ISSN 1342-2804

Research Reports on Mathematical and Computing Sciences

User Manual for **SparsePOP**: a **Sparse** Semidefinite Programming Relaxation of **P**polynomial **O**ptimization **P**roblems

Hayato Waki, Sunyoung Kim,
Masakazu Kojima, Masakazu Muramatsu
and Hiroshi Sugimoto

March 2005, B–414. Revised December 2007

Department of
Mathematical and
Computing Sciences
Tokyo Institute of Technology

SERIES B: Operations Research

B-414 User Manual for **SparsePOP**: a **Sparse** Semidefinite Programming Relaxation of **P**polynomial **O**ptimization **P**roblems

Hayato Waki[⋆], Sunyoung Kim[†], Masakazu Kojima[‡], Masakazu Muramatsu[♯]
Hiroshi Sugimoto[♭]
March 2005, revised December 2007

Abstract.
SparesPOP is a MATLAB implementation of a sparse semidefinite programming (SDP) relaxation method for approximating a global optimal solution of a polynomial optimization problem (POP) proposed by Waki *et al.* The sparse SDP relaxation exploits a sparse structure of polynomials in POPs when applying "a hierarchy of LMI relaxations of increasing dimensions" by Lasserre. The efficiency of **SparsePOP** to approximate optimal solutions of POPs is thus increased, and larger scale POPs can be handled. The software package **SparesPOP**, this manual, and a test set of POPs from the literature 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. The research was supported by Kosef R01-2005-000-10271-0. *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 and by Grant-in-Aid for Scientific Research (B) 19310096. *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*

♭ Department of Mathematical and Computing Sciences, Tokyo Institute of Technology, 2-12-1 Oh-Okayama, Meguro-ku, Tokyo 152-8552 Japan.

1 Introduction

SparsePOP is a matlab package for finding global optimal solutions of polynomial optimization problems (POPs). The package is an implementation of a sparse semidefinite programming (SDP) relaxation method for POPs in [11], proposed to improve the efficiency of Lasserre's hierarchy of LMI relaxations of increasing dimensions [6]. SparsePOP exploits the sparsity of POPs in a way that it can handle POPs of larger dimensions. See also [4, 5].

A general POP is described as follows: Let $\mathbb{R}^n$ and $\mathbb{Z}^n_+$ denote the n-dimensional Euclidean space and the set of nonnegative integer vectors in $\mathbb{R}^n$, respectively. A real-valued polynomial $f_k(\boldsymbol{x})$ in $\boldsymbol{x} = (x_1, x_2, \ldots, x_n) \in \mathbb{R}^n$ is expressed 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 $\boldsymbol{\alpha} \in \mathbb{Z}^n_+$. We consider a POP in the following form:

$$\left.\begin{array}{lll} \text{minimize} & f_0(\boldsymbol{x}) & \\ \text{subject to} & f_k(\boldsymbol{x}) \geq 0 & (k = 1, 2, \ldots, \ell), \\ & f_k(\boldsymbol{x}) = 0 & (k = \ell + 1, \ldots, m), \\ & \text{lbd}_i \leq x_i \leq \text{ubd}_i & (i = 1, 2, \ldots, n), \end{array}\right\} \tag{1}$$

where $-\infty \leq \text{lbd}_i < \infty$ and $-\infty < \text{ubd}_i \leq \infty$ $(i = 1, 2, \ldots, n)$. Let ζ^* denote the optimal value of the POP (1).

The package accepts a POP as input, and outputs solution information and statistics. The main part constructs a sparse SDP relaxation of the POP and uses SeDuMi [8] to obtain an approximate global optimal solution. The structure of the software package SparsePOP is shown in Figure 1. The function sparsePOP.m is the main function of SparsePOP. Note the difference in the names of the function sparsePOP.m and the package SparsePOP. As will be shown in Section 2, sparsePOP.m accepts two different formats of a POP: the GAMS scalar format [2] that is more readable, and the SparsePOP format, a set of MATLAB data types designed exclusively for SparsePOP. If a POP is read in the GAMS scalar format, then a subfunction readGMS.m converts a GAMS scalar format of the POP to a sparsePOP format of the POP. It is followed by checking the validity of the SparsePOP format of the POP and the parameters optionally provided by the user or given by default. Then, either SDPrelaxation.m or SDPrelaxationMex.m transforms the POP into an SDP relaxation problem, and solves it with a MATLAB SDP solver SeDuMi. Once the POP (1) is solved, the solution information is written using the MATLAB function printSolution.m. We refer to the paper [11] for numerical results from SparsePOP.

The conversion from a POP to an SDP relaxation problem can be time-consuming if implemented with MATLAB. The subfunction SDPrelaxationMex.m is developed in C++ to speed up this process. SparsePOP provides an option for choosing SDPrelaxation.m or SDPrelaxationMex.m. If the C++ source programs can be compiled and linked into mex files, the SDPrelaxationMex.m is recommended. Otherwise, the value of the parameter `param.mex` should be changed from `param.mex = 1` to `param.mex = 0` in the defaultParameter.m to choose the SDPrelaxation.m.

