NEU-Moments v0.6: A MATLAB Toolbox for Moments-based Optimization

Jose A. Lopez
Robust Systems Lab
Northeastern University

June 3, 2015

Contents

Abstract

The purpose of this toolbox is to help bridge the gap between Jean Lasserre's 2001 paper [Lasserre, 2001b], on global polynomial optimization using moments, and actual moments-based problem solving using MATLAB for readers that are new to the area. 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 work directly with other modeling languages, like YALMIP, without any modifications. The efficacy of the toolbox is demonstrated by reproducing all the examples in [Lasserre, 2001b] and a few other (more advanced) examples.

1 License

This software is covered under the BSD License to protect the author in case it gets out into the wild.

2 Release Notes and Wishlist

2.1 Release Notes

The current `extract_solution` is not fully implemented. The current approach in `extract_solution` is to use a feasible moment matrix as a guide for a local search that (hopefully) results in the same, or better, objective value.

2.2 Wishlist (of upgrades)

1. A working `extract_solution` script.

2. A version of these tools for the Julia programming language.

3 Installation Instructions

To install this toolbox, simply put the `neu_moments` folder in a directory of your choice and add the path in MATLAB. It is assumed that the user has: MATLAB, solvers (like SeDuMi, SDPT3, CSDP, etc.), and modeling tools (like CVX, YALMIP, etc.) installed on their computer. After installation, the user should step through `moment_tools_verification` to check that their solver produces the same answers (modulo numerical differences) as in the comments in `moment_tools_verification`.

4 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) [Lasserre, 2001b]. This toolbox was made to help users take such a mathematical description and write MATLAB code that will get them the (numerical) answer they are 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.

Much of the next section is from [Lopez et al., 2015, Lasserre, 2001b].

4.1 Preliminaries

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 [Lasserre, 2009]. However, its wider application has been held back because, in its fullest generality, the GMP is intractable [Lasserre, 2008]. On the other hand, recent advances in algebraic geometry have rendered the GMP tractable when the problem instance is described using polynomials [Lasserre, 2001b, Lasserre, 2009].

Formally, the GMP is given by:

$$\begin{array}{rl} \text{GMP:} & \rho_{\text{mom}} = \sup\limits_{\mu \in \mathcal{M}(\mathbb{K})_+} \displaystyle\int_{\mathbb{K}} f_0 d\mu \\ \text{subject to} & \int_{\mathbb{K}} f_j d\mu \leqq \gamma_j,\ j = 1, \ldots, m \end{array} \tag{4.1}$$

where $\mathbb{K}$ is a Borel subset of $\mathbb{R}^n$, $\mathcal{M}(\mathbb{K})$ is the space of finite signed Borel measures on $\mathbb{K}$, $\mathcal{M}(\mathbb{K})_+$ is the positive cone containing finite Borel measures

μ on $\mathbb{K}$, $\{\gamma_j \,|\, j = 1, \ldots, m\}$ is a set of real numbers, and $f_j : \mathbb{K} \to \mathbb{R}$ are integrable with respect to all $\mu \in \mathcal{M}(\mathbb{K})_+$ for every $j = 0, \ldots, m$. The $\lesseqqgtr$ stands for either an inequality or equality constraint. (If this sounds scary, that is because it is but keep reading anyway – when we get to numerically solving stuff we will not even think about Borel sets.)

Lasserre [Lasserre, 2001b, Lasserre, 2009, Lasserre, 2008, Lasserre, 2009] showed that semidefinite programming (SDP) can be used to obtain an (often) finite sequence of relaxations, to the GMP, 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.

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

$$\nu(x) = \big(1, x_1, x_2, \ldots, x_n, x_1^2, x_1x_2, \ldots, x_1x_n, x_2x_3, \ldots, x_n^2, \\ \ldots, x_1^d, \ldots, , x_n^d\big)^T \tag{4.2}$$

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_α, let $L_y(p)$ denote the linear map

$$L_y(p) = \sum_\alpha p_\alpha y_\alpha \tag{4.3}$$

which associates p with the moment variables, y_α. Essentially what we do is linearize a non-linear problem to get the optimal objective; the hard part is getting the optimizers once we get the optimal value [Henrion and Lasserre, 2005, Lasserre, 2009]. This idea is not new – LP programming has been doing it for a while (by adding redundant constraints) but the advantage that SDP brought is faster convergence. There are a couple of sections in [Lasserre, 2009] that talk about the LP approach, for the interested reader.

4.1.1 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

$$M_d(y)(\alpha, \beta) = y_{\alpha+\beta},\ \alpha, \beta \in \mathbb{N}^n,\ |\alpha|, |\beta| \leq d \tag{4.4}$$

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 (α, β) 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 illustration.

$$M_2(y) = \begin{bmatrix} y_{00} & y_{10} & y_{01} & y_{20} & y_{11} & y_{02} \\ y_{10} & y_{20} & y_{11} & y_{30} & y_{21} & y_{12} \\ y_{01} & y_{11} & y_{02} & y_{21} & y_{12} & y_{03} \\ y_{20} & y_{30} & y_{21} & y_{40} & y_{31} & y_{22} \\ y_{11} & y_{21} & y_{12} & y_{31} & y_{22} & y_{13} \\ y_{02} & y_{12} & y_{03} & y_{22} & y_{13} & y_{04} \end{bmatrix} \tag{4.5}$$

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

The classical problem of moments is concerned with determining if, for a given moment sequence $\{y_\alpha\}$, there is a measure μ so that $y_\alpha = \int x^\alpha d\mu$ for each α. 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 [Lasserre, 2009] and references therein.

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

$$M_d(gy) = L_y(g(x)\nu(x)\nu(x)^T) \tag{4.6}$$

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

$$M_2(gy) = a \begin{bmatrix} y_{00} & y_{10} & y_{01} & y_{20} & y_{11} & y_{02} \\ y_{10} & y_{20} & y_{11} & y_{30} & y_{21} & y_{12} \\ y_{01} & y_{11} & y_{02} & y_{21} & y_{12} & y_{03} \\ y_{20} & y_{30} & y_{21} & y_{40} & y_{31} & y_{22} \\ y_{11} & y_{21} & y_{12} & y_{31} & y_{22} & y_{13} \\ y_{02} & y_{12} & y_{03} & y_{22} & y_{13} & y_{04} \end{bmatrix} -$$

$$\begin{bmatrix} y_{20} & y_{30} & y_{21} & y_{40} & y_{31} & y_{22} \\ y_{30} & y_{40} & y_{31} & y_{50} & y_{41} & y_{32} \\ y_{21} & y_{31} & y_{22} & y_{41} & y_{32} & y_{23} \\ y_{40} & y_{50} & y_{41} & y_{60} & y_{51} & y_{42} \\ y_{31} & y_{41} & y_{32} & y_{51} & y_{42} & y_{33} \\ y_{22} & y_{32} & y_{23} & y_{42} & y_{33} & y_{24} \end{bmatrix} - \begin{bmatrix} y_{02} & y_{12} & y_{03} & y_{22} & y_{13} & y_{04} \\ y_{12} & y_{22} & y_{13} & y_{32} & y_{23} & y_{14} \\ y_{03} & y_{13} & y_{04} & y_{23} & y_{14} & y_{05} \\ y_{22} & y_{32} & y_{23} & y_{42} & y_{33} & y_{24} \\ y_{13} & y_{23} & y_{14} & y_{33} & y_{24} & y_{15} \\ y_{04} & y_{14} & y_{05} & y_{24} & y_{15} & y_{06} \end{bmatrix} \quad (4.7)$$

Localizing matrices are used to specify support constraints on μ.

4.1.2 Using the GMP for polynomial optimization

These problem statements follow Lasserre's notation in [Lasserre, 2001b] and are included because we will refer to $\mathbb{P}$ and $\mathbb{Q}$ in the examples.

