

Research Reports on Mathematical and Computing Sciences

User Manual for **SparsePOP**: a **S**parse
Semidefinite Programming Relaxation of
Polynomial **O**ptimization **P**roblems

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

March 2005, B-414, Revised August 2009

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**olynomial **O**ptimization **P**roblems

Hayato Waki[#], Sunyoung Kim[†], Masakazu Kojima[‡], Masakazu Muramatsu[#],
Hiroshi Sugimoto[‡] and Makoto Yamashita[‡]
March 2005, Revised August 2009

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

[#] Dept. of Computer Science, The University of Electro-Communications

[†] Dept. of Mathematics, Ewha Women’s University

[‡] Dept. of Mathematical and Computing Sciences, Tokyo Institute of Technology

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 [14], proposed to improve the efficiency of Lasserre's hierarchy of LMI relaxations of increasing dimensions [8]. SparsePOP exploits the sparsity of POPs so that it can handle POPs of larger dimensions. See also [6, 7].

We describe a general POP 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 $f_k(\mathbf{x})$ in $\mathbf{x} = (x_1, x_2, \dots, x_n) \in \mathbb{R}^n$ is expressed as

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

($k = 0, 1, 2, \dots, m$), where $\mathbf{x}^{\boldsymbol{\alpha}} = x_1^{\alpha_1} x_2^{\alpha_2} \dots x_n^{\alpha_n}$ for every $\mathbf{x} = (x_1, x_2, \dots, x_n) \in \mathbb{R}^n$ and every $\boldsymbol{\alpha} \in \mathbb{Z}_+^n$. Then, a POP is written in the form:

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

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

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 [12] or SDPA [3] 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 [4] 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 or SDPA. Once the POP (1) is solved, the solution information is written using the Matlab function printSolution.m. We refer to the paper [14] for numerical results of SparsePOP.

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

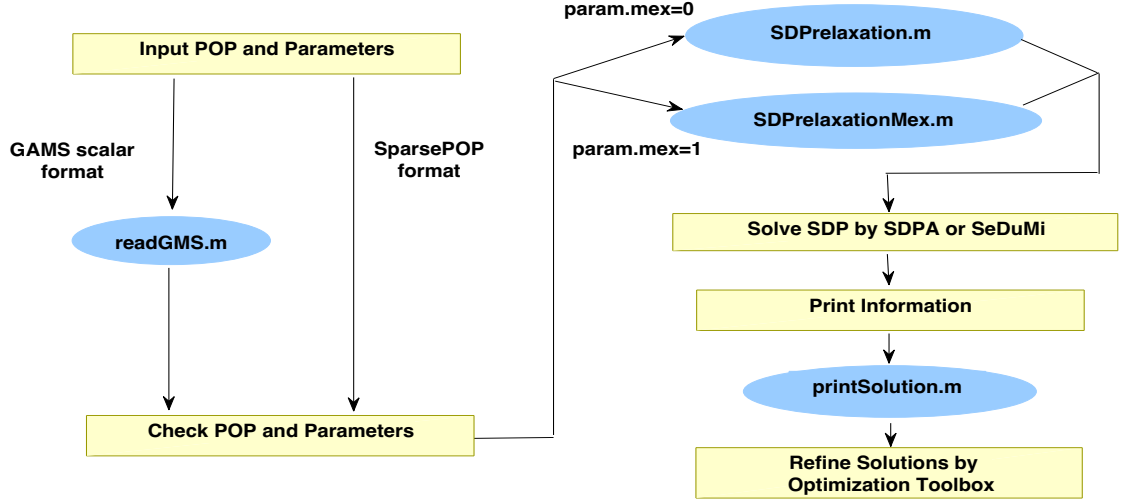


Figure 1: The structure of the main function sparsePOP.m

This manual is organized as follows: Section 2 includes the description of two kinds of format to express polynomials and POPs. An example for executing SparsePOP with SeDuMi as an SDP solver 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. Section 6 explains how we can call SDPA from SparsePOP as an SDP solver instead of SeDuMi. The parameters to the subfunctions SDPrelaxation.m and SDPrelaxationMex.m are explained in Section 7, and some numerical results are reported in Section 8.

2 Representation of polynomial optimization problems

The objective and constraint polynomials of a POP can be described in two different ways, namely, the GAMS scalar format and the SparsePOP format, 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. Alternatively, 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 :

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

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  $0.5 \leq x3 \leq 2$ , 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 POPs in the GAMS scalar format can be found in the directory
example/GMSformat/

These are from [5]; we have added and/or modified lower and upper bounds for 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 passed to the function `readGMS.m` where symbolic expansion take 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, we need to follow a set of rules: 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 [4].

