GloptiPoly: Global Optimization over Polynomials with Matlab and SeDuMi

Didier Henrion[1,2], Jean-Bernard Lasserre[1]
henrion@laas.fr, lasserre@laas.fr
www.laas.fr/∼henrion, www.laas.fr/∼lasserre

Version of February 12, 2002

Abstract

GloptiPoly is a Matlab/SeDuMi add-on to build and solve convex linear matrix inequality relaxations of the (generally non-convex) global optimization problem of minimizing a multivariable polynomial function subject to polynomial inequality, equality or integer constraints. It generates a series of lower bounds monotonically converging to the global optimum. Numerical experiments show that for most of the small- and medium-scale problems described in the literature, the global optimum is reached at low computational cost.

1 Introduction

GloptiPoly is a Matlab utility that builds and solves convex linear matrix inequality (LMI) relaxations of (generally non-convex) global optimization problems with multivariable real-valued polynomial criterion and constraints. It is based on the theory described in [5, 6]. Related results can be found also in [9] and [10]. GloptiPoly does not intent to solve non-convex optimization problems globally, but allows to solve a series of convex relaxations of increasing size, whose optima are guaranteed to converge monotonically to the global optimum.

GloptiPoly solves LMI relaxations with the help of the solver SeDuMi [11], taking full advantage of sparsity and special problem structure. Optionally, a user-friendly interface called DefiPoly, based on Matlab Symbolic Math Toolbox, can be used jointly with GloptiPoly to define the optimization problems symbolically with a Maple-like syntax.

GloptiPoly is aimed at small- and medium-scale problems. Numerical experiments illustrate that for most of the problem instances available in the literature, the global optimum is reached exactly with LMI relaxations of medium size, at a relatively low computational cost.

[1]Laboratoire d'Analyse et d'Architecture des Systèmes, Centre National de la Recherche Scientifique, 7 Avenue du Colonel Roche, 31 077 Toulouse, cedex 4, France

[2]Also with the Institute of Information Theory and Automation, Academy of Sciences of the Czech Republic, Pod vodárenskou věží 4, 182 08 Praha, Czech Republic

2 Installation

GloptiPoly requires Matlab version 5.3 or higher [8], together with the freeware solver SeDuMi version 1.05 [11]. Moreover, the Matlab source file `gloptipoly.m` must be installed in the current working directory, see

`www.laas.fr/∼henrion/software/gloptipoly`

The optional, companion Matlab source files to GloptiPoly, described throughout this manuscript, can be found at the same location.

3 Getting started

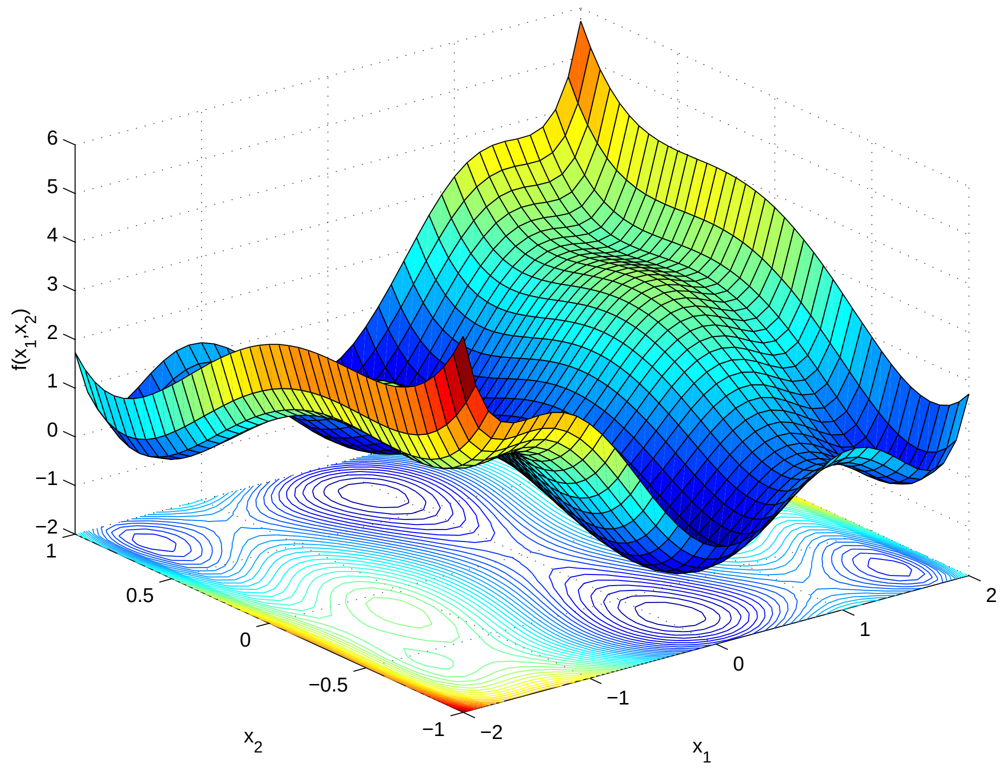

Figure 1: Six-hump camel back function.

Consider the classical problem of minimizing globally the two-dimensional six-hump camel back function [3, Pb. 8.2.5]

$$f(x_1, x_2) = x_1^2(4 - 2.1x_1^2 + x_1^4/3) + x_1x_2 + x_2^2(-4 + 4x_2^2).$$

The function has six local minima, two of them being global minima, see figure 1.

To minimize this function we build the coefficient matrix

$$P = \begin{bmatrix} 0 & 0 & -4 & 0 & 4 \\ 0 & 1 & 0 & 0 & 0 \\ 4 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ -2.1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1/3 & 0 & 0 & 0 & 0 \end{bmatrix}$$

where each entry (i, j) in P contains the coefficient of the monomial $x_1^i x_2^j$ in polynomial $f(x_1, x_2)$. We invoke GloptiPoly with the following Matlab script:

```
>> P(1,3) = -4; P(1,5) = 4; P(2,2) = 1;
>> P(3,1) = 4; P(5,1) = -2.1; P(7,1) = 1/3;
>> gloptipoly(P);
```

On our platform, a Sun Blade 100 workstation with 640 Mb of RAM running under SunOS 5.8, we obtain the following output:

```
GloptiPoly 1.00 - Global Optimization over Polynomials with SeDuMi
Number of variables = 2
Number of constraints = 0
Maximum polynomial degree = 6
Order of LMI relaxation = 3
Building LMI. Please wait..
 ...
SeDuMi 1.05 Development by Jos F. Sturm, 1998, 2001.
Alg = 2: xz-corrector, Step-Differentiation, theta = 0.250, beta = 0.500
eqs m = 27, order n = 11, dim = 101, blocks = 2
nnz(A) = 54 + 0, nnz(ADA) = 729, nnz(L) = 378
 it :     b*y       gap    delta  rate   t/tP*  t/tD*   feas cg cg
  0 :            3.86E+00 0.000
  1 :   9.86E-01 8.50E-01 0.000 0.2203 0.9000 0.9015   0.73  1  1
  2 :   9.43E-01 7.30E-02 0.000 0.0858 0.9900 0.9903   1.64  1  1
        ...
 12 :   1.03E+00 1.46E-09 0.000 0.2186 0.9000 0.9130   0.98  4  4
 13 :   1.03E+00 3.35E-10 0.000 0.2299 0.9000 0.9003   0.96  3  3
iter seconds digits       c*x               b*y
 13      1.1   9.5  1.0316284534e+00  1.0316284531e+00
|Ax-b| =   1.0e-09, [Ay-c]_+ =   1.6E-11, |x|=  6.6e+00, |y|=  1.1e+00
Max-norms: ||b||=4, ||c|| = 1,
Cholesky |add|=1, |skip| = 1, ||L.L|| = 500000.
CPU time = 0.63 sec
Norm of SeDuMI primal variables = 6.56
Norm of SeDuMI dual variables = 1.1127
Criterion = -1.0316
```

Since GloptiPoly solves convex relaxations of non-convex problems, it generally returns lower bounds on the global minimum. Here it turns out that the computed bound

$$f(\hat{x}_1,\, \hat{x}_2) \approx -1.0316$$

is equal to the global minimum.

