2017-05-01

Network Reconstruction and Vulnerability Analysis of Financial Networks

Nathan Scott Woodbury
Brigham Young University

Network Reconstruction and Vulnerability Analysis of

Financial Networks

Nathan Scott Woodbury

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

Master of Science

Sean Warnick, Chair
Scott Condie
Dan Ventura

Department of Computer Science

Brigham Young University

ABSTRACT

Network Reconstruction and Vulnerability Analysis of
Financial Networks

Nathan Scott Woodbury
Department of Computer Science, BYU
Master of Science

Passive network reconstruction is the process of learning a structured (networked) representation of a dynamic system through the use of known information about the structure of the system as well as data collected by observing the inputs into a system along with the resultant outputs. This work demonstrates an improvement on an existing network reconstruction algorithm so that the algorithm is capable of consistently and perfectly reconstructing a network when system inputs and outputs are measured without error. This work then extends the improved network reconstruction algorithm so that it functions even in the presence of noise as well as the situation where inputs into the system are unknown. Furthermore, this work demonstrates the capability of the new extended algorithms by reconstructing financial networks from stock market data, and then performing an analysis to understand the vulnerabilities of the reconstructed network to destabilization through localized attacks. The creation of these improved and extended algorithms has opened many theoretical questions, paving the way for future research into network reconstruction.

Keywords: Network reconstruction, realization theory, system identification, vulnerability analysis, financial networks, partial structure representation, linear time-invariant systems

ACKNOWLEDGMENTS

I would like to give a special thank you to Vasu Chetty, who was the author of the original passive network reconstruction paper and who provided considerable feedback throughout the development of this thesis. I would also like to thank my adviser, Dr. Sean Warnick, as well as the rest of my committee for supporting me in this project. I would also like to thank my family and my colleagues in the IDeA Labs for their continued support.

Table of Contents

List of Figures

List of Tables

List of Listings

Chapter 1

Problem Statement and Contributions

In this chapter, we briefly outline the problem we are seeking to solve in this thesis. We also provide a road-map to the theoretical foundations to and contributions from this thesis.

1.1 Problem Statement

The ultimate objective of this thesis is to analyze the vulnerability of a financial network to a single-link destabilization attack. A destabilization attack is a local attack on a single link within a network with the objective of destabilizing the system–or, in other words, causing cascading failure throughout the system. A vulnerability analysis of a network to destabilization attacks asks the question of which link in the network would require the least amount of effort to destabilize.

To perform this vulnerability analysis, we first need a model of the financial network. Rather than build such a model from first principles, we seek instead to learn the model using financial data. The process of learning a network model from observed outputs from and/or inputs into a system is known as network reconstruction.

Much of this thesis is devoted to formulating and solving a sequence of related network reconstruction problems so that the reconstruction of financial networks is possible.

1.2 Theoretical Road-map

A road-map of the theory on which this thesis is built, as well as the contributions from this thesis, is given in Figure 1.1. The first–and arguably most significant–foundation to this work is linear systems theory; in particular, the theory of dynamical structure functions (DSFs). A brief introduction to linear systems and DSF theory is provided in Chapter 2.

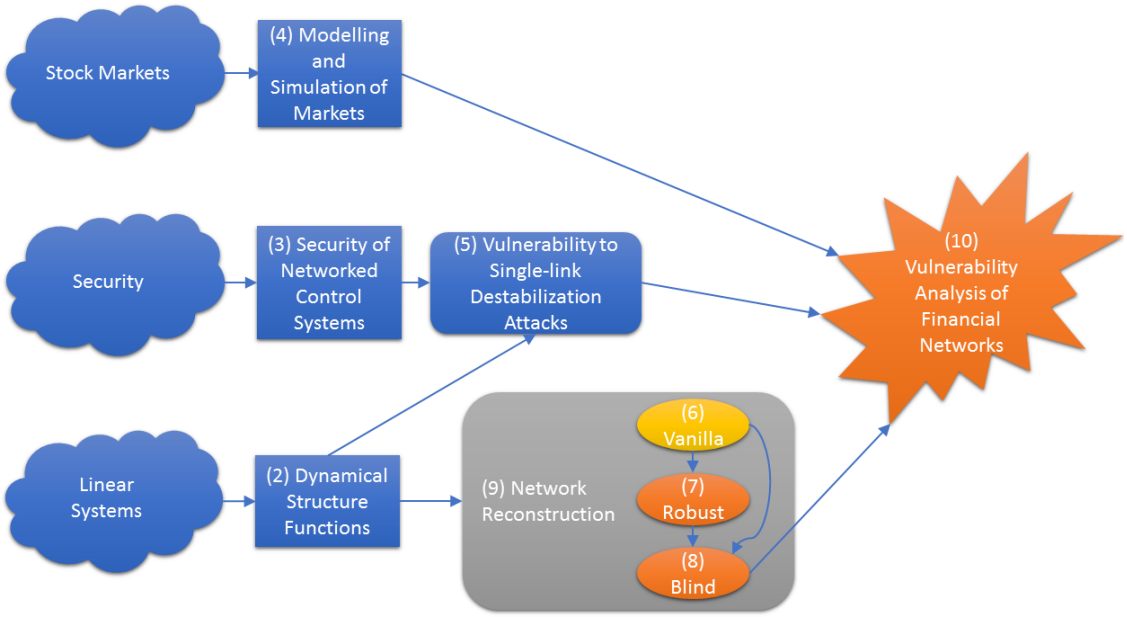


Figure 1.1 A theoretical road-map of the foundations for and contributions from this thesis. Orange nodes are a direct result of this thesis. Yellow nodes had theory that existed prior to this thesis, but also had significant contributions from this work. Nodes are prefixed by the chapters in which the node is discussed.

The second foundation is security; in particular, the security of networked control systems (NCSs). A background into the theory of security of NCSs is provided in Chapter 3. The final foundation to this work is the theory of stock markets, and in particular, the modelling and simulation of stock markets. This foundation is discussed in Chapter 4.

DSF theory has two significant applications. The first–which also has a foundation in the security of NCSs–is vulnerability analysis of networked systems to single-link destabilization attacks (which, for brevity, we will refer to as "vulnerability analysis"). This analysis was

discussed briefly in the previous section and will be discussed in further detail in Chapter 5. The second is network reconstruction, which will be discussed in further detail in Section 6.1.

This thesis refines the process of passive (meaning inputs into the system are observed, not controlled) network reconstruction of dynamical structure functions in the time domain that was originally introduced in [10]. The refined algorithm is known as the Vanilla Passive Network Reconstruction (VPNR) Algorithm and is discussed in depth in Chapter 6.

We then make an extension to the VPNR algorithm to make it more robust to additive noise on the data received. The resultant algorithm is called the Robust Passive Network Reconstruction (RPNR) Algorithm. The RPNR Algorithm is discussed in Chapter 7. We extend both the VPNR and the RPNR algorithms so that they function in the situation where the inputs into the systems are unknown and unmeasured. The extended algorithms are called the Blind Vanilla Passive Network Reconstruction (B-VPNR) Algorithm and the Blind Robust Passive Network Reconstruction (B-RPNR) Algorithms. These Algorithms are discussed in Chapter 8. Finally, open questions in passive network reconstruction are discussed in Chapter 9.

The culmination of this thesis is Chapter 10, which uses both blind reconstruction algorithms to build network models of the stock market, and then leverages the vulnerability analysis to determine where these networks may be most susceptible to attack. Note that the development of the blind reconstruction algorithms is necessary for the reconstruction of financial networks since output data (stock prices) is easily accessible, whereas a complete collection of input data (news, sentiment, external market forces, etc.) would be nearly impossible to find.

1.3 Contributions

In summary, the direct contributions of this thesis are as follows:

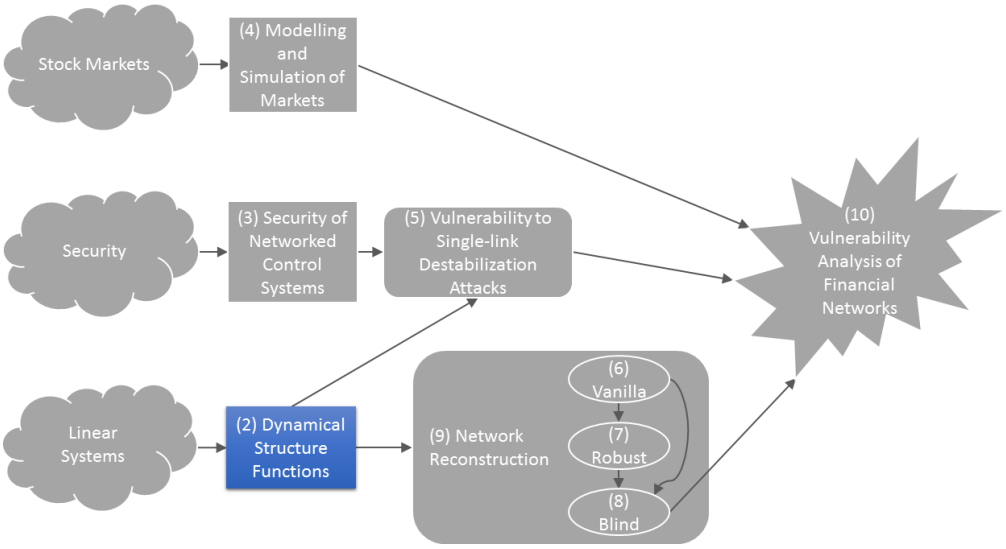
- *Vanilla Reconstruction*: Refinement of the implementation of the passive reconstruction algorithm presented in [10], enabling (as described in Chapter 6):

  - Consistent and accurate reconstruction without the need to choose initial conditions and additional parameters
  - Reconstruction of proper dynamics while evaluating a much smaller time horizon (100 instead of 600 in the numerical example given)

- *Robust Passive Reconstruction*: Extension of the passive reconstruction algorithm presented in [10] to be robust to noise (Chapter 7).
- *Blind Passive Reconstruction*: Extension of the robust passive reconstruction algorithm to consider the case where only the outputs to the network (and not the inputs) are measured (Chapter 8).
- *Reconstruction and Vulnerability Analysis of Financial Networks*: Application of the newly developed network reconstruction techniques along with an existing vulnerability analysis to stock market data (Chapter 10).

Chapter 2

Dynamical Structure Functions



The Dynamical Structure Function (DSF) is a convenient way to represent the signal structure of a linear time-invariant (LTI) system (note that, throughout this work, all references to systems are in actuality references to continuous and LTI).

2.1 Background

The DSF of a discrete-time LTI system is characterized by the pair $(Q(z), P(z))$ (or $(Q(s), P(s))$ for continuous-time LTI systems), which of matrices of real rational polynomials in terms of $z \in \mathbb{C}$. If $Y(z)$ and $U(z)$ are the frequency-domain representation of the outputs and inputs of some system, then the DSF relates inputs and outputs through the following equation:

$$Y(z) = Q(z)Y(z) + P(z)U(z). \tag{2.1}$$

Note that the transfer function matrix $G(z)$ is the black-box mapping from inputs to outputs given by

$$Y(z) = G(z)U(z). \tag{2.2}$$

By solving for $Y(z)$ in Equation (2.1) we can define the relationship between the transfer function and the DSF with the following equation:

$$G(z) = (I - Q(z))^{-1}P(z). \tag{2.3}$$

Equation (2.3) highlights the fact that the DSF is a left factorization of the transfer function and potentially contains more structural information about the system (see [7, 45]). This makes DSFs particularly useful in many applications.

Since DSFs can abstract away some of the information contained in the transfer function, a DSF can be reconstructed from input-output data while requiring less a prior knowledge about the structure of the system than is necessary for the system. It should be noted that, since input-output data can only reconstruct input-output maps such as the Transfer Function, the reconstruction of the DSF will still require some a priori knowledge about the system (see Chapter 6).

Furthermore, since DSFs can represent abstractions away from the full computational structure of a system, they can be used to represent the amount of knowledge that an attacker may have about the system, making them a friendly mathematical structure for the analysis of attacks on networked systems. Theory has been developed using DSFs to understand how local perturbations in a network systems–which are called destabilization attacks–can cause cascading failure throughout the entire system (see Chapter 5).

Since this thesis involves the reconstruction of financial networks from data, coupled with a vulnerability analysis of said networks, DSFs will form the core mathematical foundation for this work.

2.2 From State Space to Dynamical Structure Function

The process of deriving the DSF from the State Space Representation of a system is contained in [7]. Since DSFs are a critical component of this work, we include those results here. Note that [7] derives the DSF for continuous-time systems, the process is identical for discrete-time systems if the $\mathcal{Z}$-transform is used in place of the Laplace transform. We show the discrete-time derivation here.

Consider a system represented by the following state space equations:

$$\begin{aligned} \hat{x}[k+1] &= \hat{A}\hat{x}[k] + \hat{B}u[k], \\ y[k] &= \hat{C}\hat{x}[k] + \hat{D}u[k]., \end{aligned} \tag{2.4}$$

where $\hat{A} \in \mathbb{R}^{n\times n}$, $\hat{B} \in \mathbb{R}^{n\times m}$, $\hat{C} \in \mathbb{R}^{l\times n}$, and $\hat{D} \in \mathbb{R}^{l\times m}$. Then the procedure for finding the Dynamical Structure Function (DSF) representation of this system is as follows:

1. Let $p = rank(\hat{C})$. Partition and permute $\hat{C}$ so that

$$\hat{C} = \begin{bmatrix} \hat{C}_1 \\ \hat{C}_2 \end{bmatrix}, \tag{2.5}$$

with $\hat{C}_1 \in \mathbb{R}^{p\times n}$ and $rank(\hat{C}_1) = p$ (note that if $\hat{C}$ is full row rank, then $\hat{C}_1 = \hat{C}$ and $\hat{C}_2$ is empty). Also, permute the rows of y in the same manner, so that $y = [y_1 \;\; y_2]^\mathsf{T}$ with $y_1 \in \mathbb{R}^p$.

The dynamical structure function will then be given by the $(l \times p)$ and $(l \times n)$ real rational matrix functions $Q(z)$ and $P(z)$, which will be computed in the subsequent steps.

2. Create an invertible transformation $T \in \mathbb{R}^{n\times n}$ given by

$$T = \begin{bmatrix} \hat{C}_1^\mathsf{T} & E_1 \end{bmatrix}^\mathsf{T}, \tag{2.6}$$

where $E_1 \in \mathbb{R}^{n\times(n-p)}$ is any basis of the null space of $\hat{C}_1$. Then,

$$T^{-1} = \begin{bmatrix} R_1 & E_1 \end{bmatrix}, \tag{2.7}$$

where $R_1 = \hat{C}_1^{\mathsf{T}}(\hat{C}_1\hat{C}_1^{\mathsf{T}})^{-1}$.

3. Change basis such that $x[k] = T\hat{x}[k]$, yielding $A = T\hat{A}T^{-1}$, $B = T\hat{B}$, $C = \hat{C}T^{-1}$, $D = \hat{D}$, which are partitioned commensurate with the block partitioning of T and T^{-1} to give

$$\begin{aligned} \begin{bmatrix} x_1[k+1] \\ x_2[k+1] \end{bmatrix} &= \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} x_1[k] \\ x_2[k] \end{bmatrix} + \begin{bmatrix} B_1 \\ B_2 \end{bmatrix} u[k], \\ \begin{bmatrix} y_1[k] \\ y_2[k] \end{bmatrix} &= \begin{bmatrix} I & 0 \\ C_{21} & 0 \end{bmatrix} \begin{bmatrix} x_1[k] \\ x_2[k] \end{bmatrix} + \begin{bmatrix} D_1 \\ D_2 \end{bmatrix} u[k]. \end{aligned} \tag{2.8}$$

4. Assume zero initial conditions and defining X_1, X_2, U, and Y to be the $\mathcal{Z}$-transforms of $x_1[k]$, $x_2[k]$, $u[k]$, and $y[k]$ respectively, take the $\mathcal{Z}$-transforms of this system, and then solve for X_2, yielding

$$\begin{aligned} zX_1 &= \left[A_{11} + A_{12}(zI - A_{22})^{-1}A_{21}\right] X_1 \\ &\quad + \left[B_1 + A_{12}(zI - A_{22})^{-1}B_2\right] U, \\ \begin{bmatrix} Y_1 \\ Y_2 \end{bmatrix} &= \begin{bmatrix} I & 0 \\ C_{21} & 0 \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \end{bmatrix} + \begin{bmatrix} D_1 \\ D_2 \end{bmatrix}. \end{aligned} \tag{2.9}$$

5. Define:

$$W(z) = A_{11} + A_{12}(zI - A_{22})^{-1}A_{21}, \tag{2.10}$$

$$V(z) = B_1 + A_{12}(zI - A_{22})^{-1}B_2, \tag{2.11}$$

and let $D_W(z) = diag(W(z))$ be the matrix function consisting only of the diagonal entries of $W(z)$ (and zero elsewhere).

6. Let

$$\hat{Q}(z) = (sI - D_W(z))^{-1}(W(z) - D_W(z)), \tag{2.12}$$

$$\hat{P}(z) = (sI - D_W(z))^{-1}V(z). \tag{2.13}$$

Then

$$\begin{aligned} X_1 &= Q(z)X_1 + P(z)U \\ \begin{bmatrix} Y_1 \\ Y_2 \end{bmatrix} &= \begin{bmatrix} I & 0 \\ C_{21} & 0 \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \end{bmatrix} + \begin{bmatrix} D_1 \\ D_2 \end{bmatrix}. \end{aligned} \tag{2.14}$$

7. Noting from 2.14 that $X_1 = Y_1 - D_1U$, the DSF of Equation (2.4) is then given by:

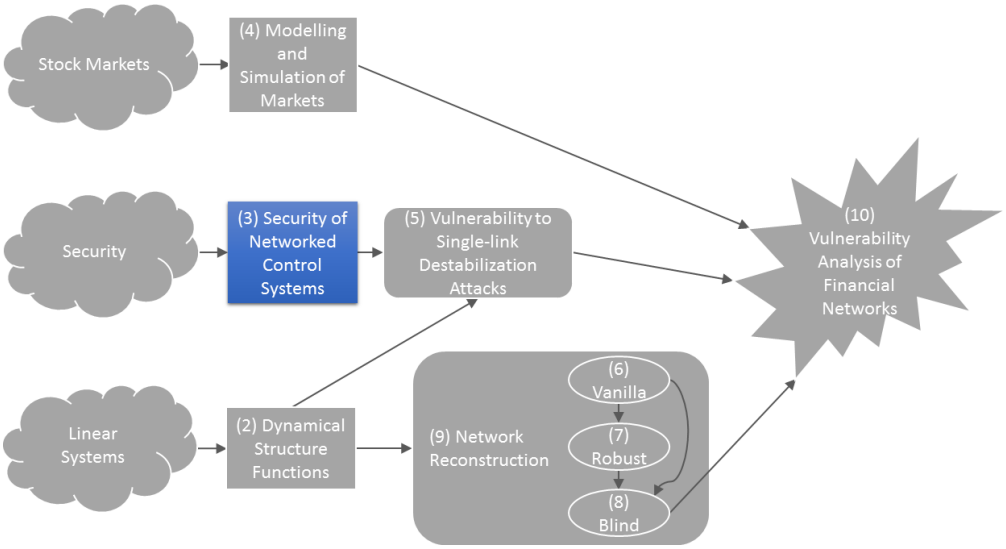
$$Q(z) = \begin{bmatrix} \hat{Q}(z) \\ C_{21} \end{bmatrix} \tag{2.15}$$

$$P(z) = \begin{bmatrix} P(z) + (I - Q(z))D_1 \\ D_2 - C_{21}D_1 \end{bmatrix}. \tag{2.16}$$

Note that when $D = 0$ and C_{21} is empty, $Q(s) = \hat{Q}(s)$ and $P(s) = \hat{P}(s)$. This simplified version of the DSF is often used in place of the extended DSF above, and will be the form used in this thesis.

Chapter 3

Vulnerability and Security of Networked Control Systems



Traditional IT systems interact primarily with information and data. As a result, information security has developed significant and mature technology that is focused entirely on the protection of information (consider, for example, authentication, access control, message integrity, privacy preservation, etc.). These technologies are necessary to the security of control systems. Unfortunately, as shown by a few of the incidents in Section 3.2, information security is often lacking in many critical NCS's, allowing attackers to cause considerable amount of damage with relative ease.

NCS's differ from IT systems in that they are created by a computational system that is connected in feedback with a physical system. As such, information security is not entirely sufficient in the protection of networked control systems. Ultimately, traditional IT research has not considered how attacks might affect the estimation and control algorithms that are

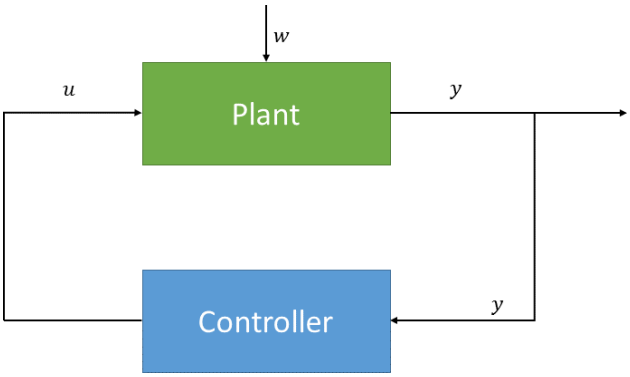


Figure 3.1 The feedback interaction between a plant and a controller. The controller is designed to stabilize the plant, driving it to a desired state.

critical to the functionality of NCS's. Furthermore, they do not consider the affect attacks may have on the physical world. As a result, security research on NCS's must consider threats at both the cyber and the physical layers of the system [5, 36, 43].

Systems and control theory has a long tradition of developing mathematical frameworks that are broad enough to model most–if not all–NCS's, and yet are precise enough to conduct rigorous and meaningful robustness analyses that are critical to understanding the security of an NCS. As such, many researchers are turning to control theory as a framework for posing and solving security problems relating to NCS's. Thus, it has been proposed that control theory can offer the following contributions to the security community [12]:

- Technology neutral tools that complement platform- or protocol-specific security systems
- Proactive (as opposed to reactive) tools
- Rigorous analysis of NCS's, including those that involve humans in the decision-making process
- Strategic wide-area perspectives of the cyber-ecosystem

3.1 Networked Control Systems

Much of the theory of control is built around Figure 3.1. In this figure, we have a plant (which could be an internet network, a power grid, an economy, an ecosystem, a robot, etc.),

which is a system with state (memory) that evolves over time. This state can be affected by actuators, which are driven by inputs into the system. Some of these inputs (u) can be controlled, while others (w) cannot. We can also measure some of the states of the system with sensors to produce a signal y.

The objective, therefore, is to design a controller that reads in the signal y from the sensors of the system and then makes decisions to drive the system to a desired stable state. The combined system consisting of the plant and controller in feedback becomes a *control system*.

As described previously, networked control systems (NCS's) are set apart from most traditional IT systems by their interaction with the physical world. However, this interaction does not necessarily have to occur through machines. For example, the following systems could be considered as NCS's:

- Financial networks–the focus of this thesis–which interact through assets, securities, prices, etc.
- Social networks which interact through people and relationships
- Modern agricultural systems [9] which interact through farm equipment, farmer decisions, biology, chemistry, etc.

Each of these non-traditional control systems are still described by the feedback interaction between a controller (which is often code that exists in cyberspace) and a plant (the physical system). The plant may be distributed physically (e.g. power lines, meters, stations, plants, etc.) with individual components connected to distributed locations on the cloud. The communications u and y between components of the plant and components then occur over standard or specialized protocols (such as TCP/IP for the web or 4G over cellular networks).

While the inclusion of a physical layer is the primary difference between a NCS and an IT system, it has been recognized that they differ in a few additional properties as well [5, 12]:

- *Availability as the Primary Objective:* NCS's such as power systems, water systems, transportation networks, etc. must provide continuous service even in the presences of failures, faults, and attacks, as opposed to IT systems where confidentiality and integrity of data is the primary concern
- *Simpler Network Dynamics:* Servers rarely change, the network exhibits a fixed topology, the user population is stable, and communication patterns are regular and use a limited number of protocols
- *Infrequent Updates:* Upgrading a system may require months of planning of how to take a system safely offline for an upgrade; therefore, patching and frequent updates are not economically feasible
- *Real-time Requirements:* A NCS typically needs to make decisions in real time, creating a stricter operating environment than most traditional IT systems

Since NCS's are specialized forms of a control systems, we can leverage the long and rich tradition of control theory to discover ways to secure NCS's from various forms of attack.

3.1.1 Common Terminology

Terms that have been associated with networked control systems include SCADA systems, cyber-physical systems, and critical infrastructure systems. Frequently, these terms are used interchangeably, though there are subtle differences between them.

Cyber-physical systems (CPS's) are defined as the integrations of physical processes with computation. Embedded computers or networks use sensors to monitor the physical processes and controls them through actuators. Thus, in CPS's, feedback allows computation to affect physical processes and vice versa [21].

Networked control systems (NCS's) are often defined as a CPS where the communications u and y between the physical system and the controller are passed through a network which may be shared with nodes outside the system (the internet or a cellular network, for

example [44]). Under this definition, the engine inside a modern vehicle can be considered to be a CPS since it is controlled through an on-board computer. However, it is not an NCS since all communication between the engine and the on-board computer is contained within the vehicle and not tied to an external network. Conceivably, however, engines in the future could become part of NCS's with the advent of autonomous vehicles, tying the control of a vehicle into a larger network.

While most of the work generally focuses on NCS's–specifically financial networks–it could also be more broadly applicable to the general class of *networked systems*, which may be entirely cyber or entirely physical, and which may utilize a variety of communication schemes.

Supervisory Control and Data Acquisition (SCADA) systems are the software frameworks for controllers on NCS's (especially large industrial NCS's). They allow the real-time monitoring of data measured by the sensors within the network, and allow human or automated controllers to drive the system to its desired state. Recently, SCADA systems have been trending to become more standardized [51].

The United States defines *Critical Infrastructure Systems* as "certain infrastructures whose incapacity or destruction would have a debilitating impact on our defense or economic security [32]." These infrastructures do not necessarily have to be NCS's or CPS's; however, they are increasingly becoming so as new technologies are developed and implemented. The US Department of Homeland Security oversees 14 categories of critical infrastructure systems, namely agriculture and food, banking and finance, the chemical sector, the communications sector, commercial facilities, critical manufacturing, dams, information technology, government facilities, health-care and public health, water, nuclear reactors, the defense industrial base, transportation, and emergency services [12].

3.2 Historic Attacks on Networked Control Systems

NCS's have been attacked on multiple occasions in the past, with varying levels of success. In this section, we explore some of the most well-known attacks on NCS's.

3.2.1 Stuxnet

In 2010, the Stuxnet worm made history as one of the earliest cyber-warfare weapons created. The worm much more complex than all preceding pieces of malware, and unlike its predecessors, it was targeted at physically destroying a military target (as opposed to stealing, manipulating, or erasing information). Interestingly, the worm was not targeted at SCADA software; rather, it was aimed at the industrial controllers that might not even be attached to a SCADA system. Furthermore, the worm was not remotely controlled and didn't even require access to the internet to function.

The worm spread primarily through USB sticks and local networks, infecting any Windows machines it could find. However, it was very careful to only attack controllers that were manufactured by Siemens. As a result, it was capable of specifically targeting the Natanz uranium enrichment plant in Iran.

Stuxnet operated as a replay attack (see Section 3.3.1), allowing the original controller to continue to operate while isolating it from the actual input/output of the system, instead feeding it prerecorded data, much like the bad guys in a Hollywood film that replace the feed of observation cameras with unsuspicious prerecorded input [15, 20].

3.2.2 Maroochy Water Services

In January of 2000, a new SCADA system was brought online to monitor and control the sewage system at Maroochy Water Services in Queensland, Australia. Soon thereafter, this new system was plagued by considerable technical issues. Radio communications sent between the control center and pumping stations were being lost, pumps were not functioning properly, and the alarms put in place to alert staff to problems were not being triggered. These

problems caused the flooding of a nearby hotel, a park, and a river with approximately a million liters of sewage.

These problems persisted for roughly three months before engineers realized that the faults were not technical, but rather attacks launched by an outside attacker. Vitek Boden, a former contractor who originally installed the control system, was perpetrating these attacks in part as revenge against the Maroochy Shire Council after failing to secure a job there and in part as an attempt to convince the company to hire him to solve the problem. Over this period, with only a laptop computer and a radio transmitter, he successfully took control of 150 sewage pumping stations and launched 46 attacks against them.

The Maroochy incident quickly became highly cited worldwide as an example of the damage that can be caused from an attack on a NCS. The incident and further investigation has revealed that often it is very difficult to detect and protect against attacks against NCS's. Furthermore, many of these systems fail to properly implement and secure their software and communication protocols [5, 42].

3.2.3 Polish Tram System Hacked by Teenager

In 2008, a 14-year-old Polish teenager modified a TV remote to change the track points of the tram system in the city of Lodz as a prank. As a result, four trains were derailed and a dozen people were injured. Fortunately, nobody was killed. The ease with which this teenager hacked this critical system is an eye opener to the importance of security in NCS's [23].

3.2.4 Power Blackouts

In 2003, an incident now known as the Northeast Blackout, caused cascading failure across seven U.S. states and Ontario. In 2011, a similar incident hit Arizona, California, and Mexico's Baja California. In both cases, inadequate information, poor planning, and human error enabled a single downed power line to leave millions of homes and businesses without

power. Experts predict that such blackouts will continue to occur at greater frequency and magnitude [14].

While these blackouts were not caused by malicious attacks, they illustrate the vulnerability of existing NCS's to small faults. Furthermore, a malicious attacker could take advantage of these vulnerabilities to cause considerable damage.

3.3 Potential Attacks on Networked Control Systems

As described previously, the security of NCS's is dependent on the principles studied in traditional IT security. However, these principles are not entirely sufficient to guarantee the security of the system. Since, at their core, NCS's are feedback control systems, classical control theory can be used to not only highlight vulnerabilities of the NCS that aren't always apparent through IT security analysis, but it can also be used to protect against these vulnerabilities.

In this section, we explore two of the most basic concepts in control theory, namely stability and observability, and highlight some of the vulnerabilities that become apparent when applying those concepts to NCS's.

3.3.1 Observability and Related Vulnerabilities

Observability is a measure of how well the state (or more precisely, the initial condition) of the system can be inferred by measuring its external outputs.

More formally, let $x(t)$ be the state of the system at time t. A state $q \neq 0$ is *unobservable* if, for initial condition $x(0) = q$ and every possible input into the system, the output is the same as if the system had started with $x(0) = 0$. In other words, the non-zero initial condition cannot be distinguished from the initial condition. An *unobservable system* is defined as a system with at least one unobservable state. An *observable system* is then defined as a system that is not unobservable [13]. As a relaxation of observability, *detectability*

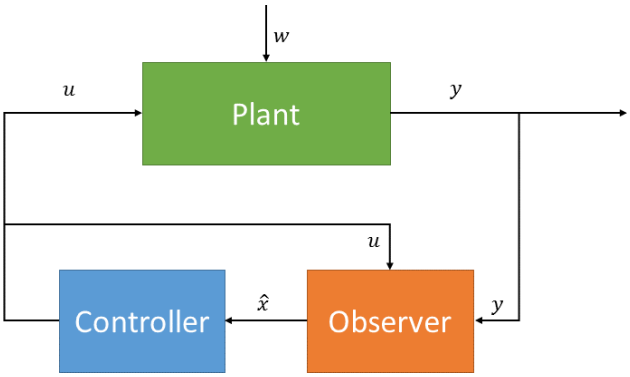


Figure 3.2 A control system equipped with an observer. The observer records the controller's inputs u into the plant and observes the output y from the plant that results from that input. Using this information, the observer provides an estimate $\hat{x}$ of the current state of the plant to the controller. Note, if the observer and controller are combined into a single system, then this figure is equivalent to Figure 3.1.

is defined if all unobservable states converge to zero, meaning that unobservable states are stable and trivial.

If the plant (the physical system) is detectable, but the state of the physical system is not known by the controller, the controller will require an *observer* to estimate the state of the system (Figure 3.2). An observer logs all the inputs u that the controller feeds into the plant and observes the outputs y that results. Internally, the observer runs a simulation of the plant using the inputs received by the actual system as well as its best estimate of the initial condition of the plant, and then compares the simulated output to the actual output from the plant. Supposing that the simulation of the plant is accurate, then the error between the predicted and observed output will be zero if the estimated initial condition is equal to the actual. If not, then the observer adjusts the initial condition until this error is minimized. Once a good estimate of the initial condition is found, then the observer can estimate the current state of the system and forward that to the controller.

If the plant and the controller are both governed by linear, time-invariant systems, then the Kalman Filter has been proven to be the optimal observer, meaning that the error between predicted and observed output converges to zero more quickly than any other possible

observer. Furthermore, the Kalman Filter is robust to noise on the output measurements and on the states of the plant [13].

Denial of Service Attacks

On a network governed by the TCP protocol, a denial of service (DoS) attack occurs when an attacker sends many requests from spoofed sources to a victim machine. Once a maximum number of connections is made, under TCP, the victim will block all future connections. Unfortunately, it is relatively cheap to launch DoS attacks anonymously; as such it is notoriously difficult to defend against such attacks [37], though proposals have been made and shown to significantly increase the cost of launching a DoS attack [3, 19].

In the context of a NCS, a DoS attack can interrupt part or all the communications between the physical system and the cyber controller. If the output of the physical system (which is used as the input to the controller) becomes sufficiently disrupted, then the physical system becomes unobservable, preventing the plant from choosing the proper control law to drive the system to the desired state. Similarly, if the output of the controller (which is the input into the physical system) becomes sufficiently disrupted, the controller becomes unobservable, preventing the system from receiving the proper control low to drive it to the desired state.

In contrast to a DoS attack on an IT system which merely prevents access to a desired service on the web, a DoS attack on an NCS can cause considerable performance problems across the entire system, possibly causing it to destabilize entirely [25].

The work in [2] seeks to find a controller to minimize the effect of random and/or deliberate dropping of packets transmitted between the system and controller, posing the problem as a constrained optimal control problem. In control theory, the optimal control problem is the choice of selecting a controller from a class of controllers that best meets some performance objective. Frequently, this performance objective is selected to minimize the energy used by the states of and inputs into the plant (the physical system, in our case)

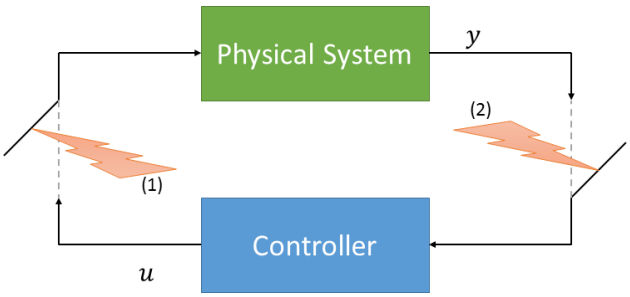


Figure 3.3 A denial of service attack on a networked control system. Attack (1) blocks part or all of the control signal u from the controller to the physical system, making the controller unobservable from the physical system. Attack (2) blocks part or all of the output signal y from the physical system to the controller, making the physical system unobservable from the controller.

required to drive the system to its desired stable state over either a finite or an infinite time horizon. Constraints to the inputs and/or states at any given time may be added as well [22].

The objective of the optimal controller presented in [2] is to generate a minimizes the total energy used to drive the physical system to the desired state in a finite horizon. Constraints are also added that, with high probability under the studied attack models (1) limit the amount of energy used at each time step and (2) keep the states and inputs within desired safety margins. The optimal control law can then be found using dynamic programming, and the paper shows that the performance of the optimal controller is robust subject to either (or both) of the random and adversarial attack models; meaning that, with high probability, the NCS performs as desired while remaining within safety margins and power requirements.

Unfortunately, it is well known in control theory that optimal controllers are not robust to parameter uncertainty. In other words, if the mathematical model of the physical system is even slightly inaccurate, then the performance of the controller selected may not be as good as expected, and in the worst case, may even destabilize the physical system [22]. As a result, an extension to [2] may consider posing the problem as a robust control problem rather than an optimal control problem.

Replay Attacks

For many NCS's utilizing an observer (as in Figure 3.2), a *detector* can be attached to the observer to flag the operator of the NCS to potential faults and attacks. In the case where the plant and the controller are governed by linear, time-invariant dynamics and the observer is a Kalman Filter, a $\mathcal{X}^2$ fault detector is widely used [29].

A $\mathcal{X}^2$ fault detector leverages well-known properties of the Kalman Filter, specifically that the error between predicted and observed output produced by the Kalman Filter will follow a Gaussian distribution with mean 0 and a known covariance computable by the Kalman Filter. If the actual errors observed over time deviates significantly from this distribution, then the $\mathcal{X}^2$ fault detector triggers an alert to the operator of the NCS [28].

In [29], it is shown that, supposing that the controller is an optimal controller, a replay attack can be used to push the physical system away from a desired steady state into an undesirable state. In this attack, the attacker does not need to know the actual state of the physical system, nor does the attacker need to understand the system dynamics (the paper notes that if the attacker knows this information, she can perform a much more subtle and powerful attack, such as a false data injection attack). However, the attacker does need to be able to modify a subset of the signals y received by the observer and the signals u sent by the controller. Traditional IT security can be used to protect against replay attacks here by ensuring the validity of the data passed through signals y and u. However, it may be difficult to guarantee end-to-end security on all of these links, and if an attacker has cracked the security on some or all of these links, a replay attack becomes possible.

The attack itself is simple; the attacker measures actual output from the system at steady state for a period, and feeds that output into the observer, repeating as necessary. The controller then chooses u according to the now false observer estimates. The it is proven that the repeated output y combined with the controller's choice of u will create the same distribution of errors between predicted and observed output that the original system at steady state did. Therefore, the detector never alerts the operator to a fault. As a result, the

attacker can modify the input into the physical system as much as desired, thus compromising the stability and performance of the system.

To mitigate against such an attack, a controller can add noise taken from a Gaussian distribution to its choice of u as an "authentication signal," which changes the distribution of errors generated by the replay attack, making the replay attack detectable. However, adding this noise makes the controller sub-optimal, meaning that it will require more time and/or energy to drive the physical system to the desired steady state.

False Data Injection Attacks

A false data injection attack, as presented in [24] and [30] is similar to a replay attack. Like a replay attack, the attacker desires to send arbitrary inputs into the plant and hide those inputs from detection from a fault detector (such as the $\mathcal{X}^2$ fault detector) by modifying the outputs observed by the observer. Also like a replay attack, the attacker needs to be able to modify a subset of the signals y and u at will. However, unlike a replay attack, the attacker also needs knowledge about the dynamics governing the plant and the observer.

Given this knowledge, an attacker can generate false measurements y that allows him to feed a constrained set of malicious inputs u into the system which can compromise the stability and performance of the system. The computation to generate y is computationally complex; however, an efficient approximation algorithm to do so is given in [30].

It is more difficult to protect against a false data injection attack than a replay attack. One way to do so, however, is to independently verify the validity of a subset of measurements y before they reach the observer [4].

3.3.2 Stability and Related Vulnerabilities

While there are many notions of stability, for simplicity, we only consider one here. An equilibrium point (or state) of a system is *asymptotically stable* if it converges to that point whenever it starts sufficiently close. For example, a ball that is placed anywhere in a bowl

will roll to the bottom of the bowl, meaning that the bottom of this bowl is an asymptotically stable equilibrium. In contrast, the top of a hill is also an equilibrium since a ball placed there will stay there; however, if the ball is placed even a small distance away from the top (or is bumped away by a small gust of wind), it will roll away. As such, the equilibrium at the top of the hill is unstable. If the system has reached a stable equilibrium and is not moving away, it is considered to be in *steady state*. For simplicity, whenever we say that a system is stable, we imply that a desired equilibrium of the system is stable. The primary purpose of feedback control (such as shown in Figure 3.1) is to create and stabilize equilibria [13].

As such, stability is a required property in most (if not all) NCS's; stability implies that the system is functioning as desired.

Destabilization Attacks

Instability in a NCS takes the form of *cascading failures*. For example, power systems are designed as a network of high-voltage transmission systems, with multiple paths between generators and customers. When one path is removed from the network (resulting from natural disasters, failures, or even attacks), the flow of current shifts nearly instantaneously to parallel paths. However, if a component on one of these parallel paths cannot handle the additional load, it may also fail diverting the combined flow to a new path. These failures can cascade throughout the entire network, causing the entire system to fail [18].

A destabilization attack is an attack on an NCS with the intent to cause a cascading failure. In [35] and [7], vulnerability of a link or a set of links (regardless if the links exist within the physical system, the controller, or along a communication path between them) is defined in terms of the amount of energy an attacker would need to expend on that link to cause a cascading failure. If an infinite amount of energy is required, then that link is secure against any destabilization attack. Otherwise, it is considered to be vulnerable. More precisely, the vulnerability of the link is the inverse of the amount of energy required to destabilize it.

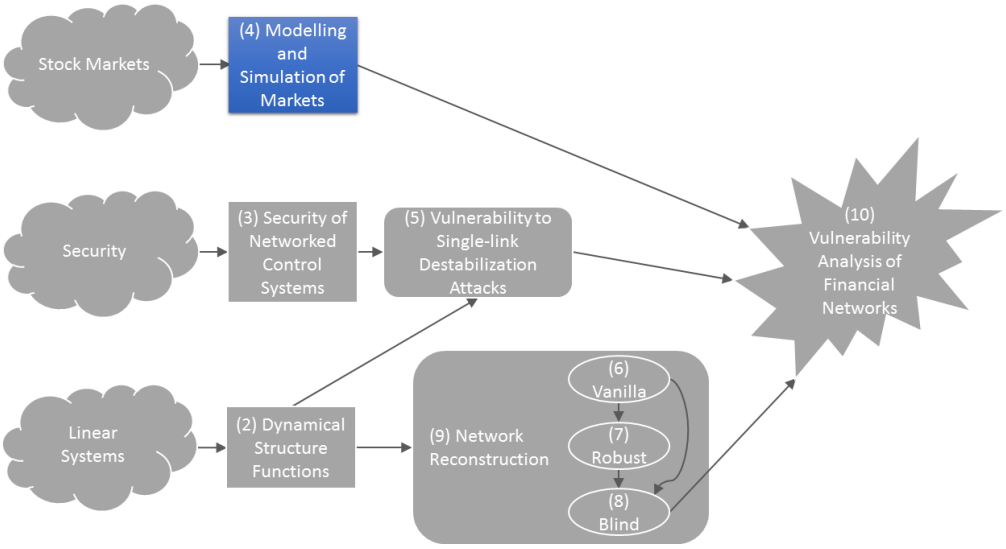
It was then proven that a link is vulnerable if and only if it exists in a cycle (meaning there exists a path through the system from the end of the link back to the beginning). This has immediate consequences on NCS's in the form shown in Figure 3.1, where every link on the communication path between the plant and the controller are, of necessity, in a cycle with each other, and are therefore vulnerable.

In [17], a hypothetical destabilization attack against a river system was explored. It was shown that perturbations on a communication link between a person reading downstream volumes of water and a person controlling an gate that releases water from a reservoir into the river could potentially destabilize the river system, leaving mid-stream farmers without water and potentially causing approximately $70 million in damages.

As the theory of destabilization attacks is used in Chapter 10 to perform a vulnerability analysis on stock market data, Chapter 5 will be devoted to laying the groundwork for analyzing the vulnerabilities of NCS's from the point of view of a destabilization attack.

Chapter 4

Modelling and Simulation of Markets



The purpose of this thesis is to perform a vulnerability analysis on models of the stock market built using data. In this chapter, we provide a brief overview of how the stock market functions, along with a sample of data sets available and which will be used later in this project.

4.1 Overview of the Stock Market

The stock market is a feedback interconnection between a market mechanism–which we will call the matching engine–and traders who buy and sell securities on the market.

As shown in Figure 4.1, traders take the stock price as inputs, along with external information about the market (such as company publications, news sources, social network opinions, etc.). Some traders may also pay to use the full limit order book as an input.

Using these inputs, traders make decisions on securities which they choose to buy and/or sell. These decisions are submitted to the stock market in the form of *limit orders* (see Section 4.2). The market receives these orders and determines how to match buyers with sellers in a fair manner, and then processes the transaction between buyer or seller. We call this market mechanism *the matching engine*. The limit order book (LOB) as well as security prices are computed and published by the matching engine (see Section 4.3).

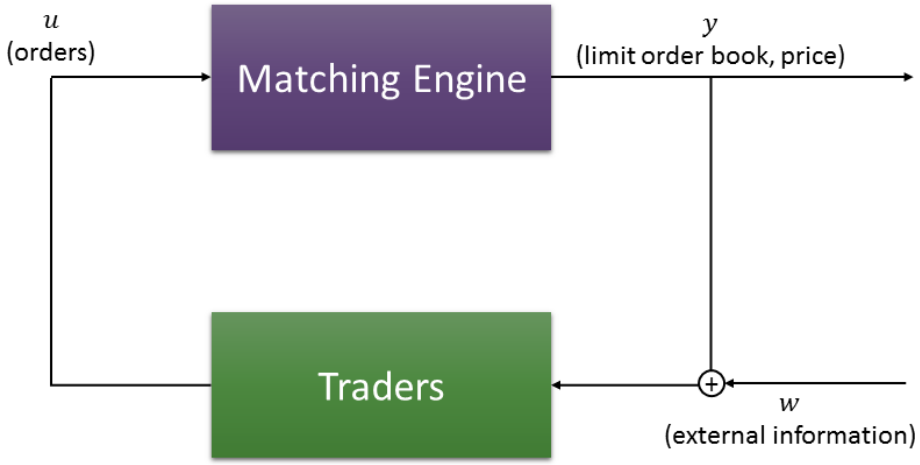


Figure 4.1 Overview of the Stock Market. Traders take stock prices and external information as inputs and submit orders to the market as outputs. A matching engine within the market takes orders and computes prices and the limit order book.

As shown in Figure 4.2, the matching engine computes prices and the LOB for a single security independent of the limit orders submitted for all other securities. However, any trader can submit orders for any set of securities desired. Furthermore, the market treats each trader as anonymous. As such, due to the feedback interconnection with traders, there may exist inter-price dynamic relationships, or in other words, causal dependencies, of the price of one security on the prices of other securities. More on this in Chapter 10.

4.2 Limit Orders

A limit order is essentially an offer to buy or sell a specified quantity of shares of a specific security at a specified price. Orders can also be submitted to modify or cancel existing orders.

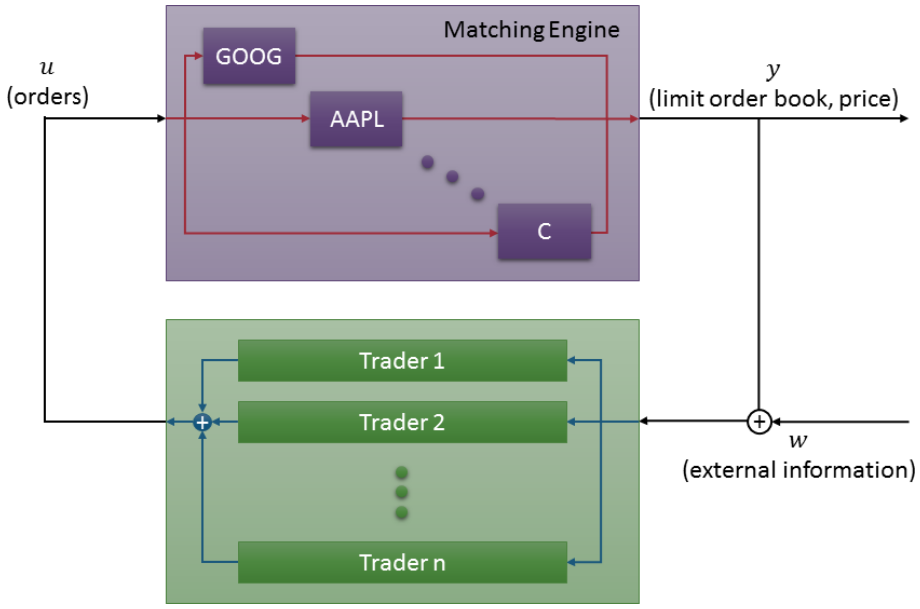


Figure 4.2 A more detailed view of Figure 4.1. Prices within the market engine of one security are made independent of the orders submitted to other securities. Inter-price dynamic relationships only exist because of trader behavior.