2.2 The SparsePOP format

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

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

The name `objPoly` is for the objective polynomial function $f_0(\mathbf{x})$ and `ineqPolySys{j}` ($j = 1, 2, \dots, m$) for the polynomials $f_j(\mathbf{x})$ ($j = 1, 2, \dots, m$) 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`, `plusPolynomials.m`, and `multiplyPolynomials.m` in the directory `subPrograms/Mfiles/` are useful when 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 using SeDuMi

Executing SparsePOP is illustrated with examples. After preparing a POP in either the GAMS scalar format or the SparsePOP format as in Section 2, SparsePOP can be invoked in one of the following three ways, described in Section 3.1 to Section 3.3, to solve the POP.

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`. SparsePOP can be executed by the following commands in the command window of Matlab:

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

Then, the following output appears in the command window:

```
SparsePOP 2.20
by H.Waki, S.Kim, M.Kojima, M.Muramatsu,
   H.Sugimoto and M. Yamashita, July 2009

- SeDuMi Start -
- SeDuMi End -

## 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.
# 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
# SeDuMi information:
SeDuMi.pars.eps        = 1.00e-099 % = param.SDPsolverEpsilon.
SDPsolverInfo.numerr   = 0      % = SDPsolverInfo.numerr.
SDPsolverInfo.pinf     = 0      % = SDPsolverInfo.pinf.
SDPsolverInfo.dinf     = 0      % = SDPsolverInfo.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.
# elapsed time:
elapsedTime.readData   =      0.02 % the elapsed time for reading the POP.
elapsedTime.conversion =      0.05
                                % the elapsed time for conversion from the
                                % POP into an SDP relaxation problem.
elapsedTime.SeDuMi     =      0.26
                                % the cpu time to solve the SDP relaxation
                                % problem by SeDuMi.
elapsedTime.total      =      0.33 % the total cpu time.
# Approximate optimal solution information:
POP.xVect =
      1:+4.7754773e-01      2:+5.1730095e-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 7, and the meaning of the 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 = $\omega_{\max} = 1$` . This provides us with an approximate solution, which turns out to be inaccurate, for `example1.gms`.

3.2 The SparsePOP format

Since `example1.m` contains the description of the POP (2) in the SparsePOP format, 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 the 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 $-1.0\text{e}+10$ and $1.0\text{e}+10$ to $\text{lbd}(i)$ and $\text{ubd}(i)$, respectively, is necessary. In particular, when a given POP is unconstrained problem, it is necessary to set $\text{ineqPolySys} = []$ (the empty set), $\text{lbd}(i) = -1.0\text{e}+10$ and $\text{ubd}(i) = 1.0\text{e}+10$ 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

$$1 + \sum_{i=1}^{n-1} (100(x_{i+1} - x_i^2)^2 + (1 - x_{i+1})^2). \quad (3)$$

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

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

Here `nDim` stands for the dimension n of the variables 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);
```

3.3 Description of polynomial least square problems in the SparsePOP format

The minimization of the Rosenbrock function (3) can be viewed as a polynomial least square problem

$$\text{minimize} \quad \sum_{i=1}^{2(n-1)} f_i(\mathbf{x})^2,$$

where

$$\begin{aligned} f_i(\mathbf{x}) &= 10(x_{i+1} - x_i)^2 \quad (i = 1, 2, \dots, n-1), \\ f_j(\mathbf{x}) &= 1 - x_{j-n+2} \quad (j = n, n+1, \dots, 2(n-1)). \end{aligned}$$

We note that the constant term 1 in the Rosenbrock function (3) is omitted here. To apply SparsePOP in the previous section, the Matlab function Rosenbrock.m expands the sum of $2(n-1)$ squares of polynomials $f_i(\mathbf{x})^2$ ($i = 1, 2, \dots, 2(n-1)$) to a single polynomial in the SparsePOP format. This kind of expansion can require tedious work if the elements of polynomials are not simple. To avoid such expansion, SparsePOP allows a cell of objective function `objPolyLS` consisting of `objPolyLS{i}` ($i = 1, 2, \dots, 2(n-1)$), where each `objPolyLS{i}` corresponds to a description of the element polynomial $f_i(\mathbf{x})$ in the SparsePOP format. For example, `objPolyLS{i}` ($i = 1, 2, \dots, n-1$) are:

```
objIdx = 0;
for i=2:nDim
    objPolyLS{objIdx}.typeCone = 1;
    objPolyLS{objIdx}.dimVar   = nDim;
    objPolyLS{objIdx}.degree   = 2;
    objPolyLS{objIdx}.noTerms  = 2;
    objPolyLS{objIdx}.supports = ...
        sparse(objPolyLS{objIdx}.noTerms, objPolyLS{objIdx}.dimVar);
    objPolyLS{objIdx}.supports(1,i) = 1;
    objPolyLS{objIdx}.supports(2,i-1) = 2;
    objPolyLS{objIdx}.coef        = [10; -10];
end
```

