SparsePOP: a **Sparse** Semidefinite Programming Relaxation of **P**olynomial **O**ptimization **P**roblems

Hayato Waki, Sunyoung Kim, Masakazu Kojima and Masakazu Muramatsu

March 2005, B–414

B-414 **SparsePOP** : a **Sparse** Semidefinite Programming Relaxation of **P**olynomial **O**ptimization **P**roblems

Hayato Waki[⋆], Sunyoung Kim[†], Masakazu Kojima[‡], Masakazu Muramatsu[♯]

March 2005

Abstract.
SparesPOP is a MATLAB implementation of a sparse semidefinite programming (SDP) relaxation method proposed for polynomial optimization problems (POPs) in the recent paper by Waki *et al.* The sparse SDP relaxation is based on "a hierarchy of LMI relaxations of increasing dimensions" by Lasserre, and exploits a sparsity structure of polynomials in POPs. The efficiency of **SparsePOP** to compute bounds for optimal values of POPs is increased and larger scale POPs can be handled. The software package **SparesPOP** and this manual with some numerical examples are available at

http://www.is.titech.ac.jp/∼kojima/SparsePOP

Key words.
Polynomial optimization problem, sparsity, global optimization, sums of squares optimization, semidefinite programming relaxation, MATLAB software package

⋆ Department of Mathematical and Computing Sciences, Tokyo Institute of Technology, 2-12-1 Oh-Okayama, Meguro-ku, Tokyo 152-8552 Japan. *Hayato.Waki@is.titech.ac.jp*

† Department of Mathematics, Ewha Women's University, 11-1 Dahyun-dong, Sudaemoon-gu, Seoul 120-750 Korea. A considerable part of this work was conducted while this author was visiting Tokyo Institute of Technology. Research was supported by KRF 2004-042-C00014. *skim@ewha.ac.kr*

‡ Department of Mathematical and Computing Sciences, Tokyo Institute of Technology, 2-12-1 Oh-Okayama, Meguro-ku, Tokyo 152-8552 Japan. Research supported by Grant-in-Aid for Scientific Research on Priority Areas 16016234. *kojima@is.titech.ac.jp*

♯ Department of Computer Science, The University of Electro-Communications, Chofugaoka, Chofu-Shi, Tokyo 182-8585 Japan. Research supported in part by Grant-in-Aid for Young Scientists (B) 15740054. *muramatu@cs.uec.ac.jp*

1 Introduction

Let $\mathbb{R}^n$ and $\mathbb{Z}^n_+$ denote the n-dimensional Euclidean space and the set of nonnegative integer vectors, respectively. We express a real-valued polynomial $f_k(\boldsymbol{x})$ in $\boldsymbol{x} = (x_1, x_2, \ldots, x_n) \in \mathbb{R}^n$ as

$$f_k(\boldsymbol{x}) \;=\; \sum_{\boldsymbol{\alpha} \in \mathcal{F}_k} c_k(\boldsymbol{\alpha}) \boldsymbol{x}^{\boldsymbol{\alpha}},\; \boldsymbol{x} \in \mathbb{R}^n,\; c_k(\boldsymbol{\alpha}) \;\in \mathbb{R},\; \mathcal{F}_k \subset \mathbb{Z}^n_+$$

$(k = 0, 1, 2, \ldots, m)$, where $\boldsymbol{x}^{\boldsymbol{\alpha}} = x_1^{\alpha_1} x_2^{\alpha_2} \cdots x_n^{\alpha_n}$ for every $\boldsymbol{x} = (x_1, x_2, \ldots, x_n) \in \mathbb{R}^n$ and every $\boldsymbol{\alpha} \in \mathbb{Z}^n_+$. A general polynomial optimization problem (POP) is of the form

$$\left.\begin{array}{ll} \text{minimize} & f_0(\boldsymbol{x}) \\ \text{subject to} & f_k(\boldsymbol{x}) \geq 0 \quad (k = 1, 2, \ldots, m). \end{array}\right\} \tag{1}$$

The software package SparsePOP is a MATLAB implementation of a sparse semidefinite (SDP) relaxation method proposed for POPs in the paper [10]. See also [3]. Based on the concept of Lasserre's hierarchy of LMI relaxations of increasing dimensions [5] for POPs, this software is designed to solve larger-sized POPs by exploiting the sparsity of polynomials in POPs.

This paper is organized as follows. In the rest of this section, we overview the structure of SparsePOP. In Section 2, we give a brief introduction for sparse SDP relaxation for the POP (1). Then we explain two formats to express polynomials and POPs in Section 3. Section 4 presents an execution example of our software. Section 5 contains details of top-level functions, and Section 6 is devoted to explain the parameters that are passed to those top-level functions. Section 7 includes a brief explanation of utility functions, and the concluding remarks follow in Section 8.

The structure of the software package SparsePOP is shown in Fig.1. The functions readGMS.m, sparsePOP.m and printSolution.m are top-level MATLAB functions. Note that the function name is sparsePOP.m, while the package name is SparsePOP.

As will be seen in Section 3, SparsePOP accepts two formats of polynomial optimization problems. The first one is GAMS scalar format which is more readable for humans, and the other is the SparsePOP format which is a set of MATLAB datatypes designed exclusively for SparsePOP. In fact, the main function sparsePOP.m only accepts the SparsePOP format, and readGMS.m works as a filter from the GAMS scalar format to the sparsePOP format. Once the POP (1) is solved by sparsePOP, the solution can be written with printSolution.m.

We recommend to use 'solveGMSproblem' for GAMS scalar format problems. Typing 'solveGMSproblem' in MATLAB environment invokes readGMS.m, sparsePOP.m, and printSolution.m in this order. Small-sized example problems described in the GAMS scalar format from [2] with lower and upper bounds added are stored in the directory gms. Any of the problems in the directory gms can be solved by changing the value of problemName in the function solveGMSproblem.m to the target file name using an editor.