Given a real-valued polynomial $p(x) : \mathbb{R}^n \to \mathbb{R}$, with real coefficients, the symbol $\mathbb{P}$ denotes the (primal) unconstrained optimization problem

$$\mathbb{P} \mapsto p^* = \min_{x \in \mathbb{R}^n} p(x) \quad (4.8)$$

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

$$\mathbb{P}_K \mapsto p_K^* = \min_{x \in K} p(x) \quad (4.9)$$

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

From[1] [Lasserre, 2001b]:

3. Unconstrained global optimization. *Let $p(x) : \mathbb{R}^n \to \mathbb{R}$ be a real-valued polynomial of degree $2m$ with coefficient vector $p \in \mathbb{R}^{s(2m)}$. Since we*

[1]The equation numbers are different.

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 positive semidefinite program):

$$\mathbb{Q} \mapsto \begin{cases} \inf_y \sum_\alpha p_\alpha y_\alpha \\ \\ M_m(y) \geq 0 \end{cases} \tag{4.10}$$

or equivalently,

$$\mathbb{Q} \mapsto \begin{cases} \inf_y \sum_\alpha p_\alpha y_\alpha \\ \\ \sum_{\alpha \neq 0} y_\alpha B_\alpha \geq -B_0 \end{cases} \tag{4.11}$$

where the matrices B_i are easily understood from the definition of $M_m(y)$. The dual problem $\mathbb{Q}^*$ of $\mathbb{Q}$ is the convex LMI problem defined by

$$\mathbb{Q}^* \mapsto \begin{cases} \sup_X \langle X, -B_0 \rangle \; (= -X(1,1)) \\ \\ \langle X, B_\alpha \rangle = p_\alpha, \quad \alpha \neq 0 \\ X \geq 0 \end{cases} \tag{4.12}$$

Also, the constrained version of these problems:

$$\mathbb{Q}_K^N \mapsto \begin{cases} \inf_y \sum_\alpha p_\alpha y_\alpha \\ \\ M_N(y) \geq 0 \\ M_{N-\tilde{w}_i}(g_i y) \geq 0, \; i = 1, \ldots, r \end{cases} \tag{4.13}$$

$$(\mathbb{Q}_K^N)^* \mapsto \begin{cases} \sup_{X, Z_i} -X(1,1) - \sum_{i=1}^r g_i(0) Z_i(1,1) \\ \langle X, B_\alpha \rangle + \sum_{i=1}^r \langle Z_i, C_{i\alpha} \rangle = p_\alpha, \; \alpha \neq 0 \\ X, Z_i \geq 0, \; i = 1, \ldots, r \end{cases} \tag{4.14}$$

At this point we are hopefully ready to do talk about code.

5 NEU-Moments Utilities

The MATLAB 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 and form a solid foundation on which users can build their own tools. 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. In practice, usage errors should become easy to spot and so I hope that the trade-off for simpler (faster, easier to maintain, modify, etc.) code will be worth any short-term inconveniences.

5.1 get_mmatrix

```
% [M, BASIS_DATA] = GET_MMATRIX(DEGREE,N_VARS,VARARGIN) generates
% a (uint32) moment matrix, M, where each moment is represented
% by a vector and the matrix itself is stored in a 3D array
% (ROW,N_VARS,COL). 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,
%                 f(x)=x1*x2^2 has two variables and degree is 3.
%         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.
%
```

5.2 get_monomials

```
% [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.
```

5.3 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 BASIS_DATA.MONOMIALS which contains an array with
% all NCHOOSEK(DEGREE + N_VARS, DEGREE) monomials,
%
% Requirements: DEGREE >= 1, N_VARS >= 1
%
% Examples: BASIS_DATA = get_monomial_basis(2,3)
%
%        BASIS_DATA =
%             monomial: {3x1 cell}
%            monomials: [10x3 double]
%
% See also expand_basis.
%
```

5.4 get_map

```
% 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] = get_monomials(M);
%              %map to CVX (or YALMIP or whatever!) variable
%              map = get_map(M2,monomials);
%              %declare CVX vector
%              variable y(n_monomials);
%              %re-assign M2, now it is 2D and will contain moments after
%              %optimization
%              M2 = assignm(y,map);
%
```

5.5 get_index

```
% INDEX = GET_INDEX(MONOMIALS,MONOMIAL) accepts a matrix of MONOMIALS 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]) );
%
```

5.6 get_index_matrices

```
% 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:
%  %construct moment matrix
%  M2 = get_mmatrix(2,2);
%  %get all monomials in mmatrix
%  [monomials,~] = get_monomials(M2);
%  %get map to optimization variables
%  map = get_map(M2,monomials);
%  %get index matrices, B is sparse
%  B = get_index_matrices(monomials,map);
%  B{1}
%
% ans =
%
%    (1,1)        1
```

5.7 assignm

```
% 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] = get_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);
%
```

5.8 localize

```
% 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 specified as
%       M1-L1_2-L2_2>=0
%
```

5.9 make_zero_one

```
% 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 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]);
```

5.10 expand_basis

```
% 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
%
```

5.11 remove_duplicates

```
% 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.
```

5.12 get_raw_moments

```
% VALUE = GET_RAW_MOMENTS(DATA, MONOMIALS) estimates the raw moments
% of DATA for the specified moment MONOMIALS.
%
% Example:
%           X = rand(10000,1);
%           mons = (0:4)';
%           gamma = get_raw_moments(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
```

5.13 moment_tools_verification

`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.

6 Examples

All of these examples can be found in `../examples`. Some of the examples solve the dual problem ($\mathbb{Q}^*$) explicitly. This was done before learning that some solvers solve both the primal and dual problems at the same time and also provide the dual variables.

6.1 Example 1

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

$$(x_1, x_2) \mapsto (x_1^2 + 1)^2 + (x_2^2 + 1)^2 + (x_1 + x_2 + 1)^2 \tag{6.1}$$

Expanding p gives

$$p(x) = x_1^4 + x_2^4 + 3x_1^2 + 3x_2^2 + 2x_1x_2 + 2x_1 + 2x_2 + 3 \tag{6.2}$$

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

$$\sum_\alpha p_\alpha y_\alpha = y_{40} + y_{04} + 3y_{20} + 3y_{02} + 2y_{11} + 2y_{10} + 2y_{01} \tag{6.3}$$

We can solve $\mathbb{Q}^*$ using the code

```
% 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

variable 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
trace(X*B{10})==0; %p30
trace(X*B{14})==0; %p31
trace(X*B{15})==1; %p40
X>=0;
cvx_end
[z10,z01]
%cvx_optval = -0.492635 matches paper
```

which yields `cvx_optval = -0.492635` and

$$X = \begin{bmatrix} 0.493 & 1 & 1 & -0.0581 & -0.0017 & -0.0581 \\ 1 & 3.12 & 1 & 0 & 0.197 & -0.197 \\ 1 & 1 & 3.12 & -0.197 & 0.197 & 0 \\ -0.0581 & 0 & -0.197 & 1 & 0 & -0.826 \\ -0.0017 & 0.197 & 0.197 & 0 & 1.65 & 0 \\ -0.0581 & -0.197 & 0 & -0.826 & 0 & 1 \end{bmatrix} \tag{6.4}$$

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

We can solve $\mathbb{Q}$ 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]);
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,

```
% 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
```

6.2 Example 2

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

$$(x_1, x_2) \mapsto (x_1^2 + 1)^2 + (x_2^2 + 1)^2 - 2(x_1 + x_2 + 1)^2 \tag{6.5}$$

Expanding p gives

$$p(x) = x_1^4 + x_2^4 - 4x_1x_2 - 4x_1 - 4x_2 \tag{6.6}$$

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

$$\sum_\alpha p_\alpha y_\alpha = y_{40} - 4y_{11} - 4y_{10} + y_{04} - 4y_{01} \tag{6.7}$$

Solving $\mathbb{Q}$ 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;

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 $\mathbb{Q}^*$ instead, we run

