SPOTless
Polynomial and Conic Optimization

Mark M. Tobenkin, Frank Permenter, Alexandre Megretski

April 17, 2013

Contents

1 Introduction

SPOTless is a software toolbox for MATLAB for posting conic [**?**] and sum-of-squares (SOS) optimization problems [**?**]. The basic functionality provided:

1. A simple, reasonably fast approximate symbolic algebra package.
2. A modeling tool for posing and pre-/post-processing conic optimization problems.
3. A modeling tool for posing SOS problems.

Several alternative tools exist for addressing these problems, notably CVX [**?**] and Yalmip [**?**]. We recommend these packages as more user-friendly modeling environments – the emphasis of SPOTless is on providing an *extensible* tool for *building other libraries*.

2 Quick Start

This section provides an explanation of the basic capabilities of SPOTless, and provides simple examples of solving conic and SOS programs and inspecting the solutions.

2.1 Basic SPOTless Work-Flow

The basic work-flow in SPOTless is as described by Figure 1.

Let's walk through the following example to provide some more detail. The input to the program is a pair of positive integers (n, m) a matrix $A \in \mathbb{R}^{n \times m}$ and a vector $b \in \mathbb{R}^n$. The program is defined by:

$$\begin{aligned} \underset{x \in \mathbb{R}^m}{\text{minimize}} \quad & \sum_{i=1}^{m} |x_i| \\ \text{subj. to} \quad & Ax = b. \end{aligned}$$

```
1  prog = spotprog;
2  [prog,x] = prog.newFree(m);
3  prog = prog.withEqs(A*x - b);
4
5  [prog,a] = prog.newPos(m);
6  prog = prog.withPos(a - x);
7  prog = prog.withPos(a - (-x));
8
9  obj = sum(a);
10
11 sol = prog.minimize(obj);
12
13 if sol.solutionQuality < 0
14     error('Solution is of very low quality.');
```

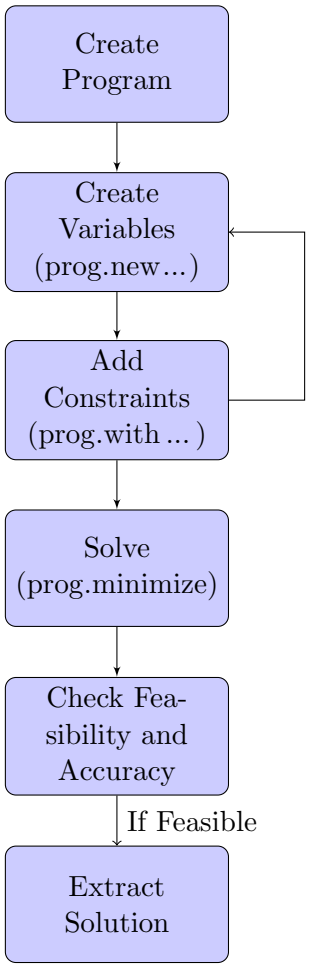

Figure 1: Basic SPOTless workflow.

```
15 elseif sol.solutionQuality < 1
16     warning('Low solution quality.');
17 end
18 if sol.primalInfeasible || sol.dualInfeasible
19     error('Infeasibility detected.');
20 end
21
22 xopt = double(sol.eval(x));
```

2.1.1 Create a program

First, one constructs a program object as in

```
1   prog = spotprog;
```

for conic programming. For SOS programming one instead uses

```
    prog = spotsosprog;
```

2.1.2 Create Variables

Next, one constructs new decision variables, e.g.

```
2 [prog,x] = prog.newFree(m);
```

updates prog to have m new free (unconstrained) variables, stored in the vector x. A similar syntax modifies the program to have m non-negative variables: as in

```
5 [prog,a] = prog.newPos(m);
```

stored in the vector a. The types supported by spotprog are listed in Table 1.

2.1.3 Add Constraints

Equality constraints are defined by expressions which must be equal to zero, i.e.

```
3 prog = prog.withEqs(A*x - b);
```

ensures $Ax = b$. Additional "conic" constraints can be added, e.g.

```
6 prog = prog.withPos(a - x);
```

requires a(i) $\geq$ x(i) for $i \in \{1, \ldots, m\}$. Similar function calls, (i.e. prog.withType()) are used to impose other kinds of constraints.

2.1.4 Solve The Program

The next step is to identify an objective, and solve the program:

```
11 sol = prog.minimize(obj);
```

here obj is an affine expression in decision parameters and sol is a solution object. Calling prog.minimize without a first argument, or with zero as the first argument solves a feasibility problem. See Section **??** for more details about specifying a solver and adding pre-/post-processing options.