If a user prefers providing input directly to the function sparsePOP.m, then modify the given function solveExample.m according to the SparsePOP format. Printing solution information after executing is also possible by calling the function printSolution.m.

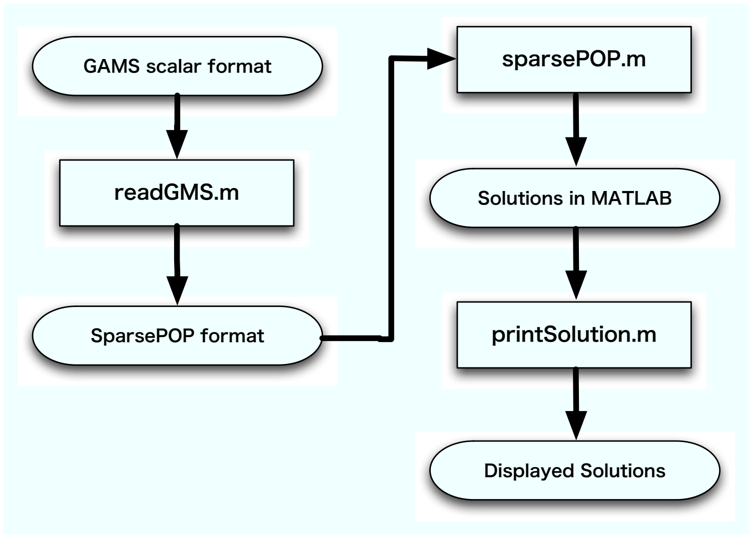

Figure 1: The structure of SparsePOP

2 SDP relaxation of POPs using csp graphs

In this section we briefly explain the sparse SDP relaxation of the POP (1) used in SparsePOP. See [10] for more details.

Let $N = \{1, 2, \ldots, n\}$. For every $k = 1, 2, \ldots, m$, let $F_k = \{i \in N : \alpha_i \geq 1 \text{ for some } \boldsymbol{\alpha} \in \mathcal{F}_k\}$. Then, a graph $G = (N, E)$ that represents the sparsity structure of the POP (1) is constructed as folllows. A pair $\{i,\ j\}$ with $i \neq\ j$ selected from the node set N is an edge or $\{i,\ j\} \in E$ if and only if either there is an $\boldsymbol{\alpha} \in \mathcal{F}_0$ such that $\alpha_i > 0$ and $\alpha_j > 0$ or $i,\ j \in F_k$ for some $k = 1, 2, \ldots, m$. The graph $G(N, E)$ is called *a correlative sparsity pattern (csp) graph.*

By construction, each F_k is a clique of $G = (N, E)$ $(k = 1, 2, \ldots, m)$. Then we make a chordal extension $G(N, E')$ of $G = (N, E)$, and let $C_1, C_2, \ldots, C_p$ be the maximum cliques of (N, E'). Note that $C_1, C_2, \ldots, C_p$ can easily be calculated because (N, E') is a chordal graph.

For every $C \subset N$ and $\psi \in \mathbb{Z}_+$, define

$$\mathcal{A}^C_\psi \quad = \quad \left\{ \boldsymbol{\alpha} \in \mathbb{Z}^n_+ : \alpha_j = 0 \text{ if } j \notin C,\ \sum_{i \in C} \alpha_i \leq \psi \right\},$$

and let $\boldsymbol{u}(\boldsymbol{x}, \mathcal{A}^C_\psi)$ denote a column vector consisting of the elements $\boldsymbol{x}^{\boldsymbol{\alpha}}$ $(\boldsymbol{\alpha} \in \mathcal{A}^C_\psi)$. We assume that the elements $\boldsymbol{x}^{\boldsymbol{\alpha}}$ $(\boldsymbol{\alpha} \in \mathcal{A}^C_\psi)$ in the vector $\boldsymbol{u}(\boldsymbol{x}, \mathcal{A}^C_\psi)$ are arranged according to the increasing order of $\boldsymbol{\alpha}$'s lexicographically. Note that $\mathbf{0} \in \mathcal{A}^C_\psi$ even when $C = \emptyset$; hence the first element of the column vector $\boldsymbol{u}(\boldsymbol{x}, \mathcal{A}^C_\psi)$ is always $\boldsymbol{x}^{\mathbf{0}} = 1$.

For every $k = 1, 2, \ldots, m$, let $\omega_k = \lceil \deg(f_k(\boldsymbol{x}))/2 \rceil$,

$$\omega_{\max} = \max\{\omega_k : k = 0, 1, \ldots, m\}, \tag{2}$$

and let $\widetilde{C}_k$ be the union of some of the maximal cliques $C_1, C_2, \ldots, C_p$ of $G(N, E')$ such that $F_k \subset \widetilde{C}_k$.

When we derive a primal SDP relaxation, we first transform the POP (1) into an equivalent polynomial SDP (PSDP)

$$\left.\begin{array}{ll}\text{minimize} & f_0(\boldsymbol{x}) \\ \text{subject to} & \boldsymbol{u}(\boldsymbol{x}, \mathcal{A}^{\widetilde{C}_k}_{\omega-\omega_k})\boldsymbol{u}(\boldsymbol{x}, \mathcal{A}^{\widetilde{C}_k}_{\omega-\omega_k})^T f_k(\boldsymbol{x}) \succeq \boldsymbol{O} \ (k = 1, 2, \ldots, m), \\ & \boldsymbol{u}(\boldsymbol{x}, \mathcal{A}^{C_\ell}_{\omega})\boldsymbol{u}(\boldsymbol{x}, \mathcal{A}^{C_\ell}_{\omega})^T \succeq \boldsymbol{O} \ (\ell = 1, 2, \ldots, p).\end{array}\right\} \quad (3)$$

Here $\boldsymbol{B} \succeq \boldsymbol{O}$ denotes that a real symmetric matrix $\boldsymbol{B}$ is positive semidefinite. The matrices $\boldsymbol{u}(\boldsymbol{x}, \mathcal{A}^{\widetilde{C}_k}_{\omega-\omega_k})\boldsymbol{u}(\boldsymbol{x}, \mathcal{A}^{\widetilde{C}_k}_{\omega-\omega_k})^T$ $(k = 1, 2, \ldots, m)$ and $\boldsymbol{u}(\boldsymbol{x}, \mathcal{A}^{C_\ell}_{\omega})\boldsymbol{u}(\boldsymbol{x}, \mathcal{A}^{C_\ell}_{\omega})^T$ $(\ell = 1, 2, \ldots, p)$ are positive semidefinite symmetric matrices of rank one for any $\boldsymbol{x} \in \mathbb{R}^n$, and have the element 1 in their upper left corner. These ensure the equivalence between the POP (1) and the PSDP (3) above.