Seconds Since Midnight (Eastern Time)	Order Number	Ticker	Action	Side	Quantity	Price
		...				
34201.5799900	7777830	GOOG	Add	Bid (buy)	100	$680.00
34202.8024923	7777830	GOOG	Cancel	Bid (buy)	100	$680.00
34203.0659387	7781190	AAPL	Add	Ask (sell)	400	$100.00
34204.9782172	8020022	C	Add	Ask (sell)	200	$42.42
		...				

Table 4.1 Example of a stream of orders arriving at the Matching Engine.

To further illustrate limit orders, consider the example in Table 4.1. In row 1, an order is submitted at roughly 9:30 AM EST indicating that the trader is willing (and committed to) buy *up to* 100 shares of Google *at or below* $680.00. A second later, in row 2, that same trader submits a second order cancelling the order submitted in row 1, meaning the trader is no longer obligated to buy should the opportunity become available.

A second later, in row 3, another trader submits an order indicating a commitment to sell *up to* 400 shares of AAPL *at or above* $100.00. Note that, although the order number is different than the order number for the order in the first row, this very well may be the same

trader who submitted that first order. Or it may be a different trader. Order numbers are tracked so that orders can be eventually modified or deleted. Orders, however, are submitted anonymously.

Finally, a second later, another order is submitted indicating a commitment to sell up to 200 shares of Citigroup at or above $42.42. And the process continues throughout the day.

NASDAQ and several other exchanges are open from 9:30 AM to 4:00 PM EST, and the vast majority of orders are submitted within this window. However, some orders are also accepted before and after this time range.

4.3 The Matching Engine, the Limit Order Book, and Prices

The Matching Engine is a market mechanism employed by several stock exchanges to track incoming orders and to execute fair trades as soon as they become available. The state of the Matching Engine is known as the Limit Order Book (LOB), which keeps track of all outstanding commitments to buy and sell. Every security has its own LOB, computed only with limit orders submitted for that security, meaning its LOB is independent from the LOB of all other securities.

To describe the dynamics of the Matching Engine, we proceed with an example. Consider a security EXPL which takes prices $1.00, $2.00, ..., $10.00. Let $x[t] \in \mathbb{Z}^10$ represent the quantity of orders on the LOB at time t for each of these prices, with positive indicating an offer to buy and negative indicating an offer to sell. For example, let

$$x[t]^\mathsf{T} = \begin{bmatrix} 0 & 3 & 5 & 1 & 0 & 0 & -5 & -2 & 0 & 0 \end{bmatrix}.$$

This is interpreted as one or more offers to buy a combined quantity of 3 shares of EXPL at $2.00, offers to buy 5 share of EXPL at $3.00, and an offer to buy 1 share of EXPL at $5.00. There are also offers to sell 5 shares of EXPL at $7.00 and 2 shares of EXPL at $8.00.

At the beginning of the day, the LOB always starts empty (if there are any outstanding orders still on the LOB at the end of the previous day, they are automatically cancelled). Therefore, $x[0] = 0$. Suppose that at time $t = 1$, a limit order arrives offering to sell 7 shares of EXPL at \$9.00. No trades are executed at this time as there are currently no offers to buy. As such, all 7 shares are placed on the LOB, giving

$$x[1]^\intercal = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -7 & 0 \end{bmatrix}.$$

Now suppose that at time $t = 2$, another order arrives to buy 5 shares at \$1.00. Since the first seller is only willing to sell at \$7.00 or greater and this buyer is only willing to buy at \$1.00 or lower, no trade is executed at this time and this second order goes straight to the LOB, resulting in

$$x[2]^\intercal = \begin{bmatrix} 0 & 5 & 0 & 0 & 0 & 0 & 0 & 0 & -7 & 0 \end{bmatrix}.$$

The gap between the best (highest) buy order (bid) and the best (lowest) sell order (ask) is known as the *bid-ask spread*. Presently, the bid-ask spread is $\$9.00 - \$1.00 = \$8.00$.

For securities with higher liquidity, the bid-ask spread tends to be small, where securities with lower liquidity have higher bid-ask spreads. Also, the spreads at the beginning of the day tend to be larger, though they narrow quickly as more orders are added to the book.

Frequently, the price of a security is computed based on the present bid-ask spread. Sometimes the highest bid–called the *bid price*–is reported, sometimes the lowest ask–called the *ask price*. The last price at which a trade was successfully executed–called the *last price*–is often also reported as the price of the security. Generally, all three are reported, and it is up to the user's discretion which to use as the price. From this point forward, we will use the last price as the price of the security.

Let us fast forward a bit in the future, and suppose that at time $t = 50$, the LOB is

$$x[50]^{\mathsf{T}} = \begin{bmatrix} 5 & 2 & 3 & 15 & 7 & -8 & -12 & -1 & -7 & -5 \end{bmatrix}.$$

Now, the bid-ask spread has narrowed considerably to $\$6.00 - \$5.00 = \$1.00$.

Suppose that at time $t = 51$, a new order arrives to buy 4 shares of EXPL at \$6.00. On the LOB, there are orders to sell up to 8 shares at or above \$6.00. Since this buyer is willing to buy at the same price, a match can be made and an order executed. There are enough sell orders at this price to fully execute the trade, and the number of sell orders decreases from 8 down to 4, giving

$$x[51]^{\mathsf{T}} = \begin{bmatrix} 5 & 2 & 3 & 15 & 7 & -4 & -12 & -1 & -7 & -5 \end{bmatrix}.$$

The bid-ask spread has not changed. However, since a trade has executed, the price $y(t)$ of EXPL has moved from where it was previously to $y(51) = \$6.00$. Note that if there were multiple orders to sell at \$6.00, the Matching Engine will prioritize the order that was submitted first to be executed first.

Now suppose that at time $t = 60$ an order is submitted to sell 16 at \$4.00. The LOB currently has traders willing to buy at \$5.00. The Matching Engine is legally required to give the seller the best trade available, and so it will favor those traders first, trading all 7 shares at to the seller at \$5.00, a better price than the seller was offering. However, the seller has 9 more shares she is willing to sell, and there are buyers willing to buy at \$4.00, and so 9 shares are executed at that price as well. This gives

$$x[60]^{\mathsf{T}} = \begin{bmatrix} 5 & 2 & 3 & 6 & 0 & -4 & -12 & -1 & -7 & -5 \end{bmatrix}.$$

The bid-ask spread has increased to $\$6.00 - \$4.00 = \$2.00$ and the price is now $p(60) = \$4.00$. The process of a trade being executed at multiple price points such as that above is known as *walking the book*.

Let an order be submitted at time $t = 65$ to buy 7 shares at \$5.00. There are no offers to sell at this price, so the trade is added directly to the LOB, giving

$$x[65]^{\mathsf{T}} = \begin{bmatrix} 5 & 2 & 3 & 6 & 7 & -4 & -12 & -1 & -7 & -5 \end{bmatrix}.$$

Since no trade was executed, the price remains at $p(65) = \$4.00$. However, the bid-ask spread has narrowed to $\$6.00 - \$5.00 = \$1.00$.

Finally, let an order be submitted at $t = 70$ to sell 12 shares at \$5.00. There are offers to buy up to 7 shares at this price, but no more. Therefore, 7 shares will be traded between buyer and seller, and the remaining 5 shares in this order will be added to the LOB, giving

$$x[65]^{\mathsf{T}} = \begin{bmatrix} 5 & 2 & 3 & 6 & -5 & -4 & -12 & -1 & -7 & -5 \end{bmatrix}.$$

The price is now $p(70) = \$5.00$ and the bid-ask spread is $\$5.00 - \$4.00 = \$1.00$.

For actual securities, prices and time scales exist at much higher resolution (prices can be added on the cent, not on the dollar, and orders can be recorded at a nanosecond resolution). Nonetheless, the matching engine still follows the same dynamics as demonstrated above. Figures 4.3 and 4.4 give examples of actual limit order books generated from the ITCH data (discussed in the next section).

4.4 Data Sources

We now give a brief tour of available sources of financial data. While there are many sources available, we consider only three in this work, namely the ITCH Data, Yahoo Finance, and the Tour de Finance (TDF).

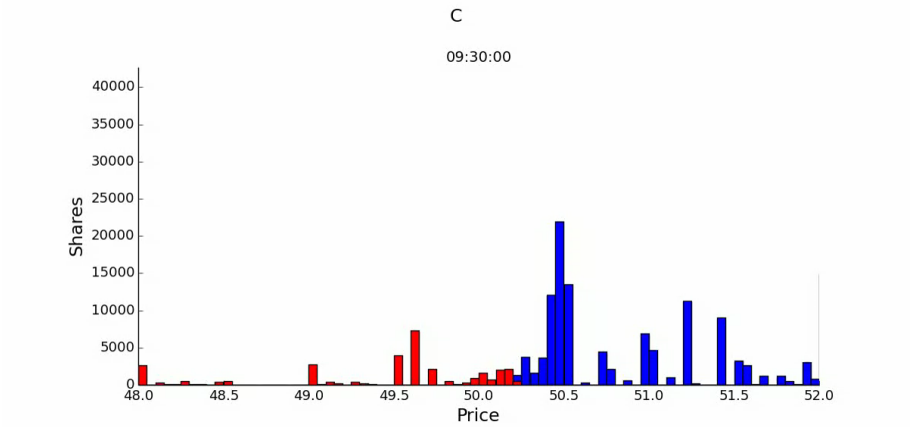


Figure 4.3 The limit order book of C (Citigroup) on 3/7/14 at 9:30 AM EST. Prices are binned so that every bar contains all limit orders across 5 cents, instead of showing the limit orders at each cent. Red bars are buy (bid) orders and Blue bars are sell (ask) orders.

4.4.1 The ITCH Data

The ITCH[1] data is a data feed offered by NASDAQ[2] for a fee containing every change made to the order book [31]. When parsed, this data takes a form similar to Table 4.1, but with one subtle difference. Submitted limit orders that are marketable (i.e. can be executed immediately) are not recorded. Likewise, no market order (a limit order where all shares are to be executed at the best price available) is recorded. Only the results of the change to the order book are recorded.

Since the ITCH data contains all changes to the order book ever made, it can be used to compute the price $y(t)$ (as well as the LOB) of any security on NASDAQ at *any* time resolution.

The ITCH data contains roughly 5 GB of data of compressed binary data per day across all 8,344 securities traded on the NASDAQ exchange. If converted into a human readable CSV, each day fills about 20 GB of data. The ticker with the smallest CSV typically requires 50 KB / day representing roughly 150 orders, whereas the largest requires 150 MB / day representing roughly 3 million orders.

[1] ITCH is not known to represent any acronym.
[2] Other exchanges offer similar services

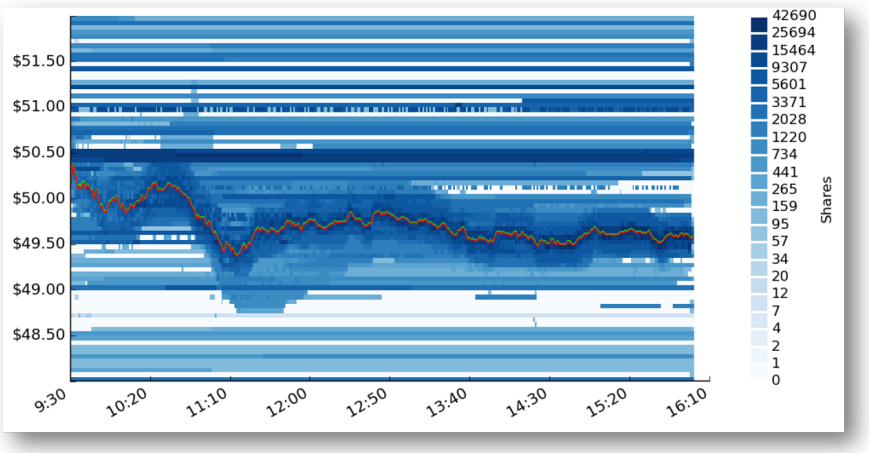


Figure 4.4 The limit order book of C (Citigroup) across the day on 3/7/14. Figure 4.3 is a cross section of the left edge of this image. The red line is the bid price over time, and all pixels below the red line represent buy (bid) orders. The green line is the ask price over time, and all pixels above the red line represent sell (ask) orders. Orders are binned in both time and price.

4.4.2 Yahoo Finance

Yahoo Finance is a web utility offered gratuitously by Yahoo which reports prices of most securities nearly in real time [48]. These prices are subject to network and processing delays between NASDAQ and other exchanges and Yahoo.

Yahoo Finance also provides a utility to query historical data. However, this service only allows historic daily closing prices of securities; if higher resolutions of data are desired, other data sources will be needed.

4.4.3 The Tour de Finance

The Tour de Finance (TDF) was built by the author of this work as a platform to test and compete algorithmic traders against each other, where an algorithmic trader is a program that autonomously reads historic stock prices and other information and makes decisions on a portfolio in which to invest.

Every minute that NASDAQ is open, TDF queries Yahoo Finance for the current prices of every security on the S&P 500 and saves those prices in a database. Traders–or agents as they are called in TDF–are initially given $\$100,000$ of paper money and can invest that cash across any or all of the securities in the S&P 500, buying securities at the current best ask price and selling at the current best bid. As the TDF tracks changes in prices among the securities, it also updates the values of each agent's portfolios, computed as the cash the agent would receive if it immediately sold off all securities. Participants can also write their trading algorithms using any language and any platform they desire, connecting to TDF through a secure RESTful API.

Since TDF uses real-time data, agents are prevented from "cheating" by using future information to make present decisions. Because of this and other features implemented in TDF, it is one of the most realistic open-source paper-trading platforms in existence today.

The TDF has been successfully been used as a class project in a Senior-Level Computer Science Course (Linear Programming and Convex Optimization) to teach the principles of learning and decision making under uncertainty. The class of approximately 30 students was divided into 8 teams, each of which invented and tested their own trading algorithm. Several of the teams were successful in earning money over a three-week period that not only included a holiday (Thanksgiving), but also an election. One team earned an 8.6% return over this period, whereas the S&P 500 index only increased by about 4.6% over the same horizon.

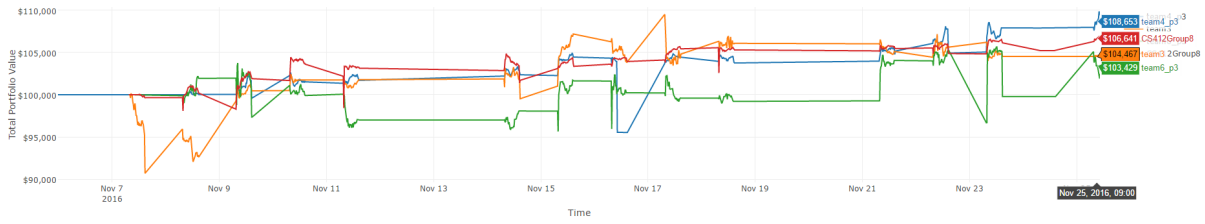
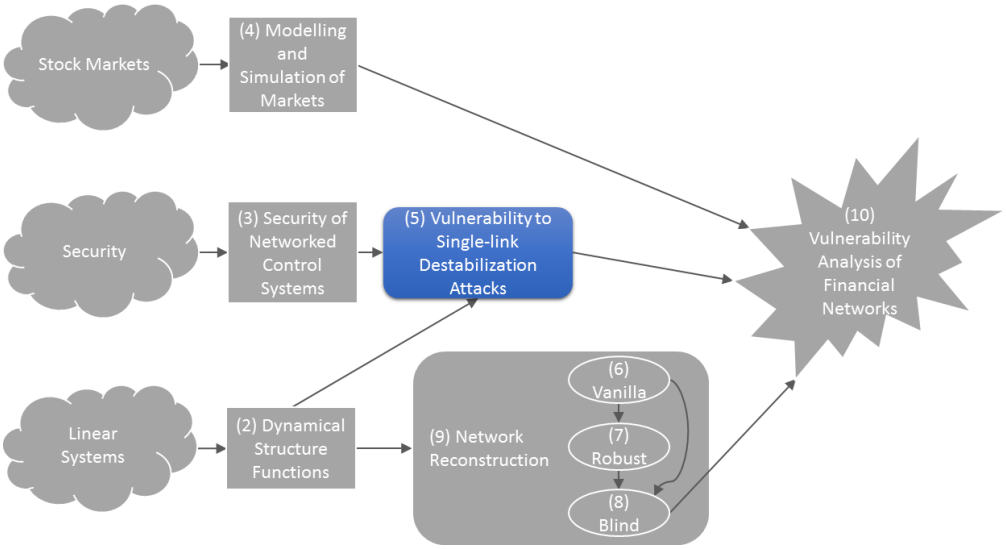


Figure 4.5 Performance of the top four teams in the Fall 2016 Linear Programming and Convex Optimization Tour de Finance Competition.

For the purposes of this project, the TDF is used a data collection and storage system which allows us to collect and utilize minute-resolution stock data.

Chapter 5

Vulnerability to Single-Link Destabilization Attacks



One objective in this thesis is to perform the vulnerability analysis presented in [8, 35] on financial networks. As the results of those papers are critical to the understanding of this work, we include them here.

5.1 Problem Formulation

Let

$$\begin{aligned} x[k+1] &= Ax[k] + Bu[k] \\ y[k] &= \begin{bmatrix} I & 0 \end{bmatrix} x[k] \end{aligned} \tag{5.1}$$

represent the state space representation of the dynamics of the system for which we are performing the vulnerability analysis[1]. We model an attack on this network system as an additive external disturbance $F\psi[k]$ to the system, where

$$\begin{aligned} x[k+1] &= Ax[k] + Bu[k] + F\psi[k] \\ y[k] &= \begin{bmatrix} I & 0 \end{bmatrix} x[k] \end{aligned} \tag{5.2}$$

Let the purpose of the attack be to destabilize the system. If we restrict ourselves such that (5.1) is a stable system, then any bounded input into the system will produce a bounded output. Or in other words, if $\psi[k]$ is treated as an external disturbance, then it is impossible for the attack $F\psi[k]$ to destabilize the system.

However, if $\psi[k] = x[k]$, or in other words, if the attacker uses information about the state of the system to formulate an attack, then it may be possible to destabilize the system (any choice of F that causes $A + F$ to have poles outside the unit circle is sufficient to destabilize the system). As such, we will only consider the situation where $\psi[k] = x[k]$.

We now transform the system into a DSF. Following the procedure in Chapter 2, we rewrite (5.2) to be of the form

$$\begin{aligned} \begin{bmatrix} y[k+1] \\ \upsilon[k+1] \end{bmatrix} &= \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} y[k] \\ \upsilon[k] \end{bmatrix} + \begin{bmatrix} B_1 \\ B_2 \end{bmatrix} u[k] + \begin{bmatrix} F_1 \\ F_2 \end{bmatrix} \psi[k] \\ y[k] &= \begin{bmatrix} I & 0 \end{bmatrix} \begin{bmatrix} y[k] \\ \upsilon[k] \end{bmatrix} \end{aligned} \tag{5.3}$$

[1] Note that, as explained in Chapter 2, so long as $rank(C) = p$ where $y[k] \in \mathbb{R}^p$, we can transform any system with output $\hat{y}[k] = Cx[k]$ into the form (5.1).

Taking the $\mathcal{Z}$-transform of (5.3) yields

$$\begin{bmatrix} zY(z) \\ z\Upsilon(z) \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} Y(z) \\ \Upsilon(z) \end{bmatrix} + \begin{bmatrix} B_1 \\ B_2 \end{bmatrix} U(z) + \begin{bmatrix} F_1 \\ F_2 \end{bmatrix} \Psi(z), \tag{5.4}$$

where $Y(z)$, $\Upsilon(z)$, and $\Psi(z)$ are the $\mathcal{Z}$-transforms of $y[k]$, $\upsilon[k]$, and $\psi[k]$ respectively.

Solving for $\Upsilon(z)$ in the second row of (5.4) gives

$$\Upsilon(z) = (zI - A_{22})^{-1}A_{21}Y(z) + (zI - A_{22})^{-1}B_2U(z) + (zI - A_{22})^{-1}F_2\Psi(z). \tag{5.5}$$

Let

$$W(z) = A_{11} + A_{12}(zI - A_{22})^{-1}A_{21}, \tag{5.6}$$

$$V(z) = B_1 + A_{12}(zI - A_{22})^{-1}A_{21}B_2, \tag{5.7}$$

$$N(z) = F_1 + A_{12}(zI - A_{22})^{-1}A_{21}F_2. \tag{5.8}$$

Then plugging (5.5) into the first row of (5.4) yields

$$zY(z) = W(z)Y(z) + V(z)U(z) + N(z)\Psi(z). \tag{5.9}$$

Finally, defining $D(z) = diag(W_{11}(z), \ldots, W_{pp}(z))$ and subtracting DY from (5.9), we get

$$Y(z) = Q(z)Y(z) + P(z)U(z) + \Delta(z)\Psi(z), \tag{5.10}$$

$$Q(z) = (zI - D(z))^{-1}(W(z) - D(Z)), \tag{5.11}$$

$$P(z) = (zI - D(z))^{-1}V(z), \tag{5.12}$$

$$\Delta(z) = (zI - D(z))^{-1}N(z). \tag{5.13}$$

Equation (5.10) represents a generalized destabilization attack model in the DSF domain. However, for this work, we are only concerned with the vulnerability of this system

to single-link attacks, a problem which we formalize in Problem 1. We then proceed to solve the problem in Section 5.2

Problem 1 (Vulnerability Analysis of Networks to Single-Link Attacks)**.** *Let exactly one entry $\Delta_{ij}(z)$, $i \neq j$, in $\Delta(z)$ be non-zero. Given the generalized attack model (5.10), compute the vulnerability v_{ij}, defined as the inverse[2] of the minimum magnitude $\|\Delta_{ij}(z)\|_\infty$ of an additive perturbation on link $Q_{ij}(z)$ (which may or may not be zero[3]) required to destabilize the entire system.*

5.2 Solution

To find the minimum magnitude $\|\Delta_{ij}\|_\infty$ required to destabilize the system, we leverage the Small Gain Theorem, which states that the system will remain stable as long as $\|\Delta_{ij}(z)\|_\infty \|M_{ij}(z)\|_\infty < 1$, where $M_{ij}(z)$ is the closed-loop transfer function seen by $\Delta_{ij}(z)$. Therefore, the smallest perturbation necessary to destabilize the system occurs when

$$\|\Delta_{ij}(z)\|_\infty \|M_{ij}(z)\|_\infty = 1, \tag{5.14}$$

or in other words,

$$\|\Delta_{ij}(z)\|_\infty = \frac{1}{\|M_{ij}(z)\|_\infty}. \tag{5.15}$$

Since the vulnerability of this link is the inverse of the quantity (5.15), we compute the vulnerability of link $Q_{ij}(z)$ as

$$v_{ij} = \|M_{ij}(z)\|_\infty. \tag{5.16}$$

The challenge now is to compute $M_{ij}(z)$.

[2] The smaller the vulnerability, the more "effort" needed to destabilize the system, hence the inverse.

[3] Previous work on vulnerability requires that the attacker take advantage of existing infrastructure in the system, and so a link is considered to be invulnerable if the link doesn't actually exist. In other words, $v_{ij} = 0$ when $Q_{ij} = 0$. In this work, however, we assume that it is easy for an attacker to create a new link where one doesn't exist (more discussion on this in Chapter 10), as such we do not artificially force the vulnerability of a link to zero if $Q_{ij}(z) = 0$.

Starting with (5.10), solve for $Y(z)$ in terms of $U(z)$ and $\Psi(z)$ to get

$$Y(z) = (I - Q(z))^{-1}P(z)U(z) + (I - Q(z))^{-1}\Delta(z)\Psi(z). \tag{5.17}$$

The relationship of inputs to outputs is given by $G(z) = (I - Q(z))^{-1}P(z)$ and the transfer function describing how $\Psi(z)$ affects the outputs is given by $(I - Q(z))^{-1}\Delta(z)$. As discussed previously, since no bounded external input can destabilize the system, we only consider the case where $\Psi(z) = Y(z)$ (which is equivalent to $\psi[k] = \begin{bmatrix} x_1[k] & \dots & x_p[k] \end{bmatrix}^\mathsf{T} = y[k]$)[4].

Since $\Delta(z)$ is zero everywhere except at $\Delta_{ij}(z)$, then the transfer function seen by the perturbation $\Delta_{ij}(z)$ on link $Q_{ij}(z)$ is found from (ignoring inputs $U(z)$):

$$\begin{bmatrix} Y_1 \\ \vdots \\ Y_j \\ \vdots Y_p \end{bmatrix} = H(z) \begin{bmatrix} 0 \\ \vdots \\ \Delta_{ij}(z)Y_j(z) \\ \vdots \\ 0 \end{bmatrix}, \tag{5.18}$$

$$H(z) = (I - Q(z))^{-1}. \tag{5.19}$$

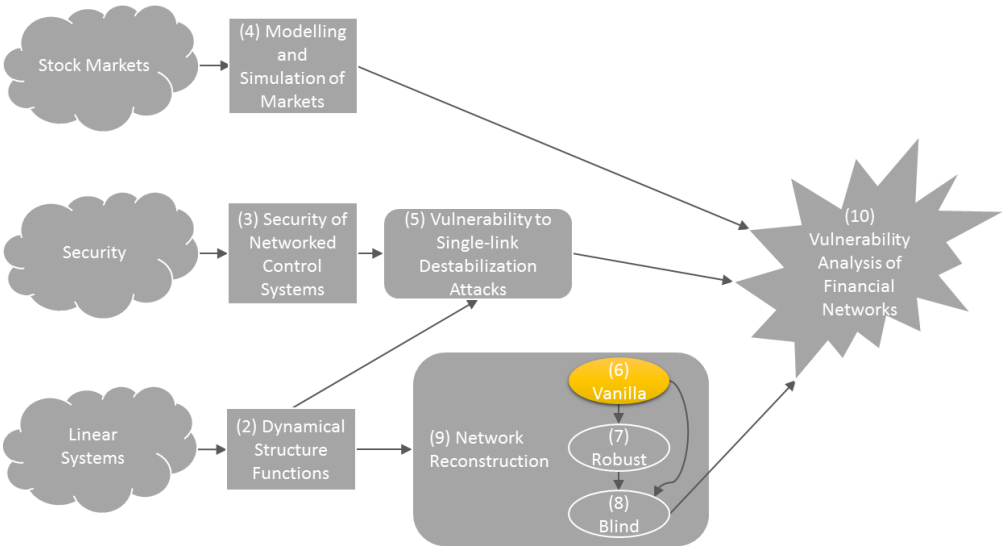
From this, we can see that $Y_j(z) = H_{ji}(z)\Delta_{ij}(z)Y_j(z)$, therefore the transfer function seen by the perturbation $\Delta_{ij}(z)$ is $M_{ij}(z) = H_{ji}(z)$. Therefore, we can precisely define the vulnerability of any link $Q_{ij}(z)$ as

$$v_{ij} = \|H_{ji}(z)\|_\infty = \| \left[(I - Q(z))^{-1}\right]_{ji} \|_\infty. \tag{5.20}$$

[4] Note that since $Q(z)$ and $\Delta(z)$ are both $p \times p$ matrices of rational transfer functions, attack $\Delta(z)$–with $\delta_{ij}(z)$, $i \neq j$–represents an additive perturbation of link (i, j) in $Q(z)$.

Chapter 6

Vanilla Passive Network Reconstruction



Passive network reconstruction is the process of identifying a structured representation of a system using observed (as opposed to controlled) input-output data and some additional information about the structure of a system. In this chapter, we improve on the existing algorithm for the passive reconstruction of dynamical structure functions through the use of an evolutionary algorithm. We also demonstrate the convergence of this algorithm to the proper network as input-output data is received over time.

This chapter is adapted from [47], which was submitted to the 2017 IEEE Conference on Decision and Control as a result of this thesis. In addition to the results presented in [47], this chapter also includes connections to the vulnerability analysis discussed in Chapter 5.

6.1 Introduction - Network Reconstruction

System Identification is the process of recovering a "black box" representation of a system–such as a transfer function–from input-output data. Network reconstruction takes this process one step further and attempts to find a structured representation of the system with more structural information than the input-output behavior.

Unfortunately, network reconstruction on input-output data is an ill-posed problem since input-output data, by itself, does not have enough information to build anything more detailed than a black box model. Therefore, network reconstruction requires additional a priori information about the structure of the system to recover the system dynamics along with a stronger notion of its structure.

If we were to recover the state space representation of a system from input-output data, the amount of structural information necessary to perform the reconstruction is large, and can become prohibitively large in many real-world applications. Therefore, we are instead concerned with the reconstruction of a signal structure representation of the system–specifically the dynamical structure function–which requires significantly less structural information to be reconstructed.

The majority of work in the network reconstruction literature focuses on reconstructing a system using frequency domain representations, such as the dynamical structure function, linear dynamical graphs, or directed information graphs [16, 27, 34]. In this chapter, we consider passive network reconstruction of a time domain representation of the dynamical structure function as opposed to the frequency domain.

Reconstruction algorithms of the dynamical structure function were originally developed for active reconstruction in areas where experiments on a system are possible, such as biochemical reaction networks [1, 49]. Passive reconstruction is applicable to a much larger range of systems since we do not need to be able to affect the network to learn its structure, e.g. financial systems or social media influence networks [11].

Reconstructing networks in the time domain allows system structure to be determined directly from data, without requiring a transformation to the frequency domain which could introduce errors due to approximation methods or noisy data. Moreover, reconstruction directly from data is more meaningful than through a transformation–the understanding of the structure and dynamics of a network can be learned incrementally over time, as opposed to other network reconstruction techniques which only give a picture of the network once all the data has been collected.

For instance, Figure 6.1 shows how the sparsity pattern and dynamics of the system can slowly be learned over time as more data is collected from the system. After five time steps, the reconstructed network appears fully connected. As more data is collected, we see after 100 time steps that the weight of the connections (i.e. the infinity norm of the transfer functions on the associated links) appear closer to the final network structure. Finally, after 500 time steps, we have gathered enough data to determine the correct sparsity structure and dynamics of the system.

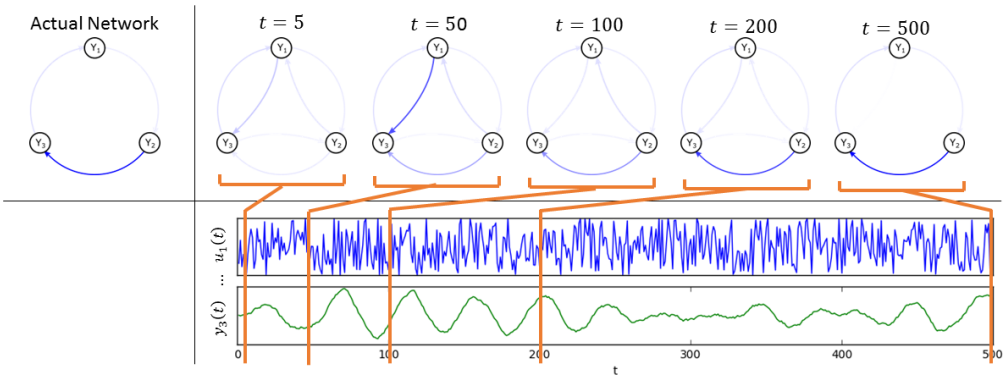


Figure 6.1 As data streams in, passive network reconstruction can build a more accurate representation of the network. Once there is enough data, then network reconstruction can recover the network perfectly. Note that the network graphs were generated using the same process as described in Figure 6.4.

Passive reconstruction of the dynamical structure function in the time domain was first developed in [10] and this chapter builds directly off the work presented in that paper. A key difference is the use of an evolutionary algorithm to fit the convolutional models (step 8

of the algorithm presented in Section 6.3). This change has provided the following significant advantages:

- The full dynamics of this network can be reconstructed with much less data (the example in Section 6.5 of this paper can now be reconstructed with with $r = 100$, whereas the previous algorithm required $r \geq 600$, where r is related to the amount of data required to be collected)
- The original algorithm required an initial guess of the parameters of the convolutional model, whereas the evolutionary algorithm does not require an initial guess.
- The original algorithm would only converge if an appropriate initial guess of the parameters of the convolutional model were chosen, which was often hard to do correctly. The evolutionary algorithm always converges, and if the necessary and sufficient conditions for reconstruction are met, it converges to the correct answer.

Furthermore, the original work indicated that an $r \geq T$ can be used (where T is the actual amount of data collected and r is at least the amount of time required for the impulse responses of all links in the network converge to zero), but the original paper only covered in detail the case where $r = T$. This chapter clarifies how to perform network reconstruction in the general case where $r \geq T$. Furthermore, in Section 6.6, this work demonstrates the convergence of the algorithm to the actual network being reconstructed as both r and T increase, showing that r should be chosen to be strictly larger than T for the algorithm to converge to the true network.

6.2 Problem Formulation

The basic problem being solved by passive network reconstruction is to take observed input-output data from a system and some a priori understanding of the structure of the system and recover the unique DSF that best fits that data.

Formally, consider a dynamic system taking m inputs and producing p outputs according to the following relationship:

$$\begin{aligned} x(t+1) &= Ax(k) + Bu(t), \\ y(t) &= Cx(t), \end{aligned} \tag{6.1}$$

where $u(t) \in \mathbb{R}^m$ and $y(t) \in \mathbb{R}^p$ are measured, but where $A \in \mathbb{R}^{n \times n}$, $B \in \mathbb{R}^{n \times m}$, $C \in \mathbb{R}^{p \times n}$, and $x(t) \in \mathbb{R}^n$ are unknown.

Let $\mathcal{D}_u \in \mathbb{R}^{T \times m}$ be the time-domain measured values[1] of the m inputs into this system over times $1, 2, \ldots, T$. Likewise, let $\mathcal{D}_y \in \mathbb{R}^{T \times p}$ be the measured values of the p outputs of the system at times $1, 2, \ldots, T$ generated by inputs $\mathcal{D}_u$. Define $\tilde{u}(t) \in \mathbb{R}^m$ to be the t'th row of $\mathcal{D}_u$ and define $\tilde{y}(t) \in \mathbb{R}^p$ to be the t'th row of $\mathcal{D}_y$. Let $\mathcal{D}_u$ and $\mathcal{D}_y$ be measured without error.

Furthermore, let $(Q(z), P(z))$ be the unknown, but unique Dynamical Structure Function (DSF) generated from (6.1) as described in Chapter 2, and assume that every link in $Q(z)$ and $P(z)$ are stable and strictly proper. This DSF maps inputs $U(z)$ in the frequency domain to outputs $Y(z)$, also in the frequency domain according to the following relationship:

$$Y(z) = Q(z)Y(z) + P(z)U(z). \tag{6.2}$$

Let

$$y(t) = Q(t) * y(t) + P(t) * u(t) \tag{6.3}$$

be the convolutional representation of the system's DSF found by taking the inverse $\mathcal{Z}$-transform of (6.2), where $*$ is the convolution operator.

[1] The term *passive* in the passive reconstruction problem comes from the idea that inputs are measured but not controlled.

Due to the fact that this is an LTI, strictly causal, and stable network, (6.3) can be rewritten as

$$\hat{y} = \hat{L}\vec{x} + e, \tag{6.4}$$

where $\hat{y} \in \mathbb{R}^{(T-1)p}$ contains only known values from $\mathcal{D}_y$, $\hat{L} \in \mathbb{R}^{(T-1)p \times r(p^2+pm)}$ contains only known values from $\mathcal{D}_y$ and $\mathcal{D}_u$, and $\vec{x} \in \mathbb{R}^{r(p^2+pm)}$ contains the unknown values of $Q(t)$ and $P(t)$ that need to be reconstructed. Vector $e \in \mathbb{R}^{r(p^2+pm)}$ represents the error in mapping any choice of $\vec{x}$ to $\hat{y}$.

And, as will be discussed later in this work, the impulse response of every link $Q_{ij}(t)$ can also be given by the following function:

$$Q_{ij}(t) = a_{q_{ij}}\delta_{(t,0)} + \sum_{n=0}^{w_{q_{ij}}} b_{n,q_{ij}}(c_{n,q_{ij}})^t, \tag{6.5}$$

where $w_{q_{ij}}$ is the number of delays in the corresponding link, $a_{q_{ij}} = \sum_{n=0}^{w_{q_{ij}}} b_{n,q_{ij}}$, and $\delta_{t,0}$ is the Kronecker delta (with value of 1 at $t = 0$ and 0 otherwise).

The first piece of the network reconstruction problem on stable discrete-time LTI systems, therefore, is to choose the *unique* to minimize the error e generated from mapping $\vec{x}$ to $\hat{y}$. Formally, choose the unique $\vec{x}^*$ such that

$$\vec{x}^* = \arg\min_{\vec{x}} \|e\|_2 = \arg\min_{\vec{x}} \|\hat{y} - \hat{L}\hat{x}\|_2. \tag{6.6}$$

Let $\tilde{Q}_{ij}(t)$ be the element of $\hat{x}^*$ corresponding to link (i, j) in Q at time t. Then the second piece of the network reconstruction problem on stable discrete-time LTI systems is to choose a function $Q_{ij}(t)$ of the form (6.5) such that

$$\epsilon_{ij}(t) = |\tilde{Q}_{ij}(t) - Q_{ij}(t)| \tag{6.7}$$

is minimized (and likewise for P).

6.3 The Vanilla Passive Network Reconstruction Algorithm

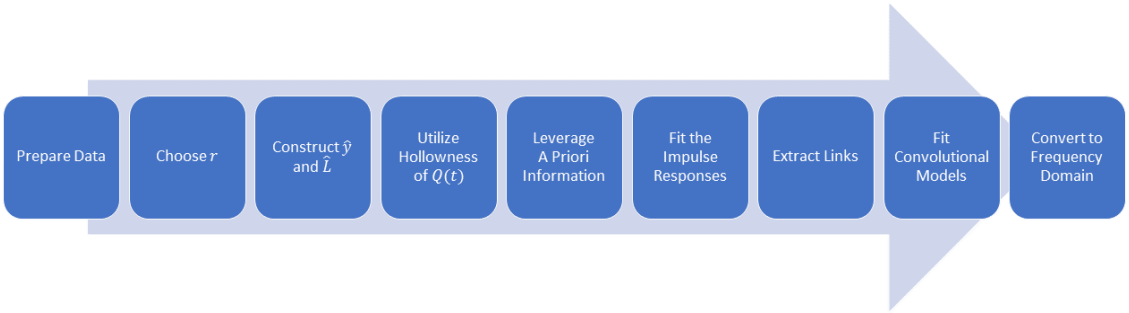


Figure 6.2 The Vanilla Passive Network Reconstruction Algorithm.

In this section, we outline the algorithm–which we call the Vanilla Passive Network Reconstruction (VPNR) Algorithm–for solving passive reconstruction of stable discrete-time LTI networks. This algorithm was originally outlined in [10], though we also propose a few enhancements to this algorithm that have had led to major performance and usability improvements on the algorithm.

The purpose of the VPNR algorithm is to take input data, output data, and some a priori knowledge about the system structure (such as entries in $P(z)$ that are known to be zero) and reconstruct $Q(z)$ and $P(z)$ as exactly as possible using this information only. To do so, the VPNR is broken up into a nine-step process, summarized in Figure 6.2 and outlined below.

1. *Prepare the Data:* Let $\mathcal{D}_u \in \mathbb{R}^{T\times m}$ and $\mathcal{D}_y \in \mathbb{R}^{T\times p}$ be the input and output data respectively as defined as in the problem formulation.

2. *Choose r:* Since the data is represented in the time domain, we will concentrate most of our efforts on the time domain representation of the DSF as expressed in (6.3). By nature of the inverse $\mathcal{Z}$-transform, the convolutional model is the impulse response of the system. In other words,

$$Q_{ij}(-\infty),\ldots,Q_{ij}(-1),Q_{ij}(0),Q_{ij}(1),\ldots,Q_{ij}(r),\ldots,Q_{ij}(\infty) \tag{6.8}$$

are the outputs of link (i, j) resulting at times $-\infty, \ldots, -1, 0, 1, \ldots, r, \ldots, \infty$ from an impulse input into that link–though not necessarily an input into the system–of magnitude 1 at time 0, and likewise for each $P_{ij}(t)$.

Since (6.1) is a linear, time invariant (LTI) system, the impulse at time $t = 0$ cannot create a non-zero output at any time $t < 0$. Furthermore, we have assumed that the dynamics on every link are strictly causal. This means that the impulse input at time $t = 0$ cannot create a non-zero output until some time $t > 0$. We have also assumed that every link is stable, which means that there exists some finite r such that the output resulting from the impulse is 0 for every time $t > r$. Due to these three facts, we can truncate the impulse response (6.8) to

$$Q_{ij}(1), \ldots, Q_{ij}(r) \tag{6.9}$$

with $Q_{ij}(t)$ for $t \leq 0, t > r$ understood to be 0 (and likewise for P).

As such, we choose a value $r \leq T$ large enough that all links in $P(t)$ and $Q(t)$ have time to converge. Since we do not know how large this must be before performing reconstruction, this step may require some trial and error.

3. *Construct $\hat{y}$ and $\hat{L}$ from Data:* By definition of convolution, and utilizing the fact that $Q(t) = 0$ and $P(t) = 0$ for $t \leq 0$ and $t > r$, (6.3) can be expanded and expressed using the following equations:

$$
\begin{aligned}
y(1) &= 0, \\
y(2) &= \begin{bmatrix} Q(1) & P(1) \end{bmatrix} \begin{bmatrix} y(1) \\ u(1) \end{bmatrix}, \\
y(3) &= \begin{bmatrix} Q(2) & Q(1) & P(2) & P(1) \end{bmatrix} \begin{bmatrix} y(1) \\ y(2) \\ u(1) \\ u(2) \end{bmatrix}, \\
&\vdots \\
y(r+1) &= \begin{bmatrix} Q(r) & \cdots & Q(1) & P(r) & \cdots & P(1) \end{bmatrix} \begin{bmatrix} y(1) \\ \vdots \\ y(r) \\ u(1) \\ \vdots \\ u(r) \end{bmatrix}, \\
&\vdots \\
y(T) &= \begin{bmatrix} 0 & \cdots & Q(r) & \cdots & Q(1) & 0 & \cdots & P(r) & \cdots & P(1) \end{bmatrix} \begin{bmatrix} y(1) \\ \vdots \\ y(T-1) \\ u(1) \\ \vdots \\ u(T-1) \end{bmatrix}.
\end{aligned}
$$

These equations can be rewritten using matrix multiplication giving the following equations (which are the Toeplitz representation of the network):

$$\bar{y}(T) = \bar{Q}(r)\bar{y}(T-1) + \bar{P}(r)\bar{u}(T-1), \tag{6.10}$$

where

$$\bar{y}(T) = \begin{bmatrix} y(1)^\mathsf{T} & y(2)^\mathsf{T} & \cdots & y(T)^\mathsf{T} \end{bmatrix}^\mathsf{T},$$
$$\bar{y}(T-1) = \begin{bmatrix} y(1)^\mathsf{T} & y(2)^\mathsf{T} & \cdots & y(T-1)^\mathsf{T} \end{bmatrix}^\mathsf{T},$$
$$\bar{u}(T-1) = \begin{bmatrix} u(1)^\mathsf{T} & u(2)^\mathsf{T} & \cdots & u(T-1)^\mathsf{T} \end{bmatrix}^\mathsf{T},$$
$$\bar{Q}(r) = \begin{bmatrix} 0 & \cdots & \cdots & 0 \\ Q(1) & \ddots & \ddots & \vdots \\ Q(2) & Q(1) & \ddots & \vdots \\ \vdots & \vdots & \ddots & 0 \\ Q(r) & Q(r-1) & \cdots & Q(1) \\ 0 & Q(r) & \cdots & Q(2) \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & Q(r) \end{bmatrix},$$
$$\bar{P}(r) = \begin{bmatrix} 0 & \cdots & \cdots & 0 \\ P(1) & \ddots & \ddots & \vdots \\ P(2) & P(1) & \ddots & \vdots \\ \vdots & \vdots & \ddots & 0 \\ P(r) & P(r-1) & \cdots & P(1) \\ 0 & P(r) & \cdots & P(2) \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & P(r) \end{bmatrix}. \tag{6.11}$$

Note that the stability assumption allows $\bar{Q}(r)$ and $\bar{P}(r)$ to be matrices of finite dimension.

Since each $Q(i) \in \mathbb{R}^{p\times p}$ is a hollow matrix, each $Q(i)$ contains exactly $p^2 - p$ unknown values. Similarly, each $P(i) \in \mathbb{R}^{p\times m}$, each $P(i)$ has exactly pm unknown

values. Therefore, in $\bar{Q}(r)$ and $\bar{P}(r)$ combined, there are exactly $r(p^2-p+pm)$ unknown values.

