PH.D. THESIS

Approximate Matrix Diagonalization for Use in Distributed Control Networks

by George A. Kantor
Advisor: P.S. Krishnaprasad

CDCSS Ph.D. 99-3
(ISR Ph.D. 99-7)

CENTER FOR DYNAMICS AND CONTROL OF SMART STRUCTURES

The Center for Dynamics and Control of Smart Structures (CDCSS) is a joint Harvard University, Boston University, University of Maryland center, supported by the Army Research Office under the ODDR&E MURI97 Program Grant No. DAAG55-97-1-0114 (through Harvard University). This document is a technical report in the CDCSS series originating at the University of Maryland.

Web site http://www.isr.umd.edu/CDCSS/cdcss.html

ABSTRACT

Title of Dissertation: APPROXIMATE MATRIX DIAGONALIZATION FOR USE IN DISTRIBUTED CONTROL NETWORKS

George A. Kantor, Doctor of Philosophy, 1999

Dissertation directed by: Professor P.S. Krishnaprasad
Department of Electrical and Computer Engineering

Distributed control networks are rapidly emerging as a viable and important alternative to centralized control. In a typical distributed control network, a number of spatially distributed nodes composed of "smart" sensors and actuators are used to take measurements and apply control inputs to some physical plant. The nodes have local processing power and the ability to communicate with the other nodes via a network. The challenge is to compute and implement a feedback law for the resulting MIMO system in a distributed manner on the network.

Our approach to this problem is based on plant diagonalization. To do this, we search for basis transformations for the vector of outputs coming from the sensors

and the vector of inputs applied to the actuators so that, in the new bases, the MIMO system becomes a collection of decoupled SISO systems. This formulation provides a number of advantages for the synthesis and implementation of a feedback control law, particularly for systems where the number of inputs and outputs is large. Of course, in order for this idea to be feasible, the required basis transformations must have properties which allow them to be implemented on a distributed control network. Namely, they must be computed in a distributed manner which respects the spatial distribution of the data (to reduce communication overhead) and takes advantage of the massive parallel processing capability of the network (to reduce computation time).

In this thesis, we present some tools which can be used to find suitable transforms which achieve "approximate" plant diagonalization. We begin by showing how to search the large collection of orthogonal transforms which are contained in the wavelet packet to find the one which most nearly, or approximately, diagonalizes a given real valued matrix. Wavelet packet transforms admit a natural distributed implementation, making them suitable for use on a control network. We then introduce a class of linear operators called recursive orthogonal transforms (ROTs) which we have developed specifically for the purpose of signal processing on distributed control networks. We show how to use ROTs to approximately diagonalize fixed real and complex matrices as well as transfer function matrices which exhibit a spatial invariance property. Numerical examples of all proposed diagonalization methods are presented and discussed.

APPROXIMATE MATRIX DIAGONALIZATION FOR USE IN DISTRIBUTED CONTROL NETWORKS

by

George A. Kantor

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
1999

Advisory Committee:

Professor P.S. Krishnaprasad, Chairman
Professor Carlos Berenstein
Professor William S. Levine
Professor Steven I. Marcus
Professor André Tits

©Copyright by

George A. Kantor

1999

DEDICATION

To Lori

ACKNOWLEDGEMENTS

There are many people without whom the successful completion of this thesis would have been impossible. I would like to thank my advisor, Professor P.S. Krishnaprasad, for his thoughtful guidance and encouragement. He gave me the freedom to follow my own ideas and helped me find the discipline to investigate them rigorously. I would also like to thank the other members of my committee, Professor Carlos Berenstein, Professor William S. Levine, Professor Steven I. Marcus, and Professor André Tits, for providing many useful comments and suggestions.

I would like to extend a special thanks to Dr. Andrew Newman. In addition to being a truly great friend and source of moral support over the past seven years, Andy has made my life considerably easier by taking care of the logistical details of submitting this thesis.

This research was supported in part by a grant from by the Army Research Office under the ODDR&E MURI97 Program Grant No. DAAG55-97-1-0114 to the Center for Dynamics and Control of Smart Structures (through Harvard University) and also by grants from the National Science Foundation's Engineering Research Centers Program: NSFD CDR 8803012, and by a Learning and Intelligent Systems Initiative Grant CMS9720334.

TABLE OF CONTENTS

List of Tables **vi**

List of Figures **vii**

1 Introduction **1**
1.1 Plant Diagonalization 4
1.1.1 Plant Structure 5
1.1.2 Diagonalizing the Plant Matrix 6
1.2 Preview of the Thesis 9

2 Approximate SVD Using Wavelet Packet Transforms **11**
2.1 Singular Value Decomposition 12
2.2 Wavelets 14
2.2.1 The Fast Wavelet Transform 15
2.2.2 The Wavelet Packet 16
2.2.3 Inverse Transforms 18
2.2.4 Higher Order Filter Banks 20
2.2.5 FWT Implementation on a Control Network 20
2.3 Approximation of SVD Factors with Wavelet Packet Transforms . . 26
2.3.1 The Cost Function 27
2.3.2 Searching the Wavelet Packet 31
2.4 Examples 33
2.4.1 Simplified Flexible Membrane 33
2.4.2 Flexible Beam at Resonance 39

3 Recursive Orthogonal Transforms **42**
3.1 Preliminaries 43
3.1.1 Vector Fields on Lie Groups 43
3.1.2 Matrix Facts 51
3.2 Parallel Signal Processing on Control Networks 52
3.2.1 Implementation on 1–D Array 52
3.2.2 Implementation on 2–D Array 60
3.3 Recursive Orthogonal Transforms 70

4 Symmetric Matrix Diagonalization and SVD using ROT **74**
4.1 Diagonalization of Symmetric Matrices 75
4.1.1 Matrix Diagonalization via Flows on $SO(n)$ 76
4.1.2 Matrix Diagonalizing Flows for ROTs 79
4.2 Singular Value Decomposition 85
4.2.1 SVD via Flows on $U(m) \times U(n)$ 89
4.2.2 SVD Flows for ROTs 92

5 Spatially Invariant Systems **103**
5.1 Infinite Sensor/Actuator Arrays 104
5.2 Periodic Arrays and Circulant Matrices 107
5.3 Diagonalizing Circulant Plants 109
5.3.1 Background: The Discrete Fourier Transform 109
5.3.2 Gradient Flows Diagonalizing Circulant Matrices 110
5.3.3 Example 117

6 Conclusions **121**
6.1 Comparison of Proposed Approximate Diagonalization Methods . . 122
6.1.1 Performance Measures 122
6.1.2 Implementation of Exact SVD Factors 125
6.1.3 Comparison Tables 126
6.2 Suggestions for Future Work 129

A Analysis of ROTs on $SO(4)$ **131**
A.1 The KAK Representation 132
A.2 ROT Representation of $SO(4)$ 134
A.3 Convergence of ROT Equations 136
A.3.1 1–Level ROT 137
A.3.2 2–Level ROT 138
A.4 3–Level ROT 140

B Flexible Beam Model **143**
B.1 Simple Beam Model 143
B.2 Sandwiched Beam Model 150
B.3 Final Beam Model 153
B.3.1 Final Model Statement 156
B.4 Finite Difference Model 158
B.4.1 Finite Difference Model Statement 166
B.5 Simulation Results 167

Bibliography **168**

LIST OF TABLES

6.1 Comparison table for membrane example. 127
6.2 Comparison table for the example of a flexible beam at its 6th resonance. 128

LIST OF FIGURES

2.1 Filter bank implementation of the fast wavelet transform. 16
2.2 Filter bank implementation of the wavelet packet. 17
2.3 Tree representation of wavelet packet. 18
2.4 Filter bank implementation of the inverse fast wavelet transform. . 19
2.5 Filter bank implementation of a wavelet packet transform along with its inverse. 19
2.6 Tree implementation of the Haar wavelet transform. 23
2.7 Connection scheme to realize hierarchy implementation of the Haar wavelet transform using a distributed control network. 24
2.8 Hierarchy implementation of Haar wavelet transform using a distributed control network. 25
2.9 Multihierarchy required to realize hierarchy implementation of a discrete wavelet transform with 4th order filter. 26
2.10 Response of the membrane to the ith input. 34
2.11 Plant matrix for flexible membrane. 35
2.12 Approximately diagonalized membrane plant matrix using a transform from the Haar–Walsh packet. 35
2.13 Diagonalized membrane plant matrix using exact SVD factors. . . . 36
2.14 Tree representation of the wavelet packet basis which approximately diagonalizes the plant matrix. 37
2.15 Graphical depiction of the implementation of the wavelet packet transform shown in Figure 2.14. 38
2.16 Plant matrix for flexible beam at 6th resonance. 40
2.17 Approximately diagonalized plant matrix for flexible beam using a transform from the Haar–Walsh packet. 41
2.18 Diagonalized flexible beam plant matrix using exact SVD factors . . 41

3.1 Graphical description of parallel output processing on a 1–D array of smart sensors. Data flows from the bottom up in the diagram. . . 57
3.2 Required connections for 2–D array. 62
3.3 Communication patterns for first 3 levels of parallel output transformation on a 2–D array of smart sensors. 68

4.1 16 × 16 random symmetric plant matrix. 86
4.2 Approximately diagonalized plant matrix using 3 level ROT. 86
4.3 Approximately diagonalized plant matrix using 7 level ROT. 87
4.4 Approximately diagonalized plant matrix using 11 level ROT. . . . 87
4.5 Approximately diagonalized plant matrix using 15 level ROT. . . . 88
4.6 Plot of approximation error versus ROT level. 88
4.7 Plant matrix for flexible membrane. 98
4.8 Approximately diagonalized membrane plant matrix using 10 level ROT. 99
4.9 Approximately diagonalized membrane plant matrix using 20 level ROT. 99
4.10 Approximately diagonalized membrane plant matrix using 31 level ROT. 100
4.11 Plant matrix for flexible beam at 6th resonance. 100
4.12 Approximately diagonalized flexible beam plant matrix using 10 level ROT. 101
4.13 Approximately diagonalized flexible beam plant matrix using 20 level ROT. 101
4.14 Approximately diagonalized flexible beam plant matrix using 31 level ROT. 102

5.1 Plant response to impulse applied to first actuator. 118
5.2 Impulse energy matrix for untransformed plant. 118
5.3 Impulse energy matrix for plant transformed with level 3 ROT. . . 119
5.4 Impulse energy matrix for plant transformed with level 5 ROT. . . 119
5.5 Impulse energy matrix for plant transformed with level 7 ROT. . . 120

B.1 Rest configuration of beam. 144
B.2 Beam displacement. 145
B.3 Kinematic diagram of differential beam element used to determine strain. 146
B.4 Strain in beam cross section. 148
B.5 Interior differential beam element and the forces acting on it. 149
B.6 Strain in sandwich cross section. 151
B.7 Example configuration of final beam. 154
B.8 Snapshots in time of simulated cantilever beam subject to a 100 volt step input to the leftmost actuator. 169
B.9 Trajectory of tip of the simulated cantilever beam subject to a 100 volt step input to the leftmost actuator 170

Chapter 1

Introduction

Economics often plays a deciding role in the design and implementation of a large control system. In the past, communication was cheap compared to computational power. As a result, centralized control became the norm, with one expensive central computer reading the data directly from all of the sensors and sending control inputs directly to all of the actuators.

Today, however, things are changing. Mass produced microprocessors now provide continually increasing amounts of computational power at continually decreasing prices. In automotive applications, for example, the cost of installing and maintaining the wiring required for a large centralized control system has exceeded the cost of the computational equipment [38]. Additionally, the centralized controller is difficult to reconfigure: adding a new sensor or actuator to the plant requires the installation of a new set of wires connecting the new element to the computer. Hence, it makes sense to investigate alternatives to centralized control.

Distributed control networks have emerged as a viable alternative to centralized control. In such a network, "intelligent" sensors and actuators make control decisions based on a combination of local information and digital commands from the

network. Each node is also capable of sending commands and important sensor information to the other nodes on the network. For example, an "intelligent" motor might receive a position command over the network, then use an on-board trajectory planner, sensor, and PID controller to get to that position. Finally, when the motor has reached its desired position, it would broadcast its new position to the other sensors and actuators on the network.

Control networks have many advantages over centralized control. Less wiring is required to connect the elements of the system together, making the network easier and cheaper to install. The simplicity of the wiring scheme makes maintenance much easier; with less wiring, there is less to go wrong. The on-board "intelligence" of the sensors and actuators allows them to perform self diagnostics, a feature which eases troubleshooting. The system is easily reconfigurable. New sensors and actuators can easily be connected to the existing network. Obsolete nodes can be replaced in the same way.

Much work has been done regarding control networks over the past few years. Brockett has investigated the stabilization of a network of intelligent motors [9]. Wong and Brockett have studied the problems of state estimation and feedback control for control networks with limited communication bandwidth [47] [46]. Hristu [24] has addressed the problem of finding stabilizing feedback laws for linear systems with limited communication. Wang and Mau [41] and Ooi, Verbout, Ludwig, and Wornell [32] present some results on the characteristics of feedback systems where the feedback data is subject to a communication delay. Many authors have investigated the problem of modeling and controlling hybrid systems [19] [5] [6] [7] [8]. Most of this work is motivated by the hybrid dynamics that result from the combination of continuous physical interaction and discrete network

interaction that takes place in a control network.

There is a broad range of applications of control networks that are being put into use today. Automated homes and offices use control networks to automatically turn lights on and off, control temperature, manage security systems, and automate a number of other operations [34]. An control network can provide a manufacturing plant with an assembly line that is easily reconfigured to make different products. Automobile manufacturers employ control networks to coordinate anti-lock braking, engine management, traction control, powered seats and windows, pollution control, and a variety of other systems which reside on a modern car [38]. Oil refineries and chemical plants use control networks with intelligent instrumentation capable of remote calibration and automated troubleshooting, minimizing the control engineer's visits to inhospitable or dangerous areas of the plant [25].

Here, we concentrate on applications of control networks where the number of inputs and outputs is large and the interaction between them is linear. Our interest in this problem is motivated by recent developments in the fields of micro–electro–mechanical systems (MEMS) and smart structures. In the field of MEMS, significant advances have been made in the construction of sensors and actuators on a very small scale. For example, Bifano et.al. [3] have fabricated a MEMS mirror array for optical image processing. The array fits 400 actuators onto an area of less than 1 square centimeter. Since most MEMS devices are currently fabricated in silicon, the idea of designing MEMS sensors and actuators with built–in local microprocessors seems natural. In the field of smart structures, a large number of sensors and actuators are incorporated into the natural structure of a mechanical system. One such application has large number of piezoelectric actuators and strain sensors embedded in a composite panel. The panel could then be a flight

surface on an aircraft and the sensor actuator system would be used to control its shape, limit vibrations, and monitor its health. With such a large number of sensors and actuators involved, the implementation of a controller on a distributed control network seems more feasible than the prospect of a centralized controller. A similar panel could be placed in the interior of an aircraft or automobile and the control network would be used to cancel noise and vibrations coming from outside. Similar approaches have been proposed to damp vibrations and control the flight surface of a helicopter blade. Another application uses "smart" actuators composed of an underwater acoustic actuator and sensor. The SmartPanel™ [17] and Composite Smart Material (CSM) Tile [45] are examples of two such devices that are currently under development. Thousands of these "smart tiles" will be mounted on the outside of the hull of a submarine, covering the entire hull. This massive sensor/actuator array will then be used to actively reduce acoustic radiation, cancel enemy sonar pulses, and perform acoustic sensing.

Clearly, there is a need to systematically address problems of this type. In this thesis, we present some results to aid in the design and implementation of feedback controllers for systems equipped with large distributed control networks. The ideas here are designed to take advantage of the parallel processing capability of the network while reducing the need for global communication.

1.1 Plant Diagonalization

The central theme of this thesis is what we call plant diagonalization. Here we define the structure of the plants we will address. We then introduce the concept of plant diagonalization and discuss the advantages that this idea presents in the context of control networks.

1.1.1 Plant Structure

In this thesis, we will consider plants of the form

$$y = Gu, \tag{1.1}$$

where y is an n-dimensional vector containing the plant outputs, u is an m-dimensional vector of plant inputs, and G is the $n \times m$ matrix which defines the interaction between the inputs and outputs. We call G the "plant matrix".

In most of this thesis, we will consider plant matrices whose elements are constant real or complex numbers. These types of matrices can be used to model two types of systems: "quasi-static" systems and "fixed frequency" systems. The term "quasi-static" is used to describe stable systems whose transients are negligible compared to the DC response of the system. The plant matrix of a quasi-static system is a constant real–valued matrix. A "fixed frequency" system is linear dynamic system whose inputs (and any disturbances) are assumed to be sinusoids of a constant frequency. The plant matrix of a fixed frequency system is the complex–valued matrix that results from evaluating the transfer function matrix of the dynamical system at the input frequency. The (i, j)th element of the plant matrix of a fixed frequency system is a complex number which represents the gain and phase shift from the jth input to the ith output. Models of this type are useful for systems which exhibit strong resonances as will be discussed in Section 2.4.2.

In addition to this basic structure, we assume that the inputs and outputs correspond to actuators and sensors spatially distributed on a control network. The network is composed of a number of nodes. Each node contains a combination of sensors and actuators along with a microprocessor and the communication hardware required to communicate with other nodes on the network. Each node has access to its own data; it can read the outputs of the sensors it contains and it can

apply inputs to the actuators it contains. It shares data with other nodes only by communicating via the network.

1.1.2 Diagonalizing the Plant Matrix

The goal of this thesis is to find an efficient method of implementing output feedback control so that the closed loop system achieves some desired performance criteria. Our approach to solving this problem is to diagonalize the plant matrix G. To do this, we search for invertible basis transformations for the input and output vectors so that, in the new bases, the transformed plant matrix appears diagonal. Here, we call a matrix Σ diagonal if $[\Sigma]_{ij} = 0$ whenever $i \neq j$. Using this terminology, it makes sense to call a non–square matrix diagonal.

Consider the basis transformations $\widetilde{y} = Q_1 y$ and $\widetilde{u} = Q_2 u$. In these new bases, the input/output behavior of the plant is described by the equation

$$\widetilde{y} = Q_1 G Q_2^{-1} \widetilde{u}. \tag{1.2}$$

Hence, the problem of finding suitable basis transformations is equivalent to finding Q_1 and Q_2 such that the matrix

$$\Sigma \triangleq Q_1 G Q_2^{-1} \tag{1.3}$$

is diagonal.

Suppose it is possible to find suitable Q_1 and Q_2. The system in the new coordinates can be written

$$\widetilde{y} = \Sigma \widetilde{u}. \tag{1.4}$$

Since Σ is diagonal, the interaction between $\widetilde{u}_i$ and $\widetilde{y}_j$ is zero for $i \neq j$. The plant has become a system of decoupled scalar subsystems,

$$\widetilde{y}_i = \sigma_{ii} \widetilde{u}_i, \tag{1.5}$$

for $i = 1, 2, \ldots, \min(n, m)$.

This representation has two major advantages, particularly when n and m are large. First, the problem of MIMO controller synthesis is transformed to the much easier problem of synthesizing a controllers for a number of decoupled SISO systems. The other advantage is closely related. Since the systems are decoupled, the feedback laws are also decoupled. Each $\widetilde{u}_i$ can be calculated based only on the value of the corresponding $\widetilde{y}_i$. As a result, once $\widetilde{y}$ is computed, $\widetilde{u}$ can be computed quickly, in parallel, and with no additional communication.

To illustrate this, we compare the task of computing the linear feedback law $u = Ky$ in the original basis with the task of computing $\widetilde{u} = \widetilde{K}\widetilde{y}$ in the transformed basis. In the original basis, u is computed by multiplying the n–vector y by the $m \times n$ matrix K. This calculation requires $\mathcal{O}(nm)$ ("on the order of nm") operations. Additionally, global communication is required; in order to compute u_i, the value of every element of y must be known. In the transformed basis, however, the matrix $\widetilde{K}$ is diagonal, so $\widetilde{u}$ can be computed in $\mathcal{O}(\min(m, n))$ operations. Due to the decentralized nature of $\widetilde{K}$, no additional communication is required to make the calculations and the calculations can be easily performed in parallel on the control network.

The computational and communication requirements of the transforms Q_1 and Q_2 are very important. In addition to the cost involved in computing $\widetilde{u}$ from $\widetilde{y}$ we must also perform the coordinate transformations $\widetilde{y} = Q_1 y$ and $u = Q_2^{-1}\widetilde{u}$. If these transforms are computationally intensive or require global communication, then the advantages of the transformed basis are lost. In most cases, this means that we will have to settle for transforms which "approximately" diagonalize the plant matrix.

In this thesis, we will implement the approximations of Q_1 and Q_2 by using a series of alternating communication and computation steps. In the communication steps, a limited amount of data is shared between selected nodes. In the computation steps, the nodes individually perform a simple manipulation of the data they have access to. By their design, these approximations take advantage of the distributed computing power available on the network while respecting the communication constraints.

Distributed Controller Implementation

Now we are ready to describe how plant diagonalization can be applied to the implementation of feedback on a distributed control network. In the proposed implementation, the controller performs three basic tasks. First, it computes $\widetilde{y} = Q_1 y$ using a series of alternating communication and computation steps as described above. The elements of the output vector y are the measurements of the sensors that are distributed spatially over the network. Because of of the distributed nature of our implementation of the transform Q_1, the elements of the resulting transformed vector $\widetilde{y}$ are also distributed over the nodes of the network. The second task of the controller is to compute the transformed output vector $\widetilde{u}$ based on $\widetilde{y}$. The plant matrix is diagonal (or "approximately diagonal") in the transformed coordinates, so each $\widetilde{u}_i$ can be chosen based soley on the corresponding $\widetilde{y}_i$. This means that each $\widetilde{u}_i$ can be computed on the node on which $\widetilde{y}_i$ resides without any additional communication. The third and final task of the controller implementation is to transfrom the vector $\widetilde{u}$ into the actual input vector u using the transformation $u = Q_2^{-1}\widetilde{u}$. Like Q_1, Q_2^{-1} is implemented in a distributed manner using a series of communication and computation steps. After the third task is

completed, each element u_i of the input vector will reside on the node containing the actuator to which u_i will be applied.

To summarize, the main goal of this thesis is to find linear coordinate transformations Q_1 and Q_2 which meet the following criteria:

1. The matrix $Q_1 G Q_2^{-1}$ is diagonal or "approximately" diagonal

2. The transforms Q_1 and Q_2 can be implemented as a series of alternating computation and communication steps. The computations necessary to perform the transform can be easily distributed over the processors on the control network, taking advantage of the network's parallel processing capability. The communication steps can be chosen to respect the spatial distribution of the data, reducing communication overhead.

1.2 Preview of the Thesis

In the following pages, we present some methods of finding suitable transforms to diagonalize or approximately diagonalize various types of plants. In Chapter 2, we show how to approximately diagonalize a constant real or complex valued matrix using transforms from the wavelet packet. The idea is inspired by the work of Chou, Guthart, and Flamm [14], who proposed the use of the wavelet transform in the spatial domain to simplify the required feedback for vibrating systems. The work presented here uses an algorithm due to Wickerhauser [43][44] to find wavelet packet transforms which approximate the SVD factors of the plant matrix. We show that the resulting transforms have a natural implementation on a hierarchical control network.

In Chapter 3, we introduce a class of transforms which we have designed specif-

ically to be implemented on a control network. We begin by developing signal processing schemes to be implemented on networks whose nodes are sensor/actuator pairs. We then generalize these ideas to create a class of transforms suitable for general control networks. We call these transforms recursive orthogonal transforms, or ROTs. Recursive orthogonal transforms are the subject of the rest of the thesis.

In Chapter 4 we consider the problem of finding an ROT which approximately diagonalizes a fixed real or complex valued plant matrix. We first consider fixed plant matrices which are real and symmetric. The approach we take is very similar to Brockett's work on diagonalizing symmetric matrices via gradient flows on orthogonal matrices [10]. We then consider the problem of diagonalizing general complex matrices. Here we extend the work of Helmke and Moore [21] to find ROTs which most closely approximate the complex SVD factors of the plant matrix.

In Chapter 5, plant matrices which exhibit a spatial invariance property are considered. This spatial invariance property is analogous to the time invariance of LTI systems. Spatial invariance can be exploited to aid in the design and implementation of feedback controllers. Our work here is motivated by a number of authors, including El-Sayed and Krishnaprasad [16] and, more recently, Bamieh [1]. We show that all spatially invariant systems are exactly diagonalized by the discrete Fourier transform. Armed with this knowledge, we seek to find an ROT capable of diagonalizing any spatially invariant plant matrix.

Chapter 6 contains a quantitative comparison of the approximate matrix diagonalization methods presented in this thesis. We also make some concluding remarks and give some suggestions for future work.

Chapter 2

Approximate SVD Using Wavelet Packet Transforms

In this chapter, we approach the problem of plant diagonalization by seeking wavelet packet transforms which approximate the SVD factors of the plant matrix. We consider systems whose plant matrix G is a fixed, real valued matrix. We also assume that G is square and has full rank. Such a model is suitable for "quasi-static" linear systems, i.e. stable linear systems whose transient response can be neglected.

The idea of using wavelet transforms in the context of feedback for large control networks comes from the work of Chou, Guthart, and Flamm [14]. These authors have shown that for a special class of linear systems, the wavelet transform can be used to transform the input and output vectors into bases which allow for the efficient computation of a feedback control law. Further, the multiresolution property of the wavelet transform leads to a natural implementation on a hierarchical control network. Motivated by this work, we seek to use transforms from the wavelet packet as the basis transformations to achieve plant diagonalization.

We begin the discussion with the naive suggestion that the matrix factorization technique of singular value decomposition can be used for plant diagonalization. The SVD factors accomplish the task of diagonalization, but, as we will show, they are not feasible to implement on a large control network because they are computationally intensive and centralized.

We then move on to introduce some of the basic concepts of wavelet theory with an eye toward using wavelet transforms for plant diagonalization. We concentrate on the implementation of transforms from the wavelet packet and show that these transforms are computationally efficient and have a natural parallel implementation on a control network.

Next, we show how to search the wavelet packet to find the transforms which most closely diagonalize the plant matrix. These transforms are approximations of the SVD factors. An algorithm to search the wavelet packet is borrowed from Wickerhauser [43]. Motivated by this work, we develop a cost function suitable for the purpose of matrix diagonalization. This cost function is then used to search the wavelet packet for the SVD approximations.

Finally, we demonstrate this idea with two examples. In the first example, wavelet packet transforms are used to approximately diagonalize the plant matrix of a system composed of a flexible membrane driven by a linear array of electrostatic actuators. The second example looks at the approximate diagonalization of the plant matrix of a flexible beam driven at resonance.

2.1 Singular Value Decomposition

We wish to find basis transformations which diagonalize the real valued plant matrix, G. This can be accomplished using the well known matrix factorization

technique of singular value decomposition (SVD). Here, we briefly discuss SVD and demonstrate how it can be used in the context of our problem. A complete description can be found in Strang [35].

Consider the real valued $n \times n$ matrix G. SVD states that G can be factored such that

$$G = Q_1 \Sigma Q_2^T, \tag{2.1}$$

where

1. The columns of $Q_1 \in \mathbb{R}^{n \times n}$ are the eigenvectors of GG^T.

2. The columns of $Q_2 \in \mathbb{R}^{n \times n}$ are the eigenvectors of $G^T G$.

3. The $n \times n$ matrix Σ is diagonal, and values on the diagonal are the square roots of the eigenvalues of GG^T and $G^T G$.

The matrices Q_1 and Q_2 are orthogonal, i.e. $Q_1 Q_1^T = Q_2 Q_2^T = \mathbb{1}_n$, where $\mathbb{1}_n$ is the $n \times n$ identity matrix. The set containing all $n \times n$ orthogonal matrices is denoted $O(n)$. Multiplying G by Q_1^T on the left and Q_2 on the right yields

$$Q_1^T G Q_2 = \Sigma. \tag{2.2}$$

In the present context, G is a plant matrix of an n-input, n-output system. In other words,

$$y = Gu, \tag{2.3}$$

where u is the n–dimensional input vector and y is the n–dimensional output vector.

Consider the change of variables $\widetilde{y} = Q_1^T y$ and $\widetilde{u} = Q_2^T u$. Substitution into the plant equation (Equation 1.1) yields

$$\widetilde{y} \;=\; Q_1^T P Q_2 \widetilde{u}$$

$$= \Sigma\widetilde{u}. \tag{2.4}$$

Hence, the ith element of the transformed input vector $\widetilde{u}$ affects only the ith element of the transformed output vector $\widetilde{y}$ for $i = 1, 2, \ldots, n$. As a result, the entire system can be controlled using local feedback, i.e. each input is determined based solely on the value of its corresponding output. In other words, the SVD factors achieve the stated goal of plant diagonalization.

Unfortunately, the transforms generated by SVD are computationally intensive to implement; for an n dimensional output vector y, it takes $\mathcal{O}(n^2)$ operations to compute $\widetilde{y}$. Additionally, the value of every element of y is required to compute each element of $\widetilde{y}$. This fact renders the decentralized nature of the resulting basis meaningless. Fortunately, we can overcome these implementation issues by using wavelet packet transforms to approximate the SVD factors. This is the subject of the remaining part of this chapter. In the next section, we lay the groundwork for this idea by reviewing some of the principles of wavelets and the wavelet packet.

2.2 Wavelets

Wavelets have been the subject of much work in recent years [36] [15]. A wavelet is a localized function which, together with its dilations and translations, forms a basis for a specified function space. In other words, if $\phi(t)$ is a suitable "mother" wavelet for a given space, then any function $f(t)$ belonging to that space can be written as

$$f(t) = \sum_{j,k\in\mathbb{Z}} a_{jk}\phi(2^j t - k) \tag{2.5}$$

The associated wavelet transform is the map which takes a function from the original space and returns the coefficients of the wavelet expansion, $a_{jk}, j, k \in \mathbb{Z}$.

The wavelet transform simply gives us another way to represent the information contained in the function $f(t)$.

Wavelets have a large number of characteristics and applications. Below we focus on a few that we will use. Since we will be using the wavelet transform on discrete signals of finite length (vectors), the following discussion is limited to that case.

2.2.1 The Fast Wavelet Transform

One of the most important features of wavelets is that the wavelet transform is extremely easy and fast to implement for discrete spaces. In fact, for discrete signals, the fast wavelet transform (FWT) can is implemented as an iterated bank of two low order digital filters combined with a down sampling step. The FWT is briefly described here. A complete discussion of wavelets and the connections between wavelets and filter banks can be found in Strang and Nguyen [36].

The Haar wavelet transform can be implemented as an iterated bank of digital filters as shown in Figure 2.1. This filter bank is called the "analysis bank". The the lowpass filter C and the highpass filter D are written in z-transform form as

$$C(z) = \frac{1}{\sqrt{2}} + \frac{1}{\sqrt{2}} z^{-1} \tag{2.6}$$

$$D(z) = \frac{1}{\sqrt{2}} - \frac{1}{\sqrt{2}} z^{-1}. \tag{2.7}$$

The output of each filter is fed into a downsampling step, denoted in Figure 2.1 as $\downarrow 2$. The output of the downsampler is the input vector with every other element removed.

The output of the highpass channel of the filter bank is kept; these numbers are part of the wavelet transform. The output of the lowpass channel is fed into

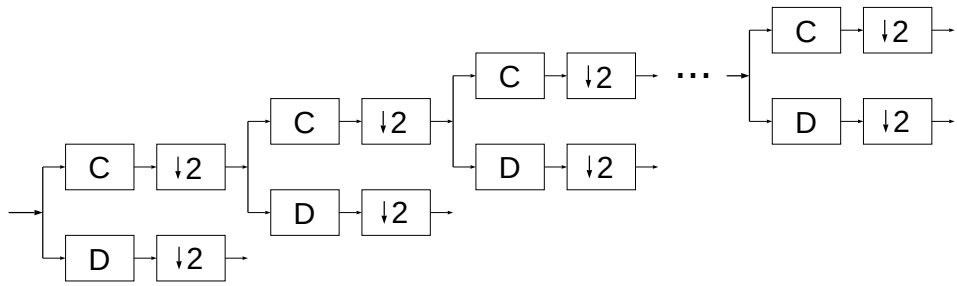


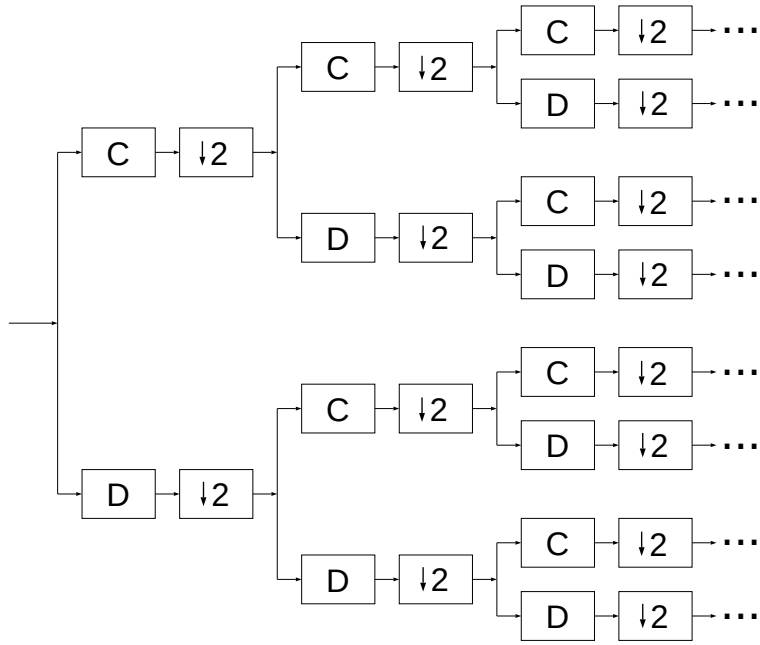
Figure 2.1: Filter bank implementation of the fast wavelet transform.

another identical filter bank and the process is iterated until the desired resolution is reached. In the case of a vector of dimension 2^n, n iterations of the filter bank completely transform the vector; the output of each rightmost highpass and lowpass channel is a single real number so that no further filtering is possible.

2.2.2 The Wavelet Packet

The wavelet packet is a collection of orthogonal transforms which can be implemented using filter banks. Where the fast wavelet transform is realized by repeatedly passing the output of the lowpass filter into another filter bank, the entire wavelet packet is realized by repeatedly passing the outputs of both channels to the next filter bank. This filtering process is represented in Figure 2.2. The packet formed using the filter bank from the Haar wavelet transform is called the Haar-Walsh Packet. For a vector of length n (an even power of two), this filtering process is iterated $\log_2(n)$ times so that the output of each filter at the end of the tree (the right of the tree as depicted in Figure 2.2) is a single real valued number.

This tree of filters contains a large number of orthonormal representations of the input vector. For example, the coefficients taken from the n filters at the end of the tree provide a representation of the input in a basis analogous to the Short Time Fourier Transform [36]. Additional bases can be found by "pruning" the tree,


C
↓2
D
↓2
C
↓2
D
↓2
C
↓2
D
↓2

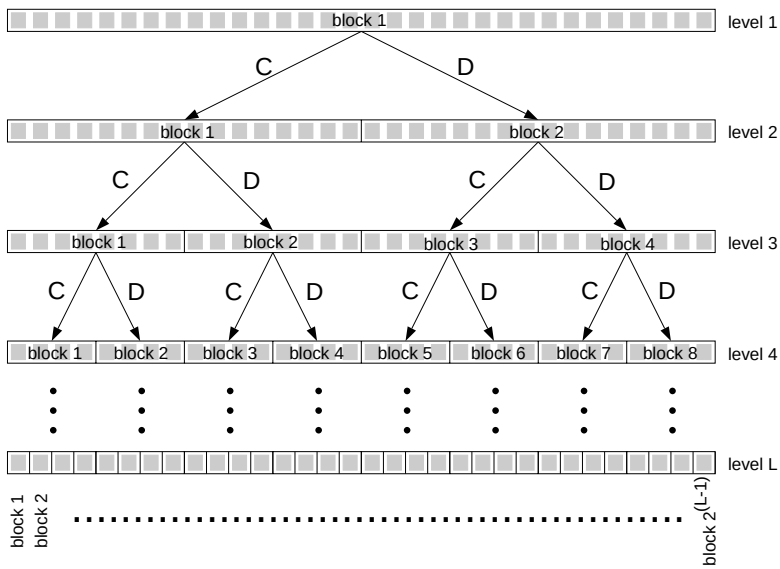


Figure 2.3: Tree representation of wavelet packet.

The entire packet can be computed in $\mathcal{O}(n\log(n))$ operations, so it is feasible to create such a library even for large n. Further, the resulting library is arranged in a tree form. As we will see, this allows the library to be searched using fast recursive algorithms.

2.2.3 Inverse Transforms

The FWT has an inverse which can also be implemented as an iterated bank of filters, as shown in Figure 2.4. Since this filter bank reconstructs the original signal it is called the "synthesis bank". Here, the inputs are upsampled before they are passed into the filters. Upsampling, denoted by $\uparrow 2$, inserts a zero between every element of the input vector. For the case of the Haar wavelet, the filters C^* and D^* are written in z-transform form as

$$C^*(z) = \frac{1}{\sqrt{2}} + \frac{1}{\sqrt{2}} z^{-1} \tag{2.8}$$

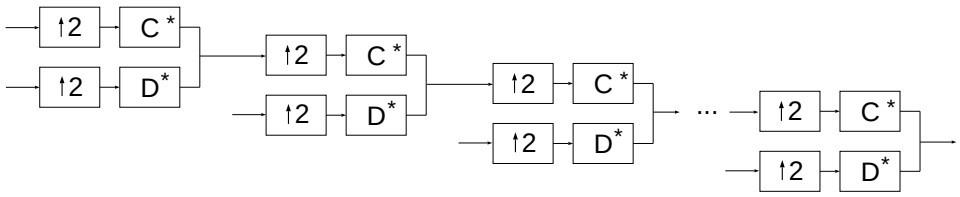


Figure 2.4: Filter bank implementation of the inverse fast wavelet transform.

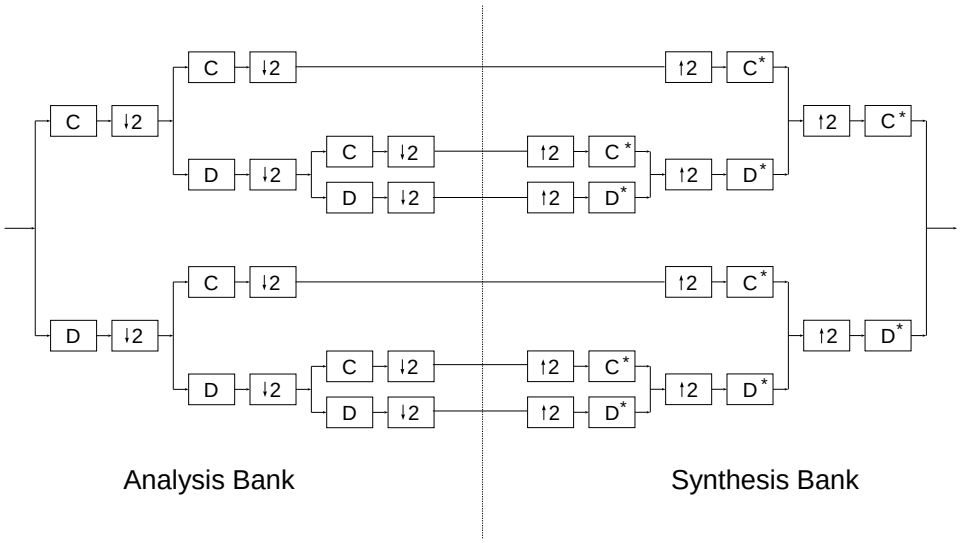


Figure 2.5: Filter bank implementation of a wavelet packet transform along with its inverse.

$$D^*(z) \quad = \quad -\frac{1}{\sqrt{2}} + \frac{1}{\sqrt{2}} z^{-1}.. \tag{2.9}$$

A similar synthesis bank can be used to invert any transform from the wavelet packet. In this case, the synthesis filter bank must be a mirror image of the analysis filter bank, with C, D, and $\downarrow 2$ replaced by C^*, D^*, and $\uparrow 2$, respectively. An example of a wavelet transform and its inverse in filter bank implementation is shown in Figure 2.5.

2.2.4 Higher Order Filter Banks

The previous three sections show the filter bank implementation of the FWT, wavelet packet transform (WPT), and their inverses for the Haar wavelet. There are many other wavelets which can be implemented using the same filter bank structure with wisely chosen FIR filters C, D, C^* and D^*. Clearly, the filters must be chosen so that the synthesis bank actually inverts the analysis bank. This feature is sometimes referred to as "perfect reconstruction". Other conditions, such as orthogonality, can also be imposed on the filters to yield good wavelet properties.

An important set of suitable filters has been introduced by Daubechies [15]. In addition to perfect reconstruction, Daubechies' filters are orthogonal and satisfy the "maximum flatness" condition, which means that the frequency responses of the filters are as flat as possible at $\omega = 0$ and $\omega = \pi$. Other choices of filters which yield good wavelets are discussed by Strang and Nguyen [36].

2.2.5 FWT Implementation on a Control Network

The FWT or any other transform from the wavelet packet can be implemented very efficiently on a distributed control network. The key to this is to exploit the recursive nature of these transforms. At each filter bank iteration, only local information is used. This local information is processed (filtered) and the important information is passed on. This process is repeated until the transform is complete. We take advantage the locality of the information at any step in the transform to distribute the necessary calculations over a control network connected to form a hierarchy. We first discuss the implementation of the Haar-Walsh wavelet packet. Then we discuss a possible extension to include more general wavelet packets.

These ideas will be made more specific in an example in Section 2.4.

The Haar Wavelet

Here, we show how to use a hierarchical control network can be used to compute the FWT for a vector of length 8 using the Haar wavelet. Once the transform is understood for an 8-vector, extension to other vector lengths is trivial. Extensions to other bases in the Haar-Walsh packet will be briefly discussed at the end of this section and illustrated by the example in Section 2.4.

As we have already discussed, the FWT using the Haar wavelet can be implemented using the filter bank in Figure 2.1 where the filters C and D are given by Equations 2.6 and 2.7. After downsampling, the output of the lowpass part of the first filter bank is

$$Cx = \frac{1}{\sqrt{2}} \begin{bmatrix} x(2) + x(1) \\ x(4) + x(3) \\ x(6) + x(5) \\ x(8) + x(7) \end{bmatrix}. \tag{2.10}$$

The output the first highpass filter after down sampling is

$$Dx = \frac{1}{\sqrt{2}} \begin{bmatrix} x(2) - x(1) \\ x(4) - x(3) \\ x(6) - x(5) \\ x(8) - x(7) \end{bmatrix}. \tag{2.11}$$

Note that each element of Cx and Dx rely on adjacent elements of x. The same is true if we look at the outputs of the next filter bank:

$$C(Cx) = \frac{1}{\sqrt{2}} \begin{bmatrix} (Cx)(2) + (Cx)(1) \\ (Cx)(4) + (Cx)(3) \end{bmatrix} \tag{2.12}$$

and

$$D(Cx) = \frac{1}{\sqrt{2}} \begin{bmatrix} (Cx)(2) - (Cx)(1) \\ (Cx)(4) - (Cx)(3) \end{bmatrix}. \tag{2.13}$$

Likewise, the outputs of the final filter bank are

$$C(CCx) = \frac{1}{\sqrt{2}} \left[(CCx)(2) + (CCx)(1) \right] \tag{2.14}$$

and

$$D(CCx) = \frac{1}{\sqrt{2}} \left[(CCx)(2) - (CCx)(1) \right]. \tag{2.15}$$

So the computation of C and D rely on adjacent elements of the previous filter bank's output at every level. This is the locality we will take advantage of with a hierarchy.

Consider the tree depicted in Figure 2.6. The nodes at the bottom level contain the elements of the input vector x. This information is passed up to the next level where the individual elements of C and D are calculated separately. The elements of Cx are then passed to the next level. The process is then repeated until we reach the top of the tree. The transform is then given by the coefficients in Dx, DCx, $DCCx$, and $CCC(x)$.

The inverse FWT can be implemented on a hierarchy by using the same idea, only in reverse. Here, we start at the top of the tree and work down, using the filters C^* and D^* instead of C and D.

Implementing the FWT and its inverse on this hierarchy has two big advantages. First, communication is kept to a minimum since each node receives only the information it needs to compute its part of the transform. Also, the connection scheme required is simple: each node needs to communicate with at most three other nodes (its parent and its two children). Second, a large number of the necessary computations are done in parallel, at the same time, on different processors.