```
% 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

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
%(cvx_optval): -11.4581
```

which (also) gives `cvx_optval = -11.4581` and

$$X^* = \begin{bmatrix} 11.5 & -2 & -2 & -1.2 & -1.11 & -1.2 \\ -2 & 2.4 & -0.886 & 0 & -0.00755 & 0.00755 \\ -2 & -0.886 & 2.4 & 0.00755 & -0.00755 & 0 \\ -1.2 & 0 & 0.00755 & 1 & 0 & -0.323 \\ -1.11 & -0.00755 & -0.00755 & 0 & 0.646 & 0 \\ -1.2 & 0.00755 & 0 & -0.323 & 0 & 1 \end{bmatrix} \tag{6.8}$$

6.3 Example 3

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

$$x \mapsto p(x) = x_1^2 x_2^2 \left(x_1^2 + x_2^2 - 1\right) \tag{6.9}$$

Expanding p gives

$$p(x) = x_1^4 x_2^2 + x_1^2 x_2^4 - x_1^2 x_2^2 \tag{6.10}$$

from here we can see that, in moment (y_α) notation, we have

$$\sum_\alpha p_\alpha y_\alpha = y_{42} + y_{24} - y_{22} \tag{6.11}$$

Solving $\mathbb{Q}_1^3$ 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;

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);
y20 = y(indices.y20);
y02 = y(indices.y02);
minimize(y42+y24-y22)
subject to
M3>=0
L>=0
y00 == 1;
cvx_end
%(cvx_optval): -0.037037 matches
```

gives `cvx_optval = -0.037037` and $x_1^2 = x_2^2 = 0.33339$ which is close to the true values of $p^* = -1/27$ and $x_1^2 = x_2^2 = 1/3$.

The code for the dual is:

```
% 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;

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
```

6.4 Example 5

Lasserre 2001, pg. 811, Example 5 Let $p(x) : \mathbb{R}^2 \to \mathbb{R}$ be the concave polynomial $x \mapsto p(x) = -(x_1 - 1)^2 - (x_1 - x_2)^2 - (x_2 - 3)^2$ and

$$K = \{(x_1, x_2) \in \mathbb{R}^2 | 1 - (x_1 - 1)^2 \geq 0; 1 - (x_1 - x_2)^2 \geq 0; 1 - (x_2 - 3)^2 \geq 0\} \tag{6.12}$$

The code below solves $\mathbb{Q}_K^2$

```
% 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);
%get localizing matrix
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
M2>=0;
-L20+2*L10>=0;
-L20+2*L11-L02+M1>=0;
-L02+6*L01-8*M1>=0;
y00 == 1;
cvx_end
[y10,y01]
%(cvx_optval): 8 true optimal = 8
```

and gives `cvx_optval = 8` at $(x_1, x_2) = (1.5135, 2.1741)$.

6.5 Problem 2.2

Lasserre 2001, pg. 812

```
% 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_mmatrix(2,6);
[monomials,n_monomials] = get_monomials(M);
maps.M2 = get_map(M,monomials);
%form localizing matrix L
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);
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);
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
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
```

6.6 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
%
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));
x9 = y(indices.x(9));
x10 = y(indices.x(10));
x1_2 = y(indices.x2(1));
x2_2 = y(indices.x2(2));
x3_2 = y(indices.x2(3));
x4_2 = y(indices.x2(4));
x5_2 = y(indices.x2(5));
x6_2 = y(indices.x2(6));
x7_2 = y(indices.x2(7));
x8_2 = y(indices.x2(8));
x9_2 = y(indices.x2(9));
x10_2 = y(indices.x2(10));

objective = -50*x1_2+48*x1-50*x2_2+42*x2-50*x3_2+48*x3-50*x4_2+45*x4-50*x5_2+44*x5-.
50*x6_2+41*x6-50*x7_2+47*x7-50*x8_2+42*x8-50*x9_2+45*x9-50*x10_2+46*x10;

minimize(objective)
subject to
y(1)==1;
M2>=0;
for k = 1:10
    L{k}>=0;
    M1 - L{k}>=0;
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)
```

6.7 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);
L = localize(M,[0 0 0 0 0 0 0 0 1 0]);
maps.L0000000010 = get_map(L,monomials);
L = localize(M,[0 0 0 0 0 0 0 0 0 1]);
maps.L0000000001 = 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]);
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.y0000000011 = get_index(monomials,[0 0 0 0 0 0 0 0 1 1]);
indices.y1010000000 = 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.y0010100000 = get_index(monomials,[0 0 1 0 1 0 0 0 0 0]);
indices.y0001010000 = get_index(monomials,[0 0 0 1 0 1 0 0 0 0]);
indices.y0000101000 = get_index(monomials,[0 0 0 0 1 0 1 0 0 0]);
indices.y0000010100 = get_index(monomials,[0 0 0 0 0 1 0 1 0 0]);
indices.y0000001010 = get_index(monomials,[0 0 0 0 0 0 1 0 1 0]);
indices.y0000000101 = get_index(monomials,[0 0 0 0 0 0 0 1 0 1]);
indices.y1000001000 = get_index(monomials,[1 0 0 0 0 0 1 0 0 0]);
indices.y1000000010 = get_index(monomials,[1 0 0 0 0 0 0 0 1 0]);
indices.y1000000001 = get_index(monomials,[1 0 0 0 0 0 0 0 0 1]);
indices.y0100000001 = get_index(monomials,[0 1 0 0 0 0 0 0 0 1]);
indices.y0001001000 = get_index(monomials,[0 0 0 1 0 0 1 0 0 0]);
%%
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
y1100000000 = y(indices.y1100000000);
y0110000000 = y(indices.y0110000000);
y0011000000 = y(indices.y0011000000);
y0001100000 = y(indices.y0001100000);
y0000110000 = y(indices.y0000110000);
y0000011000 = y(indices.y0000011000);
y0000001100 = y(indices.y0000001100);
y0000000110 = y(indices.y0000000110);
y0000000011 = y(indices.y0000000011);
y1010000000 = y(indices.y1010000000);
```

```
y0101000000 = y(indices.y0101000000);
y0010100000 = y(indices.y0010100000);
y0001010000 = y(indices.y0001010000);
y0000101000 = y(indices.y0000101000);
y0000010100 = y(indices.y0000010100);
y0000001010 = y(indices.y0000001010);
y0000000101 = y(indices.y0000000101);
y1000001000 = y(indices.y1000001000);
y1000000010 = y(indices.y1000000010);
y1000000001 = y(indices.y1000000001);
y0100000001 = y(indices.y0100000001);
y0001001000 = y(indices.y0001001000);

obj = y1100000000+y0110000000+y0011000000+y0001100000+y0000110000+...
    y0000011000+y0000001100+y0000000110+y0000000011+...
    y1010000000+y0101000000+y0010100000+y0001010000+y0000101000+...
    y0000010100+y0000001010+y0000000101+...
    y1000001000+y1000000010+y1000000001+y0100000001+y0001001000;

maximize(obj)
subject to
y(1)==1
M2>=0
L1000000000+L0100000000+L0010000000+L0001000000+...
L0000100000+L0000010000+L0000001000+L0000000100+L0000000010+...
+L0000000001-M1==0
L1000000000>=0
L0100000000>=0
L0010000000>=0
L0001000000>=0
L0000100000>=0
L0000010000>=0
L0000001000>=0
L0000000100>=0
L0000000010>=0
L0000000001>=0
cvx_end
%(cvx_optval): +0.375
```