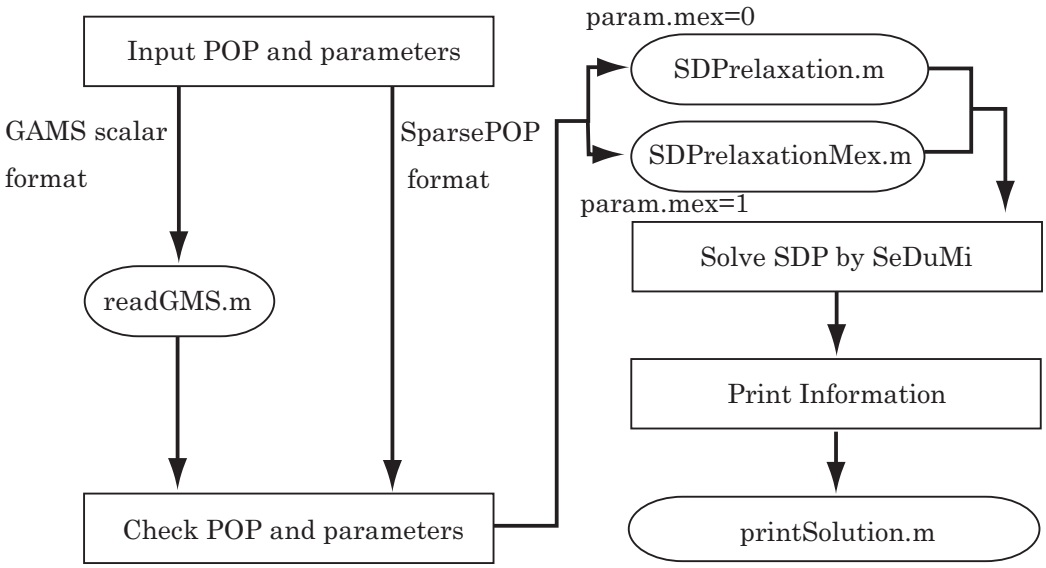

Figure 1: The structure of the main function sparsePOP.m

This manual is organized as follows: Section 2 includes the description of two formats to express polynomials and POPs. Exemplary execution is shown in Section 3. Section 4 contains the discussion of how POPs having possibly multiple optimal solutions can be treated. The main function sparsePOP.m and its subfunctions readGMS.m, SDPrelaxation.m, SDPrelaxationMex.m and printSolution.m shown in Figure 1 are described with their input and output arguments in Section 5. The parameters to the subfunctions SDPrelaxation.m and SDPrelaxationMex.m are explained in Section 6.

2 Representation of polynomial optimization problems

Polynomials in the objective function and constraints of a POP can be described in two ways to be read by SparsePOP. If the GAMS scalar format is chosen, data in the GAMS scalar format is converted into data in the SparsePOP format by the function readGMS.m. Or, we can directly describe the objective and constraint polynomials in terms of the SparsePOP format. As an illustrative example, we consider an inequality-equality constrained POP with three variables $x_1,\ x_2$ and x_3:

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

2.1 The GAMS scalar format

The GAMS scalar format describing the POP (2) is:

```
* example1.gms
* This file contains the GAMS scalar format description of the problem
*
* minimize objvar = -2*x1 +3*x2 -2*x3
* subject to
*       x1^2 + 3*x2^2 -2*x2*x3 +3*x3^2 -17*x1 +8*x2 -14*x3 >= -19,
*       x1 + 2*x2 + x3 <= 5,
*       0 <= x1 <= 2, 0 <= x2 <= 1.
*
* To solve this problem by sparsePOP.m:
* >> param.relaxOrder = 3;
* >> sparsePOP('example1.gms',param);
*
* This problem is also described in terms of the SparsePOP format
* in the file example1.m.  See Section 3 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 of 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 of 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=' (not larger than) 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 a 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 =E= 7;

* < Lower and upper bounds on variables  >
* Each line should contain exactly one bound;
* For 0.5 <= x3 <= 2, we set
* x3.lo = 0.5;
* x3.up = 2;
* A line such that 'x3.lo = 0.5; x3.up = 2;' is not allowed.

* x1 <= 2
x1.up = 2;

* x2 <= 1
x2.up = 1;

* end of example1.gms
```

Many examples of the GAMS scaler format of POPs can be found in the directory

example/GMSformat/,

which are from [3]; we have added and/or modified lower and upper bounds of some of the problems.

If polynomials in the GAMS scalar format include parentheses, then param.symbolicMath = 1 should be set. The value 1, in sparsePOP.m, is sent to the function readGMS.m where symbolic expansion takes place using the Symbolic Math Toolbox. If the Symbolic Math Toolbox is not available, expanded polynomials should be prepared in the GAMS scalar format and set param.symbolicMath = 0.

