Anomaly Detection Using Polynomial Optimization Methods

A Dissertation Presented

by

Jose A. Lopez

to

The Department of Electrical and Computer Engineering

in partial fulfillment of the requirements

for the degree of

Doctor of Philosophy

in

Electrical and Computer Engineering

Northeastern University
Boston, Massachusetts

December 2016
“Honra a tu padre y a tu madre, para que tus días se alarguen en la tierra que el Señor tu Dios te da.”

Exodo 20:12
# Contents

List of Figures vi
List of Tables viii
List of Acronyms ix
Acknowledgments x
Abstract of the Dissertation xi

1 Introduction 1

1.1 Frequency Response Methods 2
   1.1.1 Structural Model Updating 3

1.2 Filter-based Methods 3

1.3 Lyapunov Exponents 4

1.4 Non-destructive Evaluation 5

1.5 Machine Learning Approaches 5

1.6 Subspace Methods 6

1.7 Closed-loop Methods 7

1.8 Decision Methods 7
   1.8.1 Hypothesis Testing 7
   1.8.2 Bayes Decision Rule 8
   1.8.3 Sequential Hypothesis Testing 9
   1.8.4 ROC Curves 10

2 Using the Relevance Vector Machine for Anomaly Detection 13

2.1 Introduction 13

2.2 Preliminaries 14

2.3 Sparse Density Estimation 16

2.4 Examples 17
   2.4.1 Structural Health Monitoring 19

2.5 Conclusions 24
List of Figures

1.1 ROC curve for random classifier ................................................. 11
2.1 2-D sparse density estimate .................................................. 17
2.2 3-D sparse density estimate .................................................. 18
2.3 Structure model ................................................................. 19
2.4 Estimated distribution of transformed data ............................ 21
2.5 Normal and outlier samples with damaged $k_1$ ................. 21
2.6 ROC curves ........................................................................... 22
2.7 Normal and outlier samples with damaged $k_1$ ................. 23
2.8 ROC curves ............................................................... 23
3.1 $Q$-order vs. cost ................................................................. 32
3.2 $N - K$ versus CPU time in seconds .................................... 34
4.1 Hamming distance histograms .............................................. 48
4.2 Hamming Distance CDF ......................................................... 49
5.1 Bimodal distribution .............................................................. 58
5.2 ROC curves when 50 points are used for training ............... 59
5.3 ROC curves when 300 points are used for training ............... 59
5.4 2D distribution and test points ............................................. 61
5.5 ROC curves when 100 points are used for training ............... 61
5.6 ROC curves when 300 points are used for training ............... 62
5.7 Swiss-roll distribution ......................................................... 63
5.8 ROC curves when 100 points are used for training ............... 64
5.9 ROC curves when 300 points are used for training ............... 64
5.10 AUC versus $r$ ................................................................. 66
5.11 Clean and noisy images ....................................................... 67
5.12 Upper-bound image and recovered contour ...................... 67
5.13 Bearing test rig [II] ........................................................... 68
5.14 Bearing 1 mean, RMS, kurtosis, and skewness ................. 69
5.15 Upper-bound probabilities of the four channels ............... 69
5.16 Running-average of upper-bound probabilities ............... 70
5.17 Bearing 3 mean, RMS, kurtosis, and skewness ............... 71
5.18 Bearing 4 mean, RMS, kurtosis, and skewness . . . . . . . . . . . . . . . . . . . . 71
5.19 Running-average of upper-bound probabilities . . . . . . . . . . . . . . . . . . 72
# List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Commonly used test statistics</td>
<td>8</td>
</tr>
<tr>
<td>1.2</td>
<td>Confusion matrix</td>
<td>10</td>
</tr>
<tr>
<td>1.3</td>
<td>Qualitative descriptions of classifier performance</td>
<td>12</td>
</tr>
<tr>
<td>2.1</td>
<td>AUC performance using running-average filters of order $n_d$</td>
<td>22</td>
</tr>
<tr>
<td>4.1</td>
<td>Mean and standard deviation of Hamming errors</td>
<td>48</td>
</tr>
<tr>
<td>4.2</td>
<td>Boundary cases</td>
<td>49</td>
</tr>
<tr>
<td>5.1</td>
<td>AUC Summary</td>
<td>60</td>
</tr>
<tr>
<td>5.2</td>
<td>Classification performance ($N = 300$)</td>
<td>60</td>
</tr>
<tr>
<td>5.3</td>
<td>AUC Summary</td>
<td>62</td>
</tr>
<tr>
<td>5.4</td>
<td>Classification performance ($N = 300$)</td>
<td>62</td>
</tr>
<tr>
<td>5.5</td>
<td>AUC Summary</td>
<td>65</td>
</tr>
<tr>
<td>5.6</td>
<td>Classification performance ($N = 300$)</td>
<td>65</td>
</tr>
<tr>
<td>5.7</td>
<td>Classification performance using LUTs ($N = 300$)</td>
<td>73</td>
</tr>
</tbody>
</table>
List of Acronyms

**AD**  Anomaly Detection.
**DD**  Damage Detection.
**SHM**  Structural Health Monitoring.
**RVM**  Relevance Vector Machine.
**DP**  Dynamic Programming.
**ROC**  Receiver Operating Characteristic.
**TP**  True Positive.
**FP**  False Positive.
**FN**  False Negative.
**TN**  True Negative.
**TPR**  True Positive Rate.
**FPR**  False Positive Rate.
**AUC**  Area Under Curve.
**PDF**  Probability Density Function.
**RVM**  Relevance Vector Machine.
**SVM**  Support Vector Machine.
**DP**  Dynamic Programming.
**SDP**  Semidefinite Programming.
**HMM**  Hidden Markov Model.
**SVD**  Singular Value Decomposition.
**QCQP**  Quadratically Constrained Quadratic Program
**GMP**  Generalized Moment Problem.
Acknowledgments

I would like to thank Drs. Mario Sznaier and Octavia Camps for allowing me to be part of the Robust Systems Lab and showing me how professionals do optimization, and for introducing me to moments and the GMP. It is probably the coolest thing I learned in graduate school. Thank you, also, for funding part of my studies and for the support throughout my tenure.

To Dr. Bernal, thank you for teaching me about system identification, structural health monitoring, and most of my knowledge of the singular value decomposition. Thank you, also, for the discussions we had in Cushings Hall.

To Dr. Sara Wadia-Fascetti and the NSF IGERT program, thank you for funding most my studies and for giving me the opportunity to learn about civil engineering and public policy.

To Dr. Nilay Roy, thank you for showing me around the Discovery Cluster and supporting my work there. Much of my work would not have been possible without this resource and support.

Last, but not least, I would like to thank my family for their support. I hope we can catch up on some missed family time.
Abstract of the Dissertation

Anomaly Detection Using Polynomial Optimization Methods

by

Jose A. Lopez

Doctor of Philosophy in Electrical and Computer Engineering

Northeastern University, December 2016

Dr. Mario Sznaier, Advisor

This dissertation is concerned with the problem of detecting anomalies in the behavior of a system. This problem is encountered by most scientific communities and, as such, has come to be known by various names. Here, we refer to this problem generically as Anomaly Detection. As this work was funded primarily by the NSF IGERT program, the context will sometimes focus on the detection of damage in civil structures. The tools applied to this problem come from various areas: machine learning, linear filtering theory, and recent results from polynomial optimization.

The first contribution introduces a method for sparsifying a (kernel) probability density estimate. This method is motivated by the fact that structural health monitoring data sets are often large and associated kernel density estimates become exceedingly complicated. In particular, the kernel density estimates use the same number of kernels as data points. We employ a Relevance Vector Machine (RVM) and show that, by using the same type of kernel in both the density estimate and RVM, the problem reduces to a regression problem.

The next contribution is a method for sparsifying an $\mathcal{H}_2$-optimal linear filter. The solution is motivated by the need to make optimal estimations using fewer sensors, or while satisfying other communication channel constraints. We use the Youla parametrization for filters to show the problem of synthesizing sparse filters is equivalent to a convex program with equality constraints. We also provide a novel modification to the traditional linear programming (LP) relaxation for the 0-1 knapsack problem and use the modified algorithm to obtain optimal sparsity patterns, for lower-dimensional systems, when only the number of available channels is known. The modified algorithm is shown to improve performance over the standard LP relaxation approach.

Next, we present a new method for what is known to be the most difficult hidden Markov model (HMM) problem: the training problem. We employ a polynomial optimization approach to train discrete HMMs using low-order moments estimated from the observations. We compare against
standard and new approaches and show that our method does not suffer from many of the drawbacks while maintaining important advantages. Moreover, the drawbacks in our approach are encapsulated in current semidefinite programming (SDP) solver technology. In the context of anomaly detection, models obtained using our method can be used to compute the probability of observing a particular sequence of observations – which can be used to decide whether the observed sequence is anomalous. Alternatively, when a large number of observations are available, the low-order moments can be estimated from the data and compared against the moments computed analytically using our models.

We follow by presenting a new method for detecting anomalies based on recent results in polynomial optimization and the generalized moment problem (GMP). The method requires only information about the statistical moments of the normal-state distribution and compares favorably with existing approaches (such as Parzen windows and 1-class SVM) – especially when working with smaller, or higher dimensional, data sets.

Finally, we provide a MATLAB toolbox for implementing moments-based optimization solutions and code for the HMM examples.
Chapter 1

Introduction

There are certain tasks that an engineer encounters frequently. For example, finding an “optimal” solution to a problem, whatever optimal may be, another is identifying inconsistencies in data. Both of these activities are often encountered throughout the scientific community and have come to be known by many names. This work is about the latter, which we refer to as Anomaly Detection (AD) but it can also be described by any of the following:

- Outlier Detection
- Novelty Detection
- Damage Detection
- Change-point Detection
- Fault Detection
- Failure Detection

Naturally, different disciplines attach slightly different meanings to these descriptions and sometimes there are differences even within the same discipline, depending on the application. However, the differences are subtle and not widely agreed upon. Furthermore, all of these methods include, in some fashion, two components: a model which represents a normal state and a method for deciding when a datum is anomalous.
CHAPTER 1. INTRODUCTION

Example 1. Consider a gadget that is used to determine when water is boiling. The device could use a thermometer to measure the temperature of the water and signal the boiling state when the temperature is greater than or equal to 212° F. In this example, the model for normal, un-boiling, temperatures are measurements under 212° F and a decision is made using a simple threshold. A more sophisticated AD method could take into account, for example, the uncertainty in the temperature measurements and/or the boiling point of the water.

In the sequel we list various techniques which are used, by various scientific communities, to design AD systems. Since this work was primarily funded by the NSF’s IGERT program, the context is AD systems as part of a broader Structural Health Monitoring (SHM) system for civil structures – when this is the case we use the term Damage Detection (DD).

When mentioning SHM systems, we mean an SHM system as defined by Rytter [2]: a system with 4 levels of monitoring

Level 1 Detection of damage
Level 2 Localization of damage
Level 3 Assessment of damage
Level 4 Consequences of damage

1.1 Frequency Response Methods

The basic idea behind frequency response methods is that the modal parameters of a structure (natural frequencies, mode shapes, and damping) are related to the physical properties of the structure (mass, damping, and stiffness) and that a change in the physical parameters can be detected by monitoring the frequency response [3, 4]. Data from a normal state is needed for these methods to succeed. The before-and-after frequency responses can be analyzed directly or indirectly. For example, in [5] the authors used the fact that damage tends to increase phase lag to classify a frequency response as coming from a damaged structure; the authors use the sum-squared phase error to distinguish the normal from the damaged responses. It is also possible to first fit the data to a parametric model, like an ARMAX model, using system identification methods and then examine the associated modal parameters for changes [6].

While frequency response methods have been used successfully in practice [3], they are somewhat controversial because the sensitivity of a particular modal parameter depends greatly on
CHAPTER 1. INTRODUCTION

the application [2]. For example, natural frequencies may be used effectively to detect damage in some manufacturing applications but not for offshore platforms [2]. There is also the problem of differentiating between changes caused by damage from changes caused by naturally occurring phenomenon like the weather or operational load changes [3]. This may be the biggest challenge in DD.

1.1.1 Structural Model Updating

In model updating, a reference model of a structure, for example:

\[ M\ddot{x}(t) + C_d \dot{x}(t) + Kx(t) = f(t) \]  \hspace{1cm} (1.1)

where the \( M \) matrix specifies the masses of the structure, \( K \) models the stiffness of the links between the masses, \( C_d \) models the damping of the structure, and \( f \) is the force acting on the structure to produce the displacement vector \( x \), is updated to agree with new measurements and the updated parameters are compared to a reference model to provide Level 1-3 information about the damage [7, 3]. The parameters are updated using optimization methods. Consequently, this method can be approached from as many directions as there are for solving optimization problems. For example, the updating procedure can minimize the difference between the reference and recorded natural frequencies and mode shapes [7]. In [8], a method that minimized the rank of the stiffness parameters was presented. The optimization problem can also incorporate various constraints like: preservation of matrix symmetry, sparsity, and positivity [3].

1.2 Filter-based Methods

Linear filters have a long history in AD [9]. By linear filters, we mean filters of the Luenberger observer type with update equation:

\[ \hat{x}_{k+1} = A\hat{x}_k + L(y_k - C\hat{x}_k) \]  \hspace{1cm} (1.2)

where \( \hat{x} \) is the estimate of the state of the model, \( y \) is the measurement of a signal containing information about the actual state \( x \), the \( A \) and \( C \) matrices model the instrumented structure, and \( L \) is a gain matrix used to optimize the contribution of the estimation error in the update, in some sense.
CHAPTER 1. INTRODUCTION

When the noise affecting the system is assumed white and Gaussian, and \( L \) is chosen to minimize the variance of the estimation error, \( L \) is called the Kalman gain, after Rudolf Kalman.

The estimation error term in parenthesis is called the innovations sequence \([10]\). When \( L \) is the Kalman gain, and the modeling assumptions agree with reality, the innovation sequence is known to be white \([11]\); this fact can be used for AD. In \([10, 11, 12]\), hypothesis-testing was used to decide if the innovations were produced from the assumed model or not. The covariance function of the innovations was computed, for a chosen set of delays, and squared – the sum of squared Gaussian variables is known to have the \( \chi^2 \) distribution and the corresponding cumulative distribution function was used to select a threshold that corresponds to a desired Type I error.

Clearly, this method works best when the structure model is accurate; otherwise the innovations sequence will not be white from the start.

1.3 Lyapunov Exponents

Lyapunov exponents are an attempt to quantify how fast two trajectories of a system starting near each other move away from each other. Let \( x(t) \) and \( y(t) \) be two solutions of the 1-d system

\[
\dot{x} = f(x, t), \quad t \geq 0, \quad x(0) = x_0
\]

with \( f \) assumed to be Lipschitz (so that we can assume solutions exist and are unique) with Lipschitz constant \( k_T \). Suppose \( x \) and \( y \) start at \( x(0) \) and \( y(0) \), respectively. Then by the Bellman-Gronwall lemma \([13]\), the relationship between the distance in the starting points and the distance in the trajectories, as they evolve in time, is given by:

\[
|x(t) - y(t)| \leq |x_0 - y_0|e^{k_T t}
\]

which we can rearrange into:

\[
\frac{|x(t) - y(t)|}{|x_0 - y_0|} \leq e^{k_T t}
\]

\[
\frac{1}{t} \ln \frac{|x(t) - y(t)|}{|x_0 - y_0|} \leq k_T
\]
CHAPTER 1. INTRODUCTION

This expression helps clarify the definition of a Lyapunov exponent, $\lambda$, given below:

$$\lambda = \limsup_{t \to \infty} \left( \frac{1}{t} \ln \left| \frac{x(t) - y(t)}{|x_0 - y_0|} \right| \right)$$

which attempts to characterize what happens to the two trajectories in the long run.

When $f$ is vector-valued, we call the set of exponents $\{\lambda_1, \ldots, \lambda_n\}$ the Lyapunov spectrum. See [13] for more information. So, in theory the Lyapunov spectrum can serve as an invariant of a system that can be monitored for changes. The advantage of this method is that it does not assume much about $f$. In particular, it does not assume $f$ is linear; which sets it apart from other methods listed in this introduction. However, Lyapunov exponents are not easy to calculate from experimental data because some of the exponents may be much larger than others and estimates for $\lambda$ tend to distort in the direction of the larger Lyapunov exponents. Still, some authors have succeeded in using them for DD.

In [14], the Lyapunov exponent spectra was extracted from data and plotted as a function of location to provide (visual) Level 1 & 2 damage information for a highway bridge modeled by a non-linear finite element model. In [15], the authors used Lyapunov exponents for DD in a monumental arch. In [16], the authors offer a “robust alternative” to Lyapunov exponent called the “Jacobian feature vector”, which is shown to detect damage in numerical and experimental test cases.

1.4 Non-destructive Evaluation

In this method, inspectors use a sensing method to observe the physical condition of the structure, or component, in order to obtain up-to-level-3 damage information. Here, the normal state is provided by the knowledge and experience of the person doing the inspection – the decision is also obtained the same way. Some examples include using ground-penetrating radar to detect corroded rebar on a highway bridge [17] and infrared thermography for detecting damage in aerospace composite materials [18]. However, it can be as simple as an inspector using a flashlight to detect surface damage on a component [18]. See [19, 18, 20, 2] for more information.

1.5 Machine Learning Approaches

Machine learning, generally speaking, is concerned with solving regression or classification problems. Therefore, machine learning methods lend themselves to solving AD problems in either
CHAPTER 1. INTRODUCTION

the modeling or decision components, or both. The most common machine learning schemes used for AD are neural networks and the support vector machine (SVM). For example, in [21], a model representing the normal state was used to train an SVM to classify whether sensor measurements are landmines. In [22], a neural network was trained to detect boiling coolant in a nuclear reactor – in this case, the normal state was provided by the author when labeling the training data for the SVM. In [23], the authors used a 1-class SVM to learn the normal state of time-series data and classify anomalies. Later, in Chapter 2, we explain how another machine learning method, the relevance vector machine (RVM), could be used to solve an AD problem.

1.6 Subspace Methods

In subspace methods, the normal state is represented by a Hankel matrix, $H_0$, constructed using (presumably) normal measurements. In particular, the Singular Value Decomposition (SVD) of $H_0$ is used to factor $H_0$ into two matrices, one of which is referred to as the “observability” matrix. The left nullspace, $S$, of the observability matrix contains the information of the normal state. When new measurements are used to construct a second Hankel matrix $H_1$, the residual vector

$$\zeta_1 = \sqrt{N} \text{Vec}(STH_1)$$

quantifies how much the new data agrees with the normal model. Hypothesis-testing, or even a simple threshold, could then be used to decide whether new data is anomalous or not. See [24, 25].

In [26], the authors showed that this approach can succeed in distinguishing activities in video. In [27], the authors showed that the method does not need long data sets to succeed even in the presence of noise. Here, the authors defined a distance function,

$$d(H_0, H_1) = 2 - \|H_0H_0^T + H_1H_1^T\|_F$$

which was shown to be robust to noise, to label the normal states and then used these data to train an SVM to determine if new Hankel matrices conform to learned states.

The method was shown to succeed using Hankel matrices generated using only 15 measurements.
CHAPTER 1. INTRODUCTION

1.7 Closed-loop Methods

Some methods monitor the closed-loop behavior of a system for changes in control effort to detect changes. For example, in [28], damage in laminates was detected by placing a specimen under (displacement) closed-loop control and looking for a drop in control effort; which presumably corresponds to a decrease in stiffness. Similarly, in [29] the author used system identification methods to construct a model of an induction motor in a compressor – the model was then used to estimate the load torque and motor shaft speed signals, which were then compared against actual sensor measurements. In both examples, a metric/threshold pair were used to decide if the measured states were anomalous.

1.8 Decision Methods

There are many methods that can be used to decide whether a given measurement is anomalous, with respect to a normal model. Indeed, there is an entire area of study called Decision Theory that is concerned with making optimal decisions. (The decision component of an AD system is often neglected because the modeling part is generally more difficult [30].) Here, we review hypothesis testing, its sequential version, and receiver operating characteristic (ROC) curve methods.

1.8.1 Hypothesis Testing

Hypothesis testing may be the most natural decision method that can be used in the decision part of an AD system. (The four steps of the procedure themselves constitute an AD system!) Hypothesis testing attempts to answer the question: what are the chances that an observation is improbable but still normal [31]? If one imagines the bell curve of a Gaussian distribution, the tail ends are improbable but there is still a non-zero chance that they will be observed. The four parts of a hypothesis test are listed below:

**Step 1** Select the “null” hypothesis, $H_0$, which represents the default, normal, hypothesis; this is our normal model.

**Step 2** Select a “test statistic” to compare the observed measurement(s) to the normal model. This is whatever function is used to measure the distance between the measurement and what we call normal. For example, Equations (1.9) and (1.8).
CHAPTER 1. INTRODUCTION

Step 3 Convert output from test statistic to a conditional probability, the so-called P-value. The P-value represents the probability of the observation, given that it came from the normal model. In the previous examples this step was lumped-in with the distance function. In [31], the following convention was used to describe P-values:

. The observed difference is called “not significant” if $p > 0.1$
. The observed difference is called “marginally significant” if $p \leq 0.1$
. The observed difference is called “significant” if $p \leq 0.05$
. The observed difference is called “highly significant” if $p \leq 0.01$

Step 4 Select a threshold $\alpha$ and decide against the null hypothesis if the P-value is lower than $\alpha$. 0.1 and 0.05 are commonly used values for $\alpha$ [32].

Some commonly used metrics in Step 3 are listed below.

<table>
<thead>
<tr>
<th>Name</th>
<th>Equation</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chi-squared</td>
<td>$\chi^2 = \sum_{\text{categories}} \frac{(n_{\text{observed}} - n_{\text{normal}})^2}{n_{\text{normal}}}$</td>
<td>$n_{\cdot\cdot}$ is the number of (\cdot) data points</td>
</tr>
<tr>
<td>z-statistic</td>
<td>$z = \frac{x - \mu_0}{\sigma_0 / \sqrt{n}}$</td>
<td>$\mu_0$ is the normal average, $\sigma_0$ is the known standard deviation</td>
</tr>
<tr>
<td>t-statistic</td>
<td>$t = \frac{x - \mu_0}{s / \sqrt{n}}$</td>
<td>$\mu_0$ is the normal average, $s$ is the standard deviation of the samples</td>
</tr>
<tr>
<td>f-statistic</td>
<td>$f = \frac{\text{explained variance}}{\text{unexplained variance}}$</td>
<td></td>
</tr>
</tbody>
</table>

Table 1.1: Commonly used test statistics

There is still a possibility that we can make the wrong decision. We say that we have made a Type I error if we reject $H_0$ even though it is true and we say we have made a Type II error if we retain $H_0$ even though it is false [31]. It can be shown that if we base our decision on the posterior probabilities instead, we can minimize the average probability of making the wrong choice [32].

1.8.2 Bayes Decision Rule

Bayes decision rule compares the posterior probabilities directly. That is,

Decide $H_0$ if $P(H_0|x) > P(H_1|x) \leftrightarrow P(x|H_0)P(H_0) > P(x|H_1)P(H_1)$ (1.10)
Decide $H_1$ otherwise.
where $x$ is a new measurement. We can use this approach to compare hypotheses if we had prior information about them (i.e., if we know $P(H_i)$). Traditional hypothesis testing asks: what is the probability of the data given the hypothesis $H_0$? With the Bayesian approach, one asks: what is the probability of hypothesis $H_i$ given the data? Working with posterior probabilities has the advantage of allowing us to minimize a loss function, in case the cost of making a Type I error is different from the cost of making a Type II error. See [30] for more discussion on the advantages of working with posterior probabilities.

The right-most inequality in (1.10) can be rearranged to give:

$$
\text{Decide } H_0 \text{ if } \frac{P(x|H_0)}{P(x|H_1)} > \frac{P(H_1)}{P(H_0)}, \quad \text{Decide } H_1 \text{ otherwise.}
$$

(1.11)

The left side of (1.11) is called the Bayes factor [30] or the likelihood ratio [32]. Since the term on the right is independent of the data, one can think of this decision rule as selecting $H_0$ if the likelihood ratio is greater than some threshold [32].

### 1.8.3 Sequential Hypothesis Testing

So far, the hypothesis testing discussion has assumed that the data is entirely available at decision time. For some applications, it may be desirable to make a decision as soon as possible. For example, in safety applications or any application where earlier decisions result in cost savings. In these cases, one can use the sequential version of hypothesis testing.

The general idea of sequential hypothesis testing when examining only two alternatives is to examine an incoming measurement and stop if we are able to make a conclusion about either $H_i$; if not, wait for the next sample [33]. Although not rigorously proven in [33], Wald found that a substantial reduction (often 50%) in the number of samples needed to reach a conclusion could be obtained.

To describe the procedure, let $\alpha$ be the chosen probability of making a Type I error and $\beta$ the chosen probability of making a Type II error. Let the likelihood ratio \(^2\) be given by:

$$
\Lambda(x) = \frac{P(x|H_1)}{P(x|H_0)}
$$

(1.12)

where $x = \{x_1, x_2, \ldots, x_n\}$. Then the sequential hypothesis procedure can be summarized as

\(^2\)A. Wald, the author credited with being the father of sequential hypothesis testing, defined the likelihood ratio as the inverse of the Bayes factor.
CHAPTER 1. INTRODUCTION

follows:

\begin{align*}
\text{Decide } H_1 & \text{ if } \Lambda(x) \geq A(\alpha, \beta) \quad (1.13) \\
\text{Decide } H_0 & \text{ if } \Lambda(x) \leq B(\alpha, \beta) \quad (1.14) \\
& \text{ otherwise, wait for next sample.} \quad (1.15)
\end{align*}

where $A(\alpha, \beta)$ and $B(\alpha, \beta)$ denote the thresholds for which the probabilities of the Type I and Type II errors will (respectively) equal $\alpha$ and $\beta$. These are not easy to compute \cite{33}, but it was shown in \cite{33} that if $A$ and $B$ are taken as

\begin{align*}
A &= \frac{1 - \beta}{\alpha} \quad \text{and} \quad B = \frac{\beta}{1 - \alpha} \quad (1.16)
\end{align*}

the resulting error probabilities will almost equal $\alpha$ and $\beta$. For more information, see \cite{33, 34, 35}.

1.8.4 ROC Curves

Receiver operating characteristic (ROC) curves were developed by electrical and radar engineers during World War II to analyze classification accuracy of different radar receiver operators \cite{36, 37, 38}. Since then, they have gained popularity in diverse fields like: machine learning, psychology, and medical research due to its conceptually simple and visual representation \cite{36, 39, 37, 38}. We begin by considering the following table which shows every possible outcome of predicted and truth classes.

| Predicted Class | True Class |  |
|-----------------|------------|
| Positive        | Positive   | False Positive (FP) |
| Negative        | False Negative (FN) | True Negative (TN) |

Table 1.2: Confusion matrix

There are many interesting quantities that can be computed from this table (when filled with the number of decisions that result in TP, FP, FN, and TN). Here, we consider only true positive rate (TPR) and false positive rate (FPR) since they are used to make the ROC curves. For more information about other interesting quantities see \cite{40}.

\footnote{This table is called a confusion matrix \cite{40}.}
CHAPTER 1. INTRODUCTION

\[
\text{TPR} = \frac{TP}{TP + FN} \quad \text{and} \quad \text{FPR} = \frac{FP}{FP + TN} \quad (1.17)
\]

An ROC curve is a graph of FPR vs. TPR as shown in Figure 1.1. A classifier (using a single decision threshold) is represented as a point in ROC space. The point (0,0) corresponds to a classifier that always decides negative. One can think of this classifier as having an infinitely positive threshold – thus, this classifier never has any false positive errors. Similarly, the classifier at (1,1) always decides positive, one can think of it as having an infinitely negative threshold, and so it never has false negatives. As the threshold varies, a curve is generated connecting the two points. For a random classifier, its ROC curve is a line connecting (0,0) and (1,1). The point (0.5,0.5) corresponds to a random classifier using a threshold that guesses correctly half of the time. It follows that the ideal classifier has an ROC curve that is above the diagonal and crosses the point (0,1). (That is, the ideal ROC curve looks like a step function.)

![Figure 1.1: ROC curve for random classifier](image)

In Figure 1.1 classifier 2 is considered better than classifier 1 and both are considered better than random. In general, the curves for classifiers will cross – this means that one classifier is better for some thresholds and worse for others. The area under the curve (AUC) metric is a popular way of comparing the overall performance of classifiers. Clearly, the ideal classifier has an AUC of 1.
and the random classifier has an AUC of 0.5. Table 1.3 shows common qualitative descriptions for AUC scores [47].

<table>
<thead>
<tr>
<th>AUC Score</th>
<th>Classifier Quality</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.9-1.0</td>
<td>excellent</td>
</tr>
<tr>
<td>0.8-0.9</td>
<td>good</td>
</tr>
<tr>
<td>0.7-0.8</td>
<td>fair</td>
</tr>
<tr>
<td>0.6-0.7</td>
<td>poor</td>
</tr>
<tr>
<td>0.5-0.6</td>
<td>failure</td>
</tr>
</tbody>
</table>

Table 1.3: Qualitative descriptions of classifier performance

Given that each point along an ROC curve corresponds to a threshold, a reasonable question to ask is: is there a way to obtain an optimal threshold? One approach is to weigh the prevalence of FP and FN errors along with their costs [41, 42]. In [42], a slope is calculated using Equation (1.18). A line with this slope is moved from the upper left hand corner of the ROC graph, towards the lower right hand corner, until it touches the ROC curve. The point where the line touches the curve corresponds to the optimal threshold given the prevalence of FP and FN errors along with their costs.

$$m = \left(\frac{\text{cost of FP} - \text{cost of TN}}{\text{cost of FN} - \text{cost of TP}}\right) \times \left(\frac{\text{TN + FP}}{\text{TP + FN}}\right)$$  \text{(1.18)}
Chapter 2

Using the Relevance Vector Machine for Anomaly Detection

This chapter presents a new approach for sparsifying (Parzen) kernel density estimates using the relevance vector machine (RVM). It is shown that by using the same kernel in both the RVM and density estimate, the density estimation problem reduces to a simple regression problem.

