utilized to perform the computation (see Section 3.3.2) can exist even when the net effects do not converge. As such, we define the "independence pattern preserving" property for the cases where the requisite sums do indeed converge, but utilize the same node abstraction computation for the non-convergent cases as well, requiring only that the network be representable (see Section 3.3.3).

In summary, dynamic equivalence implies that–where the closed loop transfer function of the original network (W, V) is $G_{cl} = (I - W)^{-1}V$, and assuming that S indexes the first $|S|$ signals in Y–the closed loop transfer function of the abstracted network (W_S, V_S) is precisely the first $|S|$ rows in G_{cl}.

As defined in Section 3.2.1, the independence pattern of a DNF is the direct causal dependence and independence among manifest variables. When converting a network to its node abstraction, the independence pattern should be preserved, meaning that if signals in the base network were directly dependent on each other and are still manifest in the abstraction, then the direct dependence should remain in the abstraction (ignoring cancellation). However, as some signals are no longer manifest, indirect dependencies through now-latent signals may become direct dependencies in the abstracted network. For instance, suppose that, in the base network (W, V), y_k is dependent on y_j and y_j is dependent on y_i, but y_k is independent of y_i (i.e., $W_{ik} = 0$ but $W_{ij} \neq 0$ and $W_{jk} \neq 0$). If we abstract away y_j (meaning $j \notin S$), independence-pattern preserving implies that (barring cancellation) the indirect dependence of y_k on y_i through y_j in (W, V) becomes a direct dependence in (W_S, V_S) (i.e., $[W_S]_{ik} \neq 0$). Thus we preserve the computational dependence and independence pattern across manifest variables in the network.

3.3.2 Computing the Node Abstraction

We introduce a new function, $\mathcal{A} : RP^{p\times p} \times RP^{p\times m} \times S \to RP^{|S|\times|S|} \times RP^{|S|\times m}$, and show that this function maps to a node abstraction as defined by Definition 3.3.1. Furthermore, we show that the resultant network is the *unique* network that satisfies Definition 3.3.1. Let (W, V) be a DNF defining the relationship $Y = WY + VU$, let S index some subset of the signals Y with $\bar{S}$ as the complement of S indexing the remaining signals, and define the node abstraction of (W, V) over S to be $(W_S, V_S) = \mathcal{A}(W, V \mid S)$. To find (W_S, V_S), reorder and

partition $Y = WY + VU$ such that

$$\begin{bmatrix} Y_S \\ Y_{\bar{S}} \end{bmatrix} = \begin{bmatrix} W_{SS} & W_{S\bar{S}} \\ W_{\bar{S}S} & W_{\bar{S}\bar{S}} \end{bmatrix} \begin{bmatrix} Y_S \\ Y_{\bar{S}} \end{bmatrix} + \begin{bmatrix} V_{SS} \\ V_{\bar{S}\bar{S}} \end{bmatrix} U. \tag{3.9}$$

Solving for $Y_{\bar{S}}$, we get

$$Y_{\bar{S}} = (I - W_{\bar{S}\bar{S}})^{-1} W_{\bar{S}S} Y_S + (I - W_{\bar{S}\bar{S}})^{-1} V_{\bar{S}} U. \tag{3.10}$$

Plugging this back into the first row of (3.9) gives the final DNF

$$Y_S = \left[W_{SS} + W_{S\bar{S}}(I - W_{\bar{S}\bar{S}})^{-1} W_{\bar{S}S}\right] Y_S + \left[V_{SS} + W_{S\bar{S}}(I - W_{\bar{S}\bar{S}})^{-1} V_{\bar{S}\bar{S}}\right] U, \tag{3.11}$$

$$\triangleq W_S Y_S + V_S U. \tag{3.12}$$

We now wish to show that the function $\mathcal{A}$ computes the unique node abstraction as defined by Definition 3.3.1.

Theorem 3.3.2: *Let (W, V) be a DNF defining the relationship $Y = WY + VU$, let S index some subset of the signals Y with $\bar{S}$ as the complement of S indexing the remaining signals. Let $\mathcal{A}$ be a function such that $(W_S, V_S) = \mathcal{A}(W, V \mid S)$, where W and V are partitioned according to S (possibly after reordering) as given in (3.9). such that*

$$W = \begin{bmatrix} W_{SS} & W_{S\bar{S}} \\ W_{\bar{S}S} & W_{\bar{S}\bar{S}} \end{bmatrix}, \qquad P = \begin{bmatrix} P_{SS} \\ P_{\bar{S}\bar{S}} \end{bmatrix}. \tag{3.13}$$

Finally, suppose net effect of walks with respect to $\bar{S}$ is convergent[5].

[5]This convergence property is required to demonstrate the independence-pattern preserving property and uniqueness; it is not required to show dynamic equivalence. Furthermore, it is a sufficient condition and not necessary.

If (W, V) is well-posed and $(I - W_{\bar{S}\bar{S}})$ has an inverse[6] *and $\mathcal{A}(W, V \mid S)$ computes (W_S, V_S) according to (3.12), then $(W_S, V_S) = \mathcal{A}(W, V \mid S)$ is a node abstraction. Furthermore, (W_S, V_S) is the only DNF with respect to S that satisfies Definition 3.3.1.* ♢

PROOF We show that $\mathcal{A}$ is dynamically equivalent to the original network. Since (W, V) is well-posed, $(I - W)$ has a (proper) inverse by Theorem 2.5.8. Furthermore, we have assumed that $(I - W_{\bar{S}\bar{S}})$ is invertible. Thus, by block matrix inversion,

$$\begin{bmatrix} Y_S \\ Y_{\bar{S}} \end{bmatrix} = \begin{bmatrix} I - W_{SS} & -W_{S\bar{S}} \\ -W_{\bar{S}S} & I - W_{\bar{S}\bar{S}} \end{bmatrix}^{-1} \begin{bmatrix} V_{SS} \\ V_{\bar{S}\bar{S}} \end{bmatrix} U = \begin{bmatrix} N_{11} & N_{12} \\ N_{21} & N_{22} \end{bmatrix} \begin{bmatrix} V_{SS} \\ V_{\bar{S}\bar{S}} \end{bmatrix} U,$$

where

$$\begin{aligned} N_{11} &= \left[I - (W_{SS} + W_{S\bar{S}}(I - W_{\bar{S}\bar{S}})^{-1}W_{\bar{S}S})\right]^{-1} = (I - W_S)^{-1}, \\ N_{12} &= -N_{11}W_{S\bar{S}}(I - W_{\bar{S}\bar{S}})^{-1} = (I - W_S)^{-1}W_{\bar{S}S}(I - W_{\bar{S}\bar{S}})^{-1}. \end{aligned}$$

Then

$$\begin{aligned} Y_S &= [N_{11}V_{SS} + N_{12}V_{\bar{S}\bar{S}}]\,U = \left[(I - W_S)^{-1}V_{SS} + (I - W_S)^{-1}V_{\bar{S}S}(I - W_{\bar{S}\bar{S}})^{-1}V_{\bar{S}\bar{S}}\right]U \\ &= (I - W_S)^{-1}\left[V_{SS} + W_{S\bar{S}}(I - W_{\bar{S}\bar{S}})^{-1}V_{\bar{S}\bar{S}}\right]U = (I - W_S)^{-1}V_S U, \end{aligned}$$

as desired.

Now, we show that $\mathcal{A}$ preserves the independence pattern. Consider the sub-graph $(\Gamma_{V_{\bar{S}}}, \Gamma_{E_{\bar{S}}}, \Gamma_{W_{\bar{S}}})$ of (W, V) indexed by $\bar{S}$; i.e., the sub-graph consisting only of vertices in $\bar{S}$ and all edges in W connecting those vertices. Let $y_{\bar{i}}, y_{\bar{j}} \in \Gamma_{V_{\bar{S}}}$ be any pair of vertices in this sub-graph (we allow $y_{\bar{i}} = y_{\bar{j}}$). Then $[W_{\bar{S}\bar{S}}]_{\bar{j}\bar{i}}$ is the net effect of the walk from $y_{\bar{i}}$ to $y_{\bar{j}}$ containing one edge passing through vertices indexed by $\bar{S}$ only. Also $\sum_{\bar{k}\in\bar{S}} [W_{\bar{S}\bar{S}}]_{\bar{j}\bar{k}} [W_{\bar{S}\bar{S}}]_{\bar{k}\bar{i}} = \left[W^2_{\bar{S}\bar{S}}\right]_{\bar{j}\bar{i}}$ is the net effect of all walks from $y_{\bar{i}}$ to $y_{\bar{j}}$ containing exactly two edges passing through

[6] We defer the discussion of invertibility of $(I - W_{\bar{S}\bar{S}})$ to Sections 3.3.3 and 3.3.4.

vertices indexed by $\bar{S}$ only. Similarly, $\left[W^k_{\bar{S}\bar{S}}\right]_{\bar{j}\bar{i}}$ is the net effect of all walks from $y_{\bar{i}}$ to $y_{\bar{j}}$ containing exactly k edges passing through vertices indexed by $\bar{S}$ only, meaning that $\left[\sum_{k=0}^{\infty} W^k_{\bar{S}\bar{S}}\right]_{\bar{j}\bar{i}}$ is the net effect of all walks from $y_{\bar{i}}$ to $y_{\bar{j}}$ passing through vertices indexed by $\bar{S}$ only (including the walk containing 0 edges which is of net effect one when $\bar{i} = \bar{j}$ and net effect 0 otherwise). And by Lemma 3.8.3 and since the sum is convergent by assumption, $[\sum_{k=0}^{\infty} W^k_{\bar{S}\bar{S}}]_{\bar{j}\bar{i}} = [(I - W_{\bar{S}\bar{S}})^{-1}]_{\bar{j}\bar{i}}$.

Now consider the full graph $\Gamma(W, V)$. Since, for any $i, j \in S$ (where, again, we allow $i = j$),

$$\left[W_{S\bar{S}}(I - W_{\bar{S}\bar{S}})^{-1}W_{\bar{S}S}\right]_{ji} = \sum_{\bar{j}\in\bar{S}}\sum_{\bar{i}\in\bar{S}} [W_{S\bar{S}}]_{j\bar{j}} \left[(I - W_{\bar{S}\bar{S}})^{-1}\right]_{\bar{j}\bar{i}} [W_{\bar{S}S}]_{\bar{i}i}\,, \tag{3.14}$$

we have that $[W_{S\bar{S}}(I - W_{\bar{S}\bar{S}})^{-1}W_{\bar{S}S}]_{ji}$ is the net effect of all walks from y_i to y_j passing only through nodes indexed by $\bar{S}$, and $[W_{SS} + W_{S\bar{S}}(I - W_{\bar{S}\bar{S}})^{-1}W_{\bar{S}S}]_{ji}$ is the net effect of walks from y_i to y_j passing through nodes indexed by every subset of $\bar{S}$ (including the empty set); thus $\mathcal{A}$ is independence pattern preserving.

Finally, we show that $(W_S, V_S) = \mathcal{A}(W, V \mid S)$ is the unique DNF that is both a dynamically equivalent reduction and an independence-pattern-preserving reduction. This property follows immediately from the fact that independence-pattern-preserving reductions specify precisely the values of W_S and V_S, thus (W_S, V_S) must be unique. ∎

3.3.3 Representable Node Abstractions

For the purposes of this work, we desire all DNFs (W, V) to consist of matrices of rational polynomials so that they can be interpreted as reasonable approximations of networks of causal systems. We wish for any abstraction (including node abstractions) to preserve this property. Thus, we define *representable* abstractions as follows:

Definition 3.3.3: Representable Abstractions

An abstraction (W_S, V_S) of some DNF (W, V) (whether node, edge, or some other type) is said to be a *representable* abstraction of (W, V) if:

(R1) The abstracted network (W_S, V_S) exists, is unique, and is proper.

(R2) If (R1) holds for the abstraction of some network (W, V), then small perturbations to (W, V) likewise results in a proper abstracted network.

Condition (R1) in Definition 3.3.3 parallels conditions (W1-W3) for the definition of well-posedness of DNFS presented in [16], and condition (R2) parallels condition (W4) in this same definition of well-posedness (see Section 3.2.2 of this work for conditions (W1-W4)); thus we refer the reader to that work for a full discussion and justification of these conditions. In short, however, condition (W4)–and hence, (R2)–exists because we wish our DNF to remain a reasonable approximation of reality even in the presence of modeling error.

As a result, representability and well-posedness are related concepts with subtle differences. In short, *well-posedness* of some DNF (W, V) is about the generic existence, uniqueness, and properness of the closed-loop transfer function $G_{cl} = (I - W)^{-1}V$. In contrast, *representability* of the same DNF is about the generic existence, uniqueness, and properness of the matrices W and V.

With this definition, we are prepared to give the necessary and sufficient conditions for the representability of a node abstraction.

Theorem 3.3.4: *Suppose (W, V) is a proper DNF defining the relationship $Y = WY + VU$. Let S index some subset of the signals Y with $\bar{S}$ as the complement of S indexing the remaining signals, and let $(W_S, V_S) = \mathcal{A}(W, V \mid S)$ be the node abstraction of (W, V) with respect to S, where W and V are partitioned according to S (possibly after reordering) according to (3.9). Then (W_S, V_S) is a representable abstraction of (W, V) (i) if and (ii) only if $(I - W_{\bar{S}\bar{S}})$ has a proper inverse.* ♢

PROOF (i) Suppose that $(I - W_{\bar{S}\bar{S}})$ has a proper inverse. Then, by Lemma 3.8.1 and Lemma 3.8.2, $W_S = W_{SS} + W_{S\bar{S}}(I - W_{\bar{S}\bar{S}})^{-1}W_{\bar{S}S}$ and $V_S = V_{SS} + W_{S\bar{S}}(I - W_{\bar{S}\bar{S}})^{-1}V_{\bar{S}\bar{S}}$ are both proper. Furthermore, due to the density of invertible matrices, small perturbations in (W, V) will not change this (see the proof for Proposition 4 in [16] for more details).

(ii) Suppose that $(I - W_{\bar{S}\bar{S}})$ does not have a proper inverse. Then either (a) (W_S, V_S) does not exist or is not unique, or (b) as was done in Proposition 4 in [16], we can always design a perturbation on (W, V) of arbitrary size such that one or both of W_S and V_S are improper. ■

Corollary 3.3.5: *Let (W, V) be a proper DNF defining the relationship $Y = WY + VU$ and let S_Y index all of the signals in Y. Consider the set of all possible node abstractions of some proper DNF (W, V); i.e. the set $A = \{\mathcal{A}(W, V \mid S) : S \subseteq S_Y\}$ (note that A includes (W, V)). Then every abstraction in A is representable (i) if and (ii) only if every principle submatrix of $(I - W)$ has a proper inverse (or equivalently, if and only if every principle submatrix of $(I - W(\infty))$ is invertible).* ♢

PROOF (i) Suppose that every principle submatrix of $(I - W)$ has a proper inverse. Consider an arbitrary $S \subseteq S_Y, S \neq \emptyset$. Then, with $\bar{S} = S_Y - S$, we have that $(I - W_{\bar{S}\bar{S}})$ is a principle submatrix of $(I - W)$, and, by assumption, has a proper inverse. Thus, by Theorem 3.3.4, $\mathcal{A}(W, V \mid S)$ is representable. In the case where $S = \emptyset$, the properness (given by assumption) of (W, V) is equivalent to its representability.

(ii) Suppose that some principle submatrix of $(I - W)$ does not have a proper inverse. Let this principle submatrix be $(I - W_{\bar{S}\bar{S}})$ indexed by the set $\bar{S}$. Then, by Theorem 3.3.4, we have that $\mathcal{A}(W, V \mid S)$ is not representable, where $S = S_Y - \bar{S}$. ■

3.3.4 Well-Posedness of Node Abstractions

We now wish to provide necessary and sufficient conditions for when abstractions and realizations of DNF are well-posed. These conditions are contained in the following result:

Theorem 3.3.6: *Suppose (W, V) is a proper DNF defining the relationship $Y = WY + VU$. Let $(W_S, V_S) = \mathcal{A}(W, V \mid S)$ be a representable node abstraction of (W, V) with respect to some S indexing a subset of the output signals Y. Then (W_S, V_S) is well-posed (i) if and (ii) only if (W, V) is well-posed.* ♢

PROOF Permute and partition $(I - W)$ commensurate with S as given in (3.9). Now assume that (W_S, V_S) is a representable abstraction of (W, V). Then $(I - W_{\bar{S}\bar{S}})$ has a proper inverse by Theorem 3.3.4. Also notice that $(I - W_S) = (I - W_{SS}) - W_{S\bar{S}}(I - W_{\bar{S}\bar{S}})^{-1}W_{\bar{S}S}$ is precisely $(I - W)/(I - W_{\bar{S}\bar{S}})$ (i.e. $(I - W_S)$ is the Schur Complement of $(I - W_{\bar{S}\bar{S}})$ in $(I - W)$).

(i) Assume that (W, V) is well-posed. Then, by Theorem 2.5.8, $(I - W)$ has a proper inverse. Thus, since both $(I - W)$ and $(I - W_{\bar{S}\bar{S}})$ have proper inverses, $(I - W_S)$ also has a proper inverse by Lemma 3.8.6; hence (W_S, V_S) is also well-posed by Theorem 2.5.8.

(ii) Assume that (W_S, V_S) is well-posed. Then, by Theorem 2.5.8, $(I - W_S)$ has a proper inverse. Thus, since both $(I - W_S)$ and $(I - W_{\bar{S}\bar{S}})$ have proper inverses, $(I - W)$ also has a proper inverse by Lemma 3.8.6; hence (W, V) is well-posed by Theorem 2.5.8. ■

REMARK 3.3.7: The work [15] defines the concept *strong well-posedness* and gives the necessary and sufficient conditions for an abstraction to be strongly well-posed. In the language of this work, an abstraction is strongly well-posed if it is both representable and well-posed, and the conditions given for strong well-posedness in Theorems 3 and 4 of that work are conditions of representability of the immersion, which is a hollow abstraction (which we define in Section 3.4) of a node abstraction; thus those results are consistent with Theorem 3.3.6 above. However, for clarity, this work separates the notions of representability and well-posedness, as well as the notions of node abstractions and hollow abstractions.

We now give a result pertaining the well-posedness of *every* abstraction of some DNF.

Corollary 3.3.8: *Suppose (W, V) is a proper DNF defining the relationship $Y = WY + VU$. Then (W, V) and every abstraction of (W, V) is representable and well-posed if and only if every principle submatrix of (W, V) has a proper inverse.* ♢

PROOF Follows immediately from Theorem 3.3.6 applied to Corollary 3.3.5. ■

Corollary 3.3.8 is important since the existence of proper inverses for every principle submatrix of W is occasionally used as the conditions for well-posedness (see, for instance [46, 47]; see also the definition for *total well-posedness* in [15]). However, the insight we gain from Theorem 3.3.4 and Theorem 3.3.6 is that these conditions are not conditions of well-posedness, but conditions of representability. If we can guarantee the representability of every abstraction, then we can guarantee the well-posedness of every abstraction as well.

3.3.5 Sequential Node Abstractions

Node abstractions are very similar to the notion of isospectral reductions presented in [48]. Consequentially, we can leverage an important result there to understand node abstractions better. Specifically, when we sequentially apply abstractions to a DNF, the order in which we abstract away outputs does not matter; all that matters is the set of outputs remaining at the end. We begin with a definition:

Definition 3.3.9: Sequential Node Abstraction

Suppose that we wish to find a node abstraction of a given proper DNF (W, V), and then we wish to find a node abstraction of the resultant DNF, and so on. We call this iterative abstraction process *sequential abstraction*, and for notational simplicity, we define *sequential node abstraction* (W_S, V_S) such that

$$(W_S, V_S) = \mathcal{A}(W, V \mid S_1, S_2, \cdots S_a) \triangleq$$
$$\mathcal{A}(\mathcal{A}(\mathcal{A}(W, V \mid S_1) \mid S_2) \cdots \mid S_a),$$

where, with S_Y indexing all of Y and for some integer $a > 0$, we have $S_a \subseteq \ldots \subseteq S_2 \subseteq S_1 \subseteq S_Y$ with $S_a \neq \emptyset$.

Equipped with this definition, we are now able to provide a lemma and then the main result of this section, which closely follows Lemma 1.2 and Theorem 1.3 in [48], though with some minor variations.

Lemma 3.3.10: *Let (W, V) be an arbitrary proper DNF defining the relationship $Y = WY + VU$ and with $W \in RP^{p\times p}$ and $V \in RP^{p\times m}$. Let S_Y index all of Y and let $S, T \subseteq S_Y$ with $S \cap T = \emptyset$ and suppose that $\mathcal{A}(W, V \mid S \cup T)$ and $\mathcal{A}(W, V \mid S)$ are representable. Then*

$$\mathcal{A}(W, V \mid S \cup T, S) = \mathcal{A}(W, V \mid S). \tag{3.15}$$

♢

Proof Let $N = S_Y - (S \cup T)$. Then we can permute and partition Q and P such that

$$W = \begin{bmatrix} W_{SS} & W_{ST} & W_{SN} \\ W_{TS} & W_{TT} & W_{TN} \\ W_{NS} & W_{NT} & W_{NN} \end{bmatrix}, \quad V = \begin{bmatrix} V_{SS} \\ V_{TT} \\ V_{NN} \end{bmatrix}. \tag{3.16}$$

Then, by definition, we have that

$$\mathcal{A}(W, V \mid S) = (W_S, V_S) \tag{3.17}$$

with

$$W_S = W_{SS} + \begin{bmatrix} W_{ST} & W_{SN} \end{bmatrix} \begin{bmatrix} I - W_{TT} & -W_{TN} \\ -W_{NT} & I - W_{NN} \end{bmatrix}^{-1} \begin{bmatrix} W_{TS} \\ W_{NS} \end{bmatrix}, \tag{3.18}$$

$$V_S = V_{SS} + \begin{bmatrix} W_{ST} & W_{SN} \end{bmatrix} \begin{bmatrix} I - W_{TT} & -W_{TN} \\ -W_{NT} & I - W_{NN} \end{bmatrix}^{-1} \begin{bmatrix} V_{TT} \\ V_{NN} \end{bmatrix}, \tag{3.19}$$

which exist and are proper since we have assumed that $\mathcal{A}(W, V \mid S)$ is representable.

Similarly, we have that

$$\mathcal{A}(W, V \mid S \cup T) = (W_T, V_T) \tag{3.20}$$

with

$$W_T = \begin{bmatrix} W_{SS} & W_{ST} \\ W_{TS} & W_{TT} \end{bmatrix} + \begin{bmatrix} W_{SN} \\ W_{TN} \end{bmatrix} (I - W_{NN})^{-1} \begin{bmatrix} W_{NS} & W_{NT} \end{bmatrix}, \tag{3.21}$$

$$V_T = \begin{bmatrix} V_{SS} \\ V_{TT} \end{bmatrix} + \begin{bmatrix} W_{SN} \\ W_{TN} \end{bmatrix} (I - W_{NN})^{-1} V_{NN}, \tag{3.22}$$

which also exist and are proper since we have assumed that $\mathcal{A}(Q, P \mid S \cup T)$ is representable.

Defining

$$K = (I - W_{NN}), \tag{3.23}$$

$$L = (I - W_{TT}) - \left(W_{TN}(I - W_{NN})^{-1} W_{NT}\right), \tag{3.24}$$

we find that (noting K has an inverse since $\mathcal{A}(W, V \mid S \cup T)$ is representable and that L has an inverse by Lemma 3.8.6 and since both $\mathcal{A}(W, V \mid S)$ and $\mathcal{A}(W, V \mid S \cup T)$ are representable):

$$\mathcal{A}(W, V \mid S \cup T, S) = \mathcal{A}(W_{ST}, W_{ST} \mid S) = (W_{STS}, V_{STS}), \tag{3.25}$$

where

$$W_{STS} = W_{SS} + W_{SN}K^{-1}W_{NS} + \left[(W_{ST} + W_{SN}K^{-1}W_{NT})L^{-1}(W_{TS} + W_{TN}K^{-1}W_{NS})\right], \tag{3.26}$$

$$V_{STS} = V_{SS} + W_{SV}K^{-1}V_{NN} + \left[(W_{ST} + W_{SN}K^{-1}Q_{NT})L^{-1}(V_{TT} + W_{TN}K^{-1}V_{NN})\right]. \tag{3.27}$$

Using block matrix inverses, we can compute the block matrix inverse in the equations for Q_S and P_S (noting that L as defined above is precisely the Schur Complement of $(I - Q_{NN})$ in this matrix) as

$$\begin{bmatrix} I - W_{TT} & -W_{TN} \\ -W_{NT} & I - W_{NN} \end{bmatrix}^{-1} = \begin{bmatrix} L^{-1} & L^{-1}Q_{TN}K^{-1} \\ K^{-1}W_{NT}L^{-1} & K^{-1} + K^{-1}Q_{NT}L^{-1}W_{TN}K^{-1} \end{bmatrix}. \tag{3.28}$$

Plugging this back in to the equations for W_S and W_S, we find that $W_S = W_{STS}$ and $V_S = V_{STS}$, thus completing the proof. ∎

Theorem 3.3.11 (Uniqueness of Sequential Node Abstractions): *Let (W, V) be an arbitrary DNF defining the relationship $Y = WY + VU$ and with $W \in RP^{p \times p}$ and $V \in RP^{p \times m}$. Let S_Y index all of Y and let, for some integer $a > 0$, $S_a \subseteq \ldots \subseteq S_2 \subseteq S_1 \subseteq S_Y$ with $S_a \neq \emptyset$. Then (assuming that every abstraction in the sequence below is representable),*

$$\mathcal{A}(W, V \mid S_1, S_2, \cdots, S_a) = \mathcal{A}(W, V \mid S_a), \tag{3.29}$$

that is, a sequential node abstraction is a node abstraction that is completely specified by the final index set. ♢

Proof We show this by induction. If $a = 2$, then by Lemma 3.3.10, we have that $\mathcal{A}(W, V \mid S_1, S_2) = \mathcal{A}(W, V \mid S_2)$. Now, for $2 \leq k < a$, suppose that $\mathcal{A}(W, V \mid S_1, \cdots, S_k) = \mathcal{A}(W, V \mid S_k)$. Then, by Lemma 3.3.10, we have that

$$\begin{aligned} \mathcal{A}(W, V \mid S_1, \cdots, S_k, S_{k+1}) &= \mathcal{A}(W, V \mid S_k, S_{k+1}) \\ &= \mathcal{A}(W, V \mid S_{k+1}). \end{aligned}$$

Thus, by induction, we have that $\mathcal{A}(W, V \mid S_1, S_2, \cdots, S_a) = \mathcal{A}(W, V \mid S_a)$, as desired. ∎

To see the interpretation of Theorem 3.3.11, consider an arbitrary proper DNF (W, V) defining the relationship $Y = WY + VU$. Suppose Y is indexed by S_Y and consider three sets

S, T, and N such that $S \cap T = \emptyset$ and $N = S_Y - (S \cup T)$. Ultimately, we wish to abstract away everything in T and N, leaving only the outputs indexed by S manifest. The implication of Theorem 3.3.11 is that it does not matter the order in which we perform the node abstraction– we could abstract away T and then N in sequence (represented by $\mathcal{A}(W, V \mid S \cup N, S)$), we could abstract away N and then T in sequence (represented by $\mathcal{A}(W, V \mid S \cup T, S)$), or we could abstract away T and N together (represented by $\mathcal{A}(W, V \mid S)$)–and we would always end with the same DNF (W_S, V_S).

3.3.6 Identifiability Index

Recall from Section (3.2.3) that the identifiability index to reconstruct some arbitrary DNF (W, V) (with $W \in RP^{p \times p}$ and $V \in RP^{p \times n}$) is p^2. Let $(W_S, V_S) = \mathcal{A}(W, V \mid S)$ be a node abstraction of this DNF with $|S| < p$. We have that $W_S \in RP^{|S| \times |S|}$, thus the identifiability index to reconstruct the abstraction is $|S|^2 < p^2$, meaning the abstraction requires a lower identifiability index than the base network.

3.4 Edge and Hollow Abstractions

We now introduce another type of abstraction, which we call the *edge abstraction*. In contrast to the node abstraction, which removes nodes from the signal structure of some dynamic network, the edge abstraction removes edges. We define the edge abstraction as follows:

Definition 3.4.1: Edge Abstraction

Let (W, V) a DNF with $W \in RP^{p \times p}$ and $V \in RP^{p \times m}$. Also let E be a set consisting of some subset of edges in W and V. An *edge abstraction* with respect to E is another DNF (W_E, V_E) where the following properties hold:

- *Shape Preserving:* We have $W_E \in RP^{p \times p}$ and $V_E \in RP^{p \times m}$.

- *Edge Abstracting:* For each $[W_E]_{ij}, [V_E]_{ij} \in E$, we have $[W_E]_{ij} = 0$ and $[V_E]_{ij} = 0$.
- *Edge Preserving:* For each $[W_E]_{ij}, [V_E]_{ij} \notin E$, we have $[W_E]_{ij} = 0 \iff [W]_{ij} = 0$ and $[V_E]_{ij} = 0 \iff [V]_{ij} = 0$.
- *Dynamic Equivalence:* We have $G_{cl} = (I - W)^{-1}V = (I - W_E)^{-1}V_E$.

We write the function $\mathcal{E}$ mapping a DNF (W, V) to its edge abstraction (W_E, V_E) over E as $(W_E, V_E) = \mathcal{E}(W, V \mid E)$.

In other words, an edge abstraction is another DNF with the same shape and input-output behavior and where some links have been forced to zero while the remaining Boolean structure of the network is preserved.

While there may be many types of edge abstractions, we are primarily concerned with one which we call the *hollow abstraction*, defined as follows:

Definition 3.4.2: Hollow Abstraction

A *hollow abstraction* is an edge abstraction according to Definition 3.4.1 where $E = \{[W]_{ii} \mid i = 1, \ldots, p\}$. We write $(W_H, V_H) = \mathcal{H}(W, V) = \mathcal{E}(W, V \mid E)$.

In other words, a hollow abstraction is an edge abstraction where we have forced all the diagonal entries of W to be zero (i.e., we have forced W_H to be hollow).

3.4.1 Computing the Hollow Abstraction

Let $\mathcal{H}$ be a function defining our hollow abstraction; i.e., let $(W_H, V_H) = \mathcal{H}(W, V)$. To define this function, suppose our DNF (W, V) defines the relationship $Y = WY + VU$. Let $D_W = \operatorname{diag} W$ and subtract $D_W Y$ from both sides of the equation. Rearranging, we get

$$Y = (I - D_W)^{-1}(W - D_W)Y + (I - D_W)^{-1}VU \tag{3.30}$$

$$\triangleq W_E Y + V_E U. \tag{3.31}$$

Theorem 3.4.3: *Suppose* (W, V) *is a DNF with* $W \in RP^{p\times p}$ *and* $V \in RP^{p\times m}$. *Let* $(W_E, V_E) = \mathcal{H}(W, V)$, *where* (W_H, V_H) *is found according to (3.31). Then* (W_H, V_H) *is a hollow abstraction as defined in Definition 3.4.2.* ♢

Proof Since W and D_W are square, W_H and V_H will have the same shape as W and V respectively, thus (3.31) is shape preserving. Also, since $D_W \triangleq \operatorname{diag} W$, $W - D_W$ is zero along the diagonal and has the same zero structure as W on the off-diagonal. Furthermore, $(I - D_W)^{-1}$ is a diagonal matrix; thus, $[(I - D_W)^{-1}(W - D_W)]_{ij} = 0 \iff [W - D_W]_{ij} = 0$ and (3.31) is both edge abstracting and edge preserving since E contains precisely the diagonal elements of W. Also, we have

$$\begin{aligned}(I - W_H)^{-1}V_H &= \left[I - (I - D_W)^{-1}(W - D_W)\right]^{-1}(I - D_W)^{-1}V \\ &= \left[(I - D_W)^{-1}\left[(I - D_W) - (W - D_W)\right]\right]^{-1}(I - D_W)^{-1}V \\ &= (I - W)^{-1}V,\end{aligned}$$

thus (3.31) is dynamically equivalent to the original DNF. Since all four properties of an edge abstraction hold, (3.31) is an edge abstraction according to Definition 3.4.1 and hence a hollow abstraction according to Definition 3.4.2. ■

3.4.2 Importance of the Hollow Abstraction

As shown in the following result, the hollow abstraction is fundamental in relating the DNF to the DSF. Indeed, we show that the DSF is precisely the hollow abstraction of some DNF.

Proposition 3.4.4: *The DSF is a hollow abstraction of the DNF* ♢

Proof From [1], we have that, given $\tilde{W} \in RP^{p\times p}$ and $\tilde{V} \in RP^{p\times m}$ defined over a state space model[7], and given $D_{\tilde{W}} \triangleq \operatorname{diag}(\tilde{W})$, the DSF is given by (Q, P), where $Q = (sI - D_{\tilde{W}})^{-1}(\tilde{W} - D_{\tilde{W}})$ and $P = (sI - D_{\tilde{W}})^{-1}\tilde{V}$. From [16], we have that, given the same $\tilde{W}$ and

[7] In [1], $\tilde{W}$ and $\tilde{V}$ are labeled W and V respectively. In this work, we have added the tilde in order to distinguish those matrices from the W and V defining the DNF.

$\tilde{V}$ as used in the computation of the DSF (e.g., obtained by starting with the same state space model), the DNF is given by (W, V), where $W = \frac{1}{s}\tilde{W}$ and $V = \frac{1}{s}\tilde{V}$.

We now show that $(Q, P) = (W_H, V_H) = \mathcal{H}(W, V)$, with $\mathcal{H}$ as defined above. Let $D_W = \operatorname{diag} W$. Thus $D_W = \frac{1}{s}\operatorname{diag}(\tilde{W}) = \frac{1}{s}D_{\tilde{W}}$. Thus we have, from the definition of $\mathcal{H}$ above that (W_H, V_H) are given as follows:

$$W_H = \left(I - \frac{1}{s}D_{\tilde{W}}\right)^{-1}\left(\frac{1}{s}\tilde{W} - \frac{1}{s}D_{\tilde{W}}\right) = (sI - D_{\tilde{W}})^{-1}(\tilde{W} - D_{\tilde{W}}) = Q,$$
$$V_H = \left(I - \frac{1}{s}D_{\tilde{W}}\right)^{-1}\frac{1}{s}\tilde{V} = (sI - D_{\tilde{W}})^{-1}\tilde{V} = P,$$

as desired. Thus the hollow abstraction of a DNF is a DSF. ∎

3.4.3 Representability of the Hollow Abstraction

As with the node abstraction, we wish to characterize when an edge abstraction (and in particular, the hollow abstraction) is representable. We continue to use Definition 3.3.3 to define representability; i.e., an edge abstraction or a hollow abstraction (W_H, V_H) is representable if W_H and V_H are both matrices of proper rational polynomials. With this definition, we are prepared to give conditions on the representability of the hollow abstraction, which follows Theorem 3.3.4 closely in both statement and proof.

Theorem 3.4.5: *Suppose (W, V) is a DNF with $W \in RP^{p\times p}$ and $V \in RP^{p\times m}$, and let $D_W = \operatorname{diag} W$. Then $(W_H, V_H) = \mathcal{H}(W, V)$ is representable (i) if and (ii) only if $(I - D_W)$ has a proper inverse.* ♢

Proof (i) Suppose that $(I - D_W)$ has a proper inverse. Then, by Lemma 3.8.1 and Lemma 3.8.2, $W_H = (I - D_W)^{-1}(W - D_W)$ and $V_H = (I - D_W)^{-1}V$ proper. Furthermore, due to the density of invertible matrices, small perturbations in (W, V) will not change this (see the proof for Proposition 4 in [16] for more details).

(ii) Suppose that $(I - D_W)$ does not have a proper inverse. Then either (a) (W_H, V_H) does not exist or is not unique, or (b), as was done in Proposition 4 in [16], we can always

design a perturbation of arbitrary size on (W, V) such that one or both of W_H and V_H are improper. ■

In Corollary 3.3.5, we showed that every possible node abstraction of the DNF (W, V) is representable if and only if every principle submatrix of $(I - W)$ has a proper inverse. We wish to provide a similar result over the hollow abstraction. We note that there is only one possible hollow abstraction of any (W, V), and as we show in the first result below, having proper inverses for every principle submatrix of $(I - W)$ is a sufficient condition for representability.

Corollary 3.4.6: *Let (W, V) be a DNF with $W \in RP^{p\times p}$ and $V \in RP^{p\times m}$ and suppose that every principle submatrix of $(I - W)$ has a proper inverse. Then $(W_H, V_H) = \mathcal{H}(W, V)$ is representable.* ♢

Proof Suppose that every principle submatrix of $(I - W)$ has a proper inverse. Then, since $(1 - W_{ii})$ is a principle submatrix for $i = 1, \ldots, p$, $(1 - W_{ii})$ has a proper inverse. Equivalently by Lemma 3.8.4, $(1 - W_{ii}(\infty))$ is invertible, meaning $W_{ii}(\infty) \neq 1$. Hence we have that $(I - D_W(\infty)) = \text{diag}(1 - W_{11}(\infty), \ldots, 1 - W_{pp}(\infty))$ is invertible (since it is a diagonal matrix with non-zero values on every diagonal entry). Thus $(I - D_W)$ has a proper inverse by Lemma 3.8.4, and by Theorem 3.4.5, (W_H, V_H) is representable. ■

3.4.4 Well-Posedness of the Hollow Abstraction

We also wish to provide conditions on the well-posedness of the hollow abstraction.

Theorem 3.4.7: *Suppose (W, V) is a proper DNF. Let $(W_H, V_H) = \mathcal{H}(W, V)$ be a representable hollow abstraction of (W, V). Then (W_H, V_H) is well-posed (i) if and (ii) only if (W, V) is well-posed.* ♢

Proof Let (W_H, V_H) be a representable hollow abstraction of (W, V). Then, with $D_W = \text{diag}\, W$ and by Theorem 3.4.5, we have that $(I - D_W)$ has a proper inverse.

(i) Suppose (W, V) is well-posed. Then $(I - W)$ has a proper inverse by Theorem 2.5.8. Thus we have

$$\begin{aligned}(I - W_H)^{-1} &= (I - (I - D_W)^{-1}(W - D_W))^{-1} \\ &= \left[(I - D_W)^{-1}(I - W)\right]^{-1} \\ &= (I - W)^{-1}(I - D_W), \end{aligned} \tag{3.32}$$

and since $(I-W)^{-1}$ and $(I-D_W)$ are both proper by assumption, by Lemma 3.8.2 $(I-W_H)^{-1}$ must also be proper. Thus, by Theorem 2.5.8, (W_H, V_H) is well-posed.

(ii) Suppose (W_H, V_H) is well-posed. Then $(I - W_H)$ has a proper inverse by Theorem 2.5.8. Rearranging (3.32), we get that

$$(I - W)^{-1} = (I - W_H)^{-1}(I - D_W)^{-1}. \tag{3.33}$$

And since $(I - W_H)^{-1}$ and $(I - D_W)^{-1}$ are both proper, by Lemma 3.8.2, $(I - W)^{-1}$ is also proper. Thus, by Theorem 2.5.8, (W, V) is well-posed. ∎

As shown in Section 3.4.2, the DSF is a hollow abstraction of the DNF. Thus, we have that, assuming that the DSF is representable, it is well-posed if and only if the corresponding DNF is also well-posed. This is a restatement and generalization of the results on the well-posedness of abstractions contained in both [14, 15], where the additional conditions for *strong well-posedness* contained in those works are precisely the conditions required to ensure that the abstracted network is representable.

3.4.5 Identifiability Index

Recall from Section (3.2.3) that the identifiability index for the reconstruction of some DNF (W, V), $W \in RP^{p\times p}$ and $V \in RP^{p\times m}$, is the number of columns in the matrix L that need to be reduce in order to recover W and V uniquely. Since each column corresponds to a single

entry in either W and V, and an edge abstraction forces some subset E of these entries to zero, then the identifiability index are reduced by $|E|$. In particular, the identifiability index to reconstruct (W, V) is p^2, thus the identifiability index to recover $(W_E, V_E) = \mathcal{E}(W, V \mid E)$ is $p^2 - |E| < p^2$.

This is of particular interest for the hollow abstraction. We have that $(Q, P) = \mathcal{H}(W, V) = \mathcal{E}(W, V \mid E)$, where $|E| = p$ contains the diagonal entries of W, is a DSF. Thus, the identifiability index to recover a DSF (as first shown in [19]) is $p^2 - p < p^2$.

3.5 Immersions of Dynamical Structure Functions

We now turn our attention to DSFs (as opposed to DNFs) with the objective of defining abstractions over DSFs. In particular, we will define an *immersion* (from [34, 35]) as the analogy of the node abstraction over DSFs.

In general, if we find a node abstraction (as outlined in Section 3.3) of some DSF (Q, P), the abstracted network (W_S, V_S) will be a DNF instead of a DSF (meaning W_S is not hollow as required by the DSF). However, as shown in Section 3.4.2, if we find the hollow abstraction (Q_S, P_S) of (W_S, V_S), then (Q_S, P_S) will be a DSF.

Definition 3.5.1: Immersion

Let (Q, P) be some proper DSF characterizing the relationship $Y = QY + PU$, and let S index some subset of the outputs Y. Let $(W_S, V_S) = \mathcal{A}(Q, P \mid S)$ be the node abstraction of (Q, P) with respect to S, and let $(Q_S, P_S) = \mathcal{H}(W_S, V_S)$. Then we call (Q_S, P_S) the *immersion* of DSF (Q, P) with respect to S. For simplicity, we define $\mathcal{I} = \mathcal{H} \circ \mathcal{A}$, meaning that $\mathcal{I}(Q, P \mid S) = \mathcal{H}(\mathcal{A}(Q, P \mid S)$.

Since an immersion is found from the composition of a node abstraction process with the hollow abstraction process, it inherits naturally the representability and well-posedness results that we have presented previously. We summarize those here.

Theorem 3.5.2: *Let (Q, P) be some proper DSF characterizing the relationship $Y = QY + PU$, and let S index some subset of the outputs Y. Let $(W_S, V_S) = \mathcal{A}(Q, P \mid S)$ be the node abstraction of (Q, P) with respect to S, let $(Q_S, P_S) = \mathcal{H}(W_S, V_S) = \mathcal{I}(Q, P \mid S)$, and suppose that both the node and the hollow abstractions are representable. Then the immersion (Q_S, P_S) is well-posed if and only if (Q, P) is well-posed.* ♢

PROOF Follows immediately from Theorem 3.3.6 and Theorem 3.4.7. ∎

Corollary 3.5.3: *Let (W, V) be a proper DNF defining the relationship $Y = WY + VU$ and let S_Y index all of the signals in Y. Consider the set of all possible node abstractions of some proper DNF (W, V); i.e., the set $A = \{\mathcal{A}(W, V \mid S) \ : \ S \subseteq S_Y\}$ (note that A includes (W, V)). Let $(Q_S, P_S) = \mathcal{H}(W_S, V_S)$ for $(W_S, V_S) \in A$ be the hollow abstraction of (W_S, V_S) (i.e., the immersion of (W, V) with respect to S). Then* every *immersion (Q_S, P_S) is representable if every principle submatrix of $(I - W)$ has a proper inverse.* ♢

PROOF Suppose that every principle submatrix of $(I - W)$ has a proper inverse. Then, by Corollary 3.3.5, every node abstraction (W_S, V_S) of (W, V) is representable. Consider some $(W_S, V_S) = \mathcal{A}(W, V \mid S)$ for some arbitrary $S \subseteq S_Y$ and let $(W_T, V_T) = \mathcal{A}(W_S, V_S \mid T)$ for some arbitrary $T \subseteq S$. By Theorem 3.3.11, $(W_T, V_T) = \mathcal{A}(W_S, V_S \mid S, T) = \mathcal{A}(W_S, V_S \mid T)$, thus (W_T, V_T) is also a node abstraction of (W, V) and is therefore also representable. Hence, since T was arbitrary, every node abstraction of (W_S, V_S) is representable and by Corollary 3.3.5, every principle submatrix of $(I - W_S)$ has a proper inverse. Thus, by Corollary 3.4.6, $\mathcal{H}(W_S, V_S)$ is also representable, and since S was arbitrary, every immersion of (W, V) is representable.

3.5.1 Sequential Immersions

We now wish to provide a result on sequential immersions akin to Theorem 3.3.11. Specifically, when we sequentially apply immersions to a DSF, the order in which we abstract away outputs

does not matter; all that matters is the set of outputs remaining at the end. We begin with a definition:

Definition 3.5.4: Sequential Immersion

Suppose that we wish to find an immersion of a given proper DSF (Q, P), and then we wish to find an immersion of the resultant DSF, and so on. We call this iterative abstraction process *sequential immersion*, and for notational simplicity, we define the sequential immersed network (Q_S, P_S) such that

$$(Q_S, P_S) = \mathcal{I}(Q, P \mid S_1, S_2, \cdots S_a) \triangleq$$
$$\mathcal{I}(\cdots \mathcal{I}(\mathcal{I}(W, V \mid S_1) \mid S_2) \mid S_a),$$

where, with S_Y indexing all of Y and for some integer $a > 0$, we have $S_a \subseteq \ldots \subseteq S_2 \subseteq S_1 \subseteq S_Y$ with $S_a \neq \emptyset$.