We note that the GAMS scalar format is different from the GAMS format. The GAMS scalar format is produced by the program "convert" from the GAMS format. In the GAMS

scalar format that can be used in SparsePOP, the right-hand side of an inequality or an equality should be a single constant, "objvar" is reserved as a keyword to represent the value of the objective function, only multivariate polynomials are handled, and no integer constraint is allowed. As seen in example1.gms, "Variables", "Positive variables", "Equations" can not appear more than once. For more details, we refer to the examples included in SparsePOP and [2].

2.2 The SparsePOP format

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 column vector of poly.noTerms dimension.

The name objPoly is for the objective polynomial function $f_0(\boldsymbol{x})$ and ineqPolySys$\{j\}$ ($j = 1, 2, \ldots, m$) for the polynomials $f_j(\boldsymbol{x})$ ($j = 1, 2, \ldots, m$) of the constraints. The problem (2) is described using the polynomial class as follows.

```
function [objPoly,ineqPolySys,lbd,ubd] = example1;
%%%%%%%%%%%%%
% example1.m
%%%%%%%%%%%%%
%
% The SparsePOP format data for the example1:
%
% minimize -2*x1 +3*x2 -2*x3
% subject to
%       x1^2 + 3*x2^2 -2*x2*x3 +3*x3^2 -17*x1 +8*x2 -14*x3 >= -19,
%       x1 + 2*x2 + x3 <= 5,
%       5*x2 + 2*x3 = 7,
%       0 <= x1 <= 2, 0 <= x2 <= 1.
%
% To solve the problem by sparsePOP.m:
% >> param.relaxOrder = 3;
% >> sparsePOP('example1',param);
%
% This problem is also described in terms of the GAMS scalar format in the
% file example1.gms.  See Section 3 of the manual.
```

```
%
%'example1'
% 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.
% 0 <= x1, 0 <= x2, -infinity < x3:
        lbd = [0,0,-1.0e10];
% upper bounds for variables x1, x2 and x3
% x1 <= 2, x2 <= 1, x3 < infinity:
        ubd = [2,1,1.0e10];
return
% end of example1.m
```

We note that -1.0e10 in lbd and 1.0e10 in ubd mean $-\infty$ and ∞, respectively, indicating x_3 can take any value in the above example. The functions simplifyPolynomial.m, plusPoly-

nomials.m, and multiplyPolynomials.m in the directory subPrograms/Mfiles/ are useful in describing a POP in terms of the SparsePOP format. See Rosenbrock.m, and also other .m files in the directory example/POPformat/ for more general description of the SparsePOP format.

3 Sample executions of SparsePOP

Execution of SparsePOP is illustrated with examples. While SparsePOP can read a POP described in either the GAMS scalar format or the SparsePOP format shown in Section 2, there are three ways to solve a POP with SparsePOP.

3.1 The GAMS scalar format

Consider the POP (2) in Section 2, and assume that the POP is described in the GAMS scalar format in the file example1.gms. Execution of SparsePOP can be done by the following commands in Command window of MATLAB:

```
>> param.relaxOrder = 3;
>> sparsePOP('example1.gms',param);
```

Then, the following output appears in Command window:

```
SparsePOP 2.00 by H.Waki, S.Kim, M.Kojima, M.Muramatsu and H.Sugimoto
                                                        June 1, 2007

## Computational Results by sparsePOP.m with SeDuMi ##
## Printed by printSolution.m ##
# Problem File Name   = example1.gms
# parameters:
  relaxOrder          = 3   % = param.relaxOrder.
  sparseSW            = 1   % = param.sparseSW.
  SeDuMiOutFile       = 0   % = param.SeDuMiOutFile.
# SDP solved by SeDuMi:
  size of A           = [83,1085]   % = [SDPinfo.rowSize, SDPinfo.colSize];
  no of nonzeros in A = 2349  % = SDPinfo.nonzeroInA.
  no of LP variables  = 50  % = SDPinfo.noOfLPvariables.
  no of FR variables  = 35  % = SDPinfo.noOfFRvariables.
  no of SDP blocks    = 7  % <--- SDPinfo.SDPblock.
  max size SDP block  = 20  % <--- SDPinfo.SDPblock.
  ave.size SDP block  = 1.14e+01  % <--- SDPinfo.SDPblock.
# SeDuMi information:
  SeDuMi.pars.eps     = 1.00e-09  % = param.SeDuMiEpsilon.
  SeDuMiInfo.numerr   = 0  % = SeDuMiInfo.numerr.
  SeDuMiInfo.pinf     = 0  % = SeDuMiInfo.pinf.
  SeDuMiInfo.dinf     = 0  % = SeDuMiInfo.dinf.
# Approximate optimal value information:
  SDPobjValue         = -7.9550950e+00   % a lower bound for the unknown
```

```
                                        % optimal value obtained by
                                        % the sparse SDP relaxation.
  POP.objValue        = -7.9550951e+00  % an approximated optimal value.
  relative obj error  = +1.086e-09  % the relative error in the objective
                                    % value.
  POP.absError        = -1.295e-06  % the absolute error in the equality and
                                    % inequality constraints.
  POP.scaledError     = -2.642e-08  % the scaled error in equality and
                                    % inequality constraints.
# cpu time:
  cpuTime.conversion  =     0.05  % the cpu time for conversion from the POP
                                  % into an SDP relaxation problem.
  cpuTime.SeDuMi      =     0.27  % the cpu time to solve the SDP relaxation
                                  % problem by SeDuMi.
  cpuTime.total       =     0.32  % the total cpu time.
# Approximate optimal solution information:
  POP.xVect =
     1:+4.7754773e-01    2:+5.1726097e-08    3:+3.4999999e+00
```