When SparsePOP receives a cell of objective function `objPolyLS` with $\text{size}(\text{objPolyLS}, 2) \geq 2$, it expands the sum of squares of elements of polynomials described in `objPolyLS{i}` ($i = 1, 2, \dots, 2(n-1)$) to a single polynomial of objective function before constructing a sparse SDP relaxation. See the Matlab program RosenbrockLS.m for a complete description of the least square problem (3). The commands to solve RosenbrockLS(60,-1) are similar to solving Rosenbrock(60,-1) stated in the previous section;

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

3.4 Refining approximate optimal solutions using Optimization Toolbox

When Optimization Toolbox is available in Matlab, it can help refine an approximation to an optimal solution of a POP obtained from its sparse SDP relaxation by setting the parameter. That is, set `param.POPsolver` to 'active-set', 'trust-region-reflective' or 'interior-point' to apply the Matlab function `fmincom.m` from the toolbox. For example, if we issue the commands

```
>> param.relaxOrder = 3;
>> param.POPsolver = 'active-set';
>> sparsePOP('example1.gms',param);
```

then we have

SparsePOP 2.20

by H.Waki, S.Kim, M.Kojima, M.Muramatsu,
H.Sugimoto and M.Yamashita, July 2009

```
. . . . .

## Computational Results by sparsePOP.m with SeDuMi ##
## Printed by printSolution.m ##

. . . . .

# Approximate optimal value information:
SDPobjValue      = -7.9550950e+00
POP.objValue     = -7.9550951e+00
relative obj error = +1.086e-09
POP.absError     = -1.295e-06
POP.scaledError  = -2.642e-08

. . . . .

# Approximate optimal solution information:
POP.xVect =
    1:+4.7754773e-01    2:+5.1730095e-08    3:+3.4999999e+00

## fmincon or fminunc in Optimization Toolbox

Optimization terminated: first-order optimality measure less
than options.TolFun and maximum constraint violation is less
than options.TolCon.

## Computational Results by medium-scale: SQP, Quasi-Newton, line-search method
with the initial solution obtained by the sparse SDP relaxation ##
```

```

exitflag          =      1
iterations        =      2
elapsed time      =     0.76
# Approximate optimal value information:
POP.objValueL     = -7.9550954e+00
relative obj error = +6.771e-07
POP.absErrorL     = +0.000e+00
POP.scaledErrorL  = +0.000e+00
# Approximate optimal solution information:
POP.xVectL =
    1:+4.7754769e-01    2:+0.0000000e+00    3:+3.5000000e+00

```

If a given POP is a polynomial least square problem with a cell of objective function `objPolyLS` as in the `RosenbrockLS.m`, `param.POPsolver = 'lsqnonlin'` can be set to apply the Matlab function specialized for nonlinear least square problems with bounds. See the manual of Optimization Toolbox for more details of the Matlab functions `fmincon` and `lsqnonlin`.

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 significant difference between `SDPobjValue` and `POP.objValue` indicates that no accurate approximation to an optimal solution has been found. This is because **SparsePOP cannot efficiently handle 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 $1.0\text{e-}4$, and $\mathbf{p} \in \mathbb{R}^n$ be a column vector whose elements p_i ($i = 1, 2, \dots, n$) are chosen randomly from the unit interval $[0, 1]$. The objective function $f_0(\mathbf{x})$ of a given POP is then replaced by the perturbed objective function $f_0(\mathbf{x}) + \epsilon \mathbf{p}^T \mathbf{x}$ such that the resulting POP has a unique optimal solution. The optimal solution of the perturbed POP is expected to be an approximation to an optimal solution of the original POP. We refer to [14] 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 $\hat{\mathbf{x}}$ 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

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

for some small positive numbers δ_i ($i = 1, 2, \dots, n$). 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,elapsedTime,SDPsolverInfo,SDPinfo] = ...
    sparsePOP(objPoly,ineqPolySys,lbd,ubd,param);
```

```
function [param,SDPobjValue,POP,elapsedTime,SDPsolverInfo,SDPinfo] = ...
    SDPrelaxation(param,objPoly,ineqPolySys,lbd,ubd);
```

```
function [param,SDPobjValue,POP,elapsedTime,SDPsolverInfo,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 7. 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 $\text{lbd}(i) = -1.0\text{e}+10$ and $\text{ubd}(i) = +1.0\text{e}+10$ ($i = 1, 2, \dots, n$), which implies that $-\infty < x_i < \infty$ ($i = 1, 2, \dots, n$).

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 \mathbf{x} of the POP (1),

$$\text{SDPobjValue} \leq f_0(\mathbf{x}) \quad (4)$$

holds. The output argument **POP** has four components:

- **POP.xVect**: a candidate \mathbf{x}^ω for an optimal solution of the POP (1).