6.8 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]);
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
objective = -25*y200000+100*y100000-y020000+4*y010000-y002000+2*y001000-...
y000200+8*y000100-y000020+2*y000010-y000002+8*y000001-138;
minimize(objective);

subject to
y(1)==1
M2>=0
L002000-6*L001000+L000100+5*M1>=0
L000020-6*L000010+L000001+5*M1>=0
2*M1-L100000+3*L010000>=0
2*M1+L100000-L010000>=0
6*M1-L100000-L010000>=0
L100000+L010000-2*M1>=0
L100000>=0
L010000>=0
L001000-M1>=0
5*M1-L001000>=0
L000100>=0
6*M1-L000100>=0
L000010-M1>=0
5*M1-L000010>=0
L000001>=0
10*M1-L000001>=0
cvx_end
cvx_optval
```

```
%answer: -310, M1 is rank-1, so we can read-off optimizers
M1(:,1)
```

6.9 Problem 3.4

Lasserre 2001, pg. 813

$$\begin{cases} \min_x p(x) = -2x_1 + x_2 - x_3 \\ x_1 + x_2 + x_3 \leq 4 \\ x_1 \leq 2 \\ x_3 \leq 3 \\ 3x_2 + x_3 \leq 6 \\ x_1 \geq 0 \\ x_2 \geq 0 \\ x_3 \geq 0 \\ x^T B^T Bx - 2r^T Bx + r^T r - 0.25(b-v)^T(b-v) \end{cases}$$

with

$$B = \begin{bmatrix} 0 & 0 & 1 \\ 0 & -1 & 0 \\ -2 & 1 & -1 \end{bmatrix}, b = \begin{bmatrix} 3 \\ 0 \\ -4 \end{bmatrix}, v = \begin{bmatrix} 0 \\ -1 \\ -6 \end{bmatrix}, r = \begin{bmatrix} 2 \\ -0.5 \\ -5 \end{bmatrix} \tag{6.13}$$

The code below solves $\mathbb{Q}_K^4$

```
% 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]);
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);
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;-5];

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 $p^* = 4$.

6.10 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
```

6.11 Problem 4.7

Lasserre 2001, pg. 813

$$\begin{cases} \min_x p(x) = -12x_1 - 7x_2 + x_2^2 \\ \quad -2x_1^4 + 2 - x_2 = 0 \\ \quad 0 \leq x_1 \leq 3 \\ \quad 0 \leq x_2 \leq 4 \end{cases}$$

The following code solves $\mathbb{Q}_K^5$

```
% 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)

%form localizing matrices
M = get_mmatrix(2,2);
maps.M2 = get_map(M,monomials);
L = localize(M,[4,0]);
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
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 $p^* = -16.7389$.

6.12 A small 0-1 problem

Consider the problem

$$\left\{ \begin{array}{c} \max\limits_x c^T x \\ w^T b \leq b \\ x_i \in \{0,1\},\ \forall i = 1,\ldots,5 \end{array} \right.$$

where

$$c = \begin{bmatrix} 3 \\ 0.9 \\ 2 \\ 3 \\ 3 \end{bmatrix}, \; w = \begin{bmatrix} 2 \\ 6 \\ 5 \\ 1 \\ 4 \end{bmatrix}, \quad \text{and } b = 7 \tag{6.14}$$

The only difference between this program and other polynomials is that we need to call `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 [Lasserre, 2001a] for more info.

The code below:

```
% 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]';
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 $x^T = [1,0,0,1,1], w^T x = 7, p^* = 8.6626$.

6.13 A bigger 0-1 problem

Consider the problem

$$\begin{cases} \max_x c^T x \\ w^T b \leq b \\ x_i \in \{0,1\}, \, \forall i = 1, \ldots, 5 \end{cases}$$

where $b = 71$ and c and w are 75×1 vectors given by

$$\begin{bmatrix}
c(1:25) & c(26:50) & c(51:75) \\
0.2878 & 0.3888 & 2.403 \\
2.243 & 0.6752 & 0.6989 \\
2.246 & 1.05 & 2.797 \\
1.63 & 0.8613 & 2.29 \\
1.014 & 2.782 & 2.479 \\
2.497 & 0.1539 & 1.72 \\
1.658 & 1.778 & 2.378 \\
2.873 & 0.4887 & 0.9871 \\
2.678 & 2.515 & 0.6704 \\
1.07 & 0.5027 & 0.9372 \\
1.639 & 1.507 & 1.754 \\
1.04 & 2.998 & 2.49 \\
1.868 & 1.066 & 0.8714 \\
2.39 & 0.1412 & 1.208 \\
2.238 & 0.641 & 2.586 \\
0.3766 & 1.194 & 1.844 \\
2.467 & 1.001 & 2.974 \\
0.07545 & 0.6888 & 0.6111 \\
1.243 & 2.808 & 2.482 \\
2.194 & 2.05 & 2.028 \\
2.344 & 2.886 & 0.7468 \\
1.102 & 1.314 & 1.427 \\
2.235 & 2.821 & 1.197 \\
2.677 & 0.0175 & 1.798 \\
0.7278 & 1.831 & 2.402
\end{bmatrix}, \begin{bmatrix}
w(1:25) & w(26:50) & w(51:75) \\
3 & 16 & 15 \\
17 & 3 & 10 \\
17 & 13 & 13 \\
8 & 6 & 18 \\
9 & 9 & 4 \\
12 & 17 & 8 \\
15 & 4 & 20 \\
15 & 7 & 9 \\
16 & 10 & 14 \\
8 & 7 & 19 \\
9 & 16 & 20 \\
20 & 20 & 14 \\
12 & 4 & 3 \\
17 & 5 & 1 \\
6 & 15 & 13 \\
13 & 8 & 12 \\
12 & 20 & 20 \\
20 & 20 & 15 \\
2 & 13 & 14 \\
11 & 18 & 11 \\
11 & 9 & 6 \\
2 & 13 & 20 \\
19 & 20 & 11 \\
18 & 12 & 1 \\
9 & 19 & 14
\end{bmatrix} \tag{6.15}$$

The code below:

```
% AUTHOR: Jose Lopez NEU 2014
% LICENSE:This code is licensed under the BSD License.
%
% This script implements a 75 variable 0-1 problem.
%
clear;clc;
%%
n_vars = 75;
```

```
c = [0.287848249822753 2.24260098521491 2.24552687217988 1.62989829925533 1.01439694
2.49700098486158 1.65771653407787 2.87262935088606 2.678499957954 1.06951085138101 .
1.63920567897566 1.04004525376485 1.86840837197739 2.38987405592519 2.23762470462195
0.376608694064459 2.46718202027706 0.0754515042855065 1.24328664277209 2.19422240391
2.34412200082789 1.10185774539411 2.23460356872502 2.67680156469332 0.72781015883900
0.388790927512621 0.67520364638374 1.0500417570023 0.861253839438528 2.7824640215382
0.153941270450387 1.77800026561916 0.488696700653197 2.51521730901909 0.502682790633
1.50660184622842 2.99798843707716 1.06622145380364 0.141233128646502 0.6409817553623
1.19351737230812 1.00100454114088 0.688807595904593 2.8083605325062 2.04956635409995
2.88634138716859 1.31391953289839 2.82100996326272 0.0175029737171126 1.830921090209
2.40322727089836 0.698944600240944 2.79740602776294 2.28978832600123 2.4793486191080
1.7203907658146 2.37774499236068 0.987123587517743 0.670385928377558 0.9371590260577
1.75357043558567 2.48974242306457 0.871387474641804 1.20766317618937 2.5861718952994
1.8442186506492 2.97356336498938 0.611096625556866 2.48162725483996 2.02758485685832
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 .
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),']\n'];
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)
%
% [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
$x^T = [0000000000000010001001000010010101000100000101000000000100000001100000000010]$,
$c^T x = 25.451$, and $w^T x = 71$.