2.1 Introduction

Probability density estimation is useful any time one wants to make inferences about a population given a finite number of samples. In anomaly detection, one would like to determine if a sample is very improbable. However, for estimation tasks, one may want to know the most likely sample. A complication that often arises is that the distribution of interest is multi-modal, so the engineer must use non-parametric techniques that make no assumption about the underlying distribution [43]. One of the most popular non-parametric methods for density estimation is the so-called Parzen estimator, also known as a kernel density estimator [30]. It has the desirable property that given enough samples it converges to the true distribution provided certain other conditions (discussed in Section 2.3) are met [30]. However, the number of kernels used in the estimate grows with each sample. This is undesirable from a computational point of view and it also leads to less interpretable models. In this chapter, we use a recent machine learning tool, the RVM, to greatly reduce the number of kernels needed to make a kernel estimate.

Much has been written about machine learning methods in recent years. The support vector machine (SVM), in particular, has proven to be very popular and effective at solving classification,
CHAPTER 2. USING THE RELEVANCE VECTOR MACHINE FOR ANOMALY DETECTION

regression, and novelty detection problems \cite{30}. The RVM, introduced by Tipping \cite{44}, also
produces linear kernel models but offers some advantages over the SVM while achieving similar
results. Key differences are that the RVM produces posterior probabilities and sparser models. See
\cite{44,45,46,30} for more discussion on the differences between the two. While the RVM has been
shown effective for regression and classification, we have not found literature on its use for density
estimation.

Here, we show how the RVM can be used to estimate the probability density function
(PDF) of a data set using a much smaller number of kernels. Section 2.2 provides an overview of
the theory behind the RVM – it is intended for the reader who wishes to use the RVM but does
not necessarily need to implement the algorithm. For more information, the reader is referred to
\cite{30,44,45,46} which are the references the next section is based on. Section 2.4 demonstrates the
technique via examples.

2.2 Preliminaries

Consider the problem of estimating a function \( y : \mathcal{X} \rightarrow \mathcal{T} \) that relates some input space \( \mathcal{X} \)
to some output space \( \mathcal{T} \). In regression problems, one often assumes the target data is of the form:

\[ t = y(x) + \epsilon \]  

(2.1)

where the variables are \( N \)-vectors and \( \epsilon \) is the measurement noise vector. When little is known about
\( \epsilon \), it is often assumed that it is Gaussian and zero-mean. For RVM regression, we assume further that:

\[ y(x) = \sum_{i=1}^{M} w_i \phi_i(x) \]  

(2.2)

where \( \phi_i \) is a kernel function and \( w_i \) is its weight. In the case where one wishes to include a bias,
one can take one of the kernels to be 1. With the assumed model, the probability of each target is
assumed to be centered at the estimate \( y(x) \). That is,

\[ p(t|y(x), \beta) = N(t|y(x), \beta) \]  

(2.3)
CHAPTER 2. USING THE RELEVANCE VECTOR MACHINE FOR ANOMALY DETECTION

with $\beta$ the (unknown) precision of the measurement noise. Assuming each measurement is independent and identically distributed, the likelihood function is given by:

$$p(t|y(x), \beta) = \prod_{n=1}^{N} p(t_n|y(x_n), \beta)$$

(2.4)

The RVM assumes a prior distribution for the weights given by:

$$p(w|\alpha) = \prod_{i=0}^{M} \mathcal{N}(w_i|0, \alpha_i)$$

(2.5)

where each weight $w_i$ has (unknown) precision $\alpha_i$. This prior is sometimes called an Automatic Relevance Determination (ARD) prior. The parameters of the prior are called hyperparameters to distinguish them from the other parameters in the model. Since the product of two Gaussians is also Gaussian, it follows that the posterior probability is Gaussian and given by:

$$p(w|t, y(x), \alpha, \beta) = \mathcal{N}(w|m, \Sigma)$$

(2.6)

where $m$ and $\Sigma$ are the mean and covariance, respectively. It can be shown that they are given by:

$$m = \beta \Sigma \Phi^T t$$

(2.7)

$$\Sigma = (A + \beta \Phi^T \Phi)^{-1}$$

(2.8)

where $\Phi$ is the $N \times M$ design matrix which has $\phi_i(x)$ in its $i$th column, and $A$ is a diagonal matrix with $\alpha$ along its diagonal. Notice that the posterior mean and variance are functions of knowns, $\Phi$ and $t$, and unknowns, $\beta$ and $\alpha$. To obtain the unknowns, the likelihood function (with the weights marginalized out) is maximized in a process known as evidence approximation or type-2 maximum likelihood:

$$p(t|x, \alpha, \beta) = \int p(t|x, w, \beta)p(w|\alpha)dw$$

(2.9)

Maximizing the log-likelihood of this function yields update equations for $\alpha_i$ and $\beta$ which are then used to recompute $m$ and $\Sigma$ and the process repeats until convergence. As the learning proceeds, many of the $\alpha_i \to \infty$, which implies that the distribution for the weight $w_i$ is increasingly peaked at zero and the corresponding kernel $\phi_i(x)$ can be removed from the model – this is how the RVM
is able to use fewer kernels than other methods. Kernels may also be returned to the model under certain conditions which serves to counteract greediness. The implementation we use here is the freely available SparseBayes MATLAB package.

The derivation for the classification case proceeds similarly except the model \( y(x) \) includes the logistic function which prevents the weights from being marginalized-out neatly and a likelihood approximation is maximized instead.

### 2.3 Sparse Density Estimation

Suppose we make \( N \) observations \( \{x_1, \ldots, x_N\} \) from an unknown probability density \( p(x) \), with \( x \) a \( D \)-vector in some Euclidean space. If we assume the density is constant over a small cube \( V = h^D \) of side \( h \), we can obtain the estimate:

\[
p(x) = \frac{1}{NV} \sum_{i=1}^{N} k \left( \frac{x - x_i}{h} \right) \tag{2.10}
\]

where \( N \) is the number of samples and \( k \) is a kernel function that satisfies the following conditions:

\[
k(u) \geq 0
\]

\[
\int k(u) du = 1 \tag{2.11}
\]

These conditions ensure that (2.10) is a PDF.

A proof was sketched in [43] that showed that, when (2.11) and the following conditions are satisfied, (2.10) will approach the true distribution as the number of samples increases.

\[
\sup_u k(u) < \infty \tag{2.12}
\]

\[
\lim_{\|u\| \to \infty} k(u) \prod_{i=1}^{D} u_i = 0 \tag{2.13}
\]

\[
\lim_{N \to \infty} V_N = 0 \tag{2.14}
\]

\[
\lim_{N \to \infty} NV_N = \infty \tag{2.15}
\]

where \( V_N = h_N^D \) is the volume of a cube with side \( h_N \). The first two conditions are meant to ensure \( k(u) \) is well-behaved and the latter two say that \( V_N \) has to approach 0 but slower than \( N \to \infty \).
CHAPTER 2. USING THE RELEVANCE VECTOR MACHINE FOR ANOMALY DETECTION

One thing that is apparent about (2.10) is that it becomes increasingly complicated as \( N \) increases. The RVM may be able to help keep the complexity of the model under control.

Notice that the estimate looks the same as the RVM regression model in Equation (2.2), if it had a single weight parameter determined by \( h \) (and the length of the data set). For a given \( h \), if we set \( \phi_i = k \left( \frac{x_i - \bar{x}}{h} \right) \), when constructing \( \Phi \), then (sparse) density estimation becomes an RVM regression problem with the targets given by:

\[
t = \frac{1}{NK} \sum_{\text{rows}} \Phi
\]  

(2.16)

The resulting (sparse) density estimate will inherit the properties of the full estimate. The user can choose to re-normalize the sparse estimate since there may be some small differences. The approach is demonstrated in the following Section.

2.4 Examples

Figure 2.1 shows this method applied to a synthetic 2-D data set using Gaussian kernels with \( h = 1 \). (The resulting probability surface has been normalized for illustration.) The data set contains 1,000 points. On the other hand, the RVM model uses only 47 kernels, which is just 4.7% as many kernels as the Parzen estimate. The relevant kernels are circled in green.

---

Figure 2.1: 2-D sparse density estimate

1 All the examples in this section use Gaussian kernels with \( h = 1 \).
CHAPTER 2. USING THE RELEVANCE VECTOR MACHINE FOR ANOMALY DETECTION

Generally, the more complex the distribution estimate, $t$, the more relevant kernels will be needed to estimate the density function. Figure 2.2 shows the method applied to a 3-D distribution sampled from the so-called swiss-roll equations.

![Figure 2.2: 3-D sparse density estimate](image)

The upper subfigure shows the 1,500 samples used for the estimate in blue, with the 519 relevant samples circled green. The other subfigure shows a cut-away of equal probability surfaces.
2.4.1 Structural Health Monitoring

Suppose we have a structure with 4 degrees of freedom as shown below.

```
\[
A = \begin{bmatrix}
0 \\
-M^{-1}K & -M^{-1}C_d
\end{bmatrix}, \quad B = \begin{bmatrix}
0 \\
-M^{-1}b
\end{bmatrix}
\]
```

where $M$ specifies the masses, $K$ models the stiffness of the links between the masses, $C_d$ models the
damping of the structure, \( u \) represents the applied loading, and \( b \) is used to specify where the loading is applied. In our example, we assume the loading is applied (only) at \( m_1 \), the load is modeled as a uniformly distributed random variable in \([-1, 1]\), and that the displacement and velocity information from the top level are sampled every \( T = 0.04 \) seconds. Assuming classical damping and nominal values \( m_1 = m_3 = 1 \), \( m_2 = m_4 = 2 \), and \( k_i = 100 \) we have:

\[
M = \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 2 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 2
\end{bmatrix}, \quad K = \begin{bmatrix}
200 & -100 & 0 & 0 \\
-100 & 200 & -100 & 0 \\
0 & -100 & 200 & -100 \\
0 & 0 & -100 & 100
\end{bmatrix} \quad (2.18)
\]

\[
C_d = \begin{bmatrix}
1.3760 & -0.4558 & -0.0454 & -0.0362 \\
-0.4558 & 1.7494 & -0.4920 & -0.1994 \\
-0.0454 & -0.4920 & 1.3125 & -0.5555 \\
-0.0362 & -0.1994 & -0.5555 & 1.1578
\end{bmatrix}, \quad b = \begin{bmatrix}
1 \\
0 \\
0 \\
0
\end{bmatrix} \quad (2.19)
\]

Since displacement and velocity information is available only at \( m_1 \), the 8x8 measurement matrix \( C \) is all zeros except for the \((1, 1)\) and \((5, 5)\) components, which are ones. (We approximate the acceleration using the velocity measurements.) The 8x1 feedthrough matrix \( D \) is all zeros. We construct five 20,000 sample datasets made by combining 10,000 normal samples and 10,000 with each spring perturbed. The fifth data set included all normal samples. For each dataset, the first 5,000 normal samples were used for training a Parzen density estimator and the remaining 15,000 mixed observations for testing.

To use the time-series observations, we follow the method in [48, 23] and embed the displacements in a 3-D space with the velocity and acceleration samples specifying the second and third coordinates – the embedding of measurements in this way is theoretically justified because it can be shown that information from a single coordinate suffices to reconstruct the attractor of a dynamic system [48]. Finally, because each coordinate is orders-of-magnitude apart from the rest, the data is poorly scaled. A whitening transform computed from the training set and used to preprocess every test sample prior to evaluating its probability score using the density estimate.

Unlike the previous examples, this data set suffers from poor scaling, with each component orders of magnitude apart from the rest. To overcome this limitation we use a whitening transform \( A_w \) to improve the scaling of the training samples. See [32] for more information. Figure 2.4 shows the transformed data and the fitted model, which uses just 128 kernels. The right-most subfigure
CHAPTER 2. USING THE RELEVANCE VECTOR MACHINE FOR ANOMALY DETECTION

shows the equal-probability surfaces for $p(x) = \{0.1, 0.2, \ldots, 0.9\}$.

Figure 2.4: Estimated distribution of transformed data

Figure 2.5 shows normal samples in blue and outlier samples in green. The ROC curves obtained for the five datasets are shown in Figure 2.6.

Figure 2.5: Normal and outlier samples with damaged $k_1$
CHAPTER 2. USING THE RELEVANCE VECTOR MACHINE FOR ANOMALY DETECTION

Figure 2.6: ROC curves

Intuitively, as a system becomes damaged the frequency with which trajectories reach improbable states should increase. If so, using a running-average filter to process the trajectories should improve detection capability. Table 2.1 shows that this is the case up to a filter of order $n_d = 9$. Figure 2.7 shows the normal and outlier samples after filtering with a filter of order $n_d = 9$. It is easy to see there is a larger separation between the samples of the normal and damaged samples. Figure 2.8 shows the improved ROC curve.

<table>
<thead>
<tr>
<th></th>
<th>AUC ($n_d = 3$)</th>
<th>AUC ($n_d = 5$)</th>
<th>AUC ($n_d = 7$)</th>
<th>AUC ($n_d = 9$)</th>
<th>AUC ($n_d = 11$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$k_1$</td>
<td>0.6184</td>
<td>0.6398</td>
<td>0.6653</td>
<td>0.6959</td>
<td>0.7261</td>
</tr>
<tr>
<td>$k_2$</td>
<td>0.5334</td>
<td>0.5333</td>
<td>0.5445</td>
<td>0.5623</td>
<td>0.562</td>
</tr>
<tr>
<td>$k_3$</td>
<td>0.4967</td>
<td>0.4936</td>
<td>0.4902</td>
<td>0.4824</td>
<td>0.4723</td>
</tr>
<tr>
<td>$k_4$</td>
<td>0.469</td>
<td>0.463</td>
<td>0.4601</td>
<td>0.458</td>
<td>0.4637</td>
</tr>
<tr>
<td>normal</td>
<td>0.506</td>
<td>0.5047</td>
<td>0.5048</td>
<td>0.502</td>
<td>0.4917</td>
</tr>
</tbody>
</table>

Table 2.1: AUC performance using running-average filters of order $n_d$
The sparsified kernel density estimates for the filtered training data used just 104 kernels compared to 5,000 for Parzen approach.
2.5 Conclusions

The RVM was very useful in sparsifying Parzen window distribution estimates. As expected, the number of kernels needed to approximate the distribution appeared to be proportional to the complexity of the underlying distribution. For the distribution shown in Figure 2.1, the RVM identified 51 relevant vectors, which is a 95% improvement over the Parzen estimate. For the more complicated swiss-roll distribution, in Figure 2.2, the RVM used 512 kernels – which is still 66% less kernels than the Parzen estimate. The structural health monitoring example used 98% less kernels.
Chapter 3

$H_2$-Optimal Filter Design Subject To Sparse Information Constraints

The material in this chapter is based on [49]. The work presents a solution to the problem of synthesizing filters which minimize the $H_2$ norm of the transfer function between the noise and the estimation error subject to sparsity constraints on its structure. We also consider the problem of selecting the $H_2$-optimal sparsity pattern given that $K$-channels are available.

The first problem is solved efficiently from an algebraic point of view and the solution leads to a dynamic programming (DP) algorithm which can be practical for smaller (dimensional) systems. The DP algorithm is then used to demonstrate the effectiveness the LP-relaxation-based algorithm as a means of solving the optimal sparsity pattern problem for larger systems that the DP algorithm cannot handle.

3.1 Introduction

There are several reasons why one might care about sparsity constraints. A sparse filter transfer matrix might be used to represent a configuration resulting from the loss of one or more sensors, or communication channels. A sparse representation might be desirable if monetary constraints limit the number of available sensors. In structural health monitoring, one may be interested in estimating the stresses at particular locations in a structure using observations corrupted by noise, like unmeasured wind loading, while using sensors with constrained information flows. In cases where computing power is limited, one might be interested in the computational efficiency that results from having a transfer matrix with many zero entries.
Several authors have studied control design subject to sparsity constraints, for example in [50, 51, 52] where a sufficient condition called \textit{quadratic invariance} was introduced to facilitate designs. \textit{Quadratic invariance} was shown to suffice for a controller to inherit the sparsity pattern of the Youla parameter; which in turn meant control designs could be synthesized using efficient (convex) optimization tools. It was later shown in [53] that quadratic invariance was also necessary for a control problem with sparsity constraints to be solvable using convex optimization tools.

The filtering problem has not received as much attention. In [49], we used the Youla parametrization for filters to show how to formulate the problem as a convex optimization problem with equality constraints and showed that satisfying the sparsity constraints increases the cost over the standard $H_2$-optimal filter. Here, we provide an explicit solution to the problem that improves upon our previous result in terms of cost and in speed of computation.

We also consider the problem of identifying the optimal pattern given that a $K$-subset of channels are available out of $N$ available to the standard optimal filter, which does not need to satisfy sparsity constraints. Our explicit solution to the first problem leads to a dynamic programming algorithm which is useful for smaller problems and for validating the LP-relaxation approach we use to attack the $K$-channel problem.

3.1.1 Notation

The notation is fairly standard with the exception of matrix indexing. We use $A^{ij}$ to denote the element in the $i$-th row and $j$-th column of a matrix $A$. $A^i$ refers to the $i$-th row of $A$, $A^j$ the $j$-th column of $A$, and $A_1$ and $A_2$ refer to two different matrices. When we write $A^*$, we mean the vector that results when the columns of $A$ are stacked on top of each other; i.e.,

$$A^* = \begin{bmatrix} A^*^1 \\ A^*^2 \\ \vdots \end{bmatrix}$$

(3.1)

For a given transfer matrix $G$, we let $\text{Pattern}(G)$ denote the operator which maps $G$ to a binary matrix with a 1 in place of every non-zero entry of $G$ and a 0 everywhere else.

We use $G = T + QF$ to denote the Youla parametrization for filters\footnote{See [49] for a review of how to compute the parametrization and also the proof of the existence of a $Q$ based on the zero-interpolation conditions.} where the transfer matrices $T$, $Q$, and $F$ belong to $\mathcal{RH}_\infty$. Each parametrization has an associated parametrization,
\[ G_e = T_e + QF_e, \] for the transfer function from the noise to the estimation error.

To ensure the Youla parameter \( Q \) belongs to \( \mathcal{RH}_\infty \), we use a finite impulse response (FIR) approximation; by increasing the order of this approximation, we can (in theory) get arbitrarily close to the optimal cost.

### 3.1.2 Problem Description

To summarize, given a binary matrix \( S \) containing the desired pattern, we give a closed-form solution to the problem:

\[
P_Q(S) \mapsto \begin{cases} 
\min_Q & \|T_e + QF_e\|_2 = \|Q\|_2^2 \\
\text{subject to} & Q = \sum_{i=0}^{n-1} Q_i z^{-i} \\
& \text{Pattern}(T + QF) \in S
\end{cases}
\]

The second problem considered is:

\[
P_S(K) \mapsto \begin{cases} 
\min_S & P_Q(S) \\
\text{subject to} & \sum_{i,j} S^{ij} = K
\end{cases}
\]

where \( K \) is a given natural number less than the total channels available to the standard optimal filter \( T \).

For convenience, we will refer to these problems as \( P_Q(S) \) and \( P_S(K) \).

### 3.2 The solution of \( P_Q(S) \)

The solution to \( P_Q(S) \) rests on the fact that the squared \( \mathcal{H}_2 \) norm of an FIR filter is the sum of squares of its coefficients. To treat this problem from an algebraic point of view, we begin by pointing out that \( T \) and \( F \) have the same denominator. So from here on we will treat the two as matrices of polynomials. For convenience, we will assume every polynomial in each matrix has the same degree. (This does not reduce generality because we can add higher-order terms with zero coefficients if needed.)
CHAPTER 3. $\mathcal{H}_2$-OPTIMAL FILTER DESIGN SUBJECT TO SPARSE INFORMATION CONSTRAINTS

It is intuitively convenient to think of sparsity constraints as “cancellation” constraints. For example, if $S_{ij} = 0$ but $\text{Pattern}(T_{ij}) = 1$, then we need to select $Q$ so that $Q^i F^j = -T_{ij}$; i.e., to “cancel” the non-zero term $T_{ij}$.

The product of the row $Q^i$ and the column $F^j$ can be represented as a product of a Toeplitz matrix containing the coefficients of $F^j$ and a vector containing the coefficients of $Q^i$.

3.2.1 Example 1

Suppose

$$
T = \begin{bmatrix}
9 + 10z^{-1} & 11 + 12z^{-1} \\
13 + 14z^{-1} & 15 + 16z^{-1}
\end{bmatrix},
$$

$$
F = \begin{bmatrix}
1 + 2z^{-1} & 3 + 4z^{-1} \\
5 + 6z^{-1} & 7 + 8z^{-1}
\end{bmatrix}
$$

and $Q$ is a first-order FIR approximation:

$$
Q = \begin{bmatrix}
q_{01}^{11} + q_{11}^{11}z^{-1} & q_{01}^{12} + q_{11}^{12}z^{-1} \\
q_{02}^{21} + q_{12}^{21}z^{-1} & q_{02}^{22} + q_{12}^{22}z^{-1}
\end{bmatrix}
$$

If $S_{11} = 0$, we need:

$$
\begin{bmatrix}
1 & 0 & 5 & 0 \\
2 & 1 & 6 & 5 \\
0 & 2 & 0 & 6
\end{bmatrix}
\begin{bmatrix}
q_{01}^{11} \\
q_{01}^{12} \\
q_{02}^{21} \\
q_{02}^{22}
\end{bmatrix} = -\begin{bmatrix}
9 \\
10 \\
0
\end{bmatrix}
$$

(3.5)

to cancel the $T_{11}$ term.

To consider other patterns, let

$$
F_1 = \begin{bmatrix}
1 & 0 & 5 & 0 \\
2 & 1 & 6 & 5 \\
0 & 2 & 0 & 6
\end{bmatrix},
F_2 = \begin{bmatrix}
3 & 0 & 7 & 0 \\
4 & 3 & 8 & 7 \\
0 & 4 & 0 & 8
\end{bmatrix}
$$

(3.6)
be the Toeplitz matrices associated with $F^{*1}$ and $F^{*2}$, respectively. Let

$$ Q_1 = \begin{bmatrix} q_{11}^0 \\ q_{11}^1 \\ q_{12}^0 \\ q_{12}^1 \\ q_{11}^{12} \\ q_{12}^{12} \end{bmatrix}, \quad Q_2 = \begin{bmatrix} q_{21}^0 \\ q_{21}^1 \\ q_{22}^0 \\ q_{22}^1 \\ q_{21}^{22} \end{bmatrix} $$  \hspace{1cm} (3.7)

be the vectors associated with $Q^{1*}$ and $Q^{2*}$, respectively. Then, Equation (3.5) can be written as:

$$ F_1 Q_1 = -T^{11} \hspace{1cm} (3.8) $$

and it is known that the minimum-norm least-squares solution is given by $Q_1 = -F_1^\dagger T^{11}$, with $F_1^\dagger = F_1^T (F_1 F_1^T)^{-1}$, the pseudo-inverse of $F_1$. Then we can write any single equality constraint as:

$$ \begin{align*}
T^{11} : & \quad Q_1 = -F_1^\dagger T^{11} \\
T^{12} : & \quad Q_1 = -F_2^\dagger T^{12} \\
T^{21} : & \quad Q_2 = -F_1^\dagger T^{21} \\
T^{22} : & \quad Q_2 = -F_2^\dagger T^{22}
\end{align*} $$

and if we need to cancel more than one entry in the same row, we can stack the Toeplitz matrices. For example, to cancel 2 entries:

$$ \begin{align*}
T^{11}, T^{12} : & \quad Q_1 = - \begin{bmatrix} F_1 \\ F_2 \end{bmatrix}^\dagger \begin{bmatrix} T^{11} \\ T^{12} \end{bmatrix} \\
T^{21}, T^{22} : & \quad Q_2 = - \begin{bmatrix} F_1 \\ F_2 \end{bmatrix}^\dagger \begin{bmatrix} T^{21} \\ T^{22} \end{bmatrix}
\end{align*} $$

In the foregoing example, to satisfy $S^{11}$, we compute:

$$ T^{11} : \quad Q_1 = \begin{bmatrix} 0.4697 \\ -0.6364 \\ -1.8939 \\ 0.2121 \end{bmatrix} $$
which has $\|Q_1\|_2 = 2.063$. Substituting the associated transfer matrix:

$$Q = \begin{bmatrix}
0.4697 - 0.6364z^{-1} & -1.8939 + 0.2121z^{-1} \\
0 & 0
\end{bmatrix}$$

(3.9)

into $T + QF$ gives:

$$G = \begin{bmatrix}
0 & -0.8482 - 1.697z^{-1} - 0.8488z^{-2} \\
13 + 14z^{-1} & 15 + 16z^{-1}
\end{bmatrix}$$

(3.10)

which has the desired pattern. From here, we can make the following observations.

**Observation 1.** $Q$ can be computed row-by-row. For example, if the cancellation set $C = \{ij \mid S^{ij} = 0\}$ only involves $Q_1^*$, then we can just leave the other rows of $Q$ as zeroes.

**Observation 2.** In general, for a fixed $Q$-order, there is a maximum number of cancellations we can make in any row. This number is determined by the number of $F_j$’s we can stack on top of each other before the system of equations becomes over-determined. Increasing the order of $Q$ tends to make the system of equations under-determined; which means there are more degrees of freedom with which to satisfy the constraints.

The next example shows how increasing the order of $Q$ lowers the cost.
CHAPTER 3. $\mathcal{H}_2$-OPTIMAL FILTER DESIGN SUBJECT TO SPARSE INFORMATION CONSTRAINTS

3.2.2 Example 2: Cost vs. Q-order

Consider the randomly-generated polynomial matrices below:

$$T = \begin{bmatrix}
0.507 + 0.752z^{-1} & 0.829 + 0.727z^{-1} & 0.294 + 0.581z^{-1} & 0.056 + 0.212z^{-1} \\
0.2 + 0.368z^{-1} & 0.627 + 0.094z^{-1} & 0.18 + 0.637z^{-1} & 0.817 + 0.543z^{-1} \\
0.427 + 0.942z^{-1} & 0.539 + 0.878z^{-1} & 0.926 + 0.651z^{-1} & 0.529 + 0.703z^{-1} \\
0.169 + 0.017z^{-1} & 0.651 + 0.014z^{-1} & 0.068 + 0.865z^{-1} & 0.694 + 0.956z^{-1}
\end{bmatrix}$$

(3.11)

$$F = \begin{bmatrix}
0.445 + 0.796z^{-1} & 0.52 + 0.775z^{-1} & 0.152 + 0.228z^{-1} & 0.571 + 0.796z^{-1} \\
0.085 + 0.691z^{-1} & 0.954 + 0.914z^{-1} & 0.848 + 0.321z^{-1} & 0.572 + 0.442z^{-1} \\
0.057 + 0.345z^{-1} & 0.074 + 0.783z^{-1} & 0.785 + 0.83z^{-1} & 0.286 + 0.446z^{-1} \\
0.629 + 0.947z^{-1} & 0.207 + 0.296z^{-1} & 0.271 + 0.822z^{-1} & 0.699 + 0.466z^{-1}
\end{bmatrix}$$

(3.12)

With $S = I$, we obtain the following plot which illustrates the order of $Q$ versus the cost of canceling $T^{11}$ and $T^{22}$. 

31
3.3 Algorithms for $P_S(K)$

Solving $P_S(K)$ is challenging because, in the worst case, there are $\binom{N}{K}$ patterns to search through. We can use our observations to improve the situation to some extent. From Observation 1, we can see that while there may be $\binom{N}{K}$ patterns, each row can only take on $2^a$ patterns, where $a$ equals the number of columns of $T$. Using this observation, we implemented the dynamic programming algorithm \[\text{Algorithm 1}\] which we use to validate results of other algorithms, for lower-dimensional systems.

The first obstacle is simply obtaining feasible patterns. For this, we used the “Revolving Door” algorithm found in \[\text{[54]}\] which recursively generates $K$-subsets from an $N$-set in constant time. Algorithm 1 suffers from two bottlenecks. First, the need to memoize $b2^a$ costs uses a lot of memory. The second problem is that even though computing $Q$ is computationally efficient, the number of patterns to evaluate explodes. The Figure below shows the performance\[\text{[5]}\] of this algorithm,

\[\text{Figure 3.1: } Q\text{-order vs. cost}\]

\[\text{We used a modest workstation for our simulations, which uses: Ubuntu 13.10 32-bit, 3.9GB RAM, Intel Core2 CPU}\]
Algorithm 1 Dynamic Programming Algorithm

1: **Input:** $T, F$ numerators, $K$ channels, $n$ FIR order
2: **Output:** The optimal pattern and cost: $best\_pattern$ and $best\_cost$
3: $S = \text{Pattern}(T)$
4: $F_k \leftarrow$ Toeplitz matrices of $k$ column of $F$
5: $b_k \leftarrow$ cancellation vectors from $T$
6: $t\_row \leftarrow$ number of rows in $T$
7: $n\_codes = 2^{t\_row}$
8: cost_table $\leftarrow n\_codes \times t\_row$ matrix with $-1$ entries
9: $best\_cost = \infty$
10: $best\_pattern = \emptyset$
11: pattern $\leftarrow \text{GetNextPattern}(S^*, K)$
12: **while** pattern $\neq$ last_pattern **do**
13: pattern_cost $= 0$
14:  **for each** row_pattern in pattern **do**
15: if cost_table(row_pattern, row) $\geq 0$ **then**
16: pattern_cost $=$ pattern_cost + cost_table(row_pattern, row)
17: else
18: $q = -A^\dagger b_k$ where $A$ is formed with appropriate $F_k$’s
19: if $\|Aq + b_k\| > 1E-3$ **then**
20: cost_table(row_pattern, row) $= \infty$
21: pattern_cost $= \infty$
22: break
23: else
24: cost_table(row_pattern, row) $= q^T q$
25: pattern_cost $=$ pattern_cost + cost_table(row_pattern, row)
26: end if
27: end if
28: if pattern_cost $\geq best\_cost$ **then**
29: break
30: end if
31: end for
32: if pattern_cost $<$ best_cost **then**
33: best_cost $=$ pattern_cost
34: best_pattern $=$ pattern
35: end if
36: pattern $\leftarrow \text{GetNextPattern}(S^*, K)$
37: end while
CHAPTER 3. $\mathcal{H}_2$-OPTIMAL FILTER DESIGN SUBJECT TO SPARSE INFORMATION CONSTRAINTS

using carefully coded M-code, for a randomly generated $5 \times 5$ benchmark, with a first-order $Q$.