Rewrite (6.10) as

$$\bar{y}(T) = \begin{bmatrix} \bar{Q}(r) & \bar{P}(r) \end{bmatrix} \begin{bmatrix} \bar{y}(T-1) \\ \bar{u}(T-1) \end{bmatrix}.$$

Taking the transpose of both sides, we get

$$\bar{y}(T)^\mathsf{T} = \begin{bmatrix} \bar{y}(T-1)^\mathsf{T} & \bar{u}(T-1)^\mathsf{T} \end{bmatrix} \begin{bmatrix} \bar{Q}(r)^\mathsf{T} \\ \bar{P}(r)^\mathsf{T} \end{bmatrix}.$$

Expanding this and removing the known $y(1) = 0$ yields:

$$\begin{bmatrix} y(2)^\mathsf{T} & \cdots & y(T)^\mathsf{T} \end{bmatrix} = \begin{bmatrix} y(1)^\mathsf{T} & \cdots & y(T-1)^\mathsf{T} & u(1)^\mathsf{T} & \cdots & u(T-1)^\mathsf{T} \end{bmatrix} \begin{bmatrix} Q(1)^\mathsf{T} & \cdots & Q(r)^\mathsf{T} & \cdots & 0 \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & \cdots & Q(1)^\mathsf{T} & \cdots & Q(r)^\mathsf{T} \\ P(1)^\mathsf{T} & \cdots & P(r)^\mathsf{T} & \cdots & 0 \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & \cdots & P(1)^\mathsf{T} & \cdots & P(r)^\mathsf{T} \end{bmatrix}$$

$$\triangleq LX \tag{6.12}$$

Now, stack each $Q(t)$ into a vector in row-major order[2], to produce the unknown vector $\vec{q}(t) \in \mathbb{R}^{p^2}$. Likewise stack $P(t)$ into the unknown vector $\vec{p}(t) \in \mathbb{R}^{pm}$. Stack each

[2] Throughout this work, we define $\vec{m}$ to be the vectorized form of M in row-major order; e.g. if

$$M = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix},$$

then

$$\vec{m} = \begin{bmatrix} a & b & c & d & e & f & g & h & i \end{bmatrix}^\mathsf{T}.$$

of the resulting vectors in a larger vector $\vec{x} \in \mathbb{R}^{r(p^2+pm)}$ given as follows:

$$\vec{x} = \begin{bmatrix} \vec{q}(1)^\mathsf{T} & \cdots & \vec{q}(r)^\mathsf{T} & \vec{p}(1)^\mathsf{T} & \cdots & \vec{p}(r)^\mathsf{T} \end{bmatrix}^\mathsf{T}. \tag{6.13}$$

We can then write (6.12) as

$$\hat{y} = \hat{L}\vec{x}, \tag{6.14}$$

with $\hat{y} \in \mathbb{R}^{(T-1)p}$ is given by

$$\hat{y} = \begin{bmatrix} y(2)^\mathsf{T} & \cdots & y(2)^\mathsf{T} & \cdots & y(T)^\mathsf{T} \cdots & y(T)^\mathsf{T} \end{bmatrix}, \tag{6.15}$$

and $\hat{L} \in \mathbb{R}^{(T-1)p \times r(p^2+pm)}$ is

$$\hat{L} = \begin{bmatrix}
y(1)^\mathsf{T} & 0 & 0 & \cdots & 0 & 0 & 0 & \cdots & u(1)^\mathsf{T} & 0 & 0 & \cdots & 0 & 0 & 0 \\
0 & \ddots & 0 & \cdots & 0 & \ddots & 0 & \cdots & 0 & \ddots & 0 & \cdots & 0 & \ddots & 0 \\
0 & 0 & y(1)^\mathsf{T} & \cdots & 0 & 0 & 0 & \cdots & 0 & 0 & u(1)^\mathsf{T} & \cdots & 0 & 0 & 0 \\
 & \vdots & & & & \vdots & & & & \vdots & & & & \vdots & \\
y(r)^\mathsf{T} & 0 & 0 & \cdots & y(1)^\mathsf{T} & 0 & 0 & \cdots & u(r)^\mathsf{T} & 0 & 0 & \cdots & u(1)^\mathsf{T} & 0 & 0 \\
0 & \ddots & 0 & \cdots & 0 & \ddots & 0 & \cdots & 0 & \ddots & 0 & \cdots & 0 & \ddots & 0 \\
0 & 0 & y(r)^\mathsf{T} & \cdots & 0 & 0 & y(1)^\mathsf{T} & \cdots & 0 & 0 & u(r)^\mathsf{T} & \cdots & 0 & 0 & u(1)^\mathsf{T} \\
 & \vdots & & & & \vdots & & & & \vdots & & & & \vdots & \\
y(T-1)^\mathsf{T} & 0 & 0 & \cdots & y(T-r)^\mathsf{T} & 0 & 0 & \cdots & u(T-1)^\mathsf{T} & 0 & 0 & \cdots & u(T-r)^\mathsf{T} & 0 & 0 \\
0 & \ddots & 0 & \cdots & 0 & \ddots & 0 & \cdots & 0 & \ddots & 0 & \cdots & 0 & \ddots & 0 \\
0 & 0 & y(T-1)^\mathsf{T} & \cdots & 0 & 0 & y(T-r)^\mathsf{T} & \cdots & 0 & 0 & u(T-1)^\mathsf{T} & \cdots & 0 & 0 & u(T-r)^\mathsf{T}
\end{bmatrix} \tag{6.16}$$

4. *Utilize Hollowness of $Q(t)$:* Since each $Q(i)$ is hollow, we can remove all entries of $\vec{x}$ corresponding to a $Q_{jj}(i)$ for all $i = 1, \ldots, r$ and $j = 1, \ldots, p$. These will be indices $i, i(p+2), i(2p+3), \ldots, i\left((p-2)p + (p-1)\right), ip^2$ for $i = 1, \ldots, r$. We also remove the corresponding columns of $\hat{L}$, giving us $\hat{L} \in \mathbb{R}^{(T-1)p \times r(p^2-p+pm)}$.

5. *Leverage A Priori Information:* Define

$$\hat{M} = \hat{L}\hat{K}, \qquad \hat{x} = \hat{K}^\mathsf{T}\vec{x}, \tag{6.17}$$

where $\hat{K}$ encodes all a priori structural information we may have. To construct $\hat{K}$, we first define $\bar{K} \in \mathbb{R}^{(p^2-p+pm)\times k}$ as a static matrix given by

$$\bar{K} = \begin{bmatrix} \bar{K}_{11} & \bar{K}_{12} \\ \bar{K}_{21} & \bar{K}_{22} \end{bmatrix},$$

where $\bar{K}$ represents the a priori information known about the *frequency domain* representation of the network. Specifically, K_{11} is the a priori information about the system that indicates how the reduced elements of $Q(z)$ map to the original elements of $Q(z)$, K_{12} is the a priori information that indicates how the reduced elements of $Q(z)$ map to the original elements of $P(z)$, etc.

For the time-domain representation, $\bar{K}_{11}$ operates on all $Q(i)$ and represents the a priori information mapping the reduced elements of $Q(i)$ to the original elements of $Q(i)$, and so forth. This gives us

$$\hat{K} = \begin{bmatrix} \bar{K}_{11} & \cdots & 0 & \bar{K}_{12} & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & \bar{K}_{11} & 0 & \cdots & \bar{K}_{12} \\ \bar{K}_{21} & \cdots & 0 & \bar{K}_{22} & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & \bar{K}_{21} & 0 & \cdots & \bar{K}_{22} \end{bmatrix}$$

A common choice for $\bar{K}$ is to encode *target specificity*, or in other words, to encode the assumption that $P(z)$ is diagonal. Suppose that $m = p = 3$. Then, we can

encode target specificity with the following choice of $\bar{K}$

$$
\begin{aligned}
\bar{K}_{11} &= I_{6\times 6} && \text{(No known zero links in } Q(z)\text{)} \\
\bar{K}_{12} &= 0_{6\times 9} && \text{(No known } Q(z) \text{ mappings to } P(z)\text{)} \\
\bar{K}_{21} &= 0_{9\times 6} && \text{(No known } P(z) \text{ mappings to } Q(z)\text{)} \\
\bar{K}_{22} &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} && \begin{matrix} (P_{11}(z) \text{ free}) \\ (P_{12}(z) = 0) \\ (P_{13}(z) = 0) \\ (P_{21}(z) = 0) \\ (P_{22}(z) \text{ free}) \\ (P_{23}(z) = 0) \\ (P_{31}(z) = 0) \\ (P_{32}(z) = 0) \\ (P_{33}(z) \text{ free}) \end{matrix}
\end{aligned}
\tag{6.18}
$$

Target specificity is a *sufficient* choice of $\bar{K}$ ensuring reconstructability in the frequency domain and in the time domain, but it is not necessary. The necessary condition requires that $\hat{K}$ encode the same amount of information as target specificity [6, 10, 16, 33].

6. *Fit the Impulse Responses:* Given the definitions in the previous steps, we can rewrite (6.3) as

$$\hat{y} = \hat{M}\hat{x} + e, \tag{6.19}$$

where $\hat{y}$ and $\hat{M}$ contain only known values from our input-output data, where $\hat{x}$ contains the unknown values we wish to fit, and where e represents the error of the fit for any choice of $\hat{x}$. Thus, we wish to choose $\hat{x}$ such that e is minimal according to some metric.

The VPNR minimizes e in the 2-norm sense, or in other words, chooses $\hat{x}$ according to

$$\hat{x}^* = \arg\min_{\hat{x}} \|e\|_2 = \arg\min_{\hat{x}} \|\hat{y} - \hat{M}\hat{x}\|_2,$$

which is solved using ordinary least squares. Thus, the VPNR can choose a unique $\hat{x}^*$ if the following conditions are met:

- $\hat{M} = \hat{L}\hat{K}$ is injective
- $\hat{y} \in \mathcal{R}(\hat{M})$ (where $\mathcal{R}(\hat{M})$ is defined as the range of $\hat{M}$)

Note that variations on the VPNR Algorithm can be created by using a different metric for e. For example, if choose $\hat{x}$ according to

$$\hat{x}^* = \arg\min_{\hat{x}} \|\hat{y} - \hat{M}\hat{x}\|_2 + \alpha\|\hat{x}\|_1,$$

(which can be solved using Lasso Regression with α selected by optimizing the Akaike Information Criterion), then the solution becomes robust to measurement noise on the inputs and outputs. We call the modified algorithm using Lasso Regression the Robust Passive Network Reconstruction (RPNR) Algorithm, which we will discuss in Chapter 7.

7. *Extract Links from $\hat{x}^*$:* Recall that we stacked the unknown values of $Q(t)$ and $P(t)$ into vector $\hat{x}$. Now that $\hat{x}^*$ is known, we extract the entries corresponding to each link from $\hat{x}^*$ and into $\tilde{Q}_{ij}(t)$ and $\tilde{P}_{ij}(t)$.

8. *Fit the Convolutional Models:* Theorem 2 in [10] shows that the entries of the inverse $\mathcal{Z}$-transform of $Q(z)$ take the form

$$Q_{ij}(t) = a_{q_{ij}}\delta_{(t,0)} + \sum_{n=0}^{w_{q_{ij}}} b_{n,q_{ij}}(c_{n,q_{ij}})^t, \tag{6.20}$$

where $w_{q_{ij}}$ is the number of delays in the corresponding link, $a_{q_{ij}} = \sum_{n=0}^{w_{q_{ij}}} b_{n,q_{ij}}$, and $\delta_{t,0}$ is the Kronecker delta.

The purpose of $a_{q_{ij}}\delta(t,0)$ in this equation is to force $Q_{ij}(0) = 0$. In other words, (6.20) can be rewritten as:

$$Q_{ij}(t) = \begin{cases} 0 & t = 0 \\ \sum_{n=0}^{w_{q_{ij}}} b_{n,q_{ij}} (c_{n,q_{ij}})^t & t > 0 \end{cases}.$$

Note that $P_{ij}(t)$ takes the form as well.

In order to find such a function, we simply need to choose the $2(w_{q_{ij}})$ parameters $b_{n,q_{ij}}$ and $c_{n,q_{ij}}$ such that (6.3) best fits the data in $\tilde{Q}_{ij}(t)$, meaning, we wish to choose the parameters such that $|Q_{ij}(t) - \tilde{Q}_{ij}(t)|$ is minimized. Unfortunately this convolution function is nonlinear. In [10], it was recommended that a non-linear curve fitting algorithm be used to fit these parameters. However, in this chapter, we suggest the use of an evolutionary algorithm instead, which has considerable advantages over the process presented in [10].

Though the evolutionary algorithm is slower than non-linear curve fitting, it does not require an initial starting guess for the parameters. Furthermore, the evolutionary algorithm can consistently reconstruct the network, while non-linear curve fitting required a good guess of the starting parameters to successfully reconstruct. Finally, the evolutionary algorithm can successfully reconstruct networks with much smaller values r than non-linear curve fitting, making the method more scalable both in time and in memory.

9. *Convert to Frequency Domain:* The frequency-domain representation of the link, $Q_{ij}(t)$ can be recovered using the following equations:

$$\begin{aligned} \alpha_{n,q_{ij}} &= b_{n,q_{ij}} c_{n,q_{ij}}, \\ \beta_{n,q_{ij}} &= c_{n,q_{ij}}, \\ Q_{ij}(z) &= \sum_{n=0}^{w_{q_{ij}}} \frac{\alpha_{n,q_{ij}}}{z - \beta_{n,q_{ij}}}, \end{aligned} \tag{6.21}$$

and likewise for $P_{ij}(z)$.

6.4 Assumptions Necessary for Reconstruction

In order to reconstruct a network, the following assumptions must hold for the data, the a priori information known about the system, and the underlying network that generated the data (note that if these assumptions hold, we are able to reconstruct a network perfectly):

- **Linearity:** The underlying network generating the input-output data must have linear dynamics. This is due to the fact that we are reconstructing DSFs, which presently only represent linear dynamics.
- **Stability:** The underlying network generating the input-output data must be stable. This allows the Toeplitz representation of the network in (6.10) to be finite dimensional.
- **Strict Causality:** The underlying network generating the input-output data must be strictly causal, meaning changes in the present only make an impact on the network strictly in the future. This also implies that the impulse responses at time $t = 0$ are $Q_{ij}(0) = 0$ and $P_{ij}(0) = 0$.
- **Informativity Conditions:** It must be known that the underlying system generating the data is target specific or something informationally equivalent (see Step 5 and [10, 16] for more information).

- **Non-noisy Data:** The inputs and outputs must be measured perfectly, with no noise. We relax this assumption in the next chapter.
- **Richness of Data:** We require that the input-output data be "rich enough" to reconstruct. This condition can be checked directly. The data is "rich enough" if $\hat{M}$ is injective and if $\hat{y}$ is in the range of $\hat{M}$ (see Theorem 1 in [10]).
- **Large Enough r:** We require that r be chosen large enough to capture the full dynamics of the system (see Section 6.6).
- **Large Enough T:** We require that enough data be collected that we don't overfit the least squares model. Further, T should be strictly larger than r (see Section 6.6).

6.5 Numeric Example

Consider the following discrete-time LTI state space representation of some system[3]:

$$x[k+1] = \begin{bmatrix} 0.75 & 0 & 0 & 0 & 0 & 1.2 \\ -0.1 & -0.35 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.85 & -1.0 & 0 & 0 \\ 0 & -0.73 & 0 & 0.95 & 0 & 0 \\ 0 & 0 & 0.43 & 0 & -0.6 & 0 \\ 0 & 0 & 0 & 0 & 0.2 & 0.55 \end{bmatrix} x[k] + \begin{bmatrix} 1.4 & 0 & 0 \\ 0 & -.25 & 0 \\ 0 & 0 & 0.75 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix},$$

$$y = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} x[k]. \tag{6.22}$$

[3] The example (6.22) is nearly identical to the example presented in [10], differing only in that $B_{13} = 0$ here where in [10], $B_{13} = -1.4$. As such, this example will be target specific–meaning that it has a diagonal $P(z)$–where the example in [10] was not. However, as explained in [10], target specificity is a sufficient condition allowing the reconstruction process; but it is not necessary.

The dynamical structure function $(Q(z), P(z))$ representation of this system is

$$Q(z) = \begin{bmatrix} 0 & 0 & \frac{41.28}{(4z-3)(5z+3)(20z-11)} \\ \frac{-2}{20z+7} & 0 & 0 \\ 0 & \frac{292}{(20z-17)(20z-19)} & 0 \end{bmatrix}$$

$$P(z) = \begin{bmatrix} \frac{5.6}{4z-3} & 0 & 0 \\ 0 & \frac{-5}{20z+7} & 0 \\ 0 & 0 & \frac{15}{20z-17} \end{bmatrix}. \tag{6.23}$$

As we are moving in the direction of Blind Reconstruction of Discrete-Time Networks– which will not reconstruct $P(z)$; in fact, we will ignore $P(z)$ from this point forward and concern ourselves only with $Q(z)$. Note that [10] demonstrates the numeric reconstruction of a non-target-specific $P(z)$.

We have that the convolution representation of $Q(z)$ (rounded to three decimal places) is given by:

$$\begin{aligned} Q_{12}(t) &= 0 \\ Q_{13}(t) &= 0.510(0.750)^t - 0.110(-0.600)^t - 0.816(0.550)^t + 0.416\delta_{(t,0)} \\ Q_{21}(t) &= 0.286(-0.350)^t - 0.286\delta_{(t,0)} \\ Q_{23}(t) &= 0 \\ Q_{31}(t) &= 0 \\ Q_{32}(t) &= 0.882(0.850)^t - 0.882\delta_{(t,0)} \end{aligned} \tag{6.24}$$

The magnitude and the vulnerability of the links in $Q(z)$ are given in Table 6.1, and graphical representations of the magnitude and vulnerabilities are shown in the top row of Figure 6.4.

The values in Table 6.1 are also normalized such that the largest magnitude is 1.0 (and likewise for vulnerability) as the value of these measures tend to be sensitive to slight perturbations in the DSF, whereas the relative values of these measures tend to be insensitive. As such, the Normalized Magnitude and Normalized Vulnerability should be the values used to compare any reconstructed network to the actual network. Figure 6.4 and subsequent Figures are constructed using the normalized values.

Link	Magnitude	Normalized Magnitude	Vulnerability	Normalized Vulnerability
$(1,2)$	0	0	1.133	0.003
$(1,3)$	0.573	0.006	28.756	0.074
$(2,1)$	0.154	0.002	195.200	0.503
$(2,3)$	0	0	387.471	1.000
$(3,1)$	0	0	7.6934	0.020
$(3,2)$	97.333	1.000	0.5710	0.002

Table 6.1 The size ($\|Q_{ij}(z)\|_\infty$) and the vulnerability ($\|(I - Q(z))^{-1}_{ji}\|_\infty$) of links (i,j) in the actual $Q(z)$ given by Equation (6.23).

Random inputs $\mathcal{D}_u$–with each $[\mathcal{D}_u]_{ij}$ selected uniformly from $[-1,1]$–were chosen to stimulate the system. Note that these inputs need not be controlled, they just need to excite the system enough that the conditions outlined in Step 6 of the VPNR Algorithm are met. However, experiments could be designed to choose these inputs in non-random manners, so long as the aforementioned conditions continue to be met. For this example, the inputs were generated for $T = 601$ time steps. Then the inputs were simulated on the state space model given in (6.23) to construct outputs $\mathcal{D}_y$.

Choosing[4] $r = 100$, the methodology described above was then used on $\mathcal{D}_u$ and $\mathcal{D}_y$ in an attempt to reconstruct the system. The convolution representation of reconstructed

[4] One method for selecting r is to choose some starting r, such as $r = 100$, and plot the impulse responses to each link such as those shown in Figure 6.3. If all links have time to converge to 0 at some $t < r$, then r was chosen to be sufficiently large. If not, a larger r will need to be chosen.

system found through this method (rounded to three decimal places) is as follows:

$$
\begin{aligned}
Q_{12}(t) &= 0.197(0.011)^t + 2.742(-0.001)^t - 0.143(0.001)^t - 0.037(0.013)^t - 2.758\delta_{(t,0)} \\
&\approx 0 \\
Q_{13}(t) &= 0.510(0.750)^t - 0.111(-0.600)^t - 0.816(0.550)^t - 0.417\delta_{(t,0)} \\
Q_{21}(t) &= 0.086(-0.351)^t + 0.2(-0.35)^t - 0.286\delta_{(t,0)} \\
&\approx 0.286(-0.35)^t - 0.286\delta_{(t,0)} \\
Q_{23}(t) &= -4.798(0.010)^t + 2.026(0.013)^t + 2.392(-0.042)^t - 3.396(-0.035)^t + 3.776\delta_{(t,0)} \\
&\approx 0 \\
Q_{31}(t) &= -2.730(-0.001)^t + 0.916(-0.027)^t - 0.894(-0.021)^t - 1.027(-0.003)^t + 3.735\delta_{(t,0)} \\
&\approx 0 \\
Q_{32}(t) &= -8.908(0.854)^t + 8.04(0.949)^t - 0.297(0.081)^t + 1.154(0.006)^t + 0.012\delta_{(t,0)} \\
&\approx -0.868(0.85)^t - 0.297(0.081)^t - 1.165\delta_{(t,0)} \qquad (6.25)
\end{aligned}
$$

Notice that the coefficients in Equation (6.25) are nearly identical to those in Equation (6.24). Figure 6.3 further demonstrates the goodness of this fit by comparing the impulse response of the reconstructed network with the impulse response of the actual network, which are nearly identical. Furthermore, the normalized magnitudes and vulnerabilities of the links in the reconstructed network are very close to those in the original network as shown in Table 6.2 and Figure 6.4.

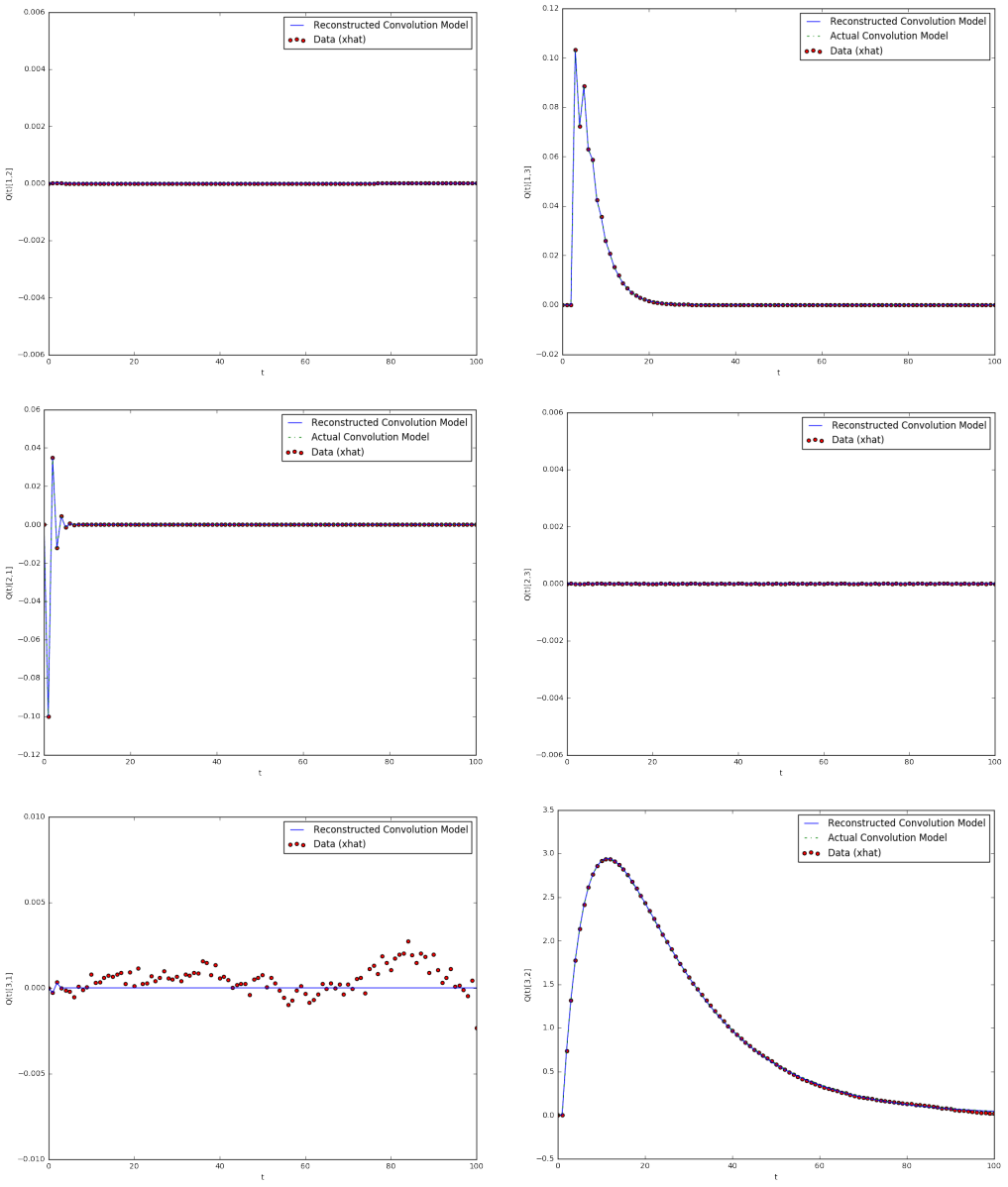


Figure 6.3 The impulse responses reconstructed using the Vanilla Network Reconstruction Algorithm on non-noisy data. The green dashed line is the actual convolution model given by Equation (6.24) (for $Q_{12}(t)$, $Q_{23}(t)$, and $Q_{31}(t)$, this line is not drawn as the convolution model as the exact fit is at 0 for all t). The red dots are the values of the impulse response $Q_{ij}(t)$ contained in $\hat{x}$ and found using least squares. The blue line is the reconstructed convolution model from Equation (6.25) using the evolutionary algorithm to fit an equation of the form (6.20) to the red dots.

Link	Magnitude	Normalized Magnitude	Vulnerability	Normalized Vulnerability
$(1,2)$	0.002	0.000	4.297	0.002
$(1,3)$	0.569	0.006	131.470	0.075
$(2,1)$	0.156	0.002	895.966	0.512
$(2,3)$	0.015	0.000	1748.452	1.000
$(3,1)$	0.010	0.000	29.208	0.017
$(3,2)$	102.256	1.000	1.835	0.001

Table 6.2 The magnitude ($\|Q_{ij}(z)\|_\infty$) and the vulnerability ($\|(I-Q(z))^{-1}_{ji}\|_\infty$) of links (i,j) in the $Q(z)$ reconstructed using the Vanilla Passive Reconstruction Algorithm.

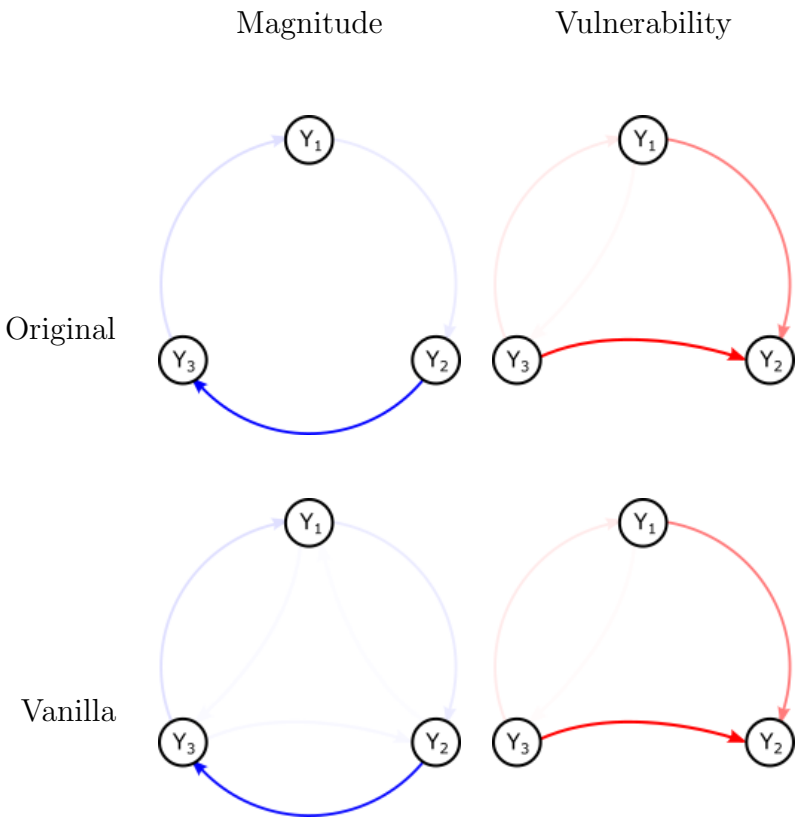


Figure 6.4 The magnitude (top left) and vulnerability (top right) of links in the original $Q(z)$ as given in Table 6.1 compared with the magnitude (bottom left) and vulnerability (bottom right) of links as given in Table 6.2. The darkness of each link is proportional to the normalized magnitudes and vulnerabilities, raised to the 0.4'th power in order to emphasize smaller links.

6.6 On the Convergence of the Vanilla Passive Network Reconstruction Algorithm

We can also use this example to explore the quality of reconstruction as we change the values of r and T. Let $Q_{ij}(t)$ be the true value at t for the impulse response of link (i, j), and let $\hat{Q}_{ij}(t)$ be the value of t of the reconstructed impulse response of the same link. Define $\hat{\epsilon}_{ij}(t) = |Q_{ij}(t) - \hat{Q}_{ij}|$ and let $\hat{e} = \frac{1}{T(p^2-p)} \sum_{t=1}^{T} \sum_{i,j=1;i\neq j}^{p} \hat{e}_{ij}(t)$ be the average error between reconstructed and true impulse responses of the network over all links and all time.

For each choice of r and T, we run 20 experiments, each with a different randomly-generated $\mathcal{D}_u$ and resultant $\mathcal{D}_y$, recording the average of $\hat{e}$ for all these experiments. The results of this "ranging" experiment are presented in Figure 6.5. Notice that, as r and T increase, the error $\hat{e}$ converges to 0, meaning that the network is reconstructed exactly. Thus, it is necessary to choose r and T large enough to recover the network, as expected.

Furthermore, for large enough T, the network has converged by $r = 100$ which is roughly the same time at which all the impulse responses in Figure 6.3 converge to 0. Finally, note that there is always some error if r is close to T, thus there should always be enough data such that T is strictly larger than r.

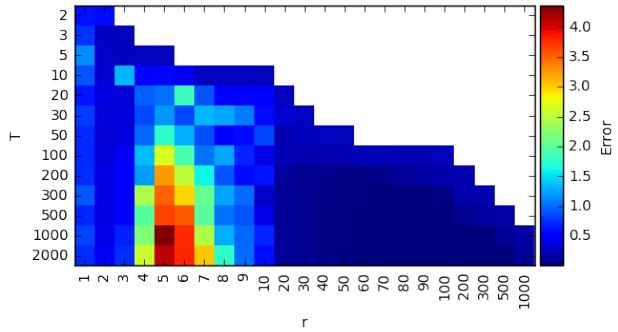


Figure 6.5 The quality of reconstruction of the VPNR Algorithm on the example in Section 6.5 at various levels of r and T.

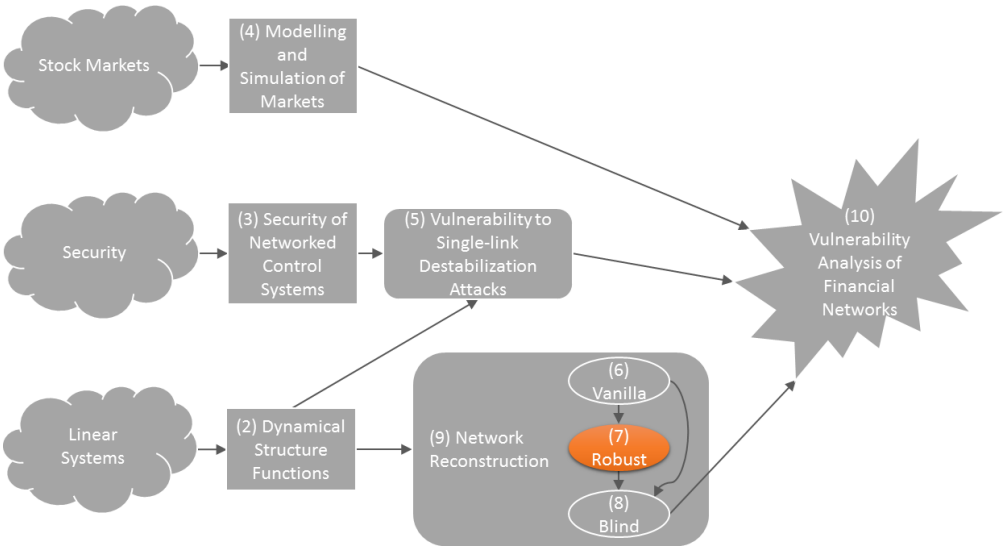
6.7 Conclusions

In conclusion, we have presented a reconstruction approach for networked systems in the time domain using observed inputs into and outputs from the system along with a priori structural knowledge about the system. The approach builds on an existing passive network reconstruction algorithm from [10] by utilizing an evolutionary algorithm to fit the convolutional models, which has led to considerable performance improvements.

This algorithm will converge to the true network so long as enough structural information is known and the input-output data is sufficiently exciting. We have also demonstrated this algorithm on a numeric example, showing that it does indeed converge to the true network given enough data.

Chapter 7

Robust Passive Network Reconstruction



In this chapter, we modify the VPNR Algorithm presented in Chapter 6 to be more robust to additive noise on the input-output data. This chapter is adapted from [46], which was submitted to the 2017 IEEE Conference on Decision and Control as a result of this thesis. In addition to the results presented in [46], this chapter includes a connection to vulnerability analysis as presented in Chapter 5.

7.1 Problem Formulation

The robust problem formulation is very similar to that in Section 6.2, differing only in how the data matrices $\mathcal{D}_u$ and $\mathcal{D}_y$ are defined. Formally, suppose that $\mathcal{D}_u$ and $\mathcal{D}_y$ are measured with additive error, meaning that if $\hat{\mathcal{D}}_u$ and $\hat{\mathcal{D}}_y$ are the actual inputs and outputs into the system, then $\mathcal{D}_u = \hat{\mathcal{D}}_u + \mathcal{E}_u$ and $\mathcal{D}_y = \hat{\mathcal{D}}_y + \mathcal{E}_y$ for $\mathcal{E}_u$ and $\mathcal{E}_y$ of appropriate dimensions. We

will assume that entries in $\mathcal{E}_u$ and $\mathcal{E}_y$ are taken from a Gaussian distribution with mean 0 and standard deviation small enough that the noise does not become more significant than the values in $\hat{\mathcal{D}}_u$ and $\hat{\mathcal{D}}_y$. Given this noisy data, the problem is to reconstruct the DSF $(Q(z), P(z))$ as described in Section 6.2.

7.2 The Robust Passive Network Reconstruction Algorithm

Now, we consider the case where there is additive noise on the inputs and/or outputs of the system, and we present a new algorithm, which we call the Robust Passive Network Reconstruction (RPNR) Algorithm. All steps are identical to the VPNR presented in Section 6.3, differing only in Step 6. We define that difference here.

Previously, we used ordinary least squares to select $\hat{x}^* = \arg\min_{\hat{x}} \|\hat{y} - \hat{M}\hat{x}\|_2$. However, now that noise has been added, the least squares will try to optimize this by making all links in $Q(z)$ and $P(z)$ non-zero to fit the noise. This is not desirable as we are trying to find both the structure and the dynamics of the DSF. As such, we wish to modify the least squares formulation in order to find the least complicated structure that still fits the data well.

To do this, we choose to solve the following problem

$$\hat{x}^* = \arg\min_{\hat{x}} \|y - \hat{M}\hat{x}\|_2 + \alpha\|\hat{x}\|_0. \tag{7.1}$$

The addition of the $\|\hat{x}\|_0$ term penalizes dense solutions, or in other words the solution seeks to make as many values of $Q(t)$ and $P(t)$ zero as possible. Unfortunately, since the zero-norm is not actually a norm, this problem is hard to solve. As such, we use the standard lasso relaxation of the problem, given as follows:

$$\hat{x}^* = \arg\min_{\hat{x}} \|y - \hat{M}\hat{x}\|_2 + \alpha\|\hat{x}\|_1. \tag{7.2}$$

We then utilize the Akaike Information Criterion (AIC) to select an α. The AIC measures the trade-off between goodness of fit of $\arg\min_{\hat{x}} \|y - \hat{M}\hat{x}\|_2$ with the complexity of

the selected model penalized by $\alpha\|\hat{x}\|_1$, relative to all other choices of α. We select the α that maximizes the AIC, and returns the $\hat{x}^*$ that optimizes the resulting problem.

Once (7.2) is solved and an optimal $\hat{x}^*$ is found, then the algorithm proceeds as described in Section 6.3, fitting a convolution model to the points in $\hat{x}$ to reconstruct the DSF.

Note that, for tractability, this scheme makes no distinction between a single link that is zero for all t (which is what we desire) and two links that are zero for half of t. Future work may benefit from enforcing zero links over zero entries.

7.3 Assumptions Necessary for Reconstruction

Unfortunately, due to the added noise on the input-output data, we are not able to reconstruct the original network exactly, at least not without additional future research. However, we are still able to get approximations of the network that are very close (see Section 7.4). In order to generate reasonable approximations, all the assumptions that were listed in Section 6.4 must also hold here, apart from the assumption of non-noisy data, which we relax here.

Specifically, we assume that the noise on the input-output data is small, but not too small. If the noise is too large, then it will overwhelm the data, and we will not be able to reconstruct. If it is too small, then the VPNR Algorithm will be able to reconstruct better than the RPNR Algorithm and should be used instead, though the RPNR Algorithm will still perform reasonably well. We do not define precisely how much noise is "too big" or "too small", though Figures 7.7, 7.12, and 7.17 can be used to give a rough idea of what they mean.

7.4 Numeric Examples

We now return to the numeric example introduced in Section 6.5, comparing the performance of the VPNR and the RPNR in situations where there is no noise on the input-output data,

and where there is noise on the inputs only, the outputs only, or on both the inputs and outputs.

7.4.1 Robust Network Reconstruction on Non-Noisy Data

As in Section 6.5, we assume no noise on inputs, states, or outputs; however, we use the RPNR Algorithm instead of the VPNR Algorithm. The parameters found are as follows:

$$
\begin{aligned}
Q_{12}(t) &= -0.374(0.574)^t + 0.549(0.421)^t + 6.280(-0.004)^t + 3.232(-0.003)^t - 9.686\delta_{(t,0)} \\
Q_{13}(t) &= 0.694(0.733)^t - 0.076(-0.644)^t - 1.005(0.583)^t - 1.799(0.016)^t + 2.165\delta_{(t,0)} \\
Q_{21}(t) &= 1.348(-0.103)^t - 0.078(0.105)^t + 0.351(0.108)^t + 0.320(0.002)^t - 2.096\delta_{(t,0)} \\
Q_{23}(t) &= -0.146(-0.014)^t - 0.810(0.003)^t + 0.298(0.012)^t - 0.790(0.004)^t + 1.449\delta_{(t,0)} \\
&\approx 0 \\
Q_{31}(t) &= -1.443(0.009)^t - 2.183(-0.001)^t - 0.001(0.019)^t - 2.654(0.007)^t + 0.973\delta_{(t,0)} \\
&\approx 0 \\
Q_{32}(t) &= 9.123(0.944)^t - 10.000(0.860)^t - 0.491(0.005)^t - 0.531(0.006)^t + 1.899\delta_{(t,0)} \qquad (7.3)
\end{aligned}
$$

Notice that the coefficients in (7.3) are not as close to the actual coefficients in (6.24). However, as can be seen in Figure 7.1, the dynamics of the link impulse responses are still close to the actual system. The notable differences are the introduction of small, but non-zero dynamics into $Q_{12}(t)$ and the errors in the dynamics for $Q_{21}(t)$. Nonetheless, the other links that should be zero are still zero, and the links that are not zero still capture many of the key features of the dynamics. The magnitude and vulnerabilities of the links are also very close to the actual system as well (see Table 7.1 and Figure 7.2).

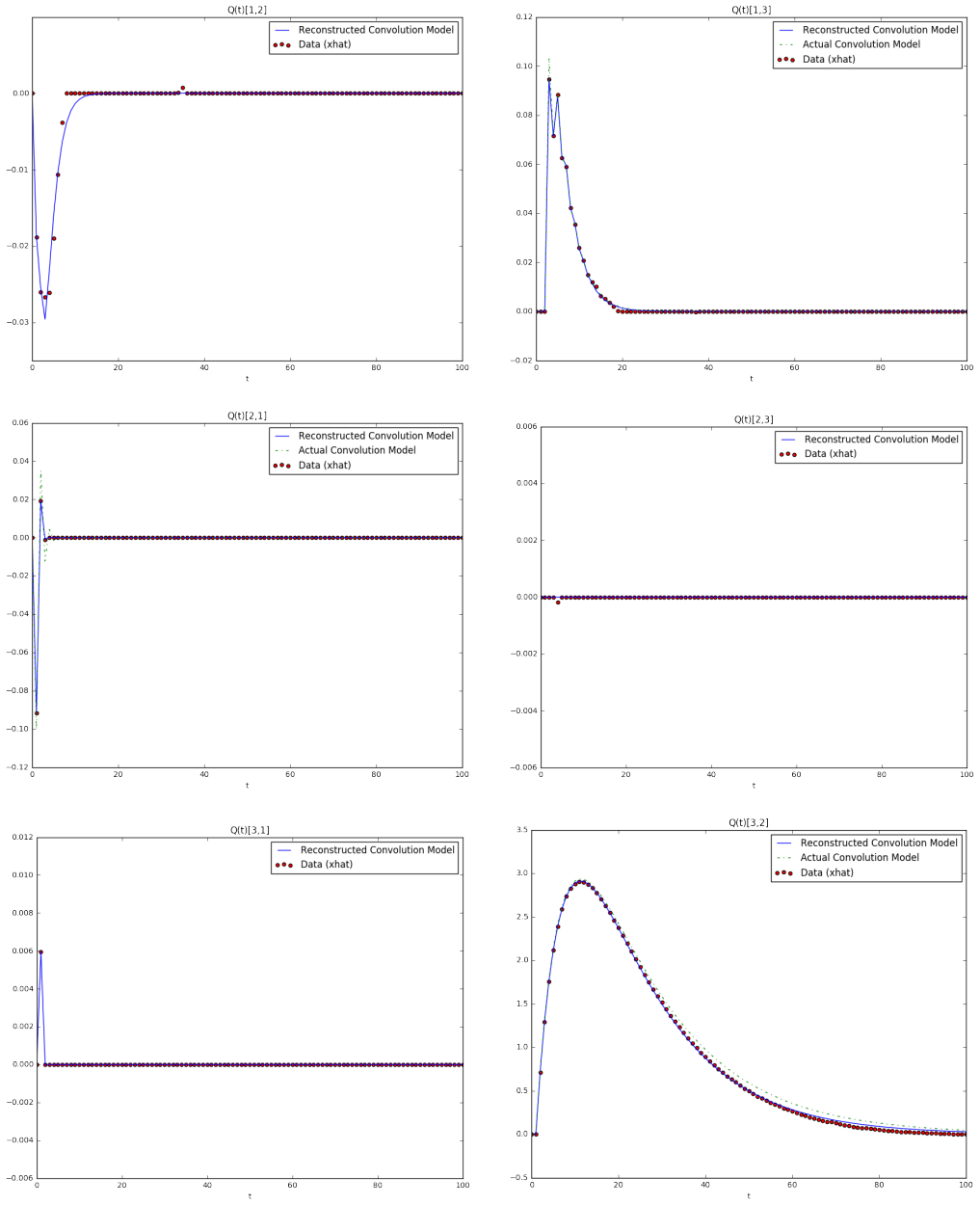


Figure 7.1 The impulse responses reconstructed using the RPNR Algorithm on non-noisy data. The green dashed line is the actual convolution model given by Equation (6.24) (for $Q_{12}(t)$, $Q_{23}(t)$, and $Q_{31}(t)$, this line is not drawn as the convolution model as the exact fit is at 0 for all t). The red dots are the values of the impulse response $Q_{ij}(t)$ contained in $\hat{x}$ and found using least squares. The blue line is the reconstructed convolution model from Equation (6.25) using the evolutionary algorithm to fit an equation of the form (6.20) to the red dots.

Link	Magnitude	Normalized Magnitude	Vulnerability	Normalized Vulnerability
$(1,2)$	0.092	0.001	0.406	0.003
$(1,3)$	0.551	0.007	10.755	0.071
$(2,1)$	0.108	0.001	75.289	0.497
$(2,3)$	0.005	0.000	151.615	1.000
$(3,1)$	0.012	0.000	2.838	0.019
$(3,2)$	88.441	1.000	0.184	0.001

Table 7.1 The magnitude ($\|Q_{ij}(z)\|_\infty$) and the vulnerability ($\|(I-Q(z))^{-1}_{ji}\|_\infty$) of links (i,j) in the $Q(z)$ of non-noisy data reconstructed using the RPNR Algorithm.

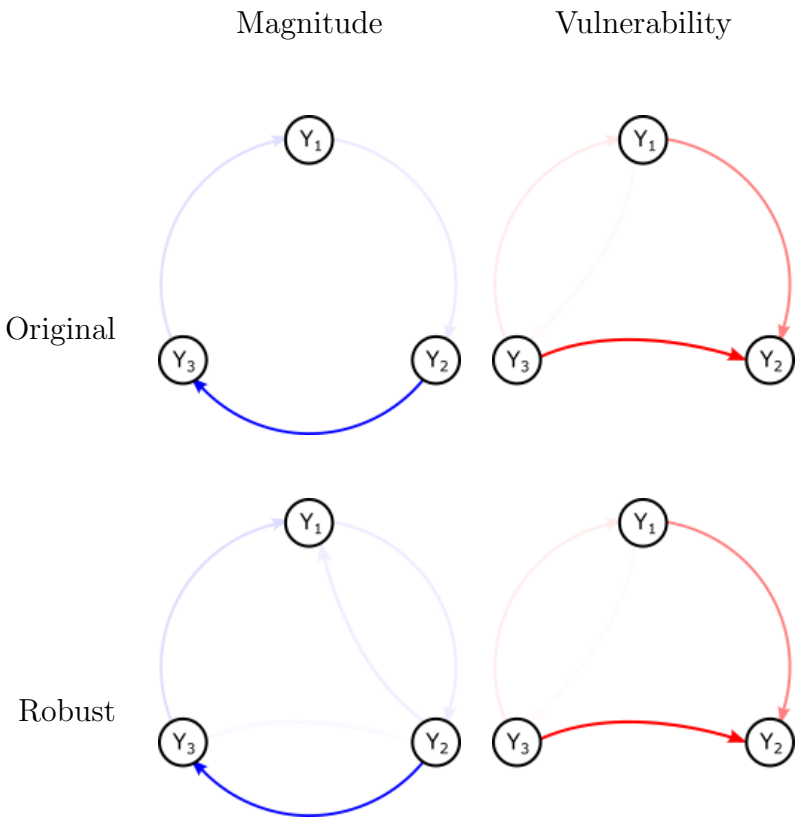


Figure 7.2 The magnitude (top left) and vulnerability (top right) of links in the original $Q(z)$ as given in Table 6.1 compared with the magnitude (bottom left) and vulnerability (bottom right) of links as given in Table 7.1. The darkness of each link is proportional to the normalized magnitudes and vulnerabilities, raised to the 0.4'th power in order to emphasize smaller links.

7.4.2 Vanilla and Robust Network Reconstruction on Noisy Inputs

Now we consider the case where there is noise on the inputs into the system. Let $\mathcal{N}(0,1)$ be the normal distribution of mean 0 and standard deviation 1, and let each call to this function generate a random variable independent from all other calls. Then we simulate the system introduced in Section 6.5 as before. This time, however, we perturb $\mathcal{D}_u$ before it is used to generate $\hat{M}$ so that, with $\tilde{\mathcal{D}}_u$ being the original input, the input seen by the reconstruction algorithm is

$$[\mathcal{D}_u]_{ij} = [\tilde{\mathcal{D}}_u]_{ij} + \gamma\mathcal{N}(0,1), \tag{7.4}$$

where $\gamma > 0$ parametrizes the magnitude of the noise. Recall that $[\tilde{\mathcal{D}}]_{ij}$ is generated randomly from a uniform distribution in the range $[-1,1]$. Therefore, any γ roughly greater than 1 will create noise of magnitude larger than the inputs themselves.

We first evaluate the VPNR Algorithm (not the RPNR Algorithm) when $\gamma = 0.2$. Notice from Figures 7.3 and 7.4 and Table 7.2 that the Vanilla Passive Reconstruction Algorithm reconstructs the non-zero links in the network well. It also computes the vulnerability of the network quite well. However, it also reconstructs significant dynamics on links that should be zero, especially on link $(1,2)$.