- **POP.objValue**: the objective function value $f_0(\mathbf{x}^\omega)$ at $\mathbf{x}^\omega = \text{POP.xVect}$.
- **POP.absError**: an absolute feasibility error at \mathbf{x}^ω .
- **POP.scaledError**: a scaled feasibility error \mathbf{x}^ω .

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

$$\min \{ \min \{ f_i(\mathbf{x}^\omega), 0 \} \ (i = 1, 2, \dots, \ell), \ -|f_j(\mathbf{x}^\omega)| \ (j = \ell + 1, \dots, m) \}, \quad (5)$$

and the scaled feasibility error is given by

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

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

$$\text{relative obj error} = \frac{\text{POP.objValue} - \text{SDPobjValue}}{\max\{1, |\text{POP.objValue}|\}}. \quad (6)$$

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

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

- **SDPinfo.rowSizeA**: the number of rows of the coefficient matrix \mathbf{A} of the SDP.
- **SDPinfo.colSizeA**: the number of columns of the coefficient matrix \mathbf{A} .
- **SDPinfo.nonzeroInA**: the number of nonzeros of the coefficient matrix \mathbf{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.

The output argument **SDPsolverInfo** contains **SDPsolverInfo.numerr**, **SDPsolverInfo.pinf**, and **SDPsolverInfo.dinf**, which are equivalent to `info.numerr`, `info.pinf`, and `info.dinf` in SeDuMi output. See [12] for the details.

When the parameter **param.POPsolver** is set to ‘active-set’ (‘trust-region-reflective’, ‘interior-point’) or ‘lsqnonlin’ to call the Matlab function `fmincon` or `lsqnonlin` from Optimization Toolbox, we also have the following output.

- **POP.xVectL**: a refined optimal solution of the POP (1).
- **POP.objValueL**: the objective function value at **POP.xVectL**.

- `POP.absErrorL`: an absolute feasibility error at `POP.xVectL`.
- `POP.scaledErrorL`: a scaled feasibility error at `POP.xVectL`.

The output argument `elapsedTime` shows various CPU time consumed by `sparsePOP.m`:

- `elapsedTime.conversion`: the CPU time consumed to convert the POP into its SDP relaxation.
- `elapsedTime.SDPsolver`: the CPU time consumed to solve the SDP.
- `elapsedTime.Total`: the CPU time for the entire process.

5.2 The Matlab function `readGMS.m`

The function declaration of the Matlab function `readGMS.m` is:

```
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 declaration of the function `printSolution.m` for printing the results is:

```
function printSolution(fileId,printLevel,dataFileName,param,SDPobjValue,...
    POP,elapsedTime,SDPsolverInfo,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 to be solved.

The rest of the input arguments, *i.e.*, `param`, `SDPobjValue`, `POP`, `elapsedTime`, `SDPsolverInfo`, and `SDPinfo`, should be the output of `SDPrelaxation.m` and `SDPrelaxationMex.m`.

6 Execution of SparsePOP using SDPA

Instead of using SeDuMi for solving SDP relaxation problems, SDPA [3] can be specified as a SDP solver in one of the following two methods. First,

```
>> param.SDPsolver = 'sdpa';
```

Then, execute sparsePOP with `param`. For example, issue a command

```
>> sparsePOP('BroydenTri(20)',param);
```

Alternatively, modify the Matlab program `defaultParameter.m` such that SDPA becomes a default SDP solver. Namely, replace the line

```
param.SDPsolver = 'sedumi';
```

by

```
param.SDPsolver = 'sdpa';
```

7 Parameters

The Matlab function `sparsePOP.m` includes `param` as an input argument, in addition to `objPoly`, `ineqPolySys`, and `lbd/ubd` for describing a POP in the SparsePOP format. 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.

These parameters can be divided into six 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 POP solvers.
5. Parameters for printing.
6. Parameters for Symbolic Math Toolbox and C++ subroutines.

7.1 Parameters to choose the relaxation method

The relaxation order should be specified as

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

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

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

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

field of param	default values	possible values
relaxOrder	ω_{\max}	a positive integer not less than ω_{\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 $> 1.0\text{e-}10$
perturbation	0.0	0.0 and a positive real number $> 1.0\text{e-}10$
reduceMomentMatSW	1	0 and 1
complementaritySW	1	0 and 1
SDPsolver	'sedumi'	'sedumi' and 'sdpa'
SDPsolverSW	1	0 and 1
SDPsolverOutFile	0	0, 1 and a file name
SDPsolverEpsilon	1.0e-7	a positive real value
sdpaDataFile	' '	' ' and a file name with the extension .dat-s
POPsolver	[]	'active-set', 'trust-region-reflective', 'interior-point' and 'lsqnonlin'
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