Since the objective function of the PSDP (3) is a real-valued polynomial and the left hand side of the matrix inequality constraints of the PSDP (3) are real symmetric matrix valued polynomials, we can rewrite the PSDP (3) as

$$\text{minimize} \quad \sum_{\boldsymbol{\alpha}\in\widetilde{\mathcal{F}}} \tilde{c}_0(\boldsymbol{\alpha})\boldsymbol{x}^{\boldsymbol{\alpha}} \quad \text{subject to} \quad \boldsymbol{M}(\boldsymbol{0}, \omega) + \sum_{\boldsymbol{\alpha}\in\widetilde{\mathcal{F}}} \boldsymbol{M}(\boldsymbol{\alpha}, \omega)\boldsymbol{x}^{\boldsymbol{\alpha}} \succeq \boldsymbol{O}.$$

for some $\widetilde{\mathcal{F}} \subset \mathbb{Z}^n_+$, some $\tilde{c}_0(\boldsymbol{\alpha}) \in \mathbb{R}$ $(\boldsymbol{\alpha} \in \widetilde{\mathcal{F}})$ and some real symmetric matrices $\boldsymbol{M}(\boldsymbol{\alpha}, \omega)$ $(\boldsymbol{\alpha} \in \widetilde{\mathcal{F}} \bigcup \{\boldsymbol{0}\})$. Note that the size of the matrices $\boldsymbol{M}(\boldsymbol{\alpha}, \omega)$ $(\boldsymbol{\alpha} \in \widetilde{\mathcal{F}} \bigcup \{\boldsymbol{0}\})$ and the number of variables $y_{\boldsymbol{\alpha}}$ $(\boldsymbol{\alpha} \in \widetilde{\mathcal{F}})$ are determined by the parameter ω. Each monomial $\boldsymbol{x}^{\boldsymbol{\alpha}}$ is replaced by a single real variable $y_{\boldsymbol{\alpha}}$, and we have an SDP relaxation problem of the (1):

$$\text{minimize} \quad \sum_{\boldsymbol{\alpha}\in\widetilde{\mathcal{F}}} \tilde{c}_0(\boldsymbol{\alpha})y_{\boldsymbol{\alpha}} \quad \text{subject to} \quad \boldsymbol{M}(\boldsymbol{0}, \omega) + \sum_{\boldsymbol{\alpha}\in\widetilde{\mathcal{F}}} \boldsymbol{M}(\boldsymbol{\alpha}, \omega)y_{\boldsymbol{\alpha}} \succeq \boldsymbol{O}. \quad (4)$$

The problem (4) is a SDP relaxation, which can be solved by the standard SDP solvers. SparsePOP can call SeDuMi [7] to solve the resulting SDP (4) and/or outputs the data of (4) in the SDPA sparse format.

Let $\hat{\zeta}_\omega$ denote the optimal objective value of the SDP (4), and $y^\omega_{\boldsymbol{\alpha}}$ $(\boldsymbol{\alpha} \in \widetilde{\mathcal{F}})$ an optimal solution of the SDP (4). We then extract $y^\omega_{\boldsymbol{\alpha}}$ $(\boldsymbol{\alpha} \in \{\boldsymbol{e}^i : i \in N\})$, which are induced from the variable x_i $(i \in N)$ in the original POP (1), to form an approximate optimal solution $\boldsymbol{x}^\omega$ of the POP (1):

$$\boldsymbol{x}^\omega \ = \ (x^\omega_1, x^\omega_2, \ldots, x^\omega_n), \ x^\omega_i = y_{\boldsymbol{e}^i} \ (i \in N).$$

The relation $\zeta_\omega \leq \zeta_{\omega'} \leq \zeta^* \leq f_0(\boldsymbol{x})$ holds if $\omega_{\max} \leq \omega \leq \omega'$ and if $\boldsymbol{x} \in \mathbb{R}^n$ is a feasible solution of the POP (1). (Recall that ζ^* denotes the optimal value of the POP (1)). Therefore, if $f_0(\boldsymbol{x}^\omega) - \zeta_\omega$ is zero (or sufficiently small) and if $\boldsymbol{x}^\omega$ satisfies (or approximately satisfies) the constraints $f_k(\boldsymbol{x}) \geq 0$ $(k = 1, 2, \ldots, m)$, then $\boldsymbol{x}^\omega$ is an optimal solution (or an approximate optimal solution, resepctively) of the POP (1).

The parameter ω, which corresponds to the parameter param.relaxOrder used in SparsePOP, will determine both the quality and the size of the SDP relaxation (4) of the POP (1). In theory, we obtain an approximate optimal solution of the POP (1) with higher quality (more precisely, not lower quality) by solving the SDP relaxation (4) as we choose a larger

ω. In practice, however, the size of the SDP relaxation (4) becomes larger and the cost of solving the SDP relaxation (4) increases very rapidly. Therefore, we usually start the SDP relaxation (4) with $\omega = \omega_{\max}$, and successively increase ω by 1 if the approximate solution obtained is not accurate.