![Figure 3.2: $N - K$ versus CPU time in seconds](image)

We found that Algorithm 1, could handle parametrizations of order about 10, but the number of patterns overwhelmed it for some $K$ values. Nevertheless, Algorithm 1 was useful for validating the LP-relaxation:

$$ P_S(K) \mapsto \left\{ \begin{array}{l}
\min_{x} c^T x \\
\text{subject to } w^T x = K \\
x \in [0, 1] \\
\sum_b x = 1
\end{array} \right. $$

where $c$ is the vectorized cost table, $w$ is a vector with the number of channels available in each row pattern (i.e., the number of 1’s in the row pattern), and $b$ is the number of rows in $T$. The last

4300 @ 1.8GHz x2, 29.5GB free space.
constraint makes sure one pattern gets selected for each row and prevents the solver from selecting the pattern of all 1’s for all the rows (corresponding to no cancellations) and “normalizing” \( w \) with \( x \) so that \( w^T x = K \).

The LP-relaxation performed well in practice. In order to use a computed solution \( x \), we simply rounded \( x \) and looked up the corresponding pattern. We also performed a pre-optimization step to reduce the number of variables needed to solve the LP-relaxation: for each row, we only stored the cheapest pattern that had a particular weight. (There is no need to store all the patterns that have a particular number of 1s.)

To test the algorithm, we generated 500 random, \( 10 \times 10 \), parametrizations of orders ranging between 1 and 5, and with randomly chosen \((N, K)\) configurations such that the maximum number of patterns did not exceed \( 25 \times 10^6 \). We also used a first-order \( Q \).

With this configuration, we found that the LP solutions matched the DP solutions exactly 457 times (more than 90% of the time). In 6 cases, the LP solution did not match the DP solution, but was feasible. In 37 cases, the LP solution returned infeasible solutions (that did not have \( K \)-channels). However, we found that 33 cases out of the 37 could be recovered by simply solving the LP program with the costs squared (16 matched the DP solution exactly). The remaining two cases were recovered by cubing the costs. In general, the quality of the LP solutions was good in the sense that the difference between the LP solution and the DP solution was much less than 1.

In light of the effectiveness of raising the (non-negative) cost-table to higher powers, we propose the following algorithm. Let

\[
\{ f_i : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0} \mid f_i(0) = 0 \text{ and } f(a) < f(b) \text{ for all } a \text{ and } b \text{ with } a < b \} \quad (3.14)
\]

denote a family of order-preserving functions. (For example, in the 500-case test we took \( f_1(x) = x \), \( f_2(x) = x^2 \), and \( f_3(x) = x^3 \).) And, let \( P_S(K, f_i) \) denote the program and \( f_i(c^T) \) denote the element-wise evaluation of \( f_i \) at \( c^T \):

\[
P_S(K, f_i) \mapsto \begin{cases} 
\min \ f_i(c^T) x \\
\text{subject to} \\
w^T x = K \\
x \in [0, 1] \\
\sum_b x = 1
\end{cases}
\]
Algorithm 2 Modified LP-relaxation Algorithm

1: **Input:** $T, F$ numerators, $K$ channels, $n$ FIR order
2: **Output:** The optimal pattern and cost.
3: for all $f_i$ do
4: $S_i \leftarrow$ Solution of $P_S(K, f_i)$
5: end for
6: best_cost $= \min_i P_Q(S_i)$
7: best_pattern $= S_i$ with best_cost $= P_Q(S_i)$
8: **Return:** best_cost, best_pattern

3.4 Conclusions

While the LP-relaxation was able to search through a huge number of patterns quickly, there was still the issue of having to compute a cost table that grew exponentially, even when excluding infeasible patterns. On our modest workstation, it took 3.34 hours to compute the cost table for a $20 \times 20$ parametrization with a $Q$-order of 1. However, we noticed that this procedure greatly benefited from parallelization. For example, on a quad-core machine, the same cost table was computed in 40 minutes. So, we are inclined to believe this method has some practical value – especially because 95% of the time it was able to match the solution obtained using exhaustive search and the remaining 5% of the time it produced nearly optimal solutions.
Chapter 4

A Semidefinite Optimization Approach
For Training Hidden Markov Models

We present a new iterative algorithm to what is widely regarded as the most difficult hidden Markov model (HMM) problem: the training problem. We employ a polynomial optimization approach to train discrete HMMs using low-order moments estimated from the observations. We provide details on the implementation including a modified re-weighting heuristic, compare against standard and new (spectral learning) approaches, and show that our approach: does not suffer from many of the drawbacks associated with these approaches while maintaining important advantages. Moreover, the drawbacks in our approach are encapsulated in current semidefinite programming (SDP) solver technology which is a heavily researched area that continues to improve.

4.1 Introduction

Hidden Markov models (HMMs) have a long history in the literature, since the 60s, and have been used to solve (once) challenging real-world problems, such as speech and character recognition [55]. Despite great advances in machine learning, HMMs are relevant today due to their ability to model discrete time-series data while offering efficient implementations that make them ideal solutions for running on devices with limited resources, like smart phones and watches.

Of the three HMM problems listed in [55]: evaluating the probability of a sequence of observations given a model, recovering the hidden state sequence given a model and observations, and the problem of estimating a model given observations, only the third problem remains open. There have been recent advances towards a solution for this problem – perhaps the most prominent of the

37
new approaches are the so-called “spectral learning” methods which compute spectral decompositions on moment matrices \[56, 57, 58, 59, 60, 61\]. These approaches are closely related to subspace system identification \[62\] and promise incredibly fast and statistically consistent training. Still, their drawbacks have not gone unnoticed \[63, 64\]. In particular, it is known that spectral methods cannot guarantee that the estimated HMM matrices will satisfy basic probability constraints. As pointed out \[63\], the root of the problem is that spectral learning tries to solve a system of polynomial equations but does not strictly enforce the probability constraints. Furthermore, training via spectral decompositions makes it difficult to enforce any constraints at all.

Strictly speaking, the method we propose here is also a method-of-moments approach because it uses first and second-order moments. However, our approach is more direct and has roots in polynomial optimization – it explicitly solves for the standard HMM representation (as opposed to observer-operator form) using semidefinite optimization. In this way, our method can easily handle structural constraints that are known to cause problems for spectral methods \[64\] and enforce basic probability constraints. Like spectral methods, our approach does not suffer from local maxima and is statistically consistent. However, our method explicitly enforces probability and moment-matching constraints – which we argue is more important than finding the exact HMM parameters since, in practice, an exact HMM may not exist.

Despite the achievements of spectral learning methods, as of this writing, the Baum-Welch approach is the most commonly used method for estimating the parameters of an HMM. Still, the Baum-Welch method has well-known drawbacks: local maxima, slow convergence, and estimates that are not statistically consistent in any sense. Our approach does not overcome the speed problem but there is room for improvement since SDP solvers continue to evolve.

4.2 Preliminaries

The notation mostly follows \[55\], which we used as our starting point, but we include a brief summary here for the reader’s convenience. We consider hidden Markov model with \(n\) states, \(\{s_1, s_2, \ldots, s_n\}\), and \(m\) symbols \(\{v_1, v_2, \ldots, v_m\}\). The state at time \(t\) is denoted as \(q_t\) and an observation at time \(t\) is denoted as \(o_t\). Thus, the first order Markov-chain assumption is written as:

\[
P[q_t = s_j \mid q_{t-1} = s_i, q_{t-2} = s_l, \ldots] = P[q_t = s_j \mid q_{t-1} = s_i].
\] (4.1)

The probability that the process transitions from state \(s_i\) to \(s_j\), is denoted by \(a_{ij} = P[q_t = s_j \mid q_{t-1} = s_i]\) for all \(1 \leq i, j \leq n\). The initial state distribution of state \(i\) is denoted \(\pi_i = P[q_1 = s_i]\), for all
CHAPTER 4. AN SDP APPROACH FOR TRAINING HMMs

1 ≤ i ≤ n, and the observation symbol distribution in state j is written \( b_j(k) = P[o_t = v_k | q_t = s_j] \) with 1 ≤ j ≤ n and 1 ≤ k ≤ m.

These probabilities are collected into transition probability matrix \( A \), initial probability vector \( \pi \), and output symbol probability matrix \( B \). An HMM model is denoted as \( \lambda = (A, B, \pi) \). Thus, the HMM training problem is to adjust \( \lambda \) to match reality in some sense.

Let \( \mathbb{N} \) be the set of positive integers. For any two matrices \( X \) and \( Y \) of equal size, we write \( \|X - Y\|_\infty \) for the vector norm \( \|\text{vec}(X) - \text{vec}(Y)\|_\infty \). When \( X \) is symmetric, we write \( X \succeq 0 \) to mean that \( X \) is positive semidefinite (i.e. it has non-negative eigenvalues).

Finally, we also make use of the following two polynomials:

\[
P_i \triangleq P[o_t = v_i] = \sum_{q_1q_2} \pi_{q_1} b_{q_1}(i) + \pi_{q_2} b_{q_2}(i) \tag{4.2}
\]

and

\[
M_{ij} \triangleq P[o_t = v_j, o_{t-1} = v_i] = \sum_{q_1q_2} \pi_{q_1} b_{q_1}(i) a_{q_1q_2} b_{q_2}(j) \tag{4.3}
\]

which are the first and second-order moments, respectively.

4.2.1 A Motivating Example

The following example is included to illustrate the drawbacks of the Baum-Welch approach. Consider, for example, the following perfectly reasonable initial guess:

\[
\lambda_0 = (A_0, B_0, \pi_0) = \left( \frac{1}{n} \cdot 1_{n \times n}, \frac{1}{m} \cdot 1_{n \times m}, \frac{1}{n} \cdot 1_{n \times 1} \cdot \frac{1}{n} \right) \tag{4.4}
\]

where \( 1_{x \times y} \) is an \( x \) by \( y \) matrix of ones. We call this a balanced initial guess. The reader can confirm through simulation that, starting from a balanced initial guess, the Baum-Welch algorithm will converge to the following model:

\[
A_{BW} \rightarrow A_0, \quad \pi_{BW} \rightarrow \pi_0 \tag{4.5}
\]
CHAPTER 4. AN SDP APPROACH FOR TRAINING HMMS

with $B_{BW}$ a matrix with repeated rows equal to:

$$
\frac{1}{T} \left[ \sum_{o_t=v_1} 1 \ \sum_{o_t=v_2} 1 \ \ldots \ \sum_{o_t=v_m} 1 \right]
$$

(4.6)

where $T$ is the number of observations. This all but guarantees that we will be unable to infer anything about the hidden state sequence, since each state is equally likely and equally likely to emit a given symbol – this model is effectively a 1-state model. Moreover, $A_{BW}$ and $\pi_{BW}$ will never change as $T$ increases.

4.2.2 QCQP Relaxations

Here, we review semidefinite programming (SDP) relaxation approach to quadratically-constrained quadratic-programs (QCQPs) \[65, 66, 67, 68, 69\]. Consider the following QCQP:

$$
\begin{align*}
\min_{x \in \mathbb{R}^n} & \quad x^T Q_0 x + q_0^T x \\
\text{subject to} & \quad x^T Q_i x + q_i^T x \leq \alpha_i, \ i = 1, 2, \ldots, l
\end{align*}
$$

(4.7)

where $l$ is a positive integer, $Q_i$ are real symmetric matrices, $q_i \in \mathbb{R}^n$, and $\alpha_i$ are real scalars. This is a non-convex problem in general. However, from the cyclic property of a trace ($x^T Q_0 x = \text{Trace}(Q_0 xx^T)$) one can write (4.7) as:

$$
\begin{align*}
\min_{X, x} & \quad \text{Trace}(Q_0 X) + q_0^T x \\
\text{subject to} & \quad \text{Trace}(Q_i X) + q_i^T x \leq \alpha_i, \ i = 1, 2, \ldots, l \\
& \quad X = xx^T
\end{align*}
$$

(4.8)

and the last (and only non-convex) constraint can be relaxed to the (convex) positive semidefinite constraint $X - xx^T \succeq 0$. After applying the Schur complement, we write:

$$
\begin{align*}
\min_{X, x} & \quad \text{Trace}(Q_0 X) + q_0^T x \\
\text{subject to} & \quad \text{Trace}(Q_i X) + q_i^T x \leq \alpha_i, \ i = 1, 2, \ldots, l \\
& \quad \begin{bmatrix} X & x \\ x^T & 1 \end{bmatrix} \succeq 0
\end{align*}
$$

(4.9)
which is an SDP. When a solution to (4.9) is found and the matrix containing $X$ and $x$ is rank-1, the solution is also optimal for (4.7) and $x^*$ can be recovered. While this approach applies to QCQP, higher-order polynomials can be cast as a QCQP problem by introducing additional, lifting variables and equality constraints. The advantage of doing so is that the resulting SDPs are smaller than other relaxations, for example.

4.3 Main Results

Consider the “urn and ball” example included in [55]. In this example, we consider $n$ urns in a room. Each urn contains $m$ different colored balls. A genie selects an initial urn, according to some distribution, and chooses a ball for an observer to note. The genie returns the ball and, according to her secret process, selects a new urn and the ball selection and presentation process is repeated.

Our method is based on the realization that any observer watching the chosen balls would notice patterns that linked selections in time. For example, if a red ball always (or never) followed a blue ball, the observer would notice this. We call this the output transition probability, which is the second-order moment, $M$. Similarly, the observer would notice if the overall frequency of each symbol changed. This is, of course, the first moment, $P$. Thus, we assert that any good $\lambda$ certainly generates observations that have the same $M$ and $P$. We say that any two models $\lambda_1$ and $\lambda_2$ that have the same first and second-order moments are statistically equivalent.

We restrict our method to pairwise observations to keep the order of the moment polynomials within the reach of current SDP solvers. Since $M$ and $P$ are estimated from the observations, it is clear that any $\lambda$ modeling a stationary process would improve, as the number of observations used for training increase, in the sense that the model will become statistically equivalent to the process under observation. This may be a more practical goal than trying to reproduce a single generating HMM because in general the process one is trying to model is more complicated than an HMM. In

\[1\text{The re-weighting heuristic in the main results can help obtain a rank-1 solution.}\]
this work, we take an optimization approach to moment-matching and solve the following program:

\[
\begin{align*}
\min_{A, B, \pi} & \quad \sum_{ij} |f_{ij}| + \sum_{i} |g_{i}| \\
\text{subject to} & \quad \sum_{j=1}^{n} a_{ij} = 1, \quad \sum_{j=1}^{m} b_{i}(j) = 1, \quad \sum_{i=1}^{n} \pi_{i} = 1 \\
& \quad a_{ij} \geq 0, \quad b_{i}(j) \geq 0, \quad \pi_{i} \geq 0 \quad i = 1, \ldots, m \\
& \quad j = 1, \ldots, n
\end{align*}
\]

where \( f_{ij} \triangleq M_{ij} - \hat{M}_{ij} \), \( g_{i} \triangleq P_{i} - \hat{P}_{i} \), and \( \hat{M} \) and \( \hat{P} \) are estimates for \( M \) and \( P \), respectively, computed from the observations. Taking this approach explicitly enforces probability constraints, moment-matching, and makes adding additional constraints (like imposing a left-to-right structure on \( A \) or including additional outputs) straightforward.

To use SDP relaxations to approximate \( \mathbb{P} \), first rename the HMM variables: \( a_{11} = x_{1}, a_{21} = x_{2}, \ldots \) and so on. Then associate one additional (lifting) variable with each quadratic term in the moment polynomials so that: \( g_{i} \) and \( f_{ij} \) can be written as linear and quadratic functions of the new variables, respectively. Lastly, add a “unity” variable that will satisfy the constraint \( x_{i}^{2} = 1 \). Then, \( x^{T} = [x_{1}, x_{2}, \ldots, x_{I}] \) and we can associate every monomial in \( \mathbb{P} \) with an entry in the matrix \( xx^{T} \). In this way, we can write \( \mathbb{P} \) in terms of \( X \). For convenience, we write \( \mathbb{P}(X) \) for the SDP relaxation of \( \mathbb{P} \).

### 4.3.1 Modified Re-weighting Heuristic For SDP Relaxations

The difficulty in applying SDP relaxations is usually one of two things: either the problem instance is too large for current solvers or one is unable to find a rank-1 solution, \( X \). There is little one can do (presently) about the first problem but a common approach for finding a rank-1 solution is to use matrix re-weightings [73, 74]. In this section, we present a modification to the usual technique that we found helps obtain a rank-1 solution with fewer iterations.
Begin by converting $\mathbb{P}$ to a feasibility program:

$$\mathbb{P}_\varepsilon \mapsto \begin{cases} 
\min_{A,B,\pi} & 0 \\
\text{subject to} & |f_{ij}| \leq \varepsilon \\
& |g_i| \leq \varepsilon \\
& \sum_{j=1}^{n} a_{ij} = 1, \sum_{j=1}^{m} b_i(j) = 1, \sum_{i=1}^{n} \pi_i = 1 \\
& a_{ij} \geq 0, b_i(j) \geq 0, \pi_i \geq 0 \\
& i = 1, \ldots, m \\
& j = 1, \ldots, n
\end{cases} \quad (4.11)$$

for some desired accuracy $\varepsilon > 0$. Then, add the objective $\text{Trace}(WX)$, where $W$ is a re-weighting matrix of compatible dimensions, to $\mathbb{P}_\varepsilon(X)$ as a proxy for $\text{Rank}(X)$. We can now summarize our method below.

**Algorithm 3 Iterative Moment Learning**

1. **Input:** $\hat{M}$, $\hat{P}$, $\varepsilon > 0$, $\eta > 0$, $\delta > 0$
2. **Output:** $X$
3. Initialize counter $k = 1$, $W^0 = I$, $\sigma_2 = \eta + 1$.
4. **while** $\sigma_2 > \eta$ **do**
5. **Solve** $\mathbb{P}_\varepsilon(X^k)$
6. **if** $X^k$ is feasible **then**
7. **return** $X^k$.
8. **end if**
9. **Compute SVD:** $X^k = USV^T$
10. $\sigma_2 \leftarrow S_{22}$
11. $W^{k+1} \leftarrow (\Delta(X^k) + I\delta)^{-1}$
12. $k \leftarrow k + 1$
13. **end while**

where $\Delta(X^k)$ returns the first principal component of $X^k$. When $\Delta$ is the identity operator, line 11 is the usual re-weighting heuristic found in the literature to minimize rank [73, 74]. However, we found that replacing $X^k$ with its principal component improves convergence of $\sigma_2 \to 0$. In the examples below, this modification cut the number of iterations by about half.
4.4 Examples

In the examples that follow, we compare against the Baum-Welch implementation from MATLAB [75] and the implementations of the spectral methods from [56, 57] that were made available in [76]. For our method, we used the SeDuMi SDP solver [77]. Code is included in Appendix [B].

4.4.1 Aliased States and Structural Constraints

Models with states that have the same output distributions are said to be aliased and are not handled by spectral algorithms [64]. Consider the following model with two aliased states:

\[
\lambda_{GT} = \begin{bmatrix}
0.2982 & 0.3081 & 0.2059 & 0.1878 \\
0.2465 & 0.3644 & 0.3377 & 0.0514 \\
0.1164 & 0.5033 & 0.1761 & 0.2042 \\
0.2039 & 0.2533 & 0.2697 & 0.273
\end{bmatrix},
\begin{bmatrix}
0.08532 & 0.08779 & 0.14 & 0.1594 & 0.2204 & 0.3071 \\
0.007775 & 0.1033 & 0.15 & 0.2548 & 0.3683 & 0.1158 \\
0.007775 & 0.1033 & 0.15 & 0.2548 & 0.3683 & 0.1158 \\
0.02594 & 0.02326 & 0.3375 & 0.0803 & 0.1336 & 0.3994
\end{bmatrix},
\begin{bmatrix}
0.4521 \\
0.3675 \\
0.08936 \\
0.09107
\end{bmatrix}.
\]

(4.12)

For this example the two spectral algorithms were unable to find models, as expected. \(\lambda_1\) and \(\lambda_2\), were obtained using our approach. For \(\lambda_2\), we arbitrarily imposed a left-to-right structure on
CHAPTER 4. AN SDP APPROACH FOR TRAINING HMMS

A.

\[
\lambda_1 = \begin{pmatrix}
0.324 & 0.3167 & 0.3594 \\
0.2379 & 0.4499 & 0.3122 \\
0.3057 & 0.3483 & 0.346
\end{pmatrix},
\]

\[
\begin{pmatrix}
0.0977 & 0.09377 & 0.1175 & 0.1691 & 0.2323 & 0.2897 \\
0 & 0.08781 & 0.1936 & 0.2473 & 0.3664 & 0.1048 \\
0.003135 & 0.08312 & 0.2028 & 0.1936 & 0.2807 & 0.2367
\end{pmatrix},
\]

\[
\begin{pmatrix}
0.4453 \\
0.2438 \\
0.3108
\end{pmatrix}
\]

(4.13)

\[
\lambda_2 = \begin{pmatrix}
0.328 & 0.3354 & 0.3365 \\
0 & 0.499 & 0.501 \\
0 & 0 & 1
\end{pmatrix},
\]

\[
\begin{pmatrix}
0.07701 & 0.09655 & 0.1379 & 0.1795 & 0.2471 & 0.262 \\
0.03107 & 0.08108 & 0.1682 & 0.1979 & 0.2899 & 0.2319 \\
0.02102 & 0.08897 & 0.1855 & 0.2128 & 0.3081 & 0.1836
\end{pmatrix},
\]

\[
\begin{pmatrix}
0.3582 \\
0.3396 \\
0.3023
\end{pmatrix}
\]

(4.14)

Both \( \lambda_1 \) and \( \lambda_2 \) satisfy \( \| M - \hat{M} \|_\infty < 0.002 \) and \( \| P - \hat{P} \|_\infty < 10^{-6} \) for \( \hat{M} \) and \( \hat{P} \) estimated using 50,000 observations. Our algorithm converged for \( \lambda_1 \) after 153 iterations vs. 286 without the modified re-weighting heuristic. \( \lambda_2 \) was obtained after 5 iterations using the heuristic vs. 8 without. (The run time is comparable to Baum-Welch with 50,000 observations.) The Baum-Welch
model below was obtained using a random initial condition:

\[
\lambda_{BW} = \begin{pmatrix}
0.3685 & 0.4433 & 0.1882 \\
0.03092 & 0.3955 & 0.5736 \\
0.3492 & 0.4662 & 0.1846 \\
0.01377 & 0.07868 & 0.1349 & 0.3584 & 0.2978 \\
0.03022 & 0.09065 & 0.1976 & 0.2228 & 0.1544 \\
0.02943 & 0.09026 & 0.1799 & 0.2232 & 0.1544 \\
1 & 0 & 0 \\
0 & 1 & 0 & 1
\end{pmatrix},
\]

\[
\lambda_{BW} \text{ has } \|M - \hat{M}\|_\infty = 0.0616 \text{ and } \|P - \hat{P}\|_\infty = 0.194. \text{ In particular, on average, this model outputs symbol } v_4 \text{ almost 20% more often than } \lambda_{GT}.\]

### 4.4.2 State Decoding

Consider the following ground truth model with 3 states and 5 symbols:

\[
\lambda_{GT} = \begin{pmatrix}
0.5722 & 0.3446 & 0.0832 \\
0.1933 & 0.6675 & 0.1392 \\
0.4729 & 0.5271 & 0.1392 \\
0.13 & 0.27 & 0.25 & 0.25 & 0.5 \\
0.3 & 0.4 & 0.2 & 0.05 & 0.05 \\
0.19 & 0.17 & 0.22 & 0.21 & 0.21 & 0.1
\end{pmatrix}, \quad (4.16)
\]

For this model, 50,000 observations were generated and used to train second-order models. (We did not assume knowledge of the number of states in this example.) For the Baum-Welch model, 5 models were made starting from different initial conditions\(^2\) and only the best one is shown below:

\[
\lambda_{BW} = \begin{pmatrix}
0.4420 & 0.5580 \\
0.2906 & 0.7094 \\
0.06289 & 0.2209 & 0.07178 & 0.333 & 0.3114 \\
0.3161 & 0.3848 & 0.2117 & 0.03745 & 0.05003 \\
0.3161 & 0.3848 & 0.2117 & 0.03745 & 0.05003 \\
0.01377 & 0.07868 & 0.1349 & 0.3584 & 0.2978 \\
0.03022 & 0.09065 & 0.1976 & 0.2228 & 0.1544 \\
0.02943 & 0.09026 & 0.1799 & 0.2232 & 0.1544 \\
1 & 0 & 0 & 1 & 0
\end{pmatrix}, \quad (4.17)
\]

\(^2\)One model was made using a balanced initial guess and 4 using random initial conditions.
CHAPTER 4. AN SDP APPROACH FOR TRAINING HMMS

The model below was trained using Algorithm 3:

\[
\lambda_1 = \begin{bmatrix}
0.6208 & 0.3792 \\
0.4564 & 0.5436 \\
0.2939 & 0.3797 & 0.2055 & 0.06045 & 0.06045 \\
0.14 & 0.2639 & 0.1139 & 0.2411 & 0.2411 \\
0.4157 \\
0.5843
\end{bmatrix},
\]

which converged using 40 iterations (vs. 115 without the heuristic). The models made using spectral algorithms had entries outside of \([0, 1]\), or with complex entries, even when allowing the same number of states as the ground truth model. The best spectral model was a second-order model made using the algorithm in [56], it had an emission matrix with one small negative entry. The entry was zeroed and the model rescaled to obtain:

\[
\lambda_2 = \begin{bmatrix}
0.8296 & 0.1704 \\
0.8168 & 0.1832 \\
0.1975 & 0.3692 & 0.1983 & 0.1785 & 0.05645 \\
0.4771 & 0.1935 & 0.08861 & 0 & 0.2408 \\
0.8295 \\
0.1705
\end{bmatrix},
\]

\[
(4.18)
\]

\[\lambda_1\] satisfies \(\|M - \hat{M}\|_\infty < 0.002\) and \(\|P - \hat{P}\|_\infty < 10^{-6}\) and \(\lambda_2\) satisfies \(\|M - \hat{M}\|_\infty \approx 0.023\) and \(\|P - \hat{P}\|_\infty \approx 0.078\). Thus, the spectral method does not always produce models with matching moments.

In order to evaluate these models, a data set of 100,000 trials was made using \(\lambda_{GT}\). Each trial generated ground truth observation and state sequences of lengths that varied (uniformly) from 2 to 100 points. Next, the models (including \(\lambda_{GT}\)) were used to find the most likely hidden state sequence\(^4\) for each trial and this sequence was compared to the true state sequence using the Hamming distance. All 100,000 Hamming distances were recorded for each model for comparison.

\(^3\)\(\lambda_{GT}\) was included in the comparison to give the reader an idea of the best case situation, given that the \(\lambda_1\) and \(\lambda_2\) have one less state.

\(^4\)See the MATLAB function \texttt{hmmdecode}.
Figure 4.1 shows the histograms of the Hamming error. \( \lambda_2 \) had the worst performance of all the models; this is due to the fact that the model greatly favors remaining in one state. The other three models had virtually identical performance, with \( \lambda_{BW} \) edging \( \lambda_1 \) slightly when it comes to the mean of the Hamming distances.

![Hamming distance histograms](image)

Figure 4.1: Hamming distance histograms

Table 4.1 shows the mean and standard deviations of the Hamming errors.

<table>
<thead>
<tr>
<th>Model</th>
<th>Mean</th>
<th>STD</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \lambda_{GT} )</td>
<td>0.33764</td>
<td>0.10402</td>
</tr>
<tr>
<td>( \lambda_{BW} )</td>
<td>0.34888</td>
<td>0.10419</td>
</tr>
<tr>
<td>( \lambda_1 )</td>
<td>0.34993</td>
<td>0.099025</td>
</tr>
<tr>
<td>( \lambda_2 )</td>
<td>0.63246</td>
<td>0.1345</td>
</tr>
</tbody>
</table>

Table 4.1: Mean and standard deviation of Hamming errors

\( \lambda_1 \) outperformed \( \lambda_{BW} \) at the lower and upper ends of the error spectrum as it had more trials with small errors and fewer trials with large errors. (See Table 4.2 and Figure 4.2) In any case, the Hamming distances were normalized with respect to observation sequence length.
the performance of $\lambda_1$ was comparable to both $\lambda_{GT}$ and the best Baum-Welch model and did not depend on finding a favorable initial condition.

<table>
<thead>
<tr>
<th>Model</th>
<th>No. Trials w/ Error $\leq 10%$</th>
<th>No. Trials w/ Error $\geq 90%$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\lambda_{GT}$</td>
<td>1821</td>
<td>240</td>
</tr>
<tr>
<td>$\lambda_{BW}$</td>
<td>1227</td>
<td>348</td>
</tr>
<tr>
<td>$\lambda_1$</td>
<td>1322</td>
<td>186</td>
</tr>
<tr>
<td>$\lambda_2$</td>
<td>714</td>
<td>2395</td>
</tr>
</tbody>
</table>

Table 4.2: Boundary cases

![Figure 4.2: Hamming Distance CDF](image)

4.5 Conclusions

We have presented a new iterative learning algorithm, based on polynomial optimization, for training HMMs. Our approach uses low-order moments, estimated from the observations, to produce models with matching moments. As with spectral methods, our approach does depend on initial conditions and is consistent (when the generating distributions are stationary). However, our method easily handles structural constraints and is more computationally expensive than spectral
approaches². We provided a modified re-weighting heuristic to improve convergence to some extent and there is room for improvement as SDP solvers evolve.

We demonstrated our method empirically using two examples. The first example is included to demonstrate the flexibility of our approach and the second example is included to illustrate the value of employing moment-matched HMMs: the estimated model achieved good performance without having to rely on a favorable initial guess.