The default is `param.relaxOrder` = ω_{\max} . If the accuracy of the obtained approximation to an optimal solution is not satisfactory, the relaxation order `param.relaxOrder` = $\omega \geq \omega_{\max}$ can be increased to obtain more accurate optimal solution. We should mention, however, that increasing the relaxation order may take longer 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 [8], or = 1 for the sparse relaxation based on [14]. 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 ($\ell = 1, 2, \dots, p$) of a chordal extension $G(N, E')$ of the csp (correlative sparsity pattern) graph induced from the POP (1) for \tilde{C}_k ($k = 1, 2, \dots, m$). Let ρ_{\max} denote the maximum over $\#C_\ell$ ($\ell = 1, 2, \dots, p$), where $\#C_\ell$ 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 \tilde{C}_k . Add another clique from C_ℓ ($\ell \in J_k$) to \tilde{C}_k if $\#(\tilde{C}_k \cup C_\ell)$ does not exceed `param.multiCliquesFactor` $\times \rho_{\max}$. Repeat this procedure to obtain the union \tilde{C}_k of some cliques from C_ℓ ($\ell \in J_k$). If `param.multiCliquesFactor` = 0, then \tilde{C}_k consists of a single clique C_ℓ for some $\ell \in J_k$. If `param.multiCliquesFactor`=`objPoly.dimVar`, then \tilde{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 \tilde{C}_k is bounded by ρ_{\max} . If the accuracy of the obtained approximation to an 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` = ω .

7.2 Switches for techniques to handle numerical difficulties

Because the POP (1) is basically a hard optimization problem, numerical difficulties frequently arise 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 occurring, 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 $\{|\text{lower bound of } x_j|, |\text{upper bound of } x_j|\}$ = 1 ($j \in J$) and that the maximum of the 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_α ($\alpha \in \tilde{\mathcal{F}}$) if `param.boundSW` = 1. If `param.boundSW`=2, some redundant bounds of variables y_α are removed from the added bounds for all y_α . Otherwise, no bounds are added to y_α . The default is `param.boundSW`=2. In particular, when every variable x_j is scaled such that $\text{lbd}(j) = 0$ and $\text{ubd}(j) = 1$ ($j = 1, 2, \dots, n$), the bounds $0 \leq y_\alpha \leq 1$ ($\alpha \in \tilde{\mathcal{F}}$) 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_α ($\alpha \in \tilde{\mathcal{F}}$) 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 $1.0\text{e-}10 < \text{param.eqTolerance}$, then each equality constraint is replaced by $f(x) = 0$ by $f(x) \geq -\text{param.eqTolerance}$ and $-f(x) \geq -\text{param.eqTolerance}$. When the SDP solver (SeDuMi or SDPA) displays numerical difficulty while solving the SDP relaxation of a POP with equality constraints, this technique with $1.0\text{e-}3 \leq \text{param.eqTolerance} \leq 1.0\text{e-}7$ often provides a more stable SDP relaxation problem that can be solved by the SDP solver. 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 [14]. The parameter `param.perturbation` is used for this purpose. If $1.0\text{e-}10 < \text{param.perturbation}$, then the objective polynomial $f_0(x)$ is modified to $f_0(\mathbf{x}) + \mathbf{p}^T \mathbf{x}$, where $0 \leq p_i \leq \text{param.perturbation}$. 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 be solved. If `param.reduceMomentMatSW = 1`, then `sparsePOP.m` eliminates redundant elements of $\mathcal{A}_\omega^{C_\ell}$ ($\ell = 1, 2, \dots, p$) in the SDP relaxation problem using the method proposed in the paper [7]. See also [14].

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 \mathbf{x}^α 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`.

7.3 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, and data on an approximate optimal solution of the problem obtained by the SDP solver (SeDuMi or SDPA), and SDPA sparse format data of the problem [3].

To obtain information on the problem and data on the obtained optimal solution, an SDP solver either SeDuMi or SDPA is called from the function `SDPrelaxation.m` or `SDPrelaxationMex.m` by setting `param.SDPsolverSW = 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.SDPsolverEpsilon` that corresponds to the parameter `pars.eps` in SeDuMi, and `epsilonStar` and `epsilonDash` in SDPA. The parameter `param.SDPsolverOutFile` is used in connection with the parameter `param.SDPsolverSW = 1`. The default value of `param.SDPsolverOutFile = 0` is used not to display the output from the SDP solver on the screen. If the name of a file such as `param.SDPsolverOutFile = 'SDP-solver.out'` is assigned, the output from the SDP solver is written in the file. Information from the SDP solver is displayed on the screen if `param.SDPsolverOutFile = 1`. The value `0` for `param.SDPsolverSW` is for just printing information on the problem without solving the SDP relaxation problem. The default value for `param.SDPsolverSW` is `1`.