4 GloptiPoly input arguments. Defining and solving an optimization problem

4.1 Handling constraints. Basic syntax

Consider the concave optimization problem of finding the radius of the intersection of three ellipses [4]:

$$\begin{array}{ll} \max & x_1^2 + x_2^2 \\ \text{s.t.} & 2x_1^2 + 3x_2^2 + 2x_1x_2 \leq 1 \\ & 3x_1^2 + 2x_2^2 - 4x_1x_2 \leq 1 \\ & x_1^2 + 6x_2^2 - 4x_1x_2 \leq 1. \end{array}$$

In order to specify both the objective and the constraint to GloptiPoly, we first transform the problem into a minimization problem over non-negative constraints, i.e.

$$\begin{array}{ll} \min & \begin{bmatrix} 1 \\ x_1 \\ x_1^2 \end{bmatrix}^T \begin{bmatrix} 0 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ x_2 \\ x_2^2 \end{bmatrix} \\ \text{s.t.} & \begin{bmatrix} 1 \\ x_1 \\ x_1^2 \end{bmatrix}^T \begin{bmatrix} 1 & 0 & -3 \\ 0 & -2 & 0 \\ -2 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ x_2 \\ x_2^2 \end{bmatrix} \geq 0 \\ & \begin{bmatrix} 1 \\ x_1 \\ x_1^2 \end{bmatrix}^T \begin{bmatrix} 1 & 0 & -2 \\ 0 & 4 & 0 \\ -3 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ x_2 \\ x_2^2 \end{bmatrix} \geq 0 \\ & \begin{bmatrix} 1 \\ x_1 \\ x_1^2 \end{bmatrix}^T \begin{bmatrix} 1 & 0 & -6 \\ 0 & 4 & 0 \\ -1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ x_2 \\ x_2^2 \end{bmatrix} \geq 0. \end{array}$$

Then we invoke GloptiPoly with four input matrices: the first input matrix corresponds to the criterion to be minimized, and the remaining input matrices correspond to the non-negative constraints to be satisfied:

```
>> P{1} = [0 0 -1; 0 0 0; -1 0 0];
>> P{2} = [1 0 -3; 0 -2 0; -2 0 0];
>> P{3} = [1 0 -2; 0 4 0; -3 0 0];
>> P{4} = [1 0 -6; 0 4 0; -1 0 0];
>> gloptipoly(P);
```

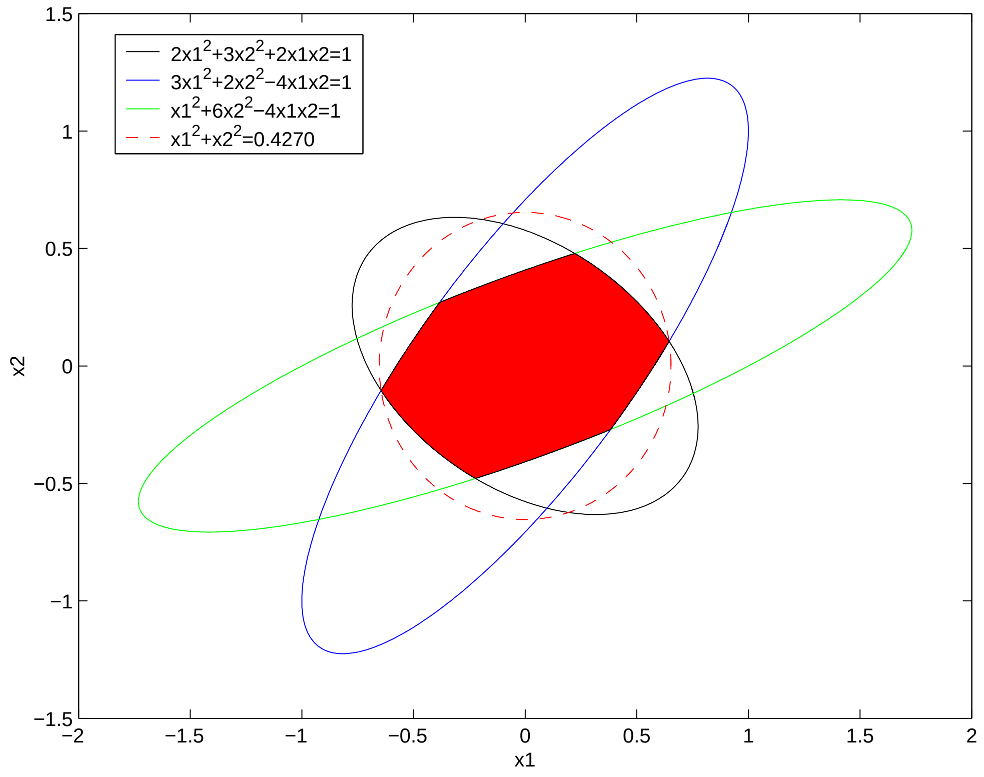

Figure 2: Radius of the intersection of three ellipses.

We obtain 0.4270 as the global optimum, see figure 2.

More generally, when input argument `P` is a cell array of coefficient matrices, the instruction `gloptipoly(P)` solves the problem of minimizing the criterion whose polynomial coefficients are contained in matrix `P{1}`, subject to the constraints that the polynomials whose coefficients are contained in matrices `P{2}`, `P{3}`.. are all non-negative.

4.2 Handling constraints. General syntax

To handle directly maximization problems, non-positive inequality or equality constraints, a more explicit but somehow more involved syntax is required. The input argument `P` must be a cell array of structures with fields:

`P{i}.c` - polynomial coefficient matrices;

`P{i}.t` - identification string, either

- `'min'` - criterion to minimize, or
- `'max'` - criterion to maximize, or
- `'>='` - non-negative inequality constraint, or
- `'<='` - non-positive inequality constraint, or

`'=='` - equality constraint.

For example, if we want to solve the optimization problem [3, Pb. 4.9]

$$\begin{array}{ll} \min & -12x_1 - 7x_2 + x_2^2 \\ \text{s.t.} & -2x_1^4 - x_2 + 2 = 0 \\ & 0 \leq x_1 \leq 2 \\ & 0 \leq x_2 \leq 3 \end{array}$$

we use the following script:

```
>> P{1}.c = [0 -7 1; -12 0 0]; P{1}.t = 'min';
>> P{2}.c = [2 -1; 0 0; 0 0; 0 0; -2 0]; P{2}.t = '==';
>> P{3}.c = [0; -1]; P{3}.t = '<=';
>> P{4}.c = [-2; 1]; P{4}.t = '<=';
>> P{5}.c = [0 -1]; P{5}.t = '<=';
>> P{6}.c = [-3 1]; P{6}.t = '<=';
>> gloptipoly(P);
```

We obtain -16.7389 as the (global) optimal value, see figure 3.

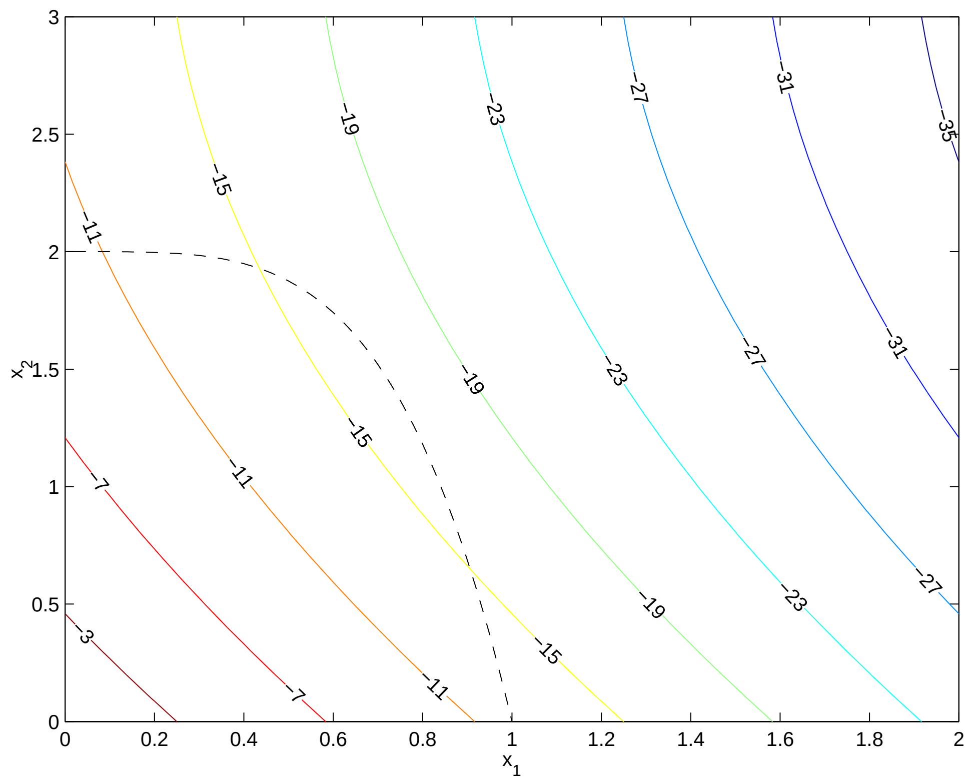

Figure 3: Contour plot of $-12x_1 - 7x_2 + x_2^2$ with constraint $-2x_1^4 - x_2 + 2 = 0$ in dashed line.