In SparsePOP, we can add polynomial equality constraints $f_k(\boldsymbol{x}) = 0$ $(k = m+1, \ldots, m')$ as well as lower and upper bounds for the variables x_i $(i \in N)$ to the POP (1). SparsePOP also deals with an unconstrained POP that only objective polynomial $f_0(\boldsymbol{x})$ is specified.

3 Representation of polynomials

The description of polynomials in the objective function and constraints is needed for SparsePOP. The polynomials can be represented in two ways. The one is to use the GAMS scalar format and convert the GAMS scalar format data into the SparsePOP format data by the function readGMS.m. The other is to directly describe the objective and constraint polynomials in terms of the SparsePOP format.

As an example, we consider an inequality constrained POP with three variables x_1, x_2 and x_3:

$$\left.\begin{array}{ll} \text{minimize} & -2x_1 + 3x_2 - 2x_3 \\ \text{subject to} & 6x_1^2 + 3x_2^2 - 2x_2x_3 + 3x_3^2 - 17x_1 + 8x_2 - 14x_3 \geq -19, \\ & x_1 + 2x_2 + x_3 \leq 5, \\ & 5x_2 + 3x_3 \leq 7, \\ & 0 \leq x_1 \leq 2,\ 0 \leq x_2 \leq 2,\ 0.5 \leq x_3 \leq 3. \end{array}\right\} \qquad (5)$$

This problem is expressed in the GAMS scalar format as follows:

```
* example.gms
* This file contains the description of the problem specified
* in solveExample.m, and is located in the directory gms.
* See Section 2 of the manual.

* To obtain a tight bound for the optimal objective value by the function
* sparsePOP.m, set the parameter param.relaxOrder = 3.

* The description consists of 5 parts except comment lines
* starting the character '*'. The 5 parts are:
* < List of the names variables >
* < List of the names of nonnegative variables >
* < List of the names of constraints >
* < The description of constraints >
* < Lower and upper bounds of variables  >

* < List of the names variables >
Variables  x1,x2,x3,objvar;
* 'objvar' represents the value of the objective function.
* < List of the names of nonnegative variables >
Positive Variables x1,x2;

* < List of the names of constraints >
Equations  e1,e2,e3,e4;

* < The description of constraints >
* Each line should start with the name of a constraint in the list of names
* of constraints,  followed * by '.. '. The symbols '*', '+', '-', '^', '=G='
* (not less than), '=E=' (equal to) and '=L=' can be used in addition to the
* variables in the list of the names of variables and real numbers. One
* constraint can be described in more than one lines; for example,
* e2..    - 17*x1 + 8*x2 - 14*x3 +6*x1^2 + 3*x2^2 - 2*x2*x3 + 3*x3^2 =G= -19;
* is equivalent to
* e2..     - 17*x1 + 8*x2 - 14*x3 +6*x1^2
*                + 3*x2^2 - 2*x2*x3 + 3*x3^2 =G= -19;
* Note that the first letter of the line can not be '*' except comment lines.

* minimize objvar = -2*x1 +3*x2 -2*x3
e1..    2*x1 - 3*x2 + 2*x3 + objvar =E= 0;

* 6*x1^2 + 3*x2^2 -2*x2*x3 +3*x3^2 -17*x1 +8*x2 -14*x3 >= -19
e2..    - 17*x1 + 8*x2 - 14*x3 +6*x1^2 + 3*x2^2 - 2*x2*x3 + 3*x3^2 =G= -19;

* x1 + 2*x2 + x3 <= 5
e3..    x1 + 2*x2 + x3 =L= 5;

* 5*x2 + 2*x3 <= 7
e4..    5*x2 + 2*x3 =L= 7;

* < Lower and upper bounds on variables  >
* Each line should contain exactly one bound;
* A line such that 'x3.up = 3; x3.lo = 0.5;' is not allowed.

x1.up = 2;
x2.up = 1;
x3.up = 3;
x3.lo = 0.5;
```

If polynomials in the GAMS scalar format include parentheses, we need to specify symbolicMath $= 1$ as the second input argument of readGMS.m so that the function readGMS.m can expand them by using the Symbolic Math Toolbox in MATLAB. When the Symbolic Math Toolbox is not available, parentheses need to be expanded manually before applying readGMS.m.

Alternatively, a POP can be described directly using the SparsePOP format. A polynomial class is defined for this purpose as follows:

poly.typeCone	=	1	if $f(\boldsymbol{x}) \in \mathbb{R}[\boldsymbol{x}]$ is used as an objective function,
	=	1	if $f(\boldsymbol{x}) \in \mathbb{R}[\boldsymbol{x}]$ is used as an inequality constraint $f(\boldsymbol{x}) \geq 0$,
	=	-1	if $f(\boldsymbol{x}) \in \mathbb{R}[\boldsymbol{x}]$ is used as an equality constraint $f(\boldsymbol{x}) = 0$.
poly.degree	=		the degree of $f(\boldsymbol{x})$.
poly.dimVar	=		the dimension of the variable vector $\boldsymbol{x}$.
poly.noTerms	=		the number of terms of $f(\boldsymbol{x})$.
poly.supports	=		a set of supports of $f(\boldsymbol{x})$, a poly.noTerms $\times$ poly.dimVar matrix.
poly.coef	=		coefficients, a poly.noTerms dimensional column vector.