The reason why SeDuMi is chosen for default is that SeDuMi has been shown to have better numerical stability than other SDP solvers. The user can change the SDP solver to SDPA by specifying `param.SDPsolver= 'sdpa'`. In general, SDPA is faster. See numerical results in Section 8.

SparsePOP can also 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 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 input file 'test.dat-s' in the SDPA sparse format, the SDP relaxation problem can be solved later by software packages such as SDPA and SDPT3 [13]. The default is `param.sdpaDataFile = ''`, *i.e.*, no data in the SDPA sparse format is created.

If the name of a file for data in the SDPA sparse format 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, \dots, k_n are integers, and a_1, \dots, a_n and b_1, \dots, b_n are real

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

numbers.

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

7.4 Parameters for POP solvers

The parameter `param.POPsolver` is valid only when Optimization Toolbox is available in Matlab. To apply `fmincon` with 'active-set' algorithm, 'trust-region-reflective' algorithm or 'interior-point' algorithm to refine the approximate optimal solution of a POP obtained by its sparse SDP relaxation, set `param.POPsolver = 'active-set'`, 'trust-region-reflective' or 'interior-point', respectively. If a given POP is a polynomial least square problem having a cell of objective function `objPolyLS` and `inEqPolySys = []` as in the Matlab program `RosenbrockLS.m`, the Matlab function `lsqnonlin` can be applied for the refinement by setting `param.POPsolver = 'lsqnonlin'`. See the manual of Optimization Toolbox for more details of the Matlab functions `fmincon` and `lsqnonlin`.

7.5 Parameters for printing numerical results

In either case of `param.SDPsolverSW= 1` or `0`, detailed information on the POP and its SDP relaxation can be stored 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.

The parameter `param.printFileName` is for displaying the computational results, such as `param`, `SDPinfo`, and `POP`. That is, `param.printFileName= 1` is for displaying the results on the 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 the 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.

7.6 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. Subroutines written in C++ can be used by setting the parameter `param.mex= 1`.

8 Numerical Results

We report numerical results to show performance of SparsePOP. We also compare SeDuMi and SDPA as an SDP solver used in SparsePOP. Numerical experiments were performed on 3.00GHz Quad-Core Intel Xeon X5365 with 48GB memory. In Tables 2 and Tables 3, “eTime” denotes the elapsed time for the SDP solver SeDuMi or SDPA to solve SDP relaxation problems (the elapsed time for the MATLAB Optimization Toolbox `fmincon` or `fminunc` to refine approximate optimal solutions of POPs obtained by the SDP solver), “rObjErr” the relative objective function error given by (6), and “absErr” the absolute feasibility error given by (5).

All the problems in Table 2 are originally from the literature [5], and stored with the prefix “B” and an extension “.gms” in the directory `example/GMSformat`; for example, the problem `ex2.1.1` is stored in the file `Bex2.1.1.gms`. They are quadratic optimization problems except `Balkyl` which involves polynomials of degree 3 in its equality constraints. We added or modified lower and upper bounds for the variables so that their SDP relaxations become more effective. The relative objective function error `rObjErr` and the absolute feasibility error are evaluated at approximate optimal solutions obtained by SeDuMi or SDPA followed by the Matlab Optimization Toolbox `fmincon`. In all problems, the obtained approximation to optimal solutions satisfy the constraints accurately as shown in the “absErr” column. But the relative objective function error `rObjErr` varies from $1.8e+0$ to $6.8e-11$. In

cases with \dagger attached to their relaxation order ω , numerical results from the dense SDP relaxation are shown because the sparse SDP relaxation did not attain high relative objective function error.

The relative objective function error `rObjErr` remained large in `ex5.2.5` case with $\omega = 1$ and `ex5.3.2` case with $\omega = 2$ because the SDP lower bounds for their optimal values were not tight. However, SparsePOP attained a smaller value when larger relaxation order ω was used, although it took much longer for the SDP solvers SeDuMi and SDPA to solve the SDP relaxation problems. We note that SDPA was more than 10 times faster in `ex5.2.5` case and 60 times faster in `ex5.3.2` case. From Table ?? that SDPA was faster to solve SDP relaxation problems, but the obtained approximate solutions were less accurate than SeDuMi in most cases.

Tables 3 shows numerical results for large-sized and highly-sparse unconstrained polynomial optimization problems from the literature [2, 10, 11]. In all cases, we took the relaxation order $\omega = 2$. The notation `sizeA` and `nnzA` stands for the size of coefficient matrix \mathbf{A} of SDP relaxation problem in SeDuMi format and the nonzeros in \mathbf{A} , respectively. We see from the Table that the size of row and column, and the number of the nonzeros of \mathbf{A} increase almost linearly as the number of variables increases. In case $n = 1000$, we applied the Matlab Optimization Toolbox `fminunc` to refine the solutions obtained by the SDP solver SeDuMi or SDPA as shown in the “+fminunc” row. It did not work effectively to improve the accuracy. It should be noted that the relative objective function error `rObjErr` defined by (6) must be nonnegative in theory. We observe, however, the “`rObjErr`” column of SeDuMi involves negative values, which are probably due to numerical errors occurred during the execution of SeDuMi. We observe that SDPA is faster than SeDuMi.