4.3 Sparse polynomial coefficients. Saving memory

When defining optimization problems with a lot of variables or polynomials of high degrees, the coefficient matrix associated with a polynomial criterion or constraint may require a lot of memory to be stored in Matlab. For example in the case of a quadratic program with 10 variables, the number of entries of the coefficient matrix may be as large as $(2+1)^{10} = 59049$.

An alternative syntax allows to define coefficient matrices of Matlab `sparse` class. Because sparse Matlab matrices cannot have more than two dimensions, we store them as sparse column vectors in coefficient field `P.c`, with an additional field `P.s` which is the vector of dimensions of the coefficient matrix, as returned by Matlab function `size` if the matrix were not sparse.

For example, to define the quadratic criterion

$$\min \sum_{i=1}^{10} i x_i^2$$

the instructions

```
>> P.c(3,1,1,1,1,1,1,1,1,1) = 1;
>> P.c(1,3,1,1,1,1,1,1,1,1) = 2;
   ...
>> P.c(1,1,1,1,1,1,1,1,1,3) = 10;
```

would create a 10-dimensional matrix `P.c` requiring 472392 bytes for storage. The equivalent instructions

```
>> P.s = 3*ones(1,10);
>> P.c = sparse(prod(P.s),1);
>> P.c(sub2ind(P.s,3,1,1,1,1,1,1,1,1,1)) = 1;
>> P.c(sub2ind(P.s,1,3,1,1,1,1,1,1,1,1)) = 2;
   ...
>> P.c(sub2ind(P.s,1,1,1,1,1,1,1,1,1,3)) = 10;
```

create a sparse matrix `P.c` requiring only 140 bytes for storage.

Note however that the maximum index allowed by Matlab to refer to an element in a vector is $2^{31} - 2 = 2147483646$. As a result, if d denotes the maximum degree and n the number of variables in the optimization problem, then the current version of GloptiPoly cannot handle polynomials for which $(d+1)^n > 2^{31}$. For example, GloptiPoly cannot handle quadratic polynomials with more than 19 variables.

4.4 DefiLin and DefiQuad: easy definition of linear and quadratic expressions

Linear and quadratic expressions arise frequently in optimization problems. In order to enter these expressions easily into GloptiPoly, we wrote two simple Matlab scripts called DefiLin and DefiQuad respectively. Refer to section 2 to download the Matlab source files `defilin.m` and `defiquad.m`.

Given a matrix `A` and a vector `b`, the instruction

```
P = defilin(A, b, type)
```

allows to define a linear expression whose type is specified by the third input argument

`min` - linear criterion $\mathtt{A}x + \mathtt{b}$ to minimize, or

`max` - linear criterion $\mathtt{A}x + \mathtt{b}$ to maximize, or

`>=` - inequality $\mathtt{A}x + \mathtt{b} \geq 0$, or

`<=` - inequality $\mathtt{A}x + \mathtt{b} \leq 0$, or

`==` - equality $\mathtt{A}x + \mathtt{b} = 0$.

By default, `b=0` and `type='>='`. Output argument `P` is then a cell array of structures complying with the sparse syntax introduced in 4.3. There are as many structures in `P` as the number of rows in matrix `A`.

Similarly, given a square matrix `A`, a vector `b` and a scalar `c`, the instruction

```
P = defiquad(A, b, c, type)
```

allows to define a quadratic expression $x^T\mathtt{A}x + 2x^T\mathtt{b} + \mathtt{c}$. Arguments `type` and `P` have the same meaning as above.

For example, consider the quadratic problem [3, Pb. 3.5]:

$$\begin{array}{ll} \min & -2x_1 + x_2 - x_3 \\ \text{s.t.} & x^TA^TAx - 2b^TAx + b^Tb - 0.25(c-d)^T(c-d) \geq 0 \\ & x_1 + x_2 + x_3 \leq 4, \quad 3x_2 + x_3 \leq 6 \\ & 0 \leq x_1 \leq 2, \quad 0 \leq x_2, \quad 0 \leq x_3 \leq 3 \end{array}$$

where

$$A = \begin{bmatrix} 0 & 0 & 1 \\ 0 & -1 & 0 \\ -2 & 1 & -1 \end{bmatrix} \quad b = \begin{bmatrix} 1.5 \\ -0.5 \\ -5 \end{bmatrix} \quad c = \begin{bmatrix} 3 \\ 0 \\ -4 \end{bmatrix} \quad d = \begin{bmatrix} 0 \\ -1 \\ -6 \end{bmatrix}.$$

To define this problem with DefiLin and DefiQuad we use the following Matlab script:

```
>> A = [0 0 1;0 -1 0;-2 1 -1];
>> b = [1.5;-0.5;-5]; c = [3;0;-4]; d = [0;-1;-6];
>> crit = defilin([-2 1 -1], [], 'min');
>> quad = defiquad(A'*A, -b'*A, b'*b-0.25*(c-d)'*(c-d));
>> lin = defilin([-1 -1 -1;0 -3 -1;eye(3);-1 0 0;0 0 -1], [4;6;0;0;0;2;3]);
>> P = {crit{:}, quad, lin{:}};
```

4.5 DefiPoly: defining polynomial expressions symbolically

When multivariable expressions are not linear or quadratic, it may be lengthy to build polynomial coefficient matrices. We wrote a Matlab/Maple script called DefiPoly to define polynomial objective and constraints symbolically. It requires the Symbolic Math Toolbox version 2.1, which is the Matlab gateway to the kernel of Maple V [7]. See section 2 to retrieve the Matlab source file `defipoly.m`.

The syntax of DefiPoly is as follows:

```
P = defipoly(poly, indets)
```

where both input arguments are character strings. The first input argument `poly` is a Maple-valid polynomial expression with an additional keyword, either

`min` - criterion to minimize, or

`max` - criterion to maximize, or

>= - non-negative inequality, or

<= - non-positive inequality, or

== - equality.

The second input argument `indets` is a comma-separated ordered list of indeterminates. It establishes the correspondence between polynomial variables and indices in the coefficient matrices. For example, the instructions

```
>> P{1} = defipoly('min -12*x1-7*x2+x2^2', 'x1,x2');
>> P{2} = defipoly('-2*x1^4+2-x2 == 0', 'x1,x2');
>> P{3} = defipoly('0 <= x1', 'x1,x2');
>> P{4} = defipoly('x1 <= 2', 'x1,x2');
>> P{5} = defipoly('0 <= x2', 'x1,x2');
>> P{6} = defipoly('x2 <= 3', 'x1,x2');
```

build the structure `P` defined in section 4.2.

When there are more than 100 entries in the coefficient matrix, DefiPoly switches automatically to GloptiPoly's sparse coefficient format, see section 4.3.

One can also specify several expressions at once in a cell array of strings, the output argument being then a cell array of structures. For example the instruction

```
>> P = defipoly({'min -12*x1-7*x2+x2^2', '-2*x1^4+2-x2 == 0', ...
   '0 <= x1', 'x1 <= 2', '0 <= x2', 'x2 <= 3'}, 'x1,x2');
```

is equivalent to the six instructions above.