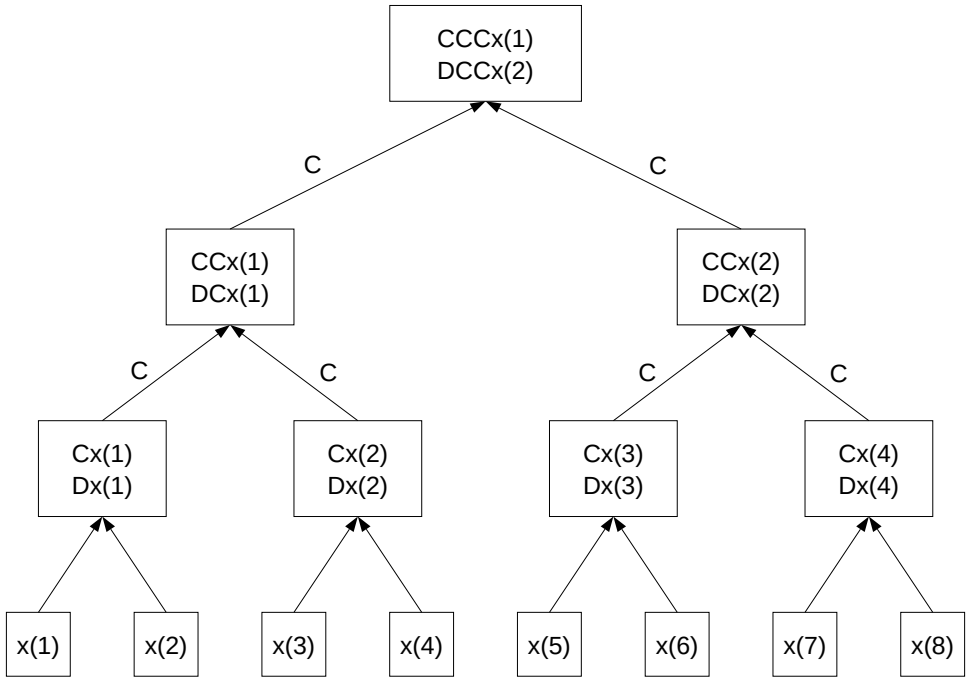


Figure 2.6: Tree implementation of the Haar wavelet transform.

For example, it takes a total of 12 operations to compute Cx (8 multiplications, 4 additions). Assuming each of these calculations takes one cycle to complete, this means it would take a single processor 12 cycles to compute Cx. Consider instead the hierarchy approach. It takes only 3 operations to compute 1 element of Cx. Since each element of Cx is calculated on its own processor, all of the elements can be computed simultaneously. Hence, in the hierarchy implementation, it takes only 3 cycles to compute all of Cx.

For these reasons, we propose to implement the FWT on a control network connected to form a hierarchy. Recall that each network node has its own processor, so massive parallel processing capability exists. The data from a given node's onboard sensor, at any point in time, can be thought of as one element of the input vector, x. Our goal is to create a control network which can emulate the hierarchy discussed above.

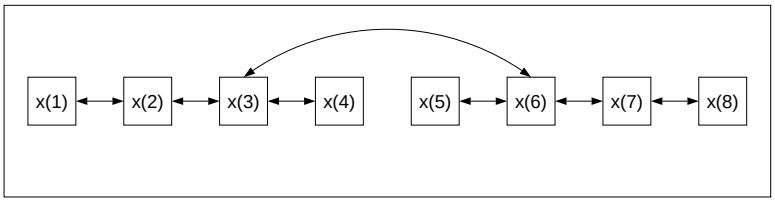


Figure 2.7: Connection scheme to realize hierarchy implementation of the Haar wavelet transform using a distributed control network.

One such possibility is shown in Figures 2.7 and 2.8. Figure 2.7 depicts a connection scheme for a linear array of eight nodes. This array could be, for example, an array of intelligent strain sensors mounted along a flexible beam. Figure 2.8 shows how this connection scheme can be used to implement the FWT. This "hierarchy" uses a limited amount of communication and takes advantage of its parallel processing abilities in computing the FWT.

A similar process can be used to implement any other basis transformation in the Haar-Walsh packet. In this case, the computations are made as in the wavelet transform case: each node can compute its elements of the filtered vector based on information from its two children. The difference lies in the communication. Instead of just passing the output of the lowpass filter up to the next level, it may be necessary to pass the output of the highpass filter or the outputs of both filters.

Higher Order Wavelets

In the previous section we showed how to efficiently implement any transform in the Haar-Walsh packet on a hierarchical control network. Since the Haar filters look at two adjacent elements of the input vector at a time and the filter is followed by a down sampling of two, a simple dyadic tree was the natural choice for the network topology. For more general wavelets, such as the family of wavelets proposed

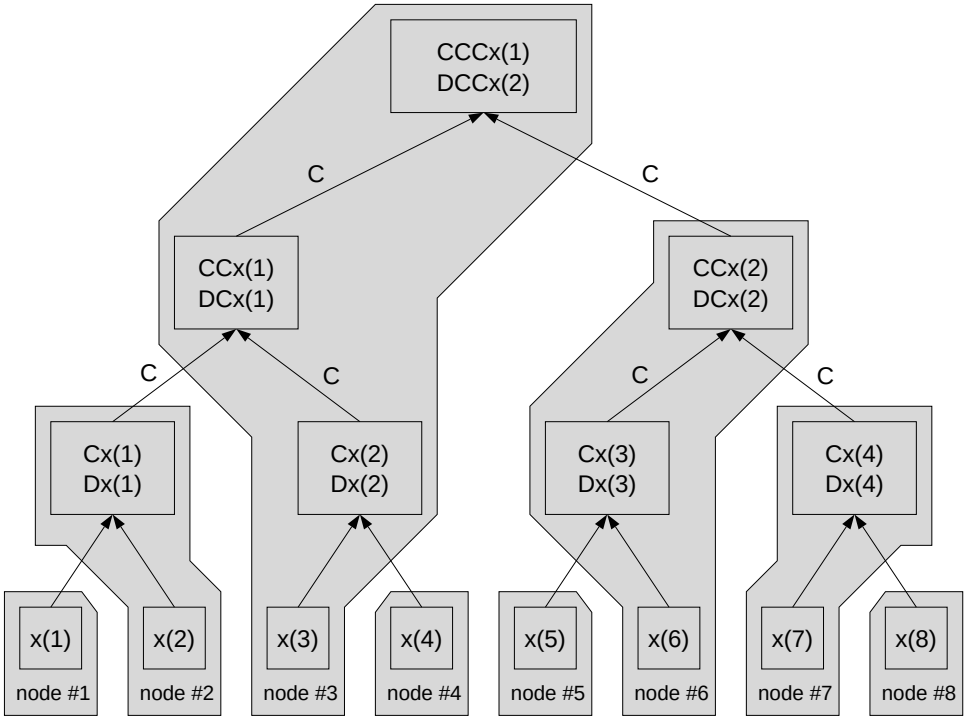


Figure 2.8: Hierarchy implementation of Haar wavelet transform using a distributed control network.

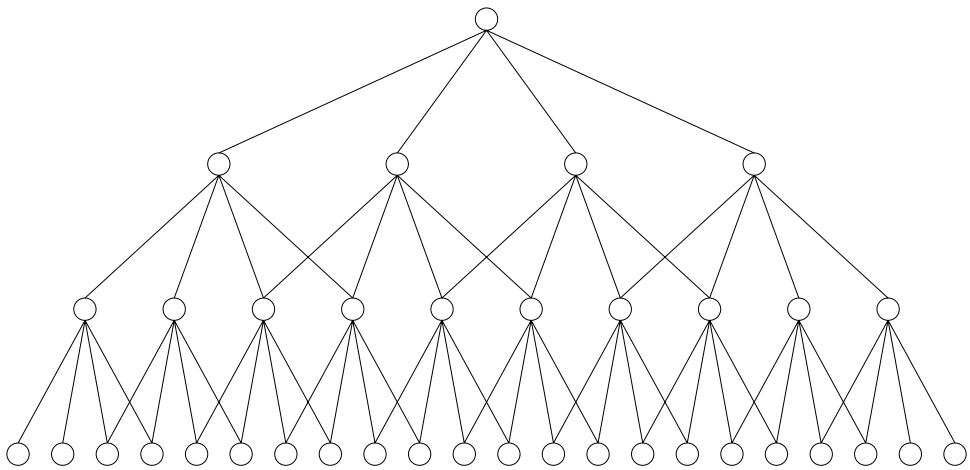


Figure 2.9: Multihierarchy required to realize hierarchy implementation of a discrete wavelet transform with 4th order filter.

by Daubechies [15], the filters look at n adjacent elements of the input vector. The resulting output is then downsampled by a factor of two. This leads to a "multihierarchy" topology. Figure 2.9 shows the structure of such a hierarchy for a wavelet filter with 4 coefficients.

2.3 Approximation of SVD Factors with Wavelet Packet Transforms

In the previous two sections we have laid the groundwork for the main result of this chapter. In Section 2.1 we showed that the SVD factors of the plant matrix can be used to diagonalize the plant, but they are not feasible to implement on a large control network. In Section 2.2 we showed that wavelet packet transforms could be efficiently implemented on a control network, but we gave no indication of how they could be used to diagonalize a plant matrix. Here, we combine these two ideas. The approach is straightforward. The wavelet packet contains a large

number of orthogonal transforms. Each SVD factor is also orthogonal. To find an "approximate" SVD transform, we simply search the wavelet packet to find the transform which is, in some sense, closest to the SVD factor.

This idea is borrowed from Wickerhauser [43], who proposed this approach to perform principal component analysis on a set of random data vectors. Using the fact that the Karhunen–Loève basis is the minimum entropy basis of a given data set, Wickerhauser was able to devise a cost function which allowed for an efficient, recursive search of the wavelet packet. A similar approach can be used to approximate the SVD factors of the plant matrix.

2.3.1 The Cost Function

We assume that the plant matrix $G \in \mathbb{R}^{n\times n}$ is square and full rank. Finding a suitable SVD factor Q_1^T is equivalent to finding the $\Theta \in O(n)$ such that Θ^T diagonalizes the symmetric matrix GG^T. Towards this goal, we develop a cost function $J_1(\Theta)$ which is globally minimized when the matrix $\Theta GG^T\Theta^T$ is diagonal.

We begin by writing

$$G = [g_1^c, g_2^c, \cdots, g_n^c]\,, \tag{2.16}$$

where $g_1^c, g_2^c, \ldots, g_n^c$ are the columns of G. Now define

$$X_\Theta \;\triangleq\; \Theta G = [\Theta g_1^c, \Theta g_2^c, \cdots, \Theta g_n^c]\,. \tag{2.17}$$

Let

$$M_\Theta \;\triangleq\; \Theta GG^T\Theta^T = X_\Theta X_\Theta^T. \tag{2.18}$$

The ith diagonal element of M_Θ is then given by

$$[M_\Theta]_{ii} \;=\; \sum_{j=1}^{n} [X_\Theta]_{ij}[X_\Theta^T]_{ji}$$

$$= \sum_{j=1}^{n} \left(\Theta g_j^c\right)_i^2 . \tag{2.19}$$

Since both G and Θ are nonsingular, each $[M_\Theta]_{ii}$ is strictly positive. We now define a cost function on $O(n)$ by taking the product of the diagonal elements of M_Θ:

$$\begin{aligned} J_1(\Theta) &\triangleq \prod_{i=1}^{n} [M_\Theta]_{ii} \\ &= \prod_{i=1}^{n} \sum_{j=1}^{n} \left(\Theta g_j^c\right)_i^2 . \end{aligned} \tag{2.20}$$

We justify that $J_1(\Theta)$ is a suitable cost function to be minimized with the following claim:

Claim 2.3.1 *Let the matrix $Q_1 \in O(n)$ be a left SVD factor of the full rank matrix $G \in \mathbb{R}^{n\times n}$, i.e. Q_1 diagonalizes GG^T. Then*

$$J_1(Q_1^T) \leq J_1(\Theta) \tag{2.21}$$

for all $\Theta \in O(n)$. Furthermore, $J_1(Q_1^T) = J_1(\Theta)$ if and only if Θ^T diagonalizes GG^T.

In order to prove this claim, we require some matrix facts. We state these facts without proof [35].

Fact 2.3.1

$$det\left(\begin{bmatrix} A & B \\ C & D \end{bmatrix}\right) = det(A)\,det(D - CA^{-1}B), \tag{2.22}$$

where A is a square, invertible matrix, D is a square matrix, B and C are matrices with dimensions compatible to A and D.

Fact 2.3.2 *A symmetric A is positive definite if and only if every square submatrix in the upper left corner of A has positive determinant.*

Now we are ready to prove the claim.

<u>Proof of Claim:</u>

First, we derive an expression for $J_1(Q_1^T)$. Note that

$$\begin{aligned} M_{Q_1^T} &= Q_1^T G G^T Q_1 \\ &= \begin{bmatrix} \sigma_1^2 & 0 & \cdots & 0 \\ 0 & \sigma_2^2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma_n^2 \end{bmatrix}, \end{aligned} \tag{2.23}$$

where $\sigma_1, \sigma_2, \ldots, \sigma_n$ are the singular values of G. Since G is full rank, all of its singular values are strictly positive. Now

$$\begin{aligned} J_1(Q_1^T) &= \prod_{i=1}^{n} \left[M_{Q_1^T} \right]_{ii} \\ &= \prod_{i=1}^{n} \sigma_i^2 \\ &= \det(GG^T). \end{aligned} \tag{2.24}$$

Note that $\det(GG^T) = \det(\Theta G G^T \Theta^T) = \det(M_\Theta)$ for any $\Theta \in O(n)$. Hence, in order to show that $J_1(Q_1^T) \leq J_1(\Theta)$ it suffices to show that

$$\det(M_\Theta) \leq \prod_{i=1}^{n} [M_\Theta]_{ii}. \tag{2.25}$$

Define $\Gamma_n \triangleq M_\Theta$. Given $\Gamma_{k+1} \in \mathbb{R}^{k+1 \times k+1}$, define $\Gamma_k \in \mathbb{R}^{k \times k}$, $b_{k+1} \in \mathbb{R}^k$ and $\gamma_{k+1} \in \mathbb{R}$ such that

$$\Gamma_{k+1} = \begin{bmatrix} \Gamma_k & b_{k+1} \\ b_{k+1}^T & \gamma_{k+1} \end{bmatrix}, \tag{2.26}$$

for $k = n-1, n-2, \ldots, 2, 1$. Note that $\Gamma_1 \in \mathbb{R}$. Let $\gamma_1 = \Gamma_1$. Note that $\gamma_k = [M_\Theta]_{kk}$ and each $\gamma_k > 0$. Since $M_\Theta > 0$, Fact 2.3.2 tells us that $\Gamma_k > 0$ for

each k. Fact 2.3.1 then gives the following expression for $\det(\Gamma_k)$:

$$\begin{aligned}\det(\Gamma_k) &= \det(\Gamma_{k-1})\det(\gamma_k - b_k^T\Gamma_{k-1}^{-1}b_k) \\ &= \det(\Gamma_{k-1})(\gamma_k - b_k^T\Gamma_{k-1}^{-1}b_k), \end{aligned} \tag{2.27}$$

$k = 2, 3, \ldots, n$. We can use this equation to recursively find the following expression for $\det(M_\Theta)$:

$$\begin{aligned}\det(M_\Theta) &= \det(\Gamma_n) \\ &= \det(\Gamma_{n-1})(\gamma_n - b_n^T\Gamma_{n-1}^{-1}b_n) \\ &= \det(\Gamma_{n-2})(\gamma_{n-1} - b_{n-1}^T\Gamma_{n-2}^{-1}b_{n-1})(\gamma_n - b_n^T\Gamma_{n-1}^{-1}b_n) \\ &\vdots \\ &= \det(\Gamma_1)\prod_{k=2}^{n}(\gamma_k - b_k^T\Gamma_{k-1}^{-1}b_k) \\ &= \gamma_1\prod_{k=2}^{n}(\gamma_k - b_k^T\Gamma_{k-1}^{-1}b_k). \end{aligned} \tag{2.28}$$

Note that each $b_k^T\Gamma_{k-1}^{-1}b_k \geq 0$ since $\Gamma_{k-1} > 0 \Rightarrow \Gamma_{k-1}^{-1} > 0$. Using Equation 2.27 and the fact that $\det(\Gamma_k) > 0$ for each k, we see that $(\gamma_k - b_k^T\Gamma_{k-1}^{-1}b_k) > 0$ for each $k = 2, 3, \ldots, n$. This gives the inequality

$$\gamma_1\prod_{k=2}^{n}(\gamma_k - b_k^T\Gamma_{k-1}^{-1}b_k) \leq \prod_{k=1}^{n}\gamma_k. \tag{2.29}$$

The positive definiteness of each Γ_{k-1}^{-1} guarantees that this inequality will be strict unless all of the off-diagonal elements of M_Θ are 0. Conversely, if all of the off-diagonal elements of M_Θ are zero, the the expression in Equation 2.29 becomes an equality. ■

Hence, we can find a suitable Q_1^T by searching $O(n)$ for the global minimum of J_1. It is convenient to redefine J_1 as

$$J_1(\Theta) = \sum_{i=1}^{n}\log\left(\sum_{j=1}^{n}\left(\Theta g_j^c\right)_i^2\right). \tag{2.30}$$

In a similar manner, we can define a cost function $J_2(\Theta)$ which can be used to search for Q_2^T.

$$J_2(\Theta) = \sum_{i=1}^{n} \log \left(\sum_{j=1}^{n} \left(\Theta g_j^r \right)_i^2 \right), \tag{2.31}$$

where $g_1^r, g_2^r, \ldots g_n^r$ are the transposes of the rows of G.

2.3.2 Searching the Wavelet Packet

In this section we address the problem of searching the wavelet packet to find the "best" approximations of the SVD factors Q_1 and Q_2. We find these approximations by minimizing the functions $J_1(\Theta)$ and $J_2(\Theta)$ for $\Theta \in L$, where L denotes the collection of orthogonal transforms contained in the wavelet packet. The $\Theta \in L$ which minimizes J_1 gives us the approximation $\widehat{Q}_1^T$ while the Θ which minimizes J_2 yields the approximation $\widehat{Q}_2^T$. $\widehat{Q}_1^T$ is the wavelet packet transform which most nearly diagonalizes the matrix GG^T and $\widehat{Q}_2^T$ is the wavelet packet transform which most nearly diagonalizes the matrix G^TG. The algorithm used to search the packet is due to Wickerhauser [43].

The transform $\widehat{Q}_1^T$ is the solution of the minimization problem

$$\min_{\Theta \in L} J_1(\Theta), \tag{2.32}$$

where

$$J_1(\Theta) = \sum_{i=1}^{n} \log \left(\sum_{j=1}^{n} \left(\Theta g_j^c \right)_i^2 \right). \tag{2.33}$$

To understand the recursive search developed by Wickerhauser, we refer back to the tree representation of the wavelet packet shown in Figure 2.3. Any transform in the wavelet packet can be represented by a collection of blocks on the tree with the property that any vertical line drawn through the tree intersects exactly one block. Such a collection is called a *graph*. Let $\mathcal{G}$ be a given graph and let Θ be

the associated wavelet packet transform. If the elements of a vector g are placed in the top block (block 1, level 1), then the transformed vector Θg is given by the coefficients located in the blocks of the graph $\mathcal{G}$. The blocks in any graph are independent of each other in the following sense: for any given block in the graph, the other blocks in the graph can be changed to create a new graph without affecting the coefficients contained in the original block.

Note that the ith component in the outer summation in the right hand side of Equation 2.33 depends only on the ith element of each transformed column of the plant matrix, Θg_j^c, $j = 1, 2, \ldots, n$. This allows us to construct a tree of costs. We begin by constructing a tree representation for each g_j^c, $j = 1, 2, \ldots, n$. We then square each element in each of these trees. Next, we add the n squared trees together on an element by element basis to create one tree called the "sum of squares" tree. Finally, the sum of squares tree is converted to the cost tree by taking the logarithm and adding the resulting numbers within each block together.

The cost tree assigns a cost to each block in the tree representation of the wavelet packet. Let Θ be a given wavelet packet transform and let $\mathcal{G}$ be the associated graph. The cost of Θ can be computed by adding together the costs associated with the blocks contained in $\mathcal{G}$. The cost computed in this manner is identical to the cost described by Equation 2.33.

Now the cost tree can be efficiently searched to find the wavelet packet transform which yields the lowest cost. This is done by comparing the cost of each block with the sum of the "best costs" of its two child blocks. Of course, finding the best cost of each child block requires comparing the child to the child's children, and so on. This recursion terminates, and the collection of blocks that results gives the wavelet packet transform that yields the lowest cost. The resulting transform

is then used as the SVD factor approximation $\widehat{Q}_1^T$. The complexity of this search, including the construction of the cost tree, is $\mathcal{O}(p^2 \log p)$, where p is the number of rows in the plant matrix. A more rigorous discussion of the properties of this search can be found in [43].

A similar search using $J_2(\Theta)$ is used to obtain the SVD factor approximation $\widehat{Q}_2^T$.

2.4 Examples

Here we introduce two example systems to demonstrate approximate diagonalization using wavelet packets. The first system is a simplified model of a flexible membrane driven by a linear array of point forces. This example is motivated by the MEMS mirror array developed by Bifano et.al. [3]. The second example is that of a flexible cantilever beam driven at resonance. These examples will also be used to demonstrate the effectiveness of other ideas which will be presented later in the thesis.

2.4.1 Simplified Flexible Membrane

The system we consider has 32 inputs and 32 outputs. The inputs correspond to electrostatic actuators, which we assume act as point forces on the membrane. The actuators are evenly spaced over the length of the array. Let this interval be denoted by Δ. The outputs correspond to the displacement of the mirror measured at the positions of the 32 actuators. We assume that the displacement y_i due to the input u_i is simply u_i. To determine the displacement y_j due to u_i, we assume that the deflection of the mirror at the positions Δ to the left of the first

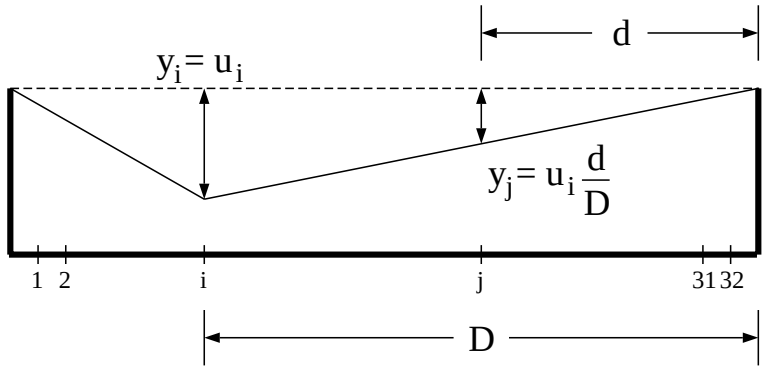


Figure 2.10: Response of the membrane to the ith input.

actuator and Δ to the right of the 32nd actuator is always zero. The displacement changes linearly between the two end points and the position of the ith actuator (see Figure 2.10). We also assume that the displacement at a given position is the sum of the displacements due to each of the actuators so that the plant can be represented by a matrix. The plant matrix G is depicted in Figure 2.11. Here, the intensities of the pixels correspond to the values of the entries in the matrix.

We applied the method described in this chapter to find the transforms from the Haar–Walsh packet which most closely diagonalize the plant matrix G. The resulting "approximately diagonal" matrix is shown in Figure 2.12. The diagonalization of the plant matrix using SVD is shown in Figure 2.13. Comparing the two, we see that the approximately diagonalized matrix resembles the matrix diagonalized with SVD. A more quantitative discussion of these results can be found in Chapter 6.

This example can be used to further illustrate the implementation of the wavelet packet transforms. In the tree representation, the wavelet packet transform which corresponds to the output transformation Q_1^T is shown in Figure 2.14. In this figure, the elements of the transformed data vector are contained in the solid black

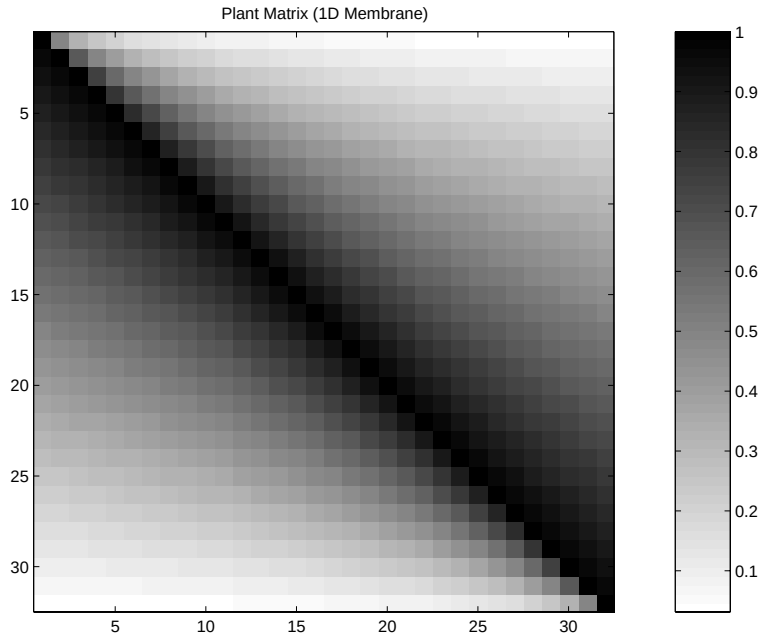


Figure 2.11: Plant matrix for flexible membrane.

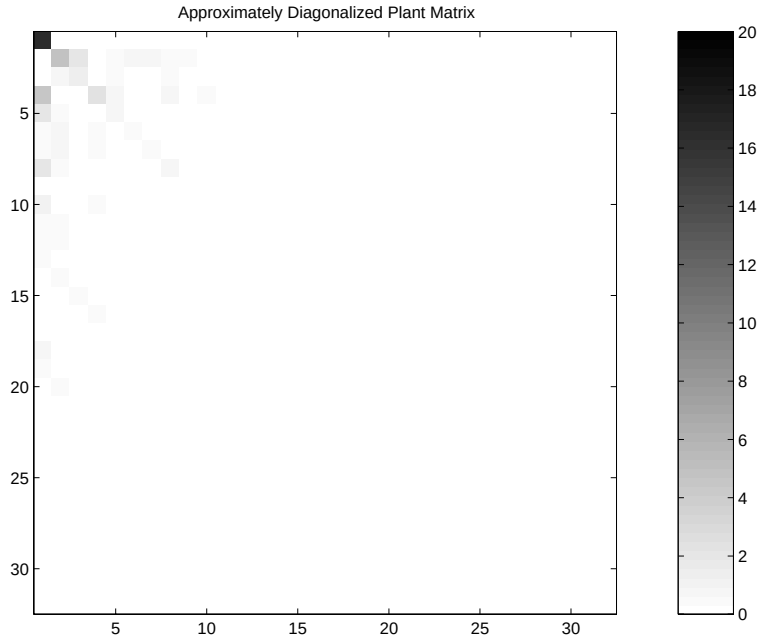


Figure 2.12: Approximately diagonalized membrane plant matrix using a transform from the Haar–Walsh packet.

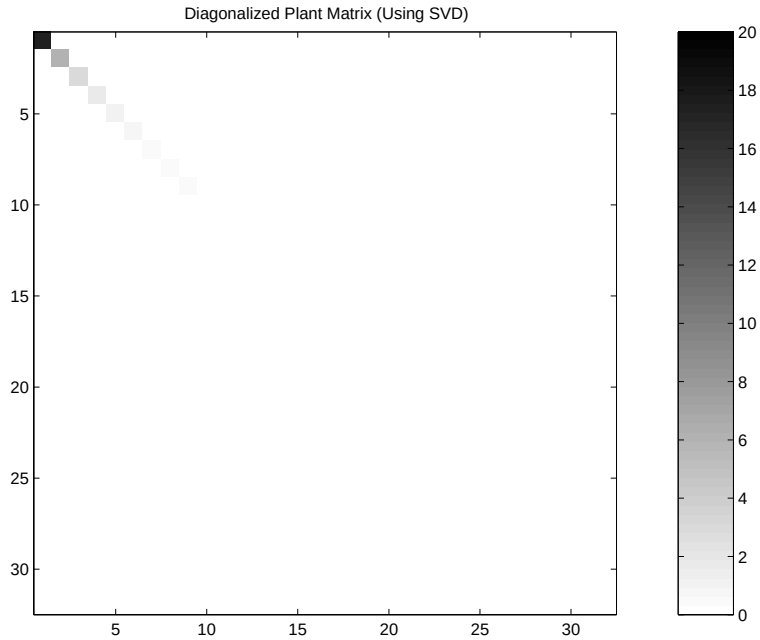


Figure 2.13: Diagonalized membrane plant matrix using exact SVD factors.

boxes. A network hierarchy which could be used to implement this transform is shown in Figure 2.15. The circles represent numbers, with the circles in the leftmost column representing the elements of y. The solid circles represent elements of the transformed output vector. A pair of circles with a box around them represents a computation step. The node on which the computation step is performed is indicated by the vertical position of the box; if the box is in the same row with y_i, then the corresponding computation step is executed on the ith node. In such a computation step, the numbers x_1 and x_2 are transformed to yield $(x_1 + x_2)/\sqrt{2}$ and $(-x_1 + x_2)/\sqrt{2}$. The lines with arrows on them indicate the communication of one number from one node to another. The transform proceeds from left to right in the diagram.

The input transformation, which in this example happens to be the inverse of the output transformation, is implemented by reversing this process. To visualize

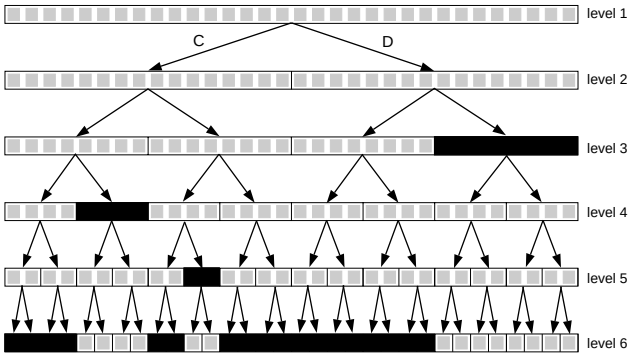


Figure 2.14: Tree representation of the wavelet packet basis which approximately diagonalizes the plant matrix.

how the inverse transform can be implemented on a distributed control network we again look to Figure 2.15. In this case, the numbers we start with are the elements of $\widetilde{u}$. The job of the transform is to transform $\widetilde{u}$ to the actual input vector u. The elements of $\widetilde{u}$ are indicated in the figure by the solid circles. At the beginning of the transform they are distributed among various nodes on the network. The transform executes across the figure from right to left with the data transfers flowing against the directions of the arrows. As in the output transformation figure, the boxes represent a computation step performed on the specified node. In this case, the numbers x_1 and x_2 are transformed to yield $(x_1 + x_2)/\sqrt{2}$ and $(x_1 - x_2)/\sqrt{2}$. After this computation is complete, the resulting numbers are then sent to the next two blocks of nodes on the hierarchy. The sums are sent to the upper blocks and the differences are sent to the lower blocks. When the transform is complete, the elements of u are distributed on the network nodes with u_i residing on the ith node.

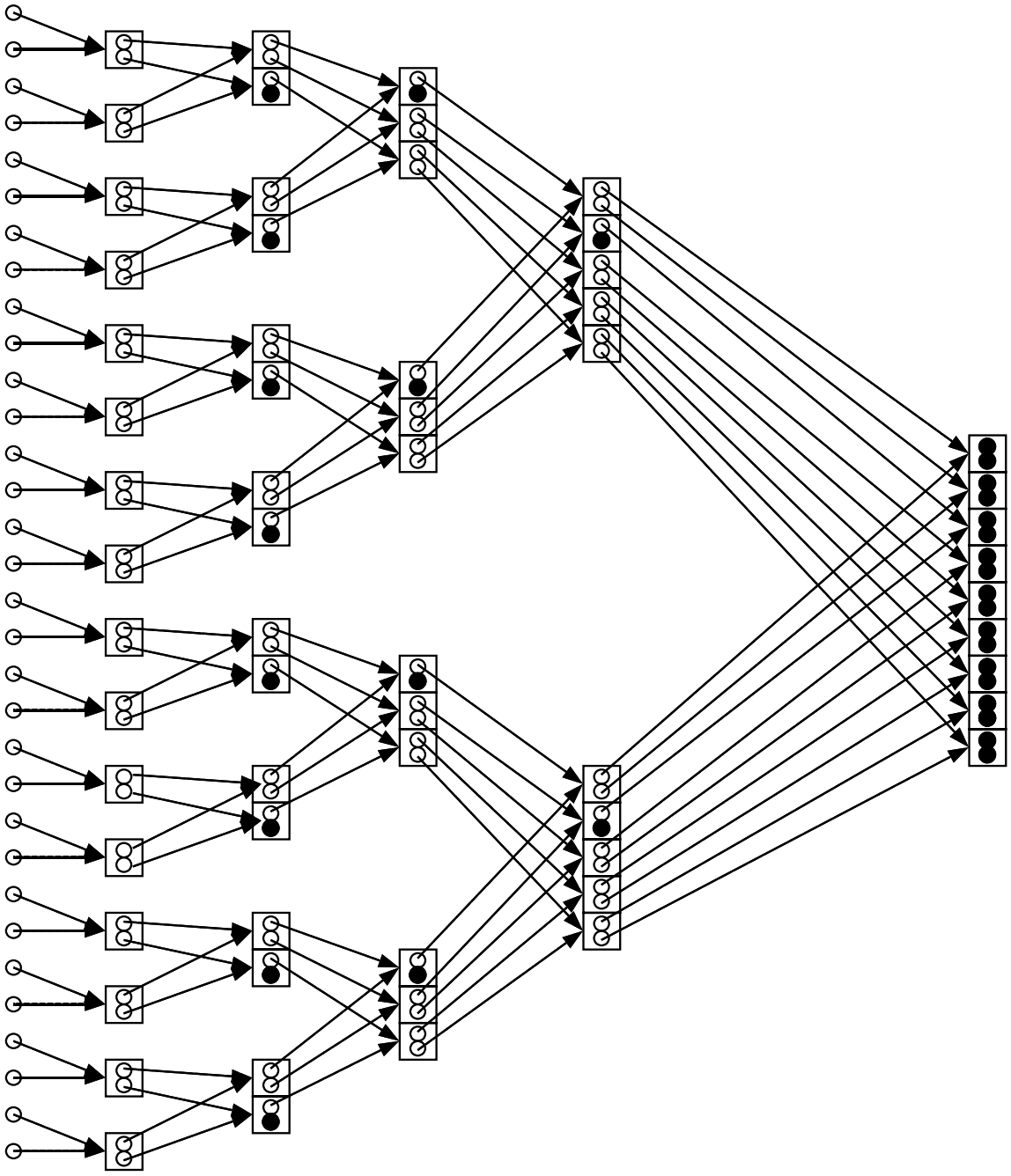


Figure 2.15: Graphical depiction of the implementation of the wavelet packet transform shown in Figure 2.14.

2.4.2 Flexible Beam at Resonance

In this example we consider a flexible cantilever beam driven by PZT actuators as described in Appendix B. Most of the energy in the response of the beam lies within narrow bands in the frequency domain which correspond to the resonant frequencies of the beam. Because the beam naturally rejects non–resonant disturbances very well, the resonant frequencies are separated by relatively large frequency bands where the gain is very small. In this sense, the resonances are isolated from one another. In most applications where feedback control is used for resonant systems, it is only necessary to apply contol in the resonant frequency bands.

One method of rejecting near–resonant disturbances in a SISO flexible beam is the so–called positive position feedback (PPF) method developed by Fanson and Caughey [18]. This technique uses a bank of bandpass filters to seperate the output into its resonances. Each resonance is then controlled independently. A similar "resonance by resonance" approach can be taken for a MIMO beam. At resonance (or at any fixed frequency) a MIMO beam can be modeled by a fixed complex matrix, i.e.

$$y = G_c u. \tag{2.34}$$

The matrix G_c is just the transfer function matrix of the beam evaluated at the resonant frequency. The complex matrix G_c can be diagonalized using the same basis transforms which diagonalize the corresponding magnitude matrix G, i.e. the matrix containing the magnitudes of the elements of G_c. The complex elements of the vectors y and u represent the magnitude and phase of a sinusiod at the resonant frequency.

Here we consider a flexible beam with 32 inputs and 32 outputs at its 6th

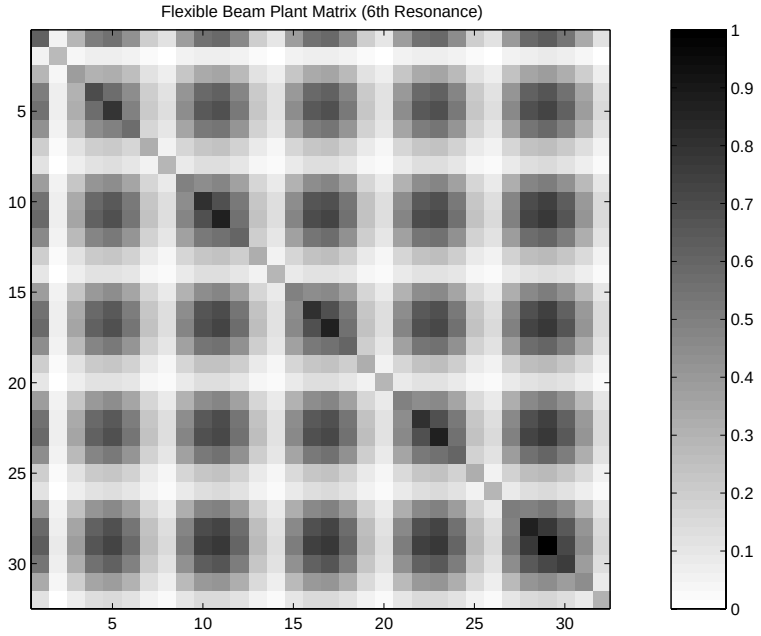


Figure 2.16: Plant matrix for flexible beam at 6th resonance.

resonant frequency. The magnitude plant matrix for this system is depicted in Figure 2.16. The wavelet packet based approximate diagonalization of the magnitude matrix is depicted in Figure 2.17. The diagonalization generated using exact SVD factors is shown in Figure 2.18.

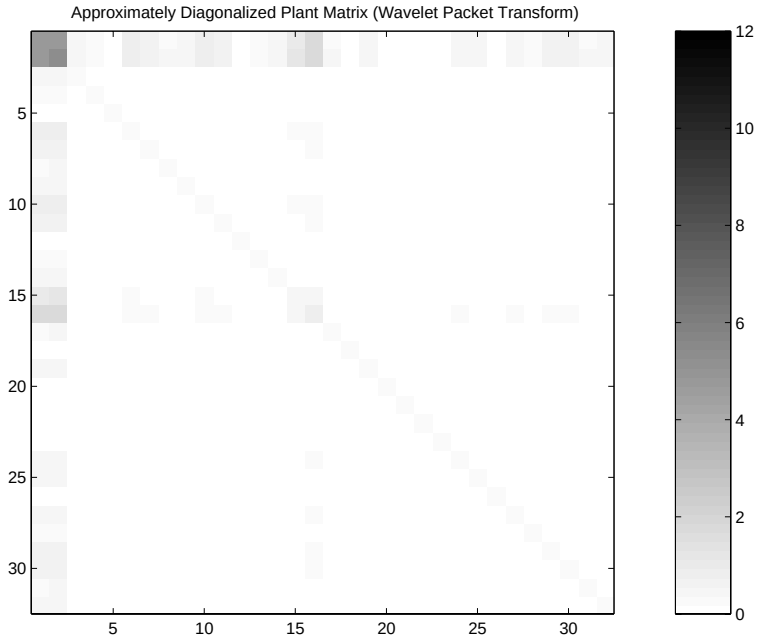


Figure 2.17: Approximately diagonalized plant matrix for flexible beam using a transform from the Haar–Walsh packet.

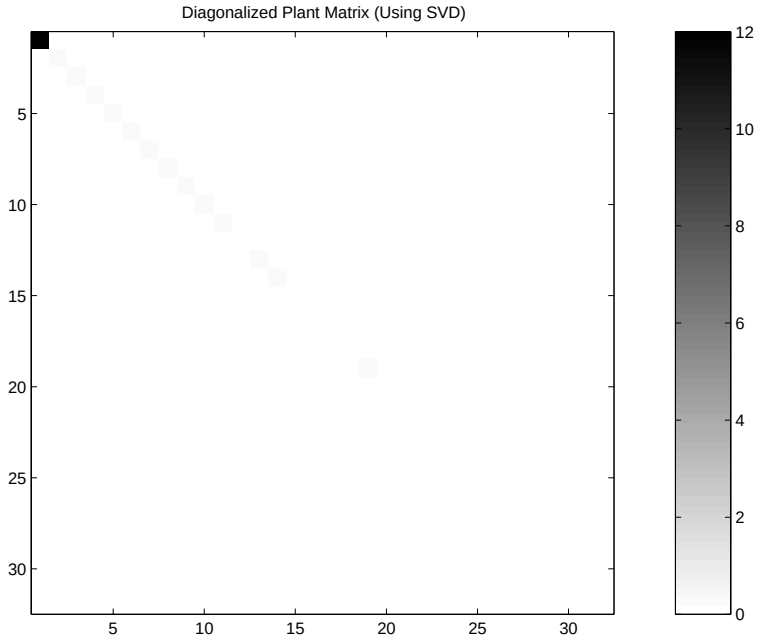


Figure 2.18: Diagonalized flexible beam plant matrix using exact SVD factors

Chapter 3

Recursive Orthogonal Transforms

In this chapter we introduce a class of linear operators that are well suited to being implemented in a distributed fashion on a control network. The operators we present have two features which make them particularly useful for implementation on a control network. First, the required computations are naturally distributed to take advantage of the parallel processing capability of the network. Second, the transforms can be chosen so that only nearest neighbor communication is required, a fact which is important for systems with limited communication bandwidth.

We begin our discussion with a review of the necessary mathematical tools and concepts that will be required here and in following chapters. We then move on to an ad hoc discussion about implementing data transformations on the input and output vectors of one and two dimensional arrays of sensor/actuator pairs. We then generalize these ideas and define the main tool which will be used in the remainder of the thesis: recursive orthogonal transforms (ROTs).

3.1 Preliminaries

Here we present the mathematical tools and concepts necessary to develop the discussion. We first review some basic properties of Lie groups and vector fields on Lie groups. Next we turn to linear algebra for a brief discussion of singular value decomposition and symmetric matrix diagonalization. Finally, we present some miscellaneous matrix facts which will be required later in this document.

3.1.1 Vector Fields on Lie Groups

Here we present some requisite information on Lie groups and their associated vector fields. A comprehensive treatment of this subject can be found in the references [40], [20], [42], and [31]. A nice overview appears in the appendix of [22]. We concentrate on the orthogonal group, $O(n)$, the special orthogonal group, $SO(n)$, and the unitary group, $U(n)$.

Smooth Manifolds

Formally, a smooth manifold is a Hausdorff topological space which is locally diffeomorphic to $\mathbb{R}^n$ around each point. An example of a smooth manifold is the circle, $S^1 = \{(x_1, x_2) \in \mathbb{R}^2 : x_1^2 + x_2^2 = 1\}$. Around each point, a neighborhood of the circle can be "flattened out" using a smooth, invertible mapping. Hence, the circle is locally diffeomorphic to $\mathbb{R}$. With this intuition, we state a more rigorous definition of a smooth manifold.

Definition 3.1.1 *A* smooth manifold *is a topological Hausdorff space M along with a collection of charts $\mathcal{A} = \{(\phi_\alpha, U_\alpha, V_\alpha) : \quad \alpha \in A\}$. For each $\alpha \in A$, U_α and V_α are open subsets of M and $\mathbb{R}^n$, respectively, and $\phi_\alpha : U_\alpha \to V_\alpha$ is a smooth, invertible map. In addition, the charts must satisfy the following:*

1. *$M = \bigcup_{\alpha \in A} U_\alpha$, i.e. the charts* cover *M.*

2. *$\phi_\beta \circ \phi_\alpha^{-1} : \phi_\alpha(U_\alpha \cap U_\beta) \subset V_\alpha \to \phi_\beta(U_\alpha \cap U_\beta) \subset V_\beta$ is a diffeomorphism for each $\alpha, \beta \in A$, i.e. the charts are* compatible *where their domains intersect.*

3. *Any chart (ϕ, U, V) which is compatible with every chart in $\mathcal{A}$ is also contained in $\mathcal{A}$, i.e. $\mathcal{A}$ is* maximal.

The collection $\mathcal{A}$ is called an *atlas*. The integer n is called the *dimension* of the manifold. The collection of charts allows us to represent an open neighborhood of any point on the manifold as an open subset of $\mathbb{R}^n$. This representation is referred to as the *local coordinate representation* of M.

Lie Groups

Here we introduce Lie groups, which are a special type of smooth manifold.