We use the name "objPoly" for the objective polynomial function $f_0(\boldsymbol{x})$ and "ineqPolySys$\{j\}$" $(j = 1, 2, \ldots, m)$ for the polynomials $f_j(\boldsymbol{x})$ $(j = 1, 2, \ldots, m)$ in the constraints. We show how the problem (5) is described using the polynomial class.

```
% Input data for 'example.gms'
% objPoly
% -2*x1 +3*x2 -2*x3
        objPoly.typeCone = 1;
        objPoly.dimVar   = 3;
        objPoly.degree   = 1;
        objPoly.noTerms  = 3;
        objPoly.supports = [1,0,0; 0,1,0; 0,0,1];
        objPoly.coef     = [-2; 3; -2];
% ineqPolySys
% 19 -17*x1 +8*x2 -14*x3 +6*x1^2 +3*x2^2 -2*x2*x3 +3*x3^2 >= 0,
        ineqPolySys{1}.typeCone = 1;
        ineqPolySys{1}.dimVar   = 3;
        ineqPolySys{1}.degree   = 2;
        ineqPolySys{1}.noTerms  = 8;
        ineqPolySys{1}.supports = [0,0,0; 1,0,0; 0,1,0; 0,0,1; ...
                                   2,0,0; 0,2,0; 0,1,1; 0,0,2];
        ineqPolySys{1}.coef     = [19; -17; 8; -14; 6; 3; -2; 3];
%
% 5 -x1 -2*x2 -x3  >= 0.
        ineqPolySys{2}.typeCone = 1;
        ineqPolySys{2}.dimVar   = 3;
        ineqPolySys{2}.degree   = 1;
        ineqPolySys{2}.noTerms  = 4;
        ineqPolySys{2}.supports = [0,0,0; 1,0,0; 0,1,0; 0,0,1];
        ineqPolySys{2}.coef     = [5; -1; -2; -1];
%
% 7 -5*x2 -2*x3 >= 0.
        ineqPolySys{3}.typeCone = 1;
        ineqPolySys{3}.dimVar   = 3;
        ineqPolySys{3}.degree   = 1;
        ineqPolySys{3}.noTerms  = 3;
        ineqPolySys{3}.supports = [0,0,0; 0,1,0; 0,0,1];
        ineqPolySys{3}.coef     = [7; -5; -2];
% lower bounds for variables x1, x2 and x3
        lbd = [0,0,0.5];
% upper bounds for variables x1, x2 and x3
        ubd = [2,1,3];
```

4 Sample execution of the sparsePOP.m

The sparsePOP.m can be executed by typing ‘solveExample’ in a MATLAB environment. It provides an optimal solution of the given problem ‘example.gms’, which is described in the function solveExample.m in the SparsePOP format. In addition, some computational data such as the parameters used, the size of SDP relaxation, information on the approximate solution and errors, and cpu time are printed by printSoltion.m.

```
>> solveExample
SeDuMi 1.05R5 by Jos F. Sturm, 1998, 2001-2003.
Alg = 2: xz-corrector, theta = 0.250, beta = 0.500
eqs m = 83, order n = 271, dim = 1461, blocks = 11
nnz(A) = 1737 + 0, nnz(ADA) = 6889, nnz(L) = 3486
 it :     b*y       gap    delta  rate   t/tP*  t/tD*   feas cg cg
  0 :            3.46E+01 0.000
  1 :   3.75E-01 2.30E+01 0.000 0.6648 0.9000 0.9000   3.75  1  1
  2 :   7.01E-01 9.81E+00 0.000 0.4262 0.9000 0.9000   2.46  1  1
     .          .          .          .          .          .
     .          .          .          .          .          .
 25 :   1.09E+00 1.19E-09 0.000 0.3539 0.9000 0.9000   1.00  3  3
iter seconds digits       c*x               b*y
 25      1.0   8.9  1.0863210013e+00  1.0863209997e+00
|Ax-b| =   4.1e-10, [Ay-c]_+ =   1.5E-11, |x|=  7.2e+00, |y|=  3.0e+00
Max-norms: ||b||=1, ||c|| = 1,
Cholesky |add|=1, |skip| = 0, ||L.L|| = 500000.

## Computational Results by sparsePOP.m with SeDuMi ##
## Printed by printSolution.m ##
# Problem File Name  = example.gms
# parameters:
  boundSW            = 1
  complementaritySW  = 0
  detailedInfFile    = 0
  eqTolerance        = 0.00e+00
  multiCliquesFactor = 1.00e+00
  perturbation       = 0.00e+00
  reduceMomentMatSW  = 1
  relaxOrder         = 3
  scalingSW          = 1
  sdpaDataFile       =
  SeDuMiSW           = 1
  SeDuMiOutFile      = 1
  sparseSW           = 1
# SDP solved:
  size of A          = [83, 1460]
  no nonzeros in A   = 3052
  no of LP variables = 160
  no of SDP blocks   = 10
  max size SDP block = 20
  ave.size SDP block = 1.10e+01
# SeDuMi information:
  SeDuMiInfo.numerr  = 0
  SeDuMiInfo.pinf    = 0
  SeDuMiInfo.dinf    = 0
# Approximate optimal value information:
  SDPobjValue        = -6.517925998312e+00
  POPobjValue        = -6.517926002094e+00
  relative obj error = +5.801e-10
  POP.absError       = -7.462e-08
  POP.scaledError    = -1.777e-09
# cpu time:
  cpuTime.conversion =       0.85
  cpuTime.SeDuMi     =       1.31
  cpuTime.total      =       2.18
# Approximate optimal solution information:
  POP.xVect =
      1:+2.589630050872e-01    2:+1.260433679787e-09    3:+2.999999995960e+00
```

5 Main MATLAB functions

The main part of the software package SparsePOP consists of three MATLAB functions sparsePOP.m, readGMS.m and printSolution.m as shown in Figure 1. We give a brief description of the inputs and outputs of these functions in this section.

The input for sparsePOP.m is basically the representation of the POP (1) in the SparsePOP format. The function sparsePOP has the following declaration:

```
function [param,SDPobjValue,POP,cpuTime,SeDuMiInfo,SDPinfo] = ...
                sparsePOP(param,objPoly,ineqPolySys,lbd,ubd);
```

The objective and constraint polynomials of a POP are expressed in objPoly and ineqPolySys in the SparsePOP format as described in Section 3. The row vector of the lower bounds for x_i $(i = 1, 2, ..., n)$ is denoted by lbd. Similarly, ubd indicates the row vector of the upper bounds for x_i $(i = 1, 2, ..., n)$. If lbd and ubd are not given, the function sparsePOP.m assigns default values which are lbd$(i) = -1.0$e+10 and ubd$(i) = +1.0$e+10 in the sparsePOP.m. The first argument param contains a set of parameters that influence the behavior of SparsePOP. The details of these parameters are described in Section 6. If a user wants to use default values, then he or she can just give an empty MATLAB struct, namely, param = struct([]).