4.6 Increasing the order of the LMI relaxation

GloptiPoly solves convex LMI relaxations of generally non-convex problems, so it may happen that it does not return systematically the global optimum. With the syntax used so far, GloptiPoly solves the simplest LMI relaxation, sometimes called Shor's relaxation. As described in [5, 6], there exist other, more complicated LMI relaxations, classified according to their order.

When the relaxation order increases, the number of variables as well as the dimension of the LMI increase as well. Moreover, the successive LMI feasible sets are inscribed within each other. More importantly, the series of optima of LMI relaxations of increasing orders converges monotonically to the global optimum. For a lot of practical problems, the exact global optimum is reached quickly, at a small relaxation order (say 2, 3 or 4).

The order of the LMI relaxation, a strictly positive integer, can be specified to GloptiPoly as follows:

```
gloptipoly(P, order)
```

The minimal relaxation order, corresponding to Shor's relaxation, is such that twice the order is greater than or equal to the maximal degree occurring in the polynomial expressions of the original optimization problem. By default, it is the order of the LMI relaxation solved by GloptiPoly when there is no second input argument. If the specified order is less than the minimal relaxation order, an error message is issued.

As an example, consider quadratic problem [3, Pb 3.5] introduced in section 4.4. When applying LMI relaxations of increasing orders to this problem we obtain a monotically increasing series of lower bounds on the global optimum, given in table 1. It turns out that the global optimum -4 is reached at the fourth LMI relaxation. See section 5.2 to learn more about how to detect global optimality of an LMI relaxation.

One can notice that the number of LMI variables and the size of the LMI problem, hence the overall computational time, increase quickly with the relaxation order.

Relaxation order	LMI optimum	Number of LMI variables	Size of LMI	CPU time in seconds
1	-6.0000	9	24	0.22
2	-5.6923	34	228	2.06
3	-4.0685	83	1200	4.13
4	-4.0000	164	4425	6.47
5	-4.0000	285	12936	32.7
6	-4.0000	454	32144	142

Table 1: Characteristics of successive LMI relaxations.

4.7 Tuning SeDuMi parameters

If the solution returned by GloptiPoly is not accurate enough, one can specify the desired accuracy to SeDuMi. In a similar way, one can suppress the screen output, change the algorithm or tune the convergence parameters in SeDuMi. This can be done by specifying a third input argument:

```
gloptipoly(P, [], pars)
```

which is a Matlab structure complying with SeDuMi's syntax:

`pars.eps` - Required accuracy, default `1e-9`;

`pars.fid` - 0 for no screen output in both GloptiPoly and SeDuMi, default 1;

`pars.alg`, `pars.beta`, `pars.theta` - SeDuMi algorithm parameters.

Refer to SeDuMi user's guide for more information on other fields in `pars` to override default parameter settings.

4.8 Unbounded LMI relaxations. Feasibility radius

With some problems, it may happen that LMI relaxations of low orders are not stringent enough. As a result, the criterion is not bounded, LMI decision variables can reach large values which may cause numerical difficulties. In this case, GloptiPoly issues a warning message saying that either SeDuMi primal problem is infeasible, or that SeDuMi dual problem is unbounded. See section 5.1 for more information about SeDuMi primal and dual problems.

As a remedy, we can enforce a compacity constraint on the variables in the original optimization problem. For example in the case of three variables, we may specify the Euclidean norm constraint `'x1^2+x2^2+x3^2 <= radius'` as an additional string argument to DefiPoly, where the positive real number `radius` is large enough, say `1e9`.

Another, slightly different way out is to enforce a feasibility radius on the LMI decision variables within the SeDuMi solver. A large enough positive real number can be specified as an additional field

`pars.radius` - Feasibility radius, default none;

in the SeDuMi parameter structure `pars` introduced in section 4.7. All SeDuMi dual variables are then constrained to a Lorenz cone.

4.9 Integer constraints

GloptiPoly can handle integer constraints on some of the optimization variables. An optional additional field

`P{i}.v` - vector of integer constraints

can be inserted into GloptiPoly's input cell array `P`. This field is required only once in the problem definition, at an arbitrary index `i`. If the field appears more than once, then only the field corresponding to the largest index `i` is valid.

Each entry in vector `P{i}.v` refers to one optimization variable. It can be either

`0` - unrestricted continuous variable, or

`-1` - discrete variable equal to -1 or $+1$, or

`+1` - discrete variable equal to 0 or $+1$.

For example, consider the quadratic 0-1 problem [3, Pb. 13.2.1.1]:

$$\begin{array}{ll}
\min & \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}^T \begin{bmatrix} 0 & 3 & 4 & 2 & -1 \\ 3 & -1/2 & 1 & 0 & 0 \\ 4 & 1 & -1/2 & 1 & 0 \\ 2 & 0 & 1 & -1/2 & 1 \\ -1 & 0 & 0 & 1 & -1/2 \end{bmatrix} \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} \\
\text{s.t.} & -1 \leq x_1x_2 + x_3x_4 \leq 1 \\
& -3 \leq x_1 + x_2 + x_3 + x_4 \leq 2 \\
& x_i \in \{-1,\ +1\}, \quad i = 1, \ldots, 4.
\end{array}$$

The problem can be solved with the following script:

```
>> P = defipoly({['min (-x1^2-x2^2-x3^2-x4^2)/2+' ...
   '2*(x1*x2+x2*x3+x3*x4)+2*(3*x1+4*x2+2*x3-x4)'],...
   '-1<=x1*x2+x3*x4', 'x1*x2+x3*x4<=1',...
   '-3<=x1+x2+x3+x4', 'x1+x2+x3+x4<=2'}, 'x1,x2,x3,x4');
>> P{1}.v = [-1 -1 -1 -1];
>> gloptipoly(P);
```

We obtain -20 at the first LMI relaxation. It turns out to be the global optimum.

Another, classical integer programming problem is the Max-Cut problem. Given an undirected graph with weighted edges, it consists in finding a partition of the set of nodes into two parts so as to maximize the sum of the weights on the edges that are cut by the partition. If w_{ij} denotes the weight on the edge between nodes i and j, the Max-Cut problem can be formulated as

$$\begin{array}{ll} \max & \frac{1}{2}\sum_{i<j} w_{ij}(1-x_i x_j) \\ \text{s.t.} & x_i \in \{-1,\ +1\}\,. \end{array}$$

Given the weighted adjacency matrix W with entries w_{ij}, the instruction

```
P = defimaxcut(W)
```

transforms a Max-Cut problem into GloptiPoly's sparse input format. To download function `defimaxcut.m`, consult section 2.

For example, consider the antiweb AW_9^2 graph [1, p. 67] shown in figure 4 with unit

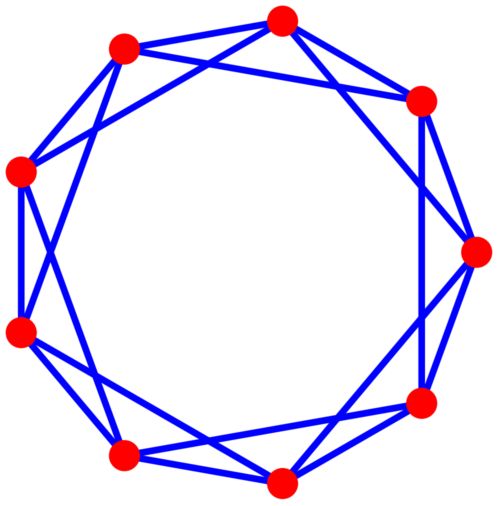

Figure 4: Antiweb AW_9^2 graph.

adjacency matrix

$$W = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{bmatrix}.$$

Entering W into Matlab's environment, and running the instruction

```
>> gloptipoly(defimaxcut(W), 3);
```

to solve the third LMI relaxation, GloptiPoly returns the global optimum 12. Note that none of the LMI relaxation methods described in [1] could reach the global optimum.

5 GloptiPoly output arguments. Interpreting SeDuMi input and output parameters

5.1 SeDuMi problem structure

With the syntax

```
[crit, sedumi] = gloptipoly(P)
```

all the information regarding the LMI problem solved by SeDuMi is stored into the structure `sedumi`. To understand the meaning of the various fields in structure `sedumi`, it is better to proceed with an example.