²Using many observations, the run time of Algorithm 3 is comparable to Baum-Welch.
Chapter 5

Robust Anomaly Detection Using Semidefinite Programming

This chapter presents a new approach, based on polynomial optimization and the method of moments, to the problem of anomaly detection. The proposed technique only requires information about the statistical moments of the normal-state distribution of the features of interest and compares favorably with existing approaches (such as Parzen windows and 1-class SVM). In addition, it provides a succinct description of the normal state. Thus, it leads to a substantial simplification of the anomaly detection problem when working with higher dimensional data sets.

5.1 Introduction

Detecting anomalies in data is a task that engineers and scientists of every discipline encounter on a fairly regular basis. Common approaches to identifying anomalous observations include: estimating the probability density function of the “normal” state using density-estimation methods, computing the Mahalanobis distance between a point and a sample distribution, using machine-learning methods to learn the normal state and perform 1-class classification, using “whiteness” tests to detect when the differences between new samples and predictions (of linear models) become statistically colored, etc. However, a disadvantage encountered while estimating the distribution of the normal state is that one never knows if the available samples are representative of all the normal behaviors. This is especially true when there is a dearth of data with which to construct a reliable probability density estimate. In this chapter, we address this challenge through the use of recent results from the method of moments and polynomial optimization. The main idea is to use
information about the statistical moments of the normal-state distribution of a given set of features to
compute an upper-bound on the probability of a given observation of a random variable with this
distribution. The observation is then deemed to be anomalous when this bound falls below a given
threshold. While in principle computing this bound is a challenging infinite-dimensional problem, as
shown in the sequel, the use of concepts from the theory of moments allows for recasting it into a
computationally tractable finite dimensional optimization.

The generalized moment problem (GMP), defined below, has tremendous modeling power
and has found use in a wide range of applications in areas such as probability, financial economics,
engineering, and operations research [72]. However, its wider application has been held back because,
in its fullest generality, the GMP is intractable [78]. On the other hand, recent advances in algebraic
geometry have rendered the GMP tractable when the problem instance is described using polynomials
[71, 72].

Formally, the GMP is given by:

\[
\text{GMP: } \rho_{\text{mom}} = \sup_{\mu \in \mathcal{M}(K)_+} \int_K f_0 d\mu
\]

subject to \( \int_K f_j d\mu \leq \gamma_j, \ j = 1, \ldots, m \)

(5.1)

where \( K \) is a Borel subset of \( \mathbb{R}^n \), \( \mathcal{M}(K) \) is the space of finite signed Borel measures on \( K \), \( \mathcal{M}(K)_+ \)
is the positive cone containing finite Borel measures \( \mu \) on \( K \), \( \{\gamma_j | j = 1, \ldots, m\} \) is a set of real
numbers, and \( f_j : K \to \mathbb{R} \) are integrable with respect to all \( \mu \in \mathcal{M}(K)_+ \) for every \( j = 0, \ldots, m \).
The \( \leq \) stands for either an inequality or equality constraint.

Lasserre [71, 72, 78, 79] showed that semidefinite programming (SDP) can be used to
obtain an (often) finite sequence of relaxations which can be solved efficiently and which converge
from above to the optimal solution when the problem has polynomial data; thereby giving us a tool
with which to attempt to solve very difficult (i.e., non-convex) medium-sized problems, given the
present state of SDP solvers. The monotonic convergence means that the \( i \)th relaxation is useful,
even when optimality has not been achieved, by providing the user with an upper (i.e. optimistic)
bound \( \rho_i \) to \( \rho_{\text{mom}} \) in (5.1).

In the sequel, we will go over the preliminaries of the moments approach, present the
aforementioned Lasserre relaxations, and demonstrate how to use them to detect anomalous data
samples.
5.2 Preliminaries

For a real symmetric matrix $A$, $A \geq 0$ means $A$ is positive semidefinite. Let $d \in \mathbb{N} \triangleq \{0, 1, 2, \ldots\}$, $s(d) = \binom{n+d}{d}$, and let $\nu \in \mathbb{R}^{s(d)}$ be the column vector containing the canonical polynomial basis monomials:

$$\nu(x) = \begin{pmatrix} 1, x_1, x_2, \ldots, x_n, x_1^2, x_1x_2, \ldots, x_1x_n, x_2x_3, \ldots, x_2^2, \ldots, x^n_1, \ldots, x^n_n \end{pmatrix}^T \quad (5.2)$$

for polynomials of the form $p(x) : \mathbb{R}^n \to \mathbb{R}$ with dimension $s(d)$ and degree at most $d$. $\mathbb{R}[x]$ denotes the set of polynomials in $x \in \mathbb{R}^n = (x_1, x_2, \ldots, x_n)$ with real coefficients.

With $\alpha \in \mathbb{N}^n = (\alpha_1, \ldots, \alpha_n)$, let $x^\alpha = (x_1^{\alpha_1}, \ldots, x_n^{\alpha_n})$ and $y = \{y_\alpha\}$ denote a finite sequence of real variables. For a polynomial $p = \sum \alpha p_\alpha x^\alpha$, with coefficients $p_\alpha$, let $L_y(p)$ denote the linear map:

$$L_y(p) = \sum \alpha p_\alpha y_\alpha \quad (5.3)$$

which associates $p$ with the moment variables, $y_\alpha$.

5.2.1 Moment and Localizing Matrices

The $d$-th order moment matrix $M_d(y)$ is indexed by up-to order-$d$ monomial coefficients $\alpha, \beta \in \mathbb{N}^n$. That is, each entry of the $s(d) \times s(d)$ matrix is given by:

$$M_d(y)(\alpha, \beta) = y_{\alpha+\beta}, \; \alpha, \beta \in \mathbb{N}^n, \; |\alpha|, |\beta| \leq d \quad (5.4)$$

where $|\alpha| = \sum_{i=1}^n \alpha_i$. For example, in the case $d = 2$ and $n = 2$, if $\alpha = (0, 1)$ and $\beta = (2, 0)$, then $M_2(y)(\alpha, \beta) = y_{21}$. Usually the $(\alpha, \beta)$ is suppressed (since it is understood that the first row and column of $M_d(y)$ contains the elements of $L_y(v)$). The entire $M_2(y)$ matrix is given below for
CHAPTER 5. ROBUST ANOMALY DETECTION USING SEMIDEFINITE PROGRAMMING

Illustration:

\[
M(y) = \begin{bmatrix}
y_{00} & y_{10} & y_{01} & y_{20} & y_{11} & y_{02} \\
y_{10} & y_{20} & y_{11} & y_{30} & y_{21} & y_{12} \\
y_{01} & y_{11} & y_{02} & y_{21} & y_{12} & y_{03} \\
y_{20} & y_{30} & y_{21} & y_{40} & y_{31} & y_{22} \\
y_{11} & y_{21} & y_{12} & y_{31} & y_{22} & y_{13} \\
y_{02} & y_{12} & y_{03} & y_{22} & y_{13} & y_{04}
\end{bmatrix}
\]

(5.5)

Thus, it is clear that \( M_d(y) = L_y(\nu(x)\nu(x)^T) \).

The classical problem of moments is concerned with determining if, for a given moment sequence \( \{y_{\alpha}\} \), there is a measure \( \mu \) so that \( y_{\alpha} = \int x^\alpha d\mu \) for each \( \alpha \). If so, we say \( y \) has a representing measure which is called determinate if it is unique. Loosely speaking, checking existence of such a measure involves testing if \( M_d(y) \) is positive semidefinite. Readers interested in the technical details are referred to [72] and references therein.

For a given polynomial \( g \in \mathbb{R}[x] \), the localizing matrix \( M_d(gy) \) is given by:

\[
M_d(gy) = L_y(g(x)\nu(x)\nu(x)^T)
\]

(5.6)

This is equivalent to shifting \( M_d(y) \) by the monomials of \( g \). For example, with \( g(x) = a - x_1^2 - x_2^2 \), we have:

\[
M_2(gy) = a \begin{bmatrix}
y_{00} & y_{10} & y_{01} & y_{20} & y_{11} & y_{02} \\
y_{10} & y_{20} & y_{11} & y_{30} & y_{21} & y_{12} \\
y_{01} & y_{11} & y_{02} & y_{21} & y_{12} & y_{03} \\
y_{20} & y_{30} & y_{21} & y_{40} & y_{31} & y_{22} \\
y_{11} & y_{21} & y_{12} & y_{31} & y_{22} & y_{13} \\
y_{02} & y_{12} & y_{03} & y_{22} & y_{13} & y_{04}
\end{bmatrix}
- \begin{bmatrix}
y_{20} & y_{30} & y_{21} & y_{40} & y_{31} & y_{22} \\
y_{30} & y_{40} & y_{31} & y_{50} & y_{41} & y_{32} \\
y_{01} & y_{11} & y_{02} & y_{21} & y_{12} & y_{03} \\
y_{20} & y_{30} & y_{21} & y_{40} & y_{31} & y_{22} \\
y_{11} & y_{21} & y_{12} & y_{31} & y_{22} & y_{13} \\
y_{02} & y_{12} & y_{03} & y_{22} & y_{13} & y_{04}
\end{bmatrix}
- \begin{bmatrix}
y_{11} & y_{21} & y_{12} & y_{31} & y_{22} & y_{13} \\
y_{22} & y_{32} & y_{23} & y_{42} & y_{33} & y_{24} \\
y_{12} & y_{22} & y_{13} & y_{32} & y_{23} & y_{14} \\
y_{22} & y_{32} & y_{23} & y_{42} & y_{33} & y_{24} \\
y_{13} & y_{23} & y_{14} & y_{33} & y_{24} & y_{15} \\
y_{04} & y_{14} & y_{05} & y_{24} & y_{15} & y_{06}
\end{bmatrix}
\]

(5.7)
Localizing matrices are used to specify support constraints on $\mu$, as described in the next section.

### 5.2.2 Upper-bounds Over Semi-algebraic Sets

The material in this subsection comes from [72] and [80] which discuss the problem of finding bounds on the probability that some random variable $x$ belongs to a set $S \subseteq \mathbb{R}^n$, given some of its moments $\gamma_\alpha$.

Suppose $S$ is of the form:

$$S = \{x \in \mathbb{R}^n | f_j(x) \geq 0, \ j = 1, \ldots, m\} \quad (5.8)$$

where $f_j$ are given polynomial constraints. In terms of the GMP, this problem can be expressed as:

$$\rho_{\text{mom}} = \sup_{\mu \in \mathcal{M}(\mathbb{R})_+} \int_{\mathbb{R}^n} 1_S d\mu$$

$$\text{s.t. } \int_{\mathbb{R}^n} x^\alpha d\mu = \gamma_\alpha, \ j = 1, \ldots, m$$

where $1_S$ is the indicator function of the set $S$. To solve this problem using SDP methods, Lasserre provided the following relaxations:

$$\mathbb{P}_S \mapsto \left\{ \begin{array}{l}
\rho_i = \sup_{y_0, z} y_0 \\
\text{s.t. } y_\alpha + z_\alpha = \gamma_\alpha, \ \alpha \in \Gamma \\
M_i(y) \geq 0 \\
M_i(z) \geq 0 \\
M_{i-v_j}(f_j y) \geq 0, \ j = 1, \ldots, m
\end{array} \right\} \quad (5.10)$$

where $v_j = \lceil \deg(f_j)/2 \rceil$ and $\Gamma$ is the set of indices that correspond to the known moment information.

The following result assures us that $\rho_i$ approaches the optimal value $\rho_{\text{mom}}$ from above and tells us when optimality has been reached. This result is included for completeness as we will generally not require an optimal solution. This is an advantage because we do not have to worry about extracting optimizers from a moment matrix that has satisfied the rank condition in Theorem

\footnote{There is (currently) no reliable method of extracting optimizers from a moment matrix.}
CHAPTER 5. ROBUST ANOMALY DETECTION USING SEMIDEFINITE PROGRAMMING

Theorem 1. [72] Let $\rho_i$ be the optimal value of the semidefinite program $\mathbb{P}_S$. Then:

(a) For every $i > v$, $\rho_i \geq \rho_{mom}$ and moreover, $\rho_i \downarrow \rho_{mom}$ as $i \to \infty$.

(b) If $\rho_i$ is attained at an optimal solution $(y, z)$ which satisfies:

$$
\left\{
\begin{array}{l}
\text{Rank } M_i(y) = \text{Rank } M_{i-v}(y) \\
\text{Rank } M_i(z) = \text{Rank } M_{i-1}(z)
\end{array}
\right.
$$

(5.11)

then $\rho_i = \rho_{mom}$.

5.2.3 Lower Bounds

To compute lower bounds, one can use the complement of $S$, instead of $S$, in $\mathbb{P}_S$ and subtract $\rho_i$ from 1 [72]. However, in the approach presented here, the lower bounds do not provide much information because the sets $S$ that we will work with are small compared to the range of the data, which means the lower bound will usually be zero.

5.3 Robust Anomaly Detection

One approach to anomaly detection is to estimate the probability density function (PDF) and select a threshold below which incoming samples are classified as anomalous. Another is to train a 1-class support vector machine (SVM) and use it to classify samples. Here, we will compare against both of these standard techniques. To quantify performance, we will use the threshold-independent receiver operating characteristic (ROC) curves and area under curve (AUC) metric; this is a standard way of assessing binary classifier performance. Both ROC and AUC are readily obtained using the MATLAB command `perfcurve`. The kernel density estimates are obtained using the MATLAB command `ksdensity` (and the KDE toolbox for higher dimensional examples [81]). For the 1-class SVM, we use the MATLAB command `fitcsvm` with the automatic tuning- and “standardized data” options enabled.

We would like to emphasize to the reader, however, that our goal is not to estimate the probability density function. Our goal is simpler: to use the upper-bound on the probability of the
observation to decide whether or not it is anomalous. To use $P_S$ for anomaly detection we select $r > 0$ to specify a neighborhood $S$ which contains an incoming measurement. The neighborhood can be a sphere or a box and is specified in the constraints $f_j$ in $P_S$. The $r$ can be selected as a function of the measurement noise, or otherwise small compared to the expected range of the data. In fact, $r$ can be set to zero if one wishes. In practice, we would like to use $r$ to account for measurement noise.

5.3.1 Moment Estimation and Data Whitening

A key advantage of the proposed method is that the information from a data set enters through the moment estimates which are computed using Equation (5.12).

$$\gamma_\alpha \approx \frac{1}{N} \sum_{i=1}^{N} x_i^\alpha, \; \alpha \in \Gamma$$  \hspace{1cm} (5.12)

It is known that it is difficult to estimate higher order moments since whatever noise is present in the data will be raised to higher powers as well. In view of Equation (5.12), one can see that if the data is poorly scaled or biased in some coordinate, $\gamma_\alpha$ will increase quickly in those directions and will therefore result in fewer available moments for those directions and may cause numerical problems for the SDP solver. Thus, it is sometimes useful to use the data whitening technique discussed in [82] to transform the data so that it has zero mean and unit variance. This may be useful in lower-dimensional instances, where it is possible to use these higher order moments without the size of the moment matrices exceeding computational resources.

Data whitening is a simple procedure that involves subtracting the mean of the data and multiplying by a transformation matrix, in the multivariate case, or dividing by the standard deviation in the univariate case. Incoming samples are then processed the same way, using the stored mean and transformation, before obtaining an upper-bound probability estimate.

5.4 Examples

In this section we present four examples. The first is a 1D example that shows that our approach compares favorably with traditional kernel probability density based anomaly detectors and with the 1-class SVM. We will demonstrate: that higher-order moments contain useful information

---

2 $S$ in $P_S$ can be any set of the form in (5.8).

3 We used MATLAB R2013b/R2014a, CVX [83], and SeDuMi [77] to solve our examples.
and that this information helps best the other two approaches especially when there are fewer data points to learn from and when the dimensionality of the data increases. These points will become apparent through the first three examples, which increase in dimension. The final example shows how our method can be used to recover contours from a noisy binary image.

5.4.1 Example 1: A Bimodal Distribution

Consider the bimodal distribution \(^8\) in Figure 5.1. This distribution poses a challenge to density estimation tools like Parzen windows because the two modes are estimated best using different window sizes.

Figure 5.1 shows the distribution and the test points, which consist of 300 inliers and 50 outliers.

![Figure 5.1: Bimodal distribution](image)
Figure 5.2: ROC curves when 50 points are used for training

Figure 5.3: ROC curves when 300 points are used for training

Table 5.1 shows the AUC values for our experiments, which use $r = 0.001$ for the moments classifier. The reader can see that AUC generally increases as more moments are used and that the performance is better with fewer points. The difference, however, is small because this is a 1D
CHAPTER 5. ROBUST ANOMALY DETECTION USING SEMIDEFINITE PROGRAMMING

example.

Table 5.1: AUC Summary

<table>
<thead>
<tr>
<th>N</th>
<th>SVM</th>
<th>Parzen</th>
<th>$M_2$</th>
<th>$M_4$</th>
<th>$M_6$</th>
<th>$M_8$</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>0.9821</td>
<td>0.9686</td>
<td>0.8701</td>
<td>0.9715</td>
<td><strong>0.9940</strong></td>
<td>0.9929</td>
</tr>
<tr>
<td>100</td>
<td>0.9907</td>
<td>0.9807</td>
<td>0.8457</td>
<td>0.9882</td>
<td><strong>0.9984</strong></td>
<td>0.9957</td>
</tr>
<tr>
<td>300</td>
<td><strong>0.9953</strong></td>
<td>0.9781</td>
<td>0.8268</td>
<td>0.9783</td>
<td><strong>0.9951</strong></td>
<td>0.9945</td>
</tr>
</tbody>
</table>

Table 5.2 shows the actual number of correct classifications obtained using the threshold corresponding to the optimal operating point on the ROC curve.

Table 5.2: Classification performance ($N = 300$)

<table>
<thead>
<tr>
<th>Classifier</th>
<th>Samples</th>
<th>Inliers</th>
<th>Outliers</th>
</tr>
</thead>
<tbody>
<tr>
<td>SVM</td>
<td>345/350</td>
<td>300/300</td>
<td>45/50</td>
</tr>
<tr>
<td>Parzen</td>
<td>330/350</td>
<td>299/300</td>
<td>31/50</td>
</tr>
<tr>
<td>$M_4$</td>
<td>333/350</td>
<td>300/300</td>
<td>33/50</td>
</tr>
<tr>
<td>$M_6$</td>
<td><strong>344/350</strong></td>
<td><strong>299/300</strong></td>
<td><strong>45/50</strong></td>
</tr>
<tr>
<td>$M_8$</td>
<td>344/350</td>
<td>298/300</td>
<td>46/50</td>
</tr>
</tbody>
</table>

5.4.2 Example 2: 2D Distribution

For higher dimensional data sets, the complexity of estimating PDFs increases greatly. Indeed, the arguably most popular engineering software tool, MATLAB, does not have a density estimation function for multi-dimensional data sets.

Consider the distribution below:

\[
\begin{align*}
  x_1 &= 2 \cos(1.5t) + \eta_1 \\
  x_2 &= 4 \operatorname{sinc}(0.9t) + \eta_2
\end{align*}
\]  

with $t \in [\pi/4, \pi]$, $\eta_1 \sim \mathcal{U}[-0.05, 0.05]$, and $\eta_2 \sim \mathcal{U}[0, 0.02]$. This distribution has a “P” shape and the closed portion presents a problem for the kernel and SVM approaches.

Figure 5.4 shows the probability surface and the 350 test points which include 50 outliers.
CHAPTER 5. ROBUST ANOMALY DETECTION USING SEMIDEFINITE PROGRAMMING

Figure 5.4: 2D distribution and test points

Figure 5.5: ROC curves when 100 points are used for training
CHAPTER 5. ROBUST ANOMALY DETECTION USING SEMIDEFINITE PROGRAMMING

Figure 5.6: ROC curves when 300 points are used for training

Table 5.3 shows the results of our experiments, using $r = 0.001$ for the moments classifier. Again, the reader can see how the performance increases as more moment information is included except this time the difference in performance is greater because the data has another dimension. This trend will continue in the third example.

Table 5.3: AUC Summary

<table>
<thead>
<tr>
<th>$N$</th>
<th>SVM</th>
<th>Parzen</th>
<th>$M_2$</th>
<th>$M_3$</th>
<th>$M_4$</th>
<th>$M_5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>0.9252</td>
<td>0.7843</td>
<td>0.6330</td>
<td>0.9722</td>
<td>0.9883</td>
<td>0.9916</td>
</tr>
<tr>
<td>150</td>
<td>0.9437</td>
<td>0.8045</td>
<td>0.6575</td>
<td>0.9751</td>
<td>0.9903</td>
<td>0.9947</td>
</tr>
<tr>
<td>300</td>
<td>0.9435</td>
<td>0.8335</td>
<td>0.6436</td>
<td>0.9705</td>
<td>0.9863</td>
<td>0.9933</td>
</tr>
</tbody>
</table>

Table 5.4 shows the actual number of correct classifications obtained using the threshold corresponding to the optimal operating point on the ROC curve.

Table 5.4: Classification performance ($N = 300$)

<table>
<thead>
<tr>
<th>Classifier</th>
<th>Samples Classified Correctly</th>
<th>Inliers Classified Correctly</th>
<th>Outliers Classified Correctly</th>
</tr>
</thead>
<tbody>
<tr>
<td>SVM</td>
<td>328/350</td>
<td>287/300</td>
<td>41/50</td>
</tr>
<tr>
<td>Parzen</td>
<td>310/350</td>
<td>291/300</td>
<td>19/50</td>
</tr>
<tr>
<td>$M_3$</td>
<td>336/350</td>
<td>290/300</td>
<td>46/50</td>
</tr>
<tr>
<td>$M_4$</td>
<td>340/350</td>
<td>297/300</td>
<td>43/50</td>
</tr>
<tr>
<td>$M_5$</td>
<td><strong>343/350</strong></td>
<td><strong>296/300</strong></td>
<td><strong>47/50</strong></td>
</tr>
</tbody>
</table>
CHAPTER 5. ROBUST ANOMALY DETECTION USING SEMIDEFINITE PROGRAMMING

5.4.3 Example 3: Swiss roll Distribution

The following distribution is the well-known “Swiss roll” distribution. The following equations were used to create the test and training samples:

\[
\begin{align*}
    x_1 &= y_1 \cos(y_1) + \eta_1 \\
    x_2 &= y_1 \sin(y_1) + \eta_2 \\
    x_3 &= y_2 + \eta_3
\end{align*}
\]  

with \( y_1 \sim \mathcal{U}[\frac{3\pi}{2}, \frac{9\pi}{2}] \), \( y_2 \sim \mathcal{U}[1, 2] \), \( \eta_1 \sim \mathcal{N}(0, 0.8^2) \), \( \eta_2 \sim \mathcal{N}(0, 0.3^2) \), and \( \eta_3 \sim \mathcal{N}(0, 0.25^2) \). Figure 5.7 shows the test points which consist of 300 inliers and 50 outliers.
Table 5.5 shows the familiar trend of increasing AUC with additional moments and excellent performance even with just 100 training samples. This example illustrates how the other two standard techniques degrade as dimensionality increases. As with the previous examples,
CHAPTER 5. ROBUST ANOMALY DETECTION USING SEMIDEFINITE PROGRAMMING

$r = 0.001$ was used for the moments classifier and the recommended automatic tuning options were used for other techniques.

Table 5.5: AUC Summary

<table>
<thead>
<tr>
<th>N</th>
<th>SVM</th>
<th>Parzen</th>
<th>$M_2$</th>
<th>$M_4$</th>
<th>$M_6$</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>0.7887</td>
<td>0.8487</td>
<td>0.8099</td>
<td>0.9194</td>
<td>0.9228</td>
</tr>
<tr>
<td>300</td>
<td>0.8001</td>
<td>0.9085</td>
<td>0.8201</td>
<td>0.9480</td>
<td><strong>0.9903</strong></td>
</tr>
</tbody>
</table>

Table 5.6 shows the actual number of correct classifications obtained using the threshold corresponding to the optimal operating point on the ROC curve.

Table 5.6: Classification performance ($N = 300$)

<table>
<thead>
<tr>
<th>Classifier</th>
<th>Samples Classified Correctly</th>
<th>Inliers Classified Correctly</th>
<th>Outliers Classified Correctly</th>
</tr>
</thead>
<tbody>
<tr>
<td>SVM</td>
<td>322/350</td>
<td>290/300</td>
<td>32/50</td>
</tr>
<tr>
<td>Parzen</td>
<td>326/350</td>
<td>296/300</td>
<td>30/50</td>
</tr>
<tr>
<td>$M_2$</td>
<td>325/350</td>
<td>292/300</td>
<td>33/50</td>
</tr>
<tr>
<td>$M_4$</td>
<td>335/350</td>
<td>299/300</td>
<td>36/50</td>
</tr>
<tr>
<td>$M_6$</td>
<td><strong>345/350</strong></td>
<td><strong>298/300</strong></td>
<td><strong>47/50</strong></td>
</tr>
</tbody>
</table>

5.4.4 AUC Sensitivity to $r$

One of the advantages of our approach is that it does not require much tuning. The importance of this property increases with the dimensionality of the dataset. For example, tuning the kernel width for Parzen estimates presently involves using rules-of-thumb [81]. For our experiments we found that AUC versus $r$ curves were smooth and in general AUC decreased with increasing $r$ as one would expect. Thus, for well-behaved distributions detection performance does not appear to be overly dependent on $r$, as can be seen in Figure 5.10.
5.4.5 Example 4: Contour Detection

The following example is included to show the potential of this method to be used in image processing applications. The contour image is from [85, 86]. In this example, we attempt to recover a contour corrupted by 5% noise. We calculated the Haralick energy, in two directions and three scales, for (9x9) neighborhoods of each pixel of the clean image, computed $\gamma_\alpha$, and the upper-bound probability surface, $M_4$, using the noisy image with $r = 0.05$. The three resulting probability images were then averaged. Since there are many more white pixels, the contour pixels should have a lower probability upper-bound. Further, as the Haralick energy is the sum of the squared elements of the co-occurrence matrix, computed for each 9x9 window, we also expect the noisy pixels to have higher probability than the contour pixels. The reader can see that this is the case, as shown in the left subfigure of Figure 5.12. After inverting, thresholding, and post-processing with morphological operations, we obtained the contour shown on the right of Figure 5.12, again, using just the information contained in $\gamma_\alpha$. 
5.4.6 Example 5: Damage Detection in a Bearing Test Rig

The following example uses data from publicly available data set [1, 87]. The data comes in the form of accelerometer readings from four bearings installed on a shaft coupled to an AC motor via rub belts. Figure 5.13 shows the test rig. The motor was run at 2000 RPM with a radial load of 6000 lbs, applied onto the shaft and bearings by a spring mechanism, until failure. See [1] for more details. The first test considered has 984 files containing 1-second accelerometer snapshots at 20 kHz, taken every 10 minutes over 1 week. At the end of the week, an “outer race failure” occurred in bearing 1.

4We consider Test #2 first and then Test #1.
CHAPTER 5. ROBUST ANOMALY DETECTION USING SEMIDEFINITE PROGRAMMING

Prior to applying our method, we downsampled the data to 1 kHz (which is well above the frequencies of interest [1]) and used a simple low-pass filter to remove components above 500 Hz. Next, we computed features frequently extracted from vibration data from rotating machinery – the mean, RMS, kurtosis, and skewness – from the first 250 snapshots. Thus, we extracted a 4-feature vector for each of the first 250 snapshots before computing the moments $\gamma_\alpha$. (Figure 5.14 shows the features extracted for bearing 1.) We then used the moment information to compute the $M_2$ upper-bounds (using $r = 0.01$) for the remaining snapshot feature vectors.

Figure 5.15 shows the upper-bound probabilities for the four accelerometer channels. Figure 5.16 shows the 8-point running average of the upper-bound probabilities. The first channel...
shows that bearing 1 started to degrade at about snapshot 651 and crossed the 2% probability threshold at snapshot 709 (2.3 and 1.9 days before the failure, respectively).

**Figure 5.14**: Bearing 1 mean, RMS, kurtosis, and skewness

**Figure 5.15**: Upper-bound probabilities of the four channels
Figures 5.17-5.19 show the features and the running-average of the upper-bound probabilities for the second test (Test #1 in the data set) in which bearings 3 and 4 experienced failures. The change in bearing 4 became apparent at about snapshot 1451 and crossed the 2% probability threshold at snapshot 1621 (6.44 and 4.96 days before the failure, respectively).
CHAPTER 5. ROBUST ANOMALY DETECTION USING SEMIDEFINITE PROGRAMMING

Figure 5.17: Bearing 3 mean, RMS, kurtosis, and skewness

Figure 5.18: Bearing 4 mean, RMS, kurtosis, and skewness
This data set was also referenced in [88, 89, 90, 91, 92, 93, 94, 95, 96] in the context of remaining-useful-life prediction, so it is not easy to compare against these works since they generally speak in terms of confidence intervals. The papers that do give a number, do not support their claims in their figures as clearly as we have here. Furthermore, the probability bounds obtained from (5.10) have a clear interpretation: for every process that could have produced data with moments $\gamma_\alpha$, there is at most $\rho_i$ probability of observing a particular data point.

### 5.5 Implementation

Generally speaking, current SDP solver technology limits our approach to small-to-medium sized problems. Moreover, for some applications it may not be feasible to solve an SDP program for each data sample. It is far more computationally efficient to use a look-up table (LUT) – which can be computed in parallel – at the cost increased memory requirements and some detection performance. We made LUTs using 100 point grids (in each coordinate) for classifiers in each synthetic example and found that the detection performance, measured by AUC, did not decrease much by using a LUT. The results in Table 5.7 were obtained on the first attempt using simple linear interpolation and with values outside the range of the table set to zero.
Table 5.7: Classification performance using LUTs ($N = 300$)

<table>
<thead>
<tr>
<th>Classifier</th>
<th>SDP (AUC)</th>
<th>LUT (AUC)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ex. 1, $M_6$</td>
<td>0.9951</td>
<td>0.9950</td>
</tr>
<tr>
<td>ex. 2, $M_5$</td>
<td>0.9933</td>
<td>0.9932</td>
</tr>
<tr>
<td>ex. 3, $M_6$</td>
<td>0.9903</td>
<td>0.9899</td>
</tr>
</tbody>
</table>

5.6 Conclusions

This chapter illustrates the ability of very recent developments in polynomial optimization and generalized moment problems to provide an elegant, computationally tractable, solution to anomaly detection problems. The proposed approach uses information from a set of moments of a probability distribution to directly provide an upper-bound $\rho$ on the probability of observing a particular sample (modulo measurement noise), without explicitly modeling the distribution. A salient feature of this technique is that all the data, whether obtained using 1 or 1000 observations, enter the problem through a finite sequence of moments, that can be updated as more data is collected, without affecting the computational complexity (since new observations affect the value, but not the size of the moment matrices). Furthermore, the detectors obtained by solving $\mathbb{P}_S$ reached peak performance with fewer samples than the kernel and SVM methods.