In the output, user-specified or default values for the parameters are stored in param. SDPobjValue contains a lower bound of the optimal objective value of the POP. POP has four components:

- POP.xVect: an approximate solution for the POP (1).
- POP.objValue: the value of the objective function at POP.xVect.
- POP.absError: an absolute feasibility error.
- POP.scaledError: a scaled feasibility error.

Suppose that the constraints under consideration are

$$f_i(\boldsymbol{x}) \geq 0 \ (i = 1, 2, \ldots, m), \ f_j(\boldsymbol{x}) = 0 \ (j = m+1, \ldots, m').$$

Then the absolute feasibility error at $\boldsymbol{x} \in \mathbb{R}^n$ is given by

$$\max \left\{ \min\{f_i(\boldsymbol{x}), \ 0\} \ (i = 1, 2, \ldots, m), \ -|f_j(\boldsymbol{x})| \ (j = m+1, \ldots, m') \right\},$$

and the scaled feasibility error is given by

$$\max \left\{ \min\{f_i(\boldsymbol{x})/\sigma_i(\boldsymbol{x}), \ 0\} \ (i = 1, 2, \ldots, m), \ -|f_j(\boldsymbol{x})|/\sigma_j(\boldsymbol{x}) \ (j = m+1, \ldots, m') \right\},$$

where $\sigma_i(\boldsymbol{x})$ denotes the maximum of the absolute values of all monomials of $f_i(\boldsymbol{x})$ evaluated at $\boldsymbol{x} \in \mathbb{R}^n$ if the maximum is greater than 1 or $\sigma_i(\boldsymbol{x}) = 1$ otherwise $(i = 1, 2, \ldots, m')$.

The output cpuTime reports various cpu times consumed by the execution of sparsePOP:

- cpuTime.conversion: cpu time consumed to convert the POP into its SDP relaxation.
- cpuTime.SeDuMi: cpu time consumed by SeDuMi to solve the SDP.
- cpuTime.Total: cpu time for the entire process.

SDPinfo has information of the SDP relaxation problem solved by SeDuMi.

- SDPinfo.rowSizeA: the number of rows of the coefficient matrix of the primal SDP
- SDPinfo.colSizeA: the number of columns of the coefficient matrix of the primal SDP
- SDPinfo.nonzeroInA: the number of nonzeros of the coefficient matrix A.
- SDPinfo.noOfLPvariables: the number of LP variables of the primal SDP.
- SDPinfo.SDPblock: the row vector of sizes of SDP blocks.

Finally, SeDuMiInfo contains SeDuMiInfo.numerr, SeDuMiInfo.pinf and SeDuMi.dinf, which are equivalent to info.numerr, infor.pinf and info.dinf in SeDuMi output. See [7] for the details.

The function readGMS has the following declaration:

```
function [objPoly,ineqPolySys,lbd,ubd] = readGMS(fileName,symbolicMath);
```

The first argument fileName that is a string of MATLAB is the name of the file where a problem is described in the GAMS scalar format. The second input argument symbolicMath is optionally set to be 1 if the Symbolic Math Tool is available. The output consisting of objPoly, ineqPolySys, lbd, and ubd is a POP data in the SparsePOP format, and can be passed to sparsePOP.

The function printSolution for printing the result has the following declaration.

```
function printSolution(fileId,printLevel,dataFileName,param,SDPobjValue,...
        POP,cpuTime,SeDuMiInfo,SDPinfo);
```

The meaning of each input is as follows.

- fileId: the fileId where output goes. If this is 1, then the result is displayed on the screen (i.e., the standard output). If a user wants to write the result to a file, he or she must first open it in the writable mode and give its fileId.
- printLevel: a larger value of printLevel gives more detailed description of the result. 2 for the default.
- dataFileName: the name of the problem solved.

The rest of the input arguments, i.e., param, SDPobjValue, POP, cpuTime, SeDuMiInfo, and SDPinfo must be the outputs of sparsePOP.

6 Parameters

In addition to objPoly, ineqPolySys, lbd and ubd that describe a POP, the MATLAB function sparsePOP.m involves param as an input argument. It is a structure consisting of many parameters that control the behavior and performance of the function. The list of parameters is shown below.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Default values of parameters; % some possible values;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% param.boundSW = 1; % 0;
% param.complementaritySW = 0; % 1;
% param.detailedInfFile = []; % 'details.out';
% param.eqTolerance = 0.0; % 1.0e-6;
% param.multiCliquesFactor = 1; % objPoly.dimVar;
% param.perturbation = 0.0; % 1.0e-6;
% param.reduceMomentMatSW = 1; % 0;
% param.relaxOrder = the minilmal relaxation order;
%                       or any positive integer;
param.relaxOrder = 3; % to solve example.gms;
% param.scalingSW = 1; % 0;
% param.sdpaDataFile = []; % 'test.dat-s';
% param.SeDuMiOutFile = 1;
%       1 for screen;
%       0 or [] for none;
%       file name, e.g., 'SeDuMi.out' for writing output to a file.
% param.SeDuMiSW = 1;
%       0 for information on the SDP without executing SeDuMi;
% param.sparseSW = 1;
%       0 for dense SDP relaxation;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

These parameters can be divided into three categories:

1. parameters to control the basic relaxation scheme.
2. switches to handle numerical difficulties.
3. parameters for SDP solvers.

6.1 Parameters to control the relaxation method

We first need to specify the relaxation order param.relaxOrder $= \omega \geq \omega_{\max}$ to excecute the sparse and dense SDP relaxation for a POP, where $\omega_{\max}$ is defined by (2) for a POP of the form (1). The default is param.relaxOrder $= \omega_{\max}$.