Consider the well-known problem of minimizing Rosenbrock's banana function

$$\min\,(1-x_1)^2+100(x_2-x_1^2)^2=-(-1+2x_1-x_1^2-100x_2^2+200x_1^2x_2-100x_1^4)$$

whose contour plot is shown on figure 5. To build LMI relaxations of this problem, we replace each monomial with a new decision variable:

$$\begin{array}{rcl} x_1 & \rightarrow & y_{10} \\ x_2 & \rightarrow & y_{01} \\ x_1^2 & \rightarrow & y_{20} \\ x_1x_2 & \rightarrow & y_{11} \\ x_2^2 & \rightarrow & y_{02} \\ x_1^3 & \rightarrow & y_{30} \\ x_1^2x_2 & \rightarrow & y_{21} \\ & \text{etc..} & \end{array}$$

Decision variables y_{ij} satisfy non-convex relations such as $y_{10}y_{01}=y_{11}$ or $y_{20}=y_{10}^2$ for example. To relax these non-convex relations, we enforce the LMI constraint

$$\left[\begin{array}{c|cc|ccc} 1 & y_{10} & y_{01} & y_{20} & y_{11} & y_{02} \\ \hline y_{10} & y_{20} & y_{11} & y_{30} & y_{21} & y_{12} \\ y_{01} & y_{11} & y_{02} & y_{21} & y_{12} & y_{03} \\ \hline y_{20} & y_{30} & y_{21} & y_{40} & y_{31} & y_{22} \\ y_{11} & y_{21} & y_{12} & y_{31} & y_{22} & y_{13} \\ y_{02} & y_{12} & y_{03} & y_{22} & y_{13} & y_{04} \end{array}\right] \in K$$

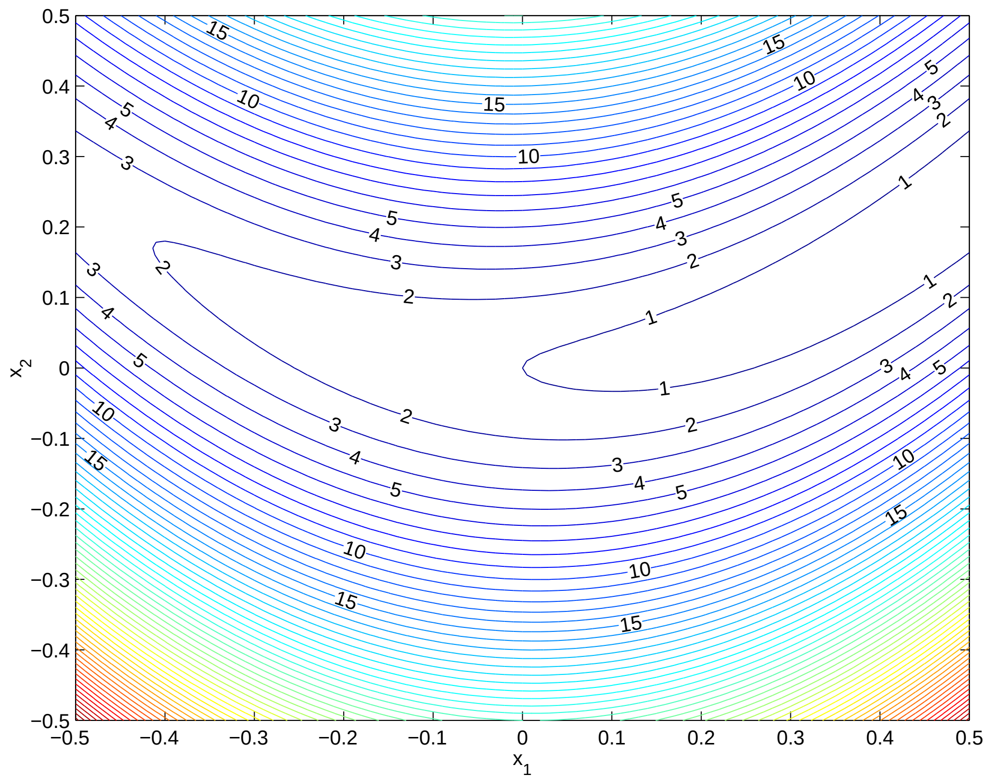

Figure 5: Contour plot of Rosenbrock's banana function.

where K is the cone of 6×6 PSD matrices. Following the terminology introduced in [5, 6], the above matrix is referred to as the moment, or measure matrix associated with the LMI relaxation. Because the above moment matrix contains relaxations of monomials of degree up to 2+2=4, it is referred to as the second-degree moment matrix. The above upper-left 3x3 submatrix contains relaxations of monomials of degree up to 1+1=2, so it is referred to as the first-degree moment matrix.

Now replacing the monomials in the criterion by their relaxed variables, the first LMI relaxation of Rosenbrock's banana function minimization reads

$$\begin{array}{ll} \max & -1 + 2y_{10} - y_{20} - 100y_{02} + 200y_{21} - 100y_{40} \\ \text{s.t.} & \left[\begin{array}{cccccc} 1 & y_{10} & y_{01} & y_{20} & y_{11} & y_{02} \\ y_{10} & y_{20} & y_{11} & y_{30} & y_{21} & y_{12} \\ y_{01} & y_{11} & y_{02} & y_{21} & y_{12} & y_{03} \\ y_{20} & y_{30} & y_{21} & y_{40} & y_{31} & y_{22} \\ y_{11} & y_{21} & y_{12} & y_{31} & y_{22} & y_{13} \\ y_{02} & y_{12} & y_{03} & y_{22} & y_{13} & y_{04} \end{array}\right] \in K. \end{array}$$

For a comprehensive description of the way LMI relaxations are build (relaxations of higher orders, moment matrices of higher degrees and moment matrices associated with constraints). the interested reader is advised to consult [5, 6]. All we need to know here is that an LMI relaxation of a non-convex optimization problem can be expressed as a convex conic optimization problem

$$\begin{array}{ll} \max & b^T y \\ \text{s.t.} & c - A^T y \in K \end{array}$$

which is called the dual problem in SeDuMi. Decision variables y are called LMI relaxed variables. Associated with the dual problem is the primal SeDuMi problem:

$$\begin{array}{ll} \min & c^T x \\ \text{s.t.} & Ax = b \\ & x \in K. \end{array}$$

In both problems K is the same self-dual cone made of positive semi-definite (PSD) constraints. Problem data can be found in the structure `sedumi` returned by GloptiPoly:

`sedumi.A`, `sedumi.b`, `sedumi.c` - LMI problem data A (matrix), b (vector), c (vector);

`sedumi.K` - structure of cone K;

`sedumi.x` - optimal primal variables x (vector);

`sedumi.y` - optimal dual variables y (vector);

`sedumi.info` - SeDuMi information structure;

with additional fields specific to GloptiPoly:

`sedumi.M` - moment matrices (cell array);

`sedumi.pows` - variable powers (matrix).

The dimensions of PSD constraints are stored in the vector `sedumi.K.s`. Some components in K may be unrestricted, corresponding to equality constraints. The number of equality constraints is stored in `sedumi.K.f`. See SeDuMi user's guide for more information on the cone structure of primal and dual problems.

The structure `sedumi.info` contains information about algorithm convergence and feasibility of primal and dual SeDuMi problems:

when `sedumi.info.pinf = sedumi.info.dinf = 0` then an optimal solution was found;

when `sedumi.info.pinf = 1` then SeDuMi primal problem is infeasible and the LMI relaxation may be unbounded (see section 4.8 to handle this);

when `sedumi.info.dinf = 1` then SeDuMi dual problem is infeasible and the LMI relaxation, hence the original optimization problem may be infeasible as well;

when `sedumi.info.numerr = 0` then the desired accuracy was achieved (see section 4.7);

when `sedumi.info.numerr = 1` then numerical problems occurred and results may be inaccurate (tuning the desired accuracy may help, see section 4.7);

when `sedumi.info.numerr = 2` then SeDuMi completely failed due to numerical problems.

Refer to SeDuMi user's guide for a more comprehensive description of the information structure `sedumi.info`.