With this definition, we have the following results:

Lemma 3.5.5: *Let (Q, P) be an arbitrary proper DNF defining the relationship $Y = WY + VU$ and with $Q \in RP^{p\times p}$ hollow and $P \in RP^{p\times m}$. Let S_Y index all of Y and let $S, T \subseteq S_Y$ with $S \cap T = \emptyset$ and suppose that $\mathcal{I}(Q, P \mid S \cup T)$ and $\mathcal{I}(Q, P \mid S)$ are representable. Then*

$$\mathcal{I}(Q, P \mid S \cup T, S) = \mathcal{I}(Q, P \mid S). \tag{3.34}$$

♢

Proof By definition of sequential immersion, our objective is to show that

$$\mathcal{H}(\mathcal{A}(\mathcal{H}(\mathcal{A}(Q, P \mid S \cup T)) \mid S)) = \mathcal{H}(\mathcal{A}(Q, P \mid S)).$$

Define:

$$(W_{ST}, V_{ST}) = \mathcal{A}(Q, P \mid S \cup T),$$

$$(W_S, V_S) = \mathcal{A}(Q, P \mid S) = \mathcal{A}(Q, P \mid S \cup T, S), \quad \text{(Theorem 3.3.11)}$$
$$(Q_S, P_S) = \mathcal{H}(W_S, V_S) = \mathcal{H}(\mathcal{A}((Q, P \mid S)) = \mathcal{I}(Q, P \mid S).$$

Permute and partition W_{ST} and V_{ST} with respect to S to give

$$W_{ST} = \begin{bmatrix} [W_{ST}]_{SS} & [W_{ST}]_{S\bar{S}} \\ [W_{ST}]_{\bar{S}S} & [W_{ST}]_{\bar{S}\bar{S}} \end{bmatrix} \quad \text{and} \quad V_{ST} = \begin{bmatrix} [V_{ST}]_{SS} \\ [V_{ST}]_{\bar{S}\bar{S}} \end{bmatrix}.$$

This gives

$$W_S = [W_{ST}]_{SS} + [W_{ST}]_{S\bar{S}} \left(I - [W_{ST}]_{\bar{S}\bar{S}}\right)^{-1} [W_{ST}]_{\bar{S}S},$$
$$V_S = [V_{ST}]_{SS} + [W_{ST}]_{S\bar{S}} \left(I - [W_{ST}]_{\bar{S}\bar{S}}\right)^{-1} [V_{ST}]_{\bar{S}\bar{S}}.$$

We also have $(Q_S, P_S) = \mathcal{H}(W_S, V_S)$, which gives, with $D_S = \operatorname{diag} W_S$

$$Q_S = (I - D_S)^{-1}(W_S - D_S) \quad \text{and} \quad P_S = (I - D_S)^{-1} V_S.$$

Let $D_{ST} = \operatorname{diag} W_{ST}$ and $D_S = \operatorname{diag} W_S$. We have that

$$\begin{aligned}(Q_{ST}, P_{ST}) &\triangleq \mathcal{I}(Q, P \mid S \cup T) = \mathcal{H}(\mathcal{A}(Q, P \mid S \cup T)) \\ &= \left((I - D_{ST})^{-1}(W_{ST} - D_{ST}), (I - D_{ST})^{-1} V_{ST}\right).\end{aligned}$$

Since D_{ST} and $(I - D_{ST})^{-1}$ are diagonal matrices, we permute and partition (Q_{ST}, P_{ST}) with respect to S such that

$$Q_{ST} = \begin{bmatrix} \left(I - [D_{ST}]_{SS}\right)^{-1} \left([W_{ST}]_{SS} - [D_{ST}]_{SS}\right) & \left(I - [D_{ST}]_{SS}\right)^{-1} [W_{ST}]_{S\bar{S}} \\ \left(I - [D_{ST}]_{\bar{S}\bar{S}}\right)^{-1} [W_{ST}]_{\bar{S}S} & \left(I - [D_{ST}]_{\bar{S}\bar{S}}\right)^{-1} \left([W_{ST}]_{\bar{S}\bar{S}} - [D_{ST}]_{\bar{S}\bar{S}}\right) \end{bmatrix},$$

$$P_{ST} = \begin{bmatrix} (I - [D_{ST}]_{SS})^{-1} [V_{ST}]_{SS} \\ (I - [D_{ST}]_{\bar{S}\bar{S}})^{-1} [V_{ST}]_{\bar{S}\bar{S}} \end{bmatrix}.$$

Note that, for W and D being square matrices of the same dimension, we have

$$\begin{aligned} \big(I - (I-D)^{-1}(W-D)\big)^{-1} &= \big((I-D)^{-1}((I-D) - (W-D)\big)^{-1} \\ &= (I-W)^{-1}(I-D). \end{aligned} \tag{3.35}$$

Thus, with $(W_{STS}, V_{STS}) = \mathcal{A}(Q_{ST}, P_{ST} \mid S)$, we have

$$\begin{aligned} W_{STS} &= (I - [D_{ST}]_{SS})^{-1} ([W_{ST}]_{SS} - [D_{ST}]_{SS}) + \\ &\quad (I - [D_{ST}]_{SS})^{-1} [W_{ST}]_{S\bar{S}} \left(I - (I - [D_{ST}]_{\bar{S}\bar{S}})^{-1} ([W_{ST}]_{\bar{S}\bar{S}} - [D_{ST}]_{\bar{S}\bar{S}})\right)^{-1} (I - [D_{ST}]_{\bar{S}\bar{S}})^{-1} [W_{ST}]_{\bar{S}S} \\ &= (I - [D_{ST}]_{SS})^{-1} \left([W_{ST}]_{SS} + [W_{ST}]_{S\bar{S}} (I - [W_{ST}]_{\bar{S}\bar{S}})^{-1} [W_{ST}]_{\bar{S}S} - [D_{ST}]_{SS}\right) \\ &= (I - [D_{ST}]_{SS})^{-1} (W_S - [D_{ST}]_{SS}), \\ V_{STS} &= (I - [D_{ST}]_{SS})^{-1} ([V_{ST}]_{SS}) + \\ &\quad (I - [D_{ST}]_{SS})^{-1} [W_{ST}]_{S\bar{S}} \left(I - (I - [D_{ST}]_{\bar{S}\bar{S}})^{-1} ([W_{ST}]_{\bar{S}\bar{S}} - [D_{ST}]_{\bar{S}\bar{S}})\right)^{-1} (I - [D_{ST}]_{\bar{S}\bar{S}})^{-1} [V_{ST}]_{\bar{S}\bar{S}} \\ &= (I - [D_{ST}]_{SS})^{-1} \left([V_{ST}]_{SS} + [W_{ST}]_{S\bar{S}} (I - [W_{ST}]_{\bar{S}\bar{S}})^{-1} [V_{ST}]_{\bar{S}\bar{S}}\right) \\ &= (I - [D_{ST}]_{SS})^{-1} V_S. \end{aligned}$$

All that is left to show now is that $(Q_H, P_H) \triangleq \mathcal{H}(W_{STS}, V_{STS}) = (Q_S, P_S)$. Define $D_{STS} = \operatorname{diag} W_{STS}$. Observe that, since $[D_{ST}]_{SS}$ and $(I - [D_{ST}]_{SS})^{-1}$ are both diagonal matrices, we have that

$$\begin{aligned} W_{STS} &= (I - [D_{ST}]_{SS})^{-1} W_S - (I - [D_{ST}]_{SS})^{-1} [D_{ST}]_{SS} \qquad \Longrightarrow \\ D_{STS} &= (I - [D_{ST}]_{SS})^{-1} (D_S - [D_{ST}]_{SS}), \end{aligned}$$

where $D_S = \operatorname{diag} W_S$. Also, from (3.35), we have that

$$(I - D_{STS})^{-1} = (I - D_S)^{-1}(I - [D_{ST}]_{SS}).$$

We have

$$\begin{aligned}
Q_H &= (I - D_{STS})^{-1}(W_{STS} - D_{STS}) \\
&= (I - D_S)^{-1}(I - [D_{ST}]_{SS}) \left((I - [D_{ST}]_{SS})^{-1} (W_S - [D_{ST}]_{SS}) - (I - [D_{ST}]_{SS})^{-1} (D_S - [D_{ST}]_{SS}) \right) \\
&= (I - D_S)^{-1}(W_S - D_S) \\
&= Q_S, \\
P_H &= (I - D_{STS})^{-1} V_{STS} \\
&= (I - D_S)^{-1}(I - [D_{ST}]_{SS}) (I - [D_{ST}]_{SS})^{-1} V_S \\
&= (I - D_S)^{-1} V_S \\
&= P_S,
\end{aligned}$$

as desired, thus completing our proof. ∎

Theorem 3.5.6: *Let (Q, P) be an arbitrary DNF defining the relationship $Y = QY + PU$ and with $Q \in RP^{p \times p}$ hollow and $P \in RP^{p \times m}$. Let S_Y index all of Y and let, for some integer $a > 0$, $S_a \subseteq \ldots \subseteq S_2 \subseteq S_1 \subseteq S_Y$ with $S_a \neq \emptyset$. Then (assuming that every abstraction in the sequence below is representable),*

$$\mathcal{I}(Q, P \mid S_1, S_2, \cdots, S_a) = \mathcal{I}(Q, P \mid S_a), \tag{3.36}$$

that is, a sequential immersion is an immersion that is completely specified by the final index set. ♢

Proof We show this by induction. If $a = 2$, then by Lemma 3.5.5, we have that $\mathcal{I}(Q, P \mid S_1, S_2) = \mathcal{I}(Q, P \mid S_2)$. Now, for $2 \leq k < a$, suppose that $\mathcal{I}(Q, P \mid S_1, \cdots, S_k) = \mathcal{I}(Q, P \mid S_k)$. Then, by Lemma 3.5.5, we have that

$$\begin{aligned}
\mathcal{I}(Q, P \mid S_1, \cdots, S_k, S_{k+1}) &= \mathcal{I}(Q, P \mid S_k, S_{k+1}) \\
&= \mathcal{I}(Q, P \mid S_{k+1}).
\end{aligned}$$

Thus, by induction, we have that $\mathcal{I}(Q, P \mid S_1, S_2, \cdots, S_a) = \mathcal{I}(Q, P \mid S_a)$, as desired. ∎

The interpretation of Theorem 3.5.6 is the same as that of Theorem 3.3.11 discussed at the end of Section 3.3.5.

3.5.2 Identifiability Index

Recall from Section (3.4.5) that the identifiability index to reconstruct some arbitrary DSF (Q, P) (with $Q \in RP^{p\times p}$ and $P \in RP^{p\times n}$) is $p^2 - p$. Let $(Q_S, P_S) = \mathcal{H}(\mathcal{A}(W, V \mid S))$ be the immersed network of this DNF with $|S| < p$. We have that $Q_S \in RP^{|S|\times|S|}$, thus the identifiability index to reconstruct the immersion is $|S|^2 - |S| < p^2 - p$, meaning the immersion requires a lower identifiability index than the base network.

3.6 Stacked Immersions

We now wish to define a different type of abstraction, which we will call a *stacked immersion*, which reduces structural information of a DSF without reducing the number of outputs modeled. We define this abstraction over DSFs as such a definition allows us to construct a spectrum of models from state space models to the transfer function. We can also define stacked abstractions over DNFs; however, such a representation is not useful for this work.

Definition 3.6.1: The Stacked Immersion

Consider a DSF characterized by $Y = QY + PU$, and a partition of Y such that (possibly after re-ordering) $Y = \begin{bmatrix} Y_{S_1}' & \cdots & Y_{S_a}' \end{bmatrix}'$ with corresponding set of index sets $T = \{S_1, \ldots, S_a\}$. A *stacked immersion* of (Q, P) is the pair (Q_T, P_T) such that

$$Q_T = \begin{bmatrix} Q_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & Q_a \end{bmatrix}, \quad P_T = \begin{bmatrix} P_1 \\ \vdots \\ P_a \end{bmatrix}, \tag{3.37}$$

where $(Q_i, P_i) = \mathcal{I}(Q \mid S_i)$ (i.e., (Q_i, P_i) is the immersion of (Q, P) over S_i).

We write the stacked immersion of (Q, P) with respect to the partition T as $(Q_T, P_T) = \mathcal{F}(Q, P \mid T)$.

Note that if $Y = QY + PU$, by Theorem 3.3.2 and Theorem 3.4.3, we also have that $Y = Q_T Y + P_T U$, meaning that any stacked immersion of some DSF is dynamically equivalent to that DSF. Furthermore, we will have that Q_S is hollow since each Q_i is hollow. A stacked immersion of (Q, P) is thus a stacking of regular immersions across a full partitioning of the outputs so that we can come up with an alternate (Q_T, P_T) that preserves the same input-output mapping.

It is very important to note that the stacked immersion (Q_T, P_T) is syntactically equivalent to a DSF (Q, P) where $Q = Q_T$ and $P = P_T$. However, semantically, they are very different. As discussed in Section 3.3, immersions preserve the independence pattern represented in a system; therefore, in (Q_T, P_T), the independence pattern is preserved within each diagonal block but hidden between blocks. In other words, for a DSF (and barring cancellations), a zero in an off-diagonal of Q entry means that the sum of all walks between the corresponding outputs must be zero, whereas, in the stacked immersion, this constraint need not hold.

In short, the stacked immersion (Q_T, P_T) is *not* a DSF (or even a DNF) since the zeros in the off-diagonal blocks do not represent independence between manifest variables. Hence, we call (Q_T, P_T) a *multi-DSF*. It should also be noted that if Y is not partitioned, meaning that $T = \{\{1, \ldots, p\}\}$, then $\mathcal{F}(Q, P \mid T) = (Q, P)$, meaning any DSF is a stacked immersion of itself, and therefore is a multi-DSF. Thus the set of DSFs is a subset of the set of multi-DSFs.

3.6.1 A Spectrum of Models

We now use the results of the previous sections to demonstrate the capability of multi-DSFs to represent LTI systems with any desired level of structural information. To accomplish

this, we will show that, for any arbitrary state space model, there exists a DSF, called the full-state DSF, that is informationally equivalent to that model. We also show that this same DSF has a stacked immersion in one-to-one correspondence with the transfer function of that state space model. We begin with a definition.

Definition 3.6.2: Full-State DSF

Suppose that we have a state space model (A, B, C, D), where $C = I$ (i.e., we measure every state). With $A \triangleq [a_{ij}] \in \mathbb{R}^{p\times p}$ and $B \triangleq [b_{ij}] \in \mathbb{R}^{n\times m}$, and following the procedure contained in [1], we derive the DSF (Q, P) of this model as

$$Q = \begin{bmatrix} 0 & \frac{a_{12}}{s-a_{11}} & \cdots & \frac{a_{1p}}{s-a_{11}} \\ \frac{a_{21}}{s-a_{22}} & 0 & \cdots & \frac{a_{2p}}{s-a_{22}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{a_{p1}}{s-a_{pp}} & \frac{a_{p2}}{s-a_{pp}} & \cdots & 0 \end{bmatrix}, \tag{3.38}$$

$$P = \begin{bmatrix} \frac{b_{11}}{s-a_{11}} & \frac{b_{12}}{s-a_{11}} & \cdots & \frac{b_{1m}}{s-a_{11}} \\ \frac{b_{21}}{s-a_{22}} & \frac{b_{22}}{s-a_{11}} & \cdots & \frac{b_{2p}}{s-a_{22}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{b_{p1}}{s-a_{pp}} & \frac{b_{p2}}{s-a_{pp}} & \cdots & \frac{b_{pm}}{s-a_{11}} \end{bmatrix} + (I - Q)D. \tag{3.39}$$

We call (Q, P) in equations (3.38) and (3.39) the *full-state DSF*.

Theorem 3.6.3: *Let (A, B, C, D) with $C = I$, and let (Q, P) be the (full state) DSF corresponding to this state space model. Then (Q, P) is the unique immersion of (A, B, C, D) and (A, B, C, D) is the unique minimal realization of (Q, P).* ♢

Proof First, (Q, P) can be recovered uniquely from (A, B, C, D) regardless of the choice of C (see [1]). To recover (A, B, C, D) from (Q, P), first note that every entry in Q is a strictly proper transfer function with a constant in the numerator and one pole in the denominator. The pole is consistent in each row, and is the value for the diagonal entry of that row. The

numerator on the off-diagonals is the value of the off-diagonals in A, and so we can recover A uniquely from Q.

Since Q is strictly proper, $(I - Q)$ will have a proper inverse [14, 15]. Let $P = \tilde{P} + (I - Q)D$. Then $(I - Q)^{-1}P = (I - Q)^{-1}\tilde{P} + D$. Since $\tilde{P}$ is strictly proper from (3.39), we have that $(I - Q)^{-1}\tilde{P}$ will be strictly proper. Furthermore, since it is strictly proper, we have that $\lim_{s\to\infty}(I - Q)^{-1}\tilde{P} = 0$. Thus $\lim_{s\to\infty}(I - Q)^{-1}P = D$ and we have recovered D uniquely. Subtract $(I - Q)D$, which is known, from P to get $\tilde{P}$. The numerators of $\tilde{P}$ specify B uniquely.

Finally, no state space realization of a smaller order can generate (Q, P) of the given dimensions; thus the (A, B, C, D) found must be minimal. ■

Thus, by Theorem 3.6.3, a full-state DSF is informationally equivalent to a (minimal) state space representation.

We now look at the other end of the spectrum, the transfer function. Again, we begin with a definition.

Definition 3.6.4: Final Immersion

A *final immersion* of a proper DSF (Q, P) is the stacked immersion $F_T = \mathcal{F}(Q, P \mid T)$ where $T = \{\{1\}, \ldots \{p\}\}$.

Theorem 3.6.5: *Let (Q, P) be a proper DSF with transfer function $G_{cl} = (I - Q)^{-1}P$, and suppose that $T = \{\{1\}, \ldots \{p\}\}$. Then the final immersion $\mathcal{F}(Q, P \mid T) = (0, G_{cl})$.* ♢

Proof By Theorem 3.3.2 and Theorem 3.4.3, we have that $(Q_F, P_F) = \mathcal{F}(Q, P \mid \{\{1\}, \ldots \{p\}\})$ is given by

$$Q_F = \begin{bmatrix} 0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 0 \end{bmatrix} = 0, \quad P_F = \begin{bmatrix} G_1 \\ \vdots \\ G_p \end{bmatrix} = G_{cl}, \tag{3.40}$$

where G_i is the i'th row of G_{cl}. ■

Theorem 3.6.5 shows us that the final immersion is in one-to-one correspondence with the transfer function of the full-state DSF. Furthermore, the transfer function of the full-state DSF is also the same as the transfer function of its corresponding state-space realization.

Thus Theorem 3.6.3 and Theorem 3.6.5 shows us that there exists a multi-DSF containing as much structural information as the state space, and a multi-DSF containing as little as the transfer function. By choosing other partitions of the outputs Y and taking a stacked immersion with that partition, we can create other multi-DSFs containing intermediate levels of information. As a result, the multi-DSF is a general model of LTI systems, capable of representing many levels of structural knowledge, many more than state space models or transfer functions alone are capable of doing. These different immersions form a spectrum of models of LTI systems containing various levels of structural information.

3.6.2 Identifiability Index

The identifiability index to reconstruct a stacked immersion of the form (3.37) is equal to the sum of the identifiability indices to reconstruct each individual (Q_i, P_i) for $i = 1, \ldots, n$. Let $Q_i \in RP^{p_i \times p_i}$, with $\sum_{i=1}^{a} p_i = p > 0$. Then the identifiability index to reconstruct the stacked immersion is

$$\sum_{i=1}^{a} (p_i^2 - p_i) = \left(\sum_{i=1}^{a} p_i^2 \right) - p \leq p^2 - p, \tag{3.41}$$

with equality only when $a = 1$. Thus, if the stacked immersion is not equal to the original DSF, the stacked immersion has a strictly smaller identifiability index required to reconstruct it from data than the original DSF.

3.7 Numeric Examples

We now provide three numeric examples illustrating some of the points of this paper.

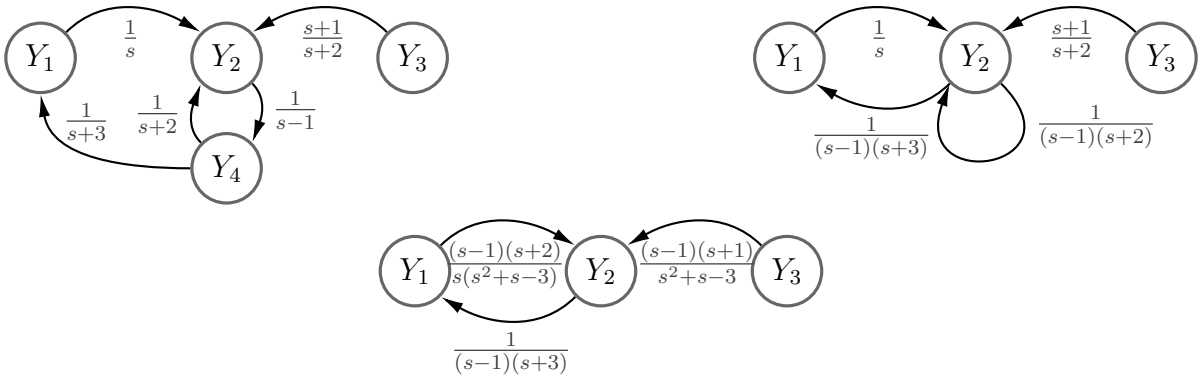


Figure 3.1 Example signal structures of a network, its node abstraction, and its immersion as computed in Section 3.7.1. Note that we are only drawing the sub-graph of the signal structure containing the nodes in Y but hiding the nodes in U. (Top-Left) the signal structure of the starting DSF (Q, P). (Top-Right) the signal structure of the DNF (W_S, V_S) in (3.43a) found by performing a node abstraction that removes node Y_4 from (Q, P). (Bottom-Middle) the signal structure of the immersed DSF (Q_S, P_S) in found by performing a hollow abstraction on (W_S, V_S).

3.7.1 Immersions

We demonstrate the computation of an immersion and show that the independence pattern is preserved as required in Definition 3.3.1. Consider a DSF $Y = QY + PU$ given by

$$\begin{bmatrix} Y_1 \\ Y_2 \\ Y_3 \\ Y_4 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & \frac{1}{s+3} \\ \frac{1}{s} & 0 & \frac{s+1}{s+2} & \frac{1}{s+2} \\ 0 & 0 & 0 & 0 \\ 0 & \frac{1}{s-1} & 0 & 0 \end{bmatrix} \begin{bmatrix} Y_1 \\ Y_2 \\ Y_3 \\ Y_4 \end{bmatrix} + I_4 \begin{bmatrix} U_1 \\ U_2 \\ U_3 \\ U_4 \end{bmatrix}. \tag{3.42}$$

See the top left of Figure 3.1 for the signal structure $\Gamma(Q)$.

We wish to abstract away Y_4 and then perform a hollow abstraction on the resulting DNF to get another DSF. In other words, we wish to find $(W_S, V_S) = \mathcal{A}(Q, P \mid S)$ for $S = \{1, 2, 3\}$ and then $(Q_S, P_S) = \mathcal{H}(W_S, V_S)$. Note also that $\bar{S} = \{4\}$ defines the nodes we are abstracting away. To find this abstraction, we first partition Q and P as described in

(3.9). This gives

$$
\begin{aligned}
Q_{SS} &= \begin{bmatrix} 0 & 0 & 0 \\ \frac{1}{s} & 0 & \frac{s+1}{s+2} \\ 0 & 0 & 0 \end{bmatrix}, & Q_{S\bar{S}} &= \begin{bmatrix} \frac{1}{s+3} \\ \frac{1}{s+2} \\ 0 \end{bmatrix}, \\
Q_{\bar{S}S} &= \begin{bmatrix} 0 & \frac{1}{s-1} & 0 \end{bmatrix}, & Q_{\bar{S}\bar{S}} &= \begin{bmatrix} 0 \end{bmatrix}, \\
P_{SS} &= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}, & P_{\bar{S}\bar{S}} &= \begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix}.
\end{aligned}
$$

Thus, we can compute $W_S = Q_{SS} + Q_{S\bar{S}}(I - Q_{\bar{S}\bar{S}})^{-1}Q_{\bar{S}S}$ and $V_S = P_{SS} + Q_{S\bar{S}}(I - Q_{\bar{S}\bar{S}})^{-1}P_{\bar{S}\bar{S}}$ as

$$
W_S = \begin{bmatrix} 0 & \frac{1}{(s-1)(s+3)} & 0 \\ \frac{1}{s} & \frac{1}{(s-1)(s+2)} & \frac{s+1}{s+2} \\ 0 & 0 & 0 \end{bmatrix}, , \quad V_S = \begin{bmatrix} 1 & 0 & 0 & \frac{1}{s+3} \\ 0 & 1 & 0 & \frac{s-1}{s+2} \\ 0 & 0 & 1 & 0 \end{bmatrix}. \tag{3.43a}
$$

See the top right of Figure 3.1 for the signal structure $\Gamma(W_S)$. Note the similarity of the signal structure of W_S to the signal structure of Q. There is a new edge from Y_2 to Y_1 in $\Gamma(W_S)$ corresponding to the walk $Y_2 \to Y_4 \to Y_1$ in $\Gamma(Q_S)$. Similarly, there is a new edge (a self loop) from Y_2 to itself corresponding to the walk $Y_2 \to Y_4 \to Y_2$. Thus all walks starting and ending at nodes in $S = \{1, 2, 3\}$ but passing only through $\bar{S} = \{4\}$ in $\Gamma(Q)$ are now manifest in $\Gamma(W_S)$ as direct edges between the endpoint manifest variables in S.

To return to an immersed DSF, we must remove all self-loops in $\Gamma(W_S)$ and re-normalize the edges entering the node with the removed self-loop to preserve the dynamics removed. To see the benefit of this re-normalization, zero out every link in the middle graph of Figure 3.1 except $Y_1 \to Y_2$ and the self-loop $Y_2 \to Y_2$. If we were to compute the closed-loop transfer function from Y_1 to Y_2, we would start with the equation $Y_2 = W_{22}Y_2 + W_{21}Y_1$, thus $Y_2 = (1 - W_{22})^{-1}W_{21}Y_1$ and the closed-loop transfer function is $(1 - W_{22})^{-1}W_{21} = \frac{(s-1)(s+2)}{s(s^2+s-3)}$.

Note from the figure on the right of Figure 3.1 that this is precisely the weight on the link $Y_1 \to Y_2$, thus the hollowness of Q_A preserves the meaning that if all but one link is zeroed out in a network, the remaining link represents the closed-loop transfer function from tail to tip.

To perform this re-normalization (i.e., to perform the edge abstraction), let

$$D_W = \text{diag}(W_S) = \text{diag}\left(0, \frac{1}{(s-1)(s+2)}, 0\right). \tag{3.44}$$

We then compute $Q_S = (I - D_W)^{-1}(W_S - D_W)$ and $P_S = (I - D_W)^{-1}V_S$. This gives

$$Q_S = \begin{bmatrix} 0 & \frac{1}{(s-1)(s+3)} & 0 \\ \frac{(s-1)(s+2)}{s(s^2+s-3)} & 0 & \frac{(s-1)(s+1)}{s^2+s-3} \\ 0 & 0 & 0 \end{bmatrix}, \quad P_S = \begin{bmatrix} 1 & 0 & 0 & \frac{1}{s+3} \\ 0 & \frac{(s-1)(s+2)}{s^2+s-3} & 0 & \frac{s-1}{s^2+s-3} \\ 0 & 0 & 1 & 0 \end{bmatrix}. \tag{3.45a}$$

The signal structure $\Gamma(Q_S)$ is given on the bottom middle of Figure 3.1. Note that, with the exception of the self-loop, all edges that exist in $\Gamma(W_S)$ likewise exist in $\Gamma(Q_S)$.

3.7.2 The DSF Versus the Multi-DSF

As mentioned in Section 3.6, a multi-DSF formed from stacked immersion is semantically different than a DSF that looks identical. We illustrate this difference using the necessary identifiability index from network reconstruction. Suppose that we have a DSF $Y = QY + PU$, where

$$\begin{bmatrix} Y_A \\ Y_B \end{bmatrix} = \begin{bmatrix} Q_A & 0 \\ 0 & Q_B \end{bmatrix} \begin{bmatrix} Y_A \\ Y_B \end{bmatrix} + \begin{bmatrix} P_A \\ P_B \end{bmatrix} U, \tag{3.46}$$

with $Q_A, P_A \in RP^{3\times 3}$ and $Q_B, P_B \in RP^{2\times 2}$. If (Q, P) were a standard DSF, then the identifiability index necessary to recover this DSF from data (see Section 3.2.3) is $p^2 - p = (3+2)^2 - (3+2) = 20$.

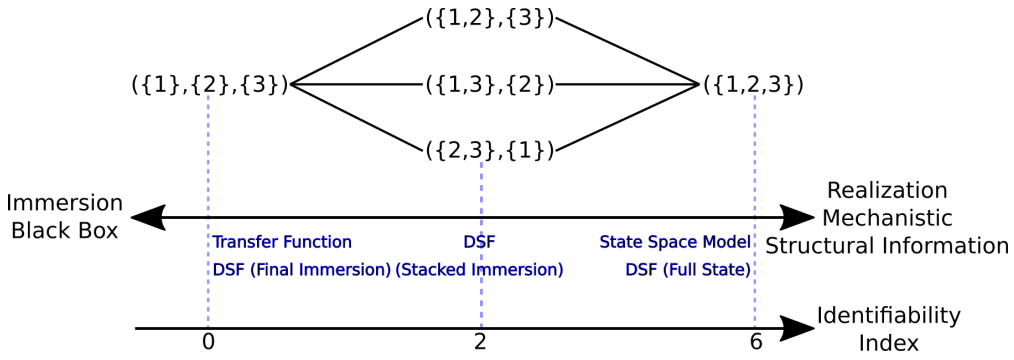


Figure 3.2 The spectrum of models corresponding to the example in Section 3.7.3. We label each node in the graph with the partition T defining the stacked immersion, so the right side of the graph corresponds to the state space model and the full-state DSF and the left side corresponds to the transfer function and the final immersion. The identifiability index, as well as structural information, increase as the graph moves from left to right.

However, if (Q, P) were a stacked immersion constructed from some realization (Q_R, P_R), with $(Q_A, P_A) = \mathcal{A}(Q_R, P_R \mid A)$ and $(Q_B, P_B) = \mathcal{A}(Q_R, P_R \mid B)$, then the identifiability index is equal to the sum of the individual identifiability indices of each DSF (Q_A, P_A) and (Q_B, P_B) (since we would reconstruct each immersion individually and then stack them together to form the stacked immersion). Thus the identifiability index is $(3)^2 - 3 + (2)^2 - 2 = 8 \neq 20$. Furthermore, we see that the cost of recovering the multi-DSF is less than the DSF itself.

It can be shown that the identifiability index of the final immersion is zero, and the identifiability index of the full-state DSF is $n^2 - n$, where n is the number of states. All other stacked immersions have a corresponding identifiability index that is strictly between 0 and $n^2 - n$; thus the identifiability index imposes a partial ordering among stacked immersions corresponding to the amount of structural information encoded by each stacked immersion. More on this in Section 3.7.3.

3.7.3 Stacked Immersions and a Spectrum of Models

Consider the following state space model:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \end{bmatrix} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} u,$$

$$y = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} x.$$

By the process in [1] and Theorem 3.6.3, the corresponding (full-state) DSF is $Y = QY + PU$ with

$$Q = \begin{bmatrix} 0 & \frac{2}{s-1} & \frac{3}{s-1} \\ \frac{4}{s-5} & 0 & \frac{6}{s-5} \\ \frac{7}{s-9} & \frac{8}{s-9} & 0 \end{bmatrix}, \quad P = \begin{bmatrix} \frac{1}{s-1} & 0 & 0 \\ 0 & \frac{1}{s-5} & 0 \\ 0 & 0 & \frac{1}{s-9} \end{bmatrix}.$$

To reconstruct this DSF (or the corresponding state space model), the identifiability index required is $p^2 - p = 3^2 - 3 = 6$.

The outputs are enumerated by the set $\{1, 2, 3\}$. This set has 5 partitions, namely $(\{1, 2, 3\})$, $(\{1, 2\}, \{3\})$, $(\{1, 3\}, \{2\})$, $(\{2, 3\}, \{1\})$, and $(\{1\}, \{2\}, \{3\})$, which are the nodes shown in Figure 3.2. When the partition is $(\{1, 2, 3\})$, we have that the stacked immersion is equal to itself (i.e., $(Q, P) = \mathcal{F}(Q, P \mid (\{1, 2, 3\}))$).

Now consider the partition $(\{1, 2\}, \{3\})$. To find the corresponding stacked immersion (Q_A, P_A), we first compute the immersions $(Q_{\{1,2\}}, P_{\{1,2\}}) = \mathcal{A}(Q, P \mid \{1, 2\})$ and $(Q_{\{3\}}, P_{\{3\}}) = \mathcal{A}(Q, P \mid \{3\})$. This yields

$$Q_{\{1,2\}} = \begin{bmatrix} 0 & \frac{2(s+3)}{d_1} \\ \frac{2(2s+3)}{d_2} & 0 \end{bmatrix}, \quad P_{\{1,2\}} = \begin{bmatrix} \frac{s-9}{d_1} & 0 & \frac{3}{d_1} \\ 0 & \frac{s-9}{d+2} & \frac{6}{d_2} \end{bmatrix},$$

where $d_1 = s^2 - 10s - 12$ and $d_2 = s^2 - 14s - 3$. Furthermore, we have that

$$Q_{\{3\}} = \begin{bmatrix} 0 \end{bmatrix}, \quad P_{\{3\}} = \begin{bmatrix} \frac{7s-3}{d_3} & \frac{8s+6}{d_3} & \frac{s^2-6s-3}{d_3} \end{bmatrix},$$

where $d_3 = s(s^2 - 15s - 18)$. Thus, the stacked immersion $(Q_A, P_A) = \mathcal{F}(Q, P \mid (\{1,2\}, \{3\}))$ is given by

$$Q_A = \begin{bmatrix} 0 & \frac{2(s+3)}{d_1} & 0 \\ \frac{2(2s+3)}{d_2} & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \; P_A = \begin{bmatrix} \frac{s-9}{d_1} & 0 & \frac{3}{d_1} \\ 0 & \frac{s-9}{d+2} & \frac{6}{d_2} \\ \frac{7s-3}{d_3} & \frac{8s+6}{d_3} & \frac{s^2-6s-3}{d_3} \end{bmatrix}.$$

The identifiability index necessary to reconstruct this multi-DSF is the sum of the identifiability indices to reconstruct $(Q_{\{1,2\}}, P_{\{1,2\}})$ and $(Q_{\{3\}}, P_{\{3\}})$, which is $(2)^2 - 2 + 1^2 - 1 = 2$. We compute the stacked immersions for partitions $(\{1,3\}, \{2\})$ and $(\{2,3\}, \{1\})$ are computed in a similar manner and find that these have the same identifiability index.

Finally, consider the partition $(\{1\}, \{2\}, \{3\})$. To find the corresponding stacked immersion (Q_F, P_F), we first compute the immersions $(Q_{\{1\}}, P_{\{1\}}) = \mathcal{A}(Q, P \mid \{1\})$, $(Q_{\{2\}}, P_{\{2\}}) = \mathcal{A}(Q, P \mid \{2\})$, and $(Q_{\{3\}}, P_{\{3\}}) = \mathcal{A}(Q, P \mid \{3\})$. We already computed the last of these above. We also have that

$$Q_{\{1\}} = \begin{bmatrix} 0 \end{bmatrix}, \quad P_{\{1\}} = \begin{bmatrix} \frac{s^2-14s-3}{d_3} & \frac{2s+6}{d_3} & \frac{3s-1}{d_3} \end{bmatrix},$$

and

$$Q_{\{2\}} = \begin{bmatrix} 0 \end{bmatrix}, \quad P_{\{2\}} = \begin{bmatrix} \frac{4s+6}{d_3} & \frac{s^2-10s-12}{d_3} & \frac{6s+6}{d_3} \end{bmatrix}.$$

Thus the final immersion $(Q_F, P_F) = \mathcal{F}(Q, P \mid (\{1\}, \{2\}, \{3\}))$ is given by

$$Q_F = 0, \quad P_F = \begin{bmatrix} \frac{s^2-14s-3}{d_3} & \frac{2s+6}{d_3} & \frac{3s-1}{d_3} \\ \frac{4s+6}{d_3} & \frac{s^2-10s-12}{d_3} & \frac{6s+6}{d_3} \\ \frac{7s-3}{d_3} & \frac{8s+6}{d_3} & \frac{s^2-6s-3}{d_3} \end{bmatrix}.$$

We can verify that $P_F = (I - Q)^{-1}P = C(sI - A)^{-1}B \triangleq G$, thus (Q_F, P_F) is equivalent to the closed-loop transfer function of this system. The identifiability index to reconstruct this multi-DSF is the sum of the identifiability index to reconstruct $(Q_{\{1\}}, P_{\{1\}})$, $(Q_{\{2\}}, P_{\{2\}})$, and $(Q_{\{3\}}, P_{\{3\}})$, which is $(1)^2 - 1 + (1)^2 - 1 + (1)^2 - 1 = 0$. This is sensible since the transfer function can be recovered from data uniquely without requiring any additional structural information.

With these computations, we can build a spectrum of models–as described in Section 3.6.1–from least structurally informative (requiring a lower identifiability index) to most structurally informative (requiring the largest identifiability index). This spectrum is shown in Figure 3.2.

3.8 Appendix

We list several known results on matrices of rational polynomials that we use throughout this work.

Lemma 3.8.1: *Let $M(s)$ and $N(s)$ be proper matrices of rational polynomials, each of dimension $m \times n$. Then $M(s) + N(s)$ is likewise proper. Furthermore, if both $M(s)$ and $N(s)$ are strictly proper, the sum is also strictly proper.* □

Proof See Lemma 2.7.3 in Chapter 2. ■

Lemma 3.8.2: *Let $M(s)$ be a matrix of rational polynomials of dimension $p \times m$, and let $N(s)$ be a matrix of rational polynomials of dimension $m \times n$. If $M(s)$ and $N(s)$ are both proper, then $M(s)N(s)$ is proper. Likewise, if one of $M(s)$ and $N(s)$ are strictly proper, then the product is also strictly proper.* □

Proof See Lemma 2.7.4 in Chapter 2. ■

Lemma 3.8.3: *Let $Q \in RP^{p\times p}$. Suppose that $(I - Q)$ has an inverse and that $\sum_{k=0}^{\infty} Q^k$ converges. Then $(I - Q)^{-1} = \sum_{k=0}^{\infty} Q^k$.* □

PROOF We must show that $(I-Q)$ is both a left and a right inverse of $\sum_{k=0}^{\infty} Q^k$. To show that it is a left inverse, we have (since $\sum_{k=0}^{\infty} Q^k$ converges):

$$(I-Q)\sum_{k=0}^{\infty} Q^k = \sum_{k=0}^{\infty} Q^k - Q\sum_{k=0}^{\infty} Q^k = \sum_{k=0}^{\infty} Q^k - \sum_{k=1}^{\infty} Q^k$$
$$= \left(Q^0 + \sum_{k=1}^{\infty} Q^k\right) - \sum_{k=1}^{\infty} Q^k = Q^0 = I.$$

To show that $(I-Q)$ is a right inverse, we have that (again since $\sum_{k=0}^{\infty} Q^k$ converges):

$$I = (I-Q)\sum_{k=0}^{\infty} Q^k = \sum_{k=0}^{\infty} Q^k - Q\sum_{k=0}^{\infty} Q^k$$
$$= \sum_{k=0}^{\infty} Q^k - \left(\sum_{k=0}^{\infty} Q^k\right) Q = \sum_{k=0}^{\infty} Q^k (I-Q).$$

Thus, since $(I-Q)$ is invertible, it is both a left and right inverse of $\sum_{k=0}^{\infty} Q^k$. ∎

Lemma 3.8.4: *Let $G(s)$ be a square and proper matrix of rational polynomials. Then the inverse of $G(s)$ exists and is proper (a) if and (b) only if $G(\infty) \triangleq \lim_{s\to\infty} G(s)$ is non-singular.* □

PROOF See Lemma 2.7.7 in Chapter 2. ∎

The remaining results deal with Schur Complements and their inverses. We extend the known results, given in Lemma 3.8.5, to the realm of matrices of proper rational polynomials.

Lemma 3.8.5 (Schur): *Let*

$$M = \begin{bmatrix} A & B \\ C & D \end{bmatrix}$$

be any arbitrary square matrix. If any two of the following matrices are invertible, then the third is as well: (i) M, (ii) D, and (iii) M/D (where $M/D = A - BD^{-1}C$ is the Schur Complement of D in M). □

Lemma 3.8.6: *Let $M(s)$ be a square matrix of proper rational polynomials such that*

$$M(s) = \begin{bmatrix} A(s) & B(s) \\ C(s) & D(s) \end{bmatrix} \tag{3.47}$$

Then, if any of the following two are true, the third is also true: (i) $M(s)$ has a proper inverse, (ii) $D(s)$ has a proper inverse, and (iii) $M(s)/D(s)$ has a proper inverse (where $M(s)/D(s) = A(s) - B(s)D^{-1}(s)C(s)$ is the Schur Complement of $D(s)$ in $M(s)$). □

PROOF This follows immediately from taking all of these matrices at $s \to \infty$, applying Lemma 3.8.5, and then recovering proper inverses using Lemma 3.8.4. ■

Chapter 4

Reconstruction of Proper LTI Dynamic Networks

We now build on the results of Chapters 2 and 3 to study the *network reconstruction* problem. In network reconstruction, we are given either input-output data or an input-output map, along with extra a priori information (either known or assumed) and seek to identify the unique structured model that produced that data or input-output map. For our purposes, the structured model we seek to identify (learn) is the DNF and the DSF.

4.1 Notation

Unless otherwise noted, we use the same notation used by previous chapters. In addition, for n arbitrary matrices $A_1, A_2, \ldots, A_n$, we have that $A = A_1 \oplus A_2 \oplus \cdots \oplus A_n$ is the direct sum of these matrices; i.e., A is a block-diagonal matrix where the i'th diagonal block is A_i. If, for some $\hat{A}$, we have $\hat{A} = A_1 = A_2 = \ldots = A_n$, then, for clarity, we write $A = \hat{A} \oplus \overset{n}{\cdots} \oplus \hat{A} = A_1 \oplus \cdots \oplus A_n$. For example, if

$$A_1 = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}, \quad A_2 = 5, \quad A_3 = \begin{bmatrix} 6 & 0 \\ 0 & 7 \end{bmatrix}, \tag{4.1}$$

then

$$A_1 \oplus A_2 \oplus A_3 = \begin{bmatrix} 1 & 2 & 0 & 0 & 0 \\ 3 & 4 & 0 & 0 & 0 \\ 0 & 0 & 5 & 0 & 0 \\ 0 & 0 & 0 & 6 & 0 \\ 0 & 0 & 0 & 0 & 7 \end{bmatrix}, \tag{4.2}$$

and

$$A_1 \oplus \overset{3}{\cdots} \oplus A_1 = \begin{bmatrix} 1 & 2 & 0 & 0 & 0 & 0 \\ 3 & 4 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 2 & 0 & 0 \\ 0 & 0 & 3 & 4 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 2 \\ 0 & 0 & 0 & 0 & 3 & 4 \end{bmatrix}. \tag{4.3}$$

4.2 Related Work and Contributions

Much of the research into network reconstruction can be split into three different problems, namely (1) full reconstruction, (2) topological reconstruction, and (3) single module identification. Here, we provide an overview of existing research tackling these three problems using both state-space models and DSFs as the model they seek to reconstruct. We also describe the contributions of this chapter to the overall reconstruction community.

REMARK 4.2.1: Though we distinguish between state-space-model-based approaches and DSF-based approaches, we note that the state-space-model-based approaches are a special case of the DSF-based approaches. This holds since the state-space-model-based assume full-state measurement (i.e., they assume that $C = I$), which are in one-to-one correspondence with a special DSF that we call the full-state DSF (see Theorem 3.6.3).

We show the equivalence with a state space model with full state measurements and a DSF. In the more general case where A is not hollow, then the equivalence is

between a DNF and a state space model, instead of a DSF and a state space model. We focus our attention on DSFs in this remark; however, the generalization to a DNF follows in the same way.

Another way to see this equivalence is to consider a DSF in the time domain (which we detail further in Section 4.4). Assume that $P = \frac{1}{z}I$ and to consider the convolutional form of this DSF given by

$$\begin{aligned} y(t) &= Q(0)y(t) + Q(1)y(t-1) + Q(2)y(t-2) + \cdots + Q(t)y(0) + Pu(t) \\ &= Q(0)y(t) + Q(1)y(t-1) + Q(2)y(t-2) + \cdots + Q(t)y(0) + z^{-1}Iu(t) \\ &= Q(0)y(t) + Q(1)y(t-1) + Q(2)y(t-2) + \cdots + Q(t)y(0) + u(t-1), \end{aligned} \tag{4.4}$$

where the last equality holds since z^{-1} is the 1-step time delay operator.