6.14 Using the running intersection property with reweighted heuristic

```
% AUTHOR: Jose Lopez NEU 2014
% LICENSE:This code is licensed under the BSD License.
%
% This example shows how to use the running intersection property and the
% reweighted heuristic to get a rank-1 moment matrix.
%
% objective:    min f0 = x1*x2 + x1*x3
% subject to
%               K =  { x1^2+x2^2<=1, x1^2+x3^2<=2 }
%
clear; clc;
%%
%%
flag = 1;
```

```
count = 1;
%first run, no reweighted heuristic
W1 = zeros(3);
W2 = zeros(3);
I = eye(3);

%%
% generally, we want to do all this setup outside the loop because this only
% needs to be done once. it makes no difference for this small problem but
% for larger problems doing this outside the loop can save a lot of time.
mons12 = [0 0 0;1 0 0; 0 1 0];
mons13 = [0 0 0;1 0 0; 0 0 1];
[M12,~] = get_mmatrix(1,3,mons12);
[M13,~] = get_mmatrix(1,3,mons13);
[monomials,n_monomials] = get_monomials({M12,M13}); %get all mons
indices.x1x2 = get_index(monomials,[1 1 0]);
indices.x1x3 = get_index(monomials,[1 0 1]);
indices.x1_2 = get_index(monomials,[2 0 0]);
indices.x2_2 = get_index(monomials,[0 2 0]);
indices.x3_2 = get_index(monomials,[0 0 2]);
indices.x = get_index(monomials,eye(3));
maps.M12 = get_map(M12,monomials);
maps.M13 = get_map(M13,monomials);
%%
while(flag)
    fprintf('Iteration %d:...\n',count);
    cvx_clear
    cvx_begin sdp
    cvx_solver sedumi

    variable y(n_monomials);

    M12 = assignm(y,maps.M12);
    M13 = assignm(y,maps.M13);
    x = y(indices.x);
    x1x2 = y(indices.x1x2);
    x1x3 = y(indices.x1x3);
    x1_2 = y(indices.x1_2);
    x2_2 = y(indices.x2_2 );
    x3_2 = y(indices.x3_2 );

    f0 = x1x2 + x1x3;

    minimize(f0+trace(W1*M12+W2*M13))
    subject to
    y(1)==1
    M12>=0
    M13>=0
    x1_2+x2_2<=1
    x1_2+x3_2<=2
    cvx_end

    S1 = svd(M12);
    S2 = svd(M13);
    W1 = inv(M12+I*S1(2));
    W2 = inv(M13+I*S2(2));
    flag = or(rank(M12,1e-4)~=1,rank(M13,1e-4)~=1);
    count = count + 1;
end
fprintf('reached rank-1 moment matrices!\n');
fprintf('one solution is: \n');
%show results
f0
x
%%
% after a couple of these reweightings:
% f0 = -1.4142
% x1* = 0.81649, x2* = -0.57727, x3* = -1.1546
%
% global solutions:
% x* = [pm 0.8165; mp 0.57734; mp 1.1547]
% f0* = -1.4142
%%
return
%%
mset clear
mpol x 3
f0 = mom(x(1)*x(2)) + mom(x(1)*x(3));
K = [x(1)^2+x(2)^2<=1;x(1)^2+x(3)^2<=2]
mu = meas;
P = msdp(min(f0),K)
[status,obj,M] = msol(P)
%%

I = eye(4);
reweight = 1;
fprintf('\n\n');
while(status<1)
    fprintf('reweight = %d \n',reweight);
    M1 = mmat(M);
    W = inv(double(M1)+I*0.01);
    f1 = trace(W*M1);

    %%
    P2 = msdp(min(f0+f1),K);
    [status,obj,M] = msol(P2);

    %%
    temp = mmat(M(1));
    svd( double(temp) )
    reweight = reweight + 1;
end
double(f0)
z = double(x)
fz = z(1)*z(2)+z(1)*z(3)

%%
```

6.15 Optimizing rational functions

```
% AUTHOR: Jose Lopez NEU 2014
% LICENSE:This code is licensed under the BSD License.
%
% This example implements an example of optimizing rational functions.
%
% minimize f(x) = (x+1)/(x+10) on [0,1] using method from
% Lasserre 2009: Moments and sos for polynomial optzn and related problems
% PRIMAL METHOD (3.2)
%
clear;clc;
close all;
%%
[M,basis_data] = get_mmatrix(2,1);
[monomials,n_monomials] = get_monomials(M);

%%
cvx_clear
cvx_begin sdp
cvx_solver sedumi

variable y(n_monomials,1);
[M,basis_data] = get_mmatrix(1,1);
M1 = assignm(y,get_map(M,monomials));
```

```
L = localize(M,1);
L1 = assignm(y,get_map(L,monomials));

y0 = y(1);
y1 = y(2);

minimize( y1 + y0)
subject to
M1>=0
L1>=0
M1-L1>=0
y1+10*y0==1
cvx_end
% cvx_optval = 0.1 as expected
% warm fuzzies: t = linspace(0,1,1000);f = (t+1)./(t+10);min(f) %=0.1

%%
% DUAL METHOD (3.5). I made this example before I knew Sedumi also computes
% the dual solution when solving the primal.
%
[M,~] = get_mmatrix(2,1);
[monomials,n_monomials] = get_monomials(M);
[M,basis_data] = get_mmatrix(1,1);
B = get_index_matrices(monomials,get_map(M,monomials));
C = get_index_matrices(monomials,get_map(localize(M,1),monomials));
%%
cvx_clear
cvx_begin sdp
cvx_solver sedumi

variable lambda;
variable X1(2,2) symmetric;
variable X2(2,2) symmetric;
variable X3(2,2) symmetric;

%(y1 + y0) / (y1 + 10*y0)
%p0 = 1, p1 = 1 p2 = 0 q0 = 10 q1 = 1 q2 = 0
p = [1,1,0,0,0];
q = [10,1,0,0,0];

maximize( lambda )
subject to
for k = 1:n_monomials
    p(k) - lambda*q(k) == trace(X1*B{k}) + trace(X2*C{k}) + trace(X3*(B{k}-C{k}));
end
```

```
X1>=0;
X2>=0;
X3>=0;
cvx_end
% cvx_optval = 0.1 as expected!
```