Output parameter `sedumi.pows` captures the correspondance between LMI relaxed variables and monomials of the original optimization variables. In the example studied above, we have

```
sedumi.pows =

     1     0
     0     1
     2     0
     1     1
     0     2
     3     0
     2     1
     1     2
     0     3
     4     0
     3     1
     2     2
     1     3
     0     4
```

For example variable y_{21} in the LMI criterion corresponds to the relaxation of monomial $x_1^2x_2$. It can be found at row 7 in matrix `sedumi.pows` so y_{21} is the 7th decision variable in SeDuMi dual vector `sedumi.y`. Similarly, variable y_{40} corresponds to the relaxation of monomial x_1^4. It is located at entry number 10 in the vector of LMI relaxed variables.

Note in particular that LMI relaxed variables are returned by GloptiPoly at the top of dual vector `sedumi.y`. They correspond to relaxations of monomials of first degree.

5.2 Detecting global optimality. Retrieving optimal solutions

When applying LMI relaxations of increasing orders it is useful to know whether the global optimum of the optimization problem has been reached, in which case it is not necessary to proceed further. It is also important to retrieve one optimal solution (or several, or all the optimal solutions if there is more than one global optimum) to the optimization problem, i.e. a set of variables satisfying the constraints and optimizing the criterion.

There are several ways how to detect global optimality and retrieve optimal solutions. In the sequel we will describe some of them.

Following the concepts introduced in section 5.1, we will denote by M_q^p the moment matrix or degree q associated with the optimal solution of the LMI relaxation of order p, as returned by GloptiPoly in matrix `sedumi.M{q}` where $1 \leq q \leq p$.

Global optimality is ensured at some relaxation order p in the following cases:

- When LMI relaxed variables satisfy all the original problem constraints and reach the objective of the LMI relaxation. See examples 5.2.1 and 5.2.2 below.
- When rank $M_q^p =$ rank M_{q-r}^p for some $q = r+1, \ldots, p$. Here r denotes the smallest non-zero integer such that $2r$ is greater than or equal to the maximum degree occurring in the polynomial constraints. See examples 5.2.3 and 5.2.4.

Note that in general, the LMI relaxed vector is not necessarily feasible for the original optimization problem. However, the LMI relaxed vector is always feasible when minimizing a polynomial over linear constraints. In this particular case, evaluating the criterion at the LMI relaxed vector provides an upper bound on the global minimum, whereas the optimal criterion of the LMI relaxation is always a lower bound, see example 5.2.2.

When global optimality is ensured at some relaxation order p, then the moment matrix of first degree can be written as:

$$M_1^p = \sum_{i=1}^{\text{rank}\, M_1^p} \lambda_i \left[\begin{array}{c} 1 \\ \hat{x}^i \end{array} \right] \left[\begin{array}{c} 1 \\ \hat{x}^i \end{array} \right]^T$$

where $\sum_i \lambda_i = 1$ and each vector $\hat{x}^i$ denotes a global optimum.

Factorizing matrix M_1^p amounts to finding simultaneously scalars λ_i and optimal vectors $\hat{x}^i$. When the moment matrix has rank one ($\lambda_1 = 1$) the factorization follows easily. Otherwise, it is necessary to perform Gaussian elimination on the Cholesky factor of a moment matrix of higher order, as explained in [2]. The order depends on the number of global optima and the number of variables, see example 5.2.4.

When the moment matrix has rank greater than one, another way out can be to slightly perturb the criterion of the LMI problem so that the perturbed moment matrix has rank one. In order to do this, there is an additional field

`pars.pert` - perturbation vector of the criterion (default zero)

in the SeDuMi parameter structure `pars` introduced in section 4.7. Field `pars.pert` can either by a positive scalar (all entries in SeDuMi dual vector y are equally perturbed in the criterion), or a vector (entries are perturbed individually). See examples 5.2.4 and 5.2.5.

In order to test whether a given vector satisfies problem constraints (inequalities and equalities) and to evaluate the corresponding criterion, we developed a small Matlab script entitled TestPoly. The calling syntax is:

```
testpoly(P, x)
```

See section 2 to download the Matlab source file `testpoly.m`.

Warning messages are displayed by TestPoly when constraints are not satisfied by the input vector. Some numerical tolerance can be specified as an optional input argument. See the examples below.

5.2.1 First example

Consider problem [3, Pb. 4.9]. Shor's relaxation (order $p = 2$) yields a feasible relaxed vector reaching the LMI criterion, so it is the global optimum.

```
>> P = defipoly({'min -12*x1-7*x2+x2^2', '-2*x1^4+2-x2 == 0', ...
   '0 <= x1', 'x1 <= 2', '0 <= x2', 'x2 <= 3'}, 'x1,x2');
>> [crit, sedumi] = gloptipoly(P);
>> crit
crit =
  -16.7389
>> x = sedumi.y(1:2)'
x =
    0.7175    1.4698
>> testpoly(P, x)
TestPoly 1.00
Tolerance = 1e-06
Criterion = -16.7389
Constraints are satisfied at given tolerance
```

5.2.2 Second example

Consider problem [3, Pb 2.2]:

```
>> P = defipoly({['min 42*x1+44*x2+45*x3+47*x4+47.5*x5' ...
   '-50*(x1^2+x2^2+x3^2+x4^2+x5^2)'],...
   '20*x1+12*x2+11*x3+7*x4+4*x5<=40',...
   '0<=x1','x1<=1','0<=x2','x2<=1','0<=x3','x3<=1',...
   '0<=x4','x4<=1','0<=x5','x5<=1'},'x1,x2,x3,x4,x5');
```

The first LMI relaxation is unbounded, so we try the second one (order $p = 2$). The LMI criterion is equal to -17.9189 and the relaxed vector returned by GloptiPoly is feasible:

```
>> [crit, sedumi] = gloptipoly(P, 2);
>> crit
crit =
  -17.9189
>> testpoly(P, sedumi.y(1:5))
TestPoly 1.00
Tolerance = 1e-06
Criterion = 18.825
Constraints are satisfied at given tolerance
```

but leads to a suboptimal criterion ($18.825 > -17.9189$) so the global optimum has not been reached.

We try the third LMI relaxation (order $p = 3$). The relaxed vector returned by GloptiPoly is feasible:

```
>> [crit, sedumi] = gloptipoly(P, 3);
>> crit
crit =
  -17.0000
>> x = sedumi.y(1:5)'
x =
    1.0000    1.0000    0.0000    1.0000    0.0000
>> testpoly(P, x)
TestPoly 1.00
Tolerance = 1e-06
Criterion = -16.9997
Constraints are satisfied at given tolerance
```

and the LMI optimal criterion is almost reached, so the relaxed vector is a global solution.

5.2.3 Third example

Consider problem [3, Pb 4.10]:

```
>> P = defipoly({'min -x1-x2',...
   'x2-2-2*x1^4+8*x1^3-8*x1^2<=0',...
   'x2-4*x1^4+32*x1^3-88*x1^2+96*x1-36<=0',...
   'x1>=0','x1<=3','x2>=0','x2<=4'},'x1,x2');
```

with fourth degree constraints, hence $r = 2$.

Shor's relaxation (order $p = 2$) yields a criterion of -7.0000. Moment matrices M_1^2 and M_2^2 have ranks 1 and 4 respectively (with relative accuracy 10^{-6}):

```
>> [crit, sedumi] = gloptipoly(P, 2);
>> crit
crit =
   -7.0000
>> sv = svd(sedumi.M{1}); sum(sv > 1e-6*sv(1))
ans =
     1
>> sv = svd(sedumi.M{2}); sum(sv > 1e-6*sv(1))
ans =
     4
```

The LMI relaxation of order $p = 3$ yields a criterion of -6.6667. Moment matrices M_1^3, M_2^3 and M_3^3 have ranks 2, 2 and 6 respectively.