Now, consider state space model such that A is hollow and $B = C = I$. The dynamics of this state-space model is given by

$$y(t) = Ay(t-1) + u(t-1). \tag{4.5}$$

Notice that (4.4) is equivalent to (4.5) in the special case where $A = Q(1)$ and $Q(0) = Q(2) = Q(3) = \cdots = Q(t) = 0$. In particular, if we follow the steps in Section 2.4.1 to convert this state space model into a DSF, we get

$$\begin{aligned} \tilde{W} &= A, \\ \tilde{V} &= B, \\ \operatorname{diag}\tilde{W} &= 0 \qquad\qquad \text{since } A \text{ is hollow} \\ Q &= (zI - \operatorname{diag}\tilde{W})^{-1}(\tilde{W} - \operatorname{diag}\tilde{W}) = Az^{-1} \\ P &= (zI - \operatorname{diag}\tilde{W})^{-1}\tilde{V} = z^{-1}I. \end{aligned}$$

Thus, we have

$$\begin{aligned} y(z) &= Q(z)y(z) + P(z)u(z) \\ &= Az^{-1}y(z) + z^{-1}u(z) \qquad \Longrightarrow \\ y(t) &= Ay(t-1) + u(t-1). \end{aligned}$$

Hence, the convolutional form of this full-state DSF is precisely (4.5) and the state-space-model-based approaches are a special case of the DSF-based approaches.

Note that several of the works in the reconstruction literature refer to DSFs under different names, such as *linear dynamical networks*, *linear dynamic influence models*, or *network Granger causal models* (this last one being a DSF in the time-domain where edges in Q are finite impulse responses).

4.2.1 Full Reconstruction

The full network reconstruction problem seeks to reconstruct both the structure (i.e., the topology or the zero-pattern of the network graph) and the dynamics (i.e., the weights or operators on the non-zero edges in the network graph) of a dynamic network given some a priori knowledge or assumptions about the structure of the network. The full reconstruction problem is the primary focus of this work.

State-Space-Model-Based Approaches

Much of the early work in network reconstruction was performed to recover biochemical reaction networks. The work in [49] gives the dynamics of biochemical reactions according to the differential equation $\dot{x} = f(x)$. It then proposed a method for finding the best fit for entries in the Jacobian of f, which they use as the structured representation of this system. This Jacobian is the matrix A in the state space model; thus, this method reconstructs a linear approximation to an autonomous system in which every state is measured. In

[50–52], information-theoretic methods–specifically Dynamic Bayesian Networks or Dynamic Probabilistic Networks (which are autonomous state space models with full state measurement where the signals are random variables and the entries in A are transition probabilities)–were used to represent models of biochemical reaction networks.

The work [53] considers the problem of reconstructing a large-scale sparse network of interconnected state-space models. In [54], the necessary identifiability index (see Definition 4.3.4) for reconstructing the general interconnection of state space models is presented and shown to be strictly greater in almost every case than the identifiability index for a DSF (meaning it is more expensive to reconstruct a network of interconnected state space models than it is to reconstruct a DSF).

DSF-Based Approaches

The paper [55] shows that the reconstruction of biochemical reaction networks and other similar systems using DSFs has considerable advantages over state-space-model-based and other previous approaches. First, the DSF is invariant to coordinate changes among hidden variables, meaning that it is a good representation of the structure between manifest variables. Second, the DSF is probably an intermediate representation of structure; in other words, the structure recovered through reconstruction has meaning concerning the relationships between manifest variables. Other network representations–such as neural networks and Bayesian networks–do not preserve these structural semantics.

Dynamical structure functions as a representation of LTI systems–as well as the frequency-domain network reconstruction algorithm (see Section 4.3) to recover a DSF from data–were first proposed in [19, 37, 55]. The works [2, 41, 43, 56] present enhancements to the frequency-domain reconstruction methodology to make the algorithms robust to noisy data. In [57], results from compressive sensing are used to reconstruct a DSF when Q is sparse. This work also presents a methodology for reconstructing a network when only some subset of the inputs are non-zero, which requires first finding an immersion of the network. The work

[58] presents an alternative procedure to recover the dynamics of a proper and well-posed network driven by rank-reduced noise using a weighted least squares in the situation where the topology of the network is known.

An important concept which has arisen from the study of the frequency-domain network reconstruction methods is an understanding of the fundamental limitations of network reconstruction algorithms. Input-output data only gives input-output maps, and additional information is necessary to get a unique structured representation of a system. See [2, 43, 59]–as well as Definition 4.3.5 and Theorem 4.3.9 below–for the precise conditions for the kind and quantity of information required to reconstruct a network. Additional studies on identifiability conditions over the reconstruction of DSFs can be found in [28, 60, 61]

Early time-domain reconstruction processes (see Section 4.4) were introduced in [62–65] and seek to solve the blind reconstruction problem (specifically, $u(t)$ is not measured, but is instead modeled by white noise). These works utilize the convolutional model (4.4), requiring $P = I$, and leverage Granger causality and lasso regression techniques to reconstruct their networks. They apply their reconstruction techniques to the reconstruction of financial networks [62] and gene expression regulatory networks [63–65]. In [66], this line of research is continued with a new lasso technique (group lasso regression) while additionally discussing sufficient conditions for consistency in their regressions, with application to gene regulatory networks (showing causal dependencies between gene expressions) and banking (showing causal dependencies between items on a balance sheet). These results are generalized further in [67], which reconstructs networks where more general non-linear dynamics govern Q.

In [68], the authors perform a comparison of Granger-causality-based reconstruction techniques (such as [62–67]) and Bayesian inference models (such as [50–52]), finding that the Bayesian inference performs better when there is little data available, and Granger-causality-based reconstruction performs better when there is more data since Granger causality is more sensitive to a small value of the interactions.

In [2, 12, 20], a more general time-domain reconstruction procedure (allowing P to be general) was introduced. It leveraged the results on the necessary and sufficient identifiability conditions developed previously for the frequency-domain algorithms, providing a framework for encoding a priori knowledge about a network. In [12, 18], variations of the time-domain reconstruction algorithm are presented that reconstruct approximations of the network to solve the blind reconstruction problem, with applications to the stock market and social media respectively. Section 4.4 below builds upon these works (the non-blind case) by generalizing even further and strengthening the identifiability results.

4.2.2 Topological Reconstruction

The topological reconstruction problem takes data and additional a priori assumptions about a network and seeks to recover the structure, but not the dynamics, of the network. In other words, it seeks to build a graph where the nodes are the inputs and outputs to the system and an edge is present if and only if the causal dynamics directly connecting those nodes (and independent of all other nodes) is non-zero. Topological reconstruction comes in two flavors: undirected (where edges are undirected, hence the graph represents direct correlations between manifest variables) and directed (where edges are directed, hence the graph represents the direct causations between manifest variables). Most research on topological reconstructed focuses on the undirected case.

State-Space-Model-Based Approaches

In the state-space-model-based approaches, the topology of interest is the (unweighted) graphical structure of A.

In [69], a state-space model where $C = I$ and B and D are both zero are considered (i.e., they assume an autonomous state space model with full-state measurements), while noting that full-state measurement is necessary for reconstruction[1]. The work then considers

[1]This result was also shown and generalized in [70] for the full reconstruction problem.

various classes of A (such as those that model consensus networks and the general problem where $A \in \mathbb{R}$ can be anything) and uses constrained Lyapunov functions to provide algorithms that reconstruct the undirected topology of those classes provided that we meet the given necessary and sufficient conditions.

DSF-Based Approaches

In the DSF-based topological reconstruction works, the topology of interest is the (unweighted) graphical structure of Q. A restricted DSF (Q, P) with $P = I$ is typically considered. As such, these networks are target-specific (see Definition 4.3.11 below) and the necessary identifiability conditions established in below are satisfied (see Section 4.3.3). The works typically also assume that the inputs u are generated by independent random processes, which happens to satisfy the necessary data informativity conditions for the time-domain network reconstruction algorithm we present in Section 4.4 (see Section 4.4.3 below). In addition, these works often assume that the inputs are unmeasured; thus the reconstruction is *blind*.

One of the earliest topological reconstruction works is contained in [55]. This work focuses primarily on the full reconstruction problem; however, it also shows that the necessary and sufficient identifiability conditions required to reconstruct the Boolean network (i.e., the topology) are the same as for the full network reconstruciton problem.

An early solution was presented in [7]. In that work, a Wiener-filter-based approach is used to recover the undirected topology of Q in the restricted case where the topology of Q is a tree. This approach is applied to stock market data, showing that securities near each other in the tree tend to be within the same industry. In [71], this approach is generalized, allowing the reconstruction of undirected topologies of Q where Q is described by a self-kin topology (which includes tree, polytree, and ring structures).

The works [72, 73] assume that the network of interest has a polytree structure and use Wiener-filters to solve the undirected topological reconstruction problem. In [72], this

methodology is used to create a network among cities in Europe using temperature data, showing that the network is related to the geographic locations of the city. It was also used to recover the topology of an undirected consensus network [74] and a power grid [75]. In [73], the authors assume that we only measure some subset of the nodes in the network (i.e., in the language of this dissertation, they assume that we measure some immersion of the network), showing that if the degree of each node that has been abstracted away is at least three, with out-degree of at least two, then the topology of the base network can be recovered using only measurements from the immersion.

The work [76] contains a more general DSF-based topological reconstruction algorithms. Causal estimators based on Granger causality are combined with the previous techniques to solve the undirected topological reconstructed problem. It requires no restrictions on the topology of Q. Furthermore, it allows the dynamics of the true network to be proper[2], and not necessarily strictly proper, so long as there exists at least one strictly proper edge in every cycle in the network. This condition is a sufficient condition for well-posedness[3], and the well-posedness results contained in Chapters 2 and 3 could be used to generalize this algorithm further.

In [77], the authors use methods from the compressive sensing literature to generalize even further. They provide an algorithm to solve the directed topology problem over a DSF where each entry in Q is assumed to be a finite impulse response[4]. The earlier work [78] likewise uses compressive sensing to reconstruct the directed topology of a network. It performs its reconstruction in the time domain (unlike the previously-cited topological

[2]The work [76] claims to be the first to consistently reconstruct a network when entries in Q are proper and not necessarily strictly proper. This claim, however, is untrue. The DSF-based frequency-domain full reconstruction algorithm discussed in Sections 4.2.1 and 4.3 were introduced nearly a decade earlier and are capable of reconstructing general proper networks without even requiring assumptions on well-posedness (see Example 4.3.12). Furthermore, the single-module identification algorithms discussed in Section 4.2.3 were introduced earlier and can also handle general proper networks.

[3]The condition is sufficient since this guarantees that every principle minor of $Q(\infty)$ is zero, implying that every principle minor of $I - Q(\infty)$ is non-singular. Since $I - Q(\infty)$ is a principal minor of itself, by Theorem 2.5.8, this implies that the network is well-posed. Furthermore, as discussed in 3.3.3 and 3.3.4, this condition is also sufficient to guarantee that every immersion of this network is both representable and well-posed.

[4]We require a similar assumption in our time-domain network reconstruction algorithm in Section 4.4.

reconstruction works, which use the frequency domain) assuming that edges in Q are finite impulse responses, and is capable of reconstructing the topology so long as the network is sufficiently sparse. In [79], a similar problem setup is used but within a probabilistic framework. A Bayesian approach (utilizing the stable-spline kernel estimator) is introduced to solve the directed topological reconstruction problem.

In [80], the authors seek to discover the directed causal interactions between a pair of nodes by using the *directed information* measure, which is based on Granger causality. This approach is used to uncover the causal relationships between measurements of the primary motor cortex of a monkey performing target reaching tasks.

4.2.3 Single Module Identification

Single-module identification assumes that the topology of some dynamic network is known, but that the dynamics are unknown. It seeks to identify the dynamics of a single link (AKA a module) in this network while leaving the dynamics of the remaining links unknown. Such methodology can result in more efficient approach (compared to full network reconstruction and topological reconstruction) regarding the number of signals that we need to measure to perform the identification. Specifically, the other problems need to measure all inputs and outputs in a dynamic system, whereas single-link identification only needs some subset.

All works within this section use the DSF as their model of a networked system. Most of these works split the inputs into two components. The first is the p-dimensional signal vector r, which defines the signals that are directly manipulated by the user. The second is the p-dimensional signal vector e, which is defined by white noise. With $H \in RP^{p\times p}$ as a proper and diagonal matrix, $v = He$ becomes a model of *process noise* and models a stationary stochastic process with a rational spectral density and where each noise signal is

independent of the others. With these definitions, the DSF becomes

$$y = Qy + \begin{bmatrix} R & H \end{bmatrix} \begin{bmatrix} r \\ e \end{bmatrix}, \tag{4.6}$$

where $Q, R \in RP^{p \times p}$. Sometimes (such as in [46]) the problem is formulated such that $R = I$.

These works often allow P and Q to be proper and not necessarily strictly proper. In consequence, they typically also assume a strong condition on well-posedness, requiring that every principal minor of $(I - Q(\infty))$ be non-zero. See Sections 3.3.3 and 3.3.4 for a discussion on why these conditions are sufficient, but not necessary, for well-posedness. The conditions guarantee that every node and hollow abstraction (and hence every immersion) are also well-posed, a feature that is desirable for several of the techniques used in these works.

We will define the *identifiability index* in Section Definition 4.3.4, but in short, this index specifies that it is necessary to know $p^2 - p$ modules in the DSF in order to perform a full reconstruction. Since H is diagonal by construction, we know that the $p^2 - p$ off-diagonal entries of H are zero; thus (4.6) meets the necessary conditions required for full reconstruction.

The single-module identification problem was first posed in [46], which extends the direct method, two-stage method, and joint-io method (which are all classic closed-loop prediction error methods) to solve this problem. In [81], these results were extended to be robust to noise on the sensors. The works [82, 83] combine Bayesian estimation with the prediction error methods used previously to solve this problem. The works [28, 44, 45] are further extensions to these methodologies and discuss conditions on identifiability (typically referring to which signals need to be measured as opposed to which modules need to be known) in the single-module setting where the network has a more general structure than (4.6).

The works [34, 36] introduce the concept of immersion (which we discuss in Section 3.5 and use this concept to solve the single module-identification problem by ceasing to measure variables not required. See Example 4.3.15 for more details on how immersion is

used to solve the single-module identification problem and how this technique is tied to the full reconstruction of an abstraction of the network. The works [35, 84, 85] follow the same approach, but use a different type of abstraction (known as the *indirect inputs method*) instead of the immersion to solve the problem. In [86, 87], another type of abstraction is performed using graphical modules. Notions of d-separation and the Wiener filter are used to perform the module identification.

Each of the works using abstractions to reduce the measured signal set seeks to find the smallest set of nodes to measure; i.e., in the language of Chapter 3, they seek to minimize $|S|$ in order to successfully identify the module of choice in the network. In [88], the authors note that these works only provide sufficient conditions on this minimal set. They then proceed to combine both the immersion and the indirect inputs method to provide necessary and sufficient conditions over this set. However, [88] still has not provided necessary and sufficient conditions over this set for the identification of a single module when all possible abstraction procedures are considered, and this problem remains open.

4.2.4 Contributions

In this chapter, we present a frequency-domain algorithm and a time-domain algorithm to solve the full network reconstruction problem (i.e., to recover both the structure and dynamics of some dynamic network). The frequency-domain algorithm is based directly on [19, 37, 43, 55, 59], and assumes that all measurements are perfect (not noisy). We also present the following novel contributions beyond this previous work:

- Generalization of the algorithm to reconstruct DNFs, including the necessary and sufficient identifiability conditions required for such reconstruction.
- Examples showing that the original assumptions that the network is strictly proper are overly strict; the procedure functions properly for proper networks and networks where the edges are defined by arbitrary fields (Examples 4.3.12 and 4.3.14).

- An example connecting this full reconstruction methodology to the single-module reconstruction methodology presented in [34, 36] (Example 4.3.15).
- Examples connecting this full reconstruction methodology to the notions of immersions and abstractions (Examples 4.3.15 and 4.3.16).

The time-domain reconstruction algorithm is based directly on [12, 20]. In addition to those works, we present the following novel contributions:

- Generalization of the algorithm to reconstruct DNFs.
- Generalization of the algorithm to reconstruct proper (and not necessarily strictly proper) networks.
- Presentation of the necessary quantity of data required for reconstruction (Lemma 4.4.5).
- Presentation of necessary data informativity conditions for reconstruction (Section 4.4.3).

4.3 Network Reconstruction in the Frequency Domain

While the focus of this work is the development of an algorithm to reconstruct proper networks in the time domain, we first turn our attention to an algorithm to reconstruct networks in the frequency domain. We do this since the frequency-domain reconstruction algorithm is more straightforward to describe and provides keen insights into the time-domain algorithm. Furthermore, the contrast between the two algorithms will provide an understanding of the advantages and disadvantages of each.

4.3.1 Problem Formulation

Let (W, V) be some unknown well-posed DNF such that $W \in RP^{p\times p}$ and $V \in RP^{p\times m}$, with corresponding transfer function $G = (I - W)^{-1}V \in RP^{p\times m}$. Let G be known (i.e., G could have been learned from data using a standard system identification technique). The problem is to recover the *unique* (W, V) given G.

REMARK 4.3.1: For simplicity of exposition, we assume that the DNF takes its values from the set RP of proper rational functions. This assumption is strong in that is not necessary for this algorithm; all results that we outline in this section will still hold if the DNF takes its values from any field. We demonstrate this generality using the examples in Section 4.3.4.

We also acknowledge that the set RP is not a field (if $x \in RP$ is strictly proper, then $x^{-1} \notin RP$ is improper). As such, the notion of well-posedness (see Chapter 2) is a sufficient condition to ensure that the transfer function G also takes its values from RP. Representability (see Chapter 3) becomes important too if we choose to reconstruct an abstraction of a network rather than the original network as representability ensures that the abstraction also takes its values from RP. If we do not require closure, or if our DNF and transfer function take their values from fields (such as the set of all rational polynomials), then we do not need to assume either representability or well-posedness.

Unfortunately, it is well-known that for any G, (W, V) is not unique (in general, there is an infinite number of (W, V) corresponding to any G). Thus, this problem is ill-posed, and we cannot solve it without additional a priori knowledge about (W, V), which we call the *identifiability conditions*. In the next section (Section 4.3.2), we will give a description of these identifiability conditions as well as the necessary and sufficient conditions required for reconstruction. In short, however, the problem is as follows:

> Given the transfer function G and the necessary and sufficient identifiability conditions required for reconstruction, find the unique (W, V) such that $G = (I - W)^{-1}V$.

4.3.2 Methodology

To solve the active reconstruction problem, note that

$$(I - W)G = V \quad \Longrightarrow \quad G = \begin{bmatrix} W & V \end{bmatrix} \begin{bmatrix} G \\ I_p \end{bmatrix}. \tag{4.7}$$

By transposing both sides, we get

$$G' = \begin{bmatrix} G' & I_m \end{bmatrix} \begin{bmatrix} W' \\ V' \end{bmatrix}. \tag{4.8}$$

Let $\vec{g} \in RP^{pm}$ be the vectorization of G in row-major order; i.e., let

$$\vec{g} = \begin{bmatrix} g_{11} & \dots g_{1m} & \dots & g_{p1} & \dots & g_{pm} \end{bmatrix}'. \tag{4.9}$$

Likewise let $\vec{w} \in RP^{p^2}$ and $\vec{v} \in RP^{pm}$ be the vectorizations of W and V respectively in row-major order, and define $\theta \in RP^{p^2+pm}$ to be

$$\theta = \begin{bmatrix} \vec{w}' & \vec{v}' \end{bmatrix}'. \tag{4.10}$$

Introduce matrix $L \in RP^{pm \times p^2+pm}$ with

$$L = \begin{bmatrix} G' \oplus \overset{p}{\dots} \oplus G' & I_m \oplus \overset{p}{\dots} \oplus I_m \end{bmatrix} = \begin{bmatrix} G' \oplus \overset{p}{\dots} \oplus G' & I_{pm} \end{bmatrix} \tag{4.11}$$

so that the mapping

$$\vec{g} = L\theta \tag{4.12}$$

from θ to $\vec{g}$ is equivalent to the mapping in (4.8).

Now, note that every entry in $\vec{g}$ and L are known, whereas θ contains precisely the $p^2 + pm$ unknown entries in both W and V. Thus, our problem reduces to this: given $\vec{g}$ and L, find the unique θ such that $\vec{g} = L\theta$.

Lemma 4.3.2: *Let $G \in RP^{p\times m}$ and let $L = [L_G \;\; I_{pm}]$, where*

$$L_G = G' \oplus \overset{p}{\cdots} \oplus G'. \tag{4.13}$$

Then $\operatorname{rank} L = \operatorname{rank} I_{pm} = pm$. ♢

Proof We have that $L \in RP^{pm\times p^2+pm}$; hence, $\operatorname{rank} L \leq \min\{pm, p^2 + pm\} = pm$. Furthermore, all of the columns in I_{pm} are trivially linearly independent; hence $\operatorname{rank} L \geq \operatorname{rank} I_{pm} = pm$. Thus $\operatorname{rank} L = pm$. ■

Corollary 4.3.3: *Let L be defined as above. It is necessary (though not sufficient) to reduce*[5] *the dimension of the domain of L by at least p^2 in order to make L injective (full column rank),* ♢

Proof By Lemma 4.3.2, $\operatorname{rank} L = pm$. Thus, the dimension of the domain of L must necessarily be less than or equal to pm in order to be full-column rank. Since the dimension of the domain of L is $p^2 + pm$, this dimension must necessarily be reduced by at least p^2. ■

In order to solve for θ uniquely, we must replace L with an injective map by reducing the dimension of its domain. Corollary 4.3.3 thus tells us that we must necessarily reduce this dimension by at least p^2 to find a unique θ. We call p^2 the *identifiability index*, which we define formally here.

Definition 4.3.4: Identifiability Index

The *identifiability index* for some transfer function G is the **necessary** amount of structural information one needs to know in order form a bijection between (W, V) and

[5] We defer the discussion of the reduction process to Definition 4.3.5.

G such that $G = (I - W)^{-1}V$. As shown in Corollary 4.3.3, if $G, V \in RP^{p\times m}$ and $W \in RP^{p\times p}$, the identifiability index is p^2.

We also define the *Identifiability Conditions* as the actual structural information required to reconstruct the network.

Definition 4.3.5: Identifiability Conditions

The *identifiability conditions* for some transfer function G is the structural information one needs to know in order to form a bijection between (W, V) and G such that $G = (I - W)^{-1}V$. Since the domain of the network corresponds to unknown entries in (W, V), the identifiability conditions required to reconstruct take the following form:

- Knowledge that some entry in W or V is zero.
- Knowledge that some entry in W or V is a linear combination of the other entries.

The identifiability index (Definition 4.3.4) tells us that we need knowledge (in one of the above forms) of at least p^2 entries of W and V.

REMARK 4.3.6: Previous work treat "identifiability conditions" and "identifiability index" as synonymous (using only the term "identifiability conditions"). However, for clarity, we find it convenient to separate the definitions. The identifiability conditions define a set of information required to reconstruct the network, whereas the identifiability index is a measure of the necessary minimum size of this set.

REMARK 4.3.7: Some previous work (e.g., [19]) often refers to the "identifiability conditions" as "informativity conditions," though others (see, for instance, [19, 43]) use "identifiability conditions" as we do here.

In this chapter, we reserve "informativity" to refer to the richness of the input-output data. For the frequency-domain reconstruction algorithm, we have implicitly assumed that the input-output data is informative; i.e., that the data is rich enough to

recover the correct transfer function G through standard system identification techniques. However, informativity becomes a more important subject in the time-domain reconstruction algorithm since it uses the input-output data rather than reconstructing from a given transfer function.

To encode the identifiability conditions, we introduce a matrix $K \in RP^{p^2+pm\times k}$. Let

$$\hat{\theta} = K^{+}\theta \in RP^{k} \quad \text{and} \quad M = LK \in RP^{pm\times k}, \tag{4.14}$$

where K^{+} is the Moore-Penrose Pseudo Inverse of K (which we never actually need to compute). Thus, the problem is reduced further to finding the unique $\hat{\theta}$ such that

$$\vec{g} = (LK)(K^{+}\theta) = M\hat{\theta}, \tag{4.15}$$

which is solvable if M is injective. Once $\hat{\theta}$ is known, we can uniquely recover θ, containing all values in (W, V), with

$$\theta = K\hat{\theta}. \tag{4.16}$$

The following result contains the necessary conditions for M to be injective:

Lemma 4.3.8: *Let $L \in RP^{pm\times p^2+pm}$ and $K \in RP^{p^2+pm\times k}$ be defined as above, with $M = LK$. Then M is injective only if $k \leq pm$ and* $\operatorname{rank} K = k$ *(i.e., K is injective).* ♢

Proof Assume, to the contrary, that $k > pm$ or that $\operatorname{rank} K \neq k$ (implying that $\operatorname{rank} K < k$). In the first case where $k > pm$, $M = LK$ cannot be injective (full column rank) since M has $k > pm$ columns and $\operatorname{rank} M \leq \min\{pm, k\} = pm < k$. Now assume that $k \leq pm$ but $\operatorname{rank} K < k$. Then we have that

$$\operatorname{rank} M = \operatorname{rank} LK \leq \min\{\operatorname{rank} L, \operatorname{rank} K\} = \min\{pm, \operatorname{rank} K\} < k. \tag{4.17}$$

Thus, M cannot be full column rank. ■

With these results, we are now prepared to give the necessary **and** sufficient conditions (recall that we have already shown that meeting the identifiability index p^2 is necessary) for reconstructing some network.

Theorem 4.3.9 (Identifiability Conditions): *Let $G \in RP^{p\times m}$ be some transfer function and let (W, V) (with $W \in RP^{p\times p}$ and $V \in RP^{p\times m}$) be a DNF such that $(I-W)^{-1}V = G$. Let $L \in RP^{pm\times p^2+pm}$ be defined as in (4.11), and let $K \in RP^{k\times p^2+pm}$ encode the identifiability conditions (i.e., the structural information known about the network). Then, there is a bijection between G and (W, V) (i.e., (W, V) can be reconstructed uniquely from G) if and only if the following conditions on K hold:*

1. *$M \triangleq LK$ is injective*
2. *$\vec{g} \in \mathcal{R}(M)$, where $\mathcal{R}(M)$ is the range of M* ♢

Proof Observe that M is the mapping from unidentified model parameters $\hat{\theta}$ (recall that all model parameters θ can be recovered uniquely from $\hat{\theta}$) to the known parameters $\vec{g}$ of the transfer function G. The two conditions listed above are well-known to ensure that $\hat{\theta}$ is computed uniquely. ■

Remark 4.3.10: Lemma 4.3.8 is a result about the identifiability conditions encoded by K and provides the conditions necessary for reconstruction. In practice, if K is designed such that these necessary conditions are satisfied, then often (but not always), the necessary and sufficient conditions outlined in Theorem 4.3.9 are also satisfied. This suggests that the work-flow should be to assume or discover enough identifiability conditions to satisfy Lemma 4.3.8, then check to see if the conditions also satisfy Theorem 4.3.9. If Theorem 4.3.9 is not satisfied, then more (or sometimes less) identifiability conditions will need to be encoded and this process repeated.

Thus, with an injective M, we can uniquely compute $\hat{\theta}$ uniquely using either matrix inversion or linear regression (we use least squares here) as follows:

$$\hat{\theta} = \begin{cases} M^{-1}\vec{g} & \text{if } \operatorname{rank} M = k = pm \\ (M'M)^{-1}M'\vec{g} & \text{if } \operatorname{rank} M = k < pm \end{cases} \tag{4.18}$$

Recall from (4.16) that $\theta = K\hat{\theta}$ is unique. Furthermore, θ encodes all values in W and V, and so we can uniquely extract our reconstructed DNF from θ, thus solving our problem.

In summary, the frequency-domain network reconstruction algorithm proceeds according to Algorithm 1.

4.3.3 Special Cases: Reconstructing a DSF and Target Specificity

Before this work, all of the network reconstruction literature presents methods to reconstruct DSFs instead of DNFs. Recall that a DSF (Q, P) is a special case of a DNF (W, V) where Q is hollow, meaning that all diagonal entries of Q are known to be zero. As such, we already know p values of (Q, P), reducing the identifiability index to $p^2 - p$.

In the further special case where we are reconstructing a DSF such $p = m$ (i.e., G and P are both square), a common choice of identifiability conditions is called *target specificity*, which we define formally below.

Definition 4.3.11: Target Specificity

Assume that $G, Q, P \in RP^{p\times p}$ with Q hollow. *Target Specificity* is the additional assumption that P is diagonal, meaning that we know that the $p^2 - p$ off-diagonal entries of P are all zero.

Since target specificity further specifies $p^2 - p$ entries in (Q, P), we have met the necessary conditions (the identifiability index) for reconstruction.

Algorithm 1 Frequency Domain Network Reconstruction

procedure ReconstructFreq(G, K) ▷ Find (W, V) from G given identifiability K
  $p, m \leftarrow \text{Shape}(G)$
  $\_, k \leftarrow \text{Shape}(K)$
  assert($G \in RP^{p \times m}$) ▷ Can relax RP to any field $\mathcal{F}$
  assert($K \in RP^{p^2+pm \times k}$) ▷ Can relax RP to any field or restrict to $\mathbb{R}$
  assert($k \leq pm$ and $\operatorname{rank} K = k$) ▷ Lemma 4.3.8
  $\vec{g} \leftarrow [G_{11},\ G_{12},\ \cdots, G_{pm}]' \in RP^{pm}$
  $L_G \leftarrow G' \oplus \overset{p}{\cdots} \oplus G' \in RP^{pm \times p^2}$
  $L \leftarrow [L_G\ \ I_{pm}] \in RP^{pm \times p^2+pm}$
  $M \leftarrow LK \in RP^{pm \times k}$
  assert($\operatorname{rank} M = k$ and $\vec{g} \in \mathcal{R}(M)$) ▷ Theorem 4.3.9
  if $k = pm$ **then** ▷ M is square, just invert
    $\hat{\theta} \leftarrow M^{-1}\vec{g} \in RP^{pm}$
  else ▷ M is "tall," use least squares
    $\hat{\theta} \leftarrow (M'M)^{-1}M'\vec{g} \in RP^{pm}$
  end if
  $\theta \leftarrow K\hat{\theta} \in RP^{p^2+pm}$
  $W \leftarrow \begin{bmatrix} \theta_1 & \cdots & \theta_p \\ \vdots & \ddots & \vdots \\ \theta_{p^2-p+1} & \cdots & \theta_{p^2} \end{bmatrix} \in RP^{p \times p}$ ▷ Extract W from θ
  $V \leftarrow \begin{bmatrix} \theta_{p^2+1} & \cdots & \theta_{p^2+m} \\ \vdots & \ddots & \vdots \\ \theta_{p^2+pm-m+1} & \cdots & \theta_{p^2+pm} \end{bmatrix} \in RP^{p \times m}$ ▷ Extract V from θ
  return W, V ▷ Return the reconstructed DNF
end procedure
procedure Shape(M) ▷ $M \in \mathcal{F}^{m \times n}$ for any field $\mathcal{F}$
  return m, n ▷ Rows, Cols in M
end procedure

4.3.4 Numeric Examples

We now demonstrate the frequency-domain reconstruction algorithm with several examples, each of which illustrates the various capabilities of this algorithm.

Example 4.3.12: A Target-Specific, Proper, and Ill-Posed DSF

This example is, perhaps, the simplest example of reconstruction. Reconstruction of target-specific strictly proper DSFs have long been demonstrated; however, the algorithm was always capable of reconstructing proper networks as well, even if they are ill-posed (see Chapter 2 for definitions of well-posed and ill-posed networks). We demonstrate that here.

Let

$$Q = \begin{bmatrix} 0 & \frac{z+2}{z+1} & 0 \\ 0 & 0 & \frac{z+3}{z+4} \\ \frac{(z+1)(z+5)}{(z+2)(z+3)} & \frac{1}{z^2+2} & 0 \end{bmatrix}, \qquad P = \begin{bmatrix} \frac{z+4}{z+1} & 0 & 0 \\ 0 & \frac{1}{z^2+2} & 0 \\ 0 & 0 & \frac{z+6}{z+3} \end{bmatrix} \tag{4.19}$$

Note that Q is hollow, meaning that we have a DSF. Note also that P is diagonal, meaning that we can assume target specificity. Note that neither Q nor P are strictly proper. Note that $p = m = 3$. Finally note that

$$I - Q(\infty) = \begin{bmatrix} 1 & -1 & 0 \\ 0 & 1 & -1 \\ -1 & 0 & 1 \end{bmatrix} \tag{4.20}$$

is singular, thus by Theorem 2.5.8, this network is ill-posed. We also have that

$$G = [G_{ij}] = \begin{bmatrix} -\frac{(z+4)\left(z^3+4z^2+z+5\right)}{(z+1)(z^2+z+5)} & -\frac{(z+2)(z+4)}{(z+1)(z^2+z+5)} & -\frac{(z+2)(z+6)\left(z^2+2\right)}{(z+1)(z^2+z+5)} \\ -\frac{(z+4)(z+5)\left(z^2+2\right)}{(z+2)(z^2+z+5)} & -\frac{z+4}{z^2+z+5} & -\frac{(z+6)\left(z^2+2\right)}{z^2+z+5} \\ -\frac{(z+4)^2(z+5)\left(z^2+2\right)}{(z+2)(z+3)(z^2+z+5)} & -\frac{(z+4)\left(z^3+5z^2+3z+13\right)}{(z+3)(z^2+2)(z^2+z+5)} & -\frac{(z+4)(z+6)\left(z^2+2\right)}{(z+3)(z^2+z+5)} \end{bmatrix}. \tag{4.21}$$

Notice that G has improper terms, violating our strong conditions that our rational functions be in RP. This is an artifact of (Q, P) being ill-posed, however, we will still be able to reconstruct since the set RP above can be relaxed to arbitrary fields.

Thus, we have that

$$L = \begin{bmatrix} G' & 0 & 0 & I_3 & 0 & 0 \\ 0 & G' & 0 & 0 & I_3 & 0 \\ 0 & 0 & G' & 0 & 0 & I_3 \end{bmatrix} \quad \text{and} \quad \vec{g} = \begin{bmatrix} -\frac{(z+4)\left(z^3+4z^2+z+5\right)}{(z+1)(z^2+z+5)} \\ -\frac{(z+2)(z+4)}{(z+1)(z^2+z+5)} \\ -\frac{(z+2)(z+6)\left(z^2+2\right)}{(z+1)(z^2+z+5)} \\ -\frac{(z+4)(z+5)\left(z^2+2\right)}{(z+2)(z^2+z+5)} \\ -\frac{z+4}{z^2+z+5} \\ -\frac{(z+6)\left(z^2+2\right)}{z^2+z+5} \\ -\frac{(z+4)^2(z+5)\left(z^2+2\right)}{(z+2)(z+3)(z^2+z+5)} \\ -\frac{(z+4)\left(z^3+5z^2+3z+13\right)}{(z+3)(z^2+2)(z^2+z+5)} \\ -\frac{(z+4)(z+6)\left(z^2+2\right)}{(z+3)(z^2+z+5)} \end{bmatrix}, \tag{4.22}$$

each of which contains known values.

To encode that we are reconstructing a target-specific DSF, choose $K \in RP^{p^2+pm \times pm}$ such that

$$K = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{matrix} (Q_{11} = 0) \\ (Q_{12} = ?) \\ (Q_{13} = ?) \\ (Q_{21} = ?) \\ (Q_{22} = 0) \\ (Q_{23} = ?) \\ (Q_{31} = ?) \\ (Q_{32} = ?) \\ (Q_{33} = 0) \\ (P_{11} = ?) \\ (P_{12} = 0) \\ (P_{13} = 0) \\ (P_{21} = 0) \\ (P_{22} = ?) \\ (P_{23} = 0) \\ (P_{31} = 0) \\ (P_{32} = 0) \\ (P_{33} = ?) \end{matrix} \tag{4.23}$$

Note that this matrix maps Q_{12} to $\hat{\theta}_1$ (since the one corresponding to the Q_{12} row is in the first column), Q_{13} to $\hat{\theta}_2$, etc.

Compute $M = LK$. It can be checked that M is square and non-singular; thus we meet both the necessary and sufficient conditions for reconstruction. Now compute

$$\hat{\theta} = M^{-1}\vec{g} = \begin{bmatrix} \frac{z+2}{z+1} \\ 0 \\ 0 \\ \frac{z+3}{z+4} \\ \frac{(z+1)(z+5)}{(z+2)(z+3)} \\ \frac{1}{z^2+2} \\ \frac{z+4}{z+1} \\ \frac{1}{z^2+2} \\ \frac{z+6}{z+3} \end{bmatrix}. \tag{4.24}$$

Finally, recover the unknown parameters in (Q, P) with

$$\theta = K\hat{\theta} = \begin{bmatrix} Q_{11} \\ Q_{12} \\ Q_{13} \\ Q_{21} \\ Q_{22} \\ Q_{23} \\ Q_{31} \\ Q_{32} \\ Q_{33} \\ P_{11} \\ P_{12} \\ P_{13} \\ P_{21} \\ P_{22} \\ P_{23} \\ P_{31} \\ P_{32} \\ P_{33} \end{bmatrix} = \begin{bmatrix} 0 \\ \frac{z+2}{z+1} \\ 0 \\ 0 \\ 0 \\ \frac{z+3}{z+4} \\ \frac{(z+1)(z+5)}{(z+2)(z+3)} \\ \frac{1}{z^2+2} \\ 0 \\ \frac{z+4}{z+1} \\ 0 \\ 0 \\ 0 \\ \frac{1}{z^2+2} \\ 0 \\ 0 \\ 0 \\ \frac{z+6}{z+3} \end{bmatrix}, \tag{4.25}$$

which gives the reconstructed DSF as

$$Q = \begin{bmatrix} 0 & \frac{z+2}{z+1} & 0 \\ 0 & 0 & \frac{z+3}{z+4} \\ \frac{(z+1)(z+5)}{(z+2)(z+3)} & \frac{1}{z^2+2} & 0 \end{bmatrix}, \qquad P = \begin{bmatrix} \frac{z+4}{z+1} & 0 & 0 \\ 0 & \frac{1}{z^2+2} & 0 \\ 0 & 0 & \frac{z+6}{z+3} \end{bmatrix}. \tag{4.26}$$

Notice that the reconstructed DSF is the same as the original DSF in (4.19).

Example 4.3.13: Exceeding the Identifiability Index

In this example, we will demonstrate the reconstruction of a DSF when $m > p$ and where we have more structural information than necessary to reconstruct. Let

$$Q = \begin{bmatrix} 0 & \frac{1}{z+1} \\ \frac{1}{z+2} & 0 \end{bmatrix}, \qquad P = \begin{bmatrix} \frac{1}{z+3} & 0 & 0 \\ \frac{2z+7}{(z+3)(z+4)} & \frac{1}{z+4} & \frac{1}{z+5} \end{bmatrix} \tag{4.27}$$

be our original unknown DSF that we wish to recover. Thus, the known transfer function is

$$G = (I-Q)^{-1}P = \begin{bmatrix} \frac{(z+2)\left(z^2+7z+11\right)}{(z+3)(z+4)(z^2+3z+1)} & \frac{z+2}{(z+4)(z^2+3z+1)} & \frac{z+2}{(z+5)(z^2+3z+1)} \\ \frac{2(z+1)(z+3)}{(z+4)(z^2+3z+1)} & \frac{(z+1)(z+2)}{(z+4)(z^2+3z+1)} & \frac{(z+1)(z+2)}{(z+5)(z^2+3z+1)} \end{bmatrix}. \tag{4.28}$$

The identifiability index over the DNF requires that we know $p^2 = 4$. We will assume that we know that $Q_{11} = Q_{22} = P_{12} = P_{13} = 0$ and that $P_{21} = P_{11} + P_{22}$. Thus we know $5 > p^2$. To encode this information, let

$$K = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{matrix} (Q_{11} = 0) \\ (Q_{12} = ?) \\ (Q_{21} = ?) \\ (Q_{22} = 0) \\ (P_{11} = ?) \\ (P_{12} = 0) \\ (P_{13} = 0) \\ (P_{21} = P_{11} + P_{22}) \\ (P_{22} = ?) \\ (P_{23} = ?) \end{matrix} \tag{4.29}$$

We also have, from the known values of the transfer function, that

$$L = \begin{bmatrix} G' & 0 & I_3 & 0 \\ 0 & G' & 0 & I_3 \end{bmatrix} \quad \text{and} \quad \vec{g} = \begin{bmatrix} \frac{(z+2)\left(z^2+7z+11\right)}{(z+3)(z+4)(z^2+3z+1)} \\ \frac{z+2}{(z+4)(z^2+3z+1)} \\ \frac{z+2}{(z+5)(z^2+3z+1)} \\ \frac{2(z+1)(z+3)}{(z+4)(z^2+3z+1)} \\ \frac{(z+1)(z+2)}{(z+4)(z^2+3z+1)} \\ \frac{(z+1)(z+2)}{(z+5)(z^2+3z+1)} \end{bmatrix}. \tag{4.30}$$

With this information, we compute

$$M = LK = \begin{bmatrix} \frac{2(z+1)(z+3)}{(z+4)(z^2+3z+1)} & 0 & 1 & 0 & 0 \\ \frac{(z+1)(z+2)}{(z+4)(z^2+3z+1)} & 0 & 0 & 0 & 0 \\ \frac{(z+1)(z+2)}{(z+5)(z^2+3z+1)} & 0 & 0 & 0 & 0 \\ 0 & \frac{(z+2)\left(z^2+7z+11\right)}{(z+3)(z+4)(z^2+3z+1)} & 1 & 1 & 0 \\ 0 & \frac{z+2}{(z+4)(z^2+3z+1)} & 0 & 1 & 0 \\ 0 & \frac{z+2}{(z+5)(z^2+3z+1)} & 0 & 0 & 1 \end{bmatrix} \tag{4.31}$$

It can be verified that M is full column rank and thus meets our necessary and sufficient identifiability conditions. Since M is not square, we use least squares instead of a matrix

inverse so solve for our unique $\hat{\theta}$ and θ. This gives

$$\hat{\theta} = (M'M)^{-1}M'\vec{g} = \begin{bmatrix} \frac{1}{z+1} \\ \frac{1}{z+2} \\ \frac{1}{z+3} \\ \frac{1}{z+4} \\ \frac{1}{z+5} \end{bmatrix} \quad \text{and} \quad \theta = K\hat{\theta} = \begin{bmatrix} Q_{11} \\ Q_{12} \\ Q_{21} \\ Q_{22} \\ P_{11} \\ P_{12} \\ P_{13} \\ P_{21} \\ P_{22} \\ P_{33} \end{bmatrix} = \begin{bmatrix} 0 \\ \frac{1}{z+1} \\ \frac{1}{z+2} \\ 0 \\ \frac{1}{z+3} \\ 0 \\ 0 \\ \frac{2z+7}{(z+3)(z+4)} \\ \frac{1}{z+4} \\ \frac{1}{z+5} \end{bmatrix}. \tag{4.32}$$

And from θ, we recover (4.27) exactly.

Example 4.3.14: A DNF from a General Field

In this example, we demonstrate the reconstruction of a DNF (instead of a DSF) where elements of the DNF are functions of some $z > 0 \in \mathbb{R}$ (instead of rational functions of $z \in \mathbb{C}$). Let (W, V), with

$$W = \begin{bmatrix} \sin(z) & \frac{\log(z)}{z} \\ 0 & \sin(z) - \frac{\log(z)}{z} \end{bmatrix} \quad \text{and} \quad V = \begin{bmatrix} z^2 + 2z + 3 & z \\ \sin(z) & z \end{bmatrix} \tag{4.33}$$

be the unknown DNF we are attempting to reconstruct. Then our known transfer function is given by

$$G = (I - W)^{-1}V$$

$$= \begin{bmatrix} -\frac{\left(z^2+2z+3\right)(-z\sin(z)+z+\log(z))+\log(z)\sin(z)}{(\sin(z)-1)(-z\sin(z)+z+\log(z))} & \frac{z(z\sin(z)-z-2\log(z))}{(\sin(z)-1)(-z\sin(z)+z+\log(z))} \\ \frac{z\sin(z)}{-z\sin(z)+z+\log(z)} & \frac{z^2}{-z\sin(z)+z+\log(z)} \end{bmatrix}. \quad (4.34)$$

To meet our identifiability conditions (we must know $p^2 = 4$ things about the structure of this network), assume that we know that $Q_{21} = 0$, $Q_{22} = Q_{11} - Q_{12}$, $P_{21} = Q_{11}$, and $P_{22} = P_{12}$. To encode this knowledge, let

$$K = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{matrix} (Q_{11} = ?) \\ (Q_{12} = ?) \\ (Q_{21} = 0) \\ (Q_{22} = Q_{11} - Q_{12}) \\ (P_{11} = ?) \\ (P_{12} = ?) \\ (P_{21} = Q_{11}) \\ (P_{22} = P_{12}) \end{matrix} \quad (4.35)$$

From knowledge of G, we have that

$$L = \begin{bmatrix} G' & 0 & I_2 & 0 \\ 0 & G' & 0 & I_2 \end{bmatrix} \quad \text{and} \quad \vec{g} = \begin{bmatrix} -\frac{\left(z^2+2z+3\right)(-z\sin(z)+z+\log(z))+\log(z)\sin(z)}{(\sin(z)-1)(-z\sin(z)+z+\log(z))} \\ \frac{z(z\sin(z)-z-2\log(z))}{(\sin(z)-1)(-z\sin(z)+z+\log(z))} \\ \frac{z\sin(z)}{-z\sin(z)+z+\log(z)} \\ \frac{z^2}{-z\sin(z)+z+\log(z)} \end{bmatrix}. \quad (4.36)$$

Given this, we can compute

$$M = LK = \begin{bmatrix} -\frac{\left(z^2+2z+3\right)(-z\sin(z)+z+\log(z))+\log(z)\sin(z)}{(\sin(z)-1)(-z\sin(z)+z+\log(z))} & \frac{z\sin(z)}{-z\sin(z)+z+\log(z)} & 1 & 0 \\ \frac{z(z\sin(z)-z-2\log(z))}{(\sin(z)-1)(-z\sin(z)+z+\log(z))} & \frac{z^2}{-z\sin(z)+z+\log(z)} & 0 & 1 \\ \frac{-z-\log(z)}{z\sin(z)-z-\log(z)} & \frac{z\sin(z)}{z\sin(z)-z-\log(z)} & 0 & 0 \\ \frac{z^2}{-z\sin(z)+z+\log(z)} & \frac{z^2}{z\sin(z)-z-\log(z)} & 0 & 1 \end{bmatrix}. \quad (4.37)$$

Since M is square and $\det M \neq 0$, we meet the necessary and sufficient identifiability conditions for reconstruction. Thus, we have

$$\hat{\theta} = \begin{bmatrix} \sin(z) \\ \frac{\log(z)}{z} \\ z^2+2z+3 \\ z \end{bmatrix} \implies \theta = K\hat{\theta} = \begin{bmatrix} Q_{11} \\ Q_{12} \\ Q_{21} \\ Q_{22} \\ P_{11} \\ P_{12} \\ P_{21} \\ P_{22} \end{bmatrix} = \begin{bmatrix} \sin(z) \\ \frac{\log(z)}{z} \\ 0 \\ \sin(z) - \frac{\log(z)}{z} \\ z^2+2z+3 \\ z \\ \sin(z) \\ z \end{bmatrix}, \quad (4.38)$$

which yields (4.33) when the values of (W, V) are extracted, as desired.