Definition 3.1.2 *A* group *is a set G along with an operation $\circ$ which has the following properties:*

1. *$g \circ h \in G$ for all $g, h \in G$, i.e. the group is* closed *under the operation $\circ$.*

2. *$(g_1 \circ g_2) \circ g_3 = g_1 \circ (g_2 \circ g_3)$ for all $g_1, g_2, g_3 \in G$, i.e. the operation $\circ$ is* associative.

3. *There exists $e \in G$ such that $g \circ e = e \circ g = g$ for all $g \in G$, i.e. the group contains an* identity *element.*

4. *For each $g \in G$, there exists $g^{-1} \in G$ such that $g \circ g^{-1} = e$, i.e. each group element has an* inverse *which is also contained in the group.*

If H is a subset of G which is also a group, then H is called a subgroup *of G.*

To simplify the notation, we will drop the use of the $\circ$ symbol so that $g \circ h$ will be written gh.

Definition 3.1.3 *A* Lie group *G is a smooth manifold with a group structure such that the group operation $\circ : G \times G \to G$, $(g, h) \mapsto gh$ is a smooth map. If H is a subgroup of G it is called a* Lie subgroup.

We are most interested in the matrix Lie groups $O(n)$, $SO(n)$, and $U(n)$. Here we define these groups and state some of their basic properties without proof.

The Orthogonal Group, $O(n)$:

Abstractly, the orthogonal group can be thought of as the set of rotations and reflections in $\mathbb{R}^n$. This group can be represented as the following matrix Lie group:

$$O(n) = \left\{\Theta \in \mathbb{R}^{n\times n} | \Theta^T\Theta = \mathbb{1}\right\}. \tag{3.1}$$

$O(n)$ is a smooth manifold with dimension $n(n-1)/2$.

The Special Orthogonal Group, $SO(n)$:

It is readily seen that each element of $O(n)$ has determinant equal to either $+1$ or -1. The subset of matrices in $O(n)$ with determinant equal to $+1$ is also a Lie group. This subgroup is called the special orthogonal group and is denoted $SO(n)$. The special orthogonal group can be thought as the set of rotations in $\mathbb{R}^n$. $SO(n)$ has the same dimension as $O(n)$.

The Unitary Group, $U(n)$:

The unitary group is the generalization of $O(n)$ to complex valued matrices. Specifically,

$$U(n) = \left\{\Theta \in \mathbb{C}^{n\times n} | \Theta^H\Theta = \mathbb{1}\right\}. \tag{3.2}$$

The real dimension of $U(n)$ is n^2. $U(n)$ contains both $O(n)$ and $SO(n)$.

Tangent Spaces and Vector Fields

At each point on a smooth manifold there is an associated vector space called the tangent space. Intuitively, if we imagine an infinitely small object which moves around on the manifold, the tangent space at a given point contains the set of all possible velocities the object can attain at that point. The tangent space of M at the point x is denoted T_xM. In the example of the circle, the tangent space at a given point x is the line tangent to the circle at x, or

$$T_xS^1 = \left\{ y \in \mathbb{R}^2 | y^T x = 0 \right\}. \tag{3.3}$$

To define more rigorously what we mean by tangent vectors and tangent spaces, we begin by defining the notion of a curve on the manifold. Specifically, a curve through $x \in M$ is a smooth function $f : (-\epsilon, \epsilon) \to M$, some real $\epsilon > 0$, with $f(0) = x$. Let (ϕ, U, V) be a coordinate chart with $x \in U$. Let F_x denote the set of all possible curves at x. In a neighborhood of x, the curve f can be represented as a curve in $\mathbb{R}^n$ using local coordinates, $\phi \circ f : (-\delta, \delta) \to V \subset \mathbb{R}^n$ for some $\delta > 0$. If we think of the argument of f as time, then the velocity of a point moving along the curve at the point x is given in local coordinates by derivative of $\phi \circ f$ at $x = 0$. Define the set

$$[f]_x = \left\{ g \in F_x \left| \frac{d}{dt}(\phi(g(t)))_{t=0} = \frac{d}{dt}(\phi(f(t))) \right. \right\}. \tag{3.4}$$

$[f]_x$ is the equivalence class of all curves through x which have the same velocity at x. This leads to the following definition:

Definition 3.1.4 *A* tangent vector *to M at the point x is an equivalence class $[f]_x$, where $f \in F_x$. The* tangent space *of M at x, T_xM, is the set of all tangent vectors to M at x. The* tangent bundle *of M, denoted TM, is the set containing*

the tangent spaces of every $x \in M$,

$$TM = \{T_x M | x \in M\}\,. \tag{3.5}$$

An element of the tangent bundle is an ordered pair (x, p) where x is a point on the manifold and p is a vector in $T_x M$. There is a natural projection from the tangent bundle to the manifold, $\pi : TM \to M$; $(x, p) \mapsto x$. We are now ready to define a vector field:

Definition 3.1.5 *A* vector field *on a smooth manifold M is a map $X : M \to TM$ with the condition that $\pi \circ X$ is the identity map.*

Hence, the vector field assigns a tangent vector p_x to each point on the manifold.

The *flow* of a vector field starting from the point $x_0 \in M$ is the curve $x(\cdot) : \mathbb{R} \to M$ that satisfies the differential equation

$$\dot{x}(t) = p_x, \qquad x(0) = x_0. \tag{3.6}$$

The Lie Bracket and Lie Algebras

Every Lie group G has associated with it a vector space called a Lie algebra denoted $\mathfrak{g}$.

Definition 3.1.6 *A* Lie algebra *is a vector space V together with the* Lie bracket $[\cdot, \cdot] : V \times V \to V$ *such that*

1. $[a, b] = -[b, a]$ *for all* $a, b \in V$.
2. $[\alpha a + \beta b, c] = \alpha[a, c] + \beta[b, c]$ *for all* $\alpha, \beta \in \mathbb{R}$, $a, b, c \in V$.
3. $[a, [b, c]] + [c, [a, b]] + [b, [c, a]] = 0$ *for all* $a, b, c \in V$ *(The Jacobi Identity).*

The Lie algebra of a Lie group G is given by the tangent space at the identity, $\mathfrak{g} = T_eG$, together with an appropriate Lie bracket. The Lie algebras of $O(n)$, $SO(n)$, and $U(n)$ are denoted $\mathfrak{o}(n)$, $\mathfrak{so}(n)$, and $\mathfrak{u}(n)$, respectively. They are

$$\mathfrak{o}(n) = \mathfrak{so}(n) = \left\{\Omega \in \mathbb{R}^{n\times n} | \Omega = -\Omega^T\right\} \tag{3.7}$$

and

$$\mathfrak{u}(n) = \left\{\Omega \in \mathbb{C}^{n\times n} | \Omega = -\Omega^H\right\}. \tag{3.8}$$

The Lie bracket for $\mathfrak{o}(n)$, $\mathfrak{so}(n)$, and $\mathfrak{u}(n)$ is given by the matrix commutator,

$$[A, B] = AB - BA. \tag{3.9}$$

One of the nice properties of Lie groups is that the tangent space at the identity can be translated to the tangent space at $g \in G$ via left multiplication by g. Hence, for matrix Lie groups, the tangent space at $g \in G$ can be written

$$T_gG = \{g\Omega | \Omega \in \mathfrak{g}\}. \tag{3.10}$$

Each vector Ω in the matrix Lie algebra naturally generates a vector field on the Lie group via the definition $X_\Omega : G \to TG$; $g \mapsto (g, g\Omega)$. Such a vector field is said to be *left invariant*. Flows of X_Ω are curves which satisfy

$$\dot{g} = g\Omega, \qquad g(0) = g_0 \in G. \tag{3.11}$$

On a matrix Lie group this equation is a linear time invariant matrix ODE. The solution is given by

$$g(t) = g_0 e^{\Omega t} \tag{3.12}$$

where the matrix exponential is defined by the series

$$e^A = \sum_{k=0}^{\infty} \frac{A^k}{k!}. \tag{3.13}$$

Riemannian Metrics, Directional Derivatives, and Gradient Flows

Each smooth function $f : M \to \mathbb{R}$ has associated with it a gradient ascent vector field. At each point in M, the gradient vector field assigns a tangent vector which points in the direction of "steepest ascent", i.e. the direction in which f increases the most. Following [22], we present the tools necessary to formalize this notion in the setting of smooth manifolds.

Definition 3.1.7 *A* Riemannian metric *on a manifold M is a collection of nondegenerate inner products $\{\langle\cdot,\cdot\rangle_x : T_xM \to \mathbb{R}\}$ for each $x \in M$ where $\langle\cdot,\cdot\rangle_x$ depends smoothly on x. A manifold with such a specified collection is called a* Riemannian manifold.

For $O(n)$ and $SO(n)$, the tangent space at any given point is a subset of $\mathbb{R}^{n\times n}$. As a result, the standard inner product on $n \times n$ real matrices, $\langle A, B\rangle = \operatorname{tr}\left(A^T B\right)$, provides a suitable Riemannian metric:

$$\begin{aligned} \langle\cdot,\cdot\rangle : T_gO(n) \times T_gO(n) &\to \mathbb{R} \\ (g\Omega_1, g\Omega_2) &\mapsto \operatorname{tr}\left(\Omega_1^T g^T g\Omega_2\right) \\ &= \operatorname{tr}\left(\Omega_1^T \Omega_2\right), \end{aligned} \tag{3.14}$$

where $\Omega_1, \Omega_2 \in \mathfrak{o}(n)$. Similarly, $U(n)$ inherits a Riemannian metric from $\mathbb{C}^{n\times n}$:

$$\begin{aligned} \langle\cdot,\cdot\rangle : T_gU(n) \times T_gU(n) &\to \mathbb{R} \\ (g\Omega_1, g\Omega_2) &\mapsto \operatorname{retr}\left(\Omega_1^H g^H g\Omega_2\right) \\ &= \operatorname{retr}\left(\Omega_1^H \Omega_2\right), \end{aligned} \tag{3.15}$$

where $\Omega_1, \Omega_2 \in \mathfrak{u}(n)$ and $\operatorname{retr}(\cdot)$ denotes the real part of the trace.

Consider a smooth function $f : M \to \mathbb{R}$. The directional derivative of f at a point $x \in M$ is a map

$$Df_x : T_xM \to \mathbb{R}. \tag{3.16}$$

We will use $Df_x(\xi)$ to denote the directional derivative of f at x in the direction $\xi \in T_xM$.

Now we consider the gradient vector field of f, which is denoted as ∇f. The gradient vector field is defined as the vector field which satisfies the following two properties:

1. $\nabla f(x) \in T_xM$ for all $x \in M$.

2. $Df_x(\xi) = \langle \nabla f(x), \xi \rangle$ for all $\xi \in T_xM$.

The first property states that the gradient vector is contained in the tangent space of M at every point in M. The second property is a compatibility condition between the gradient and the directional derivative. For a given Riemannian metric, the gradient vector field which has both properties is unique. The ascent direction gradient flow from initial condition $x_0 \in M$ is given by the solution to

$$\dot{x} = \nabla f(x), \qquad x(0) = x_0. \tag{3.17}$$

In this document we will be interested in gradient flows which evolve on the compact Lie groups $SO(n)$ and $U(n)$. Here we state some useful properties of a gradient flow on a compact manifold without proof [22].

Proposition 3.1.1 *Consider the gradient flow given by Equation 3.17 where the manifold M is compact. Then for any $x_0 \in M$:*

1. *The solution $x(t)$ exists for all time $t \in \mathbb{R}$.*

2. *The function $f(x(t))$ is a nondecreasing function of t.*

3. *The solution $x(t)$ converges to $x^* \in M$ such that $\nabla f(x^*) = 0$.*

3.1.2 Matrix Facts

Here we present some facts about matrices which will be required for upcoming calculations.

Fact 3.1.1 *Any $A \in \mathbb{C}^{n\times n}$ can be written as*

$$A = S + \Omega, \tag{3.18}$$

where S is hermitian ($S = S^H$) and (Ω) is skew–hermitian ($\Omega = -\Omega^H$). Specifically, S and Ω are given by

$$S = \frac{1}{2}\left(A + A^H\right) \tag{3.19}$$

$$\Omega = \frac{1}{2}\left(A - A^H\right). \tag{3.20}$$

This fact can be easily verified by adding the expressions for S and Ω. Clearly, this fact can be applied to $A \in \mathbb{R}^{n\times n}$ by replacing the hermitian transpose with the regular matrix transpose.

Fact 3.1.2 *Let $S = S^H \in \mathbb{C}^{n\times n}$ and $\Omega = -\Omega^H \in \mathbb{C}^{n\times n}$. Then retr$(S\Omega) = 0$.*

<u>Proof:</u>

We begin by computing

$$\begin{aligned} \mathrm{retr}\left((S\Omega)^H\right) &= \mathrm{retr}\left(\Omega^H S^H\right) \\ &= \mathrm{retr}\left(-\Omega S\right) \\ &= -\mathrm{retr}\left(S\Omega\right). \end{aligned} \tag{3.21}$$

Since $\text{retr}\,(A) = \text{retr}\left(A^H\right)$ for any complex matrix A, we then have $\text{retr}\,(S\Omega) = -\text{retr}\,(S\Omega)$. This can only be true if $\text{retr}\,(S\Omega) = 0$. ■

Clearly, if S is a real valued symmetric matrix and Ω is real valued and skew–symmetric, then $\text{tr}\,(S\Omega) = 0$.

Fact 3.1.3 *Let $A \in \mathbb{C}^{n\times n}$ be written $A = S + \Omega$ where $S = S^H$ and $\Omega = -\Omega^H$. Let $\Lambda = -\Lambda^H \in \mathbb{C}^{n\times n}$. Then $\text{retr}\,(A\Lambda) = \text{retr}\,(\Omega\Lambda)$.*

This fact follows directly from Fact 3.1.1 and Fact 3.1.2.

3.2 Parallel Signal Processing on Control Networks

Here we provide an ad hoc discussion about the implementation of coordinate transforms on the input and output vectors of a large control network. This discussion leads us to a collection of orthogonal transforms which have a special structure which allows the transforms to be efficiently implemented on the systems we consider. We restrict the discussion to control networks whose nodes are sensor/actuator pairs. We first investigate the case where the nodes are distributed spatially along a linear array. We then address a similar control network whose nodes are distributed on a rectangular array.

3.2.1 Implementation on 1–D Array

In this section we address the problem of implementing input and output data transforms on a control network where n sensor/actuator pairs are distributed spatially to form a one dimensional array. The sensor/actuator pairs, or nodes,

are indexed from left to right; the leftmost node has the index 1. For each i, the ith node contains one sensor, whose output is y_i, and one actuator, whose input is u_i. We assume that each node on the array can communicate with both of its neighbors. At the endpoints of the array, it is assumed that the first node can communicate with the last node. For $i = 1, 2, \ldots, n-1$ we say that the $(i+1)$th node is to the right of the ith node and the ith node is to the left of the $(i+1)$th node. Similarly, we say that the first node is to the right of the last node and the last node is to the left of the first node. The presented data transformation method requires only nearest neighbor communication and distributes the computations required for the transforms in a natural way.

First we concentrate on the problem of transforming output vector y. The result of this operation is the transformed output vector $\widetilde{y} = Q_1 y$, where the orthogonal transform Q_1 has a special structure. We address only the implementation of Q_1, we do not address what Q_1 actually does to the data. This question will be addressed in Chapter 4. We then show how a similar method can be used to transform the input vector, yielding $u = Q_2^{-1}\widetilde{u}$.

Transforming the Output Vector

Here we present a data transformation which can be easily applied to the output vector of a control network. We call the transformed data vector $\widetilde{y}$ and the data transformation that results is denoted by Q_1, so $\widetilde{y} = Q_1 y$. The sensors are distributed along a line and each sensor has its own microprocessor. There is also an actuator paired with each sensor. The combination of sensor, actuator, and microprocessor will be referred to as a node. The ith output y_i is known only to the ith node. The object is to compute $\widetilde{y} = Q_1 y$ while limiting the required

communication and distributing the required computation.

The implementation we propose consists of a sequence of steps or levels. We will present these steps here without motivating our choices. Our reasoning should become clear at the end of this section. Also, to simplify this presentation we assume that n is even. The extension to odd n will be addressed when we generalize these ideas in Section 3.3.

In the first level, each evenly indexed node passes the value of its output to the node to its left. As a result, for each odd i, the ith node contains y_i and y_{i+1}. The ith node then uses its local microprocessor to compute

$$\begin{bmatrix} a_i \\ a_{i+1} \end{bmatrix} = \theta^1_{(i+1)/2} \begin{bmatrix} y_i \\ y_{i+1} \end{bmatrix}, \tag{3.22}$$

where the matrix $\theta^1_{(i+1)/2} \in SO(2)$ for each odd i. The output of the first level is the intermediate vector a, which can be written

$$a = \Theta_1 y, \tag{3.23}$$

where

$$\Theta_1 = \begin{bmatrix} \theta^1_1 & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \theta^1_2 & \cdots & \mathbf{0} \\ \vdots & & \ddots & \vdots \\ \mathbf{0} & \cdots & \mathbf{0} & \theta^1_{n/2} \end{bmatrix}. \tag{3.24}$$

When the first level is complete, the ith node, i odd, contains the numbers a_i and a_{i+1}. To begin the second level, the ith node passes a_i to the node to its left and a_{i+1} to the node to its right. Recall that nth node is to the left of the first node, so a_1 gets sent to the nth node. With the exception of the nth node, the ith node for each even i contains a_i and a_{i+1}. The nth node contains a_n and a_1. Then

each ith node, $i = 2, 4, 6, \ldots, n-2$, uses its local processor to compute

$$\begin{bmatrix} b_{i-1} \\ b_i \end{bmatrix} = \theta_{i/2}^2 \begin{bmatrix} a_i \\ a_{i+1} \end{bmatrix}. \tag{3.25}$$

The nth node computes

$$\begin{bmatrix} b_{n-1} \\ b_n \end{bmatrix} = \theta_{n/2}^2 \begin{bmatrix} a_n \\ a_1 \end{bmatrix}. \tag{3.26}$$

The matrix $\theta_{i/2}^2 \in SO(2)$ for each $i = 2, 4, 6, \ldots, n$. The output of the second level is the vector b, which can be written

$$\begin{aligned}
\begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix} &= \begin{bmatrix} \theta_1^2 & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \theta_2^2 & \cdots & \mathbf{0} \\ \vdots & & \ddots & \vdots \\ \mathbf{0} & \cdots & \mathbf{0} & \theta_{n/2}^2 \end{bmatrix} \begin{bmatrix} a_2 \\ a_3 \\ \vdots \\ a_n \\ a_1 \end{bmatrix} \\
&= \begin{bmatrix} \theta_1^2 & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \theta_2^2 & \cdots & \mathbf{0} \\ \vdots & & \ddots & \vdots \\ \mathbf{0} & \cdots & \mathbf{0} & \theta_{n/2}^2 \end{bmatrix} \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 & 1 \\ 1 & 0 & \cdots & 0 & 0 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix} \\
&\triangleq \Theta_2 P_e a.
\end{aligned} \tag{3.27}$$

Substituting Equation 3.23 into this expression yields

$$b = \Theta_2 P_e \Theta_1 y. \tag{3.28}$$

The matrix P_e is the permutation matrix which implements a circular left shift of the vector a.

When the second level is finished the ith node contains b_{i-1} and b_i, for each even i. To start the third level, the ith node (i even) sends b_{i-1} to the node on its left and it sends b_i to the node on its right. Recall that the first node is to the right of the nth node, so b_n gets sent to the first node. Like the first and second steps, each oddly indexed node uses its local processor to rotate the elements of b which it contains, yielding the vector c:

$$\begin{aligned}
\begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix} &= \begin{bmatrix} \theta_1^3 & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \theta_2^3 & \cdots & \mathbf{0} \\ \vdots & & \ddots & \vdots \\ \mathbf{0} & \cdots & \mathbf{0} & \theta_{n/2}^3 \end{bmatrix} \begin{bmatrix} b_n \\ b_1 \\ b_2 \\ \vdots \\ b_{n-1} \end{bmatrix} \\
&= \begin{bmatrix} \theta_1^3 & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \theta_2^3 & \cdots & \mathbf{0} \\ \vdots & & \ddots & \vdots \\ \mathbf{0} & \cdots & \mathbf{0} & \theta_{n/2}^3 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 & \cdots & 1 \\ 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 0 \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix} \\
&\triangleq \Theta_3 P_o b. \qquad (3.29)
\end{aligned}$$

Substituting Equation 3.28 into this expression yields

$$c = \Theta_3 P_o \Theta_2 P_e \Theta_1 y. \qquad (3.30)$$

This process of local rotations and permutations is repeated for L levels, some $L > 0$, and $\widetilde{y}$ is defined to be the output of the Lth level,

$$\widetilde{y} = \Theta_L P_L \cdots \Theta_2 P_2 \Theta_1 y, \qquad (3.31)$$

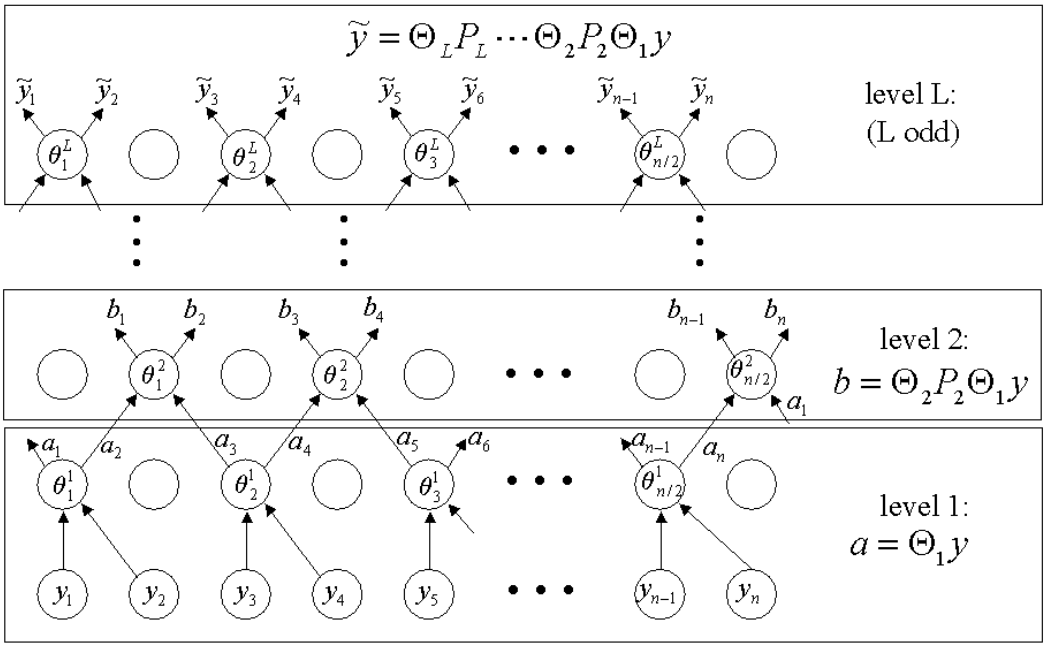


Figure 3.1: Graphical description of parallel output processing on a 1–D array of smart sensors. Data flows from the bottom up in the diagram.

where, for $i = 2, 3, \ldots, L$,

$$P_i = \begin{cases} P_o \text{ for } i \text{ odd} \\ P_e \text{ for } i \text{ even} \end{cases} \tag{3.32}$$

This process is depicted graphically in Figure 3.1.

The resulting transform Q_1 is of the form

$$Q_1 = \Theta_L P_L \cdots \Theta_2 P_2 \Theta_1. \tag{3.33}$$

Clearly, any orthogonal transform which has this structure can be implemented in a manner which naturally distributes the necessary computations and eliminates the need for global communication.

Transforming the Input Vector

In the previous section we introduced an output vector transformation Q_1 that could be implemented on the network using a series of 2×2 decoupled rotations and permutation steps. Here we consider the coordinate transformation for the input vector, Q_2. We start by assuming that, like Q_1, Q_2 is of the form

$$Q_2 = \Phi_L P_L \cdots \Phi_2 P_2 \Phi_1, \tag{3.34}$$

where the Φ_is are block diagonal rotation matrices of the same form as the Θ_is which compose Q_1. To simplify the discussion, we assume that the number of levels, in the transform Q_2 is L, the number of levels in the transform Q_1. We also assume that L is odd. The extension to more general cases will be considered in Section 3.3.

In order to understand how the input transformation is implemented, we first recall that the proposed controller implementation is based on plant diagonalization. In this form, the controller performs three basic tasks: it transforms the data vector y to $\widetilde{y}$, it chooses the transformed input vector $\widetilde{u}$ based on $\widetilde{y}$, and it transforms $\widetilde{u}$ to create the actual input vector applied to the plant, u. At the end of the output transformation step, each oddly indexed node contains two elements of the transformed output vector $\widetilde{y}$. In particular, the ith node contains the transformed outputs $\widetilde{y}_i$ and $\widetilde{y}_{(i+1)}$. [1] As seen from the transformed input and output vectors, the plant is diagonal. The controller matrix can then also be diagonal. Hence, each $\widetilde{u}_i$ can be computed based solely on the corresponding $\widetilde{y}_i$. This means that the controller can perform its second task, computing $\widetilde{u}$, in a decentralized manner by having the ith node, i odd, compute $\widetilde{u}_i$ and $\widetilde{u}_{(i+1)}$ on its local processor. So at the

[1] Remember: we have assumed that L is odd; if L was even then the outputs would be contained in the evenly indexed nodes and the ith node would contain $\widetilde{y}_{(i-1)}$ and $\widetilde{y}_i$.

beginning of the input transformation step, the transformed inputs $\widetilde{u}_i$ and $\widetilde{u}_{(i+1)}$ are contained in the ith node for each odd i. The goal of the input transformation step is to compute $u = Q_2^{-1}\widetilde{u}$ in a distributed manner on the network so that the input u_i is applied to the actuator on the ith node.

Now we are ready to describe the process used to transform $\widetilde{u}$. The input we wish to compute is

$$u = \Phi_1^T P_2^T \Phi_2^T \cdots P_L^T \Phi_L^T \widetilde{u}. \tag{3.35}$$

Like the output transformation, the input transformation is computed in levels. In the first level, the intermediate vector $a = \Phi_L^T \widetilde{u}$ is computed. The matrix Φ_L^T is block diagonal so a can be computed in a decentralized manner on the oddly indexed nodes of the network. Next, the permutation matrix P_L^T is realized using nearest neighbor communication where each odd node sends one element of a to each of its two evenly indexed neighbors. The vector $b = \Phi_{L-1}^T P_L^T a$ can then be computed in a decentralized manner on the evenly indexed nodes of the network. This process is repeated until finally the expression

$$u = \Phi_1^T P_2^T \Phi_2^T \cdots P_L^T \Phi_L^T \widetilde{u} \tag{3.36}$$

is realized. Operating under the assumption that L is odd, the elements of u will be contained in the evenly indexed nodes, with the ith node containing $u_{(i-1)}$ and u_i. The final step of the input transformation process is for each evenly indexed ith node to send the value of $u_{(i-1)}$ to the node on its left. The result is that u has been computed and each u_i resides on the ith node, ready to be applied as the input to the ith actuator.

3.2.2 Implementation on 2–D Array

In this section we show how to design specially structured orthogonal transforms Q_1 and Q_2 to be implemented on a control network where the sensor/actuator pairs are distributed spatially to form a two dimensional rectangular array. Let n_r be equal to the number of rows and n_c be equal to the number of columns in the array. To simplify the presentation, we assume that n_c and n_r are both even. The nodes in the array are numbered left to right, top to bottom. The node in the ith row and the jth column is numbered $j + n_c(i-1)$. Depending on which is more descriptive, we will refer to this node as either the (i,j)th node or the $(j + n_c(i-1))$th node. The difference should be clear from context.

The (i,j)th node is said to be *double odd* if both i and j are odd. Likewise, the (i,j)th node is said to be *double even* if both i and j are even.

The spatial arrangement of the array naturally divides the nodes on the array into three classes: interior nodes, edge nodes, and corner nodes. As the name implies, the corner nodes are at locations $(1,1)$, $(1,n_c)$, $(n_r,1)$, and (n_r,n_c). Nodes which are in the first row, last row, first column, or last column but are not corner nodes are referred to as edge nodes. Specifically, we will refer to these edges as top edge, bottom edge, left edge, and right edge, respectively. The remaining nodes are interior nodes.

Array Connections

Here we assume that the nodes on the array are connected to form a torus. To achieve this connection topology, imagine that the top and bottom edges of the array are brought together so that the array forms a cylinder with the left and right edges forming the circles at either end of the cylinder. The cylinder is then

stretched and bent to bring the left and right edges (now circles) together, forming a torus. Remember: this torus represents the topology of the connections; the physical layout of the array has not changed.

Each node on the array has eight neighbors on the torus. For interior nodes, the (i, j)th node has left and right neighbors (nodes $(i, j-1)$ and $(i, j+1)$, respectively), upper and lower neighbors (nodes $(i-1, j)$ and $(i+1, j)$, respectively), and four diagonal neighbors (nodes $(i-1, j-1)$, $(i-1, j+1)$, $(i+1, j-1)$, and $(i+1, j+1)$. Because of toroidal connection topology, nodes on the edges and corners have neighbors on the other side of the array. Modular division can be used to identify the neighbors of a general element on the array. For the (i, j)th node, the eight neighbors are

$$\begin{aligned}
\text{left neighbor} \quad & (i, ((j-2) \bmod n_c) + 1) \\
\text{right neighbor} \quad & (i, ((j) \bmod n_c) + 1) \\
\text{upper neighbor} \quad & (((i-2) \bmod n_r) + 1, j) \\
\text{lower neighbor} \quad & (((i) \bmod n_r) + 1, j) \\
\text{upper–left neighbor} \quad & (((i-2) \bmod n_r) + 1, ((j-2) \bmod n_c) + 1) \\
\text{upper–right neighbor} \quad & (((i-2) \bmod n_r) + 1, ((j) \bmod n_c) + 1) \\
\text{lower–left neighbor} \quad & (((i) \bmod n_r) + 1, ((j-2) \bmod n_c) + 1) \\
\text{lower–right neighbor} \quad & (((i) \bmod n_r) + 1, ((j) \bmod n_c) + 1)
\end{aligned} \tag{3.37}$$

The data transformation method we present here requires each node to have a two–way connection with a subset of its neighbors. In particular, each double odd node shares a connection with its right neighbor, its lower neighbor, and each of its four diagonal neighbors. This connection scheme is depicted in Figure 3.2.

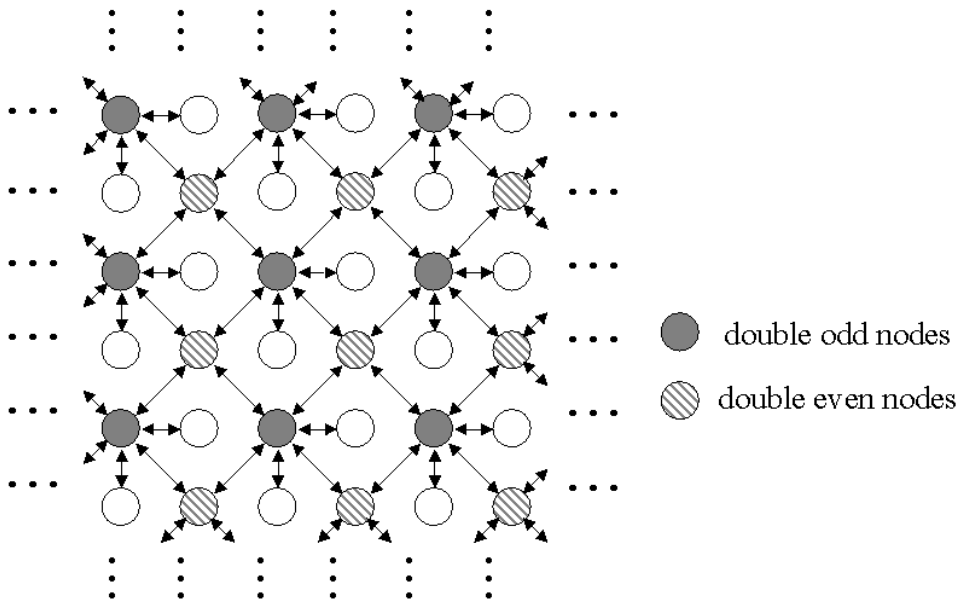


Figure 3.2: Required connections for 2–D array.

Transforming the Output Vector

Here we discuss the transformation of the output vector y on a 2–D array. The resulting transformed output vector is denoted $\widetilde{y} = Q_1 y$, where Q_1 is an orthogonal transform with a special structure. The data vector y has dimension $n_r n_c$. Let $y^\dagger(i,j)$ denote the output of the sensor on the (i,j)th node. The output of the sensor on the (i,j)th node is contained in the $(j + n_c(i-1))$th element of y, i.e. $y^\dagger(i,j) = y_{(j+n_c(i-1))}$. Conversely, the kth element of y is

$$y_k = y^\dagger\left(\left\lfloor \frac{k}{n_c} \right\rfloor + 1, ((k-1) \bmod n_c) + 1\right), \tag{3.38}$$

where $\lfloor x \rfloor$ is the largest integer less than or equal to x.

As in the case of the 1–D array, the implementation we propose consists of a sequence of levels. In the first level, each double odd node (i,j) receives the values of the sensor outputs from its right neighbor $(i, j+1)$, left neighbor $(i+1, j)$, and

lower–right neighbor $(i+1, j+1)$. As a result, each double odd node (i, j) contains the following four elements of the output vector, y:

$$y^{\dagger}(i,j) = y_{(j+n_c(i-1))} \tag{3.39}$$

$$y^{\dagger}(i,j+1) = y_{((j+1)+n_c(i-1))} \tag{3.40}$$

$$y^{\dagger}(i+1,j) = y_{(j+n_c i)} \tag{3.41}$$

$$y^{\dagger}(i+1,j+1) = y_{((j+1)+n_c i)} \tag{3.42}$$

If we order the double odd nodes from left to right, top to bottom, then the (i, j)th node is also the kth double odd node, where

$$k = \frac{j+1}{2} + \frac{n_c}{2}\frac{i-1}{2}. \tag{3.43}$$

Define the intermediate data vector a such that the elements of y contained in the kth double odd node are equal to $a_{4(k-1)+1}$ through $a_{4(k-1)+4}$. In simple language, this means that the first four elements of a are the four outputs contained in the first double odd node, the next four elements are the four outputs contained in the second double odd node, and so on. The vector a is a permutation of y. Let $\{e_1, e_2, e_3, \ldots, e_{n_r n_c}\}$ denote the canonical basis vectors for $\mathbb{R}^n$. Then we can construct the permutation matrix P_1 such that $a = P_1 y$ using the following algorithm:

for $i = \{1, 3, 5, \ldots, n_r - 1\}$
  for $j = \{1, 3, 5, \ldots, n_c - 1\}$
    let $k = \frac{j+1}{2} + \frac{n_c}{2}\frac{i-1}{2}$
    let the $(4(k-1)+1)$th column of P_1 be $e_{j+n_c(i-1)}$
    let the $(4(k-1)+2)$th column of P_1 be $e_{(j+1)+n_c(i-1)}$
    let the $(4(k-1)+3)$th column of P_1 be $e_{j+n_c i}$

let the $(4(k-1)+4)$th column of P_1 be $e_{(j+1)+n_c i}$

end j loop

end i loop

This communication step is depicted graphically in Figure 3.3.

Next, using its local microprocessor, each double odd node k computes

$$\begin{bmatrix} b_{4(k-1)+1} \\ b_{4(k-1)+2} \\ b_{4(k-1)+3} \\ b_{4(k-1)+4} \end{bmatrix} = \theta_k^1 \begin{bmatrix} a_{4(k-1)+1} \\ a_{4(k-1)+2} \\ a_{4(k-1)+3} \\ a_{4(k-1)+4} \end{bmatrix}, \tag{3.44}$$

where $\theta_k^1 \in SO(4)$, $k = 1, 2, 3, \ldots, n_c n_r/4$. The intermediate vector b is the output of the first level and can be written

$$b = \Theta_1 P_1 y, \tag{3.45}$$

where

$$\Theta_1 = \begin{bmatrix} \theta_1^1 & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \theta_2^1 & \cdots & \mathbf{0} \\ \vdots & & \ddots & \vdots \\ \mathbf{0} & \cdots & \mathbf{0} & \theta_{n_c n_r/4}^1 \end{bmatrix}. \tag{3.46}$$

In the second level, the computations will take place on the double even nodes. We number the double even nodes left to right, top to bottom so that the (i, j)th node is the kth double even node, where

$$k = \frac{j}{2} + \frac{n_c}{2}\frac{i-2}{2}. \tag{3.47}$$

Note that the kth double even node is the lower–right neighbor to the kth double odd node.

At the beginning of the second level, each double even node receives one element of b from each of its four diagonal neighbors, which are double odd nodes. As depicted in Figure 3.3 this communication step creates the new intermediate data vector c. Specifically, we set

$$c_{4(k-1)+1} = b_{4(k_{ul}-1)+4} \tag{3.48}$$

$$c_{4(k-1)+2} = b_{4(k_{ur}-1)+3} \tag{3.49}$$

$$c_{4(k-1)+3} = b_{4(k_{ll}-1)+2} \tag{3.50}$$

$$c_{4(k-1)+4} = b_{4(k_{lr}-1)+1}, \tag{3.51}$$

Where k_{ul}, k_{ur}, k_{ll}, and k_{lr} are the positions of the double odd nodes which are, respectively, the upper–left, upper–right, lower–left, and lower–right neighbors of the kth double even node. Let (i, j) denote the coordinates of the kth double even node. Then

$$k_{ul} = k \tag{3.52}$$

$$k_{ur} = \frac{j \bmod n_c + 2}{2} + \frac{n_c}{2}\frac{i-1}{2} \tag{3.53}$$

$$k_{ll} = \frac{j}{2} + \frac{n_c}{2}\frac{i \bmod n_r}{2} \tag{3.54}$$

$$k_{lr} = \frac{j \bmod n_c + 2}{2} + \frac{n_c}{2}\frac{i \bmod n_r}{2} \tag{3.55}$$

The intermediate vector c is a permutation of b. The permutation matrix P_e which satisfies $c = P_e b$ is given by the following algorithm:

for $i = \{2, 4, 6, \ldots, n_r\}$

  for $j = \{2, 4, 6, \ldots, n_c\}$

    let $k = \frac{j}{2} + \frac{n_c}{2}\frac{i-2}{2}$

    let $k_{ul} = k$

    let $k_{ur} = \frac{j \bmod n_c+2}{2} + \frac{n_c}{2}\frac{i-2}{2}$

let $k_{ll} = \frac{j}{2} + \frac{n_c}{2}\frac{i \bmod n_r}{2}$

let $k_{lr} = \frac{j \bmod n_c+2}{2} + \frac{n_c}{2}\frac{i \bmod n_r}{2}$

let the $(4(k-1)+1)$th column of P_e be $e_{4(k_{ul}-1)+4}$

let the $(4(k-1)+2)$th column of P_e be $e_{4(k_{ur}-1)+3}$

let the $(4(k-1)+3)$th column of P_e be $e_{4(k_{ll}-1)+2}$

let the $(4(k-1)+4)$th column of P_e be $e_{4(l_{lr}-1)+1}$

end j loop

end i loop

Next, using its local microprocessor, each double even node k computes

$$\begin{bmatrix} d_{4(k-1)+1} \\ d_{4(k-1)+2} \\ d_{4(k-1)+3} \\ d_{4(k-1)+4} \end{bmatrix} = \theta_k^2 \begin{bmatrix} c_{4(k-1)+1} \\ c_{4(k-1)+2} \\ c_{4(k-1)+3} \\ c_{4(k-1)+4} \end{bmatrix}, \tag{3.56}$$

where $\theta_k^2 \in SO(4)$ for each k. The intermediate vector d is the output of the second level and can be written

$$d = \Theta_2 P_e \Theta_1 P_1 y, \tag{3.57}$$

where

$$\Theta_2 = \begin{bmatrix} \theta_1^2 & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \theta_2^2 & \cdots & \mathbf{0} \\ \vdots & & \ddots & \vdots \\ \mathbf{0} & \cdots & \mathbf{0} & \theta_{n_c n_r/4}^2 \end{bmatrix}. \tag{3.58}$$

A third level can be computed in a similar manner. Here, the double even nodes pass their data back to the double odd nodes which perform the 4D rotations. The

output of the third level is the intermediate data vector

$$f = \Theta_3 P_o \Theta_2 P_e \Theta_1 P_1, \tag{3.59}$$

where the permutation matrix P_o is given by the following algorithm:

for $i = \{1, 3, 5, \ldots, n_r - 1\}$

  for $j = \{1, 3, 5, \ldots, n_c - 1\}$

    let $k = \frac{j+1}{2} + \frac{n_c}{2}\frac{i-1}{2}$

    let $k_{ul} = \frac{(j-2) \bmod n_c+1}{2} + \frac{n_c}{2}\frac{(i-2) \bmod n_r-1}{2}$

    let $k_{ur} = \frac{j+1}{2} + \frac{n_c}{2}\frac{(i-2) \bmod n_r-1}{2}$

    let $k_{ll} = \frac{(j-2) \bmod n_c+1}{2} + \frac{n_c}{2}\frac{i-1}{2}$

    let $k_{lr} = k$

    let the $(4(k-1)+1)$th column of P_e be $e_{4(k_{ul}-1)+4}$

    let the $(4(k-1)+2)$th column of P_e be $e_{4(k_{ur}-1)+3}$

    let the $(4(k-1)+3)$th column of P_e be $e_{4(k_{ll}-1)+2}$

    let the $(4(k-1)+4)$th column of P_e be $e_{4(l_{lr}-1)+1}$

  end j loop

end i loop

The communication patterns for the first 3 levels are depicted in Figure 3.3. The permutation matrices P_1, P_e, and P_o give us a mathematical representation of these communication patterns.

Levels 2 and 3 can be repeated for L levels, some $L > 0$, and $\widetilde{y}$ is defined to be the output of the Lth level,

$$\widetilde{y} = \Theta_L P_L \cdots \Theta_2 P_2 \Theta_1 P_1 y, \tag{3.60}$$

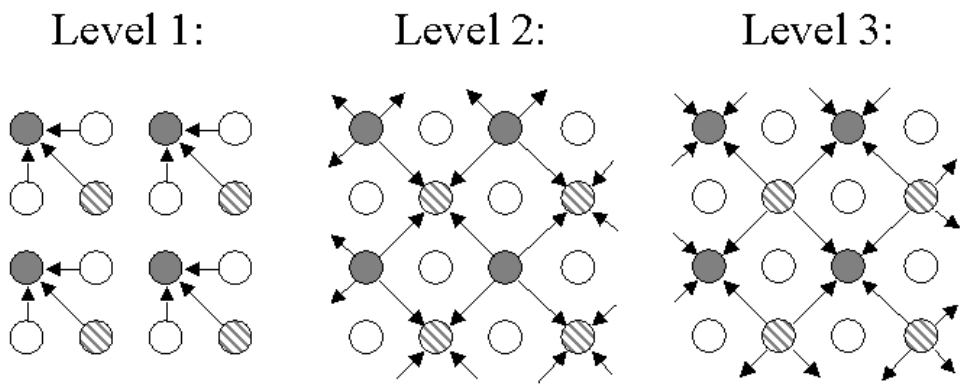


Figure 3.3: Communication patterns for first 3 levels of parallel output transformation on a 2–D array of smart sensors.

where, for $i = 2, 3, \ldots, L$,

$$P_i = \begin{cases} P_o \text{ for } i \text{ odd} \\ P_e \text{ for } i \text{ even} \end{cases} \tag{3.61}$$

Hence, we have constructed a transform of the form

$$Q_1 = \Theta_L P_L \cdots \Theta_2 P_2 \Theta_1 P_1 \tag{3.62}$$

which is easily implemented in a distributed manner on a control network.

Transforming the Input Vector

Here we consider the problem of transforming the input vector on a 2–D array. We begin by assuming that the input transformation Q_2 is of the same form as Q_1 in the previous section, i.e.

$$Q_2 = \Phi_L P_L \cdots \Phi_2 P_2 \Phi_1 P_1, \tag{3.63}$$

where the Φ_is are of the same form as the Θ_is in Q_1. The number of levels in Q_2 is equal to L, which is the same as the number of levels in Q_1. We also assume that L is odd. The task of this section is to compute the corresponding input transformation $u = Q_2^{-1}\widetilde{u}$ on the network. As in the case of the 1–D array, Q_2^{-1} can be implemented on the 2–D array by reversing the steps used in the input transformation.