Our examples implicitly assume that the observations contain enough information to make good decisions, that is, we have assumed that the samples are good features. When anomalous observations are available, it may be possible to construct optimal features to separate classes – in the same way support (or relevance) vector machine methods find separating curves – by maximizing the distance between the moment vectors of the two classes. We think this is an interesting direction for future research.
Chapter 6

Conclusions

To conclude this dissertation, we summarize the contributions and discuss directions the author would like to explore in the future.

6.1 Summary

Our first contribution introduced a method for sparsifying a (kernel) probability density estimate. We employed an RVM and showed that, by using the same type of kernel in both the density estimate and RVM, the problem reduces to a regression problem.

The next contribution is a method for sparsifying an $H_2$-optimal linear filter. We used the Youla parametrization for filters to showed the problem is equivalent to a convex program with equality constraints. We also provided a novel modification to the traditional LP relaxation for the 0-1 knapsack problem that is shown to improve performance over the standard approach.

Next, a new method was presented for the most difficult HMM problem: the training problem. We employed a polynomial optimization approach to train discrete HMMs using low-order moments estimated from the observations. We compared against standard and new approaches and showed that our method: does not suffer from many of the drawbacks while maintaining important advantages. Moreover, the drawbacks in our approach are encapsulated in current SDP solver technology.

We followed with a new method for detecting anomalies based on recent results in polynomial optimization and the GMP. The method uses only information about the statistical moments of the normal-state distribution and compares favorably with existing approaches (such as Parzen windows and 1-class SVM) – especially when working with smaller, or higher dimensional, data sets.
Finally, we provide a MATLAB toolbox for implementing moments-based optimization solutions and code for the HMM examples.

6.2 Future work

The last three contributions all centered on the method of moments. Both the learning algorithm for HMMs and the GMP-based anomaly detector used the method of moments in their solutions. The anomaly detection algorithm takes advantage of Lasserre’s work which enables the use of a large number moments (in some cases countably-many). This was not possible in the past and we believe the ability to use information from a large number of moments has given new life to the method of moments and that there are more contributions to be made in machine learning using this tool. In particular, the author would like to use these tools to solve difficult problems in the area of autonomous vehicles. Some real-world algorithms used to localize a vehicle relative to a map use exhaustive search and could benefit from a closer look from a new perspective.
Bibliography


BIBLIOGRAPHY


BIBLIOGRAPHY


BIBLIOGRAPHY


BIBLIOGRAPHY


BIBLIOGRAPHY


BIBLIOGRAPHY


BIBLIOGRAPHY


Appendix A

NEU Moments v0.1: A MATLAB Toolbox for Moments-based Optimization

The purpose of this tutorial/toolbox is to bridge the gap between Jean Lasserre’s 2001 paper [71], on global polynomial optimization using moments, and actual moments-based problem solving using MATLAB. The guide introduces the reader to several low-level utility functions intended to facilitate the construction of moments-based programs; while the utility functions are intended to work with the CVX modeling system for convex optimization, they can directly work with other modeling languages, like YALMIP, without any modifications. The efficacy of the toolbox is demonstrated by reproducing all the examples in [71] and two additional combinatorial optimization examples.

A.1 Introduction

The problem of finding global solutions to non-convex optimization problems remains unsolved. However, when the problem is described using polynomials, Lasserre showed that the problem reduces to an (often) finite sequence of convex linear matrix inequalities (LMIs) [71]. This tutorial is concerned with taking such a problem and converting it to MATLAB code which can give us the numerical answer we’re ultimately looking for. For the theory on global optimization using the method of moments, the reader can read the references at the end of this document.
The notation used in this document is from [71] and will be reviewed in the next section. This tutorial will only make sense to readers who at least skim [71].

A.1.1 Notation and Preliminaries

The notation follows Jean Lasserre’s notation in [71]. Given a real-valued polynomial $p(x) : \mathbb{R}^n \to \mathbb{R}$, the symbol $\mathbb{P}$ denotes the unconstrained optimization problem:

$$\mathbb{P} \mapsto p^* = \min_{x \in \mathbb{R}^n} p(x)$$

and the symbol $\mathbb{P}_K$ denotes the constrained optimization problem:

$$\mathbb{P}_K \mapsto p^*_K = \min_{x \in K} p(x)$$

where $K$ is a (possibly non-convex) compact set defined by polynomial inequalities, $g_i \geq 0$, $i = 1, \ldots, r$. For example, $K = \{x \in \mathbb{R}^n | 1 - \|x\|^2 \geq 0\}$.

From Lasserre 2001:

3. Unconstrained global optimization. Let $p(x) : \mathbb{R}^n \to \mathbb{R}$ be a real-valued polynomial of degree $2m$ with coefficient vector $p \in \mathbb{R}^{e(2m)}$. Since we wish to minimize $p(x)$, we may and will assume that the constant term vanishes, that is, $p_0 = 0$. Let us introduce the following convex LMI optimization problem (or SDP program):

$$\mathbb{Q} \mapsto \left\{ \begin{array}{l}
\inf_{y} \sum_{\alpha} p_\alpha y_\alpha \\
\quad M_m(y) \geq 0
\end{array} \right. \quad (A.3)$$

or equivalently,

$$\mathbb{Q} \mapsto \left\{ \begin{array}{l}
\inf_{y} \sum_{\alpha} p_\alpha y_\alpha \\
\sum_{\alpha \neq 0} y_\alpha B_\alpha \geq -B_0
\end{array} \right. \quad (A.4)$$

where the matrices $B_i$ are easily understood from the definition of $M_m(y)$. The dual problem $\mathbb{Q}^*$ of
APPENDIX A. NEU MOMENTS V0.1

$Q$ is the convex LMI problem defined by:

\[
Q^* \mapsto \begin{cases} 
  \sup_X \langle X, -B_0 \rangle (= -X(1, 1)) \\
  \langle X, B_\alpha \rangle = p_\alpha, \ \alpha \neq 0 \\
  X \geq 0 
\end{cases} \quad (A.5)
\]

\[
Q^K_N \mapsto \begin{cases} 
  \inf_y \sum_\alpha p_\alpha y_\alpha \\
  M_N(y) \geq 0 \\
  M_N - \tilde{w}_i (g_i y) \geq 0, \ i = 1, \ldots, r 
\end{cases} \quad (A.6)
\]

\[
(Q^K_N)^* \mapsto \begin{cases} 
  \sup_{X,Z_i} -X(1, 1) - \sum_{i=1}^{r} g_i(0) Z_i(1, 1) \\
  \langle X, B_\alpha \rangle + \sum_{i=1}^{r} \langle Z_i, C_i\alpha \rangle = p_\alpha, \ \alpha \neq 0 \\
  X, Z_i \geq 0, \ i = 1, \ldots, r 
\end{cases} \quad (A.7)
\]

A.2 MATLAB Utility Functions

The utilities included with this toolbox are intended to do a few things well – in the way `grep`, `sed`, `find`, etc. provide low-level (but very useful) functionality in Linux operating systems. Backwards compatibility will not be guaranteed, in future releases, but functionality will not decrease. So, the way a user codes an operation may change, but not the ability to accomplish the same task. Functionality will not be added at the expense of flexibility. If something diminishes flexibility, it belongs at the user level – at least, that’s the philosophy taken here. Additionally, the utilities will never be cluttered with unnecessary error-checking; however, the functions will include plenty of documentation and examples.

\[[M, \text{basis\_data}] = \text{get\_mmatrix}(\text{degree, dimension})\] accepts two natural numbers \text{degree} and \text{dimension} and returns the matrix $M$ with moments up to order \text{degree}
for polynomials in dimension variables. Each moment is stored as a vector of length dimension. This function also returns a structure \texttt{basis_data} which has the fields \texttt{basis_data.basis} and \texttt{basis_data.monomial}, an \((\text{dimension} + \text{degree}) \times \text{dimension}\) matrix with the monomials that represent the basis for degree order polynomials and a cell array \texttt{basis_data.monomial} of length \(k + 1\) with the \(k\)-degree monomials in element \(k + 1\).

The monomial vectors in \(M\) and \texttt{basis_data.basis} are stored as \texttt{uint32} vectors, to save memory, and \(M\) is stored as a \((\text{dimension} + \text{degree}) \times \text{dimension} \times (\text{dimension} + \text{degree})\)

\[(A.9)\]

matrix.

\[\text{[monomials, n_monomials]} = \text{get_monomials(matrix)}\]
accepts a moment matrix, or a cell array of moment matrices, and returns a minimal list of monomials that exist in all the matrices and the length of that list. This function is used by the user to declare an array of optimization variables and a map to identify each optimization variable with a monomial vector.

\[\text{out} = \text{get_monomial_basis(degree, dimension)}\]
accepts two natural numbers \texttt{degree} and \texttt{dimension} and returns a structure \texttt{basis_data} which has the fields \texttt{basis_data.basis}, an \((\text{dimension} + \text{degree}) \times \text{dimension}\) matrix with the monomials that represent the basis for degree order polynomials and a cell array \texttt{basis_data.monomial} of length \(k + 1\) with the \(k\)-degree monomials in element \(k + 1\).

\[\text{map} = \text{get_map(matrix, monomials)}\]
accepts a moment matrix and a list of monomials of length \texttt{n_monomials} and returns an \texttt{n_monomials x n_monomials} matrix that associates each monomial vector in the moment matrix with an index into the array of optimization variables used by whatever modeling language the user is using.

\[\text{index} = \text{get_index(monomials, monomial)}\]
accepts a list of monomials and a single
monomial, or a column of monomials, and returns the index into the list \texttt{monomials} where
the monomial can be found. If \texttt{monomial} is not in the list, \texttt{index} will be 0.

\begin{verbatim}
B = get_index_matrices(monomials,map) accepts a list of monomials and a map ma-
trix and returns a cell array of binary “index” matrices, which Lasserre calls \( B_\alpha \) or \( C_\alpha \), which identify each monomial with a location in the moment matrix map. Since these matrices are sparse, they are stored as \texttt{sparse} matrices in MATLAB.
\end{verbatim}

\begin{verbatim}
matrix = assignm(y,map) accepts an array of optimization variables \texttt{y} and a map matrix and returns a matrix \texttt{matrix} of the same size as map with the appropriate optimization variable in the corresponding moment location. This is matrix that the optimization environment will operate on.
\end{verbatim}

\begin{verbatim}
L = localize(M,monomial) accepts a moment matrix \texttt{M} and a single monomial vector and returns the matrix \texttt{L}, of the same size as \texttt{M} with each element containing the sum of the corresponding element in \texttt{M} and \texttt{monomial}. This function is used to create localizing matrices; which Lasserre denotes by \( M(qy) \).
\end{verbatim}

\begin{verbatim}
M = make_zero_one(M) accepts a moment matrix \texttt{M} and replaces all the powers greater than 1 by 1. This is done to set-up a moment matrix for 0-1 programs, thereby eliminating the need to include constraints of the form \( x_i^2 = x_i \) [97].
\end{verbatim}

\begin{verbatim}
moment_tools_verification is a script is used to verify that the toolbox is working as expected – it calls each example included in this toolbox. The expected answer is listed in the script for comparison.
\end{verbatim}

A.3 Examples

A.3.1 Example 1

\begin{verbatim}
Lasserre 2001, pg. 803, Example 1 Consider the polynomial \( p(x) : \mathbb{R}^2 \to \mathbb{R} \)
\end{verbatim}

\begin{verbatim}
(\( x_1, x_2 \)) \mapsto (x_1^2 + 1)^2 + (x_2^2 + 1)^2 + (x_1 + x_2 + 1)^2 \quad (A.11)
\end{verbatim}

Expanding \( p \) gives:

\begin{verbatim}
p(x) = x_1^4 + x_2^4 + 3x_1^2 + 3x_2^2 + 2x_1x_2 + 2x_1 + 2x_2 + 3 \quad (A.12)
\end{verbatim}
APPENDIX A. NEU MOMENTS V0.1

from here we can see that, in moment notation, we have:

$$
\sum_{\alpha} \alpha p_{\alpha} y_{\alpha} = y_{40} + y_{04} + 3y_{20} + 3y_{02} + 2y_{11} + 2y_{10} + 2y_{01}
$$

(A.13)

We can solve $Q^*$ using the code:

```matlab
% AUTHOR: Jose Lopez NEU 2014
% LICENSE: This code is licensed under the BSD License.
%
clear;clc;
cvx_clear;
%%
% Lasserre example 1 dual formulation
cvx_begin sdp
%cvx_solver sedumi;

M = get_mmatrix(2,2); % construct moment matrix
[monomials, ~] = get_monomials(M); % get all moments in mmatrix
map = get_map(M,monomials);
B = get_index_matrices(monomials,map); % get index matrices

target X(6,6) symmetric;
dual variable z01;
dual variable z10;
maximize(-X(1,1))

subject to
trace(X*B{2})==2 : z01; % p01
trace(X*B{4})==3; % p02
trace(X*B{7})==0; % p03
trace(X*B{11})==1; % p04
trace(X*B{3})==2 : z10; % p10
trace(X*B{5})==2; % p11
trace(X*B{8})==0; % p12
trace(X*B{12})==0; % p13
trace(X*B{6})==3; % p20
trace(X*B{9})==0; % p21
trace(X*B{13})==0; % p22
```


APPENDIX A. NEU MOMENTS V0.1

\[
\begin{bmatrix}
0.493 & 1 & 1 & -0.0581 & -0.0017 & -0.0581 \\
1 & 3.12 & 1 & 0 & 0.197 & -0.197 \\
1 & 1 & 3.12 & -0.197 & 0.197 & 0 \\
-0.0581 & 0 & -0.197 & 1 & 0 & -0.826 \\
-0.0017 & 0.197 & 0.197 & 0 & 1.65 & 0 \\
-0.0581 & -0.197 & 0 & -0.826 & 0 & 1
\end{bmatrix}
\]  
(A.14)

The difference between the results here and in Lasserre’s paper can be attributed to differences in solvers.

We can solve \( Q \) instead using the code below:

% AUTHOR: Jose Lopez NEU 2014
% LICENSE: This code is licensed under the BSD License.
% this script implements example 1 in Lasserre’s 2001 paper
clear;clc;
% it’s a good idea to do all the non CVX stuff outside of the CVX
% environment because for larger problems, especially ones that use the
% reweighted heuristic, this can save a lot of time.

[M,basis_data] = get_mmatrix(2,2); % construct moment matrix
[monomials,n_monomials] = get_monomials(M); % get all moments in mmatrix
maps.M2 = get_map(M,monomials); % map to CVX variable
indices.y40 = get_index(monomials,[4,0]);
indices.y04 = get_index(monomials,[0,4]);
APPENDIX A. NEU MOMENTS V0.1

indices.y20 = get_index(monomials,[2,0]);
indices.y02 = get_index(monomials,[0,2]);
indices.y11 = get_index(monomials,[1,1]);
indices.y10 = get_index(monomials,[1,0]);
indices.y01 = get_index(monomials,[0,1]);

%%% cvx_clear;
cvx_begin sdp
  cvx_solver sedumi;

  variable y(n_monomials); %declare CVX variables

  M2 = assignm(y,maps.M2); %declare mmatrix using CVX variables

  %get interpretable aliases to form objective
  y00 = y(1); %y00 is always y(1)
  y40 = y(indices.y40);
  y04 = y(indices.y04);
  y20 = y(indices.y20);
  y02 = y(indices.y02);
  y11 = y(indices.y11);
  y10 = y(indices.y10);
  y01 = y(indices.y01);

minimize(y40+y04+3*y20+3*y02+2*y11+2*y10+2*y01)
subject to
  M2>=0
  y00 == 1;
  cvx_end

%cvx_optval = -0.492635 matches paper
%x* = (-0.2428,-0.2428)
% M2 is rank-1 so we can read off optimizers
[y10,y01]

Running this code (also) gives cvx_optval = -0.492635 but we also get \( x_1 = y_{10} = -0.24286 \) and \( x_2 = y_{01} = -0.24286 \); which agrees with the paper.

Using YALMIP,
APPENDIX A. NEU MOMENTS V0.1

% AUTHOR: Jose Lopez NEU 2014
% LICENSE: This code is licensed under the BSD License.
%
% this script implements Lasserre's example 1 using yalmip
%
clear;clc;
yalmip('clear');

%M, basis_data] = get_mmatrix(2,2); %construct moment matrix
[monomials, n_monomials] = get_monomials(M); %get all moments in mmatrix
map = get_map(M, monomials); %get map to yalmip variables

y = sdpvar(n_monomials,1); %declare variables
M2 = assignm(y,map); %declare mmatrix using variables

%get interpretable aliases to form objective
y00 = y(1); %y00 is always y(1)
y40 = y(get_index(monomials,[4,0]));
y04 = y(get_index(monomials,[0,4]));
y20 = y(get_index(monomials,[2,0]));
y02 = y(get_index(monomials,[0,2]));
y11 = y(get_index(monomials,[1,1]));
y10 = y(get_index(monomials,[1,0]));
y01 = y(get_index(monomials,[0,1]));

objective = y40+y04+3*y20+3*y02+2*y11+2*y10+2*y01;
constraints = [M2>=0,y00==1];
%options = sdpsettings('solver','sdpa');
solvesdp(constraints,objective,[]);
double(objective) %-0.49263
%M2 is rank-1 so we can read off optimizers
double([y10,y01]) %-0.24284 -0.24284

A.3.2 Example 2

Lasserre 2001, pg. 804, Example 2 Consider the polynomial \( p(x) : \mathbb{R}^2 \to \mathbb{R} \)

\[
(x_1, x_2) \mapsto (x_1^2 + 1)^2 + (x_2^2 + 1)^2 - 2(x_1 + x_2 + 1)^2
\]  
(A.15)
APPENDIX A. NEU MOMENTS V0.1

Expanding $p$ gives:

$$p(x) = x_1^4 + x_2^4 - 4x_1x_2 - 4x_1 - 4x_2$$  \hspace{1cm} (A.16)

from here we can see that, in moment notation, we have:

$$\sum_{\alpha} p_\alpha y_\alpha = y_{40} - 4y_{11} - 4y_{10} + y_{04} - 4y_{01}$$  \hspace{1cm} (A.17)

Solving $\square$ using the code below:

```
% AUTHOR: Jose Lopez NEU 2014
% LICENSE:This code is licensed under the BSD License.
% this script implements example 2 in Lasserre's 2001 paper
clear;clc;

[M,basis_data] = get_mmatrix(2,2);
[monomials,n_monomials] = get_monomials(M);
maps.M2 = get_map(M,monomials);

cvx_clear;
cvx_begin sdp
cvx_solver sedumi;

cvx_solver sedumi;

variable y(n_monomials);
M2 = assignm(y,maps.M2);

y00 = y(1);
y40 = y(get_index(monomials,[4,0]));
y04 = y(get_index(monomials,[0,4]));
y11 = y(get_index(monomials,[1,1]));
y10 = y(get_index(monomials,[1,0]));
y01 = y(get_index(monomials,[0,1]));

minimize(y40 - 4*y11 - 4*y10 + y04 - 4*y01)
subject to
M2>=0
```
y00 == 1;
cvx_end
%(cvx_optval): -11.4581 matches
%x*=(1.3247,1.3247), M2 is rank-1 so we can just read off optimizers
[y10,y01]
yields a minimum value of $-11.458$ with $(x_1^*, x_2^*) = (1.3247, 1.3247)$.

To solve $Q^*$ instead, we run:

```matlab
% AUTHOR: Jose Lopez NEU 2014
% LICENSE: This code is licensed under the BSD License.
% This script implements Lasserre example 2, dual formulation. I made this
% before I knew how to get the dual from the primal solution. (See CVX dual
% variables and joint+ marginal example.)

clear; clc;

%M = get_mmatrix(2,2); % construct moment matrix
[monomials, ~] = get_monomials(M); % get all moments in mmatrix
B = get_index_matrices(monomials, get_map(M, monomials)); % get index matrices

% cvx_clear;
cvx_begin sdp
cvx_solver sedumi;
variable X(6,6) symmetric;
dual variables z10 z01;

maximize( -X(1,1) )

subject to
trace(X*B{2})==4 : z01; %p01
trace(X*B{4})==0; %p02
trace(X*B{7})==0; %p03
trace(X*B{11})==1; %p04
trace(X*B{3})==4 : z10; %p10
```
APPENDIX A. NEU MOMENTS V0.1

trace(X*B{5})==-4; %p11
trace(X*B{8})==0; %p12
trace(X*B{12})==0; %p13
trace(X*B{6})==0; %p20
trace(X*B{9})==0; %p21
trace(X*B{13})==0; %p22
trace(X*B{10})==0; %p30
trace(X*B{14})==0; %p31
trace(X*B{15})==1; %p40
X>=0;
cvx_end

which (also) gives \texttt{cvx\_optval} = -11.4581 and

\[
X^* = \begin{bmatrix}
11.5 & -2 & -2 & -1.2 & -1.11 & -1.2 \\
-2 & 2.4 & -0.886 & 0 & -0.00755 & 0.00755 \\
-2 & -0.886 & 2.4 & 0.00755 & -0.00755 & 0 \\
-1.2 & 0 & 0.00755 & 1 & 0 & -0.323 \\
-1.11 & -0.00755 & -0.00755 & 0 & 0.646 & 0 \\
-1.2 & 0.00755 & 0 & -0.323 & 0 & 1
\end{bmatrix} \quad (A.18)
\]

A.3.3 Example 3

Lasserre 2001, pg. 808, Example 3 Consider the polynomial \( p(x) : \mathbb{R}^2 \to \mathbb{R}, \)

\[
x \mapsto p(x) = x_1^2 x_2^2 (x_1^2 + x_2^2 - 1)
\]

(A.19)

Expanding \( p \) gives:

\[
p(x) = x_1^4 x_2^2 + x_1^2 x_2^4 - x_1^2 x_2^2
\]

(A.20)

from here we can see that, in moment \( (y_\alpha) \) notation, we have:

\[
\sum_\alpha p_\alpha y_\alpha = y_{42} + y_{24} - y_{22}
\]

(A.21)

Solving \( Q_1^3 \) using the code below:
% AUTHOR: Jose Lopez NEU 2014
% LICENSE: This code is licensed under the BSD License.

% This script implements Lasserre 2001 example 3

clear; clc;
M = get_mmatrix(3,2);
[monomials, n_monomials] = get_monomials(M);
maps.M3 = get_map(M, monomials);

%form localizing matrix L
M = get_mmatrix(2,2);
maps.M2 = get_map(M, monomials);
L = localize(M, [2,0]);
maps.L20 = get_map(L, monomials);
L = localize(M, [0,2]);
maps.L02 = get_map(L, monomials);

indices.y42 = get_index(monomials, [4,2]);
indices.y24 = get_index(monomials, [2,4]);
indices.y22 = get_index(monomials, [2,2]);
indices.y20 = get_index(monomials, [2,0]);
indices.y02 = get_index(monomials, [0,2]);

%%
cvx_clear;
cvx_begin sdp
    cvx_solver sedumi;
end

variable y(n_monomials);

M3 = assignm(y, maps.M3);
M2 = assignm(y, maps.M2);
L20 = assignm(y, maps.L20);
L02 = assignm(y, maps.L02);

a = 1;
L = a*M2-L20-L02;

%form objective
y00 = y(1);
y42 = y(indices.y42);
y24 = y(indices.y24);
y22 = y(indices.y22);
APPENDIX A. NEU MOMENTS V0.1

\begin{verbatim}
y20 = y(indices.y20);
y02 = y(indices.y02);
minimize(y42+y24-y22)
subject to
M3>=0
L>=0
y00 == 1;
cvx_end
\end{verbatim}

\begin{verbatim}%cvx_optval: -0.037037 matches
gives: cvx_optval = -0.037037 and x_2^1 = x_2^2 = 0.33339 which is close to the true
values of p^* = -1/27 and x_2^1 = x_2^2 = 1/3.
The code for the dual is given below:
\end{verbatim}

\begin{verbatim}
\% AUTHOR: Jose Lopez NEU 2014
\% LICENSE:This code is licensed under the BSD License.
\%
\% This example implements Lasserre example 3, dual formulation. I made this
\% before I knew how to get the dual from the primal..
\%
clear;clc;

\%
M = get_mmatrix(3,2);
[monomials,n_monomials] = get_monomials(M);
map = get_map(M,monomials);
B = get_index_matrices(monomials,map); \%get index matrices
M = get_mmatrix(2,2);
L20 = localize(M,[2,0]);
L02 = localize(M,[0,2]);
\%form localizing matrix L
C = get_index_matrices(monomials,get_map(M,monomials));
C20 = get_index_matrices(monomials,get_map(L20,monomials));
C02 = get_index_matrices(monomials,get_map(L02,monomials));

\%
cvx_clear;
cvx_begin sdp
cvx_solver sedumi;
\end{verbatim}
APPENDIX A. NEU MOMENTS V0.1

\begin{verbatim}
a = 1;
pa = zeros(28,1);
idx = get_index(monomials, [4 2; 2 4; 2 2]);
pa(idx(1:2)) = 1;
pa(idx(3)) = -1;

variable X(10,10) symmetric;
variable Z(6,6) symmetric;
dual variables z{n_monomials-1};

%form objective
maximize( -X(1,1) - a^2 * Z(1,1) )
subject to
X>=0;
Z>=0;
for k = 2:n_monomials
    trace(X*B{k}) + trace(Z*(a*C{k} - C20{k} - C02{k})) == pa(k) : z{k-1};
end

CVX_end
% (cvx_optval): -0.037037 matches
\end{verbatim}

A.3.4 Example 5

Lasserre 2001, pg. 811, Example 5 Let \( p(x) : \mathbb{R}^2 \rightarrow \mathbb{R} \) be the concave polynomial:
\[
x \mapsto p(x) = -(x_1 - 1)^2 - (x_1 - x_2)^2 - (x_2 - 3)^2
\]
and
\[
K = \{(x_1, x_2) \in \mathbb{R}^2 | 1 - (x_1 - 1)^2 \geq 0; 1 - (x_1 - x_2)^2 \geq 0; 1 - (x_2 - 3)^2 \geq 0\} \quad (A.22)
\]

The code below solves \( Q_K^2 \):

% AUTHOR: Jose Lopez NEU 2014
% LICENSE: This code is licensed under the BSD License.
% This example implements Lasserre example 5.
clear;clc;
M = get_mmatrix(2,2);
[monomials,n_monomials] = get_monomials(M);
maps.M2 = get_map(M,monomials);

M = get_mmatrix(1,2);
maps.M1 = get_map(M,monomials);

L = localize(M,[1,0]);
maps.L10 = get_map(L,monomials);
L = localize(M,[0,1]);
maps.L01 = get_map(L,monomials);
L = localize(M,[2,0]);
maps.L20 = get_map(L,monomials);
L = localize(M,[1,1]);
maps.L11 = get_map(L,monomials);
L = localize(M,[0,2]);
maps.L02 = get_map(L,monomials);

%%
cvx_clear;
cvx_begin sdp
  cvx_solver sedumi;

variable y(n_monomials);
M2 = assignm(y,maps.M2);
M1 = assignm(y,maps.M1);
L10 = assignm(y,maps.L10);
L01 = assignm(y,maps.L01);
L20 = assignm(y,maps.L20);
L11 = assignm(y,maps.L11);
L02 = assignm(y,maps.L02);

y00 = y(1);
y01 = y(get_index(monomials,[0,1]));
y10 = y(get_index(monomials,[1,0]));
y11 = y(get_index(monomials,[1,1]));
y20 = y(get_index(monomials,[2,0]));
y02 = y(get_index(monomials,[0,2]));

minimize(-2*y20 + 2*y11 + 2*y10 - 2*y02 + 6*y01)
subject to
APPENDIX A. NEU MOMENTS V0.1

\begin{align*}
\text{M2} & \geq 0; \\
-L_20 + 2L_{10} & \geq 0; \\
-L_20 + 2L_{11} - L_{02} + M_1 & \geq 0; \\
-L_{02} + 6L_{01} - 8M_1 & \geq 0; \\
y_{00} & = 1; \\
\text{cvx\_end} \\
[y_{10}, y_{01}] \\
\% (cvx\_optval): 8 \text{ true optimal} = 8 \\
\text{and gives: cvx\_optval} & = 8 \text{ at } (x_1, x_2) = (1.5135, 2.1741).
\end{align*}

A.3.5 Problem 2.2

Lasserre 2001, pg. 812

\begin{verbatim}
% AUTHOR: Jose Lopez NEU 2014
% LICENSE: This code is licensed under the BSD License.
% This script implements Lasserre Problem 2.2

clear; clc;