Here param is a structure with optional fields for various parameters that affect the performance of sparsePOP.m. The fields of param are explained in Section 6, and the meaning of other outputs in Section 5.

The second input argument param in the function sparsePOP.m can be omitted as

```
>> sparsePOP('example1.gms');
```

In this case, the default parameters given in the function defaultParameter.m are used. One of the default parameters is param.relaxOrder $= \omega_{\max} = 1$, which will provide us with an inaccurate approximate solution for example1.gms.

3.2 The SparsePOP format

Since example1.m contains the description of the SparsePOP format of the POP (2), the POP (2) can be solved by SparsePOP as follows:

```
>> [objPoly,ineqPolySys,lbd,ubd] = example1;
>> param.relaxOrder = 3;
>> sparsePOP(objPoly,ineqPolySys,lbd,ubd,param);
```

This gives the same output as issuing commands

```
>> param.relaxOrder = 3;
>> sparsePOP('example1.gms',param)
```

as previously discussed in Section 4.1. When the last input argument param is omitted, the default values in defaultParameter.m are used for the parameters. If a given POP doesn't have lower and upper bounds on some variable x_i of the POP, assigning -1.0e+10 and 1.0e+10 to lbd(i) and ubd(i), respectively, is necessary. In particular, when a given

POP is unconstrained problem, it is necessary to set ineqPolySys = [] (the empty set), lbd(i) = -1.0e+10 and ubd(i)= 1.0e+10 for all i.

A simpler way to solve the POP (2) using the file example1.m in the SparsePOP format of the POP (2) is:

```
>> param.relaxOrder = 3;
>> sparsePOP('example1',param);
```

or

```
>> sparsePOP('example1');
```

Executing sparsePOP.m as the last command is especially convenient when we handle a POP with varying parameters. As an example, consider a minimization of the generalized Rosenbrock function

$$1 + \sum_{i=1}^{n} \left(100(x_i - x_{i-1}^2)^2 + (1 - x_i)^2\right).$$

This function has the minimum 0 at $\boldsymbol{x}^1 = (-1, 1, 1, \ldots, 1)^T \in \mathbb{R}^n$ and $\boldsymbol{x}^2 = (1, 1, 1, \ldots, 1)^T \in \mathbb{R}^n$. The description in the SparsePOP format of this problem is written in the file Rosenbrock.m whose function declaration line is

```
function [objPoly,ineqPolySys,lbd,ubd] = Rosenbrock(nDim,s);
```

Here nDim stands for the variable dimension n of the generalized Rosenbrock function. We specify $s = -1$ for the constraint $x_1 \leq 0$, $s = 0$ for no constraint, and $s = 1$ for the constraint $x_1 \geq 0$. The commands to solve the problem of $n = 60$ subject to $x_1 \leq 0$ are

```
>> param.relaxOrder = 2;
>> sparsePOP('Rosenbrock(60,-1)',param);
```

And, for the problem of $n = 120$ subject to $x_1 \geq 0$,

```
>> param.relaxOrder = 2;
>> sparsePOP('Rosenbrock(120,1)',param);
```

4 Solving POPs with multiple optimal solutions

Consider solving the problem of minimizing the generalized RosenBrock function with $n = 40$ over the entire space by issuing the commands

```
>> param.relaxOrder = 2;
>> sparsePOP('Rosenbrock(40,0)',param);
```

The output data is

```
  . . . . . .

# Approximate optimal value information:
  SDPobjValue         = +1.0000022e+00
  POP.objValue         = +1.0099885e+02
  relative obj error  = +9.901e-01
  POP.absError        = +0.000e+00
  POP.scaledError     = +0.000e+00
  . . . . . .
```