To begin, recall that each element of the desired transformed input vector $\widetilde{u}$ resides on the same node as the corresponding element of the transformed output vector $\widetilde{y}$. As a result, $\widetilde{u}$ is distributed over the network with each double odd node containing 4 elements of $\widetilde{u}$ (here we have assumed that L is odd). In the first level, the intermediate data vector $a = \Phi_L^T\widetilde{u}$ is computed on the double odd nodes. Since Φ_L^T is a block diagonal matrix, a can be computed in a decentralized manner. Each double odd node then passes one element of a to each of its diagonal neighbors (the double even nodes), implementing

$$\begin{aligned} b &= P_e a && (3.64)\\ &= P_o^T a && (3.65)\\ &= P_L^T \Phi_L^T \widetilde{u}. && (3.66) \end{aligned}$$

The double even nodes then use their local processors to compute

$$\begin{aligned} c &= \Phi_{L-1}^T b && (3.67)\\ &= \Phi_{L-1}^T P_L^T \Phi_L^T \widetilde{u}. && (3.68) \end{aligned}$$

This process is repeated until the desired input

$$u = P_1^T \Phi_1^T P_2^T \Phi_2^T \cdots P_L^T \Phi_L^T \widetilde{u} \qquad (3.69)$$

is achieved. The final permutation step P_1^T sends the elements of u to the nodes on which they will be applied.

3.3 Recursive Orthogonal Transforms

In the previous section we discussed two possible methods of performing data transformations on the input and output vector of an array of smart sensors and actuators. Here we provide some insight into our choices for these transforms. We then generalize this notion to define what we call the recursive orthogonal transform (ROT).

The output transforms from the previous section are all of the form

$$Q = \Theta_L P_L \cdots \Theta_2 P_2 \Theta_1 P_1 \tag{3.70}$$

where, for $i = 1, 2, \ldots, L$, the Θ_is are block diagonal matrices with orthogonal blocks on the diagonal and the P_is are permutations of the identity matrix. The Θ_i represent decoupled, local rotations of the data vector performed in a distributed fashion on the control network. The permutation matrices represent a "shuffling" of the data vector resulting from the exchange of information on the network.

The wavelet packet transforms discussed in the previous chapter can be written in the form of Equation 3.70. To see this connection, consider the Haar wavelet transform, Q, operating on a 4–dimensional vector y. It is easy to verify that the wavelet transform can be written as the following matrix sequence of matrix operations:

$$Qy = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 & 0 \\ -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 & 0 \\ 0 & 0 & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 & 0 \\ 0 & 0 & -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix} y. \tag{3.71}$$

Employing a permutation matrix, this expression becomes

$$Qy = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 & 0 \\ -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 & 0 \\ -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 & 0 \\ 0 & 0 & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ 0 & 0 & -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix} y. \quad (3.72)$$

The submatrix

$$\begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix} \quad (3.73)$$

is orthogonal; it is a rotation in the plane by -45 degrees. Hence, the wavelet transform is a special case of a transform of the form given by Equation 3.70. The same is true for any transform in the wavelet packet.

Another connection to the this idea is the Euler angle representation of $SO(3)$, which can be thought of as the group of orientations of a rigid body in 3–dimensional Euclidean space [30]. The ZYZ Euler angles tell us that any orientation can be achieved by a rotation about the z axis by an angle α, followed by a rotation about the new y axis by an angle β, followed by a rotation about the z axis by an angle γ. The first rotation is represented by the matrix

$$R_z(\alpha) = \begin{bmatrix} cos(\alpha) & -sin(\alpha) & 0 \\ sin(\alpha) & cos(\alpha) & 0 \\ 0 & 0 & 1 \end{bmatrix}. \quad (3.74)$$

This matrix is block diagonal, and the blocks on the diagonal are orthogonal. The y axis rotation can be written

$$R_y(\beta) = \begin{bmatrix} cos(\beta) & 0 & -sin(\beta) \\ 0 & 1 & 0 \\ sin(\alpha) & 0 & cos(\alpha) \end{bmatrix}$$

$$
\begin{aligned}
&= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} cos(\beta) & -sin(\beta) & 0 \\ sin(\alpha) & cos(\alpha) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \\
&\triangleq P_3 R_z(\beta) P_2. \qquad (3.75)
\end{aligned}
$$

Now any $\Theta \in SO(3)$ can be represented by the composition of the three Euler rotations, i.e.

$$
\begin{aligned}
\Theta &= R_z(\gamma) R_y(\beta) R_z(\alpha) \\
&= R_z(\gamma) P_3 R_z(\beta) P_2 R_z(\alpha) \qquad (3.76)
\end{aligned}
$$

for some $\alpha, \beta, \gamma \in [0, 2\pi)$. Hence, we can represent any $\Theta \in SO(3)$ as a product of block diagonal matrices and permutations of the identity.

Our idea is to extend the concept of Euler angles to higher dimensional groups. This leads us to define the recursive orthogonal transform.

Definition 3.3.1 *A* recursive orthogonal transform (ROT) *is a linear operator of the form*

$$
\widetilde{\Theta} = P_1 \Theta_1 P_2 \Theta_2 \cdots P_L \Theta_L, \qquad (3.77)
$$

for some L, where for each $i = 1, 2, \ldots, L$,

1. *The matrix Θ_i is of the form*

$$
\Theta_i = \begin{bmatrix} \theta_1^i & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \theta_2^i & \cdots & \mathbf{0} \\ \vdots & & \ddots & \vdots \\ \mathbf{0} & \cdots & \mathbf{0} & \theta_{m_i}^i \end{bmatrix}, \qquad (3.78)
$$

$\theta_j^i \in SO(n_{ij})$ *(or $\theta_j^i \in U(n_{ij})$),*

$$
\sum_{j=1}^{m_i} n_{ij} = n, \qquad (3.79)
$$

and $\mathbf{0}$*s represent appropriately dimensioned blocks of zeros.*

2. *The matrix* P_i *is a permutation of the* $n \times n$ *identity matrix.*

The integer L is called the *level* of the ROT. The structure of the ROT is defined by L, P_i, m_i, and n_{ij} for $i = 1, 2, \ldots, L$, $j = 1, 2, \ldots, m_i$. These quantities are called the ROT *parameters*. The Θ_i, $i = 1, 2, \ldots, L$, are called the ROT *variables*.

In the remainder of this thesis, we will assume that the ROT parameters are given and fixed. We will address the problem of finding the ROT variables so that the resulting ROT most closely diagonalizes a given plant matrix.

Chapter 4

Symmetric Matrix Diagonalization and SVD using ROT

In Chapter 1 we discussed the possibility of applying linear data transformations to the input and output vectors of a MIMO system for the purpose of "diagonalizing" the plant matrix. Such transformations would make the task of designing and applying a feedback controller much easier because the MIMO system is reduced to a set of decoupled SISO systems. This is particularly advantageous for systems with large numbers of inputs and outputs. In the last chapter we introduced recursive orthogonal transforms, a class of linear operators which are designed to take advantage of the parallel processing capability of a control network while eliminating the need for global communication. Here we begin to bring these two ideas together by showing how to find ROTs which most closely diagonalize a constant matrix. The problem of finding the appropriate ROT variables is solved using a gradient flow which turns out to be closely related to the so called double

bracket equation introduced by Brockett [10] and discussed in detail by Helmke and Moore [22]. This is no coincidence since our work is heavily motivated by these authors.

Here we assume that the ROT parameters are given and fixed. In Section 4.1 we show how find the ROT variables so that the resulting ROT most nearly diagonalizes a given real–valued symmetric matrix. Section 4.2 describes how to find ROT variables to approximate the SVD factors of a given complex–valued matrix.

4.1 Diagonalization of Symmetric Matrices

The objective of this section is stated as follows: Given a symmetric $n \times n$ matrix H_0 and configuration parameters for the ROT

$$\widetilde{\Theta} = P_1 \Theta_1 P_2 \Theta_2 \cdots P_L \Theta_L, \tag{4.1}$$

find $\Theta_1, \Theta_2, \ldots, \Theta_L$ such that the matrix

$$H = \widetilde{\Theta}^T H_0 \widetilde{\Theta} \tag{4.2}$$

is, in some sense, most closely diagonalized. Our first step toward solving this problem is to find a "diagonalness" functional $\phi(H)$. Then we search for the $(\Theta_1, \Theta_2, \ldots, \Theta_L)$ which minimize ϕ by flowing along the gradient vector field $\nabla\phi$ on the configuation space of the Θ_i's. This idea is motivated by Brockett [10], who showed that the matrix diagonalization problem can be solved using by solving a matrix ODE which evolves on the group of orthogonal matrices. As a result, we begin this discussion with a review his work. We then return to our problem and construct a gradient flow designed to find the appropriate ROT variables.

4.1.1 Matrix Diagonalization via Flows on $SO(n)$

In 1988, Brockett showed that a number of interesting problems in linear algebra and computer science could be solved by the use of dynamical systems defined on matrix groups[10]. Following [22], we review his work on the diagonalization of symmetric matrices.

Finding the Gradient Vector Field

Let H_0 be a real valued, symmetric, $n \times n$ matrix. The goal is to find Θ in $SO(n)$ such that $H = \Theta^T H_0 \Theta$ is diagonal. Note that H is symmetric for every $\Theta \in SO(n)$ since

$$\begin{aligned} H^T &= \Theta^T H_0^T \Theta \\ &= \Theta^T H_0 \Theta \\ &= H. \end{aligned} \tag{4.3}$$

Define

$$J(\Theta) = \left\| N - \Theta^T H_0 \Theta \right\|^2, \tag{4.4}$$

where N is a diagonal matrix with distinct values on the diagonal and $\|\cdot\|$ is the Frobenius norm, $\|A\|^2 = \operatorname{tr}\left(A^T A\right)$. Hence, J is the distance between $\Theta^T H_0 \Theta$ and a fixed diagonal matrix. Intuitively, minimizing J makes $H = \Theta^T H_0 \Theta$ look as much like N as possible. Since N is diagonal, the Θ which minimizes J should diagonalize H. Simple matrix manipulation reveals

$$\left\| N - \Theta^T H_0 \Theta \right\|^2 = \|N\|^2 + \left\| \Theta^T H_0 \Theta \right\|^2 - 2\operatorname{tr}(NH). \tag{4.5}$$

The matrix N is constant, so $\|N\|^2$ is constant. The matrix H_0 is constant and Θ is orthogonal, so $\left\| \Theta^T H_0 \Theta \right\|^2 = \|H_0\|^2$ is also constant. As a result, the problem of

minimizing $J(\Theta)$ can be posed as the problem of maximizing the functional

$$\phi(\Theta) = \operatorname{tr}\left(N\Theta^T H_0 \Theta\right). \tag{4.6}$$

Brockett's idea is to search for the maximizing Θ by flowing along the gradient vector field of ϕ on $SO(n)$. Recall that the gradient vector field of ϕ on $SO(n)$ satisfies the following two properties:

1. $\nabla\phi(\Theta) \in T_\Theta SO(n)$ for all $\Theta \in SO(n)$.

2. $D\phi_\Theta(\xi) = \langle \nabla\phi(\Theta), \xi \rangle$ for all $\xi \in T_\Theta SO(n)$.

As a result of the first property, $\nabla\phi(\Theta)$ must be of the form $\Theta\Omega$, where $\Omega \in \mathfrak{so}(n)$. This means that $\nabla\phi(\Theta)$ can be written in the form

$$\nabla\phi(\Theta) = \Theta\Omega^{\nabla\phi(\Theta)}, \tag{4.7}$$

where $\Omega^{\nabla\phi(\Theta)}$ is skew–symmetric for each $\Theta \in SO(n)$. Combining this with the second property gives

$$\begin{aligned} D\phi_\Theta(\Theta\Omega) &= \left\langle \Theta\Omega^{\nabla\phi(\Theta)}, \Theta\Omega \right\rangle \\ &= -\operatorname{tr}\left(\Omega^{\nabla\phi(\Theta)}\Omega\right). \end{aligned} \tag{4.8}$$

Taking the directional derivative of ϕ,

$$D\phi_\Theta(\Theta\Omega) = \operatorname{tr}\left(N\Omega^T\Theta^T H_0\Theta + N\Theta^T H_0 \Theta\Omega\right). \tag{4.9}$$

Using the skew–symmetry of Ω and the fact that $\operatorname{tr}(AB) = \operatorname{tr}(BA)$ we get

$$\begin{aligned} D\phi_\Theta(\Theta\Omega) &= \operatorname{tr}\left(\left(N\Theta^T H_0\Theta - \Theta^T H_0 \Theta N\right)\Omega\right) \\ &= \operatorname{tr}\left(\left[N, \Theta^T H_0 \Theta\right]\Omega\right). \end{aligned} \tag{4.10}$$

Substituting this into 4.8 allows us to restate the second condition as

$$\operatorname{tr}\left(\left[N, \Theta^T H_0\Theta\right]\Omega\right) = -\operatorname{tr}\left(\Omega^{\nabla\phi(\Theta)}\Omega\right). \tag{4.11}$$

So the choice $\Omega^{\nabla\phi(\Theta)} = -\left[N, \Theta^T H_0 \Theta\right]$ satisfies the second condition. This is a valid choice because the matrix $\left[N, \Theta^T H_0 \Theta\right]$ is skew–symmetric, which is easily seen as follows:

$$\begin{aligned} [N, H]^T &= (NH - HN)^T \\ &= H^T N^T - N^T H^T \\ &= HN - NH \\ &= -[N, H]. \end{aligned} \tag{4.12}$$

As a result, the gradient vector field is given by the expression

$$\nabla\phi(\Theta) = -\Theta\left[N, \Theta^T H_0 \Theta\right]. \tag{4.13}$$

The ascent direction gradient flow is then given by the ODE

$$\dot{\Theta} = -\Theta\left[N, \Theta^T H_0 \Theta\right], \quad \Theta(0) = \Theta_0. \tag{4.14}$$

Convergence Properties

From the properties of a gradient flow on a compact manifold listed in Section 3.1.1 it is clear that the solution of Equation 4.14 exists for all time $t > 0$ and converges to an equilibrium point for every $\Theta_0 \in SO(n)$. Let Θ_* be such an equilibrium point and define $H \triangleq \Theta_*^T H_0 \Theta_*$. Since Θ_* is nonsingular, the matrix $[N, H]$ must be zero. The (i, j)th element of $[N, H]$ is

$$\begin{aligned} [N, H]_{ij} &= (NH)_{ij} - (HN)_{ij} \\ &= \sum_{k=1}^{n} N_{ik} H_{kj} - \sum_{k=1}^{n} H_{ik} N_{kj}. \end{aligned} \tag{4.15}$$

The matrix N is diagonal so $[N]_{ik} = 0$ for $i \neq k$. Let n_i denote the ith diagonal element of N and let h_{ij} denote the (i, j)th element of H. Then

$$[N, H]_{ij} = n_i h_{ij} - h_{ij} n_j$$

$$= \quad h_{ij}\left(n_i - n_j\right). \tag{4.16}$$

The diagonal elements of N are distinct, so for $i \neq j$, $[N, H]_{ij} = 0$ only if $h_{ij} = 0$. Therefore the matrix H must be diagonal.

4.1.2 Matrix Diagonalizing Flows for ROTs

Here we extend the results of Brockett [10] to find the recursive orthogonal transform $\widetilde{\Theta}$ which most closely diagonalizes a real–valued symmetric matrix.

Preliminaries and Definitions

Let H_0 be a real valued, symmetric, $n \times n$ matrix. This is the matrix we seek to diagonalize.

Let

$$\Theta_k = \begin{bmatrix} \theta_1^k & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \theta_2^k & \cdots & \mathbf{0} \\ \vdots & & \ddots & \vdots \\ \mathbf{0} & \cdots & \mathbf{0} & \theta_{m_k}^k \end{bmatrix}, \tag{4.17}$$

for $k = 1, 2, \ldots, L$, some $L > 0$, where $\theta_j^k \in SO(n_{kj})$,

$$\sum_{j=1}^{m_k} n_{kj} = n \quad \forall k = 1, 2, \ldots, L, \tag{4.18}$$

and the $\mathbf{0}$s represent appropriately dimensioned blocks of zeros. In simple language, each Θ_k is a block diagonal matrix where the blocks on the diagonal are orthogonal matrices. Let $M_k = SO(n_{k1}) \times SO(n_{k2}) \times \cdots \times SO(n_{km_k})$. Then Θ_k belongs to the Lie subgroup $M_k \subset SO(n)$.

Recall that the tangent space of $SO(\ell)$ at the identity is the Lie algebra

$$\mathfrak{so}(\ell) = \{\Omega \in \mathbb{R}^{\ell \times \ell} | \Omega^T = -\Omega\}. \tag{4.19}$$

The Lie algebra of M_k is then

$$T_e M_k = \mathfrak{so}(n_{k1}) \oplus \mathfrak{so}(n_{k2}) \oplus \cdots \oplus \mathfrak{so}(n_{km_k}). \tag{4.20}$$

A vector in $T_e M_k$ can then be written

$$\Omega_k = \begin{bmatrix} \omega_1^k & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \omega_2^k & \cdots & \mathbf{0} \\ \vdots & & \ddots & \vdots \\ \mathbf{0} & \cdots & \mathbf{0} & \omega_{m_k}^k \end{bmatrix}. \tag{4.21}$$

where $\omega_j^k \in \mathfrak{so}(n_{kj})$ for $j = 1, 2, \ldots, m_k$. Hence the tangent space to M_k at the point Θ_k is

$$T_{\Theta_k} M_k = \{\Theta_k \Omega_k \,|\Omega_k \in T_e M_k\}. \tag{4.22}$$

Define $\Psi \in M = M_1 \times M_2 \times \cdots \times M_L$ to be the ordered L–tuple of Θ_ks,

$$\Psi = (\Theta_1, \Theta_2, \ldots, \Theta_L). \tag{4.23}$$

Likewise, let X denote a vector in $T_\Psi M$,

$$X = (\Theta_1 \Omega_1, \Theta_2 \Omega_2, \ldots, \Theta_L \Omega_L). \tag{4.24}$$

For $k = 1, 2, 3, \ldots, L$, P_k is a fixed permutation of the $n \times n$ identity matrix. Each P_k is an element of the group $O(n)$.

Finally, for $k = 1, 2, \ldots, L$, we define two sequences of recursive orthogonal transforms:

$$\widetilde{\Theta}_k = \prod_{\ell=1}^{k} P_\ell \Theta_\ell, \tag{4.25}$$

$$\breve{\Theta}_k = \prod_{\ell=k+1}^{n} P_\ell \Theta_\ell. \tag{4.26}$$

Here the product symbol $\prod$ denotes multiplication with indices ascending from left to right. Also, $\prod_{k=a}^{b} = \mathbb{1}$ for $b < a$. We call $\widetilde{\Theta}_k$ the kth lower subtransform and $\breve{\Theta}_k$ the kth upper subtransform. The ROT $\widetilde{\Theta} = \widetilde{\Theta}_L$ is referred to as the complete transform.

Constructing the Gradient Flow

The objective of this section is to approximately diagonalize the symmetric matrix H_0. This is done by finding $\Psi = (\Theta_1, \Theta_2, \ldots, \Theta_L)$ such that the matrix

$$\begin{aligned} H(\Psi) &= \Theta_L^T P_L^T \cdots \Theta_2^T P_2^T \Theta_1^T P_1^T H_0 P_1 \Theta_1 P_2 \Theta_2 \cdots P_L \Theta_L \\ &= \widetilde{\Theta}^T H_0 \widetilde{\Theta} \end{aligned} \tag{4.27}$$

is "more diagonal" than it is for any other choice of Ψ.

Towards this goal, let N be a fixed diagonal $n \times n$ matrix. We can now define a cost function using distance between H and N given by the Frobenius norm. Simple matrix manipulation reveals

$$\|N - H(\Psi)\|^2 = \|N\|^2 + \|H(\Psi)\|^2 - 2\mathrm{tr}\left(NH(\Psi)\right). \tag{4.28}$$

Since N is fixed, $\|N\|^2$ is constant. Since H_0 is fixed and the matrix $\widetilde{\Theta}$ is orthogonal, $\|H\|^2 = \left\|\widetilde{\Theta}^T H_0 \widetilde{\Theta}\right\|^2$ is also constant. The Frobenius distance between H and N is minimized when

$$\phi(\Psi) = \mathrm{tr}\left(NH(\Psi)\right) \tag{4.29}$$

is maximized. The function $\phi : M \to \mathbb{R}$ can be thought of as a "diagonalness" functional.

The idea here is to search for the Ψ which maximizes ϕ by flowing along the gradient vector field of ϕ on the manifold M. Recall that $\nabla\phi(\Psi)$ on M is defined by the following properties:

1. $\nabla\phi(\Psi) \in T_\Psi M \quad \forall \Psi \in M.$

2. $D\phi_\Psi(X) = \langle \nabla\phi(\Psi), X\rangle \quad \forall X \in T_\Psi M.$

The first property states that $\nabla\phi(\Psi)$ in contained in the tangent space of M at the point Ψ. This means that $\nabla\phi(\Psi)$ must be of the form

$$\nabla\phi(\Psi) = \left(\Theta_1\Omega_1^{\nabla\phi}, \ldots, \Theta_L\Omega_L^{\nabla\phi}\right), \tag{4.30}$$

where $\Omega_k^{\nabla\phi} \in T_e M_k,\ k = 1, 2, \ldots, L$. For convenience of notation, we have surpressed the fact that $\Omega_k^{\nabla\phi}$ depends on Ψ.

From the second property, we must have

$$\begin{aligned} D\phi_\Psi((\Theta_1\Omega_1, \ldots, \Theta_L\Omega_L)) &= \langle \nabla\phi(\Theta_1, \ldots, \Theta_L), (\Theta_1\Omega_1, \ldots, \Theta_L\Omega_L)\rangle \\ &= \left\langle (\Theta_1\Omega_1^{\nabla\phi}, \ldots, \Theta_L\Omega_L^{\nabla\phi}), (\Theta_1\Omega_1, \ldots, \Theta_L\Omega_L)\right\rangle \\ &= \sum_{k=1}^{L} \operatorname{tr}\left((\Omega_k^{\nabla\phi})^T\Omega_k\right). \end{aligned} \tag{4.31}$$

Finding an expression for $D\phi_\Psi(X)$:

$$\begin{aligned} D\phi_\Psi(X) &= \operatorname{tr}\left(\sum_{k=1}^{L} N\left(\left(\prod_{\ell=1}^{k-1} P_\ell\Theta_\ell\right) P_k\Theta_k\Omega_k \left(\prod_{\ell=k+1}^{n} P_\ell\Theta_\ell\right)\right)^T H_0 \left(\prod_{\ell=1}^{n} P_\ell\Theta_\ell\right)\right. \\ &\qquad \left. + N\left(\prod_{\ell=1}^{n} P_\ell\Theta_\ell\right)^T H_0 \left(\prod_{\ell=1}^{k-1} P_\ell\Theta_\ell\right) P_k\Theta_k\Omega_k \left(\prod_{\ell=k+1}^{n} P_\ell\Theta_\ell\right)\right) \\ &= \operatorname{tr}\left(\sum_{k=1}^{L} N\left(\widetilde{\Theta}_k\Omega_k\breve{\Theta}_k\right)^T H_0\widetilde{\Theta} + N\widetilde{\Theta}^T H_0\widetilde{\Theta}_k\Omega_k\breve{\Theta}_k\right) \\ &= \operatorname{tr}\left(\sum_{k=1}^{L} N\breve{\Theta}_k^T\Omega_k^T\widetilde{\Theta}_k^T H_0\widetilde{\Theta} + N\widetilde{\Theta}^T H_0\widetilde{\Theta}_k\Omega_k\breve{\Theta}_k\right). \end{aligned} \tag{4.32}$$

Using $\operatorname{tr}(AB) = \operatorname{tr}(BA)$ and the skew–symmetry of Ω_k we get

$$D\phi_\Psi(X) = \operatorname{tr}\left(\sum_{k=1}^{L}\left(\breve{\Theta}_k N\widetilde{\Theta}^T H_0\widetilde{\Theta}_k - \widetilde{\Theta}_k^T H_0\widetilde{\Theta} N\breve{\Theta}_k^T\right)\Omega_k\right). \tag{4.33}$$

Using the fact that $\widetilde{\Theta} = \widetilde{\Theta}_k \breve{\Theta}_k$ for any $k = 1, 2, \ldots, L$, we get

$$\begin{aligned} D\phi_\Psi(X) &= \operatorname{tr}\left(\sum_{k=1}^{L}\left(\breve{\Theta}_k N \breve{\Theta}_k^T \widetilde{\Theta}_k^T H_0 \widetilde{\Theta}_k - \widetilde{\Theta}_k^T H_0 \widetilde{\Theta}_k \breve{\Theta}_k N \breve{\Theta}_k^T\right)\Omega_k\right) \\ &= \operatorname{tr}\left(\sum_{k=1}^{L}\left[\breve{\Theta}_k N \breve{\Theta}_k^T, \widetilde{\Theta}_k^T H_0 \widetilde{\Theta}_k\right]\Omega_k\right). \end{aligned} \tag{4.34}$$

For $k = 1, 2, \ldots, L$, define

$$H^k = \widetilde{\Theta}_k^T H_0 \widetilde{\Theta}_k \tag{4.35}$$

$$N^k = \breve{\Theta}_k N \breve{\Theta}_k^T. \tag{4.36}$$

The matrices H^k and N^k are symmetric for each k. This means that the bracket $\left[N^k, H^k\right] \in T_e O(n)$. In order to get a valid gradient flow, each Ω_k in Equation 4.34 must be multiplied on the left by $(\Omega_k^{\nabla\phi})^T$, where $\Omega_k^{\nabla\phi} \in T_e(M_k)$. To accomplish this, we introduce a set of natural projection operators, $\Pi_k : T_e O(n) \to T_e(M_k)$. Π_k projects from the set of skew–symmetric matrices to the set of block diagonal skew–symmetric matrices by setting all off–diagonal blocks to zero.

Fact 4.1.1 *Let $A \in \mathfrak{o}(n)$ and let $\Omega_i \in T_e M_i$ as defined in Equation 4.21. Then*

$$tr\left((\Pi_i A)^T \Omega_i\right) = tr\left(A^T \Omega_i\right). \tag{4.37}$$

<u>Proof:</u>

Looking at Equation 4.21 we see that the (j,k)th block of Ω_i is

$$[\Omega_i]_{jk} = \omega_{ij}\delta_k^j, \tag{4.38}$$

where δ_k^j is the Kronecker symbol. Partition A into blocks $\mathbf{a}_{jk}$ so that the block structure of A is compatible with that of Ω_i. Then

$$\operatorname{tr}\left(A^T \Omega_i\right) = \sum_{j=1}^{m_i}\sum_{k=1}^{m_i} \operatorname{tr}\left(\mathbf{a}_{jk}[\Omega_i]_{jk}\right) \tag{4.39}$$

$$= \sum_{j=1}^{m_i}\sum_{k=1}^{m_i} \operatorname{tr}\left(\mathbf{a}_{jk}\omega_{ij}\delta_k^j\right) \tag{4.40}$$

$$= \sum_{k=1}^{m_i} \operatorname{tr}\left(\mathbf{a}_{kk}\omega_{ik}\right) \tag{4.41}$$

$$= \operatorname{tr}\left((\Pi_i A)^T \Omega_i\right). \tag{4.42}$$

■

In light of this fact,

$$D\phi_\Psi(X) = \sum_{k=1}^{L} \operatorname{tr}\left((-\Pi_k\left[N^k, H^k\right])^T \Omega_i\right). \tag{4.43}$$

If we let $\Omega_k^{\nabla\phi} = -\Pi_k\left[N^k, H^k\right]$, then $\nabla\phi(\Psi)$ meets the conditions for a gradient vector field outlined above. This yields

$$\nabla\phi(\Psi) = \left(-\Theta_1\Pi_1\left[N^1, H^1\right], -\Theta_2\Pi_2\left[N^2, H^2\right], \ldots, -\Theta_L\Pi_L\left[N^L, H^L\right]\right). \tag{4.44}$$

The corresponding ascent direction gradient flow is then given by a system of coupled matrix ODEs. This system is written as

$$\dot{\Theta}_k = -\Theta_k\Pi_k\left[\breve{\Theta}_k N \breve{\Theta}_K^T, \widetilde{\Theta}_k^T H_0 \widetilde{\Theta}_k\right], \quad \Theta_k(0) = \Theta_{k0}, \tag{4.45}$$

for $k = 1, 2, \ldots, L$.

At this point it is natural to bring up the question of the convergence properties of this gradient flow. From the properties of gradient flows on compact manifolds listed in Section 3.1.1 we know that the solution to Equation 4.45 exists for all time and converges to an equilibrium point for every set of initial conditions. Except in a few special cases (see Appendix A) it is difficult to say more than this. The numerical results we have obtained are promising, however, as is demonstrated by the following example.

Example

Here we show the results of applying this technique to approximately diagonalize a 16×16 symmetric matrix with random entries. Here we used the ROT structure for a 1D array of sensor actuator pairs presented in Section 3.2.1. The original plant matrix is shown in Figure 4.1. The approximately diagonalized plant matrix for ROT levels 3,7,11, and 15 are shown in Figures 4.2, 4.3, 4.4, and 4.5, respectively. Figure 4.6 show a plot of approximation error as a function of the number of levels used in the approximating ROT. Here, the approximation error is defined to be

$$E(\widetilde{\Theta}) = \frac{\left\| Q^T G Q - \widetilde{\Theta}^T G \widetilde{\Theta} \right\|}{\|G\|}, \tag{4.46}$$

where Q is a matrix whose columns are the unit eigenvectors of G.

The dimension of $SO(n)$ is $n(n-1)/2$. The number of degrees of freedom in the ROTs being considered is $Ln/2$, where L is the level of the transform. Intuitively, when $L = n-1$ the ROT "should" have enough degrees of freedom to represent any Θ in $SO(n)$. These results seem to support this notion since the approximation error gets very close to zero for $L = 15$.

4.2 Singular Value Decomposition

In Section 4.1 we addressed the problem of finding a recursive orthogonal transform to diagonalize a real–valued symmetric matrix. In this section, we extend these results to find approximations of the SVD factors which diagonalize a general complex–valued matrix. In this case, given an $n \times m$ complex matrix H_0 and ROT configurations for $\widetilde{U} \in U(m)$,

$$\widetilde{U} = \prod_{i=1}^{L_U} P_i U_i, \tag{4.47}$$

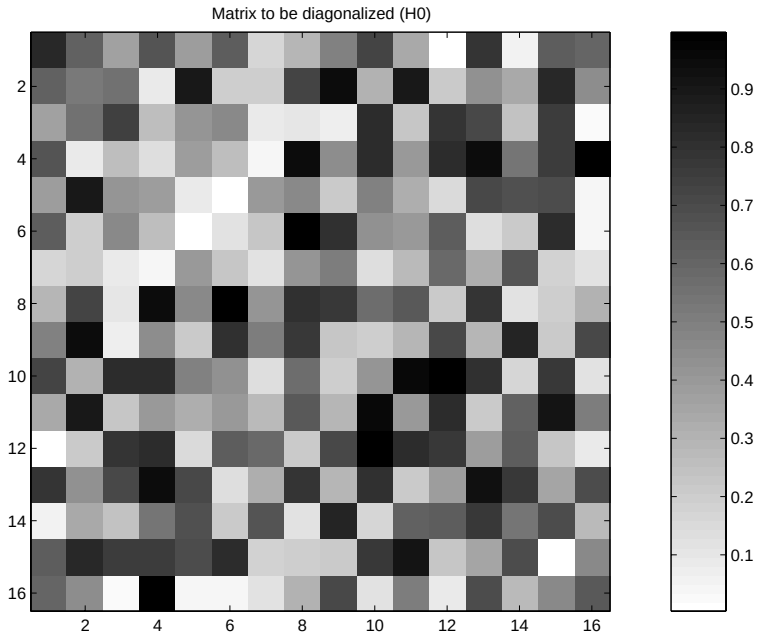


Figure 4.1: 16×16 random symmetric plant matrix.

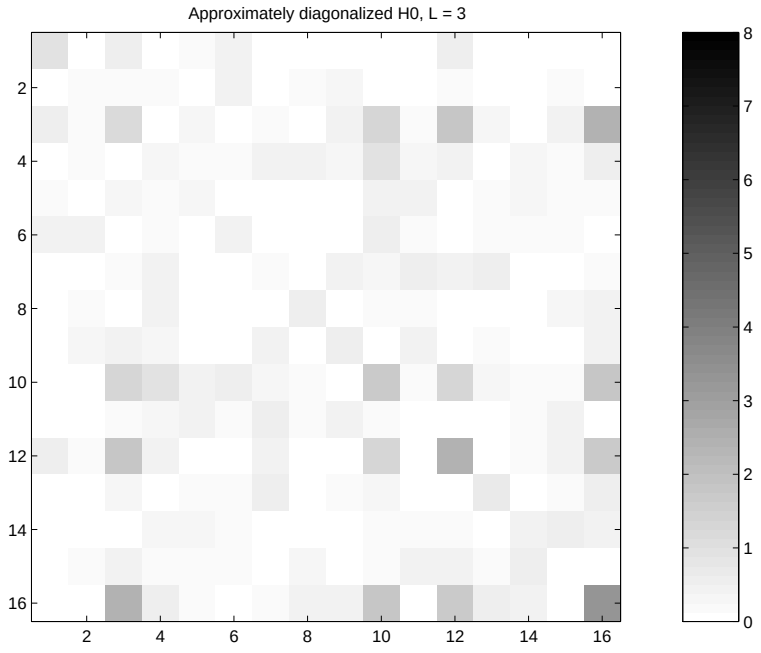


Figure 4.2: Approximately diagonalized plant matrix using 3 level ROT.

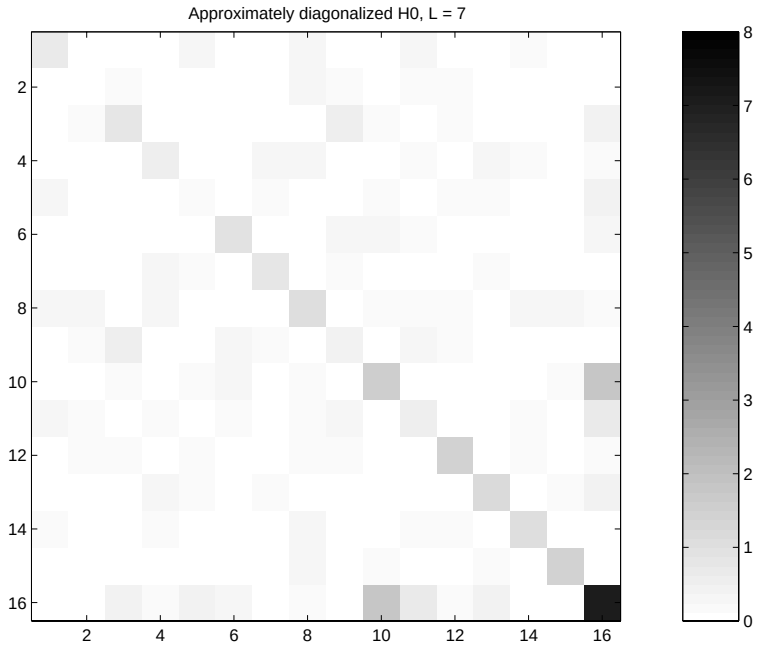


Figure 4.3: Approximately diagonalized plant matrix using 7 level ROT.

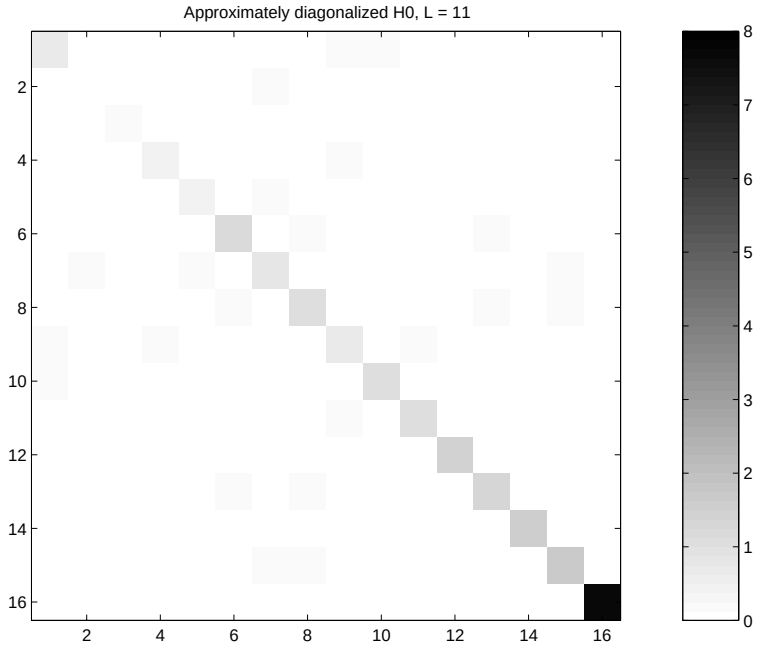


Figure 4.4: Approximately diagonalized plant matrix using 11 level ROT.

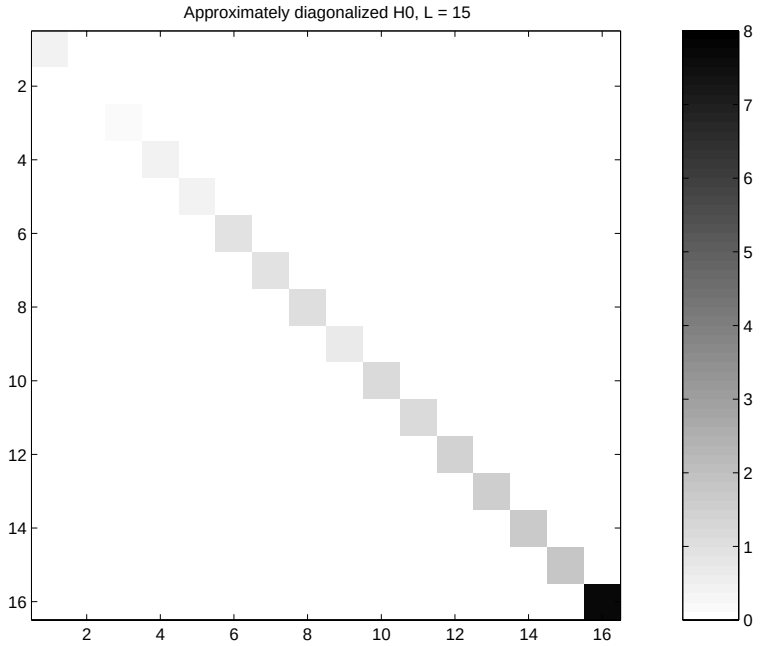


Figure 4.5: Approximately diagonalized plant matrix using 15 level ROT.

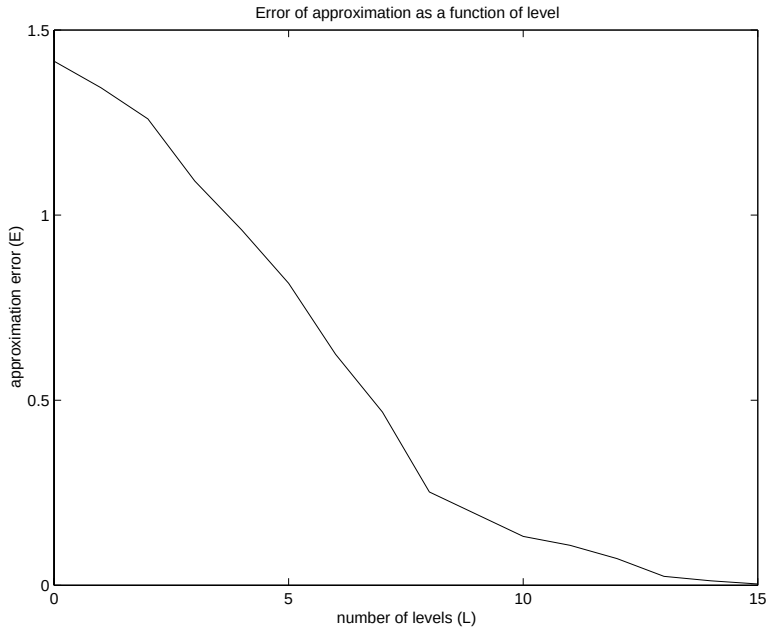


Figure 4.6: Plot of approximation error versus ROT level.

and $\widetilde{V} \in U(n)$,

$$\widetilde{V} = \prod_{i=1}^{L_V} Q_i V_i, \tag{4.48}$$

we will search for the block diagonal U_is and V_is so that the matrix $H \in \mathbb{C}^{n \times n}$,

$$H = \widetilde{V}^H H_0 \widetilde{U} \tag{4.49}$$

is most closely diagonalized. Recall that, according to our definition, a matrix A is diagonal if $[A]_{ij} = 0$ for all $i \neq j$. This definition applies to non–square matrices as well as square ones.

As in the case of diagonalizing symmetric matrices, our approach is to search for the best U_is and V_is by flowing along the gradient vector field of a "diagonalness" functional. Here we are motivated by Helmke and Moore [21], who extended Brockett's work on symmetric matrices to perform SVD via a gradient flow on $U(m) \times U(n)$. We begin this discussion with a review of their work. We then return to the problem of SVD using recursive orthogonal transforms.

4.2.1 SVD via Flows on $U(m) \times U(n)$

In 1992, Helmke and Moore [21] addressed the problem of finding the SVD factors of a complex–valued matrix by using gradient flows on the unitary group. This work is an extension to the work or Brockett [10], which we reviewed in Section 4.1.1. Here we present a brief summary of the work in [21].

Finding the Gradient Flows

Given $H_0, N \in \mathbb{C}^{n \times m}$, consider the task of finding $U \in U(m)$, $V \in U(n)$ to minimize the squared Frobenius distance

$$\left\| N - V^H H_0 U \right\|^2 = \|N\|^2 + \|H_0\|^2 - 2\mathrm{retr}\left(N^H V^H H_0 U\right). \tag{4.50}$$

N and H_0 are constant, so minimizing the above norm is equivalent to maximizing the functional

$$\phi(U,V) = 2\text{retr}\left(N^H V^H H_0 U\right). \tag{4.51}$$

If the matrix N is diagonal, then ϕ can be thought of as a "diagonalness" function for the matrix $V^H H_0 U$.

As in the case of Brockett's work, the idea is to search for the U and V which maximize ϕ by flowing along the gradient vector field of ϕ on the smooth Lie group $U(m) \times U(n)$.

The Lie algebra of $U(m) \times U(n)$ is appropriately written as the direct sum $\mathfrak{u}(m) \oplus \mathfrak{u}(n)$. Accordingly, a tangent vector at the point (U,V) is a vector of the form $(U\Omega, V\Lambda)$, where $\Omega \in \mathfrak{u}(m)$ and $\Lambda \in \mathfrak{u}(n)$.

Recall the properites of the gradient flow:

1. $\nabla\phi(U,V) \in T_{(U,V)}\left(U(m) \times U(n)\right)$ for all $(U,V) \in U(m) \times U(n)$.

2. $D\phi_{(U,V)}(U\Omega, V\Lambda) = \langle \nabla\phi(U,V), (U\Omega, V\Lambda)\rangle$ for all $(U\Omega, V\Lambda) \in T_{(U,V)}\left(U(m) \times U(n)\right)$.