The LMI relaxation of order $p = 4$ yields a criterion of -5.5080. Moment matrices M_1^4, M_2^4, M_3^4 and M_4^4 have ranks 1, 1, 1 and 6 respectively. Since rank $M_3^4 =$ rank M_1^4 (recall that $r = 2$ here), the global optimum is reached:

```
>> [crit, sedumi] = gloptipoly(P, 4);
>> crit
crit =
   -5.5080
>> x = sedumi.y(1:2)'
x =
    2.3295    3.1785
>> testpoly(P, x)
TestPoly 1.00
Tolerance = 1e-06
Criterion = -5.508
Constraints are satisfied at given tolerance
```

5.2.4 Fourth example

Consider now quadratic problem [5, Ex. 5]:

```
>> P = defipoly({'min -(x1-1)^2-(x1-x2)^2-(x2-3)^2',...
   '1-(x1-1)^2 >= 0', '1-(x1-x2)^2 >= 0',...
   '1-(x2-3)^2 >= 0'}, 'x1,x2');
```

The first LMI relaxation yields a criterion of -3 and a moment matrix M_1^1 of rank 3. The relaxed vector returned by GloptiPoly satisfies the constraints but leads to a suboptimal criterion of -1.3895.

The second LMI relaxation yields a criterion of -2 and moment matrices M_1^2 and M_2^2 of ranks 3 and 3 respectively, showing that the global optimum has been reached (since

$r = 1$ here). However the feasible relaxed vector returned by GloptiPoly still leads to a suboptimal criterion of -1.3471. Since matrix M_1^2 has rank 3 there are three distinct global optima $\hat{x}^1$, $\hat{x}^2$ and $\hat{x}^3$ satisfying

$$M_1^2 = \lambda_1 \begin{bmatrix} 1 \\ \hat{x}^1 \end{bmatrix} \begin{bmatrix} 1 \\ \hat{x}^1 \end{bmatrix}^T + \lambda_2 \begin{bmatrix} 1 \\ \hat{x}^2 \end{bmatrix} \begin{bmatrix} 1 \\ \hat{x}^2 \end{bmatrix}^T + \lambda_3 \begin{bmatrix} 1 \\ \hat{x}^3 \end{bmatrix} \begin{bmatrix} 1 \\ \hat{x}^3 \end{bmatrix}^T$$

for $\lambda_1 + \lambda_2 + \lambda_3 = 1$.

To recover optimal solutions, we follow the approach described in [2] that consists in applying Gaussian elimination on the Cholesky factor of moment matrix M_2^2. From the reduced echelon form of the Cholesky factor we can deduce values of entries of vectors $\hat{x}^i$:

```
>> [crit, sedumi] = gloptipoly(P, 2);
>> crit
crit =
   -2.0000
>> sedumi.M{2}
ans =
    1.0000    1.5867    2.2477    2.7602    3.6690    5.2387
    1.5867    2.7602    3.6690    5.1072    6.5115    8.8245
    2.2477    3.6690    5.2387    6.5115    8.8245   12.7072
    2.7602    5.1072    6.5115    9.8012   12.1965   15.9959
    3.6690    6.5115    8.8245   12.1965   15.9959   22.1084
    5.2387    8.8245   12.7072   15.9959   22.1084   32.1036
>> sedumi.pows(1:5,:)
ans =
     1     0 % x1
     0     1 % x2
     2     0 % x1^2
     1     1 % x1*x2
     0     2 % x2^2
>> [U, S]=svd(sedumi.M{2});
>> diag(S)'
ans =
   64.7887    1.7467    0.3644    0.0000    0.0000    0.0000
>> U = U*sqrt(S); rref(U(:,1:3)')'
ans =
    1.0000         0         0 % lambda1 = 1
         0    1.0000         0 % lambda2 = x1
         0         0    1.0000 % lambda3 = x2
   -2.0000    3.0000    0.0000 % -2+3*x1 = x1^2, i.e. (x1-1)*(x1-2) = 0
   -4.0000    2.0000    2.0000 % -4+2*x1+2*x2 = x1*x2, i.e. (x1-2)*(x2-2) = 0
   -6.0000   -0.0000    5.0000 % -6+6*x2 = x2^2, i.e. (x2-2)*(x2-3) = 0
```

We obtain the three equalities

$$\begin{array}{rcl}(x_1-1)(x_1-2) & = & 0\\(x_1-2)(x_2-2) & = & 0\\(x_2-2)(x_2-3) & = & 0\end{array}$$

from which we deduce that the three global optima are

$$\hat{x}^1 = \left[\begin{array}{c}2\\3\end{array}\right], \quad \hat{x}^2 = \left[\begin{array}{c}2\\2\end{array}\right], \quad \hat{x}^3 = \left[\begin{array}{c}1\\2\end{array}\right].$$

An alternative way to recover optimal solution consists in breaking the structure of the problem so that there is only one global optimum. First we slightly perturb the second entry in the objective with the help of the `pars.pert` field in the SeDuMi parameter input structure. We obtain (almost) the same optimal criterion as previously but then we can check that the relaxed vector returned by GloptiPoly is now (almost) reaching the LMI optimum, so it is (almost) a global solution $\hat{x}^1$:

```
>> pars.pert = [0 1e-3];
>> [crit, sedumi] = gloptipoly(P, 2, pars);
>> crit
crit =
   -2.0030
>> x1 = sedumi.y(1:2)'
x1 =
    2.0000    3.0000
>> testpoly(P, x1)
TestPoly 1.00
Tolerance = 1e-06
Criterion = -2
Constraints are satisfied at given tolerance
```

With another perturbation (found by a trial-and-error approach), we retrieve another global solution $\hat{x}^2$:

```
>> pars.pert = [1e-3 -1e-3];
>> [crit, sedumi] = gloptipoly(P, 2, pars);
>> crit
crit =
   -2.0000
>> x2 = sedumi.y(1:2)'
x2 =
    2.0000    2.0000
>> testpoly(P, x2)
TestPoly 1.00
Tolerance = 1e-06
Criterion = -2
Constraints are satisfied at given tolerance
```

Finally, the third (and last) global solution $\hat{x}^3$ is retrieved as follows:

```
>> pars.pert = [-1e-3 -1e-3];
>> [crit, sedumi] = gloptipoly(P, 2, pars);
>> crit
crit =
   -2.0000
>> x3 = sedumi.y(1:2)'
x3 =
    1.0000    2.0000
>> testpoly(P, x3)
TestPoly 1.00
Tolerance = 1e-06
Criterion = -2
Constraints are satisfied at given tolerance
```

It is then easy to check that $\lambda_1 = 0.2477$, $\lambda_2 = 0.3390$ and $\lambda_3 = 0.4133$ in the above rank-one factorization of matrix M_1^2.

5.2.5 Fifth example

Consider the third LMI relaxation of the Max-Cut problem on the antiweb AW_9^2 graph introduced in section 4.9. We know that the global optimum has been reached, but due to problem symmetry, the LMI relaxed vector is almost zero and infeasible:

```
>> [crit, sedumi] = gloptipoly(P, 3);
>> norm(sedumi.y(1:9))
ans =
   1.4148e-10
```

In order to recover an optimal solution, we just perturb randomly each entry in the criterion:

```
>> pars.pert = 1e-3 * randn(1, 9);
>> [crit, sedumi] = gloptipoly(P, 3, pars);
>> x = round(sedumi.y(1:9)')
x =
    -1    -1     1     1    -1     1     1    -1     1
>> testpoly(P, x)
TestPoly 1.00
Tolerance = 1e-06
Criterion = 12
No constraints
```

6 Performance

6.1 Continuous optimization problems

We report in table 2 the performance of GloptiPoly on a series of benchmark non-convex continuous optimization examples. In all reported instances the global optimum was reached exactly by an LMI relaxation of small order, reported in the column entitled 'order' relative to the minimal order of Shor's relaxation, see section 4.6. CPU times are in seconds, all the computations were carried out with Matlab 6.1 and SeDuMi 1.05 with relative accuracy `pars.eps = 1e-9` on a Sun Blade 100 workstation with 640 Mb of RAM running under SunOS 5.8. 'LMI vars' is the dimension of SeDuMi dual vector y, whereas 'LMI size' is the dimension of SeDuMi primal vector x, see section 5.1. Quadratic problems 2.8, 2.9 and 2.11 in [3] involve more than 19 variables and could not be handled by the current version of GloptiPoly, see section 4.3. Except for problems 2.4 and 3.2, the computational load is moderate.