6.16 Joint+Marginal Example 3.9

```
% AUTHOR: Jose Lopez NEU 2014
% LICENSE:This code is licensed under the BSD License.
%
% This example implements a joint+marginal example from Lasserre
%
% REFERENCE: JOINT+MARGINAL PARAMETRIC 2010, Ex. 3.9
% K = { 1 - x^2 -y^2 >=0}
% f(x,y) = -y*x^2
%
clear;clc;
close all;
set(0, 'DefaultFigurePosition', [111,243,809,679]);
set(0, 'DefaultAxesFontSize', 14);
set(0, 'DefaultLineLineWidth', 2);

%%
% I made this example before I knew Sedumi solves both the primal & dual.
% DUAL
moments = uniform_moments(0,1,0:7);

cvx_clear
cvx_begin sdp
cvx_solver sedumi

[M,~] = get_mmatrix(4,2);
[monomials,n_monomials] = get_monomials(M);
B = get_index_matrices(monomials,get_map(M,monomials));

[M,~] = get_mmatrix(3,2);
C = get_index_matrices(monomials,get_map(M,monomials));

L = localize(M,[1 0]);
C10 = get_index_matrices(monomials,get_map(L,monomials));

L = localize(M,[0 1]);
```

```
C01 = get_index_matrices(monomials,get_map(L,monomials));

L = localize(M,[2 0]);
C20 = get_index_matrices(monomials,get_map(L,monomials));

L = localize(M,[0 2]);
C02 = get_index_matrices(monomials,get_map(L,monomials));

%[(1:n_monomials)',monomials]
n_moments = length(moments);
variable p(n_moments,1);
variable X(15,15) symmetric;
variable Z1(10,10) symmetric; %h1 = 1-x^2-y^2
variable Z2(10,10) symmetric; %h2 = 1-x
variable Z3(10,10) symmetric; %h3 = x
variable Z4(10,10) symmetric; %h4 = 1-y
variable Z5(10,10) symmetric; %h5 = y

f = zeros(n_monomials,1);
f(get_index(monomials,[2 1])) = -1;
idx = get_index(monomials,[zeros(8,1),(0:7)']);
maximize( moments'*p )
subject to

for k = 1:n_monomials
    [~,idx_p] = ismember(k,idx);
    if(idx_p)
        f(k) - p(idx_p) == trace(X*B{k})+...
            trace(Z1*(C{k}-C20{k}-C02{k}))+...
            trace(Z4*(C{k}-C01{k}))+...
            trace(Z5*C01{k});
        %           trace(Z2*(C{k}-C10{k}))+...
        %           trace(Z3*C10{k})+...
    else
        f(k) == trace(X*B{k})+...
            trace(Z1*(C{k}-C20{k}-C02{k}))+...
            trace(Z4*(C{k}-C01{k}))+...
            trace(Z5*C01{k});
        %           trace(Z2*(C{k}-C10{k}))+...
        %           trace(Z3*C10{k})+...
    end
end
X>=0;
Z1>=0;
Z2>=0;
```

```
Z3>=0;
Z4>=0;
Z5>=0;
cvx_end
cvx_optval
%%
y = linspace(0,1,100);
Jy = y.*(1-y.^2);
p = -p;
% p = [0.00116694589068829;
% 0.97675523582367;
% 0.182416317893377;
% -1.7165913157714;
% 1.54416265272775;
% -1.85319093145952;
% 1.1612101865023;
% -0.29591779025793]
f = zeros(size(y));
for k = 0:7
    f = f + p(k+1)*(y.^k);
end
close all;
subplot(2,1,1);
plot(y,Jy,'-b.');
grid on;
hold on;
plot(y,f,':ro');
legend('Jy','-p');
xlabel('y');
title('Optimal Solution J(Y) AND ITS SOS APPROXIMATION');
subplot(2,1,2);
plot(Jy-f,'-b.');
grid on;
ylim([-1,1]*1e-3);
xlabel('Data Point');
ylabel('Jy-(-p)');
%%
return
%%
% PRIMAL, -0.250067 VS -0.25001786 IN LASSERRE
moments = uniform_moments(0,1,0:7);
[M,basis_data] = get_mmatrix(4,2);
[monomials,n_monomials] = get_monomials(M);
```

```
cvx_clear
cvx_begin sdp
cvx_solver sedumi

variable z(n_monomials,1);
dual variable p{8}; %p is the same as p from the dual program above
M4 = assignm(z,get_map(M,monomials));

[M,~] = get_mmatrix(3,2);
M3 = assignm(z,get_map(M,monomials));
L = localize(M,[1 0]);
L10 = assignm(z,get_map(L,monomials));
L = localize(M,[0 1]);
L01 = assignm(z,get_map(L,monomials));
L = localize(M,[2 0]);
L20 = assignm(z,get_map(L,monomials));
L = localize(M,[0 2]);
L02 = assignm(z,get_map(L,monomials));

%f(x,y) = -y*x^2

z21 = z(get_index(monomials,[2 1])); % (x^2) y
z00 = z(get_index(monomials,[0 0]));
z01 = z(get_index(monomials,[0 1])); % y
z02 = z(get_index(monomials,[0 2]));
z03 = z(get_index(monomials,[0 3]));
z04 = z(get_index(monomials,[0 4]));
z05 = z(get_index(monomials,[0 5]));
z06 = z(get_index(monomials,[0 6]));
z07 = z(get_index(monomials,[0 7]));
z08 = z(get_index(monomials,[0 8]));
z10 = z(get_index(monomials,[1 0])); % x

minimize( -z21 )
subject to
M4>=0
% L10>=0
% M3-L10>=0
L01>=0
M3-L01>=0
M3 - L20 - L02 >=0

z00 == moments(1) : p{1} %this is how to get the dual variables
z01 == moments(2) : p{2}
z02 == moments(3) : p{3}
```

```
z03 == moments(4) : p{4}
z04 == moments(5) : p{5}
z05 == moments(6) : p{6}
z06 == moments(7) : p{7}
z07 == moments(8) : p{8}
%z08 == moments(9)
cvx_end
%p = [-0.00117102748164474;
% -0.976678510580376;
% -0.183038189313921;
% 1.71914823541913;
% -1.54993335291317;
% 1.86041775847208;
% -1.16591353221268;
% 0.297156879989116]
```