Example 4.3.15: Reconstructing an Immersion

Suppose that the unknown network that we wish to reconstruct defines the relationship

$$\begin{bmatrix} Y_1 \\ Y_2 \\ Y_3 \end{bmatrix} = \begin{bmatrix} 0 & \frac{1}{z+1} & 0 \\ 0 & 0 & \frac{1}{z+2} \\ \frac{1}{z+3} & \frac{1}{z+4} & 0 \end{bmatrix} \begin{bmatrix} Y_1 \\ Y_2 \\ Y_3 \end{bmatrix} + \begin{bmatrix} \frac{1}{z+5} & 0 & 0 \\ 0 & \frac{1}{z+6} & 0 \\ 0 & 0 & \frac{1}{z+7} \end{bmatrix} \begin{bmatrix} U_1 \\ U_2 \\ U_3 \end{bmatrix} \triangleq QY + PU. \quad (4.39)$$

This implies that the transfer function G of our system defines the relationship

$$\begin{bmatrix} Y_1 \\ Y_2 \\ Y_3 \end{bmatrix} = \begin{bmatrix} \frac{(z+1)(z+3)\left(z^2+6z+7\right)}{(z+5)\left(z^4+10z^3+34z^2+45z+17\right)} & \frac{(z+2)(z+3)(z+4)}{(z+6)\left(z^4+10z^3+34z^2+45z+17\right)} & \frac{(z+3)(z+4)}{(z+7)\left(z^4+10z^3+34z^2+45z+17\right)} \\ \frac{(z+1)(z+4)}{(z+5)\left(z^4+10z^3+34z^2+45z+17\right)} & \frac{(z+1)(z+2)(z+3)(z+4)}{(z+6)\left(z^4+10z^3+34z^2+45z+17\right)} & \frac{(z+1)(z+3)(z+4)}{(z+7)\left(z^4+10z^3+34z^2+45z+17\right)} \\ \frac{(z+1)(z+2)(z+4)}{(z+5)\left(z^4+10z^3+34z^2+45z+17\right)} & \frac{(z+2)\left(z^2+5z+7\right)}{(z+6)\left(z^4+10z^3+34z^2+45z+17\right)} & \frac{(z+1)(z+2)(z+3)(z+4)}{(z+7)\left(z^4+10z^3+34z^2+45z+17\right)} \end{bmatrix} \begin{bmatrix} U_1 \\ U_2 \\ U_3 \end{bmatrix}.$$

Suppose that, while we don't know the dynamics of our network, but we do know its structure. In other words, we do know that the structure of (Q, P), which we denote with $(Q^\star, P^\star)$, is given by

$$Q^\star = \begin{bmatrix} 0 & \star & 0 \\ 0 & 0 & \star \\ \star & \star & 0 \end{bmatrix} \qquad \text{and} \qquad P^\star = \begin{bmatrix} \star & 0 & 0 \\ 0 & \star & 0 \\ 0 & 0 & \star \end{bmatrix}, \tag{4.40}$$

where a $\star$ signifies any non-zero value in RP.

The identifiability index for this DSF is $p^2 - p = 6$. With the knowledge of the structure given above, we know that the six off-diagonal entries of P are zero and that an additional two non-diagonal entries of Q are also zero, giving us more than enough information to meet the necessary conditions for reconstruction. Suppose, however, that we are only given G_S, defined as the first two rows of the transfer function (this happens, for instance, if Y_3 is impossible or expensive to measure). Since we do not know Y_3, we cannot reconstruct our base network; however, we can reconstruct an immersion.

By following the immersion procedure outlined in Chapter 3, we find that the structure $(Q_S^\star, P_S^\star)$ of the immersion (Q_S, P_S) is given by

$$Q_S^\star = \begin{bmatrix} 0 & \star \\ \star & 0 \end{bmatrix} \qquad \text{and} \qquad P_S^\star = \begin{bmatrix} \star & 0 & 0 \\ 0 & \star & \star \end{bmatrix}. \tag{4.41}$$

We note that in (4.40), a $\star$ represented a non-zero value in RP by construction. However, in (4.41, a $\star$ could be zero as well. However, a $\star$ will only be zero if there is an exact

cancellation in the immersion process, which almost never happens. Nonetheless, this issue is not significant as we can reconstruct the immersion, regardless of whether any given $\star$ is zero or not, provided that we meet the necessary and sufficient conditions for reconstruction.

For this immersion, the identifiability index is 2 (or 4 if this is a DNF, though we add knowledge that $Q_{11} = 0$ and $Q_{22} = 0$ into our identifiability conditions). Since we know the structure of the immersion, we can encode our knowledge that $P_{12} = 0$ and $P_{21} = 0$ in K. We could also encode knowledge that $P_{13} = 0$; however, this extra information is unnecessary and the least squares solution tends to run more slowly than the matrix inversion, and so we do not encode this information. We have

$$K = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{matrix} (Q_{11} = 0) \\ (Q_{12} = ?) \\ (Q_{21} = ?) \\ (Q_{22} = 0) \\ (P_{11} = ?) \\ (P_{12} = 0) \\ (P_{13} = ?) \\ (P_{21} = 0) \\ (P_{22} = ?) \\ (P_{23} = ?) \end{matrix} \tag{4.42}$$

We also have, from the known values of the transfer function G_S (which, again, is the first two rows of the transfer function G), that

$$L = \begin{bmatrix} G_S' & 0 & I_3 & 0 \\ 0 & G_S' & 0 & I_3 \end{bmatrix} \quad \text{and} \quad \vec{g} = \begin{bmatrix} \frac{(z+1)(z+3)\left(z^2+6z+7\right)}{(z+5)(z^4+10z^3+34z^2+45z+17)} \\ \frac{(z+2)(z+3)(z+4)}{(z+6)(z^4+10z^3+34z^2+45z+17)} \\ \frac{(z+3)(z+4)}{(z+7)(z^4+10z^3+34z^2+45z+17)} \\ \frac{(z+1)(z+4)}{(z+5)(z^4+10z^3+34z^2+45z+17)} \\ \frac{(z+1)(z+2)(z+3)(z+4)}{(z+6)(z^4+10z^3+34z^2+45z+17)} \\ \frac{(z+1)(z+3)(z+4)}{(z+7)(z^4+10z^3+34z^2+45z+17)} \end{bmatrix}. \tag{4.43}$$

With this information, we compute

$$M = LK = \begin{bmatrix} \frac{(z+1)(z+4)}{(z+5)(z^4+10z^3+34z^2+45z+17)} & 0 & 1 & 0 & 0 & 0 \\ \frac{(z+1)(z+2)(z+3)(z+4)}{(z+6)(z^4+10z^3+34z^2+45z+17)} & 0 & 0 & 0 & 0 & 0 \\ \frac{(z+1)(z+3)(z+4)}{(z+7)(z^4+10z^3+34z^2+45z+17)} & 0 & 0 & 1 & 0 & 0 \\ 0 & \frac{(z+1)(z+3)\left(z^2+6z+7\right)}{(z+5)(z^4+10z^3+34z^2+45z+17)} & 0 & 0 & 0 & 0 \\ 0 & \frac{(z+2)(z+3)(z+4)}{(z+6)(z^4+10z^3+34z^2+45z+17)} & 0 & 0 & 1 & 0 \\ 0 & \frac{(z+3)(z+4)}{(z+7)(z^4+10z^3+34z^2+45z+17)} & 0 & 0 & 0 & 1 \end{bmatrix}. \tag{4.44}$$

It can be verified that M is non-singular and thus meets our necessary and sufficient identifiability conditions. Thus, we have that

$$\hat{\theta} = M^{-1}\vec{g} = \begin{bmatrix} \frac{1}{z+1} \\ \frac{z+4}{(z+3)(z^2+6z+7)} \\ \frac{1}{z+5} \\ 0 \\ \frac{(z+2)(z+4)}{(z+6)(z^2+6z+7)} \\ \frac{z+4}{(z+7)(z^2+6z+7)} \end{bmatrix} \quad \text{and} \quad \theta = K\hat{\theta} = \begin{bmatrix} [Q_S]_{11} \\ [Q_S]_{12} \\ [Q_S]_{21} \\ [Q_S]_{22} \\ [P_S]_{11} \\ [P_S]_{12} \\ [P_S]_{13} \\ [P_S]_{21} \\ [P_S]_{22} \\ [P_S]_{33} \end{bmatrix} = \begin{bmatrix} 0 \\ \frac{1}{z+1} \\ \frac{z+4}{(z+3)(z^2+6z+7)} \\ 0 \\ \frac{1}{z+5} \\ 0 \\ 0 \\ 0 \\ \frac{(z+2)(z+4)}{(z+6)(z^2+6z+7)} \\ \frac{z+4}{(z+7)(z^2+6z+7)} \end{bmatrix}. \quad (4.45)$$

Notice that if we let $S = \{1, 2\}$, which encodes that we can measure Y_1 and Y_2, but not Y_3, then the immersion $(Q_S, P_S) = \mathcal{I}(Q, P \mid S)$ is given by

$$Q_S = \begin{bmatrix} 0 & \frac{1}{z+1} \\ \frac{z+4}{(z+3)(z^2+6z+7)} & 0 \end{bmatrix} \quad \text{and} \quad P_S = \begin{bmatrix} \frac{1}{z+5} & 0 & 0 \\ 0 & \frac{(z+2)(z+4)}{(z+6)(z^2+6z+7)} & \frac{z+4}{(z+7)(z^2+6z+7)} \end{bmatrix}. \quad (4.46)$$

Notice that (4.46) is precisely the DSF extracted from (4.45). Thus our network reconstruction technique is capable of reconstructing an immersion when not all input-output data cannot be measured.

In Section 4.2.3, we describe single-module identification as a different problem in network reconstruction, which is ideal for some problems as it does not require expensive measurements of all outputs to the system. This example suggests the technique that is used to perform single-module identification in [34, 36, 88].

Some additional steps, however, are necessary to perform a single module identification. As we did above, we start by assuming knowledge of the structure (but not the dynamics) of some system. The problem is to identify the dynamics of one (or some subset) of the non-zero entries (which are called modules) in Q and/or P. However, instead of choosing some arbitrary immersion, we select an immersion such that the module(s) we wish to identify is invariant under the immersion process. Thus, when we identify the complete immersion, we can read off the dynamics of the desired module(s) directly from the reconstructed network. The problem then becomes to select an S (specifying our immersion) that accomplishes this such that $|S|$ is minimized.

Example 4.3.16: Assuming a DSF When Reconstructing a DNF

Suppose that the unknown DNF that we wish to reconstruct is given by

$$W = \begin{bmatrix} \frac{1}{z+1} & \frac{1}{z+2} \\ \frac{1}{z+3} & \frac{1}{z+4} \end{bmatrix} \qquad \text{and} \qquad V = \begin{bmatrix} \frac{1}{z+5} & 0 \\ 0 & \frac{1}{z+6} \end{bmatrix}, \tag{4.47}$$

and suppose that the only information we have about the network is that $V_{12} = V_{21} = 0$. The identifiability index for this DNF is 4; however, we only know 2, and so we do not meet the necessary conditions for reconstruction. Suppose, however, that we assume that we have a DSF instead of a DNF (i.e., we make the additional and false assumption that $W_{11} = W_{22} = 0$), which allows us to meet the necessary conditions. As we will show in this example, we will not reconstruct the true network; however, the network we do reconstruct is still meaningful.

We have that

$$G = (I - W)^{-1}V = \begin{bmatrix} \frac{(z+1)(z+2)(z+3)^2}{(z+5)(z^4+8z^3+20z^2+13z-4)} & \frac{(z+1)(z+3)(z+4)}{(z+6)(z^4+8z^3+20z^2+13z-4)} \\ \frac{(z+1)(z+2)(z+4)}{(z+5)(z^4+8z^3+20z^2+13z-4)} & \frac{z(z+2)(z+3)(z+4)}{(z+6)(z^4+8z^3+20z^2+13z-4)} \end{bmatrix}, \tag{4.48}$$

and assume that this transfer function is known. This gives

$$\vec{g} = \begin{bmatrix} \frac{(z+1)(z+2)(z+3)^2}{(z+5)(z^4+8z^3+20z^2+13z-4)} \\ \frac{(z+1)(z+3)(z+4)}{(z+6)(z^4+8z^3+20z^2+13z-4)} \\ \frac{(z+1)(z+2)(z+4)}{(z+5)(z^4+8z^3+20z^2+13z-4)} \\ \frac{z(z+2)(z+3)(z+4)}{(z+6)(z^4+8z^3+20z^2+13z-4)} \end{bmatrix} \tag{4.49}$$

and

$$L = \begin{bmatrix} G' & 0 & I_2 & 0 \\ 0 & G' & 0 & I_2 \end{bmatrix}. \tag{4.50}$$

To encode our knowledge that $V_{21} = V_{12} = 0$ and our false assumption that $W_{11} = W_{22} = 0$, let

$$K = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{matrix} (W_{11} = 0) \\ (W_{12} = ?) \\ (W_{21} = ?) \\ (W_{22} = 0) \\ (V_{11} = ?) \\ (V_{12} = 0) \\ (V_{21} = ?) \\ (V_{22} = 0) \end{matrix} \tag{4.51}$$

This gives

$$M = LK = \begin{bmatrix} \frac{(z+1)(z+2)(z+4)}{(z+5)(z^4+8z^3+20z^2+13z-4)} & 0 & 1 & 0 \\ \frac{z(z+2)(z+3)(z+4)}{(z+6)(z^4+8z^3+20z^2+13z-4)} & 0 & 0 & 0 \\ 0 & \frac{(z+1)(z+2)(z+3)^2}{(z+5)(z^4+8z^3+20z^2+13z-4)} & 0 & 0 \\ 0 & \frac{(z+1)(z+3)(z+4)}{(z+6)(z^4+8z^3+20z^2+13z-4)} & 0 & 1 \end{bmatrix}. \tag{4.52}$$

We have that M is square and non-singular, thus we meet the necessary and sufficient conditions for reconstruction and can compute (noting that, since we are extracting into a DSF, we label θ with Q and P instead of our true W and V):

$$\hat{\theta} = M^{-1}\vec{g} = \begin{bmatrix} \frac{z+1}{z(z+2)} \\ \frac{z+4}{(z+3)^2} \\ \frac{z+1}{z(z+5)} \\ \frac{z+4}{(z+3)(z+6)} \end{bmatrix} \quad \text{and} \quad \theta = \begin{bmatrix} Q_{11} \\ Q_{12} \\ Q_{21} \\ Q_{22} \\ P_{11} \\ P_{12} \\ P_{21} \\ P_{22} \end{bmatrix} = \begin{bmatrix} 0 \\ \frac{z+1}{z(z+2)} \\ \frac{z+4}{(z+3)^2} \\ 0 \\ \frac{z+1}{z(z+5)} \\ 0 \\ 0 \\ \frac{z+4}{(z+3)(z+6)} \end{bmatrix}. \tag{4.53}$$

This gives our reconstructed DSF as

$$Q = \begin{bmatrix} 0 & \frac{z+1}{z(z+2)} \\ \frac{z+4}{(z+3)^2} & 0 \end{bmatrix} \quad \text{and} \quad P = \begin{bmatrix} 0 & \frac{z+1}{z(z+2)} \\ \frac{z+4}{(z+3)^2} & 0 \end{bmatrix}. \tag{4.54}$$

Notice that, as expected, (Q, P) is not equal to (W, V). However, recall from Chapter 3 that the hollow abstraction $\mathcal{H}(W, V)$ of some DNF is a DSF where we have set the diagonal entries to zero while preserving the input-output behavior. This is also what we have done with our false assumption that $W_{11} = W_{22} = 0$, and it can be checked that $(Q, P) = \mathcal{H}(W, V)$.

In Chapter 3, we noted one connection between abstractions and network reconstruction and that is that an abstraction is a network that can be reconstructed with fewer identifiability conditions (meaning a lower identifiability index) than the original network.

This example and Example 4.3.15 suggests a second and deeper connection between network reconstruction and abstractions. This connection is that **identifiability conditions, together with the set of signals that are measured, specify an abstraction** of the true network we desire to reconstruct. In the previous example, we reconstructed an immersion by ceasing to measure one of the outputs of the system. In this example, we reconstructed a hollow abstraction by assuming that the diagonal entries of W were zero, even though they were not zero in the original network. Sometimes we do not have enough information to meet the identifiability index required for some network; however, we can use abstractions to relax this condition, so long as we preserve some set of desired properties about the network.

4.4 Network Reconstruction in the Time Domain

We now turn our attention to an algorithm to reconstruct a discrete-time network in the time domain. Such an algorithm is convenient since, for many applications, the input-output data we measure and record are measurements taken at discrete time points, and this algorithm is designed to reconstruct a network directly from such data.

Remark 4.4.1: The works [12, 20] refer as the time-domain reconstruction algorithm as "passive network reconstruction." The term "passive" refers to the problem of reconstructing a network where inputs are observed but not controlled, and the time-domain reconstruction algorithm was first developed to solve that problem. In this chapter, we note that both the frequency- and the time-domain reconstruction algorithms are capable of reconstructing the network–regardless of whether or not we control the inputs. However, care must be taken to ensure that the input-output data is informative enough to reconstruct the network (see Theorem 4.4.7 for necessary data informativity conditions on a common subclass of problems).

4.4.1 Problem Formulation

Let $(W(z), V(z))$ be a discrete-time DNF (with transfer function $G(z) = (I - W(z))^{-1}V$). Let $W(t)$, $V(t)$, and $G(t)$ be the impulse responses of $W(z)$, $V(z)$, and $G(z)$ found by taking the inverse $\mathcal{Z}$-transform. Suppose that this system is defined such that

- Every $W_{ij} \in RP$ and every $V_{ij} \in RP$
- (W, V) is well-posed (which implies that every $G_{ij} \in RP$)
- G is bounded-input, bounded-output (BIBO) stable[6]
- There exists a finite r such that every $W_{ij}(t) \approx 0$ and $V_{ij}(t) \approx 0$ for every $t > r$
- All initial conditions are zero

Suppose also that $W(z)$, $V(z)$, $G(z)$, $W(t)$, $V(t)$, and $G(t)$ are unknown.

REMARK 4.4.2: The list of constraints above highlights the major disadvantage of using the time-domain reconstruction algorithm over the frequency-domain algorithm: the set of networks that it is capable of reconstructing is significantly smaller than with the frequency-domain algorithm (which does not impose any of the above constraints on the networks). Furthermore, this algorithm is not solving a new problem; the frequency-domain algorithm paired with standard system identification already provides a solution to the problem posed in this section.

That said, there are many practical advantages to using this algorithm over the frequency-domain algorithm. We defer a discussion of these advantages to Section 4.4.4.

Suppose that we are given input data $u(t)$ and output data $y(t) = G(t) * u(t) = W(t) * y(t) + V(t) * u(t)$ (where $*$ is the convolution operator) for $t = 0, 1, \ldots, T$ with $T \geq \frac{k}{p}(r+1) - 1$ (where $k \leq pm$ is the number of unknown entries in W and V that we must

[6]BIBO stability on our transfer function may be a strong and unnecessary condition, but ensures that our inputs don't become negligibly small with respect to our outputs, which leads to uninformative data preventing us from reconstructing.

reconstruct; we will discuss why this limitation on the amount of data exists in Lemma 4.4.5). The problem is to find the unique $(W(t), V(t))$ that generated the data.

REMARK 4.4.3: Previous formulations in [12, 20] seek to recover the unique frequency-domain representation $(W(z), V(z))$ instead of the time-domain representation $(W(t), V(t))$. In those works, the impulse responses for $(W(t), V(t))$ are recovered first. Each $W_{ij}(t)$ and $V_{ij}(t)$ is then projected (using a non-convex optimization algorithm) into a space which was called the convolutional form. Using the definitions for the $\mathcal{Z}$-transform and the inverse $\mathcal{Z}$-transform, the works above show that this space is equivalent to a restricted set of proper rational functions of a fixed order and where the rational function could be decomposed using a partial fraction decomposition into a sum of rational functions with degree 0 in the numerator and degree 1 in the denominator. From this convolutional form, it was a simple matter to convert back into the frequency domain.

For this work, we consider the transformation into the frequency domain to be a separate problem from network reconstruction and do not discuss it further.

As with the frequency-domain reconstruction problem, this problem is impossible to solve without additional a priori information about the structure of the network. However, we will show in this section that it can be solved with precisely the same kind and amount of information as is used in the frequency-domain reconstruction algorithm.

4.4.2 Methodology

Suppose that the true network generating our given data is given by

$$Y(z) = W(z)Y(z) + V(z)U(z). \tag{4.55}$$

Since we have assumed zero initial conditions, the inverse $\mathcal{Z}$-transform of this network yields

$$y(t) = W(t) * y(t) + V(t) * u(t), \tag{4.56}$$

where $*$ denotes the convolution operator. Consider the impulse response $W_{ij}(t)$ (or equivalently $V_{ij}(t)$). By assumption, $W_{ij}(z) \in RP$, which implies that $W_{ij}(t) = 0$ for $t < 0$ (if we assume that $W_{ij}(z)$ is also strictly proper, then we could also say that $W_{ij} = 0$, as was done in [12, 20]; however, here, we consider the more general case). Also by assumption, we have assumed that there exists a finite r such that $W_{ij}(t) \approx 0$ for $t > m(r+1)+1$. Thus, we can approximate every $W_{ij}(t)$ and every $V_{ij}(t)$ with a finite impulse response (FIR) with support in the range $t = 0, \ldots, r$.

Given these assumptions, observe that

$$
\begin{aligned}
y(0) &= \begin{bmatrix} W(0) & V(0) \end{bmatrix} \begin{bmatrix} y(0) \\ u(0) \end{bmatrix} \\
y(1) &= \begin{bmatrix} W(1) & W(0) & V(1) & V(0) \end{bmatrix} \begin{bmatrix} y(0) \\ y(1) \\ u(0) \\ u(1) \end{bmatrix} \\
&\vdots \\
y(r) &= \begin{bmatrix} W(r) & \cdots & W(0) & V(r) & \cdots & V(0) \end{bmatrix} \begin{bmatrix} y(0) \\ \vdots \\ y(r) \\ u(0) \\ \vdots \\ u(r) \end{bmatrix} \\
y(r+1) &= \begin{bmatrix} W(r) & \cdots & W(0) & V(r) & \cdots & V(0) \end{bmatrix} \begin{bmatrix} y(1) \\ \vdots \\ y(r+1) \\ u(1) \\ \vdots \\ u(r+1) \end{bmatrix} \\
&\vdots \\
y(T) &= \begin{bmatrix} W(r) & \cdots & W(0) & V(r) & \cdots & V(0) \end{bmatrix} \begin{bmatrix} y(T-r) \\ \vdots \\ y(T) \\ u(T-r) \\ \vdots \\ u(T) \end{bmatrix}
\end{aligned}
$$

Thus, we can rewrite (4.56) as the following matrix multiplication (which we call the Toeplitz representation of the network):

$$\vec{y}(T) = \bar{W}(T)\vec{y}(T) + \bar{V}(T)\vec{u}(T), \tag{4.57}$$

where $\bar{W}(T) \in \mathbb{R}^{p^2(T+1)\times p^2(T+1)}$, where $\bar{V}(T) \in \mathbb{R}^{p^2(T+1)\times pm(T+1)}$, and where

$$\vec{y}(T) = \begin{bmatrix} y(0)' & y(1)' & \cdots & y(T)' \end{bmatrix}', \tag{4.58a}$$

$$\vec{u}(T) = \begin{bmatrix} u(0)' & u(1)' & \cdots & u(T)' \end{bmatrix}', \tag{4.58b}$$

$$\bar{W}(T) = \begin{bmatrix} W(0) & 0 & \cdots & 0 & \cdots & 0 \\ W(1) & W(0) & \cdots & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ W(r) & W(r-1) & \cdots & W(0) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & W(r) & \cdots & W(0) \end{bmatrix}, \tag{4.58c}$$

$$\bar{V}(T) = \begin{bmatrix} V(0) & 0 & \cdots & 0 & \cdots & 0 \\ V(1) & V(0) & \cdots & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ V(r) & V(r-1) & \cdots & V(0) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & V(r) & \cdots & V(0) \end{bmatrix}. \tag{4.58d}$$

Rewrite (4.57) as

$$\vec{y}(T) = \begin{bmatrix} \bar{W}(T) & \bar{V}(T) \end{bmatrix} \begin{bmatrix} \vec{y}(T) \\ \vec{u}(T) \end{bmatrix}. \tag{4.59}$$

Take the transpose of both sides to get

$$\vec{y}(T)' = \begin{bmatrix} \vec{y}(T)' & \vec{u}(T)' \end{bmatrix} \begin{bmatrix} \bar{W}(T)' \\ \bar{V}(T)' \end{bmatrix}. \tag{4.60}$$

Expanding, we get

$$\begin{bmatrix} y(0)' & \cdots & y(r)' & \cdots & y(T)' \end{bmatrix} =$$

$$\begin{bmatrix} y(0)' & \cdots & y(r)' & \cdots & y(T)' & u(0)' & \cdots & u(r)' & \cdots & u(T)' \end{bmatrix} \begin{bmatrix} W(0)' & \cdots & W(r)' & \cdots & 0 \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & \cdots & W(0)' & \cdots & W(r)' \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & W(0)' \\ V(0)' & \cdots & V(r)' & \cdots & 0 \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & \cdots & V(0)' & \cdots & V(r)' \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & V(0)' \end{bmatrix}$$

Consider the first column of the above equation, defining

$$y(0)' = y(0)'W(0)' + u(0)'V(0)'. \tag{4.61}$$

Suppose that $\vec{W}(t)$ is the vectorized form of $W(t)$ stacked in row-major order (or $W(t)'$ stacked in column-major order), and that $\vec{V}(t)$ is likewise the vectorized form of $V(t)$ stacked

in row-major order. Define $\bar{Y}(t) \in \mathbb{R}^{p \times p^2}$ to be

$$\bar{Y}(t) = \begin{bmatrix} y(t)' & 0 & \cdots & 0 \\ 0 & y(t)' & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & y(t)' \end{bmatrix}. \tag{4.62}$$

Likewise define $\bar{U}(t) \in \mathbb{R}^{p \times pm}$ to be

$$\bar{U}(t) = \begin{bmatrix} u(t)' & 0 & \cdots & 0 \\ 0 & u(t)' & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & u(t)' \end{bmatrix}. \tag{4.63}$$

Then, (4.61) can be rewritten as

$$\begin{aligned} y(0) &= \bar{Y}(0)\vec{W}(0) + \bar{U}(0)\vec{V}(0) \\ &= \begin{bmatrix} \bar{Y}(0) & \bar{U}(0) \end{bmatrix} \begin{bmatrix} \vec{W}(0) \\ \vec{V}(0) \end{bmatrix}. \end{aligned} \tag{4.64}$$

REMARK 4.4.4: Since the methodology for the time-domain reconstruction algorithm is very similar to the methodology for the frequency-domain reconstruction algorithm, we will reuse much of the same notation for both convenience and to draw parallels between the two algorithms. However, unless otherwise noted, definitions for some variable in this section are not equivalent to the definitions of a variable of the same name in Section 4.3.

By utilizing these definitions, we can also rewrite the full (4.60) as

$$\vec{y}(T) = L\theta, \tag{4.65}$$

where $L \in \mathbb{R}^{p(T+1)\times(p^2+pm)(r+1)}$ is

$$L = \begin{bmatrix} \bar{Y}(0) & \bar{U}(0) & 0 & 0 & \cdots & 0 & 0 \\ \bar{Y}(1) & \bar{U}(1) & \bar{Y}(0) & \bar{U}(0) & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \bar{Y}(r) & \bar{U}(r) & \bar{Y}(r-1) & \bar{U}(r-1) & \cdots & \bar{Y}(0) & \bar{U}(0) \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \bar{Y}(T) & \bar{U}(T) & \bar{Y}(T-1) & \bar{U}(T-1) & \cdots & \bar{Y}(T-r) & \bar{U}(T-r) \end{bmatrix}, \tag{4.66}$$

and contains only known entries from the inputs and outputs, and where $\theta \in \mathbb{R}^{r(p^2+pm)}$ is

$$\theta' = \begin{bmatrix} \vec{W}(0)' & \vec{V}(0)' & \vec{W}(1)' & \vec{V}(1)' & \cdots & \vec{W}(r)' & \vec{V}(r)' \end{bmatrix} \tag{4.67}$$

and contains the unknown entries in the impulse responses for W and V.

Notice the similarity of (4.65) to (4.12) in Section 4.3. As we did before, we wish to solve for $\hat{\theta}$ uniquely; thus, we require L to be full column rank. Moreover, as before, it is necessary to reduce the dimension of the domain of L. To do so, we utilize the same identifiability conditions encoded in K that we used before.

Let $M \in \mathbb{R}^{p(T+1)\times k(r+1)}$ be

$$M = \begin{bmatrix} \begin{bmatrix} \bar{Y}(0) & \bar{U}(0) \end{bmatrix} K & 0 & \cdots & 0 \\ \begin{bmatrix} \bar{Y}(1) & \bar{U}(1) \end{bmatrix} K & \begin{bmatrix} \bar{Y}(0) & \bar{U}(0) \end{bmatrix} K & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ \begin{bmatrix} \bar{Y}(r) & \bar{U}(r) \end{bmatrix} K & \begin{bmatrix} \bar{Y}(r-1) & \bar{U}(r-1) \end{bmatrix} K & \cdots & \begin{bmatrix} \bar{Y}(0) & \bar{U}(0) \end{bmatrix} K \\ \vdots & \vdots & \ddots & \vdots \\ \begin{bmatrix} \bar{Y}(T) & \bar{U}(T) \end{bmatrix} K & \begin{bmatrix} \bar{Y}(T-1) & \bar{U}(T-1) \end{bmatrix} K & \cdots & \begin{bmatrix} \bar{Y}(T-r) & \bar{U}(T-r) \end{bmatrix} K \end{bmatrix}, \tag{4.68}$$

and let

$$\hat{\theta}' = \left[(K^+)' \begin{bmatrix} \vec{W}(0)' & \vec{V}(0)' \end{bmatrix} \quad (K^+)' \begin{bmatrix} \vec{W}(1)' & \vec{V}(1)' \end{bmatrix} \quad \cdots \quad (K^+)' \begin{bmatrix} \vec{W}(r)' & \vec{V}(r)' \end{bmatrix} \right]. \quad (4.69)$$

As before, our problem thus becomes reduced to finding the unique $\hat{\theta}$ such that

$$\vec{y}(T) = M\hat{\theta}. \quad (4.70)$$

Note that these identifiability conditions reduce each $W_{ij}(t)$ and $V_{ij}(t)$ (for $t = 0, \ldots r$) in the same way that W and V were reduced in Section 4.3. For instance, if we know that $W_{11}(z) = 0$, then encoding that information using K will ensure that $W_{11}(t) = 0$ for all $t = 0, \ldots r$. Likewise, if we know that $P_{12}(z) = P_{11}(z) + Q_{12}(z)$, then K will encode $P_{12}(t) = P_{11}(t) + Q_{12}(t)$ for all $t = 0, \ldots, r$. Note also that we can recover θ encoding all parameters in (W, V) uniquely from $\hat{\theta}$ with

$$\theta = \left[K \oplus \overset{r}{\cdots} \oplus K \right] \hat{\theta}. \quad (4.71)$$

The definition of M in (4.68) highlights a necessary condition that we must require of our data in order to reconstruct our network. We summarize this condition in the following result:

Lemma 4.4.5: *Let $L \in \mathbb{R}^{p(T+1)\times(p^2+pm)(r+1)}$ be defined as in (4.66), let $K \in \mathbb{R}^{p^2+pm\times k}$ encode the identifiability conditions of this network as defined in Section 4.3, and let $M \in \mathbb{R}^{p(T+1)\times k(r+1)}$ be computed from L and K as defined in (4.68). Then M is injective only if $T \geq \frac{k}{p}(r+1) - 1$ (or, equivalently when $k = pm$, $T \geq m(r+1) - 1$).* ♢

Proof Assume, to the contrary, that $T < \frac{k}{p}(r+1) - 1$. Then there are less than $p\left(\frac{k}{p}(r+1) - 1 + 1\right) = k(r+1)$ rows in M. Since there are exactly $k(r+1)$ columns in M, M cannot be injective since $\operatorname{rank} M \leq \min\left(p(T+1), k(r+1)\right) < k(r+1)$. ∎

We can now state the necessary and sufficient conditions on K using the transformation[7] in (4.68), which we summarize in the following result (which parallels Theorem 4.3.9):

Theorem 4.4.6: *Let $u(t)$ and $y(t)$ for $t = 0, \ldots, T$ be the discrete-time and time-domain input-output data generated from some discrete-time and proper DNF $(W(t), V(t))$ (with $W(t)$ having dimensions $p \times p$ and $V(t)$ having dimensions $p \times m$). Let $L \in \mathbb{R}^{p(T+1)\times(p^2+pm)(r+1)}$ be defined as in (4.66), let $K \in \mathbb{R}^{p^2+pm\times k}$ encode the identifiability conditions of this network as defined in Section 4.3, and let $M \in \mathbb{R}^{p(T+1)\times k(r+1)}$ be computed from L and K as defined in (4.68). Then, there is a bijection between $(u(t), y(t))$ and $(W(t), V(t))$ (i.e., $(W(t), V(t))$ can be reconstructed uniquely from our input-output data) if and only if the following conditions hold:*

1. *M is injective (i.e., $\operatorname{rank} M = k(r+1)$)*
2. *$\vec{y}(T) \in \mathcal{R}(M)$, where $\mathcal{R}(M)$ is the range of M* ♢

Proof Observe that M is the mapping from unidentified model parameters $\hat{\theta}$ (where all model parameters in θ can be recovered uniquely from $\hat{\theta}$ according to (4.71) to the known parameters $\vec{y}(T)$. The two conditions listed above are well-known to ensure that $\hat{\theta}$ is computed uniquely. ■

In summary, the time-domain network reconstruction is given in Algorithm 2.

4.4.3 Data Informativity

In the frequency-domain algorithm, we skirted the issue of data informativity by assuming that the input-output data was rich enough that system identification could recover the correct transfer function G (e.g., the choice of inputs excited all of the modes of the transfer

[7] Other and more general transformations reducing the dimension of L (such as post multiplying L by some arbitrary matrix in $\mathbb{R}^{(p^2+pm)(r+1)\times k(r+1)}$) will also work; however, they are not meaningful for this problem since we assume the structure to be time-invariant. Any reduction other than that in (4.68) would signify that our knowledge of the structure of (W, V) is different at every time in the impulse responses.

Algorithm 2 Time Domain Network Reconstruction

procedure ReconstructTime(D_u, D_y, K, r) ▷ Find $(W(t), V(t))$ from $(u(t), y(t))$ given identifiability K assuming that $W_{ij}(t) \approx 0$ and $V_{ij}(t) \approx 0$ for all $t > r$

  $T, p \leftarrow$ Shape(D_u) ▷ The t'th row of D_u is $u(t)$ transpose

  $\hat{T}, m \leftarrow$ Shape(D_y) ▷ The t'th row of D_y is $y(t)$ transpose

  assert($T = \hat{T}$) ▷ Inputs and outputs should be defined over same time

  $h, k \leftarrow$ Shape(K)

  assert($h = p^2 + pm$)

  assert($k \leq pm$ and $\operatorname{rank} K = k$)

  $\vec{y}(T) \leftarrow \begin{bmatrix}[D_y]_{0,:} & [D_y]_{1,:} & \cdots & [D_y]_{T,:}\end{bmatrix}'$ ▷ $[D_y]_{t,:}$ is the t't row of D_y

  for $t \in \{0, \ldots, T\}$ **do** ▷ Order irrelevant

    $\hat{M}(t) \leftarrow$ BLUYK(t, D_u, D_y, K)

  end for

  $$M \leftarrow \begin{bmatrix} \hat{M}(0) & 0 & \cdots & 0 \\ \hat{M}(1) & \hat{M}(0) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ \hat{M}(r) & \hat{M}(r-1) & \cdots & \hat{M}(0) \\ \vdots & \vdots & \ddots & \vdots \\ \hat{M}(T) & \hat{M}(T-1) & \cdots & \hat{M}(T-r) \end{bmatrix} \in \mathbb{R}^{p(T+1) \times k(r+1)}$$

  $\hat{\theta} \leftarrow$ Pinv(M, $\vec{y}(t)$) $\in \mathbb{R}^{k(r+1)}$

  $\theta \leftarrow \left[K \oplus \overset{r}{\cdots} \oplus K\right] \hat{\theta} \in \mathbb{R}^{(p^2+pm)(r+1)}$

  for $t \in [0, \ldots, r]$ **do** ▷ Order necessary

    $i_x \leftarrow 0$

    for $i \in [1, \ldots, p]$ **do** ▷ Order necessary

      for $j \in [1, \ldots, p]$ **do** ▷ Order necessary

        $W_{ij}(t) \leftarrow \theta_{i_x}$

        $i_x \leftarrow l + 1$

      end for

      for $j \in [1, \ldots, m]$ **do** ▷ Order necessary

        $V_{ij}(t) \leftarrow \theta_{i_x}$

        $i_x \leftarrow l + 1$

      end for

    end for

  end for

  $$W(t) \leftarrow \begin{bmatrix} W_{11}(t) & \cdots & W_{1p}(t) \\ \vdots & \ddots & \vdots \\ W_{p1}(t) & \cdots & W_{pp}(t) \end{bmatrix} \in \mathbb{R}^{p \times p \times r+1}$$

  $$V(t) \leftarrow \begin{bmatrix} V_{11}(t) & \cdots & V_{1m}(t) \\ \vdots & \ddots & \vdots \\ V_{p1}(t) & \cdots & V_{pm}(t) \end{bmatrix} \in \mathbb{R}^{p \times m \times r+1}$$

  return $W(t), V(t)$ ▷ Return the reconstructed DNF

end procedure ▷ See Algorithm 3 for helper procedures

Algorithm 3 Helpers for Algorithm 2

procedure PINV(M, y) ▷ Get the unique x such best fits $y = Mx$ with the 2-norm
  $m, n \leftarrow$ SHAPE(()M)
  `assert`(rank $M = n$) ▷ We require M to be injective
  if $m = n$ **then** $x = M^{-1}y$
  else$x = (M'M)^{-1}M'y$
  end if
  return x
end procedure
procedure BLUYK(t, D_u, D_y, K) ▷ Build the block $[\ \bar{Y}(t)\quad \bar{U}(t)\ ]K$
  $u(t) \leftarrow [D_u]'_{t,:}$ ▷ t'th row of D_u, transposed
  $y(t) \leftarrow [D_y]'_{t,:}$ ▷ t'th row of D_y, transposed

$$\bar{Y}(t) \leftarrow \begin{bmatrix} y(t)' & 0 & \cdots & 0 \\ 0 & y(t)' & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & y(t)' \end{bmatrix} \in \mathbb{R}^{p \times p^2}$$

$$\bar{U}(t) \leftarrow \begin{bmatrix} u(t)' & 0 & \cdots & 0 \\ 0 & u(t)' & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & u(t)' \end{bmatrix} \in \mathbb{R}^{p \times pm}$$

  return $[\ \bar{Y}(t)\quad \bar{U}(t)\ ]K \in \mathbb{R}^{p \times k}$
end procedure
procedure SHAPE(M) ▷ $M \in \mathcal{F}^{m \times n}$ for any field $\mathcal{F}$
  return m, n ▷ Rows, Cols in M
end procedure

function). In the time-domain algorithm, however, we cannot avoid the issue in the same manner since we utilize the input-output data directly.

The key to understanding data informativity lies in Theorem 4.4.6, which states that the necessary and sufficient conditions for reconstruction is that M (4.68) is injective. In particular, consider the extreme case where $u(t) = 0$ for $t = 0, \ldots, T$. Then, since we have assumed zero initial conditions, we have that $y(t) = 0$ as well. Thus, $M = 0$ is not injective, and we have failed to meet the necessary and sufficient conditions for reconstruction.

Unfortunately, it is difficult to separate necessary and sufficient conditions on informativity from those listed in Theorem 4.4.6. However, the following result provides necessary conditions on data informativity for a common subclass of problems knowledge of V is independent of knowledge of W and every column of V has at least one unknown entry (target specificity meets these requirements).

Theorem 4.4.7 (Data Informativity): *Let* $\hat{D}_u = [u(0) \quad u(1) \quad \cdots \quad u(T-r)]' \in \mathbb{R}^{T-r+1\times m}$. *Suppose that* $K \in \mathbb{R}^{p^2+pm\times k}$ *is defined such that* $k \leq pm$ *and* $\operatorname{rank} K = k$. *Suppose that no entry in* V *is known to be a function of any entry in* W, *and suppose that there is at least one unknown entry in each of the* m *columns of* V. *Then* M *is injective only if* $\operatorname{rank} D_u = m$ *(i.e., if* D_u *is injective).* ♢

Proof Let $\hat{k} \leq k$ be the number of unknown entries in V and $\tilde{K} = k - \hat{k}$ be the number of unknown entries in W. Let

$$K = \begin{bmatrix} K_{WW} & K_{WV} \\ K_{VW} & K_{VV} \end{bmatrix}, \tag{4.72}$$

where $K_{WV} \in \mathbb{R}^{p^2\times\hat{k}}$ defines the entries in W that are known to be functions of other entries in V and $K_{VW} \in \mathbb{R}^{pm\times\tilde{k}}$ defines the entries in V that are known to be functions of other entries in W. By assumption, no entry in V is known to be a function of any entry in W, thus $K_{VW} = 0$. This implies that, with $K_{VV} \in \mathbb{R}^{pm\times\hat{k}}$ and since K is now block triangular, we have $\operatorname{rank} K_{VV} = \hat{k} \geq m$.

Consider

$$L_u = \begin{bmatrix} \bar{U}(0) & 0 & \cdots & 0 \\ \bar{U}(1) & \bar{U}(0) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ \bar{U}(r) & \bar{U}(r-1) & \cdots & \bar{U}(0) \\ \vdots & \vdots & \ddots & \vdots \\ \bar{U}(T) & \bar{U}(T-1) & \cdots & \bar{U}(T-r) \end{bmatrix} \in \mathbb{R}^{p(T+1)\times pm(r+1)}. \tag{4.73}$$

and

$$M_u = \begin{bmatrix} \bar{U}(0)K_{VV} & 0 & \cdots & 0 \\ \bar{U}(1)K_{VV} & \bar{U}(0)K_{VV} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ \bar{U}(r)K_{VV} & \bar{U}(r-1)K_{VV} & \cdots & \bar{U}(0)K_{VV} \\ \vdots & \vdots & \ddots & \vdots \\ \bar{U}(T)K_{VV} & \bar{U}(T-1)K_{VV} & \cdots & \bar{U}(T-r)K_{VV} \end{bmatrix} \in \mathbb{R}^{p(T+1)\times \hat{k}(r+1)}. \tag{4.74}$$

Notice that, since K is upper block triangular, the columns in M_u are also present in M. Thus, if M_u is not full column rank, M cannot be injective.