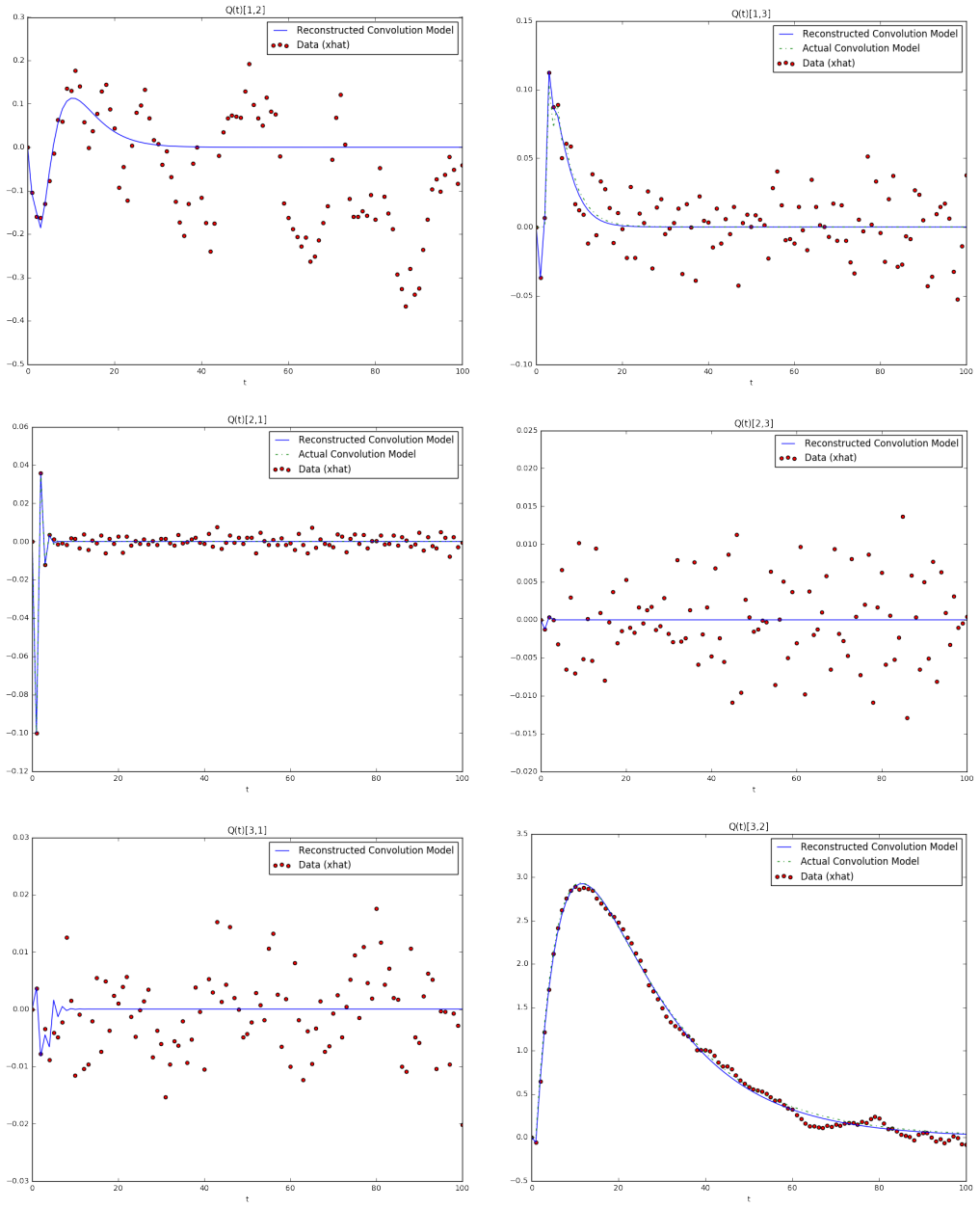


Figure 7.3 The impulse responses reconstructed using the VPNR Algorithm on noisy inputs. The green dashed line is the actual convolution model given by Equation (6.24) (for $Q_{12}(t)$, $Q_{23}(t)$, and $Q_{31}(t)$, this line is not drawn as the convolution model as the exact fit is at 0 for all t). The red dots are the values of the impulse response $Q_{ij}(t)$ contained in $\hat{x}$ and found using least squares. The blue line is the reconstructed convolution model from Equation (6.25) using the evolutionary algorithm to fit an equation of the form (6.20) to the red dots.

Link	Magnitude	Normalized Magnitude	Vulnerability	Normalized Vulnerability
$(1,2)$	40.929	0.462	0.973	0.003
$(1,3)$	0.628	0.007	24.060	0.078
$(2,1)$	0.204	0.002	149.882	0.485
$(2,3)$	0.025	0.000	308.852	1.000
$(3,1)$	0.233	0.003	6.498	0.021
$(3,2)$	88.659	1.000	0.581	0.002

Table 7.2 The magnitude ($\|Q_{ij}(z)\|_\infty$) and the vulnerability ($\|(I-Q(z))^{-1}_{ji}\|_\infty$) of links (i,j) in the $Q(z)$ of noisy inputs reconstructed using the VPNR Algorithm.

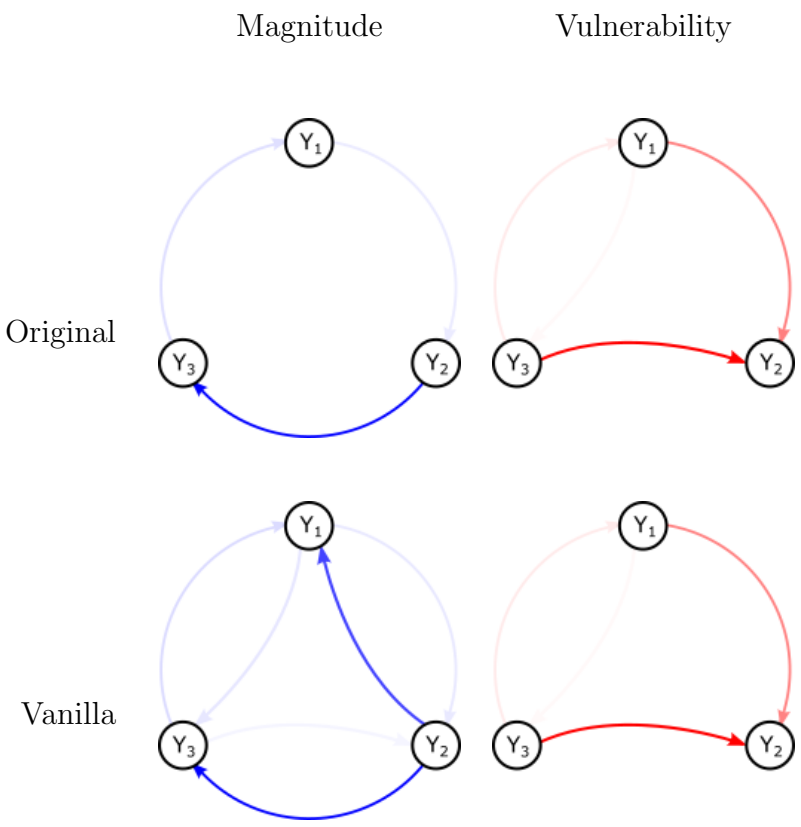


Figure 7.4 The magnitude (top left) and vulnerability (top right) of links in the original $Q(z)$ as given in Table 6.1 compared with the size (bottom left) and vulnerability (bottom right) of links as given in Table 7.2. The darkness of each link is proportional to the normalized magnitudes and vulnerabilities, raised to the 0.4'th power in order to emphasize smaller links.

Now we evaluate the RPNR Algorithm on noisy inputs when $\gamma = 0.2$. Notice from Figures 7.5 and 7.6 as well as Table 7.3 that the RPNR Algorithm performed decently as compared to the VPNR Algorithm. It did not capture the particulars of the dynamics (shown in 7.5) on links $(1, 3)$, and $(2, 1)$ quite as well. And while the vulnerabilities computed were comparable in nearness to the actual vulnerabilities as compared to the VPNR Algorithm, they weren't quite as good. Furthermore, like the VPNR Algorithm, it introduced non-zero dynamics on links that should have been zero, especially on link $(1, 2)$. However, notice the links that should have been zero were much smaller than in the VPNR Algorithm (notice that links $(2, 3)$ and $(3, 1)$ were nearly zero), which was the aim of the RPNR Algorithm.

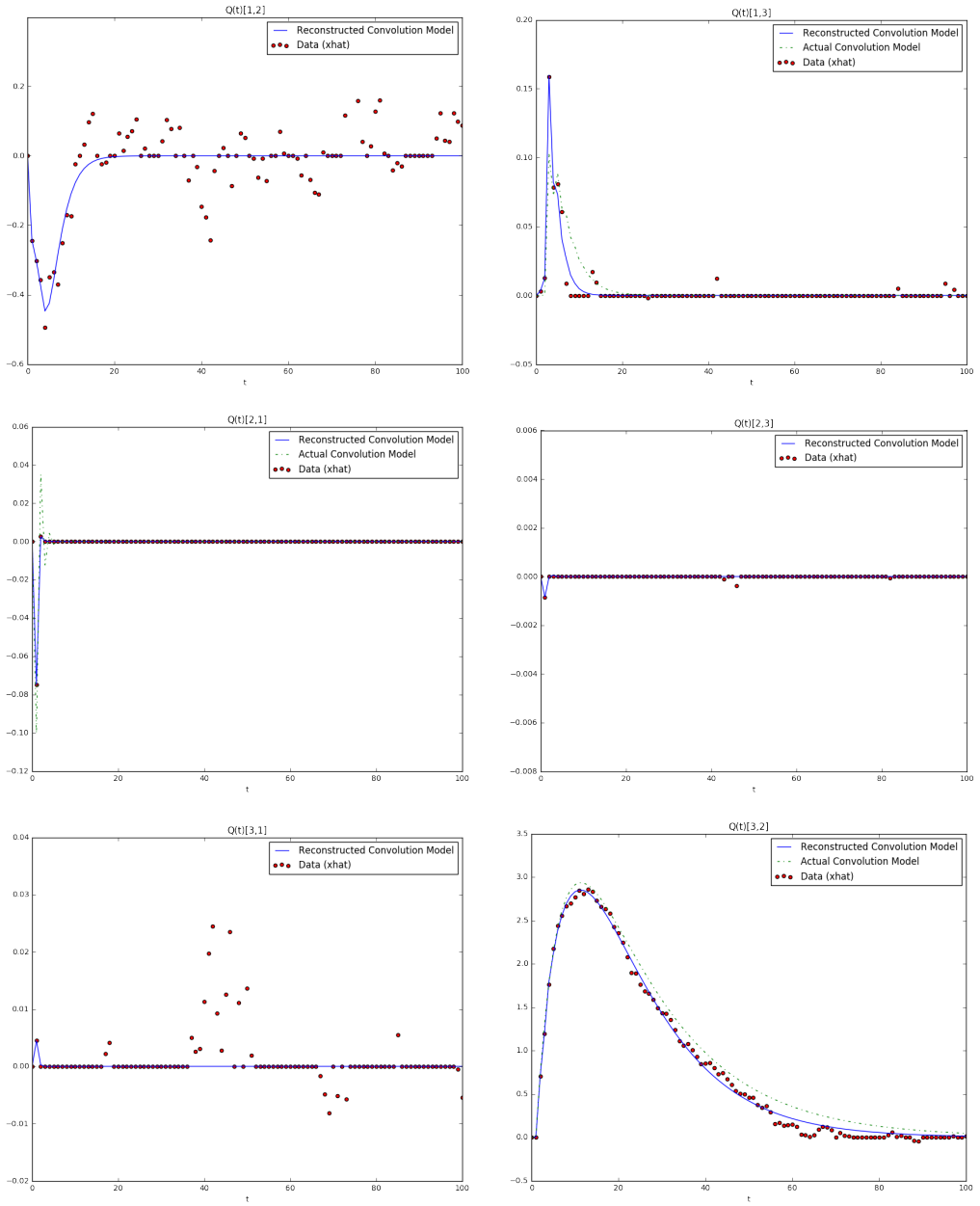


Figure 7.5 The impulse responses reconstructed using the RPNR Algorithm on noisy inputs. The green dashed line is the actual convolution model given by Equation (6.24) (for $Q_{12}(t)$, $Q_{23}(t)$, and $Q_{31}(t)$, this line is not drawn as the convolution model as the exact fit is at 0 for all t). The red dots are the values of the impulse response $Q_{ij}(t)$ contained in $\hat{x}$ and found using least squares. The blue line is the reconstructed convolution model from Equation (6.25) using the evolutionary algorithm to fit an equation of the form (6.20) to the red dots.

Link	Magnitude	Normalized Magnitude	Vulnerability	Normalized Vulnerability
$(1,2)$	3.431	0.044	0.699	0.006
$(1,3)$	0.419	0.005	10.778	0.097
$(2,1)$	0.103	0.001	58.655	0.526
$(2,3)$	0.033	0.000	111.571	1.000
$(3,1)$	0.008	0.000	3.497	0.031
$(3,2)$	78.235	1.000	0.470	0.004

Table 7.3 The size ($\|Q_{ij}(z)\|_\infty$) and the vulnerability ($\|(I - Q(z))^{-1}_{ji}\|_\infty$) of links (i,j) in the $Q(z)$ of noisy inputs reconstructed using the Robust Passive Reconstruction Algorithm.

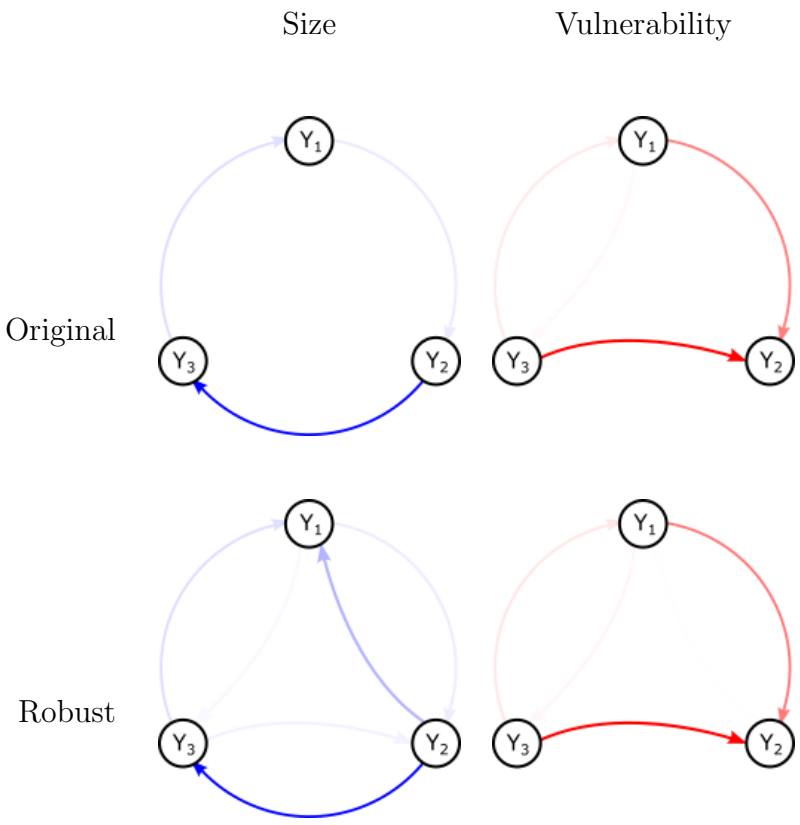


Figure 7.6 The size (top left) and vulnerability (top right) of links in the original $Q(z)$ as given in Table 6.1 compared with the size (bottom left) and vulnerability (bottom right) of links as given in Table 7.3. The darkness of each link is proportional to the normalized magnitudes and vulnerabilities, raised to the 0.4'th power in order to emphasize smaller links.

Finally, we compare the performance of both the VPNR Algorithm and the RPNR Algorithm across different magnitudes of noise γ. To do so, we introduce an error as a metric to represent the goodness of a single network reconstruction run. Let $\xi_{ij}(t)$ be the actual impulse response of link (i,j) at times $t = 0, \ldots, r$ as represented by the green dashed line in Figures 7.4 and 7.6. Let $\hat{\xi}_{ij}(t,b)$ be the reconstructed impulse response of link (i,j) at times $t = 0, \ldots, r$ in batch run[1] $b = 1, \ldots, 20$ as represented by the red dots in the same figures. Then define the batch link error $\eta_{ij}(b)$, the batch network error $\eta(b)$, and the network error η as

$$\begin{aligned}
\eta_{ij}(b) &= \frac{1}{r+1}\left(\sum_{t=0}^{r}\left(\xi_{ij}(t) - \hat{\xi}_{ij}(t,b)\right)^2\right)^{\frac{1}{2}}, \\
\eta(b) &= \frac{1}{p^2-p}\sum_{\substack{i=1,\ldots,p \\ j=1,\ldots,p \\ i\neq j}} \eta_{ij}, \\
\eta &= \frac{1}{20}\sum_{b=1}^{20}\eta(b)
\end{aligned} \tag{7.5}$$

In other words, $\eta_{ij}(b)$ is the root mean square error (RMSE) of the reconstructed impulse response with the actual impulse response on link (i,j) for batch run b, $\eta(b)$ is the average of these RMSE's across all links in $Q(t)$ in batch run b, and η is the average of these average RMSE's across all batch runs.

Figure 7.7 compares the average RMSE η of the VPNR Algorithm with the RPNR Algorithm. Notice how the VPNR Algorithm performs better (lower error) than the RPNR Algorithm at lower levels of noise. This is sensible since the RPNR Algorithm moves away from an optimal fit to penalize model complexity; as such, if there is no need to penalize model complexity, it will tend to perform worse. However, at $\gamma > 0.1$, the RPNR Algorithm begins to perform significantly better than the VPNR Algorithm, a good indication that it is properly handling the noise in the inputs.

[1] Each of the batch runs in these experiments reconstruct a network using the same inputs and outputs across all experiments, but with noise generated with different seeds.

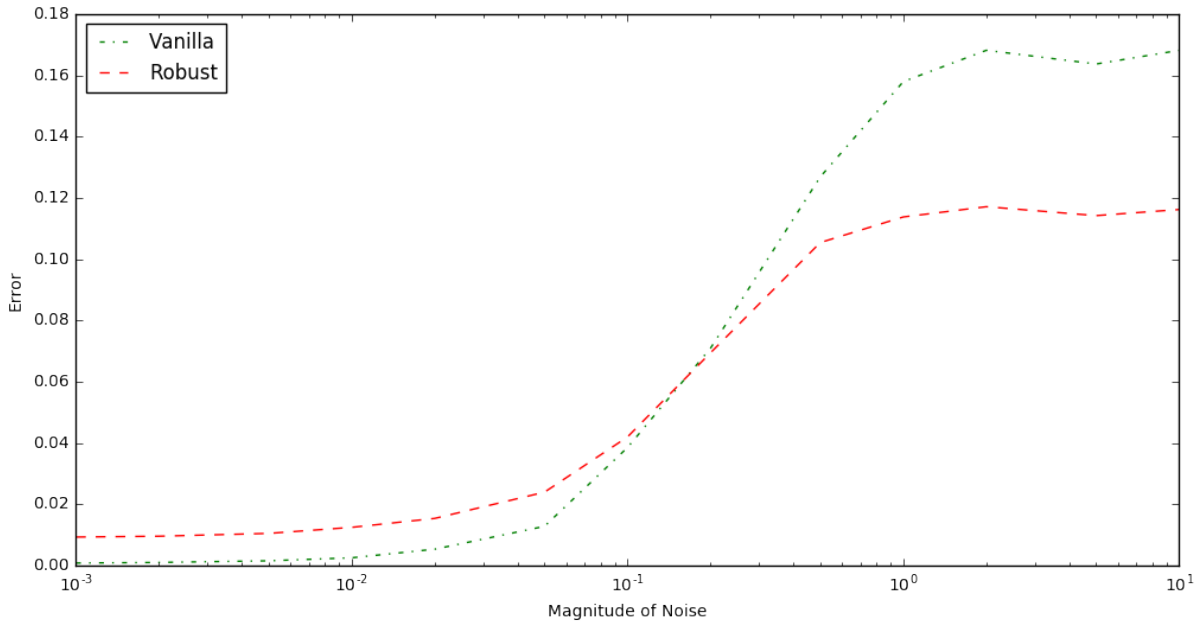


Figure 7.7 Effect of input noise on the ability of the VPNR and RPNR Algorithms to reconstruct from data.

7.4.3 Vanilla and Robust Network Reconstruction on Noisy Outputs

We now repeat the experiments from the previous section, this time exploring the effects of noise on outputs instead of on inputs. We simulate the system introduced in Section 6.5 as before, though with unperturbed inputs. Instead, we perturb $\mathcal{D}_y$ before it is measured so that, with $\tilde{\mathcal{D}}_y$ being the actual output produced by input $\mathcal{D}_u$, the output seen by the reconstruction algorithm is

$$[\mathcal{D}_y]_{ij} = [\tilde{\mathcal{D}}_y]_{ij} + \gamma\mathcal{N}(0,1), \tag{7.6}$$

where $\gamma > 0$ once again parametrizes the magnitude of the noise.

We first evaluate the VPNR Algorithm (not the RPNR Algorithm) when $\gamma = 0.1$. From Figures 7.8 and 7.9, along with Table 7.4, we see that, though the reconstructed network is able to capture many of the main characteristics of the original network, it still introduces fairly significant errors, especially in links $(3,1)$ and $(3,2)$.

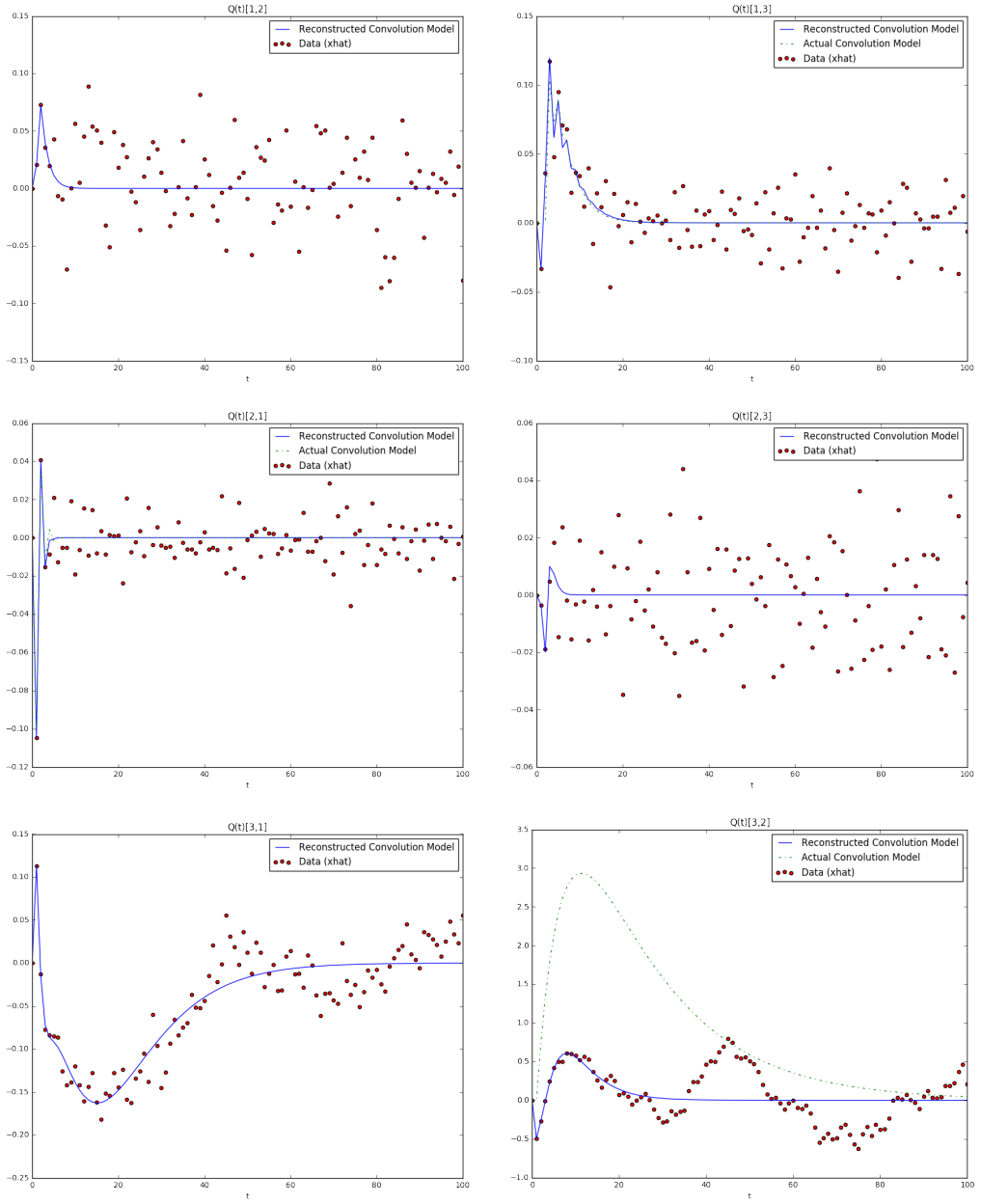


Figure 7.8 The impulse responses reconstructed using the VPNR Algorithm on noisy outputs. The green dashed line is the actual convolution model given by Equation (6.24) (for $Q_{12}(t)$, $Q_{23}(t)$, and $Q_{31}(t)$, this line is not drawn as the convolution model as the exact fit is at 0 for all t). The red dots are the values of the impulse response $Q_{ij}(t)$ contained in $\hat{x}$ and found using least squares. The blue line is the reconstructed convolution model from Equation (6.25) using the evolutionary algorithm to fit an equation of the form (6.20) to the red dots.

Link	Magnitude	Normalized Magnitude	Vulnerability	Normalized Vulnerability
$(1,2)$	0.145	0.027	2.525	0.020
$(1,3)$	0.544	0.102	58.617	0.470
$(2,1)$	0.209	0.039	60.353	0.484
$(2,3)$	0.041	0.008	124.659	1.000
$(3,1)$	3.862	0.725	13.400	0.107
$(3,2)$	5.325	1.000	1.134	0.009

Table 7.4 The size ($\|Q_{ij}(z)\|_\infty$) and the vulnerability ($\|(I - Q(z))^{-1}_{ji}\|_\infty$) of links (i,j) in the $Q(z)$ of noisy outputs reconstructed using the VPNR Algorithm.

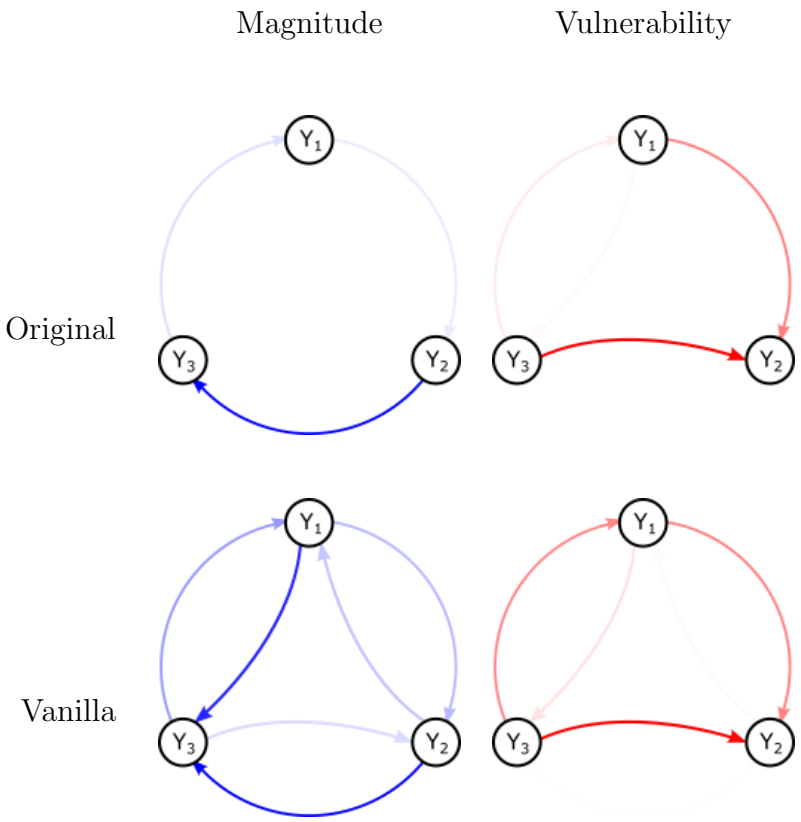


Figure 7.9 The magnitude (top left) and vulnerability (top right) of links in the original $Q(z)$ as given in Table 6.1 compared with the magnitude (bottom left) and vulnerability (bottom right) of links as given in Table 7.4. The darkness of each link is proportional to the normalized magnitudes and vulnerabilities, raised to the 0.4'th power in order to emphasize smaller links.

We now valuate the RPNR Algorithm when $\gamma = 0.1$. From Figures 7.10 and 7.11, along with Table 7.5, we see that, though the reconstructed network–like that found by the VPNR Algorithm–is able to capture many of the main characteristics of the original network, it still introduces fairly significant errors, especially in links $(3, 1)$ and $(3, 2)$. However, these errors are somewhat muted from those found in the VPNR Algorithm.

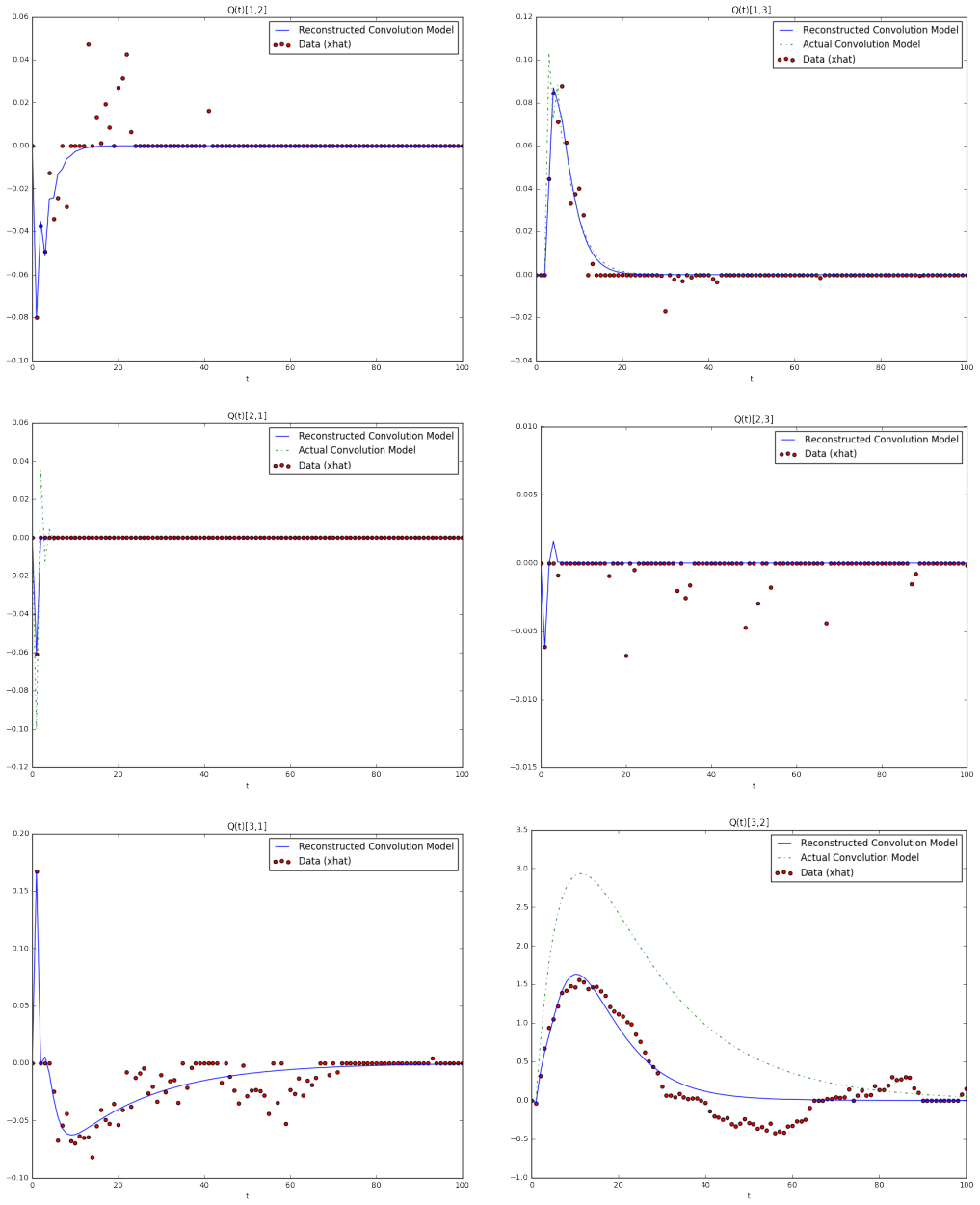


Figure 7.10 The impulse responses reconstructed using the RPNR Algorithm on noisy outputs. The green dashed line is the actual convolution model given by Equation (6.24) (for $Q_{12}(t)$, $Q_{23}(t)$, and $Q_{31}(t)$, this line is not drawn as the convolution model as the exact fit is at 0 for all t). The red dots are the values of the impulse response $Q_{ij}(t)$ contained in $\hat{x}$ and found using least squares. The blue line is the reconstructed convolution model from Equation (6.25) using the evolutionary algorithm to fit an equation of the form (6.20) to the red dots.

Link	Magnitude	Normalized Magnitude	Vulnerability	Normalized Vulnerability
$(1,2)$	0.262	0.008	0.194	0.003
$(1,3)$	0.556	0.017	5.680	0.087
$(2,1)$	0.054	0.002	33.603	0.512
$(2,3)$	0.004	0.000	65.621	1.000
$(3,1)$	1.422	0.042	1.876	0.029
$(3,2)$	33.550	1.000	0.092	0.001

Table 7.5 The magnitude ($\|Q_{ij}(z)\|_\infty$) and the vulnerability ($\|(I-Q(z))^{-1}_{ji}\|_\infty$) of links (i,j) in the $Q(z)$ of noisy outputs reconstructed using the RPNR Algorithm.

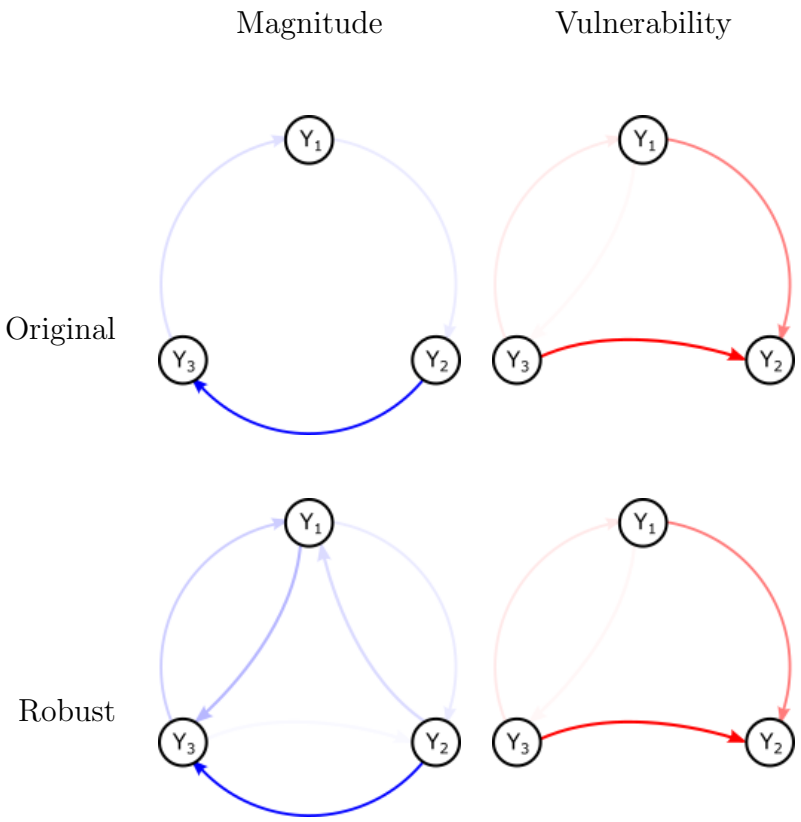


Figure 7.11 The magnitude (top left) and vulnerability (top right) of links in the original $Q(z)$ as given in Table 6.1 compared with the magnitude (bottom left) and vulnerability (bottom right) of links as given in Table 7.5. The darkness of each link is proportional to the normalized magnitudes and vulnerabilities, raised to the 0.4'th power in order to emphasize smaller links.

Finally, we compute the average RMSE of the impulse responses η in order to compare the performances of the VPNR Algorithm with the RPNR Algorithm as the magnitude of the output noise γ increases. Figure 7.12 shows the results of this analysis. Again, we see that the VPNR Algorithm performs better with low levels of noise where the RPNR Algorithm performs better with high levels of noise. However, the differences in the performances of these two algorithms are much smaller than they were before.

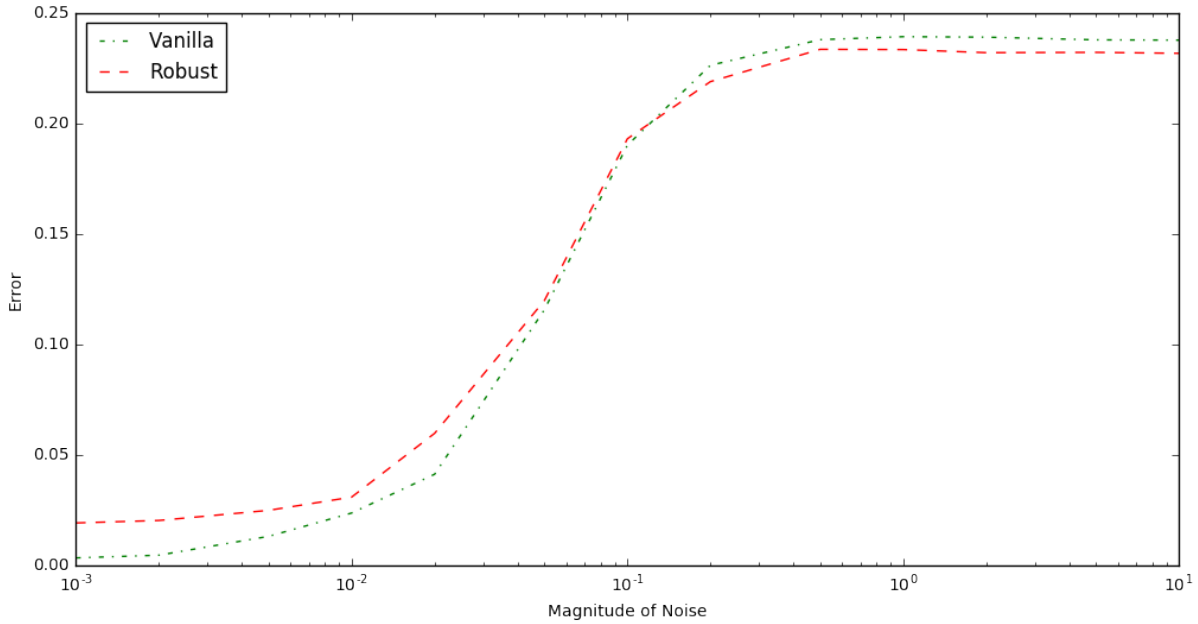


Figure 7.12 Effect of output noise on the ability of the VPNR and RPNR Algorithms to reconstruct from data.

7.4.4 Vanilla and Robust Network Reconstruction on Noisy Inputs and Outputs

Once again, we repeat the experiments of the previous sections, this time with noise affecting both the inputs into and outputs from the system. We use the same noise magnitude γ to perturb both the inputs and the outputs.

We first evaluate the VPNR Algorithm (not the RPNR Algorithm) when $\gamma = 0.05$. The results of this experiment are given in Figures 7.13 and 7.14 and in Table 7.6. Again, the algorithm performs well, though with some error.

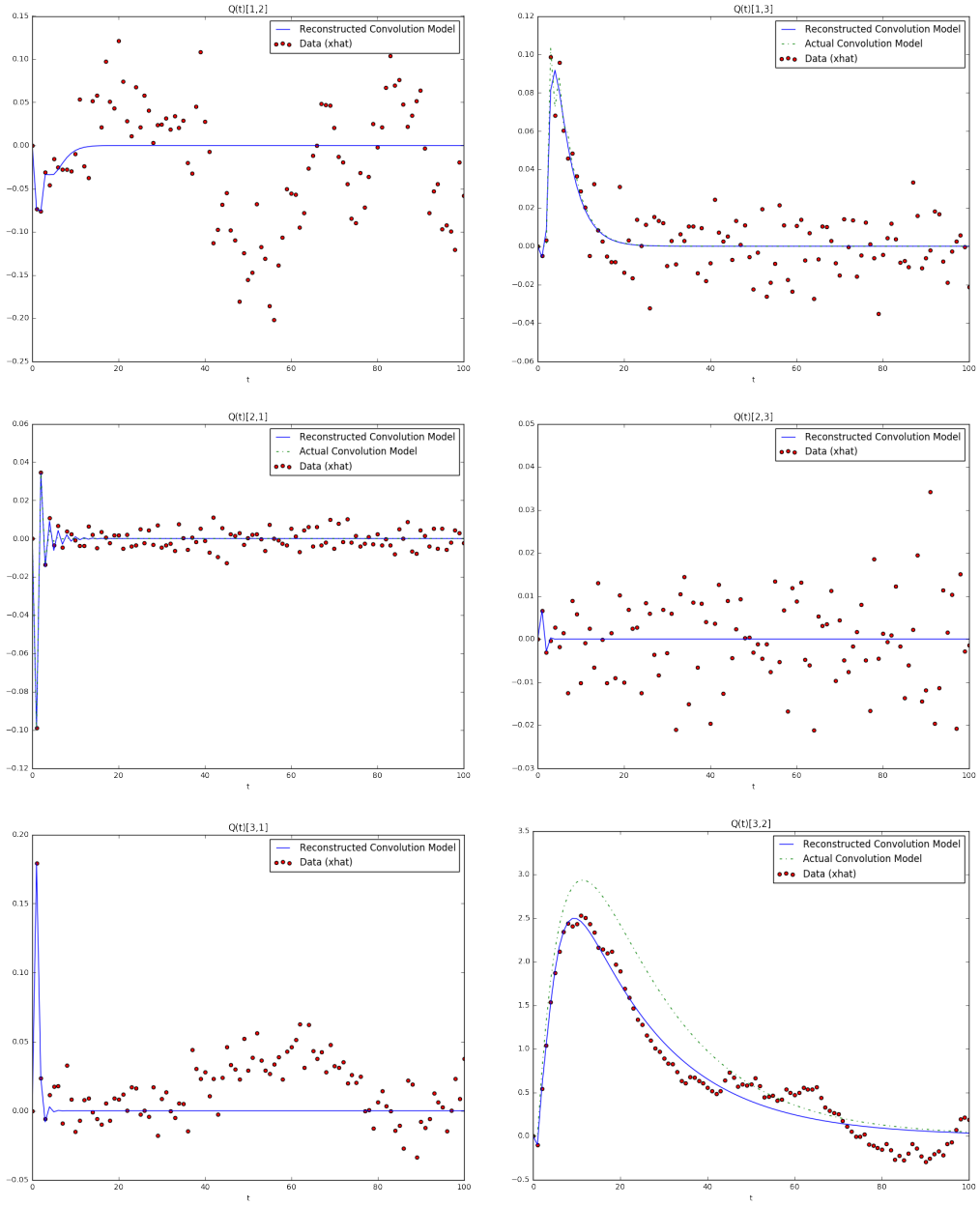


Figure 7.13 The impulse responses reconstructed using the VPNR Algorithm on noisy inputs and outputs. The green dashed line is the actual convolution model given by Equation (6.24) (for $Q_{12}(t)$, $Q_{23}(t)$, and $Q_{31}(t)$, this line is not drawn as the convolution model as the exact fit is at 0 for all t). The red dots are the values of the impulse response $Q_{ij}(t)$ contained in $\hat{x}$ and found using least squares. The blue line is the reconstructed convolution model from Equation (6.25) using the evolutionary algorithm to fit an equation of the form (6.20) to the red dots.

Link	Magnitude	Normalized Magnitude	Vulnerability	Normalized Vulnerability
$(1,2)$	0.561	0.008	0.157	0.005
$(1,3)$	0.602	0.009	1.382	0.044
$(2,1)$	0.144	0.002	18.791	0.595
$(2,3)$	0.091	0.001	31.606	1.000
$(3,1)$	0.203	0.003	0.300	0.009
$(3,2)$	69.235	1.000	0.097	0.003

Table 7.6 The magnitude ($\|Q_{ij}(z)\|_\infty$) and the vulnerability ($\|(I-Q(z))^{-1}_{ji}\|_\infty$) of links (i,j) in the $Q(z)$ of noisy inputs and outputs reconstructed using the VPNR Algorithm.

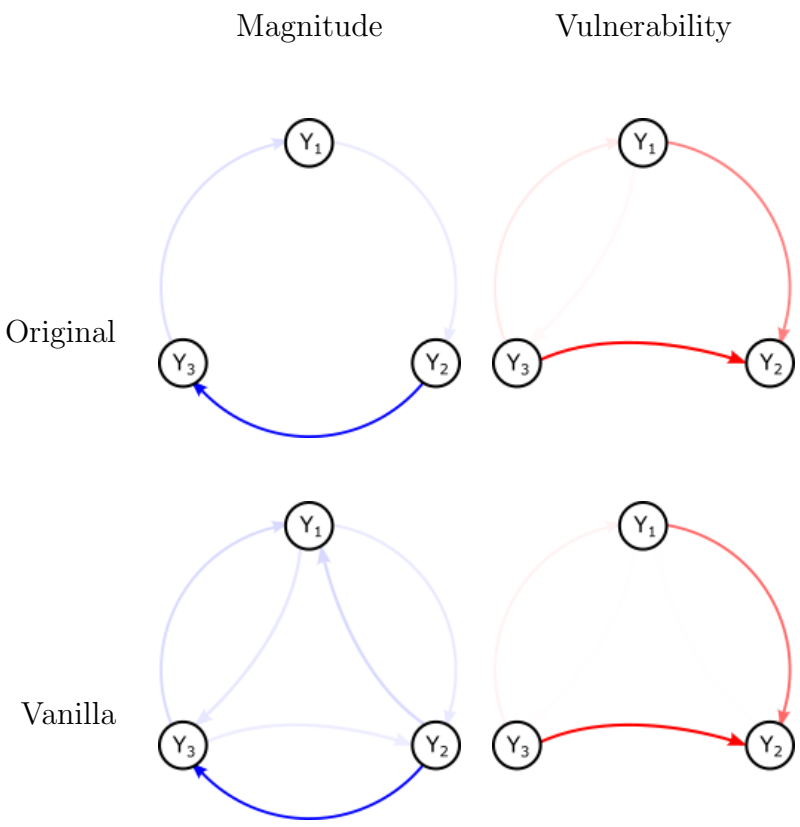


Figure 7.14 The magnitude (top left) and vulnerability (top right) of links in the original $Q(z)$ as given in Table 6.1 compared with the magnitude (bottom left) and vulnerability (bottom right) of links as given in Table 7.6.

We now valuate the RPNR Algorithm when $\gamma = 0.05$. The results of this experiment are given in Figures 7.15 and 7.16 and in Table 7.7. Once again, the algorithm performs well, though with some error. In this case, it can be said that these errors in the RPNR Algorithm are larger than those in the VPNR Algorithm.