The first property says that $\nabla\phi(U,V)$ is tangent to $U(m) \times U(n)$ at every point, which means that $\nabla\phi$ can be written

$$\nabla\phi(U,V) = \left(U\Omega^{\nabla\phi}, V\Lambda^{\nabla\phi}\right), \tag{4.52}$$

where $\left(\Omega^{\nabla\phi}, \Lambda^{\nabla\phi}\right) \in \mathfrak{u}(m) \oplus \mathfrak{u}(n)$. For convenience of notation, we have surpressed the fact that $\Omega^{\nabla\phi}$ and $\Lambda^{\nabla\phi}$ depend on (U,V). Writing $\nabla\phi$ in this way, the second property becomes

$$\begin{aligned} D\phi_{(U,V)}(U\Omega, V\Lambda) &= \left\langle \left(U\Omega^{\nabla\phi}, V\Lambda^{\nabla\phi}\right), (U\Omega, V\Lambda)\right\rangle \\ &= -\text{retr}\left(\Omega^{\nabla\phi}\Omega\right) - \text{retr}\left(\Lambda^{\nabla\phi}\Lambda\right). \end{aligned} \tag{4.53}$$

Finding an expression for $D\phi_{(U,V)}(U\Omega, V\Lambda)$:

$$\begin{aligned} D\phi_{(U,V)}(U\Omega, V\Lambda) &= 2\mathrm{retr}\left(N^H\Lambda^H V^H H_0 U + N^H V^H H_0 U\Omega\right) \\ &= 2\mathrm{retr}\left(N^H V^H H_0 U\Omega - V^H H_0 U N^H \Lambda\right) \end{aligned} \tag{4.54}$$

According to Fact 3.1.3, $N^H V^H H_0 U$ and $V^H H_0 U N^H$ can be replaced with their skew–hermetian components yielding

$$\begin{aligned} D\phi_{(U,V)}(U\Omega, V\Lambda) &= 2\mathrm{retr}\left(\frac{1}{2}\left(N^H V^H H_0 U - U^H H_0^H V N\right)\Omega \right. \\ &\qquad \left. -\frac{1}{2}\left(V^H H_0 U N^H - N U^H H_0^H V\right)\Lambda\right) \\ &= \mathrm{retr}\left(\left\{N, V^H H_0 U\right\}\Omega + \left\{N^H, U^H H_0^H V\right\}\Lambda\right), \end{aligned} \tag{4.55}$$

where

$$\begin{aligned} \{\cdot,\cdot\} : \mathbb{C}^{p\times q} \times \mathbb{C}^{p\times q} &\to \mathfrak{u}(q) \\ (A, B) &\mapsto A^H B - B^H A \end{aligned} \tag{4.56}$$

is the extended Lie bracket. Noting that $\left\{N, V^H H_0 U\right\} \in \mathfrak{u}(m)$ and $\left\{N^H, U^H H_0^H V\right\} \in \mathfrak{u}(n)$ and looking back at Equation 4.53, we see that both properties of the gradient vector field are satisfied for the choices

$$\Omega^{\nabla\phi} = -\left\{N, V^H H_0 U\right\} \tag{4.57}$$

$$\Lambda^{\nabla\phi} = -\left\{N^H, U^H H_0^H V\right\}. \tag{4.58}$$

This results in the gradient vector field

$$\nabla\phi(U, V) = \left(-U\left\{N, V^H H_0 U\right\}, -V\left\{N^H, U^H H_0^H V\right\}\right). \tag{4.59}$$

The resulting gradient flows are then given by the coupled pair of matrix ODEs evolving on $U(m) \times U(n)$. They are

$$\dot{U} = -\left\{N, V^H H_0 U\right\}, \quad U(0) = U_0 \in U(m) \tag{4.60}$$

$$\dot{V} = -\left\{N^H, U^H H_0^H V\right\}, \quad V(0) = V_0 \in U(n). \tag{4.61}$$

4.2.2 SVD Flows for ROTs

Here we extend the work of Helmke and Moore [21] to find the recursive orthogonal transforms $\widetilde{U}$ and $\widetilde{V}$ which most closely approximate the SVD factors of a given matrix H_0.

Preliminaries and Definitions

Let $H_0 \in \mathbb{C}^{n\times m}$ be the matrix to be decomposed. Let N be a fixed matrix in $\mathbb{C}^{n\times m}$.

Let

$$U_i = \begin{bmatrix} u_1^i & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & u_2^i & \cdots & \mathbf{0} \\ \vdots & & \ddots & \vdots \\ \mathbf{0} & \cdots & \mathbf{0} & u_{k_i}^i \end{bmatrix}, \tag{4.62}$$

for $i = 1, 2, \ldots, L_U$, some $L_U > 0$, where $u_j^i \in U(m_{ij})$,

$$\sum_{j=1}^{k_i} m_{ij} = m \quad \forall i = 1, 2, \ldots, L_U, \tag{4.63}$$

and the $\mathbf{0}$s represent appropriately dimensioned blocks of zeros. Let

$$M_i^U = U(m_{i1}) \times U(m_{i2}) \times \cdots \times U(m_{ik_i}). \tag{4.64}$$

Then U_i belongs to the smooth, connected Lie subgroup $M_i^U \subset U(m)$. The Lie algebra of M_i^U is then

$$T_e M_i^U = \mathfrak{u}(m_{i1}) \oplus \mathfrak{u}(m_{i2}) \oplus \cdots \oplus \mathfrak{u}(m_{ik_i}). \tag{4.65}$$

We choose to write and element of $T_e M_i^U$ as a block diagonal skew–hermetian with

the same block structure as U_i, i.e. $\Omega_i \in T_e M_i^U$ will be written

$$\Omega_i = \begin{bmatrix} \omega_1^i & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \omega_2^i & \cdots & \mathbf{0} \\ \vdots & & \ddots & \vdots \\ \mathbf{0} & \cdots & \mathbf{0} & \omega_{k_i}^i \end{bmatrix}, \tag{4.66}$$

where $\omega_j^i \in \mathfrak{u}(m_{ij})$ for $j = 1, 2, \ldots, k_i$. The tangent space of M_i^U at the point U_i is then

$$T_{U_i} M_i^U = \left\{ U_i \Omega_i \,\middle|\, \Omega_i \in T_e M_i^U \right\}. \tag{4.67}$$

We make a similar set of definitions for V_i. Let

$$V_i = \begin{bmatrix} v_1^i & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & v_2^i & \cdots & \mathbf{0} \\ \vdots & & \ddots & \vdots \\ \mathbf{0} & \cdots & \mathbf{0} & v_{\ell_i}^i \end{bmatrix}, \tag{4.68}$$

for $i = 1, 2, \ldots, L_V$, some $L_V > 0$, where $v_j^i \in U(n_{ij})$,

$$\sum_{j=1}^{\ell_i} n_{ij} = n \quad \forall i = 1, 2, \ldots, L_V. \tag{4.69}$$

Let

$$M_i^V = U(n_{i1}) \times U(n_{i2}) \times \cdots \times U(n_{i\ell_i}). \tag{4.70}$$

Then V_i belongs to the smooth, connected Lie subgroup $M_i^V \subset U(n)$. The Lie algebra of M_i^V is

$$T_e M_i^V = \mathfrak{u}(n_{i1}) \oplus \mathfrak{u}(n_{i2}) \oplus \cdots \oplus \mathfrak{u}(n_{i\ell_i}). \tag{4.71}$$

As before, we choose to write and element of $T_e M_i^V$ as a block diagonal skew–

hermetian with the same block structure as V_i, i.e. $\Lambda_i \in T_e M_i^V$ will be written

$$\Lambda_i = \begin{bmatrix} \lambda_1^i & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \lambda_2^i & \cdots & \mathbf{0} \\ \vdots & & \ddots & \vdots \\ \mathbf{0} & \cdots & \mathbf{0} & \lambda_{\ell_i}^i \end{bmatrix}, \tag{4.72}$$

where $\lambda_j^i \in \mathfrak{u}(n_{ij})$ for $j = 1, 2, \ldots, \ell_i$. The tangent space of M_i^V at the point V_i is then

$$T_{V_i} M_i^V = \left\{ V_i \Lambda_i \,\middle|\, \Lambda_i \in T_e M_i^V \right\}. \tag{4.73}$$

Define the smooth Lie groups

$$\begin{aligned} M^U &= M_1^U \times M_2^U \times \cdots \times M_{L_U}^U, \\ M^V &= M_1^V \times M_2^V \times \cdots \times M_{L_V}^V, \\ M &= M^U \times M^V. \end{aligned} \tag{4.74}$$

Let $\Psi \in M$ be an $(L_U + L_V)$–tuple of U_is and V_is,

$$\Psi = (U_1, U_2, \ldots, U_{L_U}, V_1, V_2, \ldots, V_{L_V}). \tag{4.75}$$

Then $X \in T_\Psi M$ is an $(L_U + L_V)$–tuple

$$X = (U_1\Omega_1, U_2\Omega_2, \ldots, U_{L_U}\Omega_{L_U}, V_1\Lambda_1, V_2\Lambda_2, \ldots, V_{L_V}\Lambda_{L_V}), \tag{4.76}$$

where $\Omega_i \in T_e M_i^U$, $i = 1, 2, \ldots, L_U$, and $\Lambda_i \in T_e M_i^V$, $i = 1, 2, \ldots, L_V$.

The matrices P_i, $i = 1, 2, \ldots, L_U$, are fixed permutations of the $m \times m$ identity matrix. Likewise, Q_i, $i = 1, 2, \ldots, L_V$, be fixed permutations of the $n \times n$ identity matrix. Both sets of matrices are given as part of the ROT configuration parameters.

For $i = 1, 2, \ldots, L_U$, we define the sequences of m dimensional ROTs

$$\widetilde{U}_i = \prod_{j=1}^{i} P_j U_j \tag{4.77}$$

$$\breve{U}_i = \prod_{j=i+1}^{L_U} P_j U_j. \tag{4.78}$$

Likewise, for $i = 1, 2, \ldots, L_V$ we define the n dimensional ROTs

$$\widetilde{V}_i = \prod_{j=1}^{i} Q_j V_j \tag{4.79}$$

$$\breve{V}_i = \prod_{j=i+1}^{L_V} Q_j V_j. \tag{4.80}$$

We will call these ROTs subtransforms; $\widetilde{U}_i$ and $\widetilde{V}_i$ are lower subtransforms, while $\breve{U}_i$ and $\breve{V}_i$ are upper subtransforms. The ROTs $\widetilde{U} = \widetilde{U}_{L_U}$ and $\widetilde{V} = \widetilde{V}_{L_V}$ will be called complete transforms.

Constructing the Gradient Flow

The objective of this section is to find recursive orthogonal transforms $\widetilde{U}$ and $\widetilde{V}$ of the configuration outlined above that most closely approximate the SVD factors of the matrix H_0. To do this, we will search M for the $\Psi = (U_1, \ldots, U_{L_U}, V_1, \ldots, V_{L_V})$ which most closely diagonalizes the matrix

$$H(\Psi) = \widetilde{V}^H H_0 \widetilde{U}. \tag{4.81}$$

Consider the Frobenius distance between the fixed matrix N and $H(\Psi)$,

$$\|N - H(\Psi)\|^2 = \|N\|^2 + \|H_0\|^2 - 2\mathrm{retr}\left(N^H H(\Psi)\right). \tag{4.82}$$

Clearly, this quantity is minimized when the functional

$$\phi(\Psi) = 2\mathrm{retr}\left(N^H H(\Psi)\right) \tag{4.83}$$

is maximized. If we let the matrix N be a diagonal matrix with distinct values on the diagonal, then ϕ can be thought of as a "diagonalness" function.

As in the case of diagonalizing symmetric matrices, the approach we take is to search the Lie group M for the maximizing Ψ by flowing along the gradient vector field of ϕ. The gradient vector field $\nabla\phi$ is defined by the following properties:

1. $\nabla\phi(\Psi) \in T_\Psi M \quad \forall \Psi \in M.$

2. $D\phi_\Psi(X) = \langle \nabla\phi(\Psi), X\rangle \quad \forall X \in T_\Psi M.$

To satisfy the first property, $\nabla\phi(\Psi)$ must be of the form

$$\nabla\phi(\Psi) = \left(U_1\Omega_1^{\nabla\phi}, \ldots, U_{L_U}\Omega_{L_U}^{\nabla\phi}, V_1\Lambda_1^{\nabla\phi}, \ldots V_{L_V}\Lambda_{L_V}^{\nabla\phi}\right), \tag{4.84}$$

where $\Omega_i^{\nabla\phi} \in T_e M_i^U$, $i = 1, 2, \ldots, L_U$, and $\Lambda_i^{\nabla\phi} \in T_e M_i^V$, $i = 1, 2, \ldots, L_V$. For convenience of notation, we have surpressed the fact that these matrices depend on Ψ. Writing $\nabla\phi$ in this form, the second property states

$$\begin{aligned} D\phi_\Psi(X) &= \left\langle \left(U_1\Omega_1^{\nabla\phi}, \ldots, U_{L_U}\Omega_{L_U}^{\nabla\phi}, V_1\Lambda_1^{\nabla\phi}, \ldots V_{L_V}\Lambda_{L_V}^{\nabla\phi}\right), X\right\rangle \\ &= -\sum_{i=1}^{L_U} \mathrm{retr}\left(\Omega_i^{\nabla\phi}\Omega_i\right) - \sum_{i=1}^{L_V} \mathrm{retr}\left(\Lambda_i^{\nabla\phi}\Lambda_i\right). \end{aligned} \tag{4.85}$$

Finding and expression for $D\phi_\Psi(X)$, we get

$$\begin{aligned} D\phi_\Psi(X) &= 2\mathrm{retr}\left(\sum_{i=1}^{L_U} N^H \widetilde{V}^H H_0 \widetilde{U}_i \Omega_i \breve{U}_i + \sum_{i=1}^{L_V} N^H \left(\widetilde{V}_i \Lambda_i \breve{V}_i\right)^H H_0 \widetilde{U}\right) \\ &= 2\mathrm{retr}\left(\sum_{i=1}^{L_U} \breve{U}_i N^H \widetilde{V}^H H_0 \widetilde{U}_i \Omega_i - \sum_{i=1}^{L_V} \widetilde{V}_i^H H_0 \widetilde{U} N^H \breve{V}_i^H \Lambda_i\right) \\ &= 2\mathrm{retr}\left(\sum_{i=1}^{L_U} \frac{1}{2}\left(\breve{U}_i N^H \widetilde{V}^H H_0 \widetilde{U}_i - \widetilde{U}_i^H H_0^H \widetilde{V} N \breve{U}_i^H\right)\Omega_i \right. \\ &\qquad \left. - \sum_{i=1}^{L_V} \frac{1}{2}\left(\widetilde{V}_i^H H_0 \widetilde{U} N^H \breve{V}_i^H - \breve{V}_i N \widetilde{U}^H H_0^H \widetilde{V}_i\right)\Lambda_i\right) \end{aligned}$$

$$
\begin{aligned}
&= \sum_{i=1}^{L_U} \operatorname{retr}\left(\left\{\breve{V}_i N \breve{U}_i^H, \widetilde{V}_i^H H_0 \widetilde{U}_i\right\} \Omega_i\right) \\
&\qquad + \sum_{i=1}^{L_V} \operatorname{retr}\left(\left\{\breve{U}_i N^H \breve{V}_i^H, \widetilde{U}_i^H H_0^H \widetilde{V}_i\right\} \Lambda_i\right) \qquad (4.86)
\end{aligned}
$$

In the last Equation, the matrices multiplying Ω_i and Λ_i are in $\mathfrak{u}(m)$ and $\mathfrak{u}(n)$, respectively. Recall that in order for the gradient vector to be tangent to M, each Ω_i must be multiplied on the left by $\Omega_i^{\nabla\phi}$, where $\Omega_i^{\nabla\phi} \in T_e(M_i^U)$ and each Λ_i must be multiplied on the left by $\Lambda_i^{\nabla\phi}$, where $\Lambda_i^{\nabla\phi} \in T_e(M_i^U)$. To accomplish this, we introduce a set of natural projection operators, $\Pi_i^U : T_e U(m) \to T_e(M_i^U)$, $i = 1, 2, \ldots, L_U$ and $\Pi_i^V : T_e U(n) \to T_e(M_i^V)$, $i = 1, 2, \ldots, L_V$. These operators project from the set of skew–hermetian matrices to the set of block diagonal skew–hermetian matrices by setting all off diagonal blocks to zero. Using these operators along with Fact 4.1.1, we get

$$
\begin{aligned}
D\phi_\Psi(X) &= \sum_{i=1}^{L_U} \operatorname{retr}\left(\Pi_i^U \left\{\breve{V}_i N \breve{U}_i^H, \widetilde{V}_i^H H_0 \widetilde{U}_i\right\} \Omega_i\right) \\
&\qquad + \sum_{i=1}^{L_V} \operatorname{retr}\left(\Pi_i^V \left\{\breve{U}_i N^H \breve{V}_i^H, \widetilde{U}_i^H H_0^H \widetilde{V}_i\right\} \Lambda_i\right). \qquad (4.87)
\end{aligned}
$$

Looking back to Equation 4.53, we see that the choices

$$
\begin{aligned}
\Omega_i^{\nabla\phi} &= -\Pi_i^U \left\{\breve{V}_i N \breve{U}_i^H, \widetilde{V}_i^H H_0 \widetilde{U}_i\right\}, \quad i = 1, 2, \ldots L_U, \\
\Lambda_i^{\nabla\phi} &= -\Pi_i^V \left\{\breve{U}_i N^H \breve{V}_i^H, \widetilde{U}_i^H H_0^H \widetilde{V}_i\right\}, \quad i = 1, 2, \ldots L_V. \qquad (4.88)
\end{aligned}
$$

satisfy both gradient flow properties. As a result, the gradient flow equations are

$$
\dot{U}_i = -U_i \Pi_i^V \left\{\breve{V}_i N \breve{U}_i^H, \widetilde{V}_i^H H_0 \widetilde{U}_i\right\} \quad U_i(0) = U_{i0} \in M_i^U, \qquad (4.89)
$$

for $i = 1, 2, \ldots, L_U$, and

$$
\dot{V}_i = -V_i \Pi_i^V \left\{\breve{U}_i N^H \breve{V}_i^H, \widetilde{U}_i^H H_0^H \widetilde{V}_i\right\}, \quad V_i(0) = V_{i0} \in M_i^V, \qquad (4.90)
$$

for $i = 1, 2, \ldots, L_V$.

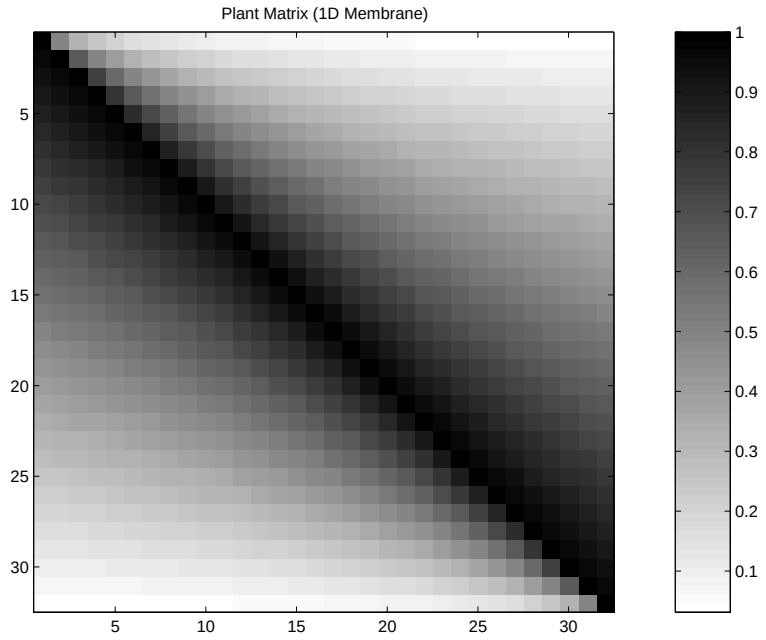


Figure 4.7: Plant matrix for flexible membrane.

Examples

Here we apply this method to the examples introduced in Section 2.4. The plant matrix for the simplified membrane example is depicted in Figure 4.7. Approximately diagonalized plant matrices using 10, 20, and 31 level ROTs are shown in Figures 4.8, 4.9, and 4.10, respectively. The plant matrix for the flexible beam at its 6th resonance is plotted in Figure 4.11. Approximately diagonalized plant matrices using 10, 20, and 31 level ROTs are shown in Figures 4.12, 4.13, and 4.14, respectively. These results will be discussed more quantitatively in Chapter 6.

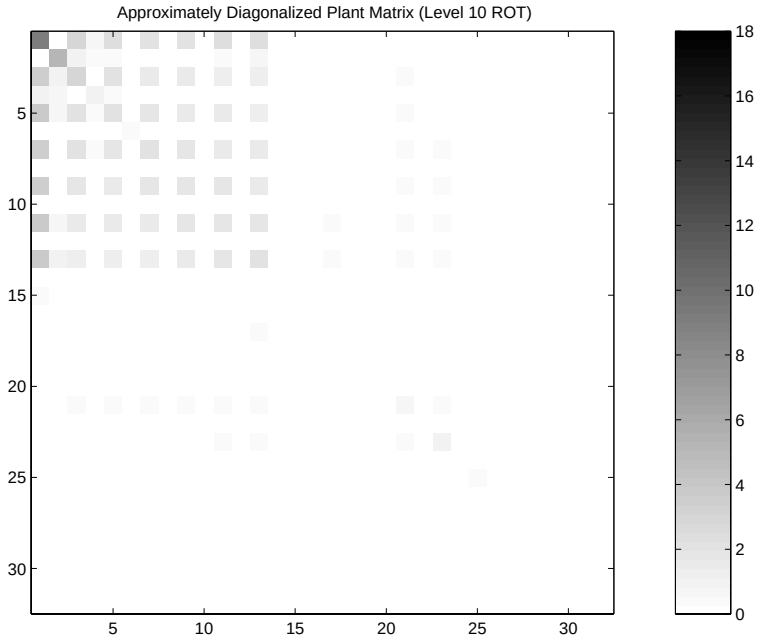


Figure 4.8: Approximately diagonalized membrane plant matrix using 10 level ROT.

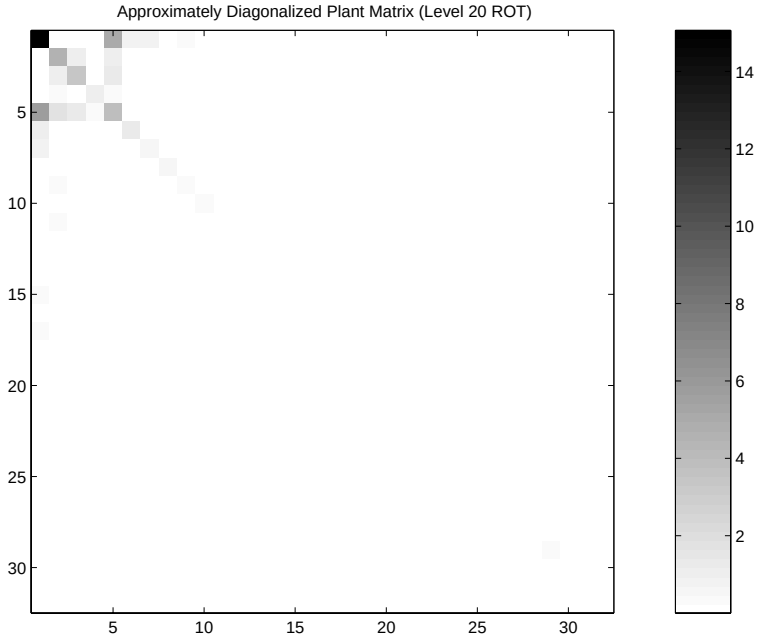


Figure 4.9: Approximately diagonalized membrane plant matrix using 20 level ROT.

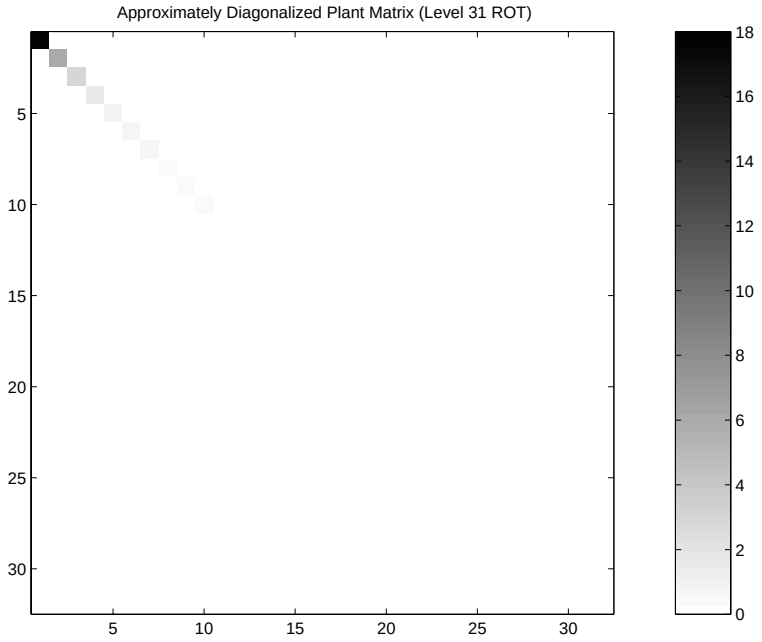


Figure 4.10: Approximately diagonalized membrane plant matrix using 31 level ROT.

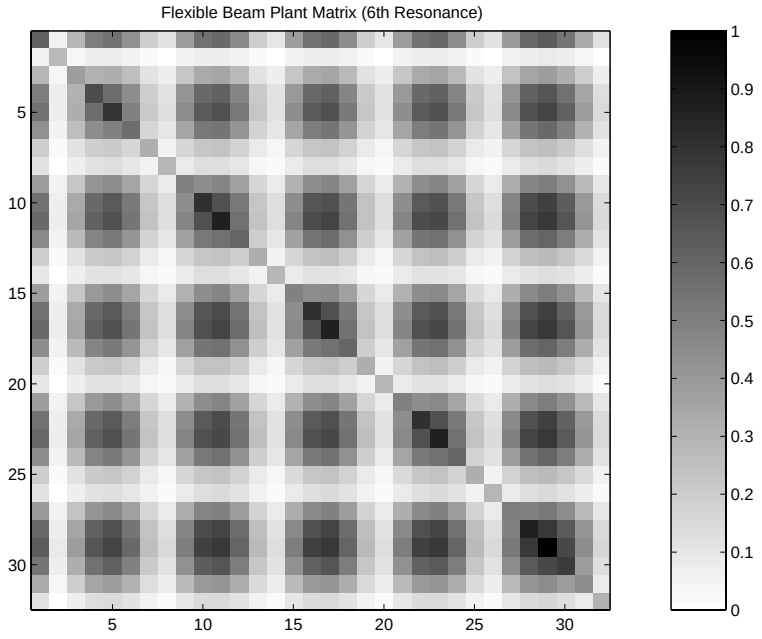


Figure 4.11: Plant matrix for flexible beam at 6th resonance.

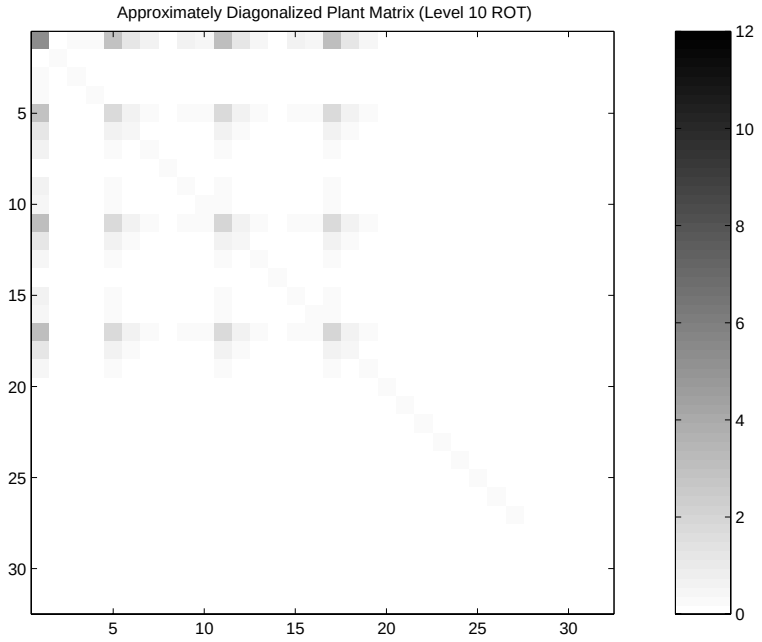


Figure 4.12: Approximately diagonalized flexible beam plant matrix using 10 level ROT.

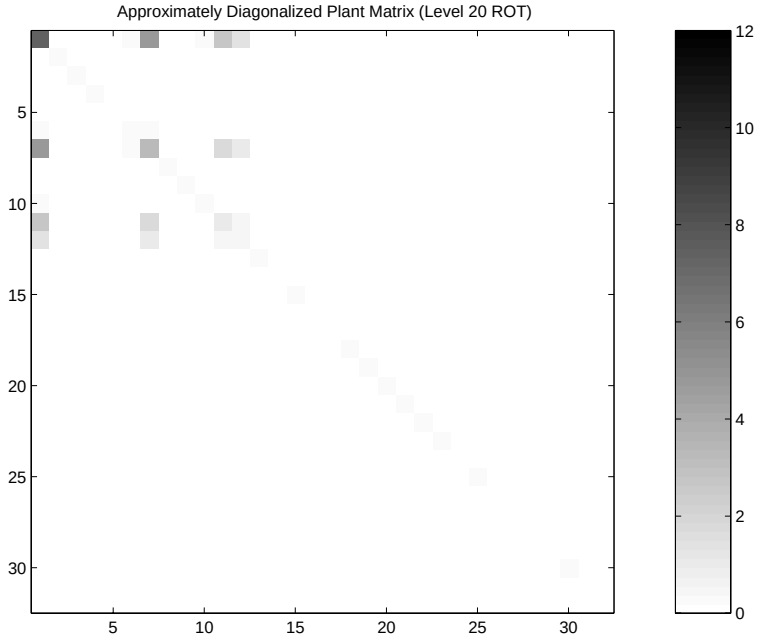


Figure 4.13: Approximately diagonalized flexible beam plant matrix using 20 level ROT.

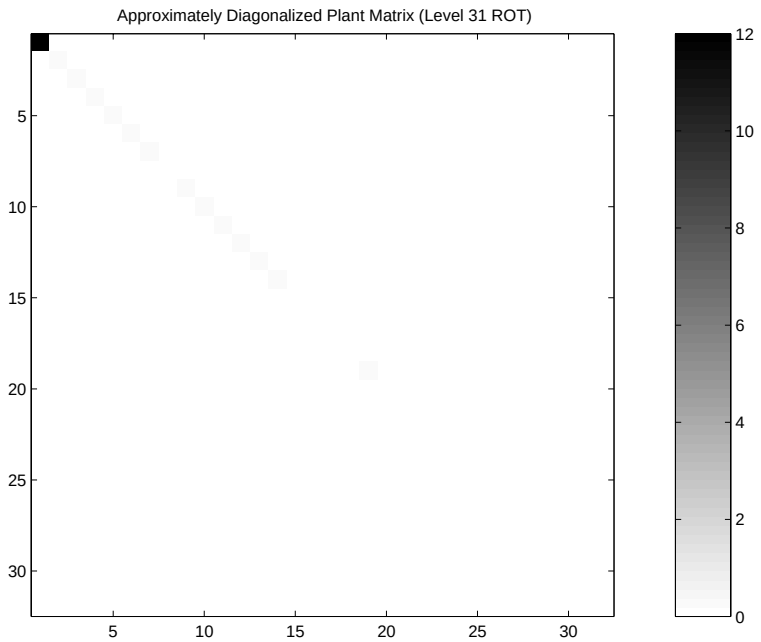


Figure 4.14: Approximately diagonalized flexible beam plant matrix using 31 level ROT.

Chapter 5

Spatially Invariant Systems

In many applications of sensor/actuator arrays, a large number of identical sensor/actuator pairs are placed at regular intervals in some homogeneous medium. Such systems often exhibit a spatial invariance property; the coupling between any two nodes depends only on the relative positions of the nodes and not on their absolute positions in the array. This property, which will be defined more rigorously below, can be exploited to aid in the design and implementation of feedback controllers. Brockett and Willems [11] [12] used the spatial invariance property found in discretized partial differential equations to develop optimal feedback laws. Melzer and Kuo [27] formulated a Riccatti equation for spatially invariant systems and then applied their results to the problem of controlling an infinite string of vehicles. El-Sayed and Krishnaprasad [16] employed a similar method to design optimal feedback laws to control the depth of a seismic cable containing an array of hydrophones. More recently, Bamieh [1] and Bamieh, Paganini, and Dahleh [2] have proposed the use of this same idea to develop controllers for large scale sensor/actuator arrays.

Here we propose to exploit the spatial invariance property for the purpose of

plant matrix diagonalization. We begin by defining a spatially invariant system and discussing some of the features that such systems exhibit. We then show that the discrete Fourier transform can be used to diagonalize the plant matrix of any spatially invariant system. Using this information, we then address the problem of diagonalizing such systems using ROTs.

5.1 Infinite Sensor/Actuator Arrays

Here we define the spatial invariance property for an infinite sensor/actuator array and we discuss how this property can be exploited to aid in controller design and implementation for such systems. Of course, this discussion is purely hypothetical since infinite arrays do not exist. In the next section we will extend the ideas discussed to arrays of finite length.

Consider a control system consisting of some physical plant equipped with a doubly infinite sensor/actuator array. Assume that each node on the array contains exactly one sensor and one actuator. Let y_i be the output measured at the ith sensor and let u_i be the input applied to the ith actuator. This system is *spatially invariant* if, for each i, the output y_i can be written as

$$y_i = \sum_{i-j \in M} g_{i-j} u_j \tag{5.1}$$

where g_{i-j} is a scalar, finite dimensional linear operator representing the dynamic coupling between y_i and and u_j. The set $M \subset \mathbb{Z}$ defines the extent of the spatial coupling between y and u. The coupling function g_{i-j} is zero whenever $i - j$ is outside of M. For example, nearest neighbor coupling can be represented by letting $M = \{-1, 0, 1\}$. This system is invariant under discrete translations. In other words, the relationship between y_i and u_j is identical to the relationship

between y_{i+k} and u_{j+k} for all $k \in \mathbb{Z}$.

In the time domain, g_{i-j} represents the convolution (in time) with a weighting function $w_{i-j}(t)$. Equation 5.1 is then be written in the time domain as

$$y_i(t) = \sum_{i-j \in M} \int_0^t w_{i-j}(t-\sigma) u_j(\sigma) d\sigma. \tag{5.2}$$

In the frequency domain, g_{i-j} represents multiplication by a finite dimensional transfer function, $g_{i-j}(s)$. In this case, Equation 5.1 becomes

$$y_i(s) = \sum_{i-j \in M} g_{i-j}(s) u_j(s), \tag{5.3}$$

where $y_i(s)$ and $u_j(s)$ are the Laplace transforms of the time varying quantities $y_i(t)$ and $u_j(t)$, respectively. For convenience, the remainder of this discussion will use the s domain representation given by Equation 5.3.

It is worthwhile at this point to take a close look at the variables in Equation 5.3. First consider the familiar Laplace variable s. The s domain is associated with the temporal frequency domain by making the assignment $s = j\omega_t$. Traditionally, the "temporal frequency domain" is simply called the "frequency domain", but in this case it is important to specify that we mean frequency in time as opposed to frequency in space. The reason for this distinction will soon become clear. The second variable in Equation 5.3 consists of the subscript i, $i-j$, or j in the case of y, g, or u, respectively. This variable represents position in space. Hence, the subscript describes the spatial behavior of the system while the Laplace variable describes the temporal behavior.

The spatial invariance exhibited by Equation 5.3 is analogous to the well known time invariance property exhibited by discrete linear time invariant systems. Here,

$$\mathbf{g}(s) = \{\ldots, g_{-1}(s), g_0(s), g_1(s), \ldots\}$$

can be thought of as a spatial impulse response. The elements of this impulse response are transfer functions. The output $\mathbf{y}(s) = \{\ldots, y_{-1}(s), y_0(s), y_1(s), \ldots\}$ can be written as $\mathbf{y}(\mathbf{s}) = \mathbf{g}(s) * \mathbf{u}(s)$, where the $*$ operator represents convolution in the spatial domain and $\mathbf{u}(s) = \{\ldots, u_{-1}(s), u_0(s), u_1(s), \ldots\}$. It follows then that we can write

$$Y(s,z) = G(s,z)U(s,z), \tag{5.4}$$

where $Y(s,z)$, $G(s,z)$, and $U(s,z)$ are the 2-sided spatial z transforms of $\mathbf{y}(s)$, $\mathbf{g}(s)$, and $\mathbf{u}(s)$, respectively. The 2-sided spatial z transform is defined to be

$$X(s,z) = \sum_{i=-\infty}^{\infty} x_i(s) z^i \tag{5.5}$$

where i indexes the spatial domain. Here, z is the spatial shift operator, i.e. $z : x_i \mapsto x_{i+1}$.

Assume that $\mathbf{y}(s)$, $\mathbf{g}(s)$, and $\mathbf{u}(s)$ satisfy conditions such that their z transforms exist on the unit circle. Then the z domain can be though of as the spatial frequency domain by assigning $z = e^{j\omega_s}$. Hence, Equation 5.4 is a frequency domain representation of the system in both time and space.

This representation provides some advantages in the development and implementation of feedback control laws for the system. Suppose we wish to develop a spatially invariant feedback law $U(s,z) = K(s,z)Y(s,z)$. Under this feedback, the closed loop transfer function becomes

$$G_{cl}(s,z) = \frac{G(s,z)}{1 + K(s,z)G(s,z)}. \tag{5.6}$$

It is possible to evaluate the stability of the $G_{cl}(s,z)$ using extensions of classical frequency domain methods such as the Nyquist and circle criterion. Under some conditions, spectral factorization techniques can be used to develop optimal spatially invariant control laws [12]. Once a controller is decided upon, implementation

on a control network is straight forward; the control law $U(s,z) = K(s,z)Y(s,z)$ is applied by each node on the network.

5.2 Periodic Arrays and Circulant Matrices

In the previous section we discussed the spatial invariance property for an infinite sensor/actuator array. As is done frequently in the time invariant case, these ideas can be applied to a finite array by assuming that the array is periodic. Consider a sensor/actuator array with n nodes where each node has exactly one sensor and one actuator. The periodic assumption means that $y_{i+kn}(s) = y_i(s)$ and $u_{i+kn}(s) = u_i(s)$ for all $i = 0, 1, \ldots, n-1$ and for all $k \in \mathbb{Z}$. We call the system spatially invariant if

$$y_i(s) = \sum_{j=0}^{n-1} g_{((i-j) \bmod n)}(s) u_j(s) \tag{5.7}$$

for $i = 0, 1, \ldots, n-1$. Physically, this means that the array must wrap around so that the $(n-1)$th node is adjacent to the zeroth node. An example of this would be a circular array.

Let $y(s) = [y_0(s), y_1(s), \ldots, y_{n-1}(s)]^T$ and $u(s) = [u_0(s), u_1(s), \ldots, u_{n-1}(s)]^T$. Then Equation 5.7 becomes

$$y(s) = P(s)u(s) \tag{5.8}$$

where

$$P(s) = \begin{bmatrix} g_0(s) & g_{n-1}(s) & g_{n-2}(s) & \cdots & g_1(s) \\ g_1(s) & g_0(s) & g_{n-1}(s) & \cdots & g_2(s) \\ g_2(s) & g_1(s) & g_0(s) & \cdots & g_3(s) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ g_{n-1}(s) & g_{n-2}(s) & g_{n-3}(s) & \cdots & g_0(s) \end{bmatrix}. \tag{5.9}$$

In this equation, $g_i(s)$ is a finite dimensional, scalar transfer function for $i = 0, 1, \ldots, n-1$. Matrices of the form given by Equation 5.9 are said to be *circulant* [12]. A finite dimensional system is spatially invariant if and only if its associated plant matrix is circulant.

Define z to be the circular shift operator, i.e.

$$z : x_i \mapsto \begin{cases} x_{i+1} & i \neq n-1 \\ x_0 & i = n-1 \end{cases} \tag{5.10}$$

Note that as a direct result of this definition,

$$z^k = z^{k \bmod n} \tag{5.11}$$

for all $k \in \mathbb{Z}$. Define the circular z transform of the n-vector $x(s)$ to be

$$X(s, z) = \sum_{i=0}^{n-1} x_i(s) z^i. \tag{5.12}$$

Then, because of the spatial invariance property of the plant matrix $P(s)$, the system given by Equation 5.8 can be written in the z domain as

$$Y(s, z) = G(s, z) U(s, z). \tag{5.13}$$

Here $G(s, z)$ is the z transform of $g(s) = [g_0(s), g_1(s), \ldots, g_{n-1}(s)]^T$.

The discrete Fourier transform (DFT) of an n dimensional vector is equivalent to evaluating the z transform at n evenly spaced points on the unit circle. In other words, the kth element of the DFT of x is $X(e^{-j2\pi k/n})$. As a result, the DFT can be used to diagonalize the system of Equation 5.8. The resulting system is written

as

$$\begin{bmatrix} Y(s,1) \\ Y(s,e^{\frac{j}{n}}) \\ \vdots \\ Y(s,e^{\frac{j(n-1)}{n}}) \end{bmatrix} = \begin{bmatrix} G(s,1) & 0 & \cdots & 0 \\ 0 & G(s,e^{\frac{j}{n}}) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & G(s,e^{\frac{j(n-1)}{n}}) \end{bmatrix} \begin{bmatrix} U(s,1) \\ U(s,e^{\frac{j}{n}})) \\ \vdots \\ U(s,e^{\frac{j(n-1)}{n}}) \end{bmatrix}. \tag{5.14}$$

To put this idea into the context of plant diagonalization as discussed in the previous chapters, consider the coordinate transformations $\widetilde{y}(s) = \mathcal{F}_n y(s)$ and $\widetilde{u}(s) = \mathcal{F}_n u(s)$. For a circulant plant matrix $P(s)$ we have

$$\begin{aligned} y(s) &= P(s)u(s) \\ \mathcal{F}_n^{-1}\widetilde{y}(s) &= P(s)\mathcal{F}_n^{-1}\widetilde{u}(s) \\ \widetilde{y}(s) &= \mathcal{F}_n P(s)\mathcal{F}_n^{-1}\widetilde{u}(s). \end{aligned} \tag{5.15}$$

It is not difficult to see that the matrix $\mathcal{F}_n P(s)\mathcal{F}_n^{-1}$ is equal to the diagonal matrix on the right hand side of Equation 5.14. Hence, the DFT matrix can be used to diagonalize any circulant plant matrix.

5.3 Diagonalizing Circulant Plants

In the previous section we showed that the discrete Fourier transform can be used to diagonalize plants which exhibit a spatial invariance property. Here we work from this result to find ROTs which diagonalize these plants.

5.3.1 Background: The Discrete Fourier Transform

Let $x = [x_0, x_1, \ldots, x_{n-1}]^T$ be a vector in $\mathbb{R}^n$ and let $X(z)$ be the circular z transform of x as defined in Equation 5.12. The DFT of x is $X(z)$ evaluated at

n evenly spaced points on the unit circle in the complex plane. Let $\mathbf{X}$ denote the DFT of x. The kth element of the $\mathbf{X}$ is $X(e^{-j2\pi k/n})$, $k = 0, 1, 2, \ldots, n-1$. Using the definition of the circular z transform, we get

$$\mathbf{X}(k) = X(e^{-j2\pi k/n}) = \sum_{\ell=0}^{n-1} x_\ell e^{-j2\pi k\ell/n}. \tag{5.16}$$

Hence the DFT can be written as a matrix whose kth row is

$$\begin{bmatrix} 1 & e^{-j2\pi k/n} & e^{-j2\pi 2k/n} & \ldots & e^{-j2\pi(n-1)k/n} \end{bmatrix}. \tag{5.17}$$

We will denote this matrix as $\mathcal{F}_n$, i.e. $\mathbf{X} = \mathcal{F}_n x$.

Using Equation 5.17, we see that the (ℓ, k)th element of $\mathcal{F}_n \mathcal{F}_n^H$ is given by

$$\left[\mathcal{F}_n \mathcal{F}_n^H\right]_{\ell k} = \sum_{m=0}^{n-1} e^{-j2\pi m(\ell-k)/n}. \tag{5.18}$$

If $\ell = k$, then all of the terms in the sum on the right hand side are equal to 1 and $\left[\mathcal{F}_n \mathcal{F}_n^H\right]_{\ell k} = n$. If $\ell \neq k$, then the terms in the sum are evenly spaced points on the unit circle. As a result, they add up to zero and $\left[\mathcal{F}\mathcal{F}^H\right]_{\ell k} = 0$. Hence, $\mathcal{F}_n \mathcal{F}_n^H = n\mathbb{1}$. Clearly, the matrix $F_n \triangleq \mathcal{F}_n/\sqrt{n}$ is unitary.

5.3.2 Gradient Flows Diagonalizing Circulant Matrices

For a given set of ROT parameters, we consider the task of finding the ROT variables which most closely diagonalize any circulant plant matrix. As we have shown in the previous sections, the DFT matrix accomplishes the goal of diagonalization and is unitary when properly scaled. In Chapter 4 we showed how to use a gradient flow to find the ROT variables which approximate the unitary SVD factors of an arbitrary complex matrix. Here we take a similar approach to find an ROT which diagonalizes any circulant matrix.