Let

$$\mathcal{C}_{i,j}(\bar{U}) = \begin{bmatrix} \bar{U}(i) \\ \bar{U}(i+1) \\ \vdots \\ \bar{U}(j) \end{bmatrix}, \tag{4.75}$$

which allows us to rewrite

$$L_u = \begin{bmatrix} \mathcal{C}_{0,T-r}(\bar{U}) & 0 & \cdots & 0 \\ \mathcal{C}_{T-r+1,2(T-r)}(\bar{U}) & \mathcal{C}_{0,T-r}(\bar{U}) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ \mathcal{C}_{r,T}(\bar{U}) & \mathcal{C}_{r-1,T-1}(\bar{U}) & \cdots & \mathcal{C}_{0,T-r}(\bar{U}) \end{bmatrix}, \tag{4.76}$$

and since L_u is lower block triangular with r copies of $\mathcal{C}_{0,T-R}$ on the diagonal, we have that $\operatorname{rank} L_u = r \operatorname{rank} \mathcal{C}_{0,T-R}$.

Observe that, by definition, and where $[D_u]_t$ is defined by the t'th row of D_u, we have

$$\mathcal{C}_{0,T-r}(\bar{U}) = \begin{bmatrix} [D_u]_0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & [D_u]_0 \\ [D_u]_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & [D_u]_1 \\ \vdots & \ddots & \vdots \\ [D_u]_{T-r} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & [D_u]_{T-r} \end{bmatrix}, \tag{4.77}$$

hence we can permute the rows of $\mathcal{C}_{0,T-r}$ to get $D_u \oplus \overset{p}{\cdots} \oplus D_u$. Thus, $\operatorname{rank} \mathcal{C}_{0,T-r} = p \operatorname{rank} D_u$ and $\operatorname{rank} L_u = r \operatorname{rank} \mathcal{C}_{0,T-R} = rp \operatorname{rank} D_u$.

Since K_{VV} encodes that we do not know any information about at least one entry in each column of V, there exists, for $i = 1, \ldots, m$, a row in K_{VV} with a one in column i and zeros everywhere else. Thus the product $\mathcal{C}_{0,T-r}(\bar{U})K_{VV}$ contains duplicates of the columns of $\mathcal{C}_{0,T-r}(\bar{U})$ corresponding to these rows in K_{VV}. And by definition of $\mathcal{C}_{0,T-r}(\bar{U})$, this implies that every column of D_u, padded with zeros, appears at least once in $\mathcal{C}_{0,T-r}(\bar{U})K_{VV}$. Thus, a

necessary condition for $\mathcal{C}_{0,T-r}(\bar{U})K_{VV}$ to be full column rank is for D_u to also be full column rank.

Observe also that

$$M_u = \begin{bmatrix} \mathcal{C}_{0,T-r}(\bar{U})K_{VV} & 0 & \cdots & 0 \\ \mathcal{C}_{T-r+1,2(T-r)}(\bar{U})K_{VV} & \mathcal{C}_{0,T-r}(\bar{U})K_{VV} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ \mathcal{C}_{r,T}(\bar{U})K_{VV} & \mathcal{C}_{r-1,T-1}(\bar{U})K_{VV} & \cdots & \mathcal{C}_{0,T-r}(\bar{U})K_{VV} \end{bmatrix}. \tag{4.78}$$

Since M_u is lower block triangular with $\mathcal{C}_{0,T-r}(\bar{U})K_{VV}$ repeated on the diagonals, M_u is injective only if $\mathcal{C}_{0,T-r}(\bar{U})K_{VV}$ is injective only if D_u is injective. ■

While there are many sets of inputs to satisfy these necessary data informativity requirements, the next provides a simple method of constructing such data.

Corollary 4.4.8: *Let $u_i(t) \sim \mathcal{N}(0,\sigma)$ for $i = 1,\dots T-r$, $t = 0,\dots T$, and some $\sigma > 0$, where $\mathcal{N}(\mu,\sigma)$ is the normal distribution with mean μ and standard deviation σ. Then $D_u = [u(0) \quad u(1) \quad \cdots \quad u(T-r)]' \in \mathbb{R}^{T-r+1\times m}$ is injective with probability one.* ♢

Proof Follows immediately from the fact that non-injective matrices are sparse in the set of matrices with more rows than columns. ■

Corollary 4.4.8 is especially important in the passive reconstruction (see Remark 4.4.1) setting where inputs are observed but not controlled. If we can assume that the uncontrolled inputs are white noise, then we meet the necessary data informativity conditions.

4.4.4 Advantages Compared to the Frequency-Domain Algorithm

We have already described the main disadvantage of the time-domain algorithm (namely that it solves a restricted subclass of a problem which has already been solved using the

frequency-domain algorithm); however, it still exhibits many advantages over the frequency-domain algorithm which make it ideal in many circumstances. Some of these advantages are as follows:

- **Computational Complexity**: The frequency-domain algorithm requires the inversion and simplification of a symbolic matrix. The complexity of such an inversion is difficult to analyze. The time-domain algorithm instead requires the inversion of a real-valued matrix. While this real-valued matrix is much larger than the equivalent symbolic matrix, bounds on the complexity are easily obtained[8]. We suspect that the time-domain algorithm will be much more scalable than the frequency-domain algorithm as m and p grow large, so long as r remains bounded.
- **Analysis of Uncertainty and Noise:** While we do not touch on noise and uncertainty in this work since the time-domain algorithm utilizes the projection theorem over real-valued matrices, well-established theory in linear regression can be leveraged to analyze and improve the performance of the algorithm under uncertainty and noise. For instance, the Fisher Information Matrix can be leveraged to identify how large T should be to reduce error below some desired threshold. Alternative linear regression techniques, such as lasso regression (see [12] for the use of lasso regression in time-domain network reconstruction) and total least squares can be used to improve performance given certain uncertainty models.
- **Data**: For many network reconstruction problems (such as the reconstruction of financial networks), the data received is discrete time and in the time domain. This algorithm is designed to work with such data directly, making analysis and use of that data convenient.

[8]Let $y = Mx$ where $M \in \mathbb{R}^{a\times b}$ $(a > b)$ and $y \in \mathbb{R}^a$ are known and $x \in \mathbb{R}^b$ is unknown. To find $x^* = \arg\min_x \|y - Mx\|_2$ (i.e., to solve the least squares problem) using a standard implementation, one must perform the following steps: (1) Compute $M'M \in \mathbb{R}^{a\times a}$, which takes $\mathcal{O}(ab^2)$ time. (2) Invert $M'M$, which takes $\mathcal{O}(b^3)$ time. (3) Compute $M'y$, which takes $\mathcal{O}(ab)$ time. And (4) compute $x^* = [(M'M)^{-1}][M'y]$, which takes $\mathcal{O}(b^3)$ time. Thus, the time complexity is $\mathcal{O}(ab^2 + ab + 2b^3) = \mathcal{O}(ab^2)$ since $a > b$. For our particular implementation, we have that $M \in \mathbb{R}^{p(T+1)\times pm(r+1)}$; hence the time complexity is given by $\mathcal{O}\big((pm(r+1))^2(p(T+1))\big) = \mathcal{O}\big(p^3m^3T^3\big)$ since $T \geq m(r+1)$.

- **Real-Time Reconstruction (Moving Horizon Estimation)**: Both algorithms as presented in this work require that the network is reconstructed from scratch whenever new data is received. However, in the time-domain algorithm, if we replace the least squares algorithm with recursive least squares, we can update the network model live as new data is streamed in. To date, no research has been done in this area.

- **Connections to Model-Predictive Control**: Model-predictive control leverages a model of some system and chooses inputs over some time horizon to optimize some objective. At every time-step, this process is repeated to refine the choice of inputs.

  If we have a network in the form (4.57) (such as one found through the time-domain network reconstruction algorithm), the choice of $\vec{u}(T)$ over some time horizon $0, \ldots, T$ can be formulated as a linear programming problem (if the objective is linear) or a quadratic programming problem (if the objective is quadratic), thus solving the model-predictive control problem with a convex algorithm.

  Since we have a structured model, linear constraints can potentially be added to the linear (quadratic) programming problem leveraging the structure of the network to meet additional robustness requirements. Furthermore, the moving horizon estimation methodology discussed above can be used to update the network model in real time. The combination of moving-horizon estimation with model-predictive control is known as adaptive control, and these combined techniques may provide a linear solution to the adaptive control problem which may be easier to analyze than existing techniques.

  No research has yet been done in this area. It should also be noted that the time-domain reconstruction algorithm must be generalized to allow for non-zero initial conditions to be useful for these problems.

- **Blind Reconstruction:** In [12, 18, 89], the blind reconstruction problem was introduced as a network reconstruction problem where the inputs into the system are not observed but are instead drawn from some known distribution. Since inputs are

not observed, we cannot build a transfer function from input to output; thus, the frequency-domain algorithm cannot be used to solve this problem. However, the time domain algorithm can be modified to leverage knowledge about the input distributions as a way of solving this problem. To date, we still consider the blind reconstruction problem to be an open problem requiring additional research.

4.4.5 Numeric Examples

We now demonstrate the time-domain reconstruction algorithm with several examples. The matrices and equations required to adequately describe each step of these examples are significantly larger than those in Section 4.3.4 and do not fit in print format. As such, we rely on figures to demonstrate intermediate steps and successful reconstruction.

Example 4.4.9: A Strictly Proper Network

We reproduce the example that was first presented in [20] to demonstrate that this algorithm–which was generalized to reconstruct proper networks–still functions when attempting to reconstruct a strictly proper network.

Suppose that we begin with the (unknown) state space model

$$\begin{aligned} x[k+1] &= Ax[k] + Bu[k] \\ y[k] &= Cx[k], \end{aligned} \tag{4.79}$$

where

$$A = \begin{bmatrix} \frac{3}{4} & 0 & 0 & 0 & 0 & \frac{6}{5} \\ -\frac{1}{10} & -\frac{7}{20} & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{17}{20} & -1 & 0 & 0 \\ 0 & -\frac{73}{100} & 0 & \frac{19}{20} & 0 & 0 \\ 0 & 0 & \frac{43}{100} & 0 & -\frac{3}{5} & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{5} & \frac{11}{20} \end{bmatrix}, \quad B = \begin{bmatrix} \frac{7}{5} & 0 & -\frac{7}{5} \\ 0 & -\frac{1}{4} & 0 \\ 0 & 0 & \frac{3}{4} \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix},$$
$$C = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}. \tag{4.80}$$

Following the procedure in Section 2.4.1 to convert a state space model to a DSF, we get the (unknown) DSF representation (Q, P) of our model, where

$$Q = \begin{bmatrix} 0 & 0 & \frac{1032}{25(4z-3)(5z+3)(20z-11)} \\ -\frac{2}{20z+7} & 0 & 0 \\ 0 & \frac{292}{(20z-19)(20z-17)} & 0 \end{bmatrix}, \quad \text{and}$$
$$P = \begin{bmatrix} \frac{28}{5(4z-3)} & 0 & -\frac{28}{5(4z-3)} \\ 0 & -\frac{5}{20z+7} & 0 \\ 0 & 0 & \frac{15}{20z-17} \end{bmatrix}. \tag{4.81}$$

Following the procedure found in [12, 20], we can find the impulse responses for every entry in Q and P. These impulse responses are plotted as the blue dashed line in Figures 4.2 and 4.3.

Assume that we know that $P_{13} = P_{11}$. We encode this knowledge with

$$K = \begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{bmatrix}
\begin{matrix}
(Q_{11} = 0) \\
(Q_{12} = ?) \\
(Q_{13} = ?) \\
(Q_{21} = ?) \\
(Q_{22} = 0) \\
(Q_{23} = ?) \\
(Q_{31} = ?) \\
(Q_{32} = ?) \\
(Q_{33} = 0) \\
(P_{11} = ?) \\
(P_{12} = 0) \\
(P_{13} = -P_{11}) \\
(P_{21} = 0) \\
(P_{22} = ?) \\
(P_{23} = 0) \\
(P_{31} = 0) \\
(P_{32} = 0) \\
(P_{33} = ?)
\end{matrix} \tag{4.82}$$

Choose $r = 200$. Also let $T = 3(r + 1) = 603$ to satisfy Lemma 4.4.5. Note that we 'cheated' in this choice of r since we know, from Figures 4.2 and 4.3 that every impulse response converges to approximately zero by $t = 200$. However, as demonstrated in [12], the quality of results converge to optimal as r increases, so long as T is sufficiently larger than r ($T \geq 3(r + 1) - 1$ should be sufficiently large). Thus, one procedure to find r could be to start small and increase until the reconstruction results converge to a single answer.

Since there is exactly one unknown entry in each column of P, and since $K \in \mathbb{R}^{p^2+pm \times k}$ with $k = pm = 9$ and with $\operatorname{rank} K = k$, we satisfy the assumptions in Theorem 4.4.7. Thus choosing $u(t)$ such that D_u is injective is a necessary condition for reconstruction. As such and as suggested by Corollary 4.4.8, we let $u(t)$ be white noise with $\sigma = 1$. We then use (4.80) to generate our output data $y(t)$, giving the input-output data shown in Figure 4.1.

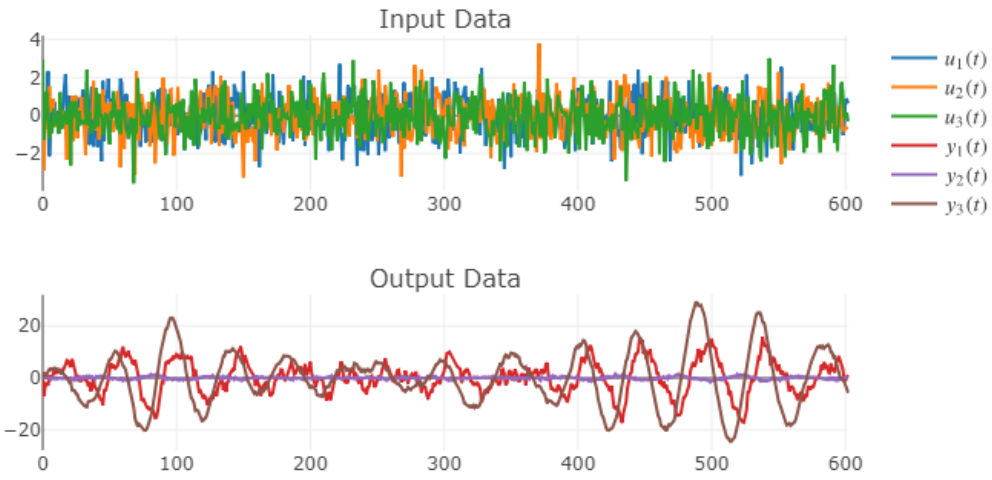


Figure 4.1 The input-output data generated from the system (4.80).

With K, $u(t)$, and $y(t)$, we can create M defined in (4.68). A quick check verifies that M is full column rank, thus meeting the necessary and sufficient conditions for reconstruction in Theorem 4.4.6. With this injective M, we can solve for $\hat{\theta}$ and θ uniquely. Then, extracting our impulse responses from θ, we get the reconstructed impulse responses that are shown with orange dots in Figures 4.2 and 4.3. Notice that the reconstructed impulse responses fit the actual impulse responses exactly; thus, we have successfully reconstructed our network.

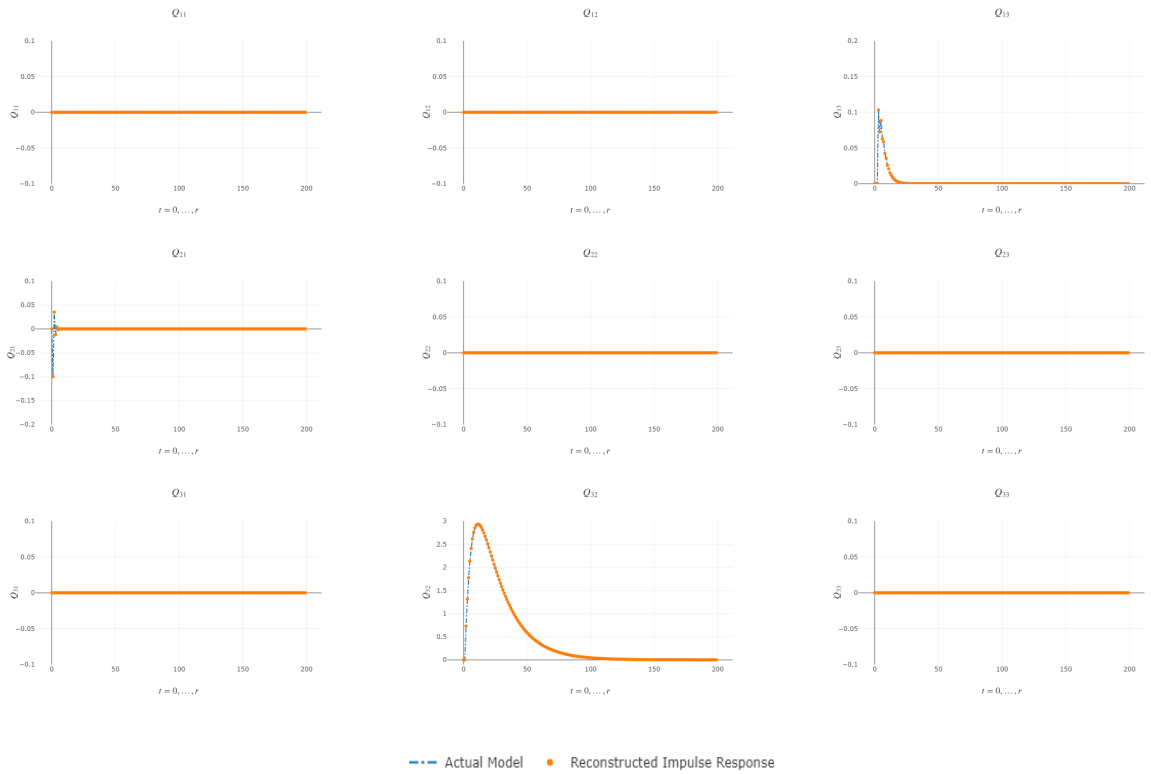


Figure 4.2 The reconstructed (orange dots) and actual (blue dashed line) impulse responses of the network given by (4.81).

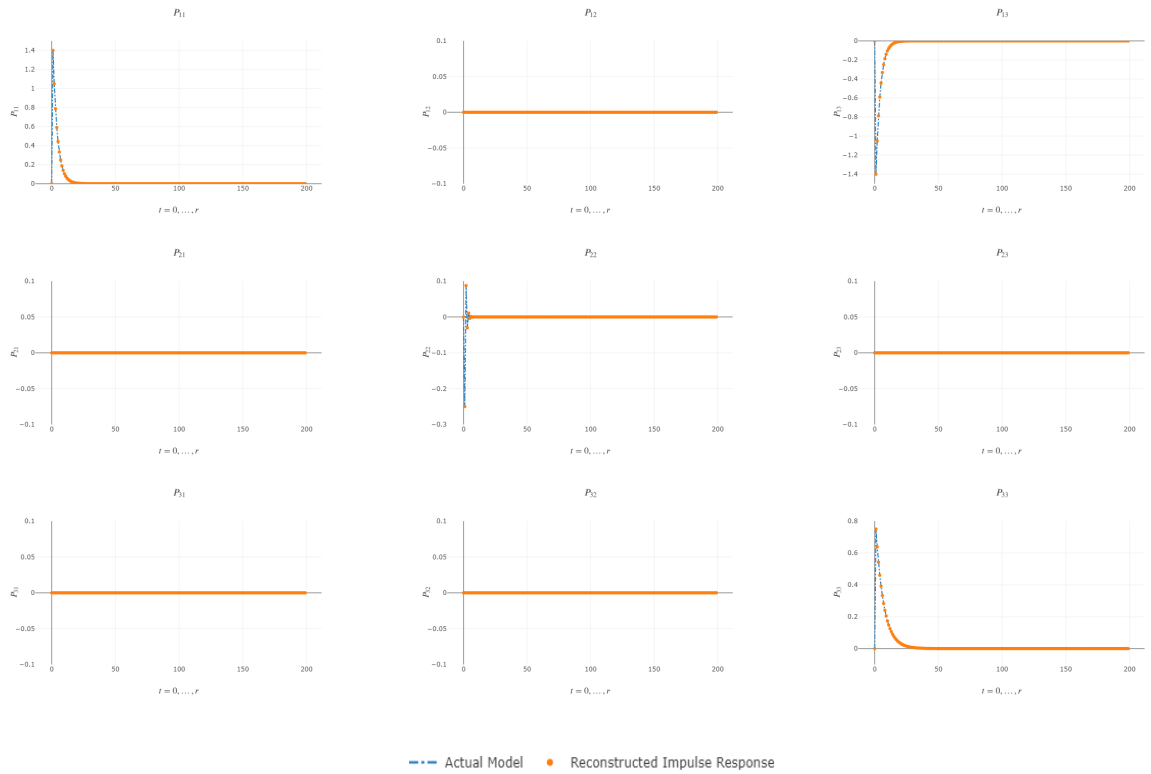


Figure 4.3 The reconstructed (orange dots) and actual (blue dashed line) impulse responses of the network given by (4.81).

Example 4.4.10: A Proper Network

We now demonstrate that the time-domain network reconstruction algorithm is capable of reconstructing proper (and not necessarily strictly proper) networks. To do so, set the system be defined by the (unknown) generalized state space model

$$\begin{aligned} x[k+1] &= Ax[k] + \hat{A}y[k] + Bu[k], \\ y[k] &= \bar{A}x[k] + \tilde{A}w[k] + \bar{B}u[k], \end{aligned} \tag{4.83}$$

where

$$A = \begin{bmatrix} \frac{3}{4} & 0 & 0 & 0 & 0 & \frac{6}{5} \\ -\frac{1}{10} & -\frac{7}{20} & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{17}{20} & -1 & 0 & 0 \\ 0 & -\frac{73}{100} & 0 & \frac{19}{20} & 0 & 0 \\ 0 & 0 & \frac{43}{100} & 0 & -\frac{3}{5} & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{5} & \frac{11}{20} \end{bmatrix} \quad \hat{A} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad B = \begin{bmatrix} \frac{7}{5} & 0 & 0 \\ 0 & -\frac{1}{4} & 0 \\ 0 & 0 & \frac{3}{4} \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix},$$

$$\bar{A} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \quad \tilde{A} = \begin{bmatrix} 4 & 1 & 1 \\ 0 & 3 & 0 \\ 0 & 1 & 2 \end{bmatrix} \quad \bar{B} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}. \tag{4.84}$$

Following the procedure in Section 2.4.2 to convert a generalized state space model to a DNF and then the procedure in Section 3.4 to find the hollow abstraction (DSF) of this DNF, we get the (unknown) DSF representation (Q, P) of our model, where

$$Q = \begin{bmatrix} 0 & -\frac{10000z^3-7000z^2-3675z+1443}{75(4z-3)(5z+3)(20z-11)} & -\frac{10000z^3-7000z^2-3675z+1443}{75(4z-3)(5z+3)(20z-11)} \\ -\frac{3}{4(5z+2)} & 0 & -\frac{1}{4(5z+2)} \\ 0 & -\frac{400z^2-720z-261}{(20z-19)(20z-17)} & 0 \end{bmatrix}, \quad \text{and}$$

$$P = \begin{bmatrix} -\frac{28}{15(4z-3)} & 0 & 0 \\ 0 & \frac{5}{8(5z+2)} & 0 \\ 0 & 0 & -\frac{15}{20z-17} \end{bmatrix}. \tag{4.85}$$

Observe that

$$(I - \tilde{A}) = \begin{bmatrix} -3 & -1 & -1 \\ 0 & -2 & 0 \\ 0 & -1 & -1 \end{bmatrix}. \tag{4.86}$$

and

$$(I - Q(\infty)) = (I - \operatorname{diag} \tilde{A})^{-1}(\tilde{A} - \operatorname{diag} \tilde{A}) = \begin{bmatrix} 1 & \frac{1}{3} & \frac{1}{3} \\ 0 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix} \tag{4.87}$$

are both non-singular. Thus, by Theorem 2.5.8, our network is well-posed. Note that we can prove that our network is well-posed in two ways. First, by directly checking $(I - Q(\infty))$ and using Theorem 2.5.8. Second, by checking that $(I - \tilde{A})$ is non-singular implying that DNF is well-posed. We then check that our hollow abstraction is representable (i.e., proper, which fails only when there is a 1 on the diagonal of $\tilde{A}$) which lets us leverage Theorem 3.4.7 to say that the DSF is also well-posed.

Following the procedure found in [12, 20], we can find the impulse responses for every entry in Q and P. These impulse responses are plotted as the blue dashed line in Figures 4.5 and 4.6.

Assume that we know that P is diagonal (target specificity). We encode this knowledge with

$$K = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{matrix} (Q_{11} = 0) \\ (Q_{12} = ?) \\ (Q_{13} = ?) \\ (Q_{21} = ?) \\ (Q_{22} = 0) \\ (Q_{23} = ?) \\ (Q_{31} = ?) \\ (Q_{32} = ?) \\ (Q_{33} = 0) \\ (P_{11} = ?) \\ (P_{12} = 0) \\ (P_{13} = 0) \\ (P_{21} = 0) \\ (P_{22} = ?) \\ (P_{23} = 0) \\ (P_{31} = 0) \\ (P_{32} = 0) \\ (P_{33} = ?) \end{matrix} \tag{4.88}$$

Choose $r = 200$ for the same reasons as the previous example and let $T = 2400 \geq 3(r+1) - 1$ to satisfy Lemma 4.4.5.

Since there is exactly one unknown entry in each column of P (as with the previous example), and since $K \in \mathbb{R}^{p^2+pm \times k}$ with $k = pm = 9$ and with $\operatorname{rank} K = k$, we satisfy the assumptions in Theorem 4.4.7. Thus choosing $u(t)$ such that D_u is injective is a

necessary condition for reconstruction. As such and as suggested by Corollary 4.4.8, we let $u(t)$ be white noise with $\sigma = 1$. We then use (4.84) to generate our output data $y(t)$, giving the input-output data shown in Figure 4.4.

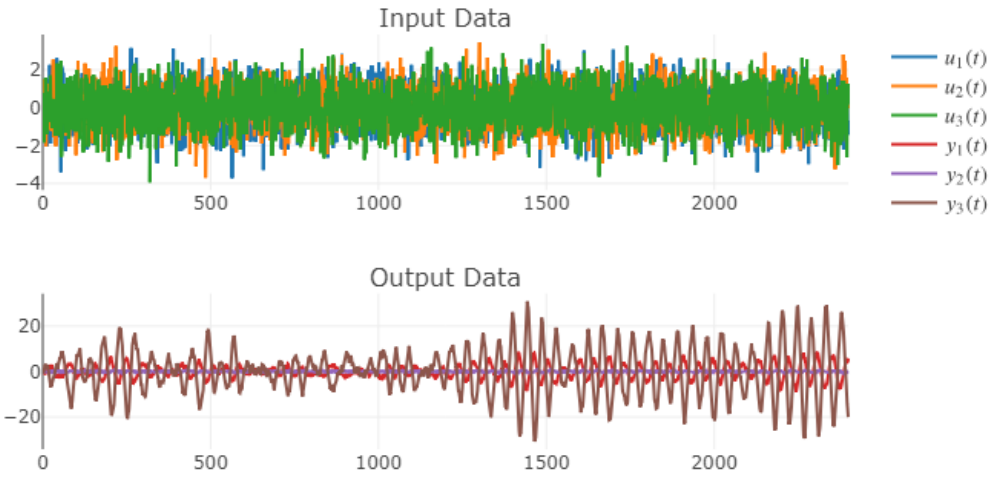


Figure 4.4 The input-output data generated from the system (4.84).

With K, $u(t)$, and $y(t)$, we can create M defined in (4.68). A quick check verifies that M is full column rank, thus meeting the necessary and sufficient conditions for reconstruction in Theorem 4.4.6. With this injective M, we can solve for $\hat{\theta}$ and θ uniquely. Then, extracting our impulse responses from θ, we get the reconstructed impulse responses that are shown with orange dots in Figures 4.5 and 4.6. Notice that the reconstructed impulse responses fit the actual impulse responses exactly; thus, we have successfully reconstructed our network.

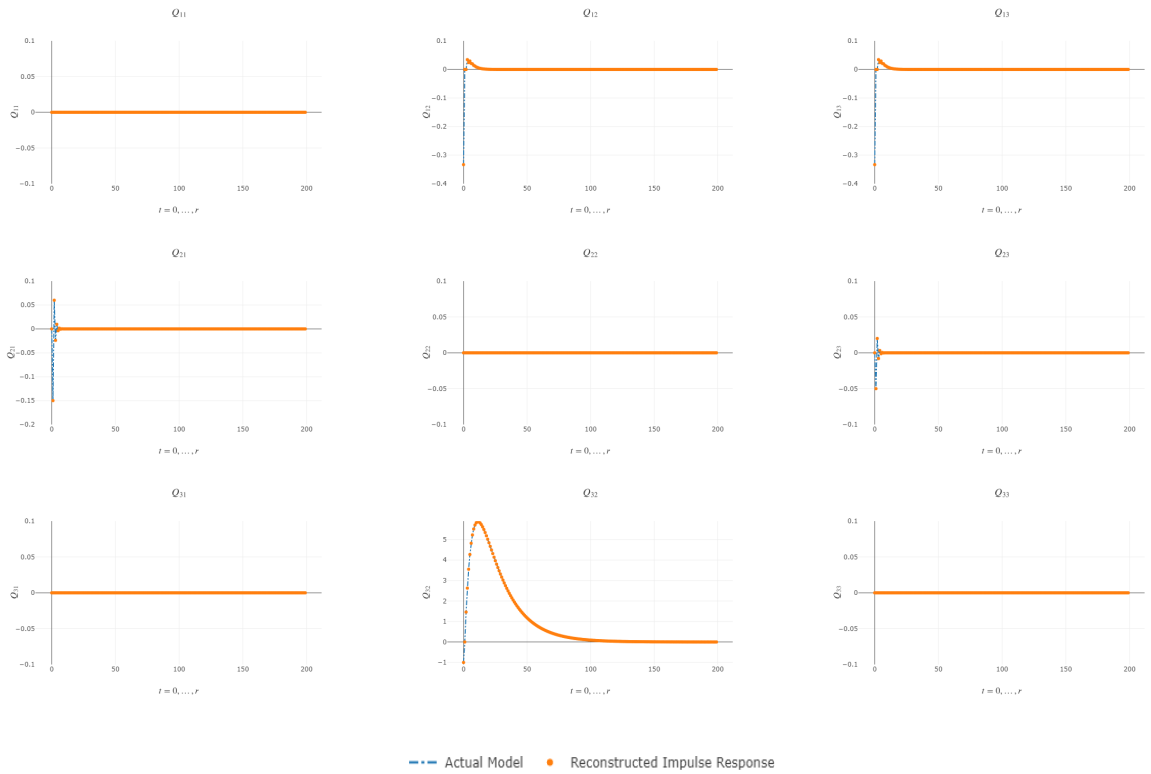


Figure 4.5 The reconstructed (orange dots) and actual (blue dashed line) impulse responses of the network given by (4.85).

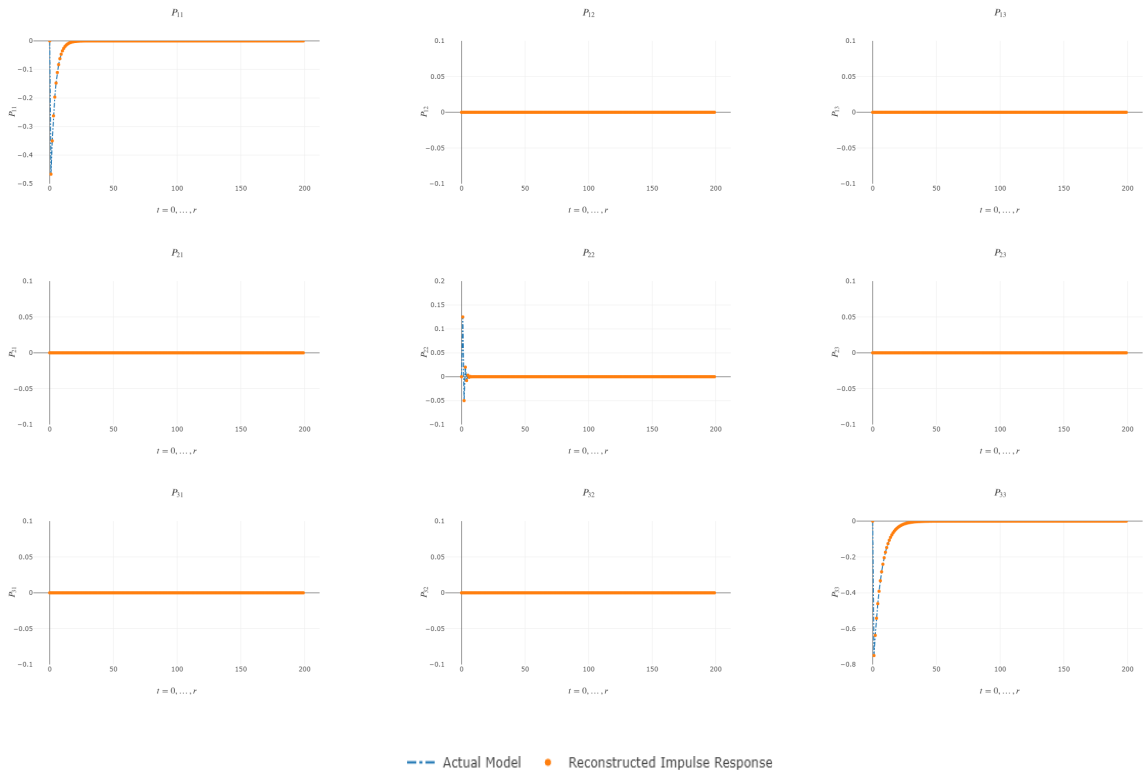


Figure 4.6 The reconstructed (orange dots) and actual (blue dashed line) impulse responses of the network given by (4.85).

4.5 Future Work

The following research directions and questions are still open for future research in the time-domain network reconstruction algorithm[9]:

- **Initial Conditions**: Presently, the time-domain algorithm assumes that initial conditions are zero. Future work should relax this constraint. The system identification literature has long studied non-zero initial conditions and will have insights into solving this problem (see [90]).
- **Rank Conditions on** L: A result analogous to Lemma 4.3.2 would strengthen the necessary conditions for reconstruction using the time-domain algorithm. It would be

[9] A variation of list was first published in [12]. In this work, we addressed several of the open questions from that original list, including proofs of identifiability conditions (though this direction can still be strengthened), exploration of what happens when our identifiability conditions make the wrong assumption (we take an abstraction as shown in Examples 4.3.15 and 4.3.16), and the reconstruction of proper (and not necessary strictly proper) networks.

reasonable that such a result would say that the rank of matrix L in (4.66) should equal the rank of the portion of L corresponding to the inputs (just as it did in Lemma 4.3.2). In other words, supposing that T satisfies Lemma 4.4.5, we would assume that $\operatorname{rank} L = pm(r+1)$. Unfortunately, numerical simulations indicate that this is, in fact, untrue. They suggest that, in reality, $\operatorname{rank} L = pm(r+3)$.

- **Reconstruction Validation**: Presently, little work has been done in the study of how to validate if a reconstructed network is correct when the true network is unknown. If we can assume that the original a priori assumptions about the structure are correct, then the reconstructed network will be correct if it is predictive of future data (since it is unique given the black box transfer function). However, validation of the a priori assumptions about the structure is undoubtedly impossible unless we can somehow modify the system. For instance, by injecting measured white noise that perturbs every output independently, we augment P with an identity matrix. The augmented P has $p^2 - p$ entries that are known to be zero, and we meet the necessary identifiability index for reconstruction (and likely meet the necessary and sufficient conditions as well). We can then reconstruct the network and validate that the reconstructed network matches our a priori information.

- **Other Regression Techniques**: In this work, we focus on least squares as the solution to our linear regression problems; however, other regression techniques could easily be used as well with various benefits, such as using the one-norm instead of the two-norm (which utilizes a linear program in its solution). Such a solution would be robust to outliers in the data. Several of the next items show use cases for other such alternative techniques.

- **Robust Reconstruction**: In this work, we have assumed that all inputs and outputs in the system are measured perfectly. The robust reconstruction problem, discussed in [12, 41, 56], assumes that our measurements (and possibly the internal processes as well) are perturbed by noise, and seeks to recover the structure and dynamics of

the network still. Of particular interest is the detection of when edges in the network are zero since reconstructing even a slightly non-zero edge will result in the incorrect structure of the network. In [12], it was shown that, by replacing the standard least squares algorithm with a lasso regression, the time-domain reconstruction algorithm will perform better with larger magnitudes of noise on the inputs and outputs, though it still only reconstructs an approximation of the actual network. Since such noise affects not only $\vec{y}(T)$, but also M, a total least squares methodology may also see success. Furthermore, by using the Fisher Information matrix, one can specify how large T should be to reduce error below a desired threshold. Furthermore, this knowledge could be used to inform thresholds for determining when a non-zero edge should be zero.

- **Bounds on Correctness**: Given some noisy input-output data and assumptions on the noise, bounds on the correctness of the network would be desirable. As suggested above, the Fisher Information matrix may inform this problem.
- **Blind Reconstruction**: The blind reconstruction problem [12, 18, 89] assumes that the inputs into the system cannot be measured. Such a problem arises naturally in many domains (e.g., the stock market, where we measure all the outputs which are the stock prices, but none of the market inputs). Typically, the problem assumes that the outputs are all perturbed by independent white noise (i.e., $P = I$ and $u_i(t) \sim \mathcal{N}(0, \sigma)$ for some σ and all $i = 1, \dots p$). As such, one way to perform this reconstruction (as suggested in [12]) is to treat all inputs as if they were measured to be zero, but have some measurement noise. We then perform a robust reconstruction on the resulting input-output data to get an approximation of the network. Unfortunately, by Theorem 4.4.7, this choice of inputs fails to meet the necessary data informativity conditions. Future work may take advantage of the knowledge of $P = I$, and possibly knowledge of σ, to find another technique that does not violate data informativity.

- **Partial Blind Reconstruction**: This is a variation of the blind reconstruction problem where we measure some, but not all, of our inputs. Depending on the nature of the unmeasured inputs, this may be equivalent to the robust reconstruction problem.
- **Reconstruction of Non-FIR Edges**: Currently, all edges of the network are required to be finite impulse responses (which allows the matrices L and M to be finite). Future work may generalize and allow for more types of impulse responses.
- **Reconstruction of Unstable Networks**: Currently, we require our network to be BIBO stable so that the inputs do not become negligibly small with respect to our outputs. Future work may allow for the reconstruction of unstable networks, perhaps by building a controller that chooses $u(t)$ to stabilize the network (so long as the input data generated satisfies Theorem 4.4.7).
- **Moving Horizon Reconstruction**: Presently, the reconstruction process must be reinitialized from scratch whenever new data is received. By leveraging recursive least squares, the algorithm could be generalized such that it updates an existing model with new data. Thus the model can be refined in real time as new data is streamed in.
- **Model Predictive Control**: This is not related directly to network reconstruction; however, given the DSF in the form (4.57), choosing $u(t)$ optimally over some time horizon such that $y(t)$ follows a desired trajectory becomes a linear program. Moreover, a demonstration of the effectiveness of this technique compared to existing model predictive control techniques would be useful to the community, especially if it can be shown that knowledge of the DSF structure can allow the engineer to design the controller to be robust in ways unavailable to black-box models.
- **Adaptive Control**: Once the moving horizon reconstruction and the model predictive control work is done, then they can be combined to solve the adaptive control problem. The combined solution would involve feedback between a recursive least squares problem and a linear program. Since both are linear, proofs on stability may be readily obtainable.

- **Linear Reconstruction of a Non-linear System**: Suppose we use the time-domain reconstruction algorithm to reconstruct a model of data that was generated from a non-linear system (such as neural activity in a brain). Does the reconstructed model have meaning with respect to the true system?

Chapter 5

Conclusions

In conclusion, we have established a foundation for the theory of proper linear dynamic networks as represented by the dynamical structure function (DSF). We introduced the dynamical network function (DNF) as a generalization of the DSF and have provided well-posedness results over the DNF. We also demonstrated that the set of state space models is not rich enough to generate the full set of DNFs; they can only generate the restricted set where W is strictly proper. However, we showed that the interconnection of many state space models, as represented by a generalized state space model, is rich enough to represent the full set of DNFs. We also demonstrated the connection between the well-posedness of generalized state space models and the well-posedness of DNFs.

We then discussed the notions of abstractions of dynamic networks, which are themselves dynamic networks preserving specific desired properties from the original network but containing less structural information. As such, abstractions are cheaper to reconstruct from data than the original network. We presented four different types of abstractions. The first, the node abstraction, occurs when we no longer model (or measure) specific outputs in the network. The abstraction preserves the input-output behavior of the network and introduces direct edges between measured variables that were once indirect paths passing through the now-hidden variables. The second abstraction is the edge abstraction which seeks to force certain edges to zero while preserving the remaining topology of the network. We discuss one such edge abstraction called the hollow abstraction, which forces all diagonal entries of W to be zero, thus converting a DNF into a DSF.

The third abstraction we discuss is called an *immersion*. We show that the immersion is a composition of a node abstraction and an edge abstraction. Thus the immersion of a DSF is a DSF, and the immersion is the node abstraction of a DSF that also results in a DSF. The final abstraction is called the stacked immersion and was designed such that, unlike the node abstraction and the immersion, the number of measured outputs does not decrease. With this construct, we were able to show that there exists a DSF containing as much structural information as a state space model, that there exists a stacked immersion of this DSF containing as little structural information as a transfer function, and that all other stacked immersions lie on a spectrum defined by structural information and lying between these two endpoints.

For every abstraction, we introduce the notion of representability. We always assume that the base network is a proper (and not necessarily strictly proper) network. However, the abstraction may not be proper, and so we provide conditions under which the abstraction is proper (which we call representability conditions). We show that the definition of representability is very similar to the definition of well-posedness. We then show that assuming the abstraction is representable (proper), the abstraction is well-posed if and only if the base network is well-posed.

Finally, we discuss the network reconstruction problem. In this problem, we begin either with input-output data or an input-output map (the transfer function) and seek to recover the unique structured model at some desired level of abstraction given that data. We show that this problem is impossible to solve without some a priori knowledge about the structure of our model and that the a priori information we choose selects a particular abstraction. We present two algorithms to solve the network reconstruction problem, one operating in the frequency domain and the other in the time domain. While these algorithms were known and published before this work, the main contribution here is to generalize these algorithms to reconstruct proper (as opposed to strictly proper) DNFs (as opposed to DSFs).

Appendix A

Source Code

The `pyrics` library is intended to become a comprehensive Python library for the representation, identification, and control of dynamic systems. It began its life as a part of this dissertation in order to provide a framework to perform the network reconstruction in Chapter 4. Every example within that chapter was produced using this framework using the Jupyter notebooks included below.

The framework also includes code for converting state space models and generalized state space models into DSFs (Chapter 2). It can check the well-posedness of those DSFs (also Chapter 2) and find immersions of those networks given some set S (Chapter 3). Presently, it does not support DNFs directly (though it will indirectly as shown with Example 4.3.15), nor does it support the remaining abstractions outlined in Chapter 3.

This code is available as at `https://gitlab.com/idealabs/pyrics`. The code included below was taken from the version 1.0 release of 'pyrics' (`https://gitlab.com/idealabs/pyrics/commit/7c8f7a9d3b352aee89fe34b88f7bae5f467f8a69`).

A.1 Jupyter Notebooks for Examples

Here, we include the Jupyter notebooks (running a Python 3 kernel; outputs suppressed) that were used to generate the examples in Chapter 4.

Example 4.3.12 - A Target-Specific, Proper, and Ill-Posed DSF

This example is, perhaps, the simplest example of reconstruction. Reconstruction of target-specific strictly proper DSFs have long been demonstrated; however, the algorithm was always capable of reconstructing proper networks as well, even if they are ill-posed (see Chapter 2 for definitions of well-posed and ill-posed networks). We demonstrate that here.

```
In [ ]: import numpy as np
        from pyrics.Representations import DTDSF, DTTF
        from pyrics.Algorithms import frequencyReconstruct as reconstruct
        from sympy.abc import z
        from sympy.matrices import Matrix
```

Step 1

Build the baseline DSF. Our final answer should equal the DSF shown here.

Note that this network is ill-posed.

```
In [ ]: Q = DTTF([
            [0, (z + 2) / (z + 1), 0],
            [0, 0, (z + 3) /(z + 4)],
            [(z + 1) * (z + 5) /(z ** 2 + 5*z + 6), 1 / (z ** 2 + 2), 0]
        ])
        P = DTTF([
            [(z + 4)/(z + 1), 0, 0],
            [0, 1/(z ** 2 + 2), 0],
            [0, 0, (z + 6)/(z + 3)]
        ])

        F = DTDSF(Q, P)
        F.display()
        print('Is Well-Posed? {}'.format(F.is_wellposed()))
```

Step 2

Convert the DSF into a transfer function. This is what we are given (along with the necessary and sufficient identifiability conditions) in the reconstruction process.

Note that this transfer function is improper, which is an artifact of using an ill-posed network.

```
In [ ]: G = F.to_TF()
        G.display()
        print(G.latex())
```

Step 3

Reconstruct the DSF from the transfer function. We are assuming target specificity (P is diagonal), which is the default parameters into the reconstruct function.