A big difference between SDPobjValue and POP.objValue indicates that no accurate approximate optimal solution has been found. This is because **SparsePOP can not handle efficiently a POP having multiple optimal solutions**. One way to resolve this difficulty is to add either $x_1 \leq 0$ or $x_1 \geq 0$ as an inequality constraint to select only one optimal solution. This is actually shown in the previous subsection. Such inequality constraints, however, can not be known before attaining an approximate optimal solution. This difficulty can be better handled by linear perturbation to the objective function of a given POP that possesses possibly multiple solutions. Let $\epsilon > 0$ be a small positive number such as 1.0e-4, and $\boldsymbol{p} \in \mathbb{R}^n$ be a column vector whose elements p_i $(i = 1, 2, \ldots, n)$ are chosen randomly from the unit interval $[0, 1]$. The objective function $f_0(\boldsymbol{x})$ of a given POP is then replaced by a perturbed objective function $f_0(\boldsymbol{x}) + \epsilon \boldsymbol{p}^T \boldsymbol{x}$ such that the resulting POP has a unique optimal solution. The optimal solution of the perturbed POP is expected to be an approximation for an optimal solution of the original POP. We refer to [11] for more details. In the minimization of the generalized Rosenbrock function, this perturbation technique is carried out as follows:

```
>> param.relaxOrder = 2;
>> param.perturbation = 1.0e-4;
>> sparsePOP('Rosenbrock(40,0)',param);
```

The result is:

```
  . . . . . .

# Approximate optimal value information:
  SDPobjValue         = +9.9991951e-01
  POP.objValue         = +1.0001360e+00
  relative obj error  = +2.164e-04
  POP.absError        = +0.000e+00
  POP.scaledError     = +0.000e+00
  . . . . . .
```

Once we obtain an approximate solution $\hat{x}$ of a POP by applying the perturbation technique, we can apply sparsePOP.m again to the original POP with updated lower and upper constraints such that

$$\begin{aligned} \mathrm{lbd}_i &= \max\{\mathrm{lbd}_i,\ \hat{x}_i - \delta_i\}\ (i = 1, 2, \ldots, n), \\ \mathrm{ubd}_i &= \min\{\mathrm{ubd}_i,\ \hat{x}_i + \delta_i\}\ (i = 1, 2, \ldots, n). \end{aligned}$$

for some small positive numbers δ_i $(i = 1, 2, \ldots, n)$. If the new lower and upper bounds separate a single optimal solution from all other optimal solutions of the original POP, this method is expected to work effectively.

5 Description of main and principal subfunctions

The main function and principal subfunctions are described in terms of input and output arguments in this section.

5.1 The MATLAB functions sparsePOP.m, SDPrelaxation.m, and SDPrelaxationMex.m

The main function sparsePOP.m, its principal subfunctions SDPrelaxation.m and SDPrelaxationMex.m shown in Figure 1 have the following function declarations:

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

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

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

respectively. These three functions have the same input and output arguments. Among the input arguments, param contains a set of parameters whose detailed description is included in Section 6. The other input arguments, if all of them are specified, describe a POP in the SparsePOP format, as presented in Section 4.2.

Although sparsePOP.m is defined with 5 input arguments, using 1 or 2 input arguments is also possible as mentioned in Section 3.

- >> sparsePOP('example1.gms') for solving a POP described in the GAMS scalar format with the default param.
- >> sparsePOP('example1.gms',param) for solving a POP described in the GAMS scalar format with the user-specified param.
- >> sparsePOP('example1') for solving a POP described in the SparsePOP format with the default param.
- >> sparsePOP('example1',param) for solving a POP described in the SparsePOP format with the user-specified param.

If the SDPrelaxation.m or the SDPrelaxationMex.m is to be utilized directly, either a set of the 5 input arguments

param, objPoly, ineqPolySys, lbd and ubd

or a set of the 3 input arguments

param, objPoly and ineqPolySys

needs to be specified. In the case that 3 input arguments are prescribed, the functions SDPrelaxation.m and SDPrelaxationMex.m assign the default values $\mathsf{lbd}(i) = -1.0\text{e}{+10}$ and $\mathsf{ubd}(i) = +1.0\text{e}{+10}$ $(i = 1, 2, \ldots, n)$, which implies that $-\infty < x_i < \infty$ $(i = 1, 2, \ldots, n)$.

For the output arguments, user-specified or default values for the parameters are stored in param. SDPobjValue contains a lower bound for the optimal objective value of the POP (1). For every feasible solution $\boldsymbol{x}$ of the POP (1),

$$\mathsf{SDPobjValue} \leq f_0(\boldsymbol{x}) \tag{3}$$

holds. The output argument POP has four components:

- POP.xVect: a candidate $\boldsymbol{x}^\omega$ of an optimal solution of the POP (1).
- POP.objValue: the objective function value $f_0(\boldsymbol{x}^\omega)$ at $\boldsymbol{x}^\omega = \mathsf{POP.xVect}$.
- POP.absError: an absolute feasibility error at $\boldsymbol{x}^\omega$.
- POP.scaledError: a scaled feasibility error $\boldsymbol{x}^\omega$.