2.1.5 Examine Feasibility and Accuracy

Ideally, a "solution" to a conic programming problem should consist of one of:

(i) A pair of primal and dual feasible points which are optimal.

(ii) A primal improving direction (proving dual infeasibility).

(iii) A dual improving direction (proving primal infeasibility).

Solution *quality* in SPOTLESS refers to a numerical measure of confidence that the solver has obtained such a solution. Both poorly posed problems and numerical errors can lead to a low quality solution. Section **??** contains some examples of programs for which no solution in the above sense exists.

Once a solver has been called, one must examine the quality of the returned solution. Note that **the interpretation of solution quality is application dependent!** Please see Section **??** for information about customizing solution quality reporting for your application. Negative solution quality is supposed to indicate a useless solution, as tested by

```
13 if sol.solutionQuality < 0
```

whereas solution quality less than 1 is supposed to be circumspect.

If the soluton quality is high, one can still have an *infeasible* program, as test by

```
18 if sol.primalInfeasible || sol.dualInfeasible
```

Primal infeasibility generally implies that no choice of decision variables satisfies the given constraints. Dual infeasibility often means that the optimal cost is unbounded below. A more thorough account is given in Section **??**.

2.1.6 Extract Solution

To extract a solution, one first uses sol.**eval** to substitute optimal decision parameters into any algebraic expression, e.g.

```
18 xopt = double(sol.eval(x));
```

Here, double is then used to transform the resulting expression from SPOTLESS's internal symbolic algebra library into a numerical vector.

Table 1: Constraint Types for spotprog

Type	Abbrev.	Vector Space	Constraint
Free	Free	$x \in \mathbb{R}^n$	
Positive	Pos	$x \in \mathbb{R}^n$	$x_i \geq 0$
Lorentz	Lor	$x \in \mathbb{R}^{n \times m}$	$x_{1j}^2 \geq \sum_{i=2}^{n} \lvert x_{ij} \rvert^2, \quad \forall j$
Rotated Lorentz	RLor	$x \in \mathbb{R}^{n \times m}$	$x_{1j} \geq 0, \quad x_{1j}x_{2j} \geq \sum_{i=3}^{n} \lvert x_{ij} \rvert^2 \quad \forall j$
Positive Semidefinite	PSD	See below.[1]	

[1] The positive semidefinite cone refers to matrices $X \in \mathbb{R}^{n \times n}$ such that $v^T X v \geq 0$ for all $v \in \mathbb{R}^n$. Two representations are used in SPOTless: a single PSD variable can be represented by an $n \times n$ msspoly and a set of N PSD variables can be represented as $\binom{n+1}{2} \times N$ msspoly. When $N = 1$ this transformation from one representation to the other is given by mss_v2s (read: vector-to-symmetric) and mss_s2v (read: symmetric-to-vector):

$$\text{mss\_s2v(X)} = \begin{bmatrix} X_{11} \\ X_{12} \\ X_{22} \\ X_{13} \\ \vdots \end{bmatrix} \qquad \text{mss\_v2s(x)} = \begin{bmatrix} x_1 & x_2 & x_4 & \dots \\ x_2 & x_3 & x_5 & \dots \\ x_5 & x_3 & x_6 & \dots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix},$$

where $X \in \mathbb{R}^{n \times n}$ and $x \in \mathbb{R}^{\binom{n+1}{2}}$. **N.B.**: Generally:

$$\text{tr(X*S)} \not\equiv \text{mss\_s2v}(X)' * \text{mss\_s2v}(S).$$

2.2 Basic Types

2.3 Conic Programming Examples

2.3.1 ℓ_∞ Fitting Example

Given an positive integers N and d find $x \in \mathbb{R}^n$ to solve:

$$\begin{array}{ll} \text{minimize} & \max\limits_{i\in\{-N,\ldots,N\}} \left\{ \left| \sum\limits_{i=0}^{d} x_{i+1} \left(\frac{i}{N}\right)^d - \left|\frac{i}{N}\right| \right| \right\}, \\ \text{subj. to} & x \in \mathbb{R}^{d+1}. \end{array}$$

examples/linf_fitting.m

```
1  % Problem Data:  absolute value evaluated at 101 points.
2  d = 5;
3  N = 50;
4  tt = linspace(-1,1,2*N+1);
5  yy = abs(tt);
6
7
8  % Fit a degree d polynomial.
9  prog = spotprog;
10
11 t = msspoly('t');
12 basis = monomials(t,0:d);
13 [prog,coeff] = prog.newFree(length(basis));
14 f = coeff'*basis;
15
16 err = yy - msubs(f,t,tt);
17
18 [prog,obj] = prog.newFree(1);
19 prog = prog.withPos(obj - err');
20 prog = prog.withPos(obj + err');
21
22 sol = prog.minimize(obj);
23
24 fopt = sol.eval(f);
```