Note that the DSF found here is the same as that we started with, indicating that the reconstruction process works as intended.

```
In [ ]: # Fr = G.reconstruct()
        Q, P = reconstruct(G.G)
        Fr = DTDSF(DTTF(Q), DTTF(P))
        Fr.display()
```

Appendix

We now print out some intermediate steps in computation.

```
In [ ]: DTTF(reconstruct.K).display(name='K')
        print(DTTF(reconstruct.K).latex(name='K'))

In [ ]: DTTF(reconstruct.L).display(name='L')
        print(DTTF(reconstruct.L).latex(name='L'))

In [ ]: DTTF(reconstruct.M).display(name='M')
        print(DTTF(reconstruct.M).latex(name='M'))

In [ ]: DTTF(reconstruct.gvec).display(name=r'\vec{g}')
        print(DTTF(reconstruct.gvec).latex(name=r'\vec{g}'))

In [ ]: DTTF(reconstruct.thetahat).display(name=r'\hat{\theta}')
        print(DTTF(reconstruct.thetahat).latex(name=r'\hat{\theta}'))

In [ ]: DTTF(reconstruct.theta).display(name=r'\theta')
        print(DTTF(reconstruct.theta).latex(name=r'\theta'))

In [ ]: # For testing only, show a immersion
        F.immerse([0,1]).display()
```

Example 4.3.13 - Exceeding the Identifiability Index

In this example, we will demonstrate the reconstruction of a DSF when $m > p$ and where we have more structural information than necessary to reconstruct.

```
In [ ]: import numpy as np
        from pyrics.Representations import DTDSF, DTTF
        from pyrics.Algorithms import frequencyReconstruct as reconstruct
        from sympy.abc import z
        from sympy.matrices import Matrix
```

Step 1

Build the baseline DSF. Our final answer should equal the DSF shown here.

```
In [ ]: Q = DTTF([
            [0, 1/(z+1)],
            [1/(z+2), 0]
        ])
        P = DTTF([
            [1/(z+3),0,0],
            [1/(z+3) + 1/(z+4), 1/(z+4),1/(z+5)]
        ])

        F = DTDSF(Q, P)
        F.display()
        print('Is Well-Posed? {}'.format(F.is_wellposed()))
```

Step 2

Encode the Identifiability Conditions.

```
In [ ]: K = np.array([
            [0, 0, 0, 0, 0],  # Q11 = 0
            [1, 0, 0, 0, 0],  # Q12 = ?1
            [0, 1, 0, 0, 0],  # Q21 = ?2
            [0, 0, 0, 0, 0],  # Q22 = 0
            [0, 0, 1, 0, 0],  # P11 = ?3
            [0, 0, 0, 0, 0],  # P12 = 0
            [0, 0, 0, 0, 0],  # P13 = 0
            [0, 0, 1, 1, 0],  # P21 = P11 + P22
            [0, 0, 0, 1, 0],  # P22 = ?4
            [0, 0, 0, 0, 1],  # P23 = ?5
        ])
        print(DTTF(K).latex())
```

Step 3

Convert the DSF into a transfer function. This is what we are given (along with identifiability conditions) in the reconstruction process.

```
In [ ]: G = F.to_TF()
        G.display()
        print(G.latex())
```

Step 4

Reconstruct the DSF from the transfer function. We are assuming target specificity (P is diagonal), which is the default parameters into the reconstruct function.

Note that the DSF found here is the same as that we started with, indicating that the reconstruction process works as intended.

```
In [ ]: # Fr = G.reconstruct()
        Q, P = reconstruct(G.G, K=K)
        Fr = DTDSF(DTTF(Q), DTTF(P))
        Fr.display()
```

Appendix

Print out intermediate computations.

```
In [ ]: DTTF(reconstruct.K).display(name='K')
        print(DTTF(reconstruct.K).latex())

In [ ]: DTTF(reconstruct.L).display(name='L')
        print(DTTF(reconstruct.L).latex(name='L'))

In [ ]: DTTF(reconstruct.M).display(name='M')
        print(DTTF(reconstruct.M).latex(name='M')))

In [ ]: DTTF(reconstruct.gvec).display(name=r'\vec{g}')
        print(DTTF(reconstruct.gvec).latex(name=r'\vec{g}'))

In [ ]: DTTF(reconstruct.thetahat).display(name=r'\hat{\theta}')
        print(DTTF(reconstruct.thetahat).latex())

In [ ]: DTTF(reconstruct.theta).display(name=r'\theta')
        print(DTTF(reconstruct.theta).latex())
```

Example 4.3.14 - A DNF from a General Field

In this example, we demonstrate the reconstruction of a DNF (instead of a DSF) where elements of the DNF are functions of some $z > 0 \in \mathbb{R}$ (instead of rational functions of $z \in \mathbb{C}$).

```
In [ ]: from sympy.matrices import Matrix, eye
        from sympy.abc import z
        from sympy import sin, cos, log, simplify, latex
        from pyrics.Algorithms import frequencyReconstruct
        from pyrics.utilities import pprint
        import numpy as np
```

Step 1

Build the baseline DSF. Our final answer should equal the DSF shown here.

```
In [ ]: Q = Matrix([
            [sin(z), log(z) / z],
            [0, sin(z) - log(z) / z]
        ])
        P = Matrix([
            [z ** 2 + 2 * z + 3, z],
            [sin(z), z]
        ])
        pprint(Q)
        pprint(P)
```

Step 2

Convert the DSF into a transfer function. This is what we are given (along with the identifiability conditions) in the reconstruction process.

```
In [ ]: G = simplify((eye(2) - Q).inv() * P)
        print(latex(G))
        pprint(G)
```

Step 3

Encode the identifiability conditions. We know that $Q_{11} = 0$, $Q_{22} = Q_{11} - Q_{12}$, $P_{21} = Q_{11}$, and $P_{22} = P_{12}$.

```
In [ ]: K = np.array([
            [1, 0, 0, 0],   # Q11 = ?1
            [0, 1, 0, 0],   # Q12 = ?2
            [0, 0, 0, 0],   # Q21 = 0
            [1, -1, 0, 0],  # Q22 = Q11 - Q12
            [0, 0, 1, 0],   # P11 = ?3
            [0, 0, 0, 1],   # P12 = ?4
            [1, 0, 0, 0],   # P21 = Q11
            [0, 0, 0, 1],   # P22 = P12
        ])
```

Step 4

Reconstruct the network. We should get what we began with.

```
In [ ]: W, V = frequencyReconstruct(G, K=K)
        pprint(simplify(W))
        pprint(simplify(V))
```

Appendix

Display intermediate computations.

```
In [ ]: gvec = simplify(frequencyReconstruct.gvec)
        pprint(gvec)
        print(latex(gvec))

In [ ]: M = simplify(frequencyReconstruct.M)
        pprint(M)
        print(latex(M))

In [ ]: thetahat = simplify(frequencyReconstruct.thetahat)
        pprint(thetahat)
        print(latex(thetahat))

In [ ]: theta = simplify(frequencyReconstruct.theta)
        pprint(theta)
        print(latex(theta))
```

Example 4.3.15 - Reconstructing an Immersion

Reconstruction of an immersion of a network.

```
In [ ]: import numpy as np
        from pyrics.Representations import DTDSF, DTTF
        from sympy.abc import z
        from sympy.matrices import Matrix
        from pyrics.Algorithms import frequencyReconstruct as reconstruct
```

Step 1

Build the baseline DSF. This will remain unknown.

```
In [ ]: Q = DTTF([
            [0, 1 / (z + 1), 0],
            [0, 0, 1/(z + 2)],
            [1 / (z + 3), 1 / (z + 4), 0]
        ])
        P = DTTF([
            [1/(z + 5), 0, 0],
            [0, 1/(z + 6), 0],
            [0, 0, 1/(z + 7)]
        ])

        F = DTDSF(Q, P)
        F.display()
        print(F.Q.latex())
        print(F.P.latex())
```

Step 2

Compute our (unknown) immersion with $S = \{1,2\}$ (or $[0,1]$ in the Python zero-indexing) and $\bar{S} = 3$. Our final answer should equal this.

```
In [ ]: FS = F.immerse([0, 1])
        FS.display()
```

Step 3

Compute the transfer function of the immersion, which we know and use to reconstruct.

```
In [ ]: G = F.to_TF()
        G.display()
        print(G.latex())
        GS = FS.to_TF()
        GS.display()
```

Step 4

Define the identifiability conditions.

```
In [ ]: K = np.array([
            [0, 0, 0, 0, 0, 0],  # Q11 = 0
            [1, 0, 0, 0, 0, 0],  # Q12 = ?1
            [0, 1, 0, 0, 0, 0],  # Q21 = ?2
            [0, 0, 0, 0, 0, 0],  # Q22 = 0
            [0, 0, 1, 0, 0, 0],  # P11 = ?3
            [0, 0, 0, 0, 0, 0],  # P12 = 0
            [0, 0, 0, 1, 0, 0],  # P13 = ?4
            [0, 0, 0, 0, 0, 0],  # P21 = 0
            [0, 0, 0, 0, 1, 0],  # P22 = ?5
            [0, 0, 0, 0, 0, 1],  # P23 = ?6
        ])
        print(DTTF(K).latex())
```

Step 5

Reconstruct the network. Note that we get the right answer.

```
In [ ]: Q, P = reconstruct(GS.G, K=K)
        Fr = DTDSF(DTTF(Q), DTTF(P))
        Fr.display()
        print(Fr.Q.latex(name='Q_S'))
        print(Fr.P.latex(name='P_S'))
```

Appendix

Print intermediate computations.

```
In [ ]: DTTF(reconstruct.L).display(name='L')
        print(DTTF(reconstruct.L).latex(name='L'))

In [ ]: DTTF(reconstruct.gvec).display(name=r'\vec{g}')
        print(DTTF(reconstruct.gvec).latex(name=r'\vec{g}'))

In [ ]: DTTF(reconstruct.M).display(name='M')
        print(DTTF(reconstruct.M).latex(name='M'))

In [ ]: DTTF(reconstruct.thetahat).display(name=r'\hat{\theta}')
        print(DTTF(reconstruct.thetahat).latex(name=r'\hat{\theta}'))

In [ ]: DTTF(reconstruct.theta).display(name=r'\theta')
        print(DTTF(reconstruct.theta).latex(name=r'\theta'))
```

Example 4.3.16 - Assuming a DSF When Reconstructing a DNF

Demonstration that incorrect assumptions for the necessary and sufficient identifiability conditions results in an abstraction of the true network.

```
In [ ]: from sympy.matrices import Matrix, eye
        from sympy.abc import z
        from sympy import sin, cos, log, simplify, latex
        from pyrics.Algorithms import frequencyReconstruct
        from pyrics.utilities import pprint
        from pyrics.Representations import DTTF
        import numpy as np
```

Step 1

Define the DNF. TODO for future, build DNF in pyrics.Representations. For now, we just build it outside and reconstruct a DSF.

```
In [ ]: W = DTTF([
            [1 / (z + 1), 1 / (z + 2)],
            [1 / (z + 3), 1 / (z + 4)]
        ])
        V = DTTF([
            [1 / (z + 5), 0],
            [0, 1 / (z + 6)]
        ])
        W.display(name='W')
        print(W.latex(name='W'))
        V.display(name='V')
        print(V.latex(name='V'))
        print(W.diag().display(name='D_W'))
```

Step 2

Compute the Hollow Abstraction of the DNF. We will show that the reconstruction that assumes the identifiability conditions for a DSF returns this answer.

```
In [ ]: I = DTTF(eye(2))
        Q = (I - W.diag()).inv() * (W - W.diag())
        P = (I - W.diag()).inv() * V

        Q.display(name='Q')
        print(Q.latex(name='Q'))
        P.display(name='P')
        print(P.latex(name='P'))
```

Step 3

Compute the Transfer Function.

```
In [ ]: G = (I - W).inv() * V
        G.display(name='G')
        print(G.latex(name='G'))
```

Step 4

Compute the reconstruction of the DNF. Note that we actually get the hollow abstraction instead.

```
In [ ]: QS, PS = frequencyReconstruct(G.G)
        QS = DTTF(QS)
        PS = DTTF(PS)
        QS.display(name='Q_S')
        print(QS.latex(name='Q_S'))
        PS.display(name='P_S')
        print(QS.latex(name='P_S'))
```

Appendix

Print intermediate results

```
In [ ]: print(DTTF(frequencyReconstruct.L).latex(name='L'))

In [ ]: print(DTTF(frequencyReconstruct.M).latex(name='M'))

In [ ]: print(DTTF(frequencyReconstruct.gvec).latex(name=r'\vec{g}'))

In [ ]: print(DTTF(frequencyReconstruct.thetahat).latex(name=r'\hat{\theta}'))

In [ ]: print(DTTF(frequencyReconstruct.theta).latex(name=r'\theta'))
```

Example 4.4.9 - A Strictly Proper Network

We reproduce the example that was first presented in [1] to demonstrate that this algorithm–which was generalized to reconstruct proper networks–still functions when attempting to reconstruct a strictly proper network.

[1] V. Chetty, J. Eliason and S. Warnick, "Passive Reconstruction of Non-Target-Specific Discrete-Time LTI Systems," American Control Conference, Boston, MA, 2016.

```
In [ ]: import numpy as np
        import sympy as sp
        from sympy import latex
        from pyrics.Representations import DTSS, DTTF, ImpulseDSF
        from pyrics.Plot import plotSimulation, plotTimeReconstructionComparison
        from IPython.display import display, Math

        import plotly
        from plotly.offline import iplot as plot
        plotly.offline.init_notebook_mode(connected=True)
```

Step 1

Define constants. We cheat to find r by knowing that all impulse responses have time to converge before 200 timesteps. We set $T = 3(r+1) > m(r+1) - 1$ since $m = 3$.

```
In [ ]: r = 200
        T = 3 * (r + 1)  # T >= m(r + 1) - 1
```

Step 2

Define the state space model of our system. We use this model to generate our data.

```
In [ ]: A = np.array([
            [0.75, 0, 0, 0, 0, 1.2],
            [-.1, -.35, 0, 0, 0, 0],
            [0, 0, 0.85, -1, 0, 0],
            [0, -0.73, 0, 0.95, 0, 0],
            [0, 0, 0.43, 0, -0.6, 0],
            [0, 0, 0, 0, 0.2, 0.55]
        ])
        B = np.array([
            [1.4, 0, -1.4],
            [0, -0.25, 0],
            [0, 0, 0.75],
            [0, 0, 0],
            [0, 0, 0],
            [0, 0, 0]
        ])

        C = np.array(
```

```
        [
            [1, 0, 0, 0, 0, 0],
            [0, 1, 0, 0, 0, 0],
            [0, 0, 1, 0, 0, 0]
        ]
    )
    ss = DTSS(A, B, C)
    ss.display()
    print(ss.is_stable())

    Du = np.random.normal(size=(T, 3))
    Dy = ss.simulate(Du)
    plotSimulation(Du, Dy, plot)
```

Step 3

Encode the identifiability conditions.

```
In [ ]: K = np.array([
            [0, 0, 0, 0, 0, 0, 0, 0, 0],  # Q11 = 0
            [1, 0, 0, 0, 0, 0, 0, 0, 0],  # Q12 = ?1
            [0, 1, 0, 0, 0, 0, 0, 0, 0],  # Q13 = ?2
            [0, 0, 1, 0, 0, 0, 0, 0, 0],  # Q21 = ?3
            [0, 0, 0, 0, 0, 0, 0, 0, 0],  # Q22 = 0
            [0, 0, 0, 1, 0, 0, 0, 0, 0],  # Q23 = ?4
            [0, 0, 0, 0, 1, 0, 0, 0, 0],  # Q31 = ?5plotPassiveComparison
            [0, 0, 0, 0, 0, 1, 0, 0, 0],  # Q32 = ?6
            [0, 0, 0, 0, 0, 0, 0, 0, 0],  # Q33 = 0
            [0, 0, 0, 0, 0, 0, 1, 0, 0],  # P11 = ?7
            [0, 0, 0, 0, 0, 0, 0, 0, 0],  # P12 = 0
            [0, 0, 0, 0, 0, 0, -1, 0, 0],  # P13 = -P11
            [0, 0, 0, 0, 0, 0, 0, 0, 0],  # P21 = 0
            [0, 0, 0, 0, 0, 0, 0, 1, 0],  # P22 = ?8
            [0, 0, 0, 0, 0, 0, 0, 0, 0],  # P23 = 0
            [0, 0, 0, 0, 0, 0, 0, 0, 0],  # P31 = 0
            [0, 0, 0, 0, 0, 0, 0, 0, 0],  # P32 = 0
            [0, 0, 0, 0, 0, 0, 0, 0, 1],  # P33 = ?9
        ])
```

Step 4

Convert the state space model into our (unknown) DSF. Convert also into convolutional form and then impulse form so that we have a basis of comparison to see that our reconstructed network is correct.

```
In [ ]: QP = ss.to_DSF()
        QP.display()
```

```
In [ ]: QPconv = QP.to_convolutional()
        QPconv.display()

In [ ]: QPimpulse = QPconv.to_impulse(r=r)
        QPimpulse.display()
        # QPimpulse.get_plotly(Qplot=plot, Pplot=plot)
        pass
```

Step 5

Reconstruct the network given the input-output data, r, and the identifiability conditions in K. Optionally convert the reconstructed network back into convolutional form and the frequency domain (commented out).

```
In [ ]: RQPimp = ImpulseDSF.reconstruct(Du, Dy, r, K=K, verbose=True, precision=1e-4)
        # RQPimp.get_plotly(Qplot=plot, Pplot=plot)
        pass

In [ ]: # RQPconv = RQPimp.to_convolutional(verbose=True, njobs=8)
        # RQPconv.display()

In [ ]: # RQP = RQPconv.to_DSF()
        # RQP.display()
```

Step 6

Plot results to compare the actual impulse responses with the reconstructed impulse responses.

```
In [ ]: plotTimeReconstructionComparison(plot, RQPimp, QPconv=None, QPact=QP)
```

Example 4.4.10 - A Proper Network

We now demonstrate that the time-domain network reconstruction algorithm is capable of reconstructing proper (and not necessarily strictly proper) networks.

```
In [ ]: import numpy as np
        import sympy as sp
        from sympy import latex
        from pyrics.Representations import DTGSS, DTTF, ImpulseDSF
        from pyrics.Plot import plotSimulation, plotTimeReconstructionComparison
        from IPython.display import display, Math

        import plotly
        from plotly.offline import iplot as plot
        plotly.offline.init_notebook_mode(connected=True)
```

Step 1

Define constants. We cheat to find r by knowing that all impulse responses have time to converge before 200 timesteps. We set $T = 2400 \geq m(r+1) - 1 = 602$.

```
In [ ]: T = 2400
        r = 200
```

Step 2

Define the state space model of our system. We use this model to generate our data.

```
In [ ]: A = np.array([
            [0.75, 0, 0, 0, 0, 1.2],
            [-.1, -.35, 0, 0, 0, 0],
            [0, 0, 0.85, -1, 0, 0],
            [0, -0.73, 0, 0.95, 0, 0],
            [0, 0, 0.43, 0, -0.6, 0],
            [0, 0, 0, 0, 0.2, 0.55]
        ])

        Ahat = np.zeros((6, 3))

        B = np.array([
            [1.4, 0, 0],
            [0, -0.25, 0],
            [0, 0, 0.75],
            [0, 0, 0],
            [0, 0, 0],
            [0, 0, 0]
        ])

        Abar = np.array(
```

```
        [
            [1, 0, 0, 0, 0, 0],
            [0, 1, 0, 0, 0, 0],
            [0, 0, 1, 0, 0, 0]
        ]
    )

    Atilde = np.array([
        [4, 1, 1],
        [0, 3, 0],
        [0, 1, 2]
    ])

    Bbar = np.array(
        [
            [0, 0, 0],
            [0, 0, 0],
            [0, 0, 0]
        ]
    )
    gss = DTGSS(A, Ahat, B, Abar, Atilde, Bbar)
    gss.display()
    print(gss.is_stable())

    Du = np.random.normal(size=(T, 3))
    Dy = gss.simulate(Du)
    plotSimulation(Du, Dy, plot)
```

Step 3

Encode the identifiability conditions. Since we can assume target specificity (encoded by default), we don't need to do anything here.

Step 4

Convert the state space model into our (unknown) DSF. Convert also into convolutional form and then impulse form so that we have a basis of comparison to see that our reconstructed network is correct.

```
In [ ]: QP = gss.to_DSF()
        QP.display()
        print(QP.is_wellposed())
        print(QP.Q.limit())

In [ ]: QPconv = QP.to_convolutional()
        QPconv.display()

In [ ]: QPimpulse = QPconv.to_impulse(r=r)
        QPimpulse.display()
```

```
        # QPimpulse.get_plotly(Qplot=plot, Pplot=plot)
        pass
```

Step 5

Reconstruct the network given the input-output data and r. Optionally convert the reconstructed network back into convolutional form and the frequency domain (commented out).

```
In [ ]: RQPimp = ImpulseDSF.reconstruct(Du, Dy, r, K=None, verbose=False, precision=1e-4)
        # RQPimp.get_plotly(Qplot=plot, Pplot=plot)
        pass

In [ ]: # RQPconv = RQPimp.to_convolutional(verbose=True, njobs=8)
        # RQPconv.display()

In [ ]: # RQP = RQPconv.to_DSF()
        # RQP.display()
```

Step 6

Plot results to compare the actual impulse responses with the reconstructed impulse responses.

```
In [ ]: # plotPassiveComparison(plot, RQPimp, RQPconv, QP)
        plotTimeReconstructionComparison(plot, RQPimp, QPconv=None, QPact=QP)
```

A.2 The pyrics Library

Here, we include the source code for the pyrics library used to run the above notebooks.

Listing 1 pyrics.Algorithms.FrequencyDomainReconstruction

```
#-------------------------------------------------------------
# pyrics/Algorithms/FrequencyDomainReconstruction.py
#
# The frequency-domain reconstruction algorithm.
#
# Copyright 2019 Nathan Woodbury
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
# http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
#-------------------------------------------------------------

from sympy import factor
from sympy.matrices import Matrix

from pyrics.Algorithms.Identifiability import checkIdentifiability

#-------------------------------------------------------------
def frequencyReconstruct(G, K=None):
  '''Reconstruct the network (Q, P) generating G.

  Note that G only needs to be a sympy matrix, and note that Q, P will also be
  sympy matrices.

  Methodology described in [1, 2], following the notation of [2].

  Attributes
  ----------
  (these are available at `active_reconstruct.attribute` and can be used to
  view and access the intermediate computations)

  Munstacked : sympy matrix (p x [p + m])
    The augmented matrix [G' I].
  Mstacked : sympy matrix (pm x [p^2 + pm])
    Munstacked that has been converted into stacked form (after vectorizing)
    the unknowns in Q and P
  M : sympy matrix (pm x pm)
    Mstacked after the identifiability conditions have been applied to reduce
    the number of columns.
  y : sympy matrix (pm x 1)
    The vector containing the known elements from G, stacked.
  x : sympy matrix (pm x 1)
    The vector containing the reconstructed elements from Q and P, stacked,
    where x = M^-1 y.

  Parameters
  ----------
  G : sympy matrix (p x m)
    The 'transfer function'
  K : np.array (p^2 + pm x pm) or None, default=None
    The identifiability conditions required to map the TF to the DSF uniquely.
    If None, assumes that P is diagonal (target specificity); however, an
    exception is raised in this case if p != m.
  display_intermediate : bool, default=False
    If true, prints the intermediate steps, including M (unstacked),
    M (stacked), y, x, Mhat, and xhat.

  Returns
```

```
    -------
    Q : sympy matrix (p x p)
    P : sympy matrix (p x m)

    Sources
    -------
    [1] J. Gonalves and S. Warnick, "Necessary and Sufficient Conditions
        for Dynamical Structure Reconstruction of LTI Networks," IEEE
        Transactions on Automatic Control, Aug. 2008.
    [2] N. Woodbury, "Representation and Reconstruction of Linear, Time-Invariant
        Networks," Ph.D Dissertation, Brigham Young Univ., Provo, UT, 2019.
    '''
    self = frequencyReconstruct
    p, m = G.shape

    K, _ = checkIdentifiability(K, p, m)
    _, k = K.shape

    Lunstacked = G.transpose().row_join(Matrix.eye(m))
    LQ = Matrix.zeros(p * m, p ** 2)
    GT = G.transpose()
    for i in range(p):
      start_row = i * m
      end_row = (i + 1) * m
      start_col = i * p
      end_col = (i + 1) * p
      LQ[start_row: end_row, start_col: end_col] = GT

    L = LQ.row_join(Matrix.eye(p * m))
    M = L * K

    gvec = Matrix.zeros((p * m), 1)
    for i in range(p):
      start = m * i
      end = m * (i + 1)
      gvec[start: end, 0] = G[i, :].transpose()

    if k == p * m:
      # from pyrics.utilities import pprint
      # pprint(M)
      thetahat = M.inv() * gvec
    else:
      thetahat = factor((factor(M.transpose() * M)).inv()) * M.transpose() * gvec

    thetahat = factor(thetahat)
    theta = factor(K * thetahat)

    Q = Matrix.zeros(p, p)
    P = Matrix.zeros(p, m)
    xix = 0
    for i in range(p):
      for j in range(p):
        Q[i, j] = theta[xix, 0]
        xix += 1

    for i in range(p):
      for j in range(m):
        P[i, j] = theta[xix, 0]
        xix += 1

    self.Lunstacked = Lunstacked
    self.L = L
    self.M = factor(M)
    self.gvec = gvec
    self.thetahat = factor(thetahat)
    self.K = K
    self.theta = theta

    return Q, P
```

Listing 2 pyrics.Algorithms.Identifiability

```
#-----------------------------------------------------------
# pyrics/Algorithms/identifiability.py
```

```
#
# Verifies and encodes identifiability conditions.
#
# Copyright 2019 Nathan Woodbury
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
# http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
#-----------------------------------------------------------------

import numpy as np
from numpy.linalg import pinv, matrix_rank as rank

def checkIdentifiability(K, p, m):
  '''Ensure that the indentifiability conditions are appropriate, or create them
  if not given (assumes target specificity with hollow Q if not given).

  The DSF being reconstructed has Q as p x p and P as p x m.

  Raises
  ------
  ValueError if K does not appropriately encode indentifiability conditions.
  This is triggered if:
    - K is not (p^2 + pm x pm)
    - K is not full column rank (i.e., rank(K) != pm)

  Parameters
  ----------
  K : None or np.array (p^2 + pm x k)
    The encoding of the identifiability conditions, or None if target
    specificity is to be encoded. Must have k <= pm and rank K = k.
  p : int > 0
  m : int > 0

  Returns
  -------
  K : (p^2 + pm x pm)
    A copy of K if K was not None, or the newly created K if it was none.
  Ki : (pm x p^2 + pm)
    The Moore-penrose pseudo inverse of K (i.e., Ki * K = I_pm, where I_pm is
    the pm x pm identity matrix).
  '''
  p2 = p * p
  pm = p * m
  p2pm = p2 + pm

  if K is None:
    if p != m:
      raise ValueError((
        'If K is not given, must have p = m (P square) in order to encode '
        'target specificity. Given p = {} and m = {}'
      ).format(p, m))
    K11 = np.eye(p2)
    K22 = np.eye(pm)
    Qkeep = []
    Pkeep = []
    ix = 0
    for i in range(p):
      for j in range(m):
        if i == j:
          Pkeep.append(ix)
        else:
          Qkeep.append(ix)
        ix += 1
    K11 = K11[:, Qkeep]
    K22 = K22[:, Pkeep]
    K12 = np.zeros((p2, m))
```

```
      K21 = np.zeros((pm, p2 - p))
      Ktop = np.concatenate((K11, K12), axis=1)
      Kbot = np.concatenate((K21, K22), axis=1)
      K = np.concatenate((Ktop, Kbot), axis=0)

    rows, cols = K.shape

    if rows != p2pm:
      raise ValueError((
        'K must have p^2 + pm rows. Given p^2 + pm = {}, but K has {} rows'
      ).format(p2pm, rows))
    if cols > pm:
      raise ValueError((
        'K must have pm or fewer columns. Given pm = {}, but K has {} columns'
      ).format(pm, cols))

    rK = rank(K)

    if rK != cols:
      raise ValueError((
        'K must be full-column rank (i.e., rank(K) must be k = {}), '
        'but rank(K) = {}'
      ).format(cols, rK))

    return K, pinv(K)
```

Listing 3 pyrics.Algorithms.TimeDomainReconstruction

```
#--------------------------------------------------------------
# pyrics/Algorithms/TimeDomainReconstruction.py
#
# The time-domain network reconstruction algorithm.
#
# Copyright 2019 Nathan Woodbury
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
# http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
#--------------------------------------------------------------

from functools import partial
import numpy as np

from pyrics.utilities import vprint as vprintfull
from pyrics.Algorithms.Identifiability import checkIdentifiability
from pyrics.Representations.ImpulseScalar import ImpulseScalar
from pyrics.Representations.ImpulseTF import ImpulseTF
from pyrics.Representations.ImpulseDSF import ImpulseDSF

#--------------------------------------------------------------
def timeReconstruct(Du, Dy, r, K=None, tol=0.001, precision=1e-6,
                    verbose=False):
  '''Reconstruct an impulse response from input data Du and output data Dy.

  Parameters
  ----------
  Du : numpy array (T x m)
    The input data, where the t'th row of Du is u(t) (transpose).
  Du : numpy array (T x p)
    The output data, where the t'th row of Dy is y(t) (transpose).
  r : int s.t. 0 < r <= T
    Must be large enough such that every impulse response in the reconstructed
    Q(t) and P(t) is approximately 0 for every t > r.
  K : np.array (p^2 + pm x pm) or None, default=None
```

```
    The identifiability conditions required to map the TF to the DSF uniquely.
    If None, assumes that P is diagonal (target specificity); however, an
    exception is raised in this case if p != m.
  tol : number or None
    If not None, performs a check on the goodness of the choice of r using
    the singular values of L. Let s1 be the rpm'th singular value and s2 be
    the (rpm + 1)'th singular value. Then, a warning is given if
    (s1 - s2) / s1 <= 1 - tol (since s2 should be small compared to s1).
  precision : number > 0
    Sets `ImpulseScalar.precision` for every impulse response generated.
  verbose : bool
    If True, prints status messages as the algorithm processes.

  Returns
  -------
  QP : ImpulseDSF
    The reconstructed network.
  '''
  vprint = partial(vprintfull, verbose=verbose)
  vprint('Initializing Algorithm')

  self = timeReconstruct
  T, m = Du.shape
  T1, p = Dy.shape

  assert p == m  # TODO - generalize

  if T1 != T:
    raise ValueError((
      'Du and Dy must be the same length (same number of rows). Given '
      'Du with {} rows and Dy with {} rows.'
    ).format(T, T1))

  vprint('Checking the Identifiability Conditions')
  K, _ = checkIdentifiability(K, p, m)

  vprint('Building yvec')
  self.yvec = Dy.reshape((T * p, 1))  # numpy does row-major order by default

  vprint('Building M')
  self.M = _buildM(Du, Dy, r, K)

  vprint('Running Least Squares to fit the impulse responses')
  theta, _, _, s = np.linalg.lstsq(self.M, self.yvec, rcond=None)
  theta = theta.reshape((r * p * p, 1))

  # Check singular values:
  if tol is not None:
    vprint('Checking the singular values')
    # Otherwise the singular values are zero, which is good
    if len(s) >= r * p * m + 1:
      ratio = (s[r * p * m - 1] - s[r * p* m]) / s[r * p * m]
      if (1 - ratio) >= tol:
        raise Warning((
          'Singular value tolerance violated (ratio={:.5f}, should be close to '
          '1); likely r is too small.'
        ).format(ratio))

  vprint('Extracting Q(t) and P(t) from theta')
  Q, P = _extractQP(theta, p, m, r, K, precision)

  vprint('Collecting and returning')
  return ImpulseDSF(Q, P)


def _buildM(Du, Dy, r, K):
  '''Builds the M matrix required for reconstruction, reducing according to
  the identifiability conditions.

  Parameters
  ----------
  Du : numpy array (T x m)
    The input data, where the t'th row of Du is u(t) (transpose).
  Du : numpy array (T x p)
    The output data, where the t'th row of Dy is y(t) (transpose).
  r : int s.t. 0 < r <= T
```

```
    Must be large enough such that every impulse response in the reconstructed
    Q(t) and P(t) is approximately 0 for every t > r.
  K : np.array (p^2 + pm x pm)
    The identifiability conditions required to map the TF to the DSF uniquely.

  Returns
  -------
  L : np.array (pT x r(pm + p^2))
  '''
  T, m = Du.shape
  _, p = Dy.shape

  Mbar_cache = {}

  #-----------
  def Ybar(k):
    # Build Ybark
    Ybark = np.zeros((p, p * p))
    for i in range(p):
      y = Dy[k, :].reshape((1, p))
      Ybark[i, i * p : (i + 1) * p] = y

    return Ybark

  #-----------
  def Ubar(k):
    # Build Ubark
    Ubark = np.zeros((p, m * p))
    for i in range(p):
      u = Du[k, :].reshape((1, m))
      Ubark[i, i * m : (i + 1) * m] = u

    return Ubark

  #-----------
  def Mbar(k):
    # Check if cached
    if k in Mbar_cache:
      return Mbar_cache[k]

    # Build Mbark
    Ybark = Ybar(k)
    Ubark = Ubar(k)
    Mbark = np.concatenate((Ybark, Ubark), axis=1)

    # Utilize Identifiability Conditions
    Mbark = Mbark.dot(K)

    # Cache and return
    Mbar_cache[k] = Mbark
    return Mbark

  # Build M
  _, width = K.shape
  M = np.zeros((p * T, r * width))
  for i in range(T):
    for j in range(r):
      k = i - j
      if k < 0:
        break

      M[i * p: (i + 1) * p, j * width: (j + 1) * width] = Mbar(k)

  return M

def _extractQP(theta, p, m, r, K, precision):
  '''Extract Q and P from theta.

  Parameters
  ----------
  theta : np.array (r * width x 1)
    The learned parameters in the impulse responses of Q and P.
  p : int > 0
  m : int > 0
  r : int > 0
  K : np.array (p^2 + pm x width)
  precision : number > 0

  Returns
  -------
  Q : ImpulseTF (p x p)
```

```
    P : ImpulseTF (p x m)
    '''
    Qa = []
    Pa = []
    for i in range(p):
      Qa.append([])
      Pa.append([])
      for j in range(p):
        Qa[i].append([])
      for j in range(m):
        Pa[i].append([])

    _, width = K.shape
    for k in range(r):
      x = theta[k * width : (k + 1) * width, :]
      vec = K.dot(x)

      ix = 0
      for i in range(p):
        for j in range(p):
          Qa[i][j].append(vec[ix, 0])
          ix += 1

      for i in range(p):
        for j in range(m):
          Pa[i][j].append(vec[ix, 0])
          ix += 1

    Q = []
    P = []
    for i in range(p):
      Q.append([])
      P.append([])
      for j in range(p):
        Q[i].append(ImpulseScalar(Qa[i][j], precision=precision))
      for j in range(m):
        P[i].append(ImpulseScalar(Pa[i][j], precision=precision))

    Q = ImpulseTF(Q)
    P = ImpulseTF(P)
    return Q, P
```

Listing 4 pyrics.Plot.PlotSimulation

```
#-----------------------------------------------------------
# pyrics/Plot/PlotSimulation.py
#
# Plotting utilities to show input-output data.
#
# Copyright 2019 Nathan Woodbury
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
# http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
#-----------------------------------------------------------

import plotly.graph_objs as go
from plotly import tools


def plotSimulation(Du, Dy, plot):
  '''Create a plot of the inputs in Du and the outputs in Dy.
```

```
    Only one plot will be generated, but with two subfigures. Inputs will all be
    drawn on the top subfigure, and outputs will all be drawn on the bottom
    subfigure.

    Parameters
    ----------
    Du : numpy array (T x m)
      The input data, where the t'th row of Du is u(t) (transpose).
    Du : numpy array (T x p)
      The output data, where the t'th row of Dy is y(t) (transpose).
    plot : function (plotly plot function)
      Plots a single plot of the input/output data using that function.
    '''
    T, p = Du.shape
    T1, m = Dy.shape

    if T1 != T:
      raise ValueError((
        'Du and Dy must be the same length (same number of rows). Given '
        'Du with {} rows and Dy with {} rows.'
      ).format(T, T1))

    fig = tools.make_subplots(
      rows=2, cols=1, print_grid=False,
      subplot_titles=['Input Data', 'Output Data']
    )

    for i in range(m):
      trace = go.Scatter(
        x=list(range(T)),
        y=Du[:, i],
        name=r'$u_{}(t)$'.format(i + 1)
      )
      fig.append_trace(trace, 1, 1)

    for i in range(p):
      trace = go.Scatter(
        x=list(range(T)),
        y=Dy[:, i],
        name=r'$y_{}(t)$'.format(i + 1)
      )
      fig.append_trace(trace, 2, 1)

    plot(fig)
```

Listing 5 pyrics.Plot.plotTimeReconstructionComparison

```
#-------------------------------------------------------------
# pyrics/Plot/PlotPassiveComparison.py
#
# Plotting utilities to compare time-domain reconstructed and actual models.
#
# Copyright 2019 Nathan Woodbury
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
# http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
#-------------------------------------------------------------

from math import ceil
import plotly.graph_objs as go


```

```
def plotTimeReconstructionComparison(plot, QPimp, QPconv=None, QPact=None,
                                     ylimbound=0.1):
  '''Generate plots (one for each impulse response) comparing the response in
  QPimp to that in QPconv and QPact (if given).

  This is designed such that QPimp is the response found through the least
  squares portion of the passive reconstruction algorith, QPconv is QPimp
  converted into convolutional form and QPact, if given, is the impulse response
  from the actual model that generated the data.

  Parameters
  ----------
  plot : function (plotly plot function)
    The plotly plotting function
  QPimp : ImpulseDSF
  QPconv : ConvolutionalDSF
  QPact : DiscreteDSF
  ylimbound : number > 0
    Prevents plotly from zooming in too far and making negligibly-zero impulse
    responses look non-zero.
    If there are any values less than 0, then the plot will begin at y no
    greater than ``-ylimbound` on the negative side of the y axis (otherwise,
    the plot can start at 0 on the y axis). If there are any positive values
    greater than zero, then the plot will end at y no less than `ylimbound`
    (otherwise, the plot can end at 0 on the y axis). If there are no non-zero
    values, then the y axis will range from `-ylimbound` to `ylimbound`
  '''
  p, m = QPimp.shape
  r = QPimp.r

  comma = ''
  if max(p, m) >= 10:
    comma = ','

  if QPconv is not None:
    p1, m1 = QPconv.shape
    assert p == p1
    assert m == m1

    QPconvimp = QPconv.to_impulse(r=r)
    QPactimp = None

  if QPact is not None:
    p2, m2 = QPact.shape
    assert p == p2
    assert m == m2

    QPactimp = QPact.to_convolutional().to_impulse(r=r)

  def getQorP(QorP, QP):
    assert QorP in ['Q', 'P']

    if QorP == 'Q':
      return QP.Q

    return QP.P

  def plotImp(QorP, i, j):
    traces = []

    if QPactimp is not None:
      traces.append(go.Scatter(
        x=list(range(r)),
        y=getQorP(QorP, QPactimp)[i, j].impulse,
        name='Actual Model',
        line=dict(
          width=2,
          dash='dashdot'
        )
      ))

    traces.append(go.Scatter(
      x=list(range(r)),
      y=getQorP(QorP, QPimp)[i, j].impulse,
      name='Reconstructed Impulse Response',
      mode='markers'
    ))
```

```
    if QPconv is not None:
      traces.append(go.Scatter(
        x=list(range(r)),
        y=getQorP(QorP, QPconvimp)[i, j].impulse,
        name='Projected Convolutional Form',
        mode='lines'
      ))

    name = r'${}_{{{}{}{}}}$'.format(QorP, i + 1, comma, j + 1)

    def ylimoneside(mx):
      if mx <= 0:
        return 0
      return max(ceil((mx + 0.01) * 10) / 10, ylimbound)

    def yf(fcn):
      mn = fcn(getQorP(QorP, QPimp)[i, j].impulse)
      if QPact is not None:
        mnact = fcn(getQorP(QorP, QPactimp)[i, j].impulse)
        mn = fcn(mnact, mn)
      if QPconv is not None:
        mnconv = fcn(getQorP(QorP, QPconvimp)[i, j].impulse)
        mn = fcn(mnconv, mn)

      return mn

    def ylim():
      lb = -ylimoneside(-yf(min))
      ub = ylimoneside(yf(max))

      if lb == 0 and ub == 0:
        return [-ylimbound, ylimbound]
      return lb, ub

    layout = go.Layout(
      title='{} Comparisons'.format(name),
      xaxis=dict(title=r'$t = 0, \ldots, r$'),
      yaxis=dict(title='{}(t)'.format(name), range=ylim()),
    )

    plot(dict(data=traces, layout=layout))

  # Plot impulse responses in Q
  for i in range(p):
    for j in range(p):
      plotImp('Q', i, j)

  # Plot impulse responses in P
  for i in range(p):
    for j in range(m):
      plotImp('P', i, j)
```

Listing 6 pyrics.Representations.ConvolutionalDSF

```
#-------------------------------------------------------------------
# pyrics/Representations/ConvolutionalDSF.py
#
# Representation of a DSF in convolutional form.
#
# Copyright 2019 Nathan Woodbury
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
# http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
```

```
# limitations under the License.
#-----------------------------------------------------------------

from .SystemRepresentation import SystemRepresentation
from .ConvolutionalTF import ConvolutionalTF


class ConvolutionalDSF(SystemRepresentation):
  '''Representation of a DSF where each entry is in convolutional form.

  Models

      Y(t) = Q(t) * Y(t) + P(t) * U(t),

  where * is the convolution operation and each Q_ij(t), P_ij(t) is an
  impulse response in convolutional form; i.e.,

      Q_ij(t) = a delta(t, 0) + sum_{n = 1}^w b_n c_n^t

  for parameters a, b_n, and c_n, n = 1, ..., w.

  Parameters
  ----------
  Q : numpy array (p x p)
  P : numpy array (p x m)
  '''

  #---------------------------------------------------------------
  def __init__(self, Q, P):
    self.Q = Q
    self.P = P

    if not isinstance(Q, ConvolutionalTF):
      raise ValueError('Q must be a ConvolutionalTF; given {}'.format(type(Q)))
    if not isinstance(P, ConvolutionalTF):
      raise ValueError('P must be a ConvolutionalTF; given {}'.format(type(P)))

    assert len(Q.shape) == 2
    assert len(P.shape) == 2

    p, p1 = Q.shape
    p2, m = P.shape

    if p != p1:
      raise ValueError(
        'Dimension Mismatch: Q must be square. Got dimensions ({}, {})'.format(
          p, p1
        )
      )

    if p != p2:
      raise ValueError(
        'Dimension Mismatch: Q and P must have the same number of '
        'rows. Got dimensions ({}, {}) and ({}, {})'.format(
            p, p1, p2, m
        )
      )

    self.p = p
    self.m = m

  ############################################################################
  #   Public Methods
  ############################################################################

  #---------------------------------------------------------------
  @property
  def shape(self):
    '''Determine the state of this DSF.

    Given the dimensions of Q as p x p and the dimensions of P as p x m, the
    shape is (p, m).
    '''
    return self.p, self.m

  ############################################################################
  #   Conversion functions
  ############################################################################

  #---------------------------------------------------------------
  def to_DSF(self):
    '''Convert this ConvolutionalDSF to a DiscreteDSF/
```

```
    Returns
    -------
    F : DiscreteDSF
    '''
    from .DiscreteDSF import DTDSF

    Q = self.Q.to_TF()
    P = self.P.to_TF()

    return DTDSF(Q, P)

  #-----------------------------------------------------------------
  def to_impulse(self, r):
    '''Convert this ConvolutionalDSF into an ImpulseDSF.

    Returns
    -------
    F : ImpulseDSF (p x m)
    '''
    from .ImpulseDSF import ImpulseDSF

    Qimpulse = self.Q.to_impulse(r)
    Pimpulse = self.P.to_impulse(r)

    return ImpulseDSF(Qimpulse, Pimpulse)

  ######################################################################
  #   Printing and representation functions
  ######################################################################
  #-----------------------------------------------------------------
  def __repr__(self):
    # return repr(self.G)
    return 'Q = {}\nP = {}'.format(repr(self.Q), repr(self.P))

  #-----------------------------------------------------------------
  def latex(self, **kwargs):
    '''Return the latex formatted version of this DSF.

    Parameters
    ----------
    Qname : str (latex formatting)
        The name to give to the Q matrix.
    Pname : str (latex formatting)
        The name to give to the P matrix.

    Returns
    -------
    latex : str
    '''
    Qname = kwargs.get('Qname', 'Q')
    Pname = kwargs.get('Pname', 'P')
    return r'{} \qquad {}'.format(
      self.Q.latex(name=Qname), self.P.latex(name=Pname)
    )
```