6.17 LP Example 5.5

```
% AUTHOR: Jose Lopez NEU 2014
% LICENSE:This code is licensed under the BSD License.
%
% This script implements Example 5.5 in Lasserre text, pg. 127. This is
% when I was checking to see if LP/moment formulations were possibly better
% than SDP/moment formulations...
%
clear;clc;
close all;
%%
cvx_clear
cvx_begin
cvx_solver sedumi
%cvx_precision high


variables y0 y1 y2 y3 y4 y5 y6 y7 y8 y9 y10 y11 y12 y13 y14 y15
f0 = y2-y1;
minimize(f0)
subject to
y0==1

y1>=0
1-y1>=0
```

```
y2>=0
-y2+y1>=0
y2-2*y1+1>=0

y3>=0
-y3+y2>=0
y3-2*y2+y1>=0
-y3+3*y2-3*y1+1>=0

y4>=0
-y4+y3>=0
y4-2*y3+y2>=0
-y4+3*y3-3*y2+y1>=0
y4-4*y3+6*y2-4*y1+1>=0

y5>=0
-y5+y4>=0
y5-2*y4+y3>=0
-y5+3*y4-3*y3+y2>=0
y5-4*y4+6*y3-4*y2+y1>=0
-y5+5*y4-10*y3+10*y2-5*y1+1>=0

y6>=0
-y6+y5>=0
y6-2*y5+y4>=0
-y6+3*y5-3*y4+y3>=0
y6-4*y5+6*y4-4*y3+y2>=0
-y6+5*y5-10*y4+10*y3-5*y2+y1>=0
y6-6*y5+15*y4-20*y3+15*y2-6*y1+1>=0

y7>=0
-y7+y6>=0
y7-2*y6+y5>=0
-y7+3*y6-3*y5+y4>=0
y7-4*y6+6*y5-4*y4+y3>=0
-y7+5*y6-10*y5+10*y4-5*y3+y2>=0
y7-6*y6+15*y5-20*y4+15*y3-6*y2+y1>=0
-y7+7*y6-21*y5+35*y4-35*y3+21*y2-7*y1+1>=0

y8>=0
-y8+y7>=0
y8-2*y7+y6>=0
-y8+3*y7-3*y6+y5>=0
y8-4*y7+6*y6-4*y5+y4>=0
-y8+5*y7-10*y6+10*y5-5*y4+y3>=0
```

```
y8-6*y7+15*y6-20*y5+15*y4-6*y3+y2>=0
-y8+7*y7-21*y6+35*y5-35*y4+21*y3-7*y2+y1>=0
y8-8*y7+28*y6-56*y5+70*y4-56*y3+28*y2-8*y1+1>=0

y9>=0
-y9+y8>=0
y9-2*y8+y7>=0
-y9+3*y8-3*y7+y6>=0
y9-4*y8+6*y7-4*y6+y5>=0
-y9+5*y8-10*y7+10*y6-5*y5+y4>=0
y9-6*y8+15*y7-20*y6+15*y5-6*y4+y3>=0
-y9+7*y8-21*y7+35*y6-35*y5+21*y4-7*y3+y2>=0
y9-8*y8+28*y7-56*y6+70*y5-56*y4+28*y3-8*y2+y1>=0
-y9+9*y8-36*y7+84*y6-126*y5+126*y4-84*y3+36*y2-9*y1+1>=0

y10>=0
-y10+y9>=0
y10-2*y9+y8>=0
-y10+3*y9-3*y8+y7>=0
y10-4*y9+6*y8-4*y7+y6>=0
-y10+5*y9-10*y8+10*y7-5*y6+y5>=0
y10-6*y9+15*y8-20*y7+15*y6-6*y5+y4>=0
-y10+7*y9-21*y8+35*y7-35*y6+21*y5-7*y4+y3>=0
y10-8*y9+28*y8-56*y7+70*y6-56*y5+28*y4-8*y3+y2>=0
-y10+9*y9-36*y8+84*y7-126*y6+126*y5-84*y4+36*y3-9*y2+y1>=0
y10-10*y9+45*y8-120*y7+210*y6-252*y5+210*y4-120*y3+45*y2-10*y1+1>=0

y11>=0
-y11+y10>=0
y11-2*y10+y9>=0
-y11+3*y10-3*y9+y8>=0
y11-4*y10+6*y9-4*y8+y7>=0
-y11+5*y10-10*y9+10*y8-5*y7+y6>=0
y11-6*y10+15*y9-20*y8+15*y7-6*y6+y5>=0
-y11+7*y10-21*y9+35*y8-35*y7+21*y6-7*y5+y4>=0
y11-8*y10+28*y9-56*y8+70*y7-56*y6+28*y5-8*y4+y3>=0
-y11+9*y10-36*y9+84*y8-126*y7+126*y6-84*y5+36*y4-9*y3+y2>=0
y11-10*y10+45*y9-120*y8+210*y7-252*y6+210*y5-120*y4+45*y3-10*y2+y1>=0
-y11+11*y10-55*y9+165*y8-330*y7+462*y6-462*y5+330*y4-165*y3+55*y2-11*y1+1>=0

y12>=0
-y12+y11>=0
y12-2*y11+y10>=0
-y12+3*y11-3*y10+y9>=0
y12-4*y11+6*y10-4*y9+y8>=0
```

```
-y12+5*y11-10*y10+10*y9-5*y8+y7>=0
y12-6*y11+15*y10-20*y9+15*y8-6*y7+y6>=0
-y12+7*y11-21*y10+35*y9-35*y8+21*y7-7*y6+y5>=0
y12-8*y11+28*y10-56*y9+70*y8-56*y7+28*y6-8*y5+y4>=0
-y12+9*y11-36*y10+84*y9-126*y8+126*y7-84*y6+36*y5-9*y4+y3>=0
y12-10*y11+45*y10-120*y9+210*y8-252*y7+210*y6-120*y5+45*y4-10*y3+y2>=0
-y12+11*y11-55*y10+165*y9-330*y8+462*y7-462*y6+330*y5-165*y4+55*y3-11*y2+y1>=0
y12-12*y11+66*y10-220*y9+495*y8-792*y7+924*y6-792*y5+495*y4-220*y3+66*y2-12*y1+1>=0

y13>=0
-y13+y12>=0
y13-2*y12+y11>=0
-y13+3*y12-3*y11+y10>=0
y13-4*y12+6*y11-4*y10+y9>=0
-y13+5*y12-10*y11+10*y10-5*y9+y8>=0
y13-6*y12+15*y11-20*y10+15*y9-6*y8+y7>=0
-y13+7*y12-21*y11+35*y10-35*y9+21*y8-7*y7+y6>=0
y13-8*y12+28*y11-56*y10+70*y9-56*y8+28*y7-8*y6+y5>=0
-y13+9*y12-36*y11+84*y10-126*y9+126*y8-84*y7+36*y6-9*y5+y4>=0
y13-10*y12+45*y11-120*y10+210*y9-252*y8+210*y7-120*y6+45*y5-10*y4+y3>=0
-y13+11*y12-55*y11+165*y10-330*y9+462*y8-462*y7+330*y6-165*y5+55*y4-11*y3+y2>=0
y13-12*y12+66*y11-220*y10+495*y9-792*y8+924*y7-792*y6+495*y5-220*y4+66*y3-12*y2+y1>=
-y13+13*y12-78*y11+286*y10-715*y9+1287*y8-1716*y7+1716*y6-1287*y5+715*y4-286*y3+78*y

y14>=0
-y14+y13>=0
y14-2*y13+y12>=0
-y14+3*y13-3*y12+y11>=0
y14-4*y13+6*y12-4*y11+y10>=0
-y14+5*y13-10*y12+10*y11-5*y10+y9>=0
y14-6*y13+15*y12-20*y11+15*y10-6*y9+y8>=0
-y14+7*y13-21*y12+35*y11-35*y10+21*y9-7*y8+y7>=0
y14-8*y13+28*y12-56*y11+70*y10-56*y9+28*y8-8*y7+y6>=0
-y14+9*y13-36*y12+84*y11-126*y10+126*y9-84*y8+36*y7-9*y6+y5>=0
y14-10*y13+45*y12-120*y11+210*y10-252*y9+210*y8-120*y7+45*y6-10*y5+y4>=0
-y14+11*y13-55*y12+165*y11-330*y10+462*y9-462*y8+330*y7-165*y6+55*y5-11*y4+y3>=0
y14-12*y13+66*y12-220*y11+495*y10-792*y9+924*y8-792*y7+495*y6-220*y5+66*y4-12*y3+y2>
-y14+13*y13-78*y12+286*y11-715*y10+1287*y9-1716*y8+1716*y7-1287*y6+715*y5-286*y4+78*
y14-14*y13+91*y12-364*y11+1001*y10-2002*y9+3003*y8-3432*y7+3003*y6-2002*y5+1001*y4-3

y15>=0
-y15+y14>=0
y15-2*y14+y13>=0
-y15+3*y14-3*y13+y12>=0
y15-4*y14+6*y13-4*y12+y11>=0
```

```
-y15+5*y14-10*y13+10*y12-5*y11+y10>=0
y15-6*y14+15*y13-20*y12+15*y11-6*y10+y9>=0
-y15+7*y14-21*y13+35*y12-35*y11+21*y10-7*y9+y8>=0
y15-8*y14+28*y13-56*y12+70*y11-56*y10+28*y9-8*y8+y7>=0
-y15+9*y14-36*y13+84*y12-126*y11+126*y10-84*y9+36*y8-9*y7+y6>=0
y15-10*y14+45*y13-120*y12+210*y11-252*y10+210*y9-120*y8+45*y7-10*y6+y5>=0
-y15+11*y14-55*y13+165*y12-330*y11+462*y10-462*y9+330*y8-165*y7+55*y6-11*y5+y4>=0
y15-12*y14+66*y13-220*y12+495*y11-792*y10+924*y9-792*y8+495*y7-220*y6+66*y5-12*y4+y3
-y15+13*y14-78*y13+286*y12-715*y11+1287*y10-1716*y9+1716*y8-1287*y7+715*y6-286*y5+78
y15-14*y14+91*y13-364*y12+1001*y11-2002*y10+3003*y9-3432*y8+3003*y7-2002*y6+1001*y5-
-y15+15*y14-105*y13+455*y12-1365*y11+3003*y10-5005*y9+6435*y8-6435*y7+5005*y6-3003*y

cvx_end
y1
y0
%  i   2    4    6    10     15
% Li -1/3 -1/3 -0.3 -0.27 -0.2695
%%
return
%%
basis_data = get_monomial_basis(15,2);
basis = double(basis_data.basis);
syms X real
g1 = X
g2 = (1-X)
n_mons = size(basis,1);
temp = [];
for k = 1:n_mons
    mon = basis(k,:);
    temp = [temp;
    expand(g1^mon(1)*g2^mon(2))];
end
```