(Recall that the relaxation order $\omega = \mathsf{param.relaxOrder}$ determines the quality of the SDP relaxation of the POP). Here the absolute feasibility error at $\boldsymbol{x}^\omega$ is given by

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

and the scaled feasibility error is given by

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

where $\sigma_i(\boldsymbol{x}^\omega)$ denotes the maximum of the absolute values of all monomials of $f_i(\boldsymbol{x})$ evaluated at $\boldsymbol{x}^\omega$ if the maximum is greater than 1, or $\sigma_i(\boldsymbol{x}^\omega) = 1$ otherwise $(i = 1, 2, \ldots, m)$. Note that both errors are always nonpositive. The relative error in the objective value at $\boldsymbol{x}^\omega$ in the output of sparsePOP.m, which has been illustrated in Section 4, is computed as

$$\mathsf{relative\ obj\ error} = \frac{\mathsf{POP.objValue} - \mathsf{SDPobjValue}}{\max\{1, |\mathsf{POP.objValue}|\}}.$$

If $\mathsf{POP.scaledError} \leq 0$ is close to 0, say $-1.0\text{e-}6 \leq \mathsf{POP.scaledError} \leq 0$, we may regard that $\boldsymbol{x}^\omega$ is feasible approximately. If, in addition, $\mathsf{relative\ obj\ error} \geq 0$ is close to 0, say $0 \leq \mathsf{relative\ obj\ error} \leq 1.0\text{e-}6$, $\boldsymbol{x}^\omega$ is an approximate optimal solution of the POP (1).

The output argument cpuTime shows various cpu times consumed by sparsePOP.m:

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

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

- SDPinfo.rowSizeA: the number of rows of the coefficient matrix $\boldsymbol{A}$ of the SDP.
- SDPinfo.colSizeA: the number of columns of the coefficient matrix $\boldsymbol{A}$.
- SDPinfo.nonzeroInA: the number of nonzeros of the coefficient matrix A.
- SDPinfo.noOfLPvariables: the number of LP variables of the SDP.
- SDPinfo.noOfFRvariables: the number of free variables of the SDP.
- SDPinfo.SDPblock: the row vector of the sizes of SDP blocks.

Finally, the output argument SeDuMiInfo contains SeDuMiInfo.numerr, SeDuMiInfo.pinf, and SeDuMi.dinf, which are equivalent to info.numerr, info.pinf, and info.dinf in SeDuMi output. See [8] for the details.

5.2 The MATLAB function readGMS.m

The MATLAB function readGMS.m has the following function declaration:

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

The first argument fileName, a string in MATLAB, is the name of the file where a problem is described in the GAMS scalar format. It must have the extension .gms such as 'example1.gms'. The second input argument symbolicMath is set to be 1 by default, assuming that the Symbolic Math Tool is available. It should be set to 0 if it is not available. The output of objPoly, ineqPolySys, lbd, and ubd is a POP data in the SparsePOP format, and can be passed to SDPrelaxation.m or SDPrelaxationMex.m.

5.3 The MATLAB function printSolution.m

The function printSolution.m for printing the results has the following function declaration.

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

The meaning of each input argument is as follows.

- fileId: fileId where output is printed. If fileId is 1, then the result is displayed on the screen (i.e., the standard output). If the result in a file is desired, the file should be open in writable mode before specifying it in fileId.
- printLevel: a larger value of printLevel gives more detailed description of the result. Default is 2.
- dataFileName: the name of the problem solved.

The rest of the input arguments, i.e., param, SDPobjValue, POP, cpuTime, SeDuMiInfo, and SDPinfo should be the output of SDPrelaxation.m and SDPrelaxationMex.m.

6 Parameters

In addition to objPoly, ineqPolySys, lbd and ubd for describing a POP in the SparsePOP format, the MATLAB function sparsePOP.m has param as an input argument. It is a structure consisting of many parameters that control the performance of the function. Table 1 shows the list of parameters defined in SparsePOP. The default values of all parameters are given in the MATLAB function defaultParameters.m. They can be modified if necessary.

Table 1: The fields of param, default values and possible values

field of param	default values	possible values
relaxOrder	$\omega_{\max}$	a positive integer not less than $\omega_{\max}$
sparseSW	1	0 and 1
multiCliquesFactor	1	0, 1 and 'objPoly.dimVar'
scalingSW	1	0 and 1
boundSW	1	0, 1 and 2
eqTolerance	0.0	0.0 and a positive real number > 1.0e-10
perturbation	0.0	0.0 and a positive real number > 1.0e-10
reduceMomentMatSW	1	0 and 1
complementaritySW	1	0 and 1
SeDuMiSW	1	0 and 1
SeDuMiOutFile	0	0, 1 and a file name
SeDuMiEpsilon	1.0e-9	a positive real value
sdpaDataFile	' '	' ' and a file name with the extension .dat-s
detailedInfFile	' '	' ' and a file name
printFileName	1	1 and a file name
printLevel	[2,2]	0,1 or 2 for both elements
symbolicMath	1	0 and 1
mex	1	0 and 1

These parameters can be divided into five categories:

1. Parameters for controlling the basic relaxation scheme.
2. Switches for techniques to reduce numerical difficulties.
3. Parameters for SDP solvers.
4. Parameters for printing.
5. Parameters for Symbolic Math Toolbox and C++ subroutines.