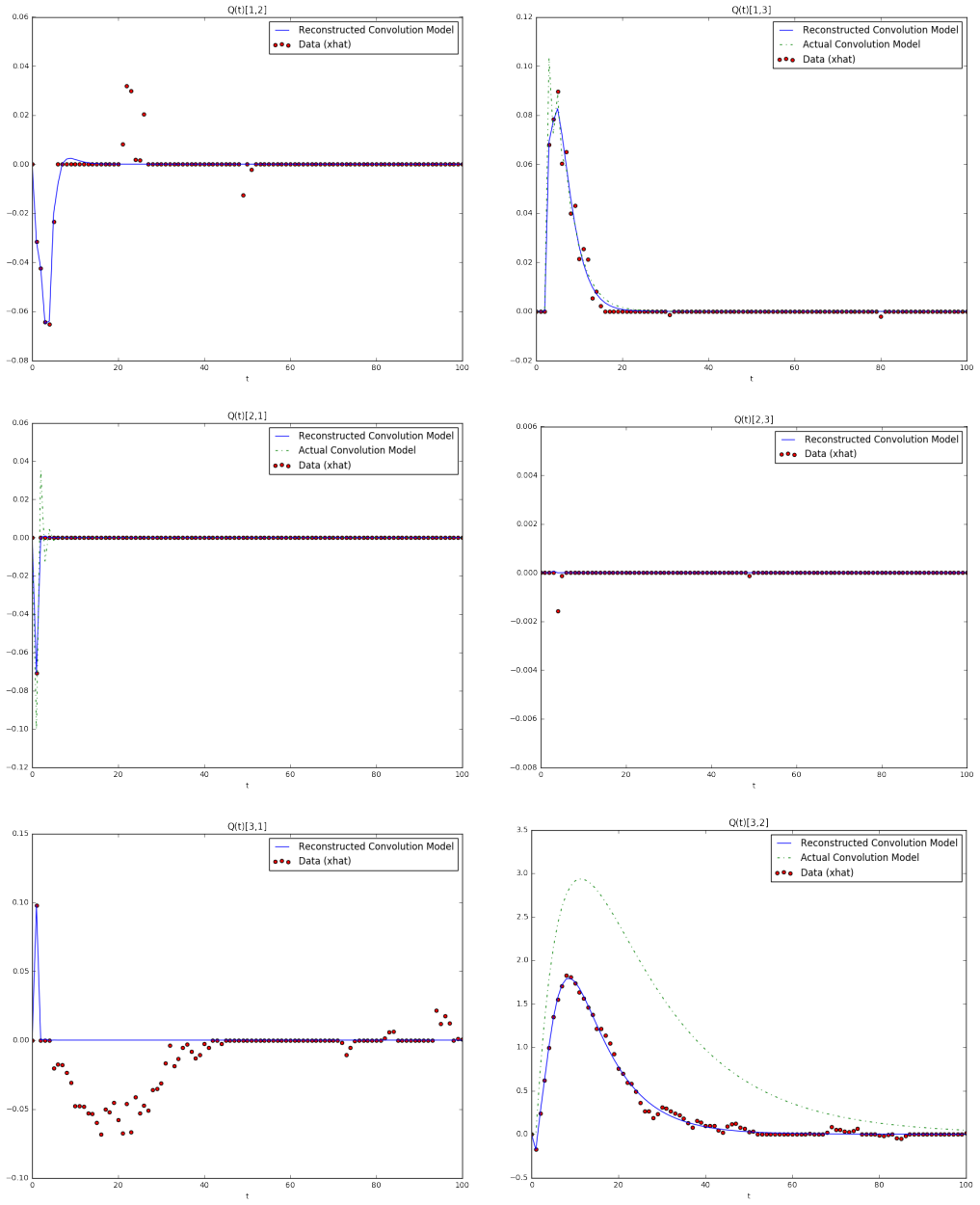


Figure 7.15 The impulse responses reconstructed using the RPNR Algorithm on noisy inputs and outputs. The green dashed line is the actual convolution model given by Equation (6.24) (for $Q_{12}(t)$, $Q_{23}(t)$, and $Q_{31}(t)$, this line is not drawn as the convolution model as the exact fit is at 0 for all t). The red dots are the values of the impulse response $Q_{ij}(t)$ contained in $\hat{x}$ and found using least squares. The blue line is the reconstructed convolution model from Equation (6.25) using the evolutionary algorithm to fit an equation of the form (6.20) to the red dots.

Link	Magnitude	Normalized Magnitude	Vulnerability	Normalized Vulnerability
$(1,2)$	0.134	0.005	0.263	0.007
$(1,3)$	0.519	0.019	3.286	0.086
$(2,1)$	0.087	0.003	17.152	0.448
$(2,3)$	0.033	0.001	38.285	1.000
$(3,1)$	0.051	0.002	1.386	0.036
$(3,2)$	27.094	1.000	0.180	0.005

Table 7.7 The magnitude ($\|Q_{ij}(z)\|_\infty$) and the vulnerability ($\|(I-Q(z))^{-1}_{ji}\|_\infty$) of links (i,j) in the $Q(z)$ of noisy inputs and outputs reconstructed using the RPNR Algorithm.

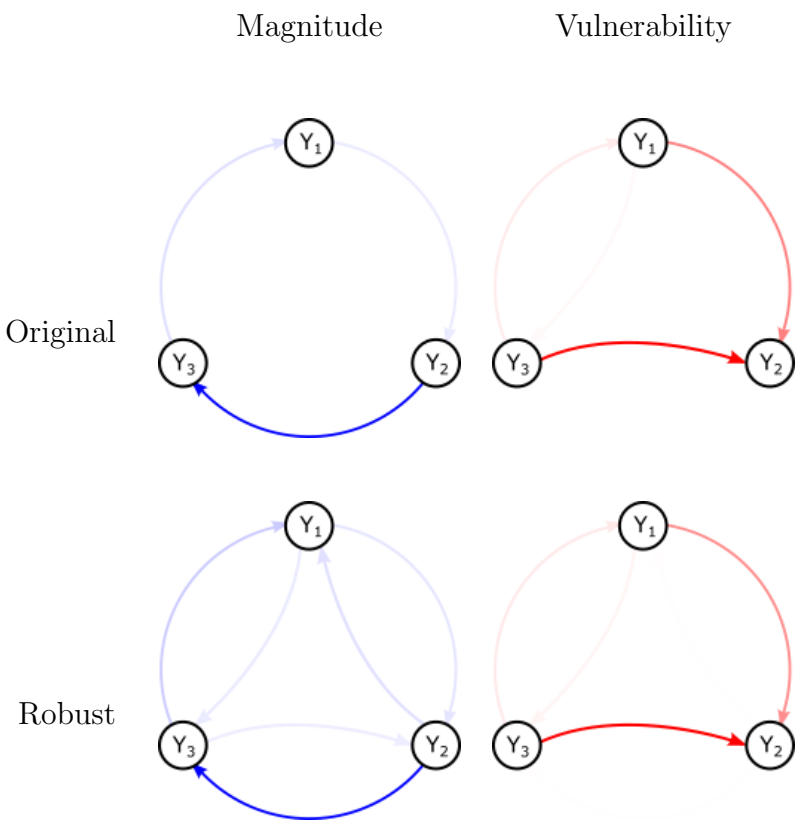


Figure 7.16 The magnitude (top left) and vulnerability (top right) of links in the original $Q(z)$ as given in Table 6.1 compared with the magnitude (bottom left) and vulnerability (bottom right) of links as given in Table 7.7.

Finally, we compute the average RMSE of the impulse responses η in order to compare the performances of the VPNR Algorithm with the RPNR Algorithm as the magnitude of the input and output noise γ increases. Figure 7.17 shows the results of this analysis. Again, we see that the VPNR Algorithm performs better with low levels of noise where the RPNR Algorithm performs better with high levels of noise, though again, the differences are fairly small.

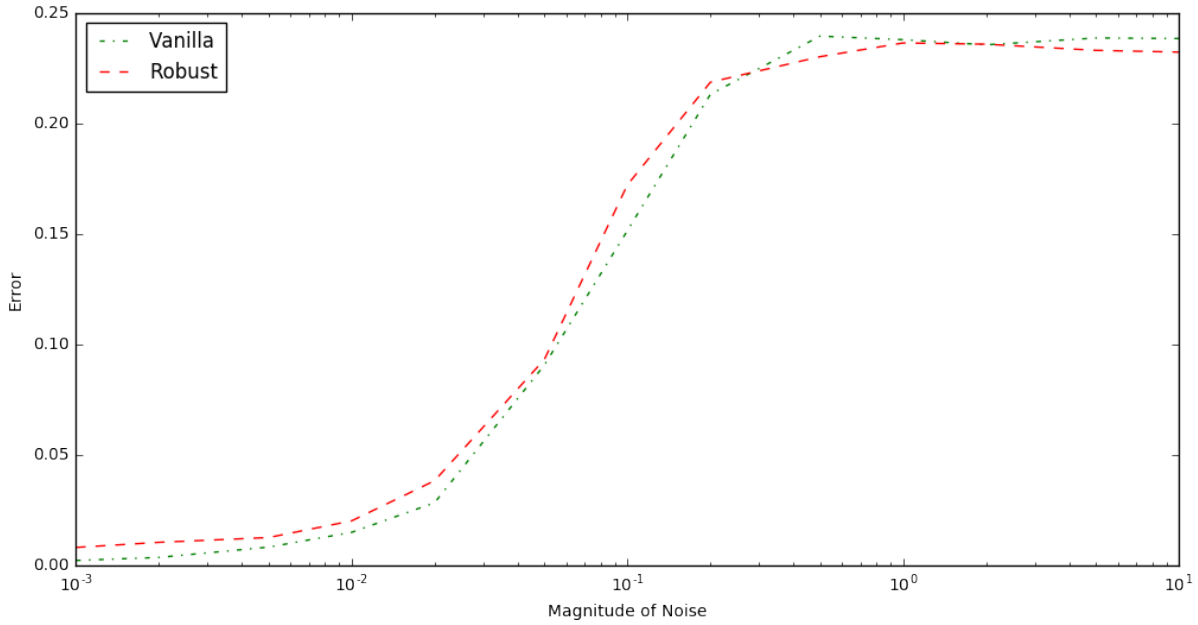


Figure 7.17 Effect of input and output noise on the ability of the VPNR and RPNR Algorithms to reconstruct from data.

7.5 On the Convergence of the Robust Passive Network Reconstruction Algorithm

As was done in Section 6.6 for the VPNR Algorithm, we can explore the convergence of the RPNR Algorithm as we change the values of r and T. Using the same measure of error and the same experiment designed introduced in Section 6.6, the quality of reconstruction using the RPNR Algorithm on non-noisy data is shown on the left of Figure 7.18. Similarly, the

quality of reconstruction using the RPNR Algorithm on noisy inputs and noisy outputs with $\gamma = 0.1$ is shown on the right of 7.18.

As before, as r and T increase, the error $\hat{e}$ converges to something near 0, meaning that though the network isn't reconstructed exactly, the reconstructed network is close and additional data doesn't change the network to which the RPNR Algorithm converges.

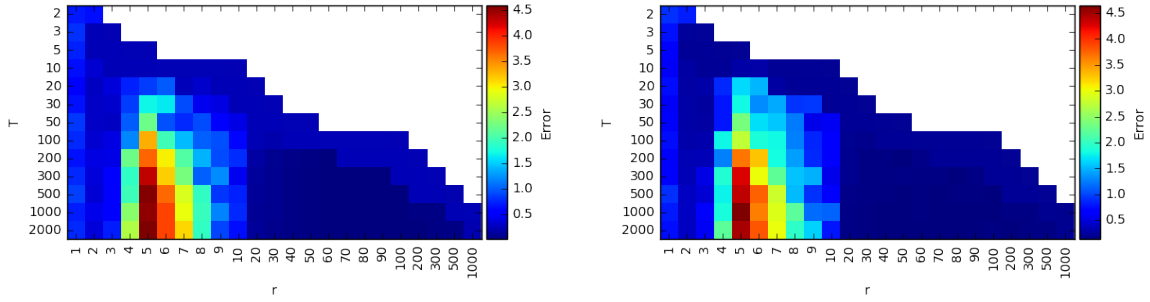


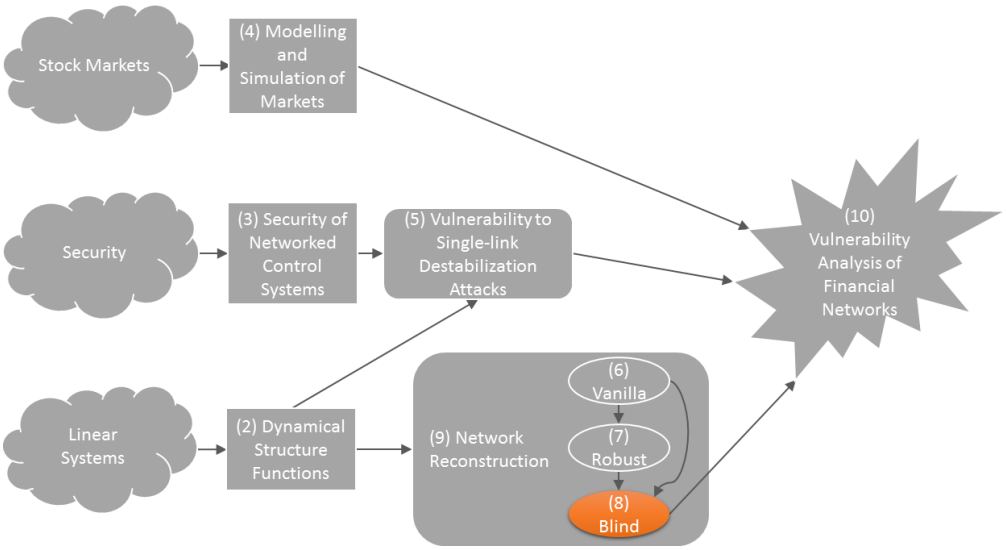
Figure 7.18 The quality of reconstruction of the RPNR Algorithm with non-noisy data (left) and with noisy inputs and outputs at $\gamma = 0.1$ (right) on the example in Section 7.4 at various levels of r and T.

7.6 Conclusions

In conclusion, we have presented an extension to the Vanilla Passive Network Reconstruction Algorithm, which we demonstrate to be more robust to additive noise on the input-output data used by the reconstruction process. This extension was made by replacing the ordinary least squares fitting in Step 6 of the VPNR Algorithm with a Lasso Regression solution.

Chapter 8

Blind Passive Network Reconstruction



In this chapter, we modify the VPNR Algorithm presented in Chapter 6 and the RPNR Algorithm presented in Chapter 7 to reconstruct a network when the inputs into the system are unknown. The algorithms presented in this chapter have been utilized in [11].

8.1 Problem Formulation

The blind formulation is similar to that presented in Section 6.2 (without noise) and in Section 7.1 (with noise). The key difference is that the inputs $\mathcal{D}_u$ are unknown and only $\mathcal{D}_y$ is measured.

8.2 The Blind Passive Network Reconstruction Algorithms

We will modify both the VPNR Algorithm and the RPNR algorithm to manage the case where inputs are not measured. The resultant algorithms we call the Blind Vanilla Passive Network Reconstruction (B-VPNR) Algorithm and the Blind Robust Passive Network Reconstruction (B-RPNR) Algorithm. The difference between the B-VPNR and the B-RPNR Algorithms is the same as the difference between the VPNR and the RPNR algorithms, namely the use of a different regression technique in Step 6 of the algorithm.

By not measuring the inputs, we must introduce variations to various steps of the VPNR and RPNR Algorithms. Steps 1 (we only prepare the output data) and 3 ($\hat{y}$ and $\hat{L}$ are constructed differently, as will be described below). Furthermore, we only extract and reconstruct Q (as opposed to both Q and P) in Steps 8, 7, and 9. Finally, the level of a priori information required to reconstruct using the blind algorithms is not presently known. As such, step 5 will require variations, but may potentially be skipped so long as the unmeasured inputs meets certain conditions.

As the largest change from the VPNR Algorithm to the B-VPNR Algorithm is in Step 3, we focus on that change here. For now, let's assume that we can measure the outputs with no error. In particular, let the network dynamics be given by

$$Y(z) = Q(z)Y(z) + P(z)\Delta(z), \tag{8.1}$$

where $\Delta(z)$ represents the unmeasured inputs. We will make some assumptions on the structure of $P(z)$, but we defer that discussion until later. We start by following the same procedure as the VPNR and RPNR Algorithms up to step .

Now partition $\hat{x}$ and $\hat{M}$ such that

$$\hat{y} = \begin{bmatrix} \hat{M}_Q & \hat{M}_P \end{bmatrix} \begin{bmatrix} \hat{x}_Q \\ \hat{x}_P \end{bmatrix}, \tag{8.2}$$

where $\hat{x}_Q$ contains only entries in $Q(t)$, $\hat{x}_P$ contains only entries in P, and $\hat{M}$ is partitioned commensurately. Note that with this partition, $\hat{M}_Q$ only contains values the known values from y and $\hat{M}_P$ only contains the unknown values from $\Delta(z)$. Note, also, that we only care about reconstructing the entries in $\hat{x}_Q$ and can leave the entries of $\hat{x}_P$ as unknown. As such, (8.2) can be rewritten as

$$\begin{aligned}\hat{y} &= \hat{M}_Q\vec{x}_Q + \hat{M}_P\vec{x}_p \\ &= \hat{M}_Q\vec{x}_Q + \zeta,\end{aligned} \tag{8.3}$$

where $\zeta \in \mathbb{R}^{(T-1)p}$ is unknown but constant, and $\hat{M}_Q$ takes the following form:

$$\hat{M}_Q = \begin{bmatrix} y(1)^\mathsf{T} & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & \ddots & 0 & \cdots & 0 & \ddots & 0 \\ 0 & 0 & y(1)^\mathsf{T} & \cdots & 0 & 0 & 0 \\ & \vdots & & & & \vdots & \\ y(r)^\mathsf{T} & 0 & 0 & \cdots & y(1)^\mathsf{T} & 0 & 0 \\ 0 & \ddots & 0 & \cdots & 0 & \ddots & 0 \\ 0 & 0 & y(r)^\mathsf{T} & \cdots & 0 & 0 & y(1)^\mathsf{T} \\ & \vdots & & & & \vdots & \\ y(T-1)^\mathsf{T} & 0 & 0 & \cdots & y(T-r)^\mathsf{T} & 0 & 0 \\ 0 & \ddots & 0 & \cdots & 0 & \ddots & 0 \\ 0 & 0 & y(T-1)^\mathsf{T} & \cdots & 0 & 0 & y(T-r)^\mathsf{T} \end{bmatrix}. \tag{8.4}$$

Redefine $\hat{M}_Q$ and define $\hat{x}_Q$ as the matrix and vector corresponding to the original $\hat{M}_Q$ and $\vec{x}_Q$ where all of the diagonal (known to be zero) entries of $Q(z)$ are removed. We wish now to choose the $\hat{x}_Q$ that best fits this data, or in other words, we wish to choose $\hat{x}_Q$ such that the error

$$e = \hat{y} - \left(\hat{M}_Q\hat{x}_Q + \zeta\right) \tag{8.5}$$

is minimized in the two-norm sense. Note that, since ζ is constant, minimizing e is the same as minimizing $\hat{e} = e + \zeta$. As such, (8.5) reduces to minimizing the error[1]

$$\hat{e} = e + \zeta = \hat{y} - \hat{M}_Q \hat{x}_Q. \tag{8.6}$$

The solution is given by

$$\hat{x}_Q^* = \arg\min_{\hat{x}_Q} \|\hat{e}\|_2 = \arg\min_{\hat{x}_Q} \|\hat{y} - \hat{M}_Q \hat{x}_Q\|_2, \tag{8.7}$$

which can be solved using ordinary least squares.

Once $\hat{x}_Q^*$ is chosen, the B-VPNR Algorithm proceeds exactly like the VPNR and the RPNR Algorithms starting at step 8. Note also that we can create a robust version of this algorithm– which we call the Blind Robust Network Reconstruction (B-RPNR) Algorithm–by replacing Equation (8.7) with

$$\hat{x}_Q^* = \arg\min_{\hat{x}_Q} \|\hat{y} - \hat{L}_Q \hat{x}_Q\|_2 + \alpha \|\hat{x}_Q\|_1, \tag{8.8}$$

where α is chosen to maximize the Akaike Information Criterion.

8.3 Assumptions Necessary for Reconstruction

Unfortunately, since we are not measuring the inputs, we are not able to reconstruct the network exactly, though we can still find approximations (see Section 8.4). The assumptions required to reconstruct are similar to those discussed in Sections 6.4 and 7.3, but with some minor variations. They are as follows:

[1] Note that (8.7) can also be interpreted as minimizing the unmeasured input dynamics ζ. This is sensible since we want to explain the dynamics of y as much as possible using the dynamics of $Q(t)$, or in other words, minimizing the necessity of $P(t)$ as much as possible.

- **Linearity:** The underlying network generating the output data must have linear dynamics. This is due to the fact that we are reconstructing DSFs, which presently only represent linear dynamics.
- **Stability:** The underlying network generating the output data must be stable. This allows the Toeplitz representation of the network in (6.10)–and hence $\hat{L}$ in (8.4)–to be finite dimensional.
- **Strict Causality:** The underlying network generating the output data must be strictly causal, meaning changes in the present only make an impact on the network strictly in the future. This also implies that the impulse responses at time $t = 0$ are $Q_{ij}(0) = 0$.
- **Informativity Conditions:** The informativity conditions are presently unknown, though they may have something to do with independence on the noise ζ.
- **Small Levels of Noise:** The outputs must be measured with no more than small levels of output noise. As shown in the next section, reconstruction is sensitive to noise, and so ideally, there would be no noise.
- **Richness of Data:** We require that the input-output data be "rich enough" to reconstruct. This condition can be checked directly. The data is "rich enough" if $\hat{M}$ is injective and if $\hat{y}$ is in the range of $\hat{M}$.
- **Large Enough r:** We require that r be chosen large enough to capture the full dynamics of the system.
- **Large Enough T:** We require that enough data be collected that we don't overfit the least squares model. Furthermore, T should be strictly larger than r.

8.4 Numeric Examples

Once again, we return to the numeric example introduced in Section 6.5, comparing the performance of the B-VPNR and the B-RPNR in situations where there is no noise on the output data and where there is noise.

8.4.1 Blind Reconstruction with No Noise

We first demonstrate the B-VPNR Algorithm on our running example with no noise on the outputs. The results of this experiment are contained in Figures 8.1 and 8.2, as well as Table 8.1. Even with no noise on the system, the Vanilla-Blind algorithm was incapable of reconstructing the network exactly. However, the network it did reconstruct was a good approximation of the actual network.

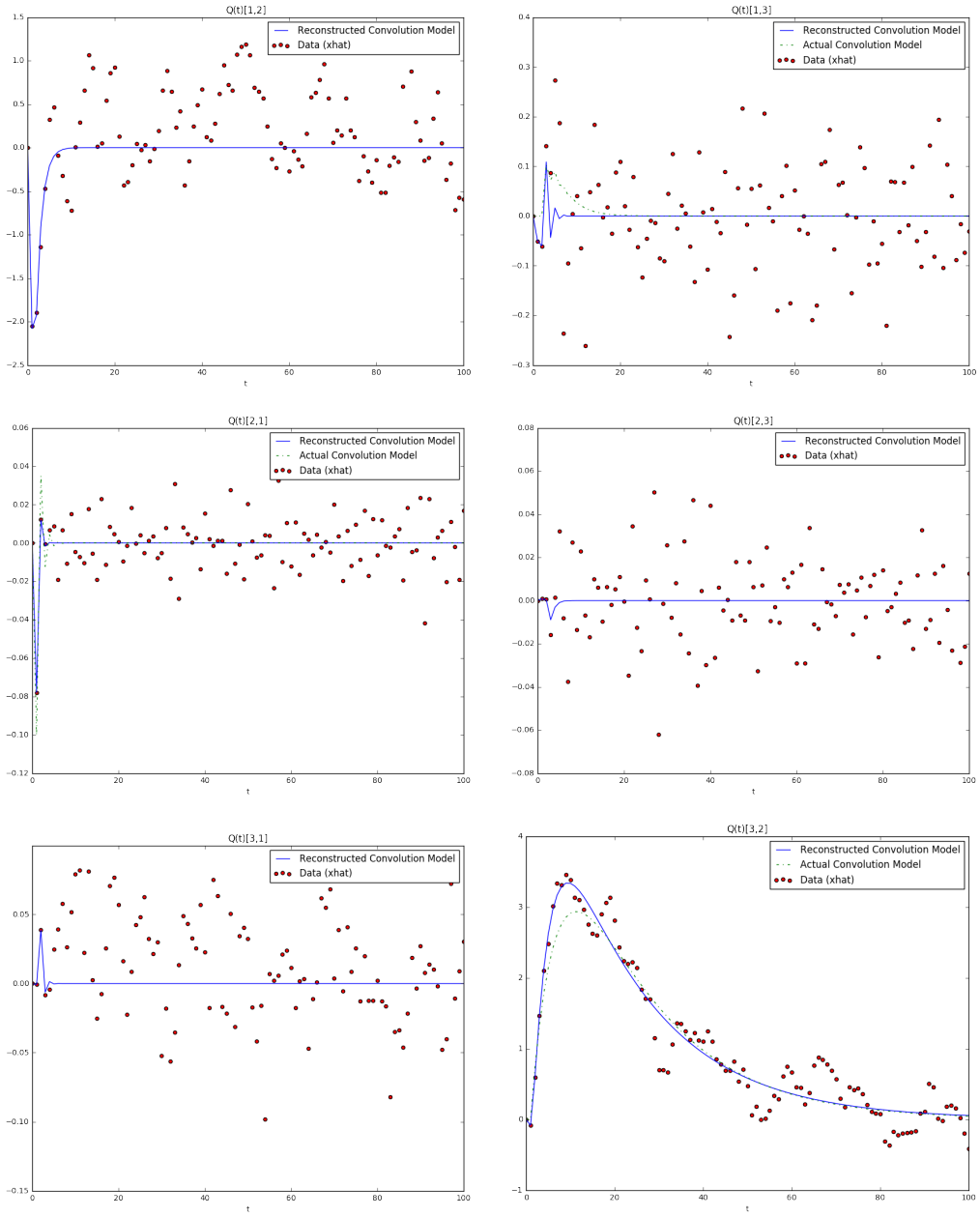


Figure 8.1 The impulse responses reconstructed using the B-VPNR Algorithm on non-noisy data. The green dashed line is the actual convolution model given by Equation (6.24) (for $Q_{12}(t)$, $Q_{23}(t)$, and $Q_{31}(t)$, this line is not drawn as the convolution model as the exact fit is at 0 for all t). The red dots are the values of the impulse response $Q_{ij}(t)$ contained in $\hat{x}$ and found using least squares. The blue line is the reconstructed convolution model from Equation (6.25) using the evolutionary algorithm to fit an equation of the form (6.20) to the red dots.

Link	Magnitude	Normalized Magnitude	Vulnerability	Normalized Vulnerability
$(1,2)$	5.644	0.061	0.060	0.001
$(1,3)$	0.173	0.002	2.683	0.034
$(2,1)$	0.058	0.001	8.594	0.107
$(2,3)$	0.013	0.000	80.072	1.000
$(3,1)$	0.059	0.001	0.174	0.002
$(3,2)$	92.502	1.000	0.016	0.000

Table 8.1 The magnitude ($\|Q_{ij}(z)\|_\infty$) and the vulnerability ($\|(I-Q(z))^{-1}_{ji}\|_\infty$) of links (i,j) in the $Q(z)$ without measuring the inputs reconstructed using the B-VPNR Algorithm.

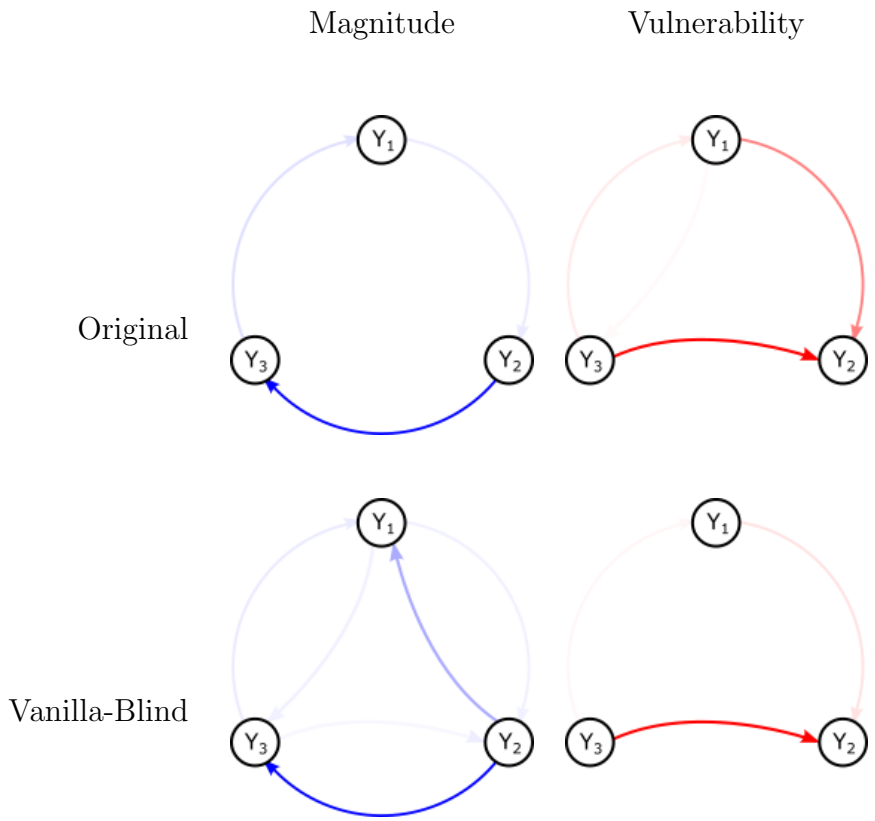


Figure 8.2 The magnitude (top left) and vulnerability (top right) of links in the original $Q(z)$ as given in Table 6.1 compared with the magnitude (bottom left) and vulnerability (bottom right) of links as given in Table 8.1. The darkness of each link is proportional to the normalized magnitudes and vulnerabilities, raised to the 0.4'th power in order to emphasize smaller links.

We now demonstrate the B-RPNR Algorithm on our running example with no noise on the outputs. The results of this experiment are contained in Figures 8.3 and 8.4, as well as Table 8.2. Again, this algorithm was incapable of reconstructing the network exactly, though it still performed reasonably well.

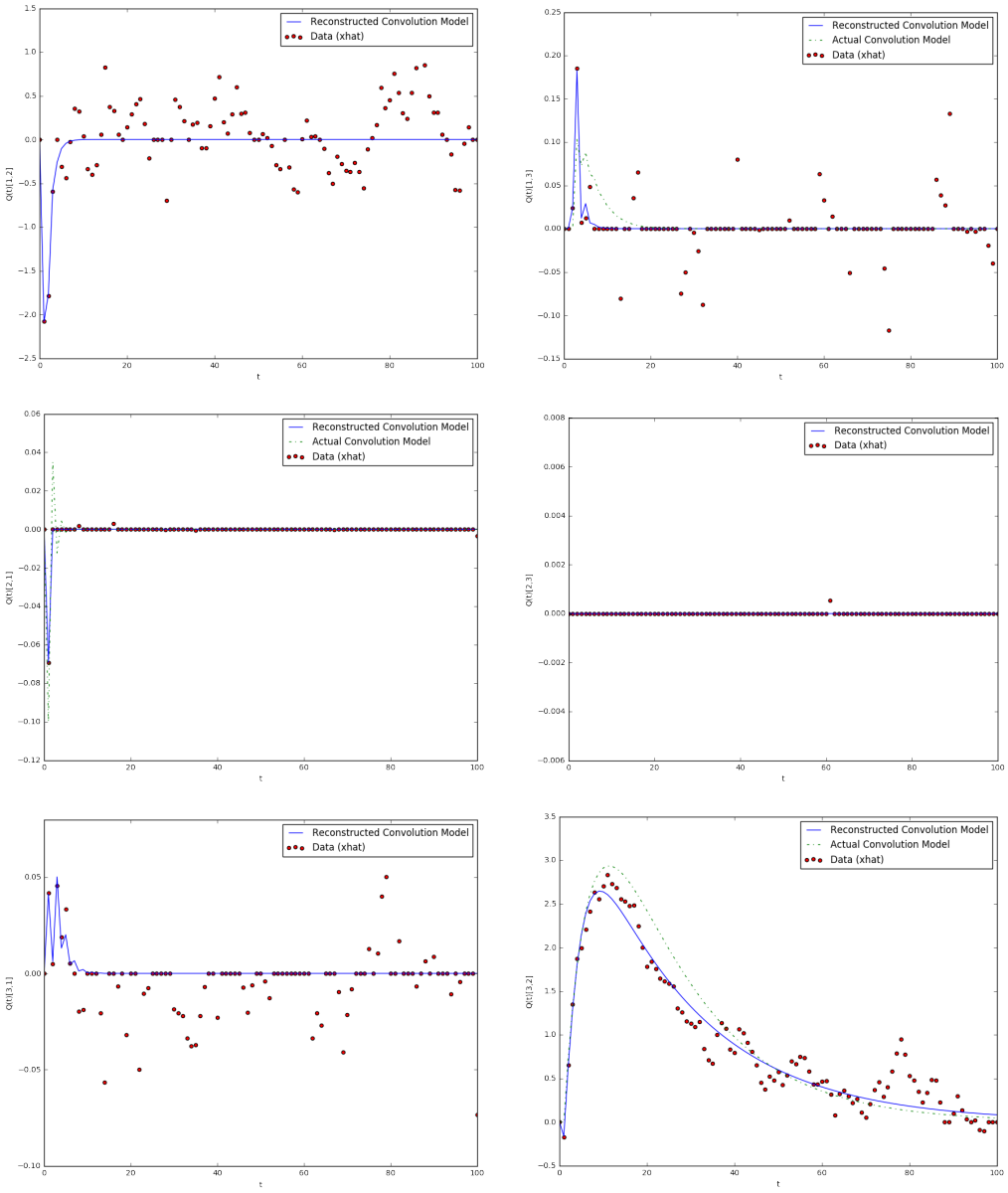


Figure 8.3 The impulse responses reconstructed using the B-RPNR Algorithm on non-noisy data. The green dashed line is the actual convolution model given by Equation (6.24) (for $Q_{12}(t)$, $Q_{23}(t)$, and $Q_{31}(t)$, this line is not drawn as the convolution model as the exact fit is at 0 for all t). The red dots are the values of the impulse response $Q_{ij}(t)$ contained in $\hat{x}$ and found using least squares. The blue line is the reconstructed convolution model from Equation (6.25) using the evolutionary algorithm to fit an equation of the form (6.20) to the red dots.

Link	Magnitude	Normalized Magnitude	Vulnerability	Normalized Vulnerability
$(1,2)$	4.723	0.054	0.099	0.002
$(1,3)$	0.270	0.003	2.672	0.052
$(2,1)$	0.051	0.001	18.334	0.354
$(2,3)$	0.001	0.000	51.850	1.000
$(3,1)$	0.137	0.002	0.507	0.010
$(3,2)$	87.332	1.000	0.025	0.000

Table 8.2 The magnitude ($\|Q_{ij}(z)\|_\infty$) and the vulnerability ($\|(I-Q(z))^{-1}_{ji}\|_\infty$) of links (i,j) in the $Q(z)$ without measuring the inputs reconstructed using the B-RPNR Algorithm.

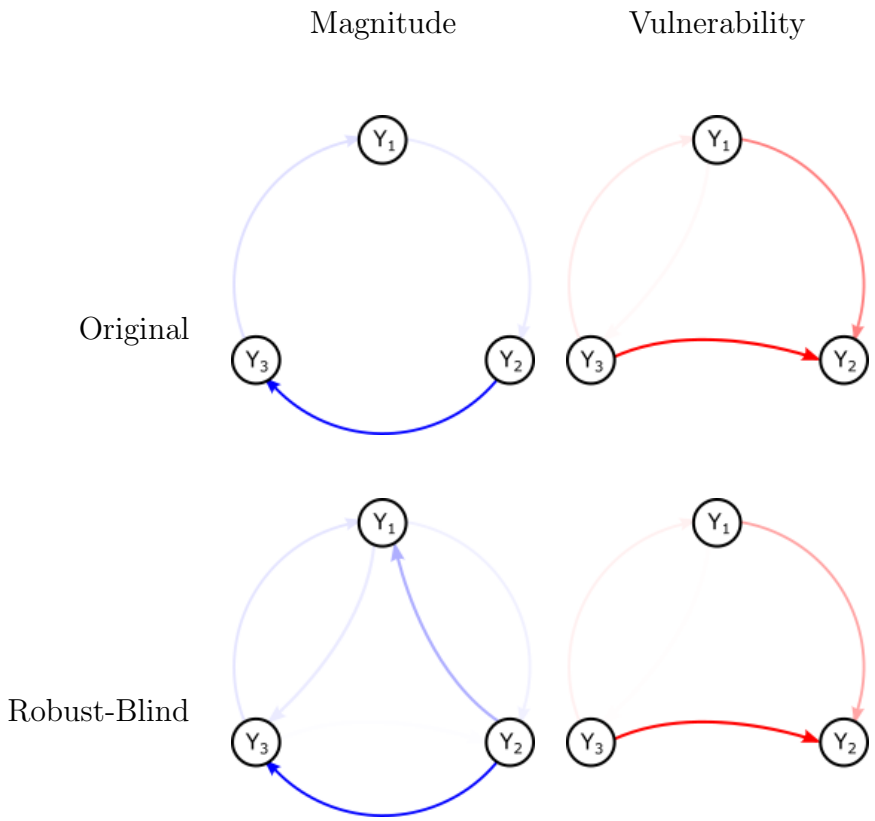


Figure 8.4 The magnitude (top left) and vulnerability (top right) of links in the original $Q(z)$ as given in Table 6.1 compared with the magnitude (bottom left) and vulnerability (bottom right) of links as given in Table 8.2. The darkness of each link is proportional to the normalized magnitudes and vulnerabilities, raised to the 0.4'th power in order to emphasize smaller links.

8.4.2 Blind Reconstruction with Noisy Outputs

We now demonstrate blind reconstruction using the B-VPNR Algorithm with noise on the outputs at $\gamma = 0.01$. The results of this experiment are contained in Figures 8.5 and 8.6, as well as Table 8.3. Notice that, even with a small magnitude of noise, the algorithm performed significantly worse than before, adding significant dynamics on non-zero links and missing the dynamics on link $(1, 3)$. Nonetheless, it was still able to approximate some of the features of the network.

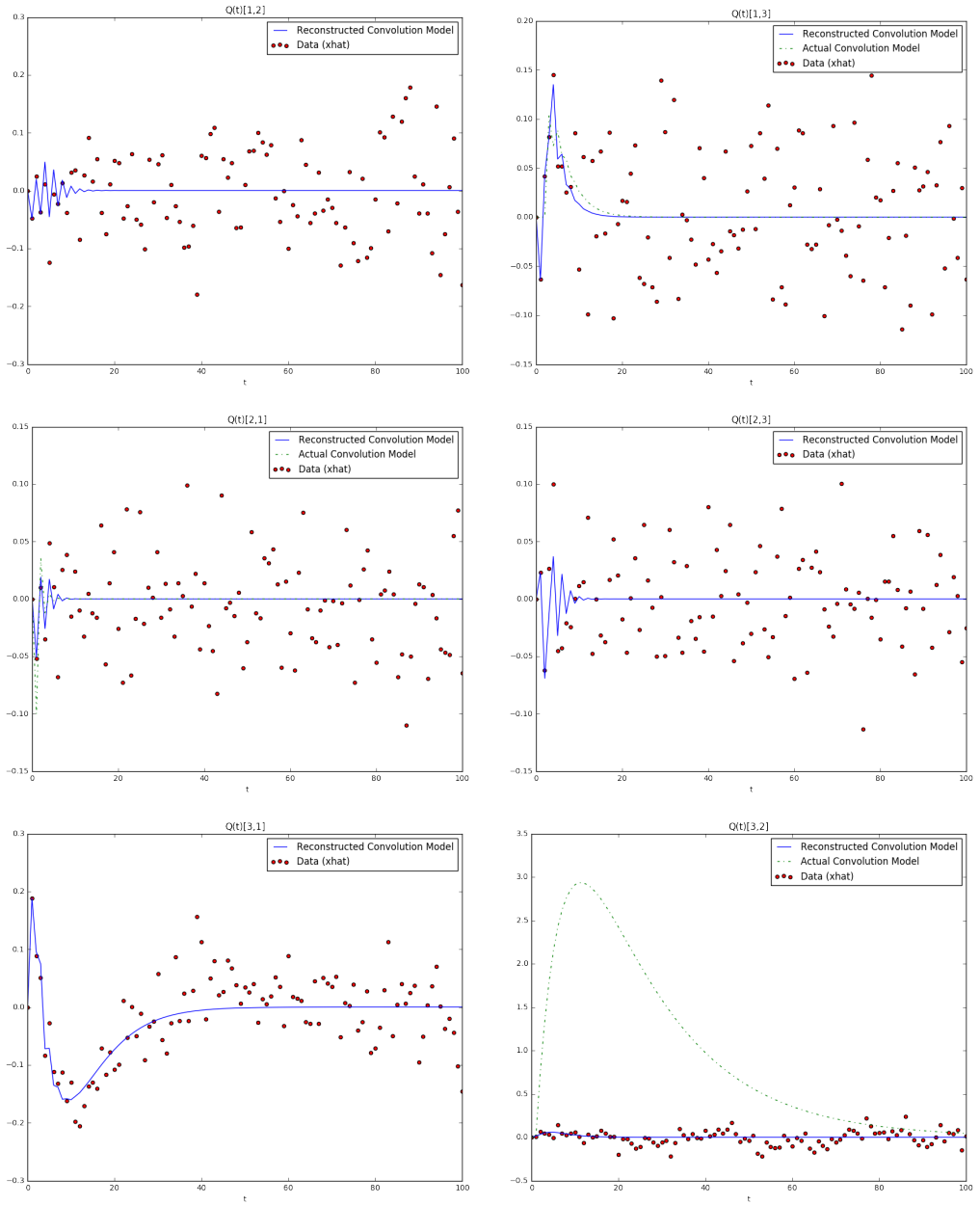


Figure 8.5 The impulse responses reconstructed using the B-VPNR Algorithm on noisy outputs. The green dashed line is the actual convolution model given by Equation (6.24) (for $Q_{12}(t)$, $Q_{23}(t)$, and $Q_{31}(t)$, this line is not drawn as the convolution model as the exact fit is at 0 for all t). The red dots are the values of the impulse response $Q_{ij}(t)$ contained in $\hat{x}$ and found using least squares. The blue line is the reconstructed convolution model from Equation (6.25) using the evolutionary algorithm to fit an equation of the form (6.20) to the red dots.

Link	Magnitude	Normalized Magnitude	Vulnerability	Normalized Vulnerability
$(1,2)$	0.231	0.139	0.140	0.047
$(1,3)$	0.130	0.259	2.994	1.000
$(2,1)$	0.103	0.062	0.357	0.199
$(2,3)$	0.141	0.085	0.850	0.284
$(3,1)$	1.659	1.000	0.906	0.303
$(3,2)$	0.428	0.258	0.145	0.049

Table 8.3 The magnitude ($\|Q_{ij}(z)\|_\infty$) and the vulnerability ($\|(I-Q(z))^{-1}_{ji}\|_\infty$) of links (i,j) in the $Q(z)$ without measuring the inputs and with noise on the outputs reconstructed using the B-VPNR Algorithm.

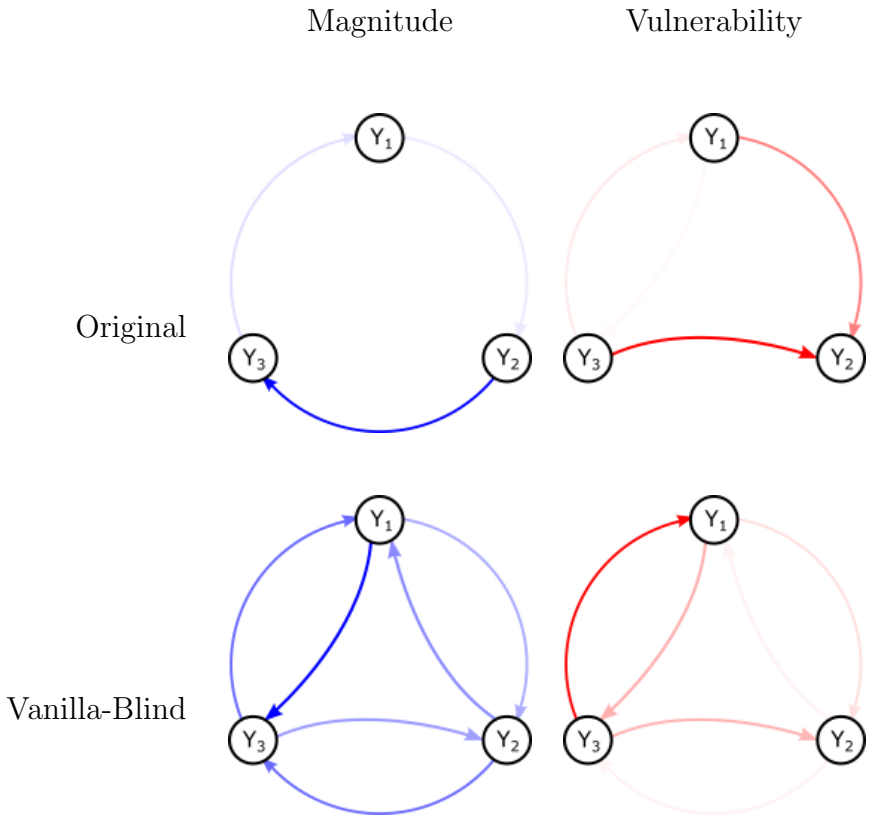


Figure 8.6 The magnitude (top left) and vulnerability (top right) of links in the original $Q(z)$ as given in Table 6.1 compared with the magnitude (bottom left) and vulnerability (bottom right) of links as given in Table 8.3. The darkness of each link is proportional to the normalized magnitudes and vulnerabilities, raised to the 0.4'th power in order to emphasize smaller links.

We now demonstrate blind reconstruction using the B-RPNR Algorithm with noise on the outputs at $\gamma = 0.01$. The results of this experiment are contained in Figures 8.7 and 8.8, as well as Table 8.4. This algorithm performed somewhat better than the Vanilla-Blind Algorithm with noise, driving many of the dynamics that should have been zero down to zero. However, it still fails to capture link $(3, 2)$ and misses some of the higher-order dynamics on $(1, 3)$ and $(2, 1)$. Nonetheless, the approximation of the network is still fairly reasonable.