Recall that in the case of diagonalizing a symmetric matrix H_0, we constructed a gradient flow to find the $\widetilde{\Theta}$ which minimized $\left\| N - \widetilde{\Theta}^T H_0 \widetilde{\Theta} \right\|^2$, where N was a diagonal matrix with the same dimension as H_0. Here instead of trying to diagonalize a single matrix H_0, we are trying to diagonalize the whole set of circulant matrices. Towards this end, we notice that the set of $n \times n$ circulant matrices forms an n dimensional vector space. As a result, there exist a set of basis vectors $\{E_0, E_1, E_2, \ldots, E_{n-1}\}$ such that any circulant matrix A can be written

$$A = \sum_{i=0}^{n-1} \alpha_i E_i, \tag{5.19}$$

with $\alpha_i \in \mathbb{R}$ for $i = 0, 1, 2, \ldots, n-1$. The task of diagonalizing (or approximately diagonalizing) any circulant matrix can then be posed as the task of simultaneously diagonalizing each of the n basis vectors. Also, each E_i is circulant, so the matrix $N_i = F E_i F^H$ is diagonal. Hence, the problem of diagonalizing any circulant matrix can be posed as the problem of finding the $\widetilde{\Theta}$ which minimizes the cost function

$$J(\widetilde{\Theta}) = \sum_{i=0}^{n-1} \left\| N_i - \widetilde{\Theta}^H E_i \widetilde{\Theta} \right\|^2. \tag{5.20}$$

As in Chapter 4, simple matrix manipulation shows that minimizing J is equivalent to maximizing the "diagonalness" function

$$\phi\left(\widetilde{\Theta}\right) = \sum_{i=0}^{n-1} \operatorname{retr}\left(N_i^H \widetilde{\Theta}^H E_i \widetilde{\Theta}\right). \tag{5.21}$$

The unitary operator $\widetilde{\Theta}$ is an ROT, i.e.

$$\widetilde{\Theta} = \prod_{k=1}^{L} P_k \Theta_k. \tag{5.22}$$

Each Θ_k, $k = 1, 2, \ldots, L$ is of the form

$$\Theta_k = \begin{bmatrix} \theta_1^k & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \theta_2^k & \cdots & \mathbf{0} \\ \vdots & & \ddots & \vdots \\ \mathbf{0} & \cdots & \mathbf{0} & \theta_{m_k}^k \end{bmatrix}, \tag{5.23}$$

where $\theta_j^k \in U(n_{kj})$,

$$\sum_{j=1}^{m_k} n_{kj} = n \quad \forall k = 1, 2, \ldots, L, \tag{5.24}$$

and the $\mathbf{0}$s represent appropriately dimensioned blocks of zeros. We define $M_k = U(n_{k1}) \times U(n_{k2}) \times \cdots \times U(n_{km_k})$. Then each Θ_k belongs the smooth Lie subgroup $M_k \in U(n)$. The Lie algebra of M_k is

$$T_e M_k = \mathfrak{u}(n_{k1}) \oplus \mathfrak{u}(n_{k2}) \oplus \cdots \oplus \mathfrak{u}(n_{km_k}). \tag{5.25}$$

A vector in $T_e M_k$ can then be written

$$\Omega_k = \begin{bmatrix} \omega_1^k & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \omega_2^k & \cdots & \mathbf{0} \\ \vdots & & \ddots & \vdots \\ \mathbf{0} & \cdots & \mathbf{0} & \omega_{m_k}^k \end{bmatrix}. \tag{5.26}$$

where $\omega_j^k \in \mathfrak{u}(n_{kj})$ for $j = 1, 2, \ldots, m_k$.

Here, we assume that the configuration parameters of the ROT are given and fixed. In other words, the level L is fixed and for $k = 1, 2, \ldots L$, the following quantities are fixed: P_k, m_k, and n_{kj}, $j = 1, 2, \ldots, m_k$. Our task is to find the ROT variables, i.e. the Θ_ks, which maximize ϕ.

Let $M = M_1 \times M_2 \times \cdots \times M_L$. The set M is a Lie group. An element of M is written as the L-tuple $\Psi = (\Theta_1, \Theta_2, \ldots, \Theta_L)$. A vector in $T_\Psi M$ is written as $X = (\Theta_1\Omega_1, \Theta_2\Omega_2, \ldots, \Theta_L\Omega_L)$, where $\Omega_k \in T_e M_k$ for each k.

The diagonalness function ϕ can now be thought of as a functional on M, $\phi : M \to \mathbb{R}$. As in Chapter 4, our approach to searching for the Ψ which maximizes ϕ is to flow along the gradient vector field of ϕ on M.

Recall that $\nabla\phi(\Psi)$ on M is defined by the following properties:

1. $\nabla\phi(\Psi) \in T_\Psi M \quad \forall \Psi \in M.$

2. $D\phi_\Psi(X) = \langle \nabla\phi(\Psi), X\rangle \quad \forall X \in T_\Psi M.$

The first property states that $\nabla\phi(\Psi)$ in contained in the tangent space of M at the point Ψ. This means that $\nabla\phi(\Psi)$ must be of the form

$$\nabla\phi(\Psi) = \left(\Theta_1\Omega_1^{\nabla\phi}, \Theta_2\Omega_2^{\nabla\phi}, \ldots, \Theta_L\Omega_L^{\nabla\phi}\right), \tag{5.27}$$

where $\Omega_k^{\nabla\phi} \in T_e M_k,\ k = 1, 2, \ldots, L$. For convenience of notation, we have suppressed the fact that $\Omega_k^{\nabla\phi}$ depends on Ψ.

From the second property, we must have

$$\begin{aligned} D\phi_\Psi((\Theta_1\Omega_1, \ldots, \Theta_L\Omega_L)) &= \langle \nabla\phi(\Theta_1, \ldots, \Theta_L), (\Theta_1\Omega_1, \ldots, \Theta_L\Omega_L)\rangle \\ &= \left\langle (\Theta_1\Omega_1^{\nabla\phi}, \ldots, \Theta_L\Omega_L^{\nabla\phi}), (\Theta_1\Omega_1, \ldots, \Theta_L\Omega_L)\right\rangle \\ &= \sum_{k=1}^{L} \operatorname{retr}\left((\Omega_k^{\nabla\phi})^H \Omega_k\right) \\ &= -\sum_{k=1}^{L} \operatorname{retr}\left(\Omega_k^{\nabla\phi}\Omega_k\right) \end{aligned} \tag{5.28}$$

Finding an expression for $D\phi_\Psi(X)$:

$$\begin{aligned} &D\phi_\Psi(X) = \\ &\operatorname{retr}\Bigg(\textstyle\sum_{k=1}^{L}\sum_{i=0}^{n-1} N_i^H \left(\left(\prod_{\ell=1}^{k-1} P_\ell\Theta_\ell\right) P_k\Theta_k\Omega_k \left(\prod_{\ell=k+1}^{n} P_\ell\Theta_\ell\right)\right)^H E_i \left(\prod_{\ell=1}^{n} P_\ell\Theta_\ell\right) \\ &\qquad + N_i^H \left(\prod_{\ell=1}^{n} P_\ell\Theta_\ell\right)^H E_i \left(\prod_{\ell=1}^{k-1} P_\ell\Theta_\ell\right) P_k\Theta_k\Omega_k \left(\prod_{\ell=k+1}^{n} P_\ell\Theta_\ell\right)\Bigg). \end{aligned} \tag{5.29}$$

To simplify notation, we define two sequences of recursive orthogonal transforms:

$$\widetilde{\Theta}_k = \prod_{\ell=1}^{k} P_\ell \Theta_\ell, \tag{5.30}$$

$$\breve{\Theta}_k = \prod_{\ell=k+1}^{n} P_\ell \Theta_\ell. \tag{5.31}$$

Using this notation, we get

$$\begin{aligned} D\phi_\Psi(X) &= \operatorname{retr}\left(\sum_{k=1}^{L}\sum_{i=0}^{n-1} N_i^H \left(\widetilde{\Theta}_k \Omega_k \breve{\Theta}_k\right)^H E_i \widetilde{\Theta} + N_i^H \widetilde{\Theta}^T E_i \widetilde{\Theta}_k \Omega_k \breve{\Theta}_k\right) \\ &= \operatorname{retr}\left(\sum_{k=1}^{L}\sum_{i=0}^{n-1} N_i^H \breve{\Theta}_k^T \Omega_k^T \widetilde{\Theta}_k^H E_i \widetilde{\Theta} + N_i^H \widetilde{\Theta}^H E_i \widetilde{\Theta}_k \Omega_k \breve{\Theta}_k\right) \\ &= \sum_{k=1}^{L} \operatorname{retr}\left(\left(\sum_{i=0}^{n-1}\left[\breve{\Theta}_k N_i^H \breve{\Theta}_k^H, \widetilde{\Theta}_k^H E_i \widetilde{\Theta}_k\right]\right)\Omega_k\right). \end{aligned} \tag{5.32}$$

Claim 5.3.1 *Let $\{E_0, E_1, \ldots, E_{n-1}\}$ be an orthonormal basis for the space of $n \times n$ circulant matrices. Let $N_i = F E_i F^H$, for $i = 0, 1, \ldots, n-1$, where F is the DFT matrix scaled by $1/\sqrt{n}$. Then the matrix*

$$\Gamma_k = \sum_{i=0}^{n-1}\left[\breve{\Theta}_k N_i^H \breve{\Theta}_k^H, \widetilde{\Theta}_k^H E_i \widetilde{\Theta}_k\right] \tag{5.33}$$

is skew–hermitian for $k = 1, 2, \ldots, L$.

<u>Proof:</u>

Expanding the Lie bracket and substituting for N_i, we get

$$\begin{aligned} \Gamma_k &= \sum_{i=0}^{n-1} \breve{\Theta}_k F^H E_i^H F \breve{\Theta}_k^H \widetilde{\Theta}_k^H E_i \widetilde{\Theta}_k - \widetilde{\Theta}_k^H E_i \widetilde{\Theta}_k \breve{\Theta}_k F^H E_i^H F \breve{\Theta}_k^H \\ &= \breve{\Theta}_k F^H \left(\sum_{i=0}^{n-1} E_i^H F \breve{\Theta}_k^H \widetilde{\Theta}_k^H E_i \widetilde{\Theta}_k \breve{\Theta}_k F^H \right. \\ &\qquad\qquad \left. - F \breve{\Theta}_k^H \widetilde{\Theta}_k^H E_i \widetilde{\Theta}_k \breve{\Theta}_k F^H E_i^H\right) F \breve{\Theta}_k^H. \end{aligned} \tag{5.34}$$

Define $A \triangleq \widetilde{\Theta}_k \breve{\Theta}_k F^H$ and define

$$\begin{aligned} K &\triangleq \sum_{i=0}^{n-1} E_i^H F \breve{\Theta}_k^H \widetilde{\Theta}_k^H E_i \widetilde{\Theta}_k \breve{\Theta}_k F^H - F \breve{\Theta}_k^H \widetilde{\Theta}_k^H E_i \widetilde{\Theta}_k \breve{\Theta}_k F^H E_i^H \\ &= \sum_{i=0}^{n-1} E_i^H A^H E_i A - A^H E_i A E_i^H. \end{aligned} \tag{5.35}$$

Substituting K into Equation 5.34 and taking the (Hermitian) transpose gives

$$\begin{aligned} \Gamma_k^H &= \left(\breve{\Theta}_k F^H K F \breve{\Theta}_k^H \right)^H \\ &= \breve{\Theta}_k F^H K^H F \breve{\Theta}_k^H. \end{aligned} \tag{5.36}$$

As a result, Γ_k is skew if and only if K is skew. By definition, K is skew if $K + K^H = 0$. Finding an expression for $K + K^H$:

$$\begin{aligned} K + K^H &= \sum_{i=0}^{n-1} E_i^H A^H E_i A - A^H E_i A E_i^H + \left(\sum_{i=0}^{n-1} E_i^H A^H E_i A - A^H E_i A E_i^H \right)^H \\ &= \sum_{i=0}^{n-1} E_i^H A^H E_i A - A^H E_i A E_i^H + A^H E_i^H A E_i - E_i A^H E_i^H A. \end{aligned} \tag{5.37}$$

Since $\{E_i\}$ is a basis for circulant matrices and E_i^H is a circulant matrix, we can write each E_i^H as

$$E_i^H = \sum_{j=0}^{n-1} \alpha_{ij} E_j, \tag{5.38}$$

where each $\alpha_{ij} \in \mathbb{R}$. Since $\{E_i\}$ is orthonormal, we have an explicit expression for α_{ij}:

$$\begin{aligned} \alpha_{ij} &= \left\langle E_i^H, E_j \right\rangle \\ &= \operatorname{retr}\left(E_i E_j \right). \end{aligned} \tag{5.39}$$

Note that $\operatorname{retr}(E_i E_j) = \operatorname{retr}(E_j E_i)$, so $\alpha_{ij} = \alpha_{ji}$ for each i and j. Substituting for E_i^H in Equation 5.37 yields

$$
\begin{aligned}
&K + K^H \\
&= \sum_{i=0}^{n-1} \left(\sum_{j=0}^{n-1} \alpha_{ij} E_j \right) A^H E_i A - A^H E_i A \left(\sum_{j=0}^{n-1} \alpha_{ij} E_j \right) \\
&\quad + A^H \left(\sum_{j=0}^{n-1} \alpha_{ij} E_j \right) A E_i - E_i A^H \left(\sum_{j=0}^{n-1} \alpha_{ij} E_j \right) A \\
&= \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} \alpha_{ij} \left(\left(E_j A^H E_i A - E_i A^H E_j A \right) + \left(A^H E_j A E_i - A^H E_i A E_j \right) \right) \\
&= \sum_{i=0}^{n-1} \sum_{j=i+1}^{n-1} \alpha_{ij} \left(\left(\left(E_j A^H E_i A - E_i A^H E_j A \right) + \left(A^H E_j A E_i - A^H E_i A E_j \right) \right) \right. \\
&\quad \left. + \alpha ji \left(\left(E_i A^H E_j A - E_j A^H E_i A \right) + \left(A^H E_i A E_j - A^H E_j A E_i \right) \right) \right) \\
&\quad + \sum_{i=0}^{n-1} \alpha_{ii} \left(\left(E_i A^H E_i A - E_i A^H E_i A \right) + \left(A^H E_i A E_i - A^H E_i A E_i \right) \right) \\
&= \sum_{i=0}^{n-1} \sum_{j=i+1}^{n-1} \left(\alpha_{ij} - \alpha_{ji} \right) \left(\left(E_j A^H E_i A - E_i A^H E_j A \right) \right. \\
&\qquad \left. + \left(A^H E_j A E_i - A^H E_i A E_j \right) \right) . \qquad (5.40)
\end{aligned}
$$

But we have already shown that $\alpha_{ij} = \alpha_{ji}$, so $K + K^H = 0$. ■

Looking back at Equation 5.28, we see that in order to have a valid gradient flow, each Ω_k in Equation 5.32 must be multiplied on the left by $-\Omega_k^{\nabla\phi}$, where $\Omega_k^{\nabla\phi} \in T_e(M_k)$. Claim 5.3.1 tells us that the matrices which multiply the Ω_ks on the left are skew–hermitian. To put them in $T_e(M_k)$ we introduce a set of natural projection operators, $\Pi_k : T_e U(n) \to T_e(M_k)$. Π_k projects from the set of skew–symmetric matrices to the set of block diagonal skew–symmetric matrices by setting all off diagonal blocks to zero. Now we can use Fact 4.1.1 to state

$$
D\phi_\Psi(X) = \sum_{k=1}^{L} \operatorname{retr} \left(\Pi_k \left(\sum_{i=0}^{n-1} \left[\breve{\Theta}_k N_i^H \breve{\Theta}_k^H, \widetilde{\Theta}_k^H E_i \widetilde{\Theta}_k \right] \right) \Omega_k \right) . \qquad (5.41)
$$

Now

$$
\Pi_k \left(\sum_{i=0}^{n-1} \left[\breve{\Theta}_k N_i^H \breve{\Theta}_k^H, \widetilde{\Theta}_k^H E_i \widetilde{\Theta}_k \right] \right) \in T_e M_k \qquad (5.42)
$$

for each k, so the gradient flow which satisfies both properties is given by the assignment

$$\Omega_k^{\nabla\phi} = -\Pi_k \left(\sum_{i=0}^{n-1} \Pi_k \left[\breve{\Theta}_k N_i^H \breve{\Theta}_k^H, \widetilde{\Theta}_k^H E_i \widetilde{\Theta}_k \right] \right). \tag{5.43}$$

The gradient flow is then given by the couple matrix ODEs

$$\dot{\Theta}_k = -\Theta_k \Pi_k \left(\sum_{i=0}^{n-1} \left[\breve{\Theta}_k N_i^H \breve{\Theta}_k^H, \widetilde{\Theta}_k^H E_i \widetilde{\Theta}_k \right] \right), \quad \Theta_k(0) = \Theta_{k0}, \tag{5.44}$$

for $k = 1, 2, \ldots, L$.

5.3.3 Example

Here we apply this plant diagonalization technique to the example of a flexible beam with periodic boundary conditions. This system can be thought of as a flexible ring. The model we use is the beam model described and modeled in Appendix B with the boundary conditions at the ends of the beam replaced by periodic boundary conditions. In this example, we consider a beam with 8 inputs and 8 outputs. The sensors and actuators are evenly spaced around the ring so that the system has the spatial invariance property.

The plots in Figure 5.1 shows the responses of the 8 outputs to an impulse applied to the first input. Figure 5.2 depicts the circulant nature of the plant. In this figure, the intensity of the (i, j)th pixel represents the energy contained in the response of the ith output to an impulse applied to the jth input. We call this matrix the impulse energy matrix.

We found diagonalizing basis transformations as described in this chapter using 3, 5, and 7 level ROTs. Figures 5.3, 5.4, and 5.5 show the impulse energy matrices for these transforms. The coordinate transforms are complex so the resulting impulse responses are complex valued. To compute the impulse energy, we take the magnitude of the impulse response.

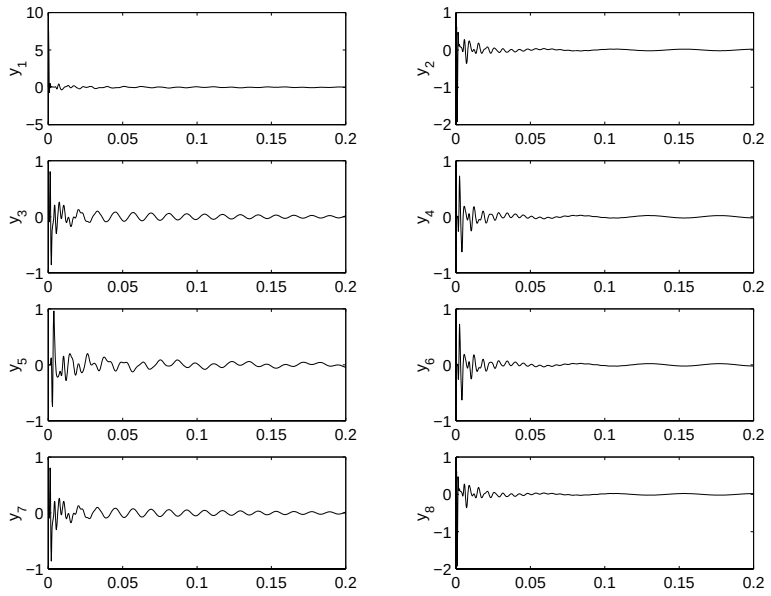


Figure 5.1: Plant response to impulse applied to first actuator.

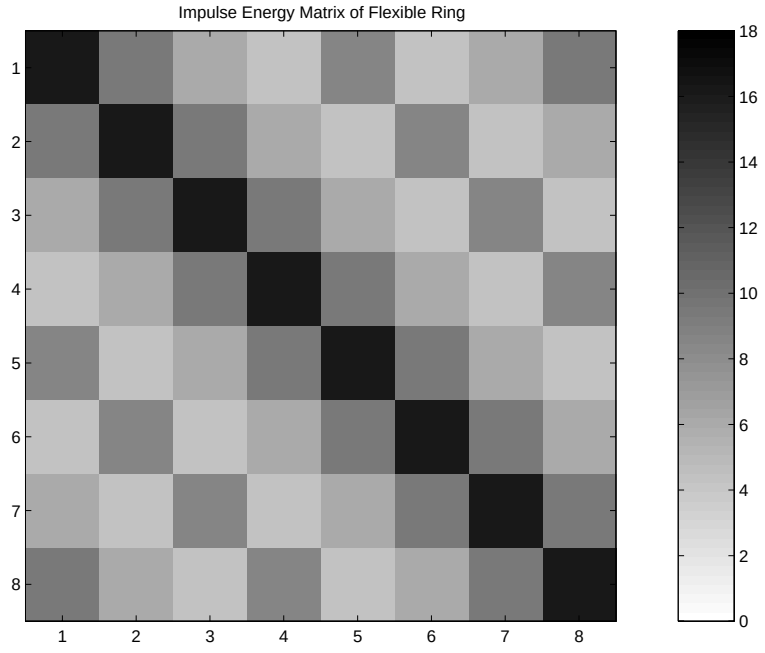


Figure 5.2: Impulse energy matrix for untransformed plant.

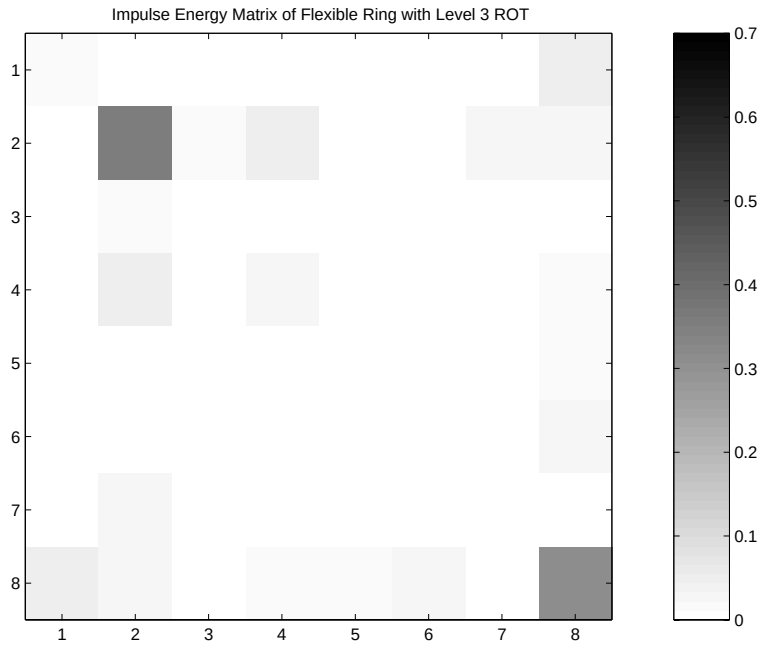


Figure 5.3: Impulse energy matrix for plant transformed with level 3 ROT.

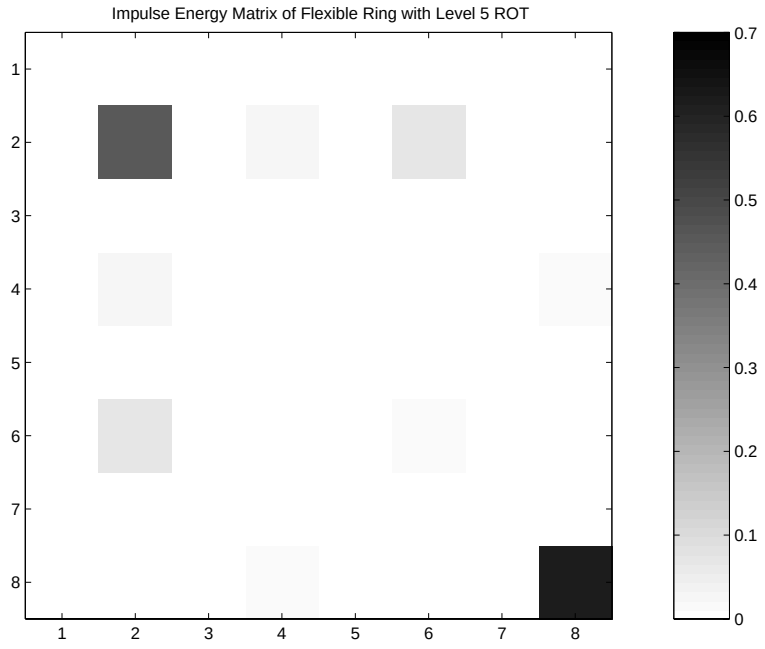


Figure 5.4: Impulse energy matrix for plant transformed with level 5 ROT.

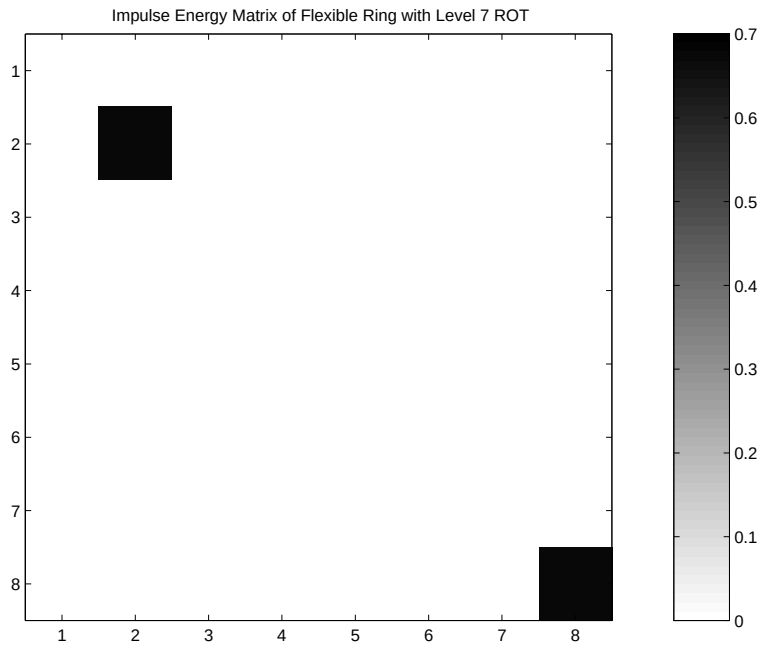


Figure 5.5: Impulse energy matrix for plant transformed with level 7 ROT.

Chapter 6

Conclusions

In this thesis, we have presented the foundations of a novel approach to signal processing and control on large distributed control networks. Our approach is to apply efficient data transformations to the input and output vector of a plant in an effort to make the resulting plant matrix look diagonal. In Chapters 2 and 4 we proposed efficient transforms capable of "approximately" diagonalizing any constant matrix. These ideas are useful for stable linear systems in which the transient behavior can be ignored as well as systems which exhibit strong, sharp peaks (resonances) in their frequency response. In Chapter 5 we developed a method of diagonalizing dynamic plants which exhibit a linear spatial invariance property.

We conclude the thesis with a quantitative comparison of the matrix diagonalization methods presented followed by some suggestions for future research.

6.1 Comparison of Proposed Approximate Diagonalization Methods

In this section we compare and contrast the methods of approximate matrix diagonalization that we have developed in this thesis. Our purpose is not only to highlight the merits of the proposed methods, but also to point out where there is room for improvement.

6.1.1 Performance Measures

Here we define the performance measures that will be used to compare the various data transformation techniques. The costs we consider are communication and computational costs associated with transforming the output vector. We also define a performance measure which indicates how well the proposed transforms diagonalize the plant matrix.

Global Data Transfers

One common network configuration is the common bus topology. In this configuration, all of the nodes on the network share one serial communication bus. The cost of a data transfer in this configuration does not depend on the positions of the sending receiving nodes. It is as expensive for a node to send information to its neighbor as it is to send information to a node on the other side of the network. In fact, for the same cost, the sending node can communicate its data to every node on the network. We call any such communication a global data transfer.

As an example, consider a 1–level ROT on a linear array as described in Section 3.2.1. Recall that the first (and only) communication step is for each evenly

indexed node to send the value of its sensor output to the node to its left. For an array with n nodes, this requires $n/2$ global data transfers.

Nearest Neighbor Data Transfers

To measure this communication cost, we assume that each node on the network can exchange information only with its nearest neighbors. The transfer of data from a node to its neighbor is called a nearest neighbor data transfer. Multiple data transfers are required to exchange information between non–neighboring nodes. For example, in order to send a piece of data from the ith node of a linear array to the jth node, a total of $|i - j|$ nearest neighbor data transfers are required.

Nearest Neighbor Transfer Time

In a nearest neighbor network, independent data transfers can be performed at the same time. We define the nearest neighbor communication time to be the total time required to carry out all communications, taking into account the fact that many can be carried out simultaneously. The unit used to measure this time is the amount of time required to carry out one nearest neighbor data transfer. We call this time a step.

To send a piece of data from the ith node of a linear array to the jth node, a total of $|i - j|$ nearest neighbor data transfers are required. These transfers cannot be performed in parallel, so the communication time is $|i - j|$ steps. On the other hand, a 1–level ROT performed on a linear array with n nodes requires $n/2$ nearest neighbor communications. All of these can be performed simultaneously, so the corresponding communication time is 1 step.

Implementation Multiplications

Here, we measure computational complexity by counting the total number of multiplications required to implement the output vector transformation.

Implementation Multiplication Time

Here we measure the amount of time required to carry out the multiplications for the output vector transformation. In this measure, we take into account the fact that some operations can be performed simultaneously. The unit of measurement is the time required to complete one multiplication. We call this time a step.

Off–line Multiplications

In addition to implementation costs, there is a cost associated with finding the transformation to be applied. We measure this cost by counting the number of multiplications required. We do not put as much weight on this cost as we do on the implementation costs; the philosophy of this thesis is that it is worth the extra off–line effort to improve the on–line costs. We include this cost in our comparison to give an indication of just how much extra off-line work is required.

Diagonal Energy Percentage

To help compare the performances of the various methods presented in this thesis, we develop a measure that indicates the "diagonalness" of an approximately diagonalized matrix. Let H be an $n \times m$ matrix. Let x be the vector whose elements are the diagonal elements of H, i.e. $x(i) = [H]_{ii}$ for $i = 1, 2, \ldots, \min(n, m)$. We define the diagonal energy percentage to be

$$\eta(H) = \frac{\|x\|_2}{\|H\|} \times 100, \tag{6.1}$$

where $\|H\|$ denotes the Frobenius norm of H.

6.1.2 Implementation of Exact SVD Factors

In Section 2.1 we pointed out that the SVD factors can be used to diagonalize a real–valued matrix. We also claimed that they require too much computation and communication to be implemented in real time on a large distributed control network. Here we briefly discuss such an implementation of the SVD factors so that we can quantify these remarks and compare the performance of SVD factors to the other matrix diagonalization techniques proposed in this thesis. We discuss only the implementation of the output vector transformation. The implementation of the input vector transformation can be inferred from this discussion.

The most straight forward method is what we refer to as the centralized implementation. In this implementation, each network node sends its piece of the output vector to a central location where the computations are carried out. If we assume that the n nodes on the network share a common bus allowing them to communicate with every node on the the network, then the communication step requires $(n-1)$ data transfers. If we instead assume that only nearest neighbor communication is required, then the fewest data transfers required to get all of the data to a central location is

$$N_c = 2 \sum_{k=1}^{n/2-1} k. \tag{6.2}$$

Once all of the data is collected, the computation which needs to be executed consists of the multiplication of an $n \times n$ matrix with an n–dimensional vector. This requires n^2 multiplications and $n(n-1)$ additions.

The SVD factors can also be realized using a parallel implementation. Here each node on the network must send its piece of the data vector to every other node

on the network. Then the matrix multiplication is carried out in parallel, with each node computing the multiplication of one row of the matrix with the data vector. If we assume that the n nodes on the network share a common bus allowing them to communicate with every node on the the network, then the communication step requires $(n-1)$ data transfers. If only nearest neighbor communication is allowed, then

$$N_p = 2\sum_{k=1}^{n-1} k \tag{6.3}$$

data transfers are required. Once all of the necessary transfers have been made, each node performs the multiplication of one row of the SVD factor by the data vector. The total number of operations necessary to complete the transform is the same as in the centralized implementation, but since the parallel implementation allows the row multiplications to be carried out simultaneously the computation time is reduced. Hence, the computations are completed in the time it takes to perform n multiplications and $(n-1)$ additions.

6.1.3 Comparison Tables

Table 6.1.3 lists these measures for a variety of approximate matrix diagonalization techniques applied to the plant matrix of a flexible membrane driven by a linear array. Table 6.1.3 compares the same techniques applied to the plant matrix of a flexible beam at its sixth resonance. In these tables, the SVD measures correspond to the parallel implementation of the SVD factors.

Though the costs and performances of the various transforms vary from example to example, we can make some generalizations about the comparisons. The wavelet packet transform generally require the lowest number of implementation multiplications, the lowest multiplication time, and the lowest off–line multiplica-

	SVD	Wavelet	ROT 10	ROT 20	ROT 31
Global Data Transfers	32	110	160	320	496
Nearest Neighbor Data Transfers	992	480	160	320	496
Nearest Neighbor Transfer Time	62 steps	40 steps	10 steps	20 steps	30 steps
Implementation Multiplications	1024	252	640	1280	1984
Implementation Mult. Time	32 steps	20 steps	40 steps	80 steps	124 steps
Off–line Multiplications	$\sim 3 \times 10^4$	$\sim 6 \times 10^3$	$\sim 10^{10}$	$\sim 10^{11}$	$\sim 10^{14}$
Diagonal Energy %	100%	93.4%	64.2%	88.7%	99.9%

Table 6.1: Comparison table for membrane example.

	SVD	Wavelet	ROT 10	ROT 20	ROT 31
Global Data Transfers	32	104	160	320	496
Nearest Neighbor Data Transfers	992	218	160	320	496
Nearest Neighbor Transfer Time	62 steps	26 steps	10 steps	20 steps	30 steps
Implementation Multiplications	1024	240	640	1280	1984
Implementation Mult. Time	32 steps	20 steps	40 steps	80 steps	124 steps
Off–line Multiplications	$\sim 3 \times 10^4$	$\sim 6 \times 10^3$	$\sim 10^{10}$	$\sim 10^{11}$	$\sim 10^{14}$
Diagonal Energy %	100%	60.6%	54.3%	69.7%	99.3%

Table 6.2: Comparison table for the example of a flexible beam at its 6th resonance.

tions. The ROTs exhibit the lowest nearest neighbor transfer time. The higher the level of the ROT, the better its performance in terms of diagonal energy, but this performance comes at the cost of every other measure listed.

6.2 Suggestions for Future Work

The main theoretical question that remains is the issue of the convergence of the ROT gradient flow equations and the characteristics of the equilibrium points to which these equations converge. We begin to address these issues in for the case of transforming a 4 dimensional data vector (See Appendix A), but a more thorough understanding is required. Such an understanding may yield bounds on the performance of an ROT or algorithms to select the "best" set of permutation matrices.

Another issue that needs to be addressed is the numerical integration of the ROT gradient flow equations. The computations required to find a 31–level, 32×32 ROT are almost prohibitive. More efficient numerical techniques are necessary if we are to find ROTs for larger systems. This boils down to the problem of integrating the nonlinear matrix ODE

$$\dot{\Theta} = \Theta\Omega(\Theta), \tag{6.4}$$

on $O(n)$ where $\Omega(\Theta) \in \mathfrak{o}(n)$ for every $\Theta \in O(n)$. It is important to use a numerical integration scheme which respects the constraint that Θ remains orthogonal. The recursive sequence

$$\Theta_{k+1} = \Theta_k \exp(\alpha\Omega(\Theta)) \tag{6.5}$$

provides one such integration method. This method is analogous to the Euler method, where the scalar α is the step size. Brockett [7] and Moore, Mahony, and

Helmke [29] have developed recursive methods for their work which combine this idea with a clever choice of α. These methods give fast convergence to the desired solution. A similar approach should be possible for the ROT equations. Other numerical gains might be found by using generalizations of the Cayley transform (see Tsiotras, Junkins, and Schaub [39]) to approximate the computationally intensive matrix exponential.

Throughout the thesis, we have assumed that the ROT parameters are fixed and given. In particular, we selected the parameters based on a nearest neighbor communication scheme. Almost certainly, better performance can be achieved by changing the ROT parameters. One idea towards this end is to use devise permutation matrices which mimic the communication schemes of the transforms in the wavelet packet. Alternatively, it may be possible to use more sophisticated searching algorithms to optimize the ROT variables and parameters simultaneously.

Appendix A

Analysis of ROTs on $SO(4)$

Here we study the some of the properties of a recursive orthogonal transform (ROT) composed of 4×4 matrices. The aim of this exercise is to provide some insight into the nature of the ROT and perhaps lay some groundwork for future research. Specifically, we examine the 3–Level ROT

$$\widetilde{\Theta} = \Theta_1 P_2 \Theta_2 P_3 \Theta_3. \tag{A.1}$$

Here,

$$\Theta_i = \begin{bmatrix} \theta_1^i & \mathbf{0}_{2\times 2} \\ \mathbf{0}_{2\times 2} & \theta_2^i \end{bmatrix}, \tag{A.2}$$

where $\theta_j^i \in SO(2)$ for $i \in \{1, 2, 3\}$, $j \in \{1, 2\}$. The permutation matrices are

$$P_2 = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix} \text{ and } P_3 = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}. \tag{A.3}$$

Equation A.1 can be implemented as a data transformation on a four node linear array using only nearest neighbor communication (See Section 3.2.1). Matrices of

the form given in Equation A.2 form a compact subgroup (torus) of $SO(4)$. We will denote this subgroup by M.

Each component in the product on the right hand side of Equation A.1 is in $SO(4)$, so $\widetilde{\Theta}$ is also in $SO(4)$. The question that naturally arises at this point is the following: what subset of $SO(4)$ can be represented by the above ROT? Or, more to the point, is it possible to represent any $\Phi \in SO(4)$ with a product of the form given in Equation A.1?

We know that the dimension of $SO(4)$ is six. In order to have any chance of representing all of $SO(4)$, $\widetilde{\Theta}$ must have at least six degrees of freedom which are, in some sense, independent. Each Θ_i in Equation A.1 has two degrees of freedom, so $\widetilde{\Theta}$ has a total of six. It is not clear that these six dimensions are independent, though. For example, if P_2 and P_3 were both the identity matrix, then $\widetilde{\Theta}$ could only represent elements in the two dimensional subgroup $M \subset SO(4)$.

To answer these questions we look to the theory of symmetric subalgebras. Here we give a brief presentation of some of these tools and then show how to apply them to the above ROT.

A.1 The KAK Representation

Here we present a brief introduction to the KAK representation of semisimple Lie groups. We follow Hermann [23] and present the main theorems without proof. Readers interested in the details of this material are referred to Hermann or Helgason [20].

Definition A.1.1 *Let G be a connected, semisimple Lie group with finite center. Let $\mathfrak{g}$ be the Lie algebra of G. Let $\mathfrak{k}$ be a subspace of $\mathfrak{g}$ and let $\mathfrak{p}$ be the complement of*

$\mathfrak{k}$ *in* $\mathfrak{g}$. *Then* $\mathfrak{k}$ *is said to be a* symmetric subalgebra *of* $\mathfrak{g}$ *if the following bracketing conditions hold:*

1. $[\mathfrak{k}, \mathfrak{k}] \subset \mathfrak{k}$.

2. $[\mathfrak{k}, \mathfrak{p}] \subset \mathfrak{p}$.

3. $[\mathfrak{p}, \mathfrak{p}] \subset \mathfrak{k}$.

.

Associated with the subalgebra $\mathfrak{k} \subset \mathfrak{g}$ is a subgroup $K \subset G$. The subgroup K is said to be a symmetric subgroup of G. Since G has a finite center, K is compact.

Since K is compact, the exponential map maps $\mathfrak{k}$ onto K [33]. As a result, for any $k \in K$ there exists $X \in \mathfrak{k}$ such that $k = \exp(X)$.

Theorem A.1.1 (Theorem 6–4 from Hermann [23])
Let G, K, $\mathfrak{g}$, $\mathfrak{k}$, *and* $\mathfrak{p}$ *be as defined above. Let* $\mathfrak{a}$ *be a maximal Abelian subalgebra of* $\mathfrak{p}$ *and let* $\mathfrak{a}'$ *be any Abelian subalgebra of* $\mathfrak{p}$. *Then there exists* $k \in K$ *such that*

$$k\mathfrak{a}'k^{-1} \subset \mathfrak{a}. \tag{A.4}$$

Theorem A.1.2 (Theorem 6–5 from Hermann [23])
Let G, K, $\mathfrak{g}$, $\mathfrak{k}$, *and* $\mathfrak{p}$ *be as defined above. Let* P *be the image of* $\mathfrak{p}$ *under the exponential map. Then any* $g \in G$ *can be represented as*

$$g = pk, \tag{A.5}$$

where $p \in P$ *and* $k \in K$.

These two theorems are combined to give us the so–called KAK representation of the group G. First, Theorem A.1.2 says that any $g \in G$ can be represented as

$\exp(X_p)k$ where $X_p \in \mathfrak{p}$ and $k \in K$. Now let $\mathfrak{a}$ be a maximal Abelian subalgebra of $\mathfrak{p}$. A direct result of Theorem A.1.1 is that every element of $\mathfrak{p}$ can be written $X_p = k_1 X_a k_1^{-1}$ where $k_1 \in K$ and $X_a \in \mathfrak{a}$. Substituting, we get

$$\begin{aligned} g &= \exp(k_1 X_a k_1^{-1})k \\ &= k_1 \exp(X_a) k_1^{-1} k. \end{aligned} \tag{A.6}$$

Since K is a group, $k_1^{-1}k \in K$. Letting A denote the image of $\mathfrak{a}$ under the exponential, we see that any $g \in G$ can be written as

$$g = k_1 a k_2, \tag{A.7}$$

where $k_1, k_2 \in K$ and $a \in A$. This is known as the KAK representation.

A.2 ROT Representation of $SO(4)$

To begin, we note that $SO(4)$ is a simple group [37], which means it is also semisimple. As a result, the theory developed in the previous section can be applied. Next, we define the following basis vectors for $\mathfrak{so}(4)$:

$$A_1 = \begin{bmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad A_2 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad A_3 = \begin{bmatrix} 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$A_4 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \quad A_5 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \quad A_6 = \begin{bmatrix} 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \tag{A.8}$$

Since M is compact, any $\Theta_i \in M$ can be written as the exponential of an element in the Lie algebra of M. Using the above basis vectors, we can write

$$\Theta_i = \exp(\alpha_1^i A_1 + \alpha_2^i A_4), \tag{A.9}$$

where α_1^i and α_2^i are real, $i \in \{1, 2, 3\}$. Substituting into Equation A.1, $\widetilde{\Theta}$ can be written

$$\begin{aligned}
\widetilde{\Theta} &= \exp(\alpha_1^1 A_1 + \alpha_2^2 A_4) P_2 \exp(\alpha_1^2 A_1 + \alpha_2^2 A_4) P_3 \exp(\alpha_1^3 A_1 + \alpha_2^3 A_4) \\
&= \exp(\alpha_1^1 A_1 + \alpha_2^2 A_4) P_2 \exp(\alpha_1^2 A_1 + \alpha_2^2 A_4) P_2^T \exp(\alpha_1^3 A_1 + \alpha_2^3 A_4) \\
&= \exp(\alpha_1^1 A_1 + \alpha_2^2 A_4) \exp(\alpha_1^2 P_2 A_1 P_2^T + \alpha_2^2 P_2 A_4 P_2^T) \exp(\alpha_1^3 A_1 + \alpha_2^3 A_4) \\
&= \exp(\alpha_1^1 A_1 + \alpha_2^2 A_4) \exp(-\alpha_1^2 A_6 + \alpha_2^2 A_2) \exp(\alpha_1^3 A_1 + \alpha_2^3 A_4).
\end{aligned} \tag{A.10}$$

Let $\mathfrak{k} = \text{span}\{A_1, A_4\}$. It is easy to check the bracket conditions in Definition A.1.1 to see that $\mathfrak{k}$ is a symmetric subalgebra of $\mathfrak{so}(4)$. Let $\mathfrak{p}$ be the complement of $\mathfrak{k}$ in $\mathfrak{so}(4)$, i.e. $\mathfrak{p} = \text{span}\{A_2, A_3, A_5, A_6\}$. Explicitly, the bracket $[A_1, A_4] = 0$ implies that $[\mathfrak{k}, \mathfrak{k}] \subset \mathfrak{k}$. The brackets

$$\begin{aligned}
[A_1, A_2] &= -A_3 & [A_4, A_2] &= A_5 \\
[A_1, A_3] &= A_2 & [A_4, A_3] &= A_6 \\
[A_1, A_5] &= -A_6 & [A_4, A_5] &= -A_2 \\
[A_1, A_6] &= A_5 & [A_4, A_6] &= -A_3
\end{aligned} \tag{A.11}$$

imply that $[\mathfrak{k}, \mathfrak{p}] \subset \mathfrak{p}$. Finally, the brackets

$$\begin{aligned}
[A_2, A_3] &= -A_1 & [A_3, A_5] &= 0 \\
[A_2, A_5] &= -A_4 & [A_3, A_6] &= A_4 \\
[A_2, A_6] &= 0 & [A_5, A_6] &= -A_1
\end{aligned} \tag{A.12}$$

imply that $[\mathfrak{p}, \mathfrak{p}] \subset \mathfrak{k}$. From the last set of brackets, is also clear that the subalgebra $\mathfrak{a} \triangleq \text{span}\{A_2, A_6\}$ is a maximal Abelian subalgebra of $\mathfrak{p}$.

Using the KAK representation, we see that any $\Phi \in SO(4)$ can be written as

$$\Phi = \exp(X_{k1})\exp(X_a)\exp(X_{k2}), \tag{A.13}$$

where $X_{k1}, X_{k2} \in \mathfrak{k}$ and $X_a \in \mathfrak{a}$. This expression is of exactly the same form as the expression for $\widetilde{\Theta}$ given in Equation A.10. Therefore, the ROT $\widetilde{\Theta}$ can be used to represent any $\Phi \in SO(4)$.