6.1 Parameters to choose the relaxation method

The relaxation order should be specified as

$$\text{param.relaxOrder} = \omega \geq \omega_{\max} = \max\{\omega_k : k = 0, 1, \ldots, m\}$$

to execute the sparse or dense SDP relaxation for a POP, where

$$\omega_k = \lceil \deg(f_k(\boldsymbol{x}))/2 \rceil \ (k = 0, 1, 2, \ldots, m).$$

The default is param.relaxOrder $= \omega_{\max}$. If the accuracy of the obtained approximate optimal solution is not satisfactory, increasing the relaxation order param.relaxOrder $= \omega \geq \omega_{\max}$ is one way to obtain more accurate optimal solution. We should mention, however, that increasing the relaxation order may take more cpu time and numerical difficulty might occur while solving the SDP relaxation problem.

The type of SDP relaxation can be chosen by setting param.sparseSW $= 0$ for the dense relaxation based on [6], or $= 1$ for the sparse relaxation based on [11]. The sparsity of the sparse SDP relaxation problem 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=objPoly.dimVar, 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 $\rho_{\max}$. If the accuracy of the obtained approximate optimal solution is not satisfactory, sparsePOP.m can be executed again with the choice of either param.multiCliquesFactor=objPoly.dimVar or param.sparseSW $= 0$ before increasing the relaxation order param.relaxOrder $= \omega$.

6.2 Switches for techniques to handle numerical difficulties

Because the POP (1) is basically a hard optimization problem, numerical difficulties are often unavoidable while solving its SDP relaxation, and/or an inaccurate approximate solution might be obtained. The switches described in this subsection are intended to prevent numerical difficulties from occuring, and improve the accuracy of an obtained solution.

With param.scalingSW$= 1$, the objective polynomial, constraint polynomials, lower and upper bounds are scaled such that the maximum of {|lower bound of x_j|, |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 for which 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 $= 1$. If param.boundSW$= 2$, some redundant bounds of variables $y_{\boldsymbol{\alpha}}$ are removed from the added bounds for all $y_{\boldsymbol{\alpha}}$. Otherwise, no bounds are added to $y_{\boldsymbol{\alpha}}$. The default is param.boundSW$= 2$. 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, our recommendation is 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 sparsePOP.m by taking param.scalingSW = 1 and param.boundSW = 1 or 2.

The parameter param.eqTolerance is used to convert every equality constraint into two inequality constraints; if 1.0e-10 < param.eqTolerance, then each equality constraint is replaced by $f(x) = 0$ by $f(x) \geq -$param.eqTolerance and $-f(x) \geq -$param.eqTolerance. When SeDuMi displays numerical difficulty while 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.*, no conversion of the equality constraints is specified.

Perturbing the objective polynomial to compute an optimal solution of a POP with multiple optimal solutions is described in Section 5. See also Section 5.1 of [11]. The parameter param.perturbation is used for this purpose. If 1.0e-10 < param.perturbation, then the objective polynomial $f_0(x)$ is modified to $f_0(\boldsymbol{x}) + \boldsymbol{p}^T\boldsymbol{x}$, where $0 \leq p_i \leq$ param.perturbation. Otherwise, no perturbation is performed. The default value for param.perturbation is 0.0, *i.e.*, no perturbation of the objective polynomial is desired.

The parameter param.reduceMomentMatSW is intended for SDP relaxations too large to solve. If param.reduceMomentMatSW = 1, then sparsePOP.m eliminates redundant elements of $\mathcal{A}_\omega^{C_\ell}$ ($\ell = 1, 2, \ldots, p$) in the SDP relaxation problem using the method proposed in the paper [5]. See also [11].

When the complementarity condition exists in the constraints of a POP, 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 SDP relaxation problem. 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 approximate optimal solution of the problem obtained by SeDuMi, and SDPA sparse format data of the problem.

For infomation on the problem and data on the obtained optimal solution, SeDuMi should be called from the function SDPrelaxation.m or SDPrelaxationMex.m by setting param.SeDuMiSW= 1. Users can increase or decrease the desired accuracy of the optimal value and optimal solutions of an SDP relaxation problem by providing a smaller or larger value for param.SeDuMiEpsilon that corresponds to the parameter pars.eps in SeDuMi. The parameter param.SeDuMiOutFile is used in connection with the parameter param.SeDuMiSW= 1. The default value of param.SeDuMiOutFile= 1 is used to display the output from SeDuMi on the screen. If the name of a file such as param.SeDuMiOutFile = 'SeDuMi.out' is assigned, the output from SeDuMi is written in the file. Information from SeDuMi is not displayed if param.SeDuMiOutFile = 0. The value 0 for param.SeDuMiSW is for just printing information on the problem without solving the SDP relaxation problem. The default value for param.SeDuMiSW is 1.

In SparsePOP, SeDuMi [8] is used for solving SDPs because SeDuMi seems to have better

numerical stability than other SDP solvers. The use of an iterative method (a variant of CG method) for solving ill-conditional linear systems in SeDuMi leads to more accurate optimal solutions than other SDP solvers. Also, SparsePOP can provide SDPA sparse format data for (1) and a file that contains necessary information to extract an approximated solution of (1).