9 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 5000 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 [14] 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] A. R. Conn, N. I. M. Gould, and Ph. L. Toint: “Testing a class of methods for solving minimization problems with simple bounds on the variables,” *Math. Comp.* **50** (1988) 399–430.

			SeDuMi+fmincon			SDPA+fmincon		
Problem	n	ω	eTime	rObjErr	absErr	eTime	rObjErr	absErr
ex2_1_1	5	3	1.0(0.5)	7.1e-08	0.0e+00	0.8(0.0)	5.6e-06	0.0e+00
ex2_1_2	6	2	0.1(0.2)	6.8e-11	0.0e+00	0.1(0.0)	9.3e-08	0.0e+00
ex2_1_3	13	2	0.1(0.0)	2.6e-09	0.0e+00	0.1(0.0)	3.9e-08	0.0e+00
ex2_1_4	6	2	0.1(0.0)	2.1e-09	0.0e+00	0.1(0.0)	3.2e-08	0.0e+00
ex2_1_5	10	2	0.6(0.0)	3.5e-10	0.0e+00	1.3(0.0)	1.9e-08	0.0e+00
ex2_1_8	24	2	15.4(0.1)	5.0e-09	-3.1e-13	5.7(0.3)	2.6e-06	-5.1e-16
ex3_1_1	8	3	0.5(0.0)	1.1e-09	-7.5e-17	0.5(0.0)	2.5e-07	-1.1e-13
ex3_1_2	5	2	0.1(0.0)	7.0e-11	0.0e+00	0.1(0.0)	2.7e-09	0.0e+00
ex3_1_4	9	4	1.0(0.0)	1.2e-06	0.0e+00	0.5(0.0)	6.4e-08	0.0e+00
ex5_2_2_case1	9	2 [†]	0.7(0.0)	3.8e-08	-4.1e-10	0.4(0.0)	2.0e-04	-2.6e-09
ex5_2_2_case2	9	2 [†]	0.6(0.0)	2.9e-07	-8.2e-09	0.4(0.0)	5.9e-04	-1.1e-09
ex5_2_2_case3	9	2 [†]	0.5(0.0)	1.4e-09	-1.9e-16	0.4(0.0)	1.4e-05	-2.5e-23
ex5_2_5	32	1	0.1(43.1)	1.8e+00	-1.1e-09	0.0(10.5)	3.0e+00	-1.0e-15
ex5_2_5	32	2	10550.4(0.8)	1.1e-03	-8.5e-09	1027.3(2.0)	2.3e-03	-2.2e-16
ex5_3_2	22	2	47.1(5.3)	1.5e-01	-9.9e-17	1.0(5.3)	1.5e-01	-5.9e-17
ex5_3_2	22	3	35535.1(0.9)	1.7e-03	-1.9e-16	504.7(0.8)	1.3e-04	-2.5e-14
ex5_4_2	8	3	0.5(0.0)	8.2e-09	-1.7e-16	0.5(0.0)	9.0e-09	-2.0e-16
ex9_1_1	13	2	0.6(0.6)	1.4e-09	-3.9e-08	0.1(0.1)	4.7e-07	-1.4e-10
ex9_1_2	10	3	2.6(0.0)	2.9e-05	-3.4e-07	0.5(0.0)	4.5e-05	-7.5e-14
ex9_1_4	10	2	0.5(0.0)	2.4e-05	-3.2e-07	0.0(0.0)	2.4e-05	-3.2e-07
ex9_1_5	13	2	0.3(0.0)	7.1e-05	-3.4e-11	0.1(0.0)	7.3e-05	-9.0e-11
ex9_1_8	14	2	0.2(0.2)	4.0e-10	-8.2e-09	0.1(0.3)	9.5e-07	-2.1e-11
ex9_2_1	10	2 [†]	0.8(0.0)	9.4e-10	-5.6e-08	0.3(0.0)	4.7e-06	-1.0e-09
ex9_2_2	10	2	0.4(0.0)	3.2e-05	-1.5e-12	0.1(0.1)	2.4e-04	-7.6e-13
ex9_2_3	16	2	0.5(0.0)	8.3e-07	-1.2e-16	0.1(0.0)	2.0e-05	-6.4e-13
ex9_2_4	8	2 [†]	0.3(0.0)	3.2e-08	-1.0e-14	0.2(0.0)	2.1e-03	-2.6e-10
ex9_2_5	8	2	0.1(0.0)	-1.1e-08	-6.3e-08	0.1(0.0)	6.8e-06	-1.9e-09
ex9_2_6	16	2 [†]	1.8(0.0)	4.2e-07	-1.3e-08	0.8(0.0)	2.6e-03	-2.1e-12
ex9_2_7	10	2	0.7(0.0)	2.8e-09	-3.9e-08	0.4(0.0)	1.3e-06	-1.4e-11
ex9_2_8	6	2 [†]	0.1(0.0)	7.4e-11	-8.4e-21	0.0(0.0)	6.7e-07	-8.4e-21
Balkyl	14	3	2.9(0.2)	1.8e-09	-1.2e-09	1.2(0.1)	1.4e-06	-2.9e-13