This program fits a degree 5 polynomial in t to the function $|t|$ over $[-1, 1]$. Line 9 constructs a new program. Lines 11 and 12 introduce the indeterminate t and constructs the vector basis = [1 ; t ; ... ; t^d]. A vector of $d + 1$ coefficients are created on Line 13, and on Line 14 used to construct a function f = coeff(1) + coeff(2)t + ... The function msubs is used on line 16 to evaluate f at each t = tt(i) (returning a row). Finally Lines 19 and 20 require the free variable obj to satisfy obj $\geq$ err(i) and $-$err(i)

respectively for each i. The program is solved (Line 22), and the optimal coefficients are substituted into f and stored in fopt on Line 24.

2.3.2 SDP Projection Example

Given $A = A' \in \mathbb{R}^{n\times n}$, solve:

$$\begin{array}{rl} \text{minimize} & \|A - X\|_F \\ \text{subj. to} & X \in \mathbb{S}_{n,+}. \end{array}$$

examples/sdp_projection.m

```
1  % Problem Data:  A random symmetric matrix.
2  n = 10;
3  A = randn(n,n);
4  A = A + A';
5
6  prog = spotprog;
7  [prog,P] = prog.newPSD(n);
8  [prog,obj] = prog.newFree(1);
9
10 prog = prog.withLor([ obj ; P(:)-A(:)]);
11
12 sol = prog.minimize(obj);
13
14 Popt = double(sol.eval(P));
```

The optimization problem begins at line 6 by constructing a new program. Line 7 constructs a new $n \times n$ PSD decision variable,P, and line 8 a 1-by-1 free variable, obj. Line 10 constraints obj to be greater than the 2-norm of $P - A$ when regarded of as a vector using a Lorentz cone constraint (see Section **??**). A solution is then generated to the problem of minimizing obj with the default solver on Line 12. Finally, Line 14 substitutes the optimizing variables into the expression for P, and then transforms that expression in a MATLAB double array stored in Popt.

2.3.3 SDP Control Design Example

Given a matrix pair $(A, B) \in \mathbb{R}^{n\times n} \times \mathbb{R}^{n\times m}$ representing a stabilizable discrete time LTI system

$$x[t+1] = Ax[t] + Bu[t],$$

find matrix $K \in \mathbb{R}^{m\times n}$ such that $A + BK$ is exponentially stable. For $\rho \in (0, 1)$ solve

$$\begin{array}{rl} \text{find} & (S, L) \in \mathbb{S}_{n,+} \times \mathbb{R}^{m\times n} \\ \text{subj. to} & \begin{bmatrix} (1-\rho)S & AS + BL \\ SA' + L'B' & S \end{bmatrix} \in \mathbb{S}_{2n,+}. \end{array}$$

and take $K = LS^{-1}$.

examples/sdp_lti_control.m

```
% Problem data matrices.
A = [2 1 0 ; 0 2 1 ; 0 0 2];
B = [ 0 0 1]';
n = size(B,1);
m = size(B,2);

prog = spotprog;
[prog,S] = prog.newPSD(n);
[prog,L] = prog.newFree(n*m);
L = reshape(L,m,n);

rho = 0.1;
prog = prog.withPSD([ (1-rho)*S (A*S+B*L) ; (A*S+B*L)' S]);

sol = prog.minimize(0);

S = double(sol.eval(S));
L = double(sol.eval(L));
K = L/S;
```

2.4 SOS Programming Examples

2.4.1 Upper Bound Polynomial On Sphere

```
% Construct a random polynomial.
n = 2;
d = 4;
x = msspoly('x',n);
basis = monomials(x,0:d);
p = randn(length(basis))'*basis;

g = 1 - x'*x;

prog = spotsosprog;
prog = prog.withIndeterminate(x);
[prog,r] = prog.newFree(1);

[prog,f] = prog.newFreePoly(x,monomials(x,0:d-2));
prog = prog.withSOS(r-p-f*g);

sol = prog.minimize(r);
```

```
18
19 sol.eval(r);
```