SDPA sparse format data of the SDP relaxation problem can be also obtained by assigning the name of a file for SDPA sparse format data to the parameter param.sdpaDataFile, for example, param.sdpaDataFile = 'test.dat-s'. With the SDPA sparse format input file 'test.dat-s', the SDP relaxation problem can be solved later by using some software packages such as SDPA [10] and SDPT3 [9]. The default is param.sdpaDataFile = '', *i.e.*, no SDPA sparse format data is created.

If the name of a file for SDPA sparse format data is provided, SparsePOP also generates the file containing necessary information for extracting an approximated solution of (1) from the SDP relaxation problem. The file has the extension "info" and the structure of the file is as follows: Here $k_1, \ldots, k_n$ are integers, and $a_1, \ldots, a_n$ and $b_1, \ldots, b_n$ are real numbers. After

$$\begin{array}{lll} k_1 & a_1 & b_1 \\ k_2 & a_2 & b_2 \\ \vdots & & \\ k_j & a_j & b_j \\ \vdots & & \\ k_n & a_n & b_n \end{array}$$

solving the SDP relaxation problem in SDPA sparse format by an SDP solver, if $k_j = -1$ for some j, set $x_j = b_j$. Otherwise, set $x_j = a_j y_{k_j} + b_j$, where y_{k_j} is the k_jth element of the optimal solution of the primal of SDP relaxation problem in SDPA sparse format. Then, $\boldsymbol{x}$ is an approximated solution for (1) computed by SparsePOP.

6.4 Parameters for printing numerical results

Whether we have param.SeDuMiSW= 1 or 0, we can store detailed information of the POP and its SDP relaxation in a file specified using param.detailedInfFile; for example, param.detailedInfFile = 'details.out'. The default is param.detailedInfFile = '', *i.e.*, no detailed information is printed.

param.printFileName is the parameter for displaying the computational results, such as param, SDPinfo, and POP. That is, param.printFileName= 1 is for displaying the results on screen, and param.printFileName= 0 prevents them from displaying. In addition, the name of a file to param.printFileName can be assigned to print the results in the file, such as param.printFileName='result.out'.

The default value 2 for param.printLevel(1) is used to display the computational result with an approximate optimal solution of the POP on screen. Setting param.printLevel(1)= 1 stops displaying an approximate optimal solution of the POP on the screen. The value 0 for param.printLevel(1) displays no computational result on the screen. The default value 2 for param.printLevel(2) is used to write the computational result with an approximate optimal solution of the POP in a file whose name is defined by param.printFileName. Setting

param.printLevel(2)= 1 prevents printing an approximate optimal solution on the file. If param.printLevel(2)= 0, no information is written in the file.

6.5 Parameters to use Symbolic Math Toolbox and C++ subroutines

The parameter param.symbolicMath indicates whether Symbolic Math Toolbox can be utilized for reading a POP in the GAMS scalar format, or not. Setting param.symbolicMath= 1 means that functions of Symbolic Math Toolbox can be used. See also the last paragraph of Section 3.1. C++ subroutines can be used by setting the parameter param.mex= 1.

7 Concluding Remarks

We have described the structure and usage of the software package SparsePOP. Solving POPs has been known difficult mainly because the size of SDP relaxation of POPs becomes increasingly large as the degree and the number of variables of POPs grow. In addition, numerical difficulties occur for various reasons. SparsePOP is, by far, one of the most successful software packages to address these issues among currently available softwares. Main advantage comes from constructing SDP relaxations of reduced size by utilizing the sparsity of POPs. In particular, unconstrained problems with 1000 variables has been solved using the sparsity. For sparse constrained problems, the number of variables of solvable problems by SparsePOP is 10-30. See the paper [11] for extensive numerical results.

References

[1] J. R. S. Blair and B. Peyton, "An introduction to chordal graphs and clique trees", in *Graph Theory and Sparse Matrix Computation*, A. George, J. R. Gilbert and J. W. H. Liu, eds., Springer-Verlag, New York, 1993, pp. 1 – 29.

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

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

[4] S. Kim, M. Kojima and H. Waki, "Generalized Lagrangian duals and sums of square relaxation of sparse polynomial optimization problems", *SIAM J. Optimization* **15** (2005) 697–719.

[5] M. Kojima, S. Kim and H. Waki, "Sparsity in sums of squares of polynomials", *Mathematical Programming* **103** (2005) 45-62.

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

[7] J. B. Lasserre: Convergent SDP-relaxations in polynomial optimization with sparsity, to appear in *SIAM Journal on Optimization*.

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

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

[10] 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.

[11] H. Waki, S. Kim, M. Kojima, and M. Muramatsu, "Sums of Squares and Semidefinite Programming Relaxations for Polynomial Optimization Problems with Structured Sparsity", *SIAM J. Optimization* **17** (2006) 218–242.