The type of relaxation can be chosen by setting param.sparseSW $= 0$ for the dense relaxation based on [5] or $= 1$ for the sparse relaxation based on [10]. The sparsity of the sparse relaxation is varied by param.multiCliquesFactor. Suppose that param.sparseSW$= 1$; otherwise this parameter is not relevant. The purpose of this parameter is to strengthen the sparse relaxation by taking the union of some of the maximal cliques C_ℓ $(\ell = 1, 2, \ldots, p)$ of a chordal extension $G = (N, E')$ of the csp graph induced from the POP (1) for $\widetilde{C}_k$ $(k = 1, 2, \ldots, m)$. Let $\rho_{\max}$ denote the maximum over $\sharp C_\ell$ $(\ell = 1, 2, \ldots, p)$, where $\sharp C_\ell$ denotes the cardinality of C_ℓ. Recall that $F_k = \{i : x_i \text{ appears in } f_k(x) \geq 0\}$. Let $J_k = \{\ell : F_k \subset C_\ell\}$. Take one clique from C_ℓ $(\ell \in J_k)$ for $\widetilde{C}_k$. Add another clique from C_ℓ $(\ell \in J_k)$ to $\widetilde{C}_k$ if $\sharp\left(\widetilde{C}_k \bigcup C_\ell\right)$ does not exceed param.multiCliquesFactor $\times \rho_{\max}$. Repeat this procedure to obtain the union $\widetilde{C}_k$ of some cliques from C_ℓ $(\ell \in J_k)$. If param.multiCliquesFactor $= 0$, then $\widetilde{C}_k$ consists of a single clique C_ℓ for some $\ell \in J_k$. If param.multiCliquesFactor $= n$, then $\widetilde{C}_k$ consists of the union of all C_ℓ $(\ell \in J_k)$. The default value is 1, which means that the cardinality of $\widetilde{C}_k$ is bounded by $\rho_{\max}$.

6.2 Switches for techniques to reduce numerical difficulties

Because the POP (1) is basically a hard optimization problem, we often encounter numerical difficulties in solving its SDP relaxation, and/or have an inaccurate approximate solution. The switches described in this subsection are intended to reduce the numerical difficulties and improve the accuracy of an obtained solution.

With param.scalingSW$= 1$, the objective polynomial, constraint polynomials, lower and upper bounds are scaled so that the maximum of $\{|\text{lower bound of } x_j|, |\text{upper bound of } x_j|\} =$

1 ($j \in J$) and that the maximum absolute value of the coefficients of all monomials in each polynomial is 1, where J denotes the set of indices j such that the variable x_j has finite lower and upper bounds; $-1.0\text{e}{+}10 < \text{lbd}(j) \leq \text{ubd}(j) < 1.0\text{e}{+}10$. This scaling technique is very effective to improve the numerical stability when solving the resulting SDP relaxation. The default is param.scalingSW = 1.

Appropriate bounds are added for all linearized variables $y_{\boldsymbol{\alpha}}$ ($\boldsymbol{\alpha} \in \widetilde{\mathcal{F}}$) if param.boundSW = 1, otherwise, no bounds are added to $y_{\boldsymbol{\alpha}}$. The default is param.boundSW= 1. In particular, when every variable x_j is scaled such that $\text{lbd}(j) = 0$ and $\text{ubd}(j) = 1$ ($j = 1, 2, \ldots, n$), the bounds $0 \leq y_{\boldsymbol{\alpha}} \leq 1$ ($\boldsymbol{\alpha} \in \widetilde{\mathcal{F}}$) are added. Empirically, we know such a bounding is very effective to improve the numerical stability in solving the SDP relaxation. Therefore, we recommend users to modify a POP so that every variable x_j is nonnegative and has a finite positive upper bound; then the desired scaling and bounding of variables $y_{\boldsymbol{\alpha}}$ ($\boldsymbol{\alpha} \in \widetilde{\mathcal{F}}$) are performed in the function sparsePOP.m by taking param.scalingSW = 1 and param.boundSW = 1.

The parameter param.eqTolerance is used to convert every equality constraint into two inequaly constraints; if 1.0e-10 < param.eqTolerance, then we replace each equality constraint $f(x) = 0$ by $f(x) \geq 0$ and $-f(x) \geq -$param.eqTolerance. When SeDuMi shows a numerical difficulty in solving the SDP relaxation of a POP with equality constraints, this technique with 1.0e-3 $\leq$ param.eqTolerance $\leq$ 1.0e-7 often provides a more stable SDP relaxation problem that can be solved by SeDuMi. The default is param.eqTolerance = 0, *i.e.* the equality constraints are kept as given.

Perturbing the objective polynomial to compute an optimal solution of a degenerate POP is described in Section 5.1 of [10]. The parameter param.perturbation is used for this purpose. If 1.0e-10 < param.perturbation, then we modify the objective polynomial $f(x)$ as $f(\boldsymbol{x}) + p^t\boldsymbol{x}$, where $0 \leq p_i \leq$ param.perturbation. Otherwise, we do not perturb. The default value for param.perturbation is 0.0, *i.e.*, no perturbation to the objective polynomial is performed.

When the SDP relaxation is too large to be solved, param.reduceMomentMatSW may help. If param.reduceMomentMatSW = 1, then sparsePOP.m eliminates redundant elements of $\mathcal{A}^{C_\ell}_{\omega}$ ($\ell = 1, 2, \ldots, p$) in the PSDP (3) using the method proposed in the paper [4]. See also [10].

When the complementarity condition exists in the constraints of a POP to be solved, we can set param.complementaritySW = 1. Suppose that $x_i x_j = 0$ appears as an equality constraint. Then any variable y_α corresponding to a monomial $\boldsymbol{x}^{\boldsymbol{\alpha}}$ such that $\alpha_i \geq 1$ and $\alpha_j \geq 1$ is set to zero and eliminated from the PSDP (3). The default is param.complementaritySW = 0.