Table 2: Numerical results on SparsePOP applied to constrained problems. Here n denotes the number of variables and ω the relaxation order with which SparsePOP is applied. The symbol [†] in the ω column indicates that the dense relaxation with setting `param.sparseSW` = 0 was applied.

	SDP relaxation problem		SeDuMi		SDPA	
Problem(n)	sizeA	nnzA	eTime	rObjErr	eTime	rObjErr
ChainedWood(1000) +fminunc	4499 \times 12491	11492	1.3 -2.8e-05 1.3(43.3) -4.1e-05		0.6 4.4e-04 0.6(1.5) 4.4e-04	
ChainedWood(5000)	22499 \times 62491	57492	6.2 -1.4e-03		3.4 2.2e-03	
ChainedWood(10000)	44999 \times 124991	114992	15.6 -1.2e-03		7.5 4.4e-03	
RosenbrockLS(1000,1) +fminunc	9988 \times 35956	34957	3.9 -8.4e-05 3.9(27.1) -8.5e-05		1.9 6.7e-06 1.9(4.4) 6.7e-06	
RosenbrockLS(5000,1)	49988 \times 179956	174957	23.2 -4.6e-04		11.7 3.3e-05	
RosenbrockLS(10000,1)	99988 \times 359956	349957	48.8 -9.3e-04		23.8 1.1e-04	
BroydenTriLS(1000) +fminunc	19974 \times 99819	98821	21.5 -1.1e-05 21.7(17.0) -1.1e-05		5.7 4.3e-06 5.8(30.7) 3.6e-06	
BroydenTriLS(5000)	99974 \times 499819	494821	57.8 -1.4e-05		32.0 5.1e-05	
ChainedSingular(1000) +fminunc	19974 \times 99800	98802	27.4 5.4e-08 27.3(2.8) 5.4e-08		7.1 8.8e-04 7.2(2.1) 8.8e-04	
ChainedSingular(5000)	99974 \times 499800	494802	86.2 5.0e-08		41.1 2.9e-04	

Table 3: Numerical results on SparsePOP applied to large-sized and highly sparse unconstrained problems. Here n denotes the number of variables.

- [3] K. Fujisawa, M. Fukuda, K. Kobayashi, M. Kojima, K. Nakata, M. Nakata and M. Yamashita: “SDPA (SemiDefinite Programming Algorithm) User’s Manual — Version 7.0.5,” Research Report B-448, Dept. of Mathematical and Computing Sciences, Tokyo Institute of Technology, Oh-Okayama, Meguro, Tokyo 152-8552, Japan, February 2008.
- [4] GAMS HomePage: <http://www.gams.com/>
- [5] GLOBAL Library: <http://www.gamsworld.org/global/globallib.htm>
- [6] 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.
- [7] M. Kojima, S. Kim and H. Waki: “Sparsity in sums of squares of polynomials,” *Mathematical Programming* **103** (2005) 45–62.
- [8] J. B. Lasserre: “Global optimization with polynomials and the problems of moments,” *SIAM Journal on Optimization* **11** (2001) 796–817.
- [9] J. B. Lasserre: “Convergent SDP-relaxations in polynomial optimization with sparsity,” *SIAM Journal on Optimization* **17** (2006) 822–843.
- [10] J. J. More, B. S. Garbow, and K. E. Hillstom: “Testing unconstrained optimization soft-ware,” *ACM Trans. Math. Software* **7** (1981) 17–41.
- [11] S. G. Nash: “Newton-type minimization via the Lanczos method,” *SIAM J. Numer. Anal.* **21** (1984) 770–788.

- [12] J. F. Strum: “SeDuMi 1.02, a MATLAB toolbox for optimization over symmetric cones,” *Optimization Methods and Software*, **11 & 12** (1999) 625–653.
- [13] R. H. Tutuncu, K. C. Toh, and M. J. Todd: “Solving semidefinite-quadratic-linear programs using SDPT3,” *Mathematical Programming* **95** (2003) 189–217.
- [14] 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.