Q = eye(5);
c = -[10.5, 7.5, 3.5, 2.5, 1.5]';
d = -10;
[M, basis_data] = get_mm(2, 6);
[monomials, n_monomials] = get_monomials(M);
maps.M2 = get_map(M, monomials);
% form localizing matrix L
M = get_mm(1, 6);
maps.M1 = get_map(M, monomials);
L = localize(M, [1 0 0 0 0 0]);
maps.L100000 = get_map(L, monomials);
L = localize(M, [0 1 0 0 0 0]);
maps.L010000 = get_map(L, monomials);
L = localize(M, [0 0 1 0 0 0]);
maps.L001000 = get_map(L, monomials);
L = localize(M, [0 0 0 1 0 0]);
maps.L000100 = get_map(L, monomials);
L = localize(M, [0 0 0 0 1 0]);
\end{verbatim}
APPENDIX A. NEU MOMENTS V0.1

maps.L000010 = get_map(L,monomials);
L = localize(M,[0 0 0 0 0 1]);
maps.L000001 = get_map(L,monomials);
indices.y200000 = get_index(monomials,[2 0 0 0 0 0]);
indices.y100000 = get_index(monomials,[1 0 0 0 0 0]);
indices.y020000 = get_index(monomials,[0 2 0 0 0 0]);
indices.y010000 = get_index(monomials,[0 1 0 0 0 0]);
indices.y002000 = get_index(monomials,[0 0 2 0 0 0]);
indices.y001000 = get_index(monomials,[0 0 1 0 0 0]);
indices.y000200 = get_index(monomials,[0 0 0 2 0 0]);
indices.y000100 = get_index(monomials,[0 0 0 1 0 0]);
indices.y000020 = get_index(monomials,[0 0 0 0 2 0]);
indices.y000010 = get_index(monomials,[0 0 0 0 1 0]);
indices.y000001 = get_index(monomials,[0 0 0 0 0 1]);

%%
cvx_clear;
cvx_begin sdp
cvx_solver sedumi;

variable y(n_monomials);

M2 = assignm(y,maps.M2);
M1 = assignm(y,maps.M1);
L100000 = assignm(y,maps.L100000);
L010000 = assignm(y,maps.L010000);
L001000 = assignm(y,maps.L001000);
L000100 = assignm(y,maps.L000100);
L000010 = assignm(y,maps.L000010);
L000001 = assignm(y,maps.L000001);

%get aliases
y000000 = y(1);
y200000 = y(indices.y200000);
y100000 = y(indices.y100000);
y020000 = y(indices.y020000);
y010000 = y(indices.y010000);
y002000 = y(indices.y002000);
y001000 = y(indices.y001000);
y000200 = y(indices.y000200);
APPENDIX A. NEU MOMENTS V0.1

\begin{verbatim}
y000100 = y(indices.y000100);
y000020 = y(indices.y000020);
y000010 = y(indices.y000010);
y000001 = y(indices.y000001);

%-x1^2/2-(21*x1)/2- x2^2/2-(15*x2)/2- x3^2/2-(7*x3)/2-x4^2/2-(5*x4)/2-x5^2/2-(3*x5)/2-10*x6
minimize( -y200000/2-21*y100000/2-y020000/2-15*y010000/2-y002000/2-7*y001000... 
/2-...
y000200-5*y000100/2-y000020/2-3*y000010/2-10*y000001 )
%minimize(c*x-0.5*x'*Q*x+d*y) %CVX complains
subject to
y000000==1;
M2>=0;
6.5*M1-6*L100000-3*L010000-3*L001000-2*L000100-L000010>=0
20*M1-10*L100000-10*L001000-L000001>=0
L000001>=0
L100000>=0
L010000>=0
L001000>=0
L000100>=0
L000010>=0
M1-L100000>=0
M1-L010000>=0
M1-L001000>=0
M1-L000100>=0
M1-L000010>=0
cvx_end
%(cvx_optval): -213.5
\end{verbatim}

A.3.6 Problem 2.6

Lasserre 2001, pg. 812

% AUTHOR: Jose Lopez NEU 2014
% LICENSE:This code is licensed under the BSD License.
% This script implements Lasserre (2001) Problem 2.6
APPENDIX A. NEU MOMENTS V0.1

% clear;clc;

% [M,basis_data] = get_mmatrix(2,10);
[monomials,n_monomials] = get_monomials(M);
maps.M2 = get_map(M,monomials);
%form localizing matrix L
M = get_mmatrix(1,10);
maps.M1 = get_map(M,monomials);
mons = basis_data.monomial{2}; %first order monomials
maps.L = cell(10,1);
for k = 1:10
    L = localize(M,mons(k,:));
    maps.L{k} = get_map(L,monomials);
end
indices.x = get_index(monomials,mons);
indices.x2 = get_index(monomials,mons+mons);

% cvx_clear;
cvx_begin sdp
cvx_solver sedumi;

variable y(n_monomials);
M2 = assignm(y,maps.M2);
M1 = assignm(y,maps.M1);

L = cell(10,1);
for k = 1:10
    L(k) = assignm(y,maps.L{k});
end

x1 = y(indices.x(1));
x2 = y(indices.x(2));
x3 = y(indices.x(3));
x4 = y(indices.x(4));
x5 = y(indices.x(5));
x6 = y(indices.x(6));
x7 = y(indices.x(7));
x8 = y(indices.x(8));
APPENDIX A. NEU MOMENTS V0.1

\[
x_9 = y(indices.x(9));
x_{10} = y(indices.x(10));
x_{1_2} = y(indices.x2(1));
x_{2_2} = y(indices.x2(2));
x_{3_2} = y(indices.x2(3));
x_{4_2} = y(indices.x2(4));
x_{5_2} = y(indices.x2(5));
x_{6_2} = y(indices.x2(6));
x_{7_2} = y(indices.x2(7));
x_{8_2} = y(indices.x2(8));
x_{9_2} = y(indices.x2(9));
x_{10_2} = y(indices.x2(10));
\]

\[
\text{objective} = -50x_{1_2} + 48x_{1} - 50x_{2_2} + 42x_{2} - 50x_{3_2} + 48x_{3} - 50x_{4_2} + 45x_{4} - 50x_{5_2} + 44x_{5} - 50x_{6_2} + 41x_{6} - 50x_{7_2} + 47x_{7} - 50x_{8_2} + 42x_{8} - 50x_{9_2} + 45x_{9} - 50x_{10_2} + 46x_{10};
\]

\[
\text{minimize(objective)}
\]

\[
\text{subject to}
\]
\[
y(1)==1;
M2>=0;
\]

\[
\text{for } k = 1:10
\]
\[
\text{L{k}}>=0;
M1 - \text{L{k}}>=0;
\]
\[
\text{end}
\]

\[
-4*M1 - (-2*L{1} - 6*L{2} - L{3} - 3*L{5} - 3*L{6} - 2*L{7} - 6*L{8} - 2*L{9} - 2*L{10}) >= 0;
22*M1 - (6*L{1} - 5*L{2} + 8*L{3} - 3*L{4} + L{6} + 3*L{7} + 8*L{8} + 9*L{9} - 3*L{10}) >= 0;
-6*M1 - (6*L{2} - 5*L{1} + 5*L{3} + 3*L{4} + 8*L{5} - 8*L{6} + 9*L{7} + 2*L{8} - 9*L{10}) >= 0;
-23*M1 - (9*L{1} + 5*L{2} - 9*L{4} + L{5} - 8*L{6} + 3*L{7} - 9*L{8} - 9*L{9} - 3*L{10}) >= 0;
-12*M1 - (7*L{2} - 8*L{1} - 4*L{3} - 5*L{4} - 9*L{5} + L{6} - 7*L{7} - L{8} + 3*L{9} - 2*L{10}) >= 0;
\]
\[
cvx\_end
\]

\[
%(cvx_optval): -39, M1 is rank-1 so we can read off optimizers
\]
\[
%x* = (1001110111)
\]
\[
y(indices.x)
\]

A.3.7 Problem 2.9

Lasserre 2001, pg. 812
% AUTHOR: Jose Lopez NEU 2014
% LICENSE:This code is licensed under the BSD License.
% This script implements Lasserre (2001) Problem 2.9

clear;clc;

%form moment matrix
M = get_mmatrix(2,10);
[monomials, n_monomials] = get_monomials(M);
maps.M2 = get_map(M,monomials);

%form localizing matrices
%do yourself a favor and put the L's in a cell array...
M = get_mmatrix(1,10);
maps.M1 = get_map(M,monomials);
L = localize(M,[1 0 0 0 0 0 0 0 0 0]);
maps.L1000000000 = get_map(L,monomials);
L = localize(M,[0 1 0 0 0 0 0 0 0 0]);
maps.L0100000000 = get_map(L,monomials);
L = localize(M,[0 0 1 0 0 0 0 0 0 0]);
maps.L0010000000 = get_map(L,monomials);
L = localize(M,[0 0 0 1 0 0 0 0 0 0]);
maps.L0001000000 = get_map(L,monomials);
L = localize(M,[0 0 0 0 1 0 0 0 0 0]);
maps.L0000100000 = get_map(L,monomials);
L = localize(M,[0 0 0 0 0 1 0 0 0 0]);
maps.L0000010000 = get_map(L,monomials);
L = localize(M,[0 0 0 0 0 0 1 0 0 0]);
maps.L0000001000 = get_map(L,monomials);
L = localize(M,[0 0 0 0 0 0 0 1 0 0]);
maps.L0000000100 = get_map(L,monomials);

indices.y1100000000 = get_index(monomials,[1 1 0 0 0 0 0 0 0 0]);
indices.y0110000000 = get_index(monomials,[0 1 1 0 0 0 0 0 0 0]);
indices.y0011000000 = get_index(monomials,[0 0 1 1 0 0 0 0 0 0]);
indices.y0001100000 = get_index(monomials,[0 0 0 1 1 0 0 0 0 0]);
APPENDIX A. NEU MOMENTS V0.1

indices.y0000110000 = get_index(monomials,[0 0 0 0 1 1 0 0 0 0]);
indices.y0000011000 = get_index(monomials,[0 0 0 0 0 1 1 0 0 0]);
indices.y0000001100 = get_index(monomials,[0 0 0 0 0 0 1 1 0 0]);
indices.y0000000110 = get_index(monomials,[0 0 0 0 0 0 0 1 1 0]);
indices.yl1010000000 = get_index(monomials,[1 0 1 0 0 0 0 0 0 0]);
indices.y0101000000 = get_index(monomials,[0 1 0 1 0 0 0 0 0 0]);
indices.y0011010000 = get_index(monomials,[0 0 1 0 1 0 0 0 0 0]);
indices.y0001101000 = get_index(monomials,[0 0 0 1 0 1 0 0 0 0]);
indices.y0000110100 = get_index(monomials,[0 0 0 0 1 0 1 0 0 0]);
indices.y0000011010 = get_index(monomials,[0 0 0 0 0 1 0 1 0 0]);
indices.y0000001110 = get_index(monomials,[0 0 0 0 0 0 1 0 1 0]);
indices.y0000000110 = get_index(monomials,[0 0 0 0 0 0 0 1 0 1]);

cvx_clear;

%%

cvx_begin sdp

cvx_solver sedumi;

variable y(n_monomials);

M2 = assignm(y,maps.M2);
M1 = assignm(y,get_map(M,monomials));
L1000000000 = assignm(y,maps.L1000000000);
L0100000000 = assignm(y,maps.L0100000000);
L0010000000 = assignm(y,maps.L0010000000);
L0001000000 = assignm(y,maps.L0001000000);
L0000100000 = assignm(y,maps.L0000100000);
L0000010000 = assignm(y,maps.L0000010000);
L0000001000 = assignm(y,maps.L0000001000);
L0000000100 = assignm(y,maps.L0000000100);
L0000000010 = assignm(y,maps.L0000000010);
L0000000001 = assignm(y,maps.L0000000001);

% get aliases and form objective
% do yourself a favor and use arrays instead of individual aliases
APPENDIX A. NEU MOMENTS V0.1

\[ y_{1000000000} = y(\text{indices}.y_{1000000000}); \]
\[ y_{0110000000} = y(\text{indices}.y_{0110000000}); \]
\[ y_{0011000000} = y(\text{indices}.y_{0011000000}); \]
\[ y_{0001100000} = y(\text{indices}.y_{0001100000}); \]
\[ y_{0000110000} = y(\text{indices}.y_{0000110000}); \]
\[ y_{0000011000} = y(\text{indices}.y_{0000011000}); \]
\[ y_{0000001100} = y(\text{indices}.y_{0000001100}); \]
\[ y_{0000000110} = y(\text{indices}.y_{0000000110}); \]
\[ y_{0000000011} = y(\text{indices}.y_{0000000011}); \]
\[ y_{1010000000} = y(\text{indices}.y_{1010000000}); \]
\[ y_{0101000000} = y(\text{indices}.y_{0101000000}); \]
\[ y_{0010100000} = y(\text{indices}.y_{0010100000}); \]
\[ y_{0001010000} = y(\text{indices}.y_{0001010000}); \]
\[ y_{0000101000} = y(\text{indices}.y_{0000101000}); \]
\[ y_{0000010100} = y(\text{indices}.y_{0000010100}); \]
\[ y_{0000001010} = y(\text{indices}.y_{0000001010}); \]
\[ y_{0000000101} = y(\text{indices}.y_{0000000101}); \]
\[ y_{1000001000} = y(\text{indices}.y_{1000001000}); \]
\[ y_{1000000010} = y(\text{indices}.y_{1000000010}); \]
\[ y_{1000000001} = y(\text{indices}.y_{1000000001}); \]
\[ y_{0100000001} = y(\text{indices}.y_{0100000001}); \]
\[ y_{0001001000} = y(\text{indices}.y_{0001001000}); \]

\[ \text{maximize} (\text{obj}) \]
\[ \text{subject to} \]
\[ y(1)==1 \]
\[ M2>=0 \]
\[ L_{1000000000}+L_{0100000000}+L_{0010000000}+L_{0001000000}+L_{0000100000}+L_{0000010000}+L_{0000001000}+L_{0000000100}+L_{0000000010}+L_{0000000001}-M1==0 \]
\[ L_{1000000000}>=0 \]
\[ L_{0100000000}>=0 \]
\[ L_{0010000000}>=0 \]
\[ L_{0001000000}>=0 \]
APPENDIX A. NEU MOMENTS V0.1

\[ L0000100000 \geq 0 \]
\[ L0000010000 \geq 0 \]
\[ L0000001000 \geq 0 \]
\[ L0000000100 \geq 0 \]
\[ L0000000010 \geq 0 \]
\[ L0000000001 \geq 0 \]
\[ \text{cvx\_end} \]
\%(cvx\_optval): +0.375

A.3.8 Problem 3.3

Lasserre 2001, pg. 813

% AUTHOR: Jose Lopez NEU 2014
% LICENSE:This code is licensed under the BSD License.
% This script implements Lasserre Problem 3.3.
clear;clc;
%
%form moment matrix
M = get_mmatrix(2,6);
[monomials,n_monomials] = get_monomials(M);
maps.M2 = get_map(M,monomials);
%form localizing matrices
M = get_mmatrix(1,6);
maps.M1 = get_map(M,monomials);
L = localize(M, [1 0 0 0 0 0]);
maps.L100000 = get_map(L,monomials);
L = localize(M, [0 1 0 0 0 0]);
maps.L010000 = get_map(L,monomials);
L = localize(M, [0 0 1 0 0 0]);
maps.L001000 = get_map(L,monomials);
L = localize(M, [0 0 0 1 0 0]);
maps.L000100 = get_map(L,monomials);
L = localize(M, [0 0 0 0 1 0]);
maps.L000010 = get_map(L,monomials);
L = localize(M, [0 0 0 0 0 1]);
maps.L000001 = get_map(L,monomials);
L = localize(M, [0 0 2 0 0 0]);
APPENDIX A. NEU MOMENTS V0.1

maps.L002000 = get_map(L,monomials);
L = localize(M,[0 0 0 0 2 0]);
maps.L000020 = get_map(L,monomials);

%%
cvx_clear;
cvx_begin sdp
cvx_solver sedumi;

variable y(n_monomials);
M2 = assignm(y,maps.M2);
M1 = assignm(y,maps.M1);
L100000 = assignm(y,maps.L100000);
L010000 = assignm(y,maps.L010000);
L001000 = assignm(y,maps.L001000);
L000100 = assignm(y,maps.L000100);
L000010 = assignm(y,maps.L000010);
L000001 = assignm(y,maps.L000001);
L002000 = assignm(y,maps.L002000);
L000020 = assignm(y,maps.L000020);

%get aliases and form objective, all the calls to get_index should be
%done above with the get_map’s...
y200000 = y(get_index(monomials,[2 0 0 0 0 0]));
y100000 = y(get_index(monomials,[1 0 0 0 0 0]));
y020000 = y(get_index(monomials,[0 2 0 0 0 0]));
y010000 = y(get_index(monomials,[0 1 0 0 0 0]));
y002000 = y(get_index(monomials,[0 0 2 0 0 0]));
y001000 = y(get_index(monomials,[0 0 1 0 0 0]));
y000200 = y(get_index(monomials,[0 0 0 2 0 0]));
y000100 = y(get_index(monomials,[0 0 0 1 0 0]));
y000020 = y(get_index(monomials,[0 0 0 0 2 0]));
y000010 = y(get_index(monomials,[0 0 0 0 1 0]));
y000002 = y(get_index(monomials,[0 0 0 0 0 2]));
y000001 = y(get_index(monomials,[0 0 0 0 0 1]));

%syms x1 x2 x3 x4 x5 x6 real;
%expand(-25*(x1-2)^2-(x2-2)^2-(x3-1)^2-(x4-4)^2-(x5-1)^2-(x6-4)^2)
%-25*x1^2 + 100*x1 -x2^2 + 4*x2 -x3^2 + 2*x3 -x4^2 + 8*x4 -x5^2 + 2*x5 -x6^2 ...
+ 8*x6 - 138
APPENDIX A. NEU MOMENTS V0.1

objective = -25\cdot y_{20000} + 100\cdot y_{10000} - y_{02000} + 4\cdot y_{01000} - y_{00200} + 2\cdot y_{00100} - ... \\
y_{00020} + 8\cdot y_{00010} - y_{00002} + 2\cdot y_{00001} - 8\cdot y_{00000} - 138;
minimize(objective);

subject to
y(1)==1
M2>=0
L_{002000} - 6\cdot L_{001000} + L_{000100} + 5\cdot M1 >= 0
L_{000200} - 6\cdot L_{000010} + L_{000001} + 5\cdot M1 >= 0
2\cdot M1 - L_{100000} + 3\cdot L_{010000} >= 0
2\cdot M1 + L_{100000} - L_{010000} >= 0
6\cdot M1 - L_{100000} - L_{010000} >= 0
L_{100000} + L_{010000} - 2\cdot M1 >= 0
L_{100000} >= 0
L_{010000} >= 0
L_{001000} - M1 >= 0
5\cdot M1 - L_{001000} >= 0
L_{000100} >= 0
6\cdot M1 - L_{000100} >= 0
L_{000010} - M1 >= 0
5\cdot M1 - L_{000010} >= 0
L_{000001} >= 0
10\cdot M1 - L_{000001} >= 0
cvx_end

\text{%answer: -310, M1 is rank-1, so we can read-off optimizers}
M1(:,1)
A.3.9 Problem 3.4

Lasserre 2001, pg. 813

\[
\begin{align*}
\min_x p(x) &= -2x_1 + x_2 - x_3 \\
x_1 + x_2 + x_3 &\leq 4 \\
x_1 &\leq 2 \\
x_3 &\leq 3 \\
3x_2 + x_3 &\leq 6 \\
x_1 &\geq 0 \\
x_2 &\geq 0 \\
x_3 &\geq 0 \\
x^T B^T B x - 2r^T B x + r^T r - 0.25(b - v)^T (b - v)
\end{align*}
\]

with

\[
B = \begin{bmatrix}
0 & 0 & 1 \\
0 & -1 & 0 \\
-2 & 1 & -1
\end{bmatrix}, \quad b = \begin{bmatrix}
3 \\
0 \\
-4
\end{bmatrix}, \quad v = \begin{bmatrix}
0 \\
-1 \\
-6
\end{bmatrix}, \quad r = \begin{bmatrix}
2 \\
-0.5 \\
-5
\end{bmatrix}
\]

The code below solves $Q^4_K$:

% AUTHOR: Jose Lopez NEU 2014
% LICENSE: This code is licensed under the BSD License.
% %
% This example implements Lasserre Problem 3.4 in his 2001 paper.
clear;clc;

% form moment matrix
M = get_mmatrix(4,3);
[monomials, n_monomials] = get_monomials(M);
maps.M4 = get_map(M,monomials);

% form localizing matrices
M = get_mmatrix(3,3);
maps.M3 = get_map(M,monomials);
L = localize(M, [1,0,0]);
maps.L3_100 = get_map(L,monomials);
L = localize(M, [0,1,0]);
APPENDIX A. NEU MOMENTS V0.1

maps.L3_010 = get_map(L,monomials);
L = localize(M,[0,0,1]);
maps.L3_001 = get_map(L,monomials);
M = get_mmatrix(2,3);
maps.M2 = get_map(M,monomials);
L = localize(M,[1,0,0]);
maps.L2_100 = get_map(L,monomials);
L = localize(M,[0,1,0]);
maps.L2_010 = get_map(L,monomials);
L = localize(M,[0,0,1]);
maps.L2_001 = get_map(L,monomials);
L = localize(M,[0,0,2]);
maps.L2_002 = get_map(L,monomials);
L = localize(M,[0,1,1]);
maps.L2_011 = get_map(L,monomials);
L = localize(M,[2,0,0]);
maps.L2_200 = get_map(L,monomials);
L = localize(M,[1,1,0]);
maps.L2_110 = get_map(L,monomials);
L = localize(M,[1,0,1]);
maps.L2_101 = get_map(L,monomials);
L = localize(M,[0,2,0]);
maps.L2_020 = get_map(L,monomials);

%%
cvx_clear;
cvx_begin sdp
cvx_solver sedumi;

variable y(n_monomials);

M4 = assignm(y,maps.M4);
M3 = assignm(y,maps.M3);
L3_100 = assignm(y,maps.L3_100);
L3_010 = assignm(y,maps.L3_010);
L3_001 = assignm(y,maps.L3_001);
M2 = assignm(y,maps.M2);
L2_100 = assignm(y,maps.L2_100);
L2_010 = assignm(y,maps.L2_010);
L2_001 = assignm(y,maps.L2_001);
APPENDIX A. NEU MOMENTS V0.1

L2_002 = assignm(y,maps.L2_002);
L2_011 = assignm(y,maps.L2_011);
L2_200 = assignm(y,maps.L2_200);
L2_110 = assignm(y,maps.L2_110);
L2_101 = assignm(y,maps.L2_101);
L2_020 = assignm(y,maps.L2_020);

B = [0,0,1;0,-1,0;-2,1,-1];
b = [3;0;-4];
v = [0;-1;-6];
r = [1.5;0.5;0];
y000 = y(1);
y100 = y(get_index(monomials,[1,0,0]));
y010 = y(get_index(monomials,[0,1,0]));
y001 = y(get_index(monomials,[0,0,1]));
minimize(-2*y100+y010-y001)
subject to
y000==1
M4>=0
4*M3-L3_100-L3_010-L3_001>=0
2*M3-L3_100>=0
3*M3-L3_001>=0
6*M3-3*L3_010-L3_001>=0
L3_100>=0
L3_010>=0
L3_001>=0
4*L2_200 - 4*L2_110 + 4*L2_101 - 20*L2_100 + 2*L2_020 - 2*L2_011 +...
9*L2_010 + 2*L2_002 - 13*L2_001 + 24*M2>=0
cvx_end
%(cvx_optval): -4 matches

and gives $p^* = 4$.

A.3.10 Problem 4.6

Lasserre 2001, pg. 813

% AUTHOR: Jose Lopez NEU 2014
% LICENSE: This code is licensed under the BSD License.
%%
clear;clc;

M4 = get_mmatrix(4,2);
M3 = get_mmatrix(3,2);
L40 = localize(M3,[4,0]);
L30 = localize(M3,[3,0]);
L20 = localize(M3,[2,0]);
L01 = localize(M3,[0,1]);
L10 = localize(M3,[1,0]);

[monomials,n_monomials] = get_monomials({M4,L40,L30});
n_monomials = size(monomials,1);
maps.M4 = get_map(M4,monomials);
maps.M3 = get_map(M3,monomials);
maps.L40 = get_map(L40,monomials);
maps.L30 = get_map(L30,monomials);
maps.L20 = get_map(L20,monomials);
maps.L01 = get_map(L01,monomials);
maps.L10 = get_map(L10,monomials);

%%
%Lasserre example 4.6
cvx_clear;
cvx_begin sdp
cvx_solver sedumi;

variable y(n_monomials);

M4 = assignm(y,maps.M4);
M3 = assignm(y,maps.M3);
L40 = assignm(y,maps.L40);
L30 = assignm(y,maps.L30);
L20 = assignm(y,maps.L20);
L01 = assignm(y,maps.L01);
L10 = assignm(y,maps.L10);

y10 = y(get_index(monomials,[1 0]));
y01 = y(get_index(monomials,[0 1]));

minimize(-y10-y01)
subject to
y(1) == 1;
M4>=0
2*L40-8*L30+8*L20+2*M3-L01>=0
4*L40-32*L30+88*L20-96*L10+36*M3-L01>=0
L10>=0
3*M3-L10>=0
L01>=0
4*M3-L01>=0
cvx_end
%answer -5.508

A.3.11 Problem 4.7

Lasserre 2001, pg. 813

\[
\min_x p(x) = -12x_1 - 7x_2 + x_2^2 - 2x_1^4 + 2 - x_2 = 0 \\
0 \leq x_1 \leq 3 \\
0 \leq x_2 \leq 4
\]

The following code solves $Q_5^5$:

% AUTHOR: Jose Lopez NEU 2014
% LICENSE:This code is licensed under the BSD License.
%
% This example implements Lasserre problem 4.7 in his 2001 paper.
%
clear;clc;

M = get_mmatrix(5,2);
[monomials,n_monomials] = get_monomials(M);
maps.M5 = get_map(M,monomials)

M = get_mmatrix(2,2);
maps.M2 = get_map(M,monomials);
L = localize(M,[4,0]);
APPENDIX A. NEU MOMENTS V0.1

maps.L2_40 = get_map(L,monomials);
L = localize(M,[0,1]);
maps.L2_01 = get_map(L,monomials);
M = get_mmatrix(1,2);
maps.M1 = get_map(M,monomials);
L = localize(M,[1,0]);
maps.L1_10 = get_map(L,monomials);
L = localize(M,[0,1]);
maps.L1_01 = get_map(L,monomials);
indices.y10 = get_index(monomials,[1,0]);
indices.y01 = get_index(monomials,[0,1]);
indices.y02 = get_index(monomials,[0,2]);
indices.y40 = get_index(monomials,[4,0]);

%%
cvx_clear;
cvx_begin sdp
cvx_solver sedumi;

variable y(n_monomials);
M5 = assignm(y,maps.M5);
M2 = assignm(y,maps.M2);
L2_40 = assignm(y,maps.L2_40);
L2_01 = assignm(y,maps.L2_01);
M1 = assignm(y,maps.M1);
L1_10 = assignm(y,maps.L1_10);
L1_01 = assignm(y,maps.L1_01);

y00 = y(1);
y10 = y(indices.y10);
y01 = y(indices.y01);
y02 = y(indices.y02);
y40 = y(indices.y40);

minimize(-12*y10-7*y01+y02)
subject to
y00==1
M5>=0
-2*L2_40+2*M2-L2_01==0
L1_10>=0
APPENDIX A. NEU MOMENTS V0.1

2*M1-L1_10>=0
L1_01>=0
3*M1-L1_01>=0
cvx_end

[y10,y01] %M1 is rank-1
%(cvx_optval): -16.7389 matches
%x* = (0.71756,1.4698)

and gives $p^* = -16.7389$.

A.3.12 A small 0-1 problem

Consider the problem:

$$\begin{align*}
\max & \quad c^T x \\
\text{s.t.} & \quad w^T b \leq b \\
& \quad x_i \in \{0, 1\}, \forall i = 1, \ldots, 5
\end{align*}$$

where

$$c = \begin{bmatrix} 3 \\ 0.9 \\ 2 \\ 3 \\ 3 \end{bmatrix}, \quad w = \begin{bmatrix} 2 \\ 6 \\ 5 \\ 1 \\ 4 \end{bmatrix}, \quad \text{and } b = 7 \quad \text{(A.24)}$$

The only difference between this program and other polynomials is that we need to call \texttt{make-zero-one} to replace higher order terms with 1 (since $x_i^n = x_i$ when $x_i$ is binary). This is done instead of adding constraints of the type $x_i^2 = x_i$. See [97] for more info.