This representation exploits a somewhat unique feature of $\mathfrak{so}(4)$: its contains Abelian subalgebras which are also symmetric. This also holds for $\mathfrak{so}(3)$, but it does not in general hold for $\mathfrak{so}(n)$ where $n > 4$. This means that the above analysis does not easily extend to address ROTs on higher dimensional spaces.

A.3 Convergence of ROT Equations

In the previous sections we showed that a 3–level ROT given in Equation A.1 can be used to represent any matrix in $SO(4)$. This means that given any symmetric 4×4 matrix H_0, there exists a $\widetilde{\Theta}$ of the form given in Equation A.1 so that the matrix $H = \widetilde{\Theta}^T H_0 \widetilde{\Theta}$ is diagonal. However, this does not necessarily mean that the gradient flow equations we derived to find the ROT variables in Chapter 4 converge to $\widetilde{\Theta}$. Hence, it is necessary to study the convergence properties of the ROT gradient flow equations.

As we pointed out in Chapter 4, the system of ROT equations evolves on a compact manifold. By the properties of a gradient flow on a compact manifold, the solution to the ROT equations exists for all time and converges to an equilibrium point of the system. The first step is understanding the convergence properties of the system of ROT equations is to investigate the nature of the equilibrium points of the system. Unfortunately this is a tedious task, and even in the relatively

simple case of 4×4 matrices it is difficult to get clear answers. Here we analyze the equilibrium points of the ROT equations on 4×4 matrices for 1–,2–, and 3–level ROTs. In particular, we study the characteristics of the approximately diagonalized matrix which result.

A.3.1 1–Level ROT

For a 1–level ROT, we simply have

$$\widetilde{\Theta} = \Theta_1, \tag{A.14}$$

where Θ_1 is of the form given in Equation A.2. Following Section 4.1.2, the gradient flow equation to search for the ROT variables which most nearly diagonalize the symmetric matrix H_0 is

$$\dot{\Theta}_1 = -\Theta_1 \Pi \left[N, \Theta_1^T H_0 \Theta_1 \right], \quad \Theta_1(0) = \Theta_{10}, \tag{A.15}$$

where N is a 4×4 diagonal matrix with distinct entries and Π is the projection operator,

$$\Pi \left(\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \right) = \begin{bmatrix} a_{11} & a_{12} & 0 & 0 \\ a_{21} & a_{22} & 0 & 0 \\ 0 & 0 & a_{33} & a_{34} \\ 0 & 0 & a_{43} & a_{44} \end{bmatrix}. \tag{A.16}$$

Let Θ_{1*} be an equilibrium point of Equation A.15 and let $H = \Theta_{1*}^T H_0 \Theta_{1*}$ be the resulting approximately diagonalized matrix. Since Θ_{1*} is nonsingular, we have

$$\Pi \left[N, H \right] = 0. \tag{A.17}$$

In Section 4.1.1 we showed that the (i, j)th element of the skew–symmetric matrix $[N, H]$ is $h_{ij} (n_i - n_j)$. Equation A.17 tells us that the $(1, 2)$th and $(3, 4)$th elements

of $[N, H]$ are zero. Since the elements of N are distinct, we have

$$\begin{aligned} h_{12} &= h_{21} = 0 \\ h_{34} &= h_{43} = 0 \end{aligned} \tag{A.18}$$

. Hence, the gradient flow equation for the 1–level ROT is guaranteed to converge to a Θ_{1*} such that the resulting approximately diagonalized matrix H has $h_{12} = h_{34} = 0$.

A.3.2 2–Level ROT

For the 2–level ROT we have

$$\widetilde{\Theta} = \Theta_1 P_2 \Theta_2, \tag{A.19}$$

where Θ_1 and Θ_2 are of the form given in Equation A.2 and P_2 is given by Equation A.3. Following Section 4.1.2, the gradient flow equations to search for the ROT variables which most nearly diagonalize the symmetric matrix H_0 are

$$\begin{aligned} \dot{\Theta}_1 &= -\Theta_1 \Pi \left[P_2 \Theta_2 N \Theta_2^T P_2^T, \Theta_1^T H_0 \Theta_1 \right], \qquad \Theta_1(0) = \Theta_{10} \\ \dot{\Theta}_2 &= -\Theta_2 \Pi \left[N, \Theta_2^T P_2^T \Theta_1^T H_0 \Theta_1 P_2 \Theta_2 \right], \qquad \Theta_2(0) = \Theta_{20}. \end{aligned} \tag{A.20}$$

Let $(\Theta_{1*}, \Theta_{2*})$ be an equilibrium point for this system of equations and let

$$H = \Theta_{2*}^T P_2^T \Theta_{1*}^T H_0 \Theta_{1*} P_2 \Theta_{2*} \tag{A.21}$$

be the resulting approximately diagonalized matrix. Looking at the equation for $\dot{\Theta}_2$ and following the calculations from the previous section, we see that $h_{12} = h_{21} = h_{34} = h_{43} = 0$. Looking at the Equation for $\dot{\Theta}_1$, we see that

$$\Pi \left[P_2 \Theta_{2*} N \Theta_{2*}^T P_2^T, \Theta_1^T H_0 \Theta_1 \right] = 0. \tag{A.22}$$

Using some algebraic manipulation we have

$$\begin{aligned}
&\left[P_2\Theta_{2*}N\Theta_{2*}^T P_2^T, \Theta_{1*}^T H_0 \Theta_{1*}\right] \\
&= \left[P_2\Theta_{2*}N\Theta_{2*}^T P_2^T, P_2\Theta_{2*}\Theta_{2*}^T P_2^T \Theta_{1*}^T H_0 \Theta_{1*} P_2 \Theta_{2*}\Theta_{2*}^T P_2^T\right] \\
&= P_2\Theta_{2*}\left[N, H\right]\Theta_{2*}^T P_2^T. \qquad \text{(A.23)}
\end{aligned}$$

We can write Θ_{2*} as a matrix using sines and cosines,

$$\Theta_{2*} = \begin{bmatrix} cos(\vartheta_{21}) & -sin(\vartheta_{21}) & 0 & 0 \\ sin(\vartheta_{21}) & cos(\vartheta_{21}) & 0 & 0 \\ 0 & 0 & cos(\vartheta_{22}) & -sin(\vartheta_{22}) \\ 0 & 0 & sin(\vartheta_{22}) & cos(\vartheta_{22}) \end{bmatrix}. \qquad \text{(A.24)}$$

. To simplify upcoming long expressions, we will use s_{ij} and c_{ij} to denote $sin(\vartheta_{ij})$ and $cos(\vartheta_{ij})$, respectively.

Now, the equilibrium condition means that the $(1,2)$th and $(3,4)$th elements of the matrix $P_2\Theta_{2*}\left[N, H\right]\Theta_{2*}^T P_2^T$ must be equal to zero. Substituting Equation A.24 into this expression and evaluating using the Maple kernel of MATLAB yields the following two equations:

$$\begin{aligned}
&h_{13}s_{22}c_{21}(n_3 - n_1) + h_{14}c_{22}c_{21}(n_4 - n_1) \\
&\qquad + h_{23}s_{22}s_{21}(n_2 - n_3) + h_{24}c_{22}s_{21}(n_2 - n_4) = 0 \\
&h_{13}c_{22}s_{21}(n_3 - n_1) + h_{14}s_{22}s_{21}(n_4 - n_1) \\
&\qquad + h_{23}c_{22}c_{21}(n_2 - n_3) + h_{24}s_{22}c_{21}(n_2 - n_4) = 0.
\end{aligned} \qquad \text{(A.25)}$$

Hence, the gradient flow equations for the 2–level ROT are guaranteed to converge to an equilibrium point $(\Theta_{1*}, \Theta_{2*})$ such that the resulting approximately diagonalized matrix H has

1. $h_{12} = h_{34} = 0$

2. The vector $[h_{13}h_{14}h_{23}h_{24}]^T$ is in the null space of the matrix

$$\begin{bmatrix} s_{22}c_{21}(n_3-n_1) & c_{22}c_{21}(n_4-n_1) & s_{22}s_{21}(n_2-n_3) & c_{22}s_{21}(n_2-n_4) \\ c_{22}s_{21}(n_3-n_1) & s_{22}s_{21}(n_4-n_1) & c_{22}c_{21}(n_2-n_3) & s_{22}c_{21}(n_2-n_4) \end{bmatrix}$$

A.4 3–Level ROT

Here we consider the 3–level ROT given by Equation A.1. Following Section 4.1.2, the gradient flow equations to search for the ROT variables which most nearly diagonalize the symmetric matrix H_0 are

$$\dot{\Theta}_1 = -\Theta_1\Pi\left[P_2\Theta_2P_3\Theta_3N\Theta_3^TP_3^T\Theta_2^TP_2^T, \Theta_1^TH_0\Theta_1\right], \qquad \Theta_1(0) = \Theta_{10} \quad \text{(A.26)}$$

$$\dot{\Theta}_2 = -\Theta_2\Pi\left[P_3\Theta_3N\Theta_3^TP_3^T, \Theta_2^TP_2^T\Theta_1^TH_0\Theta_1P_2\Theta_2\right], \qquad \Theta_2(0) = \Theta_{20} \quad \text{(A.27)}$$

$$\dot{\Theta}_3 = -\Theta_3\Pi\left[N, \Theta_3^TP_3^T\Theta_2^TP_2^T\Theta_1^TH_0\Theta_1P_2\Theta_2P_3\Theta_3\right], \qquad \Theta_3(0) = \Theta_{30}. \quad \text{(A.28)}$$

Let $(\Theta_{1*}, \Theta_{2*}, \Theta_{3*})$ be an equilibrium point for this system of equations and let

$$H = \Theta_{3*}^TP_3^T\Theta_{2*}^TP_2^T\Theta_{1*}^TH_0\Theta_{1*}P_2\Theta_{2*}P_3\Theta_{3*} \quad \text{(A.29)}$$

be the resulting approximately diagonalized matrix. Looking at the equation for $\dot{\Theta}_3$ and following the calculations from the Section A.3.1, we see that $h_{12} = h_{21} = h_{34} = h_{43} = 0$. Looking at the equation for $\dot{\Theta}_2$ and following t the calculations of Section A.3.2 we get the equations

$$\begin{aligned} h_{13}s_{32}c_{31}(n_3-n_1) + h_{14}c_{32}c_{31}(n_4-n_1) \\ +h_{23}s_{32}s_{31}(n_2-n_3) + h_{24}c_{32}s_{31}(n_2-n_4) = 0 \end{aligned} \quad \text{(A.30)}$$

$$\begin{aligned} h_{13}c_{32}s_{31}(n_3-n_1) + h_{14}s_{32}s_{31}(n_4-n_1) \\ +h_{23}c_{32}c_{31}(n_2-n_3) + h_{24}s_{32}c_{31}(n_2-n_4) = 0, \end{aligned} \quad \text{(A.31)}$$

where s_{3j} and c_{3j} are the sines and cosines that result from writing Θ_{3*} as a matrix of sines and cosines (see Equation A.24).

Now, looking at the Equation for $\dot{\Theta}_1$, we see that

$$\Pi\left[P_3\Theta_{3*}P_2\Theta_{2*}N\Theta_{2*}^TP_2^T\Theta_{3*}^TP_3^T, \Theta_{1*}^TH_0\Theta_{1*}\right] = 0. \tag{A.32}$$

Using some algebraic manipulation we see that

$$\left[P_2\Theta_{2*}N\Theta_{2*}^TP_2^T, \Theta_{1*}^TH_0\Theta_{1*}\right] = P_2\Theta_{2*}P_3\Theta_{3*}\left[N, H\right]\Theta_{3*}^TP_3^T\Theta_{2*}^TP_2^T. \tag{A.33}$$

The equilibrium condition means that the $(1,2)$th and $(3,4)$th elements of the matrix $P_2\Theta_{2*}P_3\Theta_{3*}\left[N, H\right]\Theta_{3*}^TP_3^T\Theta_{2*}^TP_2^T$ must be equal to zero. Using the sine/cosine matrix representations of Θ_{2*} and Θ_{3*} along with the Maple kernel of MATLAB yields the equations

$$\begin{aligned}
&h_{13}(s_{21}s_{32}s_{31}c_{22} + c_{21}c_{32}c_{31}s_{22})(n_3 - n_1)\\
&\qquad +h_{14}(s_{21}c_{32}s_{31}c_{22} - c_{21}s_{32}c_{31}s_{22})(n_4 - n_1)\\
&\qquad +h_{23}(s_{21}s_{32}c_{31}c_{22} - c_{21}c_{32}s_{31}s_{22})(n_3 - n_2)\\
&\qquad +h_{24}(s_{21}c_{32}c_{31}c_{22} + c_{21}s_{32}s_{31}s_{22})(n_4 - n_2) = 0
\end{aligned} \tag{A.34}$$

$$\begin{aligned}
&h_{13}(c_{21}s_{32}s_{31}s_{22} + s_{21}c_{32}c_{31}c_{22})(n_1 - n_3)\\
&\qquad +h_{14}(c_{21}c_{32}s_{31}s_{22} - s_{21}s_{32}c_{31}c_{22})(n_1 - n_4)\\
&\qquad +h_{23}(c_{21}s_{32}c_{31}s_{22} - s_{21}c_{32}s_{31}c_{22})(n_2 - n_3)\\
&\qquad +h_{24}(c_{21}c_{32}c_{31}s_{22} + s_{21}s_{32}s_{31}c_{22})(n_2 - n_4) = 0.
\end{aligned} \tag{A.35}$$

Since h_{13}, h_{14}, h_{23}, and h_{24} all enter linearly into Equations A.30, A.31, A.34, and A.35, we can rewrite these equations as

$$\Gamma\begin{bmatrix} h_{13} \\ h_{14} \\ h_{23} \\ h_{24} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \tag{A.36}$$

where Γ is a 4×4 matrix. When the matrix Γ is full rank, then we must have $h_{13} = h_{14} = h_{23} = h_{24} = 0$ at the equilibrium points of the ROT gradient flow

equations. From the $\dot{\Theta}_3$ equation we already have that $h_{12} = h_{34} = 0$, So when Γ is full rank, the matrix H must be diagonal.

Using the Maple kernel of MATLAB, we see that the determinant of Γ is

$$\det(\Gamma) = (n_4 - n_1)(n_2 - n_4)(n_3 - n_1)(n_3 - n_2)(c_{22}^2 s_{21}^2 - s_{22}^2 c_{21}^2). \tag{A.37}$$

Hence, the determinant is nonzero and Γ is full rank unless

$$\cos^2(\vartheta_{22})\sin^2(\vartheta_{21}) = \sin^2(\vartheta_{22})\cos^2(\vartheta_{21}). \tag{A.38}$$

This failure in rank leads to the possibility that the matrix H has nonzero off–diagonal elements.

We can take some comfort in the fact that Γ is nonsingular "almost everywhere", i.e. the set on which Γ is singular is a thin subset of the configuration space of the ROT variables. It is tempting to state that, for most cases, the gradient flow equations for the 3–level ROT to converge to an equilibrium point $(\Theta_{1*}, \Theta_{2*}, \Theta_{3*})$ such that the resulting H is diagonal. Based on our numerical experiments, this fact seems to be true. However the theoretical argument outlined above is incomplete. It is entirely possible that the thin set on which Γ is singular is an attractor for the system of ROT equations. In order to complete our argument, it is necessary to rule out the possibility that the ROT equations converge to an equilibrium point which lies on this thin set. This remains as an open problem.

Appendix B

Flexible Beam Model

This appendix develops a model for a flexible cantilever beam. The beam is driven by a number of surface mounted PZT actuators and the beam has a number of outputs given by surface mounted PZT sensors. We assume that the beam undergoes pure bending and satisfies the Euler–Bernoulli condition (linear strain distribution). From these assumptions, we use first principles to derive a PDE model to describe the dynamics of the beam. The PDE is then discretized using the finite difference method to give an ODE model. Simulation results using this ODE model are then presented.

B.1 Simple Beam Model

Here we derive the equations of motion for transverse vibrations in a cantilever beam. The beam has length L, width b, and thickness t_b. We assume that $L \gg b \gg t_c$. At rest, the beam is oriented as depicted in Figure B.1. The clamped end of the beam is at $x = 0$ and the free end is at $x = L$.

It is necessary to introduce some notation with which we can describe the beam

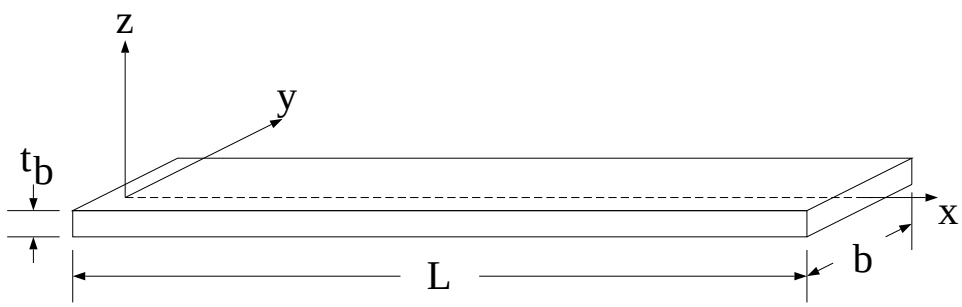


Figure B.1: Rest configuration of beam.

when it is deformed in some way. One common method of doing this is to describe the displacement of each point in the beam using rest position as a reference point. Since there are a continuum of points within the beam, these descriptions take the form of continuous functions in x, y, and z. In order to describe how the beam evolves over time, these functions must also be time dependent. Let x, y, and z be the coordinates of a point P in the beam at rest. Define the x component of the displacement of the point P at time t by $u(x, y, z, t)$. Likewise, let the y and z components be denoted by $v(x, y, z, t)$ and $w(x, y, z, t)$, respectively.

For the current case, only transverse vibrations are considered. We assume that there is no displacement in the y direction which means that v is identically zero. We assume that the beam does not undergo extension which means that u is identically zero. We also assume that there is no torsion in the beam, so w is not dependent on y. Since the beam is thin, we assume that the displacement function does not vary across the thickness of the beam. As a result of these assumptions, the configuration of the beam can be completely described using the function $w = w(x, t)$, as shown in Figure B.2.

At time t, we define the *neutral surface* to be $\{z|z = w(x, t), x \in [0, L]\}$. The strain in the neutral surface is zero as a result of the "no extension" assumption.

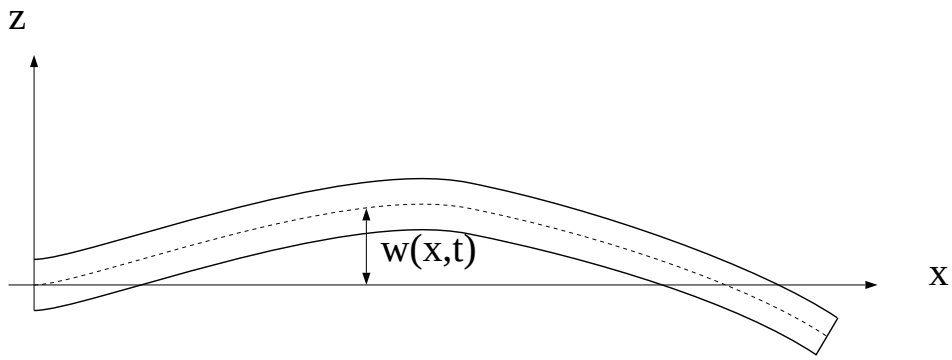


Figure B.2: Beam displacement.

This is shown in Figure B.2. Away from the neutral surface, the strain is assumed to be axial; shear strain is neglected.

We also assume that the beam is an Euler–Bernoulli beam. This assumption means that each cross section perpendicular to the x axis for the resting beam remains perpendicular to the neutral surface for all time. This assumption can be used to find a formula for the strain at any point in the beam. Figure B.3 depicts a differential element of the beam undergoing some deformation. The cross sections at x and $x + dx$ are perpendicular to the neutral surface, which is shown here as an arc connecting the two points. The angle between the two cross sections, θ, is shown to be equal to the negative curvature of the beam,

$$\theta = -\frac{\partial^2 w}{\partial x^2}(x)dx. \tag{B.1}$$

The radius of curvature, R, can be found using the relationship

$$\frac{dx}{2\pi R} = \frac{\theta}{2\pi} = -\frac{1}{2\pi}\frac{\partial^2 w}{\partial x^2}(x)dx, \tag{B.2}$$

which leads to the following expression for R:

$$R = -\left(\frac{\partial^2 w}{\partial x^2}(x)\right)^{-1}. \tag{B.3}$$

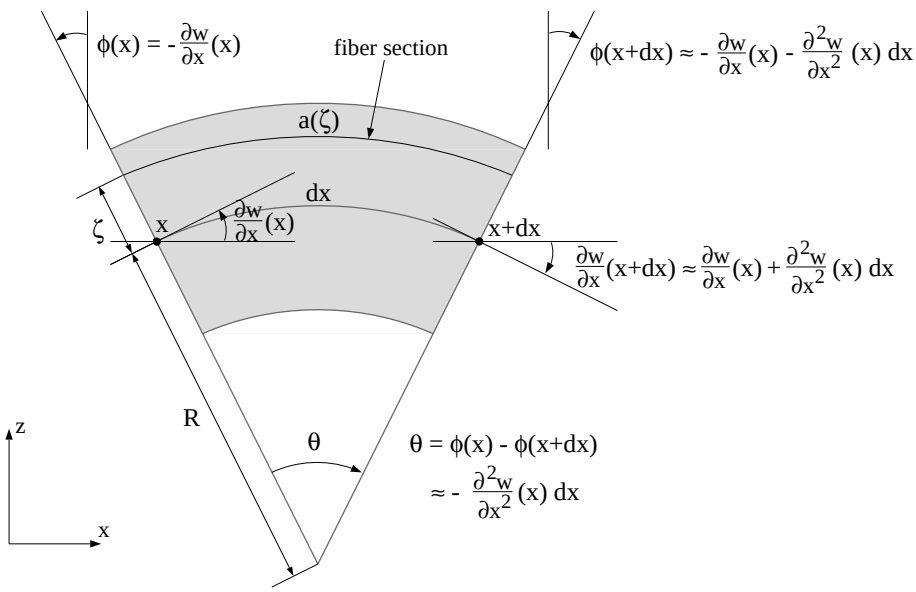


Figure B.3: Kinematic diagram of differential beam element used to determine strain.

To calculate the strain at a point on the cross section, it is necessary to look at the change in length of the "fiber" passing through that point. A fiber is defined to be a set of points in the resting beam which form a line parallel to the x axis. When the beam is deformed, the fibers are also deformed. In Figure B.3, the arc labeled "fiber section" represents the piece of a deformed fiber which is contained in the differential element. The fiber is displaced from the neutral surface by ζ. The length of the arc is denoted by $a(\zeta)$. In the undeformed beam, the length of the fiber section is dx, which is the same as the length of the differential element. In the deformed beam, we can obtain an expression for $a(\zeta)$ using the relationship

$$\frac{a(\zeta)}{2\pi(R+\zeta)} = \frac{\theta}{2\pi} = -\frac{1}{2\pi}\frac{\partial^2 w}{\partial x^2}(x)dx, \tag{B.4}$$

which yields

$$a(\zeta) = dx - \zeta\frac{\partial^2 w}{\partial x^2}(x)dx. \tag{B.5}$$

The strain in the fiber section, which is defined to be the change in length divided

by the undeformed length, is given by

$$\begin{aligned}\epsilon(\zeta) &= \frac{a(\zeta) - dx}{dx} \\ &= -\zeta \frac{\partial^2 w}{\partial x^2}(x).\end{aligned}$$

Therefore, at time t the strain at a point P in the perpendicular cross section at x is normal to the cross section with magnitude

$$\epsilon(\zeta, t) = -\zeta \frac{\partial^2 w}{\partial x^2}(x, t), \tag{B.6}$$

where ζ is the displacement between P and the neutral surface.

To get the relationship between stress and strain, we use a version of Hooke's law modified to account for viscous damping within the beam [4]. This law yields the stress on the cross section as

$$\begin{aligned}\sigma_b(\zeta) &= E_b \epsilon + E_b^* \frac{\partial \epsilon}{\partial t} \\ &= -E_b \zeta \frac{\partial^2 w}{\partial x^2}(x, t) - E_b^* \zeta \frac{\partial^3 w}{\partial t \partial x^2}(x, t),\end{aligned} \tag{B.7}$$

where E_b and E_b^* are, respectively, Young's modulus and the viscous damping constant for the beam material. Here the subscript b emphasizes that σ_b, E_b, and E_b^* correspond to the *beam*. The reason for this emphasis will become apparent in Section B.2. The force acting on a differential area dA of the beam cross section is then $\sigma_b(\zeta) dA$. This is shown in Figure B.4. Here we have chosen the convention that a normal force directed away from the cross section is positive.

The bending moment acting on the cross section can now be computed.

$$\begin{aligned}M_b(x, t) &= -\int_A \sigma_b(\zeta) \zeta \quad dA \\ &= -\int_{-t_b/2}^{t/2} \sigma_b(\zeta) b \zeta d\zeta\end{aligned}$$

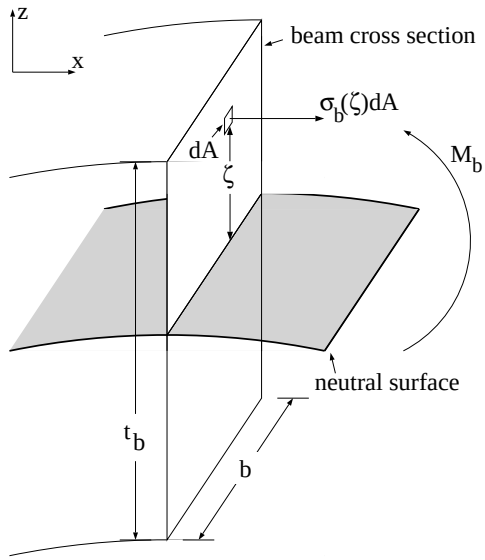


Figure B.4: Strain in beam cross section.

$$
\begin{aligned}
&= -\int_{-t_b/2}^{t/2} \left(E_b \epsilon + E_b^* \frac{\partial \epsilon}{\partial t} \right) b\zeta d\zeta \\
&= \int_{-t_b/2}^{t/2} \left(E_b \zeta \frac{\partial^2 w}{\partial x^2} + E_b^* \frac{\partial}{\partial t} (\zeta \frac{\partial^2 w}{\partial x^2}) \right) b\zeta d\zeta \\
&= (E_b \frac{\partial^2 w}{\partial x^2} + E_b^* \frac{\partial^3 w}{\partial t \partial x^2}) \int_{-t_b/2}^{t/2} b\zeta^2 d\zeta \\
&= E_b I_b \frac{\partial^2 w}{\partial x^2} + E_b^* I_b \frac{\partial^3 w}{\partial t \partial x^2}, \qquad \text{(B.8)}
\end{aligned}
$$

where I_b is the moment of inertia of the cross section. In this case, $I_b = bt_b^3/12$.

Now we consider the forces and moments acting on an interior differential element of the beam as shown in Figure B.5. Balancing the forces, we get

$$
\begin{aligned}
S - (S + \frac{\partial S}{\partial x} dx) - \rho b t_b dx \frac{\partial^2 w}{\partial t^2} &= 0 \\
\Longrightarrow \rho b t_b \frac{\partial^2 w}{\partial t^2} &= -\frac{\partial S}{\partial x}. \qquad \text{(B.9)}
\end{aligned}
$$

Ignoring the rotational inertia of the differential element, the moment balance

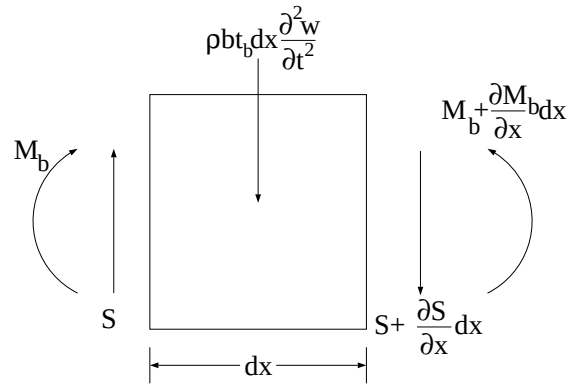


Figure B.5: Interior differential beam element and the forces acting on it.

equation is

$$\begin{aligned} M_b + Sdx - (M_b + \frac{\partial M_b}{\partial x}dx) &= 0 \\ \Longrightarrow S = \frac{\partial M_b}{\partial x}. \end{aligned} \tag{B.10}$$

Combining Equations B.9 and B.10 yields

$$\rho b t_b \frac{\partial^2 w}{\partial t^2} = -\frac{\partial^2 M_b}{\partial x^2}. \tag{B.11}$$

Substituting the expression for the bending moment from Equation B.8 into Equation B.11 gives the equation of motion for the interior of the beam. This equation is

$$\rho b t_b \frac{\partial^2 w}{\partial t^2} = -E_b I_b \frac{\partial^4 w}{\partial^4 x} - E_b^* I_b \frac{\partial^5 w}{\partial t \partial x^4} \tag{B.12}$$

At the clamped end of the beam, the displacement and slope are both zero. This yields the boundary conditions

$$w(0,t) = 0 \tag{B.13}$$

$$\frac{\partial w}{\partial x}(0,t) = 0 \tag{B.14}$$

for all time $t > 0$. There is no moment applied to the free end. This implies that

$$E_b I_b \frac{\partial^2 w}{\partial x^2}(L,t) + E_b^* I_b \frac{\partial^3 w}{\partial t \partial x^2}(L,t) = 0 \quad \forall t > 0. \tag{B.15}$$

There is also no shear force at the free end, which implies

$$E_b I_b \frac{\partial^3 w}{\partial x^3}(L,t) + E_b^* I_b \frac{\partial^4 w}{\partial t \partial x^3}(L,t) = 0 \quad \forall t > 0. \tag{B.16}$$

Given initial conditions $w(x,0)$ and $\frac{\partial w}{\partial t}(x,0)$ for $x \in [0,L]$, Equations B.12 through B.16 give a complete description of the deformation of the beam for all $t > 0$. [1]

B.2 Sandwiched Beam Model

In this section, we derive the equations of motion for transverse vibrations in a flexible beam which is sandwiched between two PZT actuators. The actuators are mounted with opposite polarity so that applying the same voltage to both actuators induces pure bending (no extension) in the beam.

A voltage applied across the terminals of a PZT actuator induces a stress tensor field within the crystal, which in turn induces a strain in the crystal. In this model, we consider only the component of stress in the axial direction; all other components are ignored. An applied voltage V_c will create an internal stress of

$$\sigma_c^{top} = -d_{31} E_c \frac{V_c}{t_c} + E_c \epsilon_c, \tag{B.19}$$

in the top crystal and

$$\sigma_c^{bottom} = d_{31} E_c \frac{V_c}{t_c} + E_c \epsilon_c, \tag{B.20}$$

[1] If the damping is small compared to the stiffness ($E_b \gg E_b^*$), boundary conditions B.15 and B.16 can be approximated by the simplified boundary conditions

$$\frac{\partial^2 w}{\partial x^2}(L,t) = 0 \tag{B.17}$$

and

$$\frac{\partial^3 w}{\partial^3 x}(L,t) = 0, \tag{B.18}$$

respectively, for all $t > 0$.

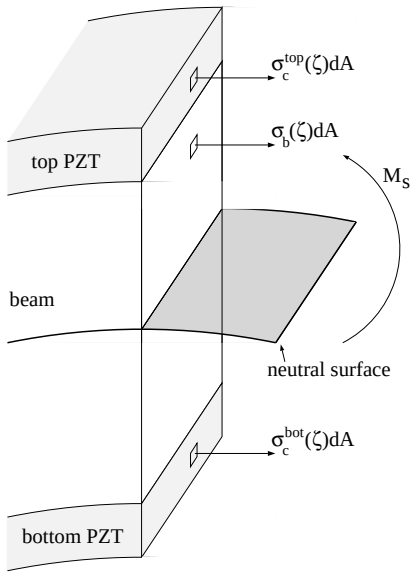


Figure B.6: Strain in sandwich cross section.

in the bottom crystal [13]. In these equations d_{31} is the piezoelectric strain coefficient, E_c is Young's modulus for the PZT crystal, t_c is the thickness of the PZT, and ϵ_c is the axial strain in the PZT. We neglect the viscous damping term in the crystal for simplicity, we will keep the viscous damping term in the beam material. The parameter d_{31} is assumed to be negative so that a positive voltage causes the top crystal to contract while the bottom expands, creating a positive curvature.

We assume that the Euler–Bernoulli bending assumption holds in the crystal as well as the beam, so the strain in the crystal is a linear function of beam curvature and distance from the neutral surface,

$$\epsilon_c(\zeta) = -\zeta \frac{\partial^2 w}{\partial x^2}. \tag{B.21}$$

Now we can compute the bending moment acting on the sandwich cross section. The forces used to compute the moment are shown in Figure B.6.

$M_s(x,t)$

$$
\begin{aligned}
&= -\int_A \sigma(\zeta)\zeta dA \\
&= -\left[\int_{-t_b/2-t_c}^{-t_b/2} \sigma_c^{bottom}(\zeta) b\zeta d\zeta + \int_{-t_b/2}^{t/2} \sigma_b(\zeta) b\zeta d\zeta + \int_{t_b/2}^{t_b/2+t_c} \sigma_c^{top}(\zeta) b\zeta d\zeta\right] \\
&= -2\int_{t_b/2}^{t_b/2+t_c} \sigma_c^{top}(\zeta) b\zeta d\zeta + M_b \\
&= -2\int_{t_b/2}^{t_b/2+t_c} \left(-d_{31}E_c\frac{V_c}{t_c} - E_c\frac{\partial^2 w}{\partial x^2}\zeta\right) b\zeta d\zeta + M_b \\
&= \frac{2bd_{31}E_cV_c}{t_c}\int_{t_b/2}^{t_b/2+t_c} \zeta d\zeta + 2bE_c\frac{\partial^2 w}{\partial x^2}\int_{t_b/2}^{t_b/2+t_c} \zeta^2 d\zeta + M_b \\
&= bd_{31}E_c(t_b+t_c)V_c + E_cI_c\frac{\partial^2 w}{\partial x^2} + M_b, \qquad \text{(B.22)}
\end{aligned}
$$

where M_b is given by Equation B.8 and I_s is the moment of inertia of the crystal cross sections about the neutral axis. In this case, $I_c = (b(2t_c + t_b)^3 - bt_b^3)/12$. [2]

As in the case of the simple beam, the forces and moments balance to give

$$
(2\rho_c t_c + \rho_b t_b)\, b\frac{\partial^2 w}{\partial t^2} = -\frac{\partial^2 M_s}{\partial x^2}. \qquad \text{(B.23)}
$$

Substituting the expression for M_s from Equation B.22 into Equation B.23 yields the equation of motion for the interior of the sandwich.

$$
(2\rho_c t_c + \rho_b t_b)\, b\frac{\partial^2 w}{\partial t^2} = -\left(E_cI_c + E_bI_b\right)\frac{\partial^4 w}{\partial x^4} - E_b^* I_b\frac{\partial^5 w}{\partial t\partial x^4}. \qquad \text{(B.24)}
$$

Note that the forcing term in M_s due to the PZT voltage is constant in x and disappears when differentiated. This term only affects the boundary elements of the sandwich. Specifically, at $x = L$ there is no externally applied moment, which means

$$
bd_{31}E_c(t_b+t_c)V_c + \left(E_cI_c + E_bI_b\right)\frac{\partial^2 w}{\partial x^2}(L,t) + E_b^* I_b\frac{\partial^3 w}{\partial t\partial x^2}(L,t) = 0 \qquad \text{(B.25)}
$$

[2] Here, we have assumed that the crystal thickness t_c constant and that the width of the crystal is everywhere equal to the width of the beam. In some applications, it may be desirable to allow one or both of these parameters to vary, causing I_s and the number which multiplies V_c to be functions of x.

for all time $t > 0$. The shear force at $x = L$ is also zero, which implies

$$(E_c I_c + E_b I_b)\frac{\partial^3 w}{\partial x^3}(L,t) + E_b^* I_b \frac{\partial^4 w}{\partial t \partial x^3}(L,t) = 0 \tag{B.26}$$

for all time $t > 0$. At the clamped end, the position and slope of the beam are both zero, yielding

$$w(0,t) = 0 \tag{B.27}$$

$$\frac{\partial w}{\partial x}(0,t) = 0 \tag{B.28}$$

$$\tag{B.29}$$

for all time $t > 0$. Given initial conditions $w(x,0)$ and $\frac{\partial w}{\partial t}(x,0)$ for $x \in [0,L]$, Equation B.24 and Equations B.25 through B.28 give a complete description of the deformation of the beam for all time $t > 0$. [3]

B.3 Final Beam Model

In this section, we derive the equations of motion for a flexible cantilever beam excited by pairs of PZT crystals mounted at various places on the beam. We assume that all of the crystals have the same thickness, t_c. Additionally, the

[3] If the damping coefficient of the beam is small compared to the stiffnesses of the beam and the crystal (i.e. E_b^* is much smaller than E_b and E_c), then the boundary conditions given by Equations B.25 and B.26 can be approximated by

$$\frac{\partial^2 w}{\partial x^2}(L,t) = -\frac{b d_{31} E_c (t_b + t_c) V_c}{E_c I_c + E_b I_b} \tag{B.30}$$

and

$$\frac{\partial^3 w}{\partial x^3} = 0, \tag{B.31}$$

respectively, for all $t > 0$.

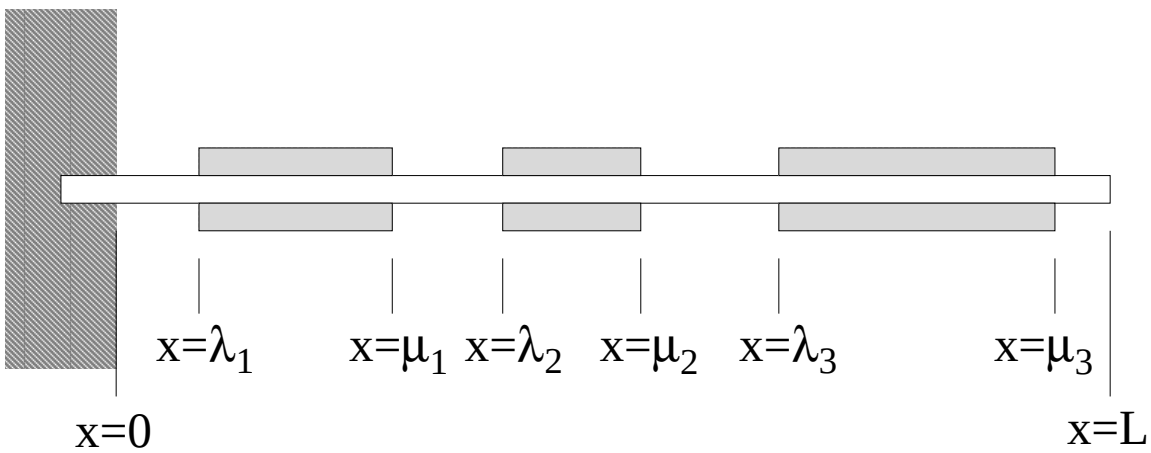


Figure B.7: Example configuration of final beam.

crystals all have the same width, b, which is also the width of the beam. An example of such a beam is depicted in Figure B.7.

First, we introduce some notation for describing the locations of the PZT crystals. Let k be the number of PZT pairs. We will use λ_i to denote the x coordinate of the left edge of the ith PZT pair. Let μ_i denote the x coordinate of the right edge of the ith PZT pair. The PZTs cannot overlap and they cannot extend beyond the dimensions of the beam. We assume that the first pair is the pair closest to the left end of the beam. We also define $\mu_0 = 0$ and $\lambda_{k+1} = L$. Then we have

$$0 = \mu_0 \leq \lambda_1 \leq \mu_1 \leq \lambda_2 \leq \mu_2 \leq \cdots \leq \lambda_k \leq \mu_k \leq \lambda_{k+1} = L. \tag{B.32}$$

We will call the sections of the beam with no PZT "simple" sections and we will call the sections with PZTs "sandwiched" sections.

Within the simple sections, the motion of beam is described by Equation B.12. Likewise, Equation B.24 describes the motion of the interiors of the sandwiched sections. In order to obtain a complete model for the beam, it is necessary to piece these sections together with the appropriate boundary conditions. The left end of the beam is clamped, so the boundary conditions at $x = 0$ are given by

Equations B.13 and B.14. At $x = L$, the beam is free so the boundary conditions are given by Equations B.15 and B.16 if the rightmost section of the beam is simple or Equations B.25 and B.26 if the rightmost section is sandwiched. We have yet to determine the boundary conditions at the points where simple section meets sandwiched section (i.e. $x \in \{\lambda_1, \mu_1, \ldots, \lambda_k, \mu_k\}$). Let V_c^i denote the voltage applied to the ith PZT pair.

First we need to introduce some more notation. Consider the function $f : \mathbb{R} \to \mathbb{R}$. We define the $f(x_0^-)$ to be the limit of $f(x)$ as x approaches x_0 from the left, i.e.

$$f(x_0^-) = \lim_{\substack{x \to x_0 \\ x < x_0}} f(x). \tag{B.33}$$

Similarly, we define $f(x_0^+)$ to be the limit of $f(x)$ as x approaches x_0 from the right.

We assume that w and $\frac{\partial w}{\partial x}$ are continuous functions in x, which means that for all $t > 0$

$$w(\lambda_i^-, t) = w(\lambda_i^+, t) \tag{B.34}$$

$$\frac{\partial w}{\partial x}(\lambda_i^-, t) = \frac{\partial w}{\partial x}(\lambda_i^+, t) \tag{B.35}$$

at the left sandwich–beam boundaries and

$$w(\mu_i^-, t) = w(\mu_i^+, t) \tag{B.36}$$

$$\frac{\partial w}{\partial x}(\mu_i^-, t) = \frac{\partial w}{\partial x}(\mu_i^+, t) \tag{B.37}$$

at the right boundaries. We assume the bending moment is continuous in the beam, so

$$M(\lambda_i^-) = M(\lambda_i^+). \tag{B.38}$$

Equations B.8 and B.22 are substituted into Equation B.38 to yield

$$E_b I_b \frac{\partial^2 w}{\partial x^2}(\lambda_i^-, t) = b d_{31} E_c (t_b + t_c) V_c^i + (E_b I_b + E_c I_c) \frac{\partial^2 w}{\partial x^2}(\lambda_i^+, t). \tag{B.39}$$

This gives us one set of boundary conditions for the left edges of the PZT. The same process computed at μ_i gives a set of boundary conditions for the right edges of the PZTs,

$$bd_{31}E_c(t_b + t_c)V_c^i + (E_bI_b + E_cI_c)\frac{\partial^2 w}{\partial x^2}(\mu_i^-, t) = E_bI_b\frac{\partial^2 w}{\partial x^2}(\mu_i^+, t). \tag{B.40}$$

Similarly, we assume the shear force is continuous in the beam. Setting $S(x^-) = S(x^+)$ at the boundaries yields additional boundary conditions

$$E_bI_b\frac{\partial^3 w}{\partial x^3}(\lambda_i^-, t) = (E_bI_b + E_cI_c)\frac{\partial^3 w}{\partial x^3}(\lambda_i^+, t) \tag{B.41}$$

and

$$(E_bI_b + E_cI_c)\frac{\partial^3 w}{\partial x^3}(\mu_i^-, t) = E_bI_b\frac{\partial^3 w}{\partial x^3}(\mu_i^+, t). \tag{B.42}$$

B.3.1 Final Model Statement

We can use the above information to piece together a model for the entire beam. It is stated here:

simple sections:

For $x \in (\mu_{i-1}, \lambda_i)$, $i = \{1, 2, 3, \ldots, k+1\}$,

$$\rho bt_b\frac{\partial^2 w}{\partial t^2} = -E_bI_b\frac{\partial^4 w}{\partial^4 x} - E_b^*I_b\frac{\partial^5 w}{\partial t\partial x^4}. \tag{B.43}$$

sandwiched sections:

For $x \in (\lambda_i, \mu_i)$, $i = \{1, 2, 3, \ldots, k\}$,

$$(\rho_c t_c + \rho_b t_b)\, b\frac{\partial^2 w}{\partial t^2} = -(E_cI_c + E_bI_b)\frac{\partial^4 w}{\partial x^4} - E_b^*I_b\frac{\partial^5 w}{\partial t\partial x^4}. \tag{B.44}$$

boundary conditions (clamped end):

$$w(0, t) = 0 \tag{B.45}$$

$$\frac{\partial w}{\partial x}(0, t) = 0 \tag{B.46}$$

for all time $t > 0$.

boundary conditions (free end):

If the rightmost section of the beam is simple, then for all $t > 0$

$$E_b I_b \frac{\partial^2 w}{\partial x^2}(L,t) + E_b^* I_b \frac{\partial^3 w}{\partial t \partial x^2}(L,t) = 0 \tag{B.47}$$

$$E_b I_b \frac{\partial^3 w}{\partial^3 x}(L,t) + E_b^* I_b \frac{\partial^4 w}{\partial t \partial x^3}(L,t) = 0. \tag{B.48}$$

If the rightmost section of the beam is sandwiched, then for all $t > 0$

$$bd_{31} E_c (t_b + t_c) V_c^k(t) + (E_c I_c + E_b I_b) \tfrac{\partial^2 w}{\partial x^2}(L,t) + E_b^* I_b \tfrac{\partial^3 w}{\partial t \partial x^2}(L,t) = 0 \tag{B.49}$$

$$(E_c I_c + E_b I_b) \frac{\partial^3 w}{\partial^3 x}(L,t) + E_b^* I_b \frac{\partial^4 w}{\partial t \partial x^3}(L,t) = 0. \tag{B.50}$$

boundary conditions (left edges of crystals):

$$w(\lambda_i^-, t) = w(\lambda_i^+, t) \tag{B.51}$$

$$\frac{\partial w}{\partial x}(\lambda_i^-, t) = \frac{\partial w}{\partial x}(\lambda_i^+, t) \tag{B.52}$$

$$E_b I_b \frac{\partial^2 w}{\partial x^2}(\lambda_i^-, t) = bd_{31} E_c (t_b + t_c) V_c^i + (E_b I_b + E_c I_c) \frac{\partial^2 w}{\partial x^2}(\lambda_i^+, t) \tag{B.53}$$

$$E_b I_b \frac{\partial^3 w}{\partial x^3}(\lambda_i^-, t) = (E_b I_b + E_c I_c) \frac{\partial^3 w}{\partial x^3}(\lambda_i^+, t) \tag{B.54}$$

for $i = \{1, 2, \ldots, k\}$, $\forall t > 0$.

boundary conditions (right edges of crystals):

$$w(\mu_i^-, t) = w(\mu_i^+, t) \tag{B.55}$$

$$\frac{\partial w}{\partial x}(\mu_i^-, t) = \frac{\partial w}{\partial x}(\mu_i^+, t) \tag{B.56}$$

$$bd_{31} E_c (t_b + t_c) V_c^i + (E_b I_b + E_c I_c) \frac{\partial^2 w}{\partial x^2}(\mu_i^-, t) = E_b I_b \frac{\partial^2 w}{\partial x^2}(\mu_i^+, t) \tag{B.57}$$

$$(E_b I_b + E_c I_c) \frac{\partial^3 w}{\partial x^3}(\mu_i^-, t) = E_b I_b \frac{\partial^3 w}{\partial x^3}(\mu_i^+, t) \tag{B.58}$$

for $i = \{1, 2, 3, \ldots, k\}$, $\forall t > 0$ if the rightmost section of the beam is simple, for $i = \{1, 2, 3, \ldots, k-1\}$, $\forall t > 0$ if the rightmost section of the beam is sandwiched.