Listing 7 pyrics.Representations.ConvolutionalScalar

```
#-----------------------------------------------------------------
# pyrics/Representations/ConvolutionalScalar.py
#
# Representation of a proper rational function in convolutional form.
#
# Copyright 2019 Nathan Woodbury
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
# http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
```

```
#-----------------------------------------------------------------

from sympy.abc import z

from .SystemRepresentation import SystemRepresentation
from .DiscreteTransferFunction import DTTF


class ConvolutionalScalar(SystemRepresentation):
  '''The convolutional representation of a linear impulse response; i.e.,

      f(t) = a delta(t, 0) + sum_{n = 1}^w b_n c_n^t

  for parameters a, b_n, and c_n, n = 1, ..., w.

  Parameters
  ----------
  params : list (lengh 2 * w + 1)
    [a, b_1, c_1, b_2, c_2, ..., b_w, c_w].
  tol : number > 0
    If either |b_i| < tol or |c_i| < tol, that parameter pair is removed from
    the list (this helps clean up the conversions).
  '''

  #-----------------------------------------------------------------
  def __init__(self, params, tol=1e-6):

    if len(params) % 2 != 1:
      raise ValueError(
        'params must have an odd number of entries. Contains {} entries'.format(
          len(params)
        )
      )

    rparams = [params[0]]
    rest = params[1:]
    for i in range(0, len(rest), 2):
      b = rest[i]
      c = rest[i + 1]
      if abs(b) < tol or abs(c) < tol:
        continue
      rparams += [b, c]

    params = rparams

    self.w = int((len(params) - 1) / 2)
    self.params = params

  ######################################################################
  #   Public functions
  ######################################################################

  def inf_norm(self, r=100):
    '''Compute the infinity norm of this convolutional transfer function.

    The infinity norm is the one norm of the corresponding impulse response.

    r : int > 0
      See parameter `r` in `to_impulse()`. The norm must convert to the finite
      impulse response before computing the norm. In the future, an option will
      be added to automatically determine r, but for now, it must predefined
      (defaults to 100).

    Returns
    -------
    norm : float >= 0
    '''
    imp = self.to_impulse(r=r)
    return imp.one_norm()

  ######################################################################
  #   Conversion functions
  ######################################################################

  #-----------------------------------------------------------------
  def to_TF(self):
    '''Converts this convolutional form into a SISO transfer function.

    Returns
    -------
    tf : DiscreteTransferFunction (1x1)
    '''
```

```
    # a = self.params[0]
    rest = self.params[1:]
    bc = [
      (rest[i], rest[i + 1]) for i in range(0, len(rest), 2)
    ]

    G = 0
    for bi, ci in bc:
      alpha = bi * ci
      beta = ci
      G += alpha / (z - beta)

    return DTTF([[G]])

  #-----------------------------------------------------------
  def to_impulse(self, r=100):
    '''Converts this convolutional form into an impulse response.

    Parameters
    ----------
    r : int > 0
      Creates an impulse response g(t) = [f(0), ..., f(r - 1)], where g(t) is
      the impulse response and f(t) is the convolutional form. Note that f(t)
      should be approximately zero for every t >= r; however, this condition is
      not checked.
      TODO - auto find a good r

    Returns
    -------
    ConvolutionalScalar
    '''
    from .ImpulseScalar import ImpulseScalar

    a = self.params[0]
    rest = self.params[1:]

    def _f(t):

      if t == 0:
        rs = a
      else:
        rs = 0

      for i in range(0, len(rest), 2):
        b = rest[i]
        c = rest[i + 1]
        rs += b * c ** t

      return rs

    g = []
    for t in range(r):
      g.append(_f(t))

    return ImpulseScalar(g)

  ###########################################################################
  #   Printing and representation functions
  ###########################################################################
  #-----------------------------------------------------------
  def __repr__(self):
    return repr(self.params)

  #-----------------------------------------------------------
  def latex(self, **kwargs):
    '''Returns the latex formatted version of this transfer function.

    kwargs
    ----------
    name : str or None, default = G
      Prefixes the latex with 'name = ' for whatever name is given. If
      name is None, just gives the equation.

    Returns
    -------
    latex : str
    '''
    name = kwargs.get('name', 'f(t)')

    # sp.init_printing(use_latex='mathjax')
    if name is not None:
      prefix = '{} = '.format(name)
```

```
    else:
      prefix = ''

    if len(self.params) == 1:
      if self.params[0] == 0:
        return '{} 0'.format(prefix)
      return '{} {:.3f}\\delta_{{(t, 0)}}'.format(prefix, self.params[0])

    a = self.params[0]
    rest = self.params[1:]
    bc = [
      '{:.3f}({:.3f})^t'.format(rest[i], rest[i + 1])
      for i in range(0, len(rest), 2)
    ]
    bcstr = ' + '.join(bc)

    return '{} {:.3f}\\delta_{{(t, 0)}} + {}'.format(prefix, a, bcstr)
```

Listing 8 pyrics.Representations.ConvolutionalTF

```
#-------------------------------------------------------------
# pyrics/Representations/ConvolutionalTF.py
#
# Representation of a transfer function in convolutional form.
#
# Copyright 2019 Nathan Woodbury
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
# http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
#-------------------------------------------------------------

from .SystemRepresentation import SystemRepresentation
from .ConvolutionalScalar import ConvolutionalScalar


class ConvolutionalTF(SystemRepresentation):
  '''Representation of a transfer function matrix in convolutional form.

  Parameters
  ----------
  G : list of lists of ImpulseScalar (p x m array)
  '''

  #-----------------------------------------------------------
  def __init__(self, G):
    self.G = G
    self.p = len(G)
    assert self.p > 0
    self.m = len(G[0])

    for inner in self.G:
      assert len(inner) == self.m
      for el in inner:
        assert isinstance(el, ConvolutionalScalar)

  #-----------------------------------------------------------
  @property
  def shape(self):
    '''Give the shape of this convulolutional TF, which is p x m.

    Returns
    -------
    shape : (int > 0, int > 0)
    '''
    return (self.p, self.m)
```

```
    ########################################################################
    #   Public functions
    ########################################################################

    def scalar_inf_norm(self, r=100):
      '''Compute a matrix of the same dimensions as this, where the entries
      are the infinity-norms of the corresponding convolutional scalars.

      Parameters
      ----------
      r : int > 0, default = 100
        An integer such that every finite impulse response in this matrix is
        approximately zero for all t >= r. See the documentation for
        `ConvolutionalScalar.inf_norm()` for more details.

      Returns
      -------
      M : np.array (p x m)
      '''
      return self.to_impulse(r=r).scalar_one_norm()

    ########################################################################
    #   Conversion functions
    ########################################################################

    #---------------------------------------------------------------
    def to_TF(self):
      '''Convert this ConvolutionalTF to a DTTF.

      Returns
      -------
      G : DTTF (p x m)
      '''
      from .DiscreteTransferFunction import DTTF

      G = []
      for i in range(self.p):
        G.append([])
        for j in range(self.m):
          G[i].append(self.G[i][j].to_TF())

      return DTTF(G)

    #---------------------------------------------------------------
    def to_impulse(self, r):
      '''Convert this ConvolutionalTF into an ImpulseTF.

      Returns
      -------
      G : ImpulseTF (p x m)
      '''
      from .ImpulseTF import ImpulseTF

      G = []
      for i in range(self.p):
        G.append([])
        for j in range(self.m):
          G[i].append(self.G[i][j].to_impulse(r=r))

      return ImpulseTF(G)

    ########################################################################
    #   Printing and representation functions
    ########################################################################

    #---------------------------------------------------------------
    def __repr__(self):
      return repr(self.G)

    #---------------------------------------------------------------
    def latex(self, **kwargs):
      '''Returns the latex formatted version of this transfer function.

      kwargs
      ----------
      name : str or None, default = G
        Prefixes the latex with 'name = ' for whatever name is given. If
        name is None, just gives the equation.

      Returns
      -------
      latex : str
```

```
      '''
      name = kwargs.get('name', 'G(t)')

      # sp.init_printing(use_latex='mathjax')
      if name is not None:
        prefix = '{} = '.format(name)
      else:
        prefix = ''

      matrixstr = '\\\\'.join(
        [' & '.join([el.latex(name=None) for el in inner]) for inner in self.G]
      )
      return '{} \\begin{{bmatrix}}{}\\end{{bmatrix}}'.format(prefix, matrixstr)

    ###########################################################################
    #   Magic Functions
    ###########################################################################
    #-------------------------------------------------------------
    def __getitem__(self, key):
      assert len(key) == 2
      return self.G[key[0]][key[1]]

    #-------------------------------------------------------------
    def __setitem__(self, key, value):
      assert len(key) == 2
      self.G[key[0]][key[1]] = value
```

Listing 9 pyrics.Representations.DiscreteDSF

```
#-------------------------------------------------------------
# pyrics/Representations/DiscreteDSF.py
#
# Representation of a discrete-time DSF.
#
# Copyright 2019 Nathan Woodbury
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
# http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
#-------------------------------------------------------------

# import warnings
from sympy import oo
from sympy.matrices import Matrix, eye
from numpy.linalg import matrix_rank as rank

from .DiscreteTime import DiscreteTime


class DTDSF(DiscreteTime):
  '''Representation of a discrete time dynamical structure function.

  Models

    Y(z) = Q(z)Y(z) + P(z)U(z)

  Parameters
  ----------
  Q : DTTF
  P : DTTF
  '''

  #-------------------------------------------------------------
  def __init__(self, Q, P):
    from . import DTTF

    if not isinstance(Q, DTTF):
      raise ValueError('Q must be a DTTF')
```

```
48        if not isinstance(P, DTTF):
49          raise ValueError('P must be a DTTF')
50
51        p, p1 = Q.shape
52        p2, m = P.shape
53
54        if p != p1:
55          raise ValueError(
56            'Dimension Mismatch: Q must be square. Got dimensions ({}, {})'.format(
57              p, p1
58            )
59          )
60
61        if p != p2:
62          raise ValueError(
63            'Dimension Mismatch: Q and P must have the same number of '
64            'rows. Got dimensions ({}, {}) and ({}, {})'.format(
65                p, p1, p2, m
66            )
67          )
68
69        self.Q = Q
70        self.P = P
71        self.p = p
72        self.m = m
73
74        self._W = None
75        self._V = None
76
77      ##########################################################################
78      #   Public Methods
79      ##########################################################################
80
81      #---------------------------------------------------------------
82      @property
83      def shape(self):
84        '''Determine the state of this DSF.
85
86        Given the dimensions of Q as p x p and the dimensions of P as p x m, the
87        shape is (p, m).
88        '''
89        return self.p, self.m
90
91      #---------------------------------------------------------------
92      def is_wellposed(self):
93        '''Determine whether this network is well-posed, according to the definition
94        in [1,2].
95
96        Returns
97        -------
98        iswp : bool
99          True if this network is well-posed (i.e., (I - Q) has a proper
100         inverse); False otherwise.
101
102       Sources
103       -------
104       [1] N. Woodbury, A. Dankers and S. Warnick, "On the Well-Posedness of
105           LTI Networks," Conference on Decision and Control, Melbourne,
106           Australia, 2017.
107       [2] N. Woodbury, A. Dankers and S. Warnick, "Dynamic Networks:
108           Representations, Abstractions, and Well-Posedness," Conference on
109           Decision and Control, Miami Beach, Florida, 2018.
110       '''
111       from . import DTTF
112
113       ImQ = DTTF(Matrix.eye(self.p)) - self.Q
114       ImQ_oo = ImQ.limit(oo)
115       return rank(ImQ_oo) == self.p
116
117     #---------------------------------------------------------------
118     def immerse(self, S):
119       '''Find the immersion of this DSF where only outputs in S (zero indexed)
120       are manifest.
121
122       TODO : when DiscreteDNF is added, separate implementation into a node
123       abstraction followed by an edge abstraction.
124
125       Attributes
126       ----------
127       (Only access these immediately after the abstract function is called,
```

```
    they are read only and are designed to show intermediate computation.
    For `F = DTDSF()`, access the attributes with `F._W` or `F._V`)
    _W : DTTF
      The intermediate W computed. Note that this would be the W in the DNF
      if only a node abstraction and not an edge abstraction is used.
    _V : DTTF
      The intermediate V computed. Note that this would be the V in the DNF
      if only a node abstraction and not an edge abstraction is used.

    Parameters
    ----------
    S : list
      A list of indices that remain manifest in the immersion (zero indexed).

    Returns
    -------
    FS : DTDSF
      The immersed DSF.
    '''
    from . import DTTF

    U = set(range(self.p))
    S = set(S)
    S &= U
    Sbar = U - S

    S = list(S)
    S.sort()
    Sbar = list(Sbar)
    Sbar.sort()

    Q = self.Q
    P = self.P

    Q11 = Q[S, S]
    Q12 = Q[S, Sbar]
    Q21 = Q[Sbar, S]
    Q22 = Q[Sbar, Sbar]

    P11 = P[S, :]
    P22 = P[Sbar, :]

    I11 = DTTF(eye(Q11.shape[0]))
    I22 = DTTF(eye(Q22.shape[0]))
    inner = Q12 * (I22 - Q22).inv()
    W = Q11 + inner * Q21
    V = P11 + inner * P22

    self._W = W
    self._V = V

    DW = W.diag()

    ImDW = (I11 - DW).inv()
    Q = ImDW * (W - DW)
    P = ImDW * V

    return DTDSF(Q, P)

  ###########################################################################
  #   Conversions
  ###########################################################################

  #---------------------------------------------------------------
  def to_TF(self):
    '''Convert this DSF into a transfer function.

    Returns
    -------
    G : DTTF
    '''
    from . import DTTF

    Iq = DTTF(eye(self.Q.shape[0]))
    G = (Iq - self.Q).inv() * self.P
    return G

  #---------------------------------------------------------------
  def to_convolutional(self):
    '''Converts this DSF into a convolutional DSF.

    Returns
    -------
    ConvolutionalDSF
    '''
```

```
      from . import ConvolutionalDSF

      Qconv = self.Q.to_convolutional()
      Pconv = self.P.to_convolutional()
      return ConvolutionalDSF(Qconv, Pconv)

    ########################################################################
    #   Printing and representation functions
    ########################################################################

    #-----------------------------------------------------------------
    def __repr__(self):
      # return repr(self.G)
      return 'Q = {}\nP = {}'.format(repr(self.Q), repr(self.P))

    #-----------------------------------------------------------------
    def latex(self, **kwargs):
      '''Return the latex formatted version of this DSF.

      Parameters
      ----------
      Qname : str (latex formatting)
          The name to give to the Q matrix.
      Pname : str (latex formatting)
          The name to give to the P matrix.

      Returns
      -------
      latex : str
      '''
      Qname = kwargs.get('Qname', 'Q')
      Pname = kwargs.get('Pname', 'P')
      return r'{} \qquad {}'.format(
        self.Q.latex(name=Qname), self.P.latex(name=Pname)
      )
```

Listing 10 pyrics.Representations.DiscreteGeneralizedStateSpace

```
#-----------------------------------------------------------------
# pyrics/Representations/DiscreteGeneralizedStateSpace.py
#
# Representation of a discrete-time genearlized state space model.
#
# Copyright 2019 Nathan Woodbury
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
# http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
#-----------------------------------------------------------------

import numpy as np
from sympy import Matrix, eye, latex
from sympy.abc import z

from .DiscreteTime import DiscreteTime


class DTGSS(DiscreteTime):
  '''A discrete time generalized state space model in intricacy-observed form,
  where dynamics are given by:

    x[k + 1] = Ax[k] + Ahat w[k] + Bu[k]
    y[k] = w[k] = Abar x[k] + Atilde w[k] + Bbar u[k]

  where k is a time index, x[k] is in R^n, u[k] is in R^m, and y[k]
  is in R^p.

  Parameters
```

```
    ----------
    A      : numpy.array (n x n)
    Ahat   : numpy.array (n x p)
    B      : numpy.array (n x m)
    Abar   : numpy.array (p x n)
    Atilde : numpy.array (p x p)
    Bbar   : numpy.array (p x m)
    '''

    #---------------------------------------------------------------
    def __init__(self, A, Ahat, B, Abar, Atilde, Bbar):
      A = np.array(A)
      Ahat = np.array(Ahat)
      B = np.array(B)
      Abar = np.array(Abar)
      Atilde = np.array(Atilde)
      Bbar = np.array(Bbar)

      # Compute Dimensions
      n, n1 = A.shape
      n3, p1 = Ahat.shape
      n2, m = B.shape
      p, n4 = Abar.shape
      p2, p3 = Atilde.shape
      p4, m1 = Bbar.shape

      # Check Dimensions
      if n1 != n:
        raise ValueError(
          'A must be square. Given A as ({} x {})'.format(n, n1)
        )
      if n2 != n:
        raise ValueError((
          'B must have the same number of rows as A. Given A as ' # pylint: disable=W1308
          '({} x {}) but B as ({} x {})'
        ).format(n, n, n2, m))
      if n3 != n:
        raise ValueError((
          'Ahat must have the same number of rows as A. Given A as ' # pylint: disable=W1308
          '({} x {}) but Ahat as ({}, {})'
        ).format(n, n, n3, p1))
      if p1 != p:
        raise ValueError((
          'The number of columns of Ahat must equal the number of rows of Abar. '
          'Given Abar as ({} x {}) but Ahat as ({}, {})'
        ).format(p, n4, n3, p1))
      if n4 != n:
        raise ValueError((
          'Abar must have the same number of columns as A. Given A as ' # pylint: disable=W1308
          '({} x {}) but Abar as ({} x {})'
        ).format(n, n, p, n4))
      if p2 != p1:
        raise ValueError((
          'Atilde must have the same number of rows as Abar. Given Abar as '
          '({} x {}) but Atilde as ({} x {})'
        ).format(p, n4, p2, p3))
      if p3 != p1:
        raise ValueError((
          'Atilde must be square. Given Atilde as ({} x {})'
        ).format(p2, p3))
      if p4 != p:
        raise ValueError((
          'Bbar must have the same number of rows as Abar. Given Abar as '
          '({} x {}) but Bbar as ({} x {})'
        ).format(p, n4, p4, m1))
      if m1 != m:
        raise ValueError((
          'Bbar must have the same number of columns as B. Given B as'
          '({} x {}) but Bbar as ({} x {})'
        ).format(n2, m, p4, m1))

      # Register Matrices and Dimensions
      self.A = A
```

```
      self.Ahat = Ahat
      self.B = B
      self.Abar = Abar
      self.Atilde = Atilde
      self.Bbar = Bbar

      self.n = n
      self.m = m
      self.p = p

    ##########################################################################
    #    Public Methods
    ##########################################################################

    #--------------------------------------------------------------
    def simulate(self, Du, x0=None):
      '''Simulate the system with the inputs in Du.

      Parameters
      ----------
      Du : numpy.array (T x m)
        The inputs where each row k is u(k)^T.
      x0 : numpy.array (n x 1)
        The initial conditions. NOT YET IMPLEMENTED.

      Returns
      -------
      Dy : numpy.array (T x p)
        The simulated outputs given the inputs and initial conditions, where
        each row k is y(k)^T.
      '''
      assert x0 is None  # TODO

      A = self.A
      Ahat = self.Ahat
      B = self.B
      Abar = self.Abar
      Atilde = self.Atilde
      Bbar = self.Bbar
      ImAtildeInv = np.linalg.inv(np.eye(Atilde.shape[0]) - Atilde)

      if x0 is None:
        x0 = np.zeros((self.n, 1))

      T, m = Du.shape
      assert m == self.m
      Dy = np.zeros((T, self.p))

      x = x0
      x = x.reshape((self.n, 1))
      for i in range(T):
        u = Du[i, :]
        u = u.reshape((self.m, 1))

        y = ImAtildeInv.dot(Abar.dot(x) + Bbar.dot(u))
        y = y.reshape((self.p, 1))
        x = A.dot(x) + Ahat.dot(y) + B.dot(u)
        x = x.reshape((self.n, 1))
        Dy[i, :] = y.T

      return Dy

    #--------------------------------------------------------------
    def is_stable(self, include_marginally=False):
      '''Determine whether this system is stable.

      This system is considered to be stable if its corresponding state space
      model is stable.

      Parameters
      ----------
      include_marginally : bool, default=True
        If True, marginally stable systems are considered to be stable,
        otherwise, marginally stable systems are considered to be unstable.

      Returns
      -------
      is_stable : bool
      '''
      ss = self.to_SS()
      return ss.is_stable(include_marginally)

    ##########################################################################
    #    Conversions
```

```
    ######################################################################
    #--------------------------------------------------------------------
    def to_SS(self):
      '''Convert this generalized state space model to a state space model.

      The state space model is found by solving for Atilde in the second equation
      and plugging in the result in to the first equation.

      Returns
      -------
      ss : DTSS
      '''
      from . import DTSS

      Ip = np.eye(self.p)
      inv = np.linalg.inv(Ip - self.Atilde)

      C = inv.dot(self.Abar)
      D = inv.dot(self.Bbar)
      A = self.A + self.Ahat.dot(C)
      B = self.B + self.Ahat.dot(D)

      return DTSS(A, B, C, D)

    #--------------------------------------------------------------------
    def to_TF(self):
      '''Converts this GSS model into a transfer function matrix.

      Returns
      -------
      G : DTTF
        The discrete time transfer function matrix representation of this system.
      '''
      from . import DTTF

      A = Matrix(self.A)
      Ahat = Matrix(self.Ahat)
      B = Matrix(self.B)
      Abar = Matrix(self.Abar)
      Atilde = Matrix(self.Atilde)
      Bbar = Matrix(self.Bbar)

      In = eye(self.n)
      Ip = eye(self.p)
      L = Atilde + Abar * (z * In - A).inv() * Ahat
      R = Bbar + Abar * (z * In - A).inv() * B
      G = (Ip - L).inv() * R
      return DTTF(G)

    #--------------------------------------------------------------------
    def to_DSF(self, factor=True):
      '''Convert this state space model into a DSF.

      TODO : Generalize
        FOR NOW ASSUMES THAT Bbar = [I 0]

      Parameters
      ----------
      factor : bool, default=True
        If True, simplifies and factors all rational polynomials (coefficients
        should be rational). Otherwise only simplifies.

      Returns
      -------
      QP : DTDSF
          The discrete time DSF representation of this system.
      '''
      from . import DTTF, DTDSF

      A = Matrix(self.A)
      Ahat = Matrix(self.Ahat)
      B = Matrix(self.B)
      # Abar = Matrix(self.Abar)
      Atilde = Matrix(self.Atilde)
      Bbar = Matrix(self.Bbar)

      p = self.p
      n = self.n
      l = n - p

      A11 = Matrix(A[:p, :p])
```

```
    if p < n:
      A12 = Matrix(A[:p, p:n])
      A21 = Matrix(A[p:n, :p])
      A22 = Matrix(A[p:n, p:n])
    else:
      A12 = Matrix([])
      A21 = Matrix([])
      A22 = Matrix([])

    Ahat1 = Ahat[:p, :]
    Ahat2 = Ahat[p:, :]
    B1 = B[:p, :]
    B2 = B[p:, :]

    I1 = eye(l)
    Wtilde = A11 + A12 * (z * I1 - A22).inv() * A21
    Rtilde = Ahat1 + A12 * (z * I1 - A22).inv() * Ahat2
    Vtilde = B1 + A12 * (z * I1 - A22).inv() * B2

    What = 1 / z * Wtilde
    Rhat = 1 / z * Rtilde
    Vhat = 1 / z * Vtilde
    # DTTF(What).display(name=r'\hat{W}')

    # Compute the final DNF
    # TODO : create DNF class and convert to DSF using edge abstraction
    Ip = eye(p)
    W = What + Rhat + (Ip - What) * Atilde
    # DTTF(W).display(name='W')
    V = Vhat + (Ip - What) * Bbar

    # Convert to DSF
    Wdiag = W * 0
    for i in range(p):
      Wdiag[i, i] = W[i, i]

    # DTTF((Ip - Wdiag).inv()).display(name='D_W')

    Q = (Ip - Wdiag).inv() * (W - Wdiag)
    P = (Ip - Wdiag).inv() * V

    if factor:
      pass  # TODO use or remove

    return DTDSF(DTTF(Q), DTTF(P))

  ##########################################################################
  #   Printing and representation functions
  ##########################################################################
  #--------------------------------------------------------------
  def __repr__(self):
    # return repr(self.G)
    return r'A = {}\nAhat = {}\nB={}\nAbar={}\nAtilde={}\nBbar={}'.format(
      repr(self.A), repr(self.Ahat), repr(self.B), repr(self.Abar),
      repr(self.Atilde), repr(self.Bbar)
    )

  #--------------------------------------------------------------
  def latex(self, **kwargs):
    '''Returns the latex formatted version of this state space model.

    kwargs
    ------
    Mname : str
      The name to give to matrix `M`, where M is any of A, Ahat, B, or Abar,
      Atilde, or Bbar.

    Returns
    -------
    latex : str
    '''
    Aname = kwargs.get('Aname', 'A')
    Ahatname = kwargs.get('Ahatname', r'\hat{A}')
    Bname = kwargs.get('Bname', 'B')
    Abarname = kwargs.get('Abarname', r'\bar{A}')
    Atildename = kwargs.get('Atildename', r'\tilde{A}')
```

```
      Bbarname = kwargs.get('Bbarname', r'\bar{B}')

      return (
        r'{} = {} \qquad {} = {} \qquad {} = {}\\\\{} = {} \qquad {} = {}'
        r'\qquad {} = {}'
      ).format(
        Aname, latex(Matrix(self.A)),
        Ahatname, latex(Matrix(self.Ahat)),
        Bname, latex(Matrix(self.B)),
        Abarname, latex(Matrix(self.Abar)),
        Atildename, latex(Matrix(self.Atilde)),
        Bbarname, latex(Matrix(self.Bbar))
      )
```

Listing 11 pyrics.Representations.DiscreteStateSpace

```
#-----------------------------------------------------------
# pyrics/Representations/DiscreteStateSpace.py
#
# Representation of a discrete-time state space model.
#
# Copyright 2019 Nathan Woodbury
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
# http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
#-----------------------------------------------------------

from operator import gt, ge
import numpy as np
from sympy import latex
from sympy.matrices import Matrix, eye
from sympy.abc import z

from .DiscreteTime import DiscreteTime


class DTSS(DiscreteTime):
  '''A discrete time state space model, where dynamics are given by:

    x[k + 1] = Ax[k] + Bu[k]
    y[k] = Cx[k] + Du[k]

  where k is a time index, x[k] is in R^n, u[k] is in R^m, and y[k]
  is in R^p.

  Parameters
  ----------
  A : numpy.array (n x n)
  B : numpy.array (n x m)
  C : numpy.array (p x n)
  D : numpy.array (p x m) or None, default=None
    If not given, will be initialized to an array of zeros.
  '''

  #---------------------------------------------------------
  def __init__(self, A, B, C, D=None):
    A = np.array(A)
    B = np.array(B)
    C = np.array(C)

    # Compute Dimensions
    n, n1 = A.shape
    n2, m = B.shape
```

```
57        p, n3 = C.shape
58
59        # Initialize D if needed
60        if D is None:
61          D = np.zeros((p, m))
62        else:
63          D = np.array(D)
64
65        p1, m1 = D.shape
66
67        # Check Dimensions
68        if n1 != n:
69          raise ValueError(
70            'A must be square. Given A as ({} x {})'.format(n, n1)
71          )
72        if n2 != n:
73          raise ValueError((
74            'B must have the same number of rows as A. Given A as ' # pylint: disable=W1308
75            '({} x {}) but B as ({} x {})'
76          ).format(n, n, n2, m))
77        if n3 != n:
78          raise ValueError((
79            'C must have the same number of columns as A. Given A as ' # pylint: disable=W1308
80            '({} x {}) but C as ({} x {})'
81          ).format(n, n, p, n3))
82        if p1 != p:
83          raise ValueError((
84            'D must have the same number of rows as C. Given C as '
85            '({} x {}) but D as ({} x {})'
86          ).format(p, n3, p1, m1))
87        if m1 != m:
88          raise ValueError((
89            'D must have the same number of columns as B. Given B as '
90            '({} x {}) but D as ({} x {})'
91          ).format(n2, m, p1, m1))
92
93        # Register Matrices and Dimensions
94        self.A = A
95        self.B = B
96        self.C = C
97        self.D = D
98
99        self.n = n
100       self.m = m
101       self.p = p
102
103   ###########################################################################
104   #   Public Methods
105   ###########################################################################
106
107   #-----------------------------------------------------------------
108   def simulate(self, Du, x0=None):
109     '''Simulate the system with the inputs in Du.
110
111     Parameters
112     ----------
113     Du : numpy.array (T x m)
114       The inputs where each row k is u(k)^T.
115     x0 : numpy.array (n x 1)
116       The initial conditions. NOT YET IMPLEMENTED.
117
118     Returns
119     -------
120     Dy : numpy.array (T x p)
121       The simulated outputs given the inputs and initial conditions, where
122       each row k is y(k)^T.
123     '''
124     assert x0 is None  # TODO
125
126     A = self.A
127     B = self.B
128     C = self.C
129     D = self.D
130
131     if x0 is None:
132       x0 = np.zeros((self.n, 1))
133
134     T, m = Du.shape
135     assert m == self.m
136     Dy = np.zeros((T, self.p))
137
138     x = x0
```

```
      x = x.reshape((self.n, 1))
      for i in range(T):
        u = Du[i, :]
        u = u.reshape((self.m, 1))
        y = C.dot(x) + D.dot(u)
        y = y.reshape((self.p, 1))
        x = A.dot(x) + B.dot(u)
        x = x.reshape((self.n, 1))
        Dy[i, :] = y.T

      return Dy

    #--------------------------------------------------------
    def is_stable(self, include_marginally=False):
      '''Determine whether this system is stable.

      The conditions of stability are as follows:
        - If the magnitude of *all* eigenvalues of A are strictly less than
          one, the system is (asymtotically) stable.
        - If the magnitude of *all* eigenvalues of A are less than or equal
          to one, and at least one eigen value has magnitude equal to one,
          then the system is marginally stable.
        - If the magnitude of *any* eigenvalue is strictly greater than
          one, then the system is unstable.

      Examples
      --------
      Check to see if the DTSS system `ss` is asymptotically stable:

        ss.is_stable() == True

      Check to see if the DTSS system `ss` is either asymptotically stable
      or marginally stable:

        ss.is_stable(include_marginally=True) == True

      Check to see if the DTSS system `ss` is marginally stable only:

        (ss.is_stable(True) and not ss.is_stable()) == True

      Parameters
      ----------
      include_marginally : bool, default=True
        If True, marginally stable systems are considered to be stable,
        otherwise, marginally stable systems are considered to be unstable.

      Returns
      -------
      is_stable : bool
      '''
      eigs, _ = np.linalg.eig(self.A)

      if include_marginally:
        comp = gt
      else:
        comp = ge

      for eig in eigs:
        if comp(abs(eig), 1):
          return False
      return True

    ###########################################################################
    #   Conversions
    ###########################################################################

    #--------------------------------------------------------
    def to_TF(self):
      '''Converts this state space model into a transfer function matrix.

      Returns
      -------
      G : DTTF
        The discrete time transfer function matrix representation of this system.
      '''
      from . import DTTF

      Asym = Matrix(self.A)
      Bsym = Matrix(self.B)
      Csym = Matrix(self.C)
      Dsym = Matrix(self.D)
```

```
    Isym = eye(self.n)
    G = Csym * (z * Isym - Asym).inv() * Bsym + Dsym

    return DTTF(G)

  #-------------------------------------------------------------
  def to_DSF(self, factor=True):
    '''Convert this state space model into a DSF.

    TODO : Generalize
      FOR NOW ASSUMES THAT C = [I 0]
    TODO : take advantage of factor

    Parameters
    ----------
    factor : bool, default=True
      If True, simplifies and factors all rational polynomials (coefficients
      should be rational). Otherwise only simplifies.

    Returns
    -------
    QP : DTDSF
        The discrete time DSF representation of this system.
    '''
    from . import DTTF, DTDSF

    A = self.A
    B = self.B
    C = self.C
    D = self.D

    p, _ = C.shape
    n, _ = A.shape
    # h, m = B.shape

    A11 = Matrix(A[:p, :p])
    if p < n:
      A12 = Matrix(A[:p, p:n])
      A21 = Matrix(A[p:n, :p])
      A22 = Matrix(A[p:n, p:n])
    else:
      A12 = Matrix([])
      A21 = Matrix([])
      A22 = Matrix([])

    B1 = B[:p, :]
    B2 = B[p:, :]

    # l, _ = A22.shape
    l = n - p
    Isym = eye(l)

    if p != n:
      W = A11 + A12 * (z * Isym - A22).inv() * A21
      V = B1 + A12 * (z * Isym - A22).inv() * B2
    else:
      W = A11
      V = B1

    Wdiag = W * 0
    for i in range(p):
      Wdiag[i, i] = W[i, i]

    I2 = eye(p)
    Wdinv = (z * I2 - Wdiag).inv()
    Q = Wdinv * (W - Wdiag)
    P = Wdinv * V + (I2 - Q) * Matrix(D)

    if factor:
      pass  # TODO use or remove

    return DTDSF(DTTF(Q), DTTF(P))

  ##########################################################################
  #   Printing and representation functions
  ##########################################################################

  #-------------------------------------------------------------
  def __repr__(self):
    # return repr(self.G)
```

```
301       return r'A = {}\nB = {}\nC={}\nD={}'.format(
302         repr(self.A), repr(self.B), repr(self.C), repr(self.D)
303       )
304
305     #---------------------------------------------------------------
306     def latex(self, **kwargs):
307       '''Returns the latex formatted version of this state space model.
308
309       kwargs
310       ------
311       Mname : str
312         The name to give to matrix `M`, where M is any of A, B, C, or D.
313
314       Returns
315       -------
316       latex : str
317       '''
318       Aname = kwargs.get('Aname', 'A')
319       Bname = kwargs.get('Bname', 'B')
320       Cname = kwargs.get('Cname', 'C')
321       Dname = kwargs.get('Dname', 'D')
322
323       return (
324         r'{} = {} \qquad {} = {}\\\\{} = {} \qquad {} = {}'
325       ).format(
326         Aname, latex(Matrix(self.A)), Bname, latex(Matrix(self.B)),
327         Cname, latex(Matrix(self.C)), Dname, latex(Matrix(self.D))
328       )
```

Listing 12 pyrics.Representations.DiscreteTime

```
1  #---------------------------------------------------------------
2  # pyrics/Representations/DiscreteTime.py
3  #
4  # Base representation for discrete-time models.
5  #
6  # Copyright 2019 Nathan Woodbury
7  #
8  # Licensed under the Apache License, Version 2.0 (the "License");
9  # you may not use this file except in compliance with the License.
10 # You may obtain a copy of the License at
11 #
12 # http://www.apache.org/licenses/LICENSE-2.0
13 #
14 # Unless required by applicable law or agreed to in writing, software
15 # distributed under the License is distributed on an "AS IS" BASIS,
16 # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
17 # See the License for the specific language governing permissions and
18 # limitations under the License.
19 #---------------------------------------------------------------
20
21 from .SystemRepresentation import SystemRepresentation
22
23
24 class DiscreteTime(SystemRepresentation):
25   '''Common functionality for all discrete time system representations.
26   '''
27
28   pass
```

Listing 13 pyrics.Representations.DiscreteTransferFunction

```
1  #---------------------------------------------------------------
2  # pyrics/Representations/DiscreteTransferFunction.py
3  #
4  # Representation of a discrete-time transfer function.
5  #
6  # Copyright 2019 Nathan Woodbury
7  #
8  # Licensed under the Apache License, Version 2.0 (the "License");
9  # you may not use this file except in compliance with the License.
```

```
10  # You may obtain a copy of the License at
11  #
12  # http://www.apache.org/licenses/LICENSE-2.0
13  #
14  # Unless required by applicable law or agreed to in writing, software
15  # distributed under the License is distributed on an "AS IS" BASIS,
16  # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
17  # See the License for the specific language governing permissions and
18  # limitations under the License.
19  #-----------------------------------------------------------
20
21  import warnings
22  from copy import copy, deepcopy
23
24  import numpy as np
25  # import sympy as sp
26  from sympy import (
27    oo, limit, latex, factor, nsimplify, apart, fraction, degree, Poly
28  )
29  from sympy.abc import z
30  from sympy.matrices import Matrix
31  from sympy.core.add import Add
32  from pyrics.PolynomialUtilities import (
33      is_proper, is_rational_polynomial
34  )
35
36  from pyrics.Algorithms import frequencyReconstruct
37  from .DiscreteTime import DiscreteTime
38
39
40  class DTTF(DiscreteTime):
41    '''Representation of a discrete time transfer function.
42
43    Parameters
44    ----------
45    G : sympy rational polynomial in z or sympy Matrix of rational polynomials.
46    '''
47
48    #-----------------------------------------------------------
49    def __init__(self, G):
50      if not isinstance(G, Matrix):
51        G = Matrix(G)
52      if len(G.shape) != 2:
53        raise ValueError(
54          'G must be a two-dimensional matrix'
55        )
56
57      # TODO : ensure that only symbol is z
58
59      self.p, self.m = G.shape
60      for i in range(self.p):
61        for j in range(self.m):
62
63          G[i, j] = nsimplify(G[i, j])
64          G[i, j] = factor(G[i, j])
65
66          if not is_rational_polynomial(G[i, j], z):
67            raise ValueError(
68              'G[{},{}] must be a rational polynomial in z, given {}'.format(
69                i, j, G[i, j]
70              )
71            )
72
73          if not is_proper(G[i, j], z):
74            warnings.warn(
75              'G[{},{}] = {} is not proper'.format(
76                i, j, G[i, j]
77              )
78            )
79      self.G = G
80
81    ##########################################################################
82    #   Public Properties
83    ##########################################################################
84
85    #-----------------------------------------------------------
86    @property
87    def shape(self):
```

```
    '''The dimensions of this transfer function.

    Returns
    -------
    m : int > 0
      The number of rows in the transfer function.
    n : int > 0
      The number of columns in the transfer function.
    '''
    return self.G.shape

  ##########################################################################
  #   Public Methods
  ##########################################################################

  #----------------------------------------------------------------
  def reconstruct(self, K=None):
    '''Reconstruct the network (DSF) generating this transfer function.

    Methodology described in [1].

    Parameters
    ----------
    K : np.array (p^2 + pm x pm) or None, default=None
      The identifiability conditions required to map the TF to the DSF uniquely.
      If None, assumes that P is diagonal (target specificity); however, an
      exception is raised in this case if p != m.

    Returns
    -------
    F : DTDSF
      The reconstructed network.

    Sources
    -------
    [1] J. Gonalves and S. Warnick, "Necessary and Sufficient Conditions
        for Dynamical Structure Reconstruction of LTI Networks," IEEE
        Transactions on Automatic Control, Aug. 2008.
    '''
    from . import DTDSF

    Q, P = frequencyReconstruct(self.G, K)
    return DTDSF(DTTF(Q), DTTF(P))

  #----------------------------------------------------------------
  def limit(self, approaches=oo):
    '''Compute the limit of this transfer function as z approaches `approaches`.

    Parameters
    ----------
    approaches : number or sympy.oo (infinity), default=infinity

    Returns
    -------
    limit : numpy.array (p x m)
    '''
    lim = np.zeros((self.p, self.m))

    for i in range(self.p):
      for j in range(self.m):
        lim[i, j] = limit(self.G[i, j], z, approaches)

    return lim

  #----------------------------------------------------------------
  def inv(self):
    '''Compute the inverse of this transfer function.

    Returns
    -------
    inv : DTTF
      The inverse.
    '''
    return DTTF(self.G.inv())

  #----------------------------------------------------------------
  def diag(self):
    '''Return a copy of this transfer function where every entry but the
    diagonals have been set to 0.

    NOTE: Only works on square transfer functions.

    Returns
    -------
    Gdiag : DTTF
```

```
      A TF with the same dimensions as this transfer function.
    '''
    m, n = self.shape
    if m != n:
      raise Exception('Can only find diag of square transfer functions')

    Gdiag = self.G * 0
    for i in range(m):
      Gdiag[i, i] = self.G[i, i]

    return DTTF(Gdiag)

  #------------------------------------------------------------
  def scalar_inf_norm(self, r=100):
    '''Compute a matrix of the same dimensions as this, where the entries
    are the infinity-norms of the corresponding transfer function scalars.

    Parameters
    ----------
    r : int > 0, default = 100
      An integer such that every finite impulse response in this matrix is
      approximately zero for all t >= r. See the documentation for
      `ConvolutionalScalar.inf_norm()` for more details.

    Returns
    -------
    M : np.array (p x m)
    '''
    return self.to_convolutional().to_impulse(r=r).scalar_one_norm()

  #------------------------------------------------------------
  @property
  def T(self):
    '''Compute and return the transpose of this DTTF.

    Returns:
    --------
    GT : DTTF (m x p)
    '''
    return DTTF(self.G.T)

  #------------------------------------------------------------
  def copy(self):
    return copy(self)

  #------------------------------------------------------------
  def deepcopy(self):
    return deepcopy(self)

  ##########################################################################
  #   Conversion functions
  ##########################################################################

  #------------------------------------------------------------
  def to_convolutional(self):
    '''Converts this DTTF to a convolutional form.

    Returns
    -------
    Gconv : ConvolutionalTF
    '''
    from .ConvolutionalScalar import ConvolutionalScalar
    from .ConvolutionalTF import ConvolutionalTF

    D = self.limit()

    Gconv = []
    for i in range(self.p):
      Gconv.append([])
      for j in range(self.m):
        tf = self.G[i, j]
        d = D[i, j]
        tf -= d
        ap = apart(tf)

        if isinstance(ap, Add):
          args = ap.args
        else:
          args = [ap]

        params = []
```

```
        bsum = 0

        for arg in args:
          if arg == 0:
            continue
          num, den = fraction(arg)
          assert degree(num) == 0
          assert degree(den) == 1
          alpha = float(num)
          scale, beta = Poly(den).coeffs()
          beta = -1 * float(beta)
          scale = float(scale)
          alpha /= scale
          beta /= scale
          b = alpha / beta
          c = beta
          params.append(b)
          params.append(c)
          bsum += b

        params = [-bsum + d] + params
        ci = ConvolutionalScalar(params)
        Gconv[i].append(ci)

    return ConvolutionalTF(Gconv)

  ###########################################################################
  #   Printing and representation functions
  ###########################################################################

  #---------------------------------------------------------------
  def __repr__(self):
    return repr(self.G)

  #---------------------------------------------------------------
  def latex(self, **kwargs):
    '''Returns the latex formatted version of this transfer function.

    kwargs
    ----------
    name : str or None, default = G
      Prefixes the latex with 'name = ' for whatever name is given. If
      name is None, just gives the matrix.

    Returns
    -------
    latex : str
    '''
    name = kwargs.get('name', 'G')

    # sp.init_printing(use_latex='mathjax')
    if name is not None:
      prefix = '{} = '.format(name)
    else:
      prefix = ''

    return '{}{}'.format(prefix, latex(self.G))

  ###########################################################################
  #   Magic Functions
  ###########################################################################

  #---------------------------------------------------------------
  def __neg__(self):
    return DTTF(-1 * self.G)

  #---------------------------------------------------------------
  def __add__(self, other):
    m, n = self.shape
    m1, n1 = self.shape

    if m != m1 or n != n1:
      raise ValueError(
        'Dimension mismatch, trying to add transfer functions of '
        'shapes ({}, {}) and ({}, {})'.format(
          m, n, m1, n1
        )
      )

    return DTTF(self.G + other.G)
```

```
334
335     #-------------------------------------------------------------
336     def __radd__(self, other):
337       return self.__add__(other)
338
339     #-------------------------------------------------------------
340     def __rmul__(self, other):
341       return other.__mul__(self)
342
343     #-------------------------------------------------------------
344     def __sub__(self, other):
345       return self.__add__(-other)
346
347     #-------------------------------------------------------------
348     def __mul__(self, other):
349       m, n = self.shape
350       m1, n1 = other.shape
351
352       if n != m1:
353         raise ValueError(
354           'Dimension mismatch, trying to multiply transfer functions of '
355           'shapes ({}, {}) and ({}, {})'.format(
356             m, n, m1, n1
357           )
358         )
359
360       return DTTF(self.G * other.G)
361
362     #-------------------------------------------------------------
363     def __pow__(self, n):
364       if not isinstance(n, int):
365         raise ValueError(
366           'Can only raise transfer functions to integer powers'
367         )
368
369       if n < 0:
370         G = self.G.inv()
371         n *= -1
372       else:
373         G = self.G
374
375       return DTTF(G ** n)
376
377     #-------------------------------------------------------------
378     def __getitem__(self, key):
379       return DTTF(self.G[key])
380
381     #-------------------------------------------------------------
382     def __setitem__(self, key, value):
383       self.G[key] = value
384
385     #-------------------------------------------------------------
386     def __copy__(self):
387       return DTTF(self.G)
388
389     #-------------------------------------------------------------
390     def __deepcopy__(self, memodict={}):  #pylint: disable=W0102
391       return DTTF(deepcopy(self.G))
```