The code below:

```matlab
% AUTHOR: Jose Lopez NEU 2014
% LICENSE: This code is licensed under the BSD License.
%
% try a simple 0-1 problem
clear;clc;
%
c = [2.7797, 0.9439, 1.8720, 2.5866, 3.2963]';
```
APPENDIX A. NEU MOMENTS V0.1

w = [2, 6, 5, 1, 4]';
b = 7;

M = get_mmatrix(1,5);
M = make_zero_one(M);
[monomials, n_monomials] = get_monomials(M);

%%
cvx_clear;
cvx_begin sdp
cvx_solver sedumi;

variable y(n_monomials);
M1 = assignm(y, get_map(M, monomials));

y00000 = y(1);
y10000 = y(get_index(monomials, [1,0,0,0,0]));
y01000 = y(get_index(monomials, [0,1,0,0,0]));
y00100 = y(get_index(monomials, [0,0,1,0,0]));
y00010 = y(get_index(monomials, [0,0,0,1,0]));
y00001 = y(get_index(monomials, [0,0,0,0,1]));

maximize(c(1)*y10000 + c(2)*y01000 + c(3)*y00100 + c(4)*y00010 + c(5)*y00001);
subject to
y00000 == 1
M1 >= 0
b - w(1)*y10000 - w(2)*y01000 - w(3)*y00100 - w(4)*y00010 - w(5)*y00001 >= 0

cvx_end
cvx_optval
x = [y10000, y01000, y00100, y00010, y00001]'
cost = w' * x

% correct answer: x = [1, 0, 0, 1, 1], cost = 7, cvx_optval = 8.6626

yields the correct result: x^T = [1, 0, 0, 1, 1], w^T x = 7, p^* = 8.6626.
A.3.13 A bigger 0-1 problem

Consider the problem:

\[
\begin{cases}
\max \limits_{x} c^T x \\
w^T b \leq b \\
x_i \in \{0, 1\}, \forall i = 1, \ldots, 5
\end{cases}
\]
APPENDIX A. NEU MOMENTS V0.1

where \( b = 71 \) and \( c \) and \( w \) are \( 75 \times 1 \) vectors given by:

\[
\begin{bmatrix}
c(1 : 25) & c(26 : 50) & c(51 : 75) \\
0.2878 & 0.3888 & 2.403 \\
2.243 & 0.6752 & 0.6989 \\
2.246 & 1.05 & 2.797 \\
1.63 & 0.8613 & 2.29 \\
1.014 & 2.782 & 2.479 \\
2.497 & 0.1539 & 1.72 \\
1.658 & 1.778 & 2.378 \\
2.873 & 0.4887 & 0.9871 \\
2.678 & 2.515 & 0.6701 \\
1.07 & 0.5027 & 0.9372 \\
1.639 & 1.507 & 1.754 \\
1.04 & 2.998 & 2.49 \\
1.868 & 1.066 & 0.8714 \\
2.39 & 0.1412 & 1.208 \\
2.238 & 0.641 & 2.586 \\
0.3766 & 1.194 & 1.844 \\
2.467 & 1.001 & 2.974 \\
0.07545 & 0.6888 & 0.6111 \\
1.243 & 2.808 & 2.482 \\
2.194 & 2.05 & 2.028 \\
2.344 & 2.886 & 0.7468 \\
1.102 & 1.314 & 1.427 \\
2.235 & 2.821 & 1.197 \\
2.677 & 0.0175 & 1.798 \\
0.7278 & 1.831 & 2.402 \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
w(1 : 25) & w(26 : 50) & w(51 : 75) \\
3 & 16 & 15 \\
17 & 3 & 10 \\
17 & 13 & 13 \\
8 & 6 & 18 \\
9 & 9 & 4 \\
12 & 17 & 8 \\
15 & 4 & 20 \\
15 & 7 & 9 \\
16 & 10 & 14 \\
8 & 7 & 19 \\
9 & 16 & 20 \\
20 & 20 & 14 \\
12 & 4 & 3 \\
17 & 5 & 1 \\
6 & 15 & 13 \\
13 & 8 & 12 \\
12 & 20 & 20 \\
20 & 20 & 15 \\
2 & 13 & 14 \\
11 & 18 & 11 \\
11 & 9 & 6 \\
2 & 13 & 20 \\
19 & 20 & 11 \\
18 & 12 & 1 \\
9 & 19 & 14 \\
\end{bmatrix}
\]

(A.25)

The code below:

\% AUTHOR: Jose Lopez NEU 2014
\% LICENSE: This code is licensed under the BSD License.
APPENDIX A. NEU MOMENTS V0.1

% This script implements a 75 variable 0-1 problem.

clear; clc;

n_vars = 75;
c = [0.287848249822753 2.24260098521491 2.24552687217988 1.62989829925533 1...0.01439694957216 ... 
2.4970098486158 1.6571653407787 2.87262935088606 2.678499957954 1... 
0.6951085138101 ... 
1.63920567897566 1.04004525376485 1.86840837197739 2.38987405592519 2... 
.2376247062195 ... 
0.376608694064459 2.46718202027706 0.0754515042855065 1.24328664277209 2... 
.19422240391881 ... 
2.34412200082789 1.10185774539411 2.23460356872502 2.67680156469332 0... 
.727810158839009 ... 
0.388790927512621 0.67520364638374 1.0500417570023 0.86125383948528 2... 
.78246402153828 ... 
0.153941270450387 1.77800026561916 0.488696700653197 2.51521730910190 0... 
.502682790633812 ... 
1.50660184622842 2.99798843707716 1.06622145380364 0.141233128646502 0... 
.640981755362397 ... 
1.19351737230812 1.0010045114088 0.688807595904593 2.8083605325062 2... 
.04956635409995 ... 
2.88634138716859 1.31391953289839 2.82100996326272 0.0175029737171126 1... 
.83092109020963 ... 
2.40322727089836 0.698944600240944 2.79740602776294 2.28978832600123 2... 
.47934861910804 ... 
1.7203907658146 2.377749923668 0.98712358717743 0.67038592877558 0... 
.937159026057761 ... 
1.75357043558567 2.48974242306457 0.871387474641804 1.20766317618937 2... 
.58617189529944 ... 
1.8442186506492 2.97356336498938 0.61109662555686 2.48162725483996 2... 
.07258485685832 ... 
0.746848410119424 1.4273568797086 1.19722568018038 1.79831474727424 2... 
.40156829741155];
w = [3 17 17 8 9 12 15 15 16 8 9 20 12 17 6 13 12 20 2 11 11 2 19 18 9 16 3 ... 
13 6 9 ... 
17 4 7 10 7 16 20 4 5 15 8 20 20 13 18 9 13 20 12 19 15 10 13 18 4 8 20 9 14 ... 
19 20 ...
APPENDIX A. NEU MOMENTS V0.1

14 3 1 13 12 20 15 14 11 6 20 11 1 14;  
b = 71;  

%% 
[M, basis_data] = get_mmatrix(1, n_vars);  
M = make_zero_one(M);  
[monomials, n_monomials] = get_monomials(M);  
%%  
cvx_clear;  
cvx_begin sdp  
cvx_solver sedumi  

variable y(n_monomials);  
M1 = assignm(y, get_map(M, monomials));  

y0 = y(1);  
%build objective  
first_order_monomials = basis_data.monomial{2};  
indices = get_index(monomials, first_order_monomials);  

x = y(indices);  
maximize(c'*x);  
subject to  
y0==1  
M1>=0  
b-w'*x>=0  
cvx_end  
cvx_optval %25.451  
x = round(x);  
format = ['x''=\{', repmat('%d', 1, n_vars), '\}']
fprintf(format, x);  
cost = w'*x %71  

%bintprog and LP relaxation give the same result:  

% [xm, objective, exit_code, output] =  
%bintprog(-c, [w'; zeros(n_vars-1, n_vars)], [b; zeros(n_vars-1, 1)]);  
% -objective %25.451  
% cost = w'*xm %71  
% norm(x-xm)  

123
% [xm2,objective2,exit_code2,output2] =
% linprog(-c,[w';zeros(n_vars-1,n_vars)],[b;zeros(n_vars-1,1)],[],...,
% [],zeros(n_vars,1),ones(n_vars,1));
% xm2 = round(xm2);
% objective2 = c'*xm2 %25.451
% cost2 = w'*xm2 %71
% norm(x-xm2)
gives \( x^T = [00000000000001000100100010101000100000101000000011010000000010], c^T x = 25.451, \) and \( w^T x = 71. \)

### A.4 Toolbox Functions

Copyright (c) 2016, Jose A. Lopez All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
A.4.1 get_mmatrix

% AUTHOR: Jose Lopez NEU 2014
% LICENSE: This code is licensed under the BSD License.
%
%M, BASIS_DATA = GET_MMATRIX(...
% DEGREE,N_VARS,VARARGIN) ... generates a (uint32)
% moment matrix, M, where each moment ... the matrix
% itself is stored in a 3D array (ROW,...
% DEGREE specifies ... the
% degree of the monomial basis used ... to index the moment matrix. ...
% N_VARS
% specifies the number of variables ... to use for each monomial. That ... is, N_VARS is
% the dimension of the variable in ... the problem.
%
% For example, in Lasserre’s notation...
% , the second order moment matrix...
% , M_2, for
% 3 variables would be generated by
%
% M2 = get_mmatrix(2,3)
% M2(:,:,1) will display the first ...
% column of M2
%
% VARARGIN is used to pass an ... optional basis vector. This is ... useful for
% generating reduced moment matrices ... to be used to implement the ... running
% intersection property.
%
% To summarize,
% INPUTS: DEGREE, a positive number.
% N_VARS, the number of ... variables in the problem. For ...
% example,
% VARARGIN, an optional ... basis vector to make a reduced ... moment matrix.
% For example, ...
% [0,0; 1,0].
%
% OUTPUTS: M, the 3D moment matrix ...
% containing (uint32) vectors of ... monomials.
% BASIS_DATA, a structure ... with fields MONOMIAL, a ...
% structure with
% DEGREE+1 elements, and ...
% BASIS, a basis vector.
%
% See also get_monomial_basis, ...
% expand_basis, ...
% rip_example_w_reweighted_heuristic....

function [M,basis_data] = ...
    get_mmatrix(degree,n_vars,...
    varargin)
if(isempty(varargin))
    basis_data = get_monomial_basis(...
        degree,n_vars);
else
    temp = uint32(varargin{1});
    max_deg = mabs(temp);
    if(degree>=max_deg)
APPENDIX A. NEU MOMENTS V0.1

basis_data = expand_basis(...
  degree,temp);
end
end
n_monomials = size(basis_data.basis...
  ,1);
M = zeros(n_monomials,n_vars,...
  n_monomials,'uint32');
M(:,:,1) = basis_data.basis;
for k = 2:n_monomials
  M(:,:,k) = bsxfun(@plus,M(:,:,1)...,
    M(k,:,1));
end

A.4.2 get_monomials

% AUTHOR: Jose Lopez NEU 2014
% LICENSE:This code is licensed ... under the BSD License.
% [MONOMIALS, N_MONOMIALS] = ...
% GET_MONOMIALS(MATRIX) gets all ... of the monomials in MATRIX. This is ... intended to be used when ... generating a map between monomials and optimization... variables which are usually ... stored in a vector.
% function [monomials, n_monomials] = ...
% get_monomials(matrix)
if(iscell(matrix))
  [~,dimension,~] = size(matrix);
  m_vec = reshape(permute(matrix...,
    [1,3,2]),[],dimension,1);
end
else
  [~,dimension,~] = size(matrix);
  m_vec = reshape(permute(matrix...,
    [1,3,2]),[],dimension,1);
end
[monomials,~,~] = unique(m_vec,'rows...');
temp = sortrows([monomials,sum(...monomials,2)],dimension+1);
monomials = temp(:,1:dimension);
[n_monomials,~] = size(monomials);

A.4.3 get_monomial_basis

% BASIS_DATA = GET_MONOMIAL_BASIS(...
% DEGREE,N_VARS) returns all the ... DEGREE-order monomials of polynomials with ... N_VARS variables. BASIS_DATA ... contains a cell array, BASIS_DATA.MONOMIAL, with ... the vectors grouped by DEGREE ... and also %NCHOOSEK(DEGREE + N_VARS, DEGREE) ... monomials,
% Requirements: DEGREE >= 1, N_VARS ... >= 1

126
APPENDIX A. NEU MOMENTS V0.1

% Examples: BASIS_DATA = ...
get_monomial_basis(2,3)
%
% BASIS_DATA =
% monomial: (3x1 cell)
% monomials: [10x3 double...]
%
% See also expand_basis.
%
function out = get_monomial_basis(...
degree,dimension)
out = expand_basis(degree,eye(...
dimension,'uint32'));

A.4.4 get_index_matrices

% AUTHOR: Jose Lopez NEU 2014
% LICENSE: This code is licensed ... under the BSD License.
%
% B = GET_INDEX_MATRICES(MONOMIALS,...
% MAP) accepts a matrix of ... MONOMIALS and a ...
% MAP matrix and returns a cell-... array, B, of so-called index ... matrices. This ...
% function was made when I was ...
% trying to write dual-programs ... manually;
% before I found out some solvers (... like SeDuMi) give you the dual ...
% answer
% when solving the primal problem. ... Anyway, it may still be useful ...
% so I am
% leaving it in the toolbox for now.
%
% Example:

A.4.5 get_raw_moments

% AUTHOR: Jose Lopez NEU 2014
% LICENSE: This code is licensed ... under the BSD License.
%
% VALUE = GET_RAW_MOMENTS(DATA, ... MONOMIALS) estimates the raw ...
% moments of
% DATA for the specified moment ...
% MONOMIALS.
%
% Example:
APPENDIX A. NEU MOMENTS V0.1

```
% X = rand(10000,1); % AUTHOR: Jose Lopez NEU 2014
% mons = (0:4)';
% gamma = get_raw_moments(... % LICENSE: This code is licensed ... under the BSD License.
X,mons);
% gives gamma = 1 0.50044 ...
0.33418 0.25084 0.2007
% which is close to the ...
% true moments for U[0,1]
% of 1 1/2 1/3 1/4 1/5
%
function value = get_raw_moments(...
data, monomials)
[n_monomials,n_vars] = size(...
omomials);
value = zeros(n_monomials,1);

% COMPUTE POWERS (ONLY ONCE)
max_deg = max(monomials(:));
lookup = cell(max_deg+1,1);
lookup{1} = ones(size(data));
lookup{2} = data;
for k = 3:max_deg+1
  lookup{k} = lookup{k-1}.*data;
end

% COMPUTE MOMENTS USING LOOKUP
for k = 1:n_monomials
  temp = lookup(monomials(k,1) + ...
           l1)(:,1);
  for j = 2:n_vars
    temp = temp.*lookup(...
      monomials(k,j) + l1{,:j}...)
    end
  value(k) = mean(temp);
end

A.4.6 get_map

% AUTHOR: Jose Lopez NEU 2014
% LICENSE: This code is licensed ...

% MAP = GET_MAP(MATRIX, MONOMIALS) ...
% returns a matrix, MAP, of ...
% scalars that
% correspond to an entry into a ...
% vector containing the ...
% optimization
% variables.
%
% For example,
% %M2 is a 3D matrix of ...
% monomials.
% [M2,basis_data] = ...
% get_mmatrix(2,2);  % %get all monomials in ...
% mmatrix
% [monomials,n_monomials...  % [monomials,n_monomials...
% ] = get_monomials(M);  % %map to CVX (or YALMIP...
% or whatever!) variable
% map = get_map(M2,...  % % declare CVX vector
% monomials);
% variable y(n_monomials...  % % re-assign M2, now it ...
% );  % is 2D and will contain moments ...
% % optimization
% M2 = assignm(y,map);  %
% %
% function map = get_map(matrix, ...
% monomials)
% [rows,dimension,cols] = size(matrix)...  %
% ;
% m_vec = reshape(permute(matrix...
% ,[1,3,2]),[],dimension,1);
```

128
APPENDIX A. NEU MOMENTS V0.1

[~,idx2] = ismember(m_vec,monomials,...
'rows');
map = reshape(idx2,rows,cols);

A.4.7 get_index

% AUTHOR: Jose Lopez NEU 2014
% LICENSE: This code is licensed ... under the BSD License.

% INDEX = GET_INDEX(MONOMIALS,...
% MONOMIAL) accepts a matrix of ... MONOMIAL and a
% MONOMIAL (can be a matrix) and ... returns an INDEX into a vector. ...
% This is used to associate a MONOMIAL to an... actual optimization variable.

% Example:
% [M,basis_data] = ...
% get_mmatrix(2,2);
% [monomials,n_monomials] = ...
% get_monomials(M);
% map = get_map(M,monomials);
% %declare CVX variable
% variable y(n_monomials);
% %suppose objective is f0 = ...
% x1*x2^2 and the corresponding
% %variable...
% y12 = y( get_index(...
% monomials,[1 2]) );

% function index = get_index(monomials...
% ,monomial)
[~,index] = ismember(monomial,...
monomials,'rows');

A.4.8 localize

% AUTHOR: Jose Lopez NEU 2014
% LICENSE: This code is licensed ... under the BSD License.

% L = LOCALIZE(M,MONOMIAL) makes ...
% localizing matrix by adding ...
% MONOMIAL to
% the elements of M.
% Example: if we want to make an ... order-1 localizing matrix for ...
% the % constraint f1(x) = 1 - x1^2 - x2^2... >=0 we would do the following
% M1 = get_mmatrix(1,2);
% L1_2 = localize(M1,[2 0]);
% L2_2 = localize(M1,[0 2]);
% ... do assignment
% then the constraint would be...
ug
% M1-L1_2-L2_2>=0

function L = localize(M,monomial)
    n_monomials = size(M,1);
    L = M;
    for k = 1:n_monomials
        L(:,:,k) = bsxfun(@plus,L(:,:,k)...
            ,uint32(monomial));
    end

A.4.9 expand_basis

% AUTHOR: Jose Lopez NEU 2014
% LICENSE: This code is licensed ... under the BSD License.

% BASIS_DATA = EXPAND_BASIS(DEGREE,...
% BASIS) accepts a (uint32) matrix
% BASIS of order-1 monomials expands... 
% it to have monomials of order 
% DEGREE. The use-case is when one ... 
% would like to construct a moment 
% matrix of selected variables. For ... 
% example, when one would like to 
% solve a moment problem using the ... 
% running intersection property.

% INPUTS: DEGREE - the degree of the... 
% desired basis 
% BASIS - a matrix of order... 
% -1 monomials 

% Example: basis_data = expand_basis... 
(2,eye(2,'uint32')) 
% basis_data.basis 
% ans = 
% 0 0 
% 1 0 
% 0 1 
% 2 0 
% 1 1 
% 0 2 

function out = expand_basis(degree,... 
  basis)
  basis = uint32(basis);
  max_degree = mabs(basis);
  n_iter = degree-max_degree;
  [~,n_vars] = size(basis);
  temp = find(sum(basis,2)==0);
  basis(temp,:) = [];
  %remove zero ... monomial (if included) for now
  last_monomial = out.monomial{i... 
    -1};
  n_monomials = size(last_monomial... 
    ,1);
  % this start business is so that... 
  % we don't have duplicates
  start = 1;
  for j = 1:n_vars-1
    start = [start,find(... 
      last_monomial(:,j),1,'... 
      last')+1];
  end
  temp = [];
  for k = 1:length(start)
    temp = [temp;
      last_monomial(start(k):... 
      n_monomials,:) + ... 
      repmat(out.monomial... 
      {2}(k,:),n_monomials... 
      -start(k)+1,1)];
  end
  out.monomial{i} = temp;
  out.basis = uint32(cell2mat(... 
    out.monomial(:)));

A.4.10 make_zero_one

% AUTHOR: Jose Lopez NEU 2014
% LICENSE:This code is licensed ... 
% under the BSD License.

% M = MAKE_ZERO_ONE(M, VARARGIN) ... 
% sets up a moment matrix for a ... 
% 0-1 program by 
% changing some/all the powers (of ... 
% variables specified in VARARGIN... 
% {1}) larger 
% than 1 to 1 and removing duplicate...
APPENDIX A. NEU MOMENTS V0.1

rows and columns, VARARGIN(1)=1...
.
%
% VARARGIN can be empty or a column ...
matrix of first order monomials ...
to be
% made 0-1.
%
% Example: M = get_mmatrix(2,2);
% %make variable with ...
monomial [1 0] be 0-1. i.e. if x...
is the
% %corresponding variable, ...
then x^2=x will be enforced in ...
M.
% M = make_zero_one(M,[1 ...
0]));
%
function M = make_zero_one(M,...
varargin)
rm_duplicates = 1;
if( isempty(varargin) )
 M(M>1) = 1;
else
 n_args = length(varargin);
basis = varargin{1};
if(n_args==2)
  rm_duplicates = varargin{2};
end
temp = cast(basis,'uint32');
n_mons = size(temp,1);
for j = 1:n_mons
  mon = temp(j,:);
  idx = find(mon,1);
  for k = 1:size(M,3)
    M( M(:,idx,k)>0, idx, k ...
    ) = 1;
  end
end
end

if(rm_duplicates)
 M = remove_duplicates(M);
end

A.4.11 assignm

% AUTHOR: Jose Lopez NEU 2014
% LICENSE:This code is licensed ...
under the BSD License.
%
% MATRIX = ASSIGNM(Y,MAP) accepts a ...
2D matrix AP, and a vector of
% optimization variable, Y, and does...
the assignment.
%
% Example:
%
% %construct moment matrix
% [M,basis_data] = ...
geet_mmatrix(2,2);
% %get all moments in ...
mmatrix
% [monomials,n_monomials] = ...
get_monomials(M);
% %map to CVX variable
% map = get_map(M,monomials)...
;
% %declare CVX variables
% variable y(n_monomials);
% % turn M2 into a 2D matrix...
of optimization variables
% M2 = assignm(y,map);
%
function matrix = assignm(y,map)
 matrix = reshape(y(map(:)),size(map)...
);

A.4.12 remove_duplicates

131
APPENDIX A. NEU MOMENTS V0.1

% AUTHOR: Jose Lopez NEU 2014
% LICENSE: This code is licensed ... under the BSD License.

% M = REMOVE_DUPLICATES(M) removes ... duplicate rows and columns
% from M. This would be needed if ... some of the variables are
% made 0-1. This function should not ... be needed outside of
% MAKE_ZERO_ONE but you never know.
% function M = remove_duplicates(M)
for k = 1:size(M,1)
    if(k>size(M,1)) % M is ... shrinking...
        break;
    end
    mon = M(k,:,1);
    [idx1,~] = ismember(M(:,:,1),mon...'rows');
    n_copies = sum(idx1);
    if(n_copies>1)
        idx2 = find(idx1,n_copies-1,'last');
        M(idx2,:,:) = [];
        M(:,:,idx2) = [];
    end
end

A.4.13 clean

function matrix = clean( matrix, ... tolerance )
    m_real = real(matrix);
    m_imag = imag(matrix);
    m_real( abs(m_real) < tolerance ) = ... 0;
    m_imag( abs(m_imag) < tolerance ) = ... example_1 %-0.492635

A.4.14 moment_tools_verification

% AUTHOR: Jose Lopez NEU 2014
% LICENSE: This code is licensed ... under the BSD License.

% This script runs the examples I've... coded to check if the answers ... are what
% we expect them to be. This is the ... closest thing I have to ... verification
% testing. It is meant to be run in ... the debugger, manually...
% clear;clc;
close all;
addpath('./examples/');

example_1 %-0.492635
example_1_dual %-0.492635
example_1_yalmip %-0.492635
example_2 %-11.4581
example_2_dual %-11.4581
example_3 %-0.037037
APPENDIX A. NEU MOMENTS V0.1

%%
example_3_dual

%%
example_5 %8

%%
exa...
Appendix B

MATLAB code for training HMMs

The code in this appendix is for the examples in Sections 4.4.1 and 4.4.2. Section B.4 contains utilities that were used to set-up the SDP programs. The same license used in Section A.4 applies here.

B.1 Code for aliased states example

```matlab
clear;clc;
close all;
addpath('./utils');

n_states = 3;
n_symbols = 6;
output = get_otp_constraints(...
    n_states,n_symbols);
output = lift_constraints(output);
fprintf('init done.
');

n_vars = length(output.vars) + ...
    length(output.quad_terms) + 1;

M = [0.00991281860808389;
    0.00266781427335545 0...
    0.00789317303667816 0...
    0.0153497297974276 0...
    0.0184439101219219 0...
    0.0266371005783163 0...
    0.0180148574386046;
    0.00485734247041623 0...
    0.180148574386046;
    0.0140994524304977 0...
    0.0288272575168445 0...
    0.0330675663038578 0...
    0.0478409452883136 0...
    0.0338600617682687;
    0.00575357113604688 0...
    0.174714486467424 0...
    0.0335692294996703 0...
    0.0408837107896584 0...
    0.059038199756267 0...
    0.0390786313986178;
```

134
APPENDIX B. MATLAB CODE FOR TRAINING HMMS

\[
\begin{align*}
&0.0082058485684807 \ 0... \\
&0.024997563039616 \ 0... \\
&0.0480101527858999 \ 0... \\
&0.0585119582309802 \ 0... \\
&0.0844965915709491 \ 0... \\
&0.0558271011255164; \\
&0.00719819711442413 \ 0... \\
&0.0194013721010077 \ 0... \\
&0.0411701412079088 \ 0... \\
&0.0453207135305999 \ 0... \\
&0.0656007390812034 \ 0... \\
&0.0494216857710488; \\
&0.00719819711442413 \ 0... \\
&0.0194013721010077 \ 0... \\
&0.0411701412079088 \ 0... \\
&0.0453207135305999 \ 0... \\
&0.0656007390812034 \ 0... \\
&0.0494216857710488; \\
&0.00719819711442413 \ 0... \\
&0.0194013721010077 \ 0... \\
&0.0411701412079088 \ 0... \\
&0.0453207135305999 \ 0... \\
&0.0656007390812034 \ 0... \\
&0.0494216857710488; \\
&0.00719819711442413 \ 0... \\
&0.0194013721010077 \ 0... \\
&0.0411701412079088 \ 0... \\
&0.0453207135305999 \ 0... \\
&0.0656007390812034 \ 0... \\
&0.0494216857710488; \\
&0.00719819711442413 \ 0... \\
&0.0194013721010077 \ 0... \\
&0.0411701412079088 \ 0... \\
&0.0453207135305999 \ 0... \\
&0.0656007390812034 \ 0... \\
&0.0494216857710488;
\end{align*}
\]

\[
P = \begin{bmatrix}
0.0444839336208605 \\
0.0890065852463041 \\
0.162552625778198 \\
0.195794791227003 \\
0.280049215321442 \\
0.228112848806192
\end{bmatrix};
\]

\[
\begin{align*}
I &= \text{eye}(n_{\text{vars}}); \\
W &= \text{zeros}(n_{\text{vars}}); \\
flag &= 1; \\
itration &= 1; \\
bound &= 1e-6;
\end{align*}
\]

\[
\text{while}(\ flag )
\]

\[
cvx\_clear; \\
cvx\_begin\ sdp\ quiet; \\
cvx\_solver\ sedumi; \\
cvx\_precision\ best;
\]

\[
\begin{align*}
\text{variable} \ Y(n_{\text{vars}}, n_{\text{vars}}) \ &... \\
\text{symmetric}; \\
\text{clear} \ f0 \ f2; \\
f0(1) &= M(1,1) - Y(31,85) - Y... \\
(38,85) &= Y(37,91) - Y... \\
(52,85) &= Y(50,91) - Y...
\end{align*}
\]
(51,99) - Y(70,93) - Y...
(69,99) - Y(80,99);
\[ f_0(10) = M(4,2) - Y(32,88) - Y... \]
(40,88) - Y(39,94) - Y...
(55,88) - Y(53,94) - Y...
(51,100) - Y(70,94) - Y...
(69,100) - Y(80,100);
\[ f_0(11) = M(5,2) - Y(32,89) - Y... \]
(40,89) - Y(39,95) - Y...
(55,89) - Y(53,95) - Y...
(51,101) - Y(70,95) - Y...
(69,101) - Y(80,101);
\[ f_0(12) = M(6,2) - Y(32,90) - Y... \]
(40,90) - Y(39,96) - Y...
(55,90) - Y(53,96) - Y...
(51,102) - Y(70,96) - Y...
(69,102) - Y(80,102);
\[ f_0(13) = M(1,3) - Y(33,85) - Y... \]
(42,85) - Y(41,91) - Y...
(58,85) - Y(56,91) - Y...
(54,97) - Y(72,91) - Y...
(71,97) - Y(81,97);
\[ f_0(14) = M(2,3) - Y(33,86) - Y... \]
(42,86) - Y(41,92) - Y...
(58,86) - Y(56,92) - Y...
(54,98) - Y(72,92) - Y...
(71,98) - Y(81,98);
\[ f_0(15) = M(3,3) - Y(33,87) - Y... \]
(42,87) - Y(41,93) - Y...
(58,87) - Y(56,93) - Y...
(54,99) - Y(72,93) - Y...
(71,99) - Y(81,99);
\[ f_0(16) = M(4,3) - Y(33,88) - Y... \]
(42,88) - Y(41,94) - Y...
(58,88) - Y(56,94) - Y...
(54,100) - Y(72,94) - Y...
(71,100) - Y(81,100);
\[ f_0(17) = M(5,3) - Y(33,89) - Y... \]
(42,89) - Y(41,95) - Y...
(58,89) - Y(56,95) - Y...
\[ f_0(18) = M(6,3) - Y(33,90) - Y... \]
(42,90) - Y(41,96) - Y...
(58,90) - Y(56,96) - Y...
(54,102) - Y(72,96) - Y...
(71,102) - Y(81,102);
\[ f_0(19) = M(1,4) - Y(34,85) - Y... \]
(44,85) - Y(43,91) - Y...
(61,85) - Y(59,91) - Y...
(57,97) - Y(74,91) - Y...
(73,97) - Y(82,97);
\[ f_0(20) = M(2,4) - Y(34,86) - Y... \]
(44,86) - Y(43,92) - Y...
(61,86) - Y(59,92) - Y...
(57,98) - Y(74,92) - Y...
(73,98) - Y(82,98);
\[ f_0(21) = M(3,4) - Y(34,87) - Y... \]
(44,87) - Y(43,93) - Y...
(61,87) - Y(59,93) - Y...
(57,99) - Y(74,93) - Y...
(73,99) - Y(82,99);
\[ f_0(22) = M(4,4) - Y(34,88) - Y... \]
(44,88) - Y(43,94) - Y...
(61,88) - Y(59,94) - Y...
(57,100) - Y(74,94) - Y...
(73,100) - Y(82,100);
\[ f_0(23) = M(5,4) - Y(34,89) - Y... \]
(44,89) - Y(43,95) - Y...
(61,89) - Y(59,95) - Y...
(57,101) - Y(74,95) - Y...
(73,101) - Y(82,101);
\[ f_0(24) = M(6,4) - Y(34,90) - Y... \]
(44,90) - Y(43,96) - Y...
(61,90) - Y(59,96) - Y...
(57,102) - Y(74,96) - Y...
(73,102) - Y(82,102);
\[ f_0(25) = M(1,5) - Y(35,85) - Y... \]
(46,85) - Y(45,91) - Y...
(64,85) - Y(62,91) - Y...
APPENDIX B. MATLAB CODE FOR TRAINING HMMS

\[
\begin{align*}
(60, 97) - Y(76, 91) - Y & \ldots \\
(75, 97) - Y(83, 97); \\
\mathbf{f}_0(26) = M(2, 5) - Y(35, 86) - Y & \ldots \\
(46, 86) - Y(45, 92) - Y & \ldots \\
(64, 86) - Y(62, 92) - Y & \ldots \\
(60, 98) - Y(76, 92) - Y & \ldots \\
(75, 98) - Y(83, 98); \\
\mathbf{f}_0(27) = M(3, 5) - Y(35, 87) - Y & \ldots \\
(46, 87) - Y(45, 93) - Y & \ldots \\
(64, 87) - Y(62, 93) - Y & \ldots \\
(60, 99) - Y(76, 93) - Y & \ldots \\
(75, 99) - Y(83, 99); \\
\mathbf{f}_0(28) = M(4, 5) - Y(35, 88) - Y & \ldots \\
(46, 88) - Y(45, 94) - Y & \ldots \\
(64, 88) - Y(62, 94) - Y & \ldots \\
(60, 100) - Y(76, 94) - Y & \ldots \\
(75, 100) - Y(83, 100); \\
\mathbf{f}_0(29) = M(5, 5) - Y(35, 89) - Y & \ldots \\
(46, 89) - Y(45, 95) - Y & \ldots \\
(64, 89) - Y(62, 95) - Y & \ldots \\
(60, 101) - Y(76, 95) - Y & \ldots \\
(75, 101) - Y(83, 101); \\
\mathbf{f}_0(30) = M(6, 5) - Y(35, 90) - Y & \ldots \\
(46, 90) - Y(45, 96) - Y & \ldots \\
(64, 90) - Y(62, 96) - Y & \ldots \\
(60, 102) - Y(76, 96) - Y & \ldots \\
(75, 102) - Y(83, 102); \\
\mathbf{f}_0(31) = M(1, 6) - Y(36, 85) - Y & \ldots \\
(48, 85) - Y(47, 91) - Y & \ldots \\
(66, 85) - Y(65, 91) - Y & \ldots \\
(63, 97) - Y(78, 91) - Y & \ldots \\
(77, 97) - Y(84, 97); \\
\mathbf{f}_0(32) = M(2, 6) - Y(36, 86) - Y & \ldots \\
(48, 86) - Y(47, 92) - Y & \ldots \\
(66, 86) - Y(65, 92) - Y & \ldots \\
(63, 98) - Y(78, 92) - Y & \ldots \\
(77, 98) - Y(84, 98); \\
\mathbf{f}_0(33) = M(3, 6) - Y(36, 87) - Y & \ldots \\
(48, 87) - Y(47, 93) - Y & \ldots \\
(66, 87) - Y(65, 93) - Y & \ldots \\
\mathbf{f}_1 = \text{trace}(Y*W); \\
\mathbf{f}_2(1) = P(1) - Y(10, 28) - Y & \ldots \\
(11, 29) - Y(12, 30); \\
\mathbf{f}_2(2) = P(2) - Y(13, 28) - Y & \ldots \\
(14, 29) - Y(15, 30); \\
\mathbf{f}_2(3) = P(3) - Y(16, 28) - Y & \ldots \\
(17, 29) - Y(18, 30); \\
\mathbf{f}_2(4) = P(4) - Y(19, 28) - Y & \ldots \\
(20, 29) - Y(21, 30); \\
\mathbf{f}_2(5) = P(5) - Y(22, 28) - Y & \ldots \\
(23, 29) - Y(24, 30); \\
\mathbf{f}_2(6) = P(6) - Y(25, 28) - Y & \ldots \\
(26, 29) - Y(27, 30); \\
\text{minimize( } \mathbf{f}_1 \text{ )} \\
\text{subject to} \\
\text{Y} \geq 0; \\
\text{% M constraints} \\
\mathbf{f}_0(:, \text{bound}) \geq 0; \\
\text{bound-}\mathbf{f}_0(:, \text{bound}) \geq 0; \\
\mathbf{f}_2(:, \text{bound}) \geq 0; \\
\end{align*}
\]
APPENDIX B. MATLAB CODE FOR TRAINING HMMS

bound-f2(:)>=0;