problem	variables	constraints	degree	LMI vars	LMI size	CPU	order
[5, Ex. 1]	2	0	4	14	36	0.41	0
[5, Ex. 2]	2	0	4	14	36	0.42	0
[5, Ex. 3]	2	0	6	152	2025	3.66	+5
[5, Ex. 5]	2	3	2	14	63	0.71	+1
[3, Pb. 2.2]	5	11	2	461	7987	31.8	+2
[3, Pb. 2.3]	6	13	2	209	1421	5.40	+1
[3, Pb. 2.4]	13	35	2	2379	17885	2810	+1
[3, Pb. 2.5]	6	15	2	209	1519	4.00	+1
[3, Pb. 2.6]	10	31	2	1000	8107	194	+1
[3, Pb. 2.7]	10	25	2	1000	7381	204	+1
[3, Pb. 2.10]	10	11	2	1000	5632	125	+1
[3, Pb. 3.2]	8	22	2	3002	71775	7062	+2
[3, Pb. 3.3]	5	16	2	125	1017	3.15	+1
[3, Pb. 3.4]	6	16	2	209	1568	4.32	+1
[3, Pb. 3.5]	3	8	2	164	4425	7.09	+3
[3, Pb. 4.2]	1	2	6	6	34	0.52	0
[3, Pb. 4.3]	1	2	50	50	1926	2.69	0
[3, Pb. 4.4]	1	2	5	6	34	0.72	0
[3, Pb. 4.5]	1	2	4	4	17	0.45	0
[3, Pb. 4.6]	2	2	6	27	172	1.16	0
[3, Pb. 4.7]	1	2	6	6	34	0.57	0
[3, Pb. 4.8]	1	2	4	4	17	0.44	0
[3, Pb. 4.9]	2	5	4	14	73	0.86	0
[3, Pb. 4.10]	2	6	4	44	697	1.45	+2

Table 2: Continuous optimization problems. CPU times and LMI relaxation orders required to reach global optima.

6.2 Discrete optimization problems

We report in table 3 the performance of GloptiPoly on a series of small-size combinatorial optimization problems. In all reported instances the global optimum was reached exactly by an LMI relaxation of small order, with a moderate computational load.

Note that the computational load can further be reduced with the help of SeDuMi's accuracy parameter. For all the examples described here and in the previous section, we set `pars.eps = 1e-9`. For illustration, in the case of the Max-Cut problem on the 12-node graph in [1] (last row in table 3), when setting `pars.eps = 1e-3` we obtain the global optimum with relative error 0.01% in 37.5 seconds of CPU time. In this case, it means a reduction by half of the computational load without significant impact on the criterion.

problem	vars	constr	deg	LMI vars	LMI size	CPU	order
QP [3, Pb. 13.2.1.1]	4	4	2	10	29	0.33	0
QP [3, Pb. 13.2.1.2]	10	0	2	385	3136	10.5	+1
Max-Cut P_1 [3, Pb. 11.3]	10	0	2	385	3136	7.34	+1
Max-Cut P_2 [3, Pb. 11.3]	10	0	2	385	3136	9.40	+1
Max-Cut P_3 [3, Pb. 11.3]	10	0	2	385	3136	8.25	+1
Max-Cut P_4 [3, Pb. 11.3]	10	0	2	385	3136	8.38	+1
Max-Cut P_5 [3, Pb. 11.3]	10	0	2	385	3136	12.1	+1
Max-Cut P_6 [3, Pb. 11.3]	10	0	2	385	3136	8.37	+1
Max-Cut P_7 [3, Pb. 11.3]	10	0	2	385	3136	10.0	+1
Max-Cut P_8 [3, Pb. 11.3]	10	0	2	385	3136	9.16	+1
Max-Cut P_9 [3, Pb. 11.3]	10	0	2	385	3136	11.3	+1
Max-Cut cycle C_5 [1]	5	0	2	30	256	0.35	+1
Max-Cut complete K_5 [1]	5	0	2	31	676	0.75	+2
Max-Cut 5-node [1]	5	0	2	30	256	0.47	+1
Max-Cut antiweb AW_9^2 [1]	9	0	2	465	16900	63.3	+2
Max-Cut 10-node Petersen [1]	10	0	2	385	3136	7.21	+1
Max-Cut 12-node [1]	12	0	2	793	6241	73.2	+1

Table 3: Discrete optimization problems. CPU times and LMI relaxation orders required to reach global optima.

7 Further developments

Even though GloptiPoly is basically meant for small- and medium-size problems, the current limitation on the number of variables (see section 4.3) is somehow restrictive. For example, the current version of GloptiPoly is not able to handle quadratic problems with more than 19 variables, whereas it is known that SeDuMi running on a standard workstation can solve Shor's relaxation of quadratic Max-Cut problems with several hundreds of variables. The limitation of GloptiPoly on the number of variables should be removed in the near future.

A more systematic procedure is also required to detect global optimality and to retrieve one (or several, or all) optimal solution(s). At the present time we are still lacking of a necessary and sufficient condition (for example on ranks of moment matrices) to ensure global optimality. The rank-one factorization of the moment matrix must also be implemented to extract automatically optimal solutions. We can follow the preliminary work in this direction reported in [2].

Scaling may also improve the numerical behavior of the LMI solver, since it is well-known that problems involving polynomial bases with monomials of increasing powers are naturally badly conditioned. If lower and upper bounds on the optimization variables are available as problem data, it may be a good idea to scale all the intervals around one. Alternative bases such as Chebyshev polynomials may also prove useful.

Acknowledgments

Many thanks to Claude-Pierre Jeannerod (INRIA Rhône-Alpes), Dimitri Peaucelle (LAAS-CNRS Toulouse) and Jean-Baptiste Hiriart-Urruty (Université Paul Sabatier Toulouse).

References

[1] M. Anjos. New Convex Relaxations for the Maximum Cut and VLSI Layout Problems. *PhD Thesis*, Waterloo University, Ontario, Canada, 2001. See `orion.math.uwaterloo.ca/∼hwolkowi`

[2] G. Chesi, A. Garulli. On the Characterization of the Solution Set of Polynomial Systems via LMI Techniques. *Proceedings of the European Control Conference*, pp. 2058–2063, Porto, Portugal, 2001.

[3] C. A. Floudas, P. M. Pardalos, C. S. Adjiman, W. R. Esposito, Z. H. Gümüs, S. T. Harding, J. L. Klepeis, C. A. Meyer, C. A. Schweiger. Handbook of Test Problems in Local and Global Optimization. *Kluwer Academic Publishers*, Dordrecht, 1999. See `titan.princeton.edu/TestProblems`

[4] D. Henrion, S. Tarbouriech, D. Arzelier. LMI Approximations for the Radius of the Intersection of Ellipsoids. *Journal of Optimization Theory and Applications*, Vol. 108, No. 1, pp. 1–28, 2001.

[5] J. B. Lasserre. Global Optimization with Polynomials and the Problem of Moments. *SIAM Journal on Optimization*, Vol. 11, No. 3, pp. 796–817, 2001.

[6] J. B. Lasserre. An Explicit Equivalent Positive Semidefinite Program for 0-1 Nonlinear Programs. To appear in *SIAM Journal on Optimization*, 2002.

[7] Waterloo Maple Software Inc. Maple V release 5. 2001. See `www.maplesoft.com`

[8] The MathWorks Inc. Matlab version 6.1. 2001. See `www.mathworks.com`

[9] Y. Nesterov. Squared functional systems and optimization problems. Chapter 17, pp. 405–440 in H. Frenk, K. Roos, T. Terlaky (Editors). High performance optimization. *Kluwer Academic Publishers*, Dordrecht, 2000.

[10] P. A. Parrilo. Structured Semidefinite Programs and Semialgebraic Geometry Methods in Robustness and Optimization. *PhD Thesis*, California Institute of Technology, Pasadena, California, 2000. See `www.control.ethz.ch/∼parrilo`

[11] J. F. Sturm. Using SeDuMi 1.02, a Matlab Toolbox for Optimization over Symmetric Cones. *Optimization Methods and Software*, Vol. 11-12, pp. 625–653, 1999. Version 1.05 available at `fewcal.kub.nl/sturm/software/sedumi.html`