6.3 Parameters for SDP Solvers

The function sparsePOP.m can provide three kinds of output for the SDP relaxation problem: information on the problem itself such as the size and the nonzero elements of the constraint matrix of the problem, data on an optimal solution of the problem obtained by SeDuMi, and SDPA sparse format data of the problem. SeDuMi should be called in the function sparsePOP.m to have information on the problem itself as well as data on the obtained optimal solution, which can be done by setting param.SeDuMiSW= 1. The parameter param.SeDuMiOutFile is used in connection with the parameter param.SeDuMiSW= 1. The default value for param.SeDuMiOutFile= 1 is used to display the output from SeDuMi on the screen. If we assign the name of a file such as param.SeDuMiOutFile = 'SeDuMi.out', the output from SeDuMi is written in the file. Choose param.SeDuMiOutFile = 0 to display no information from SeDuMi. The value 0 for param.SeDuMiSW is used to have information on the problem printed without solving the SDP relaxation problem. The default value for param.SeDuMiSW is 1. In either of the cases param.SeDuMiSW= 1 or 0, we can store detailed information of the POP and its SDP relaxation in a file specified by the parameter param.detailedInfFile; for example, param.detailedInfFile = 'details.out'. The default is param.detailedInfFile = [ ], *i.e.*, no detailed information is printed. To get SDPA sparse format data of the SDP relaxation problem, we need to assign the name of a file for SDPA sparse format data to the parameter param.sdpaDataFile; for example, param.sdpaDataFile = 'test.dat-s'. Then, the SDP relaxation problem can be solved later by using some software packages such as SDPA [9] and SDPT3 [8] with the SDPA sparse format input file 'test.dat-s'. The default is param.sdpaDataFile = [ ], *i.e.*, no SDPA sparse format data is created.

7 MATLAB functions used in the function sparsePOP.m

In Table 7, we give a brief description of utility programs in sparseSOS.

8 Concluding Remarks

We have described how the software package SparsePOP can be used. It should be mentioned that the size of POPs that can be handled by the SparsePOP currently can go up to $n = 30$ if the lower relaxation order such as 1 or 2 is used. It also depends on the sparsity of polynomials in a POP to be solved. See the paper [10] for extensive numerical results.

References

[1] GAMS HomePage, `http://www.gams.com/`

[2] GLOBAL Library, `http://www.gamsworld.org/global/globallib.htm`

[3] S. Kim, M. Kojima and H. Waki, "Generalized Lagrangian duals and sums of square relaxation of sparse polynomial optimization problems", to appear in *SIAM J. Optimization.*

[4] M. Kojima, S. Kim and H. Waki, "Sparsity in sums of squares of polynomials", to appear in *Mathematical Programming.*

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

[6] P. A. Parrilo, "Semidefinite programming relaxations for semialgebraic problems", *Mathematical Programming*, **96** (2003) 293-320.

addBoundToPOP.m	Add bounds to $y_{\boldsymbol{\alpha}}$ ($\boldsymbol{\alpha} \in \widetilde{\mathcal{F}}$).
boundToIneqPolySys.m	Include lower and upper bounds, lbd and ubd, in ineqPolySys.
checkPOP.m	Verify input data objPoly, ineqPolySys, lbd and ubd describing a POP in the sparsePOP format.
evalPolynomials.m	Evaluate a polynomial or a polynomial system.
genApproxSolution.m	Compute an approximate solution of POP using the optimal value obtained from the SDP relaxation.
genBasisIndices.m	Form basisIndices used for basisSupports from clique.
genBasisSupports.m	Generate basisSupports based on basisIndices and relaxOrder.
genClique.m	Generate clique based the correlative sparsity pattern of objective and constraint polynomials of a POP.
listupAllSupports.m	List up all support vectors used in the PSDP.
monomialSort.m	Sort monomials.
perturbObjPoly.m	Perturb objective polynomial with a given small number.
PSDPtoLSDP	Generate an SDP relaxation problem in the SDPA sparse format.
relax1EqTo2Ineqs.m	Convert every equality constraint into two inequality constraints.
saveOriginalPOP.m	Save the original POP before being modified.
scalingPOP.m	Scale objPoly, inEqPolySys, lbd and ubd of a POP.
SDPAtoSeDuMi.m	Convert SDPA sparse format data into SeDuMi format data.
setParameter.m	Set default values for parameters not specified by the user.
simplifyPolynomial.m	Simplify a polynomial or a polynomial system.
writeBasisIndices.m	Print basisIndices used for basisSupports.
writeBasisSupports.m	Print basisSupports.
writeClique.m	Print clique.
writeParameters.m	Print param.
writePolynomials.m	Print a polynomial or a polynomial system in the sparsePOP format.
writePOP.m	Print a POP expressed in the sparsePOP format.
writeResults.m	Print an approximate optimal solution and errors.
writeSDPAformatData.m	Print SDPA sparse format data on an LSDP.
writeSeDuMiInputData.m	Print SeDuMi format data on an LSDP.

[7] J. F. Strum, "SeDuMi 1.02, a MATLAB toolbox for optimization over symmetric cones", em Optimization Methods and Software, **11 & 12** (1999) 625-653.

[8] R. H Tutuncu, K. C. Toh, and M. J. Todd, "Solving semidefinite-quadratic-linear programs using SDPT3", Mathematical Programming Ser. B, 95 (2003), pp. 189–217.

[9] M. Yamashita, K. Fujisawa and M. Kojima, "Implementation and evaluation of SDPA 6.0 (SemiDefinite Programming Algorithm 6.0)", *Optimization Methods and Software*, **18** (2003) 491-505.

[10] H. Waki, S. Kim, M. Kojima, and M. Muramatsu, "Sums of Squares and Semidefinite Programming Relaxations for Polynomial Optimization Problems with Structured Sparsity", B-411, Dept. of Mathematical and Computing Sciences, Tokyo Institute of Technology, Oh-Okayama, Meguro, Tokyo 152-8552 Japan, October 2004.