% unit and z constraints
Y(103,103) - 1 <= 1;
Y(103,31) - Y(1,10) == 0;
Y(103,32) - Y(1,13) == 0;
Y(103,33) - Y(1,16) == 0;
Y(103,34) - Y(1,19) == 0;
Y(103,35) - Y(1,22) == 0;
Y(103,36) - Y(1,25) == 0;
Y(103,37) - Y(2,10) == 0;
Y(103,38) - Y(4,11) == 0;
Y(103,39) - Y(2,13) == 0;
Y(103,40) - Y(4,14) == 0;
Y(103,41) - Y(2,16) == 0;
Y(103,42) - Y(4,17) == 0;
Y(103,43) - Y(2,19) == 0;
Y(103,44) - Y(4,20) == 0;
Y(103,45) - Y(2,22) == 0;
Y(103,46) - Y(4,23) == 0;
Y(103,47) - Y(2,25) == 0;
Y(103,48) - Y(4,26) == 0;
Y(103,49) - Y(3,10) == 0;
Y(103,50) - Y(5,11) == 0;
Y(103,51) - Y(3,13) == 0;
Y(103,52) - Y(7,12) == 0;
Y(103,53) - Y(5,14) == 0;
Y(103,54) - Y(3,16) == 0;
Y(103,55) - Y(7,15) == 0;
Y(103,56) - Y(5,17) == 0;
Y(103,57) - Y(3,19) == 0;
Y(103,58) - Y(7,18) == 0;
Y(103,59) - Y(5,20) == 0;
Y(103,60) - Y(3,22) == 0;
Y(103,61) - Y(7,21) == 0;
Y(103,62) - Y(5,23) == 0;
Y(103,63) - Y(3,25) == 0;
Y(103,64) - Y(7,24) == 0;
Y(103,65) - Y(5,26) == 0;
Y(103,66) - Y(7,27) == 0;
Y(103,67) - Y(6,11) == 0;
Y(103,68) - Y(8,12) == 0;
Y(103,69) - Y(6,14) == 0;
Y(103,70) - Y(8,15) == 0;
Y(103,71) - Y(6,17) == 0;
Y(103,72) - Y(8,18) == 0;
Y(103,73) - Y(6,20) == 0;
Y(103,74) - Y(8,21) == 0;
Y(103,75) - Y(6,23) == 0;
Y(103,76) - Y(8,24) == 0;
Y(103,77) - Y(6,26) == 0;
Y(103,78) - Y(8,27) == 0;
Y(103,79) - Y(9,12) == 0;
Y(103,80) - Y(9,15) == 0;
Y(103,81) - Y(9,18) == 0;
Y(103,82) - Y(9,21) == 0;
Y(103,83) - Y(9,24) == 0;
Y(103,84) - Y(9,27) == 0;
Y(103,85) - Y(10,28) == 0;
Y(103,86) - Y(13,28) == 0;
Y(103,87) - Y(16,28) == 0;
Y(103,88) - Y(19,28) == 0;
Y(103,89) - Y(22,28) == 0;
Y(103,90) - Y(25,28) == 0;
Y(103,91) - Y(11,29) == 0;
Y(103,92) - Y(14,29) == 0;
Y(103,93) - Y(17,29) == 0;
Y(103,94) - Y(20,29) == 0;
Y(103,95) - Y(23,29) == 0;
Y(103,96) - Y(26,29) == 0;
Y(103,97) - Y(12,30) == 0;
Y(103,98) - Y(15,30) == 0;
Y(103,99) - Y(18,30) == 0;
Y(103,100) - Y(21,30) == 0;
Y(103,101) - Y(24,30) == 0;
Y(103,102) - Y(27,30) == 0;

% probability constraints
Y(103,1) + Y(103,4) + Y(103,7) ... == 1;
Y(103,2) + Y(103,5) + Y(103,8) ... == 1;
Y(103,3) + Y(103,6) + Y(103,9) ... == 1;
Y(103,10) + Y(103,13) + Y...
(103,16) + Y(103,19) + Y...
(103,22) + Y(103,25) == 1;
Y(103,11) + Y(103,14) + Y...
(103,17) + Y(103,20) + Y...
(103,23) + Y(103,26) == 1;
Y(103,12) + Y(103,15) + Y...
(103,18) + Y(103,21) + Y...
(103,24) + Y(103,27) == 1;
Y(103,28) + Y(103,29) + Y...
(103,30) == 1;

Y(103,1:30)>>0;
cvx_end
ZZ = Y;
[UY,SY,VY] = svd(Y);
SY = diag(SY);
flag = SY(2)>5.00e-5;
sigma = SY(2)*0.9;
Z = UY(:,1)*SY(1)*VY(:,1)';
%Z = Y;
W = inv(Z + I*sigma);
fprintf('ITER %d: %0.5g %0.9g 
',iteration, sum(f0), SY(2));
format shortg;
SY(1:4)
temp = diag(Y);
A = [Y(103,1), Y(103,4), Y...
(103,7);

B.2 Code for state decoding example

A = bsxfun(@rdivide,A,sum(A,2));
B = bsxfun(@rdivide,B,sum(B,2));
PI = bsxfun(@rdivide,PI,sum(PI))... ;
[zz,g0,g2,otp,sp] = get_error(...
output, A, B, PI, M, P);
if(zz<0.002)
fprintf('feasible solution ... found.
');
flag = 0;
end
zz
iteration = iteration + 1;
end
fprintf('Rank-1 matrix found!\n');
APPENDIX B. MATLAB CODE FOR TRAINING HMMS

clear;clc;
close all;
addpath('./utils');

n_states = 2;
n_symbols = 5;
output = get_otp_constraints(...
   n_states,n_symbols);
output = lift_constraints(output);

fprintf('init done.
');

n_vars = length(output.vars) + ...
   length(output.quad_terms) + 1;

M = [0.046974083 0.067495607 0... 
   0.03430483 0.02761274 0.02761274;
   0.070158129 0.101986221 0... 
   0.05144865 0.0442035 0...
   0.0442035;
   0.034656214 0.050199806 0... 
   0.02525814 0.02094292 0...
   0.02094292;
   0.033845037 0.052275833 0... 
   0.02493149 0.02747382 0... 
   0.02747382;
   0.033845037 0.052275833 0... 
   0.02493149 0.02747382 0... 
   0.02747382];

P = [0.204;
   0.312;
   0.152;
   0.166;
   0.166];

I = eye(n_vars);
W = zeros(n_vars);
flag = 1;
iteration = 1;
bound = 1e-6;

while( flag )
   cvx_clear;
   cvx_begin sdp quiet;
   cvx_solver sedumi;
   cvx_precision best;
   variable Y(n_vars, n_vars) ...
      symmetric;
   f0 = 0 f2;
   f0(1) = M(1,1) - Y(17,37) - Y...
      (23,37) - Y(22,42) - Y...
      (32,42);
   f0(2) = M(2,1) - Y(17,38) - Y...
      (23,38) - Y(22,43) - Y...
      (32,43);
   f0(3) = M(3,1) - Y(17,39) - Y...
      (23,39) - Y(22,44) - Y...
      (32,44);
   f0(4) = M(4,1) - Y(17,40) - Y...
      (23,40) - Y(22,45) - Y...
      (32,45);
   f0(5) = M(5,1) - Y(17,41) - Y...
      (23,41) - Y(22,46) - Y...
      (32,46);
   f0(6) = M(1,2) - Y(18,37) - Y...
      (25,37) - Y(24,42) - Y...
      (33,42);
   f0(7) = M(2,2) - Y(18,38) - Y...
      (25,38) - Y(24,43) - Y...
      (33,43);
   f0(8) = M(3,2) - Y(18,39) - Y...
      (25,39) - Y(24,44) - Y...
      (33,44);
   f0(9) = M(4,2) - Y(18,40) - Y...
      (25,40) - Y(24,45) - Y...
      (33,45);
APPENDIX B. MATLAB CODE FOR TRAINING HMMS

\[
\begin{align*}
\text{f0}(10) &= M(5,2) - Y(18,41) - Y_{(25,41)} - Y_{(33,46)}; \\
\text{f0}(11) &= M(1,3) - Y(19,37) - Y_{(27,37)} - Y_{(34,42)}; \\
\text{f0}(12) &= M(2,3) - Y(19,38) - Y_{(27,38)} - Y_{(34,43)}; \\
\text{f0}(13) &= M(3,3) - Y(19,39) - Y_{(27,39)} - Y_{(34,44)}; \\
\text{f0}(14) &= M(4,3) - Y(19,40) - Y_{(27,40)} - Y_{(34,45)}; \\
\text{f0}(15) &= M(5,3) - Y(19,41) - Y_{(27,41)} - Y_{(34,46)}; \\
\text{f0}(16) &= M(1,4) - Y(20,37) - Y_{(29,37)} - Y_{(35,42)}; \\
\text{f0}(17) &= M(2,4) - Y(20,38) - Y_{(29,38)} - Y_{(35,43)}; \\
\text{f0}(18) &= M(3,4) - Y(20,39) - Y_{(29,39)} - Y_{(35,44)}; \\
\text{f0}(19) &= M(4,4) - Y(20,40) - Y_{(29,40)} - Y_{(35,45)}; \\
\text{f0}(20) &= M(5,4) - Y(20,41) - Y_{(29,41)} - Y_{(35,46)}; \\
\text{f0}(21) &= M(1,5) - Y(21,37) - Y_{(31,37)} - Y_{(36,42)}; \\
\text{f0}(22) &= M(2,5) - Y(21,38) - Y_{(31,38)} - Y_{(36,43)}; \\
\text{f0}(23) &= M(3,5) - Y(21,39) - Y_{(31,39)} - Y_{(36,44)}; \\
\text{f0}(24) &= M(4,5) - Y(21,40) - Y_{(31,40)} - Y_{(36,45)}; \\
\text{f0}(25) &= M(5,5) - Y(21,41) - Y_{(31,41)} - Y_{(36,46)}; \\
f1 &= \text{trace}(Y \ast W); \\
f2(1) &= P(1) - Y_{(5,15)} - Y_{(6,16)}; \\
f2(2) &= P(2) - Y_{(7,15)} - Y_{(8,16)}; \\
f2(3) &= P(3) - Y_{(9,15)} - Y_{(10,16)}; \\
f2(4) &= P(4) - Y_{(11,15)} - Y_{(12,16)}; \\
f2(5) &= P(5) - Y_{(13,15)} - Y_{(14,16)}; \\
\text{minimize}( f1 ) \\
\text{subject to} & \\
Y &\geq 0; \\
\% \text{M constraints} & \\
f0(:)-\text{bound}\geq 0; \\
\text{bound}-f0(:)&\geq 0; \\
f2(:)-\text{bound}\geq 0; \\
\text{bound}-f2(:)&\geq 0; \\
\% \text{unit and z constraints} & \\
Y_{(47,47)} &= 1 \leq 1; \\
Y_{(47,17)} &= Y_{(1,5)} = 0; \\
Y_{(47,18)} &= Y_{(1,7)} = 0; \\
Y_{(47,19)} &= Y_{(1,9)} = 0; \\
Y_{(47,20)} &= Y_{(1,11)} = 0; \\
Y_{(47,21)} &= Y_{(1,13)} = 0; \\
Y_{(47,22)} &= Y_{(2,5)} = 0; 
\end{align*}
\]
APPENDIX B. MATLAB CODE FOR TRAINING HMMS

Y(47,23) - Y(3,6) == 0;
Y(47,24) - Y(2,7) == 0;
Y(47,25) - Y(3,8) == 0;
Y(47,26) - Y(2,9) == 0;
Y(47,27) - Y(3,10) == 0;
Y(47,28) - Y(2,11) == 0;
Y(47,29) - Y(3,12) == 0;
Y(47,30) - Y(2,13) == 0;
Y(47,31) - Y(3,14) == 0;
Y(47,32) - Y(4,6) == 0;
Y(47,33) - Y(4,8) == 0;
Y(47,34) - Y(4,10) == 0;
Y(47,35) - Y(4,12) == 0;
Y(47,36) - Y(4,14) == 0;
Y(47,37) - Y(5,15) == 0;
Y(47,38) - Y(7,15) == 0;
Y(47,39) - Y(9,15) == 0;
Y(47,40) - Y(11,15) == 0;
Y(47,41) - Y(13,15) == 0;
Y(47,42) - Y(6,16) == 0;
Y(47,43) - Y(8,16) == 0;
Y(47,44) - Y(10,16) == 0;
Y(47,45) - Y(12,16) == 0;
Y(47,46) - Y(14,16) == 0;

% probability constraints
Y(47,1) + Y(47,3) == 1;
Y(47,2) + Y(47,4) == 1;
Y(47,5) + Y(47,7) + Y(47,9) + Y(47,11) + Y(47,13) == 1;
Y(47,6) + Y(47,8) + Y(47,10) + Y(47,12) + Y(47,14) == 1;
Y(47,15) + Y(47,16) == 1;
Y(47,1:16) >= 0;

% probability constraints
SY = diag(SY);
flag = SY(2)>5.00e-5;
sigma = SY(2)*0.9;

Z = UY(:,:,1)*SY(1)*VY(:,:,1)';
%Z = Y;
W = inv(Z + I*sigma);

fprintf('ITER %d: %0.5g %0.9g \n...','iteration, sum(f0), SY(2))... 

format shortg;

SY(1:4)
temp = diag(Y);
A = [Y(47,1), Y(47,3); Y(47,2), Y(47,4)];
B = [Y(47,5), Y(47,7), Y(47,9), ...
    Y(47,11), Y(47,13); Y(47,6), Y(47,8), Y(47,10), Y(47,12), Y(47,14)];
PI = [Y(47,15); Y(47,16)];
A = bsxfun(@rdivide,A,sum(A,2));
B = bsxfun(@rdivide,B,sum(B,2));
PI = bsxfun(@rdivide,PI,sum(PI))...

[zz,g0,g2,otp,sp] = get_error(...
    output, A, B, PI, M, P);

if(zz<0.002)
    fprintf('feasible solution ... found.\n');
    flag = 0;
end
zz
iteration = iteration + 1;
end

fprintf('Rank-1 matrix found!\n');
APPENDIX B. MATLAB CODE FOR TRAINING HMMS

B.3 Code for plots in state decoding example

clear; clc;
close all;
addpath('./utils');

% ground truth
A0 = [0.5722, 0.3446, 0.0832;
    0.1933, 0.6675, 0.1392;
    0.4729, 0.5271, 0];
B0 = [0.13, 0.27, 0.1, 0.25, 0.25;
    0.3, 0.4, 0.2, 0.05, 0.05;
    0.19, 0.17, 0.22, 0.21, 0.21];
PI0 = [0.5; 0.4; 0.1];

% second order models trained on the...
% same 50k points
% BW MODEL, lambda_4
A1 = [0.441982608883187 0...
    .558017391116813;
    0.290624164590851 0...
    .709375835409149];
B1 = [0.062886796008021 0...
    .220928667362008 0...
    .0717782733049974 0...
    .33299237133066 0...
    .311413891991908; 0.31605811499689 0...
    .384796613684397 0...
    .21166384393534 0...
    .0374522066346188 0...
    .0500285242417612];
PI1 = [0; 1];

% OURS, lambda_1
A2 = [0.620804610204204 0...
    .379195389795796;
    0.456437446562425 0.543562553437575... ];
B2 = [0.293911270881401 0...
    .37966336184234 0...
    .205525990803903 0...
    .0604496881738511 0...
    .0604496882985048; 0.140024344695969 0.26385449430023 ...
    0.113914103177804 0...
    .24110352889247 0...
    .24110352803735];
PI2 = [0.415733016781078;
    0.584266983218922];

% HSU, lambda_2
A3 = [0.829613069840737 0...
    .170386930159263;
    0.816823622564101 0.183176377435899... ];
B3 = [0.197515884643471 0...
    .369207924276521 0...
    .198341438566152 0...
    .178489539831203 0...
    .0564452126826543; 0.477127904977465 0.19349354721654 ...
    0.0886053627443747 0 0...
    .2407730775565061;]
PI3 = [0.829489681709409; 0.170510318290591];

rng(123);
APPENDIX B. MATLAB CODE FOR TRAINING HMMS

n_states = 3;
n_symbols = 5;
n_trials = 100000;
n_points = randi(99,n_trials,1)+1; %...
[2,100]
temp = zeros(n_trials,2);

results = zeros(n_trials,4);
for j = 1:n_trials

    [o, q0] = hmmgenerate(n_points(j...)
        ,[0,PI0'; zeros(n_states,1)...'
            ,A0],[zeros(1,n_symbols);B0...'
        ]); %
    [q1,~] = hmmdecode(o,[0,PI{1}';...
        zeros(2,1),A{1}],[zeros(1,...
            n_symbols);B{1}]);
    [q2,~] = hmmdecode(o,[0,PI{2}';...
        zeros(2,1),A{2}],[zeros(1,...
            n_symbols);B{2}]);
    [q3,~] = hmmdecode(o,[0,PI{3}';...
        zeros(2,1),A{3}],[zeros(1,...
            n_symbols);B{3}]);
    [q4,~] = hmmdecode(o,[0,PI0';...
        zeros(3,1),A0],[zeros(1,...
            n_symbols);B0]); %GT

    q0 = q0 - 1; %remove PI "state"
    q1 = q1(2:end,:);
    q2 = q2(2:end,:);
    q3 = q3(2:end,:);
    q4 = q4(2:end,:);

    [~,q1] = max(q1);
    [~,q2] = max(q2);
    [~,q3] = max(q3);
    [~,q4] = max(q4);

    if(any(q4==3))
        disp('');

    end

    results(j,:) = [sum(q0==q4)/...
        numel(q0),
        sum(q0==q1)/numel(q0),
        sum(q0==q2)/numel(q0),
        sum(q0==q3)/numel(q0),
    ];

    fprintf('%d,%d: %0.5g %0.5g %0...
        .5g %0.5g \n','j,n_points(j)...',results(j,:));
end

%%
close all;
n_bins = 30;
legends = {'\lambda_{GT}','\lambda_{...
    BW}','\lambda_1','\lambda_2'};
for k = 1:4
    subplot(2,2,k);
    h(k) = histogram(results(:,k));
    h(k).NumBins = n_bins;
    xlim([0,1]);
    ylim([0,20000]);
    title(legends{k});
    grid on;
    pause(0.3);
    if(k==3 || k==4)
        xlabel('Hamming Distance');
    end
    if(k==3 || k==4)
        h(k).FaceAlpha = 1;
    end
end

%%
close all;
figure;
hold on;
for k = 1:4
    h2(k) = histogram(results(:,k));
end
APPENDIX B. MATLAB CODE FOR TRAINING HMMS

```matlab
h2(k).Normalization = 'cdf';
xlim([0,1]);
ylim([0,1]);
grid on;
drawnow;
pause(0.3);
if(k==1)
    ylabel('CDF');
    xlabel('Hamming Distance');
end
h2(k).DisplayStyle = 'stairs';
end
h2(1).EdgeColor = 'k';
h2(2).EdgeColor = 'b';
h2(3).EdgeColor = 'r';
h2(4).EdgeColor = [0,0.5,0];
legend({'\lambda_{GT}','\lambda_{BW}...','\lambda_1','\lambda_2'});

%%
% stats
clc;
zz = [[mean(results(:,1)), std(...
    results(:,1))];
    [mean(results(:,2)), std(results...
    (:,2))];
    [mean(results(:,3)), std(results...
    (:,3))];
    [mean(results(:,4)), std(results...
    (:,4))]]

%B.4 Utility Functions

B.4.1 get_otp_constraints

% GET_OTP_EQUATIONS(N,K) accepts ...
% number of states, N, and number ...
% of
% symbols, K, and returns a struct ...
% with K^2 equations for the OTP ...
% matrix, etc.
%
% Example:
% output = ... % get_otp_constraints(2,3);
% output %
% function output = ...
% get_otp_constraints(n,k)

% n*(n+k+1) unknowns
% k^2+2*n+1 equations, including ...
% probability constraints
if( n*(n+k+1) > k^2+2*n+1 )
    fprintf('WARNING: you have more ...
    unknowns than constraints!\n...')
```
APPENDIX B. MATLAB CODE FOR TRAINING HMMS

```matlab
for l = 1:n
    temp(l) = A(l,:)*B(:,j);
end
output.quad_terms = [...
    output.quad_terms, ...
    children(sum(temp))]
    temp = temp.*PI.*B(:,i);
output.otp(i,j) = sum(temp);
output.otp_constraints(i,j) ... = M(i,j)-output.otp(i,j)... ];
end
output.quad_terms = unique(...
    output.quad_terms());
output.vars = [A(:);B(:);PI(:)];
output.prob_constraints = [sum(A,2);...
    sum(B,2); sum(PI)];

B.4.2 lift_constraints

% LIFT_CONSTRAINTS(OUTPUT) accepts ...
% an OUTPUT structure returned by
% GET_OTP_CONSTRAINTS and lifts the ...
% constraints for final ...
% implementation
% as an SDP.
%]
%
% Example:
% output = ...
% get_otp_constraints(2,3);
% lifted_output = ...
% lift_constraints(output);
%]
% This function will probably be ...
% called from inside ...
% get_otp_constraints
% but at this point I'm leaving it ...
% as a separate function.
```

146
function output = lift_constraints(...
    output)

% this is where the lifting function... end

n_vars = length(output.vars);

n_quads = length(output.quad_terms);

% rename A,B,PI to x for convenience

x = sym('x', [n_vars,1]);

% z is a lifting variable for the ... quadratic terms

% the first n_quads are not used but... we need the offset

% because the final lifting variable... will represent

% the first and second order terms [...] x(l:n_vars);

% z(n_vars+1:n_vars+n_cons); ...

unity_variable = 'unity_variable';

% variable. see optimization-over-... graph paper by Javad Lavaei

z = sym('z', [n_quads+n_vars,1]);

unity_var_idx = n_quads + n_vars + ...

1;

temp_quad_terms = output.quad_terms;

temp_otp = output.otp;

temp_sp = output.sp;

temp_prob_constraints = ...
    output.prob_constraints;

% replace A,B,PI by x in expressions

for k = 1:n_vars
    temp_quad_terms = subs(...
        temp_quad_terms, output.vars(...
        k), x(k));
    temp_otp = subs(temp_otp,...
        output.vars(k), x(k));
    temp_sp = subs(temp_sp,...
        output.vars(k), x(k));
    temp_prob_constraints = subs(...
        temp_prob_constraints,...
        output.vars(k), x(k));

% lift quad terms in num to ...

% intermediate variable z, for ...

% convenience, which has

% the offset indices into final ...

% lifting variable Y

quad_string = cell(n_quads,1);

for k = 1:n_quads
    temp_otp = subs(temp_otp,...
        temp_quad_terms(k), z(k+...
        n_vars));
    quad_string{k} = [sprintf('Y(%d...,
    ,%d) - ',unity_var_idx,k+...
        n_vars), char( ...
        temp_quad_terms(k) ), ' == ...
        0;'];
end

lifted_otp_constraints = cell(size(...
    output.otp_constraints));

lifted_otp = cell(size(output.otp));

lifted_otp_string = cell(size(...
    lifted_otp));

lifted_otp_cons_string = cell(size(...
    lifted_otp_constraints));

for k = 1:length(output.M(:))
    lifted_otp{k} = expand(...
        temp_otp(k));
    lifted_otp_constraints{k} = ...
        output.M(k) - lifted_otp...
        {k};
    lifted_otp_cons_string{k} = ... [sprintf('f0(%d) = ',k),... char(...
        temp_quad_terms(k) ), ' == ...
        0;'];
APPENDIX B. MATLAB CODE FOR TRAINING HMMS

```matlab
% lifted_otp_string{k} = char(lifted_otp{k});
end

lifted_otp_string = strjoin(lifted_otp_string, '
');
lifted_otp_cons_string = strjoin(lifted_otp_cons_string, '
');
quad_string = strjoin(quad_string, '
');

lifted_sp_constraints = cell(size(output.sp_constraints));
lifted_sp = cell(size(output.sp));
lifted_sp_string = cell(size(output.sp));
lifted_sp_cons_string = cell(size(lifted_sp_constraints));
for k = 1:length(output.sp_constraints)
    lifted_sp{k} = expand(temp_sp(k));
    lifted_sp_constraints{k} = output.P(k) - lifted_sp{k};
    lifted_sp_cons_string{k} = ['f2(', k, ') = ', char(lifted_sp_constraints{k}), ';'];
    lifted_sp_string{k} = char(lifted_sp{k});
end

lifted_sp_string = strjoin(lifted_sp_string, '
');
lifted_sp_cons_string = strjoin(lifted_sp_cons_string, '
');

output.lifted_otp = regexprep...
(output.lifted_otp_string, 'z(', 'Y(', 'z(');
output.lifted_otp = regexprep(output.lifted_otp_string, ')', ')');

% change M variables to have matrix notation
output.lifted_otp_constraints =...
regexprep(output.lifted_otp_constraints...
', 'M(', ')');

% change P variables to have matrix notation
output.lifted_sp = regexprep...
(output.lifted_sp_string, 'x(', 'P(');
output.lifted_sp_cons = regexprep...
(output.lifted_sp_cons_string, 'P(');

% replace sp expressions
output.lifted_sp = regexprep...
(output.lifted_sp_string, 'x(');
output.lifted_sp_constraints =...
regexprep(output.lifted_sp_constraints...
', 'x(');

% replace prob constraint expressions
temp_strings = cell(size(temp_prob_constraints));
for k = 1:length(temp_prob_constraints)
    temp_strings{k} = [char(temp_prob_constraints(k)), ' == 1;'];
end

lifted_string = strjoin(temp_strings...
', '
');

output.lifted_prob_constraints = ...
```

148
APPENDIX B. MATLAB CODE FOR TRAINING HMMS

% GET_OTP(OUTPUT, A, B, PI) accepts an OUTPUT structure produced
% by GET_OTP_CONSTRAINTS, a numerical triplet (A, B, PI), and returns the
% OTP and SP matrices.
function [otp,sp] = get_otp(output, A, B, PI)
    otp = double(subs(output.otp,...
                  output.vars,[A(:);B(:);PI(:)]));
    sp = double(subs(output.sp,...
                  output.vars,[A(:);B(:);PI(:)]));
end

B.4.5 get_counts

function [counts, prob_matrix, init_prob] = get_counts(sequence,...
    ,n_symbols)
    n_points = length(sequence);
    prob_matrix = zeros(n_symbols);
    counts = zeros(n_symbols);
    init_prob = zeros(n_symbols,1);
    for k = 2:n_points
        counts(sequence(k-1),sequence(k)) = counts(sequence(k-1),...
          sequence(k)) + 1;
    end
    for k = 1:n_symbols
        %prob_matrix(k,:) = counts(k,:)/...
          sum(counts(k,:));
        init_prob(k) = sum(sequence==k);
    end
    prob_matrix = counts/(n_points-1);
    init_prob = init_prob/n_points;
end