Listing 14 pyrics.Representations.ImpulseDSF

```
1   #-------------------------------------------------------------
2   # pyrics/Representations/ImpulseDSF.py
3   #
4   # Representation of a DSF in impulse response form.
5   #
6   # Copyright 2019 Nathan Woodbury
7   #
8   # Licensed under the Apache License, Version 2.0 (the "License");
9   # you may not use this file except in compliance with the License.
10  # You may obtain a copy of the License at
11  #
12  # http://www.apache.org/licenses/LICENSE-2.0
13  #
14  # Unless required by applicable law or agreed to in writing, software
15  # distributed under the License is distributed on an "AS IS" BASIS,
16  # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
```

```
# See the License for the specific language governing permissions and
# limitations under the License.
#-----------------------------------------------------------

from .ImpulseTF import ImpulseTF
from .SystemRepresentation import SystemRepresentation

class ImpulseDSF(SystemRepresentation):
  '''Representation of a DSF where each entry is an impulse response.

  Models

      Y(t) = Q(t) * Y(t) + P(t) * U(t),

  where * is the convolution operation and each Q_ij(t), P_ij(t) is a finite
  impulse response of length r.

  Parameters
  ----------
  Q : ImpulseTF (p x p)
  P : ImpulseTF (p x m)
  '''

  #-----------------------------------------------------------
  def __init__(self, Q, P):
    self.Q = Q
    self.P = P

    if not isinstance(Q, ImpulseTF):
      raise ValueError('Q must be an ImpulseTF; got type={}'.format(type))
    if not isinstance(P, ImpulseTF):
      raise ValueError('P must be an ImpulseTF; got type={}'.format(type))

    p, p1 = Q.shape
    p2, m = P.shape
    r = Q.r
    r1 = P.r

    if p != p1:
      raise ValueError(
        'Dimension Mismatch: Q must be square. Got dimensions ({}, {})'.format(
          p, p1
        )
      )

    if p != p2:
      raise ValueError(
        'Dimension Mismatch: Q and P must have the same number of '
        'rows. Got dimensions ({}, {}) and ({}, {})'.format(
            p, p1, p2, m
        )
      )

    if r != r1:
      raise ValueError(
        'Length of impulse responses in Q and P must match. '
        'Got r={} in Q and r={} in P'.format(r, r1)
      )

    self.p = p
    self.m = m
    self.r = r

    # Reconstruction caches
    self.Ybar_cache = None
    self.Ubar_cache = None

  ############################################################################
  #   Public Methods
  ############################################################################

  #-----------------------------------------------------------
  @property
  def shape(self):
    '''Determine the state of this DSF.

    Given the dimensions of Q as p x p and the dimensions of P as p x m, the
    shape is (p, m).
    '''
    return self.p, self.m

  #-----------------------------------------------------------
```

```
    @staticmethod
    def reconstruct(Du, Dy, r, K=None, tol=0.001, precision=1e-6,
                    verbose=False):
      '''Reconstruct a ImpulseDSF from data.

      This is a wrapper around `passiveReconstruct()` from `pyrics.Algorithms`;
      please refer to the documentation of that function for all information
      about this method.
      '''
      from pyrics.Algorithms.TimeDomainReconstruction import timeReconstruct
      return timeReconstruct(
        Du, Dy, r, K=K, tol=tol, precision=precision, verbose=verbose,
      )

    ############################################################################
    #   Conversion functions
    ############################################################################

    def to_convolutional(self, order=3, bounds=10, verbose=False, tol=1e-3,
                         njobs=None):
      '''Convert this ImpulseTF into a ConvolutionalTF.

      Parameters
      ----------
      order : int > 0
        The number of poles (`w`) to add in the convolutional form, meaning the
        convolutional form will have `2 * order + 1` parameters.
      bounds : int > 0
        All parameters in the convolutional form will be bounded to be between
        -bounds and bounds.
      verbose : bool, default=False
        If true, prints status messages.
      tol : number > 0
        If the one-norm of any impulse response is less than tol, then 0 will
        be returned instead of attempting to convert to a convolutional scalar.
      njobs : int > 0 or None, default=None
        If not None, paralellizes the conversion of each scalar across a different
        job. Note that will run conversion of Q and P in sequence.

      Returns
      -------
      QP : ConvolutionalDSF
      '''
      from .ConvolutionalDSF import ConvolutionalDSF

      Q = self.Q.to_convolutional(
        order=order, bounds=bounds, verbose=verbose, tol=tol, njobs=njobs,
        name='Q'
      )
      P = self.P.to_convolutional(
        order=order, bounds=bounds, verbose=verbose, tol=tol, njobs=njobs,
        name='P'
      )

      return ConvolutionalDSF(Q, P)

    ############################################################################
    #   Printing and representation functions
    ############################################################################

    #---------------------------------------------------------------
    def __repr__(self):
      return 'Q = {}\nP = {}'.format(repr(self.Q), repr(self.P))

    #---------------------------------------------------------------
    def latex(self, **kwargs):
      '''Returns the latex formatted version of this transfer function.

      kwargs
      ----------
      name : str or None, default = G
        Prefixes the latex with 'name = ' for whatever name is given. If
        name is None, just gives the equation.
      length : int > 0, default=3
        Truncates the impulse response representation to the first `length` items.

      Returns
      -------
      latex : str
      '''
      Qname = kwargs.get('Qname', 'Q(t)')
```

```
      Pname = kwargs.get('Pname', 'P(t)')

      return r'{} \qquad {}'.format(
        self.Q.latex(name=Qname, **kwargs), self.P.latex(name=Pname, **kwargs)
      )

    #-----------------------------------------------------------------
    def get_plotly(self, Qname='Q(t)', Pname='P(t)', Qplot=None, Pplot=None,
                   Qtitle='', Ptitle='', scale=200, **kwargs):
      '''Generate and return plotly data for a time-series representation of
      this matrix of impulse responses. If `plot`, also draws a plotly figure.

      Parameters
      ----------
      Qname : str, default='Q(t)'
        The prefix to assign to each time series and plot title for the plot of Q.
      Pname : str, default='Q(t)'
        The prefix to assign to each time series and plot title for the plot of P.
      Qplot : function (plotly plot function) or None, default=None
        If None, only generates and returns the data. If a function is given,
        plots a single plot of the impulse responses in Q using that function.
      Pplot : function (plotly plot function) or None, default=None
        If None, only generates and returns the data. If a function is given,
        plots a single plot of the impulse responses in P using that function.
      Qtitle : str, default=''
        Only used if Qplot is not None. The title to assign to Q's plot.
      ptitle : str, default=''
        Only used if Pplot is not None. The title to assign to P's plot.
      scale : int > 0
        The dimensions of the figures will be p * scale x p * scale * 1.5

      kwargs
      ------
      Any additional parameter that can be used by `go.Scatter`. These will all be
      passed in to every scalar impulse responses plot function.

      Returns
      -------
      Qtraces : dict (tuple -> plotly.graph_objs)
        Maps (i, j) for the impulse response Q_ij(t) to the plot trace.
      Ptraces : dict (tuple -> plotly.graph_objs)
        Maps (i, j) for the impulse response P_ij(t) to the plot trace.
      '''
      Qtraces = self.Q.get_plotly(
        name=Qname, plot=Qplot, title=Qtitle, scale=scale, **kwargs
      )
      Ptraces = self.P.get_plotly(
        name=Pname, plot=Pplot, title=Ptitle, scale=scale, **kwargs
      )

      return Qtraces, Ptraces
```

Listing 15 pyrics.Representations.ImpulseScalar

```
#-----------------------------------------------------------------
# pyrics/Representations/ImpulseScalar.py
#
# Representation of a rational function in impulse response form.
#
# Copyright 2019 Nathan Woodbury
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
# http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
```

```
# limitations under the License.
#-----------------------------------------------------------------

import numpy as np
import plotly.graph_objs as go
from scipy.optimize import differential_evolution as evolution

from .SystemRepresentation import SystemRepresentation


class ImpulseScalar(SystemRepresentation):
  '''The representation of a linear finite impulse response; i.e.,

      f(t) = [f(0), f(1), f(2), ..., f(r - 1)]

  for some integer r > 0, and where G(t) = 0 for t < 0 and t >= r.

  Parameters
  ----------
  impulse : list (length r)
    The impulse response, where the item at index i is f(i).
  precision : number > 0
    f(t) such that |f(t)| < tol will be set to zero.
  '''
  #---------------------------------------------------------------
  def __init__(self, impulse, precision=1e-6):
    impulse = list(impulse)
    self.r = len(impulse)
    self.precision = precision

    self.impulse = []
    for ft in impulse:
      try:
        val = float(ft)
        if abs(val) < precision:
          val = 0
        self.impulse.append(val)
      except:
        raise ValueError('impulse must be a list of numbers')

  ############################################################################
  #   Public functions
  ############################################################################

  #---------------------------------------------------------------
  def one_norm(self):
    '''Get the one-norm of this impulse response.

    The one-norm is the sum of the absolute values of every entry in the impulse
    response.

    Returns
    -------
    norm : float >= 0
    '''
    return np.linalg.norm(self.impulse, ord=1)

  ############################################################################
  #   Conversion functions
  ############################################################################

  #---------------------------------------------------------------
  def to_convolutional(self, order=3, bounds=10, tol=1e-3, normord=1):
    '''Project this impulse response to the nearest convolutional form of order
    `order`.

    Parameters
    ----------
    order : int > 0
      The number of poles (`w`) to add in the convolutional form, meaning the
      convolutional form will have `2 * order + 1` parameters.
    bounds : int > 0
      Parameters a and b in the convolutional form will be bounded to be between
      -bounds and bounds. Parameter c will always be between -1 and 1
      (exclusive) so that the impulse response is stable.
    tol : number > 0
      If the `normord`-norm of this impulse response is less than tol, then a 0
      convolutional scalar will be returned without attempting to find a fit.
    normord : {int > 0, inf, -inf, 'fro', 'nuc'}, default=1
      The norm to use. Equivalent to the parameter `ord` in `np.linalg.norm()`.
```

```
    Returns
    -------
    ft : ConvolutionalScalar
    '''
    from .ConvolutionalScalar import ConvolutionalScalar

    if np.linalg.norm(self.impulse, ord=normord) < tol:
      return ConvolutionalScalar([0])

    def func(x):
      conv = ConvolutionalScalar(x)
      ximp = conv.to_impulse(r=self.r)
      dist = np.linalg.norm(
        np.array(self.impulse) - np.array(ximp.impulse), ord=normord
      )
      return dist

    boundslist = [(-bounds, bounds)]
    for _ in range(order):
      boundslist += [(-bounds, bounds), (-1 + 1e-6, 1 - 1e-6)]

    res = evolution(func, boundslist)
    return ConvolutionalScalar(res.x)

  ######################################################################
  #   Printing and representation functions
  ######################################################################

  #---------------------------------------------------------------
  def get_plotly(self, name='f(t)', plot=None, title='', **kwargs):
    '''Generate and return plotly data for a time-series representation of
    this impulse response. If `plot`, also draws a plotly figure.

    Parameters
    ----------
    name : str, default='f(t)'
      The name to assign to the time series. Useful if multiple series are added
      to the same plot, this gives each a label in the legend.
    plot : function (plotly plot function) or None, default=None
      If None, only generates and returns the data. If a function is given,
      plots a single plot of this impulse response using that function.
    title : str, default=''
      Only used if plot = True. The title to assign to the plot.

    kwargs
    ------
    Any additional parameter that can be used by `go.Scatter`. These will all be
    passed in to `go.Scatter`.

    Returns
    -------
    plotly.graph_objs
    '''
    trace = go.Scatter(
      x=list(range(self.r)),
      y=self.impulse,
      name=name,
      **kwargs
    )

    if plot is not None:
      layout = go.Layout(
        title=title,
        xaxis=dict(title='$r$'),
        yaxis=dict(title='${}$'.format(name))
      )
      plot({'data': [trace], 'layout': layout})

    return trace

  #---------------------------------------------------------------
  def __repr__(self):
    return repr(self.impulse)

  #---------------------------------------------------------------
  def latex(self, **kwargs):
    '''Returns the latex formatted version of this transfer function.
```

```
    kwargs
    ----------
    name : str or None, default = G
      Prefixes the latex with 'name = ' for whatever name is given. If
      name is None, just gives the equation.
    length : int > 0
      Truncates the impulse response representation to the first `length - 1`
      items, followed by '...', followed by the last item

    Returns
    -------
    latex : str
    '''
    name = kwargs.get('name', 'f(t)')

    # sp.init_printing(use_latex='mathjax')
    if name is not None:
      prefix = '{} = '.format(name)
    else:
      prefix = ''

    if 'length' in kwargs:
      impulse = self.impulse[:kwargs['length'] - 1]
    else:
      impulse = self.impulse

    impulse = ['{:.3f}'.format(fi) for fi in impulse]
    impulsestr = ' & '.join(impulse)

    if 'length' in kwargs:
      impulsestr += r' & \cdots & {:.3f}'.format(self.impulse[-1])

    return r'{}\begin{{bmatrix}}{}\end{{bmatrix}}'.format(prefix, impulsestr)
```

Listing 16 pyrics.Representations.ImpulseTF

```
#-------------------------------------------------------------
# pyrics/Representations/ImpulseTF.py
#
# Representation of a transfer function in impulse response form.
#
# Copyright 2019 Nathan Woodbury
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
# http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
#-------------------------------------------------------------

from functools import partial
from multiprocessing import Pool

import numpy as np
from plotly import tools

from pyrics.utilities import vprint as vprintfull
from .SystemRepresentation import SystemRepresentation
from .ImpulseScalar import ImpulseScalar


################################################################################
def _convolutionalConvert(job, verbose, order, bounds, tol, name):
  '''Utility used in converting to a convolutional form.

  Defined outside the class for paralellization.
  '''
  vprint = partial(vprintfull, verbose=verbose)

  i, j, scalar = job
```

```
    vprint('Converting {}[{}, {}]'.format(name, i + 1, j + 1))
    conv = scalar.to_convolutional(order=order, bounds=bounds, tol=tol)
    vprint('{}[{}, {}] finished'.format(name, i + 1, j + 1))
    return i, j, conv

##############################################################################
class ImpulseTF(SystemRepresentation):
    '''Representation of a transfer function matrix in impulse form.

    Parameters
    ----------
    G : list of lists of ImpulseScalar (p x m array)
    '''

    #---------------------------------------------------------------
    def __init__(self, G):
        self.G = G
        self.p = len(G)
        assert self.p > 0
        self.m = len(G[0])

        r = None
        rij = (-1, -1)
        for i, inner in enumerate(self.G):
            assert len(inner) == self.m
            for j, el in enumerate(inner):

                assert isinstance(el, ImpulseScalar)

                # Check that every r is the same
                # To consider: find the max r and replace every element with an
                #   augmented impulse response of that length.
                if r is None:
                    r = el.r
                    rij = (i, j)
                else:
                    if el.r != r:
                        raise ValueError((
                            'Every impulse response must have the same length `r`. We have '
                            'G[{}, {}].r = {} but G[{}, {}].r = {}'
                        ).format(rij[0], rij[i], r, i, j, el.r))

        self.r = r

    #---------------------------------------------------------------
    @property
    def shape(self):
        '''Give the shape of this convlolutional TF, which is p x m.

        Returns
        -------
        shape : (int > 0, int > 0)
        '''
        return (self.p, self.m)

    ############################################################################
    #   Public functions
    ############################################################################

    #---------------------------------------------------------------
    def scalar_one_norm(self):
        '''Compute a matrix of the same dimensions as this, where the entries
        are the one-norms of the corresponding impulse response.

        Returns
        -------
        M : np.array (p x m)
        '''
        M = np.zeros((self.p, self.m))
        for i in range(self.p):
            for j in range(self.m):
                M[i, j] = self[i, j].one_norm()

        return M

    ############################################################################
    #   Conversion functions
    ############################################################################

    def to_convolutional(self, order=3, bounds=10, verbose=False, tol=1e-3,
                         njobs=None, name='G'):
```

```
    '''Convert this ImpulseTF into a ConvolutionalTF.

    Parameters
    ----------
    order : int > 0
      The number of poles (`w`) to add in the convolutional form, meaning the
      convolutional form will have `2 * order + 1` parameters.
    bounds : int > 0
      All parameters in the convolutional form will be bounded to be between
      -bounds and bounds.
    verbose : bool, default=False
      If true, prints status messages.
    tol : number > 0
      If the one-norm of any impulse response is less than tol, then 0 will
      be returned instead of attempting to convert to a convolutional scalar.
    njobs : int > 0 or None, default=None
      If not None, paralellizes the conversion of each scalar across a different
      job.
    name : str, default='G'
      The name to display for this matrix in the status messages if verbose
      is True.

    Returns
    -------
    G : ConvolutionalTF
    '''
    from .ConvolutionalTF import ConvolutionalTF

    # Define the job function
    func = partial(
      _convolutionalConvert, verbose=verbose, order=order, bounds=bounds,
      tol=tol, name=name
    )

    # Build the list of jobs
    jobs = []
    for i in range(self.p):
      for j in range(self.m):
        jobs.append((i, j, self[i, j]))

    # Run the jobs
    if njobs is None:
      mapfunc = map
    else:
      p = Pool(njobs)
      mapfunc = p.map
    rs = mapfunc(func, jobs)

    # Build a placeholder for the results
    G = []
    for i in range(self.p):
      G.append([])
      for j in range(self.m):
        G[i].append(None)

    # Extract the results
    for rsi in rs:
      i, j, scalar = rsi
      G[i][j] = scalar

    return ConvolutionalTF(G)

  ###########################################################################
  #   Printing and representation functions
  ###########################################################################
  #-----------------------------------------------------------------
  def __repr__(self):
    return repr(self.G)

  #-----------------------------------------------------------------
  def latex(self, **kwargs):
    '''Returns the latex formatted version of this transfer function.

    kwargs
    ----------
    name : str or None, default = G
      Prefixes the latex with 'name = ' for whatever name is given. If
```

```
      name is None, just gives the equation.
    length : int > 0, default=3
      Truncates the impulse response representation to the first `length` items.

    Returns
    -------
    latex : str
    '''
    name = kwargs.get('name', 'G(t)')

    # sp.init_printing(use_latex='mathjax')
    if name is not None:
      prefix = '{} = '.format(name)
    else:
      prefix = ''

    matrixstr = '\\\\'.join(
      [' & '.join(
          [el.latex(name=None, length=kwargs.get('length', 3)) for el in inner]
        ) for inner in self.G
      ]
    )
    return '{} \\begin{{bmatrix}}{}\\end{{bmatrix}}'.format(prefix, matrixstr)

  #-----------------------------------------------------------------
  def get_plotly(self, name='f(t)', plot=None, title='', scale=200, **kwargs):
    '''Generate and return plotly data for a time-series representation of
    this matrix of impulse responses. If `plot`, also draws a plotly figure.

    Parameters
    ----------
    name : str, default='G(t)'
      The prefix to assign to each time series and plot title.
    plot : function (plotly plot function) or None, default=None
      If None, only generates and returns the data. If a function is given,
      plots a single plot of the impulse responses in G using that function.
    title : str, default=''
      Only used if plot is not None. The title to assign to the plot.
    scale : int > 0
      The dimensions of the figure will be p * scale x p * scale * 1.5

    kwargs
    ------
    Any additional parameter that can be used by `go.Scatter`. These will all be
    passed in to every scalar impulse responses plot function.

    Returns
    -------
    traces : dict (tuple -> plotly.graph_objs)
      Maps (i, j) for the impulse response G_ij(t) to the plot trace.
    '''
    traces = {}

    if max(self.p, self.m) < 10:
      comma = ''
    else:
      comma = ','

    subplot_titles = []
    for i in range(self.p):
      for j in range(self.m):
        if '(t)' in name:
          left, part, right = name.partition('(t)')
          label = r'${}_{{{}{}{}}}{}{}$'.format(
            left, i + 1, comma, j + 1, part, right
          )
        else:
          label = r'${}_{{{}{}{}}}$'.format(name, i + 1, comma, j + 1)

        trace = self[i, j].get_plotly(name='', **kwargs)
        traces[(i, j)] = trace
        subplot_titles.append(label)

    if plot is not None:
      fig = tools.make_subplots(
        rows=self.p, cols=self.m, subplot_titles=subplot_titles,
        print_grid=False
```

```
        )

        for ix, trace in traces.items():
          i = ix[0] + 1
          j = ix[1] + 1
          fig.append_trace(trace, i, j)

        fig['layout'].update(
          title=title,
          height=self.p * scale,
          width=self.m * scale * 1.5,
          showlegend=False
          # xaxis=dict(title='£r£'),
          # yaxis=dict(title='£{}£'.format(name))
        )
        plot(fig)

      return traces
    ##########################################################################
    #   Magic Functions
    ##########################################################################
    #-------------------------------------------------------------
    def __getitem__(self, key):
      assert len(key) == 2
      return self.G[key[0]][key[1]]

    #-------------------------------------------------------------
    def __setitem__(self, key, value):
      assert len(key) == 2
      self.G[key[0]][key[1]] = value
```

Listing 17 pyrics.Representations.SystemRepresentation

```
#-------------------------------------------------------------
# pyrics/Representations/SystemRepresnetation.py
#
# Base representation for all models.
#
# Copyright 2019 Nathan Woodbury
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
# http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
#-------------------------------------------------------------

from IPython.display import display, Math


class SystemRepresentation():
  '''Common Functionality for all representations.
  '''

  #-------------------------------------------------------------
  def display(self, **kwargs):
    '''Display the compiled latex representation within a
    jupyter/ipython/hydrogen environment.

    Parameters
    ----------
    **kwargs : key word arguments (optional)
      All arguments to pass in to the `latex()` method implemented by
      any child.
    '''
    return display(Math(self.latex(**kwargs)))
```

```
41    #-----------------------------------------------------------
42    def latex(self, **kwargs):
43      '''To be implemented in any child class. Key word arguments can and
44      should be named and specified within any implementation.
45      '''
46      raise NotImplementedError()
```

Listing 18 pyrics.PolynomialUtilities

```
1   #-----------------------------------------------------------
2   # pyrics/PolynomialUtilities.py
3   #
4   # Collection of miscellaneous utilities related to sympy polynomials.
5   #
6   # Copyright 2019 Nathan Woodbury
7   #
8   # Licensed under the Apache License, Version 2.0 (the "License");
9   # you may not use this file except in compliance with the License.
10  # You may obtain a copy of the License at
11  #
12  # http://www.apache.org/licenses/LICENSE-2.0
13  #
14  # Unless required by applicable law or agreed to in writing, software
15  # distributed under the License is distributed on an "AS IS" BASIS,
16  # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
17  # See the License for the specific language governing permissions and
18  # limitations under the License.
19  #-----------------------------------------------------------
20
21  from sympy import degree, diff, sympify, fraction
22
23  #-----------------------------------------------------------
24  def is_polynomial(expr, arg):
25    '''Determine whether `expr` is a polynomial in `arg`.
26
27    Parameters
28    ----------
29    expr : sympy expression (or something that can be cast as one)
30
31    Returns
32    -------
33    is_polynomial : bool
34    '''
35    expr = sympify(expr)
36    if expr.is_real:
37      # Base case: given a number
38      return True
39
40    d = degree(expr)
41    fin = expr
42    for _ in range(d):
43      fin = diff(fin, arg)
44
45    return fin.is_real
46
47  #-----------------------------------------------------------
48  def is_rational_polynomial(expr, arg):
49    '''Determine whether `expr` is a rational polynomial in `arg`.
50
51    Parameters
52    ----------
53    expr : sympy expression (or something that can be cast as one)
54
55    Returns
56    -------
57    is_rational_polynomial : bool
58    '''
59    expr = sympify(expr)
60    n, d = fraction(expr)
61    return is_polynomial(n, arg) and is_polynomial(d, arg)
62
63  #-----------------------------------------------------------
64  def is_proper(expr, arg):
65    '''Determine if the given expression is a proper polynomial in arg.
66
67    Parameters
```

```
    ----------
    expr : sympy expression
      If not a rational polynomial (see `is_rational_polynomial()`, raises
      `ValueError`)

    Returns
    -------
    is_proper : bool
    '''
    expr = sympify(expr)
    if not is_rational_polynomial(expr, arg):
      raise ValueError(
        'Can only check properness of rational polynomials.'
      )
    n, d = fraction(expr)
    return _degree(d) >= _degree(n)

  #-----------------------------------------------------------------
  def is_strictly_proper(expr, arg):
    '''Determine if the given expression is a strictly proper polynomial in arg.

    Parameters
    ----------
    expr : sympy expression\linespread{0.2}
      If not a rational polynomial (see `is_rational_polynomial()`, raises
      `ValueError`)

    Returns
    -------
    is_proper : bool
    '''
    expr = sympify(expr)
    if not is_rational_polynomial(expr, arg):
      raise ValueError(
        'Can only check properness of rational polynomials.'
      )
    if expr == 0:
      # Special case, 0 is considered to be strictly proper
      return True
    n, d = fraction(expr)
    return _degree(d) > _degree(n)

  #-----------------------------------------------------------------
  def _degree(p):
    '''Wrapper around sympy's `degree()`, setting the degree of numbers to be
    0.,,
    '''
    if p.is_real:
      return 0
    return degree(p)
```

Listing 19 pyrics.utilities

```
  #-----------------------------------------------------------------
  # pyrics/utilities.py
  #
  # A collection of miscellaneous utilities used within the framework.
  #
  # Copyright 2019 Nathan Woodbury
  #
  # Licensed under the Apache License, Version 2.0 (the "License");
  # you may not use this file except in compliance with the License.
  # You may obtain a copy of the License at
  #
  # http://www.apache.org/licenses/LICENSE-2.0
  #
  # Unless required by applicable law or agreed to in writing, software
  # distributed under the License is distributed on an "AS IS" BASIS,
  # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
  # See the License for the specific language governing permissions and
  # limitations under the License.
  #-----------------------------------------------------------------
```

```
21  from datetime import datetime
22  from sympy import latex
23  from IPython.display import display, Math
24
25  #-------------------------------------------------------------
26  def list_diff(left, right):
27    '''Treat each list as a set and return the set difference left - right.
28
29    Parameters
30    ----------
31    left : list
32    right : list
33
34    Returns
35    -------
36    diff : list
37    '''
38    return list(set(left) - set(right))
39
40  #-------------------------------------------------------------
41  def pprint(m):
42    '''Return a pretty (latex) formatted version of the math in m for use in
43    a Jupyter notebook.
44    '''
45    return display(Math(latex(m)))
46
47  #-------------------------------------------------------------
48  def vprint(msg, verbose):
49    '''A verbose printing utility.
50
51    If verbose, displays "[time] : [msg]"
52
53    Parameters
54    ----------
55    msg : str
56      The message to print
57    verbose : bool
58      If False, don't print anything.
59    '''
60    if verbose:
61      print(datetime.now(), ':', msg)
```

References

[1] V. Chetty and S. Warnick, "Network semantics of dynamical systems," in *IEEE Conference on Decision and Control (CDC)*. Osaka, Japan: IEEE, 2015.

[2] V. Chetty, "Theory and applications of network structure of complex dynamical systems," Ph.D Dissertation, Brigham Young University, Provo, UT, 2017.

[3] N. Woodbury and S. Warnick, "Abstractions and realizations of dynamic networks," 2019, to appear in the 2019 IEEE American Control Conference (ACC), Philadelphia, PA.

[4] C. Ward, E. Yeung, T. Brown, B. Durtschi, S. Weyerman, R. Howes, J. Goncalves, H. Sandberg, and S. Warnick, "A comparison of network reconstruction methods for chemical reaction networks," in *Foundations of Systems Biology and Engineering (FOSBE)*. Denver, CO: IFAC, 2009, pp. 197–200.

[5] A. Rai and S. Warnick, "A technique for designing stabilizing distributed controllers with arbitrary signal structure constraints," in *IEEE European Control Conference (ECC)*. Zürich, Switzerland: IEEE, 2013, pp. 3282–3287.

[6] D. Grimsman, V. Chetty, N. Woodbury, E. Vaziripour, S. Roy, D. Zappala, and S. Warnick, "A case study of a systematic attack design method for critical infrastructure cyber-physical systems," in *IEEE American Control Conference (ACC)*. Boston, MA: IEEE, 2016, pp. 296–301.

[7] D. Materassi and G. Innocenti, "Topological identification in networks of dynamical systems," *IEEE Transactions on Automatic Control*, vol. 55, no. 8, pp. 1860–1871, 2010.

[8] H. Weerts, P. M. Van den Hof, and A. Dankers, "Identification of dynamic networks operating in the presence of algebraic loops," in *IEEE Conference on Decision and Control (CDC)*. Las Vegas, NV: IEEE, 2016, pp. 4606–4611.

[9] A. Rai, D. Ward, S. Roy, and S. Warnick, "Vulnerable links and secure architectures in the stabilization of networks of controlled dynamical systems," in *IEEE American Control Conference (ACC)*. Montral, Canada: IEEE, 2012, pp. 1248–1253.

[10] N. Woodbury, "Vulnerability analysis of closed-loop systems," Undergraduate Honors Thesis, Brigham Young University, Provo, UT, 2013.

[11] V. Chetty, N. Woodbury, E. Vaziripour, and S. Warnick, "Vulnerability analysis for distributed and coordinated destabilization attacks," in *IEEE Conference on Decision and Control (CDC)*. Los Angeles, CA: IEEE, 2014.

[12] N. Woodbury, "Network reconstruction and vulnerability analysis of financial networks," M.S. Thesis, Brigham Young University, Provo, UT, 2017.

[13] A. Vosughi, C. Johnson, S. Roy, S. Warnick, and M. Xue, "Local control and estimation performance in dynamical networks: Structural and graph-theoretic results," in *IEEE Conference on Decision and Control (CDC)*. Melbourne, Australia: IEEE, 2017.

[14] N. Woodbury, A. Dankers, and S. Warnick, "On the well-posedness of LTI networks," in *IEEE Conference on Decision and Control (CDC)*. Melbourne, Australia: IEEE, 2017, pp. 4813–4818.

[15] ——, "Dynamic networks: Representations, abstractions, and well-posedness," in *IEEE Conference Decision and Control (CDC)*. Miami, FL: IEEE, 2018.

[16] ——, "On the well-posedness of LTI dynamic networks," 2018, submitted for Publication to the IEEE Transactions on Automatic Control.

[17] C. Johnson, N. Woodbury, and S. Warnick, "Graph theoretic foundations of linear dynamic networks," 2019, submitted for Publication to the IEEE Conference on Decision and Control (2019).

[18] V. Chetty, N. Woodbury, J. Brewer, K. Lee, and S. Warnick, "Applying a passive network reconstruction technique to Twitter data in order to identify trend setters," in *IEEE Conference on Control Technology and Applications (CCTA)*. Kohala Coast, HI: IEEE, 2017, pp. 1344–1349.

[19] J. Gonçalves and S. Warnick, "Necessary and sufficient conditions for dynamical structure reconstruction of LTI networks," *IEEE Transactions on Automatic Control*, vol. 53, no. 7, pp. 1670–1674, 2008.

[20] V. Chetty, J. Eliason, and S. Warnick, "Passive reconstruction of non-target-specific discrete-time LTI systems," in *IEEE American Control Conference (ACC)*. Boston, MA: IEEE, 2016, pp. 66–71.

[21] J. C. Willems, *The Analysis of Feedback Systems*. The MIT Press, 1971.

[22] K. Zhou, J. C. Doyle, and K. Glover, *Robust and Optimal Control*. Englewood Cliffs, New Jersey: Prentice Hall, 1996.

[23] R. D'Andrea and G. E. Dullerud, "Distributed control design for spacially interconnected systems," *IEEE Transactions on Automatic Control*, vol. 48, no. 9, pp. 1478–1495, Sep. 2003.

[24] G. Zames, "Realizability conditions for nonlinear feedback systems," *IEEE Transactions on Circuit Theory*, vol. 11, pp. 186–194, 1964.

[25] M. Vidyasagar, "On the well-posedness of large-scale interconnected systems," *IEEE Transactions on Automatic Control*, vol. 25, no. 3, pp. 413–421, 1980.

[26] J. Etesami and N. Kiyavash, "Directed information graphs: A generalization of linear dynamical graphs," in *IEEE American Control Conference (ACC), Portland, OR*. IEEE, 2014, pp. 2563–2568.

[27] G. E. Dullerud and F. Paganini, *A course in robust control theory: a convex approach*. Springer Science & Business Media, 2013, vol. 36.

[28] H. H. Weerts, P. M. Van den Hof, and A. G. Dankers, "Identifiability of linear dynamic networks," *Automatica*, vol. 89, pp. 247–258, 2018.

[29] F. Sepehr and D. Materassi, "Inferring the structure of polytree networks of dynamic systems with hidden nodes," in *IEEE Conference on Decision and Control (CDC)*. Las Vegas, NV: IEEE, 2016, pp. 4618–4623.

[30] E. Yeung, J. Gonçalves, H. Sandberg, and S. Warnick, "Representing structure in linear interconnected dynamical systems," in *IEEE Conference on Decision and Control (CDC)*. Atlanta, GA: IEEE, 2010, pp. 6010–6015.

[31] ——, "The meaning of structure in interconnected dynamic systems," *arXiv preprint arXiv:1108.2755*, 2011.

[32] M. Araki and M. Saeki, "A quantitative condition for the well-posedness of interconnected dynamical systems," *IEEE Transactions on Automatic Control*, vol. 28, no. 5, pp. 569–577, 1983.

[33] C. Scherer, "Theory of robust control," *Delft University of Technology*, 2001.

[34] A. Dankers, P. M. Van den Hof, X. Bombois, and P. S. Heuberger, "Identification of dynamic models in complex networks with prediction error methods: Predictor input selection," *IEEE Transactions on Automatic Control*, vol. 61, no. 4, pp. 937–952, 2016.

[35] J. Linder and M. Enqvist, "Identification and prediction in dynamic networks with unobservable nodes," *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 10 574–10 579, 2017.

[36] A. Dankers, P. M. Van den Hof, D. Materassi, and H. H. Weerts, "Conditions for handling confounding variables in dynamic networks," *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 3983–3988, 2017.

[37] J. Gonçalves, R. Howes, and S. Warnick, "Dynamical structure functions for the reverse engineering of LTI networks," in *IEEE Conference on Decision and Control (CDC)*. New Orleans, LA: IEEE, 2007, pp. 1516–1522.

[38] Y. Yuan, G.-B. Stan, S. Warnick, and J. Gonçalves, "Minimal dynamical structure realisations with application to network reconstruction from data," in *IEEE Conference on Decision and Control (CDC), held jointly with the IEEE Chinese Control Conference (CCC)*. Shanghai, China: IEEE, 2009, pp. 4808–4813.

[39] S. Warnick, "Shared hidden state and network representations of interconnected dynamical systems," in *Allerton Conference on Communication, Control, and Computing (Allerton)*. Monticello, IL: IEEE, 2015, pp. 25–32.

[40] E. Kivits and P. M. Van den Hof, "On representations of linear dynamic networks," *IFAC-PapersOnLine*, vol. 51, no. 15, pp. 838–843, 2018.

[41] Y. Yuan, G.-B. Stan, S. Warnick, and J. Goncalves, "Robust dynamical network structure reconstruction," *Automatica*, vol. 47, no. 6, pp. 1230–1235, 2011.

[42] E. Yeung, J. Kim, J. Gonçalves, and R. M. Murray, "Global network identification from reconstructed dynamical structure subnetworks: Applications to biochemical reaction networks," in *IEEE Conference on Decision and Control (CDC)*. Osaka, Japan: IEEE, 2015, pp. 881–888.

[43] V. Chetty, "Necessary and sufficient informativity conditions for robust network reconstruction using dynamical structure functions," M.S. Thesis, Brigham Young University, Provo, UT, 2012.

[44] A. S. Bazanella, M. Gevers, J. M. Hendrickx, and A. Parraga, "Identifiability of dynamical networks: which nodes need be measured?" in *IEEE Conference on Decision and Control (CDC)*. Melbourne, Australia: IEEE, 2017, pp. 5870–5875.

[45] J. M. Hendrickx, M. Gevers, and A. S. Bazanella, "Identifiability of dynamical networks with partial node measurements," *IEEE Transactions on Automatic Control*, 2018.

[46] P. M. Van den Hof, A. Dankers, P. S. Heuberger, and X. Bombois, "Identification of dynamic models in complex networks with prediction error methodsbasic methods for consistent module estimates," *Automatica*, vol. 49, no. 10, pp. 2994–3006, 2013.

[47] A. Dankers, "System identification in dynamic networks," Ph.D Dissertation, Delft University of Technology, Delft, Netherlands, 2014.

[48] L. Bunimovich and B. Webb, "Isospectral matrix reductions," in *Isospectral Transformations*. Springer, 2014, pp. 1–17.

[49] E. Sontag, A. Kiyatkin, and B. N. Kholodenko, "Inferring dynamic architecture of cellular networks using time series of gene expression, protein and metabolite data," *Bioinformatics*, vol. 20, no. 12, pp. 1877–1886, 2004.

[50] N. Friedman, K. Murphy, and S. Russell, "Learning the structure of dynamic probabilistic networks," in *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*. Madison, WI: Morgan Kaufmann notes Inc., 1998, pp. 139–147.

[51] N. Friedman, "Inferring cellular networks using probabilistic graphical models," *Science*, vol. 303, no. 5659, pp. 799–805, 2004.

[52] K. Basso, A. A. Margolin, G. Stolovitzky, U. Klein, R. Dalla-Favera, and A. Califano, "Reverse engineering of regulatory networks in human B cells," *Nature Genetics*, vol. 37, no. 4, p. 382, 2005.

[53] A. Haber and M. Verhaegen, "Subspace identification of large-scale interconnected systems," *IEEE Transactions on Automatic Control*, vol. 59, no. 10, pp. 2754–2759, 2014.

[54] V. Chetty and S. Warnick, "Necessary and sufficient conditions for identifiability of interconnected subsystems," in *IEEE Conference on Decision and Control (CDC)*. Melbourne, Australia: IEEE, 2017, pp. 5790–5795.

[55] R. Howes, L. Eccleston, J. Gonçalves, G.-B. Stan, and S. Warnick, "Dynamical structure analysis of sparsity and minimality heuristics for reconstruction of biochemical networks," in *IEEE Conference on Decision and Control (CDC)*. Cancun, Mexico: IEEE, 2008, pp. 173–178.

[56] V. Chetty, D. Hayden, J. Goncalves, and S. Warnick, "Robust signal-structure reconstruction," in *IEEE Conference on Decision and Control (CDC)*. Florence, Italy: IEEE, 2013, pp. 3184–3189.

[57] D. Hayden, Y. H. Chang, J. Goncalves, and C. J. Tomlin, "Sparse network identifiability via compressed sensing," *Automatica*, vol. 68, pp. 9–17, 2016.

[58] H. H. Weerts, P. M. Van den Hof, and A. G. Dankers, "Prediction error identification of linear dynamic networks with rank-reduced noise," *Automatica*, vol. 98, pp. 256–268, 2018.

[59] J. Adebayo, T. Southwick, V. Chetty, E. Yeung, Y. Yuan, J. Goncalves, J. Grose, J. Prince, G.-B. Stan, and S. Warnick, "Dynamical structure function identifiability conditions enabling signal structure reconstruction," in *IEEE Conference on Decision and Control (CDC)*. Maui, HI: IEEE, 2012, pp. 4635–4641.

[60] M. Gevers and A. S. Bazanella, "Identification in dynamic networks: identifiability and experiment design issues," in *IEEE Conference on Decision and Control (CDC)*. Osaka, Japan: IEEE, 2015, pp. 4005–4010.

[61] H. H. Weerts, P. M. Van den Hof, and A. G. Dankers, "Identifiability of dynamic networks with part of the nodes noise-free," *IFAC-PapersOnLine*, vol. 49, no. 13, pp. 19–24, 2016.

[62] A. Arnold, Y. Liu, and N. Abe, "Temporal causal modeling with graphical Granger methods," in *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. San Jose, CA: ACM, 2007, pp. 66–75.

[63] A. Fujita, J. R. Sato, H. M. Garay-Malpartida, R. Yamaguchi, S. Miyano, M. C. Sogayar, and C. E. Ferreira, "Modeling gene expression regulatory networks with the sparse vector autoregressive model," *BMC Systems Biology*, vol. 1, no. 1, p. 39, 2007.

[64] A. C. Lozano, N. Abe, Y. Liu, and S. Rosset, "Grouped graphical Granger modeling for gene expression regulatory networks discovery," *Bioinformatics*, vol. 25, no. 12, pp. i110–i118, 2009.

[65] A. Shojaie and G. Michailidis, "Discovering graphical Granger causality using the truncating lasso penalty," *Bioinformatics*, vol. 26, no. 18, pp. i517–i523, 2010.

[66] S. Basu, A. Shojaie, and G. Michailidis, "Network Granger causality with inherent grouping structure," *The Journal of Machine Learning Research*, vol. 16, no. 1, pp. 417–453, 2015.

[67] G. Yang, L. Wang, and X. Wang, "Reconstruction of complex directional networks with group lasso nonlinear conditional Granger causality," *Scientific Reports*, vol. 7, no. 1, p. 2991, 2017.

[68] C. Zou and J. Feng, "Granger causality vs. dynamic Bayesian network inference: a comparative study," *BMC Bioinformatics*, vol. 10, no. 1, p. 122, 2009.

[69] H. J. van Waarde, P. Tesi, and M. K. Camlibel, "Topology reconstruction of dynamical networks via constrained Lyapunov equations," *IEEE Transactions on Automatic Control*, 2019.

[70] P. E. Paré Jr, "Necessary and sufficient conditions for state-space network realization," M.S. Thesis, Brigham Young University, Provo, UT, 2014.

[71] D. Materassi and M. V. Salapaka, "On the problem of reconstructing an unknown topology via locality properties of the Wiener filter," *IEEE Transactions on Automatic Control*, vol. 57, no. 7, pp. 1765–1777, 2012.

[72] G. Innocenti and D. Materassi, "Modeling the topology of a dynamical network via Wiener filtering approach," *Automatica*, vol. 48, no. 5, pp. 936–946, 2012.

[73] D. Materassi and M. V. Salapaka, "Network reconstruction of dynamical polytrees with unobserved nodes," in *IEEE Conference on Decision and Control (CDC)*. Maui, HI: IEEE, 2012, pp. 4629–4634.

[74] S. Talukdar, D. Deka, S. Attree, D. Materassi, and M. V. Salapaka, "Learning the exact topology of undirected consensus networks," *arXiv preprint arXiv:1710.00032*, 2017.

[75] S. Talukdar, D. Deka, D. Materassi, and M. Salapaka, "Exact topology reconstruction of radial dynamical systems with applications to distribution system of the power grid," in *IEEE American Control Conference (ACC)*. Seattle, WA: IEEE, 2017, pp. 813–818.

[76] M. Dimovska and D. Materassi, "Granger-causality meets causal inference in graphical models: Learning networks via non-invasive observations," in *IEEE Conference on Decision and Control (CDC)*. Miami, FL: IEEE, 2017, pp. 5268–5273.

[77] S. Jahandari and D. Materassi, "Topology identification of dynamical networks via compressive sensing," *IFAC-PapersOnLine*, vol. 51, no. 15, pp. 575–580, 2018.

[78] B. M. Sanandaji, T. L. Vincent, and M. B. Wakin, "Exact topology identification of large-scale interconnected dynamical systems from compressive observations," in *IEEE American Control Conference (ACC)*. San Francisco, CA: IEEE, 2011, pp. 649–656.

[79] A. Chiuso and G. Pillonetto, "A Bayesian approach to sparse dynamic network identification," *Automatica*, vol. 48, no. 8, pp. 1553–1565, 2012.

[80] C. J. Quinn, T. P. Coleman, N. Kiyavash, and N. G. Hatsopoulos, "Estimating the directed information to infer causal relationships in ensemble neural spike train recordings," *Journal of Computational Neuroscience*, vol. 30, no. 1, pp. 17–44, 2011.

[81] A. Dankers, P. M. Van den Hof, X. Bombois, and P. S. Heuberger, "Errors-in-variables identification in dynamic networksconsistency results for an instrumental variable approach," *Automatica*, vol. 62, pp. 39–50, 2015.

[82] N. Everitt, G. Bottegal, and H. Hjalmarsson, "An empirical Bayes approach to identification of modules in dynamic networks," *Automatica*, vol. 91, pp. 144–151, 2018.

[83] K. R. Ramaswamy, G. Bottegal, and P. M. Van den Hof, "Local module identification in dynamic networks using regularized kernel-based methods," in *IEEE Conference on Decision and Control (CDC)*. Miami, FL: IEEE, 2018, pp. 4713–4718.

[84] J. Linder, "Indirect system identification for unknown input problems: With applications to ships," Ph.D. Dissertation, Linköping University, Linköping, Sweden, 2017.

[85] J. Linder and M. Enqvist, "Identification of systems with unknown inputs using indirect input measurements," *International Journal of Control*, vol. 90, no. 4, pp. 729–745, 2017.

[86] D. Materassi and M. V. Salapaka, "Identification of network components in presence of unobserved nodes," in *IEEE Conference on Decision and Control (CDC)*. Osaka, Japan: IEEE, 2015, pp. 1563–1568.

[87] ——, "Methods for network identification robust with respect to uncertainties in a topology," in *IEEE American Control Conference (ACC)*. Boston, MA: IEEE, 2016, pp. 4680–4680.

[88] H. H. Weerts, J. Linder, M. Enqvist, and P. M. Van den Hof, "Abstractions of linear dynamic networks for input selection in local module identification," *arXiv preprint arXiv:1901.00348*, 2019.

[89] F. N. Karameh and Z. Nahas, "A blind module identification approach for predicting effective connectivity within brain dynamical subnetworks," *Brain Topography*, vol. 32, no. 1, pp. 28–65, 2019.

[90] K. F. Aljanaideh and D. S. Bernstein, "Initial conditions in time-and frequency-domain system identification: Implications of the shift operator versus the z and discrete Fourier transforms," *IEEE Control Systems Magazine*, vol. 38, no. 2, pp. 80–93, 2018.