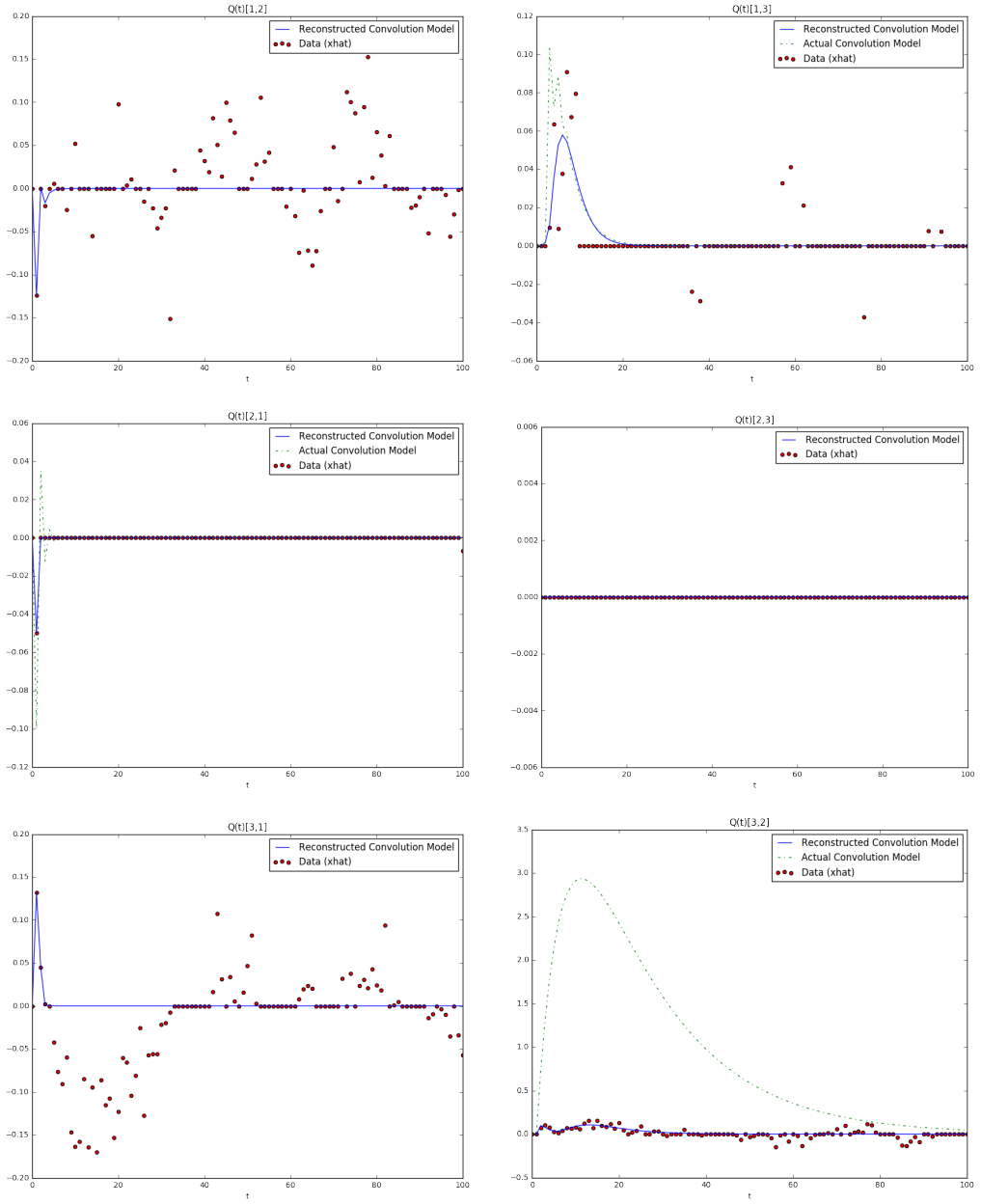


Figure 8.7 The impulse responses reconstructed using the B-RPNR Algorithm on noisy outputs. The green dashed line is the actual convolution model given by Equation (6.24) (for $Q_{12}(t)$, $Q_{23}(t)$, and $Q_{31}(t)$, this line is not drawn as the convolution model as the exact fit is at 0 for all t). The red dots are the values of the impulse response $Q_{ij}(t)$ contained in $\hat{x}$ and found using least squares. The blue line is the reconstructed convolution model from Equation (6.25) using the evolutionary algorithm to fit an equation of the form (6.20) to the red dots.

Link	Magnitude	Normalized Magnitude	Vulnerability	Normalized Vulnerability
$(1,2)$	0.148	0.100	0.073	0.051
$(1,3)$	0.227	0.153	0.234	0.162
$(2,1)$	0.067	0.045	0.373	0.258
$(2,3)$	0.024	0.016	1.446	1.000
$(3,1)$	0.192	0.129	0.222	0.153
$(3,2)$	1.483	1.000	0.036	0.025

Table 8.4 The magnitude ($\|Q_{ij}(z)\|_\infty$) and the vulnerability ($\|(I-Q(z))^{-1}_{ji}\|_\infty$) of links (i,j) in the $Q(z)$ without measuring the inputs and with noise on the outputs reconstructed using the B-RPNR Algorithm.

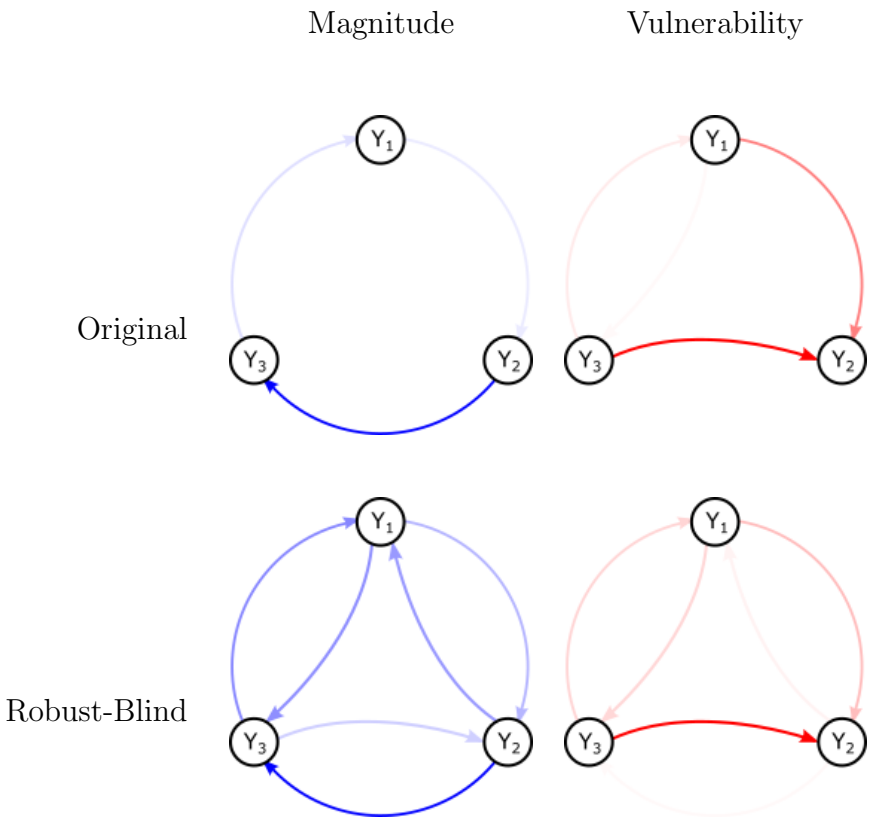


Figure 8.8 The magnitude (top left) and vulnerability (top right) of links in the original $Q(z)$ as given in Table 6.1 compared with the magnitude (bottom left) and vulnerability (bottom right) of links as given in Table 8.4. The darkness of each link is proportional to the normalized magnitudes and vulnerabilities, raised to the 0.4'th power in order to emphasize smaller links.

Finally, we compare the error η of the Vanilla-Blind and the Robust-Blind Network Reconstruction Algorithms as noise increases. The results of this experiment are shown in Figure 8.9. Notice how, even at it's best, the Robust-Blind Algorithm performs about as well as the Robust Algorithm on large magnitudes of noise on measured inputs as shown in Figure 7.7, indicating that both Blind Reconstruction Algorithms are extremely sensitive to noise on the output signals. Also notice that the Robust-Blind Algorithm outperforms the Vanilla-Blind Algorithm except for a window of noise at $0.1 \leq \gamma \leq 1$.

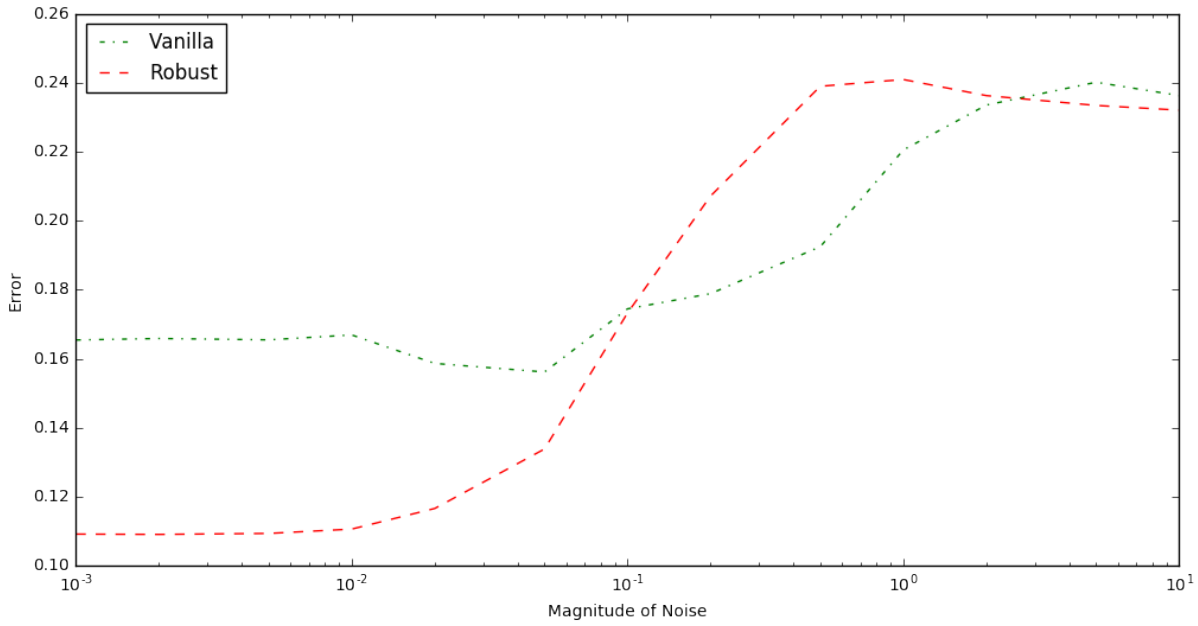


Figure 8.9 Effect of output noise on the ability of the B-VPNR and the B-RPNR Algorithms to reconstruct from data without measuring the inputs.

8.5 On the Convergence of the Blind Passive Reconstruction Algorithms

As was done in Section 6.6 for the VPNR Algorithm and Section 7.5 for the RPNR Algorithm, we can explore the convergence of the blind algorithms as we change the values of r and T. Using the same measure of error and the same experiment designed introduced in Section 6.6, the quality of reconstruction using the blind algorithms on non-noisy and noisy data is shown in Figure 8.10.

As before, as r and T increase, the error $\hat{e}$ converges to some value close to 0, meaning that though the network isn't reconstructed exactly, it gets close and increased data doesn't change the network to which the algorithms converge.

8.6 Conclusions

In conclusion, we have presented an algorithm for the reconstruction of dynamical structure functions where inputs are not measured. Since inputs are not measured, we are not able to

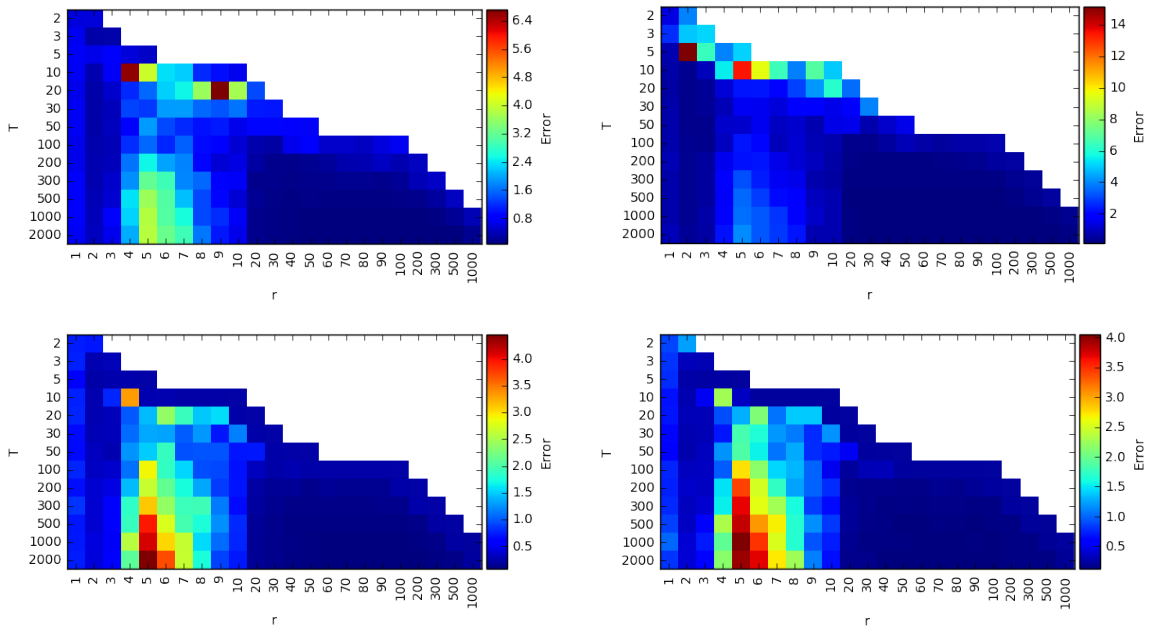
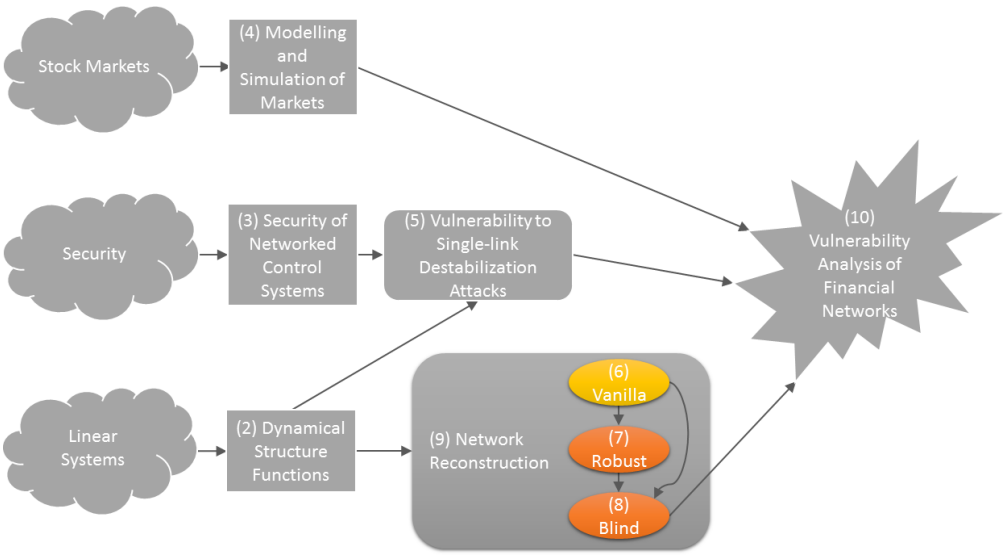


Figure 8.10 The quality of reconstruction of the B-VPNR Algorithm with non-noisy data (top left) and with noisy outputs at $\gamma = 0.05$ (top right) as well as the B-RPNR Algorithm on non-noisy data (bottom left) and with noisy outputs at $\gamma = 0.05$ (bottom right) on the example in Section 8.4 at various levels of r and T. Note that the error scale for each subfigure is different.

recover the exact network; however, we are still able to recover a reasonable approximation of the network.

Chapter 9

Open Questions in Passive Network Reconstruction



Research into network reconstruction is still a work in progress. This thesis is primarily intended as a proof of concept, demonstrating and expanding the basic theory of passive network reconstruction and applying it to actual financial data. There is still work to do until passive network reconstruction can be considered a solved problem. Areas of future research include the following:

- **Proof of Informativity Conditions on Vanilla and Robust:** For both the VPNR and the RPNR Algorithms, it was stated that the a priori information known about the network system must meet certain informativity conditions–specifically conditions that are informationally equivalent to target specificity–in order to reconstruct.

  The reason it is assumed that these informativity conditions are necessary and sufficient is that they have been proven to be the necessary and sufficient for the active

reconstruction of frequency-domain networks [16]. If they do not hold for the passive reconstruction of time-domain networks, then a frequency domain network can violate these conditions and still reconstruct by converting to the time domain and then back to the frequency domain. Furthermore, preliminary experiments (not shown here) show that if these conditions are even slightly violated, then the reconstructed network can be arbitrarily bad. For these reasons, we assume that the necessary and sufficient informativity conditions for passive reconstruction on time-domain are the same as for active reconstruction on frequency-domain networks.

Nonetheless, it would be valuable to develop a proof of these necessary and sufficient conditions directly from the passive formulation. Not only would it solidify the passive network reconstruction theory, but such a proof would provide more understanding as to the necessary and sufficient conditions for blind reconstruction (next bullet).

- **Exploration of Informativity Conditions:** As mentioned in the previous point, it is presently unknown how wrong a reconstructed network will be if the informativity conditions are even slightly violated. Some preliminary experiments suggest that the networks can be arbitrarily bad, but further investigation is warranted.
- **Informativity Conditions on Blind:** It is presently unknown what the necessary and sufficient conditions on the unknown inputs are for blind reconstruction. They are likely related to the independence of the noise generated by the unknown inputs. A proof of the necessary and sufficient conditions for the vanilla and robust algorithms may be useful in finding these conditions.
- **Proof of Convergence:** Evidence was provided that the algorithms converge to the proper network as r and T increase. However, there is presently no proof of this.
- **Reconstruction Validation:** Presently, there is no scheme for validating whether a reconstructed network is close to the actual network that generated the data, other

than knowing what the original network was to begin with. We use a known network in the previous chapters to demonstrate that the network reconstruction algorithms are functioning as desired; however, it would be valuable to determine how good the reconstructed network is even if the actual network is unknown.

One potential method for doing this may be to use the reconstructed network to predict future outputs given present inputs, and then to split the input-output data into training and validation sets. Careful thought is necessary in order to perform the two tasks of splitting the data and generating predictions.

- **Alternative Blind Algorithm:** Presently, the blind algorithm treats the noise created by unknown inputs as the same as the error in the least squares model. An alternative scheme may be to separate the error and the noise by augmenting the $\hat{M}$ matrix with the identity (in other words, replacing the side of $\hat{L}$ corresponding to $P(t)$ instead of dropping it). Additional research will be needed to determine whether this technique works and whether it performs better than the proposed technique.

- **Alternative Robust Algorithms:** This work proposes one robust algorithm by replacing the use of fitting the impulse responses by an ordinary least squares algorithm with a lasso regression algorithm. Other robust algorithms may be generated by using alternative regression techniques instead. Future work may evaluate which of these alternative schemes performs better under different assumptions on the noise on the data.

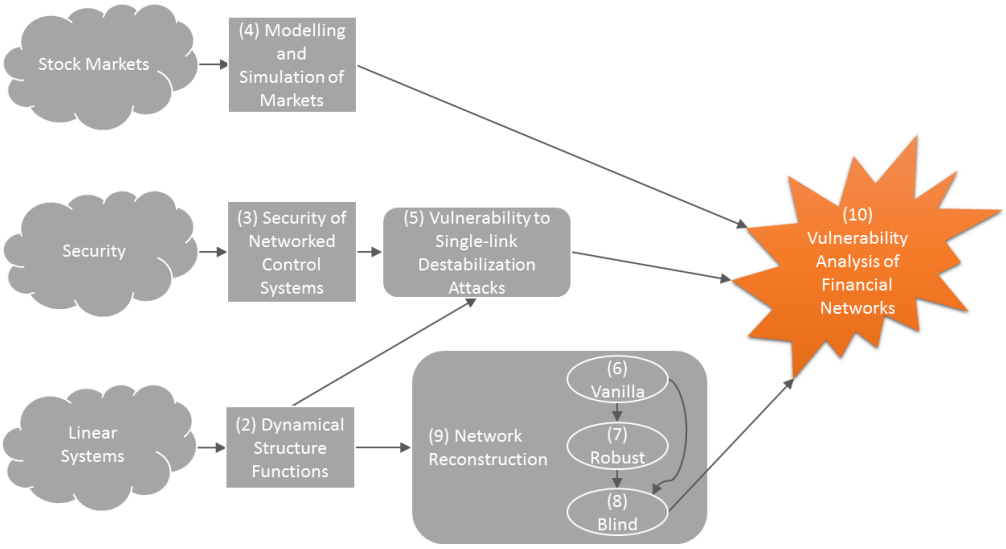
- **Link Elimination:** This work used the term *robust* somewhat differently than [1, 6, 49, 50]. Those works propose and utilize robust techniques for active network reconstruction of frequency-domain networks, which is done by forcing certain links to be zero and then performing network reconstruction, and then repeating with alternative choices of links being forced to zero. The final reconstructed network is generated by the choice of links being forced to zero that results in a network that minimizes the Akaike Information

Criterion. Similar techniques in passive reconstruction for all four algorithms may also be considered.

- **Partially Passive Algorithms:** Presently, all reconstruction literature assumes that all inputs is entirely controlled, entirely measured but not controlled, or not measured at all. Additional research may consider when a subset of the inputs are controlled and the remaining inputs are measured but not controlled. Questions may be asked about what, if anything, may be gained in terms of the quality of reconstruction and on the necessary assumptions on the system and the data by controlling some subset of the inputs.
- **Partially Blind Algorithms:** Similar to the previous bullet, research may explore the ability to reconstruct by measuring a subset of the inputs and assuming that there are additional unmeasured inputs (such as external noisy disturbances) on the system. This will allow network reconstruction to be performed on more realistic systems, and may result in higher-quality results than simply assuming all inputs are unknown.
- **Bounds on Correctness:** For all algorithms, a precise statement on the bounds of the correctness of reconstructed networks under noise or when inputs are not measured would be valuable.
- **Quality of Reconstruction of Approximately Linear Networks:** Presently, it is unknown how well network reconstruction performs on data generated by networks that aren't linear but can be approximated by linear networks reasonably well.
- **Reconstruction of Causal Networks:** Presently, network reconstruction requires that the underlying network be strictly causal. Future techniques may relax this assumption to reconstruct causal, but not necessarily strictly causal networks.
- **Reconstruction of Unstable Networks:** Presently, network reconstruction requires that the underlying network be stable. Future work may explore whether it is even possible to reconstruct unstable networks, and if so, present techniques to do so.

Chapter 10

Vulnerability Analysis of Financial Networks



In this chapter, we turn our attention to the original problem at hand, which is to analyze the vulnerability of financial networks to destabilizing attacks.

10.1 Network Reconstruction as Applied to Financial Networks

Consider again Figure 10.1, repeated from Chapter 4. The market consists of feedback interactions between traders and a Matching Engine. Recall that traders take current prices and external market information to make decisions about which securities to buy or sell, submitting these decisions as orders to the Matching Engine. The Matching Engine evaluates all orders and decides on fair trades, computing stock prices and the limit order book as outputs.

Since stock prices are publicly reported, we can treat any subset of those prices as the output to the system. And though external information is also typically public, it is complex and hard to measure; as such, we treat external information as an unmeasured input; as such, any reconstruction we perform on financial data will necessarily utilize one of the Blind Reconstruction Algorithms (in truth, these algorithms were designed precisely so that we can perform network reconstruction of financial data).

And since we are measuring only stock prices and not any inputs into the market, we are required to use either the B-VPNR or the B-RPNR Algorithm to reconstruct financial networks.

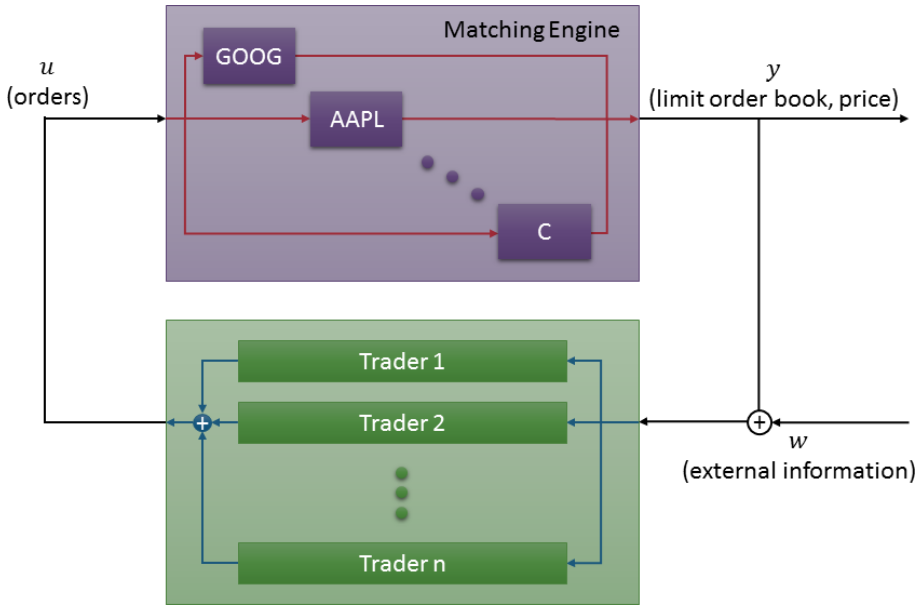


Figure 10.1 Overview of the stock market. Prices within the market engine of one security are made independent of the orders submitted to other securities. Inter-price dynamic relationships only exist because of trader behavior. A repeat of Figure 4.2 from Chapter 4.

In this section, we discuss how reconstructed financial networks and their corresponding vulnerability analyses should be interpreted.

10.1.1 Interpretation of A Reconstructed Financial Network

The system being reconstructed from financial data is the system that maps external information w to prices y, which is the full market including both trader behavior and the Matching Engine. However, recall that the prices computed for a single security by Matching Engine are independent from the orders submitted for any other security. As such, prices of one security are only dependent on prices of another security through trader behavior. Since the $Q(z)$ reconstructed provides information on the causal dependence of the price of each security on the prices of all other securities, $Q(z)$ really is a representation of trader behavior.

10.1.2 Interpretation of the Impulse Responses

Since a link (i, j) in the reconstructed $Q(z)$ define the causal impact of price j on price i, we can interpret the reconstructed impulse responses $Q(t)$ as how the prices of i will change in the future given an increase of \$1.00 in the price of i now and holding everything else in the network constant.

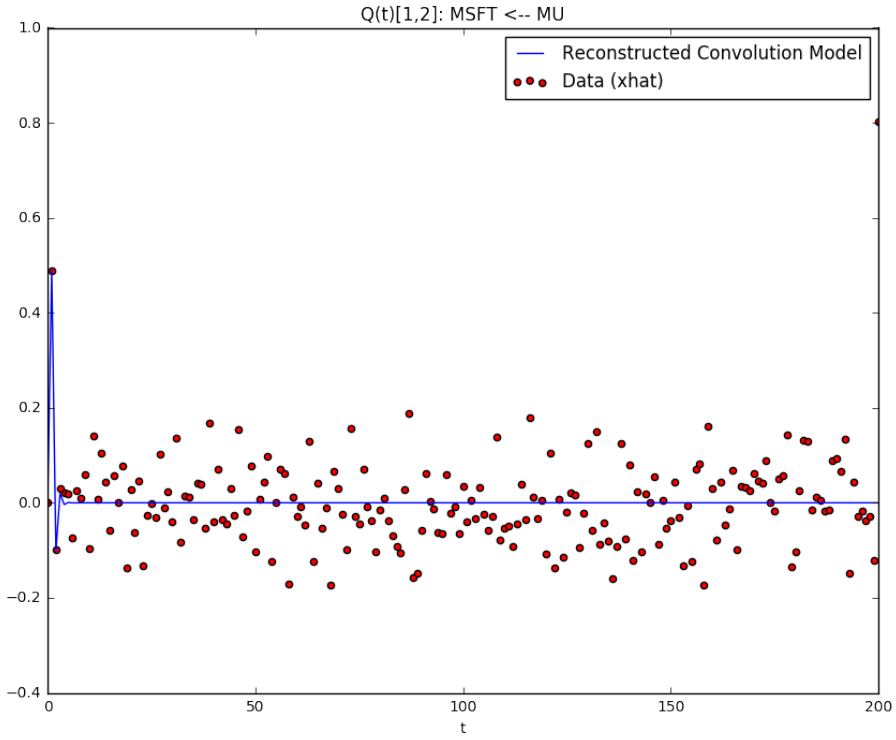


Figure 10.2 An example impulse response of financial data describing the causal impact of the prices of Micron (MU) on the prices of Microsoft (MSFT) at a daily timescale resolution. This and actual reconstructed impulse response copied from the top-left image of Figure 10.3 in Section 10.3.1.

For example, consider link $Q_{12}(t)$ shown in Figure 10.2, which shows the causal impact of the prices of Micron (MU) stock on the prices of Microsoft (MSFT) stock from day to day. Suppose that MSFT is presently at \$30.00, and suppose that right now, the price of MU increases from \$10.00 to \$11.00, only to immediately drop back down to \$10.00 tomorrow.

Then Figure 10.2 shows what the price of MSFT will be over the next 200 days (offset by \$30.00) if no other prices in the market changed. In particular, tomorrow, MSFT would be at approximately \$30.50. In two days, MSFT would be at approximately \$29.90. In three days, MSFT would be barely above \$30.00. And beyond that, the change in MU prices today have negligible impact on future prices of MSFT.

One thing to note, as shown in the previous section, the B-VPNR and the B-RPNR Algorithms are incapable of exactly computing the impulse response of the actual network.

The reconstructed impulse responses will only be rough approximations of the actual impulse responses.

10.1.3 Interpretation of Link Magnitudes

Link magnitudes are a representation of the strength of the causal relationship between the prices of two securities, relative to the other links in the network. Though impulse responses reconstructed using the B-VPNR and the B-RPNR algorithms will merely be rough approximations of the actual impulse responses, the relative reconstructed link magnitudes will be fairly close to the actual network link magnitudes.

10.1.4 Creating a Destabilizing Attack on Financial Networks

For any link in a reconstructed financial network, a non-zero vulnerability on that link indicates that a stable additive perturbation can be created to destabilize the full network. Like link magnitudes, relative vulnerabilities reconstructed using the B-VPNR and the B-RPNR algorithms tend to be fairly accurate.

An example of a stable additive perturbation on a link $A \to C$ is an algorithmic trader that observes the prices of C and for every \$1 in price that C increases, the trader buys $\$10,000$ of A.

Vulnerabilities are inversely proportional to the minimum amount of money required to be invested on each link in order to destabilize the system. As such, the most vulnerable link will be the one that is cheapest to attack. And while this analysis can indicate which link to attack and roughly the amount of money required to successfully attack the link, it does not say that all attacks of that magnitude can destabilize the system. Furthermore, it does not indicate how to attack the link to destabilize the system, only that it can be done.

If we could reconstruct the dynamics of a network exactly, it would be fairly straightforward to create both a destabilizing attack as well as a minimal (the cheapest) destabilizing attack. However, as noted previously, the impulse responses (and hence the network dy-

namics) reconstructed using the B-VPNR and the B-RPNR Algorithms are merely rough approximations of the actual network; as such, constructing a destabilizing attack is more difficult and will be relegated to future research.

10.1.5 Protecting Against a Destabilizing Attack on Financial Networks

As discussed in the previous section, it is presently fairly difficult to construct a destabilizing attack on financial networks given that the impulse responses–and hence system dynamics–reconstructed using the B-VPNR and the B-RPNR algorithms are only rough approximations of the true network. However, future research may demonstrate methods on how to create destabilizing attacks even on these rough approximations; as such, we may wish to consider methods to protecting against such attacks.

Legal considerations and regulations aside, the most effective way to protect financial networks against destabilization attacks would be to add noise to reported prices. As shown in the previous chapter, the blind algorithms are both sensitive to even small levels of noise; as such, noise added would make it extremely difficult to construct a destabilizing attack. It would be difficult even to tell where in the network the cheapest attack could be launched as even vulnerability was shown to be sensitive to this noise.

That said, adding noise to reported noise would only make it difficult to construct a destabilizing attack. However, once found (either intentionally or inadvertently), the same attack will destabilize the system regardless of whether noise has been added. Future research is needed to determine how to make financial networks both robust and resilient to destabilizing attacks.

10.2 Reconstructability of Financial Networks

In this section, we evaluate the assumptions discussed in Section 8.3 which are required for both algorithms to function properly to determine whether financial networks can even be reconstructed. These assumptions are as follows:

- **Linearity:** Network reconstruction as presented in this work assumes that the dynamics of the underlying are linear. However, it is almost guaranteed that the actual dynamics of the market are non-linear. Therefore, in performing a network reconstruction on price data, we are implicitly assuming that the dynamics are near enough to linear to be approximated by a linear network. This assumption is strong, and the most debatable of all assumptions made in this section, but we make it anyway.

- **Stability and Strict Causality:** In order to perform network reconstruction on market data, we must assume (for now at least) that links in $Q(z)$ and $P(z)$ are stable, or in other words, that the impulse responses $Q(t)$ and $P(t)$ converge to zero as $t \to \infty$. We also assume that dynamics are strictly causal on all time scales, meaning that present information only affects prices in the future. Strict causality implies that $Q(0) = 0$ and $P(0) = 0$.

These assumptions can be interpreted in context of the Efficient Market Hypothesis (EMH). As discussed in [26], the EMH is the idea that stock prices now ($t = 0$) are a reflection all available information in the market, including historical prices $y(t)$ for $t < 0$ and external market information $u(t)$ for $t \leq 0$. In other words, the impulse responses $Q(t)$ and $P(t)$ are precisely 0 for $t > 0$, though they need not be 0 for $t = 0$.

As stability is the assumption that $Q(t)$ and $P(t)$ converge to zero, the EMH is much a stronger statement forcing $Q(t)$ and $P(t)$ to converge to zero arbitrarily fast. However, the EMH is also a contradiction of causality in that in allows–and even requires–that all present information be reflected in prices immediately.

There are criticisms of the EMH, most of which reduce to the idea that prices have memory and momentum (meaning that $Q(t)$ and $P(t)$ need not converge to zero arbitrarily fast, but remain stable). It is also not a stretch to assume that information needs time to propagate through the network (meaning that $Q(t)$ and $P(t)$ are strictly causal).

For networks to be reconstructed, we require (for now at least) that the networks be strictly causal. We also require the violation of the EMH, allowing prices to have at least some memory, while remaining stable. The very fact that we are able to reconstruct networks (as shown later in this chapter) with features consistent on both the daily and the decisecond resolution levels is evidence that the EMH isn't entirely true. Otherwise, if the EMH holds, we would reconstruct $Q(z)$ to be essentially zero.

- **Informativity Conditions:** Though it is not yet known what the necessary and sufficient informativity conditions of the blind reconstruction are, we assume that they are related to the independence of the noise ζ. This assumption implies that external market information can drive the present set of prices to any other set of prices arbitrarily quickly. While this assumption is strong, it is also not unreasonable, especially given the efficient market hypothesis and the sensitivity of prices to breaking news. Nonetheless, it is presently difficult–if not impossible–to test the validity of this assumption.

- **Small Levels of Noise on Outputs**: In the previous chapter, we showed that both the B-VPNR and the B-RPNR reconstruction algorithms are highly sensitive to additive noise on the outputs; as such, we require that such noise be small. Fortunately, stock prices are reported exactly, meaning we have no noise on the outputs, additive or otherwise.

  Note also that this implies that the B-VPNR Algorithm is the proper algorithm to use to reconstruct financial networks. However, for completeness, we also reconstruct financial networks using the B-RPNR Algorithm.

- **Richness of Data:** In other words, $\hat{L}_Q$ is injective with $\hat{y} \in \mathcal{R}(\hat{L}_Q)$. In all networks reconstructed in this chapter, this condition was met.

- **Large Enough r:** The blind algorithms require that r be large enough to capture the full impulse response dynamics. For all experiments in this chapter, we choose $r = 200$.

Every impulse response had more than sufficient time to converge to an envelope around zero within this time frame (see Figure 10.2 and all subsequent impulse responses).

- **Large Enough T:** We also require a large enough T that we don't overfit the least squares model. In all examples, we chose $T = 6r + 1 = 1201$, a rule of thumb which worked well for the examples in the previous chapter.

In short, financial data meets most of the assumptions required to use the B-VPNR or the B-RPNR Algorithm. For the few assumptions which are either violated or unknown, we have some evidence that network reconstruction is possible nonetheless.

10.3 Datasets

To perform network reconstruction on financial data, we use samples from the three data sources discussed in Chapter 4. We provide details about each of the three samples here.

10.3.1 Dataset 1 (Daily Data)

The first set of data consists of closing prices taken from Yahoo! Finance daily data. The set consists of 1201 data points ranging in time from January 6, 2012 to October 13, 2016. These prices are shown in Figure 10.3.

It should be noted that this dataset skips days that are holidays or weekends. However, network reconstruction assumes that the time between each time point is consistent. As such, holidays and weekends are essentially ignored with the potentially debatable assumption that prices change the same way over the larger time change spreading over a holiday or a weekend as they would over a single day. It is possible that this is the reason why the impulse responses on this dataset (Section 10.4) only converge to an envelope around zero and never actually arrive at zero.

This dataset is designed to capture the behavior of slow traders such as day traders.

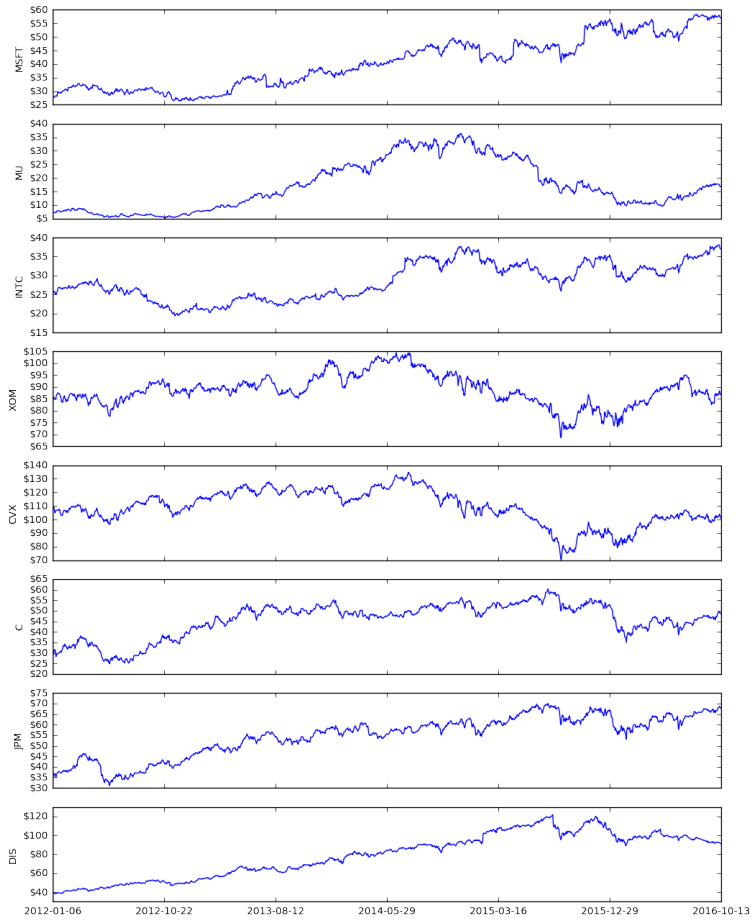


Figure 10.3 The prices in Dataset 1 taken daily from 1/6/12 to 10/13/12.

10.3.2 Dataset 2 (Minute-Resolution Data)

The second set of data consists of the minute-by-minute samples of last prices recorded by the Tour de Finance. Like Dataset 1, this set consists of 1201 data points ranging from November 14, 2016 at 9:30 AM EST to November 16, 2016 at 2:57 PM EST. These prices are shown in Figure 10.4.

There are no holidays in this time period. However, data was only collected between 9:30 AM and 4:00 PM EST. Hence–as with Dataset 1–there are large gaps in the data that are treated the same as a single time step. Furthermore, the effect of these gaps will be more pronounced than gaps within Dataset 1 as the number of minutes from market close to market open are orders of magnitude larger than the number of days in a weekend or holiday.

This dataset is designed to capture the behavior of slow algorithmic traders.

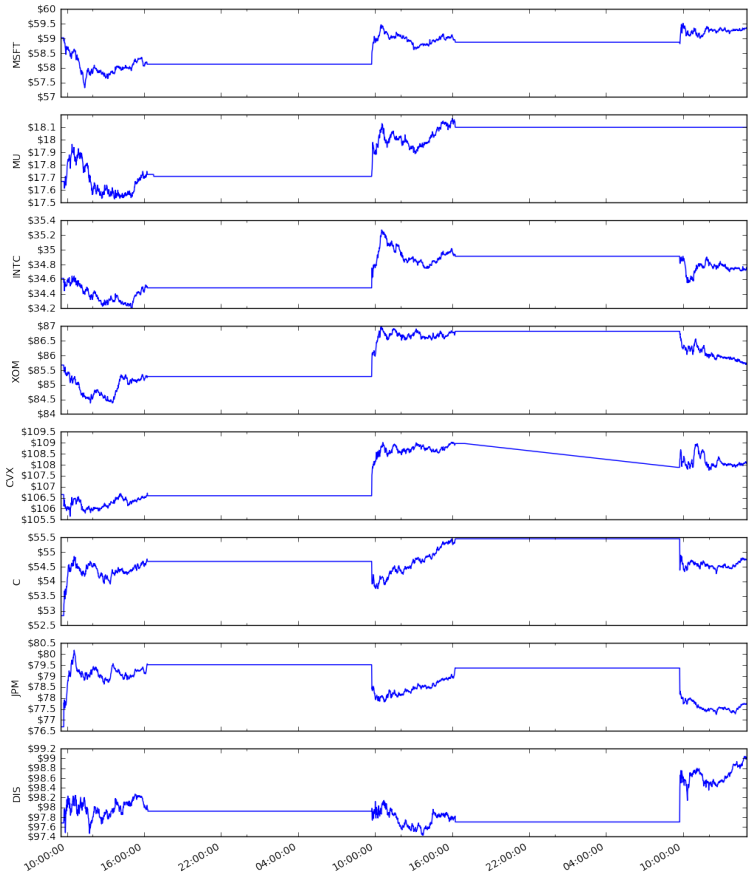


Figure 10.4 The prices in Dataset 2 taken every minute from 11/14/16 to 11/16/2016.

10.3.3 Dataset 3 (Decisecond-Resolution Data)

The final set of data is gathered from the ITCH data. Samples are taken every tenth of a second, and the last trade price is forward-filled to represent the price of each security at each decisecond. This set, like the prior two datasets, consists of 1201 data points, this time ranging from 11:00 AM to 11:20 AM EST on June 30, 2014. Unlike the previous two datasets, there are no temporal gaps in the data. These prices are shown in Figure 10.5.

This dataset is designed to capture the behavior of high frequency traders.

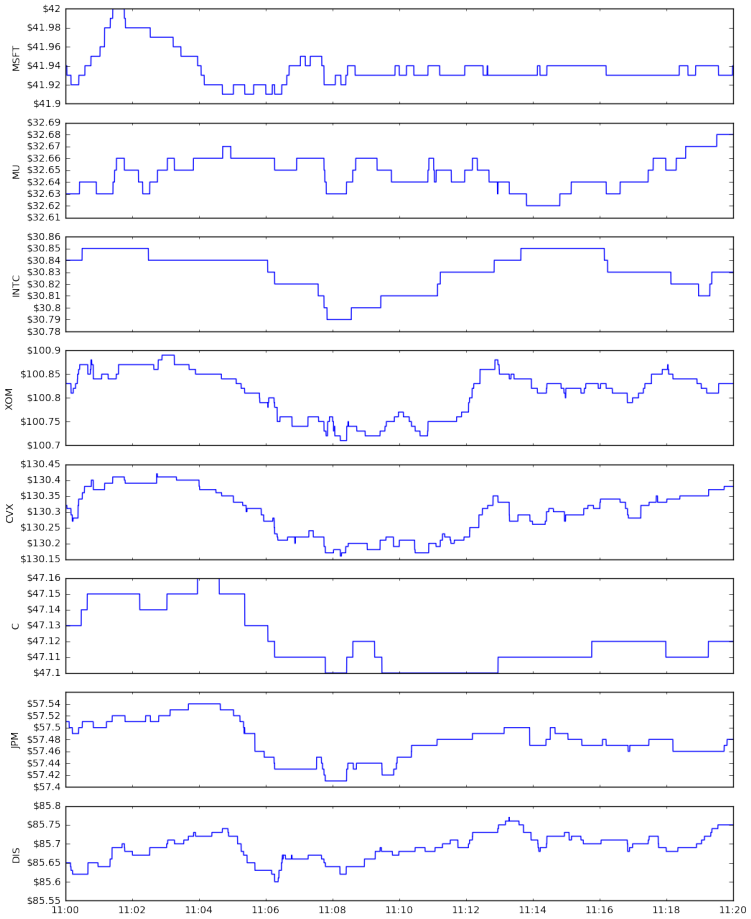


Figure 10.5 The prices in Dataset 3 taken every tenth of a second on 6/30/14.

10.4 Reconstructed Networks

We now show the results of two networks reconstructed from the three datasets described previously.

10.4.1 Network 1

The first network we reconstruct consists of three securities all taken from the technology industry: Microsoft Corporation (MSFT), Micron Technology, Inc. (MU), and Intel (INTC). The results are outlined below.

The surprising result is that the network reconstructed using the B-VPNR Algorithm on daily data (Figure 10.8) is nearly identical to the network reconstructed using decisecond-

resolution data (Figure 10.14), even though the impulse responses at each resolution are very different. This is not only evidence that the network reconstruction is working, but also it is capturing something fundamental about trader behavior that is independent of time scale (assuming that the network reconstructed from Dataset 2 is different primarily because of the large gaps in data that are not present in either Datasets 1 or 3).

In the two networks that were nearly identical, the strongest causal link is the link from Intel to Micron. This isn't entirely surprising as the two companies often collaborate, including through their joint venture IM Flash. What is surprising is that the link from Micron back to Intel is weak, though perhaps this can be explained by the fact that Micron is relatively small compared to Intel. The next strongest links are the links from Intel to MSFT and back. This isn't surprising since Intel products power many Microsoft products, leaving the two companies co-dependent.

Notice also that the B-VPNR and the B-RPNR Algorithms reconstruct nearly exactly the same networks on Datasets 1 and 3 (Figures 10.8 and 10.14), but the two algorithms perform very differently on Dataset 3 (Figure 10.11).