6.18 L1 Approximation

This example is from [Lasserre,].

```
% AUTHOR: Jose Lopez NEU 2014
% LICENSE:This code is licensed under the BSD License.
%
% this script implements:
% EXAMPLE 1 IN LASSERRE 2010, BEST L1 APPROXIMATION...
% Approximate f = x1^4 x2^2 + x1^2 x2^4 - x1^2 x2^2 + 1/27
% with a sos polynomial
```

```
%
% I made this example when I didn't know how to get the dual from the
% primal. See the joint+marginal example to see how to do it an easier way.
%
clear;clc;
%%
[M,basis_data] = get_mmatrix(3,2);
[monomials,n_monomials] = get_monomials(M); %get all moments in mmatrix
map = get_map(M,monomials);
B = get_index_matrices(monomials,map); %get index matrices
rows = size(M(:,:,1),1);

%%
cvx_clear
cvx_begin sdp
cvx_solver sedumi;

variable X(rows,rows) symmetric;
variable lambda(n_monomials,1);
variable g(n_monomials,1);

f = zeros(n_monomials,1);
f( get_index(monomials,[4 2;2 4; 2 2; 0 0]) ) = [1;1;-1;1/27];

minimize( sum(lambda) )
subject to
for k = 1:n_monomials
    lambda(k) + g(k) >= f(k);
    lambda(k) - g(k) >= -f(k);
    g(k)-trace(X*B{k})==0;
end
lambda>=0;
X>=0;
cvx_end
%answer: cvx_optval = 1.6E-2, lambda = (5.445,5.367,5.367)E-3
lambda = clean(lambda,1e-6);
idx = find(lambda);
[double(monomials(idx,:)),lambda(idx)]
```

6.19 Robust Regression

This example was requested by Prof. Sznaier.

```
% AUTHOR: Jose Lopez NEU 2015
% LICENSE:This code is licensed under the BSD License.
%
% This "robust regression" example shows how to use the running
% intersection property and the reweighted heuristic to get a rank-1
% solution.
%
% objective:                 max_(s,r) f0 = sum( s )
%                  subject to
%                            si(r'xi)=0
%                            si^2 = si
%

clear;clc;

%%
% the data comes from the line: x2 = 1.7*x1 (+ 1 outlier)
% r = [-0.8619;0.5070] = nullspace of data' without outlier
% last column of data is the outlier:
data = [-2 -1 0 1 2 3;
    -3.4 -1.7 0 1.7 3.4 1];

% r'*data = [0 0 0 0 0 -2.07878251890991]
n_points = size(data,2);

%%
n_vars = n_points + 2; % selector variables + r
relaxation = 1;
[M,basis_data] = get_mmatrix(relaxation,n_vars);
mons1 = basis_data.monomial{2}; %first order monomials
s_mons = mons1(1:n_points,:);   %these are the monomials for s and r
r_mons = mons1(n_points+1:end,:);

% since we want to use running intersection property we need
% n_points moment matrices for si*r terms
M = cell(n_points,1);
for k = 1:n_points
    temp = [zeros(1,n_vars); s_mons(k,:); r_mons]; %monomials in each 4x4 matrix
    M{k} = get_mmatrix(relaxation,n_vars,temp);
    M{k} = make_zero_one(M{k}, s_mons(k,:)); % make si 0-1
end

% get all the monomials in our problem
[monomials, n_monomials] = get_monomials(M);
```

```
% get maps to our decision vector for use in CVX (or YALMIP)
% we generally want to do this outside the reweighting loop
indices.s = get_index(monomials,s_mons);
indices.r = get_index(monomials,r_mons);
indices.r1_2 = get_index(monomials,r_mons(1,:)+r_mons(1,:));
indices.r2_2 = get_index(monomials,r_mons(2,:)+r_mons(2,:));

maps.M = cell(n_points,1);
for k = 1:n_points
    maps.M{k} = get_map(M{k},monomials);
    indices.si_r1(k,1) = get_index(monomials, s_mons(k,:) + r_mons(1,:));
    indices.si_r2(k,1) = get_index(monomials, s_mons(k,:) + r_mons(2,:));
end

% setup reweighting stuff
% spoiler alert: it takes 57 iterations

I = eye(4);
W = cell(n_points,1);
%first run, no reweighted heuristic
W(:) = {zeros(4)};
iteration = 1;
flag = 1;

%%
while ( flag )
cvx_clear;
cvx_begin sdp;
cvx_solver sedumi;

variable y(n_monomials,1);

s = y(indices.s);
r = y(indices.r);

si_r1 = y(indices.si_r1);
si_r2 = y(indices.si_r2);
r1_2 = y(indices.r1_2);
r2_2 = y(indices.r2_2);

M1 = cell(n_points,1);
f1 = 0;
% yes, there are a lot of for loops but these are just assignments...
for k = 1:n_points
    M1{k} = assignm(y,maps.M{k});
    f1 = f1 + trace(W{k}*M1{k});
end
f0 = sum(s);

maximize( f0 - f1 )
subject to
y(1)==1;
r1_2 + r2_2 == 1;
for k = 1:n_points
    M1{k}>=0;
    si_r1(k)*data(1,k) + si_r2(k)*data(2,k) == 0;
end

cvx_end

rank1_test = zeros(1,n_points);
temp = zeros(4,n_points); % the SVDs of the moment matrices
for k = 1:n_points
    temp(:,k) = svd(full(M1{k}));
    rank1_test(k) = temp(2,k);
    W{k} = inv( M1{k} + I*temp(2,k) );
end
flag = any( rank1_test > 1e-4); % stop when all are rank-1
iteration = iteration + 1;
fprintf('ITERATION: %d\n',iteration);
temp
end
fprintf('Rank-1 solution found!\n');
% r = [-0.861934215157769;0.507020126563394] = nullspace of data'
s
r
```

Bibliography

[Bertsimas et al., 2008] Bertsimas, D., Doan, X. V., and Lasserre, J. B. (2008). Approximating integrals of multivariate exponentials: A moment approach. *Oper. Res. Lett.*, 36(2):205–210.

[Blekherman et al., 2012] Blekherman, G., Parrilo, P. A., Thomas, R. R., Blekherman, G., Parrilo, P. A., and Thomas, R. R. (2012). *Semidefinite Optimization and Convex Algebraic Geometry*. SIAM.

[Henrion and Lasserre, 2005] Henrion, D. and Lasserre, J. B. (2005). Detecting global optimality and extracting solutions in gloptipoly. In *Positive polynomials in control*, pages 293–310. Springer Berlin Heidelberg.

[Lasserre,] Lasserre, J. B. Best ℓ_1-approximation of nonnegative polynomials by sums of squares. *ArXiv e-prints*.

[Lasserre, 2001a] Lasserre, J. B. (2001a). An explicit exact sdp relaxation for nonlinear 0-1 programs. In *Integer Programming and Combinatorial Optimization*, pages 293–303. Springer Berlin Heidelberg.

[Lasserre, 2001b] Lasserre, J. B. (2001b). Global optimization with polynomials and the problem of moments. *SIAM Journal on Optimization*, 11(3):796–817.

[Lasserre, 2002] Lasserre, J. B. (2002). Bounds on measures satisfying moment conditions. *Annals of Applied Probability*, pages 1114–1137.

[Lasserre, 2006] Lasserre, J. B. (2006). A sum of squares approximation of nonnegative polynomials. *SIAM J. Optim*, 16:751–765.

[Lasserre, 2008] Lasserre, J. B. (2008). A semidefinite programming approach to the generalized problem of moments. *Mathematical Programming*, 112(1):65–92.

[Lasserre, 2009] Lasserre, J. B. (2009). *Moments, positive polynomials and their applications*, volume 1. World Scientific.

[Lopez et al., 2015] Lopez, J. A., Camps, O., and Sznaier, M. (2015). Robust anomaly detection using semidefinite programming. *arXiv preprint arXiv:1504.00905*.