Given initial conditions $w(x, 0)$ and $\frac{\partial w}{\partial t}(x, 0)$ for $x \in [0, L]$, and inputs $V_c^i(t)$, $i = \{1, 2, \ldots, k\}$, $\forall t > 0$, these equations give a complete description of the deformation of the beam for all time $t > 0$.

B.4 Finite Difference Model

Here we apply the finite difference method to the partial differential equations (PDEs) and boundary conditions listed in Section B.3.1 to obtain a set of ordinary differential equations (ODEs) which approximate the PDE system.

The first step in applying the finite difference method is to discretize the spatial domain. For the beam, we consider N evenly spaced points between $x = 0$ and $x = L$. We will include the point at $x = L$. We do not include the point as $x = 0$ (it is not necessary to include this point since the boundary conditions tell us is it always zero.) We label the leftmost point x_1 and the rightmost point x_N. This allows us to write the ith point as $x_i = i\Delta$, where $\Delta = L/N$ is the spacing between the points. We denote the displacement of the beam at the point x_i and time t as $w_i(t)$,

$$w_i(t) \triangleq w(x_i, t). \tag{B.59}$$

The next step is to approximate the spatial derivatives of w using the w_is. First some notation:

$$\frac{\partial^k w_i}{\partial x^k} \triangleq \left.\frac{\partial^k w}{\partial x^k}\right|_{x=x_i}. \tag{B.60}$$

The approximations for the first four spatial derivatives are then

$$\frac{\partial w_i}{\partial x} \approx \frac{w_i - w_{i-1}}{\Delta} \qquad \text{or} \qquad \frac{\partial w_i}{\partial x} \approx \frac{w_{i+1} - w_i}{\Delta}, \tag{B.61}$$

$$\frac{\partial^2 w_i}{\partial x^2} \approx \frac{w_{i-1} - 2w_i + w_{i+1}}{\Delta^2}, \tag{B.62}$$

$$\frac{\partial^3 w_i}{\partial x^3} \approx \frac{-w_{i-2} + 3w_{i-1} - 3w_i + w_{i+1}}{\Delta^3} \text{ or } \frac{\partial^3 w_i}{\partial x^3} \approx \frac{-w_{i-1} + 3w_i - 3w_{i+1} + w_{i+2}}{\Delta^3}, \tag{B.63}$$

and

$$\frac{\partial^4 w_i}{\partial x^4} \approx \frac{w_{i-2} - 4w_{i-1} + 6w_i - 4w_{i+1} + w_{i+2}}{\Delta^4}. \tag{B.64}$$

Of course, w_i is only defined for $i = \{1, 2, \ldots, N\}$, so these approximations are only valid at the "interior points", which are loosely defined here as points far enough away from the boundaries that the above spatial derivative approximations are valid.

We will obtain a system of ODEs approximating the PDE system by substituting Equations B.61, B.62, B.63, and B.64 into Equations B.43 and B.44 for each w_i. First, we introduce some more notation:

$$\dot{w}_i \triangleq \frac{d}{dt} w_i \tag{B.65}$$

$$\ddot{w}_i \triangleq \frac{d^2}{dt^2} w_i \tag{B.66}$$

$$\Gamma_b \triangleq \frac{E_b I_b}{\rho_b b t_b} \tag{B.67}$$

$$\Gamma_b^* \triangleq \frac{E_b^* I_b}{\rho b t_b} \tag{B.68}$$

$$\Gamma_c \triangleq \frac{E_b I_b + E_c I_c}{b(\rho_b t_b + 2\rho_c t_c)} \tag{B.69}$$

Now we can use this notation to rewrite the equations of motion for the interior of a simple beam section:

$$\frac{\partial^2 w}{\partial t^2} = -\Gamma_b \frac{\partial^4 w}{\partial x^4} - \Gamma_b^* \frac{\partial^5 w}{\partial t \partial x^4}. \tag{B.70}$$

Subsituting from Equation B.64 yields

$$\ddot{w}_i = -\left(\frac{\Gamma_b + \Gamma_b^* \frac{d}{dt}}{\Delta^4} \right) \left(w_{i-2} - 4w_{i-1} + 6w_i - 4w_{i+1} + w_{i+2} \right). \tag{B.71}$$

Again, this expresseion is valid only at the "interior points" in the simple section, i.e. points x_i such that $x_{i-2}, x_{i-1}, x_i, x_{i+1}$, and x_{i+2} are all contained in the simple section. Hence, Equation B.71 provides a second order ODE for each interior point in the simple section. Likewise, within a sandwiched section we get

$$\ddot{w}_i = -\left(\frac{\Gamma_c + \Gamma_c^* \frac{d}{dt}}{\Delta^4}\right)(w_{i-2} - 4w_{i-1} + 6w_i - 4w_{i+1} + w_{i+2}). \tag{B.72}$$

Again, this expression is only valid for interior points of the sandwiched section.

So we have now derived second order ODEs to describe the motion of each interior point on the beam. Now we need to find ODEs to describe the remaining "boundary points". This can be done using the boundary conditions from the PDE model.

The problem that arises at the boundary points is that the spatial derivative approximations require the values of displacements at *points that are not defined for the original PDE*. To solve this problem, we define "ghost points". The displacement at these ghost points can be found using the boundary conditions.

For example, at the clamped end of the beam, we define ghost points $\widetilde{x}_0$ and $\widetilde{x}_{-1}$ which lie Δ and 2Δ to the left of x_1, respectively. We denote the displacement at $\widetilde{x}_i$ by $\widetilde{w}_i$. The boundary condition

$$w(0,t) = 0 \quad \forall t > 0 \tag{B.73}$$

implies that $\widetilde{w}_0 = 0 \quad \forall t > 0$. The boundary condition

$$\frac{\partial w}{\partial x}(0,t) \quad \forall t > 0 \tag{B.74}$$

implies

$$\begin{aligned} \frac{\widetilde{w}_0 - \widetilde{w}_{-1}}{\Delta} &= 0 \\ \Rightarrow \quad \widetilde{w}_{-1} &= 0 \quad \forall t > 0. \end{aligned} \tag{B.75}$$

Using these ghost points, we can now write a valid expressions for $\ddot{w}_1$ and $\ddot{w}_2$ (assuming the leftmost section of the beam is simple):

$$\begin{aligned}\ddot{w}_1 &= -\left(\frac{\Gamma_b + \Gamma_b^* \frac{d}{dt}}{\Delta^4}\right)(\widetilde{w}_{-1} - 4\widetilde{w}_0 + 6w_1 - 4w_2 + w_3) \\ &= -\left(\frac{\Gamma_b + \Gamma_b^* \frac{d}{dt}}{\Delta^4}\right)(6w_1 - 4w_2 + w_3) \end{aligned} \tag{B.76}$$

and

$$\begin{aligned}\ddot{w}_2 &= -\left(\frac{\Gamma_b + \Gamma_b^* \frac{d}{dt}}{\Delta^4}\right)(\widetilde{w}_0 - 4w_1 + 6w_2 - 4w_3 + w_4) \\ &= -\left(\frac{\Gamma_b + \Gamma_b^* \frac{d}{dt}}{\Delta^4}\right)(-4w_1 + 6w_2 - 4w_3 + w_4)\,. \end{aligned} \tag{B.77}$$

At the free end of the beam, we introduce ghost points $\widetilde{x}_{N+1}$ and $\widetilde{x}_{N+2}$. Assuming the rightmost section of the beam is simple, the zero moment condition is

$$E_b I_b \frac{\partial^2 w}{\partial x^2}(L,t) + E_b^* I_b \frac{\partial^3 w}{\partial t \partial x^2}(L,t) = 0 \quad \forall t > 0. \tag{B.78}$$

Subsituting the appropriate spatial derivative approximations yields

$$E_b I_b \left(\frac{w_{N-1} - 2w_N + \widetilde{w}_{N+1}}{\Delta^2}\right) + E_b^* I_b \left(\frac{\dot{w}_{N-1} - 2\dot{w}_N + \frac{d}{dt}\widetilde{w}_{N+1}}{\Delta^2}\right) = 0, \tag{B.79}$$

which simplifies to the ODE

$$\frac{d}{dt}\widetilde{w}_{N+1} = -\frac{E_b I_b}{E_b^* I_b}\left(w_{N-1} - 2w_N + \widetilde{w}_{N+1}\right) - \dot{w}_{N-1} + 2\dot{w}_N, \tag{B.80}$$

which must hold for all $t > 0$. The shear condition at the free end is

$$E_b I_b \frac{\partial^3 w}{\partial x^3}(L,t) + E_b^* I_b \frac{\partial^3 w}{\partial t \partial x^2}(L,t) = 0 \quad \forall t > 0. \tag{B.81}$$

Making the spatial derivative approximations yields

$$\begin{aligned} &E_b I_b \left(\frac{-w_{N-1} + 3w_N - 3\widetilde{w}_{N+1} + \widetilde{w}_{N+2}}{\Delta^3}\right) \\ &\quad + E_b^* I_b \left(\frac{-\dot{w}_{N-1} + 3\dot{w}_N - 3\frac{d}{dt}\widetilde{w}_{N+1} + \frac{d}{dt}\widetilde{w}_{N+2}}{\Delta^3}\right) = 0. \end{aligned} \tag{B.82}$$

Simplifying and substituting for $\frac{d}{dt}\widetilde{w}_{N+1}$ from Equation B.80 yields the ODE

$$\frac{d}{dt}\widetilde{w}_{N+2} = -\frac{EbIb}{E_b^* I_b}\left(2w_{N-1} - 3w_N + \widetilde{w}_{N+2}\right) - 2\dot{w}_{N-1} + 3\dot{w}_N, \tag{B.83}$$

which hold for all $t > 0$.

Using these ghost points, we can now write a valid expressions for $\ddot{w}_{N-1}$ and $\ddot{w}_N$ (assuming the rightmost section of the beam is simple):

$$\ddot{w}_{N-1} = -\left(\frac{\Gamma_b + \Gamma_b^* \frac{d}{dt}}{\Delta^4}\right)\left(w_{N-3} - 4w_{N-2} + 6w_{N-1} - 4w_N + \widetilde{w}_{N+1}\right) \tag{B.84}$$

and

$$\ddot{w}_N = -\left(\frac{\Gamma_b + \Gamma_b^* \frac{d}{dt}}{\Delta^4}\right)\left(w_{N-2} - 4w_{N-1} + 6w_N - 4\widetilde{w}_{N+1} + \widetilde{w}_{N+2}\right), \tag{B.85}$$

where the quantities $\widetilde{w}_{N+1}$ and $\widetilde{w}_{N+2}$ are given by the solutions of the ODEs in Equations B.80 and B.83 respectively. [4]

We now look at the left boundary of a PZT pair. Recall that the beam model is actually composed of different PDEs "pasted" together using boundary conditions. Consider a left boundary at $x = \lambda$. Let ℓ be a integer such that $x_{\ell-1} < \lambda \leq x_\ell$.

[4] If the stiffness of the beam is much greater than its damping, (i.e. $E_b \gg E_b^*$), then the ODEs given by Equations B.80 and B.83 can be approximated by the following algebraic expressions for all ($t > 0$):

$$\widetilde{w}_{N+1} = -w_{N-1} + 2w_N \tag{B.86}$$

$$\widetilde{w}_{N+2} = -2w_{N-1} + 3w_N. \tag{B.87}$$

As a result, the ODEs for the two rightmost points on the beam become

$$\ddot{w}_{N-1} = -\left(\frac{\Gamma_b + \Gamma_b^* \frac{d}{dt}}{\Delta^4}\right)\left(w_{N-3} - 4w_{N-2} + 5w_{N-1} - 2w_N\right) \tag{B.88}$$

$$\ddot{w}_N = -\left(\frac{\Gamma_b + \Gamma_b^* \frac{d}{dt}}{\Delta^4}\right)\left(w_{N-2} - 2w_{N-1} + w_N\right). \tag{B.89}$$

We assume that there is no other boundary "nearby" so that, for points of interest to the boundary, x_i is in a simple section for $i < \ell$ and and x_i is in a sandwiched section for $i \geq \ell$. The the displacement expression given in Equation B.71 is not valid at The simple section points $x_{\ell-2}$ and $x_{\ell-1}$ are boundary points since the displacement expression given in Equation B.71 requires the displacement values at points outside the section. Likewise, the displacement expression given in Equation B.72 is not valid for the sandwiched section points x_ℓ and $x_{\ell+1}$. Hence, we introduce ghost points $\widetilde{x}_\ell$, and $\widetilde{x}_{\ell+1}$ (to augment the simple section) and $\widetilde{x}_{\ell-2}$ and $\widetilde{x}_{\ell-1}$ (to augment the sandwiched section).

Now we use the boundary conditions given in Equations B.51 through B.58 to derive expressions for the ghost displacements $\widetilde{w}_{\ell-2}$, $\widetilde{w}_{\ell-1}$, $\widetilde{w}_\ell$, and $\widetilde{w}_{\ell+1}$ in terms of the actual displacements. First, we introduce some notation to simplify the resulting expressions. Let

$$\Psi = \frac{E_b I_b + E_c I_c}{E_b I_b} \tag{B.90}$$

and

$$\Lambda = -b d_{31} E_c (t_b + t_c). \tag{B.91}$$

The boundary condition

$$w(\lambda^-, t) = w(\lambda^+, t) \quad \forall t > 0 \tag{B.92}$$

implies

$$\widetilde{w}_\ell = w_\ell \quad \forall t > 0. \tag{B.93}$$

The boundary condition

$$\frac{\partial w}{\partial x}(\lambda^-, t) = \frac{\partial w}{\partial x}(\lambda^+, t) \quad \forall t > 0 \tag{B.94}$$

implies

$$\frac{\widetilde{w}_{\ell+1} - \widetilde{w}_\ell}{\Delta} = \frac{w_{\ell+1} - w_\ell}{\Delta}$$

$$\Rightarrow \quad \widetilde{w}_{\ell+1} = w_{\ell+1} \quad \forall t > 0. \tag{B.95}$$

The boundary condition given by Equation B.53 becomes

$$\frac{\partial^2 w}{\partial x^2}(\lambda^-, t) = -\Lambda V_c + \Psi \frac{\partial^2 w}{\partial x^2}(\lambda^+, t) \quad \forall t > 0, \tag{B.96}$$

which implies

$$w_{\ell-1} - 2\widetilde{w}_\ell + \widetilde{w}_{\ell+1} = -\Delta^2 \Lambda V_c + \Psi \left(\widetilde{w}_{\ell-1} - 2w_\ell + w_{\ell+1}\right)$$

$$\Rightarrow \quad w_{\ell-1} - 2w_\ell + w_{\ell+1} + \Delta^2 \Lambda V_c + 2\Psi w_\ell - \Psi w_{\ell+1} = \Psi \widetilde{w}_{\ell-1}$$

$$\Rightarrow \quad \widetilde{w}_{\ell-1} = \frac{\Delta^2 \Lambda}{\Psi} V_c + \frac{1}{\Psi} w_{\ell-1} + \frac{2(\Psi - 1)}{\Psi} w_\ell - \frac{\Psi - 1}{\Psi} w_{\ell+1} \tag{B.97}$$

for all $t > 0$. The boundary condition given by Equation B.54 becomes

$$\frac{\partial^3 w}{\partial x^3}(\lambda^-, t) = \Psi \frac{\partial^3 w}{\partial x^3}(\lambda^+, t) \quad \forall t > 0, \tag{B.98}$$

which implies

$$-w_{\ell-2} + 3w_{\ell-1} - 3\widetilde{w}_\ell + \widetilde{w}_{\ell+1} = \Psi \left(-\widetilde{w}_{\ell-2} + 3\widetilde{w}_{\ell-1} - 3w_\ell + w_{\ell+1}\right)$$

$$\Rightarrow \quad -w_{\ell-2} + 3w_{\ell-1} - 3w_\ell + w_{\ell+1} - 3\Psi \widetilde{w}_{\ell-1} + 3\Psi w_\ell - \Psi w_{\ell+1} = -\Psi \widetilde{w}_{\ell-2}$$

$$\begin{aligned} \Rightarrow \quad \widetilde{w}_{\ell-2} &= \frac{1}{\Psi} w_{\ell-2} - \frac{3}{\Psi} w_{\ell-1} + \frac{3}{\Psi} w_\ell - \frac{1}{\Psi} w_{\ell+1} \\ &\quad + 3\left(\frac{\Delta^2 \Lambda}{\Psi} V_c + \frac{1}{\Psi} w_{\ell-1} + \frac{2(\Psi - 1)}{\Psi} w_\ell - \frac{\Psi - 1}{\Psi} w_{\ell+1}\right) - 3w_\ell + w_{\ell+1} \end{aligned}$$

$$\Rightarrow \quad \widetilde{w}_{\ell-2} = \frac{3\Delta^2 \Lambda}{\Psi} V_c + \frac{1}{\Psi} w_{\ell-2} + \frac{3(\Psi - 1)}{\Psi} w_\ell - \frac{2(\Psi - 1)}{\Psi} w_{\ell+1}. \tag{B.99}$$

Hence, to the left of the boundary we get

$$\begin{aligned} \ddot{w}_{\ell-2} &= -\frac{\Gamma_b + \Gamma_b^* \frac{d}{dt}}{\Delta^4} \left(w_{\ell-4} - 4w_{cl-3} + 6w_{\ell-2} - 4w_{\ell-1} + \widetilde{w}_\ell\right) \\ &= -\frac{\Gamma_b + \Gamma_b^* \frac{d}{dt}}{\Delta^4} \left(w_{\ell-4} - 4w_{\ell-3} + 6w_{\ell-2} - 4w_{\ell-1} + w_\ell\right) \end{aligned} \tag{B.100}$$

and

$$\begin{aligned}
\ddot{w}_{\ell-2} &= -\frac{\Gamma_b + \Gamma_b^* \frac{d}{dt}}{\Delta^4}\left(w_{\ell-3} - 4w_{\ell-2} + 6w_{\ell-1} - 4\widetilde{w}_\ell + \widetilde{w}_{\ell+1}\right) \\
&= -\frac{\Gamma_b + \Gamma_b^* \frac{d}{dt}}{\Delta^4}\left(w_{\ell-3} - 4w_{\ell-2} + 6w_{\ell-1} - 4w_\ell + w_{\ell+1}\right). \qquad \text{(B.101)}
\end{aligned}$$

To the right of the boundary

$$\begin{aligned}
\ddot{w}_\ell &= -\frac{\Gamma_c + \Gamma_c^* \frac{d}{dt}}{\Delta^4}\left(\widetilde{w}_{\ell-2} - 4\widetilde{w}_{\ell-1} + 6w_\ell - 4w_{\ell+1} + w_{\ell+2}\right) \\
&= -\frac{\Gamma_c + \Gamma_c^* \frac{d}{dt}}{\Delta^4}\left(\left(\frac{3\Delta^2\Lambda}{\Psi}V_c + \frac{1}{\Psi}w_{\ell-2} + \frac{3(\Psi-1)}{\Psi}w_\ell - \frac{2(\Psi-1)}{\Psi}w_{\ell+1}\right)\right. \\
&\quad -4\left(\frac{\Delta^2\Lambda}{\Psi}V_c + \frac{1}{\Psi}w_{\ell-1} + \frac{2(\Psi-1)}{\Psi}w_\ell - \frac{\Psi-1}{\Psi}w_{\ell+1}\right) \\
&\quad \left.+6w_\ell - 4w_{\ell+1} + w_{\ell+2}\right) \\
&= -\frac{\Gamma_c + \Gamma_c^* \frac{d}{dt}}{\Delta^4}\left(\frac{1}{\Psi}w_{\ell-2} - \frac{4}{\Psi}w_{\ell-1} + \left(6 + \frac{5(\Psi-1)}{\Psi}\right)w_\ell\right. \\
&\qquad \left. + \left(\frac{2(\Psi-1)}{\Psi} - 4\right)w_{\ell+1} + w_{\ell+2}\right) + \frac{\Gamma_c\Lambda}{\Delta^2\Psi}V_c \qquad \text{(B.102)}
\end{aligned}$$

and

$$\begin{aligned}
\ddot{w}_{\ell+1} &= -\frac{\Gamma_c + \Gamma_c^* \frac{d}{dt}}{\Delta^4}\left(\widetilde{w}_{\ell-1} - 4w_\ell + 6w_{\ell+1} - 4w_{\ell+2} + w_{\ell+3}\right) \\
&= -\frac{\Gamma_c + \Gamma_c^* \frac{d}{dt}}{\Delta^4}\left(\frac{\Delta^2\Lambda}{\Psi}V_c + \frac{1}{\Psi}w_{\ell-1} + \frac{2(\Psi-1)}{\Psi}w_\ell\right. \\
&\quad \left. -\frac{\Psi-1}{\Psi}w_{\ell+1} - 4w_\ell + 6w_{\ell+1} - 4w_{\ell+2} + w_{\ell+3}\right) \\
&= -\frac{\Gamma_c + \Gamma_c^* \frac{d}{dt}}{\Delta^4}\left(\frac{1}{\Psi}w_{\ell-1} + \left(\frac{2(\Psi-1)}{\Psi} - 4\right)w_\ell\right. \\
&\qquad \left. + \left(6 - \frac{\Psi-1}{\Psi}\right)w_{\ell+1} - 4w_{\ell+2} + w_{\ell+3}\right) - \frac{\Gamma_c\Lambda}{\Delta^2\Psi}V_c. \qquad \text{(B.103)}
\end{aligned}$$

Here we have assumed that $\Gamma_c^* \frac{d}{dt} V_c$ is small compared to $\Gamma_c V_c$ and can be neglected. In practice, $\Gamma_c \gg \Gamma_c^*$ and V_c is smooth, so this assumption is reasonable.

A similar procedure is followed to obtain ODEs for the boundary points at the right edge of a crystal pair. The resulting equations are stated in the following section.

B.4.1 Finite Difference Model Statement

Here we summarize the previous section by stating the complete set of ODEs which comprise the finite difference approximation of the beam model.

Define $\ell(i)$ to be an integer such that $x_{\ell(i)-1} < \ell(i) \leq x_{\ell(i)}$. Likewise, define $r(i)$ to be an integer such that $x_{r(i)} \leq r(i) < x_{r(i)+1}$. additionally, let $r(0) = 2$ and $\ell(k+1) = N$.

simple sections:

For i such that $r(j) < i < \ell(j+1)$, $j \in \{0, 1, 2, \ldots, k\}$,

$$\ddot{w}_i = -\frac{\Gamma_b + \Gamma_b^* \frac{d}{dt}}{\Delta^4} \left(w_{i-2} - 4w_{i-1} + 6w_i - 4w_{i+1} + w_{i+2}\right). \tag{B.104}$$

sandwiched sections:

For i such that $\ell(j) + 2 \leq i \leq r(j) - 2$, $j \in \{1, 2, 3, \ldots, k\}$,

$$\ddot{w}_i = -\frac{\Gamma_c + \Gamma_c^* \frac{d}{dt}}{\Delta^4} \left(w_{i-2} - 4w_{i-1} + 6w_i - 4w_{i+1} + w_{i+2}\right). \tag{B.105}$$

boundary conditions (clamped end):

$$\ddot{w}_1 = -\frac{\Gamma_b + \Gamma_b^* \frac{d}{dt}}{\Delta^4} \left(6w_1 - 4w_2 + w_3\right), \tag{B.106}$$

$$\ddot{w}_2 = -\frac{\Gamma_b + \Gamma_b^* \frac{d}{dt}}{\Delta^4} \left(-4w_1 + 6w_2 - 4w_3 + w_4\right). \tag{B.107}$$

boundary conditions (free end):

$$\ddot{w}_{N-1} = -\frac{\Gamma_b + \Gamma_b^* \frac{d}{dt}}{\Delta^4} \left(w_{N-3} - 4w_{N-2} + 6w_{N-1} - 4w_N + \widetilde{w}_{N+1}\right) \tag{B.108}$$

$$\ddot{w}_N = -\frac{\Gamma_b + \Gamma_b^* \frac{d}{dt}}{\Delta^4} \left(w_{N-2} - 4w_{N-1} + 6w_N - 4\widetilde{w}_{N+1} + \widetilde{w}_{N+2}\right), \tag{B.109}$$

where $\widetilde{w}_{N+1}$ and $\widetilde{w}_{N+2}$ satisfy

$$\frac{d}{dt}\widetilde{w}_{N+1} = -\frac{E_b I_b}{E_b^* I_b} \left(w_{N-1} - 2w_N + \widetilde{w}_{N+1}\right) - \dot{w}_{N-1} + 2\dot{w}_N \tag{B.110}$$

$$\frac{d}{dt}\widetilde{w}_{N+2} = -\frac{EbIb}{E_b^* I_b} \left(2w_{N-1} - 3w_N + \widetilde{w}_{N+2}\right) - 2\dot{w}_{N-1} + 3\dot{w}_N. \tag{B.111}$$

boundary conditions (left edge of PZT pair):

For $i \in \{1, 2, 3, \ldots, k\}$,

$$\begin{aligned}
\ddot{w}_{\ell(i)} &= -\frac{\Gamma_c + \Gamma_c^* \frac{d}{dt}}{\Delta^4} \left(\frac{1}{\Psi} w_{\ell(i)-2} - \frac{4}{\Psi} w_{\ell(i)-1} + \left(6 + \frac{5(\Psi - 1)}{\Psi} \right) w_{\ell(i)} \right. \\
&\quad \left. + \left(\frac{2(\Psi - 1)}{\Psi} - 4 \right) w_{\ell(i)+1} + w_{\ell(i)+2} \right) + \frac{\Gamma_c \Lambda}{\Delta^2 \Psi} V_c^i \qquad \text{(B.112)} \\
\ddot{w}_{\ell(i)+1} &= -\frac{\Gamma_c + \Gamma_c^* \frac{d}{dt}}{\Delta^4} \left(\frac{1}{\Psi} w_{\ell(i)-1} + \left(\frac{2(\Psi - 1)}{\Psi} - 4 \right) w_{\ell(i)} \right. \\
&\quad \left. + \left(6 - \frac{\Psi - 1}{\Psi} \right) w_{\ell(i)+1} - 4 w_{\ell(i)+2} + w_{\ell(i)+3} \right) - \frac{\Gamma_c \Lambda}{\Delta^2 \Psi} V_c^i. \qquad \text{(B.113)}
\end{aligned}$$

boundary conditions (right edge of PZT pair):

For $i \in \{1, 2, 3, \ldots, k\}$,

$$\begin{aligned}
\ddot{w}_{r(i)-1} &= -\frac{\Gamma_c + \Gamma_c^* \frac{d}{dt}}{\Delta^4} \left(w_{r(i)-3} - 4 w_{r(i)-2} + \left(6 - \frac{\Psi - 1}{\Psi} \right) w_{r(i)-1} \right. \\
&\quad \left. + \left(\frac{2(\Psi - 1)}{\Psi} - 4 \right) w_{r(i)} + \frac{1}{\Psi} w_{r(i)+1} \right) - \frac{\Gamma_c \Lambda}{\Delta^2 \Psi} V_c \qquad \text{(B.114)} \\
\ddot{w}_{r(i)} &= \frac{\Gamma_c + \Gamma_c^* \frac{d}{dt}}{\Delta^4} \left(w_{r(i)-2} + \left(\frac{2(\Psi - 1)}{\Psi} - 4 \right) w_{r(i)-1} \right. \\
&\quad \left. + \left(6 + \frac{5(\Psi - 1)}{\Psi} \right) w_{r(i)} - \frac{4}{\Psi} w_{r(i)+1} + \frac{1}{\Psi} w_{r(i)+2} \right) + \frac{\Gamma_c \Lambda}{\Delta^2 \Psi} V_c^i. \qquad \text{(B.115)}
\end{aligned}$$

Given initial conditions $w_i(0)$ and $\dot{w}_i(0)$ for $i = \{1, 2, \ldots, N\}$, $\widetilde{w}_{N+1}(0)$, and $\widetilde{w}_{N+2}(0)$, and inputs $V_c^j(t)$, $j = \{1, 2, \ldots, k\}$, $\forall t > 0$, these equations give a complete description of the deformation of the beam for all time $t > 0$. Hence, we have approximated the PDEs listed in Section B.3.1 with a set of N second order ODEs along with two first order ODEs.

B.5 Simulation Results

In this section we present results of the simulation of a flexible beam based on the above model. The N second order ODEs listed in Section B.4.1 were first

transformed into state-space form to creating $2N$ first order ODEs. These were then numerically integrated using the fourth order Runge–Kutta method.

For the simulation, we chose to model an aluminum beam with a length of 1 meter, a width of 0.05 meters, and a thickness of 2.5 millimeters. Four pairs of PZT crystals are evenly spaced along the length of the beam.

Figures B.8 and B.9 depict the response of the beam to a 100 volt step applied to the leftmost actuator pair at time $t = 0$. Figure B.8 shows the configuration of the entire beam at ten successive "snapshots" in time. Figure B.9 plots the position of the tip of the beam as a function of time. Both figures compare well with experimentally observed results for a similar beam [26].

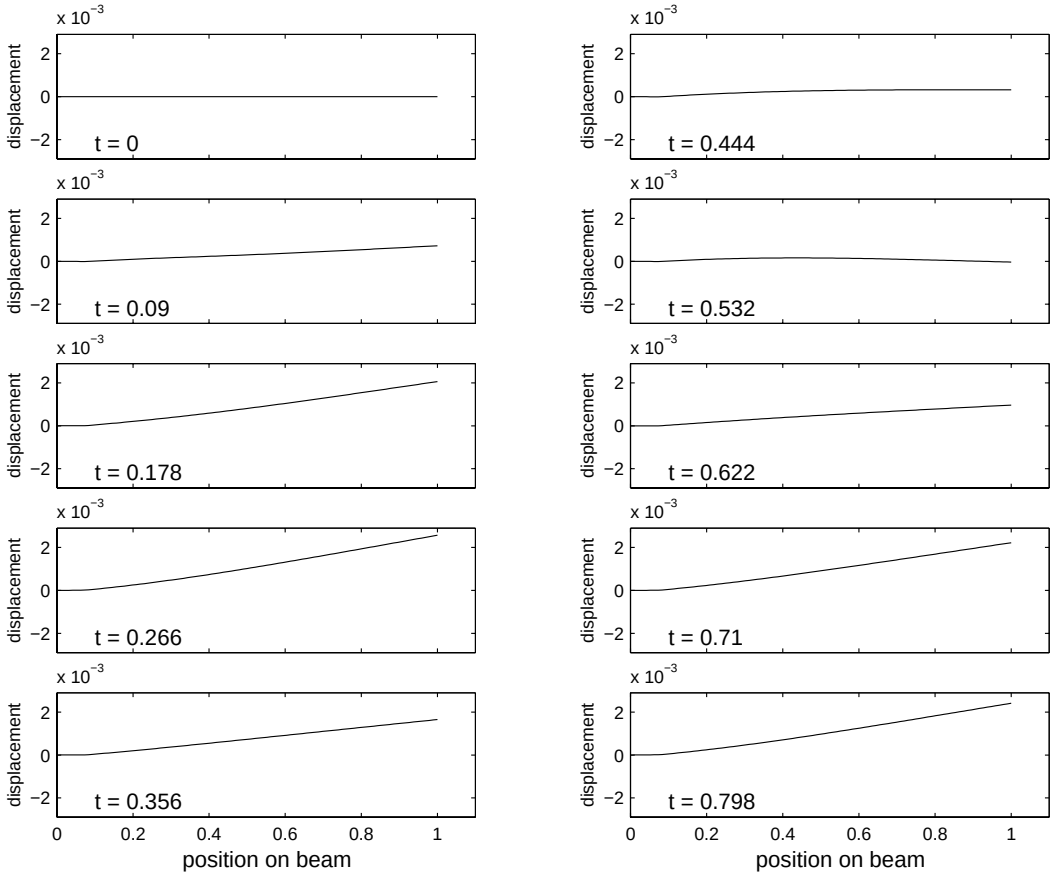


Figure B.8: Snapshots in time of simulated cantilever beam subject to a 100 volt step input to the leftmost actuator.

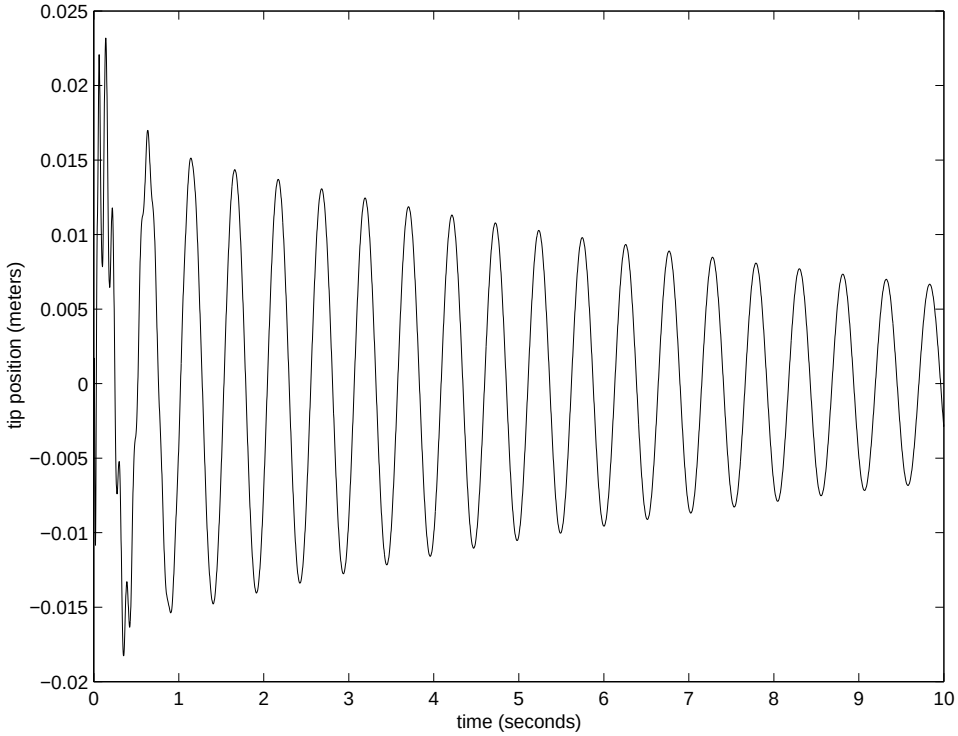


Figure B.9: Trajectory of tip of the simulated cantilever beam subject to a 100 volt step input to the leftmost actuator .

BIBLIOGRAPHY

[1] B. Bamieh. The structure of optimal controllers of spatially invariant distributed parameter systems. In *Proceedings of the 36th IEEE Conference on Decision and Control*, December 1997.

[2] B. Bamieh, F. Paganini, and M. Dahleh. Optimal control of distributed actuator and sensor arrays. In *Proceedings of the SPIE 5th Annual International Symposium on Smart Structures and Materials*, March 1998.

[3] T.G. Bifano, J. Perreault, R. Krishnamoorthy, and M.N. Horenstein. Micreelectromechanical deformable mirrors. *http://eng.bu.edu/AME/lab/perl*, 1999.

[4] J. Bontsema. *Dymanic Stabilization of Large Flexible Space Structures.* 1989.

[5] M.S. Branicky. Topology of hybrid systems. In *Proceedings of the 32nd IEEE Conference on Decision and Control*, pages 2309–2314, December 1993.

[6] M.S. Branicky. Universal computation and other capabilities of hybrid and continuous dynamical systems. *Theoretical Computer Science, Special Issue on Hybrid Systems*, 138(1):67–81, February 1995.

[7] R. Brockett. Hybrid models for motion control systems. In H. Trentelman and J.C. Willems, editors, *Perspectives in Control*, pages 29–54. Birkhauser, Boston, 1993.

[8] R. Brockett. Language driven hybrid systems. In *Proceedings of the 33rd IEEE Conference on Decision and Control*, pages 4210–4214, 1994.

[9] R. Brockett. Stabilization of motor networks. In *Proceedings of the 34rd Conference on Decision and Control*, pages 1484–1488, December 1995.

[10] R.W. Brockett. Dynamical systems that sort lists, diagonalize matrices, and solve linear programming problems. In *Proceedings of the 1988 IEEE Conference on Decision and Control*, pages 799–803, 1988.

[11] R.W. Brockett and J.L. Willems. Operational mehtods for treating least squares and stability problems in distributed parameter systems. In *Proceedings of the 1971 Conference Internacional Sobre Sistemas, Redes, y Computadors*, volume 2, pages 721–725, 1971.

[12] R.W. Brockett and J.L. Willems. Discretized partial differential equations: Examples of control systems defined on modules. *Automatica*, 10:507–515, 1974.

[13] I. Chopra. Smart structures. Class notes for ENAE 651 at University of Maryland, College Park, 1996.

[14] K. Chou, G. Guthart, and D. Flamm. A multiscale approach to the control of smart materials. In *Proceedings of the SPIE Conference on Smart Structures and Materials*, volume 2447, pages 249–263, 1995.

[15] I. Daubechies. *Ten Lectures on Wavelets*. SIAM, 1992.

[16] M.L. El-Sayed and P.S. Krishnaprasad. Homogeneous interconnected systems: An example. *IEEE Transactions on Automatic Control*, 26(4):894–901, August 1981.

[17] R. Gentilman et al. Piezoelectric composites fo active surface control. In *Smart Materials and Structures: DARPA Technology Interchange Meeting #4*, pages 79–110, August 1997.

[18] J.L. Fanson and T.K. Caughey. Positive position feedback control for large space structures. In *Proceedings of the 28th AIAA/ASME/ASC/AHS Structures, Structural Dynamics, and Materials Conference*, volume 2, pages 588–598, 1987.

[19] D.N. Godbole, J. Lygeros, and S. Sastry. Hierarchical hybrid control: a case study. In *Proceedings of the 33rd Conference on Decision and Control*, pages 1592–1597, December 1994.

[20] S. Helgason. *Differential Geometry, Lie Groups, and Symmetric Spaces*. Academic Press, 1978.

[21] U. Helmke and J.B. Moore. Singular–value decomposition via gradient and self–equivalent flows. *Linear Algebra and Its Applications*, 169:223–248, 1992.

[22] U. Helmke and J.B. Moore. *Optimization and Dynamical Systems*. Springer-Verlag, 1994.

[23] R. Hermann. *Lie Groups for Physicists*. W.A. Benjamin, Inc., 1966.

[24] D. Hristu. *Optimal Control with Limited Communication*. PhD thesis, Harvard University, 1999.

[25] D. Johnson. Intelligent instrumentation expands functionality through communication options. *Control Engineering*, pages 69–76, January 1997.

[26] G.A Kantor. Linear control theory as applied to smart structures. Master's thesis, University of Maryland, College Park, 1996. advisor: W.P. Dayawansa.

[27] S.M. Melzer and B.C. Kuo. Optimal regulation of systems described by a countably infinite number of objects. *Automatica*, 7:359–366, 1971.

[28] A.R. Mitchell and D.F. Griffiths. *The Finite Difference Method in Partial Differential Equations*. Wiley, 1980.

[29] J.B. Moore, R.E. Mahoney, and U. Helmke. Numerical gradient algorithms for eigenvalue and singular value decompostion. *SIAM Journal on Matrix Analysis and Applications*, 15(3):881–902, 1994.

[30] R.M. Murray, Z. Li, and S. Sastry. *A Mathematical Introduction to Robotic Manipulation*. CRC Press, 1994.

[31] H. Nijmeijer and A. van der Schaft. *Nonlinear Dynamical Control Systems*. Springer-Verlag, 1990.

[32] J.M. Ooi, S.M. Verbout, J.T. Ludwig, and G.W. Wornell. A separation theorem for periodic sharing information paterns in decentralized control. *IEEE Transactions on Automatic Control*, 42(11):1546–1550, November 1997.

[33] J.F. Price. *Lie Groups and Compact Groups*. Cambridge University Press, 1977.

[34] R.S. Raji. Smart networks for control. *IEEE Spectrum*, 31(6):49–55, June 1994.

[35] G. Strang. *Linear Algebra and Its Applications*. Saunders HBJ, 1988.

[36] G. Strang and T. Nguyen. *Wavelets and Filter Banks*. Wellesley-Cambridge Press, 1996.

[37] H.K. Struemper. *Motion Control for Nonholonomic Systems on Matrix Lie Groups*. PhD thesis, University of Maryland, 1997.

[38] M. Talbot and R. Nayer. CAN and USB - the serial future. *Electronic Engineering*, pages 63–64, September 1997.

[39] P. Tsiotras, J.L. Junkins, and H. Schaub. Higher order cayley transforms with applications to attitude representations. *Journal of Guidance, Control, and Dynamics*, 20(3):528–536, 1997.

[40] V.S. Varadarajan. *Lie Groups, Lie Algebras, and Their Representations.* Prentice-Hall, Inc., 1984.

[41] W.J. Wang and L.G. Mau. Stabilization and estimation for perturbed discrete time-delay large-scale systems. *IEEE Transactions on Automatic Control*, 42(9):1277–1282, September 1997.

[42] F.W. Warner. *Foundations of Differential Manifolds and Lie Groups.* Scott, Foresman and Company, 1971.

[43] M.V. Wickerhauser. Large-rank approximate principal component analysis with wavelets for signal feature discrimination and the inversion of complicated maps. *J. Chem. Inf. Comput. Sci.*, 34:1036–1046, 1993.

[44] M.V. Wickerhauser. Two fast approximate wavelet algorithms for image processing, classification, and recognition. *Optical Engineering*, 33(7):2225–2235, July 1994.

[45] S.R. Winzer. Composite smart material for defense and dual use applications. In *Smart Materials and Structures: DARPA Technology Interchange Meeting #4*, pages 113–133, August 1997.

[46] W.S. Wong and R.W. Brockett. State estimation with finite communication bandwidth constraints II: Feedback control problems. preprint, 1995.

[47] W.S. Wong and R.W. Brockett. State estimation with finite communication bandwidth constraints I: State estimation problems. *IEEE Transactions on Automatic Control*, 42(9):1294–1299, September 1997.