Dataset 1

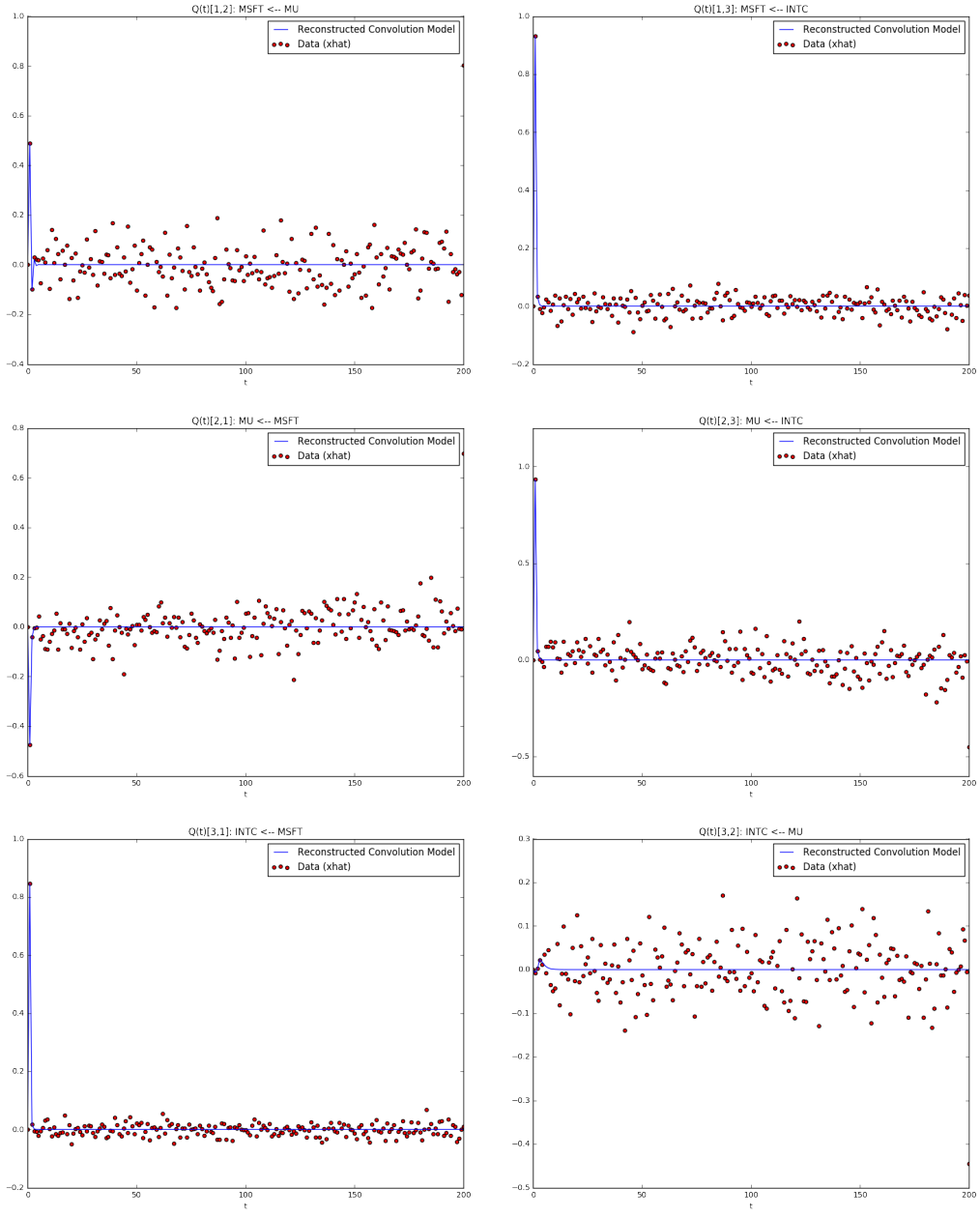


Figure 10.6 The impulse responses of links in Network 1 (MSFT, MU, and INTC) using Dataset 1 (daily data) reconstructed using the B-VPNR Algorithm.

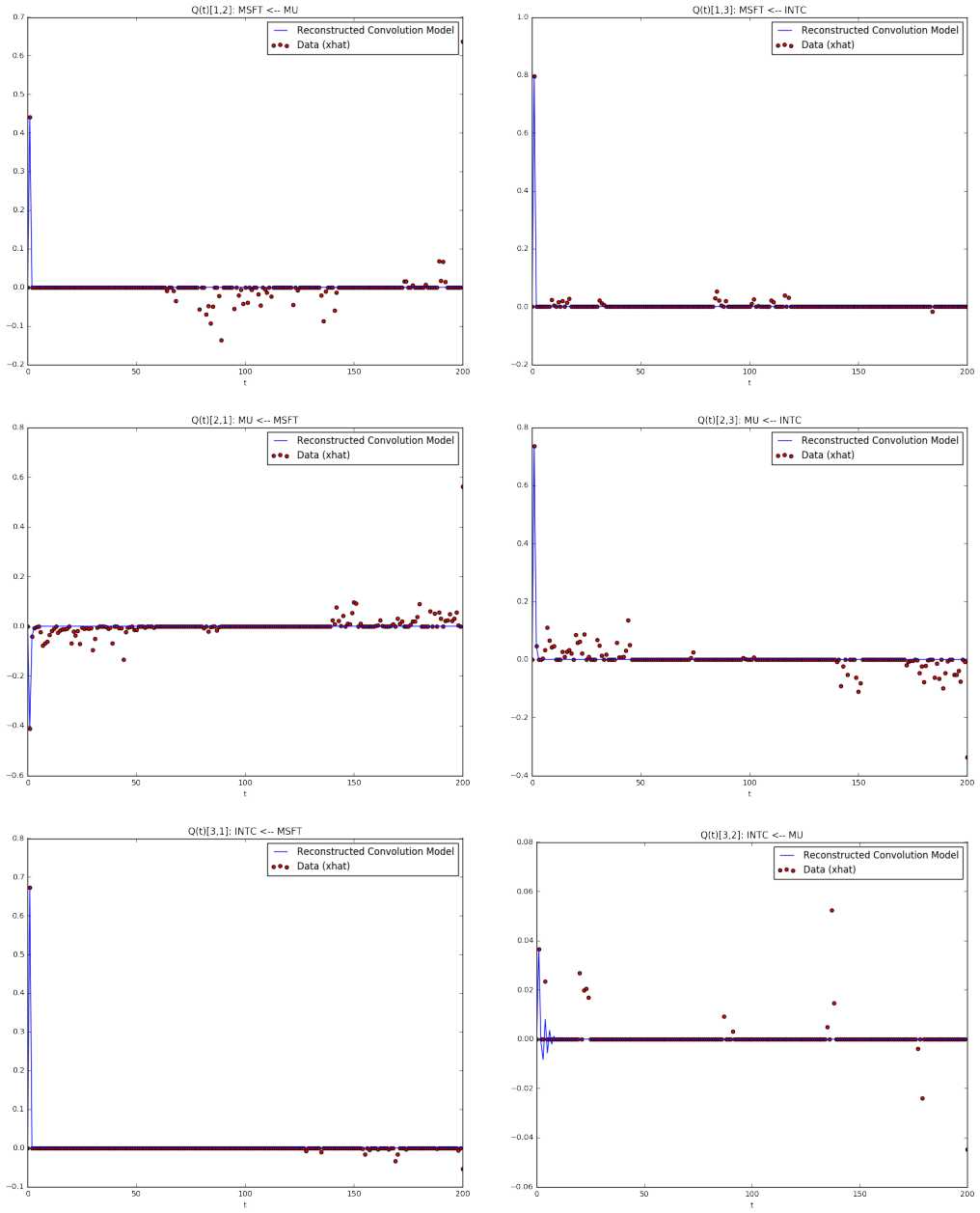


Figure 10.7 The impulse responses of links in Network 1 (MSFT, MU, and INTC) using Dataset 1 (daily data) reconstructed using the B-RPNR Algorithm.

Link	Magnitude	Normalized Magnitude	Vulnerability	Normalized Vulnerability
MSFT ← MU	0.614	0.622	56.225	0.246
MSFT ← INTC	0.970	0.981	139.070	0.609
MU ← MSFT	0.519	0.525	77.421	0.339
MU ← INTC	0.988	1.000	68.704	0.301
INTC ← MSFT	0.866	0.876	228.344	1.000
INTC ← MU	0.062	0.063	80.950	0.355

Table 10.1 The magnitude and vulnerability of links in Network 1 (MSFT, MU, and INTC) using Dataset 1 (daily data) reconstructed using the B-VPNR Algorithm.

Link	Magnitude	Normalized Magnitude	Vulnerability	Normalized Vulnerability
MSFT ← MU	0.444	0.557	1.047	0.380
MSFT ← INTC	0.797	1.000	1.583	0.575
MU ← MSFT	0.458	0.574	1.126	0.409
MU ← INTC	0.788	0.988	0.794	0.288
INTC ← MSFT	0.674	0.846	2.753	1.000
INTC ← MU	0.050	0.063	1.234	0.448

Table 10.2 The magnitude and vulnerability of links in Network 1 (MSFT, MU, and INTC) using Dataset 1 (daily data) reconstructed using the B-RPNR Algorithm.

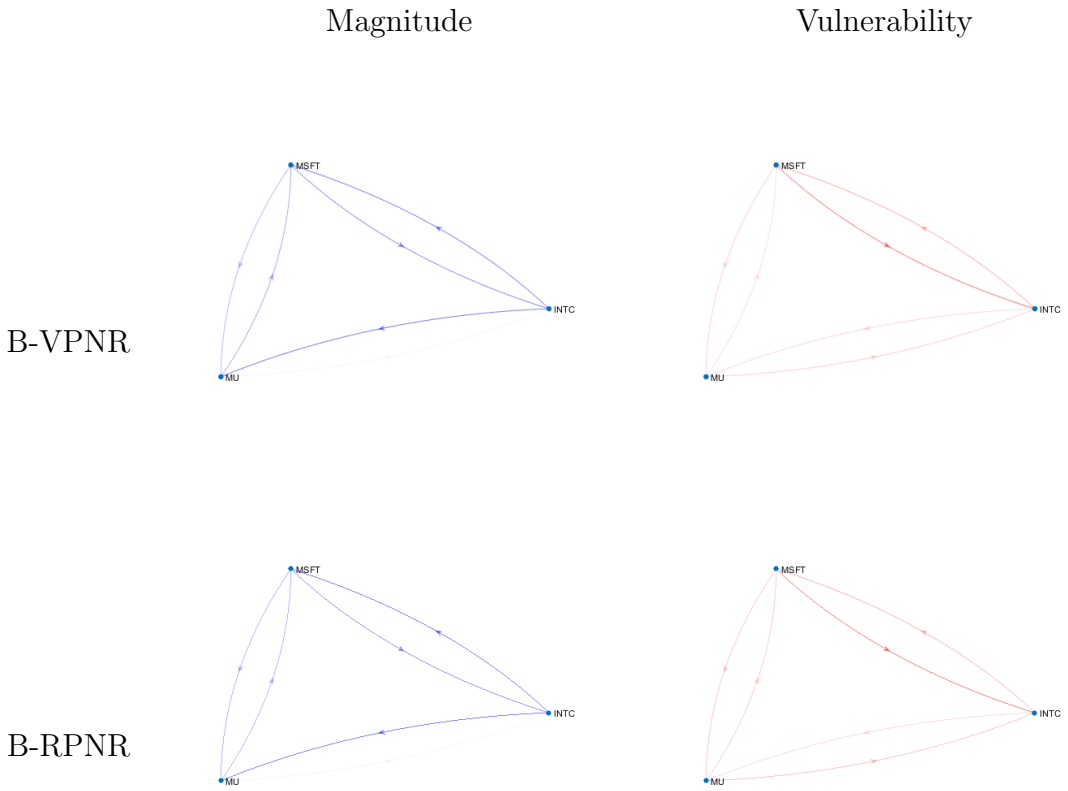


Figure 10.8 The normalized magnitude (top left) and vulnerability (top right) of links in Network 1 on Dataset 1 reconstructed using the B-VPNR Algorithm as given in Table 10.1 and the same (bottom left and right) using the B-RPNR Algorithm as given in Table 10.2.

Dataset 2

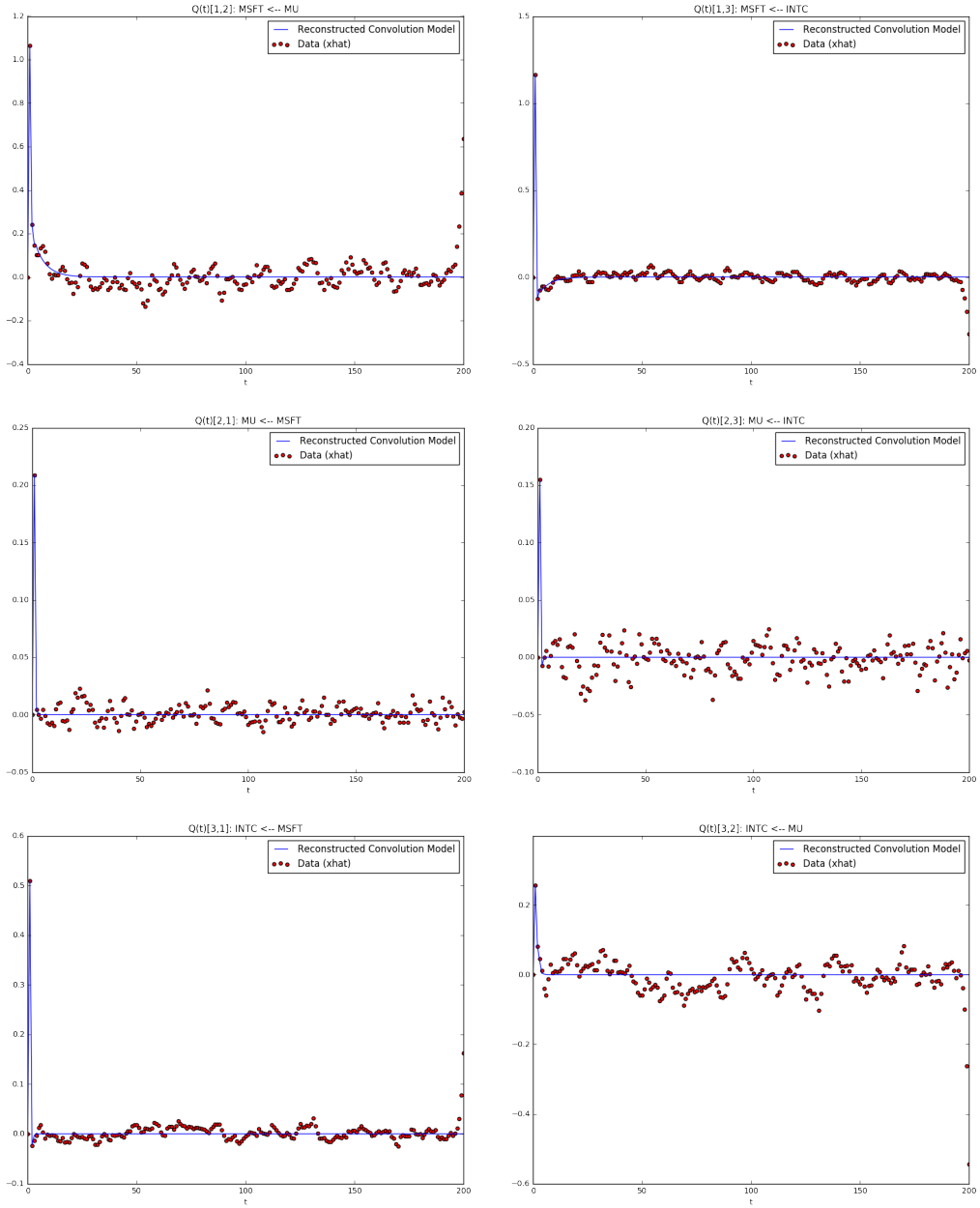


Figure 10.9 The impulse responses of links in Network 1 (MSFT, MU, and INTC) using Dataset 2 (minute-resolution data) reconstructed using the B-VPNR Algorithm.

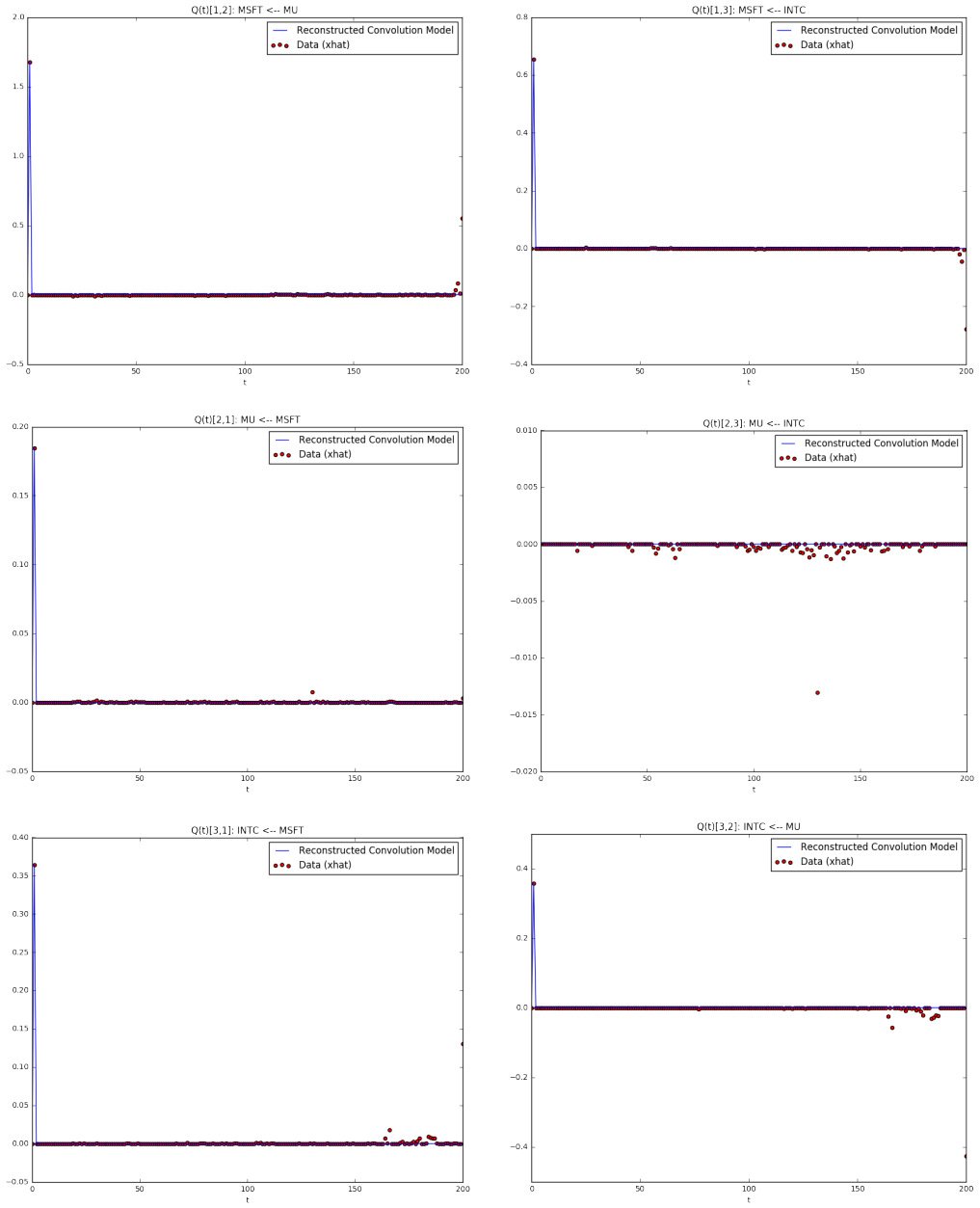


Figure 10.10 The impulse responses of links in Network 1 (MSFT, MU, and INTC) using Dataset 2 (minute-resolution data) reconstructed using the B-RPNR Algorithm.

Link	Magnitude	Normalized Magnitude	Vulnerability	Normalized Vulnerability
MSFT ← MU	2.251	1.000	21.909	0.116
MSFT ← INTC	1.253	0.557	43.717	0.230
MU ← MSFT	0.213	0.095	189.684	1.000
MU ← INTC	0.161	0.072	113.878	0.600
INTC ← MSFT	0.526	0.234	68.183	0.359
INTC ← MU	0.387	0.172	20.432	0.108

Table 10.3 The magnitude and vulnerability of links in Network 1 (MSFT, MU, and INTC) using Dataset 2 (minute-resolution data) reconstructed using the B-VPNR Algorithm.

Link	Magnitude	Normalized Magnitude	Vulnerability	Normalized Vulnerability
MSFT ← MU	1.689	1.000	0.454	0.096
MSFT ← INTC	0.657	0.389	1.056	0.223
MU ← MSFT	0.185	0.109	4.730	1.000
MU ← INTC	0.000	0.000	2.385	0.504
INTC ← MSFT	0.367	0.217	1.618	0.342
INTC ← MU	0.358	0.212	0.299	0.063

Table 10.4 The magnitude and vulnerability of links in Network 1 (MSFT, MU, and INTC) using Dataset 2 (minute-resolution data) reconstructed using the B-RPNR Algorithm.

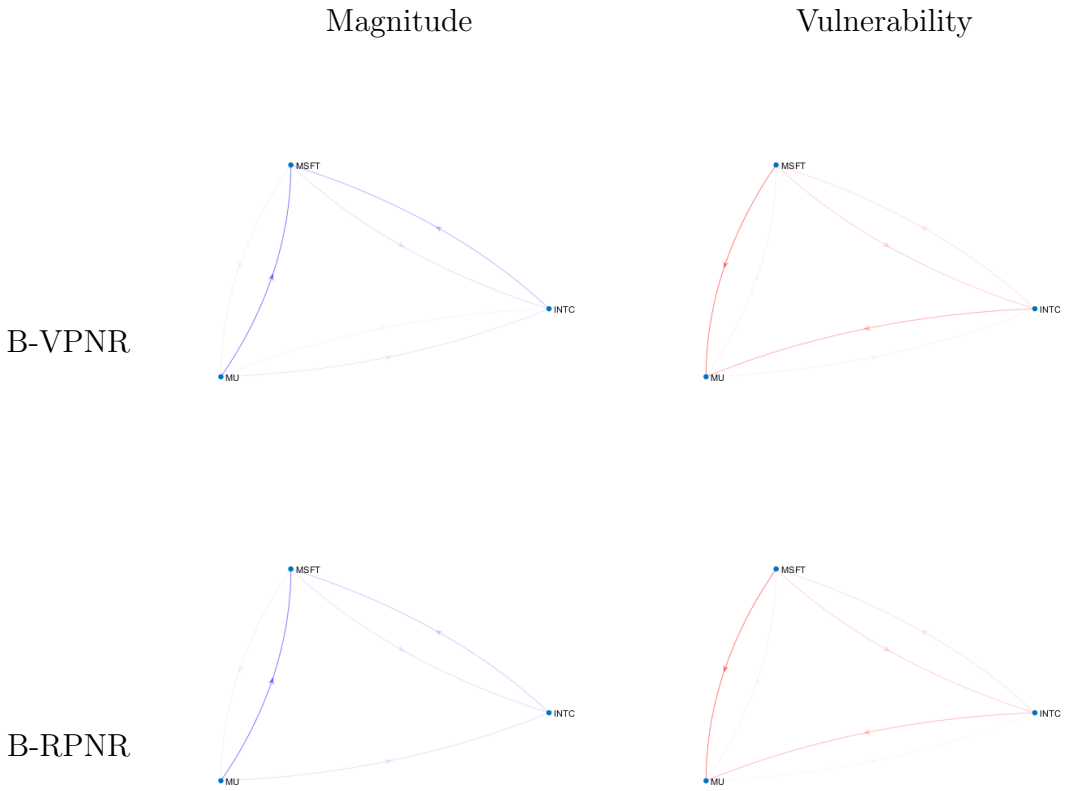


Figure 10.11 The normalized magnitude (top left) and vulnerability (top right) of links in Network 1 on Dataset 2 reconstructed using the B-VPNR Algorithm as given in Table 10.3 and the same (bottom left and right) using the B-RPNR Algorithm as given in Table 10.4.

Dataset 3

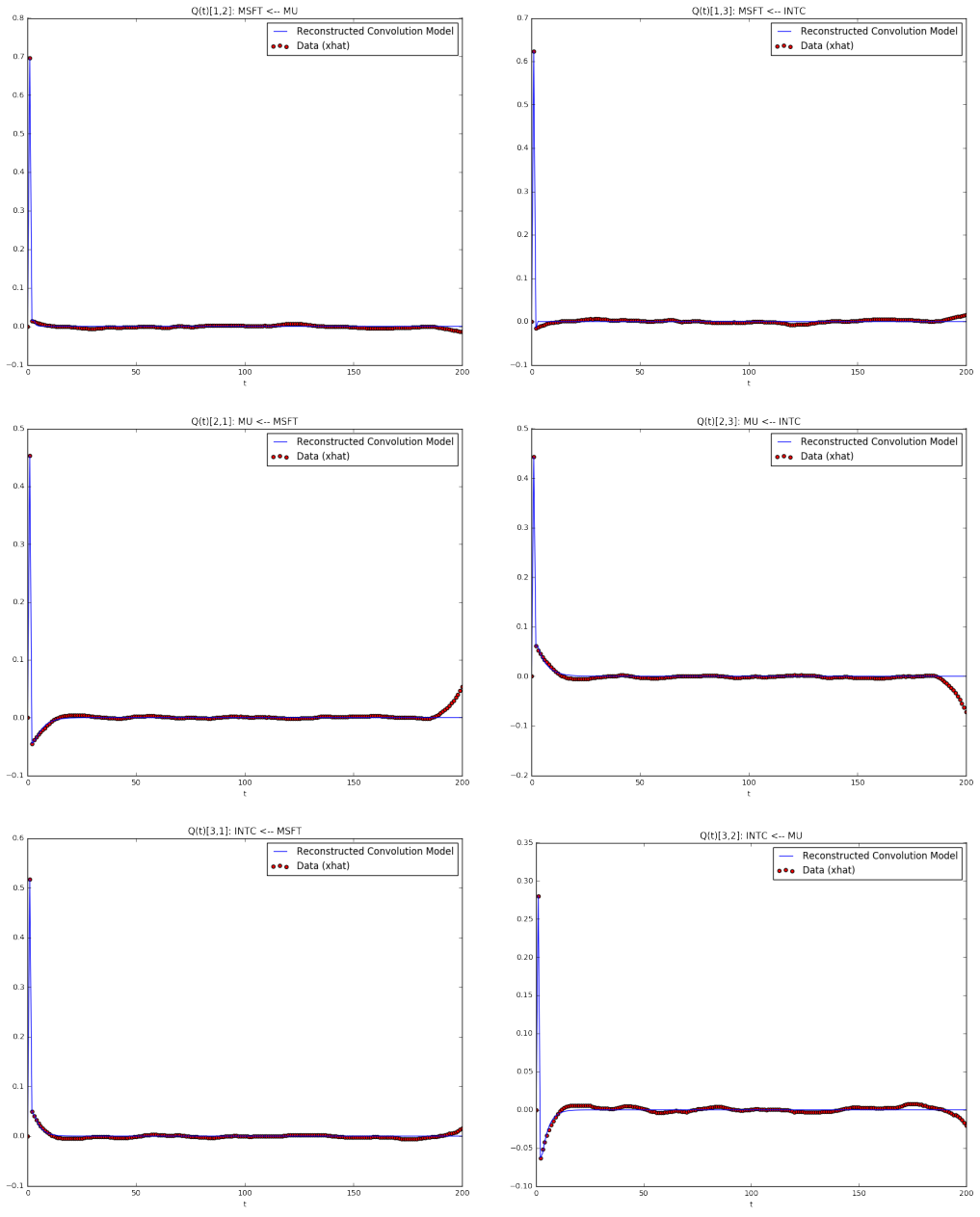


Figure 10.12 The impulse responses of links in Network 1 (MSFT, MU, and INTC) using Dataset 3 (decisecond-resolution data) reconstructed using the B-VPNR Algorithm.

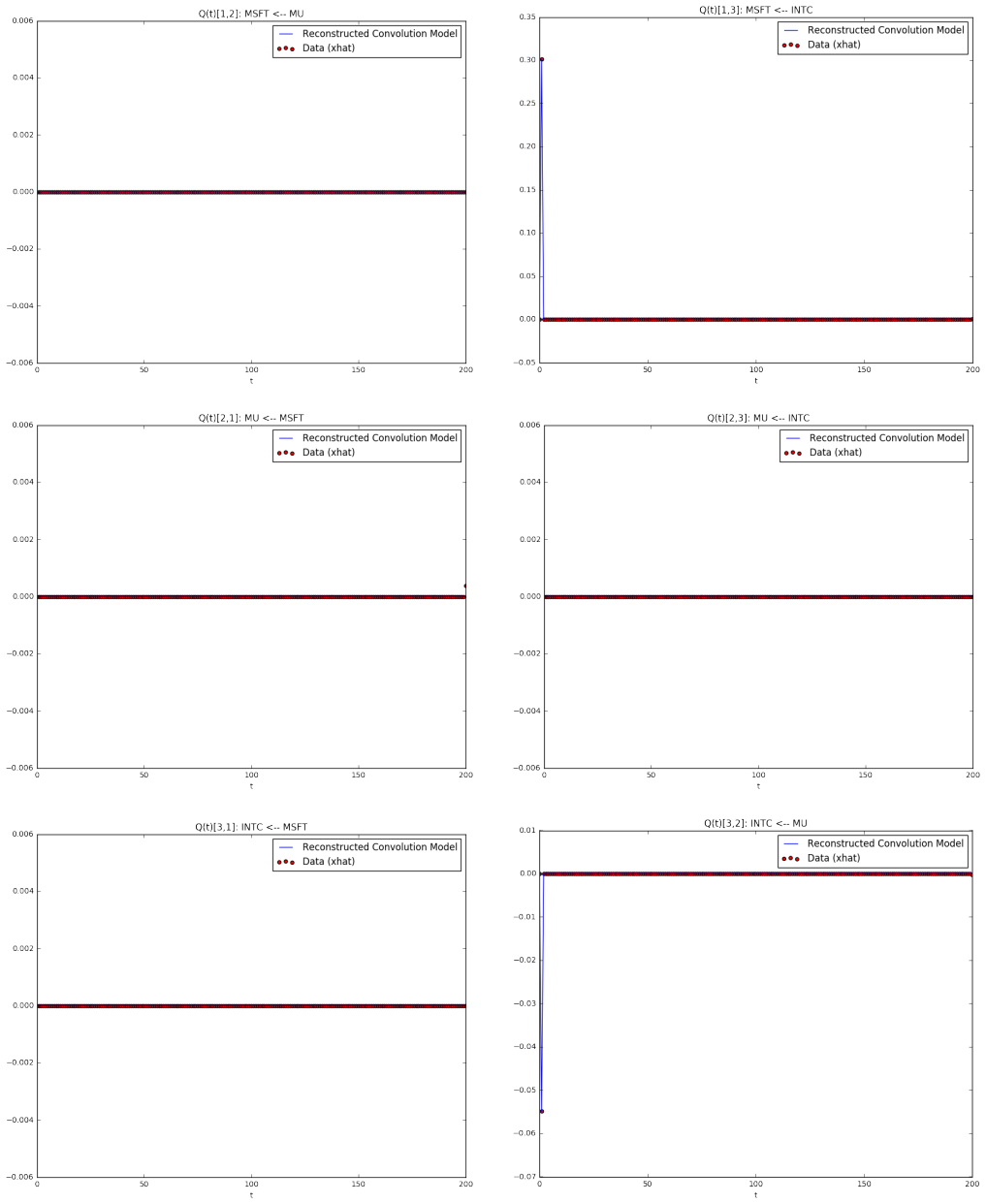


Figure 10.13 The impulse responses of links in Network 1 (MSFT, MU, and INTC) using Dataset 3 (decisecond-resolution data) reconstructed using the B-RPNR Algorithm.

Link	Magnitude	Normalized Magnitude	Vulnerability	Normalized Vulnerability
MSFT ← MU	0.731	0.929	41.464	0.653
MSFT ← INTC	0.639	0.812	39.136	0.616
MU ← MSFT	0.477	0.606	39.463	0.622
MU ← INTC	0.787	1.000	29.036	0.457
INTC ← MSFT	0.729	0.926	63.486	1.000
INTC ← MU	0.318	0.404	48.714	0.767

Table 10.5 The magnitude and vulnerability of links in Network 1 (MSFT, MU, and INTC) using Dataset 3 (decisecond-resolution data) reconstructed using the B-VPNR Algorithm.

Link	Magnitude	Normalized Magnitude	Vulnerability	Normalized Vulnerability
MSFT ← MU	0.000	0.000	0.000	0.000
MSFT ← INTC	0.301	1.000	0.000	0.000
MU ← MSFT	0.000	0.000	0.017	0.055
MU ← INTC	0.000	0.000	0.055	0.182
INTC ← MSFT	0.000	0.000	0.301	1.000
INTC ← MU	0.055	0.182	0.000	0.000

Table 10.6 The magnitude and vulnerability of links in Network 1 (MSFT, MU, and INTC) using Dataset 3 (decisecond-resolution data) reconstructed using the B-RPNR Algorithm.

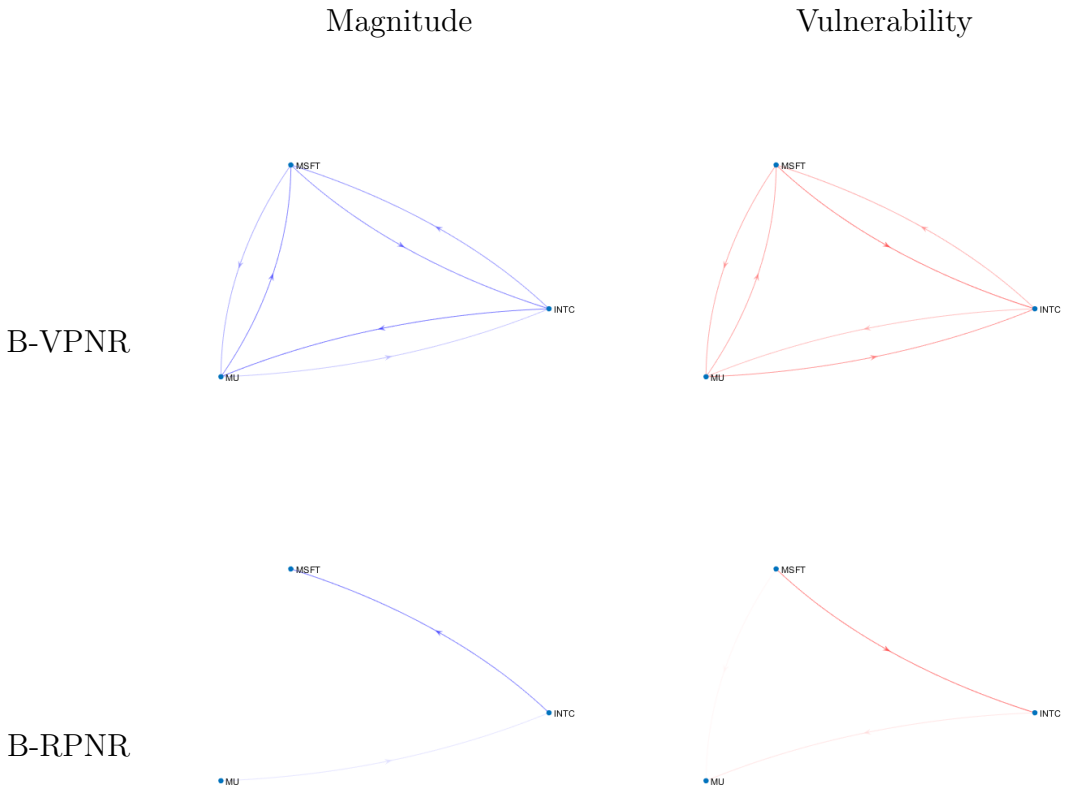


Figure 10.14 The normalized magnitude (top left) and vulnerability (top right) of links in Network 1 on Dataset 3 reconstructed using the B-VPNR Algorithm as given in Table 10.5 and the same (bottom left and right) using the B-RPNR Algorithm as given in Table 10.6.

10.4.2 Network 2

For Network 2, we consider a somewhat larger set of securities, though still small enough that we can perform network reconstruction. This network consists of five securities in three industries, which are:

- **Energy:** Exxon Mobil Corporation (XOM) and Chevron Corporation (CVX)
- **Finance:** Citigroup Inc (C) and JPMorgan Chase & Co (JPM)
- **Entertainment:** Walt Disney Co (DIS)

This time, the networks reconstructed by the B-VPNR reconstruction algorithm are very different across datasets. However, the networks reconstructed using the B-RPNR

Algorithm are very similar across all three sets[1], therefore, once again, we are capturing something fundamental about the market.

In all three networks reconstructed using the B-RPNR algorithm, there is a strong relationship between XOM and CVX, which is sensible as they are both in the same industry. On Datasets 2 and 3, there is also a strong relationship between DIS and C, whereas in 1, it is between JPM and C. In Datasets 1 and 3, there is also a relatively-strong relationship between JPM and C.

[1] More investigation is required in order to determine why the B-VPNR Algorithm provides more consistent results for the smaller network and the B-RPNR Algorithm for the larger network. As noted previously, the larger the network, the stronger the unmeasured ζ is compared to the links in $Q(t)$. As such, it is possible that the larger the network grows, the more ζ acts as noise which needs to be reduced using the B-RPNR Algorithm.

Dataset 1

Link	Magnitude	Normalized Magnitude	Vulnerability	Normalized Vulnerability
XOM ← CVX	1.109	0.175	3.513	0.030
XOM ← C	1.541	0.243	2.526	0.022
XOM ← JPM	3.046	0.481	7.784	0.066
XOM ← DIS	0.242	0.038	13.957	0.119
CVX ← XOM	1.325	0.209	4.538	0.039
CVX ← C	3.553	0.561	1.045	0.009
CVX ← JPM	5.832	0.920	4.516	0.038
CVX ← DIS	0.472	0.074	11.139	0.095
C ← XOM	6.337	1.000	19.776	0.168
C ← CVX	0.230	0.036	14.372	0.122
C ← JPM	0.675	0.106	23.578	0.201
C ← DIS	0.138	0.022	52.791	0.450
JPM ← XOM	2.895	0.457	3.372	0.029
JPM ← CVX	1.818	0.287	20.640	0.176
JPM ← C	1.804	0.285	7.954	0.068
JPM ← DIS	0.117	0.018	117.373	1.000
DIS ← XOM	0.700	0.110	6.108	0.052
DIS ← CVX	0.536	0.085	1.269	0.011
DIS ← C	2.344	0.370	9.739	0.083
DIS ← JPM	2.218	0.350	2.262	0.019

Table 10.7 The magnitude and vulnerability of links in Network 2 (XOM, CVX, C, JPM, and DIS) using Dataset 1 (daily data) reconstructed using the B-VPNR Algorithm.

Link	Magnitude	Normalized Magnitude	Vulnerability	Normalized Vulnerability
XOM ← CVX	0.595	0.483	2.324	1.000
XOM ← C	0.000	0.000	0.193	0.083
XOM ← JPM	0.000	0.000	0.133	0.057
XOM ← DIS	0.037	0.030	0.658	0.283
CVX ← XOM	0.955	0.775	1.452	0.625
CVX ← C	0.199	0.162	0.133	0.057
CVX ← JPM	0.000	0.000	0.103	0.044
CVX ← DIS	0.001	0.001	0.650	0.280
C ← XOM	0.051	0.042	0.414	0.178
C ← CVX	0.000	0.000	0.653	0.281
C ← JPM	0.431	0.350	0.676	0.291
C ← DIS	0.034	0.027	0.738	0.318
JPM ← XOM	0.000	0.000	0.318	0.137
JPM ← CVX	0.000	0.000	0.432	0.186
JPM ← C	0.490	0.398	0.650	0.280
JPM ← DIS	0.060	0.048	1.659	0.714
DIS ← XOM	0.000	0.000	0.113	0.049
DIS ← CVX	0.236	0.191	0.122	0.053
DIS ← C	0.024	0.019	0.075	0.032
DIS ← JPM	1.232	1.000	0.101	0.044

Table 10.8 The magnitude and vulnerability of links in Network 2 (XOM, CVX, C, JPM, and DIS) using Dataset 1 (daily data) reconstructed using the B-RPNR Algorithm.

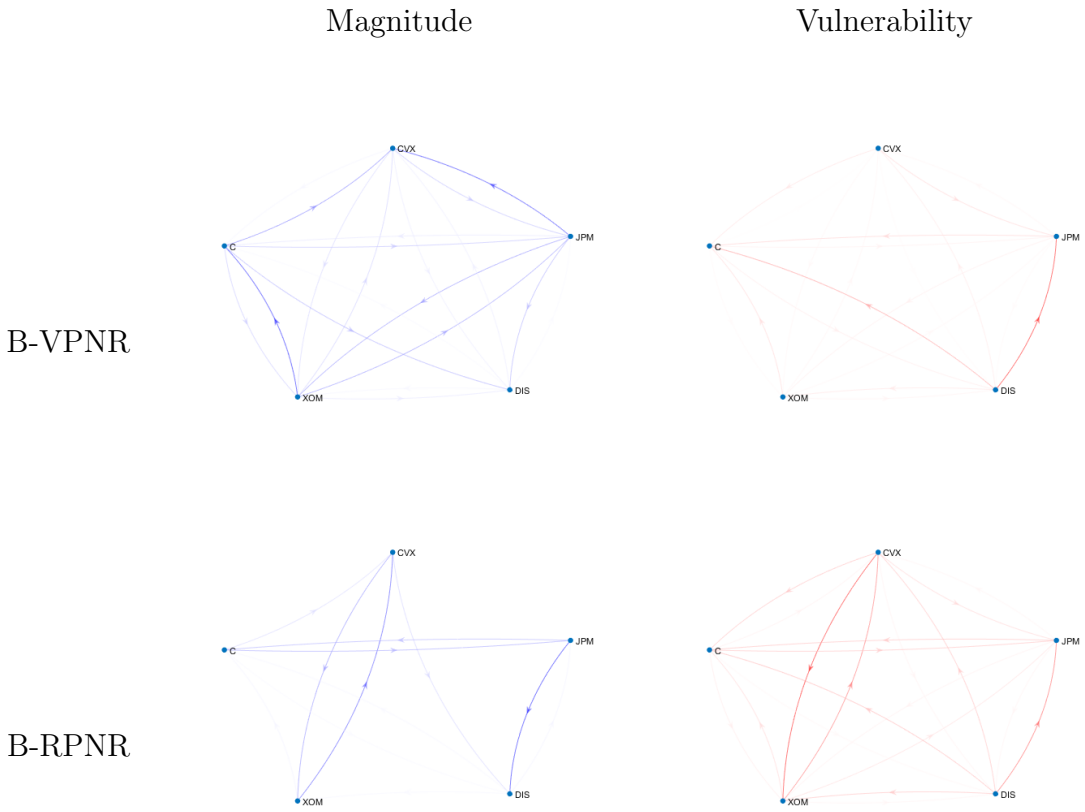


Figure 10.15 The normalized magnitude (top left) and vulnerability (top right) of links in Network 2 on Dataset 1 reconstructed using the B-VPNR Algorithm as given in Table 10.7 and the same (bottom left and right) using the B-RPNR Algorithm as given in Table 10.8.

Dataset 2

Link	Magnitude	Normalized Magnitude	Vulnerability	Normalized Vulnerability
XOM ← CVX	0.568	0.262	3.336	0.111
XOM ← C	1.139	0.526	2.293	0.076
XOM ← JPM	0.362	0.167	4.078	0.135
XOM ← DIS	0.360	0.166	3.839	0.127
CVX ← XOM	1.413	0.653	3.085	0.102
CVX ← C	1.032	0.477	0.581	0.019
CVX ← JPM	0.399	0.185	2.199	0.073
CVX ← DIS	0.136	0.063	1.731	0.057
C ← XOM	0.258	0.119	24.117	0.800
C ← CVX	0.037	0.017	30.149	1.000
C ← JPM	0.580	0.268	9.942	0.330
C ← DIS	0.067	0.031	16.752	0.556
JPM ← XOM	0.209	0.097	9.718	0.322
JPM ← CVX	0.310	0.143	11.820	0.392
JPM ← C	2.143	0.990	3.153	0.105
JPM ← DIS	0.358	0.165	6.230	0.207
DIS ← XOM	1.767	0.816	0.934	0.031
DIS ← CVX	1.447	0.668	1.195	0.040
DIS ← C	2.164	1.000	0.258	0.009
DIS ← JPM	0.208	0.096	0.585	0.019

Table 10.9 The magnitude and vulnerability of links in Network 2 (XOM, CVX, C, JPM, and DIS) using Dataset 2 (minute-resolution data) reconstructed using the B-VPNR Algorithm.

Link	Magnitude	Normalized Magnitude	Vulnerability	Normalized Vulnerability
XOM ← CVX	0.225	0.362	0.740	1.000
XOM ← C	0.000	0.000	0.011	0.015
XOM ← JPM	0.056	0.090	0.072	0.098
XOM ← DIS	0.053	0.085	0.062	0.084
CVX ← XOM	0.621	1.000	0.269	0.363
CVX ← C	0.000	0.000	0.018	0.025
CVX ← JPM	0.227	0.366	0.117	0.157
CVX ← DIS	0.004	0.007	0.100	0.135
C ← XOM	0.000	0.000	0.054	0.073
C ← CVX	0.000	0.000	0.100	0.134
C ← JPM	0.000	0.000	0.307	0.415
C ← DIS	0.186	0.300	0.347	0.469
JPM ← XOM	0.000	0.000	0.126	0.170
JPM ← CVX	0.093	0.150	0.312	0.421
JPM ← C	0.293	0.472	0.005	0.007
JPM ← DIS	0.000	0.000	0.026	0.035
DIS ← XOM	0.000	0.000	0.055	0.074
DIS ← CVX	0.088	0.143	0.035	0.047
DIS ← C	0.322	0.519	0.198	0.268
DIS ← JPM	0.000	0.000	0.057	0.077

Table 10.10 The magnitude and vulnerability of links in Network 2 (XOM, CVX, C, JPM, and DIS) using Dataset 2 (minute-resolution data) reconstructed using the B-RPNR Algorithm.

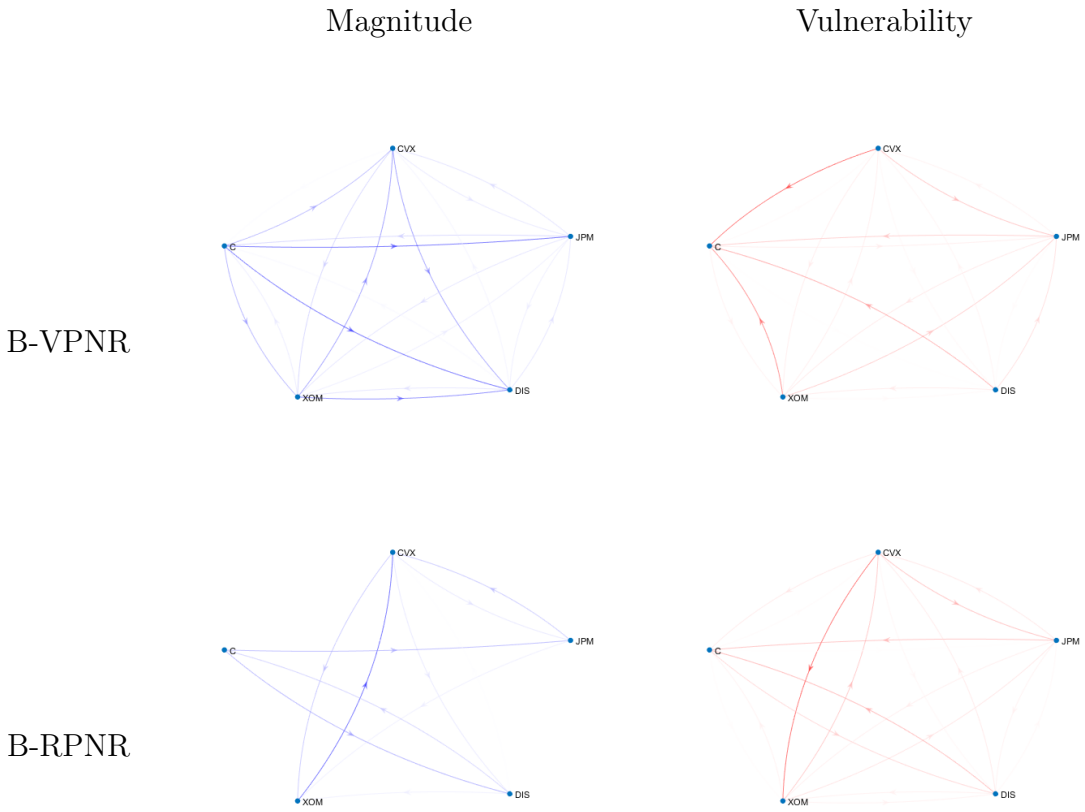


Figure 10.16 The normalized magnitude (top left) and vulnerability (top right) of links in Network 2 on Dataset 2 reconstructed using the B-VPNR Algorithm as given in Table 10.9 and the same (bottom left and right) using the B-RPNR Algorithm as given in Table 10.10.

Dataset 3

Link	Magnitude	Normalized Magnitude	Vulnerability	Normalized Vulnerability
XOM ← CVX	0.590	0.638	2.553	0.901
XOM ← C	0.210	0.227	1.358	0.479
XOM ← JPM	0.195	0.210	1.142	0.403
XOM ← DIS	0.268	0.289	1.885	0.665
CVX ← XOM	0.619	0.668	1.585	0.559
CVX ← C	0.797	0.861	0.512	0.181
CVX ← JPM	0.328	0.354	0.604	0.213
CVX ← DIS	0.463	0.500	0.710	0.251
C ← XOM	0.126	0.136	1.667	0.588
C ← CVX	0.155	0.168	2.834	1.000
C ← JPM	0.293	0.316	1.089	0.384
C ← DIS	0.196	0.212	1.144	0.404
JPM ← XOM	0.275	0.297	1.953	0.689
JPM ← CVX	0.191	0.207	1.962	0.692
JPM ← C	0.601	0.650	0.968	0.341
JPM ← DIS	0.133	0.143	1.487	0.525
DIS ← XOM	0.865	0.935	1.136	0.401
DIS ← CVX	0.461	0.498	1.737	0.613
DIS ← C	0.231	0.249	0.606	0.214
DIS ← JPM	0.926	1.000	0.930	0.328

Table 10.11 The magnitude and vulnerability of links in Network 2 (XOM, CVX, C, JPM, and DIS) using Dataset 3 (decisecond-resolution data) reconstructed using the B-VPNR Algorithm.

Link	Magnitude	Normalized Magnitude	Vulnerability	Normalized Vulnerability
XOM ← CVX	0.177	0.340	0.575	1.000
XOM ← C	0.000	0.000	0.000	0.000
XOM ← JPM	0.000	0.000	0.000	0.000
XOM ← DIS	0.000	0.000	0.000	0.000
CVX ← XOM	0.522	1.000	0.195	0.340
CVX ← C	0.000	0.000	0.000	0.000
CVX ← JPM	0.000	0.000	0.000	0.000
CVX ← DIS	0.000	0.000	0.000	0.000
C ← XOM	0.000	0.000	0.000	0.000
C ← CVX	0.000	0.000	0.000	0.000
C ← JPM	0.000	0.000	0.002	0.004
C ← DIS	0.357	0.685	0.000	0.001
JPM ← XOM	0.000	0.000	0.000	0.000
JPM ← CVX	0.000	0.000	0.000	0.000
JPM ← C	0.002	0.004	0.051	0.088
JPM ← DIS	0.235	0.450	0.142	0.247
DIS ← XOM	0.000	0.000	0.000	0.000
DIS ← CVX	0.000	0.000	0.000	0.000
DIS ← C	0.000	0.000	0.369	0.642
DIS ← JPM	0.138	0.264	0.242	0.421

Table 10.12 The magnitude and vulnerability of links in Network 2 (XOM, CVX, C, JPM, and DIS) using Dataset 3 (decisecond-resolution data) reconstructed using the B-RPNR Algorithm.

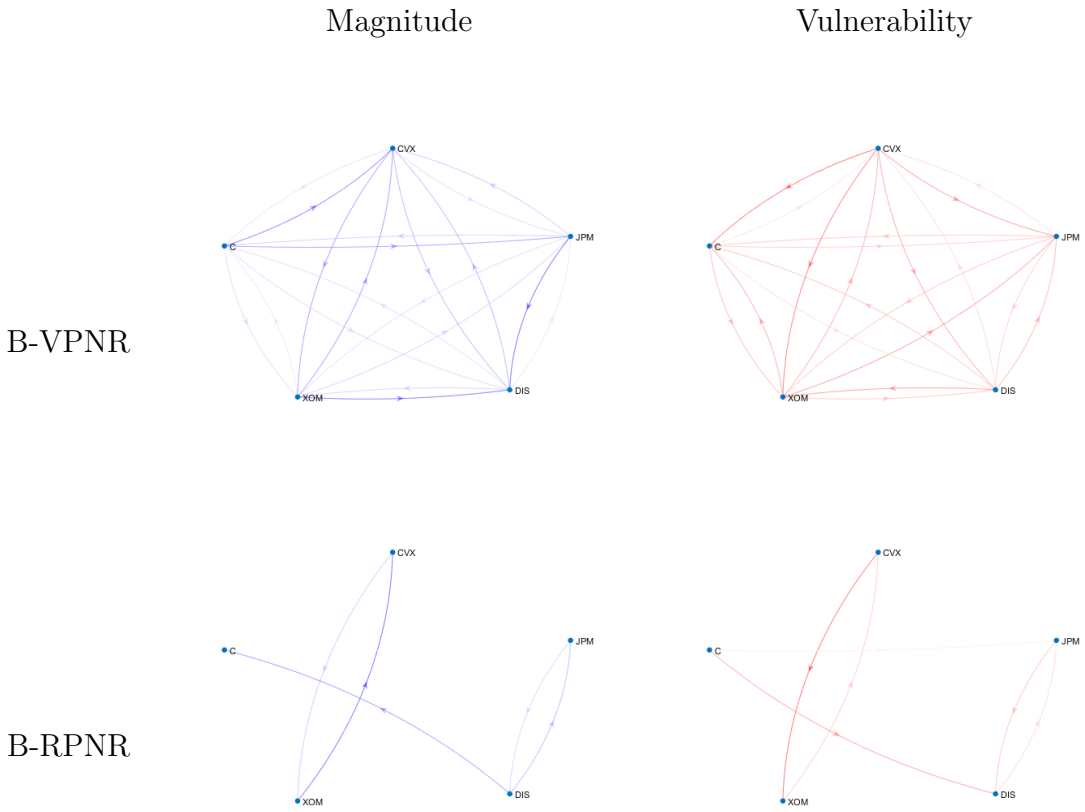


Figure 10.17 The normalized magnitude (top left) and vulnerability (top right) of links in Network 2 on Dataset 3 reconstructed using the B-VPNR Algorithm as given in Table 10.11 and the same (bottom left and right) using the B-RPNR Algorithm as given in Table 10.12.

10.5 Discussion

In the two reconstructed financial networks, features of the networks were discovered that were independent of both time and time resolution. These features both provide evidence that reconstruction on financial networks is possible as well as pose interesting questions to be investigated in future work.

Furthermore, we have shown that these networks are vulnerable to destabilizing attacks. Primarily, vulnerability analysis is a proof of concept, demonstrating the versatility of this vulnerability analysis. Future research, however, is merited to further understand these vulnerabilities and how to protect against them.

Chapter 11

Conclusions

In conclusion, we have extended an algorithm for passive network reconstruction of time-domain networks such that (a) the algorithm consistently computes an answer (the VPNR Algorithm), (b) given the proper assumptions, the algorithm converges to the correct answer faster than the original algorithm would, (c) the algorithm can be modified to be more robust to noise on the input-output data (the RPNR Algorithm), and (d) the algorithm can be modified to reconstruct even if the input data is not measured (the B-VPNR and B-RPNR Algorithms).

We then used stock market data from a variety of sources and used the B-VPNR and B-RPNR Algorithms on this data to reconstruct networks of financial data. Links in these networks represent the causal dependence of the price of one security on the changes in price of another security. We also analyzed these networks to detect which links in these networks are must susceptible to additive perturbations which could lead to a destabilization of the entire financial network. These additive perturbations take the form of trading one security based on the price of another security, and vulnerability is inversely related to the difficulty or expense required to create such a trading scheme to destabilize the market.

There is still a large amount of work to be done to solidify the theory of passive network reconstruction; nonetheless, this thesis is a demonstration of the functionality of the proposed algorithms and the types of analyses that can be performed on a reconstructed network.

Appendices

Appendix A

Implementation Notes

A Python implementation of network reconstruction methodology that is used in this and the subsequent chapters is available publicly at `https://gitlab.com/idealabs/netreco`.

Due to the sparsity of L and M, a CSR Matrix from the sparse matrix library from SciPy [39] was used to represent the data structures, enabling greater speed and lower memory consumption in the construction and execution of the algorithm.

The least squares algorithm used to learn the Toeplitz representation of Q and P from data for the non-robust (vanilla) network reconstruction algorithm was implemented using the least squares implementation included in the scipy.sparse.linalg library [41] in order to take advantage of the sparse matrix representation of M.

The least squares algorithm used to learn the Toeplitz representation of Q and P from data for the robust network reconstruction algorithm was implemented using the Lasso Model Selection: Cross Validation algorithm (LassoLarsIC) included in Scikit Learn's linear_model library [38], using Akaike's Information Criterion (AIC) to select the regularization parameter α. Unfortunately, this implementation requires dense matrices as inputs; therefore the sparse M had to be converted into a dense matrix before use.

The differential evolutionary algorithm in SciPy's optimization library [40] was used to fit the convolution model to the Toeplitz representation in both the vanilla and the robust network reconstruction algorithms. Though considerably slower in execution than the curve-fitting algorithms recommended in [10], it was found that the evolutionary algorithm

was able to find a good model more far more consistently than [10] using far less data (the correct model could be found consistently with $r = 100$ in the example above, where [10] stated a need for $r = 600$ with several restarts at random initial conditions).

The evolutionary algorithm only attempts to fit the b_i and c_i in the equation, constraining $a_k = -\sum_{i=1} w_k b_i$.

In order to fit the convolution model, the number of delays in the corresponding link (the order) as well as bounds on all b_i needs to be specified a priori (note also that given the assumption that the system is stable, all a_i are bounded such that $-1 < a_i < 1$). If the order and the bounds are chosen to be larger than needed, then the evolutionary algorithm may run slowly, but it will select the appropriate number of a_i and/or b_i to be zero (or two or more b_i that are nearly identical, allowing the terms to be summed) in order to fit the link with the proper error, thus choosing the remaining non-zero a_i and b_i correctly. However, if either the order or the bounds are selected to be too small, then the fit will not be good. For the running numeric example, since the order is at most 3 for any link, and the largest magnitude b_i on any link is 8.588, the evolutionary algorithm was chosen to have order 4 with b_i bounded at $-10 < b_i < 10$. As shown in the examples above, these choices led to near-perfect fits for the vanilla algorithm and good fits for the robust and blind algorithms. Wider bounds for b_i and significantly higher orders were also tried, with little impact on the results.

Appendix B

Source Code

The source code, not including the usage examples but including additional iPython Notebooks used to generate the results of this thesis, can be found publicly at `http://gitlab.com/idealabs/netreco`, master branch, commit `8154cd96`.

```
import time
from functools import partial

import numpy as np
import matplotlib.pyplot as plt

from scipy.sparse import csr_matrix
from scipy.sparse.linalg import lsqr
from scipy.optimize import differential_evolution

from sklearn import linear_model

from .ReconstructorLegacy import Profile

np.set_printoptions(precision=2, threshold=np.nan, linewidth=200,
                    suppress=True)


class Reconstructor:
    """Reconstructs a network from data.

    Available kwargs
    ----------------
    debug : bool, default=False
        True if debug or status messages should be displayed.
    r : int > 0
        The approximator for the convolution (smaller is faster, larger
        is more likely to reconstruct). If given equal or larger than f,
        will be thresholded to f-1 (where f is the number of data points
        passed into the algorithm's run function).
    order : number > 1, default=3
        w_k in the equations above. In other words, the number of poles
        to find in P and Q. There will be 2*order + 1 parameters in
        the fit function Q_ij(t) (a_k plus a b_i and a c_i for each i
        from 0 to order).
    bounds : number > 0, default=10
        All parameters b_i of the fit convolution Q_ij(t) (in other words,
        all a_k, b_i, and c_i) will be constrained to be between
        (-order,order). Note, all parameters c_i are assumed to be stable, or
        in other words, -1 < c_i < 1.
    robust : boolean, default=False
        True if the robust algorithm should be used (lasso least squares),
        False otherwise (regular least squares).
    T22 : numpy array (pm x 1) or None, default=None
        The a-priory information known about the system (see the paper)
        for more information. Only used by the measured reconstruction.
    Pbool : numpy array (p x m) containing only 0's and 1's, or None.
            Default=None
        The known boolean structure of P. (Probably will only work if
        each row and each column has exactly one 1). Only used by the measured
        reconstruction.
    criterion : str ('bic' (default) or 'aic')
        The information criterion to do the robust (Lasso) least squares.
        bic is the Bayes Information Criterion and aic is the Akaike
        Information Criterion.

    Sources
    -------
    [1] V. Chetty, J. Eliason and S. Warnick, "Passive Reconstruction of
        Non-Target-Specific Discrete-Time LTI Systems," American Control
        Conference, Boston, MA, 2016.
    """

    def __init__(self, **kwargs):
        self.debug = kwargs.get('debug', False)

        self.robust = kwargs.get('robust', False)
```

```
        self.alpha = kwargs.get('alpha', 1)
        self.T22 = kwargs.get('T22', None)
        self.Pbool = kwargs.get('Pbool', None)
        self.order = kwargs.get('order', 3)
        self.bounds = kwargs.get('bounds', 10)
        self.test_params = kwargs.get('test_params', False)
        self.criterion = kwargs.get('criterion', 'aic')  # or bic
        self.real_params = kwargs.get('real_params', {})

        self.r = kwargs.get('r', 3)

        self.executed = False

    ###########################################################################
    #   Main Algorithms
    ###########################################################################

    def measured(self, y, u, ij=None, plot=False, titles={}):
        """Runs a network reconstruction for measured inputs. See [1] for the
        algorithm. Can use the robust or non-robust algorithm.

        Attempts to build the convolutional representation of the DSF given by

            y(t) = Q(t)*y(t) + P(t)*u(t)

        where * denotes the convolution operator. Q(t) is an (p x p) matrix
        of equations (see below), and P(t) is a (p x m) matrix of equations.

        Let n = max(t-1, r), where r <= f (f is the number of datapoints in the
        input/output set). This convolution is then given by

            y(1) = 0  AND
            y(t) = sum_{i = 1}^{n} [Q(r)y(n-i+1) + Pi(t)u(n-i+1)],  t > 1

        (Note, setting r < f creates an approximation of the convolution, which
        works for large enough r since, for stable systems, Q(r) = 0 as r
        approaches infinity).

        Each Q_ij(t) is fit to a
        function of the form

            Q_ij(t) = a_k delta_t,0 + sum_{i = 0}^{w_k} b_i(c_i)^t

        For parameter set a_k, b_i, and c_i (there will be 1 + 2*w_k
        parameters).

        Parameters
        ----------
        y : numpy array (f x p)
            The measured outputs.
        u : numpy array (f x m)
            The measured inputs.
        ij : 2-tuple or None, default=None
            If given, specifies a single entry in Q to fit a convolution to,
            otherwise all entries in Q will be fit. Note: 1-indexed. So
            ij = (1, 3) refers to Qis[0][2].
        plot : boolean, default=False
            If True, plots the reconstruction results.
        titles : dict {int => str}
            Labels for each output i (1-indexed)

        Returns
        -------
        params : dict of dict of list (size self.order * 2)
            Each param[i][j] is a list of the best-fit parameters for Q(i, j).
        """
        if self.executed:
            raise Exception('Already used a reconstruction on this object.'
                            'Please create a new reconstruction object.')

        beginning = time.time()

        T22 = self.prepare_measured(y, u)
        LQrows, LQcols, LQdata = self.build_LQ(y)
        LProws, LPcols, LPdata = self.build_LP(
            u, T22, LQrows, LQcols, LQdata
        )
        M = self.build_M(LProws, LPcols, LPdata, is_measured=True)

        yhat = self.build_yhat(y)
        xhat = self.run_lsqr(M, yhat)

        Qis = self.extract_Qis(xhat)
        params = self.fit_convolution(Qis, ij, plot, titles)

        self.executed = True

        self.dprint('\nReconstruction Complete! Took {:.3f} Seconds\n'.format(
            time.time() - beginning
        ))

        return params

    def unmeasured(self, y, ij=None, plot=False, titles={}):
        """Runs a network reconstruction for unmeasured inputs. See [1] for the
        algorithm. Must use the robust algorithm (forces it to True).

        Attempts to build the convolutional representation of the DSF given by

            y(t) = Q(t)*y(t) + Delta(t)

        where * denotes the convolution operator. Q(t) is an (p x p) matrix
```

```
        of equations (see below), and P(t) is a (p x m) matrix of equations.

        Let n = max(t-1, r), where r <= f (f is the number of datapoints in the
        input/output set). This convolution is then given by

            y(1) = 0  AND
            y(t) = sum_{i = 1}^{n} [Q(r)y(n-i+1) + Delta(t)],  t > 1

        (Note, setting r < f creates an approximation of the convolution, which
        works for large enough r since, for stable systems, Q(r) = 0 as r
        approaches infinity).

        Each Q_ij(t) is fit to a
        function of the form

            Q_ij(t) = a_k delta_t,0 + sum_{i = 0}^{w_k} b_i(c_i)^t

        For parameter set a_k, b_i, and c_i (there will be 1 + 2*w_k
        parameters).

        Parameters
        ----------
        y : numpy array (f x p)
            The measured outputs.
        ij : 2-tuple or None, default=None
            If given, specifies a single entry in Q to fit a convolution to,
            otherwise all entries in Q will be fit. Note: 1-indexed. So
            ij = (1, 3) refers to Qis[0][2].
        plot : boolean, default=False
            If True, plots the reconstruction results.
        titles : dict {int => str}
            Labels for each output i (1-indexed)

        Returns
        -------
        params : dict of dict of list (size self.order * 2)
            Each param[i][j] is a list of the best-fit parameters for Q(i, j).
        """
        if self.executed:
            raise Exception('Already used a reconstruction on this object.'
                            'Please create a new reconstruction object.')

        beginning = time.time()

        self.prepare_unmeasured(y)
        LQrows, LQcols, LQdata = self.build_LQ(y)
        M = self.build_M(LQrows, LQcols, LQdata, is_measured=False)

        yhat = self.build_yhat(y)
        xhat = self.run_lsqr(M, yhat)

        Qis = self.extract_Qis(xhat)
        params = self.fit_convolution(Qis, ij, plot, titles)

        self.executed = True

        self.dprint('\nReconstruction Complete! Took {:.3f} Seconds\n'.format(
            time.time() - beginning
        ))

        return params

    ##########################################################################
    #   Algorithm Steps
    ##########################################################################

    def prepare_measured(self, y, u):
        """Sets up a measured reconstruction.

        Parameters
        ----------
        y : numpy array (f x p)
            The measured outputs.
        u : numpy array (f x m)
        """
        with Profile(self.debug, 'Preparing for a Measured Reconstruction'):
            f, p = y.shape
            f1, m = u.shape

            self.r = min(self.r, f - 1)

            assert f == f1

            self.p = p    # Number of outputs
            self.m = m    # Number of inputs
            self.f = f    # Number of data points

            if self.Pbool is not None:
                p1, m1 = self.Pbool.shape
                assert p == p1
                assert m == m1
                self.T22 = self._build_T22(self.Pbool)
                ps = p1 * m1 - len(self.T22)
            else:
                raise Exception('Not implemented - cannot reconstruct with '
                                'measured inputs without a Pbool.')

            self.ps = ps

        return self.T22

    def prepare_unmeasured(self, y):
```

```
        """Sets up a measured reconstruction.

        Parameters
        ----------
        y : numpy array (f x p)
            The measured outputs.
        """
        with Profile(self.debug, 'Preparing for an Unmeasured Reconstruction'):
            f, p = y.shape

            self.r = min(self.r, f - 1)

            self.p = p    # Number of outputs
            self.f = f    # Number of data points

            # self.robust = True

    def build_LQ(self, y):
        """
        Builds the Q half of L (see paper [1]).

        Parameters
        ----------
        y : numpy array (f x p)

        Returns
        -------
        LQ : numpy array (p(f-1) x r(p^2 - p))
            The Q half of L. Note that the paper is not entirely correct on the
            construction of L. It only goes to r(f-1) rows, using the first f-1
            points from y. Also, either L has r rows of zeros at the beginning
            (making rf rows) or yhat goes from y2 to yf. We choose the latter.
        """
        with Profile(self.debug, 'Building LQ'):
            width = self.p * self.p - self.p

            # Ignore all Q_ii
            to_ignore = [()]
            count = 0
            for i in range(self.p):
                for j in range(self.p):
                    if i == j:
                        to_ignore.append(count)
                    count += 1

            # Build the individual blocks
            Brows = [None for i in range(self.f - 1)]
            Bcols = [None for i in range(self.f - 1)]
            Bdats = [None for i in range(self.f - 1)]
            for i in range(self.f - 1):
                Brows[i], Bcols[i], Bdats[i] = \
                    self._build_block(y[i, :], to_ignore)

            # Build LQ
            rows = []
            cols = []
            data = []

            for i in range(self.f - 1):
                start_row = i * self.p
                for j in range(min(i + 1, self.r)):
                    start_col = j * width
                    block_index = i - j

                    rows.append(Brows[block_index] + start_row)
                    cols.append(Bcols[block_index] + start_col)
                    data.append(Bdats[block_index])

        return rows, cols, data

    def build_LP(self, u, T22, rows, cols, data, col_offset=None):
        """
        Builds the P half of L (see paper [1]).

        Parameters
        ----------
        u : numpy array (f x p)
        T22 : numpy array (mp x l)

        Returns
        -------
        LQ : numpy array (p(f-1) x rl))
            The P half of L. Note that the paper is not entirely correct on the
            construction of L. It only goes to r(f-1) rows, using the first f-1
            points from y. Also, either L has r rows of zeros at the beginning
            (making rf rows) or yhat goes from y2 to yf. We choose the latter.
        """
        with Profile(self.debug, 'Building LP'):

            if col_offset is None:
                col_offset = self.r * (self.p * self.p - self.p)

            l = self.m * self.p - len(T22)

            # Build the individual blocks
            Brows = [None for i in range(self.f - 1)]
            Bcols = [None for i in range(self.f - 1)]
            Bdats = [None for i in range(self.f - 1)]
            for i in range(self.f - 1):
                Brows[i], Bcols[i], Bdats[i] = self._build_block(u[i, :], T22)

            # Build LP
```

```
            for i in range(self.f - 1):
                start_row = i * self.p
                for j in range(min(i + 1, self.r)):
                    start_col = col_offset + j * l
                    block_index = i - j

                    rows.append(Brows[block_index] + start_row)
                    cols.append(Bcols[block_index] + start_col)
                    data.append(Bdats[block_index])

        return rows, cols, data

    def build_M(self, rows, cols, data, is_measured):
        """Builds the M sparse matrix from LQ (if unmeasured) or LP (if
        measured).

        Parameters
        ----------
        rows : list of lists
            The row indices of non-zero entries of LQ or LP.
        cols : list of lists
            The col indices of non-zero entries of LQ or LP. 1-1 with rows.
        data : list of lists
            The non-zero entry values of LQ or LP. 1-1 with rows and cols
            (meaning M[rows[i], cols[i]] = data[i]).

        Returns
        -------
        M : csr_matrix
        """
        with Profile(self.debug, 'Building M'):
            height = (self.f - 1) * self.p
            if is_measured:
                width = self.r * (self.ps + self.p * self.p - self.p)
            else:
                width = self.r * (self.p * self.p - self.p)

            M = csr_matrix((np.concatenate(data),
                            (np.concatenate(rows), np.concatenate(cols))),
                           (height, width))

        return M

    def build_yhat(self, y):
        """
        Stacks y into [y2' ... y2' ... yf' ... yf']''

        Parameters
        ----------
        y : numpy array (f x p)

        Returns
        -------
        yhat : numpy array (p(f-1) x 1)
        """
        with Profile(self.debug, 'Building yhat'):
            ybar = y[1:, :]
            yhat = ybar.reshape(self.p * (self.f - 1), 1)

        return yhat

    def run_lsqr(self, M, yhat):
        """
        Finds the xhat that "best" fits yhat = M*xhat.

        If the robust algorithm is not running, this is regular least squares,
        or in other words:

            xhat = argmin_x || yhat - M*xhat ||_2

        If the robust algorithm is running, this is the lasso relaxation of
        the sparse least squares, or in other words:

            xhat = argmin_x || yhat - M*xhat ||_2 + alpha * || xhat ||_1

        where alpha is set when the algorithm is called.
        """
        with Profile(self.debug, 'Running Least Squares to get xhat'):
            if self.robust:
                model = linear_model.LassoLarsIC(criterion=self.criterion)
                y = list(yhat.reshape(max(yhat.shape)))
                model.fit(M.toarray(), y)

                self.dprint(
                    '\tFinished in {} iterations'.format(model.n_iter_)
                )
                xhat = np.array(model.coef_).reshape(
                    (len(model.coef_), 1)
                )

                xhat = np.array(model.coef_).reshape(
                    (len(model.coef_), 1)
                )

            else:
                x, istop, itn, r1norm, r2norm, arnorm, acond, arnorm, xnorm, \
                    var = lsqr(M, yhat)
                xhat = np.array(x).reshape((len(x), 1))
                self.dprint('\tFinished lsq: Error = {:.3f}'.format(r1norm))

        return xhat
```

```
    def extract_Qis(self, xhat):
        """
        Extracts the Qi's from the given xhat vector.
        """
        with Profile(self.debug, 'Extracting Qis'):
            Qis = {}
            width = self.p * self.p - self.p

            for i in range(self.r):
                vec = xhat[i * width: i * width + width]

                pos = 0
                for j in range(self.p):

                    if j not in Qis:
                        Qis[j] = {}

                    for k in range(self.p):
                        if j == k:
                            continue

                        curr = Qis[j].get(k, [0])
                        curr.append(vec[pos][0])
                        Qis[j][k] = curr

                        pos += 1

        return Qis

    def fit_convolution(self, Qis, ij, plot, titles):
        """Fits the convolutional representation to Q.
        """
        if ij is not None:
            i = ij[0]
            j = ij[1]
            sub = self._fit_ij_conv(Qis, i - 1, j - 1, plot)

            params = {i: {j: sub}}
            return params
        else:
            params = {}
            for i in range(self.p):
                params[i] = {}
                for j in range(self.p):
                    if i == j:
                        continue

                    params[i][j] = self._fit_ij_conv(Qis, i, j, plot, titles)

            return params

    ###########################################################################
    #    Helpers
    ###########################################################################

    def _fit_ij_conv(self, Qis, i, j, plot, titles):
        """Fits the convolutional representation to Q(i,j)
        """
        with Profile(self.debug, 'Learning Convolution for Q({},{})'.format(
            i + 1, j + 1
        )):

            t = list(range(self.r + 1))
            vals = Qis[i][j]

            bounds = []
            stab_tol = 0.001  # Prevent the system from being marginally stable
            for k in range(self.order):
                bounds.append((-self.bounds, self.bounds))
                bounds.append((-1 + stab_tol, 1 - stab_tol))

            assert len(t) == len(vals)

            obj = partial(self._fiterr, act=vals)
            rs = differential_evolution(obj, bounds=bounds)

            if plot:
                err, exp = self._fiterr(rs.x, vals, True)

                plt.figure(figsize=(10, 8))
                plt.scatter(t, vals, c='r', label='Data (xhat)')
                plt.plot(t, exp, c='b',
                         label='Reconstructed Convolution Model')

                if (i + 1, j + 1) in self.real_params:
                    rp = self.real_params[(i + 1, j + 1)]
                    err1, real = self._fiterr(rp, vals, True)
                    plt.plot(t, real, color='g', linestyle='-.',
                             label='Actual Convolution Model')

                plt.xlim((0, self.r))
                plt.xlabel('t')

                linkname = ''
                if (i + 1) in titles and (j + 1) in titles:
                    linkname = ': {} <-- {}'.format(titles[i + 1],
                                                    titles[j + 1])

                plt.title('Q(t)[{},{}]{}'.format(i + 1, j + 1, linkname))
                plt.legend()
```

```
            self._print_dsf(i, j, rs.x, t, vals, titles)

        return rs.x

    def _fiterr(self, params, act, return_exp=False):
        """Computes the error of a convolutional representation parameterized
        by params to the data act.
        """
        n = len(act)
        exp = np.zeros(n)
        for t in range(1, n):
            for i in range(int(len(params) / 2)):
                exp[t] += params[i * 2] * params[i * 2 + 1] ** t

        sqerr = (np.array(act) - exp) ** 2
        rmse = np.sqrt(sqerr.mean())
        # abserr = abs(np.array(act) - exp)
        # rmse = abserr.mean()

        # print(toterr)

        if return_exp:
            return rmse, exp
        else:
            return rmse

    def _build_T22(self, PBool):
        """
        Takes a boolean structure of P and cconverts it into a T22.
        """
        to_delete = []

        index = 0
        for i in range(self.p):
            for j in range(self.m):
                if PBool[i, j] == 0:
                    to_delete.append(index)

                index += 1

        return to_delete

    def _build_block(self, v, to_ignore=[]):
        """
        Builds the diagonal portion of the block used in both LQ and LP, which
        is [v' 0 ... 0; 0 v' ... 0 ; ... ; 0 0 ... v'], where the number of
        rows is p.

        The block is actually just a dictionary of {(row, col) => value}

        Parameters
        ----------
        v : numpy array (length l = length p or length m or arbitrary)
            The data belonging in the block. It is either a vector from u or
            from y.
        to_ignore : list (len <= p*l)
            Columns to remove from the block, where the index is referring
            to one of the p*l columns, not the l entries of v.

        Returns
        -------
        row : np.array (len = p*l - len(to_ignore))
            The row indices of the non-zero data
        col : np.array (len = p*l - len(to_ignore))
            The column indices of the non-zero data, corresponding to row
        data : np.array (len = p*l - len(to_ignore))
            The non-zero entries, where data[i] is located at index
            (row[i], col[i])
        """
        vt = v.transpose().tolist()
        l = len(vt)

        row = []
        col = []
        data = []
        loc = 0
        for i in range(self.p):
            for j in range(l):
                pos = i * l + j
                if pos in to_ignore:
                    continue

                row.append(i)
                col.append(loc)
                data.append(vt[j])
                loc += 1

        return np.array(row), np.array(col), np.array(data)

    def _print_dsf(self, i, j, params, t, data, titles):
        if not self.debug:
            return

        linkname = ''
        if (i + 1) in titles and (j + 1) in titles:
            linkname = ' {} <-- {}'.format(titles[i + 1], titles[j + 1])

        msg = '\tQ({},{}){}: '.format(i + 1, j + 1, linkname)
        poles = []

        a = 0
        for i in range(self.order):
```

```
            b = i * 2
            c = b + 1
            poles.append('{:.3f}*({:.3f})^t'.format(params[b], params[c]))
            a += -params[b]

        msg += ' + '.join(poles)
        msg += ' + {:3f} * delta(t,0)'.format(a)
        print(msg)

        err = self._fiterr(params, data)
        # actual = np.array(data)

        # errors = actual - expected
        # rmse = np.sqrt((errors ** 2).mean())
        print('\t\tRMSE = {:.3f}'.format(err))

        print('\t\tMatlab: {}'.format(params))

    def dprint(self, msg):
        """
        Prints the debug msg `msg` conditional on debug being set to True.
        """
        if self.debug:
            print(msg)
```

Listing B.1 `Reconstructor.py`: Library for reconstructing networks from data. Contains the VPNR, RPNR, B-VPNR, and B-RPNR.

```
import numpy as np


def ss_sim(A, B, C, u):
    """
    Simulates the discrete time system (assuming x[0] = 0):

        x[k+1] = Ax[k] + Bu[k]
        y[k] = Cx[k]

    Parameters
    ----------
    A : numpy array (n x n)
    B : numpy array (n x m)
    C : numpy array (p x n)
    u : numpy array (f x m)
        Row i is u[i]'

    Returns
    -------
    y : numpy array (f x p)
        Row i is y[i]'
    """
    n, n1 = A.shape
    n2, m = B.shape
    p, n3 = C.shape
    f, m1 = u.shape

    assert n == n1
    assert n == n2
    assert n == n3
    assert m == m1

    xi = np.zeros((n, 1))[:, 0]
    y = np.zeros((f, p))

    for i in range(f):
        y[i] = C.dot(xi)
        xi = A.dot(xi) + B.dot(u[i])

    return y
```

Listing B.2 `ss.py`: Computes the outputs of the given state space model over time using the given inputs.

```
import numpy as np
from netreco import ss_sim


A = [
    [0.75, 0, 0, 0, 0, 1.2],
    [-.1, -.35, 0, 0, 0, 0],
    [0, 0, .85, -1, 0, 0],
    [0, -.73, 0, .95, 0, 0],
    [0, 0, .43, 0, -.6, 0],
    [0, 0, 0, 0, .2, .55]

]
A = np.array(A)
B = [
    [1.4, 0, 0],
    [0, -.25, 0],
    [0, 0, 0.75],
    [0, 0, 0],
    [0, 0, 0],
    [0, 0, 0]
]
B = np.array(B)
```

```
C = [
    [1, 0, 0, 0, 0, 0],
    [0, 1, 0, 0, 0, 0],
    [0, 0, 1, 0, 0, 0]
]
C = np.array(C)

u = np.random.rand(601, 3) * 2 - 1
y = ss_sim(A, B, C, u)
```

Listing B.3 `ex_ss.py`: Example usage of `ss.py`. Generates inputs $\mathcal{D}_u \in \mathbb{R}^{T \times m}$, where $m = 3$, $T = 601$, and each entry is taken from a uniform distribution spanning from -1 to 1. The script then simulates the outputs of the state space system introduced in Section 6.5.

```
import numpy as np

from netreco import Reconstructor
from ex_ss import u, y

r = 100

# The a priori information known about the system.
# We assume target specificity, meaning each input affects exactly one measured state
# and each measured state is affected by exactly one input.
Pbool = np.identity(3)

# The parameters of the actual convolutional model.
# Used for plotting and verifying that the reconstructed network is correct.
real_params = {
    (1, 3): [.51, .75, -.11, -.6, -.816, .55],
    (2, 1): [.286, -.35],
    (3, 2): [7.684, .95, -8.588, .85]
}

recon = Reconstructor(debug=True, r=r, Pbool=Pbool, max_iterations=1000, bounds=10, order=4,
                      real_params=real_params, robust=False)
parameters = recon.measured(y, u, plot=True)
```

Listing B.4 `ex_vpnr.py`: Example usage of the VPNR Algorithm.

```
import numpy as np

from netreco import Reconstructor
from ex_ss import u, y

r = 100

# The a priori information known about the system.
# We assume target specificity, meaning each input affects exactly one measured state
# and each measured state is affected by exactly one input.
Pbool = np.identity(3)

# The parameters of the actual convolutional model.
# Used for plotting and verifying that the reconstructed network is correct.
real_params = {
    (1, 3): [.51, .75, -.11, -.6, -.816, .55],
    (2, 1): [.286, -.35],
    (3, 2): [7.684, .95, -8.588, .85]
}

recon = Reconstructor(debug=True, r=r, Pbool=Pbool, max_iterations=1000, bounds=10, order=4,
                      real_params=real_params, robust=True)
parameters = recon.measured(y, u, plot=True)
```

Listing B.5 `ex_rpnr.py`: Example usage of the RPNR Algorithm.

```
import numpy as np

from netreco import Reconstructor
from ex_ss import u, y

r = 100

# The a priori information known about the system.
# We assume target specificity, meaning each input affects exactly one measured state
# and each measured state is affected by exactly one input.
Pbool = np.identity(3)

# The parameters of the actual convolutional model.
# Used for plotting and verifying that the reconstructed network is correct.
real_params = {
    (1, 3): [.51, .75, -.11, -.6, -.816, .55],
    (2, 1): [.286, -.35],
    (3, 2): [7.684, .95, -8.588, .85]
}

recon = Reconstructor(debug=True, r=r, Pbool=Pbool, max_iterations=1000, bounds=10, order=4,
                      real_params=real_params, robust=False)
parameters = recon.unmeasured(y, u, plot=True)
```

Listing B.6 `ex_bvpnr.py`: Example usage of the B-VPNR Algorithm.

```python
import numpy as np

from netreco import Reconstructor
from ex_ss import u, y

r = 100

# The a priori information known about the system.
# We assume target specificity, meaning each input affects exactly one measured state
# and each measured state is affected by exactly one input.
Pbool = np.identity(3)

# The parameters of the actual convolutional model.
# Used for plotting and verifying that the reconstructed network is correct.
real_params = {
    (1, 3): [.51, .75, -.11, -.6, -.816, .55],
    (2, 1): [.286, -.35],
    (3, 2): [7.684, .95, -8.588, .85]
}

recon = Reconstructor(debug=True, r=r, Pbool=Pbool, max_iterations=1000, bounds=10, order=4,
                      real_params=real_params, robust=True)
parameters = recon.unmeasured(y, u, plot=True)
```

Listing B.7 `ex_brpnr.py`: Example usage of the B-RPNR Algorithm.

References

[1] Julius Adebayo, Taylor Southwick, Vasu Chetty, Enoch Yeung, Ye Yuan, Jorge Goncalves, J Grose, J Prince, Guy-Bart Stan, and Sean Warnick. Dynamical structure function identifiability conditions enabling signal structure reconstruction. In *Decision and Control (CDC), 2012 IEEE 51st Annual Conference on*, pages 4635–4641. IEEE, 2012.

[2] Saurabh Amin, Alvaro A Cárdenas, and S Shankar Sastry. Safe and secure networked control systems under denial-of-service attacks. In *Hybrid Systems: Computation and Control*, pages 31–45. Springer, 2009.

[3] Cristina Basescu, Raphael M Reischuk, Pawel Szalachowski, Adrian Perrig, Yao Zhang, Hsu-Chun Hsiao, Ayumu Kubota, and Jumpei Urakawa. Sibra: Scalable internet bandwidth reservation architecture. *arXiv preprint arXiv:1510.02696*, 2015.

[4] Rakesh B Bobba, Katherine M Rogers, Qiyan Wang, Himanshu Khurana, Klara Nahrstedt, and Thomas J Overbye. Detecting false data injection attacks on dc state estimation. In *Preprints of the First Workshop on Secure Control Systems, CPSWEEK*, volume 2010, 2010.

[5] Alvaro A Cárdenas, Saurabh Amin, and Shankar Sastry. Research challenges for the security of control systems. In *Hot Topics in Security (HotSec), 2008 3rd Usenix Workshop on*, 2008.

[6] Vasu Chetty. Necessary and sufficient informativity conditions for robust network reconstruction using dynamical structure functions. M.S. Thesis, Brigham Young University, Provo, UT, 2012.

[7] Vasu Chetty and Sean Warnick. Network semantics of dynamical systems. In *Decision and Control (CDC), 2015 IEEE 54th Annual Conference on*, pages 1557–1562. IEEE, 2015.

[8] Vasu Chetty, Nathan Woodbury, Elham Vaziripour, and Sean Warnick. Vulnerability analysis for distributed and coordinated destabilization attacks. In *Decision and Control (CDC), 2014 IEEE 53rd Annual Conference on*, pages 511–516. IEEE, 2014.

[9] Vasu Chetty, Nathan Woodbury, and Sean Warnick. Farming as feedback control. In *American Control Conference (ACC), 2014*, pages 2688–2693. IEEE, 2014.

[10] Vasu Chetty, Joel Eliason, and Sean Warnick. Passive reconstruction of non-target-specific discrete-time lti systems. In *American Control Conference (ACC), 2016*, pages 66–71. IEEE, 2016.

[11] Vasu Chetty, Nathan Woodbury, Jacob Brewer, Kenneth Lee, and Sean Warnick. Applying a passive network reconstruction technique to twitter data in order to identify trend setters. Submitted for publication at the IEEE Conference on Control Technology and Applications, 2017.

[12] Ann Cox, Sandip Roy, and Sean Warnick. A science of system security. In *Decision and Control (CDC), 2014 IEEE 53rd Annual Conference on*, pages 487–492. IEEE, 2014.

[13] Mohammed Dahleh, Munther Dahleh, and George Verghese. *Dynamic systems and control*. MIT OpenCourseWare, 2011. URL `http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-241j-dynamic-systems-and-control-spring-2011/readings/`. online textbook.

[14] Peter Fairley. Blackout threat unmitigated a decade after the northeast went dark. *IEEE Spectrum*, August 2013. URL `http://spectrum.ieee.org/energywise/energy/the-smarter-grid/blackout-threat-unmitigated-a-decade-after-the-northeast-went-dark`. [Online; posted 14-August-2013].

[15] Nicolas Falliere, Liam O Murchu, and Eric Chien. W32. stuxnet dossier. *White paper, Symantec Corp., Security Response*, 5, 2011.

[16] Jorge Gonçalves and Sean Warnick. Necessary and sufficient conditions for dynamical structure reconstruction of lti networks. *IEEE Transactions on Automatic Control*, 53 (7):1670–1674, 2008.

[17] David Grimsman, Vasu Chetty, Nathan Woodbury, Elham Vaziripour, Sandip Roy, Daniel Zappala, and Sean Warnick. A case study of a systematic attack design method for critical infrastructure cyber-physical systems. In *American Control Conference (ACC), 2016*, pages 296–301. IEEE, 2016.

[18] Paul Hines, Karthikeyan Balasubramaniam, and Eduardo Cotilla Sanchez. Cascading failures in power grids. *Potentials, IEEE*, 28(5):24–30, 2009.

[19] Min Suk Kang, Virgil D Gligor, and Vyas Sekar. Spiffy: Inducing cost-detectability tradeoffs for persistent link-flooding attacks. *Network and Distributed System Security Symposium*, 2016. URL `http://www.comp.nus.edu.sg/~kangms/papers/spiffy.pdf`.

[20] Ralph Langner. Stuxnet: Dissecting a cyberwarfare weapon. *Security & Privacy, IEEE*, 9(3):49–51, 2011.

[21] Edward A Lee. Cyber physical systems: Design challenges. In *Object Oriented Real-Time Distributed Computing (ISORC), 2008 11th IEEE International Symposium on*, pages 363–369. IEEE, 2008.

[22] William S Levine. *The control handbook*. CRC press, 1996.

[23] John Leyden. Polish teen derails tram after hacking train network. *The Register*, January 2008. URL `https://www.theregister.co.uk/2008/01/11/tram_hack/`. [Online; posted 11-January-2008].

[24] Yao Liu, Peng Ning, and Michael K Reiter. False data injection attacks against state estimation in electric power grids. *ACM Transactions on Information and System Security (TISSEC)*, 14(1):13, 2011.

[25] Men Long, Chwan-Hwa Wu, and John Y Hung. Denial of service attacks on network-based control systems: impact and mitigation. *Industrial Informatics, IEEE Transactions on*, 1(2):85–96, 2005.

[26] Burton G Malkiel. The efficient market hypothesis and its critics. *The Journal of Economic Perspectives*, 17(1):59–82, 2003.

[27] Donatello Materassi and Giacomo Innocenti. Topological Identification in Networks of Dynamical Systems. *Transactions on Automatic Control*, 55(8):1860–1871, 2010.

[28] Raman K Mehra and J Peschon. An innovations approach to fault detection and diagnosis in dynamic systems. *Automatica*, 7(5):637–640, 1971.

[29] Yilin Mo and Bruno Sinopoli. Secure control against replay attacks. In *Communication, Control, and Computing, 2009. Allerton 2009. 47th Annual Allerton Conference on*, pages 911–918. IEEE, 2009.

[30] Yilin Mo, Emanuele Garone, Alessandro Casavola, and Bruno Sinopoli. False data injection attacks against state estimation in wireless sensor networks. In *Decision and Control (CDC), 2010 49th IEEE Conference on*, pages 5967–5972. IEEE, 2010.

[31] NASDAQ Trader. Nadaq totalview-itch. http://www.nasdaqtrader.com/Trader.aspx?id=Totalview2, 2017. Accessed: 2017-02-13.

[32] President's Commission on Critical Infrastructure Protection. Critical foundations: Protecting America's infrastructures, 1997. URL https://fas.org/sgp/library/pccip.pdf.

[33] Philip E Paré, Vasu Chetty, and Sean Warnick. On the necessity of full-state measurement for state-space network reconstruction. In *Global Conference on Signal and Information Processing (GlobalSIP), 2013 IEEE*, pages 803–806. IEEE, 2013.

[34] Christopher Quinn, Negar Kiyavash, and Todd Coleman. Directed Information Graphs. *Transactions on Information Theory*, 61(12):6887–6909, 2015.

[35] Anurag Rai, David Ward, Sandip Roy, and Sean Warnick. Vulnerable links and secure architectures in the stabilization of networks of controlled dynamical systems. In *American Control Conference (ACC), 2012*, pages 1248–1253. IEEE, 2012.

[36] Henrik Sandberg, Saurabh Amin, and K Johansson. Cyberphysical security in networked control systems: an introduction to the issue. *Control Systems, IEEE*, 35(1):20–23, 2015.

[37] Christoph Schuba, Ivan Krsul, Markus Kuhn, Eugene Spafford, Aurobindo Sundaram, and Diego Zamboni. Analysis of a denial of service attack on tcp. In *Security and Privacy, 1997 IEEE Symposium on*, pages 208–223. IEEE, 1997.

[38] Scikit Learn Lasso. Lasso model selection: Cross-validation aic/bic. http://scikit-learn.org/stable/auto_examples/linear_model/plot_lasso_model_selection.html, 2016. Accessed: 2016-11-08.

[39] Scipy. Sparse matrices (scipy.sparse). https://docs.scipy.org/doc/scipy-0.18.1/reference/sparse.html, 2016. Accessed: 2016-11-08.

[40] Scipy Optimize Differential Evolution. scipy.optimize.differential_evolution. https://docs.scipy.org/doc/scipy-0.18.1/reference/generated/scipy.optimize.differential_evolution.html, 2016. Accessed: 2016-11-08.

[41] Scipy Sparse Least Squares. scipy.sparse.linalg.lsqr. https://docs.scipy.org/doc/scipy-0.18.1/reference/generated/scipy.sparse.linalg.lsqr.html#scipy.sparse.linalg.lsqr, 2016. Accessed: 2016-11-08.

[42] Jill Slay and Michael Miller. *Lessons learned from the maroochy water breach*. Springer, 2007.

[43] Antonio Teixeira, Kin Cheong Sou, Henrik Sandberg, and Karl H Johansson. Secure control systems: A quantitative risk management approach. *Control Systems, IEEE*, 35 (1):24–45, 2015.

[44] Fei-Yue Wang and Derong Liu. *Networked control systems*. Springer, 2008.

[45] Sean Warnick. Shared hidden state and network representations of interconnected dynamical systems. In *Communication, Control, and Computing (Allerton), 2015 53rd Annual Allerton Conference on*, pages 25–32. IEEE, 2015.

[46] Nathan Woodbury, Vasu Chetty, and Sean Warnick. Robust passive reconstruction of stable discrete-time lti networks. Submitted for publication at the IEEE Conference on Decision and Control, 2017.

[47] Nathan Woodbury, Vasu Chetty, and Sean Warnick. Passive reconstruction of stable discrete-time lti networks using an evolutionary algorithm. Submitted for publication at the IEEE Conference on Decision and Control, 2017.

[48] Yahoo Finance. Yahoo finance. `http://finance.yahoo.com`, 2017. Accessed: 2017-02-13.

[49] Ye Yuan, Guy-Bart Stan, Sean Warnick, and Jorge Gonçalves. Robust dynamical network reconstruction. In *Decision and Control (CDC), 2010 49th IEEE Conference on*, pages 810–815. IEEE, 2010.

[50] Ye Yuan, Guy-Bart Stan, Sean Warnick, and Jorge Goncalves. Robust dynamical network structure reconstruction. *Automatica*, 47(6):1230–1235, 2011.

[51] Bonnie Zhu, Anthony Joseph, and Shankar Sastry. A taxonomy of cyber attacks on scada systems. In *Internet of things (iThings/CPSCom), 2011 international conference on and 4th international conference on cyber, physical and social computing*, pages 380–388. IEEE, 2011.